Demopříklady v Kit-Basicu pro komunikaci s PLC firmy TECO

Příručka uživatele a programátora

***SofCon*®s.r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: (02) 20 180 454
e-mail: sofcon@sofcon.cz
www : http://www.sofcon.cz

Obsah :

1. Úvod

Firma *SofCon s.r.o.* vytvořila programové vybavení pro komunikaci s PLC zařízeními Tecomat a Tecoreg firmy Teco a.s., s jehož pomocí je možno připojit průmyslový terminál Term10 firmy *SofCon* ke zvolenému počtu PLC Tecomatů jako vizualizační či operátorský panel. Dále je možno připojit terminál Term10 (nebo i více terminálů) i jako podřízenou jednotku - Slave.

Na distribuční disketě je několik demonstračních příkladů na komunikaci se zařízeními Tecomat. Programy jsou psané v jazyce Kit-Basic pro prostředí Kit-Builder s grafickým terminálem Term10. Posílaná a čtená data mezi terminálem Term10 a PLC Tecomatem jsou zprostředkována pomocí sériové komunikace RS485 s protokolem Tecomat.

Všechny demopříklady používají komunikačních automatů naimplementovaných v tzv. „include“ souborech (.PRI) nebo v uživatelských procedurách Kit-Basicu (Proc3, Proc4 a Proc5, Proc6). Automaty se dělí na Master automaty pro obsluhu podřízených Tecomatů a Slave automaty pro odpovídání na dotazy nadřazených Tecomatů. Master automaty zajišťují navázání a udržování spojení s požadovaným počtem připojených PLC Tecomatů, čtení a zápis požadovaných registrů. Slave automaty zajišťují odpovídání na zprávy nadřízené stanice.

2. Programy pro PLC Tecomat pro jednotlivé demopříklady

Pro správný chod jednotlivých demopříkladů je třeba do PLC Tecomatu nahrát vždy určitý program .950.

Programy .950 pro jednotlivé demopøíklady:

```
DTECO1   .PRG \
DTECO2   .PRG - TECO_REG.950
DTECO3   .PRG /
DTECOIO  .PRG - TECO_DIO.950
DTECOIO2.PRG /
DTECOSL  .PRG - TECO_MAS.950
```

2.1. Popis programu TECO_REG.950

Program TECO_REG.950 definuje dva automaty implementující simulaci dvou jednoduchých P regulátorů. Princip obou regulátorů spočívá v postupném přibližování měřené hodnoty k požadované (po expinenciále). Druhý regulátor se od prvního liší v tom, že navíc ukládá postupně požadované a měřené hodnoty do dvou archivů (archiv měřených a archiv požadovaných hodnot). Proto vizualizační program v Term10 sledující průběh regulace může číst pouze právě měřenou a požadovanou hodnotu nebo i celý archiv měřených a požadovaných hodnot za určitou poslední dobu (např. pro grafické znázornění průběhu regulace).

2.2. Popis programu TECO_DIO.950

Program TECO_DIO.950 obsluhuje 8 digitálních vstupů a výstupů tak, že periodicky zapisuje hodnotu v regiastru R0 na výstupy a obraz vstupů ukládá do registru R1. Vizualizační program v Term10 periodicky zapisuje rotující danou hodnotu do registru R0 a zpětně čte obsah registru R1 - obě hodnoty zobrazuje na displeji. Pokud jsou u Tecomatu navzájem propojeny digitální vstupy s výstupy, je na signalizačních LED vstupů a výstupů vidět průběh komunikace (bez propojení vstupů a výstupů je průběh komunikace vidět jen na LED výstupů).

2.3. Popis programu TECO_MAS.950

Program obsluhuje 3 podřízené stanice s adresami 1, 2 a 3. Z první stanice periodicky čte R registry R0 až R9. Do druhé stanice periodicky zapisuje do R registrů R10 až R19. Ze třetí stanice zároveň čte R registry R0 až R9 a zároveň zapisuje do R registrů R10 až R19. Hodnoty čtených registrů třetí stanice pravidelně kopíruje do zapisovaných registrů, to znamená, že při úspěšné komunikaci by ve třetí podřízené stanici měl být obsah čtených i zapisovaných R registrů stejný.

3. Popis demonstračních příkladů

3.1. Stručný přehled všech demopříkladů

DTECO1 .PRG - Obsluhuje 1 podřízený PLC Tecomat, do jehož P regulátoru zapisuje požadovanou hodnotu. Zpětně čte archiv měřených a požadovaných hodnot, které potom zobrazuje ve formě časového grafu na displeji. Používá soubor **TECOM_1.PRI**.

DTECO2 .PRG - Podobný pøíklad jako DTECO1 - Obsluhuje 1 podøízený PLC Tecomat, do jehož P regulátoru zapisuje požadovanou hodnotu. Zpìtnì ète aktuální mìøenou a požadovanou hodnotu, které potom ukládá do archivù. Tyto dva archivy potom zobrazuje ve formì èasového grafu na displeji. Používá soubor **TECOM_1.PRI**.

DTECO3 .PRG - Podobný pøíklad jako DTECO2 - Obsluhuje 2 podøízené PLC Tecomaty, do jejichž P regulátorù zapisuje požadovanou hodnotu. Zpìtnì ète aktuální mìøenou a požadovanou hodnotu, které potom ukládá do archivù. Tyto archivy potom zobrazuje ve formì dvou èasových grafù na displeji. S obìma Tecomaty komunikuje stejné registry. Používá soubor **TECOM_2.PRI**.

DTECOIO .PRG - Obsluhuje 1 podøízený PLC Tecomat, do jehož registru R0 zapisuje požadovanou hodnotu digitálních výstupù. Tuto hodnotu periodicky rotuje, takže signalizaèní LED na digitálních výstupech Tecomatu svítí dle aktuálního stavu výstupù. Program zpìtnì ète registr R1, který obsahuje obraz digitálních vstupù. Obì hodnoty jsou potom zobrazovány na displeji terminálu. Používá soubor **TECOM_1.PRI**.

DTECOIO2.PRG - Podobný pøíklad jako DTECOIO s tím rozdílem, že pro komunikaci nepoužívá komunikaèní automat v daném .PRI souboru, ale používá komunikaèního Master automatu prostøednictvím uživatelských procedur Proc5 (inicializace komunikace) a Proc6 (zrušení komunikace).

DTECOSL .PRG - Odpovídá na ètení a zápis R registrù z nadøízeného Tecomatu - definuje pole 20-ti R registrù, z nichž prvních deset je pro ètení a zbylé jsou pro zápis. Hodnoty všech registrù zobrazuje na displeji terminálu. Registry pro ètení periodicky plní náhodnými hodnotami. Nadøízený Tecomat periodicky ète tyto hodnoty a zpìtnì je zapisuje do registrù pro zápis.

3.2. Podrobnìjší popis demonstraèních pøíkladù

Demopříklady DTeco1, DTeco2 a DTeco3 slouží pro vizualizaci dat daného zařízení PLC Tecomat. Všechny tyto demonstrační příklady zasílají do PLC Tecomatu požadovanou hodnotu, simulátor regulátoru v PLC Tecomatu podle této hodnoty upraví aktuální měřenou hodnotu, kterou pak pošle zpět terminálu Term10 jako výsledek. Průběh požadovaných a měřených hodnot je průběžne zaznamenáván do grafů a zobrazován na displeji terminálu Term10. Demonstrační programy prostřednictvím Master automatů zajišťují obsluhu požadovaného počtu připojených PLC Tecomatů, jako je navázání a udržování komunikace, zasílání a přijímání zpráv a jejich následné vyhodnocování.

Demopříklady DTecoIO a DTecoIO2 také slouží pro vizualizaci dat daného zařízení PLC Tecomat, v tomto případě se jedná o vizualizaci digitálních vstupů a výstupů připojeného Tecomatu.

Demopříklad DTeco-Sl slouží pro názorný příklad, jak z terminálu Term10 udělat podřízenou stanici.

3.2.1. DTeco1

Tento program ovládá pouze jeden PLC Tecomat prostřednictvím komunikačního automatu naimplementovaném v přilinkovaném include souboru TECOM1.PRI (viz 3.3 Popis a používání include souborů TECOM1.PRI a TECOM2.PRI). V hlavním souboru DTECO1.PRG se definují parametry komunikace (jako je například počet čtených a zapisovaných registrů) a obrazovky terminálu. Z připojeného PLC Tecomatu se čtou dvě hotová pole dat. Tyto pole slouží jako archiv požadovaných a měřených hodnot z regulátoru naimplementovaném v PLC Tecomatu programem TECO_REG.950. Velikost přenosu dat mezi terminálem Term10 a PLC Tecomatem se pohybuje okolo 200 bytů.

3.2.2. DTeco2

Tento program ovládá také pouze jeden PLC Tecomat prostřednictvím komunikačního automatu naimplementovaném ve stejném souboru TECOM1.PRI. V hlavním souboru DTECO2.PRG se definují (stejně jako v DTECO1.PRG) parametry komunikace a obrazovky terminálu, ale navíc se definují vedlejší procesy (automaty) na zpracování přečtených hodnot z PLC Tecomatu do polí grafů, blikání LED na panelu terminálu Term10 apod. Automaty na zpracovávání přečtených dat do polí požadovaných a měřených hodnot jsou obdobné automatům provádějící stejnou činnost v programu

TECO_REG.950 v PLC Tecomatu. Z toho vyplývá, že se nemusí po komunikační lince přenášet tolik dat pro grafické znázornění průběhu regulace. Velikost přenosu dat mezi terminálem Term10 a PLC Tecomatem je podstatně menší a to asi 10 bytů.

3.2.3. DTeco3

Tento program ovládá již dva PLC Tecomaty prostřednictvím komunikačního automatu naimplementovaném v přilinkovaném include souboru TECOM2.PRI (viz 3.3 Popis a používání include souborů TECOM1.PRI a TECOM2.PRI). Tento demostrační příklad je rozšířením příkladu DTECO2.PRG pro dva PLC Tecomaty se stejnou datovou strukturou. V hlavním souboru DTECO3.PRG se definují parametry komunikace, obrazovky terminálu a vedlejší procesy (automaty) na zpracování přečtených hodnot do polí grafů apod. Jak již bylo řečeno, oba dva připojené PLC Tecomaty mají stejný program i strukturu registrů, a proto jsou obrazovky terminálu Term10 pro oba dva PLC Tecomaty definovány společně, vždy se pouze v záhlaví dané obrazovky určí, který Tecomat je zobrazován. Velikost přenosu dat mezi terminálem Term10 a jedním PLC Tecomatem se pohybuje okolo 10 bytů.

3.2.4. DTecoIO

Tento program ovládá jeden PLC Tecomat prostřednictvím komunikačního automatu naimplementovaném v přilinkovaném include souboru TECOM1.PRI (viz 3.3 Popis a používání include souborů TECOM1.PRI a TECOM2.PRI). V hlavním souboru DTECOIO.PRG se definují parametry komunikace (jako je například počet čtených a zapisovaných registrů) a obrazovky terminálu. Do připojeného PLC Tecomatu se zapisuje požadovaná hodnota digitálních výstupů a zpětně se čte hodnota digitálních vstupů. Program dále definuje automat, který pro lepší názornost periodicky rotuje o 1 bit hodnotu zapisovanou do PLC Tecomatu na dig. výstupy. Velikost přenosu dat mezi terminálem Term10 a PLC Tecomatem se pohybuje do 10 bytů.

3.2.5. DTecoIO2

Tento program ovládá jeden PLC Tecomat prostřednictvím komunikačního Master automatu naimplementovaném přímo v prostředí KitBuilder (viz 3.5 Popis Master automatu a uživatelských procedur Proc5 a Proc6). V hlavním souboru DTECOIO2.PRG se definují parametry komunikace (jako je například počet čtených a zapisovaných registrů) a obrazovky terminálu. Do připojeného PLC Tecomatu se zapisuje požadovaná hodnota digitálních výstupů a zpětně se čte hodnota digitálních vstupů. Program dále definuje automat, který pro lepší názornost periodicky rotuje o 1 bit hodnotu zapisovanou do PLC Tecomatu na dig. výstupy. Velikost přenosu dat mezi terminálem Term10 a PLC Tecomatem se pohybuje do 10 bytů.

3.2.6. DTeco_Sl

Tento program prostřednictvím komunikačního Slave automatu (viz. 3.4 Popis Slave automatu a uživatelských procedur Proc3 a Proc4) odpovídá na zprávy nadřízené stanice. Komunikační Slave automat se spustí dle nastavených parametrů uživatelskou procedurou Proc3. Program definuje pole 20-ti R registrů, z nichž prvních deset je pro čtení

a ostatní jsou pro zápis. Na první obrazovce terminálu je zobrazen stav komunikace a počet přijatých požadavků na zápis či čtení všech druhů registrů (X, Y, S, R i DataBoxu). Na další obrazovce zobrazuje hodnoty všech dvaceti R registrů. Registry R pro ètení periodicky plní náhodnými hodnotami. Nadøízený Tecomat periodicky ète tyto hodnoty a zpìtnì je zapisuje do registrù pro zápis. To znamená, že pøi úspìšné komunikaci by mìl být obsah prvních deseti R registrù (ètené registry) v terminálu Term10 shodný s obsahem druhých deseti R registrù (zapisované registry).

3.3. Popis a používání include souborů TECOM1.PRI a TECOM2.PRI

Include soubory TECOM1.PRI a TECOM2.PRI definují automaty, které zajišťují navázání a udržování spojení s podřízenými PLC Tecomaty a čtení a zápis požadovaných registrů podle nastavených proměnných záznamu daného Tecomatu. Oba soubory se liší pouze v počtu obsluhovaných PLC Tecomatů. I když soubor TECOM1.PRI definuje automat obsluhující pouze jeden PLC Tecomat, je již naprogramován tak, aby jednoduchou změnou mohl obsluhovat i více PLC Tecomatů. Proto se soubory TECOM1.PRI a TECOM2.PRI liší jen minimálně a uživatel má tak názorný příklad, jak přidělat do některého z těchto souborů obsluhu dalších PLC Tecomatů. Pro usnadnění hledání těchto míst, jsou v souborech komentáře s následující syntaxí: "!Pro vice Tecomatu".

Pozn: Jak již bylo řečeno výše soubor TECOM1.PRI je použit v demopříkladech DTeco1, DTeco2 a DTecoIO, a soubor TECOM2.PRI je použit v demopříkladu DTeco3.

Tyto include soubory se do vlastního aplikačního programu přilinkují pomocí direktivy *include* (viz manuál Kit-Builder) a to zpravidla na jeho začátku. Je však před jejich přilinkováním nutné definovat určité konstanty a definice, které .PRI soubory vyžadují.

Přehled a význam konstant, které se musí v aplikaci definovat před přilinkováním vlastního .PRI souboru:

cTC_MaxPLC
: Konstanta definující počet obsluhovaných a maximálně připojených PLC stanic (pro include soubor TECOM1.PRI je rovna 1, pro include soubor TECOM2.PRI je rovna 2 apod).

cTC_SndMaxTOut
: Konstanta definující maximální dobu pro čekání do konce vysílání zprávy (timeout). Hodnota se udává v desítkách milisekund.

cTC_RecMaxTOut
: Konstanta definující maximální dobu pro čekání na odpověď (timeout). Hodnota se udává v desítkách milisekund.

cTC_TryConnect
: Konstanta definující prodlevu mezi testováním připojených PLC zprávou Connect. Hodnota se udává v sekundách.

cTC_SizeRead
: Konstanta definující velikost bufferu pro čtení registrů z PLC. Hodnota se udává v bytech a platí pro ni jistá pravidla:

  1. musí být dělitelná čtyřma

  2. musí být v intervalu 0 až 244

cTC_SizeWrite
: Konstanta definující velikost bufferu pro zápis registrů do PLC. Hodnota se udává v bytech a platí pro ni jistá pravidla:

  1. musí být dělitelná čtyřma
  2. musí být v intervalu 0 až 244

cTC_ParamStr
: Řetězcová konstanta s parametry komunikace (viz komunikační knihovny ChnXxx nebo Manuál Kit-Builder).

Include soubory .PRI dále definují konstanty a řídící proměnné pro oblsuhu Tecomatů, strukturu záznamů dat jednotlivých Tecomatů a procedury pro nastavování proměnných pro přístup k datům jednotlivých Tecomatů. Následuje popis těch konstant, proměnných, struktur a procedur, které může uživatel použít ve své aplikaci.

3.3.1. Struktura záznamu dat jednoho Tecomatu

Data jednoho Tecomatu jsou v paměti uložena ve formě symbolických názvů jednotlivých částí pole, které je alokováno v paměti (např: TC1_Addr, TC2_Addr atd.). Pravidla a příklady pro alokaci tohoto pole je možno nalézt v include souborech .PRI. Dále nebudeme toto pole nazývat jen obecně "pole", ale specifičtěji "záznam". Následuje seznam položek záznamu určených pro potřeby uživatele a aplikačního programu:

`TC_DNode = byte(TC1_Addr)[cTC_DNode];`
: Položka definuje adresu stanice PLC Tecomatu v síti.

`TC_FlConnect = byte(TC1_Addr)[cTC_Flconnect];`
: Položka definuje příznak připojení daného PLC Tecomatu.

`TC_SWritePerm = byte(TC1_Addr)[cTC_SWritePerm];`
: Položka definuje příznak permanentního vysílání zprávy pro zápis registrů. Pokud má položka hodnotu 1, bude se zpráva pro zápis registrů vysílat neustále, naopak (pokud má položka hodnotu 0) se bude zpráva pro zápis registrů vysílat pouze při nastaveném příznaku TC_FlSWrite.

`TC_FlSWrite = byte(TC1_Addr)[cTC_FlSWrite];`
: Pokud položka TC_SWritePerm má hodnotu 1, potom tato položka nemá význam. V opačném případě definuje příznak, zda se má (hodnota 1) či nemá (hodnota 0) vyslat jednorázově zpráva pro zápis registrů. Po případném vyslání zprávy pro zápis registrů se tato položka automaticky snuluje.

`TC_SReadCt = longint(TC1_Addr)[cTC_SReadCt];`
: Položka definuje čítač vyslaných zpráv pro čtení registrů.

`TC_RReadCt = longint(TC1_Addr)[cTC_RReadCt];`
: Položka definuje čítač přijatých odpovědí na zprávu pro čtení registrů.

`TC_SWriteCt = longint(TC1_Addr)[cTC_SWriteCt];`
: Položka definuje čítač vyslaných zpráv pro zápis registrů.

`TC_RWriteCt = longint(TC1_Addr)[cTC_RWriteCt];`
: Položka definuje čítač přijatých odpovědí na zprávu pro zápis registrů.

Položky vysílacího bufferu pro čtení registrů

`TC_SRead_Typ = byte(TC1_Addr)[cTC_SRead_Typ];`
: Položka definuje typ registrů, které se mají zpravou pro čtení registrů přečíst (viz

níže - konstanty typů registrů PLC Tecomatu).

`TC_SRead_Frst = byte(TC1_Addr)[cTC_SRead_Frst];`

Položka definuje adresu prvního registru, který se má přečíst z PLC Tecomatu. Po nastavení této položky je třeba zavolat proceduru TecoReInit (viz XXX) pro úpravu jiných vnitřních položek na této položce závislých.

`TC_SRead_Len  = byte(TC1_Addr)[cTC_SRead_Len];`

Položka definuje počet registrů (bytů), které se mají přečíst z PLC Tecomatu.

Položky přijímacího bufferu pro čtení registrů

`TC_RRead_Data = byte(TC1_Addr)[cTC_Read_Data];`

Položka definuje pole o velikosti cTC_SizeRead bytů (viz výše), kam jsou uloženy hodnoty přečtených registrů z PLC Tecomatu.

Položky vysílacího bufferu pro zápis registrů

`TC_SWrite_Typ = byte(TC1_Addr)[cTC_SWrite_Typ];`

Položka definuje typ registrů, které se mají zapsat do PLC Tecomatu (viz níže - konstanty typů registrů PLC Tecomatu).

`TC_SWrite_Frst= byte(TC1_Addr)[cTC_SWrite_Frst];`

Položka definuje adresu prvního registru, který se má zapsat do PLC Tecomatu. Po nastavení této položky je třeba zavolat proceduru TecoReInit pro úpravu jiných vnitřních položek na této položce závislých.

`TC_SWrite_Len = byte(TC1_Addr)[cTC_SWrite_Len];`

Položka definuje počet registrů (bytů), které se mají zapsat do PLC Tecomatu. Po nastavení této položky je třeba zavolat proceduru TecoReInit pro úpravu jiných vnitřních položek na této položce závislých.

`TC_SWrite_Data= byte(TC1_Addr)[cTC_Write_Data];`

Položka definuje pole hodnot registrů o velikosti cTC_SizeWrite bytů (viz výše) pro zapsání do PLC Tecomatu.

3.3.2. Konstanty typů registrů PLC Tecomatu

`cTC_RegX  = 0;`

Konstanta pro přístup k registrům X (registry vstupů) PLC Tecomatu.

`cTC_RegY  = 1;`

Konstanta pro přístup k registrům Y (registry výstupů) PLC Tecomatu.

`cTC_RegS  = 2;`

Konstanta pro přístup k registrům S (systémové registry) PLC Tecomatu.

`cTC_RegR  = 3;`

Konstanta pro přístup k registrům R (uživatelské registry) PLC Tecomatu.

`cTC_DBox0  = $80;`

Konstanta pro přístup k přídavné paměti DataBox adresy $00000 - $0FFFF.

`cTC_DBox1  = $81;`

Konstanta pro přístup k přídavné paměti DataBox adresy $10000 - $1FFFF.

`cTC_DBox2  = $82;`

Konstanta pro přístup k přídavné paměti DataBox adresy $20000 - $2FFFF.

`cTC_DBox3  = $83;`

Konstanta pro přístup k přídavné paměti DataBox adresy $30000 - $3FFFF.

`cTC_DBox4  = $84;`

Konstanta pro přístup k přídavné paměti DataBox adresy $40000 - $4FFFF.

`cTC_DBox5 = $85;`
Konstanta pro přístup k přídavné paměti DataBox adresy $50000 - $5FFFF.

`cTC_DBox6 = $86;`
Konstanta pro přístup k přídavné paměti DataBox adresy $60000 - $6FFFF.

`cTC_DBox7 = $87;`
Konstanta pro přístup k přídavné paměti DataBox adresy $70000 - $7FFFF.

3.3.3. Proměnné pro přístup k datům záznamů jednotlivých Tecomatů

`TC_ComTeco = byte;`
Proměnná definující číslo aktuálního PLC Tecomatu, s nímž se právě komunikuje. Aplikační program nesmí měnit její hodnotu.

`TC_ViewTeco = byte;`
Proměnná definující číslo aktuálního PLC, jehož registry (data) jsou zobrazovány (slouží pro nastavení adresy pro zobrazování).

`TC_ComAdr = word;`
Proměnná definující offsetovou adresu pro přístup k záznamům PLC Tecomatu, se kterým se komunikuje. Aplikační program nesmí měnit její hodnotu.

`TC_ViewAdr = word;`
Proměnná definující offsetovou adresu pro přístup k záznamům PLC Tecomatu, jehož hodnoty jsou zobrazovány. Aplikační prograj ji smí nastavovat pouze procedurou TC_SetViewTecoAdr (viz níže). Pokud aplikační program změní hodnotu této proměnné, měl by následně zavolat proceduru TC_SetViewTecoAdr (viz níže).

`TC_PomTeco = byte;`
Proměnná definující číslo aktuálního PLC Tecomatu pro potřeby aplikace.

`TC_PomAdr = word;`
Proměnná definující offsetovou adresu pro přístup k záznamům PLC Tecomatu, jehož číslo je v proměnné TC_PomTeco. Hodnota této proměnné se smí nastavovat pouze prostřednictvím procedury TC_SetPomTecoAdr (viz níže).

`TC1_Offs = word;`
Proměnná definující offsetovou adresu záznamu 1.Tecomatu. Podobně jsou definovány i offsetové adresy dalších záznamů Tecomatů. Aplikační program nesmí měnit hodnoty těchto adres.

Uživatel se v aplikačním programu odkazuje na jednotlivé položky záznamu daného Tecomatu buď pomocí přímé offsetové adresy záznamu daného Tecomatu (např: `TC_FlSWrite[TC1_Offs] := ...`), nebo pomocí jiné offsetové adresy, která se nastavuje podle proměnných s aktuálními čísly daného PLC Tecomatu (např: `TC_FlSWrite[TC_AdrView] := ...`).

3.3.4. Popis procedur

`procedure TC_SetViewTecoAdr;`
Podle proměnné TC_ViewTeco nastaví hodnotu offsetové adresy TC_ViewAdr na přímou offsetouvou adresu daného Tecomatu (TC1_Offs, TC2_Offs nebo atd.).

`procedure TC_SetPomTecoAdr;`

Podle proměnné TC_PomTeco nastaví hodnotu offsetové adresy TC_PomAdr na přímou offsetouvou adresu daného Tecomatu (TC1_Offs, TC2_Offs nebo atd.).

`procedure TecoInit;`

Tato procedura nastaví implicitně hodnoty všech položek záznamů všech Tecomatů. Je ji nutné zavolat v proceduře INIT, která se volá vždy po resetu, a to ještě před definováním vlastních hodnot jednotlivých položek!.

`procedure TecoReInit;`

Tato procedura provede úpravu vnitřních položek záznamů všech Tecomatů podle uživatelského nastavení veřejných položek (například podle počtu zapisovaných registrů do PLC Tecomatu upraví délku vysílané zprávy pro zápis registrů). Proto ji je nutné volat po změně hodnoty těch položek, které tuto vlastnost mají (viz výše Struktura záznamu dat jednoho Tecomatu).

3.3.5. Definování symbolických názvů registrů

V include souborech .PRI jsou definovány pro hodnoty čtených a zapisovaných registrů pouze pole bytů. Uživatel si může položky těchto polí (nejen byty ale i wordy, longinty atd.) pro přehlednost symbolicky pojmenovat.

Např:
```
MerenaTepl = word   (TC1_Addr)[cTC_Read_Data  + 0];
CisloMer   = longint(TC1_Addr)[cTC_Read_Data  + 2];
PozadTepl  = word   (TC1_Addr)[cTC_Write_Data + 3];
```

Symbolická proměnná MerenaTepl je typu word a je to první až druhý čtený registr z prvního PLC Tecomatu. Symbolická proměnná CisloMer je typu longint a je to třetí až šestý čtený registr z prvního PLC Tecomatu. Tato symbolická proměnná nemůže být definována v poli čtených registrů na místě [cTC_Read_Data +1], protože by se překrývala s druhým čteným registrem pro symbolickou proměnnou MerenaTepl. Symbolická proměnná PozadTepl je typu word a je to třetí až čtvrtý zapisovaný registr do prvního PLC Tecomatu.

3.4. Popis Slave automatu a uživatelských procedur Proc3 a Proc4

Slave automat se spouští (vìtšinou v INIT proceduøe vlastního programu) uživatelskou procedurou **Proc3**, která má tøi parametry. První parametr typu byte je poèátek inicializaèního øetìzce komunikace, který obsahuje adresu dané Slave stanice, adresu a èíslo pøerušení (IRQ) komunikaèního kanálu, pøenosouvou rychlost, paritu apod. Druhý parametr také typu byte je poèátek tabulky, jejíž položky urèují poèty a adresy pøístupných registrù pro ètení a zápis. Tøetí parametr také typu byte je poèátek záznamu, do jehož položek budou zaznaménávány informace o pøíchozích zprávách a stavu probíhající komunikace.

Slave automat se ukonèuje uživatelskou procedurou **Proc4**. Ve vìtšinì pøípadù není tøeba tuto proceduru volat (vìtšinou se Slave komunikace nainicializuje a bìží po celou dobu programu), jen pokud by uživatel chtìl tuto komunikaci v prùbìhu programu zrušit.

Proceduru **Proc3** je možno volat opakovanì. To znamená, že pokud uživatel chce nastavit napøíklad jinou komunikaèní rychlost, provede pøíslušnou úpravu øetìzce s parametry komunikace a zavolá proceduru Proc3. Ta provede zrušení dosavadní komunikace a její novou inicializaci dle nového nastavení. Lze to samozøejmì provést i pøes proceduru Proc4, a to následovnì:

```
  Proc4(...); {poèátek zrušení komunikace}
  repeat until ChnState=cCh_None; {èekání do úplného zrušení
                                   komunikace}
  Proc3(...); {nová inicializace}
```

3.4.1. Detailní popis jednotlivých parametrù procedury Proc3

1. parametr - inicializaèní øetìzec komunikace

Pøíklad deklarace a naplnìní inicializaèního øetìzce komunikace:

```
symbol
  ParamStr  = string:128;        {øetìzec parametrù komunikace}
  PocParStr = byte(ParamStr[0]); {poèátek øetìzce}
:
  ParamStr = "NOD=3 NAM=COM ADD=$2310 IRQ=3 BD=19200 BIT=8 STO=2
              PAR=E LRB=1000";
            {pøíklad naplòení øetìzce -
                  adresa stanice = 3
                  adresa COM portu = $2310
                  èíslo pøerušení = 3
                  komun.rychlost = 19200Bd
                  datových bitù = 8
                  stop bitù = 2
                  parita = sudá
                  vyrovnávací buffer pro pøíchozí znaky = 1000}
```

2. parametr - poèátek tabulky s poèty a adresami ètených a zapisovaných registrù

Pøíklad deklarace tabulky s informacemi o rozložení jednotlivých registrù v datové oblasti Kit-Builderu:

```
symbol
  TableReg = Word:96;         {definice tabulky v pameti -
                               nasleduje struktura polozek tabulky}
    BgI_X_Rd = TableReg[ 0]; {index na pocatek v dat.oblasti KB
                              pro cteni X registru}
    Poc_X_Rd = TableReg[ 2]; {delka dat.oblasti KB pro cteni X
                              registru}
    BgI_Y_Rd = TableReg[ 4]; {index na pocatek v dat.oblasti KB
                              pro cteni Y registru}
    Poc_Y_Rd = TableReg[ 6]; {delka dat.oblasti KB pro cteni Y
                              registru}
    BgI_S_Rd = TableReg[ 8]; {index na pocatek v dat.oblasti KB
                              pro cteni S registru}
    Poc_S_Rd = TableReg[10]; {delka dat.oblasti KB pro cteni S
                              registru}
    BgI_R_Rd = TableReg[12]; {index na pocatek v dat.oblasti KB
                              pro cteni R registru}
    Poc_R_Rd = TableReg[14]; {delka dat.oblasti KB pro cteni R
                              registru}
    BgI_1_Rd = TableReg[16]; {index na pocatek v dat.oblasti KB pro
                              cteni registru z 1.DataBoxu}
    Poc_1_Rd = TableReg[18]; {delka dat.oblasti KB pro cteni
                              registru z 1.DataBoxu}
    BgI_2_Rd = TableReg[20]; {index na pocatek v dat.oblasti KB pro
                              cteni registru z 2.DataBoxu}
    Poc_2_Rd = TableReg[22]; {delka dat.oblasti KB pro cteni
                              registru z 2.DataBoxu}
    BgI_3_Rd = TableReg[24]; {index na pocatek v dat.oblasti KB pro
                              cteni registru z 3.DataBoxu}
    Poc_3_Rd = TableReg[26]; {delka dat.oblasti KB pro cteni
                              registru z 3.DataBoxu}
    BgI_4_Rd = TableReg[28]; {index na pocatek v dat.oblasti KB pro
                              cteni registru z 4.DataBoxu}
    Poc_4_Rd = TableReg[30]; {delka dat.oblasti KB pro cteni
```

```
                                registru z 4.DataBoxu}
    BgI_5_Rd = TableReg[32]; {index na pocatek v dat.oblasti KB pro
                                cteni registru z 5.DataBoxu}
    Poc_5_Rd = TableReg[34]; {delka dat.oblasti KB pro cteni
                                registru z 5.DataBoxu}
    BgI_6_Rd = TableReg[36]; {index na pocatek v dat.oblasti KB pro
                                cteni registru z 6.DataBoxu}
    Poc_6_Rd = TableReg[38]; {delka dat.oblasti KB pro cteni
                                registru z 6.DataBoxu}
    BgI_7_Rd = TableReg[40]; {index na pocatek v dat.oblasti KB pro
                                cteni registru z 7.DataBoxu}
    Poc_7_Rd = TableReg[42]; {delka dat.oblasti KB pro cteni
                                registru z 7.DataBoxu}
    BgI_8_Rd = TableReg[44]; {index na pocatek v dat.oblasti KB pro
                                cteni registru z 8.DataBoxu}
    Poc_8_Rd = TableReg[46]; {delka dat.oblasti KB pro cteni
                                registru z 8.DataBoxu}
    BgI_X_Wr = TableReg[48]; {index na pocatek v dat.oblasti KB pro
                                zapis X registru}
    Poc_X_Wr = TableReg[50]; {delka dat.oblasti KB pro zapis X
                                registru}
    BgI_Y_Wr = TableReg[52]; {index na pocatek v dat.oblasti KB pro
                                zapis Y registru}
    Poc_Y_Wr = TableReg[54]; {delka dat.oblasti KB pro zapis Y
                                registru}
    BgI_S_Wr = TableReg[56]; {index na pocatek v dat.oblasti KB pro
                                zapis S registru}
    Poc_S_Wr = TableReg[58]; {delka dat.oblasti KB pro zapis S
                                registru}
    BgI_R_Wr = TableReg[60]; {index na pocatek v dat.oblasti KB pro
                                zapis R registru}
    Poc_R_Wr = TableReg[62]; {delka dat.oblasti KB pro zapis R
                                registru}
    BgI_1_Wr = TableReg[64]; {index na pocatek v dat.oblasti KB pro
                                zapis registru z 1.DataBoxu}
    Poc_1_Wr = TableReg[66]; {delka dat.oblasti KB pro zapis
                                registru z 1.DataBoxu}
    BgI_2_Wr = TableReg[68]; {index na pocatek v dat.oblasti KB pro
                                zapis registru z 2.DataBoxu}
    Poc_2_Wr = TableReg[70]; {delka dat.oblasti KB pro zapis
                                registru z 2.DataBoxu}
    BgI_3_Wr = TableReg[72]; {index na pocatek v dat.oblasti KB pro
                                zapis registru z 3.DataBoxu}
    Poc_3_Wr = TableReg[74]; {delka dat.oblasti KB pro zapis
                                registru z 3.DataBoxu}
    BgI_4_Wr = TableReg[76]; {index na pocatek v dat.oblasti KB pro
                                zapis registru z 4.DataBoxu}
    Poc_4_Wr = TableReg[78]; {delka dat.oblasti KB pro zapis
                                registru z 4.DataBoxu}
    BgI_5_Wr = TableReg[80]; {index na pocatek v dat.oblasti KB pro
                                zapis registru z 5.DataBoxu}
    Poc_5_Wr = TableReg[82]; {delka dat.oblasti KB pro zapis
                                registru z 5.DataBoxu}
    BgI_6_Wr = TableReg[84]; {index na pocatek v dat.oblasti KB pro
                                zapis registru z 6.DataBoxu}
    Poc_6_Wr = TableReg[86]; {delka dat.oblasti KB pro zapis
                                registru z 6.DataBoxu}
    BgI_7_Wr = TableReg[88]; {index na pocatek v dat.oblasti KB pro
                                zapis registru z 7.DataBoxu}
    Poc_7_Wr = TableReg[90]; {delka dat.oblasti KB pro zapis
                                registru z 7.DataBoxu}
    BgI_8_Wr = TableReg[92]; {index na pocatek v dat.oblasti KB pro
                                zapis registru z 8.DataBoxu}
    Poc_8_Wr = TableReg[94]; {delka dat.oblasti KB pro zapis
                                registru z 8.DataBoxu}

  PocTableReg = Byte(TableReg)[0]; {pocatek tabulky Table Reg}
```

3. parametr - poèátek záznamu pro ukládání informací o stavu komunikace

Pøíklad deklarace záznamu pro ukládání informací o pøijatých zprávách z nadøízené stanice a o stavu komunikace.

```
symbol
  TabStsCom = Word:6;
    FlgsRead = byte(TabStsCom)[0]; {priznaky prijeti zprav pro
                                    cteni}
      FlReadRegX  = FlgsRead.7;    {priznak prijeti zpravy na cteni
                                    X registru}
      FlReadRegY  = FlgsRead.6;    {priznak prijeti zpravy na cteni
                                    Y registru}
      FlReadRegS  = FlgsRead.5;    {priznak prijeti zpravy na cteni
                                    S registru}
      FlReadRegR  = FlgsRead.4;    {priznak prijeti zpravy na cteni
                                    R registru}
      FlReadDBox  = FlgsRead.3;    {priznak prijeti zpravy na cteni
                                    DataBoxu}
    FlgsWrite= byte(TabStsCom)[1]; {priznaky prijeti zprav pro
                                    zapis}
      FlWriteRegX = FlgsWrite.7;   {priznak prijeti zpravy pro
                                    zapis X registru}
      FlWriteRegY = FlgsWrite.6;   {priznak prijeti zpravy pro
                                    zapis Y registru}
      FlWriteRegS = FlgsWrite.5;   {priznak prijeti zpravy pro
                                    zapis S registru}
      FlWriteRegR = FlgsWrite.4;   {priznak prijeti zpravy pro
                                    zapis R registru}
      FlWriteDBox = FlgsWrite.3;   {priznak prijeti zpravy pro
                                    zapis DataBoxu}
    ChnState = byte(TabStsCom)[2]; {stav komunikace}
    ChnResult= word(TabStsCom)[4]; {status (chyby) komunikace}

  PocTabStsCom = Byte(TabStsCom)[0]; {pocatek tabulky TabStsCom}
```

Pøíklad volání procedury Proc3

```
Proc3(PocParStr,PocTableReg,PocTabStsCom);
```

3.4.2. Popis parametrù procedury Proc4

Tato procedura nevyžaduje principielnì žádné parametry, ale pro dodržení syntaxe uživatelských procedur, které mají tøi parametry, musíme v programu nìjaké parametry zadat. Napøíklad staèí zavolat `Proc4(B0,B0,B0);`

3.4.3. Stavy komunikaèního kanálu

Slave automat obsahuje nìkolik stavù komunikaèního kanálu.
Konstanty stavù komunikaèního kanálu:

```
  constant
    cCh_None       = 0; { kanal zavren }
    cCh_Open       = 1; { kanal se otvira }
    cCh_Connect    = 2; { kanal se otvira }
    cCh_Rec        = 3; { kanal otevren - ceka se na prijem zpravy }
    cCh_SendReply  = 4; { posila se odpoved }
    cCh_DisConnect = 5; { kanal se zavira }
    cCh_Close      = 6; { kanal se zavira }
```

Tyto konstanty jsou po zavolání procedury Proc3 navraceny v promìnné **ChnState** v záznamu **TabStsCom** (viz víše).

Na zaèátku je kanál ve stavu cCh_None.
Po zavolání procedury Proc3 se kanál snaží dostat pøes stavy cCh_Open a cCh_Connect do stavu cCh_Rec, kde probíhá èekání na pøíjem zprávy od nadøízené stanice. Po jejím pøijetí a správném dekódování se sestaví patøièná odpovìï a kanál pøejde do stavu cCh_SendReply, kde se tato odpovìï odvysílá. Po jejím odvysílání kanál pøejde opìt do stavu cCh_Rec.

3.5. Popis Master automatu a uživatelských procedur Proc5 a Proc6

Master automat se spouští (vìtšinou v INIT proceduøe vlastního programu) uživatelskou procedurou **Proc5**, která má tøi parametry. První parametr typu byte je poèátek tabulky s globálními parametry komunikace, který obsahuje maximální èasové zprodlevy (tzv. TimeOuty) pøi vysílání a pøíjmu jednotlivých zpráv, dále èasovou periodu pro pokus o navázání spojení s nepøipojenými podøízenými stanicemi a nakonec øetìzec s HW parametry komunikace. (Z programátorského hlediska se nejedná pøímo o øetìzec - string, ale jen o pole bytù, do kterého se pøíslušný øetìzec vloží) Tento øetìzec by mìl obsahovat adresu dané Master stanice, adresu a èíslo pøerušení (IRQ) komunikaèního kanálu, pøenosouvou rychlost, paritu apod. Druhý parametr také typu byte je poèátek tabulky, jejíž položky urèují poèty a adresy pøístupných registrù pro ètení a zápis. Tøetí parametr také typu byte je poèátek záznamu, do jehož položek budou zaznaménávány informace o pøíchozích zprávách a stavu probíhající komunikace.

Slave automat se ukonèuje uživatelskou procedurou **Proc6**. Ve vìtšinì pøípadù není tøeba tuto proceduru volat (vìtšinou se Master komunikace nainicializuje a bìží po celou dobu programu), jen pokud by uživatel chtìl tuto komunikaci v prùbìhu programu zrušit.

Proceduru **Proc5** je možno volat opakovanì. To znamená, že pokud uživatel chce nastavit napøíklad jinou komunikaèní rychlost, provede pøíslušnou úpravu øetìzce s parametry komunikace a zavolá proceduru Proc5. Ta provede zrušení dosavadní komunikace a její novou inicializaci dle nového nastavení. Lze to samozøejmì provést i pøes proceduru Proc6, a to následovnì:

```
Proc5(...); {poèátek zrušení komunikace}
repeat until ChnState=cCh_None; {èekání do úplného zrušení
                                 komunikace}
Proc6(...); {nová inicializace}
```

3.5.1. Detailní popis jednotlivých parametrù procedury Proc5

1. parametr - tabulka s globáními parametry komunikace

Tato tabulka obsahuje maximální doby (TimeOuty) pøi vysílání a pøijímání zpráv, øetìzec s HW parametry komunikace a periodu pro pokus o navázání spojení s nepøipojenými podøízenými stanicemi.

Pøíklad deklarace a naplnìní tabulky s globálními parametry komunikace:

```
symbol
  ComGlbRec = word:136;
    SndMaxTOut = word(ComGlbRec)[0]; {maximalni timeout v [ms] pro
                                      vysilani}
    RecMaxTOut = word(ComGlbRec)[2]; {maximalni timeout v [ms] pro
                                      prijem odpovedi}
    PerConnect = word(ComGlbRec)[4]; {perioda v [s] pro pokus o
                                      navazani spojeni nepripojene
```

```
                                        stanice}
    NumMsg     = byte(ComGlbRec)[6]; {pocet obsluhovanych zprav
                                      Master automatem}
    ParamStr   = byte(ComGlbRec)[7]; {pocatek retezce parametru
                                      komunikace - pro tento
                                      retezec je vyhrazeno 128 bytu
                                      vcetne prvniho(delkoveho)}

  PocComGlbRec = byte(ComGlbRec)[0]; {pocatek struktury ComGlbRec}

symbol
  PomS = string:128; {pomocny retezec pro HW parametry komunikace}
:
  SndMaxTOut :=  200; {ms}
  RecMaxTOut := 1000; {ms}
  PerConnect := 10;   {s}
  NumMsg     := 2;
  PomS       := "NOD=100 NAM=COM ADD=$2310 IRQ=3 BD=19200 BIT=8
                 STO=2 PAR=E LRB=1000";
            {pøíklad naplòení pomocného øetìzce -
                 adresa stanice = 100
                 adresa COM portu = $2310
                 èíslo pøerušení = 3
                 komun.rychlost = 19200Bd
                 datových bitù = 8
                 stop bitù = 2
                 parita = sudá
                 vyrovnávací buffer pro pøíchozí znaky = 1000}
  Move(PomS,ParamStr,PomS[0]+1);
            {pøesun obsahu pomocného øetezce do pole bytù
             vyhrazeného pro vlastní øetìzec HW parametrù komunik.}
```

2. parametr - poèátek pole se záznamy obsahujícími poèty a adresy ètených èi zapisovaných registrù

Délka tohoto pole je závislá na poètu obsluhovaných zpráv Master automatem (viz. promìnná NumMsg). Velikost záznamu jedné zprávy je 14 bytù. To znamená, že celková velikost pole se záznamy jednotlivých zpráv je NumMsg*14.

Pøíklad deklarace pole se záznamy s parametry jednotlivých zpráv:

```
constant
  SizeAllMsgRec = 2*14;
symbol
  MsgRec = word:SizeAllMsgRec; {deklarace pole pro 2 zpravy}
 {R}FlConnect  = byte(MsgRec)[0]; {priznak pripojeni dane stanice}
    RW_Perm    = byte(MsgRec)[1]; {typ a zpusob vysilani teto
                                   zpravy}
 {W}  MsgType  = byte(MsgRec)[1]; {typ zpravy - spodni 4 bity - viz
                                   konstanty cRead, cWrite...}
 {W}  FlPermWr = RW_Perm.4;       {priznak cyklickeho vysilani teto
                                   zpravy, jinak vyslat jen je-li
                                   priznak FlWrite = 1}
 {RW} FlWrite  = RW_Perm.5;       {priznak jednorazoveho vyslani
                                   teto zpravy, po jejim vyslani se
                                   snuluje, tento priznak ma vyznam
                                   jen je-li FlPermWr = 0}
 {W}DNo        = word(MsgRec)[2]; {adresa cilove stanice}
 {W}AdrRW      = word(MsgRec)[4]; {adresa bufferu pro cteni/zapis
                                   dat - index v dat.oblasti KB}
 {W}BegI       = word(MsgRec)[6]; {pocatek (adresa) v PLC
                                   ctenych/zapisovanych dat}
 {W}Poc        = byte(MsgRec)[8]; {pocet ctenych/zapisovanych bytu}
 {W}RegType    = byte(MsgRec)[9]; {typ ctenych ci zapisovanych dat
                                   - viz konstanty RegX,RegY...}
```

```
{R}CountS     = word(MsgRec)[10];{pocitadlo poctu vyslanych zprav}
{R}CountR     = word(MsgRec)[12];{pocitadlo poctu prijatych
                                  odpovedi}

PocMsgRec = byte(MsgRec)[0]; {pocatek pole MsgRec}
```

Na první pohled se mùže zdát, že definicí jednotlivých položek jsme nadefinovaly strukturu pouze prvního záznamu v poli zpráv. Ale není problém se pomocí indexù odkázat napøíklad na položku urèující typ komunikovaných dat 2. zprávy a to následovnì: RegType[14]. Obecnì se na nebitovou položku I-té zprávy odkážeme zpùsobem: Item[(I-1)*14] a na bitovou položku Item[(I-1)*14*8]. Tento rozdíl v odkazování na bitové a nebitové položky je z dùvodu rozdílné implementace v KitBuilderu.

Poznámka: Pøed jednotlivými položkami jsou symboly {R}, {W}, nebo {RW}, které znamenají následující:

R - tuto položku nastavuje komunikaèní automat
W - tuto položku nastavuje uživatel
RW- tuto položku nastavuje jak uživatel tak i komunikaèní automat

3. parametr - poèátek záznamu pro ukládání informací o stavu komunikace

Pøíklad deklarace záznamu pro ukládání informací o stavu komunikace.

```
symbol
  TabStsCom = Word:4;
    ChnState = byte(TabStsCom)[2]; {stav komunikace}
    ChnResult= word(TabStsCom)[4]; {status (chyby) komunikace}

  PocTabStsCom = Byte(TabStsCom)[0]; {pocatek tabulky TabStsCom}
```

Pøíklad volání procedury Proc5

```
Proc5(PocComGlbRec,PocMsgRec,PocTabStsCom);
```

3.5.2. Popis parametrù procedury Proc6

Tato procedura nevyžaduje principielnì žádné parametry, ale pro dodržení syntaxe uživatelských procedur, které mají tøi parametry, musíme v programu nìjaké parametry zadat. Napøíklad staèí zavolat `Proc6(B0,B0,B0);`

3.5.3. Stavy komunikaèního kanálu

Master automat obsahuje nìkolik stavù komunikaèního kanálu.
Konstanty stavù komunikaèního kanálu:

```
constant
  tCh_None       = 0; { kanal zavren }
  tCh_Open       = 1; { kanal se otevira }
  tCh_Connect    = 2; { kanal se otevira }
  tCh_Send       = 3; { pocatek vysilani }
  tCh_MsgConnect = 4; { pocatek zpravy Connect }
  tCh_SendingConn= 5; { probiha vysilani zpravy Connect }
  tCh_RecConn    = 6; { probiha prijem odpovedi na zpravu Connect }
  tCh_Sending    = 7; { probiha vysilani }
  tCh_Rec        = 8; { probiha prijem odpovedi }
  tCh_DisConnect = 9; { kanal se zavira }
  tCh_Close      =10; { kanal se zavira }
```

Tyto konstanty jsou po zavolání procedury Proc5 navraceny v promìnné **ChnState** v záznamu **TabStsCom** (viz víše).

Na zaèátku je kanál ve stavu tCh_None.
Po zavolání procedury Proc5 se kanál snaží dostat pøes stavy tCh_Open a tCh_Connect do stavu tCh_MsgConnect, kde zapoène cyklická obsluha všech nadefinovaných zpráv v poli MsgRec. Automat se nejprve snaží zprávou Connect (viz. definice protokolu Tecom) navázat s danou stanicí spojení a po jejím úspìšném navázání probíhá vysílání vlastní zprávy (stavy tCh_Send a tCh_Sending) a následné èekání na odpovìï (stav tCh_Rec). Poté se jde zpìt do stavu tCh_MsgConnect, kde se zapoène obsluha další zprávy.

4. Nìkolid rad na závìr

Pøed nahrátím programu (.BIN souboru) **do terminálu** Term10 **zkontroluje správné nastavení parametrù komunikace** v pøíslušném .PRG souboru (vìtšinou se jedná o textový øetìzec ze zaèátku programu), zejména adresy a èísla pøerušení IRQ komunikaèního kanálu.

Zkontrolujte správné nastavení komunikaèního kanálu v PLC Tecomatu (zda se jedná o režim PC nebo MAS, komunikaèní rychlost apod.).