microPEL

PESnet

POPIS PROTOKOLU SÍTĚ SYSTÉMU PES VERZE 3.38 - 3.56

03/2000

PESnet
Popis protokolu V3.38 - 3.56
03.2000
2. verze dokumentu

Změny a doplňky
proti 1. verzi dokumentu :
změny s ohledem na použití jazyka SIMPLE
změny s ohledem na řady MPC300, PES-K, PES-M

http://www.micropel.cz

Stručný popis funkce sítě PESNET

PESNET verze 3 je síť typu "multi-master, token-passing". Jinak řečeno : všechny stanice v síti jsou na stejné hierarchické úrovni, všechny mají oprávnění k vysílání dat a pokyn k vysílání (tzv. TOKEN) si navzájem předávají v logickém kruhu. I když stanice nemají žádná data k vysílání, síť běží "naprázdno" - stanice si neustále dokola předávají token.

Pokud dojde k narušení komunikace na síti (např. zkratem na vedení, vnucením log. úrovně nula na lince, nebo vlivem extrémně silného rušení), přeruší se provoz na síti a stanice začnou časovat tzv. "mrtvý čas", po jehož uplynutí se opět pokusí nastartovat provoz na síti. Tento čas je mimo jiné závislý i na adrese stanice (aby se zabránilo kolizím při restartu). Stanice která první dočasuje mrtvý čas do konce zahájí vysílání a začne zkoušet předání token na všechny síťové adresy dokud nedostane kladnou odpověď. Tento mechanismus funguje rovněž vždy po zapnutí stanic a zabezpečuje tak spolehlivý start sítě. Maximální mořná délka mrtvého času (závisí též na adresách použitých v síti) je asi 5 sekund. Pokud je právě zapnuta další stanice na již aktivní síti, trvá její zařazení do logického kruhu o něco déle (zpravidla do 20 sekund - závisí i na použité baudové rychlosti) - každá stanice totiž neustále prohledává volný prostor adres před sebou a toto prohledávání se nemůže dít příliš často, aby se zbytečně nesnižovala propustnost sítě.

Přenos dat po síti probíhá po malých kvantech - tzv. "rámcích". Každý rámec obsahuje adresu cílové stanice (lze použít i speciální globální adresu - pak je rámec určen všem stanicím v síti) a kontrolní součet. Pokud stanice detekuje při příjmu rámce chybu kontrolního součtu, ignoruje celý tento rámec. Interpreter STUPID chápe všechny síťové proměnné jako globální, všechny sdílené síťové bity M64 až M127 a 16-ti bitové proměnné D32 až D63 jsou tedy vysílány jako rámce s globální adresou.

Komunikace na RS485 je poloduplexní, pro každý přenos rámce vždy ten, který zahajuje vysílání (tj. ten, který vysílá hlavičku rámce) je MASTER a ten který přijímá je SLAVE. Přenos se uskutečňuje buď jednofázově - MASTER vyšle rámec a SLAVE nevyšle žádnou odezvu, nebo dvoufázově - MASTER vyšle rámec a SLAVE vyšle odpověď.

Protože síť je typu multi-master, po určité době (buďto po překročení limitu - max. 6 odvysílaných rámců, nebo naopak v případě, že není co vysílat) odevzdává stávající MASTER svoje oprávnění k vysílání dalšímu - vysílá rámec TOKEN na určitou adresu a kontroluje odezvu na lince. Pokud ve sledované době (tato doba trvá cca 3 znaky, závisí tedy na nastavené baudové rychlosti) zjistí aktivitu na lince (což znamená, že nový MASTER se chopil vesla), přechází do režimu SLAVE. Pokud je po celou tuto dobu linka neaktivní, zkouší stávající MASTER vysílat TOKEN na další a další adresy, dokud nenalezne někoho aktivního.

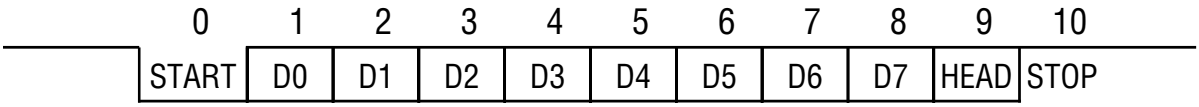
Přenosové vedení - fyzická vrstva

PESNET je používán na síťovém vedení RS485, tedy poloviční duplex a ovládání vysílání do linky (TX-ENABLE).

Způsob přenosu - linková vrstva

PESNET používá sériový asynchronní přenos s jedním startbitem, jedním stopbitem a devíti významovými bity (prvních osm obsahuje vždy přenášený datový byte a devátý bit označuje hlavičky rámců). Momentálně jsou podporovány přenosové rychlosti 2400, 9600, 19200 a 57600 Bd. Jednotlivé stanice nejsou schopny dynamicky měnit přenosovou rychlost, tato musí být nastavena shodně u všech stanic v síti.

Tvar značky :



START - vždy 0, začátek značky
HEAD - 1 pro hlavičku rámce, 0 pro všechny ostatní datové byty rámce
STOP - vždy 1, konec značky

Rámce

Veškerá data jsou přenášena v rámcích, rámce se dělí do 3 základních skupin:

a) Rámce pro předávání práva na vysílání (TOKEN)
b) Rámce pro přenos příkazů a čtení hodnot (CMDF, CMDRQ, REQUEST)
c) Rámce pro nastavování sdílených proměnných (SST_CLRB, SST_SETB, SST_WORD)

Každý rámec má tuto strukturu :

a) HEAD (jeden byte s nastaveným 9. bitem - udává typ rámce)
b) DATA (pole o různém počtu bytů - závisí na typu rámce)
c) SUM (jeden byte - součet modulo 256 všech bytů kromě SUM)

Výjimku tvoří rámec **REQUEST** - ten obsahuje pouze část HEAD, nic víc.

Typ rámce je vždy jednoznačně určen již 1. bytem - hlavičkou rámce. Hlavička rámce má (jako jediný byte) nastaven devátý bit na 1. Rámce mohou být dále chápány jako GLOBÁLNÍ (jsou adresovány všem stanicím v síti) nebo jako LOKÁLNÍ (jsou určeny pouze stanici s vybranou adresou). I rámec umožňující lokální adresování lze změnit na globální zadáním nejvyšší možné adresy (01FH tj. 31). Rámce pro nastavování sdílených proměnných jsou vždy globální, ostatní rámce jsou lokální a mají v hlavičkovém bytu vyhrazeno spodních 5 bitů pro nastavení adresy.

Kódování hlaviček :

01001111	SST_CLRB	
01011111	SST_SETB	
01101111	SST_WORD	
100aaaaa	CMDF	(Command with fast response)
101aaaaa	CMDRQ	(Command with response by request)
110aaaaa	TOKEN	
111aaaaa	REQUEST	
00xxxxxx	REZERVOVANO	

Pozn.: aaaaa = adresa stanice (00H-1FH), x = libovolná hodnota (zatím)

Odezva na rámec

Aby byla zajištěna alespoň částečná kontrola průběhu provádění příkazů, vysílá SLAVE po přijetí rámce CMDF nebo CMDRQ odezvu - buď určitý počet bytů dat v případě, že se jednalo o příkaz čtení dat, nebo, pokud se jednalo o příkaz nevyžadující odpověď - vysílá jako odpověď stavový byte ERROR (pokud je ERROR=0, je vše O.K.). Výjimku z příkazů CMDF a CMDRQ tvoří skupina příkazů IO - na ty není vysílána žádná odezva. Samozřejmě veškeré odezvy (tak jako veškeré rámce) jsou zakončeny jedním bytem kontrolního součtu (prostý součet modulo 256 přes všechny byty vysílané odezvy). Rámce vysílané na globální adresu (01FH) jsou vždy bez odezvy ! - až na jednu výjimku - příkaz Cm_GetACK. Ten vrací odezvu i na globální adrese aby bylo možné např. globálně v celé síti testovat nastavenou Baudovou rychlost apod. Při obracení směru vysílání (okamžik, kdy MASTER dovysílal rámec a SLAVE bude vysílat odpověď) čeká SLAVE vždy po dobu minimálně jednoho znaku (doba tedy závisí na nastavené rychlosti) než otevře vysílač a začne vysílat odpověď.

Je třeba mít na paměti logický fakt, že stanice při příjmu rámce s globální adresou nedává žádnou odezvu - nelze tedy nastavit globální adresu a pak vyčítat data.

Popis typů rámců

SST_CLRB (nulovaní sdíleného bitu M64…M127)

SST_SETB (nastavení sdíleného bitu M64…M127)

Formát rámce :

HEAD	ADDR	SUM

POZOR : Jako parametr ADDR se používá posunutá adresa bitu (kvůli rychlejšímu zpracování komunikační vrstvou) v intervalu 128-191 (odpovídá M64...M127)!

SST_WORD (nastavení sdílené 16-bitové proměnné D32…D63)

Formát rámce :

HEAD	ADDR	DATA Hi	DATA Lo	SUM

POZOR : Jako parametr ADDR se používá posunutá adresa proměnné (kvůli rychlejšímu zpracování komunikační vrstvou) v intervalu 96-127 (odpovídá D32...D63)!

TOKEN (předání oprávnění k vysílání)

Formát rámce :

HEAD	MY ADDR	SUM

Součástí hlavičky je adresa cílové stanice. TOKEN nelze předat na globální adresu. V datovém poli uvádí stávající MASTER svoji vlastní adresu (byte MYADDR, viz obrázek) - to je důležité pro eliminování kolizí stanic se stejnou adresou na síti, dále se tento parametr využívá při monitorování provozu sítě. V bytu MYADDR je platných pouze spodních 5 bitů adresy, horní 3 bity se mohou měnit.

Skupina rámců CMDF, CMDRQ a REQUEST

Rámce slouží pro ovládání automatu - zápis a čtení hodnot (RAM a EEPROM), dále pro předávání výkonných příkazů (RUN, STOP, STEP, RESET atd.)

Rámce CMDF a CMDRQ mají stejnou strukturu, přenášejí totožnou sadu příkazů, liší se pouze způsobem odezvy automatu. To, jestli rámec bude typu CMDF nebo CMDRQ je dáno pouze hlavičkou rámce. Na rámec typu CMDF odpovídá automat zcela standardně daným počtem bytů a na závěr datového pole odvysílá ještě kontrolní součet. Na rámec typu CMDRQ si automat připraví úplně stejnou odpověď, ale vyšle jen jeden byte. O další byte musí MASTER požádat vysláním rámce REQUEST. Celá odpověď proběhne takto zpomaleně byte po bytu. Tento podivný režim byl zaveden kvůli neschopnosti některých počítačů PC stíhat příjem dat na rychlosti 57600 Bd (a částečně kvůli neschopnosti autora napsat rychlý komunikační software pro PC).

Rámce CMDF a CMDRQ mají tuto strukturu :

Formát rámce :

HEAD	CMD	DATA ..... DATA	SUM

Struktura datového pole a struktura odezvy je daná bytem CMD (číslo příkazu) a je podrobně popsána níže - viz soubor příkazů.

Pro "krokování" odezvy u rámců CMDRQ je určen speciální rámec REQUEST (má jen jediný byte - hlavičku). REQUEST je lokální rámec a musí se vysílat vždy se správnou adresou stanice (jinak stanice nereaguje). Pokud stanice zachytí jakýkoli jiný rámec než REQUEST se správnou adresou, přechází do stavu "ABORT" a od toho okamžiku přestává reagovat na rámce typu REQUEST (až do příchodu dalšího rámce typu CMDRQ). Rovněž po odvysílání posledního bytu odezvy (tedy bytu SUM) přestává stanice reagovat na rámce typu REQUEST.

SOUBOR PŘÍKAZŮ (pro rámce typu CMDF, CMDRQ)

Příkazy se dále dělí do 2 skupin. Jsou to příkazy vykonávané automatem bezprostředně při jejich obdržení (značeny symboly Cm_xxx) a příkazy vyžadující určitý čas - jsou tedy vykonávány až následně (značeny symboly CmIO_xxx). Např. operace s EEPROM, řízení běhu, nastavení RTC apod.

Vzhledem k tomu, že příkazy CmIO_xxx se neprovádějí ihned, nedává na ně automat vůbec žádnou odezvu !!!

Práce s příkazy CmIO_xxx je poněkud náročnější. Aby nebylo třeba řešit problém front příkazů, parametrů a jejich přetékání, je v automatech pro vstup i výstup dat pro příkazy typu CmIO_xxx vyhrazen právě jeden buffer o kapacitě 6 bytů (tzv. IO-buffer). Informace o stavu tohoto vstupně/výstupního bufferu obsahuje (kromě jiného) stavový byte STATUS. Tento byte lze kdykoli přečíst příkazem **Cm_GetStatus**.Pokud potřebujeme provést akci typu CmIO_xxx, je třeba nejprve otestovat STATUS, zda je IO-buffer volný (bit ST_InBusy = 0), poté je možno vyslat sekvenci bytů pro vybraný příkaz CmIO_xxx a příkaz se v nejbližší možné době provede (typicky do 50 ms). Pokud potřebujeme, aby příkaz CmIO_xxx vrátil nějaká data, provedeme stejným způsobem předání příkazu a potom průběžně testujeme STATUS - bit ST_OutReady. Automat totiž po zpracování příkazu zapíše kýžená data opět do IO-bufferu a nastaví v bytu STATUS bit ST_OutReady na 1. Potom lze IO-buffer vyčíst již jednoduchým příkazem **Cm_GetIOBuf**. Dojde-li k vyslání příkazu CmIO_xxx do automatu aniž by byl buffer volný, je nastaven bit ST_Error v proměnné STATUS a zároveň je nastaven kód chyby do proměnné Error. Proměnná Error udržuje kód poslední chyby tak dlouho, dokud není vynulována příkazem Cm_ClrError. Chyba je rovněž hlášena po příkazu Cm_GetIOBuf, pokud v IO-bufferu nebyla k dispozici žádná data.

Příkaz Cm_GetIOBuf vrací jako odezvu tolik bytů, kolik přísluší k tomu kterému příkazu CmIO_xxx a na závěr samozřejmě SUM (kontrolní součet).

Pozn.: Pokud je stále IO-buffer "BUSY" pro vstup, může to být tím, že jsou v něm připravená data k vyzvednutí. Potom pomůže jednoduché "vybrání" dat příkazem Cm_GetIOBuf.

Význam jednotlivých bitů stavové proměnné STATUS :

BIT	SYMB. NÁZEV	VÝZNAM
0	ST_Error	Došlo k chybě - kód chyby je v proměnné Error
1	ST_InBusy	IO-buffer není volný pro vstup
2	ST_OutReady	V IO-bufferu jsou připravena žádaná data
4	ST_ErrorRAM	Nesouhlasí kontrolní součet programové paměti
6	SRun	Uživatelský program SIMPLE právě běží
7	SIdle	Uživatelský program SIMPLE momentálně pozastaven

SIMPLE - adresy proměnných, fyzické umístění v paměti

Při komunikaci s automatem je třeba znát absolutní adresy proměnných typu WORD nebo BIT a v některých případech (při používání blokových operací s pamětí) je třeba znát i fyzické umístění těchto oblastí v paměti.

SIMPLE počítá se základní velikostí paměti RAM cca 8kB. Bity jsou umístěny "zhuštěně" v oblasti 0200H...0227H. Bity jsou složeny v bytech tak, že vždy bit s nejnižší adresou z osmice je umístěn na pozici bitu LSB v bytu. Wordy jsou v oblasti 0000H...01FFH, jako první (na sudé pozici) je vždy byte MSB a druhý (na liché pozici) je LSB. Pro SIMPLE-kód je vyhrazen prostor v oblasti 0400H...13FFH a pro SIMPLE-STACK na ukládání uživatelských hodnot je vyhrazen prostor 1800H...1FFFH (tedy 2kB resp. 1024 wordů).

	zápis v jazyce SIMPLE	absolutní adresa	adresa RAM
BITY	dig. vstupy X0 ... X31	0 31	0200H 0203H
	dig. výstupy Y0 ... Y31	32 63	0204H 0207H
	vnitřní bity M0 ... M127	64 191	0208H 0217H
	spec. funkční bity B0 ... B127	192 ... 319	0218H 0227H
WORDY	anl. vstupy I0 ... I31	0 31	0000H 003FH
	anl. výstupy O0 ... O31	32 63	0040H 007FH
	proměnné D0 ... D63	64 127	0080H 00FFH
	spec. funkční proměnné W0...W127	128 ... 255	0100H 01FFH

STACK (zásobník)

Jeho umístění a práce s ním se poněkud liší podle typu automatu.

Starší modely PES-A, PES-T (již nevyráběné)

S verzí firmware 3.37 a nižší :

Bez ohledu na velikost paměti je STACK umístěn fyzicky od adresy 1800H a má velikost cca 1024 wordů (2048 B).

S verzí firmware 3.38 až 3.52 :

U modelů s 8kB paměti je STACK opět od 1800H, velikost 2048B.

U modelů s 32 kB RAM je možno STACK dynamicky přemístit - v automatu je v EEPROM uložena informace o jeho začátku. Po spuštění systému lze přes komunikační příkaz **Cm_GetMemCfg** dostat konfigurační byte paměti s touto strukturou :

bit MSB 0 = 8 kB RAM, 1 = 32 kB RAM

bity 0-7 stránka paměti, kde začíná stack (rozměr stránky je 256 bytů)

POZN.: U verze firmware nižší než 3.38 nelze stack dynamicky přemístit a není ani k dispozici příkaz Cm_GetMemCfg !

Modely PES-K, MPC300

Mají STACK vždy pevně od 1800H a s pevnou délkou 11776 položek typu WORD (tedy 23 552 B).

Kromě toho korektně poskytují informaci přes Cm_GetMemCfg.

Model PES-M66 (miniautomat)

Má STACK pevně od 1800H a s pevnou délkou 1024 položek typu WORD (tedy 2 048 B).

STRUČNÝ POPIS PŘÍKAZŮ PESNET

Symbolické názvy jednotlivých příkazů korespondují s názvy používanými v knihovnách PESlib. I když je dnes na programování používán výhradně jazyk SIMPLE, některé názvy příkazů jsou zachovány ještě z dob používání jednoduchého interpreteru jazyka STUPID.

Cm_GetACK

Jako odezvu vrátí číslo hex 055H. Slouží k identifikaci přítomnosti automatu na lince. Je to jediný příkaz, který vrací odezvu i na globální adrese. Lze použít k identifikaci provozní Baudové rychlosti automatu : zkusíme vyslat Cm_GetACK na různých rychlostech, tam kde příjem rámce bude O.K. (tedy bude souhlasit kontrolní součet SUM) vrátí automaty odezvu.

Pozn.: odezva je pouze na vlastní a na globální adrese, na cizí adresy automat žádnou odezvu nedává.

Cm_GetStatus

Vrátí byte STATUS.

Cm_GetNETVer

Vrátí celkem 6 bytů (ASCII kódů) tvořících 6-znakový text, označující verzi síťového protokolu.

Cm_GetSTPVer

Vrátí celkem 6 bytů (ASCII kódů) tvořících 6-znakový text, označující podporovanou verzi jazyka STUPID na automatu.

Cm_WriteXByte

Zapíše Byte do RAM automatu na adresu Addr (rozsah adres 0000-1FFFH)

Cm_ReadXByte

Přečte Byte z RAM automatu z adresy Addr (rozsah adres 0000-1FFFH)

Cm_Abort

Ukončí provádění příkazu (ať je to cokoliv)

Cm_SetXPtr

Nastaví pointer XPtr do RAM na hodnotu "Addr" - používá se pro zápisy a čtení s "auto-inkrementací" - viz následující příkazy.

Cm_WriteXPB

Zapíše Byte do RAM na adresu danou XPtr a zvětší XPtr o 1.

Cm_WriteXPW

Zapíše do RAM na adresu XPtr hodnotu ByteH, zvětší XPtr o 1, zapíše ByteL a zvětší XPtr o 1.

Cm_WriteXPDW

Podobně jako předchozí příkaz zapíše postupně Byte3 až Byte0.

Cm_ReadXPB

Přečte Byte z RAM z pozice XPtr a zvětší jej o 1.

Cm_ReadXPW

Přečte ByteH z RAM z pozice XPtr, zvětší jej o 1, přečte ByteL a opět posune XPtr.

Cm_ReadXPDW

Podobně jako předchozí příkaz vyčte postupně Byte3 až Byte0.

Cm_WriteSWord

Zapíše word (tedy ByteH a ByteL) do tabulky proměnných SIMPLE. Jako adresa proměnné (Addr) slouží absolutní adresa wordu - viz SIMPLE - tabulka adres proměnných. Jako Addr stačí 1 byte, neboť 16-bitových proměnných má SIMPLE celkem 256.

Cm_ReadSWord

Přečte word (tedy ByteH a ByteL) z tabulky proměnných SIMPLE. Jako adresa proměnné (Addr) slouží absolutní adresa wordu - viz SIMPLE - tabulka adres proměnných. Jako Addr stačí 1 byte, neboť 16-bitových proměnných má SIMPLE celkem 256.

Cm_SetSBit

Nastaví bit v tabulce proměnných jazyka SIMPLE. Jako adresa proměnné (AddrH a AddrL) slouží absolutní adresa bitu - viz SIMPLE - tabulka adres proměnných. Na Addr jsou potřeba 2 byty, neboť SIMPLE má celkem 320 bitových proměnných.

Cm_ClrSBit

Vynuluje bit v tabulce proměnných jazyka SIMPLE. Jako adresa proměnné (AddrH a AddrL) slouží absolutní adresa bitu - viz SIMPLE - tabulka adres proměnných. Na Addr jsou potřeba 2 byty, neboť SIMPLE má celkem 320 bitových proměnných.

Cm_ReadSBit

Přečte bit z tabulky proměnných jazyka SIMPLE. Jako adresa proměnné (AddrH a AddrL) slouží absolutní adresa bitu - viz SIMPLE - tabulka adres proměnných. Na Addr jsou potřeba 2 byty, neboť SIMPLE má celkem 320 bitových proměnných. Příkaz vrací buď 0 pokud byl bit nulový, nebo jakoukoli jinou hodnotu, byl-li bit nenulový.

Cm_ClrError

Vynuluje proměnnou Error a rovněž nuluje bit ST_Error v proměnné STATUS.

Cm_GetPIOName

Příkaz existuje jen u verzí firmware vyšších než 3.38

Tento komunikační příkaz slouží ke zjištění názvů všech implementovaných typů automatů, obsažených v EPROM. Jako parametr "Num" se zadává pořadové číslo driveru počínaje nulou a služba vrací 6 bytů ASCII - název driveru. Implementované drivery jsou seřazeny za sebou od nuly, všechny další pozice po posledním existujícím driveru vrací 6 nulových bytů.

U automatů MPC300, PES-K a PES-M je přítomen vždy právě jeden typ, na pozici 0.

Cm_GetError

Vrátí proměnnou Error (kód chyby, týkající se poslední operace) :

ERR_NONE	00	OK, žádná chyba
ERR_BADCMD	01	neznámý příkaz
ERR_CHECKSUM	02	chyba kontrolního součtu při příjmu rámce
ERR_BADFRAME	03	neznámý typ rámce
ERR_SELFRX	04	chyba při kontrole ozvěny vlastního vysílače
ERR_IOFULL	05	vstupní IO-buffer plný - rámec ztracen
ERR_NODATA	06	žádná data v IO-bufferu nejsou k dispozici

Cm_GetMemCfg

Příkaz existuje jen u verzí vyšších než 3.38

Konfigurace paměti - viz kap. Stack.

Cm_GetIOBuf
Vrátí IO-buffer, pokud v něm něco je.

CmIO_WriteEEB
Zapíše Byte do EEPROM na adresu Addr (rozsah adres 000 - 1FFH)

CmIO_ReadEEB
Předá do IO-bufferu Byte z EEPROM z adresy Addr (rozsah adres 000 - 1FFH)

CmIO_StupidRST
Provede inicializaci programu SIMPLE.

CmIO_StupidStep - *Příkaz zrušen !*

CmIO_StupidStop
Zastaví uživatelský program SIMPLE.

CmIO_StupidRun
Spustí uživatelský program SIMPLE.

CmIO_Reboot
Provede kompletní studený start automatu.

CmIO_SetTime
Nastaví čas do RTC

CmIO_SetDate
Nastaví datum do RTC, Week je den v týdnu (1..7).
Při používání funkce nastavení času z PC, kterou nabízí program SETPES, je třeba počít s tím, že : 1=Ne, 2=Po, 3=Út atd.

SOUHRN PŘÍKAZŮ

Závěrečná tabulka na následující straně zkráceně shrnuje všechny příkazy. Ve sloupcích "DATA" jsou vždy uvedeny příslušné byty tak, jak jdou za sebou.
POZOR ! v tabulce není uváděn byte kontrolního součtu (SUM), který následuje za daty rámce a byte SUM, který následuje za daty odezvy.

Symbolický název příkazu	VYSÍLANÝ RÁMEC					PŘIJÍMANÁ DATA (ODEZVA)					
	CMD	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA
Cm_GetACK	01					055H					
Cm_GetStatus	02					Status					
Cm_GetNETVer	03					Char5	Char4	Char3	Char2	Char1	Char0
Cm_GetSTPVer	04					Char5	Char4	Char3	Char2	Char1	Char0
Cm_WriteXByte	05	AddrH	AddrL	Byte		Error					
Cm_ReadXByte	06	AddrH	AddrL			Byte					
Cm_Abort	07					Error					
Cm_SetXPtr	08	AddrH	AddrL			Error					
Cm_WriteXPB	09	Byte				Error					
Cm_WriteXPW	10	ByteH	ByteL			Error					
Cm_WriteXPDW	11	Byte3	Byte2	Byte1	Byte0	Error					
Cm_ReadXPB	12					Byte					
Cm_ReadXPW	13					ByteH	ByteL				
Cm_ReadXPDW	14					Byte3	Byte2	Byte1	Byte0		
Cm_WriteSWord	15	Addr	ByteH	ByteL		Error					
Cm_ReadSWord	16	Addr				ByteH	ByteL				
Cm_SetSBit	17	AddrH	AddrL			Error					
Cm_ClrSBit	18	AddrH	AddrL			Error					
Cm_ReadSBit	19					Byte					
Cm_ClrError	20					Error					
Cm_GetError	21					Error					
Cm_GetIOBuf	22					(Byte)	(Byte)	(Byte)	(Byte)	(Byte)	(Byte)
Cm_GetPIOName	23*	Num				(Byte)	(Byte)	(Byte)	(Byte)	(Byte)	(Byte)
Cm_GetMemCfg	24*					Byte					
						data vrácená do IO-bufferu					
CmIO_WriteEEB	128	AddrH	AddrL	Byte							
CmIO_ReadEEB	129	AddrH	AddrL			Byte					
CmIO_StupidRST	130										
CmIO_StupidStep	131										
CmIO_StupidStop	132										
CmIO_StupidRun	133										
CmIO_Reboot	134										
CmIO_SetTime	135	Hour	Min	Sec							
CmIO_SetDate	136	Year	Month	Day	Week						