PROGRAMMABLE CONTROLLERS

Jerzy Kasprzyk

Lecture: IEC 61131-3: Programming Languages

Contents

1. Standard IEC 1131: Programmable Controllers

since 1998 IEC 61131, EN 61131
PC – Programmable Controllers
PADT – Programming And Debugging Tools
TE – Test Equipment
MMI – Man-Machine Interface

IEC 61131 calls 10 other standards (*IEC 50, IEC 559, IEC 617-12, IEC 617-13, IEC 848, ISO/AFNOR, ISO/IEC 646, ISO 8601, ISO 7185, ISO 7498*)

Parts:

1. General Information (2003)
2. Equipment Requirements and Tests (2003)
3. Programming Languages (2003)
4. User Guidelines (Technical Report) (2004)
5. Communications (2003)
6. (reserved)
7. Fuzzy Control Programming (2000)
8. Guidelines for the Application and Implementation of Programming Languages. (Technical Report) (2003)

IEC 61131-3 Programming languages

Textual languages:
- IL **–** Instruction List
- ST – Structured Text

Graphical languages:
- LD **–** Ladder Diagram
- FBD –. Function Block Diagram

+ SFC **–** Sequential Function Chart

The elements of PLC programming languages:

- Data types
- Program organization units (POU): Functions, Function Blocks (FB), Programs
- Sequential Function Chart elements
- Configuration elements: global variables, resources, tasks, access paths

2. Common elements

- Delimiters + - $ = := # ; ( ) * space
- Keywords

  Declarations: *PROGRAM, FUNCTION, VAR_INPUT, END_PROGRAM*

  Elementary data types: *BOOL, INT, REAL, TIME*

  Standard Functions and FBs: *AND, ADD, MOVE, SHL, TON, CTU* etc. and names of input/output parameters

  IL operators, ST instructions and operators, SFC elements, *EN/ENO*
- Literals (e.g. True, 15, 16#ff, t#5s)
- Identifiers:

  Configurations, global variables, resources, tasks, access paths,

  Functions, Function Blocks, Programs,

  Variables and Data types,

  Steps, transitions and actions in SFC,

  Jump and network labels,

  Constants in enumerated data types
- Comments *(* *)*

3. Literals

Table 1. Numeric literals

Feature description	Examples
Integer literals	*-12 0 123_456 +986*
Real literals	*-12.0 0.0 0.456 3.14159_26*
Real literals with exponents	*-1.34E-12* or *-1.34e-12 1.234E6* or *1.234e6*
Base 2 literals	*2#1111_1111 (255) 2#1110_0000 (240)*
Base 8 literals	*8#377 (255) 8#340 (240)*
Base 16 literals	*16#FF* or *16#ff (255) 16#E0* or *16#e0 (240)*
Boolean zero and one	*0 1*
Boolean *FALSE* and *TRUE*	*FALSE TRUE*

Table 2. Character string literal feature

No.	Example	Explanation
1	’’	Empty string (length zero)
2	*’A’*	String of length one containing the single character *A*
3	’ ’	String of length one containing the *space* character
4	*’$”*	String of length one containing the *single quote* character
5	*’RL’* or *’$0D$0A’*	String of length two containing *CR* and *LF* characters
6	*’$$1.00’*	String of length five which should be printed as *$1.00*

Table 3. Duration literal features

Feature description	Examples
Duration literals without underlines – short prefix	*T#14ms T#14.7s T#14.7m T#14.7h t#14.7d t#5d14h12m18s3.5ms*
Duration literals without underlines – long prefix	*TIME#14ms time#14.7s*
Duration literals with underlines – short prefix	*T#14ms T#14.7s T#14.7m T#14.7h t#25h_15m t#5d_14h_12m_18s_3.5ms*
Duration literals with underlines – long prefix	*TIME#25h_15m time#5d_14h_12m_18s_3.5ms*

Table 4. Date and time of day literals

No.	Feature description	Prefix keyword
1	Date literals	*DATE# D#*
2	Time of day literals	*TIME_OF_DAY# TOD#*
3	Date and time literals	*DATE_AND_TIME# DT#*

Table 5. Examples of date and time of day literals

Long prefix notation	Short prefix notation
DATE#1984-06-25 *date#1984-06-25*	*D#1984-06-25* *d#1984-06-25*
TIME_OF_DAY#15:36:55.36 *time_of_day#15:36:55.36*	*TOD#15:36:55.36* *tod#15:36:55.36*
DATE_AND_TIME#1984-06-25-15:36:55.36 *date_and_time#1984-06-25-15:36:55.36*	*DT#1984-06-25-15:36:55.36* *dt#1984-06-25-15:36:55.36*

4. Data types

Table 6. Elementary data types

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

Table 7. Generic data types

<table>
<tr><td colspan="5">ANY</td></tr>
<tr><td>ANY_BIT</td><td colspan="3">ANY_NUM</td><td>ANY_DATE</td><td rowspan="3">TIME
STRING
and derived data types</td></tr>
<tr><td rowspan="2">BOOL
BYTE
WORD
DWORD
LWORD</td><td colspan="2">ANY_INT</td><td>ANY_REAL</td><td rowspan="2">DATE
TIME_OF_DAY
DATE_AND_TIME</td></tr>
<tr><td>INT
SINT
DINT
LINT</td><td>UINT
USINT
UDINT
ULINT</td><td>REAL
LREAL</td></tr>
</table>

Table 8. Data type declaration feature

No.	Feature/textual example
1	Direct derivation from elementary types, e.g.: *TYPE FLOATING : REAL; END_TYPE*
2	Enumerated data type, e.g.: *TYPE* *ANALOG_SIGNAL_TYPE : (SINGLE_ENDED, DIFFERENTIAL); END_TYPE*
3	Subrange data type, e.g.: *TYPE ANALOG_DATA : INT (-4095..4095); END_TYPE*
4	Array data type, e.g.: *TYPE* *ANALOG_16_INPUT: ARRAY [1..16] OF ANALOG_DATA;* *ANALOG_ARRAY: ARRAY[1..4,1..16] OF ANALOG_DATA;* *END_TYPE*
5	Structured data type, e.g.: *TYPE* *ANALOG_CHANNEL_CONFIG :* *STRUCT* *RANGE : ANALOG_RANGE;* *MIN_SCALE : ANALOG_DATA;* *MAX_SCALE : ANALOG_DATA;* *END_STRUCT;* *ANALOG_16_INPUT_CONFIG :* *STRUCT* *SIGNAL_TYPE : ANALOG_SIGNAL_TYPE;* *FILTER_PARAMETER : SINT (0..99);* *CHANNEL : ARRAY [1..16] OF ANALOG_CHANNEL_CONFIG;* *END_STRUCT;* *END_TYPE*

Table 9. Default initial values

Data type(s)	Initial value
BOOL, SINT, INT, DINT, LINT	*0*
USINT, UINT, UDINT, ULINT	*0*
BYTE, WORD, DWORD, LWORD	*0*
REAL, LREAL	*0.0*
TIME	*T#0S*
DATE	*D#0001-01-01*
TIME_OF_DAY	*TOD#00:00:00*
DATE_AND_TIME	*DT#0001-01-01-00:00:00*
STRING	“ (the empty string)

Table 10. Data type initial value declaration features

No.	Feature/textual example
1	*TYPE NEW_BOOL : BOOL := 1; END_TYPE*
2	*TYPE ANALOG_RANGE :* *(BIPOLAR_10V, (* -10 to +10 VDC *)* *UNIPOLAR_1_10V, (* +1 to+10 VDC *)* *UNIPOLAR_0_10V, (* 0 to+10 VDC *)* *UNIPOLAR_1_5V, (* +1 to+10 VDC *)* *UNIPOLAR_0_5V, (* 0 to+10 VDC *)* *UNIPOLAR_4_20MA, (* +4 to+20 mADC *)* *UNIPOLAR_0_20MA, (* 0 to+20 mADC *)* *) := UNIPOLAR_1_5V ;* *END_TYPE*
3	*TYPE ANALOG_DATAZ : INT (-4095..4095) := 0 ; END_TYPE*
4	*TYPE* *ANALOG_16_INPUT: ARRAY [1..16] OF ANALOG_DATA := 8(-4095), 8(4095);* *END_TYPE*

5. Variables

Table 11. Variable declaration keywords

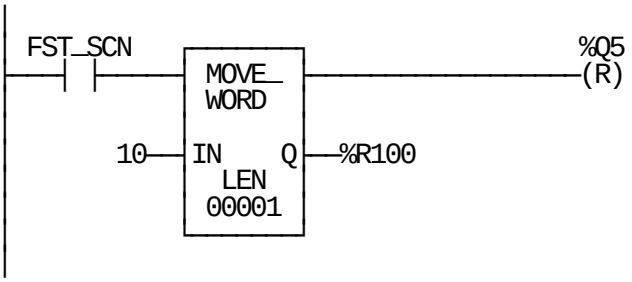
Keyword	Description
VAR	Internal to POU variables
VAR_INPUT	Externally supplied, not modifiable within POU
VAR_OUTPUT	Supplied by POU to external entities
VAR_IN_OUT	Supplied by external entities, can be modified within POU
VAR_EXTERNAL	Supplied by configuration via *VAR_GLOBAL*, can be modified within POU
VAR_GLOBAL	Global variable declaration
VAR_ACCESS	Access path declaration

Table 12. Attributes (qualifiers) of variables

Attribute	Meaning
AT	Location assignment
RETAIN	Retentive variable (battery back-up)
CONSTANT	Constant variable (cannot be modified)
R_EDGE	Rising edge
F_EDGE	Falling edge
READ_ONLY	Write-protected
READ_WRITE	Can be read and written to

```
(*↓ variable type       ↓ attribute *)
VAR_OUTPUT              RETAIN

(*↓ variable name       ↓ data type    ↓ initial value *)
  My_Variable           : DINT   :=    100;

END_VAR
```

Example 1. Variable declaration



Example 2. Initial values declaration in GE Fanuc

Table 13. Use of attributes (qualifiers) for variable types

Variable type	***RETAIN***	***CONSTANT***	***R_EDGE F_EDGE***	***READ_ONLY READ_WRITE***
VAR	+	+	-	-
VAR_INPUT	-	-	+	-
VAR_OUTPUT	+	-	-	-
VAR_IN_OUT	-	-	-	-
VAR_EXTERNAL	-	-	-	-
VAR_GLOBAL	+	+	-	-
VAR_ACCESS	-	-	-	+

```
VAR CONSTANT              (* constant *)
  PI : REAL := 3.1415925;
  One1     :  INT   := 1;
  One2     :  DINT :=1;
END_VAR
VAR_OUTPUT RETAIN      (* output retentive variable *)
  Q : WORD;
END_VAR
VAR_INPUT                 (* input variables *)
  IN1 : BOOL  R_EDGE;  (* input variable – rising edge *)
  IN2 : BOOL  F_EDGE;  (* input variable – falling edge *)
END_VAR
VAR_ACCESS                (* access paths *)
  CSX : P1.Z : REAL  READ_ONLY;
END_VAR
```

Example 3. Attributes declaration for variables

Single-element variables

%QX75 or *%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 addressing

Example 4. Directly represented variables

Table 14. Location and size prefix features for directly represented variables

No.	Prefix	Meaning
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*	Double word (32 bits) size
9	*L*	Long word (64 bits) size

```
VAR
  (* Directly represented variables *)
  AT %IW1   : INT;          (* Input word location 1,  variable of type INT *)
  AT %QD8  : DINT;          (* Output double word location 8,  variable of type DINT *)

  (* located variables with symbolic representation *)
  XXX   AT %QW3  : INT;     (* Output word location 3, variable of type INT *)
  YYY   AT %QX16 : BOOL;  (* Output bit location 16 *)

  (* unlocated variables with symbolic representation *)
  A,B,C : INT;   (* 3 variables of type INT in 3 consecutive words in memory *)
  DDD : DINT  (* a variables of type DINT in 2 consecutive words in memory *)
  RRR : REAL;  (*a variables of type REAL in 2 consecutive words in memory *)
END_VAR
...
LD %IW1  (* using of directly represented variable *)
ST XXX
LD %IW1   (* load a variable of type INT *)
ST DDD    (* store of CR in a variable of type DINT – error!!!*)
…
```

Example 5. Declaration and using of single variables

Multi-element variables

```
VAR
  MY_INPUTS  AT %IW1  : ARRAY[0..3] OF INT;          (* array – 1 dimension*)
  INPUT_TAB : ANALOG_ARRAY;                          (* array – 2 dimensions *)
  MODULE_CONFIG : ANALOG_16_INPUT_CONFIG;   (* structure *)
END_VAR
```

Example 6. Multi-element variable declaration

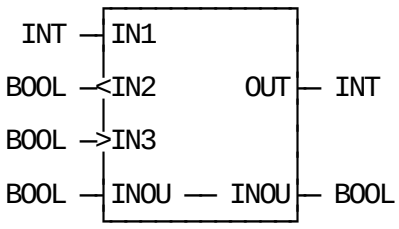
```
INPUT_TAB[4,%MB6]
MODULE_CONFIG.SIGNAL_TYPE:=SINGLE_ENDED;
MODULE_CONFIG.CHANNEL[5].RANGE:=BIPOLAR_10V;
```

Example 7. Element addressing for multi-element variables

Assignment of initial values at the start of the program

- Battery-backed with *RETAIN* (*warm restart*)
- Initial value from declaration (*cold restart*)
- Initial value from data type (*cold restart*)

Graphical declaration of input and output variables




```
VAR_INPUT
  IN1  : INT;
  IN2  : BOOL F_EDGE;
  IN3  : BOOL R_EDGE;
END_VAR
VAR_OUTPUT
  OUT  : INT;
END_VAR
VAR_IN_OUT
  INOU : BOOL;
END_VAR
```

Example 8. Graphical and textual declaration of input and output variables (external interface)

6. Program organization units (POU)

Elements of POU:

- POU's type (*PROGRAM, FUNCTION_BLOCK, FUNCTION*) and name
- Variable declaration (input, output, internal)
- POU's body
- *END_PROGRAM, END_ FUNCTION_BLOCK, END_ FUNCTION*

Table 15. Graphical negation of Boolean signals

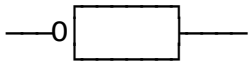
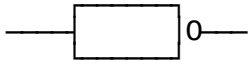
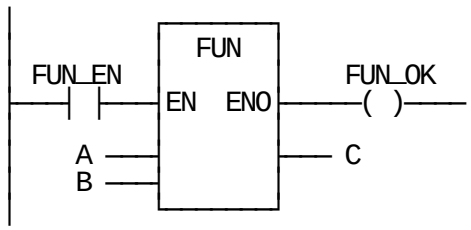
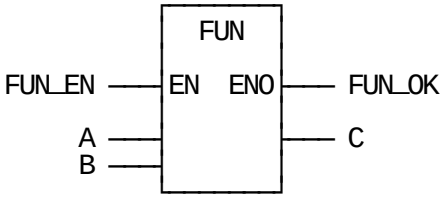
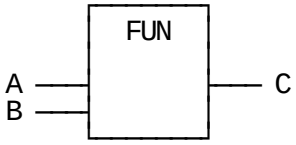
No.	Feature	Representation
1	Negated input	
2	Negated output	

Table 16. Use of *EN* input and *ENO* output

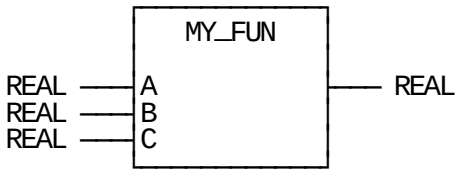
No.	Feature	Example
1	Use of *EN* and *ENO* – **required** for LD	
2	Use of *EN* and *ENO* – **optional** for FBD	
3	FBD without *EN /ENO*.	

Function declaration

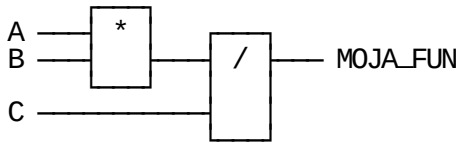
```
FUNCTION MY_FUN : REAL      (* Function name and type *)
  VAR INPUT                 (* Input parameters *)
    A, B : REAL;
    C : REAL:=1.0 ;         (* Initial value *)
  END_VAR
  MY_FUN:=A*B/C ;           (* Output assignment *)
END_FUNCTION
```

Example 9. Function declaration in ST

```
FUNCTION
(* Function name and external interface *)
```




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




```
END_FUNCTION
```

Example 10. Function declaration in FBD.

Typing, overloading and type conversion

Table 17. Typed and overloaded functions

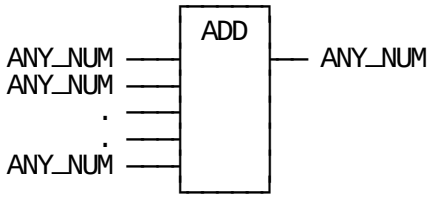
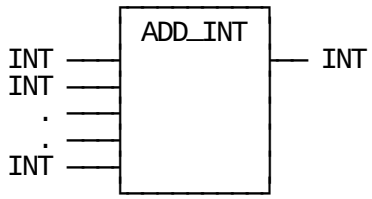
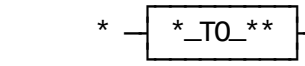
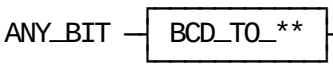
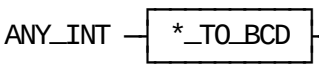
No.	Feature	Example
1	Overloaded function	
2	Typed function	

Table 18. Type conversion functions

No.	Graphical form	Usage example in ST
1		*A:=REAL_TO_INT(B);*
2		*A:=TRUNC(B);*
3		*A:=BCD_TO_INT(C);*
4		*C:=INT_TO_BCD(A);*

```
(* Variable declaration for the 1st example *)
VAR
  A :INT;
  B :REAL;
  C :WORD;
END_VAR

(* Example 1 in IL language *)
LD B
REAL_TO_INT
ST A
```

Table 19. Examples of explicit type conversion with overloaded functions

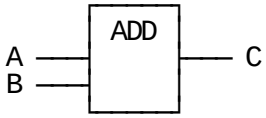
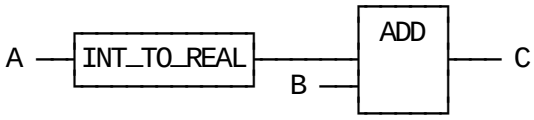
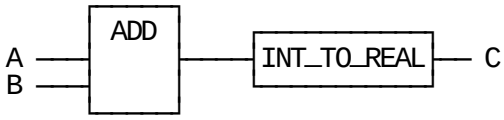
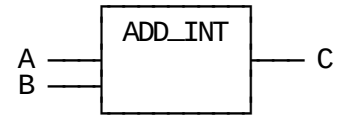
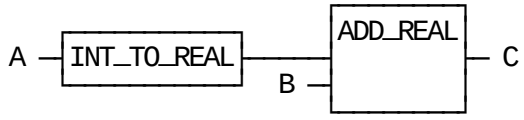
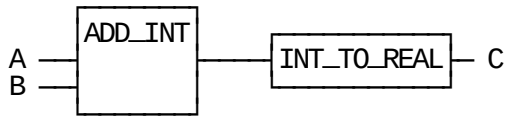
No.	Type declaration (ST)	Examples in FBD and ST
1	*VAR* *A : INT ;* *B : INT ;* *C : INT ;* *END_VAR*	 C:= A + B;
2	*VAR* *A : INT ;* *B : REAL ;* *C : REAL ;* *END_VAR*	 *C:=INT_TO_REAL(A) + B;*
3	*VAR* *A : INT ;* *B : INT ;* *C : REAL ;* *END_VAR*	 *C:=INT_TO_REAL(A + B);*

Table 20. Examples of explicit type conversion with typed functions

No.	Type declaration (ST)	Examples in FBD and ST
1	*VAR* *A : INT ;* *B : INT ;* *C : INT ;* *END_VAR*	 C:=ADD_INT(A, B);
2	*VAR* *A : INT ;* *B : REAL ;* *C : REAL ;* *END_VAR*	 *C:=ADD_REAL(INT_TO_REAL(A), B);*
3	*VAR* *A : INT ;* *B : INT ;* *C : REAL ;* *END_VAR*	 *C:=INT_TO_REAL(ADD_INT(A, B));*

Standard functions

Table 21. Standard functions of one numeric variable

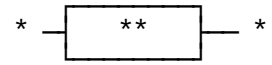
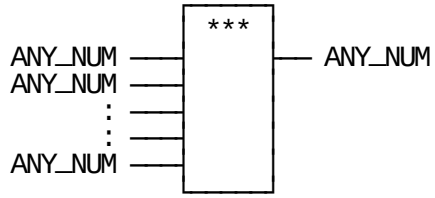
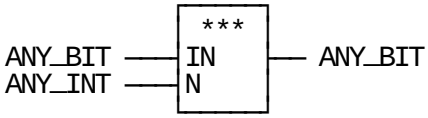
Graphical form			**Usage example**
 * – I/O type ** – Function name			*A:=SIN(B); (* ST *)* *LD B (* IL *)* *SIN* *ST A*
No.	**Function name**	**I/O type**	**Description**
General functions			
1	*ABS*	*ANY_NUM*	Absolute value
2	*SQRT*	*ANY_REAL*	square root
Logarithmic functions			
3	*LN*	*ANY_REAL*	Natural logarithm
4	*LOG*	*ANY_REAL*	Logarithm base 10
5	*EXP*	*ANY_REAL*	Natural exponential
Trigonometric functions			
6	*SIN*	*ANY_REAL*	Sine of input in radians
7	*COS*	*ANY_REAL*	Cosine of input in radians
8	*TAN*	*ANY_REAL*	Tangent of input in radians
9	*ASIN*	*ANY_REAL*	Principal arc sine
10	*ACOS*	*ANY_REAL*	Principal arc cosine
11	*ATAN*	*ANY_REAL*	Principal arc tangent

Table 22. Standard arithmetic functions

Graphical form	Usage example
 *** – Name or symbol	*A:=ADD(B,C,D); (* ST *)* or *A:=B+C+D;* *LD B (* IL *)* *ADD C* *ADD D* *ST A*

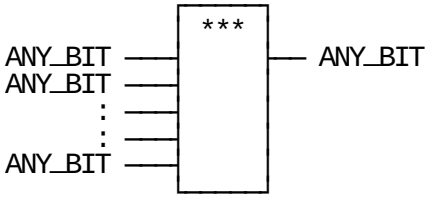
No.	Name	Symbol	Description
Extensible arithmetic functions			
12	*ADD*	+	$OUT := IN_1+IN_2+\ldots+IN_n$
13	*MUL*	*	$OUT := IN_1*IN_2*\ldots*IN_n$
Non-extensible arithmetic functions			
14	*SUB*	–	$OUT := IN_1 - IN_2$
15	*DIV*	/	$OUT := IN_1 / IN_2$
16	*MOD*		$OUT := IN_1$ *modulo* IN_2
17	*EXPT*	**	$OUT := IN_1^{IN_2}$
18	*MOVE*	:=	$OUT := IN$

Table 23. Standard bit shift functions

Graphical form	Usage example
 *** – Function Name	*A:=SHL(IN:=B, N:=5); (* ST *)* *LD B (* IL *)* *SHL 5* *ST A*

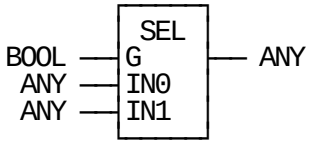
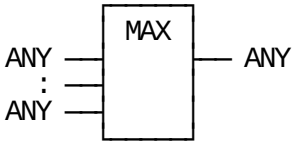
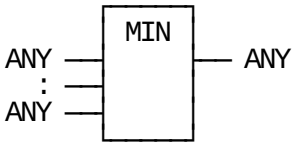
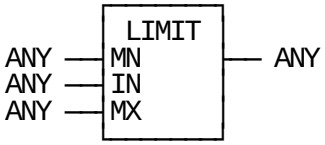
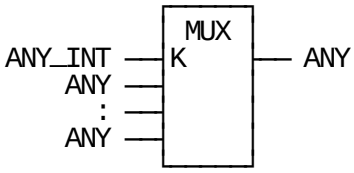
No.	Name	Description
1	*SHL*	*OUT:= IN* left-shifted by *N* bits, zero-filled on right
2	*SHR*	*OUT:= IN* right-shifted by *N* bits, zero-filled on left
3	*ROR*	*OUT:= IN* right-rotated by *N* bits, circular
4	*ROL*	*OUT:= IN* left-rotated by *N* bits, circular

Table 24. Standard bitwise Boolean functions

Graphical form	Usage example
 *** – Function Name or symbol	*A:=AND(B,C,D); (* ST *)* or *A:=B&C&D;* *LD B (* IL *)* *AND C* *AND D* *ST A*

No.	Name	Symbol	Description
5	*AND*	*&*	$OUT:=IN_1 \& IN_2 \& \ldots \& IN_n$
6	*OR*	*>=1*	$OUT:=IN_1$ *OR* IN_2 *OR …* IN_n
7	*XOR*	*=2k+1*	$OUT:=IN_1$ *XOR* IN_2 *XOR …* IN_n
8	*NOT*		$OUT:=NOT\ IN_1$

Table 25. Standard selection functions

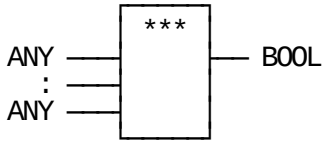
No.	Graphical form	Explanation / example in ST
1		Binary selection: *OUT:=IN$_0$* if *G=0* *OUT:=IN$_1$* if *G=1* Example: *A:=SEL(G:=0, IN0:=X, IN1:=255);* Result: *A:=X.*
2a		Extensible maximum function: *OUT:=MAX{IN$_1$, IN$_2$, …, IN$_n$}* Example: *A:=MAX(B, C, D);*
2b		Extensible minimum function: *OUT:=MIN{IN$_1$, IN$_2$, …, IN$_n$}* Example: *A:=MIN(B, C, D);*
3		Limiter: *OUT:=MIN{MAX{IN, MN}, MX}* Example: *A:=LIMIT(IN:=B, MN:=0, MX:=255);* Result: *A:=0* if *B < 0*, *A:=255* if *B > 255*, else *A:=B.*
4		Extensible multiplexer: select one of *N* inputs depending on input *K*. Example: *A:=MUX(K:=0, IN0:=B, IN1:=C, IN2:=D);* Result: *A:=B.*

Example of *MUX* in LD:

```
LD 0
MUX B, C, D
ST A
```

(*Run-time error* if *K < 0* or > the number of the remaining inputs)

Table 26. Standard comparison functions

Graphical form	Usage example in ST
 *** – Function Name or symbol	*A := GT(B, C, D);* or *A := (B > C) & (C > D);*

No.	Name	Symbol	Description
5	*GT*	>	*=1* if decreasing sequence: $OUT:=(IN_1 > IN_2) \& (IN_2 > IN_3) \& \ldots \& (IN_{n-1} > IN_n)$
6	*GE*	>=	*=1* if monotonic sequence: $OUT:=(IN_1 >= IN_2) \& (IN_2 >= IN_3) \& \ldots \& (IN_{n-1} >= IN_n)$
7	*EQ*	=	*=1* if equality: $OUT:=(IN_1 = IN_2) \& (IN_2 = IN_3) \& \ldots \& (IN_{n-1} = IN_n)$
8	*LE*	<=	*=1* if monotonic sequence: $OUT:=(IN_1 <= IN_2) \& (IN_2 <= IN_3) \& \ldots \& (IN_{n-1} <= IN_n)$
9	*LT*	<	*=1* if increasing sequence: $OUT:=(IN_1 < IN_2) \& (IN_2 < IN_3) \& \ldots \& (IN_{n-1} < IN_n)$
10	*NE*	<>	*=1* if inequality (non-extensible): $OUT:=(IN_1 <> IN_2) \& (IN_2 <> IN_3) \& \ldots \& (IN_{n-1} <> IN_n)$

Example in IL (*A* is of type *BOOL*, *B,C,D* are of type *ANY*, but the same):

```
LD B
GT C      (*CR := Boolean result of comparison (B > C) *)
AND(      (* CR is stored *)
LD C
GT D      (* CR := Boolean result of comparison (C > D) *)
)         (* End of nesting – CR := CR AND CR stored *)
ST A
```

Table 27. Standard character string functions

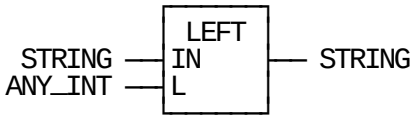
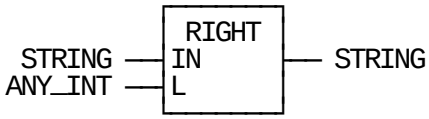
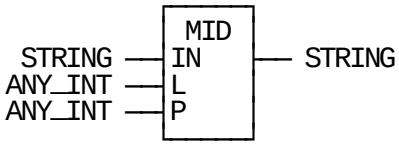
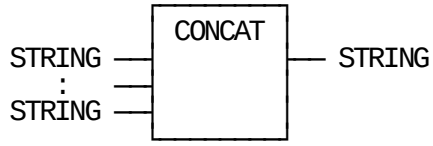
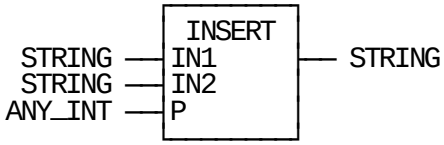
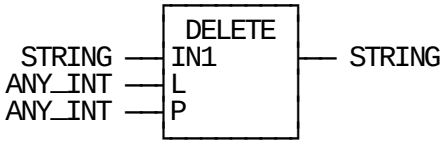
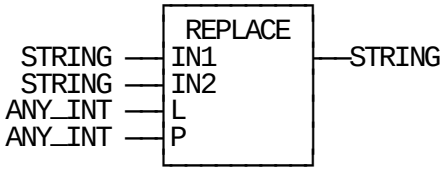
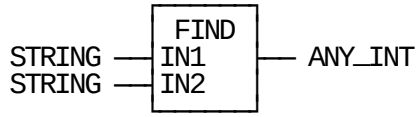
No.	Graphical form	Explanation / example in ST
1		String length function, e.g.: *A:=LEN('ASTRING');* Result: *A:=7.*
2		Leftmost *L* characters of *IN*, e.g.: *A:=LEFT(IN:='ASTR', L:=3)* Result: *A:='AST'.*
3		Rightmost *L* characters of *IN*, e.g.: *A:=RIGHT(IN:='ASTR', L:=3);* Result: *A:='STR'.*
4		*L* characters of *IN* beginning at the *P*-th, e.g.: *A:=MID(IN:='ASTR', L:=2, P:=2);* Result: *A:='ST'.*
5		Extensible concatenation, e.g.: *A:=CONCAT('AB','CD','E');* Result: *A:='ABCDE'.*
6		Insert *IN2* into *IN1* after the *P*-th position, e.g.: *A:=INSERT(IN1:='ABC', IN2:='XY',P:=2);* Result: *A:='ABXYC'.*
7		Delete *L* characters of *IN* beginning at the *P*-th character position, e.g.: *A:=DELETE(IN:='ABXYC', L:=2, P:=3);* Result: *A:='ABC'.*
8		Replace *L* characters of *IN1* by*IN2* starting at the *P*-th character position, e.g.: *A:=INSERT(IN1:='ABCD',IN2:='X',L:=2,P:=2);* Result: *A:='AXD'.*
9		Find the character position of the beginning of the first occurrence of *IN2* in *IN1*, e.g.: *A:=FIND(IN1:='ABCBC', IN2:='BC');* Result *A:=2.*

Table 28. Standard functions of time data type

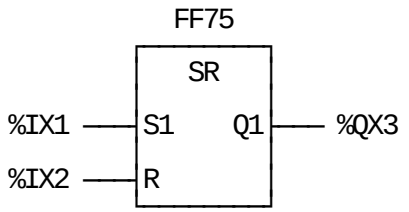
No.	Numeric and concatenation functions				
	Name	**Symbol**	**IN1**	**IN2**	**OUT**
1 2 3	*ADD*	+	*TIME* *TIME_OF_DAY* *DATE_AND_TIME*	*TIME* *TIME* *TIME*	*TIME* *TIME_OF_DAY* *DATE_AND_TIME*
4 5 6 7 8 9	*SUB*	–	*TIME* *DATE* *TIME_OF_DAY* *TIME_OF_DAY* *DATE_AND_TIME* *DATE_AND_TIME*	*TIME* *DATE* *TIME* *TIME_OF_DAY* *TIME* *DATE_AND_TIME*	*TIME* *TIME* *TIME_OF_DAY* *TIME* *DATE_AND_TIME* *TIME*
10 11	*MUL* *DIV*	* /	*TIME* *TIME*	*ANY_NUM* *ANY_NUM*	*TIME* *TIME*
12	*CONCAT*		*DATE*	*TIME_OF_DAY*	*DATE_AND_TIME*
Type conversion functions					
13 14	*DATE_AND_TIME_TO_TIME_OF_DAY* *DATE_AND_TIME_TO_DATE*				

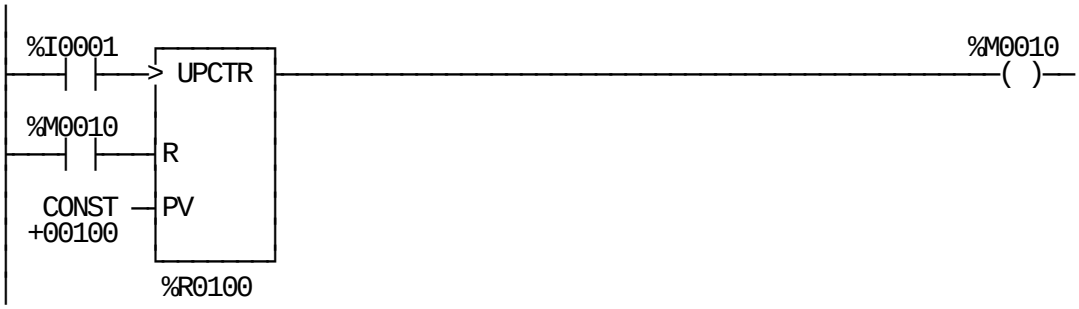
Function block (FB)

```
VAR
 (* ↓ Name              ↓ Data type or FB type *)
    Alpha       :      BOOL;   (* variable declaration – instance of data type BOOL *)
    Counter1 :         CTU;    (* instance of FB type CTU*)
    Counter2 :         CTU;    (* the other instance of FB type CTU *)
END_VAR
```

Example 11. Variable declaration and FB instantiation

Table 29. Instantiation and invocation of FB

Graphical form (FBD)	Textual form (ST)
	*(* Instantiation *)* *VAR FF75: SR; END_VAR* *(* Invocation *)* *FF75(S1:=%IX1, R:=%IX2);* *(* Output assignment *)* *%QX3:= FF75.Q1;*



Example 12. Usage of FB UPCTR in LD (GE Fanuc)

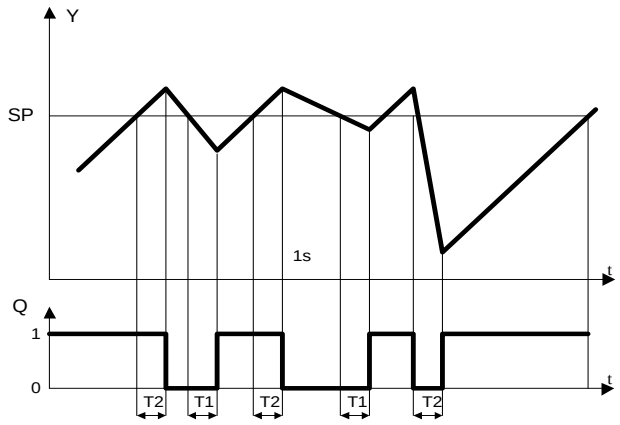
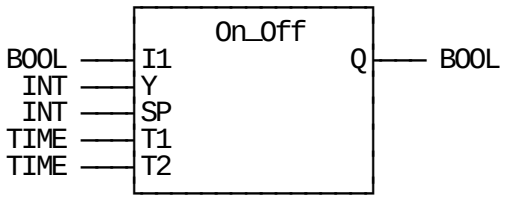


Figure 1. Timing diagrams for on-off controller with time hysteresis

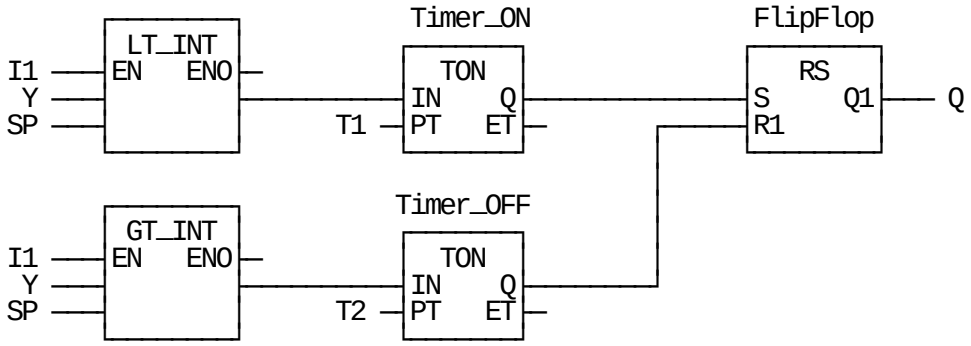
```
FUNCTION_BLOCK On_Off

(* External interface *)
VAR_INPUT
  I1      : BOOL;                 (* Input enables FB, default = 0 *)
  Y       : INT;                  (* Process Variable *)
  SP      : INT;                  (* Set Point *)
  T1      : TIME := t#100ms;      (* On delay, default = 100ms *)
  T2      : TIME := t#100ms;      (* Off delay, default = 100ms *)
END_VAR
VAR_OUTPUT
  Q       : BOOL;                 (* default = 0 *)
END_VAR

VAR                               (* Internal variables *)
    Timer_ON    : TON;            (* Instance names*)
    Timer_OFF   : TON;
    FlipFlop    : SR;
END_VAR

(* FB body *)
IF I1 THEN                        (* Enabling the on-off controller *)

  IF (Y < SP) THEN
     Timer_ON(IN:=I1, PT:=T1);    (* set time delay for ON *)
  END_IF

  IF (Y > SP) THEN
     Timer_OFF(IN:=I1, PT:=T2);   (* set time delay for OFF *)
  END_IF

  FlipFlop(S:=Timer_ON.Q, R1:=Timer_OFF.Q);
  Q:=FlipFlop.Q1;                 (* output assignment *)

END_IF

END_FUNCTION_BLOCK
```

Example 13. FB declaration in ST

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




```
(** FB body **)
```

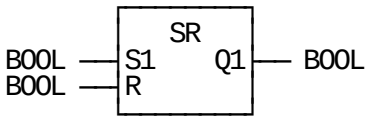
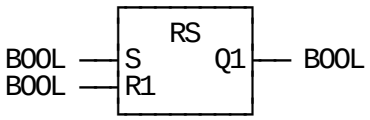
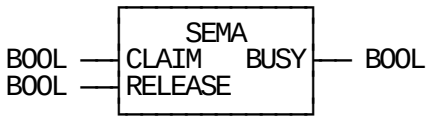



```
END_FUNCTION_BLOCK
```

Example 14. FB declaration in FBD

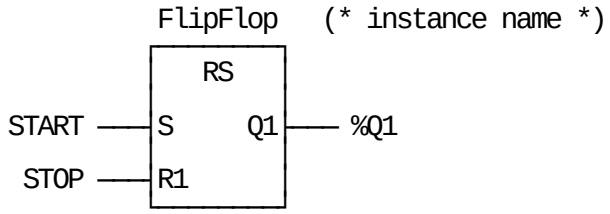
Standard function blocks

Table 30. Standard bistable FBs

No.	Graphical form	Description
1		Flipflop SR *S1* – set (dominant) *R* – reset
2		Flipflop RS *S* – set *R1* – reset (dominant)
3		Semaphore (non-interruptible) *CLAIM* – set semaphore *RELEASE* – release semaphore

in LD: (S) and (R) or (SM) and (RM) coils

(FBD *)*

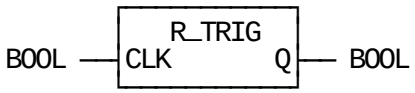


(textual languages *)*

```
VAR
  FlipFlop : RS;                    (* instantiation *)
END_VAR

(* ST *)
  FlipFlop(S:=START, R1:=STOP);     (* invocation *)
  %Q1:=FlipFlop.Q1;                 (* output assignment *)

(* IL *)
  CAL FlipFlop(S:=START, R1:=STOP);  (* invocation *)
  LD FlipFlop.Q1
  ST %Q1                            (* output assignment *)
```

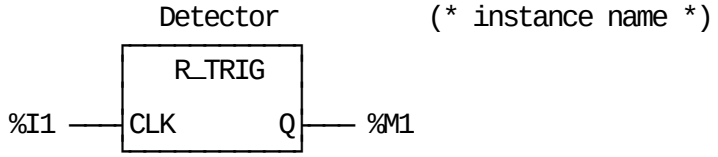
Example 15. Bistable element *RS* in FBD, ST and IL.

Table 31. Standard edge detection FBs

No.	Graphical form	Description
1		Rising edge detector *CLK* – tested input
2		Falling edge detector *CLK* – tested input

in LD: | P | and | N | contacts or (P) and (N) coils

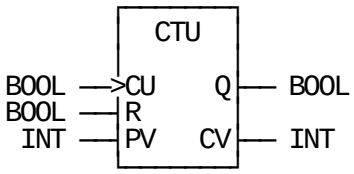
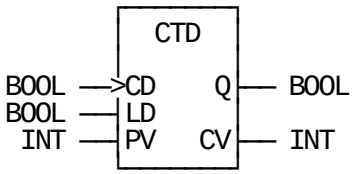
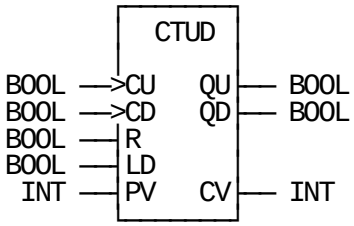
```
(* FBD *)
```




```
(* textual languages *)
VAR
  Detector : R_TRIG;          (* instantiation *)
END_VAR

(* ST *)
 Detector(CLK:=%I1);          (* invocation *)
 %M1:= Detector.Q;            (* output assignment *)

(* IL *)
 CAL Detector (CLK:=%I1);     (* invocation *)
 LD Detector.Q
 ST %M1                       (* output assignment *)
```

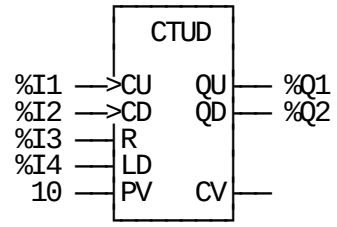
Example 16. Rising edge detection in FBD, ST and IL.

Table 32. Standard counter FBs

No.	Graphical form	Description
1		Up counter *CU* – input (rising edges are counted up) *R* – reset counter (*CV:=0*) *PV* – preset value *Q* – output (:=*1* if *CV >= PV*) *CV* – current value
2		Down counter *CD* – input (rising edges are counted down) *LD* – load counter with the *PV* *PV* – preset value *Q* – output (:=*1* if *CV <= 0*) *CV* – current value
3		Up-down counter *CU* – input (rising edges are counted up) *CD* – input (rising edges are counted down) *R* – reset counter (*CV:=0*) *LD* – load counter with the *PV* *PV* – preset value *QU* – output (:=*1* if *CV >= PV*) *QD* – output (:=*1* if *CV <= 0*) *CV* – current value

(FBD *)*

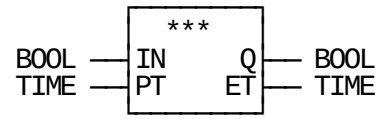
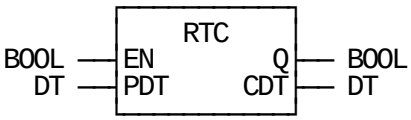
MyCounter (* instance name of FB *)



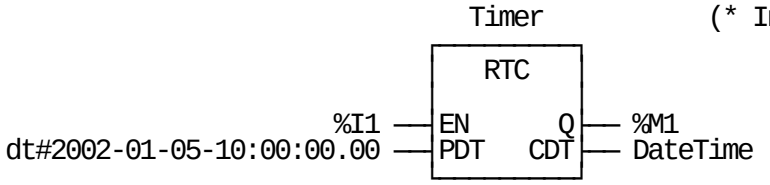

```
(* textual languages *)
VAR
  MyCounter : CTUD;                          (* Instance name of FB *)
END_VAR
(* ST *)
 MyCounter(CU:=%I1, CD:=%I2, R:=%I3, LD:=%I4);          (* Invocation *)
 %Q1:= MyCounter.QU;                          (* Output assignment to %Q1 *)
 %Q2:= MyCounter.QD;                          (* Output assignment to %Q2 *)
  (* IL *)
 CAL MyCounter(CU:=%I1, CD:=%I2, R:=%I3, LD:=%I4);      (* Invocation *)
 LD MyCounter.QU
 ST %Q1                                       (* Output assignment to %Q1 *)
 LD MyCounter.QD
 ST %Q2                                       (* Output assignment to %Q2 *)
```

Example 17. Usage of *CTUD* in FBD, ST and IL

Table 33. Standard timer FBs

No.	Graphical form	Description
1 2 3	 *** denotes: *TP, TON, TOF*	*TP* (Pulse generator) *TON* (Timer ON delay) *TOF* (Timer OFF delay) *IN* – Input turning on the timer *PT* – Preset Time *ET* – Expired Time
4		Real Time Clock *PDT* – Preset Date and Time (loaded at rising edge of *EN*) *CDT* – Current Date and Time (while *EN = 1*) *Q* – copy of *EN*

(FBD *)*



(textual languages *)*

```
VAR
  Timer : RTC;                                   (* Instance name *)
END_VAR

(* ST *)

  Timer(IN:=%I1, PDT:= dt#2002-01-05-10:00:00.00);          (* Invocation *)
  %M1:= Timer.Q;                                  (* Output assignment to %M1 *)
  DateTime:= Timer.CDT;                          (* Output assignment to DateTime *)

(* IL *)

  CAL Timer(IN:=%I1, PDT:= dt#2002-01-05-10:00:00.00)        (* Invocation *)
  LD Timer.Q
  ST %M1                                          (* Output assignment to %M1 *)
  LD Timer.CDT
  ST DateTime                                    (* Output assignment to DateTime *)
```

Example 18. Usage of *RTC* in FBD, ST and IL

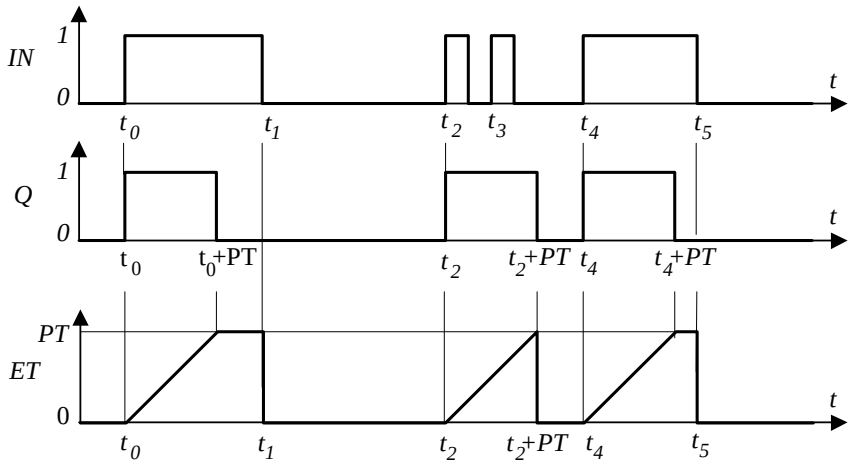


Figure 2. Timing diagrams – TP

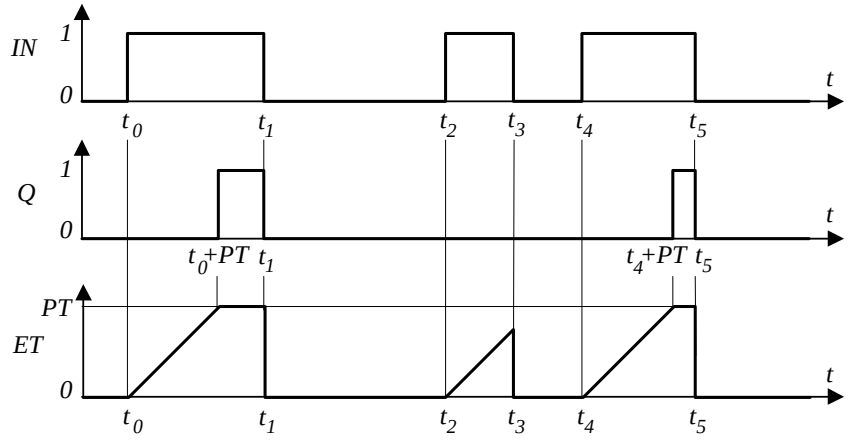


Figure 3. Timing diagrams – TON

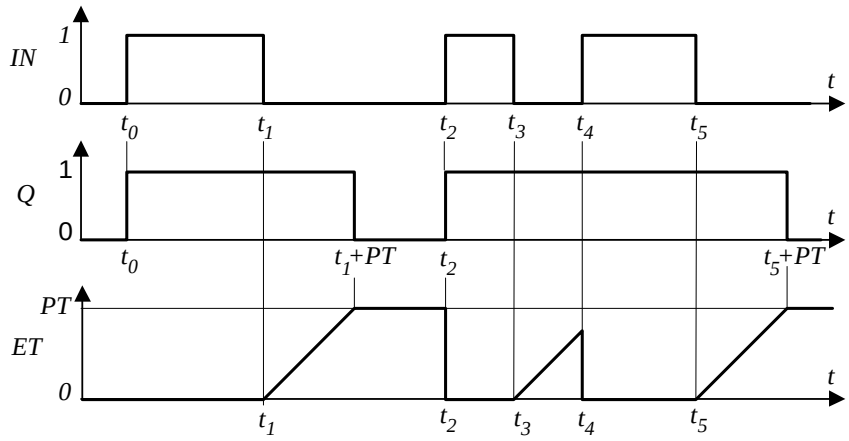
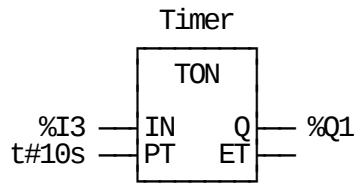


Figure 4. Timing diagrams – TOF

(FBD *)*



(Textual languages *)*

```
VAR
  Timer : TON;                    (* Instance name of FB *)
END_VAR

(* ST *)
  Timer(IN:=%I3, PT:=t#10s);      (* Invocation *)
  %Q1:= Timer.Q;                  (* Output assignment to %Q1 *)

(* IL *)
  CAL Timer(IN:=%I3, PT:=t#10s)   (* Invocation *)
  LD Timer.Q
  ST %Q1                          (* Output assignment to %Q1 *)
```

Example 19. Usage of *TON* in FBD, ST and IL

```
    (* ST *)                                    (* IL *)
    (* ↓                  setting delay           ↓ *)
  Timer.PT:= t#10s;                             LD t#10s
                                                ST Zegar.PT
                              .....
                       (* timer invocation *)
  Timer(IN:=%I3);                               LD %I3
                                                ST Timer.IN
                                                CAL Timer
                              .....
                       (* output assignment to %Q1 *)
  %Q1:= Zegar.Q;                                LD Timer.Q
                                                ST %Q1
```

Example 20. The other way of *TON* usage in textual languages

Calling Functions and FBs

- Programs may call functions and FBs, but not inversely;
- FBs may call other FBs;
- FBs may call functions, but not inversely;
- Functions may call only other functions;
- Recursive calls are forbidden.

```
FUNCTION FF1 :BOOL;
  VAR_INPUT
    X : INT;
  END_VAR

IF FF1(X) THEN .....  (* Forbidden recursive call of declared function *)

END_IF

END_FUNCTION
```

Example 21. Invalid recursive call of function in ST

```
FUNCTION_BLOCK FB1
VAR_INPUT
  IN1 : INT;
END_VAR
VAR
  MyFB : FB1;       (* Forbidden recursive instantiation for declared FB *)
  XX : INT;
END_VAR

MyFB(XX);           (* Forbidden recursive call of declared FB *)

END_FUNCTION_BLOCK
```

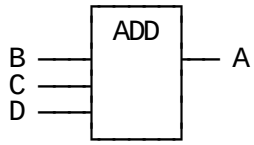
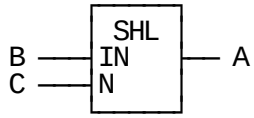
Example 22. Invalid recursive call of FB in ST

```
(* Declaration of function FF1 *)
                                    FUNCTION FF1 : BOOL;
 VAR_INPUT
                                    X : INT;
 END_VAR
 VAR
                                    Y : REAL;
 END_VAR
      ...
 Y:=FF2(0.0);          (* Forbidden call of FF2 causing the recurrence *)
      ...
                                    END_FUNCTION

(* Declaration of function FF2 *)
                                    FUNCTION FF2 : REAL;
 VAR_INPUT
                                    X : REAL;
 END_VAR
      ...
 IF FF1(5) THEN .....  (* Forbidden call of FF1 causing the recurrence *)
      ...
 END_IF
      ...
                                    END_FUNCTION
```

Example 23. Invalid recursion by mutual invocation in ST

Formal and actual parameters

Table 34. Use of formal parameter names

Example	Explanation
	Graphical (FBD) use of *ADD* function – no formal parameter names
A := ADD(B, C, D) ;	Textual (ST) use of *ADD* function – no formal parameter names
	Graphical (FBD) use of *SHL* function – formal parameter names
A := SHL(IN:=B, N:=C);	Textual (ST) use of *SHL* function – formal parameter names

```
FUNCTION_BLOCK Fblock      (* Declaration of FB *)
  VAR_INPUT
    Par1 : BOOL;
    Par2: TIME;
    Par3 : INT;
  END_VAR
  (* FB body *)
END_FUNCTION_BLOCK
```

Example 24. FB declaration with three input parameters

```
VAR
  FB1, FB2, FB3 : Fblock;  (* Three instances of Fblock *)
  AT %I1 : BOOL;
  AT %IW1 : INT;
END_VAR

(* Complete FB call  *)
CAL FB1(Par1 := %I1, Par2 := t#10s, Par3:= %IW1)

(* Complete FB call with parameters in changed order *)
CAL FB2(Par3 := %IW1, Par1 := %I1, Par2:= t#10s)

(* Incomplete FB call with parameters in changed order *)
CAL FB1(Par2 := t#10s, Par1:= %I1)
```

Example 25. FB call with parameters omitted and in a different order, written in IL

```
VAR
  C1, C2  : CTU;                                  (* 2 instances of CTU *)
END_VAR

(* instructions *)

C2(CU := C1.Q, PV := 100, R:= %I1);                (* Invocation of C2 *)
%QW10 := INT_TO_BCD(C2.CV);       (* Instance name of C2 as actual parameter*)
```

Example 26. Using the FB instance name as actual parameter

7. Programming languages

Common elements

- *TYPE...END_TYPE;*
- *VAR...END_VAR;*
- *VAR_INPUT...END_VAR;*
- *VAR_OUTPUT...END_VAR;*
- *VAR_IN_OUT...END_VAR;*
- *VAR_EXTERNAL...END_VAR;*
- *FUNCTION...END_FUNCTION;*
- *FUNCTION_BLOCK...END_FUNCTION_BLOCK;*
- *PROGRAM...END_PROGRAM;*
- *STEP...END_STEP;*
- *TRANSITION...END_TRANSITION;*
- *ACTION...END_ACTION.*

Instruction List (IL)

Table 35. Examples of instruction fields

Label	Operator	Operand	Comment
START:	*LD*	*%IX1*	*(* PUSH BUTTON *)*
	ANDN	*%MX5*	*(* NOT INHIBITED *)*
	ST	*%QX2*	*(* FAN ON *)*

```
VAR
  Operand1, Operand2, Result : INT :=0;
END_VAR
Et1: LD Operand1    (* CR ← Operand1, here value 0 *)
     ADD 10         (* CR ← CR + 10, here CR = 10 *)
     ST Result      (* Result ← CR, value of CR remains *)
     GT 0           (* CR > 0? – yes, so CR := TRUE *)
     JMPC Et2       (* jump to label Et2, if CR = TRUE, value of CR remains *)
     ADD Operand2   (* Add Operand2 to CR – Error !!! – data type mismatch *)
Et2:
```

Example 27. The universal accumulator of IL (CR – Current Result)

```
LD X1           (* CR ← X1 *)
MUL( X2         (* CR ← X2 *)
  SUB( X3       (* CR ← X3 *)
    ADD X4      (* CR ← X3 + X4 *)
  )             (* CR ← X2 – (X3 + X4) *)
)               (* CR ← X1 * (X2 – (X3 + X4)) *)
ST Y            (* CR remains *)
```

Example 28. Computation of nested expressions in parentheses

```
FUNCTION  MyFun : INT;    (* Function declaration *)
  VAR_INPUT
    IN1, IN2, IN3 : INT;        (* Input parameters *)
  END_VAR
(* Function body *)
  LD    IN1
  ADD IN2
  ADD IN3
  ST    MyFun                 (* Function output *)
END_FUNCTION

(* calling MyFun in POU *)
VAR
  Par1, Par2, Par3, Result : INT;     (* Variable declaration *)
END_VAR

LD  Par1                      (* First input parameter *)
MyFun  Par2, Par3             (* Function call with remaining parameters *)
ST  Result                    (* Store the result of MyFun in location Result *)
```

Example 29. Function call in IL

Table 36. IL operators

No.	Operator	Modifiers	Operand	Semantics
1	*LD*	*N*	*	Set *CR* equal to operand (*LoaD*)
2	*ST*	*N*	*	Store *CR* to operand location (*STore*)
3	*S* *R*		*BOOL* *BOOL*	If *CR = 1* then set Boolean operand to *1* (*Set*) If *CR = 1* then reset Boolean operand (*Reset*)
4	*AND*	*N, (, N(*	*BOOL*	Boolean *AND*
5	*OR*	*N, (, N(*	*BOOL*	Boolean *OR*
6	*XOR*	*N,(, N(*	*BOOL*	Boolean (*eXclusive OR*)
7	*ADD*	*(*	*	*ADDition*
8	*SUB*	*(*	*	*SUBtraction*
9	*MUL*	*(*	*	*MULtiplication*
10	*DIV*	*(*	*	*DIVision*
11	*GT*	*(*	*	Comparison: *CR* > operand (*Greater Than*)
12	*GE*	*(*	*	Comparison: *CR* >= operand (*Greater than or Equal*)
13	*EQ*	*(*	*	Comparison: *CR* = operand (*Equal*)
14	*NE*	*(*	*	Comparison: *CR* <> operand (*Not Equal*)
15	*LE*	*(*	*	Comparison: *CR* <= operand (*Less than or Equal*)
16	*LT*	*(*	*	Comparison: *CR* < operand (*Less Than*)
17	*JMP*	*C, CN*	*LABEL*	*JuMP to label*
18	*CAL*	*C, CN*	*NAME*	*CALl* function block *NAME*
19	*RET*	*C, CN*		*RETurn* from called function or FB
20	*)*			Evaluate deferred operation

* – these operators are either overloaded (type *ANY*) or typed, the CR and the operand shall be of the same type

Table 37. Function block invocation feature for IL assuming it was declared *VAR C10 : CTU; END_VAR*

No.	Description/Example
1	*CAL* with input list: *CAL C10(CU:=%I10, R:= %I11, PV:=15)*
2	*CAL* with load/store of inputs: *LD 15* *ST C10.PV* *LD %I10* *ST C10.CU* *LD %I11* *ST C10.R* *CAL C10*
3	Use of input operators (see Table 38.) *LD 15* *PV C10* *LD %I1* *CU C10* *LD %I11* *R C10* *CAL C10*

Output assignment in the example presented above:

```
LD C10.Q
ST %QX1   (*Output of the counter fires the output %QX1 *)
LD C10.CV
ST CurrentValue  (* Count value stored at variable CurrentValue that has to be
                    declared as INT type *)
```

Table 38. Standard function block input operators for IL language

No.	Operators	Function blocks
1	*S1, R*	*SR*
2	*S, R1*	*RS*
3	*CLK*	*R_TRIG*
4	*CLK*	*F_TRIG*
5	*CU, R, PV*	*CTU*
6	*CD, LD, PV*	*CTD*
7	*CU, CD, R, LD, PV*	*CTUD*
8	*IN, PT*	*TP*
9	*IN, PT*	*TON*
10	*IN, PT*	*TOF*

Table 39. Operators of the ST language

No.	Symbol	Operation	Example
1	(*expression*)	Parenthesization	*(X+Y)*(X-Y)*
2	*Fun_Name*(argument list)	Function evaluation	*LN(A), MAX(X, Y),*
3	**	Exponentiation	*X**Y*
4	–	Negation	*–10, –X*
5	*NOT*	Complement	*NOT (X > Y)*
6	*	Multiply	*X * Y*
7	/	Divide	*X / Y*
8	*MOD*	Modulo	*13 MOD 10* (Result: *3*)
9	+	Add	*X + Y*
10	–	Subtract	*X – Y*
11	<, >, <=, >=	Comparison	*T#1h > T#30m* (Result: *TRUE*)
12	=	Equality	*T#1d = T#24h* (Result: *TRUE*)
13	<>	Inequality	
14	*AND* or *&*	Boolean AND	*(X >Y) AND (X < Z)*
15	*XOR*	Boolean Exclusive OR	*TRUE XOR FALSE*
16	*OR*	Boolean OR	*TRUE OR FALSE*

Precedence: highest – No. 1, lowest – No. 16

Table 40. ST language statements

No.	Statement	Examples
1	Assignment	*A:=B; CV:=CV+1; Y:=SIN(X); D:=INT_TO_REAL(C)*
2	FB invocation FB output usage	*My_TMR(IN:=%IX5, PT:=T#300ms);* *A:=My_TMR.Q;*
3	*RETURN*	*RETURN;*
4	*IF*	*D:=B*B-4*A*C;* *IF D < 0.0 THEN NROOTS:=0; (* no roots *)* *ELSIF D = 0.0 THEN (* one root *)* *NROOTS := 1;* *X1 := –B / (2.0 * A);* *ELSE (* two roots *)* *NROOTS := 2;* *X1 := (–B + SQRT(D)) / (2.0 * A);* *X2 := (–B – SQRT(D)) / (2.0 * A);* *END_IF;*
5	*CASE*	*ERROR:=0;* *XW:=BCD_TO_INT(Y);* *CASE XW OF* *1,4: DISPLAY := TEKST1;* *2: DISPLAY := TEKST2;* *Y := SIN(Z);* *3,5..10: DISPLAY := STATUS (XW – 3);* *ELSE DISPLAY := ’’; (* XW outside 1..10 *)* *ERROR := 1;* *END_CASE;*
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 := 1;* *WHILE J <= 100 AND WORDS(J) <> ’KEY’ DO* *J := J+2;* *END_WHILE;*
8	*REPEAT*	*J := –1;* *REPEAT* *J := J+2;* *UNTIL J = 101 OR WORDS(J) = ’KEY’* *END_REPEAT;*
9	*EXIT*	*EXIT;*

```
SUM:=0
FOR I:=1 TO 3 DO                    (* the first loop *)
  FOR J:=1 TO 2 DO                  (* the nested loop *)
    IF FLAG THEN EXIT;  END_IF      (* exit from the nested loop *)
    SUM:=SUM+J;
  END_FOR;                          (* end of the nested loop *)
  SUM:=SUM+I;
END_FOR;                            (* end of the first loop *)
```

Example 30. EXIT statement example

Ladder Diagram (LD)

Table 41. Execution control elements

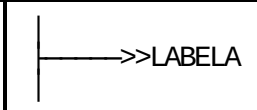
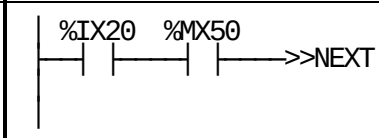
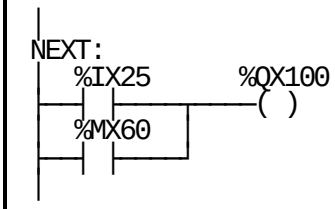
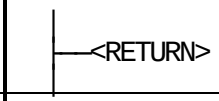
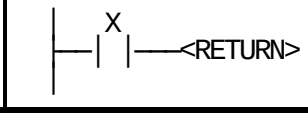
No.	Symbol/Example	Explanation
1		Unconditional Jump to *LABELA*
2		Conditional Jump to label *NEXT* (jump is executed only if there is a power flow to symbol >>)
		Jump target
3		Unconditional return from POU
4		Conditional return from POU

Table 42. Power rails and link elements

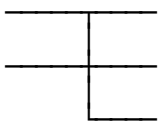
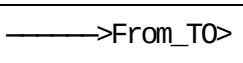
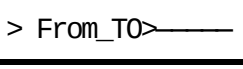
No.	Symbol	Description
1		Left power rail (with attached horizontal link)
2		Right power rail (with attached horizontal link)
3		Horizontal link
4		Vertical link (with attached horizontal links)
5	 	Connector Continuation of connected line

Table 43. Contacts

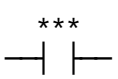
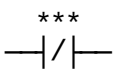
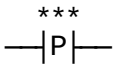
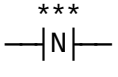
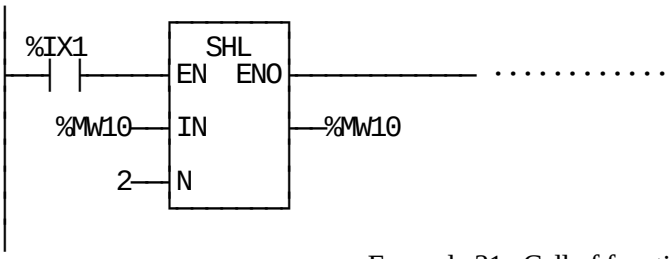
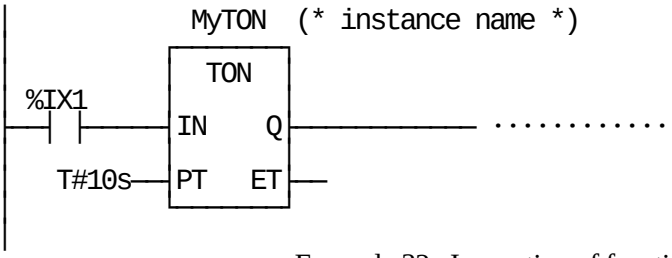
Contact	Symbol	Description
Static contacts		Normally open contact The state of the left link is copied to the right link if the state of the associated Boolean variable (indicated by ***) is *ON*. Otherwise, the state of the right link is *OFF*.
		Normally closed contact The state of the left link is copied to the right link if the state of the associated Boolean variable (indicated by ***) is *OFF*. Otherwise, the state of the right link is *OFF*.
Transition-sensing contacts		Positive transition-sensing contact The state of the right link is *ON* from one evaluation of this element to the next when a transition of the associated Boolean variable from *OFF* to *ON* is sensed at the same time that the state of the left link is *ON*. The state of the of the right link shall be *OFF* at all other times.
		Negative transition-sensing contact The state of the right link is *ON* from one evaluation of this element to the next when a transition of the associated Boolean variable from *ON* to *OFF* is sensed at the same time that the state of the left link is *ON*. The state of the of the right link shall be *OFF* at all other times.

Table 44. Coils

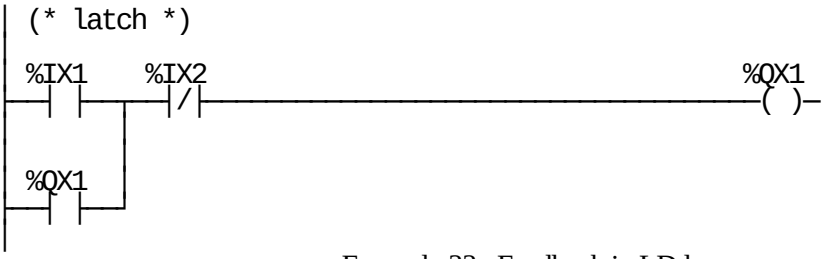
Coils	Symbol	Description
Momentary coils	*** —( )—	Coil The state of the left link is copied to the associated Boolean variable and to the right link.
	*** —(/)—	Negated coil The state of the left link is copied to the right link. The inverse of the state of the left link is copied to the associated Boolean variable.
Latched coils	*** —(S)—	Set coil, Latch coil The associated Boolean variable is set to the *ON* state when the left link is in the *ON* state, and remains set until reset by a (R) coil.
	*** —(R)—	Reset coil, Unlatch coil The associated Boolean variable is reset to the *OFF* state when the left link is in the *ON* state, and remains reset until set by a (S) coil.
Retentive coils	*** —(M)—	Retentive coil, Memory coil
	*** —(SM)—	Set retentive coil
	*** —(RM)—	Reset retentive coil
Transition-sensing coils)	*** —(P)—	Positive transition-sensing coil The state of the associated Boolean variable is *ON* from one evaluation of this element to the next when a transition of the left link from *OFF* to *ON* is sensed. The state of the left link is always copied to the right link.
	*** —(N)—	Negative transition-sensing coil The state of the associated Boolean variable is *ON* from one evaluation of this element to the next when a transition of the left link from *ON* to *OFF* is sensed. The state of the left link is always copied to the right link.



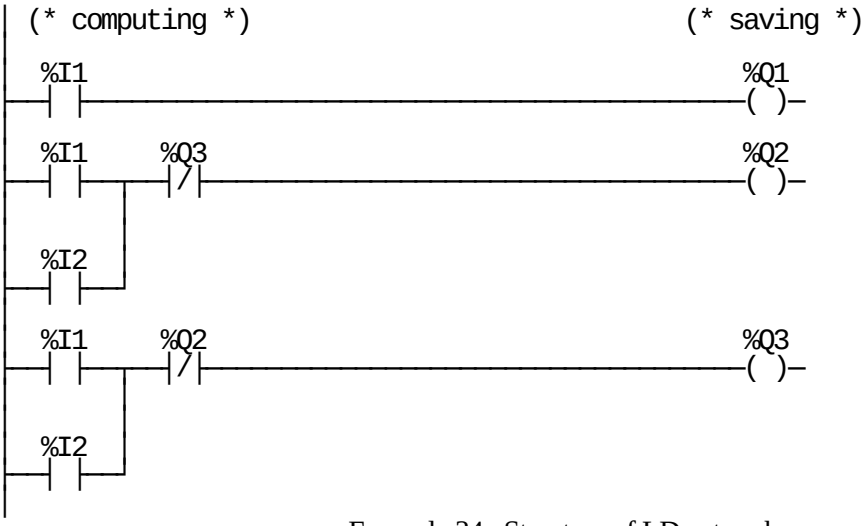
Example 31. Call of function SHL



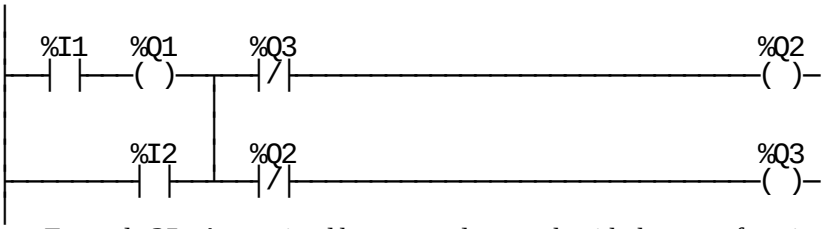
Example 32. Invocation of function block TON



Example 33. Feedback in LD language



Example 34. Structure of LD network



Example 35. A permitted but unusual network with the same functionality as in Example 34.

Function Block Diagram (FBD)

Table 45. Connections in FBD

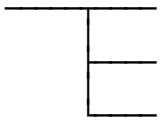
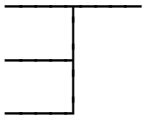
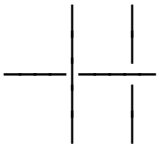
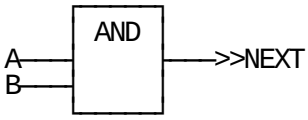
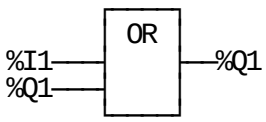
No.	Symbol	Description
1		Horizontal connection
2		Vertical connection with horizontal connections (splitting).
3		Forbidden connection („wired OR”), see Example 36.
4		Line crossings without connection.

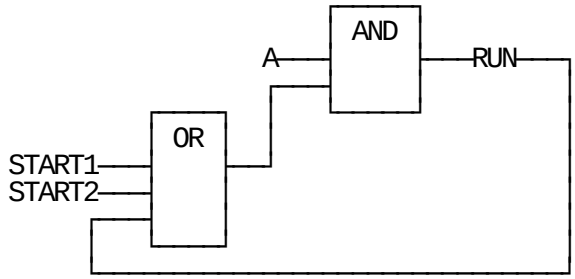
Table 46. Graphical elements of execution control in FBD

No.	Symbol	Description
1	1——>>LABELA	Unconditional jump to *LABELA*
2	X——>>LABELB  NEXT:  	Conditional jump (if *X = 1*) to *LABELB* Example of conditional jump: Jump condition (both *A* and *B* shall be equal to *1*) Jump target (marked by label)
3	X——<RETURN>	Conditional return from function or FB
4	END_FUNCTION END_FUNTION_BLOCK	Unconditional return from function Unconditional return from FB

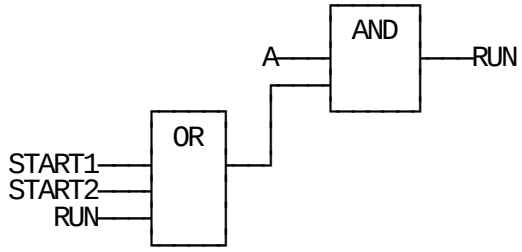
„wired-OR" in LD *Function OR in FBD*

Example 36. Boolean OR examples

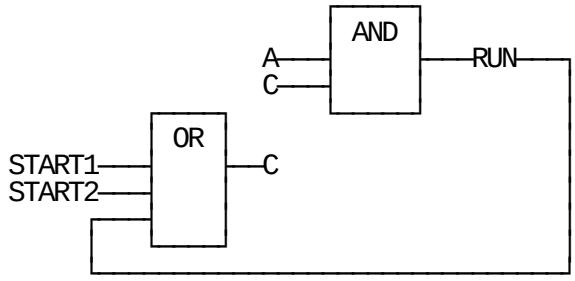
```
(* example 1 - explicit connection *)
```




```
(* example 2 - implicit connection *)
```




```
(* example 3 – implicit connection *)
```



Example 37. FBD network with feedback – explicit and implicit connections

8. Sequential Function Chart (SFC)

Steps and transitions

Table 47. Step features

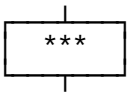
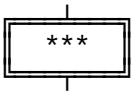
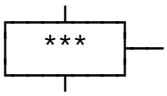
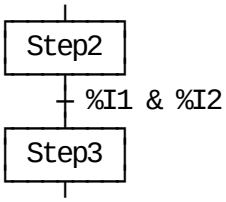
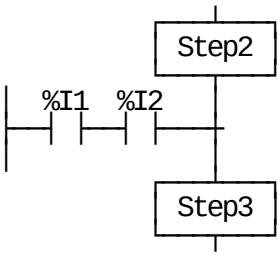
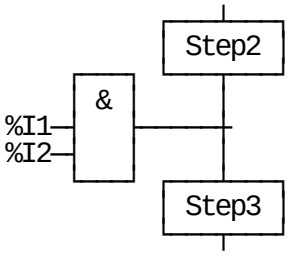
No.	Graphical form	Textual form	Description
1		*STEP *** :* *(* Step body *)* *END_STEP*	Step *** denotes step name
		*INITIAL_STEP *** :* *(* Step body *)* *END_STEP*	Initial step The upper directed link is not required if the initial step has no predecessors.
2	***.*X*		Step flag – general form
			Step flag – direct connection of Boolean variable ***.*X* to right side of step
3	***.*T*		Step elapsed time – general form ***.*T* is a variable of type *TIME*

Table 48. Transition and transition conditions in graphical form

No.	Example	Description
1		Predecessor step Transition condition in ST Successor step
2		Predecessor step Transition condition in LD Successor step
3		Predecessor step Transition condition in FBD Successor step

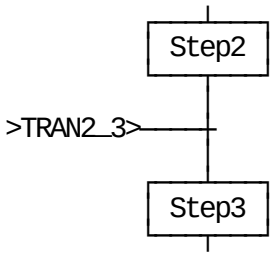
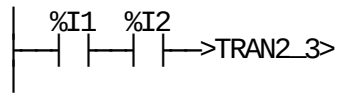
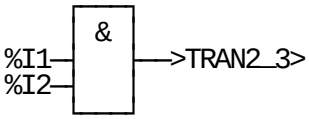
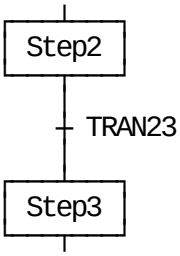
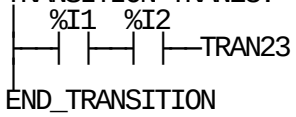
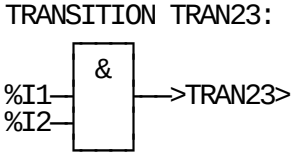
No.	Example	Description
4		Use of connector: Predecessor step Transition connector Successor step
4a		Transition condition in LD
4b		Transition condition in FBD
5		Use of transition name: Predecessor step Transition name Successor step
5a		Transition condition in LD
5b		Transition condition in FBD
5c	*TRANSITION TRAN23:* *LD %I1* *AND %I2* *END_TRANSITION*	Transition condition in IL
5d	*TRANSITION TRAN23:=* *%I1 & %I2;* *END_TRANSITION*	Transition condition in ST

Table 36 continued

```
STEP STEP2:
      … (* Declaration of STEP2 *)
END_STEP

(* Declaration of transition from STEP2 to STEP3 *)
TRANSITION FROM STEP2 TO STEP3:=

(* Transition condition in ST *)
   %I1 & %I2;

(* Transition condition in IL *)
   LD  %I1
   AND %I2
END_TRANSITION   (* End of transition declaration *)

STEP STEP3:
      … (*Declaration of STEP3 *)
END_STEP
```

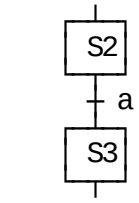
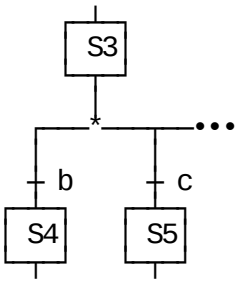
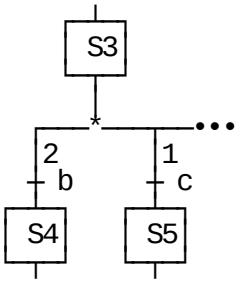
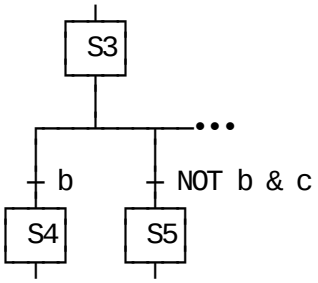
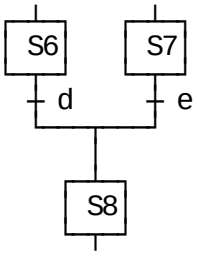
Example 38. Transition and transition condition in textual form

```
TRANSITION FROM (S1, S2, S3) TO (S4,S5):=
   (* Boolean expression for transition conditions *)
END_TRANSITION
```

Example 39. Transition in textual form for greater number of steps (predecessors or successors)

Sequence evolution

Table 49. Sequence evolution

No.	Example	Rule
1		*Single sequence:* The alternation step – transition is repeated in series. Example: An evolution from *S2* to *S3* shall take place if and only if step *S2* is *active* and the transition condition *a* is *true*.
2a		*Divergence of sequence selection:* A selection between several sequences is represented by as many transition symbols, *under* the horizontal line, as there are different possible evolutions. The asterisk denotes left-to-right priority. Example: An evolution shall take place from *S3* to *S4* only if *S3* jest *active* and the transition condition *b* is *TRUE*, or from *S3* to *S5* only if *S3* is *active* and *c* is *TRUE* and *b* is *FALSE*.
2b		*Divergence of sequence selection:* The asterisk follow by numbered branches, indicates a user-defined priority of transition evaluation, with the lowest-numbered branch having the highest priority. Example: An evolution shall take place from *S3* to *S5* only if *S3* jest *active* and the transition condition *c* is *TRUE*, or from *S3* to *S4* only if *S3* is *active* and *b* is *TRUE* and *c* is *FALSE*.
2c		*Divergence of sequence selection:* The connection of the branch indicates that the user must assure that transition conditions are mutually exclusive. Example: An evolution shall take place from *S3* to *S4* only if *S3* jest *active* and the transition condition *b* is *TRUE*, or from *S3* to *S5* only if *S3* is *active* and *b* is *FALSE* and *c* is *TRUE*.
3		*Convergence of sequence selection:* The end of sequence selection is represented by as many transition symbols, *above* the horizontal line, as there are selection paths to be ended. Example: An evolution shall take place from *S6* to *S8* only if *S6* jest *active* and the transition condition *d* is *TRUE*, or from *S7* to *S8* only if *S7* is *active* and *e* is *TRUE*.

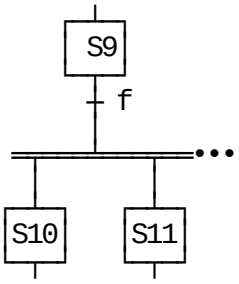
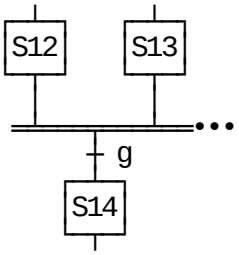
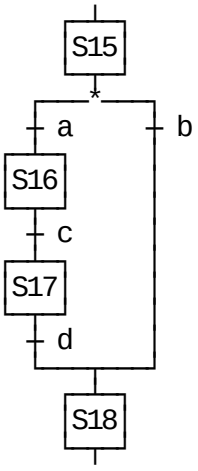
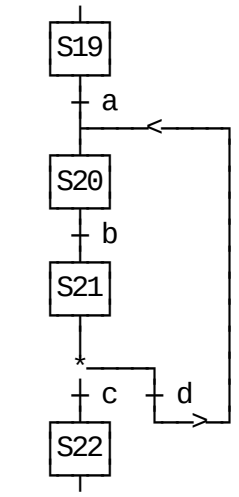
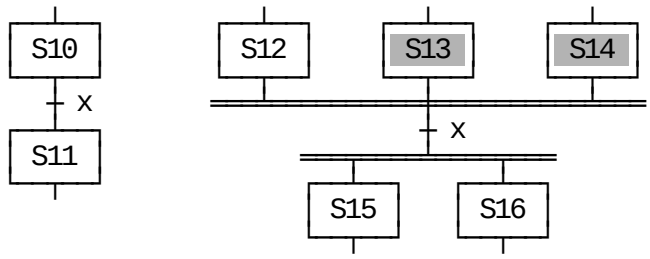
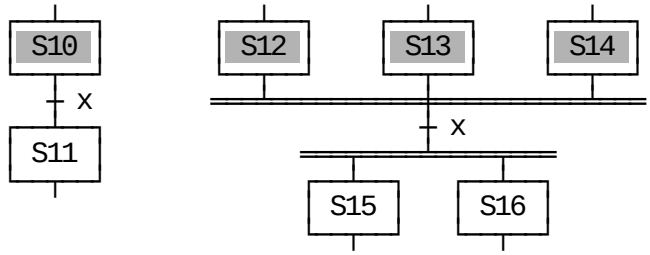
No.	Example	Rule
4		*Simultaneous sequence - divergence:* *Only one common* transition symbol shall be possible, immediately *above* the *double* horizontal line of synchronization. Example: An evolution shall take place from *S9* to *S10, S11,…* only if *S9* jest *active* and the transition condition *f* associated to the common transition is *TRUE*. After the simultaneous activation of *S10, S11,*etc., the evolution of each sequence proceeds independently.
5		*Simultaneous sequence - convergence:* *Only one common* transition symbol shall be possible, immediately *under* the *double* horizontal line of synchronization. Example: An evolution shall take place from *S12, S13,…* to *S14* to only if *all* steps above and connected to the double horizontal line are *active* and the transition condition *g* associated to the common transition is *TRUE*.
6a 6b 6c		*Sequence skip:* A sequence skip is a special case of sequence selection (feature 2) in which one or more of the branches contain no steps. Feature 6a, 6b and 6c correspond to the representation options given in features 2a, 2b and 2c, respectively. Example (feature 2a shown): An evolution shall take place from *S15* to *S18* if *a* is *FALSE* and *b* is *TRUER*, that is the sequence (*S16, S17*) will be skipped.
7a 7b 7c		*Sequence loop:* A sequence loop is a special case of sequence selection (feature 2) in which one or more of the branches returns to a preceding step. Feature 7a, 7b and 7c correspond to the representation options given in features 2a, 2b and 2c, respectively. Example (feature 7a shown): An evolution shall take place from *S21* to *S20* if *c* is *FALSE* and *d* is *TRUER*, that is the sequence (*S20, S21*) will be repeated.

Table 49. – continued

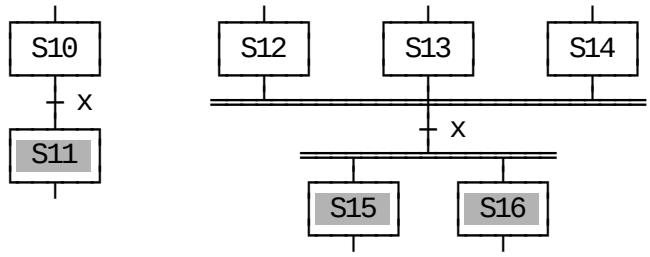
a)



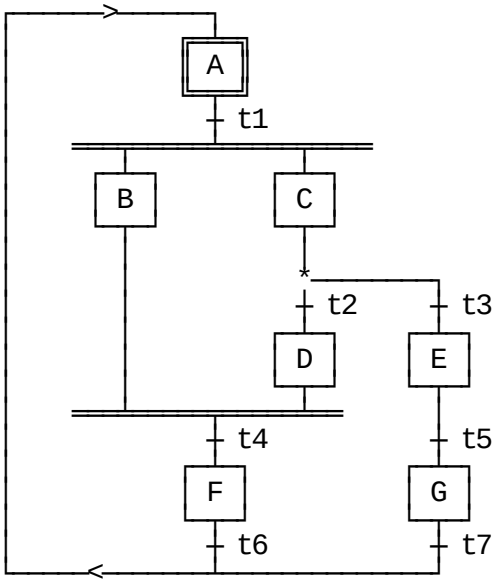
b)



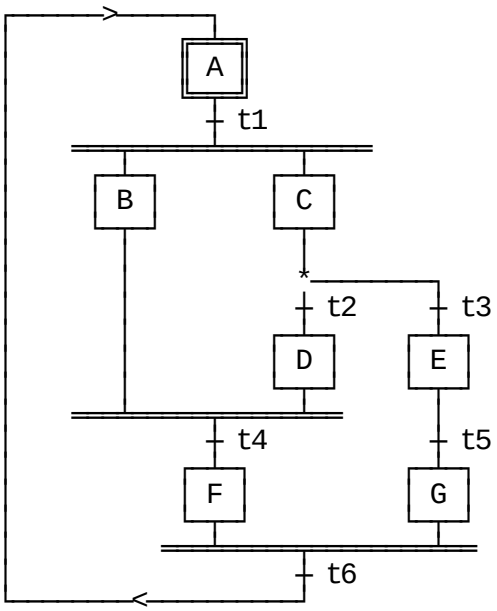
c)



Example 40. Rules of evolution



Example 41. SFC errors – an „unsafe” SFC



Example 42. SFC errors – an „unreachable" SFC

Actions

Mechanisms for the declaration of actions:

- Any Boolean variable declared in a *VAR* or *VAR_OUTPUT* block, or their graphical equivalents, can be an action.
- see Table 50.

Table 50. Declaration of actions

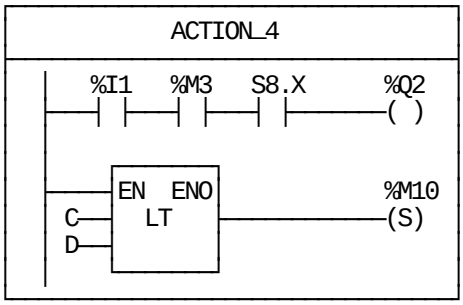
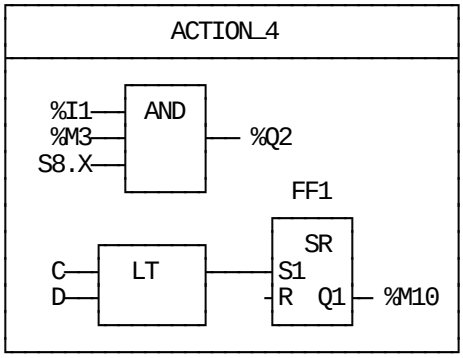
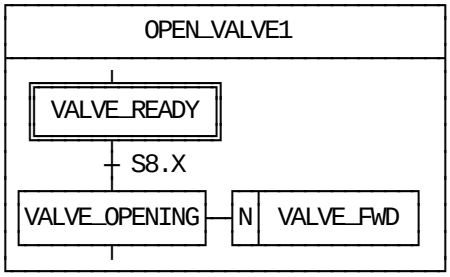
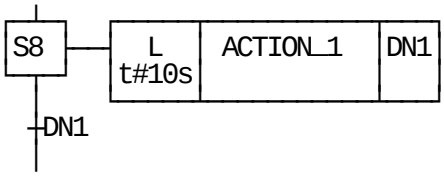
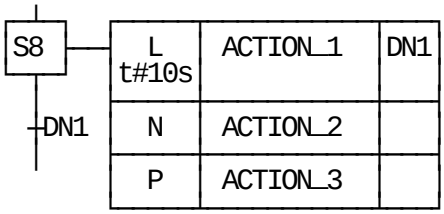
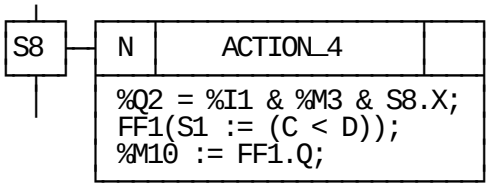
No.	Example	Feature
1		Graphical declaration in LD
2		Graphical declaration in FBD
3	*ACTION ACTION_4:* *%Q2 := %I1 & %M3 & S8.X;* *FF1(S1 := (C < D));* *%M10 := FF1.Q;* *END_ACTION*	Textual declaration in ST
4	*ACTION ACTION_4:* *LD S8.X* *AND %I1* *AND %M3* *ST %Q2* *LD C* *LT D* *S1 FF1* *LD FF1.Q* *ST %M10* *END_ACTION*	Textual declaration in IL
5		Inclusion of SFC elements in action

Table 51. Step/action association

No.	Example	Description
1		Action block
2		Concatenated action blocks
3	*STEP S8:* *ACTION_1(L, t#10s, DN1);* *ACTION_2(N);* *ACTION_3(P);* *END_STEP*	Textual step body
4		Declaration of action in action block (see Figure 5)

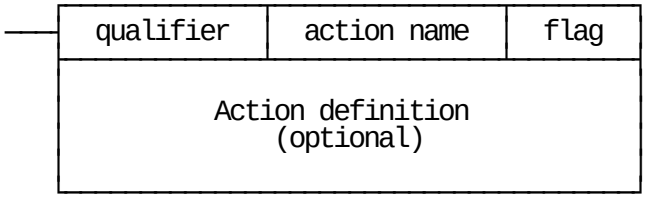
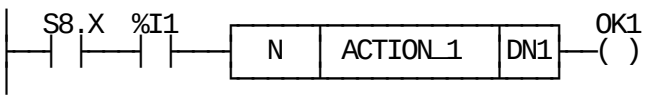
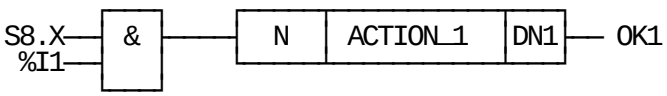


Figure 5. Graphical form of action block definition

```
(* Use of action block in LD *)
```



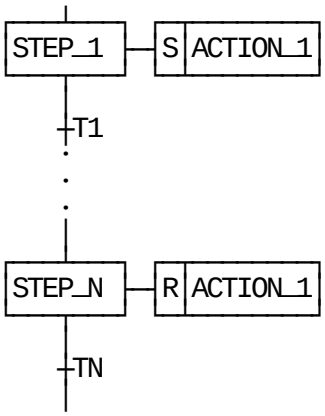

```
(* Use of action block in FBD *)
```



Example 43. Use of action block in graphical languages

Table 52. Action qualifiers

No.	Qualifier	Description
1	None	Non-stored (null qualifier). An action is executed only if associated step is *active*.
2	*N*	Non-stored – description as above.
3	*R*	Overriding *Reset*
4	*S*	*Set* (stored)
5	*L*	time *Limited*
6	*D*	time *Delayed*
7	*P*	*Pulse*
8	*SD*	*Stored* and time *Delayed* (combination of *S* and *D*)
9	*DS*	time *Delayed* and *Stored* (combination of *D* and *S*)
10	*SL*	*Stored* and time *Limited* (combination of *S* and *L*)



Example 44. Fragment of SFC network

Examples of SFC

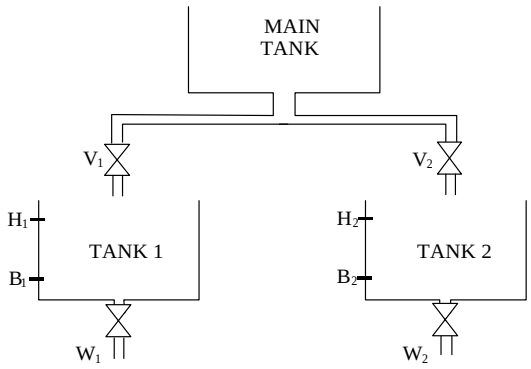
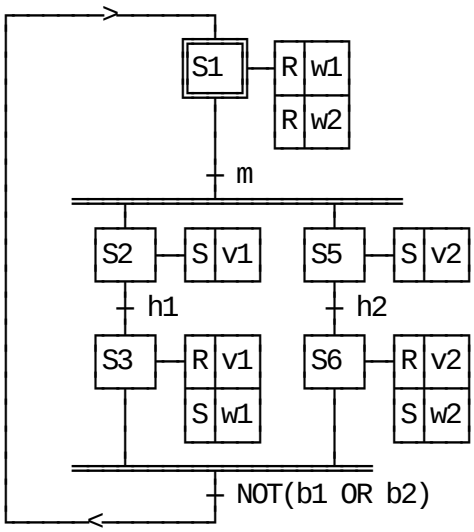
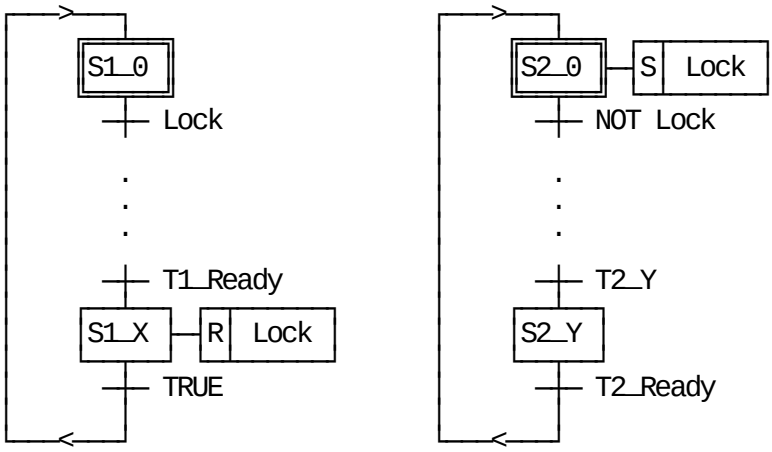


Figure 6. Two tanks filling.

- The initial state – both tanks are empty.
- If push button *m* is pressed, both tanks are filled by opening valves.
- When a tank is full, filling stops and its contents start to be used.
- Filling may only start up again when both tanks are empty and if the button *m* is pressed.



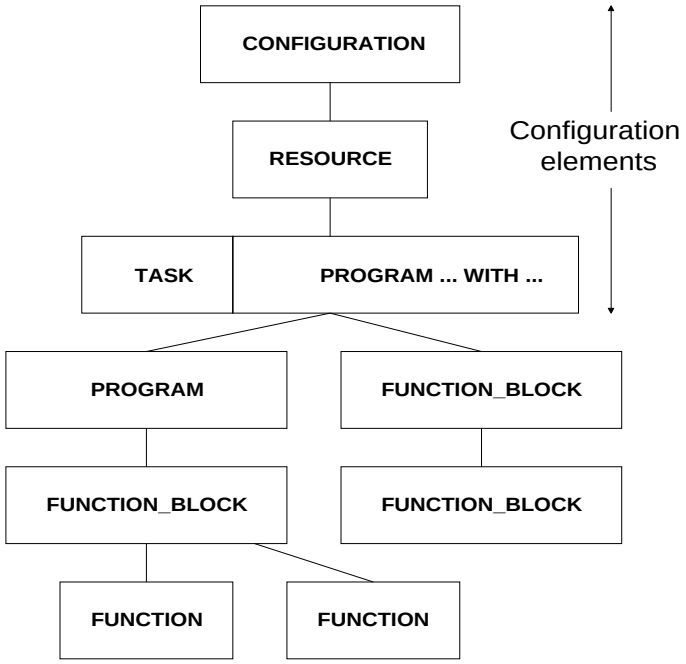
Example 45. SFC to control of two tanks filling



Example 46. Two SFC networks synchronized by the Boolean action *Lock* using the qualifiers *S* and *R*

9. Configuration elements

Overall software structure according to IEC 61131-3:



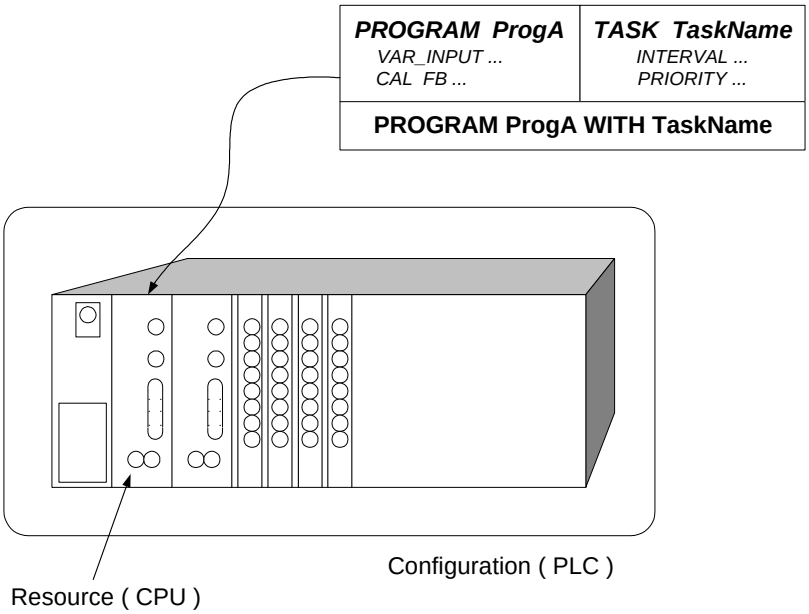
Configuration elements assign properties to POUs:

- PROGRAMs and FB's are assigned run-time properties on a particular hardware;
- Program variables are assigned to PLC hardware;
- Communication relationships are defined between configurations.

Configuration elements match the elements in real-world PLC systems:

- *Configuration* – a PLC system, e.g. a controller in a rack with multiple CPUs, controlling a cell of machines;
- *Resource* – one CPU in a PLC, possibly capable of *multitasking*;
- *Task* – run-time properties for programs and FBs ("type" of a PLC program);
- *Run-time program* – unit consisting of a PROGRAM or FUNCTION_BLOCK and the TASK associated with it.

Relations between configuration elements and real-world PLC hardware:



Configuration: - Definition of global variables (valid within this configuration)
- Combination of all resources of a PLC system
- Definition of access paths between configurations
- Declaration of directly represented variables

Resource: - Definition of global variables (valid within this resource)
- Assignment of tasks and programs to a resource
- Invocation of run-time programs with input and output parameters
- Declaration of directly represented variables

Task: - Definition of run-time properties and assignment them to a PROGRAM or FB

Table 53. Configuration and resource declaration features

No.	Description
1	*CONFIGURATION...END_CONFIGURATION* construction
2	*VAR_GLOBAL...END_VAR* construction within *CONFIGURATION*
3	*RESOURCE...ON...END_RESOURCE* construction
4	*VAR_GLOBAL...END_VAR* construction within *RESOURCE*
5a	Periodic *TASK* construction within *RESOURCE*
5b	Non-periodic *TASK* construction within *RESOURCE*
6a	*PROGRAM* declaration with *PROGRAM* – to – *TASK* association
6b	*PROGRAM* declaration with *FB* – to – *TASK* association
6c	*PROGRAM* declaration with no *TASK* association
7	Declaration of directly represented variables in *VAR_GLOBAL*
8a	Connection of directly represented variables to *PROGRAM* inputs
8b	Connection of *GLOBAL* variables to *PROGRAM* inputs
9a	Connection of *PROGRAM* outputs to directly represented variables
9b	Connection of *PROGRAM* outputs to *GLOBAL* variables
10a	*VAR_ACCESS...END_VAR* construction
10b	Access paths to directly represented variables
10c	Access paths to *PROGRAM* inputs
10d	Access paths to *GLOBAL* variables in *RESOURCES*
10e	Access paths to *GLOBAL* variables in *CONFIGURATIONS*
10f	Access paths to *PROGRAM* outputs

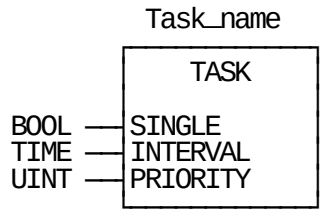
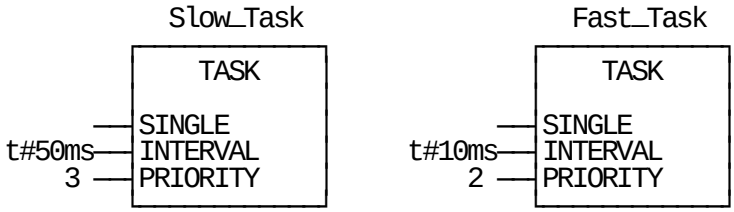
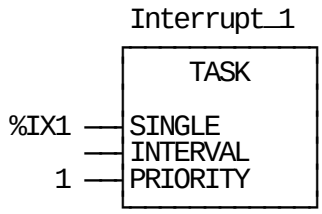
```
CONFIGURATION Configuration_name                    (* Declaration of Configuration *)
   TYPE                                             (* Types definitions *)
      ...
   END_TYPE
   VAR_GLOBAL                                       (* Global variables declarations *)
      ...
   END_VAR
   RESOURCE Resource_name_1 ON Hardware_name_1      (* Resources declaration *)
   RESOURCE Resource_name_2 ON Hardware_name_2
      ...
   VAR_ACCESS                                       (* Access paths declarations *)
      ...
   END_VAR
END_CONFIGURATION
```

Example 47. Consecutive elements in *CONFIGURATION* declaration

```
RESOURCE Resource_name ON Hardware_name             (* Resources declaration *)
   VAR_GLOBAL                                       (* Global variables declarations *)
      ...
   END_VAR
   TASK Task_name_1                                 (* Tasks declaration )
   TASK Task_name 2
      ...
   (* Programs instantiation and program to task association *)
   PROGRAM Program_instance_name_1 WITH Task_name_  : Program_name
   PROGRAM ... WITH ...
      ...
END_RESOURCE
```

Example 48. Consecutive elements in *RESOURCE* declaration

Table 54. Graphical representation of *TASK* within *RESOURCE*

Graphical form	Description
	General task representation
	Examples of periodic tasks
	Examples of non-periodic task

```
TASK Slow_Task      (INTERVAL:= t#50ms, PRIORITY:=3);
TASK Fast_Task      (INTERVAL:= t#5ms, PRIORITY:=2);
TASK Interrupt_1    (SINGLE:= %IX1, PRIORITY:=1);
```

Example 49. Textual representation for examples in Table 55.

- Start of *RESOURCE* within *CONFIGURATION – TASK* is *enabled*;
- Stopping *RESOURCE* within *CONFIGURATION – TASK* is *disabled.*

Table 55. Graphical representation of tasks - POUs association

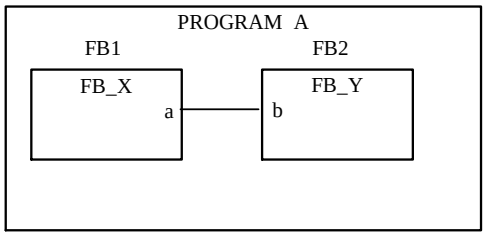
Graphical form	Description
RESOURCE STATION_1 (* Resource name *) Prog_1 Prog_4 (* Instances of programs *) Prog_A Prog_B (* Programs *) Task_1 Task_2 (* Associated tasks *) END_RESOURCE	Program–task association within a resource
RESOURCE STATION_2 (* Resource name *) Prog_1 (* Program instance *) Prog_C (* Program *) FB_1 FB_2 (* FB instances *) FB_A FB_B Task_1 Task_2 (* Associated tasks *) END_RESOURCE	FB–task association within program inside a resource

PROGRAM Prog_1 WITH Task_1 : Prog_A;
PROGRAM Prog_4 WITH Task_2 : Prog_B;
PROGRAM Prog_1 : Prog_C(FB_1 WITH Task_1, FB_2 WITH Task_2);

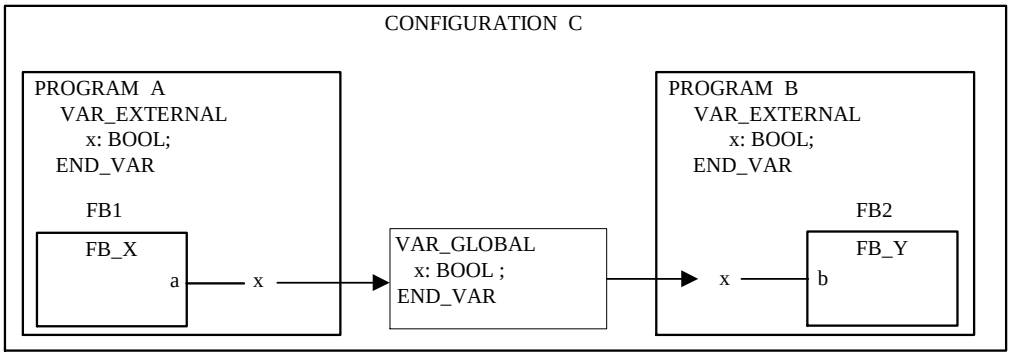
Example 50. Textual representation of tasks - POUs association

Communication model

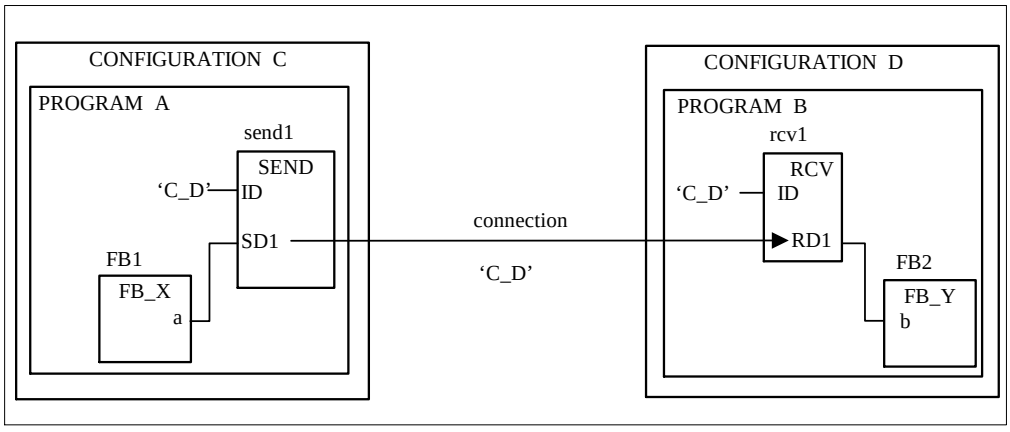
Data flow connection within a PROGRAM:



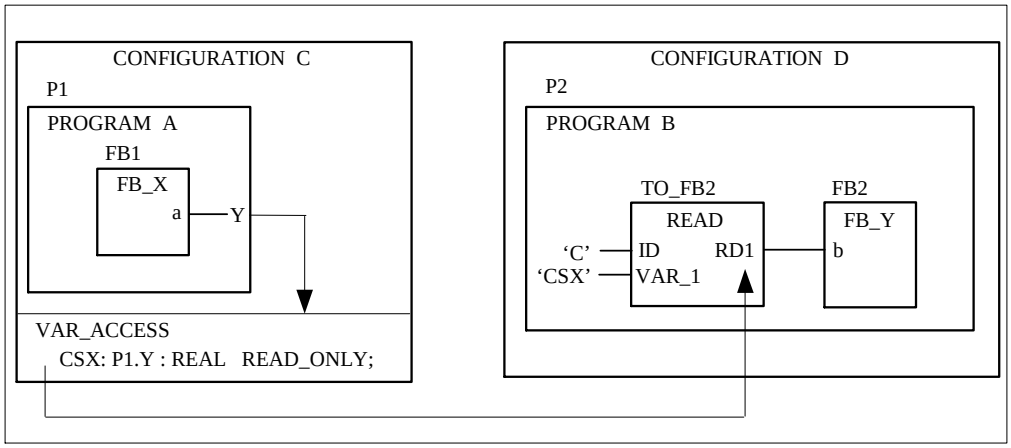
Communication via GLOBAL variables:



Communication function blocks (IEC 61131-5):



Communication via ACCESS paths:



Variables and access paths declaration

```
                                    PROGRAM Prog_A
  VAR_OUTPUT  YOUT : BOOL; END_VAR     (* output *)
  VAR_INPUT       XIN : INT;      END_VAR     (* input *)
     ...   (* etc., program declaration continued *)
                                    END_PROGRAM
```

Example 51. Program declaration with input and output parameters

```
                                    VAR_ACCESS
(* Access path          Variables accessible from          Data type and optionally
     names                    other resources                  read/write qualifier   *)
  ALPHA          :   STATION_1.%IX1.1          :     BOOL  READ_ONLY;
  BETA           :   STATION_1.PROG1.X2        :     UINT   READ_WRITE;
END_VAR
```

Example 52. Access paths declaration

Example of configuration

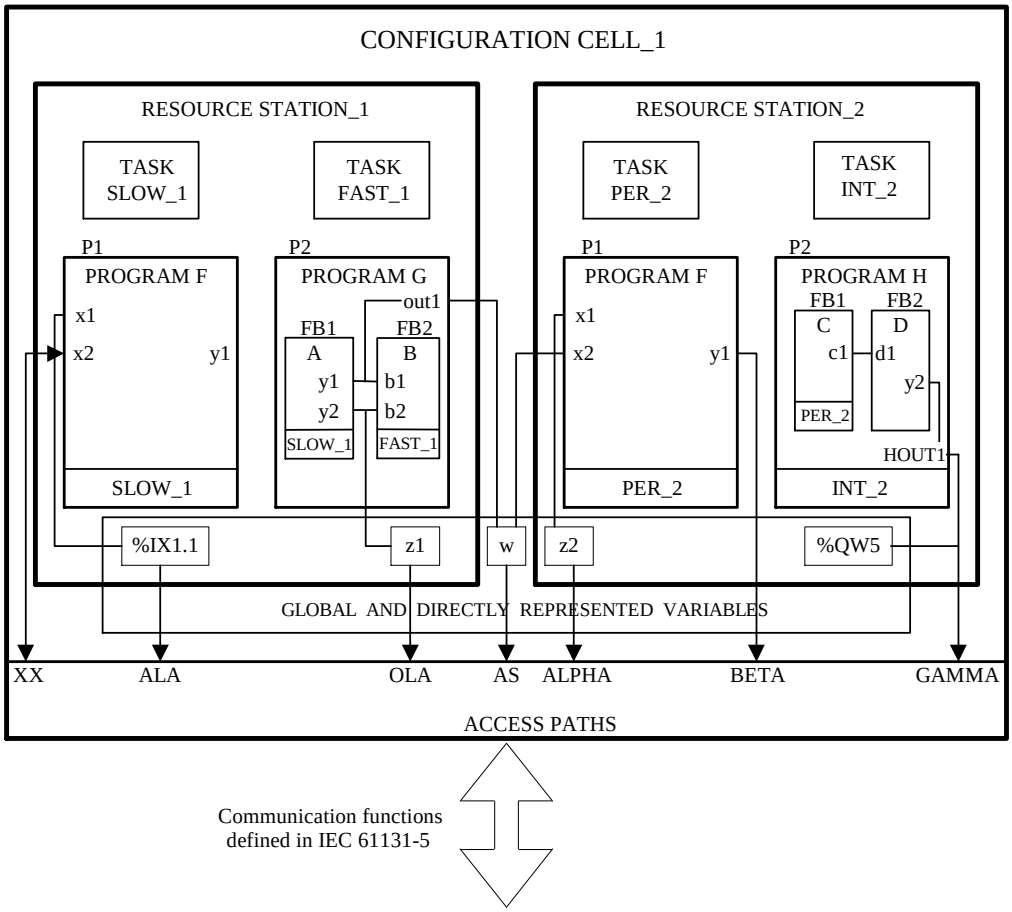


Figure 7. Graphical example of a configuration

```
PROGRAM F
   VAR_INPUT                          (* Inputs *)
     x1:BOOL;
     x2:UINT;
   END_VAR
   VAR_OUTPUT y1:BYTE; END_VAR      (* Output *)
       …                             (* Other declarations and program body *)
END_PROGRAM

PROGRAM G
   VAR_OUTPUT out1:UINT; END_VAR    (* Output *)
   VAR_EXTERNAL z1:BYTE; END_VAR    (* Declaration of global variable usage *)
   VAR FB1:A; FB2:B; END_VAR        (* FB instances *)
       …                             (* Other declarations and program body *)
   FB1(…);                           (* FB1 invocation *)
   out1 := FB1.y1;                   (* y1 from FB1 to program output *)
   z1 := FB1.y2;                     (* y2 from FB1 assigned to a global variable *)
      …
   FB2(b1 := FB1.y1, b2 := FB1.y2);  (* FB2 invocation *)
      …
END_PROGRAM

PROGRAM H
   VAR_OUTPUT HOUT1:INT; END_VAR  (* Output *)
   VAR FB1:C; FB2:D; END_VAR       (* FB instances *)
       …                           (* Other declarations and program body *)
   FB1(…);                         (* FB1 invocation *)
      …
   FB2(d1 := FB1.c1);              (* FB2 invocation *)
   HOUT1 := FB2.y2;                (* y2 from FB2 to program output *)
      …
END_PROGRAM
```

Example 53. Outline of programs declaration for example of *CONFIGURATION* (see Figure 7)

```
FUNCTION_BLOCK A
   VAR_OUTPUT                    (* Output *)
      y1:UINT;
      y2:BYTE;
   END_VAR
      …                          (* Other declarations and FB body *)
END_FUNCTION_BLOCK

FUNCTION_BLOCK B
   VAR_INPUT                     (* Inputs *)
      b1:UINT;
      b2:BYTE;
 END_VAR
      …                          (* Other declarations and FB body *)
END_FUNCTION_BLOCK

FUNCTION_BLOCK C
   VAR_OUTPUT  c1:BOOL;          (* Output *)
   END_VAR
      …                          (* Other declarations and FB body *)
END_FUNCTION_BLOCK

FUNCTION_BLOCK D
   VAR_INPUT  d1:BOOL;END_VAR    (* Input *)
   VAR_OUTPUT  y2:INT;END_VAR    (* Output *)
      …                          (* Other declarations and FB body *)
END_FUNCTION_BLOCK
```

Example 54. Broad outline of FBs declaration for configuration on Figure 7

```
CONFIGURATION CELL_1                                    (* feature 1 in Table 53. *)
  VAR_GLOBAL w : UINT; END_VAR                          (* feature 2 in Table 53. *)
  RESOURCE STATION_1 ON PROCESSOR_1                     (* feature 3 in Table 53. *)
    VAR_GLOBAL z1 : Byte; END_VAR                       (* feature 4 in Table 53. *)
    TASK SLOW_1(INTERVAL := t#20ms, PRIORITY := 2);     (* feature 5a in Table 53. *)
    TASK FAST_1(INTERVAL := t#10ms, PRIORITY := 1);     (* feature 5a in Table 53. *)
      PROGRAM P1 WITH SLOW_1 :                          (* feature 6a in Table 53. *)
      F(x1 := %IX.1);                                   (* feature 8a in Table 53. *)
      PROGRAM P2 : G(                                   (* feature 6c in Table 53. *)
                        OUT1 => w,                      (* feature 9b in Table 53. *)
                        FB1 WITH SLOW_1,                (* feature 6b in Table 53. *)
                        FB2 WITH FAST_1);               (* feature 6b in Table 53. *)
  END_RESOURCE                                          (* feature 3 in Table 53. *)
  RESOURCE STATION_2 ON PROCESSOR_2                     (* feature 2 in Table 53. *)
    VAR_GLOBAL z2 : BOOL;                               (* feature 4 in Table 53. *)
      AT %QW5 : INT;                                    (* feature 7 in Table 53. *)
    END_VAR
    TASK PER_2(INTERVAL := t#50ms, PRIORITY := 2);      (* feature 5a in Table 53. *)
    TASK INT_2(SINGLE := z2, PRIORITY := 1);            (* feature 5b in Table 53. *)
    PROGRAM P1 WITH PER_2 :                             (* feature 6a in Table 53. *)
       F(x1 := z2, x2 := w);                            (* feature 8b in Table 53. *)
    PROGRAM P2 WITH INT_2 :                             (* feature 6a in Table 53. *)
      H(HOUT1 => %QW5,                                  (* feature 9a in Table 53. *)
        FB1 WITH PER_2);                                (* feature 6b in Table 53. *)
  END_RESOURCE                                          (* feature 3 in Table 53. *)
  VAR_ACCESS                                            (* feature 10a in Table 53. *)
    XX    : STATION_1.P1.x2      : UINT READ_WRITE;     (* feature 10c in Table 53. *)
    ALA   : STATION_1.%IX1.1     : BOOL READ_ONLY;      (* feature 10b in Table 53. *)
    OLA   : STATION_1.z1         : BYTE;                (* feature 10d in Table 53. *)
    AS    : w                    : UINT READ_ONLY;      (* feature 10e in Table 53. *)
    ALPHA : STATION_2.z2         : BOOL READ_WRITE;     (* feature 10d in Table 53. *)
    BETA  : STATION_2.P1.y1      : BYTE READ_ONLY;      (* feature 10f in Table 53. *)
    GAMMA : STATION_2.P2.HOUT1   : INT READ_ONLY;       (* feature 10f in Table 53. *)
  END_VAR                                               (* feature 10a in Table 53. *)
END_CONFIGURATION                                       (* feature 1 in Table 53. *)
```

Example 55. Declarations for configuration and resources

Execution times:

- *STATION_1: P1 – 2 ms, P2 – 8 ms* (without execution times of *FB1* and *FB2* because *FB1* and *FB2* are associated with tasks *SLOW_1* and *FAST_1*, while *P2* is not associated with any task), *FB1* and *FB2 – 2 ms* each;
- *STATION_2: P1 – 30 ms, P2 – 5 ms* (without execution time of *FB1* because *FB1* is associated with task *PER_2*, while *P2* with task *INT_2*), *FB1* in *P2 – 10 ms*.

Table 56. Task features – examples of non-preemptive scheduling

Example for *STATION_1*			
t (ms)	Running	Waiting	Comment
0	P2.FB2@1	P1@2, P2.FB1@2, P2	
2	P1@2	P2.FB1@2, P2	
4	P2.FB1@2	P2	
6	P2	-	
10	P2	P2.FB2@1	P2 yet not finished
14	P2.FB2@1	P2	FB2 restarts with delay
16	P2	-	P2 restarts
20	P2	P2.FB2@1, P1@2, P2.FB1@2	P2 yet not finished
24	P2.FB2@1	P1@2, P2.FB1@2, P2	
26	P1@2	P2.FB1@2, P2	
28	P2.FB1@2	P2	
30	P2.FB2@1	P2	
32	P2		P2 restarts
40	P2.FB2@1	P1@2, P2.FB1@2, P2	
Example for *STATION_2* Task *INT_2* is triggered at *t = 25, 50, 90, ... ms*			
t (ms)	Running	Waiting	Comment
0	P1@2	P2.FB1@2	
25	P1@2	P2.FB1@2, P2@1	*z2*:=1, P1 yet not finished
30	P2@1	P2.FB1@2	P2 starts
35	P2.FB1@2	-	
45	-	-	P2.FB1 finished
50	P2@1	P1@2, P2.FB1@2	*z2*:=1, P2 starts
55	P1@2	P2.FB1@2	P1 starts with delay
85	P2.FB1@2		
90	P2.FB1@2	P2@1	*z2*:=1, FB1 yet not finished
95	P2@1		P2 starts

100	P1@2	P2.FB1@2	

Table 57. Task features – examples of preemptive scheduling

Example for *STATION_1*			
t (ms)	Running	Waiting	Comment
0	P2.FB2@1	P1@2, P2.FB1@2, P2	
2	P1@2	P2.FB1@2, P2	
4	P2.FB1@2	P2	
6	P2	-	P2 starts
10	P2.FB2@1	P2	P2 interrupted
12	P2	-	P2 continued
16	P2	-	P2 restarts
20	P2.FB2@1	P1@2, P2.FB1@2, P2	P2 interrupted
Example for *STATION_2* Task *INT_2* is triggered at *t = 25, 50, 90, ... ms*			
t (ms)	Running	Waiting	Comment
0	P1@2	P2.FB1@2	
25	P2@1	P1@2, P2.FB1@2	P1 interrupted
30	P1@2	P2.FB1@2	P1 continued
35	P2.FB1@2	-	
45	-	-	P2.FB1 finished
50	P2@1	P1@2, P2.FB1@2	*z2*:=1, P2 starts
55	P1@2	P2.FB1@2	
85	P2.FB1@2		
90	P2@1	P2.FB1@2	P2.FB1 interrupted
95	P2.FB1@2		P2.FB1 continued
100	P1@2	P2.FB1@2	