PŘÍMÉ POUŽITÍ SDÍLENÝCH KNIHOVEN
DIRECT USING OF SHARED LIBRARIES

František Dušek

KŘP FEI Univerzita Pardubice

Abstrakt

Článek se zabývá přímým použitím funkcí sdílených DLL knihoven z programového prostředí MATLAB/SIMULINK pod OS Windows. Volání funkcí z MATLABu je ukázáno na příkladě sdílené knihovny určené pro volání z jazyka „C“, která obsahuje funkce pro komunikaci s hardware fy National Instruments. Je uveden příklad S-funkce (Level-2 M-file S-function) realizující periodickou komunikaci s zařízením NI USB6009 s volitelnou periodou včetně synchronizace simulačního času s reálným časem (soft real-time) pomocí timeru MATLABu.

Kromě přístupu k funkcím knihovny z MATLABu je ukázán i způsob komunikace prostřednictvím ovladačů NIDAQmx. V příloze článku je uveden zdrojový text vytvořené S-funkce.

Abstract

This article is aimed to direct utilization of shared DLL libraries in MATLAB/SIMULINK environment under OS Windows. The libraries functions calling from MATLAB are shown on an example of the library which functions are designed to calling from “C” programming language and to communication with National Instruments hardware. The example of the S-function (Level-2 M-file S-function) that ensures periodic communication with NI USB6009 device is given. The period of communication is optional and the synchronization of simulation time with real time (soft real-time) is ensured by means of MATLAB timer

The library’s functions using from MATLAB environment and the communication via NIDAQmx drivers are shown, too. The source code of the S-function is introduced in an appendix of the article.

1 Úvod

Programové prostředí MATLAB/SIMULINK poskytuje, kromě základních služeb výpočetního a grafického jádra, množství dalších možností. Jde zejména o rozsáhlou podporu programování. Od verze 7.8 (R2009a) jde v podstatě o plnohodnotný objektově orientovaný jazyk. Možnost vytvářet složitější programy vyžaduje výměnu dat s okolním prostředím. Některé z možností byly v MATLABu k dispozici od začátku, jiné se postupně objevovaly v dalších verzích. Od počátku existuje možnost volat z prostředí MATLABu vlastní program (mex-file) napsaný v jazyce „C“ nebo FORTRAN. Tato nejuniverzálnější ale také nejpracnější možnost je poměrně podrobně popsána v dokumentaci [5]. Příklad použití této možnosti je např. v [1].

Uživatelsky nejsnazší cestou je používat hotové jednodušší či složitější funkce MATLABu pro výměnu dat. V současnosti MATLAB obsahuje standardně funkce pro práci se soubory (jak „low-level“ funkce ekvivalentní funkcím v jazyce „C“ tak i „high-level“ funkce pro čtení a zápis souborů v běžných formátech pro ukládání zvukových, obrazových i číselných dat), funkce či toolboxy pro komunikaci přes softwarová rozhranní (DDE, Aktive X, ODBC/JDBC – Databáze Tbx, OPC Tbx) a funkce či toolboxy pro komunikaci s hardwarovými prostředky (RS232/USB, zvuková karta, Data Acquisition Tbx, Image Acquisition Tbx, Instrument Control Tbx). Použití funkcí MATLABu pro práci se sériovou linkou je popsáno např. v [2].

Tento článek se zabývá jednou z méně známých možností výměny informací s okolím v prostředí MS Windows – přímým voláním funkcí sdílených (DLL Dynamic Linking Library) knihoven. Bezpro-střední motivací pro použití přímého volání funkcí sdílené knihovny byla situace, kdy autor chtěl vyu-žít Data Acquisition Toolbox (dále DAQ) pro komunikaci se zařízením NI

USB6009 v SIMULINKu. Verze DAQ, která byla součástí MATLABu R2007b, neobsahovala SIMULINKový blok umožňující použití nebufrovaných analogových výstupů (DAQ ve verzi MATLABu R2009a již tento blok obsahuje). Pod pojmem bufrovaný vstup/výstup se rozumí vlastnost ovladače přijmout požadavek na čtení/zápis určitého počtu vzorků (minimálně 2) s definovanou frekvencí, jejíž dodržení zajišťuje obvykle zařízení. Tento požadavek je pak samostatně jednorázově či opakovaně realizován nezávisle na uživatelském programu. Změřená/zadávaná data jsou přenášena samostatným požadavkem. Nebufrovaný vstup/výstup znamená jednorázovou realizaci požadavku čtení/zápisu včetně přenosu dat v okamžiku jeho přijmu. Jednoduché zařízení NI USB6009 nepodporuje bufrované analogové výstupy a tedy jeho výstupy nebylo možné pomocí bloků DAQ pro SIMULINK nastavovat. Nebufrované výstupy lze ale z MATLABu nastavovat standardní funkcí DAQ `putsample`. Bylo tedy nutné vytvořit vlastní S-funkci, která by využívala funkci `putsample`. A když už bylo nutné psát vlastní S-funkci, proč tedy používat univerzální funkce z DAQ, které stejně jen zprostředkovávají přístup k funkcím ovladače dodávaného se zařízením, a nepoužívat funkce příslušné funkce ovladače přímo? Výsledkem je jeden, na míru možnostem daného zařízení napsaný, blok SIMULINKu, který nepotřebuje DAQ, jednoduše se používá a méně zatěžuje systém.

Použití standardních DLL knihoven je ukázáno na příkladě volání funkcí z knihovny \windows\system32**nicaiu.dll**, která se instaluje jako součást univerzálních ovladačů NI-DAQmx zpřístupňujících hardware fy National Instruments. Uvedená knihovna obsahuje funkce pro komunikaci s hardware určené pro volání z jazyka „C“ [4].

Kromě základních informací jak pracovat s funkcemi MATLABu `loadlibrary`, `calllib` a `libfunction` je ukázáno jejich použití na příkladě komunikace s zařízením NI USB6009. Pro pochopení činnosti výsledné S-funkce je nejprve popsán způsob komunikace prostřednictvím funkcí ovladačů NIDAQmx se zařízením obsahujícím bufrované analogové vstupy, dvouhodnotové vstupy, nebufrované analogové výstupy a dvouhodnotové výstupy. Po stručném popisu struktury S-funkce je pozornost věnována také realizaci časové synchronizace a zajištění reentrantnosti S-funkce. V příloze je uveden zdrojový kód S-funkce (Level-2 M-file S-function), která zajišťuje periodické čtení/zápis všech vstupů/výstupů a synchronizaci výpočtu (simulační čas) s reálným časem (soft real-time) prostřednictvím časovače MATLABu.

2 Vlastnosti zařízení a princip komunikace

NI USB6009 fy National Instruments je jednoduché a levné zařízení (viz obr. 1) připojené k počítači přes univerzální rozhranní USB, ze kterého je také napájené. Dovoluje měřit až osm napěťových signálů, nastavovat dva napěťové signály, zjišťovat a nastavovat dvanáct dvou-úrovňových signálů a zjišťovat počet změn úrovní jednoho signálu. Stručný přehled možností je uveden v Tabulce 1. Další informace je možné získat např. v článku [3] nebo na adrese www.ni.com.

Obrázek 1 Zařízení NI USB USB6009

Se zařízením jsou dodávány ovladače NIDAQmx a další software, který je společný pro všechna zařízení NI. Z uživatelského hlediska je důležitý program Measurement & Automation (MAX), pomocí kterého lze ověřit funkčnost zařízení a zejména zjistit (a případně nastavit) jaké jméno je zařízení přiděleno (standardně Dev01).

Tabulka 1 ZÁKLADNÍ VLASTNOSTI NI USB6009

Typ	*Počet*	*Vlastnosti*	
AI	8 (s.e.)	14-bit, 48 kS/s impedance 144 kΩ	vstupní rozsah ± 10 V
	nebo 4 (diff.)		vstupní rozsah ± 20 V, ± 10 V, ± 5 V, ± 4 V, ± 2.5 V, ± 2 V, ± 1.25 V, ± 1.0 V
AO	2	12-bit, max. 150 Hz (software timed), 0-5 V, impedance 50 Ω, proud 5 mA	
DIO	8	(P0.0-7), CMOS, TTL, LVTTL, každý kanál samostatně nastavitelný	
DIO	4	(P1.0-3), CMOS, TTL, LVTTL, každý kanál samostatně nastavitelný	
counter	1	čítá náběžné/sestupné hrany, 32 bit, max 5 MHz	
napájení		USB (4.10 – 5.25 VDC) typicky 80 mA, max 500 mA	

Základní komunikaci se zařízením NI USB6009 bylo možné realizovat pomocí cca 13 funkcí. Prototypy funkcí jsou v souboru `..\NI-DAQ\DAQmx ANSI C Dev\include\`**`NIDAQmx.h`**. Tento hlavičkový soubor je sice určen pro standardní knihovnu `..\NI-DAQ\DAQmx ANSI C Dev\lib\NIDAQmx.lib`, ale je použitelný i pro sdílenou knihovnu **`nicaiu.dll`**.

Všechny funkce vracejí příznak provedení (celé číslo). Hodnota 0 znamená bezchybné provedení požadavku, nenulová hodnota signalizuje chybu při provedení požadavku s tím, že hodnota představuje kód chyby. Číselný kód lze převést na vysvětlující text pomocí funkce `DAQmxGetErrorString()`. Prvním vstupním parametrem všech funkcí je vždy ukazatel (handler, číslo typu uint32) na datovou strukturu úlohy (tasku).

Pokud chceme s nějakým zařízením komunikovat prostřednictvím univerzálních ovladačů NIDAQmx je nutné dodržet určitá pravidla. Prvním krokem je vytvoření úlohy (task) pro každou operaci, kterou chceme se zařízením provádět. V našem případě potřebujeme vytvořit 4 úlohy. Úlohu pro zjištění (čtení) naměřených hodnot napětí, úlohu pro nastavení (zápis) výstupních hodnot napětí, úlohu pro zjištění aktuálního stavu dvouhodnotových vstupů a úlohu pro nastavení zvolených úrovní dvouhodnotových výstupů. Všechny možnosti a funkce ovladačů NIDAQmx jsou uvedeny v on-line dokumentaci [4], která je součástí instalace. Pro vytvoření úlohy se použije funkce

`DAQmxCreateTask()`	vytvoření úlohy (task)

která naplní ukazatel adresou (či referencí) na vytvořenou datovou strukturu nové úlohy.

Druhým krokem je nastavení vlastností jednotlivých úloh tj. určení operace (typu požadavku) a napojení vybraných kanálů zvoleného fyzického zařízení na úlohu. V našem případě budou realizovány čtyři požadavky

`DAQmxCreateAIVoltageChan()`	požadavek měření napětí vůči společné zemi (ai0-ai7, ± 10 V)
`DAQmxCreateDIChan()`	požadavek čtení dvouhodnotových vstupů (P0.0-P0.7)
`DAQmxCreateAOVoltageChan()`	požadavek nastavení výstupního napětí (ao0-ao1, 0-5 V)
`DAQmxCreateDOChan()`	požadavek nastavení dvouhodnotových výstupů (P1.0-3)

Třetím krokem je nastavení počtu vzorků, frekvence vzorkování a dalších parametrů pro bufrované vstupy či výstupy. V našem případě je potřeba nastavit parametry bufrované operace pouze pro úlohu čtení měřeného napětí

`DAQmxCfgSampClkTiming()`	nastavení vlastností bufrované operace měření napětí

Čtvrtým krokem je spuštění vytvořených a nastavených úloh. Úlohy stačí spustit jednou (zjistit zda jsou splněny všechny podmínky pro spuštění). Každý požadavek čtení či zápisu v případě potřeby (data nejsou k dispozici) příslušnou úlohu spustí implicitně

`DAQmxStartTask()`	požadavek spuštění připravené úlohy

Nyní je možné opakovat funkce pro čtení a zápis, které jsou rozděleny podle typu požadavku

`DAQmxReadAnalogF64()`	požadavek vyčtení naměřených hodnot napětí
`DAQmxReadDigitalU8()`	požadavek vyčtení aktuálního stavu osmice DI
`DAQmxWriteAnalogF64()`	požadavek nastavení napětí na analogových výstupech
`DAQmxWriteDigitalU8()`	požadavek nastavení stavu na osmici DO

Po skončení práce je potřeba ukončit všechny spuštěné úlohy a pak je všechny zrušit (uvolnit přidělené prostředky pro použití dalšími úlohami)

`DAQmxStopTask()`	ukončení běžící úlohy
`DAQmxClearTask()`	zrušení ukončené úlohy

3 Použití funkcí sdílené knihovny z MATLABu

Chceme-li používat funkce sdílené (DLL) knihovny z MATLABu narazíme na tři principiální problémy. První problém je syntaxe volání funkcí tj. určení vstupních bodů (název funkce), počtů a typů parametrů jednotlivých funkcí. Druhý problémem jsou konverze standardních datových typů parametrů příslušné funkce na datové typy používané MATLABem. Třetím problémem jsou vstupní parametry typu odkaz (ukazatel) používanými ať už pro předání informace do nebo z funkce.

První problém je v MATLABu řešen tím, že při požadavku na načtení knihovny musí být k dispozici informace o prototypech funkcí v knihovně. Tato informace může být buď ve formě standardního hlavičkového souboru jazyka „C“ nebo rovnou ve formě speciální funkce (m-file), která po spuštění vytvoří strukturu s prototypy funkcí – interface do příslušné knihovny. Použijeme-li hlavičkový soubor, nejprve proběhne vytvoření dočasného souboru s funkcí, která se pak použije pro vytvoření struktury s prototypy potřebnými při vlastním volání funkcí. Proto je vhodné jednorázově tento soubor s potřebnou strukturu generující funkcí vytvořit a urychlit tak všechna následující zavádění knihovny do paměti. Vytvořený m-file je možné editovat a např. odstranit prototypy nepoužívaných funkcí a snížit tak paměťové nároky při používání sdílené knihovny (např. knihovna nidcaiu.dll obsahuje cca 2000 funkcí a v popisovaném případě je jich použito 15). M-file s názvem NIDAQprototypes.m se vytvoří na základě hlavičkového souboru NIDAQmx.h (v aktuálním adresáři) pro knihovnu **nicaiu.dll** (v systémovém adresáři ...\windows\system32) pomocí funkce MATLABu `loadlibrary` s parametrem „mfilename“

```
[notfound,warnings]=loadlibrary('nicaiu.dll','NIDAQmx.h',...
 'mfilename','NIDAQprototypes');
```

Načtení knihovny do paměti s využitím již vytvořeného souboru `NIDAQprototypes.m` (v aktuálním adresáři) včetně přidělení alias jména „NI“ se provede příkazem

```
loadlibrary('nicaiu.dll',@NIDAQprototypes,'alias','NI');
```

Po ukončení práce s knihovnou je potřeba oznámit, že ji již nepoužíváme, aby ji OS mohl případně (když ji nevyužívá jiná úloha) uvolnit z paměti. To se provede příkazem

```
unloadlibrary('NI');
```

Vlastní volání funkce ze zavedené knihovny se provede pomocí funkce `calllib` se syntaxí

```
[argOu,...]=calllib('nazev knihovny','nazev funkce',argIn,...);
```

Názvy funkcí zůstávají při volání z MATLABu stejné. Syntaxe funkcí (počet výstupních parametrů a datové typy) je změněna a souvisí s řešení druhého a třetího problému.

Druhý problém (konverze datových typů) řeší MATLAB (funkce `calllib`) automaticky pro základní datové typy včetně dvourozměrných polí (matice).

Třetí problém (odkazy v parametrech) řeší MATLAB tak, že vstupní parametry původní funkce volané odkazem jsou nahrazeny objektem typu `libpointer` a navíc jsou hodnoty kopírovány do dodatečných výstupních parametrů. To znamená, že syntaxe volání funkce v MATLABu se odlišuje od syntaxe původní knihovní funkce. Jak přesně syntaxe volání z MATLABu vypadá, lze zjistit pomocí funkce `libfunctions`. Dále uvedený příklad vygeneruje syntaxe volání (počet a datové typy parametrů) všech funkcí ze zavedené knihovny „NI“ do proměnné `proto` typu cell array

```
proto=libfunctions('NI','-full');
```

Dále je uveden příklad syntaxe původní funkce `DAQmxReadAnalogF64()` a syntaxe odpovídajícího volání funkce z MATLABu (odpovídající položka cell array `proto`)

```
int32 DAQmxReadAnalogF64(TaskHandle taskHandle, int32 numSampsPerChan,
float64 timeout, bool32 fillMode, float64 readArray[],
uInt32 arraySizeInSamps, int32 *sampsPerChanRead, bool32 *reserved);

{'[int32, doublePtr, int32Ptr, uint32Ptr] DAQmxReadAnalogF64(uint32,
int32,double, uint32, doublePtr, uint32, int32Ptr, uint32Ptr)';}
```

Konverzi datových typů i náhradu vstupních odkazů provádí MATLAB automaticky. Ke každému vstupnímu parametru typu odkaz se vytvoří nový výstupní parametr, do kterého se kopíruje hodnota určená vstupním odkazem nebo hodnota, kterou tam funkce měla zapsat. Na místě vstupního odkazu lze použít jak objekt `libpointer` (s informacemi o proměnné) tak i proměnnou či konkrétní hodnotu. Dokonce i v případě, vstupní parametr je ukazatel na hodnotu, do které se zapisuje, je možné použít všechny uvedené možnosti. Pozor na to, že z toho co uvedeme na místě vstupního odkazu zjišťuje MATLAB kromě hodnoty také datový typ a rozměry. To je důležité především v případě vstupního odkazu, podle kterého se má zapisovat. Proto je jistější, zejména v případě odkazů na složitější datové typy (např. matice), tyto odkazy nahrazovat explicitně pomocí `libpointer`.

Různé možnosti zápisu ukažme na příkladě pátého parametru `float64 readArray[]` výše uvedené funkce `DAQmxReadAnalogF64`. Jde o vstupní parametr ve významu ukazatele na pole hodnot typu double (= float64), kam má volaná funkce uložit naměřená data. Předpokládejme, že funkce vrací 8 hodnot typu double. Protože ve vstupních parametrech jsou tři odkazy, odpovídající funkce v MATLABu má tři výstupní parametry navíc oproti původní funkci. Ve volání v MATLABu je informace o výsledných datech uložena do druhého výstupního parametru (první odkaz `readArray[]` ve vstupních parametrech odpovídá prvnímu přidanému parametru `doublePtr` ve výstupních parametrech atd.). Konkrétní volání (s explicitním určením informací) je pak možné např.

```
dat=zeros(8,1);                         % informace o typu a rozměru
dIn=libpointer('doublePtr',dat);        % reference na (double) dat
[1,dat1,3,4]=calllib('NI','DAQmxReadAnalogF64',1,2,3,4,dIn,6,7,8);
dat2=get(dIn,'Value');                  % výsledky také přes dIn
```

Po provedení funkce jsou proměnné `dat1` i `dat2` shodné matice s naměřenými hodnotami. Počet hodnot a orientace (sloupec / řádek) jsou určené pomocnou proměnnou `dat`. Je potřeba si uvědomit rozdíl mezi ukazatelem v jazyce „C“ a referencí v objektu `libpointer`. Po provedení funkce se změní reference na proměnnou v objektu `dIn` (ačkoliv je to vstupní parametr funkce), ale nezmění se hodnota proměnné, na kterou původní reference ukazovala. Před provedením objekt `dIn` obsahoval referenci na matici `dat`, po provedení obsahuje referenci na matici výsledků tj. proměnná `dat` se nezměnila. Pokud by šlo o ukazatel jazyka „C“, pak by se proměnná `dat` naplnila novými hodnotami.

Jiné možné volání je bez použití objektu `libpointer`. V tomto příkladě je proměnná `dat` použita ve vstupních parametrech jako „vzor“ (datový typ a rozměry) a zároveň je identifikátor `dat` použit ve výstupních parametrech pro označení pole výsledků.

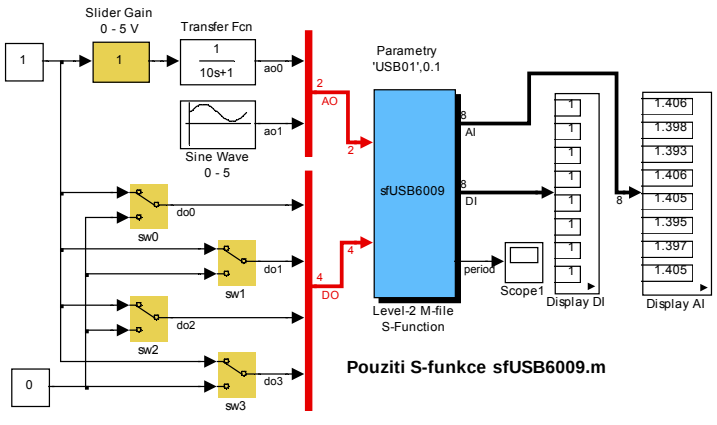
```
dat=zeros(8,1);                         % informace o typu a rozměru
[1,dat,3,4]=calllib('NI','DAQmxReadAnalogF64',1,2,3,4,dat,6,7,8);
```

Nebezpečná by mohla být situace, kdy funkce vrací rozsáhlý výsledek a my uvedeme rozměr výsledku menší. Pak může dojít k přepsání obsazené paměti. Omezení velikosti přepisované paměti je šestým parametrem (`uInt32 arraySizeInSamps`) použité funkce. Korektní řešení je tedy použít na místě šestého parametru v obou případech hodnotu `length(dat)`.

4 Vytvoření S-funkce pro komunikaci s NI USB6009

Aby bylo možné zařízení ze SIMULINKu jednoduše používat, byl navržen jeden diskrétní blok umožňující získání aktuálních měřených hodnot (napětí na 8 AI a úrovně na 8 DI) a současné nastavení výstupních hodnot (napětí na 2 AO a úrovně na 4 DO) na zařízení NI USB6009 identifikovaném názvem (první parametr bloku). Měření i nastavování výstupů probíhá v jednom volání bloku s volitelnou periodou (druhý parametr bloku) a simulační čas je synchronizován podle nastavené periody (v sekundách) s reálným časem.

Dva vstupy (AO a DO) a dva výstupy (AI, DI) bloku budou tvořeny složenými signály (1-D array, double). Hodnota výstupu (period) bude představovat skutečný čas v sekundách uplynulý od minulého volání bloku. Hodnoty signálů AO a AI budou odpovídat příslušnému napětí ve Voltech. Hodnoty signálů DO a DI budou nabývat úrovně 0 nebo 1 (čísla typu double). Tento blok bude realizován jednou S-funkcí (Level-2 M-file). Testovací SIMULINKový model, kde je vidět použití vytvořeného bloku, je na obr. 2.



Obrázek 2 Použití bloku v modelu SIMULINKu

4.1 Struktura S-funkce

Jako základ vytvářené S-funkce byl použit soubor **msfuntmpl.m** (Level-2 M-file Template), který byl dále modifikován. Z volaných metod byly využity pouze některé, tak aby byly splněny požadované funkce bloku při co nejjednodušší komunikaci se zařízením pomocí minima funkcí NIDAQmx. Blok je diskrétní s konstantní periodou zadanou jako parametr bloku.

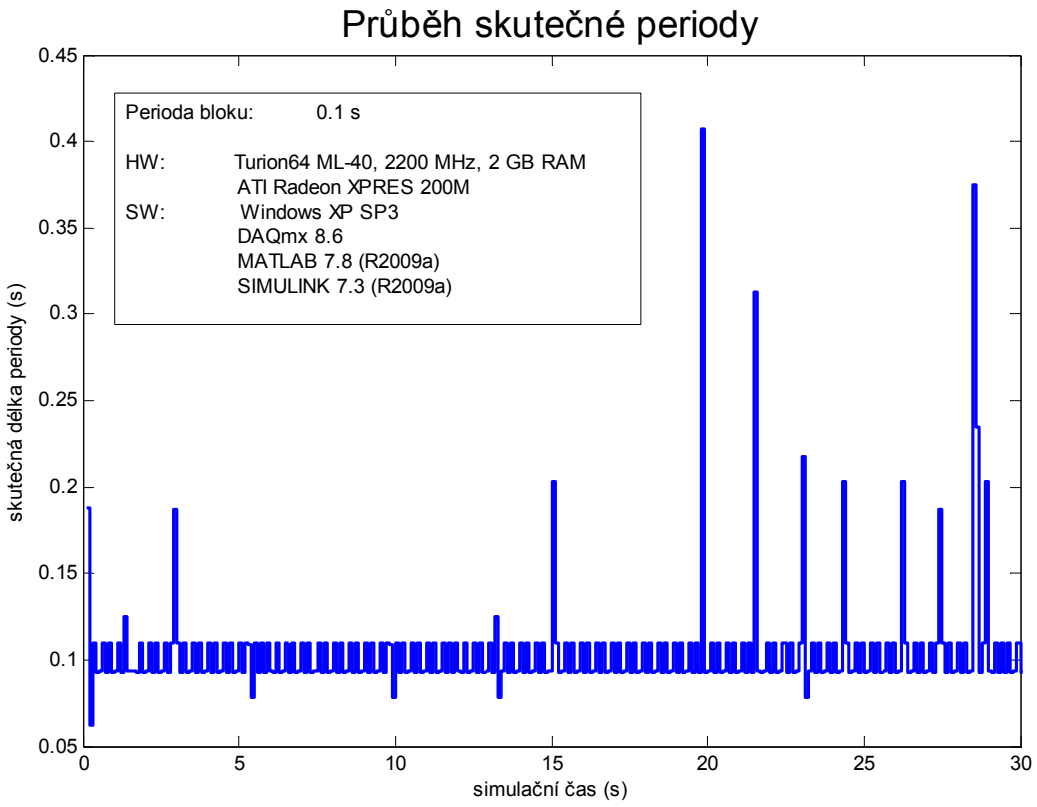
Určitou komplikací byla potřeba předávat mezi funkcemi (metodami) odkazy na společné objekty tj. handlery úloh a odkaz na objekt `Timer`. Jednoduché řešení by bylo využít možnosti deklarovat potřebné proměnné jako globální. Toto řešení by však nebylo reentrantní tj. nebylo by možné použít stejný blok současně pro obsluhu dalších zařízení. Pro korektní předávání handlerů úloh byl použit vektor diskrétního stavu bloku. Stavový vektor se vytváří samostatně pro každou instanci bloku čímž je reentrantnost zajištěna. Tento způsob byl možný díky tomu, že handler úlohy v NIDAQmx je datového typu uint32 a tento datový typ je, jako položka vektoru stavu, podporován. Toto jednoduché řešení nešlo použít pro objekt `Timer` (datový typ objekt nemůže být prvkem stavového vektoru). Pro zajištění toho, že každá instance bloku bude používat vlastní `Timer` bylo využito faktu, že `Timer` se vytváří v pracovní paměti MATLABu a v proměnné je pouze reference na tento objekt. To znamená, že po ukončení funkce, která o vytvoření objektu požádala, tento objekt existuje dále i když lokální proměnná s referencí na objekt zanikla. Referenci na objekt `Timer` lze pomocí funkce `timerfind` znovu obnovit. Reentrantnost je zajištěna tím, že při každém použití bloku se vytvoří jiná instance objektu `Timer` rozlišená jménem odpovídajícím jménu zařízení v parametrech bloku.

Pro základní orientaci v kompletním programu bloku uvedeném v Dodatku má sloužit Tabulka 2, ve které je uvedeno kdy a v jakém pořadí jsou jednotlivé metody (funkce) tvořící S-funkci volány.

4.2 Synchronizace na reálný čas (soft real-time)

Pokud v simulačním modelu používáme reálné zařízení je nutné synchronizovat čas simulace s reálným časem. Zde je použit jednoduchý způsob využívající objekt `Timer`, který je standardní součástí MATLABu. Objekt `Timer` umožňuje, krom jiného, zařadit periodické volání uživatelské funkce, jejíž provádění má přednost před prováděním hlavního programu. Tato funkce (nazvaná `TimerFcn`) přičte do uživatelsky přístupné proměnné objektu `.UserData` hodnotu skutečně uplynulého času od předchozího volání (proměnná objektu `.InstantPeriod`). Proměnná objektu `.UserData` je využita k synchronizaci simulačního času s reálným časem. Reálný čas je dán periodou volání funkce `TimerFcn`. Po vytvoření objektu `Timer` je proměnná `.UserData` vynulována a spuštěno periodické volání funkce `TimerFcn` s periodou zadanou v parametru bloku. V metodě `Outputs` S-funkce, která je opakovaně volaná v simulačním čase odpovídajícím násobku zadané periody diskrétního bloku, se v programové smyčce čeká na nenulovou hodnotu proměnné `.UserData`. Aby nebyly blokovány ostatní procesy na stejné či nižší prioritě, je do čekací smyčky zařazeno volání funkce MATLABu `pause` s parametrem 0.01 s. Po zjištění nenulové hodnoty `.UserData` je tato vynulována a simulační výpočet pokračuje. Teoreticky je tedy možné dodržet zadanou periodu s přesností ±0.01 s.

Správná synchronizace simulační a reálného času závisí na splnění dvou předpokladů. Prvním předpokladem je to, že během zadaného intervalu se stihnou provést potřebné simulační výpočty. Druhým předpokladem je, že je dodržena perioda časovače. První předpoklad souvisí se složitostí simulačního schématu a výkonem procesoru, druhý s celkovým zatížením počítače a zejména s operačním systémem. Dodržení zadaného časování lze sledovat na průběhu výstupu bloku period (skutečný čas od předchozího volání). Na obr. 3 je pro zajímavost zaznamenán průběh skutečné délky periody v průběhu 30 s simulačního času pro jednoduchý simulační model uvedený na obr.2. Je vidět občasné výrazné nedodržení požadované periody i na nepříliš zatíženém systému (cca 50% dle Správce úloh OS Windows). Požadavek periody komunikace 0.1 sec je pro zkušební systém na hranici použitelnosti.



Obrázek 3 Průběh skutečné periody volání bloku

5 Závěr

V článku je popsán způsob použití funkcí sdílených knihoven přímo z prostředí MATLAB. Jsou uvedeny základní informace nutné pro používání funkcí MATLABu `loadlibrary` a `calllib`. Jejich použití je ukázáno na příkladě používání knihovny **nicaiu.dll** zpřístupňující funkce ovladačů NIDAQmx fy National Instruments. Kromě problematiky používání funkcí sdílených knihoven je ukázána komunikace a ovládání zařízení NI USB6009. Dále je ukázáno vytvoření S-funkce (Level-2 M-file) zajišťující kromě komunikace s USB6009 pomocí volání funkcí sdílené knihovny také jednoduchou synchronizaci simulačního času s časem reálným (soft-real time). Pozornost je věnována také zajištění reentrantnosti S-funkce tj. řešení umožňujícímu použití stejného bloku pro současnou komunikaci s různými zařízeními s různou periodou. Součástí článku je kompletní komentovaný zdrojový kód vytvořené S-funkce.

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

Literatura

[1] **DUŠEK, F.** *Jedno řešení sériové komunikace.* In.: 16th Annual Conference Proceedings of Technical Computing Prague 2008, Kongresové centrum ČVUT, Praha, November 11, 2008, p. 26, ISBN 978-80-7080-692-0 (plný text 13 stran na doprovodném CD)

[2] **DUŠEK,F.; HONC,D.** *Využití sériové linky pod MATLABem verze 6.* In: 10. Konference MATLAB 2002, 11.10. 2002, Kongresové centrum ČVUT, Praha, s. 65-70, ISBN 80-7080-500-5

[3] **VLACH JAROSLAV** *Multifunkční karta a její aplikace.* Automatizace, 51(12), 2008, s.784-786, ISSN 0005-125X

[4] **NI-DAQmx C Reference Help** on-line dokumentace (též www.ni.com)

[5] **MATLAB** on-line dokumentace

Tabulka 2 POUŽITÉ METODY S-FUNKCE

Metoda	*Volána*	*Kdy*
	Činnost	
Setup	***jednou***	***první při vložení bloku,Update, při spuštění***
	určení počtu, rozměrů, datových typů parametrů, vstupů a výstupů určení typu bloku (diskrétní s konstantní periodou) registrace použitých metod	
CheckPrms	***jednou***	***druhá při vložení bloku, při spuštění***
	kontrola parametrů	
DoPostPropSetup	***jednou***	***třetí při spuštění***
	vytvoření vektoru diskrétních stavů bloku	
Start	***jednou***	***čtvrtá při spuštění***
	načtení knihovny vytvoření úloh pro čtení AI a DI, vytvoření úloh pro zápis AO a DO napojení na zařízení a fyzické kanály nastavení parametrů bufrovaného čtení AI spuštění úloh pro čtení (AI, DI) a zápis (AO, DO) vytvoření a příprava časovače	
InitializeConditions	***jednou***	***pátá po spuštění***
	první čtení AI s čekáním na dokončení a aktualizace výstupů bloku první čtení DI a aktualizace výstupů bloku první zápis AO a DO dle aktuálních vstupů start časovače	
Outputs	***opakovaně***	***šestá po spuštění***
	čekání na vypršení času časovače, zápis délky periody na výstup bloku přečtení vstupů (AI, DI) ze zařízení a aktualizace výstupů bloku zápis výstupů (AO, DO) na zařízení dle aktuálních vstupů bloku	
terminace	***jednou***	***sedmá před ukončením***
	zastavení časovače a zrušení časovače ukončení všech běžících úloh, zrušení všech úloh uvolnění knihovny	

doc. Ing. František Dušek, CSc.
KŘP FEI, Univerzita Pardubice
nám. Čs. legií 565
53210 Pardubice
frantisek.dusek@upce.cz

Dodatek – zdrojový kód S-funkce

soubor sfUSB6009.m

```
function sfUSB6009(block)
% S-funkce zpristupnujici HW NI USB6009 ze SIMULINKu
% vyzaduje instalovany software NIDAQmx (nicaiu.dll)
% + prototypy funkci (NIDAQprototypes.m)
% Frantisek Dusek 7.8.2009
% parametry: napr. 'USB01',0.1 =
%      = jmeno zarizeni (NIDAQmx), perioda vzorkovani (sec)
%

setup(block);

%% Function: setup ===========================================================
function setup(block)
% (1) volano pri vlozeni bloku do schematu a pri spusteni
% Register number of ports
  block.NumInputPorts  = 2;          % fixni pocet vstupu bloku AO + DO
  block.NumOutputPorts = 3;          % fixni pocet vystupu bloku AI+DI+per
% Override input port properties AO(ao0,ao1)
  block.InputPort(1).Complexity  ='Real'; block.InputPort(1).DataTypeId =0;
  block.InputPort(1).SamplingMode='Sample';block.InputPort(1).Dimensions=2;
  block.InputPort(1).DirectFeedthrough=false;
% Override input port properties DO(do0,do1,do2,do3)
  block.InputPort(2).Complexity  ='Real'; block.InputPort(2).DataTypeId= 0;
  block.InputPort(2).SamplingMode='Sample';block.InputPort(2).Dimensions=4;
  block.InputPort(2).DirectFeedthrough=false;
% Override output port properties AI(ai0,ai1,ai2,ai3,ai4,ai5,ai6,ai7)
  block.OutputPort(1).Complexity ='Real'; block.OutputPort(1).DataTypeId=0;
block.OutputPort(1).SamplingMode='Sample';block.OutputPort(1).Dimensions=8;
% Override output port properties DI(di0,di1,di2,di3,di4,di5,di6,di7)
  block.OutputPort(2).Complexity ='Real'; block.OutputPort(2).DataTypeId=0;
block.OutputPort(2).SamplingMode='Sample';block.OutputPort(2).Dimensions=8;
% Override output port properties err
  block.OutputPort(3).Complexity ='Real'; block.OutputPort(3).DataTypeId=0;
block.OutputPort(3).SamplingMode='Sample';block.OutputPort(3).Dimensions=1;
% Register parameters
  block.NumDialogPrms     = 2;
  block.DialogPrmsTunable = {'Nontunable','Nontunable'};
% Register sample times
  T = block.DialogPrm(2).Data;
  block.SampleTimes = [T 0]; % fixni vzorkovani T s
% Specify if Accelerator should use TLC or call back into M-file
  block.SetAccelRunOnTLC(false);
% Register methods called during update diagram/compilation
  block.RegBlockMethod('CheckParameters',     @CheckPrms);
%   Functionality: Setup work areas and state variables
  block.RegBlockMethod('PostPropagationSetup', @DoPostPropSetup);
% Register methods called at run-time
%   Functionality: Called in order to initialize state and work area values
  block.RegBlockMethod('InitializeConditions',@InitializeConditions);
%   Functionality: Called in order to initialize state and work area values
  block.RegBlockMethod('Start',               @Start);
%   Functionality: Called to generate block outputs in simulation step
  block.RegBlockMethod('Outputs',             @Outputs);
%   Functionality: Called at the end of simulation for cleanup
  block.RegBlockMethod('Terminate',           @Terminate);
return          % setup
```

```
% The local functions below
%% Function: CheckPrms ================================================
function CheckPrms(block)
% (2) volano po funkci setup (pri vlozeni bloku a spusteni)
% kontrola parametru, parametry jsou Read Only
  Name = block.DialogPrm(1).Data;           % nazev zarizeni dle NIDAQmx
  if ~ischar(Name), error('Nazev zarizeni napr. USB01'); end
  T    = block.DialogPrm(2).Data;           % perioda (sec)
  if T<0.1, error('Perioda musi byt vetsi nez 0.1 s'); end;
  block.SampleTimes = [T 0];                % fixni vzorkovani T s
return          % CheckPrms

%% Function: DoPostSetup ==============================================
function DoPostPropSetup(block)
% (3) 1x volano pri spusteni
% pro ulozeni tasku TaskAI, TaskAO, TaskDI, TaskDO
  Dev = block.DialogPrm(1).Data;            % nazev zarizeni dle NIDAQmx
  block.NumDworks                 = 1;
  block.Dwork(1).Name             = Dev;
  block.Dwork(1).Dimensions       = 4;      % 4 ulohy
  block.Dwork(1).DatatypeID       = 7;      % uint32 (handler)
  block.Dwork(1).Complexity       = 'Real';
  block.Dwork(1).UsedAsDiscState = true;
return          % PostPropagationSetup

%% Function: Start ====================================================
function Start(block)
% (4) 1x volano pri spusteni
% definice tasku TaskAI, TaskAO, TaskDI, TaskDO
%global TaskAI TaskAO TaskDI TaskDO tTimer
% definice pouzitych konstant pro vice funkcí
  DAQmx_Val_Volts       = int32(10348);     % jednotky Volty
  DAQmx_Val_Rising      = int32(10280);     % na nabeznou hranu signalu
  DAQmx_Val_ChanForAllLines = int32(1);     % jeden kanal pro cely port
% nacteni s kontrolou, zda je knihovna nactena
  if ~libisloaded('NI') % v aktualnim adresari musi byt NIDAQprototypes.m
      loadlibrary('nicaiu.dll',@NIDAQprototypes,'alias','NI');
  end
% ===CreateTask=======================================================
  TaskAI=libpointer('uint32Ptr',0);         % vytvor prazdny ukazatel
  TaskDI=libpointer('uint32Ptr',0);         % vytvor prazdny ukazatel
  TaskAO=libpointer('uint32Ptr',0);         % vytvor prazdny ukazatel
  TaskDO=libpointer('uint32Ptr',0);         % vytvor prazdny ukazatel
  Dev   =block.DialogPrm(1).Data;           % nazev zarizeni dle NIDAQmx
%int32 __CFUNC DAQmxCreateTask(const char taskName[],
%      TaskHandle *taskHandle);
%[int32, cstring, uint32Ptr] DAQmxCreateTask(cstring, uint32Ptr)
% vytvor task urceny pro cteni AI
  taskName=strcat(Dev,'-AI');               % vytvor unikatni jmeno tasku
  [err,taskName,hTaskAI]=calllib('NI','DAQmxCreateTask',taskName,TaskAI);
  if err~=0,VypisErr(err,'DAQmxCreateTask', taskName), end    % je-li chyba
  block.Dwork(1).Data(1)=hTaskAI;           % uloz handler
% vytvor task urceny pro cteni DI
  taskName=strcat(Dev,'-DI');               % vytvor unikatni jmeno tasku
  [err,taskName,hTaskDI]=calllib('NI','DAQmxCreateTask',taskName,TaskDI);
  if err~=0,VypisErr(err,'DAQmxCreateTask', taskName), end    % je-li chyba
  block.Dwork(1).Data(2)=hTaskDI;           % uloz handler
% vytvor task urceny pro zapis AO
  taskName=strcat(Dev,'-AO');               % vytvor unikatni jmeno tasku
  [err,taskName,hTaskAO]=calllib('NI','DAQmxCreateTask',taskName,TaskAO);
  if err~=0,VypisErr(err,'DAQmxCreateTask', taskName), end    % je-li chyba
  block.Dwork(1).Data(3)=hTaskAO;           % uloz handler
% vytvor task urceny pro zapis DO
```

```
  taskName=strcat(Dev,'-DO');                % vytvor unikatni jmeno tasku
  [err,taskName,hTaskDO]=calllib('NI','DAQmxCreateTask',taskName,TaskDO);
  if err~=0,VypisErr(err,'DAQmxCreateTask', taskName), end    % je-li chyba
  block.Dwork(1).Data(4)=hTaskDO;            % uloz handler
% ===CreateXXChan=====================================================
  DAQmx_Val_RSE = int32(10083);   % nonreferenced single-ended mode
% napoj se na analogove vstupy
%int32 __CFUNC DAQmxCreateAIVoltageChan(TaskHandle taskHandle,
%      const char physicalChannel[], const char nameToAssignToChannel[],
%      int32 terminalConfig, float64 minVal, float64 maxVal, int32 units,
%      const char customScaleName[]);
%[int32, cstring, cstring, cstring] DAQmxCreateAIVoltageChan(uint32,
%      cstring, cstring, int32, double, double, int32, cstring)
  [err,physicalChannel,chanName,customScaleName]=...
      calllib('NI','DAQmxCreateAIVoltageChan',hTaskAI,...
      strcat(Dev,'/ai0:7'),'y',DAQmx_Val_RSE,0,10,DAQmx_Val_Volts,'');
  if err~=0,VypisErr(err,'DAQmxCreateAIVoltageChan', chanName), end
% napoj se na digitalni vstupy
%int32 __CFUNC DAQmxCreateDIChan(TaskHandle taskHandle, const char lines[],
%      const char nameToAssignToLines[], int32 lineGrouping);
%[int32, cstring, cstring] DAQmxCreateDIChan(uint32, cstring, cstring,
%      int32)
  [err,lines,linName]=calllib('NI','DAQmxCreateDIChan',...
      hTaskDI,strcat(Dev,'/port0'),'di',DAQmx_Val_ChanForAllLines);
  if err~=0,VypisErr(err,'DAQmxCreateDIChan', linName), end   % je-li chyba
% napoj se na fyzické kanaly ao0-ao1 pro zapis napeti
%int32 __CFUNC DAQmxCreateAOVoltageChan(TaskHandle taskHandle,
%      const char physicalChannel[], const char nameToAssignToChannel[],
%      float64 minVal, float64 maxVal, int32 units,
%      const char customScaleName[]);
%[int32, cstring, cstring, cstring] DAQmxCreateAOVoltageChan(uint32,
%      cstring, cstring, double, double, int32, cstring)
  [err,physicalChannel,chanName,customScaleName]=...
      calllib('NI','DAQmxCreateAOVoltageChan',...
      hTaskAO,strcat(Dev,'/ao0:1'),'u',0,5,DAQmx_Val_Volts,'');
  if err~=0,VypisErr(err,'DAQmxCreateAOVoltageChan', chanName), end
% napoj se na digitalni vystupy
%int32 __CFUNC DAQmxCreateDOChan(TaskHandle taskHandle, const char lines[],
%      const char nameToAssignToLines[], int32 lineGrouping);
%[int32, cstring, cstring] DAQmxCreateDOChan(uint32, cstring, cstring,
%      int32)
  [err,lines,linName]=calllib('NI','DAQmxCreateDOChan',...
      hTaskDO,strcat(Dev,'/port1'),'do',DAQmx_Val_ChanForAllLines);
  if err~=0,VypisErr(err,'DAQmxCreateDOChan', linName), end   % je-li chyba
% ===CfgSampClkTiming=================================================
% nastav 1000 Hz a jeden vzorek na kanal pro cteni AI
%int32 __CFUNC DAQmxCfgSampClkTiming(TaskHandle taskHandle,
%      const char source[],float64 rate, int32 activeEdge,
%      int32 sampleMode, uInt64 sampsPerChan);
%[int32, cstring] DAQmxCfgSampClkTiming(uint32, cstring, double, int32,
%      int32, uint64)
  rate=double(1000);                         % frekvence 1000 Hz
  DAQmx_Val_FiniteSamps=int32(10178);        % konecny pocet vzorku
  sampsPerChanToAcquire = uint64(2);    % jeden vzorek na kanal jedno cteni
  [err,source]=calllib('NI','DAQmxCfgSampClkTiming',hTaskAI,...
      'OnBoardClock',rate,DAQmx_Val_Rising,DAQmx_Val_FiniteSamps,...
      sampsPerChanToAcquire);
  if err~=0,VypisErr(err,'DAQmxCfgSampClkTiming',strcat(Dev,'-AI')), end
% kontrola pripravy tasku (prechod do stavu COMMIT)
  DAQmx_Val_Task_Commit =3;
  err=calllib('NI','DAQmxTaskControl',hTaskAI,DAQmx_Val_Task_Commit);
  if err~=0,VypisErr(err,'DAQmxTaskControl',strcat(Dev,'-AI')), end
% ===DAQmxStartTask===================================================
%int32 __CFUNC DAQmxStartTask(TaskHandle taskHandle)
```

```
%int32 DAQmxStartTask(uint32)
  err=calllib('NI','DAQmxStartTask',hTaskDI);
  if err~=0,VypisErr(err,'DAQmxStartTask',strcat(Dev,'-DI')), end
  err=calllib('NI','DAQmxStartTask',hTaskAO);
  if err~=0,VypisErr(err,'DAQmxStartTask',strcat(Dev,'-AO')), end
  err=calllib('NI','DAQmxStartTask',hTaskDO);
  if err~=0,VypisErr(err,'DAQmxStartTask',strcat(Dev,'-DO')), end
  T    = block.SampleTimes(1);              % interval volani
  % vytvoreni casovace
  tTimer=timer('Period',T,'ExecutionMode','fixedRate');  % objekt Timer
  set(tTimer,'Name',Dev);                   % jednoznacne oznaceni casovace
  set(tTimer,'TimerFcn',@TimerFcn);         % funkce po spusteni casovace
  set(tTimer,'TasksToExecute',Inf);         % opakovani neomezene
  tTimer.UserData = 0;                      % Timer neprobehl
return              % Start(block)

%% Function: InitializeConditions ========================================
function InitializeConditions(block)
% (5) 1x volano pri spusteni (po Start)
  hTaskAI=block.Dwork(1).Data(1);   hTaskDI=block.Dwork(1).Data(2);
  hTaskAO=block.Dwork(1).Data(3);   hTaskDO=block.Dwork(1).Data(4);
% prvni mereni s cekanim
  dAI=ReadAI(hTaskAI);   nAI=block.OutputPort(1).Dimensions;
  for o=1:nAI, block.OutputPort(1).Data(o)= dAI(o); end
  dDI=ReadDI(hTaskDI);   nDI=block.OutputPort(2).Dimensions; mask=uint8(1);
  for o=1:nDI,
      if bitand(dDI,bitshift(mask,o-1)), block.OutputPort(2).Data(o) = 1;
      else                               block.OutputPort(2).Data(o) = 0;
      end
  end
% provedeni prvniho nastaveni
  AOd=[block.InputPort(1).Data(1),block.InputPort(1).Data(2)];
  DOd=[block.InputPort(2).Data(1),block.InputPort(2).Data(2),...
       block.InputPort(2).Data(3),block.InputPort(2).Data(4)];
  WriteAO(hTaskAO,AOd);                     % realizace vystupu AO
  WriteDO(hTaskDO,DOd);                     % realizace vystupu DO
% start casovace (Timer)
  Dev = block.DialogPrm(1).Data;            % nazev zarizeni dle NIDAQmx
  tTim= timerfind('Name',Dev);              % najdi Timer
  start(tTim);
  block.OutputPort(3).Data = 0;
return              % InitializeConditions(block)

%% Function: Outputs =====================================================
function Outputs(block)
% (6) opakovane volani pri behu (pred Update)
  Dev = block.DialogPrm(1).Data;            % nazev zarizeni dle NIDAQmx
  tTim = timerfind('Name',Dev);             % najdi Timer a napln odkaz
  hTaskAI=block.Dwork(1).Data(1);   hTaskDI=block.Dwork(1).Data(2);
  hTaskAO=block.Dwork(1).Data(3);   hTaskDO=block.Dwork(1).Data(4);
% mereni spusteno, prvni start(tTimer) je ve fci InitializeConditions
  AOd=[block.InputPort(1).Data(1),block.InputPort(1).Data(2)];
  DOd=[block.InputPort(2).Data(1),block.InputPort(2).Data(2),...
       block.InputPort(2).Data(3),block.InputPort(2).Data(4)];
% !!!! zde SIMULINK ceka !!!!
  while tTim.UserData==0,
      pause(0.01);                          % cekej na UserData<>0
  end
  block.OutputPort(3).Data=tTim.UserData;   % doba od posl. volani
  tTim.UserData=0;
% provedeni nastaveni
  WriteAO(hTaskAO,AOd);                     % realizace vystupu AO
  WriteDO(hTaskDO,DOd);                     % realizace vystupu DO
```

```
% mereni s cekanim
  dAI=ReadAI(hTaskAI);   nAI=block.OutputPort(1).Dimensions;
  for o=1:nAI, block.OutputPort(1).Data(o)= dAI(o); end
  dDI=ReadDI(hTaskDI);   nDI=block.OutputPort(2).Dimensions; mask=uint8(1);
  for o=1:nDI,
      if bitand(dDI,bitshift(mask,o-1)), block.OutputPort(2).Data(o) = 1;
      else                               block.OutputPort(2).Data(o) = 0;
      end
  end
return              % Outputs(block)

%% Function: Terminate ===================================================
function Terminate(block)
% (7) volano pri ukonceni (po Update)
  Dev = block.DialogPrm(1).Data;            % nazev zarizeni dle NIDAQmx
  tTim= timerfind('Name',Dev);
  hAI=block.Dwork(1).Data(1);   hDI=block.Dwork(1).Data(2);
  hAO=block.Dwork(1).Data(3);   hDO=block.Dwork(1).Data(4);
  stop(tTim);                               % odstav Timer a mereni
  delete(tTim);                             % odstran objekt
%int32 __CFUNC DAQmxStopTask(TaskHandle taskHandle)
%int32 DAQmxStopTask(uint32)
%int32 __CFUNC DAQmxClearTask(TaskHandle taskHandle)
%int32 DAQmxClearTask(uint32)
  e=calllib('NI','DAQmxStopTask',hAI);e=calllib('NI','DAQmxClearTask',hAI);
  e=calllib('NI','DAQmxStopTask',hAO);e=calllib('NI','DAQmxClearTask',hAO);
  e=calllib('NI','DAQmxStopTask',hDI);e=calllib('NI','DAQmxClearTask',hDI);
  e=calllib('NI','DAQmxStopTask',hDO);e=calllib('NI','DAQmxClearTask',hDO);
return              % Terminate(block)

%% ======= blok uzivatelskych funkci ====================================
function VypisErr(err,fcnName, taskName)
% vypis identifikace funkce a textu chyby
%int32 __CFUNC DAQmxGetErrorString(int32 errorCode, char errorString[],
%      uInt32 bufferSize);
%[int32, cstring] DAQmxGetErrorString(int32, cstring, uint32)
  Msg=char(zeros(1,512)+32);                % 512 mezer
  [err,Msg]=calllib('NI','DAQmxGetErrorString',err,Msg,512);
  disp(strcat(fcnName,' (',taskName,') ',Msg))
return              % VypisErr(fcnName, taskName)

function TimerFcn(obj,event)                % pro Timer
  obj.UserData = obj.UserData+obj.InstantPeriod; % pricti skutec.delky per.
return              % TimerFcn(obj,event)

function Y=ReadAI(hTaskAI)
% USB NI6009 - cteni 8xAI pomoci existujiciho tasku
  DAQmx_Val_GroupByChannel=uint32(1);       % neprokládaně
  PtrRes=libpointer('uint32Ptr');           % vytvor ukazatel null
  d16   =zeros(16,1);
  PtrBuf=libpointer('doublePtr',d16);       % vytvor ukazatel na bufer
%int32 __CFUNC DAQmxReadAnalogF64(TaskHandle taskHandle,
%      int32 numSampsPerChan, float64 timeout, bool32 fillMode,
%      float64 readArray[], uInt32 arraySizeInSamps,
%      int32 *sampsPerChanRead, bool32 *reserved);
% [int32, doublePtr, int32Ptr, uint32Ptr] DAQmxReadAnalogF64(uint32,
%      int32, double, uint32, doublePtr, uint32, int32Ptr, uint32Ptr)'
  [err,d16,Poc,d]=calllib('NI','DAQmxReadAnalogF64',...
      hTaskAI,2,0.2,DAQmx_Val_GroupByChannel,PtrBuf,length(d16),0,PtrRes);
  if err~=0,VypisErr(err,'DAQmxReadAnalogF64', 'AI'), end     % je-li chyba
  Y=(d16(1:8)+d16(9:16))/2;
return              % ReadAI(TaskAI)
```

```
function b=ReadDI(hTaskDI)
% USB NI6009 - cteni 8xDI pomoci existujiciho tasku
  PtrRes=libpointer('uint32Ptr');           % vytvor ukazatel null
  u8=uint8(0);                              % bufer pro vysledek
  PtrBuf=libpointer('uint8Ptr',u8);         % vytvor ukazatel na bufer
%int32 __CFUNC DAQmxReadDigitalU8(TaskHandle taskHandle,
%      int32 numSampsPerChan, float64 timeout, bool32 fillMode,
%      uInt8 readArray[], uInt32 arraySizeInSamps, int32 *sampsPerChanRead,
%      bool32 *reserved);
%[int32, uint8Ptr, int32Ptr, uint32Ptr] DAQmxReadDigitalU8(uint32, int32,
%      double, uint32, uint8Ptr, uint32, int32Ptr, uint32Ptr)'
  [err,u8,Poc,d]=calllib('NI','DAQmxReadDigitalU8',...
      hTaskDI,1,0.2,0,PtrBuf,1,0,PtrRes);
  if err~=0,VypisErr(err,'DAQmxReadDigitalU8', 'DI'), end     % je-li chyba
  b=u8;
return              % ReadDI(TaskDI)

function WriteAO(hTaskAO,data)
% USB NI6009 - zápis 2xAO pomoci existujiciho tasku
  AO=[data(1),data(2)];
  if (data(1)<0), AO(1)=0; end;    if (data(1)>5), AO(1)=5; end
  if (data(2)<0), AO(2)=0; end;    if (data(2)>5), AO(2)=5; end
  PtrRes=libpointer('uint32Ptr');       % vytvor ukazatel null na reserved
  PtrBuf=libpointer('doublePtr',AO);        % vytvor ukazatel na bufer
%int32 __CFUNC DAQmxWriteAnalogF64(TaskHandle taskHandle,
%      int32 numSampsPerChan, bool32 autoStart, float64 timeout,
%      bool32 dataLayout, const float64 writeArray[],
%      int32 *sampsPerChanWritten, bool32 *reserved);
%[int32, doublePtr, int32Ptr, uint32Ptr] DAQmxWriteAnalogF64(uint32, int32,
%      uint32, double, uint32, doublePtr, int32Ptr, uint32Ptr)
  [err,d2,Poc,d]=calllib('NI','DAQmxWriteAnalogF64',...
      hTaskAO,1,0,0.2,0,PtrBuf,0,PtrRes);
  if err~=0,VypisErr(err,'DAQmxWriteAnalogF64', 'AO'), end    % je-li chyba
return               % WriteAO(TaskAO,data)

function WriteDO(hTaskDO,data)
% USB NI6009 - zápis 4xDO pomoci existujiciho tasku
  d=0;
  if (data(1)~=0), d=d+1; end;              % bit c.0
  if (data(2)~=0), d=d+2; end;              % bit c.1
  if (data(3)~=0), d=d+4; end;              % bit c.2
  if (data(4)~=0), d=d+8; end;              % bit c.3
  PtrRes=libpointer('uint32Ptr');       % vytvor ukazatel null na reserved
  ub=uint8(d);                              % preved na byte
  PtrBuf=libpointer('uint8Ptr',ub);         % vytvor ukazatel na bufer
%int32 __CFUNC DAQmxWriteDigitalU8(TaskHandle taskHandle,
%      int32 numSampsPerChan, bool32 autoStart, float64 timeout,
%      bool32 dataLayout, const uInt8 writeArray[],
%      int32 *sampsPerChanWritten, bool32 *reserved);
%[int32, uint8Ptr, int32Ptr, uint32Ptr] DAQmxWriteDigitalU8(uint32, int32,
%      uint32, double, uint32, uint8Ptr, int32Ptr, uint32Ptr)
  [err,d2,Poc,d]=calllib('NI','DAQmxWriteDigitalU8',hTaskDO,...
      1,1,0.2,0,PtrBuf,0,PtrRes);
  if err~=0,VypisErr(err,'DAQmxWriteDigitalU8','DO'), end     % je-li chyba
return               % WriteDO(TaskDO,data)
%% ====== konec bloku uzivatelskych funkci ================================
```