SCL-Codegenerierung mit BORIS

Kurzeinführung

Inhalt

0 Zweck dieser Kurzeinführung

Diese Kurzeinführung dient dem schnellen Einstieg in die Arbeit mit dem SCL-AutoCode-Generator zum blockorientierten Simulationssystem BORIS, insbesondere zur Erzeugung von Testcode für einfache Strukturen im Zusammenhang mit der Demo-Version des Codegenerators. Grundkenntnisse in der Nutzung von BORIS selbst werden dabei vorausgesetzt.

1 Installation der Demo-Version

Die Nutzung der Demo-Version des SCL-Codegenerators setzt eine bestehende WinFACT 7-Installation voraus; ggf. genügt eine Demo-Version von WinFACT 7. Die Installation muss dabei unbedingt **vor** der Installation des SCL-Codegenerators erfolgen.

> Eine Demo-Version von WinFACT 7 finden Sie bei Bedarf im Internet unter http://www.kahlert.com/web/download.php .

Die Installation der Demo-Version des SCL-Codegenerators erfolgt anschließend durch Aufruf des Installationsprogramms *Setup.exe*. Wesentlich für eine korrekte Funktion des Codegenerators ist dabei, dass als Installationsziel für den Codegenerator das WinFACT 7-Programmverzeichnis (standardmäßig *c:\Programme\Kahlert\WinFACT 7*) angegeben wird (Abbildung 1).

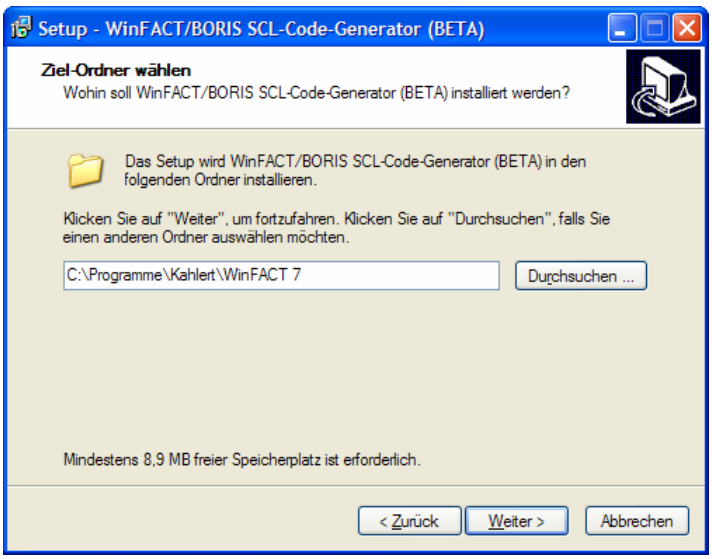

Abbildung 1 Angabe des Zielverzeichnisses für den SCL-Codegenerator

Nach der Installation des Codegenerators finden Sie unterhalb des WinFACT 7-Programmverzeichnisses drei neue Verzeichnisse, die für die Arbeit mit dem SCL-Codegenerator benötigt werden (Abbildung 2):

Unterverzeichnis	**Bedeutung**
.Code-Generator-Install-Backup	Enthält die zur korrekten Deinstallation des SCL-Codegenerators benötigten Dateien
AutoCode\SCL	Enthält die SCL-Quelldateien und Verweisdateien
Examples\SCL-Code-Generator	Beispieldateien zur Codegenerierung

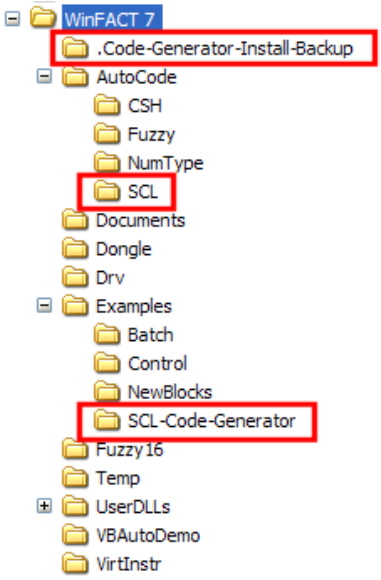

Abbildung 2 Vom SCL-Codegenerator eingerichtete Unterverzeichnisse

2 Leistungsumfang des SCL-Codegenerators

Der SCL-Codegenerator zum blockorientierten Simulationssystem BORIS ermöglicht die Generierung von Quellcode für Speicherprogrammierbare Steuerungen (SPS) gemäß der Spezifikation der *Structured Control Language* SCL nach DIN 61131-3[1]. Dieser Quellcode kann dann mit Hilfe der entsprechenden SPS-Programmierumgebung (z. B. STEP 7) zur direkten Programmierung der SPS benutzt werden.

> Beachten Sie bitte, dass in der Demo-Version des SCL-Codegenerators maximal 20 Blöcke zu SCL-Code generiert werden können und zudem eine Generierung nur für bestimmte Blocktypen möglich ist. Außerdem ist die Laufzeit der Demo-Version auf 30 Tage ab Installationszeitpunkt begrenzt.

Der SCL-Codegenerator stellt somit eine Alternative zur Kommunikation zwischen BORIS und der SPS über den entsprechenden SPS-Treiber dar. Während im letzteren Fall das eigentliche Steuerprogramm unter BORIS läuft, arbeitet die SPS mit Hilfe des vom SCL-Codegenerator erzeugten Codes autark. BORIS wird damit zur eigentlichen Steuerung nicht mehr benötigt, kann aber natürlich weiterhin zur Datenerfassung oder – im Zusammenwirken mit dem *Flexible Animation Builder* FAB – zur Prozessvisualisierung herangezogen werden. Abbildung 3 verdeutlicht die einzelnen Optionen.

Die folgenden Abschnitte geben anhand einfacher Beispiele einen kurzen Einblick in die Arbeit mit dem SCL-Codegenerator. Die Portierung des erzeugten SCL-Codes auf die SPS ist dabei herstellerabhängig und wird daher im Rahmen dieser Kurzeinführung nur am Rande betrachtet; Hinweise dazu können Sie der Dokumentation der entsprechenden Programmierumgebung (z. B. SIEMENS STEP 7) entnehmen.

[1] Im deutschsprachigen Raum wird SCL-Code als *Strukturierter Text* ST bezeichnet.

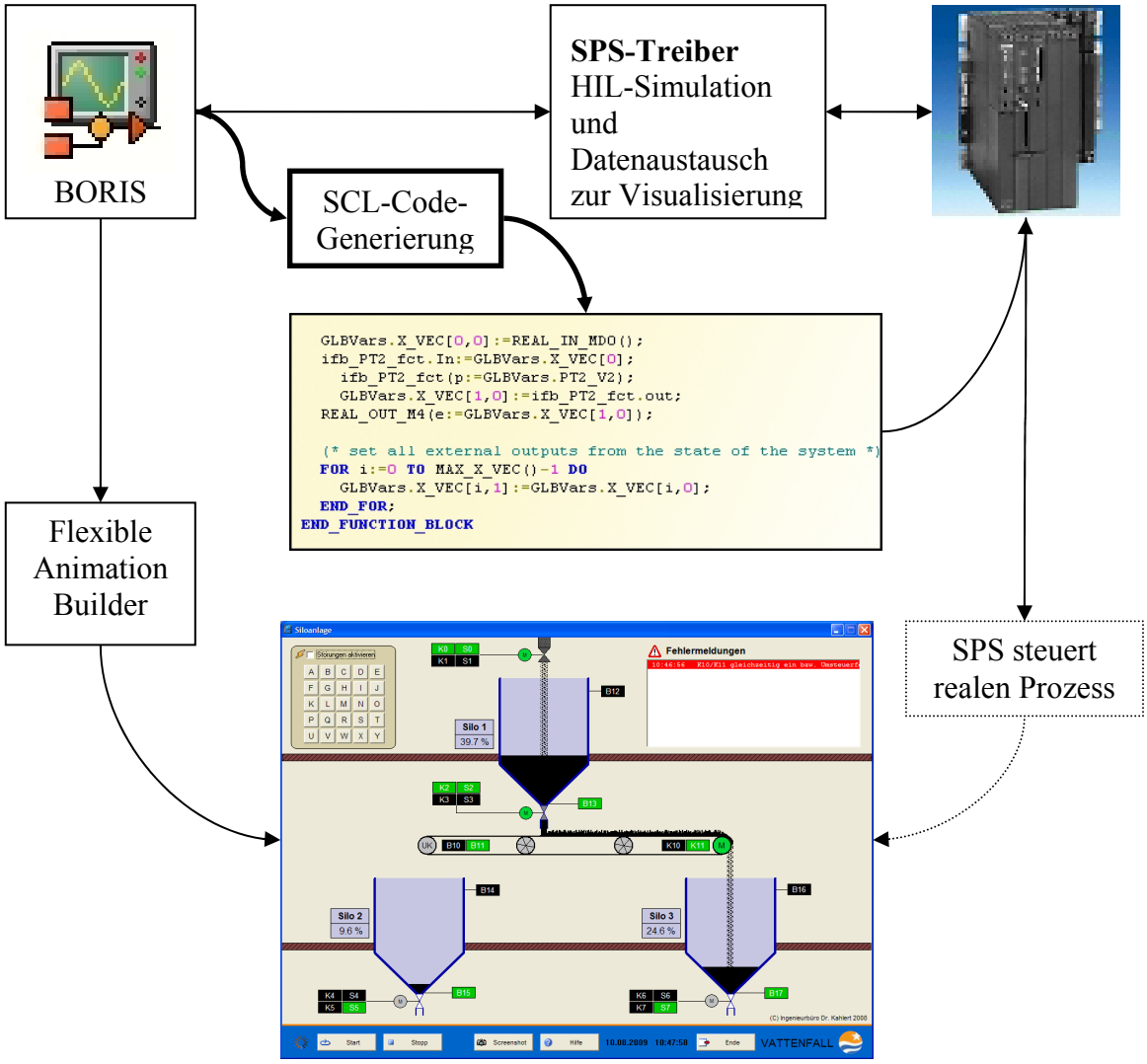

Abbildung 3 HIL-Simulation, Code-Generierung und Visualisierung mit BORIS

3 Arbeiten mit dem SCL-Codegenerator

3.1 Einführendes Beispiel

Um die grundlegenden Arbeitsschritte bei der SCL-Codegenerierung kennen zu lernen, soll zunächst lediglich SCL-Code für einen einzelnen BORIS-Systemblock generiert werden. Starten Sie dazu – falls nicht bereits geschehen – BORIS und fügen Sie ein *Logikgatter mit zwei Eingängen* ein. Anschließend öffnen Sie das Popup-Menu des Blockes (rechte Maustaste auf dem Block betätigen) und wählen dort die Menüoption IN CODE-GENERIERUNGSLISTE EINTRAGEN. Der Blocktitel sollte daraufhin wie in Abbildung 4 dargestellt mit gelbem Hintergrund erscheinen.

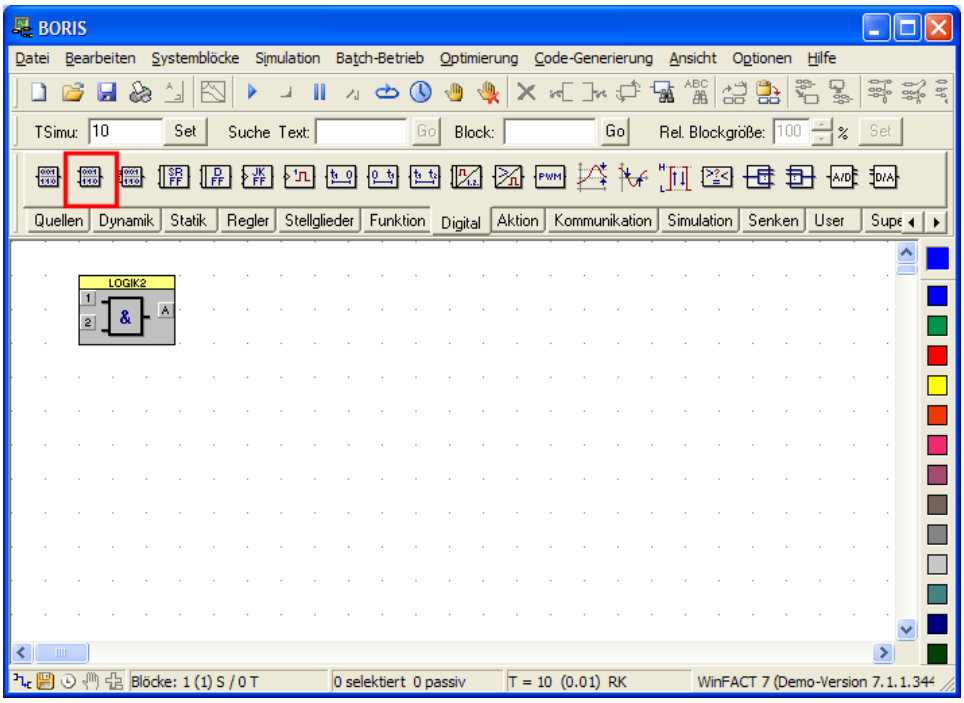

Abbildung 4 BORIS mit Logikgatter, das zu SCL-Code generiert werden soll

Nunmehr müssen die Einstellungen für die Codegenerierung vorgenommen werden. Dazu dient die Menüoption CODE-GENERIERUNG | CODE-GENERIERUNGS-EINSTELLUNGEN... . Nehmen Sie im daraufhin erscheinenden Dialog auf dem Registerblatt *Dateieinstellungen* die folgenden Einstellungen vor (Abbildung 5) [2]:

Verweisdatei	WF7_AC_S7_SCL_wo_was.csv
SCL-Datei	c:\temp\Logik2.scl
Symboltabellendatei-Datei	c:\temp\Logik2.sdf

Auf dem Registerblatt *Rahmenfunktionen* lassen Sie die Voreinstellung *Keinen Aufruf erzeugen* ausgewählt (Abbildung 6).

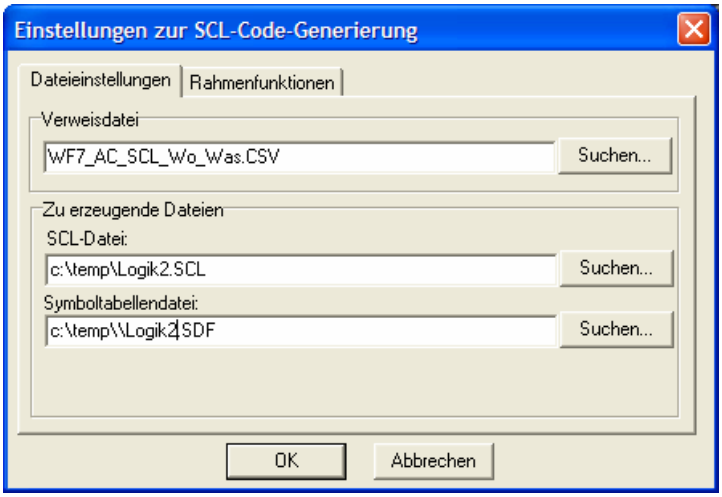

Abbildung 5 Dateieinstellungen zur Code-Generierung

[2] Anstelle des Verzeichnisses *c:\temp* können Sie als Zielverzeichnis für die SCL- bzw. Symboltabellendatei auch jedes beliebige andere (existierende) Verzeichnis angeben.

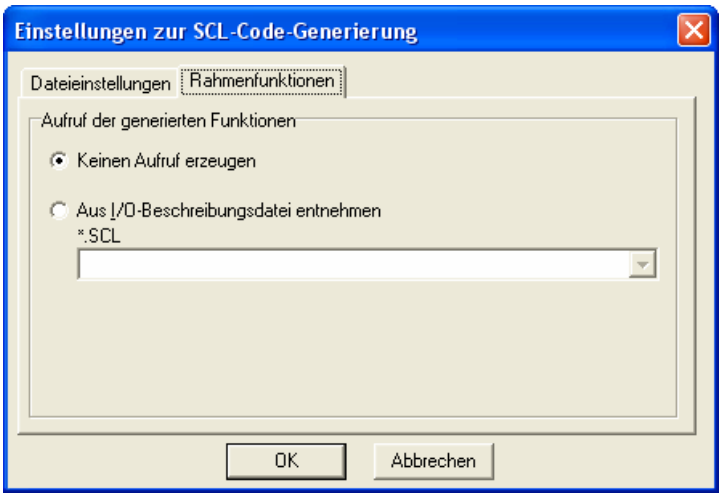

Abbildung 6 Registerkarte zur Auswahl einer Rahmenfunktion

Damit sind alle für dieses einfache Beispiel erforderlichen Einstellungen getätigt und es ist möglich Code zu generieren. Wählen Sie hierzu den Menüpunkt CODE-GENERIERUNG | CODE GENERIEREN… . Ein Dialogfenster zeigt Ihnen während der Codegenerierung den Verlauf, den Status und weitere Informationen bezüglich der Code-Generierung an (Abbildung 7).

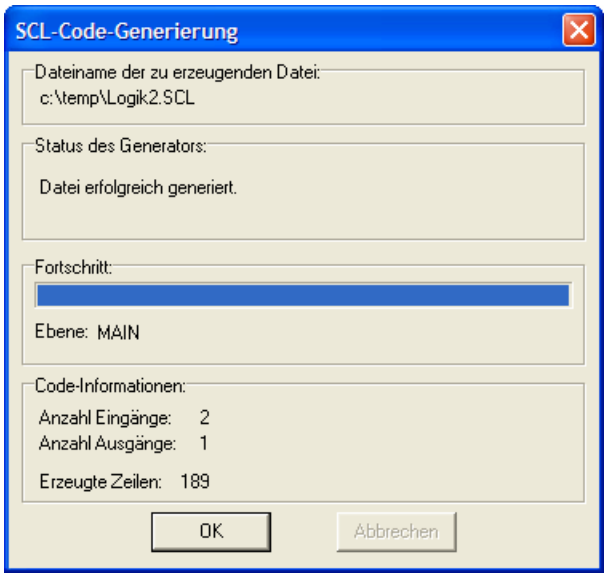

Abbildung 7 Fortschritt der Code-Generierung

Während der Codegenerierung werden die beiden zuvor spezifizierten Dateien erzeugt. Der eigentliche SPS-Programmcode befindet sich in der SCL-Datei. Diese enthält alle notwendigen Funktionen, Funktionsbausteine, Datenbausteine und ggf. auch Organisationsbausteine des SPS-Programms. Da hier keine Rahmenfunktion verwendet wurde, ist auch kein Organisationsbaustein in der SCL-Datei enthalten. Dieser Fall ist also nur dann sinnvoll, wenn ein eigenes SCL-Programm um die Funktion einer BORIS-Struktur ergänzt werden soll.

> Wundern Sie sich nicht, wenn der Logikgatter-Block in BORIS nach erfolgter Codegenerierung einen grün-gelben Titelbalken aufweist. Auf diese Weise kennzeichnet BORIS Blöcke der Code-Generierungsliste, die offene Ein- und/oder Ausgänge besitzen.

Die generierten Funktionsblöcke erhalten einen Namen, der aus dem Dateinamen (hier *Logik2*) besteht, dem zusätzlich noch ein _INIT_CONTROL oder ein _CONTROL angefügt wurde. Der Funktionsbaustein mit der Endung _INIT_CONTROL dient zur Initialisierung und ist nur einmal aufzurufen. Erst dann darf _CONTROL beliebig oft – also für alle folgenden Zyklen – ausgeführt werden. Die Bausteine unterscheiden sich bezüglich des Aufrufs nicht voneinander. Listing 1 zeigt den Kopf des CONTROL-Bausteins für dieses Beispiel. Die Ein- und Ausgangsparameter der Funktionsbausteine entsprechen den Ein- und Ausgängen des zu Code generierten BORIS-Systemblocks, hier also des UND-Gatters.

```
FUNCTION_BLOCK Logik2_CONTROL
  VAR_INPUT
    LOGIK2_0I1 : REAL;
    LOGIK2_0I2 : REAL;
  END_VAR

  VAR_OUTPUT
    LOGIK2_0O1 : REAL;
  END_VAR

  VAR
    i : INT;
  END_VAR
 .
 .
 .
END_FUNKTION_BLOCK
```

Listing 1 Kopf des CONTROL-Funktionsbausteins für Beispiel

Der Aufruf aus einem eigenen SCL-Programm (z. B. Funktionsbaustein) ist nun sehr einfach; Listing 2 zeigt dazu ein Beispiel.

```
FUNCTION_BLOCK Logik_CTRL
  VAR
    INITIALISIERUNGS_ANLAUF : BOOL:=TRUE;
    ictrl : Logik2_INIT_CONTROL;
    ctrl  : Logik2_CONTROL;
  END_VAR
 .
 .
 .
  IF INITIALISIERUNGS_ANLAUF THEN
    ictrl(LOGIK2_0I1:=WF_BOOL_TO_REAL(%E0.0),
          LOGIK2_0I2:=WF_BOOL_TO_REAL(%E0.1));
    %A0.0 := WF_REAL_TO_BOOL(ictrl.LOGIK2_0O1);
  ELSE
    ctrl(LOGIK2_0I1:=WF_BOOL_TO_REAL(%E0.0),
         LOGIK2_0I2:=WF_BOOL_TO_REAL(%E0.1));
    %A0.0 := WF_REAL_TO_BOOL(ictrl.LOGIK2_0O1);
  END_IF;
 .
 .
 .
END_FUNKTION_BLOCK
```

Listing 2 Aufruf des erzeugten SCL-Codes aus eigenem Programm

Bitte beachten Sie, dass hier eine BORIS-Struktur zu Code generiert wurde wurde, die nicht zeitäquidistant arbeiten muss, da sie keinerlei dynamische Glieder wie Integrierer, PID-Regler

o. ä. enthält. Daher muss innerhalb des übergeordneten SPS-Programms auch keine Sorge dafür getragen werden, dass der generierte Funktionsbaustein zu äquidistanten Zeitpunkten aufgerufen wird.

Die ebenfalls erzeugte SDF-Datei enthält eine Symboltabelle im CSV-Format.[3] Diese kann bei Bedarf in den Symboltabelleneditor der SPS-Entwicklungsumgebung importiert werden. Listing 3 zeigt die für das vorgestellte Beispiel generierte Datei.

```
"MAX_DYNAMIC_ORDER", "FC 1","FC 1","   "
"MAX_DGLSYS_ORDER", "FC 2","FC 2","   "
"MAX_INPUT", "FC 3","FC 3","   "
"MAX_X_VEC", "FC 4","FC 4","   "
"MAX_I_VEC", "FC 5","FC 5","   "
"SIGNALMAX", "FC 6","FC 6","   "
"SIGNALMIN", "FC 7","FC 7","   "
"HIGHLEVEL", "FC 8","FC 8","   "
"LOWLEVEL", "FC 9","FC 9","   "
"THRESHOLD", "FC 10","FC 10","   "
"PI_DIV_2", "FC 11","FC 11","   "
"PI_MUL_2", "FC 12","FC 12","   "
"PI", "FC 13","FC 13","   "
"EXP1", "FC 14","FC 14","   "
"WF_REAL_TO_BOOL", "FC 15","FC 15","   "
"WF_BOOL_TO_REAL", "FC 16","FC 16","   "
"WF_TIME_TO_REAL", "FC 17","FC 17","   "
"WF_REAL_TO_TIME", "FC 18","FC 18","   "
"LOGIC2Struct", "UDT 1","UDT 1","   "
"Logic2_fct", "FC 19","FC 19","   "
"Logik2_INIT_CONTROL", "FB 1","FB 1","   "
"Logik2_CONTROL", "FB 2","FB 2","   "
"GLBVars", "DB 1","DB 1","   "
```

Listing 3 Erzeugte Symboltabellendatei

3.2 Verwendung von Ein-/Ausgängen der SPS

3.2.1 I/O-Beschreibungsdatei und I/O-Code-Abschnitt

Weist die zu SCL-Code generierte BORIS-Struktur wie im vorangegangenen Beispiel offene Ein- und/oder Ausgänge auf, so erhält der generierte Funktionsbaustein entsprechende Ein- bzw. Ausgangsparameter. Sollen Ein- und Ausgänge von BORIS-Blöcken jedoch direkt mit Ein- und/oder Ausgängen der SPS verbunden werden, so kann dies ohne manuelle „Nachprogrammierung“ erreicht werden. Dazu verfügt BORIS über einen speziellen Block, der bei der Code-Generierung eine vordefinierte Code-Sequenz schreibt. Diese Code-Sequenz wird aus einer zusätzlichen Datei genommen, die für diesen Zweck vorgesehen ist – einer so genannten *I/O-Beschreibungsdatei*. Die Datei ist unterteilt in Abschnitte wobei jeder Abschnitt eine zusammengehörige Code-Sequenz enthält. Diese wiederum muss eine Funktion enthalten, die dann als Aufruf in das generierte Programm übernommen wird. Der zugrunde liegende BORIS-Blocktyp wird als *I/O-Code-Abschnitt* bezeichnet und ist am rechten Rand der Registerkarte *Sonstiges* der Systemblock-Toolbar von BORIS zu finden (Abbildung 8).

[3] CSV steht für *Comma Separated Values*

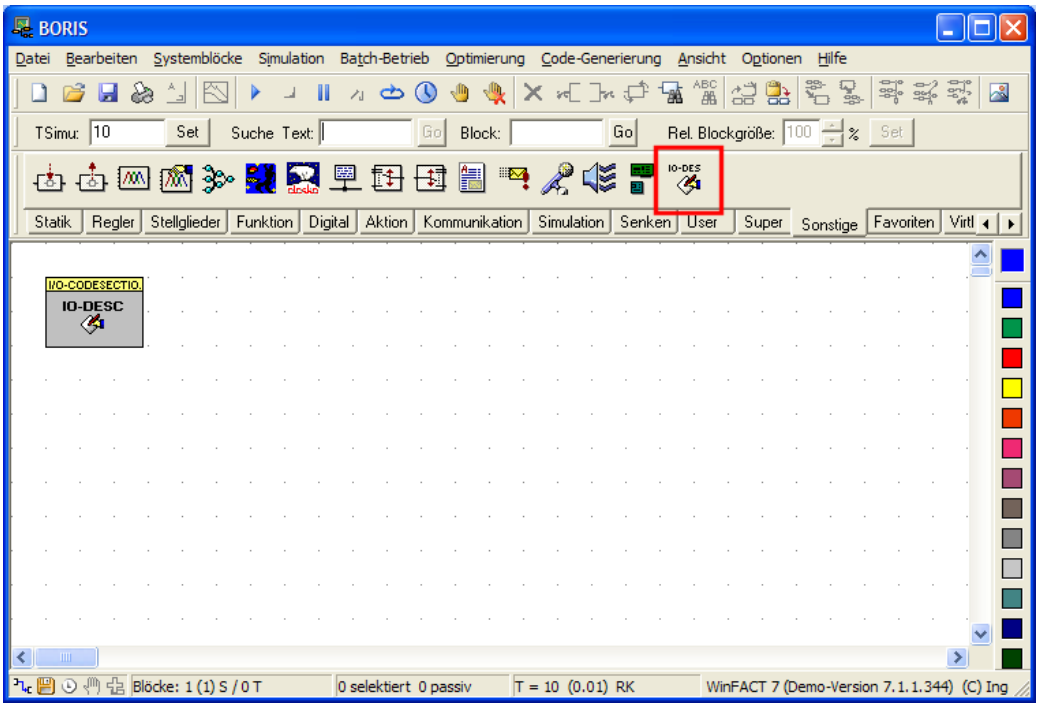

Abbildung 8 Blocktyp I/O-Code-Abschnitt

> Nach dem Einfügen eines I/O-Code-Abschnitt-Blocks besitzt dieser wie in Abbildung 8 gezeigt zunächst keine Funktionalität und daher auch weder Ein- noch Ausgang. Erst nach Auswahl des entsprechenden Code-Abschnitts aus der I/O-Beschreibungsdatei weist der Block dann Ein- bzw. Ausgänge auf.

Eine I/O-Beschreibungsdatei enthält Angaben zu den auf der Ziel-SPS vorhandenen Ein- und Ausgängen und ist daher in der Regel sowohl hersteller- als auch konfigurationsabhängig. In den nachfolgenden Abschnitten wird eine mit der Demo-Version des SCL-Codegenerators installierte I/O-Beschreibungsdatei verwendet. Es ist daher möglich, dass Ihre SPS nicht über die Ein- und Ausgänge verfügt, die durch diese Datei beschrieben werden. Da im Rahmen der beschriebenen Arbeitsschritte aber keinerlei generierter SCL-Code an die SPS übertragen wird, ist dieser Umstand unwesentlich. In Abschnitt ### wird auf die Anpassung der I/O-Beschreibungsdatei an die anwendereigene SPS eingegangen.

3.2.2 Auswahl der I/O-Beschreibungsdatei

Die I/O-Beschreibungsdatei kann über die Menüoption CODE-GENERIERUNG | I/O-BESCHREIBUNGSDATEI | DIREKT WÄHLEN... ausgewählt werden. Dazu öffnet sich ein Dialog, der anzeigt, welche Abschnitte und Funktionen die aktuell ausgewählte I/O-Beschreibungsdatei enthält. Wählen Sie wie in Abbildung 9 gezeigt die Datei *WF7_SPS_Beispiel_IO.scl* aus dem Unterverzeichnis *AutoCode\SCL* Ihrer WinFACT-Installation als I/O-Beschreibungsdatei aus. Im unteren Teil des Dialogs werden nun alle in der ausgewählten Datei gefundenen Code-Abschnitte aufgelistet.

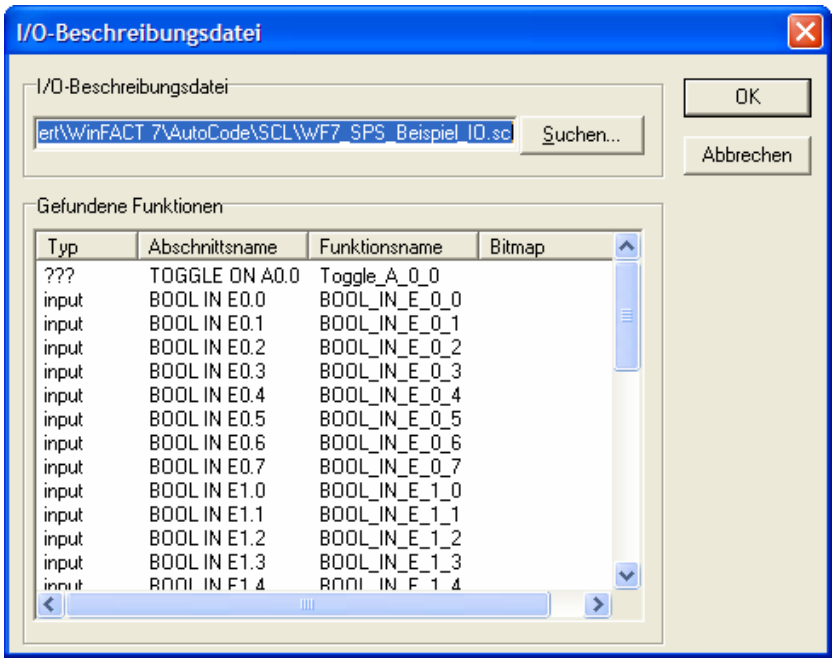

Abbildung 9 Direkte Auswahl einer I/O-Beschreibungsdatei

3.2.3 Einfügen eines SPS-Ein-/Ausgangs als Block

Nachdem eine I/O-Beschreibungsdatei ausgewählt wurde, stehen die in der Beschreibungsdatei enthaltenen Code-Abschnitte zur Parametrierung des *I/O-Code-Abschnitt*-Blocktyps zur Verfügung. Wir wollen zur Verdeutlichung das in Abschnitt 3.1 vorgestellte Beispiel derart erweitern, dass die beiden Eingänge des UND-Logikgatters mit den Eingängen E0.0 und E0.1 der SPS verbunden werden und der Ausgang des UND-Gatters mit dem SPS-Ausgang A0.0. Gehen Sie dazu wie folgt vor:

1) Löschen Sie eine ggf. noch vorhandene Systemstruktur.
2) Fügen Sie ein Logikgatter mit zwei Eingängen ein und verschieben Sie es ein wenig nach rechts.
3) Öffnen Sie durch Anklicken mit der rechten Maustaste das Popup-Menü des Logikgatters und fügen Sie den Block über die Menüoption IN CODE-GENERIERUNGSLISTE EINTRAGEN in die Code-Generierungsliste ein.
4) Fügen Sie links vom Logikgatter-Block zwei *I/O-Code-Abschnitt*-Blöcke ein, rechts davon einen. Ihr BORIS-Arbeitsblatt sollte nun Abbildung 10 entsprechen.

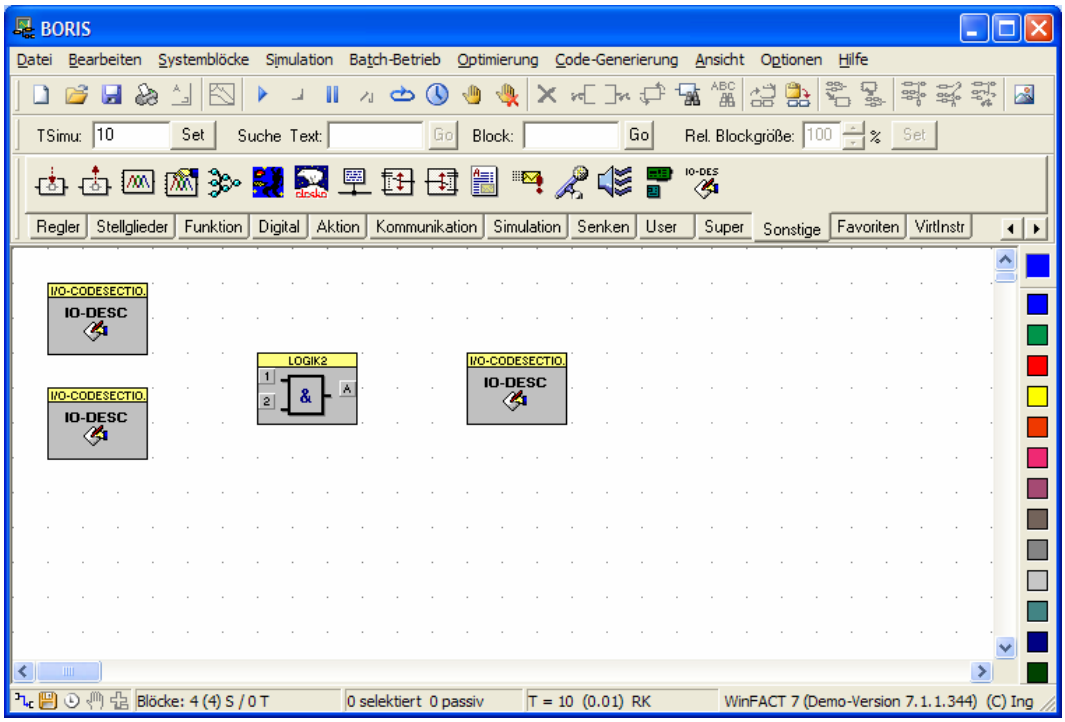

Abbildung 10 BORIS nach Einfügen aller Blöcke

5) Doppelklicken Sie nun zunächst auf den oberen der beiden linken *I/O-Code-Abschnitt*-Blöcke. Wählen Sie im daraufhin erscheinenden Dialog den Code-Abschnitt `BOOL IN E0.0` aus (Abbildung 11) und verlassen Sie den Dialog über die *OK*-Schaltfläche. Der Block weist nunmehr einen Ausgang auf und zeigt zudem in seinem Blocktitel den ausgewählten Abschnittsnamen an.

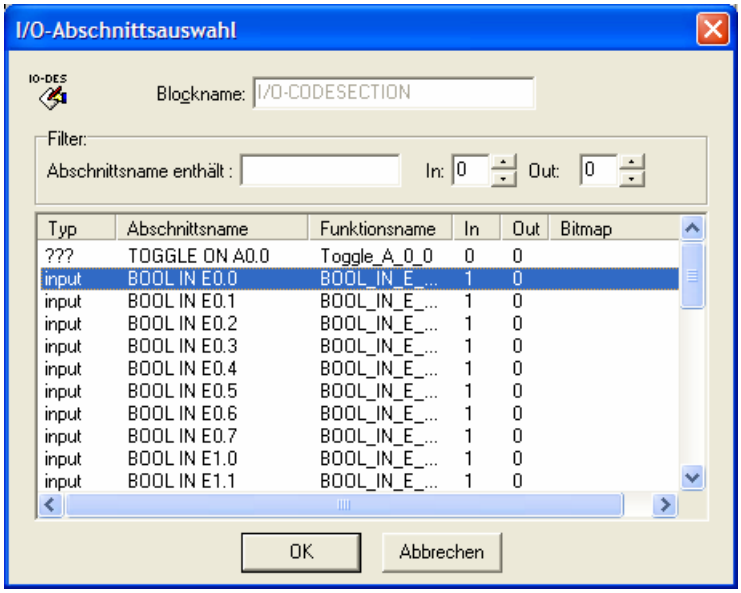

Abbildung 11 Parametrierung des *I/O-Code-Abschnitt*-Blocks als SPS-Eingang E0.0.

6) Weisen Sie nun auf analoge Weise dem unteren linken *I/O-Code-Abschnitt*-Block den Code-Abschnitt `BOOL IN E0.1` und dem rechten *I/O-Code-Abschnitt*-Block den Abschnitt `BOOL OUT A0.0` zu. Anschließend sollte Ihr BORIS-Arbeitsblatt etwa wie in Abbildung 12 aussehen.

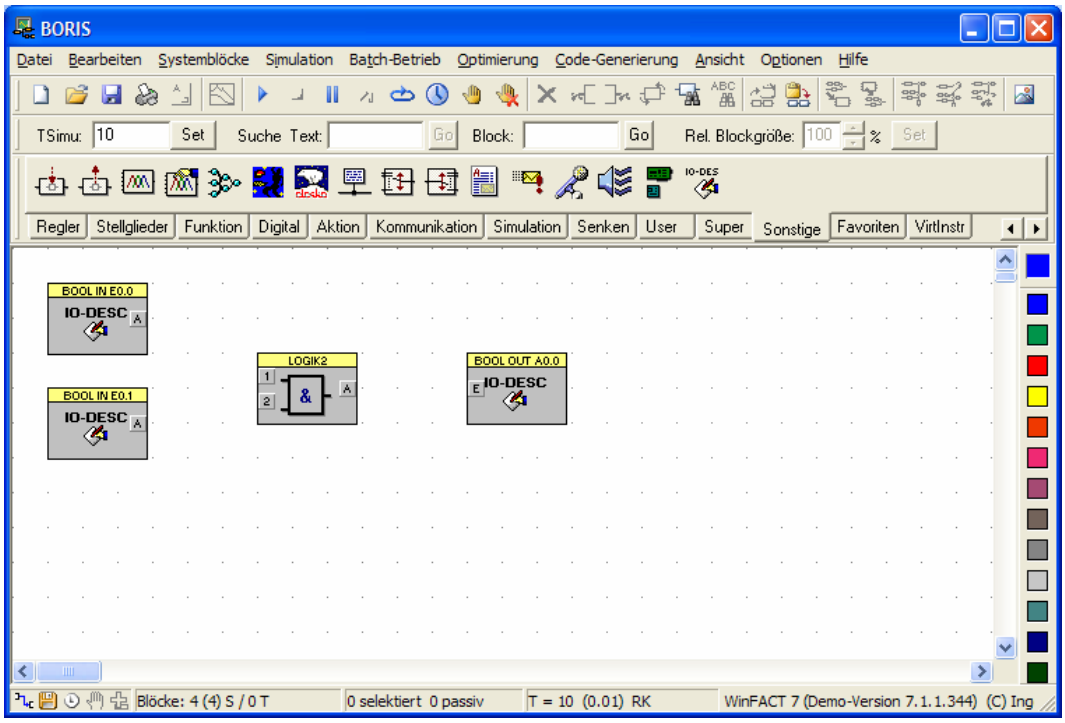

Abbildung 12 BORIS nach Parametrierung der *I/O-Code-Abschnitt*-Blöcke

7) Verbinden Sie abschließend die Blöcke wie in Abbildung 13 gezeigt.

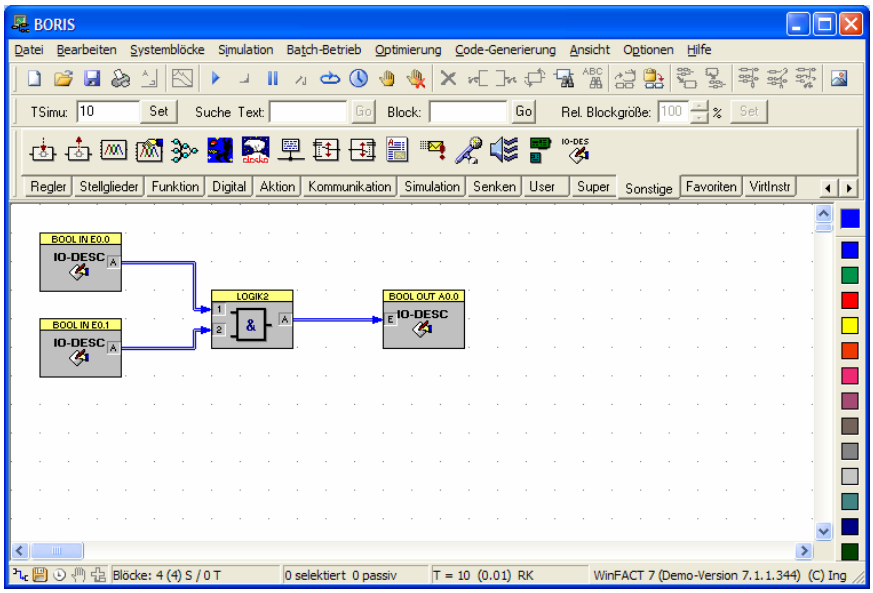

Abbildung 13 BORIS nach Verbinden der Blöcke

Die Systemstruktur ist nunmehr fertig gestellt. Wählen Sie wie bereits in Abschnitt 3.1 beschrieben die zu erzeugenden Code-Dateien aus und starten Sie die Codegenerierung. Listing 4 zeigt den `CONTROL`-Abschnitt des nunmehr erzeugten SCL-Codes. Da die zu Code generierte Systemstruktur im Gegensatz zum in Abschnitt 3.1 behandelten Beispiel keine offenen Ein- bzw. Ausgänge besitzt, weist der generierte Funktionsbaustein weder Ein- noch Ausgangsparameter auf. Stattdessen werden die Eingangswerte des UND-Gatters jetzt über die entsprechenden Funktionsaufrufe von den SPS-Eingängen E0.0 bzw. E0.1 eingelesen und der Ausgangswert des Gatters auf SPS-Ausgang A0.0 geschrieben; die Funktionsaufrufe sind im Programmlisting durch Fettschrift hervorgehoben. Die entsprechenden Funktionen wurden

dabei aus den entsprechenden Code-Abschnitten der I/O-Beschreibungsdatei übernommen und finden sich ebenfalls in der generierten SCL-Datei wieder (Listing 5).

```
FUNCTION_BLOCK Logik2_CONTROL
  VAR_INPUT
  END_VAR

  VAR_OUTPUT
  END_VAR

  VAR
    i : INT;
  END_VAR


  GLBVars.X_VEC[0,0]:=WF_BOOL_TO_REAL(BOOL_IN_E_0_1());
  GLBVars.X_VEC[2,0]:=WF_BOOL_TO_REAL(BOOL_IN_E_0_0());
  GLBVars.X_VEC[1,0]:=WF_BOOL_TO_REAL(Logic2_fct(p:=GLBVars.LOGIK2_V2,
    e1:=WF_REAL_TO_BOOL(GLBVars.X_VEC[2,0]),
    e2:=WF_REAL_TO_BOOL(GLBVars.X_VEC[0,0])));
  BOOL_OUT_A_0_0(e:=WF_REAL_TO_BOOL(GLBVars.X_VEC[1,0]));

  (* set all external outputs from the state of the system *)
  FOR i:=0 TO MAX_X_VEC()-1 DO
    GLBVars.X_VEC[i,1]:=GLBVars.X_VEC[i,0];
  END_FOR;
END_FUNCTION_BLOCK
```

Listing 4 CONTROL-Funktionsbaustein für Beispiel

```
  FUNCTION BOOL_IN_E_0_0:BOOL
    BOOL_IN_E_0_0:=%E0.0;
  END_FUNCTION

  FUNCTION BOOL_IN_E_0_1:BOOL
    BOOL_IN_E_0_1:=%E0.1;
  END_FUNCTION

  FUNCTION BOOL_OUT_A_0_0:VOID
  VAR_INPUT
    e : BOOL;
  END_VAR
    %A0.0:=e;
  END_FUNCTION
```

Listing 5 Aus I/O-Beschreibungsdatei in SCL-Datei übernommene Code-Abschnitte für die SPS-Eingänge E0.0 und E0.1 bzw. SPS-Ausgang A0.0

Hinweis: Wird eine Struktur, die *I/O-Code-Abschnitt*-Blöcke enthält, innerhalb von BORIS simuliert, so sind die *I/O-Code-Abschnitt*-Blöcke dabei ohne Funktion. Besitzt ein solcher Block einen oder mehrere Ausgänge, so liefern diese während der Simulation einen Wert von 0.

Bei komplexeren SPS-Konfigurationen mit sehr vielen SPS-Ein- und/oder Ausgängen kann die entsprechende I/O-Beschreibungsdatei eine Vielzahl von I/O-Code-Abschnitten aufweisen, sodass die Abschnittsliste im Dialog des *I/O-Code-Abschnitt*-Blocks sehr lang und damit unübersichtlich wird. Der Dialog erlaubt es daher, die Abschnittsliste zu filtern, sodass nur die gerade relevanten Abschnitte angezeigt werden. Abbildung 14 zeigt dazu ein Beispiel, bei

dem lediglich solche Code-Abschnitte angezeigt werden, deren Name die Zeichenfolge „E0“ enthält.

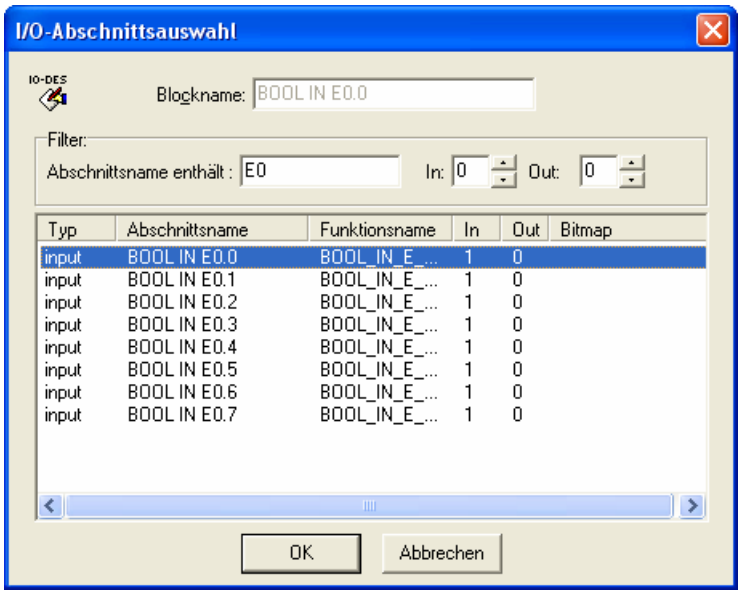

Abbildung 14 Parameterdialog des *I/O-Code-Abschnitt*-Blocks bei aktiver Filterung

Zudem ist zu beachten, dass bereits verwendete Ausgänge beim erneuten Aufruf des Dialogs nicht mehr in der Liste aufgeführt werden. Hierdurch wird eine versehentliche Doppelverwendung von Ausgängen verhindert.

3.2.4 Verwendung einer Rahmenfunktion

In der I/O-Beschreibungsdatei können sich zusätzlich zu den Code-Abschnitten zur Verwendung der Ein- und Ausgänge auch Abschnitte befinden, die eine *Rahmenfunktion* darstellen. Eine Rahmenfunktion erledigt den automatischen Aufruf der generierten Funktionsblöcke `xxx_INIT_CONTROL` oder `xxx_CONTROL`. Das grundsätzliche Aussehen der Rahmenfunktion entspricht dem im Abschnitt 3.1 dargestellten Funktionsblock. Da ein Code-Abschnitt mehr als nur eine SCL-Funktion enthalten darf, können alle zur Vervollständigung des generierten Codes notwendigen Bausteine wie der Organisationsbaustein OB1 dort abgelegt werden. In diesem Fall müssen Sie keine Zeile SCL-Code mehr selbst programmieren!

In der hier verwendeten I/O-Beschreibungsdatei *WF7_SPS_Beispiel_IO.scl* existieren zwei verschiedene Rahmenfunktionen, die vollständige SCL-Programme ergeben:

OB1 CALLS CONTROL BY TIMED PULSE / OB35 EMPTY

OB35 CALLS CONTROL/OB1 EMPTY

In der Regel sollten Sie die erste dieser beiden Rahmenfunktionen verwenden. Diese benutzt zur Erzeugung der zeitäquidistanten Funktionsbausteinaufrufe ein TP-Zeitglied [4]. Für Systeme, die mit einer Abtastzeit im Bereich von 100 ms und mehr arbeiten, reicht die hierdurch erzielte Genauigkeit vollkommen aus. Stellt das Programm fest, dass die Abtastschrittweite nicht hinreichend genau eingehalten wird, so wird diese automatisch vergrößert. Auf diese Weise erhalten Sie ein Feedback, mit welcher Abtastrate das System überhaupt auf der SPS laufen kann.

[4] TP steht für *Timed Pulse* – dies ist ein durch die IEC 61131-3 genormter SPS-Funktionsbaustein

Die zweite Rahmenfunktion ist bei Einhaltung der Abtastschrittweite wesentlich exakter, da hier der Hardwaretimer der SPS (bei der S7-300/400) verwendet wird. Dieser muss in der Konfigurierung der SPS entsprechend eingerichtet werden.

Stellen Sie für das im vorangegangenen Abschnitt begonnene Beispiel nun als Rahmenfunktion die erste der obigen Alternativen ein, indem Sie die Menüoption CODE-GENERIERUNG | CODE-GENERIERUNGS-EINSTELLUNGEN... und innerhalb des Dialogs die Registerkarte *Rahmenfunktion* wählen. Dort lässt sich dann die gewünschte Auswahl vornehmen (Abbildung 15).

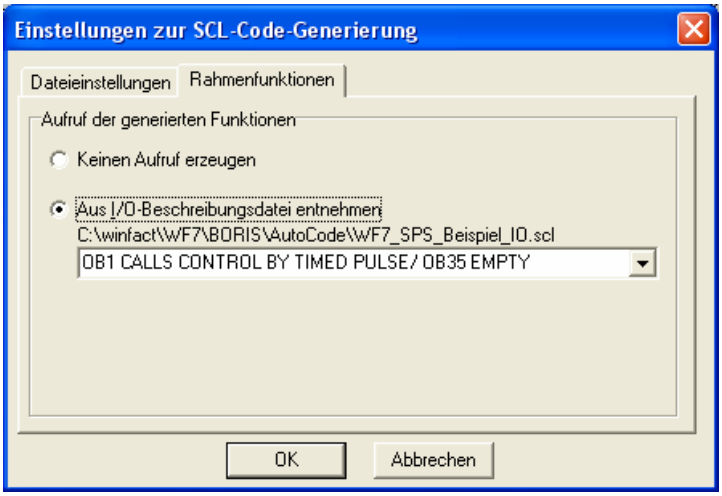

Abbildung 15 Auswahl der Rahmenfunktion

Das generierte SPS-Programm soll später auf der SPS mit einer Abtastzeit von 100 ms laufen; diese Zeit muss daher vor der Codegenerierung in BORIS als Simulationsschrittweite eingestellt werden Abbildung 16).

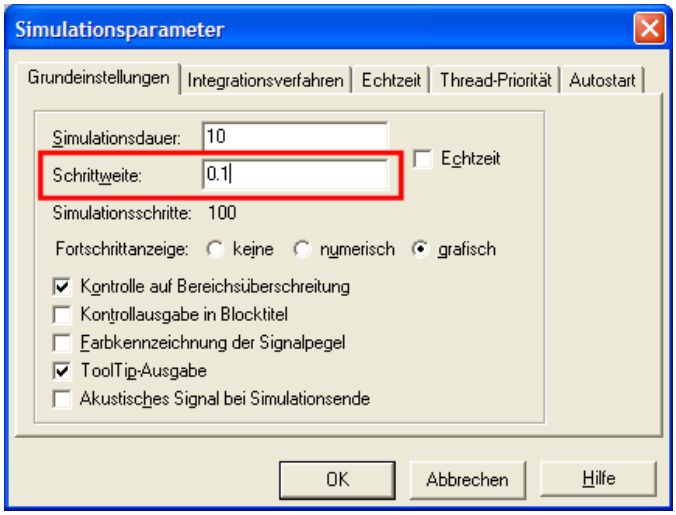

Abbildung 16 Einstellen der Abtastzeit bzw. Simulationsschrittweite

Führen Sie anschließend erneut eine Codegenerierung durch. Nach durchgeführter Codegenerierung erscheint ein Infofenster, welches auf eine eventuell nicht einhaltbare Abtastzeit hinweist (Abbildung 17).

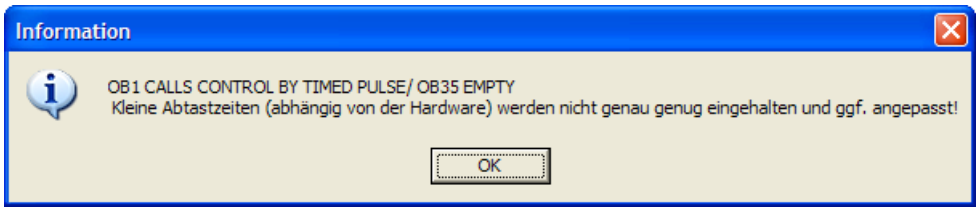

Abbildung 17 Infofenster bei Verwendung der Rahmenfunktion
OB1 CALLS CONTROL BY TIMED PULSE / OB35 EMPTY

In der generierten SCL-Datei finden wir die gewählte Rahmenfunktion nun im Funktionsbaustein `Logik2_CTRL` wieder, der vom Organisationsbaustein OB1 aufgerufen wird; OB35 ist bei Verwendung der gewählten Rahmenfunktion leer (Listing 6).

```
FUNCTION_BLOCK Logik2_CTRL
  VAR
    INITIALISIERUNGS_ANLAUF : BOOL:=TRUE;
    ictrl : Logik2_INIT_CONTROL;
    ctrl  : Logik2_CONTROL;
    ctrl_tp : TP;
         et          : TIME;
         check : DINT;
  END_VAR

   (*Only to have an easy view to the elapsed time*)
   et := ctrl_tp.ET;

  IF INITIALISIERUNGS_ANLAUF THEN
     GLBVars.DELTAT := TIME#0.1s;
         check:=0;
         (* use DINT_TO_REAL(TIME_TO_DINT(GLBVars.DELTAT)) to have a
         real value *)
    ctrl_tp(PT:=GLBVars.DELTAT, IN:=TRUE);
    ictrl();
    INITIALISIERUNGS_ANLAUF:=FALSE;
  ELSE
    IF FALSE THEN
      IF (NOT ctrl_tp.Q) THEN
        (*Endlosschleife zur Zykluszeitüberwachung. *)
        IF check<5 THEN
         GLBVars.DELTAT := GLBVars.DELTAT + GLBVars.DELTAT / 20;
               ctrl_tp(PT:=GLBVars.DELTAT, IN:=TRUE);
               ictrl();
         ELSE
               ctrl_tp(IN:=TRUE);
               ctrl();
         END_IF;
         check:=0;
         END_IF;
      check:=check+1;
       ctrl_tp(IN:=FALSE);
    ELSE
       ctrl();
     END_IF;
  END_IF;

END_FUNCTION_BLOCK
```

Listing 6 (Anfang) Funktionsbaustein (Rahmenfunktion) `Logik2_CTRL`

```
ORGANIZATION_BLOCK OB1
VAR_TEMP
  HEADER : ARRAY[1..20] OF BYTE;
END_VAR
  Logik2_CTRL.DB10();
END_ORGANIZATION_BLOCK

ORGANIZATION_BLOCK OB35
VAR_TEMP
  HEADER : ARRAY[1..20] OF BYTE;
END_VAR
   ;
END_ORGANIZATION_BLOCK
```

Listing 6 (Fortsetzung) Organisationsbausteine OB1 und OB35

3.3 Beispiel Ablaufsteuerung

Als etwas komplexeres Beispiel soll nun eine Ablaufsteuerung gemäß Abbildung 18 programmiert werden. Die Ablaufsteuerung wird über die Eingänge E0.0 und E0.7 gestartet. Anschließend kann über die Eingänge E0.1 und E0.2 in den jeweils nächsten Schritt geschaltet werden. Über E0.3 kann die Steuerung wieder zurückgesetzt werden.

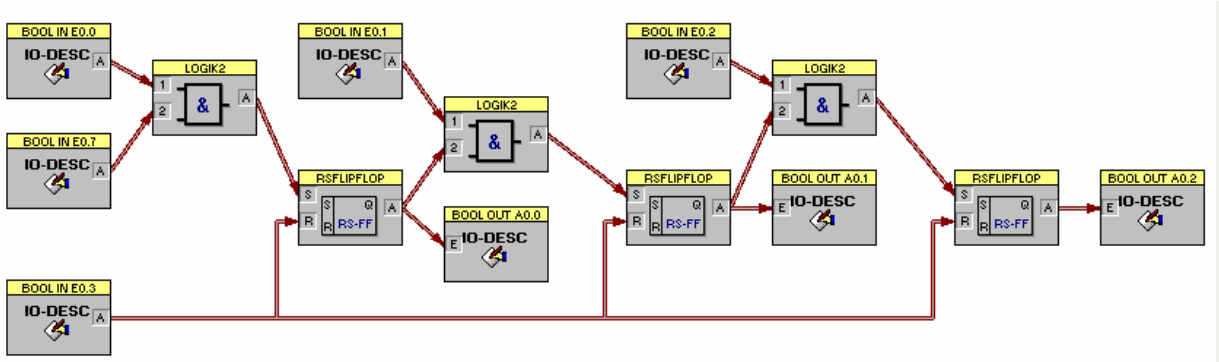

Abbildung 18 Ablaufsteuerung als Blockstruktur

Der Aufbau dieses Systems ist schnell von Hand erledigt (Parametereinstellungen sind nur bei den *I/O-Code-Abschnitt*-Blöcken erforderlich). Falls Sie diesen Schritt überspringen möchten, so finden Sie die fertige BORIS-Struktur in Ihrem WinFACT 7-Programmverzeichnis unter *Examples/SCL-Code-Generator/Ablaufsteuerung.bsy*.

Der erzeugt Code besteht wiederum aus einer SCL- und einer SDF-Datei. Die SCL-Datei entspricht strukturell der im Abschnitt 3.2.4 erzeugten Datei, ist aber aufgrund der komplexeren Systemstruktur naturgemäß wesentlich größer. Die SDF-Datei entspricht der Symboltabelle und deklariert die in der SCL-Datei verwendeten Bausteine und Symbole. Dies sind:

- Benutzerdefinierte Typen (z. B. Strukturen),
- Datenbausteine,
- Funktionen,
- Funktionsbausteine und
- Organisationsbausteine.

In der Regel werden beide Dateien vom SCL-Compiler der SPS-Programmierumgebung benötigt.

3.4 Anpassung des Codegenerators an die eigene SPS

Die Anpassung der I/O-Beschreibungsdatei an eine bestimmte SPS-Konfiguration (verwendete Ein- und Ausgänge) per Hand ist recht umständlich und fehleranfällig. Der SCL-Codegenerator bietet daher eine komfortable Möglichkeit, die I/O-Beschreibungsdatei anhand eines vorgegebenen Schemas automatisch zu erzeugen. Dazu dient die Menüoption CODE-GENERIERUNG | I/O-BESCHREIBUNGSDATEI | ANHAND EINES SCHEMAS ERZEUGEN... . Es erscheint der Dialog des so genannten *Schema-Umsetzers*, der zur Erzeugung der I/O-Beschreibungsdatei zwei Dateien benötigt: eine *Code-Schemadatei* sowie eine *Code-Parameterdatei*. Die in den vorangegangenen Abschnitten jeweils benutzte I/O-Beschreibungsdatei *WF7_SPS_Beispiel-IO.SCL* wurde auf Basis der Code-Schemadatei *WF7_AC_SPS_IO.CSH* sowie der Code-Parameterdatei *WF7_AC_Beispiel-IO.CPF* erstellt (Abbildung 19). Beide Dateien finden Sie im Unterverzeichnis *AutoCode\SCL* Ihrer WinFACT 7-Installation.

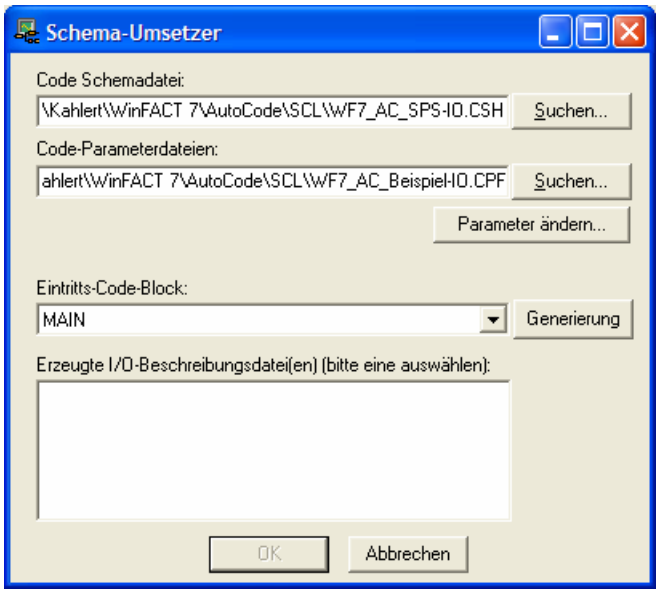

Abbildung 19 Schema-Umsetzer zur automatischen Erzeugung einer I/O-Beschreibungsdatei

Auf die Nutzung des Schema-Umsetzers zur Erzeugung eigener I/O-Beschreibungsdateien soll im Rahmen dieser Einführung nur in aller Kürze eingegangen werden. Löschen Sie zunächst eine eventuell noch vorhandene BORIS-Systemstruktur. Wählen Sie anschließend die in Abbildung 19 gezeigte Code-Schemadatei bzw. Code-Parameterdatei aus und betätigen Sie anschließend die Schaltfläche *Parameter ändern...* Sie gelangen daraufhin in einen Dialog, in dem Sie die in der Code-Parameterdatei spezifizierten Parameterwerte ändern können (Abbildung 20).

Ändern Sie nun die Menge der SPS-Adressen, die als boolesche Eingänge (Wert des Parameters *Bool_Inputs*) arbeiten sollen, wie folgt:

Von: *0, 1*

Auf: *3..5, 21, 32*

Des Weiteren soll der Parameter OUTFILENAME geändert werden, der den Namen der generierten I/O-Beschreibungsdatei festlegt:

Von: *WF7_SPS_Beispiel_IO.scl*

Auf: *Tutorial_Beispiel_IO.scl*

Hierdurch werden insgesamt 32 (jeweils 8 für jeden Wert des Parameters *Bool_Inputs*) verschiedene boolesche Eingänge erzeugt (%E3.0, %E3.1 … %E3.7, %E4.0 …) und die neu zu

erzeugende I/O-Beschreibungsdatei erhält den Dateinamen *Tutorial_Beispiel_IO.scl* (Abbildung 21).

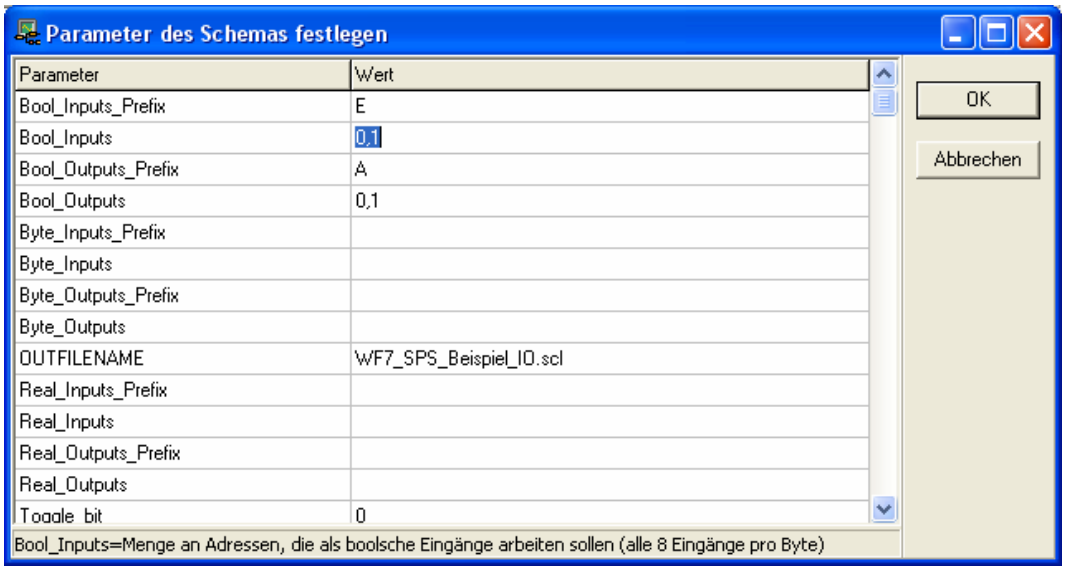

Abbildung 20 Dialog zur Modifizierung von Parametern eines Code-Schemas

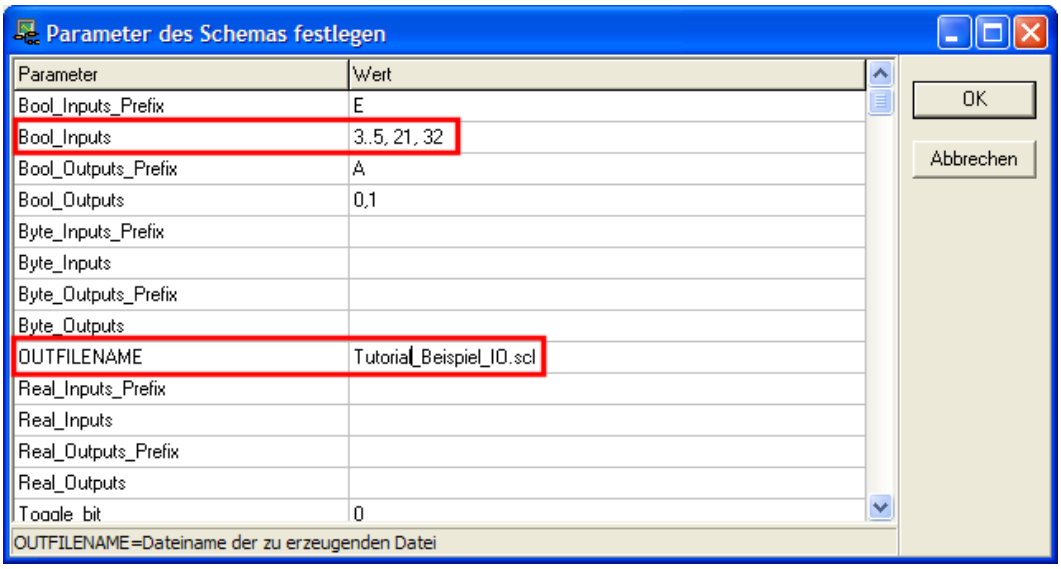

Abbildung 21 Dialog mit durchgeführten Änderungen

Beim Verlassen des Dialogs über die *OK*-Schaltfläche wird der neue Parametersatz in der verwendeten Code-Parameterdatei gespeichert. Wurde keine Code-Parameterdatei angegeben, so erscheint ein Dialog, der Sie zur Eingabe resp. Auswahl eines Dateinamens auffordert.

Im Dialog des Schema-Umsetzers betätigen Sie nun die Schaltfläche *Generierung*. Hierdurch wird die neue I/O-Beschreibungsdatei erzeugt und in die Liste der erzeugten I/O-Beschreibungsdateien eingetragen (Abbildung 22). Wählen sie diese anschließend durch Anklicken als aktuelle I/O-Beschreibungsdatei aus und verlassen den Dialog über die *OK*-Schaltfläche.

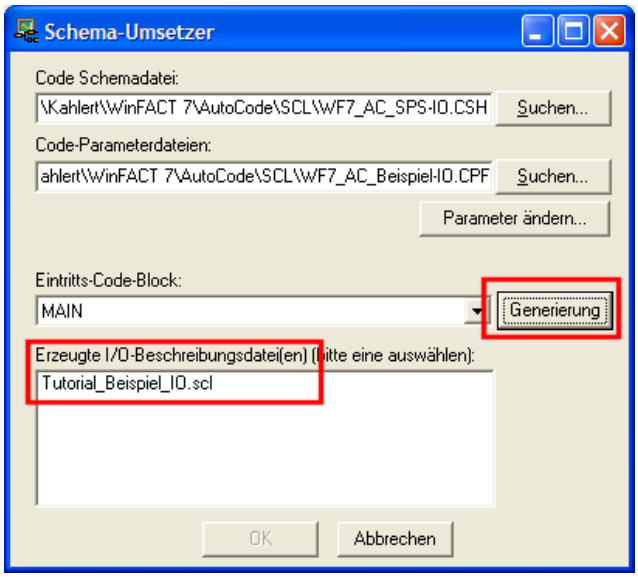

Abbildung 22 Dialog des Schema-Umsetzers nach Generierung der neuen I/O-Beschreibungsdatei

Fügen Sie zur Kontrolle nun einen *I/O-Code-Abschnitt*-Block ein und doppelklicken Sie ihn. Sie sollten nun auf die in der neu erstellten I/O-Beschreibungsdatei spezifizierten SPS-Eingänge zugreifen können (Abbildung 23).

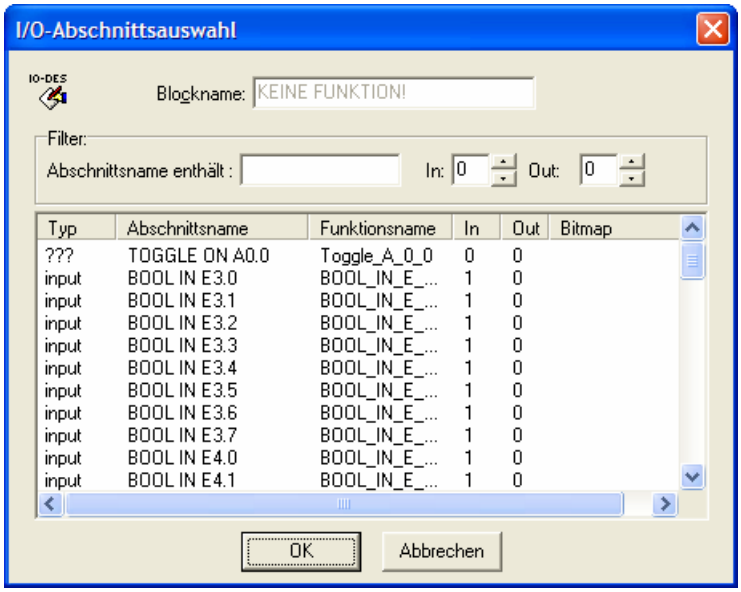

Abbildung 23 Dialog zur I/O-Abschnittsauswahl

3.5 Einbinden des generierten Codes in die SPS-Programmierumgebung (STEP 7)

Abschließend soll kurz das Einbinden der vom SCL-Codegenerator erzeugten Dateien in die SPS-Programmierumgebung erläutert werden. Dazu wird beispielhaft die Programmierumgebung STEP 7 der Firma Siemens gewählt.

Die generierte SCL-Datei enthält den eigentlichen Quelltext und ist daher in die Gruppe *Quellen* des S7-Programms einzufügen. Dazu klicken Sie innerhalb des SIMATIC Managers mit der rechten Maustaste auf das *Quellen*-Icon und wählen in den daraufhin erscheinenden Kontextmenüs die Option NEUES OBJEKT EINFÜGEN | EXTERNE QUELLE... (Abbildung 24 oben). Anschließend wählen Sie die einzubindende SCL-Datei (hier z. B. die in Abschnitt 3.1 gene-

rierte Datei *Logik2.scl*) aus. Die ausgewählte Datei wird daraufhin im rechten Teil des Programmfensters angezeigt (Abbildung 24 unten).

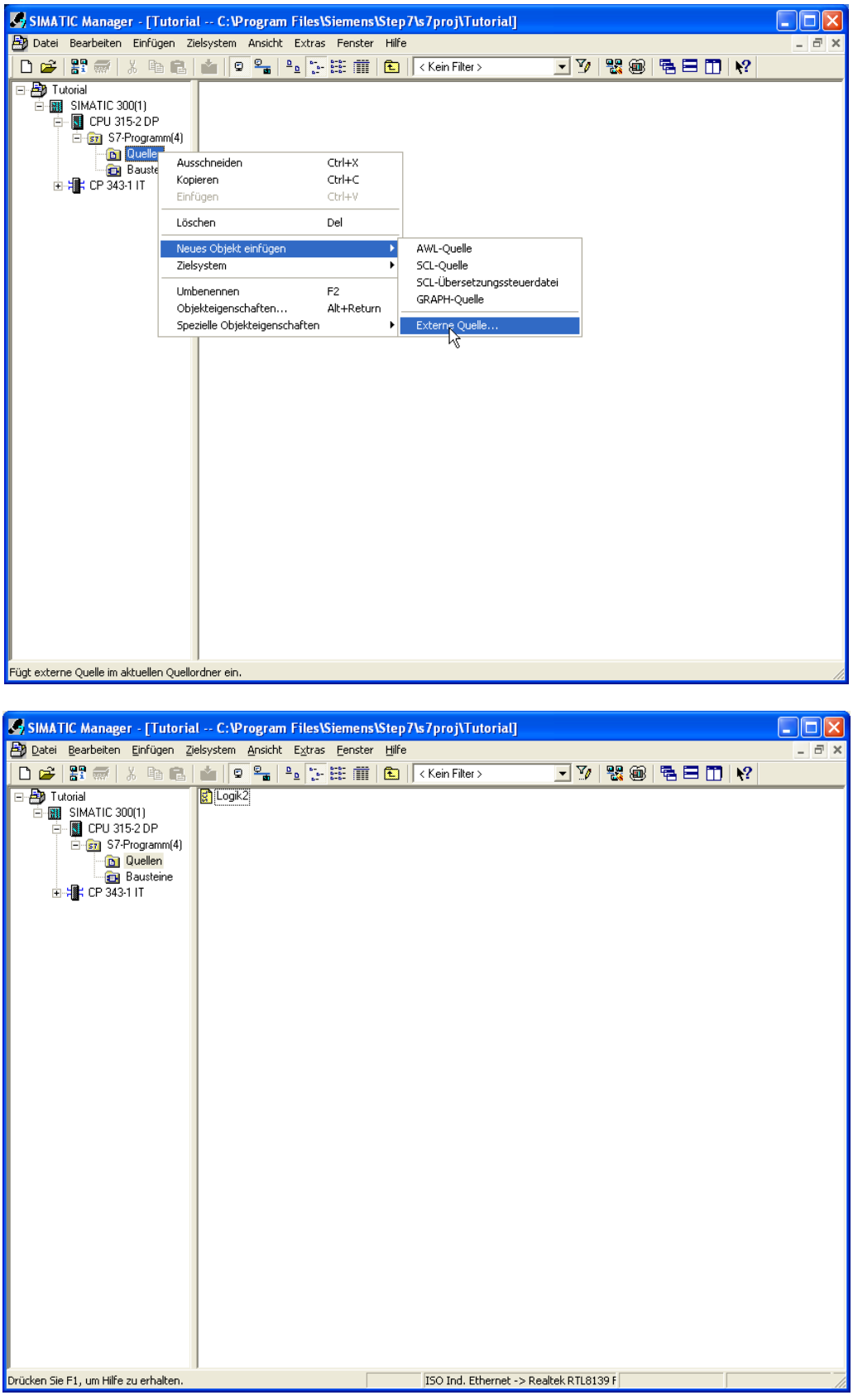

Abbildung 24 Einbinden einer SCL-Datei in STEP 7

Zum Einbinden der Symboltabellendatei (SDF-Datei) wechseln Sie in den Symbol-Editor von STEP 7 und wählen dort die Menüoption TABELLE | IMPORTIEREN... (Abbildung 25 oben). Nach Auswahl der zu importierenden Symboltabelle (hier z. B. die in Abschnitt 3.1 generierte Datei *Logik2.sdf*) werden die importierten Symbole angezeigt (Abbildung 25 unten).

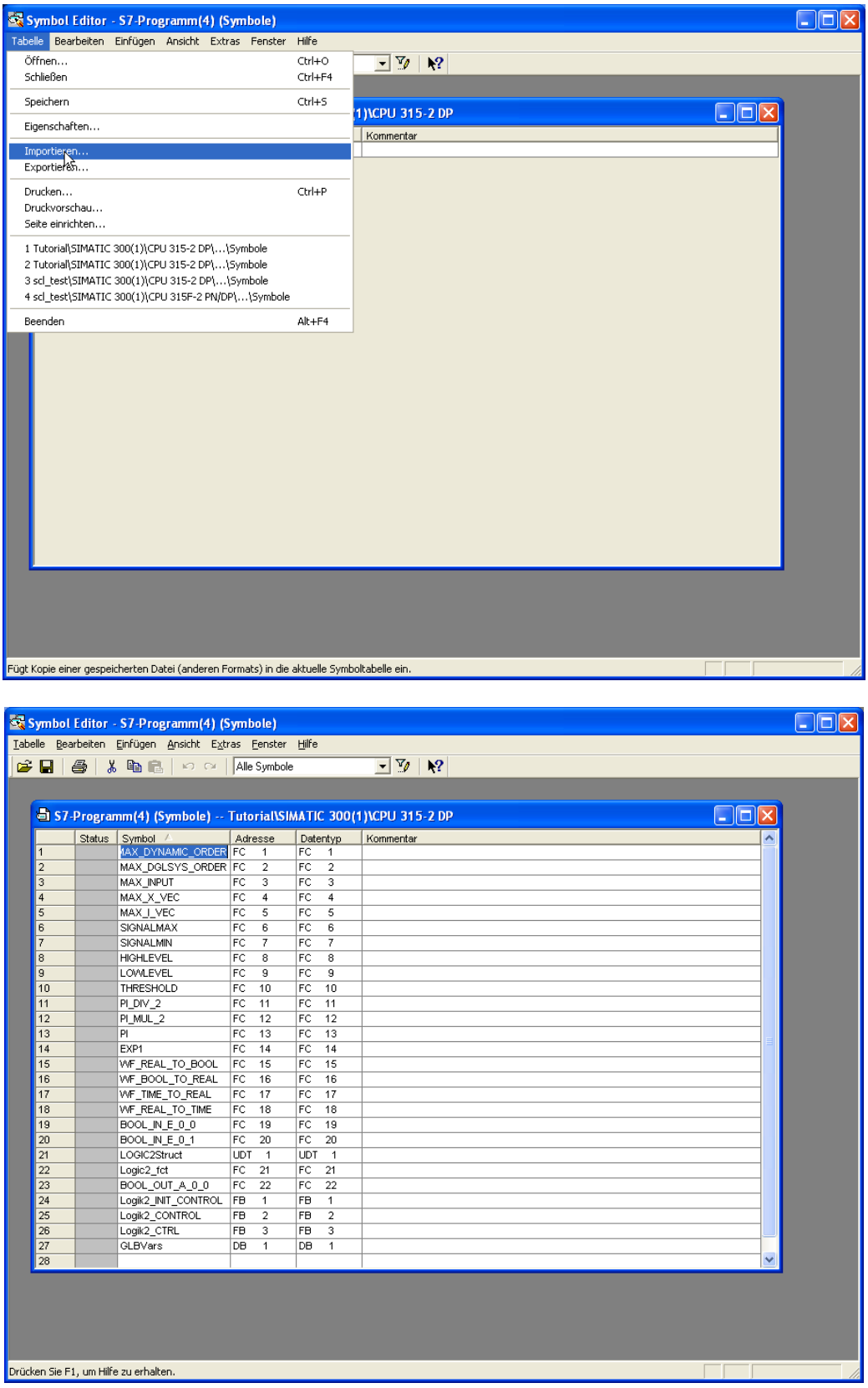

Abbildung 25 Einbinden der Symboltabellendatei in STEP 7