3

3. ZÁKLADNÍ INSTRUKCE JAZYKA TECHNOL

Jazyk TECHNOL je určen pro efektivní programování interfejsu pro systémy CNC8x9 a CNC872. Jazyk používá výhradně symbolických adres a to i při práci s jednotlivými bity paměti.

3.1 Zápis zdrojového programu

Zdrojové programy interfejsu lze napsat v prostředí ladícího programu Wintechnol nebo v libovolném textovém ASCII editoru. Programování je prováděno v mnemonickém kódu. Pro zápis programu platí podobná pravidla jako pro zápis zdrojového programu assembleru.

Programový řádek obsahuje:

a) Návěští
Je nepovinné a určuje symbolicky adresu místa v paměti pro proměnnou nebo na kterou má být např. proveden skok v programu. Návěští může obsahovat max. 31 znaků, t.j. písmen, číslic a tří speciálních znaků:

_ podtrhovátko
? otazník
@ "zavináč"

Prvním znakem nesmí být číslice. Návěští musí být ukončeno dvojtečkou. Návěští může být uvedeno na samostatném řádku, přičemž se vztahuje k nejbližšímu následně uvedenému řádku s operačním kódem, konstantou nebo alokací paměti.

b) Operační kód
Je mnemonický zápis kódu příslušné instrukce. Jako operační kód jsou povoleny instrukce, uvedené v další části návodu. Kromě toho je umožněno používat i instrukce assembleru 80386 i když pro zápis programu interfejsu to není nutné.

c) Operand
Jednotlivé instrukce mohou být bez operandu (např. instrukce BCD) nebo většinou s jedním operandem (např. `LDR ALFA`), případně některé instrukce mohou mít víc operandů oddělených čárkami.

d) Komentář
Text, uvedený za středníkem se považuje za komentář a ignoruje se. Komentář může být uveden na samostatném řádku nebo na řádku společně s kódem.

Příklad:

```
                              ;toto je komentar
PAM12:      LDR   ALFA        ;toto je komentar
```

3.2 Pracovní registry jazyka TECHNOL

Instrukce jazyka TECHNOL využívají tyto registry:

a) Jednobitový takzvaný **registr logických operací**, který se bude v dalším textu označovat zkratkou **RLO**. (Fyzicky se jedná o bit s váhou 40h - registru AH mikroprocesoru.)

b) **Datový registr** , který se bude v dalším textu označovat zkratkou **DR**. Instrukce mohou pracovat s datovým registrem se šířkou slova 8, 16, 32, nebo 64 bitů. Pro operce v reálných číslech se používá 64 bitový registr `DR_REAL`.
(Fyzicky se jedná o registr ECX mikroprocesoru. V případě rozšířeného 64 bitového DR registru jsou to registry ECX a EDX).

c) Zásobník - jedná se o 8 buňek typu WORD.

Pokud se používají pouze instrukce jazyka TECHNOL, není fyzická reprezentace registrů pro programátora důležitá. Registry se musí respektovat pouze při eventuelním programování částí programu v assembleru 80386.

Změna hodnoty v registru RLO neovlivní datový registr DR a obráceně, změna v datovém registru DR neovlivní registr RLO. Jazyk TECHNOL automaticky rozpoznává, kdy se má použít bajtový, wordový nebo double-wordový přístup jak k registru DR, tak k paměti.

3.3 Deklarace paměti

Jazyk TECHNOL umožňuje pracovat s libovolnými adresami v symbolické formě. Jako symbolické adresy lze definovat adresy v programu i adresy bitových, 8-bitových (typu BYTE), 16-bitových (typu WORD) a 32-bitových (typu DWORD) buněk paměti RAM. Symbolická adresa je definována jako slovo o maximálně 31 znacích (délka není omezená, ale významných je prvních 31 znaků), které může obsahovat jak písmena, tak čísla, přičemž jako první musí být vždy uvedeno písmeno. Toto slovo však nesmí být uvedeno v seznamu klíčových slov, která mají již předem daný význam. Seznam klíčových slov je uveden v příloze.
Přiřazení symbolické adresy určité fyzické adrese lze provést dvěma způsoby. Jedná-li se o deklaraci adresy (místa) v programu nebo deklaraci slova v paměti RAM, postačí napsat za příslušný symbol dvojtečku a adresa je pak dána polohou tohoto symbolu nebo-li adresa se vztahuje na instrukci následující za dvojtečkou.

Příklad:
V programu, který je vyjádřen instrukcemi 1 až 5, chceme definovat adresu instrukce 2 a 3 symboly ALFA a BETA.

```
            INSTRUKCE         1
ALFA:       INSTRUKCE         2
BETA:       INSTRUKCE         3
            INSTRUKCE         4
            INSTRUKCE         5
```

3.4 Definice bitových a datový proměnných a konstant

instrukce	**DFM**

funkce **definice bitu v paměti**

syntax **name: DFM [bit0],[bit1],,,,,,[bit7]**

parametry	**„bit0,..bit7"**	**symbolické názvy bitových proměnných**
	„name"	**název slabiky BYTE**

Chceme-li symbolicky definovat jednotlivé bity paměti RAM, použijeme speciální instrukce **DFM**. Jedna instrukce DFM definuje vždy v místě svého zápisu jednu slabiku paměti (8 bitů). Celou slabiku můžeme symbolicky pojmenovat pomocí symbolu, zapsaného před tuto instrukci a zakončeného dvojtečkou. Takto definovaná slabika umožňuje kromě bitového i bajtový přístup. Symboly, které jsou uvedeny za příkazem DFM, deklarují pak jednotlivé bity této slabiky paměti. První symbol náleží bitu d0, druhý bitu d1 atd., až osmý symbol náleží bitu d7. Symboly musí být odděleny čárkami. V případě, že některý bit definovat nechceme, můžeme příslušný symbol vynechat. Čárky však uvedeny být musí!

Jednotlivé vstupy a výstupy jednotek periferních obvodů jsou snímány a plněny po osmicích. Abychom si tvorbu programu PLC usnadnili, budeme pracovat s těmito vstupy a výstupy výhradně tímto způsobem. Na začátku programu přečteme pomocí speciální instrukce všechny vstupy do paměti RAM a na konci programu PLC naopak vymezené paměti RAM zapíšeme do výstupů. Jednotlivé vstupy či výstupy pak můžeme deklarovat a obsluhovat stejně jako bity paměti RAM.

instrukce	**DS 1** **DS 2** **DS 4** **DS n**

funkce	**DS 1**	**vymezení paměti o délce 1 BYTE**
	DS 2	**vymezení paměti o délce 1 WORD**
	DS 4	**vymezení paměti o délce 1 DWORD**
	DS n	**vymezení paměti o délce n bajtů**

syntax
name: DS 1
name: DS 2
name: DS 4
name: DS n

parametry	**„1,2,4,n"**	**počet bajtů paměti**
	„name"	**název datové proměnné**

Instrukce **DS** se používá pro definování proměnných a inicializaci paměti. Operandem se deklaruje délka vymezené paměti v bajtech. Instrukce DS 1 s návěštím proměnné deklaruje tuto proměnnou jako BYTE, instrukce DS 2 deklaruje proměnnou jako WORD a DS 4 deklaruje proměnnou jako DWORD.

instrukce	**EQUI**

funkce **definice konstanty**

syntax **EQUI const, immed**

1.parametr **„const"** **název konstanty**
2.parametr **„immed"** **hodnota konstanty**

Instrukce **EQUI** definuje konstanty použité v programu. První parametr je symbolický název konstanty a druhý parametr je jeho hodnota. Hodnota konstanty může být zapsána :

```
1. dekadicky            např. 123, 54213
2. hexadecimálně        např. 0F8H, 15H
3. znakově              např. 'A', '8'
```

Příklad:
V paměti RAM chceme definovat slabiku GAMA a její jednotlivé bity d0 až d7 jako symboly GAMA0 až GAMA7. Na adrese DELTA chceme symbolicky definovat bit d0 = DELTAM, bit d2 = DELTAG a bit d7 = DELTAT.

```
GAMA:       DFM         G0,G1,G2,G3,G4,G5,G6,G7
DELTA:      DFM         DELTAM,,DELTAG,,,,,DELTAT
```

Příklad:
Pro bajtovou proměnnou ALFA vymezte délku 1 byte.
Pro double-wordovou proměnnou BETA vymezte délku 4 byte.
Pro proměnnou GAMA vymezte pole o délce 30 byte.
Definujte konstantu DELTA s hodnotou 1000.

```
ALFA:       DS          1
BETA:       DS          4
GAMA:       DS          30
EQUI        DELTA, 1000
```

3.5 Logické operace s bity paměti a RLO

instrukce	**LDR**

funkce **čtení bitu paměti do RLO /LOAD RLO/**

syntax
```
LDR     [-] bit
LDR     [-] [adr.]  bit   složitější způsoby adresace
LDR     [-] [$+xx.] bit
```

parameter **„bit"** **symbolický název bitu**

Instrukce **LDR** provádí plnění registru logických operací RLO bitem zadané paměti.

Instrukce LDR má povinný operand. Tímto operandem je symbolický název bitu paměti RAM, definovaný pomocí instrukce DFM. Je-li tento symbolický název bitu paměti uveden bez znaménka nebo se znaménkem plus, je příslušný bit čten přímo. Je-li před symbolickým názvem bitu znaménko -, čte se příslušný bit paměti negovaný.

Složitější způsoby adresace bitu:

V případě, že je potřeba načíst bit z jiného místa paměti, než je daný bit definovaný, uvede se adresa paměti před názvem bitu a oddělí se tečkou. Místo adresy místa je možno použít i název bitu, který je tam definovaný, způsoby indexace pomocí [BX] a prvky struktury. Tímto je například umožněna práce s rozsáhlejším bitovým polem. Když potřebujeme načíst bit z místa o několik bajtů dál něž je bit definován (například o 12), můžeme místo vlastního názvu použít znak $+ ($ + 12). Složitější způsoby adresace bitu je možno použít u instrukcí LDR, LA, LO, LX a pro instrukce WR, FL1 a FL.
Složitější způsoby adresace bitu se používají i v případě, když je potřeba načíst bit s příslušnou váhou s libovolného místa paměti. V PLC programu se formálně nadeklarují univerzální názvy bitů, například B0,B1,...,B7, které se pak používají pro přístup k libovolnému místu paměti.

Příklady adresace bitu:
```
DFM   B0,B1,B2,B3,B4,B5,B6,B7 ;formální definice bitů

LDR   ALFA                    ;načtení bitu ALFA z paměti, kde je definován
LDR   BUN5.ALFA               ;z buňky BUN5 se načte bit ze stejné pozice,
                              ;jak je původně definován bit ALFA.
LDR   BUN5.B4                 ;načtení 4.bitu (váha 10h) z buňky BUN5
LDR   -(BUN5+3).ALFA          ;načte se negace bitu z buňky BUN5+3
LDR   BUN5[BX].ALFA           ;použití indexace (viz kapitola.18)
LDR   -(BUN5[BX]-6).ALFA      ;další kombinace
LDR   (BUN5[BX]+PRVNI+2).ALFA ;PRVNI je prvek struktury
LDR   $+3.ALFA                ;načtení bitu ALFA z pozice o 3 bajty dál
```

3.6 Princip zásobníku a koncových instrukcí

Jsou-li dvě nebo více instrukcí LDR zapsány v programu za sebou, aniž byl mezi nimi registr log. operací nějak využit (pro zápis do paměti či podmíněný skok), ukládá se před další instrukcí LDR předchozí stav RLO do zásobníku. S takto uloženými hodnotami lze potom pracovat pomocí instrukcí LA, LO nebo LX bez udání operandu. Pomocí zásobníku je takto umožněno programovat závorkové operace.

Vyrovnanost zásobníku se kontroluje na konci každého modulu (viz dále) a na konci každého sekvenčního logického celku. Nevyrovnanost zásobníku ohlásí chybu v úvodní fázi kompilace překladače TECHNOL. Kontrolu vyrovnanosti zásobníku ve fázi odlaďování programu možno vnutit pomocnou instrukcí **CHECK** (viz popis pomocných instrukcí).

Každá logická rovnice naprogramovaná pomocí instrukcí **LDR, LA, LO** a **LX** musí být ukončena tzv. **koncovou instrukcí**, která ukončuje logickou rovnici. Koncová instrukce nuluje vnitřní příznak vnořování do zásobníku, takže u první následující instrukce **LDR** nedojde k uložení RLO do zásobníku.

Koncové instrukce v jazyku TECHNOL jsou:

- WR
- JL0, JL1
- STO1
- FL1
- CONRD
- EX, EX0, EX1, BEX
- TEX0, TEX1
- TIM

instrukce	**LA** **LO** **LX**

funkce	**LA**	**logický součin s RLO /AND/**
	LO	**logický součin s RLO /OR/**
	LX	**nonekvivalence s RLO /XOR/**
syntax 1	**LA**	
	LO	
	LX	
syntax 2	**LA**	**[-] bit**
	LO	**[-] bit**
	LX	**[-] bit**
	LA,LO,LX	**[-] [adr.] bit složitější způsoby adresace**
	LA,LO,LX	**[-] [$+xx.] bit**
parametr	**„bit"**	**symbolický název bitu**

Instrukce **LA**, **LO** a **LX** mají operand volitelný, t.j. může být zadán, ale nemusí. Tímto operandem je symbolický název bitu paměti RAM, definovaný pomocí instrukce DFM. Je-li tento symbolický název bitu paměti uveden bez znaménka nebo se znaménkem plus, je příslušný bit čten přímo. Je-li před symbolickým názvem bitu znaménko -, čte se příslušný bit paměti negovaný.

Jsou-li dvě nebo více instrukcí LDR zapsány v programu za sebou, aniž byl mezi nimi registr log. operací nějak

využit (pro zápis do paměti či podmíněný skok), ukládá se před další instrukcí LDR předchozí stav RLO do zásobníku. S takto uloženými hodnotami lze potom pracovat pomocí instrukcí LA, LO nebo LX bez udání operandu. Instrukce LA provede logický součin naposledy uložené hodnoty zásobníku s RLO a výsledek uloží do RLO. Instrukce LO provede obdobně logický součet a instrukce LX nonekvivalenci. Vyrovnanost zásobníku se kontroluje na konci každého modulu (viz dále) a na konci každého sekvenčně logického celku. Nevyrovnanost zásobníku ohlásí chybu v úvodní fázi kompilace překladače TECHNOL.

Složitější způsoby adresace jsou popsány u instrukce LDR.

instrukce	**CA**
funkce	**negace RLO**
syntax	**CA**

Instrukce **CA** je bez operandu a provádí negaci obsahu registru logických operací RLO. Má-li před touto instrukcí registr log. operací hodnotu 1, po této instrukci bude mít hodnotu 0 a naopak.

instrukce	**EDGE_H** **EDGE_L**	
funkce	**EDGE_H**	**detekce nástupné hrany**
	EDGE_L	**detekce sestupné hrany**
syntax	**EDGE_H**	**bit**
	EDGE_L	**bit**
parametr	**„bit"**	**symbolický název bitu**

Instrukce **EDGE_H** a **EDGE_L** testují příchod hrany u bitové proměnné. V případě výskytu příslušné hrany nastaví registr RLO na hodnotu log.1. Když není hrana bitové proměnné detekována, nastaví registr RLO na hodnotu log.0.
Instrukce nedetekují výskyt první hrany po startu PLC, nebo po stopu PLC a opětovném startu. V tomto případě se jedná jen o první nastavení hodnot proměnných, například podle aktuálního stavu vstupů.
Instrukce je možno použít i v rychlém modulu MODULE_FAST a také v časových blocích DFTM.

Příklady:
Mějme symbolicky definovat bity paměti A1, A2. Naplňte registr logických operací v závislosti na stavu těchto pamětí dle logického výrazu:

```
RLO = A1 + A2

řešení a):  LDR         A1
            LDR         A2
            LO
```

Je-li za instrukcí `LA`, `LO` nebo `LX` uveden operand (symbolicky označený bit paměti), má instrukce následující význam:

Instrukce `LA` s operandem provede logický součin zadaného bitu paměti s registrem logických operací a výsledek opět uloží do RLO. Instrukce `LO` provede obdobně logický součet instrukce `LX` nonekvivalenci. Podíváme-li se zpět na příklad 3, zjistíme,že jeho další možné řešení je následující:

```
řešení b):      LDR         A1
                LO          A2
```

Řešení 3b je co do funkce naprosto rovnocenné řešení 3a, délka programu je však v tomto případě kratší.

Pomocí výše uvedených instrukcí (`LDR`, `LA`, `LO`, `LX`, `CA`) lze tedy naprogramovat libovolnou logickou podmínku (logický výraz). Postup programování názorně ukazují následující příklady 4, 5, 6.

Příklad:
Naprogramujte logický výraz (pozn.: <> je NONEKVIVALENCE):

```
RLO = (ALFA <> BETA) + (GAMA <> DELTA)

      LDR         ALFA
      LX          BETA
      LDR         GAMA        ;ULOŽÍ RLO DO ZÁSOBNÍKU A NAČTE BIT GAMA
      LX          DELTA
      LO                      ;LOG.SOUČET RLO SE ZÁSOBNÍKEM
```

Příklad:
Naprogramujte logický výraz:

```
         ----     ----     ----                ----
RLO = [(A1.A2.A3) + (B1.B2)] .  (C2 + C3)

      LDR         -A1
      LA          A2
      LA          -A3
      LDR         -B1   ;ULOŽÍ RLO DO ZÁSOBNÍKU A NAČTE NEGACI BITU B1
      LA          B2
      LO                ;LOG.SOUČET RLO SE ZÁSOBNÍKEM
      LDR         C2    ;ULOŽÍ MEZIVÝSLEDEK DO ZÁSOBNÍKU A NAČTE BIT C2
      LO          -C3
      LA                ;LOG.SOUČIN RLO SE ZÁSOBNÍKEM
```

Příklad:
Naprogramujte logický výraz:

$$RLO = \overline{A1.A2.A3} + \overline{B1.B2.B3}$$

```
      LDR         A1
      LA          A2
      LA          A3
      CA
      LDR         B1
      LA          B2
      LA          B3
      CA
      LO
```

Příklad:
Když signál ALFA má nástupní hranu, nastavte do bitu BETA log.1, jinak log.0:

```
      EDGE_H      ALFA
      WR          BETA
```

3.7 Zápis bitů do paměti

instrukce	WR

```
funkce      zápis obsahu RLO do paměti

syntax 1    WR          bit

syntax 2    WR          bit1 [,bit2 ...]
            WR          [adr.] bit  složitější způsoby adresace
            WR          [adr.] bit  [,[adr2.]bit2 ...]
            WR          [$+xx.]bit  [,[$+yy.]bit2 ...]

parametry   „bit"       symbolický název bitů
```

Instrukce **WR** provádí zápis obsahu RLO do vyjmenovaných bitů jedné slabiky paměti. Obsah RLO a DR se po vykonání této instrukce nemění. Instrukce WR je ***koncová instrukce*** pro logické rovnice.
Instrukce umožňuje v parametrech instrukce použít více bitových operandů, které nemusí být umístěny v jednom bajtu. Operandy se oddělí čárkou.
Složitější způsoby adresace jsou popsány u instrukce LDR.

instrukce	FL

```
funkce      nastavování bitů v paměti

syntax 1    FL    0, bit
            FL    1, bit

syntax 2    FL    0, bit1 [,bit2 ...]
            FL    1, bit1 [,bit2 ...]
            FL    0, [adr.] bit          složitější způsoby adresace
            FL    1, [adr.]  bit [,[adr2.]bit2 ...]
            FL    0, [$+xx.] bit [,[$+yy.]bit2 ...]

1.parametr  „0, 1"      logická hodnota pro plnění bitů
2.parametr  „bit"       symbolický název bitů
```

Instrukce **FL** zajistí, nezávisle na obsahu RLO, plnění bitů jedné slabiky paměti nulou nebo jedničkou. Za příkazem FL, jako první operand, následuje hodnota 0 nebo 1. Jako druhý operand je nutné uvést seznam bitů, do kterých se bude uvedená hodnota plnit. Bity nemusí být umístěny v jednom bajtu. Obsah RLO a DR se po vykonání této instrukce nemění.
Instrukce umožňuje v parametrech instrukce použít více bitových operandů, které nemusí být umístěny v jednom bajtu. Operandy se oddělí čárkou.

Složitější způsoby adresace jsou popsány u instrukce LDR.

instrukce	FL1

```
funkce      podmíněné nastavování bitů v paměti

syntax 1    FL1   0,bit
            FL1   1,bit

syntax 2    FL1   0, bit1 [,bit2 ...]
            FL1   1, bit1 [,bit2 ...]
            FL1   0, [adr.] bit          složitější způsoby adresace
            FL1   1, [adr.]  bit [,[adr2.]bit2 ...]
            FL1   0, [$+xx.] bit [,[$+yy.]bit2 ...]

1.parametr  „0, 1"      logická hodnota pro plnění bitů
2.parametr  „bit"       symbolický název bitů
```

Instrukce **FL1** zajistí plnění bitů jedné slabiky paměti nulou nebo jedničkou jenom tehdy, když je v registru RLO hodnota **1**. Za příkazem FL1, jako první operand, následuje hodnota 0 nebo 1. Jako druhý operand je nutné uvést seznam bitů, do kterých se bude uvedená hodnota plnit. Bity nemusí být umístěny v jednom bajtu. Obsah RLO a DR se po vykonání této instrukce nemění. Instrukce FL1 je ***koncová instrukce*** pro logické rovnice. Složitější způsoby adresace jsou popsány u instrukce LDR.

Instrukce FL1 se dá nahradit pomocí dvou instrukcí a jednoho návěští:

```
      JL0   OBSKOK
      FL    1,BIT       ekvivalent: FL1   1,BIT
OBSKOK
```

instrukce	MOVR MOVR1

```
funkce      přímý přesun bitů v paměti

syntax      MOVR (MOVR1)      dst, src
            MOVR (MOVR1)      [adr.]dst, [adr.]src

1.parametr  „dst" symbolický název cílového bitu
2.parametr  „src" symbolický název zdrojového bitu
```

Instrukce **MOVR** zajistí přímý přesun bitu z jedné slabiky paměti (2.operand) do druhé slabiky paměti (1.operand). Obsah RLO a DR se po vykonání této instrukce nemění. Instrukce **MOVR1** provede přesun bitu podmíněně, jenom když je v registru RLO hodnota **1**. Instrukce MOVR1 je ***koncová instrukce*** pro logické rovnice. Instrukce umožňuje použít složitější způsoby adresace, které jsou popsány u instrukce LDR.

Příklad:
Symbolicky označené bity paměti PETR, IVAN, JANA naplňte v závislosti na pamětech PAVEL a EVA takto:

```
PETR = IVAN = PAVEL + EVA
JANA = PAVEL . EVA

JMENA1:     DFM         PAVEL,EVA, , , , , ,
JMENA2:     DFM         PETR,IVANA,JANA, , , , , ,
            ...
            LDR         PAVEL
            LO          EVA
            WR          PETR,IVAN
            LDR         PAVEL
            LA          EVA
            WR          JANA
```

Příklad:
Mějme definované bity paměti těmito symboly:

B1, B2, A2
Naplňte tyto paměti na hodnotu log.1:

```
      FL          1,A2,B1,B2
```

3.8 Větvení programu

instrukce	**JUM** **JL0** **JL1**

funkce	**JUM**	**nepodmíněný skok**
	JL0	**skok, je-li RLO = 0**
	JL1	**skok, je-li RLO = 1**
syntax	**JUM**	**adr**
	JL0	**adr**
	JL1	**adr**
parametr	**„adr"**	**adresa programu**

Všechny tyto instrukce, které umožňují větvení programu, nebo-li programový skok, musí mít jako operand uvedenu adresu instrukce, která má být vykonávána dál v případě splnění příslušné podmínky skoku. Není-li tato podmínka splněna, pokračuje procesor ve zpracování instrukce následující (skok se neprovede). Tato adresa se zadává symbolickou formou, přičemž příslušný symbol lze definovat pomocí dvojtečky v kterémkoli místě programu.
Instrukce **JUM** nevyžaduje splnění žádné podmínky, skok se tedy provede vždy. Instrukce **JL0** zajistí skok na zadanou adresu pouze v případě, že RLO = 0. Instrukce **JL1** zajistí skok na zadanou adresu pouze v případě, že RLO = 1. Instrukce JL0 a JL1 jsou ***koncové instrukce*** pro logické rovnice.

Příklad:
Zapište pomocí instrukcí jazyka TECHNOL následující logický algoritmus:

```
Je-li A1 <> A2, naplň 0 do B1 a B2.
Je-li A1 . A2 = 1, naplň 1 do B3.

            LDR         A1
            LX          A2
            JL1         NAV1
            FL          0,B1,B2
 NAV1:      LDR         A1
            LA          A2
            FL1         1,B3
```

3.9 Způsoby předefinování typu u datových proměnných

Jazyk TECHNOL v datových operacích přistupuje k operandům automaticky podle toho, jaká byla jejich definice. Například instrukce LOD načte hodnotu typu BYTE, WORD, DWORD nebo konstantu podle způsobu definice operandu:

```
DEFINICE:                       OPERACE:

BUNKA1:     DS 1          ..... LOD   BUNKA1      načte BYTE do DR registru

BUNKA2:     DS 2          ..... LOD   BUNKA2      načte WORD do DR registru

BUNKA3:     DS 4          ..... LOD   BUNKA3      načte DWORD do DR registru

EQUI        CON,124       ..... LOD   CON         načte konstantu 124 do DR
```

Některé instrukce, které pracují s DR registrem, mají možnost při svém vykonávání předefinovat typ datové proměnné. Následující popis se bude týkat instrukcí `LOD, STO, STO1, AD, SU, EQ, EQ1, LT, GT, LE, GE, RR, RL, TM, TIM, TEX0, TEX1, MULB a DIVB.`

Instrukce můžou mít nepovinný prefix před názvem proměnné (TYPE.), který může předefinovat nebo dodefinovat typ proměnné na :

- **CNST**
- **BYTE**
- **WORD**
- **HIGH**
- **DWRD**
- **QWRD**
- **REAL**

Prefix se připojí k operandu instrukce pomocí tečky.
Instrukce `LOD, AD, SU, EQ, EQ1, LT, GT, LE, GE, RR, RL, MULB, DIVB` mohou mít jako operand konstantu, která není definovaná pomocí instrukce **`EQUI`**. V tomto případě se uvede před hodnotou konstanty typ: **`"CNST."`** Použití prefixu CNST. způsobí, že instrukce budou pracovat s 32-bitovým DR registrem. Pokud přímo zadaná hodnota obsahuje desetinnou tečku, považuje se to za zadání čísla v reálném tvaru..

Příklad:
Zapište do buňky odměřování systému v ose X:

```
BUNKA:      DS    4

            CLI                     ;zákaz přerušení
            LOD   DWRD.B_POL        ;načte do rozšířeného DR registru B_POL
            AD    DWRD.B_INK        ;připočte dvě slova B_INK
            STI                     ;povolení přerušení
            STO   BUNKA             ;zapíše do BUNKA 32 bitů
```

Příklad:
Použití prefixů:

```
            LOD   CNST.123          ;přímé naplnění konstantou
            AD    CNST.50h          ;připočte 50 hexadecimálně
```

```
AD    BYTE.ALFA          ;připočte spodní byte ALFA
STO   WORD.BETA          ;zapíše do BETA jako WORD
```

Příklad:

```
LOD   CNST.1.8           ;přímé naplnění reálnou konstantou
AD    REAL.BUN_R         ;připočte reálnou buňku
```

Možnosti deklarace a předefinování typu u datových proměnných:

	DEKLARACE				***MOŽNOST PŘEDEFINOVÁNÍ TYPU***						
	BYTE	**WORD**	**CNST**	**DWRD**	**BYTE.**	**WORD.**	**HIGH.**	**CNST.**	**DWRD.**	**QWRD.**	**REAL.**
LOD	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STO	✓	✓	.	✓	✓	✓	✓	.	✓	✓	✓
STO1	✓	✓	.	✓	✓	✓	✓	.	✓	✓	✓
TM	✓	✓	.	.	✓	✓	✓	.	.	.	.
CU,CD	✓	✓	.	.	.	.	.	.	.	.	.
CUBCD	✓	✓	✓	.	.	.	.	.	.	.	.
AD,SU	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MULB	✓	✓	✓	✓	✓	✓	✓	✓	✓	.	✓
DIVB	✓	✓.	✓	✓	✓	✓	✓	✓	✓	.	✓
RR,RL	✓	.	✓	.	✓	.	✓	✓	.	.	.
EQ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EQ1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LT,GT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LE,GE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TIM	✓	✓	.	.	✓	✓	✓	.	.	.	.
TEX0,1	✓	✓	.	.	✓	✓	✓	.	.	.	.

3.10 Zápis a čtení do paměti z datového registru DR

instrukce	**LOD**

funkce **čtení paměti typu BYTE,WORD,DWORD,QWORD,REAL, konstanty do DR**

syntax
```
LOD   [-] val
LOD   [-] [TYPE.] val
LOD   [-] TYPE.(val+n)
LOD   [-] CNTS.immed
LOD       CNTS.ireal
               TYPE = BYTE. WORD. HIGH. DWRD. REAL. QWRD.
```

parametr **„val"** **symbolický název datové proměnné**

Instrukce **LOD** zajistí načtení obsahu paměti o délce BYTE, WORD, DWORD, QWORD, REAL nebo konstanty do datového registru DR (nebo DR_REAL). Adresa paměti se zadává formou operandu instrukce v symbolickém tvaru. Konstanta musí být definována pomocí instrukce **EQUI**, nebo pomocí prefixu "**CNST.**". Použití instrukce s prefixem "CNST." způsobí naplnění DR registru velkosti DWORD. Pokud přímo zadaná hodnota obsahuje desetinnou tečku, považuje se to za zadání čísla v reálném tvaru. Instrukce před načtením vynuluje obsah DR registru a po načtení znaménkově rozšíří DR registr až na 64 bitů.

Zadání znaménka - způsobí, že hodnota bude vzata do DR registru záporně. Ve skutečnosti se jedná o dvojkový doplněk z načtené hodnoty.

instrukce LOD	typ operace (výsledek DR64)	deklarace proměnné	popis
LOD Value	BYTE -> DR8	Value: DS 1	přístup podle deklarace proměnné
LOD Value	WORD -> DR16	Value: DS 2	přístup podle deklarace proměnné
LOD Value	DWORD -> DR32	Value: DS 4	přístup podle deklarace proměnné
LOD Value	Immed -> DR32	EQUI Value,Immed	přístup podle deklarace konstanty
LOD BYTE.Value	BYTE -> DR8		přetypování na BYTE
LOD WORD.Value	WORD -> DR16		přetypování na WORD
LOD DWRD.Value	DWORD -> DR32		přetypování na DWORD
LOD QWRD.Value	QWORD -> DR64		přetypování na QWORD
LOD Immed	Immed -> DR32		přímá hodnota celočíselná
LOD CNST.Immed	Immed -> DR32		přímá hodnota
LOD BYTE.Immed	Immed -> DR8		přímá hodnota + přetypování
LOD WORD.Immed	Immed -> DR16		přímá hodnota + přetypování
LOD DWRD.Immed	Immed -> DR32		přímá hodnota + přetypování
LOD REAL.Value	REAL -> DRr,DR64	Value: DS 8	reálná hodnota proměnné
LOD CNST.Ireal	Ireal -> DRr,DR64		přímá hodnota reálná

Instrukce pro práci s reálnými čísly musí mít povinně prefix „REAL" nebo „CNST" s číslem s desetinnou tečkou. Pro všechny instrukce s reálnými čísly platí, že po operaci zůstává výsledek jak v reálném registru DR_REAL, tak v celočíselném registru DR64. Proto je možno kdykoli pokračovat standardními celočíselnými operacemi. Nelze ale kombinovat celočíselné a reálné operace.

Popis s reálnými čísly bude popsán dále.

Popis:

`Value ..`	název datové proměnné
`Immed ...`	přímá hodnota celočíselná (bez desetinné tečky, může být i hexadecimálně)
`Ireal ...`	přímá hodnota reálná (číslo obsahuje desetinnou tečku)
`REAL...`	reálná hodnota proměnné (64 bitů)
`DR8 ...`	datový registr DR 8 bitů
`DR16 ...`	datový registr DR 16 bitů
`DR32...`	datový registr DR 32 bitů
`DR64...`	datový registr DR 64 bitů celočíselný
`DRr...`	datový registr DR 64 bitů reálný

instrukce	**STO**

`funkce` **zápis DR do paměti typu BYTE,WORD,DWORD,QWORD,REAL**

`syntax`

```
STO          val
STO          [TYPE.] val
STO          TYPE.(val+n)
                   TYPE = BYTE. WORD. HIGH. DWRD. REAL. QWRD.
```

`parametr` **„val“** **symbolický název datové proměnné**

Instrukce **STO** umožňuje zapsat obsah datového registru DR do zadané paměti o délce BYTE, WORD, DWORD, QWORD nebo REAL. Adresa paměti se opět zadává formou operandu instrukce v symbolickém tvaru.

Možnosti operandu instrukce a přetypování je popsáno v kapitole "Způsoby předefinování typu u datových proměnných".

instrukce STO, STO1	typ operace	deklarace proměnné	popis
`STO Value`	`DR8 -> BYTE`	`Value: DS 1`	přístup podle deklarace proměnné
`STO Value`	`DR16 -> WORD`	`Value: DS 2`	přístup podle deklarace proměnné
`STO Value`	`DR32 -> DWORD`	`Value: DS 4`	přístup podle deklarace proměnné
`STO BYTE.Value`	`DR8 -> BYTE`		přetypování na BYTE
`STO WORD.Value`	`DR16 -> WORD`		přetypování na WORD
`STO DWRD.Value`	`DR32 -> DWORD`		přetypování na DWORD
`STO QWRD.Value`	`DR64 -> QWORD`		přetypování na QWORD
`STO REAL.Value`	`DRr -> REAL`	`Value: DS 8`	reálná hodnota proměnné

instrukce	**STO1**

funkce **podmíněný zápis DR do paměti typu BYTE,WORD,DWORD,QWORD,REAL**

```
syntax      STO1        val
            STO1        [TYPE.] val
            STO1        TYPE.(val+n)
                              TYPE = BYTE. WORD. HIGH. DWRD. REAL. QWRD.

parametr    „val"       symbolický název datové proměnné
```

Instrukce **STO1** je podobná instrukci **STO**, ale zápis DR do paměti se provede jenom tehdy, když je v registru RLO hodnota 1. Instrukce STO1 je zavedena pro kompatibilitu s automatem NS915. Důležitý rozdíl vzhledem k instrukci STO je, že instrukce **STO1** je ***koncová instrukce*** vzhledem k logickým operacím.

Je možno použít i obdobnou instrukci **STO0**, která provede podmíněný zápis DR do paměti tehdy, když je v registru RLO hodnota 0.

Možnosti operandu instrukce a přetypování je popsáno v kapitole "Způsoby předefinování typu u datových proměnných".

Instrukce STO1 se dá nahradit pomocí dvou instrukcí a jednoho návěští:

```
            JL0   OBSKOK
            STO   BUNKA       ekvivalent:  STO1  BUNKA
OBSKOK:
```

instrukce	**MOVE** **MOVE1**

funkce **přímý přesun dat v paměti typu BYTE,WORD,DWORD,QWORD,REAL**

```
syntax      MOVE (MOVE1)      dst, src
            MOVE (MOVE1)      [TYPE.]dst, [TYPE.]src
            MOVE (MOVE1)      TYPE.(dst+n), TYPE.(src+n)
                              TYPE = BYTE. WORD. HIGH. DWRD. REAL. QWRD.

1.parametr  „dst"       symbolický název cílové datové proměnné
2.parametr  „src"       symbolický název zdrojové datové proměnné
```

Instrukce **MOVE** provede přímý přesun ze zdrojové buňky z paměti o délce podle typu (2.operand) do druhé cílové buňky paměti (1.operand). Zdrojová a cílová buňka musí mít stejný typ dat. Obsah RLO a DR se po vykonání této instrukce nemění.

Možnosti operandu instrukce a přetypování je popsáno v kapitole "Způsoby předefinování typu u datových proměnných".

Instrukce **MOVE1** je podobná instrukci **MOVE**, ale přesun se provede jenom tehdy, když je v registru RLO hodnota 1. Důležitý rozdíl vzhledem k instrukci MOVE je, že instrukce **MOVE1** je ***koncová instrukce*** vzhledem k logickým operacím.

Příklady:
Mějme symbolicky definované slabiky paměti RAM těmito symboly:

ALFA, BETA, GAMA

Obsah slabiky ALFA zapište do slabiky BETA, konstantu 123H do GAMA.

```
EQUI        K123, 123H

LOD         ALFA
STO         BETA
LOD         K123
STO         GAMA
```

Způsob zápisu je stejný i pokud se jedná o délky typu WORD nebo DWORD.

Příklad:
Mějme symbolicky definované slabiky paměti RAM těmito symboly: ALFA typu WORD a BETA typu BYTE. Záporný obsah slabiky BETA zapište do spodního byte proměnné ALFA.

```
LOD         -BETA
STO         BYTE.ALFA
```

Příklad:
Přepište hodnotu z buňky NASTAVENI do buňky BUNKA, když výraz ALFA*BETA je roven jedné:

```
LDR         ALFA
LA          BETA
LOD         NASTAVENI
STO1        BUNKA
```

Příklad:
Zapíše reálnou hodnotu -10.2 do buňky rBun.

```
LOD         CNST.-10.2
STO         REAL.rBun
```

3.11 Realizace časově závislých funkcí

instrukce	DFTM01 DFTM1 DFTM10 DFTM100

funkce	**DFTM01**	**úsek programu aktivován po 0,1 s**
	DFTM1	**úsek programu aktivován po 1 s**
	DFTM10	**úsek programu aktivován po 10 s**
	DFTM100	**úsek programu aktivován po 100 s**
syntax	**DFTM01**	**end**
	DFTM1	**end**
	DFTM10	**end**
	DFTM100	**end**
parametr	**„end"**	**adresa konce úseku programu DFTM**

Aby bylo dosaženo co nejoptimálnějšího využití strojového času procesoru a pro definici nastavované doby časovačů **TM** a **TIM,** jsou časově závislé funkce realizovány ve zvláštních blocích. Takovým programovým blokem je úsek programu, který je na začátku ohraničen instrukcí **DFTM01**, **DFTM1**, **DFTM10** či **DFTM100** a na konci návěštím "end".

Zvláštností těchto programových bloků je to, že jsou aktivovány v delších časových intervalech než ostatní části řídicího programu PLC, které jsou aktivovány po 20ms. Programový blok, definovaný instrukcí DFTM01, je aktivován ("procházen procesorem") po intervalech 0,1 sec. Programový blok, definovaný instrukcí DFTM1, je aktivován po intervalech 1 sec. Programový blok, definovaný instrukcí DFTM10, je aktivován po intervalech 10 sec a programový blok definovaný instrukcí DFTM100 je aktivován po 100 sec.
Instrukce DFTM mohou být použity ve všech souborech i vícekrát.

instrukce	TM

funkce	**časovač závislý na DR, RLO a bloku DFTM**	
syntax	**TM**	**count**
	TM	**[TYPE.] count**
	TM	**TYPE.(count+n)**
	TM	**-**
		TYPE = BYTE. WORD.
parametr	**„count"**	**název čítače časovače**

Instrukce **TM** pracuje jak s registrem DR, tak s registrem RLO. Jako operand této instrukce je nutno zadat symbolickou adresu slabiky paměti, která bude sloužit pro čítání příslušného času (typ BYTE nebo WORD).
Parametr je nepovinný. V tomto případě si překladač definuje "automatickou" proměnnou pro tento čítač. Čítání času v instrukci TM je závislé od toho, v jakém časovém bloku DFTM se instrukce TM nachází. Když se například

instrukce TM nachází v úseku programu definovaném instrukcí DFTM10 (je aktivován po 10 sec.), nastavená doba je v desítkách vteřin.

Instrukce TM pracuje takto:

1) Je-li RLO = 0, obsah zadaného čítače se nuluje a instrukce tím končí.

2) Je-li RLO = 1, provede se porovnání obsahu zadaného čítače s obsahem datového registru DR.
   a) Je-li /count/ >= DR, nastaví se RLO do stavu 1 a instrukce tím končí.
   b) Je-li /count/ < DR, nastaví se RLO do stavu 0 a provede se zvětšení zadaného čítače o jedničku (V bloku DFTM01 to znamená čas 0,1 sec. a v bloku DFTM1 čas 1 sec.).

Kromě vlastních instrukcí TM musí být v bloku časovačů ještě instrukce pro nastavení registru DR a RLO počátečními podmínkami a samozřejmě také instrukce pro nastavení jedné nebo více paměťových buněk dle výsledku této instrukce (RLO).

Příklad:
Realizujte tuto časově závislou funkci:
Je-li bit ALFA po dobu delší než 0,4 sec. ve stavu 1, nastavte do stavu 1 též bit GAMA. K čítání času použijte slabiku paměti CITACA.

```
             EQUI        DOBA,4

             DFTM01      NAV30
             ...
             LDR         ALFA
             LOD         DOBA
             TM          CITACA
             FL1         1,GAMA
             ...
NAV30:
```

Příklad:
Je-li logický součin A1 a A2 po dobu delší než 5 sec. roven nule, vynulujte bit GAMA a vynulujte buňku BYTE. K čítání času se použije automatická definice čítače.

```
             DFTM1       NAV50
             ...
             LDR         A1
             LA          A2
             CA
             LOD         CNST.5
             TM          -
             FL1         0,GAMA
             LOD         CNTS.0
             STO1        BYTE
             ...
NAV50:
```

instrukce	CU CD CUBCD

funkce	**CU**	**čítač nahoru závislý na DR, RLO a bloku DFTM**
	CD	**čítač dolů závislý na DR, RLO a bloku DFTM**
	CUBCD	**BCD čítač nahoru závislý na DR, RLO a bloku DFTM**
syntax	**CU**	**count**
	CD	**count**
	CUBCD	**count**
parametr	**„count"**	**název čítače**

Čítačové instrukce jsou určeny k realizaci čítacích funkcí. Instrukcí čítač nahoru **CU** se provádí v případě, že je splněna podmínka pro čítání (nulovací vstup čítače je sepnut), inkrementování buňky, určené adresovou částí instrukce CU. Ke změně stavu čítače dojde pouze při změně stavu bitu, určeného jako vstup čítače ze stavu log.0 do stavu log.1. Zároveň je porovnáván stav datového registru DR, ve kterém je uložena předvolba čítače se stavem adresovaného čítače. V případě rovnosti čítače a DR je nastaven RLO do log. 1, v opačném případě do log.0. Po skončení instrukce CU je obsah adresovaného čítače uložen v datovém registru DR.

Instrukce čítače nahoru CU pracuje s daty v binárním kódu v rozsahu BYTE nebo WORD. Po dosažení nastavené hodnoty se čítače nenulují a není žádné přednastavení čítačů.

Instrukce čítač dolů **CD** pracuje podobně s tím rozdílem, že provádí dekrementaci čítače. Instrukce **CUBCD** pracuje podobně jako instrukce CU, ale v BCD kódu.

Příklad:
Změny vstupního signálu ALFA z nuly do jedničky jsou čítány, pokud blokovací vstup BETA je v jedničce.

```
            LDR         ALFA        ;VSTUP
            LA          BETA        ;BLOKOVÁNÍ
            LOD         GAMA        ;NAČTENÍ PŘEDVOLBY
            CU          DELTA       ;ČÍTAČ (BYTE, WORD)
            JL0         NAV1
            ...
NAV1:
```

Příklad:
Vynulování čítače po dočítání na zadanou předvolbu GAMA

```
            LDR         ALFA        ;VSTUP
            LOD         GAMA        ;NAČTENÍ PŘEDVOLBY
            CU          DELTA       ;ČÍTAČ (BYTE, WORD)
            LOD         CNTS.0
            STO1        DELTA       ;VYNULOVÁNÍ
```

3.12 Aritmetické a logické instrukce s operandy a DR registrem

Výše uvedené instrukce jsou určeny k provádění aritmetických nebo logických operací, přičemž většinou platí pravidlo, že operace se provádí mezi DR a obsahem paměti nebo konstantou. Adresa paměti se uvádí jako parametr instrukce, vyjma instrukcí `INR, DCR, BIN, BCD, ABS, INV, RR a RL` které jsou bez operandu.

instrukce	**AD** **SU**

funkce	**AD**	**sčítání BYTE,WORD,DWORD,QWORD,REAL nebo konstanty do DR**
	SU	**odčítání BYTE,WORD,DWORD,QWORD,REAL nebo konstanty od DR**
syntax	**AD (SU)**	**val**
	AD (SU)	**[TYPE.] val**
	AD (SU)	**TYPE.(val+a)**
	AD (SU)	**CNTS.immed**
	AD (SU)	**CNTS.ireal**
		TYPE = BYTE. WORD. HIGH. DWRD. REAL. QWRD.
parametr	**„val"**	**název datové proměnné**

Instrukce **AD** sečte obsah DR registru (nebo DR_REAL) s obsahem paměti, na kterou ukazuje operand nebo konstantou. Pokud přímo zadaná hodnota obsahuje desetinnou tečku, považuje se to za zadání čísla v reálném tvaru. Výsledek operace zůstane v DR registru (nebo DR_REAL). Instrukce může pracovat s operandy typu BYTE, WORD, DWORD,QWORD nebo REAL .

Instrukce **SU** odečte od obsahu DR registru (nebo DR_REAL) obsah paměti, na kterou ukazuje operand nebo konstantu. Výsledek operace zůstane v DR registru (nebo DR_REAL).

Instrukce pracují s registrem DR se stejnou šířkou slova, jaká je určena operandem instrukce. Využívají tak z DR registru 8, 16 nebo 32 bitů. Nevyužitá část DR registru zůstává nezměněná. U reálných operací zůstává výsledek jak v reálném registru DR_REAL, tak v celočíselném registru DR64.

instrukce AD (SU)	**typ operace**	**deklarace proměnné**	**popis**
`AD Value`	`DR8 + BYTE`	`Value: DS 1`	přístup podle deklarace proměnné
`AD Value`	`DR16 + WORD`	`Value: DS 2`	přístup podle deklarace proměnné
`AD Value`	`DR32 + DWORD`	`Value: DS 4`	přístup podle deklarace proměnné
`AD Value`	`DR32 + Immed`	`EQUI Value,Immed`	přístup podle deklarace konstanty
`AD BYTE.Value`	`DR8 + BYTE`		přetypování na BYTE
`AD WORD.Value`	`DR16 + WORD`		přetypování na WORD
`AD DWRD.Value`	`DR32 + DWORD`		přetypování na DWORD
`AD QWRD.Value`	`DR64 + QWORD`		přetypování na QWORD
`AD Immed`	`DR32 + Immed`		přímá hodnota
`AD CNST.Immed`	`DR32 + Immed`		přímá hodnota
`AD BYTE.Immed`	`DR8 + Immed`		přímá hodnota + přetypování
`AD WORD.Immed`	`DR16 + Immed`		přímá hodnota + přetypování
`AD DWRD.Immed`	`DR32 + Immed`		přímá hodnota + přetypování
`AD REAL.Value`	`DRr + REAL`	`Value: DS 8`	reálná hodnota proměnné
`AD CNST.Ireal`	`DRr + Ireal`		přímá hodnota reálná

instrukce	**MULB** **DIVB**

funkce **MULB** **násobení DR registru a operandu**
DIVB **celočíselné dělení DR registru s operandem**

syntax **MULB(DIVB) val**
MULB(DIVB) [TYPE.] val
MULB(DIVB) CNST. immed
MULB(DIVB) CNST. ireal
TYPE = BYTE. CNTS. HIGH. WORD. DWRD. REAL.

parametr **„val"** **název datové proměnné**

Instrukce **MULB** vynásobí registr DR (nebo DR_REAL) s obsahem paměti, na kterou ukazuje operand. Jedná se o znaménkové násobení. Výsledek operace zůstane v DR registru (nebo DR_REAL). Užitečná šířka slova v DR registru je po vynásobení dvojnásobná, ale instrukce vždy znaménkově rozšíří DR registr až na 64 bitů (je to pro případ, že po násobení bezprostředně následuje dělení).
Pokud přímo zadaná hodnota obsahuje desetinnou tečku, považuje se to za zadání čísla v reálném tvaru.
U reálných operací zůstává výsledek jak v reálném registru DR_REAL, tak v celočíselném registru DR64.

instrukce MULB	typ operace (výsledek DR64)	deklarace proměnné	popis (přístup)
MULB Value	DR8 * BYTE	Value: DS 1	podle deklarace proměnné
MULB Value	DR16 * WORD	Value: DS 2	podle deklarace proměnné
MULB Value	DR32 * DWORD	Value: DS 4	podle deklarace proměnné
MULB Value	DR32 * Immed	EQUI Value,Immed	podle deklarace konstanty
MULB BYTE.Value	DR8 * BYTE		přetypování
MULB WORD.Value	DR16 * WORD		přetypování
MULB DWRD.Value	DR32 * DWORD		přetypování
MULB Immed	DR32 * Immed		přímá hodnota
MULB CNST.Immed	DR32 * Immed		přímá hodnota
MULB BYTE.Immed	DR8 * Immed		přímá hodnota + přetypování
MULB WORD.Immed	DR16 * Immed		přímá hodnota + přetypování
MULB DWRD.Immed	DR32 * Immed		přímá hodnota + přetypování
MULB REAL.Value	DRr * REAL	Value: DS 8	reálná hodnota proměnné
MULB CNST.Ireal	DRr * Ireal		přímá hodnota reálná

Instrukce **DIVB** vydělí registr DR (nebo DR_REAL) s obsahem paměti, na kterou ukazuje operand. Výsledek operace zůstane v DR registru (nebo DR_REAL). Dělení je celočíselné s ohledem na znaménko.
Pokud je požadováno dělení hodnotou BYTE nebo WORD, instrukce dělení vždy před samotnou operací znaménkově rozšíří DR registr na 64 bitů a operand na 32 bitů. Pokud je požadováno dělení hodnotou DWORD, předpokládá se, že rozšířený DR registr 64 bitů je před operací platný (to znamená, že předcházelo násobení, dělení nebo instrukce LOD). Instrukce vždy znaménkově rozšíří výsledek v DR registru až na 64 bitů.
Pokud přímo zadaná hodnota obsahuje desetinnou tečku, považuje se to za zadání čísla v reálném tvaru.
U reálných operací zůstává výsledek jak v reálném registru DR_REAL, tak v celočíselném registru DR64.

instrukce DIVB	typ operace (výsledek DR64)	deklarace proměnné	popis (přístup)
DIVB Value	DR16 / BYTE	Value: DS 1	podle deklarace proměnné
DIVB Value	DR32 / WORD	Value: DS 2	podle deklarace proměnné
DIVB Value	DR64 / DWORD	Value: DS 4	podle deklarace proměnné
DIVB Value	DR64 / Immed	EQUI Value,Immed	podle deklarace konstanty
DIVB BYTE.Value	DR16 / BYTE		přetypování
DIVB WORD.Value	DR32 / WORD		přetypování
DIVB DWRD.Value	DR64 / DWORD		přetypování
DIVB Immed	DR64 / Immed		přímá hodnota
DIVB CNST.Immed	DR64 / Immed		přímá hodnota
DIVB BYTE.Immed	DR16 / Immed		přímá hodnota + přetypování
DIVB WORD.Immed	DR32 / Immed		přímá hodnota + přetypování
DIVB DWRD.Immed	DR64 / Immed		přímá hodnota + přetypování
DIVB REAL.Value	DRr / REAL	Value: DS 8	reálná hodnota proměnné
DIVB CNST.Ireal	DRr / Ireal		přímá hodnota reálná

Příklad:
Hodnotu z buňky ALFA (BYTE) zmenšenou o 23h vynásobte hodnotou z buňky BETA (BYTE) a výsledek zapište do buňky GAMA (WORD).

```
LOD         ALFA            ;načte ALFA do DR
SU          CNST.23H        ;odečte 23h
MULB        BETA            ;vynásobení s BETA (výsledek 16 bitů)
STO         GAMA            ;zapíše do GAMA
```

Příklad:
Podíl 32 bitové buňky DELENEC (DWORD) a 16 bitové buňky DELITEL (WORD) zapište do buňky PODIL (WORD).

```
LOD         DELENEC         ;načte dělenec 32 bitů do DR
DIVB        DELITEL         ;dělení DR32 s dělitelem 16 bitů
STO         PODIL           ;výsledek v PODIL 16 bitů
```

Příklad:
Vynásobte DR registr (32 bitů) zlomkem, pro CITATEL (DWORD) a DELITEL (DWORD) zapište do buňky VYSLEDEK (DWORD).

```
MULB        CITATEL     ;násobení DR32*CITATEL, výsledek 64 bitů
DIVB        DELITEL     ;dělení DR64 s dělitelem 32 bitů
STO         VYSLEDEK    ;zápis výsledku 32 bitů
```

Příklad:
Reálné operace:

```
LOD         REAL.rBun   ;načte reálnou proměnnou rBun
MULB        REAL.rBun2  ;DR_REAL <- rBun * rBun2
DIVB        CNST.4.8    ;DR_REAL <- rBun * rBun2 / 4.8
STO         REAL.rBun3
```

instrukce	**ORB** **ANDB** **XORB**

funkce	**ORB**	**logický OR po bitech mezi DR registrem a pamětí,konst.**
	ANDB	**logický AND po bitech mezi DR registrem a pamětí, konst.**
	XORB	**logický XOR po bitech mezi DR registrem a pamětí, konst.**

syntax	**ORB (ANDB,XORB)**	**val**
	ORB (ANDB,XORB)	**[TYPE.] val**
	ORB (ANDB,XORB)	**TYPE. (val+n)**
	ORB (ANDB,XORB)	**CNTS. immed**
		TYPE = BYTE. WORD. HIGH. DWRD.

parametr	**„val"**	**název datové proměnné**

Instrukce **ORB,ANDB** a **XORB** jsou určeny k provádění logických operací, přičemž platí pravidlo, že operace se provádí mezi DR a obsahem paměti nebo konstantou. Adresa paměti se uvádí jako parametr instrukce.

Instrukce provedou logický OR, AND nebo XOR po jednotlivých bitech mezi DR registrem a pamětí nebo konstantou.

Přístupy a způsoby přetypování jsou stejné jako u instrukcí AD a SU.

3.13 Bezoperandové instrukce pro práci s DR registrem

Některé bezoperandové instrukce pro operace s datovým DR registrem mohou pracovat s rozšířeným 32 nebo 64 bitovým DR registrem (DWORD, QWORD) nebo reálným registrem DR_REAL. Jedná se o instrukce **INR, DCR, INV, ABS, RR, RL, CONDR, CONRD**. V tomto případě je nutné modifikovat bezoperandové instrukce pomocí parametru **DWRD** nebo **QWRD**.

instrukce	**INR** **DCR** **INRBCD**

funkce	**INR**	**inkrement DR registru**
	DCR	**dekrement DR registru**
	INRBCD	**inkrement DR registru v BCD tvaru**
syntax	**INR (DCR)**	
	INR (DCR)	**[DWRD]**
	INR (DCR)	**[QWRD]**
	INR (DCR)	**[REAL]**
	INRBCD	

Instrukce **INR** bez operandu zvětší registr DR o jedničku. Při překročení rozsahu začíná od nuly.
Instrukce **DCR** bez operandu zmenší registr DR o jedničku.
Instrukce **INRBCD** zvětší registr DR o jedničku v BCD kódu. Pracuje v rozsahu 0.. 9999.

Instrukce INR a DCR s parametrem „DWRD" zvětšují nebo zmenšují rozšířený 32 bitový DR registr.
Instrukce INR a DCR s parametrem „QWRD" zvětšují nebo zmenšují rozšířený 64 bitový DR registr.

Instrukce INR a DCR s parametrem „REAL" zvětšují nebo zmenšují reálný registr DR_REAL. U reálných operací zůstává výsledek jak v reálném registru DR_REAL, tak v celočíselném registru DR64.

instrukce	**BIN** **BCD**

funkce	**BIN**	**převod BCD → BIN kódu**
	BCD	**převod BIN → BCD kódu**
syntax	**BIN**	
	BCD	
	BIN	**[DWRD]**
	BCD	**[DWRD]**

Instrukce **BIN** bez operandu převede číslo ve tvaru BCD, uložené v registru DR (maximálně 16 bitů) na binární číslo. Výsledek zůstane v DR registru.
Instrukce **BIN** s parametrem **DWRD** převede číslo v rozšířeném 32 bitovém DR registru. Záporné BCD číslo před převodem má nastavenou na jedničku bit s váhou 31.

Instrukce **BCD** bez operandu převede číslo v binárním tvaru, uložené v registru DR na BCD číslo. Výsledek zůstane v DR registru. Hodnota v DR registru před převodem nesmí být větší než 9999d = 270Fh.
Instrukce **BCD** s parametrem **DWRD** převede číslo v rozšířeném 32 bitovém DR registru. Hodnota v DR registru před převodem nesmí být větší než 99999999d = 5F5E0FFh.

instrukce	RL RR

funkce	RL	logický posuv DR registru vlevo
	RR	logický posuv registru vpravo
syntax	RL (RR)	n
	RL (RR)	[TYPE.] n
	RL (RR)	[TYPE.] n [,DWRD]
	RL (RR)	[TYPE.] n [,QWRD]
		TYPE = BYTE. HIGH. CNST.
parametr	„n“	počet rotací

Instrukce **RL** provede logický posuv DR vlevo o "n" bitů. Operand je konstanta nebo hodnota v buňce (maximálně o velikosti 8 bitů) udávající počet rotací.
Instrukce **RR** provede logický posuv DR vpravo o "n" bitů. Operand je konstanta nebo hodnota v buňce (maximálně o velikosti 8 bitů) udávající počet rotací.
Instrukce mohou mít druhý parametr „DWRD“ nebo „QWRD“, který udává, že se provede posuv rozšířeného 32 nebo 64 bitového DR registru.

instrukce	INV ABS

funkce	INV	negace DR registru (dvojkový doplněk)
	ABS	absolutní hodnota DR registru
syntax	INV (ABS)	
	INV (ABS)	[DWRD]
	INV (ABS)	[QWRD]
	INV (ABS)	[REAL]

Instrukce **INV** provede negaci DR registru, to je dvojkový doplněk.
Instrukce **ABS** provede absolutní hodnotu DR registru.
Instrukce mohou mít nepovinný parametr „DWRD“ nebo „QWRD“, který modifikuje instrukce tak, že negace nebo absolutní hodnota se provede nad rozšířeným 32 nebo 64 bitovým DR registrem.

Instrukce mohou mít nepovinný parametr „REAL“, který modifikuje instrukce tak, že negace nebo absolutní hodnota se provede v reálném registru DR_REAL. U reálných operací zůstává výsledek jak v reálném registru DR_REAL, tak v celočíselném registru DR64.

Příklad:
Různé operace:

```
LOD         ALFA             ;načtení ALFA (typ podle deklarace)
INR                          ;inkrementace DR (16 bitů)
ABS                          ;absolutní hodnota DR (16 bitů)
RL          POSUN            ;posun DR doleva o hodnotu v POSUN
STO         VYSLEDEK1        ;zápis do VYSLEDEK1
INV                          ;dvojkový doplněk DR (16 bitů)
RR          CNST.3           ;posun DR doprava o 3 bity
STO         VYSLEDEK2        ;zápis do VYSLEDEK2

LOD         DWRD.BETA        ;načtení 32 bitů z BETA do DR
ABS         DWRD             ;absolutní hodnota DR 32 bitů
INR         DWRD             ;inkrementace DR 32 bitů
RR          CNST.18,DWRD     ;posun o 18 bitů vpravo registru DR32
STO         VYSLEDEK3        ;zápis do VYSLEDEK3 32 bitů
```

3.14 Logické instrukce s operandy a DR registrem

Skupina logických instrukcí provede porovnání DR registru (nebo DR_REAL) s obsahem paměti nebo konstantou a na základě výsledku porovnání se nastaví bitový RLO registr.

instrukce	**EQ** **EQ1** **LT** **GT** **LE** **GE**

```
funkce      EQ    porovnávání DR registru s operandem
            EQ1   podmíněné porovnávání DR s operandem, když RLO = 1
            LT    DR je menší než operand
            GT    DR je větší než operand
            LE    DR je menší nebo rovno než operand
            GE    DR je větší nebo rovno než operand

syntax      EQ (EQ1,LT,GT,LE,GE)    val
            EQ (EQ1,LT,GT,LE,GE)    [TYPE.] val
            EQ (EQ1,LT,GT,LE,GE)    TYPE. (val+n)
            EQ (EQ1,LT,GT,LE,GE)    CNTS. immed
            EQ (EQ1,LT,GT,LE,GE)    CNTS. ireal
                                TYPE = BYTE. HIGH. WORD. DWRD. QWRD. REAL.

parametr    „val" název datové proměnné
```

Instrukce **EQ** je logická a provádí porovnání DR registru (nebo DR_REAL) s obsahem paměti, na kterou ukazuje operand nebo konstantou. Je-li DR shodný s obsahem paměti nebo s konstantou, je nastaven RLO registr do jedničky, v opačném případě je RLO vynulován. Operandy mohou být typu BYTE, WORD, DWORD, QWORD, REAL nebo konstanta .

Instrukce **LT** resp. **LE** jsou logické a provádí porovnání DR registru (nebo DR_REAL) s obsahem paměti, na kterou ukazuje operand nebo konstantou. Je-li DR menší resp. menší nebo roven než obsah paměti nebo konstanty, je nastaven RLO registr do jedničky, v opačném případě je RLO vynulován.

Instrukce **GT** resp. **GE** jsou logické a provádí porovnání DR registru (nebo DR_REAL) s obsahem paměti, na kterou ukazuje operand nebo konstantou. Je-li DR větší resp. větší nebo roven než obsah paměti nebo konstanty, je nastaven RLO registr do jedničky, v opačném případě je RLO vynulován.
Instrukce **EQ1** je zavedena pro přiblížení se ke kompatibilitě s automatem NS915. Porovnání se provede jenom tehdy, když je v registru RLO hodnota **1**, jinak zůstane v RLO hodnota 0. Instrukci `EQ1` možno použít na porovnání větších paměťových oblastí, protože instrukce EQ1 je možné zřetězit. Instrukce `EQ1` se dá nahradit pomocí dvou instrukcí a jednoho návěští:

```
            JL0   OBSKOK
            EQ    BUNKA       ekvivalent:       EQ1   BUNKA
OBSKOK:
```

Předefinování typu je popsáno v kapitole "Způsoby předefinování typu u datových proměnných".

instrukce EQ (LT,GT,LE,GE)	typ operace	deklarace proměnné	popis
EQ Value	DR8 ~ BYTE	Value: DS 1	přístup podle deklarace proměnné
EQ Value	DR16 ~ WORD	Value: DS 2	přístup podle deklarace proměnné
EQ Value	DR32 ~ DWORD	Value: DS 4	přístup podle deklarace proměnné
EQ Value	DR32 ~ Immed	EQUI Value,Immed	přístup podle deklarace konstanty
EQ BYTE.Value	DR8 ~ BYTE		přetypování na BYTE
EQ WORD.Value	DR16 ~ WORD		přetypování na WORD
EQ DWRD.Value	DR32 ~ DWORD		přetypování na DWORD
EQ QWRD.Value	DR64 ~ QWORD		přetypování na QWORD
EQ Immed	DR32 ~ Immed		přímá hodnota
EQ CNST.Immed	DR32 ~ Immed		přímá hodnota
EQ BYTE.Immed	DR8 ~ Immed		přímá hodnota + přetypování
EQ WORD.Immed	DR16 ~ Immed		přímá hodnota + přetypování
EQ DWRD.Immed	DR32 ~ Immed		přímá hodnota + přetypování
EQ REAL.Value	DRr ~ REAL	Value: DS 8	reálná hodnota proměnné
EQ CNST.Ireal	DRr ~ Ireal		přímá hodnota reálná

Příklad:
Skok na návěští MENSI, když buňka BUNKA1 je menší něž buňka BUNKA2

```
LOD         BUNKA1          ;načte BUNKA1 do DR
LT          BUNKA2          ;když je DR < BUNKA2 tak RLO=1, jinak 0
JL1         MENSI           ;skok na menší, když je RLO=1
```

Příklad:
Porovnejte hodnotu z buňky NASTAVENI s buňkou BUNKA, když výraz ALFA*NOT(BETA) je roven 1. Výsledek zapište do GAMA:

```
LDR         ALFA            ;načte bit ALFA do RLO
LA          -BETA           ;logický součin RLO s negací BETA
LOD         NASTAVENI       ;načte buňku NASTAVENI do DR
EQ1         BUNKA           ;podmíněné porovnání když RLO=1
WR          GAMA            ;zápis RLO do GAMA
```

Příklad:
Porovnejte mezi sebou oblasti paměti PAM1 a PAM2 o velikosti 8 BYTE.

```
LOD         DWRD.PAM1       ;načte 4 BYTE z PAM1 do DR 32 bitů
EQ          DWRD.PAM2       ;porovnání DR 32 bitů z 4 BYTE PAM2
LOD         DWRD.(PAM1+4)   ;načte další 4 BYTE s PAM1 do DR 32 bitů
EQ1         DWRD.(PAM2+4)   ;další porovnání se provede jen když
                            ;při prvním byla rovnost RLO=1
JL1         ROVNO           ;odskok při rovnosti
```

Příklad:
Nastavte bit AKCE do hodnoty 1, když buňka MATTL je rovna kódu ´W´

```
LOD         MATTL           ;načte buňku MATTL do DR
EQ          CNST.'W'        ;porovnání DR s konst. a nastavení RLO
FL1         1,AKCE          ;když RLO=1 nastaví bit AKCE na 1
```

3.15 Konverze a přesuny registrů a paměti

instrukce	CONDR CONRD

funkce	**CONDR**	**konverze DR → RLO**
	CONRD	**konverze RLO → DR**
syntax	**CONDR (CONRD)**	
	CONDR	**0,1,..,31**
	CONDR (CONRD)	**[DWRD]**
parametr	**„0,1,..,31"**	**číslo bitu pro přesun DR → RLO**

Instrukce **CONDR** provede konverzi registru DR do bitového registru RLO. Pokud je registr DR nulový, naplní se RLO registr na hodnotu 0. Pokud je registr DR nenulový, naplní se RLO registr na hodnotu 1. Instrukce CONDR neovlivní obsah DR registru.
Pokud instrukce CONDR obsahuje operand, kterým je číslo 0 - 31, instrukce přesune z datového registru DR do bitového registru RLO jen odpovídající bit.

Instrukce **CONRD** provede konverzi bitového registru RLO do datového DR registru. Pokud je registr RLO nulový, naplní se DR registr na hodnotu 0h. Pokud je registr RLO roven hodnotě 1, naplní se DR registr na hodnotu FFFFh. Instrukce CONRD neovlivní obsah RLO registru. Instrukce CONRD je ***koncová instrukce*** pro logické výrazy.
Při použití parametru **DWRD** se provede konverze s rozšířeným 32 bitovým DR registrem.

instrukce	MV

funkce	**MV**	**přesun paměti**
syntax	**MV**	**src, dest, num**
1.parametr	**„src"**	**adresa zdroje pro přesun dat**
2.parametr	**„dest"**	**adresa cíle, kam přesunout**
3.parametr	**„num"**	**počet přesouvaných bajtů**

Instrukce **MV** slouží pro přesun oblasti paměti. Parametr "src" je adresa zdroje - odkud se bude přesouvat, parametr "dest" je adresa cíle - kam se bude paměťová oblast přesouvat a parametr "num" je počet přesouvaných bajtů paměti.
Instrukce může mít parametry „src" a „dest" zadány i s offsetem a mohou být použity i pro plnění zálohované paměti PLC.

Příklad:

```
MV    PLC_MEM_BACKUP+100,BUN5,8      ;naplní 8 bajtů z BUN5 do zálohované paměti
```

instrukce	REAL	
funkce	**REAL**	**konverze DR -> DR_REAL**
syntax	**REAL**	
	REAL	**[DWRD]**
	REAL	**[QWRD]**

Instrukce **REAL** provede konverzi registru DR do reálného registru DR_REAL. Po operaci zůstává výsledek jak v reálném registru DR_REAL, tak v celočíselném registru DR64.
Pokud je požadován převod hodnoty BYTE, WORD nebo DWORD instrukce vždy před převodem znaménkově rozšíří DR registr na 64 bitů (DR64).
Pokud instrukce má parametr „QWRD" a je tedy požadován převod z rozšířeného 64 bitového DR registru, předpokládá se že rozšířený registr DR64 je před operací platný (to znamená, že předcházelo násobení, dělení nebo instrukce LOD).

instrukce	CLEAR	
funkce	**CLEAR**	**nulování paměti**
syntax	**CLEAR**	**begin, end**
	CLEAR	**GLOBAL, ALL**
	CLEAR	**INTERNAL, ALL**
1.parametr	**„begin"**	**adresa začátku nulované oblasti**
2.parametr	**„end"**	**adresa konce nulované oblasti**

Instrukce **CLEAR** slouží pro nulování paměti. Parametr "begin" je adresa začátku nulované oblasti a parametr "end" je adresa konce nulované oblasti. Instrukce vynuluje jak paměť pro globální proměnné, tak paměť pro lokální proměnné. Příslušná oblast je určena podle výskytu adres proměnných v parametru instrukce.

Instrukce "**CLEAR GLOBAL, ALL**" vynuluje všechna globální data včetně inicializačních proměnných mechanizmů a časovačů. Instrukce se používá například v modulu MODULE_CLEAR, aby nulování (start a stop) PLC programu bylo ekvivalentní se stavem, který proběhne jen při zapnutí stroje.

Instrukce "**CLEAR INTERNAL, ALL**" vynuluje všechna lokální data definovaná návrhářem PLC programu včetně "automatických" proměnných definovaných v rozvoji instrukcí jazyka TECHNOL. Instrukce se používá například v modulu MODULE_CLEAR.

3.16 Reálné operace

Dále uvedeme některé předpoklady pro práci s reálnými čísly v PLC programu. Reálná čísla se do PLC programu mohou dostat například z povelového bloku, z PLC tabulek nebo ze sdílené oblasti PLC paměti.

Instrukce, které mohou pracovat s reálnými čísly jsou:
`LOD, STO, STO1, AD, SU, MULB, DIVB, INR, DCR, INV, ABS, EQ, EQ1, LT, GT, LE, GE, REAL`

Instrukce pro práci s reálnými čísly musí mít povinně prefix „`REAL`" nebo „`CNST`" s číslem s desetinnou tečkou. Pro všechny instrukce platí, že po operaci zůstává výsledek jak v reálném registru DR_REAL, tak v celočíselném registru DR64. Proto je možno kdykoli pokračovat standardními celočíselnými operacemi. Nelze ale kombinovat celočíselné a reálné operace.

Pro zadání přímých reálných hodnot lze použít prefix „`CNST`". Pokud přímo zadaná hodnota obsahuje desetinnou tečku, považuje se to za zadání čísla v reálném tvaru. Hodnota může obsahovat znaménko.

Příklady:

```
rBUN1:      DS    8          ;pro reálnou hodnotu
rBUNDef:    DS    8
dwBUN5:     DS    4

      LOD   cnst.2.8         ;načte reálnou hodnotu 2.8
      STO   real.rBUN1       ;zapíše do reálné buňky rBUN1
      MULB  cnst.-4.8        ;DR_REAL <- rBUN1 * (-4.8)
      INR   real             ;inkrementuje DR_REAL
      DIVB  real.rBUN2       ;DR_REAL <- DR_REAL / rBUN2
      STO   real.rBUN3       ;zapíše do rBUN3

      LOD   cnst.2           ;načte celočíselnou hodnotu 2
      REAL  qwrd             ;převede na reálné číslo a nastaví DR_REAL
      AD    real.rBUN4       ;připočte rBUN4
      STO   real.rBUN5       ;zapíše reálnou hodnotu do rBUN5
      STO   dwBUN5           ;zapíše DWORD celočíselně do dwBUN5

      LOD   real.rBUN1       ;načte reálnou buňku rBUN1
      EQ    cnst.1.5         ;porovná s reálnou hodnotou 1.5
      WR    Rovnost          ;zapíše bit

      LOD_REAL_BLOCK H       ;načte hodnotu H z povelového bloku
      MULB  cnst.1000.0      ;vynásobí H * 1000 reálně
      STO   real.rBUN5       ;zapíše H*1000 reálně
      STO   dwBUN5           ;zapíše H*1000 celočíselně

      LOD   cnst.0.2         ;reálná defaultní hodnota pro konfiguraci
      STO   real.rBunDef
      CNF_GET_REAL rBUN1,'RealParam1',rBUNDef  ;načte PLC konfiguraci
```

3.17 Procedury

Ve všech souborech PLC programu mohou být definovány procedury (podprogramy) a také ve všech souborech mohou být libovolná volání procedur definovaných i v jiných souborech.

instrukce	**PROC_BEGIN** **PROC_END** **PROC_CALL**

funkce	**PROC_BEGIN**	**začátek definice procedury**
	PROC_END	**konec definice procedury**
	PROC_CALL	**volání procedury**
syntax	**PROC_BEGIN**	**name**
	PROC_END	**name**
	PROC_CALL	**name**
parametr	**„name"**	**jméno procedury**

Definice procedury se provede pomocí příkazů **PROC_BEGIN** a **PROC_END**, které mají jako parametr název procedury. Umístění procedury při její definici může být na libovolném místě v modulu MODULE_MAIN. Sled vykonávání instrukcí přeskočí definiční oblast procedury. Procedura se vykoná jen pomocí instrukce **PROC_CALL** s příslušným parametrem, který je názvem procedury. Po vykonání procedury se sled vykonávání instrukcí vrátí za místo volání procedury. Procedury mohou být volány různě ze všech souborů PLC.

Událostní procedury:
V jazyku TECHNOL jsou rezervovány některé názvy procedur, které slouží pro konkrétní účel a jsou automaticky spuštěny při dané události. Jedná se o tyto názvy procedur:

_ON_ESET	Procedura se zavolá automaticky při vzniku PLC chyby. Umístění procedury může být v libovolném souboru s PLC programem. Překladač TECHNOL zkoumá existenci takovéto procedury a v případě, že existuje, spustí ji automaticky z rozvoje instrukce ESET. (viz kapitolu „Chybová hlášení, varování a informační hlášení z PLC programu".
_ON_CMD	Procedura se zavolá automaticky při stisku tlačítka nebo softwarového menu, které mají v konfiguraci nastaven "PLCCommand". Procedura v PLC programu je nepovinná. Umístění procedury může být v libovolném souboru. Překladač TECHNOL zkoumá existenci takovéto procedury a v případě, že existuje, spustí ji automaticky při stisku tlačítka. Vstupní parametry procedury jsou: RLO = 1 ..... tlačítko stisknuto RLO = 0 ..... tlačítko puštěno DR (DWORD) ... kód z konfigurace "PLCCommand" kódem je PLC konstanta načtená pomocí instrukce CONST_GET (viz. „Konfigurace pro PLC")
_ON_EVENT	Procedura se zavolá automaticky při některých vybraných událostech systému. Procedura v PLC programu je nepovinná. Umístění procedury může být v libovolném souboru. Překladač TECHNOL zkoumá existenci takovéto procedury a v případě, že existuje, spustí ji automaticky při vybraných událostech. Typ události se předává v DR registru (DWORD):

	Hodnoty předávané v DR registru: PLCEVENT_STOP_REQ ..... požadavek na STOP PLCEVENT_STOP_EMERGENCY_REQ .... požadavek na Nouzový STOP PLCEVENT_STOP ..... STOP vykonán PLCEVENT_START ..... příkaz START PLCEVENT_START_REQ ..... požadavek na START

3.18 Práce s textovými řetězci

Pro práci s textovými řetězci je potřeba mít možnost definovat řetězec, skládat a přesouvat řetěze a konvertovat čísla na řetězce.

instrukce	**STR**

```
funkce      STR   definice textového řetězce

syntax      TX1:  STR   n
            TX1:  STR   n [,'ABCDabcd..']
            TX1:  STR   n [,PLC_MEM_BACKUP + xy]
            TX1:  STR   n [,val + xy]

1.parametr  „n"         maximální počet znaků v řetězci
2.parametr  „text2"     přímé nebo nepřímé zadání řetězce
```

Instrukce **STR** definuje textový řetězec. Instrukce musí mít povinně návěští, které budě dále sloužit pro identifikaci definovaného řetězce. Návěští bude známo a přístupno ve všech souborech PLC programu. Instrukce STR se může umístit v libovolném modulu PLC programu. (Překladač ve skutečnosti definuje také obecný pointer, který je nasměrovaný na příslušný řetězec.)

1. parametr „n"

První parametr instrukce je povinný a vyjadřuje maximální počet znaků v řetězci. (Překladač ve skutečnosti vymezí o jeden bajt víc pro koncový znak řetězce.)

Příklad:

```
TX_POKUS:   STR   50
```

2. parametr „text2"

Druhý parametr instrukce je nepovinný a může obsahovat:

Druhý parametr je přímé zadání textového řetězce. Text musí být ohraničen apostrofy a měl by mít maximálně tolik znaků, kolik vymezuje 1. parametr instrukce. (Překladač zabezpečí předplnění zadaného řetězce při každém přenosu nového PLC programu)

Příklad:

```
TX_ZPR:     STR   10,'Zapnuto'
```

Druhý parametr je odkaz do pamětí PLC_MEM_BACKUP. Paměťová oblast se používá jako zálohovaná paměť.

Příklady:

```
TX_SCR1:    STR   15, PLC_MEM_BACKUP+124

EQUI        TEXT_ZAP,124                ;symbolický offset
TX_SCR2:    STR   15, PLC_MEM_BACKUP+TEXT_ZAP
```

Druhý parametr je odkaz na libovolnou paměťovou proměnou, kromě definice textu, například definovaných pomocí instrukcí DFM a DS. Potom instrukce STR musí být umístěna v tom samém souboru.

Příklad:

```
PAM25:      DS    20          ;pole v datové části PLC programu

TX_BUN25:   STR   15,PAM25
```

instrukce	**STRCPY** **STRADD**

```
funkce   STRCPY      kopírování textových řetězců
         STRADD      spojení textových řetězců

syntax   STRCPY (STRADD)   text1, text2
         STRCPY (STRADD)   text1, 'ABCDabcd..'
         STRCPY            text1, text2 [,num]
         STRCPY            text1+xx, text2+yy
         STRCPY            text1+xx, text2+yy, num
         STRCPY            text1+BX, text2+yy, num

1.parametr „text1"    cílový textový řetězec
2.parametr „text2"    zdrojový textový řetězec
```

Instrukce **STRCPY** překopíruje textový řetězec na který ukazuje druhý operand, do textového řetězce na který ukazuje první operand. Když je druhý operand přímé zadání textového řetězce, překopíruje jej do řetězce podle prvního operandu. Oba řetězce jsou definovány pomocí instrukce STR. Řetězec podle prvního operandu musí mít rezervován dostatek prostoru.

Počet kopírovaných znaků může být určen:

Pokud v instrukci není zadán 3.parametr, který určuje počet znaků pro překopírování:		
	1.	Kopírování se ukončí výskytem koncového znaku ve zdrojovém řetězci (text2). Koncový znak je binární 0 a překopíruje se jako poslední. (cílový řetězec může být zkrácen)
	2.	Kopírování se ukončí, když je dosažena velkost cílového řetězce (text1). Zdrojový řetězec by se do cílového řetězce nevešel.

Pokud je v instrukci zadán 3. parametr, který určuje počet znaků:		
	1.	Kopírování se ukončí výskytem koncového znaku ve zdrojovém řetězci (text2). Koncový znak je binární 0 a překopíruje se jako poslední. (cílový řetězec může být zkrácen)
	2	Kopírování se ukončí, když je dosažena velkost cílového řetězce (text1). Zdrojový řetězec by se do cílového řetězce nevešel. (cílový řetězec zůstane v původní délce)
	3.	Kopírování se ukončí dosažením zadaného počtu znaků. (cílový řetězec nebude zkrácen)

Pokud potřebujeme kopírovat řetězec do jiného řetězce na konkrétní pozici, můžeme použít instrukci s offsetem (text1+10)

Když řetězce obsahují libovolná binární čísla (včetně binární 0), tak musíme místo instrukce STRCPY použít instrukci MEMCPY (viz dále).

Instrukce **STRADD** připojí textový řetězec na který ukazuje druhý operand, do textového řetězce na který ukazuje první operand. Když je druhý operand přímé zadání textového řetězce, připojí jej do řetězce podle prvního operandu.

Příklady:

```
TEXT1:      STR   50, 'prvni text '
TEXT2:      STR   20, 'druhy text '
TEXT3:      STR   50, STCH_OUT_FIELD + 100
TEXT4:      STR   3 , 'ABC'

      STRADD      TEXT1, TEST2      ;spojení textů TEXT1 <- TEXT1+TEXT2

      STRCPY      TEXT3,'OBR. '     ;naplní TEXT3 v STCH_OUT_FIELD+100
      STRADD      TEXT3,TEXT2       ;připojí k TEXT3 ještě TEXT2

      STRCPY      TEXT2+6, TEXT4    ;v řetězci TEXT2 přepíše ,text‘ na ,ABC‘
                                    ;řetězec bude zkrácen

      STRCPY      TEXT2+6, TEXT4,2  ;v řetězci TEXT2 přepíše ,te‘ na ,AB‘
                                    ;řetězec nebude zkrácen

      STRCPY      TEXT3+BX, TEXT4   ;v řetězci TEXT3 na offsetu BX se
                                    ;se přepíše 'ABC'
```

instrukce	**BINSTR** **BCDSTR**

funkce **BINSTR** **převod binárního čísla na řetězec**
BCDSTR **převod BCD hodnoty na řetězec**

syntax **BINSTR (BCDSTR) text1**
BINSTR (BCDSTR) text1 [,DWRD]

parametr **„text1"** **cílový textový řetězec**

Instrukce **BINSTR** převede binárně hodnotu z datového registra DR na řetězec na který ukazuje první parametr instrukce. První parametr instrukce ukazuje na řetězec, který musí mít velkost definovanou instrukcí STR minimálně 1 a maximálně 10 znaků (bajtů). Instrukce podle velkosti řetězce přizpůsobí převod. Pokud je číslo v DR registru menší, doplní se řetězec na prvních místech (první zleva) znaky nul.

Instrukce **BCDSTR** je podobná jako instrukce BINSTR, ale hodnotu v DR registru převádí na řetězec jako hodnotu v BCD kódu..

Instrukce mohou mít druhý parametr „DWRD", který udává, že se provede převod z rozšířeného 32 bitového DR registru.

Příklad:

```
TEXT1: STR        4                ;pro převod 4 cifry
TEXT2: STR        30

      LOD         BUN1             ;wordová buňka
      BINSTR      TEXT1            ;převede word na řetězec TEXT1
      STRCPY      TEXT2,'POCET-'
      STRADD      TEXT2,TEXT1      ;připojí převedený řetězec
```

instrukce	**STRCMP**

funkce **STRCMP** **porovnání textových řetězců**

syntax **STRCMP** **text1, text2**
STRCMP **text1, 'ABCDabcd..', num**
STRCMP **text1, text2 [,num]**
STRCMP **text1+xx, text2+yy**
STRCMP **text1+xx, text2+yy, num**

1.parametr **„text1"** **cílový textový řetězec**
2.parametr **„text2"** **zdrojový textový řetězec**

Instrukce **STRCMP** porovnává textové řetězce na které ukazuje první a druhý operand. Druhý operand může být přímé zadání textového řetězce. Řetězce jsou definovány pomocí instrukce STR. Pokud jsou textové řetězce stejné, instrukce nastaví registr RLO na hodnotu 1, jinak bude RLO mít hodnotu 0. Pokud je zadán 3. parametr „num", tak se z obou řetězců porovná jen zadaný počet znaků (pokud porovnávání nebude dřív ukončeno nerovností nebo výskytem koncového znaku 0).

Pokud instrukce neobsahuje 3. parametr o počtu znaků, bude se porovnávat počet znaků podle velkosti řetězce, na který ukazuje druhý parametr instrukce včetně koncového znaku.

Příklad:

```
TEXT1: STR        8,'abcdefgh'
TEXT2: STR        8,'abcde'
TEXT4: STR        3,'cde'


      STRCMP      TEXT1+2, TEXT4,3  ;porovnání řetězců - shoda
      JL1         OK                ;je nastaveno RLO=1

      STRCMP      TEXT1+2, TEXT4    ;porovnání řetězců - neshoda
      JL1         OK                ;je nastaveno RLO=0

      STRCMP      TEXT2+2, TEXT4    ;porovnání řetězců - shoda
      JL1         OK                ;je nastaveno RLO=1

      STRCMP      TEXT1+2, 'cde',3  ;porovnání řetězců - shoda
      JL1         OK                ;je nastaveno RLO=1
```

instrukce MEMCPY

funkce **MEMCPY** **kopírování binárních řetězců**

syntax **MEMCPY** **text1, text2 [,num]**
MEMCPY **text1, text2**
MEMCPY **text1+xx, text2+yy**
MEMCPY **text1+xx, text2+yy, num**

1.parametr „**text1**" **cílový binární řetězec**
2.parametr „**text2**" **zdrojový binární řetězec**

Instrukce **MEMCPY** překopíruje binární řetězec na který ukazuje druhý operand, do řetězce na který ukazuje první operand. Oba řetězce jsou definovány pomocí instrukce STR. Řetězec podle prvního operandu musí mít rezervován dostatek prostoru.

Instrukce je podobná instrukci STRCPY s rozdílem, že řetězce nemají definovaný koncový znak 0. Kopírování se proto neukončí výskytem koncového znaku ve zdrojovém řetězci (text2). V instrukci MEMCPY se proto vždy doporučuje definovat počet kopírovaných znaků pomocí 3. parametru. Pokud v instrukci není 3.parametr zadán, kopírování se ukončí, když je dosažena velkost cílového řetězce (text1).

instrukce MEMCMP

funkce **MEMCMP** **porovnání binárních řetězců**

```
syntax   MEMCMP          text1, text2 [,num]
         MEMCMP          text1, text2
         MEMCMP          text1+xx, text2+yy
         MEMCMP          text1+xx, text2+yy, num
```

Instrukce **MEMCMP** porovnává binární řetězce na které ukazuje první a druhý operand. Řetězce jsou definovány pomocí instrukce STR. Pokud jsou řetězce stejné, instrukce nastaví registr RLO na hodnotu 1, jinak bude RLO mít hodnotu 0.

Instrukce je podobná instrukci STRCMP s rozdílem, že řetězce nemají definovaný koncový znak 0. Porovnávání se proto neukončí výskytem koncového znaku v řetězci. V instrukci MEMCMP se vždy doporučuje definovat počet porovnávaných znaků pomocí 3. parametru.

Pokud je zadán 3. parametr „num", tak se z obou řetězců porovná jen zadaný počet znaků. Pokud instrukce neobsahuje 3. parametr o počtu znaků, bude se porovnávat počet znaků podle velkosti řetězce, na který ukazuje druhý parametr instrukce.

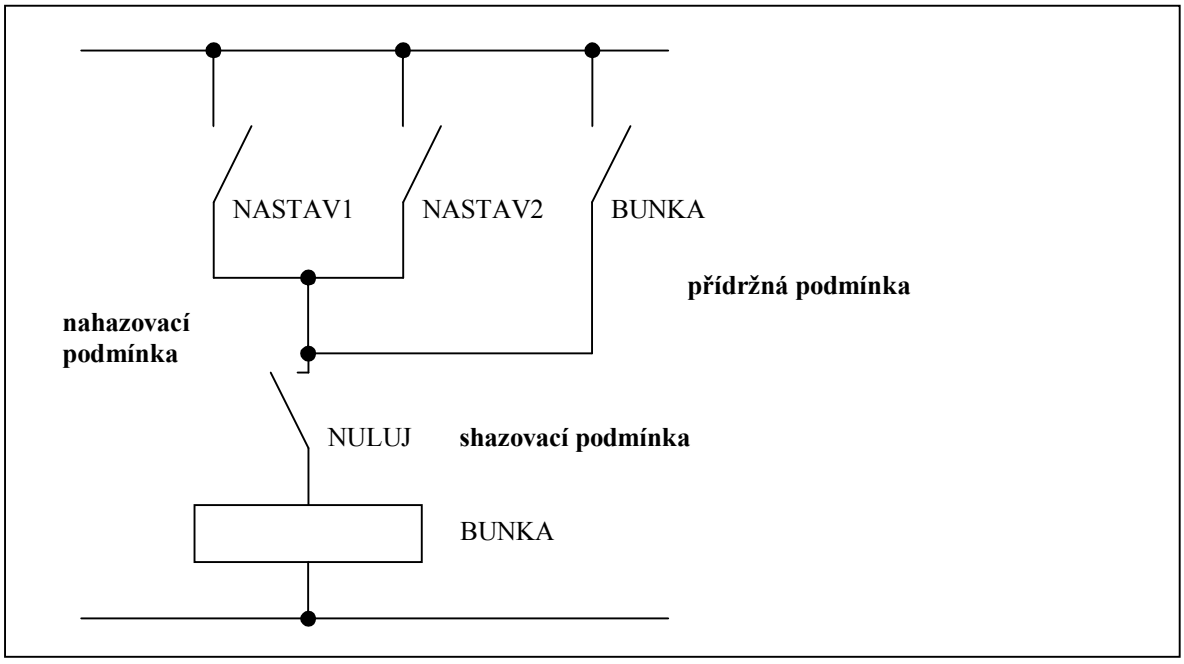
4

4. LOGICKÉ SEKVENČNÍ CELKY

4.1 Strukturalizace PLC programu

PLC program může být vytvořen různými způsoby. Klasický přístup při návrhu PLC programu je založen na navržení sekvenčně-kombinační logiky nebo přepsání reléové logiky do instrukcí jazyka PLC836. Každé relé má pak deklarovanou vlastní vnitřní bitovou buňku. Při řízení zápisu do této buňky je definována pomocí instrukcí ***nahazovací***, ***přídržná a shazovací*** podmínka. V programu je pak zápis do buňky na jediném místě.

Příklad:
Příklad klasického návrhu PLC programu.



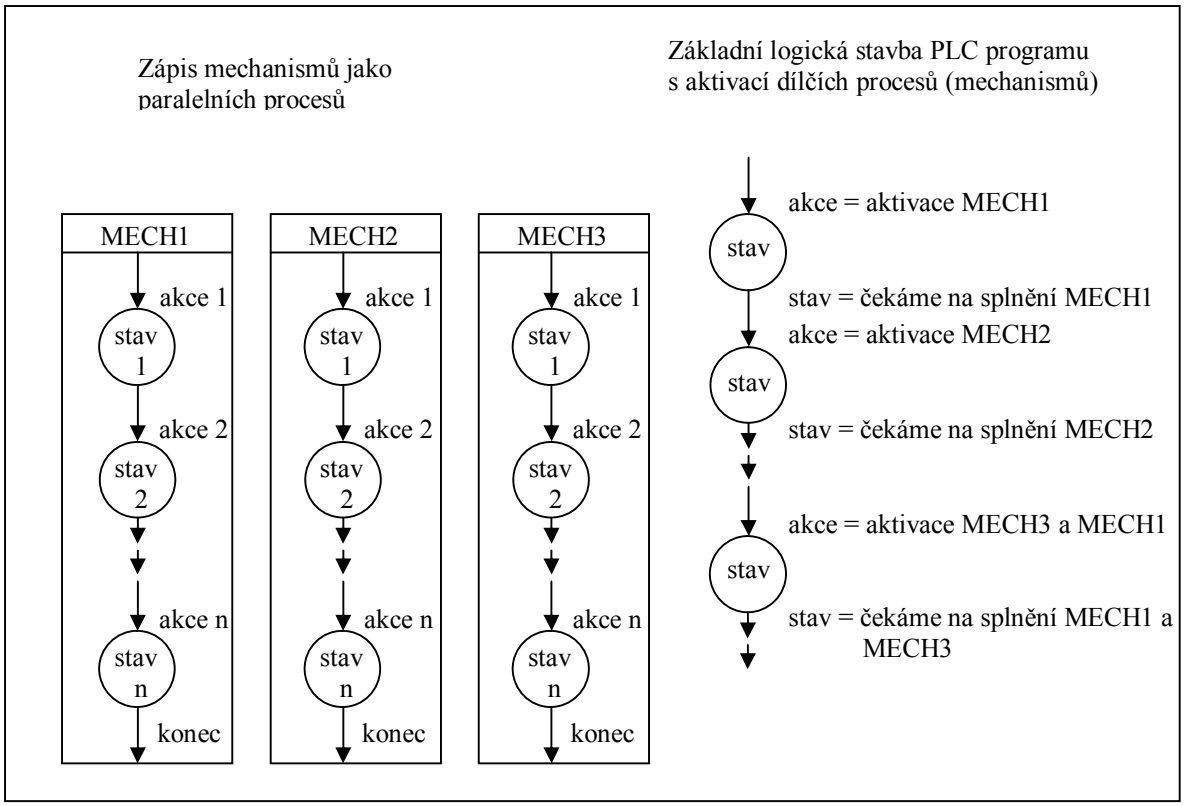

```
LDR     NASTAV1     ;nahazovací podmínka je logický součet
LO      NASTAV2     ;bitů NASTAV1 NASTAV2
LO      BUNKA       ;přídržná podmínka
LA      -NULUJ      ;bit NULUJ vynuluje klopný obvod
WR      BUNKA       ;paměťová bitová proměnná
```

Modernější přístup při návrhu PLC programu je založen na popisu jednotlivých procesů pomocí vývojových diagramů, které tvoří logické sekvenční celky (mechanismy).

Sekvenční logický celek (dále jen MECHANISMUS) je programový úsek, který se skládá z akcí a stavů. Akcí se rozumí nastavení příslušných bitů, které aktivují jednotlivé části mechanismů jako např. zapnutí hydrauliky, vypnutí stykačů ventilátorů atd. Stavem se rozumí kontrola splnění požadovaných vstupních podmínek mechanismů, např. "čekej na sepnutí kontaktu stykače od hydrauliky". Mechanismus lze znázornit vývojovým nebo stavovým diagramem.

Mechanismus se aktivuje aktivační proměnnou (NAZEV). Jedná se o bitovou proměnnou, deklarovanou automaticky zápisem mechanismu. Tuto proměnnou je možno různě nastavovat nebo testovat v programu interfejsu a tak řídit chod daného mechanismu. Mechanismus se aktivuje nastavením aktivační proměnné a po dokončení funkce mechanismu se tato proměnná automaticky vynuluje.

PLC program může obsahovat větší počet mechanismů pro řešení všech dílčích procesů stroje. (start vřetena, reverzace vřetena, stop vřetena, upínání a uvolňování os, orientovaný stop, přesun ramene, start a stop chlazení, jednotlivé kroky výměny nástroje, vyhledávání nástroje, řazení otáčkových řad atd...). V jeden okamžik může být aktivován libovolný počet mechanismů a tyto běží současně, nezávisle na sobě.



Návrh PLC programu se pomocí mechanismů velmi zjednoduší a zpřehlední. PLC program z hlediska návrhu získá ***strukturalizaci***. Strukturalizace je umožněna tím, že se pomocí mechanismů popíší všechny dílčí procesy stroje a tak hlavní logická stavba PLC programu už není zatížena jejich problematikou. Hlavní logická stavba PLC programu jenom aktivuje jednotlivé dílčí procesy (mechanismy) a kontroluje jejich splnění a případný výskyt chyb. Mechanismy se skládají z akcí a stavů. Pro každý stav jsou kontrolovány jen podmínky nevyhnutně potřebné pro přechod do následujícího stavu.

V každý okamžik možno sledovat průběh jednotlivých mechanismů: které z nich jsou aktivovány a v jakém jsou stavu. Tak přirozeně dochází k velkému ulehčení z hlediska ladění a diagnostikování provozu stroje, protože každý stav, ve kterém se mechanismus nachází je dán tzv. ***podmínkou pokračování*** (podmínková analýza). Podmínky pokračování (pro přechod do následujícího stavu) je užitečné sledovat při zastavení chodu mechanismu. Silnou stránkou mechanismů je i to, že splnění podmínek může být časově omezeno a tak nedojde k trvalému zastavení chodu mechanismu. Mechanismus se ukončí s chybou a v popisu chyby na obrazovce systému může být přesně v textové podobě vyjádřena podmínka pokračování daného mechanismu i s čísly svorek jednotlivých vstupů, které podmínka obsahuje. Tak jsou velmi ulehčeny servisní práce při poruchách.

Kromě podmínek, které možno sledovat a jsou užitečné v případě zastavení chodu mechanismu, možno sledovat i časový průběh chodu mechanismu, to je jak dlouho se průběh mechanismů zdržel v jednotlivých stavech a kterými stavy procházel (tzv. časová analýza).

4.2 Instrukce pro logické sekvenční celky

instrukce	**MECH_BEGIN** **MECH_END** **MECH_INIT**

funkce	**MECH_BEGIN**	**začátek sekvenčního celku**
	MECH_END	**konec sekvenčního celku**
	MECH_INIT	**inicializace sekvenčního celku**
syntax	**MECH_BEGIN**	**mech**
	MECH_END	**mech**
	MECH_INIT	**mech**
paraemtr	**„mech"**	**název sekvenčního celku**

Instrukce **MECH_BEGIN** musí být zapsána na začátku mechanismu. Proměnná "mech" je názvem mechanismu a současně názvem jeho aktivační bitové proměnné. Instrukce **MECH_END** musí být zapsána na konci příslušného mechanismu. Proměnná "mech" má stejný význam, t.j. je totožná s názvem v MECH_BEGIN.

Instrukce **MECH_INIT** uvede mechanismus do klidového stavu. Používá se v inicializaci interfejsu nebo v obsluze chybových stavů. Proměnná "mech" má opět stejný význam jako u MECH_BEGIN. Viz "Společné zásady" (dále).

instrukce	EX
	EX0
	EX1
	BEX

funkce	**EX**	**přerušení po dobu jednoho cyklu interfejsu**
	EX0	**přerušení činnosti pokud RLO = 0**
	EX1	**přerušení činnosti pokud RLO = 1**
	BEX	**začátek podmínky**
syntax	**EX**	
	EX0	
	EX1	
	BEX	

Instrukce pro definici stavu **EX, EX0, EX1** zastaví provádění sekvenčního logického celku do doby splnění kontrolované podmínky. Instrukce možno používat jenom uvnitř logických sekvenčních celků. Tato přerušení se nazývají stavy mechanismu.

Instrukce EX způsobí nepodmíněné zastavení provádění sekvencí na dobu jednoho cyklu interfejsu.

Instrukce BEX je podobná jako instrukce EX, ale neprovede se zastavení provádění sekvencí. Instrukce se používá pro definici začátku logické podmínky před instrukcemi EX0, EX1, TEX0 a TEX1. Instrukci je výhodné použít v časově náročných procesech, kdy ztráta jednoho cyklu interfejsu může vadit.

Instrukce EX0, resp. EX1 pozastaví provádění dalších sekvencí mechanismu po dobu, pokud RLO=0, resp. RLO=1. Před instrukcemi EX0 a EX1 může být napsána logická podmínka pro pokračování v daném stavu. Pro daný stav v každém cyklu dojde k vyhodnocování podmínek zapsaných v úseku mezi předposlední a poslední dosaženou instrukcí typu EX.

Ve skutečnosti instrukce EX0 a EX1 nečekají na daném místě pokud není splněna podmínka, ale provedou skok na konec mechanismu. V dalším průchodu mechanismem se skočí ze začátku mechanismu na předposlední instrukci typu EX a znovu se vyhodnotí podmínky po instrukci EX0 nebo EX1. Viz kapitolu "Společné zásady mechanismů" (dále).

Instrukce EX, EX0, EX1, TEX0, TEX1 a BEX jsou ***koncové instrukce*** pro logické rovnice. Instrukce nezachovávají RLO a DR registr.

instrukce	TEX0 TEX1

```
funkce      TEX0        přerušení, pokud RLO = 0 s časovou kontrolou
            TEX1        přerušení, pokud RLO = 1 s časovou kontrolou

syntax      TEX0 (TEX1)       count, time, error
            TEX0 (TEX1)       count, time, error [, code ]
            TEX0 (TEX1)       [TYPE.] count, time, error [, code ]
            TEX0 (TEX1)       TYPE. (count+n), (time+m), error [, chyba]
            TEX0 (TEX1)       - , time, error [, code]
                                          TYPE = BYTE.  WORD.

1.parametr  „count"     čítač
2.parametr  „time"      maximální doba pro přerušení čekání na podmínku
3.parametr  „error"     návěští pro obsluhu přerušení
4.parametr  „code"      vstupní hodnota pro obsluhy přerušení
```

Instrukce **TEX0**, resp. **TEX1** pozastaví provádění dalších sekvencí mechanismu, pokud RLO=0, resp. RLO=1, ale maximálně po předem stanovenou dobu "time" (BYTE, WORD, konstanta). Pokud logická podmínka pokračování, která je napsána před instrukcemi TEX0 a TEX1, nebude splněna po zadanou dobu, bude program pokračovat od návěští, které je zadáno v parametru instrukce "error".

Parametr "count" je čítač a může být typu BYTE nebo WORD. Tento parametr je nepovinný. V tomto případě si instrukce deklaruje automatickou proměnnou v lokálních datech. Místo 1. parametru se v tomto případě napíše znak pomlčky nebo “NIL”.

Jedná se o instrukce z kterých se mechanismy skládají, především protože umožňují ošetřit chybové stavy a tak nikdy nedojde k trvalému zastavení chodu mechanismu. V textovém popisu případné chyby může být pak přesně popsaná ***podmínka pokračování*** v daném stavu a tak je umožněno diagnostikování stroje.

Pro daný stav v každém cyklu dojde k vyhodnocování podmínek zapsaných v úseku mezi předposlední a posledně dosaženou instrukcí typu EX. Instrukce vyžaduje tři povinné parametry. Určení časového členu "count", žádanou dobu zpoždění "time" a návěští "error", od kterého bude program pokračovat, jestliže nebude splněna podmínka za stanovenou dobu. "Time" může být přímá hodnota nebo adresa proměnné. Viz "Společné zásady" (dále).

Čtvrtým nepovinným parametrem "code" může být číselná hodnota (většinou číslo chyby), která zůstane v datovém DR registru při skoku na návěští "error". Tím je umožněno, že z různých instrukcí TEX0 a TEX1 možno v mechanismu skákat na stejné místo "error" se společnou obsluhou chybového stavu mechanismu.

Instrukce TEX0 a TEX1 jsou ***koncové instrukce*** pro logické rovnice. Instrukce nezachovávají RLO a DR registr, kromě odskoku na návěští "error", když je použitý čtvrtý parametr "chyba".

Možnost předefinování typu operandů "citac" a "doba" je popsáno v kapitole "Způsoby předefinování typu u datových proměnných. Předefinování typu se vztahuje na oba operandy "citac" a "doba", proto v tomto případě není umožněno použít operandu "doba" deklarovaného jako konstanta.

instrukce	TIM	
funkce	**TIM**	**časové zpoždění**
syntax	**TIM**	**count, time**
	TIM	**[TYPE.] count, time**
	TIM	**TYPE.(count+n), (time+m)**
	TIM	**- , time**
		TYPE = BYTE. WORD.
1.parametr	**„count"**	**čítač**
2.parametr	**„time"**	**doba zpoždění**

Instrukce časového zpoždění **TIM** zastaví provádění sekvenčního logického celku na stanovenou dobu. Instrukce vyžaduje dva parametry. Určení časového členu "count" a žádanou dobu zpoždění "time". "Time" může být konstanta nebo adresa proměnné. Oba parametry mohou být typu BYTE nebo WORD. Instrukce TIM má podobné působení jako instrukce EX a může být použita jen uvnitř mechanismu.

Pro systémy řady CNC8x9 – DUAL od verze 6.028 je první parametr instrukce pro "citac" nepovinný. V tomto případě si instrukce deklaruje automatickou proměnnou v lokálních datech. Místo 1. parametru se v tomto případě napíše znak pomlčky nebo “NIL”.

Možnost předefinování typu operandů "citac" a "doba" je popsáno v kapitole "Způsoby předefinování typu u datových proměnných. Předefinování typu se vztahuje na oba operandy "citac" a "doba", proto v tomto případě není umožněno použít operandu "doba" deklarovaného jako konstanta.

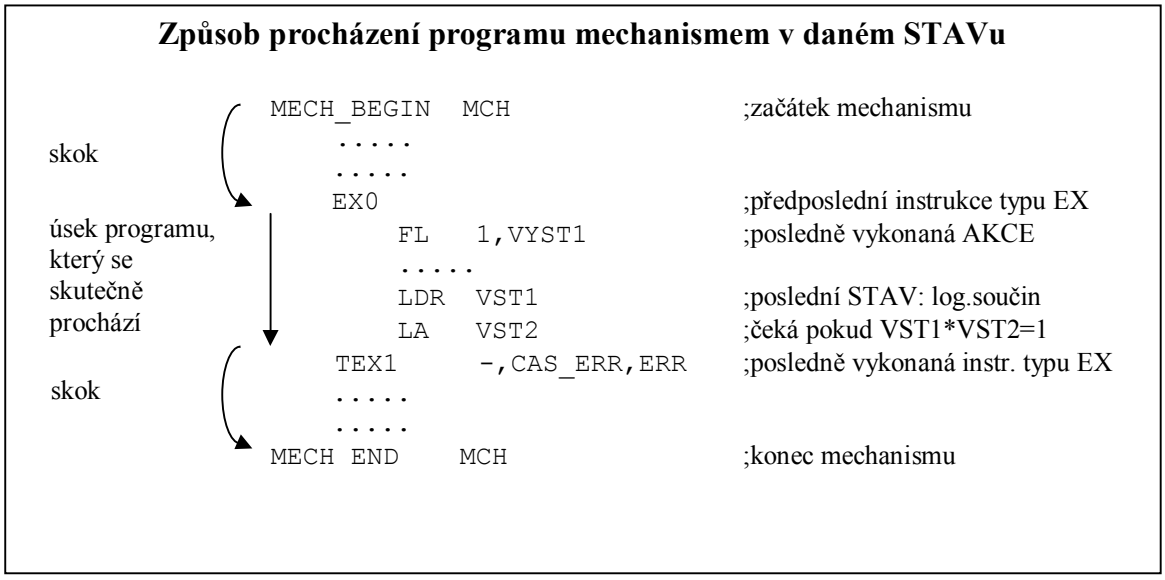
4.3 Společné zásady mechanismů

Instrukce EX, EX0, EX1, TEX0, TEX1 a TIM mohou být použity pouze uvnitř sekvenčních logických celků, vymezených instrukcemi MECH_BEGIN a MECH_END. Budeme je nazývat společně: ***instrukce typu EX***.

Modul přípravných a závěrečných funkcí je také sekvenčním logickým modulem a proto v nich lze využít instrukce všechny instrukce typu EX i samostatně. Použití těchto instrukcí v uvedených modulech způsobí zastavení provádění přípravných nebo závěrečných funkcí do doby splnění kontrolované podmínky (viz příklad).

Instrukce EX0, EX1, TEX0, TEX1 a TIM pozastaví provádění dalších sekvencí mechanismu po dobu danou podmínkou (pokud RLO=0, resp. RLO=1) nebo dobu danou nastaveným časem (u instrukci TIM). Před instrukcemi EX0, EX1, TEX0 a TEX1 může být napsána logická podmínka pro pokračování v daném stavu. Pro daný stav v každém cyklu dojde k vyhodnocování podmínek zapsaných v úseku mezi předposlední a poslední dosaženou instrukcí typu EX.

Ve skutečnosti instrukce typu EX nečekají na daném místě pokud není splněna podmínka, ale provedou skok na konec mechanismu. V dalším průchodu mechanismem se skočí ze začátku mechanismu na předposlední instrukci typu EX a znovu se vyhodnotí podmínky po poslední instrukci typu EX.



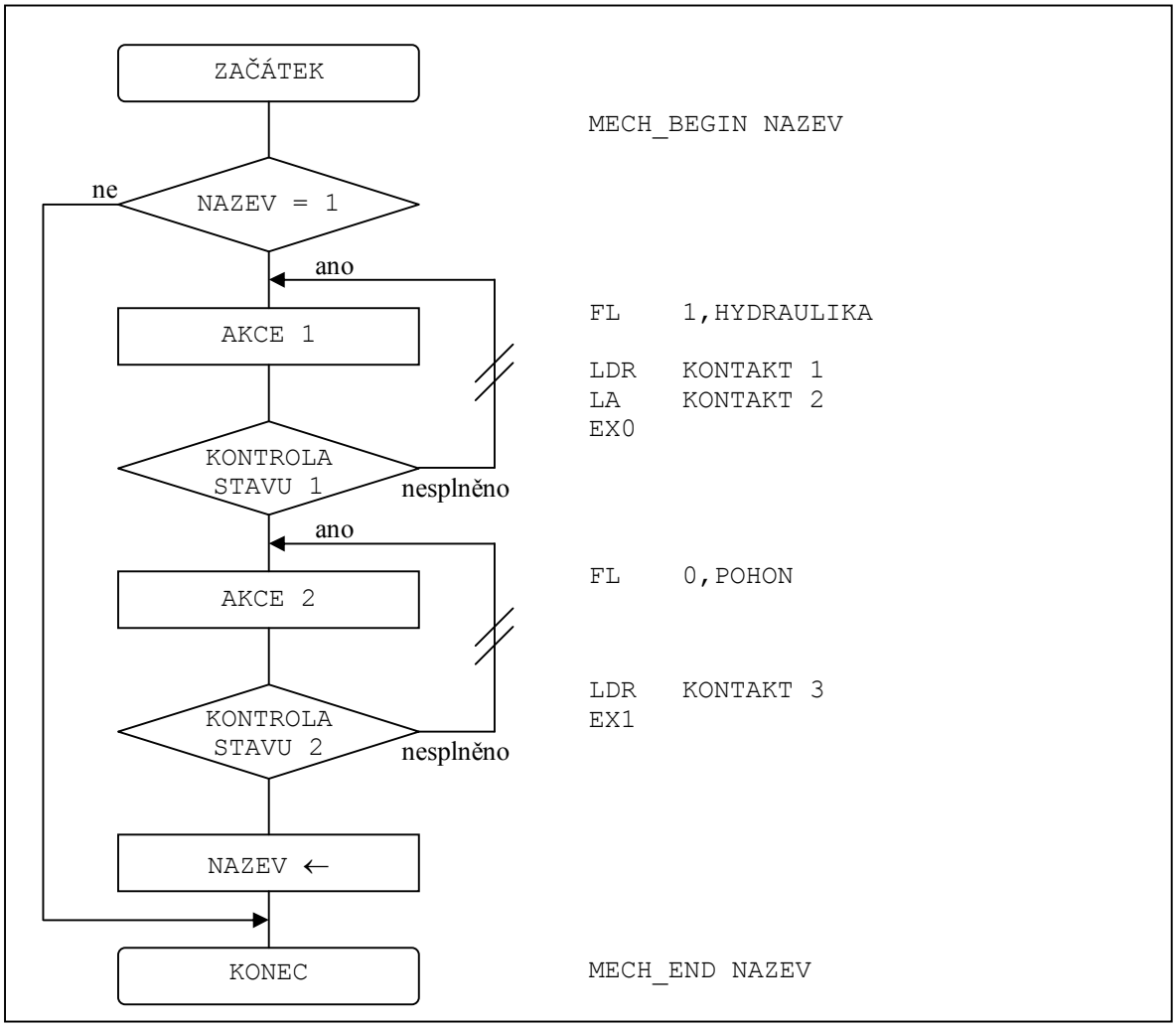
Mechanismy možno aktivovat a dezaktivovat i z jiných mechanismů. Případ ***dezaktivace mechanismů*** se provede instrukci MECH_INIT, která vynuluje bitovou řídící proměnnou mechanismu a nastaví pokračující adresu na začátek. Dezaktivaci mechanismů je výhodné použít v případě ***protichůdných mechanismů,*** kdy mechanismus s větší prioritou dezaktivuje z bezpečnostních důvodů svůj protichůdný mechanismus. Například mechanismus stopu vřetene dezaktivuje případně rozpracovaný mechanismus pro start vřetene.

Mechanismy je také možno zacyklit a tak se mohou s výhodou používat i ***trvale zacyklené mechanismy.*** Tyto pak mají vlastnost sekvenčních driverů. Trvale zacyklený mechanismus musí mít v těle cyklu aspoň jednu instrukci typu EX !

Definicí mechanismu *"mech"* se provede automatická deklarace bitové proměnné s názvem "*mech*", deklarace paměťové buňky typu WORD pro pokračující adresu mechanismu a paměťové buňky typu WORD s názvem "mech_LINE" pro aktuální linku programu, ve které se mechanismus nachází. Při kompilaci PLC programu programem TECHNOL (viz. dále) se vytvoří také kontrolní listing programu s rozšířením ".LS1" který je doplněn o čísla řádků (linky) programu. Při každé definici stavu pomocí příkazů EX, EX0, EX1, TEX0, TEX1 a TIM se pro kontrolu zapisuje číslo linky do buňky *"mech*_LINE". Pomocí ní je umožněno jednoduché sledování, ve kterém stavu se mechanismus nachází. Pokud mechanismus není aktivován, je buňka "*mech*_LINE" vynulována.

Příklad:
Nastartování mechanismu s názvem "CW" v přípravních funkcích a čekání na provedení mechanismu:

```
FL    1,CW  ;Nastavení aktivační proměnné.
EX
LDR   CW    ;Kontrola vykonání mechanismu.
EX1         ;Čeká, pokud CW = 1
```



Příklad:
Nastartování mechanismu s názvem "CW" v přípravných funkcích a čekání na provedení mechanismu po dobu 10 sec. Jestli se mechanismus za 10 sec neprovede, pokračuj na obsluhu chyby ERR1:

```
      FL          1,CW              ;Nastavení aktivační proměnné.
      EX
      LDR         CW                ;Kontrola vykonání mechanismu.
      TEX1        CIT1,500,ERR1     ;Čeká, pokud CW = 1, ale maximálně
      ....                          ;10 sec. Jestli CW = 1 déle než 10
                                    Úsec., pokračuje na ERR1.

ERR1: ESET        CHYBA1            ;Obsluha chyby. Překročena časová
                                    ;kontrola správnosti chodu mechanismu.
      ....
```

Příklad:
Zapište mechanismus pro reverzaci otáček vřetena z CW do CCW. Mechanismus nazvěte CWCCW.

```
MECH_BEGIN        CWCCW             ;Začátek mechanismu.
      FL          0,SMP             ;Vynuluj stykač motoru-pozitiv.
      LDR         KSMP
      EX1                           ;Čekej, pokud kontakt stykače
                                    ;motoru-pozitiv je v jedničce.
      TIM         -,D02             ;Zpoždění 0,2 sec.
      FL          1,SMN             ;Zapni stykač motoru-negativ.
      LDR         KSMN
      EX0                           ;Čekej, pokud kontakt stykače
                                    ;motoru-negativ je v nule.
MECH_END          CWCCW             ;Konec mechanismu.
```

Poznámka:
V reálném případě by bylo vhodnější v mechanismu CWCCW místo instrukcí EX1 a EX0 použít instrukce TEX1 a TEX0, tak jak je to v následujícím příkladě.

Příklad:
Zapište mechanismus pro reverzaci otáček vřetena z CW do CCW. Mechanismus nazvěte CWCCW. Jestli nespadne kontakt od stykače motoru-pozitiv KSMP do doby 1 sec., zahlas chybu 4.04. Jestli nepřijde kontakt od stykače motoru-negativ KSMN do doby 1 sec., vypni stykač motoru a zahlas chybu 4.05.

```
      EQUI        D1,50             ;Doba 1 sec.
      EQUI        D02,10            ;Doba 0,2 sec.

MECH_BEGIN        CWCCW             ;Začátek mechanismu.
      FL          0,SMP             ;Vypni stykač motoru-pozitiv.
      LDR         KSMP
      TEX1        -,D1,CW_ERR,04    ;Čekej, pokud kontakt stykače motoru
                                    ;pozitiv je v jedničce.
      TIM         -,D02             ;Zpoždění 0,2 sec.
      FL          1,SMN             ;Zapni stykač motoru negativ.
      LDR         KSMN
      TEX0        -,D1,CW_ERR,05    ;Čekej, pokud kontakt stykače
                                    ;motoru-negativ je v nule, ale
                                    ;maximálně 1 sec.
      JUM         CW_END            ;Skok na konec mechanismu.
                                              ;Obsluha chyby mechanismu.
CW_ERR:
      FL          0,SMN             ;Vypni stykač motoru-negativ.
      ESET                          ;Nastav chybu 05, 04
CW_END:
MECH_END          CWCCW             ;Konec mechanismu.
```

5

5. STRUKTURA PLC PROGRAMU

Struktura PLC programu je navržena s ohledem na co nejefektivnější návrh programu při přizpůsobení CNC systému na stroj.

5.1 Moduly jazyka TECHNOL

Moduly jazyka PLC836 byly vytvořeny pro zjednodušení práce při návrhu PLC programu. Zjednodušení nastává ze dvou různých pohledů na tvorbu PLC programu. V první řadě se jedná o zjednodušení navázání a synchronizace PLC automatu na CNC systém. Například modul přípravných funkcí (popsáno dále) se nastartuje jen po odstartování bloku, má vlastnosti jako sekvenční logický celek (mechanismus) a prochází jednorázově. To znamená, že všechny příkazy, které budou v tomto modulu umístěny se vykonají automaticky po startu bloku v přípravných funkcích. Další přínos spočívá v strukturalizaci PLC programu, jak už bylo popsáno dříve. Modul přípravných a závěrečných funkcí slouží jako aktivační modul jednotlivých mechanismů, které řeší dílčí procesy stroje.

Při zápisu programu programovatelného interfejsu je třeba dodržovat určitá pravidla a doporučení. Struktura programu je pevně stanovena a programátor ji musí dodržet. Program musí začínat klíčovým slovem **DATA**, za kterým programátor definuje použité proměnné a definuje jejich délku.

Za klíčovým slovem **DATA_END** následují další moduly, uvedené na obr. Názvy modulů jsou povinné. Na jejich pořadí nezáleží, doporučuje se však zachovat pořadí uvedené na obrázku. Některé moduly musí být uvedeny povinně a některé jsou nepovinné.

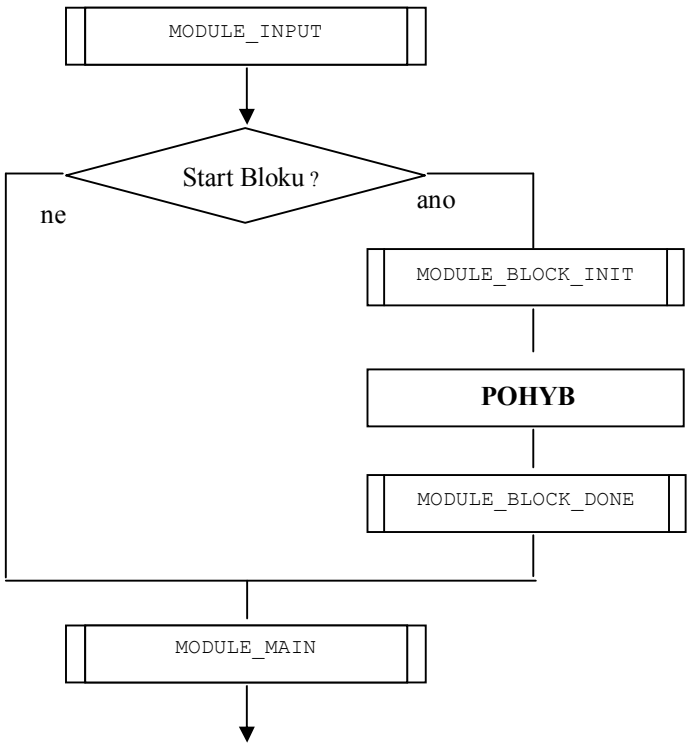
Za posledním modulem 1. souboru musí být uvedeno klíčové slovo **STOP.**

Na obrázku je uvedena struktura minimální verze programu, který se bezchybně přeloží překladačem TECHNOL. Tento program samozřejmě nevykonává žádné funkce PLC programu. Takovou prázdnou strukturu PLC programu možno nazvat ***nulový PLC program***. Všechny důležité proměnné v rozhraní PLC-CNC systém jsou přednastaveny tak, že CNC systém i s nulovým PLC programem může jezdit a vykonávat všechny funkce, které nesouvisí s technologií stroje. Návrh nového PLC programu je proto vhodné postupně vytvářet ve struktuře nulového PLC programu.

Program PLC programu prochází modulem `MODULE_INPUT`. Za modulem `MODULE_INPUT` se program větví v závislosti na činnosti systému. Pokud není odstartován nový blok partprogramu, program neprochází moduly `MODULE_BLOCK_INIT` a `MODULE_BLOCK_DONE`, ale pokračuje modulem `MODULE_MAIN`. Pokud je odstartován nový blok, projde program výše uvedenými moduly, mezi kterými nastane, pokud je programován, také pohyb souřadnic. Moduly `MODULE_INIT`, `MODULE_CLEAR`, `MODULE_DONE` a `MODULE_HALT` se vykonají pouze v případě, že jsou vyvolány. V další kapitole je přesnější popis modulů.

Jednotlivé moduly se programují podle funkce, ke které jsou určeny a která vyplývá již z jejich názvu. Jak již bylo uvedeno, nemusí být moduly vůbec naplněny. V dalším textu jsou uvedeny nejčastější funkce, které se obyčejně v daných modulech programují.

```
DATA
    ;definice globálních dat
    ;přístupnost dat ve všech souborech
DATA_END

DATA_LOCAL
    ;definice lokálních dat
    ;přístupnost dat jen v jednom souboru
DATA_LOCAL_END

MODULE_INPUT
    ;modul vstupů
    ;průběh 20ms
MODULE_INPUT_END

MODULE_BLOCK_INIT
    ;modul přípravných funkcí
    ;rychlý průběh
MODULE_BLOCK_INIT_END

MODULE_BLOCK_DONE
    ;modul závěrečných funkcí
    ;rychlý průběh
MODULE_BLOCK_DONE_END

MODULE_MAIN
    ;modul provoz
    ;průběh 20ms
MODULE_MAIN_END

MODULE_INIT
    ;inicializační modul
    ;jednoprůchodový, v dalších souborech nepovinný
MODULE_INIT_END

MODULE_DONE
    ;ukončení činnosti PLC
    ;jednoprůchodový,  nepovinný
MODULE_DONE_END

MODULE_CLEAR
    ;nulování PLC programu
    ;jednoprůchodový, v dalších souborech nepovinný
MODULE_CLEAR_END

MODULE_HALT
    ;závažná chyba systému
    ;jednoprůchodový
MODULE_HALT_END

MODULE_FAST
    ;rychlý modul PLC programu
    ;průběh 1ms, nepovinný
MODULE_FAST_END

MODULE_CONT
    ;nepřerušitelný modul PLC
    ;průběh 20ms, nepovinný
MODULE_CONT_END

STOP
```



MODULE_INIT

(MODULE_DONE)

MODULE_CLEAR

MODULE_HALT

(MODULE_FAST)

(MODULE_CONT)

Struktura PLC programu

modul DATA

Modul globálních dat začíná klíčovým slovem **DATA** a končí klíčovým slovem **DATA_END**.
Modul je povinný v každém souboru PLC.

Každý soubor PLC programu musí povinně začínat klíčovým slovem DATA, za kterým následují deklarace proměnných použitých v PLC programu. Data deklarovaná v tomto modulu mají **globální** charakter, to znamená, že jsou automaticky známá a přístupná ve všech souborech PLC programu. Modul DATA může být použit v každém souboru s PLC programem jen jednou a to na samém začátku souboru.

Příklad:

```
DATA                                        ;Začátek deklarace dat

      BUN1:       DS    2                   ;Word
      PAM10:      DFM ,,ALFA,,,BETA,,       ;Bitové proměnné

DATA_END                                    ;Konec deklarace dat
```

modul DATA_LOCAL

Modul lokálních dat začíná klíčovým slovem **DATA_LOCAL** a končí klíčovým slovem **DATA_LOCAL_END**
Modul je nepovinný. Modul může být použit v každém souboru PLC i vícekrát.

Jedná se o nepovinný modul PLC programu pro deklarování lokálních proměnných. Data deklarovaná v tomto modulu mají **lokální** charakter, to znamená, že jsou známá a přístupná jen v souboru PLC programu, kde se modul vyskytuje. Modul DATA_LOCAL může být použit v každém souboru s PLC programem i vícekrát a může být přitom vnořen do jiných modulů. Lokální data se používají i pro definování "automatických" proměnných v rámci rozvoje některých instrukcí jazyka TECHNOL.

Data definovaná v tomto modulu jsou v této verzi neviditelná i pro ladící program WINTECHNOL. Když je potřeba pro ladění PLC programu zviditelnit lokální proměnné, dočasně přemístíme modul DATA_LOCAL do těla modulu DATA. Když je modul DATA_LOCAL umístěn uvnitř modulu DATA, který má globální charakter, zviditelní se lokální data také pro WINTECHNOL.

modul MODULE_INPUT

Modul začíná klíčovým slovem **MODULE_INPUT** a končí klíčovým slovem **MODULE_INPUT_END**.
Modul je povinný v 1. souboru PLC a v dalších souborech PLC se nesmí použít.
Modul je procházen v rastru 20ms.

Modul se aktivuje jako první v průběhu každého PLC cyklu a nemá žádná omezení. V tomto modulu se obvykle provádí čtení vstupních portů do deklarované paměti PLC. Ty vstupy, které přímo ovlivňují blok zpětného hlášení (například limitní a referenční spínače) se přepíší v požadované formě do bloku zpětného hlášení.

modul	**MODULE_BLOCK_INIT**

Modul začíná klíčovým slovem M**ODULE_BLOCK_INIT** a končí klíčovým slovem **MODULE_BLOCK_INIT_END**.
Modul je povinný v 1. souboru PLC a v dalších souborech PLC se nesmí použít.
Modul je procházen v rychlém rastru po startu bloku (závisí jen od výkonnosti procesoru).

Modul MODULE_BLOCK_INIT se odstartuje jen po odstartování bloku. Modul má vlastnosti jako sekvenční logický celek (mechanismus). Modul může sloužit jako aktivační modul jednotlivých mechanismů, které řeší dílčí procesy stroje.

V modulu se obvykle řeší akce, které jsou typické jako přípravné nebo též počáteční funkce bloků partprogramu, např. roztočení vřetena, zapnutí chlazení nebo uvolnění osy atd. Modul je v činnosti pouze při startu bloku.

Modul přípravných funkcí je logický sekvenční celek a proto se v něm můžou používat všechny **instrukce typu EX** , platné pro sekvenční celky (viz kapitola "Logické sekvenční celky"). Řízení průchodu v modulu přípravných funkcí je popsáno v kapitole "Řízení průchodu supervizorem interfejsu".

modul	**MODULE_BLOCK_DONE**

Modul začíná klíčovým slovem **MODULE_BLOCK_DONE** a končí klíčovým slovem **MODULE_BLOCK_DONE_END**.
Modul je povinný v 1. souboru PLC a v dalších souborech PLC se nesmí použít.
Modul je procházen v rychlém rastru v závěrečných funkcích odstartovaného bloku (závisí jen od výkonnosti procesoru).

V modulu se obvykle řeší akce, které jsou typické jako závěrečné funkce bloků partprogramu, např. stop vřetena, vypnutí chlazení atd. Modul přípravných funkcí je logický sekvenční celek a proto se v něm můžou používat všechny **instrukce typu EX** , platné pro sekvenční celky (viz kapitola "Logické sekvenční celky").

modul	**MODULE_MAIN**

Modul začíná klíčovým slovem **MODULE_MAIN** a končí klíčovým slovem **MODULE_MAIN_END**.
Modul je povinný v každém souboru PLC.
Modul je procházen v rastru 20ms.

V modulu se obvykle řeší funkce, které musí být trvale procházeny (základní logika stroje). Je zde vhodné umístit mechanismy. **Instrukce typu EX** , platné pro sekvenční celky (viz kapitola "Logické sekvenční celky") možno použít jen v rámci mechanismů.
Provádí se zde též vysílání výstupů. Do tohoto modulu je možné začlenit PLC programy, využívající změnové signály od systému.

modul	**MODULE_INIT**

Modul začíná klíčovým slovem **MODULE_INIT** a končí klíčovým slovem **MODULE_INIT_END**.
Modul je povinný v 1. souboru PLC a v dalších souborech PLC je nepovinný.
Moduly ze všech PLC souborů se prochází jednoprůchodově při startu PLC programu.

Tento modul může být použitý ve všech souborech PLC programu. Modul slouží např. k inicializaci proměnných PLC programu a jiných akcí, které je nutné provést při startu PLC programu. Ke startu PLC programu může dojít při prvním zapnutí systému nebo na příkaz „**START PLC**“ například z ladícího programu Wintechnol. Moduly nejsou volány trvale, ale provedou se jednorázově pouze po startu PLC programu.

PLC program umístěný v tomto modulu může získat informaci, zda byl start PLC programu způsoben prvním zapnutím systému nebo to byl příkaz „START PLC“ například z ladícího programu Wintechnol. Informaci získá z datového DR registru za začátkem modulu.

Pokud inicializace PLC programu trvá nějakou dobu, například se čtou data z PLC tabulky a plní se sdílená paměť, tak se systému musí dát zpráva o ukončení této činnosti pomocí instrukce **MODULE_INIT_FINISHED.** Táto instrukce je nepovinná a pokud nebude použita, tak inicializace proběhne hned po průchodu modulem MODULE_INIT. Pokud ale v daném souboru je instrukce MODULE_INIT_FINISHED použita, systém čeká s pokračováním činnosti při inicializaci až do doby průchodu touto instrukcí (až potom proběhne například 1. centrální anulace). V tomto případě je vhodné v modulu MODULE_INIT aktivovat mechanizmus, který je umístěn standardně v modulu MODULE_MAIN a v něm na konci po ukončení činnosti inicializace je použita instrukce MODULE_INIT_FINISHED. Instrukce MODULE_INIT_FINISHED může být použita ve všech souborech PLC programu a systém pak čeká s ukončením inicializace na všechny tyto instrukce.

Datový registr DR	Význam
0	Start PLC programu po zapnutí systému
1	Příkaz pro start PLC programu (například po stopu PLC programu)

Příklad:

```
MODULE_INIT                              ;Začátek modulu inicializace
      EQ          CNST.1                 ;Je to na příkaz START PLC ?
      JL1         STARTPLC
      ;....

      FL          1,MECH_INICIALIZACE    ;Start mechanizmu inicializace
                                         ;na který se musí čekat
      ;....


MODULE_MAIN                              ;umístěno v MODULE_MAIN
      ;....

      MECH_BEGIN  MECH_INICIALIZACE      ;Mechanizmus inicializace
            ;....

            MODULE_INIT_FINISHED         ;Konec čekání na inicializaci
      MECH_END    MECH_INICIALIZACE

      ;....
```

modul	**MODULE_DONE**

Modul začíná klíčovým slovem **MODULE_DONE** a končí klíčovým slovem **MODULE_DONE_END**.
Modul je nepovinný a může se použít ve všech souborech PLC.
Moduly ze všech PLC souborů se prochází jednoprůchodově při ukončení PLC programu.

Tento modul může být použitý ve všech souborech PLC programu. Modul slouží k činnosti potřebné při stopu PLC, např. k deaktivaci pohonů a k zálohování. Ke stopu PLC programu může dojít při vypínání systému nebo na příkaz „**STOP PLC**“ například z ladícího programu Wintechnol. Moduly nejsou volány trvale, ale provedou se jednorázově pouze při stopu PLC programu.

PLC program umístěný v tomto modulu může získat informaci, zda byl stop PLC programu způsoben vypínáním systému nebo to byl příkaz „STOP PLC“ například z ladícího programu Wintechnol. Informaci získá z datového DR registru za začátkem modulu.

Pokud závěrečné operace PLC programu trvají nějakou dobu, například se deaktivují pohony, tak se systému musí dát zpráva o ukončení této činnosti pomocí instrukce **MODULE_DONE_FINISHED.** Táto instrukce je nepovinná a pokud nebude použita, tak ke stopu dojde hned po průchodu modulem MODULE_DONE. Pokud ale v daném souboru je instrukce MODULE_DONE_FINISHED použita, systém čeká s pokračováním činnosti při ukončování až do doby průchodu touto instrukcí (až potom se vypne systém). V tomto případě je vhodné v module MODULE_DONE aktivovat mechanizmus, který je umístěn standardně v modulu MODULE_MAIN a v něm na konci po ukončení činnosti je použita instrukce MODULE_DONE_FINISHED. Instrukce MODULE_DONE_FINISHED může být použita ve všech souborech PLC programu a systém pak čeká s ukončením činnosti PLC na všechny tyto instrukce.

Datový registr DR	Význam
0	Stop PLC programu při vypínání systému
1	Příkaz pro stop PLC programu (například pro načtení nového PLC programu)

Příklad:

```
MODULE_DONE                                ;Začátek modulu ukončení
      EQ          CNST.1                   ;Je to na příkaz STOP PLC ?
      JL1         STOPPLC
      ;....

      FL          1,MECH_DEAKTIVACE        ;Start mechanizmu ukončení
                                           ;na který se musí čekat
      ;....


MODULE_MAIN                                ;umístěno v MODULE_MAIN
      ;....

      MECH_BEGIN  MECH_DEAKTIVACE          ;Mechanizmus deaktivace
            ;....

            MODULE_DONE_FINISHED           ;Konec čekání na deaktivaci
      MECH_END    MECH_DEAKTIVACE

      ;....
```

modul	**MODULE_CLEAR**

Modul začíná klíčovým slovem **MODULE_CLEAR** a končí klíčovým slovem **MODULE_CLEAR_END**.
Modul je povinný v 1. souboru PLC a v dalších souborech PLC je nepovinný.
Moduly ze všech PLC souborů se prochází jednoprůchodově při nulování PLC programu.

Modul může sloužit k nulování proměnných, uvedení PLC programu do výchozího stavu. Modul se spustí pouze na příkaz pro nulování PLC (například z ladícího programu Wintechnol).
Tento modul může být použitý ve všech souborech PLC programu.

modul	**MODULE_HALT**

Modul začíná klíčovým slovem **MODULE_HALT** a končí klíčovým slovem **MODULE_HALT_END**.
Modul je povinný v 1. souboru PLC a v dalších souborech PLC se nesmí použít.
Modul se prochází jednoprůchodově při vážné chybě systému, kdy budě ukončena činnost.

V modulu se programují činnosti, které se mají vykonat při závažné chybě systému předtím, než systém skončí ve stavu HALT. Doporučuje se zde programovat například vypnutí silové části stroje.

modul	**MODULE_FAST**

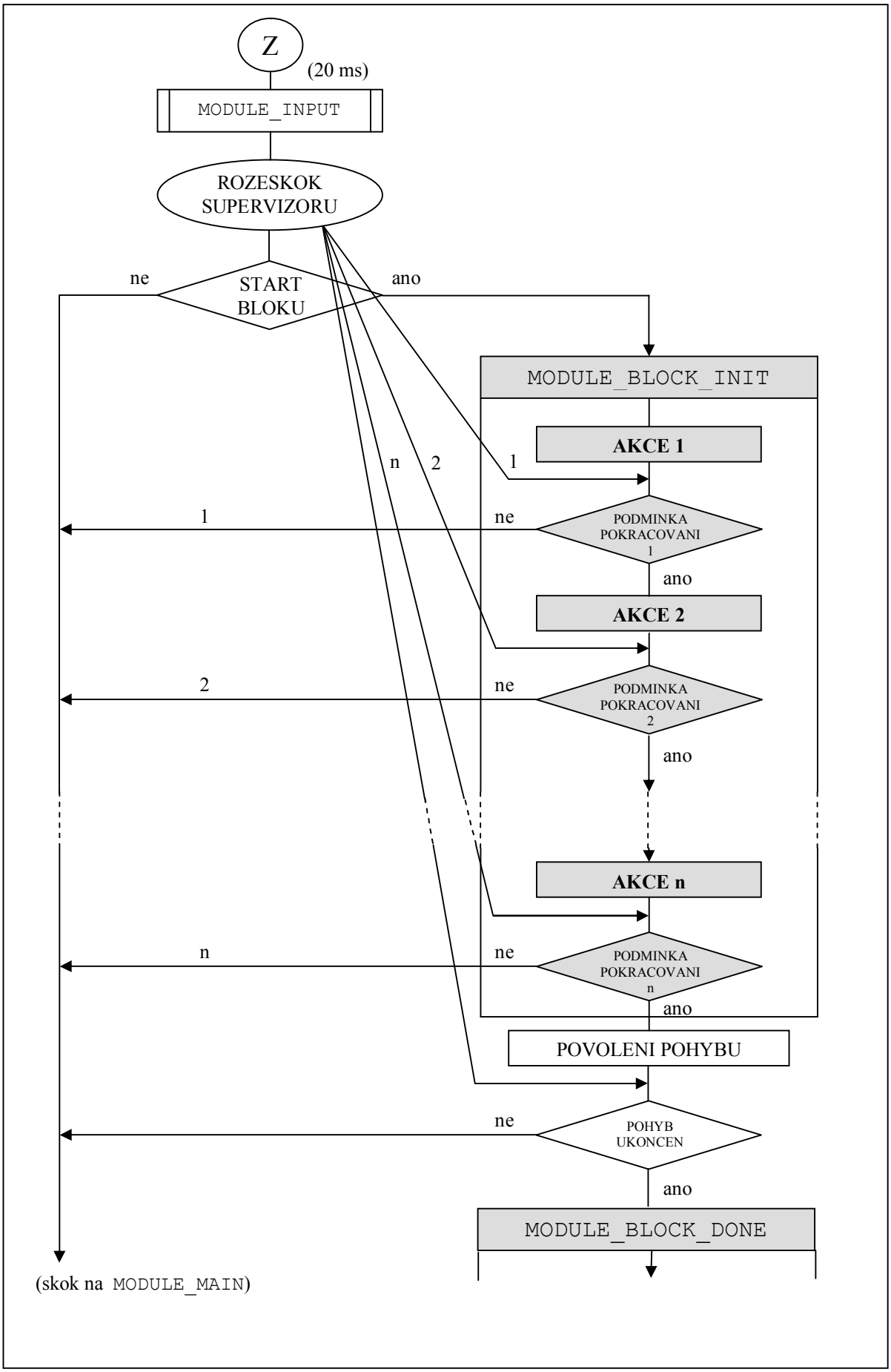
Modul začíná klíčovým slovem **MODULE_FAST** a končí klíčovým slovem **MODULE_FAST_END**.
Modul je nepovinný a může se použít ve všech PLC souborech.
Modul je procházen v rastru interpolátoru a servosmyček (1ms).

V modulu se programují činnosti, které mají probíhat v rychlejším časovém rastru než 20 ms. Modul MODULE_FAST je aktivován ve stejných časových intervalech jako softwerová polohová vazba – 1ms. V modulu MODULE_FAST můžou být naprogramovány logické sekvenční celky. V modulu může být povoleno ladění pro sledování registrů, ale je zakázáno použít break-pointy.

modul	**MODULE_CONT**

Modul začíná klíčovým slovem **MODULE_CONT** a končí klíčovým slovem **MODULE_CONT_END**.
Modul je nepovinný, smí se použít v 1. souboru PLC a v dalších souborech PLC se nesmí použít.
Modul je procházen v rastru 20ms.

V modulu se programují činnosti, které nejsou přerušitelné ladícími prostředky. Tato vlastnost může být při ladění programu vhodná pro naprogramování „životně důležitých funkcí stroje“. Logika naprogramovaná v modulu MODULE_CONT se jeví, že probíhá paralelně s hlavními moduly PLC programu.


Z
(20 ms)
MODULE_INPUT
ROZESKOK SUPERVIZORU
START BLOKU
ne
ano
MODULE_BLOCK_INIT
AKCE 1
n
2
1
1
ne
PODMINKA POKRACOVANI 1
ano
AKCE 2
2
ne
PODMINKA POKRACOVANI 2
ano
AKCE n
n
ne
PODMINKA POKRACOVANI n
ano
POVOLENI POHYBU
ne
POHYB UKONCEN
ano
MODULE_BLOCK_DONE
(skok na MODULE_MAIN)

5.3 Řízení průchodu supervizorem interfejsu

Supervizor programovatelného interfejsu řídí průchod PLC programu následujícím způsobem. Po odstartování bloku odevzdá řízení do modulu **MODULE_BLOCK_INIT** a vykoná se úsek programu po první výskyt instrukce definice stavu, to je splnění určité podmínky. Jedná se o instrukci **typu EX** (viz kapitola "Logické sekvenční celky"). V každém dalším průchodu interfejsu se kontroluje jen splnění této poslední podmínky pokračování, to je oblast programu mezi předposlední a poslední instrukcí typu EX.

Po jejím splnění pokračuje průchod modulu přípravných funkcí po další podmínku. Podmínkami může být i provedení aktivovaných mechanismů, jak to bylo popsáno v kapitole "Logické sekvenční celky" - příklad nastartování mechanismu :

```
FL          1,CW        ;nastavení aktivační proměnné
EX
LDR         CW          ;kontrola vykonání mechanismu
EX1
```

Po vykonání celého modulu přípravných funkcí supervizor interfejsu povolí případný pohyb pro interpolátor a čeká v tomto stavu, pokud není splněna podmínka dosažení programované polohy.
Po potvrzení programované polohy supervizor odevzdá řízení modulu **MODULE_BLOCK_DONE.** Modul závěrečných funkcí je také logický sekvenční celek a program ním projíždí stejným způsobem, jako v modulu přípravných funkcí.

Rychlost průběhu bloku bude závislá na tom, jak se navrhne modul přípravných a závěrečných funkcí. Instrukce `EX` například způsobí prodlevu v průběhu PLC programu. Tyto instrukce je ale velmi vhodné použít, když má dojít k čekání na splnění určité podmínky, což je naprogramováno pomocí instrukcí `EX0, EX1, TEX0` nebo `TEX1`. Návrhář PLC programu by měl dbát o to, aby v případě že se jedná o čistě pohybový blok ve kterém není programovaná žádná technologie, nedošlo ke zdržení v modulech `MODULE_BLOCK_INIT` a `MODULE_BLOCK_DONE`. Při nedodržení této podmínky by mohlo dojít k zasekávání plynulé jízdy. **Pohybový blok bez technologie musí moduly MODULE_BLOCK_INIT a MODULE_BLOCK_DONE procházet jednoprůchodově.**

Modul **MODULE_MAIN** se startuje v každém taktu PLC programu (20 ms) a tento už není sám o sobě logickým sekvenčním celkem. V něm jsou umístěny sekvenční celky (mechanismy) pomocí příkazů **MECH_BEGIN** a **MECH_END**.

5.4 Více souborů pro psaní PLC programu

PLC program se skládá z hlavního souboru, který obsahuje všechny povinné moduly programu (`MODULE_INPUT`, `MODULE_BLOCK_INIT`, `MODULE_BLOCK_DONE`, `MODULE_MAIN`, `MODULE_HALT`, `MODULE_CLEAR` a `MODULE_INIT`). Kromě hlavního souboru, PLC program může být napsán v dalších samostatných souborech (v současné verzi celkem maximálně 64 souborů). Další soubory můžou mít deklarována data a jsou pokračováním modulu `MODULE_MAIN` z hlavního souboru.

Soubory PLC programu je možno využít pro připojování odladěných knihovních funkcí PLC programu.
Soubory PLC programu musí splňovat:

a) Každý z dalších souborů PLC programu musí povinně obsahovat moduly:

```
DATA                          ;Globální data
      ;… deklarace globálních dat
DATA_END


MODULE_MAIN
      ;… základní logika, mechanizmy
MODULE_MAIN_END
```

Další moduly jsou nepovinné:

```
DATA_LOCAL                    ;Lokální data
      ;… deklarace lokálních dat
DATA_LOCAL_END


MODULE_INIT                   ;Start PLC
      ;… inicializace dat
MODULE_INIT_END


MODULE_DONE                   ;Stop PLC
      ;… ukončovací operace
MODULE_DONE_END


MODULE_CLEAR                  ;Nulování PLC
      ;… nulování dat
MODULE_CLEAR_END
```

Všechny moduly jsou pokračováním stejných modulů z hlavního souboru PLC programu.

b) Modul **`DATA`** musí být uveden jako první. Všechna data, která jsou definována v libovolném souboru PLC programu včetně hlavního souboru, mají **globální** charakter, což znamená, že jsou přístupna ve všech ostatních souborech.

c) Modul **`DATA_LOCAL`** je nepovinný modul PLC programu pro deklarování **lokálních** proměnných. Data deklarovaná v tomto modulu jsou přístupná jen v souboru PLC programu, kde se modul vyskytuje. Modul `DATA_LOCAL` může být použit v každém souboru s PLC programem i vícekrát a může být přitom vnořen do jiných modulů. Lokální data se používají i pro definování “automatických”

proměnných v rámci rozvoje některých instrukcí jazyka TECHNOL. Data definovaná v tomto modulu jsou v této verzi neviditelná i pro ladící program WINTECHNOL. Když je potřeba pro ladění PLC programu zviditelnit lokální proměnné, dočasně přemístíme modul DATA_LOCAL do těla modulu DATA. Když je modul DATA_LOCAL umístěn uvnitř modulu DATA, který má globální charakter, zviditelní se lokální data také pro WINTECHNOL.

d) Modul **MODULE_MAIN** je pokračováním stejného modulu z předešlých souborů PLC programu. Soubor může obsahovat mechanizmy a může volat mechanizmy, které jsou definovány v jiných souborech PLC programu. Také instrukce MECH_INIT může být použita ve všech souborech, i když tam není mechanizmus definován. V modulech může být použita instrukce DEBUG.

e) Moduly **MODULE_INIT, MODULE_DONE** a **MODULE_CLEAR** jsou v dalších souborech nepovinné. Pokud jsou použity, jsou také pokračováním stejných modulů v předešlých souborech.

f) Všechny soubory mohou obsahovat libovolné definice procedur **PROC_BEGIN** – **PROC_END** a také libovolná volání procedur definovaných v jiných souborech **PROC_CALL**.

g) Ve všech souborech mohou být použity instrukce pro definici časových úseků **DFTM01, DFTM1, DFTM10, DFM100** a mohou být použity v jednom souboru i vícekrát.

7

7. LADĚNÍ PLC PROGRAMU

7.1 Instrukce pro ladění programu

instrukce	DEBUG

funkce	**definice oblasti pro ladění PLC programu**	
syntax1	**DEBUG**	**[ON]**
syntax2	**DEBUG**	**OFF**
parametr	**„ON,OFF"**	**vypnutí a zapnutí ladění**

Instrukce **DEBUG** vymezuje oblast programu, ve které je povoleno trasování a nastavování break-pointů. Instrukce DEBUG nebo DEBUG ON zahájí generaci kódu, ve kterém může být nastaven break-point. Instrukce DEBUG OFF ukončí generaci kódu s možností nastavování break-pointů. Jedná se o softwerový debuger, proto nastavení režimu DEBUG má za následek zpomalení chodu PLC programu (viz dále). Instrukcí DEBUG je umožněno trasování a nastavování break-pointů v PLC programu.

7.2 Ladící program WINTECHNOL

Program Wintechnol komunikuje se systémem přes sériový kanál RS232C. Na systému musí být ladění povoleno v registrech Windows:

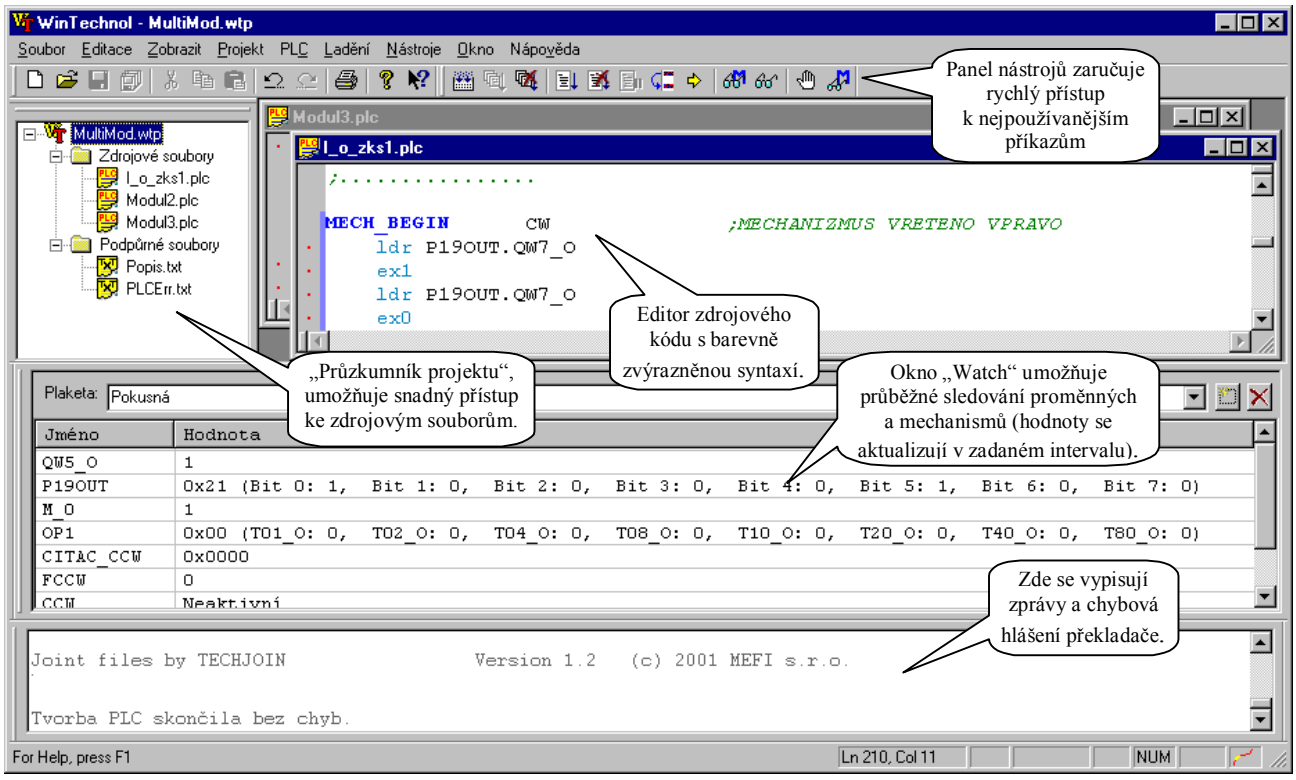
Nastavení se provede ve složce „`HKEY_CURRENT_USER\`"

Povolení ladění:
`\Software\MEFI\WinCNC\Channels\Channel0\Debug\PlcDebug = 1`
(defaultně je ladění zakázáno `PlcDebug = 0`)

Nastavení pro sériový kanál – číslo COM kanálu a rychlost:
`\Software\MEFI\WinCNC\Channels\Channel0\Debug\PlcDebugComNo = 1`
`\Software\MEFI\WinCNC\Channels\Channel0\Debug\PlcDebugComSpeed = 115200`
(defaultní hodnoty jsou: `PlcDebugComNo = 1, PlcDebugComSpeed = 115200`)

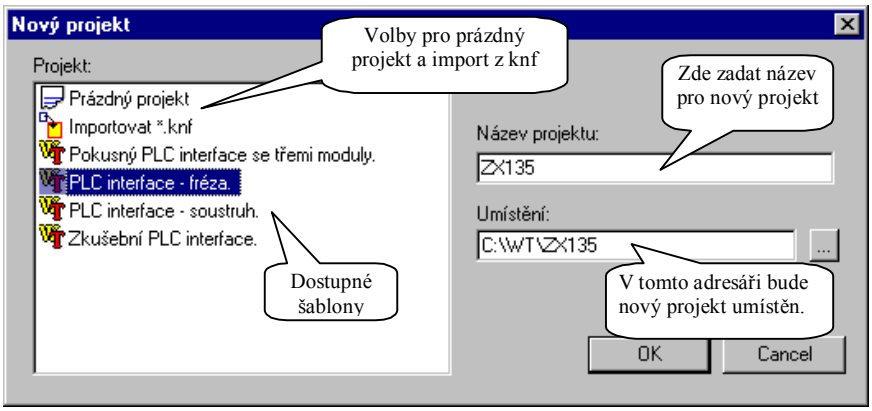
Program WINTECHNOL má vestavěnou interaktivní a kontextovou nápovědu, proto na tomto místě budou uvedeny jen základní informace o programu.

Aplikace WinTechnol tvoří integrované vývojové prostředí pro návrh uživatelského PLC programu pro řídící systémy řady DUAL. Program nabízí správu zdrojových souborů PLC a dalších podpůrných souborů interfacu ve formě projektu. K překladu využívá externě překladač Technol firmy MEFI a turbo assembler z MS-DDK. Ladění interfacu probíhá ve vlastním grafickém rozhraní na libovolném počítači s MS Windows. Tento počítač je třeba propojit s řídícím systémem sériovým kabelem.



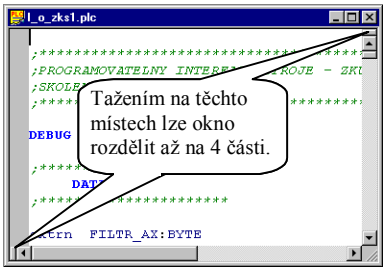
7.2.1 Vytvoření a správa projektu

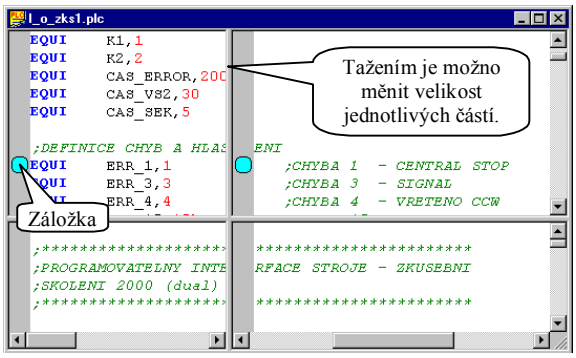
Při vytváření nového interfacu je možné začínat buď úplně od začátku vytvořením prázdného projektu, nebo je možné importovat „projekt“ určený pro starší způsob překladu přímo v prostředí DOSu (import z knf souboru), a nebo lze vyjít z připravené šablony. Jako šablona může posloužit v podstatě libovolný projekt WinTechnolu. Po vytvoření nového projektu se automaticky otevře konfigurační dialog, ve kterém se nastavují vlastnosti projektu (kdykoli později je dialog přístupný přes volbu „Projekt – Nastavení…“).
Při vytváření projektů je vhodné dodržovat zásadu, že pro každý projekt se vytvoří vlastní adresář. V tomto adresáři by se měl nacházet jednak soubor projektu (*.wtp), a jednak všechny zdrojové a doprovodné soubory projektu. V adresáři se také vytvářejí dva podadresáře, standardně pojmenované „Output“ a „Temp“. Do adresáře „Temp“ se ukládají veškeré dočasné soubory, které se využívají při překladu a ladění interfacu, do adresáře „Output“ se ukládá výsledný tvar PLC interfacu, který se pak používá v systému.
Pro snadný přístup ke zdrojovým a podpůrným souborům projektu slouží „Průzkumník projektu“ standardně umístěný v levé části okna aplikace (jeho umístění lze pochopitelně měnit, nebo jej skrýt úplně). Dvojklikem na soubor v „Průzkumníkovi“ se daný soubor otevře pro editaci.

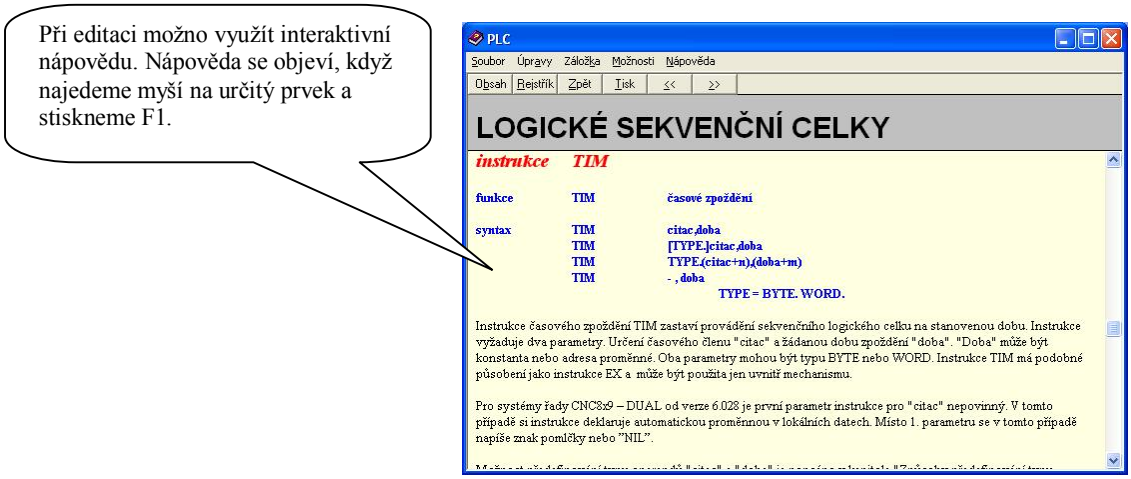


7.2.2 Editor

Aplikace WinTechnol používá vestavěný editor s mnoha funkcemi, které mají za cíl usnadnit práci programátora. K nim patří zejména barevné zvýraznění syntaxe jazyka PLC, možnost otevřít pro jeden soubor více oken současně a každé takové okno rozdělit až na čtyři části. V rámci souboru lze umisťovat záložky, které usnadní pohyb a orientaci v textu. V textu je možno vyhledávat, a to i s využitím regulárních výrazů, nalezený text může být automaticky nahrazován jiným. Pokud bude soubor, který je otevřen vestavěným editorem, změněn mimo tento editor, WinTechnol na tuto skutečnost upozorní a umožní načtení změn z upraveného souboru.





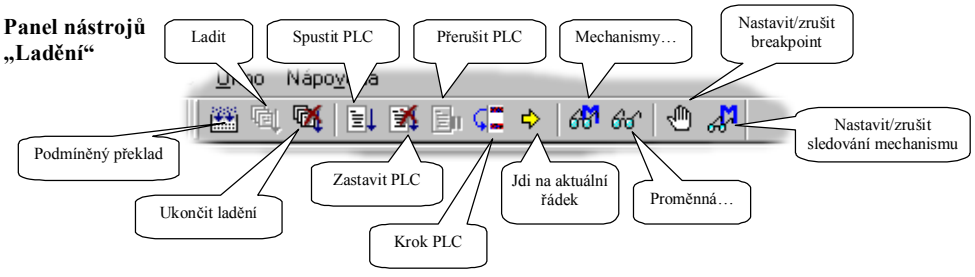


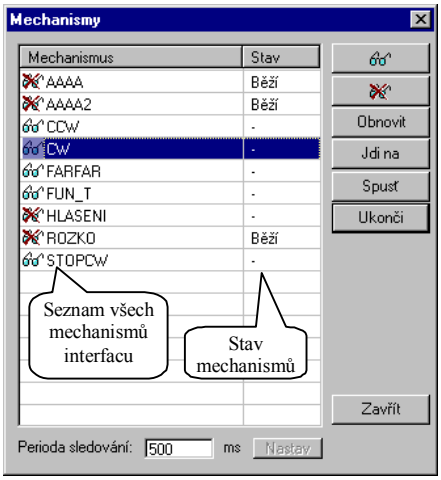
7.2.3 Překlad PLC interfacu

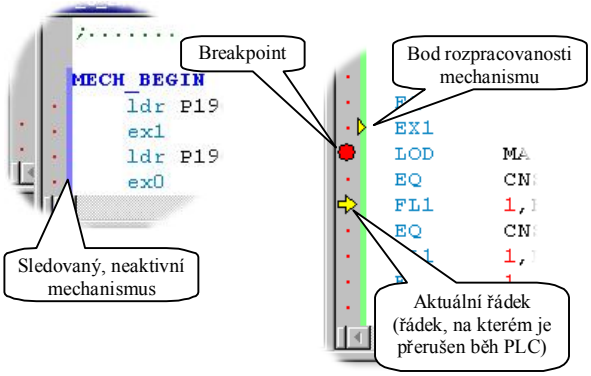
WinTechnol využívá pro překlad PLC externí překladač Technol a překladač assembleru a linker. Kromě toho však umožňuje snadnou volbu verze překladače v závislosti na verzi softwaru systému a snadný přechod k novějším verzím překladače bez nutnosti nové instalace celé aplikace. Umístění a verze překladače se nastavují pro každý projekt zvlášť, čímž je zaručeno, že interface bude vždy přeložen správným překladačem.
Překlad je možno kdykoliv spustit příkazem „Projekt – Překlad“. Větší uplatnění však nalezne spíše příkaz „Podmíněný překlad“, který nejdříve zkontroluje, zda bylo vůbec nastavení pro projekt nebo zdrojové soubory od posledního přeložení modifikovány, a zabrání tak zbytečnému překladu v případech, že ho není třeba. Překlad se též v případě potřeby automaticky spouští před zahájením ladění interfacu příkazem „Ladit“.
Zprávy o průběhu překladu a chybová hlášení jsou průběžně vypisovány do „okna výstupu“, které je standardně umístěno v dolní části okna aplikace. Pokud dojde při překladu k chybám, je možné místo jejich výskytu snadno identifikovat pomocí příkazů „Následující chyba“ a „Předchozí chyba“ v nabídce „Zobrazit“.

7.2.4 Ladění PLC interfacu

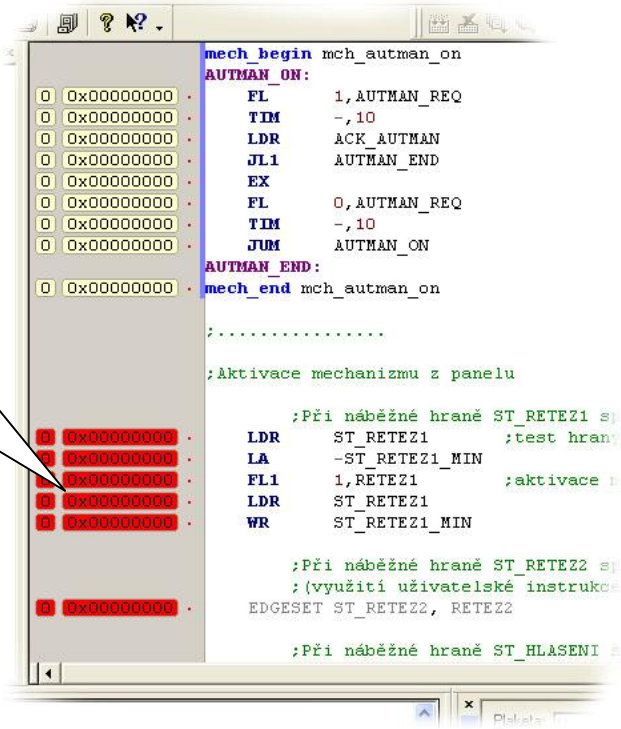
Ladění interfacu se spouští příkazem „Ladit“ (nabídka „Projekt“, nebo panel nástrojů). Vlastnímu ladění předchází překlad interfacu a jeho přenos do systému. Obojí se děje automaticky a pouze tehdy, když je to potřeba, tj. když byl interface od posledního přeložení resp. přenosu do systému. Ladění se ukončuje příkazem „Ukončit ladění“.
Nezávisle na tom, zda je spuštěn ladící režim, je možno (částečně) ovládat běh PLC interfacu, zjišťovat informace o něm a některé další úkony. Po zapnutí ladícího režimu je možné dále krokovat PLC program, nastavovat a rušit breakpointy, sledovat a nastavovat mechanismy a proměnné. Při sledování jsou hodnoty proměnných, resp. stav sledovaných mechanismů, v pravidelných intervalech zasílány ze systému a zobrazovány. Periodu sledování je možno nastavit a to nezávisle pro proměnné a mechanismy. Nejmenší velikost intervalu použitelného intervalu je dána zejména přenosovou kapacitou sériové linky a výkonem počítače. Kromě sledování lze též jednorázově zobrazit hodnotu proměnné či stav mechanismu a také je měnit.

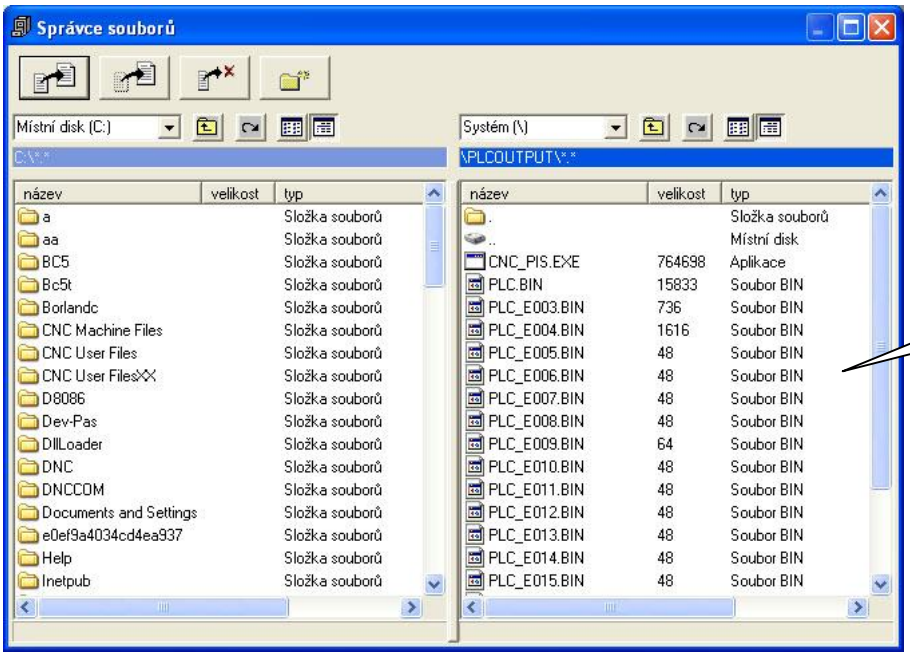






Zobrazení aktuálního stavu datového a bitového registru. Hodnoty jsou platné v okamžiku průchodu daným místem programu. Červená barva označuje místa, kde program v daný okamžik prochází.
Výpis registrů může sloužit na sledování historie.





Správce souborů v programu Wintechnol. Umožňuje libovolně kopírovat, přesouvat a mazat soubory na systému.

7.3 Instalační program pro PLC

Wintechnol může být nakonfigurovaný tak, že při překladu PLC vznikne instalační program, který umožní přenesení PLC (včetně všech doprovodných souborů a konfigurace) na řídící systém. Jméno instalátoru obsahuje informaci o revizi WinCNC, pro kterou je PLC přeloženo a volitelně název a revizi PLC (pokud je použita správa verzí pro PLC, viz níže). Příklad jména instalátoru: `"PLCSetup_for_WinCNC_0_0_222.exe"`.

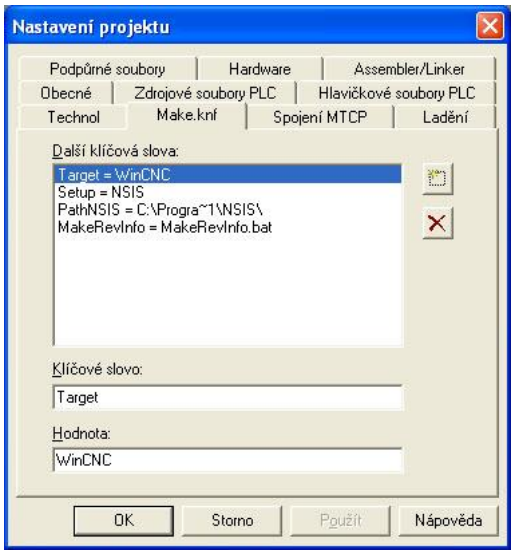
Pro vytváření instalátorů se používá volně šířený software NSIS (Nullsoft Install System), které je možné stáhnout z adresy http://nsis.sourceforge.net. Pro úspěšné vytvoření instalátoru je potřeba, aby Make.knf obsahoval následující klíčová slova (nastavuje se na záložce Make.knf v konfiguraci projektu):

- `Targe = WinCNC`
  Informuje překladač o tom, že se vytváří PLC pro WinCNC. Pro DOS verze softwaru řídícího systému se instalátor nevytváří.

- `Setup = NSIS`
  Udává způsob vytváření instalátoru. Zatím není podporován jiný, než NSIS.

- `PathNSIS = C:\Progra~1\NSIS\`
  Cesta k adresáři, do kterého je nainstalován NSIS. Cesta by měla být v "krátkém" formátu (nesmí obsahovat mezery).

- `MakeRevInfo = MakeRevInfo.bat`
  Vygeneruje soubory s informacemi o revizi (viz dále)



Aby bylo možné instalátor vytvořit, obsahuje projekt soubor „`PLC.NSI`“, ve kterém jsou doplňující instrukce pro vytvoření instalátoru/odinstalátoru. Soubor musí obsahovat následující čtyři funkce:

- `InstallPlcFiles`
  Tato funkce musí obsahovat instrukce pro nainstalování všech podpůrných souborů (automaticky se instalují pouze soubory vlastního PLC).
- `un.InstallPlcFiles`
  Tato funkce musí obsahovat instrukce pro odinstalování všech podpůrných souborů (funkce se spouští při odinstalaci a musí odebrat všechny soubory nainstalované při instalaci z funkce InstallPlcFiles).
- `InstallPlcConfig`
  Tato funkce musí obsahovat instrukce pro nainstalování konfigurace PLC do registru windows. Konfigurace pro PLC by měla být v klíči `"HKEY_LOCAL_MACHINE\"Software\MEFI\WinCNC\Machine"`, volitelně může obsahovat přednastavení uživatelské konfigurace, která se ukládá do klíče `"HKEY_CURRENT_USER\"Software\MEFI\WinCNC"`.
- `un.InstallPlcConfig`
  Tato funkce by měla obsahovat instrukce pro odebrání konfigurace z registru Windows nainstalované funkcí InstallPlcConfig. Vzhledem k tomu, že odinstalátor automaticky odstraní celý klíč `"HKEY_LOCAL_MACHINE\"Software\MEFI\WinCNC\Machine"` s konfigurací pro PLC, může tato funkce často zůstat prázdná.

Bližší dokumentaci k syntaxy skriptů NSIS viz dokumentace NSIS.

7.4 Správa verzí

Software pro správu verzí (krom jiného) umožňuje pohodlný návrat k libovolné z předchozích verzí PLC (vhodné např. při hledání chyb zanesených při úpravách PLC), bezproblémovou spolupráci více vývojářů na jednom PLC a snadné zálohování. Jako software pro správu verzí je předpokládán volně dostupný "Subversion" doplněný o plug-in pro průzkumníka Windows "Tortoise SVN".

Správa verzí pomocí Subversion pracuje na souborové úrovni a pro WinTechnol je zcela transparentní. Jediným styčným bodem je překlad PLC a vytváření instalátoru. Do výsledného PLC a instalátoru je možné vložit informaci o revizi PLC.

Postup pro zprovoznění:

1. Zařadit pro verzování všechny soubory projektu (tj. celý adresář projektu kromě podadresářů Temp a Output), blíže viz. dokumentace pro Subversion a TortoiseSVN.
2. Upravit soubor PlcRevInfo.src. Na jeho základě se bude generovat soubor pro vytváření instalátoru s informacemi o PLC a jeho revizi. Upravit je potřeba pouze informace o PLC tak, aby odpovídaly vytvářenému PLC.
3. Zařadit do procesu překladu PLC generování souborů s informacemi o revizi. To se ve WinTechnolu (do verze 2.0.12 provede tak, že se v konfiguraci projektu na záložce Make.knf přidá klíčové slovo "MakeRevInfo" a jeho hodnota se nastaví na "MakeRevInfo.bat". Hodnotou tohoto klíčového slova je program, který se spouští v průběhu překladu PLC a vygeneruje soubory s informacemi o revizi (v daném případě dávka MakeRevInfo.bat, která vygeneruje příslušný soubor na základě souboru PlcRevInfoSrc.nsh).

Software pro správu verzí je možné získat na stránkách http://subversion.tigris.org a http://www.tortoisesvn.org

9

9. ŘÍZENÍ BINÁRNÍCH VSTUPŮ A VÝSTUPŮ

9.1 CAN-BUS periferie pro binární vstupy a výstupy

element **CANChannel**		**Nastavení kanálu CAN-BUSu**	
	atribut **No**	**číslo kanálu CAN-BUS**	
		1	pro periferie CAN-BUS
		xx	
	atribut **Active**	**je CAN kanál aktivní ?**	
		0	CAN kanál neaktivní (default)
		1	CAN kanál aktivní
	atribut **PhysicalCanChannel**	**číslo fyzického CAN kanálu**	
		1	pro periferie CAN-BUS
		xx	
	atribut **CanSpeed**	**komunikiační rychlost CAN-busového kanálu [bit/s]**	
		1000000	komunikační rychlost 1 MBd (default)
		500000	komunikační rychlost 0.5 MBd
		250000	komunikační rychlost 0.25 MBd
		125000	komunikační rychlost 0.125 MBd
	atribut **ServicePeriod**	**perioda obsluhy CAN-busového kanálu v mikro sekundách**	
		250	perioda obsluhy CAN-BUSu po ¼ ms (default)
		500	perioda obsluhy CAN-BUSu po ½ ms
		1000	perioda obsluhy CAN-BUSu po 1 ms
	atribut **SyncPeriod**	**perioda posílání SYNCu jako násobek základního taktu**	
		1	perioda vysílání SYNC po 1 ms (default)
		1,2,..,15	perioda vysílání SYNC
	atribut **InOut08Cnt**	**Počet periferií INOUT08 připojených k danému kanálu**	
		0	žádná periferie INOUT08 (default)
		1,2,..,32	počet periferií INOUT08 – maximálně 32
	atribut **Kla50Cnt**	**Počet periferií KLA50 (tlačítka panelu) připojených ke kanálu**	
		0	žádná periferie KLA50 (default)
		1,2,3,4	počet periferií KLA50 – maximálně 4
	atribut **KnobCnt**	**Počet točítek**	
		0	(default)
		1,2	počet točítek

Nastavení **NODE-ID** pro systémové CAN-BUS periferie:

skupina (group)		**Pořadové číslo jednotky ve skupině (board)**						
		1	**2**	**3**	**4**	**5**	**6**	**...**
INOUT08	**1**	21h	22h	23h	24h	25h	26h	20h+board
Panel, KLA50,EPU	**2**	41h	42h	43h	44h			40h+board
Točítko	**3**	45h	46h					44h+board
Ruční panel	**4**	47h	48h					46h+board

9.2 Binární vstupy a výstupy u jednotek INOUT08

PLC program snímá vstupy a vysílá výstupy pro externí periferie INOT08 jen pomocí publikovaných (veřejných) bitových polí. PLC program má možnost si bitové pole přepsat do vlastního bitového pole, ve kterém jsou pomocí instrukce DFM definovány vlastní názvy bitových proměnných nebo použít složitější způsoby adresace bitu (viz Kapitola 3, Základní instrukce jazyka - syntax instrukce LDR).

PLC program může řídit vyhodnocování chyb výstupů, může například provést reakci na zkrat sledovaného výstupu apod.

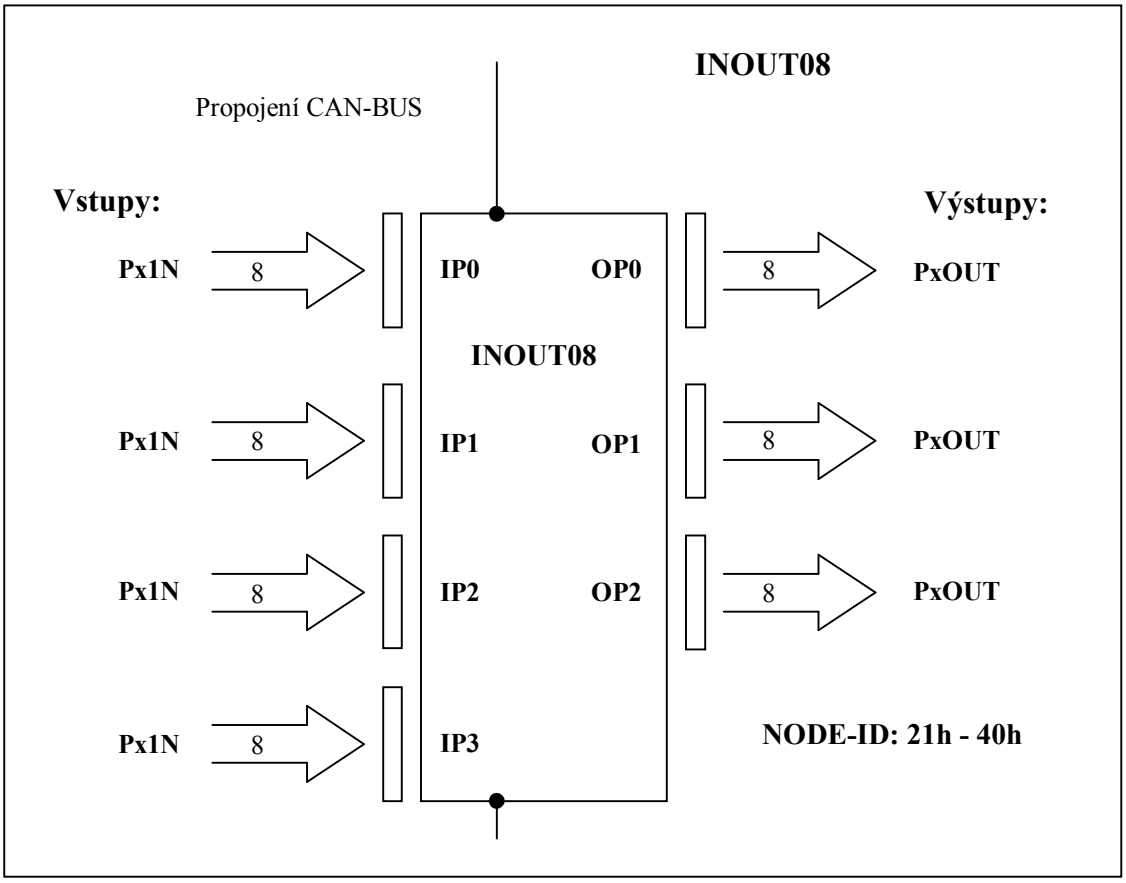
Příklad pro snímání binárních vstupů a vysílání výstupů:

```
;V deklaraci dat
IP0:  DFM   ALFA1, ALFA2, , , ,ALFA6, ,   ;bity z portu P1IN
IP1:  DFM   BETA1, BETA2, , , , , ,       ;bity z portu P7IN
OP0:  DFM   GAMA1, GAMA2, , , , , ,       ;bity na port P4OUT



;V modulu MODULE_INPUT
      LOD   P1IN        ;přepis portu P1IN do buňky IP0
      STO   IP0
      LOD   P7IN        ;přepis portu P7IN do buňky IP1
      STO   IP1

;Na konci modulu MODULE_MAIN
      LOD   OP0         ;přepis buňky  OP0 na port P4OUT
      STO   P4OUT
```

periferie 1.kanál	NODE-ID	vstup-popis	vstup PLC	výstup-popis	Výstup PLC
1. INOUT08 - CAN2	21h	IP0	P1IN	OP0	P0OUT
		IP1	P2IN	OP1	P1OUT
		IP2	P3IN	OP2	P2OUT
		IP3	P4IN		
2. INOUT08 - CAN2	22h	IP0	P5IN	OP0	P3OUT
		IP1	P6IN	OP1	P4OUT
		IP2	P7IN	OP2	P5OUT
		IP3	P8IN		
3. INOUT08 - CAN2	23h	IP0	P9IN	OP0	P6OUT
		IP1	P10IN	OP1	P7OUT
		IP2	P11IN	OP2	P8OUT
		IP3	P12IN		
4. INOUT08 - CAN2	24h	IP0	P13IN	OP0	P9OUT
		IP1	P14IN	OP1	P10OUT
		IP2	P15IN	OP2	P11OUT
		IP3	P16IN		
5. INOUT08 - CAN2	25h	IP0	P17IN	OP0	P12OUT
		IP1	P18IN	OP1	P13OUT
		IP2	P19IN	OP2	P14OUT
		IP3	P20IN		
6. INOUT08 - CAN2	26h	IP0	P21IN	OP0	P15OUT
		IP1	P22IN	OP1	P16OUT
		IP2	P23IN	OP2	P17OUT
		IP3	P24IN		
7. INOUT08 - CAN2	27h	IP0	P25IN	OP0	P18OUT
		IP1	P26IN	OP1	P19OUT
		IP2	P27IN	OP2	P20OUT
		IP3	P28IN		
8. INOUT08 - CAN2	28h	IP0	P29IN	OP0	P21OUT
		IP1	P30IN	OP1	P22OUT
		IP2	P31IN	OP2	P23OUT
		IP3	P32IN		
9. INOUT08 - CAN2	29h	IP0	P33IN	OP0	P24OUT
		IP1	P34IN	OP1	P25OUT
		IP2	P35IN	OP2	P26OUT
		IP3	P36IN		
10. INOUT08 - CAN2	2Ah	IP0	P37IN	OP0	P27OUT
		IP1	P38IN	OP1	P28OUT
		IP2	P39IN	OP2	P29OUT
		IP3	P40IN		
11. INOUT08 - CAN2	2Bh	IP0	P41IN	OP0	P30OUT
		IP1	P42IN	OP1	P31OUT
		IP2	P43IN	OP2	P32OUT
		IP3	P44IN		
12. INOUT08 - CAN2	2Ch	IP0	P45IN	OP0	P33OUT
		IP1	P46IN	OP1	P34OUT
		IP2	P47IN	OP2	P35OUT
		IP3	P48IN		
13. INOUT08 - CAN2	2Dh	IP0	P49IN	OP0	P36OUT
		IP1	P50IN	OP1	P37OUT
		IP2	P51IN	OP2	P38OUT
		IP3	P52IN		

14. INOUT08 - CAN2	2Eh	IP0	P53IN	OP0	P39OUT
		IP1	P54IN	OP1	P40OUT
		IP2	P55IN	OP2	P41OUT
		IP3	P56IN		
15. INOUT08 - CAN2	2Fh	IP0	P57IN	OP0	P42OUT
		IP1	P58IN	OP1	P43OUT
		IP2	P59IN	OP2	P44OUT
		IP3	P60IN		
16. INOUT08 - CAN2	30h	IP0	P61IN	OP0	P45OUT
		IP1	P62IN	OP1	P46OUT
		IP2	P63IN	OP2	P47OUT
		IP3	P64IN		
17. INOUT08 - CAN2	31h	IP0	P65IN	OP0	P48OUT
		IP1	P66IN	OP1	P49OUT
		IP2	P67IN	OP2	P50OUT
		IP3	P68IN		
18. INOUT08 - CAN2	32h	IP0	P69IN	OP0	P51OUT
		IP1	P70IN	OP1	P52OUT
		IP2	P71IN	OP2	P53OUT
		IP3	P72IN		
19. INOUT08 - CAN2	33h	IP0	P73IN	OP0	P54OUT
		IP1	P74IN	OP1	P55OUT
		IP2	P75IN	OP2	P56OUT
		IP3	P76IN		
20. INOUT08 - CAN2	34h	IP0	P77IN	OP0	P57OUT
		IP1	P78IN	OP1	P58OUT
		IP2	P79IN	OP2	P59OUT
		IP3	P80IN		
21. INOUT08 - CAN2	35h	IP0	P81IN	OP0	P60OUT
		IP1	P82IN	OP1	P61OUT
		IP2	P83IN	OP2	P62OUT
		IP3	P84IN		
22. INOUT08 - CAN2	36h	IP0	P85IN	OP0	P63OUT
		IP1	P86IN	OP1	P64OUT
		IP2	P87IN	OP2	P65OUT
		IP3	P88IN		
23. INOUT08 - CAN2	37h	IP0	P89IN	OP0	P66OUT
		IP1	P90IN	OP1	P67OUT
		IP2	P91IN	OP2	P68OUT
		IP3	P92IN		
24. INOUT08 - CAN2	38h	IP0	P93IN	OP0	P69OUT
		IP1	P94IN	OP1	P70OUT
		IP2	P95IN	OP2	P71OUT
		IP3	P96IN		
25. INOUT08 - CAN2	39h	IP0	P97IN	OP0	P72OUT
		IP1	P98IN	OP1	P73OUT
		IP2	P99IN	OP2	P74OUT
		IP3	P100IN		
26. INOUT08 - CAN2	3Ah	IP0	P101IN	OP0	P75OUT
		IP1	P102IN	OP1	P76OUT
		IP2	P103IN	OP2	P77OUT
		IP3	P104IN		
27. INOUT08 - CAN2	3Bh	IP0	P105IN	OP0	P78OUT
		IP1	P106IN	OP1	P79OUT
		IP2	P107IN	OP2	P80OUT
		IP3	P108IN		

28. INOUT08 - CAN2	3Ch	IP0	P109IN	OP0	P81OUT
		IP1	P110IN	OP1	P82OUT
		IP2	P111IN	OP2	P83OUT
		IP3	P112IN		
29. INOUT08 - CAN2	3Dh	IP0	P113IN	OP0	P84OUT
		IP1	P114IN	OP1	P85OUT
		IP2	P115IN	OP2	P86OUT
		IP3	P116IN		
30. INOUT08 - CAN2	3Eh	IP0	P117IN	OP0	P87OUT
		IP1	P118IN	OP1	P88OUT
		IP2	P119IN	OP2	P89OUT
		IP3	P120IN		
31. INOUT08 - CAN2	3Fh	IP0	P121IN	OP0	P90OUT
		IP1	P122IN	OP1	P91OUT
		IP2	P123IN	OP2	P92OUT
		IP3	P124IN		
32. INOUT08 - CAN2	40h	IP0	P125IN	OP0	P93OUT
		IP1	P126IN	OP1	P94OUT
		IP2	P127IN	OP2	P95OUT
		IP3	P128IN		



9.3 Řízení a vyhodnocování chyb externích periferií

Pro každou externí periferii INOUT08 je možno řadit nebo potlačit vyhodnocování chyb pro zobrazování na obrazovce systému, pro informaci PLC programu a pro časovou kontrolu funkčnosti externí periferie.

Každá periferie má přidělenou jednu systémovou strukturu **INOUTS** (viz dále). PLC program má možnost pracovat se systémovými strukturami pomocí instrukcí CAN_SYSIO_READ a CAN_SYSIO_WRITE.

Struktura INOUTS obsahuje také řídicí bitový záznam **CONTROL**

Popis bitů pro záznam CONTROL ve struktuře INOUTS pro vyhodnocování chyb

CONTROL_INOUT	název bitu pro PLC	funkce
bit 0	IO_DIS_ERR	Hodnota log.**0** povoluje a hodnota log.**1** blokuje výpis chyb výstupů na obrazovce systému. Blokování nezpůsobí jiný efekt, takže PLC program si může chybu zjistit. Také dojde k odpojení zkratovaného výstupu. (předvolba řízení po zapnutí systému je v 1.dekádách strojních konstant R330-337)
bit 1	IO_DIS_ERPIS	Hodnota log.**0** povoluje a hodnota log.**1** blokuje zprávu o chybě výstupů pro PLC program. Pokud je v tomto bite nastavena hodnota log.0, systém bude trvale nulovat ty výstupy, které měly chybu (zkrat), až PLC program chybu nepotvrdí. Bitové pole pro PLC program je ERRLI_PxOUT a bode popsáno dále. (Předvolba řízení po zapnutí systému je v 2.dekádách strojních konstant R330-337.)
bit 2	IO_DIS_TMOUT	Hodnota log.**0** povoluje a hodnota log.**1** blokuje časovou kontrolu externí periferie. Blokování způsobí, že systém nezahlásí chybu například při odpojení nebo při ztrátě napájení periferie. Při obnovení činnosti bude periferie normálně funkční. (Předvolba řízení po zapnutí systému je v 3.dekádách strojních konstant R330-337.)
bit 6	IO_DIS_NULL	Hodnota log.**0** povoluje a hodnota log.**1** blokuje nulování všech periferií při vážné chybě této periferie.

CHYBA NÍZKÉ IMPEDANCE (ZKRAT)

Tato chyba je předávána do PLC programu (pokud není pro periferii nastaven bit IO_DIS_ERPIS na hodnotu log.1) v bitovém poli **ERRLI_PxOUT**. Pokud není zvláštní požadavek na ošetření chyb, PLC program nemusí na bity bitového pole ERRLI_PxOUT regovat.

Bitové pole ERRLI_PxOUT má stejné označení s portami PxOUT (například ERRLI_P4OUT je chybové pole k portu P4OUT). Pro orientaci v chybovém poli se s výhodou může použít „složitější adresace bitu“ a přitom se využije označení odpovídajícího bitu z pole PxOUT.

Příklad:

```
OP4:  DFM   ALFA, , ,BETA, , , ,

      LOD   OP4
      STO   P4OUT             ;výstup OP4 s bitem BETA

      LDR   ERRLI_P4OUT.BETA  ;test výstupu BETA na zkrat
      JL1   Error_Beta
```

NEZÁVISLÉ NASTAVOVÁNÍ VSTUPŮ A VÝSTUPŮ

Pro nezávislé nastavování vstupů a výstupů slouží bitová pole **SETL_PxxIN**, **SETH_PxxIN**, **SETL_PxxOUT** a **SETH_PxxOUT:**

Ruční řízení výstupů je nezávislé na nastavované hodnotě výstupu z PLC programu a řízení vstupů je nezávislé na fyzické sejmuté hodnotě vstupu. Najednou je možno ručně ovládat libovolný počet vstupů a výstupů.

bitové pole	význam
SETL_PxxIN	nastavení vstupů pro PLC na hodnotu log.0
SETH_PxxIN	nastavení vstupů pro PLC na hodnotu log.1
SETL_PxxOUT	nastavení výstupů z PLC na hodnotu log.0
SETH_PxxOUT	nastavení výstupů z PLC na hodnotu log.1

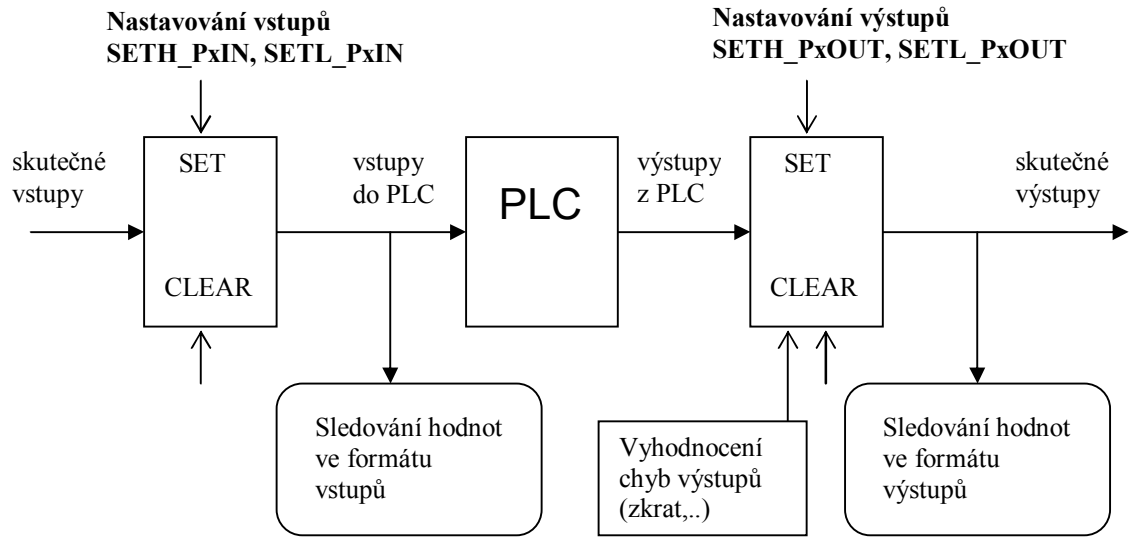
Pro orientaci v bitovém poli se s výhodou může použít „složitější adresace bitu“ a přitom se využije označení odpovídajícího bitu z pole PxOUT.

Nezávislé nastavování vstupů a výstupů se může použít provádět ladícím programem Wintechnol.
Pro zobrazování stavu vstupů a výstupů s ohledem na ruční ovládání slouží standardní formáty pro sledování vstupů, výstupů a matice panelu stroje. Pomocí těchto formátů můžeme získat přehled o všech vstupech a výstupech, které byly přednastaveny pomocí ručního ovládání.

Pokud má zobrazovaný bit políčko podbarveno **zeleně**, je tento bit právě ručně ovládán a jeho hodnota by měla odpovídat našemu ručnímu nastavení. Pokud je bit podbarven zeleně, ale jeho hodnota neodpovídá požadovanému ručnímu nastavení, pravděpodobně není příslušná periferní jednotka nakonfigurovaná (R231). Když se příslušný bit uvolní, zelené podbarvení se musí ztratit.

Pokud má zobrazovaný bit políčko podbarveno **červeně**, vykazuje tento bit právě chybu. Z největší pravděpodobnosti se jedná o chybu zkratu. Pokud chyba nebude potvrzena v příslušném chybovém poli od PLC programu, zůstane bit v chybě až do vypnutí systému.

Blokové schéma pro nezávislé nastavování vstupů a výstupů:



HLÁŠENÍ VÁŽNÝCH CHYB PRO PLC

Každá periferie má přidělenou jednu systémovou strukturu INOUTS (viz dále), v které se nachází také bajtové položky ERROR a ERR_CODE. V buňce ERROR je okamžitý stav „error registru 1001“ z jednotky a v buňce ERR_CODE je paměť posledně vzniknuté chyby.

PLC program má možnost číst struktury INOUTS pomocí instrukce CAN_SYSIO_READ (viz dále), nebo testovat společné bajtové buňky **CAN_IO_ERROR_CODE** a **CAN_IO_ERROR_NUM**:

Skupina 1. (INOUT08)

buňka (BYTE)	hodn.	význam
CAN_IO_ERROR_CODE	0, xx	bitové pole „error registru 1001“
CAN_IO_ERROR_NUM	1, 2,..	Pořadové číslo jednotky INOUT08
CAN_IO_TIMEOUT_CODE	0, 1	1 = time-out pro INOUT08
CAN_IO_TIMEOUT_NUM	1, 2,..	Pořadové číslo jednotky INOUT08

Skupina 2. (KLA50)

buňka (BYTE)	hodn.	význam
CAN_MAT_ERROR_CODE	0, xx	bitové pole „error registru 1001“
CAN_MAT_ERROR_NUM	1, 2,..	Pořadové číslo jednotky KLA50
CAN_MAT_TIMEOUT_CODE	0, 1	1 = time-out pro KLA50
CAN_MAT_TIMEOUT_NUM	1, 2,..	Pořadové číslo jednotky KLA50

Skupina 3. (TOC)

buňka (BYTE)	hodn.	význam
CAN_TOC_ERROR_CODE	0, xx	bitové pole „error registru 1001“
CAN_TOC_ERROR_NUM	1, 2,..	Pořadové číslo jednotky TOC
CAN_TOC_TIMEOUT_CODE	0, 1	1 = time-out pro TOC
CAN_TOC_TIMEOUT_NUM	1, 2,..	Pořadové číslo jednotky TOC

Definice chyb v „error registru“ pro jednotky INOUT08, KLA50 a TOC

bit	označení pro PLC	význam
bit 0.	ERR_IO8_ERROR	Bit je nastaven při chybě vždy
bit 1.	ERR_IO8_SHORT	Chyba výstupů – zkrat *)
bit 2.	ERR_IO8_LOW_SUPPLY	Malé napájecí napětí
bit 3.	ERR_IO8_TEMP_OVER	Překročena teplota
bit 4.	ERR_IO8_COMM	Chyba komunikace (systém zjistí sám, například „time-out“)
bit 5.	-	
bit 6.	-	
bit 7.	ERR_IO8_INPUT	Chyba vstupů

*)
Chyba zkratu výstupů se nepočítá mezi vážné chyby periferie (nenulují se výstupy na všech periferiích). Systém při chybě zkratu hlásí chybu s číslem periferie. Automaticky si vyžádá informaci o zkratu z periferie a nastaví ve struktuře INOUTS bitové pole **SHORT_OUT0**, **SHORT_OUT1** a **SHORT_OUT2.** Na obrazovce systému možno zjistit zkratované výstupy ve formátu sledování výstupů. Výstupy, které jsou ve zkratu budou podbarveny červeně.

Příklad:
Test funkčnosti jednotek INOUT08:

```
LDR   CAN_IO_ERROR_CODE.ERR_IO8_ERROR     ;společný bit pro errory
JL1   ErrorIO
```

9.4 Sdílený přístup pro CAN kanál

Sdílené použití CAN kanálu v PLC programu znamená, že PLC program používá CAN-BUS kanál společně se systémem, nebo alespoň pro jeho řízení používá systémové prostředky. Inicializaci CAN kontroléru a řízení vysílání a příjmu paketů také provádí systém. PLC program jen žádá a zařazení svých paketů do fronty pro vyslání a čte frontu přijmutích paketů.

Masky bitů pro vyhodnocování		
Symbol	**Maska**	**Význam**
CAN_ERR_OK	00h	Vše v pořádku
CAN_ERR_XMTFULL	01h	Plná fronta pro vysílání
CAN_ERR_RCVEMPTY	02h	Nepřijala se nová zpráva

INSTRUKCE PRO PRÁCI S CAN-BUSEM SE SDÍLENÝM PŘÍSTUPEM

Funkce pro sdílenou obsluhu CAN-BUSu jsou do PLC programu implementovány jako 2 speciální instrukce.

Instrukce pro sdílený přístup	**Popis**
CAN_SEND	Zařazení paketu do fronty pro vysílání na periferii
CAN_RECV	Čtení paketu z příjmové fronty

instrukce	**CAN_SEND**

funkce	**Zařazení paketu do výstupní fronty**	
syntax	**CAN_SEND**	**can, pmsg**
1.parametr	**„can"**	**číslo CAN kanálu**
2.parametr	**„pmsg"**	**pointer na paket**

Instrukce slouží pro zařazení paketu do výstupní fronty paketů, které jsou vysílány přes CAN kanál na periferii. Instrukce po vykonání vrátí v datovém registru výsledek, který je složen z masek chyb (popsaných dříve). Pokud při řazení nevznikla chyba, vrácená hodnota bude CAN_ERR_OK (00h). pokud se paket do fronty nepodařilo zařadit (fronta je přeplněna), bude vrácená hodnota CAN_ERR_XMTFULL (01h)

Význam parametrů:

Parametr	Hodnota (příklad)	Význam
can	1, 2	Logický CAN kanál (CAN1, CAN2)
pmsg	CANBUFF_OUT	Pointer, který ukazuje na vyhrazené místo 12 bajtů. Návěští u instrukce STR, kde je místo pro paket

Příklad:

```
CANBUFF_OUT:      STR   12                ;výstupní paket

      CAN_SEND    2, CANBUFF_OUT          ;Zařazení paketu
      EQ          CAN_ERR_OK              ;Bylo zařazeno ?
      JL1         OK                      ;zařazeno
                                          ;nezařazeno
```

Podrobnější příklad bude uveden v závěru této kapitoly

instrukce	**CAN_RECV**

funkce **Čtení paketu z příjmové fronty (Nedoporučuje se používat)**

syntax **CAN_RECV can, pmsg**

1.parametr **„can“ číslo CAN kanálu**
2.parametr **„pmsg“ pointer na paket**

Instrukce je uvedena jen pro kompatibilitu a nedoporučuje se používat. Místo ní je zavedena instrukce **CAN_RECV_REQ**, které popis je dále.

Instrukce slouží pro čtení paketu z fronty paketů, které byly přijaty z periferii přes CAN kanál. Instrukce po vykonání vrátí v datovém registru výsledek, který je složen z masek chyb (popsaných dříve). Pokud při čtení nevznikla chyba, vrácená hodnota bude CAN_ERR_OK (00h). pokud se žádný paket pro PLC nepřijal, bude vrácená hodnota CAN_ERR_RCVEMPTY (02h)

Význam parametrů:

Parametr	Hodnota (příklad)	Význam
can	2	Logický CAN kanál (CAN2)
pmsg	CANBUFF_IN	Pointer, který ukazuje na vyhrazené místo 12 bajtů. Návěští u instrukce STR, kde je místo pro paket

Příklad:

```
CANBUFF_IN:       STR   12                ;vstupní paket

      CAN_RECV    2, CANBUFF_IN           ;Příjem paketu
      EQ          CAN_ERR_OK              ;Bylo přijato ?
      JL1         OK                      ;přijato
                                          ;nic se nepřijalo
```

Podrobnější příklad bude uveden v závěru této kapitoly

instrukce	**CAN_RECV_REQ**

funkce **Požadavek pro volání událostní procedury po přijmu paketu**

syntax **CAN_RECV_REQ can, objl, objh, idl, idh, proc, pmsg**

1.parametr	**„can"**	**číslo CAN kanálu**
2.parametr	**„objl"**	**minimální ID objektu zprávy**
3.parametr	**„objh"**	**maximální ID objektu zprávy**
4.parametr	**„idl"**	**minimální NODE-ID**
5.parametr	**„idh"**	**maximální NODE-ID**
6.parametr	**„proc"**	**název událostní procedury**
7.parametr	**„pmsg"**	**pointer na paket**

Instrukce má návratové hodnoty:

RLO=1	... operace dokončena bez chyb
RLO=0	... operace dokončena, ale požadavek nebyl akceptován
DR (DWRD)	... handle žádosti o volání událostní procedury

Instrukce slouží pro nastavení žádosti o volání událostní procedury v PLC po příjmu určitých „objednaných" paketů z CAN kanálu (nastavení „CALLBACK" volání). Když instrukce proběhne bez chyb (RLO=1) , tak si PLC program může uchovat „handle žádosti" do vlastní proměnné typu DWORD. Tuto proměnnou může PLC program využít v případě, že by později chtěl žádost o volání procedury zrušit.
Instrukce může být použita v modulu MODULE_INIT.

Význam parametrů:

Parametr	Hodnota (příklad)	Význam
can	1, 2	Logický CAN kanál (CAN1, CAN2)
objl	například 580h	Minimální ID objektu zprávy, které příjem se očekává. Jedná se o horní 4 bity z 11-bitové adresy COB-ID (podle normy CanOpen)
objh	-	Maximální ID objektu zprávy, které příjem se očekává. Parametr je nepovinný. Musí se uvést, když je požadován příjem intervalu zpráv. Pokud parametr nebude uveden (bude nahrazen znakem „-") tak se bude očekávat příjem zpráv jednoho objektu.
idl	například 21h	Minimální NODE-ID objektu zprávy, které příjem se očekává. Jedná se o 7 dolních bitů z 11-bitové adresy COB-ID.
idh	-	Maximální NODE-ID objektu zprávy, které příjem se očekává. Parametr je nepovinný. Musí se uvést, když je požadován příjem intervalu zpráv. Pokud parametr nebude uveden (bude nahrazen znakem „-") tak se bude očekávat příjem zpráv jednoho NODE-ID.
proc	například P_ON_CANRECV	Název událostní procedury, která se spustí po příjmu požadovaného paketu. Jedná se o standardní proceduru napsanou v PLC programu. Procedura po zavolání bude mít v DR registru nastaveno číslo logického CAN kanálu (1, 2).
pmsg	například sCANRECV	Pointer, který ukazuje na vyhrazené místo 12 bajtů. Návěští u instrukce STR (kde je místo pro paket) do kterého se naplní celý přijatý paket, který odpovídá požadovanému COB-ID.

Příklad bude uveden později.

Struktura COB-ID (**C**ommunication **Ob**ject **Id**entifier) podle normy CANopen :

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	x	x	x	x	x	**Function (0-0Fh)**				**NODE-ID (0-7Fh)**						

instrukce	**CAN_RECV_DEL**

funkce	**Zrušení požadavku pro volání událostní procedury přijmu paketu**	
syntax	**CAN_RECV_END**	**handle**
1.parametr	**„handle"**	**handle žádosti o volání událostní procedury**

Instrukce má návratové hodnoty:

RLO=1 ... operace dokončena bez chyb
RLO=0 ... operace dokončena, ale požadavek nebyl akceptován

Instrukce slouží pro zrušení žádosti volání událostní procedury v PLC po příjmu paketů z CAN kanálu. Jako parametr instrukce se musí zadat název proměnné typu DWORD, kde je uloženo „handle žádosti", které vrátila instrukce CAN_RECV_REQ.

Příklad:

Žádost o zachytávání odpovědi na SDO pakety (objekt 580h) pro jednotku INOUT08 s NODE-ID=21h.

```
;Buňka DWORD pro uložení Handle
dwHANDLE:   ds    4

;Místo pro přijatý paket:
sCANRECV:   STR   12


;Událostní procedura pro zpracování odpovědi na SDO paket
PROC_BEGIN  P_ON_CANRECV
      LOD   word.sCANRECV.CAN_DATA+1      ;Index zprávy
      EQ    cnst.1008h
      ;....
      ;....
PROC_END    P_ON_CANRECV


;Žádost o volání procedury P_ON_CANRECV
      CAN_RECV_REQ  2, 580h, -, 21h, -, P_ON_CANRECV, sCANRECV
      STO   dwHANDLE


;Zrušení žádosti o volání procedury
      CAN_RECV_END  dwHANDLE
```

PŘÍKLAD PRO VSTUPY A VÝSTUPY NA NESYSTÉMOVOU CAN PERIFERII SE SDÍLENÝM PŘÍSTUPEM

Příklad:
ID adresa periferie je nastavena na 10h.

V deklaraci dat:

```
CANBUFF_IN:       STR 12      ;misto na prijem paketu
CANBUFF_OUT:      STR 12      ;misto pro vysilani paketu
```

V modulu MODULE_MAIN:

```
      CAN_RECV 2,CANBUFF_IN                    ;Cteni paketu
      EQ         CAN_ERR_OK                    ;Je novy paket ?
      JL0        NIC_NEPRIJATO

;Test na prijem paketu ID=10h+1, delka=2, Byte1=Opc=1, Byte2=Input
      LOD        WORD.CANBUFF_IN.CAN_ID        ;Test ID
      ANDB       CNST.7FFh                     ;11 bitove ID
      EQ         CNST.10h+1                    ;Odpoved od ID=10h ?
      JL0        CAN_OTH
      LOD        BYTE.CANBUFF_IN.CAN_LEN       ;Delka
      EQ         CNST.2                        ;Je delka = 2 ?
      JL0        CAN_OTH
      LOD        BYTE.CANBUFF_IN.CAN_DATA+0    ;Opcode
      EQ         CNST.1                        ;Opcode 1 pro vstupy ?
      JL0        CAN_OTH
      LOD        BYTE.CANBUFF_IN.CAN_DATA+1    ;Sejmuti vstupnich dat !
      STO        IP_CAN                        ;Místo pro nova data
CAN_OTH:                                       ;Jiny paket, opakujeme


;**** Obsluha CAN-BUSu ... vysilani vystupu ****
      LOD        CNST.10h                      ;Vyslani na ID=10h
      STO        WORD.CANBUFF_OUT.CAN_ID
      LOD        CNST.0h                       ;Neni "remote request"
      STO        BYTE.CANBUFF_OUT.CAN_RTR
      LOD        CNST.2                        ;delka paketu
      STO        BYTE.CANBUFF_OUT.CAN_LEN
      LOD        CNST.2                        ;Opcode = 2
      STO        BYTE.CANBUFF_OUT.CAN_DATA+0
      LOD        IP0                           ;data pro vyslani
      STO        BYTE.CANBUFF_OUT.CAN_DATA+1

      CAN_SEND  2,CANBUFF_OUT                  ;Zarazeni paketu do fronty
```

9.5 Ovládání systémových CAN-BUS periferií

PLC program má možnost číst a nastavovat struktury pro systémové CAN-BUS periferie, jako jsou INOUT08 a KLA50. Kromě toho zůstává možnost použít instrukce CAN_SEND a CAN_RECV pro sdílený přístup.

Funkce pro přístup k systémovým strukturám CAN-BUS periferií jsou do PLC programu implementovány jako 2 speciální instrukce.

Instrukce pro sdílený přístup	Popis
CAN_SYSIO_READ	Čtení prvku z pole systémových struktur pro CAN-BUS periferie
CAN_SYSIO_WRITE	Zápis prvku do pole systémových struktur pro CAN-BUS periferie

Každá periferie má přidělenou jednu strukturu INOUTS. Prvky struktury je možno sledovat také pomocí diagnostické obrazovky „CAN-BUS peripherals diagnostic“. Pomocí této obrazovky je umožněno také vysílat a přijímat SDO pakety.

```
INOUTS  STRUC
        SIZE            DW      0       ;velkost
        PRESENT         DB      0       ;Je jednotka pritomna ?
        MODE            DB      0       ;mod (0=standard, 1=analog, 2=matice)
        VENDORID        DD      0       ;SDO 1018
        DEVICENAME      DD      0       ;SDO 1008
        HWVERSION       DD      0       ;SDO 1009
        SWVERSION       DD      0       ;SDO 100A

        IN0             DB      0       ;vstupy
        IN1             DB      0
        IN2             DB      0
        IN3             DB      0

        OUT0            DB      0       ;vystupy
        OUT1            DB      0
        OUT2            DB      0
                        DB      0

        ANL0            DB      0       ;analog
        ANL1            DB      0
        ANL2            DB      0
        ANL3            DB      0
        ANL4            DB      0
        ANL5            DB      0
        ANL6            DB      0
        ANL7            DB      0

        ERROR           DB      0       ;error registr 1001 - okamzity
        ERR_CODE        DB      0       ;kod chyby ... zapise pri vzniku chyby

        ERR_STAT        DB      0       ;hlaseni chyb - record E_IOR
                                        ;bit0(IO_ER_HLI_ACT) = chyba (INOUT_ERR_CODE)
                                        ;bit1 (IO_ER_TM_ACT)= time-out
                                        ;bit2 (IO_ER_HLI) = chyba (INOUT_ERR_CODE)
                                        ;bit3 (IO_ER_TMOUT) = time-out pamet
                                        ;bit4 (IO_ER_NULL) = zadost o nuluvani periferii
        CONTROL         DB      0       ;rizeni - record C_IOR
                                        ;bit0 (IO_DIS_ERR) = blokovani vypisu chyb
                                        ;bit1 (IO_DIS_ERPIS) = blokovani chyb pro PLC
                                        ;bit2 (IO_DIS_TMOUT) = blokovani chyb time-out
                                        ;bit6 (IO_DIS_NULL) = blokovani nulovani periferii
        COUNT           DW      0       ;citac pro time-out

        SEND_REQ        DB      0       ;zadost o vyslani SDO paketu INOUT_SEND
        RESP_COUNT      DB      0       ;citac prijmu
        SHORT_SEND_REQ  DB      0       ;zadost o informaci zkratu

        SHORT_OUT0      DB      0       ;informace o zkratu
        SHORT_OUT1      DB      0
```

```
        SHORT_OUT2      DB      0

        SEND            TCANMSGS2 <0>  ;paket pro vyslani
        RESP            TCANMSGS2 <0>  ;response paket

        RPDO_COUNT      DW      0       ;dgn cistac zarazenych RPDO paketu pro vyslani
        TPDO_COUNT      DW      0       ;dgn cistac prijmutych TPDO paketu

        PxIN            DW      0       ;cislo vstupniho PLC portu 1=P1IN
        PxOUT           DW      0       ;cislo vystupniho PLC portu 1=P1OUT

        lpINOUT_PxIN            DW 2 DUP (0)   ;pointry na PLC oblasti
        lpINOUT_PxOUT           DW 2 DUP (0)
        lpINOUT_ERRHI_PxOUT     DW 2 DUP (0)
        lpINOUT_SETH_PxOUT      DW 2 DUP (0)
        lpINOUT_SETH_PxIN       DW 2 DUP (0)

        MODE_ACT        DB      0       ;mod aktual
        MODE_FAZE       DB      0

        TL0             DB      0       ;tlacitka matice
        TL1             DB      0
        TL2             DB      0
        TL3             DB      0

        IRC0            DW      0       ;irc
        IRC1            DW      0
INOUTS  ENDS
```

Definice chyb v „**error registru**“ pro jednotky INOUT08, KLA50 a TOC

bit	označení pro PLC	význam
bit 0.	ERR_IO8_ERROR	Bit je nastaven při chybě vždy
bit 1.	ERR_IO8_SHORT	Chyba výstupů – zkrat *)
bit 2.	ERR_IO8_LOW_SUPPLY	Malé napájecí napětí
bit 3.	ERR_IO8_TEMP_OVER	Překročena teplota
bit 4.	ERR_IO8_COMM	Chyba komunikace (systém zjistí sám, například „time-out“)
bit 5.	-	
bit 6.	-	
bit 7.	ERR_IO8_INPUT	Chyba vstupů

Definice bitů ve „**statusu**“ jednotek INOUT08, KLA50 a TOC:

bit	označení pro PLC	význam
bit 0.	IO_ER_HLI_ACT	Chyba - kód je v error-registru (okamžitý stav)
bit 1.	IO_ER_TM_ACT	Chyba Time-Out (okamžitý stav)
bit 2.	IO_ER_HLI	Chyba - kód je v error-registru (paměť)
bit 3.	IO_ER_TMOUT	Chyba Time-Out (paměť)
bit 4.	IO_ER_NULL	Žádost o nulování všech periferií

Definice bitů v „**control-registru**“ jednotek INOUT08, KLA50 a TOC:

bit	označení pro PLC	význam
bit 0.	IO_DIS_ERR	Blokování výpisu chyb výstupů
bit 1.	IO_DIS_ERPIS	Blokování hlášení chyb pro PLC
bit 2.	IO_DIS_TMOUT	Blokování chyb Time-Out
bit 6.	IO_DIS_NULL	Blokování nulování všech periferií při vážné chybě

instrukce	**CAN_SYSIO_READ**

funkce **Čtení prvku z pole systémových struktur pro CAN-BUS periferie**

syntax **CAN_SYSIO_READ can, group, board, item**

1.parametr	**„can"**	**číslo CAN kanálu**
2.parametr	**„group"**	**skupina systémových CAN-BUS periferií**
3.parametr	**„board"**	**pořadové číslo jednotky ve skupině**
4.parametr	**„item"**	**identifikátor prvku systémové struktury INOUTS**

Instrukce **CAN_SYSIO_READ** slouží pro načtení jednoho prvku z pole struktur INOUTS. Instrukce načte do DR registru bajt, word nebo double-word podle typu zadaného prvku. Pokud se instrukce provede bez chyb (prvek ve struktuře se najde), tak vrací RLO registr nastaven na hodnotu 0. Pokud RLO registr je nastaven na hodnotu 1, potom instrukce skončila s chybou a data v DR registru nejsou platná.

Význam parametrů:

Parametr	**Hodnota (příklad)**	**Význam**
can	2	Logický CAN kanál (CAN2)
group	1,2,3	Skupina CAN-BUS periferií. 1 = jednotky vstupů a výstupů INOUT08 2 = jednotky maticových vstupů KLA50 3 = panýlek s točítkem a displejem CAN-BUS
board	1,2,...,32	Pořadové číslo jednotky ve skupině. Každá skupina má maximálně 32 jednotek. NODE-ID jednotky musí být nastaveno s ohledem na pořadové číslo jednotky a s ohledem na skupinu.
item	(MODE, IN0, ...)	Identifikátor prvku struktury podle definice struktury INOUTS

Příklad:
Načtení analogového napětí z 3.jednotky INOUT08 (NODE-ID = 23h):

```
CAN_SYSIO_READ    2, 1, 3, ANL0     ;1.skupina,3.jednotka,prvek ANL0
JL1               Error
```

instrukce	**CAN_SYSIO_WRITE**

funkce **Zápis do prvku v poli systémových struktur pro CAN-BUS periferie**

syntax **CAN_SYSIO_WRITE can, group, board, item**

1.parametr **„can“ číslo CAN kanálu**
2.parametr **„group“ skupina systémových CAN-BUS periferií**
3.parametr **„board“ pořadové číslo jednotky ve skupině**
4.parametr **„item“ identifikátor prvku systémové struktury INOUTS**

Instrukce **CAN_SYSIO_WRITE** slouží pro zápis jednoho prvku do pole struktur INOUTS. Instrukce zapíše obsah DR registru do struktury jako bajt, word nebo double-word podle typu zadaného prvku. Pokud se instrukce provede bez chyb (prvek ve struktuře se najde), tak vrací RLO registr nastaven na hodnotu 0. Pokud RLO registr je nastaven na hodnotu 1, potom instrukce skončila s chybou a data se do struktury nezapsala.

Význam parametrů:

Parametr	**Hodnota (příklad)**	**Význam**
can	2	Logický CAN kanál (CAN2)
group	1,2,3	Skupina CAN-BUS periferií. 1 = jednotky vstupů a výstupů INOUT08 2 = jednotky maticových vstupů KLA50 3 = panýlek s točítkem a displejem CAN-BUS
board	1,2,...,32	Pořadové číslo jednotky ve skupině. Každá skupina má maximálně 32 jednotek. NODE-ID jednotky musí být nastaveno s ohledem na pořadové číslo jednotky a s ohledem na skupinu.
item	(MODE, IN0, ...)	Identifikátor prvku struktury podle definice struktury INOUTS

Příklad:
Namódování 4.jednotky INOUT08 pro snímání analogových vstupů:

```
LOD              CNST.1           ;mode=1 pro analogové vstupy
CAN_SYSIO_WRITE  2, 1, 4, MODE    ;1.skupina,4.jednotka,prvek MODE
JL1              Error
```

9.6 Přímá komunikace z PLC pomocí SDO paketů

PLC program má možnost pomocí sdíleného použití CAN kanálu, přímo ovládat nebo zjistit libovolné informace z periferií INOUT08. PLC program používá CAN-BUS kanál společně se systémem a pro jeho řízení používá systémové prostředky. PLC program jen žádá a zařazení svých paketů do fronty pro vyslání a čte frontu přijmutích paketů.

Funkce pro sdílenou obsluhu CAN-BUSu jsou (už byly popsány):

Instrukce pro sdílený přístup	Popis
CAN_SEND	Zařazení paketu do fronty pro vysílání na periferii
CAN_RECV	Čtení paketu z příjmové fronty

SDO pakety pro komunikaci s periferiemi jsou odvozeny z normy CAN-OPEN Cia DS301 a Cia DS401.

Použité COB-ID pro PDO pakety:

jednotka	paket	COB-ID
INOUT08	RPDO1	220h + board, 221h,...
	TPDO1	1A0h + board, 1A1h,...
INOUT08 + analog.vstupy:	TPDO2	2A0h + board, 2A1h,...
KLA50	RPDO1	240h + board, 241h,...
	TPDO1	1C0h + board, 1C1h,...
TOC	RPDO1	244h + board, 245h,...
	TPDO1	1C4h + board, 1C5h,...

Komunikovat pomocí SDO paketů je umožněno i pomocí diagnostické obrazovky pro CAN-BUS periferie. Tato komunikace slouží jen pro diagnostické účely.

Dále uvádíme úplný přehled implementovaných SDO paketů u jednotky INOUT08:

```
   INOUT08 software v1.00:
   ----------------------

Implemented SDO's:
------------------

  unsigned long DeviceType[1];      /* 1000h */
  unsigned char ErrorRegister[1];   /* 1001h */
  unsigned char Devname[20];        /* 1008h - Device name          */
  unsigned char HWVer[20];          /* 1009h - Hardware version     */
  unsigned char SWVer[20];          /* 100ah Software version       */
  unsigned long ConsHeartbeat[1];   /* 1016h Consumer Heartbeat t.  */
  unsigned int  HeartbeatTime[1];   /* 1017h Producer Heartbeat t.  */
  unsigned long VendorID[1];        /* 1018h Identity - VendorID    */
  unsigned char ErrBeh[3];          /* 1029h Error behaviour        */
  unsigned long RPDOCommPar[2];     /* 1400h  */
  PDO_MAP_STRUC RPDOMapPar[8];      /* 1600h  */
  unsigned long TPDOCommPar[2];     /* 1800h  */
  unsigned long TPDO2CommPar[2];    /* 1801h  */
  PDO_MAP_STRUC TPDOMapPar[8];      /* 1a00h  */
  PDO_MAP_STRUC TPDO2MapPar[4];     /* 1a01h  */
  unsigned char StopPgm[1];         /* 1f51h   Stop program cmd     */
  unsigned long Filler[1];          /* Only for align               */
  unsigned char RealOutput[4];      /* 2100h  */
  unsigned char ShortedOutput[4];   /* 2101h  */
  unsigned char EnableFast[4];      /* 2102h  */
  unsigned char FastTrap[4];        /* 2103h  */
  unsigned long InpErrCnt[1];       /* 2104h  */
  unsigned int  ADCOut[28];         /* 2105h  */
  unsigned int  TimeCInp[1];        /* 2106h  */
```

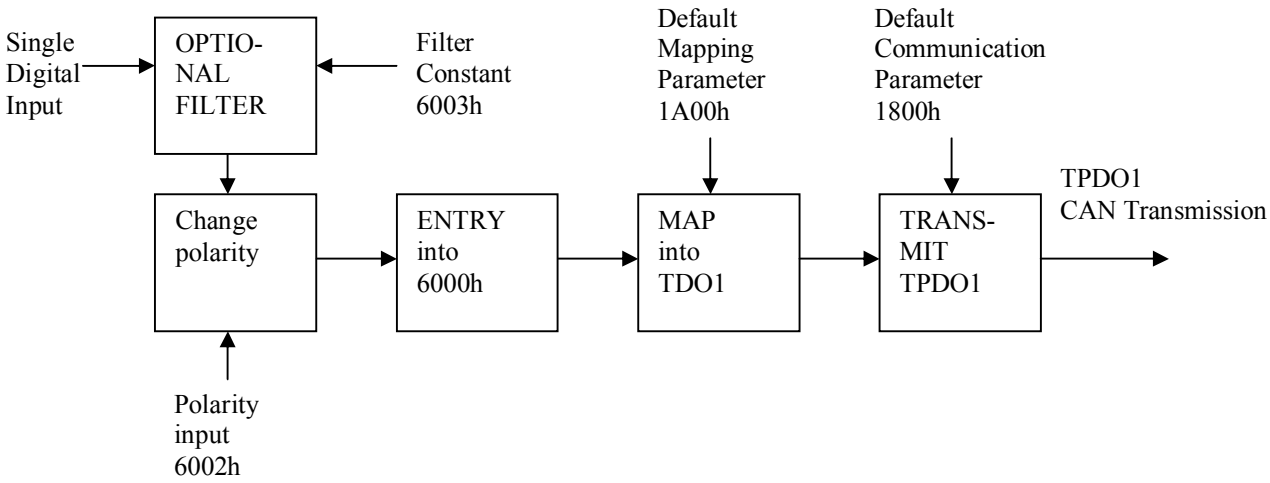
```
  unsigned int  TimeCShort[1];      /* 2107h  */
  unsigned char MatrixInp[4];       /* 2108h  */
  unsigned int  MatOutCnt[1];       /* 2109h max 8 */
  unsigned int  SyncGuard[1];       /* 210ah Max. sync time         */
  unsigned int  CapVoltage[1];      /* 210bh  */
  unsigned int  UnderVoltage[1];    /* 210ch  */
  unsigned char ReadInput[4];       /* 6000h  */
  unsigned char PolarInput[4];      /* 6002h  */
  unsigned char FilterInput[4];     /* 6003h  */
  unsigned long GlobalIntEDig[1];   /* 6005h   - boolean!           */
  unsigned char IntMAnyChInput[4];  /* 6006h  */
  unsigned char IntMLtHInput[4];    /* 6007h  */
  unsigned char IntMHtLInput[4];    /* 6008h  */
  unsigned char WriteOutput[4];     /* 6200h  */
  unsigned char PolarOutput[4];     /* 6202h  */
  unsigned char ErrModeOutput[4];   /* 6206h  */
  unsigned char ErrValueOutput[4];  /* 6207h  */
  unsigned char FilterOutput[4];    /* 6208h  */
  unsigned int  AnalogInput[4];     /* 6401h  */
  unsigned char AnalogIMfSp[8];     /* 6404h  */

  SDO's 1000h - 1f51, 6000h - 6404 are described in
  Cia DS-301 and Cia DS-401. SDO's 2100h - 2109h
  are device specific and are described below:

Device specific SDO's:
----------------------

 RealOutput[4]       2100h -
  Real state on the digital output pins.
  1 - switched on
  0 - switched off
  (Read only)

 ShortedOutput[4]    2101h -
  Digital outputs, whitch are switched off due to
  detected short circuit.
  1 - Output is switched off due to the short circuit.
  0 - No short circuit detected.
  (Read only)

 EnableFast[4];      2102h -
  Enables autonomous fast reaction of the output pin
  to the digital input pin:

   RealOutput = (WriteOutput & ~FastTrap) ^ PolarOutput;
   FastTrap = (FastTrap | ReadInput) & ~WriteOutput;

  1 - Autonomous fast reaction to the input pin enabled
  0 - Autonomous fast reaction disabled
  (Read/Write)

 FastTrap[4];        2103h -
  Input latch for the autnomous output reaction to the
  input pin. Set by ReadInput. Cleared by WriteOutput.
  (Read only)

 InpErrCnt[1];       2104h -
  Count of detected input errors.
  (Read only)

 ADCOut[28];         2105h -
  Raw input from A/D converter.
  Subindex 0 : Length of the array - 28
  Subindex 1..24 : Voltage on the output pins:
    0xffff - Voltage equal to the 24V supply voltage.
  (Read only)

 TimeCInp[1];        2106h -
  Time constant [in uS] for digital input filter.
  Used for the inputs which have set 1 in
  FilterInput (6003h). Minimal time constant is
  approx. 1000uS.
  (Read/Write)

 TimeCShort[1];      2107h -
  Time constant [in uS] for output short circuit
```

```
  detection. If short circuit condition lasts longer
  then this time constant, the output is switched off.
  (Read/Write)

 MatrixInp[4];       2108h -
  Up to 4 switched on inputs conected in matrix:
  Subindex 0 : Length of the array - 4.
  Subindex 1..4 : Switched on inputs.
    0xff - No input switched on.
    Other value - Code of the matrix input switched on.
  For matrix input are used inputs IP0/0..IP0/7 and
  IP1/0..IP1/7, and oututs from OP2/0.
  (Read only)

 MatOutCnt[1];       2109h -
  Count of digital outputs, which are used for the matrix
  input scan. Maximum value = 8.
  0 - No output used for matrix input scan.
  1 - OP2/0 used.
  2 - OP2/0, OP2/1 used.
  ...
  8 - OP2/0 .. OP2/7 used.
  (Read/Write)

 SyncGuard[1];       210ah -
  Checking of the time periode between sync messages.
  0 - No checking of sync messages.
  Other value - maximum alowable time between
      sync messages in 1.024 miliseconds .
  If there is too long time without sync message,
  communication error in the status is set,
  and outputs are set to the state defined by SDO 6207h.
  (Read/Write)

 CapVoltage[1];      210bh -
  Voltage on the supply capacitor in [0.1V].
  Voltage may vary with current consumption of the board.
  (Read only)

 UnderVoltage[1];    210ch -
  If CapVoltage is less then this value, low
  supply voltage error is reported.
  (Read/Write)

 Encoder[3];         210dh -
  Value of encoder counter:
  Subindex 0 : Length of the array - 3.
  Subindex 1 : Encoder counter 1 (unsigned int).
  Subindex 2 : Encoder counter 2 (unsigned int).
  Subindex 3 : Encoder in wheel mode (unsigned int).
  Counter is cleared (0000) at reset.
  Overflows from 0xFFFF to 0000 and
  underflows from 0000 to 0xFFFF.
  (Read only)

 VOvolt[1];          210eh -
  Control of VO output voltage (PWM D/A converter - Display Brightness):
  0xff - approx. 5V
  0x00 - approx. 0V
  (Read/Write)

 Display[19];         210fh -
   Output to the graphic display:
  Subindex 0 : Length of the array - 19.
  Subindex 1 : 1 - initialise, 2 - display test.
  Subindex 2 : Display enable/disable. 0 - disable, 1 enable.
  Subindex 3 : Display RAM start. 0 - 63 (location of 1.displayed byte).
  Subindex 4 : Display back light.           0 - off 1 - on.
  Subindex 5 : reserved
# Subindex 6 : Row - in pixels, from upper left corner, 0-63.
# Subindex 7 : Column - in pixels, from upper left corner, 0-127.
# Subindex 8 : Mask(8 bits): 0 - keep previous pixel state, 1 - write
# Subindex 9 : Inversion(8 bits): 1 - pixel from character generator
                                    inverted.
# Subindex 10 :Text size/Graphic 0 - single size text (6x8 dots/char)
                                 1 - double size text (12x16)
                                 8 - graphic
```

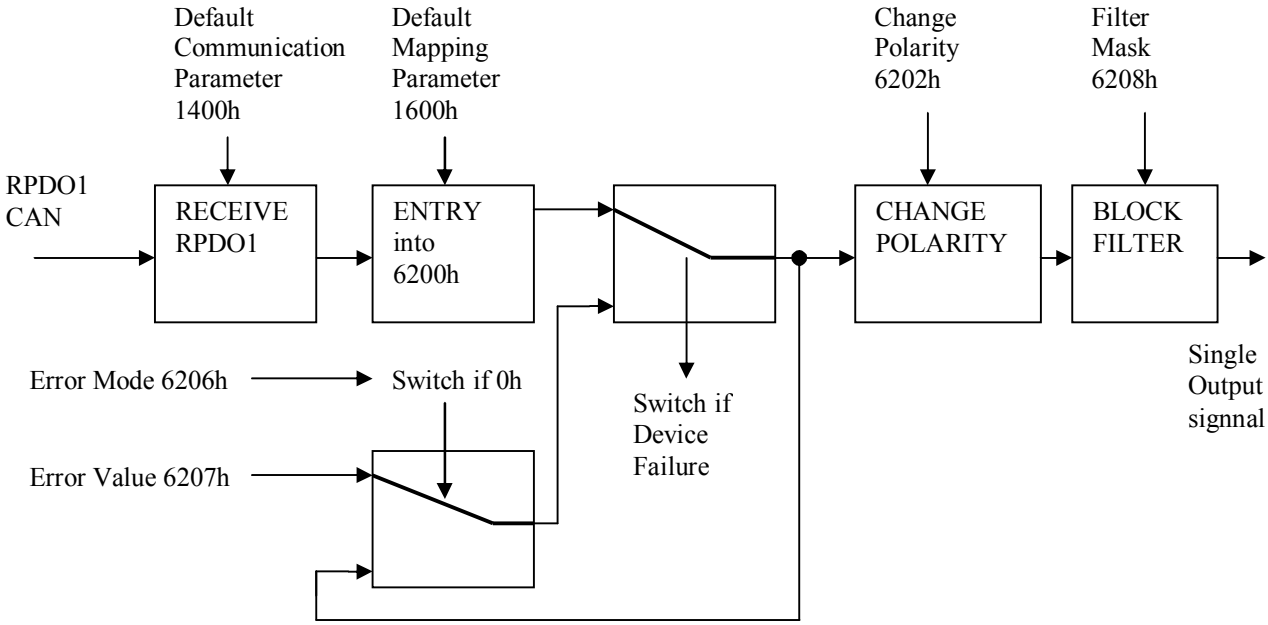
```
# Subindex 11 : Count of bytes (or chars) in subindex 12 - 19 to be written.
                (Not active for SDO transfer - every byte will be written)
# Subindex 12 - 19 : Up to 8 byte data to be written on the display.
               If graphic is selected (subindex 11), bit 0 of the 1.byte
               is written in the position specified at subindex 6 and 7.
               Bits 1-7 are written below it at the same column.
               Next byte is written at the right side.
               If alphanumeric is selected, upper left corner of 1.char is
               written to the position specified at subindex 6 and 7.
               Next char is written to the right side.
               At the SDO transfer, only setting of subindex 12 sets column
               as is at subindex 7. Setting of subindex 13 - 19 writes to
               the next position at the right side.
 #-subindex can be mapped in RPDO packet.
  (Read/Write)

 WheelMode[1];    2110h -
  If nonzero, manual wheel mode is set - display can be initialised and
  wheel mode keyboard at subindex 2111h is active.
  (Read/Write)

 WheelKeyb[2];    2111h -
  Wheel mode keyboard. Every bit represents one key. 1 - Key is pressed.
  Subindex 0 : Length of the array - 2.
  Subindex 1 : Manual wheel keyboard bits 0 - 7.
  Subindex 2 : Manual wheel keyboard bits 8 - 15.
  (Read only)

SDO's inplemented by differrent way than
  described in Cia DS-301 and Cia DS-401:
-----------------------------------------

 HeartBeat parameters (1016h and 1017h):
  Time is specified in 1.024ms interval.
  (Instead of 1ms)
  For Consumer Heartbeat time interval 1 cannot
  be specified. (Imediate error after first Heartbeat).

 PDO communications parameter: (1400h, 1800h, 1801h)
  Suported transmission type (subindex 2) -
  only type 1-240 supported, for TPDO also type 252.
  Default value for transmission type is 1.
  For types 1-240 RTR (remote frame) not supported.

 RPDO, TPDO mapping parameter: (1600h, 1a00h)
  Only fields with 8 bit length supported.
  Count of mapped fields (Subindex 0) fixed to 8.
  Unused fields must be mapped with dummy maping.

 TPDO2 mapping parameter: (1a01h)
  TPDO2 mapping fixed - set to analog inputs
  (Index 6401h Subindex 1 - 2).
  Length of bit fields fixed to 16.
  Count of mapped fields fixed to 2.

 AnalogIMfSp[5];     6404h -
  Up to 4 potentiometers can be connected.
  Subindex 1 - state of the first potentiometer,
  Subindex 2 - state of the second potentiometer....
      0 - rotated to the left end position
   0xff - rotated to the right end position
  Subindex 5 - fixed value 33h.
  (Read only)


SDO's inplemented by differrent way than
  described in Cia DS-301 and Cia DS-401:
-----------------------------------------

 SDO Error Register (1001h):
  Manufacturer specific error (bit 80h)
  represents Input error (set when bad
  connection with the input subboard detected).

 HeartBeat parameters (1016h and 1017h):
  Time is specified in 1.024ms interval.
  (Instead of 1ms)
```

```
  For Consumer Heartbeat time interval 1 cannot
  be specified. (Imediate error after first Heartbeat).

 PDO communications parameter: (1400h, 1800h, 1801h)
  Suported transmission type (subindex 2) -
  only type 1-240 supported, for TPDO also type 252.
  Default value for transmission type is 1.
  For types 1-240 RTR (remote frame) not supported.

 RPDO, TPDO mapping parameter: (1600h, 1a00h)
  Only fields with 8 bit length supported.
  Count of mapped fields (Subindex 0) fixed to 8.
  Unused fields must be mapped with dummy maping.

 TPDO2 mapping parameter: (1a01h)
  TPDO2 mapping fixed - set to analog inputs
  (Index 6401h Subindex 1 - 4).
  Length of bit fields fixed to 16.
  Count of mapped fields fixed to 4.

 AnalogIMfSp[8];     6404h -
  Up to 7 potentiometers can be connected to output
  port OP1: common ref. input to OP1/0,
  rider of the first pot. to OP1/1,...
  rider of the sevnth pot. to OP1/7.
  Subindex 1 - state of the first potentiometer,
  Subindex 2 - state of the second potentiometer....
      0 - rotated to the left end position
    0xff - rotated to the right end position
   (Read only)
```

Blokové schéma pro číslicové vstupy:



Blokové schéma pro číslicové výstupy:



9.7 Popis jednotky INOUT08

NAPÁJENÍ:

Karta INOUT07 je napájena z napájecího napětí vstupů a výstupů. Od systému je galvanicky oddělena na straně komunikační linky - max. 2kV. Vstupy a výstupy nejsou navzájem galvanicky odděleny.

Napájecí napětí:
Jmenovité napájecí napětí vstupů a výstupů: 24V ss.
Maximální napájecí napětí vstupů a výstupů: 36V šš.
Minimální napájecí napětí vstupů a výstupů: 16V min.

VSTUPY:

Všechny vstupy (32 na kartě) mají společný záporný konec spojený se zápornou svorkou napájecího kondensátoru. Napájecí kondensátor je spojen se zápornou svorkou napájení přes diodu, s kladnou přímo.Vstupy systému se chovají jako odpor 10kOhm +-5% proti záporné svorce napájecího kondensátoru desky. (Funkční schéma desky viz výkres INOUT08S.pdf)

Maximální napětí na vstupu: 36V šp.
Minimální napětí na vstupu:-36V šp.

Doba, za kterou se vstupy dostanou do systému na vstup programovatelného automatu: 1.3ms (při cyklu automatu 1ms).
Napětí pro log. 1 : > 50% napájecího napětí +3V (15V pro napájecí napětí 24V).
Napětí pro log. 0: < 33% napájecího napětí (8V pro napájecí napětí 24V)

Toto napětí se mění souhlasně se změnou napájecího napětí desky. Logické úrovně vstupů lze zjistit na panelu systému po stisknutí tlačítka WIN a navolení obrazovky diagnostika externích vstupů. Pokud je napájecí napětí

příliš nízké, karta o tom podá zprávu programovatelnému automatu. Pokud jsou přívody vedeny v prostorech se zvýšeným elektromagnetickým rušením, je třeba použít buď vodiče opatřené stíněním spojeným se záporným pólem napájecího napětí, nebo přídavné zatěžovací odpory nebo kondensátory, připojené mezi vstupní svorku a záporný pól napájecího napětí. Vstupní svorky jsou označeny IP0(0..7),IP1(0..7),IP2(0..7) a IP3(0..7) souhlasně se vstupy automatu.

VÝSTUPY:

Všechny výstupy (24 na kartě) mají společný kladný konec spojený s kladnou svorkou napájecího napětí. Mají jmenovitý výstupní proud 0.1A a jsou chráněné proti zkratu. Při zkratu dojde k vypnutí daného výstupu, a porucha je nahlášena do systému. K obnovení činnosti dojde po uvedení desky.do počátečního stavu – nové spuštění systému nebo nové zapnutí.

Doba, za kterou se výstupy dostanou z programovatelného automatu na výstup: 1.3ms. (při cyklu automatu 1ms).

Doba, za kterou je schopna karta autonomně reagovat na podnět: 0.7ms. (Princip použití autonomního řízení výstupů viz výkres INOUT08F.pdf)

Po ztrátě komunikace se systémem, poruše snímání vstupů na desce nebo poklesu napájecího napětí se všechny výstupy nastaví do stavu, který je dán.parametrem ErrValueOutput - Index 6207h (Při počátečním nastavení vypnuty)

Při přepólování napájecího napětí se karta nepoškodí, avšak všechny výstupy se budou chovat jako sepnuté ke kladné svorce napájecího napětí, a ochrana proti zkratu nebude funkční. Výstupy mohou být zatíženy jmenovitým proudem trvale a všechny. Všechny výstupy mohou být přetíženy až na max. 0.12A po dobu max 1 minuty.Napěťové špičky při spínání indukční zátěže jsou omezeny na hodnotu 58V+-5% mezi výstupem a kladnou svorkou napájecího napětí ochrannými obvody na desce. Unikající proud při vypnutém výstupu a max. napájecím napětí: max. 0.25mA.

Pokles napětí mezi napájecím napětím a zapnutým výstupem: max. 1.5V.

Logické úrovně výstupů a případné vypadnutí zkratové ochrany lze zjistit na panelu systému po stisknutí tlačítka WIN a navolení obrazovky diagnostika externích vstupů.

Výstupní svorky jsou označeny OP0(0..7),OP1(0..7) a OP2(0..7) souhlasně se vstupy automatu.

KOMUNIKACE:

Deska komunikuje s řídícím systémem po sběrnici CAN protokolem CANOPEN rychlostí 1MBaud, 800kBaud, 500kBaud, 250kBaud nebo 125kBaud Rychlost komunikace se nastaví automaticky po spuštění řídícího systému. Než je navázána komunikace s řídícím systémem, deska indikuje blikáním identifikační číslo (deviceID) v hexadecimální soustavě. Pokud má deska základní nastavení (je systému jediná a není třeba identifikační číslo měnit, indikuje číslo 21h – blikne 2x, krátká pauza, blikne 1x, dlouhá pauza. Po úspěšném navázání komunikace s řídícím systémem začne svítit trvale.

Pokud je třeba identifikační číslo změnit, postupujeme takto:
Stiskneme pomocí vhodného nástroje (např. šroubovák) skrz otvory v horních dvou deskách tlačítko na spodní desce po dobu 3s. Bezprostředně poté (do 1s) stiskneme tlačítko krátce tolikrát, kolikátá deska INOUT08 to bude.

Příklad.:
budou li v systému 4 desky a nastavovaná deska má být poslední, stiskneme tlačítko na 3s, krátce uvolníme a rychle stiskneme 4x po sobě. Pokud se to podařilo, bude od té doby deska před zahájením komunikace s řídícím systémem indikovat blikáním číslo 24h - – blikne 2x, krátká pauza, blikne 4x, dlouhá pauza.

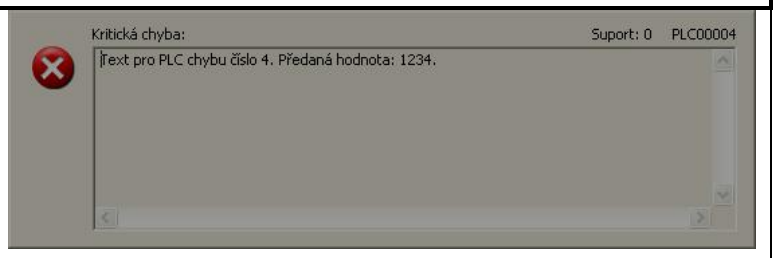
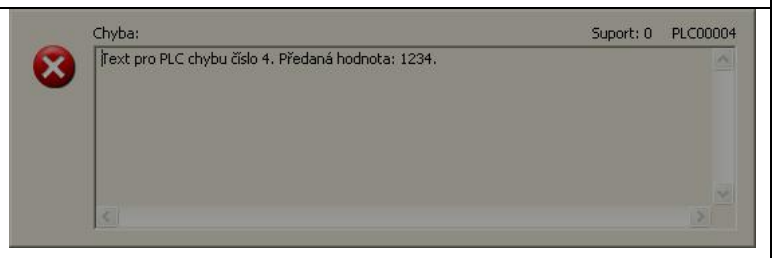
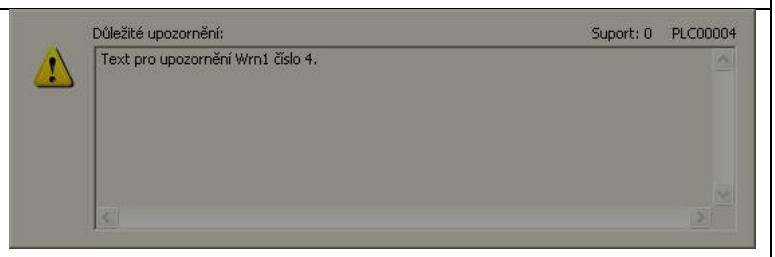
Poslední zařízení připojené ke sběrnici CAN by mělo zakončovat komunikační linku odporem. Pokud je tím posledním zařízením deska INOUT08, lze připojit k lince zakončovací odpor připravený na desce: provede se to tak, že se propojí vodičem svorky T a L komunikačního konektoru CAN. (Příklad připojení viz výkres INOUT08E.pdf)

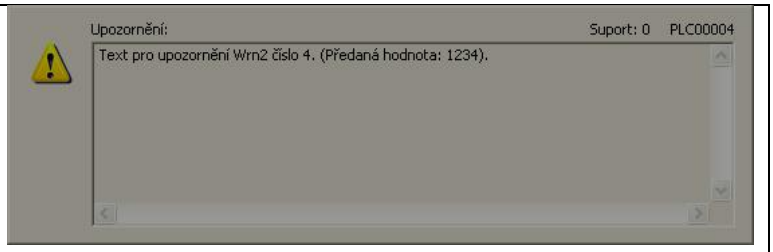
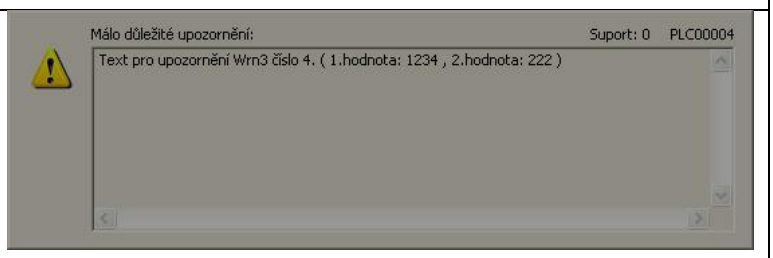
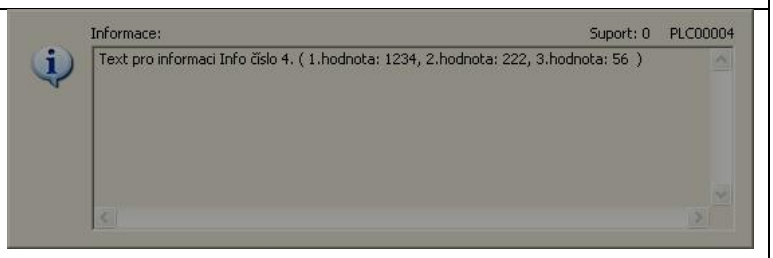
14

14. CHYBOVÁ HLÁŠENÍ A UDÁLOSTI

14.1 Rozdělení hlášení podle typu

PLC program má možnost využít hlášení těchto typů:

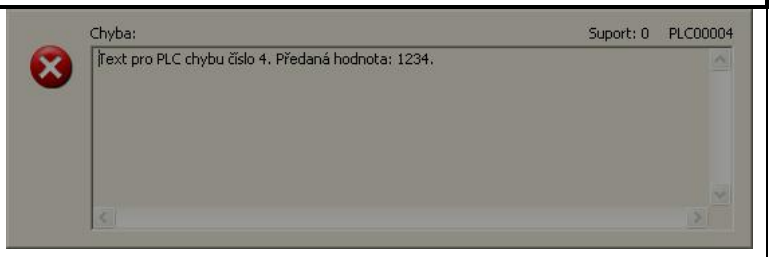
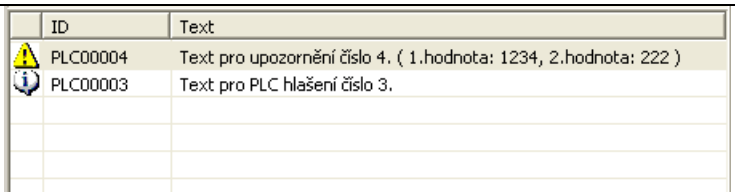
Typ hlášení	`Type`	Příklad zobrazení pro hlášení s přechodnou platností
Kritická chyba Systém nemůže dál pokračovat v činnosti. Zpráva o chybě se zapíše také do záznamu událostí.	`ErrCritical`	
Chyba Vážná chyba, PLC program například vnutí STOP. Zpráva o chybě se zapíše také do záznamu událostí.	`Err`	
Důležité upozornění PLC program hlásí důležité upozornění. Zpráva o se zapíše také do záznamu událostí.	`Wrn1`	

Upozornění PLC program hlásí běžné upozornění pro obsluhu.	**Wrn2**	
Málo důležité upozornění PLC program hlásí málo důležité upozornění pro obsluhu.	**Wrn3**	
Informace PLC program poskytuje informaci pro obsluhu.	**Info**	
Událost PLC program zapíše zprávu do záznamu událostí. Na obrazovce se neobjeví žádné hlášení.	**Event**	

Ikony pro jednotlivé typy hlášení jsou stejné pro hlášení s přechodnou nebo trvalou platností. Zobrazované symboly možno modifikovat a jsou umístěny v podadresáři HTML.

14.2 Rozdělení hlášení podle platnosti

Všechny typy hlášení mohou mít přechodnou nebo trvalou platnost.

Platnost	**Validity**	**Příklad zobrazení**
Přechodná platnost Obsluha musí hlášení potvrdit a tím se hlášení z obrazovky odstraní.	**Transient**	
Trvalá platnost Obsluha hlášení nepotvrzuje. Zobrazení hlášení řídí PLC program.	**Persistent**	

Hlášeních s trvalou platností může být zobrazeno na obrazovce víc najednou. Obsluha si je může prohlížet pomocí rolovacího okna.

Hlášení s přechodnou platností může být v jeden okamžik zobrazeno jen jedno. Systém má zásobník na zprávu PLC hlášení a proto když vznikne víc hlášení z přechodnou platností, tak obsluha postupným potvrzováním se dozví o všech hlášených z PLC programu.

14.3 Hlášení z PLC programu

14.3.1 Jednoprůchodové (událostní) řízení PLC hlášení

Pro řízení všech druhů hlášení z PLC programu slouží instrukce ESET a ESET1. Jedná se o jednoprůchodové (událostní) řízení pro hlášení z PLC programu. To znamená, že hlášení se vygeneruje vždy každým průchodem instrukce ESET. (Stavové řízení pro PLC hlášení je popsáno dále.)

instrukce	**ESET** **ESET1**

funkce **nastavení PLC hlášení**

syntax
```
ESET (ESET1)      [error],[subid],[set],[par1],[par2],[par3]
ESET (ESET1)
ESET (ESET1)      error
```

```
1.parametr  „error"     číslo hlášení
2.parametr  „subid"     podčíslo
3.parametr  „set"       začátek a konec trvalých hlášení
4.parametr  „par1"      1.předávaná hodnota z PLC do výpisu hlášení
5.parametr  „par2"      2.předávaná hodnota z PLC do výpisu hlášení
6.parametr  „par3"      3.předávaná hodnota z PLC do výpisu hlášení
```

Instrukce **ESET** a **ESET1** slouží pro nastavení PLC hlášení. Instrukce ESET provede nastavení hlášení vždy, na rozdíl od instrukce ESET1, která nastaví PLC hlášení podmíněně jen když je obsah registru RLO = 1.

parametr	název	popis
1.parametr	**error**	Číslo PLC hlášení Buňka typu WORD nebo konstanta s číslem PLC hlášení. Parametr je nepovinný, pokud není uveden, převezme se číslo hlášení z DR registru.
2.parametr	**subid**	Dodatkové číslo hlášení.
3.parametr	**set**	Začátek a konec trvalých hlášení (platí jen pro trvalá hlášení). 1 ... PLC hlášení se zobrazí v okně pro trvalá hlášení 0 ... PLC hlášení se zmaže z okna pro trvalá hlášení
4.-6.parametr	**par1** **par2** **par3**	Předávaná hodnota z PLC do výpisu hlášení. Parametry jsou nepovinné. Umožňují předat číselné hodnoty DWORD z PLC do textu hlášení. Způsob a umístění se definuje při textu chyby (viz dále).

Instrukce ESET umožní při každém vzniku chyby zavolat událostní proceduru v PLC programu. Tato procedura musí být definována pomocí klíčových slov PROC_BEGIN a PROC_END, musí mít pevný název **_ON_ESET** a může být umístěna v libovolném souboru s PLC programem. Když v celém PLC programu se procedura s názvem _ON_ESET nevyskytuje, nebude při provádění instrukce ESET žádná procedura zavolána. Procedura dostane při volání naplněný DR registr na číslo chyby.

Systém má implementovaný filtr pro všechny typy hlášení a událostí. Předpis pro filtr pro hlášení a chyby má jméno **ErrorDialog.ErrFilter** a pro události **EventLog.ErrFilter** Soubory jsou umístěny v adresáři Config a jsou v XML tvaru.

Definice textů PLC hlášení je v souboru **Plc0.PlcErrors** v adresáři PLC (viz dále).

Příklady:

```
      LOD   CNST.0012
      ESET                    ;vznikne PLC chyba PLC00012

      EDGE_H ERR_I            ;test nástupní hrany
      ESET1 67,0,1            ;začátek trvalé chyby PLC00067

      EDGE_L ERR_I            ;test spádové hrany
      ESET1 67,0,0            ;konec trvalé chyby PLC00067

      LOD   CNST.123456789
      STO   BunErr            ;Buňka typu DWORD
      ESET  12,-,-,BunErr     ;PLC chyba PLC00012 s předanou hodnotou
```

Příklad:
Definice procedury, která je spuštěna při vzniku chyby

```
PROC_BEGIN  _ON_ESET
            ......
            STO   BUFF
            ......
PROC_END    _ON_ESET
```

Příklad:
Vyhodnocení chyby poklesu tlaku. Je požadavek dokončení bloku, který se jede.

;Hlídání poruchy tlaku
;včleněno do modulu MODULE_MAIN

```
            LDR             TLAK            ;hlídání tlaku
            LOD             59              ;vznik chyby tlaku (59)
            STO1            ERR_STOPPB      ;uchování erroru
                                            ;v přípravných funkcích
            FL1             1,STOPPB        ;stop po bloku
```

;Vyhodnocení méně důležitých chyb
;včleněno na začátek modulu MODULE_BLOCK_INIT

```
ERROR_PO_BLOKU:
            LDR             STOPPB          ;je požadavek na error?
            JL0             PRIPR_E         ;není
            FL              0,STOPPB        ;nulujeme požadavek
            ESET            ERR_STOPPB      ;zápis chyby
            FL              1,STOPPI        ;stop z PLC
            EX
            LDR             CAPI            ;čekáni na CANUL
            EX0
PRIPR_E:
```

14.3.2 Stavové řízení PLC hlášení

Pro řízení všech druhů hlášení z PLC programu může též sloužit instrukce EDEF. Jedná se o stavové řízení pro hlášení z PLC programu. To znamená, že hlášení se vygeneruje jen při změně stavového bitu definovaného v PLC programu, který je s příslušným hlášením svázaný pomocí instrukce EDEF.

instrukce EDEF

funkce **definice pro stavové řízení PLC hlášení**

syntax **EDEF [val],bit,error,[subid],mod,[par1],[par2],[par3]**

1.parametr **„val"** **název bajtu, kde je definován stavový bit**
2.parametr **„bit"** **jméno stavového bitu**
3.parametr **„error"** **číslo hlášení**
4.parametr **„subid"** **dodatkové číslo**
5.parametr **„mod"** **definuje způsob řízení**
6.parametr **„par1"** **1.předávaná hodnota z PLC do výpisu hlášení**
7.parametr **„par2"** **2.předávaná hodnota z PLC do výpisu hlášení**
8.parametr **„par3"** **3.předávaná hodnota z PLC do výpisu hlášení**

parametr	název	popis
1.	**val**	Název Bajtové proměnné, ve které je definován stavový bit (například návěští u instrukce „DFM"). Parametr může mít zadán offset v řetězci (+xx, +BX).
2.	**bit**	Jméno bitové proměnné. (definované pomocí instrukce DFM)
3.	**error**	Číslo PLC hlášení. Buňka typu WORD nebo konstanta s číslem PLC hlášení.
4.	**subid**	Dodatkové číslo hlášení. Buňka typu DWORD nebo konstanta.
5.	**mod**	Způsob řízení pro generaci PLC hlášení. Bity 0,1 …. Podmínky pro začátek hlášení Bity 2,3 …. Podmínky pro konec hlášení Hodnoty bitů: 00b…neúčinné, 01b…nástupní hrana, 10b…sestupná hrana
6. - 8.	**par1** **par2** **par3**	Předávaná hodnota z PLC do výpisu hlášení. Parametry jsou nepovinné. Umožňují předat číselné hodnoty DWORD z PLC do textu hlášení. Způsob a umístění se definuje při textu chyby (viz dále). Parametry se předávají odkazem (ne hodnotou).

Instrukce **EDEF** slouží pro definici stavového řízení PLC hlášení. Instrukce musí být použita jen jednorázově v modulech „**MODULE_INIT**" PLC programu, protože se nejedná o skutečné vysílání PLC hlášení, ale jen o definici stavových bitů, které budou řízení chyb obhospodařovat pomocí systémových prostředků v reálném čase. Instrukce vlastně definuje vazbu mezi PLC hlášením a příslušným bitem definovaným v PLC programu, který bude sloužit jako stavový bit pro generaci hlášení. Dále instrukce pomocí 5. parametru „mod" určí způsob generace PLC hlášení.

Jako stavové bity pro generaci PLC hlášení mohou být použity i mapované vstupy pomocí instrukce MAP_IN. V tomto případě se 1.parametr „val" v instrukci EDEF nezadává.

Příklady pro nastavení způsobu řízení (5.parametr „mod“):

0001b ….. Vysvítí chybu typu „Transient“ při nástupní hraně stavového bitu
0010b ….. Vysvítí chybu typu „Transient“ při sestupné hraně stavového bitu
1001b ….. Vysvítí chybu typu „Persistent“ při nástupní hraně a smaže ji při sestupné hraně

Příklad:

```
;V deklaraci dat:
bErr0:      DFM   ,,eErr2, eErr3,,,,      ;stavové bity pro hlášení
dwBunErr:   DS    4                       ;předávaná hodnota


;V inicializačním modulu
MODULE_INIT
...
  EDEF      bErr0, eErr2, 1003, -, 0001b             ;Error 1003 Transient
  EDEF      bErr0, eErr3, 1004, -, 1001b, dwBunErr   ;Error 1004 Persistent
...
MODULE_INIT_END
```

Příklad:

```
;Příklad pro vysvícení chyby číslo 1234 typu „Transient“ přímo od nástupné
hrany vstupu inPressErr:


;V inicializačním modulu
MODULE_INIT
...
  MAP_IN  inPressErr,'inPressErr'         ;Mapovaný vstup
  EDEF    -, inPressErr, 1234, 0, 0001b   ;Error 1234 od mapovaného vstupu
...
MODULE_INIT_END
```

14.4 Definice textů a vlastností PLC hlášení

Definice textů a vlastností PLC hlášení je v souboru **`Plc0.PlcErrors`** v adresáři PLC v XML tvaru.

Na tomto místě se nebudeme detailně zabývat přesnou definicí syntaxe XML tvaru, ale na příkladu si ukážeme jak takový soubor vypadá. Pro praktické použití to bude postačovat.

Jeden text je definován pomocí elementu `PlcError` a vnořeného elementu `Text`.

element **`PlcError`**		**Definice PLC hlášení**	
	atribut **`No`**	**Číslo PLC hlášení**	
		`xx`	Číslo PLC hlášení nebo události
	atribut **`Type`**	**Typ PLC hlášení**	
		`ErrCritical`	Kritická chyba
		`Err`	Chyba
		`Wrn1`	Důležité upozornění
		`Wrn2`	Upozornění
		`Wrn3`	Málo důležité upozornění
		`Info`	Informace
		`Event`	Událost
	atribut **`Validity`**	**Platnost PLC hlášení**	
		`Transient`	Přechodná platnost PLC hlášení
		`Persistent`	Trvalá platnost /PLC hlášení
	element **`Text`**	**Text PLC hlášení** Text PLC hlášení nebo události. Text může obsahovat maximálně 3 předávané číselné hodnoty z PLC programu: %d .... dekadické zobrazení čísla %x .... hexadecimální zobrazení čísla	

Příklady:

Přechodná (potvrzovací) chyba č.2

```
<PlcError No="2" Type="Err" Validity="Transient">
      <Text>
            Text pro PLC chybu číslo 2.
      </Text>
</PlcError>
```

Trvalá informace č.3

```
<PlcError No="3" Type="Info" Validity="Persistent">
      <Text>
            Text pro PLC hlašení číslo 3.
      </Text>
</PlcError>
```

Přechodná (potvrzovací) chyba č.4, která má dvě předávané hodnoty z PLC

```
<PlcError No="4" Type="Err" Validity="Transient">
      <Text>
            Text pro chybu číslo 4. (1.předaná hodnota: %d, 2.hodnota %d)
      </Text>
</PlcError>
```

Přechodná chyba č.12

```
<PlcError No="12" Type="Err" Validity="Transient"> <!-- ERR_KONSTANTY -->
      <Text>
            Chyba načtení PLC konstant - konstantu se nepodařilo načíst.
      </Text>
</PlcError>
```

14.5 Přerušena komunikace se sekundárním procesorem

Jedná se o nejvážnější chybu systému. Sekundární procesor, na kterém běží reálný čas „RTM“ se musel z vážných důvodů zastavit (přešel do HALTu). V tomto případě se většinou zaznamená a zobrazí v textu chybového hlášení tzv. „Halt-status“, podle kterého je možno pátrat po příčinách HALTu. Pokud se jedná o systémovou chybu, je potřeba informovat výrobce a zaslat celý popis chyby a nejlépe také záznam události systému. Pokud se jedná o selhání PLC programu, je možné využít informace z „Halt-statusu“ pro nalezení místa Haltu.

Na obrazovce se vysvítí chyba:

RT0096 Přerušena komunikace se sekundárním procesorem
Halt status: `<HALT>`
prog.counter EIP: `<EIP>`
selectors DS, CS: `<DS>, <CS>`

Halt status	
30	Neznámy interrupt
13	Obecná chyba ochrany (error protected mode)
2	Dělení nulou, nebo přetečení dělení (platí pro celé čísla)
3	Chyba koprocesoru, chyba při operacích s reálnými čísly
10	Dvojité přerušení výpočtového rastru 1ms
4	Dvojité vnoření výpočtového rastru 1ms
1	Časová hlídání BSP procesoru (procesoru Windows)
6	PLC nebo systém si vyžádal Halt sekundárního procesoru
7	Příliš velká změna přírůstku dráhy (chyba interpolátoru apod.)
11	Neočekávaný interrupt od jednotky souřadnic SU05
14	Chybí interrupt od jednotky souřadnic SU05
16	Interrupt od RTX pro interní časovač APIC
17	Interrupt od APIC pro RTX

Výpis některých selektorů

Selektor		
008h	CODE1	Instrukční segment
010h	DATA_SEC	Tabulky GDT
018h		Segment 4GB
020h		Stack segment
028h		Tabulky IGT
030h	DATA_COM	Datová komunikační oblast DCOM
068h	DATA1	Datový segment
078h	CODE2	PLC systém
080h	CODE4	PLC uživatelský 1.segment (Main)
088h	CODE5	PLC uživatelský 2.segment
0A0h	DATA_USR	Datový segment pomocný
0C8h	CODE_DEBUG	Instrukční segment
0D0h	CODE_UNIT03	PLC uživatelský 2. soubor
0D8h	CODE_UNIT04	PLC uživatelský 3. soubor
0E0h	CODE_UNIT05	PLC uživatelský 4. soubor
0E8h	CODE_UNIT06	PLC uživatelský 5. soubor
0F0h	CODE_UNIT07	PLC uživatelský 6. soubor
0F8h	CODE_UNIT08	PLC uživatelský 7. soubor

100h	CODE_UNIT09	PLC uživatelský 8. soubor
140h	_TEXT32	FLAT model data, 32 bitů
148h	_TEXT32	FLAT model kód, 32 bitů
150h	_TEXT32	FLAT model stack, 32 bitů
168h	DATA_PLC	Lokální a automatické proměnné PLC
170h	CODE_UNIT10	PLC uživatelský 9. soubor
178h	CODE_UNIT11	PLC uživatelský 10. soubor
180h	CODE_UNIT12	PLC uživatelský 11. soubor
188h	CODE_UNIT13	PLC uživatelský 12. soubor
190h	CODE_UNIT14	PLC uživatelský 13. soubor
198h	CODE_UNIT15	PLC uživatelský 14. soubor
1A0h	CODE_UNIT16	PLC uživatelský 15. soubor
1A8h	CODE_UNIT17	PLC uživatelský 16. soubor
1F0h	CODE_INT	Parabolický interpolátor
258h	CAN_TXT	Instrukční segment pro CAN-BUS
268h		Instrukční segment pro CAN-BUS
278h		Instrukční segment pro CAN-BUS
290h		Buffer bloků NCP programu
2A0h	CAN_TXT	Instrukční segment pro CAN-BUS
2B0h	DATA_USR2	Datový segment pomocný
2B8h	DATA_USR3	Datový segment pomocný
2C0h	PLCCONST	Segment pro PLC konstanty
2C8h	PLCCNF	Segment pro PLC konfiguraci
2E0h	SYSVIEW_TXT	Segment pro systémové sdílení
2E8h	PLCVIEW_TXT	Segment pro PLC sdílení
2F0h	SYSNAME_TXT	Segment pro systémové sdílení - jména
2F8h	PLCNAME_TXT	Segment pro PLC sdílení - jména
300h		FLAT model data pro RTMDLL, 32 bitů
308h		FLAT model kód pro RTMDLL, 32 bitů
320h		Datový segment pro sdílenou paměť SA – více suportů
328h		Datový segment pro dávková měření
330h	DATA_USR4	Datový segment pomocný
338h	CODE_UNIT18	PLC uživatelský 17. soubor
340h	CODE_UNIT19	PLC uživatelský 18. soubor
348h	CODE_UNIT20	PLC uživatelský 19. soubor
350h	CODE_UNIT21	PLC uživatelský 20. soubor
358h	CODE_UNIT22	PLC uživatelský 21. soubor
360h	CODE_UNIT23	PLC uživatelský 22. soubor
368h	CODE_UNIT24	PLC uživatelský 23. soubor
370h	CODE_UNIT25	PLC uživatelský 24. soubor
378h	CODE_UNIT26	PLC uživatelský 25. soubor
380h	CODE_UNIT27	PLC uživatelský 26. soubor
388h	CODE_UNIT28	PLC uživatelský 27. soubor
390h	CODE_UNIT29	PLC uživatelský 28. soubor
398h	CODE_UNIT30	PLC uživatelský 29. soubor
3A0h	CODE_UNIT31	PLC uživatelský 30. soubor
3A8h	CODE_UNIT32	PLC uživatelský 31. soubor
430h	CODE_UTIL	Instrukční segment pro utility
458h	CODE_UNIT33	PLC uživatelský 32. soubor
460h	CODE_UNIT34	PLC uživatelský 33. soubor
468h	CODE_UNIT35	PLC uživatelský 34. soubor
470h	CODE_UNIT36	PLC uživatelský 35. soubor
478h	CODE_UNIT37	PLC uživatelský 36. soubor
480h	CODE_UNIT38	PLC uživatelský 37. soubor
488h	CODE_UNIT39	PLC uživatelský 38. soubor
490h	CODE_UNIT40	PLC uživatelský 39. soubor
498h	CODE_UNIT41	PLC uživatelský 40. soubor
4A0h	CODE_UNIT42	PLC uživatelský 41. soubor

4A8h	CODE_UNIT43	PLC uživatelský 42. soubor
4B0h	CODE_UNIT44	PLC uživatelský 43. soubor
4B8h	CODE_UNIT45	PLC uživatelský 44. soubor
4C0h	CODE_UNIT46	PLC uživatelský 45. soubor
4C8h	CODE_UNIT47	PLC uživatelský 46. soubor
4D0h	CODE_UNIT48	PLC uživatelský 47. soubor
4D8h	CODE_UNIT49	PLC uživatelský 48. soubor
4E0h	CODE_UNIT50	PLC uživatelský 49. soubor
4E8h	CODE_UNIT51	PLC uživatelský 50. soubor
4F0h	CODE_UNIT52	PLC uživatelský 51. soubor
4F8h	CODE_UNIT53	PLC uživatelský 52. soubor
500h	CODE_UNIT54	PLC uživatelský 53. soubor
508h	CODE_UNIT55	PLC uživatelský 54. soubor
510h	CODE_UNIT56	PLC uživatelský 55. soubor
518h	CODE_UNIT57	PLC uživatelský 56. soubor
520h	CODE_UNIT58	PLC uživatelský 57. soubor
528h	CODE_UNIT59	PLC uživatelský 58. soubor
530h	CODE_UNIT60	PLC uživatelský 59. soubor
538h	CODE_UNIT61	PLC uživatelský 60. soubor
540h	CODE_UNIT62	PLC uživatelský 61. soubor
548h	CODE_UNIT63	PLC uživatelský 62. soubor
550h	CODE_UNIT64	PLC uživatelský 63. soubor

10

10. MAPOVÁNÍ BINÁRNÍCH A ANALOGOVÝCH VSTUPŮ A VÝSTUPŮ

10.1 Princip mapování

Mapování vstupů a výstupů umožňuje přiřazovat fyzické vstupy a výstupy pro periferie MEFI k PLC programu jen na základě konfigurace a bez nutnosti změny PLC programu. PLC program může mít nadefinován velké množství vstupů a výstupů ale jen některé z nich se propojí s fyzickými vstupy a výstupy podle konfigurace na danou specifikaci stroje.

Důležité upozornění:
V současné době existují prostředky pro logické propojování obecných vstupů a výstupů různých modulů (včetně PLC), které tento způsob mapování úplně nahrazují a poskytují další možnosti logického propojování. Proto doporučujeme při nové tvorbě PLC přejít na návod: **„Logické vstupy a výstupy modulů systému.“** Nové možnosti logického mapování se neomezují jen na CAN-BUS periferie MEFI, ale umožní například propojení PLC - EtherCAT apod. Vzhledem k zachování kompatibility musí ale zůstat plně funkční všechny dále uvedené postupy.

V PLC programu všechny nadefinované binární a analogové vstupy a výstupy budeme nazývat přívlastkem „**logické**“. Namapováním podle konfigurace tak dochází k propojení některých logických vstupů a výstupů s „**fyzickými**“ vstupy a výstupy.

Při konfiguraci jednoho vstupu nebo výstupu se vyplňují takové parametry, jako je textový identifikátor, číslo fyzického portu a váha bitu s kterým je logický vstup propojen (mapování), jestli je skutečně propojen, jestli je invertován a defaultní hodnota, pokud není propojen.

Na jeden fyzický výstup může být namapováno i vícero logických výstupů. (Výsledná hodnota je logické OR, pokud je použita přímá logika.)

Pokud logické vstupy a výstupy nejsou propojeny s fyzickými, uplatní se z konfigurace předvolba nastavení bitu (defaultní hodnota).

Všechny logické vstupy a výstupy mají automaticky nastaveno „**přímé sdílení**“ a tak se značně zjednoduší přístup uživatelských obrazovek a dialogů k jejím hodnotám. Na logické vstupy a výstupy není potřeba používat instrukce `SHARE_VAR` a `SHARE_BIT`.

10.2 Konfigurace pro binární vstupy a výstupy

Konfigurace pro binární vstupy a výstupy je v souboru typu „ChannelConfig“ v elementu „Plc“.

Konfigurace pro binární vstupy:

element **Plc**			**Konfigurace pro PLC**	
	element **Input**		**Konfigurace pro jeden binární vstup.**	
		atribut **ID**	**Identifikátor vstupu**	
			Abcd	Textový řetězec, který slouží jako identifikátor vstupu
		atribut **Port**	**Číslo fyzického portu**	
			0	číslo fyzického vstupního portu (default)
			0-128	číslo fyzického vstupního portu
		atribut **Bit**	**Číslo bitu na fyzickém portu**	
			0	číslo bitu (default)
			0-7	číslo fyzického bitu, na který je logický bit propojen
		atribut **Invert**	**Inverze**	
			0	fyzický vstup je při přímo propojen s logickým (default)
			1	fyzický vstup je při propojení na logický vstup invertován
		atribut **Default**	**Přednastavena hodnota vstupu**	
			0	Přednastavena hodnota v případě, že vstup není propojen
			1	Přednastavena hodnota v případě, že vstup není propojen
		atribut **Connected**	**Propojení logického a fyzického vstupu**	
			0	logický vstup není propojen s fyzickým vstupem (default)
			1	logický vstup je propojen s fyzickým vstupem

Příklad:

Příklad mapovaní dvou vstupů `inLimX` a `inRefX`:

```
<Plc>
  <!-- Vstupy  -->
  <Input ID="inLimX" Port="1" Bit="2" Invert="0" Default="0" Connected="1">
  </Input>
  <Input ID="inRefX" Port="1" Bit="3" Invert="1" Default="0" Connected="0">
  </Input>
</Plc>
```

Konfigurace pro binární výstupy:

element **Plc**			**Konfigurace pro PLC**	
	element **Output**		**Konfigurace pro jeden binární výstup.**	
		atribut **ID**	**Identifikátor výstupu**	
			Abcd	Textový řetězec, který slouží jako identifikátor výstupu
		atribut **Port**	**Číslo fyzického portu**	
			0	číslo fyzického výstupního portu (default)
			0-128	číslo fyzického výstupního portu
		atribut **Bit**	**Číslo bitu na fyzickém portu**	
			0	číslo bitu (default)
			0-7	číslo fyzického bitu, na který je logický bit propojen
		atribut **Invert**	**Inverze**	
			0	fyzický výstup je při přímo propojen s logickým (default)
			1	fyzický výstup inverzně propojen s logickým výstupem
		atribut **Default**	**Přednastavena hodnota výstupu**	
			0	Přednastavena hodnota v případě, že výstup není propojen
			1	Přednastavena hodnota v případě, že výstup není propojen
		atribut **Connected**	**Propojení logického a fyzického výstupu**	
			0	logický výstup není propojen s fyzickým výstupem (default)
			1	logický výstup je propojen s fyzickým výstupem

Příklad:

Příklad mapovaní dvou výstupů `outXEn` a `outYEn`:

```
<Plc>
 <!-- Vystupy  -->
 <Output ID="outXEn" Port="1" Bit="0" Invert="1" Default="0" Connected="1">
 </Output>
 <Output ID="outYEn" Port="1" Bit="1" Invert="1" Default="0" Connected="0">
 </Output>
</Plc>
```

10.3 Konfigurace pro analogové kanály

Konfigurace pro analogové vstupy a výstupy je v souboru typu „ChannelConfig“ v elementu „Plc“.

Konfigurace pro analogové vstupy:

element **Plc**			**Konfigurace pro PLC**	
	element **AnalogInput**		**Konfigurace pro jeden analogový vstup.**	
		atribut **ID**	**Identifikátor vstupu**	
			Abcd	Textový řetězec, který slouží jako identifikátor vstupu
		atribut **PhysNo**	**Číslo fyzického analogového vstupu**	
			0	číslo fyzického vstupního portu (default)
			0-99	číslo fyzického vstupního portu (ANALOG_INPUT_PHYSICAL[100])
		atribut **Scale**	**Měřítko na převod napětí na požadovanou fyzikální veličinu**	
			xx	měřítko
		atribut **Offset**	**Posunutí vstupního napětí**	
			xx	posunutí
		atribut **MinVal**	**Nejmenší přípustná hodnota**	
			xx	nejmenší přípustná hodnota
		atribut **MaxVal**	**Největší přípustná hodnota**	
			xx	největší přípustná hodnota
		atribut **Default**	**Přednastavena hodnota vstupu**	
			xx	Přednastavena hodnota v případě, že vstup není propojen
		atribut **Connected**	**Propojení logického a fyzického vstupu**	
			0	logický vstup není propojen s fyzickým vstupem (default)
			1	logický vstup je propojen s fyzickým vstupem

Postup zpracování vstupního napětí:

- posunutí (přičte se Offset)
- měřítko (vynásobí se hodnotou Scale)
- oříznutí na rozsah <MinVal, MaxVal>

Příklad:

Příklad mapovaní analogového vstupu :

```
<Plc>
  <!—Analogovy vstup  -->
  <AnalogInput ID="PlasmaVoltage" PhysNo="2" Scale="28.82" Offset="0"
  Default="120.0" MinVal="20.0" MaxVal="250.0" Connected="1">
  </AnalogInput>
</Plc>
```

Konfigurace pro analogové výstupy:

element **Plc**			**Konfigurace pro PLC**	
	element **AnalogOutput**		**Konfigurace pro jeden analogový výstup.**	
		atribut **ID**	**Identifikátor výstupu**	
			Abcd	Textový řetězec, který slouží jako identifikátor výstupu
		atribut **PhysNo**	**Číslo fyzického analogového výstupu**	
			0	číslo fyzického výstupního portu (default)
			0-99	číslo fyzického výstupního portu (ANALOG_OUTPUT_PHYSICAL[100])
		atribut **Scale**	**Měřítko na převod konkrétní fyzikální veličiny na napětí**	
			xx	měřítko
		atribut **Offset**	**Posunutí výstupního napětí**	
			xx	posunutí
		atribut **MinVal**	**Nejmenší přípustná hodnota**	
			xx	nejmenší přípustná hodnota
		atribut **MaxVal**	**Největší přípustná hodnota**	
			xx	největší přípustná hodnota
		atribut **Default**	**Přednastavena hodnota výstupu**	
			xx	Přednastavena hodnota v případě, že výstup není propojen
		atribut **Connected**	**Propojení logického a fyzického výstupu**	
			0	logický výstup není propojen s fyzickým výstupem (default)
			1	logický výstup je propojen s fyzickým výstupem

Postup zpracování výstupní hodnoty na napětí:

- ořízntutí na rozsah <MinVal, MaxVal>
- měřítko (vynásobí se hodnotou Scale)
- posunutí (přičte se Offset)

Příklad:

Příklad mapovaní analogového výstupu :

```
<Plc>
  <!—Analogovy vystup  -->
  <AnalogOutput ID="CutFlow" PhysNo="1" Scale="15.62" Offset="0"
  Default="30.0" MinVal="0.0" MaxVal="60.0" Connected="1">
  </Analogoutput>
</Plc>
```

10.4 Definice binárních vstupů a výstupů v PLC programu

Pro definici binárních vstupů a výstupů s možnosti mapování slouží instrukce **MAP_IN** a **MAP_OUT**. Instrukce musí být umístěny (nebo volány) v inicializačním modulu MODULE_INIT.

instrukce	**MAP_IN** **MAP_OUT**

```
funkce      MAP_IN      Definice binárního vstupu s možností mapování
            MAP_OUT     Definice binárního výstupu s možností mapování


syntax      MAP_IN (MAP_OUT)  bit1, poin2
            MAP_IN (MAP_OUT)  bit1, 'TEXT2'
            MAP_IN (MAP_OUT)  bit1, 'TEXT2' [, def3 ]
            MAP_IN (MAP_OUT)  bit1, 'TEXT2' [, def3, FAST ]

1.parametr  „bit1"            název bitu vstupy nebo výstupu
2.parametr  „poin2,TEXT2"     pointer nebo textový řetězec identifikátoru
3.parametr  „def3"            defaultní hodnota
4.parametr  „FAST"            klíčové slovo pro rychlé vstupy a výstupy
```

parametr	název	význam	typ
1.	**bit1**	název bitové proměnné pro vstup nebo výstup. Bitová proměnná se automaticky definuje a proto v PLC programu se už nepoužívá definice pomocí instrukce DFM.	bit
2.	**poin2**	Ukazatel na buffer, kde je umístěný text s klíčovým slovem konfiguračního parametru - náveští u řetězce definovaného instrukcí "str".	pointer
	text2	Přímé zadání textu s klíčovým slovem konfiguračního parametru v apostrofech	řetězec
3.	**def3**	Nepovinný parametr. Přímé zadání defaultní hodnoty, číslo se naplní do vstupu nebo výstupu a použije se v případě, že se nenajde konfigurace mapování v souboru ChannelConfig.	číselná hodnota
4.	**FAST**	Nepovinný parametr. Pokud má instrukce jako 4. parametr klíčové slovo „FAST", bude se daný vstup nebo výstup obsluhovat v rastru průběhu rychlého modulu (1ms).	klíčové slovo

Příklad:
Definování vstupu `inLimX, inRefX` z předchozího příkladu. Vstup `LimX` je požadován jako rychlý vstup. Definování výstupu `outXEn`. Příklad práce se vstupy a výstupy v PLC programu.

```
MODULE_INIT

            MAP_IN      inLimX,'inLimX',-,FAST
            MAP_IN      inRefX,'inRefX',

            MAP_OUT     outXEn,'outXEn'

MODULE_INIT_END

;.....

MODULE_MAIN

      FL    1, outXEn   ;nastaví bit0 na výstupním portu 1 (P1OUT.B0)
                        ;na hodnotu log.0 (inverzně)
                        ;(Port="1" Bit="0" Invert="1")

      LDR   inLimX      ;načte přímo bit2 ze vstupního portu 1 (P2IN.B2)
                        ;(Port="1" Bit="2" Invert="0").
```

10.5 Definice analogových vstupů a výstupů v PLC

Pro definici analogových vstupů a výstupů s možnosti mapování slouží instrukce **`MAP_AIN`** a **`MAP_AOUT`**. Instrukce musí být umístěny (nebo volány) v inicializačním modulu `MODULE_INIT`.

instrukce	**MAP_AIN** **MAP_AOUT**

```
funkce      MAP_AIN     Definice analogového vstupu s možností mapování
            MAP_AOUT    Definice analogového výstupu s možností mapování


syntax      MAP_AIN (MAP_AOUT)      val1, poin2
            MAP_AIN (MAP_AOUT)      val1, 'TEXT2'
            MAP_AIN (MAP_AOUT)      val1, 'TEXT2' [, def3 ]
            MAP_AIN (MAP_AOUT)      val1, 'TEXT2' [, def3, FAST ]

1.parametr  „val1"              název reálné buňky vstupu nebo výstupu
2.parametr  „poin2,TEXT2"       pointer nebo textový řetězec identifikátoru
3.parametr  „def3"              defaultní hodnota – typ real
4.parametr  „FAST"              klíčové slovo pro rychlé vstupy a výstupy
```

parametr	název	význam	typ
1.	**val1**	název reálné proměnné pro analogový vstup nebo výstup. reálná proměnná se automaticky definuje a proto v PLC programu se už nepoužívá definice pomocí instrukce DS.	bit
2.	**poin2**	Ukazatel na buffer, kde je umístěný text s klíčovým slovem konfiguračního parametru - náveští u řetězce definovaného instrukcí "str".	pointer
	text2	Přímé zadání textu s klíčovým slovem konfiguračního parametru v apostrofech	řetězec
3.	**def3**	Nepovinný parametr. Přímé zadání reálné defaultní hodnoty, číslo se naplní do vstupu nebo výstupu a použije se v případě, že se nenajde konfigurace mapování v souboru `ChannelConfig`. Číselná hodnota musí být zadána s desetinnou tečkou.	číselná hodnota
4.	**FAST**	Nepovinný parametr. Pokud má instrukce jako 4. parametr klíčové slovo „`FAST`", bude se daný vstup nebo výstup obsluhovat v rastru průběhu rychlého modulu (1ms).	klíčové slovo

PLC program má přístup k reálným polím **ANALOG_INPUT_PHYSICAL** a **ANALOG_OUTPUT_PHYSICAL**, kde jsou umístěny analogové vstupy a výstupy fyzických kanálů.

```
ANALOG_INPUT_PHYSICAL[100]    ... typ REAL
ANALOG_OUTPUT_PHYSICAL[100]   ... typ REAL
```

Stav namapování v konfiguraci pro fyzické analogové vstupy a výstupy může PLC program zjistit pomocí pole **ANALOG_CONNECTED** typu BYTE, ve kterém jsou definovány bity **AIN_CONNECTED** a **AOUT_CONNECTED**. Hodnota log.1 v příslušném bitu znamená, že příslušny fyzický analogový kanál je namapován.

```
ANALOG_CONNECTED[100]  ... typ BYTE, bity .. AIN_CONNECTED, AOUT_CONNECTED
```

Příklad:

Příklad definování analogového vstupu pro měření napětí plazmy :

```
MODULE_INIT

      MAP_AIN     rPlasmaVoltage,'PlasmaVoltage'

MODULE_INIT_END

;..........
```

```
MODULE_MAIN

      LOD   real.rPlasmaVoltage     ;čtení normovaného napětí plazmy [V]
                                    ;(například 127.6 V)

            ;Příklad pro přístup k fyzické hodnotě analogového vstupu
            ;pro měření napětí plazmy
            ;pro  PhysNo="2", 3.fyzický analog.vstup

      STO   real.ANALOG_INPUT_PHYSICAL+16       ;reálná vstupní hodnota
                                                ;ve voltech
                                                ;(například 6.2 V)


      LDR   ANALOG_CONNECTED+2.AIN_CONNECTED    ;test zda je 3.fyzický
                                                ;analog.vstup
                                                ;namapován
```

8

8. ROZHRANÍ CNC SYSTÉM - PLC PROGRAM

Komunikační rozhraní CNC systém - PLC program tvoří bloky:

POVELOVÝ BLOK	povel z CNC systému do PLC programu (bude označován `PBxx`)
BLOK ZPĚTNÉHO HLÁŠENÍ	zpětné hlášení z PLC programu do CNC systému (bude označováno `BZHxx`)
KOMUNIKAČNÍ OBLAST	komunikační oblast mezi primárním a sekundárním procesorem
RTM OBLAST	zveřejněné **s**ystémové proměnné z modulu reálného času

8.1 PLC rozhraní

8.1.1 Odměřování a diference

Systém standardně používá několik transformací, které jsou zařazeny na výstup interpolátora. Jednotlivé transformace oddělují „**prostory**“, ve kterých je definovaná aktuální poloha:

Transformace	**Prostor**
Interpolace (fiktivní poloha)	
	POS0
Programová transformace	
	POS1
Transformace polotovaru	
	POS2
Délková korekce	
	POS3
Posunutí 1	
	POS4
Posunutí 2	
	POS5
Transformace stroje (reálná poloha)	
	POS6

Prostor	pole 6x16xQWORD [1/64000 µm]	pole 6xDWORD [1/8 µm]	Popis
POS0	**B_POL_EX, B_INK_EX**	B_POL, B_INK	poloha v POS0
POS1	**POSITION_POS1_EX**		poloha v POS1
POS2	**POSITION_POS2_EX**		poloha v POS2
POS3	**POSITION_POS3_EX**		poloha v POS3
POS4	**POSITION_POS4_EX**		poloha v POS4
POS5	**POSITION_POS5_EX**		poloha v POS5
POS6			poloha v POS6

Prostor	pole 16xDWORD [µm]		Popis
POS6	**DIFCIT_X**		diferenční čítač

PLC program má k dispozici odměřování systému pro zjištění okamžité polohy stroje. Může jej využít například pro řízení mazání souřadnic od polohy stroje. Odměřování je součet bufferu polohy B_POL a bufferu inkrementu B_INK. Odměřování je platné od najetí do reference. Přístup k jednotlivým osám je relativní.

PLC program nesmí do bufferů polohy jednoducho zapisovat, protože při změně je potřeba vždy provést všechny transformace (inverzní nebo přímé) a nastavit všechny polohy stroje.

Pro nastavení polohy souřadnic (například při PLC referenci), slouží buňka HOMING_REQ, která spustí v systému okamžitý výpočet transformací (viz. popis pro nájezd do reference)

Pro přepínání prostorů z PLC programu například při ručních pojezdech slouží buňky POS_SPACE_PLC a POS_SPACE_MMM (viz návod „Ruční pojezdy“).

Příklad:
Hlídání přetečení diference v ose Y podle limitu LIMIT_Y

```
CLI                     ;zákaz přerušení (ASM86)
LOD   DWRD.(DIFCIT_X+4) ;načte diferenci Y
STI                     ;povolení přerušení (ASM86)
ABS   DWRD              ;absolutní hodnota DR 32 bitů
GE    DWRD.LIMIT_Y      ;porovnání
JL1   ERROR_Y           ;skok na chybu
```

Příklad:
Zjistění polohy v ose Y (- jen kladné míry)

```
CLI                     ;zákaz přerušení (ASM86)
LOD   DWRD.(B_INK+4)    ;načte buffer inkrementu Y
AD    DWRD.(B_POL+4)    ;načte buffer polohy Y
STI                     ;povolení přerušení (ASM86)
RR    CNST.3,DWRD       ;převod na mikrony
STO   DWRD.POLOHA_Y     ;zápis do polohy Y
```

Příklad mazání podle ujeté dráhy je v kapitole "Užitečné příklady."

8.1.2 Otáčky vřetene

Dále je uveden seznam důležitých systémových proměnných souvisejících s problematikou vřetene. Podrobně je problematika vřeten popsána v kapitole: „Rotační osy a vřetena“.

název	velikost	popis
PB11, PB12	2xBYTE	(jen pro čtení) Binární hodnota, která reprezentuje výstupní napětí (0 – 7FFFh), které odpovídá programovaným otáčkám vřetene. Hodnota není zkorigována vzhledem k procentu S a neprojeví se ani řízení v průběhu konstantní řezné rychlosti.
VYSOVERS	WORD	(jen pro čtení) Binární hodnota, která reprezentuje výstupní napětí (0 – 7FFFh), které odpovídá skutečným otáčkám vřetene. Hodnota je zkorigována vzhledem k aktuálnímu stavu procenta S, pokud je tato korekce povolena (není programována G33) . V průběhu konstantní řezné rychlosti je velikost hodnoty řízená od interpolátoru v závislosti na poloze osy X.
REQ_SPEED_PM	DWORD	(jen pro čtení) Požadované otáčky vřetene v tisícinách otáčky za minutu. Hodnota zohledňuje aktuální stav procenta S a konstantní řeznou rychlost. jedná se o výsledné požadované otáčky vřetene. Hodnota buňky je vypočtena z aktuálního stavu VYSOVERS, z aktuálního stavu převodového stupně a maximálního napětí pro daný převodový stupeň.
REQ_SPEED_PT	DWORD	(jen pro čtení) Požadované otáčky vřetene v tisícinách stupně za výpočtový takt systému. Hodnota odpovídá buňce REQ_SPEED_PM, jen je v jiných jednotkách.
ACT_SPEED_PM **ACT_SPEED2_PM** **AVG_SPEED_PM** **AVG_SPEED2_PM**	DWORD	(jen pro čtení) Aktuální otáčky vřetene v tisícinách otáčky za minutu. Hodnota je vypočtena z odměřovacího čidla na vřetenu. Při výpočtu je zohledněn vnitřní inkrement vřetene. K dispozici také filtrovaná (průměrná) hodnota aktuálních otáček AVG_SPEED_PM a také hodnoty pro druhé vřeteno: ACT_SPEED2_PM a AVG_SPEED2_PM
ACT_SPEED_PT **ACT_SPEED2_PT** **AVG_SPEED_PT** **AVG_SPEED2_PT**	DWORD	(jen pro čtení) Aktuální otáčky vřetene v tisícinách stupně za výpočtový takt systému. Hodnota je vypočtena z odměřovacího čidla na vřetenu. Při výpočtu je zohledněn vnitřní inkrement vřetene podle strojní konstanty R60. K dispozici také filtrovaná (průměrná) hodnota aktuálních otáček AVG_SPEED_PT a také hodnoty pro druhé vřeteno: ACT_SPEED2_PT a AVG_SPEED2_PT
SIM_SPEED_PT **(OTACKY_VR)**	DWORD	(pro zápis) Systémová proměnná pro aktuální otáčky vřetene v tisícinách stupně za výpočtový takt systému. Buňka se používá při simulaci otáček vřetene.
REQW_SPEED2_PM	DWORD	(pro zápis) Požadované otáčky pro 2. vřeteno pro výstup na obrazovku.

8.1.3 Aktuální převodový stupeň

```
SPINDLE_GEAR_ACT_1      ....Převodový stupeň pro 1.vřeteno

( WORD čtení/zápis )

REQ_EXT_SPINDLE_GEAR    ....Externí řízení převodů z PLC
REQ_MAN_SPINDLE_GEAR    ....Manuální nastavování převodů z PLC

( bity čtení/zápis )
```

Aktuální převodový stupeň pro 1.vřeteno je v buňce **SPINDLE_GEAR_ACT_1** (WORD 1,2,..,32). Pokud se používají maximálně 4 převodové stupně pomocí funkcí M41-M44, zapíše se do buňky SPINDLE_GEAR_ACT_1 příslušný stupeň (1 až 4) přímo ze systému. V jiných případech musí aktuální převodový stupeň do buňky zapsat PLC program. V tomto případě musí PLC žádat o externí řízení převodových stupňů nastavením bitů REQ_EXT_SPINDLE_GEAR a REQ_MAN_SPINDLE_GEAR. Aktuální převodový stupeň má vliv na výpočet výstupné hodnoty pro zadané otáčky vřetene (viz. „Rotační osy a vřetena“).

8.1.4 Povolení pohybu od PLC

```
MPxPI               ....Povolení pohybu

( 16 bitů, čtení/zápis,  x = X,Y,Z,4,5,6,7,...,15,16 )
```

PLC program může povolit nebo zakázat pohyb pro jednotlivé osy bity **MPxPI,** definované v BZH08PI a BZH08PI2.

- log. 1 povolení pohybu z PLC
- log. 0 zákaz pohybu z PLC

Implicitně jsou bity MPxPI nastaveny na hodnotu log. 1, takže je pohyb povolen. Implicitní nastavení se provede v rámci instrukce MODULE_INIT.

V bloku zpětného hlášení jsou bity MPx, které jsou výsledkem všech podmínek povolení pohybu. Systémový software zapisuje do bitu MPx výsledek součinu povolení pohybu od různých zdrojů. Do tohoto součinu je zařazeno povolení pohybu např. od modulu interpolace, od modulu reálního času - kde jsou zařazené i koncové spínače (i softwarové) a od modulu servosmyčky. Do součinu jsou zahrnuty i bity MPxPI, které umožňují blokovat pohyb z uživatelského PLC programu. Supervizor interfejsu umožní pohyb na základě nastaveného bitu MPx do log.1 a na základě ukončení průchodu modulem přípravných funkcí.

PLC program může zakázat pohyb i počas pohybu. V tomto případě se pohyb zastaví dojezdovou rampou. Při nastavení log.1 do MPxPI bude pohyb povolen a souřadnice se rozjede rozjezdovou rampou.

Skutečné povolení pohybu nastane v případě požadavku na pohyb PO_OSxPI (viz dále), nastavením log.1 v MPxPI a celým průchodem modulu přípravné funkce (viz kapitola "Struktura PLC programu"). Pro případ, že PLC program nepotřebuje blokovat pohyb od poruch (blokování pohybu od průchodu PLC programu přípravnými a závěrečnými funkcemi zabezpečuje supervizor interfejsu), nemusí být bity MPxPI v PLC programu vůbec nastavovány, protože mají implicitní hodnotu log.1. Ovládání bitů MPxPI se doporučuje prakticky jen v případě poruch souřadnic, které vznikly počas pohybu.

V pomocných ručních pojezdech je možno pomocí konfigurace zvolit, zda má být také pohyb povolován od bitů MPxPI nebo od bitů MPxMAN (viz. kapitola: „Pomocné ruční pojezdy“)

Příklad:
Zablokujte pohyb v ose Z od poruchy ERROR_Z .

```
LDR   ERROR_Z          ;načte poruchový signál
FL1   0,MPYPI          ;podmíněný zákaz pohybu
```

8.1.5 Povolení pohybu pro pomocné ruční pojezdy

MPxMAN ...Povolení pohybu pro pomocné ruční pojezdy

(16 bitů, čtení/zápis, x = X,Y,Z,4,5,6,7,...,15,16)

PLC program může povolit nebo zakázat pohyb pro jednotlivé osy v pomocných ručních pojezdech, bity **MPxMAN,** definované v BZH08MAN a BZH08MAN2.

Aktivace bitů musí být povolena v konfiguraci. Podrobně je tato problematika popsána v kapitole "Pomocné ruční pojezdy".

8.1.6 Povolení pohybu celkové

MPx ....Povolení pohybu

(6 bitů, čtení, x = X,Y,Z,4,5,6)

Bity **MPx** jsou výsledkem všech podmínek povolení pohybu pro interpolátor. Systémový software zapisuje do bitu MPx výsledek součinu povolení pohybu od různých zdrojů. Do tohoto součinu je zařazeno povolení pohybu např. od modulu interpolace, od modulu reálného času - kde jsou zařazené i koncové spínače (i softwarové) a od modulu servosmyčky. Do součinu jsou zahrnuty i bity MPxPI, které umožňují blokovat pohyb z uživatelského PLC programu. Supervizor interfejsu umožní pohyb na základě nastaveného bitu MPx do log.1 a na základě ukončení průchodu modulem přípravných funkcí.
PLC program nesmí do bitů MPx zapisovat.

8.1.7 Pohyb v osách

PO_OSxPI ....Pohyb v osách

(16 bitů, čtení, x = X,Y,Z,4,5,6,7,...,15,16)

Požadavek na pohyb v osách. Pro jednotlivé souřadnice je v bitech **PO_OSxPI** v PB20PI a PB20PI2**.**

- log. 1 požadavek na pohyb
- log. 0 není požadavek na pohyb

Bit PO_OSxPI je součtem všech požadavků na pohyb. Na rozdíl od PO_OSx v povelovém bloku PB20 (viz dále) obsahuje i pohyb od polohovacích jednotek, pomocných ručních pojezdů a pod. Po skončení pohybu se bit PO_OSxPI vynuluje.

Příklad:
Nastartujte mechanizmus uvolnění osy X - UVOL_X v přípravných funkcích, když je pohyb programován

```
            LDR   PO_OSXPI          ;načte požadavek na pohyb
            JL0   NENI_POX          ;obskok, když neni pohyb
            FL    1,UVOL_X          ;start mechanizmu uvolnění osy X
            EX                      ;
            LDR   UVOL_X            ;čekání na uvolnění
            EX1                     ;
NENI_POX:                           ;
```

8.1.8 Směr pohybu

SM_POxPI ....Směr pohybu servosmyčky

(16 bitů, čtení, x = X,Y,Z,4,5,6,7,...,15,16)

Směr pohybu pro jednotlivé servosmyčky je v bitech **SM_POxPI**

- log. 0 kladný směr pohybu
- log. 1 záporný směr pohybu

Směr pohybu je platný až po skutečném rozjetí pohybu, to znamená až po průchodu modulem přípravných funkcí. Pro případ, že je potřeba řídit nějakou činnost na základě směru pohybu v přípravných funkcích (například sepnutí směrových spojek), je potřeba spustit mechanizmus od průchodu přípravných funkcí a tak se zabezpečí jeho paralelní chod s pohybem. Tento případ je vysvětlen na následujícím příkladu:

Příklad:
Ovládání směrových spojek stroje pro osu Y:

```
Konec modulu MODULE_BLOCK_INIT:

            LDR   PO_OSYPI          ;načtení požadavku na pohyb
            FL1   1,SPOJKA_Y        ;podmíněná aktivace mechanizmu SPOJKA_Y
MODULE_BLOCK_INIT_END               ;konec přípravných funkcí - nečekáme na
                                    ;dokončení

                                    ;mechanizmus SPOJKA_Y
V modulu MODULE_MAIN definován mechanizmus:
MECH_BEGIN  SPOJKA_Y
            EX                      ;začátek pohybu na 1 průchod
            FL    0,MPYPI           ;zákaz pohybu
            LDR   SM_POYPI          ;načte platný směr pohybu
            WR    SPOJ_MINUS_Y_O    ;nastaví zápornou spojku  log.1=minus
            CA                      ;negace RLO
            WR    SPOJ_PLUS_Y_O     ;nastaví kladnou spojku log.1=plus
            LDR   KSOJ_Y_I          ;kontrolní vstup
            EX0                     ;čekání na sepnutí spojky
            FL    1,MPYPI           ;povolení pohybu
MECH_END    SPOJKA_Y                ;konec mechanizmu
```

8.1.9 Vypínání a zapínání polohové vazby

VAZBA_x ....Polohová vazba

(16 bitů, čtení/zápis, x = X,Y,Z,4,5,6,7,...,15,16)

Vypínání a zapínání polohové vazby se řídí bitem **VAZBA_x** v VAZBAPI a VAZBAPI2**.**

- log. 0 vypnutí polohové vazby
- log. 1 zapnutí polohové vazby

Systém má softwarovou polohovou vazbu. PLC program může polohovou vazbu nejen zapínat a vypínat, ale může také nastavovat parametry polohové servosmyčky. Podrobně je tato problematika rozepsána v kapitole "Nastavení parametrů servopohonů a jejich řízení PLC programem".

Implicitně jsou bity VAZBA_x nastaveny do log.1, to znamená že je polohová vazba zapnuta. Implicitní nastavení se provede v rámci instrukce MODULE_INIT. Pokud PLC program nepotřebuje polohovou vazbu vypínat, nemusí bity VAZBA_x vůbec obsluhovat.

Vypnutí polohové vazby by se mělo provádět, když je souřadnice v klidu. Nastavením log.0 do příslušného bitu VAZBA_x se odpojí účinek diferenčného čítače na vysílání rychlosti pro pohon. Odměřování souřadnice zůstává dál plně v činnosti. Pokud nebudou použity ještě jiné prostředky (například přepnutí do indikace nebo nulování diference), není v tomto případě vhodné, aby souřadnice jezdila. Odměřování je stále v činnosti a ujetá dráha by se kumulovala v diferenčním čítači. Po čase by mohlo dojít k jeho přetečení. Při opětovném zapnutí polohové vazby by pohon najednou dostal velký napěťový skok, protože polohová vazba by se snažila vynulovat diferenční odchylku.

Vypnutí polohové vazby je výhodné použít například pro upínání souřadnic a pro případ, když je víc souřadnic řízeno stejným pohonem.

Příklad:
Zapínání polohové vazby při uvolnění souřadnice Z.

```
V přípravných funkcích:
            LDR   PO_OSZPI    ;požadavek na pohyb
            JL0   POZ_E       ;obskok
            FL    1,UVOL_Z    ;start mechanizmu uvolňování
            EX                ;
            LDR   UVOL_Z      ;čekáme na uvolnění
            EX1               ;
            FL    1,VAZBA_Z   ;zapnutí polohové vazby
POZ_E:                        ;v závěrečných funkcích bude vypnutí vazby
                              ;společně s upnutím souřadnice
```

8.1.10 Polohování vřetene

```
POLOHV_x          ....Polohování vřetene

( 16 bitů, čtení/zápis,  x = X,Y,Z,4,5,6,7,...,15,16 )
```

Příznak pro polohování vřetene je v bitu **POLOHV_x** v POLOHV a POLOHVPI2.

- log. 1 rotační souřadnice je ve stavu nájezdu na nulový puls
- log. 0 rotační souřadnice dosáhla nulový puls

Příznak `POLOHV_x` se používá v režimu přepnutí vřetene na rotační souřadnici (polohování rotační souřadnice). Používá se v součinosti instrukce `SPI_AX_x`, která změní rychlostní vazbu rotační souřadnice na polohovou a tato se začne pohybovat dojížděcím posuvem a zastaví se až po dosažení nulového pulsu. Nájezd na nulový puls trvá určitou dobu, a proto PLC program musí pozastavit vykonávání dalších funkcí do dokončení nájezdu. Pro indikování najetí rotační souřadnice slouží bitové proměnné `POLOHV_x` (viz kapitolu"Popis řízení regulátorů pohonů rotačních os a vřeten").

8.1.11 Ruční pojezdy

Ruční pojezdy jsou podrobně popsány v kapitole „Ruční pojezdy“ a zde uvádíme jen přehled nastavovacích a informačních signálů rozhraní:

Proměnná	typ	akce	popis
ACK_AUTMAN	bit	čtení	Pomocné ruční pojezdy aktivní
INPOS_STOP	bit	čtení	Systém je v poloze posledního stopu
EN_AUTMAN	bit	čtení, zápis	Povolení ručních pojezdů (přednastaven na 1)
AUTMAN_CONT_NC	bit	čtení	Řízení ručních pojezdů z panelu systému
AUTMAN_CONT_TOC	bit	čtení	Řízení ručních pojezdů z panýlku točítka
AUTMAN_CONT_SELECT	bit	čtení	Provádí se předvolba pohybu
AUTMAN_CONT_G00	bit	čtení	Požadován rychloposuv
EXM_x0	bity	zápis	Externí požadavek na pohyb v kladném směru
EXM_x1	bity	zápis	Externí požadavek na pohyb v záporném směru
REQ_EXT_G00_AUTMAN	bit	zápis	Externí požadavek na rychloposuv
REQ_EXT_SELECT_AUTMAN	bit	zápis	Externí požadavek na předvolbu
REQ_EXT_CONT_AUTMAN	bit	zápis	Externí požadavek na řízení pohybu
REQ_EXT_FEED_AUTMAN	bit	zápis	Externí požadavek na zadání rychlosti
REQ_EXT_G00_AUTMAN	bit	zápis	Externí požadavek na zadání rychloposuvu
EXT_FEED_AUTMAN	word	zápis	Externí zadání rychlosti [mm/min]
EXT_FEED_G00_AUTMAN	word	zápis	Externí zadání pro rychloposuv [mm/min]
AUTMAN_REQ	bit	zápis	Externí požadavek na aktivaci ručních pojezdů

8.1.12 Stop z PLC programu

```
STOPPI              ....Stop z PLC programu
STOP_REQ
STOP_EMERGENCY_REQ

( bity, čtení/zápis )
```

Bit **STOPPI** hodnotou log.1 vyvolá STOP obdobně jako stisknutí tlačítka STOP na ovládacím panelu. Nastavením bitu STOPPI se vyvolá stop celého bloku. Systém reaguje na nástupní hranu signálu.
Stav bitu STOPPI je potřeba v PLC programu vrátit do stavu log.0, to se provede například odčasováním. Minimální délka pulsu signálu STOPPI je asi 200 ms.

Bit **STOP_REQ** je systémem potvrzovaný stop. PLC program hodnotou log.1 vyvolá STOP podobně jako u signálu STOPPI, ale systém po převzetí požadavku bit STOP_REQ sám vynuluje. PLC program se nemusí starat o nulování bitu.

Bit **STOP_EMERGENCY_REQ** je systémem potvrzovaný nouzový stop. PLC program hodnotou log.1 vyvolá nouzový STOP a systém po převzetí požadavku bit STOP_EMERGENCY_REQ sám vynuluje. Nouzový stop používá jiné nastavení ramp pro zastavení.

Přehled použitých ramp z konfigurace pro zastavení při stopu v závislosti na druhu provozu:

Provoz	-	STOP	MPxPI	STOP EMERGENCY
Parabolické rampy AccelerationType="1"	ParabAccel LinearAccel	ParabAccelRT LinearAccelRT	okamžité zastavení	ParabAccelEmergency LinearAccelEmergency
Lineární rampy AccelerationType="0"	Acceleration	AccelerationRT	Acceleration Emergency	
Ruční pojezdy	Acceleration	Acceleration	Acceleration Emergency	AccelerationEmergency

Signály používá PLC program, když nastane v průběhu vykonávání bloku chyba. Například při chybě v přípravných funkcích se zastaví další chod pomocí "nekonečné instrukce typu EX", nastavíme bit STOPPI a supervizor interfejsu přeruší vykonávání bloku.

Příklad:
Ošetření chyby v přípravných funkcích:

```
            LDR   TLAK_I                        ;načte vstup o tlaku
            TEX0  CITAC_TL,CAS_ERROR,ERROR,15h  ;časová kontrola tlaku
            ......
            ......
ERROR:      ESET                                ;nastavení chybového hlášení
            FL    1,STOPPI                      ;STOP z PLC programu
            EX
            LDR   CAPI                          ;"nekonečný stav"
            EX0                                 ;zabránění dalšímu výkonu
                                                ;přípravných funkcí
```

8.1.13 Blokování a pozdržení startu

```
START_DISABLE          ....Blokování startu
START_SUSPEND          ....Pozdržení startu

( bit, čtení/zápis )
```

Je-li bit **START_DISABLE** nastaven do log.1, nelze na systému nic odstartovat.

Blokování se týká všech existujících způsobů startu. Bit START_DISABLE se využije například, když je nutno zakázat i malý případný pohyb, který by mohl vzniknout při okamžitém stopu po odstartování bloku.

Po startu systému je bit START_DISABLE přednastaven do log.0.

Při vykonávání STARTu se nejdřív zavolá v PLC programu událostní procedura _ON_EVENT se vstupním kódem PLCEVENT_START_REQ (viz. „Základní instrukce jazyka Technol“) a bit **START_SUSPEND** se přednastaví na log.0. Pokud PLC program potřebuje pozdržet průběh startu systému, tak hned v událostní proceduře nastaví bit START_SUSPEND na hodnotu log.1. Průběh startu bude pokračovat až když PLC program nastaví START_SUSPEND na hodnotu log.0.

8.1.14 Blok rozpracován

```
PO_F                   ....Potvrzení funkcí

( bit, čtení )
```

Bit **PO_F** hodnotou log.1 potvrzuje převzetí funkcí a pohybu. Je řízen supervisorem interfejsu a na panelu systému se podle něj řídí druhá signálka: Funkce rozpracovány. Po vykonání všech funkcí, to znamená na konci bloku, bit PO_F je shozen do log.0. Při stopu, když blok není ukončen zůstává bit PO_F ve stavu log.1. PLC program může bitem PO_F zjišťovat, zda je stroj v klidu a podle toho řídit případné další akce.

8.1.15 Dosažení zadané polohy

```
PO_POL                 ....Potvrzení dosažení polohy

( bit, čtení )
```

Bit **PO_POL** hodnotou log.1 potvrzuje, že byla dosažena zadaná poloha souřadnic. Bit nastavují moduly interpolace v kazetě systému. Bit je nahozen až po dosažení zadané tolerance polohy (MaxDifference). Po stopu pohybu se bit PO_POL nenastaví, protože naprogramovaná poloha nebyla dosažena.

8.1.16 Potvrzení stopu

PO_STOP ....Potvrzení stopu

(bit, čtení)

Bit **PO_STOP** hodnotou log.1 potvrzuje vykonání stopu. Uživatelský PLC program smí tento bit pouze číst, nastavení provádí supervisor PLC programu.
Bit PO_STOP nabývá stavu log.1 jen tehdy, je-li proveden stop běžícího programu a ne je-li pouze stlačeno tlačítko STOP v době, kdy není co stopovat.
Stop může byt vyvolán různým způsobem, například také signálem STOPPI z PLC programu.

Bit PO_STOP se po stopu dostane do stavu log.1 a v něm zůstane až po nový start systému, kdy se vrátí do stavu log.0. Nejedná se o kopírování tlačítka STOP.

Bit je pro PLC program vhodný pro zjištění, zda je systém ve stavu STOP .

8.1.17 Pohyb ukončen

INPOS ....Poloha v toleranci

(bit, čtení)

Je-li bit **INPOS** ve stavu log.1, je poloha os v toleranci. Bit je řízen programem interpolace, není-li v žádné ose polohová odchylka serva větší než je dovoleno pro jednotlivé osy ve konfiguraci (MaxDifference)..

Rozdíl mezi bity PO_POL a INPOS je ten, že u bitu INPOS se netestuje, zda byla dosažena programovaná poloha. To znamená, že bit INPOS se nastaví do log.1 i při stopu pohybu, pokud byla dosažena tolerance polohové odchylky.

Bit INPOS je pro PLC program užitečný, když PLC program musí zjistit, zda se vykonává pohyb os. PLC program nesmí bit INPOS nastavovat.

8.1.18 RTM chyby

BZH13 ....Číslo chyby z RTM

(bajt, čtení)

PLC program může zjistit čísla chyb z reálného času. Jedná se vždy o kritické chyby řízení servosmyčky, chyby odměřování, chyby CAN-BUSu a pod.
PLC program může buňku pouze číst a na základe nenulové hodnoty například deaktivovat pohony.

8.1.19 Řízení procenta rychlosti z PLC

```
FEED_OVR          .....Řízení procenta rychlosti z PLC

( bit, čtení/zápis )


BZH09, BZH10      .....Volba promile pro nastavení rychlosti

( 2 bajty, čtení/zápis )


FEED_OVR_MULTIPLIER ...Násobící koeficient (promile) pro rychlost

( word, čtení/zápis )
```

Bit **FEED_OVR** hodnotou v log.1 provede odstavení potenciometru %F na hlavním ovládacím panelu a informaci o nastaveném procentu rychlosti přebírá z bajtů **BZH09** a **BZH10**.
Do bajtů BZH09 a BZH10 se zapisuje promile rychlosti ve formátu šestnáctibitového slova BIN.
Osa pojede programovanou rychlostí, bude-li v BZH09 a BZH10 hodnota 1000.

Bit FEED_OVR se využije při dálkovém ovládání, například z pomocného ovládacího panelu.

Systém nekontroluje hodnoty zapsané do BZH09 a BZH10, souřadnice však nepojede v žádné ose větší rychlostí než je zadaná v konfiguraci pro rychloposuv.

Použití násobícího koeficientu **FEED_OVR_MULTIPLIER** je jinou možností, jak PLC program může ovlivňovat rychlost. Jedná se o wordovou buňku, přednastavenou na hodnotu 1000. PLC program změnou hodnoty mění rychlost s citlivostí na promile, přitom zůstává funkční i standardní potenciometr %F.

$$F_k = F_z \times \frac{(\%F)}{100} \times \frac{FEED\_OVR\_MULTIPLIER}{1000}$$

F_k ..... rychlost korigovaná
F_z ..... rychlost zadaná

Příklad:
Řízení rychlosti z PLC programu.

```
FL    1,FEED_OVR        ;nastavení požadavku na řízení rychlosti z PLC
LOD   PROMILE           ;promile požadované rychlosti
STO   WORD.BZH09        ;zápis do BZH09 a BZH10
```

8.1.20 Řízení procenta otáček z PLC

SPEED_OVR .....Řízení procenta otáček z PLC

(bit, čtení/zápis)

SPEED_OVR_EXT .....Volba promile pro řízení otáček

(2 bajty, čtení/zápis)

SPEED_OVR_MULTIPLIER ...Násobící koeficient (promile) pro otáčky

(word, čtení/zápis)

Bit **SPEED_OVR** hodnotou v log.1 provede odstavení potenciometru %S na hlavním ovládacím panelu a informaci o nastaveném procentu rychlosti přebírá z bajtů **SPEED_OVR_EXT**.
Do bajtů SPEED_OVR_EXT se zapisuje promile rychlosti ve formátu šestnáctibitového slova BIN.

Použití násobícího koeficientu **SPEED_OVR_MULTIPLIER** je jinou možností, jak PLC program může ovlivňovat otáčky. Jedná se o wordovou buňku, přednastavenou na hodnotu 1000. PLC program změnou hodnoty mění otáčky s citlivostí na promile, přitom zůstává funkční i standardní potenciometr %S.

$$S_k = S_z \times \frac{(\%S)}{100} \times \frac{SPEED\_OVR\_MULTIPLIER}{1000}$$

S_k ..... rychlost korigovaná
S_z ..... rychlost zadaná

8.1.21 Prodleva

PRODLEVA ....Signálka prodleva

(bit, čtení/zápis)

Nastavení bitu **PRODLEVA** do log.1 způsobí rozsvícení signálky "PRODLEVA" na ovládacím panelu. Signálka "PRODLEVA" se rozsvítí, je-li programována časová prodleva pomocí G04.

8.1.22 Restart

```
BLOCK_RESTART     ....Restart bloku

( bit, čtení )


STATUS_RESTART    .... Stav procházeni bloku při Restartu

( bajt, čtení )
```

Pod „restartem bloku“ se rozumí opětovný start bloku po předchozím stopu programu. Stav procházení bloku při restartu může nabývat hodnot:

BLOCK_IDLE	0	Nečinnost
BLOCK_INIT	1	Přípravné funkce
BLOCK_MOVE	2	Pohyb
BLOCK_DELAY	3	Prodleva
BLOCK_DONE	4	Závěrečné funkce

8.1.23 Reference

Problematika referencí je podrobně popsána v kapitole „Způsoby reference“ a zde uvádíme jen přehled nastavovacích a informačních signálů rozhraní:

Proměnná	typ	akce	popis
HOMING	WORD bity os	čtení/zápis	Jednotlivé bity jsou příznaky pro platnou referenci NC os. Bity testuje systém v případě, že NC osa má v konfiguraci nastaveny atributy „NeedRef“ nebo „AutNeedRef“.
HOMING_REQ	WORD bity os	čtení/zápis	Jednotlivé bity jsou žádost o nastavení nulového bodu v prostoru POS5 z PLC. Nulový bod se nastavuje podle atributu „MachineNullPoint“. Systém po nastavení provede všechny zpětné transformace.
REFPOINT_DIST	16x QWORD	čtení/zápis	Posunutí od nulového bodu stroje při zadání HOMING_REQ [1/64000 μm]. Nastavuje PLC.
HOMING_START_REQ	WORD bity os	čtení/zápis	Jednotlivé bity jsou žádost z PLC o vykonání kompletní reference pro jednotlivé osy. Po provedení reference systém bit vynuluje. Reference se provede podle aktuální konfigurace.
HOMING_SINGLE	bit	čtení/zápis	Žádost o referenci jedné osy v závislosti na konfiguraci „RefMethod“. Bit nastavuje systém.
HOMING_PLC	bit	čtení/zápis	Žádost o referenci jedné osy pro PLC. PLC bit pro převzetí vynuluje. Bit nastavuje systém.
HOMING_GROUP	bit	čtení/zápis	Žádost o skupinovou referenci pro PLC. PLC bit pro převzetí vynuluje. Bit nastavuje systém.
SLS_REFER	WORD bity os	čtení/zápis	Jednotlivé bity povolují test na softwarové limitní spínače. Každý bit slouží pro jednu NC osu.
NlcAxisEnable	WORD bity os	čtení/zápis	Jednotlivé bity povolují nelineární softwarové korekce a tepelnou kompenzaci. Každý bit slouží pro jednu řídicí NC osu pro nelineární korekce.

NlcServoEnable	WORD bity serv	čtení/zápis	Jednotlivé bity povolují nelineární softwarové korekce a tepelnou kompenzaci. Každý bit slouží pro jednu kompenzovanou servosmyčku. Systém bity po zapnutí přednastaví na hodnotu log.1.
KRx	16 bitů	čtení/zápis	referenční spínače
KRRx	16 bitů	čtení/zápis	reverzační spínače
ZPRx	16 bitů	čtení/zápis	zpomalovací referenční spínače

8.1.24 Limitní a zpomalovací spínače

```
KHx0              ....Limitní spínače v kladném směru
KHx1              ....Limitní spínače v záporném směru
ZPx0              ....Zpomalovací limitní spínač
ZPx1              ....Zpomalovací limitní spínač

(16 bitů, čtení/zápis,  x = X,Y,Z,4,5,6,7,...,15,16 )
```

Hodnota log.1 v **KHx0** nebo v **KHx1** způsobí zastavení pohybu v dané ose. Zastavení řídí interpolátor dojezdovou rampou podle konfigurace „AccelerationEmergency", nebo v případe parabolických ramp podle sady pro rychloposuv „DynamicControlRT".

Vyjetí z limitního spínače je možné pouze v pomocných ručních pojezdech. V režimu AUT způsobí najetí na jakýkoliv limitní koncový spínač zastavení pohybu. Jestliže by se signál KHx0 nebo KHx1 opět vrátil do stavu log.0, pohyb souřadnice bude pokračovat (není samodrž). Případnou samodrž musí zabezpečit PLC program.
Hodnota log.1 v bitech **ZPx0** a **ZPx1** způsobí přechod na zpomalovací rampu. až pokud nebude dosažena velikost zpomalovacího posuvu. CNC systém u zpomalovacích limitních spínačů nevyhodnocuje směr. Když je potřeba vyhodnocovat směr pohybu, musí to provést PLC program na základě signálů SM_POxPI.

8.2 Povelový blok – kompatibilní verze s CNC8x9

Tato verze povelového bloku je uvedena jen pro usnadnění přechodu ze starších PLC programů pro systém CNC8x9 a CNC8x6 na novější (Windowskou) verzi systému CNC872. Při tvorbě nového PLC programu pro systém CNC872 se **nedoporučuje používat !**

8.2.1 Přehled použitých M funkcí podle skupin

```
Skupina 1:  M00   programový stop   (M00_PID, M01_PID, M02_PID, M30_PID)
            M01   volitelný stop
            M02   konec partprogarmu
            M30   konec partprogramu

Skupina 2:  M03   start vřetene CW             (M03PI, M04PI, M19PI a PB06)
            M04   start vřetene CCW
            M05   stop vřetene
            M19   stop vřetene v orientovaném bodě

Skupina 3:  M41   otáčky vřetene - rozsah 1    (M41PI, M42PI, M43PI, M44PI)
            M42   otáčky vřetene - rozsah 2
            M43   otáčky vřetene - rozsah 3
            M44   otáčky vřetene - rozsah 4
            M40   automatická převodovka

Skupina 4:  ...   Funkce dle konfigurace „MFunctions"

Skupina 5:  M07   zapnutí chlazení 1      (M07PI, M08PI)
            M08   zapnutí chlazení 2
            M09   vypnutí chlazení 1 a 2
            M17   zapnutí chlazení 1 a 2

Skupina 6:  M50   zapnutí chlazení 3      (M50PI, M51PI)
            M51   zapnutí chlazení 4
            M52   zapnutí chlazení 3 a 4
            M53   vypnutí chlazení 3 a 4

Skupina 7:  M10   upnutí obrobku          (M10PID, M11PID)
            M11   uvolnění obrobku

Skupina 8:  M48   zrušení feed-overide
            M49   aktivace feed-overide

Skupina 9:  M06   Výměna nástroje         (M06PID a M60PI)
            M60   Výměna obrobku

Skupina 10: ...   Funkce dle konfigurace „MFunctions" (PB03)

Skupina 11: ...   Funkce dle konfigurace „MFunctions" (PB13)

Skupina 12: ...   Funkce dle konfigurace „MFunctions" (PB14)

Skupina 13: ...   Funkce dle konfigurace „MFunctions" (PB15)

Skupina 14: ...   Ostatní funkce                      (PB16)
```

8.2.2 Režim

```
AUTPI              ....Režim AUT
REFPI              ....Režim nájezdu do reference
CAPI               ....Režim centrální anulace

( bity, čtení )
```

Bit **AUTPI** signalizuje hodnotou log.1, že byl odstartován blok v automatickém režimu (AUT), včetně všech možností modifikace (BB, AVP, M01, ND a LOM). Bit se vysílá jen po startu bloku a není průběžně aktualizován. Případná změna režimu se projeví až novým startem bloku, například odstartováním centrální anulace.

Bit **REFPI** signalizuje hodnotou log.1, že byl odstartován nájezdu do reference. Bit je nastavován při skutečném nájezdu do reference a při pseudoreferenci. V případě simulace reference a nulování reference nastavován není.

Bit **CAPI** signalizuje hodnotou log.1, že byl odstartován blok v režimu centrální anulace. Bit se vysílá jen po startu bloku a není průběžně aktualizován. Případná změna režimu se projeví až novým startem bloku.

8.2.3 Závitování

```
G33PI              ....Závitování

( bit, čtení )
```

Bit **G33PI** indikuje hodnotou log.1, že v bloku je programováno závitování funkcí `G33`. Bit `G33PI` využívají systémové prostředky interpolátoru v RTM.

8.2.4 Rychloposuv

```
G00PI              ....Rychloposuv

( bit, čtení )
```

Bit **G00PI** signalizuje hodnotou log.1, že byl odstartován blok, ve kterém je programován rychloposuv. Bit se vysílá jen po startu bloku a není průběžně aktualizován. Případná změna režimu se projeví až novým startem bloku.
Bit `G00PI` se dá využít například pro uvolnění zpomalovacích limitních spínačů, a tak se zabezpečí, že nebudou reagovat pro pracovní posuv, ale jen pro rychloposuv.

8.2.5 BCD funkce v povelovém bloku

V další části budou popsány funkce, které vystupují v povelovém bloku v BCD tvaru. Jedná se o všechny skupiny M-funkcí a funkce S, H. Změna v BCD hodnotách u těchto funkcí je vždy doprovázena příslušným změnovým bitem.

```
PB03              ....M funkce skupiny 10
PB05              ....H funkce
PB07-PB10         ....T funkce
PB06              ....M funkce skupiny 2
PB13              ....M funkce skupiny 11
PB14              ....M funkce skupiny 12
PB15              ....M funkce skupiny 13
PB16              ....M funkce skupiny 14

( bajty, čtení )
```

V buňkách je BCD kód M funkce ze příslušné skupiny. Funkce sdružené do skupin 10,11,12,13 a 14 jsou určeny k volnému použití. Do příslušné skupiny jsou zařazeny funkce, které jsou uvedeny v konfiguraci podle atributu „GroupNo“:

Příklad:
Zařazení funkce M92 do skupiny 12:

```
<MFunctions>
      <MFunc No="92" GroupNo="12" BreakCont="1"></MFunc>
```

8.2.6 Změnové bity

```
ZMSHPI            ....změnový bit H funkce
ZMSTPI            ....změnový bit T funkce
ZMMxPI            ....změnoví bit M funkce x.skupiny

( bity, čtení,  x = 1,2,3,4,...,14 )
```

Změnové bity se nastavují při změně hodnoty příslušné funkce a zůstanou nastaveny po dobu trvání bloku.
Systém nastavuje změnové signály ve dvou případech:

- Je změna v dané technologické funkci (BCD výstup nebo dekódované funkce). Změna vznikla naprogramováním nové hodnoty, která je jiná než předcházející hodnota.
- Není změna hodnoty v dané technologické funkci, ale tato byla opět v bloku programována.

Změnové signály se v PLC programu testují hlavně v přípravných a závěrečných funkcích a na základě nich se až dekódují BCD výstupy v povelovém bloku nebo dekódované funkce. Aby nedocházelo ke zpomalení průchodu modulem přípravných a závěrečných funkcí, první instrukce typu EX se napíše až za obskokem, jak je patrné na příkladu:

Příklad:
Zapnutí vřetena (pokud je ovládáno pouze z NC programu).

```
LDR   ZMM2PI      ;test změnového signálu 2. skupiny M funkcí
JL0   VRETENO_END ;skok na konec
LDR   M03PI       ;test funkce M03
JL0   SKOKM4      ;skok na další testy
FL    1,CW        ;start mechanizmu CW
EX                ;
LDR   CW          ;čekání na dokončení mechanizmu CW
EX1               ;
JUM   VRETENO_END ;skok na konec
```

8.2.7 Dekódované funkce 1. skupiny M

```
M00_PID           ...Dekódovaná funkce M00
M01_PID           ...Dekódovaná funkce M01
M02_PID           ...Dekódovaná funkce M02
M30_PID           ...Dekódovaná funkce M30

( bity, čtení )
```

Bit **M00_PID** se nastaví do log.1 na dobu jednoho bloku, je-li v bloku programována funkce M00.
Bit **M01_PID** se nastaví do log.1 na dobu jednoho bloku, je-li v bloku programována funkce M01. Jestli se funkce M01 v bloku skutečně uplatní, závisí od toho, zda je aktivní modifikace M01 režimu AUT.
Bit **M02_PID** se nastaví do log.1 na dobu jednoho bloku, je-li v bloku programována funkce M02.
Bit **M30_PID** se nastaví do log.1 na dobu jednoho bloku, je-li v bloku programována funkce M30.

8.2.8 Dekódované funkce 2.skupiny M

```
M03PI             ....Dekódovaná funkce M03
M04PI             ....Dekódovaná funkce M04
M19PI             ....Dekódovaná funkce M19

( bity, čtení )
```

Dekódovaný bit **M03PI** je statický a je určen pro roztočení vřetena ve směru točení hodinových ručiček. Je-li v bloku NC programu programována funkce M03, je nastaven bit M03PI do log.1. Tento stav setrvá do doby, než je jinou funkcí z 2. skupiny M změněn.
Funkce M03 nastaví v BCD tvaru buňku PB06 na hodnotu 03h.

Dekódovaný bit **M04PI** je statický a je určen pro roztočení vřetena proti směru točení hodinových ručiček. Je-li v bloku NC programu programována funkce M04, je na počátku výkonu tohoto bloku nastaven bit M04PI do log.1. Tento stav setrvá do doby, než je jinou funkcí z 2. skupiny M změněn.
Funkce M04 nastaví v BCD tvaru buňku PB06 na hodnotu 04h.

Dekódovaný bit **M19PI** je statický a slouží pro programování zapolohování vřetena. Celé zapolohování vřetena musí provést PLC program. Bit M19PI se nastaví na hodnotu log.1, když je funkce M19 v bloku programována a tento stav trvá do doby, než je jinou funkcí z 2. skupiny M změněn.
Funkce M19 nastaví v BCD tvaru buňku PB06 na hodnotu 19h.

Je-li v bloku NC programu programována funkce M05, mají bity M03PI, M04PI a M19PI hodnotu log.0 a buňka PB06 je nastavena na hodnotu 05h.

8.2.9 Dekódované funkce 3.skupiny M

```
M41PI              ....Dekódovaná funkce M41
M42PI              ....Dekódovaná funkce M42
M43PI              ....Dekódovaná funkce M43
M44PI              ....Dekódovaná funkce M44

( bity, čtení )
```

Dekódované bity **M41PI, M42PI, M43PI a M44PI** jsou statické a nastavují se podle funkcí M41, M42, M43 a M44. Funkce M41 je určena k zařazení prvního převodového stupně vřetene. Funkce M42, M43 a M44 je určena k zařazení druhého, třetího a čtvrtého převodového stupně vřetene.
Je-li v bloku NC programu programována funkce M41, je v bloku nastaven bit M41PI do log.1. Je-li v některém dalším bloku programu programována některá z funkci M42, M43, M44, je v bloku shozen bit M41PI do log.0 a nahozen příslušný bit M42PI, M43PI nebo M44PI podle převodového stupně. CNC systém kontroluje, zda zadané otáčky odpovídají převodovému stupni podle konfigurace .

Funkce M40 je automatická převodovka a způsobuje, že systém automaticky nastavuje proměnné M41PI, M42PI, M43PI, M44PI v závislosti na programované funkci S. Podrobněji je o problematice řízení vřetena pojednáno v kapitole "Zadávání otáček vřetena".

8.2.10 Dekódované funkce 5.skupiny M

```
M07PI              ....Dekódovaná funkce M07
M08PI              ....Dekódovaná funkce M08

( bity, čtení )
```

Funkce	M07PI	M08PI
M07 zapnutí chlazení 1	1	0
M08 zapnutí chlazení 2	0	1
M09 vypnutí chlazení 1 a 2	0	0
M17 zapnutí chlazení 1 a 2	1	1

Dekódovaný bit **M07PI** je statický a je určen pro zapnutí chlazení 1. Je-li v bloku NC programu programována funkce M07 nebo M17, je v bloku nastaven bit M07PI do log.1. Je-li v bloku programována funkce M08 nebo M09, je bit M07PI shozen do log.0.

Dekódovaný bit **M08PI** je statický a je určen pro zapnutí chlazení 2. Je-li v bloku NC programu programována funkce M08 nebo M17, je v bloku nastaven bit M08PI do log.1. Je-li v bloku programována funkce M07 nebo M09, je bit M08PI shozen do log.0.

8.2.11 Dekódované funkce 6.skupiny M

```
M50PI              ....Dekódovaná funkce M50
M51PI              ....Dekódovaná funkce M51

( bity, čtení )
```

Funkce	M50PI	M51PI
M50 zapnutí chlazení 3	1	0
M51 zapnutí chlazení 4	0	1
M52 zapnutí chlazení 3 a 4	1	1
M53 vypnutí chlazení 3 a 4	0	0

Dekódovaný bit **M50PI** je statický a je určen pro zapnutí chlazení 3. Je-li v bloku NC programu programována funkce M50 nebo M52, je v bloku nastaven bit M50PI do log.1. Je-li v bloku programována funkce M51 nebo M53, je bit M50PI shozen do log.0.

Dekódovaný bit **M51PI** je statický a je určen pro zapnutí chlazení 4. Je-li v bloku NC programu programována funkce M51 nebo M52, je v bloku nastaven bit M51PI do log.1. Je-li v bloku programována funkce M50 nebo M53, je bit shozen M51PI do log.0.

8.2.12 Dekódované funkce 7.skupiny M

```
M10PID             ....Dekódovaná funkce M10
M11PID             ....Dekódovaná funkce M11

( bity, čtení )
```

Bity **M10PID** a **M11PID** jsou dynamické a nastavují se jen po dobu trvání jednoho bloku. Nastavením hodnoty log.1 se určuje, že v bloku byla programována funkce upnutí obrobku M10 nebo funkce uvolnění obrobku M11.

8.2.13 Dekódované funkce 9.skupiny M

```
M06PID             ....Dekódovaná funkce M06
M60PI              ....Dekódovaná funkce M60

( bity, čtení )
```

Bity **M06PID a M60PI** jsou dynamické s platností jednoho bloku. Nastavením hodnoty log.1 se určuje, že v bloku je programována funkce pro výměnu nástroje M06 nebo M60.

8.3 Povelový blok – verze pro CNC872

Pro verzi systému CNC872 (Windows) možno použít všechny prvky PLC rozhraní popsány dříve. Místo starší kompatibilní verze povelového bloku se doporučuje používat novější verzi pro CNC872.

8.3.1 Bitové proměnné povelového bloku

Pro načtení bitových proměnných se doporučuje používat instrukce `LDR, LA LO` a `LX` s prefixem „`BLK[xx]`" (viz dále). Všechny bity může PLC program jenom číst.

Název bitu	Popis
M0,M1,...,M14	Změnové bity pro jednotlivé skupiny M funkcí
FirstBlock	První blok NC programu
LastBlock	Poslední blok NC programu
Stop	Programován stop v NC programu (M00)
CondStop	Podmínění stop v NC programu (M01)
Backward	Jízda nazpátek
SelBlock	Volba bloku
Cont	Blok s přískokem z místa mimo dráhu
TchlOff	Blok se má jet bez technologie
Partial	Blok jede odprostřed, (stop a opětovný start)
HProg	Změnový bit pro H funkci
TProg	Změnový bit pro T funkci
RevFeed	Programována rychlost v mm/otáčku (G95)
ContinuousMode	Programována plynulá jízda (G23)
ConstCutSpeed	Programována konstantní řezná rychlost (G96)
AxNC0Move **AxNC1Move** **AxNC2Move,...**	Příznaky pohybu pro jednotlivé NC osy
IPlc0 **IPlc1** **IPlc2,...**	Změnové bity pro celočíselné programované parametry pro PLC
RPlc0 **RPlc1** **RPlc2,...**	Změnové bity pro reálné programované parametry pro PLC

8.3.2 Přístup k prvkům bloku použitím prefixu „BLK[xx]“

Pro zjednodušení přístupu PLC programu k prvkům struktury aktuálního bloku nebo příštích bloků slouží zvláštní prefix **„BLK[xx].“** pro instrukce **LDR,LA,LO** a **LX**. V hranaté závorce prefixu se zadává pořadové číslo z fronty připravených bloků. Pořadové číslo „0“ znamená přístup k aktuálnímu bloku, „1“ je pro první příští blok a pod. PLC program má možnost zjistit aktuální počet připravených bloků v proměnné **„CNT_NEXT_BLOCKS“** (typ WORD), která se aktualizuje při startu průchodu modulu „MODULE_BLOCK_INIT“. V hranaté závorce prefixu může být zadán také index-registr „BX“.

Příklad:
Použití instrukcí LDR, LA, LO a LX s prefixem BLK[xx]:

```
LDR   BLK[0].M3         ;načtení změnového bitu 3.skupiny M
JL1   JE_NOVA

LDR   BLK[0].M2         ;načtení změnového bitu 2.skupiny M
LA    BLK[0].FirstBlock ;jen z 1. bloku programu
JL1   JE_NOVA

LDR   BLK[1].TchlOff    ;Je příští blok s vypnutou technologií ?
JL1   TCH_OFF

LDR   BLK[BX].IPlc1     ;Je změna 2.celočíselného PLC parametru ?
JL1   PLC1_CHANG        ;(ve frontě příštích bloků podle BX)
```

8.3.3 Celočíselné proměnné povelového bloku

Pro práci s celočíselnými (DWORD) proměnnými je možno používat instrukce **LOD, AD, SU, ORB, ANDB, XORB MULB, DIVB, EQ, LE, LT, GE a GT** s prefixem **„BLK[xx]“**. Všechny buňky může PLC program jenom číst. Pro pole je možno použít indexaci, přitom index je číslován po DWORDech od 0 (0,1,2,.. jen pro typ DWORD). U instrukcí je povoleno také používat přetypování, pokud požadujeme jiný typ něž DWORD (přetypování musí být uvedeno jako další prefix za prefixem BLK[xx]).

Název	Popis
M	Programované M funkce jednotlivých skupin, pole 15xDWORD
IPlcParams	Programované celočíselné parametry, pole 5xDWORD
T	Programovaná hodnota adresy T DWORD
AxG	Definice geometrických os (definované jako index do pole NC os) pole 3xDWORD

Příklad:
Použití instrukce `LOD (AD,SU,ORB,ANDB,XORB,MULB,DIVB,EQ,LT,GE,LE,GT):`

```
LDR   BLK[0].M11              ;změnový bit 11.skupiny M
JL0   NENI_NOVA
LOD   BLK[0].M[11]            ;načte hodnotu 11.skupiny M (DWORD)
EQ    CNST.95h                ;test na M95
WR    FUN_95

LOD   BLK[0].IPlcParams[2]    ;načte 3.celočíselný PLC parametr
...
EQ    BLK[0].WORD.T           ;porovnání s hodnotou T jako WORD
...
LOD   BLK[0].BYTE.M[12]       ;načte hodnotu 12.skupiny M (BYTE)
...
```

8.3.4 Reálné proměnné povelového bloku

Pro práci s reálnými proměnnými je možno používat instrukce **`LOD, AD, SU, MULB, DIVB, EQ, LE, LT, GE a GT`** s prefixy **„`BLK[xx]`"** a **„`REAL`"**. Všechny buňky může PLC program jenom číst. Pro pole je možno použít indexaci, přitom index je číslován po reálných typech 0 (0,1,2,.. jen pro typ REAL). U instrukcí je nutno použít přetypování na „`REAL`" a přetypování musí být uvedeno jako další prefix za prefixem `BLK[xx]`.

Název	Popis
`H`	Programovaná hodnota adresy H QWORD FLOATING
`RPlcParams`	Programované reálné parametry, pole 5x QWORD FLOATING
`Delay`	Časová prodleva programovaná v bloku QWORD FLOATING
`AxNC`	Absolutní poloha pro NC osy v Pos 0 [mm] pole 16x QWORD FLOATING
`BlockLen`	Délka dráhy bloku v Pos 0 [mm]. Pro přepočet na skutečnou délku dráhy v Pos 5 lze použít měřítko (proměnná Scale). QWORD FLOATING
`Scale`	Měřítko bloku počítané z programové trasformace a transformace polotovaru. Měřítko je platné pouze v případě, že transformace neobsahují změnu měřítka různou v jednotlivých osách. QWORD FLOATING
`FeedProgr`	Programovaná rychlost suportu pro pracovní posuv [mm/min] QWORD FLOATING
`FeedMax`	Maximální rychlost suportu.[mm/min] QWORD FLOATING
`FeedCriterial`	Kriteriální rychlost suportu na začátku bloku. QWORD
`ParabAccel`	Parabolické zrychlení pro dynamické řízení rychlosti QWORD FLOATING
`LinearAccel`	Lineární zrychlen pro dynamické řízení rychlosti QWORD FLOATING
`SpindleSpeed`	Otáčky vřetene [ot/min] QWORD FLOATING
`SpindleSpeedLimit`	Limit otáček vřetene pro konstantní řeznou rychlost [ot/min] QWORD FLOATING
`ConstCuttingSpeed`	Rychlost pro režim konstantní řezné rychlosti [mm/min] QWORD FLOATING

Pro všechny instrukce s reálnými čísly platí, že po operaci zůstává výsledek jak v reálném registru DR_REAL, tak v celočíselném registru DR64. Proto je možno kdykoli pokračovat standardními celočíselnými operacemi. Nelze ale kombinovat celočíselné a reálné operace (viz „Základní instrukce – Reálné operace“).

Příklad:
Použití instrukce `LOD (AD,SU,MULB,DIVB,EQ,LT,GE,LE,GT):`

```
LOD   BLK[0].REAL.H                ;Načte hodnotu funkce H přímo
STO   REAL.rBunH                   ;Zapíše jako reálné číslo

LOD   BLK[0].REAL.RPlcParams[2]    ;Načte hodnotu 3.real.parametru
AD    BLK[0].REAL.RPlcParams[3]    ;Připočte hodnotu 4.real.parametru
...
LOD   BLK[BX].REAL.BlockLen        ;Načte délku bloku v um
...                                ;ve frontě příštích bloků podle BX
```

8.3.5 Konfigurace M funkcí

Přehled použitých M funkcí podle skupin (Význam je popsán v kapitole: „Povelový blok – kompatibilní verze CNC8x9“):

skupina	M funkce
1	M00 M01 M02 M30
2	M03 M04 M05 M19
3	M41 M42 M43 M44 M40
4	
5	M07 M08 M09 M17
6	M50 M51 M52 M53
7	M10 M11
8	M48 M49
9	M06 M60
10	
11	
12	
13	
14	Ostatní funkce

Konfigurace M funkcí se provede v souboru typu „ChannelConfig“

element **MFunctions**			**konfigurace M funkcí**	
	element **MFuncs**		**konfigurace jedné M funkce**	
		atribut **No**	**Číslo M funkce**	
			xx	číslo M funkce (00 – 100)
		atribut **GroupNo**	**Skupina, do které daná M funkce patří**	
			xx	skupina M funkci (1 – 14)
			14	skupina pro ostatní funkce (default)
		atribut **BreakCont**	**Ruší daná M funkce plynulou jízdu na začátku bloku?**	
			0	M funkce neruší plynulou jízdu na začátku bloku (default)
			1	M funkce ruší plynulou jízdu na začátku bloku
		atribut **BreakContEnd**	**Ruší daná M funkce plynulou jízdu na konci bloku?**	
			0	M funkce neruší plynulou jízdu na konci bloku (default)
			1	M funkce ruší plynulou jízdu na konci bloku
		atribut **ChangePos**	**Mění daná M funkce polohu stroje?**	
			0	M funkce nemění polohu stroje (default)
			1	M funkce mění polohu stroje

Obecné pravidla pro definici M funkcí:

- Do všech skupin je možno přidávat nové M funkce.
- Nově přidaným M funkcím je možno libovolně nastavit atributy „BreakCont, BreakContEnd a ChangePos“.
- Pevně přiřazené M funkce není možno ze skupin vyřadit.
- Pevně přiřazeným M funkcím je možno měnit atributy „BreakCont, BreakContEnd a ChangePos“
- Ostatní M funkce, které nejsou definovány, se přiřadí do skupiny 14.

Příklad:

V příklade se definují v 10. skupině funkce M91,M92,M93 a M94 které ruší plynulou jízdu na začátku bloku. V 11. skupině se definují fumkce M95, M96, M97 a M98, které ruší plynulou jízdu na konci bloku. Pro funkci M6 se nastavuje vlastnost, že ruší plynulou jízdu na začátku bloku a že se také mění poloha stroje.

```
<MFunctions>
            <!-- Skupina 9 -->
      <MFunc No="6" GroupNo="9" BreakCont="1" ChangePos="1"></MFunc>

            <!-- Skupina 10 -->
      <MFunc No="91" GroupNo="10" BreakCont="1"></MFunc>
      <MFunc No="92" GroupNo="10" BreakCont="1"></MFunc>
      <MFunc No="93" GroupNo="10" BreakCont="1"></MFunc>
      <MFunc No="94" GroupNo="10" BreakCont="1"></MFunc>
            <!-- Skupina 11 -->
      <MFunc No="95" GroupNo="11" BreakContEnd="1"></MFunc>
      <MFunc No="96" GroupNo="11" BreakContEnd="1"></MFunc>
      <MFunc No="97" GroupNo="11" BreakContEnd="1"></MFunc>
      <MFunc No="98" GroupNo="11" BreakContEnd="1"></MFunc>

</MFunctions>
```

8.3.6 Status

PLC program má možnost číst status z modulu reálného času pomocí bitů:

bit	popis
SF_RUN	"Systém v chodu" (stroj "jede", a/nebo jsou prováděny technologické funkce)
SF_BLOCKINPROGR	Blok rozpracován
SF_BLOCKFINISHED	Blok ukončen
SF_STOP	Všechny druhy stopu (Stop, M00, M01, BB, stop po volbě programu a pod.). Všechny bloky dokončeny, když SF_BLOCKINPROGR není nastaveno, nebo vykonávání přerušeno, když je SF_BLOCKINPROGR nastaveno.
SF_NOTINPOS	Dosud nebylo dosaženo požadované polohy (některá z os jede nebo diference některé z os neklesla pod zadanou mez).
SF_DELAYINPROGR	Časová prodleva - po dobu prodlevy programované funkcí G04.
SF_INDICATIONMODE	Indikační režim
SF_SIMULATIONMODE	Simulační režim ("ježdění bez hardwaru")
SF_HIGHSPEEDMODE	Zrychlený režim
SF_CONDSTOPMODE	Podmíněny stop aktivní (pro bloky s M01)
SF_MANUALMODE	Manuální režim ("pomocné ruční pojezdy")
SF_BLOCKBYBLOCK	Režim blok po bloku
SF_LS	Limitní spínač
SF_SLS	Softwarový limitní spínač
SF_BACKWARD	Couvání
SF_STOPPOSCHANGED	Bude nastaven, když při stopu programu bylo ručně popojeto

8.4 Speciální prvky rozhraní

8.4.1 Nastavení limitu rychlosti

Nastavení maximálního limitu rychlosti (platí i pro parabolické rampy) z PLC programu. Limit se uplatní v automatických režimech a v pomocných ručních pojezdech. Nastavení limitu pro pomocné ruční pojezdy se provede pomocí **REQ_EXT_FEED_AUTMAN** a **EXT_FEED_AUTMAN** (viz kapitolu „pomocné ruční pojezdy“ v „Návodu k programování PLC“.)

buňka	typ	význam
LIMIT_FEED_REQ	bit	1 = Požadavek na aktivaci limitu pracovní rychlosti.
LIMIT_FEED_G00_REQ	bit	1 = Požadavek na aktivaci limitu rychlosti pro rychloposuv.
LIMIT_FEED	DWORD	Nastavení limitu pracovní rychlosti. Nastavuje se 1/64000-tisících μm/ms, to znamená, že když potřebujeme nastavit rychlost v μm/ms nebo v mm/s, stačí nastavit horní word buňky.
LIMIT_FEED_G00	DWORD	Nastavení limitu rychlosti pro rychloposuv. Nastavuje se 1/64000-tisících μm/ms, to znamená, že když potřebujeme nastavit rychlost v μm/ms nebo v mm/s, stačí nastavit horní word buňky.

Příklad:
Zadání limitu rychlosti pro pracovní posun na 1 m/min
1 m/min = 1000 mm/min = 1000/60 mm/s = 17 mm/s

```
LOD   CNST.17
STO   WORD.LIMIT_FEED+2       ;naplní 17 mm/s (horní word)
FL    1,LIMIT_FEED_REQ
```

8.4.2 Čas a datum pro PLC program

V PLC programu jsou zpřístupněny struktury pro zjišťování času a datumu (požité formáty proměnných jsou standardní):

```
      SYST_TIME_INFO          ....Systémový čas
      SYST_DATE_INFO          ....Systémový datum
      UTC_TIME_INFO           ....Systémový čas UTC
      UTC_DATE_INFO           ....Systémový datum UTC
      FILETIME_INFO           ....Systémový čas UTC ve formátu FILETIME

      (struc, čtení)

TIME_INFOS      struc
  TIME_MIN      DB      0     ;Minuty
  TIME_HOUR     DB      0     ;Hodiny
  TIME_HUND     DB      0     ;Setiny sekundy
  TIME_SEC      DB      0     ;Sekundy
TIME_INFOS      ends

DATE_INFOS      struc
  DATE_YEAR     DW      0     ;Rok
  DATE_DAY      DB      0     ;Den
  DATE_MON      DB      0     ;Mesic
DATE_INFOS      ends

FILETIME_INFOS  struc
  LOW_FILETIME  DD      0     ;Formát FILETIME
  HIGH_FILETIME DD      0
FILETIME_INFOS  ends
```

Příklad:

```
      LOD   BYTE.SYST_TIME_INFO.TIME_MIN        ;načte minuty
      LOD   BYTE.SYST_TIME_INFO.TIME_HOUR       ;načte hodiny
      LOD   BYTE.SYST_TIME_INFO.TIME_SEC        ;načte sekundy
      LOD   WORD.SYST_DATE_INFO.DATE_YEAR       ;načte rok
      LOD   BYTE.SYST_DATE_INFO.DATE_DAY        ;načte den
      LOD   BYTE.SYST_DATE_INFO.DATE_MON        ;načte měsíc
      LOD   BYTE.UTC_TIME_INFO.TIME_HOUR        ;načte hodiny UTC
      LOD   QWRD.FILETIME_INFO                  ;načte formát FILETIME
```

8.4.3 Točítko s displejem

K systému je možnost připojení dvou externích panýlků s ručním ovládáním, s točítkem a s displejem.

Tlačítka z panýlku točítka se snímají v PLC programu pomocí buněk `PTLTOC_P1`, `PTLTOC_P1_2` pro první kanál, nebo pomocí buněk `PTLTOC_P2`, `PTLTOC_P2_2` pro druhý kanál.

Pro ovládání displeje je pro PLC program definována struktura:

```
;Struktura pro tocitko s displejem

TOC_DISPLS  struc
      TOC_ROW           DB    0     ;radek      (0,1,...v pixelech)
      TOC_COLUMN        DB    0     ;sloupec    (0,1,...v pixelech)
      TOC_CHAR1         DB    0     ;1.znak     ASCII hodnota, 0FFh=přeskok
      TOC_CHAR2         DB    0     ;2.znak
      TOC_CHAR3         DB    0     ;3.znak
      TOC_CHAR4         DB    0     ;4.znak
      TOC_SIZE          DB    0     ;velikost
      TOC_INV           DB    0     ;inverse
TOC_DISPLS  ends
```

Na displej je možno zapsat najednou 4 znaky na pozici danou zvoleným řádkem a zvoleným sloupcem. Znaky se zadávají ASCII hodnotou. Hodnota 0FFh je transparentní znak (výpis pozici přeskočí), to znamená že na dané pozici zůstane svítit minulý zobrazený znak

Pole pro ovládání displeje pro 1.kanál je **`DISPTOC1`** (8 byte) a pole pro ovládání displeje pro 2.kanál je **`DISPTOC2.`** Pro řízení rozpracovanosti slouží bity **`BUSY_DISPTOC1`** a **`BUSY_DISPTOC2.`** Bity nastavuje PLC program na hodnotu log.1 v době, když data v polích `DISPTOC1` a `DISPTOC2` jsou ještě neplatná a způsobí zákaz obsluhy displeje.

Příklad:
Zobrazení znaků Abc na pozici [0,16]:

```
      FL    1, BUSY_DISPTOC1        ;zákaz obsluhy
      LOD   CNST.0                  ;0 pixelů pro řádky
      STO   BYTE.DISPTOC1.TOC_ROW
      LOD   CNST.16                 ;16 pixelů pro sloupce
      STO   BYTE.DISPTOC1.TOC_COLUMN
      LOD   CNST.'A'                ;A
      STO   BYTE.DISPTOC1.TOC_CHAR1
      LOD   CNST.'b'                ;b
      STO   BYTE.DISPTOC1.TOC_CHAR2
      LOD   CNST.'c'                ;c
      STO   BYTE.DISPTOC1.TOC_CHAR3
      LOD   CNST.0FFh               ;transparent
      STO   BYTE.DISPTOC1.TOC_CHAR4
      FL    0, BUSY_DISPTOC1        ;povolení obsluhy
```

8.4.4 Plynulá jízda

PLC program může testovat, zda se právě jede plynule (G23 aktivní) a také kolik času zbývá do konce plynulého úseku bloků.

G23ACT ....plynulá jízda

(bit, čtení)

TIME_DIST ....aktualizovaný čas do konce plynulého úseku [ms]

(DWORD, čtení)

Hodnota času v buňce **TIME_DIST** je platná jen pokud je nastaven bit **G23ACT** a zmenšuje se reálně jak se blíží konec plynulé jízdy bloků. Ten nastane, když je uhel mezi bloky příliš velký, když je programován rychloposuv, když je nepohybový blok nebo když blok obsahuje technologii která vyžaduje zrušení plynulosti.

8.4.5 Zálohovaná PLC paměť

PLC program má k dispozici zálohovanou paměťovou oblast **PLC_MEM_BACKUP** o velikosti 10000 bajtů (bližší popis viz „Sdílená a zálohovaná paměť“). Při regulérním ukončení systému se automaticky celá oblast uloží na disk. Při startu systému se oblast automaticky načte. PLC program má možnost si vyžádat okamžité uložení oblasti na disk.

REQ_BACKUP_MEM ....žádost o uložení zálohované oblasti

(bit, čtení/zápis)

PLC program nastavením hodnoty log.1 do bitu **REQ_BACKUP_MEM** způsobí okamžité uložení celé paměťové oblasti `PLC_MEM_BACKUP[10000]` na disk. Po uložení zálohované oblasti na disk, systém bit `REQ_BACKUP_MEM` vynuluje.

Příklad:
Uložení oblasti PLC_MEM_BACKUP v mechanizmu s čekáním na provedení akce.

```
FL    1, REQ_BACKUP_MEM       ;žádost o uložení na disk
EX
LDR   REQ_BACKUP_MEM          ;čeká na uložení
EX1
```

8.4.6 Zrychlení a rychlosti

PLC program má možnost zjistit hodnoty zrychlení a rychlostí z konfigurace NC os v souboru typu „ChannelConfig"

Název	Typ	Popis
dwAxNCAcceleration	pole 16xDWORD	Lineární zrychlení posuvu [1/64000 um/T^2] (používá se při referenci, pro synchronní osy, pro ruční pojezdy)
dwAxNCAccelerationRT	pole 16xDWORD	Lineární zrychlení pro rychloposuv [1/64000 um/T^2]
dwAxNCAccelerationEmergency	pole 16xDWORD	Lineární zrychlení pro najetí na limit a „Emergency Stop" [1/64000 um/T^2]
rAxNCParabAccel	pole 16xREAL	Parabolické zrychlení (používá se jen pro synchronní osy)
rAxNCParabAccelRT	pole 16xREAL	Parabolické zrychlení pro rychloposuv
rAxNCParabAccelEmergency	pole 16xREAL	Parabolické zrychlení pro najetí na limit a „Emergency Stop"
dwAxNCLinearAccel	pole 16xDWORD	Lineární zrychlení pro lineární část parabolických ramp (používá se jen pro synchronní osy) [1/64000 um/T^2]
dwAxNCLinearAccelRT	pole 16xDWORD	Lineární zrychlení pro parabolické rampy a rychloposuv [1/64000 um/T^2]
dwAxNCLinearAccelEmergency	pole 16xDWORD	Lineární zrychlení pro parabolické rampy pro najetí na limit a „Emergency Stop"
dwAxNcFeed	pole 16xDWORD	Rychlost [1/64000 um/T] (používá se pro synchronní osy, pro ruční pojezdy)
dwAxNcFeedRT	pole 16xDWORD	Maximální rychlost [1/64000 um/T] (používá se pro ruční pojezdy)
rAxGParabAccel	REAL	Parabolické zrychlení pro geometrické osy. Standardně se zadává v přípravě bloku NC programu. PLC program má možnost zrychlení zadat, když je nastaven bit REQ_PARABACCEL na hodnotu 1. [m/s^3 * 65.536]
dwAxGLinearAccel	DWORD	Lineární zrychlení pro lineární část parabolických ramp pro geometrické osy. Standardně se zadává v přípravě bloku NC programu. PLC program má možnost zrychlení zadat, když je nastaven bit REQ_LINEARACCEL na hodnotu 1. [1/64000 um/T^2]

8.4.7 Licence pro PLC program

PLC program má možnost zjistit stav licencí určených pro volné využití v PLC programu. Licence jsou vázány na hardwarový klíč. Pro zjišťování licencí slouží instrukce **LDR_LIC.**

instrukce	**LDR_LIC**

funkce **čtení stavu licence do RLO**

syntax **LDR_LIC** **feature**

parameter **„feature"** **symbolická konstanta pro určení licence**

Instrukce **LDR_LIC** načte stav některé PLC licence do RLO registru. Instrukce může také zjistit stav připojení hardwarového klíče. Pokud instrukce vrátí hodnotu log.1, tak je licence pro danou vlastnost pořádku.

Licence pro dané vlastnosti PLC programu, jsou určeny symbolickými konstantami:

Licence	Význam
HwKeyConnected	Stav připojení hardwarového klíče
FeaturePlc0	Licence pro uvolnění vlastnosti PLC0
FeaturePlc1	Licence pro uvolnění vlastnosti PLC1
FeaturePlc2	Licence pro uvolnění vlastnosti PLC2
FeaturePlc3	Licence pro uvolnění vlastnosti PLC3
FeaturePlc4	Licence pro uvolnění vlastnosti PLC4
FeaturePlc5	Licence pro uvolnění vlastnosti PLC5
FeaturePlc6	Licence pro uvolnění vlastnosti PLC6
FeaturePlc7	Licence pro uvolnění vlastnosti PLC7
FeaturePlc8	Licence pro uvolnění vlastnosti PLC8
FeaturePlc9	Licence pro uvolnění vlastnosti PLC9

Příklad:

```
      LDR_LIC     FeaturePlc1       ;načíst stav licence PLC1
      JL0         LBL_DISABLE_PLC1  ;skok, když není platná licence
```

12

12. POPIS ŘÍZENÍ ROTAČNÍCH OS A VŘETEN

12.1 Definice pojmů, adresace

Kapitola se zaobírá podrobněji problematikou nastavování parametrů regulátorů, přepínání rotačních os na vřetena a obráceně a řízením vřeten.

Systém může řídit souvisle 6 os . Analogových nebo CAN-BUSouvých výstupů pro řízení všech os, vřeten a ostatních regulátorů může být maximálně 16. Analogové výstupy a snímání čidel IRC jsou na jednotkách SU05**.** Řízení vřeten je plně ve správě programovatelného interfejsu stroje.

Podle druhu řízení rozlišujeme vřetena:

- Řízené vřeteno analogovým napětím bez snímání otáček čidlem IRC nebo se snímáním otáček čidlem IRC vlastním PLC programem. V dalším textu budeme nazývat **OBYČEJNÉ VŘETENO**.
Na obyčejné vřeteno se nedá využít funkce otáčkového posuvu (G95), závitování nožem (G33), konstantní řezná rychlost (G96,G97) a polohování vřetena.

- Řízené vřeteno analogovým napětím se snímáním otáček čidlem IRC systémovými prostředky s možností polohování a přechodu na rotační souřadnici s možností otáčkového posuvu (G95), závitování (G33) a konstantní řeznou rychlostí (G96,G97). V dalším textu budeme nazývat **ROTAČNÍ OSA**.

Obyčejných vřeten může být větší počet, pokud se nepřesáhne maximální počet analogových výstupů (16). Rotační osy jsou jako normální souřadnice, jenom se dráha zadává ve stupních. Rotační osa se může PLC programem přepínat z rotační souřadnice na vřeteno a nazpět. Je důležité pravidlo, že se může přepnout v daný okamžik na vřeteno **jenom jedna** rotační osa. Přepnutím rotační osy na vřeteno předá PLC program systémovému programu zprávu. Podle rotační osy, která byla naposled přepnuta na vřeteno, řídí systémový program například otáčkový posuv, závitování, polohování a konstantní řeznou rychlost.

Pokud má systém dvě rotační souřadnice a obě jsou přepnuty na vřeteno, tak PLC program musí učit, které vřeteno budě hlavní (řídící).

Obyčejná vřetena možno adresově umístit libovolně na volné adresové pozice analogových výstupů. Mohou se umístit i na adresové pozice, které implicitně patří nějaké souřadnici, pokud tato osa není v dané aplikaci systému použita (například místo šesté osy). Rotační osa přitom může být umístěna jenom na adresném místě souřadnice.

Pro řízení vřeten se používají jednotky souřadnic **SU05**. Jejich popis, přiřazení kanálů, adresace jednotek, nastavení strojních konstant a parametrů regulátorů je v kapitole **"Nastavení parametrů servopohonů a jejich řízení PLC programem".**

12.2 Princip řízení "obyčejných vřeten"

V řízení obyčejných vřeten programovatelný interfejs stroje jenom vysílá hodnotu na analogový výstupní port (přes SU05 nebo CAN-BUS).. Vyslání hodnoty vždy vyžaduje dvě akce:

- přiřazení aktivního portu obyčejných vřeten ke konkrétnímu číslu portu (hodnota 1,2,...,16).
- vyslání binární hodnoty na aktivní analogový výstup pro obyčejná vřetena.

instrukce	**ANALOG_PORT**

funkce	**přiřazení kanálu pro analogový výstup**	
syntax	**ANALOG_PORT**	**port**
parametr	**„port"**	**číslo portu**

Přiřazení aktivního kanálu ke konkrétnímu číslu portu se provede instrukcí **ANALOG_PORT** s jedním parametrem "port", kterým je číslo portu v rozsahu 1 až 16. Pokud není použita nová instrukce ANALOG_PORT, platí trvalé přiřazení aktivního portu pro obyčejné vřeteno. Instrukce neovlivní přiřazení rotačních os ke konkrétním portům.
Analogová hodnota může být vyslaná jednotkou SU05 nebo přes CAN-BUS kanál.

Způsob vyslání je určen konfigurací v souboru „Channel0.ChannelConfig“ v elementu `DriveChannel` a atributu `DriveType`:

element **DriveChannel**		**Nastavení výstupního kanálu**	
	atribut **DriveType**	**Typ výstupního kanálu**	
		0	žádný (default)
		1	SU05 v plném provozu
		2	SU05 s vyřazením testů
		3	Kollmorgen ServoStar, řady 400 a 600
		4	Maxon - Epos
		5	TGA 24
		6	Berger-Lahr CPD 17
		7	Control Techniques UniDrive
		8	Control Techniques UniDrive SP (inicializace při startu systému)
		9	Control Techniques UniDrive SP (inicializace z PLC)
		12	TGPower
		17	Telemecanique ATV (Schneider)
		18	Plovoucí RTM – simulace stroje
		19	TGPower trajectory ID1,2 + TGPowerSpeed ID3

instrukce	**ANALOG**

funkce	**vysílání hodnoty na analogový výstup**	
syntax	**ANALOG**	**val**
	ANALOG	
parametr	**„val"**	**vysílaná hodnota**

Vyslání 16.bitové binární hodnoty "val" v doplňkovém kódu na aktivní analogový výstup se provede instrukcí **ANALOG** s jedním parametrem, kterým je žádané napětí. Pokud instrukce ANALOG nemá žádný parametr, vyšle se hodnota z datového "DR" registru.

Maximální hodnota výstupu …. +/-32766 (7FFFh)
Pokud je parametr větší než maximální napětí, bude na tuto hodnotu omezen při vyslání.

Pokud používáme jenom jedno obyčejné vřeteno, je možno instrukci ANALOG_PORT použít jenom v modulu MODULE_INIT.

Zadávání hodnoty otáček prostřednictvím funkce "S" se budeme zaobírat v kapitole "Zadávání otáček vřetena".

Příklad:
Vyslání hodnoty **S** , kterou systém zadá v BCD kódu na obyčejné vřeteno, které je adresově umístěné na čísle portu 6 (místo páté osy).

```
      LDR               ZMSSPI            ;testuje změnový bit S
      JL0               ANA_END           ;skok,není-li změna
      LOD               WORD.PB11         ;BCD hodnota S typu WORD
      BIN                                 ;převod na binární hodnotu
      ANALOG_PORT 6                       ;přiřazení aktivního portu na 6
      ANALOG                              ;vyslání hodnoty z DR na port
ANA_END:
```

12.3 Princip řízení "rotačních os"

Pokud je rotační osa namódovaná jako souřadnice, řídí ji kompletně systém v polohové vazbě. Může být řízena souvisle v interpolaci s jinými osami. PLC program musí zabezpečit, aby obsluha omylem nezadala v programu otáčky pro rotační souřadnici, nesmí je vyslat na analogový port a je vhodné zahlásit chybové hlášení o chybě obsluhy.

Pokud je rotační osa namódovaná jako vřeteno, řídí ji PLC program. Systém ji může řídit jenom v režimu konstantní řezné rychlosti a v režimech závitování a otáčkového posuvu snímá informace z odměřování vřetena. PLC například při vysílání napětí pro rotační osu se může řídit hodnotou některé funkce z povelového bloku (S ... PB11).

PLC program musí zabezpečit, aby obsluha omylem nezadala v programu pohyb pro rotační souřadnici. V tom případě je vhodné zahlásit chybové hlášení o chybě obsluhy.

instrukce	**SPI_AX**

funkce **změna rychlostní vazby vřetena na polohovou vazbu**

syntax **SPI_AX [feed1], [feed2], [angle], [accel], [axis]**

1.parametr	**„feed1"**	**1.rychlost polohovaní**
2.parametr	**„feed2"**	**2.rychlost pro přídavní natočení**
3.parametr	**„angle"**	**velikost přídavného natočení**
4.parametr	**„accel"**	**zrychlení**
5.parametr	**„axis"**	**číslo osy**

Instrukce **SPI_AX** změní rychlostní vazbu vřetena na polohovou vazbu příslušné osy. Po zapnutí polohové vazby se rotační souřadnice začne pohybovat dojížděcím posuvem a zastaví se až po dosažení nulového pulsu a přídavného úhlu natočení.
Polohování vřeten pomocí instrukcí SPI_AX se může pro jednotlivé osy časově překrývat, takže může najednou polohovat vícero vřeten. Této vlastnosti se například využije, když má stroj dvě rotační souřadnice.

O metodě zapolohování rotační souřadnice rozhoduje v konfiguraci pro danou NC osu atribut `RefMethod`

element **NCAxis**		**konfigurace NC osy**	
	atribut **RefMethod**	**typ reference**	
		5	Reference pro kódovaná pravítka (Heidenhain, Essa)
		9	způsob „rychlý nájezd do reference" pomocí polohovacích jednotek (default)

Volba v konfiguraci, zda při nájezdu testovat referenční spínač.

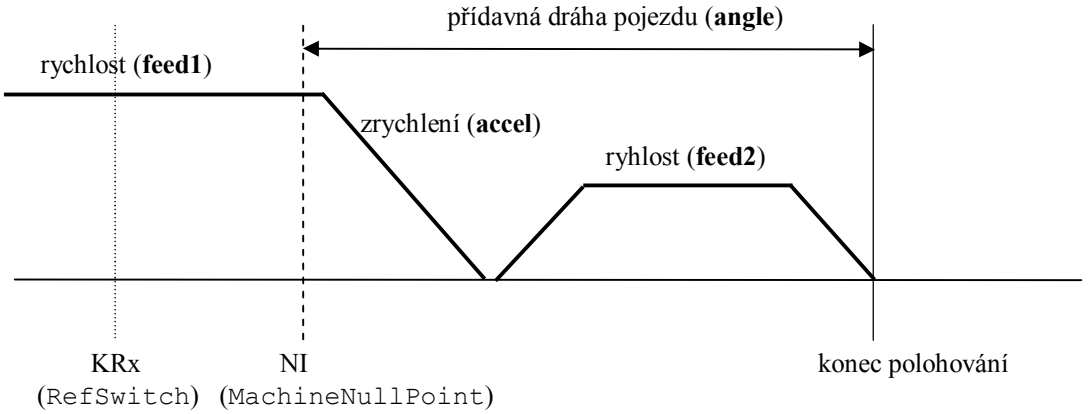
element **NCAxis**		**konfigurace NC osy**	
	atribut **RefSwitch**	**referenční spínač**	
		0	Referenční spínač netestovat
		1	Referenční spínač testovat hranově (default)
		2	Referenční spínač testovat úrovňově

Volba v konfiguraci, zda při nájezdu testovat zpomalovací referenční spínač.

element **NCAxis**		**konfigurace NC osy**	
	atribut **SlowDownSwitch**	**zpomalovací spínač**	
		0	Zpomalovací spínač netestovat (default)
		1	Zpomalovací spínač testovat

Popis parametrů instrukce `SPI_AX`:

`SPI_AX`	parametr	popis
1. parametr	**`feed1`** WORD	Prvním parametrem se může zadat **rychlost dojížděcího posuvu** od okamžiku naprogramování instrukce SPI_AX až po nájezd na nulový puls a po něm následné zpomalení po rampě na nulovou rychlost . Rotační souřadnice přejede nulový puls a zastaví se až tehdy, když rychlost neklesne po rampě na nulovou hodnotu. Do odměřování je přitom započten nulový bod stroje**.** Zadává se jako binární hodnota v doplňkovém kódu v tisícinách stupně za minutu. Záporně zadaný dojížděcí posuv znamená, že souřadnice se bude pohybovat na druhou stranu. Když instrukce žádný parametr nemá („-“ , „NIL“), posuv se zvolí podle atributu: `RefSpeedLow` (v současné verzi jen podle 1.NC osy). V případě, že je požadován test referenčních spínačů**,** systém začne testovat nulový puls souřadnice až po nastavení hodnoty 1 v bitech referenčních spínačů KRx.
2. parametr	**`feed2`** WORD	Druhým nepovinným parametrem se může zadat **rychlost přídavného posuvu** od konce první fáze polohování (zastavení po nájezdu na nulový puls), až po dosažení přídavného úhlu natočení. Zadává se jako binární hodnota v doplňkovém kódu v tisícinách stupně za minutu a může být i záporný. Když instrukce žádný parametr nemá („-“ , „NIL“), posuv se zvolí podle atributu: `RefSpeedLow`. (v současné verzi jen podle 1.NC osy).
3. parametr	**`angle`** DWORD	Třetím nepovinným parametrem je **velikost přídavné dráhy** pojezdu v tisícinách stupně. Když instrukce třetí parametr nemá („-“ , „NIL“), přídavná dráha pojezdu se zvolí podle atributu `PosAfterRef`. Když je hodnota přídavné dráhy nulová, přídavný pojezd se neprovede. Velikost přídavného pojezdu je přírůstek dráhy vztažen k okamžiku průjezdu souřadnice nad nulovým pulsem.
4. parametr	**`accel`** WORD	Čtvrtým nepovinným je **zrychlení** (rampa) pro všechny fáze polohování. Pokud parametr není uveden, použije se hodnota podle atributu `Acceleration` . Hodnota se zadává v [mm/sec^2]..
5. parametr	**`axis`**	Číslo NC osy 1, 2, 3, ... 16.



Znázornění polohování rotační souřadnice

Nájezd na nulový puls trvá určitou dobu, a proto interfejs musí pozastavit vykonávání funkcí do dokončení nájezdu. Pro indikování najetí rotační souřadnice slouží bitové proměnné "**POLOHV_X**, **POLOHV_Y** až **POLOHV_16**." Pokud jsou tyto proměnné nastavené na hodnotu "1", je souřadnice stále ve stavu nájezdu.

Proces polohování může být přerušen:

- vynulováním bitu **POLOHV_x**
- aktivní signál **STOP**
- zakázáno povolení pohybu **MPxPI** nebo **MPxMAN**
- najeto na limitní koncový spínač **KHx0** **KHx1**.

instrukce	**AX_SPI**

funkce	**změna polohové vazby na rychlostní vazbu**	
syntax	**AX_SPI**	**axis**
	AX_SPI	**axis, [spi]**
1.parametr	**„axis"**	**číslo osy**
2.parametr	**„spi"**	**parametry pro vřeteno**

Instrukce **AX_SPI** změní polohovou vazbu na rychlostní vazbu příslušné osy. V prvním parametru „axis" se zadává pořadové číslo NC osy, která má být přepnuta na vřeteno. Druhý parametr „spi" je nepovinný a určuje, zda po přepnutí bude vřeteno jako hlavní, nebo pomocné:

2. parametr **„spi"**	popis
0 (off)	Vypnutí vřetena
1 (main)	Rotační souřadnice bude přepnuta na hlavní vřeteno a na obrazovce systému zobrazena na 1. pozici
2 (assist)	Rotační souřadnice bude přepnuta na pomocné vřeteno a na obrazovce systému bude zobrazena na 2. pozici
S (set)	Rotační souřadnice bude přepnuta na vřeteno. Neovlivní se pořadí na obrazovce a ani to, zda bude vřeteno hlavní. Tyto vlastnosti budou zachovány z použití instrukce SPI_MAIN.

instrukce	**SPI_MAIN**

funkce	**změna hlavního vřetene**	
syntax	**SPI_MAIN**	**axis**
parametr	**„axis"**	**číslo osy**

Instrukce **SPI_MAIN** určí, které z vřeten bude hlavní. V parametru „axis" se zadává pořadové číslo NC osy, která je přepnuta na vřeteno a která má být hlavním vřetenem. Od hlavního vřetena se bude počítat otáčkový posuv G95 a závitování G33 (viz dále).

instrukce	**ANALOG**

funkce	**vysílání hodnoty na analogový výstup**	
syntax	**ANALOG**	**val, axis**
	ANALOG	**val, axis, [spi]**
1.parametr	**„val"**	**vysílaná hodnota**
2.parametr	**„axis"**	**číslo osy**

Vyslání 16.bitové binární hodnoty "val" v doplňkovém kódu na aktivní analogový výstup. V tomto případě se nepoužívá aktivace analogového portu instrukcí ANALOG_PORT, protože ten se pro rotační souřadnici nastaví použitím instrukce **AX_SPI**. Instrukce pro zadání hodnoty pro vřeteno musí v tomto případě mít 2 parametry. Prvním parametrem je binární hodnota otáček "val", které se mají vyslat na rotační souřadnici a druhý parametr "axis" určuje, že hodnota nemá být vyslána na analogový port "obyčejného vřetena", ale na aktivní analogový port rotační souřadnice. Do druhého parametru se může pro názornost zadat jméno rotační souřadnice. Pokud instrukce1. parametr nemá („NIL", „-") na rotační souřadnici se vyšle hodnota z DR registru.
Třetí nepovinný parametr **„**spi**"** se používá pro stroje které mají 2 rotační souřadnice. Hodnota **„**spi**"**, stejně jako u instrukce SPI_AX, nabývá hodnot 0,1 a 2. Pomocí této hodnoty je analogový výstup aktuálně nasměrován buď na hlavní, nebo na pomocné vřeteno (viz. 2 rotační souřadnice).

instrukce	**ANALOG_SPI**

funkce	**přímé vysílání analogového výstupu na osu**	
syntax	**ANALOG_SPI**	**val, axis**
1.parametr	**„val"**	**vysílaná hodnota**
2.parametr	**„axis"**	**číslo osy**

Instrukce **ANALOG_SPI** je alternativní instrukce k instrukci ANALOG, která umožňuje přímé vysílání napětí na osu. Prvním nepovinným parametrem je binární hodnota otáček "val", které se mají vyslat na rotační souřadnici. Pokud instrukce 1. parametr nemá („NIL", „-"), na rotační souřadnici se vyšle hodnota z DR registru. V parametru „axis" se zadává pořadové číslo NC osy
Instrukce byla zavedena také pro stroje, které používají 2 vřetena a pro možnost vysílání napětí na konkrétní vřeteno bez ohledu na to, zda je vřeteno v daném čase hlavní nebo pomocné.

Na rozdíl od instrukce ANALOG s jedním parametrem se tato instrukce neřídí „kanálovou" strukturou analogových výstupů, ale používá „logickou" organizaci podle definice os v systému.

Na rozdíl od instrukce ANALOG se dvěma parametry se analogový výstup neřídí posledním průběhem instrukce AX_SPI, ale je jednoznačně přiřazen podle 2.parametru „axis" .

Příklad:
Přepnutí čtvrté osy "C" z rotační souřadnice na vřeteno.

```
      AX_SPI            4                ;přepnutí osy na vřeteno
      FL                0,HOMING.B3      ;zrušení reference

      LOD               PB11             ;binární hodnota S
      MULB              MIRKA            ;násobení měřítkem
      STO               HODNOTA          ;
      ANALOG            HODNOTA,4        ;vyslání hodnoty na rotač. souř.
```

Příklad:
Vysílání binární hodnoty "S" na "obyčejné vřeteno" na čísle portu 8 a BCD hodnoty "P" na rotační souřadnici "Y" přepnuté na vřeteno.

```
      LDR               ZMSSPI      ;testuje změnový bit S
      JL0               ANA_1       ;skok,  není-li změna
      LOD               PB11        ;BCD hodnota S
      MULB              MIRKA       ;násobení měřítkem
      ANALOG_PORT       8           ;přiřazení aktivního portu na 8
      ANALOG                        ;vyslání hodnoty z DR na port 8

ANA_1: LDR              ZMSSPI      ;testuje změnový bit P
      JL0               ANA_2       ;skok, není-li změna
      LOD               PB04        ;BCD hodnota P
      BIN                           ;převod na binární hodnotu
      MULB              MIRKA2      ;násobení měřítkem
      ANALOG            -,2         ;vyslání hodnoty z DR na port Y
```

12.4 Použití dvou rotačních souřadnic

Systém má možnost používat 2 rotační souřadnice. PLC program řídí pomocí instrukcí AX_SPI, od které rotační souřadnice se bude počítat otáčkový posuv G95 a závitování G33.

Pro práci se dvěma rotačními souřadnicemi se používá dříve popsaných instrukcí:

akce	instrukce	popis
Polohování vřetena	`SPI_AX  d1,d2,u,r`	Zapolohování způsobem pro rychlou referenci, instrukce má 4 parametry. Polohování se může pro jednotlivé osy časově překrývat, takže může najednou polohovat vícero vřeten.
Přepnutí na hlavní vřeteno a určení pozice zobrazování	`AX_SPI x,1`	Přepnutí na hlavní vřeteno. Instrukce musí mít 2 parametry a druhým parametrem je hodnota **1**. Toto vřeteno bude řídit otáčkový posuv a závitování.
Přepnutí na pomocné vřeteno a určení pozice zobrazování	`AX_SPI x,2`	Přepnutí na pomocné vřeteno. Instrukce musí mít 2 parametry a druhým parametrem je hodnota **2**.
Přepnutí na hlavní vřeteno	`SPI_MAIN x`	Samotné přepnutí na hlavní vřeteno.
Vyslání analogového napětí	`ANALOG val,x,1`	Vysílání analogového napětí „`val`“ na hlavní vřeteno. Na kterou osu se bude vysílat záleží na tom, kterou osu určí instrukce AX_SPI jako hlavní vřeteno.
	`ANALOG val,x,2`	Vysílání analogového napětí „`val`“ na pomocné vřeteno. Na kterou osu se bude vysílat záleží na tom, kterou osu určí instrukce AX_SPI jako pomocné vřeteno.
	`ANALOG_SPI val,x`	Přímé vysílání analogového výstupu na osu bez ohledu na to, zda je osa právě přepnuta na hlavní nebo na pomocné vřeteno

Příklad:

Přepínání hlavního a pomocného vřetena:

```
AX_SPI          4, 1        ;4. Osa je hlavní vřeteno
AX_SPI          5, 2
                            ;přepnutí !
AX_SPI          4, 2
AX_SPI          5, 1        ;5. Osa je hlavní vřeteno


ANALOG_SPI      BUN1,4      ;pevné vysílání na 4. osu
ANALOG_SPI      BUN2,5      ;pevné vysílání na 5. osu

ANALOG          BUN1,4,1    ;vysílání se přepíná podle AX_SPI
ANALOG          BUN2,5,2    ;vysílání se přepíná podle AX_SPI
```

12.5 Zadávání otáček vřetena

Systém může zadávat hodnotu otáček pro vřeteno prostřednictvím funkce "S". PLC program si zadanou hodnotu přečte v povelovém bloku "PB11,PB12 (rSPINDLE_OUTPUT_1)". Na hodnotu, kterou systém vyšle do povelového bloku, mají vliv další funkce a parametry konfigurace. Konfigurace pro vřeteno se zadává v elementu **Spindle**:

Číslo vřetene:

element **Spindle**		**konfigurace vřetene**	
	atribut **No**	**číslo vřetene**	
		1	1. vřeteno (default)
		2	2. vřeteno

Počet převodových stupňů.

element **Spindle**		**konfigurace vřetene**	
	atribut **GearCount**	**počet převodových stupňů**	
		0	vřeteno nepoužito (default)
		1,2,..32	počet převodových stupňů

Max. otáčky vřetene. Bude-li programováno více, omezí se na zadanou hodnotu. Pokud je zadaná hodnota omezení nulová, budou maximální otáčky omezeny na maximální otáčky nejvyššího převodového stupně.

element **Spindle**		**konfigurace vřetene**	
	atribut **SpeedLimit**	**maximální otáčky vřetene [ot/min]**	
		0	maximální otáčky podle nejvyššího převodového stupně (default)
		xxx	maximální otáčky [ot/min]

Pro každý převodový stupeň musí být zadány maximální otáčky a odpovídající napětí pro daný stupeň. Hodnoty se zadávají v elementu **Gear**, který je vnoření v elementu **Spindle**:

element **Spindle**			**konfigurace vřetene**	
	element **Gear**		**parametry převodového stupně**	
		atribut **No**	**převodový stupeň**	
			xx	číslo převodového stupně (0,1, .. ,31)
		atribut **MaxSpeed**	**maximální otáčky vřetene [ot/min]**	
			xx	maximální otáčky dosažitelné pro daný převod [ot/min]
		atribut **MaxOutput**	**výstupní hodnota ("napětí"), která odpovídá rMaxSpeed**	
			xx	Výstupní hodnota ("napětí"), která odpovídá maximálním otáčkám převodového stupně 0-7FFFh.

Příklad:
V bloku je programován 2. převodový stupeň M42 a hodnota otáček S...630. Hodnota maximálních otáček pro 2. převodový stupeň je 800 a hodnota odpovídajícího napětí je 8.2V,

```
MaxSpeed = 800
MaxOutput = 7FFFh*0.82 = 26869

       630
      ------ * 26869= 21159 (52A7h)
       800

      PB11 .... 0A7h
      PB12 .... 052h
```

12.6 Řízení převodových stupňů z PLC programu

```
SPINDLE_GEAR_ACT_1      ....Převodový stupeň pro 1.vřeteno

( WORD čtení/zápis )

REQ_EXT_SPINDLE_GEAR    ....Externí řízení převodů z PLC
REQ_MAN_SPINDLE_GEAR    ....Manuální nastavování převodů z PLC

( bity čtení/zápis )
```

Aktuální převodový stupeň pro 1.vřeteno je v buňce **SPINDLE_GEAR_ACT_1** (WORD hodnoty: 1,2,..,32). Pokud se používají maximálně 4 převodové stupně pomocí funkcí M41-M44, zapíše se do buňky SPINDLE_GEAR_ACT_1 příslušný stupeň (1 až 4) přímo ze systému.

V jiných případech, například je větší počet převodových stupňů, musí aktuální převodový stupeň do buňky zapsat PLC program. PLC program musí žádat o externí řízení převodových stupňů nastavením bitů REQ_EXT_SPINDLE_GEAR a REQ_MAN_SPINDLE_GEAR. Aktuální převodový stupeň má vliv na výpočet výstupné hodnoty pro zadané otáčky vřetene.

12.7 Otáčky vřetene řízené pomocí %S

Povelový blok vysílá systém do PLC při startu bloku. Pokud je požadavek řídit otáčky vřetena plynule pomocí "%S" na panelu systému, je nutné v PLC programu převzít hodnotu otáček z buňky VYSOVERS (typ WORD), kde je hodnota průběžně aktualizována.

Příklad:
Vysílání otáček s ohledem na %S:

```
LOD             VYSOVERS        ;otáčky řízené pomocí %S
STO             HODNOTA
ANALOG_PORT     5               ;přiřazení portu
ANALOG          HODNOTA         ;vyslání na obyčejné vřeteno
```

12.8 Zadávání hodnoty pomocí rampy

Pro pohony rotačních os, u kterých nesmí být skoková změna zadávané hodnoty, se používá instrukce **RAMP**.

instrukce	**RAMP**

funkce	**řízení podle rampy**	
syntax	**RAMP**	**vysl, accel**
1.parametr	**„val"**	**vysílaná hodnota**
2.parametr	**„accel"**	**strmost**

Instrukce **RAMP** způsobí postupné zvětšování nebo zmenšování vysílané hodnoty "val" podle zadané strmosti "accel" tak, aby se dosáhla hodnota v DR registru. Po ukončení instrukce je hodnota " val " v DR registru.

Příklad:
Vyslání hodnoty **S** , kterou systém zadá v BCD kódu na obyčejné vřeteno, které je adresově umístěné na čísle portu 6 (místo páté osy) rampou se strmostí 100/20 ms.

```
;program je umístněn v průběžné větvi 20ms
      EQUI              K100,100

      LOD               WORD.PB11         ;BCD hodnota S typu WORD
      BIN                                 ;převod na binární hodnotu
      RAMP              HODN,K100         ;zadání rampy
      ANALOG_PORT       6                 ;přiřazení aktivního portu na 6
      ANALOG            HODN              ;vyslání hodnoty HODN na port
```

Příklad:
Vysílání otáček na rotační osu přepnutou na vřeteno s ohledem na %S a rampou ze strmostí nastavitelnou v PLC konfiguraci.

```
      LOD               VYSOVERS          ;otáčky řízené pomocí % S
      RAMP              HODN,DELTA        ;zadání rampy se strmostí DELTA
      ANALOG            HODN, C           ;vyslání HODN na rot. osu C


;nastaveno v inicializaci PLC (modul MODULE_INIT)
      CNF_GET_INT       DELTA, 'SPEED_ACCELERATION', 100
```

12.9 Důležité systémové proměnné pro řízení vřetene

Dále je uveden seznam důležitých systémových proměnných souvisejících s problematikou vřetene:

název	velikost	popis
PB11, PB12	2xBYTE	Binární hodnota, která reprezentuje výstupní napětí (0 – 7FFFh), které odpovídá programovaným otáčkám vřetene. Hodnota není zkorigována vzhledem k procentu S a neprojeví se ani řízení v průběhu konstantní řezné rychlosti. Reálná forma výstupného napětí je v buňce `rSPINDLE_OUTPUT_1`.
VYSOVERS	WORD	(jen pro čtení) Binární hodnota, která reprezentuje výstupní napětí (0 – 7FFFh), které odpovídá skutečným otáčkám vřetene. Hodnota je zkorigována vzhledem k aktuálnímu stavu procenta S, pokud je tato korekce povolena (není programována G33) . V průběhu konstantní řezné rychlosti je velikost hodnoty řízená od interpolátoru v závislosti na poloze osy X. Reálná forma : `rSPINDLE_PRS_OUTPUT_1`.
REQ_SPEED_PM	DWORD	(jen pro čtení) Požadované otáčky vřetene v tisícinách otáčky za minutu. Hodnota zohledňuje aktuální stav procenta S a konstantní řeznou rychlost. jedná se o výsledné požadované otáčky vřetene. Hodnota buňky je vypočtena z aktuálního stavu `VYSOVERS`, z aktuálního stavu převodového stupně a maximálního napětí pro daný převodový stupeň.
REQ_SPEED_PT	DWORD	(jen pro čtení) Požadované otáčky vřetene v tisícinách stupně za výpočtový takt systému. Hodnota odpovídá buňce `REQ_SPEED_PM`, jen je v jiných jednotkách.
ACT_SPEED_PM **ACT_SPEED2_PM** **AVG_SPEED_PM** **AVG_SPEED2_PM**	DWORD	(jen pro čtení) Aktuální otáčky vřetene v tisícinách otáčky za minutu. Hodnota je vypočtena z odměřovacího čidla na vřetenu. Při výpočtu je zohledněn vnitřní inkrement vřetene. K dispozici také filtrovaná (průměrná) hodnota aktuálních otáček `AVG_SPEED_PM` a také hodnoty pro druhé vřeteno: `ACT_SPEED2_PM` a `AVG_SPEED2_PM`
ACT_SPEED_PT **ACT_SPEED2_PT** **AVG_SPEED_PT** **AVG_SPEED2_PT**	DWORD	(jen pro čtení) Aktuální otáčky vřetene v tisícinách stupně za výpočtový takt systému. Hodnota je vypočtena z odměřovacího čidla na vřetenu.. K dispozici také filtrovaná (průměrná) hodnota aktuálních otáček `AVG_SPEED_PT` a také hodnoty pro druhé vřeteno: `ACT_SPEED2_PT` a `AVG_SPEED2_PT`
SIM_SPEED_PT **(OTACKY_VR)**	DWORD	(pro zápis) Systémová proměnná pro aktuální otáčky vřetene v tisícinách stupně za výpočtový takt systému. Buňka se používá při simulaci otáček vřetene.
REQW_SPEED2_PM	DWORD	(pro zápis) Požadované otáčky pro 2. vřeteno pro výstup na obrazovku.

Využití systémových proměnných :

Kontrola dosažených otáček vřetene:

Pro kontrolu dosažených otáček vřetene je možno v PLC programu porovnávat proměnné REQ_SPEED_PT a ACT_SPEED_PT, nebo REQ_SPEED_PM a ACT_SPEED_PM. Porovnání je nutno provést s určitou tolerancí a při testu je nutno vyloučit dynamické stavy vřetene. Hodnoty REQ_SPEED_PT a REQ_SPEED_PM nejsou zohledněny vzhledem k případné úpravě hodnoty VYSOVERS (například pomocí instrukci RAMP) v PLC programu před vysláním na vřeteno.

Simulace otáček vřetene:

Simulace otáček vřetene se používá v případě, že vřeteno není osazeno snímačem IRC. V tomto případě se v PLC programu ve vhodných situacích přepíše hodnota REQ_SPEED_PT do buňky SIM_SPEED_PT. V PLC programu nesmí být použita instrukce AX_SPI_x.

13

13. NASTAVENÍ PARAMETRŮ SERVOPOHONŮ A JEJICH ŘÍZENÍ PLC PROGRAMEM

13.1 Sady parametrů regulátorů

Systém má softwarovu polohovou, případně rychlostní vazbu. Pomocí změny parametrů je možné modifikovat dynamické parametry servopohonů bez zásahu do hardware systému nebo měničů. Vyskytuje se také požadavek modifikovat dynamické parametry servopohonů v provozu. V tomto případě musí parametry serva modifikovat PLC program. Ovlivňování parametrů se používá například tehdy, když stroj používá mechanickou převodovku. Pokud v přípravných funkcích zadá PLC program povel k řazení na jiný převodový stupeň, může také změnit parametry servopohonu (například kv, omezení skluzu,...). PLC program má možnost změnit **"sadu parametrů regulátrů"**. Sadou parametrů regulátoru pro jednu osu se rozumí souhrn všech parametrů. Po zapnutí systému se implicitně nastaví 1. sada parametrů pro každou osu. Nastavení se provede v čase, když ještě není aktivní softwarová polohová vazba. Pak se provede inicializační modul PLC programu "MODULE_INIT", který může sadu parametrů regulátorů pro některé osy změnit. Až po této akci se uvede do provozu softwarová polohová vazba.

Jednotlivé hodnoty parametrů regulátoru se definují v konfiguraci „Channel0.ChannelConfig“. Můžou tam být nadefinovány 4 sady parametrů pro všechny osy.

instrukce	**REGUL_X, REGUL_Y, ..., REGUL_6**
funkce	**nastavení parametrů regulátoru**
syntax	**REGUL_X sada**
parametr	„**set**“ **sada parametrů regulátoru**

Instrukce **REGUL_X** až **REGUL_6** nastavuje podle zadaného parametru "sada" příslušnou sadu parametrů (1,2,3,4) regulátoru pro danou osu. Nastavení parametrů se doporučuje provádět, je-li osa v klidu a při vypnuté polohové vazbě pro danou osu. Vypínání a zapínání vazby možno řídit pomocí bitových proměnných **"VAZBA_X, VAZBA_Y, ..,VAZBA_6"**.

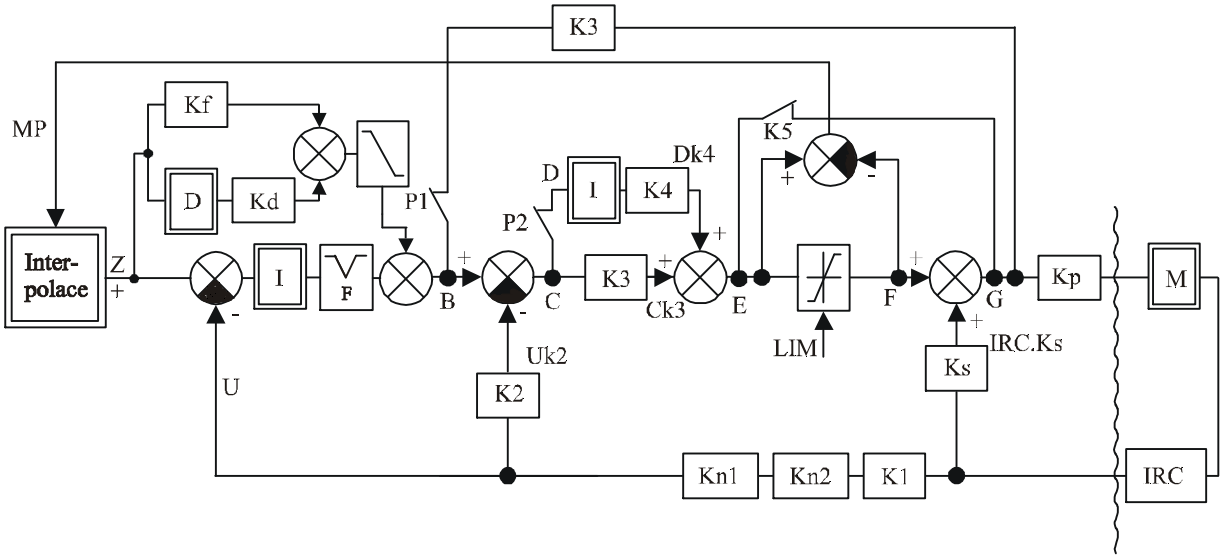
Příklad:
V přípravných funkcích nastavme 2. převodový stupeň pro osu "Y" a změňme parametry regulátoru pro osu "Y" podle 2. sady parametrů a pro 4. osu podle 3. sady parametrů.

```
FL          0,VAZBA_Y         ;vypnutí polohové vazby pro osu Y
FL          0,VAZBA_4         ;vypnutí polohové vazby pro 4. osu Y
FL          1,PREVOD          ;nastartování mechanismu "PREVOD"
EX
LDR         PREVOD            ;čekání na zpřevodování
EX1
REGUL_Y     2                 ;nastavení 2. sady parametrů pro Y
REGUL_X     3                 ;nastavení 3. sady pro 4. osu
FL          1,VAZBA_Y         ;zapnutí polohové vazby pro osu Y
FL          1,VAZBA_4         ;zapnutí polohové vazby pro 4. osu
```

13.2 Souhrn parametrů regulátorů

Souhrn parametrů regulátoru, které je možno ovlivnit:

1. zařazení, nebo vyřazení regulačního obvodu rychlosti (skluzu).
2. nastavení proporcionálního zesílení polohové servosmyčky
3. zesílení zpětné vazby v rychlostní smyčce
4. proporcionální zesílení v rychlostní smyčce a v případě vyřazené rychlostní smyčky, citlivost regulační odchylky polohy
5. integrační konstanta regulátoru v rychlostní smyčce
6. zařazení nebo vyřazení integrační složky regulátoru
7. omezení skluzu pro regulační obvod skluzu
8. snímání rychlosti pro regulační obvod skluzu
9. nastavení feedforwardu



Celkové schéma servosmyčky:

Význam položek:

```
Z     zadán přírůstek
B     diference - odchylka polohy - zadaná rychlost
F     skluz
MP    povolení pohybu
P1    vyřazení rychlostní vazby
P2    vyřazení I-regulátoru
LIM   zadaná hodnota omezení skluzu
K1    konstanta odměřování
K2    zesílení zpětné vazby rychlostní smyčky
K3    proporcionální zesílení
K4    integrační konstanta
K5    řazení regulace s omezeným skluzem
Ks    konstanta snímání rychlosti
Kn    blok nelineárních korekcí
F     Filtr pro frekvenční zádrž
Kf    Proporcionální složka feedforwardu
Kd    Derivační složka feedforwardu
```

Konstanta odměřování **K1** se nastavuje atributy `MeasConstNumerator` a `MeasConstDenominator`: v elementu `Servo`:

element **Servo**		**konfigurace servosmyčky**	
	atribut **MeasConstNumerator**	**konstanta odměřování - čitatel**	
		xx	Celočíselná hodnota čitatele
	atribut **MeasConstDenominator**	**konstanta odměřování - jmenovatel**	
		xx	Celočíselná hodnota jmenovatele

Pokud je požita interní softwarová polohová servosmyčka, tak platí:

$$\text{počet pulzů odměřování} * \frac{\text{MeasConstNumerator}}{\text{MeasConstDenominator}} = \text{míra v mikrometrech}$$

Pokud jsou použity externí polohové servosmyčky přímo v pohonech (např. pro CAN-BUS trajectory), tak platí:

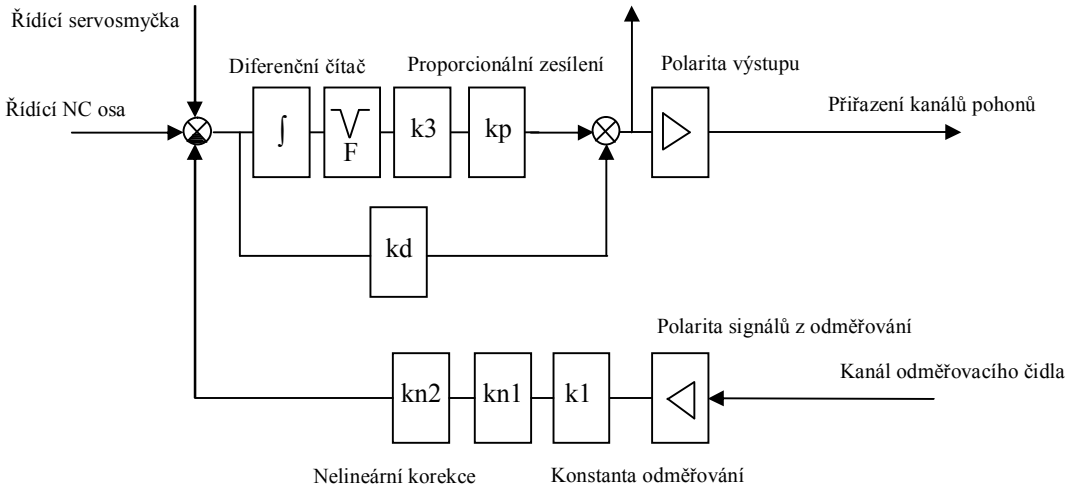
$$\text{míra v mikrometrech} * 2^{16} * \frac{\text{MeasConstNumerator}}{\text{MeasConstDenominator}} = \text{počet inkrementů pohonu}$$

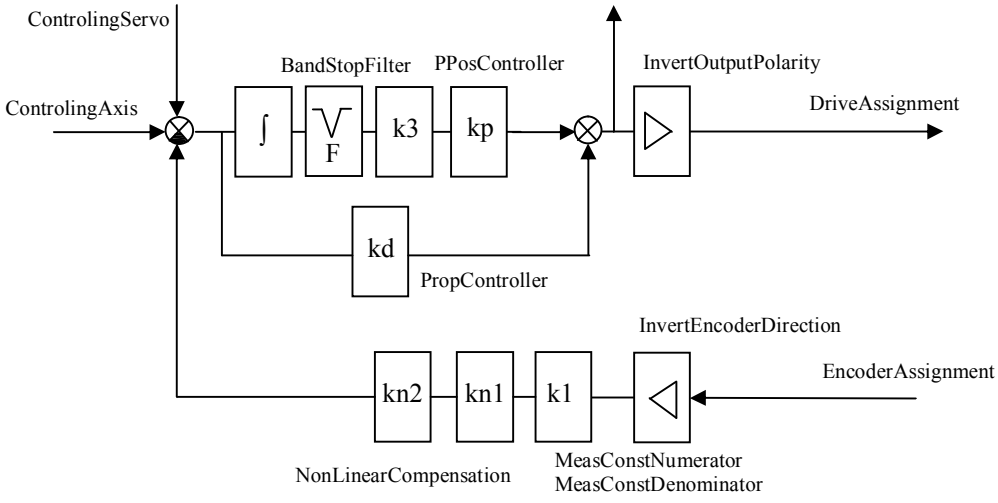
13.3 Zařazení nebo vyřazení regulačního obvodu rychlosti (skluzu) "P1"

Konfigurace pro řazení rychlostní servosmyčky pro danou sadu:

element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **ServoParamP1**	**parametr P1 daného serva**	
			0	rychlostní regulační smyčka je vypnuta (default)
			1	rychlostní regulační smyčka je zapnuta

Pro pohony, které mají vlastní regulační obvod rychlosti s nastavitelnou integrační vazbou se přepínač **P1** vynuluje a tím se softwarový regulační obvod vyřadí. Blokové schéma servosmyčky se zmodifikuje podle obrázku.





13.4 Řazení I-regulátoru "P2

Konfigurace pro řazení integrační složky rychlostní vazby pro danou sadu:

element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **ServoParamP2**	**parametr P2 daného serva**	
			0	integrační složka rychlostní vazby je vypnuta (default)
			1	integrační složka rychlostní vazby je zapnuta trvale
			2	integrační složka rychlostní vazby je zapnuta na dojíždění

V případě zařazeného regulačního obvodu rychlosti (P1=1), je možno pomocí přepínače **P2** zařadit integrační složku. Když P2=0, je integrační složka vyřazena. Když P2=1, je integrační složka zařazena s integrační konstantou **K4** - viz dále.

Často je potřeba u pohonů, které nemají vlastní integrační složku, zařadit softwarový integrál jen pro "dotažení polohy" podle zadané tolerance. Tento **dojížděcí integrál** se zařadí při P2=2 a jeho účinek začne při konci interpolace a ukončí se dosažením zadané tolerance polohy.

13.5 Nastavení zesílení zpětné vazby rychlostní smyčky "K2"

V případě zařazeného regulačního obvodu rychlosti (P1=1) je nutné nastavit zesílení zpětné vazby rychlostní smyčky. Jedná se o softwarovou náhradu tachodynama. Hodnota K2 má podstatný vliv na dynamiku servosmyčky a je nepřímo úměrná parametru **Kv**. To znamená, že čím je větší konstanta K2, tím je větší časová konstanta servopohonu (menší Kv).

Vzhledem k nutnosti provádět výpočty servosmyčky v rychlém časovém rastru, nenastavují se hodnoty konstant přesně. Nastavuje se jenom počet rotací vpravo nebo vlevo, které je nutno provést se zadanou hodnotou. Rotace vlevo zvětší hodnotu konsatnty. Na určení počtu rotací slouží dvouciferný kód :

00	žádná rotace (jednotkový přenos) (00=50)		
01	1 rotace vlevo (*2)	51	1 rotace vpravo (/2)
02	2 rotace vlevo (*4)	52	2 rotace vpravo (/4)
.....			
16	16 rotací vlevo	66	16 rotací vpravo

element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **ServoParamK2**	**parametr K2 daného serva – zesílení rychlostního regulátoru**	
			0	jednotkový přenos rychlostního regulátoru (default)
			01,02,..,16	stupeň zesílení pro přenos rychlostního regulátoru
			51,52,..,66	stupeň zeslabení pro přenos rychlostního regulátoru

13.6 Nastavení proporcionálního zesílení hrubě "K3"

Proporcionální zesílení polohové servosmyčky se nastavuje „hrubě“ pomocí parametru „`ServoParamK3`“ a „jemně“ pomocí parametru „`PPosController`“. Toto nastavení ovlivňuje parametr **Kv** servosmyčky.

Vzhledem k nutnosti provádět výpočty servosmyčky v rychlém časovém rastru, nenastavují se hodnoty konstant přesně. Nastavuje se jenom počet rotací vpravo nebo vlevo, které je nutno provést se zadanou hodnotou. Rotace vlevo zvětší hodnotu konstanty. Na určení počtu rotací slouží dvouciferný kód:

00	žádná rotace (jednotkový přenos) (00=50)		
01	1 rotace vlevo (*2)	51	1 rotace vpravo (/2)
02	2 rotace vlevo (*4)	52	2 rotace vpravo (/4)
.....			
16	16 rotací vlevo	66	16 rotací vpravo

element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **ServoParamK3**	**parametr K3 daného serva – proporcionální zesílení hrubě**	
			0	jednotkový přenos proporcionálního regulátoru (default)
			01,02,..,16	stupeň zesílení pro polohovou servosmyčku
			51,52,..,66	stupeň zeslabení pro polohovou servosmyčku

13.7 Nastavení proporcionálního zesílení jemně

Nastavení se provádí pro jednotlivé souřadnice v každé sadě parametrů regulátorů. Toto nastavení ovlivňuje parametr **Kv** servosmyčky.
Zesílení se nastavuje v setinách (minimální hodnota parametru je 0.01 a maximální je 99.99).

element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **PPosController**	**proporcionální zesílení polohové servosmyčky jemně**	
			1.0	jednotkový přenos polohové servosmyčky (default)
			0.01,..,99.99	zesílení pro přenos polohové servosmyčky

13.8 Nastavení integrační konstanty "K4"

V případě zařazeného regulačního obvodu rychlosti (P1=1) a zařazené integrační složky (P2=1) je nutno nastavit integrační konstantu **K4**.

Nastavuje se jenom počet rotací vpravo nebo vlevo, které je nutno provést se zadanou hodnotou. Rotace vlevo zvětší hodnotu konstanty. Na určení počtu rotací slouží dvouciferný kód:

00	žádná rotace (jednotkový přenos) (00=50)		
01	1 rotace vlevo (*2)	51	1 rotace vpravo (/2)
02	2 rotace vlevo (*4)	52	2 rotace vpravo (/4)
.....			
16	16 rotací vlevo	66	16 rotací vpravo

I v případě nastavení většího počtu rotací vpravo, nedojde k podtečení hodnoty, protože se uchovávají i řády s váhou nižší než 2^0. To znamená, že i při nastavení velmi malé integrační konstanty dojde k naintegrování I regulátoru a ovlivnění serva.

element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **ServoParamK4**	**parametr K4 daného serva – integrační konstanta rychlostní vazby**	
			0	jednotkový přenos integračního regulátoru (default)
			01,02,..,16	stupeň zesílení pro integrační regulátor
			51,52,..,66	stupeň zeslabení pro integrační regulátor

13.9 Proporcionální složka feedforwardu "Kf"

Při vyšších rychlostech obrábění je potřeba kompenzovat regulační odchylku polohové vazby pro dosažení požadované přesnosti obrábění. Snahou je kompenzovat regulační odchylku až na nulovou hodnotu a to i při dynamických stavech stroje. (Parametry jsou aktivní od softwarových verzí sekundárního procesoru 6.020 pro řadu systémů CNC8x9.)

Přenosová funkce feedforwardu (tvar v Laplaceove transformaci) je:

$$F(p) = Ts + pTsTv$$

Ts je časová konstanta polohové servosmyčky, pro kterou platí: $Ts = 1/Kv$
Tv je časová konstanta podřízené rychlostní servosmyčky.

Při ustálené rychlosti obrábění (například při lineárním pohybu a když není změna rychlosti) platí, že proporcionální složka feedforwardu je rovna převrácené hodnotě parametru Kv:

$$\lim F(p) = Ts = 1/Kv \qquad \text{pro } p \to 0$$

Pro lepší zadávání hodnot se nastavuje proporcionální složka feedforwardu v desetinách převrácené hodnoty časové konstanty polohové servosmyčky 1/Ts. Časová konstanta Ts je v sekundách. Při 100 procentním feedforwardu je tato hodnota přímo rovna desetinám parametru Kv. Na nastavení parametru jsou pro každou souřadnici v každé sadě parametrů regulátorů rezervovány 4 dekády ve strojních konstantách.
Například pro souřadnici, která má Kv = 32.4 [1/s] se parametr pro proporcionální složku feedforwardu (při sto-procentním feedforwardu), nastaví na hodnotu 0324.

Aktuální hodnota parametru Kv se na systému zjistí v diagnostické obrazovce pro sledování odchylky dráhy nebo výpočtem:

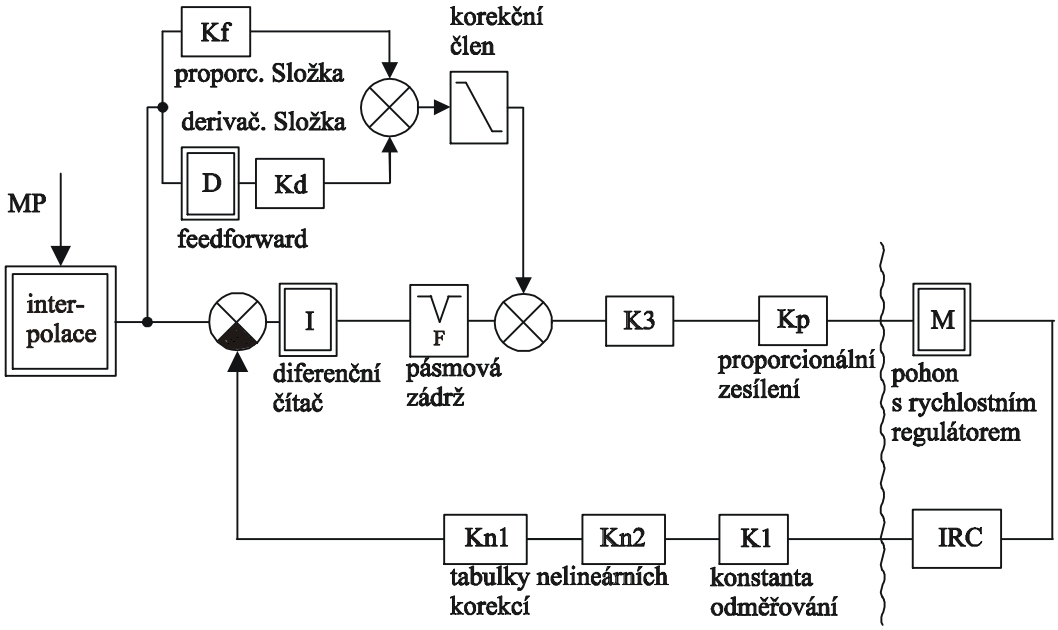
Kv = V / Ep V je skutečná rychlost v mm/s
Ep je regulační odchylka polohy (aktuální stav diferenčního čítače) v mm

(například pro ustálenou rychlost 600mm/min se parametr Kv vypočte: Kv = 10 / Ep)

Pomocí konfigurace je možno řídit řazení feedforwardu. Konstanta slouží na určení, ve kterých situacích má být feedforward aktivní. Nastavení dynamiky a přejezdů pro aktivní feedforward je náročnější a proto obvykle aktivujeme feedforward jen pro pracovní posuv v automatickém režimu.

element **Common**		**obecné parametry konfigurace os**	
	atribut **FeedForward**	**řízení feedforwardu**	
		0	Feedforward je neaktivní
		1	Feedforward je vždy aktivní
		2	Řazení feedforwardu se řídí podle atributů `FeedForwardAut` a `FeedForwardMan`
	atribut **FeedForwardAut**	**způsob řešení feedforwardu pro automatický režim**	
		0	feedforward není zařazen v režimu AUT
		1	feedforward je zařazen v režimu AUT pro pracovní posuv
		2	feedforward je zařazen v režimu AUT pro rychloposuv
		3	feedforward je zařazen v režimu AUT vždy
	atribut **FeedForwardMan**	**způsob řešení feedforwardu pro ruční pojezdy**	
		0	feedforward není zařazen pro ruční pojezdy
		1	feedforward je zařazen pro ruční pojezdy pro pomalý posuv
		2	feedforward je zařazen pro ruční pojezdy pro rychloposuv
		3	feedforward je zařazen pro ruční pojezdy vždy

element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **FeedForwardTs**	**proporcionální složka feedforwardu Kf**	
			0.0	proporcionální složka feedforwardu vyřazena (default)
			xx	proporcionální složka feedforwardu



Blokové schéma servosmyčky se zařazením feedforwardem

13.10 Derivační složka feedforwardu "Kd"

Derivační složka feedforwardu má kompenzovat přejezdy v dynamických stavech stroje. Když systém zadává rychlost po lineární rampě a jedná se o rovnoměrně zrychlený nebo zpomalený pohyb a za předpokladu, že přenos podřízené soustavy rychlostní vazby možno přibližně nahradit soustavou prvního řádu, je větev derivační složky schopna vykompenzovat překmity regulační odchylky polohy.

Přenosová funkce podřízené soustavy rychlostní vazby je:

$$S(p) = Kv / (1 + pTv)$$

Pro lepší zadávání hodnot se nastavuje derivační složka feedforwardu v desetinách převrácené hodnoty časové konstanty rychlostní smyčky 1/Tv. Časová konstanta Tv je v sekundách. Na nastavení parametru jsou pro každou souřadnici v každé sadě parametrů regulátorů rezervovány 4 dekády ve strojních konstantách. Maximální hodnota je 7999.
Například pro souřadnici, která má 1/T = 200 [1/s] se parametr pro derivační složku feedforwardu nastaví na hodnotu 2000.

FILTR DERIVAČNÍ SLOŽKY FEEDFORWARDU

Parametrem se nastavuje filtr pro derivační složku feedforwardu. Filtr je potřeba nastavit pro některé typy pohonů. Jedná se o pohony, které mají úzké a řidší strobování vstupního signálu, takže by nemusely zachytit všechny pulsy z derivační složky feedforwardu.
Každá dekáda parametru R381 nastavuje exponenciální filtr pro derivační složku ve stupních 1 až 9, přitom 1. dekáda nastavuje filtr pro 1. souřadnici, 2. dekáda pro 2. souřadnici apod. Hodnota 0 v příslušné dekádě znamená, že filtr je pro danou souřadnici vyřazen.

element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **FeedForwardFilter**	**filtr derivační složky feedforwardu**	
			0	filtr pro feedforward je vyřazen (default)
			1,2,…,9	filtr derivační složky feedforwardu

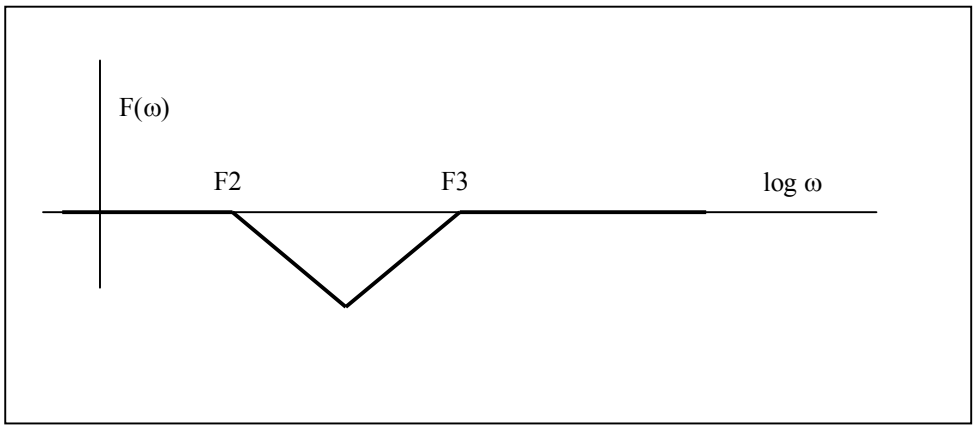
element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **FeedForwardTv**	**derivační složka feedforwardu**	
			0	filtr pro feedforward je vyřazen (default)
			1,2,…,9	filtr derivační složky feedforwardu

13.11 Filtr pro frekvenční pásmovou zádrž

V softwarové servosmyčce může být zařazen filtr pro pásmovou zádrž. Filtr může pomoci potlačit rezonanční kmity stroje. V servosmyčce může být je zařazen filtr s „nekonečnou impulsovou odezvou (IIR) “ druhého řádu navrhnutý jako pásmová zádrž. Pro nastavení filtru slouží 3 parametry, označené jako **F1**, **F2** a **F3**.

- Parametr **F1** představuje proporcionální přenos filtru.
- Parametr **F2** představuje integrační konstantu filtru a je nepřímo úměrná časové konstantě integračního článku. Přesná hodnota je ale závislá na periodě vzorkování (2,8ms, 1,5ms..)
- Parametr **F3** představuje derivační konstantu filtru a je nepřímo úměrná časové konstantě derivačního článku. Přesná hodnota je ale závislá na periodě vzorkování.

Pokles a nárůst frekvenční charakteristiky je 20dB/dek. Náčrtek logaritmické amplitudové frekvenční charakteristiky filtru je:



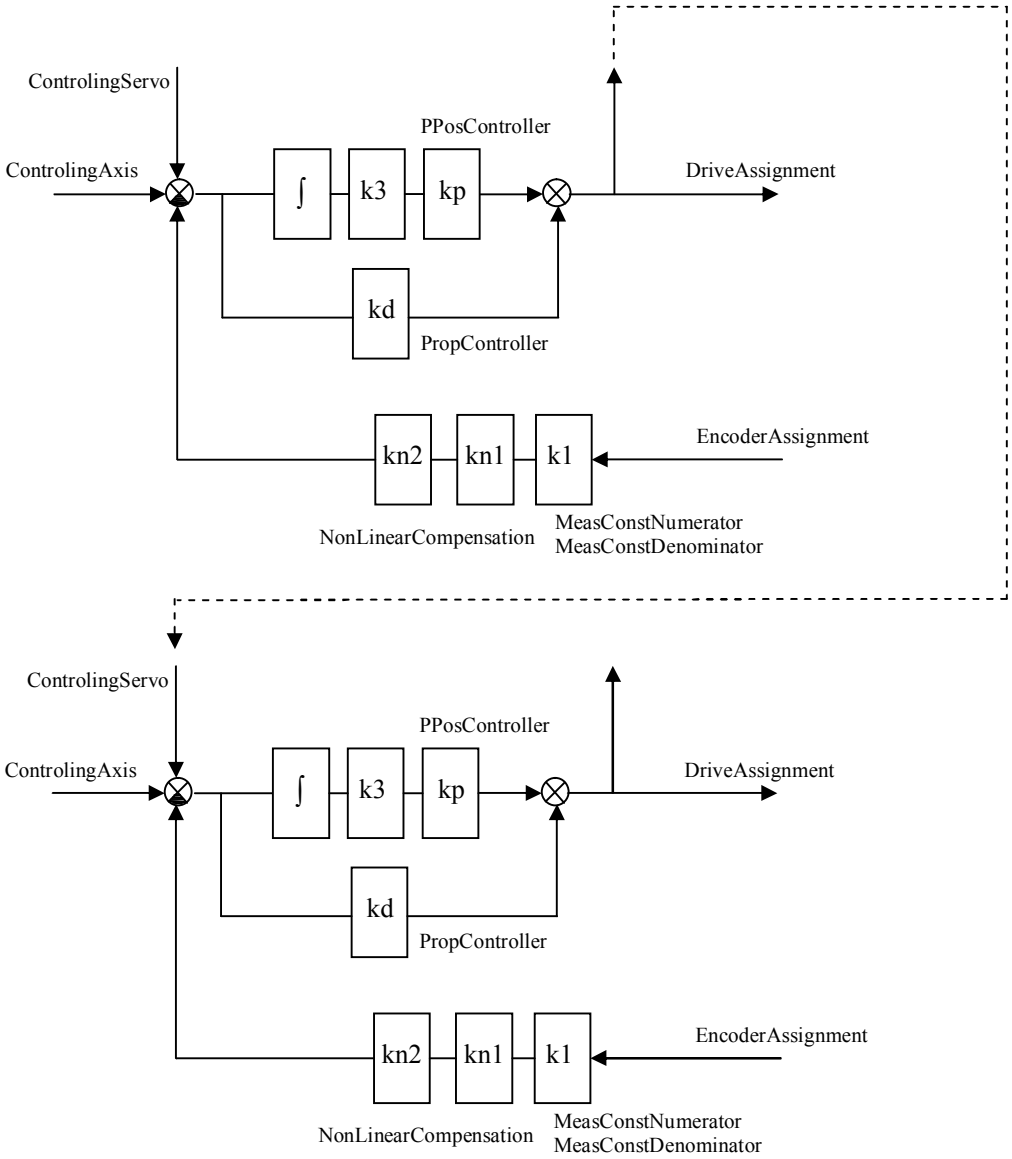
Filtr je zařazen do servosmyčky těsně za diferenční čítač a může měnit své parametry podobně jako se mění parametry servosmyčky v závislosti na platné sadě parametrů regulátorů. Systém má k dispozici 4 pásmové filtry, u kterých je možné nastavit, pro kterou souřadnici a pro kterou sadu parametrů ragulátorů, je filtr aktivní.

element **Servo**			**konfigurace servosmyčky**	
	element **ServoParamSet**		**sada parametrů servosmyčky**	
		atribut **FeedForwardTv**	**derivační složka feedforwardu**	
			0	filtr pro feedforward je vyřazen (default)
			1,2,…,9	filtr derivační složky feedforwardu

element **Servo**				**konfigurace servosmyčky**	
	element **ServoParamSet**			**sada parametrů servosmyčky**	
		element **BandStopFilter**		**parametry pásmové zádrže**	
			atribut **FilterActive**	**řízení pásmové zádrže**	
				0	pásmová zádrž je neaktivní
				1	pásmová zádrž je aktivní
				2	pásmová zádrž je aktivní v době pohybu
			atribut **BandStopFilterF1**	**parametr F1 filtru pásmové zádrže**	
				0	proporcionální přenos filtru nulový (default)
				xx	proporcionální přenos
			atribut **BandStopFilterF2**	**parametr F2 filtru pásmové zádrže**	
				0	integrační složka filtru nulová (default)
				xx	integrační konstanta filtru
			atribut **BandStopFilterF3**	**parametr F3 filtru pásmové zádrže**	
				0	derivační složka filtru nulová (default)
				xx	derivační složka filtru

13.12 Kaskádní řazení servosmyček

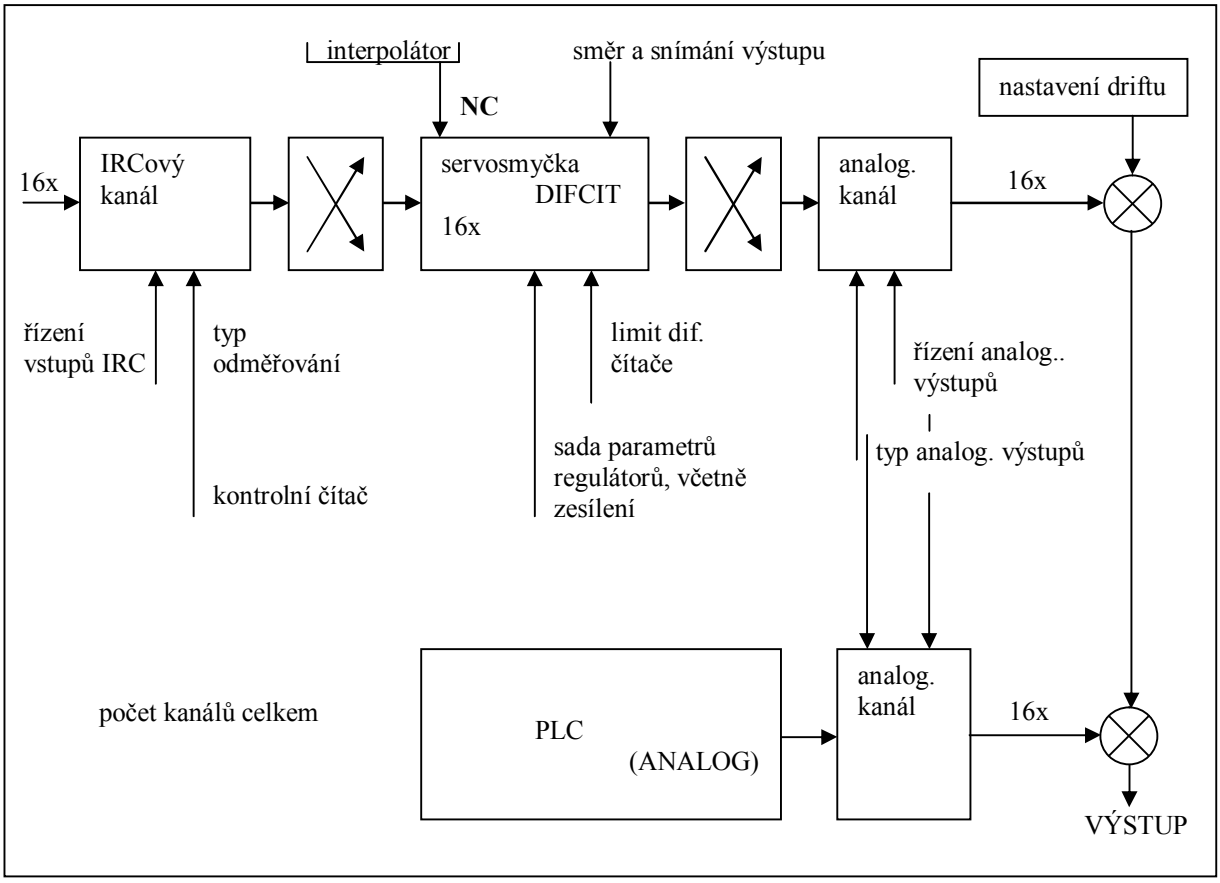
Pro některé speciální účely je možno použít kaskádní řazení servosmyček. Jeden příklad je použití pro lepší implementaci polohové a rychlostní vazby. Druhý příklad je, když se potřebujeme vypořádat s vůlemi souřadnice a máme k dispozici dvojí odměřování (z pohonu a ze suportu). Pro kaskádní napojení souřadnic slouží atribut „`ControlingServo`“.



13.13 Použití jednotek SU05

13.13.1 Všeobecný popis

Jednotka odměřování, výstupu řídícího napětí nebo řídících pulsů pro pohony 4 os. Na čelním panelu má jeden konektor CANNON25 A čtyři konektory CANNON15. Konektor CANNON25 (dole) slouží pro výstup řídícího napětí a řídících pulsů pro pohony os, konektory CANNON15 pro připojení odměřování. Spodní konektor CANNON15 slouží pro připojení odměřování 1. osy (obvykle X). Jednotka nastavuje automaticky napájecí napětí pro snímače odměřování s ohledem na úbytek napětí na napájecím kabelu a hlídá přetržení vodičů odměřování. V případě zjištění chyby vypne napájecí napětí pro snímač. Pro správnou funkci automatického nastavení napětí pro snímač odměřování je vhodné, aby průřezy napájecích vodičů snímačů pro 0V a pro +5V byly přibližně stejné. Kompenzace úbytku funguje pro odběr snímače 0.3A a průřez napájecí žíly 0.5 mm2 do délky kabelu 70m. (V případě zdvojených žil 140m.). Vyhodnocení odměřování umožňuje rychlost až 1000000 inkrementů/sec.



Maximálnímu rozsahu napětí odpovídají binární čísla v doplňkovém kódu z intervalu +/- 7FFFh.

13.13.2 Popis konfigurace pro nastavení jednotek SU05.

Celkový počet kanálů SU05

element **Common**		**obecné parametry konfigurace os**	
	atribut **SU05ChannelCount**	**Celkový počet kanálů SU05**	
		0	jednotky SU05 nejsou použity (default)
		1,2,..	počet kanálů na jednotkách SU05

Nastavení limitů pro hlídání diferenčních čítačů

Hodnota 0 nebo znaménko minus u příslušné konstanty odstaví kontrolu hlídání. Při přetečení diferenčního čítače přes nastavený limit se diferenční čítač vynuluje, shodí se reference, zastaví se pohyb a ohlásí se chyba. PLC program má možnost zjistit číslo chyby v buňce BZH13

element **Servo**		**konfigurace servosmyčky**	
	atribut **FollowingErrorLimit**	**Limit pro hlídání diferenčních čítačů [mm]**	
		0	limit pro hlídání diferenčních čítačů je zrušen
		1.0	limit pro hlídání diferenčních čítačů 1 mm (default)
		xx	limit pro hlídání diferenčních čítačů je nastaven
		-xx	limit pro hlídání diferenčních čítačů je zrušen

Nastavení zóny kontrolního čítače IRCů

Zóna kontrolního čítače IRC je počet pulsů mezi dvěma nulovými pulzy (po vynásobení 4x). Při chybě kontrolního čítače se diferenční čítač vynuluje, shodí se reference, zastaví se pohyb a ohlásí se chyba. PLC program má možnost zjistit číslo chyby v buňce BZH13.

element **Servo**		**konfigurace servosmyčky**	
	atributy **CheckCounter1** **CheckCounter2** **CheckCounter3**	**Zóna kontrolního čítače IRC**	
		0	kontrola na kontrolní čítač je neaktivní
		1.0	limit pro hlídání diferenčních čítačů 1 mm (default)
		-xx	kontrola na kontrolní čítač je neaktivní

Překlenutí diferenčních čítačů

Nastavení hodnoty 1 do příslušné dekády způsobí překlenutí diferenčního čítače. Překlenutí znamená, že výstup z interpolátoru (dráha za takt) se vyšle rovnou na výstup servosmyčky. Hodnota z interpolátoru je upravena o proporcionální zesílení příslušné sady parametrů regulátorů. Překlenutí diferenčního čítače se používá například u krokových motorů bez přídavného odměřování

element **Servo**		**konfigurace servosmyčky**	
	atributy **DiffCountKill**	**Překlenutí diferenčních čítačů**	
		0	diferenční čítač aktivní (default)
		1	diferenční čítač je vyřazen

Přímý vstup do diferenčních čítačů z odměřování

Při aktivním překlenutí se hodnota z odměřování, upravená konstantou odměřování, naplní přímo do diferenčního čítače. Tuto hodnotu může dál zpracovávat například PLC program. V tomto případě nesmí být zařazena rychlostní smyčka regulátoru.

element **Servo**		**konfigurace servosmyčky**	
	atributy **DiffCountDirect**	**Přímý vstup do diferenčních čítačů**	
		0	přímý vstup do diferenčního čítače je vyřazen (default)
		1	přímý vstup do diferenčního čítače je aktivní

Směr snímání signálů z odměřování

element **Servo**		**konfigurace servosmyčky**	
	atributy **InvertEncoderDirection**	**Směr snímání signálů z odměřování**	
		0	přímý směr signálů z odměřování (default)
		1	inverze směru signálů z odměřování

Absolutní hodnota výstupu

Když je absolutní hodnota výstupu aktivována, tak systém provede absolutní hodnotu výstupní hodnoty a pak její polaritu upraví pomocí `InvertOutputPolarity`

element **Servo**		**konfigurace servosmyčky**	
	atributy **AbsOutput**	**Absolutní hodnota výstupu**	
		0	přímý směr signálů z odměřování (default)
		1	absolutní hodnota výstupu

Polarita výstupu

element **Servo**		**konfigurace servosmyčky**	
	atributy **InvertOutputPolarity**	**Polarita výstupu**	
		0	přímá hodnota výstupu (default)
		1	invertovaná hodnota výstupu

Řízení IRCových vstupních kanálů

element **EncoderChannel**		**Nastavení kanálu odměřování**	
	atribut **ChannelType**	**Typ odměřovacího kanálu**	
		0	žádné odměřování (default)
		1	odměřování SU05 v plném provozu
		2	odměřování SU05 s vyřazením testů
		...	další typy, které nesouvisí s SU05

Typ odměřování

Nastavuje se typ odměřování, nastavení platí pro `ChannelType=1,2`.

element **EncoderChannel**		**Nastavení kanálu odměřování**	
	atribut **EncoderType**	**Typ odměřování**	
		0	standard (default)
		1	kódovaná pravítka typu Heidenhain, ESSA
		2	odměřování typu NS010
		3	Odměřování typu Limat
		4	Nastavovaná pravítka ESSA
		5	SLM technologie

Zúžení nulového pulsu

element **EncoderChannel**		**Nastavení kanálu odměřování**	
	atribut **LenReferMarks**	**Zúžení nulového pulsu**	
		0	Zúžit nulový pulz u jednotek SU05 (default)
		1	Ponechat nulový pulz u jednotek SU05 v původní šíře

Počátečné napětí pro IRC

V jednotkách SU04 se nastavuje počáteční napětí pro napájení čidel IRC ve voltech.

element **EncoderChannel**		**Nastavení kanálu odměřování**	
	atribut **EncoderBegVoltage**	**Počáteční napětí pro IRC u jednotek SU05 ve voltech**	
		0.0	žádné počáteční napětí pro IRC
		xx	počáteční napětí pro IRC ve voltech

Řízení analogových výstupních kanálů

element **DriveChannel**		**Nastavení výstupního kanálu**	
	atribut **DriveType**	**Typ výstupního kanálu**	
		0	žádný výstup (default)
		1	výstup na SU05 v plném provozu
		2	výstup na SU05 s vyřazením testů
		...	další typy, které nesouvisí s SU05

Nastavení driftu pro analogové kanály

Drift se nastavuje ve voltech a možno zadat hodnotu v rozmezí cca +/- 3.1V).

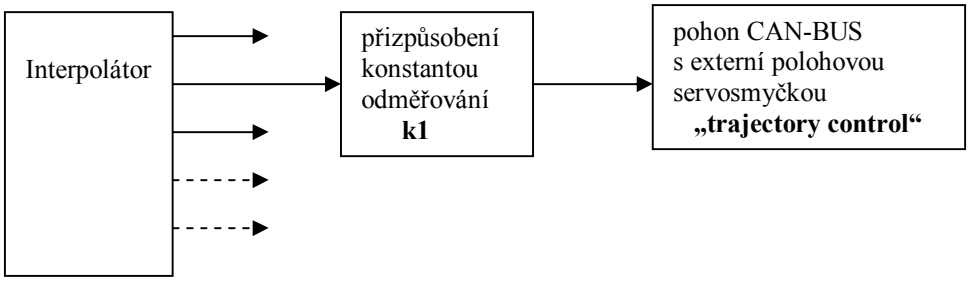
element **DriveChannel**		**Nastavení výstupního kanálu**	
	atribut **SetDrift**	**Drift analogového výstupu [V]**	
		0	drift analogového kanálu 0V (default)
		xx	drift analogového kanálu ve voltech

13.14 Pohony připojené pomocí sběrnice CAN-BUS v režimu „trajectory control“

Pohony se řídí v módu **„trajectory control“**, to znamená, že polohová servosmyčka je uzavřena mimo systém v pohonu. Tím je umožněno dosáhnout lepších dynamických parametrů osy a také jsou menší nároky na CAN-BUSovou komunikaci s pohonem v porovnání s módem „speed control“. Jedná se o digitální připojení pohonu, čím se získá řada výhod. Například u digitálního připojení pohonu nejsou problémy s nastavením driftu.

Při nájezdu do reference CAN-BASová souřadnice se automaticky přemóduje na **„homing control“**, což je vlastně speciální „motion block“. Proto všechny parametry nájezdu do reference, jako jsou rychlost, rozběhová a dojezdová rampa, se nastavují přímo v pohonu. Referenční spínače jsou přivedeny přímo do pohonu.

Pokud by systém měl všechny souřadnice připojené přes CAN-BUS v režimu „trajectory control“, nemusí být v systému osazena jednotka souřadnic SU05. Odměřování pro polohovou servosmyčku získává přímo pohon buď přímo s vlastního resolveru, nebo s externího odměřování přivedeného přímo do pohonu. (Pohon většinou neumí zpracovat odměřování z kódovaných pravítek (HEIDENHAIN, LARM). Pokud je nutné použít referenci podle kódovaných pravítek, tak systém musí obsahovat vlastní odměřování a řízení souřadnice se může provádět v režimu „speed control“.)



Všechny parametry pro nastavení dynamiky, způsobu reference, nastavení rozlišení apod. se nastavují přímo v pohonu (pomocí sériového rozhraní).

CAN-BUSová komunikace je na rychlosti 1MBd. Na jeden kanál může být připojeno maximálně 6 os. Synchronizační povel je vysílán po každé milisekundě. Mapování komunikačních paketů je co nejúspornější, takže do pohonů jsou vysílány po dvojicích sdružené pakety o žádáné absolutní poloze a pohon vysílá do systému paket s polohovou odchylkou (following error), části rozšířeného statusu (manufacturer status) a části základního statusu.

Komunikační pakety obsahují 11-bitové ID, které je složeno ze 7-bitové adresy pohonu a 4-bitového kódu závislém na typu komunikace**. Adresu pohonu** je nutno nastavit předem přímo v pohonu a nastavuje se vzestupně od hodnoty 1 (1,2,3,..). Na pohonech je také nutno nastavit **rychlost komunikace** (1MBd). Schéma kabelu pro připojení pomocí CAN-BUSu je v příloze návodu a má označení **K18.**

13.14.1 Základní konfigurace pro sběrnici CAN-BUS

Nastavení CAN-BUSu pro pohony se provede pomocí konfigurace:

element **CANChannel**		**Nastavení kanálu CAN-BUSu**	
	atribut **No**	**číslo kanálu CAN-BUS**	
		0	pro pohony CAN-BUS
		xx	
	atribut **Active**	**je CAN kanál aktivní ?**	
		0	CAN kanál neaktivní (default)
		1	CAN kanál aktivní
	atribut **PhysicalCanChannel**	**číslo fyzického CAN kanálu**	
		0	pro pohony CAN-BUS
		xx	
	atribut **CanSpeed**	**komunikiační rychlost CAN-busového kanálu [bit/s]**	
		1000000	komunikační rychlost 1 MBd (default)
		500000	komunikační rychlost 0.5 MBd
		250000	komunikační rychlost 0.25 MBd
		125000	komunikační rychlost 0.125 MBd
		100000	komunikační rychlost 0.1 MBd
	atribut **ServicePeriod**	**perioda obsluhy CAN-busového kanálu v mikro sekundách**	
		250	perioda obsluhy CAN-BUSu po ¼ ms (default)
		500	perioda obsluhy CAN-BUSu po ½ ms
		1000	perioda obsluhy CAN-BUSu po 1 ms
	atribut **SyncPeriod**	**perioda posílání SYNCu jako násobek základního taktu**	
		1	perioda vysílání SYNC po 1 ms (default)
		1,2,..,15	perioda vysílání SYNC

13.14.2 Nastavení serv pro CAN-BUS „trajectory control“

Servo, které je řízené pomocí CAN-BUSu, zadává řídicí hodnoty přímo interpolátor. Polohová i rychlostní servosmyčka je uzavřena přímo v pohonu („trajectory control“), proto pro takovou souřadnici neplatí žádné parametry pro nastavení dynamiky servosmyček.

element **Servo**		**konfigurace servosmyčky**	
	atribut **ServoType**	**typ servosmyčky**	
		0	servosmyčka neaktivní (default)
		1	standardní softwarová servosmyčka
		3	Kollmorgen ServoStar, řady 400 a 600
		4	Maxon - Epos
		5	TGA 24
		6	Berger-Lahr CPD 17 nebo Lexium 04
		7	Control Techniques UniDrive
		8	Control Techniques UniDrive SP (inicializace při startu systému)
		9	Control Techniques UniDrive SP (inicializace z PLC)
		10	Berger-Lahr CPD17 + IFX ID4,5
		11	Berger-Lahr CPD17 + IFX ID6
		12	TGPower
		13	Kollmorgen + TGA24 ID4,5
		14	Kollmorgen + TGA24 ID6
		15	TGPower + TGA24 ID4,5
		16	TGPower + TGA24 ID6
		17	Telemecanique ATV (Schneider)
		18	Plovouci RTM - matematický model stroje (CVUT)
		19	TGPowerTrajectory ID1,2 + TGPowerSpeed ID3
		20	Berger-Lahr CPD 17 nebo Lexium 04, aktivace z PLC
		21	Servostar 300, Lexium 15
		22	Sanyo RS1
		23	Lexium 32 (Schneider)
		24	Estun EDC v3.10, Pronet
		25	Altivar ATV71 (Schneider)
		26	Estun EDC v3.11
		27	Gefran XVy

Konstanty odměřování

Konstanty odměřování pro CAN-BUSové osy slouží pro přizpůsobení na požadovaný počet mikrometrů na otáčku motoru. Stejný počet mikrometrů na otáčku musí být také zadán přímo v pohonu. Pohon musí být nastaven na příslušné rozlišení (například 2^{20} pulsů na otáčku pro Kollmorgen).

Pro trajectory mód platí vztah:

$$\text{míra v mikrometrech} * 2^{16} * \frac{\text{MeasConstNumerator}}{\text{MeasConstDenominator}} = \text{počet inkrementů pohonu}$$

Doporučuje se vzorec použít pro jednu otáčku motoru, potom bude platit:

```
M =  požadovaný počet mikrometrů na 1 otáčku mororu
T =  počet pulsů motoru na otáčku (Kollmorgen  2^20, CPD17 2^14)
```

$$M * 2^{16} * \frac{MeasConstNumerator}{MeasConstDenominator} = T$$

13.14.3 Rozhraní pro PLC program

Pro PLC program je zpřístupněna wordová pole CAN_DRIVE_STAT, CAN_DRIVE_MSTAT a CAN_DRIVE_CMD. Každé wordové pole má velkost 16 wordů (jeden word na souřadnici). Ve vordech jsou definovány významové bity, takže PLC program pro práci s jednotlivými bity může využít „složitější adresaci bitů“.

Význam jednotlivých wordových polí:

Název pole	Popis
CAN_DRIVE_STAT	Základní status pohonu (status register)
CAN_DRIVE_MSTAT	Rozšířený status pohonu (manufacturer status register)
CAN_DRIVE_CMD	Řízení z PLC (command)

Význam jednotlivých bitů pro pohony KOLLMORGEN, BERGER-LAHR:

Základní status pohonu - CAN_DRIVE_STAT		
Bit	**Název bitu pro PLC**	**Popis**
bit 0	CAN_AX_READY	Připraveno pro zapnutí (Ready to switch on)
bit 1	CAN_AX_ON	Zapnuto (Switched on)
bit 2	CAN_AX_ENBLD	Uvolněno (Operation enable)
bit 3	CAN_AX_FAULT	Chyba (Fault)
bit 4	CAN_AX_VOLTAGE	Zákaz napěti (Disable voltage)
bit 5	CAN_AX_QSTOP	Rychlý stop inverzně (Quick stop)
bit 6	CAN_AX_BRKD	Zapnutí zakázáno – zabrzděno (Switch on disabled)
bit 7	CAN_AX_WARN	Hlášení (Warning)

Rozšířený status pohonu - CAN_DRIVE_MSTAT		
Bit	**Název bitu pro PLC**	**Popis**
bit 0	CAN_WRN_I2T	Překročen práh I^2t (I^2t threshold exceeded)
bit 1	CAN_WRN_BALLAST	Dosažen plný výkon (Full ballast power reached)
bit 2	CAN_WRN_FOLLOW	Překročena max. polohová odchylka (Following error)
bit 3	CAN_WRN_RESP	Aktivace monitoringu (Response monitoring activated)
bit 4	CAN_WRN_POWER	Chyba fáze (Power suply phase missing)
bit 5	CAN_WRN_LIMIT1	Aktivní limit 1 (Software limit-switch + has been activated)
bit 6	CAN_WRN_LIMIT2	Aktivní limit 2 (Software limit-switch + has been activated)
bit 7	CAN_WRN_MOTION	Špatný posuvný blok (Faulty motion task started)
2. Byte (offset = +1)		
bit 0	CAN_WRN_MOTREF	Nenajeta reference (No reference point set of motion blok)
bit 1	CAN_WRN_PSTOP	Aktivní PSTOP (PSTOP activated)
bit 2	CAN_WRN_NSTOP	Aktivní NSTOP (NSTOP activated)
bit 3	CAN_WRN_DEF	Motor má default hodnoty (Motor default values were loaded)

bit 4	CAN_WRN_BOARD	Chyba karty (Expansion board not functioning correctly)
bit 5	CAN_WRN_PHASE	Fáze motoru (Motor phae)
bit 6.	CAN_WRN_VCT	Chyba VCT (Erroneous VCT entry selected)

Řízení z PLC - CAN_DRIVE_CMD		
Bit	**Název bitu pro PLC**	**Popis**
bit 0	CAN_AX_EN	Příkaz pro uvolnění pohonu (Operation enable)
bit 1	CAN_AX_BRK	Příkaz pro zabrzdění pohonu (Brake)

V případě, že PLC program dá povel pro zabrzdění pohonu, automaticky se současně zruší jeho uvolnění. Když je pohon zabrzděn, tak se neprovede jeho uvolnění, pokud se nejdříve neodbrzdí. Pohon se může nacházet ve 3 stavech:

	CAN_AX_EN	CAN_AX_BRK
pohon zabrzdit	x	1
pohon uvolnit	1	0
pohon neuvolnit	0	0

Význam jednotlivých bitů pro pohony CONTROL TECHNIQUES - UNIDRIVE:

Základní status pohonu - CAN_DRIVE_STAT		
Bit	**Název bitu pro PLC**	**Popis**
bit 0	CAN_UAX_HEALTY	(10.01) Drive healty
bit 1	CAN_UAX_RUN	(10.02) Drive running
bit 2	CAN_UAX_ZERO	(10.03) Zero speed
bit 3	CAN_UAX_RUNBEL	(10.04) Running at or below min speed
bit 4	CAN_UAX_BELOW	(10.05) Below set speed
bit 5	CAN_UAX_AT	(10.06) At speed
bit 6	CAN_UAX_ABOVE	(10.07) Above set speed
bit 7	CAN_UAX_LOAD	(10.08) Load reached

Řízení z PLC - CAN_DRIVE_CMD		
Bit	**Název bitu pro PLC**	**Popis**
bit 0	CAN_UAX_EN	Příkaz pro uvolnění pohonu (6.15)
bit 1	CAN_UAX_SEQ0	Příkaz pro zabrzdění pohonu (6.30)
bit 2	CAN_UAX_SEQ1	(6.31)
bit 3	CAN_UAX_SEQ2	(6.32)
bit 4	CAN_UAX_TRIP	Způsobí chybu pohonu tr52
bit 5	CAN_UAX_SET0	(1.45)
bit 6	CAN_UAX_SET1	(1.46)
bit 7	CAN_UAX_APP1	(18.31)
2. Byte (offset = +1)		
bit 0	CAN_UAX_APP2	(18.32)
bit 1	CAN_UAX_M0	Maska pro bit0 (mask 6.15)
bit 2	CAN_UAX_M1	Maska pro bit1 (mask 6.30)
bit 3	CAN_UAX_M2	Maska pro bit2 (mask 6.31)
bit 4	CAN_UAX_M3	Maska pro bit3 (mask 6.32)
bit 5	CAN_UAX_APP3	(18.33)
bit 6	CAN_UAX_M5	Maska pro bit5 (mask 1.45)
bit 7	CAN_UAX_M6	Maska pro bit6 (mask 1.46)

Příklady:

Uvolnění 2. souřadnice v mechanizmu a test na potvrzení:

```
FL    1,(CAN_DRIVE_CMD+2).CAN_AX_EN            ;povel pro uvolnění
EX
LDR   (CAN_DRIVE_STAT+2).CAN_AX_ENBLD          ;čeká na potvrzení
EX0
```

Zabrzdění 3. souřadnice v mechanizmu a test na potvrzení:

```
FL    0,(CAN_DRIVE_CMD+4).CAN_AX_EN            ;zákaz uvolnění
FL    1,(CAN_DRIVE_CMD+4).CAN_AX_BRK           ;povel pro zabrzdění
EX
LDR   (CAN_DRIVE_STAT+4).CAN_AX_BRKD           ;čeká na potvrzení
EX0
```

13.14.4 Vyslání SDO paketu z PLC programu

PLC program má možnost vyslat na pohon asynchronně SDO paket. Pro vyslání slouží instrukce **CAN_AX_SEND.**

instrukce	**CAN_AX_SEND**
funkce	**vyslání paketu na pohon**
syntax	**CAN_AX_SEND axis**
parametr	**„axis" číslo souřadnice**

Parametr „axis" určuje pořadové číslo souřadnice pro „trajectory mód" nebo pořadové číslo výstupního kanálu pro „speed control".

V PLC programu jsou zpřístupněna datová pole **CAN_AX_SEND_PACKET** a **CAN_AX_RECV_PACKET ,** která mají typ struktury CAN-BUS (12 bajtů TCANMSGS). Pole CAN_AX_SEND_PACKET slouží na vyslání paketu do pohonu a pole CAN_AX_RECV_PACKET slouží pro příjem paketu z pohonu.

Instrukce sama nastaví CAN_ID podle čísla osy a podle nastavené konfigurace. CAN_RTR a CAN_LEN jsou také přednastaveny, proto PLC program vyplní jen datové pole paketu CAN_DATA (max.8 bajtů)

Instrukce při zavolání nastaví buňku **CAN_AX_BUSY** (bajt) na hodnotu 0FFh. Po příjmu odpovědi na SDO paket z pohonu, se buňka automaticky vynuluje. Pokud PLC program potřebuje znát odpověď na vyslaný SDO paket nebo chce zkontrolovat zda pohon příjmul SDO paket v pořádku, tak musí buňku CAN_AX_BUSY testovat a připadne vyslání SDO paketu opakovat.

```
;CAN-Message
TCANMSGS    STRUC
        CAN_ID          DW 0        ;11 Bit-ID
        CAN_RTR         DB 0        ;true, if remote request
        CAN_LEN         DB 0        ;Number of valid Data bytes (0..8)
        CAN_DATA        DB 0        ;Databytes 0..7
        CAN_DATA_1      DB 0        ;Data 1
        CAN_DATA_2      DB 0        ;Data 2
        CAN_DATA_3      DB 0        ;Data 3
        CAN_DATA_4      DB 0        ;Data 4
        CAN_DATA_5      DB 0        ;Data 5
        CAN_DATA_6      DB 0        ;Data 6
        CAN_DATA_7      DB 0        ;Data 7
TCANMSGS    ENDS
```

Příklad:
Příklad pro UNIDRIVE, vyslání hodnoty 1 do registru 6.15 (Enable) s opakováním vysílání.

```
MECH_BEGIN SendPacket1
SendPacket1_cykl:
      lod         cnst.2Fh
      sto         byte.CAN_AX_SEND_PACKET.CAN_DATA
      lod         cnst.2006h
      sto         word.CAN_AX_SEND_PACKET.CAN_DATA_1       ;index 2006h
      lod         cnst.10h
      sto         byte.CAN_AX_SEND_PACKET.CAN_DATA_3       ;subindex 10h
      lod         cnst.01
      sto         byte.CAN_AX_SEND_PACKET.CAN_DATA_4       ;data 01
      CAN_AX_SEND 1                                        ;vyslani paketu
      ex                                                   ;ceka 20ms
      ldr         CAN_AX_BUSY.b0
      jl1         SendPacket1_cykl                         ;opakuje vyslani
MECH_END SendPacket1
```

Poznámka:
Jiný způsob nastavení Enable pro UNIDRIVE (6.15 =1) je pomocí CAN_DRIVE_CMD. Tyto dva způsoby nastavování se nedoporučuje kombinovat pro nastavování stejného parametru.

```
      fl          1,(CAN_DRIVE_CMD+1).CAN_UAX_M0           ;odmaskovani
      fl          1,(CAN_DRIVE_CMD+0).CAN_UAX_EN           ;Enable Unidrive
```

13.14.5 Chybová hlášení

Přehled chybových hlášení, které vzniknou při konfiguraci CAN-BUSu, nebo jako chybové hlášení pohonu (emergency message).

Číslo chyby	Popis
4201	Chyba inicializace CAN ovladače (1)
4202	CAN ovladač hlásí plný přijmový buffer (2)
4203	CAN ovladač hlásí chybu zběrnice (3)
4204	CAN ovladač hlásí přerušení zběrnice (4)
4205	Chyba driveru 250us (5)
4206	Problém vysílání při módování (6)
4208	Pohon %d neodpovídá
4209	Špatná odezva na SDO povel pro %d. pohon
4210	Nepřišel PDO paket po SYNC pro %d. pohon
4211	Problém s vysíláním při provozu - paket %d
4212	Chyba módování pro referenci - pohon %d. neodpovědel
4213	Nenašla se karta PCI-CAN %d.kanal pro CAN-BUS pohon (CAN1)
4214	Chyba v úvodní inicializaci %d. pohonu na test statusu.
4215	Chyba v úvodní inicializaci %d. pohonu při přepínání na režin MOVE-PTP
4216	Pohon %d. hlásí signál TRIP
4217	Emergency hlašení z %d. pohonu, chyba: %d, emergency: %x
4218	Chyba při EDS konfiguraci: %d. ID pohonu: %x. Index: %x
4219	Chyba TIME-OUT pohonu: %d

Přehled chybových hlášení pohonu **Kollmorgen** (emergency massage)

chyba	Popis originál Kollmorgen – Servostar 600	Popis
1	(1000h) Generic error mandatory	Všeobecná chyba
2	(1080h) No BTB/RTO (status not ready for operation)	Chybí BTB/RTO
3	(2330h) Earth short (F22)	Zkrat zemí
4	(3100h) No mains/line – BTB (F16)	Chybí hlav.přívod BTB
5	(3110h) Overvoltage in DC-bus/DC-link (F02)	Překročeno napětí
6	(3120h) Undervoltage in DC-bus/DC-link (F05)	Podpětí
7	(3130h) Supply line phase missing (with PMODE=2) (F19)	Chybí fáze
8	(4110h) Ambient temperature too high (F13)	Překročena teplota okolí
9	(4210h) Heat sink temperature too high (F01)	Překročena teplota chladiče
10	(4310h) Motor temperature too high (F06)	Překročena teplota motoru
11	(5111h) Fault in +/-15V auxililiary (F07)	Chyba v příslušenství +/-15V
12	(5380h) Fault in A/D converter (F17)	Chyba v A/D převodníku
13	(5400h) Fault in output stage (F14)	Chyba ve výstupném stupni
14	(5420h) Ballast (chopper) (F18)	Zátěž
15	(5441h) Operating error for AS-option (F27)	Operační chyba v AS
16	(5530h) Serial EEPROM (F09)	Sériová EEPROM
17	(5581h) Flash EEPROM (F10)	Flash EEPROM
18	(6010h) Watchdog (software reset, F32)	Hlídání
19	(6181h) BCC error (table)	BCC chyba (tabulky)
20	(6182h) BCC error (system macro)	BCC chyba (systémové makro)
21	(6183h) BCC error (serial EEPROM)	BCC chyba (sériová EEPROM)
22	(6184h) FPGA error	Chyba FPGA
23	(6185h) Fault/error (table)	Chyba tabulky

24	(6281h) User software BCC (macro, F32)	BCC uživatelského software
25	(6282h) Faulty user software (macro, F32)	Chyba parametru
26	(6320h) Parameter error	Chyba parametrů
27	(7111h) Braking error/fault (F11)	Chyba brzdy
28	(7122h) Commutation error (F25)	Chyba komutování
29	(7181h) Could not enable SERVOSTAR	Neumožněno pro SERVOSTAR
30	(7182h) Command only possible in disabled status	Příkaz je možný v režimu disable
31	(7303h) Feedback device error (F04)	Chyba v zařízení Feedback
32	(8053h) Handling error (F21)	Chyba v řízení
33	(8181h) Response monitoring activated	Aktivována monitorovací odezva
34	(8182h) CAN bus off (F23)	CAN bus je vypnutý
35	(8281h) Status machine not in operation enable condition	Stav neumožněn v provozu
36	(8282h) Wrong mode setting	Špatně nastaven mód
37	(8331h) I2t torque fault (F15)	Chyba momentu I2t
38	(8480h) Overspeed (F08)	Překročena rychlost
39	(8611h) Lag/following error	Překročena polohová odchylka
40	(8681h) Invalid motion task number	Špatné číslo posuv.bloku
41	(8682h) External trajectory error (F28) (only with Sercos)	Chyba v externí dráze
42	(FF01h) Serious exception error (F32)	Vážná výjimka
43	(FF02h) Error in PDO elements	Chyba v PDO prvku
44	(FF03h) Operating mode	Operační mód
45	(FF04h) Slot error (F20)	Chyba slotu
46	(FF06h) Warning display as error (F24)	Hlášení jako chyba
47	(FF07h) Homing error (drove onto HW limit switch) (F26)	Chyba reference
48	(FF08h) Sercos error (F29)	Chyba SERCOS
49	another error	jiná chyba

Přehled chybových hlášení pohonu **Maxon-Epos** (emergency massage)

chyba	**Popis originál Maxon – Epos**	**Popis**
1	(1000h) Generic error mandatory	Všeobecná chyba
2	(2310h) Over Current Error	Překročení proudu
3	(3210h) Over Voltage Error	Přepětí
4	(3220h) Under Voltage	Podpětí
5	(4210h) Over Temperature	Překročení těploty
6	(5113h) Supply Voltage (+5V) too low	Nízké napájecí napětí 5V
7	(6100h) Internal software Error	Interní softwarová chyba
8	(6320h) Software Parameter Error	Chyba softwarových parametrů
9	(7320h) Sensor Positon Error	Chyba snímače polohy
10	(8110h) CAN Overrun error	Chyba přetečení CAN
11	(8120h) CAN Passive Mode Error	CAN je v pasivním módu
12	(8130h) CAN Life Gard Error	Chyba ochrany CAN
13	(81FDh) CAN Bus Off	CAN-BUS je rozpojený
14	(81FEh) CAN Rx Queue Overrun	Přetečení příjmové fronty v CAN
15	(81FFh) CAN Tx Rx Queue Overrun	Přetečení vysílací fronty v CAN
16	(8611h) Lag/following error	Překročena polohová odchylka
17	(FF01h) Hall Sensor Error	Chyba halových snímačů
18	(FF02h) Index Processing Error	Chyba nulového pulsu snímače
19	(FF03h) Encoder Resolution Error	Chyba v nastavení snímače
20	(FF04h) Hallsensor not found Error	Chyba v detekci halového snímače
21	(FF05h) Over speed Error	Překročena rychlost
22	(FF06h) Negative Limit Error	Záporní limitní spínač
23	(FF07h) Positive Limit Error	Kladní limitní spínač
24	(FF08h) Hall Angle detection Error	Chyba halové sondy
25	(FF09h) Software Position Limit Error	Chyba minimální posiční chyby
26	(FF0Ah) Position Sensor Breach	Porušení posičního sensoru

Přehled chybových hlášení pohonu **TGA-24** (emergency massage)

chyba	**Popis originál TGA–24**	
1	Zkrat	
2	Poziční chyba	
3	Proudové přetížení	
4	Externí ENABLE	
5	Resolver motoru	
6	Termistor serva	
7	Termistor motoru	
8	Chyba zápisu do Flash paměti	
9		
10	Chyba režimu CAN Trajectory	

Přehled chybových hlášení pohonu **BERGER LAHR CPD17** (emergency massage)

chyba	**Popis**	**index**
1	power amplifier overcurent	2300
2	ballast resistor overcurrent	2301
3	mains power supply phase fault	3100
4	DC bus overvoltage	3200
5	DC bus low voltage	3201
6	DC bus low voltage	3202
7	Motor encoder supply voltage	3203
8	DC bus low voltage warning	3206
9	Output stage excess temperature	4100
10	Power amplif. overtemper.warning	4101
11	Output stage overload I2T warning	4102
12	Unit overtemperature	4200
13	Motor overtemperature	4300
14	Motor overtemperature warning	4301
15	Motor overload i2t warning	4302
16	Ballast resistor overload i2t warning	4303
17	No connection motor encoder	5200
18	errors in motor sensor comunication	5201
19	motor encoder is not supported	5202
20	no connection to the motor encoder	5203
21	connection to motor encoder lost	5204
22	CAN overlow	8110
23	CAN controller in error passive	8120
24	Heartbeat or life guard error	8130
25	CAN controller was in Busoff	8140
26	CAN controller in Busoff	8141
27	drive in state FAULT	A308
28	drive not in state „operation enable“	A309
29	power amplifier not active	A310
30	profile generation interrupt	A312
31	position over-run present	A313
32	no reference position	A314
33	referencing active	A315
34	overrun on acceleration calculation	A316
35	drive not at standstill	A317
36	operating mode active	A318

37	manual/autotuning: distance range overlow	A319
38	manual/autotuning: amlitude/offset set to high	A31A
39	STOP requested	A31B
40	illegal position setting with software limit switch	A31C
41	speed range exceeded	A31D
42	interruption by pos. software limit switch	A31E
43	interruption by neg. software limit switch	A31F
44	position lag error	A320
45	error when referencig	A324
46	approach limit switch not activated	A325
47		
48		
49	another error	

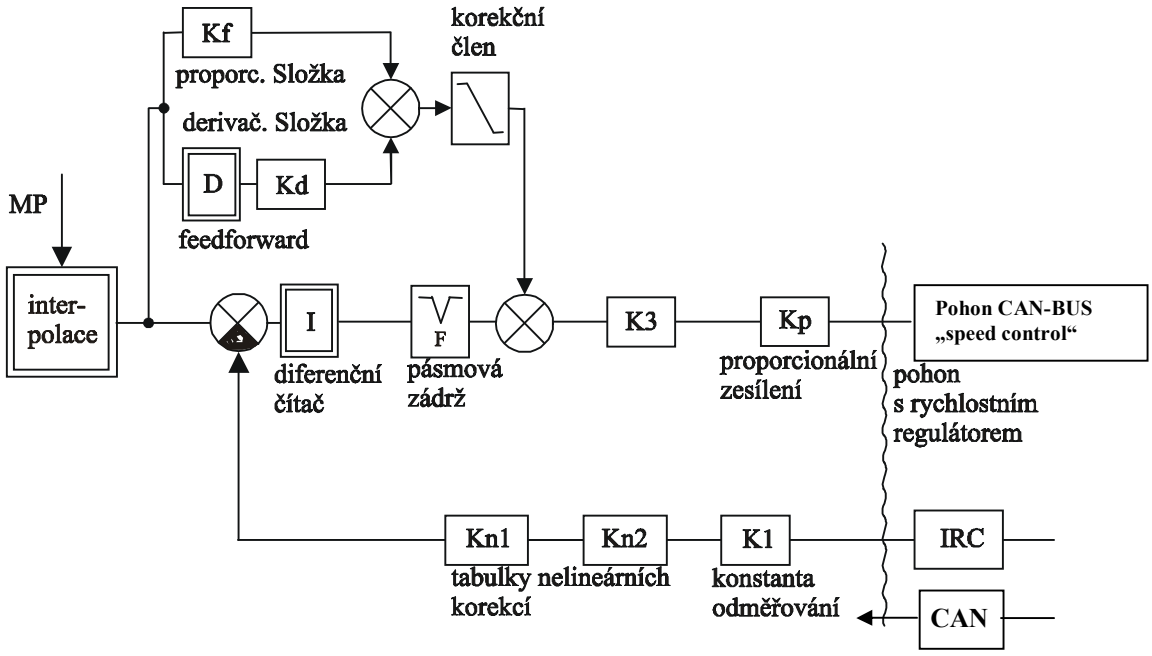
13.15 Pohony připojené pomocí sběrnice CAN-BUS v režimu „speed control“

Systém má možnost řídit pohony přes sběrnici CAN-BUS i v režimu **„speed control“**.

Systém používá vlastní polohovou sysrvosmyčku a vlastní odměřování. Jen výstup na pohon je poslán místo na D/A převodník, přímo na kanál CAN-BUS. Tento způsob připojení není tak výhodný jako „trajectory control“, protože systém musí být také osazen jednotkou odměřování (SU05). Také interní polohová servosmyčka má pomalejší výpočtový rastr (1 ms) v porovnání s externí polohovou servosmyčkou. Přes tyto nevýhody, získá se digitální připojení pohonu, které sebou nese řadu výhod. Například u digitálního připojení pohonu nejsou problémy s nastavením driftu.

Režim „speed control“ se používá také v případě, že pohon „trajectory režim“ nepodporuje nebo z jiných důvodů jej není možno použít.

Také je možné v režimu „speed control“ využít odměřování z pohonu získané z CANu.



Všechny parametry pro nastavení dynamiky, způsobu reference, nastavení rozlišení apod. se nastavují normálně v systému pomocí konfigurace.

Komunikační pakety obsahují 11-bitové ID, které je složeno ze 7-bitové adresy pohonu a 4-bitového kódu závislém na typu komunikace**. Adresu pohonu** je nutno nastavit předem přímo v pohonu a nastavuje se vzestupně od hodnoty 1 (1,2,3,..). Na pohonech je také nutno nastavit **rychlost komunikace** (1MBd). Schéma kabelu pro připojení pomocí CAN-BUSu je v příloze návodu a má označení **K18.**

Základní konfigurace CAN-BUSu, rozhraní pro PLC program a Chybová hlášení jsou popsána v předešlé podkapitole („Pohony připojené pomocí CAN-BUSu v režimu „trajectory control“.)

Kombinace nastavení „speed control“ a „trajectory control“ je pro současnou verzi zakázána.

13.15.1 Nastavení pro pohony CAN-BUS „speed control“

Souřadnici, která je řízená pomocí CAN-BUSu, zadává výstupní hodnotu pro pohon interní polohová servosmyčka. Rychlostní servosmyčka je uzavřena v pohonu („speed control“), proto pro takovou souřadnici platí všechny parametry pro nastavení dynamiky servosmyček v systému.

Řízení výstupních kanálů

element **DriveChannel**		**Nastavení výstupního kanálu**	
	atribut **DriveType**	**Typ výstupního kanálu**	
		0	žádný výstup (default)
		1	*výstup na SU05 v plném provozu*
		2	*výstup na SU05 s vyřazením testů*
		3	Kollmorgen ServoStar, řady 400 a 600
		4	Maxon - Epos
		5	TGA 24
		6	Berger-Lahr CPD 17
		7	Control Techniques UniDrive
		8	Control Techniques UniDrive SP (inicializace při startu systému)
		9	Control Techniques UniDrive SP (inicializace z PLC)
		12	TGPower
		17	Telemecanique ATV (Schneider)
		18	Plovouci RTM - matematický model stroje (CVUT)
		19	TGPowerTrajectory ID1,2 + TGPowerSpeed ID3
		20	Berger-Lahr CPD 17 nebo Lexium 04, aktivace z PLC
		21	Servostar 300, Lexium 15
		22	Sanyo RS1
		23	Lexium 32 (Schneider)
		24	Estun EDC v3.10, Pronet
		25	Altivar ATV71 (Schneider)
		26	Estun EDC v3.11
		27	Gefran XVy

Řízení vstupních kanálů

element **EncoderChannel**		**Nastavení kanálu odměřování**	
	atribut **ChannelType**	**Typ odměřovacího kanálu**	
		0	žádné odměřování (default)
		1	*odměřování SU05 v plném provozu*
		2	*odměřování SU05 s vyřazením testů*
		3	Kollmorgen ServoStar, řady 400 a 600
		4	Maxon - Epos
		5	TGA 24
		6	Berger-Lahr CPD 17
		7	Control Techniques UniDrive
		8	Control Techniques UniDrive SP (inicializace při startu systému)
		9	Control Techniques UniDrive SP (inicializace z PLC)
		12	TGPower
		17	Telemecanique ATV (Schneider)
		18	Plovouci RTM - matematický model stroje (CVUT)
		19	TGPowerTrajectory ID1,2 + TGPowerSpeed ID3
		20	Berger-Lahr CPD 17 nebo Lexium 04, aktivace z PLC
		21	Servostar 300, Lexium 15
		22	Sanyo RS1
		23	Lexium 32 (Schneider)
		24	Estun EDC v3.10, Pronet
		25	Altivar ATV71 (Schneider)
		26	Estun EDC v3.11
		27	Gefran XVy

13.16 EDS soubory pro CAN-BUS konfiguraci

EDS soubory mohou sloužit pro automatickou konfiguraci CAN-BUS pohonů. Načtení EDS souborů se řídí konfigurací systému:

element **CANChannel**			**Nastavení kanálu CAN-BUSu**	
	atribut **UseEdsFiles**		**Použít pro daný CAN kanál konfigurační soubory EDS?**	
			0	EDS soubory nejsou použity (default)
			1	EDS soubory jsou použity
	atribut **SdoDelay**		**Prodleva mezi vysíláním SDO paketů v průběhu konfigurace jako násobek základního taktu**	
			0	žádná prodleva (default)
			xx	prodleva mezi SDO pakety z EDS souboru [ms]
	element **EdsFile**		**konfigurace EDS souborů**	
		atribut **UseEdsFile**	**Použít daný EDS soubor?**	
			0	EDS soubor nepoužit (default)
			1	EDS soubor použit
		atribut **NodeID**	**ID zařízení, pro které je určen daný EDS soubor**	
			0	žádné ID (default)
			xx	ID pohonu CAN-BUS (1,2,3,...)
		atribut **FirstSdoIndex**	**Index prvního SDO paketu, který se použije z daného SDO souboru**	
			0	žádný index (default)
			0xNNNN	index prvního SDO paketu, například: `0x2003`
		atribut **EdsFileName**	**Jméno EDS souboru**	
			0	žádné jméno (default)
			abc.dcf	jméno EDS souboru, hledá se v podadresáři Config

13.16.1 Pravidla pro vysílání paketů

Do pohonu se neodvysílají všechny pakety podle EDS souboru. Předpokládáme že pohon je nastaven na defaultní hodnoty a tak se do pohonu odvysílají jen ty, které jsou rozdílné od defaultního stavu. Také se do pohonu neodvysílají pakety pro mapování PDO paketů, protože mapování provádí systém podle svých požadavků.

Pravidla pro odvysílání SDO paketu:	
1.	EDS soubor se prohledáva od klíčového slova `[MANUFACTUREROBJECTS]`
2.	EDS soubor se začne prohledávat od indexu indexu určeném atributem `FirstSDOIndex`. (pokud se dále objeví index s nižší hodnotou, tak se může uplatnit)
3.	Parametr musí mít povolen zápis `AccessType=RW`
4.	Musí být definována defaultní hodnota `DefaultValue=xxx` (za znakem = musí být uvedeno číslo)
5.	Hodnota parametru musí být rozdílná od defaultní hodnoty `DefaultValue <> ParameterValue`
6.	Typ dat musí být některý z výčtu 2,3,4,5,6,7 `DataType=0x3`

Příklad definice SDO paketu 60F9 v EDS souboru, kdy bude paket odvysílán (0x60F9 > 0x2003)

```
[60F9sub1]
ParameterName=Speed Regulator P-Gain
DataType=0x3
LowLimit=
HighLimit=
AccessType=RW
DefaultValue=680
ParameterValue=12000
PDOMapping=1
```

13.17 Pohony připojené pomocí sběrnice EtherCAT, společné zásady

V dalším popisu budeme navazovat na návody „Periferie připojené na EtherCAT“ a „Logické vstupy a výstupy modulů systému“.

Sběrnici EtherCAT obsluhuje program „KPA EtherCAT Master“. Jedná se o úlohu reálného času nad REAL-TIME jádrem „RTX“. Mezi jeho základní vlastnosti patří: „Distribuovaný Clock“ (DC), vstupy a výstupy přes „Process Image“ (PI), čtení a zápis parametrů „MailBox“ typu „CoE“ nebo „SoE“.

Konfigurace pohonů a prvotní inicializace je definovaná v souboru typu XML. Tento soubor je generovaný externím programem „EtherCAT Studio“ například od firmy „Koenig-pa GmbH“. Tak konfigurace EtherCAT pohonů není závislá na CNC systému. Pro připojení libovolného pohonu je nutno mít aktuální verzi XML souboru pro danou verzi firmware od výrobce pohonu.

V EtherCAT studiu se nastavuje “mapování“, kde se skládá sestava prvků do synchronizačních objektů, které zprostředkují cyklickou výměnu dat. Dále se nastavují všechny příkazy, které se mají do zařízení vyslat při inicializaci. Nastavuje se také „Distribuovaný Clock“ pro všechna zařízení a určí se „Referenční Clock“ pro celou sběrnici. Všechny informace se vygenerují do konfiguračního XML souboru (Master.xml), kterým se řídí EtherCAT Master.

Na EtherCATu jsou tři základné typy komunikace.

1. Inicializace a konfigurace všech zařízení. EtherCAT Maseter vysílá postupně příkazy při přechodech do jednotlivých stavů:

   ```
   INIT -> PRE-OPERATIONAL -> SAFE-OPERATIONAL -> OPERATIONAL
   ```

2. Cyklická výměna dat. EtherCAT Maseter řídí cyklickou výměnu dat v reálném čase z periodou například 250μs nebo 1ms. PLC program má možnost definovat prvky, které se cyklické výměny účastní pomocí logických vstupů a výstupů.

   ```
   <Connection Source="RTM.Servo[2].EdrvDriveVeloReq"
               Destination="ECAT.LXM32M.Outputs.Target velocity"
               Connected="1"></Connection>
   ```

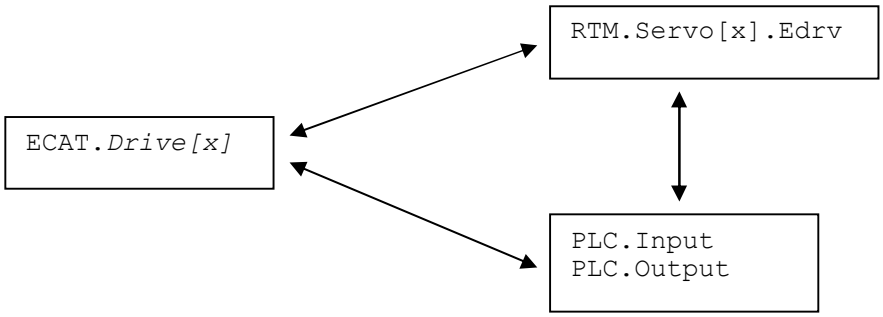
3. Acyklická komunikace. Jedná se o jednorázovou komunikaci typu „Mailbox“ a používá se komunikační profil CoE a SoE. PLC program může použít instrukce:

   ```
   ECAT_CoE_READ,    ECAT_SoE_READ
   ECAT_CoE_WRITE.   ECAT_SoE_WRITE
   ```

13.18 Pohony připojené pomocí sběrnice EtherCAT v režimu „trajectory control“

Pohony se řídí v módu **„trajectory control“**, to znamená, že polohová servosmyčka je uzavřena mimo systém v pohonu. Tím je umožněno dosáhnout lepších dynamických parametrů.

Propojení pohonu se systémem (nebo také PLC programem) je tvořeno tzv. logickými spoji (virtuální spoje, viz. Návod: „Logické vstupy a výstupy modulů systému.“). V tomto případě se jedná o logické spoje mezi moduly ECAT, RTM.Servo[x].Edrv a PLC.



Konfigurace pohonu v souboru “channelconfig”:

element **Servo**		**konfigurace servosmyčky**	
	atribut **ServoType**	**typ servosmyčky**	
		0	*servosmyčka neaktivní (default)*
		1	*standardní softwarová servosmyčka*
		….	
		28	Pohon EtherCAT v trajectory režimu, komunikační profil CoE
		29	Pohon EtherCAT v trajectory režimu, komunikační profil SoE

Pohony připojené sběrnicí EtherCAT, které ovládá systémová software, mají název **EDRV** pohony.

13.18.1 Konfigurace pro pohony EtherCAT „trajectory control“

Seznam konfiguračních atributů v elementu <Servo> v souboru “channelconfig”, které je možno použít pro EtherCAT pohon v trajectory režimu:

Atribut	**popis**
Servotype	Typ servosmyčky (28)
MeasConstNumerator	Konstanta odměřování – čitatel
MeasConstDenominator	Konstanta odměřování – jmenovatel
FollowingErrorLimit	Limit pro hlídání velkosti polohové odchylky
InvertOutputPolarity	Polarita výstupu
MaxDifference	Maximální zbytková odchylka
MachineClearance	Kompenzace vůlí stroje
NonLinearCompensation	Element pro nastavení nelineárních korekcí serva

Pro EtherCAT pohon v trajectory režimu platí pro výpočet odměřovací konstanty:

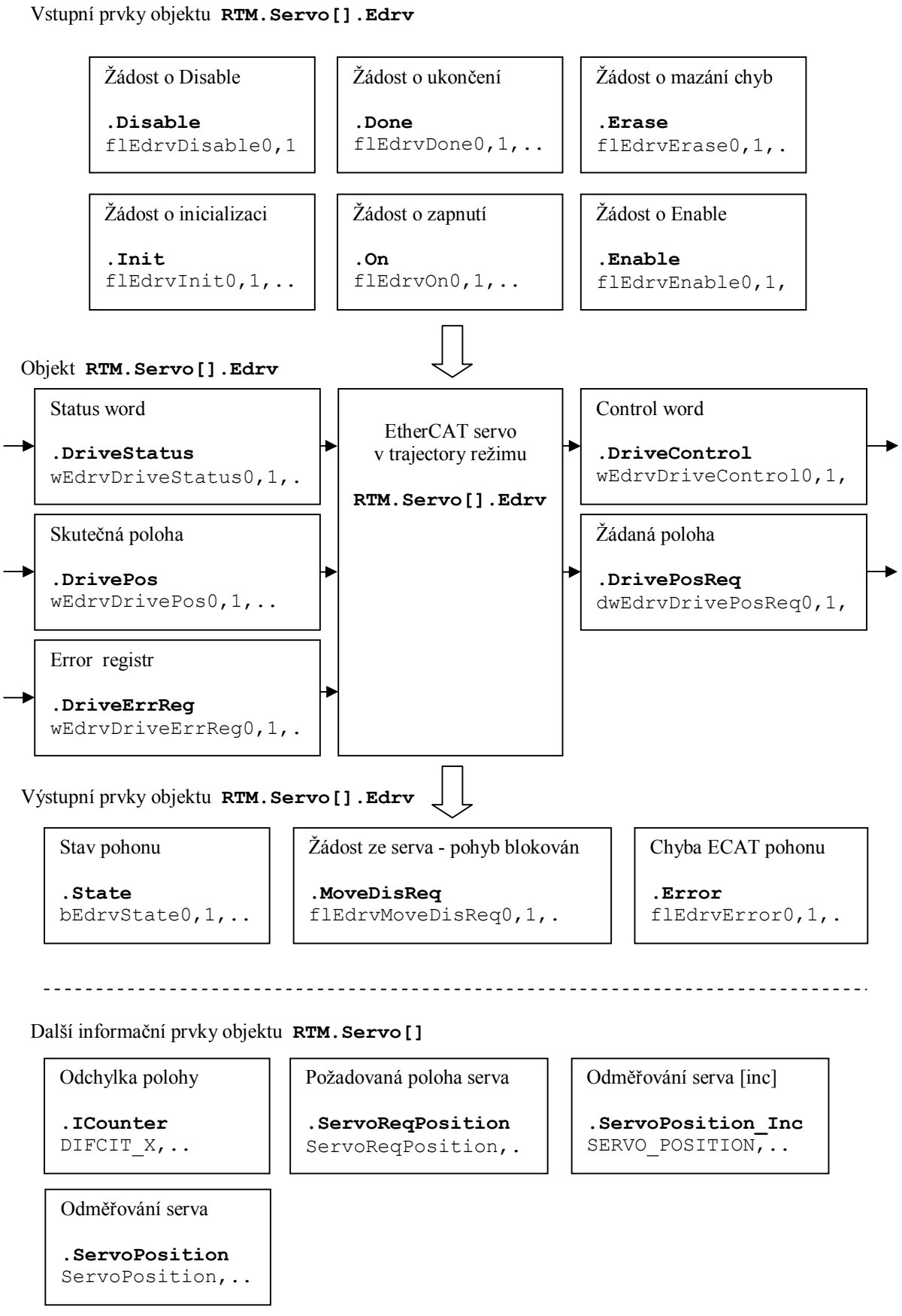
$$\text{míra v mikrometrech} * 2^{16} * \frac{\text{MeasConstNumerator}}{\text{MeasConstDenominator}} = \text{počet inkrementů pohonu}$$

Control word CoE		
Bit	název	popis
0	CoE_CNT_ON	Switch on
1	CoE_CNT_VOLT	Enable voltage
2	CoE_CNT_QSTOP	Quick stop
3	CoE_CNT_ENABLE	Enable operation
4	CoE_CNT_M0	Operation mode specific
5	CoE_CNT_M1	
6	CoE_CNT_M2	
7	CoE_CNT_FAULT	Fault reset

Status word CoE		
Bit	název	popis
0	CoE_STAT_RDY	Ready to switch on
1	CoE_STAT_ON	Switched on
2	CoE_STAT_ENABLE	Operation enabled
3	CoE_STAT_FAULT	Fault
4	CoE_STAT_VOLT	Voltage enabled
5	CoE_STAT_QSTOP	Quick stop
6	CoE_STAT_DIS	Switch on disabled
7	CoE_STAT_WARN	Warning

Příklad: `LDR wEdrvDriveStatus0.CoE_STAT_FAULT`

13.18.2 Schéma propojení EtherCAT pohonu „trajectory control“



Přehled řídicích a stavových prvků

Prvky pro logické spoje **RTM.Servo[x].Edrv**	Rozhraní pro PLC	Popis
.DriveStatus	wEdrvDriveStatus0,1,..	[in] Stavové slovo pohonu (Status word 6041:0) [UNS16]
.DrivePos	dwEdrvDrivePos0,1,..	[in] Skutečná poloha motoru (Position actual value 6064:0) [INT32, inc]
.DriveErrReg	wEdrvDriveErrReg0,1,..	[in] Chybový registr pohonu (Error register 603F:0) [UNS16]
.DriveControl	wEdrvDriveControl0,1,.	[out] Řídicí slovo pohonu (Control word 6040:0) [UNS16]
.DrivePosReq	dwEdrvDrivePosReq0,1,..	[out] Žádaná poloha (Target position 607A:0) [INT32, inc]
.DriveVeloReq	dwEdrvDriveVeloReq0,1,.	[out] Žádaná rychlost (Velocity demand value 60FF:0) [INT32]
.State	bEdrvState0,1,..	[out] Stav pohonu (EDRV_INIT, EDRV_ON, EDRV_ENABLE) [UNS8]
.MoveDisReq	flEdrvMoveDisReq0,1,.	[out] Žádost ze serva o blokování pohybu pro danou osu [BIT]
.Error	flEdrvError0,1,.	[out] Pohon je v chybě [BIT]
.Init	flEdrvInit0,1,..	[in] Žádost o inicializaci pohonu [BIT]
.On	flEdrvOn0,1,..	[in] Žádost o zapnutí pohonu (nastaví bity QSTOP, VOLT, ON) [BIT]
.Enable	flEdrvEnable0,1,..	[in] Žádost Enable pohonu (nastaví ENABLE) [BIT]
.Disable	flEdrvDisable0,1,..	[in] Žádost Disable pohonu (vynuluje ENABLE) [BIT]
.Done	flEdrvDone0,1,..	[in] Žádost ukončení (vynuluje control word) [BIT]
.Erase	flEdrvErase0,1,..	[in] Žádost o mazání chyb pohonu [BIT]
.ScaleSpeed	rEdrvScaleSpeed0,1,..	[in] Měřítko pro zadání rychlosti [REAL]
	flEdrvSimul0,1,..	[in] Pohon v simulaci [BIT]
Prvky pro logické spoje **RTM.Servo[x].**	Rozhraní pro PLC	Popis
.ICounter	DIFCIT_X,+4,+8,..	Odchylka polohy [INT32]
.ServoReqPosition	ServoReqPosition,+4,..	Požadovaná poloha serva [INT32]
.ServoPosition	ServoPosition,+4,..	Odměřování serva [INT32]
.ServoPosition_Inc	SERVO_POSITION,..	Odměřování serva [INT32 inc]

EtherCAT servo je možno propojit s fyzickým pohonem pomocí logických spojů, nebo pomocí rozhraní přístupného v PLC programu. Dále je uveden příklad pro konfiguraci logických spojů pro pohony Lexium32 Schneider (název Slave na EtherCATu je LXM32M).

```
<!-- Servo[1] LXM32 trajectory -->
<Connection Source="RTM.Servo[1].Edrv.DriveControl"
            Destination="ECAT.LXM32M.Outputs.Control word"
            Connected="1"></Connection>

<Connection Source="RTM.Servo[1].Edrv.DrivePosReq"
            Destination="ECAT.LXM32M.Outputs.Target position"
            Connected="1"></Connection>

<Connection Source="ECAT.LXM32M.Inputs.Status word"
            Destination="RTM.Servo[1].Edrv.DriveStatus"
            Connected="1"></Connection>

<Connection Source="ECAT.LXM32M.Inputs.Position actual value"
            Destination="RTM.Servo[1].Edrv.DrivePos"
            Connected="1"></Connection>

<Connection Source="ECAT.LXM32M.Inputs.Drive error number"
            Destination="RTM.Servo[1].Edrv.DriveErrReg"
            Connected="1"></Connection>
```

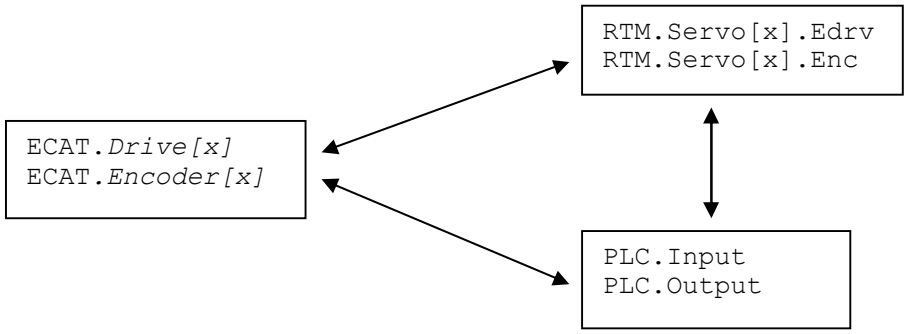
Ovládání EtherCAT pohonu z PLC programu:

```
;Inicializace EDRV pohonu
            FL          1,flEdrvInit0           ;Pohon 0
            EX
            LDR         flEdrvInit0
            EX1

;Zapnutí EDRV pohonu
            FL          1,flEdrvOn0             ;Pohon 0
            EX
            LDR         flEdrvOn0
            EX1

;Enable EDRV pohonu
            FL          1,flEdrvEnable0         ;Pohon 0
            EX
            LDR         flEdrvEnable0
            EX1

;Disable EDRV pohonu
            FL          1,flEdrvDisable0        ;Pohon 0
            EX
            LDR         flEdrvDisable0
            EX1

;Ukončení činnosti pohonu
            FL          1,flEdrvDone0           ;Pohon 0
            EX
            LDR         flEdrvDone0
            EX1
```

13.19 Pohony připojené pomocí sběrnice EtherCAT v režimu „speed control“

Systém používá vlastní polohovou sesrvosmyčku a výstup žádané rychlosti na pohon je poslán přímo do pohonu, který musí být v rychlostním režimu. Konfigurací je možno zvolit mezi interním odměřováním přímo z EtherCAT pohonu, nebo externím odměřováním například z jednotky EtherCAT EL5101 od firmy Beckhoff.

Propojení pohonu se systémem (nebo také PLC programem) je tvořeno tzv. logickými spoji (virtuální spoje, viz. Návod: „Logické vstupy a výstupy modulů systému.“). V tomto případě se jedná o logické spoje mezi moduly ECAT, RTM.Servo[x] a PLC.



Konfigurace pohonu v souboru “channelconfig”:

element **Servo**		**konfigurace servosmyčky**	
	atribut **ServoType**	**typ servosmyčky**	
		0	*servosmyčka neaktivní (default)*
		1	standardní softwarová servosmyčka
		….	

element **EncoderChannel**		**Nastavení kanálu odměřování**	
	atribut **ChannelType**	**Typ odměřovacího kanálu**	
		0	*žádné odměřování (default)*
		...	
		28	Odměřování z pohonu EtherCAT, komunikační profil CoE
		29	Odměřování z pohonu EtherCAT, komunikační profil SoE
		30	Odměřování z jednotky EtherCAT EL5101 Beckhoff

element **DriveChannel**		**Nastavení výstupního kanálu**	
	atribut **DriveType**	**Typ výstupního kanálu**	
		0	*žádný výstup (default)*
		...	
		28	Výstup na pohon EtherCAT, komunikační profil CoE
		29	Výstup na pohon EtherCAT, komunikační profil SoE

13.19.1 Konfigurace pro pohony EtherCAT „speed control“

Pro EtherCAT pohon v režimu „speed control“ v elementu <Servo> v souboru “channelconfig” je možno použít všechny dostupné atributy. Nakonfigurovaná je standardní softwarová servosmyčka.

Konfigurační možnosti v souboru “channelconfig”:

Odměřování z EtherCAT pohonu a výstup na EtherCAT pohon v režimu speed control

```
<EncoderChannel No="2"
      ChannelType="28"
      EncoderType="0">
</EncoderChannel>

<DriveChannel No="2"
      DriveType="28">
</DriveChannel>
```

Odměřování z EtherCAT jednotky EL5101 a výstup na EtherCAT pohon v režimu speed control

```
<EncoderChannel No="3"
      ChannelType="30"
      EncoderType="0">
</EncoderChannel>

<DriveChannel No="3"
      DriveType="28">
</DriveChannel>
```

Samotné odměřování z jednotky EL5101

```
<EncoderChannel No="4"
      ChannelType="30"
      EncoderType="0">
</EncoderChannel>

<DriveChannel No="4"
      DriveType="2">
</DriveChannel>
```

Samotné výstup na EtherCAT pohon v režimu speed control bez odměřování

```
<DriveChannel No="5"
      DriveType="28">
</DriveChannel>
```

Pro EtherCAT pohon v speed control režimu platí pro výpočet odměřovací konstanty:

$$\text{počet pulzů odměřování} * \frac{\text{MeasConstNumerator}}{\text{MeasConstDenominator}} = \text{míra v mikrometrech}$$

13.19.2 Schéma propojení EtherCAT pohonu „speed control“

Vstupní prvky objektu **RTM.Servo[].Edrv**

Stav pohonu	Žádost ze serva - pohyb blokován	Chyba ECAT pohonu
.State bEdrvState0,1,..	**.MoveDisReq** flEdrvMoveDisReq0,1,.	**.Error** flEdrvError0,1,.

Další informační prvky objektu **RTM.Servo[]**

Odchylka polohy	Požadovaná poloha serva	Odměřování serva [inc]
.ICounter DIFCIT_X,..	**.ServoReqPosition** ServoReqPosition,.	**.ServoPosition_Inc** SERVO_POSITION,..

Odměřování serva
.ServoPosition ServoPosition,..

Přehled řídicích a stavových prvků

Prvky pro logické spoje **RTM.Servo[x].Edrv**	Rozhraní pro PLC	Popis
.DriveStatus	wEdrvDriveStatus0,1,..	[in] Stavové slovo pohonu (Status word 6041:0) [UNS16]
.DrivePos	dwEdrvDrivePos0,1,..	[in] Skutečná poloha motoru (Position actual value 6064:0) [INT32, inc]
.DriveErrReg	wEdrvDriveErrReg0,1,..	[in] Chybový registr pohonu (Error register 603F:0) [UNS16]
.DriveControl	wEdrvDriveControl0,1,.	[out] Řídicí slovo pohonu (Control word 6040:0) [UNS16]
.DriveVeloReq	dwEdrvDriveVeloReq0,1,..	[out] Žádaná rychlost (Target velocity 60FF:0) [INT32, inc]
.DriveVeloReq	dwEdrvDriveVeloReq0,1,.	[out] Žádaná rychlost (Velocity demand value 60FF:0) [INT32]
.State	bEdrvState0,1,..	[out] Stav pohonu (EDRV_INIT, EDRV_ON, EDRV_ENABLE) [UNS8]
.MoveDisReq	flEdrvMoveDisReq0,1,.	[out] Žádost ze serva o blokování pohybu pro danou osu [BIT]
.Error	flEdrvError0,1,.	[out] Pohon je v chybě [BIT]
.Init	flEdrvInit0,1,..	[in] Žádost o inicializaci pohonu [BIT]
.On	flEdrvOn0,1,..	[in] Žádost o zapnutí pohonu (nastaví bity QSTOP, VOLT, ON) [BIT]
.Enable	flEdrvEnable0,1,..	[in] Žádost Enable pohonu (nastaví ENABLE) [BIT]
.Disable	flEdrvDisable0,1,..	[in] Žádost Disable pohonu (vynuluje ENABLE) [BIT]

.Done	flEdrvDone0,1,..	[in] Žádost ukončení (vynuluje control word) [BIT]
.Erase	flEdrvErase0,1,..	[in] Žádost o mazání chyb pohonu [BIT]
.ScaleSpeed	rEdrvScaleSpeed0,1,..	[in] Měřítko pro zadání rychlosti [REAL]
	flEdrvSimul0,1,..	[in] Pohon v simulaci [BIT]
Prvky pro logické spoje **RTM.Servo[x].**	Rozhraní pro PLC	Popis
.ICounter	DIFCIT_X,+4,+8,..	Odchylka polohy [INT32]
.ServoReqPosition	ServoReqPosition,+4,..	Požadovaná poloha serva [INT32]
.ServoPosition	ServoPosition,+4,..	Odměřování serva [INT32]
.ServoPosition_Inc	SERVO_POSITION,..	Odměřování serva [INT32 inc]
Prvky pro logické spoje **RTM.Servo[x].Enc**	Rozhraní pro PLC	Popis
.CounterValue	dwEdrvEncCounterValue0,1	[in] Čítač externího odměřování (jednotky EL5101) [INT32 inc]
.LatchValue	dwEdrvEncLatchValue0,1,.	[in] Buffer odměřování pro zachycení stavu v okamžiku NI [INT32 inc]
.LatchValid	flEdrvEncLatchValid0,1,.	[in] Zachycena hodnota v Bufferu je platná (EL5101) [BIT]
.LatchEn	flEdrvEncLatchEn0,1,.	[out] Povolení zachytávání nové hodnoty do Bufferu (EL5101) [BIT]

20

20. RUČNÍ POJEZDY

20.1 Všeobecný popis

Ruční pojezdy není samostatný režim systému. Pomocné ruční pojezdy se takto jeví jako okamžitý přechod do režimu MAN, ale bez změny režimu. Z toho je patrné, že nejvýznamnější využití pomocných ručních pojezdů bude v režimech AUT, AUT po stopu, AUT – BB. Využije se ale i rychlé operativní popojíždění například v režimu CA (centrální anulace).

Pro pomocné ruční pojezdy budeme používat označení: „**AUTMAN**“.

Při ukončení pomocných ručních pojezdů ve stopnutém režimu AUT nebo v režimu AUT -BB máme na výběr tyto způsoby :

- Provede se návrat na dráhu v režii pomocných ručních pojezdů a automatický režim po opětovném startu pokračuje přesně podle programované dráhy.
- Pomocné ruční pojezdy se použijí jen pro odjetí z místa stopu (například s točícím se vřetenem). Potom následuje **centrální anulace** a opětovný start programu pomocí **volby bloku,** nebo zrychlenou volbou **cont.**

Pro řízení ručních pojezdů možno využít sadu systémových příkazů „**Command**“ a sadu příkazů pro reálný čas „**RtmCommand**“, které jsou přímo definované v definičních souborech klávesnic typu „**KbdConfig**“. Popis příkazů a jejich použití je popsáno v kapitole: **„Ovládací panely“.**

V této kapitole bude popsán i způsob nezávislého posouvání dráhy, tzv režim „**SHIFT**“.

20.2 Bitové signály pro PLC program

PLC program má pro pomocné ruční pojezdy a nezávislé posouvání dráhy k dispozici tyto bitové signály:

20.2.1 Základní bitové signály

Základní bitové signály pro pomocné ruční pojezdy slouží pro PLC program na určení pohybu, směru pohybu, povolení pohybu, návratu do místa stopu a pod.

- Bit **ACK_AUTMAN** definovaný v BZH00. Hodnota log.1 signalizuje, že je požadován a kazetou byly potvrzeny pomocné ruční pojezdy (AUTMAN). Hodnota log.1 trvá po celou dobu aktivace pomocných ručních pojezdů.

- Bity **PO_OSxPI** definované v PB20PI se dynamicky nahazují na hodnotu log.1 při požadavku na pohyb v dané ose a při ukončení pohybu přejdou na hodnotu log.0. Bity se nahodí na hodnotu log.1 už při požadavku na pohyb i v případě, že je pohyb zakázán například od signálu MPxPI. Tyto bity proto může PLC program využít na zapínání vazby, řazení spojek a uvolňování os (pokud se nutno provádět) a až po provedení akce povolit příslušný pohyb. Bity PO_OSxPI není možno využít pro řízení uvolňování os tehdy, když požadujeme pohyb od ručního točítka bez předchozího pohybu v dané souřadnici. Na řízení uvolňování os pro točítko je možno využít bitů AUTMAN_CONT_SELECT a MM_x0, jak o tom bude popsáno dále.

- Bity **SM_POxPI** definované v PB20PIS určují směr pojezdu a jsou platné v okamžiku nahození signálů PO_OSxPI ještě před startem pohybu. Hodnota log.0 určuje kladný směr a hodnota log.1 určuje záporný směr pohybu. Tyto bity možno využít například pro řazení směrových spojek.

- Bity **MPxPI** definované v BZH08PI mohou sloužit pro povolování pohybu stejně jako u standardních pohybů systému. Podrobně je jejich význam popsán v PLC návodu kapitole " Důležité bitové proměnné pro PLC program". Bity MPxPI slouží pro povolování pohybu jen v případě, když 4.dekáda strojní konstanty R233 je nastavena na hodnotu 1. Tehdy na povolení pohybu bity MPxMAN nemají vliv.

- Bity **MPxMAN** definované v BZH08MAN slouží speciálně pro povolování pomocných ručních pohybů, podobně jako bity MPxPI, ale jen v případě, že 4.dekáda strojní konstanty R233 je nastavena na hodnotu 2. V tomto případě povolení pohybu nezávisí od bitů MPxPI,ale jen od povolovacích bitů MPxMAN určených pro pomocné ruční pojezdy.

- Bit **INPOS_STOP** definovaný v EXT_CONT_AUTMAN3 svou hodnotou log.1 určuje, že systém je v poloze posledního stopu. Signál je definovaný jen v režimu AUT a ve všech jiných režimech má hodnotu log.1. Při pomocném ručním pojezdu v libovolné ose se signál INPOS_STOP nastaví na hodnotu log.0. Pomocí tohoto signálu může PLC program například blokovat opětovný START programu, pokud nebudou všechny souřadnice v poloze po stopu. Signál se nastaví na hodnotu log.1 opět tehdy, když bude proveden návrat na dráhu (v režii pomoc.ručních pojezdů) pro všechny osy.

- Bit **EN_AUTMAN** definovaný v BZH08MAN svou hodnotou log.1 povoluje pomocné ruční pojezdy. Bit je přednastaven na hodnotu log.1 při startu systému a dál je plně k dispozici pro PLC program. PLC program může pomocí tohoto bitu zakázat aktivaci pomocných ručních režimů.

- Bit **REQ_ESHIFT** je nastaven na hodnotu „1" po celou dobu zvoleného režimu nezávislého posouvání dráhy „SHIFT". PLC program může číst i nastavovat (viz dále).

- Bit **ACT_ESHIFT** je nastaven na hodnotu „1" po celou dobu aktivace posouvání dráhy „SHIFT". PLC program může číst i nastavovat (viz dále).

20.2.2 Informační signály

Informační bitové signály jsou určeny jen pro čtení. PLC program si může na základě nich zjistit způsob řízení pomocných ručních pojezdů z panelu systému nebo z panýlku točítka.

- Bit **AUTMAN_CONT_NC** definovaný v MAN_NC svou hodnotou log.1 informuje PLC program, že řízení pomocných ručních pojezdů se provádí z panelu systému.

- Bit **AUTMAN_CONT_TOC** definovaný v CONT_AUTMAN2 svou hodnotou log.1 informuje PLC program, že řízení pomocných ručních pojezdů se provádí z panýlku točítka.

- Bit **AUTMAN_CONT_SELECT** definovaný v CONT_AUTMAN svou hodnotou log.1 informuje PLC program, že se provádí předvolba pohybu pro pomocné ruční pojezdy z panelu systému nebo z panýlku točítka. Například, když se na panýlku točítka stiskne souřadnice, pohyb se nevykoná, ale příslušná souřadnice se předvolí.

- Bity **MM_x0** a **MM_x1** definované v AX_AUTMAN informují PLC program o řízení ručních pojezdů. Bity MM_X0, MM_Y0, .. ,MM_60 určují požadavek na pohyb v kladném směru a bity MM_X1, MM_Y1, .. ,MM_61 určují požadavek na pohyb v záporném směru.

- Bit **AUTMAN_CONT_G00** definovaný v CONT_AUTMAN svou hodnotou log.1 informuje PLC program, že při řízení pohybu z panelu systému nebo z panýlku točítka je právě požadován rychloposuv. Každá souřadnice pojede svým vlastním rychloposuvem, který je zadán v konfiguraci.

- Buňka **SHIFT_CONTROL** Jednotlivé bity jsou nastaveny na hodnotu „1“ při aktivním posunu konkrétní souřadnice (viz dále).

- Pole **SHIFT_x** typu DWORD. Aktuální hodnota nezávislého posunutí pro jednotlivé souřadnice [1/8 μm].

20.2.3 Přímo sdílené proměnné

Přímo sdílené proměnné typu „**RtmVar**“ slouží pro předání stavu ručních pojezdů a režimu „SHIFT“ pro indikaci přímo do HTML oken a dialogů.

- Pole **KnobSelAx0, KnobSelAx1,..** typu BYTE slouží pro indikaci předvolené osy pro ruční pojezdy a pro nezávislé posouvání dráhy. Na základě hodnoty v buňce se například mohou měnit obrázky pro indikaci v HTML okně (viz příklad dále). Každá buňka indikuje stav jedné NC osy a může nabývat hodnot:
  0 = osa nevyvolena pro ruční pojezdy
  1 = osa je předvolena pro ruční pojezdy a případně pro posun z točítka
  2 = osa je předvolena pro nezávislý posun „SHIFT“

- Pole **Pos6ShiftAx0, Pos6ShiftAx1,..** typu REAL slouží pro indikaci aktuálního nezávislého posunutí pro jednotlivé osy „SHIFT“ [mm].

- Symbol **ReqShift** je přímé sdílení bitu REQ_ESHIFT

- Symbol **ActShift** je přímé sdílení bitu ACT_ESHIFT

- Symbol **KnobStep** je přímé sdílení buňky KNOB_STEP typu WORD.

- Symbol **ShiftEnable** je přímé sdílení buňky SHIFT_ENABLE typu BYTE

20.2.4 Externí řízení ručních pojezdů z PLC

Bity pro externí řízení pohybu v pomocných ručních pojezdech slouží pro zadávání pohybu, směru pohybu, rychlosti pohybu.

- Bity **EXM_x0 a EXM_x1** definované v EXT_CONT_AUTMAN slouží pro zadávání pohybu a směru pohybu v pomocných ručních pojezdech. Bit EXM_X0 při nastavení na hodnotu log.1 je požadavkem na pohyb v ose X v kladném směru. Bit EXM_X1 při nastavení na hodnotu log.1 je požadavkem na pohyb v ose X v záporném směru. Nastavení bitů EXM_x0 a EXM_x1 je platné jen tehdy, když je bit REQ_EXT_CONT_AUTMAN nastaven na log.1 (bude pojednáno dále). V opačném případě na hodnotách bitů EXM_x0 a EXM_x1 nezáleží. Pro požadavky na pohyb může být nastaveno i víc bitů najednou.

- Bit **REQ_EXT_G00_AUTMAN** definovaný v EXT_CONT_AUTMAN je externí požadavek pro rychloposuv. Když PLC program požaduje posun v pomocných ručních pojezdech rychloposuvem, nastaví bit REQ_EXT_G00_AUTMAN na hodnotu log.1. Rychloposuv pro každou souřadnici je určen v konfiguraci. Když má tento bit hodnotu log.0, souřadnice se pohybují rychlostí podle atributu „Feed" nebo rychlostí zadanou externě z PLC programu. V každém případě se rychlost ovlivňuje pomocí potenciometru procenta F nebo externím řízením procenta rychlosti z PLC programu.

- Bit **REQ_EXT_SELECT_AUTMAN** definovaný v EXT_CONT_AUTMAN je externí požadavek na předvolbu pohybu v pomocných ručních pojezdech. PLC program hodnotou log.1 v bitu REQ_EXT_SELECT_AUTMAN a nastavením příslušného bitu EXM_x0 předvolí pohyb v dané souřadnici. Osa se nerozjede, ale bude předvolena pro následující pohyb zadávaný od ručního kolečka.

- Bit **REQ_EXT_CONT_AUTMAN** definovaný v EXT_CONT_AUTMAN3 je požadavek na externí řízení pohybu v pomocných ručních pojezdech. Když je bit nastaven na hodnotu log.0, zadávání souřadnic a směru pohybu se provádí z panelu systému nebo z panýlku točítka (interní řízení pohybu). V tomto případě na nastavení bitů EXM_x0 a EXM_x1 nezáleží. Když je bit REQ_EXT_CONT_AUTMAN nastaven na hodnotu log.1, zadávání souřadnic a směru pohybu se provádí externě z PLC programu pomocí bitů EXM_x0 a EXM_x1 (externí řízení pohybu). V tomto případě se nedá pohyb ovlivnit z panelu systému nebo z panýlku točítka. Bit je po zapnutí systému přednastaven na hodnotu log.0.

- Bit **REQ_EXT_FEED_AUTMAN** definovaný v EXT_CONT_AUTMAN3 je požadavek na externí řízení rychlosti pohybu v pomocných ručních pojezdech. Když je bit nastaven na hodnotu log.0, zadávání rychlosti pohybu se provádí z panelu systému (interní řízení rychlosti). V tomto případě na nastavení buňky EXT_FEED_AUTMAN nezáleží. Když je bit REQ_EXT_FEED_AUTMAN nastaven na hodnotu log.1, zadávání rychlosti pohybu se provádí externě z PLC programu pomocí buňky EXT_FEED_AUTMAN (externí řízení rychlosti). V tomto případě se nedá rychlost pohybu ovlivnit z panelu systému. Bit je po zapnutí systému přednastaven na hodnotu log.0.

- Buňka **EXT_FEED_AUTMAN** typu WORD slouží pro zadání externí rychlosti v pomocných ručních pojezdech z PLC programu. Takto zadaná rychlost se projeví, jen když je bit REQ_EXT_FEED_AUTMAN nastaven na hodnotu log.1. Hodnota rychlosti se zadává v binárním kódu v mm/min nebo v μm/ot. V každém případě se rychlost ovlivňuje pomocí potenciometru procenta F nebo externím řízením procenta rychlosti z PLC programu.

- Bit **REQ_EXT_FEED_G00_AUTMAN** je požadavek na externí řízení rychlosti pohybu v pomocných ručních pojezdech rychloposuvem. Když je bit REQ_EXT_FEED_G00_AUTMAN nastaven na hodnotu log.1, zadávání rychloposuvu se provádí externě z PLC programu pomocí buňky EXT_FEED_G00_AUTMAN (externí řízení rychloposuvu). Bit je po zapnutí systému přednastaven na hodnotu log.0.

- Buňka **EXT_FEED_G00_AUTMAN** typu WORD slouží pro zadání externí rychlosti pro rychloposuv v pomocných ručních pojezdech z PLC programu. Takto zadaná rychlost se projeví, jen když je bit REQ_EXT_G00_AUTMAN nastaven na hodnotu log.1. Hodnota rychlosti se zadává v binárním kódu v mm/min.

- Bit **G95_AUTMAN** je požadavek na otáčkový posuv. Když PLC program nastaví bit G95_AUTMAN na hodnotu log.1, jsou všechny rychlosti posuvů (interní i externí) v pomocných ručních pojezdech vykonány otáčkovým posuvem v µm/ot. Tento bit neovlivňuje rychloposuv signalizovaný v AUTMAN_CONT_G00 nebo zadaný v REQ_EXT_G00_AUTMAN. Bit G95_AUTMAN je po zapnutí systému přednastaven na hodnotu log.0.

- Bit **AUTMAN_REQ** definovaný v BZH00 je požadavek na aktivaci pomocných ručních pojezdů. Hodnota log.1 v bitu AUTMAN_REQ aktivuje pomocné ruční pojezdy, pokud jejich aktivace je v systému povolena. PLC program se o aktivaci pomocných ručních pojezdů může přesvědčit pomocí signálu ACK_AUTMAN. Když PLC program vyžaduje aktivaci pomocných ručních pojezdů, musí držet bit AUTMAN_REQ na hodnotě log.1 po celou dobu požadované aktivace. Pokud byly pomocné ruční pojezdy aktivovány z PLC programu a bit AUTMAN_REQ se shodí na hodnotu log.0, budou pomocné ruční pojezdy deaktivovány.

- Bit **AUTMAN_SELECT** definovaný v AUTMAN_SEL je požadavek na aktivaci pomocných ručních pojezdů. Hodnota log.1 aktivuje pomocné ruční pojezdy, pokud jejich aktivace je v systému povolena. PLC program jen nastavuje žádost hodnotou log.1 a systém po převzetí žádosti bit vždy vynuluje, a to bez ohledu na úspěšnost žádosti.

- Bit **AUTMAN_CANCEL** definovaný v AUTMAN_SEL je požadavek na deaktivaci pomocných ručních pojezdů. Hodnota log.1 zruší pomocné ruční pojezdy, pokud jejich deaktivace je v systému povolena (mohou být ještě jiné požadavky). PLC program jen nastavuje žádost hodnotou log.1 a systém po převzetí žádosti bit vždy vynuluje, a to bez ohledu na úspěšnost žádosti.

- Buňka **POS_SPACE_PLC** typu BYTE slouží pro určení prostoru, ve kterém má externí řízení pohybu probíhat. Systém standardně používá několik transformací, které jsou zařazeny na výstup interpolátoru. Jednotlivé transformace oddělují „**prostory**“, ve kterých je definovaná aktuální poloha (viz „Rozhraní CNC-PLC“).

- Buňka **POS_SPACE_MMM** typu BYTE slouží pro určení prostoru, ve kterém má probíhat pohyb řízený pomocí příkazů typu „RtmCommand“. Příkazy jsou konfigurovány například v souborech typu „*.KbdConfig“ (například RtmCommand="RTMID_IMMM_MOVEY", viz návod „Ovládací panely“).

- Bit **AUTMAN_KNOB1_DIS** definovaný v AUTMAN_SEL řídí povolení ovládání z panelu 1.točítka. Hodnota log.1 zakáže funkčnost tlačítek z 1.točítka.

- Bit **AUTMAN_KNOB2_DIS** definovaný v AUTMAN_SEL řídí povolení ovládání z panelu 2.točítka. Hodnota log.1 zakáže funkčnost tlačítek z 2.točítka.

- Bit **AUTMAN_WHEEL_DIS** definovaný v AUTMAN_SEL2 zakáže automatickou volbu osy v ručních posuvech (RTMCommand) pro další ovládání točítkem.

- Bit **AUTMAN_KNOB_TOGGLE** definovaný v AUTMAN_SEL2 modifikuje způsob předvolby na ovládacím točítku. Příkazy RTMCommand (RTMID_MMM_SEL..) pro předvolbu os střídavě aktivují a dezaktivují možnost ovládání os z panýlku točítka.

- Buňka **KNOB_STEP** typu WORD slouží pro zadání kroku točítka.

- Buňka **SHIFT_ENABLE.** Jednotlivé bity slouží pro povolení nezávislého posouvání dráhy (viz dále).

20.3 Nezávislé posouvání dráhy

Jedná se o možnost nezávislého posouvání dráhy pomocí ručního kolečka i v průběhu jetí programu. Posun je k dráze připočten až na výstupu z interpolátoru a tak neovlivní průběh programu, všechny typy posunutí dráhy a ani interpolaci v rámci bloku.

Pro nezávislé posouvání dráhy budeme používat označení: „**SHIFT“**.

20.3.1 Volba režimu a aktivace posouvání dráhy

Volba režimu posouvání se provádí například pomocí tlačítka panelu systému, které má v definičním souboru typu „KbdConfig“ nastaven RtmCommand="**RTMID_MMM_SHIFT_REQ**", viz návod: „Ovládací panely“.

Volbu režimu posouvání SHIFT lze provést i z PLC nastavením bitu **REQ_ESHIFT** na hodnotu log.1

Zrušení režimu posouvání se provádí pomocí stejného příkazu, jako pro volbu posouvání. Při zrušení režimu posouvání dráhy systém odjede na původní místo před posunem dráhy.

Aktivace posunu dráhy se může provést například pomocí tlačítka panelu systému, které má v definičním souboru typu „KbdConfig“ nastaven RtmCommand="**RTMID_MMM_SHIFT_ACT".**

Aktivaci posouvání SHIFT lze provést i z PLC nastavením bitu **ACT_ESHIFT** na hodnotu log.1.

V aktivním stavu posouvání dráhy je možno volit osu pro posun pomocí standardních příkazů předvolby. (například RtmCommand="RTMID_MMM_SEL00"), viz návod: „Ovládací panely“.
Pomocí ručního kolečka je možno aktuální posun kdykoli měnit.

Po posunutí dráhy je vhodné aktivaci zrušit stejným příkazem, jak byla aktivace zvolena. Ruční točítko přestane dráhu posouvat a zruší se poslední volba osy. Všechny aktuální posuny dráhy zůstanou nadále platné.

Na obrazovce systému je vhodné indikovat:

- režim SHIFT je aktivní
- Která osa se právě posouvá
- Aktuální posun pro všechny osy

Pro snadnou tvorbu HTML oken slouží přímo sdílené systémové proměnné (už bylo popsáno) :

- KnobSelAx0,..
- Pos6ShiftAx0,..
- ActShift

Příklad tvorby HTML okna bude uveden dále.

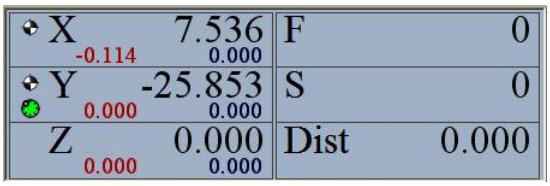
20.3.2 Informace pro PLC pro nezávislý posun

PLC program má k dispozici několik signálů, které se týkají nezávislých posunů dráhy:

Název	Typ	Popis
REQ_ESHIFT	BIT	PLC program může číst a nastavovat. Bit je nastaven na hodnotu "1" po celou dobu zvoleného režimu posouvání.
ACT_ESHIFT	BIT	PLC program může číst a nastavovat. Bit je nastaven na hodnotu "1" po dobu aktivace posunu dráhy v platném režimu posouvání.
SHIFT_CONTROL	BYTE	PLC program může jen číst. Jednotlivé bity jsou nastaveny na hodnotu "1" při aktivním posunu konkrétní souřadnice. Bity jsou přiřazeny osám postupně od nejnižší váhy podle pořadí definice os (bit 0=1.osa, bit 1=2. osa, …). Hodnota v buňce SHIFT_CONTROL je platná pouze, když je bit ACT_ESHIFT nastaven na hodnotu "1".
SHIFT_ENABLE	BYTE	PLC program může nastavovat. Jednotlivé bity jsou svou hodnotu "1" povolují pohyb při aktivním posunu pro konkrétní souřadnice. Bity jsou přiřazeny osám postupně od nejnižší váhy podle pořadí definice os (bit 0=1.osa, bit 1=2. osa, …). Proměnná SHIFT_ENABLE je přednastavena na hodnotu 3Fh.
SHIFT_x	DWORD	PLC program může jen číst. Pole 6 double-wordů. Aktuální hodnota posunutí pro jednotlivé souřadnice v osminách mikronu

20.4 Příklad tvorby okna pro ruční režimy

Dále uvedeme příklad tvorby okna pro indikaci předvolby osy točítka a nezávislého posunu dráhy. Upraví se například okno „MAIN“ (soubor MAIN.HTML) tak, aby zeleným kolečkem před názvem souřadnice se indikovala předvolená osa pro točítko a červeným kolečkem osa, která má aktivní nezávislý posuv (SHIFT). Červený údaj udává o kolik je osa posunuta v režimu SHIFT. Kompletní řešení je uvedeno ve vzorovém projektu PLC pro školení.




```
<!-- Výpis názvu souřadnice -->
<DIV id="AxisX_Lbl" class="LabelBig">X</DIV>

<!-- Výpis aktuální polohy -->
<DIV id="AxisX_Pos"   class="ValueBig" CNCType="AsyncIndicatorPainter"
  AIPLibrary="StdPlugins" AIPType="Tool" AIPOptions="Channel: 0; Axis: X;
  ValType: Pos0;">+0000.000</DIV>

<!-- Výpis značky reference -->
<DIV id="AxisX_InRef" class="ValueMedium" CNCType="AsyncIndicatorPainter"
  AIPLibrary="StdPlugins" AIPType="Axis" AIPOptions="Channel: 0; Axis: 0;
  ValType: InRef; GraphType: Image; ImgWidth: 15; ImgHeight: 15;">*</DIV>

<!-- Výpis distance -->
<DIV id="AxisX_Dist"  class="ValueMedium" CNCType="AsyncIndicatorPainter"
  AIPLibrary="StdPlugins" AIPType="Axis" AIPOptions="Channel: 0; Axis: 0;
  ValType: Distance;">+0000.000</DIV>

<!-- Výpis značky předvolené osy točítka a SHIFT -->
<DIV id="AxisX_Knob" CNCType="AsyncIndicatorPainter"
  AIPLibrary="StdPlugins" AIPType="RtmVar" AIPOptions="Channel: 0;
  Variable: 'KnobSelAx0'; GraphType: Image; Img0: KnobSel0.png; Img1:
  KnobSel1.png; Img2: KnobSel2.png;">*</DIV>

<!-- Výpis aktuálního posunutí SHIFT -->
<DIV id="AxisX_Shift" class="ValueMedium" CNCType="AsyncIndicatorPainter"
  AIPLibrary="StdPlugins" AIPType="RtmVar" AIPOptions="Channel: 0;
  Variable: 'Pos6ShiftAx0'; NumberPrecision: 3;">+00.000</DIV>
```

V hlavičce HTML souboru je definován „stylopis“ pro každý prvek. Například pro výpis značky předvolby točítka je:

```
<STYLE type="text/css">
 #AxisX_Knob, #AxisY_Knob, #AxisZ_Knob
   {position: absolute; left: 9px; top: 24px; width: 16px; height: 16px; }
</STYLE>
```

20.5 Některé možnosti řízení z PLC programu

Krátce pojednáme o jednotlivých možnostech řízení pomocných ručních pojezdů z PLC programu.

20.5.1 Příklad řízení, když jsou všechny souřadnice trvale v polohové vazbě

V případě, že na stroji jsou všechny souřadnice trvale v polohové vazbě, je situace velmi jednoduchá. PLC program nemusí mít žádnou podporu pro pomocné ruční pojezdy.
V případě, že je potřeba uvolňovat a upínat osy nebo ovládat spojky pro souřadnice, můžeme si vybrat, zda budeme pro pomocné ruční pojezdy blokovat pohyb pomocí signálů **MPxPI** nebo **MPxMAN**. Výběr provedeme pomocí nastavení 4.dekády strojní konstanty R233, jak už bylo popsáno výše.

20.5.2 Příklad řízení pro 1 osu s upínáním bez panýlku s ručním točítkem

Příklad pro řízení osy X v části `MODULE_MAIN`, pro blokování pohybu použijeme signál `MPxMAN` (4. dekáda strojní konstanty R233 je 2). Bity `MPxMAN` jsou v základním stavu v hodnotě log.**0**. V tomto případě není použitý panýlek s ručním točítkem. Ovládání můžeme napsat v části `MODULE_MAIN`:

```
      ....
      LDR   ACK_AUTMAN        ;test aktivace pomocných ručních pojezdů
      LA    PO_OSXPI          ;je požadován pohyb v ose X ?
      FL1   1,MCH_MAN_X       ;podmíněná aktivace mechanizmu pohybu X
      ....
```

```
MECH_BEGIN MCH_MAN_X          ;definice mechanizmu pro řízení pohybu X
      ....
      ....  uvolnění osy , zapnutí vazby ,….
      ....
      FL    1,MPXMAN          ;povolení pohybu v ose X pro ruční pojezdy
      EX
      LDR   PO_OSXPI
      LA    ACK_AUTMAN
      EX1                     :čekání a ukončení pohybu pro ruční pojezdy
      FL    0,MPXMAN          ;zákaz pohybu pro ruční pojezdy
      ....
      ....  upnutí osy, vypnutí vazby ,…..
      ....
MECH_END  MCH_MAN_X
```

20.5.3 Příklad pro externí řízení pohybu pomocí potenciometrů

Na stroji jsou potenciometry pro zadávání rychlosti pohybu a přepínače pro volbu směru pohybu.

Vstupy z panelu stroje:

```
            ;X_SLOW_I         pomalý stupeň
            ;X_RAPID_I        rychlý stupeň
            ;X_ON_I           potenciometr zapnut
            ;X_OFF_I          potenciometr vypnut
            ;X_MINUS_I        záporný směr
            ;X_PLUS_I         kladný směr
POT10: DFM  X_SLOW_I,X_RAPID_I,X_ON_I,X_OFF_I,X_MINUS_I,X_PLUS_I,,

            ;minulé stavy a příznaky
POT11: DFM  X_SLOW_M,X_RAPID_M,X_ON_M,X_OFF_M,X_MINUS_M,X_PLUS_M,,


ANALOG_POT_X:     DS    4
POT_FEED_X:       DS    4



;Mechanizmus pro test volby směru pro potenciometry
MECH_BEGIN  MechPotencSmerX
      fl    1,MechPotencFeedX            ;zadávání rychlosti pot.
      fl    1,MechPotencStopX            ;stop pohybu pot.
PotencSmerX:
      ex
      ldr   MechPotencStopX              ;čeká na konec případného stopu
      ex1
      ldr   X_MINUS_I                    ;čeká na volbu směru
      lo    X_PLUS_I
      ex0
      ldr   X_MINUS_I                    ;nesmi byt oboje
      la    X_PLUS_I
      jl1   PotencSmerX

      ldr   X_MINUS_I                    ;paměť směru
      wr    X_MINUS_M
      ldr   X_PLUS_I
      wr    X_PLUS_M
      fl    1,MechPotencPohybX           ;mohou se testovat podmínky pohybu
      ex
      ldr   X_MINUS_I                    ;kontroluje změnu směru při běhu
      lx    X_MINUS_M
      ldr   X_PLUS_I
      lx    X_PLUS_M
      lo
      lo    -MechPotencPohybX
      ex0
      fl    1,MechPotencStopX            ;stop pohybu pot.
      jum   PotencSmerX
MECH_END    MechPotencSmerX
```

```
;Externí zadávání rychlosti pro potenciometry
MECH_BEGIN  MechPotencFeedX
PoteFeedC:
      ex
      ldr   MechPotencStopX              ;čeká na konec případného stopu
      ex1
      lod   cnst.100
      ldr   X_SLOW_I                     ;pomalá rychlost
      wr    X_SLOW_M
      sto1  POT_FEED_X
      lod   cnst.5000
      ldr   X_RAPID_I                    ;velká rychlost
      wr    X_RAPID_M
      sto1  POT_FEED_X
      ex
      lod   POT_FEED_X
      mulb  ANALOG_POT_X                 ;sejmuta analog.hodnota pot.
      divb  cnst.1000
      sto   EXT_FEED_AUTMAN              ;externí zadání rychlosti
      ldr   X_SLOW_I                     ;test změny rychlosti
      lx    X_SLOW_M
      ldr   X_RAPID_I
      lx    X_RAPID_M
      lo
      lo    -MechPotencPohybX
      ex0
      fl    1,MechPotencStopX            ;stop
      jum   PoteFeedC
MECH_END    MechPotencFeedX

;Stop pohybu potenciometru
MECH_BEGIN  MechPotencStopX
      fl    0,EXM_X0,EXM_X1              ;konec pohybu
      mech_init MechPotencPohybX
      ldr   PrPosXVyp                    ;čeká na vytočení pot. do nuly
      la    -X_ON_I
      ex0
MECH_END    MechPotencStopX


;Test podmínek pohybu pro potenciometry
MECH_BEGIN  MechPotencPohybX
      ldr   X_ON_I                       ;čeká na pootočení pot.
      la    -X_OFF_I
      ex0
      fl    1,AUTMAN_REQ                 ;požadavek na AUTMAN
      fl    1,REQ_EXT_CONT_AUTMAN,REQ_EXT_FEED_AUTMAN      ;externí řízení
      ex
      ldr   X_PLUS_M                     ;aktivace pohybu !
      wr    EXM_X0
      ldr   X_MINUS_M
      wr    EXM_X1
      bex
      ldr   X_ON_I                       ;testuje pootočení pot.
      la    -X_OFF_I
      ex1
      fl    1,MechPotencStopX
MECH_END    MechPotencPohybX
```

```
;Konec režimu pro potenciometry
MECH_BEGIN  MechPotencEnd
      fl    0,EXM_X0,EXM_X1              ;konec pohybu
      fl    0,AUTMAN_REQ
      fl    0,REQ_EXT_CONT_AUTMAN,REQ_EXT_FEED_AUTMAN
      mech_init MechPotencSmerX
      mech_init MechPotencPohybX
      mech_init MechPotencFeedX
      mech_init MechPotencStopX
MECH_END    MechPotencEnd
```

15

15. REFERENCE

Systém podporuje různé typy referencí. Referenci může provést jak systémová část software, tak PLC program.

15.1 Konfigurace pro referenci

O metodě zreferování souřadnice rozhoduje v konfiguraci pro danou NC osu atribut „RefMethod". Metoda zreferování se uplatní pro referenci typu „single" nebo pro referenci z PLC, která se provádí pomocí instrukce SPI_AX nebo příkazem HOMING_START_REQ.

element **NCAxis**		**konfigurace NC osy**	
	atribut **RefMethod**	**typ reference**	
		1	Nájezd do reference provede PLC.
		5	Reference pro kódovaná pravítka (Heidenhain, Essa)
		9	Způsob „rychlý nájezd do reference" pomocí polohovacích jednotek (default)
		10	Reference na spínače pomocí polohovacích jednotek s reverzací
		11	Reference na spínače pomocí polohovacích jednotek bez reverzace
		12	Reference na spínače s reverzací a s rovnáním
		13	Simulace reference (nastavení nulového bodu)
		14	Reference s absolutním odměřováním
		15	Reference na spínače s reverzací a s rovnáním - postupné zastavování

Volba v konfiguraci „RefViaPlc", zda referovat vždy prostřednictvím PLC. Pokud je nastaveno, systém nikdy nespouští referenci sám, ale žádá o ni PLC pomocí bitu HOMING_SINGLE. PLC může (po vlastní přípravě) požádat o provedení vlastní reference systém nastavením příslušného bitu v HOMING_START_REQ a reference se provede způsobem podle konfigurace RefMethod (RefMethod nesmí být typu PLC ... 1)

element **NCAxis**		**konfigurace NC osy**	
	atribut **RefViaPlc**	**referovat vždy prostřednictvím PLC**	
		0	Referenci může provést systém (default)
		1	Reference se provede vždy prostřednictvím PLC

Volba v konfiguraci „RefSwitch“, zda při nájezdu testovat referenční spínač.

element **NCAxis**		**konfigurace NC osy**	
	atribut **RefSwitch**	**referenční spínač**	
		0	Referenční spínač netestovat
		1	Referenční spínač testovat hranově (default)
		2	Referenční spínač testovat úrovňově

Volba v konfiguraci „SlowDownSwitch“, zda při nájezdu testovat zpomalovací referenční spínač.

element **NCAxis**		**konfigurace NC osy**	
	atribut **SlowDownSwitch**	**zpomalovací spínač**	
		0	Zpomalovací spínač netestovat (default)
		1	Zpomalovací spínač testovat

Volba v konfiguraci „InvertRefDirection“, která určuje směr nájezdu na referenční spínač.

element **NCAxis**		**konfigurace NC osy**	
	atribut **InvertRefDirection**	**směr nájezdu do reference**	
		0	Do reference se najíždí v kladném směru (default)
		1	Do reference se najíždí v záporném směru

Zadání rychlosti nájezdu do reference

element **NCAxis**		**konfigurace NC osy**	
	atribut **RefSpeed**	**Rychlost nájezdu do reference [mm/min]**	
		10000.0	Do reference se najíždí rychlostí 10000 mm/min (default)
		xxx	Rychlost nájezdu do reference
	atribut **RefSpeedLow**	**Rychlost posuvu při nájezdu do reference po nájezdu na referenční spínač, nebo po reverzaci [mm/min]**	
		50.0	Rychlost nájezdu do reference po referenčním spínači (default)
		xxx	Rychlost nájezdu do reference po zpomalení
	atribut **RefSpeedLowest**	**Nejnižší rychlost posuvu při nájezdu do reference (pro SLM pohony) [mm/min]**	
		0.0	Nejnižší rychlost nájezdu do reference
		xx	Nejnižší rychlost nájezdu do reference

Zadání pro přídavný odjezd po nájezdu do reference

element **NCAxis**		**konfigurace NC osy**	
	atribut **MoveToPosAfterRef**	**Přídavný odjezd po nájezdu do reference na zadanou pozici**	
		0	Nevykoná se přídavný odjezd (default)
		1	Po nájezdu do reference se vykoná přídavný odjezd relativně
		2	Po nájezdu do reference se vykoná přídavný odjezd absolutně
	atribut **PosAfterRef**	**Poloha, na kterou se má dojet bezprostředně po provedení reference [mm]**	
		0.0	Nulová poloha (default)
		xxx	Poloha, ne kterou se má odjet po nájezdu do reference

Volba nulového bodu stroje.

element **NCAxis**		**konfigurace NC osy**	
	atribut **MachineNullPoint**	**Nulový bod stroje**	
		0	V okamžiku referenční značky se dosadí nulová míra (default)
		xxx	Míra, která se dosadí v okamžiku nájezdu na referenční značku

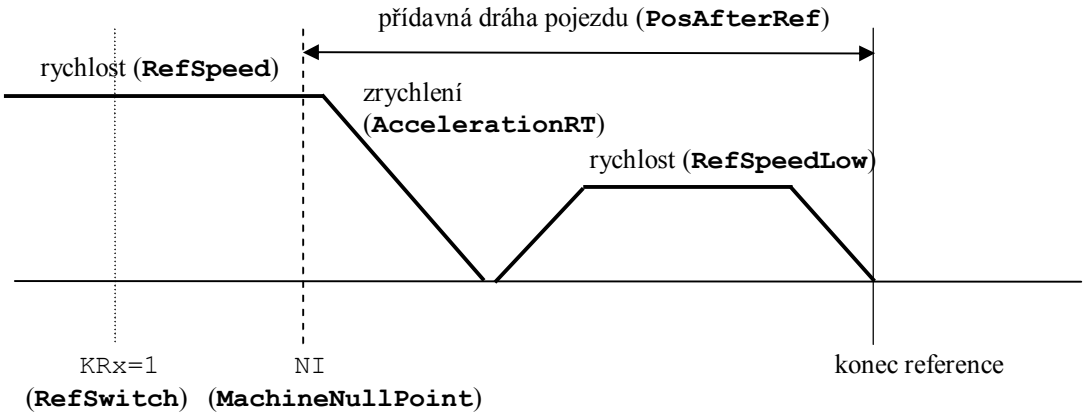
Konfigurace, zda je při nájezdu do reference povolen používat feed override a zda má být referenční blok obsloužen PLC programem jako normální blok.

element **NCAxis**		**konfigurace NC osy**	
	atribut **FeedOvrDuringRef**	**Je při nájezdu do reference povolen používat feed override?**	
		0	Rychlost nájezdu do reference ovlivňuje feed override (default)
		1	Rychlost nájezdu do reference neovlivňuje feed override
	atribut **RefBlckAsNormal**	**Má být referenční blok obsloužen PLC programem jako normální blok?**	
		0	Reference není obsloužena jako normální blok (default)
		1	Reference je obsloužena jako normální blok

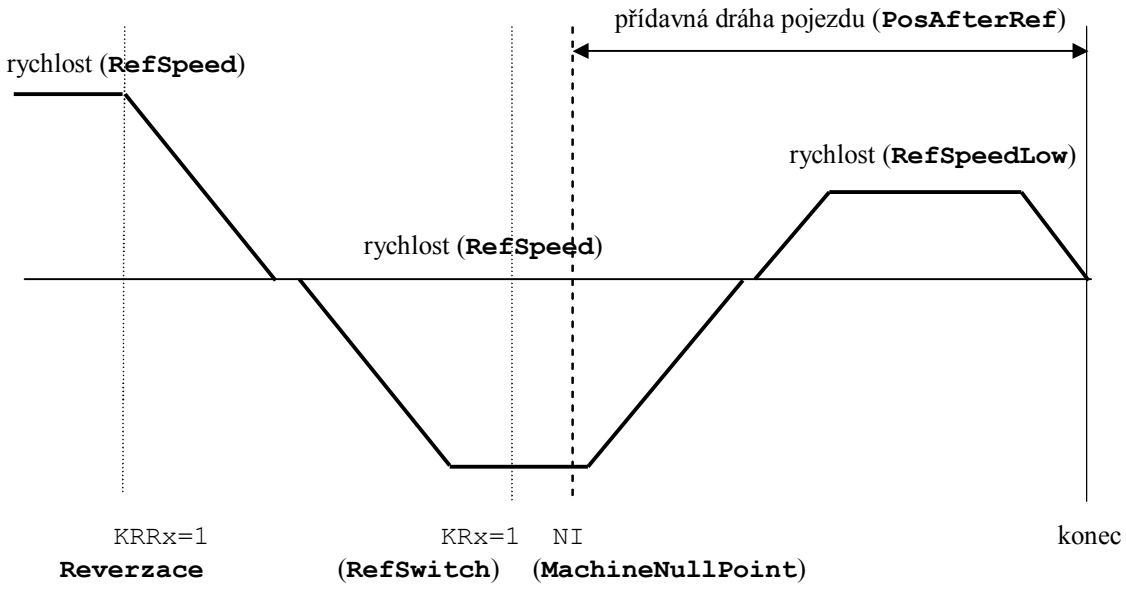
S referencí také souvisí nastavitelná vlastnost, zda NC osa vyžaduje pro jízdu v programu referenci.

element **NCAxis**		**konfigurace NC osy**	
	atribut **NeedRef**	**Vyžaduje osa referenci, aby mohla být ovládána z programu?**	
		0	Osa pro jetí z programu nevyžaduje referenci (default)
		1	Osa pro jetí z programu vyžaduje referenci
	atribut **AutNeedRef**	**Je pro start programu potřeba, aby byla osa v referenci?**	
		0	Osa pro start programu nevyžaduje referenci (default)
		1	Osa pro start programu vyžaduje referenci

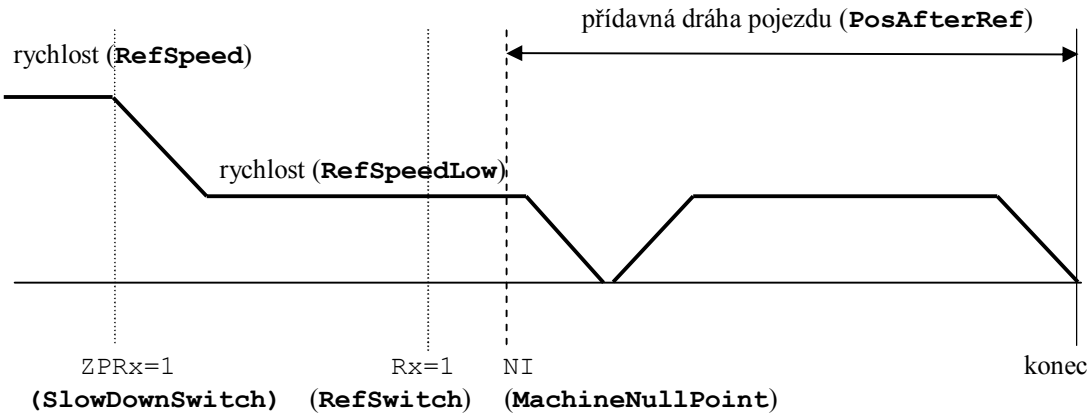
Znázornění referování souřadnice s přídavnou dráhou a bez reverzace (`RefMethod="9"`)



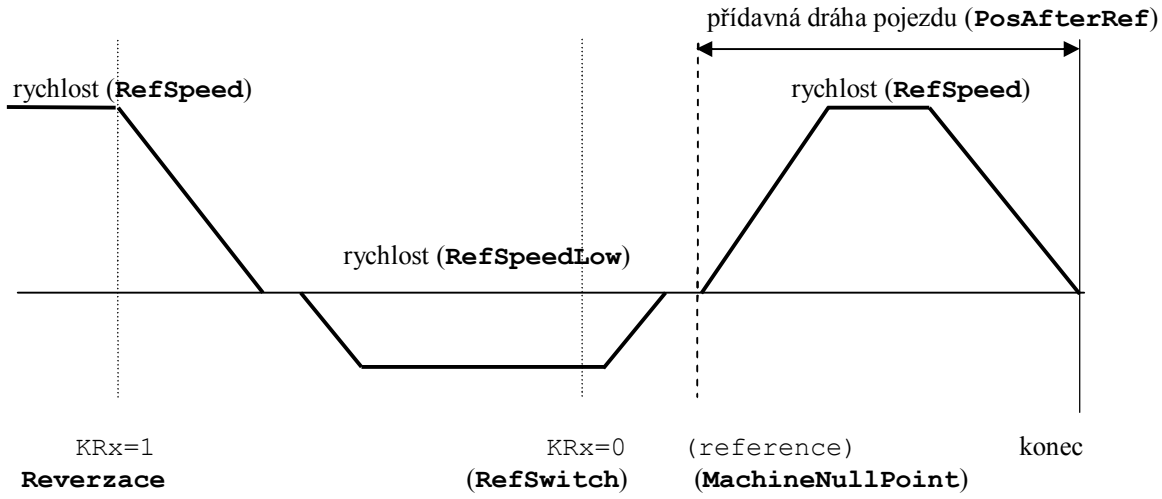
Znázornění referování souřadnice s přídavnou dráhou a s reverzací (`RefMethod="9"`)



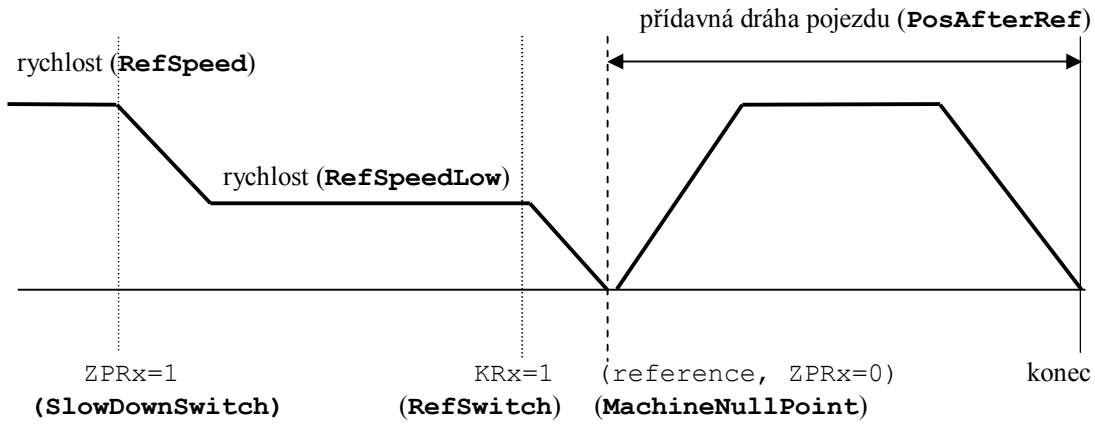
Znázornění referování souřadnice se zpomalovacím spínačem a s přídavnou dráhou (`RefMethod="9"`)



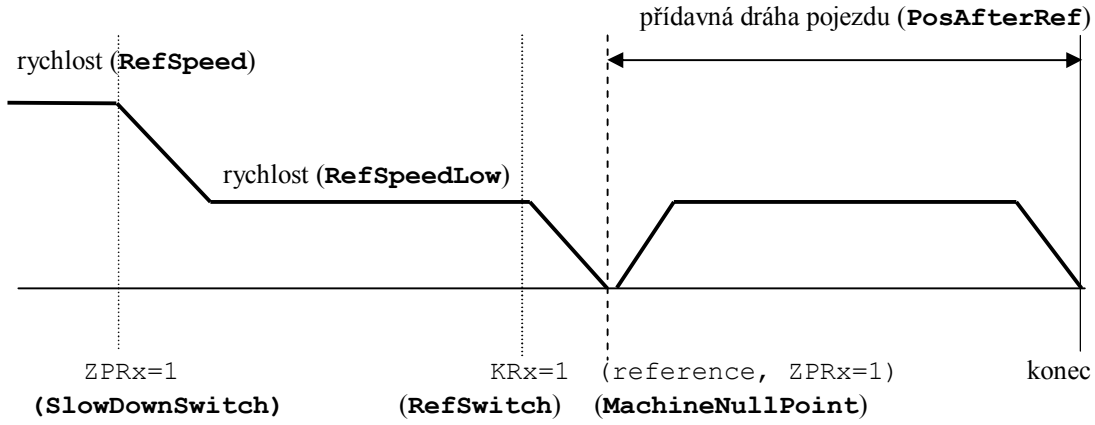
Znázornění referování souřadnice „jen na spínače“ s přídavnou dráhou a reverzací (`RefMethod="10"`)



Znázornění referování „na spínače“ se zpomalovacím spínačem a s přídavnou dráhou (`RefMethod="11"`)



Znázornění referování „na spínače“ se zpomalovacím spínačem a s přídavnou dráhou (`RefMethod="11"`)



15.2 Rozhraní pro referenci

Systém standardně používá několik transformací, které jsou zařazeny na výstup interpolátoru. Jednotlivé transformace oddělují „**prostory**“, ve kterých je definovaná aktuální poloha. Reference vždy musí probíhat v prostoru `POS5` nebo `POS6`. Systém po nastavení reference musí zabezpečit zpětnou transformaci nulového bodu do všech prostorů. Pokud referenci provádí PLC program, žádost o zpětnou transformaci se provede pomocí bitů `HOMING_REQ` (viz dále).

Transformace	Prostor
Interpolace (fiktivní poloha)	
	`POS0`
Programová transformace	
	`POS1`
Transformace polotovaru	
	`POS2`
Délková korekce	
	`POS3`
Posunutí 1	
	`POS4`
Posunutí 2	
	`POS5`
Transformace stroje (reálná poloha)	
	`POS6`

Přehled nastavovacích a informačních signálů rozhraní pro reference:

Proměnná	typ	akce	popis
`HOMING`	WORD bity os	čtení/zápis	Jednotlivé bity jsou příznaky pro platnou referenci NC os. Bity testuje systém v případě, že NC osa má v konfiguraci nastaveny atributy „`NeedRef`“ nebo „`AutNeedRef`“.
`HOMING_REQ`	WORD bity os	čtení/zápis	Jednotlivé bity jsou žádost o nastavení nulového bodu v prostoru `POS5` z PLC. Nulový bod se nastavuje podle atributu „`MachineNullPoint`“. Systém po nastavení provede všechny zpětné transformace.
`REFPOINT_DIST`	16x QWORD	čtení/zápis	Posunutí od nulového bodu stroje při zadání `HOMING_REQ` [1/64000 μm]. Nastavuje PLC.
`HOMING_START_REQ`	WORD bity os	čtení/zápis	Jednotlivé bity jsou žádost z PLC o vykonání kompletní reference pro jednotlivé osy. Po provedení reference systém bit vynuluje. Reference se provede podle aktuální konfigurace.
`HOMING_SINGLE`	bit	čtení/zápis	Žádost o referenci jedné osy v závislosti na konfiguraci „`RefMethod`“. Bit nastavuje systém.
`HOMING_PLC`	bit	čtení/zápis	Žádost o referenci jedné osy pro PLC. PLC bit pro převzetí vynuluje. Bit nastavuje systém.
`HOMING_GROUP`	bit	čtení/zápis	Žádost o skupinovou referenci pro PLC. PLC bit pro převzetí vynuluje. Bit nastavuje systém.
`SLS_REFER`	WORD bity os	čtení/zápis	Jednotlivé bity povolují test na softwarové limitní spínače. Každý bit slouží pro jednu NC osu.
`NlcAxisEnable`	WORD bity os	čtení/zápis	Jednotlivé bity povolují nelineární softwarové korekce a tepelnou kompenzaci. Každý bit slouží pro jednu řídicí NC osu pro nelineární korekce.
`NlcServoEnable`	WORD bity serv	čtení/zápis	Jednotlivé bity povolují nelineární softwarové korekce a tepelnou kompenzaci. Každý bit slouží pro jednu kompenzovanou servosmyčku. Systém bity po zapnutí přednastaví na hodnotu log.1.

KRx	16 bitů	čtení/zápis	referenční spínače
KRRx	16 bitů	čtení/zápis	reverzační spínače
ZPRx	16 bitů	čtení/zápis	zpomalovací referenční spínače

15.3 Způsoby reference

Typ reference (povel z menu)	**Způsob nájezdu do reference „RefMethod"**	**Popis**
„Single" reference jedné osy systém nastaví: **HOMING_SINGLE** <- 1	1 (pro RefViaPlc=1, HOMING_START_REQ Je RefMethod = 5,9,10,11,12,13,14,15)	**Referenci provede PLC**. systém nastaví: **HOMING_PLC** <- 1 **PB_HOMING_AXIS** <- bit osy (bit 0 – 15) PLC žádá ve vhodný okamžik o nastavení „nulového bodu do POS5 příslušným bitem v **HOMING_REQ.** Systém provede všechny zpětné transformace polohy. Po zreferování PLC nastaví příslušný bit v **HOMING.** PLC podle potřeby musí také povolit nelineární korekce a test na softwarové limitní spínače. PLC program po provedení vynuluje bit **HOMING_SINGLE**. Průběh reference ovlivňuje nastavení: MachineNullPoint
	5	**Reference pro kódovaná pravítka** (typ Heidenhain). Referenci provede systém. systém nastaví: **REFPI** <- 1 **PO_OSxPI** <- 1 pro příslušnou osu Po zreferování se nastaví příslušný bit v **HOMING.** Průběh reference ovlivňuje nastavení: RefSpeed, MachineNullPoint
	9 (default)	**Základní reference** (tzv.rychlá). Je možná reverzace pohybu. Referenci provede systém. systém nastaví: **REFPI** <- 1 **PO_OSxPI** <- 1 pro příslušnou osu Po zreferování se nastaví příslušný bit v **HOMING.** Průběh reference ovlivňuje nastavení: RefSwitch (KRx), reverzace (KRRx), SlowDownSwitch, RefSpeed, RefSpeedLow, MoveToPosAfterRef, PosAfterRef, FeedOvrDuringRef, InvertRefDirection,

		MachineNullPoint
	10	**Obyčejný nájezd na spínače s reverzací pohybu.** Je možné zpomalení rychlosti po reverzaci pohybu. Referenci provede systém. systém nastaví: **REFPI** <- 1 **PO_OSxPI** <- 1 pro příslušnou osu reverzace a zpomalení pro **KRx**=1 reference po reverzaci pro **KRx**=0 Po zreferování se nastaví příslušný bit v **HOMING.** Průběh reference ovlivňuje nastavení: RefSwitch, SlowDownSwitch, RefSpeed, RefSpeedLow, MoveToPosAfterRef, PosAfterRef, FeedOvrDuringRef, InvertRefDirection, MachineNullPoint
	11	**Obyčejný nájezd na spínače bez reverzace pohybu**. Je možné zpomalení rychlosti pro zpomalovací referenční spínač. Referenci provede systém. systém nastaví: **REFPI** <- 1 **PO_OSxPI** <- 1 pro příslušnou osu zpomalení pro **ZPRx**=1 reference pro **KRx**=1 Po zreferování se nastaví příslušný bit v **HOMING.** Průběh reference ovlivňuje nastavení: RefSwitch, SlowDownSwitch, RefSpeed, RefSpeedLow, MoveToPosAfterRef, PosAfterRef, FeedOvrDuringRef, InvertRefDirection, MachineNullPoint
	12, 15	**Nájezd na spínače s reverzací pohybu a s rovnáním** Provede se rovnání všech servosmyček připojených k ose. Je možné zpomalení rychlosti po reverzaci pohybu. Referenci provede systém. systém nastaví: **REFPI** <- 1 **PO_OSxPI** <- 1 pro příslušnou osu Pro serva x,y (x,y = 1,2,3,…16) reverzace a zpomalení pro: **REF_SERV_x**=1 AND **REF_SERV_y**=1

		Pro RefMethod=12 jednotlivá serva pokračují v jízdě až najedou všechny spínače. Pro RefMethod=15 jednotlivá serva zastaví, když najedou na svůj spínač. Po reverzaci jednotlivá serva zastaví, když **REF_SERV_x**=0 reference po reverzaci pro: **REF_SERV_x**=0 AND **REF_SERV_y**=0 Po zreferování se nastaví příslušný bit v **HOMING.** Průběh reference ovlivňuje nastavení: SlowDownSwitch, RefSpeed, RefSpeedLow, MoveToPosAfterRef, PosAfterRef, FeedOvrDuringRef, InvertRefDirection, MachineNullPoint
	13	**Simulace reference** Netestují se referenční spínače. Po zreferování se nastaví příslušný bit v **HOMING.** Průběh reference ovlivňuje nastavení: MoveToPosAfterRef, PosAfterRef, MachineNullPoint
	14	**Reference s absolutním odměřováním** Netestují se referenční spínače. Reference provede výpočet a zpětnou transformaci z POS5 do POS0 podle aktuální hodnoty **SERVO_POSITION**, kde je poloha v inkrementech pohonu. Orientace odměřování se nastaví podle hodnoty v InvertRefDirection Po zreferování se nastaví příslušný bit v **HOMING.** Průběh reference ovlivňuje nastavení: MoveToPosAfterRef, PosAfterRef, MachineNullPoint, InvertRefDirection

„PLC" reference PLC systém nastaví: **HOMING_PLC** <- 1		**Referenci provede PLC** systém nastaví: **PB_HOMING_AXIS** <- maska os (bity 0-15) PLC žádá ve vhodný okamžik o nastaveni „nulového bodu" do POS5 příslušným bitem v **HOMING_REQ.** Systém provede všechny zpětné transformace polohy. Po zreferování PLC nastaví příslušný bit v **HOMING.** PLC podle potřeby musí také povolit nelineární korekce a test na softwarové limitní spínače. PLC program po provedení vynuluje bit **HOMING_PLC**. Průběh reference ovlivňuje nastavení: MachineNullPoint
„Pseu" Pseudoreference		**Pseudoreference** Pseudoreferenci provede systém. Po zreferování se nastaví příslušný bit v **HOMING.** Vynuluje se odměřování v POS5 a provedou se všechny zpětné transformace polohy. Zruší se všechny nelineární korekce a tepelná kompenzace. Zakáží se softwarové limitní spínače.
„Simul" Simulovaná reference		**Simulaci reference** Simulaci reference provede systém Nastaví se příslušný bit v **HOMING.**
„Clear" Zrušit referenci		**Zrušení reference** Zrušení reference provede systém Vynuluje se příslušný bit v **HOMING.**
„Group" Skupinová reference systém nastaví: **HOMING_GROUP** <- 1		**Skupinová reference** Referenci provede PLC. PLC může žádat o provedení reference systém instrukcí **SPI_AX** a **HOMING_START_REQ** nebo referenci provede PLC ve své režii. Když referenci provádí systém (instrukce SPI_AX), tak platí všechna nastavení jako pro referenci typu „Single" včetně způsobu reference „RefMethod".

		V případě, že PLC provede referenci ve své režii, musí žádat ve vhodný okamžik o nastaveni „nulového bodu“ do `POS5` příslušným bitem v **`HOMING_REQ`**. Systém provede všechny zpětné transformace polohy. Po zreferování PLC nastaví příslušný bit v **`HOMING`**. PLC podle potřeby musí také povolit nelineární korekce a test na softwarové limitní spínače. PLC program po provedení vynuluje bit **`HOMING_GROUP`**. Průběh reference ovlivňuje nastavení: `MachineNullPoint`

19

19. POLOHOVACÍ JEDNOTKA

Polohovací jednotka umožňuje PLC programu pohybovat libovolnou souřadnicí zadanou rychlostí na zadanou míru. Souřadnice se rozjíždí po rampě se zadanou strmostí na rychlost nastavenou PLC programem a dojíždí na koncovou míru opět po rampě. PLC program řídí jen takovou souřadnici, která je v polohové vazbě a v době, kdy není ovládána z NC systému.

19.1 Princip polohovací jednotky

Polohovací jednotka je programový modul plně ovládaný z PLC programu. PLC program má k dispozici 6 polohovacích jednotek. Ovládání jednotky se provádí pomocí bitů v povelovém dvou-bajtovém příkazu POS_CONTROL a stavové signály z polohovací jednotky možno přečíst z bitů dvou-bajtového záznamu POS_STATUS. Kromě toho polohovací jednotka musí mít zadanou rychlost pojezdu, dráhu pojezdu a strmost rampy.

Polohovací jednotka se ovládá z PLC programu pomocí čtyřech instrukcí.

První instrukce **POS_INIT** slouží na inicializaci jednotky.

Druhá instrukce **POS_MODE** slouží na namódování jednotky. V parametrech této instrukce se zadává dráha pojezdu, rychlost pojezdu, zrychlení pro nastavení strmosti rampy. Kromě toho se zadává také odkaz na dvou-bajtový záznam POS_CONTROL, ve kterém jsou řídicí bity polohovací jednotky.

Třetí instrukce **POS_CONTROL** zabezpečuje přenos aktuální rychlosti do polohovací jednotky v průběhu pohybu nebo může pozastavit pohyb pomocí řídicího bitu MP. Kromě toho se v parametru instrukce zadává odkaz na dvou-bajtový záznam POS_STATUS, ve kterém jsou stavové bity polohovací jednotky. Z nich může například PLC program přečíst informaci o dosažení žádané polohy.

Čtvrtá instrukce **POS_RESET** ukončí pro PLC práci s polohovací jednotkou a osu odevzdá pro NC řízení.

19.2 Záznamy stavových a řídicích bitů polohovací jednotky

Význam bitů v 1. bajtu stavového záznamu POS_STATUS1_x

bit 0.
POS_STV_JEDE ..... 1 = souřadnice je v pohybu

bit 1.
POS_STV_SMER ..... Informace o směru pohybu. 0= kladný směr
1=záporný směr

bit 3.
POS_STV_POL ..... 1 = dosažení zadané polohy

Význam bitů ve 2. bajtu stavového záznamu POS_STATUS2_x

bit 0.
POS_STV_DISP ..... 1 = polohovací jednotka je k dispozici

bit 1.
POS_STV_PROG ..... 1 = polohovací jednotka je naprogramovaná

bit 2.
POS_STV_ERR ..... 1 = error polohovací jednotky

Význam bitů ve 2. bajtu řídicího záznamu POS_CONTROL2_x

Signály uvedené v tomto bajtu se nenastavují v PLC programem přímo, protože je nastavují instrukce pro ovládání jednotky.

bit 0.
POS_CNT_MP ..... 1 = povolení pohybu
0 = zakázání pohybu
(povolení pohybu se řídí parametrem instrukce POS_CONTROL)

bit 1.
POS_CNT_RESET ..... 1 = reset polohovací jednotky
(reset polohovací jednotky automaticky
nastaví instrukce POS_INIT)

bit 2.
POS_CNT_INIC ..... 1 = start iniciátoru pohybu polohovací jednotky
(tento bit nastaví instrukce POS_MODE)

bit 3.
POS_CNT_CONT ..... 1 = start kontinuátoru pohybu
(tento bit nastaví iniciátor pohybu)

19.3 Instrukce pro řízení polohovací jednotky

instrukce	**POS_INIT**

funkce	**POS_INIT**	**inicializace polohovací jednotky**
syntax	**POS_INIT**	**axis, [space]**
1.parametr	**„axis"**	**číslo osy (1,2,..,16)**
2.parametr	**„space"**	**prostor transformací (0,1,..,6)**

Instrukce **POS_INIT** slouží na inicializaci jednotky. První parametr je číslo NC osy, kterou bude PLC používat. Po inicializaci už nemůže osou jezdit standardní interpolátor.
Instrukce kromě jiného vygeneruje impuls na bitu POS_CNT_RESET a trvale na hodnotu 1 nastaví bit POS_STV_DISP. Od tohoto okamžiku je řízení souřadnice i servosmyčka k dispozici jen pro PLC program.

V případě, že polohovací jednotka byla již v činnosti, instrukce POS_INIT způsobí zastavení pohybu bez možnosti dalšího pokračování pohybu na zadanou míru. Další pohyb je umožněn jen novým naprogramováním polohovací jednotky pomocí instrukce POS_MODE.

Systém standardně používá několik transformací, které jsou zařazeny na výstup interpolátoru. Jednotlivé transformace oddělují „**prostory**", ve kterých je definovaná aktuální poloha. Druhý parametr instrukce určuje, v jakém prostoru bude PLC program s osou jezdit.

Transformace	**Prostor**
Interpolace (fiktivní poloha)	
	0
Programová transformace	
	1
Transformace polotovaru	
	2
Délková korekce	
	3
Posunutí 1	
	4
Posunutí 2	
	5
Transformace stroje (reálná poloha)	
	6

instrukce	POS_RESET

funkce	**POS_RESET**	**reset polohovací jednotky**
syntax	**POS_RESET**	**axis**
parametr	**„axis"**	**číslo osy (1,2,..,16)**

Instrukce **POS_RESET** slouží na reset jednotky. Parametrem je číslo NC osy. Instrukce kromě jiného vygeneruje impuls na bitu POS_CNT_RESET a trvale na hodnotu 0 nastaví bit POS_STV_DISP. Od tohoto okamžiku je řízení souřadnice i servosmyčka k dispozici jen pro systémové prostředky (řízení NC).

instrukce	POS_MODE

funkce	**POS_MODE**	**naprogramování polohovací jednotky**
syntax	**POS_MODE**	**control, len, accel, axis**
1.parametr	**„control"**	**řídící záznam**
2.parametr	**„len"**	**dráha pojezdu**
3.parametr	**„accel"**	**strmost rozjezdové rampy**
4.parametr	**„axis"**	**číslo osy (1,2,..,16)**

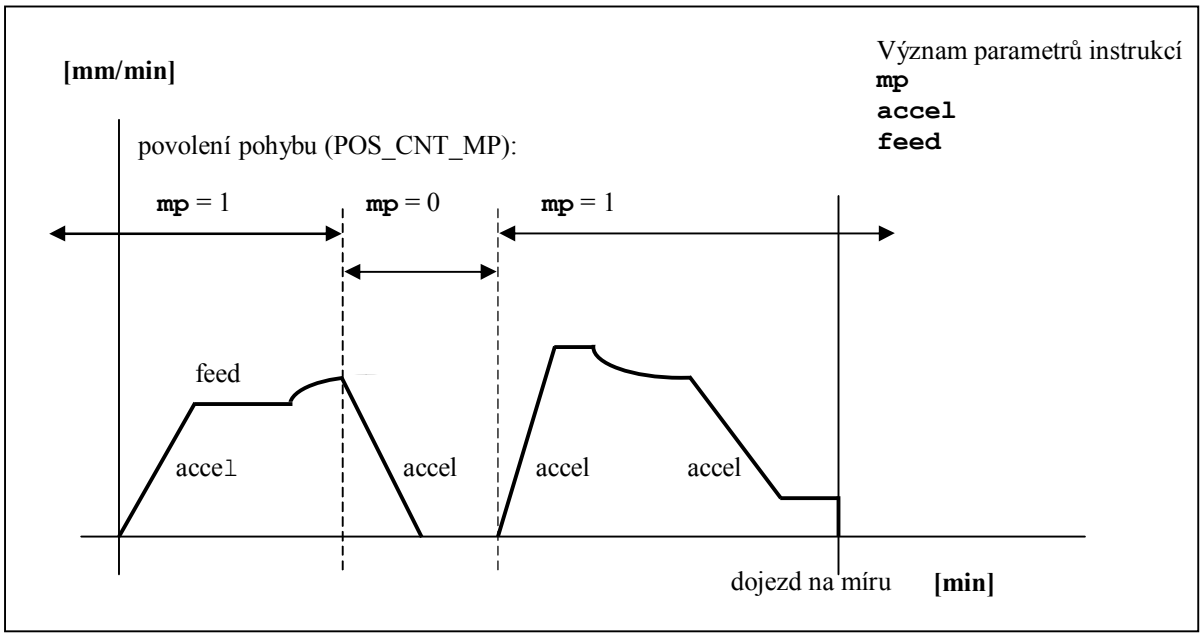
Instrukce **POS_MODE** slouží pro naprogramování polohovací jednotky. V parametrech této instrukce se zadává dráha pojezdu a zrychlení pro nastavení strmosti rampy. Kromě toho se zadává také odkaz na dvou-bajtový záznam POS_CONTROL ve kterém jsou řídicí bity polohovací jednotky. Instrukce nastaví požadovanou rychlost na nulovou hodnotu.

Parametr **„control"** je odkaz na dvou-bajtový řídicí záznam POS_CONTROL, který byl popsán v předešlé části.
Parametr **„len"** je odkaz na double-wordovou buňku pro zadání dráhy pojezdu. Dráha pojezdu se zadává v mikrometrech v dolpňkovém kódu.
Parametr **„accel"** je odkaz na wordovou buňku a slouží pro nastavení strmosti rozjezdové a dojezdové rampy. Zrychlení se nastavuje v [mm/sec**2].
Parametr **„axis"** je číslo NC osy (1,2,..,16).

instrukce	**POS_CONTROL**

funkce	**POS_CONTROL**	**řízení pohybu polohovací jednotky**
syntax	**POS_CONTROL**	**status, feed, [mp], axis**
1.parametr	**„status"**	**stavový záznam**
2.parametr	**„feed"**	**požadovaná rychlost**
3.parametr	**„mp"**	**povolení pohybu**
4.parametr	**„axis"**	**číslo osy (1,2,..,16)**

Parametr **„status"** je odkaz na dvou-bajtový stavový záznam POS_STATUS, ve kterém jsou stavové bity polohovací jednotky. Pro potřebu trvalého sledování stavu polohovací jednotky a trvalého zadávání aktuální požadované rychlosti je vhodné, aby instrukce POS_CONTROL byla volána průběžně v době pohybu polohovací jednotky.
Parametr **„feed"** je odkaz na double-wordovou buňku pro zadání rychlosti pojezdu. Rychlost pojezdu se zadává v hodnotách 1/64000 mm/min, to znamená, že když požadujeme rychlost zadanou v mm/min, zadá se hodnota jen do horního wordu rychlosti.
Parametr **„mp"** je odkaz na bit povolení pohybu. Parametr je nepovinný, v tomto případě se neuvede nebo se zadá klíčové slovo NIL. Instrukce způsobí přenastavení bitu POS_CNT_MP v řídicím záznamu POS_CONTROL2_x.
Parametr **„axis"** je číslo NC osy (1,2,..,16).



Příklad:
Naprogramování pohybu ve 4. souřadnici v PLC programu pomocí mechanizmu. Povolení pohybu je odvozeno od vstupního signálu POV_I:

```
; v deklaraci dat:
EQUI        K0,0
CONTR1:     DFM   ,,,,,,,
CONTR2:     DFM   POS_MP,POS_RESET,,,,,,
STAV1:      DFM   ,,,POS_POL,,,,
STAV2:      DFM   POS_DISP,POS_PROG,,,,,,
POSUN:      DS    4
ZRYCH:      DS    2
RYCHLOST:   DS    4


MECH_BEGIN  POSUN

;Inicializace jednotky a zadani dat
            POS_INIT    4
            LOD         zelana_draha            ;nastaveni zelane
            STO         POSUN                   ;drahy, zrychleni
            LOD         zelana_strmost
            STO         ZRYCH

;Naprogramovani polohovaci jednotky
            POS_MODE    CONTR1,POSUN,ZRYCH,4
            EX

;Rizeni pohybu a cekani na dojezd
            LOD         zelana_rychlost         ;zadavani rychlosti
            STO         RYCHLOST
            POS_CONTROL STAV1,RYCHLOST,POV_I,4
            LDR         POS_POL                 ;cekani na konec pohybu
            EX0
            POS_RESET   4                       ;reset pol. jednotky

MECH_END    POSUN
```

Příklad:
Naprogramování pohybu ve 1. souřadnici v PLC programu pomocí mechanizmu. (Řízení pohybu pomocí směrových tlačítek stroje.)

```
;RIDICI BITY
CONTRX1:    DFM ,,,,,,,
CONTRX2:    DFM PJ_X_MP,PJ_X_RESET,,,,,,

;STAV POL.JEDNOTKY
STAVX1:     DFM PJ_X_JEDE,,,PJ_X_POL,,,,
STAVX2:     DFM PJ_X_DISP,PJ_X_PROG,,,,,,

RYCHL_MAX:  DS 4
PROCENTO_F: DS 2

POSUN_X:    DS 4
ZRYCH_X:    DS 2
RYCHL_X:    DS 4


;START POHYBU POLOHOVACI JEDNOTKOU !!
```

```
      LOD   CNST.60000000
      STO   DWRD.zadana_poloha
      FL    1,MECH_START_POHYBU_X
      FL    1,MECH_HLIDANI_X        ;MECH. HLIDACI

;.....................................

;MECHANIZMUS PRO NAPROGRAMOVANI POLOH. JEDNOTKY
;mechanizmus je aktivni pokud se jede

MECH_BEGIN MECH_START_POHYBU_X
      POS_INIT    1
      LOD   DWRD.zadana_poloha      ;v mikronech doplnkovy binarni kod
      STO   POSUN_X
      POS_MODE    CONTRX1,POSUN_X,ZRYCH_X,1
      FL    1,MECH_POHYB_X          ;MECHANIZMUS RIZENI POHYBU
      FL    1,PJ_X_MP               ;POVOLENI POHYBU
      EX
      LDR   MECH_POHYB_X            ;CEKANI NA DOJEZD
      EX1
MECH_END MECH_START_POHYBU_X

;.....................................

;MECHANIZMUS RIZENI POHYBU
;mechanizmus zadava aktualni rychlost a ceka na dosazeni polohy

MECH_BEGIN MECH_POHYB_X
       EX
       LOD  DWRD.aktualni_rychlost       ;RYCHLOST V 1/64000 mm/min
       STO  RYCHL_X
       POS_CONTROL      STAVX1,RYCHL_X,PJ_X_MP,1
       LDR  PJ_X_POL
       EX0
       MECH_INIT  MECH_HLIDANI_X
       POS_RESET  1
MECH_END MECH_POHYB_X

;.................................

;HLIDACI MECHANIZMUS PRO UKONCENI POHYBU
;mechanizmus testuje podminky jeti a v pripade splneni ukonci pohyb

MECH_BEGIN MECH_HLIDANI_X
      EX
;;;;... PODMINKY BLOKOVANI:    (NAPRIKLAD OD UVOLNENI TLACITKA PRO JETI)
      LDR   LIMIT_C
      EX1
      MECH_INIT  MECH_POHYB_X
      LOD   CNST.0
      STO   RYCHL_X
      FL    0,PJ_X_MP
      POS_CONTROL  STAVX1,RYCHL_X,PJ_X_MP,1
      LDR   PJ_X_JEDE
      EX1
      POS_RESET   1
MECH_END MECH_HLIDANI_X
```

27

27. LOGICKÉ VSTUPY A VÝSTUPY MODULŮ SYSTÉMU

27.1 Princip

V kapitole jsou popsány prostředky pro logické propojování obecných vstupů a výstupů různých modulů systému (včetně PLC). Nové možnosti logického mapování pomocí instrukcí `DEF_IN` a `DEF_OUT` se neomezují jen na CAN-BUS periferie MEFI, jako je to v případě mapování pomocí instrukci `MAP_IN` a `MAP_OUT`, ale umožní například propojení PLC - EtherCAT apod.

Definice logických vstupů a výstupů umožňuje jejich propojování jen na základě konfigurace a bez nutnosti změny PLC programu. PLC program může mít nadefinován velké množství vstupů a výstupů ale jen některé z nich se propojí podle konfigurace na danou specifikaci stroje.

Jeden logický spoj je definován dvěma textovými identifikátory pro vstup a pro výstup.

Pokud logické vstupy a výstupy nejsou propojeny, může se uplatnit přednastavení buněk podle defaultní hodnoty.

Pro každý logický vstup a výstup možno určit požadovanou dobu obsluhy „virtuálního spoje“ (`Service Period`). Tem může být podle cyklu PLC (`SLOW`), podle cyklu interpolátoru (`FAST`), podle cyklu EtherCAT Mastra (`FASTEST`), nebo „na vyžádání“ (`ONDEMAND`).

Při definici logických prvků pro „virtuální spoje“ v PLC programu se proměnné automaticky deklarují – vymezí se jim požadovaný paměťový prostor.

Datové prvky, které definují logický spoj, mohou být definovány jako pole s ohledem na typ proměnné. V tomto případě logický spoj k danému prvku určuje jeho jméno a koncový index.

Pro každý logický spoj možno definovat konverzi, která se uplatní při propojení prvků.

Při propojování prvků s různými datovými typy dochází automaticky k přetypování proměnné podle typu cílové proměnné. Například se provede znaménkové rozšíření, nebo převod na reálné číslo.

Všechny logické vstupy a výstupy mají automaticky nastaveno „**přímé sdílení**“ a tak se značně zjednoduší přístup uživatelských obrazovek a dialogů k jejím hodnotám. Na logické vstupy a výstupy není potřeba používat instrukce `SHARE_VAR` a `SHARE_BIT`.

27.2 Popis modulů pro logické spoje

V „modulech“ jsou definovány logické vstupy a výstupy pro tvorbu virtuálních spojů. Název modulu se uvádí v identifikátoru prvku na prvním místě.

Modul	popis
CAN[1]	Druhý CAN-BUS kanál pro obsluhu periferií vstupů, výstupů, panelů a pod. (První CAN-BUS kanál je určen pro obsluhu pohonů). Jedná se o možnost připojení k periferiím MEFI: INOUT08. Z hlediska propojování jsou fyzické binární vstupy definovány jako logické výstupy a fyzické binární výstupy jsou logické vstupy. Typ prvků je BIT nebo BYTE a možnost obsluhy je FAST (viz dále).
VKeyboard	Virtuální klávesnice. Konfiguruje se v pořadí jako 7. klávesnice (pořadí 6 od 0). Nejedná se o fyzickou klávesnici nebo panel. Virtuální klávesnice má definováno 256 virtuálních tlačítek (s 256 SCAN kódy), ke kterým se možno připojit a ovládat je jako kdyby to byly standardní tlačítka. Virtuální tlačítka svými stisky tak mohou vykonávat příkazy podle konfigurace, která je zadána v souboru typu „KbdConfig“. Tanto soubor musí být zaregistrován pro 7.klávesnici. Virtuální klávesnice v podstatě umožňuje PLC programu pomocí bitu spouštět příkazy definovány pomocí atributů: Command, RtmCommand, PlcCommand, nebo přímo zavolat dialogové okno definováno v elementu <Dialog>…</Dialog> (viz návod „Ovládací panely“). Typ prvků je BIT a možnost obsluhy je SLOW (viz dále).
PLC.Input PLC.Output	Logické vstupy a výstupy definované v PLC programu pomocí instrukcí DEF_IN a DEF_OUT (viz dále). Perioda obsluhy a typ jsou definovány v instrukcích v PLC programu.
ECAT	Připojení k logickým vstupům a výstupům definovaným v „Process image“ pro EtherCAT komunikaci. Pro jeho tvorbu je nutno použít „EtherCAT Studio“. (viz návod „Periferie připojené na EtherCAT“). Typ je definován v „Process image“. Možnost periody obsluhy: FASTEST (viz dále).
RTM.Axis[x]	Logické vstupy a výstupy definované v systémovém programu pro NC osy (RTM je systémový program reálného času). Index NC osy „x“ může nabývat hodnot 0 – 15 podle pořadového čísla osy. Výčet hodnot je uveden v následující tabulce. Možnost periody obsluhy: FAST (viz dále).
RTM.Servo[x]	Logické vstupy a výstupy definované v systémovém programu pro servosmyčky. Index servosmyčky „x“ může nabývat hodnot 0 – 15 podle pořadového čísla serva. Výčet hodnot je uveden v následující tabulce. Možnost periody obsluhy: FAST.
RTM.Spindle	Logické vstupy a výstupy definované v systémovém programu pro vřetena. Výčet hodnot je uveden v následující tabulce. Možnost periody obsluhy: FAST.
RTM.Man	Logické vstupy a výstupy definované v systémovém programu pro manuální režim. Výčet hodnot je uveden v následující tabulce. Možnost periody obsluhy: FAST.
RTM.Control	Logické vstupy a výstupy definované v systémovém programu pro řízení běhu PLC apod.). Výčet hodnot je uveden v následující tabulce. Možnost periody obsluhy: FAST.

27.3 Zásady tvorby identifikátorů logických prvků

Logické virtuální spoje jsou definovány dvěma textovými identifikátory. Identifikátor logického výstupu a identifikátor logického vstupu. Textové identifikátory používají „tečkovou konvenci“ podle dále popsaných zásad.

Datové prvky, které definují logický spoj, mohou být v PLC programu definovány jako pole s ohledem na typ proměnné. V tomto případě logický spoj k danému prvku určuje jeho jméno a také koncový index. Proto koncový index není součástí identifikátoru. Identifikátor ale může obsahovat i další indexy (které nejsou koncové) a ty pak musí být jeho součástí.

Identifikátory pro vstupy a výstupy EtherCATu je nutno podle potřeby upravit v EtherCAT studiu.

Možnosti tvorby textových identifikátorů

Skupiny pro logické spoje	**Popis**
Periferie CAN-BUS INOUT08 – logické výstupy – bitový přístup	

`CAN[1].INOUT08[xx].IP0.Bit[xx]`
`CAN[1].INOUT08[xx].IP1.Bit[xx]`
`CAN[1].INOUT08[xx].IP2.Bit[xx]`
`CAN[1].INOUT08[xx].IP3.Bit[xx]`
	Připojení k fyzickým vstupům INOUT08.
	Typ: `BIT`.
	Možnost periody obsluhy: `FAST`

Periferie CAN-BUS INOUT08 – logické vstupy – bitový přístup	

`CAN[1].INOUT08[xx].OP0.Bit[xx]`
`CAN[1].INOUT08[xx].OP1.Bit[xx]`
`CAN[1].INOUT08[xx].OP2.Bit[xx]`
	Připojení k fyzickým výstupům INOUT08.
	Typ: `BIT`.
	Možnost periody obsluhy: `FAST`

Periferie CAN-BUS INOUT08 – logické výstupy – bajtový přístup	

`CAN[1].INOUT08[xx].IP0`
`CAN[1].INOUT08[xx].IP1`
`CAN[1].INOUT08[xx].IP2`
`CAN[1].INOUT08[xx].IP3`
	Připojení k fyzickým vstupům INOUT08.
	Typ: `BYTE`.
	Možnost periody obsluhy: `FAST`

Periferie CAN-BUS INOUT08 – logické vstupy – bajtový přístup	

`CAN[1].INOUT08[xx].OP0`
`CAN[1].INOUT08[xx].OP1`
`CAN[1].INOUT08[xx].OP2`
	Připojení k fyzickým výstupům INOUT08.
	Typ: `BYTE`.
	Možnost periody obsluhy: `FAST`

Virtuální klávesnice – logické vstupy	
`VKeyboard.VKey[xx]`	Připojení na virtuální klávesnici. Typ: `BIT` Možnost periody obsluhy: `SLOW`
PLC proměnné – logické vstupy	
`PLC.Input.<jmeno>[xx]`	Připojení k logickým vstupům definovaných v PLC programu (`<jmeno>` je definováno instrukcí `DEF_IN`) Perioda obsluhy a typ jsou definovány v instrukci `DEF_IN`
PLC proměnné – logické výstupy	
`PLC.Output.<jmeno>[xx]`	Připojení k logickým výstupům definovaných v PLC programu (`<jmeno>` je definováno instrukcí `DEF_OUT`) Perioda obsluhy a typ jsou definovány v instrukci `DEF_OUT`
EtherCAT – logické vstupy a výstupy	
`ECAT.<jmeno>[xx]`	Připojení k logickým vstupům a výstupům definovaným v „Process image“ pro EtherCAT komunikaci. Typ je definován v „Process image“. Možnost periody obsluhy: `FASTEST`

RTM proměnné – NC osy	
`RTM.Axis[xx].<jmeno>`	Připojení k logickým vstupům a výstupům definovaných v RTM programu pro NC osy (Výčet hodnot `<jmeno>` je uveden v následující tabulce). Možnost periody obsluhy: `FAST`
RTM proměnné – servosmyčky	

`RTM.Servo[xx].<jmeno>`
`RTM.Servo[xx].Edrv.<jmeno>`
`RTM.Servo[xx].Enc.<jmeno>`
	Připojení k logickým vstupům a výstupům definovaných v RTM programu pro servosmyčky (Výčet hodnot `<jmeno>` je uveden v následující tabulce).
	Možnost periody obsluhy: `FAST`

RTM proměnné – vřetena	
`RTM.Spindle.<jmeno>`	Připojení k logickým vstupům a výstupům definovaných v RTM programu pro vřetena (Výčet hodnot `<jmeno>` je uveden v následující tabulce). Možnost periody obsluhy: `FAST`
RTM proměnné – manuální režim	
`RTM.Man.<jmeno>[xx]`	Připojení k logickým vstupům a výstupům definovaných v RTM programu pro manuální režim (Výčet hodnot `<jmeno>` je uveden v následující tabulce) Možnost periody obsluhy: `FAST`
RTM proměnné – řízení	
`RTM.Control.<jmeno>[xx]`	Připojení k logickým vstupům a výstupům definovaných v RTM programu pro řízení běhu PLC programu apod. (Výčet hodnot `<jmeno>` je uveden v následující tabulce) Možnost periody obsluhy: `FAST`

Pravidla pro definici identifikátoru prvku v PLC, v EtherCAT apod. (pomocí instrukce `DEF_IN`, `DEF_OUT`)

- Při definici v PLC programu se uvádí jen jméno prvku `<jmeno>` bez názvu skupiny.
- Na velikosti písmen nezáleží.
- Použití nulových indexů (`[0]`, `[00]`, `[000]`…) je ekvivalentní zápisu bez indexů. Možno je libovolně používat. Nulové indexy nejsou součástí identifikátoru.
- Vedoucí nuly v indexech se automaticky odstraňují (`[05] = [5]`), nejsou součástí identifikátoru. Možno je používat.
- Nesmí se v definici uvést žádný koncový index `[xx]`, ten není součástí identifikátoru a může se uvést až v pravidlech pro vyhledávání log.spoje.
- Jméno prvku `<jmeno>` může obsahovat tečky a indexy `[xx]`, které jsou součástí identifikátoru.

Pravidla pro vyhledávaní pro předpis log.spoje v konfiguraci ... element `<Connection>` a v HTML stránkách

- Na začátku může být nepovinně klíč „`CNC.`“
- Zápis s indexem (`[0]`, `[00]`, `[000]`…) na libovolné pozici je ekvivalentní zápisu bez indexu. Možno jej používat
- Vedoucí nuly v indexech se automaticky odstraňují a možno je používat (`[05] = [5]`).
- Na velikosti písmen nezáleží.
- Analogové hodnoty mohou mít koncový index `[xx]`, který se uplatní podle typu proměnné.
- Jméno prvku `<jmeno>` může obsahovat tečky a indexy `[xx]`, které jsou součástí identifikátoru.
- Jméno prvku `<jmeno>` může mít přetypování na anonymní přístup k bitu „`.Bit[xx]`“

Příklady pro definici instrukcích DEF_IN, DEF_OUT v PLC programu:

```
DEF_IN flBitIn1,               'BitIn1',                          TYPE_BIT
DEF_IN flBitOut2,              'BitOut2',                         TYPE_BIT
DEF_IN rBunOut3[10],           'BunOut3',                         TYPE_REAL
DEF_IN dwBunIn1,               'BunIn1[2].Prvek',                 TYPE_DWORD
DEF_IN inLaserEgBodyTouch,     'PrecitecEG8030[0].BodyTouch',     TYPE_BIT
DEF_IN rLaserEgDistanceLinear,'PrecitecEG8030.DistanceLinear', TYPE_REAL
```

Příklady pro předpis v konfiguraci v elementu `<Connection>` *:*

```
Source="PLC.Output.BitOut2"
Destination="CAN[1].INOUT08[2].OP1.Bit[2]"

Source="PLC.Output.BitOut2"
Destination="VKeyboard.Vkey[22]"

Source="PLC.Output.BunOut3[4]"
Destination="PLC.Input.dwBunIn1

Source="ECAT.ServoX.Inputs.Status word"
Destination="PLC.Input.Servo[8].StatusWord"

Source="ECAT.ServoX.Inputs.Position actual value"
Destination="PLC.Input.Servo[8].PositionAct"

Source="ECAT.CANOpen.EG8030_Inputs.TxPDO1_wDeviceStateEG.Bit[4]"
Destination="PLC.Input.PrecitecEG8030.TipTouch"

Source=" RTM.Axis[2].Pos0_AX"
Destination="PLC.Input.Position_2"
```

27.4 Tabulka RTM prvků pro logické spoje

Tabulka systémových „RTM“ prvků pro možnost vytváření logických spojů.

RTM.Axis[xx].		
ReqShift	BIT	Požadavek na režim SHIFT
ActShift	BIT	Režim SHIFT aktivní
ShiftEnable_Ax	pole BIT (x = 0,1,..,15)	Povolení SHIFT z PLC
Pos5Inc	REAL[mm/T]	Prostorový aktuální přírůstek dráhy v GEO osách v POS5
Pos6Inc	REAL[mm/T]	Prostorový aktuální přírůstek dráhy v GEO osách v POS6
Pos5IncPm	REAL[mm/min]	Prostorový aktuální přírůstek dráhy v GEO osách v POS5
Pos6IncPm	REAL[mm/min]	Prostorový aktuální přírůstek dráhy v GEO osách v POS6
IncPt_Ax	pole REAL[mm/T] (x = 0,1,..,15)	Aktuální přírůstek dráhy v ose
Inc_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy v ose
Angle5Ax1	REAL[rad]	1.fyzická osa rotace pro naklápění
Angle5Ax2	REAL[rad]	2.fyzická osa rotace pro naklápění
Pos0_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS0
Pos1_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS1
Pos2_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS2
Pos3_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS3
Pos4_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS4
Pos5_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS5
Pos6_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS6
ToolVectorX	REAL	Jednotkový prostorový vektor orientace nástroje - složka X
ToolVectorY	REAL	Jednotkový prostorový vektor orientace nástroje - složka Y
ToolVectorZ	REAL	Jednotkový prostorový vektor orientace nástroje - složka Z
KnobStep	UNS16	Krok točítka
Ax1BevelAngle	REAL[rad]	Fyzický úhel 1.rotace při úkosové interpolaci (mezivýsledek)

Ax2BevelAngle	REAL[rad]	Fyzický úhel 2.rotace při úkosové interpolaci (mezivýsledek)
CircleAngle	REAL[rad]	Aktuální úhel kruhové interpolace
CircleDist	REAL[mm]	Distance pro kruhovou interpolaci
Distanc_Ax	pole REAL[mm] (x = 0,1,..,15)	Složkové distance pro jednotlivé osy
Distanc	REAL[mm]	Prostorová distance os
ToolTangentialAngleXY	REAL[rad]	Průmět aktuálního tečnového úhlu úkosu TollTangentialAngle
TangentialAngleAct	REAL[rad]	Aktuální tečnový úhel bloku
ToolBevelAngleAct	REAL[rad]	Aktuální úhel úkosu pro úkosovou interpolaci
ToolTangentialAngleAct	REAL[rad]	Aktuální tečnový úhel úkosu v úkosové rovině
Pos6Shift_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Posun SHIFT
TouchPos0_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS0 naměřena dotykovou sondou
TouchPos1_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS1 naměřena dotykovou sondou
TouchPos2_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS2 naměřena dotykovou sondou
TouchPos3_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS3 naměřena dotykovou sondou
TouchPos4_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS4 naměřena dotykovou sondou
TouchPos5_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS5 naměřena dotykovou sondou
TouchPos6_Ax	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS6 naměřena dotykovou sondou
TouchProbe0Stop	BIT	Stop pohybu od dotykové sondy
TouchProbe0	BIT	Kontakt dotykové sondy
IncManPos1_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS1
IncManPos2_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS2
IncManPos3_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS3
IncManPos4_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS4
IncManPos5_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS5

RTM.Servo[xx].			
ICounter		pole REAL[mm] (xx = 0,1,..,15)	Diferenční čítač polohové servosmyčky
ICounter_S		pole INT32[µm] (xx = 0,1,..,15)	Diferenční čítač polohové servosmyčky
ServoPosition		pole REAL[mm] (xx = 0,1,..,15)	Poloha servosmyček podle odměřování
ServoPosition_S		pole INT32[µm] (xx = 0,1,..,15)	Poloha servosmyček podle odměřování
ServoPosition_Inc		pole INT32[inc] (xx = 0,1,..,15)	Poloha servosmyček v inkrementech pohonu
ServoReqPosition		pole REAL[mm] (xx = 0,1,..,15)	Požadovaná poloha servosmyček
ServoReqPosition_S		pole INT32[µm] (xx = 0,1,..,15)	Požadovaná poloha servosmyček
IncCan_Ax		pole INT32[inc] (xx = 0,1,..,15)	Přírůstky dráhy pro CAN-BUS a EtherCAT trajectory
Edrv.			EtherCAT pohon „EDRV“
	State	pole UNS8 (xx = 0,1,..,15)	Stav EtherCAT pohonu EDRV (EDRV_INIT, EDRV_DONE,..)
	MoveDisReq	pole BIT (xx = 0,1,..,15)	Žádost o blokování pohybu od pohonů EDRV
	Error	pole BIT (xx = 0,1,..,15)	Chyba EtherCAT pohonu EDRV
	Simul	pole BIT (xx = 0,1,..,15)	EDRV pohon v simulaci
	Init	pole BIT (xx = 0,1,..,15)	Žádost o INIT EtherCAT pohonu typu EDRV
	Done	pole BIT (xx = 0,1,..,15)	Žádost o DONE EtherCAT pohonu typu EDRV
	Enable	pole BIT (xx = 0,1,..,15)	Žádost o ENABLE EtherCAT pohonu typu EDRV
	Disable	pole BIT (xx = 0,1,..,15)	Žádost o DISABLE EtherCAT pohonu typu EDRV
	On	pole BIT (xx = 0,1,..,15)	Žádost o ON (zapnutí) EtherCAT pohonu typu EDRV
	Erase	pole BIT (xx = 0,1,..,15)	Žádost o ERASE EtherCAT pohonu EDRV
	DriveStatus	pole UNS16 (xx = 0,1,..,15)	[in] „Drive status word“ EDRV pohonu (CoE, SoE)
	DrivePos	pole INT32[inc] (xx = 0,1,..,15)	[in] „Position feedback“ EDRV pohonu
	DriveErrReg	pole UNS16 (xx = 0,1,..,15)	[in] „Drive error registr“ EDRV pohonu
	DriveControl	pole UNS16 (xx = 0,1,..,15)	[out] „Master control word“ EDRV pohonu
	DriveVeloReq	pole INT32[inc] (xx = 0,1,..,15)	[out] „Velocity demand value“ EDRV pohonu
	DrivePosReq	pole INT32[inc] (xx = 0,1,..,15)	[out] „Position demand value“ EDRV pohonu
	ScaleSpeed	pole REAL (x = 0,1,..,15)	[in] Měřítko pro výstup rychlosti EDRV pohonu

Enc.			Externí odměřování EtherCAT
	CounterValue	pole INT32[inc] (xx = 0,1,..,15)	[in] „Encoder counter value“ (EL5101 EDRV)
	LatchValue	pole INT32[inc] (xx = 0,1,..,15)	[in] „Encoder latch value“ (EL5101 EDRV)
	LatchEn	pole BIT (xx = 0,1,..,15)	[out] „Encoder Enable latch“ (EL5101 EDRV)
	LatchValid	pole BIT (xx = 0,1,..,15)	[in] „Encoder Latch C valid“ (EL5101 EDRV)

RTM.Spindle.		
Spindle1Output	REAL[10V]	Výstupní napětí pro 1.vřeteno (bez procenta S)
Spindle2Output	REAL[10V]	Výstupní napětí pro 2.vřeteno (bez procenta S)
Spindle1PrsOutput	REAL[10V]	Výstupní napětí pro 2. vřeteno (s ohledem na procento S)
ReqSpeedPm	REAL[ot/min]	Požadované otáčky vřetene. (s ohledem na procento S)
ReqSpeedPt	REAL[ot/T]	Požadované otáčky vřetene. (s ohledem na procento S)
ActSpeedPm	REAL[ot/min]	Aktuální okamžité otáčky vřetene
ActSpeed2Pm	REAL[ot/min]	Aktuální okamžité otáčky 2.vřetene
ActAvgSpeedPm	REAL[ot/min]	Aktuální průměrované otáčky vřetene
ActAvgSpeed2Pm	REAL[ot/min]	Aktuální průměrované 2.otáčky 2.vřetene [ot/min]
ActSpeedPt	REAL[ot/T]	Aktuální okamžité otáčky vřetene
ActSpeed2Pt	REAL[ot/T]	Aktuální okamžité otáčky 2.vřetene
ActAvgSpeedPt	REAL[ot/T]	Aktuální průměrované otáčky vřetene
ActAvgSpeed2Pt	REAL[ot/T]	Aktuální průměrované otáčky 2.vřetene
SpindleGearAct1	UNS16	Číslo aktuálního převodového stupně pro 1.vřeteno
SpindleGearAct2	UNS16	Číslo aktuálního převodového stupně pro 2.vřeteno

RTM.Man.		
AckMan	BIT	Pomocné ruční pojezdy jsou aktivní (ACK_AUTMAN)
InposStop	BIT	Systém je v poloze posledního stopu (INPOS_STOP)
ManEnable	BIT	Povolení ručních pojezdů (EN_AUTMAN)
ManControlNC	BIT	Řízení MAN je z panelu NC systému (AUTMAN_CONT_NC)
ManControlKnob	BIT	Řízení MAN je z panýlku točítka (AUTMAN_CONT_TOC)
ManControlSelect	BIT	Předvolba pohybu MAN (AUTMAN_CONT_SELECT)
ManMovePlus_Ax[xx]	pole BIT (x = 0,1,..,5)	Pohyb v režimu MAN v kladném směru (MM_x0)
ManMoveMinus_Ax[xx]	pole BIT (x = 0,1,..,5)	Pohyb v režimu MAN v záporném směru (MM_x1)
ManControlRT	BIT	Řízení MAN rychloposuvem (AUTMAN_CONT_G00)
Shift_Ax[xx]	pole BIT (x = 0,1,..,5)	Pohyb v režimu SHIFT (SHIFT_CONTROL)
ExtManMovePlus_Ax[xx]	pole BIT (x = 0,1,..,5)	Externí pohyb v režimu MAN v kladném směru (EXM_x0)
ExtManMoveMinus_Ax[xx]	pole BIT (x = 0,1,..,5)	Externí pohyb v režimu MAN v záporném směru (EXM_x0)
ReqExtManRT	BIT	Externí požadavek pro G00 MAN (REQ_EXT_G00_AUTMAN)
ReqExtManSelect	BIT	Externí požadavek pro předvolbu REQ_EXT_SELECT_AUTMAN
ReqExtMan	BIT	Externí požadavek na pohyb (REQ_EXT_CONT_AUTMAN)
ReqExtFeedMan	BIT	Externí požadavek na rychlost (REQ_EXT_FEED_AUTMAN)
ExtFeedMan	REAL[mm/min]	Hodnota externí rychlosti v MAN (EXT_FEED_AUTMAN)
ExtFeedRTMan	REAL[mm/min]	Hodnota externí rychlosti v MAN (EXT_FEED_G00_AUTMAN)
ReqG95Man	BIT	Požadavek na otáčkový posuv pro MAN (G95_AUTMAN)
Knob1Disable	BIT	Zákaz ovládání z 1. panýlku (AUTMAN_KNOB1_DIS)
Knob2Disable	BIT	Zákaz ovládání z 2. panýlku (AUTMAN_KNOB2_DIS)

RTM.Control.		
MpPlc_Ax[xx]	pole BIT (xx = 0,1,..15)	Povolení pohybu z PLC pro osy
InRef_Ax[xx]	pole BIT (xx = 0,1,..15)	NC osa v referenci
MpMan_Ax[xx]	pole BIT (xx = 0,1,..15)	Povolení pohybu pro osy pro manuální režim (MPxMan)
Mp_Ax[xx]	pole BIT (xx = 0,1,..15)	Celkové povolení pohybu pro osy (MPx)
Move_Ax[xx]	pole BIT (xx = 0,1,..15)	Pohyb v osách (PO_OSxPI)
Direct_Ax[xx]	pole BIT (xx = 0,1,..15)	Směr pohybu pro osy (SM_POxPI)
Coupling_S[xx]	pole BIT (xx = 0,1,..15)	Vazba polohové servosmyčky (VAZBA_x)
Homing_Ax[xx]	pole BIT (xx = 0,1,..15)	Polohování osy (POLOHV_x)
StartDisable	BIT	Zákaz startu (START_DISABLE)
BlockInProgress	BIT	Blok rozpracován (PO_F)
PosReached	BIT	Programovaná poloha dosažena (PO_POL)
StopAck	BIT	Potvrzení stopu (PO_STOP)
InPos	BIT	Poloha v toleranci (INPOS)
ExtFeedOvr	BIT	Externí řízení FeedOverride (FEED_OVR)
ExtSpeedOvr	UNS16	Externí řízení procenta otáček z PLC (SPEED_OVR)
Delay	BIT	Prodleva G04
LimPlus_Ax[xx]	pole BIT (xx = 0,1,..15)	Limitní spínače v kladném směru (KHx0)
LimMinus_Ax[xx]	pole BIT (xx = 0,1,..15)	Limitní spínače v záporném směru (KHx1)
DragLimPlus_Ax[xx]	pole BIT (xx = 0,1,..5)	Zpomalovací limitní spínače v kladném směru (ZPx0)
DragLimMinus_Ax[xx]	pole BIT (xx = 0,1,..5)	Zpomalovací limitní spínače v záporném směru (ZPx1)
ModeAut	BIT	Režim AUT (AUTPI)
ModeRefer	BIT	Ržim REFER (REFPI)
ModeCanul	BIT	Režim CANUL (CAPI)
Thread	BIT	Závitování G33 (G33PI)
RapidTraverse	BIT	Rychloposuv G00 (G00PI)
SwLimPlus_Ax[xx]	pole BIT (xx = 0,1,..5)	SW limitní spínače v kladném směru (SLS_LIMIT_PLUS)
SwLimMinus_Ax[xx]	pole BIT	SW limitní spínače v záporném

	(xx = 0,1,..5)	směru (SLS_LIMIT_MINUS)
SwDragPlus_Ax[xx]	pole BIT (xx = 0,1,..5)	SW zpomal. spínače v kladném směru (SLS_DRAG_PLUS)
SwDragMinus_Ax[xx]	pole BIT (xx = 0,1,..5)	SW zpomal. spínače v záporném směru (SLS_DRAG_MINUS)
SwRefEnable_Ax[xx]	pole BIT (xx = 0,1,..5)	Povolení softwarových spínačů od reference (SLS_REFER)
HomingPoint_Ax[xx]	pole BIT (xx = 0,1,..5)	Referenční spínače (KRx)
DragHoming_Ax[xx]	pole BIT (xx = 0,1,..5)	Zpomalovací referenční spínače (ZPRx)
HomingRevers_Ax[xx]	pole BIT (xx = 0,1,..5)	Referenční spínače reverzační (KRRx)
M00	BIT	Dekódovaná funkce M00 1.skupina M (M00_PID)
M01	BIT	Dekódovaná funkce M01 1.skupina M (M01_PID)
M02	BIT	Dekódovaná funkce M02 1.skupina M (M02_PID)
M30	BIT	Dekódovaná funkce M02 1.skupina M (M30_PID)
M03	BIT	Dekódovaná funkce M03 2.skupina M (M03PI)
M04	BIT	Dekódovaná funkce M04 2.skupina M (M04PI)
M19	BIT	Dekódovaná funkce M19 2.skupina M (M19PI)
M41	BIT	Dekódovaná funkce M41 3.skupina M (M41PI)
M42	BIT	Dekódovaná funkce M42 3.skupina M (M42PI)
M43	BIT	Dekódovaná funkce M43 3.skupina M (M43PI)
M44	BIT	Dekódovaná funkce M44 3.skupina M (M44PI)
M07	BIT	Dekódovaná funkce M07 5.skupina M (M07PI)
M08	BIT	Dekódovaná funkce M08 5.skupina M (M08PI)
M50	BIT	Dekódovaná funkce M50 6.skupina M (M50PI)
M51	BIT	Dekódovaná funkce M51 6.skupina M (M51PI)
M10	BIT	Dekódovaná funkce M10 7.skupina M (M10PID)
M11	BIT	Dekódovaná funkce M11 7.skupina M (M11PID)
M06	BIT	Dekódovaná funkce M06 8.skupina M (M06PID)
M60	BIT	Dekódovaná funkce M60 8.skupina M (M60PI)
SF_Run	BIT	Systém v chodu, stroj jede, nebo jsou technologické funkce
SF_BlockInProgr	BIT	Blok rozpracován
SF_BlockFinished	BIT	Blok ukončen

SF_Stop	BIT	Všechny druhy stopu bloku
SF_NotInpos	BIT	Dosud nebylo dosaženo požadované polohy
SF_DelayInProgr	BIT	Časová prodleva
SF_IndicationMode	BIT	Indikační režim
SF_SimulationMode	BIT	Simulační režim
SF_HighspeedMode	BIT	Zrychlený režim
SF_CondstopMode	BIT	Podmíněny stop aktivní (pro bloky s M01)
SF_ManualMode	BIT	Manuální režim
SF_BlockByBlock	BIT	Režim blok po bloku
SF_LS	BIT	Limitní spínač
SF_SLS	BIT	Softwarový limitní spínač
SF_Backward	BIT	Couvání
SF_StopPosChanged	BIT	Bude nastaven, když při stopu programu bylo ručně popojeto
BlockToggle	BIT	Klopní bit pro nový blok
MeasStart	BIT	Start pro dávková měření
MeasStop	BIT	Stop pro dávková měření
MeasErr	BIT	Chyba dávkového měření
MeasControl	UNS32	Řízení pro dávková měření
BlockRestart	BIT	Start po stopu bloku
LimitFeedSpotReq	BIT	Požadavek na okamžitý limit (složkový)
G33Gradual	BIT	G33 s plynulým koncem (G33 -> G95)
TouchProbe0Stop	BIT	Stop pohybu od dotykové sondy
TouchProbe0	BIT	Dotyková sonda
KnobSel_Ax[xx]	pole UNS8 (xx = 0,1,..15)	Volba osy z točítka pro indikaci
PlcStatus	UNS8	Stav PLC
FeedOvrVal	UNS16	Hodnota promile pro externí řízení rychlosti
FeedOvrMultiplier	UNS16	Násobící koeficient pro externí řízení rychlosti
SpeedOvrVal	UNS16	Hodnota promile pro řízení otáček (SPEED_OVR_EXT)
SpeedOvrMultiplier	UNS16	Násobící koeficient pro otáčky

SF_PTTYPE	UNS32	Výčet pro programovou transformaci
SF_WTTYPE	UNS32	Výčet pro transformaci polotovaru
PLCPeriod	REAL [s]	Definovaná hodnota periody pro základní běh PLC (slow .. 20ms)
PLCPeriodFast	REAL [s]	Definovaná hodnota periody pro rychlý běh PLC (1 ms)
PLCPeriodFastest	REAL [s]	Definovaná hodnota periody pro nejrychlejší běh PLC (200 µs)
FastestModuleCount	UNS32	Diagnostický čítač průběhu MODULE_FASTEST
FastestModulePresent	BIT	Modul MODULE_FASTEST je aktivní
FastestInterrupt	REAL [s]	Definovaná hodnota periody pro nejrychlejší přerušení (200 µs)
FlagG33	UNS8	Fáze průběhu závitování G33
LOCATE_ANGLE	INT32	Žádaný uhel pro tečnovou interpolaci
ACTUAL_ANGLE	INT32	Aktuální uhel pro tečnovou interpolaci
CANErrorCode	UNS8	Chybový registr pro PLC od CAN2
CANErrorNum	UNS8	Číslo jednotky v poruše na CAN2
CANTimeOutCode	UNS8	Chyba time-out pro PLC od CAN2
CANTimeOutNum	UNS8	Číslo jednotky v poruše time-out na CAN2
PLCMemBackup[xx]	pole UNS8 (xx = 0,1,..9999)	Zálohovaná oblast PLC paměti (PLC_MEM_BACKUP)
BSPProcNumber	UNS32	Číslo procesoru BSP
SECProcNumber	UNS32	Číslo procesoru SEC

27.5 Konfigurace pro logické spoje

Konfigurace pro logické spoje jsou v souboru typu „ChannelConfig“ v elementu „Connections“.

element **Connections**			**Konfigurace pro logické (virtuální) spoje**	
	element **Connection**		**Konfigurace pro jeden logický (virtuální) spoj.**	
		atribut **Source**	**Identifikátor výstupu**	
			Abcd	Textový řetězec, který slouží jako identifikátor výstupu
		atribut **Destination**	**Identifikátor vstupu**	
			Abcd	Textový řetězec, který slouží jako identifikátor vstupu
		atribut **Invert**	**Inverze (platí pro bitové spoje)**	
			0	logický vstup je při přímo propojen s výstupem (default)
			1	logický výstup je při propojení na logický vstup invertován
		atribut **Scale**	**Měřítko (platí pro „analogové“ spoje)**	
			1.0	bez měřítka (default)
			xx.xx	Měřítko se uplatní při propojení logického výstupu na analogový logický vstup
		atribut **Offsun**	**Posun (platí pro „analogové“ spoje)**	
			0.0	bez posunu (default)
			xx.xx	Posun se uplatní při propojení logického výstupu na analogový logický vstup
		atribut **Connected**	**Propojení logického vstupu a výstupu**	
			0	logický vstup není propojen s logickým výstupem (default)
			1	logický vstup je propojen s logickým výstupem

Příklad:

Příklad logických spojů: `(flOut1 -> flIn1),(flOut1 -> INOUT08.OP0.Bit[7])`
`(flOut1 -> Bun1.Bit[2])`

```
<Connections>

  <Connection Source="PLC.Output.flOut1" Destination="PLC.Input.flIn1"
  Invert="0" Connected="1"></Connection>

  <Connection Source="PLC.Output.flOut1"
  Destination="CAN[1].INOUT08.OP0.Bit[7]" Connected="1"></Connection>

  <Connection Source="PLC.Output.flOut1"
  Destination="PLC.Input.Bun1.Bit[2]" Connected="1"></Connection>

</Connections>
```

27.6 Definice logických vstupů a výstupů v PLC programu

Pro definici logických (bitových i analogových) vstupů a výstupů s možnosti propojování slouží instrukce **DEF_IN** a **DEF_OUT**. Instrukce musí být umístěny (nebo volány) v inicializačním modulu MODULE_INIT. Instrukce automaticky definují vstupní nebo výstupní buňku podle zadaného typu (může být i pole) a automaticky nastaví přímé sdílení pro proměnné (už se nemusí použít instrukce SHARE_VAR).

instrukce	**DEF_IN** **DEF_OUT**

```
funkce      DEF_IN      Definice logického vstupu s možností propojování
            DEF_OUT     Definice logického výstupu s možností propojování


syntax      DEF_IN (DEF_OUT)  val,    poin,  typ
            DEF_IN (DEF_OUT)  val,   'TEXT', typ
            DEF_IN (DEF_OUT)  val[x],'TEXT', typ
            DEF_IN (DEF_OUT)  val,   'TEXT', typ [, def ]
            DEF_IN (DEF_OUT)  val,   'TEXT'  typ [, def,  period ]

1.parametr  „val"             název bitu nebo proměnné
2.parametr  „poin,TEXT"       pointer nebo textový řetězec identifikátoru
3.parametr  „typ"             zadání typu proměnné (viz dále)
4.parametr  „def"             defaultní hodnota
5.parametr  „period"          požadovaná doba obsluhy spoje (viz dále)
```

parametr	název	význam	typ
1.	**val**	Název bitu nebo datové proměnné pro vstup nebo výstup. Proměnná se automaticky definuje a proto v PLC programu se už nepoužívá definice pomocí instrukce DFM, a DS. Název může obsahovat počet prvků pole v hranaté závorce. Pokud se požaduje pole, automaticky se vymezí prostor pro celé pole s ohledem na typ proměnné.	Data podle typu
2.	**poin**	Ukazatel na buffer, kde je umístěný text s klíčovým slovem konfiguračního parametru - náveští u řetězce definovaného instrukcí "str".	pointer
	text	Přímé zadání textu s klíčovým slovem konfiguračního parametru v apostrofech	řetězec
3.	**typ**	Zadání typu proměnné (viz dále)	klíčové slovo
4.	**def**	Nepovinný parametr. Přímé zadání defaultní hodnoty, číslo se naplní do vstupu nebo výstupu.	číselná hodnota
5.	**period**	Nepovinný parametr pro požadovanou obsluhu virtuálního spoje. Mohou být použity klíčová slova: SLOW, FAST, FASTEST, ONDEMAND (viz dále)	klíčové slovo

Tabulka typů proměnné pro sdílení (3.parametr instrukce DEF_IN, DEF_OUT):

Typ	popis
TYPE_BIT	bitová proměnná
TYPE_INT_8	celočíselný typ se znaménkem – 8 bitů
TYPE_UNS_8, TYPE_BYTE	celočíselný typ bez znaménka – 8 bitů
TYPE_INT_16	celočíselný typ se znaménkem – 16 bitů
TYPE_UNS_16, TYPE_WORD	celočíselný typ bez znaménka – 16 bitů
TYPE_INT_32	celočíselný typ se znaménkem – 32 bitů
TYPE_UNS_32, TYPE_DWORD	celočíselný typ bez znaménka – 32 bitů
TYPE_INT_64	celočíselný typ se znaménkem – 64 bitů
TYPE_UNS_64, TYPE_QWORD	celočíselný typ bez znaménka – 64 bitů
TYPE_REAL	reálný typ
TYPE_STR	textový řetězec
TYPE_BIN	binární řetězec

Požadovaná obsluha virtuálního spoje (5.parametr instrukce DEF_IN, DEF_OUT):

Period	popis
SLOW	Systém zabezpečí periodickou obsluhu virtuálního spoje v taktu základní smyčky PLC (20 ms). Defaultní obsluha. (viz dále).
FAST	Systém zabezpečí periodickou obsluhu virtuálního spoje v rychlém taktu interpolátoru a servosmyček (1 ms). Pro rychlou obsluhu spoje je nutné, aby logický vstup a současně i logický výstup měly nastaveny atributy FAST. (viz dále).
FASTEST	Systém zabezpečí periodickou obsluhu virtuálního spoje v nejrychlejším taktu pro EtherCAT (250 us). (viz dále).
ONDEMAND	Na vyžádání. Systém nezabezpečí periodickou obsluhu spoje. PLC program má možnost okamžitě zjistit stav logického vstupu, na který směruje virtuální spoj, pomocí instrukce UPDATE_IN (viz dále).

Příklad:

```
MODULE_INIT

      DEF_IN      inLimX,      'inLimX',   TYPE_BIT, -, FAST
      DEF_IN      flBitIn1,    'BitIn1',   TYPE_BIT
      DEF_OUT     flBitOut,    'BitOut',   TYPE_BIT, 1

      DEF_IN      dwBunIn1,    'BunIn1',   TYPE_DWORD, 222, ONDEMAND
      DEF_OUT     rBunOut1,    'BunOut1',  TYPE_REAL,  1.7

      DEF_IN      dwBunIn2[5],'BunIn2',   TYPE_DWORD
      DEF_OUT     rBunOut2[8],'BunOut2',  TYPE_REAL


MODULE_INIT_END
```

27.7 Stav logického vstupu na vyžádání z PLC programu

PLC program má možnost zjistit okamžitě stav logického vstupu, na který je nastaven „virtuální spoj“. Čas zjištění aktuální hodnoty logického vstupu je tak určen PLC programem a nevyužije se periodická obsluha spoje pomocí systémových prostředků.

Požadovaná obsluha virtuálního spoje se udává jako 5. parametr v instrukcích `DEF_IN` a `DEF_OUT`. V tomto případe se musí nastavit do 5. parametru klíčové slovo „**`ONDEMAND`**“.

instrukce	**`UPDATE_IN`**

`funkce` **`UPDATE_IN`** **Okamžité zjištění stavu propojeného logického vstupu**

`syntax` **`UPDATE_IN val`**
`UPDATE_IN val[x]`

`1.parametr` **„val“** **název bitu nebo proměnné**

parametr	název	význam	typ
1.	**`val`**	Název bitu nebo datové proměnné pro vstup. Název proměnné musí být stejný, jaký je použit u instrukci `DEF_IN`, kde je proměnná definovaná a kde jsou další informace potřebné pro logický spoj. Název může obsahovat číslo prvku v hranaté závorce. V tomto případě se jedná a prvek pole a zjistí se aktuální stav jen toho jednoho prvku z pole.	Data podle typu

Příklad:

V inicializaci jsou definovány logické vstupy:

```
DEF_IN      flBitIn1,    'BitIn1',    TYPE_BIT,   -,     ONDEMAND
DEF_IN      dwBunIn1,    'BunIn1',    TYPE_DWORD, 222,   ONDEMAND
DEF_IN      rBunIn2[10],'BunIn2',    TYPE_REAL,  -,     ONDEMAND
```

Na vhodných místech v PLC programu je čtení log.vstupů „na vyžádání“.

```
UPDATE_IN   flBitIn1
UPDATE_IN   dwBunIn1
UPDATE_IN   rBunIn2[6]
```

27.8 Perioda obsluhy virtuálního spoje

Systém zabezpečí periodickou obsluhu virtuálního spoje. Jako požadovaná perioda obsluhy spoje se nastaví maximální hodnota prvků, mezi kterými je spoj definován.

27.8.1 Perioda obsluhy „SLOW“

Systém zabezpečí periodickou obsluhu virtuálního spoje v taktu základní smyčky PLC (20 ms). Jedná se o defaultní periodu obsluhy (instrukce `DEF_IN` a `DEF_OUT` nemusí mít 5. parametr vyplněný).
Propojení spojů s periodou SLOW se provede před voláním běžného průchodu PLC a také po volání běžného průchodu PLC.

27.8.2 Perioda obsluhy „FAST“

Systém zabezpečí periodickou obsluhu virtuálního spoje v taktu interpolátoru, servosmyček a PLC modulu `MODULE_FAST` (1 ms). Propojení spojů s periodou FAST se provede tehdy, když instrukce `DEF_IN` a `DEF_OUT` mají 5.parametr nastavený na FAST a druhý prvek spoje má možnost periody obsluhy alespoň FAST nebo FASTEST.

Data propojované s periodou FAST jsou určeny pro zpracování v modulu `MODULE_FAST` a propojení spojů se provede před jeho průchodem. To samé platí i pro případné použití funkce RTMDLL: `RtmFunServo()`.

Pro cizí reálný čas RTX může být v PLC použit modul `MODULE_CYCLICOPERATION`, který běží v taktu FAST (1 ms). Tento modul se časově zpracovává na začátku obsluhy smyčky FAST. V tomto případě propojení spojů se provede i před průchodem tohoto modulu. Druhé propojení spojů se provede standardně po jeho průchodu a před průchodem `MODULE_FAST`.

27.8.3 Perioda obsluhy „FASTEST“

Perioda obsluhy FASTEST se používá jen pro cizí reálný čas RTX. Systém zabezpečí periodickou obsluhu virtuálního spoje v taktu FASTEST, který je definován nastavením v registru `FastestPeriod`. Pokud je v registru defaultní hodnota, tak se použije nastavení podle `MasterCycleTime` pro sběrnici EtherCAT. Defaultní hodnota pro periodu `MasterCycleTime` je 1000 µs (podle cyklu EtherCAT Mastra). Perioda může nabývat hodnot: 100, 200, 250, 500, 1000 µs.

Pokud existují virtuální spoje FASTEST, takže instrukce `DEF_IN` a `DEF_OUT` mají 5.parametr nastavený na FASTEST a druhý prvek spoje má možnost periody obsluhy FASTEST, systém v tomto rastru volá modul `MODULE_FASTEST` a také funkci RTMDLL: `RtmFunPlcFastest()`. Propojení spojů FASTEST se provede před i po průchodu modulu `MODULE_FASTEST`.

18

18. SDÍLENÁ A ZÁLOHOVANÁ PAMĚŤ, PŘÍMÉ SDÍLENÍ PROMĚNNÝCH

18.1 Zálohovaná paměť PLC

18.1.1 Práce se zálohovanou pamětí

PLC program má k dispozici zálohovanou paměťovou oblast: **PLC_MEM_BACKUP** o velikosti 10000 bajtů. Při regulérním ukončení systému se automaticky oblast uloží na disk. Při startu systému se oblast automaticky načte. PLC program má možnost si vyžádat okamžité uložení oblasti na disk.

REQ_BACKUP_MEM ....žádost o uložení zálohovaných oblastí

(bit, čtení/zápis)

PLC program nastavením hodnoty log.1 do bitu **REQ_BACKUP_MEM** způsobí okamžité uložení paměťové oblasti `PLC_MEM_BACKUP[10000]` na disk. Po uložení zálohované oblasti na disk, systém bit `REQ_BACKUP_MEM` vynuluje.

Příklad:
Uložení oblasti PLC_MEM_BACKUP v mechanizmu s čekáním na provedení akce.

```
      FL    1,REQ_BACKUP_MEM        ;žádost o uložení na disk
      EX
      LDR   REQ_BACKUP_MEM          ;čeká na uložení
      EX1
```

Pro orientaci v zálohované paměti PLC je možno využít symbolické offsety, datové struktury nebo indexaci.

Příklady:

```
EQUI  PRVEK_A,50        ;definice symbolického offsetu

      STO   WORD.PLC_MEM_BACKUP+PRVEK_A        ;přístup s offsetem

;...
```

```
ALFA  struc             ;deklarace struktury
      PRVEK_A     DB    ?
      PRVEK_B     DW    ?
ALFA  ends

      STO   WORD.PLC_MEM_BACKUP.PRVEK_B        ;přístup podle struktury

;...

      LOD   MemIdx                             ;načte index (offset)
      MOV   BX,CX
      LOD   WORD.PLC_MEM_BACKUP[BX]            ;přístup podle indexu

;...
```

Pro zápis větších datových oblastí je možno použít instrukci „MV“ (viz „Základní instrukce PLC“).

Příklady:

```
AREA1:      DS    100

      MV    AREA1,PLC_MEM_BACKUP,100      ;kopíruje z AREA1 do zál.paměti
      MV    AREA1,PLC_MEM_BACKUP+20,100
      MV    AREA1+10,PLC_MEM_BACKUP.PRVEK_B,100
      MV    PLC_MEM_BACKUP.PRVEK_B,AREA1,100
```

18.1.2 Zálohování dat při vypínání systému

Pro zálohování dat při vypínání systému možno použít nepovinný modul „MODULE_DONE“ (viz „Struktura PLC programu“). Doporučuje se ale kromě toho provádět i průběžné zálohování pomocí bitu „REQ_BACKUP_MEM“ pro případ výpadku proudu nebo zaběhnutí programu.

Modul „MODULE_DONE“ slouží k činnosti potřebné při stopu PLC, např. k deaktivaci pohonů a k zálohování. Pokud požadujeme zálohování dat při závěrečných operacích PLC které není jednoprůchodové (například se zálohují data ze sdílené paměti), tak se systému musí dát zpráva o ukončení této činnosti pomocí instrukce **MODULE_DONE_FINISHED.** Táto instrukce je nepovinná a pokud nebude použita, tak ke stopu dojde hned po průchodu modulem MODULE_DONE. Pokud ale v daném souboru je instrukce MODULE_DONE_FINISHED použita, systém čeká s pokračováním činnosti při ukončování až do doby průchodu touto instrukcí (až potom se vypne systém). V tomto případě je vhodné v modulu MODULE_DONE aktivovat mechanizmus, který je umístěn standardně v modulu MODULE_MAIN a v něm na konci po ukončení činnosti je použita instrukce MODULE_DONE_FINISHED.

Příklad:

```
EQUI        BSTATE0,200                  ;offset pro STATE0
EQUI        BSTATE1,204                  ;offset pro STATE1
STATE0:     DS    4                      ;zálohovaná buňka STATE0
STATE1:     DS    4                      ;zálohovaná buňka STATE1

MODULE_DONE                              ;Začátek modulu ukončení
      FL          1,MECH_ZALOHA          ;Start mechanizmu zálohování
MODULE_DONE_END
```

```
MODULE_MAIN                              ;umístěno v MODULE_MAIN
      ;....
      MECH_BEGIN  MECH_ZALOHA            ;Mechanizmus zálohování
        SA_READ   0,OFFS_STATE0,4,STATE0       ;načte STATE0 z SA
        EX0
        SA_READ   0,OFFS_STATE1,4,STATE1       ;načte STATE1 z SA
        EX0
        MV        AREA1,PLC_MEM_BACKUP,100     ;kopíruje z AREA1
        LOD       STATE0
        STO       DWRD.PLC_MEM_BACKUP+BSTATE0  ;záloha STATE0
        LOD       STATE1
        STO       DWRD.PLC_MEM_BACKUP+BSTATE1  ;záloha STATE1
        MODULE_DONE_FINISHED             ;Konec čekání na zálohování
      MECH_END    MECH_ZALOHA
```

18.2 Přímé sdílení proměnných

18.2.1 Instrukce pro sdílení PLC proměnných

Systémová část software a PLC program mají možnost poskytnout své proměnné pro přímé sdílení. V některých případech se tak může značně zjednodušit přístup uživatelských obrazovek a dialogů k PLC proměnným, protože se nemusí proměnná přepisovat do sdílené paměti pomocí instrukce SA_WRITE a nemusí se definovat PLC konstanta pro orientaci ve sdílené paměti. Přímo sdílené proměnné je možno také jednoduše sledovat pomocí osciloskopu.

Systémová část software sekundárního procesoru poskytuje pro sdílení pevně stanovený seznam proměnných (viz. „Tabulka systémových sdílených proměnných“).

PLC program si může určit, které proměnné poskytne pro přímé sdílení pomocí instrukcí **SHARE_VAR** a **SHARE_BIT**.

Přímo sdílené proměnné definované pomocí SHARE_VAR a SHARE_BIT mají automaticky nastaven příznak OUTPUT a tak slouží jen pro výstup z PLC programu. Pokud požadujeme zapisovat do sdílené proměnné, například při definici „virtuálního spoje“, nebo ze vstupního HTML dialogu, musíme pro definici sdílení použít instrukci **DEF_IN**, která je popsána v návode: **„Logické vstupy a výstupy modulů systému.“**

Všechny logické a analogové vstupy a výstupy mají automaticky nastaveno přímé sdílení a tak se značně zjednoduší přístup uživatelských obrazovek a dialogů k jejím hodnotám. Na logické vstupy a výstupy není potřeba používat instrukce SHARE_VAR a SHARE_BIT.

instrukce	**SHARE_VAR**

```
funkce      SHARE_VAR   sdílení datové proměnné definované v PLC


syntax      SHARE_VAR   val1,  'TEXT2', type3
            SHARE_VAR   val1,  'TEXT2', type3 [, min4] [, max5]
            SHARE_VAR   poin1, 'TEXT2', type3 [, len4]
            SHARE_VAR   val1,   poin2 , type3

1.parametr  „val1, poin1“     pointer nebo název sdílené proměnné
2.parametr  „poin2,text2“     pointer nebo textový řetězec jména sdílení
3.parametr  „type3“           typ dat
4.parametr  „min4, len4“      název proměnné pro minimum (REAL), nebo délka dat
5.parametr  „max5“            název proměnné pro maximum (REAL)
```

Instrukce má návratové hodnoty:

RLO=1 ... operace dokončena bez chyb
RLO=0 ... operace dokončena, ale proměnná nebyla zařazena pro sdílení

Popis parametrů instrukce SHARE_VAR

parametr	název	význam	typ
1.	**poin1**	Ukazatel na buffer poskytnutý pro sdílení - náveští u řetězce definovaného instrukcí "str". Parametr může mít zadán offset v řetězci (+xx, +BX).	pointer
	val1	název datové proměnné poskytnuté pro sdílení Parametr může mít zadán offset v řetězci (+xx, +BX). - typ BYTE, WORD, DWRD,REAL..	data
2.	**poin2**	Ukazatel na buffer, kde je umístěný text s jménem sdílení - náveští u řetězce definovaného instrukcí "str".	pointer
	text2	Přímé zadání textu s jménem sdílení - text je zadán v apostrofech	řetězec
3.	**type3**	Zadání typu proměnné (viz dále)	typ
4.	**min4** **len4**	název proměnné pro určení minima, musí být typu REAL (nepovinné) nebo délka dat pro binární řetězec	data
5.	**max5**	název proměnné pro určení maxima, musí být typu REAL (nepovinné)	data

Tabulka typů proměnné pro sdílení (3.parametr instrukce SHARE_VAR):

Typ	popis
TYPE_INT_8	celočíselný typ se znaménkem – 8 bitů
TYPE_UNS_8, TYPE_BYTE	celočíselný typ bez znaménka – 8 bitů
TYPE_INT_16	celočíselný typ se znaménkem – 16 bitů
TYPE_UNS_16, TYPE_WORD	celočíselný typ bez znaménka – 16 bitů
TYPE_INT_32	celočíselný typ se znaménkem – 32 bitů
TYPE_UNS_32, TYPE_DWORD	celočíselný typ bez znaménka – 32 bitů
TYPE_INT_64	celočíselný typ se znaménkem – 64 bitů
TYPE_UNS_64, TYPE_QWORD	celočíselný typ bez znaménka – 64 bitů
TYPE_REAL	reálný typ
TYPE_STR	textový řetězec
TYPE_BIN	binární řetězec

Instrukce SHARE_VAR musí být umístěna nebo volána v modulu „MODULE_INIT".
Při ladění PLC programu, v okamžiku načítání nového PLC, všem sdíleným proměnným skončí platnost po okamžik nového načtení okna nebo dialogu se sdílenou proměnnou.

Příklad:

```
dwBUN1:     DS    4     ;buňka - typ DWORD
rBUN2:      DS    8     ;buňka – typ REAL
rMAX:       DS    8     ;buňka pro maximum - typ REAL
rMIN:       DS    8     ;buňka pro minimum – typ REAL
bArray:     DS    20    ;pole 20xByte
sTXShare:   STR   32,'Sdileni text'
```

```
MODULE_INIT

      LOD   CNST.0.0          ;naplní minimum a maximum
      STO   REAL.rMIN         ;jako reálné hodnoty
      LOD   CNST.200.0
      STO   REAL.rMAX

      SHARE_VAR  dwBUN1,    'SHARE_DWORD', TYPE_DWORD
      SHARE_VAR  rBUN2,     'DOBA_MAZANI', TYPE_REAL
      SHARE_VAR  dwBUN1,    'TLAK',        TYPE_DWORD, rMIN, rMAX
      SHARE_VAR  bArray+12,'PRVEK12',     TYPE_INT8

      SHARE_VAR  sTXShare, 'TEXT1', TYPE_STR
      SHARE_VAR  bArray,   'AREA2', TYPE_BIN, 10

MODULE_INIT_END
```

instrukce SHARE_BIT

funkce **SHARE_BIT sdílení bitové proměnné definované v PLC**

syntax **SHARE_BIT val1, 'TEXT2', bit3**
SHARE_VAR val1, poin2 , bit3

1.parametr **„val1"** **název Bajtu, kde je definován sdílený bit**
2.parametr **„poin2,text2"** **pointer nebo textový řetězec jména sdílení**
3.parametr **„bit3"** **jméno sdíleného bitu**

Instrukce má návratové hodnoty:

RLO=1 ... operace dokončena bez chyb
RLO=0 ... operace dokončena, ale bitová proměnná nebyla zařazena pro sdílení

parametr	název	význam	typ
1.	**val1**	Název Bajtové proměnné, ve které je definován sdílený bit (například návěští u instrukce „DFM"). Parametr může mít zadán offset v řetězci (+xx, +BX). - typ BYTE	data
2.	**poin2**	Ukazatel na buffer, kde je umístěný text s jménem sdílení - návěští u řetězce definovaného instrukcí "str".	pointer
	text2	Přímé zadání textu s jménem sdílení - text je zadán v apostrofech	řetězec
3.	**bit3**	Jméno bitové proměnné - definované pomocí instrukce DFM	bit

Instrukce SHARE_BIT musí být umístěna nebo volána v modulu „MODULE_INIT".
Při ladění PLC programu, v okamžiku načítání nového PLC, všem sdíleným proměnným skončí platnost po okamžik nového načtení okna nebo dialogu se sdílenou proměnnou.

První parametr je typicky název Bajtové proměnné, ve které je definován sdílený bit. V tomto případě je to návěští u instrukce „DFM". Při použití složitější adresace bitu nebo u bitových polí, to může být jiné jméno Bajtu, než je to, ve kterém je bit definován. V tomto případě se u prvního parametru může použít také offset (+xx, +BX).

Všechny logické vstupy a výstupy mají automaticky nastaveno přímé sdílení a tak na logické vstupy a výstupy není potřeba používat instrukce SHARE_VAR a SHARE_BIT.

Příklad:

```
PAM100:     DFM   ,,flPressOn,,,,,
IP0:        DFM   ,inLimXMin,,,,inLimXMax,,,

MODULE_INIT

      SHARE_BIT   PAM100,'PRESS_ON',flPressOn
      SHARE_BIT   IP0,'LIMIT_X_MAX',inLimXMax

      SHARE_BIT   CAN_DRIVE_STAT+2,'CAN_Y_READY',CAN_AX_READY
      SHARE_BIT   CAN_DRIVE_STAT+4,'CAN_Z_READY',CAN_AX_READY

MODULE_INIT_END
```

18.2.2 Sdílení systémových proměnných

Systémová část software sekundárního procesoru poskytuje pro sdílení pevně stanovený seznam proměnných. Umožní se tak přístup uživatelských obrazovek a dialogů k systémovým proměnným úplně bez účasti PLC. Přímo sdílené proměnné je možno také jednoduše sledovat pomocí osciloskopu.

Při sestavování názvu pro sdílenou systémovou proměnnou je umožněno v některých případech používat indexaci v poli a orientaci ve struktuře. Dále je uveden seznam možných typu a zápisů sdílených proměnných.

Možnosti pro zápis jména proměnných pro přímé sdílení:

Zápis	Popis
`Name`	Přímé sdílení jednoduché systémové proměnné
`Name[xx]`	a) typy: `UNS8,INT8,UNS16,INT16,UNS32,INT32,UNS64,INT64,REAL` Systémová proměnná je prvek pole. Zadává se vždy index do pole od nuly (xx=0,1,2..). b) typ `BIT` Anonymní přistup k bitu pro proměnné o velkosti 8, 16 nebo 32 bitů. Index udává váhu bitu (xx=0,1,2..).
`Name.Item`	a) typy: `UNS8,INT8,UNS16,INT16,UNS32,INT32,UNS64,INT64,REAL` Systémová proměnná je prvek struktury. b) typ: `BIT` Přístup k pojmenovanému bitu pro proměnné o velkosti 8, 16 nebo 32 bitů.
`Name[xx].Item`	typ `BIT` přístup k pojmenovanému bitu k prvku pole o velikosti 8, 16 nebo 32 bitů. Zadává se vždy index do pole od nuly (xx=0,1,2..).
`Name.Item[xx]`	a) typy: `UNS8,INT8,UNS16,INT16,UNS32,INT32,UNS64,INT64,REAL` Systémová proměnná je prvek struktury, která je pole. Zadává se vždy index do pole od nuly (xx=0,1,2..). b) typ `BIT` Anonymní přistup k bitu pro prvek struktury o velkosti 8, 16 nebo 32 bitů. Index udává váhu bitu (xx=0,1,2..).

Použité zkratky pro typy:

Typ	popis
INT8	celočíselný typ se znaménkem – 8 bitů
UNS8	celočíselný typ bez znaménka – 8 bitů
INT16	celočíselný typ se znaménkem – 16 bitů
UNS16	celočíselný typ bez znaménka – 16 bitů
INT32	celočíselný typ se znaménkem – 32 bitů
UNS32	celočíselný typ bez znaménka – 32 bitů
INT64	celočíselný typ se znaménkem – 64 bitů
UNS64	celočíselný typ bez znaménka – 64 bitů
REAL	reálný typ
BIT	bit
STRUC	struktura

Příklady zápisu:

BlockInProgress	Blok rozpracován, typ BIT
Pos5_Ax[2]	Poloha 3.osy v POS5, typ REAL [mm]
PotControl_PotF.PT_VAL	Hodnota potenciometru %F v promile, typ UNS16
ActualBlock_Flags.SelBlock	První blok po volbě bloku, typ BIT
ActualBlock_RPlcParams[2]	Programovaná reálná hodnota RPLC2, typ REAL
InputOutput_CAN2_0.INOUT_IN[1]	Fyzické vstupy z 2.portu 1.INOUT08, typ UNS8
InputOutput_CAN2_0.INOUT_IN2[1]	2.bit na 3.portu vstupů 1.INOUT08, typ BIT
DriveStatus_CAN1[2]	Status 3.pohonu CAN-BUS, typ UNS16
DriveStatus_CAN1[2].CAN_AX_ENBLD	Stav „Enable“ 3.pohonu CAN-BUS, typ BIT
InputPort[5]	6.port logických vstupů, typ UNS8
InputPort[5].BIT2	3.bit na 6.portu logických vstupů, typ BIT

18.2.3 Tabulka systémových sdílených proměnných

(stav pro revizi 745)

Název	Typ	Popis
Aktuální blok v RTM		
ActualBlock_BlockType	UNS32	Typ aktuálního bloku (btAut/btAutIns/btCAnul/btRef)
ActualBlock_Flags.	UNS32 UNS32.BIT	Příznaky pro aktuální blok (možnost čtení po bitech)
. FirstBlock	prvek bitové struktury	První blok programu určený pro předání modulu reálného času
. LastBlock	prvek bitové struktury	Poslední blok programu určený pro předání modulu reálného času
. StopBlock	prvek bitové struktury	Je v bloku programován stop (M0)?
. CondStop	prvek bitové struktury	Je v bloku programován podmíněný stop
. Backward	prvek bitové struktury	Jede se blok pozpátku (couvání)
. SelBlock	prvek bitové struktury	První blok po volbě bloku
. Cont	prvek bitové struktury	První blok po CONT?
. TchlOff	prvek bitové struktury	Má se blok jet bez technologie
. Partial	prvek bitové struktury	Neúplný blok
ActualBlock_H	REAL	Programovaná hodnota adresy H
ActualBlock_T	INT32	Programovaná hodnota adresy T
ActualBlock_AddrProgr.	UNS32 UNS32.BIT	Změnové bity pro aktuální blok (možnost čtení po bitech)
. Hprog	prvek bitové struktury	Programována adresa H
.Tprog	prvek bitové struktury	Programována adresa T
ActualBlock_M[xx]	pole UNS32 (xx = 0,1,..,14)	Hodnoty programovaných M funkce jednotlivých skupin
ActualBlock_MProgr[xx]	pole BIT (xx = 0,1,..,14)	Programované M funkce jednotlivých skupin
ActualBlock_IPlcParams[xx]	pole INT32 (xx = 0,1,..,4)	Programované celočíselné hodnoty pro PLC
ActualBlock_IPlcParamsChanged[xx]	pole BIT (xx = 0,1,..,4)	Programované celočíselné hodnoty pro PLC – změny
ActualBlock_RPlcParams[xx]	pole REAL (xx = 0,1,..,4)	Programované reálné hodnoty pro PLC
ActualBlock_RPlcParamsChanged[xx]	pole BIT (xx = 0,1,..,4)	Programované reálné hodnoty pro PLC– změny
ActualBlock_ModeProgr.	UNS32 UNS32.BIT	Příznaky bloku
. ContinuousMode	prvek bitové struktury	Plynule navázat blok na předchozí blok
. RevolutionFeed	prvek bitové struktury	Otáčková rychlost (G95)

	. ConstCutSpeed	prvek bitové struktury	Použít konstantní řeznou rychlost
	. TouchProbeMeas	prvek bitové struktury	Provést v bloku měření dotykovou sondou?
	. TouchProbeFallingEdge	prvek bitové struktury	Reagovat na sestupnou hranu dotykové sondy?
	. TouchProbeCont	prvek bitové struktury	Po příchodu hrany ze sondy pokračovat až do cílového bodu
ActualBlock_InterpolationType		UNS32	Typ interpolace
ActualBlock_Delay		REAL[s]	Časová prodleva programovaná v bloku
ActualBlock_AxNC[xx]		pole REAL[mm] (xx = 0,1,..,15)	Absolutní poloha NC os v POS0
ActualBlock_NCMoveProgr[xx]		pole BIT (xx = 0,1,..,15)	Příznaky pohybu pro jednotlivé NC osy
ActualBlock_AxG[xx]		pole UNS32 (xx = 0,1,2)	Definice geometrických os
ActualBlock_TFlags[xx]		pole BIT (xx = 0,1,..,4)	Příznaky změny transformací / posunutí
ActualBlock_PTMatrix[xx]		pole REAL (xx = 0,1,..,15)	Programová transformace (řazeno po sloupcích)
ActualBlock_PTInvMatrix[xx]		pole REAL (xx = 0,1,..,15)	Inverzní programová transformace
ActualBlock_WTMatrix[xx]		pole REAL (xx = 0,1,..,15)	Transformace polotovaru (řazeno po sloupcích)
ActualBlock_WTInvMatrix[xx]		pole REAL (xx = 0,1,..,15)	Inverzní transformace polotovaru
ActualBlock_LenComp[xx]		pole REAL[mm] (xx = 0,1,2)	Délkové korekce
ActualBlock_Offs1[xx]		pole REAL[mm] (xx = 0,1,..,15)	Vektory posunutí 1
ActualBlock_Offs2[xx]		pole REAL[mm] (xx = 0,1,..,15)	Vektory posunutí 2
ActualBlock_BlockLen		REAL[mm]	Délka dráhy bloku
ActualBlock_VectorC[xx]		pole REAL[mm] (xx = 0,1,2,3)	Informace pro interpolaci kružnice
ActualBlock_HalfRevCount		UNS32	Počet půlotáček pro kruhovou interpolaci
ActualBlock_Axis1		UNS32	Číslo první osy - udává interpolační rovinu
ActualBlock_Axis2		UNS32	Číslo druhé osy - udává interpolační rovinu
ActualBlock_Helix		UNS32	Vytváření šroubovice
ActualBlock_SpiralPitch		REAL[mm/ot]	Stoupání šroubovice
ActualBlock_SpiralAxis		UNS32	Číslo osy pro šroubovici
ActualBlock_ThreadRunInLen		REAL[mm]	Délka nájezdu závitu
ActualBlock_ThreadRunInAngle		REAL[rad]	Úhel nájezdu závitu
ActualBlock_ThreadRunOutLen		REAL[mm]	Délka výjezdu závitu
ActualBlock_ThreadRunOutAngle		REAL[rad]	Úhel výjezdu závitu

ActualBlock_Scale	REAL	Výsledné měřítko
ActualBlock_FeedProgr	REAL [mm/min]	Programovaná rychlost suportu pro pracovní posuv
ActualBlock_FeedMax	REAL [mm/min]	Maximální rychlost suportu
ActualBlock_FeedCriterial	REAL [mm/min]	Kriteriální rychlost suportu na začátku bloku
ActualBlock_AccelerationType	UNS32	Způsob rampování
ActualBlock_SRampJerk	REAL [m/s^3]	Jerk (Parabolické zrychlení)
ActualBlock_LinearAccel	REAL [m/s^2]	Lineární zrychlení
ActualBlock_SpindleSpeed	REAL [ot/min]	Otáčky vřetene
ActualBlock_SpindleSpeedLimit	REAL [ot/min]	Limit otáček vřetene pro konstantní řeznou rychlost
ActualBlock_ConstCuttingSpeed	REAL	Rychlost pro režim konstantní řezné rychlosti
ActualBlock_Reference_Type	UNS32	Typ reference
ActualBlock_Reference_RefAxisMask	UNS32	Maska os pro referenci
ActualBlock_FiveAxTType	UNS32	Typ pětiosé transformace (podle způsobu zadávání a interpolace)
ActualBlock_ToolVect	pole REAL (xx = 0,1,2)	Programovaná orientace nástroje na konci bloku nebo zadání úkosu
ActualBlock_ToolLen	REAL[mm]	Délka nástroje pro pětiosou transformaci
Periferie CAN-BUS		
InputOutput_CAN2_0.	STRUC INOUTS	Systémová struktura pro 1.periferii INOUT
.INOUT_PRESENT	UNS8 prvek struc INOUTS	Je INOUT přítomna ?
.INOUT_MODE	UNS8 prvek struc INOUTS	mód INOUT (0=standard, 1=analog, 2=matice)
.INOUT_VENDORID	UNS32 prvek struc INOUTS	Výrobce INOUT
.INOUT_DEVICENAME	STR prvek struc INOUTS	Název jednotky INOUT
.INOUT_HWVERSION	STR prvek struc INOUTS	HW verze INOUT
.INOUT_SWVERSION	STR prvek struc INOUTS	SW verze INOUT
.INOUT_IN[xx]	pole UNS8 (0,1,2,3) prvek struc INOUTS	Fyzické vstupy INOUT
.INOUT_IN0[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Fyzické vstupy INOUT, 1.port
.INOUT_IN1[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Fyzické vstupy INOUT, 2.port
.INOUT_IN2[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Fyzické vstupy INOUT, 3.port
.INOUT_IN3[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Fyzické vstupy INOUT, 4.port
.INOUT_OUT[xx]	pole UNS8 (0,1,2) prvek struc INOUTS	Fyzické výstupy INOUT

	.INOUT_OUT0[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Fyzické výstupy INOUT, 1.port
	.INOUT_OUT1[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Fyzické výstupy INOUT, 2.port
	.INOUT_OUT2[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Fyzické výstupy INOUT, 3.port
	.INOUT_ANL[xx]	pole UNS16 (0,1,2,3) prvek struc INOUTS	Analogové vstupy
	.INOUT_ERROR	UNS8 prvek struc INOUTS	Aktuální chybový registr
	.INOUT_ERR_CODE	UNS8 prvek struc INOUTS	Kód chyby
	.INOUT_ERR_STAT	UNS8 prvek struc INOUTS	Chybové hlášení
	.INOUT_CONTROL	UNS8 prvek struc INOUTS	Řízení INOUT
	.INOUT_COUNT	UNS16 prvek struc INOUTS	Čítač pro time-out
	.INOUT_SEND_REQ	UNS8 prvek struc INOUTS	Žádost o vyslání SDO paketu
	.INOUT_RESP_COUNT	UNS8 prvek struc INOUTS	Čítač příjmu
	.INOUT_TL[xx]	UNS8 (x=0,1,2,3) prvek struc INOUTS	Kódy tlačítek (4x)
	.INOUT_TL0[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Tlačítka po bitech
	.INOUT_TL1[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Tlačítka po bitech
	.INOUT_RPDO_COUNT	UNS16 prvek struc INOUTS	Čítač zařazených RPDO paketů do fronty na vyslání
	.INOUT_TPDO_COUNT	UNS16 prvek struc INOUTS	Čítač přijmutích TPDO paketů
	.INOUT_SHORT_OUT[xx]	pole UNS8 (0,1,2) prvek struc INOUTS	Informace o zkratu výstupů
	.INOUT_SHORT_OUT0[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Informace o zkratu výstupů, 1.port
	.INOUT_SHORT_OUT1[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Informace o zkratu výstupů, 2.port
	.INOUT_SHORT_OUT2[xx]	pole BIT (x = 0,1,..,7) prvek struc INOUTS	Informace o zkratu výstupů, 3.port
	.INOUT_IRC[xx]	UNS16 (x = 0,1) prvek struc INOUTS	Encodery (2x)
InputOutput_CAN2_1.		STRUC INOUTS	Systémová struktura pro 2.periferii INOUT
InputOutput_CAN2_2.		STRUC INOUTS	Systémová struktura pro 3.periferii INOUT
InputOutput_CAN2_3.		STRUC INOUTS	Systémová struktura pro 4.periferii INOUT
InputOutput_CAN2_4.		STRUC INOUTS	Systémová struktura pro 5.periferii INOUT
InputOutput_CAN2_5.		STRUC INOUTS	Systémová struktura pro 6.periferii INOUT
InputOutput_CAN2_6.		STRUC INOUTS	Systémová struktura pro 7.periferii INOUT
InputOutput_CAN2_7.		STRUC INOUTS	Systémová struktura pro 8.periferii INOUT

Panel_CAN2_0.	STRUC INOUTS	Systémová struktura pro 1.ovládací panel
Panel_CAN2_1.	STRUC INOUTS	Systémová struktura pro 2.ovládací panel
Panel_CAN2_2.	STRUC INOUTS	Systémová struktura pro 3.ovládací panel
Panel_CAN2_3.	STRUC INOUTS	Systémová struktura pro 4.ovládací panel
Knob_CAN2_0.	STRUC INOUTS	Systémová struktura pro 1. panel s ručním točítkem
Knob_CAN2_1.	STRUC INOUTS	Systémová struktura pro 2. panel s ručním točítkem
HandPanel_CAN2_0.	STRUC INOUTS	Systémová struktura pro 1. ruční panel
HandPanel_CAN2_1.	STRUC INOUTS	Systémová struktura pro 2. ruční panel
CountErrOverrun_CAN1	UNS32	Čítač chyb příjmu s přepsáním pro CAN1
CountErrOverrun_CAN2	UNS32	Čítač chyb příjmu s přepsáním pro CAN2
CountErrFull_CAN1	UNS32	Čítač chyb plného bufferu pro vysláni v controleru pro CAN1
CountErrFull_CAN2	UNS32	Čítač chyb plného bufferu pro vysláni v controleru pro CAN2
CountErrInt_CAN1	UNS32	Čítač chyb - za periodu SYNC se vše neodvysílalo pro CAN1
CountErrInt_CAN2	UNS32	Čítač chyb - za periodu SYNC se vše neodvysílalo pro CAN2
Keyb1_PLCCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro PLC 1. panel
Keyb1_RTMCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro RTM 1. panel
Keyb2_PLCCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro PLC 2. panel
Keyb2_RTMCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro RTM 2. panel
Keyb3_PLCCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro PLC 3. panel
Keyb3_RTMCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro RTM 3. panel
Keyb4_PLCCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro PLC 4. panel
Keyb4_RTMCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro RTM 4. panel
Knob1_PLCCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro PLC 1. panel s ručním točítkem
Knob1_RTMCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro RTM 1. panel s ručním točítkem
HandPanel1_PLCCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro PLC 1. ruční panel
HandPanel1_RTMCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro RTM 1. ruční panel
HandPanel2_PLCCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro PLC 2. ruční panel
HandPanel2_RTMCommand[xx]	pole UNS32 (xx = 0,1,..255)	Příkaz po stisku tlačítka pro RTM 2. ruční panel

NumInputOutput_CAN2	UNS8	Počet periferií INOUT na CAN2
NumInputOutput_CAN3	UNS8	Počet periferií INOUT na CAN3
NumInputOutput_CAN4	UNS8	Počet periferií INOUT na CAN4
NumPanel_CAN2	UNS8	Počet ovládacích panelů na CAN2
NumKnob_CAN2	UNS8	Počet panelů s ručním točítkem na CAN2
NumHandPanel_CAN2	UNS8	Počet ručních panelů na CAN2
CanTPDOLostPanel	UNS8	Sledování ztráty paketu pro panel
CanTPDOLostInout[xx]	pole UNS8 (xx = 0,1,..,31)	Sledování ztráty paketu pro jednotky INOUT
InputPort[xx]	pole UNS8 (0,1,..,19) UNS8[xx].BIT	Logické vstupy
OutputPort[xx]	pole UNS8 (0,1,..,14) UNS8[xx].BIT	Logické výstupy
BIT0,BIT1,..,BIT15	prvek bitové struktury	Anonymní přístup k bitovým prvkům
Pohony CAN-BUS		
DriveStatus_CAN1[xx].	pole UNS16 UNS16[xx].BIT	Drive status z pohonu CAN-BUS (CAN_DRIVE_STAT)
.CAN_AX_READY	prvek bitové struktury	Pohon vpořádku
.CAN_AX_ON	prvek bitové struktury	Pohon má aktivní výkonový můstek
.CAN_AX_ENBLD	prvek bitové struktury	Pohon je ve stavu Enable
.CAN_AX_FAULT	prvek bitové struktury	Pohon v poruše
.CAN_AX_VOLTAGE	prvek bitové struktury	Napětí pohonu
.CAN_AX_QSTOP	prvek bitové struktury	Rychlý stop
.CAN_AX_BRKD	prvek bitové struktury	Brzda pohonu
.CAN_AX_WARN	prvek bitové struktury	Upozornění
ManufactoryStatus_CAN1[xx].	pole UNS16 UNS16[xx].BIT	Manufactory drive status (CAN_MDRIVE_STAT)
DriveCommand_CAN1[xx].	pole UNS16 UNS16[xx].BIT	Drive command pro 1.pohon (CAN_DRIVE_CMD)
.CAN_AX_EN	prvek bitové struktury	Příkaz „Enable“
.CAN_EX_BRK	prvek bitové struktury	Příkaz „Brake“
Jednotka odměřování a analog.výstupů SU05, SU06		
SU5_Counter[xx]	pole UNS16 (xx = 0,1,..15)	Čítač odměřování SU05 (SU_IRC0)
SU5_RefBuff[xx]	pole UNS16 (xx = 0,1,..15)	Buffer reference SU05 (SU_REF0)
SU5_Uout[xx]	pole REAL[10V] (xx = 0,1,..15)	Výstupní napětí pro analogové výstupy SU05

SU5_Status[xx].	pole UNS8 UNS8[xx].BIT	Status jednotky SU05
. SU_ErrIrc	prvek bitové struktury	Zkrat nebo přerušení snímače IRC
.SU_ErrAnl	prvek bitové struktury	Špatná funkce analogových výstupů
.SU_ErrWrite	prvek bitové struktury	Nesprávný počet zápisů
.SU_ErrBoard	prvek bitové struktury	Chyba v připojení na sběrnici
.SU_ErrHver	prvek bitové struktury	Chyba verze hardware jednotky SU05
. SU_ErrSU67	prvek bitové struktury	Chybí subdeska SU67 u jednotky SU06
SU5_Error[xx].	pole UNS8 UNS8[xx].BIT	Chyby jednotky SU05
.SU_ErrPhase	prvek bitové struktury	Chyba fáze signálů snímače IRC
.SU_ErrCheck	prvek bitové struktury	Chyba kontrolního čítače IRC
EncoderChannelType[xx]	pole UNS16 (xx = 0,1,..15)	Konfigurace „ChannelType“ v elementu „EncoderChannel“
EncoderType[xx]	pole UNS16 (xx = 0,1,..15)	Konfigurace „EncoderType“ v elementu „EncoderChannel“
DriveType[xx]	pole UNS16 (xx = 0,1,..15)	Konfigurace „ DriveType“ v elementu „DriveChannel“
SU5_NIPulse[xx].	pole UNS32 UNS32[xx].BIT	Pole pro sledování nulocých pulsů
. SU_NI	prvek bitové struktury	Nulový impuls
SU5_UoutPlc[xx]	pole REAL[10V] (xx = 0,1,..15)	Výstupní napětí pro analogové výstupy z PLC (SU_OUT_PLC)
SU5_CheckCounter[xx]	pole UNS32 (xx = 0,1,..15)	Čítač odměřování pro kontrolní čítač (SU_CHECK_ODM)
SU5_NICounter[xx]	pole UNS32 (xx = 0,1,..15)	Čítač odměřování SU05 po zónách (SU_CHECK_NI)
SU5_NIAbsCounter[xx]	pole UNS32 (xx = 0,1,..15)	Absolutní poloha při nulovém pulsu (SU_CHECK_NI_ABS)
Potenciometry		
TabSensitivity_PotF[xx]	pole UNS16 (xx = 0,1,..999)	Tabulka průběhu potenciometru %F (promile)
TabSensitivity_PotS[xx]	pole UNS16 (xx = 0,1,..999)	Tabulka průběhu potenciometru %S (promile)
TabSensitivity_Pot1[xx]	pole UNS16 (xx = 0,1,..999)	Tabulka průběhu potenciometru P1 (promile)
TabSensitivity_Pot2[xx]	pole UNS16 (xx = 0,1,..999)	Tabulka průběhu potenciometru P2 (promile)
TabSensitivity_Pot3[xx]	pole UNS16 (xx = 0,1,..999)	Tabulka průběhu potenciometru P3 (promile)
TabSensitivity_Pot4[xx]	pole UNS16 (xx = 0,1,..999)	Tabulka průběhu potenciometru P4 (promile)
TabSensitivity_Pot5[xx]	pole UNS16 (xx = 0,1,..999)	Tabulka průběhu potenciometru P5 (promile)
TabSensitivity_Pot6[xx]	pole UNS16 (xx = 0,1,..999)	Tabulka průběhu potenciometru P6 (promile)
PotControl_PotF.	STRUC PRXWR	Systémová struktura pro řízení potenciometru %F

	.PT_VAL	UNS16 prvek struc PRXWR	Hodnota potenciometru v promile
	.PT_INIT	UNS16 prvek struc PRXWR	Řádek v tabulce po zapnutí systému
	.PT_MAX	UNS16 prvek struc PRXWR	Poslední řádek v tabulce
	.PT_IDX	UNS16 prvek struc PRXWR	Aktuální index do tabulky
	.PT_IRC	UNS16 prvek struc PRXWR	Fyzická hodnota čítače
	.PT_SETVAL	UNS16 prvek struc PRXWR	Hodnota pro přímé nastavení z PLC
	.PT_LNK	UNS8 prvek struc PRXWR	Přiřazení fyzického potenciometru
	.PT_LOCK	UNS8 prvek struc PRXWR	Řízení je umožněno jen z panelu
	.PT_PLC	UNS8 prvek struc PRXWR	Řízení potenciometru je umožněno jen z PLC
	.PT_STAT	UNS8 prvek struc PRXWR	Odkud je řízení potenciometru
	.PT_SET	UNS8 prvek struc PRXWR	Přímé nastavení hodnoty z panelu
PotControl_PotS.		STRUC PRXWR	Systémová struktura pro řízení potenciometru %S
PotControl_Pot1.		STRUC PRXWR	Systémová struktura pro řízení potenciometru P1
PotControl_Pot2.		STRUC PRXWR	Systémová struktura pro řízení potenciometru P2
PotControl_Pot3.		STRUC PRXWR	Systémová struktura pro řízení potenciometru P3
PotControl_Pot4.		STRUC PRXWR	Systémová struktura pro řízení potenciometru P4
PotControl_Pot5.		STRUC PRXWR	Systémová struktura pro řízení potenciometru P5
PotControl_Pot6.		STRUC PRXWR	Systémová struktura pro řízení potenciometru P6
PLC			
CANErrorCode		UNS8	Chybový registr pro PLC od CAN2
CANErrorNum		UNS8	Číslo jednotky v poruše na CAN2
CANTimeOutCode		UNS8	Chyba time-out pro PLC od CAN2
CANTimeOutNum		UNS8	Číslo jednotky v poruše time-out na CAN2
PLCMemBackup[xx]		pole UNS8 (xx = 0,1,..9999)	Zálohovaná oblast PLC paměti (PLC_MEM_BACKUP)
BSPProcNumber		UNS32	Číslo procesoru BSP
SECProcNumber		UNS32	Číslo procesoru SEC
MpPlc_Ax[xx]		pole BIT (xx = 0,1,..15)	Povolení pohybu z PLC pro osy
InRef_Ax[xx]		pole BIT (xx = 0,1,..15)	NC osa v referenci
KnobSel_Ax[xx]		pole UNS8 (xx = 0,1,..15)	Volba osy z točítka pro indikaci

MpMan_Ax[xx]	pole BIT (xx = 0,1,..15)	Povolení pohybu pro osy pro manuální režim (MPxMan)
Mp_Ax[xx]	pole BIT (xx = 0,1,..15)	Celkové povolení pohybu pro osy (MPx)
Move_Ax[xx]	pole BIT (xx = 0,1,..15)	Pohyb v osách (PO_OSxPI)
Direct_Ax[xx]	pole BIT (xx = 0,1,..15)	Směr pohybu pro osy (SM_POxPI)
Coupling_S[xx]	pole BIT (xx = 0,1,..15)	Vazba polohové servosmyčky (VAZBA_x)
Homing_Ax[xx]	pole BIT (xx = 0,1,..15)	Polohování osy (POLOHV_x)
StartDisable	BIT	Zákaz startu (START_DISABLE)
BlockInProgress	BIT	Blok rozpracován (PO_F)
PosReached	BIT	Programovaná poloha dosažena (PO_POL)
StopAck	BIT	Potvrzení stopu (PO_STOP)
InPos	BIT	Poloha v toleranci (INPOS)
ExtFeedOvr	BIT	Externí řízení FeedOverride (FEED_OVR)
FeedOvrVal	UNS16	Hodnota promile pro externí řízení rychlosti
FeedOvrMultiplier	UNS16	Násobící koeficient pro externí řízení rychlosti
ExtSpeedOvr	UNS16	Externí řízení procenta otáček z PLC (SPEED_OVR)
SpeedOvrVal	UNS16	Hodnota promile pro řízení otáček (SPEED_OVR_EXT)
SpeedOvrMultiplier	UNS16	Násobící koeficient pro otáčky
Delay	BIT	Prodleva G04
LimPlus_Ax[xx]	pole BIT (xx = 0,1,..15)	Limitní spínače v kladném směru (KHx0)
LimMinus_Ax[xx]	pole BIT (xx = 0,1,..15)	Limitní spínače v záporném směru (KHx1)
DragLimPlus_Ax[xx]	pole BIT (xx = 0,1,..5)	Zpomalovací limitní spínače v kladném směru (ZPx0)
DragLimMinus_Ax[xx]	pole BIT (xx = 0,1,..5)	Zpomalovací limitní spínače v záporném směru (ZPx1)
SwRefEnable_Ax[xx]	pole BIT (xx = 0,1,..5)	Povolení softwarových spínačů od reference (SLS_REFER)
HomingPoint_Ax[xx]	pole BIT (xx = 0,1,..5)	Referenční spínače (KRx)
DragHoming_Ax[xx]	pole BIT (xx = 0,1,..5)	Zpomalovací referenční spínače (ZPRx)
ModeAut	BIT	Režim AUT (AUTPI)
ModeRefer	BIT	Ržim REFER (REFPI)
ModeCanul	BIT	Režim CANUL (CAPI)
Thread	BIT	Závitování G33 (G33PI)

RapidTraverse	BIT	Rychloposuv G00 (G00PI)
M00	BIT	Dekódovaná funkce M00 1.skupina M (M00_PID)
M01	BIT	Dekódovaná funkce M01 1.skupina M (M01_PID)
M02	BIT	Dekódovaná funkce M02 1.skupina M (M02_PID)
M30	BIT	Dekódovaná funkce M02 1.skupina M (M30_PID)
M03	BIT	Dekódovaná funkce M03 2.skupina M (M03PI)
M04	BIT	Dekódovaná funkce M04 2.skupina M (M04PI)
M19	BIT	Dekódovaná funkce M19 2.skupina M (M19PI)
M41	BIT	Dekódovaná funkce M41 3.skupina M (M41PI)
M42	BIT	Dekódovaná funkce M42 3.skupina M (M42PI)
M43	BIT	Dekódovaná funkce M43 3.skupina M (M43PI)
M44	BIT	Dekódovaná funkce M44 3.skupina M (M44PI)
M07	BIT	Dekódovaná funkce M07 5.skupina M (M07PI)
M08	BIT	Dekódovaná funkce M08 5.skupina M (M08PI)
M50	BIT	Dekódovaná funkce M50 6.skupina M (M50PI)
M51	BIT	Dekódovaná funkce M51 6.skupina M (M51PI)
M10	BIT	Dekódovaná funkce M10 7.skupina M (M10PID)
M11	BIT	Dekódovaná funkce M11 7.skupina M (M11PID)
M06	BIT	Dekódovaná funkce M06 8.skupina M (M06PID)
M60	BIT	Dekódovaná funkce M60 8.skupina M (M60PI)
SF_Run	BIT	Systém v chodu, stroj jede, nebo jsou technologické funkce
SF_BlockInProgr	BIT	Blok rozpracován
SF_BlockFinished	BIT	Blok ukončen
SF_Stop	BIT	Všechny druhy stopu bloku
SF_NotInpos	BIT	Dosud nebylo dosaženo požadované polohy
SF_DelayInProgr	BIT	Časová prodleva
SF_IndicationMode	BIT	Indikační režim
SF_SimulationMode	BIT	Simulační režim
SF_HighspeedMode	BIT	Zrychlený režim

SF_CondstopMode	BIT	Podmíněny stop aktivní (pro bloky s M01)
SF_ManualMode	BIT	Manuální režim
SF_BlockByBlock	BIT	Režim blok po bloku
SF_LS	BIT	Limitní spínač
SF_SLS	BIT	Softwarový limitní spínač
SF_Backward	BIT	Couvání
SF_StopPosChanged	BIT	Bude nastaven, když při stopu programu bylo ručně popojeto
RtmConfig[xx]	pole UNS32 (xx = 0,1,..799)	RTM Konfigurace (BUKON00)
RtmBeginMem[xx]	pole UNS8	Operační paměť pro sekundární procesor
RtmBeginCom[xx]	pole UNS8	Komunikační paměť pro sekundární procesor
BlockToggle	BIT	Klopní bit pro nový blok
Ruční pohyby		
AckMan	BIT	Pomocné ruční pojezdy jsou aktivní (ACK_AUTMAN)
InposStop	BIT	Systém je v poloze posledního stopu (INPOS_STOP)
ManEnable	BIT	Povolení ručních pojezdů (EN_AUTMAN)
ManControlNC	BIT	Řízení MAN je z panelu NC systému (AUTMAN_CONT_NC)
ManControlKnob	BIT	Řízení MAN je z panýlku točítka (AUTMAN_CONT_TOC)
ManControlSelect	BIT	Předvolba pohybu MAN (AUTMAN_CONT_SELECT)
ManMovePlus_Ax[xx]	pole BIT (x = 0,1,..,5)	Pohyb v režimu MAN v kladném směru (MM_x0)
ManMoveMinus_Ax[xx]	pole BIT (x = 0,1,..,5)	Pohyb v režimu MAN v záporném směru (MM_x1)
ManControlRT	BIT	Řízení MAN rychloposuvem (AUTMAN_CONT_G00)
Shift_Ax[xx]	pole BIT (x = 0,1,..,5)	Pohyb v režimu SHIFT (SHIFT_CONTROL)
ExtManMovePlus_Ax[xx]	pole BIT (x = 0,1,..,5)	Externí pohyb v režimu MAN v kladném směru (EXM_x0)
ExtManMoveMinus_Ax[xx]	pole BIT (x = 0,1,..,5)	Externí pohyb v režimu MAN v záporném směru (EXM_x0)
ReqExtManRT	BIT	Externí požadavek pro G00 MAN (REQ_EXT_G00_AUTMAN)
ReqExtManSelect	BIT	Externí požadavek pro předvolbu REQ_EXT_SELECT_AUTMAN
ReqExtMan	BIT	Externí požadavek na pohyb (REQ_EXT_CONT_AUTMAN)
ReqExtFeedMan	BIT	Externí požadavek na rychlost (REQ_EXT_FEED_AUTMAN)
ExtFeedMan	REAL[mm/min]	Hodnota externí rychlosti v MAN (EXT_FEED_AUTMAN)

ExtFeedRTMan	REAL[mm/min]	Hodnota externí rychlosti v MAN (EXT_FEED_G00_AUTMAN)
ReqG95Man	BIT	Požadavek na otáčkový posuv pro MAN (G95_AUTMAN)
Knob1Disable	BIT	Zákaz ovládání z 1. panýlku (AUTMAN_KNOB1_DIS)
Knob2Disable	BIT	Zákaz ovládání z 2. panýlku (AUTMAN_KNOB2_DIS)
IncManPos1_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS1
IncManPos2_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS2
IncManPos3_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS3
IncManPos4_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS4
IncManPos5_Ax	pole REAL[mm/min] (x = 0,1,..,15)	Aktuální přírůstek dráhy MAN v POS5
Řízení os a interpolace		
ReqShift	BIT	Požadavek na režim SHIFT
ActShift	BIT	Režim SHIFT aktivní
KnobStep	UNS16	Krok točítka
ShiftEnable_Ax[xx]	pole BIT (x = 0,1,..,15)	Povolení SHIFT z PLC
Pos5Inc	REAL[mm/T]	Prostorový aktuální přírůstek dráhy v GEO osách v POS5
Pos6Inc	REAL[mm/T]	Prostorový aktuální přírůstek dráhy v GEO osách v POS6
Pos5IncPm	REAL[mm/min]	Prostorový aktuální přírůstek dráhy v GEO osách v POS5
Pos6IncPm	REAL[mm/min]	Prostorový aktuální přírůstek dráhy v GEO osách v POS6
Angle5Ax1	REAL[rad]	1.fyzická osa rotace pro naklápění
Angle5Ax2	REAL[rad]	2.fyzická osa rotace pro naklápění
Pos0_Ax[xx]	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS0
Pos1_Ax[xx]	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS1
Pos2_Ax[xx]	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS2
Pos3_Ax[xx]	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS3
Pos4_Ax[xx]	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS4
Pos5_Ax[xx]	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS5
Pos6_Ax[xx]	pole REAL[mm] (x = 0,1,..,15)	Poloha osy v POS6
ToolVectorX	REAL	Jednotkový prostorový vektor orientace nástroje - složka X

ToolVectorY	REAL	Jednotkový prostorový vektor orientace nástroje - složka Y
ToolVectorZ	REAL	Jednotkový prostorový vektor orientace nástroje - složka Z
Ax1BevelAngle	REAL[rad]	Fyzický úhel 1.rotace při úkosové interpolaci (mezivýsledek)
Ax2BevelAngle	REAL[rad]	Fyzický úhel 2.rotace při úkosové interpolaci (mezivýsledek)
ToolTangentialAngleXY	REAL[rad]	Průmět aktuálního tečnového úhlu úkosu TollTangentialAngle
TangentialAngleAct	REAL[rad]	Aktuální tečnový úhel bloku
ToolBevelAngleAct	REAL[rad]	Aktuální úhel úkosu pro úkosovou interpolaci
ToolTangentialAngleAct	REAL[rad]	Aktuální tečnový úhel úkosu v úkosové rovině
IncPt_Ax[xx]	pole REAL[mm/T] (x = 0,1,.,15)	Aktuální přírůstek dráhy v ose
Inc_Ax[xx]	pole REAL[mm/min] (x = 0,1,.,15)	Aktuální přírůstek dráhy v ose
Pos6Shift_Ax[xx]	pole REAL[mm/min] (x = 0,1,.,15)	Posun SHIFT
ICounter[xx]	pole REAL[mm] (xx = 0,1,.,15)	Diferenční čítač polohové servosmyčky
ICounter_S[xx]	pole INT32[µm] (xx = 0,1,.,15)	Diferenční čítač polohové servosmyčky
ServoPosition[xx]	pole REAL[mm] (xx = 0,1,.,15)	Poloha servosmyček podle odměřování
ServoPosition_S[xx]	pole INT32[µm] (xx = 0,1,.,15)	Poloha servosmyček podle odměřování
ServoPosition_Inc[xx]	pole INT32[inc] (xx = 0,1,.,15)	Poloha servosmyček v inkrementech pohonu
ServoReqPosition[xx]	pole REAL[mm] (xx = 0,1,.,15)	Požadovaná poloha servosmyček
ServoReqPosition_S[xx]	pole INT32[µm] (xx = 0,1,.,15)	Požadovaná poloha servosmyček
CircleAngle	REAL[rad]	Aktuální úhel kruhové interpolace
CircleDist	REAL[mm]	Distance pro kruhovou interpolaci
Distanc_Ax[xx]	pole REAL[mm] (x = 0,1,.,15)	Složkové distance pro jednotlivé osy
Distanc	REAL[mm]	Prostorová distance os
TouchPos0_Ax[xx]	pole REAL[mm] (x = 0,1,.,15)	Poloha osy v POS0 naměřena dotykovou sondou
TouchPos1_Ax[xx]	pole REAL[mm] (x = 0,1,.,15)	Poloha osy v POS1 naměřena dotykovou sondou
TouchPos2_Ax[xx]	pole REAL[mm] (x = 0,1,.,15)	Poloha osy v POS2 naměřena dotykovou sondou
TouchPos3_Ax[xx]	pole REAL[mm] (x = 0,1,.,15)	Poloha osy v POS3 naměřena dotykovou sondou
TouchPos4_Ax[xx]	pole REAL[mm] (x = 0,1,.,15)	Poloha osy v POS4 naměřena dotykovou sondou
TouchPos5_Ax[xx]	pole REAL[mm] (x = 0,1,.,15)	Poloha osy v POS5 naměřena dotykovou sondou
TouchPos6_Ax[xx]	pole REAL[mm] (x = 0,1,.,15)	Poloha osy v POS6 naměřena dotykovou sondou

TouchProbe0Stop	BIT	Stop pohybu od dotykové sondy
TouchProbe0	BIT	Kontakt dotykové sondy
IncCan_Ax[xx]	pole INT32[inc] (xx = 0,1,..,15)	Přírůstky dráhy pro CAN-BUS a EtherCAT trajectory
RTM.Servo[xx].Edrv.State	pole UNS8 (xx = 0,1,..,15)	Stav EtherCAT pohonu EDRV (EDRV_INIT, EDRV_DONE,..)
RTM.Servo[xx].Edrv.MoveDisReq	pole BIT (xx = 0,1,..,15)	Žádost o blokování pohybu od pohonů EDRV
RTM.Servo[xx].Edrv.Error	pole BIT (xx = 0,1,..,15)	Chyba EtherCAT pohonu EDRV
RTM.Servo[xx].Edrv.Simul	pole BIT (xx = 0,1,..,15)	EDRV pohon v simulaci
RTM.Servo[xx].Edrv.Init	pole BIT (xx = 0,1,..,15)	Žádost o INIT EtherCAT pohonu typu EDRV
RTM.Servo[xx].Edrv.Done	pole BIT (xx = 0,1,..,15)	Žádost o DONE EtherCAT pohonu typu EDRV
RTM.Servo[xx].Edrv.Enable	pole BIT (xx = 0,1,..,15)	Žádost o ENABLE EtherCAT pohonu typu EDRV
RTM.Servo[xx].Edrv.Disable	pole BIT (xx = 0,1,..,15)	Žádost o DISABLE EtherCAT pohonu typu EDRV
RTM.Servo[xx].Edrv.On	pole BIT (xx = 0,1,..,15)	Žádost o ON (zapnutí) EtherCAT pohonu typu EDRV
RTM.Servo[xx].Edrv.Erase	pole BIT (xx = 0,1,..,15)	Žádost o ERASE EtherCAT pohonu EDRV
RTM.Servo[xx].Edrv.DriveStatus	pole UNS16 (xx = 0,1,..,15)	[in] „Drive status word“ EDRV pohonu (CoE, SoE)
RTM.Servo[xx].Edrv.DrivePos	pole INT32[inc] (xx = 0,1,..,15)	[in] „Position feedback“ EDRV pohonu
RTM.Servo[xx].Edrv.DriveErrReg	pole UNS16 (xx = 0,1,..,15)	[in] „Drive error registr“ EDRV pohonu
RTM.Servo[xx].Edrv.DriveControl	pole UNS16 (xx = 0,1,..,15)	[out] „Master control word“ EDRV pohonu
RTM.Servo[xx].Edrv.DriveVeloReq	pole INT32[inc] (xx = 0,1,..,15)	[out] „Velocity demand value“ EDRV pohonu
RTM.Servo[xx].Edrv.DrivePosReq	pole INT32[inc] (xx = 0,1,..,15)	[out] „Position demand value“ EDRV pohonu
RTM.Servo[xx].Edrv.ScaleSpeed	pole REAL (x = 0,1,..,15)	[in] Měřítko pro výstup rychlosti EDRV pohonu
RTM.Servo[xx].Enc.CounterValue	pole INT32[inc] (xx = 0,1,..,15)	[in] „Encoder counter value“ (EL5101 EDRV)
RTM.Servo[xx].Enc.LatchValue	pole INT32[inc] (xx = 0,1,..,15)	[in] „Encoder latch value“ (EL5101 EDRV)
RTM.Servo[xx].Enc.LatchEn	pole BIT (xx = 0,1,..,15)	[out] „Encoder Enable latch“ (EL5101 EDRV)
RTM.Servo[xx].Enc.LatchValid	pole BIT (xx = 0,1,..,15)	[in] „Encoder Latch C valid“ (EL5101 EDRV)
Vřeteno		
Spindle1Output	REAL[10V]	Výstupní napětí pro 1.vřeteno (bez procenta S)
Spindle2Output	REAL[10V]	Výstupní napětí pro 2.vřeteno (bez procenta S)
Spindle1PrsOutput	REAL[10V]	Výstupní napětí pro 2. vřeteno (s ohledem na procento S)

ReqSpeedPm	REAL[ot/min]	Požadované otáčky vřetene. (s ohledem na procento S)
ReqSpeedPt	REAL[ot/T]	Požadované otáčky vřetene. (s ohledem na procento S)
ActSpeedPm	REAL[ot/min]	Aktuální okamžité otáčky vřetene
ActSpeed2Pm	REAL[ot/min]	Aktuální okamžité otáčky 2.vřetene
ActAvgSpeedPm	REAL[ot/min]	Aktuální průměrované otáčky vřetene
ActAvgSpeed2Pm	REAL[ot/min]	Aktuální průměrované 2.otáčky 2.vřetene [ot/min]
ActSpeedPt	REAL[ot/T]	Aktuální okamžité otáčky vřetene
ActSpeed2Pt	REAL[ot/T]	Aktuální okamžité otáčky 2.vřetene
ActAvgSpeedPt	REAL[ot/T]	Aktuální průměrované otáčky vřetene
ActAvgSpeed2Pt	REAL[ot/T]	Aktuální průměrované otáčky 2.vřetene
SpindleGearAct1	UNS16	Číslo aktuálního převodového stupně pro 1.vřeteno
SpindleGearAct2	UNS16	Číslo aktuálního převodového stupně pro 2.vřeteno
Nelineární korekce		
NoLinComp0_0.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 0
. NLC_CONTROLLING	UNS8 prvek NOLINCOMPS	Řídící souřadnice
. NLC_ENABLE	UNS8 prvek NOLINCOMPS	Povolení nelineární korekce
. NLC_STEP	REAL[mm] prvek NOLINCOMPS	Krok
. NLC_BEGIN_TAB	REAL[mm] prvek NOLINCOMPS	Počátek tabulky
. NLC_LENGTH_TAB	REAL[mm] prvek NOLINCOMPS	Délka korigované dráhy
NoLinComp0_1.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 1
NoLinComp0_2.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 2
NoLinComp0_3.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 3
NoLinComp0_4.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 4
NoLinComp0_5.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 5
NoLinComp0_6.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 6
NoLinComp0_7.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 7
NoLinComp0_8.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 8
NoLinComp0_9.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 9
NoLinComp0_10.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 10

NoLinComp0_11.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 11
NoLinComp0_12.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 12
NoLinComp0_13.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 13
NoLinComp0_14.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 14
NoLinComp0_15.	STRUC NOLINCOMPS	Struktura pro 1.nelineární korekci pro servosmyčku 15
NoLinComp1_0.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 0
NoLinComp1_1.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 1
NoLinComp1_2.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 2
NoLinComp1_3.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 3
NoLinComp1_4.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 4
NoLinComp1_5.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 5
NoLinComp1_6.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 6
NoLinComp1_7.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 7
NoLinComp1_8.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 8
NoLinComp1_9.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 9
NoLinComp1_10.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 10
NoLinComp1_11.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 11
NoLinComp1_12.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 12
NoLinComp1_13.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 13
NoLinComp1_14.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 14
NoLinComp1_15.	STRUC NOLINCOMPS	Struktura pro 2.nelineární korekci pro servosmyčku 15
NoLinComp1ActVal[xx]	pole REAL[mm] (x = 0,1,..,15)	Aktuální hodnota 1.kompenzace
NoLinComp2ActVal[xx]	pole REAL[mm] (x = 0,1,..,15)	Aktuální hodnota 2.kompenzace
NoLinComp1Active[xx]	pole BIT (x = 0,1,..,15)	1.nelineární korekce je aktivní
NoLinComp2Active[xx]	pole BIT (x = 0,1,..,15)	2.nelineární korekce je aktivní
NlcAxisEnable[xx]	pole BIT (x = 0,1,..,15)	Nelineární korekce, povolení pro řídící souřadnici
NlcServoEnable[xx]	pole BIT (x = 0,1,..,15)	Nelineární korekce, povolení pro servosmyčku od PLC

Diagnostické prvky		
FastRunCount_MIN	UNS64	Minimum z naměřeného času průběhu smyčky FAST
FastRunCount_MAX	UNS64	Maximum z naměřeného času průběhu smyčky FAST
FastPeriodCount	UNS64	Definovaná hodnota periody pro rychlou smyčku FAST
FastRunTime_MIN	REAL [s]	Minimum z naměřeného času průběhu smyčky FAST
FastRunTime_MAX	REAL [s]	Maximum z naměřeného času průběhu smyčky FAST
PLCPeriod	REAL [s]	Definovaná hodnota periody pro základní běh PLC (slow .. 20ms)
PLCPeriodFast	REAL [s]	Definovaná hodnota periody pro rychlý běh PLC (1 ms)
PLCPeriodFastest	REAL [s]	Definovaná hodnota periody pro nejrychlejší běh PLC (200 µs)
FastestModuleCount	UNS32	Diagnostický čítač průběhu MODULE_FASTEST
FastestModulePresent	BIT	Modul MODULE_FASTEST je aktivní
FastestInterrupt	REAL [s]	Definovaná hodnota periody pro nejrychlejší přerušení (200 µs)
MeasStart	BIT	Start pro dávková měření
MeasStop	BIT	Stop pro dávková měření
MeasErr	BIT	Chyba dávkového měření
MeasControl	UNS32	Řízení pro dávková měření
EtherCAT		
EcatDiagnostic.	Struc	Základní diagnostické informace pro EtherCAT
.EcatSlaveCount	UNS16	Skutečný počet Slave připojených na EtherCAT
.EcatSlaveCountReq	UNS16	Požadovaný počet Slave
.EcatSlaveCountMaxDiag	UNS16	Maximální počet Slave pro diagnostiku
.EcatSlaveCountReqDiag	UNS16	Požadovaný počet Slave pro diagnostiku
EcatMasterDiagnostic.	Struc	Diagnostické informace Mastra
.EcatMasterDiagState	UNS32	Stav Mastra (bitové pole)
.EcatValidLicense	UNS16	Licence Mastra
EcatSlaveDiagnostic.	Struc	Diagnostické informace 1.Slave
.EcatSlaveDiagState	UNS32	Stav Slave (bitové pole)
EcatSlaveDiagnostic_1.	Struc	Diagnostické informace 2.Slave

EcatSlaveDiagnostic_1.	Struc	Diagnostické informace 2.Slave
EcatSlaveDiagnostic_2.	Struc	Diagnostické informace 3.Slave
EcatSlaveDiagnostic_3.	Struc	Diagnostické informace 4.Slave
EcatSlaveDiagnostic_4.	Struc	Diagnostické informace 5.Slave
EcatMasterCountDgn.	Struc	Čítače změn stavu Mastra
.EcatMasterUpdateCount	UNS32	Diagnostic completed successfully
.EcatMasterSendRecErrCount	UNS32	Error while sending/receiving a frame
.EcatMasterParseErrCount	UNS32	Error while processing the received frame
.EcatMasterLinkDownCount	UNS32	No connection between the NIC adapter and slaves
.EcatMasterWrongConfCount	UNS32	Wrong configuration
.EcatMasterS2STimeoutCount	UNS32	Slave-to-slave timeout
.EcatMasterDefDataSetCount	UNS32	Default data was set
.EcatMasterWatchDogCount	UNS32	WatchDog Time-Out
.EcatMasterIsLockedCount	UNS32	Master Is Locked
.EcatMasterDCIntEstabCount	UNS32	Internal DC synchronisation is established
.EcatMasterDCExtEstabCount	UNS32	External DC synchronisation is established
.EcatMasterDCPropDelCount	UNS32	DC Propagation delay initialized
EcatSlaveCountDgn.	Struc	Čítače změn stavu Slave
.EcatSlaveOffLineCount[xx]	Pole UNS32 (0,1,2,3,…)	Slave doesn't respond to commands
.EcatSlaveErrStateCount[xx]	Pole UNS32 (0,1,2,3,…)	Slave's state is different as the set one
.EcatSlaveNotConfCount[xx]	Pole UNS32 (0,1,2,3,…)	Slave is not configured
.EcatSlaveWrongConfCount[xx]	Pole UNS32 (0,1,2,3,…)	Slave's configuration doesn't match the configuration
.EcatSlaveInitCmdErrCount[xx]	Pole UNS32 (0,1,2,3,…)	Error while Init Cmd
.EcatSlaveMailboxErrCount[xx]	Pole UNS32 (0,1,2,3,…)	Error while Mailbox Init Cmd
EcatStatistic.	Struc	Statistické informace pro EtherCAT
.EcatFramesPerSecond	UNS16	Frames rate (frames per second)
.EcatRxPackets	UNS32	Number of received frames
.EcatTxPackets	UNS32	Number of transmitted frames
.EcatRxBytes	UNS32	Number of received bytes

	.EcatTxBytes	UNS32	Number of transmitted bytes
	.EcatRxErrors	UNS32	Number of wrongly received frames
	.EcatTxErrors	UNS32	Number of wrongly transmitted frames
	.EcatRxDropped	UNS32	Number of dropped received frames
	.EcatTxDropped	UNS32	Number of dropped transmitted frames
	.EcatMulticast	UNS32	Number of multicast frames
	.EcatCollisions	UNS32	Number of collisions
	.EcatAvgCycleTime	UNS32	Average observed cycle time
	.EcatAvgCycleJitter	UNS32	Average observed cycle jitter
	.EcatMinCycleTime	UNS32	Minimal observed cycle time
	.EcatMaxCycleTime	UNS32	Maximal observed cycle time
	.EcatAvgSubCycleTime	UNS32	Average observed subcycle time
	.EcatAvgSubCycleJitter	UNS32	Average observed subcycle jitter
	.EcatMinSubCycleTime	UNS32	Minimal observed subcycle time
	.EcatMaxSubCycleTime	UNS32	Maximal observed subcycle time
	.EcatSendErrors	UNS32	Number of send-errors
	.EcatReceiveErrors	UNS32	Number of receive-errors
	.EcatWrongWC	UNS32	Number of received frames with wrong working counter
	.EcatParseErrors	UNS32	Number of parse errors
	.EcatCPULoad	UNS32	CPU load (percents multiplied by 10)
	.EcatBusLoad	UNS32	Bus load (Bytes per second)
	.EcatRespondTime	UNS32	Slave response time

18.2.4 Zobrazování a zápis do sdílených proměnných

Uvedeme příklad pro zobrazení:

1. systémová sdílená proměnná `ICounter[1] [REAL mm]` (zobrazí `DIFCIT_Y`)
2. PLC sdílená proměnná pro výstup `SHARE_DWORD (DWORD)`
3. PLC sdílený bit pro výstup `PRESS_ON (BIT)`

Příklad pro zápis:

1. PLC sdílená proměnná pro vstup (viz. DEF_IN) `BunDiag1 (REAL)`
2. PLC sdílená proměnná pro vstup (viz. DEF_IN) `BitIn1 (BIT)`

Části HTML souboru pro PLC diagnostiku „`PlcDiagnostics.html`". Jsou použity kaskádové styly. Na tomto místě nebudeme detailně rozepisovat umístění jednotlivých prvků.

Zobrazení hodnot se provede například v elementu `DIV` s atributem:
`CNCType="AsyncIndicatorPainter" a AIPType="PlcVar"`

Zápis hodnot se provede například v elementu `INPUT` s atributem:
`CNCType="PlcVarEdit" a Variable="Name: 'xxxx';"`

Část HTML kódu `BODY`:

```
<BODY scroll="no">
  <DIV ID="Background">

    <DIV class="WindowTitle"> <!-- nadpis -->
      Diagnostika PLC
    </DIV>

    <DIV id="Values">
      <DIV id="Diag0_Lbl" class="LabelMedium">DIFCIT_Y:</DIV>
      <DIV id="Diag0_Val" class="ValueMedium"
       CNCType="AsyncIndicatorPainter" AIPLibrary="StdPlugins"
       AIPType="RtmVar" AIPOptions="Channel: 0; Variable: 'ICounter[1]';
       NumberPrecision:3;">+000</DIV>

      <DIV id="Diag1_Lbl" class="LabelMedium">PLC_DWORD:</DIV>
      <DIV id="Diag1_Val" class="ValueMedium"
       CNCType="AsyncIndicatorPainter" AIPLibrary="StdPlugins"
       AIPType="PlcVar" AIPOptions="Channel: 0; Variable: 'SHARE_DWORD';
       NumberPrecision:0;">+000</DIV>

      <DIV id="Diag2_Lbl" class="LabelMedium">TLAK:</DIV>
      <DIV id="Diag2_Val" class="ValueMedium"
       CNCType="AsyncIndicatorPainter" AIPLibrary="StdPlugins"
       AIPType="PlcVar" AIPOptions="Channel: 0; Variable: 'PRESS_ON';
       NumberPrecision:0;">+000</DIV>

      <DIV id="Diag3_Lbl" class="LabelMedium">Zadat BunDiag1:</DIV>
      <INPUT id="Diag3_Val" class="EditMedium"  type="text"
       CNCType="PlcVarEdit" Variable="Name: 'PLC.Input.BunDiag1';">


      <DIV id="Diag4_Lbl" class="LabelMedium">Zadat BitIn1:</DIV>
      <INPUT id="Diag4_Val" class="EditMedium"  type="text"
       CNCType="PlcVarEdit" Variable="Name: 'PLC.Input.BitIn1';">
```

```
    </DIV>
  </DIV>
</BODY>
```

Vybrané prvky "AsyncIndicatorPainter" nabízené knihovnou StdPlugins pro sdílení:

AIPLibrary="StdPlugins" **CNCType="AsyncIndicatorPainter"**				
AIPType="RtmVar"			**Zobrazení systémové (RTM) sdílené proměnné**	
	AIPOptions		Vlastnosti prvku	
		Channel	0,1,..	číslo suportu (0,1,2..)
		Variable	'text'	textový řetězec se jménem sdílení
AIPType="PlcVar"			**Zobrazení sdílené proměnné PLC**	
	AIPOptions		Vlastnosti prvku	
		Channel	0,1,..	číslo suportu (0,1,2..)
		Variable	'text'	textový řetězec se jménem sdílení
AIPType="PlcVarText"			**Zobrazení sdílené textové proměnné PLC**	
	AIPOptions		Vlastnosti prvku	
		Channel	0,1,..	číslo suportu (0,1,2..)
		Variable	'text'	textový řetězec se jménem sdílení

18.3 Sdílená paměť

18.3.1 Instrukce pro práci se sdílenou pamětí

Sdílená paměť „SA" (Shared Area) je úsek paměti sdílené mezi PLC a primárním procesorem. V současné verzi jsou k dispozici 2 sdílené oblasti, každá o velikosti 32000 bajtů. Pro zabezpečení synchronizace přenosu dat je pro každou oblast definován vlastní mutex. Sdílená oblast je společná pro všechny kanály systému, kdy běží najednou více modulů reálného času (více suportové řízení).

PLC program má k dispozici pro přístup ke sdílené oblasti dvě nové instrukce **SA_READ** a **SA_WRITE**.

instrukce	**SA_READ** **SA_WRITE**

```
funkce      SA_READ     čtení dat ze sdílené oblasti PLC paměti
            SA_WRITE    zápis dat do sdílené oblasti PLC paměti


syntax      SA_READ (SA_WRITE)      SAIdx, Offset, Size, Poin
            SA_READ (SA_WRITE)      SAIdx, Offset, Size, Val


1.parametr  „SAIdx"           index oblasti SA
2.parametr  „Offset"          offset dat od začátku oblasti
3.parametr  „Size"            velkost dat
4.parametr  „Poin, Val"       pointer nebo název proměnné
```

Instrukce **SA_READ** slouží pro čtení dat ze sdílené oblasti PLC. V případě úspěchu vrací RLO=1, jinak RLO=0. Aby čtení dat proběhlo úspěšně, musí zadaná sdílená oblast existovat a všechna požadovaná data musí ležet uvnitř této oblasti (tj. „Offset" musí být větší nebo rovno nule a „Offset" + „Size" musí být menší než velikost dané oblasti). Funkce zajišťuje synchronizaci přístupu k dané paměťové oblasti, tzn. v průběhu jejího provádění nemůže do dané oblasti současně přistupovat nikdo jiný.

Instrukce **SA_WRITE** slouží pro zápis dat do sdílené oblasti PLC. V případě úspěchu vrací RLO=1, jinak RLO=0. Aby zápis dat proběhl úspěšně, musí zadaná sdílená oblast existovat a data, která se mají zapsat se musí do této oblasti vejít V případě úspěchu vrací RLO=1, jinak RLO=0. Aby zápis dat proběhl úspěšně, musí zadaná sdílená oblast existovat a všechna zapisovaná data musí ležet uvnitř této oblasti. Funkce zajišťuje synchronizaci přístupu k dané paměťové oblasti, tzn. v průběhu jejího provádění nemůže do dané oblasti současně přistupovat nikdo jiný.

Návratové hodnoty instrukcí:

Instrukce se musí používat v mechanizmech a jsou typu „EX".
Vrácené datové hodnoty z tabulky se zapisují do řetězce na který ukazuje parametr **„Poin",** nebo přímo do datové proměnné **„Val"**.
Všechny instrukce se mohou volat průchodově a mají návratové hodnoty:

```
RLO=0,      DR=0  ....... stav čekání na dokončení operace
RLO=1,      DR=0  ....... operace dokončena bez chyb
RLO=0,      DR<>0 ....... operace dokončena, ale při výkonu vznikla chyba
```

Instrukce v případě úspěchu vrací RLO nastaveno na 1. Pokud instrukce nastaví RLO na hodnotu 0 (a současně je nulový datový DR registr), je v daném okamžiku přístup do sdílené oblasti zakázán a přenos je potřeba opakovat například v příštím cyklu PLC programu.

Pokud instrukce skončí s chybou (například offset s délkou přenášených dat je větší něž délka sdílené oblasti) instrukce vrátí registr RLO nastaven na hodnotu 0 a nenulový datový DR registr.

Popis parametrů instrukcí:

parametr	název	význam	typ
1.	**SAIdx**	číslo sdílené oblasti (od nuly)	word
2.	**Offset**	Ofset dat, kde se bude zapisovat, od začátku sdílené oblasti	word
3.	**Size**	Délka dat, které se budou kopírovat	word
4.	**Poin**	Ukazatel na buffer , z kterého se zkopírují požadovaná data - náveští u řetězce definovaného instrukcí "str". Parametr může mít zadán offset v řetězci (+xx).	pointr
	Val	Název datové proměnné. - typ BYTE, WORD, DWRD,..	data

18.3.2 Orientace ve sílené paměti

Sdílenou paměť využívá pro komunikaci PLC program, systémová část, uživatelské obrazovky a pod. Offsety pro orientaci ve sdílené paměti se proto nezadávají přímo hodnotou, ale symbolicky a jsou společné pro všechny části, které sdílenou oblast potřebují. Pro tvorbu symbolických offsetů se proto používají PLC konstanty (viz „PLC konfigurace a konstanty“).

PLC konstanty pro definici offsetů ve sdílené paměti jsou zadány v souboru „`SAVars.PlcConstans`“. Všechny PLC konstanty se zadávají v XML tvaru pomocí elementu `PlcConstants`.

Příklad:

Příklad pro konstanty offsetů ve sdílené paměti:

```
<PlcConstants>

  <Constant ID="OFFS_R_FEED"  Value="SA0REAL"></Constant>  <!-- Feed -->
  <Constant ID="OFFS_I_STATE" Value="SA0INT"></Constant>   <!-- Stav -->

</PlcConstants>
```

Načtení PLC konstant pro offsety v modulu inicializace:

```
OFFS_R_FEED:      DS    2     ;jsem se načte offset pro FEED
OFFS_I_STATE:     DS    2     ;jsem se načte offset pro STATE
R_FEED:           DS    8     ;buňka pro načtení reálné hodnoty FEED
STATE:            DS    4     ;buňka pro zápis DWORD hodnoty STATE

PROC_BEGIN Inicializace
      CONST_GET   OFFS_R_FEED,'OFFS_R_FEED'     ;načte offset pro FEED
      LA          FL_OK
      WR          FL_OK
      CONST_GET   OFFS_I_STATE,'OFFS_I_STATE'   ;načte offset pro STATE
      LA          FL_OK
      WR          FL_OK

      LDR         -FL_OK
```

```
      ESET1       2           ; PLC0002: Načítání PLC konstant se nepodařilo
PROC_END Inicializace
```

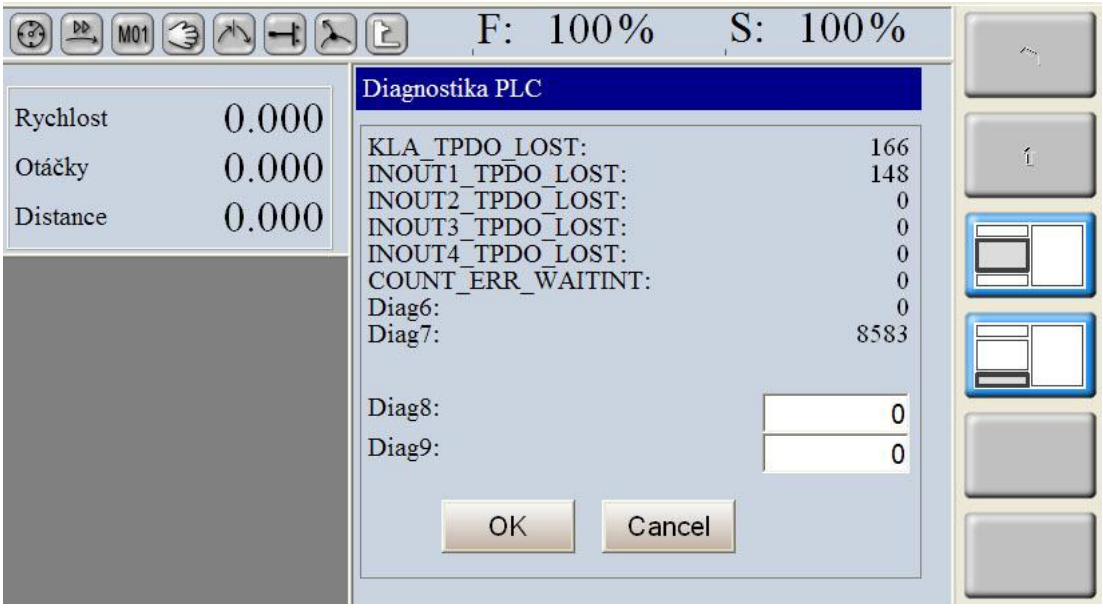
Obsluha čtení a zápisu se sdílené oblasti pomocí instrukci SA_READ a SA_WRITE.

```
MECH_BEGIN  M_SA_RW
CYK:  EX
      SA_READ     0,OFFS_R_FEED,8,R_FEED  ;načte reální hodnotu FEED
      EX0
      SA_WRITE    0,OFFS_I_STATE,4,STATE  ;zapíše DWORD hodnotu STATE
      EX0
      JUM         CYK
MECH_END    M_SA_RW
```

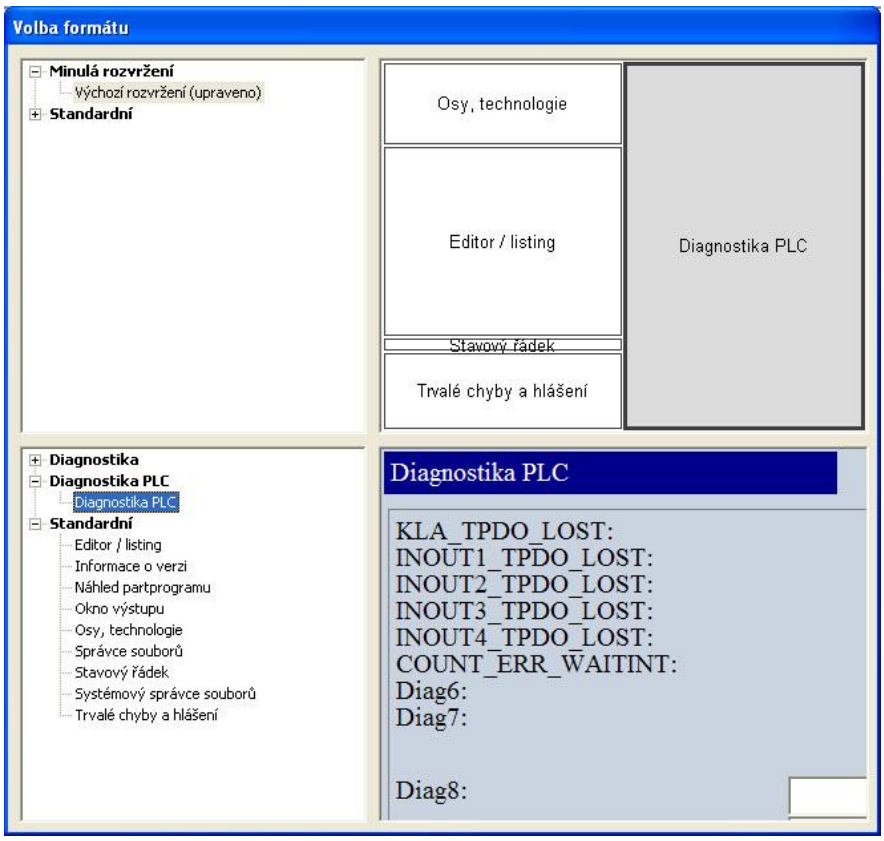
18.3.3 Využití sdílené paměti pro tvorbu okna

Uvedeme příklad použití sdílené paměti pro tvorbu diagnostického okna PLC. Okno má obsahovat osm výstupních a dvě vstupní pole. V systému bude začleněno jako normální okno, které se vyvolá prostřednictvím volby indikace.

Upozornění: Příklad slouží jen jako ukázka práce se sdílenou pamětí. V praxi se sdílená paměť na tyto účely nepoužívá, protože mnohem snadněji dosáhneme stejného výsledku využitím přímého sdílení (instrukce SHARE_VAR,…) .



Volba diagnostického okna PLC prostřednictvím „volby indikace“:



Definice dat v PLC programu. Pro zjednodušení budeme uvažovat 2 vstupní a 2 výstupní pole:

```
;Offsety pro diagnostiku
OFFS_DIAG0:             DS    2           ;PLC diagnostika
OFFS_DIAG1:             DS    2
OFFS_DIAG2:             DS    2
OFFS_DIAG3:             DS    2

;Buňky pro PLC diagnostiku
Diag0:                  DS    4           ;výstup do okna
Diag1:                  DS    4
Diag2:                  DS    4           ;vstup z okna
Diag3:                  DS    4
```

Načtení offsetů pro sdílenou paměť v inicializaci:

```
PROC_BEGIN INICIALIZACE_CNF
            FL          1,CNFOK

            CONST_GET   OFFS_DIAG0,'OFFS_DIAG0'
            LA          CNFOK
            WR          CNFOK
            CONST_GET   OFFS_DIAG1,'OFFS_DIAG1'
            LA          CNFOK
```

```
            WR          CNFOK
            CONST_GET   OFFS_DIAG2,'OFFS_DIAG2'
            LA          CNFOK
            WR          CNFOK
            CONST_GET   OFFS_DIAG3,'OFFS_DIAG3'
            LA          CNFOK
            WR          CNFOK

            LDR         -CNFOK
            ESET1       ERR_KONSTANTY     ;Chyba načtení PLC konstant
PROC_END INICIALIZACE_CNF
```

Mechanizmus pro kontinuální obsluhu PLC diagnostiky. Zapisují a čtou se data z okna diagnostiky:

```
            FL          1, M_PLCDGN       ;mechanizmus trvale běží

MECH_BEGIN  M_PLCDGN
M_CYKL:     LOD         BYTE.KLA_TPDO_LOST      ;plnění buněk výstupu
            STO         Diag0
            LOD         BYTE.INOUT_TPDO_LOST
            STO         Diag1

            SA_WRITE    0,OFFS_DIAG0,4,Diag0    ;Diagnostika - výstup
            EX0
            EX
            SA_WRITE    0,OFFS_DIAG1,4,Diag1
            EX0
            EX
            SA_READ     0,OFFS_DIAG2,4,Diag2    ;Diagnostika - vstup
            EX0
            EX
            SA_READ     0,OFFS_DIAG3,4,Diag3
            EX0
            EX
            JUM         M_SCYKL
MECH_END    M_PLCDGN
```

PLC konstanty pro definici offsetů ve sdílené paměti jsou zadány v souboru „SAVars.PlcConstans“.

```
<PlcConstants>
  <!-- Diagnostics -->
  <Constant ID="OFFS_DIAG0" Value="SA0INT"></Constant> <!-- Diagnostika -->
  <Constant ID="OFFS_DIAG1" Value="SA0INT"></Constant>
  <Constant ID="OFFS_DIAG2" Value="SA0INT"></Constant>
  <Constant ID="OFFS_DIAG3" Value="SA0INT"></Constant>
</PlcConstants>
```

Části HTML souboru pro PLC diagnostiku „PlcDiagnostics.html“. Jsou použity kaskádové styly. Na tomto místě nebudeme detailně rozepisovat umístění jednotlivých prvků.

Zobrazení hodnot se provede například v elementu DIV s atributem CNCType=“AsyncIndicatorPainter“ a vstup hodnot se provede pomocí elementu INPUT s atributem name=“PlcSAValueEdit“ (viz dále).

Část HTML kódu BODY:

```
<BODY scroll="no">
  <DIV ID="Background">

    <DIV class="WindowTitle"> <!-- nadpis -->
      Diagnostika PLC
    </DIV>

    <DIV id="Values">
      <DIV id="Diag0_Lbl" class="LabelMedium">KLA_TPDO_LOST:</DIV>
      <DIV id="Diag0_Val" class="ValueMedium"
       CNCType="AsyncIndicatorPainter" AIPLibrary="StdPlugins"
       AIPType="PlcSAValue" AIPOptions="Channel: 0; Area: 0;
       Offset: OFFS_DIAG0; DataType: UByte; NumberPrecision:0;">+000</DIV>

      <DIV id="Diag1_Lbl" class="LabelMedium">INOUT1_TPDO_LOST:</DIV>
      <DIV id="Diag1_Val" class="ValueMedium"
       CNCType="AsyncIndicatorPainter" AIPLibrary="StdPlugins"
       AIPType="PlcSAValue" AIPOptions="Channel: 0; Area: 0;
       Offset: OFFS_DIAG1; DataType: UByte; NumberPrecision:0;">+000</DIV>

      <DIV id="Diag8_Lbl" class="LabelMedium">Diag8:</DIV>
      <INPUT id="Diag8_Val" type="text" class="EditMedium"
       name="PlcSAValueEdit" Area="0" Offset="OFFS_DIAG2"DataType="UDWord">

      <DIV id="Diag9_Lbl" class="LabelMedium">Diag9:</DIV>
      <INPUT id="Diag9_Val" type="text" class="EditMedium"
       name="PlcSAValueEdit" Area="0" Offset="OFFS_DIAG3"DataType="UDWord">


      <BUTTON id="OK" class="Button">OK</BUTTON>
      <BUTTON id="Cancel" class="Button">Cancel</BUTTON>
    </DIV>
  </DIV>
</BODY>
```

Kopírování a registrace pro tvorbu SETUPu PLC v souboru „PLC.NSI“:

Část pro instalaci souborů:

```
Function InstallPlcFiles

File "/oname=Html\PlcDiagnostics.html" "..\Html\PlcDiagnostics.html"
```

Část pro odinstalování:

```
Function un.InstallPlcFiles

Delete "$INSTDIR\Html\PlcDiagnostics.html"
```

Část pro registraci dialogů:

```
Function InstallPlcConfig

WriteRegStr HKLM ~
"Software\MEFI\WinCNC\Machine\UserInterface\Dialogs\PlcDiagnostics" ~
"Library" "StdPlugins"
WriteRegStr HKLM ~
"Software\MEFI\WinCNC\Machine\UserInterface\Dialogs\PlcDiagnostics" ~
"Type" "PlcSAEditor"
WriteRegStr HKLM ~
"Software\MEFI\WinCNC\Machine\UserInterface\Dialogs\PlcDiagnostics" ~
"HtmlFile" "PlcDiagnostics.html"
```

Část pro registraci oken:

```
WriteRegStr HKLM ~
"Software\MEFI\WinCNC\Machine\UserInterface\Windows\PlcDiagnostics" ~
"Library" "StdPlugins"
WriteRegStr HKLM ~
"Software\MEFI\WinCNC\Machine\UserInterface\Windows\PlcDiagnostics" ~
"Type" "PlcSAEditor"
WriteRegStrHKLM ~
"Software\MEFI\WinCNC\Machine\UserInterface\Windows\PlcDiagnostics" ~
"HtmlFile" "PlcDiagnostics.html"
WriteRegStr HKLM ~
"Software\MEFI\WinCNC\Machine\UserInterface\Windows\PlcDiagnostics" ~
"Category" "Diagnostika PLC"
WriteRegStr HKLM ~
"Software\MEFI\WinCNC\Machine\UserInterface\Windows\PlcDiagnostics" ~
"Name" "Diagnostika PLC"
```

(znak „~“ znamená pokračování řádku – ve skutečnosti řádek nesmí být rozdělen)

18.3.4 Zobrazování dat ze sdílené paměti

Zobrazení hodnot ze sdílené paměti se provede v HTML kódu pomocí atributu `CNCType="AsyncIndicatorPainter"`. Dále uvedeme možnosti použití tohoto atributu, které souvisí se sdílenou pamětí.

Některé vybrané Prvky `"AsyncIndicatorPainter"` nabízené knihovnou `StdPlugins`.

AIPLibrary="StdPlugins" **CNCType="AsyncIndicatorPainter"**				
AIPType="PlcSAValue"			**Zobrazení hodnot různou formou**	
	AIPOptions		Vlastnosti prvku	
		Channel	0,1,..	číslo suportu (0,1,2..)
		Area	0,1,..	číslo sdílené oblasti (0,1)
		Offset	xxx	offset prvku ve sdílené paměti (v bajtech od začátku)
		DataMask	xxx	maska
		DataType	BIT	bitová proměnná
			UBYTE	celočíselná hodnota BYTE bez znaménka
			SBYTE	celočíselná hodnota BYTE se znaménkem
			UWord	celočíselná hodnota WORD bez znaménka
			SWord	celočíselná hodnota WORD se znaménkem
			UDWord	celočíselná hodnota DWORD bez znaménka
			SDWord	celočíselná hodnota DWORD se znaménkem
			Real	reálná hodnota
		GraphType	Number	číselná hodnota
			Time	formát času
			Image	obrázek
			Text	text
			HBar	horizontální indikátor
			VBar	vertikální indikátor
			Meter120,..	potenciometr 120,...
		NumberWidth	xx	
		NumberPrecision	0,1,2,..	
		ImgN	N=0,1,2,..	obrázek řazený podle hodnoty proměnné (N=0..31)
		TextN	N=0,1,2,..	text řazený podle hodnoty proměnné (N=0..31)
		Výčet dalších vlastností: `MinVal, MaxVal, NumberSign, PointerImg, FaceImg, ImgDefault, ImgWidth, ImgHeight, TextDefault`		
AIPType="PlcSAData"			**Výpis paměti (hexadecimálně)**	
	AIPOptions		Vlastnosti prvku	
		Channel	0,1,..	číslo suportu (0,1,2..)
		Area	0,1,..	číslo sdílené oblasti (0,1)
		Offset	xxx	offset prvku ve sdílené paměti (v bajtech od začátku oblasti)
		DataSize	xxx	počet
		GroupBy	BIT	bitová proměnná
			BYTE	proměnná BYTE
			WORD	proměnná WORD
			DWORD	proměnná DWORD

18.3.5 Využití sdílené paměti pro zobrazování obrázků

Uvedeme příklad použití sdílené paměti pro vykreslení obrázků na základě hodnoty v datové proměnné. Budeme řídit zobrazování vlastní signálky (LED diody) v horní části obrazovky (oblast TOP). Signálka má mít 3 stavy a bude se řídit z PLC pomocí buňky „MACHINE_STATE“:

MACHINE_STATE	stav	obrázky signálek	názvy souborů
0	not ready	 šedá	Machine0.png
1	ready	 zelená	Machine1.png
2	error	 červená	Machine2.png

Definice dat v PLC programu.

```
OFFS_MACHINE_STATE:     DS    2     ;offset pro stavovou buňku

MACHINE_STATE:          DS    4     ;stavová buňka (0, 1, 2)
```

Načtení offsetu pro sdílenou paměť v inicializaci:

```
PROC_BEGIN INICIALIZACE_CNF
            FL          1,CNFOK

            CONST_GET   OFFS_MACHINE_STATE,'OFFS_MACHINE_STATE'
            LA          CNFOK
            WR          CNFOK

            LDR         -CNFOK
            ESET1       ERR_KONSTANTY     ;Chyba načtení PLC konstant
PROC_END INICIALIZACE_CNF
```

Obsluha zápisu stavu do sdílené oblasti

```
            FL          1, M_SA_STATE     ;mechanizmus trvale běží

MECH_BEGIN  M_SA_STATE
CYK:        LOD         cnst.0            ;nastavení MACHINE_STATE
            STO         MACHINE_STATE
            LDR         MReady            ;Stroj ready ?
            LOD         cnst.1
            STO1        MACHINE_STATE
            LDR         MError            ;Error stroje ?
            LOD         cnst.2
            STO1        MACHINE_STATE
            EX
            SA_WRITE    0,OFFS_MACHINE_STATE,4,MACHINE_STATE
            EX0
            JUM         CYK
MECH_END    M_SA_STATE
```

Úprava souboru „TOP.HTML“ pro zavedení nové signálky. Soubor TOP.HTML překopírujeme z instalační části z adresáře HTML do projektu PLC. V části pro signálky přidáme řádek:

```
<TD width="36" height="32" CNCType="AsyncIndicatorPainter"
 AIPLibrary="StdPlugins" AIPType="PlcSAValue" AIPOptions="Channel: 0;
 Area: 0; Offset: OFFS_MACHINE_STATE; GraphType: Image; DataType: UDWord;
 Img0: Machine0.png; Img1: Machine1.png; Img2: Machine2.png; ">LED</TD>
```

Zavedení PLC konstanty pro offset ve sdílené paměti v souboru „SAVars.PlcConstans“.

```
<Constant ID="OFFS_MACHINE_STATE" Value="SA0INT"></Constant> <!-- stav -->
```

Upravíme SETUP pro PLC aby se soubor TOP.HTML kopíroval do adresáře „CNC Machine Files“

Část pro instalaci souborů:

```
Function InstallPlcFiles

File "/oname=Html\Top.html" "..\Html\Top.html"
```

Část pro odinstalování:

```
Function un.InstallPlcFiles

Delete "$INSTDIR\Html\Top.html"
```

16

16. PLC KONFIGURACE A KONSTANTY

16.1 Konfigurace pro PLC program

PLC program má k dispozici pro přístup ke své konfiguraci instrukce **CNF_GET_INT, CNF_GET_REAL, CNF_GET_STR** a **CNF_GET_BIN**. Konfigurace pro PLC program je uložena například v souboru Channelconfig, který má XML formát pod elementem **Plc**. Zápis parametrů konfigurace není povinný a pokud se příslušný parametr v konfiguračním souboru nevyskytuje, v parametru pro PLC se objeví defaultmí hodnota.
Konfigurace je pro PLC k dispozici už v době průchodu modulu MODULE_INIT, proto se doporučuje načíst celou konfiguraci v proceduře, která je zavolaná z modulu MODULE_INIT.

element **Plc**			**Konfigurace pro PLC**	
	element **PlcParam**		**Jeden parametr konfigurace pro PLC.**	
		atribut **ID**	**Identifikátor parametru**	
			Abcd	Textový řetězec, který slouží jako identifikátor parametru
		atribut **Type**	**Typ parametru konfigurace**	
			Int	Celočíselné a logické hodnoty
			Real	reálné hodnoty
			String	Textové řetězce
			Binary	Binární data
			Hodnota parametru	
			xx	Hodnota se zapisuje jako obsah elementu

Příklad:
Příklad zápisu parametrů PLC konfigurace v XML tvaru:

```
<Plc>
  <PlcParam ID="LIM_SWITCH" Type="Int">1</PlcParam> <!-- lim spínač -->
  <PlcParam ID="TIM_WATER"  Type="Int">4</PlcParam> <!-- prodlení vody  -->
</Plc>
```

Instrukce pro načtení konfigurace pro PLC:

instrukce	**CNF_GET_INT** **CNF_GET_REAL** **CNF_GET_STR** **CNF_GET_BIN** **CNF_GET_BIT**

funkce	**CNF_GET_INT**	**Načtení celočíselné hodnoty z konfigurace**
	CNF_GET_REAL	**Načtení reálné hodnoty z konfigurace**
	CNF_GET_STR	**Načtení textového řetězce z konfigurace**
	CNF_GET_BIN	**Načtení binárního řetězce z konfigurace**
	CNF_GET_BIT	**Načtení bitové hodnoty z konfigurace**

syntax	**CNF_GET_xx**	**poin1, poin2 [, poin3]**
	CNF_GET_xx	**val1, poin2 [, val3]**
	CNF_GET_xx	**poin1, 'TEXT2' [, immed3]**
	CNF_GET_xx	**poin1, 'TEXT2' [,'TEXT3']**

1.parametr	**„val1,poin1"**	**pointer nebo název cílové proměnné**
2.parametr	**„poin2,text"**	**pointer nebo textový řetězec identifikátoru**
3.parametr	**„val3,poin3,immed"**	**pointer, název nebo přímo defaultní hodnota**

Vrácené datové hodnoty z konfigurace se zapisují do řetězce na který ukazuje parametr **poin1,** nebo přímo do datové proměnné **val1** a **bit**.
Všechny instrukce mají návratové hodnoty:

RLO=1 ... operace dokončena bez chyb
RLO=0 ... operace dokončena, ale konfigurační parametr se nenašel. Instrukce vrátí defaultní hodnotu.

parametr	název	význam	typ
1.	**poin1**	Ukazatel na buffer, do kterého se zkopírují požadovaná data, - náveští u řetězce definovaného instrukcí "str". Parametr může mít zadán offset v řetězci (+xx).	pointer
	val1	název datové proměnné do které se zkopírují požadovaná data, - typ BYTE, WORD, DWRD,..	data
	bit1	název bitové proměnné, která se nastaví podle konfigurace	bit
2.	**poin2**	Ukazatel na buffer, kde je umístěný text s klíčovým slovem konfiguračního parametru - náveští u řetězce definovaného instrukcí "str".	pointer
	text2	Přímé zadání textu s klíčovým slovem konfiguračního parametru v apostrofech	řetězec
3.	**poin3**	Ukazatel na buffer pro defaultní hodnotu, z které se zkopírují data do parametru 1, pokud se nenajde požadované klíčové slovo v konfiguraci. - náveští u řetězce definovaného instrukcí "str". Parametr může mít zadán offset v řetězci (+xx).	pointer
	val3	Název datové proměnné pro defaultní hodnotu, z které se zkopírují data do parametru 1, pokud se nenajde požadované klíčové slovo v konfiguraci. - typ BYTE, WORD, DWRD,..	data
	text3	Přímé zadání defaultního textu, který se zkopíruje do parametru 1, pokud se nenajde požadované klíčové slovo v konfiguraci	řetězec
	immed	Přímé zadání defaultní hodnoty, číslo se naplní do parametru 1 Reálná hodnota se zadává s desetinnou tečkou	číselná hodnota

výčet možností syntaxe:

```
      CNF_GET_xxx       poin1, poin2
      CNF_GET_xxx       poin1, poin2,   poin3
      CNF_GET_xxx       poin1, poin2,   val3
      CNF_GET_xxx       poin1, poin2,   'TEXT3'
      CNF_GET_xxx       poin1, poin2,   immed

      CNF_GET_xxx       val1,  poin2
      CNF_GET_xxx       val1,  poin2,   poin3
      CNF_GET_xxx       val1,  poin2,   val3
      CNF_GET_xxx       val1,  poin2,   'TEXT3'
      CNF_GET_xxx       val1,  poin2,   immed

      CNF_GET_xxx       poin1, 'TEXT2'
      CNF_GET_xxx       poin1, 'TEXT2', poin3
      CNF_GET_xxx       poin1, 'TEXT2', val3
      CNF_GET_xxx       poin1, 'TEXT2', 'TEXT3'
      CNF_GET_xxx       poin1, 'TEXT2', immed

      CNF_GET_xxx       val1,  'TEXT2'
      CNF_GET_xxx       val1,  'TEXT2', poin3
      CNF_GET_xxx       val1,  'TEXT2', val3
      CNF_GET_xxx       val1,  'TEXT2', 'TEXT3'
      CNF_GET_xxx       val1,  'TEXT2', immed
```

Příklady:

; různé možnosti načtení konfigurace

```
CNF2:       STR   20,'DOBA_MAZANI'
TEXT1:      STR   20
BITX:       DFM   ,,,,bit_A,,,
dwTime:     DS    4
rBun1:      DS    8     ;pro reálné hodnoty
rBunDef:    DS    8


      CNF_GET_INT       dwTime, CNF2, 125
      CNF_GET_INT       dwTime, 'DOBA_MAZANI', 125
      CNF_GET_STR       TEXT1, 'TextMazani','Probiha mazani'

      CNF_GET_REAL      rBun1, 'Zesileni', cnst.0.2
                                        ;reálná defaultní hodnota
```

; načte konfiguraci podle předchozího příkladu

```
      CNF_GET_BIT       bit_A, 'LIM_SWITCH', 0  ;načte bitovou proměnnou
      CNF_GET_INT       dwTime, 'TIM_WATER', 6  ;načte prodlení pro vodu
```

16.2 Konstanty pro PLC program

PLC konstanty slouží pro symbolické definování konstant, které mají být k dispozici jak v PLC programu. tak v „Panelu“ systému. Konstanty lze například využít na definici příkazů pro PLC nebo na definici symbolických offsetů pro orientaci ve sdílené paměti mezi PLC programem a Panelem systému (instrukce `SA_READ` a `SA_WRITE`).

PLC konstanty pro definici offsetů ve sdílené paměti jsou zadány v souboru „`SAVars.PlcConstans`“.
PLC konstanty pro definici příkazů pro PLC jsou zadány v souboru „`Commands.PlcConstans`“.

Všechny PLC konstanty se zadávají v XML tvaru pomocí elementu `PlcConstants`.

element **PlcConstants**			**Konstanty pro PLC**	
	element **Constant**		**Jedna konstanta**	
		atribut **ID**	**Identifikátor parametru**	
			Abcd	Textový řetězec, který slouží jako identifikátor konstanty
		atribut **Value**	**Hodnota konstanty**	
			xx	Přímo číselná hodnota PLC konstanty
			SA0INT	Konstanta je offset na DWORD ve sdílené paměti
			SA0REAL	Konstanta je offset na REAL ve sdílené paměti
			PLCCOMMAND	Konstanta je příkaz pro PLC

Pokud atribut „`Value`“ neobsahuje přímo číselnou hodnotu konstanty, ale některou z identifikátorů „`SA0INT,SA0REAL a PLCCOMMAND`“ dojde k automatickému číslování konstant s ohledem na typ. Tato metoda se doporučuje, protože nemůže docházek k překrytí číselných hodnot konstant.

Příklad:

Příklad pro konstanty offsetů ve sdílené paměti:

```
<PlcConstants>

  <Constant ID="OFFS_R_FEED"  Value="SA0REAL"></Constant>  <!-- Feed -->
  <Constant ID="OFFS_I_STATE" Value="SA0INT"></Constant>   <!-- Stav -->

</PlcConstants>
```

Příklad pro konstanty příkazů pro PLC:
(příklad s přímým zadáním hodnot konstant, lepší je použít `Value="PLCCOMMAND"`):

```
<PlcConstants>

  <Constant ID="PUMP_START" Value="12"></Constant> <!-- Zapne pumpu  -->
  <Constant ID="JET1_UP" Value="13"></Constant>    <!-- JET1 nahoru  -->

</PlcConstants>
```

PLC konstanty zabezpečí provázanost mezi PLC programem, systémem a dialogovými okny.

Různé příklady použití PLC konstant mimo PLC program:
PLC konstantu pro příkaz budeme označovat: "Cmd"
PLC konstantu pro offset ve sdílené paměti: "Offs"

typ souboru	atribut s PLC konstantou	popis
*.KbdConfig	PlcCommand="Cmd"	Tlačítko pro PLC program definované v technologické části tlačítek panelu jako atribut elementu KeyConfig
*.HTML	PlcCommand="Cmd"	Softwarové tlačítko pro PLC program definované v dialogovém okně jako atribut v elementu BUTTON
*.SoftMenu	PlcCommand="Cmd"	Softwarové tlačítko pro PLC program definované v softwarovém menu jako atribut v elementu SoftMenuItem
*.HTML	Offset="Offs"	Vstupní hodnota z dialogového okna, která se zapíše na zadaný offset do sdílené paměti, definovaný jako atribut elementu INPUT
*.HTML	Offset="Offs"	Zobrazovaná hodnota v okně, která se zobrazuje ze zadaného offsetu ze sdílené paměti, definovaný např. jako atribut elementu DIV s atributem: CNCType = "AsyncIndicatorPainter"
*.HTML	Offset="Offs"	Zobrazení obrázků podle hodnoty proměnné ve sdílené paměti, definovaný jako atribut elementu IMG s atributem: CNCType = "AsyncIndicatorPainter"

Všechny soubory typu „PlcConstans" musí být zařazeny v projektu PLC a v PLC SETUPu musí být zabezpečeno, že se všechny překopírují z projektu do adresáře PLC v prostředí „CNC Machine Files". Proto v souboru „PLC.NSI" musí být:

Část pro instalaci souborů:

```
Function InstallPlcFiles

File "/oname=Plc\Commands.PlcConstants" "..\Plc\Commands.PlcConstants"
File "/oname=Plc\SAVars.PlcConstants"   "..\Plc\SAVars.PlcConstants"
```

Část pro odinstalování:

```
Function un.InstallPlcFiles

Delete "$INSTDIR\Plc\Commands.PlcConstants"
Delete "$INSTDIR\Plc\SAVars.PlcConstants"
```

instrukce	**CONST_GET**

```
funkce      CONST_GET     Načtení PLC konstanty

syntax      CONST_GET     val1, poin2
            CONST_GET     val1, 'TEXT2'

1.parametr  „val1"        název cílové proměnné kam se nečte konstanta
2.parametr  „poin2,text"  pointer nebo textový řetězec identifikátoru
```

Vrácená datová hodnota PLC konstanty se naplní do DR registru (DWORD). Návratové hodnoty jsou:

```
RLO=0, DR=0 ....... PLC konstanta se nenašla
RLO=1, DR   ....... operace dokončena v pořádku
```

Konstanty pro PLC jsou k dispozici už v době průchodu modulu MODULE_INIT, proto se doporučuje načíst všechny konstanty v proceduře, která je zavolaná z modulu MODULE_INIT.

parametr	název	význam	typ
1.	**val1**	název datové proměnné do které se zkopírují požadovaná data, - typ BYTE, WORD, DWRD	data
2.	**poin2**	Ukazatel na buffer, kde je umístěný text ID PLC konstanty - náveští u řetězce definovaného instrukcí "str".	pointer
	text	Přímé zadání textu ID PLC konstanty v apostrofech	řetězec

Konstanty příkazů se v PLC programu zpracovávají pomocí událostní procedury **_ON_CMD**.
(viz kapitolu „Základní instrukce“).

Procedura se zavolá automaticky při stisku tlačítka nebo softwarového menu, které mají v konfiguraci nastaven "PLCCommand". Procedura v PLC programu je nepovinná. Umístění procedury může být v libovolném souboru. Překladač TECHNOL zkoumá existenci takovéto procedury a v případě, že existuje, spustí ji automaticky při stisku tlačítka.

Vstupní parametry procedury jsou:

```
RLO = 1 ...... tlačítko stisknuto
RLO = 0 ...... tlačítko puštěno
DR (DWORD) ... kód z konfigurace "PLCCommand"
               kódem je PLC konstanta načtená pomocí CONST_GET
```

Konstanty symbolických offsetů pro orientaci ve sdílené paměti se použijí v parametru instrukce **SA_READ** a **SA_WRITE**.

Příklad:

Příklad načtení konstant z předchozího příkladu:

```
OFFS_R_FEED:      DS    2     ;jsem se načte offset pro FEED
OFFS_I_STATE:     DS    2     ;jsem se načte offset pro STATE
CMD_PUMP_START:   DS    4     ;buňka pro kód příkazu PUMP_START
CMD_JET1_UP:      DS    4     ;buňka pro kód příkazu JET1_UP
R_FEED:           DS    8     ;buňka pro načtení reálné hodnoty FEED
STATE:            DS    4     ;buňka pro načtení DWORD hodnoty STATE

PROC_BEGIN Inicializace
      CONST_GET   OFFS_R_FEED,'OFFS_R_FEED'    ;načte offset pro FEED
      LA          FL_OK
      WR          FL_OK
      CONST_GET   OFFS_I_STATE,'OFFS_I_STATE'  ;načte offset pro STATE
      LA          FL_OK
      WR          FL_OK
      CONST_GET   CMD_PUMP_START,'PUMP_START'  ;načte kód pro PUMP_START
      LA          FL_OK
      WR          FL_OK
      CONST_GET   CMD_JET1_UP,'JET1_UP'        ;načte kód pro JET1_UP
      LA          FL_OK
      WR          FL_OK

      LDR         -FL_OK
      ESET1       2           ; PLC0002: Načítání PLC konstant se nepodařilo
PROC_END Inicializace
```

Příklad pro rozkódování příkazů v událostní proceduře `_ON_CMD`.

```
;Událostní procedura pro příkazy PLC
PROC_BEGIN  _ON_CMD
      JL0         TL_PUSTENO              ;tlačítko puštěno ?
      EQ          CMD_PUMP_START          ;je start pumpy ?
      FL1         1,M_PUMP_START          ;mechanizmus start pumpy
      EQ          CMD_JET1_UP             ;je příkaz JET1 nahoru ?
      FL1         1,JET1_UP_O             ;výstup JET1 nahoru - začátek
      JUM         TL_END

TL_PUSTENO:                               ;pro konec příkazu
      EQ          CMD_JET1_UP             ;je příkaz JET1 nahoru ?
      FL1         0,JET1_UP_O             ;výstup JET1 nahoru - konec
TL_END:
PROC_END    _ON_CMD
```

Příklad pro čtení se sdílené oblasti pomocí instrukci `SA_READ` a `SA_WRITE`.

```
MECH_BEGIN  M_SA_READ
      EX
      SA_READ     0,OFFS_R_FEED,8,R_FEED  ;načte reální hodnotu FEED
      EX0
      SA_READ     0,OFFS_I_STATE,4,STATE  ;načte DWORD hodnotu STATE
      EX0
MECH_END    M_SA_READ
```

11

11. OVLÁDACÍ PANELY

11.1 Základní popis

V dalším textu se budeme zabývat standardními periferiemi systému, které jsou připojeny prostřednictvím sběrnice CAN-BUS nebo EtherCAT a slouží pro ovládání.

typ	max. počet	Popis
Panel Kla50 „PanelKla50“	4	Připojení na sběrnici CAN-BUS, bez galvanického oddělení. Obsahuje maximálně 112 (8x14) tlačítek zapojených do matice a dvě inkrementální kolečka (například pro ovládání procenta rychlosti). Řízení panelu vykonává jednotka KLA50.
Externí panel EPU		Připojení na sběrnici CAN-BUS s galvanickým oddělením. Obsahuje maximálně 48 tlačítek zapojených do matice. Řízení panelu vykonává jednotka KLA56.
Joystick Kla50 „JoystickKla50“		Připojení na sběrnici CAN-BUS. Obsahuje joystick a 18 tlačítek. Řízení vykonává jednotka KLA50
Panel Kla60 „PanelKla60“		Připojení na sběrnici EtherCAT. Obsahuje maximálně 112 (8x14) tlačítek zapojených do matice a dvě inkrementální kolečka (například pro ovládání procenta rychlosti). Řízení panelu vykonává jednotka KLA60.
Točítko Kla50 „KnobKla50“	2	Připojení na sběrnici CAN-BUS. Obsahuje maximálně 9 nezávislých tlačítek, grafický displej 64x128 bodů a inkrementální kolečko. Řízení vykonává jednotka KLA50
Virtuální klávesnice „Virtual“	1	Virtuální klávesnice. Nejedná se o fyzickou klávesnici nebo panel. Virtuální klávesnice má definováno 256 virtuálních tlačítek (s 256 `SCAN` kódy), ke kterým se možno připojit a ovládat je jako kdyby to byly standardní tlačítka. Virtuální tlačítka svými stisky tak mohou vykonávat příkazy podle konfigurace, která je zadána v souboru typu „`KbdConfig`“. Virtuální klávesnice v podstatě umožňuje PLC programu pomocí bitu spouštět příkazy definovány pomocí atributů: `Command`, `RtmCommand`, `PlcCommand`, `<Dialog>` (viz dále a návod „Logické vstupy a výstupy modulů“).

11.2 Konfigurace ovládacích panelů

Typy ovládacích panelů jsou zapojeny na sběrnici CAN-BUS: „CAN2“ nebo EtherCAT. Počet připojených periferií se konfiguruje v souboru typu „`ChannelConfig`“.

Nastavení **NODE-ID** pro systémové CAN-BUS periferie:

skupina (group)		**Pořadové číslo jednotky ve skupině (board)**						
		1	**2**	**3**	**4**	**5**	**6**	**...**
INOUT08	**1**	21h	22h	23h	24h	25h	26h	20h+board
Panel, KLA50, EPU	**2**	41h	42h	43h	44h			40h+board
Točítko	**3**	45h	46h					44h+board

Systémové panely připojené na sběrnici EtherCAT musí mít totožný název v konfiguračním souboru „`ChannelConfig`“ a v projektu EtherCAT studia.

Pro systémové CAN-BUS periferie platí:

element **`CANChannel`**		**Nastavení kanálu CAN-BUSu**	
	atribut **`No`**	**číslo kanálu CAN-BUS**	
		`1`	pro periferie CAN-BUS
		`xx`	
	atribut **`Active`**	**je CAN kanál aktivní ?**	
		`0`	CAN kanál neaktivní (default)
		`1`	CAN kanál aktivní
	atribut **`DeviceType`**	**Podporované typy CAN zařízení**	
		`MCAN01`	Systémový CAN kanál je připojen na jednotku MACAN01
		`KLA60`	Systémový CAN kanál je připojen na jednotku KLA60
	atribut **`DeviceName`**	**Název CAN zařízení pro KLA60 EtherCAT**	
		`ECAT.`*`KLA60_1`*	Název zařízení pro systémovou sběrnici CAN-BUS připojenou na jednotku KLA60 pro EtherCAT. Název musí být totožný s definicí v projektu EtherCAT studia.
	atribut **`PhysicalCanChannel`**	**číslo fyzického CAN kanálu**	
		`1`	pro periferie CAN-BUS
		`xx`	
	atribut **`CanSpeed`**	**komunikiační rychlost CAN-busového kanálu [bit/s]**	
		`1000000`	komunikační rychlost 1 MBd (default)
		`500000`	komunikační rychlost 0.5 MBd
		`250000`	komunikační rychlost 0.25 MBd
		`125000`	komunikační rychlost 0.125 MBd
		`100000`	komunikační rychlost 0.1 MBd

	atribut **ServicePeriod**	**perioda obsluhy CAN-busového kanálu v mikro sekundách** (Pro KLA60 na sběrnici EtherCAT se atribut neuvádí)	
		250	perioda obsluhy CAN-BUSu po ¼ ms (default)
		500	perioda obsluhy CAN-BUSu po ½ ms
		1000	perioda obsluhy CAN-BUSu po 1 ms
	atribut **SyncPeriod**	**perioda posílání SYNCu jako násobek základního taktu**	
		1	perioda vysílání SYNC po 1 ms (default)
		1,2,..,15	perioda vysílání SYNC
	atribut **InOut08Cnt**	**Počet periferií INOUT08 připojených k danému kanálu**	
		0	(default)
		1-32	počet INOUT08
	atribut **Kla50Cnt**	**Počet standardních panelů včetně EPU**	
		0	(default)
		1,2,3,4	počet panelů
	atribut **KnobCnt**	**Počet točítek**	
		0	(default)
		1,2	počet točítek

Příklad:
Připojení sběrnice CAN-BUS na jednotku KLA60 přes EtherCAT.

```
<CANChannel No="1"
      Active="1"
      DeviceType="KLA60"
      DeviceName="ECAT.KLA60_1"
      PhysicalCANChannel="1"
      SyncPeriod="8"
      InOut08Cnt="0"
      CANSpeed="500000"
      Kla50Cnt="1"
      KnobCnt="1">
</CANChannel>
```

O připojení periferií na sběrnici EtherCAT pojednává blíže návod „Periferie EtherCAT“ a o propojení logických (virtuálních) spojů „Logické vstupy a výstupy modulů systému“.

Konfigurace „**HMI**“ zařízení:

element **HMIDevice**		**Nastavení HMI zařízení**	
	atribut **No**	**číslo HMI zařízení**	
		xx	Číslo HMI zařízení (0,1,2, .. , 7) 0 = standardní panel systému
	atribut **Active**	**je HMI zařízení aktivní ?**	
		0	HMI zařízení neaktivní (default)
		1	HMI zařízení aktivní
	atribut **DeviceType**	**Podporované typy HMI zařízení**	
		PanelKla50	Připojení na panel typu KLA50 na sběrnici CAN-BUS.
		PanelKla60	Připojení na panel typu KLA60 na sběrnici EtherCAT.
		KnobKla50	Připojení na točítko typu KLA50 na sběrnici CAN-BUS.
		JoystickKla50	Připojení na joystick typu KLA50 na sběrnici CAN-BUS.
		Virtual	Virtuální klávesnice
	Atribut **DeviceName**	**Název HMI zařízení**	
		CAN[1].KLA50[x]	Název zařízení typu KLA50 na sběrnici CAN-BUS. Parametr [x] určuje NODE-ID zařízení: 41h … 0 42h … 1 43h … 2 …
		ECAT.*KLA60_1*	Název zařízení typu KLA60 pro EtherCAT. Název za předponou „ECAT.“ musí být totožný s definicí v projektu EtherCAT studia.

Příklad:
Příklad pro jeden standardní panel typu KLA60 na sběrnici EtherCAT a jeden panel typu KLA50 na sběrnici CAN-BUS s NODE-ID: 41h

```
<HMIDevice No="0"
      Active="1"
      DeviceName="ECAT.KLA60_1"
      DeviceType="PanelKla60">
</HMIDevice>
<HMIDevice No="1"
      Active="1"
      DeviceName="CAN[1].KLA50[0]"
      DeviceType="PanelKla50">
</HMIDevice>
```

Konfigurace pro jednotlivá tlačítka všech ovládacích panelů jsou v XML tvaru uvedena v souborech typu „KbdConfig". Pro systém je potřeba všechny použité ovládací panely zaregistrovat v registrech Windows. Registraci hlavního panelu systému provede SETUP pro WinCNC a registrace dalších ovládacích panelů se musí provést pomocí SETUPu pro PLC ve skriptu „PLC.NSI".

Registrace se provede v pomocí klíče „Keyboards":
SOFTWARE\MEFI\WinCNC\Install\Channels\Channel0\Keyboards

Přidělení klíčů pro jednotlivé typy ovládacích panelů:

Typ	Klíč	popis
Panel	**Keyboard0**	Hlavní panel systému
Panel nebo joystick	**Keyboard1**	Externí panel (například galvanicky oddělený EPU)
Panel nebo joystick	**Keyboard2**	Externí panel
Panel nebo joystick	**Keyboard3**	
Točítko	**Keyboard4**	Panýlek s kolečkem s tlačítky a displejem
Točítko	**Keyboard5**	
	Keyboard6	Virtuální klavesnice
	Keyboard7	

Registrace pro dva konfigurační soubory pro hlavní ovládací panel systému (v oblasti Install):

```
HKEY_LOCAL_MACHINE\SOFTWARE\MEFI\WinCNC\Install\Channels\Channel0\Keyboards
\Keyboard0 "ConfigFiles"="BaseArea.KbdConfig TechnologyArea.KbdConfig"
```

Registrace pro točítko (v oblasti Machine):

```
HKEY_LOCAL_MACHINE\SOFTWARE\MEFI\WinCNC\Machine\Channels\Channel0\Keyboards
\Keyboard4 "ConfigFiles"="KnobArea.KbdConfig"
```

Skript „PLC.NSI" pro PLC Setup musí obsahovat příkaz pro kopírování a mazání definičních souborů typu „KbdConfig" a zaregistrování do registrů Windows ve tvaru:

```
${registry::Write}
"HKLM\Software\MEFI\WinCNC\Machine\Channels\Channel0\Keyboards\Keyboard4"
"ConfigFiles" "KnobArea.KbdConfig" "REG_MULTI_SZ" $R0
```

11.3 Konfigurace tlačítek

Konfigurace tlačítek se provádí v souborech typu "KbdConfig" v XML tvaru v elementu <KeyboardConfig>.

11.3.1 Konfigurace tlačítek s jednou funkcí

Pro každé tlačítko je možno definovat stisky kláves jako na PC klávesnici, přímé zadání kódu tlačítka, příkazy pro systémový software, příkazy pro programy reálného času, příkazy pro PLC program, vyvolání dialogu a pod.

Každé tlačítko na příslušném ovládacím panelu je určeno podle fyzického kódu tlačítka („scan-kód“). Tento kód lze zjistit například pomocí diagnostické obrazovky pro externí periferie (CanView), nebo pro test panelu.

element **KeyConfig**		**Konfigurace pro jedno tlačítko**	
	atribut **ScanCode**	**„Scan“ kód tlačítka**	
		0xXX	fyzický kód tlačítka
	atribut **Type**	**Typ tlačítka**	
		Normal	Normální typ tlačítka
		Multi	Tlačítko multifunkční – pořadí stisků mění funkci
	atribut **VirtualKey**	**Standardní kód PC klávesnice**	
		Výčet	Tlačítko simuluje stisk standardní PC klávesnice
	atribut **EnableRepeat**	**Samočinné opakování stisků – „repeat“**	
		False	Samočinné opakování stisků zakázáno (default)
		True	Samočinné opakování stisků povoleno
	atribut **UnicodeCharLower**	**Přímé zadání kódu**	
		0xXX	hexadecimální zápis kódu – Unicode (bez SHIFT)
	atribut **UnicodeCharUpper**	**Přímé zadání kódu**	
		0xXX	hexadecimální zápis kódu – Unicode (aktivní SHIFT)
	atribut **Command**	**Příkaz pro CNC**	
		Výčet	Příkaz pro systémovou část software
	atribut **PlcCommand**	**Příkaz pro PLC**	
		Výčet	Příkaz pro PLC program (příkazy jsou definovány jako PLC konstanty)
	atribut **RtmCommand**	**Příkaz pro reálný čas (RTM)**	
		Výčet	Příkaz pro programy reálného času (viz dále)
	element **Dialog**	**Vyvolání dialogu**	
		xxx	Název dialogového okna
	element	**Vyvolání rozvržení**	
	Layout	xxx	Název rozvržení
	element	**Multifunkční tlačítko**	
	Multikey	xxx	

Příklad:
Konfigurace jednoduchých tlačítek v XML tvaru:

```
<KeyboardConfig>

<KeyConfig ScanCode="0x6C" Type="Normal"
      Command="ID_MMM_MAN">
</KeyConfig>                                    <!-- volba MAN -->

<KeyConfig ScanCode="0x6F" Type="Normal"
      PlcCommand="CMD_PERF">
</KeyConfig>                                    <!-- příkaz pro PLC -->

<KeyConfig ScanCode="0x03" Type="Normal"
      RtmCommand="RTMID_MMM_SEL00" PlcCommand="CMD_LUBR1">
</KeyConfig>                                    <!-- RTM a PLC příkaz -->

<KeyConfig ScanCode="0x6D" Type="Normal">
      <Dialog>SamplePlcTab</Dialog>
</KeyConfig>                                    <!-- Editor PLC tabulky -->

</KeyboardConfig>
```

11.3.2 Konfigurace tlačítek s více funkcemi

Tlačítka mohou mít více funkcí definovaných pro jeden „scan-kód“, podobně jako je na počítačových klávesnicích příkaz `SHIFT` a `CAPS-LOCK`. V současné verzi software může mít každé tlačítko až 4 funkce. Každá funkce tlačítka může mít vlastní virtuální kód systémový příkaz, příkaz pro reálný čas, příkaz pro PLC program nebo jiné vyvolání dialogu apod. Přepínání funkcí tlačítka se provede pomocí příkazů pro RTM a jsou typu přepínací: „Toggle“, nastavovací: „Switch“ nebo s účinkem po dobu držení (viz dále).

element **KeyConfig**			**Konfigurace pro jedno multifunkční tlačítko**	
	atribut **ScanCode**		**„Scan“ kód tlačítka**	
			`0xXX`	fyzický kód tlačítka
	element **Function**		**Funkce tlačítka** Další funkce tlačítka (podobně jako SHIFT)	
		atribut **No**	**Číslo funkce tlačítka**	
			`0`	Základní funkce tlačítka je 0 (default)
			`1,2,3`	Další funkce tlačítka
		atribut **Type**	**Typ tlačítka**	
			`Normal`	Normální typ tlačítka
			`Multi`	Tlačítko multifunkční – pořadí stisků mění funkci
		atribut **VirtualKey**	**Standardní kód PC klávesnice**	
			`Výčet`	Tlačítko simuluje stisk standardní PC klávesnice
		atribut **EnableRepeat**	**Samočinné opakování stisků – „repeat“**	
			`False`	Samočinné opakování stisků zakázáno (default)
			`True`	Samočinné opakování stisků povoleno
		atribut **UnicodeCharLower**	**Přímé zadání kódu**	
			`0xXX`	hexadecimální zápis kódu – Unicode (bez SHIFT)
		atribut **UnicodeCharUpper**	**Přímé zadání kódu**	
			`0xXX`	hexadecimální zápis kódu – Unicode (aktivní SHIFT)
		atribut **Command**	**Příkaz pro CNC**	
			`Výčet`	Příkaz pro systémovou část software
		atribut **PlcCommand**	**Příkaz pro PLC**	
			`Výčet`	Příkaz pro PLC program (příkazy jsou definovány jako PLC konstanty)
		atribut **RtmCommand**	**Příkaz pro reálný čas (RTM)**	
			`Výčet`	Příkaz pro programy reálného času (viz dále)
		element **Dialog**	**Vyvolání dialogu**	
			`xxx`	Název zaregistrovaného dialogu
		element **Layout**	**Vyvolání rozvržení**	
			`xxx`	Název rozvržení

Příklad:
Konfigurace vícefunkčního tlačítka v XML tvaru:

```
<KeyConfig ScanCode="0x6A">
      <Function No="0" Type="Normal"
            RtmCommand="RTMID_IMMM_MOVE01">
      </Function>
      <Function No="1" Type="Normal"
            RtmCommand="RTMID_IMMM_MOVE02">
      </Function>
      <Function No="2" Type="Normal"
            RtmCommand="RTMID_IMMM_MOVE03">
      </Function>
</KeyConfig>
```

PLC program má možnost sledovat, v jaké funkci se zařízení nachází a podle toho například měnit stav indikace na obrazovce systému.

Přepínání funkcí tlačítka se provede pomocí příkazů pro RTM a jsou typu:

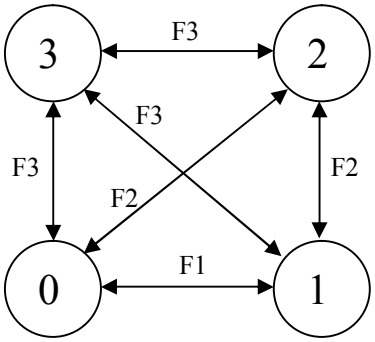
TOGGLE
Přepínání
	Přepínání funguje podobně jako funkce `CAPS-LOCK` na PC klávesnici ale pro 4 úrovně. Každá funkce pro každou klávesnici má svou stavovou proměnnou. Tato proměnná se funkcí `TOGGLE` invertuje.
	
	Pro klávesnici „x“ a funkci „y“ existují RTM příkazy typu:
	`RTMID_KEYBx_FUNCTIONy_TOGGLE`
	
	`x = 0,1,2,..,7`
	`y = 1,2,3`
	
	Stavové proměnné pro každou klávesnici (přístupné z PLC), (viz stavový diagram)
	
	`TLyW.TL_Fun1State` pro přepínání funkci 1
	`TLyW.TL_Fun2State` pro přepínání funkci 2
	`TLyW.TL_Fun3State` pro přepínání funkci 3

SWITCH
Nastavování
	Přímé nastavování funkcí tlačítka. Stavové proměnné klávesnic se také nastaví na požadovaný stav funkce.
	
	Pro klávesnici „x“ a funkci „y“ existují RTM příkazy typu:
	`RTMID_KEYBx_FUNCTIONy_SWITCH`
	
	`x = 0,1,2,..,7`
	`y = 0,1,2,3`

PUSH
Po dobu držení
	Nastavování funkcí tlačítka s účinkem po dobu držení.
	
	Pro klávesnici „x“ a funkci „y“ existují RTM příkazy typu:
	`RTMID_KEYBx_FUNCTIONy`
	
	`x = 0,1,2,..,7`
	`y = 0,1,2,3`

Stavový diagram
Pro přepínání funkcí
TOGGLE

```
S1  S2  S3
0   0   1
0   1   1          S1  S2  S3
1   0   1          0   1   0
1   1   1          1   1   0
```




```
S1  S2  S3         S1  S2  S3
0   0   0          1   0   0
```

PLC program má možnost zjistit informace o každé klávesnici ze struktury `TLxWS`. Dále uvedeme užitečné informace pro PLC, které tato struktura obsahuje:

Prvek struktury	typ	popis
`TL_Function`	BYTE 0, 1, 2, 3	Aktuální platná funkce, ve které se klávesnice nachází.
`TL_FunctionReq`	BYTE 0, 1, 2, 3	Požadovaná funkce pro danou klávesnici
`TL_Fun1State`	BYTE 0, 0FFh	Stavová proměnná pro funkci „F1“
`TL_Fun2State`	BYTE 0, 0FFh	Stavová proměnná pro funkci „F2“
`TL_Fun3State`	BYTE 0, 0FFh	Stavová proměnná pro funkci „F3“

Struktury pro jednotlivé klávesnice:
TL1W až TL4W …. Klávesnice na panelech (Keyboard0 až Keyboard3)
TL1K až TL2K …. Klávesnice na točítkách (Keyboard4 až Keyboard5)
TL1P …. Virtuální klávesnice (Keyboard6)

Příklad:

PLC zjistí aktuální číslo funkce pro 1. klávesnici:

```
LOD   BYTE.TL1W.TL_Function        ;načte číslo funkce (0,1,2,3)
```

Indikace stavu funkce 1 pro 2. klávesnici:

```
LDR   TL2W.TL_Fun1State.B0         ;načte bit stavu funkce F1
WR    flShift
```

11.3.3 Standardní kódy (VirtualKey)

Dále je uveden výčet všech standardních kódu PC klávesnice, které je možno použít v konfiguraci pomocí atributu „VirtualKey“.

VK_LBUTTON	VK_7	VK_F2	VK_MEDIA_PLAY_PAUSE
VK_RBUTTON	VK_8	VK_F3	VK_LAUNCH_MAIL
VK_CANCEL	VK_9	VK_F4	VK_LAUNCH_MEDIA_SELECT
VK_MBUTTON	VK_A	VK_F5	VK_LAUNCH_APP1
VK_XBUTTON1	VK_B	VK_F6	VK_LAUNCH_APP2
VK_XBUTTON2	VK_C	VK_F7	VK_OEM_1
VK_BACK	VK_D	VK_F8	VK_OEM_PLUS
VK_TAB	VK_E	VK_F9	VK_OEM_COMMA
VK_CLEAR	VK_F	VK_F10	VK_OEM_MINUS
VK_RETURN	VK_G	VK_F11	VK_OEM_PERIOD
VK_SHIFT	VK_H	VK_F12	VK_OEM_2
VK_CONTROL	VK_I	VK_F13	VK_OEM_3
VK_MENU	VK_J	VK_F14	VK_OEM_4
VK_PAUSE	VK_K	VK_F15	VK_OEM_5
VK_CAPITAL	VK_L	VK_F16	VK_OEM_6
VK_KANA	VK_M	VK_F17	VK_OEM_7
VK_HANGEUL	VK_N	VK_F18	VK_OEM_8
VK_HANGUL	VK_O	VK_F19	VK_OEM_AX
VK_JUNJA	VK_P	VK_F20	VK_OEM_102
VK_FINAL	VK_Q	VK_F21	VK_ICO_HELP
VK_HANJA	VK_R	VK_F22	VK_ICO_00
VK_KANJI	VK_S	VK_F23	VK_PROCESSKEY
VK_ESCAPE	VK_T	VK_F24	VK_ICO_CLEAR
VK_CONVERT	VK_U	VK_NUMLOCK	VK_PACKET
VK_NONCONVERT	VK_V	VK_SCROLL	VK_OEM_RESET
VK_ACCEPT	VK_W	VK_OEM_NEC_EQUAL	VK_OEM_JUMP
VK_MODECHANGE	VK_X	VK_OEM_FJ_JISHO	VK_OEM_PA1
VK_SPACE	VK_Y	VK_OEM_FJ_MASSHOU	VK_OEM_PA2
VK_PRIOR	VK_Z	VK_OEM_FJ_TOUROKU	VK_OEM_PA3
VK_NEXT	VK_LWIN	VK_OEM_FJ_LOYA	VK_OEM_WSCTRL
VK_END	VK_RWIN	VK_OEM_FJ_ROYA	VK_OEM_CUSEL
VK_HOME	VK_APPS	VK_LSHIFT	VK_OEM_ATTN
VK_LEFT	VK_SLEEP	VK_RSHIFT	VK_OEM_FINISH
VK_UP	VK_NUMPAD0	VK_LCONTROL	VK_OEM_COPY
VK_RIGHT	VK_NUMPAD1	VK_RCONTROL	VK_OEM_AUTO
VK_DOWN	VK_NUMPAD2	VK_LMENU	VK_OEM_ENLW
VK_SELECT	VK_NUMPAD3	VK_RMENU	VK_OEM_BACKTAB
VK_PRINT	VK_NUMPAD4	VK_BROWSER_BACK	VK_ATTN
VK_EXECUTE	VK_NUMPAD5	VK_BROWSER_FORWARD	VK_CRSEL
VK_SNAPSHOT	VK_NUMPAD6	VK_BROWSER_REFRESH	VK_EXSEL
VK_INSERT	VK_NUMPAD7	VK_BROWSER_STOP	VK_EREOF
VK_DELETE	VK_NUMPAD8	VK_BROWSER_SEARCH	VK_PLAY
VK_HELP	VK_NUMPAD9	VK_BROWSER_FAVORITES	VK_ZOOM
VK_0	VK_MULTIPLY	VK_BROWSER_HOME	VK_NONAME
VK_1	VK_ADD	VK_VOLUME_MUTE	VK_PA1
VK_2	VK_SEPARATOR	VK_VOLUME_DOWN	VK_OEM_CLEAR
VK_3	VK_SUBTRACT	VK_VOLUME_UP	.
VK_4	VK_DECIMAL	VK_MEDIA_NEXT_TRACK	.
VK_5	VK_DIVIDE	VK_MEDIA_PREV_TRACK	.
VK_6	VK_F1	VK_MEDIA_STOP	.

11.3.4 Příkazy pro CNC (Command)

Dále je uveden výčet všech příkazů pro systémovou část CNC, které je možno použít v konfiguraci pomocí atributu „Command“.

Příkaz	Funkce
ID_SMTOP_BTN0 - ID_SMTOP_BTN15	Simulace stisku tlačítek horního softwarového menu (horní softwarové menu se zpravidla nepoužívá)
ID_SMLEFT_BTN0 - ID_SMLEFT_BTN15	Simulace stisku tlačítek levého softwarového menu (levé softwarové menu se zpravidla nepoužívá)
ID_SMRIGHT_BTN0 - ID_SMRIGHT_BTN15	Simulace stisku tlačítek pravého softwarového menu
ID_SMBOTTOM_BTN0 - ID_SMBOTTOM_BTN15	Simulace stisku tlačítek dolního softwarového menu
ID_SM_LAYOUT ID_SM_MAIN ID_SM_SWITCH ID_SM_LAYOUTCOND ID_SM_MAINCOND	Příkazy pro manipulaci se soft menu
ID_FILE_SAVE_ALL ID_FILE_NEW ID_FILE_OPEN ID_FILE_OPENSYSTEM ID_FILE_CLOSE ID_FILE_SAVE ID_FILE_SAVE_AS ID_FILE_PAGE_SETUP ID_FILE_PRINT_SETUP ID_FILE_PRINT ID_FILE_PRINT_DIRECT ID_FILE_PRINT_PREVIEW ID_FILE_UPDATE ID_FILE_SAVE_COPY_AS ID_FILE_SEND_MAIL ID_FILE_NEW_FRAME	Příkazy pro práci se soubory
ID_FILE_MRU_FIRST ID_FILE_MRU_FILE1 - ID_FILE_MRU_FILE16	Příkazy pro otevření naposled editovaných souborů
ID_EDIT_SELECTINGMODE ID_EDIT_CLEAR ID_EDIT_CLEAR_ALL ID_EDIT_COPY ID_EDIT_CUT ID_EDIT_FIND ID_EDIT_PASTE ID_EDIT_PASTE_LINK ID_EDIT_PASTE_SPECIAL ID_EDIT_REPEAT ID_EDIT_REPLACE ID_EDIT_SELECT_ALL ID_EDIT_UNDO ID_EDIT_REDO ID_EDIT_FIND_PREVIOUS ID_EDIT_TOGGLE_BOOKMARK0 - _BOOKMARK9 ID_EDIT_GO_BOOKMARK0 - _BOOKMARK9 ID_EDIT_CLEAR_BOOKMARKS ID_EDIT_TOGGLE_BOOKMARK ID_EDIT_GOTO_NEXT_BOOKMARK ID_EDIT_GOTO_PREV_BOOKMARK ID_EDIT_CLEAR_ALL_BOOKMARKS ID_EDIT_GOTO_LAST_CHANGE ID_EDIT_FIND_INCREMENTAL_FORWARD ID_EDIT_FIND_INCREMENTAL_BACKWARD ID_EDIT_MATCHBRACE, ID_EDIT_LOWERCASE ID_EDIT_UPPERCASE ID_EDIT_CAPITALIZE ID_EDIT_SENTENCE	Příkazy pro editaci souboru

ID_EDIT_SWAPCASE ID_EDIT_GOTO	
ID_READ_ONLY	Nastavit/zrušit příznak "jen pro čtení" pro aktuální soubor
ID_VIEW_LAYOUTCONFIG	Zobrazit "volbu indikace"
ID_FORMAT_BPR ID_FORMAT_SHOW_ADDRESS ID_FORMAT_SHOW_HEX ID_FORMAT_SHOW_ASCII ID_FORMAT_WIDE_ADDRESS ID_FORMAT_BPR_1 - ID_FORMAT_BPR_9 ID_FORMAT_FONT	Příkazy pro formátování
ID_TOOLS_OPTIONS ID_TOOLS_TOOL0 - ID_TOOLS_TOOL9	Příkazy pro nástroje
ID_WINDOW_MDINEXT ID_WINDOW_MDIPREV	MDI okna
ID_APP_ABOUT ID_APP_EXIT ID_SYSTEM_SWITCHOFF	Příkazy pro aplikaci
ID_CNC_SELECTCANUL ID_CNC_SELECTREF0 - ID_CNC_SELECTREF9 ID_CNC_SELECTNCP ID_CNC_SELECTBLOCK ID_CNC_SELECTBLOCK0 - ID_CNC_SELECTBLOCK9 ID_CNC_START ID_CNC_STOP ID_CNC_CONT ID_CNC_REVERSE ID_CNC_FORWARD ID_CNC_BACKWARD ID_CNC_GOFORWARD ID_CNC_GOBACKWARD ID_CNC_IMMBACKWARD ID_CNC_BLOCKBYBLOCK ID_CNC_CONDSTOPMODE ID_CNC_STARTPLC ID_CNC_STOPPLC	Centrální anulace Reference - konfigurace 0 – 9 Volba programu Volba bloku Volba bloku - konfigurace 0 – 9 Start Stop Pokračování po stopu i v případě, že bylo odjeto z místa stopu Změna směru vykonávání programu (vpřed/vzad) Směr vykonávání programu – vpřed Směr vykonávání programu – vzad Vykonávání programu vpřed po dobu držení tlačítka , stisk ve stopu Vykonávání programu vzad po dobu držení tlačítka, stisk ve stopu Vykonávání programu vzad po dobu držení tlačítka, stisk v průběhu jízdy Režim blok po bloku (zapnout/vypnout) Režim podmíněného stopu (zapnout/vypnout, ovlivňuje chování M01) Start PLC Stop PLC
ID_MMM_MAN ID_MMM_MANEND ID_MMM_RAPIDTRAVERSE	Zapnutí ručních pojezdů Ukončení ručních pojezdů Rychloposuv (lze přimáčknout při ruční jízdě)
ID_MMM_MOVE00 - ID_MMM_MOVE15 ID_MMM_MOVE00NEG – ID_MMM_MOVE15NEG	Ruční pohyb NC osy v kladném či záporném (...NEG) směru (doporučuje se použít funkce RtmCommand)
ID_MMM_MOVEX ID_MMM_MOVEY ID_MMM_MOVEZ ID_MMM_MOVEXNEG ID_MMM_MOVEYNEG ID_MMM_MOVEZNEG ID_MMM_MOVEXY ID_MMM_MOVEXNEGY ID_MMM_MOVEXYNEG ID_MMM_MOVEXNEGYNEG ID_MMM_MOVEZX ID_MMM_MOVEZNEGX ID_MMM_MOVEZXNEG ID_MMM_MOVEZNEGXNEG ID_MMM_MOVEYZ ID_MMM_MOVEYNEGZ ID_MMM_MOVEYZNEG ID_MMM_MOVEYNEGZNEG	Ruční pohyb jedné či dvou geometrických os (doporučuje se použít funkce RtmCommand)
ID_MMM_BACK0 - ID_MMM_BACK9	Návrat na dráhu - konfigurace 0 - 9

ID_OVERRIDE_FEED_INC ID_OVERRIDE_FEED_DEC ID_OVERRIDE_SPINDLESPEED_INC ID_OVERRIDE_SPINDLESPEED_DEC ID_OVERRIDE_PLC1_INC ID_OVERRIDE_PLC1_DEC ID_OVERRIDE_PLC2_INC ID_OVERRIDE_PLC2_DEC ID_OVERRIDE_PLC3_INC ID_OVERRIDE_PLC3_DEC ID_OVERRIDE_PLC4_INC ID_OVERRIDE_PLC4_DEC ID_OVERRIDE_PLC5_INC ID_OVERRIDE_PLC5_DEC ID_OVERRIDE_PLC6_INC ID_OVERRIDE_PLC6_DEC	Ovládání overidů (inkrementace, dkrementace, používá konfiguraci daného overidu, Inc/Dec odpovídá jednomu "cvaknutí" potenciometru)

11.3.5 Příkazy pro PLC (PlcCommand)

Problematika příkazů pro PLC program je popsána v kapitole „PLC konfigurace a konstanty“. Je nutno definovat společné konstanty pro PLC a CNC část.

PLC konstanty slouží pro symbolické definování konstant, které mají být k dispozici jak v PLC programu. tak v CNC části systému. Konstanty pak lze využít na definici příkazů pro PLC.

Příkazy se v PLC programu zpracovávají pomocí událostní procedury `_ON_CMD`.

11.3.6 Příkazy pro RTM (RtmCommand)

Jedná se o příkazy, které zpracuje přímo program reálného času. Vzhledem k tomu, že i obsluha tlačítek se provádí v reálném čase, nehrozí tak žádné prodlení při zpracování příkazů.

Příkaz	Funkce
RTMID_MMM_SEL00 - RTMID_MMM_SEL15	Předvolba NC osy pro ruční jízdu (například u točítka)
RTMID_IMMM_SEL00 - RTMID_IMMM_SEL15	Předvolba NC osy pro ruční jízdu s okamžitou volbou MAN
RTMID_MMM_SELX RTMID_MMM_SELY RTMID_MMM_SELZ	Předvolba geometrické osy pro ruční jízdu (například u točítka)
RTMID_IMMM_SELX RTMID_IMMM_SELY RTMID_IMMM_SELZ	Předvolba geometrické osy pro ruční jízdu s okamžitou volbou MAN
RTMID_MMM_MOVE00 - RTMID_MMM_MOVE15	Ruční pohyb NC osy v kladném směru
RTMID_IMMM_MOVE00 - RTMID_IMMM_MOVE15	Ruční pohyb NC osy v kladném směru s okamžitou volbou MAN
RTMID_MMM_MOVE00NEG - RTMID_MMM_MOVE15NEG	Ruční pohyb NC osy v záporném směru
RTMID_IMMM_MOVE00NEG - RTMID_IMMM_MOVE15NEG	Ruční pohyb NC osy v záporném směru s okamžitou volbou MAN
RTMID_MMM_MOVEX RTMID_MMM_MOVEY RTMID_MMM_MOVEZ	Ruční pohyb geometrické osy v kladném směru
RTMID_IMMM_MOVEX RTMID_IMMM_MOVEY RTMID_IMMM_MOVEZ	Ruční pohyb geometrické osy v kladném směru s okamžitou volbou MAN
RTMID_MMM_MOVEXNEG RTMID_MMM_MOVEYNEG RTMID_MMM_MOVEZNEG	Ruční pohyb geometrické osy v záporném směru
RTMID_IMMM_MOVEXNEG RTMID_IMMM_MOVEYNEG RTMID_IMMM_MOVEZNEG	Ruční pohyb geometrické osy v záporném směru s okamžitou volbou MAN
RTMID_MMM_MOVEXY RTMID_MMM_MOVEXNEGY RTMID_MMM_MOVEXYNEG RTMID_MMM_MOVEXNEGYNEG RTMID_MMM_MOVEZX RTMID_MMM_MOVEZNEGX RTMID_MMM_MOVEZXNEG RTMID_MMM_MOVEZNEGXNEG RTMID_MMM_MOVEYZ RTMID_MMM_MOVEYNEGZ RTMID_MMM_MOVEYZNEG RTMID_MMM_MOVEYNEGZNEG	Ruční pohyb dvou geometrických os v kladném či záporném (...NEG) směru
RTMID_IMMM_MOVEXY RTMID_IMMM_MOVEXNEGY RTMID_IMMM_MOVEXYNEG RTMID_IMMM_MOVEXNEGYNEG RTMID_IMMM_MOVEZX RTMID_IMMM_MOVEZNEGX RTMID_IMMM_MOVEZXNEG RTMID_IMMM_MOVEZNEGXNEG RTMID_IMMM_MOVEYZ RTMID_IMMM_MOVEYNEGZ RTMID_IMMM_MOVEYZNEG RTMID_IMMM_MOVEYNEGZNEG	Ruční pohyb dvou geometrických os v kladném či záporném (...NEG) směru s okamžitou volbou MAN
RTMID_MMM_MOVE00RT - RTMID_MMM_MOVE15RT	Ruční pohyb NC osy v kladném směru rychloposuvem

RTMID_IMMM_MOVE00RT - RTMID_IMMM_MOVE15RT	Ruční pohyb NC osy v kladném směru rychloposuvem s okamžitou volbou MAN
RTMID_MMM_MOVE00NEGRT - RTMID_MMM_MOVE15NEGRT	Ruční pohyb NC osy v záporném směru rychloposuvem
RTMID_IMMM_MOVE00NEGRT - .._IMMM_MOVE15NEGRT	Ruční pohyb NC osy v záporném směru rychloposuvem s okamžitou volbou MAN
RTMID_MMM_MOVEXRT RTMID_MMM_MOVEYRT RTMID_MMM_MOVEZRT	Ruční pohyb geometrické osy v kladném směru rychloposuvem
RTMID_IMMM_MOVEXRT RTMID_IMMM_MOVEYRT RTMID_IMMM_MOVEZRT	Ruční pohyb geometrické osy v kladném směru rychloposuvem s okamžitou volbou MAN
RTMID_MMM_MOVEXNEGRT RTMID_MMM_MOVEYNEGRT RTMID_MMM_MOVEZNEGRT	Ruční pohyb geometrické osy v záporném směru rychloposuvem
RTMID_IMMM_MOVEXNEGRT RTMID_IMMM_MOVEYNEGRT RTMID_IMMM_MOVEZNEGRT	Ruční pohyb geometrické osy v záporném směru rychloposuvem s okamžitou volbou MAN
RTMID_MMM_MOVE RTMID_MMM_MOVENEG	Ruční pohyb podle předvolby v kladném či záporném (...NEG) směru
RTMID_IMMM_MOVE RTMID_IMMM_MOVENEG	Ruční pohyb podle předvolby v kladném či záporném směru s okamžitou volbou MAN
RTMID_MMM_MOVERT RTMID_MMM_MOVENEGRT	Ruční pohyb podle předvolby v kladném či záporném (...NEG) směru rychloposuvem
RTMID_IMMM_MOVERT RTMID_IMMM_MOVENEGRT	Ruční pohyb podle předvolby v kladném či záporném směru rychloposuvem s okamžitou volbou MAN
RTMID_MMM_RAPIDTRAVERSE	Přimáčknutí rychloposuvu
RTMID_MMM_KNOB_ENABLE1 RTMID_MMM_KNOB_DISABLE1 RTMID_MMM_KNOB_ENABLE2 RTMID_MMM_KNOB_DISABLE2	Povolení ovládání z 1.točítka Zákaz ovládání z 1.točítka Povolení ovládání z 2.točítka Zákaz ovládání z 2.točítka
RTMID_IMMM_KNOB_ENABLE1 RTMID_IMMM_KNOB_DISABLE1 RTMID_IMMM_KNOB_ENABLE2 RTMID_IMMM_KNOB_DISABLE2	Povolení ovládání z 1.točítka s okamžitou volbou MAN Zákaz ovládání z 1.točítka s okamžitou volbou MAN Povolení ovládání z 2.točítka s okamžitou volbou MAN Zákaz ovládání z 2.točítka s okamžitou volbou MAN
RTMID_MMM_SHIFT_REQ RTMID_MMM_SHIFT_ACT RTMID_MMM_SHIFT_ENABLE RTMID_MMM_SHIFT_DISABLE RTMID_MMM_SHIFT_ACTIVATE RTMID_MMM_SHIFT_DEACTIVATE	Povolení/zrušeni režimu SHIFT Aktivace/deaktivace režimu SHIFT Povolení režimu SHIFT Zrušení režimu SHIFT Aktivace režimu SHIFT Deaktivace režimu SHIFT
RTMID_IMMM_SHIFT_REQ RTMID_IMMM_SHIFT_ACT RTMID_IMMM_SHIFT_ENABLE RTMID_IMMM_SHIFT_DISABLE RTMID_IMMM_SHIFT_ACTIVATE RTMID_IMMM_SHIFT_DEACTIVATE	Povoleni/zrušeni režimu SHIFT s okamžitou volbou MAN Aktivace/deaktivace režimu SHIFT s okamžitou volbou MAN Povolení režimu SHIFT s okamžitou volbou MAN Zrušení režimu SHIFT s okamžitou volbou MAN Aktivace režimu SHIFT s okamžitou volbou MAN Deaktivace režimu SHIFT s okamžitou volbou MAN
RTMID_MMM_KNOB_STEP1 RTMID_MMM_KNOB_STEP2 RTMID_MMM_KNOB_STEP5 RTMID_MMM_KNOB_STEP10 RTMID_MMM_KNOB_STEP20 RTMID_MMM_KNOB_STEP50 RTMID_MMM_KNOB_STEP100 RTMID_MMM_KNOB_STEP200 RTMID_MMM_KNOB_STEP500 RTMID_MMM_KNOB_STEP1000 RTMID_MMM_KNOB_STEP_UP RTMID_MMM_KNOB_STEP_DOWN	Předvolba kroku kolečka – 1 µm Předvolba kroku kolečka – 2 µm Předvolba kroku kolečka – 5 µm Předvolba kroku kolečka – 10 µm Předvolba kroku kolečka – 20 µm Předvolba kroku kolečka – 50 µm Předvolba kroku kolečka – 100 µm Předvolba kroku kolečka – 200 µm Předvolba kroku kolečka – 500 µm Předvolba kroku kolečka – 1000 µm Zvětšení kroku kolečka v řade Zmenšení kroku kolečka v řade
RTMID_MMM_MOVE_KNOB_STEP_UP RTMID_MMM_MOVENEG_KNOB_STEP_DOWN	Sdružené funkce, pohyb podle předvolby nebo změna kroku v režimu SHIFT

RTMID_IMMM_MOVE_KNOB_STEP_UP RTMID_IMMM_MOVENEG_KNOB_STEP_DOWN	Sdružené funkce, pohyb podle předvolby nebo změna kroku v režimu SHIFT s okamžitou volbou MAN
RTMID_MMM_KNOB_INC RTMID_MMM_KNOB_DEC	Simulace kolečka, posun o jeden krok v kladném či záporném směru
RTMID_IMMM_KNOB_INC RTMID_IMMM_KNOB_DEC	Simulace kolečka, posun o jeden krok v kladném či záporném směru s okamžitou volbou MAN
RTMID_MMM_HOLD	Aktivace/deaktivace „přídrže“ pro ruční pohyb
RTMID_KEYB0_FUNCTION1 RTMID_KEYB0_FUNCTION2 RTMID_KEYB0_FUNCTION3 RTMID_KEYBx_FUNCTION1 (x=0,1,2,….,7) RTMID_KEYBx_FUNCTION2 (x=0,1,2,….,7) RTMID_KEYBx_FUNCTION3 (x=0,1,2,….,7)	Funkce 1 pro klávesnici KEYB0, aktivní po dobu držení Funkce 2 pro klávesnici KEYB0 Funkce 3 pro klávesnici KEYB0 Funkce 1 pro klávesnici KEYBx (x=0,1,2,….,7) Funkce 2 pro klávesnici KEYBx (x=0,1,2,….,7) Funkce 3 pro klávesnici KEYBx (x=0,1,2,….,7)
RTMID_KEYB0_FUNCTION0_TOGGLE RTMID_KEYB0_FUNCTION1_TOGGLE RTMID_KEYB0_FUNCTION2_TOGGLE RTMID_KEYB0_FUNCTION3_TOGGLE RTMID_KEYBx_FUNCTION0_TOGGLE (x=0,1,2,….,7) RTMID_KEYBx_FUNCTION1_TOGGLE (x=0,1,2,….,7) RTMID_KEYBx_FUNCTION2_TOGGLE (x=0,1,2,….,7) RTMID_KEYBx_FUNCTION3_TOGGLE (x=0,1,2,….,7)	Funkce 0 pro klávesnici KEYB0, přepínací (TOGGLE) Funkce 1 pro klávesnici KEYB0 Funkce 2 pro klávesnici KEYB0 Funkce 3 pro klávesnici KEYB0 Funkce 0 pro klávesnici KEYBx (x=0,1,2,….,7) Funkce 1 pro klávesnici KEYBx (x=0,1,2,….,7) Funkce 2 pro klávesnici KEYBx (x=0,1,2,….,7) Funkce 3 pro klávesnici KEYBx (x=0,1,2,….,7)
RTMID_KEYB0_FUNCTION0_SWITCH RTMID_KEYB0_FUNCTION1_SWITCH RTMID_KEYB0_FUNCTION2_SWITCH RTMID_KEYB0_FUNCTION3_SWITCH RTMID_KEYBx_FUNCTION0_SWITCH (x=0,1,2,….,7) RTMID_KEYBx_FUNCTION1_SWITCH (x=0,1,2,….,7) RTMID_KEYBx_FUNCTION2_SWITCH (x=0,1,2,….,7) RTMID_KEYBx_FUNCTION3_SWITCH (x=0,1,2,….,7)	Funkce 0 pro klávesnici KEYB0, nastavovací (SWITCH) Funkce 1 pro klávesnici KEYB0 Funkce 2 pro klávesnici KEYB0 Funkce 3 pro klávesnici KEYB0 Funkce 0 pro klávesnici KEYBx (x=0,1,2,….,7) Funkce 1 pro klávesnici KEYBx (x=0,1,2,….,7) Funkce 2 pro klávesnici KEYBx (x=0,1,2,….,7) Funkce 3 pro klávesnici KEYBx (x=0,1,2,….,7)

Příklad:
Příklad pro konfiguraci točítka s aktivací režimu SHIFT v souboru „`KnobArea.KbdConfig`“ v XML tvaru:

```
<KeyboardConfig>
	<Comment>
		Točítko
		scan code:
			3 2 1
			0 7 8
			5 4 6
	</Comment>


<KeyConfig ScanCode="0x03" Type="Normal" RtmCommand="RTMID_MMM_SEL00"></KeyConfig>
<KeyConfig ScanCode="0x02" Type="Normal" RtmCommand="RTMID_MMM_SEL01"></KeyConfig>
<KeyConfig ScanCode="0x01" Type="Normal" RtmCommand="RTMID_MMM_SEL02"></KeyConfig>
<KeyConfig ScanCode="0x00" Type="Normal" RtmCommand="RTMID_MMM_SEL03"></KeyConfig>
<KeyConfig ScanCode="0x07" Type="Normal" RtmCommand="RTMID_MMM_SEL04"></KeyConfig>
<KeyConfig ScanCode="0x08" Type="Normal" RtmCommand="RTMID_MMM_SHIFT_ACT"></KeyConfig>
<KeyConfig ScanCode="0x05" Type="Normal" RtmCommand="RTMID_MMM_MOVENEG"></KeyConfig>
<KeyConfig ScanCode="0x04" Type="Normal" RtmCommand="RTMID_MMM_RAPIDTRAVERSE"></KeyConfig>
<KeyConfig ScanCode="0x06" Type="Normal" RtmCommand="RTMID_MMM_MOVE"></KeyConfig>

</KeyboardConfig>
```

26

26. PERIFERIE PŘIPOJENÉ NA ETHERCAT

26.1 EtherCAT – základy

Systém umožňuje připojit různé periferie (pohony, vstupy a výstupy,..) pomocí sběrnice EtherCAT. Na připojení se využívá samostatný vhodný port ETHERNET. Všechny připojené periferie pracují v reálném čase s periodou obsluhy například 200 µs.

Sběrnici EtherCAT obsluhuje program „KPA EtherCAT Master“. Jedná se o úlohu reálného času nad REAL-TIME jádrem „RTX“. Mezi jeho základní vlastnosti patří: „Distribuovaný Clock“ (DC), vstupy a výstupy přes „Process Image“ (PI), čtení a zápis parametrů „MailBox“ typu „CoE“ nebo „SoE“.

Kompletní obsluha EtherCAT periferií je zpřístupněna v PLC programu.

Konfigurace, prvotní inicializace, konfigurace při změně stavu a definice cyklické výměny dat (Process image) je definovaná v souboru typu XML. Tento soubor je generovaný externím programem „EtherCAT Studio“ například od firmy „Koenig“. Tak konfigurace EtherCAT periferií ovlivní jen PLC program a není závislá na CNC systému. Konfigurační XML soubor má název „MASTER.XML“ a je uložený v adresáři „CNC Machine Files\Config\“ . Do tohoto adresáře se musí překopírovat prostřednictvím SETUPu PLC programu (soubor „PLC.NSI“).

Pro aktivaci sběrnice EtherCAT je nutno nastavit registr Windows. Nastavení se provede automaticky už při instalaci WinCNC (musí se zvolit typ EtherCAT). Další nastavení slouží jen pro speciální požadavky a lze je provést v SETUPu PLC programu v souboru PLC.NSI:

```
WriteRegDWORD HKLM ”Software\MEFI\WinCNC\Machine\Common”
                        .. ”RtxEtherCAT”                1
                        .. ”EtherCATSynchMode”          2
                        .. ”EtherCATDebug”              0 (7)
                        .. ”EtherCATPhaseAdjust”        1
                        .. ”EtherCATDelayOP”            500
                        .. ”EtherCATMasterConfigFile”   Master.xml
                        .. ”EtherCATMasterCycleTime”    1000
                        .. ”FastestPeriod”              1000
                        .. ”HALTimerPeriod”             1000
                        .. ” EtherCATStatistic”         0
                        .. ”EtherCATWatchMaster”        1
                        .. ”EtherCATWatchSlaves”        0
                        .. ”EtherCATOPViaPLC”           0
```

Význam všech možností pro nastavení EtherCATu v registru Windows bude vysvětlen dále.

CNC systém musí mít zakoupené Run-Time licence pro „RTX“ a pro „KPA EtherCAT Master – RTX“.

26.2 Cyklická výměna dat

26.2.1 Obraz „Process Image“ v PLC programu

V PLC programu je „Process Image“ definován jako proměnné, které jsou definovány jako logické vstupy a výstupy pomocí instrukcí **DEF_IN** a **DEF_OUT** (viz návod „Logické vstupy a výstupy modulů systému“).

Systém automaticky definuje logické vstupy a výstupy, které souvisí s EtherCATem na základě konfiguračního XML souboru, který je vytvořen pomocí EtherCAT studia.

Definice „Process Image“ formou logických vstupů a výstupů umožňuje jejich propojování jen na základě konfigurace a bez nutnosti změny PLC programu. Při definici logických prvků pro „virtuální spoje“ v PLC programu se proměnné automaticky deklarují – vymezí se jim požadovaný paměťový prostor.

Datové prvky, které definují logický spoj, mohou být definovány jako pole s ohledem na typ proměnné. V tomto případě logický spoj k danému prvku určuje jeho jméno a koncový index.

Pro každý logický spoj možno definovat konverzi, která se uplatní při propojení prvků.

Všechny logické vstupy a výstupy mají automaticky nastaveno „**přímé sdílení**“ a tak se značně zjednoduší přístup uživatelských obrazovek a dialogů k jejím hodnotám. Na logické vstupy a výstupy není potřeba používat instrukce `SHARE_VAR` a `SHARE_BIT`.

Příklad pro definici prvků „`Process Image`“:

```
;Drive status word, servo 0
DEF_IN      wEcatDriveStatus0,'PIInDriveStatus0',TYPE_UNS_16,-,FAST

;Position feedback, servo 0
DEF_IN      dwEcatDrivePos0,'PIInDrivePos0',TYPE_INT_32,-,FAST

;Master control word, servo 0
DEF_OUT     wEcatDriveControl0,'PIOutDriveControl0',TYPE_UNS_16,-,FAST

;Velocity demand value,  servo 0
DEF_OUT     dwEcatDriveVeloReq0,'PIOutDriveVeloReq0',TYPE_INT_32,-,FAST
```

Pro práci s „ControlWordem“ a „StatusWordem“ je možno deklarovat formální definice bitů a využít složitější adresaci bitu.

Příklad:

```
;Formalni definice bitu pro Controlword a Statusword pro CoE a SoE

;Statusword CoE
CoEStat0: DFM  StatCoE_RDY, StatCoE_ON, StatCoE_ENABLE, StatCoE_FAULT,
    StatCoE_VOLT, StatCoE_QSTOP, StatCoE_DISABLE, StatCoE_WARNING
;StatCoE_RDY            Ready to switch on
;StatCoE_ON             Switched on
;StatCoE_ENABLE         Operation enable
;StatCoE_FAULT          Fault
;StatCoE_VOLT           Disable voltage
;StatCoE_QSTOP          Quick stop
;StatCoE_DISABLE        Switch on disabled
;StatCoE_WARNING        Warning

;Statusword SoE
SoEStat0: DFM , , , StatSoE_COMMAND, , , ,
SoEStat1: DFM StatSoE_M0, StatSoE_M1, StatSoE_M2, , , , StatSoE_READY,
   StatSoE_POWER
;StatSoE_COMMAND        Status command value processing
;StatSoE_M0             Operation mode 0
;StatSoE_M1             Operation mode 1
;StatSoE_M2             Operation mode 2
;StatSoE_READY          ready to operate 0   drive ready
;StatSoE_POWER          ready to operate 1   main power applied

;Controlword CoE
CoECnt0: DFM CntCoE_ON, CntCoE_VOLT, CntCoE_QSTOP, CntCoE_ENABLE,
    CntCoE_M0, CntCoE_M1, CntCoE_M2, CntCoE_FAULT
;CntCoE_ON              Switch ON (1)
;CntCoE_VOLT            Disable Voltage (0)
;CntCoE_QSTOP           Quick Stop (0)
;CntCoE_ENABLE          Enable Operation
;CntCoE_M0              Operation mode specific
;CntCoE_M1              Operation mode specific
;CntCoE_M2              Operation mode specific
;CntCoE_FAULT           Reset Fault

;Controlword SoE
SoECnt0: DFM ,,,,,,,
SoECnt1: DFM CntSoE_OP0, CntSoE_OP1, CntSoE_SYNCH, CntSoE_OP2,
   ,CntSoE_HALT, CntSoE_ENABLE, CntSoE_ON
;CntSoE_OP0             Operation Mode 0
;CntSoE_OP1             Operation Mode 1
;CntSoE_SYNCH           Control unit synchronisation bit
;CntSoE_OP2             Operation Mode 2
;CntSoE_HALT            Halt/Restart drive
;CntSoE_ENABLE          Enable drive
;CntSoE_ON              Drive ON/OFF
```

Příklad mechanizmu pro ENABLE pohonu:

Příklad:

```
; Mechanizmus provádí "ENABLE" pohonu pro servo 0 (typ SoE).
MECH_BEGIN  M_ECAT_DRIVE0_ENABLE
      ; Jako žádanou polohu nastavit aktuální polohu hlášenou pohonem
  LOD       dwEcatDrivePos0
  STO       dwEcatDrivePosReq0
      ;V ControlWordu daného pohonu nastavit příslušné bity
  FL        1,wEcatDriveControl0+1.CntSoE_ON
  FL        1,wEcatDriveControl0+1.CntSoE_ENABLE
  FL        1,wEcatDriveControl0+1.CntSoE_HALT
  FL        1,wEcatDriveControl0+1.CntSoE_SYNCH
  EX
      ;Počkat, až pohon oznámí splnění požadavku
  LDR       wEcatDriveStatus0.StatSoE_COMMAND
  LA        wEcatDriveStatus0+1.StatSoE_READY
  LA        wEcatDriveStatus0+1.StatSoE_POWER
  TEX0      -, 350, LBL_ECAT_DRIVE0_ENABLE_ERR
  ...
```

26.2.2 Propojení „Process Image“ s obrazem v PLC

Propojování jednotlivých prvků se provede jen formou konfigurace, bez nutnosti změny PLC programu.

Logické virtuální spoje jsou definovány dvěma textovými identifikátory. Identifikátor logického výstupu a identifikátor logického vstupu. Textové identifikátory používají „tečkovou konvenci“ podle zásad popsaných v návodu „Logické vstupy a výstupy modulů systému“.

Identifikátor pro připojení k logickým vstupům a výstupům definovaným v „Process image“ pro EtherCAT komunikaci: **`ECAT.<jmeno>[xx]`** .Typ prvku je definován v „Process image“ (vytvořeno v EtherCAT studiu a zaznamenáno v XML souboru). Možnost periody obsluhy: `FAST`.

Konfigurace pro logické spoje jsou v souboru typu „`ChannelConfig`“ v elementu „`Connections`“.

Příklad:

```
<Connections>

  <Connection Source="PLC.Output.PIOutDriveControl0"
              Destination="ECAT.Drive 0.Outputs.Control word"
              Connected="1"></Connection>

  <Connection Source="PLC.Output.PIOutDriveVeloReq0"
              Destination="ECAT.Drive 0.Outputs.Target velocity"
              Connected="1"></Connection>

  <Connection Source="ECAT.Drive 0.Inputs.Status word"
              Destination="PLC.Input.PIInDriveStatus0"
              Connected="1"></Connection>

  <Connection Source="ECAT.Drive 0.Inputs.Position actual value"
              Destination="PLC.Input.PIInDrivePos0"
              Connected="1"></Connection>

</Connections>
```

26.2.3 Start cyklické výměny dat

Cyklická výměna dat se odstartuje automaticky po přechodu Mastru do stavu `OPERATIONAL`. Jen v případě, že se požaduje zpožděný start cyklické výměny, který řídí PLC program a je nastaven klíč `EtherCATOPViaPLC = 1,` tak cyklickou výměnu datmůže odstartovat PLC program prostřednictvím bitu:

`flEcatStartCyclicOperation`.

Nastavením bitu „`flEcatStartCyclicOperation`“ na hodnotu log.1 se spustí cyklická výměna dat mezi EtherCAT Mastrem a všemi EtherCAT Slave. Současně se bude měnit stav Mastru:

`INIT -> PRE-OPERATIONAL -> SAFE-OPERATIONAL -> OPERATIONAL`

Každá změna stavu Mastru je doprovázena vysláním příkazů podle definic v souboru „`MASTER.XML`“. Tím dojde k inicializaci a k nastavení všech potřebných parametrů v EtherCAT periferiích.

Po úspěšném dosažení stavu „`OPERATIONAL`“ systém dá zprávu PLC programu nastavením bitu „**`flEcatOperational`**“. PLC program potom může pokračovat v inicializaci periferií, například přechod do stavu „`ENABLE`“

`flEcatStartCyclicOperation`	Žádost z PLC o start cyklické výměny dat
`flEcatOperational`	Zpráva pro PLC o dosažení stavu „`OPERATIONAL`“

26.3 Stavy EtherCAT Mastru

EtherCAT Master se může nacházet ve stavech:

Stav	Hodnota	Symbol pro PLC
BOOTSTRAP	3	ECAT_BOOTSTRAP
INIT	1	ECAT_INIT
PRE-OPERATIONAL	2	ECAT_PREOPERATIONAL
SAFE-OPERATIONAL	4	ECAT_SAFEOPERATIONAL
OPERATIONAL	8	ECAT_OPERATIONAL

Po správné inicializaci se EtherCAT MASTER nachází ve stavu OPERATIONAL. (je nahozen bit flEcatOperational). PLC program má možnost žádat o změnu stavu MASTRA. Tato změna musí být uvážená, protože se týká všech EtherCAT periferií. Například pohony musí být ve stavu DISABLE. Při opětovném přechodu do stavu OPERATIONAL , je potřeba zrušit referenci a znovu načíst polohu motoru.

Žádosti z PLC o změnu stavu EtherCAT MASTRu:

flEcatBootStrapReq	Žádost z PLC o přechod MASTRu do stavu „BOOTSTRAP“
flEcatInitReq	Žádost z PLC o přechod MASTRu do stavu „INIT“
flEcatPreOperationalReq	Žádost z PLC o přechod MASTRu do stavu „PRE-OPERATIONAL“
flEcatSafeOperationalReq	Žádost z PLC o přechod MASTRu do stavu „SAFE-OPERATIONAL“
flEcatOperationalReq	Žádost z PLC o přechod MASTRu do stavu „OPERATIONAL“

Jako potvrzení změny stavu MASTRu se příslušný bit žádosti sám vynuluje.

Příklad:

```
;Mechanizmus pro přechod do stavu PRE-OPERATIONAL:

MECH_BEGIN M_ECAT_PREOPERATIONAL
      FL    1, flEcatPreOperationalReq
      EX
      LDR   flEcatPreOperationalReq
      TEX1  -, 500, LBL_ECAT_PREOP_ERR
      ...
MECH_END    M_ECAT_PREOPERATIONAL
```

26.4 Mailbox typu CoE a SoE, příkazy pro Mastra

Modul „Mailbox“, který je součástí EtherCAT Mastru slouží pro čtení a nastavování parametrů jednotlivých periferií. Jedná se vždy o jednorázové – neperiodické přístupy k parametrům jednotlivých „Slave“.

26.4.1 Čtení a zápis parametru typu CoE

Pro komunikaci s EtherCAT periferií se využívá komunikační profil „CANopen“ (CoE = „CANopen over EtherCAT“) podle normy: „CiA Standard 402 (DSP402, Device profile for drives and motion control)“. Z toho se využívají hlavně komunikační objekty „SDO (Service Data Object)“ a pro adresaci se využívá „INDEX“ a „SUBINDEX“ objektu podle normy CANopen.

Pro SDO komunikaci byly zavedeny instrukce pro čtení a zápis. Jedná se o instrukce, které mohou být použity jen v mechanizmech, protože se vždy čeká na výsledek komunikace.

instrukce	**ECAT_COE_READ** **ECAT_COE_WRITE**

funkce	**ECAT_COE_READ**	**Čtení registru z EtherCAT periferie typu CoE**
	ECAT_COE_WRITE	**Zápis do registru EtherCAT periferie typu CoE**
syntax	**ECAT_COE_READ**	**Master, Slave, Index, Subindex, Val, Len**
	ECAT_COE_WRITE	**Master, Slave, Index, Subindex, Val, Len**
1.parametr	**„Master“**	**ID Mastra**
2.parametr	**„Slave“**	**ID Slave**
3.parametr	**„Index“**	**Index pro SDO (WORD)**
4.parametr	**„Subindex“**	**Subindex pro SDO (BYTE)**
5.parametr	**„Val“**	**pointer nebo název proměnné pro data**
6.parametr	**„Len“**	**název proměnné pro uložení délky dat (DWORD)**

parametr	název	význam	typ
1.	**Master**	ID EtherCAT Mastra (0,1,..). Pokud je použit jen jeden, tak hodnota 0.	Byte
2.	**Slave**	ID EtherCAT Slave (0,1,..). Adresa „Slave“ je v pořadí jak jsou připojeny na ETHERNET.	Word
3.	**Index**	Index pro adresaci registru v „Slave“ typu SDO	Word
4.	**Subindex**	SubIndex SDO registru v „Slave“	Byte
5.	**Val**	Název datové proměnné nebo pole pro významová data. Parametr se předává instrukci odkazem (ne hodnotou).	Data
6.	**Len**	Název datové proměnné pro uložení délky dat typu DWORD. Parametr se předává instrukci odkazem (ne hodnotou).	Data

Návratové hodnoty instrukcí:

Instrukce se musí používat v mechanizmech a jsou typu „EX“.
Vrácené datové hodnoty se zapisují přímo do datové proměnné **„Val“** a **„Len“** .
Datová proměnná **„Val“** musí mít rezervovaný dostatečný prostor pro zápis požadovaných hodnot.
Datová proměnná **„Len**“ pro délku slouží jak pro vstup požadované délky dat, tak jako návratovou hodnotu skutečné délky načtených dat.
Všechny instrukce se mohou volat průchodově a mají návratové hodnoty v registrech:

```
RLO=0,      DR=0  ....... stav čekání na dokončení operace
RLO=1,      DR=0  ....... operace dokončena bez chyb
RLO=1,      DR<>0 ....... operace dokončena, ale při výkonu vznikla chyba
```

Příklad:
Čtení statusu CoE ze Slave 0:

```
bEcatData:        DS    4           ;data
dwEcatDataLen:    DS    4           ;pro uložení délky


MECH_BEGIN M_COE_READ
      LOD         CNST.2
      STO         dwEcatDataLen
      ECAT_COE_READ 0,0, 6041h, 0, bEcatData, dwEcatDataLen
      EX0
      EQ          CNST.0
      JL0         LBL_COE_ERR       ;Error
      ...
MECH_END M_COE_READ
```

26.4.2 Čtení a zápis parametru typu SoE

Pro komunikaci s EtherCAT periferií se využívá komunikační profil „SERCOS“ (SoE = „Servo Profile over EtherCAT“) podle normy: „IEC 61491“. Využívají se komunikační objekty „IDN“ typu „S“ a „P“.

Pro SERCOS komunikaci byly zavedeny instrukce pro čtení a zápis. Jedná se o instrukce, které mohou být použity jen v mechanizmech, protože se vždy čeká na výsledek komunikace.

instrukce	**ECAT_SOE_READ** **ECAT_SOE_WRITE**

```
funkce      ECAT_SOE_READ     Čtení registru z EtherCAT periferie typu SoE
            ECAT_SOE_WRITE    Zápis do registru EtherCAT periferie typu SoE

syntax      ECAT_SOE_READ     Master, Slave, DrNo, ElemFl, IDN, Val, Len
            ECAT_SOE_WRITE    Master, Slave, DrNo, ElemFl, IDN, Val, Len

1.parametr  „Master“          ID Mastra
2.parametr  „Slave“           ID Slave
3.parametr  „DrNo“            DriveNo (BYTE)
4.parametr  „ElemFl“          název proměnné pro uložení ElementFlags (BYTE)
5.parametr  „IDN“             IDN (WORD)
6.parametr  „Val“             pointer nebo název proměnné pro data
7.parametr  „Len“             název proměnné pro uložení délky dat (DWORD)
```

parametr	název	význam	typ
1.	**Master**	ID EtherCAT Mastra (0,1,..). Pokud je použit jen jeden, tak hodnota 0.	Byte
2.	**Slave**	ID EtherCAT Slave (0,1,..). Adresa „Slave“ je v pořadí jak jsou připojeny Slave na ETHERNET.	Word
3.	**DrNo**	DriveNo (číslo zařízení)	Byte
4.	**ElemFl**	Název datové proměnné pro uložení ElementFlags (příznaky elementu). Parametr se předává instrukci odkazem (ne hodnotou).	Byte
5.	**IDN**	IDN registru	Word
6.	**Val**	Název datové proměnné nebo pole pro významová data. Parametr se předává instrukci odkazem (ne hodnotou).	Data
7.	**Len**	Název datové proměnné pro uložení délky dat typu DWORD. Parametr se předává instrukci odkazem (ne hodnotou).	Data

Návratové hodnoty instrukcí:

Instrukce se musí používat v mechanizmech a jsou typu „EX“.
Vrácené datové hodnoty se zapisují přímo do datové proměnné **„Val“** a **„Len“** .
Datová proměnná **„ElemFl“**, slouží jak pro vstup, tak pro výstup ElementFlags příslušného registru.
Datová proměnná **„Val“** musí mít rezervovaný dostatečný prostor pro zápis požadovaných hodnot.
Datová proměnná **„Len“** pro délku slouží jak pro vstup požadované délky dat, tak jako návratová hodnota skutečné délky načtených dat.

Všechny instrukce se mohou volat průchodově a mají návratové hodnoty v registrech:

```
RLO=0,      DR=0  ....... stav čekání na dokončení operace
RLO=1,      DR=0  ....... operace dokončena bez chyb
RLO=1,      DR<>0 ....... operace dokončena, ale při výkonu vznikla chyba
```

Příklad:
Nulování chyb SoE v Slave 0, na IDN S-0-0099

```
bEcatData:        DS    4           ;data
dwEcatDataLen:    DS    4           ;pro uložení délky
bEcatElem:        DS    1           ;pro ElementFlags


MECH_BEGIN M_SOE_CLEAR_ERR
      LOD         cnst.40h
      STO         bEcatElem         ;ElementFlags = „VAL“
      LOD         CNST.3
      STO         DWRD.bEcatData    ;data = 3
      LOD         CNST.2
      sto         dwEcatDataLen     ;délka
      ECAT_SOE_WRITE    0,0,0,bEcatElem,99,bEcatData,dwEcatDataLen
      EX0
      JL1         LBL_SOE_ERR       ;Error
MECH_END M_SOE_CLEAR_ERR
```

26.4.3 Zasílání příkazů pro EtherCAT Mastra

Pro zasílání příkazů pro EtherCAT Mastra z PLC programu slouží instrukce ECAT_COMMAND. Jedná se o instrukci, která může být použita jen v mechanizmech, protože se vždy čeká na výsledek komunikace.

instrukce	**ECAT_COMMAND**

```
funkce      ECAT_COMMAND      Příkaz pro EtherCAT Mastra


syntax      ECAT_COMMAND      Master, Slave, Cmd
            ECAT_COMMAND      Master, Slave, Cmd [, Val, Len ]

1.parametr  „Master"          ID Mastra
2.parametr  „Slave"           ID Slave
3.parametr  „Cmd"             Příkaz podle výčtu ECATCOMMAND (WORD)
4.parametr  „Val"             pointer nebo název proměnné pro data
5.parametr  „Len"             název proměnné pro uložení délky dat (DWORD)
```

parametr	název	význam	typ
1.	**Master**	ID EtherCAT Mastra (0,1,..). Pokud je použit jen jeden, tak hodnota 0.	Byte
2.	**Slave**	ID EtherCAT Slave (0,1,..). Adresa „Slave" je v pořadí jak jsou připojeny na ETHERNET.	Word
3.	**Cmd**	Požadovaný příkaz podle výčtu ECATCOMMAND	Word
4.	**Val**	Zadává se jen u příkazů, které jej vyžadují. Název datové proměnné nebo pole pro významová data. Parametr se předává instrukci odkazem (ne hodnotou).	Data
5.	**Len**	Zadává se jen u příkazů, které jej vyžadují Název datové proměnné pro uložení délky dat typu DWORD. Parametr se předává instrukci odkazem (ne hodnotou).	Data

Návratové hodnoty instrukcí:

Instrukce se musí používat v mechanizmech a jsou typu „EX".
Vrácené datové hodnoty se zapisují přímo do datové proměnné **„Val"** a **„Len"** .
Datová proměnná **„Val"** musí mít rezervovaný dostatečný prostor pro zápis požadovaných hodnot.
Datová proměnná **„Len"** pro délku slouží jak pro vstup požadované délky dat, tak jako návratovou hodnotu skutečné délky načtených dat.
Všechny instrukce se mohou volat průchodově a mají návratové hodnoty v registrech:

```
RLO=0,      DR=0  ....... stav čekání na dokončení operace
RLO=1,      DR=0  ....... operace dokončena bez chyb
RLO=1,      DR<>0 ....... operace dokončena, ale při výkonu vznikla chyba
```

Přehled příkazů ECATCOMMAND

Symbol příkazu	Hodnota	Popis
ECATCMD_MASTER_BOOTSTRAP	3	Žádost o přechod Mastra do stavu BOOTSTRAP
ECATCMD_MASTER_INIT	1	Žádost o přechod Mastra do stavu INIT
ECATCMD_MASTER_PREOPERATIONAL	2	Žádost o přechod Mastra do stavu PRE-OPERATIONAL
ECATCMD_MASTER_SAFEOPERATIONAL	4	Žádost o přechod Mastra do stavu SAFE-OPERATIONAL
ECATCMD_MASTER_OPERATIONAL	8	Žádost o přechod Mastra do stavu OPERATIONAL
ECATCMD_SLAVE_BOOTSTRAP	13	Žádost o přechod jednoho Slave do stavu BOOTSTRAP
ECATCMD_SLAVE_INIT	11	Žádost o přechod jednoho Slave do stavu INIT
ECATCMD_SLAVE_PREOPERATIONAL	12	Žádost o přechod jednoho Slave do stavu PRE-OPERATIONAL
ECATCMD_SLAVE_SAFEOPERATIONAL	14	Žádost o přechod jednoho Slave do stavu SAFE-OPERATIONAL
ECATCMD_SLAVE_OPERATIONAL	18	Žádost o přechod jednoho Slave do stavu OPERATIONAL
ECATCMD_MASTER_STATE	20	Žádost o zjištění aktuálního stavu Mastru. Příkaz musí mít parametry pro návratovou hodnotu (Val, Len).
ECATCMD_SLAVE_STATE	21	Žádost o zjištění aktuálního stavu jednoho Slave. Příkaz musí mít parametry pro návratovou hodnotu (Val, Len).

Příklad:

```
Přechod 2.EtherCAT Slave (ID=1) do stavu SAFE-OPERATIONAL a zjištění jeho
aktuálního stavu

bEcatData:        DS    8           ;data
dwEcatDataLen:    DS    4           ;pro uložení délky
dwCmdResult:      DS    4           ;pro test chyby


MECH_BEGIN M_ECAT_SLAVE
      ECAT_COMMAND      0,1,ECATCMD_SLAVE_SAFEOPERATIONAL
      EX0
      STO               dwCmdResult
      ECAT_COMMAND      0,1,ECATCMD_SLAVE_STATE,bEcatData,dwEcatDataLen
      EX0
      STO               dwCmdResult
      LOD               BYTE.bEcatData
      EQ                ECAT_SAFEOPERATIONAL
      …
MECH_END M_ECAT_SLAVE
```

17

17. PLC TABULKY

PLC program může používat pro různé účely PLC tabulky. Pro práci s PLC tabulkami slouží speciální sada instrukcí.
Data z PLC tabulky se používají pro zpracování v PLC, mohou mít přímé využití v NC programu nebo se přes sdílenou paměť PLC „SA" (viz Sdílená paměť pro PLC program) uplatní v různých dialogových oknech.

17.1 Definice struktury PLC tabulky

17.1.1 Soubor pro definici struktury PLC tabulky

Definice struktury PLC tabulky se provede pomocí definičního souboru **„PlcTDef**", který je v XML tvaru. Pro úplnost na tomto místě návodu uvedeme základy pro způsob definice tabulky.

element **PLCTableDef**				**definice struktury PLC tabulky**	
	element **Definition**			**definice struktury PLC tabulky**	
		element **DefinitionID**		**ID definice tabulky**	
				Abcd	Název PLC tabulky
		element **ColsCount**		**počet sloupců**	
				1,2,..	Počet sloupců PLC tabulky
		element **Col**		**sloupec tabulky**	
			atribut **No**	**číslo sloupce tabulky**	
				0,1,2..	pořadové číslo sloupce tabulky (od nuly)
			element **ColID**	**ID sloupce tabulky**	
				Abcd	Název sloupce tabulky
			element **ColType**	**typ dat pro daný sloupec tabulky**	
				REAL	reálná data
				INT	celočíselná data (DWORD)
				STRING	textový řetězec
				BINARY	binární řetězec

Příklad:

Začátek definice tabulky materiálu, která má 6 sloupců. Kompletní definice je v souboru „Sample.PlcTDef".

```
<PLCTableDef>
      <Definition>
            <DefinitionID>Materials</DefinitionID>
            <ColsCount>6</ColsCount>
            <Col No="0">                  <!-- Material thickness (mm) -->
                  <ColID>MaterialThickness</ColID>
                  <ColType>REAL</ColType>
            </Col>
            <Col No="1">                  <!-- Feed (mm/min) -->
                  <ColID>Feed</ColID>
                  <ColType>REAL</ColType>
            </Col>
            ....
```

17.1.2 Registrace definičních souborů PLC tabulek

CNC systém musí mít k definičnímu souboru tabulky přístup, proto se musí definice struktury tabulky zaregistrovat v registrech Windows. V registrech se uvede název definičního souboru pod klíčovými slovy PlcTableDef0, PlcTableDef1,.. (podle čísla tabulky).

Tuto registraci se nedoporučuje provést „ručně" přímým zápisem do registrů, protože PLC tabulka musí patřit do celkového projektu PLC. Překladem PLC v prostředí Wintechnolu vznikne Setup, který musí způsobit zaregistrování definice PLC tabulky. Tím je zaručena také opakovatelnost a obnova konkrétní aplikace systému na daný stroj. Zaregistrování se provede pomocí skriptu který vytváří Setup: „PLC.nsi".

Příklad:

Umístění definice 1. PLC tabulky „Sample.PlcTDef" v registrech Windows:

```
HKLM\Software\MEFI\WinCNC\Machine\PLC\Tables\PlcTableDef0 Sample.PlcTDef
```

Příkaz pro zaregistrování při Setupu v souboru skriptu „PLC.nsi":

```
Function InstallPlcConfig

WriteRegStr HKLM "Software\MEFI\WinCNC\Machine\PLC\Tables" "PlcTableDef0" ~
      "Sample.PlcTDef"

FunctionEnd
```

(znak „~" znamená pokračování řádku – ve skutečnosti řádek nesmí být rozdělen)

17.2 Data PLC tabulky

17.2.1 Soubor pro uložení dat PLC tabulky

Soubor s konkrétními daty PLC tabulky je také v XML tvaru. Obsluha ale nemusí s formátem XML přijít do styku. Pro úplnost na tomto místě návodu uvedeme základy pro uložení dat tabulky.

element **PLCTable**				**PLC tabulka**	
	element **Definition**			**definice struktury tabulky**	
		element **DefinitionID**		**ID definice tabulky**	
				`Abcd`	Název PLC tabulky (musí souhlasit s ID definice tabulky)
		element **LinesCount**		**počet řádků**	
				`1,2,..`	Aktuální počet řádků PLC tabulky
	element **Line**			**řádek tabulky**	
		atribut **No**		**číslo řádku tabulky**	
				`0,1,2..`	pořadové číslo řádku tabulky (od nuly)
		element **Col**		**sloupec tabulky**	
			atribut **ColID**	**ID sloupce tabulky**	
				`Abcd`	Název sloupce tabulky
				data (obsah elementu Col)	
				`xxx`	data jednoho prvku

Příklad:

Začátek PLC tabulky z předchozího příkladu. Kompletní definice je v souboru „`Sample1.PlcT`".

```
<PLCTable>
      <Definition>
            <DefinitionID>Materials</DefinitionID>
            <LinesCount>5</LinesCount>
      </Definition>
      <Line No="0">
            <Col ColID="MaterialThickness">1</Col>
            <Col ColID="Feed">2000</Col>
            <Col ColID="RadiusComp">1.0</Col>
            <Col ColID="PerforationTime">3</Col>
            <Col ColID="LinAccel">100.200</Col>
            <Col ColID="ParabAccel">123.567</Col>
      </Line>
      <Line No="1">
            <Col ColID="MaterialThickness">2</Col>
            <Col ColID="Feed">1000</Col>
            <Col ColID="RadiusComp">1.1</Col>
            <Col ColID="PerforationTime">5</Col>
```

17.2.2 Registrace PLC tabulek

CNC systém musí mít k datům souboru tabulky přístup, proto se musí tabulka zaregistrovat v registrech Windows. V registrech se uvede název souboru PLC tabulky pod klíčovými slovy `PlcTable0`, `PlcTable1`,.. (podle čísla tabulky).

Tuto registraci se nedoporučuje provést „ručně“ přímým zápisem do registrů, protože PLC tabulka musí patřit do celkového projektu PLC. Překladem PLC v prostředí Wintechnolu vznikne Setup, který musí způsobit zaregistrování PLC tabulky. Tím je zaručena také opakovatelnost a obnova konkrétní aplikace systému na daný stroj. Zaregistrování se provede pomocí skriptu který vytváří Setup: „`PLC.nsi`“.

Příklad:

Umístění dat 1. PLC tabulky „`Sample1.PlcT`“ v registrech Windows:

```
HKLM\Software\MEFI\WinCNC\Machine\PLC\Tables\PlcTable0 Sample1.PlcT
```

Příkaz pro zaregistrování při Setupu v souboru skriptu „`PLC.nsi`“:

```
Function InstallPlcConfig

WriteRegStr HKLM  "Software\MEFI\WinCNC\Machine\PLC\Tables"  "PlcTable0"  ~
      "Sample1.PlcT"

FunctionEnd
```

(znak „~“ znamená pokračování řádku – ve skutečnosti řádek nesmí být rozdělen)

17.3 Editor PLC tabulek

17.3.1 Tvorba dialogového okna pro editor tabulek

Vizualizaci a editaci PLC tabulek neprovádí automaticky CNC systém. PLC tabulky mohou mít velkou rozmanitost použití od čeho se odvíjí i rozmanitost tvaru a forem dialogových oken. Dialogová okna pro editaci PLC tabulek si proto musí navrhnout návrhář PLC programu a systém jen poskytuje dispozice pro tento návrh. Proto také dialogová okna pro PLC tabulky patří do celkového projektu PLC.

Dialogová okna se navrhují podobně jako stránky pro webové aplikace. Jsou v HTML formátu, který je obohacen o elementy CNC systému, které například zabezpečí propojení dat se systémem. Při návrhu se doporučuje používat kaskádové styly HTML, které zabezpečí jednotnou vizáž všech oken.

Pro úplnost na tomto místě návodu uvedeme základy pro tvorbu dialogu pro PLC tabulky.

Pro zobrazení obsahu PLC tabulky se používá standardní element `TABLE` doplněný o speciální atributy a pro editaci buňky se používá standardní element `INPUT` také doplněný o speciální atributy.

element **TABLE**					**tabulka (HTML)**	
	atribut **id**				**klíčové slovo pro CNC**	
					`PlcTTableVieW`	Vykreslení PLC tabulky CNC systémem
	element **THEAD**				**označení řádků v hlavičce tabulky (HTML)**	
		element **TR**			**řádek tabulky (HTML)**	
			element **TD**		**buňka tabulky (HTML)**	
				atribut **PlcTColID**	**ID sloupce tabulky**	
					`Abcd`	Název sloupce tabulky (podle definice)
				atribut **ClickAction**	**klíčové slovo pro CNC**	
					`EditedLineSet`	Editovatelná položka
					data (obsah elementu TD)	
					`Abcd`	Nadpis pro sloupec tabulky

element **INPUT**			
	atribut **name**	**klíčové slovo pro CNC**	
		`PlcTValueEdit`	Editovatelná položka
	atribut **PlcTColID**	**ID sloupce tabulky**	
		`Abcd`	Název sloupce tabulky (podle definice)
	atribut **DdxOptions**	**klíčové slovo pro CNC**	
		`NumberWidth: xx`	celkový počet cifer
		`NumberPrecision: xx`	počet desetinných míst

Příklad:

Uvedeme části HTML kódu pro zobrazení a editaci PLC tabulky. Celý příklad je uveden v souboru „`SamplePlcTab.html`".

Definice tabulky:

```
<TABLE id="PlcTTableVieW" width="100%" cellspacing="0" class="PlcTable">
  <THEAD>
    <TR>
      <TD PlcTColID="MaterialThickness"
          ClickAction="EditedLineSet">Tlouštka<BR>materiálu</TD>
      <TD PlcTColID="Feed" ClickAction="EditedLineSet" >Rychlost<BR></TD>
      ....
```

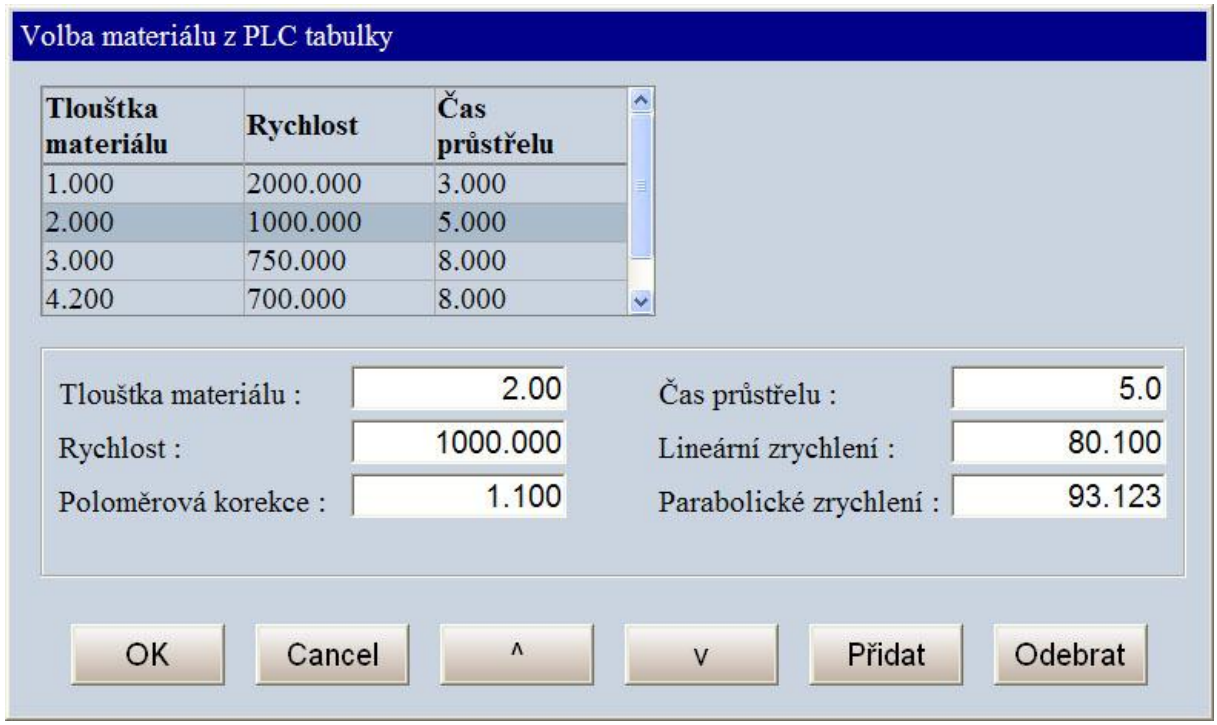
Definice editačních polí:

```
<DIV id="MaterialThickness_Lbl" class="LabelMedium">Tloušťka :</DIV>
<INPUT id="MaterialThickness_Val" type="text" size="10" class="EditMedium"
       name="PlcTValueEdit" PlcTColID="MaterialThickness"
       DdxOptions="NumberWidth: 0; NumberPrecision: 2">

<DIV id="Feed_Lbl" class="LabelMedium">Rychlost :</DIV>
<INPUT id="Feed_Val" type="text" size="10" class="EditMedium"
       name="PlcTValueEdit" PlcTColID="Feed"
       DdxOptions="NumberWidth: 0; NumberPrecision: 3">
.....
```

Příklad umístění pomocí kaskádových stylů:

```
<STYLE type="text/css">

<!-- Prvky v poli EditArea -->
#MaterialThickness_Lbl  {position: absolute; top: 18px; left: 10px;}
#MaterialThickness_Val  {position: absolute; top: 10px; left: 200px;}
#Feed_Lbl               {position: absolute; top: 53px; left: 10px;}
#Feed_Val               {position: absolute; top: 45px; left: 200px;}
......

</STYLE>
```

Dialog pro editaci tabulky z příkladu. V příkladu se zobrazují jen 3 vybrané sloupce tabulky, ale po vybrání řádku se v editačním poli zobrazují data ze všech 6 sloupců. Všechna data možno editovat.



Dialog v příkladu má použita také tlačítka pro ovládání editace tabulky. Pro úplnost zde uvedeme její zápis v HTML tvaru:

```
<!-- Buttons -->
<BUTTON id="EditedLinePrev" class="Button">^</BUTTON>
<BUTTON id="EditedLineNext" class="Button">v</BUTTON>
<BUTTON id="AddLine" class="Button">Přidat</BUTTON>
<BUTTON id="RemoveLine" class="Button">Odebrat</BUTTON>
<BUTTON id="OK" class="Button">OK</BUTTON>
<BUTTON id="Cancel" class="Button">Cancel</BUTTON>
```

17.3.2 Registrace editoru PLC tabulek

CNC systém musí mít k dialogu přístup, proto se musí HTML soubor zaregistrovat v registrech Windows.

Tuto registraci se nedoporučuje provést „ručně“ přímým zápisem do registrů, protože editor PLC tabulky musí patřit do celkového projektu PLC. Překladem PLC v prostředí Wintechnolu vznikne Setup, který musí způsobit zaregistrování editoru PLC tabulky. Zaregistrování se provede pomocí skriptu který vytváří Setup: „`PLC.nsi`“.

Příklad:

Příkaz pro zaregistrování při Setupu v souboru skriptu „`PLC.nsi`“:

```
Function InstallPlcConfig

WriteRegStr HKLM ~
Software\MEFI\WinCNC\Machine\UserInterface\Dialogs\SamplePlcTab" ~
"Library" "StdPlugins"

WriteRegStr HKLM ~
Software\MEFI\WinCNC\Machine\UserInterface\Dialogs\SamplePlcTab" ~
"Type" " PlcTableEditor"

WriteRegStr HKLM ~
Software\MEFI\WinCNC\Machine\UserInterface\Dialogs\SamplePlcTab" ~
" HtmlFile" "SamplePlcTab.html"

FunctionEnd
```

(znak „~“ znamená pokračování řádku – ve skutečnosti řádek nesmí být rozdělen)

17.3.3 Aktivace editoru PLC tabulky

Zobrazení dialogu editoru PLC tabulky možno provést například pomocí softwarového menu nebo pomocí libovolného tlačítka panelu.

Přidání softwarového tlačítka do menu se provede pomocí elementu „`Dialog`" v příslušném souboru s definicí softwarového menu typu „`SoftMenu`".

Příklad:

Příklad pro přidání tlačítka „`Volba materiálu`" do menu technologie v souboru „`TechnolgyCSY.SoftMenu`"

```
<SoftMenuItem>
      <Text>Volba<Br/>materiálu</Text>
      <Dialog>SamplePlcTab</Dialog>
</SoftMenuItem>
```

Zobrazení dialogu na základě stisku tlačítka se může provést například v definičním souboru pro technologická tlačítka pomocí elementu „`Dialog`" v souboru typu „`KbdConfig`".

Příklad:

Příklad pro aktivaci editoru přímo z tlačítka panelu v souboru „`TechnologyArea.KbdConfig`"

```
<KeyConfig ScanCode="0x6D" Type="Normal">
<Dialog>SamplePlcTab</Dialog></KeyConfig>      <!-- Editor PLC tabulky -->
```

17.4 Tabulkové operace v PLC programu

17.4.1 Čtení a zápis do PLC tabulky

Instrukce pro tabulkové operace jsou víceprůchodové a pro synchronizaci přístupu k datům používají vlastní mutex. Proto všechny dále uvedené instrukce se mohou používat jen v mechanizmech (viz „Logické sekvenční celky“)

instrukce	**PLCT_GET_INT** **PLCT_GET_REAL** **PLCT_GET_STR** **PLCT_GET_BIN** **PLCT_SET_INT** **PLCT_SET_REAL** **PLCT_SET_STR** **PLCT_SET_BIN**

funkce	**PLCT_GET_INT**	**Načtení celočíselné hodnoty z buňky tabulky**
	PLCT_GET_REAL	**Načtení reálné hodnoty z buňky tabulky**
	PLCT_GET_STR	**Načtení textového řetězce z buňky tabulky**
	PLCT_GET_BIN	**Načtení binárního řetězce z buňky tabulky**
	PLCT_SET_INT	**Zapsání celočíselné hodnoty do buňky tabulky**
	PLCT_SET_REAL	**Zapsání reálné hodnoty do buňky tabulky**
	PLCT_SET_STR	**Zapsání textového řetězce do buňky tabulky**
	PLCT_SET_BIN	**Zapsání binárního řetězce do buňky tabulky**
syntax	**PLCT_GET_xx**	**TabIdx, Line, Col, Val**
	PLCT_GET_xx	**TabIdx, Line, Col, Poin**
	PLCT_SET_xx	**TabIdx, Line, Col, Val**
	PLCT_SET_xx	**TabIdx, Line, Col, Poin**
1.parametr	**„TabIdx“**	**index tabulky**
2.parametr	**„Line“**	**řádek v tabulce (0,1,..)**
3.parametr	**„Col“**	**slopec v tabulce (0,1,..)**
4.parametr	**„Val,Poin“**	**pointer nebo název proměnné**

Význam parametrů instrukcí:

parametr	název	význam	typ
1.	**TabIdx**	Index tabulky (0,1,..)	Byte
2.	**Line**	Řádek v tabulce (0,1,..)	Word
3.	**Col**	Sloupec v tabulce (0,1,..)	Word
4.	**Poin**	Náveští u řetězce definovaného instrukcí "str" Parametr může mít zadán offset v řetězci (+xx).	Pointer
	Val	Název datové proměnné typu (BYTE,WORD,DWRD,..)	Data

Návratové hodnoty instrukcí:

Instrukce se musí používat v mechanizmech a jsou typu „EX“.
Vrácené datové hodnoty z tabulky se zapisují do řetězce na který ukazuje parametr **„Poin“,** nebo přímo do datové proměnné **„Val“**.
Všechny instrukce se mohou volat průchodově a mají návratové hodnoty:

```
RLO=0,      DR=0  ....... stav čekání na dokončení operace
RLO=1,      DR=0  ....... operace dokončena bez chyb
RLO=1,      DR<>0 ....... operace dokončena, ale při výkonu vznikla chyba
```

Příklady:

```
ColTxt: str 20
PokTxt: str 20,'Novy text'
Bun4:   DS  4
BunReal:DS  8
```

Čtení DWORD z tabulky do řetězce `ColTxt` (Index tabulky=0, řádek=2, sloupec=1)

```
PLCT_GET_INT  0,2,1,ColTxt
EX0
EQ            cnst.0
JL0           TabError
```

Čtení DWORD z tabulky do buňky BUN4 (Index tabulky=0, řádek=2, sloupec=1)

```
PLCT_GET_INT  0,2,1,BUN4
EX0
EQ            cnst.0
jl0           TabError
```

Zápis řetězce do tabulky (Index tabulky=0, řádek=2, sloupec=0)

```
PLCT_SET_STR  0,2,0,PokTxt
EX0
EQ            cnst.0
JL0           TabError
```

Čtení reálné hodnoty z tabulky podle ID sloupce (Index tabulky=0, řádek=2, sloupec=5)

```
PLCT_GET_REAL  0,2,5,BunReal
EX0
EQ             cnst.0
JL0            TabError
```

17.4.2 Zjištění indexu a ID sloupce

instrukce	PLCT_COL_INDEX PLCT_COL_ID

```
funkce      PLCT_COL_INDEX  Zjištění indexu sloupce podle zadaného ID sloupce
            PLCT_COL_ID     Zjištění ID sloupce podle zadaného indexu sloupce


syntax      PLCT_COL_INDEX  TabIdx, 'TEXT', Val
            PLCT_COL_ID     TabIdx, Poin

1.parametr  „TabIdx"        index tabulky
2.parametr  „TEXT"          textový řetězec pro ID sloupce
3.parametr  „Val"           název datové proměnné, kam se zapíše index
```

Popis funkce

Index sloupce je celočíselná hodnota (DWORD).
ID sloupce je textový řetězec (STRING).

Instrukce PLCT_COL_INDEX nastaví podle ID sloupce (textový řetězec s názvem sloupce), který je uveden jako 2.parametr „TEXT", celočíselnou hodnotu indexu sloupce (0,1,2..) do proměnné „Val".

Instrukce PLCT_COL_ID nastaví do pointru „Poin" ID sloupce (textový řetězec s názvem sloupce) podle indexu sloupce, který je předem nastaven v „Poin".

Doporučuje se používat instrukce PLCT_COL_INDEX na zjištění skutečného indexu sloupce tabulky. Tím se dosáhne toho, že PLC program nebude závislý na struktuře tabulky.

Význam parametrů instrukcí:

parametr	název	význam	typ
1.	**TabIdx**	Index tabulky (0,1,..)	Byte
2.	**Text**	Přímé zadání textu s jménem sloupce (ID sloupce) - text je zadán v apostrofech	řetězec
3.	**Val**	Název datové proměnné, kam se zapíše index sloupce. - typ BYTE, WORD, DWRD	Data

Návratové hodnoty instrukcí:

Instrukce se musí používat v mechanizmech a jsou typu „EX".
Všechny instrukce se mohou volat průchodově a mají návratové hodnoty:

```
RLO=0,      DR=0  ....... stav čekání na dokončení operace
RLO=1,      DR=0  ....... operace dokončena bez chyb
RLO=1,      DR<>0 ....... operace dokončena, ale při výkonu vznikla chyba
```

Příklady:

```
wCOL:    DS  2
R_FEED:  DS  8    ;reálná hodnota rychlosti
```

Čtení reálné hodnoty z tabulky podle ID sloupce (2.řádek)

```
      PLCT_COL_INDEX    0,'Feed',wCOL     ;zjistí index sloupce
      EX0
      EQ                cnst.0
      JL0               TabError1         ;nenašel se sloupec ID='Feed'
      PLCT_GET_REAL     0,2,wCOL,R_FEED
      EX0
      EQ                cnst.0
      JL0               TabError2         ;chyba při čtení dat z tabulky
```

Zjištění ID sloupce podle Indexu sloupce

```
      LOD               cnst.3
      STO               word.IDX_COL
      PLCT_COL_ID       0,IDX_COL         ;zjistí ID sloupce
      EX0
      EQ                cnst.0
      JL0               TabError
```

17.4.3 Zjištění datového typu sloupce

instrukce	**PLCT_COL_TYPE**

```
funkce      PLCT_COL_TYPE   Zjištění datového typu pro zadaný sloupec


syntax      PLCT_COL_TYPE   TabIdx, Col, Val

1.parametr  „TabIdx"        index tabulky
2.parametr  „Col"           slopec v tabulce (0,1,..)
3.parametr  „Val"           název proměnné
```

Instrukce zapíše do proměnné „Val" datový typ zadaného sloupce.

Přehled datových typů pro sloupce tabulky

```
PlcTabType_Int    ....  1 .... Double-wordová hodnota DWRD
PlcTabType_Real   ....  2 .... Reálná hodnata QWORD
PlcTabType_Str    ....  3 .... Textový řetězec
PlcTabType_Bin    ....  4 .... Binární řetězec
```

Návratové hodnoty instrukce:

Instrukce se musí používat v mechanizmech a jsou typu „EX“.

```
RLO=0,      DR=0  ....... stav čekání na dokončení operace
RLO=1,      DR=0  ....... operace dokončena bez chyb
RLO=1,      DR<>0 ....... operace dokončena, ale při výkonu vznikla chyba
```

Příklad:

```
BUN1: DS 1
```

Zjištění typu sloupce do buňky BUN1 (Index tabulky=0, sloupec=3)

```
      PLCT_COL_TYPE  0,3,BUN1
      EX0
      EQ             cnst.0
      JL0            TabError
      LOD            BUN1                ;Typ=4 (PlcTabType_Bin)
```

17.4.4 Vyvolený řádek tabulky

instrukce	**PLCT_GET_SELLINE** **PLCT_SET_SELLINE**

funkce	**PLCT_GET_SELLINE**	**Zjištění vyvoleného řádku**
	PLCT_SET_SELLINE	**Nastavení vyvoleného řádku**
syntax	**PLCT_GET_SELLINE**	**TabIdx, Val**
	PLCT_SET_SELLINE	**TabIdx, Immed**
1.parametr	**„TabIdx“**	**index tabulky**
2.parametr	**„Val,Immed“**	**název proměnné, nebo přímá hodnota**

Návratové hodnoty instrukcí:

Instrukce se musí používat v mechanizmech a jsou typu „EX“.

```
RLO=0,      DR=0  ....... stav čekání na dokončení operace
RLO=1,      DR=0  ....... operace dokončena bez chyb
RLO=1,      DR<>0 ....... operace dokončena, ale při výkonu vznikla chyba
```

Příklady:

```
BUN1: DS 1
```

Zjištění vyvoleného řádku tabulky do BUN1

```
      PLCT_GET_SELLINE  0,BUN1      ;přečte vyvolený řádek
      EX0
      EQ          cnst.0
      JL0         TabError
      LOD         BUN1
```

Nastavení vyvoleného řádku v tabulce

```
      PLCT_SET_SELLINE  0,8         ;nastaví vyvolený řádek 8
      EX0
      EQ          cnst.0
      JL0         TabError
```

17.4.5 Test změny v tabulce

instrukce	**PLCT_CHANGED**

```
funkce      PLCT_CHANGED    Zjištění změny v tabulce


syntax      PLCT_CHANGED    TabIdx, Val

1.parametr  „TabIdx“        index tabulky
2.parametr  „Val“           název proměnné
```

Instrukce nastaví v proměnné „Val“ některou z hodnot pro test změny tabulky.

Návratové hodnoty pro změny v tabulce

```
PlctItemChanged   .... 1  .... Změna prvku tabulky
PlctSelLineChanged ... 2  .... Změna zvoleného řádku
```

Příklad:

Zjištění změny v tabulce

```
      PLCT_CHANGED  0, BUN1
      EX0
      EQ          cnst.0
      JL0         TabError
      LOD         BUN1
      EQ          cnst.1
      JL1         ZmenaPrvkuTabulky       ;Změna prvku
      EQ          cnst.2
      JL1         ZmenaZvolenehoRadku     ;Změna řádku
```

17.4.6 Aktuální počet řádků tabulky

instrukce	**PLCT_GET_LINESCOUNT**

```
funkce      PLCT_GET_LINESCOUNT   Zjištění aktuálního počtu řádků tabulky

syntax      PLCT_GET_LINESCOUNT   TabIdx, Val

1.parametr  „TabIdx"         index tabulky
2.parametr  „Val"            název proměnné
```

Instrukce zapíše do proměnné „Val" aktuální počet řádků tabulky.

Návratové hodnoty instrukce:

Instrukce se musí používat v mechanizmech a jsou typu „EX".

```
RLO=0,      DR=0  ....... stav čekání na dokončení operace
RLO=1,      DR=0  ....... operace dokončena bez chyb
RLO=1,      DR<>0 ....... operace dokončena, ale při výkonu vznikla chyba
```

Příklad:

```
wLineCnt:   DS 2
```

Zjištění aktuálního počtu řádků tabulky (Index tabulky=0)

```
      PLCT_GET_LINESCOUNT  0, wLineCnt
      EX0
      EQ             cnst.0
      JL0            TabError
      LOD            wLineCnt            ;počet řádků
```

17.4.7 Zpracování dat z PLC tabulky

Načtení dat z PLC tabulky se musí provést v rámci mechanizmu a není předem určeno, jak dlouho bude tato operace trvat. PLC program musí být proto navržen tak, aby se vypořádal se situací, že data z PLC tabulky nedostane okamžitě.

Vážná situace může nastat při startu PLC programu, kdy data z PLC tabulky mají vliv například na průchod prvního bloku centrální anulace. V tomto případě se musí v modulu `MODULE_INIT` zavolat mechanizmus pro čtení a zpracování dat z PLC tabulky a na konci tohoto mechanizmu se použije instrukce `MODULE_INIT_FINISHED` (viz „Struktura PLC programu“). Systém tak bude čekat na vykonání mechanizmu čtení a zpracování dat z PLC tabulky a až potom se inicializace systému ukončí a provede se start prvního bloku centrální anulace.

Data z PLC tabulky se používají pro zpracování v PLC, mohou mít přímé využití v NC programu nebo se přes sdílenou paměť PLC „`SA`“ (viz Sdílená paměť pro PLC program) uplatní v různých dialogových oknech.

Příklad:

;Modul inicializace PLC

```
MODULE_INIT
      FL                1, M_TAB_TECHNOL  ;Start mechanizmu pro čtení
                                          ;dat z PLC tabulky
MODULE_INIT_END
```

;Načtení dat z PLC tabulky z vyvoleného řádku podle předchozích příkladů

```
MECH_BEGIN  M_TAB_TECHNOL

      PLCT_GET_SELLINE  0, wLine          ;zjištění vyvoleného řádku
      EX0
      EQ                CNST.0
      JL0               MERR_LINEERROR    ;Nenašel se řádek v tabulce
;~~
      PLCT_COL_INDEX    0,'Feed', wCOL    ;zjistí index podle ID
      EX0
      EQ                CNST.0            ;test chyby
      JL0               MERR_COLERROR     ;Nenašel se sloupec v tabulce
      PLCT_GET_REAL     0,wLine,wCOL,R_FEED
      EX0
      EQ                CNST.0            ;test chyby
      JL0               MERR_DATAERROR    ;Chyba při získání dat z  tabulky
;~~
      PLCT_COL_INDEX    0,'RadiusComp',wCOL  ;zjistí index podle ID
      EX0
      EQ                CNST.0            ;test chyby
      JL0               MERR_COLERROR     ;Nenašel se sloupec v tabulce
      PLCT_GET_REAL     0,wLine,wCOL,R_RADIUSCOMP
      EX0
      EQ                CNST.0            ;test chyby
      JL0               MERR_DATAERROR    ;Chyba při získání dat z tabulky
;~~
```

```
      PLCT_COL_INDEX    0,'PerforationTime',wCOL      ;zjistí index podle ID
      EX0
      EQ                CNST.0            ;test chyby
      JL0               MERR_COLERROR     ;Nenašel se sloupec v tabulce
      PLCT_GET_REAL     0,wLine,wCOL,R_PERFORATIONTIME
      EX0
      EQ                CNST.0            ;test chyby
      JL0               MERR_DATAERROR    ;Chyba při získání dat z tabulky
;~~
      ....


      MODULE_INIT_FINISHED                ;konec inicializace PLC

MECH_END    M_TAB_TECHNOL
```

17.5 Tabulky pro zobrazení sdílených proměnných

17.5.1 Tvorba dialogového okna pro zobrazení sdílených proměnných

Pro lepší přehlednost aktuálních hodnot sdílených proměnných je možné použít zobrazení pomocí tabulky. Používá standardní element `TABLE` doplněný o speciální atributy.

element **TABLE**					**tabulka (HTML)**	
	atribut **CNCType**				**klíčové slovo pro CNC**	
					`SharedVarInfo`	Vykreslení tabulky sdílených proměnných
	atribut **SVIOptions**				**klíčové slovo pro CNC**	
					`Channel: 0`	Číslo suportu, se kterým chceme pracovat
					`VarSource: PLC`	Typ sdílených proměnných (PLC, System, All, ...)
					`Type: Input Output`	Typ proměnných/portů, které chceme zobrazit (Input, Output, AInput, AOutput, Simple)
					`Sort: Port Bit`	Názvy sloupců, podle kterých chceme tabulku seřadit
					`Filter: Connected=1`	Filtrování podle předem definovaných pravidel
	Element **THEAD**				**označení řádků v hlavičce tabulky (HTML)**	
		element **TR**			**řádek tabulky (HTML)**	
			element **TD**		**buňka tabulky (HTML)**	
				atribut **SVIValType**	**Název požadované hodnoty**	
					`VarName`	Hodnota, kterou chceme zobrazit v aktuálním sloupci
				atribut **TDClass**	**Název třídy**	
					`ValueMedium`	Třída pro formátování daného sloupce.
				atribut **ExtraAIPOptions**	**Dodatečné nastavení**	
					`NumberPrecision: 0;` ...	Počet desetinných míst

Příklad:

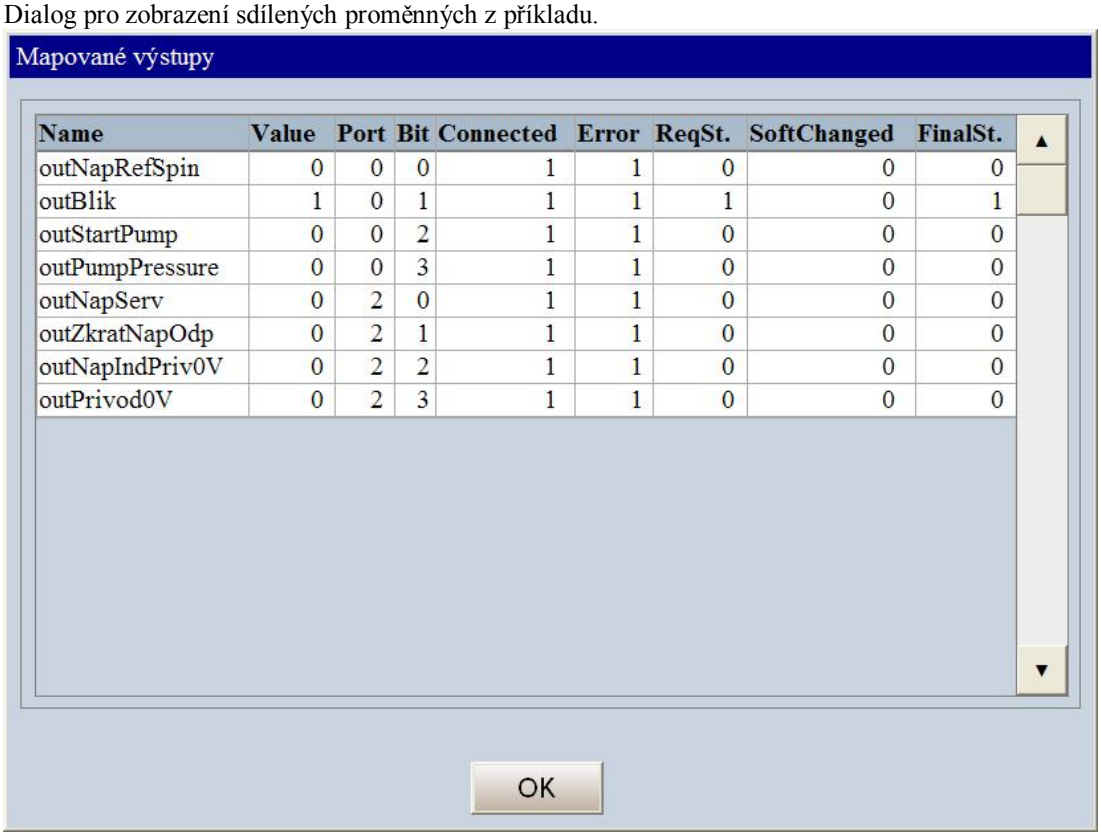
Části HTML kódu pro zobrazení tabulky. Příklad umístění pomocí kaskádových stylů:

```
<STYLE type="text/css">
#SVI_Area   {position: absolute; left:  10px; top:  50px; width: 800px;
                              height: 460px; border: solid 1px gray;}

/* Třídy SVITable ... nastavení šířky jednotlivých sloupců tabulky */
.SVITable_TD_Name             {width: 120px;}
.SVITable_TD_Value            {width:  70px;}
.SVITable_TD_No               {width:  35px;}
.SVITable_TD_Connected        {width:  90px;}
......
</STYLE>
```

Definice tabulky:

```
<TABLE id="PlcTTableVieW" width="100%" cellspacing="0" class="PlcTable">
  <THEAD>
    <TR>
      <TD PlcTColID="MaterialThickness"
          ClickAction="EditedLineSet">Tlouštka<BR>materiálu</TD>
      <TD PlcTColID="Feed" ClickAction="EditedLineSet" >Rychlost<BR></TD>
      ....

<TABLE id="SVI_Area_Table"
            CNCType="SharedVarInfo"
            SVIOptions="Channel: 0;
                        VarSource: PLC;
                        Type: AInput;
                        Sort: Port Bit Name;
                        Filter: Connected=1;"
cellSpacing="0">
  <THEAD>
    <TR>
      <TD SVIValType="VarName"
            TDClass="TextMedium  SVITable_TD_Name"
            ExtraAIPOptions="">Name</TD>

      <TD SVIValType="VarValue"
            TDClass="ValueMedium SVITable_TD_Value"
            ExtraAIPOptions="NumberPrecision: 3;">Value</TD>

    </TR>
  </THEAD>
<TBODY>
</TBODY>
</TABLE>
```

Dialog pro zobrazení sdílených proměnných z příkladu.



Name	Value	Port	Bit	Connected	Error	ReqSt.	SoftChanged	FinalSt.
outNapRefSpin	0	0	0	1	1	0	0	0
outBlik	1	0	1	1	1	1	0	1
outStartPump	0	0	2	1	1	0	0	0
outPumpPressure	0	0	3	1	1	0	0	0
outNapServ	0	2	0	1	1	0	0	0
outZkratNapOdp	0	2	1	1	1	0	0	0
outNapIndPriv0V	0	2	2	1	1	0	0	0
outPrivod0V	0	2	3	1	1	0	0	0

17.6 Učící režim systému

17.6.1 Zaregistrování dialogu pro učící režim

WriteRegStr	HKLM "Software\MEFI\WinCNC\Machine\UserInterface\Dialogs\Teachin"		
	"Library"	"WinCNC"	
	"Type"	**"Teachin"**	
	"HtmlFile"	"Teachin.html"	
WriteRegDWORD	HKLM "Software\MEFI\WinCNC\Machine\UserInterface\Dialogs\Teachin"		
	"Left"	880	…

17.6.2 Možnosti a použití atributu TeachInOptions

element **INPUT**		**HTML element**		
	atribut **TeachInOptions**	**klíčové slovo pro CNC**		
		TypeN	Typ elementu	
			`Fixed`	Parametr ValueN bude přímo zapsán do bloku NCP programu
			`Element`	Parametr ElementIDN udává ID elementu z jehož Value se přečte text co se vloží do NCP
			`RTMVar`	Aktuální hodnota požadované systémové proměnné bude zapsána do souboru
			`PLCVar`	Aktuální hodnota požadované PLC proměnné bude zapsána do souboru
		ValueN `Pouze pro TypeN: Fixed`	Hodnota elementu	
			`řetězec`	Řetězec bude přímo zapsán do bloku NCP programu
		ElementIDN `Pouze pro TypeN: Element`	ID elementu	
			`element ID`	Řetězec obsahuje ID elementu z jehož Value se přečte text co se vloží do NCP
		VarChannel `Pouze pro TypeN: RTMVar,PLCVar`	Číslo CNC kanálu	
			`0`	Výchozí hodnota
		VarName `Pouze pro TypeN: RTMVar,PLCVar`	Název sdílené proměnné	
			`řetězec`	Řetězec obsahuje název sdílené proměnné
		NumberWidth `Pouze pro TypeN: RTMVar,PLCVar`	Pokud je zápis čísla kratší než zadaný počet číslic, doplní se zleva nulami	
			`0`	Výchozí hodnota
		NumberPrecision `Pouze pro TypeN: RTMVar,PLCVar`	Počet desetinných míst v zápisu čísla	
			`3`	Výchozí hodnota
		FilterType	Typ filtru, který se má uplatnit na všechny elementy na stránce. Uplatní se pouze pokud Value="1".	
			`Write`	Budou zpracovány a zapsány pouze elementy, které jsou výslovně uvedeny v parametru FilterValue
		FilterValue	Hodnoty potřebné pro aktuální filtr	
			`řetězec`	Řetězec obsahuje hodnoty potřebné pro aktuální filtr
	atribut **ID**	hodnota `řetězec`	Jedinečný identifikátor elementu, potřebný i pro použití filtrů	

	atribut **Value**	hodnota 0	Element je/není vybrán, takže bude/nebude zpracován a zapsán do NCP programu
	atribut **Type**	hodnota checkbox	Libovolný typ elementu s atributem Value(CheckBox, Text, ...)

17.6.3 Parametry pro nastavení učícího a editačního režimu v elementu BODY

```
<BODY onload="" style="" CNCDirectory="DIR_USER SUBDIR_NCP"
    InitText     = "N0 PROGRAM"     ... Na začátku programu je požadován
                                        text z elementu InitText.
    EditBlock    = "1"              ... Požadavek na zapnutí editačního módu
                                        režimu Teachin.
    BlockCounter = "1"              ... Požadavek zápisu aktuálního čísla
                                        kroku do komentáře bloku.
    BlockText    = "krok!"          ... V komentáři každého bloku je
                                        požadován text z elementu.
    FinalText    = "N ENDPROGRAM"> ... Na konci programu je požadován text
                                        z elementu FinalText.
```

17.6.4 Příklady dialogů učícího režimu

Kompaktní dialog

Vygenerovaný výsledný program

```
N PROGRAM

N "Zápis 1. kroku!
 F400
 G0
 TECHNOLOGY_ON
 X164.341
 Y1900.938
 Z786.309
 O-0.009
 P0.731
```

```
 Q-0.682
 A0.000
 B0.000

N G5 N ENDPROGRAM
```

Rozšířený dialog

Vygenerovaný výsledný program

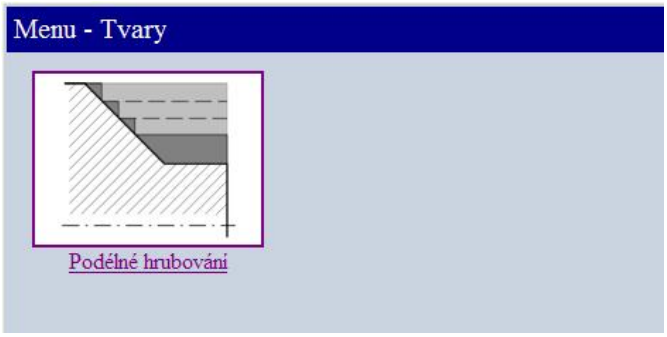
```
N PROGRAM   N G23

N "Zápis 1. kroku!
 G0
 In6
 A164.341
 B1900.938
 C786.309
 U0.000
 V0.000
 W0.000
 O0.000
 P0.683
 Q2.000

N ENDPROGRAM
```

25

25. TVORBA HTML DIALOGŮ PRO PEVNÉ TVARY

25.1 Přidání nového tlačítka do menu

Abychom mohli zpřístupnit nový pevný tvar do systému, je třeba přidat nové tlačítko do Menu. V našem případě se jedná o příčné hrubování. V `Menu - Tvary` (Menu.html) je prozatím pouze podélné hrubování.



Stránku Menu.html tvoří tabulka o čtyřech sloupcích s neviditelnými okraji

```
<table width="100%">
      <tr height="200">
            <td align="center" style="width: 25%;">
            <a href="RoughHorizont1.html">
            <img src="Thumb RoughHorizont1.gif"><br>
            <SPAN LocStrID="IDS_RoughHorizont1">Podélné hrubování</SPAN>
            <br></a><br> </td>
```

Na místo prázdné buňky

```
            <td align="center" style="width: 25%;"></td>
```

Vložíme odkaz na nový tvar

```
            <td align="center" style="width: 25%;">
            <a href="RoughVertical1.html">
            <img src="Thumb RoughVertical1.gif"><br>
            <SPAN LocStrID="IDS_RoughVertical1">Příčné hrubování</SPAN>
            <br></a><br> 
            </td>

            <td align="center" style="width: 25%;"></td>
            <td align="center" style="width: 25%;"></td>
      </tr>
</table>
```

Odkaz na nový tvar obsahuje:

- Název souboru HTML stránky nového tvaru
  `<a href="RoughVertical1.html">`
- Název obrázku miniatury pro tlačítko (použité rozměry 160x120px)
  `<img src="Thumb RoughVertical1.gif">`
- Odkaz na lokalizaci popisku Identifikátor pro nový tvar zvolíme **IDS_RoughVertical1**
  `<SPAN LocStrID="IDS_RoughVertical1">Příčné hrubování</SPAN>`

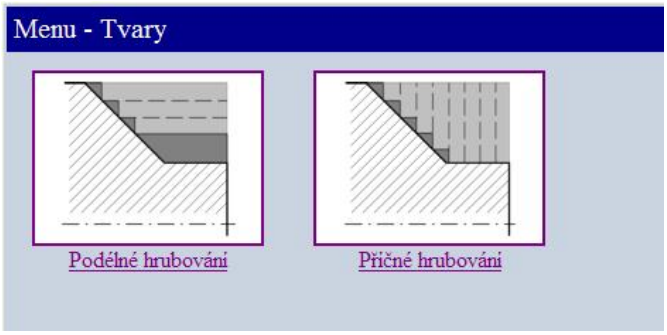
Do sekce **<!-- String tables -->** uvnitř stránky Menu.html je třeba doplnit popisek pro nový tvar v příslušném jazyku.

```
<DIV id="StringTableCSY" class="StringTable">
      <SPAN id="IDS_WindowTitle"    >Menu - Tvary</SPAN>
      <SPAN id="IDS_RoughHorizont1" >Podélné hrubování</SPAN>
      <SPAN id="IDS_RoughVertical1" >Příčné hrubování</SPAN>
</DIV>
```

Příslušný jazyk rozlišuje hodnota identifikátoru například:

- Čeština `id="StringTableCSY"`
- Polština `id="StringTablePLK"`
- Angličtina `id="StringTableENU"`

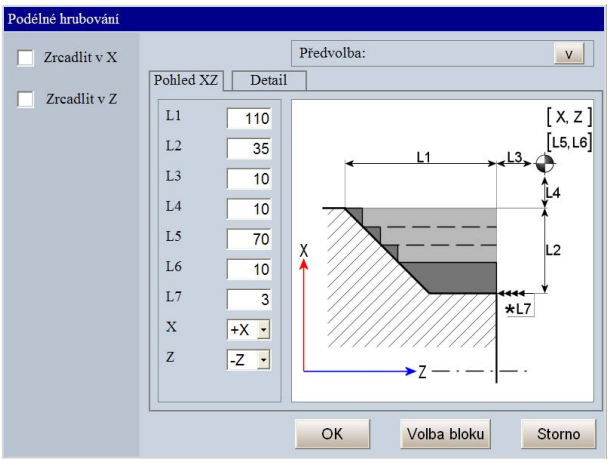
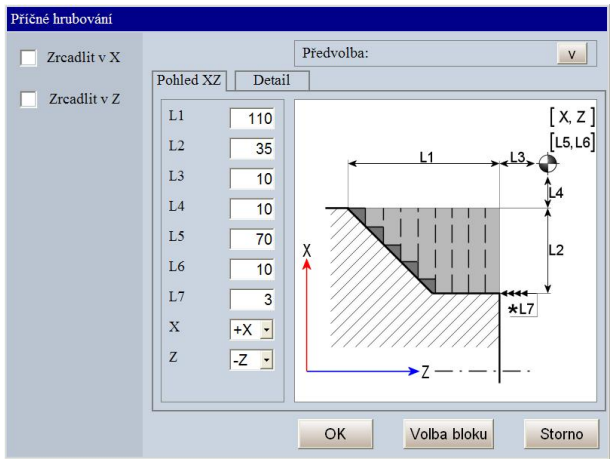
Úspěšně přidané nové tlačítko do `Menu - Tvary`



25.2 Přidání nového pevného tvaru

Při tvorbě nového dialogu je lepší vycházet z jiného hotového dialogu, který splňuje nejlépe požadavky pro nový tvar. V našem případě budeme vycházet podélného hrubování `RoughHorizont1.html`, který okopírujeme do souboru s názvem `RoughVertical1.html`. Od této chvíle začne fungovat odkaz, na který odkazuje nové tlačítko z menu.

Po okopírování HTML stránky se zobrazí dialog Podélné hrubování, který budeme upravovat.

Dialog pro zadávání parametrů před úpravou	Dialog pro zadávání parametrů po úpravě
	

25.2.1 Struktura HTML stránky tvořící dialog

Stručný náhled struktury HMTL souboru, tvořícího dialog

```
<HTML>
      <HEAD>
            <STYLE>
                  … předdefinování vzhledu dialogu
            </STYLE>
            <SCRIPT type="text/JavaScript">
                  … ovládání různých interaktivních prvků
            </SCRIPT>
      </HEAD>

      <BODY>
            <!-- String tables -->
                  <DIV id="StringTableCSY" class="StringTable">
                        … lokalizace popisků dialogu
                  </DIV>

            <DIV id="DialogBackground">
                  <!-- Titulek dialogu -->
                  <DIV class="WindowTitle" LocStrID="IDS_WindowTitle">
                        Rough Vertical 1
                        … nadpis dialogu
                  </DIV>

                  <!-- Společné prvky (zrcadlení, natočení,
                        měřítko, množení, ... ) -->
                  <DIV id="CommonArea">
                        … vložení levého pomocného menu ze souboru MenuCopy.html
                        <IFRAME src="MenuCopy.html" frameborder="0"
                        style="POSITION:absolute; WIDTH:100%;
                        HEIGHT:100%">
```

```
                        </IFRAME>
                    </DIV>

                    <!-- Prvky specifické danému tvaru -->
                    <DIV id="SpecificArea">
                        … obsah jednotlivých záložek (Tab0, Tab1, Tab2, …)
                        <DIV id="Tab0">
                            <!-- Parametry tvaru -->
                            <DIV id="ParamsArea">
                            </DIV>

                            <!-- Obrázek -->
                            <DIV id="PictureArea">
                                <IMG  id="Picture_Shape0"
                                    src="RoughVertical1.png">
                            </DIV>
                        </DIV>

                        <DIV id="Tab1">
                            …
                        </DIV>

                        <DIV id="Tab2">
                            …
                        </DIV>

                        <!-- Předvolby -->
                        <DIV  id="PreselectionsArea"
                            innerHTMLFile="Preselections.html">
                        … umožňuje ukládat nastavení jednotlivých parametrů do souboru
                        </DIV>


                        … tlačítka umožňující výběr jednotlivých záložek
                            (Tab0Top, Tab1Top, Tab2Top, …)

                        <DIV  id="Tab0Top" class="LabelMedium"
                            LocStrID="IDS_SelTab0"
                            onclick="ViewTab('Tab0')">
                            View XZ
                        </DIV>
                        <DIV  id="Tab1Top">
                            …
                        </DIV>
                        <DIV  id="Tab2Top">
                            …
                        </DIV>


                    </DIV>
                </DIV>
            </BODY>
</HTML>
```

25.2.2 Obrázek dialogu

Upřesňující obrázek pro daný tvar má předem nadefinovanou velikost a pozici v dialogu pomocí kaskádových stylů. Pro danou záložku se nachází v sekci označené `<!-- Obrázek -->`. Stačí v této sekci změnit jméno z `RoughHorizont1.png` na nový obrázek například **`RoughVertical1.png`**. (Obrázek má rozměry 400x400px)

```
<!-- Obrázek -->
<DIV id="PictureArea">
      <IMG id="Picture_Shape0" src="RoughVertical1.png">
</DIV>
```

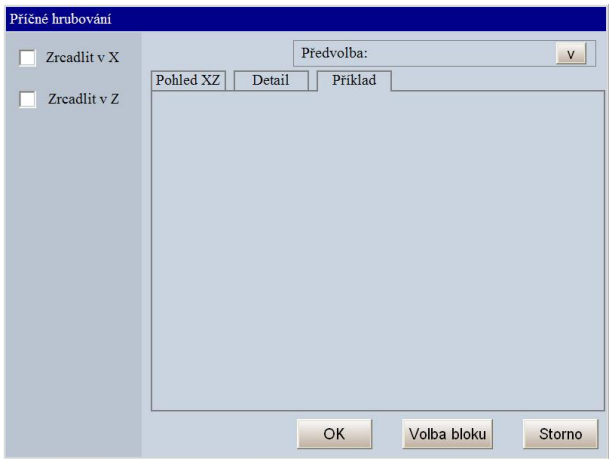
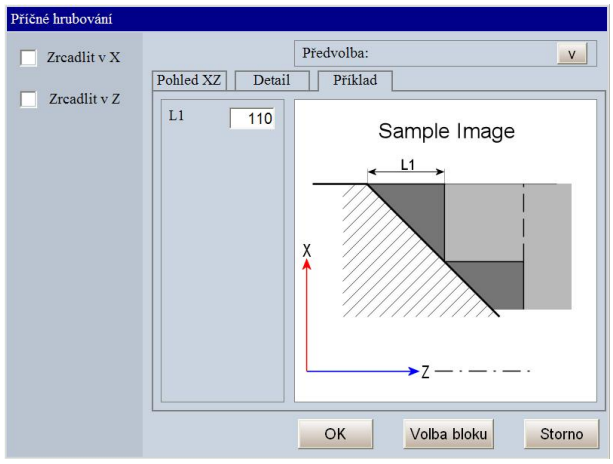
25.2.3 Nadpis dialogu

Nadpis nalezneme v sekci **`<!-- Titulek dialogu -->`**. Název stačí přepsat z `Rough Horizontal 1` na **`Rough Vertical 1`**. Kvůli lokalizaci, změnit `IDS_WindowTitle` v sekci `<!-- String tables -->` pro všechny jazykové verze.

```
<!-- Titulek dialogu -->
<DIV class="WindowTitle" LocStrID="IDS_WindowTitle">
      Rough Vertical 1
</DIV>
```

25.2.4 Záložka dialogu

Pro přehlednější zadávání parametrů pevného tvaru dialog umožňuje práci se záložkami. Následující postup popisuje přidání nové záložky pojmenované **`Sample`** a práci s ní.

Přidána prázdná záložka	Přidaný nový parametr a obrázek
	

➤ **Úprava JavaScriptu**

Využití skriptovacího jazyka JavaScript umožňuje ovládání ovládáni různých interaktivních prvků uvnitř dialogů(tlačítka, textová políčka, rolovací nabídky). JavaScript obsluhující záložky nalezneme uvnitř v HTML stránce. Pro přidání nové záložky přidáme do skriptu zvýrazněné řádky.

```
<SCRIPT type="text/JavaScript">

function ViewTab(strTabID)
{
      var oTab    = document.getElementById(strTabID);
      var oTab0   = document.getElementById("Tab0");
      var oTab1   = document.getElementById("Tab1");
      var oTab2   = document.getElementById("Tab2");
```

```
      var oTabTop  = document.getElementById(strTabID + "Top");
      var oTab0Top = document.getElementById("Tab0Top");
      var oTab1Top = document.getElementById("Tab1Top");
      var oTab2Top = document.getElementById("Tab2Top");

      oTab.style.zIndex       = "1";
      oTab.style.visibility   = "visible"
      oTabTop.style.zIndex    = "2";

      if (strTabID != "Tab0") {
            oTab0.style.zIndex      = "0";
            oTab0.style.visibility  = "hidden"
            oTab0Top.style.zIndex   = "0";
      }
      if (strTabID != "Tab1") {
            oTab1.style.zIndex      = "0";
            oTab1.style.visibility  = "hidden"
            oTab1Top.style.zIndex   = "0";
      }
      if (strTabID != "Tab2") {
            oTab2.style.zIndex      = "0";
            oTab2.style.visibility  = "hidden"
            oTab2Top.style.zIndex   = "0";
      }
}
</SCRIPT>
```

➢ **Úprava kaskádových stylů**

Každá záložka má předdefinovaný vzhled pomocí kaskádových stylů. Kaskádové styly pro záložky nalezneme uvnitř HTML stránky. Pro novou záložku vložíme zvýrazněné řádky.

```
<STYLE>
#Tab0, #Tab1, #Tab2     {
                        position: absolute; top: 76px; left: 10px;
                        width:  603px;  height:  426px;  background-color:
                        #CAD4E0; border: gray solid; border-width: 2px;
                        }

#Tab0Top    {
            position: absolute; top: 50; left: 10;  width: 100;
            height: 28; background-color: #CAD4E0; border: gray solid;
            border-bottom-width: 0px; border-left-width: 2px;
            border-right-width: 2px; border-top-width: 2px;
            text-align: center; cursor: default;
            }
#Tab1Top    {
            position: absolute; top: 50; left: 120; width: 100;
            height: 28; background-color: #CAD4E0; border: gray solid;
            border-bottom-width: 0px; border-left-width: 2px;
            border-right-width: 2px; border-top-width: 2px;
            text-align: center; cursor: default;
            }
#Tab2Top    {
            position: absolute; top: 50; left: 230; width: 100;
            height: 28; background-color: #CAD4E0; border: gray solid;
            border-bottom-width: 0px; border-left-width: 2px;
            border-right-width: 2px; border-top-width: 2px;
            text-align: center; cursor: default;
            }
</STYLE>
```

Stačí zvětšit hodnotu parametru **`left: 120;`** udávající posun záložky z levé strany o šířku záložky (110px) na hodnotu **`left: 230;`**

➢ **Zobrazení tlačítka záložky**
Pro zobrazení nového tlačítka záložky přidáme do HTML kódu zvýrazněnou část.

```
<DIV id="Tab0Top" class="LabelMedium" LocStrID="IDS_SelTab0"
      onclick="ViewTab('Tab0')">
      View XZ
</DIV>
<DIV id="Tab1Top" class="LabelMedium" LocStrID="IDS_SelTab1"
      onclick="ViewTab('Tab1')">
      Detail
</DIV>
<DIV id="Tab2Top" class="LabelMedium" LocStrID="IDS_SelTab2"
      onclick="ViewTab('Tab2')">
      Sample
</DIV>
```

➢ **Lokalizace názvu**
Nadpis záložky lokalizujeme přidáním zvýrazněné řádky do sekce **`<!-- String tables -->`** uvnitř stránky.

```
<!-- String tables -->
<DIV id="StringTableCSY" class="StringTable">
      <SPAN id="IDS_SelTab0">Pohled XZ</SPAN>
      <SPAN id="IDS_SelTab1">Detail</SPAN>
      <SPAN id="IDS_SelTab2">Příklad</SPAN>
</DIV>
<DIV id="StringTableENU" class="StringTable">
      <SPAN id="IDS_SelTab0">View XZ</SPAN>
      <SPAN id="IDS_SelTab1">Detail</SPAN>
      <SPAN id="IDS_SelTab2">Sample</SPAN>
</DIV>
```

➢ **Obsah záložky**
V HTML stránce je obsah každé záložky umístěn v HTML tagu `<DIV id="TabX">`.

```
<DIV id="Tab0">
      …
</DIV>
<DIV id="Tab1">
      …
</DIV>
<DIV id="Tab2">
</DIV>
```

Nyní je v dialogu přidána nová prázdná záložka

Příklad přidání nového parametru a obrázku do záložky

```
<DIV id="Tab2">
      <!-- Parametry tvaru -->
      <DIV id="ParamsAreaSample">
            <DIV id="Param_R00">
                  <DIV id="Param_R00_Lbl"   class="LabelMedium" >L1</DIV>
                  <INPUT id="Param_R00_Val" class="EditMedium"
                        storageName="MAC_ROUGH_SAMPLE" type="text"
                        value="1000" NumberCheck="NumberType: Real;"
                        NAME="Param_R00_Val">
                  </DIV>
            </DIV>

      <!-- Obrázek -->
      <DIV id="PictureAreaSample">
            <IMG id="Picture_Shape2" src="Sample1.png">
      </DIV>
</DIV>
```

Pro správné rozmístění prvků na stránce je třeba doplnit do kaskádových stylů:

```
<STYLE>
#ParamsAreaSample {position: absolute; left:  10px; top:  10px;
                  width: 164px; height: 402px; border: 1px solid gray;
                  }

#Param_R00        {position: absolute; left:  10px; top:  10px;
                  width: 164px; height: 40px;
                  }
#Param_R00_Lbl    {position: absolute; left: 0px; top: 0px;
                  width: 80px;
                  }
#Param_R00_Val    {position: absolute; left: 80px; top:   0px;
                  width:  60px;
                  }

#ParamsAreaSample {position: absolute; left:  10px; top:  10px;
                  width: 164px; height: 402px; border: 1px solid gray;
                  }

#Picture_Shape2   {width: 400px; height: 400px; border: 1px solid gray;
                  background-color: white;
                  }
</STYLE>
```

- **Předání nového parametru z dialogu do NCP programu**

Aby bylo možné předávat parametry z HTML dialogu jsou HTML tagy doplněny o parametry:

storageName	udává název proměnné v NCP programu, jejíž hodnota je naplněna z HTML dialogu
NumberCheck	umožňuje zkontrolovat typ zadaného čísla (Real, Integer,..)

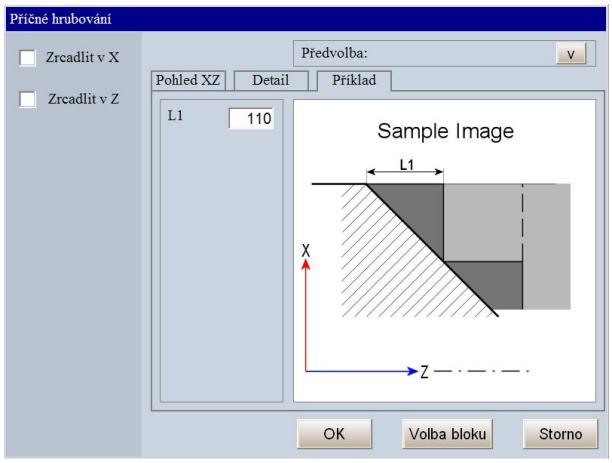
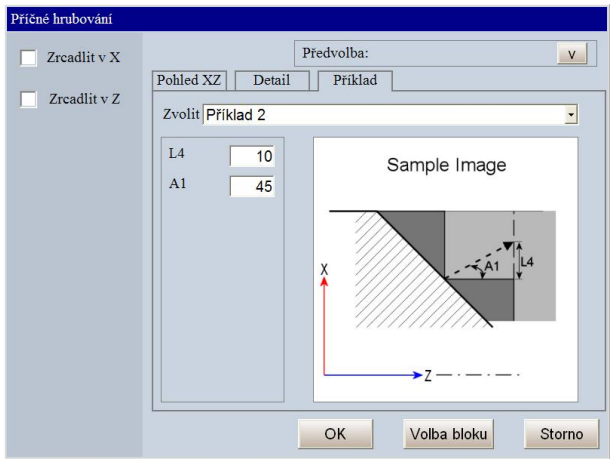
```
<INPUT id="Param_R00_Val" class="EditMedium"
                        storageName="MAC_ROUGH_SAMPLE" type="text"
                        value="1000" NumberCheck="NumberType: Real;"
                        NAME="Param_R00_Val">
```

Předání výběr obsahu záložky pomocí roletové nabídky

Do hlavičkového souboru X.NCH stačí nadefinovat stejnojmennou proměnnou
`$MAC_ROUGH_SAMPLE R30`

25.2.5 Rolovací nabídka

Rolovací nabídku můžeme použít v dialogu, pro rozšíření možností zadávání parametrů.

Záložka s jedním parametrem	Výběr obsahu záložky pomocí rolovací nabídky
	

➤ **Úprava JavaScriptu**

```
<SCRIPT type="text/JavaScript">

function DrillSelect(iSelectID)
{
      switch(iSelectID)
      {
            case 10:
                  DivCycleVisibility('SAMPLE1',0);
                  DivCycleVisibility('SAMPLE2',0);

                  CycleName = document.getElementById('MAC SELECT SAMPLE').
                        options[document.getElementById('MAC SELECT SAMPLE'
                        ).selectedIndex].value;

                  DivCycleVisibility(CycleName,1);

                  Picture Shape2.src= CycleName + '.png';

            break;

      }
}
</SCRIPT>
```

➤ **Úprava kaskádových stylů**

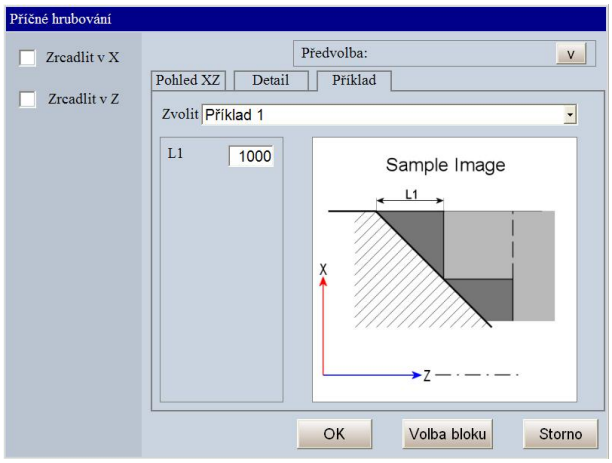
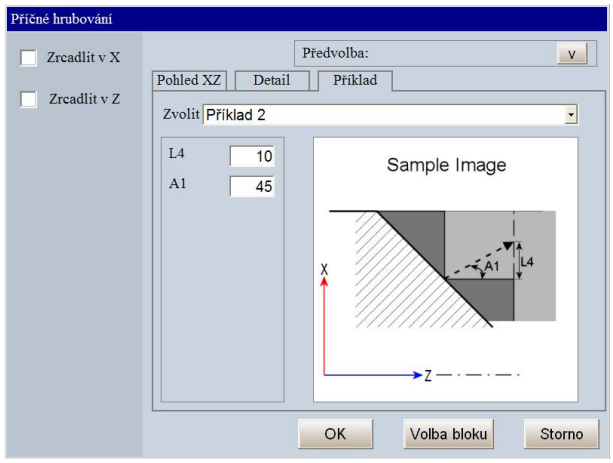
```
#ParamsAreaSample {position: absolute; left:  10px; top:  60px;
                   width: 164px; height: 352px; border: 1px solid gray;
                   }
#PictureAreaSample{position: absolute; left: 212px; top:  60px;
                   width: 300px; height: 340px;
                   }
#Picture_Shape2    {width: 350px; height: 350px; border: 1px solid gray;
                   background-color: white;
                   }
#SelectSample      {width: 500px;
                   }
#SelectSampleShow {position: absolute; left:  10px; top:  13px;
                   }
```

➤ **Lokalizace názvu**

```
<!-- String tables -->
            <DIV id="StringTableCSY" class="StringTable">
                  <SPAN id="IDS_Sample1">Příklad 1</SPAN>
                  <SPAN id="IDS_Sample2">Příklad 2</SPAN>
            </DIV>
```

➤ **Princip funkce rolovací nabídky**

Každá položka rolovací nabídky mění obsah konkrétní záložky a je umístěna v HTML oddílu s označením `<DIV id="....._HIDDEN"></DIV>`. Všechny tyto oddíly mají přednastavenu vlastnost `visibility = hidden` , tím nejsou do zavolání příslušného oddílu vidět. Viditelnost upravuje JavaScript podle vybrané položky v rolovací nabídce.

Volba 1	**Volba 2**
	


```
<DIV id="SAMPLE1_HIDDEN" style="visibility=hidden;">
      … obsah stránky při zvolené volbě 1 pomocí rolovací nabídky
</DIV>
<DIV id="SAMPLE2_HIDDEN" style="visibility=hidden;">
      … obsah stránky při zvolené volbě 2 pomocí rolovací nabídky
</DIV>
```

- **Obsah záložky**

```
<DIV id="Tab2">
<!-- Parametry tvaru -->
      <FORM id="SelectSampleShow">
            <TABLE>
            <TR>
            <TD class="LabelMedium" LocStrID="IDS_SelectProperty"
                  nowrap>Select</TD>
            <TD>
            <SELECT class="EditMedium"
                  storageName="MAC_SELECT_SAMPLE"   name="MAC_SELECT_SAMPLE"
                  ID="SelectSample" onpropertychange="DrillSelect(10);">

            <OPTION value="SAMPLE1" LocStrID="IDS_Sample1">Sample1
            <OPTION value="SAMPLE2" LocStrID="IDS_Sample2">Sample2
            </SELECT>

            </TD>
            </TR>
            </TABLE>
      </FORM>

<DIV id="ParamsAreaSample">
      <DIV id="SAMPLE1_HIDDEN" style="visibility=hidden;">
            <DIV id="Param_R00">
                  <DIV id="Param_R00_Lbl"   class="LabelMedium" >L1</DIV>
                        <INPUT id="Param_R00_Val" class="EditMedium"
                         storageName="MAC_ROUGH_SAMPLE"          type="text"
                         value="1000"   NumberCheck="NumberType: Real;"
                         NAME="Param_R00_Val">
            </DIV>
      </DIV>

      <DIV id="SAMPLE2_HIDDEN" style="visibility=hidden;">

            <DIV id="Param_R00">
                  <DIV id="Param_R00_Lbl"   class="LabelMedium" >L4</DIV>
                        <INPUT id="Param_R00_Val" class="EditMedium"
                         storageName="MAC_ROUGH_SAMPLESIZE" type="text"
                         value="10"   NumberCheck="NumberType: Real;"
                         NAME="Param_R00_Val">
                  </DIV>

            <DIV id="Param_R01">
                  <DIV id="Param_R01_Lbl"   class="LabelMedium" >A1</DIV>
                        <INPUT id="Param_R01_Val" class="EditMedium"
                         storageName="MAC_ROUGH_SAMPLEANGLE" type="text"
                         value="45"   NumberCheck="NumberType: Real;"
                         NAME="Param_R00_Val">
                  </DIV>

            </DIV>

      </DIV>

<!-- Obrázek -->
      <DIV id="PictureAreaSample">
            <IMG id="Picture_Shape2" src="Sample1.png">
      </DIV>
</DIV>
```

➤ **Předání nových parametrů z dialogu do NCP programu**

Předání dat z rolovací nabídky

```
<SELECT storageName="MAC_SELECT_SAMPLE">

      <OPTION value="SAMPLE1">Sample1
      <OPTION value="SAMPLE2">Sample2

</SELECT>
```

Do proměnné **MAC_SELECT_SAMPLE** se uloží vybraná položka z rolovací nabídky. Přidáme ji tedy do hlavičkového souboru NCP programu.

```
$MAC_SELECT_SAMPLE              R33
```

Jednotlivé položky z rolovací nabídky jsou konstanty, pro lepší názornost je pojmenujeme a nadefinujeme v hlavičkového souboru NCP programu.

```
$SAMPLE1                        150
$SAMPLE2                        151
```

Výstup z rolovací nabídky v NCP programu vypadá následovně:

Pro první volbu
MAC_SELECT_SAMPLE=SAMPLE1

Pro druhou volbu
MAC_SELECT_SAMPLE=SAMPLE2

Souhrn všech proměnných přidaných do hlavičkového souboru NCP programu potřebných pro tuto záložku

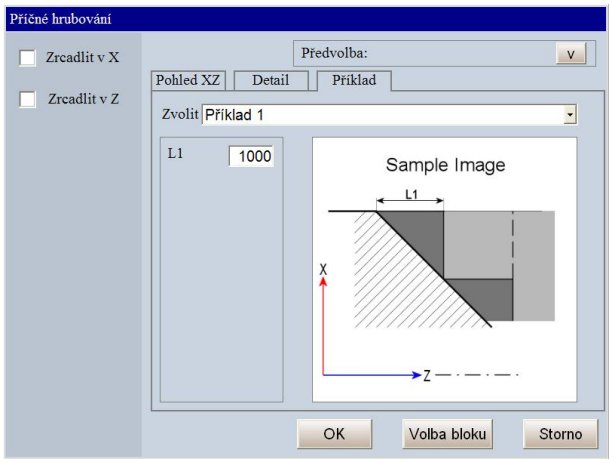
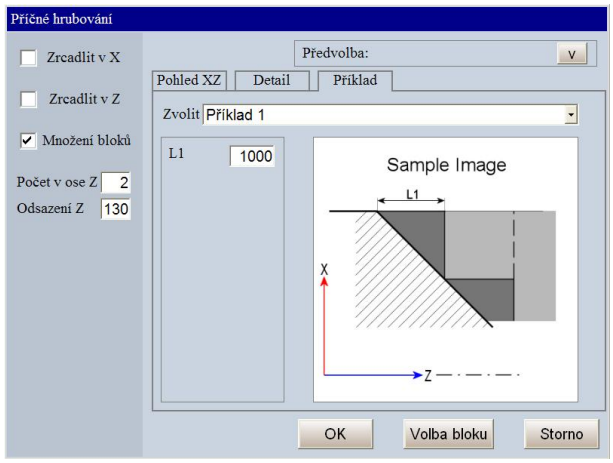
```
$MAC_ROUGH_SAMPLE               R30
$MAC_ROUGH_SAMPLESIZE           R31
$MAC_ROUGH_SAMPLEANGLE          R32
$MAC_SELECT_SAMPLE              R33

$SAMPLE1                        150
$SAMPLE2                        151
```

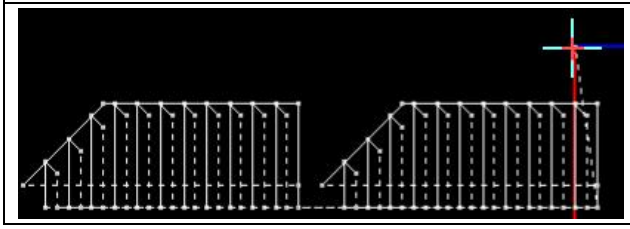
Předané hodnoty do NCP programu

```
MAC_SELECT_SAMPLE=SAMPLE1
MAC_ROUGH_SAMPLE=1000
MAC_ROUGH_SAMPLESIZE=10
MAC_ROUGH_SAMPLEANGLE=45
```

25.2.6 Levé menu

Levé menu umožňuje jednoduché operace s vytvořeným pevným tvarem pomocí tohoto dialogu jako zrcadlení, kopírování, natáčení.

Dialog pro zadávání parametrů	Rozšíření levého menu o kopírování v ose Z
	

Zobrazení výsledného programu	
	

➤ **Rozšíření levého menu o novou funkci – Kopírování v ose Z**

Dialog načítá levé menu ze souboru MenuCopy.html. Další požadovanou položku menu stačí vložit na konec této HTML stránky těsně před ..... `</BODY></HTML>`

```
<FORM id="MAC_COPY_Z_MENU_MAIN">
      <TABLE style="width=100%">
      <TR>
      <TD width="40px"><INPUT class="CheckBox"
            storageName="MAC_COPY_Z_MENU_COUNT_VIS" type="checkbox"
            name="chbxVisibility" InitialValue="0"
            onClick="MenuCopyVisibility('MAC_COPY_Z_MENU_MAIN')">
      </TD>
      <TD LocStrID="IDS_COUNT" class="LabelMedium" nowrap>
            Multiplication
      </TD>
      </TR>
      </TABLE>
</FORM>

<FORM id="MAC_COPY_Z_MENU_MAIN_HIDDEN" style="display=none">
      <TABLE style="width=100%">
      <TR>
      <TD LocStrID="IDS_COUNTZ" class="LabelMedium" nowrap>Count in Z</TD>
      <TD><INPUT class="EditMedium" storageName="MAC_COPY_COUNTZ"
            type="text" value="2" style="WIDTH:40px"
            NumberCheck="NumberType: Integer;">
```

```
      </TD>
      </TR>
      <TR>
      <TD LocStrID="IDS_DIFERENCEZ" class="LabelMedium" nowrap>
            Offset Z
      </TD>
      <TD><INPUT class="EditMedium" storageName="MAC_COPY_DIFERENCEZ"
            type="text" value="130" style="WIDTH:40px"
            NumberCheck="NumberType: Real;">
      </TD>
      </TR>
      </TABLE>
</FORM>
```

- **Lokalizace názvu**

```
<!-- String tables -->
            <DIV id="StringTableCSY" class="StringTable">
                  <SPAN id="IDS_COUNT">Množení bloků</SPAN>
                  <SPAN id="IDS_COUNTZ">Počet v ose Z</SPAN>
                  <SPAN id="IDS_DIFERENCEZ">Odsazení Z</SPAN>
            </DIV>
```

- **Předání nových parametrů do NCP programu**

Souhrn všech proměnných přidaných do hlavičkového souboru NCP programu

```
$MAC_COPY_Z_MENU_COUNT_VIS    I61
$MAC_COPY_COUNTZ              I62

$MAC_COPY_DIFERENCEZ          R93

$MAC_COPY_Z                   152
```

- **Vytvoření NCP programu**

Do již existujícího makra **MAC_COPY** obsluhující operace s vygenerovaným pevným tvarem vložíme část kódu, která po stisku tlačítka Množení bloků skočí na program **MAC_COPY_Z**, který to zajistí.

```
"********** COPY ************
"***************************

N BEGIN(MAC_COPY)
N     SUBOPT(SUBOPT_RESTOREPTRANSFORM,1)
N     G23
N     IF (EQ(MAC_COPY_Z_MENU_COUNT_VIS,1))
            " Pokud je v menu zvoleno Množení bloků v ose Z
            CALLMACRO(MAC_COPY_Z)
      ENDIF


      IF (EQ(MAC_COPY_MENU_SCALE_VIS,1))
            " Pokud je v menu zvoleno Měřítko
            MAC_COPY_SCALE = MAC_COPY_SCALE/100
            ASCALE(MAC_COPY_SCALE)
      ENDIF
      ……
N30 END
```

Nový program MAC_COPY_Z volá opakovaně přednastavený pevný tvar příkazem **CALLMACRO(MAC_COPY_PART)** a vždy provede transformaci souřadnic v záporném směru osy Z o požadovanou vzdálenost zadanou přes levé menu příkazem
ATRANSLATE(0,0, - MAC_COPY_DIFERENCEZ)

```
"******** MAC_COPY_Z ********
"***************************


N BEGIN(MAC_COPY_Z)
N     "Vrátit transformaci po skončení makra do původního stavu
      SUBOPT(SUBOPT_RESTOREPTRANSFORM,1)
N     G23

N10
      IF(EQ(MAC_COPY_COUNTZ,0))
            JMP(30)
            " Byl dokončen poslední prvek v řádku, tak konec kopírování
      ENDIF


N     CALLMACRO(MAC_COPY_PART)

N     " Požadovaná operace
      " V našem případě se jedná o množení pevného
      " tvaru v záporném směru v ose Z

      ATRANSLATE(0,0, - MAC_COPY_DIFERENCEZ)

      MAC_COPY_COUNTZ = MAC_COPY_COUNTZ - 1

N     JMP(10)

N30 END
```

22

22. TVORBA UŽIVATELSKÝCH INSTRUKCÍ A MAKER

Od verze překladače PLC 6.041 je umožněno si definovat a používat vlastní instrukce pro překlad PLC programu. Rozvoj uživatelských instrukcí může být definován jak na úrovni jazyka TECHNOL, tak na úrovni assembleru 386 a vyšším. Uživatelské instrukce mohou přebírat formální parametry a mohou si definovat vlastní lokální proměnné a návěští.

Uživatelské instrukce mohou být definovány v samostatném souboru, který se připojuje ke zdrojovému textu v době překladu.

22.1 Připojování externích definičních souborů

Uživatelské instrukce, symbolické identifikátory chyb a informačních hlášení (viz. Nastavování chyb – kapitola 14.) nebo různá makra, mohou být definovány v samostatných souborech, které se připojují ke zdrojovému textu v době překladu pomocí instrukce T_INCLUDE.

instrukce	**T_INCLUDE**

funkce	**T_INCLUDE**	**připojení definičního souboru**
syntax	**T_INCLUDE**	**file**
parametr	**"file"**	**název souboru**

Připojení definičního souboru ke zdrojovému textu. Parametr **"file"** je název souboru, který může obsahovat absolutní cestu. Pokud název žádnou cestu neobsahuje, bude se hledat ve stejném adresáři, kde se nachází zdrojový PLC program.

Pokud je uveden název souboru v apostrofech (‘) a neobsahuje absolutní cestu, předpokládá se umístění v systémovém adresáři SYSTEM.

Je zvykem umísťovat instrukce **T_INCLUDE** hned na začátek zdrojového programu a v názvech pro definiční soubory požívat příponu **INC** .

Definiční soubory mohou obsahovat definice symbolických konstant (chyb), definice maker a uživatelských

instrukcí a nesmějí obsahovat přímý výkonný instrukční kód (kromě kódu definovaného v makrech).

Příklad:

```
T_INCLUDE    VXR50.INC          ;Definiční soubor VXR50.INC se bude hledat
                                ;v adresáři, kde se nachází zdrojový PLC
                                ;program (VXR50.PLC).

T_INCLUDE    VXR50\VXR50.INC    ;Umístění definičního souboru VXR50.INC
                                ;v podadresáři VXR50 adresáře, kde je
                                ;umístěn zdrojový PLC program.
```

22.2 Definice uživatelských instrukcí a maker

Rozvoj uživatelských instrukcí může být definován jak na úrovni jazyka TECHNOL, tak na úrovni assembleru. Uživatelské instrukce mohou přebírat formální parametry a mohou si definovat vlastní lokální proměnné a návěští.

instrukce	**DEF_T_MACRO**

funkce	**DEF_T_MACRO začátek definice instrukce (makra)**	
syntax	**DEF_T_MACRO**	**name [par1, par2, ….]**
1.parametr	**"name"**	**jméno makra**
2.parametr	**"par1"**	**formální parametry**

Instrukce **DEF_T_MACRO** označuje začátek definice uživatelského makra, nebo uživatelské instrukce.

První parametr „**name"** je povinný a udává název makra nebo instrukce. Pod tímto názvem se potom makro nebo instrukce volá pro její vykonání, přičemž se automaticky provede rozvoj makra podle definice.

Další parametry jsou formální parametry makra nebo instrukce a jejich počet závisí od konkrétní implementace. Formální parametry slouží pro předávání skutečných proměnných do rozvoje makra nebo instrukce při jejím výkonu a mohou to být například konstanty, bitové proměnné a různé datové proměnné.

Volání uživatelských maker a instrukcí se provede prostým voláním podle názvu makra a výčtem skutečných parametrů:

```
name  par1, par2
```

Definice maker mohou být do sebe vnořovány, takže z těla jednoho makra možno volat jiné makro.

instrukce	**END_T_MACRO**

funkce **END_T_MACRO** **konec definice instrukce (makra)**

syntax **END_T_MACRO** **[modif]**

Instrukce **END_T_MACRO** označuje konec definice uživatelského makra, nebo uživatelské instrukce.

Instrukce nemusí mít žádný parametr. Pokud instrukce má parametry, jedná se o seznam řídících příznaků, které slouží pro dodatečné upřesnění uživatelské instrukce. Příznaky upřesňují „debugovatelnost“, práci se zásobníkem při závorkových operacích a konverzi pro předání parametrů. Popis jednotlivých příznaků bude uveden dále u instrukci (APPEND_T_MACRO) v části “Řízení uživatělských instrukcí”.

Poznámka:

Často se definice maker nezaobejde bez použití instrukcí assembleru, které se budou kombinovat se standardními instrukcemi v TECHNOLu. V tomto případe je nutné znát několik pravidel. Fyzická reprezentace bitu v RLO registru je bit s váhou 40h v AH registru mikroprocesoru. Datový registr odpovídá registru ECX. Nedoporučuje se používat SI s ESI registr, protože se nezachová jeho obsah ve standardních instrukcích TECHNOL. Lepší je nepočítat se zachováním obsahů registrů, když jsou mezi naše instrukce vkládány standardní instrukce TECHNOL.

Příklad:

```
DEF_T_MACRO       ERRNUM
                  EQUI        ERR_VR1,    4512h       ;chyba 1.12.45
                  EQUI        ERR_VR2,    4612h       ;chyba 1.12.46

                  EQUI        DD123,      123
END_T_MACRO
```

22.3 Formální parametry a lokální symboly maker

Makro obsahuje při své definici formální parametry. Formální parametry slouží pro předávání skutečných proměnných do rozvoje makra nebo instrukce při jejím výkonu (rozvoji makra).

Kromě formálních parametrů, může makro běžně používat všechny globální a lokální proměnné, které jsou v okamžiku výkonu makra k dispozici.

Když je potřeba při definici makra použít některý z formálních parametrů pro instrukce TECHNOL, je nutné použít před názvem formálního parametru prefix: “ **.TMAC** “. Tento prefix způsobí, že instrukce TECHNOLu přebere formální parametr tak, aby došlo ke správné náhradě skutečného parametru v okamžiku výkonu makra s ohledem na její název a typ. Prefix “ .TMAC “ se doporučuje psát jako první před případnými dalšími prefixy.

Příklad:

```
Příklady použití formálních parametrů:

DEF_T_MACRO       POKUS       PAR1, PAR2, PAR3
                  LOD         TMAC.PAR3          ;načte PAR3 podle jeho typu
                  LDR         TMAC.PAR1          ;načte bit PAR1
                  LO          -TMAC.PAR2         ;log. OR s negací bitu PAR2
                  WR          TMAC.PAR3.PAR1     ;zápis bitu na adresu PAR3
                                                 ;s váhou PAR1
                                                 ;složitější adresace bitu)
                  LDR         ALFA               ;načte globální bit ALFA
                  LO          TMAC.PAR2          ;log. OR s bitem PAR2
                  STO1        TMAC.BYTE.PAR3     ;podmíněný zápis do PAR3
                                                 ;typ je změněn prefixem BYTE
END_T_MACRO

;Volání makra:
;ALFA a BETA jsou bitové proměnné a BUNX je datová proměnná

POKUS       ALFA, BETA, BUNX        ;Volání uživatelského makra
```

Makro může ve svém rozvoji definovat vlastní návěští a vlastní data. Když by makro potom bylo v programu použito vícekrát, došlo by ke chybě překladu následkem vícenásobné definice symbolů. Pro odstranění tohoto problému slouží instrukce T_LOCAL.

instrukce	**T_LOCAL**

funkce	**T_LOCAL**	**definice lokálních symbolů makra**
syntax	**T_LOCAL**	**sym1, [sym2, sym3, ….]**

Instrukce **T_LOCAL** musí být umístěná bezprostředně za instrukcí pro začátek definice makra DEF_T_MACRO a může být použita vícekrát. Instrukce T_LOCAL se používá pro specifikování lokálních symbolů v rámci makra. Lokálními symboly mohou být návěští, datové a bitové proměnné, které jsou použity jen v rozvoji makra. Instrukci je nutno použít vždy, kdy takové symboly jsou v rámci makra definovány a kdy se předpokládá vícenásobné použití makra (instrukce) ve zdrojovém kódu.

Bitové a datové proměnné deklarované v makru musí mít lokální charakter a proto se musí definovat v modulu **DATA_LOCAL,** včleněném přímo v makru (viz. Popis modulů – Kapitola 5., Struktura PLC programu). Pro definici datových proměnných možno použít instrukci **DS** a pro definici bitových proměnných možno použít instrukci **DFM**. Jediná výjimka je, že v instrukci DFM musí být povinně definováno všech osm bitů.

Příklad:

```
Definice lokálních dat

DEF_T_MACRO POKUS3      PAR1, PAR2, PAR3
            T_LOCAL     BUN_M1, BUN_M2, BUN_BIT      ;lokální symboly makra
            T_LOCAL     BIT0,BIT1,BIT2,BIT3,BIT4,BIT5,BIT6,BIT7

            DATA_LOCAL
                  BUN_M1:     DS    1     ;lokální bajtová proměnná
                  BUN_M2:     DS    2     ;lokální wordová proměnná
                  BUN_BIT:    DFM BIT0,BIT1,BIT2,BIT3,BIT4,BIT5,BIT6,BIT7
            DATA_LOCAL_END
```

Příklad:

```
První a druhý parametr makra jsou bitové proměnné a třetí parametr je
datová proměnná typu WORD


DEF_T_MACRO POKUS4      PAR1, PAR2, PAR3
      T_LOCAL     NAVM                                        ;lokální návěští
      T_LOCAL     BIT0,BIT1,BIT2,BIT3,BIT4,BIT5,BIT6,BIT7   ;lokální symboly

      DATA_LOCAL
            DFM BIT0,BIT1,BIT2,BIT3,BIT4,BIT5,BIT6,BIT7    ;lokální bity
      DATA_LOCAL_END

      LDR         -TMAC.PAR1        ;čtení negace formálního bitu PAR1
      LO          TMAC.PAR2         ;log. OR s formálním bitem PAR2
      LA          -ALFA             ;log. AND s globálním bitem ALFA
      WR          BIT0              ;zápis do lokálního bitu makra BIT0
      JL0         NAVM              ;podmíněný skok
      LDR         TMAC.PAR2         ;čtení formálního bitu PAR3
      FL1         1,BIT1            ;podmíněný zápis do lokálního bitu BIT1
NAVM:                               ;lokální návěští makra
      LOD         TMAC.PAR3         ;čtení z formálního parametru(word)
END_T_MACRO


;Volání makra:

POKUS4      ALFA, BETA, BUNX        ;Volání uživatelského makra
```

22.4 Řízení uživatelských instrukcí

Mezi další možnosti řízení uživatelských instrukcí patří možnost nastavení ladění, konverzí a práce se zásobníkem. Také je umožněno tzv. přetěžování základních instrukcí jazyka TECHNOL uživatelskými instrukcemi.

instrukce	**APPEND_T_MACRO**

```
funkce      APPEND_T_MACRO    řízení uživatelské instrukce
syntax      APPEND_T_MACRO    name, alias, [modif]
```

Instrukce **APPEND_T_MACRO** slouží pro připojení názvu k rezervovaným názvům překladače TECHNOL a pro nastavení příznaků. Tato instrukce se samotná používá hlavně pro přetěžování názvů instrukcí a vzhledem k její speciálnějšímu významu se budeme hlavně zabývat seznamem příznaků, které jsou v ni uvedeny. Tento seznam se také používá v parametrech instrukce **END_T_MACRO** , kde je jeho hlavní použití. Příznaky jsou odděleny čárkou.

Přehled nastavování příznaků:

1.parametr		**2.parametr**		**3.parametr**	
Vztah k zásobníku log.instrukcí		Konverze vstup.parametrů		Nastavování breakpointů (DEBUG)	
T_NORMAL*	Nemá vztah k zásobníku	C_0*	Bez konverze	D_OFF*	Instrukce nemá povolen breakpoint
T_BEGIN	Vyprázdnění zásobníku	C_1	Změna závorek na řetězce _op, _cl, …	D_ON	Instrukce má povolen breakpoint
T_END	Koncová instrukce, podobně jako WR.				
T_PUSH	Uložení obsahu RLO do zásobníku, podobně jako LDR.				
T_POP	Vybrání RLO ze zásobníku, jako samotné LO, LA.				

Implicitní nastavení pro instrukce je T_NORMAL, C_0, D_OFF.

Pokud v ukončovací instrukci definice makra END_T_MACRO žádné parametry neuvedeme, instrukce nebude mít žádný vztah vzhledem k zásobníku, nebude mít konverzi parametrů a nebude mít povolen breakpoint.

Příklad:

```
Pro konec definice makra:
Uživatelská instrukce má být typu koncové instrukce (WR, FL1,..) , nemá mít
konverzi a je bez ladění:

END_T_MACRO       T_END, C_0, D_OFF

APPEND_T_MACRO    ALFA, BETA, T_END, C_0, D_OFF
```

instrukce **CONTROL_T_MACRO**

funkce **CONTROL_T_MACRO** **řízení uživatelských instrukcí**

syntax **CONTROL_T_MACRO** **par**

Instrukce **CONTROL_T_MACRO** slouží pro řízení vykonávání všech uživatelských instrukcí a maker. Instrukce má jeden parametr, kterým je řídící klíčové slovo. Instrukce může být v programu použita vícekrát.

instrukce	parametr	význam
CONTROL_T_MACRO	POS*	(implicitní nastavení) Uživatelské instrukce se provádí až po rozdekódování standardních instrukcí TECHNOLu (posprocesor). V tomto případě se nedá použít přetěžování standardních instrukcí.
	PRE	Uživatelské instrukce se provádí před rozdekódováním standardních instrukcí TECHNOLu (preprocesor). V tomto případě je možno použít přetěžování standardních instrukcí.

23

23. DODATKY

23.1 Konstantní řezná rychlost

23.1.1 Popis konstantní řezné rychlosti

Konstantní řezná rychlost (KŘR) je způsob řízení otáček vřetene, kdy se ze zadané velikosti řezné rychlosti a okamžité polohy řídící souřadnice, neustále vypočítávají požadované otáčky vřetene.

KŘR není omezena na pohybové bloky a také ji možno plně řídit z PLC programu. Systém i PLC mají možnost omezit otáčky vřetene na zadanou hodnotu..

Pro obvodovou rychlost platí:

$$v = \omega.r = 2.\pi.n.x$$

x = poloha řídicí souřadnice (korigována vzhledem ke korekcím a posunutí)
v = obvodová rychlost (KŘR)
n = otáčky vřetene

Vypočtené napětí, které se zadává pro vřeteno (rozsah 0-7FFFh)

$$U_x = \frac{\frac{v}{2.\pi.x}.U.U_m}{P}$$

U_x = vysílané napětí na vřeteno(rozsah 0-7FFFh)

U = maximální napětí pro vřeteno pro daný převodový stupeň (poměr k max.napětí)
U_m= maximální napětí (7FFFh)
P = maximální otáčky dané převodové řady

23.1.2 Konstantní řezná rychlost – ovládání z NC programu

Systém používá pro KŘR dvě funkce:
G96 – Konstantní řezná rychlost s posuvem mm/ot
G97 – Konstantní řezná rychlost s posuvem mm/min

Řezná rychlost se zadává funkcí **„S"**, která má v tomto případě význam nikoli otáček, ale řezné rychlosti v metrech za minutu [m/min]. Například řezná rychlost `50m/min` by se zadala hodnotou „`S50.0`". Kromě toho je možno při programování využít systémový parametr **„CCS"**. V tomto případě se naprogramuje: „`CCS=50.0`".

Z NC programu možno kdykoli zadat omezení otáček pro KŘR pomocí systémového parametru **„SLIM".** Systém omezuje otáčky vzhledem k menší hodnotě z maximálních otáček převodového stupně a ze zadaného omezení.

Příklad:

```
N10 G0 X300 Z100 M44 M3       "rychloposuv na průměr 300, otáčky 100ot/min
    SLIM=1000 S100            "omezení maximálních otáček na 1000ot/min
N20 G96 G1 F0.2 S90 X50       "zařadí KŘR 90m/min, sjetí na průměr 50, posuv
                              "F = 0.2mm/ot
```

Při programování rychloposuvu se mění otáčky rovněž podle průměru, pokud není funkce G96 odvolána programováním G94.

KŘR není omezena jen na pohybové bloky NC programu. Zařazení KŘR může proto být i v nepohybovém bloku. Také změna délkové korekce a změna posunutí počátku se uplatní okamžitě a nemusí se čekat na pohybový blok. Korekce a posunutí mají vliv na KŘR, protože KŘR se počítá na špičku nástroje a ne na suport. KŘR se může také uplatňovat v ručních pojezdech nebo v jiných druzích pohybu, například vlečení nebo pro pohyby řízené z PLC programu.

KŘR nikdy nepřepíná převodový stupeň, ale omezí otáčky na maximální otáčky daného převodového stupně (pokud není ještě jiné omezení otáček).

23.1.3 Konstantní řezná rychlost – ovládání z PLC programu

PLC program má k dispozici pole double-wordových hodnot pro sledování stavu KŘR. Na systému je možno jednotlivé hodnoty sledovat v prohlížení paměti systému , nebo v lad9c9m programu Wintechnol.

název	adresa	Popis
AKRR_ACT_V	A178h	Aktuální konstantní řezná rychlost [mm/min] (G96 nebo PLC)
AKRR_ACT_SMAX	A17Ch	Aktuální maximální otáčky [1/1000 ot/min] (SLIM nebo PLC)
AKRR_ACT_DIST	A180h	Aktuální korekce poloměru [1/8 µm] (jen PLC)
AKRR_ACT_VCORR	A184h	Aktuální korekce řezné rychlosti [mm/min] (jen PLC)
AKRR_ACT_R	A188h	Aktuální poloměr [1/8µm]
AKRR_ACT_P	A18Ch	Aktuální max.otáčky dle převodové řady [1/1000 ot/min]
AKRR_ACT_U	A190h	Aktuální rozsah napětí dle převodové řady [0-7FFFh]
AKRR_ACT_S	A194h	Aktuální vypočtené otáčky [1/1000 ot/min]
AKRR_ACT_OUT	A198h	Aktuální vypočtené napětí [0-7FFFh]

PLC program má možnost zadávat velikost KŘR, maximální otáčky, korekci poloměru pro výpočet KŘR a korekci řezné rychlosti. Také může KŘR aktivovat a případně modifikovat výpočet. Pro řízení PLC program používá bity umístěné v řídicím bajtu **AKRR_CNTR.**

Název bitu	Váha		popis
EXT_AKRR_V	0	0	Velikost KŘR se řídí z NC programu (G96,G97+ S)
		1	Velikost KŘR zadává PLC program v buňce **AKRR_V**
EXT_AKRR_SMAX	1	0	Maximální otáčky zadává NC program (G76)
		1	Maximální otáčky zadává PLC program v buňce **AKRR_SMAX**
EXT_AKRR_DIST	2	0	PLC program nezadává korekci poloměru
		1	PLC program zadává korekci poloměru v buňce **AKRR_DIST**
EXT_AKRR_VCORR	3	0	PLC program nezadává korekci řezné rychlosti
		1	PLC program zadává korekci řezné rychlosti v **AKRR_VCORR**
EXT_AKRR_REQ	4	0	Aktivace KŘR se řídí z NC programu (G96,G97)
		1	Aktivace KŘR z PLC programu
EXT_AKRR_SUPORT	5	0	Standardní výpočet KŘR
		1	Výpočet KŘR nezapočítává korekce a posunutí
EXT_AKRR_P	6	0	Standardní zadání max.otáček převodu v „ChannelConfig“
		1	Externí zadání max.otáček převodového stupně
EXT_AKRR_U	7	0	Standardní zadání max.rozsahu převodu v „ChannelConfig“
		1	Externí zadání maximálního rozsahu výstupu
EXT_AKRR_R		0	Standardní zadání ze systému
		1	Externí zadání poloměru pro KŘR z PLC

název	adresa	Popis
AKRR_V	A19Ch	Velikost KŘR zadávaná z PLC [mm/min]
AKRR_SMAX	A1A0h	Maximální otáčky zadávané z PLC [1/1000 ot/min]
AKRR_DIST	A1A4h	Korekce poloměru zadávaná z PLC [1/8 µm]
AKRR_VCORR	A1A8h	Korekce řezné rychlosti zadávaná z PLC [mm/min]
AKRR_P	A1ACh	Max.otáčky přev.stupně z PLC [1/1000 ot/min]
AKRR_U	A1B0h	Rozsah napětí dle převod.rady z PLC [0-7FFFh]
AKRR_R	A1B4h	Poloměr z PLC [1/8 um]

23.2 Nelineární korekce

23.2.1 Popis a konfigurace pro nelineární korekce

Nelineární korekce umožňují korigovat nepřesnosti v odměřování a tím významně zvýšit přesnost obrábění. Nepřesnosti se většinou objeví při nepřímém odměřování. Nepřesnost může nastat například i vlastní vahou suportu při poloze ve větší vzdálenosti od stojanu (tzv. „padání“ osy). Všechny tyto nepřesnosti lze odstranit nelineárními korekcemi.

Každá rovinná nelineární korekce je vytvořena jedním párem tabulek pro kladný a záporný směr řídící souřadnice. Každá tabulka je funkčním vztahem, který určuje závislost řízené souřadnice na řídící souřadnici. Každá souřadnice může mít 2 nezávislé nelineární korekce (NK1 a NK2).

Například pro rovinnou korekci souřadnice X (řízena souřadnice je X) platí 4 funkční závislosti :

$\Delta X_1 = NK1_{PLUS}$ (1.řídicí osa) ;například 1.řídící souřadnice může být X - osa koriguje sama sebe
$\Delta X_1 = NK1_{MINUS}$ (1.řídicí osa)
$\Delta X_2 = NK2_{PLUS}$ (2.řídicí osa) ;například 2.řídící souřadnice může být Y- závislost jedné souřa -
$\Delta X_2 = NK2_{MINUS}$ (2.řídicí osa) ;dnice na druhé

ΔX_1…… korekce polohy řízené souřadnice od 1. nelineární korekce
ΔX_2…… korekce polohy řízené souřadnice od 2. nelineární korekce
$NK1_{PLUS}$ …. 1. nelineární korekce pro kladný směr první řídící souřadnice
$NK2_{MINUS}$ … 2. nelineární korekce pro záporný směr druhé řídící souřadnice

Systém si načítá nelineární korekce po zapnutí systému ze souborů. Názvy těchto souborů (pro jednotlivé osy a směry) jsou spolu s dalšími informacemi uvedené v konfiguračním souboru typu „`ChannelConfig`“ v XML tvaru. Soubory musí být umístěné v adresáři „`Config`“ a doporučuje se použít skupinu „`CNC User Files`“. Soubory musí být také přidány do skriptu „`PLC.NSI`“ pro tvorbu SETUPu PLC.

Příklad:

Příklad konfigurace pro nelineární korekce (přesný význam atributů je uveden dále).
Do elementu „`Servo`“ je přidáno:

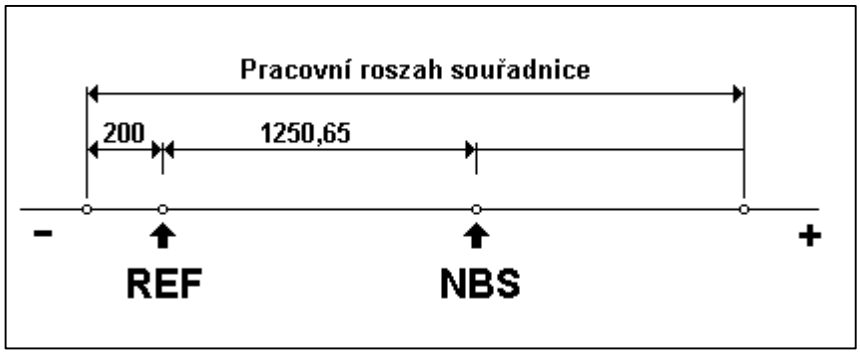
```
<NonLinearCompensation
      Active="1"
      Step="25.0"
      Begin="10.5"
      FileNamePos="Nk2plus.txt"
      FileNameNeg="Nk2minus.txt">
</NonLinearCompensation>
```

Soubory `Nk2plus.txt a Nk2minus.txt` jsou v adresáři „`CNC User Files\Config`“.

element **Servo**			**konfigurace servosmyčky**	
	element **NonLinearCompensation**		**nelineární korekce** (jedna servosmyčka může používat až 3)	
		atribut **Active**	**Je daná nelineární korekce použita?**	
			0	Nelineární korekce je zakázána (default)
			1	Nelineární korekce je povolena
		atribut **Type**	**Typ nelineární korekce**	
			0	základní (default)
			1	cyklická
			2	dynamická
		atribut **ControllingAxis**	**Řídící NC osa pro danou nelineární korekci**	
			0,1,..	řídicí NC osa – stejná jako číslo serva (default)
			xx	jiné přiřazení pro řídicí NC osu
		atribut **Step**	**Krok po kterém je korekce uvedena v tabulce [mm]**	
			1	krok tabulky 1 mm (default)
			xx	krok tabulky, maximálně 32 mm
		atribut **Begin**	**Umístění bodu, ke kterému se vztahuje první položka tabulky** (v mm vzhledem k nulovému bodu stroje)	
			0	nulový bod stroje a začátek tabulky jsou totožné (default)
			xx	míra v mm
		atribut **StepCount**	**Počet kroků tabulky**	
			xx	počet kroků, maximálně 1000 (default)
		atribut **CycleLen**	**Délka cyklu pro cyklickou nelineární korekci [mm]**	
			xx	Délka cyklu
		atribut **FileNamePos**	**Jméno souboru, ze kterého se má načíst tabulka pro kladný směr** Soubor je umístěn v adresáři „Config“	
			Abc..	Jméno souboru pro kladný směr
		atribut **FileNameNeg**	**Jméno souboru, ze kterého se má načíst tabulka pro záporný směr** Soubor je umístěn v adresáři „Config“	
			Abc..	Jméno souboru pro záporný směr

Nastavení začátku tabulky (atribut „Begin“)

Začátek tabulky od nulového bodu stroje (NBS) řídící souřadnice se zadává v atributu „Begin“. Pokud není nula stroje (NBS) na kraji pracovního prostoru souřadnice (v záporném směru), zadá se jako parametr „Begin“ vzdálenost NBS od záporné krajní polohy pracovního rozsahu souřadnice. Tato vzdálenost je totožná se začátkem působení nelineárních korekcí. Vzdálenost NBS a reference je uvedena pro jednotlivé osy v konfiguraci pro NC osy (element „NCAxis“ atribut „MachineNullPoint“). Pokud souřadnice „jezdí“ i za referenční bod , přičte se ještě tato vzdálenost a součet se zadá jako parametr „Begin“. Pro uvedený příklad na obrázku pro osu X by bylo nastaveno MachineNullPoint="-1250.65".
K této hodnotě se přičte -200.
Atribut „Begin“ by měl hodnotu: Begin="-1450.65".



23.2.2 Soubory s tabulkou korekcí

V souboru s korekcemi jsou uvedeny v textovém tvaru naměřené hodnoty korekcí pro příslušnou osu a směr. Každá hodnota korekce je uvedena na samostatném řádku. Řádek začíná pořadovým číslem kroku s dvojtečkou (velikost kroku je uvedena v konfiguraci v atributu „`Step`"), za kterým následuje hodnota korekce.

Velikost kroku nelineárních korekcí může být v rozsahu 1 až 32.767. V rámci zadaného kroku systém určuje aktuální korekci pomocí lineární interpolace.
V tabulce nelineárních korekcí nemusí být obsaženy všechny kroky. Nezadané údaje systém opět dopočítá lineární interpolací z nejbližších zadaných hodnot. Této vlastnosti se využívá tehdy, když potřebujeme mít větší rozestup měřených hodnot.

Příklad:
Zadané hodnoty korekce mají rozestup 100mm (100000 µm). Velikost kroku může být maximálně 32.767 µm, proto si zvolíme krok například 10.000 mm a do tabulky korekcí zadáme jen každou desátou hodnotu:

```
000: 00
010: 08           „řádek odpovídá 10x10000 = 100000 µm
020: 02
... atd.
```

Systém si vnitřně automaticky doplní nezadané kroky pomocí lineární interpolace:

například systém vnitřně dopočítá úsek mezi kroky 10 a 20 takto:

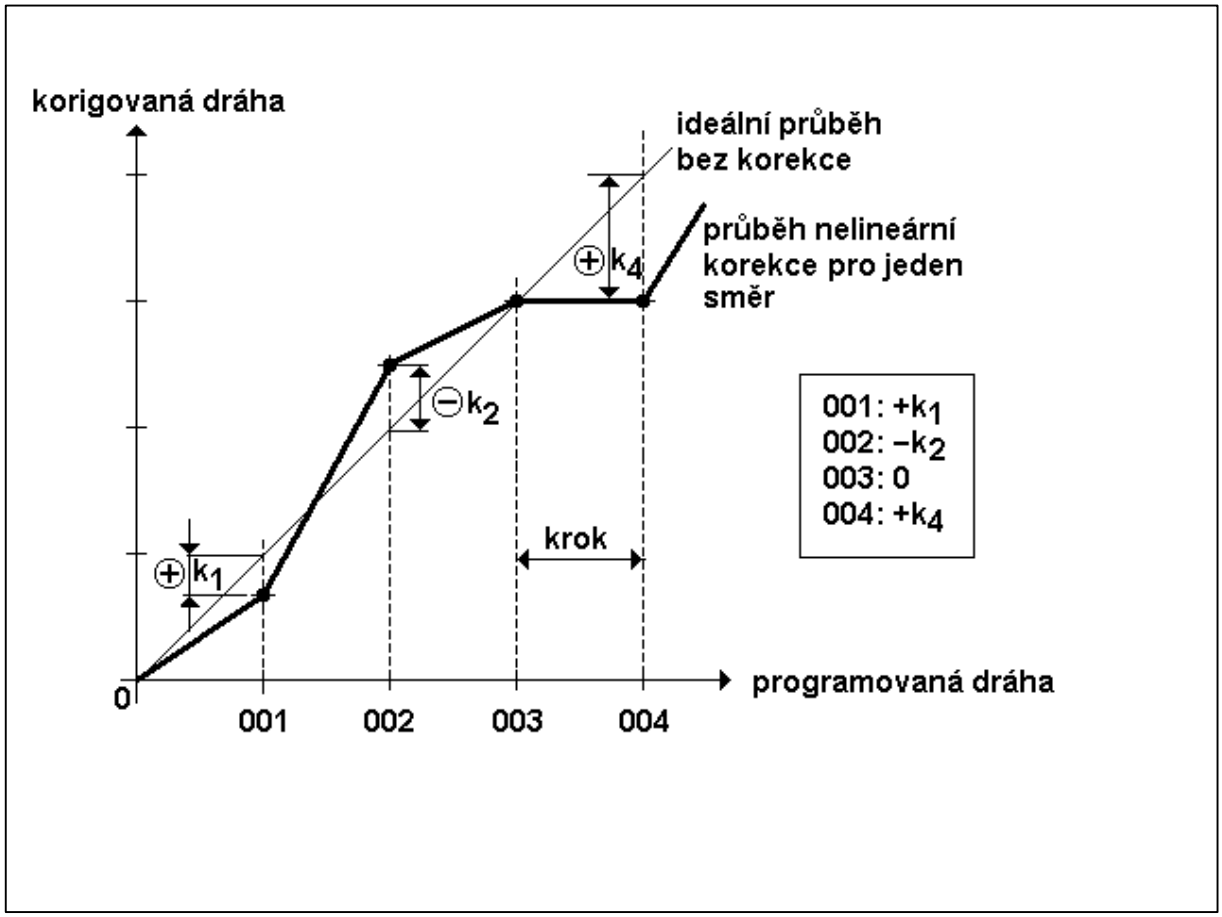
```
( 010: 08
  011: 07
  012: 06
  013: 06
  014: 05
  015: 05
  016: 04
  017: 03
  018: 03
  019: 02
  020: 02 )
```

Syntaktická pravidla pro rovinné korekce:

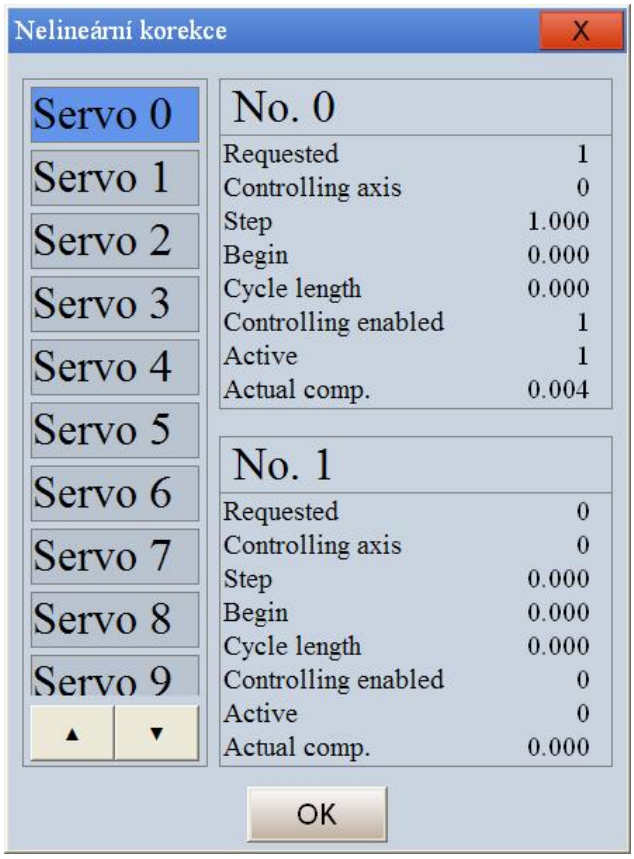
Soubor může obsahovat komentáře, uvozené uvozovkami.
Číslo kroku je trojmístné číslo počínaje 001(s eventuelními úvodními nulami), následované povinně dvojtečkou.
Za dvojtečkou následuje kladné nebo záporné číslo (znaménko plus se neuvádí), udávající v mikronech korekční hodnotu. Maximální povolená hodnota čísla je +/-32000 (mikronů).
V řádcích může být libovolný počet mezer a tabulátorů.
Pořadové číslo kroku musí být vzestupné, ale nemusí být obsaženy všechny kroky. Nezadané údaje systém dopočítá lineární interpolací z nejbližších zadaných hodnot.

23.2.3 Znaménka rovinných korekcí

Na obrázku je uvedeno zadávání znamének v korekčním souboru. Znaménko plus udává, že je nutné korekci přičíst, abychom se dostali na ideální průběh, znaménko mínus udává, že je nutné korekci odečíst, abychom se dostali na ideální průběh.



Pro kontrolu účinku korekcí je možno použít diagnostiku nelineárních korekcí:



Název	Popis	Odpovídá
Requested	Nelineární korekce je požadována	Element `NonLinearCompensation` Atribut `Active`
Controlling axis	Řídicí NC osa pro danou korekci (0,1,2,..)	Element `NonLinearCompensation` Atribut `ControllingAxis`
Step	Krok korekce v tabulce [mm]	Element `NonLinearCompensation` Atribut `Step`
Begin	Umístění bodu, ke kterému se vztahuje první položka tabulky	Element `NonLinearCompensation` Atribut `Begin`
Cycle length	Délka cyklu pro cyklickou nelineární korekci [mm]	Element `NonLinearCompensation` Atribut `CycleLen`
Controlling enabled	Aktivace nelineární korekce od řídící osy (např. osa byla zreferována)	`NlcAxisEnable` Jednotlivé bity povolují nelineární korekci pro jednotlivé osy (NELKOR_FLAG)
Active	Nelineární korekce je aktivní	Všechny podmínky pro aktivaci nelin.korekce (`NlcAxisEnable, NlcServoEnable, NlcSysServoEnable,…`)
Actual comp.	Aktuální hodnota kompenzace [mm]	Právě vybraná a interpolovaná hodnota korekce z tabulky

23.3 Sériové porty pro PLC

23.3.1 Konfigurace pro sériové porty PLC

Konfigurace pro sériové porty pro PLC se provede v konfiguračním souboru „`ChannelConfig`".

element **`ComChannel`**		**nastavení pro jeden sériový kanál pro PLC**	
	atribut **`No`**	**Logické číslo COM kanálu (0, ...). Pod tímto číslem s ním pracuje PLC**	
		0,1,..	logické číslo COM kanálu
	atribut **`Active`**	**Je daný sériový kanál použit?**	
		0	Sériový kanál je zakázán (default)
		1	Sériový kanál je povolen
	atribut **`ServiceMethod`**	**Způsob obsluhy sériového kanálu**	
		Buffered	komunikace je pomocí bufferu předávána primárnímu procesoru, data nejsou zpracovávána v reálném čase (default)
		RT	sériový port obsluhuje přímo daný procesor, komunikace je obsluhována v reálném čase (připravuje se)
	atribut **`PhysicalComNo`**	**Číslo sériového portu podle Windows (COM1, ...)**	
		1,2,..	číslo sériového portu podle Windows
	atribut **`ComSpeed`**	**Přenosová rychlost sériového portu (bit/s)**	
		115200	rychlost 11520 bit/s (default)
		xx	zadaná přenosová rychlost
	atribut **`ComDataBits`**	**Počet datových bitů**	
		8	počet datových bitů 8 (default)
		7	počet datových bitů 7
	atribut **`ComStopBits`**	**Počet stop bitů**	
		2	2 stop bity (default)
		1	1 stop bit
	atribut **`ComParity`**	**Parita**	
		None	bez parity (default)
		Odd	lichá parita
		Even	sudá parita
		Mark	paritní bit log.1
		Space	paritní bit log.0

Příklad:

```
<ComChannel No="0"
      Active="1"
      PhysicalComNo="2"
      ComDataBits="8"
      ComSpeed="19200">
</ComChannel>
```

23.3.2 Instrukce pro sériové porty PLC

Funkce pro obsluhu sériového portu pro PLC program jsou implementovány jako 4 speciální instrukce.

Instrukce pro sdílený přístup	Popis
COM_SEND	Zařazení znaku do fronty pro vysílání na COM port
COM_SEND_STR	Zařazení řetězce do fronty pro vysílání na COM port
COM_RECV	Čtení znaku z příjmové fronty z COM portu
COM_CONTROL	Řízení COM portu z PLC

instrukce	**COM_SEND**

funkce **Zařazení znaku do fronty pro vyslání na COM port**

syntax **COM_SEND com**
COM_SEND com [, val]

1.parametr **„com"** **číslo logického COM kanálu**
2.parametr **„val"** **hodnota pro vyslání**

Instrukce slouží pro zařazení znaku do výstupní fronty, které jsou vysílány přes COM kanál na periferii. Znak se očekává v DR registru nebo ve 2. parametru instrukce. Pokud při řazení nevznikla chyba, například je plný buffer, instrukce vrátí v RLO registru hodnotu log.1. Pokud instrukce proběhla pořádku, tak RLO registr bude nastaven na hodnotu log.0.
Význam parametrů:

Parametr	Hodnota (příklad)	Význam
com	1	Logický COM kanál (ne fzyický)
val	xx	Znak pro vyslání (bajt)

Příklad:

```
LOD         SendChar          ;Znak pro vyslání
COM_SEND    1                 ;Zařazení znaku do fronty na log.kanál 1
JL1         Error             ;nezařazeno = plná fronta
                              ;zařazeno
```

instrukce	**COM_SEND_STR**

funkce **Zařazení řetězce do fronty pro vyslání na COM port**

syntax **COM_SEND com, poin, len**

1.parametr **„com"** **číslo logického COM kanálu**
2.parametr **„poin"** **pointr nebo název řetězce pro vyslání**
3.parametr **„len"** **počet znaků které se mají vyslat**

Instrukce slouží pro zařazení řetězce do výstupní fronty, která je vyslána přes COM kanál na periferii. Pointr na řetězec je definován názvem datové proměnné nebo návěštím u instrukci „str". Dále je nutno zadat délku dat pro vyslání.
Rozdíl oproti vícenásobnému použití instrukce COM_SEND je v tom, že systém zabezpečí vyslání řetězce najednou bez možné časové prodlevy mezi vysíláními znaky.
Pokud při řazení nevznikla chyba, například je plný buffer, instrukce vrátí v RLO registru hodnotu log.1. Pokud instrukce proběhla v pořádku, tak RLO registr bude nastaven na hodnotu log.0.

Význam parametrů:

Parametr	Hodnota (příklad)	Význam
com	1	Logický COM kanál (ne fzyický)
poin	xx	Název datové proměnné nebo pole výstupního bufferu. Parametr se předává instrukci odkazem (ne hodnotou). Může být také návěstí u instrukci „str".
len	xx	Počet znaků pro vyslání

Příklad:

```
BUF_COM:    str   20,'Pokus vysilani'
PACK_1:     DS    20


  COM_SEND_STR  1,BUF_COM,14  ;Zařazení řetězce do fronty na log.kanál 1
  JL1           Error         ;nezařazeno = plná fronta
                              ;zařazeno

  COM_SEND_STR  1, PACK_1,14  ;Zařazení bufferu z pole PACK_1
  JL1           Error         ;nezařazeno = plná fronta
                              ;zařazeno
```

instrukce	**COM_RECV**

funkce **Čtení znaku z příjmové fronty**

syntax **COM_RECV com**
COM_RECV com [,val]

1.parametr **„com"** **číslo logického COM kanálu**
2.parametr **„val"** **buňka pro příjem**

Instrukce slouží pro příjem znaku z fronty, které byly přijaty z periferii přes COM kanál. Instrukce po vykonání vrátí v datovém registru načtený znak (nebo bude naplněn 2. parametr instrukce). Pokud byla prázdná fronta a žádný znak se nepřijal, instrukce vrátí v RLO registru hodnotu log.1.
Význam parametrů:

Parametr	Hodnota (příklad)	Význam
com	1	Logický COM kanál (ne fzyický)
val	xx	Buňka pro příjem znaku (bajt)

Příklad:

```
COM_RECV    1                   ;Příjem znaku z COM kanalu
JL1         Neprijato           ;Skok, když se nic nepřijalo
STO         RecvChar            ;Přijmutí znak
```

instrukce **COM_CONTROL**

funkce **Řízení COM kanálu**

syntax **COM_CONTROL** **com, mode**

1.parametr **„com"** **číslo logického COM kanálu**
2.parametr **„mode"** **řízení**

Instrukce slouží pro některé operace s COM kanálem.

Význam parametrů:

Parametr	Hodnota (příklad)	Význam
com	1	Logický COM kanál (ne fzyický)
mode	CLEAR	Vynulování příjmové fronty
	RECEIVED	Počet znaků v příjmové frontě
	STATE	Načtení statusu COM kanálu

Význam hodnot pro dekódování stavu COM kanálu:

název	hodnota	význam
PLC_COM_ERROR	1	COM kanál je v chybě
PLC_COM_REQ	2	COM kanál není požadován a není nakonfigurován
PLC_COM_SEND_OVER	4	přetečení vysílacího bufferu
PLC_COM_RECV_OVER	8	přetečení příjmového bufferu

23.4 Instrukce pro sdílený přístup CAN kanálu

Sdílené použití CAN kanálu v PLC programu znamená, že PLC program používá CAN-BUS kanál společně se systémem, nebo alespoň pro jeho řízení používá systémové prostředky. Inicializaci CAN kontroléru a řízení vysílání a příjmu paketů také provádí systém. PLC program jen žádá a zařazení svých paketů do fronty pro vyslání a čte frontu přijmutích paketů.

Masky bitů pro vyhodnocování		
Symbol	**Maska**	**Význam**
CAN_ERR_OK	00h	Vše v pořádku
CAN_ERR_XMTFULL	01h	Plná fronta pro vysílání
CAN_ERR_RCVEMPTY	02h	Nepřijala se nová zpráva

23.4.1 Instrukce pro práci s CAN-BUSem se sdíleným přístupem

Funkce pro sdílenou obsluhu CAN-BUSu jsou do PLC programu implementovány jako 2 speciální instrukce.

Instrukce pro sdílený přístup	**Popis**
CAN_SEND	Zařazení paketu do fronty pro vysílání na periferii
CAN_RECV	Čtení paketu z příjmové fronty

instrukce	**CAN_SEND**

funkce **Zařazení paketu do výstupní fronty**

syntax **CAN_SEND can, pmsg**

1.parametr **„can"** **číslo CAN kanálu**
2.parametr **„pmsg"** **pointer na paket**

Instrukce slouží pro zařazení paketu do výstupní fronty paketů, které jsou vysílány přes CAN kanál na periferii. Instrukce po vykonání vrátí v datovém registru výsledek, který je složen z masek chyb (popsaných dříve). Pokud při řazení nevznikla chyba, vrácená hodnota bude CAN_ERR_OK (00h). pokud se paket do fronty nepodařilo zařadit (fronta je přeplněna), bude vrácená hodnota CAN_ERR_XMTFULL (01h)

Význam parametrů:

Parametr	**Hodnota (příklad)**	**Význam**
can	1, 2	Logický CAN kanál (CAN1, CAN2)
pmsg	CANBUFF_OUT	Pointer, který ukazuje na vyhrazené místo 12 bajtů. Návěští u instrukce STR, kde je místo pro paket

Příklad:

```
CANBUFF_OUT:      STR   12               ;výstupní paket

      CAN_SEND    2, CANBUFF_OUT         ;Zařazení paketu
      EQ          CAN_ERR_OK             ;Bylo zařazeno ?
      JL1         OK                     ;zařazeno
                                         ;nezařazeno
```

Podrobnější příklad bude uveden v závěru této kapitoly

instrukce	**CAN_RECV**

funkce **Čtení paketu z příjmové fronty (Nedoporučuje se používat)**

syntax **CAN_RECV can, pmsg**

1.parametr **„can“ číslo CAN kanálu**
2.parametr **„pmsg“ pointer na paket**

Instrukce je uvedena jen pro kompatibilitu a nedoporučuje se používat. Místo ní je zavedena instrukce **CAN_RECV_REQ**, které popis je dále.

Instrukce slouží pro čtení paketu z fronty paketů, které byly přijaty z periferii přes CAN kanál. Instrukce po vykonání vrátí v datovém registru výsledek, který je složen z masek chyb (popsaných dříve). Pokud při čtení nevznikla chyba, vrácená hodnota bude CAN_ERR_OK (00h). pokud se žádný paket pro PLC nepřijal, bude vrácená hodnota CAN_ERR_RCVEMPTY (02h)

Význam parametrů:

Parametr	Hodnota (příklad)	Význam
can	2	Logický CAN kanál (CAN2)
pmsg	CANBUFF_IN	Pointer, který ukazuje na vyhrazené místo 12 bajtů. Návěští u instrukce STR, kde je místo pro paket

Příklad:

```
CANBUFF_IN:       STR   12               ;vstupní paket

      CAN_RECV    2, CANBUFF_IN          ;Příjem paketu
      EQ          CAN_ERR_OK             ;Bylo přijato ?
      JL1         OK                     ;přijato
                                         ;nic se nepřijalo
```

Podrobnější příklad bude uveden v závěru této kapitoly

instrukce	**CAN_RECV_REQ**

funkce **Požadavek pro volání událostní procedury po přijmu paketu**

syntax **CAN_RECV_REQ can, objl, objh, idl, idh, proc, pmsg**

1.parametr	**„can"**	**číslo CAN kanálu**
2.parametr	**„objl"**	**minimální ID objektu zprávy**
3.parametr	**„objh"**	**maximální ID objektu zprávy**
4.parametr	**„idl"**	**minimální NODE-ID**
5.parametr	**„idh"**	**maximální NODE-ID**
6.parametr	**„proc"**	**název událostní procedury**
7.parametr	**„pmsg"**	**pointer na paket**

Instrukce má návratové hodnoty:

RLO=1	... operace dokončena bez chyb
RLO=0	... operace dokončena, ale požadavek nebyl akceptován
DR (DWRD)	... handle žádosti o volání událostní procedury

Instrukce slouží pro nastavení žádosti o volání událostní procedury v PLC po příjmu určitých „objednaných" paketů z CAN kanálu (nastavení „CALLBACK" volání). Když instrukce proběhne bez chyb (RLO=1) , tak si PLC program může uchovat „handle žádosti" do vlastní proměnné typu DWORD. Tuto proměnnou může PLC program využít v případě, že by později chtěl žádost o volání procedury zrušit.
Instrukce může být použita v modulu MODULE_INIT.

Význam parametrů:

Parametr	Hodnota (příklad)	Význam
can	1, 2	Logický CAN kanál (CAN1, CAN2)
objl	například 580h	Minimální ID objektu zprávy, které příjem se očekává. Jedná se o horní 4 bity z 11-bitové adresy COB-ID (podle normy CanOpen)
objh	-	Maximální ID objektu zprávy, které příjem se očekává. Parametr je nepovinný. Musí se uvést, když je požadován příjem intervalu zpráv. Pokud parametr nebude uveden (bude nahrazen znakem „-") tak se bude očekávat příjem zpráv jednoho objektu.
idl	například 21h	Minimální NODE-ID objektu zprávy, které příjem se očekává. Jedná se o 7 dolních bitů z 11-bitové adresy COB-ID.
idh	-	Maximální NODE-ID objektu zprávy, které příjem se očekává. Parametr je nepovinný. Musí se uvést, když je požadován příjem intervalu zpráv. Pokud parametr nebude uveden (bude nahrazen znakem „-") tak se bude očekávat příjem zpráv jednoho NODE-ID.
proc	například P_ON_CANRECV	Název událostní procedury, která se spustí po příjmu požadovaného paketu. Jedná se o standardní proceduru napsanou v PLC programu. Procedura po zavolání bude mít v DR registru nastaveno číslo logického CAN kanálu (1, 2).
pmsg	například sCANRECV	Pointer, který ukazuje na vyhrazené místo 12 bajtů. Návěští u instrukce STR (kde je místo pro paket) do kterého se naplní celý přijatý paket, který odpovídá požadovanému COB-ID.

Příklad bude uveden později.

Struktura COB-ID (**C**ommunication **Ob**ject **Id**entifier) podle normy CANopen :

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	x	x	x	x	x	**Function (0-0Fh)**				**NODE-ID (0-7Fh)**						

instrukce	**CAN_RECV_DEL**

funkce	**Zrušení požadavku pro volání událostní procedury přijmu paketu**	
syntax	**CAN_RECV_END**	**handle**
1.parametr	**„handle"**	**handle žádosti o volání událostní procedury**

Instrukce má návratové hodnoty:

RLO=1 ... operace dokončena bez chyb
RLO=0 ... operace dokončena, ale požadavek nebyl akceptován

Instrukce slouží pro zrušení žádosti volání událostní procedury v PLC po příjmu paketů z CAN kanálu. Jako parametr instrukce se musí zadat název proměnné typu DWORD, kde je uloženo „handle žádosti", které vrátila instrukce CAN_RECV_REQ.

Příklad:

Žádost o zachytávání odpovědi na SDO pakety (objekt 580h) pro jednotku INOUT08 s NODE-ID=21h.

```
;Buňka DWORD pro uložení Handle
dwHANDLE:   ds    4

;Místo pro přijatý paket:
sCANRECV:   STR   12


;Událostní procedura pro zpracování odpovědi na SDO paket
PROC_BEGIN  P_ON_CANRECV
      LOD   word.sCANRECV.CAN_DATA+1      ;Index zprávy
      EQ    cnst.1008h
      ;....
      ;....
PROC_END    P_ON_CANRECV


;Žádost o volání procedury P_ON_CANRECV
      CAN_RECV_REQ  2, 580h, -, 21h, -, P_ON_CANRECV, sCANRECV
      STO   dwHANDLE


;Zrušení žádosti o volání procedury
      CAN_RECV_END  dwHANDLE
```

23.4.2 Příklad pro vstupy a výstupy na nesystémovou CAN periferii se sdíleným přístupem

Příklad:
ID adresa periferie je nastavena na 10h.

V deklaraci dat:

```
CANBUFF_IN:       STR 12      ;misto na prijem paketu
CANBUFF_OUT:      STR 12      ;misto pro vysilani paketu
```

V modulu MODULE_MAIN:

```
      CAN_RECV 2,CANBUFF_IN                    ;Cteni paketu
      EQ          CAN_ERR_OK                   ;Je novy paket ?
      JL0         NIC_NEPRIJATO

;Test na prijem paketu ID=10h+1, delka=2, Byte1=Opc=1, Byte2=Input
      LOD         WORD.CANBUFF_IN.CAN_ID       ;Test ID
      ANDB        CNST.7FFh                    ;11 bitove ID
      EQ          CNST.10h+1                   ;Odpoved od ID=10h ?
      JL0         CAN_OTH
      LOD         BYTE.CANBUFF_IN.CAN_LEN      ;Delka
      EQ          CNST.2                       ;Je delka = 2 ?
      JL0         CAN_OTH
      LOD         BYTE.CANBUFF_IN.CAN_DATA+0   ;Opcode
      EQ          CNST.1                       ;Opcode 1 pro vstupy ?
      JL0         CAN_OTH
      LOD         BYTE.CANBUFF_IN.CAN_DATA+1   ;Sejmuti vstupnich dat !
      STO         IP_CAN                       ;Místo pro nova data
CAN_OTH:                                       ;Jiny paket, opakujeme

;**** Obsluha CAN-BUSu ... vysilani vystupu ****
      LOD         CNST.10h                     ;Vyslani na ID=10h
      STO         WORD.CANBUFF_OUT.CAN_ID
      LOD         CNST.0h                      ;Neni "remote request"
      STO         BYTE.CANBUFF_OUT.CAN_RTR
      LOD         CNST.2                       ;delka paketu
      STO         BYTE.CANBUFF_OUT.CAN_LEN
      LOD         CNST.2                       ;Opcode = 2
      STO         BYTE.CANBUFF_OUT.CAN_DATA+0
      LOD         IP0                          ;data pro vyslani
      STO         BYTE.CANBUFF_OUT.CAN_DATA+1

      CAN_SEND  2,CANBUFF_OUT                  ;Zarazeni paketu do fronty
```