Informační a řídicí systémy I.
Programování PLC III. – IEC 61131-3

Pavel Balda
ZČU v Plzni, FAV, KKY

Osnova přednášky

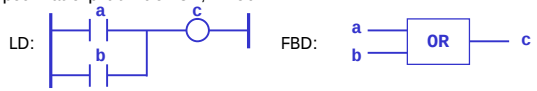
- Schémata složená z funkčních bloků
- Organizace programů
  - Funkce
  - Funkční bloky
  - Programy
  - Úlohy (tasks)

Schémata z funkčních bloků (FBD)

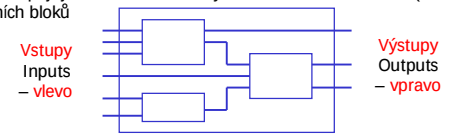
- Schémata z funkčních bloků (Function Block Diagrams, FBD) jsou grafickým programovacím jazykem
  - Interpretují tok signálu mezi jednotlivými prvky schématu
  - Analogické k toku signálu v elektronických obvodech
  - Popisují chování funkcí, funkčních bloků a programů jako množiny propojených grafických bloků (funkcí a funkčních bloků)
  - FBD lze použít pro detailní popis podmínek přechodů a akcí v SFC

Propojení prvků ve schématu FBD (1/2)

- Jednotlivé prvky schémat jsou propojeny lomenými čarami s následujícími vlastnostmi
  - Čáry znázorňují tok signálu ve schématu
  - Výstupy bloků se nemohou přímo spojovat („zkratovat“)
  - Tj. není dovoleno spojování výstupů jako v jazyku LD. Místo něj je třeba používat explicitní blok OR, viz obr.



- Bloky se spojují do schémat složených z elementárních bloků (funkcí nebo funkčních bloků

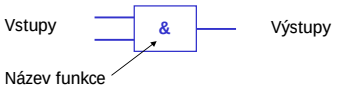


Propojení prvků ve schématu FBD (2/2)

- Vstupní proměnné programu ve FBD
  - Musí být připojeny ke vstupům bloků
  - Typ každé proměnné musí být odpovídat typu připojeného vstupu
  - Vstupem FBD může být konstantní výraz, vnitřní, vstupní nebo výstupní proměnná
- Výstupní proměnné programu ve FBD
  - Musí být připojeny k výstupům bloků
  - Typ každé proměnné musí být odpovídat typu připojeného výstupu
  - Výstupem FBD může být vnitřní nebo výstupní proměnná
- Jednotlivá propojovací čára může být použita k propojení:
  - Vstupní proměnné a vstupu bloku
  - Výstupu bloku a vstup jiného bloku
  - Výstupu bloku a výstupní proměnné
- Propojení má následující vlastnosti:
  - Je orientované, ve směru zleva doprava
  - Levé a pravé zakončení propojovací čáry je stejného typu
  - Lze použít několik pravých zakončení, která značí, že informace z levého zakončení je přenášena na několik dalších zakončení (všechny musí být stejného typu)

Schématická značka bloku

- Funkce bloku je vyznačena uvnitř obdélníkového symbolu bloku



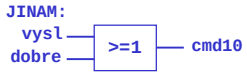
- Propojovací čára může být zakončena symbolem negace
  - Malé kolečko na pravém konci propojovací čáry, viz příklad

Ekvivalentně v ST:

```
c := a AND NOT(b);
```


Příkaz JUMP

- Pro řízení pořadí zpracování sítě FBD lze dále používat návěští a skok na něj pomocí příkazu **JUMP**
  - Při splnění Booleovské podmínky (**TRUE**) se vykonávání programu přenese za symbol návěští, viz příklad

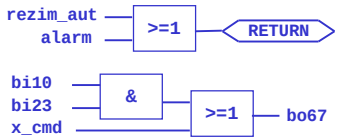


Ekvivalentně v IL (v ST nejsou skoky!):

```
        LD   rucne
        AND  b1
        JMPC JINAM
        ...  ...
JINAM:
        LD   vysl
        OR   dobre
        ST   cmd10
```


Příkaz RETURN a vyhodnocování FBD

- Příkaz **RETURN** umožňuje podmíněně ukončit vyhodnocování dané POU
  - Nabude-li připojená Booleovská hodnota **TRUE**, ukončí se daný program a zbývající část se nevykoná



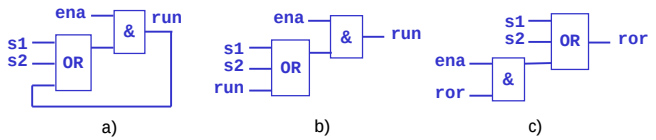
Ekvivalentně v ST:

```
IF rezim_aut OR alarm THEN
  RETURN;
END_IF;
bo67 := (bi10 AND bi23)
   OR x_cmd;
```

- Pořadí vyhodnocování sítě FBD se řídí následujícím pravidlem:
  - Vyhodnocení sítě je ukončeno před započetím vyhodnocování jiné sítě, využívající jeden nebo několik výstupů předchozí vyhodnocované sítě
  - Pořadí vyhodnocování dané sítě je implementačně závislé (!!!), obvykle shora dolů a zleva doprava

Zpětnovazební spojení bloků ve FBD

- Ve schématu existuje zpětná vazba (zpětnovazební cesta, feedback path), pokud je výstup nějaké funkce nebo funkčního bloku použit jako vstup funkce nebo funkčního bloku, který jej předchází (je dříve vyhodnocován)
  - Explicitní zpětná vazba – propojení výstupu daného bloku se vstupem dříve vyhodnocovaného bloku (obr. a)
  - Implicitní zpětná vazba – případ, kdy je výstup přiřazen do proměnné, použité pro vstup dříve vyhodnocovaného bloku (obr. b, c)
  - V případě explicitní zpětné vazby (obr. a)) není jednoznačně určeno pořadí, jak ji vykonávat (zda jako případ b nebo c)
  - Poznámka: v LD lze používat jen implicitní zpětnou vazbu



Programové organizační jednotky (POU)

- Programová organizační jednotka (Program Organization Unit, POU)
  - Základní stavební jednotka programového vybavení podle IEC 61131-3
  - Existují 3 typy POU: funkce, funkční blok a program
  - Struktura a chování POU je definována v deklaraci typu
  - POU by neměly rekurzivně volat POU téhož typu – chování rekurzivního volání v reálném čase nelze předvídat
  - Umožňují sdílet dříve vyvinutý kód v dalších aplikacích (software reuse) na „makro“ úrovni (programy) i na „mikro“ úrovni (funkce, funkční bloky)

POU: Funkce

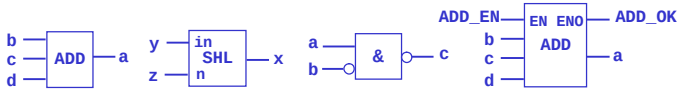
- Funkce je definována jako POU, která po vykonání vrací jednu návratovou hodnotu (result) a libovolně mnoho dalších výstupních hodnot (**VAR_OUTPUT** a **VAR_IN_OUT**)
  - Návratová hodnota může být vícehodnotová (tj. pole nebo struktura)
  - Funkce lze volat v textových jazycích (ST, IL) jako operand ve výrazu
  - Funkce nemají vnitřní stav, tj. zavolání funkce se stejnými argumenty (vstupními proměnnými **VAR_INPUT** a vstupně-výstupními proměnnými **VAR_OUTPUT**) vrací vždy stejné hodnoty (výstupní proměnné **VAR_OUTPUT**, vstupně-výstupní proměnné **VAR_IN_OUT** a výsledek)
  - Jakákoliv deklarovaná funkce může být od okamžiku deklarace použita v jiných POU
- Funkce a jejich volání mohou být reprezentovány graficky (ve FBD a LD) nebo textově (v ST a IL), viz dále

Textová reprezentace funkcí

- V textových jazycích lze předávat vstupní hodnoty dvěma způsoby:
  - Pomocí formálního seznamu parametrů (*)
    - Vstupním proměnným se přiřazují hodnoty
    - Nezáleží na pořadí vstupních proměnných
    - Libovolný počet použitých vstupních proměnných; neuvedené parametry mají implicitní hodnoty
    - `A := LIMIT(EN:=COND, IN:=B, MX:=5, ENO=>TEMPL);`
  - Pomocí neformálního seznamu parametrů
    - Vstupním proměnným se hodnoty nepřiřazují
    - Pevné pořadí vstupů
    - Pevný počet vstupních proměnných
    - `A := LIMIT(1, B, 5);`
      `(* stejné jako LIMIT(EN:=TRUE, MN:=1, IN:=B, MX:=5); *)`
- Přiřazení výstupních hodnot buď není použito nebo se přiřazuje do proměnných pomocí operátoru **=>**
- Pro přiřazení do **VAR_IN_OUT** hodnot je třeba používat proměnných
- Přiřazení do argumentů **VAR_INPUT** je buď prázdné (viz (*)) nebo se přiřazuje konstanta, proměnná nebo návratová hodnota funkce

Grafická reprezentace funkcí

- Funkce jsou graficky reprezentovány obdélníkem (nebo čtvercem)
- Velikost může záviset na počtu vstupů/výstupů a dalších informací
- Zpracování probíhá zleva (vstupy) doprava (výstupy)
- Název nebo symbol funkce je uvnitř bloku
- Na levé vnitřní straně symbolu bloku mohou ale nemusí být jména parametrů funkce (obr. a,b); nejsou-li uvedena u standardních funkcí (viz dále) jsou jména parametrů **IN1**, **IN2**, ... (v případě jediného parametru je jméno **IN**)
- Argumenty a výsledek se připojují pomocí propojovacích čar (flow lines)
- V místě připojení vstupu/výstupu může být malé kolečko znamenající negaci příslušného vstupu/výstupu (obr. c)
- Lze používat přídavný vstup **EN** a/nebo výstup **ENO**, viz dále. Je-li některý z nich použit, je vždy uveden jako první vstup/výstup shora (obr. d)



a) b) c) d)

Řízení spouštění funkcí

- Pro řízení spouštění funkcí se používají přídavné Booleovské signály – vstup **EN** a výstup **ENO** (mohou být použity oba)
- Obě proměnné jsou implicitně deklarovány jako:
  - `VAR_INPUT  EN: BOOL := 1; END_VAR`
    `VAR_OUTPUT ENO: BOOL; END_VAR`
- Spouštění funkce se při použití těchto proměnných řídí pravidly:
  - Pro **EN=FALSE** (0) při vyvolání funkce se neprovedou operace uvnitř těla a PLC shodí výstup **ENO** na **FALSE**
  - Jinak PLC nastaví **ENO** na **TRUE** (1) a vykonají se operace uvnitř těla funkce, kde se může rovněž nastavovat hodnota **ENO**
  - Vyskytne-li se chyba při provádění těla standardní funkce, je výstup **ENO** shozen na **FALSE**. V případě uživatelských funkcí musí být hodnota **ENO** explicitně přiřazena programátorem
  - Je-li hodnota **ENO=FALSE** je nastavení výstupů funkce (**VAR_OUTPUT**, **VAR_IN_OUT** a výsledku) závislé na implementaci

Textové deklarace funkcí

- Textové deklarace funkcí se skládají z:
  - **FUNCTION** *<jmeno_fn>* **:** *<typ_vysl>*
  - **VAR_INPUT** ... **END_VAR** konstrukce specifikující jména a typy vstupních proměnných
  - **VAR_IN_OUT** ... **END_VAR** a **VAR_OUTPUT** ... **END_VAR** konstrukce specifikující jména a typy vstupně-výstupních a výstupních proměnných
  - **VAR** ... **END_VAR** konstrukce specifikující jména a typy vnitřních proměnných
  - Tělo funkce zapsané v ST nebo IL specifikující operace se vstupními parametry a nastavující vstupně-výstupní a výstupní proměnné a výsledek funkce
  - Ukončující klíčové slovo **END_FUNCTION**
- Sekce **VAR_IN_OUT** a **VAR_OUTPUT** jsou nepovinné

Příklad:

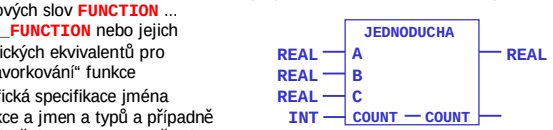
```
FUNCTION JEDNODUCHA : REAL
  (*Spec. externiho rozhrani*)
  VAR_INPUT
    A,B : REAL;
    C : REAL := 1.0;
  END_VAR
  VAR_IN_OUT
    COUNT : INT;
  END_VAR
  VAR COUNTP1 : INT ; END_VAR

  (*Specifikace tela funkce*)
  COUNTP1 := ADD(COUNT,1)
  COUNT := COUNTP1 ;
  JEDNODUCHA := A*B/C;
END_FUNCTION
```


Grafické deklarace funkcí

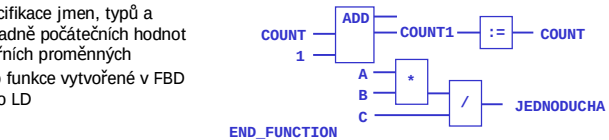
- Grafické deklarace funkcí se skládají z:
  - Klíčových slov **FUNCTION** ... **END_FUNCTION** nebo jejich grafických ekvivalentů pro „uzávorkování“ funkce
  - Grafická specifikace jména funkce a jmen a typů a případně počátečních hodnot proměnných a výsledku
  - Specifikace jmen, typů a případně počátečních hodnot vnitřních proměnných
  - Tělo funkce vytvořené v FBD nebo LD

Příklad:

```
FUNCTION
  (*Spec. externiho rozhrani*)
```




```
(*Specifikace tela funkce*)
```

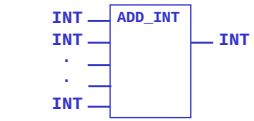



```
END_FUNCTION
```

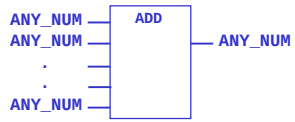
Typové a přetížené funkce

- Funkce mohou být vytvořeny pro:
  - Konkrétní typ, např. **INT** (obr. a)
  - Generický typ, např. **ANY_NUM** (obr. b). Takové funkce se nazývají přetížené

a)



b)



Hierarchie jednoduchých typů

```
ANY
  ANY_DERIVED
  ANY_ELEMENTARY
    ANY_MAGNITUDE
      ANY_NUM
        ANY_REAL
          LREAL, REAL
        ANY_INT
          LINT, DINT,
          INT, SINT
          ULINT, UDINT,
          UINT, USINT
      TIME
    ANY_BIT
      LWORD, DWORD, WORD,
      BYTE, BOOL
    ANY_STRING
      STRING, WSTRING
    ANY_DATE
      DATE_AND_TIME, DATE,
      TIME_OF_DAY
```


Standardní funkce

- Standardní funkce jsou funkce běžně vestavěné do PLC
- Mohou mít proměnný počet vstupů (parametrů)
  - Na každý z přidaných vstupů aplikují stejnou operaci
  - Např. funkce pro sčítání vrací jako výsledek sumu všech svých vstupů
  - Maximální počet vstupů rozšiřitelných funkcí je závislý na implementaci
- Standardní funkce lze rozdělit do následujících skupin:
  - Funkce pro převod typů
  - Numerické funkce
  - Funkce pro práci s bitovými řetězci
  - Funkce pro výběr a porovnávání
  - Funkce pro práci se řetězci
  - Funkce pro práci s časem

Funkce pro převod typů

* – vstupní typ ** – výstupní typ

Konverzní funkce ***_TO_****

- **A := INT_TO_REAL(B);**

Při převodu z reálných typů na celočíselné provádějí zaokrouhlení. Je-li zaokrouhlované číslo přesně uprostřed intervalu, provede se zaokrouhlení k nejbližšímu sudému číslu

Funkce **TRUNC** odřízne z čísla desetinnou část na celé číslo směrem k nule

- **TRUNC(-1.6)** je **-1**, **TRUNC(1.6)** je **1**

Funkce ***_BCD_TO_**** a ***_TO_BCD_**** konvertují data z typů **BYTE**, **WORD**, **DWORD** a **LWORD** na typy **USINT**, **UINT**, **UDINT** a **ULINT** (čísla v BCD formátu jsou uložena v typech bitových řetězců)

- **USINT_TO_BCD_BYTE(25)** je **2#0010_0101**
- **WORD_BCD_TO_UINT(2#0011_0110_1001)** je **369**

Numerické funkce

* – vstupní a výstupní typ

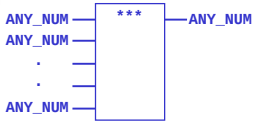
** – jméno funkce

Příklad:

```
A := SIN(B);
```

Jméno	V/V typ	Popis
	Obecné funkce	
ABS	**ANY_NUM**	Absolutní hodnota
SQRT	**ANY_REAL**	Druhá odmocnina
	Logaritmické funkce	
LN	**ANY_REAL**	Přirozený logaritmus
LOG	**ANY_REAL**	Dekadický logaritmus
EXP	**ANY_REAL**	Exponenciální funkce se základem *e*
	Trigonometrické funkce	
SIN	**ANY_REAL**	Sinus se vstupem v radiánech
COS	**ANY_REAL**	Kosinus se vstupem v radiánech
TAN	**ANY_REAL**	Tangens se vstupem v radiánech
ASIN	**ANY_REAL**	Arkus sinus
ACOS	**ANY_REAL**	Arkus kosinus
ATAN	**ANY_REAL**	Arkus tangens

Aritmetické funkce



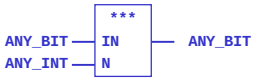
*** – jméno nebo symbol

Příklad:
```
A := ADD(B, C, D);
```
nebo
```
A := B + C + D;
```

Jméno	Symbol	Popis
	Rozšiřitelné aritmetické funkce	
ADD	+	OUT := IN1 + IN2 + … + INn
MUL	*	OUT := IN1 * IN2 * … * INn
	Nerozšiřitelné aritmetické funkce	
SUB	-	OUT := IN1 - IN2
DIV	/	OUT := IN1 / IN2
MOD		OUT := IN1 modulo IN2
EXPT	**	Umocňování: OUT := $IN1^{IN2}$
MOVE	:=	OUT := IN

Standardní funkce pro posouvání bitů

- Standard bit shift functions

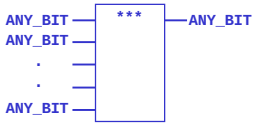


*** – jméno funkce

Příklad:
```
A := SHL(IN:=B, N:=5);
```

Jméno	Popis
SHL	OUT := IN posunut doleva o N bitů, doplněn zprava nulami
SHR	OUT := IN posunut doprava o N bitů, doplněn zleva nulami
ROL	OUT := IN rotován doleva o N bitů, cyklicky
ROR	OUT := IN rotován doprava o N bitů, cyklicky

Standardní bitové Booleovské funkce



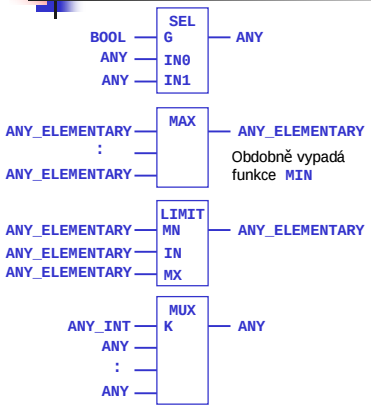
*** – jméno nebo symbol

Příklad:
```
A := AND(B, C, D);
```
nebo
```
A := B & C & D;
```

Jméno	Symbol	Popis
AND	& [1]	OUT := IN1 & IN2 & … & INn
OR	>=1 [2]	OUT := IN1 OR IN2 OR … OR INn
XOR	=2k+1 [2]	OUT := IN1 XOR IN2 XOR … XOR INn
NOT		OUT := NOT IN [3]

[1] Symbol lze užít v textových jazycích ST a IL
[2] Symbol není vhodný jako operátor v textových jazycích ST a IL
[3] Pro negaci se v grafických jazycích mohou používat symboly kolečka na vstupu/výstupu

Standardní funkce pro výběr



SEL – Binární výběr
- **OUT := IN0** když **G=0** jinak **OUT := IN1**
- `A := SEL(G:=0, IN0:=X, IN1:=5);`

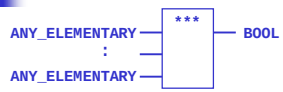
MAX (**MIN**) – Rozšiřitelná funkce max (min)
- **OUT := MAX (IN1, IN2, … , INn)**
- **OUT := MIN (IN1, IN2, … , INn)**
- `A := MAX (B, C, D);`

Obdobně vypadá funkce `MIN`

LIMIT – Omezovač (Limiter)
- **OUT := MIN (MAX(IN, MN), MX)**
- `A := LIMIT(IN:=B, MN:=0, MX:=5);`

MUX – Rozšiřitelný multiplexer
- Podle hodnoty **K** vybírá jeden z **N** vstupů
- `A := MUX(0, B, C, D);`
  je v tomto případě totéž jako `A := B;`

Standardní porovnávací funkce



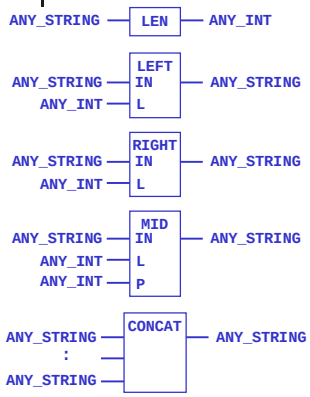
*** – jméno funkce

Příklad:

```
A := GT(B, C, D);  nebo
A := (B>C) & (C>D);
```

Jméno	Symbol	Popis
GT	>	Větší (klesající posloupnost) **OUT := (IN1>IN2) & (IN2>IN3) & … & (INn-1>Inn)**
GE	>=	Větší nebo rovno (monotónní posloupnost) **OUT := (IN1>=IN2) & (IN2>=IN3) & … & (INn-1>=Inn)**
EQ	=	Rovnost: **OUT := (IN1=IN2) & (IN2=IN3) & … & (INn-1=Inn)**
LE	<=	Menší nebo rovno (monotónní posloupnost) **OUT := (IN1<=IN2) & (IN2<=IN3) & … & (INn-1<=Inn)**
LT	<	Menší (rostoucí posloupnost) **OUT := (IN1<IN2) & (IN2<IN3) & … & (INn-1<Inn)**
NE	<>	Nerovnost (nerozšiřitelná): **OUT := (IN1<>IN2)**

Práce se znakovými řetězci (1/2)



LEN – Délka řetězce

- `A := LEN('Ahoj');` je ekvivalent k `A:=4;`

LEFT – **L** znaků nejvíc vlevo z **IN**

- `A := LEFT(IN:='Ahoj', L:=2);` je ekvivalent k `A:= 'Ah';`

RIGHT – **L** znaků nejvíc vpravo z **IN**

- `A := RIGHT(IN:='Ahoj', L:=3);` je ekvivalent k `A:= 'hoj';`

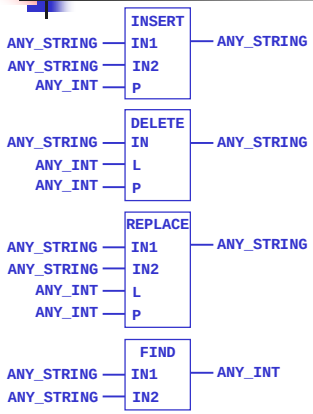
MID – **L** znaků z **IN** od pozice **P**

- `A := MID(IN:='Ahoj', L:=2, P:=2);` je ekvivalent k `A:= 'ho';`

CONCAT – rozšiřitelné spojování řetězců

- `A := CONCAT('AB','CD','E');` je ekvivalent k `A:= 'ABCDE';`

Práce se znakovými řetězci (2/2)



INSERT – Vkládá **IN2** do **IN1** za **P**-tý znak

- `A := INSERT(IN1:='ABC',IN2:='XY', P:=2);` je ekvivalent k `A:='ABXYC';`

DELETE – Odstraňuje **L** znaků z **IN** od pozice **P**

- `A := DELETE(IN:='ABXYC',L:=2,P:=2);` je ekvivalent k `A:='ABC';`

REPLACE – Nahrazuje **L** znaků řetězce **IN1** od pozice **P** znaky z **IN2**

- `A := REPLACE(IN1:='ABCDE',IN2:='X', L:=2,P:=3);` je ekvivalent k `A:='ABXE';`

FIND – Hledá pozici začátku prvního výskytu **IN2** v **IN1**. Není-li nalezena je **OUT:=0**.

- `A := FIND(IN1:='ABCBC', IN2:='BC');` je ekvivalent k `A:=2;`

Funkce pro práci s časem

- Je uvedena jen nová syntaxe, starší nebude dále podporována

Jméno	Symbol	IN1	IN2	OUT
ADD nebo ADD_TIME	+	TIME	TIME	TIME
ADD_TOD_TIME		TIME_OF_DAY	TIME	TIME_OF_DAY
ADD_DT_TIME		DATE_AND_TIME	TIME	DATE_AND_TIME
SUB nebo SUB_TIME	-	TIME	TIME	TIME
SUB_DATE_DATE		DATE	DATE	TIME
SUB_TOD_TIME		TIME_OF_DAY	TIME	TIME_OF_DAY
SUB_TOD_TOD		TIME_OF_DAY	TIME_OF_DAY	TIME
SUB_DT_TIME		DATE_AND_TIME	TIME	DATE_AND_TIME
SUB_DT_DT		DATE_AND_TIME	DATE_AND_TIME	TIME
MULTIME		TIME	ANY_NUM	TIME
DIVTIME		TIME	ANY_NUM	TIME
CONCAT_DATE_TOD		DATE	TIME_OF_DAY	DATE_AND_TIME
DT_TO_TOD	Funkce pro konverzi typů – vybírají z daného typu požadovanou část			
DT_TO_DATE				

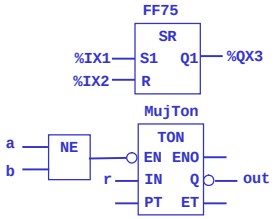
POU: Funkční bloky

- Funkční blok (FB) je POU, který po vykonání vrací jednu nebo několik hodnot
  - Od jednoho bloku může být vytvářeno více instancí (kopií)
  - Každá instance má svůj identifikátor (jméno) a datovou strukturu obsahující její výstupní a interní proměnné a v závislosti na implementaci hodnoty nebo odkazy na vstupní parametry
  - Hodnoty všech výstupů a potřebných interních proměnných zůstávají zachovány (persistence) mezi jednotlivými spuštěními bloku
  - Na rozdíl od funkcí proto nemusí dvě spuštění téhož bloku se stejnými argumenty vést ke stejným výstupům
  - Vně funkčního bloku jsou dostupné pouze jeho vstupy a výstupy, vnitřní proměnné zůstávají skryty
  - Funkční blok je podobný objektu z OOP, není však tak „volný“
- Funkční bloky a jejich volání mohou být reprezentovány graficky (ve FBD a LD) nebo textově (v ST a IL), viz dále

Reprezentace funkčních bloků

- FB lze reprezentovat graficky i textově, viz příklady
- Všechny kombinace čtení a zápisu nejsou přípustné. Není dovoleno:
  - Číst vstupy funkčního bloku mimo funkční blok
  - Zapisovat do vstupů funkčního bloku zevnitř bloku
  - Zapisovat do výstupu funkčního bloku vně funkčního bloku
- Případný vstup **EN** a výstup **ENO** se zpracovává stejně jako u funkcí

Grafická reprezentace ve FBD



Textová reprezentace v ST

```
VAR FF75 : SR; END_VAR   (* Deklarace *)
FF75(S1:=%IX1, R:=%IX2); (* Volání *)
%QX3 := FF75.Q1;     (* Přiřaď výstup *)

VAR
  a,b,out : BOOL;
  MujTON  : TON;
END_VAR
MujTon(EN := NOT(A<>B),
       IN := r,
       NOT Q => out);
```


Deklarace funkčních bloků

- Funční bloky se deklarují obdobně jako funkce textově nebo graficky s následujícími rozdíly:
  - Klíčová slova pro deklaraci FB jsou **FUNCTION_BLOCK** ... **END_FUNCTION_BLOCK**
  - Pro interní a výstupní proměnné bloku lze používat kvalifikátor **RETAIN**
  - Hodnoty proměnných přenášené do funkčního bloku pomocí konstrukce **VAR_EXTERNAL** mohou být v bloku modifikovány
  - Hodnoty výstupů jiných funkčních bloků přenášené do funkčního bloku pomocí konstrukce **VAR_INPUT**, **VAR_IN_OUT** nebo **VAR_EXTERNAL** mohou být v bloku použity, ale nesmí být modifikovány
  - V textových jazycích lze používat kvalifikátory **R_EDGE** a **F_EDGE** pro detekci náběžné a sestupné hrany vstupního signálu

```
VAR_INPUT
   X : BOOL R_EDGE;
   Y : BOOL F_EDGE;
END_VAR
```

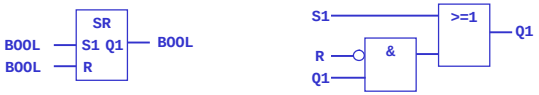

Standardní funkční bloky

- Standardní funkční bloky jsou funkce běžně vestavěné do PLC
- Mohou být přetížené (overloaded) a mohou mít proměnný počet vstupů a výstupů
- Standardní funkce lze rozdělit do následujících skupin:
  - Bistabilní funkční bloky
  - Bloky pro detekci hran
  - Čítačové funkční bloky (čítače)
  - Časovací funkční bloky (časovače)

Bistabilní funkční bloky

Grafická reprezentace Tělo funkčního bloku

SR klopný obvod, dominantní je SET (vstup **S1**)



RS klopný obvod, dominantní je RESET (vstup **R1**)

Funkční bloky pro detekci hran

Grafická reprezentace Definice

R_TRIG – detekce náběžné hrany (rising edge)

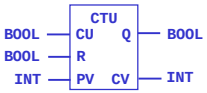

```
FUNCTION_BLOCK R_TRIG
  VAR_INPUT  CLK :BOOL; END_VAR
  VAR_OUTPUT Q   :BOOL; END_VAR
  VAR        M   :BOOL; END_VAR
  Q := CLK AND NOT M;
  M := CLK;
END_FUNCTION_BLOCK
```

F_TRIG – detekce sestupné hrany (falling edge)


```
FUNCTION_BLOCK F_TRIG
  VAR_INPUT  CLK :BOOL; END_VAR
  VAR_OUTPUT Q   :BOOL; END_VAR
  VAR        M   :BOOL; END_VAR
  Q := NOT CLK AND NOT M;
  M := NOT CLK;
END_FUNCTION_BLOCK
```

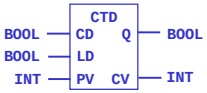

Čítačové funkční bloky

CTU – čítač směrem vzhůru (Up-counter)



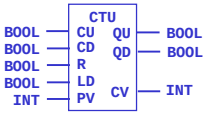

```
IF R THEN CV := 0;
ELSIF CU AND (CV < PVmax)
  THEN CV := CV+1;
END_IF;
Q := (CV >= PV);
```

CTD – čítač směrem dolů (Down-counter)




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

CTUD – obousměrný čítač (Up-down counter)



Dále existují obdobné bloky **CTUD_DINT**, **CTUD_LINT** a **CTD_ULINT** pro příslušný typ vstupu **PV** a výstupu **CV**

Dále existují obdobné bloky **CTU_DINT**, **CTU_LINT**, **CTU_UDINT** a **CTU_ULINT**, **CTD_DINT**, **CTD_LINT**, **CTD_UDINT** a **CTD_ULINT**, pro příslušný typ vstupu **PV** a výstupu **CV**

```
IF R THEN CV := 0;
ELSIF LD THEN CV := PV;
ELSE
  IF NOT (CU AND CD) THEN
    IF CU AND (CV < PVmax)
      THEN CV := CV+1;
    ELSIF CD AND (CV>Pvmin)
      THEN CV := CV-1;
    END_IF;
  END_IF;
END_IF;
QU := (CV >= PV);
QD := (CV <= 0);
```


Časovací funkční bloky (časovače)

- Vstupy a výstupy mají následující význam
  - **IN** – příznak pro spuštění časovače
  - **PT** – přednastavená doba časování
  - **Q** – příznak vypršení přednastavené hodnoty
  - **ET** – průběžná doba běhu (po ukončení časování nabývá hodnotu **PT**)

*** je jménem bloku:

- TP (Pulse) – příchodem náběžné hrany vstupu **IN** se nahodí výstup **Q** po dobu **PT**. Přijde-li další náběžná hrana během časování, je ignorována
- TON (On-delay) – výstup **Q** zpožďuje náběžnou hranu vstupu **IN** je zpožděna o čas **PT**. Sestupná hrana pulsu je zachována. Pokud puls vstupu **IN** trvá kratší dobu než **PT**, je ingorován
- TOF (Off-delay) – výstup **Q** je nahozen s náběžnou hranou vstupu **IN**, sestupná hrana je zpožděna o čas **PT**. Pokud mezi sestupnou hranou **IN** a následující náběžnou hranou je kratší čas než **PT**, je tato sestupná hrana ignorována

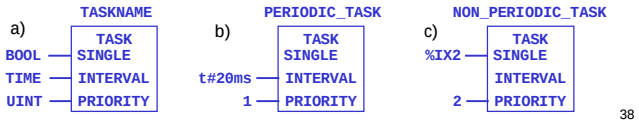
POU: Programy

- Program je definován v IEC 61131-1 jako „logické spojení (assembly) všech prvků programovacích jazyků a konstrukcí nezbytných pro záměrné zpracování signálů vyžadovaných pro řízení stroje nebo procesu pomocí PLC“
- Deklarace programů je obdobná deklaraci FB s následujícími rozdíly:
  - Pro deklaraci se užívají klíčová slova **PROGRAM** ... **END_PROGRAM**
  - Program může obsahovat konstrukci **VAR_ACCESS** ... **END_VAR**. Tato konstrukce umožňuje specifikovat proměnné, které mají být dostupné pomocí komunikačních služeb specifikovaných v IEC 61131-5
  - Instance programů mohou být vytvořeny pouze ve zdrojích (resources), zatímco instance FB mohou být vytvářeny v programech a v jiných FB.

37

Úlohy (tasks)

- Úloha (task) je definován jako prováděcí řídicí prvek schopný volat množinu POUs
  - Volání může být periodické nebo jednorázově aktivované náběžnou hranou specifikované Booleovské proměnné
  - Množina POUs zahrnuje programy a funkční bloky specifikované v deklaraci programů
- Task (grafická podoba na obr. a) může být spouštěn:
  - Periodicky s periodou danou nenulovou hodnotou přivedenou na vstup **INTERVAL** (viz obr. b)
  - Neperiodicky náběžnou hranou signálu přivedeného na vstup **SINGLE** (obr. c)
  - V obou případech má prioritu danou vstupem **PRIORITY** (0 je nejvyšší priorita)



38

Rozvrhování úloh (Task scheduling)

- Priorita POU (tj. priorita tasku, který ji obsahuje) může být použita pro rozvrhování úloh
  - Nepreemptivní rozvrhování. V okamžiku dostupnosti CPU je spuštěna POU s nejvyšší prioritou. Je-li takových POU několik, je spuštěna ta, která čeká nejdéle
  - Preemptivní rozvrhování. V okamžiku přidělení času dané POU, může být přerušeno vykonávání jiné POU s nižší prioritou. Toto přerušení může trvat dokud není dokončeno zpracování POU s vyšší prioritou. POU by nemělo přerušovat POU se stejnou prioritou
- V závislosti na prioritách nemusí být daná POU spuštěna okamžitě po naplánování. Výrobce by měl poskytnout informace, které umožní uživateli zjistit, zda budou dodrženy všechny mezní časy (deadlines) v dané konfiguraci
- Program, který není zařazen do žádné úlohy má nejnižší systémovou prioritu. Po svém ukončení je spuštěn znovu

39