Jedno řešení sériové komunikace

František Dušek

KŘP FEI Univerzita Pardubice

Abstrakt

Článek se zabývá řešením měření a ovládání externího zařízení ze standardního PC v případě, kdy fyzickou komunikaci se zařízením lze realizovat přes sériový port (RS232). Řešení je použitelné i při použití USB portu na straně PC. Popisované řešení maximálně zjednodušuje použití na straně aplikace za cenu vytvoření speciálního interface k univerzální části knihovny, která řeší obsluhu komunikace se zařízením pomocí standardních služeb OS Windows.

Interface pro použití z prostředí MATLABu řeší problematiku používání externí knihovny (inicializace a odpojení knihovny, volání funkcí jádra knihovny) a zejména předávání dat tj. konverze standardních datových typů na datové struktury MATLABu. Interface řeší i předzpracování zpráv zařízení s ohledem na jednoduchost používání uživatelem V příloze článku je uveden zdrojový text interface (v jazyce „C“).

1 Úvod

Často – zejména v laboratorních podmínkách – se dostáváme do situace, kdy by bylo výhodné použít známé výpočetní prostředí (např. MATLAB, Excel, Visual Basic) pracující pod OS Windows nejen pro zpracování naměřených dat ze souboru, ale i pro měření a ovládání reálného zařízení. Kromě specializovaných programů typu LabView běžně používané programy neobsahují přímo podporu pro sběr dat a práci v reálném čase. Pokud příslušné výpočetní prostředí dovoluje používat dynamicky linkované knihovny (DLL) otvírá se cesta použít externí programy (knihovny), ať už vlastní nebo dodávané výrobcem hardware např. akviziční karty, pro rozšíření o měření. Použití dodaných knihoven vyžaduje, aby se na úrovni aplikace řešila jak komunikace s konkrétní knihovnou tak i kompletní komunikace se zařízením.

Na platformě Windows existuje také řešení využít výměnu procesních dat přes OPC server (dodává výrobce zařízení) a OPC klienta (součást aplikace). Toto univerzální profesionální řešení standardizuje komunikaci s různými zařízeními a umožňuje jednoduchou výměnu dat v síťovém prostředí. Nicméně použití tohoto univerzálního řešení pro jednoúčelové aplikace v laboratorních podmínkách není obvyklé.

V článku popsané řešení vychází z myšlenky umožnit velmi jednoduché používání komunikace v aplikaci a použít pouze standardně vybavené PC či notebook. Pod standardním PC se rozumí vybavenost sériovým portem a operačním systémem Windows 2000 či vyšším (podpora asynchronních V/V operací). V případě nepřítomnosti sériového portu je možné použít konvertor RS232/USB a k němu standardně dodávané programy pro realizaci virtuálního sériového portu. V tomto případě je možné využít fyzicky USB port na PC a programově s ním pracovat jako s dalším sériovým portem.

Vlastní řešení je tvořeno dvěma samostatnými částmi – programové vybavení na straně PC využívající sériový port a realizace sériové komunikace na straně zařízení. Programové řešení na straně PC představuje DLL knihovna obsahující univerzální část (jádro) a interface pro konkrétní aplikaci. Jádro knihovny se stará o na aplikaci nezávislou obsluhu komunikačního portu a příjem a vysílání datových zpráv. Všechny funkce jádra (kromě inicializace a ukončení) jsou asynchronní tj. při jejich volání se nečeká na realizaci požadavku.

Interface pro konkrétní aplikaci využívá funkce jádra knihovny k získání a odeslání konkrétních zpráv, zajišťuje jejich předzpracování s ohledem na jednoduché využití v aplikaci a zejména řeší datové konverze a další specifika konkrétní aplikace. Teprve na úrovni interface se řeší např. situace, kdy zařízení posílá data periodicky či data jsou posílána na až na základě vyslání požadavku. Konkrétní interface tedy závisí jak na typu aplikace tak i na konkrétním zařízení (způsob komunikace a konkrétní zprávy). Byly řešeny interface pro použití rozšířeného provedení laboratorního zařízení „Hydraulicko pneumatická soustava“ (HPS) z MATLABu, Excelu (Visual Basic for Aplication), PROMOTICu (Visual Ba-

sic Script) a volání z uživatelské aplikace v Borland C++. Základní informace o zjednodušené verzi HPS lze získat např. v [1], [2]. Použití rozšíření verze HPS včetně použití komunikační knihovny z MATLABu je popsáno v [3], [4].

Řešení na straně zařízení musí fyzicky realizovat port RS232 a programově řešit předávání dat prostřednictvím uvedeného protokolu. V případě HPS byla na straně zařízení použita řídicí jednotka (ŘJ) vybavená mikroprocesorem, která kromě komunikace zajišťuje vlastní měření, generování nestandardních ovládacích signálů, regulační a logické funkce a umožňuje lokální ovládání zařízení.

2 Koncepce komunikace se zařízením

V případě používání zařízení HPS z prostředí MATLABu bylo nutno provést:

Na straně PC návrh univerzální části

a) návrh jednoduchého komunikačního protokolu dovolujícího přenášet binární data v blocích různé délky
b) návrh a realizace jádra knihovny (práce s sériovým portem, asynchronní V/V operace, thread spouštěný při dokončení příjmu zprávy atd.)
c) návrh a realizace funkcí pro přístup k jádru knihovny

Na straně zařízení – rozšíření funkcí řídicí jednotky zařízení HPS

a) navrhnout formát a obsah jednotlivých zpráv přenášených po komunikační lince
b) rozšířit programové vybavení mikroprocesoru o část zajišťující periodickou přípravu dat zprávy (a iniciovat vyslání zprávy) a zpracování přijatých zpráv (reakci na data přijaté zprávy)
c) rozšířit programové vybavení mikroprocesoru o část zajišťující odeslání zprávy (zabalení dat do formátu zprávy a odeslání jednotlivých byte) a příjem zprávy (příjem jednotlivých byte a vyjmutí datového obsahu po úspěšném přijetí zprávy)

Návrh interface pro použití jádra knihovny z prostředí MATLABu a pro komunikaci se zařízením HPS, který splňuje požadavky:

a) jednoduché používání – specializované funkce umožňující získání měřených dat a zadání hodnot akčních veličin ve formátu přizpůsobeném účelu použití
b) rychlost – funkce nesmí zdržovat provádění programu v MATLABu
c) poskytovat možnost časování – synchronizace na periodicky přicházející data

2.1 Komunikační protokol a formát zpráv

Aby bylo možné posílat binární data, je používán jednoduchý kódově transparentní protokol (UCTP **U**niversal **C**ode **T**ransparent **P**rotocol). Tento protokol umožňuje pomocí řídicích znaků rozpoznat začátek a konec bloku binárních dat, která patří k sobě = datový obsah zprávy. Kromě toho umožňuje rozpoznání chyby při přenosu bloku dat pomocí kontrolního součtu. Jsou využívány pouze 3 řídicí znaky – **STX** (**S**tart of **T**e**X**t = 02h), **ETX** (**E**nd of **T**e**X**t = 03h) a **DLE** (**D**ata **L**ine **E**scape = 10h). Znak **DLE** je využíván jako prefix ostatních řídicích znaků tj. řídicí znaky se vyskytují vždy v dvojici. Pokud se v datech zprávy vyskytuje byte hodnoty **DLE**, je tento datový byte zdvojen tj. ve zprávě se objeví dva byte hodnoty DLE. Formát každé zprávy je potom tvořena následujícím rámcem

DLE	**STX**	**…**	**DLE**	**DLE**	**…**	kontr.suma	**DLE**	**ETX**

Sekvence **DLE**, **ETX** je povinný začátek zprávy, poté následují datové byte o libovolné hodnotě - pokud některý byte má hodnotu **DLE**, tak se zdvojí. Poslední byte před povinným ukončením zprávy **DLE**, **ETX** je byte kontrolní sumy. Tato hodnota je vypočtena prostým součtem datových byte zprávy (bez případných zdvojených hodnot **DLE**) tj. zabezpečení podélnou paritou. Zpráva je tedy delší minimálně o 5 byte + počet byte dat o hodnotě DLE než je počet původních datových byte.

Mezi ŘJ a dalším systémem se vyměňují informace pomocí zpráv, které v prvních dvou datových byte obsahují identifikaci zprávy. Aplikační zprávy obsahují data závislé na konkrétní aplikaci. Ze signálů rozhranní RS232 jsou využity signály RTS a CTS. Signál RTS (RTS na ŘJ) signalizuje funkčnost ŘJ. V případě HPS je základní zpráva (identifikační číslo 64) je posílána automaticky z ŘJ pokud je aktivní signál CTS rozhranní RS232 (CTS na ŘJ). Je tedy možné vysílání této zprávy zakázat či povolit.

2.1.1 Datové zprávy

Řídicí jednotka HPS každou sekundu posílá základní zpráva (Ident=64) s dvaceti aktuálními měřenými a stavovými hodnotami do PC. Data této 44 bytové zprávy mají následující význam

Tabulka 1 ZÁKLADNÍ DATOVÁ A STAVOVÁ ZPRÁVA (ŘJ → PC)

	B1	***B2***	***B3***	***B4***	***Hodnota***	***Ozn.***	***Význam***
00	0-127	0			64	Ident	základní datová a stavová zpráva
02	0-255	0-3			0-1023	AI01	JP1=P tlak v přívodním potrubí
04	0-255	0-3			0-1023	AI02	JP2=P_A atmosférický tlak
06	0-255	0-3			0-1023	AI03	JP3=P_L tlak v dolním pneumat. objemu
08	0-255	0-3			0-1023	AI04	JP4=P_H tlak v horním pneumat. objemu
10	0-255	0-3			0-1023	AI05	JP5=P_{LH} tlak (hladina) v levé horní nádrži
12	0-255	0-3			0-1023	AI06	JP6=P_{RH} tlak (hlad.) v pravé horní nádrži
14	0-255	0-3			0-1023	AI07	JP7=P_{LL} tlak (hladina) v levé dolní nádrži
16	0-255	0-3			0-1023	AI08	JP8=P_{RL} tlak (hladina) v pravé dolní nádrži
18	0-255	0-3			0-1023	AI09	Pot1=potenc. Po1 / ext. napětí 0-10 V
20	0-255	0-3			0-1023	AI10	Pot2=potenc. Po2 / ext. napětí 0-10 V
22	0-255	0-3			0-1023	AI11	Pot3=potenc. Po3 / ext. napětí 0-10 V
24	0-255	0-3			0-1023	AI12	Pot4=potenc. Po4 / ext. napětí 0-10 V
26	0-255	0-3			0-1023	AI13	REF02 (vnitřní teplota 2.1 mV / K)
28	0-255	0-255			???	C0	průtokoměr – pulsů / s
30	0-255	0-1			11110000,1	DI	Přepínače Př4-Př1, 0/1 – blokace ne/ano
32	0-255	0-3			0-1023	USe1	aktuální nastavení serva 1
34	0-255	0-3			0-1023	USe1	aktuální nastavení serva 2
36	0-255	0-3			0-1023	UCe	aktuální příkon čerpadla
38	0-255	0-3			0-1023	WTlak	aktuální hodnota žádané hodnoty PI tlaku
40	0-255	0-255	0-255	0	0-200000	Idle	indikace vytížení CPU (0% = 204265)

Datové zprávy (Ident = 1-4) posílané z PC do ŘJ obsahují informace o nastavení jedné nebo všech akčních veličin. Realizace a interpretace zasílaných hodnot závisí na nastavení ovládacích prvků na připojovacím a ovládacím panelu.

Tabulka 2 ZPRÁVA S NASTAVENÍM VŠECH TŘÍ AKČNÍCH VELIČIN (PC → ŘJ)

	B1	***B2***	***B3***	***B4***	***Hodnota***	***Ozn.***	***Význam***
00	0-127	0			4	Ident	nastavení všech akčních najednou
02	0-255	0-3			0-1023	Se1	hodnota polohy serva 1
04	0-255	0-3			0-1023	Se2	hodnota polohy serva 2
06	0-255	0-3			0-1023	CeP	hodnota Příkon / Žádaný tlak

Tabulka 3 ZPRÁVA S NASTAVENÍM AKČNÍ VELIČINY SERVO 1 (PC → ŘJ)

	B1	***B2***	***B3***	***B4***	***Hodnota***	***Ozn.***	***Význam***
00	0-127	0			1	Ident	nastavení polohy serva 1
02	0-255	0-3			0-1023	Se1	hodnota polohy serva 1

Tabulka 4 ZPRÁVA S NASTAVENÍM AKČNÍ VELIČINY SERVO 2 (PC → ŘJ)

	B1	***B2***	***B3***	***B4***	***Hodnota***	***Ozn.***	***Význam***
00	0-127	0			2	Ident	nastavení polohy serva 2
02	0-255	0-3			0-1023	Se2	hodnota polohy serva 2

Tabulka 5 ZPRÁVA S NASTAVENÍM AKČNÍ VELIČINY PŘÍKON / TLAK (PC → ŘJ)

	B1	***B2***	***B3***	***B4***	***Hodnota***	***Ozn.***	***Význam***
00	0-127	0			3	Ident	nastavení Příkon / Tlak
02	0-255	0-3			0-1023	CeP	hodnota Příkon / Žádaný tlak

Tabulka 6 ZPRÁVA ODBLOKOVÁNÍ URJ (PC → ŘJ)

	B1	*B2*	*B3*	*B4*	*Hodnota*	*Ozn.*	*Význam*
00	0-127	0			5	Ident	zpráva Odblokování

2.1.2 Uživatelské funkce pro MATLAB

V prostředí MATLABu má uživatel k dispozici formálně jednu funkci MatHPS, která realizuje (podle hodnoty prvního parametru) pět požadavků. Tato funkce představuje jediný vstupní bod (s povinným názvem **`void mexFunction`**) do knihovny (viz Kap. 3.3), která je v souboru MatHPS.mexw32 (nebo MatHPS.dll). Funkce MatHPS má minimálně jeden vstupní parametr (určení požadavku – rozhodující je první písmeno textu) a minimálně dva výstupní parametry (flg = příznak výsledku funkce, stav = stav knihovny). Hodnota 0 příznaku výsledku funkce flg indikuje bezchybné provedení požadavku. Kromě požadavku „Open" a „Close" ostatní požadavky nečekají na dokončení. Podrobnější popis významu parametrů viz výpis souboru MatHPS.m (help) v dodatku.

Předzpracovaný výběr (z posledních bez chyby přijatých dat) okamžitě vrací požadavek `[flg,stav,`***`mes`***`]=MatHPS(`**`'Read'`**`)` v proměnné ***mes***, která má následující pojmenované položky (strukturu):

Tabulka 7 DATOVÁ STRUKTURA PROMĚNNÉ OBSAHUJÍCÍ MĚŘENÁ DATA

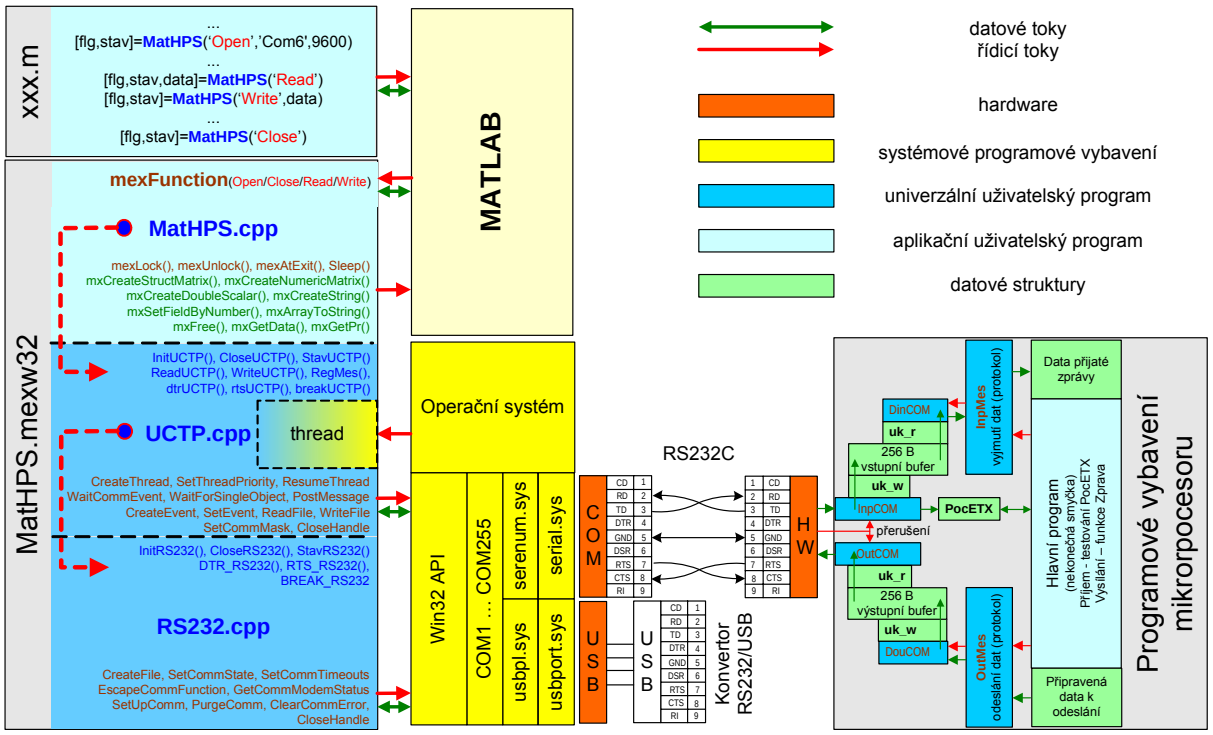
Položka	Rozsah	Význam
`P: []`	0-1023 = 0-100 kPa	tlak v přívodním potrubí
`PA:[]`	0-1023 = 80-120 kPa	absolutní atmosférický tlak
`PL:[]`	0-1023 = -7 +10 kPa	přetlak v dolním pneum. objemu
`PH:[]`	0-1023 = -7 +10 kPa	přetlak v horním pneum. objemu
`PLH:[]`	0-1023 = 0-30 kPa	dif. tlak v levé horní nádrži (hladina)
`PRH:[]`	0-1023 = 0-30 kPa	dif. tlak v pravé horní nádrži (hladina)
`PLL:[]`	0-1023 = 0-30 kPa	dif. tlak v levé dolní nádrži (hladina)
`PRL:[]`	0-1023 = 0-30 kPa	dif. tlak v pravé dolní nádrži (hladina)
`Pot1:[]`	0-1023 = 0-10 V	poloha potenciometru č.1
`Pot2:[]`	0-1023 = 0-10 V	poloha potenciometru č.2
`Pot3:[]`	0-1023 = 0-10 V	poloha potenciometru č.3
`Pot4:[]`	0-1023 = 0-10 V	poloha potenciometru č.4
`Prutok:[]`	0-65535	počet pulsů/s
`Prep1:[]`	'Local'/'Remote'	poloha přepínače ovládání servo 1
`Prep2:[]`	'Local'/'Remote'	poloha přepínače ovládání servo 2
`Prep3:[]`	'Local'/'Remote'	poloha přepínače ovládání čerpadlo
`Prep4:[]`	'Manual'/'Automat'	poloha přepínače stavu regulace
`Blokace:[]`	0/1	stav blokace
`Servo1:[]`	0-1023 = ±60°	aktuální hodnota nastavení serva č.1
`Servo2:[]`	0-1023 = ±60°	aktuální hodnota nastavení serva č.2
`Cerpadlo:[]`	0-1023 = 0-100%	aktuální hodnota příkonu čerpadla
`ZadTlakP:[]`	0-1023 = 0-25 kPa	akt. žádaná hodnota tlaku v přívod. potrubí

Po zavolání funkce jsou vybrané hodnoty z poslední přijaté zprávy vráceny okamžitě (nečeká se na nová data). Hodnota proměnné `flag` indikuje zda nová zpráva (data) nejsou (≠0) či jsou (=0) k dispozici. Tato vlastnost dovoluje jednoduchou synchronizaci s přijatými daty viz funkce `Mer.m`.

```
function data=Mer
% precti data s cekáním
flg=1;
while flg~=0,
  [flg,stav,data]=MatHPS('Read');
  pause(0.01);
end
return
```

3 Programové řešení

Programové řešení je tvořeno řešením na straně zařízení (rozšířením programového vybavení řídicí jednotky zařízení) a DLL knihovnou na straně PC. Tato knihovna má tři vrstvy. První se stará o komunikační port RS232, druhá má na starosti řízení komunikace pod OS tj. příjem a odeslání požadovaných dat ve formě zpráv a třetí vrstva je interface pro konkrétní aplikaci a konkrétní zařízení. První



Obrázek 1 Schéma programového řešení

dvě vrstvy jsou relativně univerzální. Blokové schéma celkového programového řešení s vyznačením názvů použitých funkcí (uživatelských i systémových) je na obr. 1

3.1 Princip řešení na straně zařízení

Řešení na straně zařízení vyžaduje UART (hardwarově řešený paralelně sériový převodník) a využití přerušení při přijmu znaku a dokončení odeslání znaku. V případě HPS byl použit mikroprocesor řady I51 (Philips 80C552 + externích 32 kB EPROM, 32 kB SRAM). Nicméně dále popsaná koncepce programového řešení byla použita i při realizaci komunikace s jiným zařízením vybaveným mikroprocesorem ATMEL AVR ATmega32 (2 kB SRAM, 32 kB flash ROM).

Základem programového řešení jsou dva kruhové vyrovnávací bufry o velikosti 256 byte, každý doplněný ukazatelem pro zápis do bufru `uk_w` a pro čtení z bufru `uk_r`. Tyto ukazatele jsou velikosti 1 byte, takže jejich přetečením při povyšování je velmi jednoduše zajištěna kruhovost bufru (a zároveň dána velikost bufru 256 B). U vstupního bufru je zapisovací ukazatel `uk_w` povyšován (a data ukládána do bufru) v obsluze přerušení od příjmu znaku (přerušení povoleno trvale). Zároveň je v obsluze přerušení kontrolována posloupnost dvojice znaků **DLE**, **ETX**. V případě indikace této posloupnosti (konec zprávy) je povýšen čítač počtu přijatých zpráv `PocETX`. V hlavním programu je tento příznak pravidelně testován a pokud je nenulový vyjme se zpráva pomocí čtecího ukazatele `uk_r` (zjistit začátek, odstranit zdvojené DLE, počítat kontrolní sumu a testovat konec zprávy) a datový obsah se zkopíruje do paměti přijatých dat. Potom podle významu prvních dvou datových byte (Ident) se data zpracují a čítač počtu přijatých zpráv se sníží.V případě požadavku hlavního programu na odeslání připravených dat je volána funkce, která pomocí zapisovacího ukazatele `uk_w` byte po byte zapíše do výstupního bufru včetně doplnění hlavičky, zdvojení datových byte hodnoty **DLE**, kontrolní sumy a konce zprávy. Při zápisu do výstupního bufru se kontroluje zda probíhá vysílání znaků. Pokud ne, tak se vysílání iniciuje. Data z výstupního bufru jsou pomocí čtecího ukazatele `uk_r` vybírána a vysílána v obsluze přerušení od dokončení odeslání znaku. V případě vyprázdnění výstupního bufru (rovnost hodnot čtecího a zapisovacího ukazatele) si obsluha přerušení svoje přerušení zakáže.

3.2 Princip řešení na straně PC

Na straně PC je řešení komplikovanější. Přímý přístup k obsluze přerušení a vstupním a výstupním bufrům není možný, ovládání je plně pod režií OS (ovladače sériového či USB portu). Program může použít pouze standardní služby OS (Win32 API). Např. přístup k datům je pomocí standardních funkcí API `ReadFile / WriteFile` (popis těchto a následujících funkcí API viz např. [6]).

Jádro knihovny má dvě vrstvy. První vrstva se zabývá komunikačním portem. Základem je „otevření“ zvoleného portu pomocí funkce `CreateFile` a nastavení jeho vlastností. Kromě velikosti vstupních a výstupních bufrů (`SetupComm`), omezení časového čekání – timeout (`SetCommTimeouts`), nastavení úrovně výstupních signálů rozhranní RS232 tj. DTR a RTS (`EscapeCommFunction`) se nastavuje množství specifických parametrů komunikace (struktura `DCB` a funkce `SetCommState`). Důležitá je zde položka . `EvtChar` struktury DCB – hodnota byte při jehož přijmu může dojít k vyvolání události. Je použita hodnota **ETX** tj. znak signalizující pravděpodobný konec zprávy. Pravděpodobný proto, že hodnota **ETX** neurčuje jednoznačně konec zprávy (mohou se vyskytnout binární data této hodnoty). Jednoznačný konec je posloupnost **DLE**, **ETX** a proto obsluha této události v druhé vrstvě musí tuto situaci řešit.

Při otevření portu je použita málo využívaná vlastnost – používat asynchronní operace (příznak `FILE_FLAG_OVERLAPPED`). To znamená, že např. funkce `ReadFile / WriteFile` pouze zadají požadavek a řízení se hned vrací do volajícího programu. Příslušný ovladač pak při výskytu události související se zadaným požadavkem nastaví synchronizační objekt typu `Event` do signalizovaného stavu (signaled state). Odkaz na tento objekt musí být zařazen v struktuře `OVERLAPPED`, která je součástí požadavku. Konkrétně základem obsluhy komunikace je vlastnost ovladače sériové linky generovat událost (`EV_RXFLAG`) při přijmu znaku definované hodnoty.

Druhá vrstva knihovna se zabývá řízením komunikace a vyjmutím / zařazením dat do komunikační zprávy. Požadavek na inicializaci **Init_UCTP()** nejprve iniciuje první vrstvu a pak vytvoří synchronizační objekt typu `Event` (`CreateEvent`). Dále vytvoří nový thread (`CreateThread`) a přidělí mu vyšší prioritu (`SetThreadPriority`). Dále nastaví (`SetCommMask`) na komunikačním portu požadavek při jakých událostech se má objekt `Event` sdružený s tímto požadavkem převést do signalizovaného stavu. Nastaveny jsou události:

`EV_BREAK`	detekován datový signál BREAK
`EV_CTS|EV_DSR`	došlo ke změně signálů rozhranní RS232 CTS nebo DSR
`EV_RXFLAG`	byl přijat a umístěn do bufru speciální znak (nastaven ETX)
`EV_TXEMPTY`	prázdný výstupní bufer
`EV_ERR`	přijat znak s chybou (parita, rámec, nevyjmutý předešlý znak)

Inicializace končí funkcí `ResumeThread`, která thread spustí. Programový kód threadu je součástí knihovny. Je tvořen cyklem testujícím na začátku příznak požadavku ukončení threadu. Tělo cyklu zahajuje volání funkce `WaitCommEvent` (nastav objekt `Event` do nesignalizovaného stavu) a funkce `WaitForSingleObject` (čekej na signalizovaný stav zadaného synchronizačního objektu). Návrat z této funkce je až při přechodu synchronizačního objektu do signalizovaného stavu. To znamená, že zajišťuje převedení threadu do čekacího stavu a jeho pokračování po výskytu události. Po pokračování se zjistí k jaké události došlo, provede se příslušná obsluha (např. vyčtení do té doby přijatých byte `ReadFile` a jejich zpracování) a cyklus se uzavře návratem na začátek. Thread zajišťuje automatickou aktualizaci (nezávisle na činnosti aplikace) informací o zařízení – udržuje aktuální obraz stavu signálů rozhranní RS232 a zejména datový obsah poslední bez chyby přijaté zprávy.

Požadavek na ukončení komunikace a uvolnění knihovny **Close_UCTP** znamená vyvolat ukončení práce s komunikačním portem v první vrstvě, nastavit příznak ukončení threadu a pomocí funkce `SetEvent` uvolnit jeho provádění. Následně se thread sám ukončí. Pak se uvolní všechny při inicializaci vyžádané systémové prostředky.

Požadavek na čtení dat **Read_UCTP()** pak pouze kopíruje aktuální data zařízení (poslední přijatá data) a přidá příznak zda tato data již byla / nebyla jednou čtena. Při požadavku na zápis dat **Write_UCTP()** se zajistí převedení dat do formátu zprávy, předání do výstupního bufru (`WriteFile`) a okamžitý návrat do volajícího programu.

Požadavek **RegMessage()** zajistí zaregistrování požadavku, aby po bezchybném příjmu zprávy byla z threadu poslána (`PostMessage`) do zadané fronty zpráv (uživatelského programu) zpráva zadaného čísla (oba údaje jsou parametry požadavku). Tato služba knihovny dovoluje, aby se v uživatelském programu (který musí zjistit handler příslušné fronty zpráv, zjistit nepoužívané číslo zprávy a zaregistrovat vlastní obsluhu nové zprávy) automaticky spouštěla zvolená funkce při příchodu nových dat.

Jde o využití standardního systému posílání zpráv OS Windows stejně jako když se např. v GUI aplikaci Windows po kliknutí myší na tlačítko na obrazovce spustí funkce sdružená s tímto objektem.

3.3 Řešení interface pro MATLAB

Protože výsledná knihovna bude používána z prostředí MATLAB musí splňovat určité formální požadavky. MATLAB je především prostředí pro výpočty a tím je dán jednoduchý (omezený) způsob volání externích programů (MEX file). Základní omezení je v tom, že externí program musí mít právě jeden vstupní bod (nazvaný **`mexFunction`**) a jméno funkce v MATLABu se musí shodovat se jménem souboru s programem. Toto omezení lze jednoduše překonat rozlišením požadované činnosti nikoliv jménem funkce ale hodnotami parametrů (viz syntaxy volání Kap. 2.1.2). Druhé omezení – nerozlišuje se první a další volání – se projeví zejména v našem případě. Proto se na úrovni interface musí zajistit inicializace knihovny (pouze) při prvním volání (`"Open"`) a její uzamčení (**`mexLock`**) v paměti po celou dobu používání funkcí knihovny. Uvolnění knihovny (**`mexUnLock`**) a povinné ukončení práce s komunikační knihovnou je možné až na základě explicitního požadavku aplikace (`"Close"`) nebo při nestandardním ukončení. Právě možnost nestandardního ukončení (a povinnost ukončení komunikace) vyžaduje registraci (**`mexAtExit`**) v interface definované ukončovací funkce (**`ExitFcn`**), která pak je automaticky volána při nestandardním ukončení.

Dalším požadavkem je forma souboru externího programu. Nejjednodušší cestou jak soubor v požadovaném formátu získat je použít příkaz MATLABu **`mex`**. Je-li použit s parametrem ***`-setup`*** umožní zvolit překladač (buď vestavěný LCC nebo některý z nainstalovaných na PC a MATLABem podporovaných). Vlastní vytvoření knihovny ve formátu požadovaném MATLABem (včetně překladu zdrojových textů) se zajistí příkazem:

```
mex MatHPS.cpp UCTP.cpp RS232.cpp
```

Výsledkem je soubor **MatHPS.mexw32** tj. (dynamicky připojovaná) knihovna pro použití z prostředí MATLABu. Soubor RS232.cpp obsahuje zdrojový kód první vrstvy a soubor UCTP.cpp obsahuje kód druhé vrstvy jádra knihovny. Název přípony (a vlastní formát) záleží na OS a verzi MATLABu. Uvedená přípona se generuje ve verzi MATLABu R2007b a 32-bitové verzi Windows XP. Bližší informace o používání a tvorbě externích programů včetně syntaxe použitých funkcí viz [5] (zejména část Matlab\External Interfaces).

Zdrojový kód v souboru MatHPS.cpp (viz Dodatek) obsahuje interface pro MATLAB. Přístup z prostředí MATLABu zajišťuje funkce MatHPS (viz Kap. 2.1.2). Interface využívá pouze některé funkce druhé vrstvy jádra knihovny. Z jádra knihovny – objekt třídy UCTP – jsou volány metody (funkce):

.Init_UCTP()	inicializace komunikace, vytvoření threadu
.RTS_UCTP()	nastavení signálu RTS rozhranní RS232 (pin 7) na hodnotu 0/1
.Stav_UCTP()	dotaz na stav
.Read_UCTP()	dotaz na poslední přijatou zprávu
.Write_UCTP()	požadavek na odeslání zprávy
.Close_UCTP	ukončení komunikace, zrušení threadu

V interface je zajištěna kontrola na přípustné parametry včetně informování o případné chybě (**`mexErrMsgTxt`**, **`mexPrintf`**). Z pohledu uživatele jsou základní požadavky (`"Read"`) a (`"Write"`). Při požadavku (`"Read"`) interface zajišťuje výběr některých hodnot z posledně přijaté zprávy Ident=64, jejich naplnění do datové struktury s pojmenovanými položkami a okamžitý návrat do volajícího programu. Při požadavku (`"Write"`) se data převedou na zprávu podle protokolu, předají na odeslání a bez čekání na vyřízení požadavku se řízení vrátí volajícímu programu. Kompletní komentovaný zdrojový kód je uveden v Dodatku.

4 Závěr

V článku je popsán jeden ze způsobů jak řešit komunikaci prostřednictvím sériové linky RS232 s externím zařízením pod operačním systémem Windows. Na straně zařízení je popsán princip programového řešení komunikace využívající možnost přímého přístupu k hardwarovým prostředkům použitého mikroprocesoru. Na straně PC je programové řešení rozděleno na část zajišťující obecnou část komunikace využívající funkce API Win32 jak pro přístup ke standardnímu sériovému portu tak i

k řešení synchronizace zpracování příchodu zpráv s jejich zpracováním. Druhá část (interface) je závislá na programovém prostředku, ze kterého chceme komunikaci využívat a na konkrétním zařízení. K této části jsou uvedeny jak poznámky k používání a vytváření externích programů (mex file) v MATLABu tak i konkrétní řešení (zdrojový kód) interface pro komunikaci s laboratorním zařízením HPS z prostředí MATLABu.

Konkrétní použití ve dvou aplikacích MATLABu je popsáno v článcích [3], [4].

Tato práce byla realizována v rámci projektu MSM 0021627505 v části "Řízení, optimalizace a diagnostika složitých systémů".

5 Literatura

[1] HONC,D .; DUŠEK,F.: *Hydraulic-pneumatic system - technical data.* In: 6th International Scientific-Technical Conference Process Control 2004, Kouty nad Desnou 8-11, June 2004, Czech Republic, University of Pardubice, 2004, p. 265, ISBN 80-7194-662-1 (plný text 6 stran na doprovodném CD)

[2] MACHÁČEK, J.; HONC, D.; DUŠEK, F.: *Výukový laboratorní model hydraulicko-pneumatické soustavy.* Automa 8-9, 2005, s. 108-109

[3] DUŠEK,F.; HONC,D.: *Automatizovaný experiment v MATLABu.*In: 12. Konference MATLAB 2004, 4.11. 2004, Kongresové centrum ČVUT, Praha, s. 97-104 (díl I.), ISBN 80-7080-550-1

[4] DUŠEK, F.; ŠKRABÁNEK, P. ; MAREŠ, J.: *Experimentální identifikace stavového modelu.* [v tomto sborníku]

[5] MATLAB on-line dokumentace

[6] Microsoft SDK – funkce API Win32

doc. Ing. František Dušek, CSc.
Fakulta elektrotechniky a informatiky
Katedra řízení procesů
nám. Čs. legií 565
53210 Pardubice
frantisek.dusek@upce.cz

6 Dodatek

6.1 Obsah souboru s nápovědou

```
% MatHPS.DLL    modul pro spojeni modelu hydraulicko-pneumatické
%   soustavy (HPS) s MATLABem prostřednictvím sériové linky RS232.
%   Formát zpráv je pevně dán a pro vlastni přenos je použit kódově
%   transparentní protokol.
%   Použiti modulu z MATLABu:
% *********     povinna inicializace komunikace
%   [flg,stav]=MatHPS('Open',port,rychlost)
%       flg..... = 0 (inicializace O.K.), <>0 (chyba)
%       stav..... = (struktura) stavová slova UCTP (UINT32)
%       port..... název portu (např. 'COM16')
%       rychlost. přenosová rychlost (150,300,..,115200)
% *********     povinné ukončeni
%   [flg,stav]=MatHPS('Close')
%       flg..... = 0 (uzavřeni O.K.), <>0 (komunikace nebyla inic.)
%       stav..... = (struktura) stavová slova UCTP (UINT32)
% *********     dotaz na stavová slova modulu
%   [flg,stav]=MatHPS('Stav')
%       flg..... = 0
%       stav..... = (struktura) stavová slova UCTP (UINT32)
% *********     dotaz na poslední přijatou zprávu
%   [flg,stav,mes]=MatHPS('Read')
%       flg..... = 0 (zpráva O.K.) <> 0 (žádná zpráva)
%       stav..... = (struktura) stavová slova UCTP (UINT32)
%       mes..... = (struktura) přijatá zpráva
% *********     odesláni zprávy
%   [flg,stav]=MatHPS('Write',mes)
%       flg..... = 0 (zápis O.K.) <>0 (chyba)
%       stav..... = (struktura) stavová slova dll_UCTP (UINT32)
%       mes..... vektor odeslané zprávy
%             =[1,0-1023] ... nastav servo 1
%             =[2,0-1023] ... nastav servo 2
%             =[3,0-1023] ... nastav čerpadlo/žádanou tlaku
%             =[4,0-1023,0-1023,0-1023] ... nastav vše
%             =[5] ... odblokuj
```

6.2 Zdrojový text interface pro prostředí MATLAB

Standardní (Borland C) barevné rozlišení zdrojového textu (první tři položky) je doplněno o další barevné odlišení názvů funkcí a struktur MATLABu a druhé vrstvy knihovny.

normální zelená	komentáře
normální červená	řetězce
normální modrá	klíčová slova jazyka „C“
tučná zelená	předdefinované struktury MATLABu (mex.h)
tučná červená	funkce MATLABu (prototypy z mex.h)
tučná černá	lokální pomocné funkce
tučná šikmá černá	funkce (metody) druhé vrstvy knihovny (UCTP.h)
šikmá černá	struktury knihovny (UCTP.h)

Vstupním bodem do interface je volání funkce **mexFunction**, jejichž parametry dovolují zjistit kolik vstupních parametrů (hodnota **nlhs**) a jakého typu (odkaz na pole ***plhs[]**) a kolik výstupních parametrů (hodnota **nrhs**) a jakého typu (odkaz na pole ***prhs[]**) má volající funkce z MATLABu.

```
/* MatHPS.DLL     modul pro komunikaci hydraulicko-pneumatické soustavy
 (HPS)s MATLABem prostrednictvim seriove linky RS232. Format zprav je pevne
 dan a pro vlastni prenos je pouzit kódove transparentni protokol (UCTP).
 Pouziti modulu z MATLABu:
 [flg,stav]=MatHPS('Open',port,rychlost)  povinna inicializace komunikace
 [flg,stav]=MatHPS('Close')         povinne ukonceni
```

```
 [flg,stav]=MatHPS('Stav')          dotaz na stavova slova modulu
 [flg,stav,mes]=MatHPS('Read')      dotaz na posledni prijatou zpravu
 [flg,stav]=MatHPS('Write',mes)     odeslani zpravy
      další moduly      +UCTP.CPP+RS232.CPP     */

#include    <windows.h>
#include    <winbase.h>
#include    "e:\Matlab7\extern\include\mex.h"
#include    "UCTP.H"
extern void _main();
static      UCTP  o;                        // vytvor objekt UCTP
static int  Init=0;                         // priznak inicializace komunikace

// pomocne vnitrni funkce ================================================
static void ExitFcn(void)
// volana MATLABem pri jeho ukonceni
{     if (Init)
         {o.RTS_UCTP(0);                    // nastav RTS na LOW
          o.Close_UCTP();                   // ukonci cinnost
          Init=0;                           // priznak ukonceni komunikace
          mexUnlock();};                    // odemkni MEX v pameti
}                                   // end of ExitFcn

mxArray* Status(void)
// vytvoreni struktury a naplneni hodnotami stavu
{ unsigned long int     StavPort=11,StavRS232=22,StavUCTP=33,BInBuf=44;
  mxArray               *tmp,*out;          // ukazatele na matice MATLABu
  const char* stav[]={"StavPort","StavRS232","StavUCTP","BInBuf"};

  o.Stav_UCTP(&StavPort,&StavRS232,&StavUCTP,&BInBuf);
  // vytvor strukturu
  out = mxCreateStructMatrix(1, 1, 4, stav);
  // naplneni polozek struktury
  tmp = mxCreateNumericMatrix(1, 1, mxUINT32_CLASS, mxREAL); // vytvor pole
  *(DWORD *) mxGetData(tmp)=StavPort;     // napln pole dle ukazatele
  mxSetFieldByNumber(out,0,0,tmp);        // napln polozku strukturu polem
  tmp = mxCreateNumericMatrix(1, 1, mxUINT32_CLASS, mxREAL); // vytvor pole
  *(DWORD *) mxGetData(tmp)=StavRS232;    // napln pole dle ukazatele
  mxSetFieldByNumber(out,0,1,tmp);  // napln polozku strukturu polem
  tmp = mxCreateNumericMatrix(1, 1, mxUINT32_CLASS, mxREAL); // vytvor pole
  *(DWORD *) mxGetData(tmp)=StavUCTP;     // napln pole dle ukazatele
  mxSetFieldByNumber(out,0,2,tmp);        // napln polozku strukturu polem
  tmp = mxCreateNumericMatrix(1, 1, mxUINT32_CLASS, mxREAL); // vytvor pole
  *(DWORD *) mxGetData(tmp)=BInBuf;       // napln pole dle ukazatele
  mxSetFieldByNumber(out,0,3,tmp);        // napln polozku strukturu polem
  return(out);                          // end of Status
}

mxArray* Zprava(mes_UCTP *z)
// vytvoreni struktury a naplneni hodnotami datove zpravy
{ int       i;
  mxArray   *tmp,*out;  // ukazatele na matice MATLABu
  const char*     data[]={"P","PA","PL","PH","PLH","PRH","PLL","PRL",
                  "Pot1","Pot2","Pot3","Pot4","Prutok",
                  "Prep1","Prep2","Prep3","Prep4","Blokace",
                   "Servo1","Servo2","Cerpadlo","ZadTlakP"};
  short int*      bufer;

  bufer=(short int*) z->buf;
  // vytvor strukturu
  out = mxCreateStructMatrix(1, 1, 22, data);
  //if (z->length==0) return(out);        // nejsou data
  if (bufer[0]!=64) return(out);          // jina nez datova zprava
  for (i=0;i<12;i++)                      // AI01-AI12
```

```
      {tmp = mxCreateDoubleScalar(bufer[i+1]); // vytvor pole a napln
       mxSetFieldByNumber(out,0,i,tmp);   // napln polozku strukturu polem
      };
  tmp = mxCreateDoubleScalar(bufer[14]); // vytvor pole
  mxSetFieldByNumber(out,0,12,tmp); // napln polozku strukturu polem
  if (bufer[15]&0x0010) tmp = mxCreateString("Remote"); // vytvor pole
  else                  tmp = mxCreateString("Local");
  mxSetFieldByNumber(out,0,13,tmp);       // napln polozku strukturu polem
  if (bufer[15]&0x0020) tmp = mxCreateString("Remote"); // vytvor pole
  else                  tmp = mxCreateString("Local");
  mxSetFieldByNumber(out,0,14,tmp);       // napln polozku strukturu polem
  if (bufer[15]&0x0040) tmp = mxCreateString("Remote"); // vytvor pole
  else                  tmp = mxCreateString("Local");
  mxSetFieldByNumber(out,0,15,tmp);       // napln polozku strukturu polem
  if (bufer[15]&0x0080) tmp = mxCreateString("Automat"); // vytvor pole
  else                  tmp = mxCreateString("Manual");
  mxSetFieldByNumber(out,0,16,tmp);       // napln polozku strukturu polem
  if (bufer[15]&0x0100) tmp = mxCreateString("Ano"); // vytvor pole
  else                  tmp = mxCreateString("Ne");
  mxSetFieldByNumber(out,0,17,tmp);       // napln polozku strukturu polem
  for (i=18;i<22;i++)                     // USe1,Use2,UCe,WTlak
      {tmp = mxCreateDoubleScalar(bufer[i-2]); // vytvor pole
       mxSetFieldByNumber(out,0,i,tmp);   // napln polozku strukturu polem
      };
  return(out);                      // end of Zprava
}
// vstupni bod mex-funkce ================================================
void mexFunction(int  nlhs,         // pocet par. na leve strane =
                 mxArray *plhs[],   // pole odkazu na par. leve strany
                 int  nrhs,         // pocet par. na prave strane =
           const mxArray *prhs[])   //pole odkazu na par. prave strany
{ char                  *funkce,*port;
  int                   n,i,typ,fce;
  double                x;
  unsigned long int     StavPort,Baud;
  unsigned short int    *ustav16;
  double                *udouble;         // ukazatel na pole double
  mes_UCTP              zprava;
  short int             bufer[32];        // bufer pro prijem i vysilani

  /* kontrola na pocet argumentu */
  if (nrhs < 1)   mexErrMsgTxt("MatHPS: minimalne 1 vstupni parametr\n");
  if (!(mxIsChar(prhs[0]))) mexErrMsgTxt("MatHPS: 1.parametr retezec\n");
  // vyjmuti pozadovane funkce (prvni parametr)
  funkce= mxArrayToString(prhs[0]);// ukazatel na ANSI ret. (dyn.alokovan)
  fce=*funkce;                      // prvni znak retezce
  mxFree(funkce);                   // uvolneni pameti pro retezec
   switch(fce)    // rozliseni pozadovane funkce podle prvniho pismene
  {case 'O':      // [flg,stav]=MatHPS('Open',port,rychlost) = inicializace
      // kontrola vstupnich parametru =MatHPS('Open',port,rychlost)
      if (nrhs != 3) mexErrMsgTxt("MatHPS.Open: 3 vstupni parametry\n");
      if (!(mxIsChar(prhs[1]))) mexErrMsgTxt("MatHPS.Open: 2.parametr
                                              retezec\n");
      // kontrola vystupnich parametru [flg,stav]=
      if (nlhs != 2) mexErrMsgTxt("MatHPS.Open: 2 vystupni parametry\n");
      // obsluha napojeni na vykonnou knihovnu dll_UCTP
      if (Init==0)                  // neinicializovano (==0)
         {mexLock();                // uzamkni MEX v pameti
          mexAtExit((void (*)(void))ExitFcn); // reg. fce vol. pri ukonceni
          x = mxGetScalar(prhs[2]); // data jako double (rychlost)
          Baud=x;                   // konverze do UINT32
          // ziskej nazev portu - ukazatel na retezec (dynamicky alokovan)
          port = mxArrayToString(prhs[1]);
```

```
        // kompletni inicializace komunikace (Dll_UCTP)
        StavPort=o.Init_UCTP(port,Baud); // StavPort=0 .. O.K.
        if (StavPort) StavPort=1; // pri chybe priznak 1
        Sleep(200);               // pockej vytvoreni threadu
        o.RTS_UCTP(1);            // nastav RTS (povol vysilani URJ)
        mxFree(port);             // uvolneni pameti pro retezec
        Init=1;
       }
     else {mexPrintf("MatHPS.Open: jiz inicializovano\n"); StavPort=2;};
     // vytvor vystupni parametr flg(1,1)
     plhs[0] = mxCreateNumericMatrix(1, 1, mxUINT16_CLASS, mxREAL);
     // ukazatel na data realne casti prveho vystupniho parametru
     ustav16=(WORD *) mxGetData(plhs[0]);     // ukazatel na pole UINT16
     ustav16[0]=StavPort;         // flg = hodnota priznaku
     // vytvor a napln druhy vystupni parametr stav(1,1(4))
     plhs[1]=Status();
break;                            // end of case 'O'
case 'C':      // [flg,stav]=MatHPS('Close') = ukonceni komunikace
   if (Init==0)                   // neinicializovano .. fatalni chyba
      mexErrMsgTxt("MatHPS.Close: neni Open");// vypis a ukonci MEX
   // kontrola vystupnich parametru [flg,stav]=
   if (nlhs != 2) mexErrMsgTxt("MatHPS.Close: 2 vystupni parametry\n");
   // vytvor vystupni parametr flg(1,1)
   plhs[0] = mxCreateNumericMatrix(1, 1, mxUINT16_CLASS, mxREAL);
   // ukazatel na data realne casti prveho vystupniho parametru
   ustav16=(WORD *) mxGetData(plhs[0]);     // ukazatel na pole UINT16
   ustav16[0]=0;                  // flg = O.K.
   // vytvor a napln druhy vystupni parametr stav(1,1(4))
   plhs[1]=Status();
   // ukonceni komunikace
   o.RTS_UCTP(0);                 // zrus RTS
   StavPort=o.Close_UCTP();
   // uvolneni pameti
   Init=0;                        // priznak neinicializovano
   mexUnlock();                   // odemkni MEX v pameti
break;                            // end of case 'C'
case 'S':      // [flg,stav]=MatHPS('Stav') = stavova slova komunikace
   if (Init==0)                   // neinicializovano .. fatalni chyba
      mexErrMsgTxt("MatHPS.Stav: neni Open");// vypis a ukonci MEX
   // kontrola vystupnich parametru [flg,stav]=
   if (nlhs != 2) mexErrMsgTxt("MatHPS.Stav: 2 vystupni parametry\n");
   // vytvor vystupni parametr flg(1,1)
   plhs[0] = mxCreateNumericMatrix(1, 1, mxUINT16_CLASS, mxREAL);
   // napln prvni vystupni parametr
   *(WORD *) mxGetData(plhs[0])=0;     // flg = O.K.
   plhs[1]=Status(); //vytvor a napln druhy vystup. par. stav(1,1(4))
   break;                                // end of case 'S'
case 'R':            // [flg,stav,mes]=MatHPS('Read') = precteni zpravy
   if (Init==0)                   // neinicializovano .. fatalni chyba
      mexErrMsgTxt("MatHPS.Read: neni Open");// vypis a ukonci MEX
   // kontrola vystupnich parametru [flg,stav,mes]=
   if (nlhs != 3) mexErrMsgTxt("MatHPS.Read: 3 vystupni parametry\n");
   zprava.buf=(char *) bufer;     // adresa bufru pro prijem
   zprava.length=sizeof(bufer);  // max. velikost prij. zpravy
   Baud=o.Read_UCTP(&zprava);     // vraci pocet prectenych byte
   // vytvor vystupni parametr flg(1,1)
   plhs[0] = mxCreateNumericMatrix(1, 1, mxUINT16_CLASS, mxREAL);
   // ukazatel na data realne casti prveho vystupniho parametru
   ustav16=(WORD *) mxGetData(plhs[0]);     // ukazatel na pole UINT16
   if (Baud==0) ustav16[0]=1;     // flg = chyba
   else         ustav16[0]=0;     // flg = O.K.
   plhs[1]=Status(); // vytvor a napln druhy vystup. par. stav(1,1(4))
   plhs[2]=Zprava(&zprava); // vytvor a napln treti vys.par.mes(1,1(22))
break;                            // end of case 'R'
```

```
   case 'W':            // [flg,stav]=MatHPS('Write',mes) = odeslani zpravy
      if (Init==0)                  // neinicializovano .. fatalni chyba
         mexErrMsgTxt("MatHPS.Write: neni Open\n");// vypis a ukonci MEX
      // kontrola vstupnich parametru =MatHPS('Write',mes)
      if (nrhs != 2) mexErrMsgTxt("MatHPS.Write: 2 vstupni parametry\n");
      // kontrola vystupnich parametru [flg,stav]=
      if (nlhs != 2) mexErrMsgTxt("MatHPS.Write: 2 vystupni parametry\n");
      udouble=mxGetPr(prhs[1]);     // ukazatel na pole double
      typ=udouble[0];               // typ zpravy
      if ((typ<1)||(typ>5)) mexErrMsgTxt("MatHPS.Write: Neznamy typ
                                          zpravy\n");
      bufer[0]=typ;                 // priprava vystupni zpravy
      if (typ==5) zprava.length=2;  // odblokovani
      else {if (typ==4)             // nastav 3 najednou
              {zprava.length=8; n=3;}
            else {zprava.length=4; n=1;}; // nastav pouze 1
            for (i=1;i<=n;i++)
              {bufer[i]=udouble[i];
               if (udouble[i]<0)    bufer[i]=0;
               if (udouble[i]>1023) bufer[i]=1023;
              };
           };
      zprava.buf=(char *) bufer;
      Baud=o.Write_UCTP(&zprava);   // vraci pocet zapsanych byte
      // vytvor vystupni parametr flg(1,1)
      plhs[0] = mxCreateNumericMatrix(1, 1, mxUINT16_CLASS, mxREAL);
      // ukazatel na data realne casti prveho vystupniho parametru
      ustav16=(WORD *) mxGetData(plhs[0]);      // ukazatel na pole UINT16
      if (Baud==0) ustav16[0]=1;    // flg = chyba
      else         ustav16[0]=0;    // flg = O.K.
      // vytvor a napln druhy vystupni parametr stav(1,1(4))
      plhs[1]=Status();
   break;                           // end of case 'W'
   default:
      mexPrintf("MatHPS: neznama funkce\n");
   break;
  };                                // end of switch
  return;                      // end of mexFunction
}
```