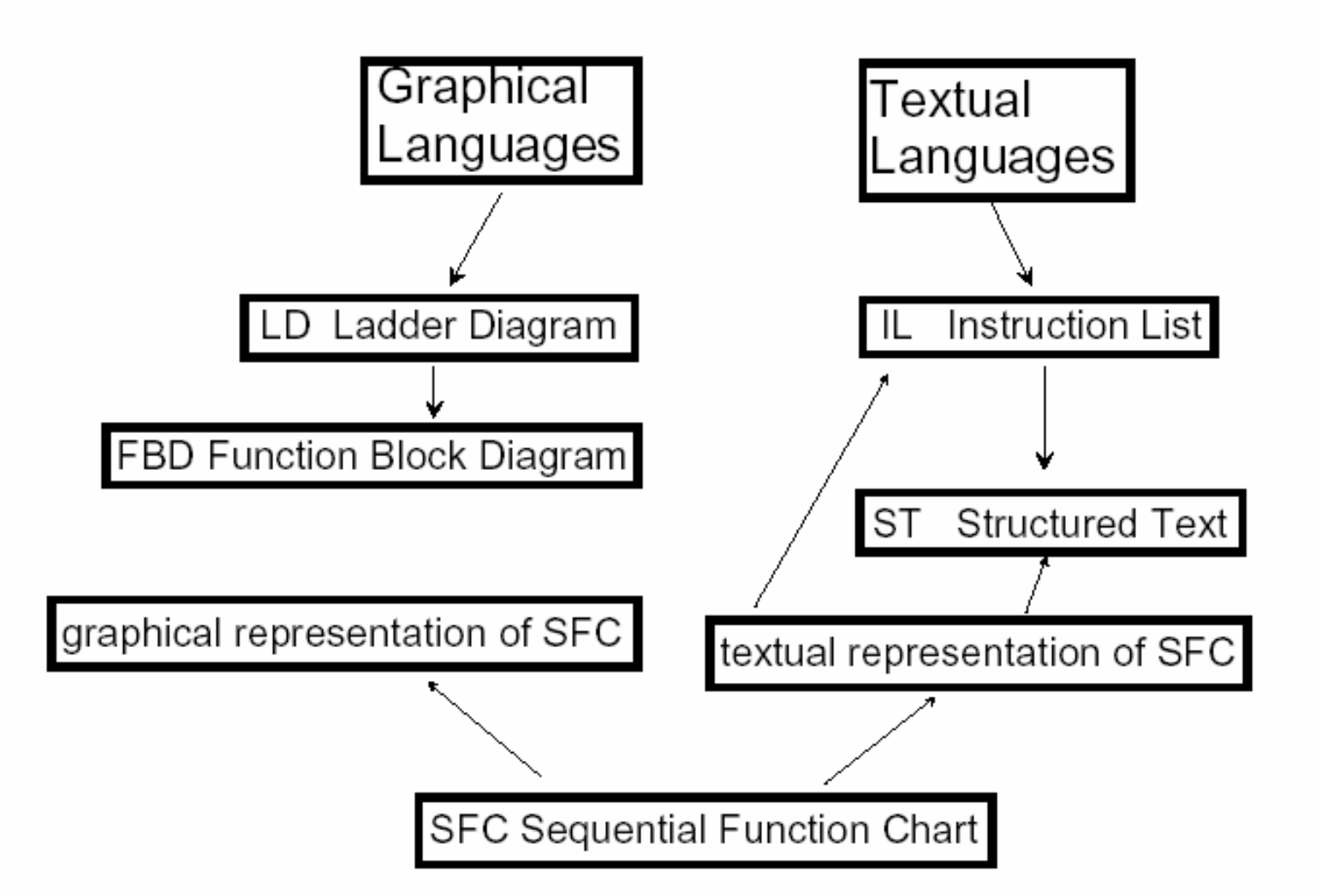
Kapitel 15 Grundlagen zur SPS

Übersicht über die Programmiersprachen der IEC 61131-3



Ladder Diagram (LD)

Bei dieser Programmiersprache handelt es sich um die im deutschen Bereich als Kontaktplan (KOP) bekannte Programiermöglichkeit. Hierbei wird die Steuerungsaufgabe als Stromlaufplan einer Schütz- oder Relaisschaltung dargestellt, deren Verknüpfungen (Oder-Verknüpfung als Parallelschaltung von Kontakten; Und-Verknüpfung als Reihenschaltung von Kontakten) dann von der Zielmaschine zyklisch abgearbeitet werden.

Function Block Diagram (FBD)

Hierbei werden Funktionsbausteine signalmäßig miteinander verschaltet. Diese Möglichkeit wurde je nach Hersteller als Funktionsplan (FUP) oder besser als Logikplan (LOP) bezeichnet. Diese Funktionsbausteine entsprechen natürlich Instanzen von Klassen, die entweder vom Hersteller als Standardfunktionsbausteine mitgeliefert werden oder aber als spezielle Bausteine von Benutzer selbst programmiert werden können.

Instruction List (IL)

Diese Programmiersprache entspricht der alten Anweisungsliste (AWL). Allerdings sind - wie schon erwähnt - die syntaktischen Spielregeln hier endlich festgelegt worden. Bei der Instruction List wird im ersten Teil eine Verknüpfung gebildet, die im nächsten Teil einem oder mehreren Operanden zugewiesen wird. Bei vielen insbesondere älteren SPS war der Maschinencode so definiert, dass er 'eins-zu-eins' aus der AWL übersetzt werden konnte. Man hat deshalb diese Programmiersprache deshalb auch als sehr maschinennah bezeichnet.

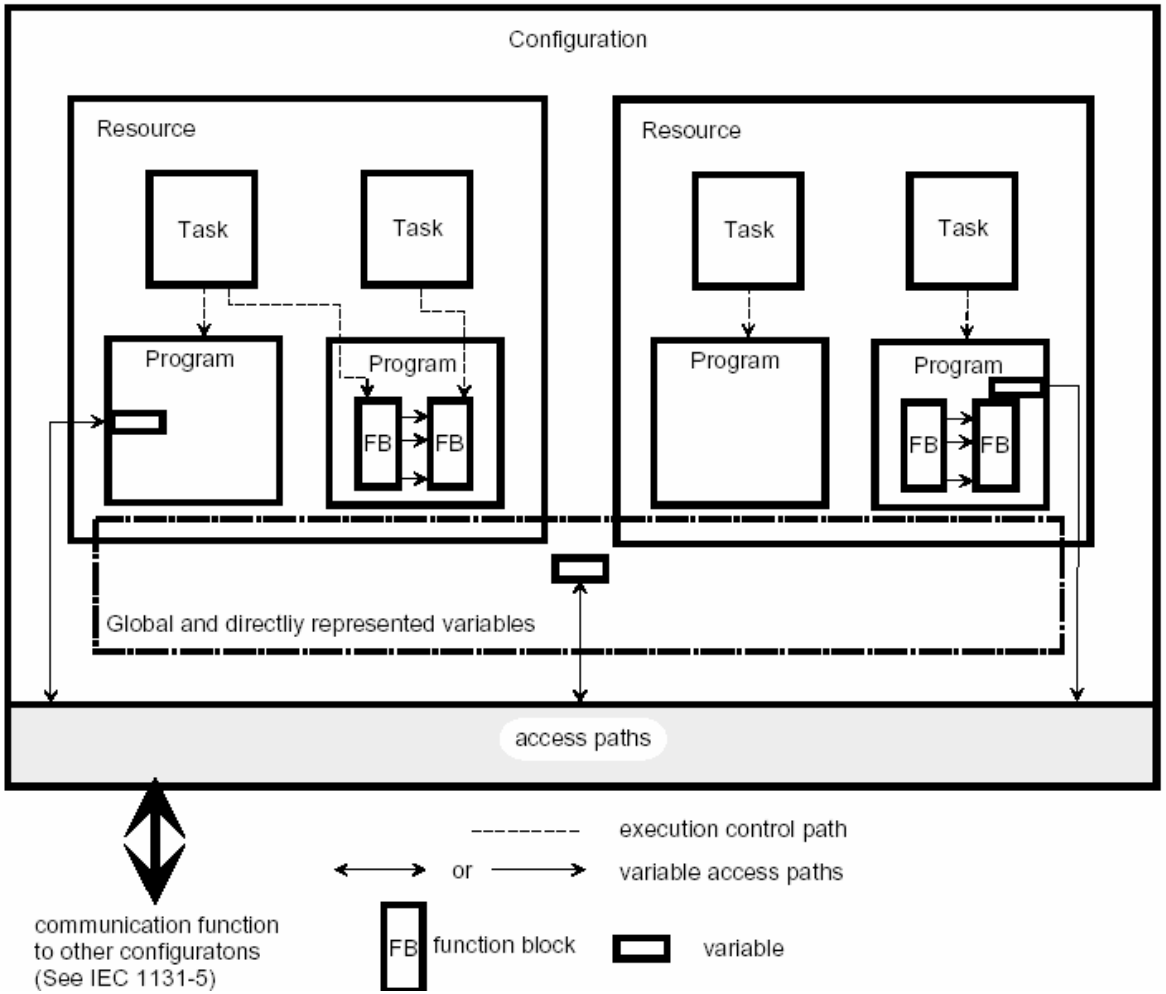
Sequential Function Chart (SFC)

Mit dieser Sprache lassen sich steuerungstechnische Abläufe sehr gut formulieren. Sie geht von der Darstellungsphilosophie der Petrinetze aus. Teile der Ablaufsprachen sind in dem alten Funktionsplan (FUP) bei einigen Herstellern bereits realisiert gewesen.

Structured Text (ST)

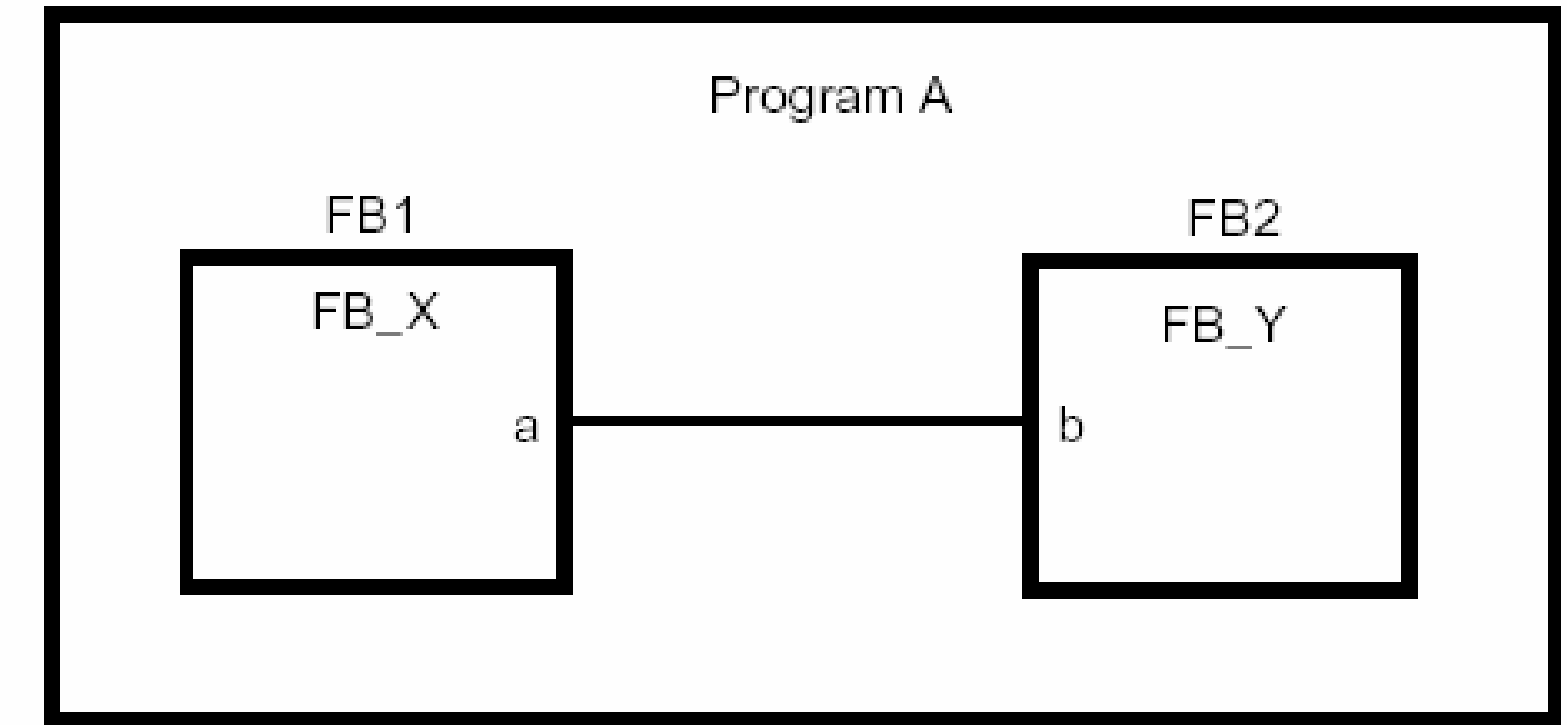
Structured Text ist eine - sehr pascal-ähnliche - Hochsprache, mit der sich viele unterschiedliche Dinge realisieren lassen. Für regelungstechnische Anwendungen, bei denen naturgemäß arithmetische Berechnungen notwendig sind, ist diese Sprache vorzusehen.

Aufbau eines komplexen SPS-Systems



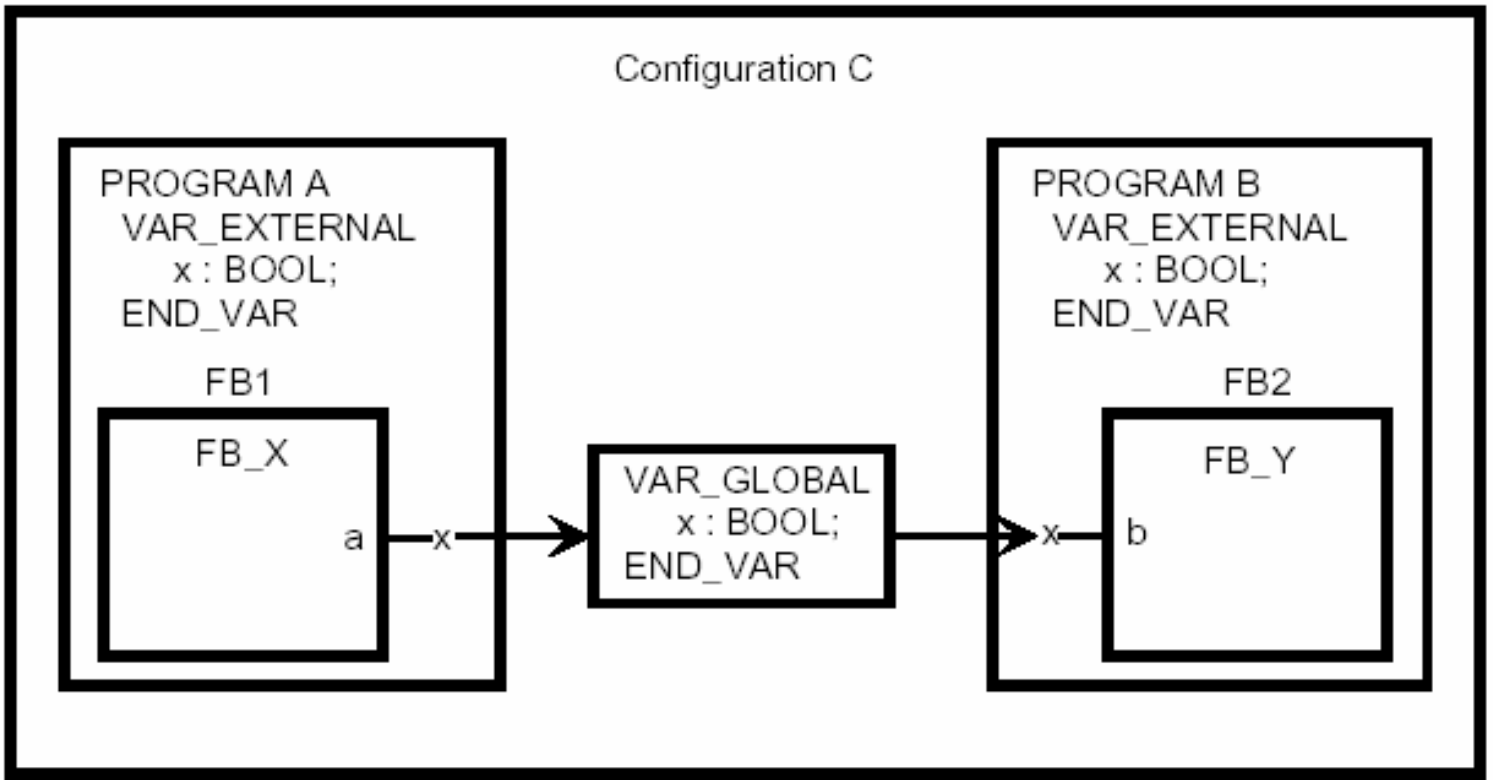
Zunächst einmal muss festgestellt werden, dass nicht jede Implementierung alle diese Möglichkeiten umfassen muss. Subsets sind durchaus möglich. So sind z.B. in der IEC1131-3 als eine Form der Kapselung Programme vorgesehen, obwohl diese nicht unbedingt erforderlich sind. Eine Resource entspricht i.A. einer SPS oder einem Rechner. Eine Configuration könnte einem Rechnerverbund (z.B. SPS mit mehreren Zentraleinheiten) entsprechen. Es ist nun möglich, sowohl einzelne Funktionsbausteine als auch - wenn implementiert - Programme komplett mit den darin eingeschlossenen Funktionsbausteinen an eine Task zur Abarbeitung anzubinden.

Kommunikation von Funktionsbausteinen



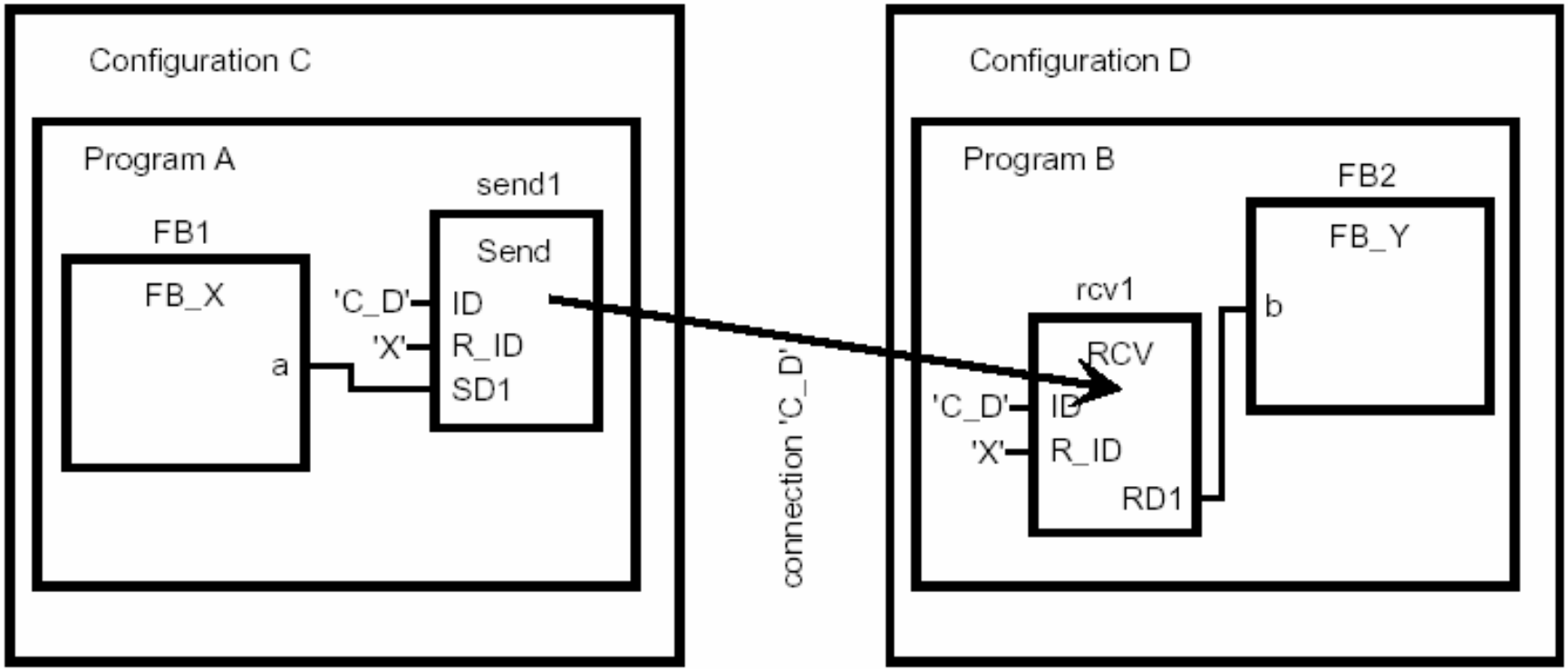
Der Funktionsbaustein FB1 ist eine Instanz von FB_X, FB2 eine Instanz von FB_Y. Beide Instanzen sind im Programm A abgelegt. Der Ausgang a der Instanz FB1 kommuniziert mit dem Eingang b der Instanz FB2.

Kommunikation von Funktionsbausteinen in verschiedenen Programmen einer Konfiguration über globale Variablen



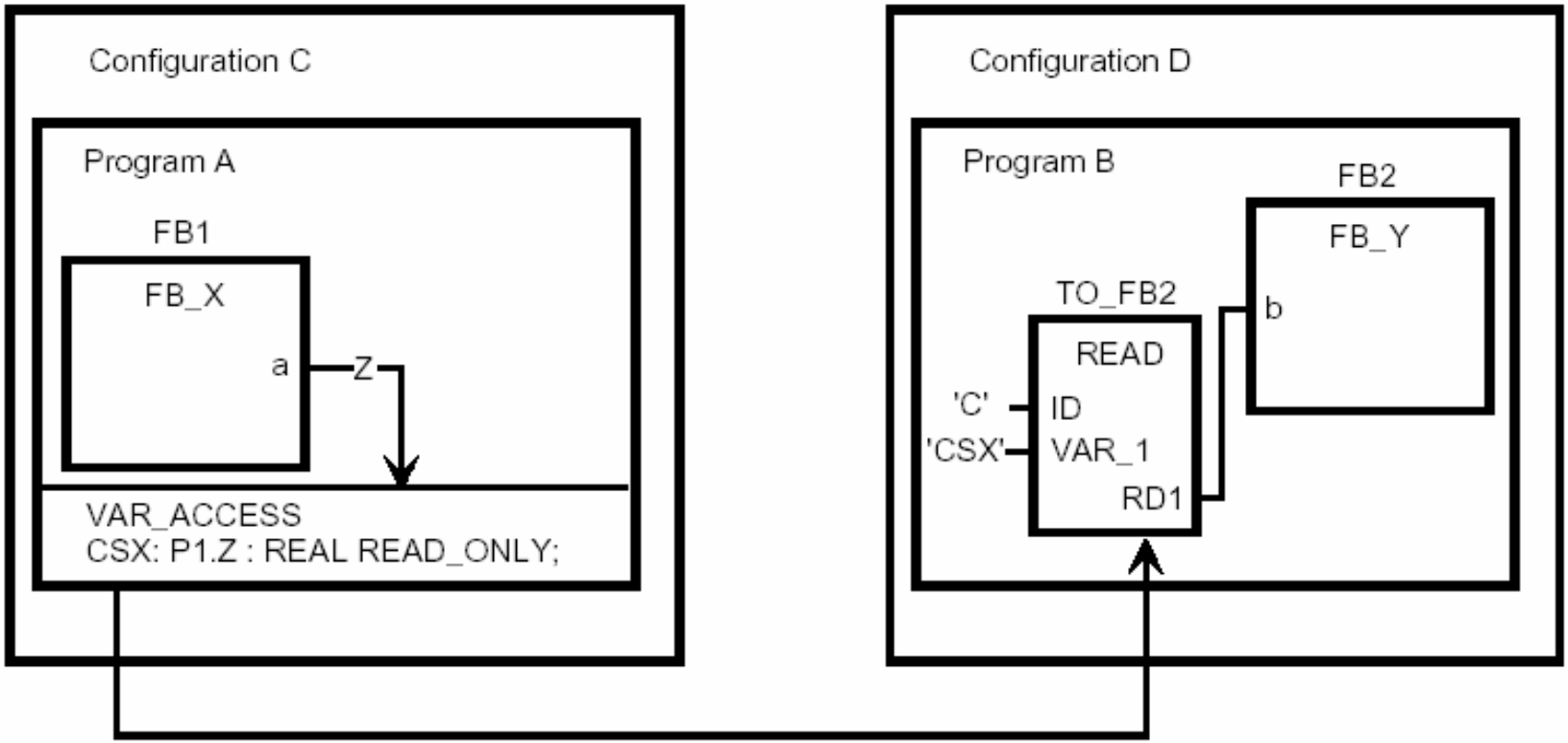
Die Instanzen FB1 und FB2 liegen in unterschiedlichen Programmen A und B. Die Kommunikation zwischen den Ausgängen a und b geschieht über eine global angelegte Variable x.

Kommunikation von Funktionsbausteinen in verschiedenen Programmen und Konfigurationen über ein Interface (I)



Die Instanzen FB1 und FB2 liegen in unterschiedlichen Programmen in den Konfigurationen C und D. Die Kommunikation zwischen den Ausgängen a und b geschieht über ein Interface.

Kommunikation von Funktionsbausteinen in verschiedenen Programmen und Konfigurationen über ein Interface (II)



Die Instanzen FB1 und FB2 liegen in unterschiedlichen Programmen in den Konfigurationen C und D. Die Kommunikation zwischen den Ausgängen a und b geschieht über ein Interface.

Klassifikation der Elemente der IEC 61131-3

- Data types
- Program organization units
  - Functions
  - Function Blocks
  - Programs
- Sequential Function Chart Elements
- Configuration Elements
  - Global variables
  - Resources
  - Tasks
  - Access paths

Funktionen geben einen oder mehrere Variablen als Return-Value zurück. Funktionen können in einer textuellen Programmiersprache innerhalb eines arithmetischen oder logischen Ausdrucks aufgerufen werden.
Funktionen sollten auf keine gespeicherten Variablen zurückgreifen, sondern ausschließlich auf übergebene Variablen.

No.	Keyword	Data type	Bits	Range
1	BOOL	Boolean	1	
2	SINT	Short integer	8	2)
3	INT	Integer	16	2)
4	DINT	Double integer	32	2)
5	LINT	Long integer	64	2)
6	USINT	Unsigned short integer	8	3)
7	UINT	Unsigned integer	16	3)
8	UDINT	Unsigned double integer	32	3)
9	ULINT	Unsigned long integer	64	3)
10	REAL	Real numbers	32	4)
11	LREAL	Long reals	64	5)
12	TIME	Duration	1)	6)
13	DATE	Date (only)	1)	6)
14	TIME_OF_DAY or TOD	Time of day (only)	1)	6)
15	DATE_AND_TIME or DT	Date and time of Day	1)	6)
16	STRING	Variable-length character string	1)	7)
17	BYTE	Bit string of length 8	8	7)
18	WORD	Bit string of length 16	16	7)
19	DWORD	Bit string of length 32	32	7)
20	LWORD	Bit string of length 64	64	7)

Bemerkungen:

1) Die Größe dieser Datentypen ist implementierungsabhängig

2) Der Wertebereich dieser Datentypen ist von -2^{Bits-1} bis $2^{Bits-1}-1$

3) Der Wertebereich dieser Datentypen ist von 0 bis $2^{Bits}-1$

4) Der Wertebereich dieser Variablen wird in der IEC 559 für den Floatingpointtyp single definiert.

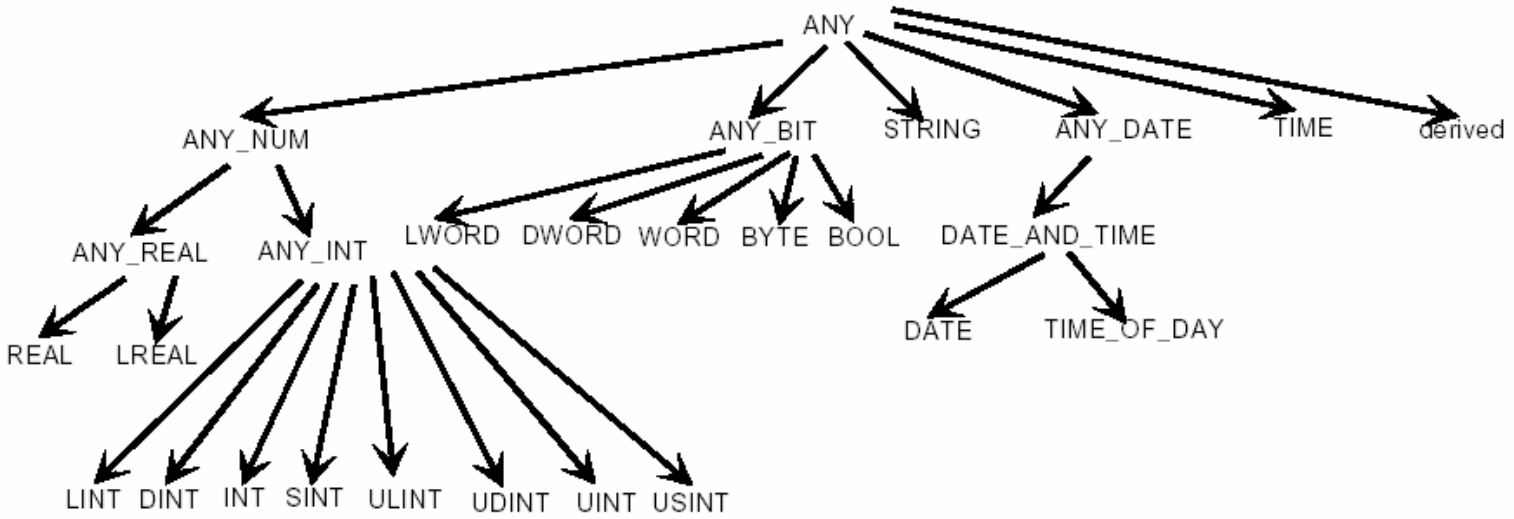
5) Der Wertebereich dieser Variablen wird in der IEC 559 für den Floatingpointtyp double definiert.

6) Der Werte von Variablen dieses Datentyps ist implementierungsabhängig

7) Für diese Datentypen gibt es keinen Wertebereich

8) Mögliche Werte für diesen Datentyp sind 0 oder 1, die auch mit den Keywords FALSE bzw. TRUE korrespondieren

Übersicht über Datentypen der IEC 61131-3



Die Vorteil dieser generischen Datentypen liegt z.B. in der Möglichkeit, es offen zu lassen, welche Variable an einen Funktions-Baustein angeschlossen wird. Voraussetzung ist natürlich ein Mechanismus, der es möglich macht, zur Laufzeit den aktuell angeschlossenen Datentyp zu erkennen.

Derived Data Types

Die IEC1131-3 bietet auch die Möglichkeit als Benutzer eigene Datentypen zu spezifizieren. Solche Typdefinitionen werden durch das Keyword

```
TYPE
```

eingeleitet und durch

```
END_TYPE
```

beendet. Dabei ergeben folgende Möglichkeiten:

Direct deriviation from elementary types

Hierbei wird einfach der neue Datentyp direkt von einen Grunddatentyp abgeleitet.

Beispiel:

```
TYPE
  myRealType : REAL;
END_TYPE
```

Enumerated Data Types

Hierbei kann eine Variable nur die Werte annehmen, die in einer Liste in der Definition vorgegeben sind.

Beispiel:

```
TYPE
  Analog_Signal_Type : (Single_Ended, Differential);
END_TYPE
```

Subrange data types

Hierbei kann der Wertebereich eines Grunddatentyps eingeschränkt werden.

Beispiel:

```
TYPE
  Analog_Data : INT (-4095..4095);
END_TYPE
```

Array data types

Hiermit wird ein Array bestehend aus Elementen des Gunddatentyps definiert.

Beispiel:

```
TYPE
  Analog_16_Input_Data : ARRAY[1..16] OF Analog_Data;
END_TYPE
```

Structured data types

Hiermit können - wie bei anderen höheren Programmiersprachen auch - Grunddatentypen oder auch bereits strukturierte Datentypen in einer neuen Struktur zusammengefasst werden.

Beispiel:

```
TYPE
  Analog_Channel_Configuration : STRUCT
        Range : Analog_Signal_Range;
        Min_Scale : Analog_Data;
        Max_Scale : Analog_Data;
  END_STRUCT
  Analog_16_Input_Configuration : STRUCT
        Signal_Type : Analog_Signal_Type;
        Filter_Parameter : SINT(0..99);
        Channel : ARRAY[1..16] OF Analog_Channel_Configuration;
  END_STRUCT
END_TYPE
```

Initialisation

Wenn nicht anders spezifiziert, werden alle Variablen mit 'Null' vorbelegt. Ein STRING ist dabei leer. Es kann jedoch bereits bei der Typdeklaration ein Initialisierungswert spezifiziert werden, mit dem die Variable bei Instanzierung vorbelegt wird. Dieser Initialisierungswert folgt der Angabe des Datentyps nach einem ':=' .

Beispiel:

```
TYPE
   Str10 : STRING(10) := 'ABCDEF';
   PI : LREAL := 3.1415925;
   Analog_16_Input_DataI : ARRAY[1..16] OF Analog_Data := 8(-
4095), 8(4095);
   Analog_Channel_ConfigZ : Analog_Channel_Configuration(Min_Scale
:= 0,
                                MaxScale := 9999);
END_TYPE
```

No.	Beschreibung der Eigenschaft	Beispiele
1	Integer literals	-12 0 123_456 +986
2	Real literals	-13.0 0.0 0.4567 3.14159_26
3	Real literals with exponents	-1.34E-12 oder -1.34e-12 1.0E+6 oder 1.0e+6 1.234E6 or 1.234e6
4	Base 2 literals	2#1111_1111 (255 dezimal) 2#1110_0000 (240 dezimal)
5	Base 8 literals	8#377 (255 dezimal) 8#340 (240 dezimal)
6	Base 16 literals	16#FF oder 16#ff (255 dezimal) 16#E0 oder 16#e0 (240) dezimal
7	Boolean zero and one	0 1
8	Boolean FALSE and TRUE	FALSE TRUE
Bemerkung: Die Keywords FALSE und TRUE korrespondieren jeweils zu den bool'schen Werten 0 und 1		

Beispiel	Erklärung
''	Leerer String (Länge null)
'A'	String der Länge eins. Enthält den Character A
' '	String der Länge eins. Enthält den 'Space'-Character
'$''	String der Länge eins. "Enthält den Single Quote Character"
'RL' '$0D$0A'	String der Länge zwei. Enthält CR und LF Characters
'$$1.00'	String der Länge fünf. Enthält $1.00

No.	Beschreibung der Eigenschaft	Prefix Keyword
1	Date literals (long prefix)	DATE#
2	Date literals (short prefix)	D#
3	Time of day literals (long prefix)	TIME_OF_DAY#
4	Time of day literals (long prefix)	TOD#
5	Date and time literals (long prefix)	DATE_AND_TIME#
6	Date and time literals (short prefix)	DT#

Long prefix notation	Short prefix notation
DATE#1999-12-21 date#1999-12-21	D#1999-12-21 d#1999-12-21
TIME_OF_DAY#13:24:55:22 time_of_day#13:24:55:22	TOD#13:24:55:22 tod#13:24:55:22
DATE_AND_TIME#1999-12-21-13:24:55:22 date_and_time#1999-12-21-13:24:55:22	DT#1999-12-21-13:24:55:22 dt#1999-12-21-13:24:55:22

Die Angabe der Zeitdauer (Duration) wird wahlweise durch die Keywords

```
T#
TIME#
t#
time#
```

eingeleitet. Nachfolgend die Möglichkeiten der Darstellung an Hand von Beispielen.

No.	Beschreibung der Eigenschaft	Beispiele
1a	Duration Literals without underlines: short prefix	T#14ms T#14.7s T#14.7m T#14.7h t#14.7d t#25h15m t#5d14h12m18s2.5ms
1b	Duration Literals without underlines: long prefix	TIME#14ms time#14.7s
2a	Duration Literals with underlines: short prefix	T#14ms T#14.7s T#14.7m T#14.7h t#14.7d t#25h_15m t#5d_14h_12m_18s_2.5ms
2b	Duration Literals with underlines: long prefix	TIME#25h_15m time#5d_14h_12m_18s_2.5ms

Variables

Darstellung

Single-element variables

Eine single-element variable besteht aus keinem Array oder Struct sondern besteht nur aus einem elementary type oder einem davon direkt abgeleiteten Datentyp. Interessant sind in diesem Zusammenhang die direkt dargestellten (directly represented) Datentypen, wie sie traditionell bei den älteren SPS-Sprachem immer schon Verwendung fanden. Sie werden mit einem %-Zeichen eingeleitet.

No.	Prefix	Bedeutung
1	I	Input location
2	Q	Output location
3	M	Memory location
4	X	Single bit size
5	None	Single bit size
6	B	Byte (8 bits) size
7	W	Word (16 bits) size
8	D	Doube word (32 bits) size
9	L	Long (quad) word (64 bit) size
Bemerkung : Wenn nicht anders deklariert, ist der Datentyp einer direkt adressierten 'single ended' - Variablen immer BOOL		

%QX75 %Q75	Output bit 75
%IW215	Input word location 215
%QB7	Output byte location 7
%MD48	Double word at memory location 48
%IW2.5.7.1	Hierarchical address

Multi-element variables

Multielement-Variablen sind Arrays und Structs. Hier ist eigentlich kein großer Unterschied zu den Programmiersprachen PASCAL und C festzustellen. Der indizierte Zugriff auf Array-Elemente erfolgt über eckige Klammern und die Bezeichner der hierarchischen Struktur eines Structs wird durch Punkte von einander getrennt.

Beispiel:

```
Module_5_Config.Signal_Type := Single_Ended;
Module_5_Config.Channel[5].Range := Bipolar_10V;
```

Deklaration

Vor der Variablenliste muss mit Hilfe eines entsprechenden Keywords spezifiziert werden, wie die Variable vom System behandelt werden soll. Dabei ergeben sich folgende Möglichkeiten:

Keyword	Gebrauch der Variablen
VAR	Intern innerhalb des Organization Unit
VAR_INPUT	Extern befriedigt, kann nicht vom Organization Unit aus verändert werden
VAR_OUTPUT	Wird vom Organization Unit für externen Zugriff zur Verfügung gestellt
VAR_IN_OUT	Wird extern zur Verfügung gestellt, kann vom Organization Unit aus verändert werden
VAR_EXTERNAL	Eine Variable, die durch eine Configuration via VAR_GLOBAL zur Verfügung gestellt wird, sie kann vom Organization Unit aus verändert werden
VAR_GLOBAL	Globale Variablendeklaration
VAR_ACCESS	Deklaration eines Access Paths
RETAIN	Aufbewahrende (retentive) Variable, die auch bei Stromausfall ihren Wert speichert
CONSTANT	Konstante (kann nicht verändert werden)
AT	Location assignement (Zuweisung auf einen bestimmte Adresse)

Variablendeklaration als Input- und Outputvariablen

```
VAR_INPUT
   W, Y : LREAL;
END_VAR
VAR_OUTPUT
   U : LREAL;
END_VAR
```

No.	Eigenschaft / Beispiele	
1	Declaration of directly represented, non-retentive variables	

	`VAR`
	`  AT %IW6.2 : WORD;`
	`  AT %MW6 : INT;`
	`END_VAR`
		16-bit String (Bemerkung 2)
		16-bit Integer Anfangswert = 0

2	Declaration of directly represented, retentive variables	

	`VAR RETAIN`
	`  AT %QW5 : WORD;`
	`END_VAR`
		Beim Cold-Restart wird %QW5 mit einem 16-bit String des Wertes 0 initialisiert

3	Declaration of locations of symbolic variables	

	`VAR_GLOBAL`
	`  LIM_SW_S5 : BOOL AT %IX27;`
	`  CONV_START : BOOL AT %QX25;`
	`  Temperature AT %IW28 : INT;`
	`END_VAR`
		Weist das Input-Bit der boolschen Variable LIM_SW_5 zu
		Weist das Output-Bit der boolschen Variable CONV_START zu
		Weist das Input-Word der Integer-Variable Temperature zu

4	Array location assignment	

	`VAR`
	`  Inary AT %IW6 :`
	`  ARRAY[0..9] OF INT;`
	`END_VAR`
		Deklariert ein Array von 10 Integern, die beginnend bei %IW6 fortlaufend abgebildet werden. (Bemrkung 2)

5	Automatic memory allocation of symbolic variables	

	`VAR`
	`  Condition_Red : BOOL;`
	`  Ibounce : WORD;`
	`  MyDub : DWORD;`
	`  AWord, BWord, CWord : INT;`
	`  MyStr : STRING(10);`
	`END_VAR`
		Alloziiert Memory entsprechend den Größen der einzelnen Variablen. Der String hat dabei die Größe von 10 Charactern und ist anfangs ein leerer String ''.

6	Array declaration	

	`VAR`
	`  Three : ARRAY[1..5,1..10,1..8]`
	`  OF INT;`
	`END_VAR`
		Alloziiert 400 Speicherworte für ein dreidimensionales Array.

7	Retentive array declaration	

	`VAR RETAIN`
	`  RtBt : ARRAY[1..2,1..3] of INT;`
	`END_VAR`
		Deklariert ein retantive Array RtBt, das mit Cold-Start-Werten von 0 vorbelegt wird.

8	Declaration of structured variables	

	`VAR`
	`  Module_8_Config :`
	`Analog_16_Input_Configuration;`
	`END_VAR`
		Deklaration einer Variablen eines abgeleiteten Datentyps

Bemerkung 1: Die Eigenschaften 1 bis 4 können nur in PROGRAM- oder VAR_GLOBAL-Deklarationen verwandt werden
Bemerkung 2: Die Initialisierung von System-Eingängen ist implementierungsabhängig.

No.	Eigenschaft / Beispiele	
1	Initialization of directly represented, non-retentive variables	

	`VAR`
	`  AT %QX5.1 : BOOL := 1;`
	`  AT %MW6 : INT := 8;`
	`END_VAR`
		Bolscher Typ, Anfangswert 1
		Memory-Word mit dem Integer-Wert 8 initalisiert

2	Initialization of directly represented, retentive variables	

	`VAR RETAIN`
	`  AT %QW5 : WORD := 16#FF00;`
	`END_VAR`
		Beim Cold-Restart werden die 8 Most-Significant Bits von %QW5 mit 1 und die 8 Least-Significant Bits mit 0 initialisiert

3	Location and initial value assignment to symbolic variables	

	`VAR`
	`  Valve_Pos : AT %QW28 :`
	`  INT := 100;`
	`END_VAR`
		Weist dem Input-Wort 28 den Integer-Wert 100 zu.

4	Array location assignment and initialization	

	`VAR`
	`  Outary AT %QW6 :`
	`  ARRAY[0..9] OF INT :=`
	`10(1);`
	`END_VAR`
		Deklariert ein Array von 10 Integern, die beginnend bei %IW6 fortlaufend abgebildet werden. Jedes Element wird mit 1 initialisiert

5	Initialization of symbolic variables	

	`VAR`
	`  MyBit : BOOL := 1;`
	`  Okay : STRING(10) := 'OK';`
	`END_VAR`
		Alloziiert 1 Memory-Bit mit dem Anfangswert 1, Alloziiert Memory für ein Maximum von 10 Character. Nach der Initialisierung enthält der String diem beiden Character O und K

6	Array initialization	

	`VAR`
	`  Bits : ARRAY[0..7] OF BOOL`
	`  := 1,1,0,0,0,1,0,0;`
	`  TBT : ARRAY[1..2,1..3] OF INT`
	`  := 1,2,3(4),6;`
	`END_VAR`
		Alloziiert 7 Memory-Bits mit den Anfangswerten
		Bits[0] := 1, Bits[1] := 1, . . .
		Bits[6] := 0, Bits[7] := 0.
		Alloziiert ein 2 mal 3 Integer-Array mit den Anfangswerten
		TBT[1,1] := 1, TBT[1,2] := 2,
		TBT[1,3] := 4, TBT[2,1] := 4,
		TBT[2,2] := 4, TBT[2,3] := 6,

7	Retentive array declaration and initialization	

	`VAR RETAIN`
	`  RtBt : ARRAY[1..2,1..3] of INT`
	`  := 1,2,3(4);`
	`END_VAR`
		Deklariert ein retantive Array RtBt, das mit folgenden Cold-Start-Werten vorbelegt wird:
		RtBt[1,1] := 1, RtBt[1,2] := 2,
		RtBt[1,3] := 4, RtBt[2,1] := 4,
		RtBt[2,2] := 4, RtBt[2,3] := 0,

8	Initialization of structured variables	

	`VAR`
	`  Module_8_Config :`
	`  Analog_16_Input_Configuration`
	`  (Signal_Type := Differential,`
	`  Channel[5].Range`
	`  :=Bipolar_10_V,`
	`  Channel[5].Min_Scale := 0,`
	`  Channel[5].Max_Scale :=`
	`500);`
	`END_VAR`
		Deklaration und Initialisierung einer Variablen eines abgeleiteten Datentyps

9	Initialization of Constants
	VAR CONSTANT PI : REAL := 3.141592; END_VAR
Bemerkung : Die Eigenschaften 1 bis 4 können nur in PROGRAM- oder VAR_GLOBAL-Deklarationen verwandt werden.	

Function Blocks / Function Block Diagram (FBD)

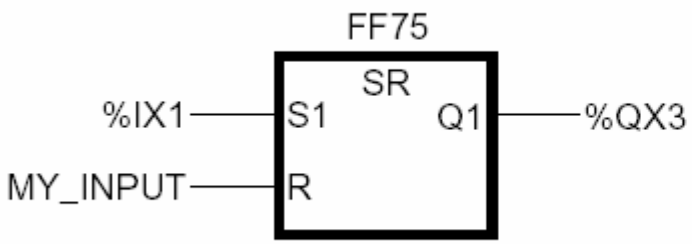
Darstellung / Instanzierung

Function Blocks stellen Programmorganisationseinheiten dar, die eine Kapselung erlauben, Ein- und Ausgänge besitzen und im Inneren Variablen aufweisen können. Natürlich liegt hier eine nahe Verwandtschaft zu den Klassen objektorientierter Strukturen vor. Wie nachfolgende Tabelle zeigt, kann ein Function Block graphisch oder in einer textuellen Programmiersprache instanziert werden. Bei der textuellen Deklaration wird das Sprachkonstrukt

```
VAR   FB_NAME : FB_TYPE;  END_VAR
```

verwendet. Bei der graphischen Darstellung gilt die Regel, dass der Klassenname (Typ des Function Block) und die Namen der Ein- und Ausgänge innerhalb des Blocks dargestellt werden. Der Name der Instanz (des Exemplars) und die aktuelle Belegung der Ein- und Ausgänge steht außerhalb.

Graphische Darstellung (FBD - Sprache)	Textuelle Darstellung (ST - Sprache)

	`VAR FF75 : SR; END_VAR (* Declaration *)`
	`FF75(S1 := %IX1, R := MY_INPUT); (* Invocation *)`
	`%QX3 := FF75.Q1; (* Assign Output *)`

Gebrauch	Innerhalb des Function Block	Außerhalb des Function Block
Input Read	`IF S1 THEN . . .`	Nicht erlaubt
Input Write	Nicht erlaubt	`FF75(S1 := %IX1, R := MY_INPUT);`
Output Read	`Q1 := Q1 AND NOT R;`	`%QX3 := FF75.Q1;`
Output Write	`Q1 := 1;`	Nicht erlaubt
Die hier nicht erlaubten Zugriffe können in Abhängigkeit von der jeweiligen Implementierung zu unerwünschten Seiteneffekten führen.		

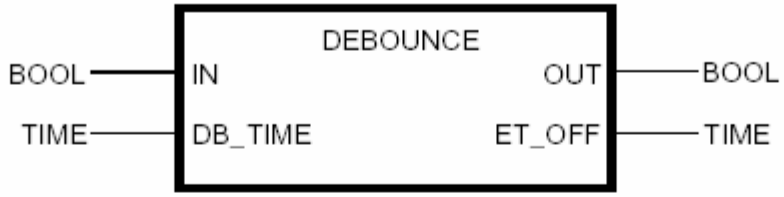
Deklarierung

Function Blocks können textuell und graphisch deklariert werden. Die IEC1131-3 sieht eine Vielzahl von Möglichkeiten vor, wie Function Blocks verschachtelt angeordnet und miteinander verschaltet werden können. Dies soll in den nachfolgenden Darstellungen von Beispielen, die direkt der Norm entnommen wurden, aufgezeigt werden.

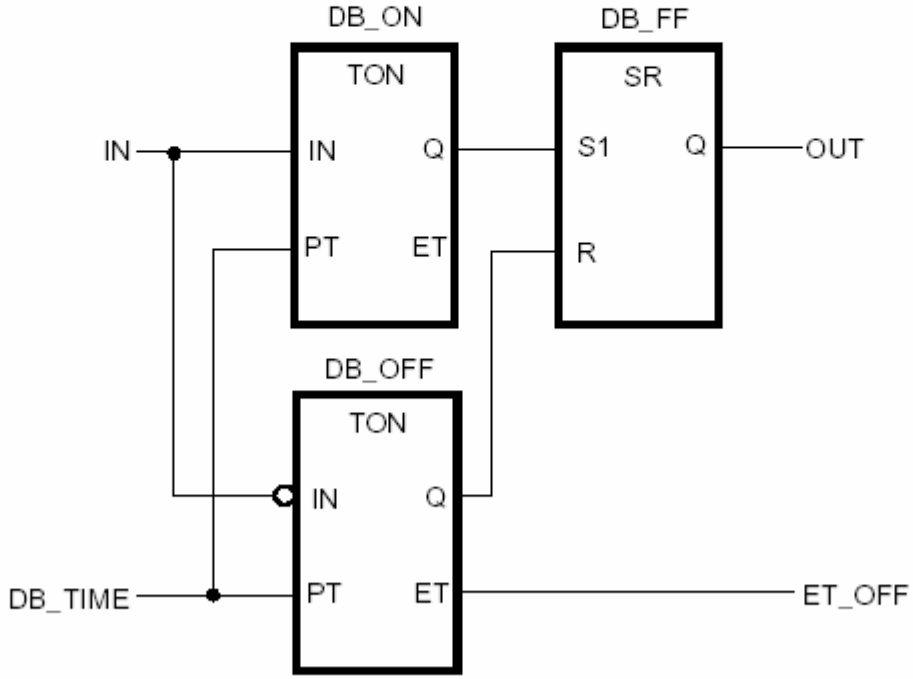
```
FUNCTION_BLOCK DEBOUNCE
(** External Interface **)

VAR INPUT
   IN       : BOOL;              (* Default = 0 *)
   DB_TIME  : TIME := t#10ms;    (* Default = t#10ms *)
END_VAR

VAR OUTPUT
   OUT      : BOOL;              (* Default = 0 *)
   ET_OFF   : TIME;              (* Default = t#0s *)
END_VAR

VAR
   DB_ON    : TON;               (* Internal Variables and Instances of Function Blocks *)
   DB_OFF   : TON;
   DB_FF    : SR;
END_VAR

(** Function Block Body **)
DB_ON(IN := IN, PT := DB_TIME);          (* DB_ON Timer Inputs *)
DB_OFF(IN := NOT IN, PT := DB_TIME);     (* DB_OFF Timer Inputs *)
DB_FF(S1 := DB_ON.Q, R := DB_OFF.Q);     (* DB_FF Flip Flop Inputs *)
OUT := DB_FF.Q;                       (* Get FF Output and Write it to Debounce Output *)
ET_OFF := DB_OFF.ET;                     (* Report Elapsed Time *)

END_FUNCTION_BLOCK
```

```
DB_ON(IN := IN, PT := DB_TIME);          (* DB_ON Timer Inputs *)
DB_OFF(IN := NOT IN, PT := DB_TIME);     (* DB_OFF Timer Inputs *)
DB_FF(S1 := DB_ON.Q, R := DB_OFF.Q);     (* DB_FF Flip Flop Inputs *)
OUT := DB_FF.Q;                       (* Get FF Output and Write it to Debounce Output *)
ET_OFF := DB_OFF.ET;                     (* Report Elapsed Time *)
```

```
FUNCTION BLOCK

(** External Interface **)
```



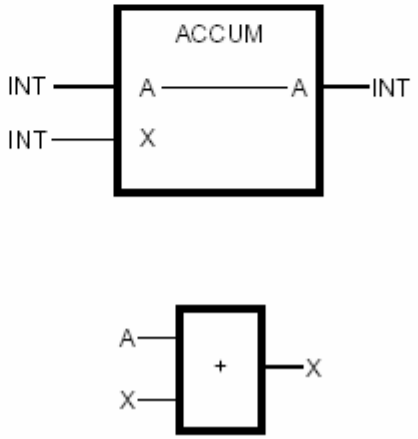

```
(** Function Block Body **)
```




```
END_FUNCTION_BLOCK
```

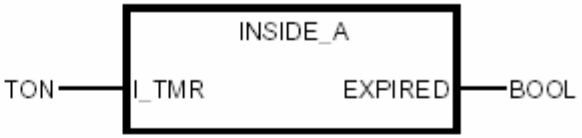
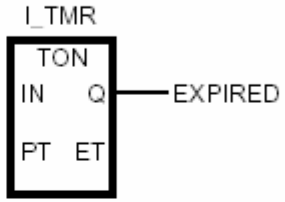
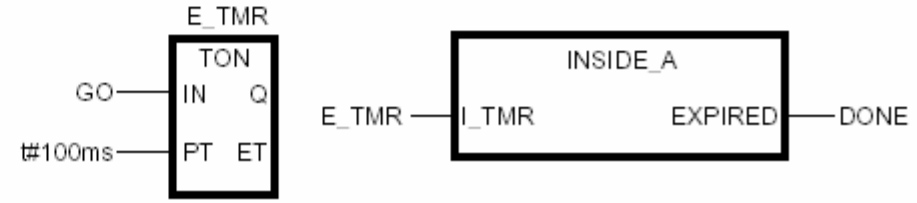
No.	Beschreibung	Beispiel
1	RETAIN Qualifier auf interne Variablen	`VAR RETAIN X : REAL; END_VAR`
2	RETAIN Qualifier auf Output-Variablen	`VAR_OUTPUT RETAIN X : REAL; END_VAR`
3	RETAIN Qualifier auf interne Function Blocks	`VAR RETAIN TMR1 : TON; END_VAR`

4a	Input/Output - Deklarieriung (textual)	`VAR_INPUT X : INT; END_VAR`
		`VAR_IN_OUT A : INT; END_VAR`
		`A := A + X;`

4b	Input/Output - Deklarierung (graphisch)	

```
FUNCTION_BLOCK

VAR_IN_OUT A : INT; END_VAR
VAR_INPUT X : INT; END_VAR

   A := A + X;

END_FUNCTION_BLOCK
```

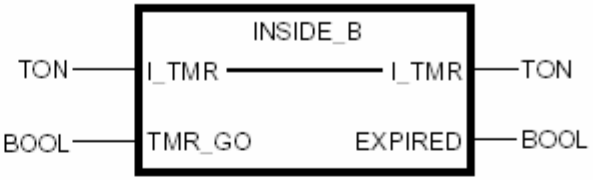
5a	Instanzname eines Function Block als Input (textual)	`VAR_INPUT I_TMR : TON; END_VAR` `EXPIRED := I_TMR.Q; (* Bemerkung 1 *)`

5b	Instanzname eines Function Block als Input (graphisch)	`FUNCTION_BLOCK`
		`(* External interface *)`
		 
		`(* Function block body *)`
		 
		`FUNCTION_BLOCK`
		`(* External interface *)`
		 
		`(* Function block body *)`
		 
		`END_FUNCTION_BLOCK`

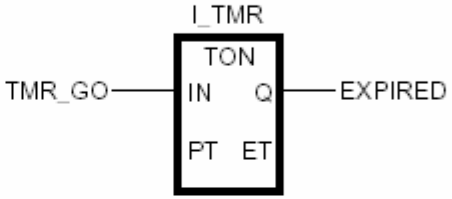
6a	Instanzname eines Function Block als Input/Output (textual)
6b	Instanzname eines Function Block als Input/Output (graphisch)

```
VAR_IN_OUT IO_TMR : TOF; END_VAR
IO_TMR(IN : A_VAR, PT := T#10s);
EXPIRED := IO_TMR.Q; (* Bemerkung 1 *)
```

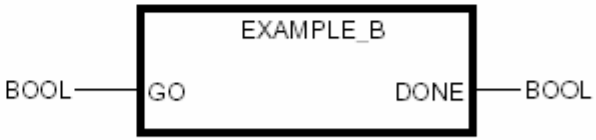
```
FUNCTION_BLOCK
(* External interface *)
```



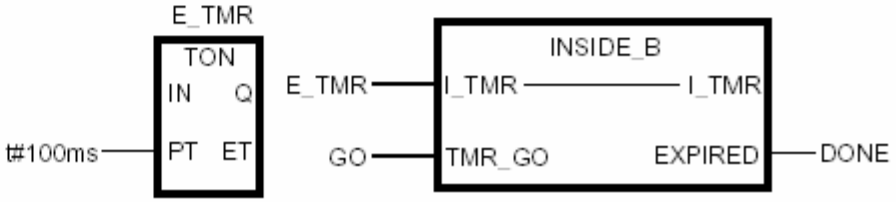

```
(* Function block body *)
```




```
FUNCTION_BLOCK
(* External interface *)
```




```
(* Function block body *)
```



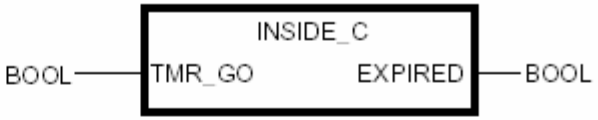

```
END_FUNCTION_BLOCK
```

7a	Instanzname eines Function Block als externe Variable (textual)	VAR_EXTERNAL EX_TMR : TOF; END_VAR EX_TMR(IN : A_VAR, PT := T#10s); EXPIRED := IO_TMR.Q; (* Bemerkung 1 *)

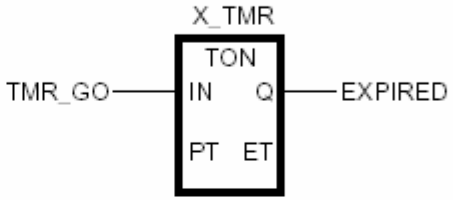
7b

Instanzname eines Function Block als externe Variable (graphisch)

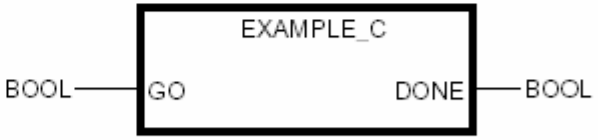
```
FUNCTION_BLOCK
(* External interface *)
```



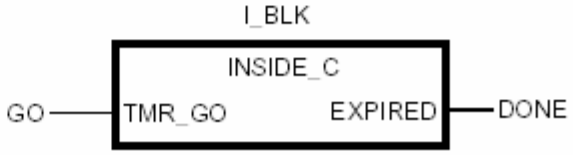

```
VAR EXTERNAL X_TMR : TON; END_VAR
(* Function block body *)
```



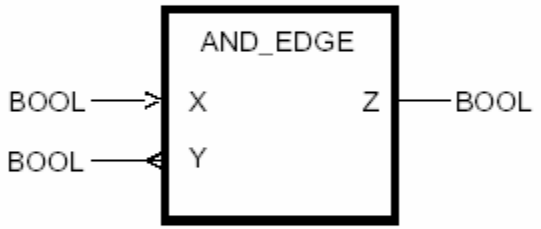
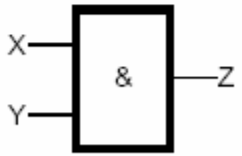

```
PROGRAM
(* External interface *)
```




```
VAR_GLOBAL X_TMR : TON; END_VAR
(* Program body *)
```




```
END_PROGRAM
```

8a 8b	Textuale Deklarierung von Rising Edge Inputs Falling Edge Inputs	FUNCTION_BLOCK AND_EDGE (* Bemerkung 2 *) VAR_INPUT X : BOOL R_EDGE; Y : BOOL F_EDGE; END_VAR VAR_OUTPUT Z : BOOL; END_VAR Z := X AND Y; END_FUNCTION_BLOCK
9a 9b	Graphische Deklarierung von Rising Edge Inputs Falling Edge Inputs	FUNCTION_BLOCK (* External interface *)  (* Function block body *)  END_FUNCTION_BLOCK (* Bemerkung 2 *)

Bemerkung 1: In diesen Beispielen wird vorausgesetzt, dass die Variablen `EXPIRED` und `A_VAR` vom Typ BOOL sind.

Bemerkung 2: Die Deklarierung des Function Block AND_EDGE im obigen Beispiel ist äquivalent zu dem folgenden Programmcode (ST) wobei die Standard Function Blocks R_TRIG und F_TRIG verwendet werden:

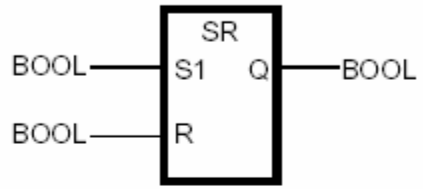
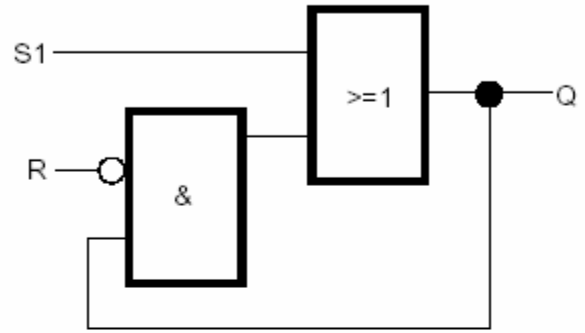
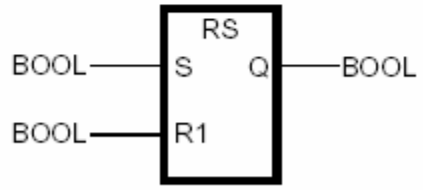
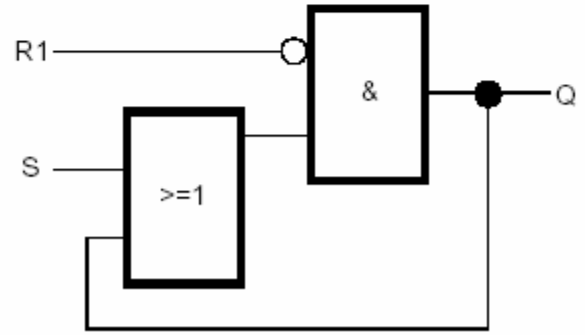
```
FUNCTION_BLOCK AND_EDGE

VAR_INPUT XCLK : BOOL; YCLK : BOOL; END_VAR
VAR_OUTPUT Z : BOOL;
VAR       X_TRIG : R_TRIG; Y_TRIG : F_TRIG; END_VAR

   X_TRIG(CLK:=XCLK);
   X := X_TRIG.Q;
   Y_TRIG(CLK:=YCLK);
   Y := Y_TRIG.Q;
   Z := X AND Y;

END_FUNCTION_BLOCK
```

Standard - Function Blocks

In einer IEC1131-3 - Implementierung sind gewöhnlich eine Reihe von Funktion Blocks mit enthalten. Die Norm gibt eine Reihe von Standard - Function Blocks vor und definiert, wie sie funktionieren müssen.

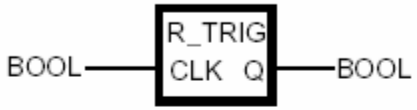
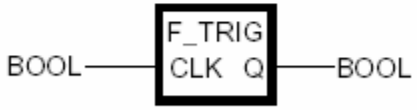
Flip-Flops

Bei RS-Flip-Flops ist grundsätzlich zwischen setz- und rücksetzdominanten Flip-Flops zu unterscheiden. Nach Instanzierung soll der Ausgang Q immer gleich 0 sein.

No.	Graphische Form	Function Block body
1	RS - Flipflop mit setzdominantem Eingang	
		
2	RS - Flipflop mit rücksetzdominantem Eingang	
		

Flankenerkennung

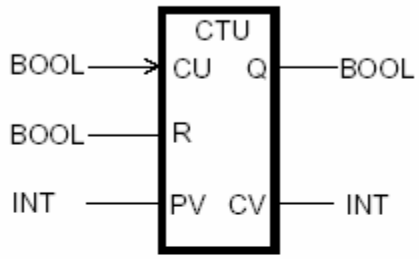
Diese Function Blocks treten oft - wie erwähnt - impliziet an den BOOL-Eingängen verschiedenster Bausteine auf.

No.	Graphische Form	Definition in ST
1	Rising-Edge detector	
		FUNCTION_BLOCK R_TRIG VAR_INPUT CLK : BOOL; END_VAR VAR_OUTPUT Q : BOOL; END_VAR VAR M : BOOL := 0; END_VAR Q := CLK AND NOT M; M := CLK; END_FUNCTION_BLOCK
2	Falling-Edge detector	
		FUNCTION_BLOCK R_TRIG VAR_INPUT CLK : BOOL; END_VAR VAR_OUTPUT Q : BOOL; END_VAR VAR M : BOOL := 1; END_VAR Q := NOT CLK AND NOT M; M := NOT CLK; END_FUNCTION_BLOCK

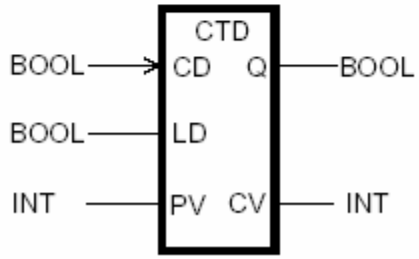
Counter

Hier die Darstellung der definierten Counter-Typen:

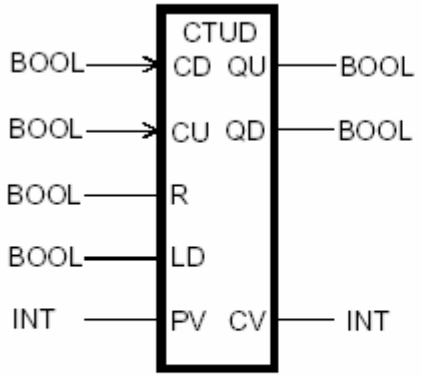
No.	Graphische Form	Definition in ST
1	Up-counter	

		`IF R THEN CV := 0;`
		`ELSIF CU AND (CV < PVmax)`
		`  THEN CV := CV + 1;`
		`END_IF;`
		`Q := (CV >= PV);`

2	Down-counter	

		`IF LD THEN CV := PV;`
		`ELSIF CD AND (CV > PVmin)`
		`  THEN CV := CV - 1;`
		`END_IF;`
		`Q := (CV <= 0);`

3 **Up-down-counter**

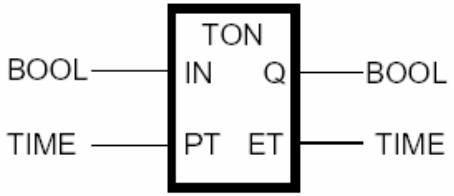


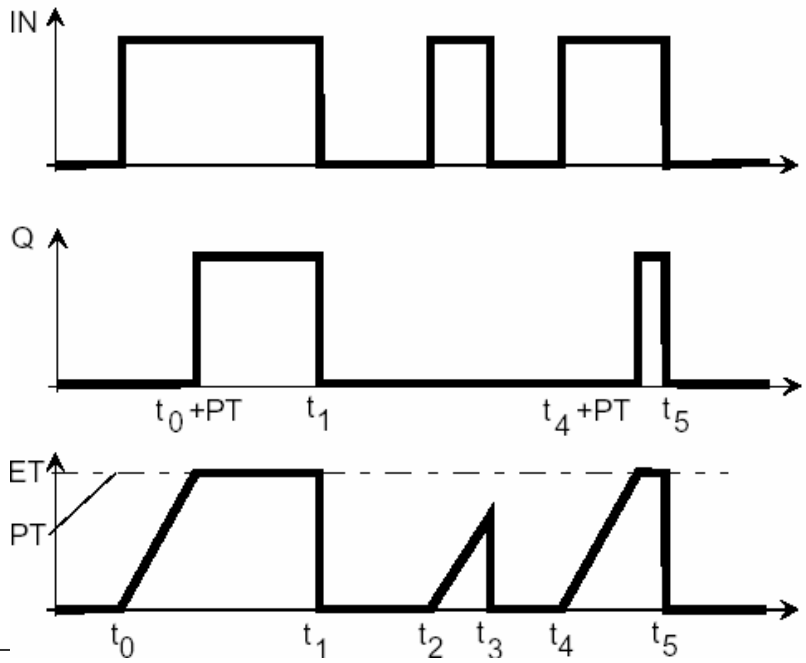

```
IF R THEN CV := 0;
ELSIF LD THEN CV := PV;
ELSIF CU AND (CV < PVmax)
   THEN CV := CV + 1;
ELSIF CD AND (CV > PVmin)
   THEN CV := CV - 1;
END_IF;
QU := (CV >= PV);
QD := (CV <= 0);
```

Bemerkung: Die Werte für PVmin und PVmax sind von der jeweiligen Implementierung abhängig

Timer

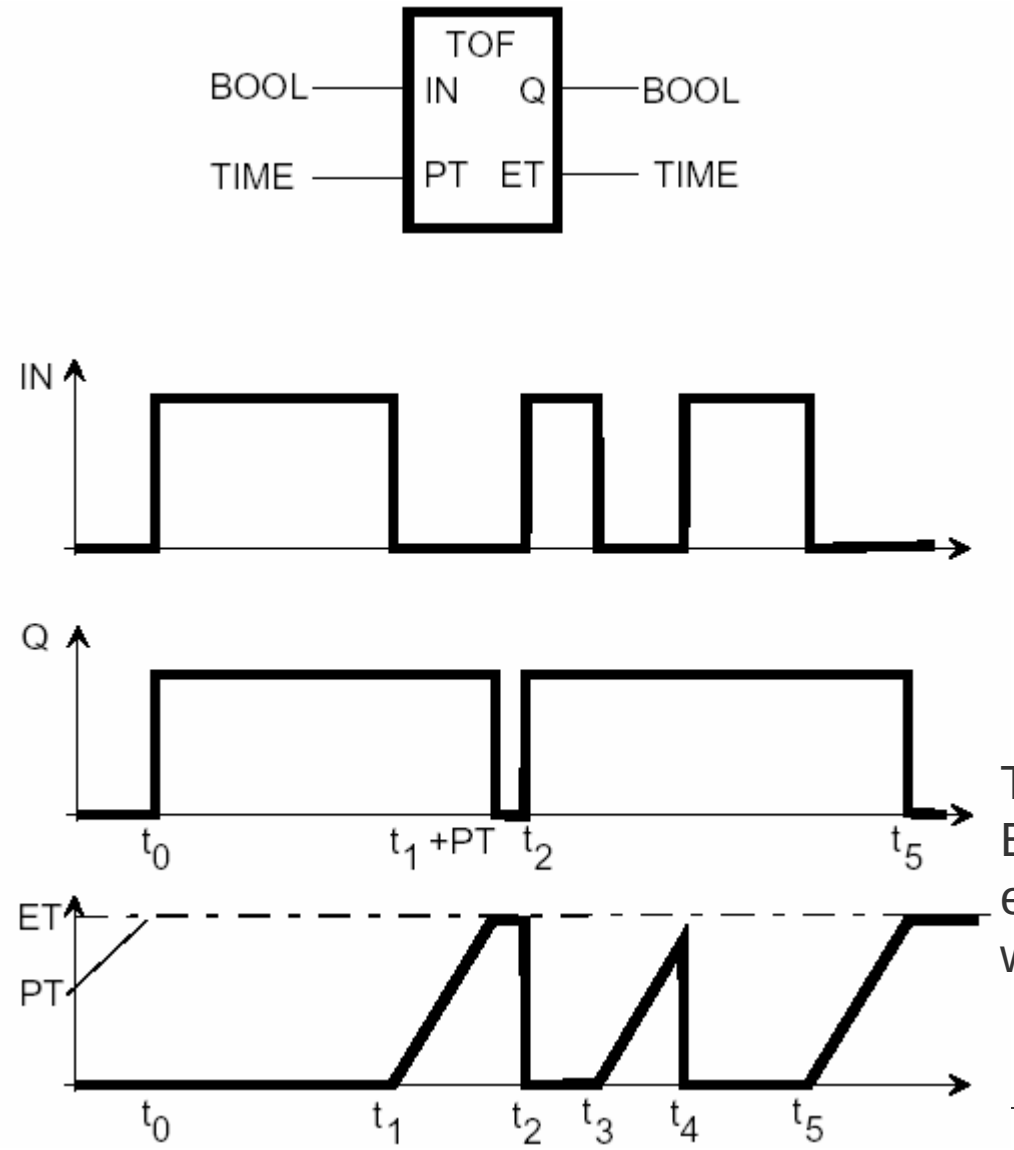
Als Standard-Timer sind eine Einschaltverzögerung (TON-Timer) und eine Ausschaltverzögerung (TOF-Timer) in der Norm vorgesehen. Die nachfolgenden Bilder zeigen die Symbole in der FBD und die Timingdiagramme.





TON kann für die Rampenauswertung eines Dimmers genutzt werden.

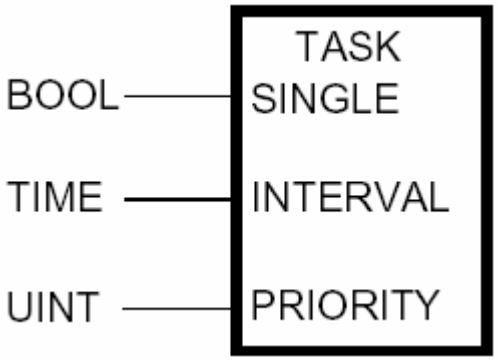
TON kann für die Einschaltverzögerung eines Aktors genutzt werden.



TOF kann für die Einschaltverzögerung eines Aktors genutzt werden.

Tasks

In der IEC1131-3 können Tasks auch wie Function Blocks dargestellt werden. Bild 1.4.9 zeigt eine solche FBD-Darstellung.



Eine textuelle Darstellung könnte wie folgt aussehen:

```
CONFIGURATION DriveControl ON FPS1
   VAR_GLOBAL x,y,z : LREAL; END_VAR
   TASK FastTask(INTERVAL = t#10ms, PRIORITY := 2);
   TASK SlowTask(INTERVAL = t#100ms, PRIORITY := 3);
   PROGRAM P1 WITH SlowTask :
         FB1(v1 := x, v2 := y, v3 := z),
         FB2(w1 := x, w2 := y) WITH FastTask;
END_CONFIGURATION
```

Zunächst wird deutlich, dass sowohl Programme an Tasks angebunden werden können (P1 an SlowTask) als auch Funktion Blocks (FB2 an FastTask). Grundsätzlich gelten im - Wesentlichen - folgende Spielregeln:

1. Eine Task, deren Intervalzeit auf Null gesetzt ist (INTERVAL = T#0s) führt genau dann einen Zykuls aus, wenn am Eingang SINGLE ein positiver Flankenwechsel erfolgt.

2. Liegt am Eingang INTERVAL eine von Null verschiedene Zeit an, so arbeitet die Task alle mit ihr verbundenen Elemente mit dieser Zykluszeit ab, solange der Eingang SINGLE mit 0 (FALSE) belegt ist. Bei einer Belegung mit 1 (TRUE) stopt die Task.

3. Es ist sowohl preemptive als auch non-preemptive Scheduling möglich.

Ladder-Diagram (LD)

Das Ladderdiagram (früher Kontaktplan KOP genannt) stellt einen formalisierten Stromlaufplan einer Schütz- oder Relaisschaltung dar. Hierbei werden logische Verknüpfungen durch die Reihen- und Parallelschaltung von (Schütz-) Kontakten realisiert. Dabei wird ein Schließer durch

```
----|   |----
```

und ein Öffner (Negation) durch

```
----| / |----
```

dargestellt. Bei den Spulen gibt es – im Wesentlichen - folgende Möglichkeiten:

Zuweisung des Verknüpfungsergebnisses auf Variable	---( )---
Negierte Zuweisung des Verknüpfungsergebnisses auf Variable	---(/)---
Setzen der Variablen, wenn Verknüpfungsergebnis TRUE	---(S)---
Rücksetzen der Variablen, wenn Verknüpfungsergebnis TRUE	---(R)---

Nachfolgend ein kleines Beispiel für einen Motor, der innerhalb eines Funktionsbausteines aus- oder eingeschaltet wird.

LDMotor (FB-KOP)

```
0001 FUNCTION_BLOCK LDMotor
0002 VAR_INPUT
0003     Motor_ein, Motor_aus : BOOL;
0004 END_VAR
0005 VAR_OUTPUT
0006     Motorschalter : BOOL;
0007 END_VAR
0008
```

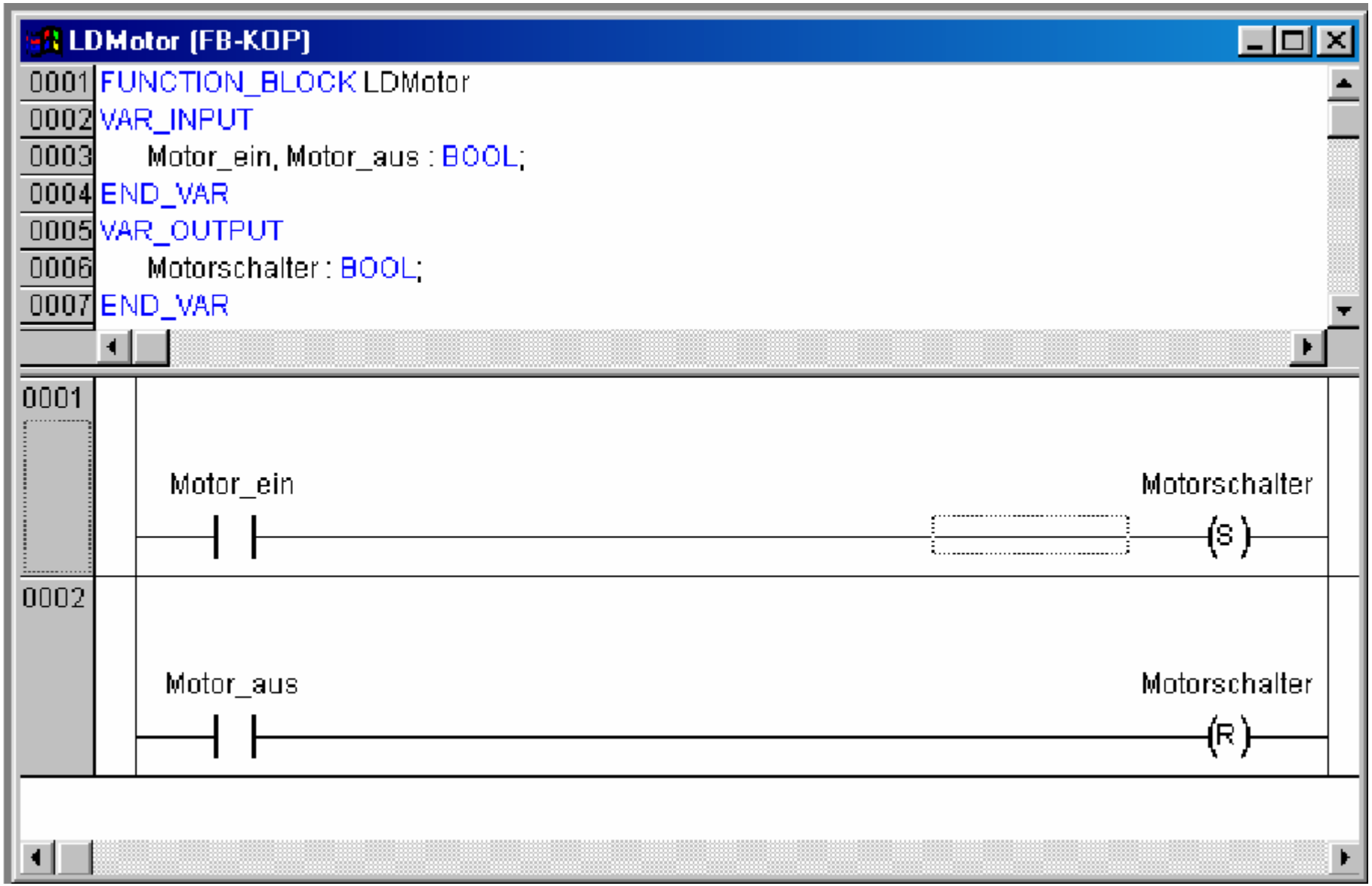
Natürlich kann man Speicherfunktionen – wie in der Schütztechnik gewohnt – durch Selbsthaltekontakte realisieren. Bild 1.5.1 zeigt diese Realisierungsmöglichkeit. Man benutzt dann nur die direkte Zuweisung auf die Spule. Einfacher geht es, wenn man die Möglichkeit der speichernden Zuweisung nutzt. Diese Variante ist in Bild 1.5.2 dargestellt.

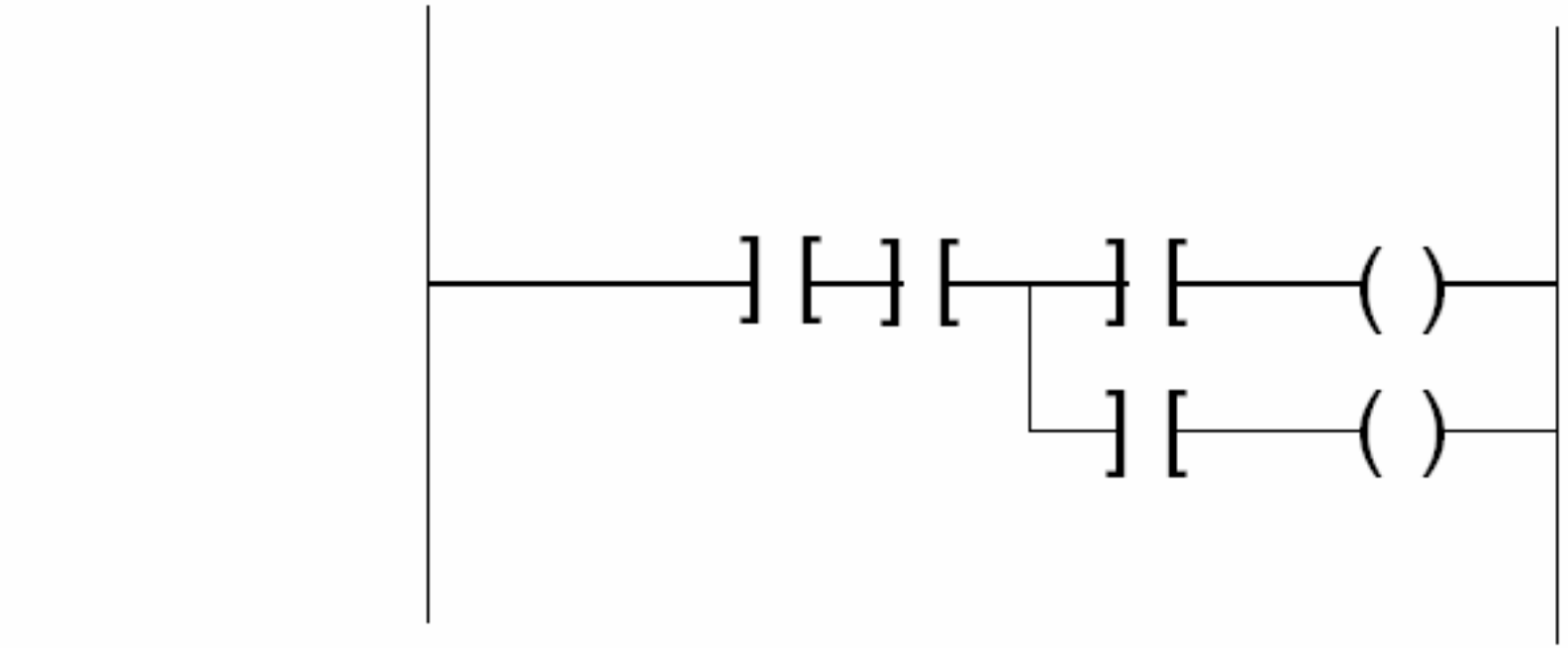
Hierbei

	0 / FALSE	1 / TRUE
---( )---	0 / FALSE	1 / TRUE
---(R)---	y(k-1)	0 / FALSE
---(S)---	y(k-1)	1 / TRUE
---(/)---	1 / TRUE	0 / FALSE

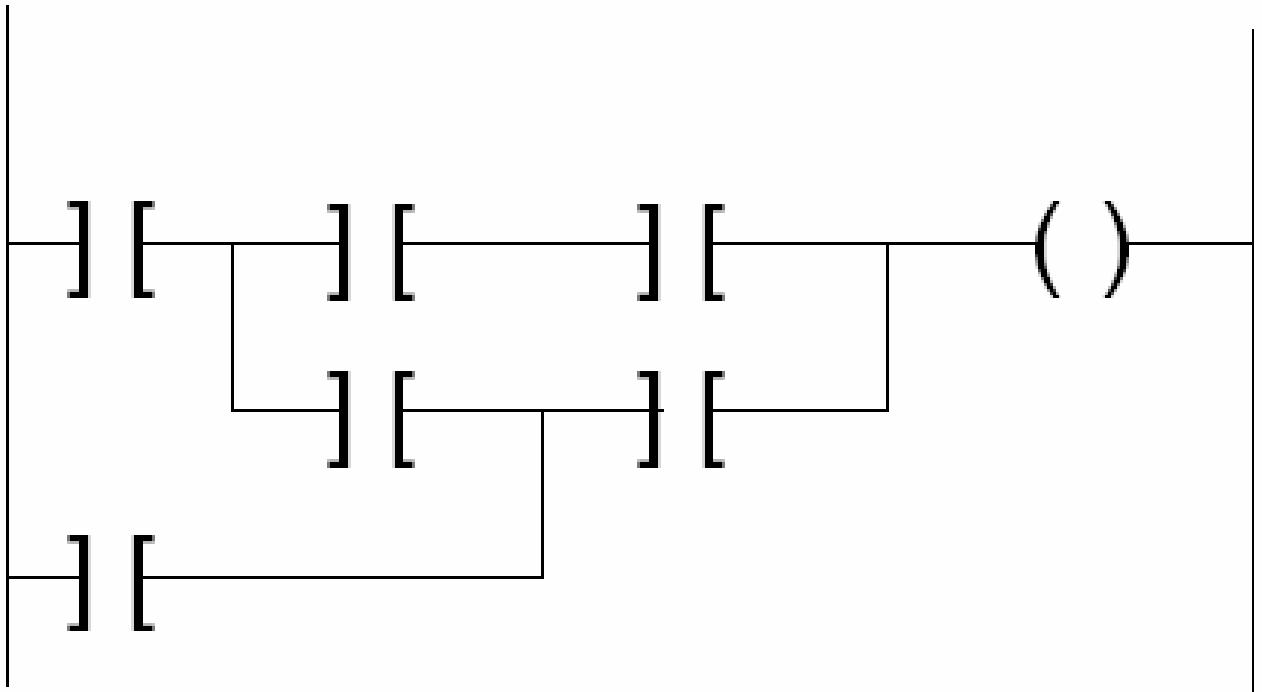
Die Darstellung von logischen Verknüpfungen mit LD entspricht nicht vollständig dem Stromlaufplan einer Schütz- oder Relaisschaltung.

- Eine Schützspule kann in einer Schaltung nur einmal vorhanden sein. Ein Zieloperand (Coil = Spule) in einem LD kann mehrfach auftauchen. Dabei ist die letzte Zuweisung die Entscheidende.
- Es ist zwar möglich mehrere Schützspulen parallel zu schalten. Verboten sind aber gemeinsame Wurzeln, bei denen logische Terilstrukturen aus einer gemeinsamen logischen Struktur gespeist werden. Bild 1.5.3 zeigt eine solche verbotene Struktur.
- Teilstrukturen, die nicht Eintoren (Zweipolen) entsprechen, sind verboten. Bild 1.5.4 zeigt eine solche verbotene Struktur.


LDMotor (FB-KOP)
0001 FUNCTION_BLOCK LDMotor
0002 VAR_INPUT
0003 Motor_ein, Motor_aus : BOOL;
0004 END_VAR
0005 VAR_OUTPUT
0006 Motorschalter : BOOL;
0007 END_VAR
0001
Motor_ein
Motorschalter
(S)
0002
Motor_aus
Motorschalter
(R)



Verbotene gemeinsame Wurzel bei LD-Programmen



Verbotene Verschachtelung bei LD-Programmen

Instruction List (IL)

Die Instruction-List kommt der alten Anweisungsliste (AWL) sehr nahe. Allerdings sind hier endlich alle Anweisungen samt den zugehörigen Spielregeln genormt.

Nummer	Operator	Modifiers	Operand	Semantics
1	LD	N	(Bemerkung 1)	Setzt das aktuelle Ergebnis auf den Wert des Operanden
2	ST	N	(Bemerkung 1)	Speichert das aktuelle Ergebnis an die Speicherstelle des Operanden
3	S	(Bemerkung 2)	BOOL	Setzt bool'schen Operanden auf 1
	R	(Bemerkung 2)	BOOL	Setzt bool'schen Operanden auf 0 zurück
4	AND	N, (	BOOL	Bool'sches UND
5	&	N, (	BOOL	Bool'sches UND
6	OR	N, (	BOOL	Bool'sches ODER
7	XOR	N, (	BOOL	Bool'sches Exklusiv – ODER
8	ADD	(	(Bemerkung 1)	Addition
9	SUB	(	(Bemerkung 1)	Subtraktion
10	MUL	(	(Bemerkung 1)	Multiplikation
11	DIV	(	(Bemerkung 1)	Division

12	GT	(	(Bemerkung 1)	Vergleich : >
13	GE	(	(Bemerkung 1)	Vergleich : >=
14	EQ	(	(Bemerkung 1)	Vergleich : =
15	NE	(	(Bemerkung 1)	Vergleich : <>
16	LE	(	(Bemerkung 1)	Vergleich : <=
17	LT	(	(Bemerkung 1)	Vergleich : <
18	JMP	C, N	LABEL	Sprung auf Label
19	CAL	C, N	NAME	Aufruf des Function-Blocks (Bemerkung 4)
20	RET	C, N		Return von aufgerufener Function oder Function – Block
21	)			Berechne vorhergehende Operation

Bemerkung 1:
Diese Operatoren können auch überladen oder typisiert werde. Das Verknüpfungsergebnis ist dann vom gleichen Typ, wie der Operand.

Bemerkung 2:
Diese Operationen werden nur durchgeführt, wenn das Verknüpfungsergebnis boolsch und vom Wert 1 ist.

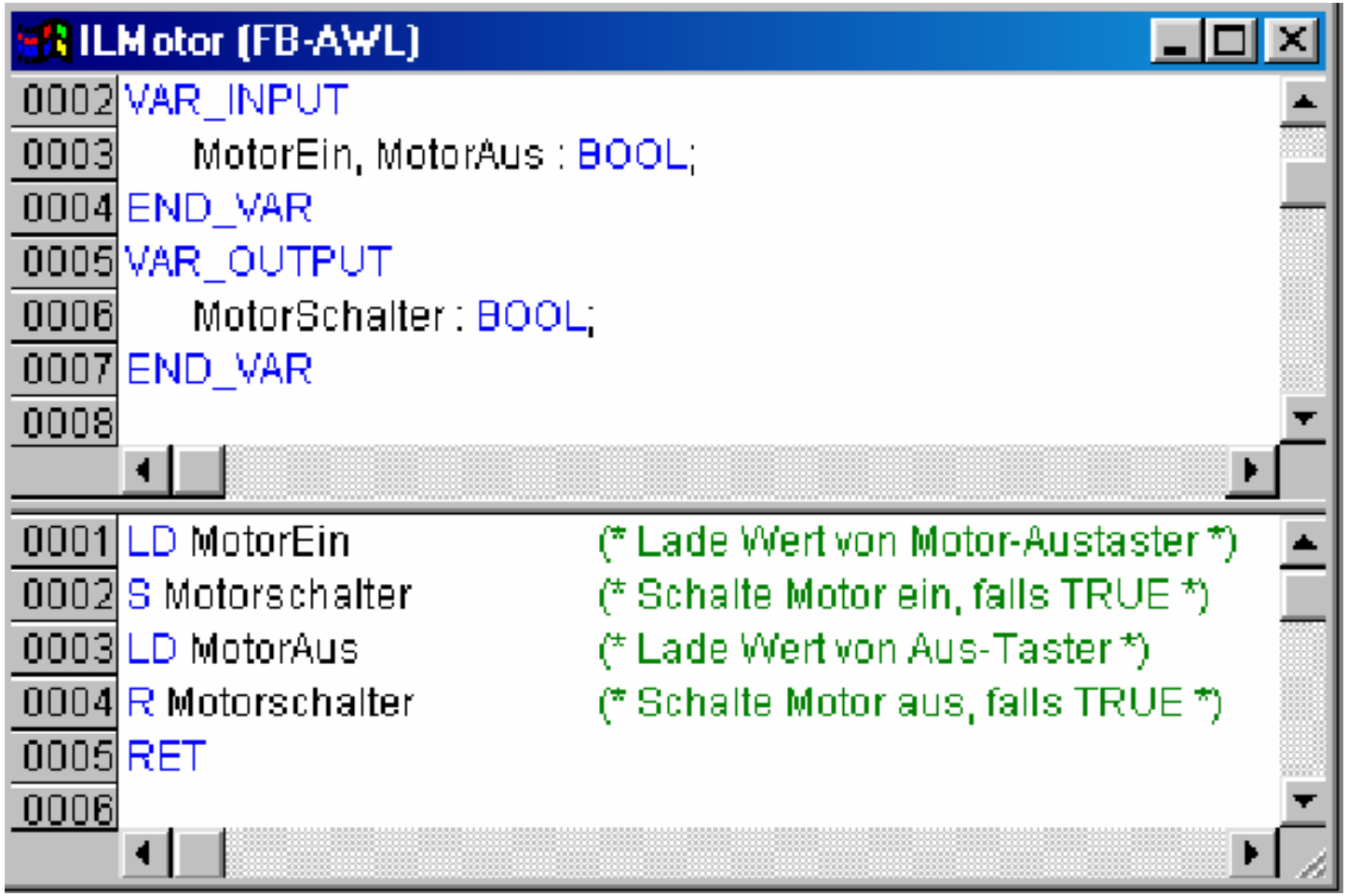
Nachfolgend zwei Beispiele für Anwendungen in IL.

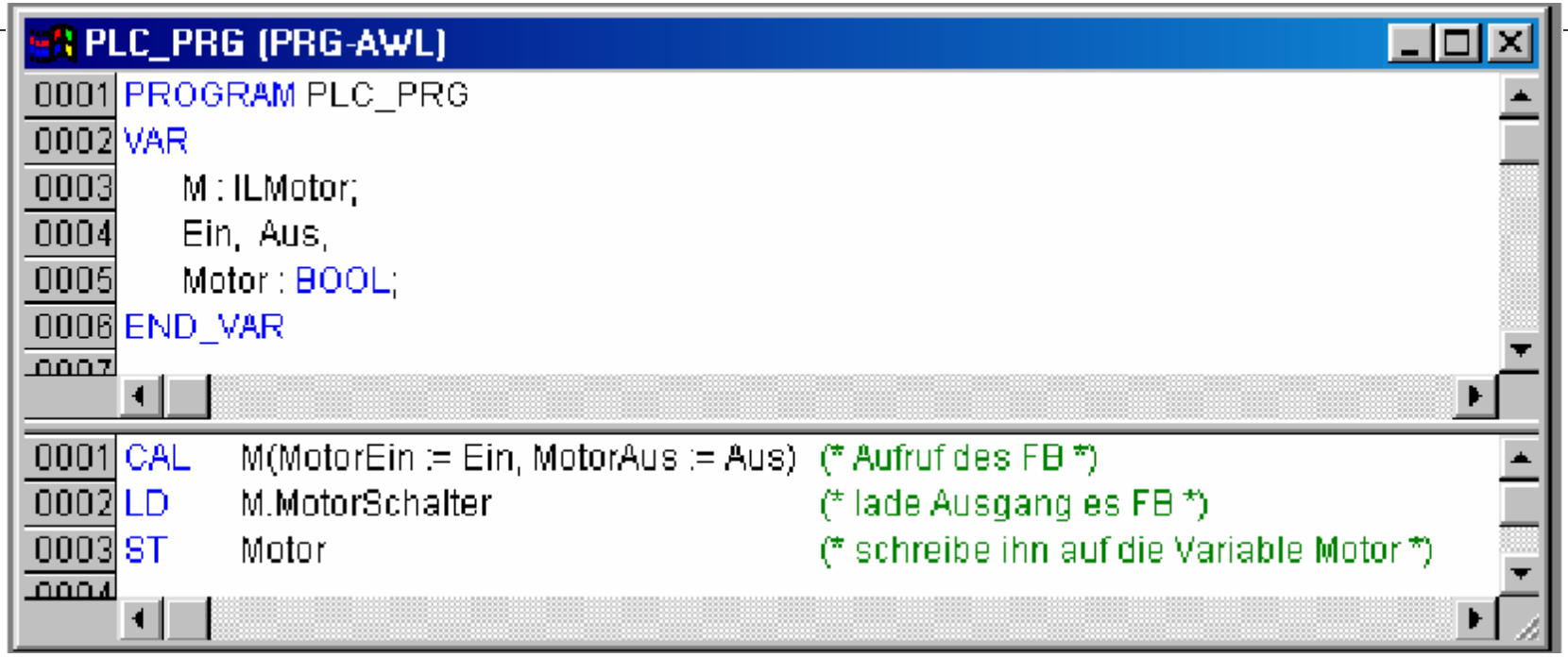
ILMotor (FB-AWL)

```
0002 VAR_INPUT
0003     MotorEin, MotorAus : BOOL;
0004 END_VAR
0005 VAR_OUTPUT
0006     MotorSchalter : BOOL;
0007 END_VAR
0008
```

```
0001 LDN MotorAus          (* Lade negierten Wert von Aus-Taster *)
0002 AND (MotorSchalter    (* Undiere mit  altem Schaltwert *)
0003 OR MotorEin           (* geodert mit Ein-Taster *)
0004 )
0005 ST Motorschalter      (* speichere neuen Wert auf Ausgangsvariable *)
0006 RET
0007
0008
```

Das erste Beispiel nach Bild 1.5.5 zeigt eine Zuweisung mit Haltekontakt (vgl: Bild 1.5.1)





Im zweiten Beispiel nach Bild 1.5.6 (vgl. Bild 1.5.2) ist das gleiche Problem mit einer setzenden und rücksetzenden Zuweisung gelöst. Diese Art von Anweisungen werden immer nur dann ausgeführt, wenn das vorhergehende Verknüpfungsergebnis vom Typ BOOL und vom Wert 1 (TRUE) ist. Da nur der zuletzt zugewiesene Wert bleibt, ist die Wirkung des Aus-Tasters hier dominant.

Structured Text (ST)

Structured Text ist eine textuelle Hochsprache mit starker Ähnlichkeit zu Pascal. Sie bietet die meisten Freiheitsgrade und stellt damit die Prozesssprache der IEC1131-3 dar, die gewöhnlich Informatikern am meisten ‚liegt'.

Tabelle 1.5.4 zeigt die möglichen Operatoren in logischen oder arithmetischen Verknüpfungen.

No.	Operation	Symbol	Precedence
1	Paranthesiazion	(expression)	Highest
2	Function evaluation Examples:	Identifier(argument list) LN(A), MAX(X,Y), . . .	
3	Exponentiation	**	
4	Negation	-	
5	Complement	Not	
6	Multiply	*	
7	Divide	/	
8	Modulo	MOD	
9	Add	+	
10	Subtract	-	
11	Comparison	<, >, <=, >=	
12	Equality	=	
13	Inequality	<>	
14	Boolean AND	&	
15	Boolean AND	AND	
16	Boolean Exclusive OR	XOR	
17	Boolean OR	OR	Lowest

Auch die Kontrolstrukturen sehen, wie in Tabelle 1.5.5 dargestellt, der Sprache Pascal sehr ähnlich. Geklammert werden hier mehrere Statements nicht mit ‚**begin**' und ‚**end**' wie bei Pascal oder mit ‚**{**, und ‚**}**' wie bei C sondern mit einem typisierten ‚**end_xyz**', das auf den Typ des einleitenden Kontrolstatements xyz hinweist.

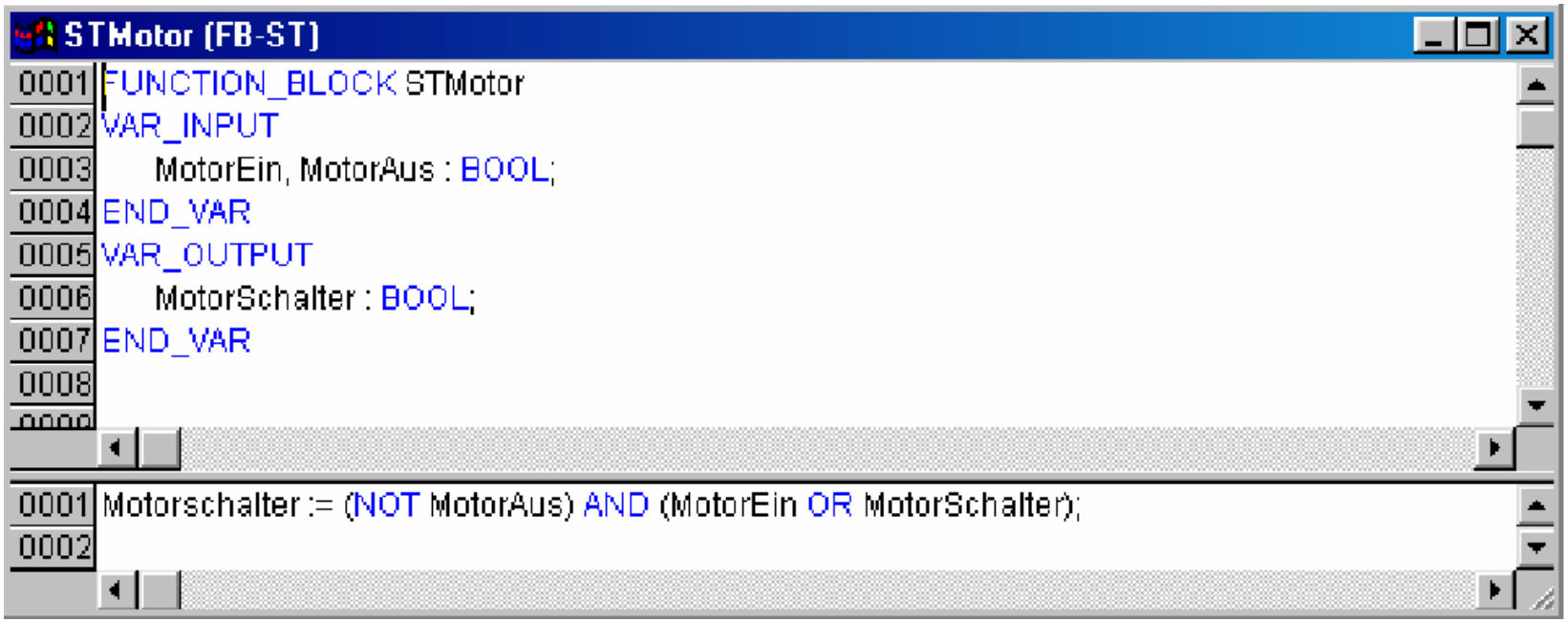
No.	Statement type / Reference	Example
1	Assignment	A := B; CV := CV + 1; C:= SIN(X)
2	Function block Invocation and FB output usage	CMD_TMR(IN = %IX5, PT = T#300ms); A := CMD_TMR.Q;
3	RETURN	RETURN
4	IF	D := B * B – 4 * A * C; IF D < 0.0 THEN NROOTS := 0; ELSIF D = 0.0 THEN NROOTS := 1; X1 := - B / (2.0 * A);; ELSE NROOTS := 2; X1 := (- B + SQRT(D)) / (2.0 * A); X2 := (- B - SQRT(D)) / (2.0 * A); END_IF;

5	CASE	TW := BCD_TO_INT(THUMBWHEEL); TW_ERROR := 0; CASE TW OF 1, 5 : DISPLAY := OVEN_TEMP; 2 : DISPLAY := MOTOR_SPEED;
		3 : DISPLAY := GROSS – TARE; 4, 6 ..10 : DISPLAY := STATUS(TW – 4); ELSE DISPLAY := 0; TW_ERROR := 1; END_CASE; %QW100 := INT_TO_BCD(DISPLAY);

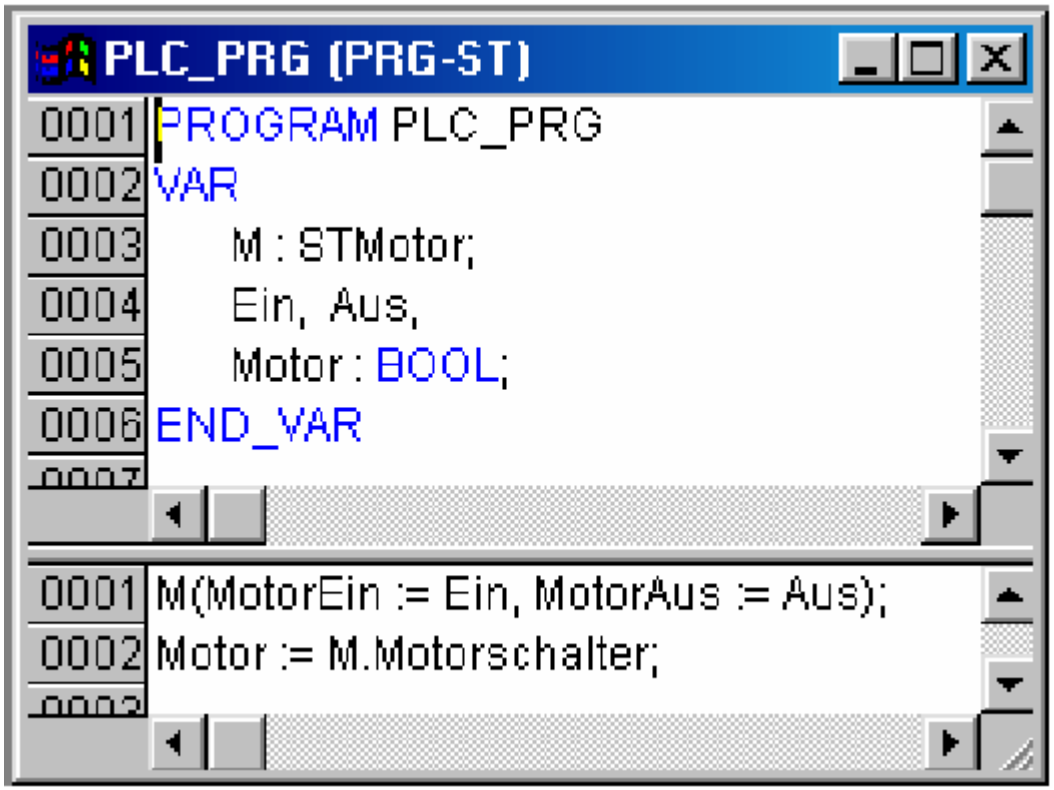
6	FOR	J := 101;
		FOR I := 1 TO 100 BY 2 DO
		IF WORDS[I] = 'KEY' THEN
		J := I;
		EXIT;
		END_IF;
		END_FOR;

7	WHILE	J := 101; WHILE J <= 100 & WORDS[J] <> 'KEY' DO J := J + 2; END_WHILE;
8	REPEAT	J := -1; REPEAT J := J + 2; UNTIL J = 101 ORWORDS[J] = 'KEY'; END_REPEAT;
9	EXIT	EXIT;
10	Empty Statement	;;
Bemerkung: Wenn das EXIT – Statement (9) unterstützt wird, dann soll es in allen Iterations-Statements (FOR, WHILE, REPEAT) verwendet werden können. Hierbei erfolgt ein vorzeitiger Abbruch der Schleife.		

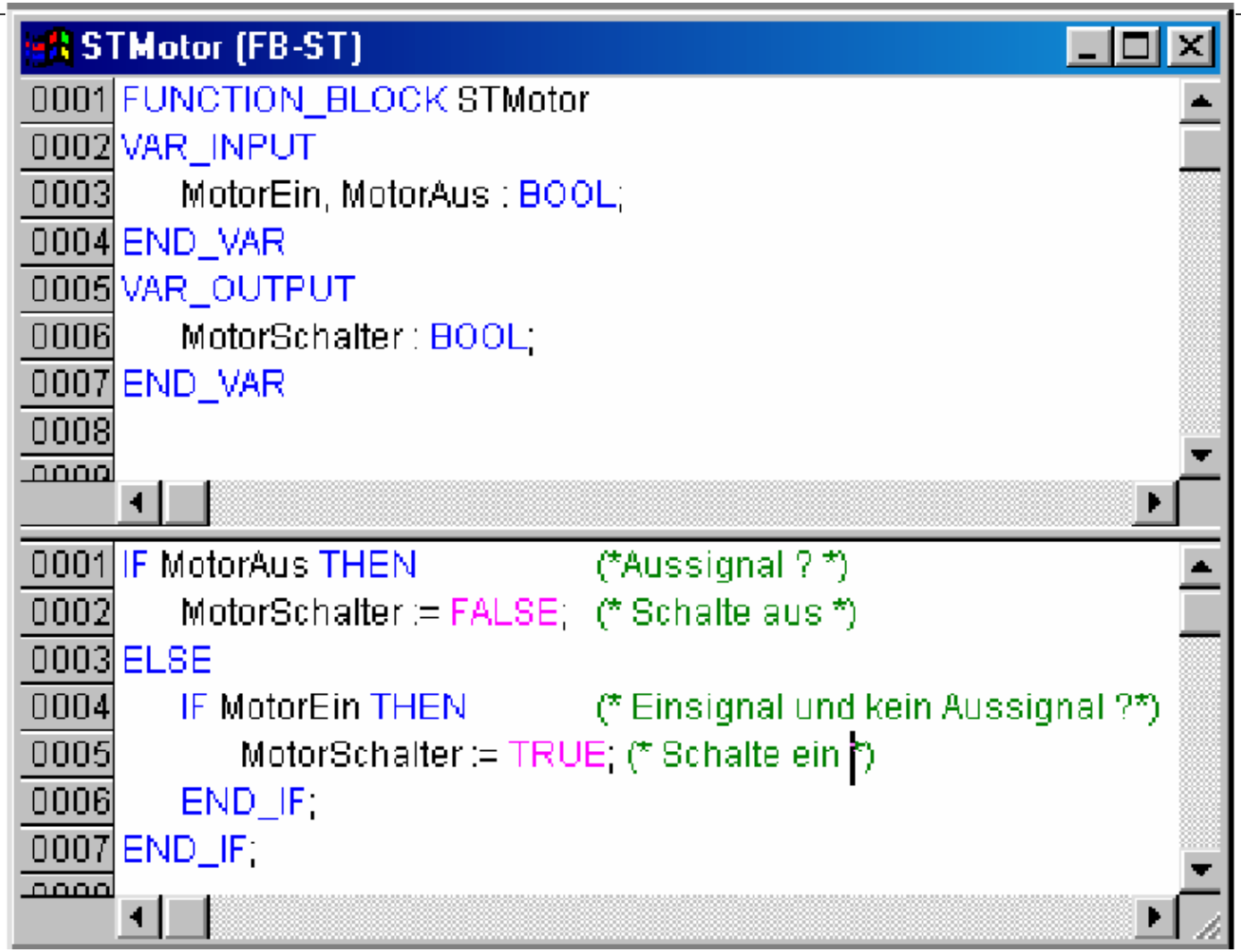
Auch sollen unsere beiden Beispiele eines Motors, der mit einem Austaster ausgeschaltet und mit einem Einschalter eingeschaltet wird, wieder verwendet werden.



Die logischen Verknüpfungen nach Bild 1.5.7 beschreiben die gewohnte Struktur.



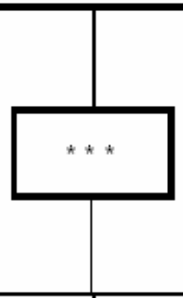
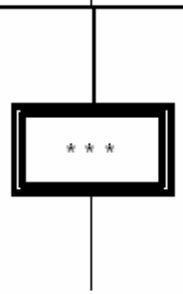
Beim Aufruf einer Instanz eines Funktionsbausteins – in Bild 1.5.6 ist es ‚**M**' – müssen nicht alle Parameter übergeben werden. Die Identifikation der Parameter erfolgt hierbei nicht durch die Reihenfolge sondern durch die explizite Zuweisung. Ein Aufruf des FB bedeutet auch immer zugleich, dass dieser operiert wird. Die Ausgänge des FB können dann einzeln in einem Ausdruck des entsprechenden Datentyps abgefragt werden.



In der Realisierung des FB nach Bild 1.5.9 werden die Eingänge in einer geschachtelten IF-Abfrage behandelt. Da das Einschalten nur im ELSE-Teil der Abfrage des Aussignals erfolgt, hat dieses Signal Vorrang.

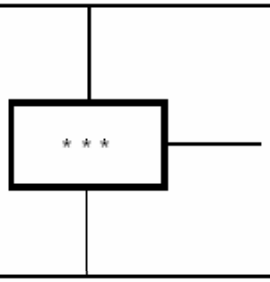
Sequential Function Chart (SFC)

Steps

No.	Representation	Description
1		Graphische Darstellung mit gerichteten Links *** = Name des Step
		Initial step (Anfangsschritt) mit gerichteteten Links *** = Name des Initial step Bemerkung: Der obere Link ist nicht erforderlich, wenn der Step keine Vorgänger hat

2	`STEP *** :`
	`  (* Step body *)`
	`END_STEP`
		Step in textueller Form ohne gerichtete Links.
		
		*** = Name des Step

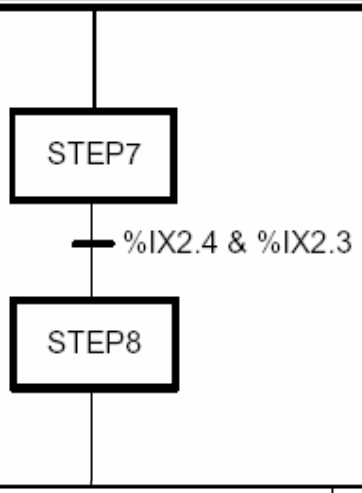
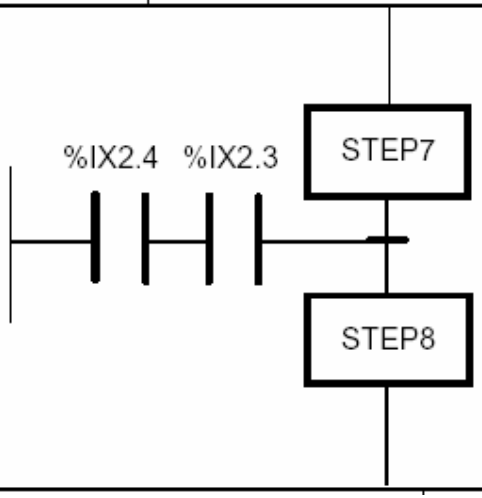
	`INITIAL_STEP *** :`
	`  (* Step body *)`
	`END_STEP`
		Initial-Step in textueller Form ohne gerichtete Links.
		
		*** = Name des Step

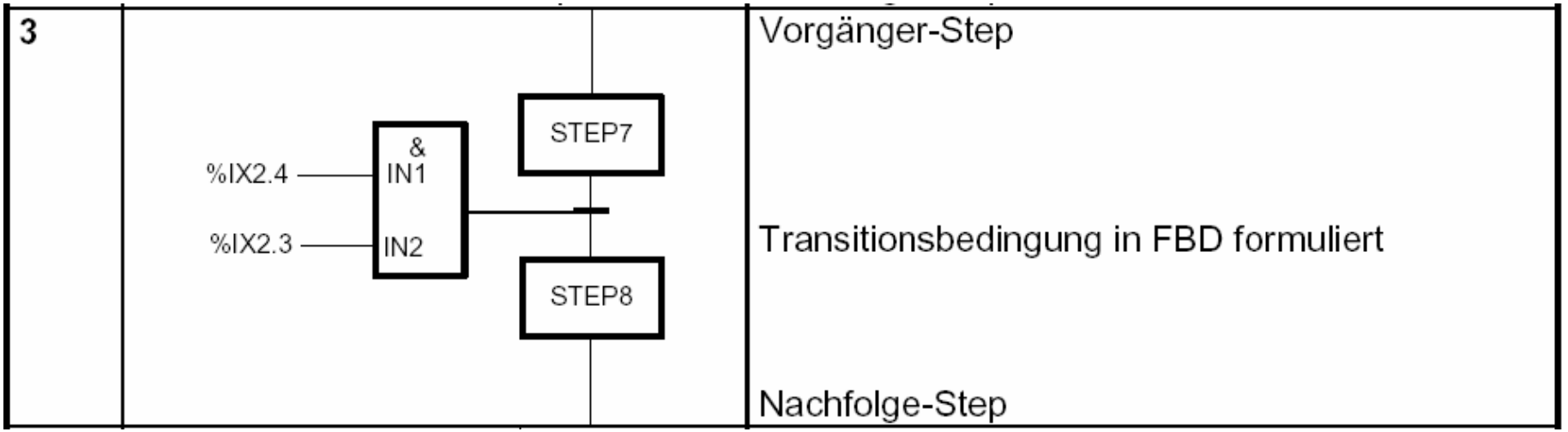
3a	***.X	Step-Flag, Allgemeine Form *** = Name des Step ***.X = Boolean 1 wenn *** aktiv ist, sonst Boolean 0
3b		Step-Flag Direkte Anbindung der bool'schen Variable ***.X an die rechte Seite des Step ***
4	***.T	Step elapsed time (abgelaufene Zeit) *** = Name des Step ***.T ist eine Variable vom Typ TIME

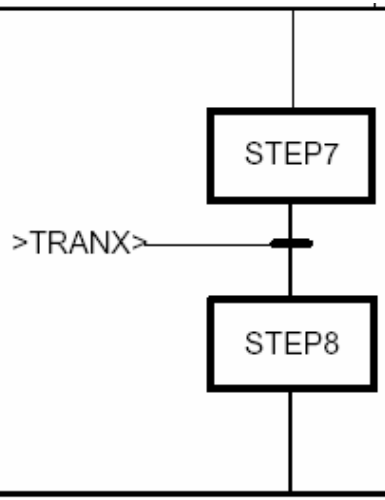
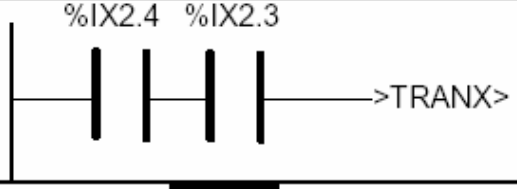
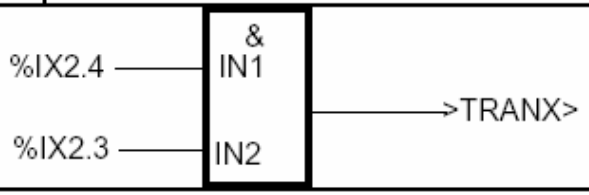
Bemerkung: Wenn die Eigenschaften 3a, 3b oder 4 nicht unterstützt werden, muss es zu einem Fehlerführen, wenn versucht wird, die entsprechende Variable zu verändern. Damit würden folgende Statements zu Fehlermeldungen führen:

```
S4.X := 1;                S4.T := t#100ms;
```

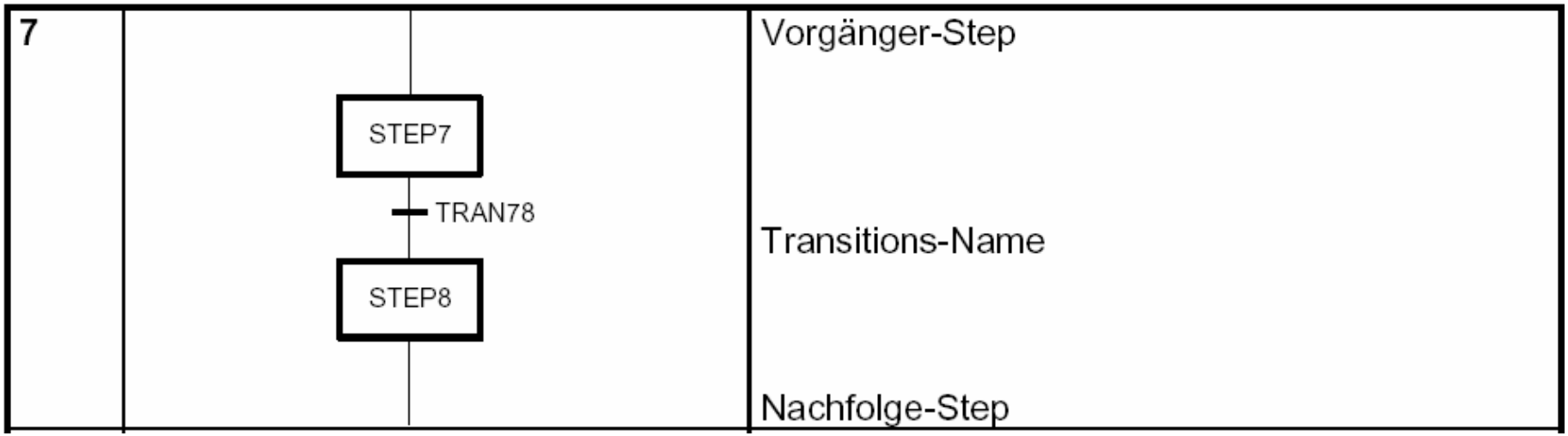
Transitions

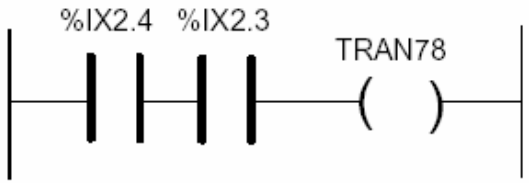
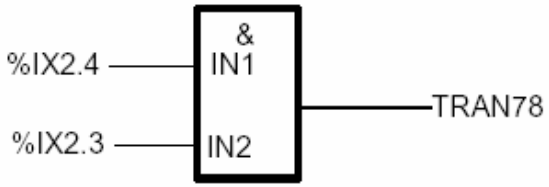
No.	Example	Description
1		Vorgänger-Step Transitionsbedingung in ST formuliert Nachfolge-Step
2		Vorgänger-Step Transitionsbedingung in LD formuliert Nachfolge-Step


3
Vorgänger-Step
STEP7
&
%IX2.4
IN1
%IX2.3
IN2
Transitionsbedingung in FBD formuliert
STEP8
Nachfolge-Step

4		Vorgänger-Step Transition connector Nachfolge-Step
4a		Transitionsbedingung in LD formuliert
4b		Transitionsbedingung in FBD formuliert

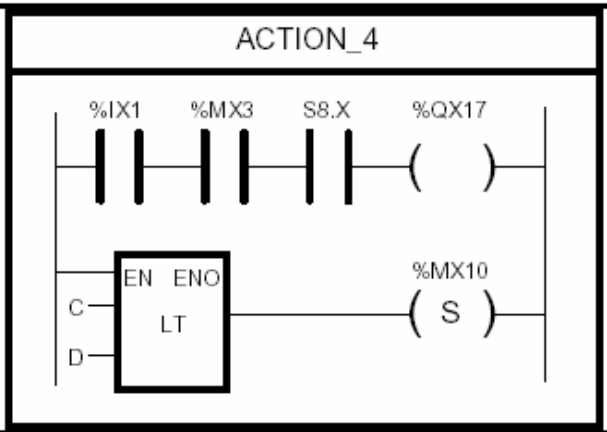
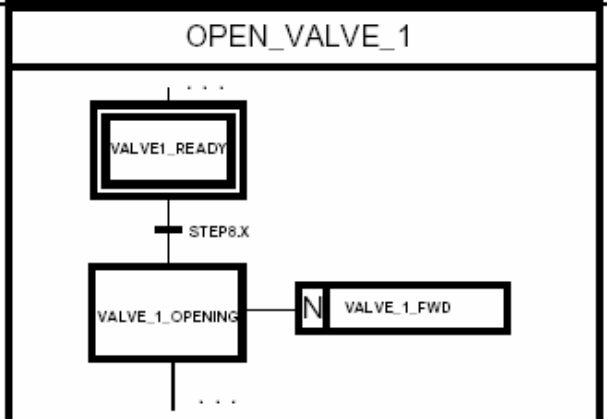
5	STEP STEP7: END_STEP TRANSITION FROM STEP7 TO STEP8 := %IX2.4 & %IX2.3; END_TRANSITION STEP STEP8: END_STEP	Textliches Äquivalent zu Eigenschaft 1 in ST formuliert
6	STEP STEP7: END_STEP TRANSITION FROM STEP7 TO STEP8 LD %IX2.4 AND %IX2.3 END_TRANSITION STEP STEP8: END STEP	Textliches Äquivalent zu Eigenschaft 1 in IL formuliert


7
STEP7
TRAN78
STEP8
Vorgänger-Step
Transitions-Name
Nachfolge-Step

7a	TRANSITION TRAN78:  END_TRANSITION	Transitionsbedingung in LD formuliert
7b	TRANSITION TRAN78:  END_TRANSITION	Transitionsbedingung in FBD formuliert
7c	TRANSITION TRAN78: LD %IX2.4 AND %IX2.3 END_TRANSITION	Transitionsbedingung in IL formuliert
7d	TRANSITION TRAN78: := %IX2.4 & %IX2.3; END_TRANSITION	Transitionsbedingung in ST formuliert

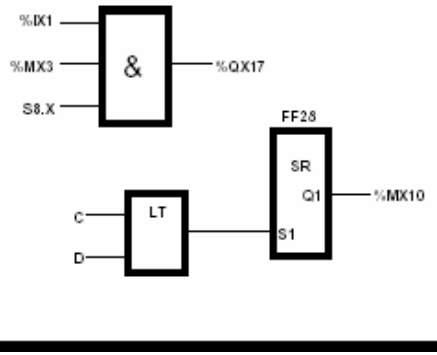
Actions

No.	Feature
1	Jede bool'sche Variable, die in einem VAR oder VAR_OUTPUT Block deklariert wurde oder ihre graphischen Äquivalente können eine Action sein.

No.	Example	Feature
2l		Graphische Darstellung in LD
2s		Einschluss von SFC-Elementen in einer Action

2f

ACTION_4



Graphische Darstellung in FBD

3s

```
ACTION ACTION_4:
    %QX17 := %IX1 & %MX3 & S8.X;
    FF28(S1 := (C<D));
    %MX10 := FF28.Q1;
END_ACTION
```

Textuelle Darstellung in ST

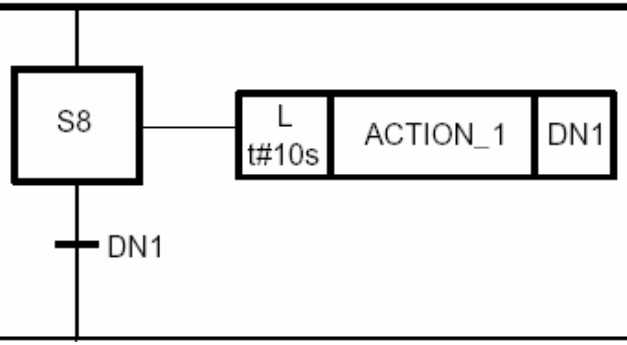
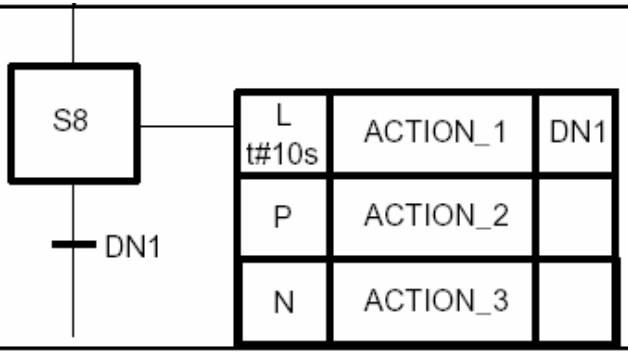
3i

```
ACTION ACTION_4:
LD      S8.X
AND     %IX1
AND     %MX3
ST      %QX17
LD      C
LT      D
S1      FF28
ST      FF28.Q1
ST      %MX10
END_ACTION
```

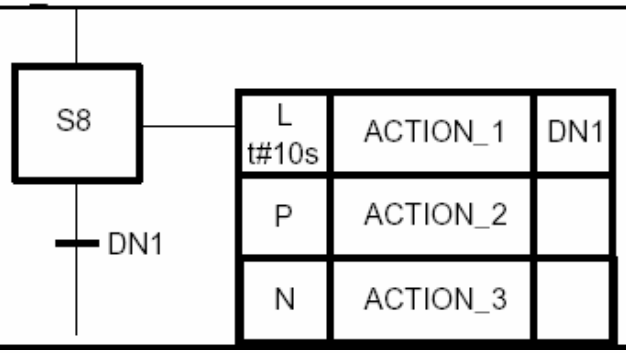
Graphische Darstellung in IL

Bemerkung:
Das Step-Flag S8.X wird in den Beispielen benötigt, um sicherzustellen, dass wenn der Step S8 deaktiviert wird, %QX17 := 0 wird.

Action-Associations mit Steps

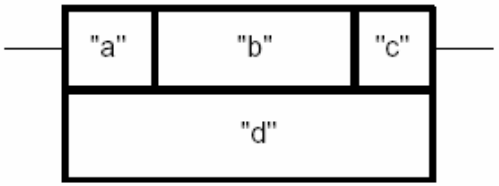
No.	Example	Feature
1		Action-Block
2		Aneinandergereihte Action-Blocks

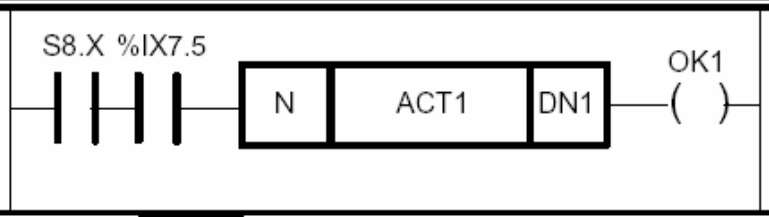
3	`STEP S8:`
	`  ACTION_1(L,t#10s,DN1);`
	`  ACTION_2(P);`
	`  ACTION_3(N);`
	`END_STEP`
		Textuell formulierter Step-Body

4		Action-Block “d” - Feld

Bemerkung:
Wenn die Eigenschaft 4 benutzt wird, kann der korrespondierende Name der Action nicht in einem anderen Action-Block verwendet werden.

Action-Blocks

No.	Feature	Graphical form
1	"a": Qualifier nach Tab. 1.5.11	
2	"b": Name der Action	
3	"c": Bool'sche Feedback-Variable	
	"d": Action formuliert in	
4	IL	
5	ST	
6	LD	
7	FBD	

No.	Feature / Example
8	
9	

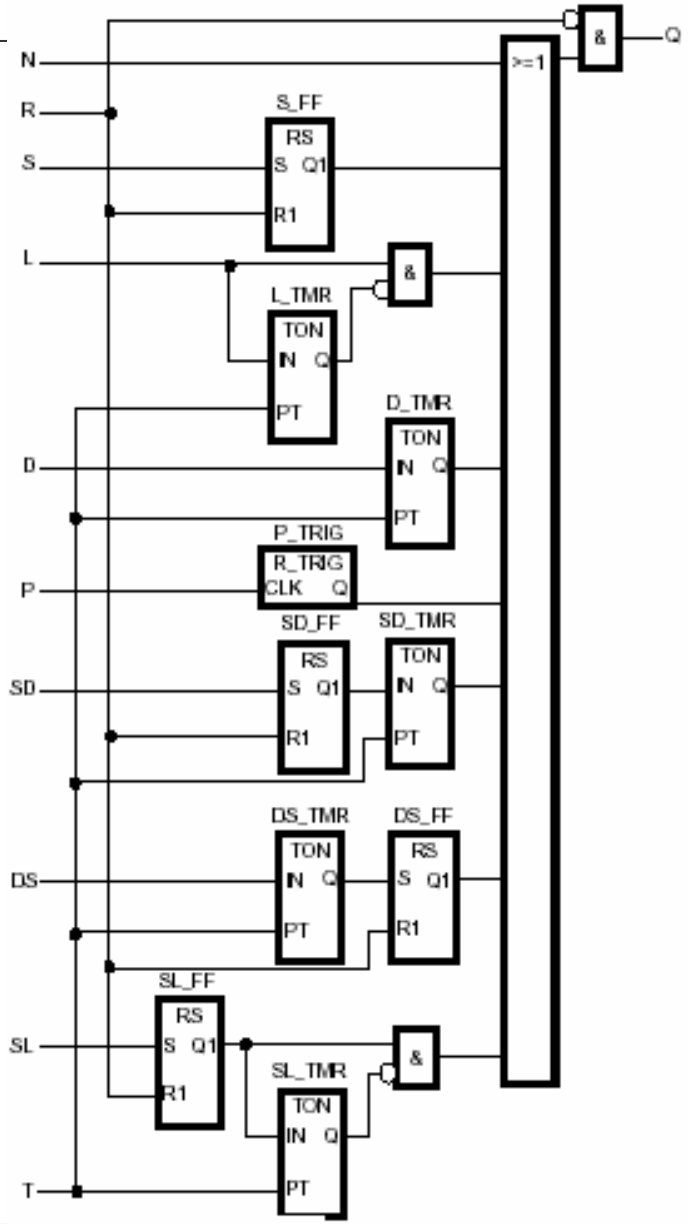
Bemerkung 1 : Das Feld "a" kann weggelassen werden, wenn der Qualifier "N" ist.
Bemerkung 2 : Das Feld „c" kann wegelassen werde, wenn keine Feedback-Variable verwendet wird.

Action-Qualifier

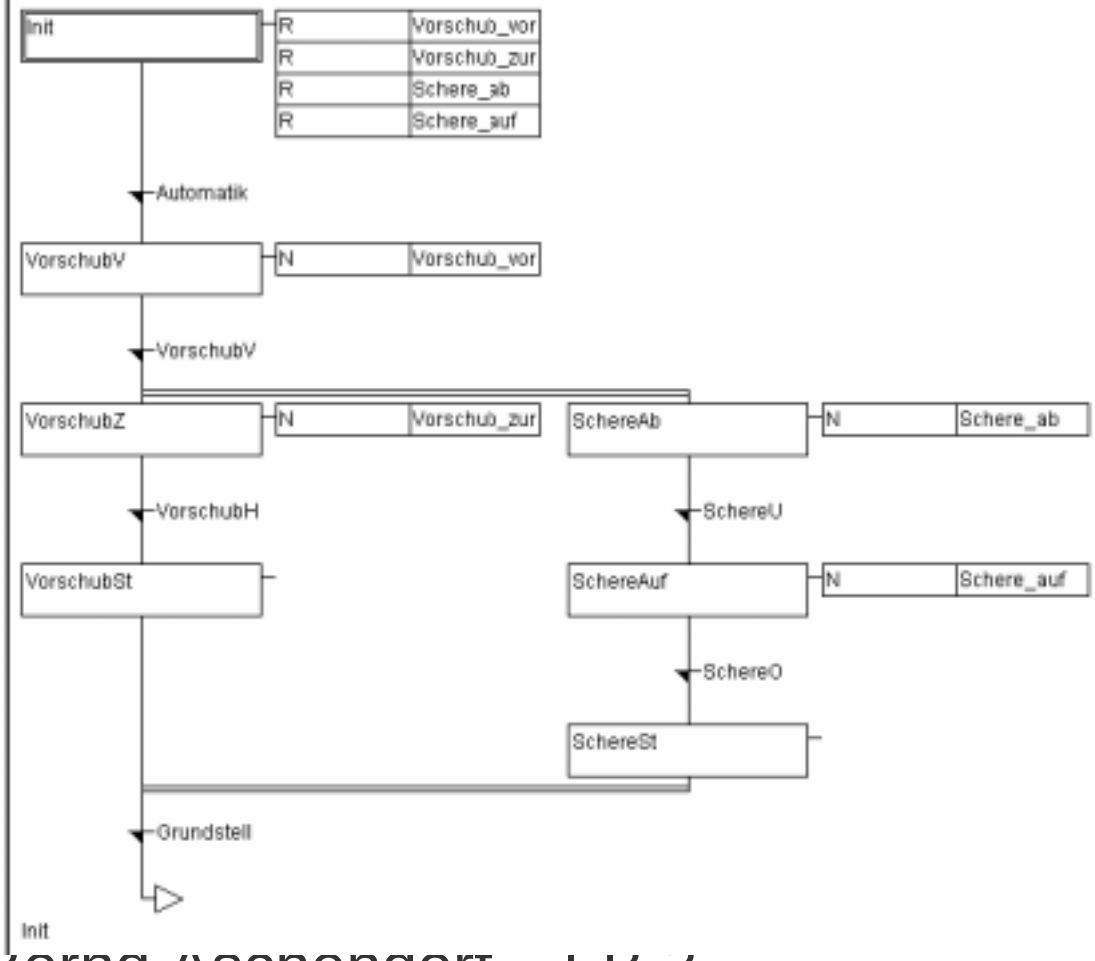
Die Sequential Function Chart bietet eine große Pallette von Möglichkeiten hinsichtlich des Modus des Actions-Controls. Tabelle 1.5.11 zeigt die vorgesehenen Qualifier auf.

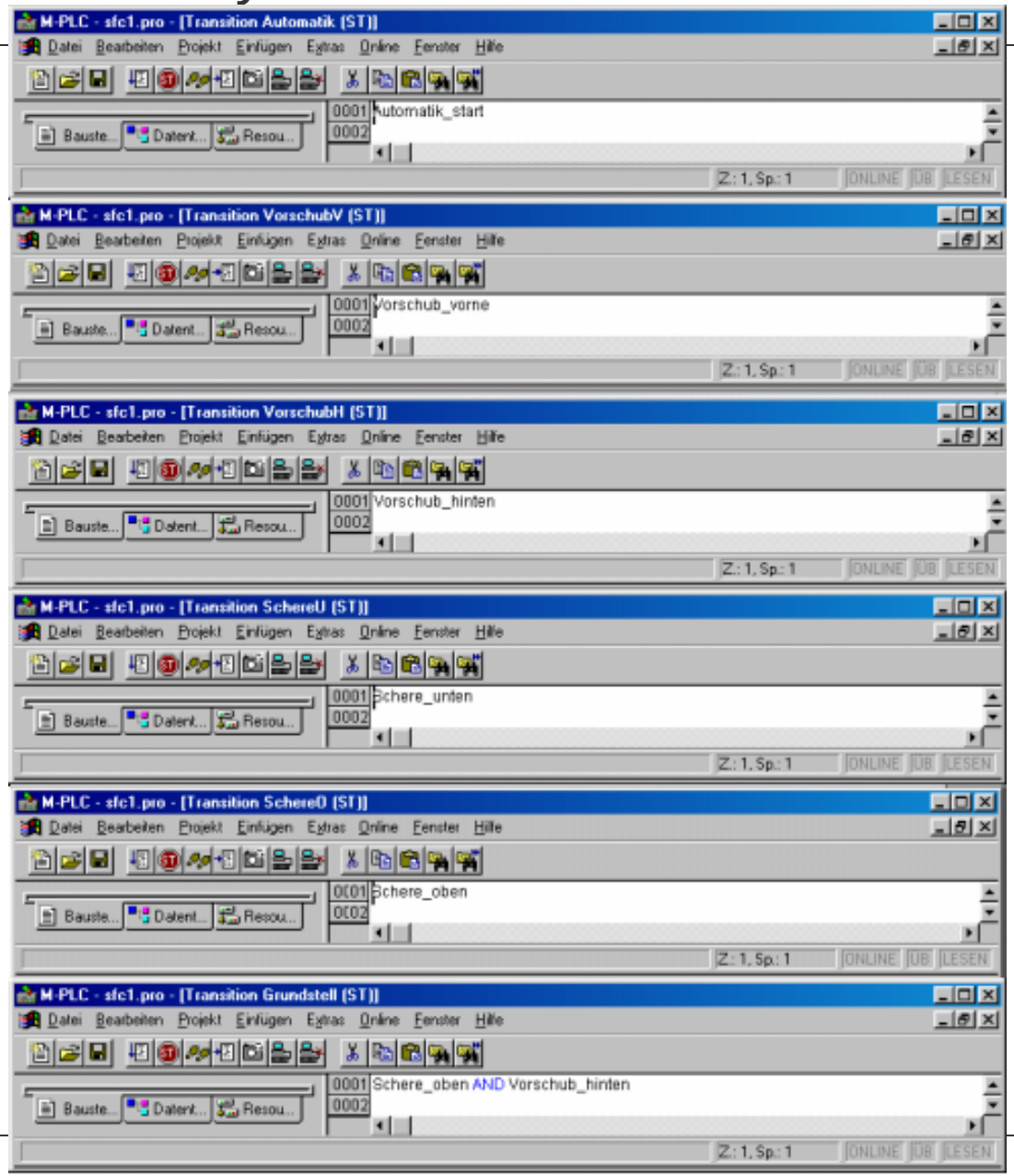
No.	Qualifier	Explanation
1	None	Non-stored (null qualifier)
2	**N**	**N**on-stored
3	**R**	overriding **R**eset
4	**S**	**S**et (stored)
5	**L**	time **L**imited
6	**D**	time **D**elayed
7	**P**	**P**ulse
8	**SD**	**S**tored and time **D**elayed
9	**DS**	**D**elayed and **S**tored
10	**SL**	**S**tored and time **L**imited

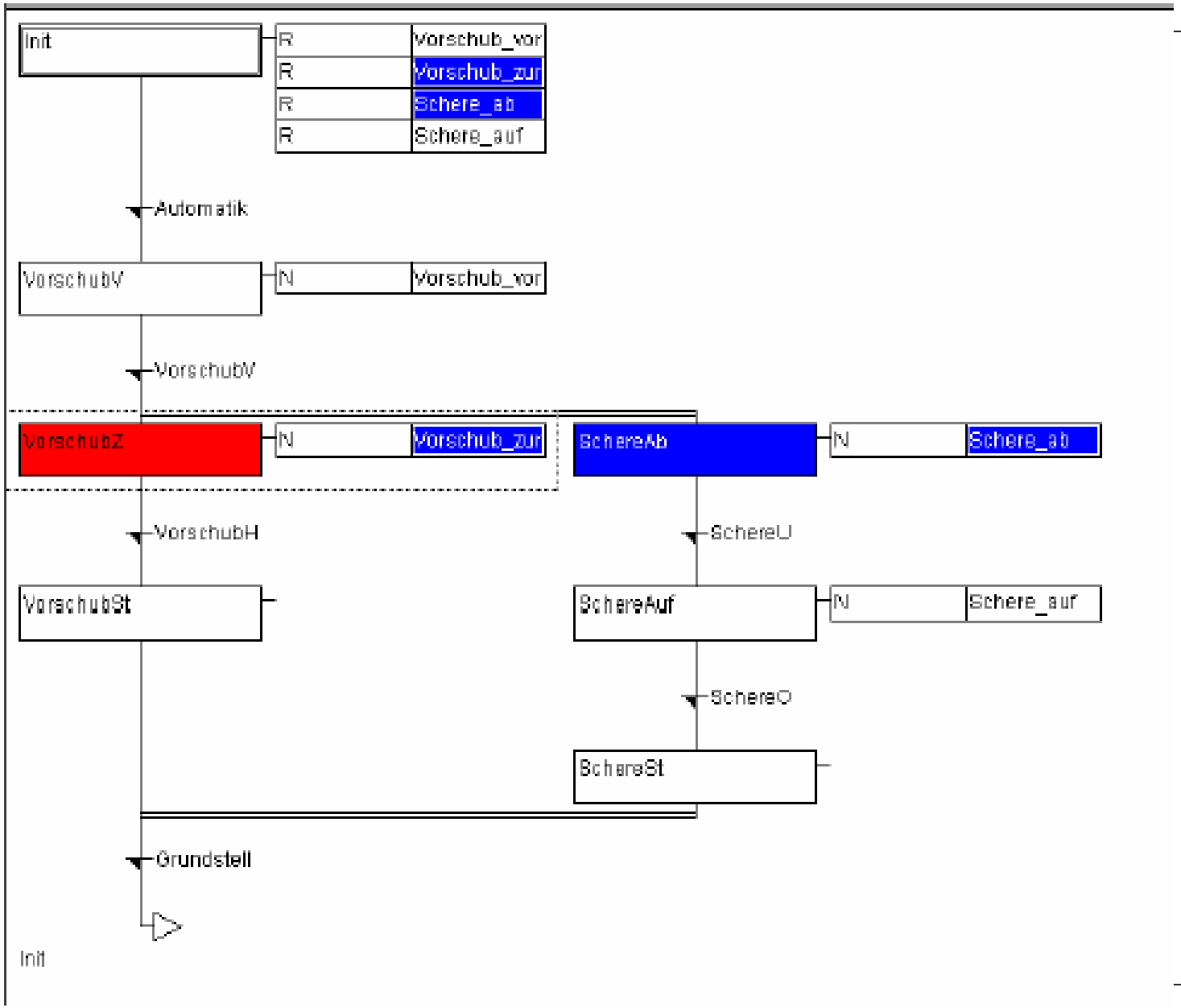
Bild 1.5.11 zeigt zum Verständnis ein Schaltbild, das die Funktion der Qualifier beschreibt. Der Qualifier **R** bewirkt ein sofortiges Rücksetzen der hiermit assoziierten Action.


N
R
S
L
D
P
SD
DS
SL
T
S_FF
RS
S Q1
R1
L_TMR
TON
IN Q
PT
D_TMR
P_TRIG
R_TRIG
CLK Q
SD_FF
SD_TMR
DS_TMR
DS_FF
SL_FF
SL_TMR
&
>=1
Q

```
0001 FUNCTION_BLOCK Coil
0002 VAR_INPUT
0003     Automatik_start,
0004     Vorschub_vorne, Vorschub_hinten, Schere_unten, Schere_oben : BOOL;
0005 END_VAR
0006 VAR_OUTPUT
0007     Vorschub_vor, Vorschub_zur, Schere_ab, Schere_auf : BOOL;
0008 END_VAR
```




M-PLC - sfc1.pro - [Transition Automatik (ST)]
Datei Bearbeiten Projekt Einfügen Extras Online Fenster Hilfe
Bauste... Datent... Resou...
0001 Automatik_start
0002
Z: 1, Sp: 1 ONLINE ÜB LESEN
M-PLC - sfc1.pro - [Transition VorschubV (ST)]
Datei Bearbeiten Projekt Einfügen Extras Online Fenster Hilfe
Bauste... Datent... Resou...
0001 Vorschub_vorne
0002
Z: 1, Sp: 1 ONLINE ÜB LESEN
M-PLC - sfc1.pro - [Transition VorschubH (ST)]
Datei Bearbeiten Projekt Einfügen Extras Online Fenster Hilfe
Bauste... Datent... Resou...
0001 Vorschub_hinten
0002
Z: 1, Sp: 1 ONLINE ÜB LESEN
M-PLC - sfc1.pro - [Transition SchereU (ST)]
Datei Bearbeiten Projekt Einfügen Extras Online Fenster Hilfe
Bauste... Datent... Resou...
0001 Schere_unten
0002
Z: 1, Sp: 1 ONLINE ÜB LESEN
M-PLC - sfc1.pro - [Transition SchereO (ST)]
Datei Bearbeiten Projekt Einfügen Extras Online Fenster Hilfe
Bauste... Datent... Resou...
0001 Schere_oben
0002
Z: 1, Sp: 1 ONLINE ÜB LESEN
M-PLC - sfc1.pro - [Transition Grundstell (ST)]
Datei Bearbeiten Projekt Einfügen Extras Online Fenster Hilfe
Bauste... Datent... Resou...
0001 Schere_oben AND Vorschub_hinten
0002
Z: 1, Sp: 1 ONLINE ÜB LESEN


Init
R
Vorschub_vor
R
Vorschub_zur
R
Schere_ab
R
Schere_auf
Automatik
VorschubV
N
Vorschub_vor
VorschubV
VorschubZ
N
Vorschub_zur
SchereAb
N
Schere_ab
VorschubH
SchereU
VorschubSt
SchereAuf
N
Schere_auf
SchereO
SchereSt
Grundstell
Init

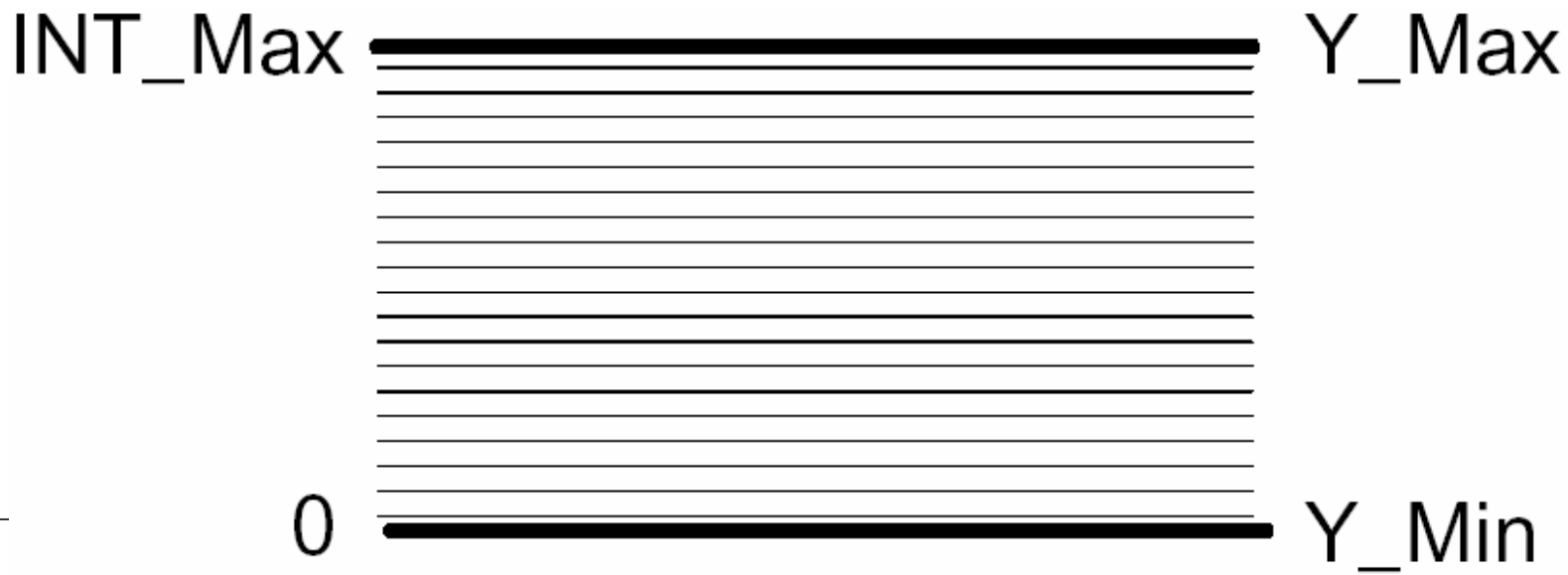
Beispiele für den Einsatz der IEC61131-3

Das Skalierungsproblem

AD-Wandler liefern als Integergrößen als Messwert. DA-Wandler verlangen nach Integerwerten als Eingangsgrößen. Intern wird bei Reglern natürlich mit Floatingpoint-Werten gerechnet. Hieraus ergibt sich die Notwendigkeit, sowohl der den Input- als auch bei den Outputgrößen eine Skalierung vorzunehmen, so dass intern möglichst mit Problemgrößen weitergerechnet werden kann.

Skalierung bei Unsigned - Größen

Viele Wandler arbeiten mit Unsigned – Größen. Bei einem 16-Bit-Wandler z.B. kann der Integerwert somit von 0 bis 65535 variieren. Für die Skalierung werden nun die Werte Y_Min für das untere Ende des Wertebereiches und Y_Max für das obere Ende des Wertebereiches vorgegeben. Es soll also, wie in Bild 1.6.1 dargestellt, Y_Min auf den Integerwert 0 und Y_Max auf den Integerwert INT_Max abgebildet werden.



Für die analoge Eingabe wird nach Gl(1.6.1) verfahren,

$$x = m\,y + b$$

für die Ausgabe nach Gl.(1.6.2).

$$y = \frac{x - b}{m}$$

In beiden Fällen berechnen sich die Größen m und b nach den Beziehungen:

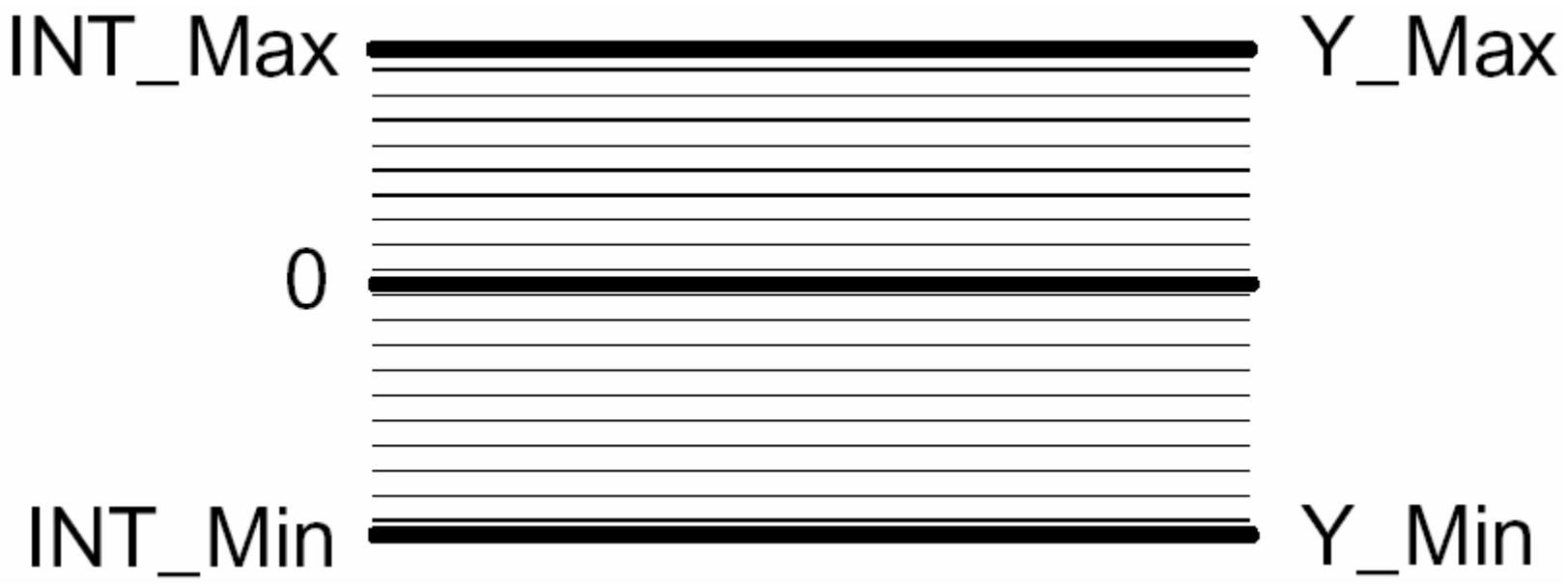
$$m = \frac{INT\_Max}{Y\_Max - Y\_Min}$$

$$y(x = 0) = \frac{-b}{m} \stackrel{!}{=} Y\_Min$$

$$\rightarrow \quad b = -m\,Y\_Min$$

Skalierung bei Signed - Größen

Bei der Verwendung von Signed – Größen erstreckt sich der Wertebereich von INT_Min bis INT_Max. Für einen 16-Bit-Wandler wäre dies der Bereich von INT_Min = –32767 bis INT_Max = 32766. Nun wird, wie in Bild 1.6.2 dargestellt, Y_Min auf INT_MIN und Y_Max auf INT_Max abgebildet.



Für die analoge Eingabe wird auch hier nach Gl(1.6.1) verfahren,

$$x = m\,y + b$$

für die Ausgabe nach Gl.(1.6.2).

$$y = \frac{x - b}{m}$$

$$m = \frac{INT\_Max - INT\_Min}{Y\_Max - Y\_Min}$$

$$y(x = INT\_Min) = \frac{INT\_Min - b}{m} \stackrel{!}{=} Y\_Min$$

$$\rightarrow \quad b = -\,m\,Y\_Min + INT\_MIN$$

Nachfolgende Function Blocks zeigen eine Implementierung für eine analoge Eingabe und eine analoge Ausgabe.

```
FUNCTION_BLOCK AnalogInput
(* created by F. Ley on March 2000 *)
VAR_INPUT
   DINTInput : DINT;      (*DINT value from  ADC *)
   yDefault : LREAL;                    (* secure default value for error situation *)
   Design : BOOL;                (* forces design of conversion *)
   yMin : LREAL := -10.0;        (* minimum value in user coordinates at  lower limit *)
   yMax : LREAL := 10.0;         (* maximum value in user coordinates at upper limit *)
END_VAR
VAR_OUTPUT
   Err : UINT;                                  (* error in scaling *)
   y : LREAL;                                   (* output value *)
END_VAR
VAR
   firstOperation : BOOL := TRUE;
   designOK : BOOL;
   DesignTrigger : R_TRIG;
   factor, offset : LREAL;
   iHelp : DINT;
   vHelp : LREAL;
END_VAR
VAR CONSTANT
   MagicLREAL : LREAL := 65535.0;
   HalfMagicLREAL : LREAL := 32767.0;
END_VAR
```

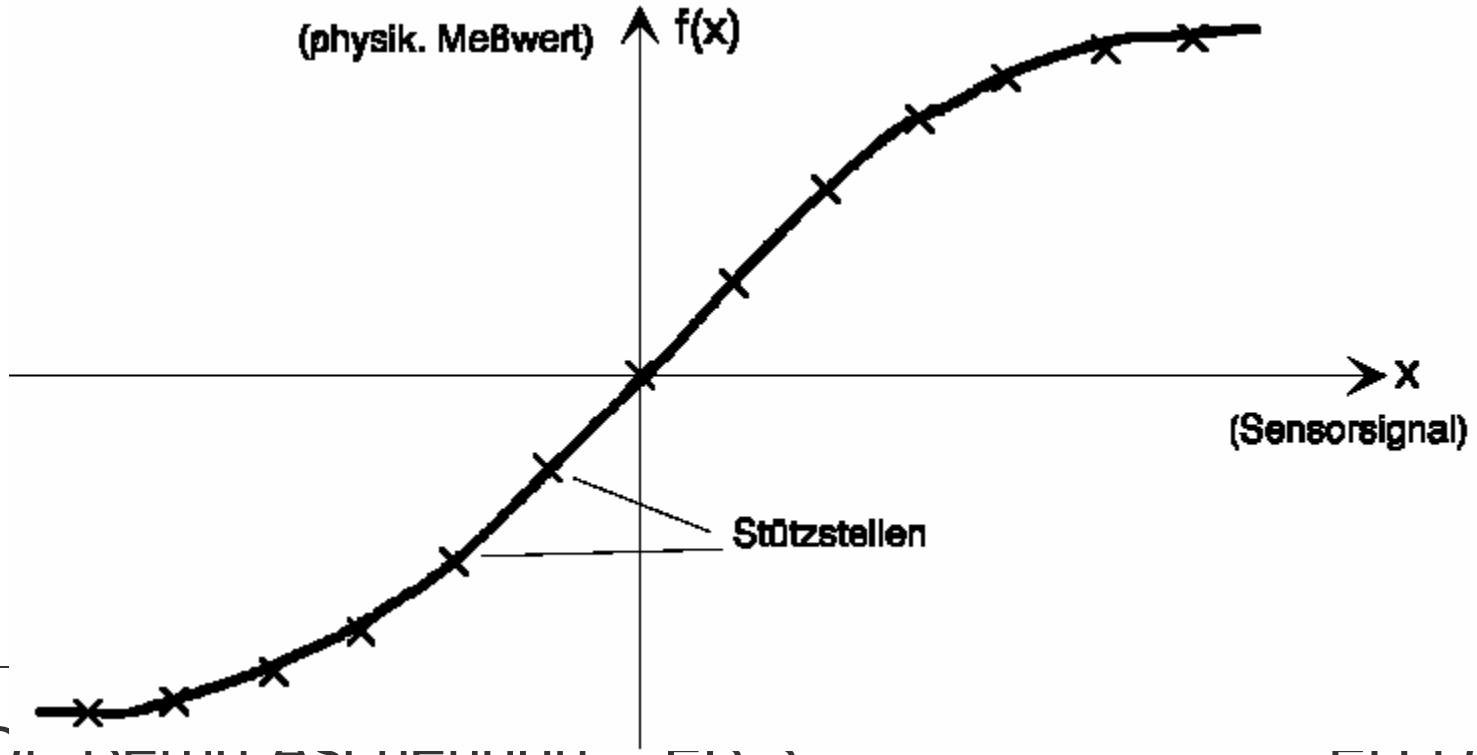
```
 DesignTrigger(S1 := Design);     (* look for rising edge of design input *)


IF firstOperation OR DesignTrigger.Q0 THEN
                  (* this part only for first operation and design input strobed *)
   firstOperation := 0;          (* quit first operation *)
   IF ABS(yMax - yMin) < 1e-30 THEN                    (* scaling ok ? *)
          Err := 1;                        (* not ok, error *)
          designOK := 0;                   (* no operation allowed *)
   ELSE
          Err := 0;                        (* ok, no error *)
          designOK := 1;                   (* operation allowed *)
          factor := MagicLREAL / ( yMax - yMin );      (* compute factor *)
          offset := - factor * yMin - HalfMagicLREAL;  (* compute offset *)
   END_IF;
END_IF;

IF designOK THEN
   y := (DINTInput  - offset) / factor;                (* operate scaling *)
ELSE
   y := yDefault;                (* no operation, output default value *)
END_IF;
END_FUNCTION_BLOCK;
```

```
FUNCTION_BLOCK AnalogOutput
(* created by F. Ley on March 2000 *)
VAR_INPUT
   u : LREAL;                              (* input value *)
   uDefault : DINT;                 (* secure default for error situation *)
   Design : BOOL;           (* forces design of conversion *)
   uMin : LREAL;                    (* minimum value in user coordinates at lower limit *)
   uMax : LREAL;                    (* maximum value in user coordinates at upper limit *)
END_VAR
VAR_OUTPUT
   Err : UINT;                             (* error in scaling *)
   DINTOutput : DINT;       (* DINT value to DAC *)
   convertedOutput : LREAL; (* value to output due to resctrictions *)
END_VAR
VAR
   firstOperation : BOOL := TRUE;
   designOK : BOOL;
   DesignTrigger : R_TRIG;
   factor, offset : LREAL;
   iHelp : DINT;
   vHelp : LREAL;
END VAR
VAR CONSTANT
   MagicLREAL : LREAL := 65535.0;
   HalfMagicLREAL : LREAL := 32767.0;
   HalfMagicDINT : DINT := 32767;
END_VAR
```

```
IF firstOperation OR DesignTrigger.Q0 THEN      (* this part only for first operation and
design input strobed *)
   firstOperation := 0;                                    (* quit first operation *)
   IF ABS(uMax - uMin) < 1e-30 THEN                        (* scaling ok ? *)
          Err := 1;                                        (* not ok, error *)
          designOK := 0;                                   (* no operation allowed *)
   ELSE
          Err := 0;                                        (* ok, no error *)
          designOK := 1;                                   (* operation allowed *)
          factor := MagicLREAL / ( uMax - uMin );          (* compute factor *)
          offset := - factor * uMin - HalfMagicLREAL;      (* compute offset *)
   END_IF;
END_IF;
IF designOK THEN
   vHelp := u * factor + offset;                           (* operate scaling *)
   IF vHelp > HalfMagicLREAL THEN                          (* upper limit ? *)
          DINTOutputV := HalfMagicDINT;                    (* value is on upper limit *)
          convertedOutput := ( HalfMagicLREAL - offset) / factor;
                                                           (* report limited value *)
   ELSIF vHelp < - HalfMagicLREAL THEN                     (* lower limit ? *)
          DINTOutputV := -HalfMagicDINT;                   (* value is on lower limit *)
          convertedOutput := ( - HalfMagicLREAL - offset) / factor;
                                                           (* report limited value *)
   ELSE
          DINTOutputV := LREAL_TO_DINT(vHelp);             (* value is ok, convert it *)
          convertedOutput := u;                            (* report requested value *)
   END_IF;
ELSE
    DINTOutputV := uDefault;                    (* no operation, output default value *)
END_IF;
END_FUNCTION_BLOCK;
```

Linearisierung

Ausgangspunkt einer Linearisierung einer nichtlinearen Kennlinie eines Sensors ist die Kenntnis von Stützstellen als diskrete Wertepaare hinsichtlich des aufgenommenen physikalischen Meßwertes **f** und der übermittelten Signalgröße **x**. Diese können z.B. vom Hersteller in einem Kalibrierungsprotokoll bezogen auf den konkreten Sensor in tabellarischer Form vermerkt sein. Aufgabenstellung der Linearisierung ist es nun, eine Linearisierungsfunktion f(x) zu finden, die die Stützstellen möglichst gut trifft und glatt miteinander verbindet. Mit Hilfe dieser Funktion läßt sich die Nichtlinearität des Sensors wieder beseitigen und - falls es sich um die tatsächlich gemessenen Werte des konkreten Sensors handelt - die Exemplarstreuung des Sensors eliminieren. Die Genauigkeit der Messung hängt dann im wesentlichen nur noch von der Langzeitkonstanz des Sensors und der Genauigkeit der Kalibrierungsmessung ab.



Vorliegendes Verfahren eignet sich für angesetzte Polynome beliebiger Ordnung der Form

$$f(x) = \sum_{i=0}^{m} a_i \, x^i \quad (1.6.9) \quad .$$

Allerdings steigt der Aufwand mit der Ordnung m überproportional. Aus diesem Grund soll das Verfahren für eine Parabelapproximation, also für ein Polynom der Ordnung 2

$$f(x) = a_0 + a_1 \, x + a_2 \, x^2 \quad (1.6.10)$$

eingeführt werden. Ist die Zahl n der Stützstellen $y(x_k) = y_k$ gleich der Anzahl der Koeffizienten a_i des Polynoms, handelt es sich um ein vollständig bestimmtes lineares Gleichungssystem der Ordnung n = m - 1. Ein Polynom 2.Ordnung z.B. benötigt also zur vollständigen Bestimmung 3 Stützstellen. Liegen mehr Stützstellen vor, ist das System überbestimmt. Dies bedeutet, daß die Stützstellen nicht mehr alle exakt durch das Linearisierungs-Polynom getroffen werden können. Es wäre allerdings möglich, die Abweichung aller Stützstellen vom Polynomgraph zu minimieren. Hierzu wird die Summe der Fehlerquadrate

$$[\, f(x_k) - y_k \,]^2$$

gebildet :

$$S = \sum_{k=1}^{n} [\, f(x_k) - y_k \,]^2 \quad (1.6.11)$$

Für das angesetzte Polynom 2.Ordnung nach Gl(1.6.10) ergibt sich die Fehlerquadrat-Summe zu :

$$S(a_0, a_1, a_2) = \sum_{k=1}^{n} [\, a_0 + a_1 \, x_k + a_2 \, x_k^2 - y_k \,]^2 \quad (1.6.12)$$

$$S(a_0, a_1, a_2) = \sum_{k=1}^{n} [\, (a_0 + a_1 \, x_k + a_2 \, x_k^2)^2 - 2 \, yk \, (a_0 + a_1 \, x_k + a_2 \, x_k^2) + y_k^2 \,] \quad (1.6.13)$$

Dieses vom Vektor

$$\underline{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \quad (1.6.14)$$

abhängige Skalarfeld besitzt offenbar dort ein lokales Extremum (hoffentlich ein Minimum!), wo der Gradient dieser Funktion verschwindet :

$$\text{grad } S = \begin{bmatrix} \frac{\partial}{\partial a_0} \\ \frac{\partial}{\partial a_1} \\ \frac{\partial}{\partial a_2} \end{bmatrix} S = \begin{bmatrix} \frac{\partial S}{\partial a_0} \\ \frac{\partial S}{\partial a_1} \\ \frac{\partial S}{\partial a_2} \end{bmatrix} \stackrel{!}{=} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = 0 \quad (1.6.15)$$

Diese dreidimensionale Vektorgleichung führt zu drei Bestimmungsgleichungen hinsichtlich der der Koeffizienten a_i des gesuchten Polynoms :

$$\frac{\partial S}{\partial a_0} = \sum_{k=1}^{n} [\, 2\, (a_0 + a_1\, x_k + a_2\, x_k^2) - 2\, y_k \,] = 0 \quad (1.6.16a)$$

$$\frac{\partial S}{\partial a_1} = \sum_{k=1}^{n} [\, 2\, (a_0 + a_1\, x_k + a_2\, x_k^2)\, x_k - 2\, y_k\, x_k \,] = 0 \quad (1.6.16b)$$

$$\frac{\partial S}{\partial a_2} = \sum_{k=1}^{n} [\, 2\, (a_0 + a_1\, x_k + a_2\, x_k^2)\, x_k^2 - 2\, y_k\, x_k^2 \,] = 0 \quad (1.6.16c)$$

Die Bildung von Einzelsummen und Umstellen liefert :

$$n\,a_0 + a_1 \sum_{k=1}^{n} x_k + a_2 \sum_{k=1}^{n} x_k^2 = \sum_{k=1}^{n} y_k \quad (1.6.17a)$$

$$a_0 \sum_{k=1}^{n} x_k + a_1 \sum_{k=1}^{n} x_k^2 + a_2 \sum_{k=1}^{n} x_k^3 = \sum_{k=1}^{n} y_k\, x_k \quad (1.6.17b)$$

$$a_0 \sum_{k=1}^{n,} x_k^2 + a_1 \sum_{k=1}^{n} x_k^3 + a_2 \sum_{k=1}^{n} x_k^4 = \sum_{k=1}^{n} y_k\, x_k^2 \quad (1.6.17c)$$

Zur Lösung des Gleichungssystems soll eine Matrixdarstellung der Form

$$\underline{\underline{A}}\ \underline{a} = \underline{y} \quad (1.6.18)$$

verwendet werden. Das Gleichungssystem entspricht dann der Matrixgleichung

$$\begin{bmatrix} n & \sum_{k=1}^{n} x_k & \sum_{k=1}^{n} x_k^2 \\ \sum_{k=1}^{n} x_k & \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k^3 \\ \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k^3 & \sum_{k=1}^{n} x_k^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{n} y_k \\ \sum_{k=1}^{n} y_k\ x_k \\ \sum_{k=1}^{n} y_k\ x_k^2 \end{bmatrix} \quad (1.6.19) \quad .$$

Auflösen nach dem Koeffizientenvektor liefert :

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} n & \sum_{k=1}^{n} x_k & \sum_{k=1}^{n} x_k^2 \\ \sum_{k=1}^{n} x_k & \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k^3 \\ \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k^3 & \sum_{k=1}^{n} x_k^4 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{k=1}^{n} y_k \\ \sum_{k=1}^{n} y_k \, x_k \\ \sum_{k=1}^{n} y_k \, x_k^2 \end{bmatrix} \qquad (1.6.20)$$

Es lässt sich leicht zeigen, dass für höhere Polynomordnung m > 2 die folgende Gleichung gilt:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \dots \\ a_m \end{bmatrix} = \begin{bmatrix} n & \sum_{k=1}^{n} x_k & \sum_{k=1}^{n} x_k^2 & \dots & \sum_{k=1}^{n} x_k^m \\ \sum_{k=1}^{n} x_k & \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k^3 & \dots & \sum_{k=1}^{n} x_k^{m+1} \\ \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k^3 & \sum_{k=1}^{n} x_k^4 & \dots & \sum_{k=1}^{n} x_k^{m+2} \\ \dots & \dots & \dots & \dots & \dots \\ \sum_{k=1}^{n} x_k^m & \sum_{k=1}^{n} x_k^{m+1} & \sum_{k=1}^{n} x_k^{m+2} & \dots & \sum_{k=1}^{n} x_k^{2m} \end{bmatrix}^{-1} \begin{bmatrix} \sum_{k=1}^{n} y_k \\ \sum_{k=1}^{n} y_k\, x_k \\ \sum_{k=1}^{n} y_k\, x_k^2 \\ \dots \\ \sum_{k=1}^{n} y_k\, x_k^m \end{bmatrix}$$

(1.6.21)

Hinweis : Falls die Kalibrierung so erfolgt, daß immer die gleichen Sensorausgangs-Signale x_k als Stützstellen erscheinen, braucht die Matrix-Inversion nicht erneut durchgeführt werden.