TwinCAT-Training: Программирование

Beckhoff
Industrial PC

Beckhoff
TwinCAT

Beckhoff
Lightbus

Beckhoff
Embedded PC

Beckhoff Bus
Terminal

Beckhoff
Fieldbus Box

Beckhoff
PC Fieldbuscards,
Switches

Beckhoff
EtherCAT

Beckhoff Drive
Technology

Содержание

Стандарт IEC 61131-3

1 Основные определения и типовые базовые функции. (циклические процессы, входной и выходной образ процесса)

2 Условия окружающей среды и классы управления и программирования устройств. (температура, влажность воздуха)

3 Правила применения языков программирования ПЛК

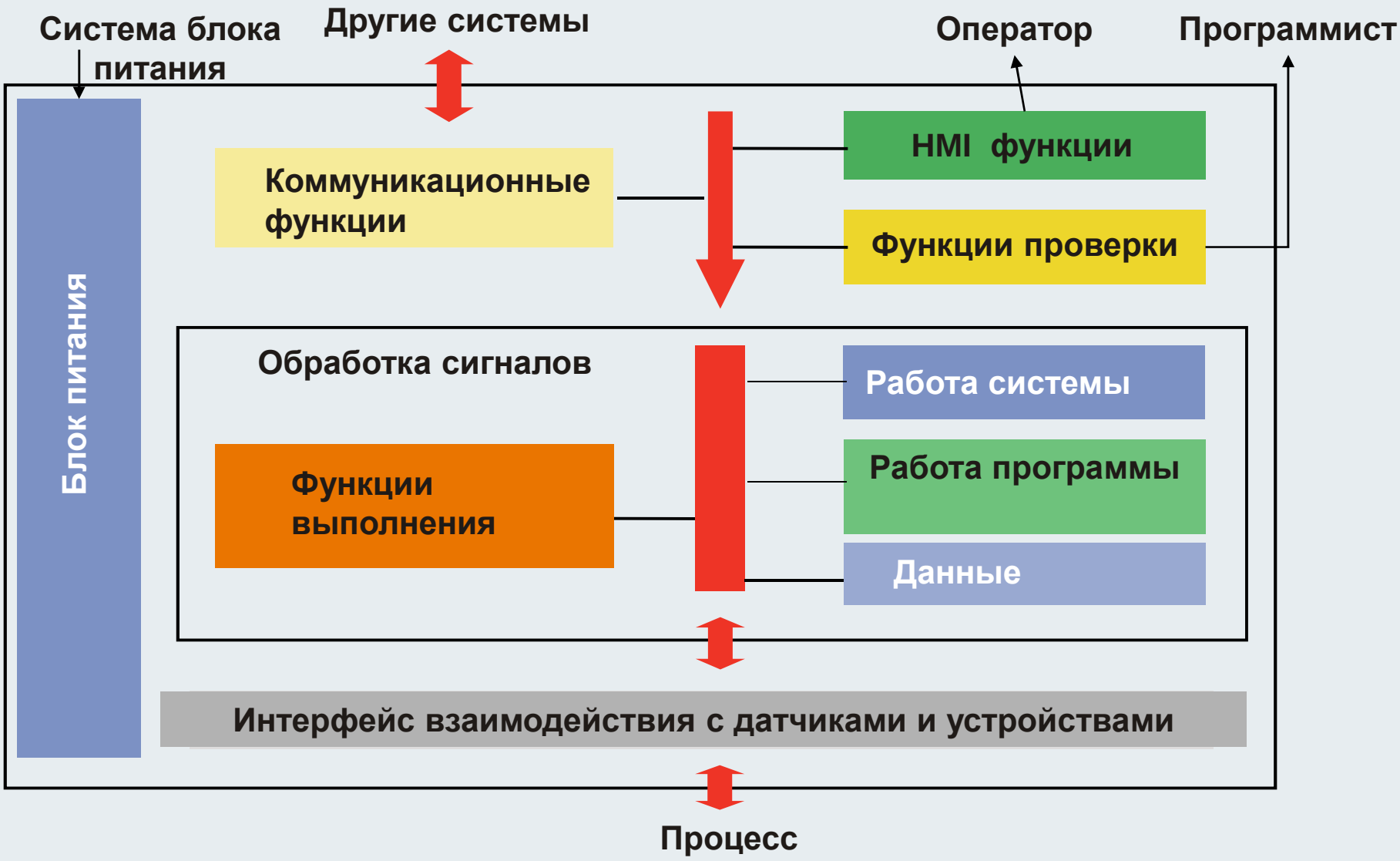
4 Положения по пользовательской системной аналитике, выборе систимы, реализации приложений, а также настройка и обслуживание

5 Определение коммуникации через ФБ и коммуникации через пути доступа (дополнительно до 3)

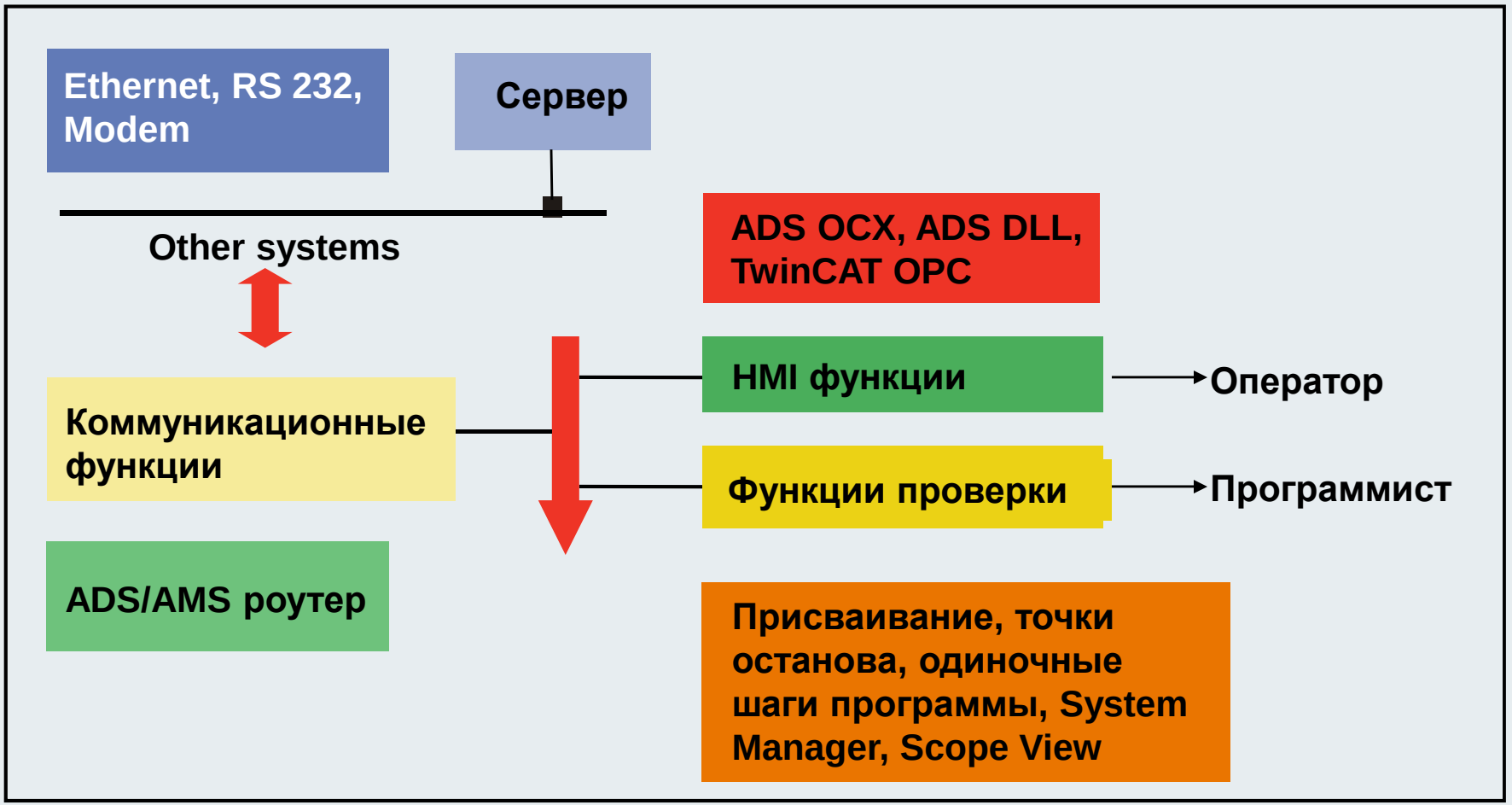
6 Коммуникация по промышленным шинам

7 Системы нечеткой логики в ПЛК

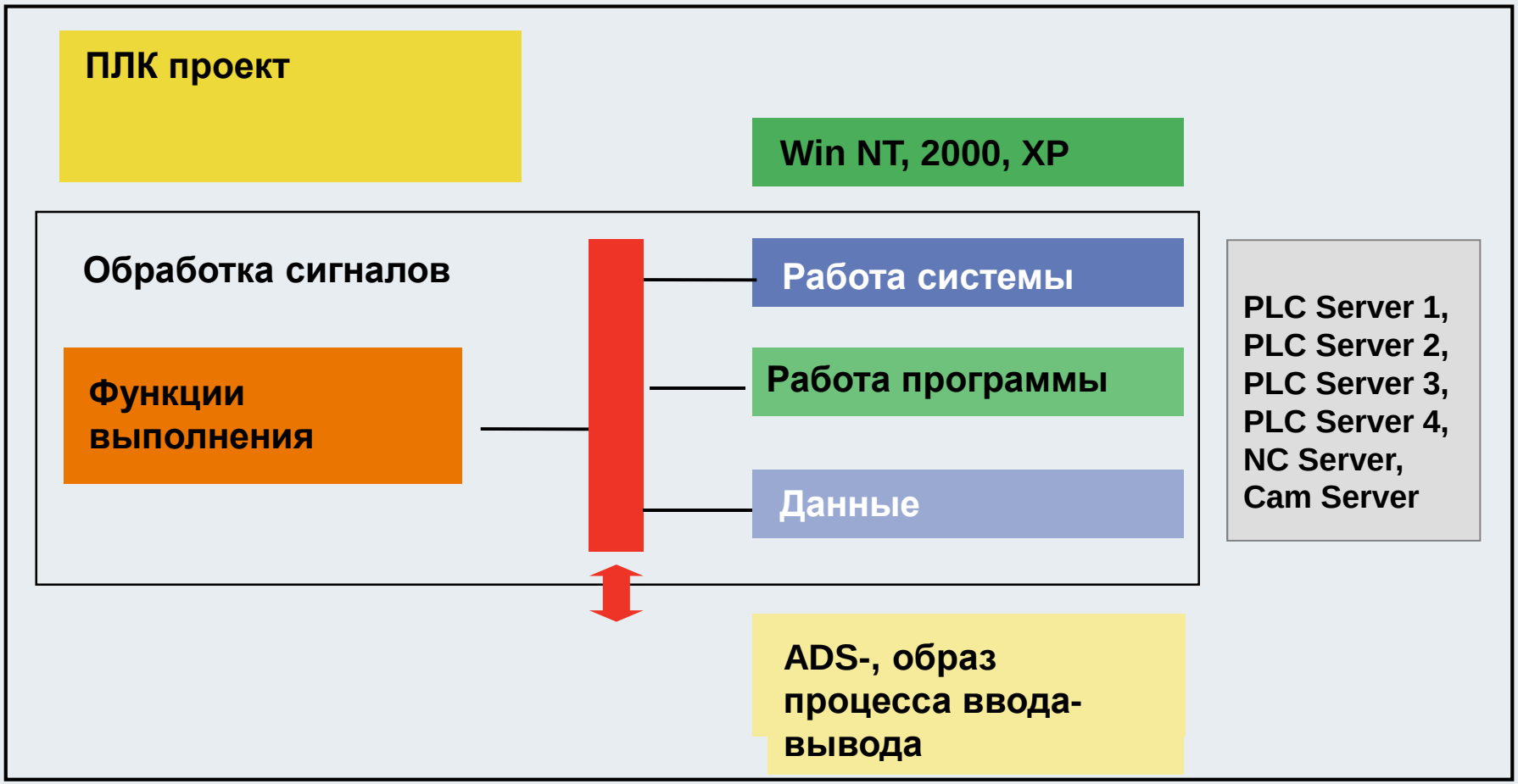
Функциональная структура ПЛК



Коммуникационные функции

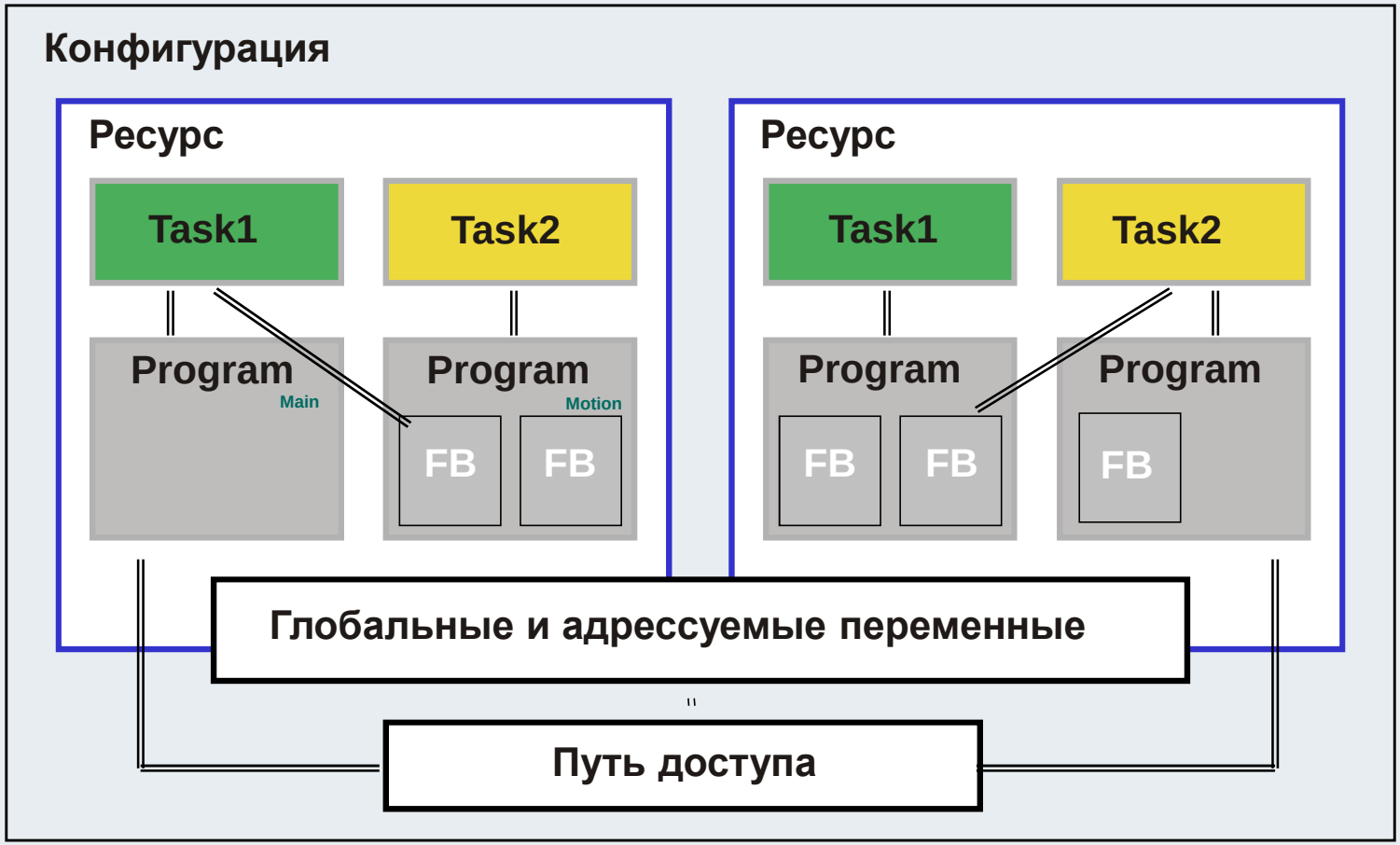


Обработка сигналов



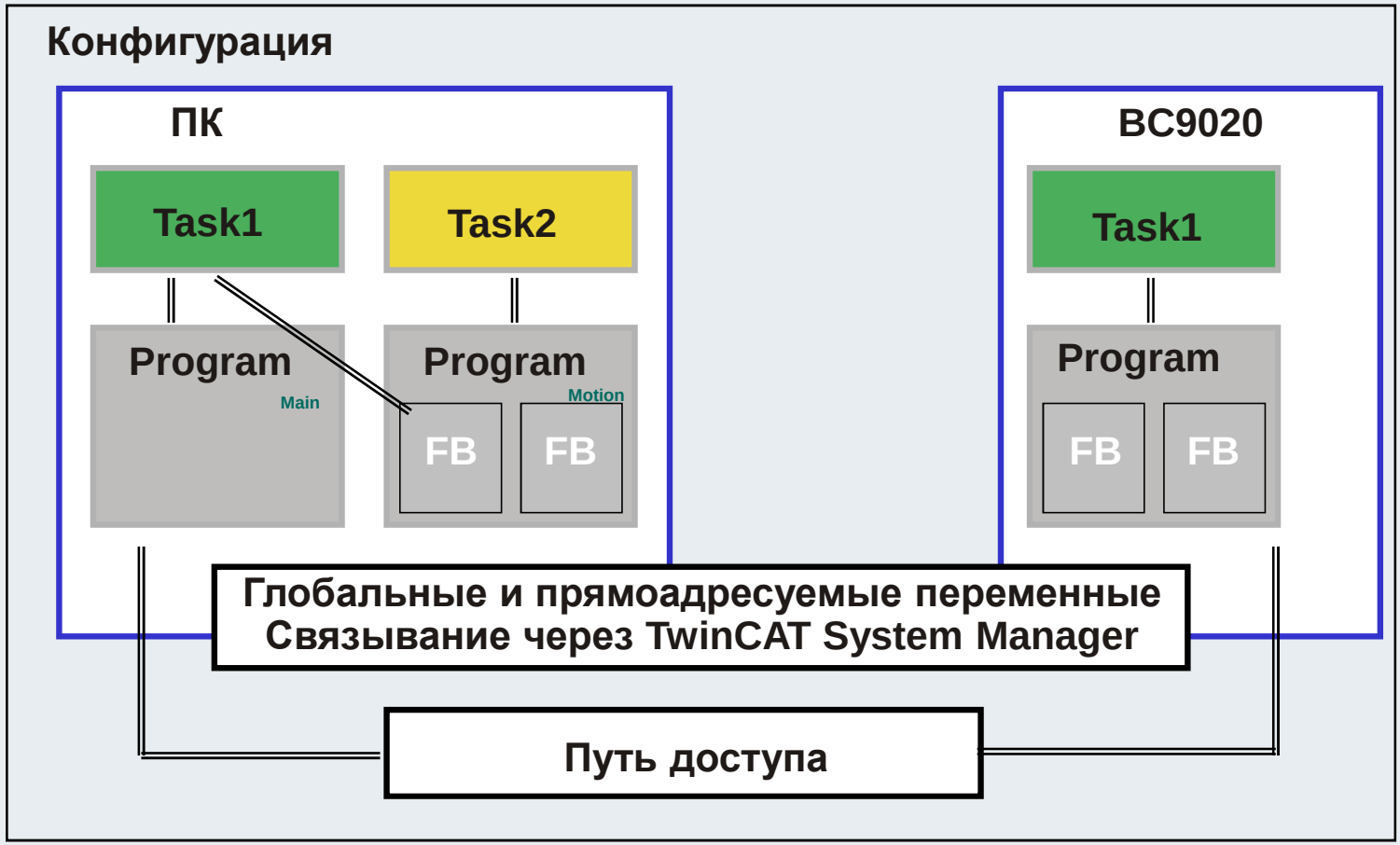
Интерфейс датчиков и устройств

Программная модель



Программная модель - пример

Пример: ПК ПЛК с 1 run time и двумя задачами, 1 BC9020 (Ethernet контроллер)



Идентификатор

Идентификатор используется для индивидуального назначения имени переменных, типов данных, функций …

- Идентификатор начинается с буквы или символа подчеркивания
- Далее используются буквы, цифры или символы подчеркивания
- Нет различий между большими и маленькими буквами

Запрещается

- Специальные символы (!,“,§,$..)
- Пробелы
- Несколько подчеркиваний подряд
- Невидимые символы

Префикс

Существует несколько методов использования префиксов в программировании.

Несколько рекомендаций:

Венгерская запись: Запись нескольких частей слов вместе. Первая буква каждой части слова должна быть заглавной.

b – Booleen	bEndschalterLinks	
r – Real	rSollPosition	
s - String	sRxDaten	
ST_ - объявление структуры		
st – инициализация структуры	ST_MotorDaten	(declaration)
FB_ - объявление ФБ	stM1Parameter	(instance)
fb – инициализация ФБ		
M_ - объявление метода	FB_Ueberlast	(declaration)
	fbM1Ueberlast	(instance)

Ключевые слова и команды

- Ключевые слова определены стандартоми IEC61131-3. Они используются в синтаксисе языка и запрещается их использовать для других целей.

  TRUE, FALSE, AND, FUNCTION,...

  Ключевые слова всегда записываются заглавными буквами. Для автоформатирования есть опция редактора „AutoFormat“
- Комментарии заключаются в символы (* в начале и *) в конце. Комментарии могут размещатся в там, где разрешен символ пробел.
  Исключение: внутри инициализации строки.

```
(*digital inputs*)
bStart AT%IX0.0:BOOL;(*machine start*)
(*analog inputs*)
TemK1 AT%IW10(*Byte 10-11*):WORD;
```

Базовые типы данных

Тип	ANY-Type	Ключевое слово	Размер (Bit)	Нач. знач.	Диапазон
Boolean	ANY_Bit	BOOL	1	FALSE	TRUE/FALSE
Bit string (8)		BYTE	8	0	0..16#FF
Bit string (16)		WORD	16	0	0..16#FFFF
Bit string (32)		DWORD	32	0	0..16#FFFF_FFFF
Short integer	ANY_Num	SINT	8	0	$-2^7...2^7-1$
Integer		INT	16	0	$-2^{15}...2^{15}-1$
Double integer		DINT	32	0	$-2^{31}...2^{31}-1$
Unsigned short integer		USINT	8	0	$0...2^8-1$
Unsigned integer		UINT	16	0	$0...2^{16}-1$
Unsigned double integer		UDINT	32	0	$0...2^{32}-1$

Базовые типы данных

Тип	ANY-Type	Ключевое слово	Размер (Bit)	Нач. знач.	Диапазон
Real	ANY_Real	REAL	32	0.0	$-1.18*10^{-38}$.. $3.4*10^{38}$
Long real		LREAL	64	0.0	$-2.22*10^{-308}$.. $1.798*10^{308}$
Date	ANY_Date	DATE (D)	32	D#1970-01-01	
Time of day		TIME_OF_DAY (TOD)	32	TOD#00:00	TOD#00:00.. TOD#23:59
Date Time		DATE_AND_TIME (DT)	32	DT#1970-01-01-00:00	
Time	ANY_Time	TIME	32	T#0ms	T#0ms..T#71582m47s295ms
String	ANY_String	STRING	(80+1)*8	“	

Константы

Тип переменной	Примеры		
BOOL	0,1	16#0, 16#1	TRUE (equal)
BYTE, WORD, DWORD	10	16#0A	2#1010 (equal)
DWORD, UINT	32768	16#8000	2#1000_0000 (equal)
INT	-10	---	---
TIME	T#1h2m4s11ms,	T#62m4s11ms,	T#3724011ms (equal)
REAL, LREAL	0.22, 2.2e-1 (equal) 1000, 1000.0, 1e3 (equal)		
STRING	‘‘ пустая строка, ‘Текст‘ ‘Text$0D$0A‘, ‘Text$R$N‘ (равнозначны) Текст с управляющими символами		

Строка

Тип переменной STRING может содержать любое количество символов. Размер определенный при объявлении строки указывает сколько места в памяти требуется зарезервировать под переменную.

Если не указан размер (от 1 до 255) , то по умолчанию используется строка в 80 символов.

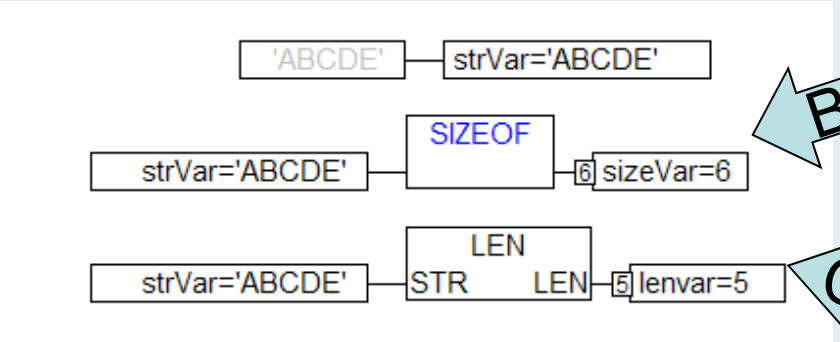
```
VAR
    strVar :STRING(5);
    lenVar: INT;
    sizeVar: INT;
END_VAR
```

Строки терминируются нулем, это означает, что последний символ в строке ноль. Каждый символ строки требует один байт.

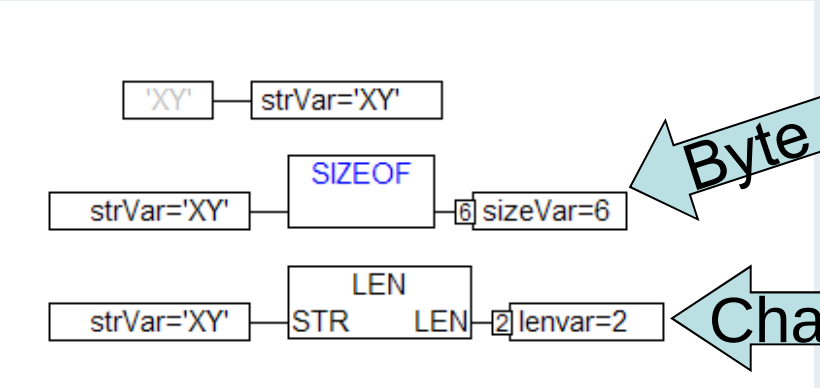
Строка

Нулевое терминирование, LEN и SIZEOF

Память ПЛК



65
66
67
68
69
0



88
89
0
68
69
0

```
VAR
    strVar :STRING(5);
    lenVar: INT;
    sizeVar: INT;
END_VAR
```

LEN определяет кол-во символов в строке

SIZEOF определяет размер переменной в байтах

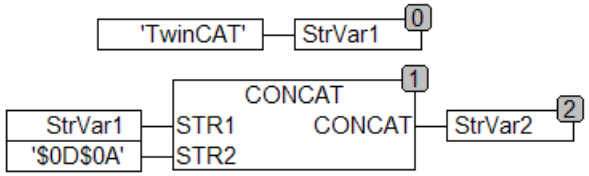
Специальные символы

Если требуется добавить специальный символ в строку, он должен начинаться с символа $.

Специальные символы

символ	описание
$$	dollar signs
$‘	single quotation mark
$L or $l	line feed
$N or $n	new line
$P or $p	page feed
$R or $r	line break
$T or $t	tab

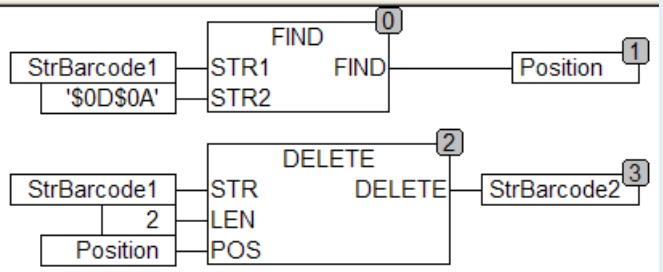
```
0001 PROGRAM MAIN
0002 VAR
0003     StrVar1 :STRING;
0004     StrVar2 :STRING;
0005
0006 END_VAR
```



Добавить„CR и LF“

StrVar1 = 'TwinCAT'
StrVar2 = 'TwinCATRN'

```
0001 PROGRAM MAIN
0002 VAR
0003     StrBarcode1 :STRING:='12368432$0D$0A';
0004     StrBarcode2 :STRING;
0005     Position: INT;
0006 END_VAR
```



Найти и удалить „CR и LF“

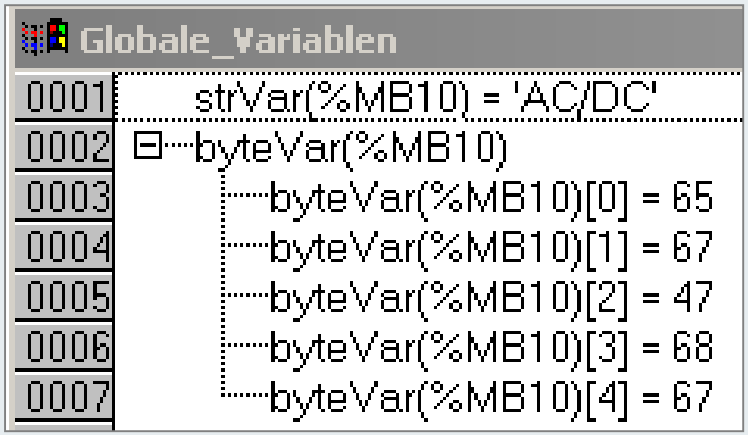
StrBarcode1 = '12368432RN'
StrBarcode2 = '12368432'
Position = 9

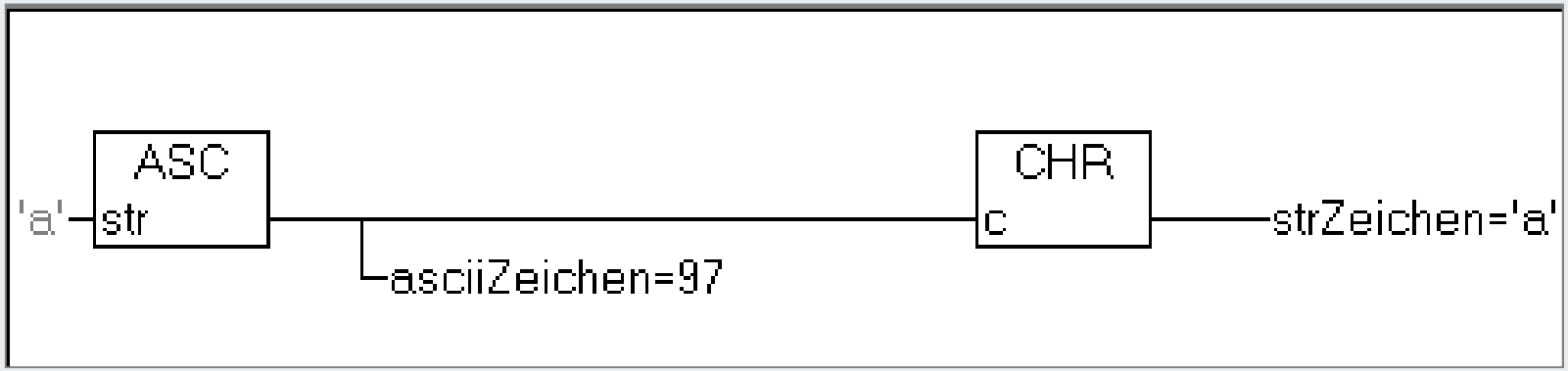
ASCII <-> CHR

Если символ в программе требуется конвертировать в ASCII символ,

Разрешены две процедуры:

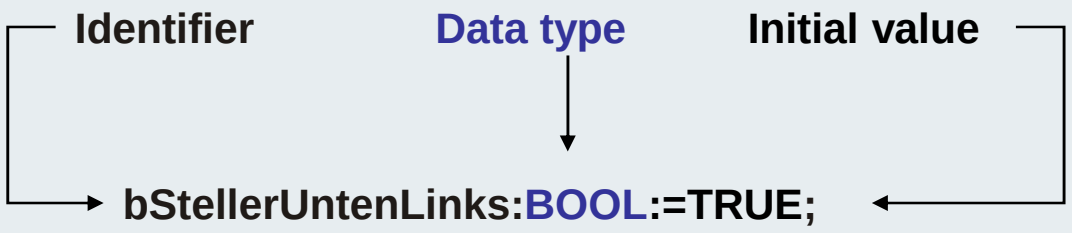
1. Напрямую в памяти изменить значение.
2. Использовать ФБ ASC и CHR, которые находятся в библиотеке ChrAsc.lib.





Объявление переменных базового типа

Переменным можно присваивать начальные значения после объявления имени и типа. В процессе выполнения программы это значение может изменяться.

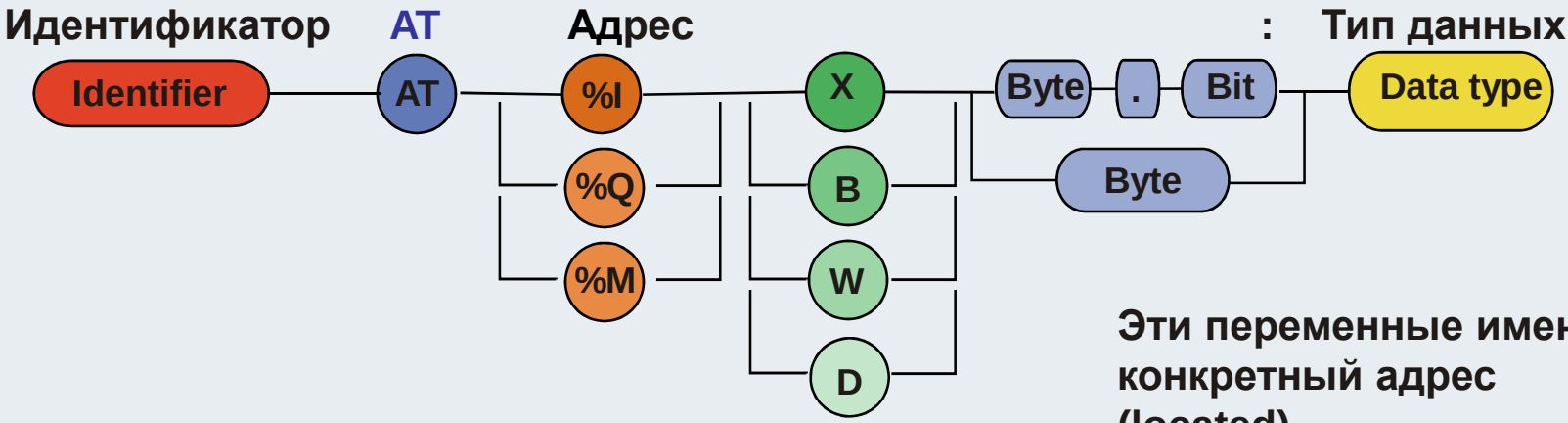


Физическое расположение переменной не известно оператору (unlocated)

Ограничения при назначении идентификатора можно найти на предыдущих слайдах.

Адресные переменные

При объявлении переменной ее можно разместить по определенному адресу. Для связывания входов и выходов с символьными переменными, расположение переменных имеет важное значение.



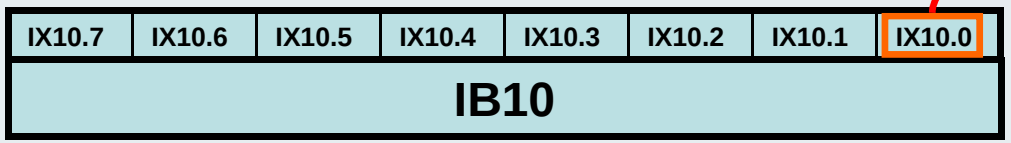
Эти переменные имеют конкретный адрес (located)

bStellerUntenLinks AT%IX0.0:BOOL;

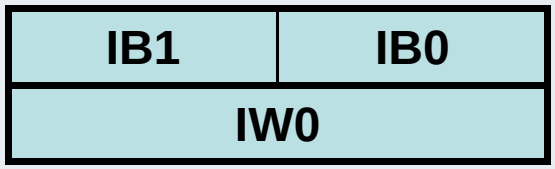
С версии TwinCAT 2.8 возможна автоматическая адресация. Тогда программа работает не с полностью определенным адресом переменной. bStellerUntenLinks AT%I*:BOOL;

Назначение адреса

Примеры:



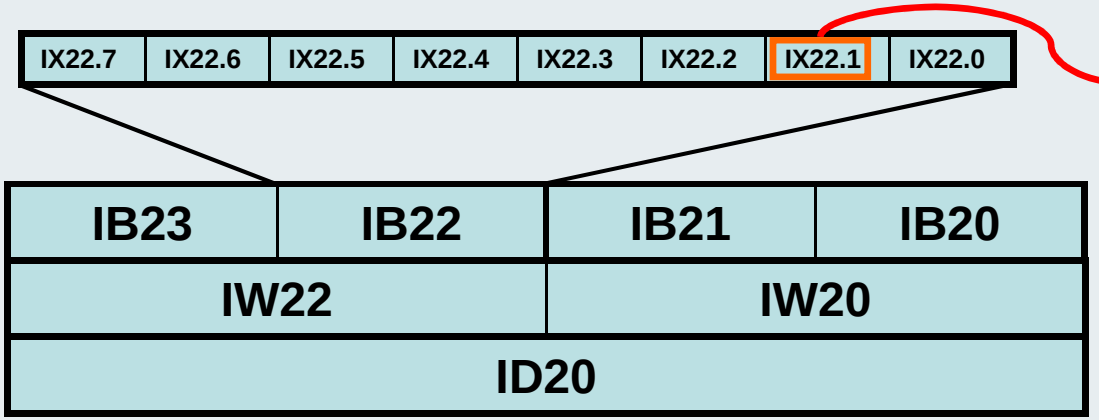
Din0 AT%IX10.0 : BOOL;



Ain AT%IB0 : INT;

equal

Ain AT%IW0 : INT;



BitVar AT%IX22.1 : BOOL;

Posi AT%IB20 : UDINT;

equal

Posi AT%ID20 : UDINT;

Классы переменных, допустимые диапазоны

Локальные переменные используются в программной секции где были объявлены.

Ключевые слова

VAR ..

END_VAR
VAR_INPUT ..

END_VAR
VAR_IN_OUT ..

END_VAR
VAR_OUTPUT ..

END_VAR

Глобальные переменные «известны» всему ПЛК проекту.

Ключевые слова

VAR_GLOBAL ..

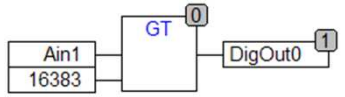
END_VAR
VAR_CONFIG ..

END_VAR

Пример VAR_CONFIG

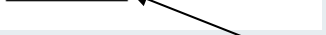
В функциональных блоках входы и выходы должны быть локальными переменными.

```
0001 FUNCTION_BLOCK Station
0002 VAR_INPUT
0003 END_VAR
0004 VAR_OUTPUT
0005 END_VAR
0006 VAR
0007     DigIn0 AT%I* :BOOL;
0008     DigIn1 AT%I* :BOOL;
0009     DigIn2 AT%I* :BOOL;
0010     DigIn3 AT%I* :BOOL;
0011
0012     DigOut0 AT%Q* :BOOL;
0013     DigOut1 AT%Q* :BOOL;
0014     DigOut2 AT%Q* :BOOL;
0015     DigOut3 AT%Q* :BOOL;
0016
0017     Ain1 AT%I*:INT;
0018 END_VAR
```



Создание блока

```
0002 VAR
0003     Station1: Station;
0004     Station2: Station;
0005 END_VAR
0006
```



Вызов блока

```
VAR_CONFIG
    MAIN.Station1.DigIn0 AT %IX0.0 : BOOL;
    MAIN.Station1.DigIn1 AT %IX0.1 : BOOL;
    MAIN.Station1.DigIn2 AT %IX0.2 : BOOL;
    MAIN.Station1.DigIn3 AT %IX0.3 : BOOL;
    MAIN.Station1.DigOut0 AT %QX0.0 : BOOL;
    MAIN.Station1.DigOut1 AT %QX0.1 : BOOL;
    MAIN.Station1.DigOut2 AT %QX0.2 : BOOL;
    MAIN.Station1.DigOut3 AT %QX0.3 : BOOL;
    MAIN.Station1.Ain1 AT %IB1 : INT;
    MAIN.Station2.DigIn0 AT %IX3.0 : BOOL;
    MAIN.Station2.DigIn1 AT %IX3.1 : BOOL;
    MAIN.Station2.DigIn2 AT %IX3.2 : BOOL;
    MAIN.Station2.DigIn3 AT %IX3.3 : BOOL;
    MAIN.Station2.DigOut0 AT %QX0.4 : BOOL;
    MAIN.Station2.DigOut1 AT %QX0.5 : BOOL;
    MAIN.Station2.DigOut2 AT %QX0.6 : BOOL;
    MAIN.Station2.DigOut3 AT %QX0.7 : BOOL;
    MAIN.Station2.Ain1 AT %IB4 : INT;
END_VAR
```

Mapping входов и выходов в экземпляре блока.

Этот список создается в ручную или автоматически в TwinCAT System.

Доступ к локальной переменной

Из программы A прямой доступ к локальной переменной ‚locVar‘ программы B возможен по адресу %MB2.

Проект программы:

```
PROGRAM A
VAR
END_VAR
```

```
LD   %MB2
▪
▪
```

```
PROGRAM B
VAR
locVar AT%MB2:WORD;
END_VAR
```

```
▪
▪
▪
▪
```

Разрешенное наложение имен переменных

Проект программы:

```
VAR_GLOBAL
    Var1:WORD;
END_VAR
```

```
PROGRAM A
VAR
   Var1 :WORD;
END_VAR
```

```
LD   Var1
▪
▪
```

На примере происходит наложение имен Var1.

В этом случае локальная переменная Var1 будет загружена в аккумулятор.

Компилятор на выдаст ошибки при проверке!

Атрибуты

Атрибуты используются для определения специальных свойств переменной.

Пример:

Переменные должны сохраняться при выключении ПЛК и загружаться при новом старте.

```
VAR RETAIN
        Counter:UINT;
END_VAR
```

При загрузке измененного проекта обнуляются

```
VAR PERSISTENT
        Counter:UINT;
END_VAR
```

При загрузке измененного проекта остаются

Атрибуты

Начальные значения
устанавливаемые при старте или сбросе ПЛК.

```
VAR
    AccelerationTime  : TIME := T#3s200ms;
END_VAR
```

Атрибуты (константы)

Проект

```
VAR_GLOBAL CONSTANT
  ▪
END_VAR
```

```
PROGRAM A
VAR CONSTANT
  ▪
END_VAR
```

```
  ▪
  ▪
  ▪
```

Если Вы хотите использовать математические, конструкторские и пр. надо объявить их конструкцией VAR_GLOBAL .. END_VAR с ключевым словом CONSTANT.

Константы доступны только для ЧТЕНИЯ!

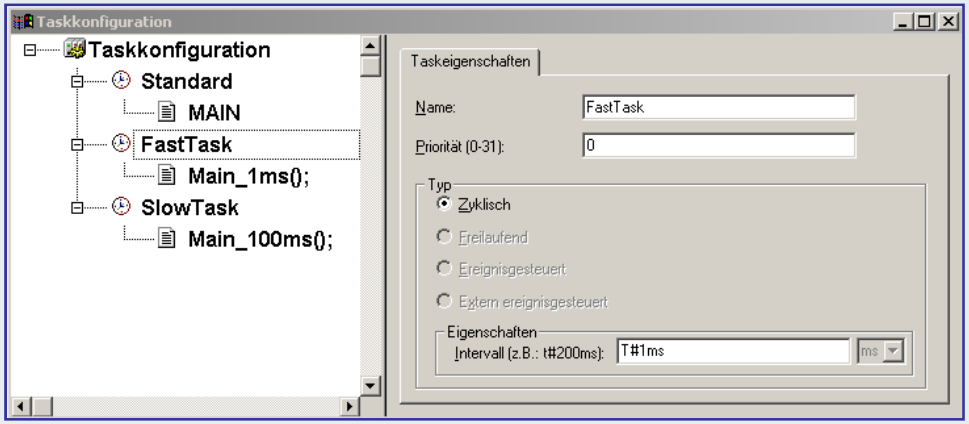
```
VAR_GLOBAL CONSTANT
    pi:REAL:=3.141592654;
END_VAR
```

Задачи и программные модули (program organisation units)

Стандарт IEC 61131-3 содержит три вида программных модулей POU`s (PROGRAM ORGANISATION UNIT):

- Program
- Function block
- Function

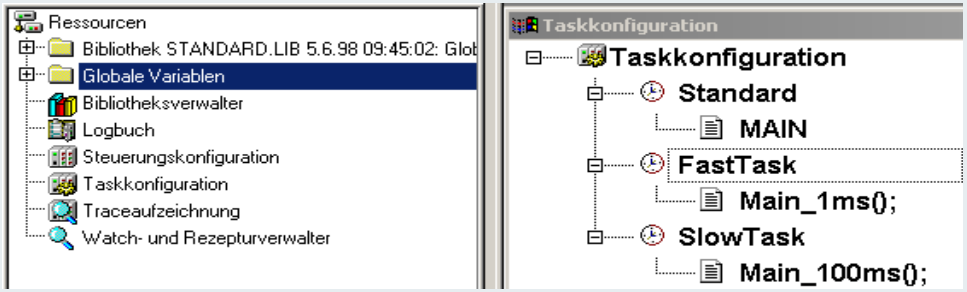
Вызов *POU настраивается в* task configurator.



Программа PRG

Программа PRG

- Вызов из конфигуратора задач (одна программа вызывает другую)
- вызовы: ФБ, Функций, (программ)
- Локальные переменные: статичные т.е. локальные данные доступны в следующем цикле.
- Входы: обычно нет, но VAR_INPUT возможны
- Выходы: обычно нет, but VAR_OUTPUT возможны
- Изменение входных данных VAR_IN_OUT возможны
- Отладка: Локальные данные напрямую видимы в режиме online в PLC Control



Функциональный блок FB

Функциональный блок FB

- Вызывается программой или другим ФБ
- Вызывает: ФБ или функции
- Локальные переменные: статичные т.е. локальные данные доступны в следующем цикле. Каждый ФБ имеет собственные данные.
- Входы: 0,1,2,3 VAR_INPUT
- Выходы: 0,1,2,3 VAR_OUTPUT
- Изменение входных данных: 0,1,2,3 VAR_IN_OUT
- Мониторинг: В режиме online в PLC Control открывается экземпляр ФБ. После чего переменные доступны для просмотра каждый цикл.
- Использование: многократное использование ФБ, каждый имеет собственные данные.

Функции: FC

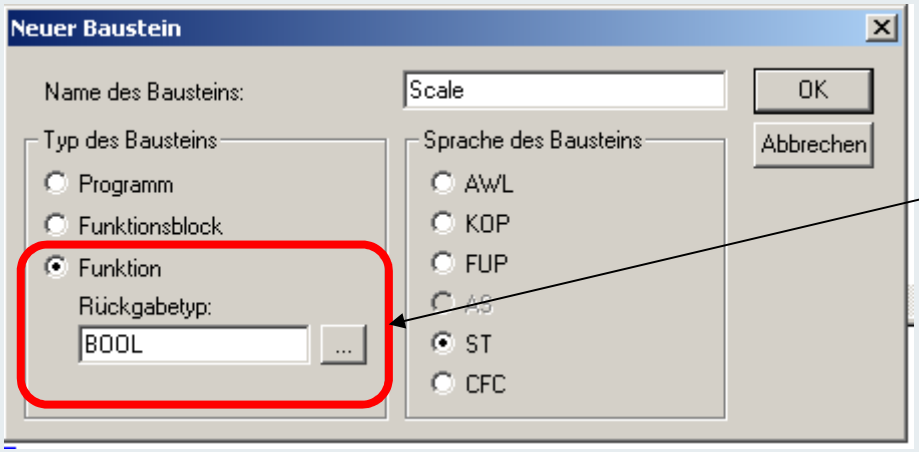
Функции : FC

- Вызывается программой, ФБ или другой функцией
- вызывает: функции
- Локальные переменные: временные, т.е доступны только во время выполнения функции.
- Входы: 1,2,3..... VAR_INPUT
- Выходы: только 1!, но структурная переменная возможна. Имя выхода является и именем функции.
- Изменение входных данных: 0,1,2,3 VAR_IN_OUT,
- Отладка: Локальные переменные в режиме online в PLC Control всегда „???“, т.к. в цикле ПЛК данные могут изменяться при каждом вызове функции.

Подсказка: Отладка с точками останова (breakpoints)

- Использование: Алгоритны, в которых результат даступен сразу по завершению. Масштабирование, сравнение.....

FC особенности



При создании функции тип возвращаемого значения задается сразу

Имя функции

Входы

Локальные переменные доступны только во время выполнения функции

```
FUNCTION Scale : REAL
VAR_INPUT
    x:                  REAL;
    xug, xog, yug, yog: REAL;
END_VAR
VAR
    Scalierung, Offset :    REAL;
END_VAR
Scalierung:= (yog-yug)/(xog-xug);
Scale:= Scalierung * x + Offset;
```

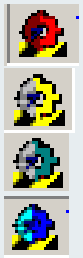
Возвращаемое значение

Имя выходной переменной - Scale.

Scale может использоваться как локальная переменная (Write/Read)

REAL TIME, TwinCAT System Service

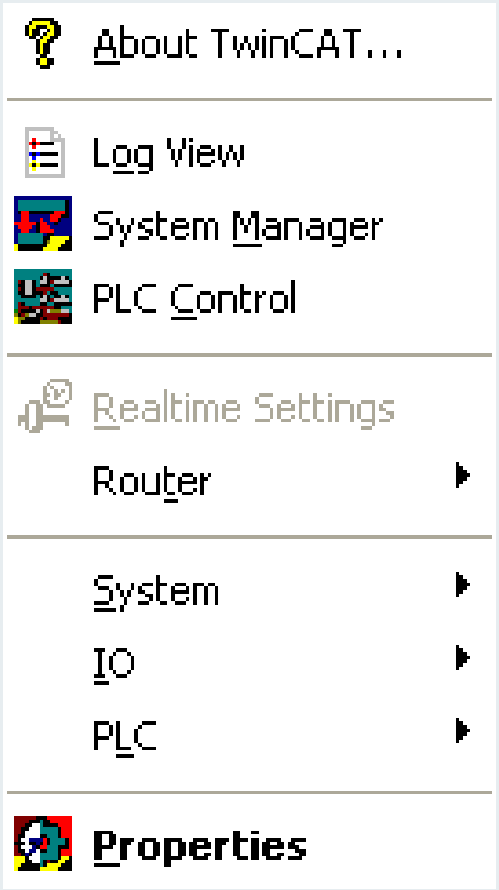
The TwinCAT System Service работает как пользовательский Windows NT сервис. Поэтому, TwinCAT System Service запускается Windows NT после авторизации. Иконка работы с TwinCAT System Service находится в task bar рабочего стола. В дополнении, ее цвет показывает состояние системы TwinCAT.



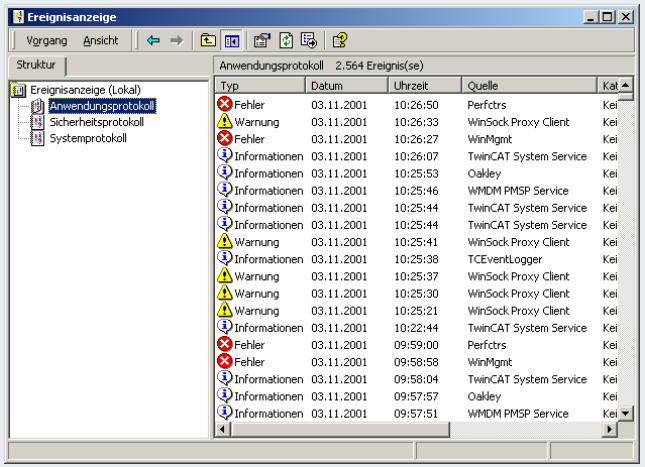
TwinCAT stopped
TwinCAT starting
TwinCAT running
TwinCAT Config Mode

TwinCAT System Service отвечает за старт и останов TwinCAT run time системы. Он загружает все сконфигурированные сервисы и инициализационные значение во время старта TwinCAT.

TwinCAT System Service



Log View – программа мониторинга событий в системе. Event logging service стартует автоматически, при запуске Windows NT.



Подсистема TwinCAT I/O может быть обнулена через TwinCAT System Service. Для этого вызывается функция меню. Сброс применяется ко все подсистемам ввода-вывода.

Управление производительностью ПК

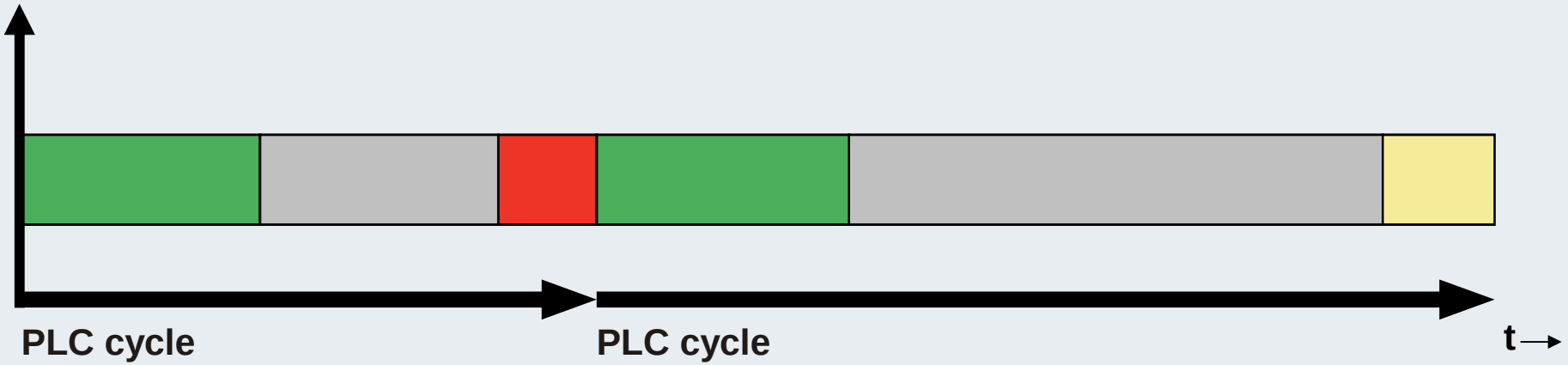
Real time программного ПЛК на классическом контроллере

Чтение входов

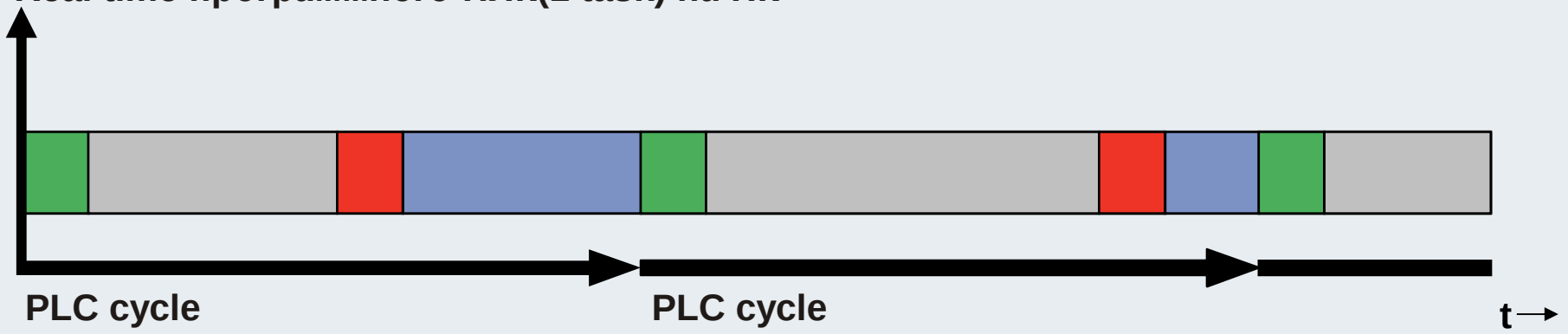
Работа программы

Запись выходов

Win NT & HMI



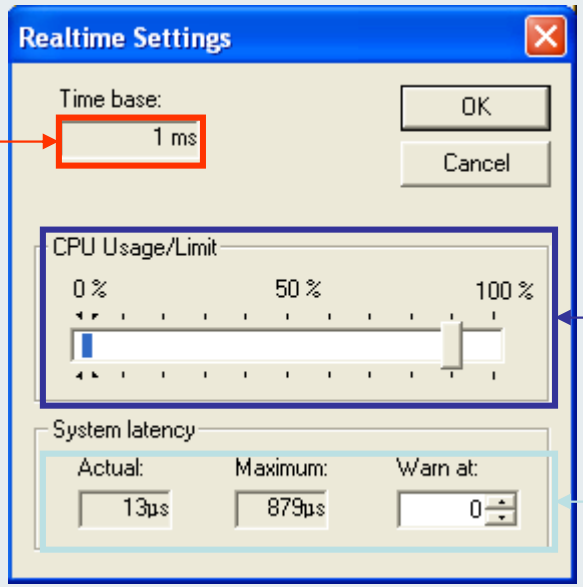
Real time программного ПЛК(1 task) на ПК



Регулятор использования времени процессора

TwinCAT real-time system конфигурируется через меню TwinCAT System Service.

Размерность шкалы времени



Можно ограничивать загрузку процессора задачей real-time. Это значит базовое время 1 мс, TwinCATу доступно максимум 800µs каждую миллисекунду.

Показывается максимальное и текущее значение «нестабильности» системы.

Работа Real time

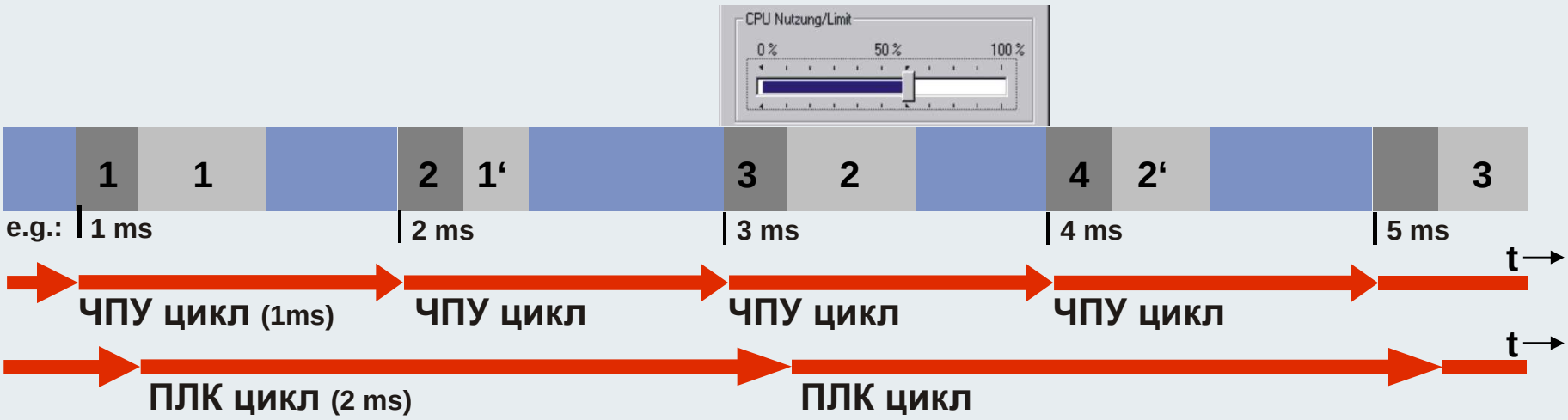
- Задачи ПЛК и ЧПУ выполняются детерминированно в соответствии с многозадачностью.

Работа в Real time ПЛК программы и ЧПУ задачи на ПК

ЧПУ

ПЛК программа

Win NT и HMI

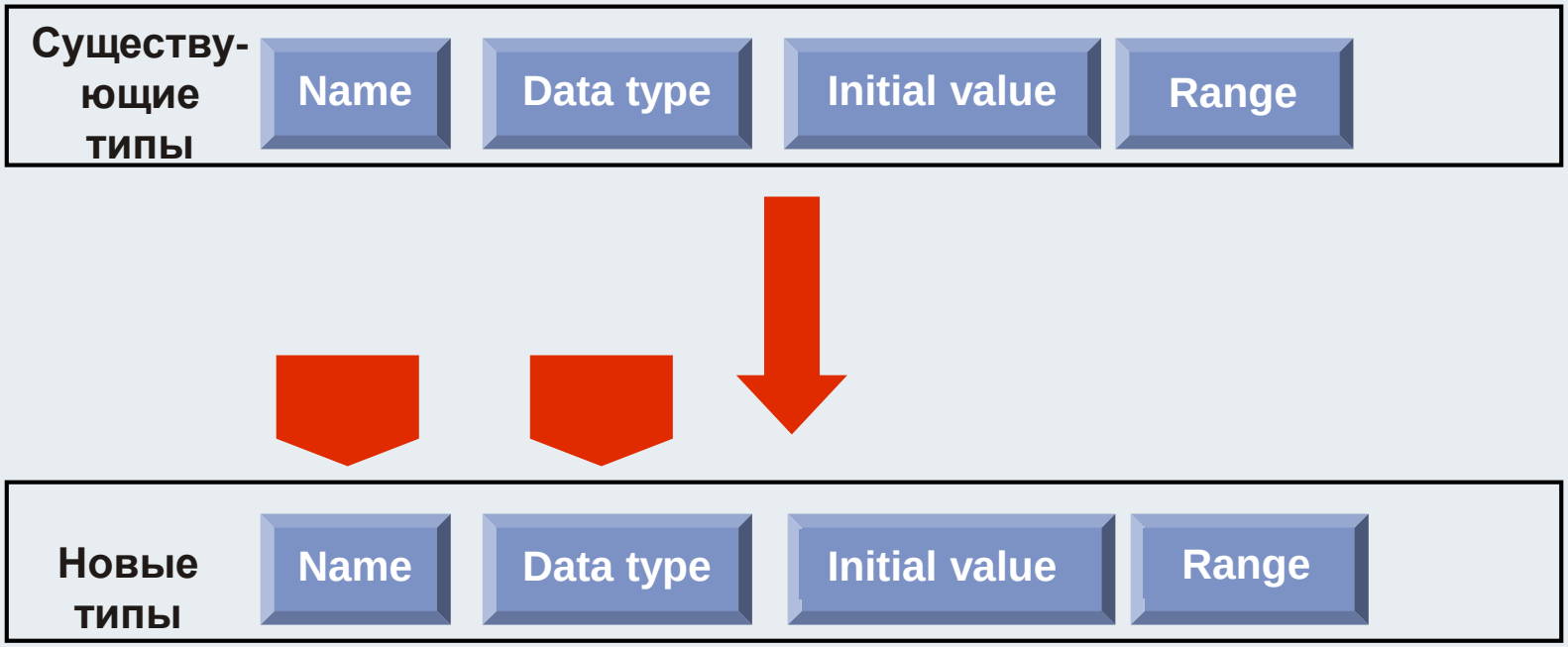


Произвольные типы данных

Пользователь может создавать собственные типы данных на основе базовых или уже созданных типов данных. Новые типы данных доступны всему проекту

.

Объявление начинается с TYPE и заканчивается END_TYPE.



Произвольные типы данных (псевдоним типов)

Собственные типы данных могут использоваться как альтернативные имена переменных, констант или ФБ.

Syntax:

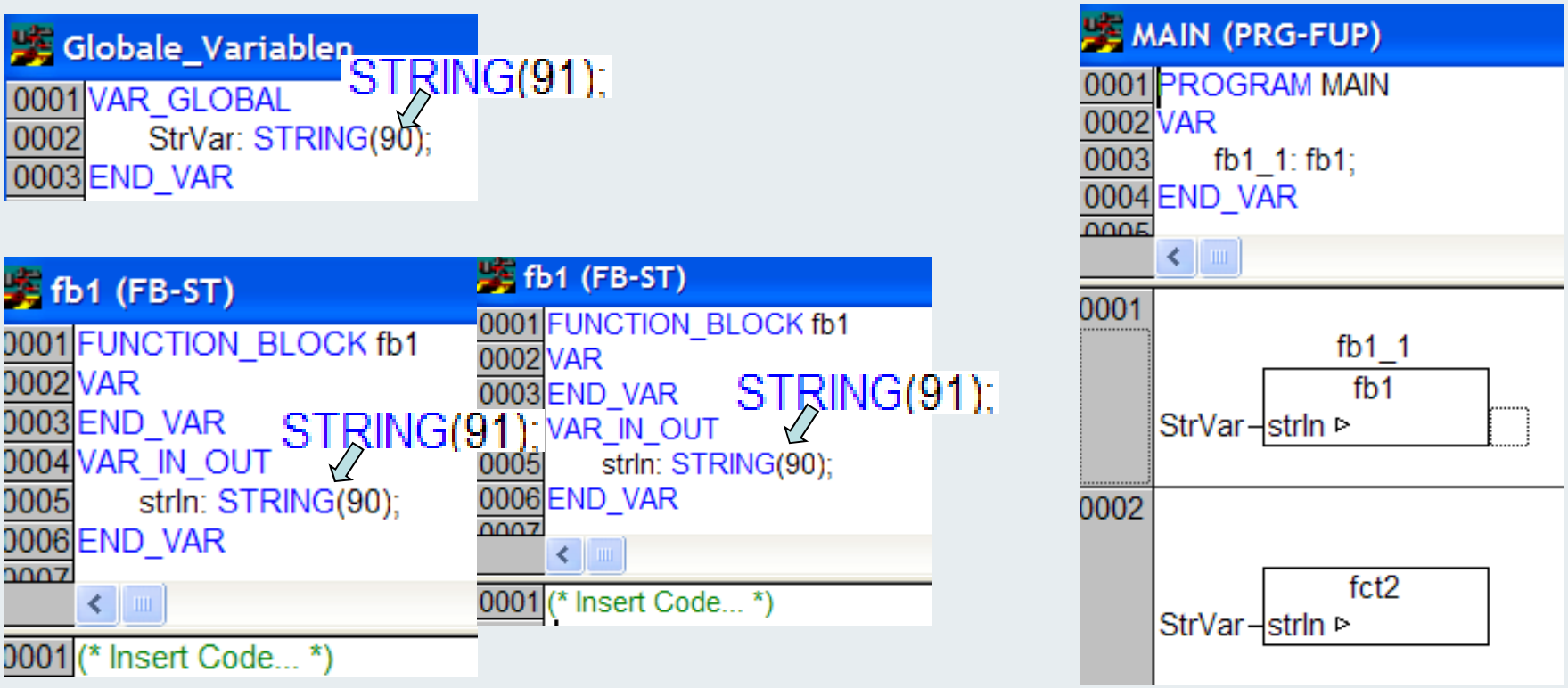
```
TYPE
 <Identifier>:<Assignment term>;
END_TYPE
```

Example: Ads_Net_ID

```
TYPE
    Net_ID:STRING(23);
END_TYPE
```

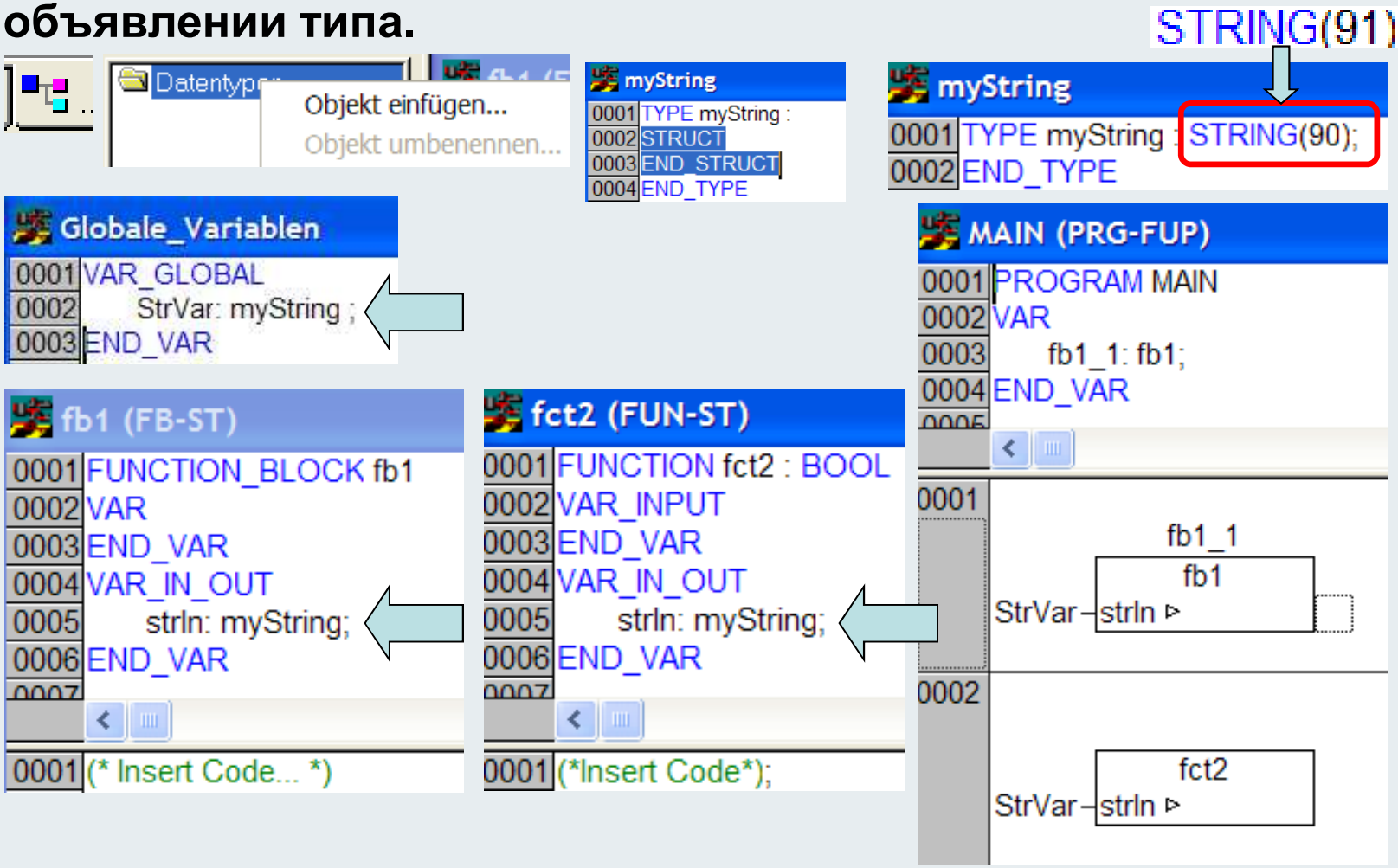
Произвольные типы данных (псевдоним типов)

ПРимер:Глобальная строковая переменная импользуется в нескольких блоках. После изменения глобальной переменной надо изменить объявление в блоках.



Произвольные типы данных (псевдоним типов)

Ести тип уже создан для строки, изменения требуются только в объявлении типа.



Перечисление

Перечисление - собственный тип объявления констант. Эти константы называются перечисляемыми значениями. Они доступны во всем проекте.

Объявление начинается с TYPE и заканчивается END_TYPE.

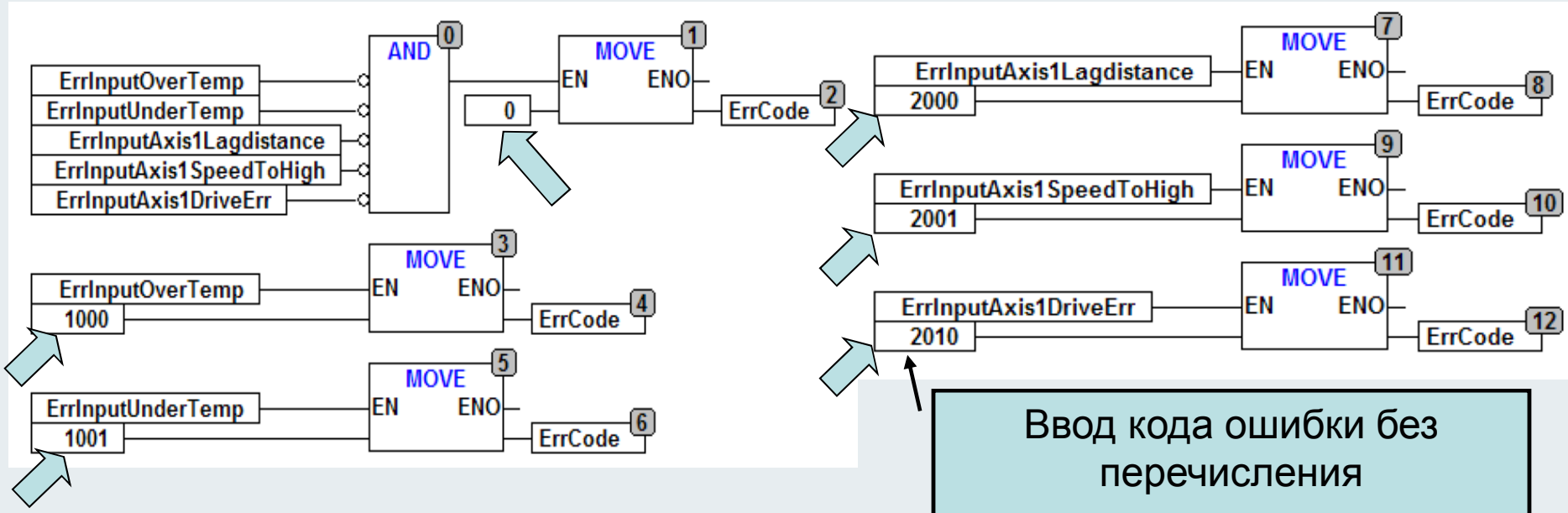
```
Syntax:
TYPE <Identifier>:(<Enum_0> ,<Enum_1>, ...,<Enum_n>);
END_TYPE

Example:
TYPE Week:(Mo, Di, Mi, Dn, Fr, Sa, So:=10);(*Mo = 0 Di = 1..
                                              .. Sa = 6 So = 10*)
END_TYPE

TYPE Direction:(Up, Dn);(*Up = 0 Dn = 1*)
END_TYPE
```

Одинаковые имена перечислений не допускаются

Перечисление

Пример: Назначаем переменную в зависимости от кода ошибки.

```
VAR
    ErrCode : UINT;(* 0: ERRCODE_NO_ERR *)
                   (* 1000: ERRCODE_OVERTEMP *)
                   (* 1001: ERRCODE_UNDERTEMP *)

                   (* 2000: ERRORCODE_AXIS1_LAGDISTANCE *)
                   (* 2001: ERRORCODE_AXIS1_SPEEDTOHIGH *)
                   (* 2010: ERRORCODE_AXIS1_DRIVEERR *)
                                    ErrInputOverTemp :BOOL;
                                    ErrInputUnderTemp :BOOL;
                                    ErrInputAxis1Lagdistance :BOOL;
                                    ErrInputAxis1SpeedToHigh :BOOL;
                                    ErrInputAxis1DriveErr :BOOL;
END_VAR
```



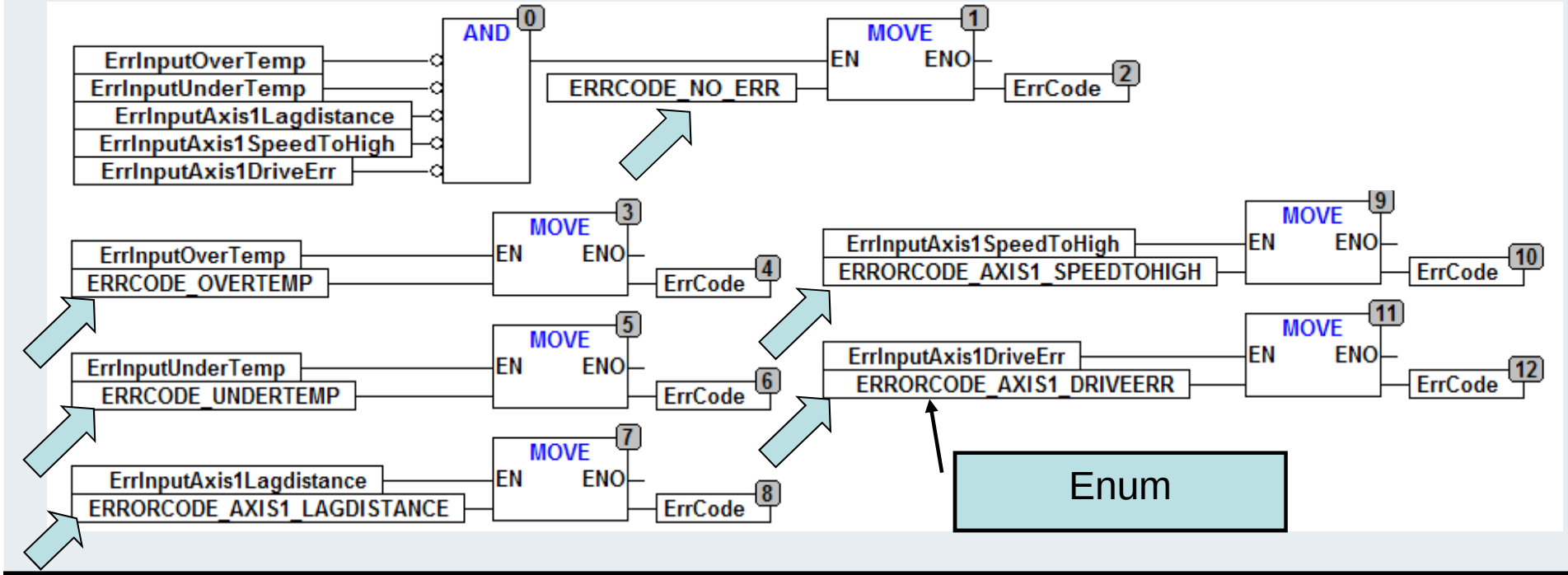
Ввод кода ошибки без перечисления

Перечисление Пример с ENUM

Datentypen
e_Errcode (ENUM)

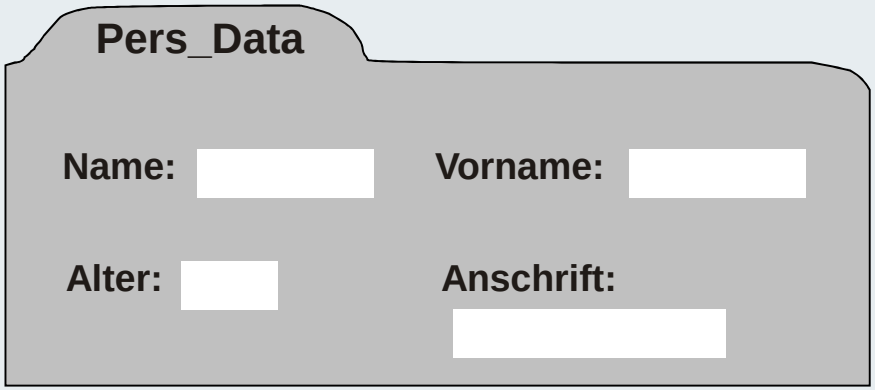
```
TYPE e_Errcode :
    (
    ERRCODE_NO_ERR,                     (* 0 *)
    ERRCODE_OVERTEMP:=1000,             (* 1000 *)
    ERRCODE_UNDERTEMP,                  (* 1001 *)
    ERRORCODE_AXIS1_LAGDISTANCE:=2000,  (* 2000 *)
    ERRORCODE_AXIS1_SPEEDTOHIGH,        (* 2001 *)
    ERRORCODE_AXIS1_DRIVEERR:=2010      (* 2010 *)
    );
END_TYPE
```

```
VAR
    ErrCode :  e_Errcode ;
    ErrInputOverTemp :BOOL;
    ErrInputUnderTemp :BOOL;
    ErrInputAxis1Lagdistance :BOOL;
    ErrInputAxis1SpeedToHigh :BOOL;
    ErrInputAxis1DriveErr :BOOL;
END_VAR
```



Объявление структуры

Форма



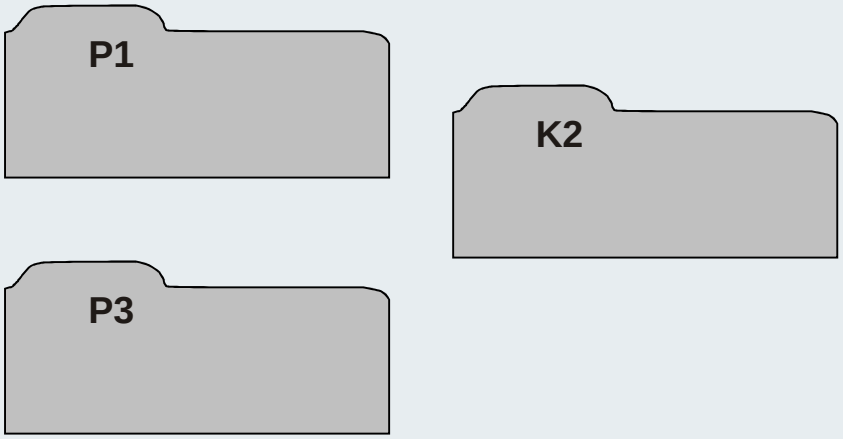

```
TYPE Pers_Data :
STRUCT
    Name: STRING(25);
    Vorname: STRING(25);
    Alter: USINT;
    Anschrift: STRING(55);
END_STRUCT
END_TYPE
```

Новый тип данных

Объявление полей

-
-
-

Использование структур



P1
Name:=‚Müller‘
First name:=‚Peter‘
Age:=32
Address:=‚Postweg 34‘

P3
Name:=‚Koschnik‘
First name:=‚Heinz‘
Age:=37
Address:=‚Domplatz 10‘

```
VAR
    P1, P3 : Pers_Data;
END_VAR
VAR_OUTPUT
    K2 : Pers_Data;
END_VAR
```

- ■
- ■

```
VAR_INPUT
    Employees: Pers_Data;
END_VAR
```

```
Name_Voll:=CONCAT(P3.First name,
                  P3.Name)
(*HeinzKoschnik*)
```

Массивы

Все элементы массива одного типа. Массивы могут быть собственного типа (структуры).

Возможны одно-, двух- и трехмерные массивы.

```
VAR
        Feld_1 :ARRAY[0..9] OF BYTE;          <-- одномерные
        Feld_2 :ARRAY[0..9, 0..1] OF UINT;    <-- двухмерные
        Feld_3 :ARRAY[0..9, 0..1,0..1] OF DINT;   <-- трехмерные
END_VAR
```

- Расположение массива по адресу
-
```
VAR
    Feld_1 AT%MB100:ARRAY[1..10] OF BYTE;
END_VAR
```

- Доступ к элементу массива

```
Feld_1[2] := 120;  (* прямой*)
Feld_2[i,j] := EXPT(i,j); (*по индексу*)
```

Одномерный массив, пример с инициализацией

Одномерный

Identifier Array Data type Initial value

```
DBZeiten :ARRAY [0..6] OF TIME:= T#1s, T#2s, T#1s, 4(T#0s);
```

Фактор

Значение

0	1	2	3	4	5	6
T#1s	T#2s	T#1s	T#0s	T#0s	T#0s	T#0s

Динамическое изменение размера массива невозможно

Access:

```
VAR
    WertAusArray : TIME;
END_VAR
WertAusArray := DBZeiten[1];
```

Двумерные массивы, пример инициализации

Identifier Field Date type Initial value

```
Supportpoint:ARRAY [0..1, 0..6] OF REAL:= 0, 1.7, 2, 4(3.33), 6, 6(1.2);
```

Фактор

Значение

	0	1	2	3	4	5	6
0	0	1.7	2	3.33	3.33	3.33	3.33
1	6	1.2	1.2	1.2	1.2	1.2	1.2

Access:

```
VAR
    WertAusArray : REAL;
END_VAR
```

```
WertAusArray := Supportpoints[1 ,0];
```

Инициализация массива с комментариями

Пример: Управление осью

```
Drivingjobs:ARRAY [0..3, 0..1] OF LREAL:=
                    (* Target position,      Velocity   *)
(*Job 0*)           20.0,                   30.0,
(*Job 1*)           33.75,                  30.0,
(*Job 2*)           45.0,                   30.0,
(*Job 3*)           70.75,                  30.0;
```

Трехмерные массивы, пример инициализации

Stuetzpunkte:ARRAY [0..1, 0..2, 0..3] OF UINT:=
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23;

[1,0,0] =12	[1,0,1] =13	[1,0,2] =14	[1,0,3] =15
[1,1,0] =16	[1,1,1] =17	[1,1,2] =18	[1,1,3] =19
[1,2,0] =20	[1,2,1] =21	[1,2,2] =22	[1,2,3] =23

```
access:
VAR
    WertAusArray : UINT;
END_VAR
WertAusArray :=
Stuetzpunkte[ 1,1,3 ];
```

[0,0,0] =0	[0,0,1] =1	[0,0,2] =2	[0,0,3] =3
[0,1,0] =4	[0,1,1] =5	[0,1,2] =6	[0,1,3] =7
[0,2,0] =8	[0,2,1] =9	[0,2,2] =10	[0,2,3] =11

ARRAY [**0..1**, 0..2, 0..3]

ARRAY [0..1, **0..2**, 0..3]

ARRAY [0..1, 0..2, **0..3**]

Выход за границы

Опасное состояние ПЛК программы возникает при обращении за пределы массива.

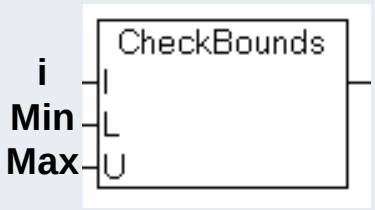
```
VAR
  Feld_1 :ARRAY[1..10] OF BYTE;
  Feld_2 :ARRAY[1..10, 2..5] OF UINT;
  Feld_3 :ARRAY[1..10] OF DINT;
END_VAR
```

i:= 9 Feld_1[i+2] := 120;	9
▪	
▪	
Feld_1[9];	0
Feld_2[1,2];	120

CheckBounds (FUN)

Мониторинг доступаможет выполняться самим ПЛК.

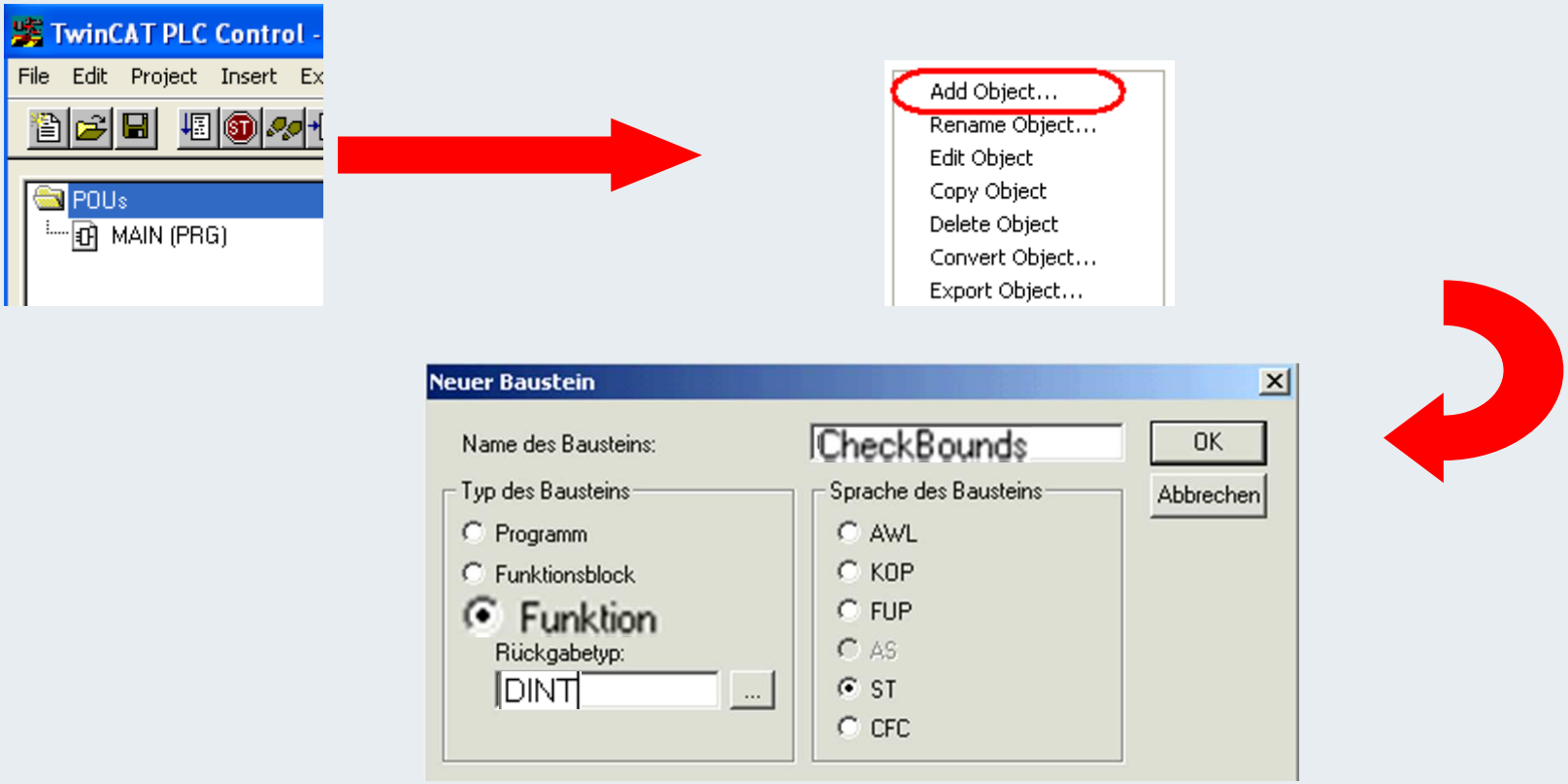
Данная функция распознает и предотвращает выход индекса за допустимые пределы.




```
FUNCTION CheckBounds :DINT
VAR_INPUT
        I,L,U : DINT;
END_VAR

IF I< L THEN
Error case      CheckBounds := L;
ELSIF I > U THEN
Error case      CheckBounds := U;
ELSE
„OK“ case       CheckBounds := I;
END_IF
```

Создание CheckBounds 1 (FUN)

CheckBounds можно скопировать “Copy Project“ из другого проекта в текущий (напр. из учебного). Или создать Checkbounds его заново.



Создание CheckBounds 2 (FUN)

Для успешной компиляции CheckBounds, запрещается изменять:

- **Имя и тип входов I,L и U**
- **Имя (CheckBounds) и возвращаемое значение (DINT)**

Изменения внутри функции свободные.

```
0001 FUNCTION CheckBounds : DINT
0002 (* check the array boundaries of all arrays in the project automatically *)
0003 VAR_INPUT
0004     I,L,U : DINT; (* dont change this interface ! *)
0005 END_VAR
```

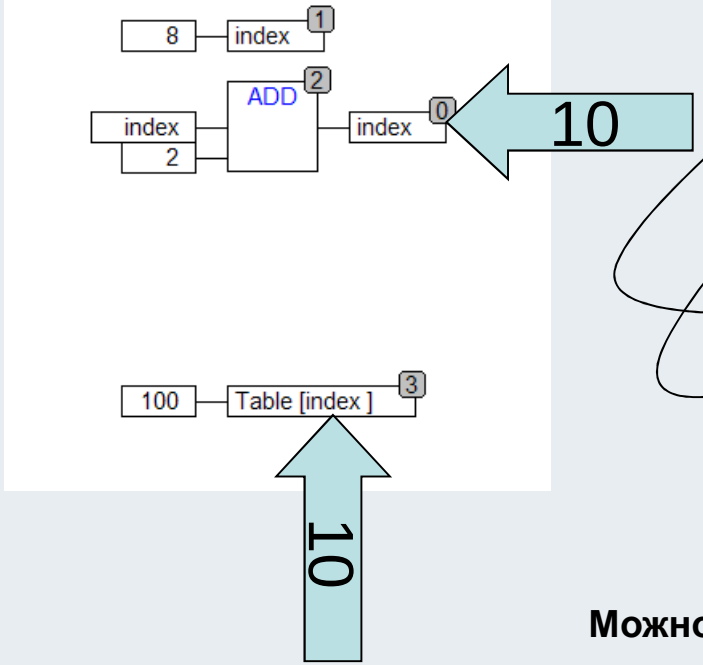
```
0001 (* you can add/modify the code (i.e. write to logfile, set flag *)
0002 IF I< L THEN
0003     CheckBounds := L;   (* returns lower bound L, if index I is lower than lower bound L *)
0004 ELSIF I > U THEN
0005     CheckBounds := U;   (* returns upper bound U, if index I is greater than upper bound U *)
0006 ELSE
0007     CheckBounds := I;   (* returns index I, if index I is in the bounds *)
0008 END_IF
```

CheckBounds (FUN) режим работы

Пример ошибки

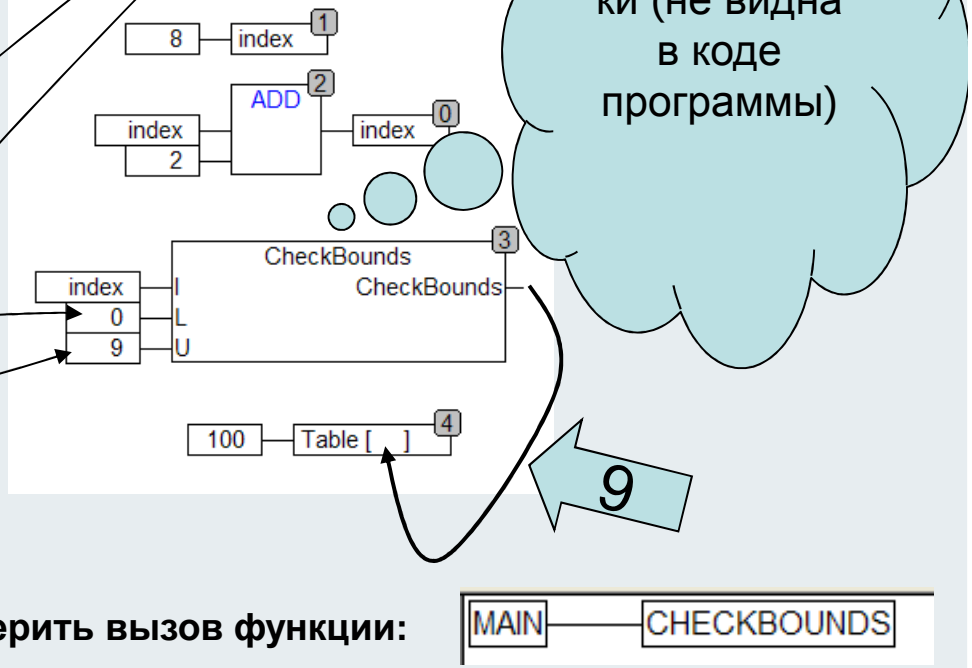
Исходный код

```
0001 PROGRAM MAIN
0002 VAR
0003     Table:ARRAY[0..9] OF INT;
0004     index :INT;
0005 END_VAR
0006
```



Checkbounds компилируется„автоматически“

```
0001 PROGRAM MAIN
0002 VAR
0003     Table:ARRAY[0..9] OF INT;
0004     index :INT;
0005 END_VAR
0006
```



Выполняется автоматическ ки (не видна в коде программы)

Можно проверить вызов функции:

Примечание: Дополнительные функции проверки

- Check division by 0
- CheckDivByte
- CheckDivWord
- CheckDivDWord
- CheckDivReal

- Check value ranges
- CheckRangeSigned
- CheckRangeUnsigned

(see appendix)

Комбинирование: структур и массивов

Массив может содержать структуры:

Structure:

```
TYPE DrillPos :
STRUCT
        XPos:           LREAL;
        FeedrateX:      LREAL;
        AccelerationX:  LREAL;
        DeccelerationX: LREAL;
        JerkX:          LREAL;
        YPos:           LREAL;
        FeedrateY:      LREAL;
        AcceleartionY:  LREAL;
        DeccelerationY: LREAL;
        JerkY:          LREAL;
        FeedDrill:      LREAL;
        Kuehlen:        BOOL;    (*Pump ?*)
END_STRUCT
END_TYPE
```

Объявление массива:

```
Positions :ARRAY[0..100] OF DrillPos;
```

Комбинирование: структур и массивов

Доступ к „Drillpos 55“:

```
MoveXAx (*FB Instanz*)
    (
    Execute:=           TRUE,
    Position:=          Positions[55].XPos ,
    Velocity:=          Positions[55].FeedrateX
    Acceleration:=              Positions[55].AccelerationX,
    Deceleration:=              Positions[55].DeccelerationX,
    Jerk:=                      Positions[55].JerkX,
    Direction:=         .........,
    Axis:=                      ..............,
     );
```

ST „Структурный текст“- операторы

Операция	Объявление	Приоритет
Выражение в ()	(выражение)	Больший приоритет
Вызов функции	Имя функции(лист параметров)	
Экспонента	EXPT	
Negate	-	
Bulid complements	NOT	
Multiply	*	Вычисление слева на право (10/2*5 = 25)
Divide	/	
Modulo	MOD	
Add	+	Одинаковый приоритет
Subtract	-	
Сравнение	<,>,<=,>=	
Равенство	=	
Неравенство	<,>	
BOOL AND	AND	
BOOL XOR	XOR	
BOOL OR	OR	Меньший приоритет

ST „Структурный текст“- инструкции

Instruction	Example
Assignment :=	Posvalue := 10;
Call of function block	Ton1(IN:=Start, PT:=T2s); Output:= Ton1.Q;
RETURN	RETURN;
IF	More details and examples on the following pages
CASE	
FOR	
WHILE	
REPEAT	
EXIT	
Empty instruction	;

Инструкция IF

Используется, если требуется разветвить программу в зависимости от условий.

IF не позволяет «прыгать» назад в цикле ПЛК.

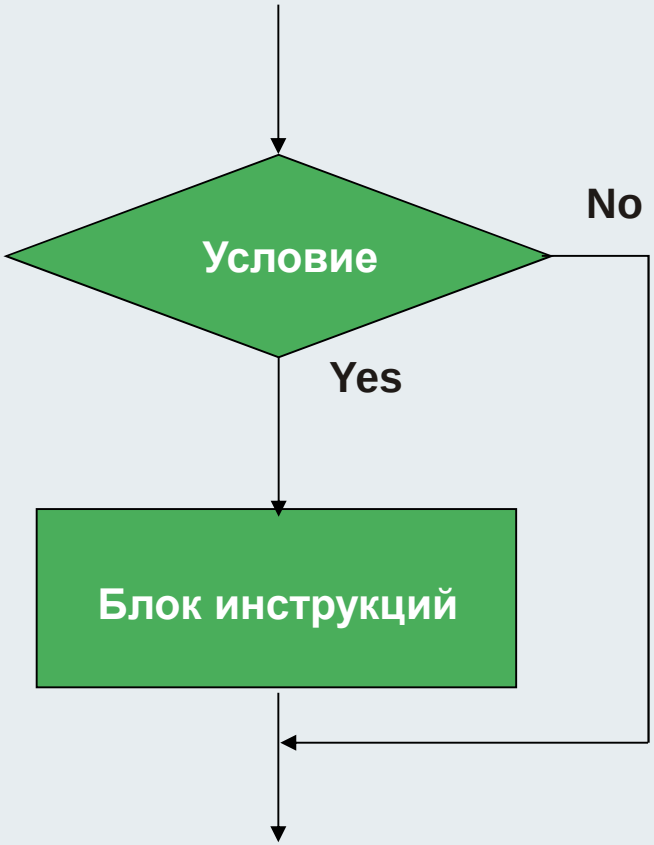
„GOTO“ так же не доступен

Ключевые слова:

```
IF          THEN
ELSIF
ELSE
END_IF
```

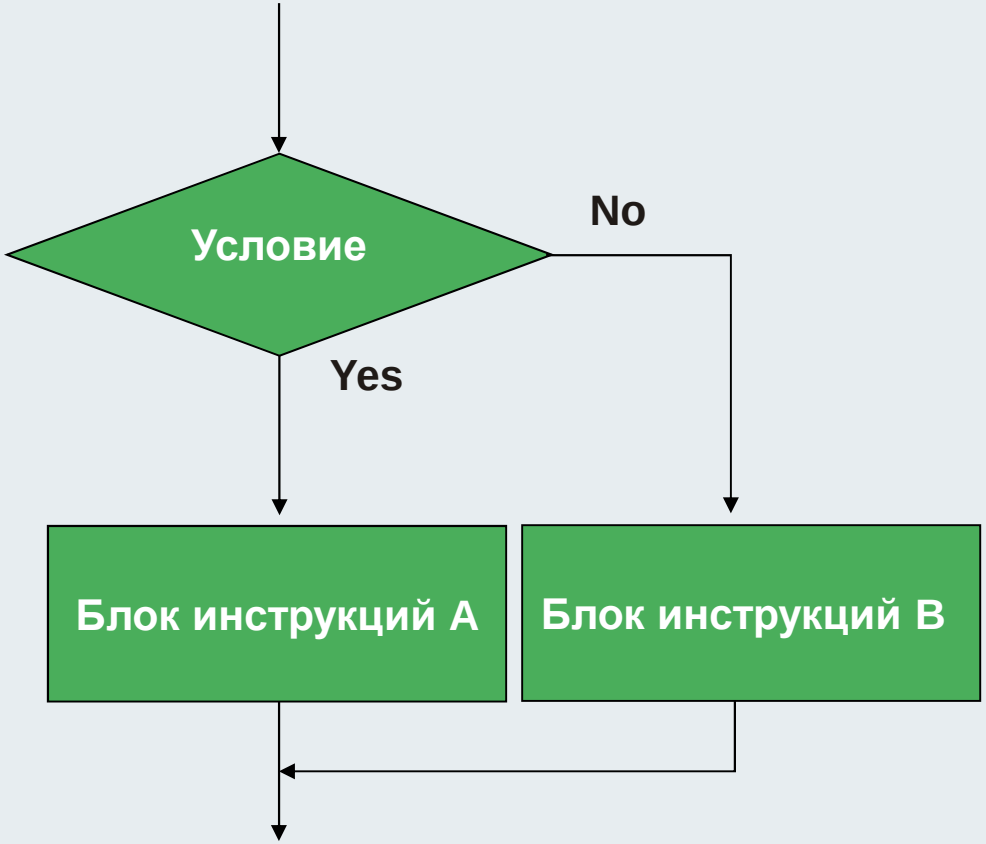
Инструкция IF

```
IF Условие THEN
    Блок инструкций;
END_IF
```



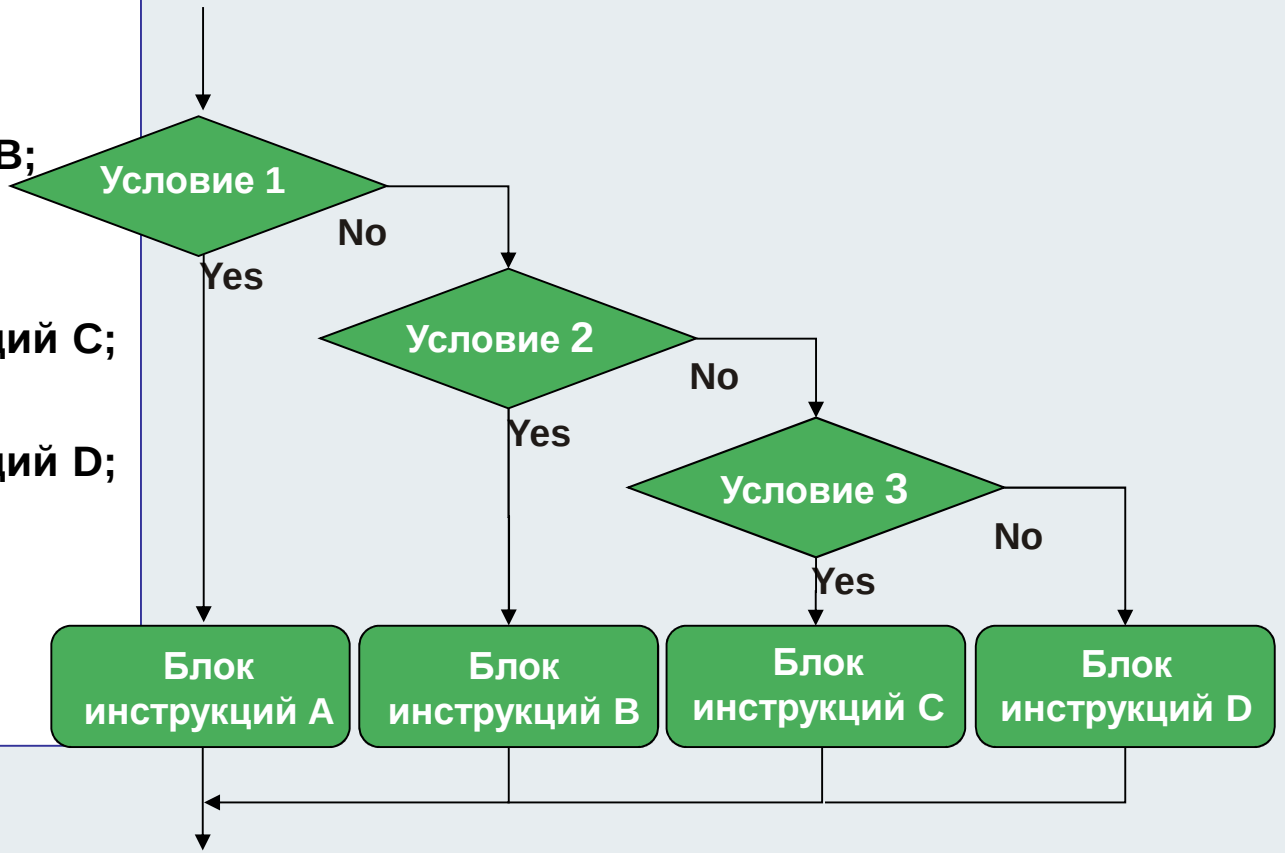
Инструкция IF

```
IF a>b THEN
    Блок инструкций A;
ELSE
    Блок инструкций B;
END_IF
```



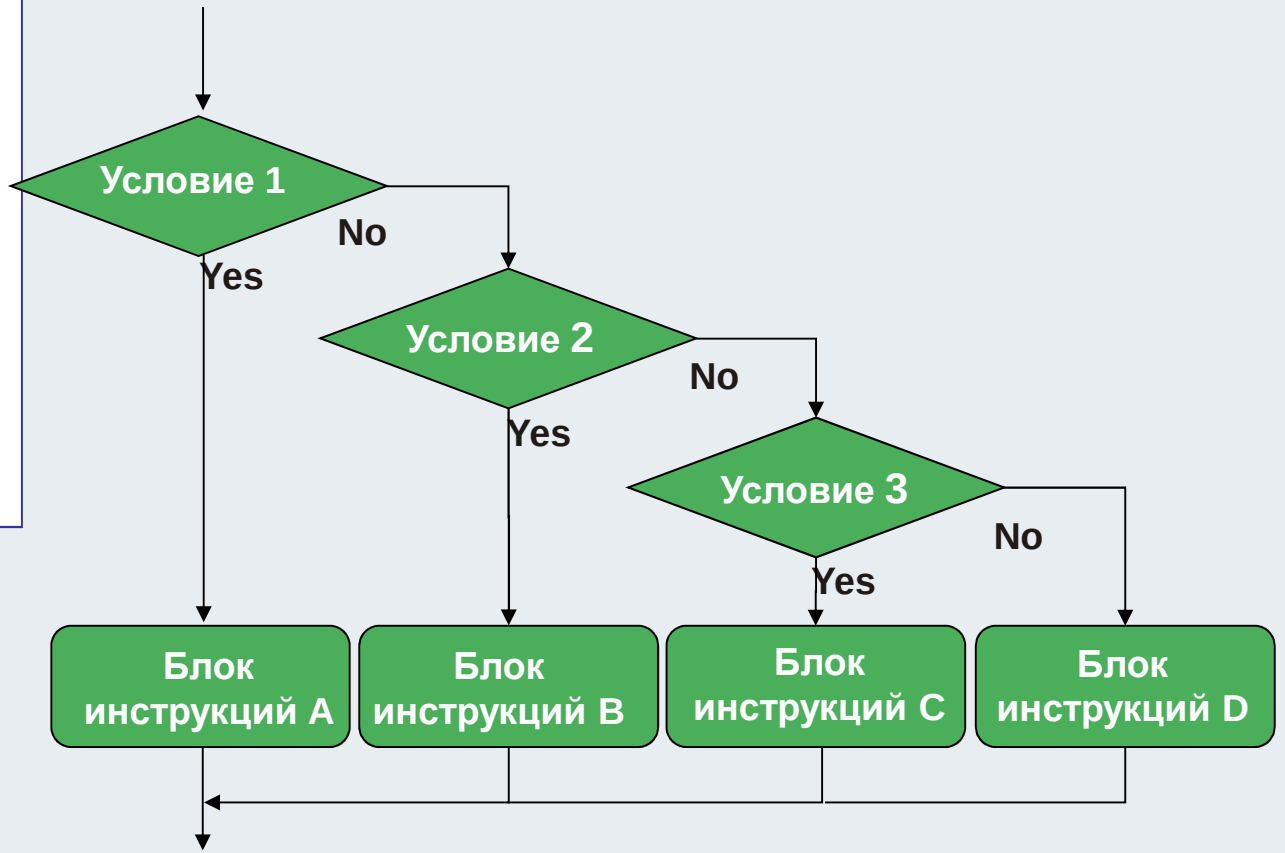
Инструкция IF

```
IF Условие 1 THEN
      Блок инструкций A;
ELSE
     IF Условие 2 THEN
          Блок инструкций B;
     ELSE
          IF Условие 3 THEN
               Блок инструкций C;
          ELSE
               Блок инструкций D;
          END_IF
     END_IF
END_IF
```



Инструкция IF

```
IF Условие 1 THEN
    Блок инструкций A;
ELSIF Условие 2 THEN
    Блок инструкций B;
ELSIF Условие 3 THEN
    Блок инструкций C;
ELSE
    Блок инструкций D;
END_IF
```



Инструкция IF

Каким может быть „Булевое выражение“ ?

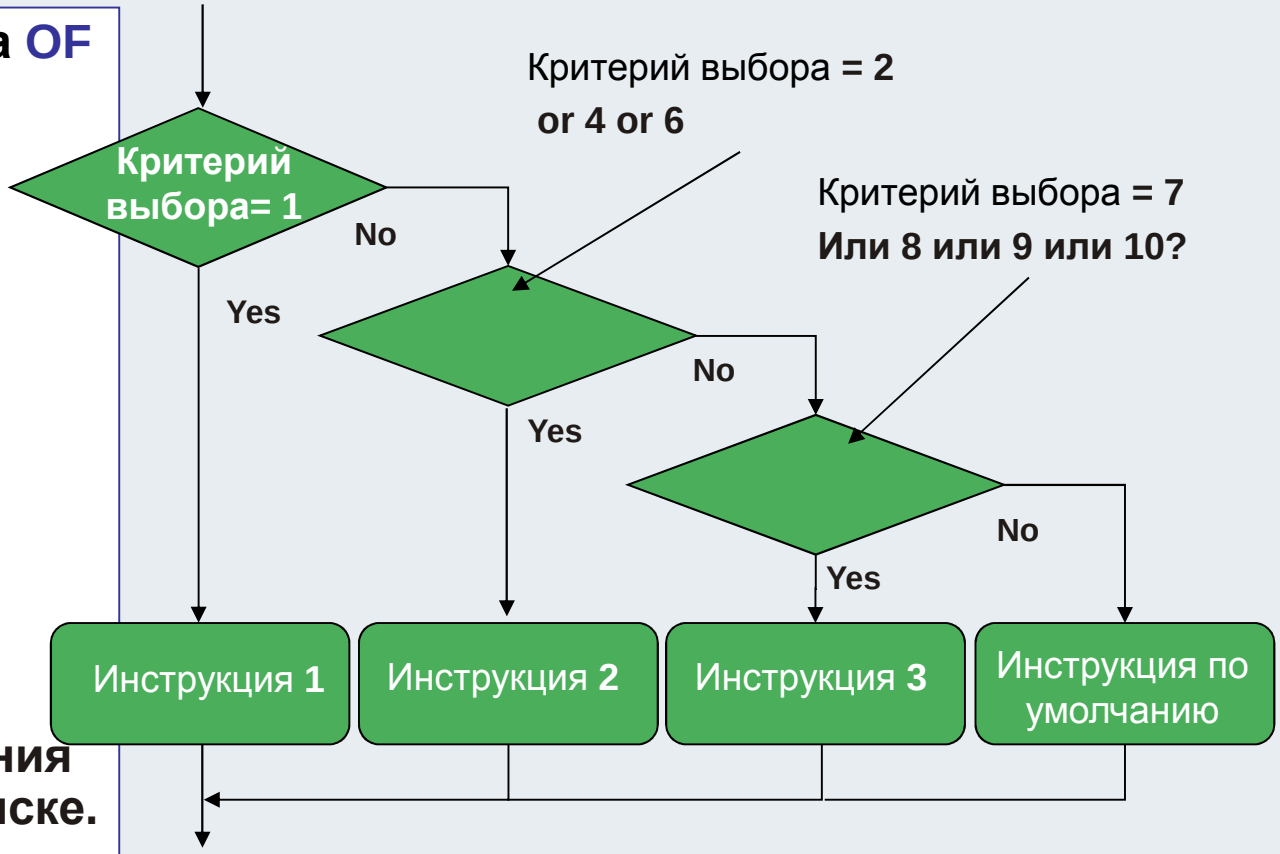
Условие:

- Булевая переменная → `IF bVar THEN`
  `.`
- Сравнение → `IF a>b THEN`
  `.`
- Вызов функции → `IF LEFT(STR:= strVar, SIZE:=7) = 'TwinCAT' THEN`
  `.`
- Значение выхода ФБ → `IF Ton1.Q THEN`
  `.`
- Нельзя вызывать ФБ! → ~~`IF Ton1(IN:=bVar, PT:=T#1s ) THEN`~~

Инструкция CASE

```
CASE Критерий выбора OF

1:          Инструкция 1
2, 4, 6:    Инструкция 2
7..10:      Инструкция 3
..
ELSE

    Инструкция по
    умолчанию

END_CASE;
```

Два одинаковых значения не может быть в списке.



Инструкция CASE : описание машины состояний

```
CASE State OF
   0:        Q0:=TRUE;
             IF Transition THEN state := 1; END_IF
   1:        Q1:=TRUE;
             IF Transition THEN state := 2; END_IF
   2:        Q2:=TRUE;
             IF Transition THEN state := 3; END_IF
   3:        Q3:=TRUE;
             IF Transition THEN state := 0; END_IF
END_CASE
```

Инструкция шага (Actions)

„Условие переключения“ (Transition)

Инструкция CASE „Целочисленный селектор“ с константами

```
CASE State OF
    0:      Инструкции ;(*State=0*)
            IF ……THEN
    1:      Инструкции ;(*State=1*)

    2:      Инструкции ;(*State=2*)

    3:      Инструкции ;(*State=3*)

END_CASE
```

Инструкции IF State = 0

Инструкции IF State = 1

Инструкции IF State = 2

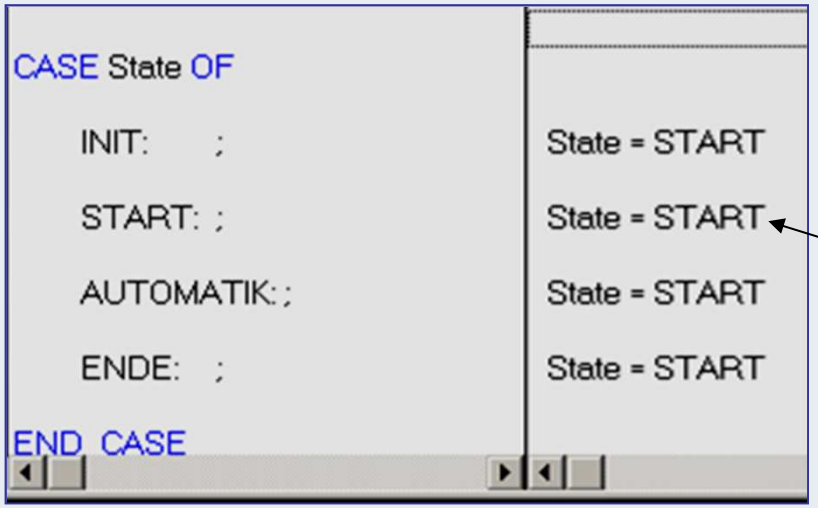
Инструкции IF State = 3

Инструкция CASE „Целочисленный селектор“ с перечисляемым типом

Enum-Typ:

```
TYPE Steps :
    (INIT:=0, START, AUTOMATIC, END);
END_TYPE
```

```
CASE State OF
   INIT:        Инструкции ;(*State=0*)
   START:       Инструкции ;(*State=1*)
   AUTOMATIC:   Инструкции ;(*State=2*)
   END:         Инструкции ;(*State=3*)
END_CASE
```



Если целочисленная переменная state объявлена перечисляемым типом, в online режиме она видна в виде текста.

```
VAR
    State:Steps;
    (* State:INT also possible*)
END_VAR
```

Инструкция CASE : описание машины состояний

```
TYPE Steps :
    ( INIT:=0, START, AUTOMATIC, END);
END_TYPE
```

```
CASE State OF
   INIT:        Q0:=TRUE;
                IF Transition THEN state := START; END_IF
   START:       Q1:=TRUE;
                IF Transition THEN state := AUTOMATIC; END_IF
   AUTOMATIC:   Q2:=TRUE;
                IF Transition THEN state := END; END_IF
   END:         Q3:=TRUE;
                IF Transition THEN state := INIT; END_IF
END_CASE
```

Инструкции для след. шага (actions)

„Условие переключения“ (Transition)

Step

Инструкция CASE „Целочисленный селектор“ с константами

```
VAR CONSTANT
    Step1 : INT:=   0;
    Step2 : INT:=   1;
    Step3 : INT:=   2;
    Step4 : INT:=   3;
END_VAR

VAR
    State:INT;
END_VAR
```

```
CASE State OF
   Step1:          Инструкции;(*State=0*)
   Step2:          Инструкции;(*State=1*)
   Step3..Step4:   Инструкции;(*State=2 oder 3*)
END_CASE
```

Повторяемые инструкции

Обработка данных процесса часто требует многократного выполнения одинакового программного кода. Количество повторений определяется во время выполнения run time.

Недостатки циклов:

В случае программной ошибки может потребоваться большое количество повторов.

В случае невыполнения циклической операции до временного ограничения времени real-time, задачи в высшим приоритетом выполняться во время. Задачи с низким приоритетом успеют выполниться.

Циклы (Обзор)

Любой цикл можно завершить инструкцией EXIT, не смотря на условия выхода из цикла.

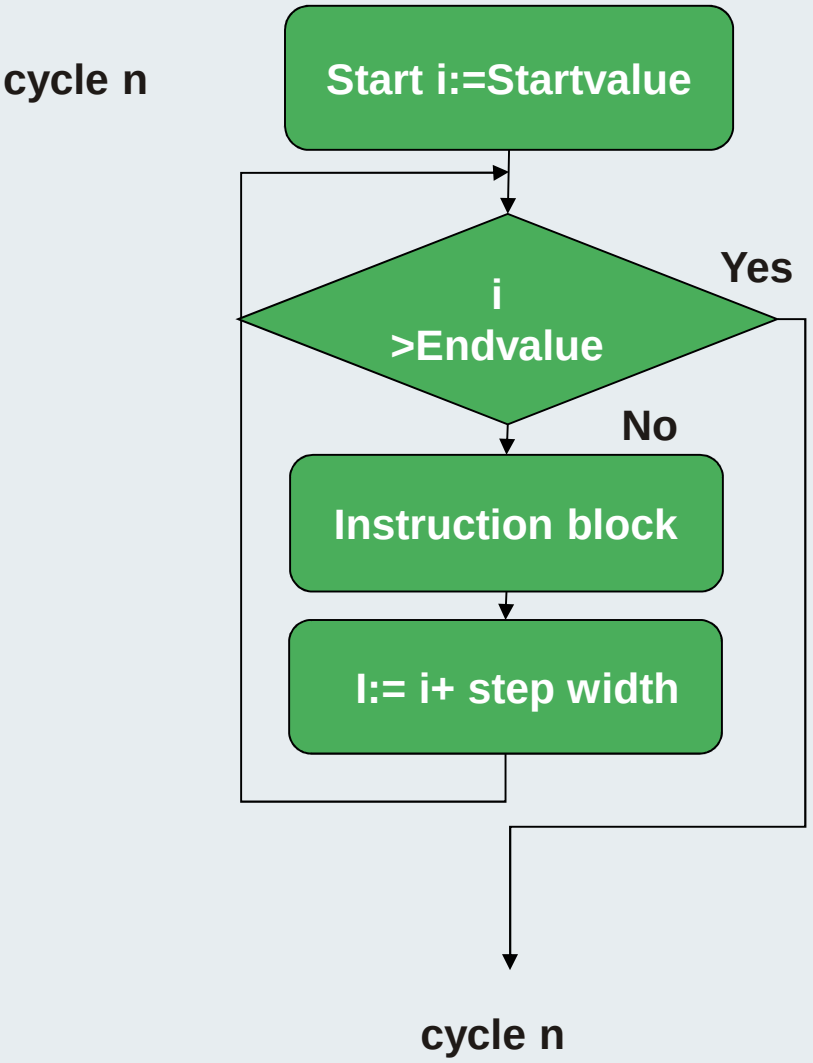
	Выражение	Режим работы	Фиксированное кол-во циклов
FOR	SINT/INT/DINT	Пред условие	Да
WHILE	BOOL	Пред условие	Нет
REPEAT	BOOL	Пост условие	Нет

Цикл FOR

В начале цикла переменная i определяется со стартовым значением (см. пример). Переменная увеличивается или уменьшается кажыдй цикл в зависимости от величины шага (значение после ключевого слова BY)

При достижении i конечного значения (после TO), цикл более не выполняется.

```
FOR  i:=1 TO 12 BY 2 DO
    Field[i]:=i*2;(* Инструкции *)
END_FOR
```



Цикл WHILE

Блок инструкций цикла WHILE выполняется до тех пор пока булевое выражение равно TRUE.
Условие выхода содержит переменную, которая должна измениться в блоке инструкций.
Если булевое выражение FALSE, блок инструкций цикла WHILE не выполняется.

```
i:=0;
WHILE  i<100 DO
   Field[i]:=i*2;(*Instruction*)
   i:=i+1;
END_WHILE
```

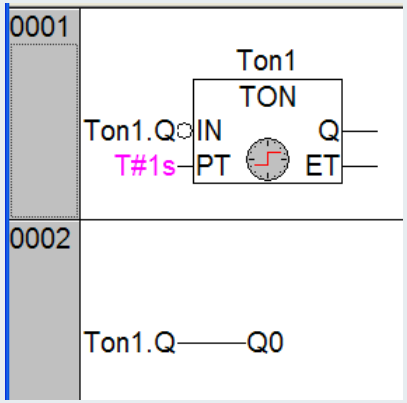
Цикл REPEAT

Блок инструкций цикла REPEAT выполняется так долго (UNTIL) пока булевое выражение не перестанет соблюдаться. Блок инструкций выполняется хотя бы один раз.

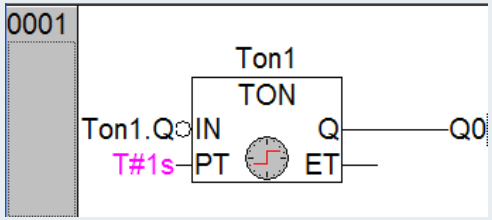

```
i:=0;

REPEAT
   Field[i]:=i*2;(*Instruction*)
   i:=i+1;
UNTIL i>100

END_REPEAT
```

Вызов ФБ в ST

```
VAR
  TON1:TON;
END_VAR
```

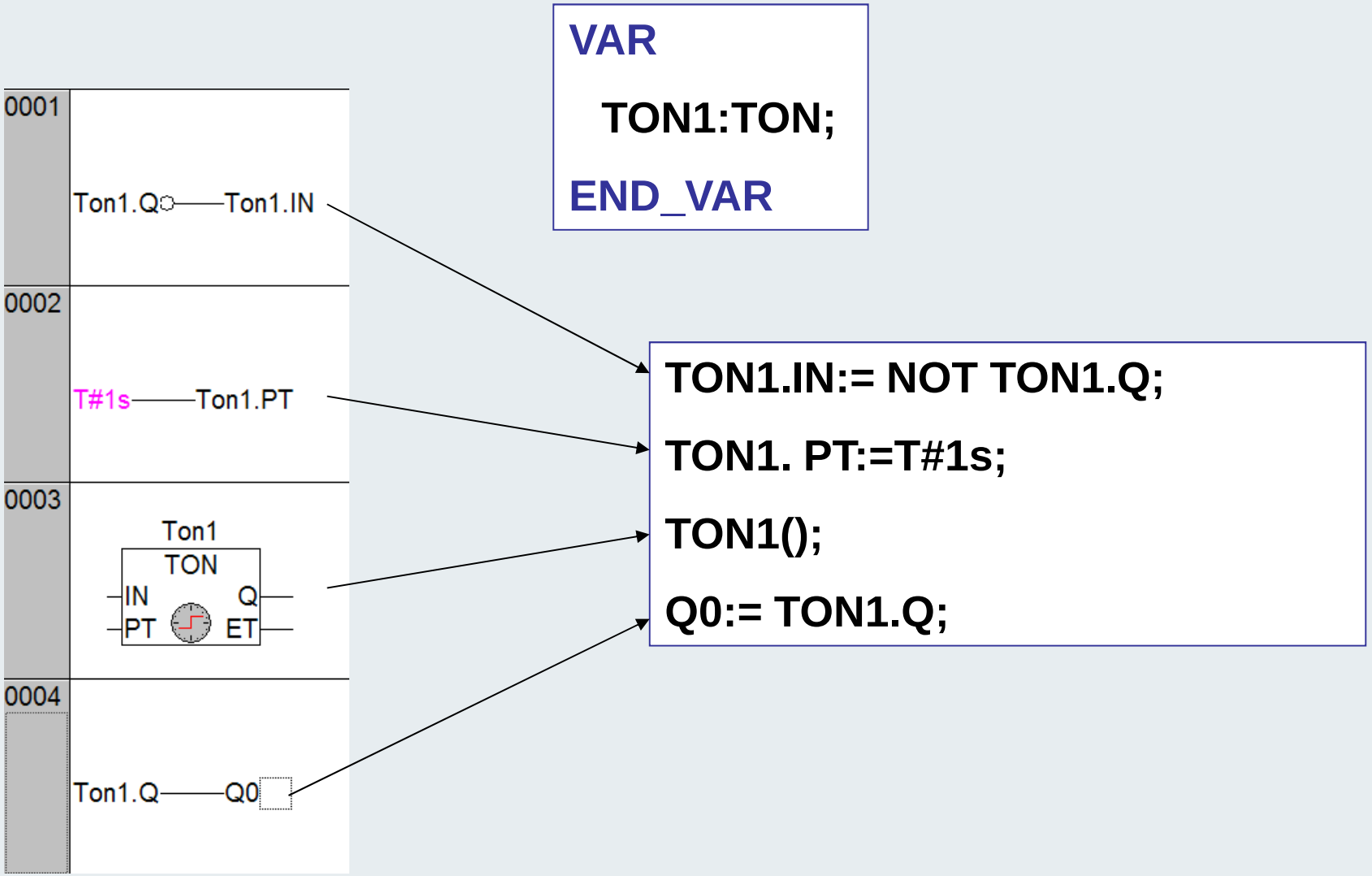



```
TON1 (IN:= NOT TON1.Q , PT:=T#1s );
Q0:= TON1.Q;
```



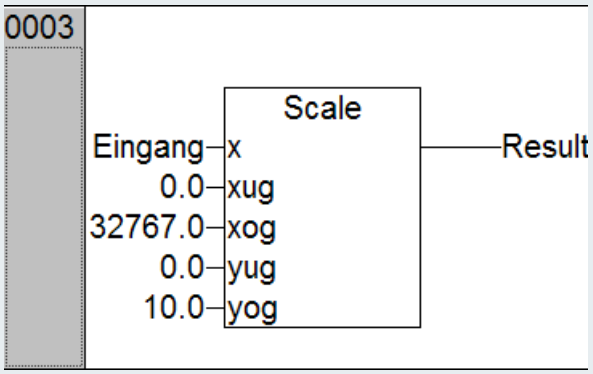

```
TON1(IN:= NOT TON1.Q, PT:=T#1s , Q=>Q0 );
```

Вызов ФБ в ST (альтернатива)

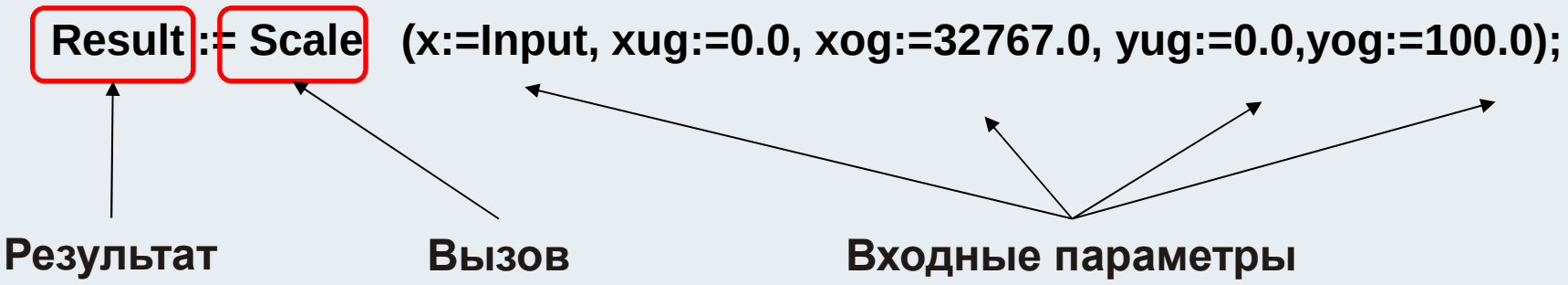


Вызов функции в ST

```
Result:=Scale    (x:=Input, xug:=0.0, xog:=32767.0, yug:=0.0,yog:=100.0);
(* equal:*)
Result:=Scale  (Input, 0.0, 32767.0, 0.0, 100.0);
(* equal:*)
Result:=Scale  (
          x:=      Input,
          xug:=   0.0,
          xog:=   32767.0,
          yug:=   0.0,
          yog:=   100.0
          );
```

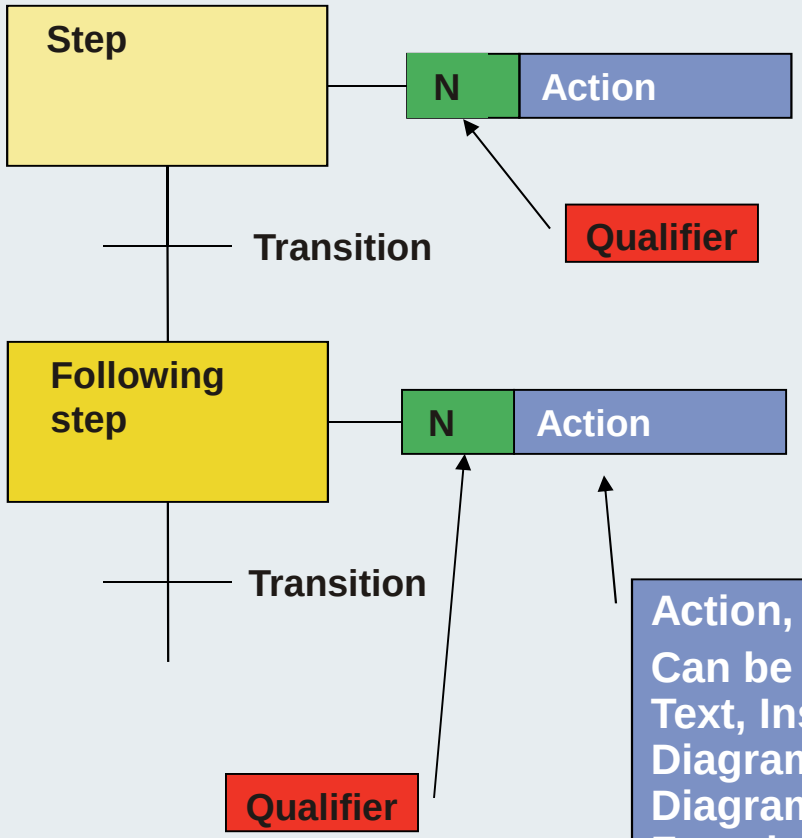


Вызов функции в ST




```
(* equal:*)
Result:=Scale    (
                 x:=      Input,
                 xug:=    0.0,
                 xog:=    32767.0,
                 yug:=    0.0,
                 yog:=    100.0
                 );
```

SFC Sequential Function Chart

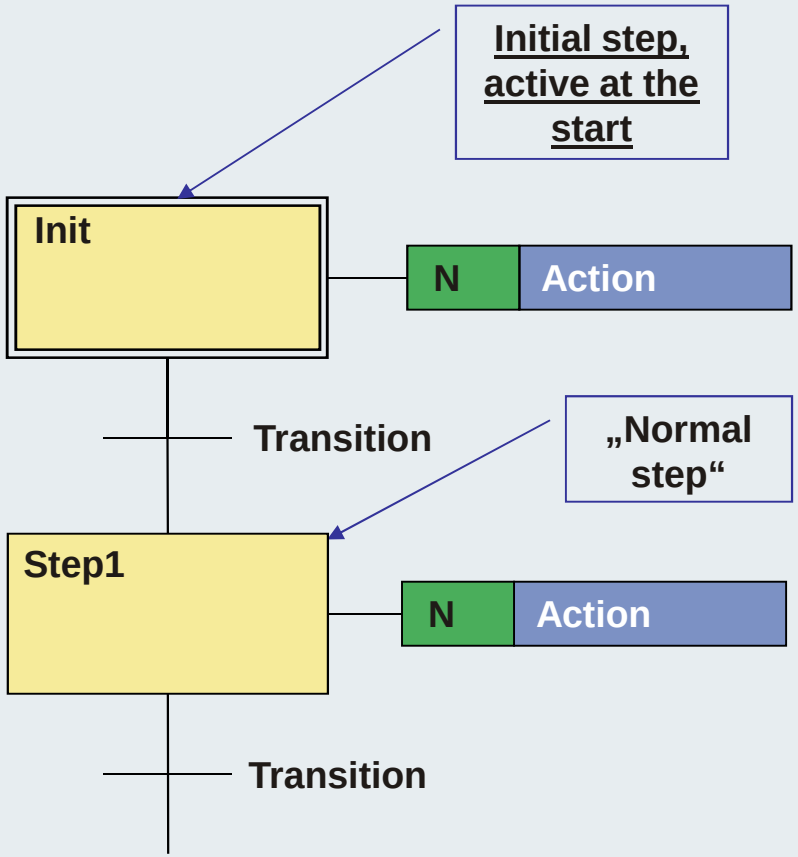


- Only one step is active at a time
- The condition to change from one step to another is the transition.
- In the action must be programmed what should be executed during the active step.

Action,

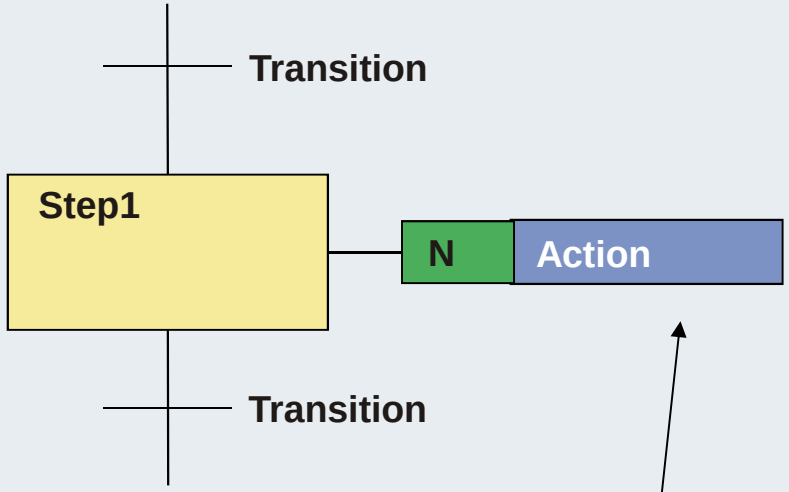
Can be written in Structured Text, Instruction list, Ladder Diagram, Function Block Diagram and in Sequential Function Chart .

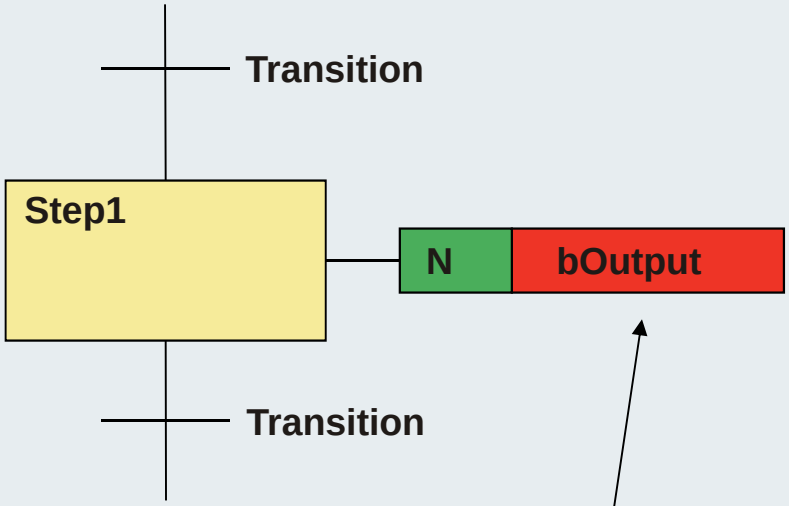
Steps



- The activity of a step can be requested with Stepname.X.
- The duration of the activity of a step can be requested with Stepname.T
- Both are components of a structure, which are created automatically from PLC Control. At the programming only the stepname has to be defined
- Stepname.X and Stepname.T are local variable and can only be read.

Actions



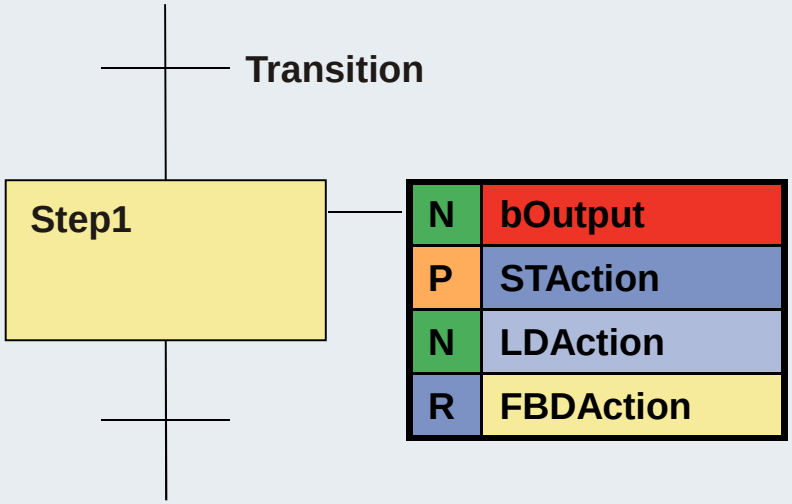


Action, can be programmed in
-> Structured Text,
-> Instruction List,
-> Ladder Diagram,
-> Functionblock Diagram,CFC/FBD
-> Sequential Function Chart, SFC

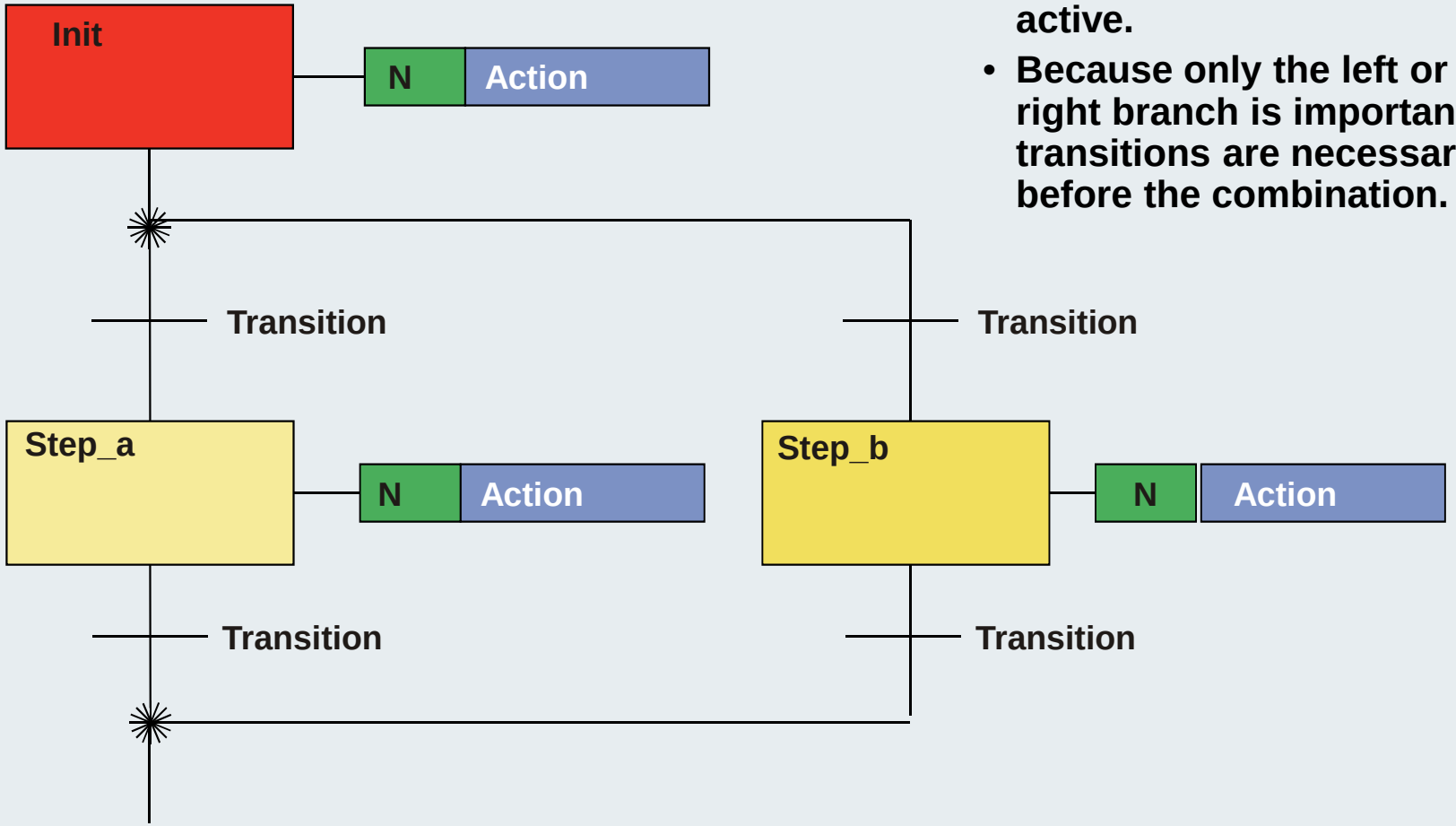
Action can be a variable of type BOOL.

The variable is TRUE by activating the the step and FALSE by leaving the step.

Actions, several allowed per step



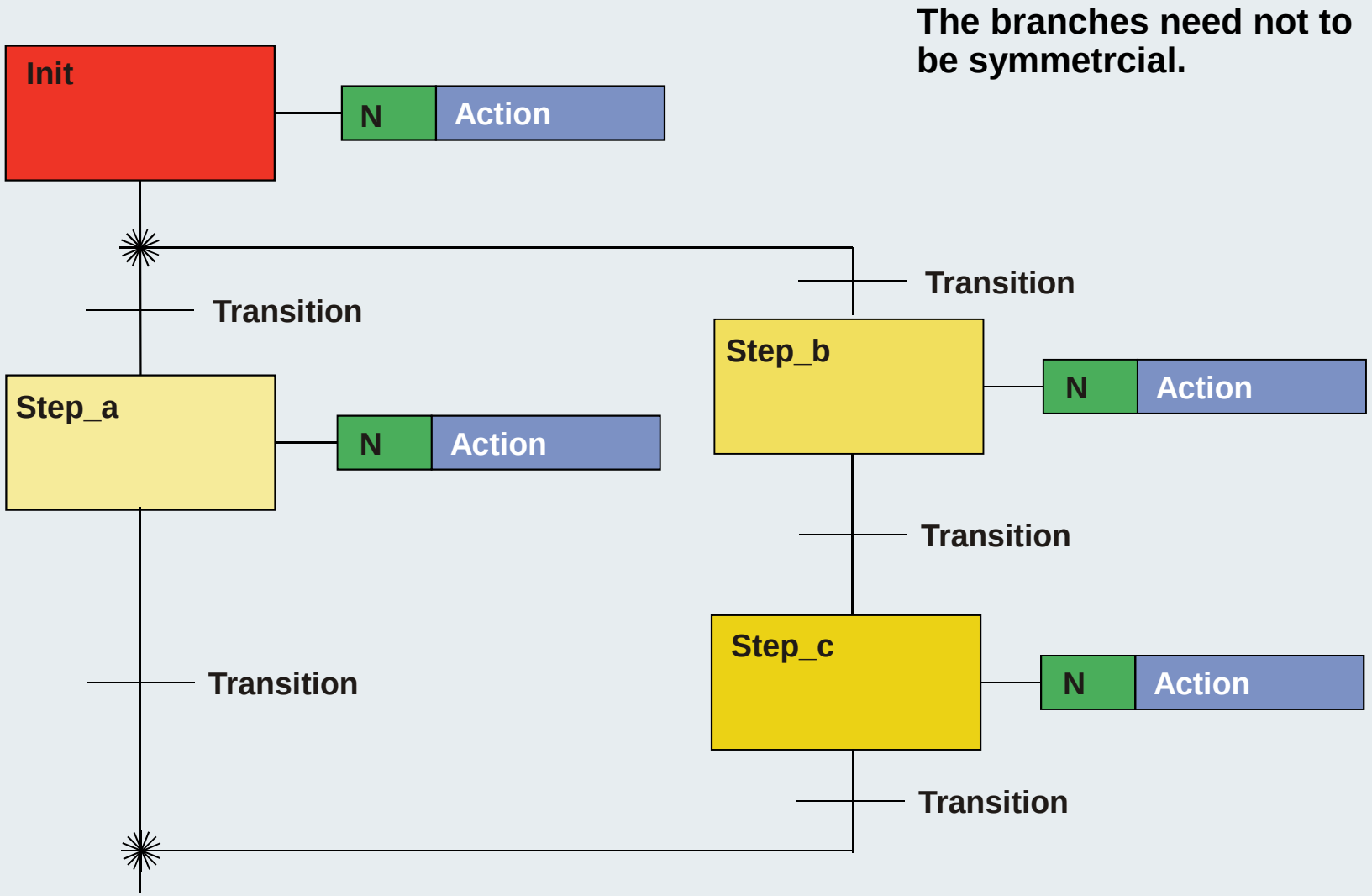
Steps /alternative branches



Only one branch can become active.

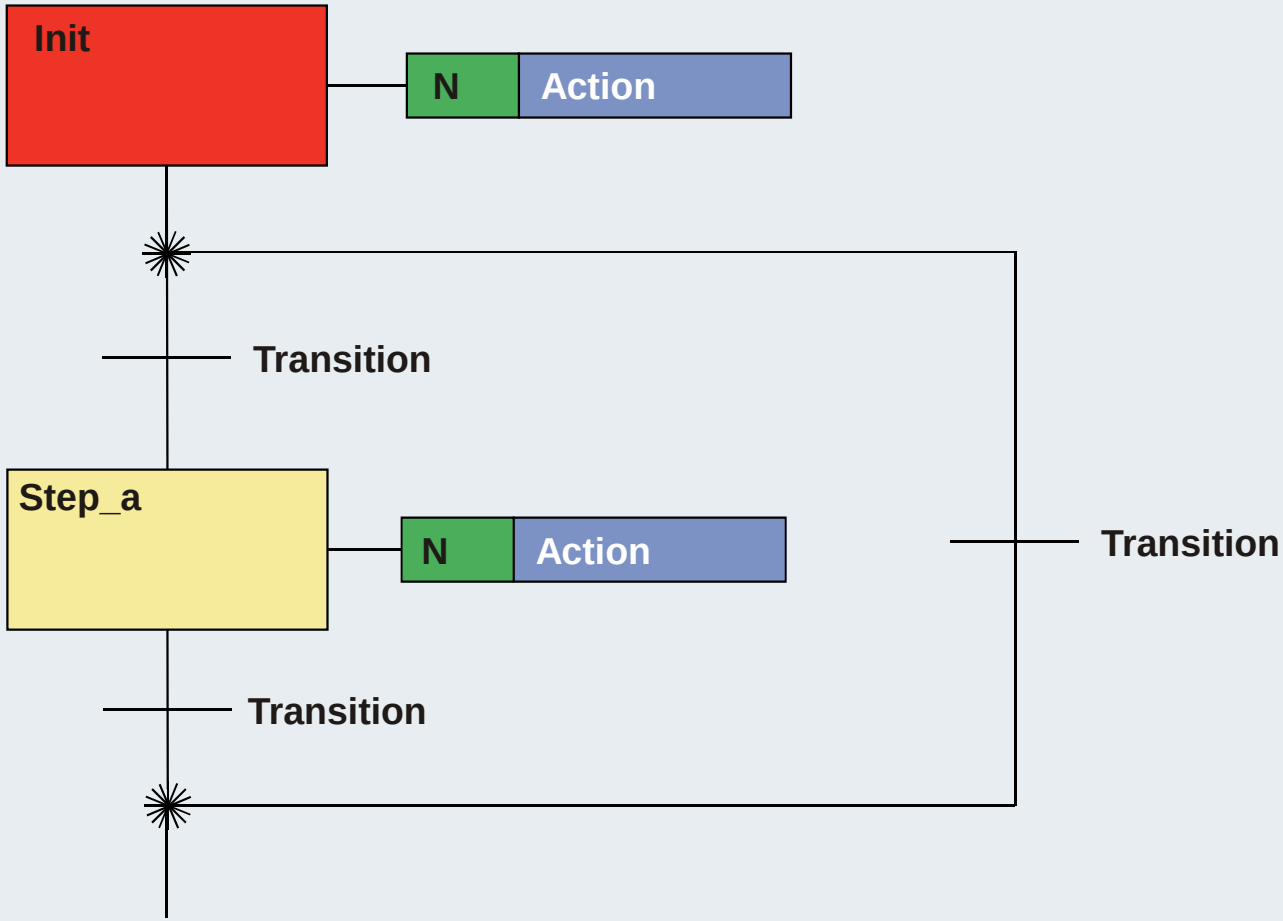
- Because only the left or the right branch is important, two transitions are necessary before the combination.

Steps /alternative branches

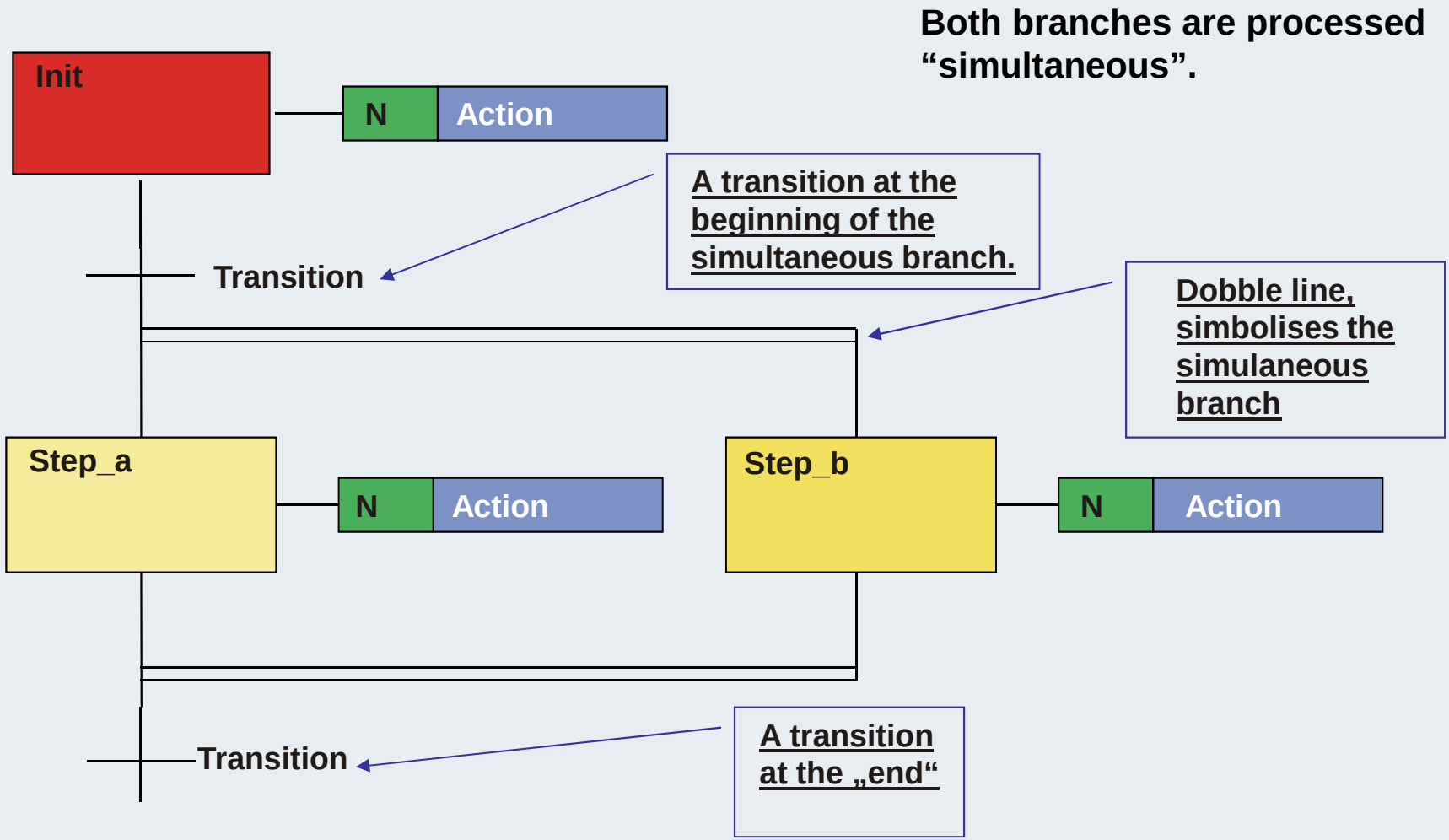


Steps /alternative branches

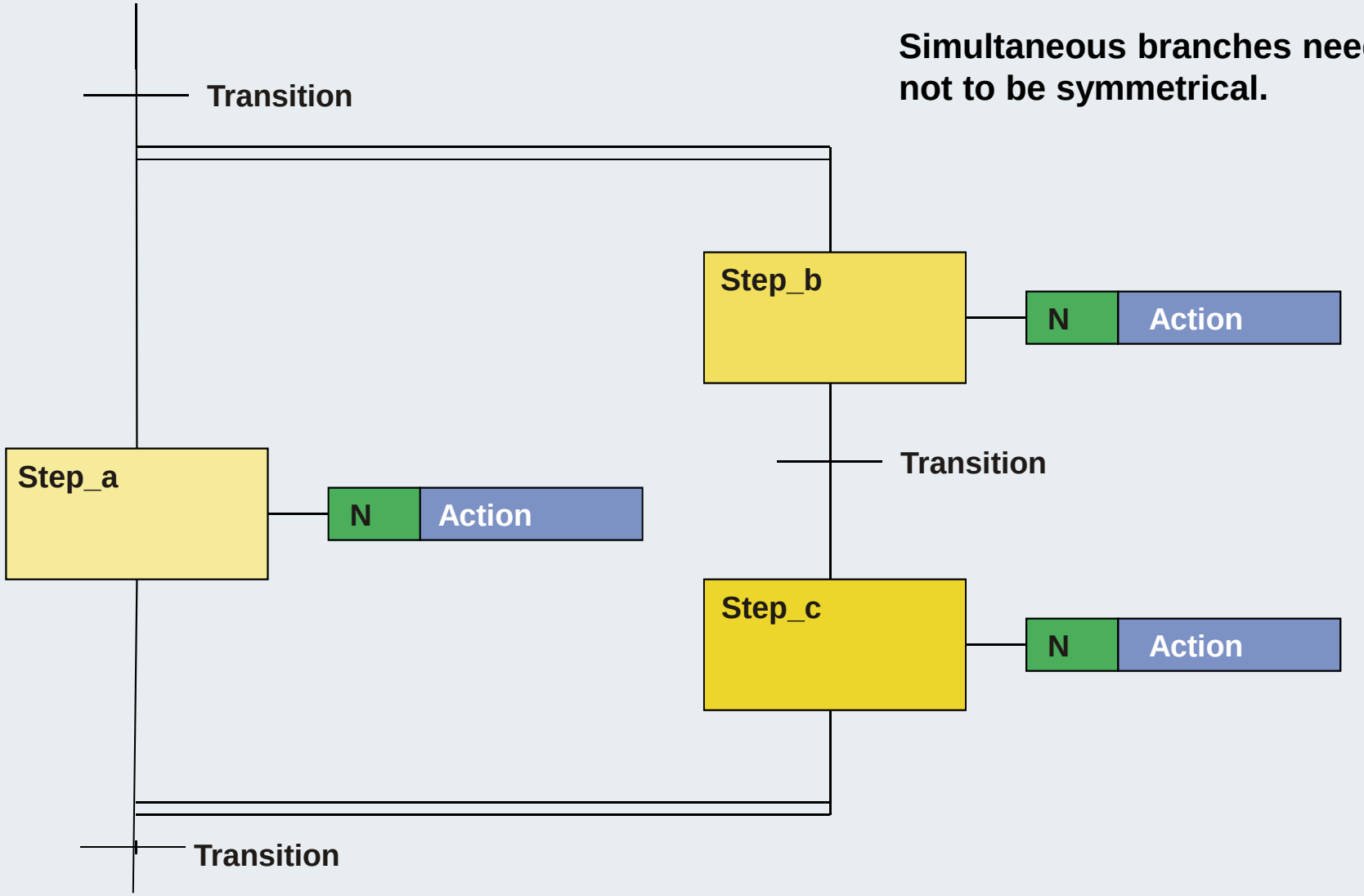
Branches can be skipped.



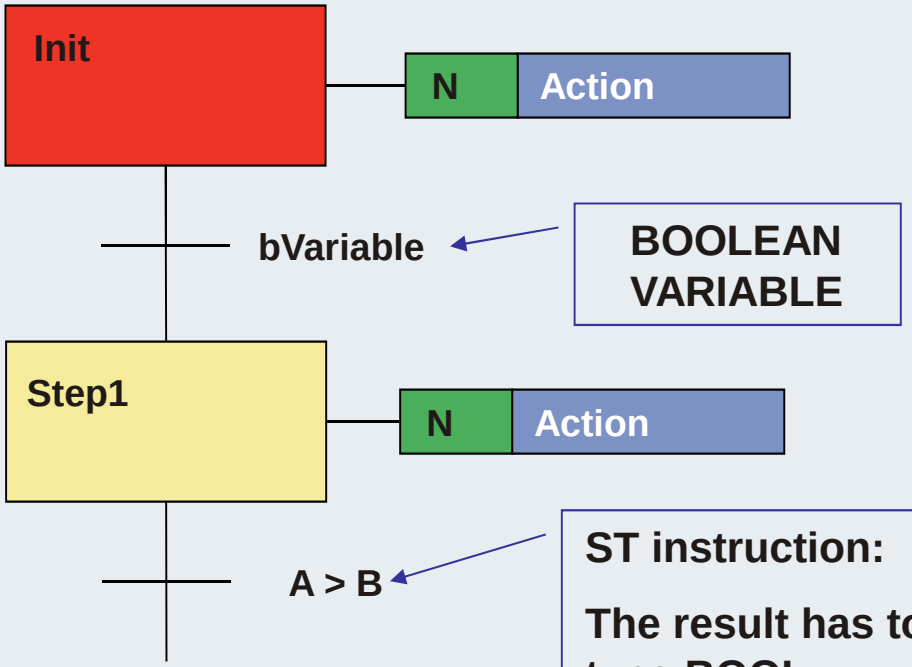
Steps /simultaneous branches



Steps /simultaneous branches



Transitions



A transition has to be of type „BOOL“. Possibilities:

- boolean variable
- ST instruction
- „programmed“ transition

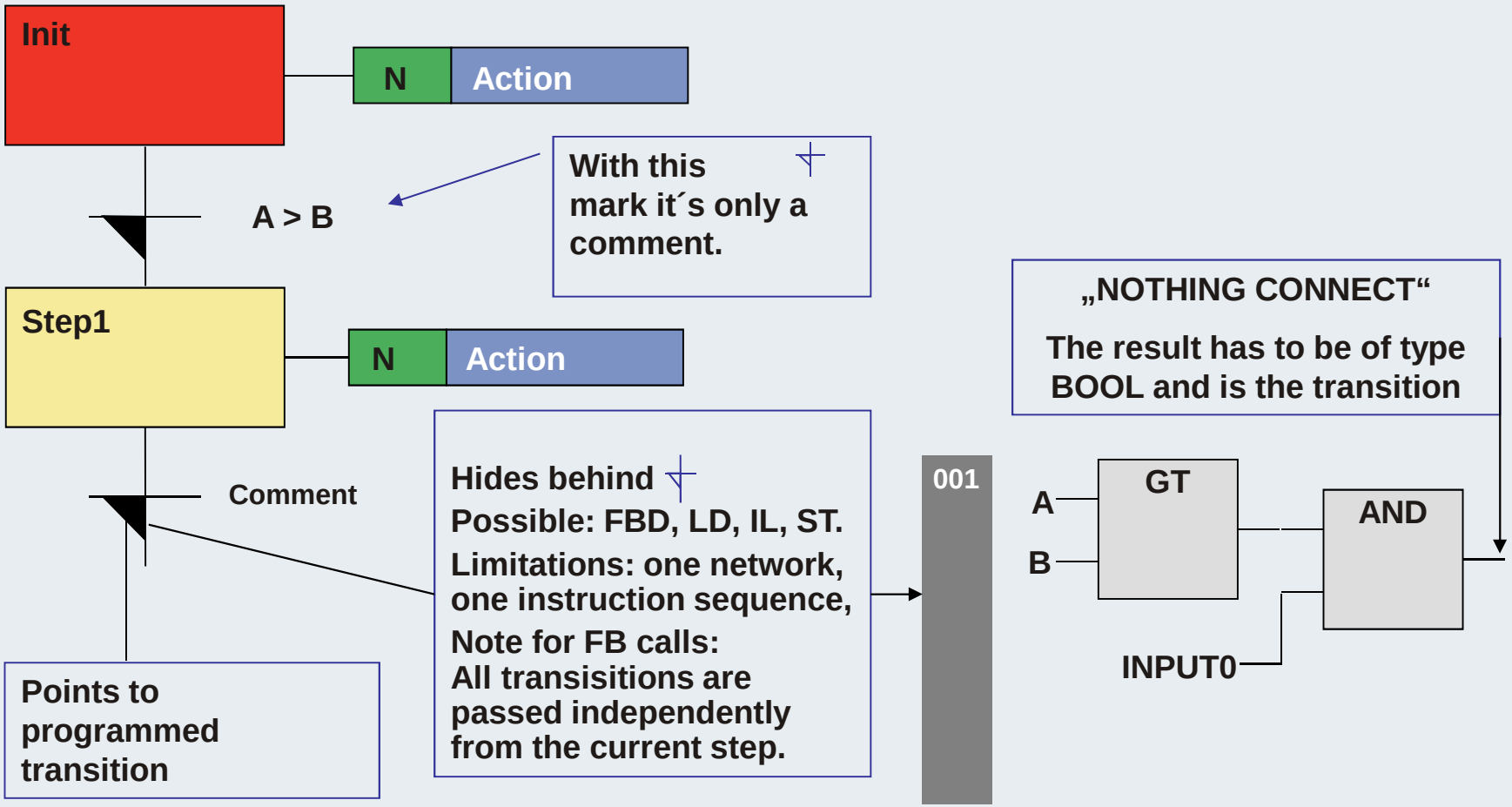
ST instruction:

The result has to be of type BOOL.

Note:
If the instructions are too long, the display will be shorten automatically.

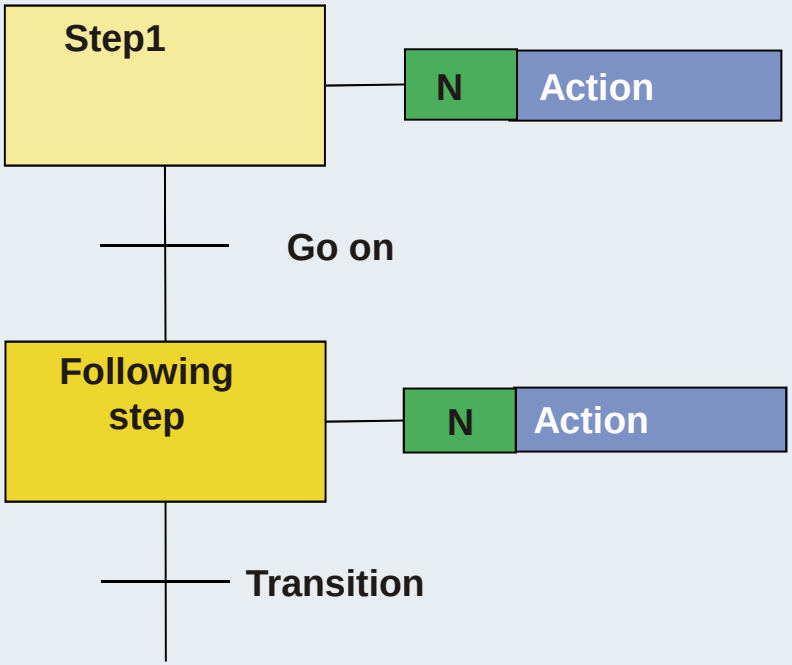
Transitions

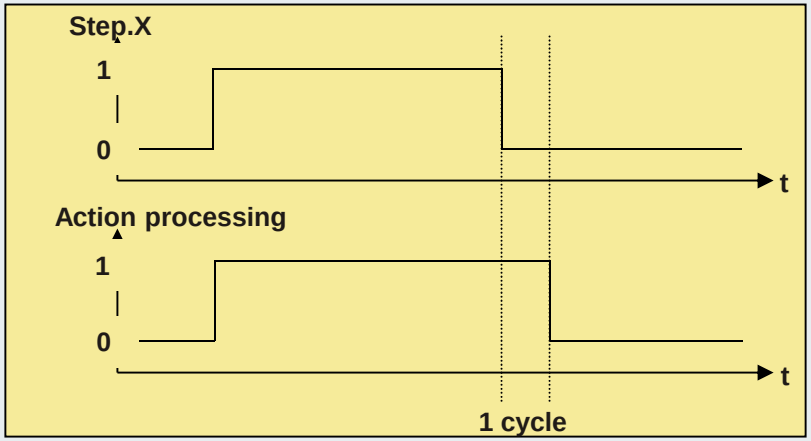
Programmed Transitions



Final-Scan

If a step is left, the processing takes exactly one more cycle. This behaviour can be used for “cleaning”in the action. Example: Reset outputs.



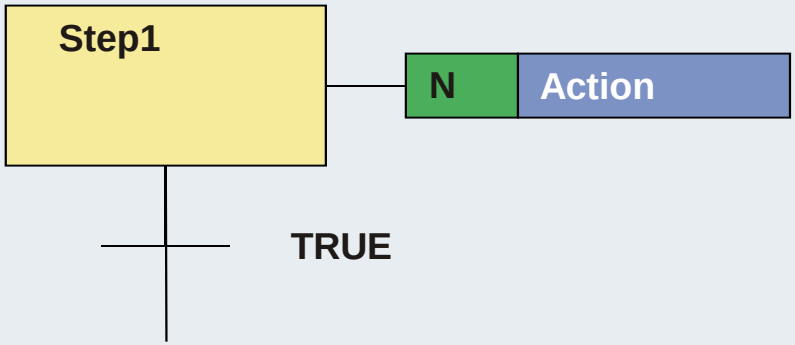


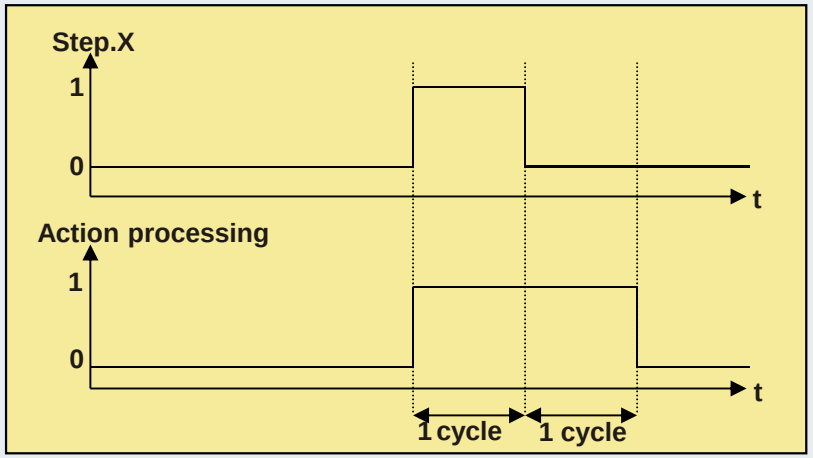
Step1.X — Output

At the last pass the step.X = FALSE. Thus the variable „Output “ is FALSE .

Final-Scan

At a certain action the final scan leads to an unwanted behaviour.





Behaviour:

```
Counter := Counter +1;
(*Counter increases at 2*)
```

Remedy: the step flag is only for one cyle 1:

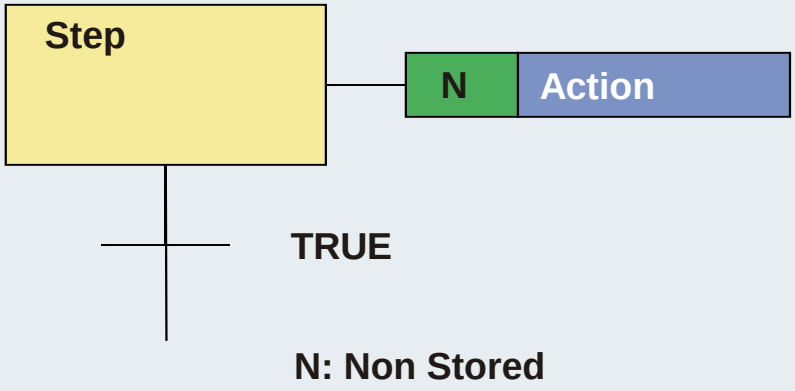
```
IF Schritt1.X THEN
    Counter := Counter +1;
END_IF
(*Counter increases at 1*)
```

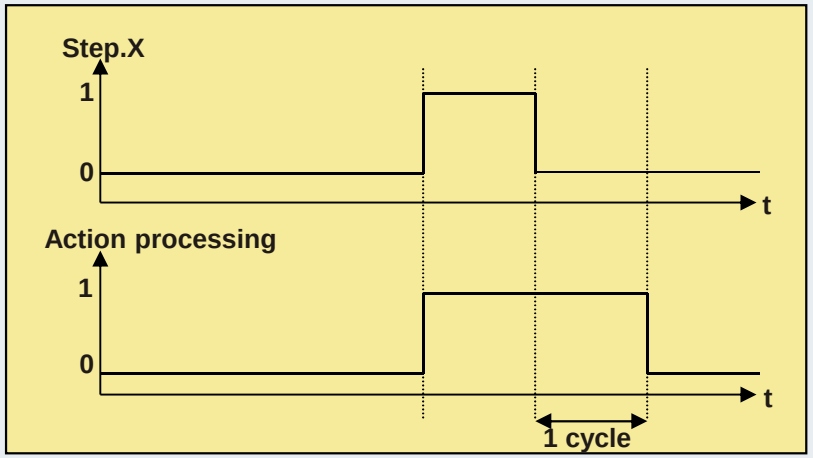
Counter := Counter +1;

Counter := Counter +1;

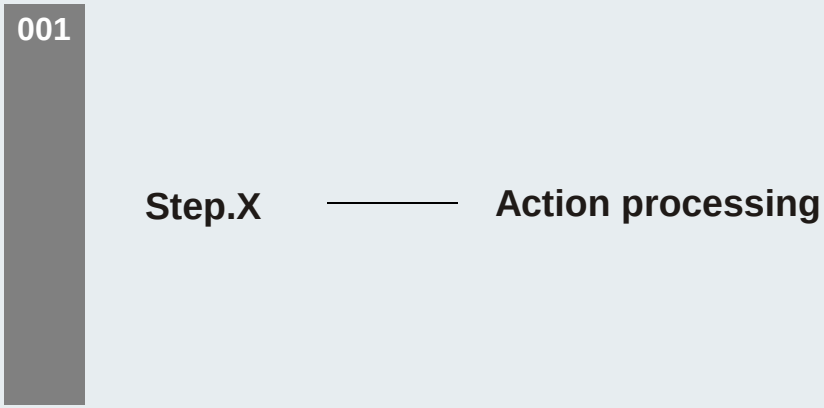
Qualifier

Controls the action processing after activating a step. **N: Non Stored**



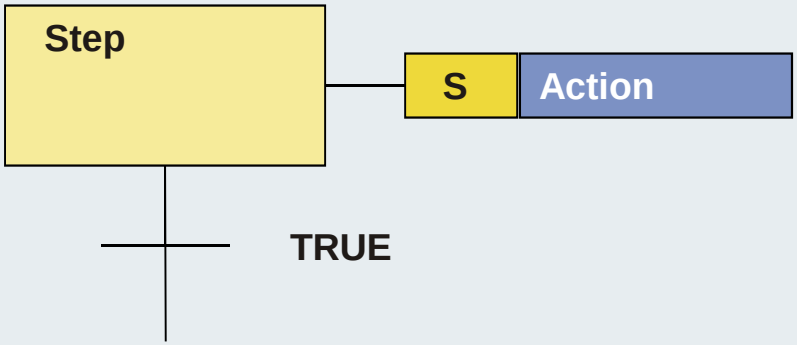


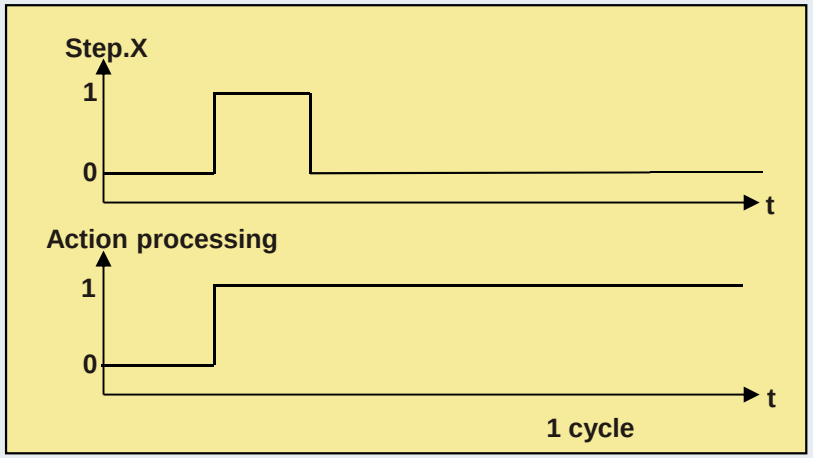
Context in FBD



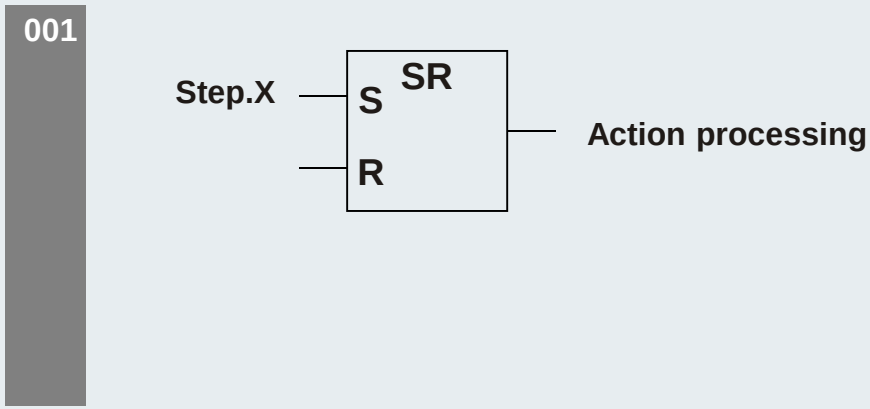
Qualifier

Controls the action processing after activating a step. **S: SET**



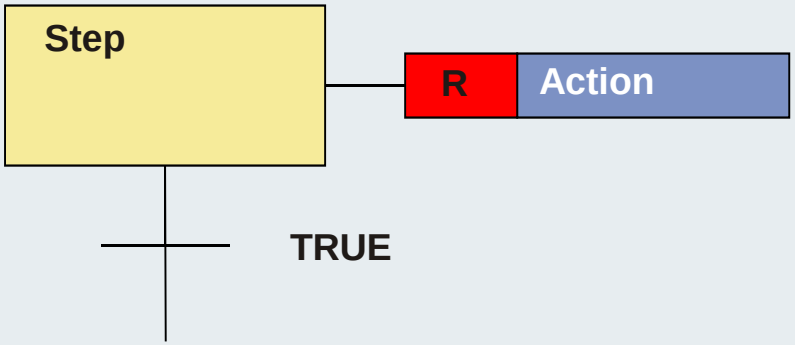


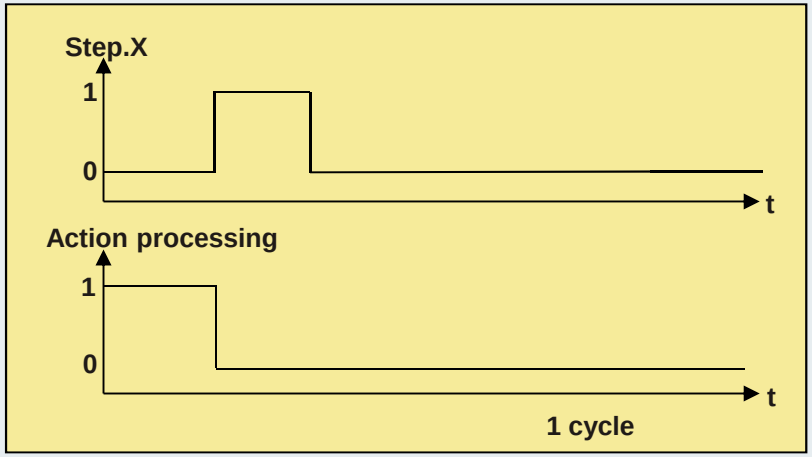
Conetxt in FBD



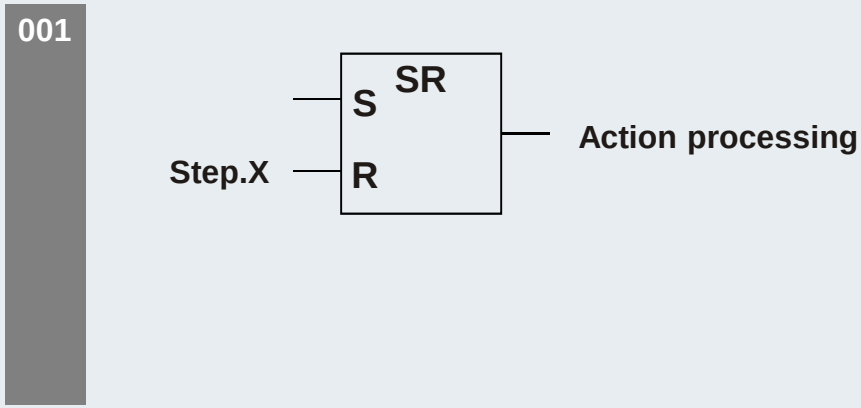
Qualifier

Controls the action processing after activating a step. R: RESET



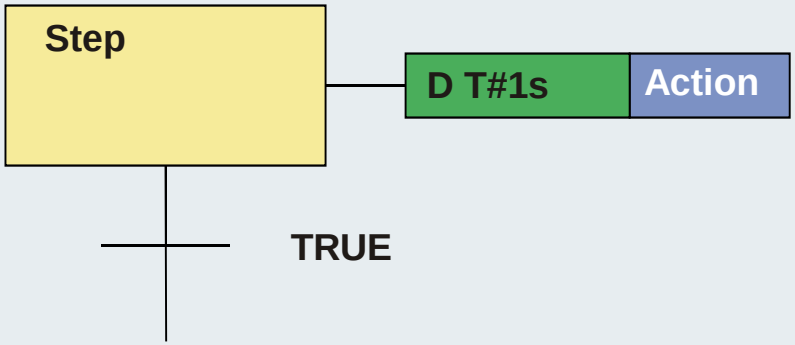


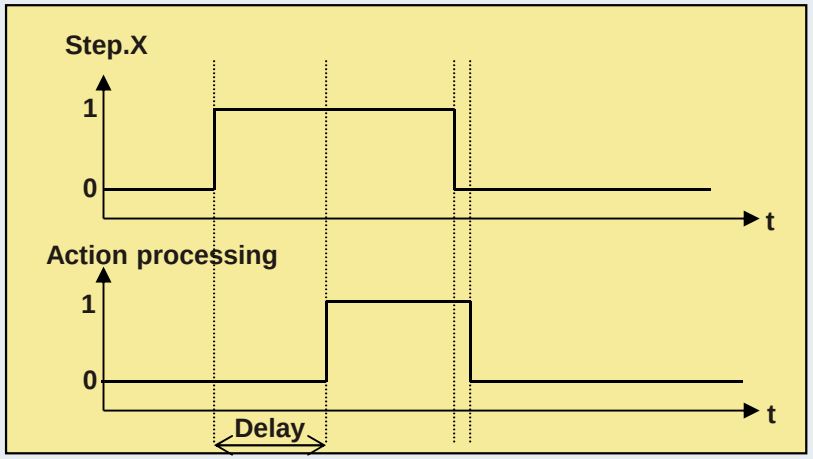
Context in FBD



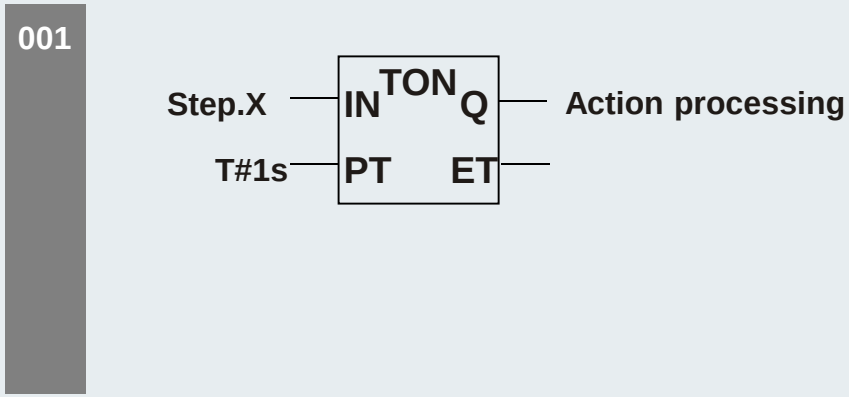
Qualifier

Controls the action processing after activating a step. D: DELAY



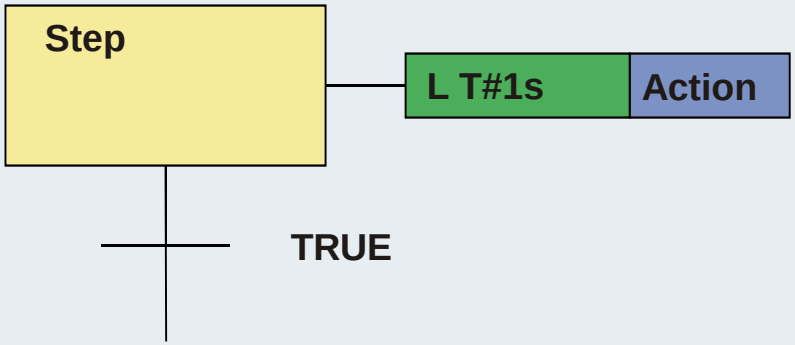


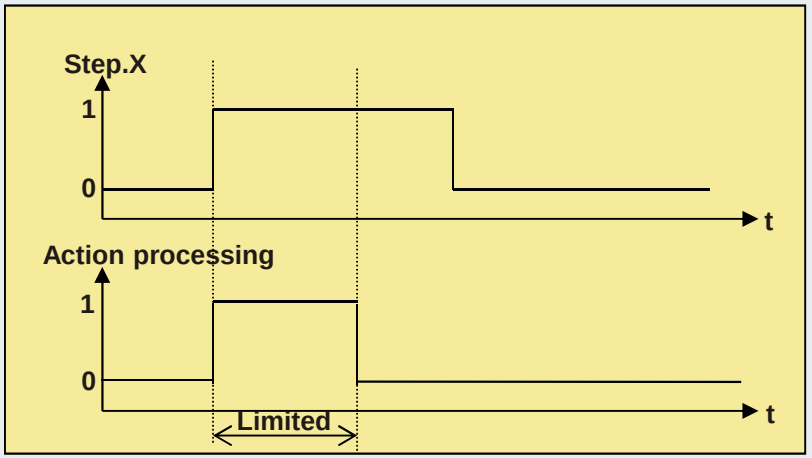
Context in FBD



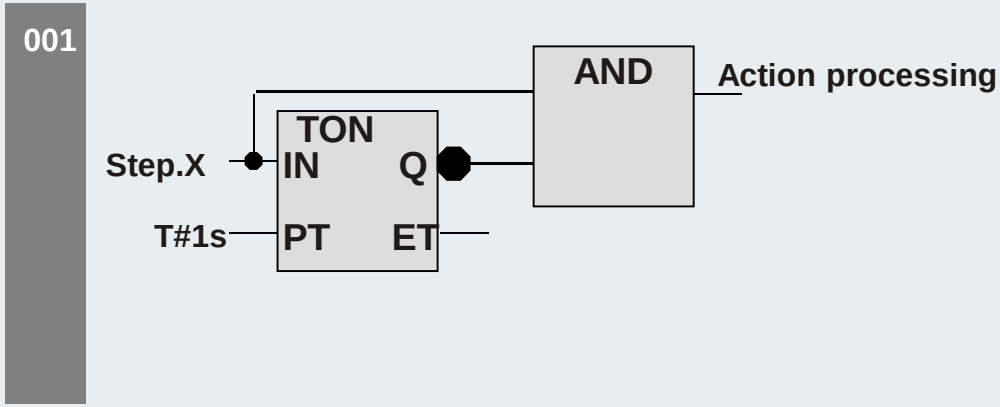
Qualifier

Controls the action processing after activating a step. L: LIMITED



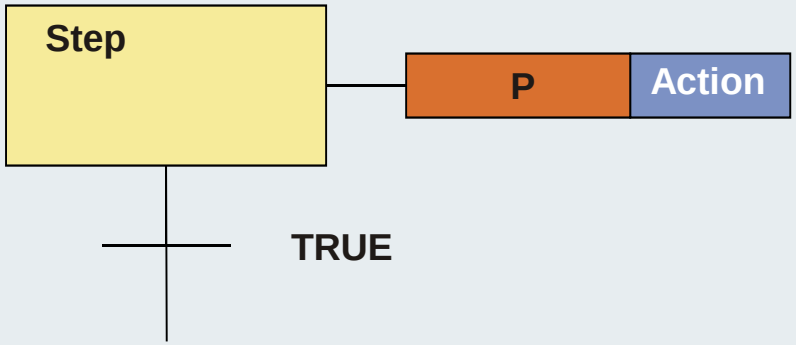


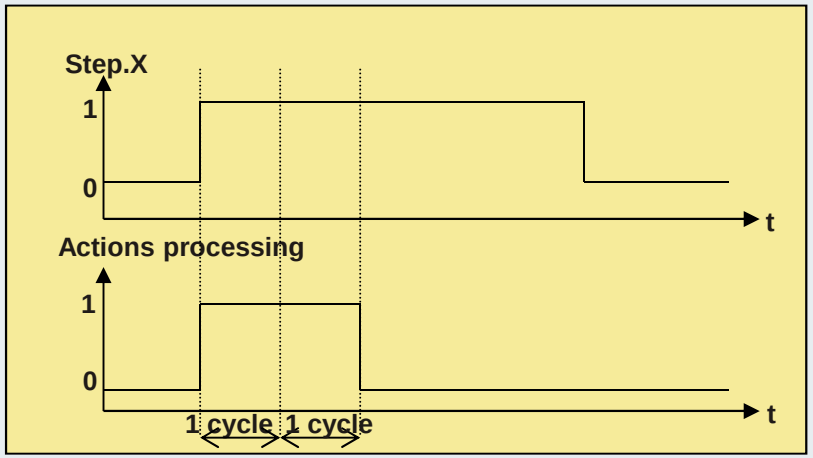
Context in FBD



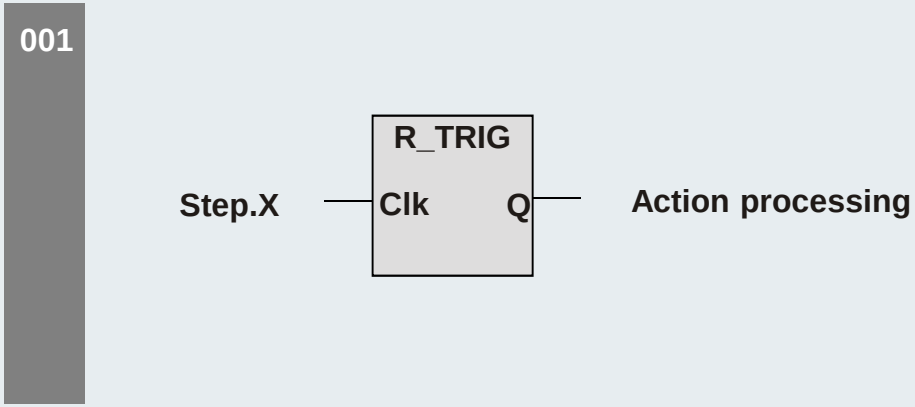
Qualifier

Controls the action processing after activating a step. P: PULSE





Context in FBD



ATTENTION: A SECOND FLOW PROCESSES!

Qualifier, Combinations

SD: Stored and delayed

DS: Delayed and stored

SL: Stored and time limited

Sequential Function chart - step diagnosis

```
VAR
    SFCEnableLimit:         BOOL;
    (*This variable is of the type BOOL. When it has the value
    TRUE, the timeouts of the steps will be registered in SFCError.
    Other timeouts will be ignored.*)

    SFCInit:                BOOL;
    (*When this boolean variable has the value TRUE the sequential
    function chart is set back to the Init step. The other SFC flags
    are reset too (initialization).
    The Init step remains active, but is not executed, for as long as
    the variable has the value TRUE. It is only when SFCInit is
    again set to FALSE that the block can be processed normally.*)
```

Sequential Function chart - step diagnosis

```
SFCReset:                    BOOL;
(* This variable, of type BOOL, behaves similarly to SFCInit.
Unlike the latter, however,  further processing takes place after
the initialization of the Init step. Thus for example the
SFCReset flag could be re-set to FALSE in the Init step *)
```

Sequential Function chart - step diagnosis

SFCQuitError: BOOL;

(* Execution of the SFC diagram is stopped for as long as this boolean variable has the value TRUE whereby a possible timeout in the variable SFCError is reset. All previous times in the active steps are reset when the variable again assumes the value FALSE.*)

SFCPause: BOOL;

(*Execution of the SFC diagram is stopped for as long as this boolean variable has the value TRUE.*)

SFCTrans: BOOL;

(* This boolean variable takes on the value TRUE when a transition is actuated.*)

Sequential Function chart - step diagnosis

SFCError: **BOOL;**

(* This Boolean variable is TRUE when a timeout has occurred in a SFC diagram. If another timeout occurs in a program after the first one, it will not be registered unless the variable SFCError is reset first.*)

SFCErrorStep: **STRING;**

This variable is of the type STRING. If SFCError registers a timeout, in this variable the name of the step is stored which has caused the timeout.

SFCErrorPOU: **STRING;**

In case of timeout this variable of the type STRING contains the name of the block in which a timeout has occurred.

Sequential Function chart - step diagnosis

SFCCurrentStep: : **STRING;**

This variable is of the type STRING. The name of the step is stored in this variable which is active, independently of the time monitoring. In the case of simultaneous sequences the step is stored in the branch on the outer right.
No further timeout will be registered if a timeout occurs and the variable SFCError is not reset again.

Sequential Function chart

```
SFCErrorAnalyzation:        STRING;
(* This variable, of type STRING, provides the transition
expression as well as every variable in an assembled
expression which gives a FALSE result for the transition and
thus produces a timeout in the preceding step. A requirement
for this is declaration of the SFCError flag, which registers the
timeout. SFCErrorAnalyzation refers back to a function called
AppedErrorString in the TcSystem.Lib library. The output string
separates multiple components with the symbol “|”. *)
```

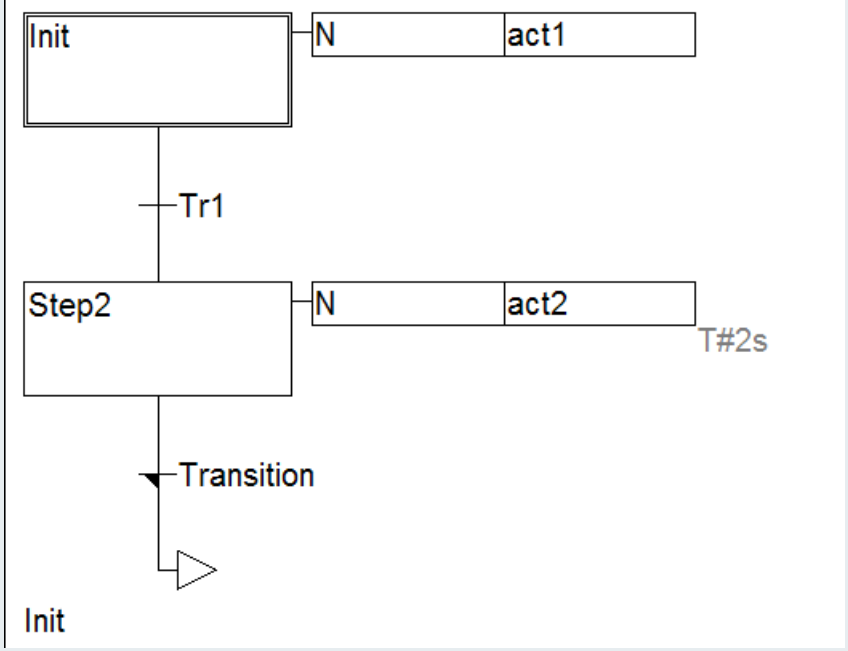
Sequential Function chart

```
    SFCTip:              BOOL;
    SFCTipMode:                  BOOL;
    (* This variables of type BOOL allow inching mode of the SFC.
    When this is switched on by SFCTipMode=TRUE, it is only
    possible to skip to the next step if SFCTip is set to TRUE. As
    long as SFCTipMode is set to FALSE, it is possible to skip even
    over transitions.*)
END_VAR
```

Example Diagnosis

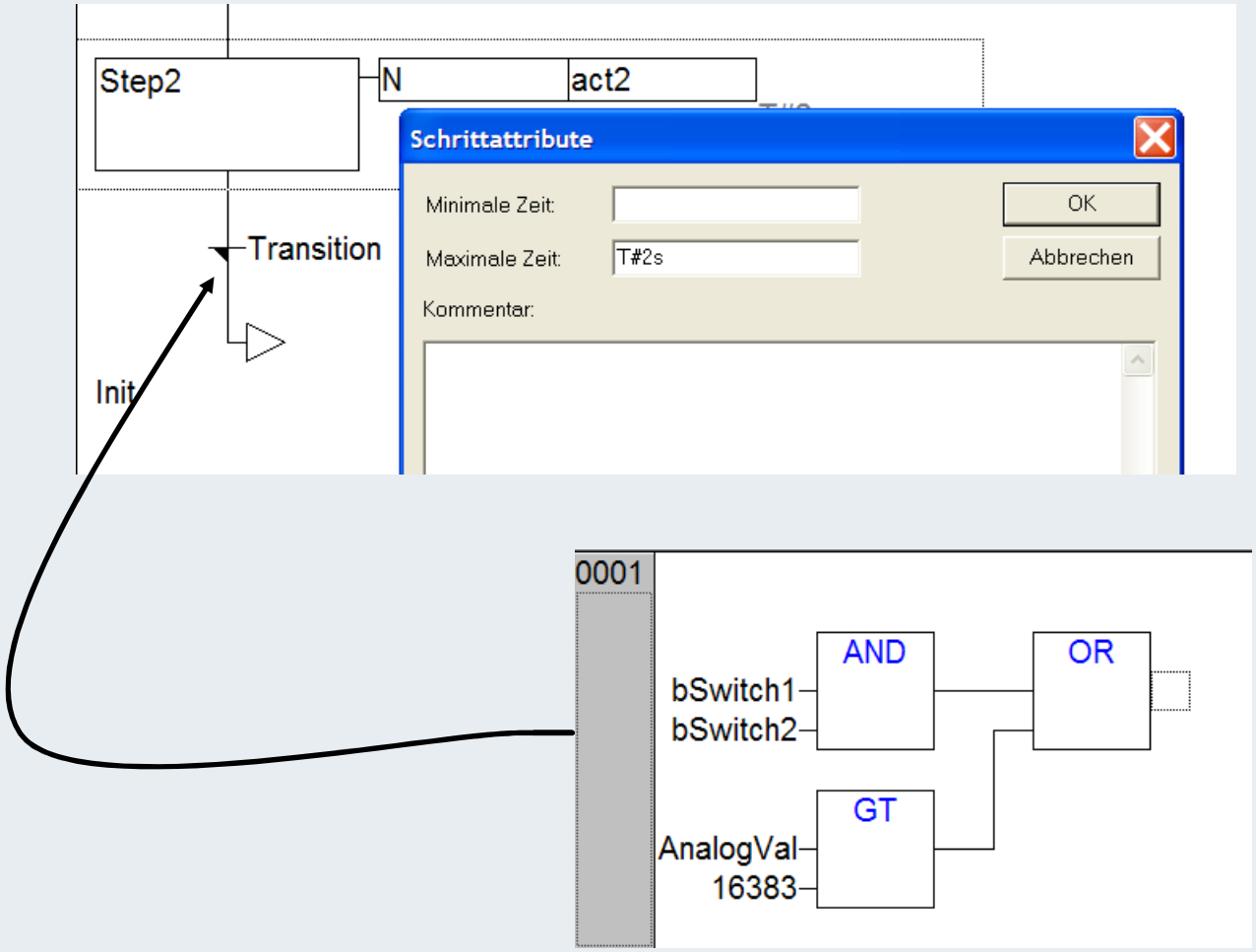
```
0001 PROGRAM demo
0002 VAR
0003     SfcEnableLimit :BOOL;
0004     SfcError :BOOL;
0005     SfcErrorStep :STRING;
0006     SfcErrorAnalyzation :STRING;
0007     SfcErrorPou :STRING;
0008
```

Declare variable

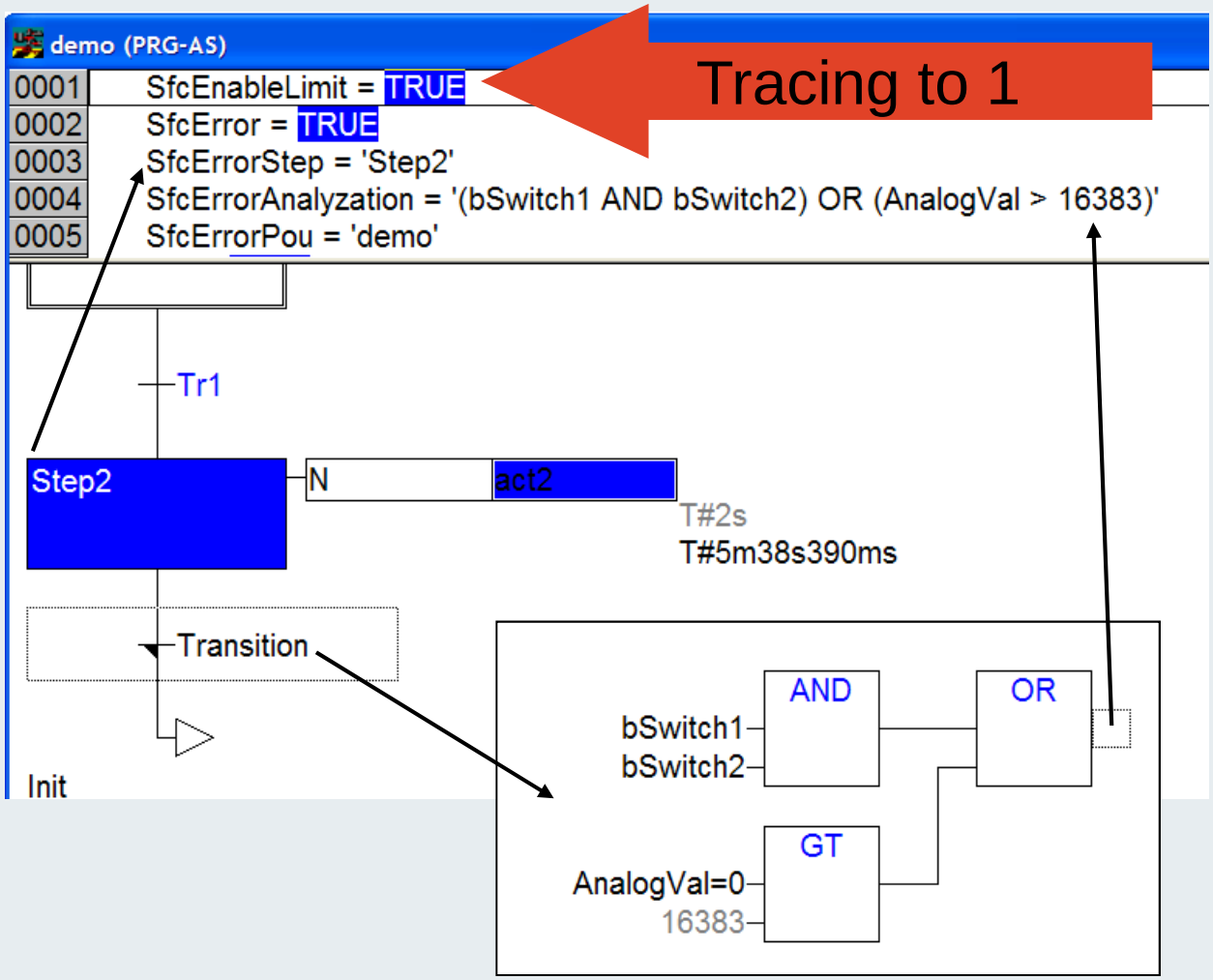


Example Diagnosis

Set step attributes for the step to be observed.

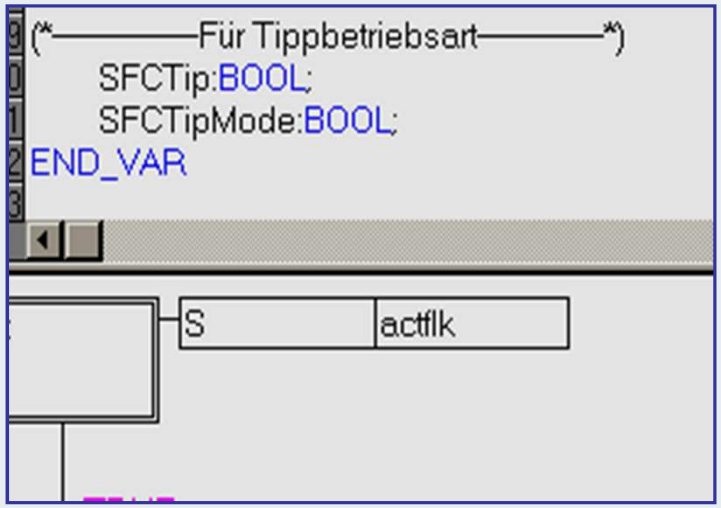


Example Diagnosis



Tip mode

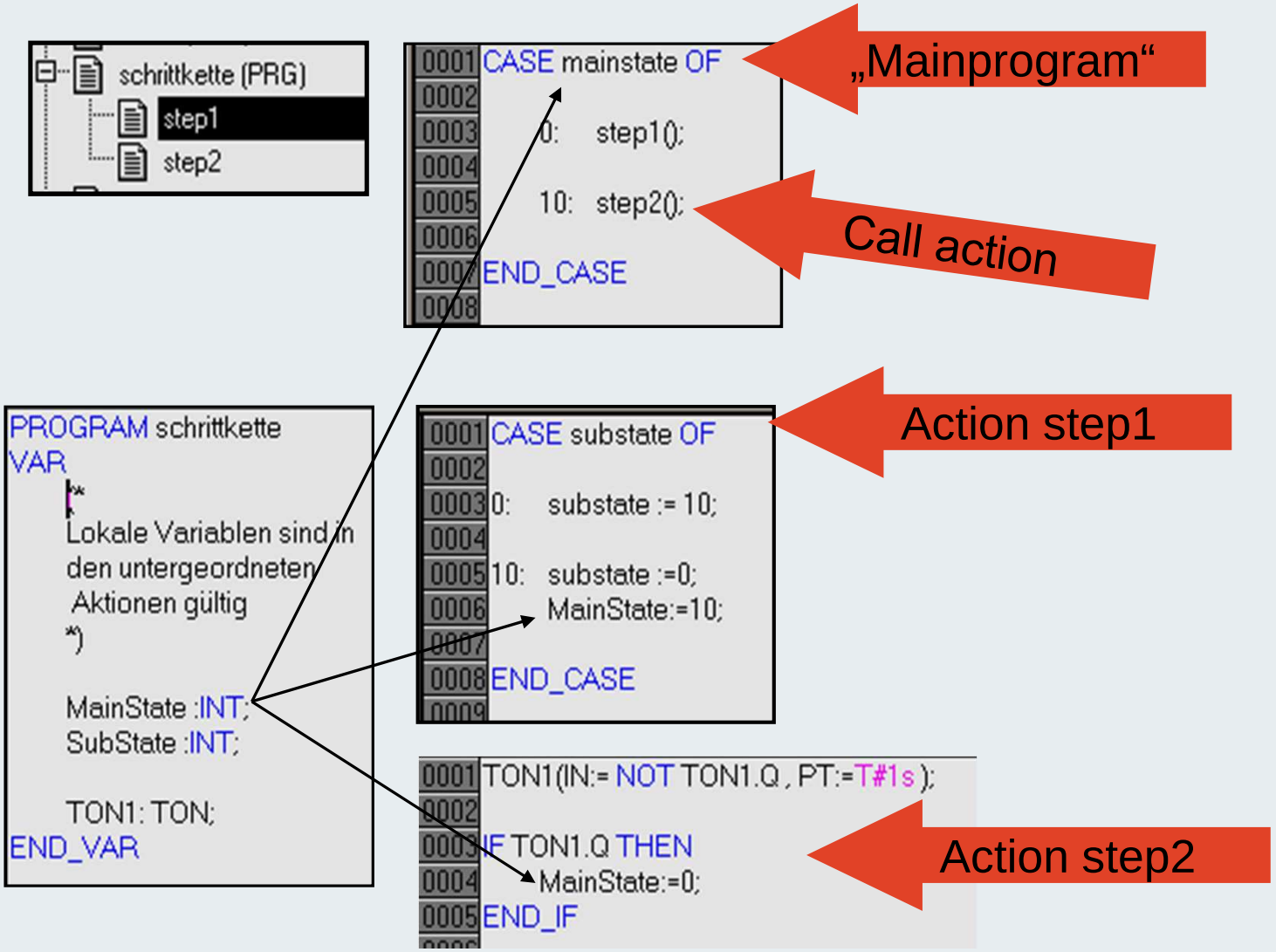
Insert implicit variable:



Effect:

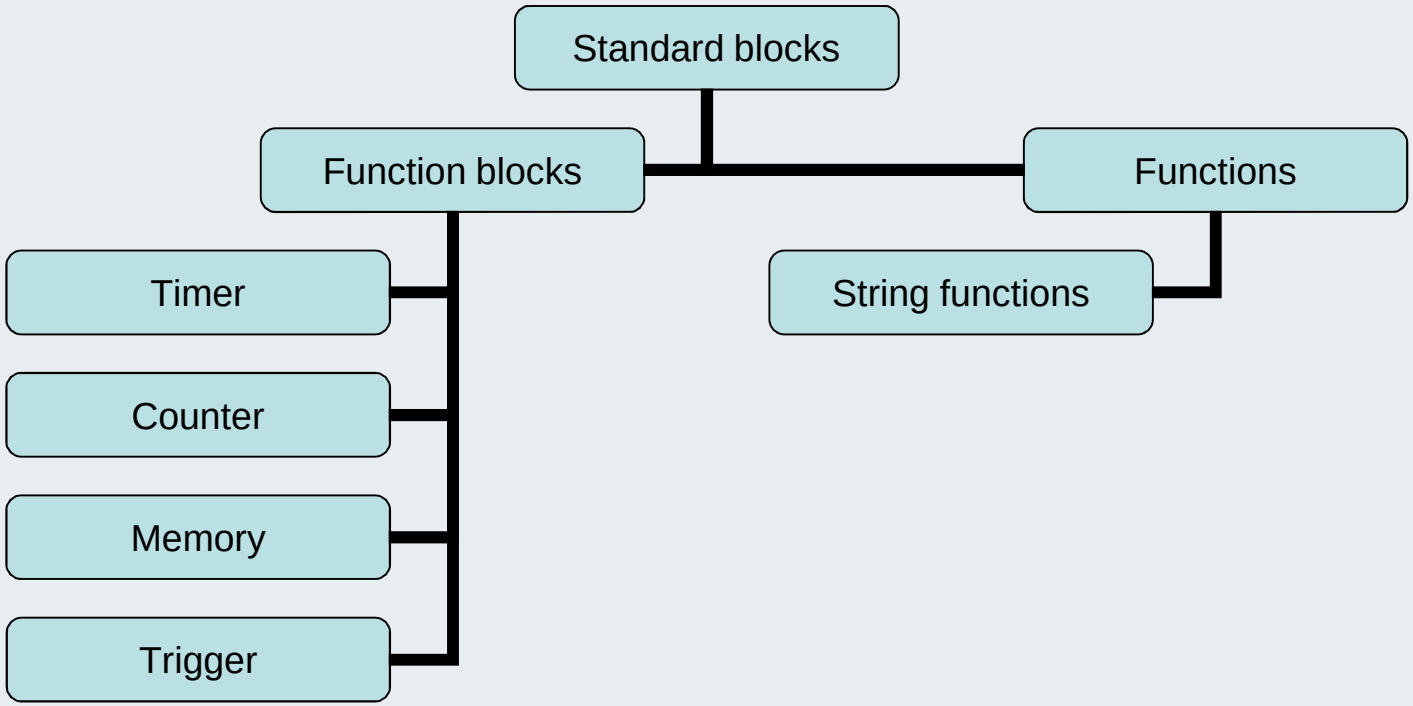
SFCTipMode	SFCTip	Transition	Effect
TRUE	FALSE	TRUE	Process stays in current step
TRUE	TRUE	TRUE	Change to next step
TRUE	TRUE	FALSE	Change to next step
FALSE	TRUE	FALSE	Process stays in current step
FALSE	FALSE	TRUE	Change to next step

Actions also in other IEC languages possible!

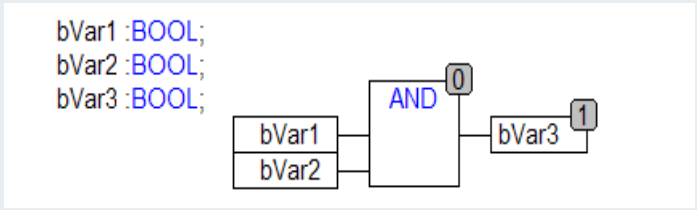
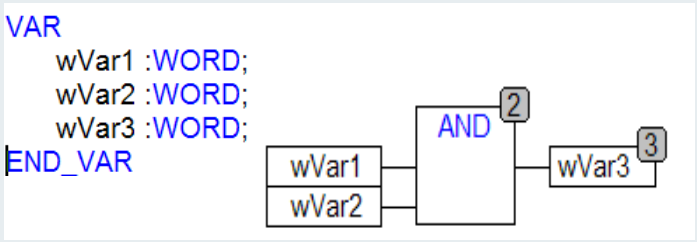
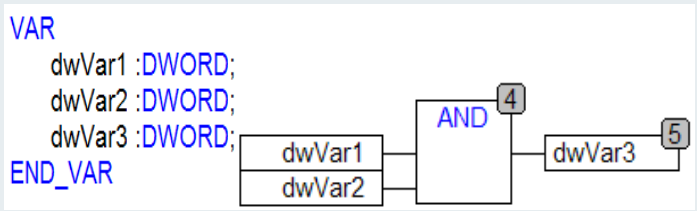


Стандартные МЭК операторы и ФБ

Функциональные блоки и функции доступны благодаря встроенной библиотеке Standard.LIB. Она подключаентся автоматически при создании нового проекта ПЛК.



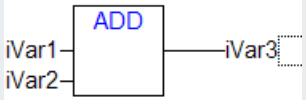
Логические операторы

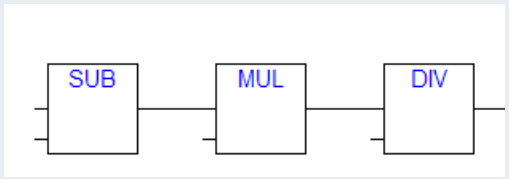
FUP / CFC	ST	Комментарии
	bVar3 := bVar1 AND bVar2 ;	BOOL AND
	wVar3 := wVar1 AND wVar2 ;	WORD AND
	dwVar3 := dwVar1 AND dwVar2 ;	DWORD AND

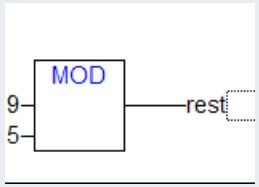
Числовые операторы

FUP / CFC

```
VAR
    iVar1 :INT;
    iVar2 :INT;
    iVar3 :INT;
END_VAR
```







ST

```
iVar3 := iVar1 + iVar2 ;
```

- * /

```
rest := 9 MOD 5;
```

Комментарии

Арифметические операторы

Возможно переполнение, в соответствии с типом (INT).

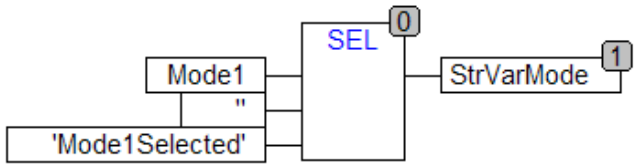
Остаток от деления (в примере 4)

Операторы выбора, SEL

Оператор

FUP / CFC

```
0001 PROGRAM MAIN
0002 VAR
0003     Mode1 :BOOL;
0004     StrVarMode :STRING;
0005 END_VAR
0006
```

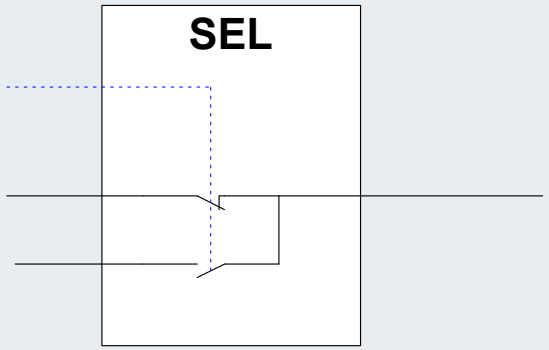


ST

StrVarMode := SEL(Mode1, ‘’, ‚Mode1Selected);

Комментарии

Выбранный вход переключается



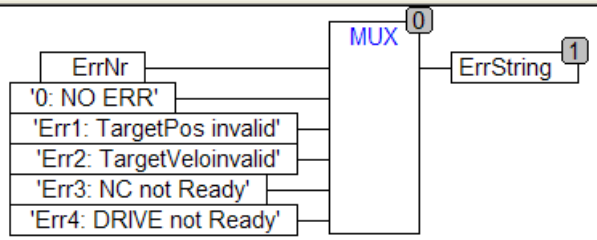
Пример:

If Mode1 = TRUE , StrVarmode равен ‘Mode1Selected‘ иначе пустая строка.

Операторы выбора, MUX

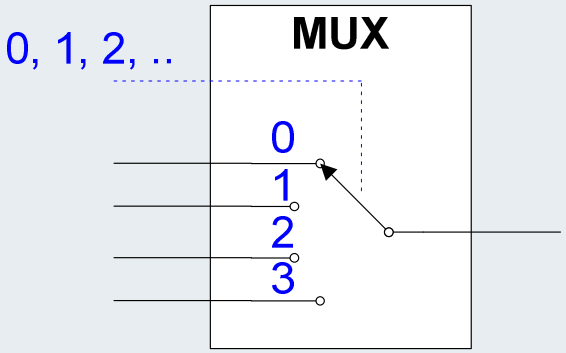
FUP /CFC

```
0001 PROGRAM MAIN
0002 VAR
0003     ErrNr    :INT;
0004     ErrString :STRING;
0005
0006 END_VAR
```




```
ErrNr    = 32767
ErrString = 'Err4: DRIVE not Ready'
ErrNr    = 0
ErrString = '0: NO ERR'
ErrNr    = 1
ErrString = 'Err1: TargetPos invalid'
ErrNr    = 2
ErrString = 'Err2: TargetVeloinvalid'
```

Выбор переменной типа integer, которой переключаются входы на выход

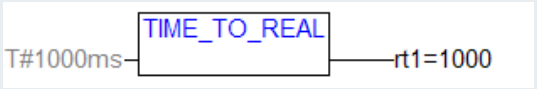
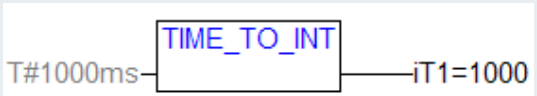
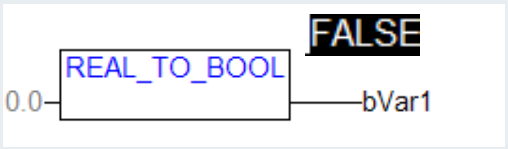
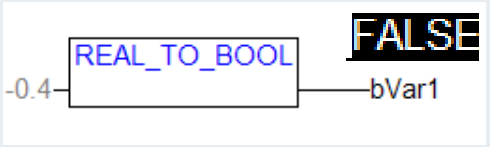
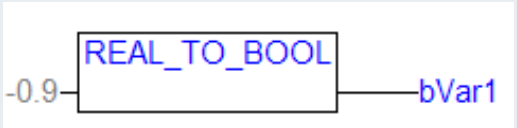


ST:

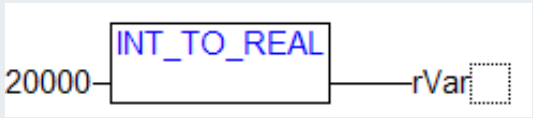
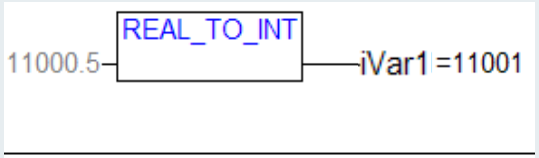
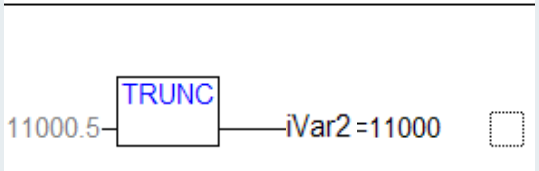
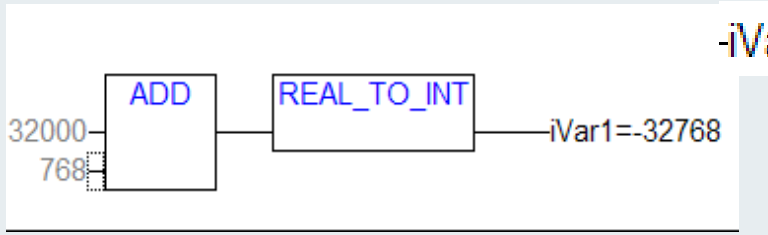
```
ErrString := MUX(ErrNr ,
                 '0: NO ERR',
                 'Err1: TargetPos invalid',
                 'Err2: TargetVeloinvalid',
                 'Err3: NC not Ready',
                 'Err4: DRIVE not Ready');
```

Online:

Преобразование типов

Оператор	ST	Комментарии
	`rT1 := TIME_TO_REAL(T#1s);`	**Значение времени в ms rt1:REAL**
	`iT1 := TIME_TO_INT(T#1s);`	**iT1:INT**
	`bVar1:= REAL_TO_BOOL(0.0);`	**bVar1:BOOL**
		**Округление:** **0.4 становится FALSE,** **>=0.5 становится TRUE**
		

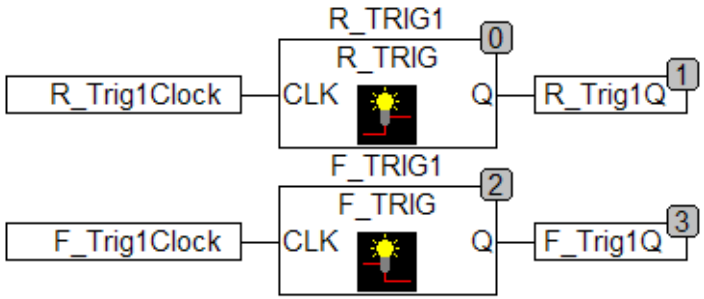
Преобразование типов

FUP / CFC	ST	Комментарии
	rVar:= INT_TO_REAL(20000);	**Результат 20000.0** **Implicit conversion also possible**
	iVar1:= REAL_TO_INT(11000.5);	**Конвертирование в INT с округлением up/down**
	iVar2 := TRUNC(11000.5);	**Конвертирование в INT без округлением up/down**
		**В соответствии с переполнением** **(отрицательный результат)**

Обработка фронта сигнала R_TRIG F_TRIG

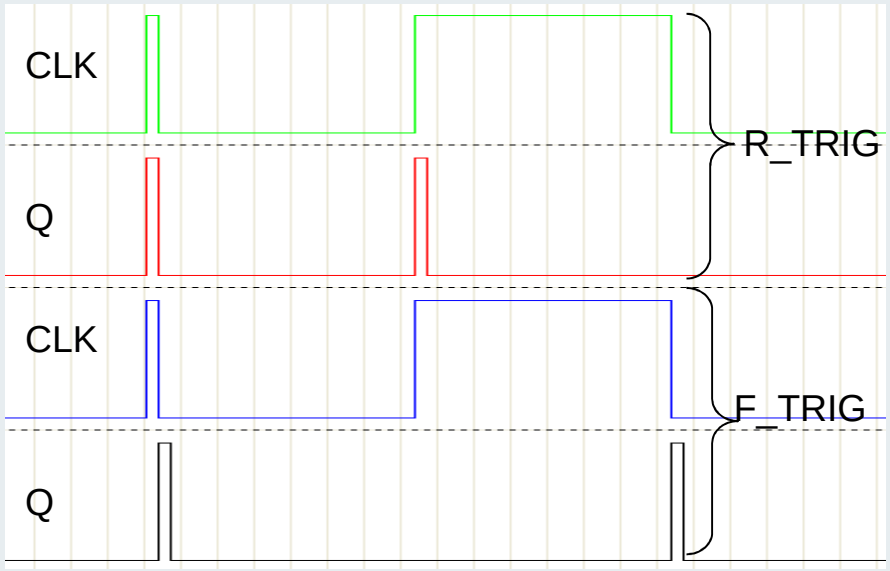
FUP / CFC

```
VAR
    R_TRIG1:        R_TRIG;     F_TRIG1: F_TRIG;
    R_Trig1Clock :  BOOL;       F_Trig1Clock: BOOL;
    R_Trig1Q :      BOOL;       F_Trig1Q: BOOL;
END_VAR
```



ST

```
R_TRIG1(CLK:=R_Trig1Clock  , Q=>R_Trig1Q  );
F_TRIG1(CLK:=F_Trig1Clock  , Q=>F_Trig1Q  );
```

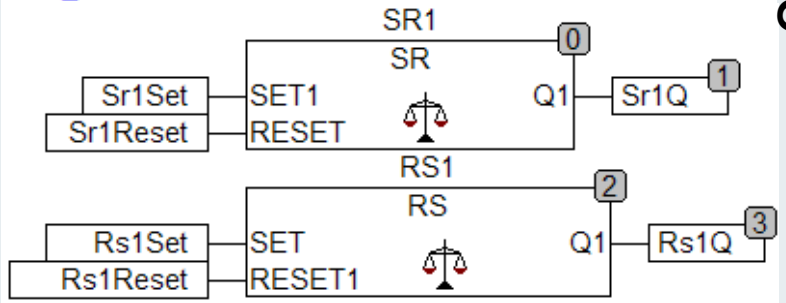


CLK	**BOOL**	**Trigger input**	**R_TRIG: Rising Edge,**
Q	**BOOL**	**Signal output (one for 1 PLC cycle)**	**F_TRIG: Falling Edge**

Триггер состояния SR RS

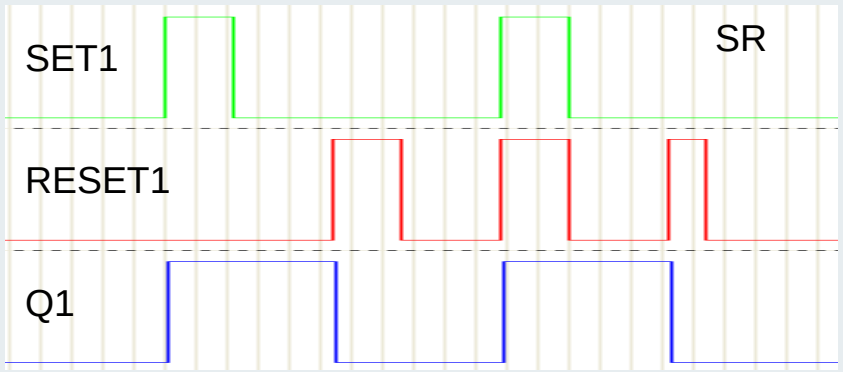
FUP / CFC

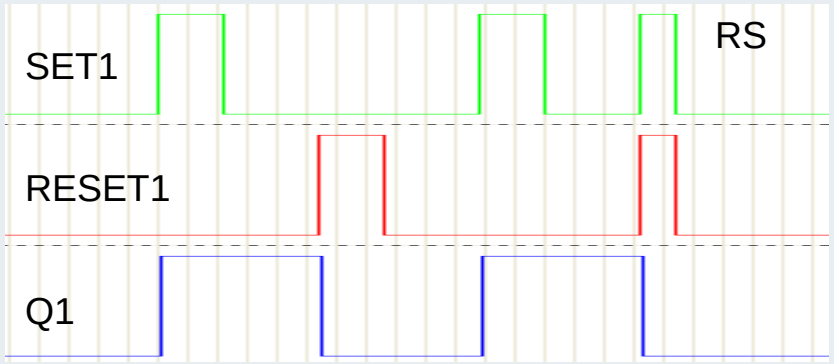
```
VAR
    SR1: SR;          RS1: RS;
    Sr1Set: BOOL;     Rs1Set: BOOL;
    Sr1Reset: BOOL;   Rs1Reset: BOOL;
    Sr1Q: BOOL;       Rs1Q: BOOL;
END_VAR
```



ST

```
SR1(SET1:=Sr1Set , RESET:=Sr1Set , Q1=>Sr1Q );
RS1(SET:=Rs1Set , RESET1:=Rs1Reset , Q1=>Rs1Q );
```





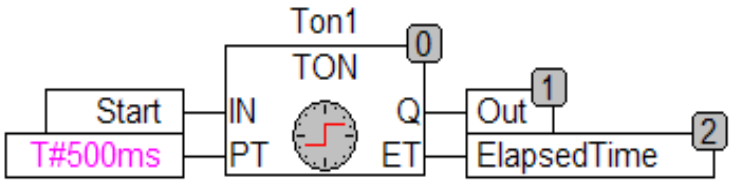
SET1	**BOOL**	**Set**
RESET1	**BOOL**	**Reset**
Q1	**BOOL**	**Output**

SR: Приоритет set

RS: Приоритет reset

Временная задержка на включение TON

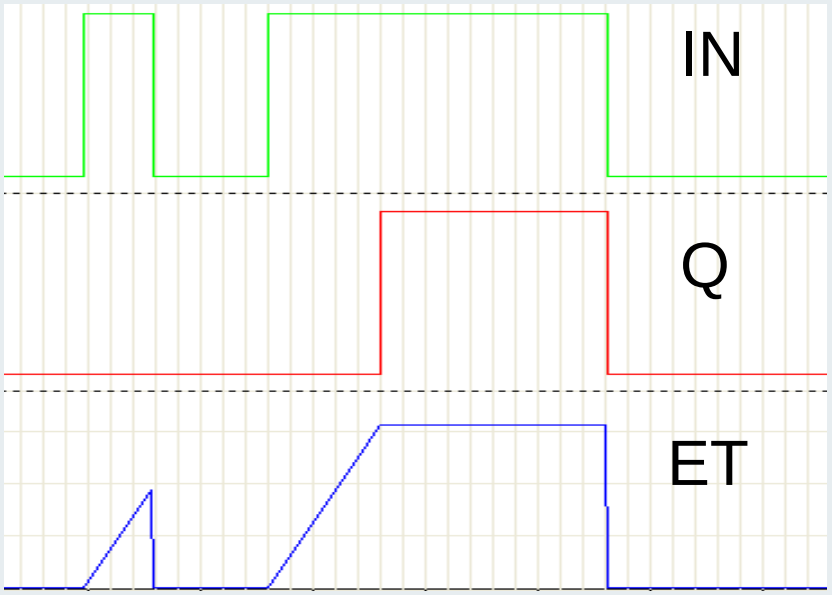
```
VAR
    Ton1: TON;
    Start: BOOL;
    Out: BOOL;
    ElapsedTime: TIME;
END_VAR
```



FUP / CFC

```
Ton1(   IN:=Start ,
        PT:=T#500ms ,
        Q=>Out ,
        ET=>ElapsedTime );
```

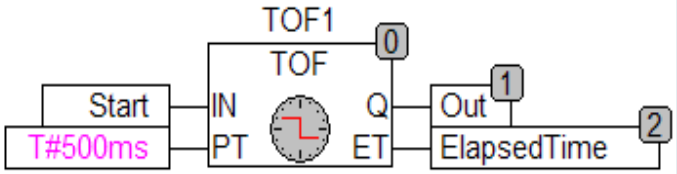
ST



IN	**BOOL**	**Сигнальный вход**	**Старт по переднему фронту**
PT	**TIME**	**Установленное время задержки**	
Q	**BOOL**	**Выход**	
ET	**TIME**	**Значение текущего времени**	

Временная задержка на выключение TOF

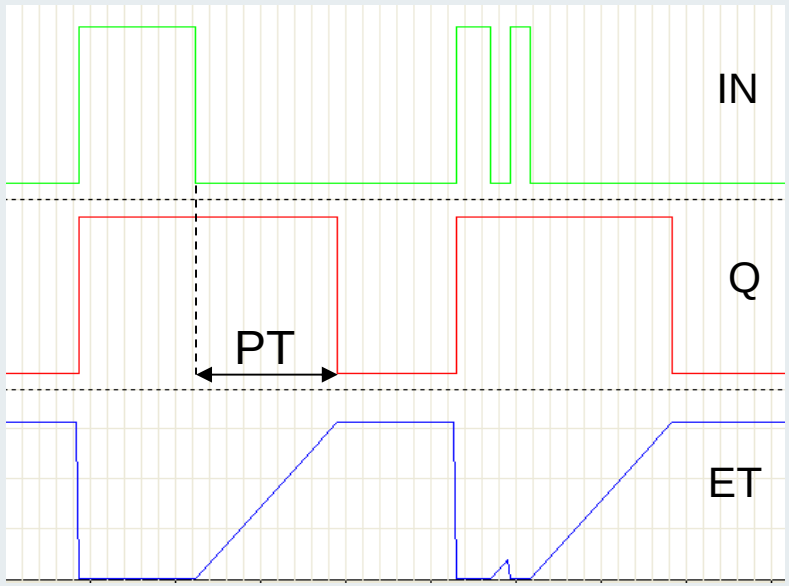
```
VAR
    TOF1: TOF;
    Start: BOOL;
    Out: BOOL;
    ElapsedTime: TIME;
END_VAR
```



FUP / CFC

```
TOF1(  IN:=Start ,
       PT:=T#500ms ,
       Q=>Out ,
       ET=>ElapsedTime  );
```

ST



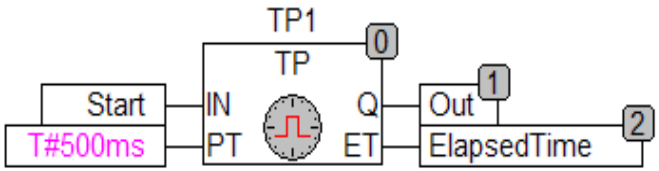
IN	BOOL	Сигнальный вход
PT	TIME	Установленное время задержки
Q	BOOL	Выход
ET	TIME	Значение текущего времени

Старт по обратному фронту на IN, новый фронт продлевает выходной сигнал

Триггер TP

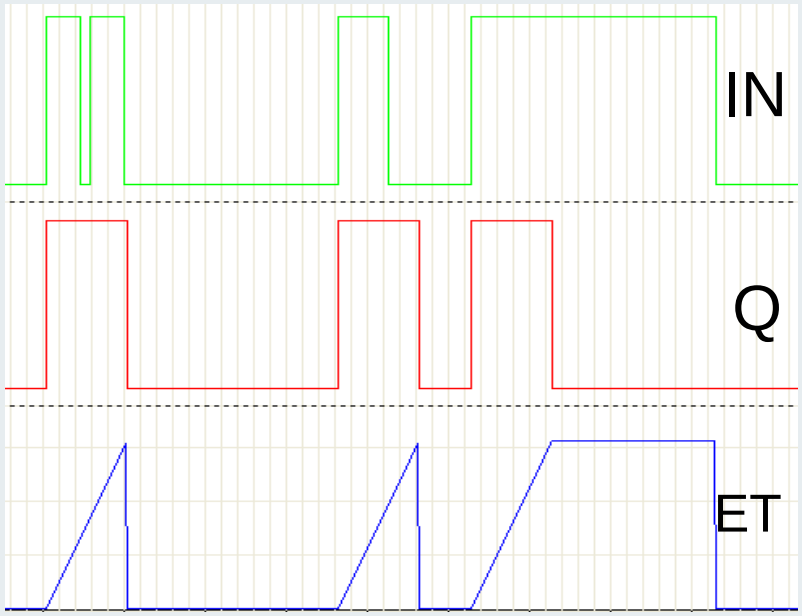
```
VAR
    TP1: TP;
    Start: BOOL;
    Out: BOOL;
    ElapsedTime: TIME;
END_VAR
```

FUP / CFC



ST

```
Tp1(    IN:=Start ,
        PT:=T#500ms ,
        Q=>Out ,
        ET=>ElapsedTime );
```

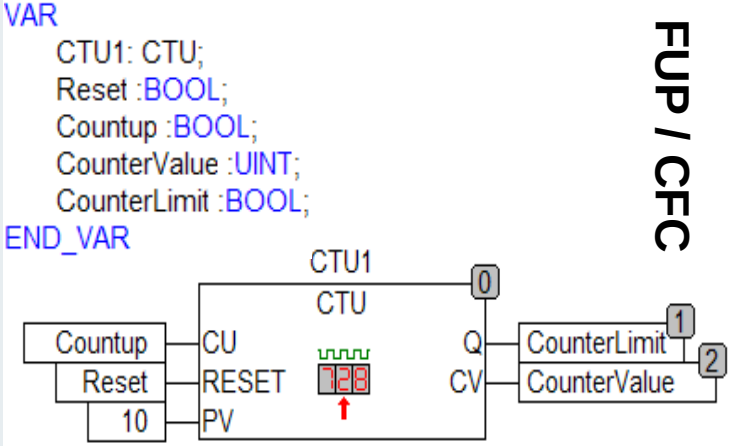


IN	BOOL	Сигнальный вход
PT	TIME	Установленное время импульса
Q	BOOL	Выход
ET	TIME	Значение текущего времени

Старт по переднему фронту, после старта не подтверждается

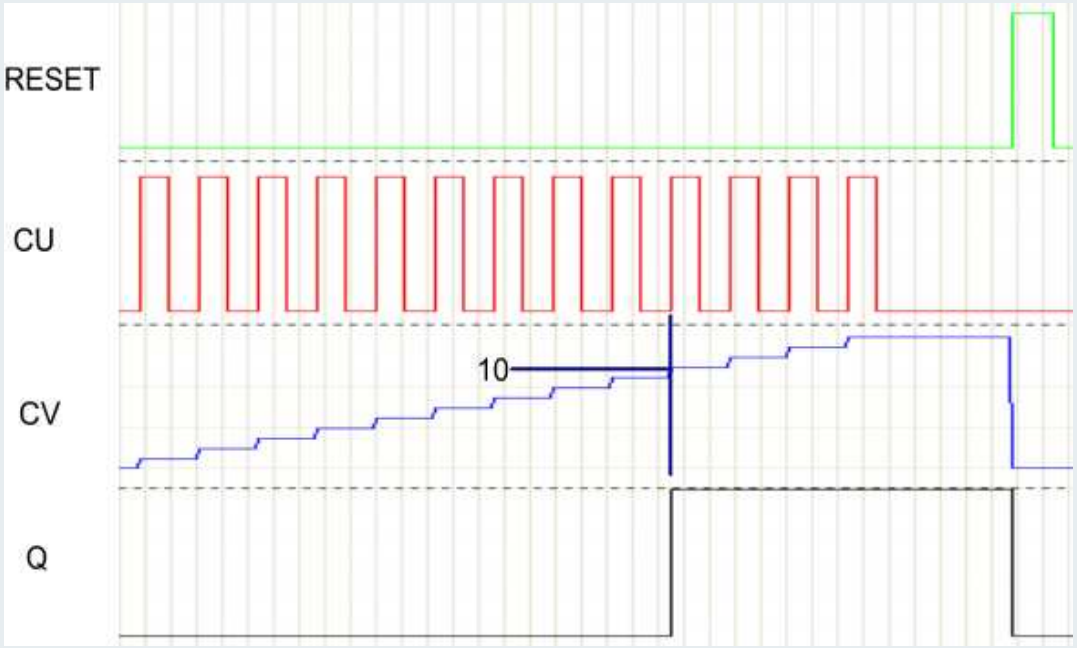
Прямой счетчик CTU

FUP / CFC



ST

```
CTU1(
    CU:=        Countup ,
    RESET:=     Reset ,
    PV:=        10,
    Q=>         CounterLimit ,
    CV=>        CounterValue);
```



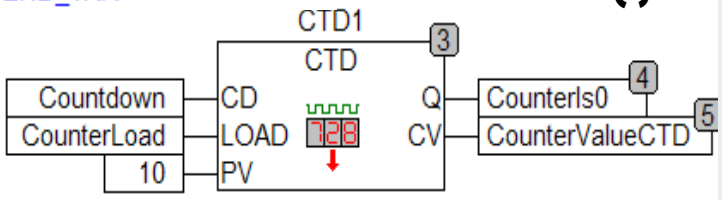
CU	**BOOL**	**Count Up, входной сигнал**
RESET	**BOOL**	**Reset счетчика**
PV	**UINT**	**Предельное значение**
Q	**BOOL**	**значение счетчика достигло PV**
CV	**WORD**	**Текущее значение счетчика**

Counter with edge at CU, output is set if counter value has the same value as PV.

Обратный счетчик CTD

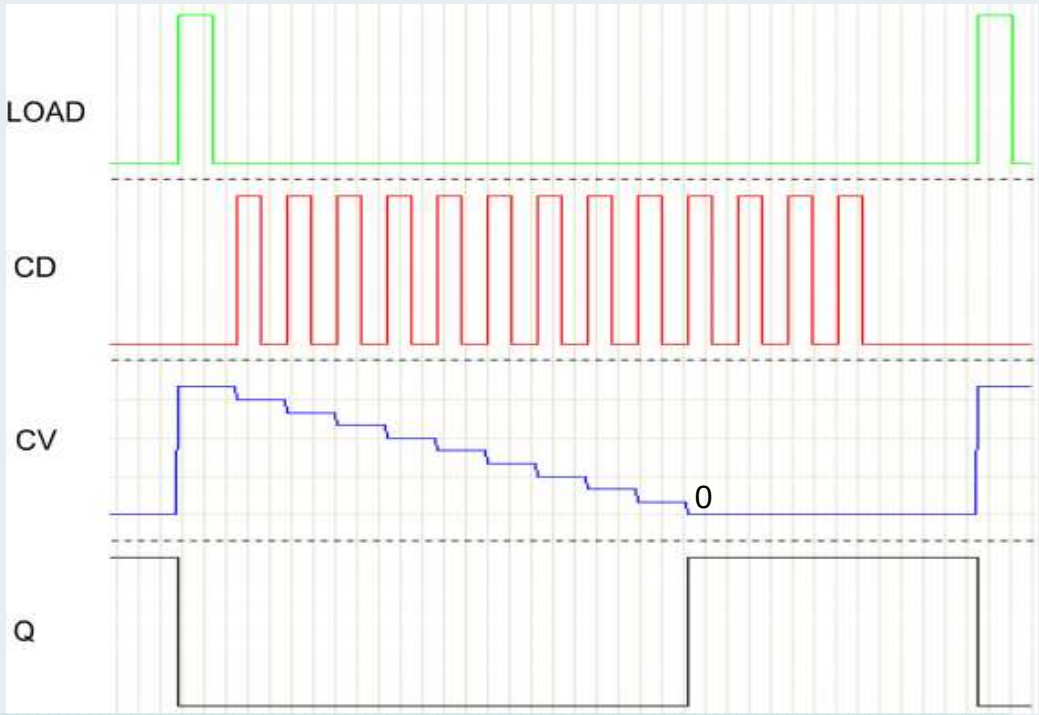
FUP / CFC

```
VAR
    CTD1 :CTD;
    CounterLoad :BOOL;
    Countdown :BOOL;
    CounterValueCTD :UINT;
    CounterIs0 :BOOL;
END_VAR
```



ST

```
CTD1(
    CD:=        Countdown ,
    LOAD:=      CounterLoad ,
    PV:=        10,
    Q=>         CounterIs0 ,
    CV=>        CounterValueCTD);
```



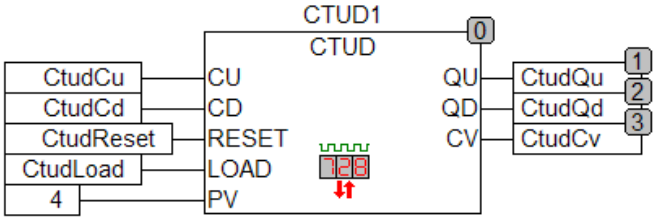
CD	BOOL	Count Down, входной сигнал
LOAD	BOOL	Загрузить в счетчик значение PV
PV	UINT	Предустановленное значение
Q	BOOL	Значение счетчика стало 0
CV	UINT	Текущее значение счетчика

LOAD загружает счетчик, Счетчик уменьшается по фронту CD, Q = True если счетчик обнуляется.

Реверсивный счетчик CTUD

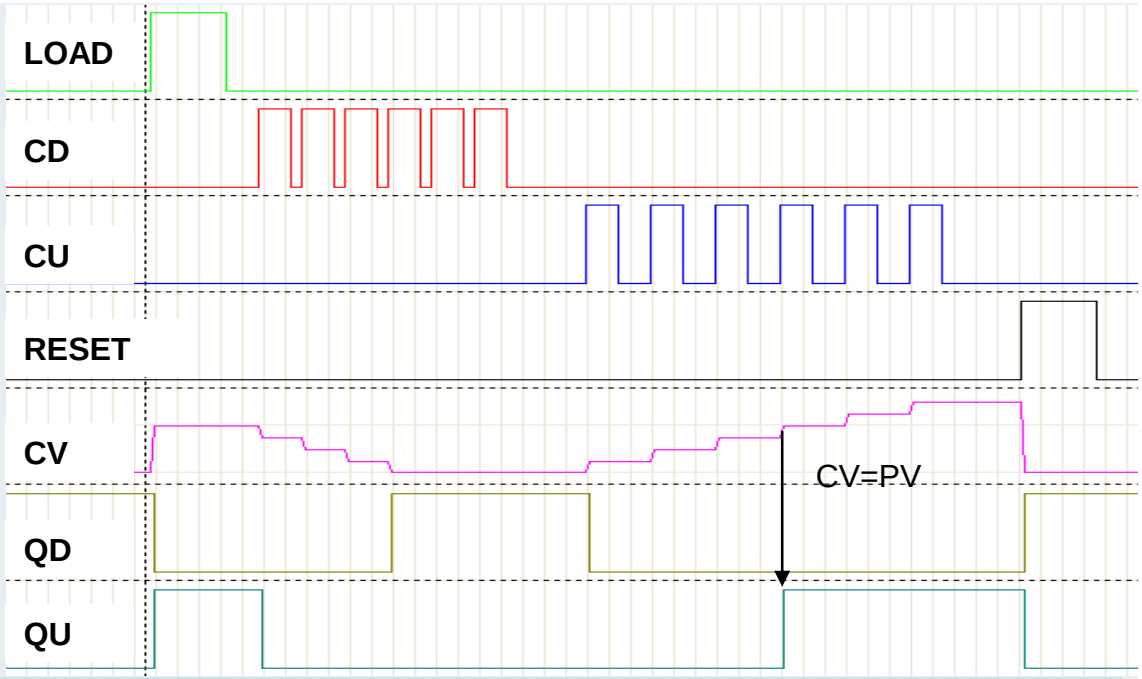
FUP / CFC

```
VAR
    CTUD1: CTUD;        CtudQu:BOOL;
    CtudCu :BOOL;       CtudQd:BOOL;
    CtudCd: BOOL;       CtudCv:UINT;
    CtudReset:BOOL;     CtudLoad:BOOL;
    CtuDPv:UINT;
END_VAR
```



ST

```
CTUD1(
    CU:= CtudCu ,       CD:= CtudCd  , RESET:=CtudReset
    LOAD:= CtudLoad ,   PV:= CtuDPv ,
    QU=>   CtudQu ,     QD=>CtudCd  ,CV=>CtudCv  );
```



CU	BOOL	Count UP, входной сигнал+	CV	UINT	Current counter value
CD	BOOL	Count DOWN, входной сигнал-	Счетчик изменяется по фронтам CD и CU, Отрицательного значения счетчика быть не может		
RESET	BOOL	Reset			
LOAD	BOOL	Загрузить значение PV			
PV	UINT	Предустановленное значение			
QU	BOOL	Счетчик достиг значения „PV“			
QD	BOOL	Счетчик достиг значения 0			