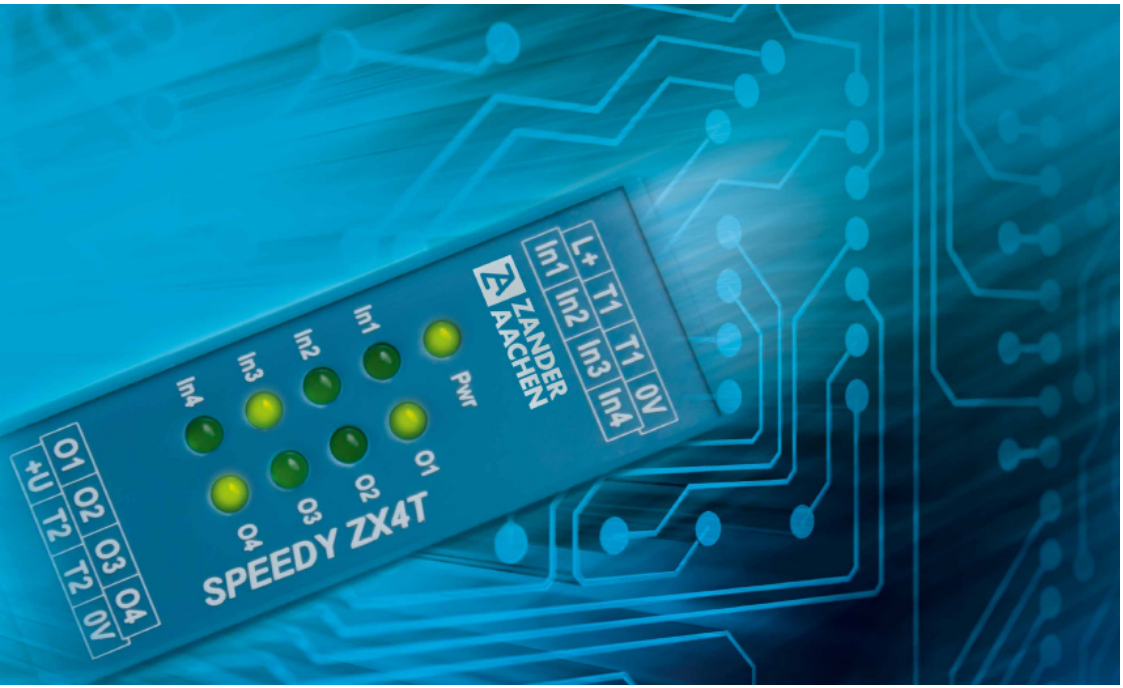
So einfach ist EX_PRESS 4

ZANDER AACHEN

Betriebsanleitung



Ein Programmierkurs für die SPS-Sprache Strukturierter Text nach IEC1131-3

So einfach ist EX_PRESS 4!

Ein Programmierkurs für die SPS-Sprache

Strukturierter Text nach IEC1131-3

Von Dipl.-Ing. Alfons Austerhoff

Dieser kleine Programmierkurs ermöglicht einen schnellen und leichten Einstieg in die Programmierung der ZANDER-Steuerungen EX16 und SPEEDY anhand von praktischen, leicht nachvollziehbaren Beispielen. Er vermittelt die wesentlichen Kenntnisse im Umgang mit der SPS-Programmiersprache "Strukturierter Text". Weitere Informationen bietet die Online-Hilfe der Programmiersoftware EX_PRESS 4.

© 1999, 2012, 2017 by H. ZANDER GmbH & Co. KG, Am Gut Wolf 15, 52070 Aachen
Kein Teil dieses Handbuches darf ohne schriftliche Genehmigung der ZANDER GmbH & Co. KG reproduziert oder vervielfältigt werden. Der Inhalt wurde mit großer Sorgfalt erstellt, jedoch kann keine Garantie für Fehlerfreiheit oder für Schäden übernommen werden, die aus der Benutzung dieses Handbuches resultieren.

Diese Dokumentation basiert auf EX_PRESS 4 V4.4, Februar 2017.

Zander GmbH & Co. KG
Am Gut Wolf 15
52070 Aachen, Deutschland
info@zander-aachen.de
www.zander-aachen.de

Ausgabe: J02

Dieses Dokument ist das Originaldokument.

Technische Änderungen vorbehalten, alle Angaben ohne Gewähr.

Inhaltsverzeichnis

1. Installation der Programmiersoftware EX_PRESS 4 4
2. Das Programmgerüst 4
3. Das erste EX_PRESS 4-Programm 6
3.1 Die Steuerung auswählen 10
3.2 Compilieren und Fitten 11
3.3 Download zur SPS 12
4. Logikoperatoren NOT, AND, OR, XOR 14
5. Selbsthaltung 15
6. Register 16
7. Timer 17
8. Interne Variablen 19
9. Die IF-Anweisung 20
10. Arrays 21
11. Vergleichsoperatoren 21
12. CASE-Anweisung 21
13. Online-Hilfe, Beispielprogramme 23
Anhang 1 . Befehle und Schlüsselwörter von EX_PRESS 4 24
Anhang 2 . Programmkapazität des CPLD 25
Anhang 3 . Häufig gestellte Fragen 26
Anhang 4 . EX_PRESS Troubleshooting 27

1. Installation der Programmiersoftware EX_PRESS 4

Für die Installation des Programmiersystems EX_PRESS 4 ist als Betriebs-system WINDOWS XP, 7, 8.1, 10 (32 or 64 bit) vorgesehen.

Für die Installation muss der USB-Stick mit dem Rechner verbunden werden. Über den Explorer bzw. die Funktion "Ausführen" des Startmenüs (Taskleiste) wird das Programm "EX_PRESS_Setup_de1.exe“ (deutsche Sprache) oder das „EX_PRESS_Setup_en2.exe“ (englische Sprache) gestartet. Jetzt führen Anweisungen auf dem Bildschirm durch den weiteren Installationsprozess.

Achtung: Eine parallele Installation der beiden Sprachen ist möglich. Jedoch wird bei De-installation einer Sprachversion die verbliebene Sprachversion unbrauchbar. Z.B.: Sie haben zuerst EX_PRESS_Setup_de1.exe(deutsche Sprache) installiert, und installieren danach EX_PRESS_Setup_en2.exe (englischen Sprache). Nach der Deinstallation von EX_PRESS 4e wird die verbleibende Version EX_PRESS 4d nicht mehr funktionieren.

2. Das Programmgerüst

Jedes EX_PRESS-Programm besitzt eine Struktur nach folgendem Schema (siehe Bild 1):

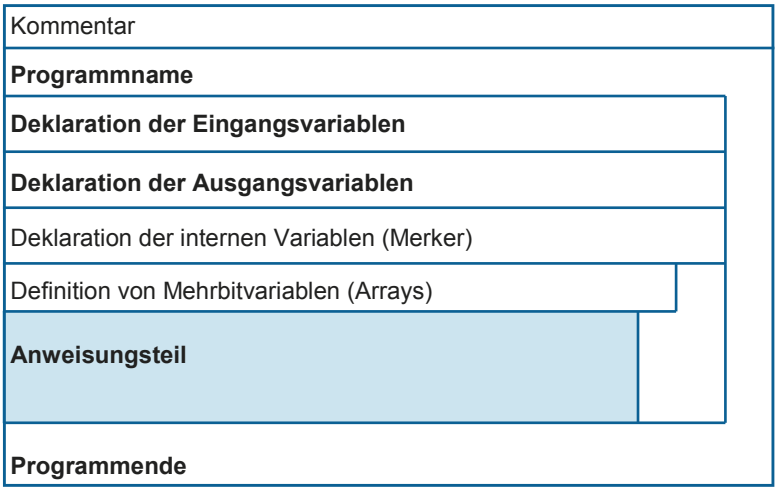


Bild 1: Das Programm-gerüst

Die fett gedruckten Einträge sind unbedingt erforderlich, die übrigen sind optional bzw. abhängig von der Anwendung.

Am besten lassen sich die einzelnen Programmteile erklären, indem wir unser erstes lauffähiges Programm schreiben. Es ist empfehlenswert, die einzelnen Schritte am PC nachzuvollziehen.

Das Programmiersystem *EX_PRESS 4* wird über das Startmenü der Taskleiste geöffnet. Es erscheint ein leeres Textfeld, in dem der Programmcode erstellt werden kann. Falls ein *EX_PRESS*-Programm geöffnet war, kann das leere Textfeld über den Menüpunkt "Neu" im Dateimenü erzeugt werden (siehe Bild 2).

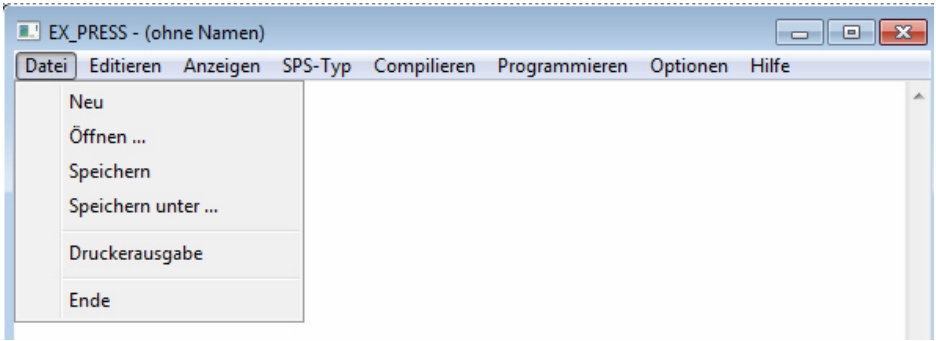


Bild 2: Neues Programm

Der Übersichtlichkeit halber ist das Programm in blau gedruckt und die Erklärungen dazu in schwarz.

3. Das erste EX_PRESS-Programm

Vor der eigentlichen Programmierung müssen wir uns natürlich Klarheit über die Funktionalität verschaffen, die unser erstes Beispiel erfüllen soll. Wir wollen eine Motorsteuerung entwerfen, bei der wir 3 Eingänge und 3 Ausgänge der SPS verwenden (Bild 3).

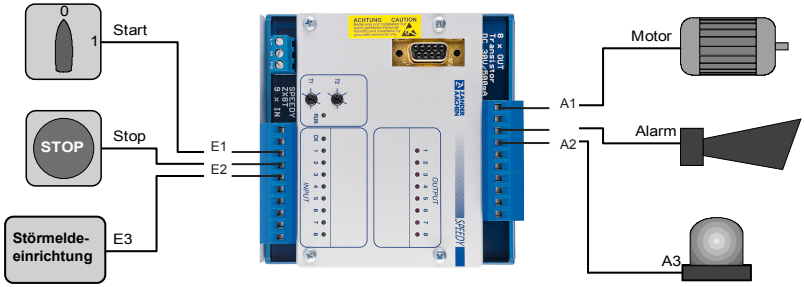


Bild 3: Die erste Applikation

Der Motor soll laufen, solange der Schalter *Start* geschlossen ist. Er darf aber nicht laufen, wenn der Eingang *Stop* ein Signal erhält, und er soll ebenfalls ausgeschaltet werden, wenn der Ausgang *Alarm* gesetzt ist. Der Ausgang *Alarm* soll sich invertiert zum Störmeldeeingang E3 verhalten, d.h. er schaltet ein, wenn das Signal an E3 wegfällt. Am Ausgang *A3* ist eine Warnleuchte angeschlossen. Er wird aktiv, wenn *Alarm* eingeschaltet und gleichzeitig der Schalter *Start* geschlossen ist.

Kommentar

Beginnen wir mit dem Kommentarteil. Dieser ist optional, aber empfehlenswert für die spätere Programmpflege, denn hier können eine kurze Funktionsbeschreibung, Erstellungsdatum, Versionsnummer und zum Beispiel Hinweise für die Inbetriebnahme eingetragen werden.

```
Dies ist das erste EX_PRESS-Programm. Es enthält alles
notwendige, um eine lauffähige EX16- oder Speedy-
Anwendung zu erzeugen.

Letzte Änderung: 20.10.1999
Version: 1.0
Autor: Dipl.-Ing. Alfons Austerhoff, H. ZANDER GmbH & Co.
KG, Aachen
```

Programmname

Im nächsten Schritt müssen wir dem Programm einen Namen geben. Dies geschieht mit dem Befehl "PROGRAM", der gleichzeitig dem Compiler mitteilt, dass hier der Kommentarteil endet und der eigentliche Programmtext beginnt. Nach einem Leerzeichen folgt der Name, der aus maximal 12 Zeichen bestehen darf.

Bitte beachten: Alle Befehle und reservierten Schlüsselwörter, wie hier der Befehl "PROGRAM", bestehen aus Großbuchstaben. Der Programmname und auch Variablennamen dürfen aus Groß– und/oder Kleinbuchstaben bestehen sowie ab dem zweiten Zeichen auch Ziffern enthalten.

Ab dem Schlüsselwort PROGRAM muss weiterer Kommentar in der Form **(* Kommentar *)** eingefügt werden.

```
PROGRAM Beispiel   (* Das Programm heißt Beispiel *)
```

Deklaration der Input Variablen

Damit Ein– und Ausgangssignale sowie Merker verwendet werden können, müssen entsprechende Variablen deklariert werden, das heißt, es werden Namen vereinbart, mit denen später im Anweisungsteil die logischen Zustände der Eingänge abgefragt bzw. Ausgänge oder Merker gesetzt werden können. Zunächst werden die Eingangsvariablen deklariert. In unserem Beispiel sollen 3 Eingänge abgefragt werden (Bild 4):

```
VAR_INPUT
     Start AT E1;
     Stop AT E2;
     E3 AT E3;
END_VAR;
```

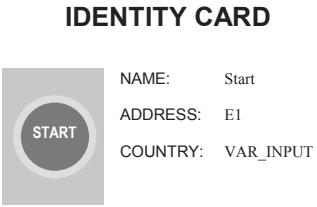


Bild 4: Input Variablen

Die Eingangsvariablen werden in der Form

<Name> AT <Ort>;

deklariert, wobei <Name> beliebig gewählt werden kann. Er darf sogar, wie im Beispiel bei der dritten Variablen zu sehen, identisch mit dem Ort sein.

<Ort> definiert die Zugehörigkeit der Variablen zu einer der Eingangsklemmen **E1** bis **E16** (Speedy: E1 bis E9).

Achtung: Jede Zeile der Variablendeklaration muss mit einem Semikolon abgeschlossen werden, auch hinter den Compilerbefehl **END_VAR** gehört ein Semikolon.

Deklaration der Output Variablen

Die Deklaration der Ausgangsvariablen hat die gleiche Form wie die der Eingangsvariablen:

<Name> AT <Ort>;

Der <Name> kann wieder frei gewählt werden, <Ort> definiert die Zuordnung zu einer der Ausgangsklemmen **A1** bis **A16** (Speedy A1 bis A8).

```
VAR_OUTPUT
    Motor AT A1;
    Alarm AT A2;
    A3 AT A3;
END_VAR;
```

Die Deklaration von internen Variablen und Arrays werden wir später behandeln, da diese nur bei bestimmten Problemstellungen notwendig sind.

Anweisungsteil

Nachdem nunmehr alle Variablennamen, die benutzt werden sollen, festgelegt sind, folgt jetzt der wichtigste Teil unseres Programmes, nämlich der Anweisungsteil, in dem die von der Steuerung auszuführenden Funktionen programmiert werden. Jeder Ausgangsvariablen muss dabei ein logischer Zustand zugewiesen werden. Dies geschieht in Form einer Zuweisungsgleichung:

<Ausgangsvariable> := <Logischer Ausdruck>;

<Ausgangsvariable> ist ein im Bereich *VAR_OUTPUT* deklarierter Name, das Zeichen := ist der Zuweisungsoperator, <Logischer Ausdruck> besteht

aus einer Kombination von Variablennamen und den logischen Operatoren *AND*, *OR*, *XOR* und *NOT*.

Da alle Signale parallel verarbeitet werden, ist die Reihenfolge der Zuweisungen beliebig, so dass z.B. zuerst *A3*, dann *Motor* und zum Schluss *Alarm* programmiert werden kann.

Achtung: Wie bei der Deklaration der Variablen muss auch im Anweisungsteil jede Zuweisung mit einem Semikolon abgeschlossen werden.

```
A3 := Alarm AND Start;

Motor := Start AND NOT Stop AND NOT Alarm;

Alarm := NOT E3;
```

Damit ist das erste lauffähige Programm fertig gestellt, es fehlt nur noch die Anweisung an den Compiler, dass das Programmende erreicht ist:

```
END_PROGRAM;
```

Zur weiteren Bearbeitung muss das Programm jetzt auf der Festplatte gespeichert werden. Dazu wird mit der Maus der Menüpunkt "Speichern" im Dateimenü angeklickt (Bild 5). Es erscheint ein Fenster, das die Eingabe des Dateinamens ermöglicht: Wir wählen "Beispiel.s16".

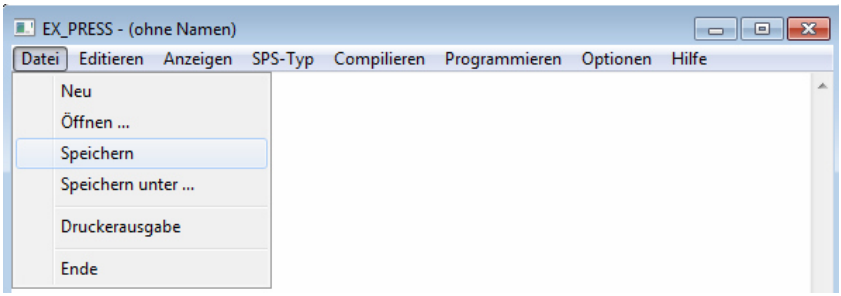


Bild 5: Speichern

Beim Drücken des Befehls „Öffnen“ im Menü Datei findet man im Ordner BeispieleEX_PRESS weitere Applikationen, welche nach Zander

Steuerungstyp (EX16 / SPEEDY ZX4T, ZX4TE, ZX8T) sortiert sind. Die Anwendungen FBand, Zaehler sind im Installationsverzeichnis abgelegt.

Der Standard-Pfad ist:

C:\Zander\EX_PRESS4d\BeispieleEX_PRESS

3.1 Auswahl der Steuerung

Das Menu SPS-Typ beinhaltet die folgenden Selektionsmöglichkeiten:

- EX16
- SPEEDY (ZX4T, ZX4TE, ZX8T)

EX_PRESS 4 unterstützt die Programmierung der Speicherprogrammierbaren Steuerungen ZX4T, ZX4TE, ZX8T und EX16, von denen Sie die Ziel-SPS mit diesen Menüpunkten durch einfaches Auswählen selektieren können. Die aktuelle SPS wird durch einen Haken (✓) in der Menüzeile visualisiert (Bild 6).

Bild 6: SPS Typ auswählen

Die Wahl des E-Typs betrifft die Auswahl für die verwendete Variante des CPLDs in der Steuerung. Für die EX16 wählen Sie bitte keinen E-Typ, für die SPEEDY-Familie ZX8T und ZX4T/TE den E-Typ.

Sollte es zu einer Fehlermeldung bezüglich des CPLD-Typs bei der Programmierung kommen, versuchen Sie bitte, das Programm mit dem geänderten E-Typ zu übersetzen (Compile und Fit).

3.2 Compilieren und Fitten

Das Programm muss nun in eine Struktur übersetzt werden, die in den programmierbaren Logikbaustein der Steuerung übertragen werden kann. Dies geschieht in zwei Schritten: Der Compiler sorgt für die Übersetzung des *Strukturierten Textes* in Boole'sche Gleichungen, es entsteht also ein logischer Zwischencode. Danach wird der Fitter gestartet, der den sogenannten Jedec-Code erzeugt, der in den Logikbaustein übertragen werden kann.

Um die genannten Schritte durchzuführen, muss im Menü "Compiler" der Unterpunkt "Compilieren und Fitten" angeklickt werden (Bild 7).

Bild 7: Compilieren und Fitten

Nach erfolgreichem Durchlauf erscheint im Anzeigefenster "LOG" die Meldung "Compilevorgang beendet, Fehler: 0, Warnungen: 0. Fitterdurchgang erfolgreich beendet".

Falls der Compiler Fehler entdeckt (Bild 8), können detaillierte Informationen, d.h. Art der Fehler und Zeilennummern, in denen sie lokalisiert wurden, über

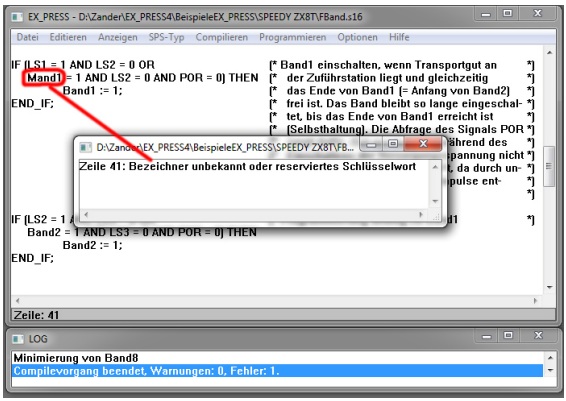


Bild 8: Fehler

das Menü "Anzeigen", Unterpunkt "Fehler" abgerufen werden (Bild 9).

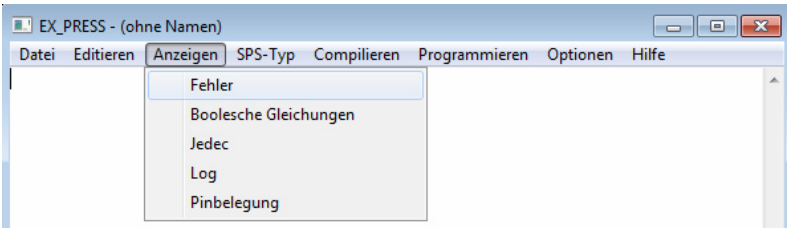


Bild 9: Anzeige Fehler

Meistens werden beim Auftreten eines einzelnen Fehlers mehrere Folgefehler erkannt, so dass immer der erste angezeigte Fehler korrigiert und das Programm dann neu compiliert werden sollte.

3.3 Download zur SPS

Zum Testen des Beispielprogramms müssen die Daten in die Steuerung übertragen werden. Dazu wird das Programmierkabel an die USB Schnittstelle des PC gesteckt und auf der anderen Seite mit der Steuerung verbunden. PC und Steuerung sollten dabei ausgeschaltet sein. Dann erst das EX_PRESS 4 Programm starten. (Das parallele Schnittstellenkabel läuft ebenfalls mit EX_PRESS 4 V4.4.)

Im Menü "Optionen" ist die verwendete Schnittstelle im Unterpunkt "Schnittstelle konfigurieren" auszuwählen (Bild 10).

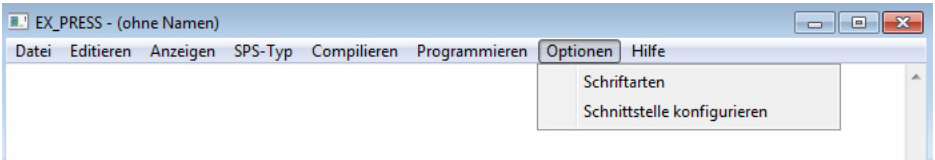


Bild 10: Schnittstelle konfigurieren

Überprüfe im Log Fenster ob das USB Kabel, bzw. die Steuerung erkannt wurde. Die automatische Treiberkonfiguration kann unter Umständen einige Momente dauern (Bild 11).

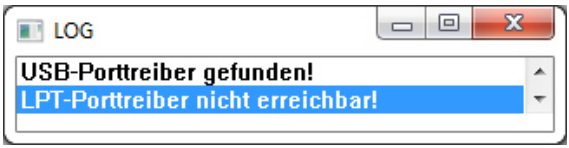


Bild 11: LOG

Nach Anlegen der Versorgungsspannung kann das Programm mit dem Menüpunkt "Download zum EX16/SPEEDY" im Menü "Programmieren" übertragen werden (Bild 12).

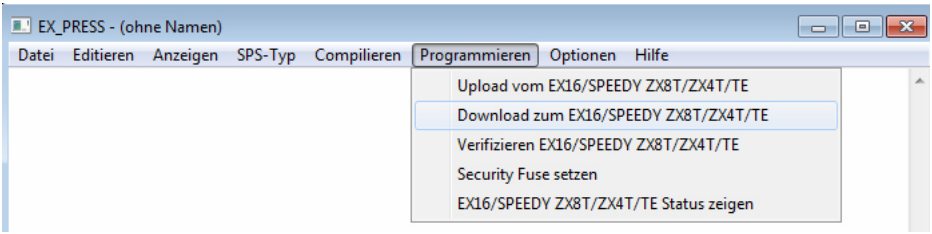


Bild 12: Download

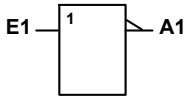
Wir haben unser erstes Anwendungsprogramm damit komplett erledigt und können uns von der korrekten Funktion durch Anlegen von Signalen an die Eingänge der Steuerung und Beobachten der Ausgangs-LEDs überzeugen.

Prinzipiell können nach diesem Schema sämtliche Problemstellungen gelöst werden, die nur logische Verknüpfungen der Ein– und Ausgangssignale erfordern. In den folgenden Kapiteln werden diese Kenntnisse noch ein wenig vertieft, zusätzlich werden Register und Timer erklärt, und zum Schluss werden spezielle Programmkonstrukte des Strukturierten Textes, die IF– und CASE-Anweisungen behandelt.

4. Logikoperatoren NOT, AND, OR, XOR

Mit den Logikoperatoren NOT, AND, OR und XOR können alle logischen Steuerungsaufgaben realisiert werden. Die nachfolgende Übersicht (Bild 13) zeigt für jede Verknüpfungsart einen grafischen Funktionsbaustein, die Wahrheitstabelle und die Syntax im *Strukturierten Text.*

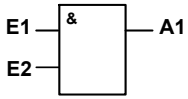
NOT



E1	A1
0	1
1	0

```
A1 := NOT E1;
```

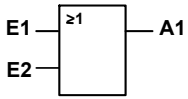
AND



E1	E2	A1
0	0	0
0	1	0
1	0	0
1	1	1

```
A1 := E1 AND E2
```

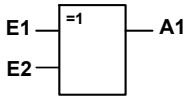
OR



E1	E2	A1
0	0	0
0	1	1
1	0	1
1	1	1

```
A1 := E1 OR E2;
```

XOR

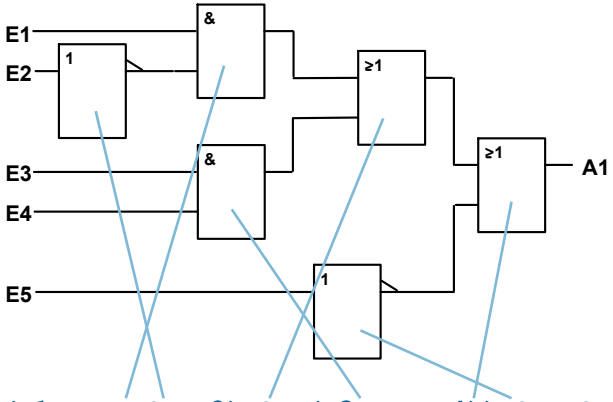


E1	E2	A1
0	0	0
0	1	1
1	0	1
1	1	0

```
A1 := E1 XOR E2;
```

Bild 13: Logkoperatoren

Wenn eine Zeichnung in Form von Logiksymbolen vorliegt, ist es relativ einfach, diese in *Strukturierten Text* umzuwandeln. Man muss nur darauf achten, dass alle Eingangssignale, die zu einem Logiksymbol gehören, in Klammern zusammengefasst werden (Bild 14).




```
A1 := ((E1 AND NOT E2) OR (E3 AND E4)) OR NOT E5;
```

Bild 14: Eingangsgruppensignale

5. Selbsthaltung

Im Beispielprogramm wurde ein Motor durch Aktivieren des Eingangs "Start" eingeschaltet. Sobald das Signal nicht mehr anliegt, wird der Motor ausgeschaltet. Es wäre aber komfortabler, den Motor durch kurzes Drücken eines Tasters "Start" einzuschalten und durch Betätigen eines Tasters "Stop" wieder auszuschalten. Dafür sind folgende Änderungen im Beispiel erforderlich:

Die Eingangsvariable POR (=Power-On-Reset) wird der Deklaration hinzugefügt:

```
VAR_INPUT
      ...
      POR AT POR;
END_VAR;
```

Das Signal POR ist ein Einschalt-Reset-Signal, das durch eine spezielle Baugruppe in der Steuerung erzeugt wird und beim Einschalten der Betriebsspannung für ca. 200ms aktiv ist. Dieses wird benötigt, damit der Ausgang "Motor" nicht ungewollt in Selbsthaltung geht, weil die Signalpegel während des Einschaltens kurzzeitig undefiniert sind.

Die Zuweisungsgleichung für "Motor" wird durch eine Programmzeile erweitert:

```
Motor := Start AND NOT Stop AND NOT Alarm
         OR Motor AND NOT Stop AND NOT Alarm AND NOT POR;
```

Wenn der Starttaster betätigt wird, ist der Teil vor dem OR wahr, so dass der Motor eingeschaltet wird. Sobald dies geschehen ist, wird der Teil hinter dem OR ebenfalls wahr, so dass der Motor weiterläuft, wenn der Starttaster losgelassen wird. Erst die Betätigung des Stop-Tasters oder die Aktivierung des Ausgangs "Alarm" bewirken, dass der Teil hinter dem OR logisch 0 wird und somit den Motor ausschaltet. Die Selbsthaltung kann durch das Einfügen von "AND NOT POR" aber erst dann aktiviert werden, wenn das Einschalt-Reset-Signal auf 0 abgefallen ist.

6. Register

Der programmierbare Logikbaustein des EX16 bzw. SPEEDY besitzt neben den beschriebenen AND-, OR– und XOR-Gattern zusätzlich Register, durch die logische Zustände gespeichert werden können. Am einfachsten erkennt man die Funktion in folgender Grafik (Bild 15):

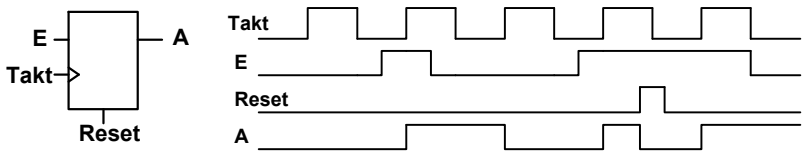


Bild 15: Register

Die Programmierung von Registern mit EX_PRESS erfolgt mit dem Befehl

<Variable>.CLK := <Taktsignal>;

Bei Bedarf kann das Register über seinen Reset-Eingang asynchron zurück-

gesetzt werden. Dieses Signal wird definiert über:

<Variable>.RE := <Reset-Signal>;

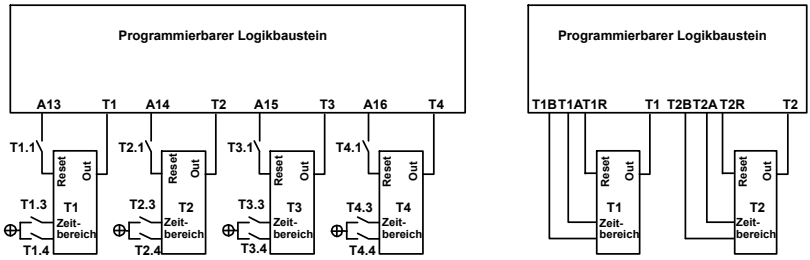
Das folgende Beispiel erzeugt exakt das in der Grafik dargestellte Ausgangssignal:

```
Programm zur Erläuterung der Register
PROGRAM Register

VAR_INPUT
     E AT E1;
     Takt AT T1;
     Reset AT E2;
END_VAR;

VAR_OUTPUT
     A AT A1;
END_VAR;

A.CLK := Takt;   (* Durch diese Zeile wird der Ausgang
                    ein Register *)
A.RE := Reset;   (* Zuordnung des asynchronen Reset *)
A := E;          (* Logikeingang des Registers *)
END_PROGRAM;
```

7. Timer

Im Programm "Register" wurde der Timer T1 als Taktgeber benutzt. Wenn nichts anderes angegeben ist, beginnt der Zeitablauf der Timer mit Anlegen der Betriebsspannung, jedoch besteht auch die Möglichkeit, ihn vom Programm aus zu starten und zu stoppen: Über die Reset-Eingänge können die Timer gestartet (Low) oder angehalten (High) werden.

Während bei EX16 die Zeitbereiche über DIL-Schalter eingestellt werden, können sie bei SPEEDY über reservierte Ausgänge des Logikbausteins verändert werden (siehe Bild 16).



EX16: Steuerung der Timer

SPEEDY: Steuerung der Timer

Bild 16: Steuerung der Timer

Eine Tabelle der Zeitbereiche und der zugehörigen Schalterstellungen bzw. Zustände der reservierten Ausgänge befindet sich in der Online-Hilfe unter dem Stichwort "Timer".

Im folgenden wird das Programm "Register" um die Funktion "anzugsverzögerter Ausgang" ergänzt, d.h. der Ausgang A schaltet nach Ablauf einer durch T1 vorgegebenen Zeit ein. Das Beispiel ist für die Steuerung *SPEEDY* vorgesehen, für EX16 bitte den eingefügten Kommentar beachten.

```
PROGRAM Register

VAR_INPUT
     E AT E1;
     Takt AT T1;
     Reset AT E2;
END_VAR;

VAR_OUTPUT
     A AT A1;
     T1Reset AT T1R;  (* EX16: T1Reset AT A13; *)
     T1A AT T1A;      (* Zeitbereich, entfällt bei EX16 *)
     T1B AT T1B;      (* Zeitbereich, entfällt bei EX16 *)
END_VAR;

A.CLK := Takt;        (* Durch diese Zeile wird der
                         Ausgang ein Register *)
A.RE := Reset;        (* Zuordnung des asynchronen Reset *)
```

```
A := 1;            (* A wird nach Ablauf der Zeit T1 (=Takt)
                     eingeschaltet, da am Logikeingang des
                     Registers immer eine 1 anliegt. Das
                     Ausschalten erfolgt durch Reset *)
T1Reset := NOT E; (* T1Reset wird 0, wenn E aktiv wird, und
                     damit läuft Timer T1 *)
END_PROGRAM;
```

8. Interne Variablen

Die programmierbaren Logikbausteine des EX16 und SPEEDY besitzen 44 1-Bit-Register, die nicht mit Ein– oder Ausgängen verknüpft sind. Diese kann man zum Zwischenspeichern von logischen Zuständen nutzen, zum Beispiel, wenn an einem Eingang nur ein kurzer Impuls anliegt.

Interne Variablen werden im Deklarationsteil "VAR_INTERN" definiert und erhalten nur einen Namen, keinen Ort. Sie verhalten sich wie Ausgangsvariablen, können also asynchron (Ausgang eines logischen Gatters) oder mit der .CLK-Erweiterung als Register definiert werden.

Im obigen Beispiel kann man eine interne Variable benutzen, um die Zeit zwischen dem Anliegen des Eingangssignals "E" bis zum Schalten des Ausgangs "A" zu überbrücken, so dass nur ein kurzer Impuls am Eingang "E" notwendig ist. Da eine Selbsthaltung programmiert werden soll, muss auch noch die Eingangsvariable "POR" (Power-On-Reset) deklariert werden.

```
VAR_INPUT
     ...
     POR AT POR;
END_VAR;

VAR_INTERN
     M;
END_VAR;

M := E                              (* Einschaltbedingung *)
OR M AND NOT A AND NOT POR;         (* Selbsthaltung mit
                                       Ausschaltbedingung *)
```

Die Zuweisungsgleichung "`T1Reset := NOT E;`" wird ersetzt durch:

```
T1Reset := NOT M;
```

9. IF-Anweisung

Die Programmiersprache "Strukturierter Text" zeichnet sich durch einige Programmkonstrukte aus, die ein Programm übersichtlicher und besser lesbar machen. Eines davon ist die IF-Anweisung, die logische Bedingungen und Anweisungen optisch voneinander trennt und eine größere Funktionalität als die bisher behandelten logischen Gleichungen besitzt. Die Form der IF-Anweisung lautet:

IF (<Bedingung>) THEN

<Zuweisung(en);>

ELSIF (<Bedingung>) THEN

<Zuweisung(en);>

ELSE

<Zuweisung(en);>

ENDIF;

Die Anweisungen "ELSIF" und "ELSE" sind optional und daher nicht fettgedruckt dargestellt.

Beispiel:

```
PROGRAM IF_Anw

VAR_INPUT
     E1 AT E1;
     E2 AT E2;
     E3 AT E3;
     E4 AT E4;
END_VAR;

VAR_OUTPUT
     A1 AT A1;
     A2 AT A2;
     A3 AT A3;
     A4 AT A4;
     A5 AT A5;
     A6 AT A6;
END_VAR;
```

```
IF (E1 AND (E2 XOR E3) OR NOT E4) THEN
     A1 := 1;
     A3 := 1;
ELSIF (NOT (E2 XOR E3)) THEN
     A2 := 1;
     A5 := 1;
ELSE
     A4 := 1;
     A6 := 1;
END_IF;

END_PROGRAM;
```

10. Arrays

EX_PRESS bietet die Möglichkeit, mehrere 1-Bit-Variablen zu einer Mehrbit-variablen (max. 8 Bit) zusammenzufassen. Dies geschieht im Deklarationsteil mit

VAR_ARRAY

<NAME> [<Var1>, <Var2, ...];

END_VAR;

Dabei müssen <Var1>, <Var2> usw. bereits vorher als Variablen deklariert worden sein. <VAR1> ist das höchstwertige Bit.

11. Vergleichsoperatoren

Mit den Vergleichsoperatoren **=**, **>**, **<**, **>=**, **<=** und **<>** ist es möglich, logische Vergleiche eines Arrays mit einem festen Wert, aber auch von 1-Bit-Variablen mit 0 oder 1 durchzuführen.

12. CASE-Anweisung

Die CASE-Anweisung schließlich bietet die Möglichkeit, abhängig vom Wert einer Variablen verschiedene Zuweisungen durchzuführen in der Form:

CASE <Variable> of

<Wert1>: <Zuweisung(en)>;

<Wert2>: <Zuweisung(en)>;

...

ELSE

<Zuweisung(en);>

END_CASE;

Beispiel zu Arrays, Vergleichsoperatoren und CASE-Anweisung:

```
PROGRAM CASE_Anw

VAR_INPUT
     E1 AT E1;
     E2 AT E2;
END_VAR;

VAR_OUTPUT
     A1 AT A1;
     A2 AT A2;
     A3 AT A3;
     A4 AT A4;
     A5 AT A5;
     A6 AT A6;
END_VAR;

VAR_ARRAY           (* Hier wird das Array deklariert *)
E [E2, E1];
END_VAR;

IF (E >= 2) THEN    (* Verwendung eines Vergleichsopera-
                       tors *)
                    A1 := 1;
END_IF;
CASE E OF           (* CASE-Anweisung *)
     0:   A2 := 1;

     1:   A3 := 1;
          A4 := 1;

     2:   A2 := 1;
          A5 := 1;

     3:   A6 := 1;
```

```
END_CASE;

END_PROGRAM;
```

13. Online-Hilfe, Beispielprogramme

Nachdem nun die gesamte Funktionalität von *EX_PRESS* vorgestellt wurde, sei an dieser Stelle darauf hingewiesen, dass die Online-Hilfe von *EX_PRESS* und die im Installationsverzeichnis befindlichen Beispielprogramme weitere detaillierte Informationen enthalten, die bei speziellen Fragen weiterhelfen.

Anhang 1 – Befehle und Schlüsselwörter von EX_PRESS

schlüsselwort	Funktion, Bedeutung	Abschnitt im Hilfesystem (Menü “Hilfe”, Funktion “Inhalt”, “Einführung in die Sprache ST-16”)
AND, &	Logische UND-Verknüpfung	Zuweisungen und logische Operatoren
CASE...OF...	Vielfachverzweigung, -fallunterscheidung	Programmkonstrukte
ELSE	Alternative bei CASE oder IF	Programmkonstrukte
ELSIF...THEN	Alternativbedingung bei IF	Programmkonstrukte
ENDIF	Ende einer IF-Abfrage	Programmkonstrukte
END_CASE	Ende einer CASE-Verzweigung	Programmkonstrukte
END_PROGRAM	Ende des Programm-Quellcodes	Allgemeines zum Strukturierten Text
END_VAR	Ende einer Variablendeklaration	Variablen- und Namenskonventionen
IF...THEN...	Verzweigung, Fallunterscheidung	Programmkonstrukte
NOT, -	Logische Negation	Zuweisungen und logische Operatoren
OR	Logische ODER-Verknüpfung	Zuweisungen und logische Operatoren
PROGRAM	Anfang des Programm-Quellcodes	Allgemeines zum Strukturierten Text
VAR_ARRAY	Deklaration von Mehrbit-Variablen	Variablen- und Namenskonventionen
VAR_INPUT...AT...	Deklaration von Eingangsvariablen	Variablen- und Namenskonventionen
VAR_INTERN	Deklarationen von Merkern und Flags	Variablen- und Namenskonventionen
VAR_OUTPUT...AT...	Deklaration von Ausgangsvariablen	Variablen- und Namenskonventionen
XOR	Logische Antivalenz-Verknüpfung	Zuweisungen und logische Operatoren
.CLK	Definition einer Registervariablen	Eigenschaften von Variablen
.CRIT	Definition eines zeitkritischen Signals	Eigenschaften von Variablen
.OE	Deaktivierung einer Ausgangsvariablen	Eigenschaften von Variablen
.RE	Rücksetzeingang für Registervariablen	Eigenschaften von Variablen
:=	Zuweisungsoperator	Zuweisungen und logische Operatoren
=	Vergleichsoperator: gleich	Zuweisungen und logische Operatoren
<>	Vergleichsoperator: verschieden	Zuweisungen und logische Operatoren
<	Vergleichsoperator: kleiner als	Zuweisungen und logische Operatoren
<=	Vergleichsoperator: kleiner oder gleich	Zuweisungen und logische Operatoren
>	Vergleichsoperator: größer als	Zuweisungen und logische Operatoren
>=	Vergleichsoperator: größer oder gleich	Zuweisungen und logische Operatoren
(...)	Änderung der Auswertungspriorisierung	Zuweisungen und logische Operatoren
(*...*)	Kommentar	Allgemeines zum Strukturierten Text

Anhang 2 – Programmkapazität des CPLD

Der programmierbare Logikbaustein des EX16 bzw. SPEEDY besitzt insgesamt 96 Register, von denen 36 den Ein- und Ausgängen direkt zugeordnet sind. Von den übrigen 60 Registern wird pro definiertem Ausgang jeweils eines benötigt, die verbleibenden können in Programmen als Merker, Flags, Zählvariablen usw. verwendet werden.

Innerhalb des CPLD stehen etwa 300 Oder- und 5000 Und-Gatter zur Verfügung. Dabei kann jedes Signal bei Bedarf invertiert werden. Diese Gatter sind in Form von Logikblöcken angeordnet und können über eine Verknüpfungsmatrix miteinander verbunden werden.

In der Praxis hat sich gezeigt, dass in der Regel nicht die Anzahl der verfügbaren Logikverknüpfungen, sondern die Anzahl der Ein- und Ausgänge pro Logikblock die Programmkapazitätsgrenzen des CPLD bestimmen, wobei die Fitter-Software sowohl die Zuordnung der Logikfunktionen zu den Logikblöcken als auch die Verdrahtung der Blöcke untereinander bzw. mit den Ein- und Ausgängen übernimmt und so weit wie möglich optimiert.

Um die benötigten Ressourcen innerhalb des CPLD für ein bestimmtes Programm abschätzen zu können, sei darauf hingewiesen, dass neben unmittelbar programmierten Logik- und Registerfunktionen weitere als Verknüpfungshilfe benötigt werden können; es tritt also ein gewisser "Verschnitt" auf. Dabei gilt: Je komplexer die Gleichungen sind, desto höher ist der Verschnitt; als Durchschnittswert kann man etwa 20% annehmen.

Anhang 3 – Häufig gestellte Fragen

Wie groß ist der Programmspeicher des EX16/Speedy?

Man kann die Speicherkapazität nicht wie bei Mikrocontrollersteuerungen in Kilobyte angeben, weil das Programm ja in festverdrahtete Logik umgewandelt wird. Daher kann man nur Aussagen über die vorhandene Anzahl logischer Gatter und der Anzahl der verfügbaren elektrischen Leitungen innerhalb des programmierbaren Logikbausteins machen. Siehe auch *Anhang 2– Programmkapazität des CPLD* .

Wie oft kann ich EX16/Speedy neu programmieren?

Der Hersteller des programmierbaren Logikbaustein garantiert mindestens 1000 Schreibzyklen.

Welches Betriebssystem benötige ich für EX_PRESS?

EX_PRESS läuft unter Windows XP, 7, 8.1, 10 (32 oder 64 Bit)

Gibt es per Software programmierbar Timer?

Der Logikbaustein besitzt keine internen Timer, so dass als Basistakt eines der Timersignale T1 bis T4 benutzt werden muss. SPEEDY stellt bei Linksanschlag der Potentiometer eine kalibrierte Zeitbasis von 100ms zur Verfügung. Durch Programmierung eines Zählers kann dieser Takt untersetzt werden, so dass mit einem Taktsignal verschiedene Zeiten generiert werden können. Ein Beispiel für die Programmierung eines Zählers findet sich in der Online-Hilfe unter dem Punkt "Einführung in die Sprache ST-16", Unterpunkt "Ein komplettes Beispiel", wo mit den Variablen Z1 bis Z4 ein 4-Bit-Zähler programmiert wurde.

Wie viele Merker gibt es?

Es stehen mindestens 44 1-Bit-Register zur Verfügung. Siehe auch *Anhang 2– Programmkapazität des CPLD* .

Anhang 4: EX_PRESS Troubleshooting für EX_PRESS ab Version 4.0

1. Lange Dateinamen

Pfadnamen dürfen seit EX_PRESS V4.0 praktisch beliebig lang sein, so dass auch Unterverzeichnisse im Ordner „Eigene Dateien“ angelegt werden können. Beim Namen der Quelldateien (*.s16 bzw. *.fup) muss aber weiterhin das Format „8.3“ verwendet werden, also 8 Zeichen für den Dateinamen vor dem Punkt und 3 Zeichen für die Erweiterung „s16“ bzw. „fup“ (z.B. Test1234.s16) . Der Fitter verlangt dieses Format, und obwohl EX_PRESS versucht, lange Dateinamen entsprechend umzuwandeln, hat die Praxis gezeigt, dass dies unter bestimmten Umständen nicht gelingt und zu Fehlfunktionen führen kann.

2. Fitter startet nicht (unter Windows XP)

Wenn sichergestellt ist, dass die Installation des EX_PRESS-Systems ohne Fehlermeldung abgeschlossen wurde, aber der Fitter nicht aufgerufen wird (es müsste sich ein Fenster mit weißem Hintergrund und schwarzer Schrift öffnen), bitte folgenden Sachverhalt überprüfen:

Im Startmenü unter „Einstellungen“ die Systemsteuerung öffnen, dann Doppelklick auf „Regions– und Sprachoptionen“. Die Registerkarte Sprachen anwählen und überprüfen, ob der Punkt „Dateien für ostasiatische Sprachen installieren“ aktiviert ist. Falls ja, diesen bitte deaktivieren und nach Aufforderung den Computer neu starten.

3. USB-Kabel wird beim Einstecken nicht erkannt (unter Windows XP)

Es kann passieren, dass beim Setup von EX_PRESS der Treiber für das USB-Kabel nicht richtig instlaliert wird. Bitte führen Sie das Programm FTDI_USB.exe im Verzeichnis USB Ihres Installationspfades aus.

4. Problem mit POR_Signal

Damit das POR_Signal funktioniert muss nach dem Download des Programmes der Programmierstecker von der SPS abgezogen werden.

So einfach ist EX_PRESS 4

- Safety
- Automation
- Engineering

Zander GmbH & Co. KG
Am Gut Wolf 15
52070 Aachen, Deutschland
info@zander-aachen.de
www.zander-aachen.de