Библиотеки для программирования ПЛК «Tecomat» в соответствии с IEC 61 131-3

TXV 003 22.03
восьмой выпуск
март 2006г.
фирма оставляет за собой право проводить изменения

История изменений

Дата	Издание	Описание изменений
Август 2004г. - февраль 2006г.	1 - 7	Описание библиотек является составной частью документа TXV 003 21.01 Изменения см. TXV 003 21.01
Март 2006г.	8	Описание библиотек перемещено в отдельный документ TXV 003 xx.01

1 БИБЛИОТЕКИ

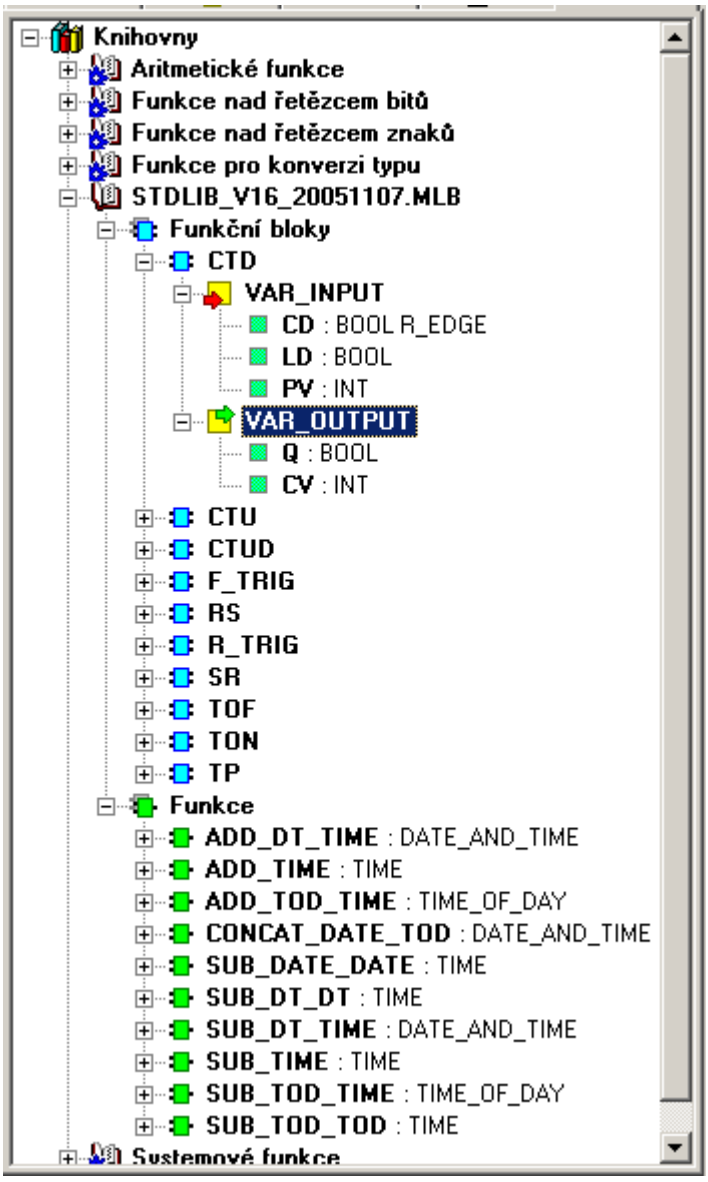
Библиотеки функций и функциональных блоков являются неделимой составной частью системы среды программирования «Mosaic». С точки зрения их постройки библиотеки можно разделить на следующие типы:

- встроенные (built-in) библиотеки
- стандартно поставляемые экстернные библиотеки
- библиотеки для определенных пользователей

Библиотека может содержать описание функций, функциональных блоков, типов данных и глобальных переменных.

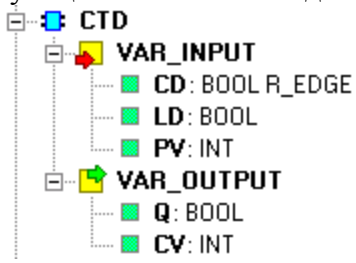
2 СТАНДАРТНАЯ БИБЛИОТЕКА STDLIB

Стандартная библиотека содержит функциональные блоки счетчиков **CTD**, **CTU** и **CTUD**, таймеров **TON**, **TOF** и **TP**, триггерных схем **RS** и **SR** и функциональные блоки детектирования граней **R_TRIG** и **F_TRIG**. Кроме того, содержит функции для работы с датой и временем. На следующем рисунке указана структура библиотеки StdLib в среде «Mosaic».



2.1 Функциональный блок счетчика вниз CTD

Функциональный блок для считывания вниз.



Входные переменные :

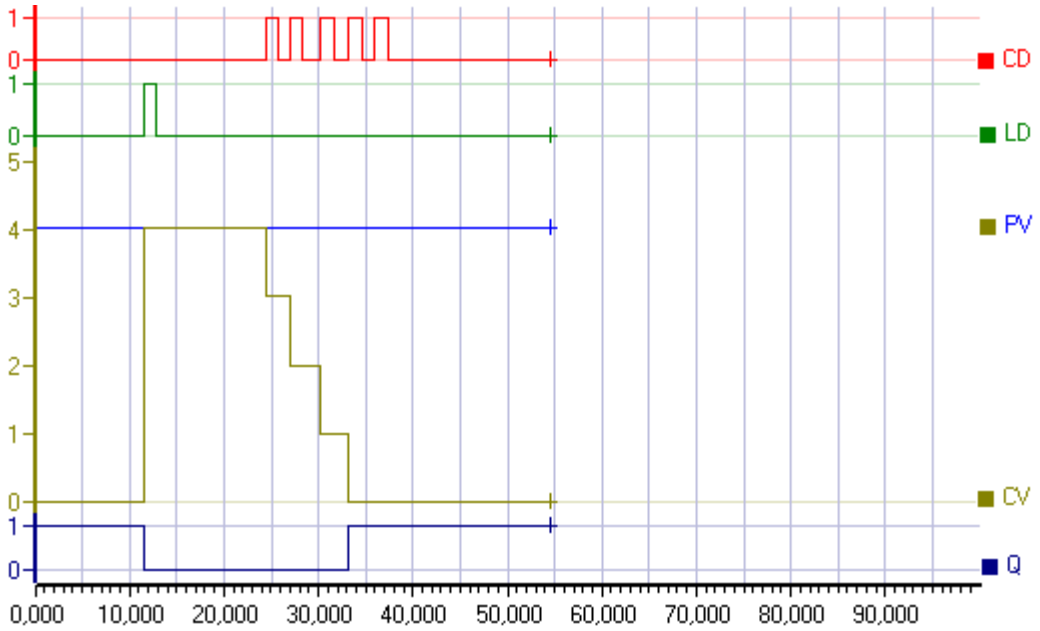
CD	вход для считывания вниз
LD	вход для настройки предварительного поиска
PV	предварительный поиск счетчика

Выходные переменные:

Q	выход счетчика
CV	показатель счетчика

Если входная переменная **LD** имеет значение **TRUE**, то значение предварительного поиска **PV** записывается в показатели счетчика **CV**. Еслиá входная переменная **LD** имеет значение **FALSE** и входная переменная **CD** перейдет от состояния **FALSE** в состояние **TRUE** (передний торец), показатель счетчика **CV** снижается на 1. Если показатель счетчика **CV** равняется нулю, выход счетчика **Q** устанавливается на величину **TRUE**. В противном случае имеет значение **FALSE**.

Поведение счетчика **CTD** объясняет следующий рисунок.

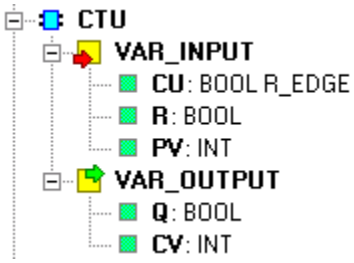


Пример программы с вызовом функционального блока **CTD** :

```
PROGRAM Counter
  VAR
    impulses          : BOOL;
    setCTD          : BOOL;
    counterCTD      : CTD;              // инстанция счетчика
    output          : BOOL;
  END_VAR

  counterCTD( CD := impulses, LD := setCTD, PV := 4, Q => output);
END_PROGRAM
```

2.2 Функциональный блок счетчика вверх CTU

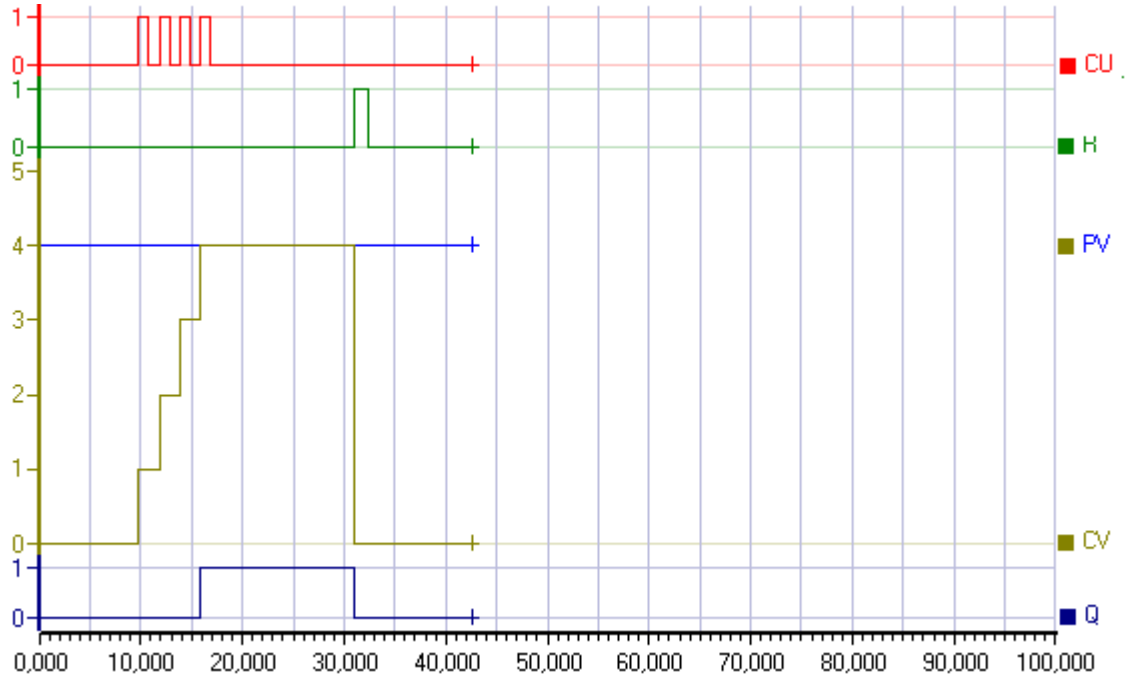
Функциональный блок для считывания вверх.



Входные переменные :
CU вход для считывания вверх
R повторный запуск счетчика
PV предварительный поиск счетчика
Выходные переменные:
Q выход счетчика
CV показатель счетчика

Если входная переменная **R** имеет значение **TRUE**, показатель счетчика **CV** приведен к нулю. Если входная переменная **R** имеет значение **FALSE** и входная переменная **CU** перейдет из состояния **FALSE** в состояние **TRUE** (передний торец), показатель счетчика **CV** на 1. Если показатель счетчика **CV** больше или равняется предварительному поиску **PV**, выход счетчика **Q** устанавливается на величину **TRUE**. В противном случае – на величину **FALSE**.

Поведение счетчика **CTU** объясняет следующий рисунок.

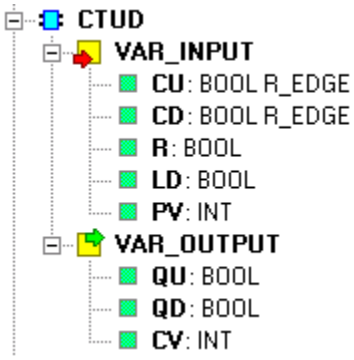


Пример программы с вызовом функционального блока **CTU** :

```
PROGRAM Counter
  VAR
    impulses       : BOOL;
    reset          : BOOL;
    counterCTU     : CTU;                  // инстанция счетчика
    output         : BOOL;
  END_VAR

  counterCTU( CU := impulses, R := reset, PV := 4, Q => output);
END_PROGRAM
```

2.3 Функциональный блок реверсивного счетчика CTUD

Функциональный блок для реверсивного считывания.



Входные переменные :

CU вход для считывания вверх

CD вход для считывания вниз

R повторный запуск счетчика

LD вход для настройки предварительного поиска

PV предварительный поиск счетчика

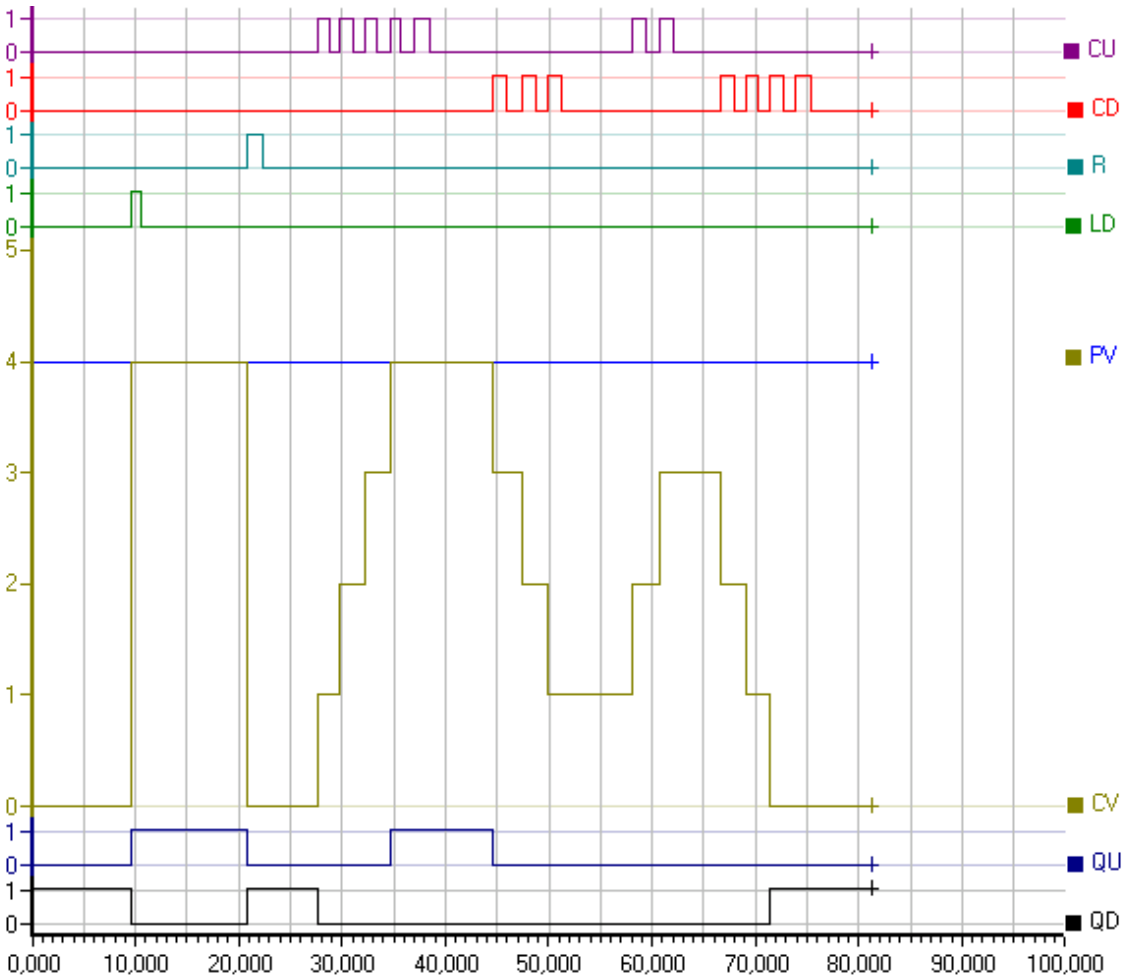
Выходные переменные:

QU выход счетчика вверх

QD выход счетчика вниз

CV показатель счетчика

Поведение счетчика **CTUD** объясняет следующий рисунок.



Если входная переменная **R** имеет величину **TRUE**, показатель счетчика **CV** будет приведен к нулю. Если входная переменная **LD** имеет значение **TRUE**, показатель счетчика **CV** будет приведен к нулю на значение предварительного поиска **PV**.

Если входная переменная **CU** перейдет из состояния **FALSE** в состояние **TRUE** (передний торец), показатель счетчика **CV** увеличится на 1, счетчик считывает вверх. То же самое в случае, если входная переменная **CD** перейдет из состояния **FALSE** в состояние **TRUE** (передний торец), показатель счетчика **CV** снижается на 1, счетчик считывает вниз.

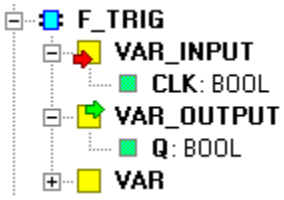
Если показатель счетчика **CV** больше или равняется предварительному поиску **PV**, выход счетчика **QU** установлен на значяение **TRUE**, в противном случае – имеет значение **FALSE**. Если показатель счетчика **CV** равен нулю, выход счетчика **QD** установлен на значение **TRUE**, в противном случае – имеет значение **FALSE**.

Пример программы с вызовом функционального блока **CTUD** :

```
PROGRAM Counter
  VAR
    impulseUP       : BOOL;
    impulseDOWN     : BOOL;
    resetCTUD     : BOOL;
    setCTUD       : BOOL;
    counterCTUD   : CTUD;                    // инстанция счетчика
    limitUP       : BOOL;
    limitDOWN     : BOOL;
  END_VAR

  counterCTUD( CU := impulseUP,   CD := impulseDOWN,
               R  :=  resetCTUD, LD := setCTUD,
               PV := 4,
               QU => limitUP, QD => limitDOWN );
END_PROGRAM
```

2.4 Функциональный блок F_TRIG

Функциональный блок для детектирования задней грани.

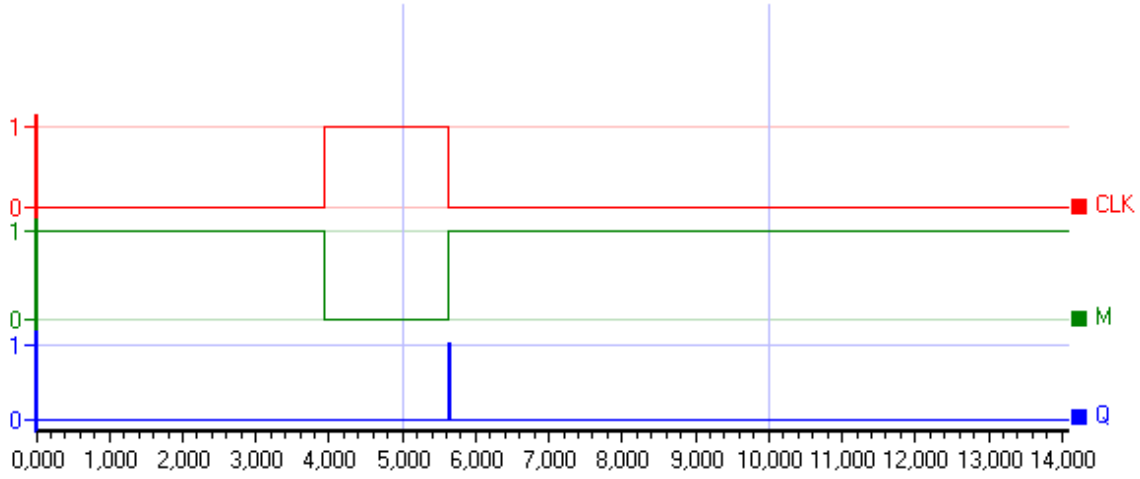


Входные переменные :
CLK

Выходные переменные:
Q

Поведение функционального блока **F_TRIG** отвечает следующей программе на языке ST.

```
function_block F_TRIG
//--------------------------------------
//  Falling Edge Detector
//
  var_input  CLK : BOOL;          end_var
  var_output Q   : BOOL;          end_var
  var        M   : BOOL := TRUE; end_var

  Q := not CLK and not M; M := not CLK;
end_function_block
```

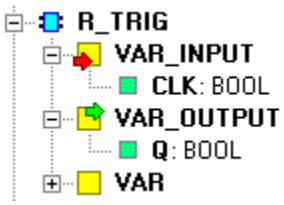


Пример программы с вызовом функционального блока **F_TRIG** :

```
PROGRAM EdgeDetect
  VAR
    input          : BOOL;
    inst_FTRIG     : F_TRIG;              // инстанция FB F_TRIG
    output         : BOOL;
  END_VAR

  inst_FTRIG( CLK := input, Q => output);
END_PROGRAM
```

2.5 Функциональный блок R_TRIG

Функциональный блокдля для детектирования переднего торца.

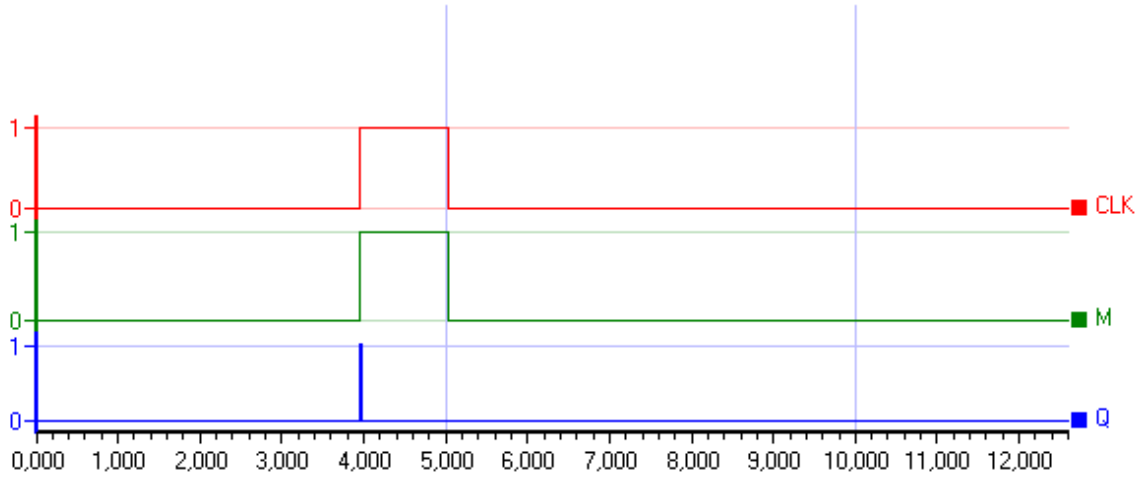


Входные переменные :
CLK

Выходные переменные:
Q

Поведение функционального блока **R_TRIG** отвечает следующей программе на языке ST.

```
function_block R_TRIG
//------------------------------
//  Rising Edge Detector
//
  var_input  CLK : BOOL; end_var
  var_output Q   : BOOL; end_var
  var        M   : BOOL; end_var

  Q := clk and not M; M := CLK;
end_function_block
```

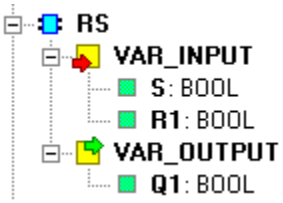


Пример программы с вызовом функционального блока R_TRIG :

```
PROGRAM EdgeDetect
  VAR
    input          : BOOL;
    inst_RTRIG     : R_TRIG;                 // инстанция FB R_TRIG
    output         : BOOL;
  END_VAR

  inst_RTRIG( CLK := input, Q => output);
END_PROGRAM
```

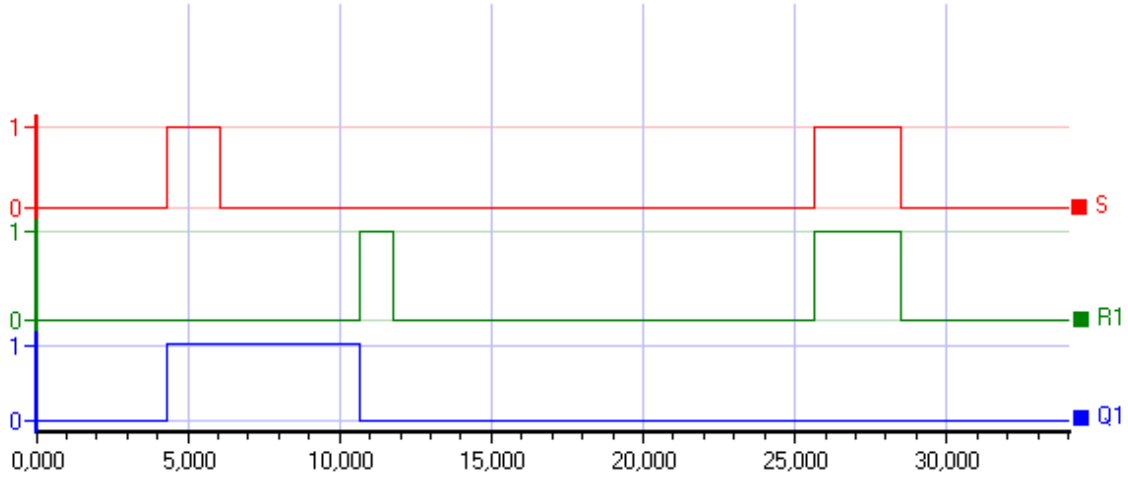
2.6 Функциональный блок RS

Функциональный блок для реализации двухпозиционной триггерной схемы с доминирующей функцией **RESET**.



Поведение функционального блока **RS** отвечает следующей программе на языке ST.

```
function_block RS
//-------------------------------------
//  Flip-Flop (RESET Dominant)
//
  var_input   S, R1      : BOOL; end_var
  var_output Q1          : BOOL; end_var

  Q1 := not R1 and (S or Q1);
end_function_block
```

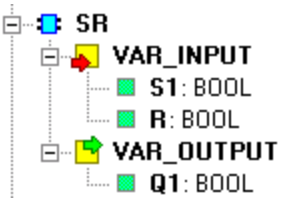


Пример программы с вызовом функционального блока **RS** :

```
PROGRAM BistableBlock
  VAR
    inputSET       : BOOL;
    inputRES       : BOOL;
    inst_RS        : RS;
    output         : BOOL;
  END_VAR

  inst_RS( S := inputSET, R1 := inputRES, Q1 => output);
END_PROGRAM
```

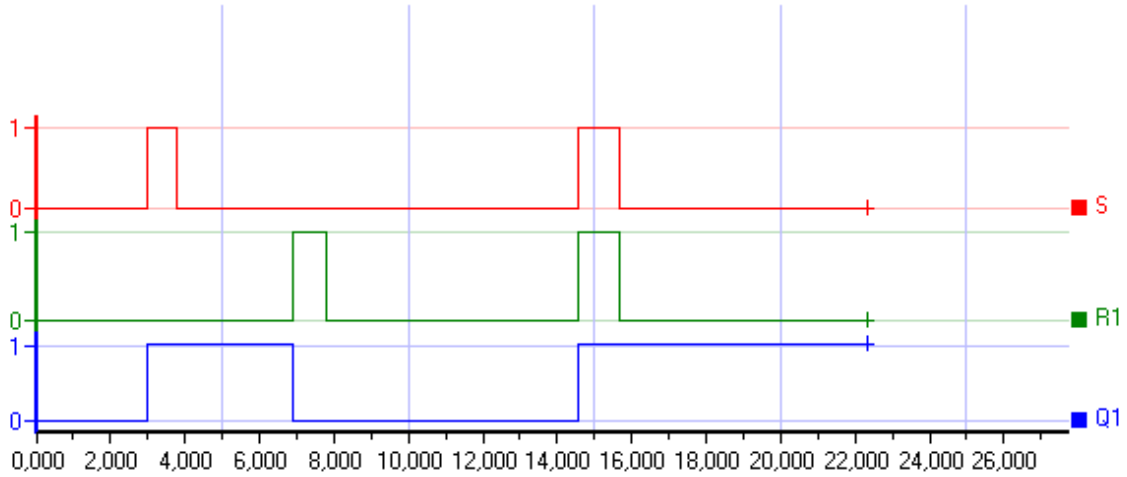
2.7 Функциональный блок SR

Функциональный блок для реализации двухпозиционной триггерной схемы с доминирующей функцией **SET**.



Поведение функционального блока **SR** отвечает следующей программе на языке **ST**.

```
function_block SR
//--------------------------------------
//  Flip-Flop (Set Dominant)
//
  var_input  S1, R       : BOOL; end_var
  var_output Q1          : BOOL; end_var

  Q1 := S1 or (not R and Q1);
end_function_block
```

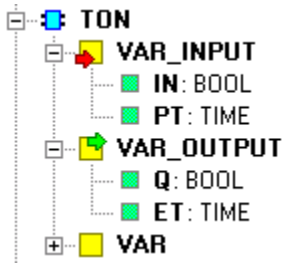


Пример программы с вызовом функционального блока **SR** :

```
      PROGRAM BistableBlock
  VAR
    inputSET       : BOOL;
    inputRES       : BOOL;
    inst_SR        : SR;
    output         : BOOL;
  END_VAR

  inst_SR( S1 := inputSET, R := input ПОВТОРНЫЙ ЗАПУСК, Q1 => output);
END_PROGRAM
```

2.8 Функциональный блок таймера TON

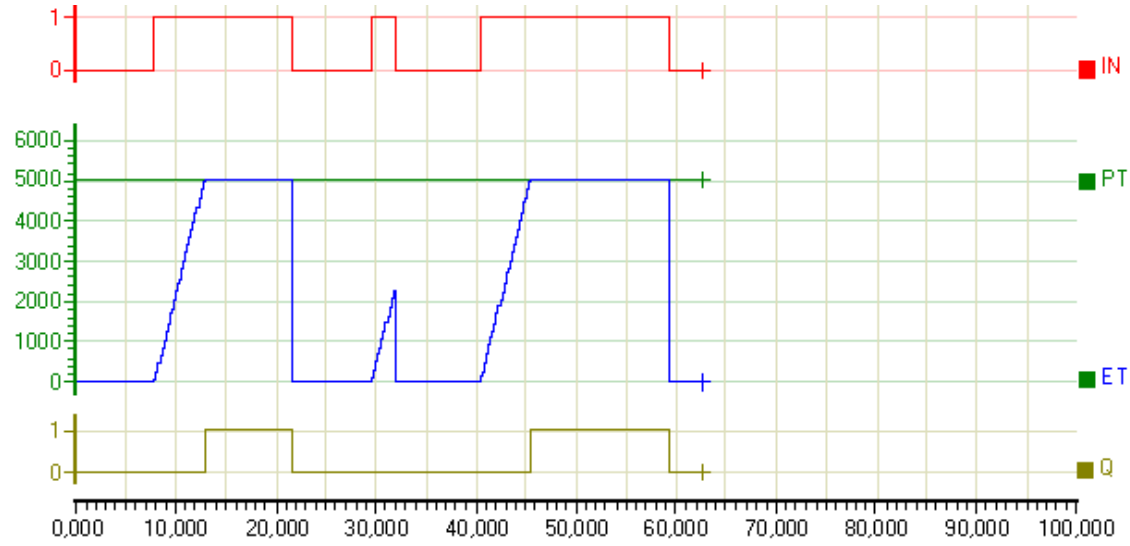
Функциональный блок **TON** (Timer On Delay) реализует выдержку на передний торец, т.е. реле с запаздывающим притяжением.



Входные переменные :
IN вход таймера
PT предварительный поиск таймера

Выходные переменные:
Q выход таймера
ET актуальное значение таймера

Если вход **IN** - **FALSE**, выход **Q** - **FALSE** и **ET** имеет значение **0**. Как только вход **IN** перейдет в состояние **TRUE**, актуальное значение таймера **ET** начнет увеличиваться и в момент достижения предварительного поиска **PT** выход **Q** установится на значение **TRUE**. Актуальное значение таймера потом не увеличивается. Поведение таймера **TON** объясняет следующий рисунок.

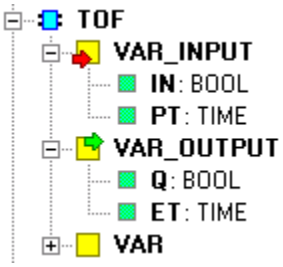


Пример программы с вызовом функционального блока **TON** :

```
PROGRAM Timer
  VAR
    start          : BOOL;
    timerTON       : TON;
    output         : BOOL;
  END_VAR

  timerTON( IN := start, PT :=T#5s, Q => output );
END_PROGRAM
```

2.9 Функциональный блок таймера TOF

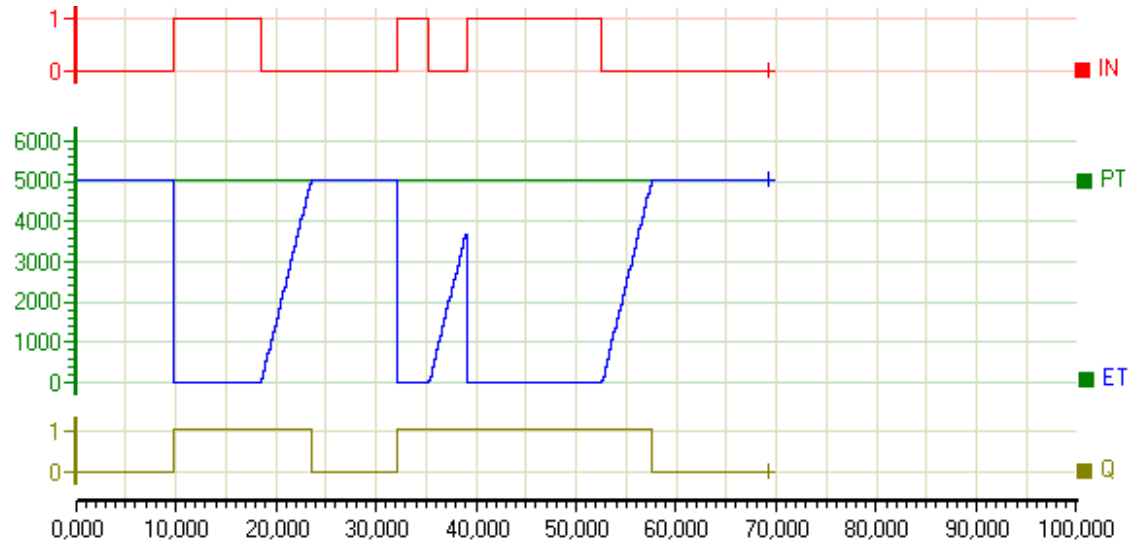
Функциональный блок **TOF** (Timer Off Delay) реализует выдержку на заднюю грань, т.е. реле с запаздывающим одтяжением.



Входные переменные :
IN вход таймера
PT предварительный поиск таймера

Выходные переменные:
Q выход таймера
ET актуальное значение таймера

Как только вход **IN** перейдет в состояние **FALSE**, актуальное значение таймера **ET** начнет увеличиваться и в момент достижения предварительного поиска **PT** выход **Q** установится на значение **TRUE**. Актуальное значение таймера потом не увеличивается. Выход **Q** находится в состоянии **FALSE** всегда, когда вход **IN FALSE** и одновременно актуальное значение **ET** равняется предварительному поиску **PT**. Поведение таймера **TOF** описано на следующем рисунке.

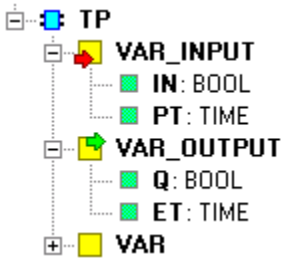


Пример программы с вызовом функционального блока **TOF** :

```
PROGRAM Timer
  VAR
    start          : BOOL;
    timerTOF       : TOF;
    output         : BOOL;
  END_VAR

  timerTOF( IN := start, PT :=T#5s, Q => output );
END_PROGRAM
```

2.10 Функциональный блок таймера TP

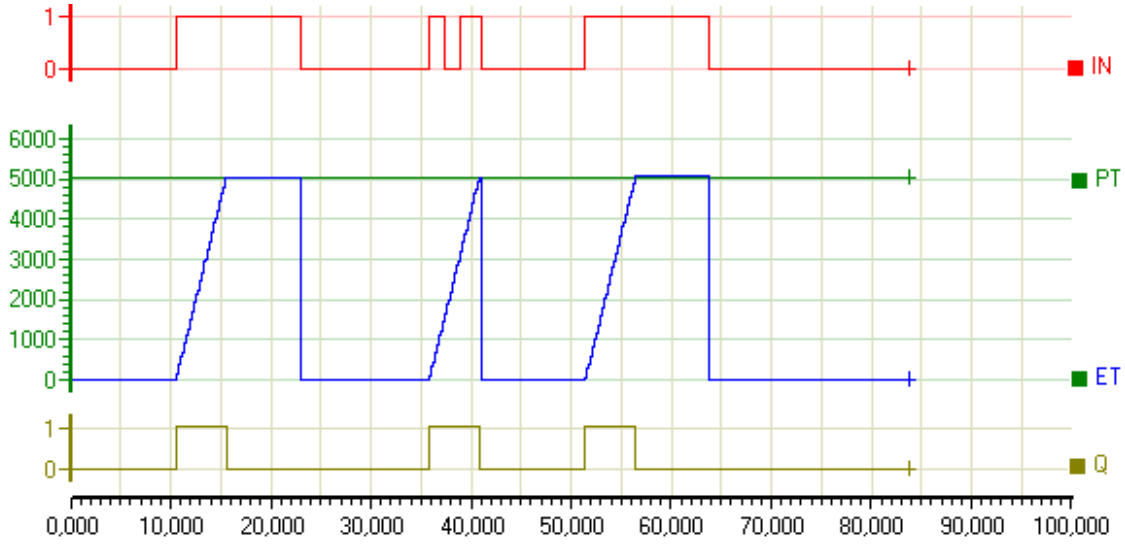
Функциональный блок **TP** (Timer Pulse) генерирует импульс данной ширины на передний торец.



Входные переменные :
IN вход таймера
PT предварительный поиск таймера

Выходные переменные:
Q выход таймера
ET актуальное значение таймера

Как только вход **IN** перейдет в состояние **TRUE**, актуальное значение таймера **ET** начнет увеличиваться до момента достижения предварительного поиска **PT**. Актуальное значение таймера потом повышается. Выход **Q** настроен на значение **TRUE** ели была детектирован передний торец на входе **IN** и одновременно актуальное значение **ET** меньше чем предварительный поиск **PT**. В противном случае выход **Q** имеет значение **FALSE**. Поведение таймера **TP** указано на следующем рисунке



Пример программы с вызовом функционального блока **TP** :

```
PROGRAM Timer
  VAR
    start          : BOOL;
    timerTP        : TP;
    output         : BOOL;
  END_VAR

  timerTP( IN := start, PT :=T#5s, Q => output );
END_PROGRAM
```

2.11 Функция ADD_TIME

Функция **ADD_TIME** сосчитает две входные переменные типа **TIME**. Величина возврата функции имеет тип **TIME**.

```
PROGRAM Example_ADD_TIME
  VAR
    time1, time2, time3  : TIME;
  END_VAR

  time1 := TIME#11h12m13s;
  time2 := ADD_TIME( IN1 := time1, IN2 := T#1h);  // T#12h12m13.0s
  time3 := ADD_TIME( T#10m, time2);               // T#12h22m13.0s
END_PROGRAM
```

2.12 Функция ADD_TOD_TIME

Функция **ADD_TOD_TIME** сосчитает входную переменную типа **TIME_OF_DAY** с переменной типа **TIME.** Величина возврата функция имеет тип **TIME_OF_DAY**.

```
PROGRAM Example_ADD_TOD_TIME
  VAR
    time1   : TIME_OF_DAY := TOD#11:38:52.35;
    time2   : TIME        := T#15:22:11.120;
    time3   : TIME_OF_DAY;
  END_VAR

  time3 := ADD_TOD_TIME( IN1 := time1, IN2 := time2);   // 27:01:03.155
END_PROGRAM
```

2.13 Функция ADD_DT_TIME

Функция **ADD_DT_TIME** сосчитает входную переменную типа **DATE_AND_TIME** с переменной типа **TIME.** Величина возврата функция имеет тип **DATE_AND_TIME**.

```
PROGRAM Example_ADD_DT_TIME
  VAR
    varDT       : DATE_AND_TIME := DT#2004-05-30-00:00:00;
    varTIME     : TIME          := TIME#12:55:02.0;
    sum         : DATE_AND_TIME;
  END_VAR

  sum := ADD_DT_TIME(IN1 := varDT, IN2 := varTIME);
                                                  //DT#2004-05-30-12:55:02
END_PROGRAM
```

2.14 Функция SUB_TIME

Функция **SUB_TIME** вычтет две входные переменные типа **TIME**. Величина возврата функция имеет тип **TIME**.

```
PROGRAM Example_SUB_TIME
  VAR
   time1, time2, time3   : TIME;
  END_VAR

  time1:= TIME#11h12m13s;
  time2 := SUB_TIME( IN1 := time1, IN2 := T#1h);  // T#10h12m13.0s
  time3 := SUB_TIME( time2, T#10m);               // T#10h02m13.0s
END_PROGRAM
```

2.15 Функция SUB_DATE_DATE

Функция **SUB_DATE_DATE** вычтет две входные переменные типа **DATE**. Величина возврата функция имеет тип **TIME**.

```
PROGRAM Example_SUB_DATE_DATE
  VAR
    varDate1      : DATE := D#2005-11-03;
    varDate2      : DATE := D#2005-10-21;
    varTime1      : TIME;
    varTime2      : TIME;
    varTime3      : TIME;
    maxTime       : TIME := T#24d20h31m23.647s;
    timeOverFlow  : BOOL;
    timeUnderFlow : BOOL;
  END_VAR

  varTime1 := SUB_DATE_DATE( IN1 := varDate1, IN2 := varDate2);
              // T#13d00h00m00s

  varTime2 := SUB_DATE_DATE( D#2005-11-03, D#2005-05-21);
              // T#24d20h31m23.647s

  IF varTime2 = maxTime THEN
    timeOverFlow := TRUE;
  ELSE
    timeOverFlow := FALSE;
  END_IF;

  varTime3 := SUB_DATE_DATE(IN1 := varDate2, IN2 := D#2005-10-22);
  IF varTime3 < T#0s THEN
    timeUnderFlow := TRUE;
  ELSE
    timeUnderFlow := FALSE;
  END_IF;

END_PROGRAM
```

2.16 Функция SUB_TOD_TIME

Функция **SUB_TOD_TIME** вычтет входную переменнуютипа **TIME** от переменной типа **TIME_OF_DAY.** Величина возврата функция имеет тип **TIME_OF_DAY**.

```
PROGRAM Example_SUB_TOD_TIME
  VAR
    varTOD      : TIME_OF_DAY := TOD#19:22:33;
    varTIME     : TIME        := TIME#12:00:00;
    result1     : TIME_OF_DAY;
  END_VAR

  IF varTOD >= TIME_TO_TIME_OF_DAY( varTIME) THEN
    result1 := SUB_TOD_TIME(IN1 := varTOD, IN2 := varTIME); //TOD#07:22:33
  ELSE
    result1 := TOD#00:00:00;
  END_IF;

END_PROGRAM
```

2.17 Функция SUB_TOD_TOD

Функция **SUB_TOD_TOD** вычтет входную переменнуютипа **TIME_OF_DAY** от переменной типа **TIME_OF_DAY.** Величина возврата функция имеет тип **TIME**.

```
PROGRAM Example_SUB_TOD_TOD
  VAR
    varTOD1     : TIME_OF_DAY := TOD#19:22:33;
    varTOD2     : TIME_OF_DAY := TOD#10:12:13;
    varTime     : TIME;
  END_VAR

  varTime := SUB_TOD_TOD( varTOD1, varTOD2); // T#09h10m20.0s
END_PROGRAM
```

2.18 Функция SUB_DT_TIME

Функция **SUB_DT_TIME** вычтет входную переменную типа **TIME** от переменной типа **DATE_AND_TIME.** Величина возврата функция имеет тип **DATE_AND_TIME**.

```
PROGRAM Example_SUB_DT_TIME

  VAR
    varDateTime : DT   := DT#2005-12-05-06:00:00.0;
    varTime     : TIME := T#12h;
    result      : DATE_AND_TIME;
    i : INT := 20;
  END_VAR

  result := SUB_DT_TIME(IN1 := varDateTime, IN2 := varTime);
            // DT#2005-12-04-18:00:00.0
END_PROGRAM
```

2.19 Функция SUB_DT_DT

Функция **SUB_DT_DT** вычтет входную переменную типа **DATE_AND_TIME** от переменной типа **DATE_AND_TIME.** Величина возврата функция имеет тип **TIME**.

```
PROGRAM Example_SUB_DT_DT
  VAR
    varDT1      : DT := DT#2005-06-12-13:00:30.0;
    varDT2      : DT := DT#2005-06-11-12:00:15.0;
    varTIME     : TIME;
  END_VAR

  varTIME := SUB_DT_DT(IN1 := varDT1, IN2 := varDT2);
             // T#1d01h00m15.0s
END_PROGRAM
```

2.20 Функция CONCAT_DATE_TOD

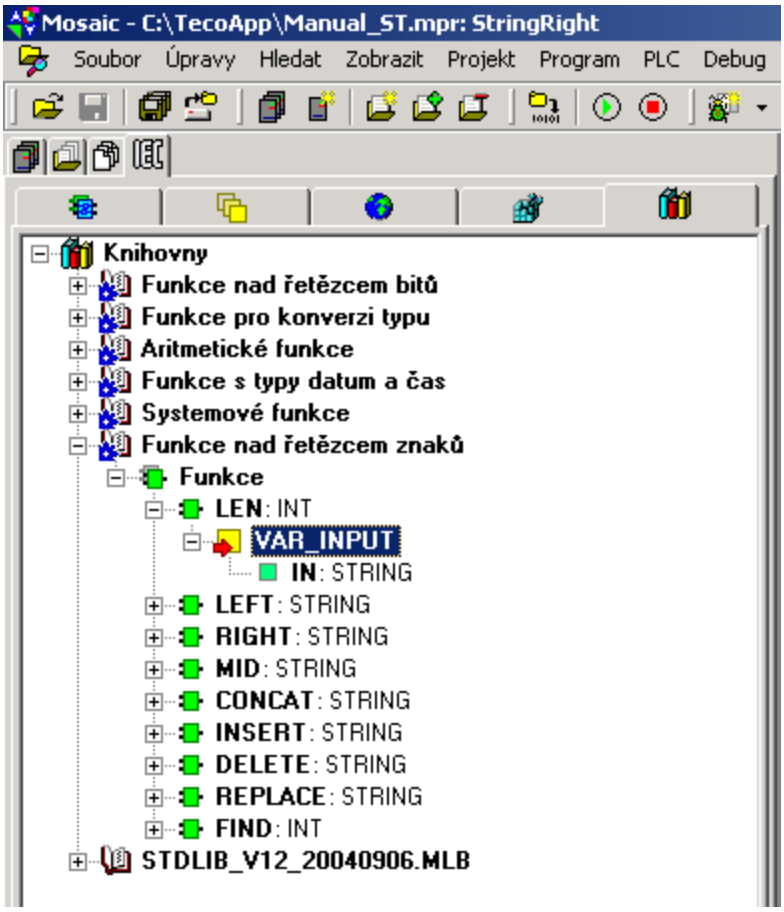
Функция **CONCAT_DATE_TOD** сосчитает входную переменную типа **DATE** с переменной типа **TIME_OF_DAY.** Величина возврата функция имеет тип **DATE_AND_TIME**.

```
PROGRAM Example_CONCAT_DATE_TOD
  VAR
    varDate : DATE        := D#2005-02-28;
    varTOD  : TIME_OF_DAY := TOD#08:20:40.0;
    varDT   : DATE_AND_TIME;
  END_VAR

  varDT := CONCAT_DATE_TOD( varDate, varTOD);  // DT#2005-02-28-08:20:40.0
END_PROGRAM
```

3 ФУНКЦИИ НАД ПОСЛЕДОВАТЕЛЬНОСТЬЮ СИМВОЛОВ

Данная библиотека содержит функции для работы с последовательностью символов (тип данных **STRING**). Содержит функцию для определения длины последовательности **LEN**, функция для выбора части последовательности **LEFT**, **RIGHT** а **MID**, функция для соединения последовательностей **CONCAT**, функция для вставления последовательности в последовательность **INSERT**, функция для сброса части последовательности **DELETE**, функция для замена части последовательности другой последовательносттью **REPLACE** и наконец функция для обнаружения позиции последовательности в другой последовательности **FIND**. На следующем рисунке указано включение функций над последовательностью символов в библиотеку в среде «Mosaic».

Внедрение типа данных **STRING** в системах «Tecomat» отвечает последовательностям на языке C.



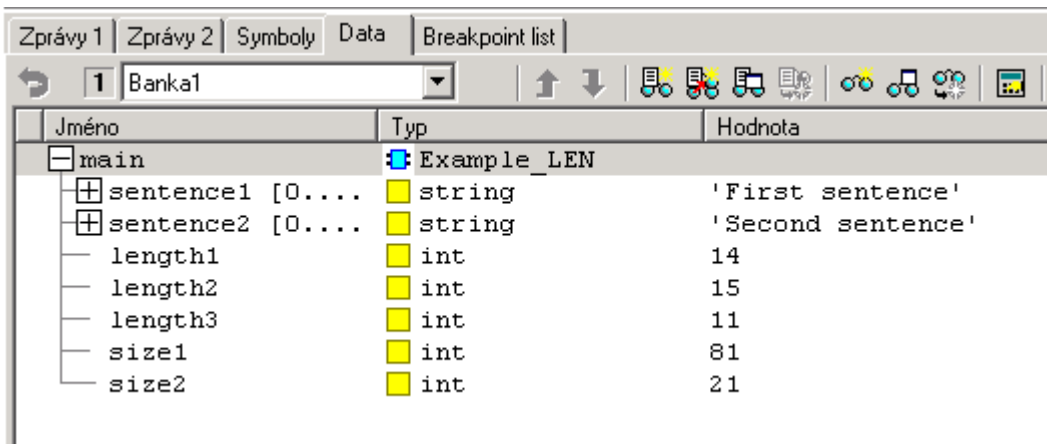
3.1 Функция LEN

Функция вернет длину входной последовательности **IN**. Длина последовательности отвечает актуальному количеству знаков в последовательности. Величина возврата имеет тип **INT**.

Входные переменные :
IN последовательность знаков
Величина возврата: длина последовательности

Использование функции **LEN** показано на следующем примере. Что касается длины последовательности надо осознать, что актуальная длина последовательности в большинстве случаев не отвечает величине переменной, в которой последовательность уложена. Величина переменной типа **STRING** определяется описанием переменной и ее можно определить с помощью функции **SIZEOF** в то время как длина последовательности в переменной определена количеством ASCII знаков в последовательности и определяется с помощью функции **LEN**.

```
PROGRAM Example_LEN
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := 'Second sentence';
    length1,
    length2,
    length3 : INT;
    size1,
    size2   : INT;
  END_VAR

  // length of string
  length1 := LEN( sentence1);
  length2 := LEN( IN := sentence2);
  length3 := LEN( 'Hello world');

  // size of variable
  size1 := sizeof( sentence1);
  size2 := sizeof( sentence2);

END_PROGRAM
```



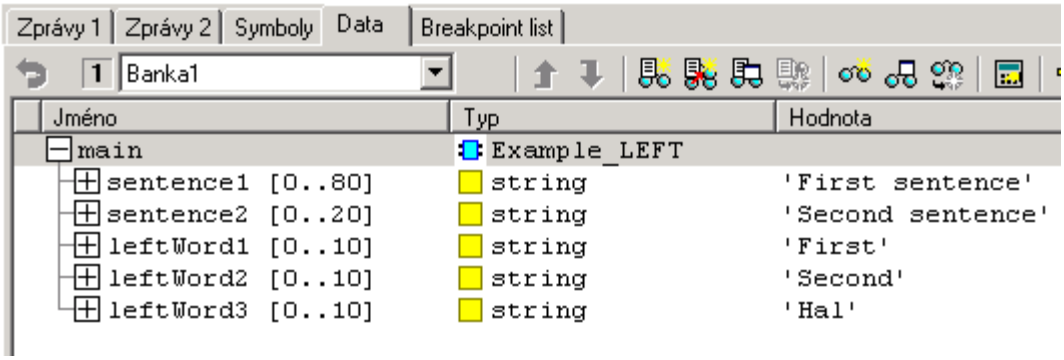
3.2 Функция LEFT

Функция **LEFT** вернет **L** знаков из входной последовательности **IN**. Знаки возвращаются слева, т.е. от начала входной последовательности.

Входные переменные :
IN входная последовательность
L количество знаков
Величина возврата : последовательность

Использование функции **LEFT** показано на следующем примере.

```
PROGRAM Example_LEFT
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := 'Second sentence';
    leftWord1,
    leftWord2,
    leftWord3     : STRING[10];
  END_VAR

  leftWord1 := LEFT( sentence1, 5);
  leftWord2 := LEFT( IN := sentence2, L := 6);
  leftWord3 := LEFT( 'Hallo world', 3);

END_PROGRAM
```



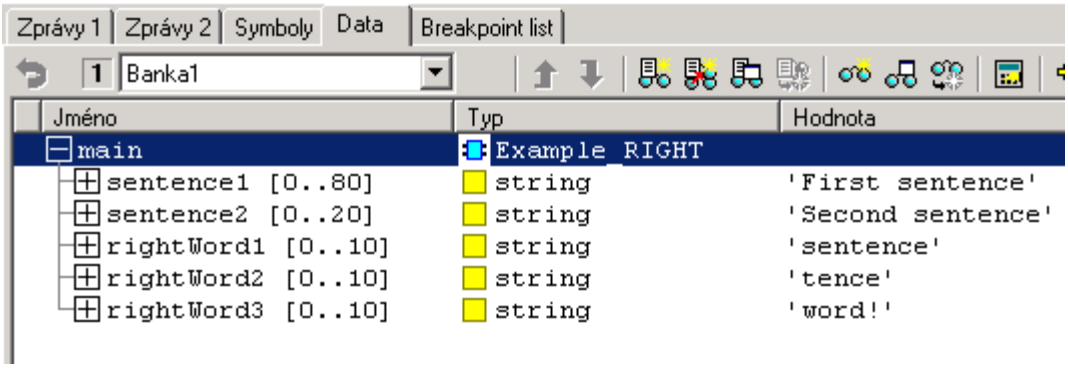
3.3 Функция RIGHT

Функция **RIGHT** вернет **L** знаков ze входной последовательности **IN**. Знаки возвращаютсяzprava, т.е. od konce входной последовательности.

Входные переменные :
IN входная последовательность
L количество знаков
Величина возврата : последовательность

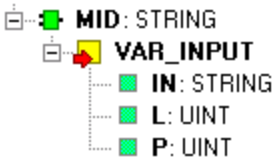
Использование функция **RIGHT** указано на следующем примере.

```
PROGRAM Example_RIGHT
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := 'Second sentence';
    rightWord1,
    rightWord2,
    rightWord3    : STRING[10];
  END_VAR

  rightWord1 := RIGHT( sentence1, 8);
  rightWord2 := RIGHT( IN := sentence2, L := 5);
  rightWord3 := RIGHT( 'Hello word!', 5);

END_PROGRAM
```



3.4 Функция MID

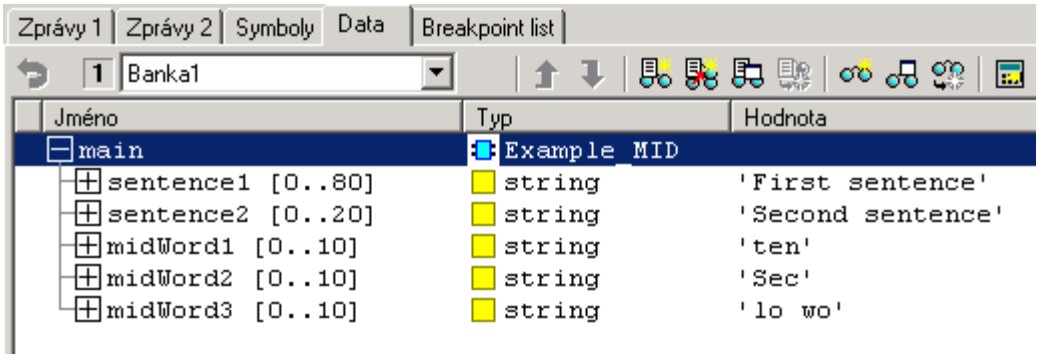
Функция **MID** вернет **L** знаков из входной последовательности **IN**. Знаки возвращаются от позиции **P** в входной последовательности. Первый знак в последовательности имеет позицию 1.



Входные переменные :
IN последовательность знаков
L количество знаков
P позиция во входной последовательности
Величина возврата : последовательность

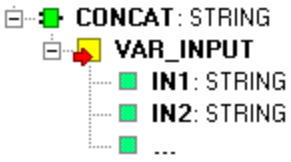
Использование функции **MID** показано на следующем примере.

```
PROGRAM Example_MID
  VAR
    sentence1     : STRING := 'First sentence';
    sentence2     : STRING[20];
    midWord1,
    midWord2,
    midWord3      : STRING[10];
  END_VAR

  midWord1  := mid( sentence1, 3, 10);
  sentence2 := 'Second sentence';
  midWord2  := mid( IN := sentence2, L := 3, P := 1);
  midWord3  := mid( 'Hello world', 5, 4);

END_PROGRAM
```



3.5 Функция CONCAT

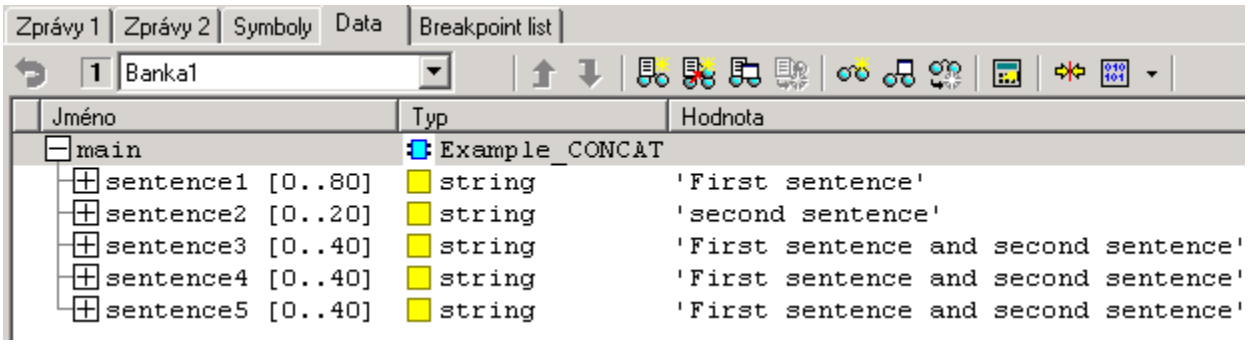
Функция **CONCAT** соединит несколько последовательностей во входную последовательность.



Входные переменные :
IN1 последовательность знаков
IN2 последовательность знаков
Величина возврата : последовательность

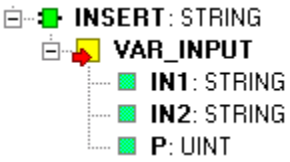
Использование функции **CONCAT** указано на следующем примере. Вызов функции можно проводить классически через наименование функции или с помощью перенагруженного оператора "**+**". Функция **CONCAT** расширяема, поэтому она может иметь различное количество входных последовательностей.

```
PROGRAM Example_CONCAT
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := 'second sentence';
    sentence3,
    sentence4,
    sentence5     : STRING[40];
  END_VAR

  sentence3 := CONCAT( sentence1, ' and ', sentence2);
  sentence4 := CONCAT( IN1 := sentence1, IN2 := ' and ', IN3 := sentence2);
  sentence5 := sentence1 + ' and ' + sentence2;
END_PROGRAM
```



3.6 Функция INSERT

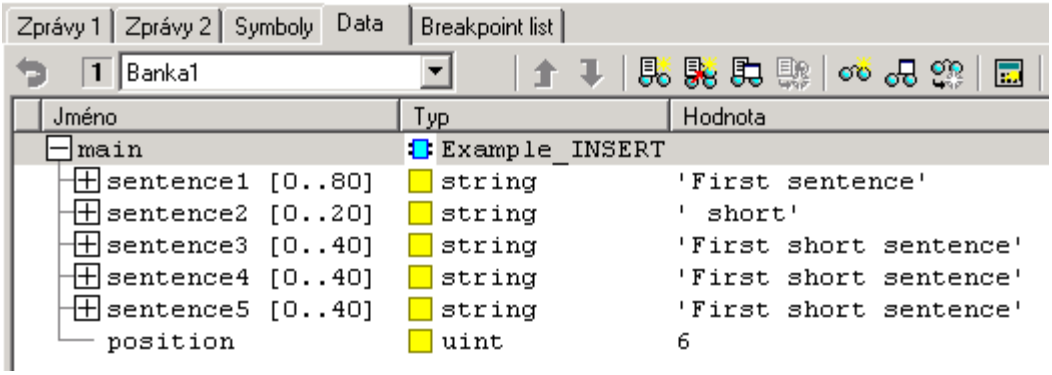
Функция **INSERT** запишет последовательность **IN2** в последовательность **IN1** на позицию **P**. Созданная таким образом последовательность возвращается в качебстве величины возврата функции.



Входные переменные :
IN1 входная последовательность
IN2 записываемая последовательность
P позиция во входной последовательности
Величина возврата : последовательность

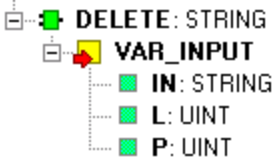
Использование функции **INSERT** указано на следующем примере.

```
PROGRAM Example_INSERT
  VAR
    sentence1     : STRING     := 'First sentence';
    sentence2     : STRING[20] := ' short';
    sentence3,
    sentence4,
    sentence5     : STRING[40];
    position      : UINT := 6;
  END_VAR

  sentence3 := INSERT( sentence1, ' short', 6);
  sentence4 := INSERT( IN1 := sentence1, IN2 := ' short', P := 6);
  sentence5 := INSERT( sentence1, sentence2, position);
END_PROGRAM
```



3.7 Функция DELETE

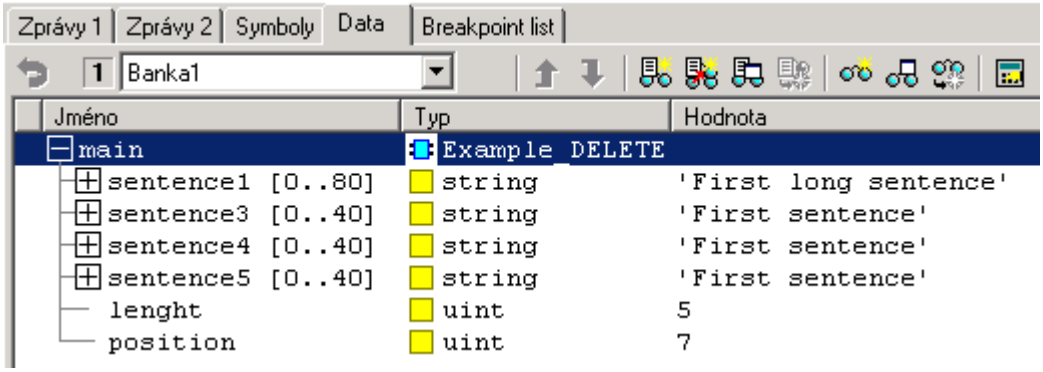
Функция **DELETE** исключит количество знаков **L** из входной последовательности **IN**. Знаки исключены из позиции **P**. Созданная таким образом последовательность возвращается в качестве величины возврата функции.



Входные переменные :
IN входная последовательность
L количество исключенных знаков
P позиция, из которой знаки исключены
Величина возврата : последовательность

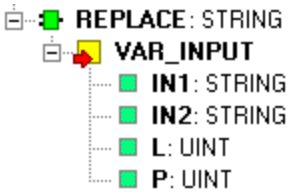
Использование функции **DELETE** указано на следующем примере.

```
PROGRAM Example_DELETE
  VAR
    sentence1     : STRING     := 'First long sentence';
    sentence3,
    sentence4,
    sentence5     : STRING[40];
    lenght        : UINT := 5;
    position      : UINT := 7;
  END_VAR

  sentence3 := DELETE( sentence1, 5, 7);
  sentence4 := DELETE( IN := sentence1, L := lenght, P := 7);
  sentence5 := DELETE( sentence1, lenght, position);
END_PROGRAM
```



3.8 Функция REPLACE

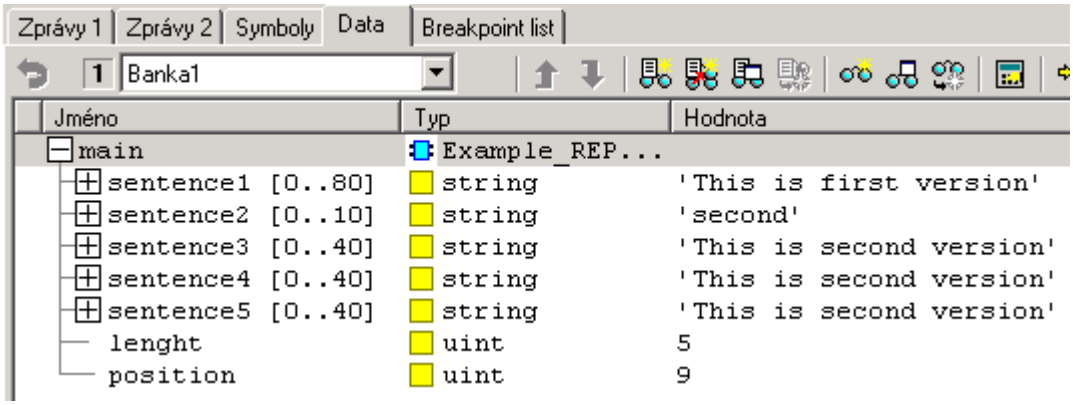
Функция **REPLACE** сначала исключит **L** знаков из входной последовательности **IN1**. Знаки исключены от позиции **P**После этого на ту же позицию **P** запишет последовательность **IN2**. Созданная таким образом последовательность возвращается в качестве величины возврата функции.



Входные переменные :
IN1 входная последовательность
IN2 записываемая последовательность
L количество исключенных знаков
P позиция во входной последовательности
Величина возврата : последовательность

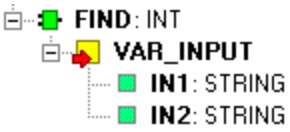
Использование функции **REPLACE** указано на следующем примере.

```
PROGRAM Example_REPLACE
  VAR
    sentence1     : STRING     := 'This is first version';
    sentence2     : STRING[10] := 'second';
    sentence3,
    sentence4,
    sentence5     : STRING[40];
    lenght        : UINT := 5;
    position      : UINT := 9;
  END_VAR

  sentence3 := REPLACE( sentence1, 'second', 5, 9);
  sentence4 := REPLACE( IN1 := sentence1, IN2 := sentence2, L := 5, P := 9);
  sentence5 := REPLACE( sentence1, sentence2, lenght, position);
END_PROGRAM
```



3.9 Функция FIND

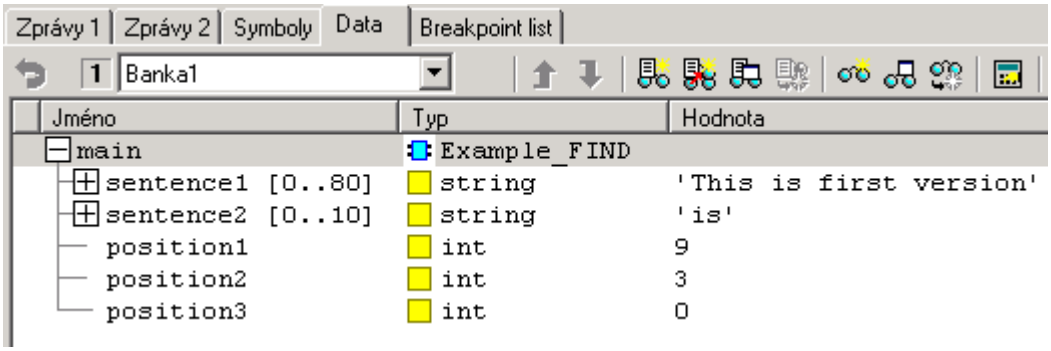
Функция **FIND** вернет позицию последовательности **IN2** во входной последовательности **IN1**. Если последовательность **IN2** не имеется во входной последовательности **IN1**, функция вернется к **0**. Если последовательность **IN2** имеется в последовательности **IN1** многократно, функция **FIND** вернет позицию первого местонахождения. При поиске принимается во внимание большие и маленькие буквы. Величина возврата функции имеет тип **INT**.



Входные переменные :
IN1 входная последовательность
IN2 искомая последовательность
Величина возврата : позиция IN2 в последовательности IN1

Использование функция **FIND** указана на следующем примере.

```
PROGRAM Example_FIND
  VAR
    sentence1     : STRING     := 'This is first version';
    sentence2     : STRING[10] := 'is';
    position1,
    position2,
    position3     : INT;
  END_VAR

  position1 := FIND( sentence1, 'first');
  position2 := FIND( IN1 := sentence1, IN2 := sentence2);
  position3 := FIND( sentence1, 'First');
END_PROGRAM
```

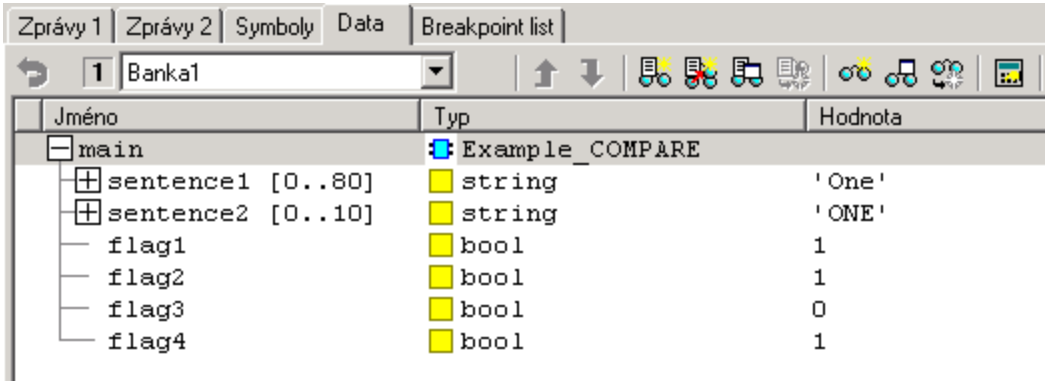


3.10 Функции сопоставления последовательностей

Стандартные функции для сопоставления (больше, меньше, больше или равняется, ...) перегружены также для типа **STRING**, поэтому последовательности можно между собой сравнивать. Последовательности сравниваются знак по знаку, большие и маленькие буквы отличаются между собой. с точки зрения сопоставления маленькие буквы имею “большую величину”, так как код ASCII маленьких букв больше чем больших букв.

Использование функций для сопоставления последовательностей указано на следующем примере.

```
PROGRAM Example_COMPARE
  VAR
    sentence1     : STRING     := 'One';
    sentence2     : STRING[10] := 'ONE';
    flag1,
    flag2,
    flag3,
    flag4   : BOOL;
  END_VAR

  flag1 := sentence1 = 'One';
  flag2 := sentence1 <> sentence2;
  flag3 := sentence1 = sentence2;
  flag4 := sentence1 > sentence2;
END_PROGRAM
```



4 ФУНКЦИИ ДЛЯ ПРЕОБРАЗОВАНИЯ ТИПА

Эти функции предназначены для преобразования величин между элементарными типами данных.

4.1 Функции ANY_TO_BOOL

Преобразование переменной произвольного элементарного типа данных на тип **BOOL**.

К данной группе функций принадлежит следующие функции:
SINT_TO_BOOL, INT_TO_BOOL, DINT_TO_BOOL
USINT_TO_BOOL, UINT_TO_BOOL, UDINT_TO_BOOL
REAL_TO_BOOL, LREAL_TO_BOOL
STRING_TO_BOOL
TIME_TO_BOOL, TOD_TO_BOOL, DATE_TO_BOOL, DT_TO_BOOL

Результатом преобразования **..._TO_BOOL** является значение **TRUE**, если входом функции является ненулевое значение. Если входом значение **0**, результатом преобразования является значение **FALSE**.

В случае функции **STRING_TO_BOOL** результатом является значение **TRUE**, если входом последовательности “**true**” (величина букв не имеет значения). В остальных случаях результатом преобразования являляется значение **FALSE**.

Использование преобразования **..._TO_BOOL** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_BOOL
  VAR_OUTPUT
    varBool     : ARRAY[1..20] OF BOOL;
  END_VAR

  varBool[1]  :=   SINT_TO_BOOL( 2);                  // 1
  varBool[2]  :=    INT_TO_BOOL( 0);                  // 0
  varBool[3]  :=   DINT_TO_BOOL( 1_235_678);          // 1
  varBool[4]  :=  USINT_TO_BOOL( 255);                // 1
  varBool[5]  :=   UINT_TO_BOOL( UINT#16#FFFF);       // 1
  varBool[6]  :=  UDINT_TO_BOOL( 0);                  // 0
  varBool[7]  :=   REAL_TO_BOOL( -12.6);              // 1
  varBool[8]  :=  LREAL_TO_BOOL( 123.4567);           // 1
  varBool[9]  :=   TIME_TO_BOOL( T#0:00:00.00);       // 0
  varBool[10] :=   BYTE_TO_BOOL( BYTE#16#AF);         // 1
  varBool[11] :=   WORD_TO_BOOL( 1122);               // 1
  varBool[12] :=  DWORD_TO_BOOL( 12345678);           // 1
  varBool[13] := STRING_TO_BOOL( 'FALSE');            // 0
  varBool[14] := STRING_TO_BOOL( 'True');             // 1
  varBool[15] := STRING_TO_BOOL( '0');                // 0
  varBool[16] := STRING_TO_BOOL( '1');                // 1
  varBool[17] := STRING_TO_BOOL( 'Anything');         // 0

END_FUNCTION_BLOCK
```

4.2 Функции ANY_TO_SINT

Преобразование переменной произвольного элементарного типа данных на тип **SINT**.

К данной группе функций принадлежит следующая функция:
BOOL_TO_SINT
INT_TO_SINT, DINT_TO_SINT
USINT_TO_SINT, UINT_TO_SINT, UDINT_TO_SINT
REAL_TO_SINT, LREAL_TO_SINT
STRING_TO_SINT
TIME_TO_SINT, TOD_TO_SINT, DATE_TO_SINT, DT_TO_SINT

Результатом преобразования **..._TO_SINT** является цифра в пределах <-128, +127>. При конвертировании от большего типа данных к меньшему (напр. **DINT_TO_SINT**) рискуем потерей информации в случае, если входное значение находится вне указанного диапазона.

При преобразовании из типов данных **REAL** и **LREAL** входное значение сначала закругляется на ближайшую целую цифру и после этого проводится преобразование.

Использование преобразования **..._TO_SINT** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_SINT
  VAR_OUTPUT
    varSint     : ARRAY[1..15] OF SINT;
  END_VAR

  varSint[1]  :=   BOOL_TO_SINT( BOOL#0);              // 0
  varSint[2]  :=   BOOL_TO_SINT( true);                // 1
  varSint[3]  := STRING_TO_SINT( '+127');              // 127
  varSint[4]  :=    INT_TO_SINT( INT#-128);            // -128
  varSint[5]  :=   DINT_TO_SINT( 1_235_678);           // -34
  varSint[6]  :=  USINT_TO_SINT( 255);                 // -1
  varSint[7]  :=   UINT_TO_SINT( UINT#16#FFFF);        // -1
  varSint[8]  :=  UDINT_TO_SINT( 16#FFFF_FFFE);        // -2
  varSint[9]  :=   REAL_TO_SINT( -12.6);               // -13
  varSint[10] :=  LREAL_TO_SINT( 123.4567);            // 123
  varSint[11] :=   TIME_TO_SINT( T#0:00:00.100);       // 100
  varSint[12] :=   BYTE_TO_SINT( BYTE#16#AF);          // -81
  varSint[13] :=   WORD_TO_SINT( 1122);                //
  varSint[14] :=  DWORD_TO_SINT( 12345678);            // 78

END_FUNCTION_BLOCK
```

4.3 Функции ANY_TO_INT

Преобразование переменной произвольного элементарного типа данных на тип **INT**.

К данной группе функций принадлежит следующая функция:
BOOL_TO_INT
SINT_TO_INT, DINT_TO_INT
USINT_TO_INT, UINT_TO_INT, UDINT_TO_INT
REAL_TO_INT, LREAL_TO_INT
STRING_TO_INT
TIME_TO_INT, TOD_TO_INT, DATE_TO_INT, DT_TO_INT

Результатом преобразования **..._TO_INT** является цифра в пределах <–32 768, +32 767>. Если конвертируем от большего типа данных к меньшему (напр. **DINT_TO_INT**) рискуем потерей информации в случае, если входное значение находится вне указанного диапазона.

При преобразовании от типов данных **REAL** и **LREAL** входное значение сначала заокруглено на ближайшую целую цифру и после этого проводится преобразование.

Использование преобразования **..._TO_INT** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_INT
  VAR_OUTPUT
    varInt     : ARRAY[1..10] OF INT;
  END_VAR

  varInt[1]  :=   BOOL_TO_INT( true);            // 1
  varInt[2]  :=   SINT_TO_INT( 99);              // 99
  varInt[3]  :=   REAL_TO_INT( 123.5678);        // 124
  varInt[4]  :=  LREAL_TO_INT( LREAL#5.21E3);    // 5210
  varInt[5]  := STRING_TO_INT( '-12.5');         // -12
  varInt[6]  := STRING_TO_INT( '4.47E+06');      // 4
  varInt[7]  :=   TIME_TO_INT( T#12s25ms);       // 12025
  varInt[8]  :=   BYTE_TO_INT( BYTE#16#AF);      // 175
  varInt[9]  :=   WORD_TO_INT( 1122);            // 1122
  varInt[10] :=  DWORD_TO_INT( 16#1234_5678);    // 16#5678

END_FUNCTION_BLOCK
```

4.4 Функции ANY_TO_DINT

Преобразование переменной произвольного элементарного типа данных на тип **DINT**.

К данной группе функций принадлежит следующие функции:
BOOL_TO_DINT
SINT_TO_DINT, INT_TO_DINT
USINT_TO_DINT, UINT_TO_DINT, UDINT_TO_DINT
REAL_TO_DINT, LREAL_TO_DINT
STRING_TO_DINT
TIME_TO_DINT, TOD_TO_DINT, DATE_TO_DINT, DT_TO_DINT

Результатом преобразования **..._TO_DINT** является цифра в пределах <–2 147 483 648, +2 147 483 647>.

При преобразовании с типов данных **REAL** и **LREAL** входное значение сначала закругляется на ближайшую целую цифру и после этого проводится преобразование.

Использование преобразования **..._TO_DINT** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_DINT
  VAR_OUTPUT
    varDint     : ARRAY[1..10] OF DINT;
  END_VAR

  varDint[1]  :=   BOOL_TO_DINT( true);           // 1
  varDint[2]  :=   SINT_TO_DINT( 99);             // 99
  varDint[3]  :=   REAL_TO_DINT( 123.5678);       // 124
  varDint[4]  :=  LREAL_TO_DINT( 5.21E3);         // 5210
  varDint[5]  := STRING_TO_DINT( '-12.5');        // -12
  varDint[6]  := STRING_TO_DINT( '4.47E+06');     // 4
  varDint[7]  :=   TIME_TO_DINT( T#12s25ms);      // 12025
  varDint[8]  :=   BYTE_TO_DINT( BYTE#16#AF);     // 175
  varDint[9]  :=   WORD_TO_DINT( 1122);           // 1122
  varDint[10] :=  DWORD_TO_DINT( 1234_5678);      // 1234_5678

END_FUNCTION_BLOCK
```

4.5 Функции ANY_TO_USINT

Преобразование переменной произвольного элементарного типа данных на тип **USINT**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_USINT
SINT_TO_USINT, INT_TO_USINT, DINT_TO_USINT
UINT_TO_USINT, UDINT_TO_USINT
REAL_TO_USINT, LREAL_TO_USINT
STRING_TO_USINT
TIME_TO_USINT, TOD_TO_USINT, DATE_TO_USINT, DT_TO_USINT

Результатом преобразования **..._TO_USINT** является цифра в пределах < 0, 255>. Если конвертируем от большего типа данных до меньшего (напр. **UDINT_TO_USINT**) рискуем потерей информации в случае, если входное значение находится вне указанного диапазона.

При преобразовании с типов данных **REAL** и **LREAL** входное значение сначала закруглено на ближайшую целую цифру и после этого проводится преобразование.

Использование преобразования **..._TO_USINT** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_USINT
  VAR_OUTPUT
    varUsint     : ARRAY[1..15] OF USINT;
  END_VAR

  varUsint[1]  :=   BOOL_TO_USINT( BOOL#0);              // 0
  varUsint[2]  :=   BOOL_TO_USINT( true);                // 1
  varUsint[3]  := STRING_TO_USINT( '+127');              // 127
  varUsint[4]  :=   SINT_TO_USINT( -1);                  // 255
  varUsint[5]  :=    INT_TO_USINT( INT#-128);            // 128
  varUsint[6]  :=   DINT_TO_USINT( DINT#1_345_678);      // 142
  varUsint[7]  :=   UINT_TO_USINT( UINT#16#FFFF);        // 255
  varUsint[8]  :=  UDINT_TO_USINT( 16#FFFF_FFFE);        // 254
  varUsint[9]  :=   REAL_TO_USINT( 12.6);                // 13
  varUsint[10] :=  LREAL_TO_USINT( 123.4567);            // 123
  varUsint[11] :=   TIME_TO_USINT( T#0:00:00.100);       // 100
  varUsint[12] :=   BYTE_TO_USINT( BYTE#16#AF);          // 175
  varUsint[13] :=   WORD_TO_USINT( 1122);                // 98
  varUsint[14] :=  DWORD_TO_USINT( 12345678);            // 78

END_FUNCTION_BLOCK
```

4.6 Функции ANY_TO_UINT

Преобразование переменной произвольного элементарного типа данных на тип **UINT**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_UINT
SINT_TO_UINT, INT_TO_UINT, DINT_TO_UINT
USINT_TO_UINT, UDINT_TO_UINT
REAL_TO_UINT, LREAL_TO_UINT
STRING_TO_UINT
TIME_TO_UINT, TOD_TO_UINT, DATE_TO_UINT, DT_TO_UINT

Результатом преобразования **..._TO_UINT** является цифра в пределах < 0, +65535>. Если конвертируем от большего типа данных к меньшему (напр. **UDINT_TO_UINT**) рискуем потерей информации в случае, если входное значение находится вне указанного диапазона.

При преобразовании с типов данных **REAL** и **LREAL** входное значение сначала закруглено на ближайшую целую цифру и после этого проводится преобразование.

Использование преобразования **..._TO_UINT** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_UINT
  VAR_OUTPUT
    varUint     : ARRAY[1..15] OF UINT;
  END_VAR

  varUint[1]  :=   BOOL_TO_UINT( BOOL#0);              // 0
  varUint[2]  :=   BOOL_TO_UINT( true);                // 1
  varUint[3]  := STRING_TO_UINT( '+127');              // 127
  varUint[4]  :=   SINT_TO_UINT( -1);                  // 65535
  varUint[5]  :=    INT_TO_UINT( INT#-128);            // 65408
  varUint[6]  :=   DINT_TO_UINT( DINT#1_345_678);      // 34958
  varUint[7]  :=  USINT_TO_UINT( USINT#16#FF);         // 255
  varUint[8]  :=  UDINT_TO_UINT( 16#FFFF_FFFE);        // 65534
  varUint[9]  :=   REAL_TO_UINT( 12.6);                // 13
  varUint[10] :=  LREAL_TO_UINT( 123.4567);            // 123
  varUint[11] :=   TIME_TO_UINT( T#0:00:00.100);       // 100
  varUint[12] :=   BYTE_TO_UINT( BYTE#16#AF);          // 175
  varUint[13] :=   WORD_TO_UINT( 1122);                // 1122
  varUint[14] :=  DWORD_TO_UINT( 12345678);            // 24910

END_FUNCTION_BLOCK
```

4.7 Функции ANY_TO_ UDINT

Преобразование переменной произвольного элементарного типа данных на тип **UDINT**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_UDINT
SINT_TO_UDINT, INT_TO_UDINT, DINT_TO_UDINT
USINT_TO_UDINT, UINT_TO_UDINT
REAL_TO_UDINT, LREAL_TO_UDINT
STRING_TO_UDINT
TIME_TO_UDINT, TOD_TO_UDINT, DATE_TO_UDINT, DT_TO_UDINT

Результатом преобразования **..._TO_UDINT** является цифра в пределах <0, +4 294 967 295>. Если конвертированное значение находится вне пределов данного диапазона (напр. при преобразовании **REAL_TO_UDINT** или в случае отрицательных чисел) не определен результат преобразования.

При преобразовании с типов данных **REAL** и **LREAL** входное значение сначала закруглено на ближайшую целую цифру и после этого проводится преобразование.

Использование преобразования **..._TO_UDINT** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_UDINT
  VAR_OUTPUT
    varUdint     : ARRAY[1..15] OF UDINT;
  END_VAR

  varUdint[1]  :=   BOOL_TO_UDINT( BOOL#0);              // 0
  varUdint[2]  :=   BOOL_TO_UDINT( true);                // 1
  varUdint[3]  := STRING_TO_UDINT( '+127');              // 127
  varUdint[4]  :=   SINT_TO_UDINT( -1);                  // 4 294 967 295
  varUdint[5]  :=    INT_TO_UDINT( INT#-128);            // 4 294 967 168
  varUdint[6]  :=   DINT_TO_UDINT( DINT#1_345_678);      // 1 345 678
  varUdint[7]  :=  USINT_TO_UDINT( USINT#16#FF);         // 255
  varUdint[8]  :=   UINT_TO_UDINT( UINT#16#FFFE);        // 65534
  varUdint[9]  :=   REAL_TO_UDINT( 12.6);                // 13
  varUdint[10] :=  LREAL_TO_UDINT( 123.4567);            // 123
  varUdint[11] :=   TIME_TO_UDINT( T#0:00:00.100);       // 100
  varUdint[12] :=   BYTE_TO_UDINT( BYTE#16#AF);          // 175
  varUdint[13] :=   WORD_TO_UDINT( 1122);                // 1122
  varUdint[14] :=  DWORD_TO_UDINT( 12345678);            // 12345678

END_FUNCTION_BLOCK
```

4.8 Функции ANY_TO_ REAL

Преобразование переменной произвольного элементарного типа данных на тип **REAL**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_REAL
SINT_TO_REAL, INT_TO_REAL, DINT_TO_REAL
USINT_TO_REAL, UINT_TO_REAL, UDINT_TO_REAL
LREAL_TO_REAL
STRING_TO_REAL
TIME_TO_REAL, TOD_TO_REAL, DATE_TO_REAL, DT_TO_REAL

Использование преобразования **..._TO_REAL** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_REAL
  VAR_OUTPUT
    varReal     : ARRAY[1..20] OF REAL;
  END_VAR

  varReal[1]  :=   BOOL_TO_REAL( 0);            // 0.0
  varReal[2]  :=   BOOL_TO_REAL( true);         // 1.0
  varReal[3]  :=   SINT_TO_REAL( -99);          // -99.0
  varReal[4]  :=    INT_TO_REAL( -9900);        // -9900.0
  varReal[5]  :=   DINT_TO_REAL( -1_235_678);   // -1 235 678.0
  varReal[6]  :=  USINT_TO_REAL( 99);           // 99.0
  varReal[7]  :=   UINT_TO_REAL( 9900);         // 9900.0
  varReal[8]  :=  UDINT_TO_REAL( 1_235_678);    // 1 235 678.0
  varReal[9]  := STRING_TO_REAL( '-12.5');      // -12.5
  varReal[10] := STRING_TO_REAL( '4.47E6');     // 4470000.0
  varReal[11] := STRING_TO_REAL( '4.47-E6');    // 4.47
  varReal[12] :=   TIME_TO_REAL( T#12s25ms);    // 12025.0
  varReal[13] :=  BYTE_TO_REAL( 16#AF);         // 175.0
  varReal[14] :=  WORD_TO_REAL( 1122);          // 1122.0
  varReal[15] := DWORD_TO_REAL( 1234567);       // 1234567.0

END_FUNCTION_BLOCK
```

4.9 Функции ANY_TO_ LREAL

Преобразование переменной произвольного элементарного типа данных на тип **LREAL**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_LREAL
SINT_TO_LREAL, INT_TO_LREAL, DINT_TO_LREAL
USINT_TO_LREAL, UINT_TO_LREAL, UDINT_TO_LREAL
REAL_TO_LREAL
STRING_TO_LREAL
TIME_TO_LREAL, TOD_TO_LREAL, DATE_TO_LREAL, DT_TO_LREAL

Использование преобразования **..._TO_LREAL** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_LREAL
  VAR_OUTPUT
    varLreal    : ARRAY[1..10] OF LREAL;
  END_VAR

  varLreal[1] :=   BOOL_TO_LREAL( false);             // 0.0
  varLreal[2] :=   BOOL_TO_LREAL( true);              // 1.0
  varLreal[3] :=   SINT_TO_LREAL( 99);                // 99.0
  varLreal[4] :=   DINT_TO_LREAL( 1_235_678);         // 1 235 678.0
  varLreal[5] := STRING_TO_LREAL( '-12.5');           // -12.5
  varLreal[6] := STRING_TO_LREAL( '4.47E+05');        // 447000.0
  varLreal[7] :=   TIME_TO_LREAL( T#12s25ms);         // 12025.0
  varLreal[8] :=    TOD_TO_LREAL( TOD#12:33:05.120);  // 45185120.0
                // 45185120.0 = 120 + 5*1000 + 33*1000*60 + 12*1000*60*60

END_FUNCTION_BLOCK
```

4.10 Функции ANY_TO_ STRING

Преобразование переменной произвольного элементарного типа данных на тип **STRING**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_STRING
SINT_TO_STRING, INT_TO_STRING, DINT_TO_STRING
USINT_TO_STRING, UINT_TO_STRING, UDINT_TO_STRING
REAL_TO_STRING, LREAL_TO_STRING
TIME_TO_STRING, TOD_TO_STRING, DATE_TO_STRING, DT_TO_STRING

Результатом преобразования **..._TO_STRING** является текстовая последовательность.

Результатом преобразования **ANY_NUM** на тип **STRING** является последовательность, содержащая цифру в десятичной системе. Знак + у положительных результатов не указывается. Возможная десятичная часть цифры отделена десятичной точкой.

У преобразования времени и даты начинается конечная текстовая последовательность сокращением переводимого типа данных (напр. у преобразования **TIME_TO_STRING** будет конечная последовательность начинаться '**T#**...'). Потом следует текущее время в формате, который отвечает литералам времени при инициализации переменных времени.

Использование преобразования **..._TO_STRING** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_STRING
  VAR_OUTPUT
    varString   : ARRAY[1..20] OF STRING[30];
  END_VAR

  varString[1]  :=  BOOL_TO_STRING( false);           // '0'
  varString[2]  :=  BOOL_TO_STRING( true);            // '1'
  varString[3]  :=  SINT_TO_STRING( 99);              // '99'
  varString[4]  :=   INT_TO_STRING( -9_998);          // '-9998'
  varString[5]  :=  DINT_TO_STRING( 1_235_678);       // '1235678'
  varString[6]  := USINT_TO_STRING( 255);             // '255'
  varString[7]  :=  UINT_TO_STRING( 16#FFFF);         // '65535'
  varString[8]  := UDINT_TO_STRING( 16#FFFF_FFFF);    // '4294967295'
  varString[9]  :=  REAL_TO_STRING( -123E5);          // '-12300000.00000'
  varString[10] := LREAL_TO_STRING( 123.4567);        // '123.456700'
  varString[11] :=  TIME_TO_STRING( TIME#12s25ms);    // 'T#0:00:12.025'
  varString[12] :=   TOD_TO_STRING( TOD#12:33:05.120);// 'TOD#12:33:5.120'
  varString[13] :=  DATE_TO_STRING( DATE#2003-10-21); // 'D#2003-10-21'
  varString[14] :=    DT_TO_STRING( DT#2003-10-21-16:35:59);
                                                      // 'DT#2003-10-21-16:35:59.000'
  varString[15] :=  BYTE_TO_STRING( 16#AF);           // '175'
  varString[16] :=  WORD_TO_STRING( 1122);            // '1122'
  varString[17] := DWORD_TO_STRING( 12345678);        // '12345678'

END_FUNCTION_BLOCK
```

4.11 Функции ANY_TO_ TIME

Преобразование переменной произвольного элементарного типа данных на тип **TIME**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_TIME
SINT_TO_TIME, INT_TO_TIME, DINT_TO_TIME
USINT_TO_TIME, UINT_TO_TIME, UDINT_TO_TIME
REAL_TO_TIME, LREAL_TO_TIME
STRING_TO_TIME
TOD_TO_TIME, DATE_TO_TIME, DT_TO_TIME

Результатом преобразования **..._TO_TIME** является цифры, которые определяют количество милисекунд.

При преобразовании с типа **STRING** должна входная последовательность отвечать литералу **TIME**.

Использование преобразования **..._TO_TIME** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_TIME
  VAR_OUTPUT
    varTime     : ARRAY[1..11] OF TIME;
  END_VAR

  varTime[1]  :=   BOOL_TO_TIME( BOOL#0);            // T#00h 00m 00.000s
  varTime[2]  :=   BOOL_TO_TIME( true);              // T#00h 00m 00.001s
  varTime[3]  :=    INT_TO_TIME( 1000);              // T#00h 00m 01.000s
  varTime[4]  :=   DINT_TO_TIME( 10*1000);           // T#00h 00m 10.000s
  varTime[5]  :=  UDINT_TO_TIME( 60*1000);           // T#00h 01m 00.000s
  varTime[6]  :=   REAL_TO_TIME( 60.*60.*1000.);     // T#01h 00m 00.000s
  varTime[7]  :=  LREAL_TO_TIME( 11.*60.*60.*1000.); // T#11h 00m 00.000s
  varTime[8]  := STRING_TO_TIME( 'T#06:30:15.250');  // T#06h 30m 15.250s
  varTime[9]  := STRING_TO_TIME( '06:30:15.250');    // T#00h 00m 00.000s
  varTime[10] := DT_TO_TIME( DT#1970-01-01-12:34:56);// T#12h 34m 56.000s

END_FUNCTION_BLOCK
```

4.12 Функции ANY_TO_ TIME_OF_DAY

Преобразование переменной произвольного элементарного типа данных на тип **TIME_OF_DAY**. Для данного типа данных можно также использовать сокращенное обозначение **TOD**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_TIME_OF_DAY
SINT_TO_TIME_OF_DAY, INT_TO_TIME_OF_DAY, DINT_TO_TIME_OF_DAY
USINT_TO_TIME_OF_DAY, UINT_TO_TIME_OF_DAY, UDINT_TO_TIME_OF_DAY
REAL_TO_TIME_OF_DAY, LREAL_TO_TIME_OF_DAY
STRING_TO_TIME_OF_DAY
TIME_TO_TIME_OF_DAY, DATE_TO_TIME_OF_DAY, DT_TO_TIME_OF_DAY
resp.
BOOL_TO_TOD
SINT_TO_TOD, INT_TO_TOD, DINT_TO_TOD
USINT_TO_TOD, UINT_TO_TOD, UDINT_TO_TOD
REAL_TO_TOD, LREAL_TO_TOD
STRING_TO_TOD
TIME_TO_TOD, DATE_TO_TOD, DT_TO_TOD

Результатом преобразования **..._TO_TIME_OF_DAY** являются цифры, которые определяют количество милисекунд.

При преобразования от типа **STRING** должна входная последовательность отвечать литералу **TOD**.

Использование преобразования **..._TO_TIME_OF_DAY** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_TIME_OF_DAY
  VAR_OUTPUT
    varTod     : ARRAY[1..11] OF TOD;
  END_VAR

  varTod[1]  :=   BOOL_TO_TOD( BOOL#0);              // TOD#00:00:00.000
  varTod[2]  :=   BOOL_TO_TOD( true);                // TOD#00:00:00.001
  varTod[3]  :=    INT_TO_TOD( 1000);                // TOD#00:00:01.000
  varTod[4]  :=   DINT_TO_TOD( 10*1000);             // TOD#00:00:10.000
  varTod[5]  :=  UDINT_TO_TOD( 60*1000);             // TOD#00:01:00.000
  varTod[6]  :=   REAL_TO_TOD( 60.*60.*1000.);       // TOD#01:00:00.000
  varTod[7]  :=  LREAL_TO_TOD( 11.*60.*60.*1000.);   // TOD#11:00:00.000
  varTod[8]  := STRING_TO_TOD( 'TOD#06:30:15.250');  // TOD#06:30:15.250
  varTod[9]  := STRING_TO_TOD( '06:30:15.250');      // TOD#00:00:00.000
  varTod[10] :=  DT_TO_TOD( DT#1970-01-01-12:34:56); // TOD#12:34:56.000

END_FUNCTION_BLOCK
```

4.13 Функции ANY_TO_DATE

Преобразование переменной произвольного элементарного типа данных на тип **DATE**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_DATE
SINT_TO_DATE, INT_TO_DATE, DINT_TO_DATE
USINT_TO_DATE, UINT_TO_DATE, UDINT_TO_DATE
REAL_TO_DATE, LREAL_TO_DATE
STRING_TO_DATE
TIME_TO_DATE, TOD_TO_DATE, DT_TO_DATE

Результатом преобразования **..._TO_DATE** является цифра, целую часть которой определяют количество секунд от 00:00:00 1.1.1970.

При преобразовании с типа **STRING** должна входная последовательность отвечать литералу **DATE**.

Использование преобразования **..._TO_DATE** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_DATE
  VAR_OUTPUT
    varDate     : ARRAY[1..11] OF DATE;
  END_VAR

  varDate[1]  :=   BOOL_TO_DATE( BOOL#0);                 // D#1970-01-01
  varDate[2]  :=   BOOL_TO_DATE( true);                   // D#1970-01-01
  varDate[3]  :=   DINT_TO_DATE( 24*60*60);               // D#1970-01-02
  varDate[4]  :=  UDINT_TO_DATE( 11*24*60*60);            // D#1970-01-12
  varDate[5]  :=   REAL_TO_DATE( 365.*24.*60.*60.);       // D#1971-01-01
  varDate[6]  :=  LREAL_TO_DATE( 10.*365.*24.*60.*60.);   // D#1979-12-30
  varDate[7]  :=   TIME_TO_DATE( T#25:00:00.0);           // D#1970-01-02
  varDate[8]  := STRING_TO_DATE( 'D#2003-12-24');         // D#2003-12-24
  varDate[9]  := STRING_TO_DATE( '2003-12-24');           // D#1970-01-01
  varDate[10] :=     DT_TO_DATE( DT#2002-11-25-00:00:00); // D#2002-11-25
  varDate[11] :=     DT_TO_DATE( DT#2002-11-25-12:22:33); // D#2002-11-26

END_FUNCTION_BLOCK
```

4.14 Функции ANY_TO_ DATE_AND_TIME

Преобразование переменной произвольного элементарного типа данных на тип **DATE_AND_TIME**.

К данной группе функций принадлежат следующие функции:
BOOL_TO_DATE_AND_TIME
SINT_TO_DATE_AND_TIME, INT_TO_DATE_AND_TIME,
DINT_TO_DATE_AND_TIME, USINT_TO_DATE_AND_TIME
UINT_TO_DATE_AND_TIME, UDINT_TO_DATE_AND_TIME
REAL_TO_DATE_AND_TIME, LREAL_TO_DATE_AND_TIME
STRING_TO_DATE_AND_TIME
TIME_TO_DATE_AND_TIME, TOD_TO_DATE_AND_TIME
DATE_TO_DATE_AND_TIME
resp.
BOOL_TO_DT
SINT_TO_DT, INT_TO_DT, DINT_TO_DT
USINT_TO_DT, UINT_TO_DT, UDINT_TO_DT
REAL_TO_DT, LREAL_TO_DT
STRING_TO_DT
TIME_TO_DT, TOD_TO_DT
DATE_TO_DT

Результатом преобразования **..._TO_DATE_AND_TIME** является цифра, целая часть которой определяют количество секунд от 00:00:00 1.1.1970. Десятичная часть цифры потом представляет милисекунды.

При преобразовании с типа **STRING** должна входная последовательность отвечать литералу **DATE_AND_TIME**.

Использование преобразования **..._TO_DATE_AND_TIME** указано на следующем примере:

```
FUNCTION_BLOCK Example_ANY_TO_DT
  VAR_OUTPUT
    varDt     : ARRAY[1..11] OF DATE_AND_TIME;
  END_VAR

  varDt[1]  :=  BOOL_TO_DT( BOOL#0);              // DT#1970-01-01-00:00:00
  varDt[2]  :=  BOOL_TO_DT( true);                // DT#1970-01-01-00:00:01
  varDt[3]  :=  DINT_TO_DT( 24*60*60);            // DT#1970-01-02-00:00:00
  varDt[4]  := UDINT_TO_DT( 11*24*60*60 + 35);    // DT#1970-01-12-00:00:35
  varDt[5]  :=  REAL_TO_DT( 365.*24.*60.*60.);    // DT#1971-01-01-00:00:00
  varDt[6]  := LREAL_TO_DT( 10.*365.*24.*60.*60.);//DT#1979-12-30-00:00:00
  varDt[7]  :=   TIME_TO_DT( T#25:00:00.0);       // DT#1970-01-02-01:00:00
  varDt[8]  := STRING_TO_DT( 'DT#2003-12-24-18:10:20');
                                                  // DT#2003-12-24-18:10:20
  varDt[9]  := STRING_TO_DT( '2003-12-24');       // DT#1970-01-01-00:00:00
  varDt[10] :=   DATE_TO_DT( DATE#2002-11-25);    // DT#2002-11-25-00:00:00

END_FUNCTION_BLOCK
```

5 АРИФМЕТИЧЕСКИЕ ФУНКЦИИ

5.1 Функция ABS Абсолютное значение

Функция **ABS** вернет абсолютное значение входного параметра. Входной параметр может быть типа **ANY_NUM**.

```
PROGRAM ExampleABS
  VAR
    varA         : INT  := -22;
    varR         : REAL := -12.5;
    absA         : INT;
    absR         : REAL;
    absI,absL    : LREAL;
  END_VAR

  absA := ABS( varA);                    // 22
  absR := ABS( varR);                    // 12.5
  absI := ABS( INT_TO_LREAL(-123));      // 123
  absL := ABS( LREAL#-345.678);          // 345.678
END_PROGRAM
```

5.2 Функция SQRT Корень

Функция **SQRT** вернет квадратный корень входного параметра. Входной параметр не должен быть отрицательным.

```
PROGRAM ExampleSQRT
  VAR
    varR         : REAL := 144;
    sqrtR        : REAL;
    sqrtI,sqrtL  : LREAL;
  END_VAR

  sqrtR := SQRT( varR);                    // 12
  sqrtI := SQRT( INT_TO_LREAL(-123));      // NaN
  sqrtL := SQRT( 1024.0);                  // 32.0
END_PROGRAM
```

5.3 Функция LN Натуральный логарифм

Функция **LN** вернет натуральный логарифм входного параметра.

```
PROGRAM ExampleLN
  VAR
    varA         : INT  := 1;
    varR         : REAL := 2.718282;
    lnA, lnR     : REAL;
    lnI, lnL     : LREAL;
  END_VAR

  lnA := LN( INT_TO_REAL(varA));       // 0
  lnR := LN( varR);                    // 1.00000
  lnI := LN( INT_TO_LREAL(-123));      // NaN
  lnL := LN( 22026.3);                 // 9.99999
END_PROGRAM
```

5.4 Функция LOG Десятичный логарифм

Функция **LOG** вернет десятичный логарифм входного параметра.

```
PROGRAM ExampleLOG
  VAR
    varA         : INT  := 1;
    varR         : REAL := 100.0;
    logA, logR   : REAL;
    logI, logL   : LREAL;
  END_VAR

  logA := LOG( INT_TO_REAL(varA));       // 0
  logR := LOG( varR);                    // 2.0
  logI := LOG( INT_TO_LREAL(-123));      // NaN
  logL := LOG( LREAL#1_000.0);           // 3.0
END_PROGRAM
```

5.5 Функция EXP Естественная показательная функция

Функция **EXP** вернет значение e^x, где x это входной параметр.

```
PROGRAM ExampleEXP
  VAR
    exp1, exp2  : REAL;
    exp3        : LREAL;
  END_VAR

  exp1 := EXP( REAL#2.0);                  //  7.3890
  exp2 := EXP( 0.0);                       //  1.0
  exp3 := EXP( 1.0);                       //  2.7182
END_PROGRAM
```

5.6 Функция SIN Синус входного угла

Функция **SIN** вернет синус входного параметра, заданного в радианах. Входной параметр должен быть в интервале < -π/2, π/2 >.

```
PROGRAM ExampleSIN
  VAR CONSTANT
    PI          : LREAL := LREAL#3.14159265358979323846;
  END_VAR
  VAR
    sin1, sin2  : REAL;
    sin3, sin4  : LREAL;
  END_VAR

  sin1 := SIN( REAL#-3.14159 / 2.0);       // -1.0
  sin2 := SIN( 0.0);                       //  0.0
  sin3 := SIN( PI / 2.);                   //  1.0
  sin4 := SIN( 2.0 * PI);                  //  0.0
END_PROGRAM
```

5.7 Функция COS Косинус входного угла

Функция **COS** вернет косинус входного параметра, заданного в радианах. Входной параметр должен быть в интервале < -π/2, π/2 >.

```
PROGRAM ExampleCOS
  VAR CONSTANT
    PI          : LREAL := LREAL#3.14159265358979323846;
  END_VAR
  VAR
    cos1, cos2  : REAL;
    cos3, cos4  : LREAL;
  END_VAR

  cos1 := COS( REAL#-3.1415 / 2.0);        //  0.0
  cos2 := COS( 0.0);                       //  1.0
  cos3 := COS( PI / 2.);                   //  0.0
  cos4 := COS( 2.0 * PI);                  //  1.0
END_PROGRAM
```

5.8 Функция TAN Тангенс входного угла

Функция **TAN** вернет тангенс входного параметра, заданного в радианах. Входной параметр должен быть в интервале < -π/2, π/2 >.

```
PROGRAM ExampleTAN
  VAR CONSTANT
    PI          : LREAL := LREAL#3.14159265358979323846;
  END_VAR
  VAR
    tan1, tan2  : REAL;
    tan3, tan4  : LREAL;
  END_VAR

  tan1 := TAN( REAL#-3.14159 / 4.0);       //  -1.0
  tan2 := TAN( 0.0);                       //   0.0
  tan3 := TAN( PI / 2.);                   //  +INF
  tan4 := TAN( -PI);                       //  -INF
END_PROGRAM
```

5.9 Функция ASIN Дуга синуса

Функция **ASIN** вернет дугу синуса входного параметра. Входной параметр должен быть в интервале < -1, 1 >.

```
PROGRAM ExampleASIN
  VAR
    asin1, asin2  : REAL;
    asin3, asin4  : LREAL;
  END_VAR

  asin1 := ASIN( REAL#-1.0);            // -1.570796 (-PI/2)
  asin2 := ASIN( 0.0);                  //  0.0
  asin3 := ASIN( 0.0);                  //  0.0
  asin4 := ASIN( 1.0);                  //  1.570796 (PI/2)
END_PROGRAM
```

5.10 Функция ACOS Дуга косинуса

Функция **ACOS** вернет дугу косинуса входного параметра. Входной параметр должен быть в интервале < -1, 1 >.

```
PROGRAM ExampleACOS
  VAR
    acos1, acos2  : REAL;
    acos3         : LREAL;
  END_VAR

  acos1 := ACOS( REAL#-1.0);            //  3.14159 (PI)
  acos2 := ACOS( 0.0);                  //  1.15079 (PI/2)
  acos3 := ACOS( 1.0);                  //  0.0
END_PROGRAM
```

5.11 Функция ATAN Дуга тангенса

Функция **ATAN** вернет дугу тангенса входного параметра.

```
PROGRAM ExampleATAN
  VAR CONSTANT
    PI          : LREAL := LREAL#3.14159265358979323846;
  END_VAR
  VAR
    atan1, atan2  : REAL;
    atan3         : LREAL;
  END_VAR

  atan1 := ATAN( REAL#-3.14159 / 2.0);      //  -1.0
  atan2 := ATAN( 0.0);                      //   0.0
  atan3 := ATAN( PI / 2.);                  //  +1.0
END_PROGRAM
```

СОДЕРЖАНИЕ