Knihovny pro programování PLC Tecomat podle IEC 61 131-3

TXV 003 22.01
osmé vydání
březen 2006
změny vyhrazeny

Historie změn

Datum	Vydání	Popis změn
Srpen 2004 až únor 2006	1 až 7	Popis knihoven je součástí dokumentu TXV 003 21.01 Změny viz TXV 003 21.01
Březen 2006	8	Popis knihoven přesunut do samostatného dokumentu TXV 003 xx.01

1 KNIHOVNY

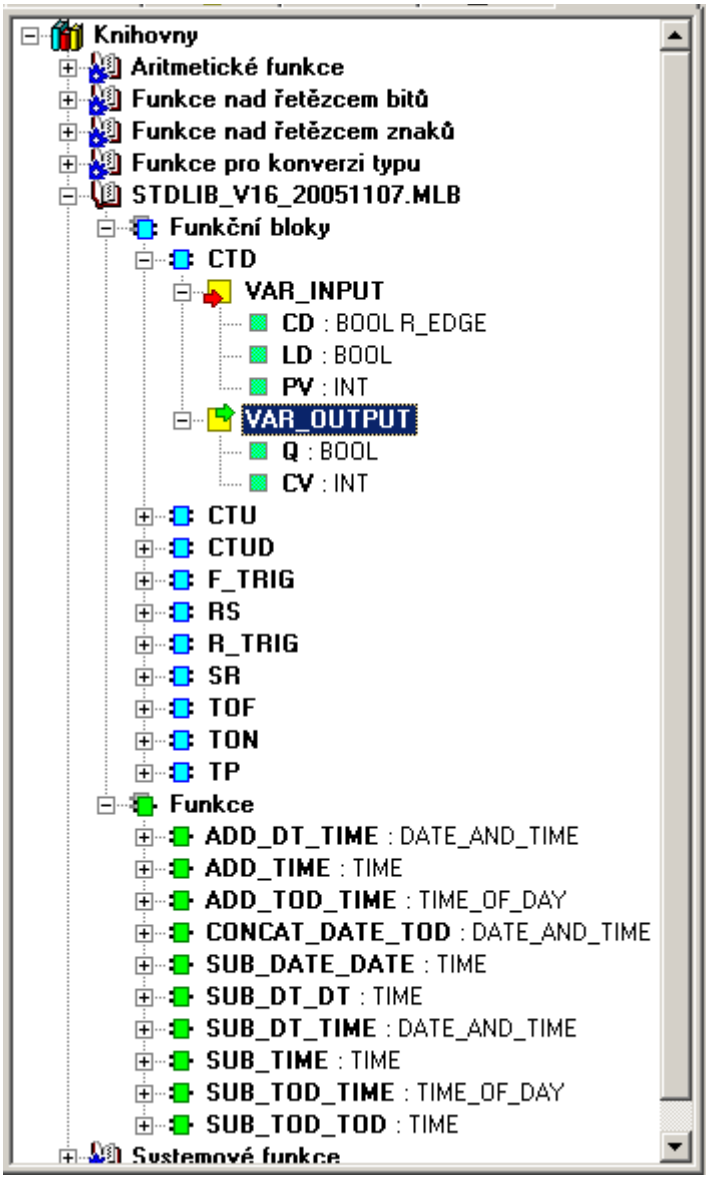
Knihovny funkcí a funkčních bloků jsou nedílnou součástí instalace programovacího prostředí Mosaic. Z hlediska jejich výstavby je možné knihovny rozdělit na následující typy:

- vestavěné (built-in) knihovny
- standardně dodávané externí knihovny
- uživatelsky definované knihovny

Knihovna může obsahovat deklarace funkcí, funkčních bloků, datových typů a globálních proměnných.

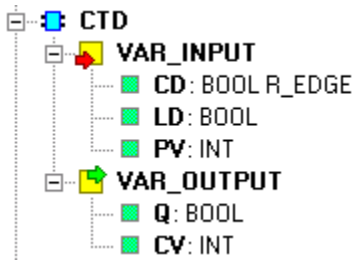
2 STANDARDNÍ KNIHOVNA STDLIB

Standardní knihovna obsahuje funkční bloky čítačů **CTD**, **CTU** a **CTUD**, časovačů **TON**, **TOF** a **TP**, klopných obvodů **RS** a **SR** a funkční bloky detekce hran **R_TRIG** a **F_TRIG**. Dále obsahuje funkce pro práci s datumem a časem. Následují obrázek ukazuje strukturu knihovny StdLib v prostředí Mosaic.



2.1 Funkční blok čítače dolů CTD

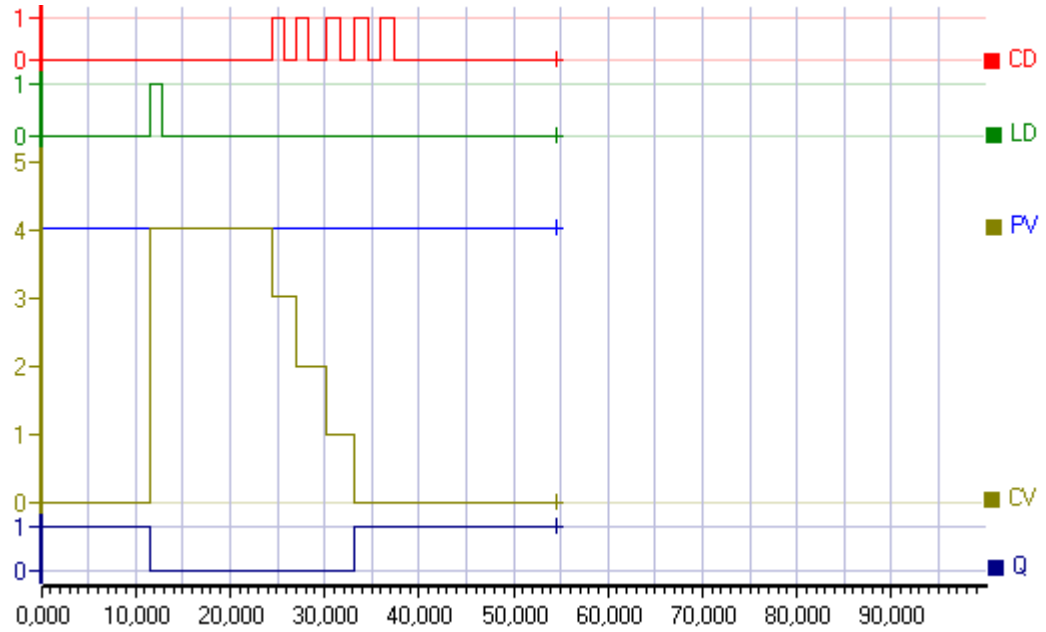
Funkční blok pro čítání dolů.



Vstupní proměnné :
CD vstup pro čítání dolů
LD vstup pro nastavení předvolby
PV předvolba čítače
Výstupní proměnné :
Q výstup čítače
CV hodnota čítače

Pokud má vstupní proměnná **LD** hodnotu **TRUE**, pak se hodnota předvolby **PV** naplní do hodnoty čítače **CV**. Pokud má vstupní proměnná **LD** hodnotu **FALSE** a vstupní proměnná **CD** přejde ze stavu **FALSE** do stavu **TRUE** (náběžná hrana), hodnota čítače **CV** je snížena o 1. Je-li hodnota čítače **CV** rovna nule, výstup čítače **Q** je nastaven na hodnotu **TRUE**. Jinak má hodnotu **FALSE**.

Chování čítače **CTD** vysvětluje následující obrázek.

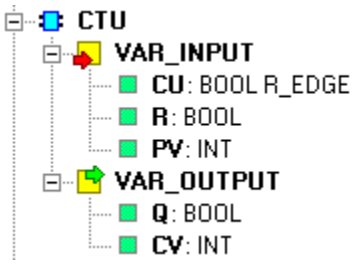


Příklad programu s voláním funkčního bloku **CTD** :

```
PROGRAM Citac
  VAR
    pulzy          : BOOL;
    setCTD         : BOOL;
    counterCTD     : CTD;              // instance čítače
    output         : BOOL;
  END_VAR

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

2.2 Funkční blok čítače nahoru CTU

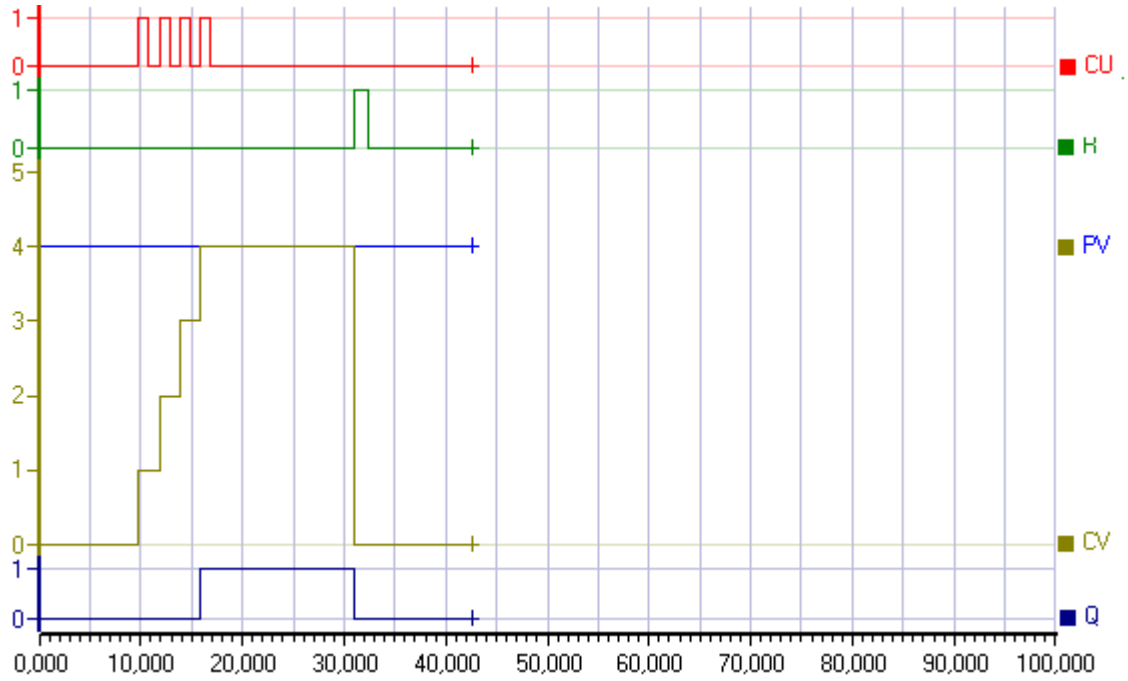
Funkční blok pro čítání nahoru.



Vstupní proměnné :
CU vstup pro čítání nahoru
R reset čítače
PV předvolba čítače
Výstupní proměnné :
Q výstup čítače
CV hodnota čítače

Pokud má vstupní proměnná **R** hodnotu **TRUE**, hodnota čítače **CV** je vynulovaná. Pokud má vstupní proměnná **R** hodnotu **FALSE** a vstupní proměnná **CU** přejde ze stavu **FALSE** do stavu **TRUE** (náběžná hrana), hodnota čítače **CV** je zvýšena o 1. Je-li hodnota čítače **CV** větší nebo rovná předvolbě **PV**, výstup čítače **Q** je nastaven na hodnotu **TRUE**. Jinak má hodnotu **FALSE**.

Chování čítače **CTU** vysvětluje následující obrázek.

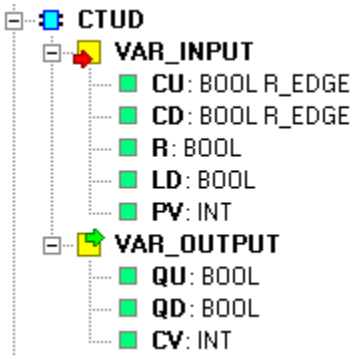


Příklad programu s voláním funkčního bloku **CTU** :

```
PROGRAM Citac
  VAR
    pulzy          : BOOL;
    resetCTU       : BOOL;
    counterCTU     : CTU;                  // instance čítače
    output         : BOOL;
  END_VAR

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

2.3 Funkční blok obousměrného čítače CTUD

Funkční blok pro obousměrné čítání.



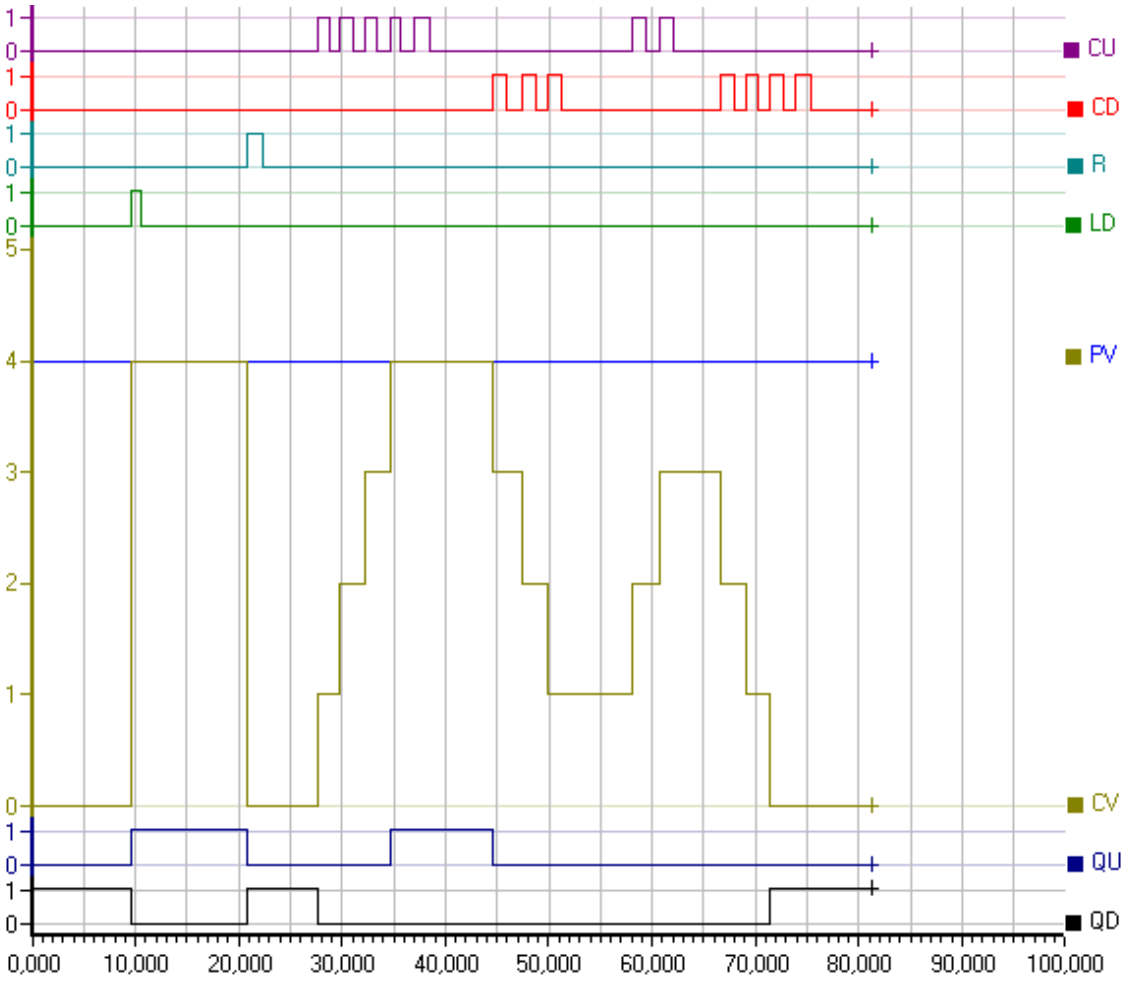
Vstupní proměnné :

CU	vstup pro čítání nahoru
CD	vstup pro čítání dolů
R	reset čítače
LD	vstup pro nastavení předvolby
PV	předvolba čítače

Výstupní proměnné :

QU	výstup čítače nahoru
QD	výstup čítače dolů
CV	hodnota čítače

Chování čítače **CTUD** vysvětluje následující obrázek.



Jestliže má vstupní proměnná **R** hodnotu **TRUE**, hodnota čítače **CV** bude vynulovaná. Jestliže má vstupní proměnná **LD** hodnotu **TRUE**, hodnota čítače **CV** bude nastavena na hodnotu předvolby **PV**.

Když vstupní proměnná **CU** přejde ze stavu **FALSE** do stavu **TRUE** (náběžná hrana), hodnota čítače **CV** je zvýšena o 1, čítač čítá nahoru. Podobně pokud vstupní proměnná **CD** přejde ze stavu **FALSE** do stavu **TRUE** (náběžná hrana), hodnota čítače **CV** je snížena o 1, čítač čítá dolů.

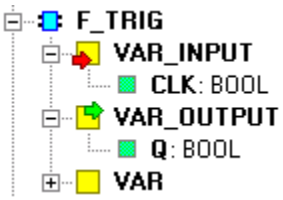
Je-li hodnota čítače **CV** větší nebo rovná předvolbě **PV**, výstup čítače **QU** je nastaven na hodnotu **TRUE**, jinak má hodnotu **FALSE**. Je-li hodnota čítače **CV** rovna nule, výstup čítače **QD** je nastaven na hodnotu **TRUE**, jinak má hodnotu **FALSE**.

Příklad programu s voláním funkčního bloku **CTUD** :

```
PROGRAM Citac
  VAR
    pulzyUP       : BOOL;
    pulzyDOWN     : BOOL;
    resetCTUD     : BOOL;
    setCTUD       : BOOL;
    counterCTUD   : CTUD;                    // instance čítače
    limitUP       : BOOL;
    limitDOWN     : BOOL;
  END_VAR

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

2.4 Funkční blok F_TRIG

Funkční blok pro pro detekci sestupné hrany.

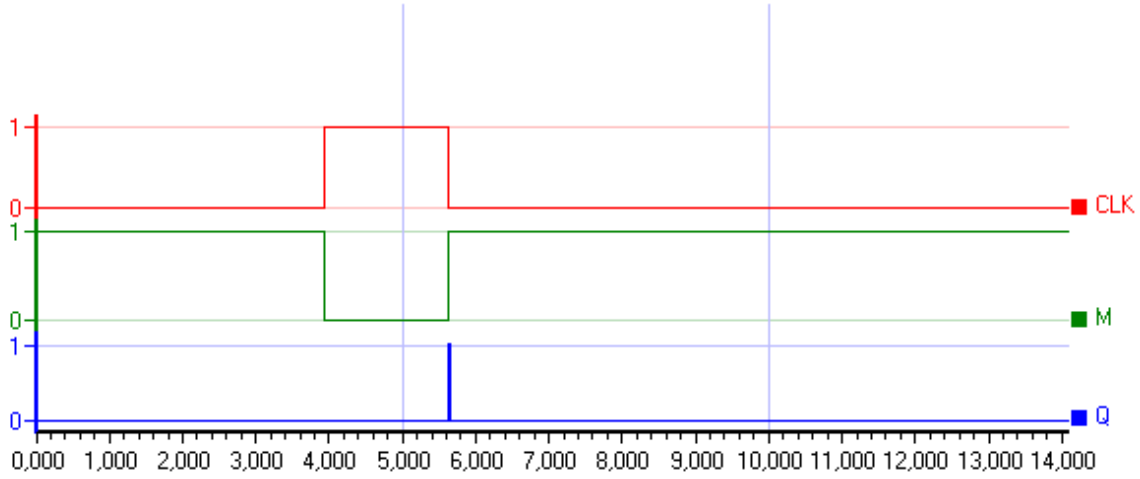


Vstupní proměnné :
CLK

Výstupní proměnné :
Q

Chování funkčního bloku **F_TRIG** odpovídá následujícímu programu v jazyce 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
```

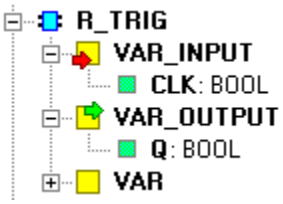


Příklad programu s voláním funkčního bloku **F_TRIG** :

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

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

2.5 Funkční blok R_TRIG

Funkční blok pro pro detekci náběžné hrany.

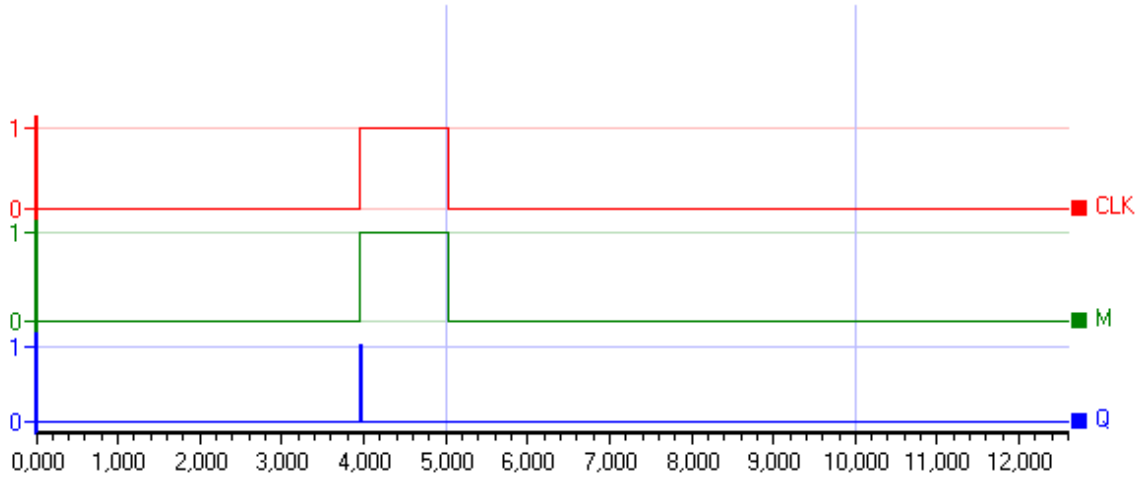


Vstupní proměnné :
CLK

Výstupní proměnné :
Q

Chování funkčního bloku **R_TRIG** odpovídá následujícímu programu v jazyce 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
```

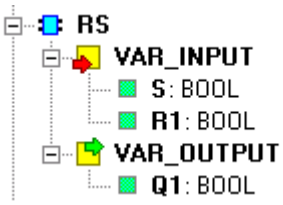


Příklad programu s voláním funkčního bloku R_TRIG :

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

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

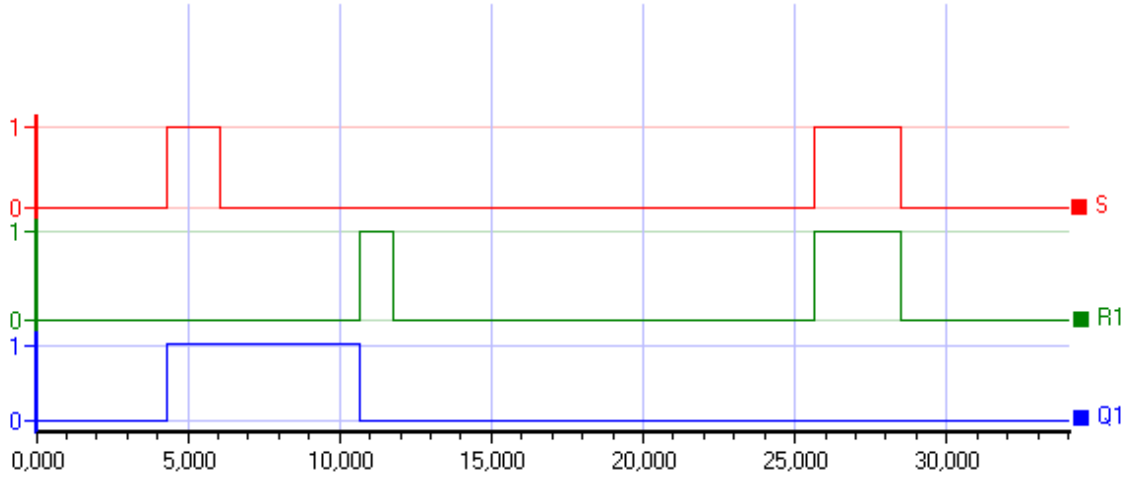
2.6 Funkční blok RS

Funkční blok pro realizaci dvoustavého klopného obodu s dominantní funkcí **RESET**.



Chování funkčního bloku **RS** odpovídá následujícímu programu v jazyce 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
```

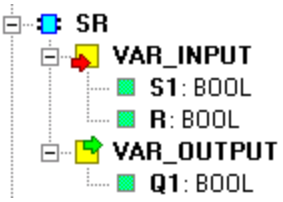


Příklad programu s voláním funkčního bloku **RS** :

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

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

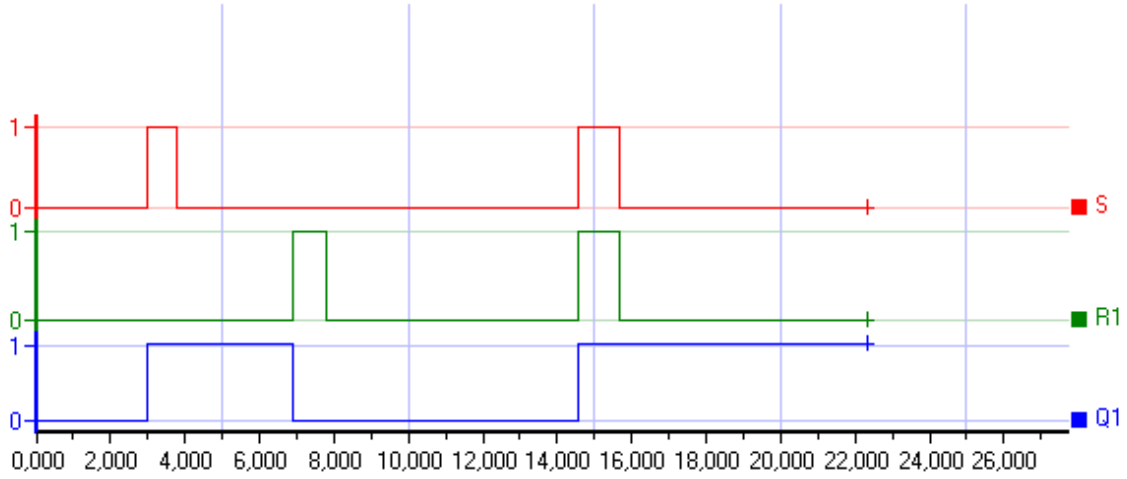
2.7 Funkční blok SR

Funkční blok pro realizaci dvoustavého klopného obodu s dominantní funkcí **SET**.



Chování funkčního bloku **SR** odpovídá následujícímu programu v jazyce **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
```

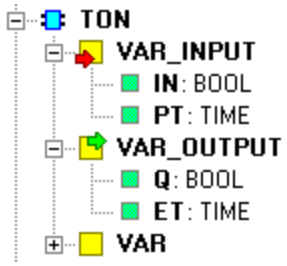


Příklad programu s voláním funkčního bloku **SR** :

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

  inst_SR( S1 := inputSET, R := inputRESET, Q1 => output);
END_PROGRAM
```

2.8 Funkční blok časovače TON

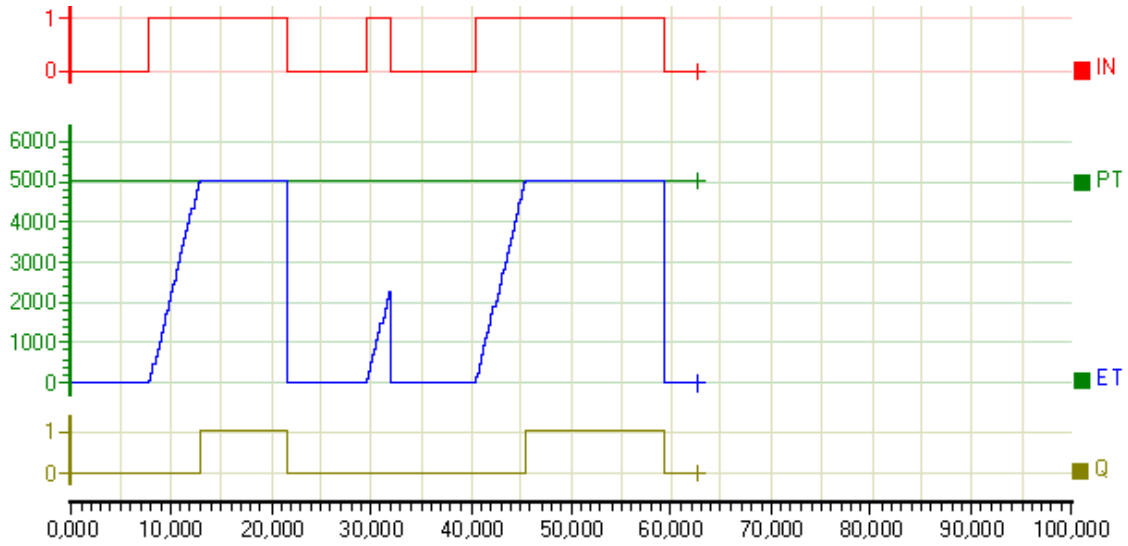
Funkční blok **TON** (Timer On Delay) realizuje prodlevu na náběžnou hranu, tj. vlastně relé se zpožděným přítahem.



Vstupní proměnné :
IN vstup časovače
PT předvolba časovače

Výstupní proměnné :
Q výstup časovače
ET aktuální hodnota časovače

Jestliže vstup **IN** je **FALSE**, výstup **Q** je **FALSE** a **ET** má hodnotu **0**. Jakmile vstup **IN** přejde do stavu **TRUE**, aktuální hodnota časovače **ET** se začne zvětšovat a ve chvíli kdy dosáhne předvolby **PT** výstup **Q** se nastaví na hodnotu **TRUE**. Aktuální hodnota časovače se dále nezvětšuje. Chování časovače **TON** vysvětluje následující obrázek.

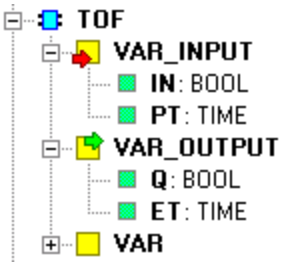


Příklad programu s voláním funkčního bloku **TON** :

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

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

2.9 Funkční blok časovače TOF

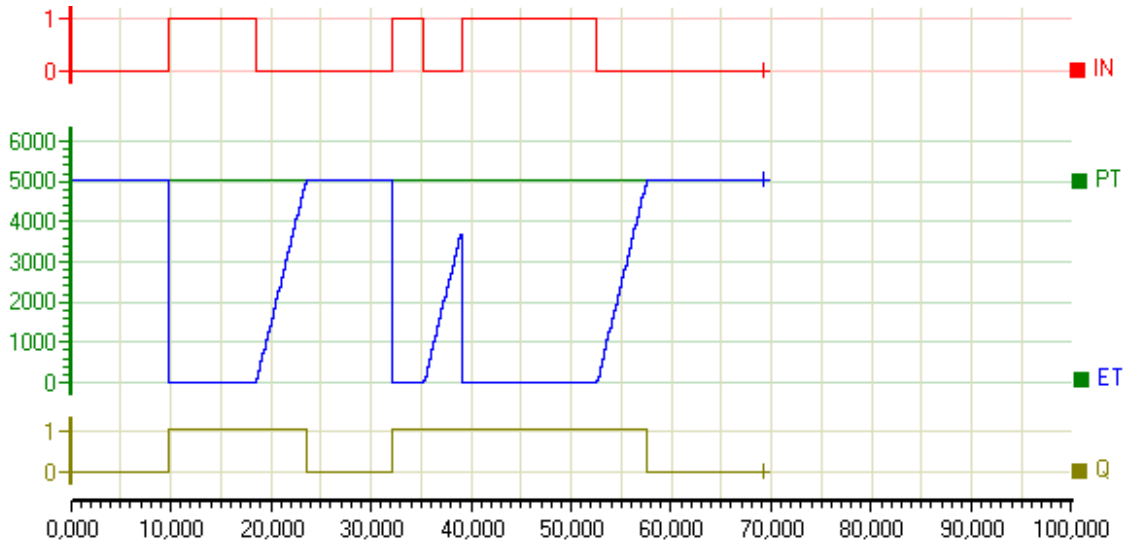
Funkční blok **TOF** (Timer Off Delay) realizuje prodlevu na sestupnou hranu, tj. vlastně relé se zpožděným odpadem.



Vstupní proměnné :
IN vstup časovače
PT předvolba časovače

Výstupní proměnné :
Q výstup časovače
ET aktuální hodnota časovače

Jakmile vstup **IN** přejde do stavu **FALSE**, aktuální hodnota časovače **ET** se začne zvětšovat a ve chvíli kdy dosáhne předvolby **PT** výstup **Q** se nastaví na hodnotu **TRUE**. Aktuální hodnota časovače se pak dále nezvětšuje. Výstup **Q** je ve stavu **FALSE** vždy, když je vstup **IN FALSE** a zároveň aktuální hodnota **ET** je rovna předvolbě **PT**. Chování časovače **TOF** popisuje následující obrázek.

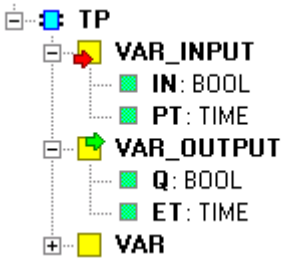


Příklad programu s voláním funkčního bloku **TOF** :

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

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

2.10 Funkční blok časovače TP

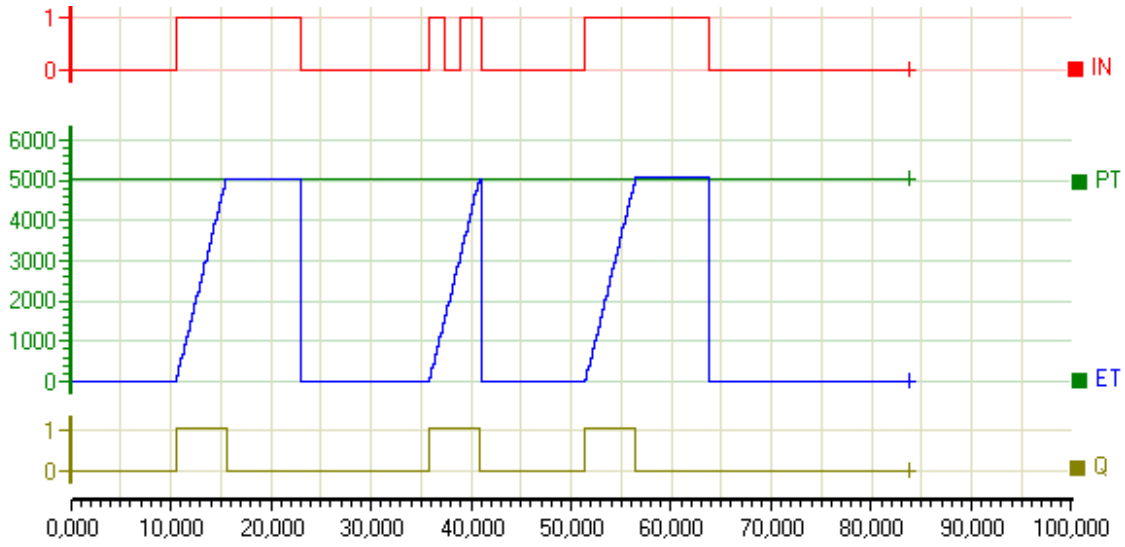
Funkční blok **TP** (Timer Pulse) generuje pulz dané šířky na náběžnounou hranu.



Vstupní proměnné :
IN vstup časovače
PT předvolba časovače

Výstupní proměnné :
Q výstup časovače
ET aktuální hodnota časovače

Jakmile vstup **IN** přejde do stavu **TRUE**, aktuální hodnota časovače **ET** se začne zvětšovat až do chvíle kdy dosáhne předvolby **PT**. Aktuální hodnota časovače se pak dále nezvětšuje. Výstup **Q** je nastaven na hodnotu **TRUE** jestliže byla detekovaná náběžná hrana na vstupu **IN** a zároveň aktuální hodnota **ET** je menší než předvolba **PT**. Jinak má výstup **Q** hodnotu **FALSE**. Chování časovače **TP** popisuje následující obrázek



Příklad programu s voláním funkčního bloku **TP** :

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

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

2.11 Funkce ADD_TIME

Funkce **ADD_TIME** sečte dvě vstupní proměnné typu **TIME**. Návratová hodnota funkce je typu **TIME**.

```
PROGRAM Example_ADD_TIME
  VAR
    cas1, cas2, cas3  : TIME;
  END_VAR

  cas1 := TIME#11h12m13s;
  cas2 := ADD_TIME( IN1 := cas1, IN2 := T#1h);  // T#12h12m13.0s
  cas3 := ADD_TIME( T#10m, cas2);               // T#12h22m13.0s
END_PROGRAM
```

2.12 Funkce ADD_TOD_TIME

Funkce **ADD_TOD_TIME** sečte vstupní proměnnou typu **TIME_OF_DAY** s proměnnou typu **TIME.** Návratová hodnota funkce je typu **TIME_OF_DAY**.

```
PROGRAM Example_ADD_TOD_TIME
  VAR
    cas1    : TIME_OF_DAY := TOD#11:38:52.35;
    cas2    : TIME        := T#15:22:11.120;
    cas3    : TIME_OF_DAY;
  END_VAR

  cas3 := ADD_TOD_TIME( IN1 := cas1, IN2 := cas2);   // 27:01:03.155
END_PROGRAM
```

2.13 Funkce ADD_DT_TIME

Funkce **ADD_DT_TIME** sečte vstupní proměnnou typu **DATE_AND_TIME** s proměnnou typu **TIME.** Návratová hodnota funkce je typu **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;
    suma        : DATE_AND_TIME;
  END_VAR

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

2.14 Funkce SUB_TIME

Funkce **SUB_TIME** odečte dvě vstupní proměnné typu **TIME**. Návratová hodnota funkce je typu **TIME**.

```
PROGRAM Example_SUB_TIME
  VAR
    cas1, cas2, cas3  : TIME;
  END_VAR

  cas1 := TIME#11h12m13s;
  cas2 := SUB_TIME( IN1 := cas1, IN2 := T#1h);  // T#10h12m13.0s
  cas3 := SUB_TIME( cas2, T#10m);               // T#10h02m13.0s
END_PROGRAM
```

2.15 Funkce SUB_DATE_DATE

Funkce **SUB_DATE_DATE** odečte dvě vstupní proměnné typu **DATE**. Návratová hodnota funkce je typu **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 Funkce SUB_TOD_TIME

Funkce **SUB_TOD_TIME** odečte vstupní proměnnou typu **TIME** od proměnné typu **TIME_OF_DAY.** Návratová hodnota funkce je typu **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 Funkce SUB_TOD_TOD

Funkce **SUB_TOD_TOD** odečte vstupní proměnnou typu **TIME_OF_DAY** od proměnné typu **TIME_OF_DAY.** Návratová hodnota funkce je typu **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 Funkce SUB_DT_TIME

Funkce **SUB_DT_TIME** odečte vstupní proměnnou typu **TIME** od proměnné typu **DATE_AND_TIME.** Návratová hodnota funkce je typu **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 Funkce SUB_DT_DT

Funkce **SUB_DT_DT** odečte vstupní proměnnou typu **DATE_AND_TIME** od proměnné typu **DATE_AND_TIME.** Návratová hodnota funkce je typu **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 Funkce CONCAT_DATE_TOD

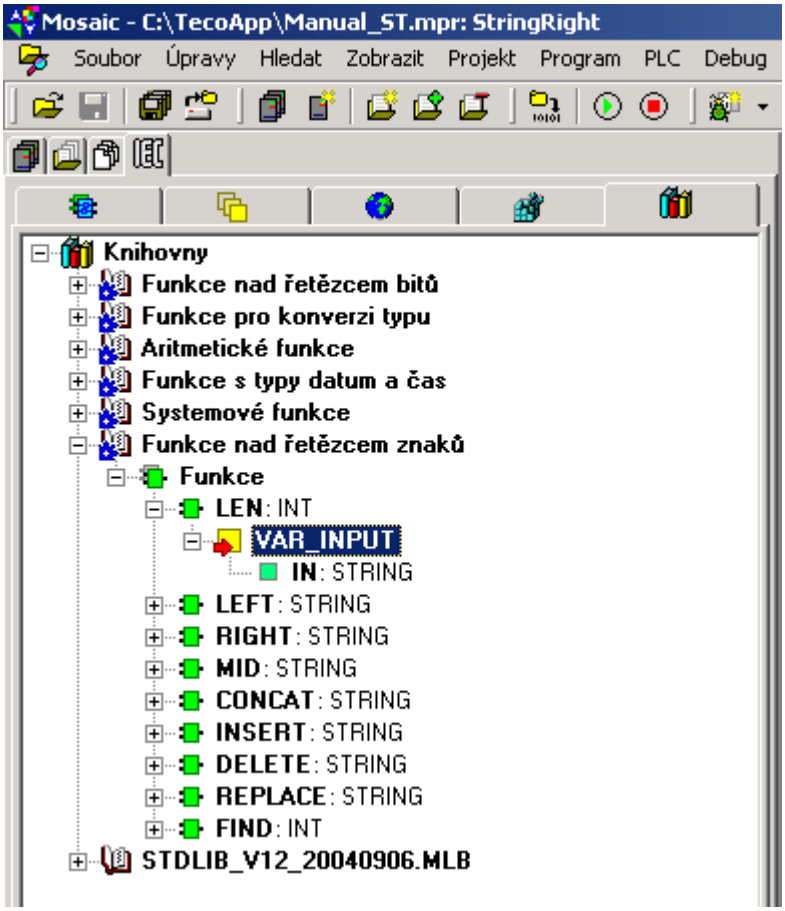
Funkce **CONCAT_DATE_TOD** sečte vstupní proměnnou typu **DATE** s proměnnou typu **TIME_OF_DAY.** Návratová hodnota funkce je typu **DATE_AND_TIME**.

```
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 FUNKCE NAD ŘETĚZCEM ZNAKŮ

Tato knihovna obsahuje funkce pro práci s řetězci znaků (datový typ **STRING**). Obsahuje funkci pro zjištění délky řetězce **LEN**, funkce pro vybírání části řetězce **LEFT**, **RIGHT** a **MID**, funkci pro spojování řetězců **CONCAT**, funkci pro vkládání řetězce do řetězce **INSERT**, funkci pro vypouštění části řetězce **DELETE**, funkci pro náhradu části řetězce jiným řetězcem **REPLACE** a konečně funkci pro nalezení pozice řetězce v jiném řetězci **FIND**. Následují obrázek ukazuje začlenění funkcí nad řetězcem znaků do knihovny v prostředí Mosaic.

Implementace datového typu **STRING** v systémech Tecomat odpovídá řetězcům v jazyce C.



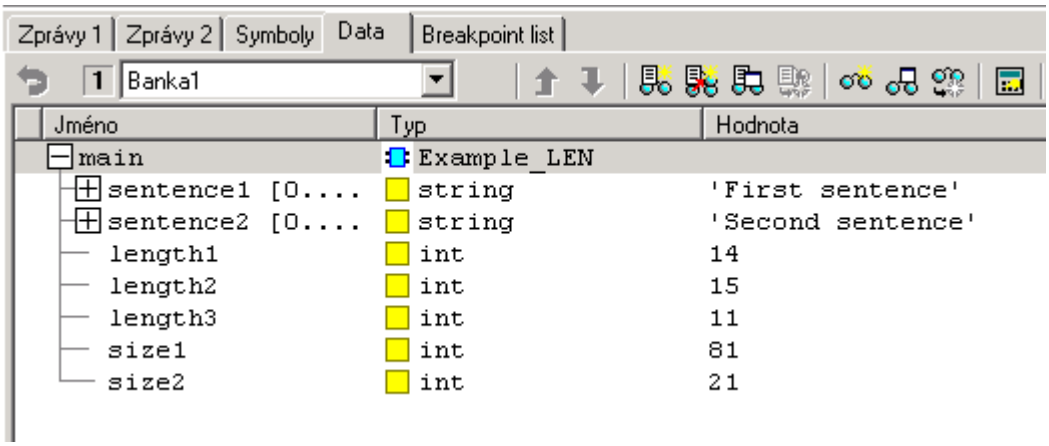
3.1 Funkce LEN

Funkce vrátí délku vstupního řetězce **IN**. Délka řetězce odpovídá aktuálnímu počtu znaků v řetězci. Návratová hodnota je typu **INT**.

Vstupní proměnné :
IN znakový řetězec
Návratová hodnota : délka řetězce

Použití funkce **LEN** ukazuje následující příklad. V souvislosti s délkou řetězce je dobré si uvědomit, že aktuální délka řetězce ve většině případů neodpovídá velikosti proměnné, ve které je řetězec uložen. Velikost proměnné typu **STRING** je daná deklarací proměnné a lze jí zjistit pomocí funkce **SIZEOF** zatímco délka řetězce v proměnné je daná počtem ASCII znaků v řetězci a zjišťuje se pomocí funkce **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
```



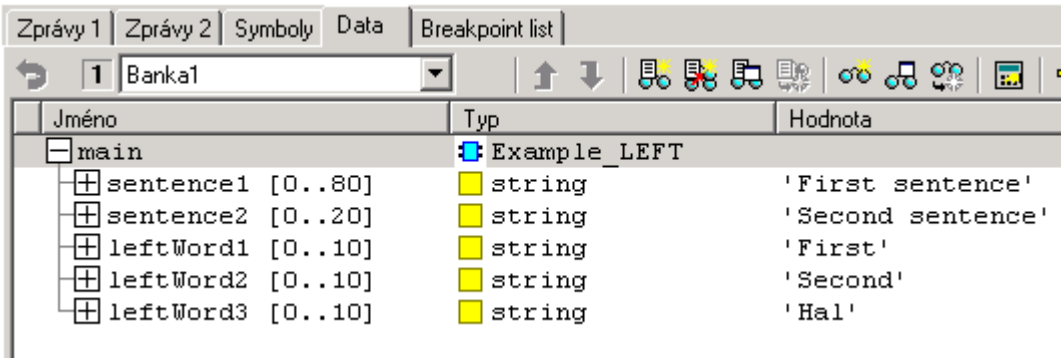
3.2 Funkce LEFT

Funkce **LEFT** vrátí **L** znaků ze vstupního řetězce **IN**. Znaky jsou vráceny zleva, tj. od začátku vstupního řetězce.

Vstupní proměnné :
IN vstupní řetězec
L počet znaků
Návratová hodnota : řetězec

Použití funkce **LEFT** ukazuje následující příklad.

```
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
```



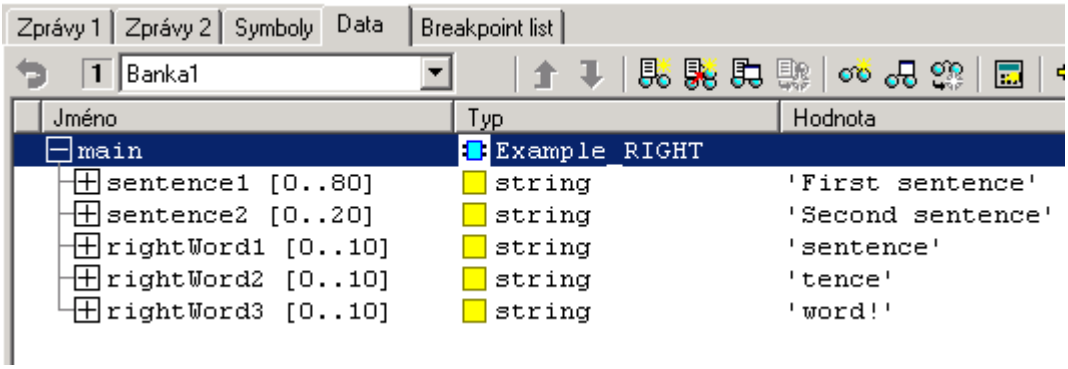
3.3 Funkce RIGHT

Funkce **RIGHT** vrátí **L** znaků ze vstupního řetězce **IN**. Znaky jsou vráceny zprava, tj. od konce vstupního řetězce.

Vstupní proměnné :
IN vstupní řetězec
L pocet znaků
Návratová hodnota : řetězec

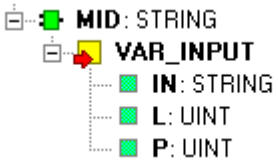
Použití funkce **RIGHT** ukazuje následující příklad.

```
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 Funkce MID

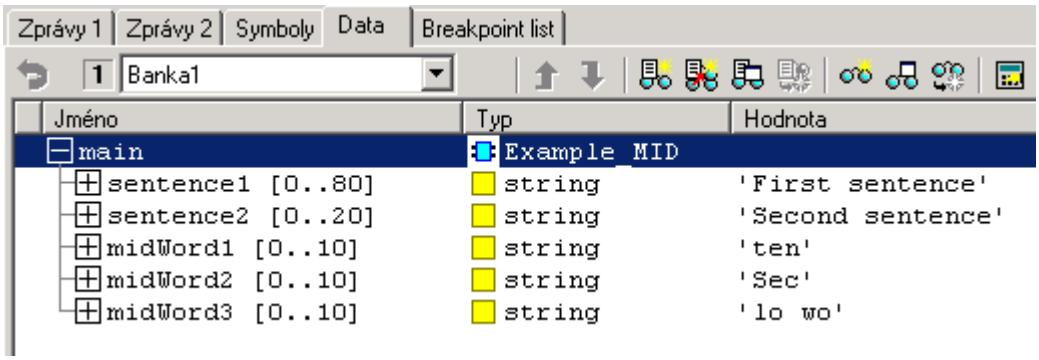
Funkce **MID** vrátí **L** znaků ze vstupního řetězce **IN**. Znaky jsou vráceny od pozice **P** ve vstupním řetězci. První znak v řetězci má pozici 1.



Vstupní proměnné :
IN znakový řetězec
L pocet znaků
P pozice ve vstupním řetězci
Návratová hodnota : řetězec

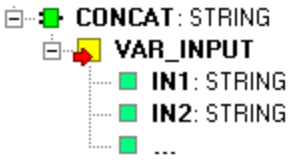
Použití funkce **MID** ukazuje následující příklad.

```
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 Funkce CONCAT

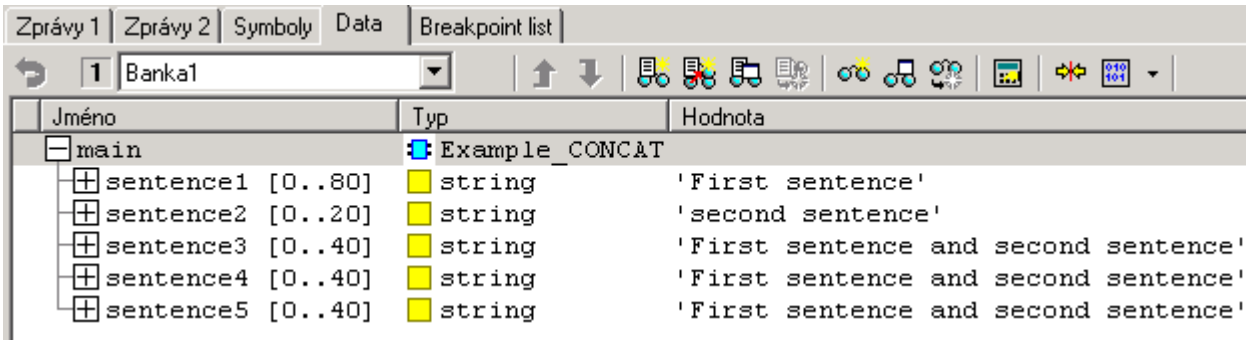
Funkce **CONCAT** sloučí několik řetězců do výstupního řetězce.



Vstupní proměnné :
IN1 znakový řetězec
IN2 znakový řetězec
Návratová hodnota : řetězec

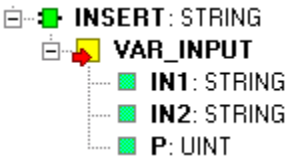
Použití funkce **CONCAT** ukazuje následující příklad. Volání funkce je možné provádět buď klasicky jménem funkce nebo pomocí přetíženého operátoru "**+**". Funkce **CONCAT** je rozšiřitelná, takže může mít různý počet vstupních řetězců.

```
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 Funkce INSERT

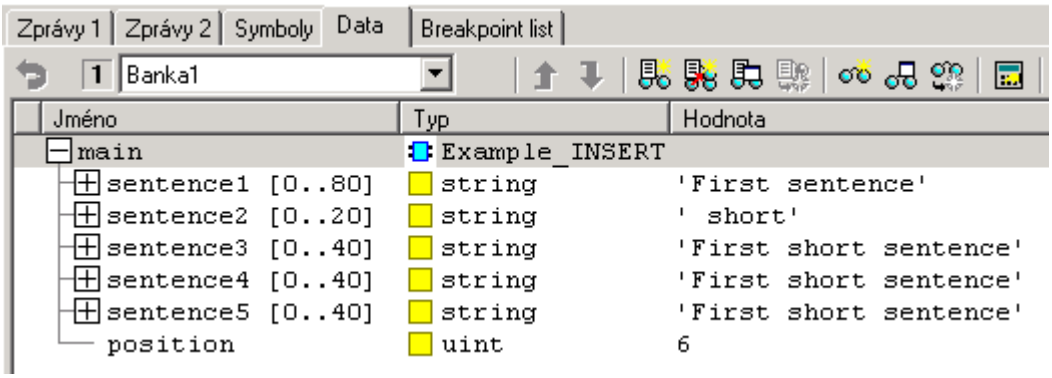
Funkce **INSERT** vloží řetězec **IN2** do řetězce **IN1** na pozici **P**. Takto vytvořený řetězec je vrácen jako návratová hodnota funkce.



Vstupní proměnné :
IN1 vstupní řetězec
IN2 vkládaný řetězec
P pozice ve vstupním řetězci
Návratová hodnota : řetězec

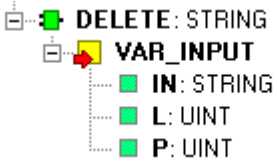
Použití funkce **INSERT** ukazuje následující příklad.

```
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 Funkce DELETE

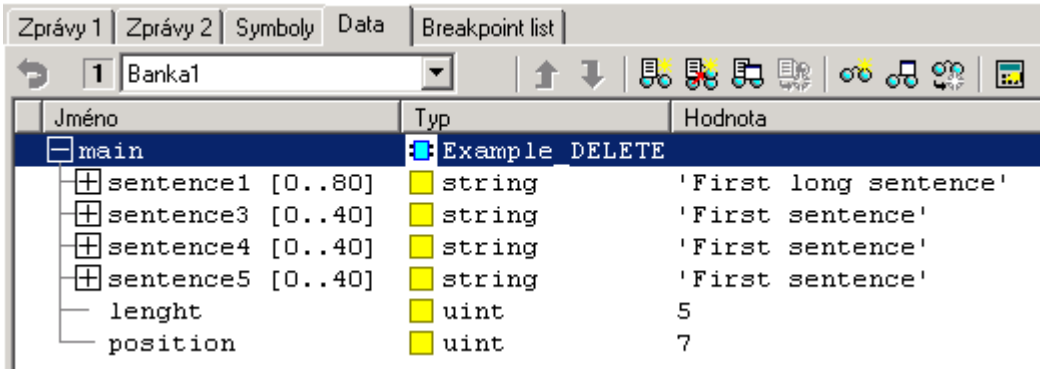
Funkce **DELETE** vypustí počet znaků **L** ze vstupního řetězce **IN**. Znaky jsou vypuštěny od pozice **P**. Takto vytvořený řetězec je vrácen jako návratová hodnota funkce.



Vstupní proměnné :
IN vstupní řetězec
L pocet vypouštěných znaků
P pozice, od které se znaky vypustí
Návratová hodnota : řetězec

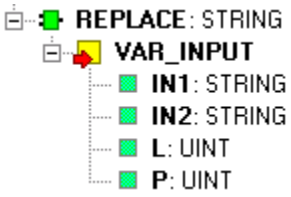
Použití funkce **DELETE** ukazuje následující příklad.

```
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 Funkce REPLACE

Funkce **REPLACE** nejprve vypustí **L** znaků ze vstupního řetězce **IN1**. Znaky jsou vypuštěny od pozice **P**. Poté se na stejnou pozici **P** vloží řetězec **IN2**. Takto vytvořený řetězec je vrácen jako návratová hodnota funkce.



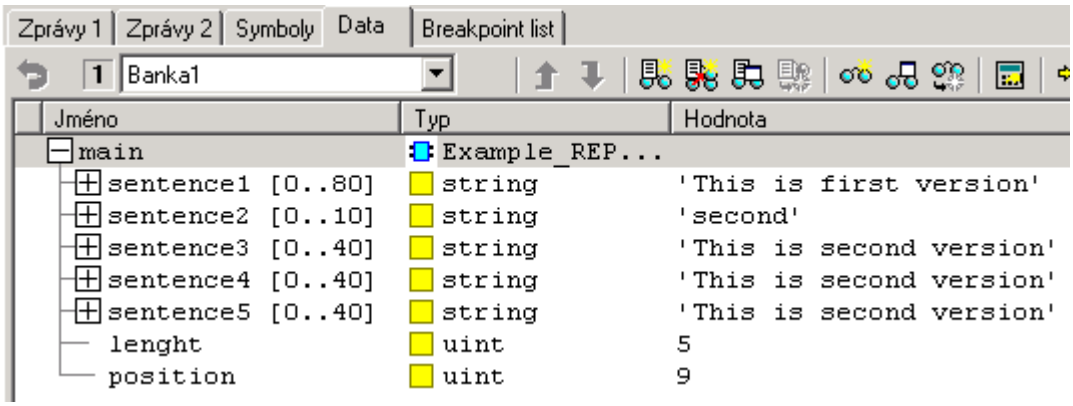
Vstupní proměnné :

IN1	vstupní řetězec
IN2	vkládaný řetězec
L	pocet vypouštěných znaků
P	pozice ve vstupním řetězci

Návratová hodnota : řetězec

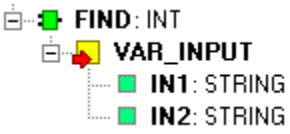
Použití funkce **REPLACE** ukazuje následující příklad.

```
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 Funkce FIND

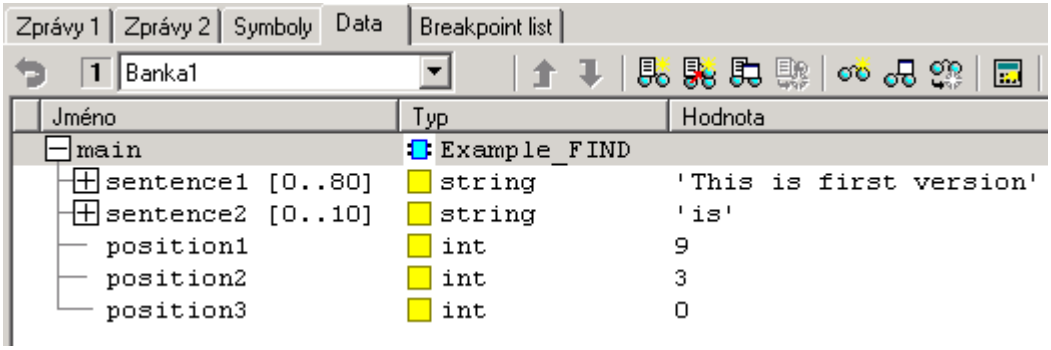
Funkce **FIND** vrátí pozici řetězce **IN2** ve vstupním řetězci **IN1**. Pokud pokud řetězec **IN2** není ve vstupním řetězci **IN1** obsažen, funkce vrátí **0**. Pokud je řetězec **IN2** obsažen v řetězci **IN1** vícekrát, funkce **FIND** vrátí pozici prvního výskytu. Při vyhledávání se berou v úvahu velká a malá písmena. Návratová hodnota funkce je typu **INT**.



Vstupní proměnné :
IN1 vstupní řetězec
IN2 hledaný řetězec
Návratová hodnota : pozice IN2 v řetězci IN1

Použití funkce **FIND** ukazuje následující příklad.

```
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
```

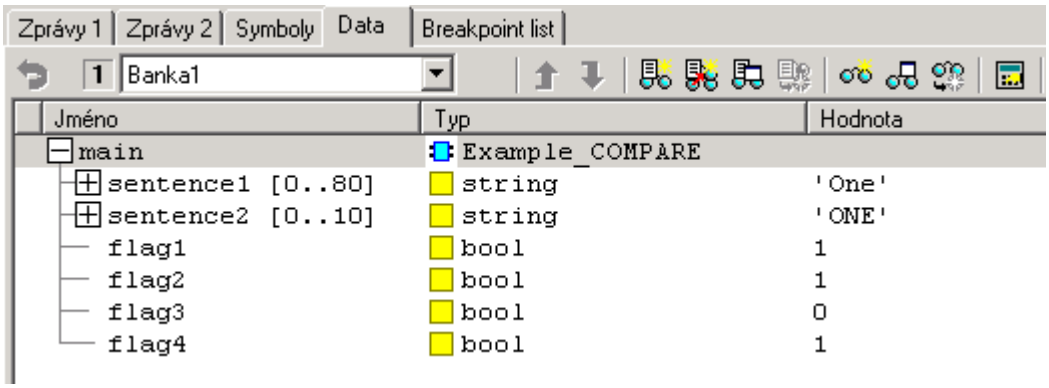


3.10 Funkce porovnání řetězců

Standardní funkce pro porovnání (větší, menší, větší nebo rovno, ...) jsou přetížené i pro typ **STRING**, takže řetězce lze mezi sebou porovnávat. Řetězce jsou porovnávány znak po znaku, velká a malá písmena se rozlišují. Z pohledu porovnání mají malá písmena "větší hodnotu", neboť ASCII kód malých písmen je větší než velkých písmen.

Použití funkcí pro porovnání řetězců ukazuje následující příklad.

```
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 FUNKCE PRO KONVERZI TYPU

Tyto funkce slouží pro konverzi hodnot mezi elementárními datovými typy.

4.1 Funkce ANY_TO_BOOL

Konverze proměnné libovolného elementárního datového typu na typ **BOOL**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_BOOL** je hodnota **TRUE**, jestliže vstupem funkce je nenulová hodnota. Pokud je vstupem hodnota **0**, výsledkem konverze je hodnota **FALSE**.

V případě funkce **STRING_TO_BOOL** je výsledkem hodnota **TRUE**, pokud je vstupem řetězec "**true**" (velikost písmen nehraje roli). V ostaních případech je výsledkem konverze hodnota **FALSE**.

Použití konverze **..._TO_BOOL** ukazuje následující příklad:

```
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 Funkce ANY_TO_SINT

Konverze proměnné libovolného elementárního datového typu na typ **SINT**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_SINT** je číslo v rozsahu <-128, +127>. Pokud konvertujeme z většího datového typu do menšího (např. **DINT_TO_SINT**) riskujeme ztrátu informace v případě, že je vstupní hodnota mimo uvedený rozsah.

Při konverzi z datových typů **REAL** a **LREAL** je vstupní hodnota nejprve zaokrouhlena na nejbližší celé číslo a poté se provede konverze.

Použití konverze **..._TO_SINT** ukazuje následující příklad:

```
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 Funkce ANY_TO_INT

Konverze proměnné libovolného elementárního datového typu na typ **INT**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_INT** je číslo v rozsahu <–32 768, +32 767>. Pokud konvertujeme z většího datového typu do menšího (např. **DINT_TO_INT**) riskujeme ztrátu informace v případě, že je vstupní hodnota mimo uvedený rozsah.

Při konverzi z datových typů **REAL** a **LREAL** je vstupní hodnota nejprve zaokrouhlena na nejbližší celé číslo a poté se provede konverze.

Použití konverze **..._TO_INT** ukazuje následující příklad:

```
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 Funkce ANY_TO_DINT

Konverze proměnné libovolného elementárního datového typu na typ **DINT**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_DINT** je číslo v rozsahu <–2 147 483 648, +2 147 483 647>.

Při konverzi z datových typů **REAL** a **LREAL** je vstupní hodnota nejprve zaokrouhlena na nejbližší celé číslo a poté se provede konverze.

Použití konverze **..._TO_DINT** ukazuje následující příklad:

```
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 Funkce ANY_TO_USINT

Konverze proměnné libovolného elementárního datového typu na typ **USINT**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_USINT** je číslo v rozsahu < 0, 255>. Pokud konvertujeme z většího datového typu do menšího (např. **UDINT_TO_USINT**) riskujeme ztrátu informace v případě, že je vstupní hodnota mimo uvedený rozsah.

Při konverzi z datových typů **REAL** a **LREAL** je vstupní hodnota nejprve zaokrouhlena na nejbližší celé číslo a poté se provede konverze.

Použití konverze **..._TO_USINT** ukazuje následující příklad:

```
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 Funkce ANY_TO_UINT

Konverze proměnné libovolného elementárního datového typu na typ **UINT**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_UINT** je číslo v rozsahu < 0, +65535>. Pokud konvertujeme z většího datového typu do menšího (např. **UDINT_TO_UINT**) riskujeme ztrátu informace v případě, že je vstupní hodnota mimo uvedený rozsah.

Při konverzi z datových typů **REAL** a **LREAL** je vstupní hodnota nejprve zaokrouhlena na nejbližší celé číslo a poté se provede konverze.

Použití konverze **..._TO_UINT** ukazuje následující příklad:

```
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 Funkce ANY_TO_ UDINT

Konverze proměnné libovolného elementárního datového typu na typ **UDINT**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_UDINT** je číslo v rozsahu <0, +4 294 967 295>. Pokud je konvertovaná hodnota mimo tento rozsah (např. při konverzi **REAL_TO_UDINT** nebo v případě záporných čísel) není výsledek konverze definován.

Při konverzi z datových typů **REAL** a **LREAL** je vstupní hodnota nejprve zaokrouhlena na nejbližší celé číslo a poté se provede konverze.

Použití konverze **..._TO_UDINT** ukazuje následující příklad:

```
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 Funkce ANY_TO_ REAL

Konverze proměnné libovolného elementárního datového typu na typ **REAL**.

Do této skupiny funkcí patří následující funkce:
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

Použití konverze **..._TO_REAL** ukazuje následující příklad:

```
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 Funkce ANY_TO_ LREAL

Konverze proměnné libovolného elementárního datového typu na typ **LREAL**.

Do této skupiny funkcí patří následující funkce:
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

Použití konverze **..._TO_LREAL** ukazuje následující příklad:

```
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 Funkce ANY_TO_ STRING

Konverze proměnné libovolného elementárního datového typu na typ **STRING**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_STRING** je textový řetězec.

Výsledkem konverze **ANY_NUM** na typ **STRING** je řetězec, obsahující číslo v desítkové soustavě. Znaménko + se u kladných výsledků neuvádí. Případná desetinná část čísla je oddělena desetinnou tečkou.

U konverzí času a datumu začíná výsledný textový řetězec zkratkou převáděného datového typu (např. u konverze **TIME_TO_STRING** bude výsledný řetězec začínat '**T#**...'). Pak následuje časový údaj ve formátu, který odpovídá časovým literálům při inicializaci časových proměnných.

Použití konverze **..._TO_STRING** ukazuje následující příklad:

```
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 Funkce ANY_TO_ TIME

Konverze proměnné libovolného elementárního datového typu na typ **TIME**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_TIME** je číslo, které udává počet milisekund.

Při konverzi z typu **STRING** musí vstupní řetězec odpovídat literálu **TIME**.

Použití konverze **..._TO_TIME** ukazuje následující příklad:

```
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 Funkce ANY_TO_ TIME_OF_DAY

Konverze proměnné libovolného elementárního datového typu na typ **TIME_OF_DAY**. P|ro tento datový typ je možno také používat zkrácené označení **TOD**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_TIME_OF_DAY** je číslo, které udává počet milisekund.

Při konverzi z typu **STRING** musí vstupní řetězec odpovídat literálu **TOD**.

Použití konverze **..._TO_TIME_OF_DAY** ukazuje následující příklad:

```
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 Funkce ANY_TO_DATE

Konverze proměnné libovolného elementárního datového typu na typ **DATE**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_DATE** je číslo, jehož celá část udává počet sekund od 00:00:00 1.1.1970.

Při konverzi z typu **STRING** musí vstupní řetězec odpovídat literálu **DATE**.

Použití konverze **..._TO_DATE** ukazuje následující příklad:

```
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 Funkce ANY_TO_ DATE_AND_TIME

Konverze proměnné libovolného elementárního datového typu na typ **DATE_AND_TIME**.

Do této skupiny funkcí patří následující funkce:
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

Výsledkem konverze **..._TO_DATE_AND_TIME** je číslo, jehož celá část udává počet sekund od 00:00:00 1.1.1970. Desetinná část čísla pak reprezentuje milisekundy.

Při konverzi z typu **STRING** musí vstupní řetězec odpovídat literálu **DATE_AND_TIME**.

Použití konverze **..._TO_DATE_AND_TIME** ukazuje následující příklad:

```
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 ARITMETICKÉ FUNKCE

5.1 Funkce ABS Absolutní hodnota

Funkce **ABS** vrátí absolutní hodnotu vstupního parametru. Vstupní parametr může být typu **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 Funkce SQRT Odmocnina

Funkce **SQRT** vrátí druhou odmocninu vstupního parametru. Vstupní parametr nesmí být záporný.

```
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 Funkce LN Přirozený logaritmus

Funkce **LN** vrátí přirozený logaritmus vstupního parametru.

```
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 Funkce LOG Desítkový logaritmus

Funkce **LOG** vrátí desítkový logaritmus vstupního parametru.

```
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 Funkce EXP Přirozená exponenciální funkce

Funkce **EXP** vrátí hodnotu e^x, kde x je vstupní parametr.

```
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 Funkce SIN Sinus vstupního úhlu

Funkce **SIN** vrátí sinus vstupního parametru, zadaného v radiánech. Vstupní parametr musí být v intervalu $< -\pi/2,\ \pi/2 >$.

```
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 Funkce COS Kosinus vstupního úhlu

Funkce **COS** vrátí cosinus vstupního parametru, zadaného v radiánech. Vstupní parametr musí být v intervalu < -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 Funkce TAN Tangens vstupního úhlu

Funkce **TAN** vrátí tangens vstupního parametru, zadaného v radiánech. Vstupní parametr musí být v intervalu < -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 Funkce ASIN Arcus sinus

Funkce **ASIN** vrátí arc sinus vstupního parametru. Vstupní parametr musí být v intervalu < -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 Funkce ACOS Arcus kosinus

Funkce **ACOS** vrátí arc cosinus vstupního parametru. Vstupní parametr musí být v intervalu < -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 Funkce ATAN Arcus tangens

Funkce **ATAN** vrátí arc tangens vstupního parametru.

```
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
```

OBSAH