INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Programování dle normy IEC 61 131

Autoři: **Ing. Josef Kovář**
Ing. Zuzana Prokopová
Ing. Ladislav Šmejkal, CSc.

Partneři projektu:

Rostra s.r.o. **Trimill, a.s.**

Výukový materiál byl vytvořen v rámci projektu „Implementace programování PLC automatů dle evropské normy IEC 61 131 do výuky žáků středních škol“, reg. č. CZ.1.07/1.1.08/01.0016.

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

1. Norma IEC 61 131 – základní pojmy

Než začneme doporučuji stáhnou si text příručky *Programování PLC Tecomat podle normy IEC 61 131-3*, ze které budeme čerpat obecné informace a provést instalaci poslední verze prostředí Mosaic, která je volně přístupná na adrese www.tecomat.cz.

Norma IEC 61 131 pro programovatelné řídící systémy má sedm základních částí a představuje souhrn požadavků na moderní řídicí systémy. Je nezávislá na konkrétní organizaci či firmě a má širokou mezinárodní podporu. Jednotlivé části normy jsou věnovány jak technickému tak programovému vybavení těchto systémů.

V ČR byly přijaty jednotlivé části této normy pod následujícími čísly a názvy:

ČSN EN 61 131-1 Programovatelné řídicí jednotky - Část 1: Všeobecné informace
ČSN EN 61 131-2 Programovatelné řídicí jednotky - Část 2: Požadavky na zařízení a zkoušky
ČSN EN 61 131-3 Programovatelné řídicí jednotky - Část 3: Programovací jazyky
ČSN EN 61 131-4 Programovatelné řídicí jednotky - Část 4: Podpora uživatelů
ČSN EN 61 131-5 Programovatelné řídicí jednotky - Část 5: Komunikace
ČSN EN 61 131-7 Programovatelné řídicí jednotky - Část 7: Programování fuzzy řízení

V Evropské unii jsou tyto normy přijaty pod číslem EN IEC 61 131.

1.1. *Základní myšlenky normy IEC 61 131-3*

Norma IEC 61 131-3 je třetí částí skupiny norem řady IEC 61 131. Dělí se na dvě základní části:

- **q** společné prvky
- **q** programovací jazyky

1.1.1. Společné prvky

Typy dat

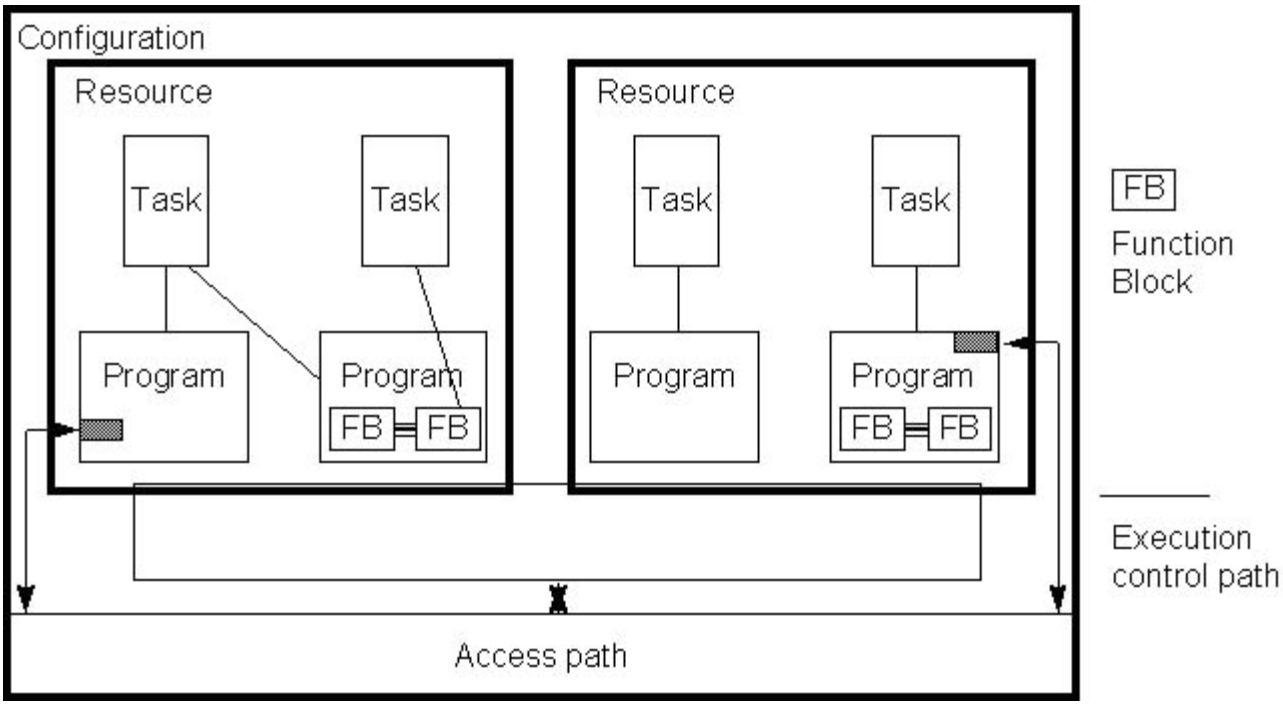
V rámci společných prvků jsou definovány typy dat. Definování datových typů napomáhá prevenci chyb v samém počátku tvorby projektu. Je nutné definovat typy všech použitých parametrů. Běžné datové typy jsou BOOL, BYTE, WORD, INT (Integer), REAL, DATE, TIME, STRING atd. Z těchto základních datových typů je pak možné odvozovat vlastní uživatelské datové typy, tzv. odvozené datové typy. Tímto způsobem můžeme např. definovat jako samostatný datový typ analogový vstupní kanál a opakovaně ho používat pod definovaným jménem.
Pod pojmem data obvykle rozumíme proměnné, případně konstanty. Bude – li programátor používat určitou proměnnou, musí ji nejprve deklarovat. Deklarace proměnné spočívá v jejím pojmenování, přidělení datového typu, případně inicializační hodnoty. Proměnné mohou být přiřazeny explicitně k hardwarovým adresám (např. vstupům, výstupům) pouze v konfiguracích, zdrojích nebo programech (viz struktura programu). Tímto způsobem je dosaženo vysokého stupně hardwarové nezávislosti a možnosti opakovaného využití softwaru na různých hardwarových platformách.

Lokální a globální proměnné

Proměnné mohou být lokálního, nebo globálního charakteru podle oblasti své působnosti. Použití lokální proměnné je omezeno pouze na tu programovou organizační jednotku, ve které byla deklarována. Globální proměnné mají působnost v rámci celého projektu, jejich deklarace není součástí žádné programové organizační jednotky, deklarují se samostatně.

Struktura programu (konfigurace, zdroje a úlohy)

Pro lepší pochopení může posloužit softwarový model, který je definován v normě a znázorněn na obrázku 1.



Obrázek 1 Softwarový model programu

Konfigurace (Configuration) definuje nejvyšší úroveň celého softwarového řešení určitého problému řízení. Konfigurace je závislá na konkrétním řídicím systému, včetně uspořádání hardwaru, jako jsou například typy procesorových jednotek, paměťové oblasti přiřazené vstupním a výstupním kanálům a charakteristiky systémového programového vybavení (operačního systému).

Zdroje (Resource)
V rámci konfigurace můžeme pak definovat jeden nebo více tzv. zdrojů. Na zdroj se můžeme dívat jako na nějaké zařízení, které je schopno vykonávat IEC programy. Zdrojem je obvykle centrální procesorová jednotka (CPU).

Úloha (Task)
Uvnitř zdroje můžeme definovat jednu nebo více tzv. úloh (Task). Úlohy řídí provádění souboru programů a/nebo funkčních bloků. Tyto jednotky mohou být prováděny buď periodicky nebo po vzniku speciální spouštěcí události, což může být např. změna proměnné.
Programy jsou vystavěny z řady různých softwarových prvků, které jsou zapsány v některém z jazyků definovaném v normě.
Často je program složen ze sítě funkcí a funkčních bloků, které jsou schopny si vyměňovat data. Funkce a funkční bloky jsou základní stavební kameny, které obsahují datové struktury a algoritmus.

Programové organizační jednotky

Funkce, funkční bloky a programy jsou v rámci normy IEC 61 131 nazývány společně programové organizační jednotky (Program Organization Units, někdy se pro tento důležitý a často používaný pojem používá zkratka POUs).

Funkce
IEC 61 131-3 definuje standardní funkce a uživatelem definované funkce. Standardní funkce jsou např. ADD pro sčítání, ABS pro absolutní hodnotu, SQRT pro odmocninu, SIN pro sinus a COS pro cosinus. Jakmile jsou jednou definovány nové uživatelské funkce, mohou být používány opakovaně.

Funkční bloky
Na funkční bloky se můžeme dívat jako na integrované obvody, které reprezentují hardwarové řešení specializované řídicí funkce. Obsahují algoritmy i data, takže mohu zachovávat informaci o minulosti, (tím se liší od funkcí). Mají jasně definované rozhraní a skryté vnitřní proměnné, podobně jako integrovaný obvod nebo černá skříňka. Umožňují tím jednoznačně oddělit různé úrovně programátorů nebo obslužného personálu. Klasickými příklady funkčního bloku jsou např. regulační smyčka pro teplotu nebo PID regulátor. Jakmile je jednou funkční blok definován, může být používán opakovaně v daném programu, nebo v jiném programu, nebo dokonce i v jiném projektu. Je tedy univerzální a mnohonásobně použitelný. Funkční bloky mohou být zapsány v libovolném z jazyků definovaném v normě. Mohou být tedy plně definovány

uživatelem. Odvozené funkční bloky jsou založeny na standardních funkčních blocích, ale v rámci pravidel normy je možno vytvářet i zcela nové zákaznické funkční bloky.
Interface funkcí a funkčních bloků je popsán stejným způsobem: Mezi deklarací označující název bloku a deklarací pro konec bloku je uveden soupis deklarací vstupních proměnných, výstupních proměnných a vlastní kód v tzv. těle bloku.

Programy

Na základě výše uvedených definic lze říci, že program je vlastně sítí funkcí a funkčních bloků. Program může být zapsán v libovolném z jazyků definovaných v normě.

1.2. Programovací jazyky

V rámci standardu je definováno pět programovacích jazyků. Jejich sémantika i syntaxe je přesně definována a neponechává žádný prostor pro nepřesné vyjadřování. Zvládnutím těchto jazyků se tak otevírá cesta k používání široké škály řídicích systémů, které jsou na tomto standardu založeny.
Programovací jazyky se dělí do dvou základních kategorií:

1.2.1. Textové jazyky

IL - Instruction List - jazyk seznamu instrukcí
Textový jazyk nejvíce připomínající assembler

ST - Structured Text - jazyk strukturovaného textu
Je velmi výkonný vyšší programovací jazyk, který má kořeny ve známých jazycích Pascal a C. Obsahuje všechny podstatné prvky moderního programovacího jazyka, včetně větvení (IF-THEN-ELSE a CASE OF) a iterační smyčky (FOR, WHILE a REPEAT). Tyto prvky mohou být vnořovány. Tento jazyk je vynikajícím nástrojem pro definování komplexních funkčních bloků, které pak mohou být použity v jakémkoliv jiném programovacím jazyku.

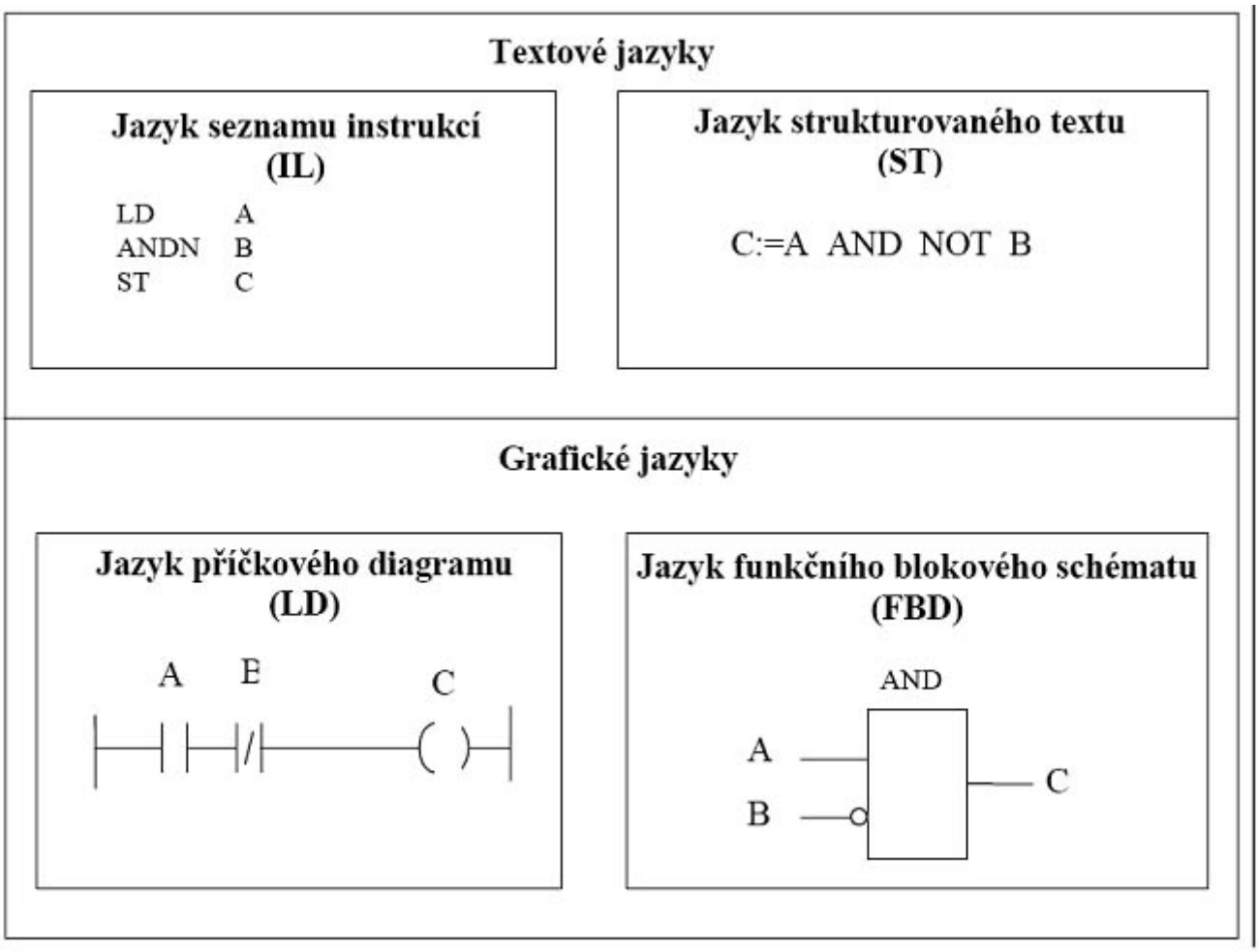
1.2.2. Grafické jazyky

LD - Ladder Diagram - jazyk příčkového diagramu (jazyk kontaktních schémat).
Má původ v USA. Je založen na grafické reprezentaci reléové logiky.

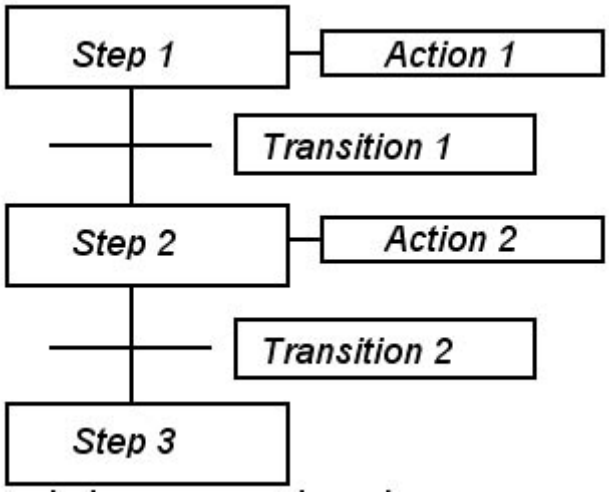
FBD - Function Block Diagram - jazyk funkčního blokového schématu
je velmi blízký procesnímu průmyslu. Vyjadřuje chování funkcí, funkčních bloků a programů jako soubor vzájemně provázaných grafických bloků, podobně jako v elektronických obvodových diagramech. Je to určitý systém prvků, které zpracovávají signály.

SFC - Sequential Function Chart – jazyk sekvenčních diagramů podobný Petriho sítím. Algoritmus je tvořen posloupností kroků (Steps), oddělených přechodovými podmínkami (Transitions). Ke krokům mohou být připojeny akce (Actions).

Pro první představu je na Obr.2 stejná logická funkce, a to součin proměnné A a negované proměnné B s výsledkem ukládaným do proměnné C, vyjádřen ve čtyřech základních programovacích jazycích, příklad struktury programu vytvořeného v jazyku SFC vidíme na Obr. 3.



Obr. 2. Ukázka programovacích jazyků



Obr. 3. Ukázka programovacího jazyka SFC

1.3. *Základní stavební bloky programu*

Základním pojmem při programování podle normy IEC 61 131-3 je termín ***Programová Organizační Jednotka*** nebo zkráceně ***POU*** *(Program Organisation Unit*). Jak vyplývá z názvu, POU je nejmenší nezávislá část uživatelského programu. POU mohou být dodávány od výrobce řídícího systému nebo je může napsat uživatel. Každá POU může volat další POU a při tomto volání může volitelně předávat volané POU jeden nebo více parametrů.

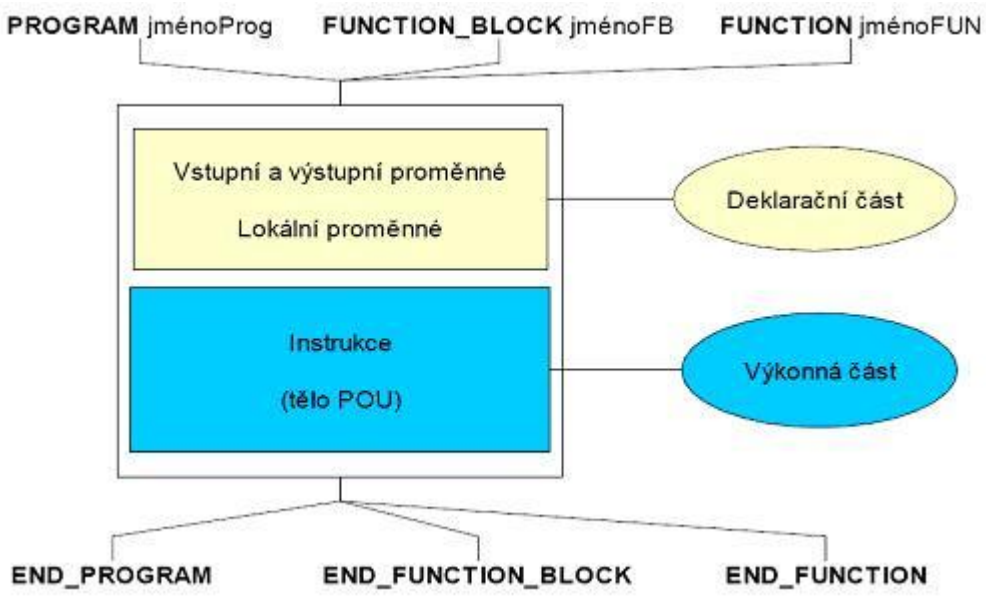
Existují tři základní typy POU :
funkce *(function, FUN)*
funkční blok *(function block, FB)*
program *(program, PROG)*

1.3.1. Funkce
Je nejjednodušší POU, její hlavní charakteristikou je to, že pokud je volána se stejnými vstupními parametry, musí produkovat stejný výsledek (funkční hodnotu). Funkce může vracet pouze jeden výsledek.

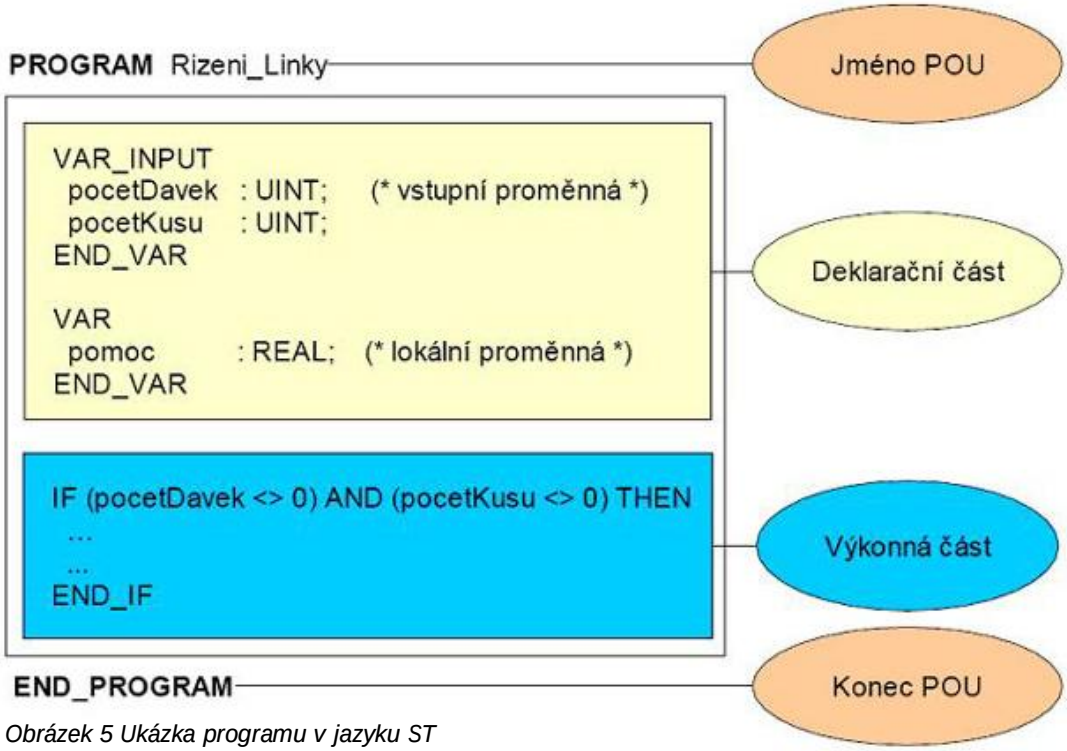
1.3.2. Funkční blok
Dalším typem POU je funkční blok, který si na rozdíl od funkce, může pamatovat některé hodnoty z předchozího volání (např. stavové informace). Ty pak mohou ovlivňovat výsledek. Hlavním rozdílem mezi funkcí a funkčním blokem je tedy schopnost funkčního bloku vlastnit paměť pro zapamatování hodnot některých proměnných. Tuto schopnost funkce nemají a jejich výsledek je tedy jednoznačně určen vstupními parametry při volání funkce. Funkční blok může také (na rozdíl od funkce) vracet více než jeden výsledek.

1.3.3. Program
Posledním typem POU je program, který představuje vrcholovou programovou jednotku v uživatelském programu. Centrální jednotka PLC může zpracovávat více programů a programovací jazyk ST obsahuje prostředky pro definice spouštění programů (v jaké periodě vykonávat program, s jakou prioritou, apod.).

Struktura POU
Každá POU se skládá ze dvou základních částí : ***deklarační a výkonné***, jak je vidět na obr.4. V deklarační části POU se definují proměnné potřebné pro činnost POU. Výkonná část pak obsahuje vlastní příkazy pro realizaci požadovaného algoritmu.
Příklad struktury POU typu **PROGRAM** vidíme na Obr. 5. Definice POU začíná klíčovým slovem **PROGRAM** a je ukončena klíčovým slovem **END_PROGRAM**. Tato klíčová slova vymezují rozsah POU.
Za klíčovým slovem **PROGRAM** je uvedeno jméno POU. Poté následuje deklarační část POU. Ta obsahuje definice proměnných uvedené mezi klíčovými slovy **VAR_INPUT** a **END_VAR** resp. **VAR** a **END_VAR**.
Na závěr je uvedena výkonná část POU obsahující příkazy jazyka ST pro zpracování proměnných. Texty uvedené mezi znaky (* a *) jsou poznámky (komentáře).



Obrázek 4 Struktura POU



Obrázek 5 Ukázka programu v jazyku ST

Deklarační část POU

Deklarační část POU obsahuje definice proměnných potřebných pro činnost POU.
Proměnné jsou používány pro ukládání a zpracování informací. Každá ***proměnná*** je definována ***jménem proměnné a datovým typem***. Datový typ určuje velikost proměnné v paměti a zároveň do značné míry určuje způsob zpracování proměnné. Pro definice proměnných jsou k dispozici standardní datové typy (**BOOL, BYTE, INT**, ...). Použití těchto typů závisí na tom, jaká informace bude v proměnné uložena (např. typ **BOOL** pro informace typu ANO-NE, typ **INT** pro uložení celých čísel se znaménkem apod.). Uživatel má samozřejmě možnost definovat svoje vlastní datové typy.
Umístění proměnných v paměti PLC systému zajišťuje automaticky programovací prostředí. Pokud je to potřeba, může umístění proměnné v paměti definovat i uživatel.
Proměnné můžeme rozdělit podle použití na ***globální a lokální***.
Globální proměnné jsou definovány vně POU a mohou být použity v libovolné POU (jsou viditelné z libovolné POU).
Lokální proměnné jsou definovány uvnitř POU a v rámci této POU mohou být používány (z ostatních POU nejsou viditelné).
A konečně proměnné jsou také používány pro předávání parametrů při volání POU. V těchto případech mluvíme o ***vstupních*** resp. ***výstupních proměnných***.

Příklad deklarace proměnných v POU

```
FUNCTION_BLOCK PrikladDeklaraceProm
VAR_INPUT (* vstupní proměnné *)
  logPodminka : BOOL; (* binární hodnota *)
END_VAR
VAR_OUTPUT (* výstupní proměnné *)
  vysledek : INT; (* celočíselná hodnota se znaménkem *)
END_VAR
VAR (* lokální proměnné *)
  kontrolniSoucet : UINT; (* celočíselná hodnota *)
  mezivysledek : REAL; (* reálná hodnota *)
END_VAR
(* tady bude výkonná část POU *)
END_FUNCTION_BLOCK
```

Výkonná část POU

Výkonná část POU následuje za částí deklarační a obsahuje příkazy a instrukce, které jsou zpracovány centrální jednotkou PLC. Ve výjimečných případech nemusí definice POU obsahovat žádnou deklarační část a potom je

výkonná část uvedena bezprostředně za definicí začátku POU.
Příkladem může být POU, která pracuje pouze s globálními proměnnými, což sice není ideální řešení, ale může existovat.
Výkonná část POU může obsahovat volání dalších POU. Při volání mohou být předávány parametry pro volané funkce resp. funkční bloky.

1.3.4. Ukázka programu v jazyku ST

```
VAR_GLOBAL
// inputs
   Start1 AT %X0.0,
   Start2 AT %X0.1,
   Stop1 AT %X0.2,
   Stop2 AT %X0.3 : BOOL;
// outputs
   Linka1 AT %Y0.0,
   Linka2 AT %Y0.1 : BOOL;
END_VAR
FUNCTION_BLOCK fbStartMotoru
//-------------------------------------------------------------------------
VAR_INPUT
   start : BOOL;
   stop : BOOL;
END_VAR
VAR_OUTPUT
   vystup : BOOL;
END_VAR
   vystup := ( vystup OR start) AND NOT stop;
END_FUNCTION_BLOCK

PROGRAM Motory
//-------------------------------------------------------------------------
VAR
   motor1 : fbStartMotoru;
   motor2 : fbStartMotoru;
END_VAR
   motor1( start := Start1, stop := Stop1, vystup => Linka1);
   motor2( start := Start2, stop := Stop2, vystup => Linka2);
END_PROGRAM
```

1.4. Společné prvky programovacích jazyků

Každý program pro PLC se skládá ze základních jednoduchých prvků, určitých nejmenších jednotek, ze kterých se vytvářejí deklarace a příkazy. Tyto jednoduché prvky můžeme rozdělit na :
oddělovače (Delimiters),
identifikátory (Identifiers)
literály (Literals)
klíčová slova (Keywords)
komentáře (Comments)

Oddělovače
jsou speciální znaky (např. (,), =, :, mezera, apod.) s různým významem.

Identifikátory
jsou alfanumerické řetězce znaků, které slouží pro vyjádření jmen uživatelských funkcí, návěští nebo programových organizačních jednotek (např. **Tepl_N1, Spinac_On, Krok4, Pohyb_doprava** apod.).

Literály
slouží pro přímou reprezentaci hodnot proměnných (např. 0,1; 84; 3,79; TRUE ; zelena apod.).

Klíčová slova
jsou standardní identifikátory (např. **FUNCTION, REAL, VAR_OUTPUT**,apod.). Jejich přesný tvar a význam odpovídá normě IEC 61 131-3. Klíčová slova se nesmějí používat pro vytváření jakýchkoli uživatelských jmen. Pro zápis klíčových slov mohou být použita jak velká tak malá písmena resp. jejich libovolná kombinace.

Komentáře
nemají syntaktický ani sémantický význam, jsou však důležitou částí dokumentace programu. Při překladu jsou tyto řetězce ignorovány, takže mohou obsahovat i znaky národních abeced.
Překladač rozeznává dva druhy komentářů :
obecné komentáře - řetězce znaků začínající dvojicí znaků **(*** a ukončené dvojicí znaků ***)**
řádkové komentáře - začínající dvojicí znaků **//** a jsou ukončeny koncem řádku

Datové typy
Pro programování v některém z jazyků podle normy IEC 61 131-3 jsou definovány datové typy:
elementární, předdefinované, (Elementary data types)
rodové datové typy (Generic data type) pro příbuzné skupiny datových typů.
odvozené (uživatelské) datové typy (Derived data type, Type definition).

Elementární datové typy
Elementární datové typy jsou charakterizované šířkou dat (počtem bitů) a případně i rozsahem hodnot.
Inicializace elementárních datových typů
Důležitým principem při programování podle normy IEC 61 131-3 je, že všechny proměnné v programu mají inicializační (počáteční) hodnotu. Pokud uživatel neuvede jinak, bude proměnná inicializována implicitní (předdefinovanou, default) hodnotou podle použitého datového typu. Předdefinované počáteční hodnoty pro elementární datové typy jsou převážně nuly, u data je to D#1970-01-01.
Přehled podporovaných datových typů je uveden v tabulce.

Klíčové slovo	Anglicky	Datový typ	Bitů	Rozsah hodnot
BOOL	Boolean	Boolovské číslo	1	0,1
SINT	Short integer	Krátké celé číslo	8	–128 až 127
INT	Integer	Celé číslo	16	–32 768 až +32 767
DINT	Double integer	Celé číslo, dvojnásobná délka	32	–2 147 483 648 až +2 147 483 647
USINT	Unsigned short integer	Krátké celé číslo bez znaménka	8	0 až 255
UINT	Unsigned integer	Celé číslo bez znaménka	16	0 až 65 535
UDINT	Unsigned double integer	Celé číslo bez znaménka, dvojnásobná délka	32	0 až +4 294 967 295
REAL	Real (single precision)	Číslo v pohyblivé řádové čárce (jednoduchá přesnost)	32	±2.9E-39 až ±3.4E+38 Podle IEC 559
LREAL	Long real (double precision)	Číslo v pohyblivé řádové čárce (dvojnásobná přesnost)	64	Podle IEC 559
TIME	Duration	Trvání času	24d 20:31:23.647	
DATE	Date (only)	Datum	Od 1.1.1970 00:00:00	
TIME_OF_DAY nebo TOD	Time of day (only)	Denní čas	24d 20:31:23.647	
DATE_AND_TIME nebo DT	Date and time of day	„Absolutní čas“	Od 1.1.1970 00:00:00	
STRING	String	Řetězec	Max.255 znaků	
BYTE	Byte(bit string of 8 bits)	Sekvence 8 bitů	8	Není deklarován rozsah
WORD	Word (bit string of 16bits)	Sekvence 16 bitů	16	Není deklarován rozsah
DWORD	Double word (bit string of 32 bits)	Sekvence 32 bitů	32	Není deklarován rozsah

Rodové datové typy
Rodové datové typy vyjadřují vždy celou skupinu (rod) datových typů. Jsou uvozeny prefixem **ANY**. Např. zápisem **ANY_BIT** se rozumí všechny datové typy uvedené v následujícím výčtu:
DWORD, WORD, BYTE, BOOL.
Přehled rodových datových typů je uveden v tabulce.

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

Odvozené datové typy
Odvozené typy, tzn. typy specifikované buď výrobcem nebo uživatelem, mohou být deklarovány pomocí textové konstrukce **TYPE...END_TYPE**. Jména nových typů, jejich datové typy, případně i s jejich inicializačními hodnotami, jsou uvedena v rámci této konstrukce. Tyto odvozené datové typy se pak mohou dále používat spolu s elementárními datovými typy v deklaracích proměnných. Definice odvozeného datového typu je globální, tj. může být použita v kterékoliv části programu pro PLC.
Příklad jednoduchých odvozených datových typů

```
TYPE
  TMyINT : INT := 15; // jednoduchy odvozeny datovy typ
  TRoomTemp : REAL := 20.0; // novy datovy typ s inicializaci
  THomeTemp : TRoomTemp;
  TPumpMode : ( off, run, fault); // novy typ deklarovany vyctem hodnot
END_TYPE
PROGRAM SingleDerivedType
VAR
  pump1Mode : TPumpMode;
  display : STRING;
  temperature : THomeTemp;
END_VAR
CASE pump1Mode OF
  off : display := 'Pump no.1 is off';
  run : display := 'Pump no.1 is running';
  fault : display := 'Pump no.1 has a problem';
END_CASE;
END_PROGRAM
```

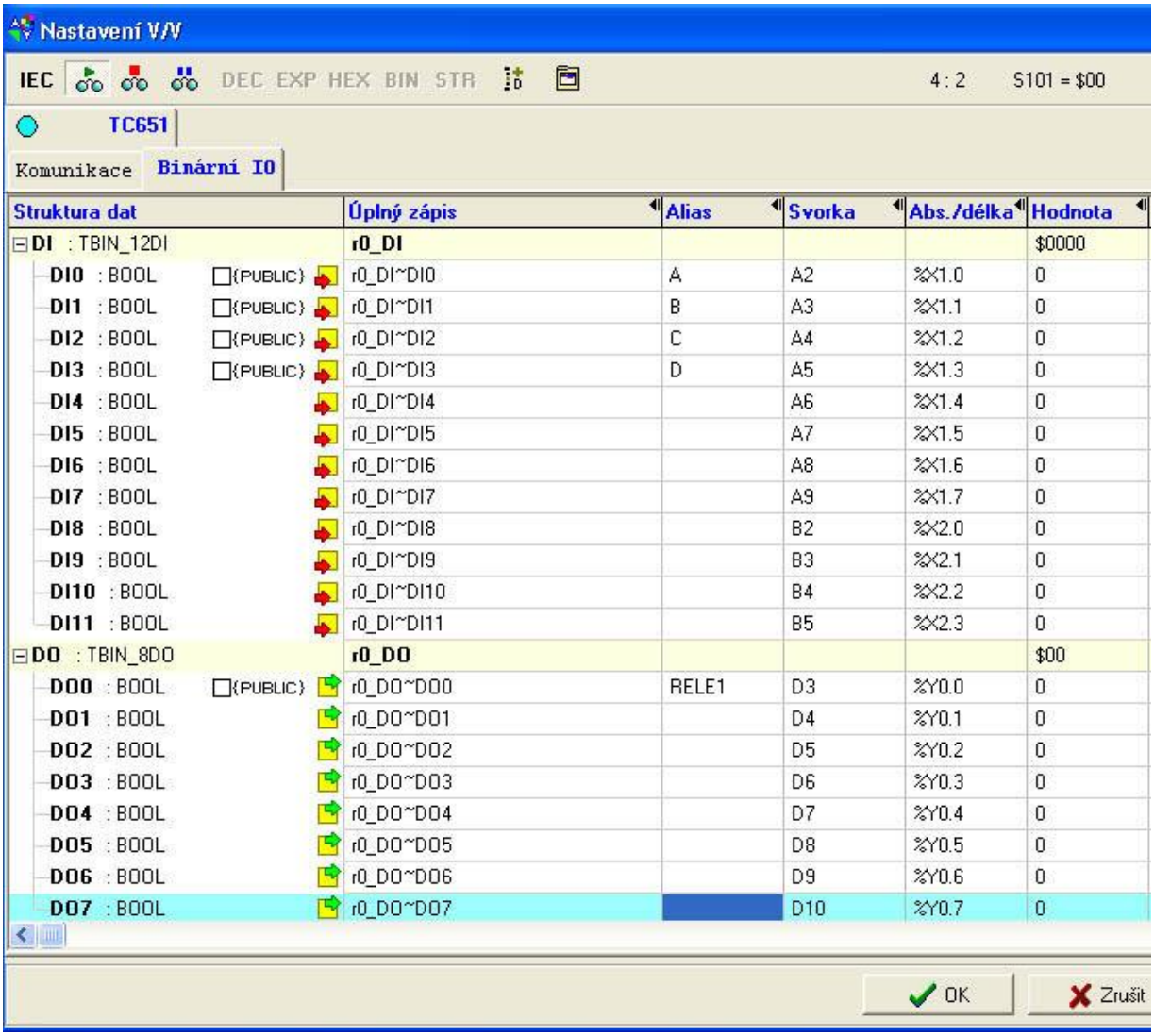
2. Programování jednoduchého kombinačního algoritmu v jazycích ST, LD a FBD.

Tvorbu programů pro PLC Tecomat si předvedeme na jednoduchém příkladu. Jeho algoritmus vytvoříme několika programovacími metodami podle normy IEC.
Zadání úlohy.
Zapněte relé RELE1 připojené k výstupní svorkovnici PLC, budou-li současně zapnuty vstupy A, B, nebo bude zapnut vstup C a vypnut vstup D.

2.1. Realizace příkladu v jazyce ST

Podle postupu, který jsme si popsali v kapitole 3, založíme projektovou skupinu s názvem Kombinacni_Obvod a projekt, který nazveme KO_ST. Jako HW platformu zvolíme kompaktní PLC TC651.
Založíme program, zvolíme programovací jazyk ST a pojmenujeme instanci programu.
Teď je třeba otestovat adresaci vstupů a výstupů a deklarovat jim symbolická jména. Nejprve se připojíme k PLC pomocí sériového komunikačního kanálu CH1, nebo rozhraním ETHERNET. K navázání komunikace použijeme manažer projektu, kde zvolíme jako typ připojení sériový port. Nastavíme parametry komunikace, které musí být shodné s parametry nastavenými v EEPROM PLC, a stiskem tlačítko Připojit navážeme komunikaci.

Projekt přeložíme příkazem PROGRAM/PŘELOŽIT, vyšleme a spustíme pomocí příkazu PLC/RUN. Aktivitu vstupů a výstupů můžeme monitorovat nástrojem Nastavení V/V (žlutá ikona IO v horní části obrazovky).



Struktura dat		Úplný zápis	Alias	Svorka	Abs./délka	Hodnota
DI : TBIN_12DI		r0_DI				$0000
DI0 : BOOL	{PUBLIC}	r0_DI~DI0	A	A2	%X1.0	0
DI1 : BOOL	{PUBLIC}	r0_DI~DI1	B	A3	%X1.1	0
DI2 : BOOL	{PUBLIC}	r0_DI~DI2	C	A4	%X1.2	0
DI3 : BOOL	{PUBLIC}	r0_DI~DI3	D	A5	%X1.3	0
DI4 : BOOL		r0_DI~DI4		A6	%X1.4	0
DI5 : BOOL		r0_DI~DI5		A7	%X1.5	0
DI6 : BOOL		r0_DI~DI6		A8	%X1.6	0
DI7 : BOOL		r0_DI~DI7		A9	%X1.7	0
DI8 : BOOL		r0_DI~DI8		B2	%X2.0	0
DI9 : BOOL		r0_DI~DI9		B3	%X2.1	0
DI10 : BOOL		r0_DI~DI10		B4	%X2.2	0
DI11 : BOOL		r0_DI~DI11		B5	%X2.3	0
DO : TBIN_8DO		r0_DO				$00
DO0 : BOOL	{PUBLIC}	r0_DO~DO0	RELE1	D3	%Y0.0	0
DO1 : BOOL		r0_DO~DO1		D4	%Y0.1	0
DO2 : BOOL		r0_DO~DO2		D5	%Y0.2	0
DO3 : BOOL		r0_DO~DO3		D6	%Y0.3	0
DO4 : BOOL		r0_DO~DO4		D7	%Y0.4	0
DO5 : BOOL		r0_DO~DO5		D8	%Y0.5	0
DO6 : BOOL		r0_DO~DO6		D9	%Y0.6	0
DO7 : BOOL		r0_DO~DO7		D10	%Y0.7	0

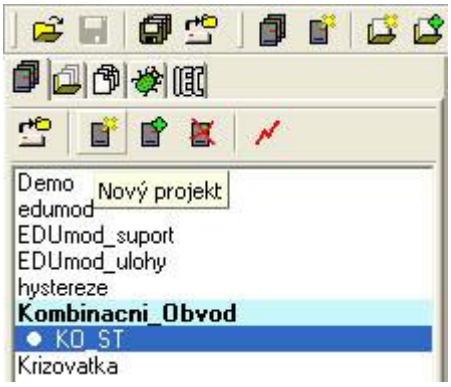
Postupně spínáme vypínače, sledujeme, na které adrese se mění hodnota a do slouce Alias napíšeme jména proměnných A,B,C,D. Deklarovali jsme tímto globální proměnné typu BOOL vázané na vstupy systému. Teď zjistíme adresaci relé zápisem 1 do zvoleného výstupu. Po potvrzení klávesou ENTER musí některý z výstupů sepnout (rozsvícená LED). Pomocí Alias deklarujeme RELE1 a potvrdíme.
Zastavíme provádění programu příkazem PLC/HALT a napíšeme program. V jazyce ST je v našem případě tvořen jediným řádkem:

```
RELE1:= A AND B OR C AND NOT D;
```

Projekt přeložíme, spustíme v PLC a můžeme ověřit náš algoritmus. Stav proměnných můžeme sledovat přímo na pracovní ploše (vpravo od příslušného řádku programu)nebo v okně Data do kterého intuitivním způsobem (např. klávesa Ins) umístíme proměnné, které chceme sledovat. Ti, kdo budou program spouštět v simulaci, mohou zde zadávat hodnoty vstupů.

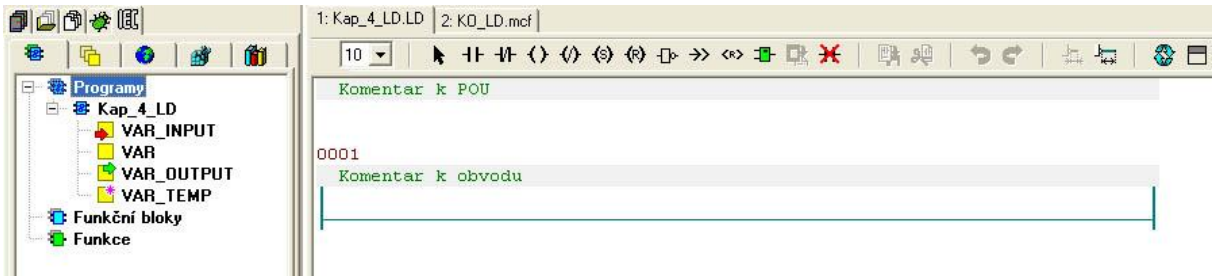
2.2. *Realizace příkladu v jazyce LD*

Nyní naprogramujeme stejný příklad v grafickém jazyce kontaktních schémat, který se nejvíce blíží klasickému releovému schématu.
V rámci projektové skupiny založíme nový projekt (viz obr. 7)

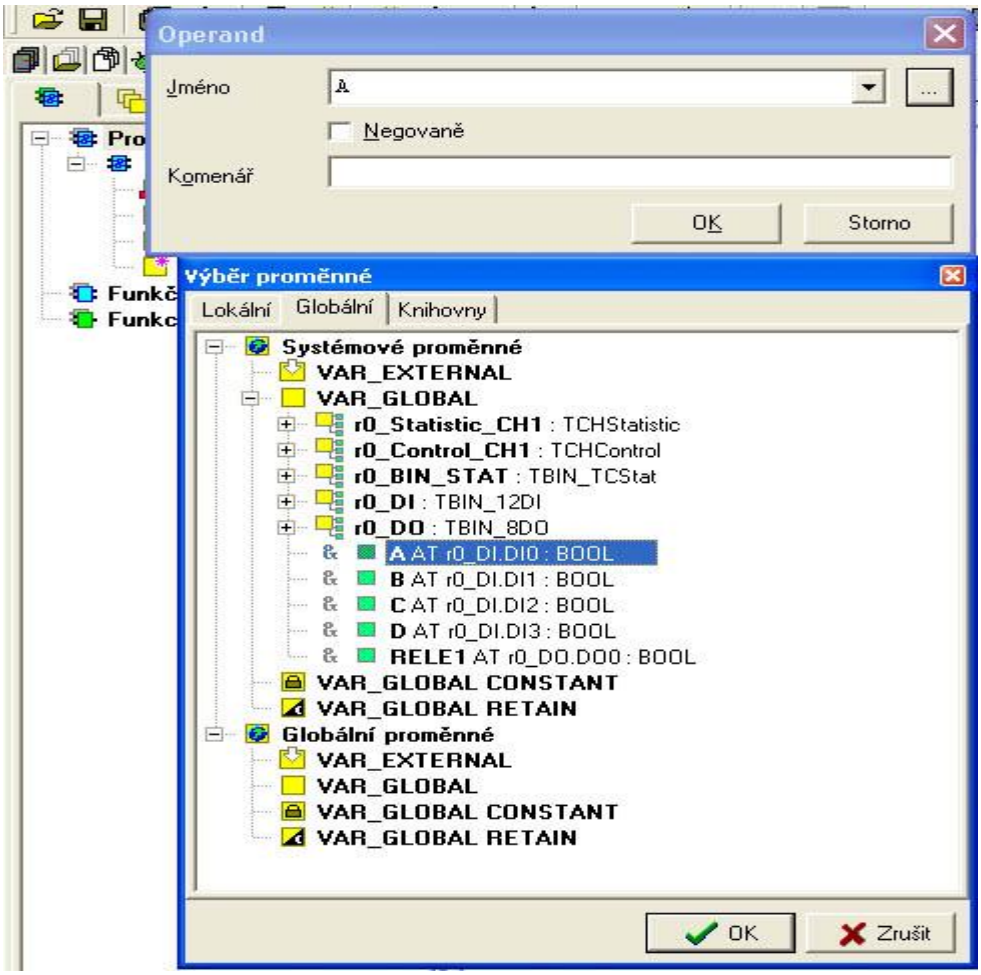


obr 7: Založení projektu

Zvolíme stejný HW jako v předchozím projektu, založíme program, zvolíme programovací jazyk LD a založíme instanci (program jsem nazval Kap_4_LD, instanci i_Kap_4_LD). Pomocí nástroje Nastavení V/V provedeme adresaci proměnných.
Objeví se pracovní plocha s prvním obvodem kontaktního schématu. Viz obr 8.
Teď je třeba vytvořit kontaktní obvod se dvěma paralelně zapojenými větvemi, v každé po dvou sériově zapojených kontaktech. Při programování klikneme levým tlačítkem myši na grafický symbol zvolené instrukce. V obvodech se objeví červené a modré body pro umístění této instrukce. Červený bod znamená do série, modrý bod paralelně.
Zadáme proměnnou (buď ji napíšeme přímo z klávesnice, nebo ji po stisku tlačítka vpravo vyhledáme) a potvrdíme.

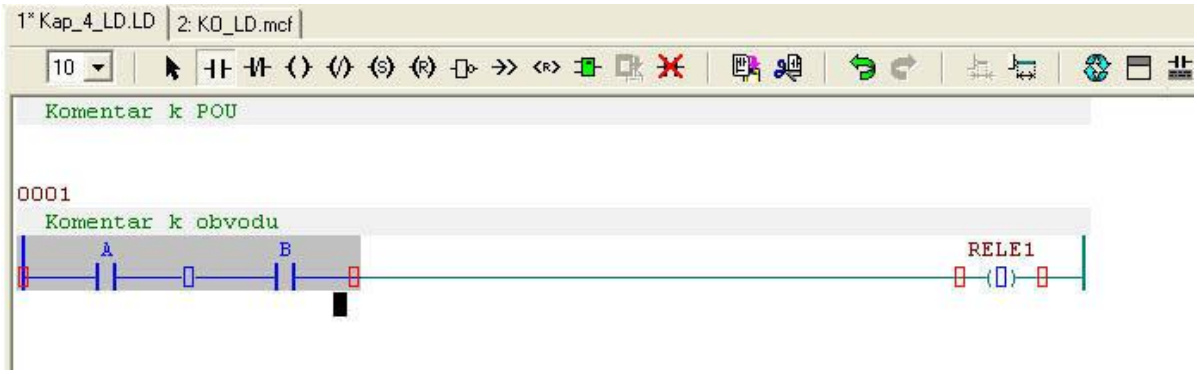


obr 8 Pracovní plocha editoru LD



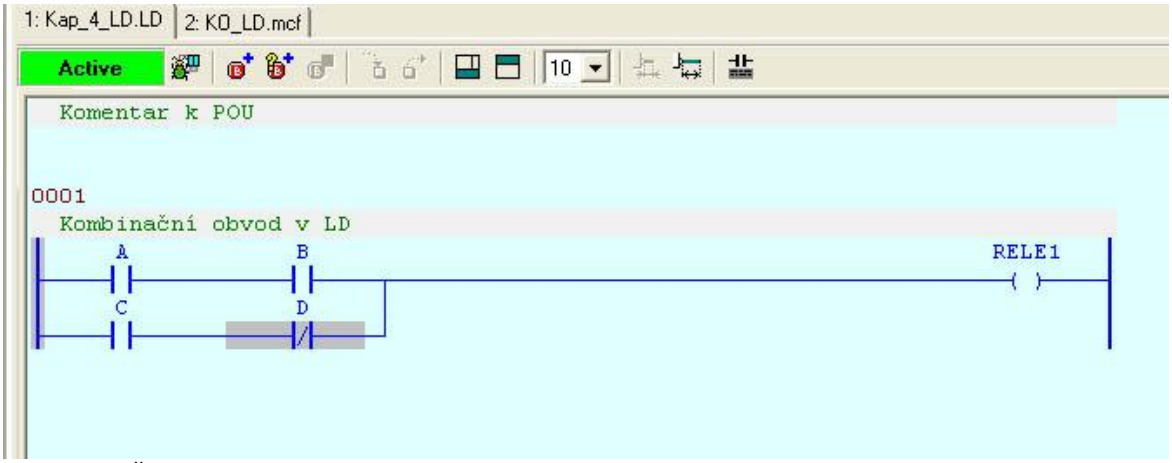
obr 18 Zadání proměnné

Tímto postupem vytvoříme celou větev. Stiskem levého tlačítka myši a tažením označíme část schématu, ke kterému chceme přidat paralelní větev. Vybereme symbol instrukce druhé větve a v obvodu se objeví modrý bod, pomocí kterého obvod rozvětvíme.



obr 19 Vytvoření paralelní větve

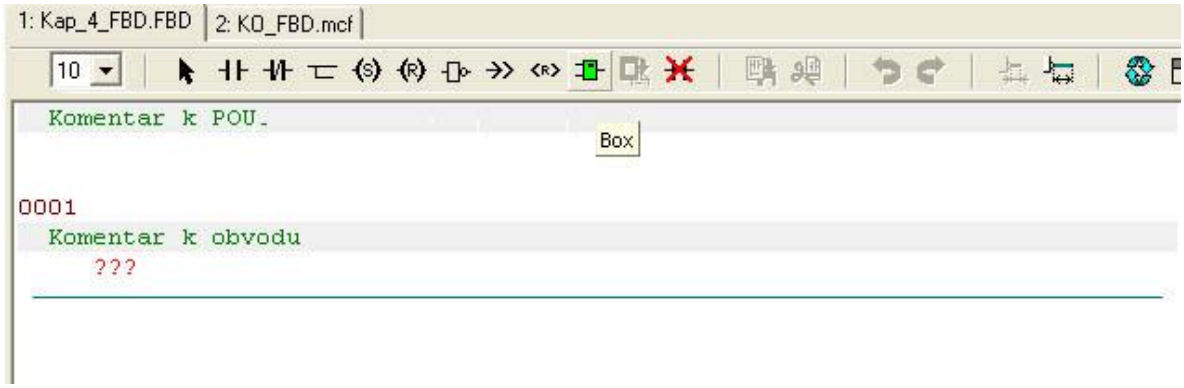
Obvod dokončíme (můžeme přidat komentář), přeložíme a spustitíme v PLC. Na pracovní ploše vidíme v režimu on-line aktivitu vstupů a výstupů, pracujeme-li v simulaci, můžeme kliknutím myši "kontakty" přepínat.



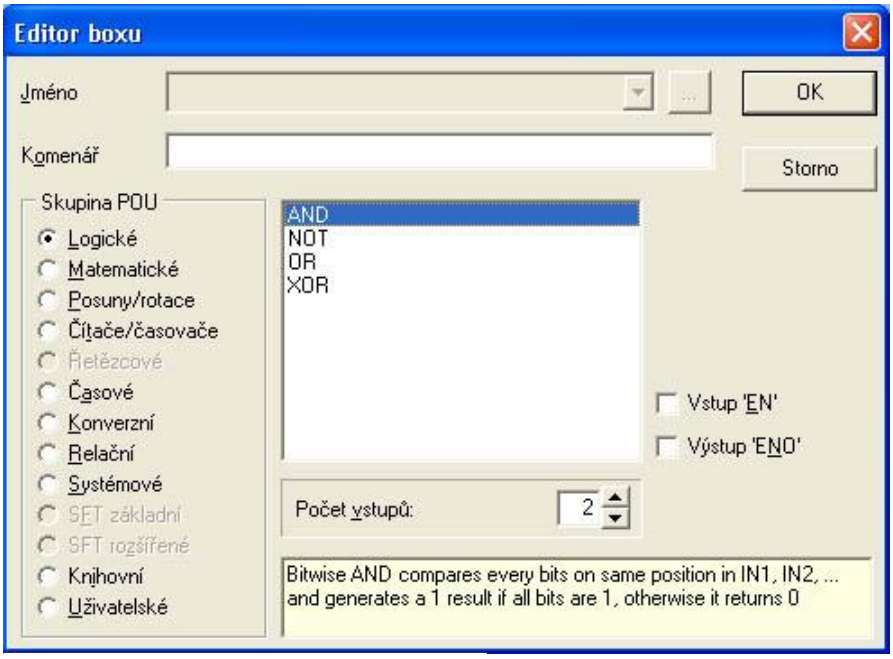
obr 20 Odladění projektu

2.3. *Realizace příkladu v jazyce FBD*

Programování ve funkčních blocích je založeno na podobných principech jako programování v kontaktním schématu. Pro náš příklad si opět vložíme nový projekt do skupiny, založíme program, zvolíme programovací metodu a vytvoříme instanci programu. Znovu definujeme vstupy a výstupy, které budeme používat a otevřeme editor FBD.



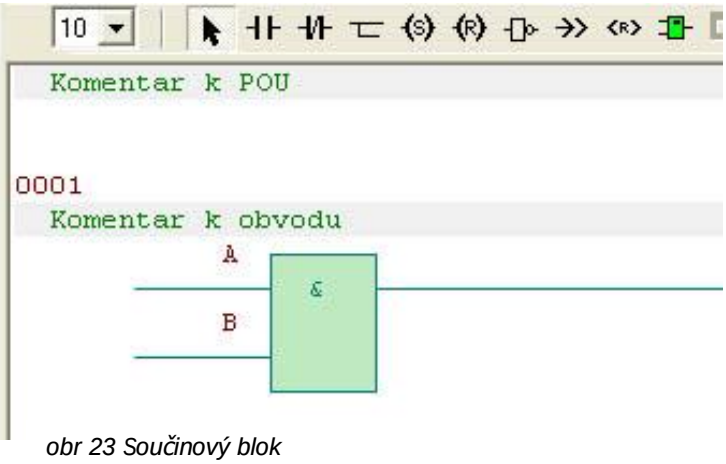
obr 21 Pracovní plocha editoru FBD



Zvolíme symbol **Box** a umístíme ho do schématu. V editoru boxu vybereme skupinu logických POU, zvolíme blok AND se 2 vstupy.

Potvrdíme a na pracovní ploše se objeví součinové hradlo, které očekává zadání adres svých vstupů. Kliknutím myši vyvoláme okno a operandy zadáme. Máme vytvořený první součinový blok.

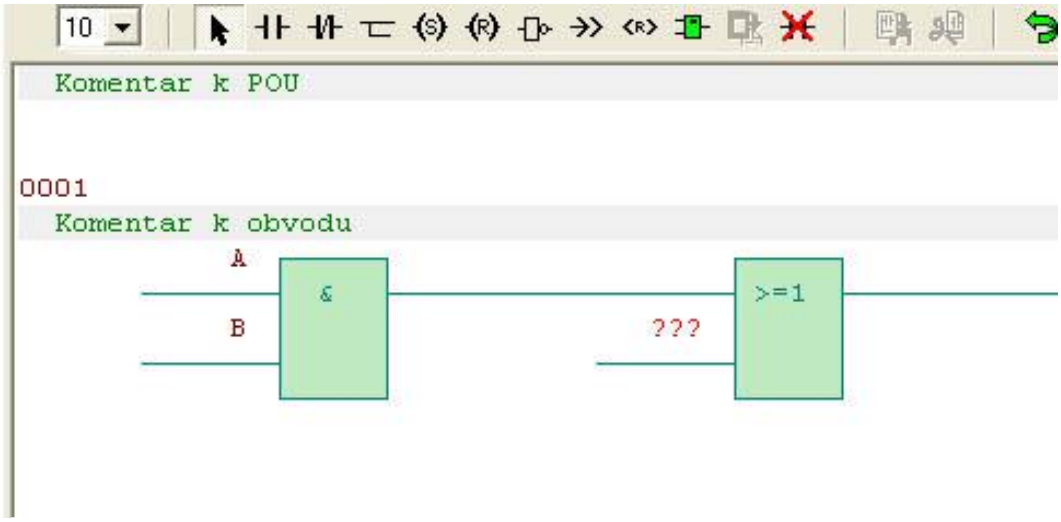
obr 23 Editor bloku



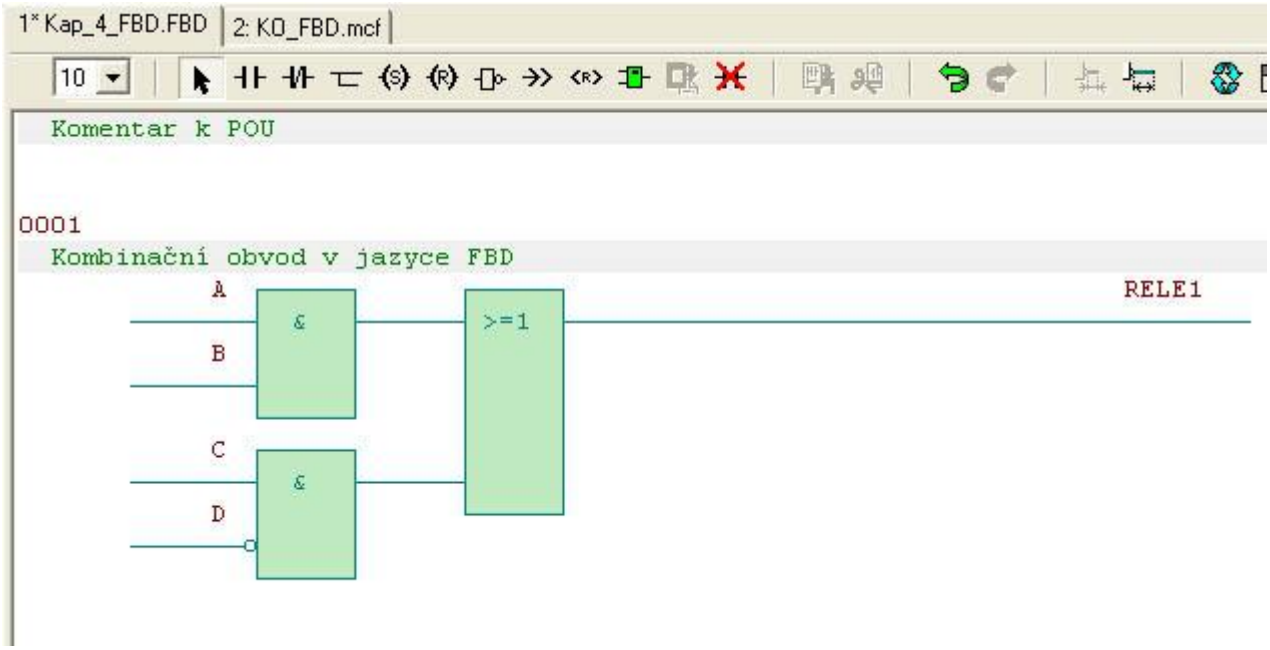
obr 23 Součinový blok

Připravíme si blok logického součtu se dvěma vstupy, který umístíme na výstup již vytvořeného součinového bloku.

Zvolíme další box a umístíme ho na volný vstup součtového bloku, zvolíme blok logického součinu a zadáme proměnnou **C**, u proměnné **D** zaškrtneme negaci. Jako
poslední zadáme instrukci pro zápis do výstupu **RELE1**. Použijeme příkaz ST (třetí symbol zleva) a zapíšeme adresu.



obr 23 Vytvoření součtového bloku



obr 23 Program v jazyce FBD

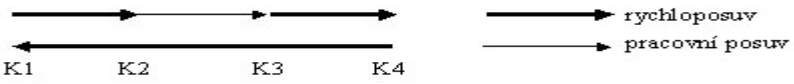
Projekt přeložíme, vyšleme do PLC a monitorujeme obdobným způsobem jako v LD.

Na závěr projektovou skupinu archivujeme postupem popsaným v třetí kapitole.

3. Programování sekvenčního algoritmu v jazycích ST, LD a FBD.

Zadání úlohy.
Vytvořte program pro řízení hydraulické posuvové jednotky podle zadaného režimu.
Po stisku tlačítka START se suport rozjede rychloposuvem z výchozí pozice (K1), na K2 se změní rychlost na pracovní posuv, mezi snímači K3 a K4 jede opět rychloposuvem a na konci dráhy se zastaví a vrátí se rychloposuvem zpět. K nouzovému zastavení suportu slouží tlačítko STOP.

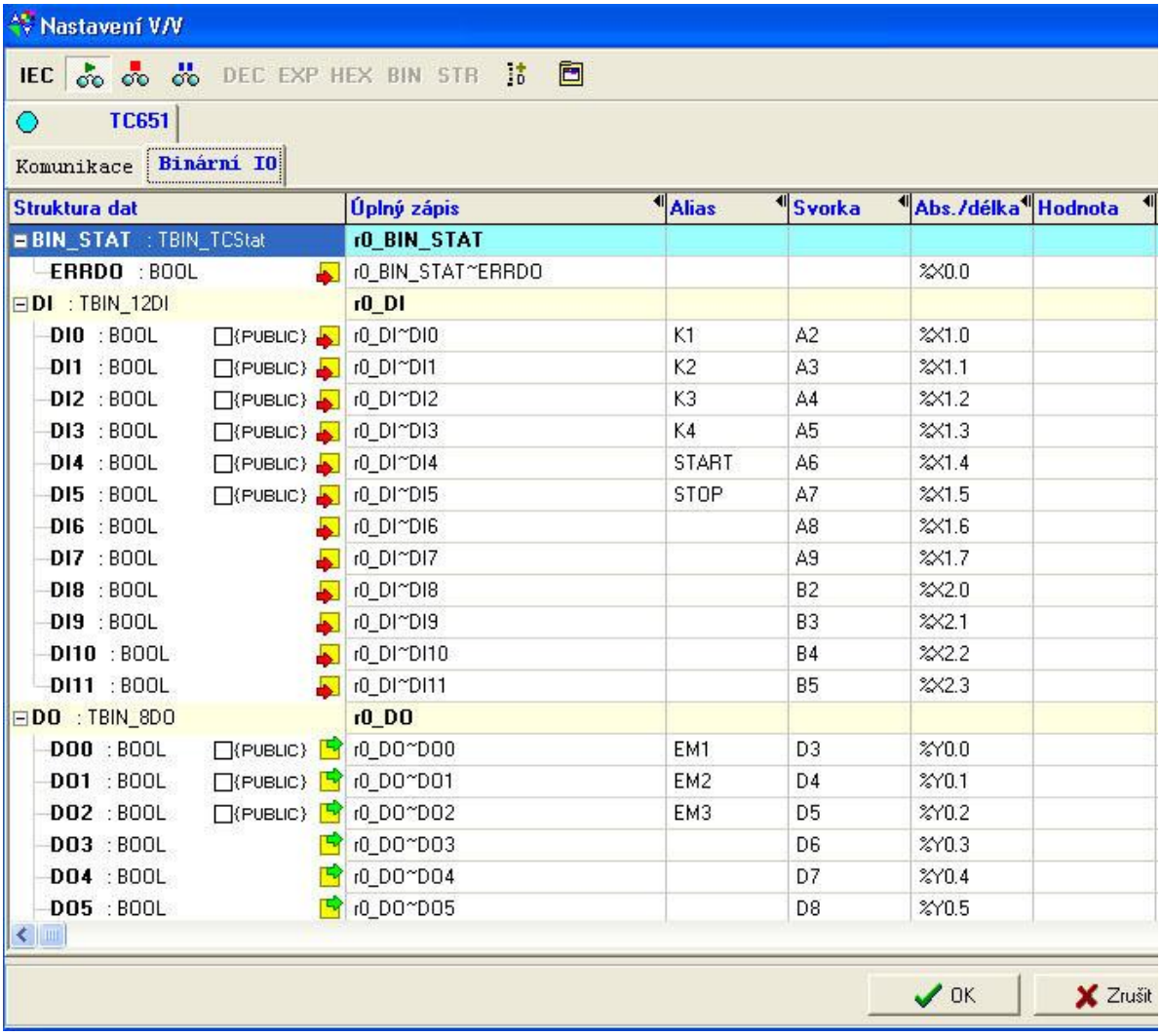
Zadání je graficky znázorněno na obrázku



obr 30 Zadání příkladu

3.1. Realizace příkladu v jazyce ST

Podle známého postupu založíme projektovou skupinu např. s názvem suport_kap5 a projekt, který nazveme sup5_ST. Jako HW platformu zvolíme kompaktní PLC TC651.
Založíme program, zvolíme programovací jazyk ST a pojmenujeme instanci programu.
Navážeme spojení s PLC pomocí sériového komunikačního kanálu CH1, nebo rozhraním ETHERNET a otestujeme adresaci vstupů a výstupů a deklarujeme jim symbolická jména.
Pomocí nástroje Nastavení V/V se nejprve pokusíme zapsat do výstupů EM1, EM2, EM3 a tím "rozjet" suport.



Struktura dat		Úplný zápis	Alias	Svorka	Abs./délka	Hodnota
BIN_STAT : TBIN_TCStat		r0_BIN_STAT				
ERRDO : BOOL		r0_BIN_STAT~ERRDO			%X0.0	
DI : TBIN_12DI		r0_DI				
DI0 : BOOL	{PUBLIC}	r0_DI~DI0	K1	A2	%X1.0	
DI1 : BOOL	{PUBLIC}	r0_DI~DI1	K2	A3	%X1.1	
DI2 : BOOL	{PUBLIC}	r0_DI~DI2	K3	A4	%X1.2	
DI3 : BOOL	{PUBLIC}	r0_DI~DI3	K4	A5	%X1.3	
DI4 : BOOL	{PUBLIC}	r0_DI~DI4	START	A6	%X1.4	
DI5 : BOOL	{PUBLIC}	r0_DI~DI5	STOP	A7	%X1.5	
DI6 : BOOL		r0_DI~DI6		A8	%X1.6	
DI7 : BOOL		r0_DI~DI7		A9	%X1.7	
DI8 : BOOL		r0_DI~DI8		B2	%X2.0	
DI9 : BOOL		r0_DI~DI9		B3	%X2.1	
DI10 : BOOL		r0_DI~DI10		B4	%X2.2	
DI11 : BOOL		r0_DI~DI11		B5	%X2.3	
DO : TBIN_8DO		r0_DO				
DO0 : BOOL	{PUBLIC}	r0_DO~DO0	EM1	D3	%Y0.0	
DO1 : BOOL	{PUBLIC}	r0_DO~DO1	EM2	D4	%Y0.1	
DO2 : BOOL	{PUBLIC}	r0_DO~DO2	EM3	D5	%Y0.2	
DO3 : BOOL		r0_DO~DO3		D6	%Y0.3	
DO4 : BOOL		r0_DO~DO4		D7	%Y0.4	
DO5 : BOOL		r0_DO~DO5		D8	%Y0.5	

obr 31 Adresace V/v

Zároveň sledujeme stav vstupů, kam zapisují snímače K1, K2, K3, K4.
Výsledek adresace vidíme na obr. 31.

Algoritmus můžeme realizovat v jazyku ST třemi příkazovými řádky, kde zapíšeme logické výrazy pro výstupy EM1, EM2, EM3.

1: prgMain.ST | 2: sup5_ST.mcf

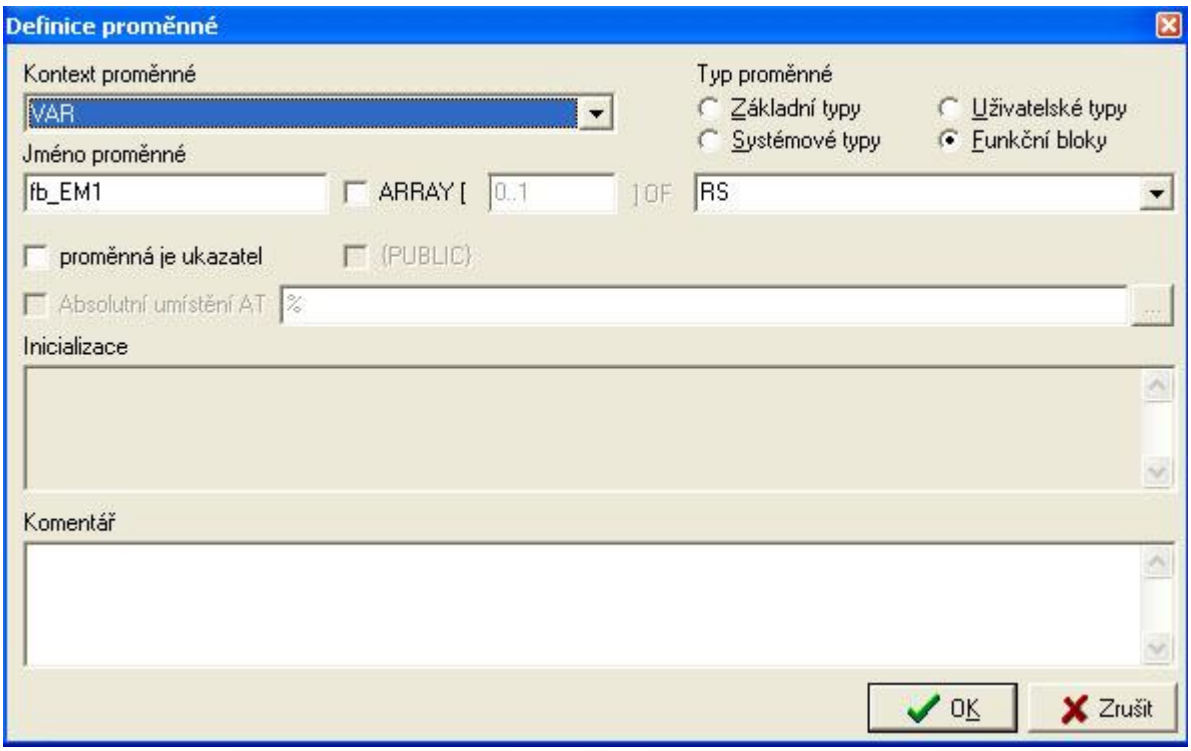
```
PROGRAM prgMain
  EM1:=((START AND K1) OR EM1) AND NOT K4 AND NOT STOP;
  EM2:=(K4 OR EM2) AND NOT K1 AND NOT STOP;
  EM3:=((K2 AND EM1) OR EM3) AND NOT K3 AND NOT STOP;
  //vnitřní závorky jsou jen pro přehlednost, platí priorita operátorů
END_PROGRAM
```

obr 32 Logické rovnice

Projekt přeložíme příkazem PROGRAM/PŘELOŽIT, vyšleme a spustíme pomocí příkazu PLC/RUN. Funkčnost programu ověříme na modelu hydraulické posuvové jednotky. Kdo nemá model k dispozici, může jeho funkci simulovat nástrojem Nastavení V/V, do snímačů polohy K1 až K4 musí však zapisovat ručně.

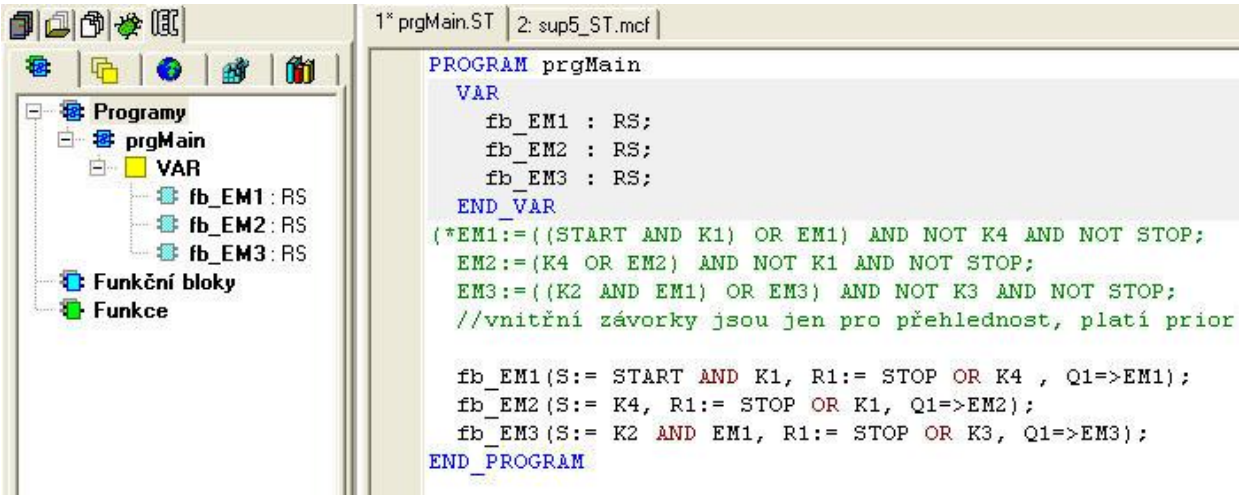
Další možností, jak realizovat zadaný algoritmus je použití funkčních bloků. Funkční bloky, jak víme z úvodních kapitol jsou programové organizační jednotky, které mohou mít instance v programu. Princip jejich volání připomíná práci s podprogramy, pomocí proměnných VAR_INPUT, VAR_OUTPUT je můžeme parametrizovat. Funkční bloky můžeme vytvářet sami v některém z programovacích jazyků, případně můžeme využívat funkční bloky již připravené. Mezi ně patří RS klopné obvody, čítače, časovače a další. Pro náš program použijeme funkční blok realizující funkci klopného obvodu RS. Ovládáme 3 výstupy, potřebujeme tedy 3 instance klopných obvodů.

Instanci funkčního bloku vytvoříme stejným způsobem jako lokální proměnnou. Klikneme pravým tačítkem myši v IEC manažeru na program a zvolíme přidat proměnnou, definici proměnné provedeme dle obr. 33.



obr 33 Vytvoření instance funkčního bloku

Stejným způsobem vytvoříme ostatní instance. Náš program bude pak tvořen pouze voláním těchto bloků s příslušnými vstupními a výstupními parametry.
Přitom využijeme služeb pomocníka pro práci s proměnnými. Napíšeme část jména instance a stiskneme klávesy **CTRL + mezerník**. Systém nám nabídne všechny proměnné začínající těmito znaky, vybereme instanci, kterou chceme použít a povrdíme. Systém nám nabídne další možnosti, ze kterých vybereme **kompletní volání**. Potom už jen dosadíme za vstupní a výstupní parametry.
Vytvořený program vidíme na obr. 34 (předchozí program jsem označil jako komentář - není

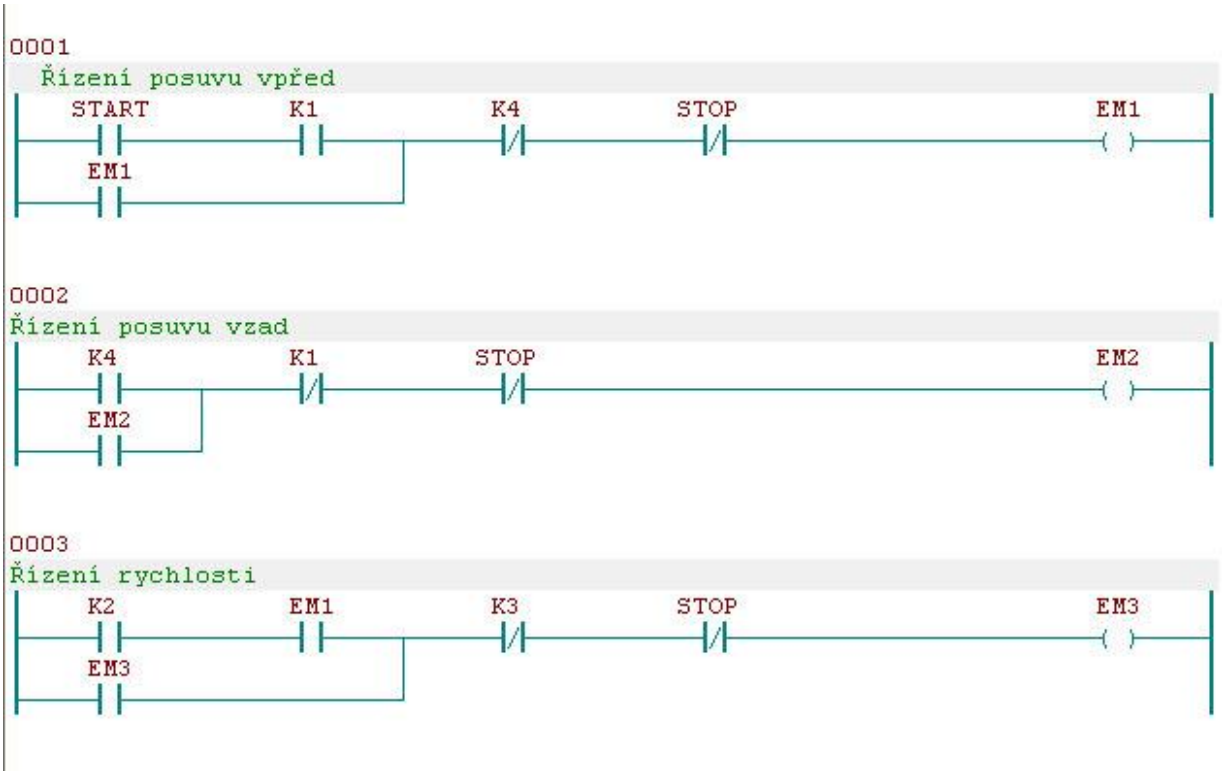


obr 34 Použití funkčních bloků

překládán).

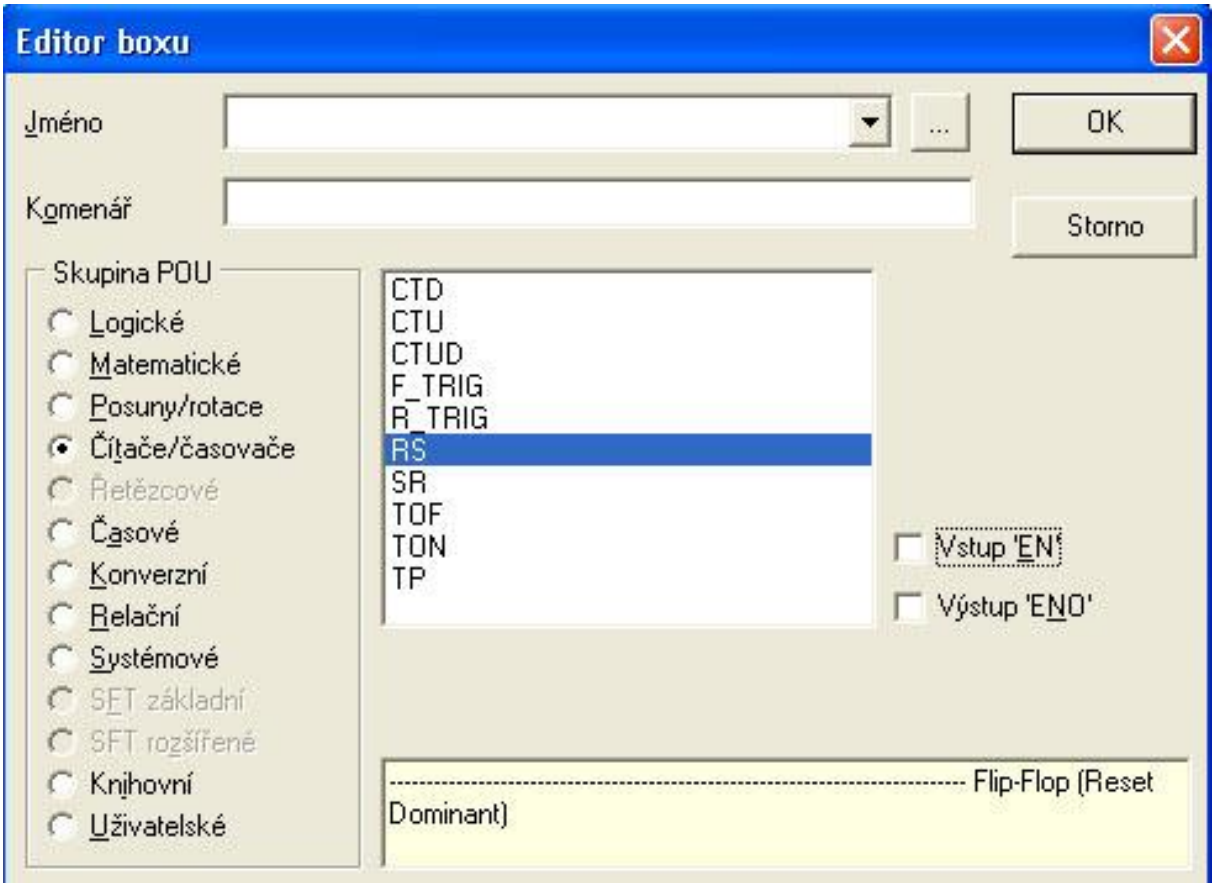
3.2. Realizace příkladu v jazyce LD

Nyní naprogramujeme stejný příklad v grafickém jazyce kontaktních schémat.
V rámci projektové skupiny založíme nový projekt a postupujeme stejně, jako v předchozí kapitole. Výsledný program vidíme na obr. 35.



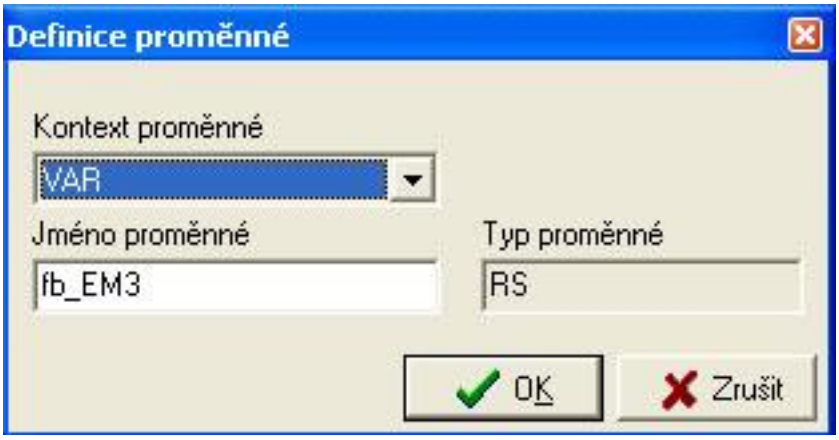
obr 35 Program v jazyku LD

Podobně, jako v jazyku ST, můžeme i zde použít funkční bloky Při vkládání klopného obvodu zvolíme položku **Box** a vybereme z nabídky **Čítače / časovače** položku **RS**. Zrušíme povolení vstupu a výstupu a potvrdíme (obr. 36)



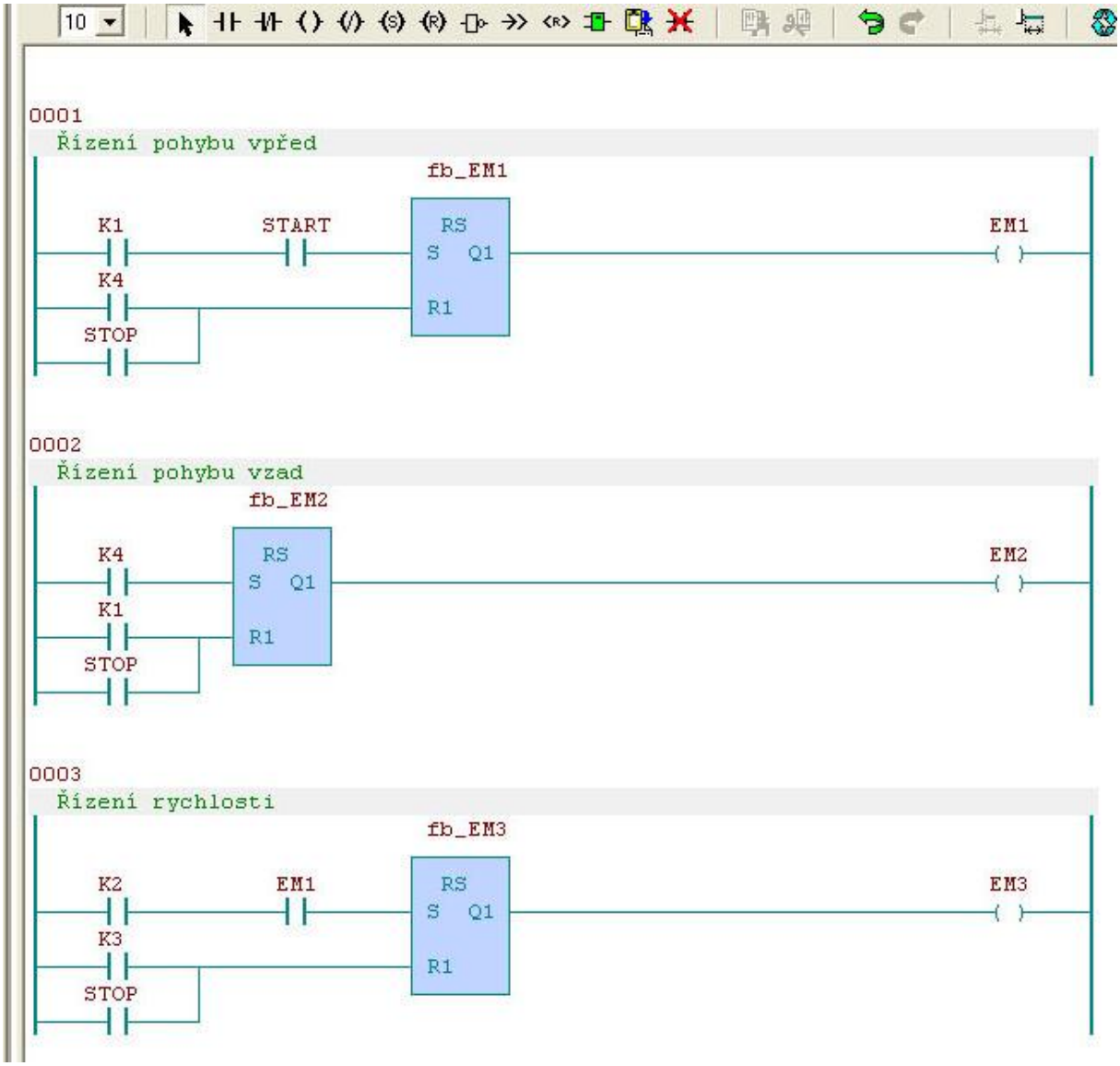
obr 36 Vytvoření RS klopného obvodu

Následující okno nás vyzve k vytvoření a pojmenování instance funkčního bloku.



obr 37 Definice instance funkčního bloku

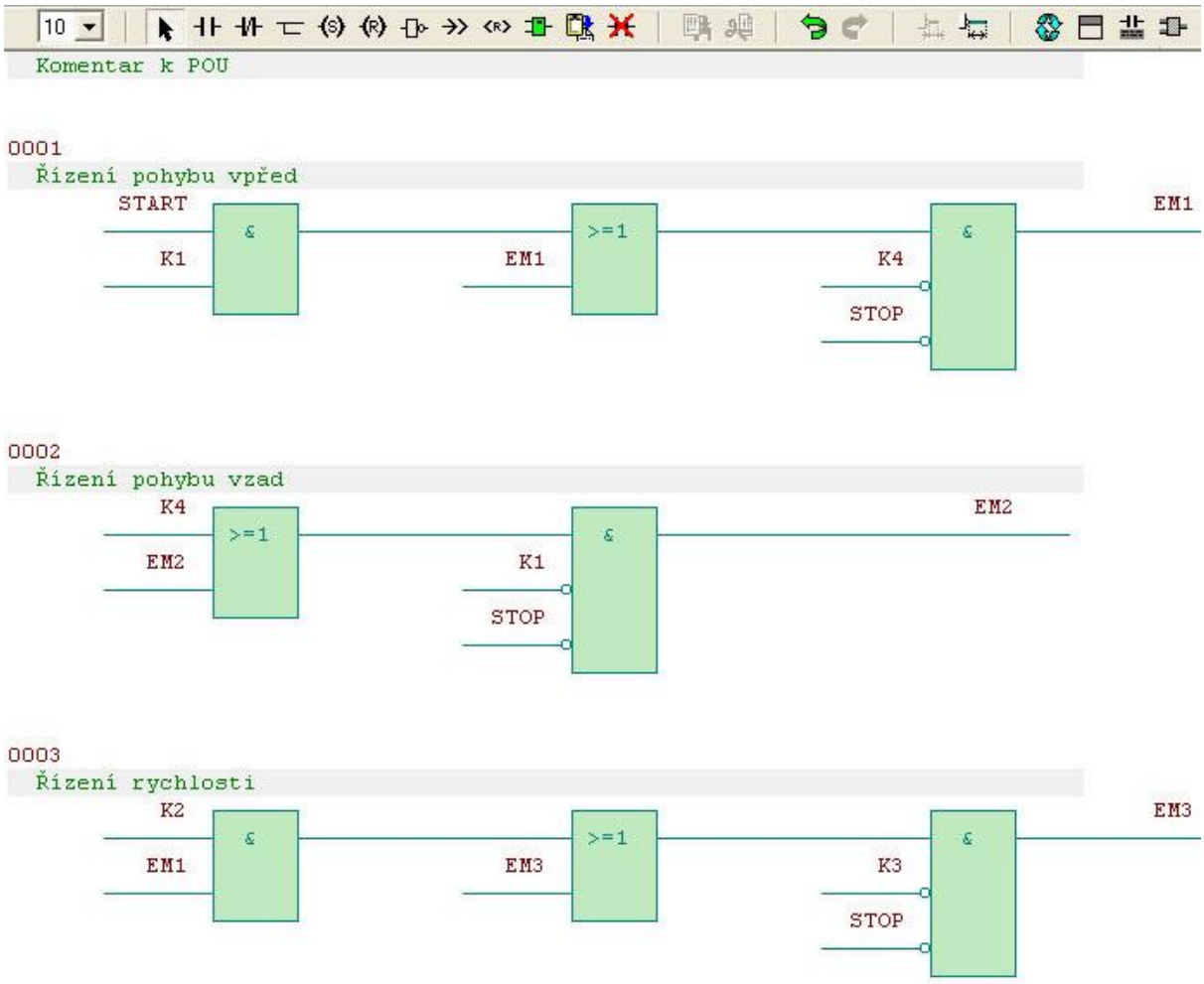
Další postup již známe z předchozí kapitoly je zcela intuitivní. Výsledný program vidíme na obrázku 38



obr 38 Program v jazyku LD s využitím funkčních bloků

3.3. Realizace příkladu v jazyce FBD

Programování ve FBD ve velice podobné způsobu programování v LD.



obr 39 Program v jazyku FBD

Projekt přeložíme, vyšleme do PLC a monitorujeme obdobným způsobem jako v LD.
Na závěr projektovou skupinu archivujeme postupem popsaným v třetí kapitole.

4. Příkazy jazyka ST a jejich použití

V dnešní lekci se budeme zabývat programovacím jazykem ST a jeho příkazy. Předvedeme si jejich použití na příkladu časového řízení modelu křižovatky.

4.1. Jazyk strukturovaného textu ST

Jazyk strukturovaného textu je jedním z jazyků definovaných normou IEC 61 131-3. Je to velmi výkonný vyšší programovací jazyk, který má kořeny ve známých jazycích Ada, Pascal a C. Je objektově orientován a obsahuje všechny podstatné prvky moderního programovacího jazyka, včetně větvení (IF-THEN-ELSE a CASE OF) a iterační smyčky (FOR, WHILE a REPEAT). Tyto prvky mohou být vnořovány. Tento jazyk je vynikajícím nástrojem pro definování komplexních funkčních bloků.
Algoritmus zapsaný v jazyce ST lze rozdělit na jednotlivé příkazy (statements). Příkazy se používají pro výpočet a přiřazení hodnot, řízení toku vykonávání programu a pro volání resp. ukončení POU. Část příkazu, která vypočítává hodnotu, je nazývána výraz. Výrazy produkují hodnoty nezbytné pro provádění příkazů.

Výrazy

Výraz je konstrukce, ze které se po vyhodnocení vygeneruje hodnota odpovídající některému z datových typů, které byly definovány v druhé kapitole.
Výraz se skládá z operátorů a operandů. Operandem může být literál, proměnná, volání funkce nebo jiný výraz. Operátory jazyka strukturovaného textu ST jsou přehledně uspořádány v tabulce na obr. 40.

Operátor	Operace	Priorita
( )	Závorky	Nejvyšší
**	Umocňování	
- NOT	Znaménko Doplněk	
* / MOD	Násobení Dělení Modulo	
+ -	Sčítání Odčítání	
<, >, <= ,>=	Porovnávání	
= <>	Rovnost Nerovnost	
&, AND	Boolovské AND	
XOR	Boolovské exkluzivní OR	
OR	Boolovské OR	Nejnižší

obr 40 Operátory v jazyku strukturovaného textu ST

Vyhodnocení výrazu spočívá v aplikaci operátorů na operandy a to s ohledem na prioritu vyjádřenou tabulkou na obrázku 40. Operátory s nejvyšší prioritou ve výrazu jsou aplikovány nejdříve, pak následují další operátory směrem k nižší prioritě dokud není vyhodnocování dokončeno.Operátory se stejnou prioritou se vyhodnocují tak jak jsou zapsány ve výrazu směrem odleva doprava.

4.1.1. Souhrn příkazů v jazyce ST

Seznam příkazů jazyka strukturovaného textu ST je souhrnně uveden v tabulce na obrázku 41. Příkazy jsou ukončeny středníkem.Se znakem konec řádku se v tomto jazyku zachází stejně jako se znakem mezery (space).

Příkaz	Popis	Příklad	Poznámka
:=	Přiřazení	A := 22;	Přiřazení hodnoty vypočtené na pravé straně do identifikátoru na levé straně
	Volání funkčního bloku	**InstanceFB(** par1 := 10, par2 := 20);	Volání funkčního bloku s předáváním parametrů
IF	Příkaz výběru	**IF** A > 0 **THEN** B := 100; **ELSE** B := 0; **END_IF;**	Výběr alternativy v podmíněný výrazem BOOL
CASE	Příkaz výběru	**CASE** kod **OF** 1 : A := 11; 2 : A := 22; **ELSE** A := 99; **END_CASE;**	Výběr bloku příkazů podmíněný hodnotou výrazu „kod“

FOR	Iterační příkaz smyčka FOR	**FOR** i := 0 **TO** 10 **BY** 2 **DO**
		j := j + i;
		END_FOR;
			Vícenásobná smyčka bloku příkazů s počáteční a koncovou podmínkou a hodnotou inkrementu

WHILE	Iterační příkaz smyčka WHILE	**WHILE** i > 0 **DO**
		n := n * 2;
		END_WHILE;
			Vícenásobná smyčka bloku příkazů s podmínkou ukončení smyčky na začátku

REPEAT	Iterační příkaz smyčka REPEAT	**REPEAT**
		k := k + i;
		UNTIL i < 20;
		END_REPEAT;
			Vícenásobná smyčka bloku příkazů s podmínkou ukončení smyčky na konci

EXIT	Ukončení smyčky	**EXIT;**	Předčasné ukončení iteračního příkazu
RETURN	Návrat	**RETURN;**	Opuštění právě vykonávané POU a návrat do volající POU
;	Prázdný příkaz	;;	

obr 41 Seznam příkazů jazyka strukturovaného textu ST

Příkaz přiřazení

Přiřazovací příkaz nahrazuje aktuální hodnotu jednoduché nebo složené proměnné výsledkem, který vznikne po vyhodnocení výrazu. Přiřazovací příkaz se skládá z odkazu na proměnnou na levé straně, za ním následuje operátor přiřazení **:=** , za kterým je uveden výraz, který se má vyhodnotit.
Jednoduché příkazy přiřazení jsme již v předchozích kapitolách používali k zápisu výsledku logického výrazu do výstupní proměnné. Příkaz přiřazení může přiřadit, jako v našich příkladech, jednoduchou proměnnou ale poradí si také se složitou datovou strukturou

Příkaz volání funkčního bloku

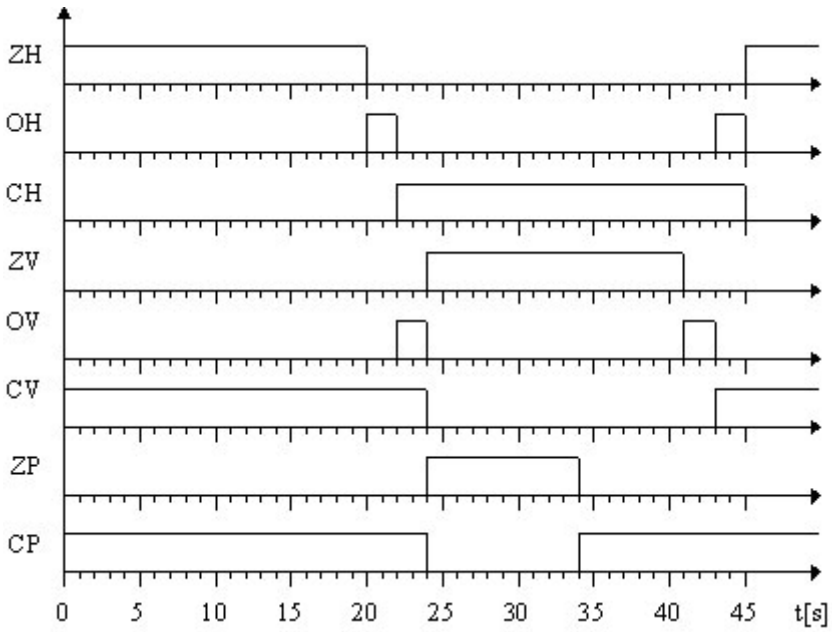
Funkční bloky se volají příkazem, který se skládá ze jména instance funkčního bloku, za kterým následuje seznam pojmenovaných vstupních parametrů s přiřazenými hodnotami.Na pořadí, v němž jsou parametry v seznamu při volání funkčního bloku uvedeny, nezáleží. Při každém volání funkčního bloku nemusí být přiřazeny všechny

vstupní parametry. Pokud nějakému parametru není přiřazena hodnota před voláním funkčního bloku, pak se použije hodnota naposledy přiřazená (nebo hodnota počáteční, pokud nebylo ještě provedeno žádné přiřazení). Volání standardního funkčního bloku jsme používali v páté kapitole pro realizaci RS klopného obvodu

Příkaz IF
Příkaz IF specifikuje, že se má provádět skupina příkazů jedině v případě, že se přiřazený boolovský výraz vyhodnotí jako pravdivý (TRUE). Pokud je podmínka nepravdivá, pak se neprovádí buď žádný příkaz nebo se provádí skupina příkazů, které jsou uvedeny za klíčovým slovem ELSE (nebo za klíčovým slovem ELSIF, pokud jemu přiřazená podmínka je pravdivá).
Příkaz začíná klíčovým slovem **IF**, za kterým následuje boolovský výraz (podmínka) a klíčové slovo **THEN** s příkazy, prováděnými v případě pravdivosti podmínky.
Volitelným klíčovým slovem je **ELSIF** s podmínkou a příkazy, které se provedou, je-li podmínka za **IF** nepravdivá a podmínka **ELSIF** pravdivá. konstrukcí **ELSIF** může být libovolné množství.
Konstrukce s klíčovým slovem **ELSE** je podobná konstrukci **ELSIF**, chybí podmínka a příkazy se provádějí v případě, jsou-li všechny předchozí pomínky vyhodnoceny jako nepravdivé.
Příkaz IF je velmi silný a často používaný nástroj programátora. Jeho použití přibližuje programování PLC k programování ve standardních jazycích, které studenti již většinou znají, řešené algoritmy pak lépe chápou. Pokusme se pomocí tohoto příkazu naprogramovat časové řízení modelu křižovatky.

4.2. Řízení modelu křižovatky

Zadání
Napište program pro řízení křižovatky podle časového režimu z obr 43:

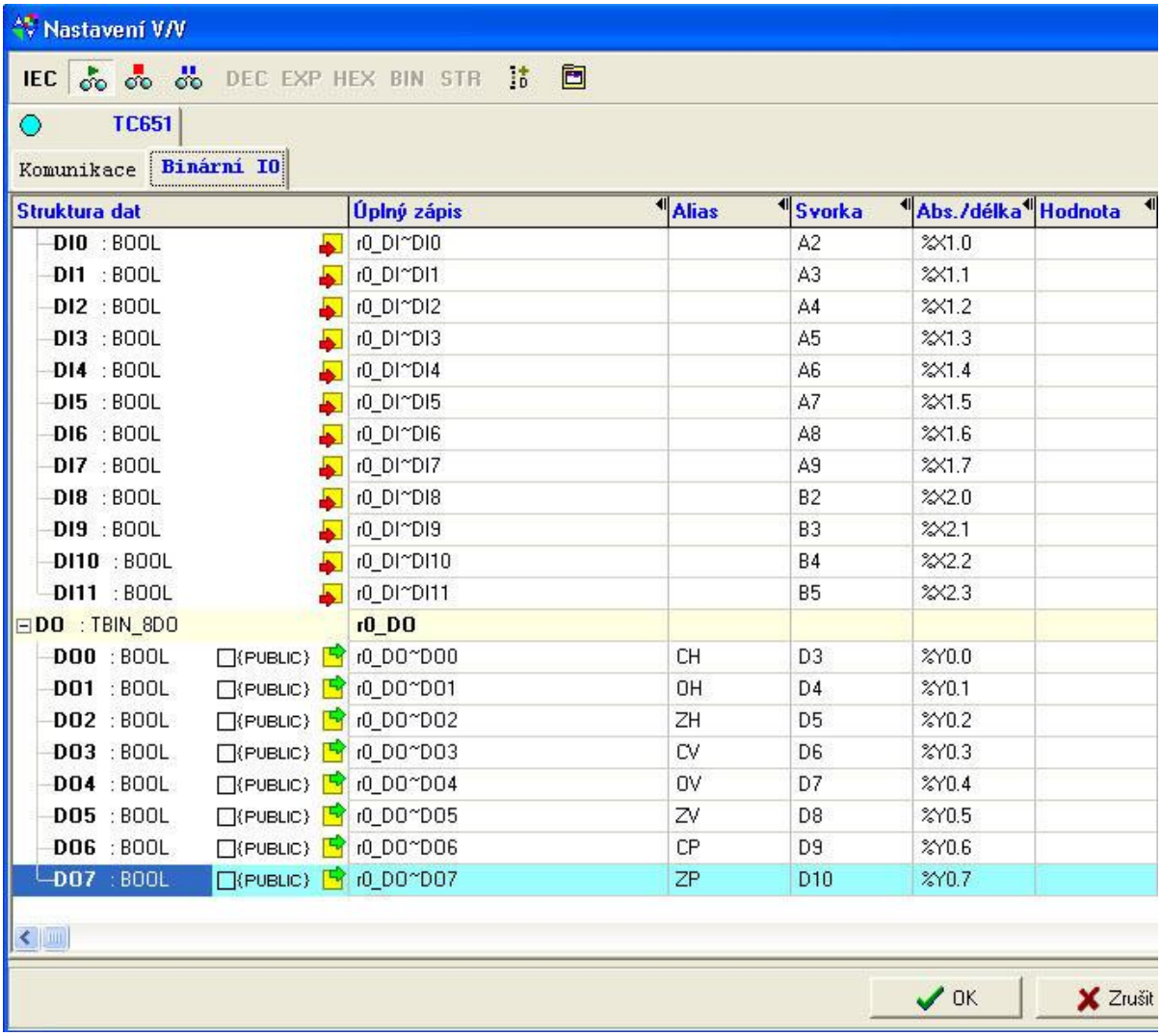


***obr 43** Časový režim řízení křižovatky*

Řešení
V tomto příkladu bude patrně nejsnazší použít synchronní algoritmus - všechny akce budou tedy odvozeny od jedné časoměrné proměnné, která se bude měnit cyklicky v intervalu 0 až 45s. Časoměrnou proměnnou vytvoříme časovačem (například TON), který budeme vždy po dosažení předvolby nulovat. Příkazem IF budeme testovat hodnotu časoměrné proměnné a ovládat digitální výstupy PLC, řídící křižovatku.

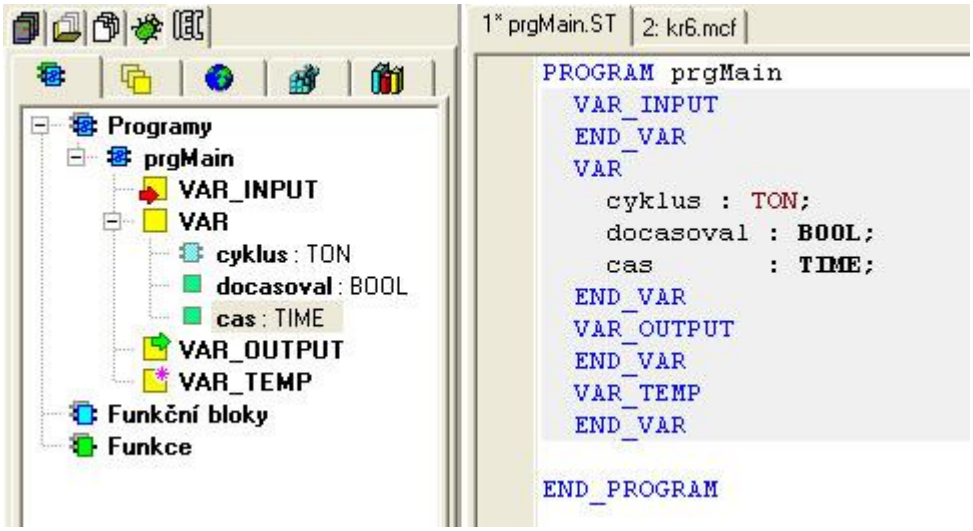
Založíme si tedy novou skupinu projektů, projekt, vytvoříme POU typu PROGRAM a jeho instanci. Jako programovací metodu zvolíme jazyk ST.

Nejprve musíme otestovat adresaci výstupů (vstupy dnes nebudeme potřebovat žádné). Pomocí nástroje **Nastavení V/V** postupně zapisujeme do jednotlivých výstupů a sledujeme, které LED modelu se rozsvěcují. Proměnné si pojmenujeme do sloupce **Alias**, v mé konfiguraci to vypadá asi takto:



obr 44 Adresace LED křižovatky

Nyní si připravíme lokální proměnné, které budeme potřebovat. Založíme instanci standardního funkčního bloku časovače TON a časoměrnou proměnnou a pomocnou proměnnou pro příznak dokončení cyklu (nejsou nutné, používám je pouze pro větší srozumitelnost řešení).



obr 45 Založení lokálních proměnných

Prvním naším příkazem bude volání funkčního bloku časovače TON. Funkci námi použitého časovače můžeme přirovnat k relé se zpožděným zapnutím. Má dva vstupní a dva výstupní parametry:

vstupní parametry
IN - řídicí boolovská proměnná pro spuštění časovače
PT - předvolba proměnná typu TIME

výstupní parametry
Q - výstupní boolovská proměnná, která nabývá hodnoty TRUE po dosažení předvolby
ET - proměnná typu TIME udávající aktuální stav časovače

Napíšeme prvnípísmena jména instance časovače a stiskneme klávesy **CTRL + MEZERNIK**. Z nabídky vybereme naší instanci, potvrdíme, zvolíme kompletní volání a dasadíme vstupní a výstupní parametry.
Příkazový řádek bude vypadat asi takto:
cyklus(IN := NOT docasoval , PT := T#45s , Q => docasoval , ET => cas);
Touto konstrukcí dosáhneme toho, že časoměrná proměnná **cas** se bude cyklicky měnit v intervalu 0 až 45s (dá se to samozřejmě naprogramovat i jinak).
Jádrem programu bude jediný příkaz **IF**, který bude testovat stav proměnné **cas** a provádět obsluhu příslušných výstupů.
Základní kostru příkazu IF vytvoříme pomocníkem, kterého vyvoláme současným stiskem kláves **CTRL + J**. Výsledná podoba programu je na obr. 46.

1* prgMain.ST | 2: kr6.mcf

```
PROGRAM prgMain
  VAR_INPUT
  END_VAR
  VAR
    cyklus : TON;
    docasoval : BOOL;
    cas       : TIME;
  END_VAR
  VAR_OUTPUT
  END_VAR
  VAR_TEMP
  END_VAR
  cyklus(IN := NOT docasoval, PT := T#45s, Q =>docasoval ,ET => cas);
  IF cas >= T#0s AND cas < T#20s THEN
      CH:=false; OH:=false; ZH:=true; CV:=true; OV:=false; ZV:=false;
      CP:=true; ZP:=false;
  ELSIF  cas >= T#20s AND cas < T#22s THEN
      CH:=false; OH:=true; ZH:=false; CV:=true; OV:=false; ZV:=false;
      CP:=true; ZP:=false;
  ELSIF  cas >= T#22s AND cas < T#24s THEN
      CH:=true; OH:=false; ZH:=false; CV:=true; OV:=true; ZV:=false;
      CP:=true; ZP:=false;
  ELSIF  cas >= T#24s AND cas < T#34s THEN
      CH:=true; OH:=false; ZH:=false; CV:=false; OV:=false; ZV:=true;
      CP:=false; ZP:=true;
  ELSIF  cas >= T#34s AND cas < T#41s THEN
      CH:=true; OH:=false; ZH:=false; CV:=false; OV:=false; ZV:=true;
      CP:=true; ZP:=false;
  ELSIF  cas >= T#41s AND cas < T#43s THEN
      CH:=true; OH:=false; ZH:=false; CV:=false; OV:=true; ZV:=false;
      CP:=true; ZP:=false;
  ELSIF  cas >= T#43s  THEN
      CH:=true; OH:=true; ZH:=false; CV:=true; OV:=false; ZV:=false;
      CP:=true; ZP:=false;
  END_IF;
END_PROGRAM
```

obr 46 Program řízení křižovatky v jazyku ST

Projekt přeložíme, vyšleme do PLC a vyzkoušíme na modelu křižovatky. Projektovou skupinu archivujeme postupem popsaným v třetí kapitole.

5. Použitá literatura:

1. http://www.tecomat.cz

6. Obsah