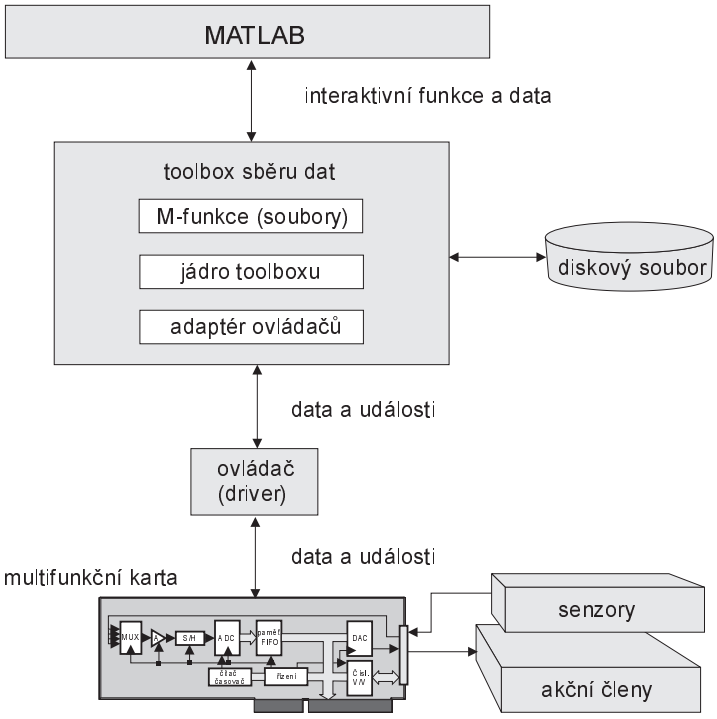
4. Sběr dat pomocí systému MATLAB

Matlab je vysoce výkonný jazyk pro technické výpočty. Zahrnuje výpočty, grafické zobrazení a programování ve srozumitelném prostředí, v němž jsou problémy a jejich řešení vyjádřeny běžným matematickým zápisem. Rychlý přehled Matlabu získáme příkazem

helpdesk

Systém obsahuje vlastní interpret jazyka Matlab, ve kterém lze připravit jak dávkové soubory, tak i definovat nové funkce. Tyto funkce mohou být interpretovány buď přímo z textové podoby souborů nazývaných **M-file** nebo z předzpracované podoby **P-file**. Systém dále umožňuje přidávat moduly (soubory **MEX-file**) zkompilované do strojového kódu procesoru. Jazykem zdrojových souborů může být jazyk C, C++, Fortran nebo po použití MCC (Matlab to C compiler) i funkce uložená v M-file.

Důležitou částí instalace Matlabu jsou knihovny funkcí (ve skutečnosti adresáře s M a MEX soubory), nazývané **toolboxy**. Obsahují vždy uceleným způsobem zpracovaný některý obor numerické matematiky včetně dokumentace a příkladů použití. Matlab je vybaven grafickým intuitivním prostředím **Simulink**, které umožňuje schématicky znázornit téměř libovolný dynamický systém a provést simulaci jeho chování. Matlab se Simulinkem jsou nejen prostředky řešení teoretických problémů, ale umožňují i spojení s reálným světem pomocí knihoven funkcí nazvaných **Real Time Toolbox** [25] a **Data Acquisition Toolbox** [26]. Tyto programové prostředky spolu se zásuvnými kartami pro sběr dat, případně běžnými zvukovými kartami, jsou silným prostředkem využitelným v měřicí a řídící technice. V této kapitole se seznámíme se základními vlastnostmi těchto programových prostředků.



Obrázek 4.1: Struktura systému umožňujícího práci s reálnými daty v systému MATLAB

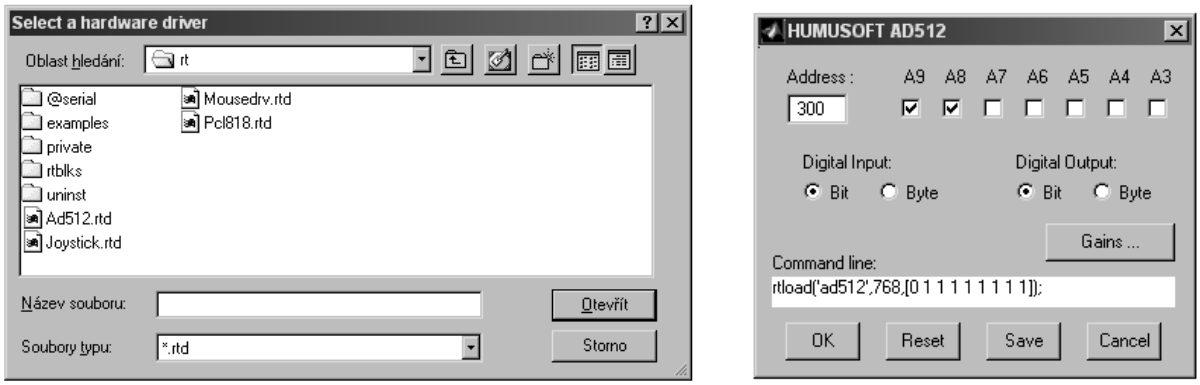
Celý systém, jehož využívá Matlab pro sběr dat lze znázornit blokovým schématem

podle obr. 4.1. Vlastní toolbox, který zajišťuje komunikaci s reálným světem sestává ze tří částí vyznačených v obrázku.

Vlastní jádro toolboxu plní své úkoly **asynchronně** s činností Matlabu, takže Matlab lze současně využít i pro řešení dalších problémů.

4.1 Real Time Toolbox

Real Time Toolbox (dále RTT) umožňuje spojit Matlab a Simulink s reálným světem. K tomu lze využít jak běžné programování pomocí instrukcí (Matlab) tak i grafické modelování systémů (Simulink).



Obrázek 4.2: a) GUI pro výběr ovládače karty b) Nastavení parametrů ovládače karty

4.1.1 Základní příkazy

Pro spojení s okolím je nutno vybavit počítač kartou pro sběr dat (viz kap. 3) a zavést do systému její ovládač (driver), pokud již tam není (seznam karet, které podporuje RTT najdeme na internetové stránce **www.humusoft.cz**). Potom RTT aktivuje kartu pomocí příkazu

rtload

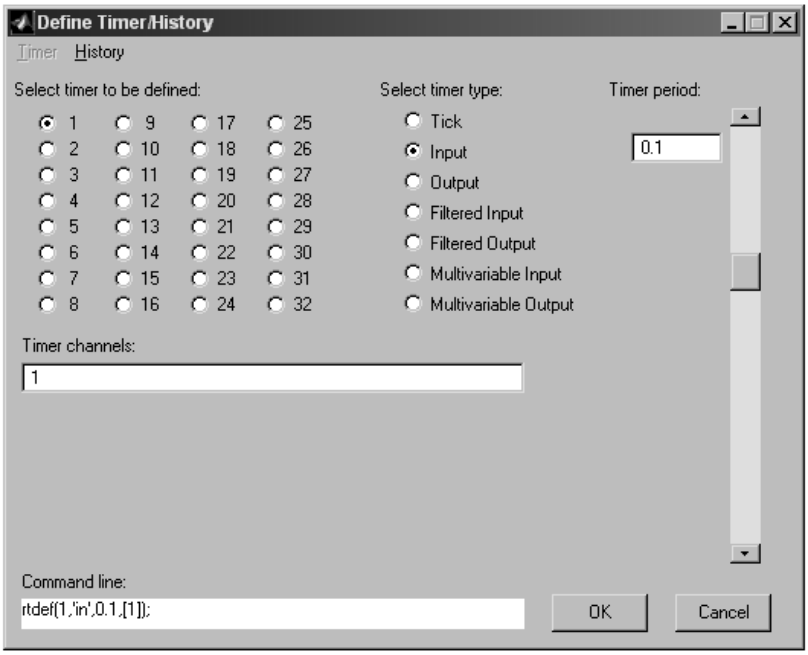
zapsaného do příkazového okna Matlabu. Objeví se grafické uživatelské rozhraní (graphical user interface - GUI) podle obr. 4.2a. Dvojím kliknutím na zvolený ovládač otevřeme další okno, např. podle obr. 4.2b. Toto okno platí pro kartu Humusoft AD512. GUI nám umožní jednoduše zvolit takové vlastnosti karty, které potřebujeme. V řádku Commmand line je vidět zapsaná instrukce. Její složitost je snadno překonána pomocí GUI. Zapíše se adresa, která je na kartě nastavena přepínači, tlačítkem GAINS se zvolí rozsah analogového vstupu a formát číslicových vstupů/výstupů (bit nebo byte). Současně může být aktivní i několik karet, musí však mít nastavené různé adresy. Chceme-li aktivní karty deaktivovat zvolíme příkaz

rtunload

Činnosti, které řídí RTT jsou **synchronní** a proto je nutno definovat časovače (timers) použitím příkazu

rtdef

Objeví se okno uvedené na obr. 4.3. V tomto okně musíme provést tato nastavení:
1) Je zde 32 políček, z nichž každému může být přiřazen jeden časovač. Odpovídající číslo charakterizuje v systému daný časovač a současně určuje jeho prioritu.
2) Ke každému časovači přiřadit typ procesu (Select timer type).



Obrázek 4.3: Definování časovačů

3) Zvolenému časovači přiřadit kanály použité multifunkční karty. Např. kanál 1 odpovídá prvnímu analogovému vstupu karty obyčejně označenému AD0. Vstupní a výstupní kanály jsou číslovány nezávisle na sobě.
4) Zvolíme periodu časovače (Timer period).
Volbu zkontrolujeme příkazem

rtwho

který vyvolá výpis tohoto typu:

```
» rtwho

Real Time Toolbox version 3.00                    (C) HUMUSOFT 1992-1999
Matlab performance = 100.0%
Kernel timeslice period = 1 ms

TIMERS:  Number      Type     Period   Running

           1          In       0.1      No
           2          In       0.1      No

DRIVERS:                            Name    Address   Inputs  Outputs

                          HUMUSOFT AD512      0x360     16      10
```

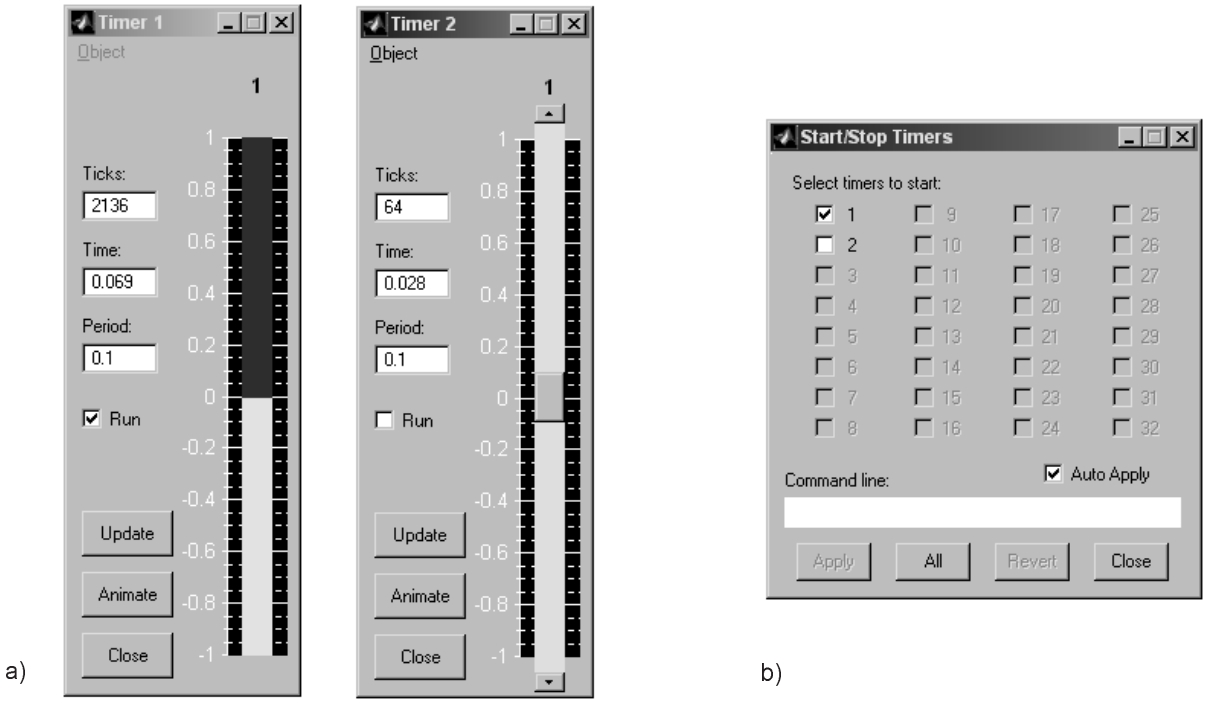
Chceme-li zvolené časovače odstranit, zapíšeme

rtclear

Pro indikaci čtených nebo zapisovaných dat otevřeme příslušná okna (obr. 4.4a) pomocí příkazů

rtrd rtwr

Příkaz bez argumentu otevře vždy první kanál. Pro otevření dalších kanálů zapíšeme např. rtrd(2) a pod. Volba Run spouští a zastavuje časovač. Okno začne být aktivní po stisknutí tlačítka Animate. Je nutno si zapamatovat, že nastartování časovače a oživení okna jsou dvě zcela nezávislé činnosti. V režimu Animate nelze zadávat instrukce do příkazového okna Matlabu. Proto je zde ještě tlačítko Freeze/Update, které nám to umožní za cenu dočasného potlačení zobrazovaných dat.



Obrázek 4.4: a) Vstupní a výstupní časovač b) Start a stop časovačů

Chceme-li spustit několik časovačů současně tak, aby pracovaly synchronně použijeme příkaz

rtstart

a pro jejich současné zastavení příkaz

rtstop

Tyto příkazy otevřou okno (obr. 4.4b), v němž zatrhneme čísla vybraných časovačů. Pokud je zaškrtnuta volba Auto Apply, pak po zaškrtnutí čísla časovače dojde ihned k jeho spuštění (příp. zastavení). Chceme-li jich ovládat více najednou, zrušíme volbu Auto Apply a použijeme tlačítko Apply. Tlačítko Revert umožňuje kontrolu aktuálního stavu časovačů. Pokud chceme selektivně ovládat časovače pouze příkazem (bez otevření GUI) použijeme např. příkaz rtstart[1 2], který spustí časovače 1 a 2.

Příkaz určený do programu nemá běžně otvírat GUI, proto musí být zapsán ve tvaru, který se v každém GUI objevuje v řádku Command line. Tvorbu a ukládání příkazů do souboru zjednodušuje GUI vyvolané příkazem

rtscript

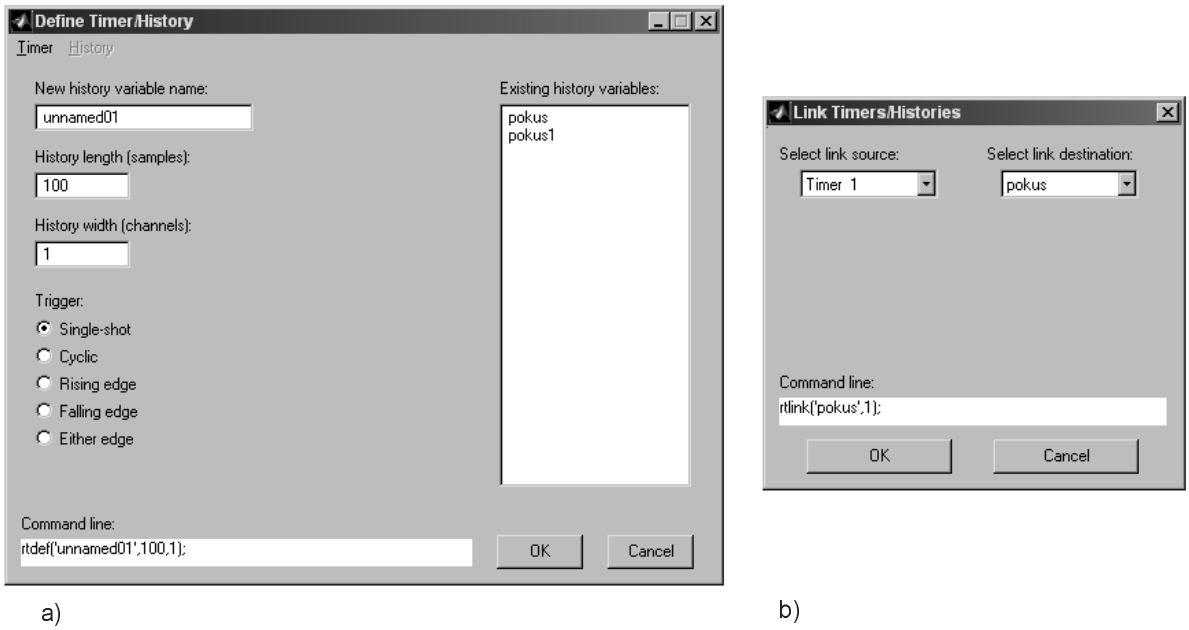
Následující program změří odezvu na skokovou vstupní změnu zařízení připojeného mezi vstupní kanál č.1 a výstupní kanál č.1:

```
Ts=0.1;              % perioda vzorkovani
N=400;               % pocet vzorku
A=1;                 % amplituda skoku
Ns=100;              % zacatek skoku
y=zeros(N,1);        % priprava vystupniho vektoru
rtload('AD512');     % aktivace karty
rtdef(1,'out',Ts,1);% definuj vystupni casovac
```

```
rtdef(2,'in',Ts,1); % definuj vstupni casovac
rtwr(1,'y',0);      % zapis pocatecni hodnotu
rtstart all;        % start casovacu

for i=1:N;
  while(rtrd(2,'t'))<i;end;% cekej na dalsi vzorek
  y(i)=rtrd(2,'y'); % cti vstupni hodnotu
  if i==Ns;
     rtwr(1,'y',A); % proved skok
  end;
end;

plot(y);            % zobraz
rtclear;            % zrus casovace
rtunload;           % deaktivuj kartu
```



Obrázek 4.5: a) Určení parametrů záznamu b) Propojení časovače a záznamu

4.1.2 Rychlé zpracování dat

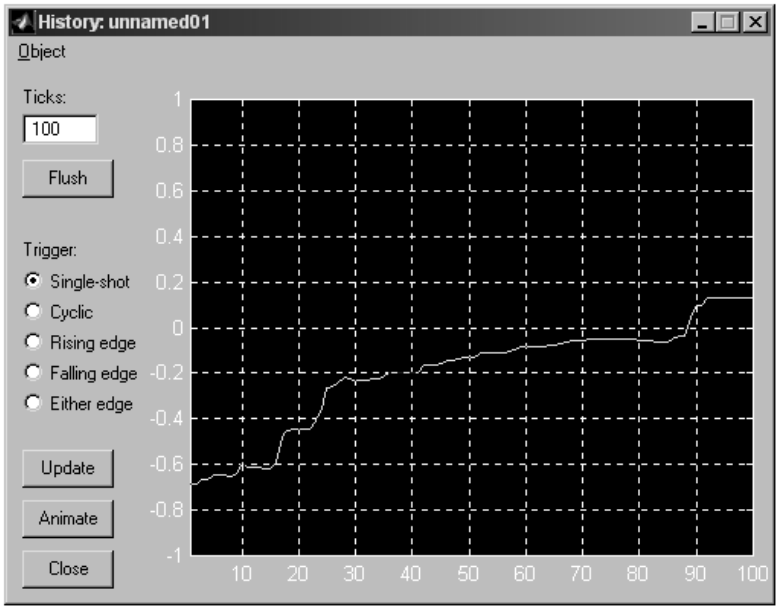
Provádění napsaného programu může někdy zpomalovat rychlost zpracování dat. Potom je rychlost vzorkování určena rychlostí programu a nikoliv časovačem. Proto je RTT vybaven instrukcemi umožňujícími rychlou činnost.

Pomocí instrukce rtdef můžeme kromě časovače též definovat paměť pro záznam dat. Volíme-li v okně podle obr. 4.3 položku History, otevře se okno podle obr. 4.5a. V okně zvolíme délku záznamu (History length), počet kanálů (History width), způsob spouštění záznamu: Single shot znamená jednorázový zápis nebo čtení dat (viz dále příkaz rtlink), Cyclic znamená opakovaný zápis nebo čtení dat, Rising edge znamená start na vzestupnou hranu vstupního signálu, Falling edge na sestupnou a Either edge na vzestupnou i na sestupnou hranu. V řádku Command line vidíme zápis instrukce použitelný do programu.

Aby data měla význam je nutno spojit paměť s časovačem. To provedeme příkazem

rtlink

který otevře okno uvedené na obr. 4.5b. Zde jsou dvě okénka pro volbu zdroje a cíle. Např. paměť může být zdrojem pro výstupní časovač a cílem pro vstupní časovač. V případě, že paměť určíme jako cíl pro výstupní časovač, pak vysílaná data jsou současně zapisována do paměti. V případě, že určíme paměť jako zdroj vstupnímu časovači, bude systém číst data z paměti a nikoliv z karty. V řádku Command line vidíme zápis instrukce použitelný do programu.



Obrázek 4.6: Záznam vstupních dat

Záznam dat lze využít též pomocí instrukcí rtrd a rtwr. Tyto instrukce vyvolají okno podle obr. 4.3. Pokud v tomto okně zvolíme Object, otevře se okno zobrazené na obr. 4.6. Aternativně lze použít instrukci

rtrd('name')

kde name je název uložených dat definovaný v okně podle obr. 4.5a. V okně podle obr. 4.6 se zobrazí data snímaná z multifunkční karty. Tlačítka Update a Animate mají stejnou funkci jako v obr. 4.4a. Tlačítko Flush vynuluje čítač vzorků, podobně jako instrukce

rtflush('name')

Rychlé čtení a zobrazení dat dosáhneme instrukcí

rtscan(ch,len,per)

která z kanálu ch přečte len vzorků s periodou per. Instrukci rtscan lze použít pouze pro kartu s řadičem DMA (např. karta PCL 818). Chceme-li vstupní data uložit jako proměnnou y zapíšeme instrukci např. y=rtscan(1,100,1e-4). Ta otevře okno na obr. 4.7. Je to rovněž nejsnazší cesta jak zobrazit vstupní signál.

Potřebujeme-li generovat signál použijeme instrukci

rtwave(ch, data, per)

která z paměti označené data vyšle uložené vzorky na kanál ch s periodou per.

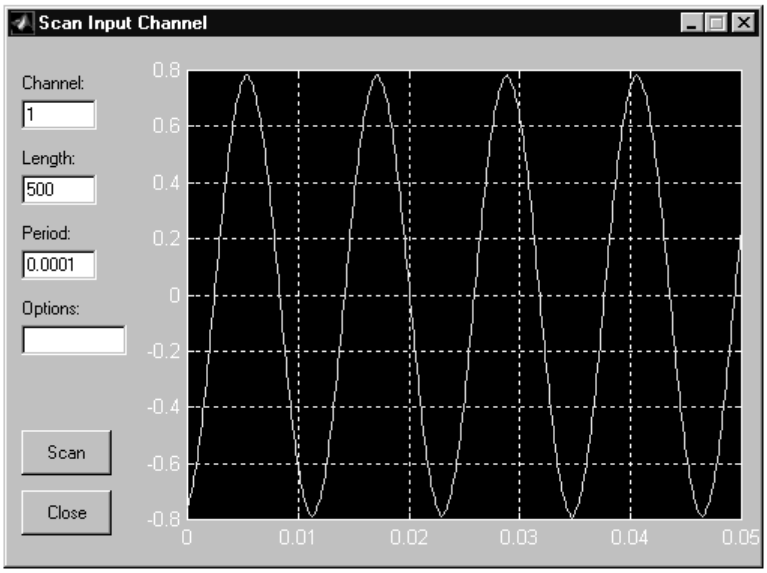
Okamžité čtení dat zajistí instrukce

rtin(ch)

Instrukce

rtout(ch, val)

pošle ihned hodnotu val na kanál ch.



Obrázek 4.7: Záznam dat funkcí **rtscan**

4.1.3 Knihovna RTT Simulink

Nabídku možností RTT a Simulinku otevřeme otevřeme příkazem

simulink

Objeví se okno uvedené na obr. 4.8a.

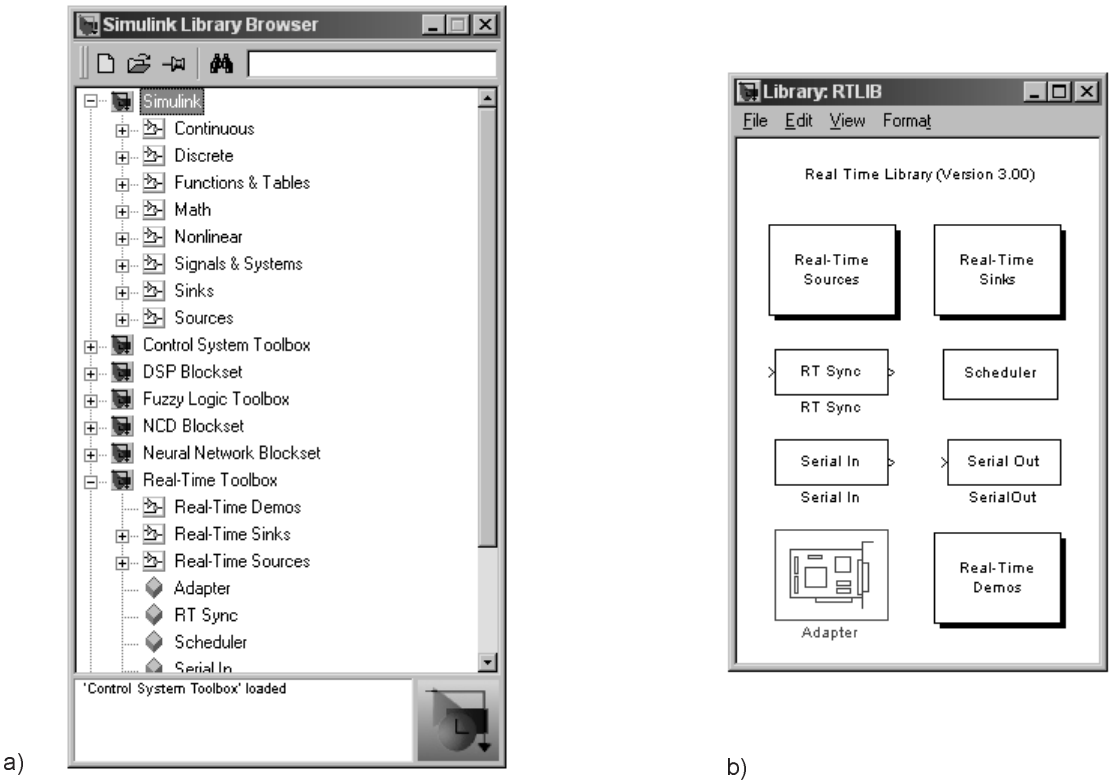
Knihovnu RTT bloků použitelných v systému Simulink otevřeme příkazem

rtlib

Objeví se okno uvedené na obr. 4.8b. Poklepáním myší na heslo Simulink otevřeme nabídku podle obr. 4.8a, ze které dále volíme **Sinks** nebo **Sources** t.j. zdroje nebo měřiče signálu. V nabídce RTT poklepáním myší na ikony **Real-Time Sources** a **Real-Time Sinks** otevřeme okna uvedená na obr. 4.9a,b. Tím máme k dispozici všechny základní bloky umožňující vytvoření úplného systému pro sběr dat.

Příklad na vstup analogového signálu:
Pracovní okno otevřeme poklepáním na nabídku **File** v některém z uvedených oken a volbou **New** a **Model**. Přetažením ikon jednotlivých prvků, vytvoříme systém zobrazující signál pohybu myši podle obr. 4.10. Poklepáním na ikonu **RT In** otevřeme okno, v němž zvolíme periodu vzorkování a počet vstupních kanálů. Poklepáním na ikonu **Adapter** otevřeme okno podle obr. 4.2a, v němž zvolíme ovládač myši **Mousedrv.rtd**. Poklepáním na ikonu **Scope** otevřeme obrazovku osciloskopu. Dobu simulace nastavíme v nabídce **Simulation**. Je zde i nabídka **Start**, kterou spouštíme činnost. V obr. 4.10 vidíme signál získaný kmitavým pohybem myši postupně ve dvou navzájem kolmých směrech. Signál byl současně uložen do souboru **untitled.mat**.

Na obrazovce osciloskopu obr. 4.10 vidíme, že signál nabývá hodnot od -1 do +1. Je to rozsah proměnných v jádře toolboxu. Skutečný vstupní nebo výstupní rozsah je



Obrázek 4.8: Okno které otevřeme příkazem a) **simulink** b) **rtlib**

nastaven spínači na kartě (např. ±5 V). Potom +5 V na vstupu karty odpovídá v jádře toolboxu +1 a -5 V na vstupu karty odpovídá v jádře toolboxu -1.

Na obr. 4.9a vidíme nabídku vstupních bloků. Standardní vstupní blok **RT In** je vhodný např. pro řízení v reálném čase, kdy každý vstupní vzorek musí být ihned zpracován procesorem pro dosažení rychlé odezvy, bez jakéhokoliv dalšího ukládání nebo zpoždění. Čas potřebný k činnosti procesoru potom může nepříznivě ovlivnit rychlost vzorkování.

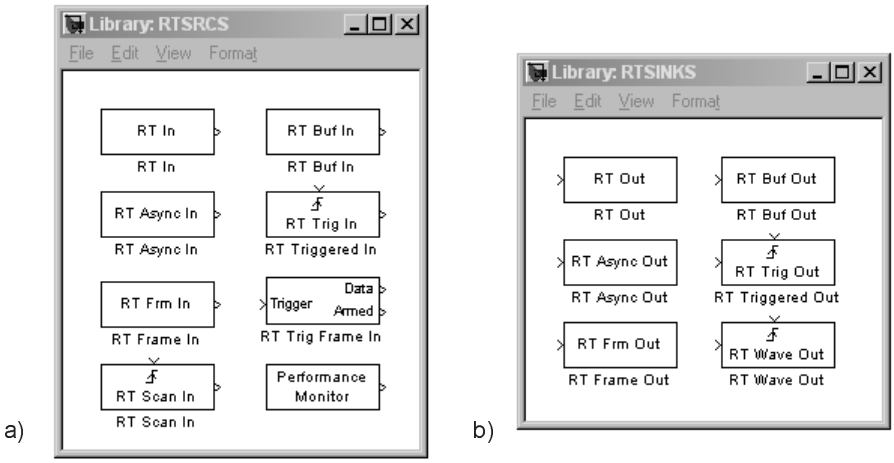
Vstupní blok **RT Buf In** je vhodný v případě, že potřebujeme nabrat rychle vstupní data a připouštíme jejich zpracování až po nabrání zvoleného počtu vzorků. Počet vzorků (Buffer Size) nastavíme v okně, které otevřeme poklepáním na ikonu RT Buf In.

Blok **RT Frm In** se využívá v případě přenosu dat po blocích, např. pro FFT.

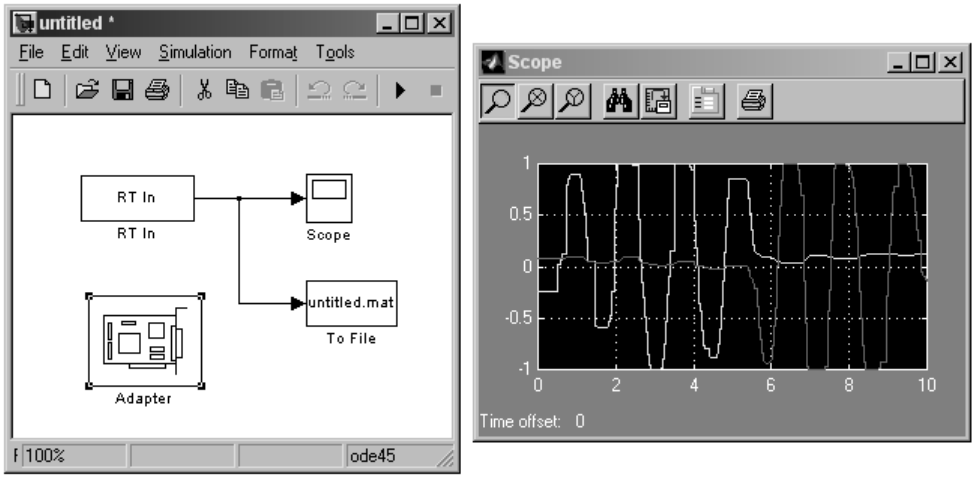
Blok **RT Trig Frame In** umožňuje ruční nebo programový start ukládání dat, včetně pretriggeringu a posttriggeringu (viz odst. 4.2.1), na zvolenou úroveň vzestupného nebo sestupného signálu a z vybraného kanálu.

K rychlému vzorkování dat se používá blok **RT Scan In**, který zajistí rychlý přenos dat do paměti s využitím DMA nebo s využitím přerušení (viz odst. 2.2.3). Rychlost vzorkování dat je omezena vlastnostmi použité karty a může dosahovat až stovek kHz [25].

Příklad na výstup analogového signálu:
Podobně jako v předchozím příkladě vytvoříme systém generující analogový signál. Použijeme blok **RT Out**, na který přivedeme signál ze signálního generátoru v nabídce Simulink/ Sources. Generovaný signál pozorujeme na osciloskopu, který připojíme na analogový



Obrázek 4.9: Okno a) vstupních bloků b) výstupních bloků



Obrázek 4.10: Zobrazení signálů z pohybu myši a jejich uložení do souboru

výstupní kanál použité karty.

Použití výstupních bloků je stejné jako u odpovídajících vstupních bloků.

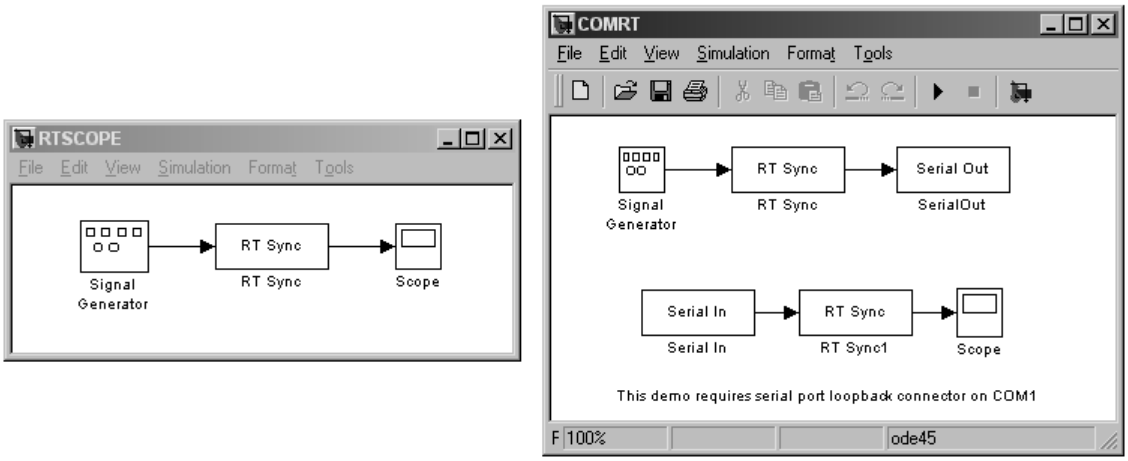
Synchronizační blok **RT Sync** zajišťuje synchronizaci mezi bloky v případě, že nenastává skutečný vstup nebo výstup signálu do nebo z počítače (obr. 4.11).

Bloky **Serial In** a **Serial Out** zajišťují sériovou komunikaci s okolím. V případě aplikací v reálném čase je nutno použít synchronizační bloky.

Scheduler je speciální blok používaný v případech pomalého vzorkování (0,1 s a více), který blokuje signál z klávesnice nebo z myši po dobu činnosti.

Performance Monitor je blok dávající informaci v rozsahu 0 a 1, která představuje poměr mezi aktuálním výkonem systému a výkonem bez simulace v reálném čase. Informace se obyčejně zobrazuje blokem Scope.

Simulink samozřejmě umožňuje současně aktivovat více ovládačů různých karet. Pro každý ovládač musíme použít jeden blok Adapter. Adresy kanálů jsou nezávislé.



Obrázek 4.11: Použití synchronizačního bloku a bloků pro sérové spojení

Jméno ovládače se pak zapíše do okna použitého vstupního nebo výstupního bloku (**HW Adapter**). Je rovněž možno použít stejný ovládač, avšak na různých adresách karet.

Sériová komunikace

RTT umožňuje sériovou komunikaci s okolím prostřednictvím sériového portu s rozhraním RS232. Použití tohoto rozhraní si ukážeme na příkladu komunikace dvou počítačů: Kabelem propojíme konektory COM1 (nebo COM2) obou počítačů.

```
c = serial            %vytvorime identifikator serioveho spojeni
                      %soucasne otevreme okno podle obr.4.12
fopen(c)              %otevre seriovy port
get(c)                %kontrola parametru

fprintf(c,'zprava')   %vyslani zpravy jednim pocitacem
fscanf(c,'%c')        %prijeti zpravy druhym pocitacem
```

4.2 Data Acquisition Toolbox

Data Acquisition Toolbox budeme dále stručně označovat DAQ. Informace o DAQ získáme zapsáním následujících příkazů do příkazového okna Matlabu

info daq help daq

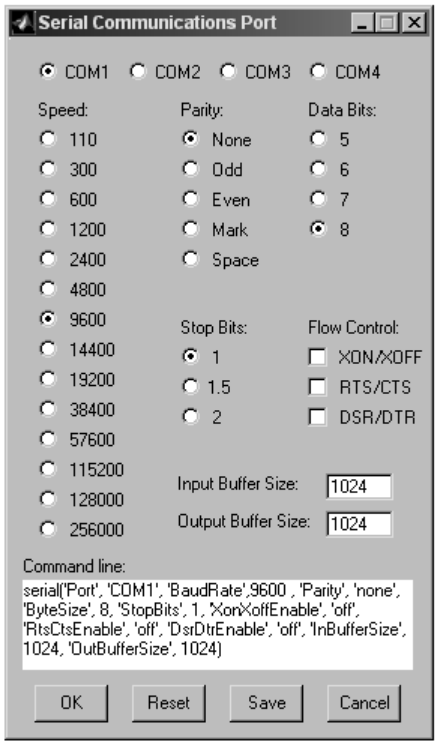
Další informace lze získat pomocí příkazů

help daqdemos demo tooolbox daq daqschool

Úvodem bude nejlépe, podívat se jak DAQ umožňuje realizovat základní činnosti pomocí zvukové karty:

1. **Čtení analogového signálu**

```
AI=analoginput('winsound');      % vytvoreni vstupniho objektu AI
                                 % s vyuzitim zvukove karty jejiz
```



Obrázek 4.12: Okno pro nastavení parametrů sériového spojení

```
                                % ovladac ma jmeno 'winsound'
addchannel (AI,1);              % k objektu AI prirad kanal 1
Fs=8000;\\                      % vzorkovaci kmitocet 8000 Hz
set(AI,'SampleRate',Fs);        % objektu AI priradi hodnotu
                                % vzorkovaciho kmitoctu
duration=2;                     % doba odberu vzorku v sekundach
set(AI,'SamplesPerTrigger',duration*Fs);
start(AI)                       % start cteni
data=getdata(AI);               % nactene udaje preved pod nazev
                                % 'data'
delete(AI)                      % zrus
plot(data)                      % zobraz
```

2. **Generace analogového signálu**

```
AO=analogoutput('winsound');    % vytvor analog. vystupni objekt
addchannel(AO, 1:2);            % objektu AO priradi
                                % vystupni kanaly 1 a 2
set(AO, 'SampleRate', 8000);
data=sin(linspace(0, 400*pi, 8000)');    % vytvori signal
plot(data)
putdata(AO, [data data]);       % priprav data na vystup
start(AO)
while strcmp(AO.Running,'On')   % cekej dokud AO neskonci
```

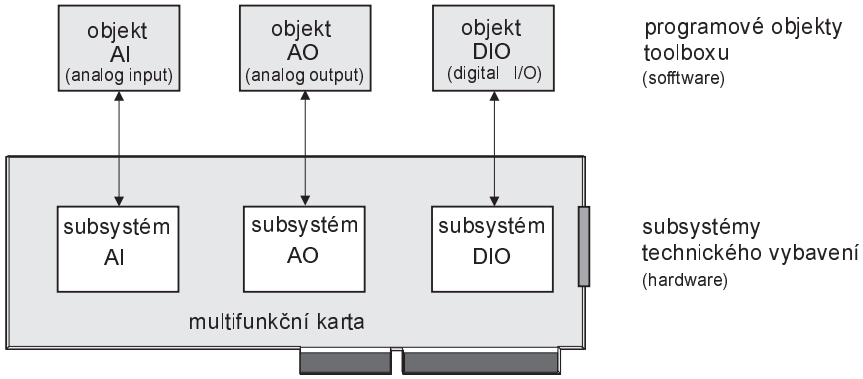
```
end
delete(AO)
```

3. **Vyslání a čtení číslicových signálů** (pomocí karty vybavené číslicovými vstupy/ výstupy):

```
DIO = digitalio('ovladac',1);     % vytvor cislicovy I/O objekt DIO
                                  % Pozn.: 1 je ID pouzite karty, lze
                                  % ho zjistit pomoci prikazu:
                                  % daqhwinfo('ovladac')
addline(DIO, 0:7, 'out');         % prirad vodice objektu
pval = [1 1 1 1 0 1 0 1];
putvalue(DIO, pval);              % zapis na cislicovy vystup
gval = getvalue(DIO);             % cti z cislicoveho vstupu
delete(DIO)
```

Z uvedených příkladů je patrno, že čtení dat ze senzorů probíhá ve dvou krocích:
1. Data se z karty uloží do jádra toolboxu.
2. Data jsou přenesena z jádra toolboxu a uložena do Matlabu, nebo do diskového souboru. Podobně při generování signálů jsou v prvním kroku data z Matlabu přenesena do jádra toolboxu a odtud v druhém kroku na výstupy použité karty.

Základními prvky toolboxu jsou **programově vytvořené objekty**, které mohou ovládat **analogový vstup**, **analogový výstup** a **číslicové vstupy/výstupy**. Spojení mezi programovými objekty a odpovídajícími subsystémy na multifukční kartě uvádí obr.4.13. K vytvořenému objektu musíme přiřadit **kanály**, případně **jednotlivé linky**.



Obrázek 4.13: Propojení programového a technického vybavení pomocí toolboxu

Toolbox dává možnost definovat obecné i speciální vlastnosti jednotlivých objektů, kanálů i jednotlivých linek. Současně je možno tyto nastavené vlastnosti číst. O tom pojednáme v následujících odstavcích.

Z obecného hlediska je často nutno zjistit jak velkou paměť budeme potřebovat. Určuje to vztah

požadovaná paměť = počet ukládaných vzorků x počet kanálů x počet bitů vzorku

Máme-li program hotový, můžeme po jeho proběhnutí zjistit rozsah použité paměti příkazem

daqmem(AI)

Dostaneme odezvu např.
ans=

Object: [1x1 analoginput]
BytesUsed: 40968
BytesAvailable: 2.1474e+009

4.2.1 Ovládání analogových vstupů

Ovládač použité multifunkční karty je automaticky registrován. Informaci o nainstalovaných ovládačích podají příkazy

daqhwinfo daqhwinfo('driver')

Analogový vstupní objekt nazvaný např. AI, vytvoříme příkazem

AI = analoginput('driver',ID)

kde 'driver' je název ovládače karty a ID je hardwarový identifikátor použité karty. Pro zvukovou kartu je implicitně nastaveno číslo 0 (nemusí se uvést). ID jiných použitých karet získáme pomocí výše uvedených příkazů. Programový objekt lze zrušit příkazem

delete(AI)

Je-li objekt nainstalován lze o něm získat informace příkazy

get(AI) whos AI

Funkce get slouží k zobrazování hodnot definovaných vlastností objektů, kanálů i linek. Např. vlastnosti kanálu 1 zobrazíme příkazem get(AI.Channel(1)).

K vytvořenému objektu připojíme hardwarové kanály příkazem

data = addchannel(obj, hwch, index, names)

kde obj je analogový vstupní objekt,
hwch specifikuje čísla kanálů,
index a names jsou nepovinné parametry blíže specifikující použité kanály,
data je nepovinný název vektoru do kterého chceme ukládat čtená data.
Např. příkazem addchannel(AI, 0:3) přiřadíme objektu první 4 kanály na kartě, které si Matlab očísluje 1 až 4.

Funkcí set lze nastavit následující **vlastnosti vstupních kanálů**:
* Channel - hardwarové kanály přiřazené objektu,
* SampleRate - vzorkovací kmitočet kanálu,
* SamplesPerTrigger - počet vzorků na každý **start záznamu dat** (trigger), který přijde,
* TriggerType - specifikuje způsob startu záznamu dat.

Např. set(AI, 'SampleRate', 44100); nastaví vzorkovací kmitočet na 44,1 kHz. Podobně set(AI, 'SamplesPerTrigger', 8000); nastaví odběr 8000 vzorků na každý start záznamu.

Vlastnosti objektu se vypíší na obrazovku, jestliže do příkazového řádku napíšeme jméno objektu, např. AI. Vlastnosti kanálu vypíše příkaz AI.Channel.

Čtení dat je zahájeno příkazem

start(AI)

Pokud se paměť naplní, pak jsou přepisována. Chceme-li je přesunout použijeme funkci

getdata. Např. příkaz

data = getdata(AI, 100)

uloží pod názvem data prvních 100 vzorků do matice 100 x n, kde n je počet vstupních kanálů. Čtení dat lze zastavit příkazem

stop(AI)

Trvalý záznam dat začne probíhat od okamžiku, který jsme nazvali **start záznamu dat** (trigger). Od tohoto okamžiku jsou data ukládána do paměti jádra toolboxu nebo do diskového souboru. Podívejme se podrobněji na **typ startu záznamu dat**. Lze jej zvolit následovně:
* Immediate - okamžitě při provedení instrukce start,
* Manual - ručním spuštěním funkce trigger,
* Software - začne ukládat vzorky po splnění podmínky záznamu dat označené TriggerCondition.
Kromě toho existují specifické příkazy spouštění činností desek podle výrobců karet (National Instruments, Hewlett-Packard), jejichž popis je nad rámec tohoto textu [26].

Zobrazení uvedených možností v příkazovém okně Matlabu dosáhneme příkazem set(AI, 'TriggerType'). Např. ruční spuštění záznamu dat definujeme příkazem

set(AI, 'TriggerType', 'Manual')

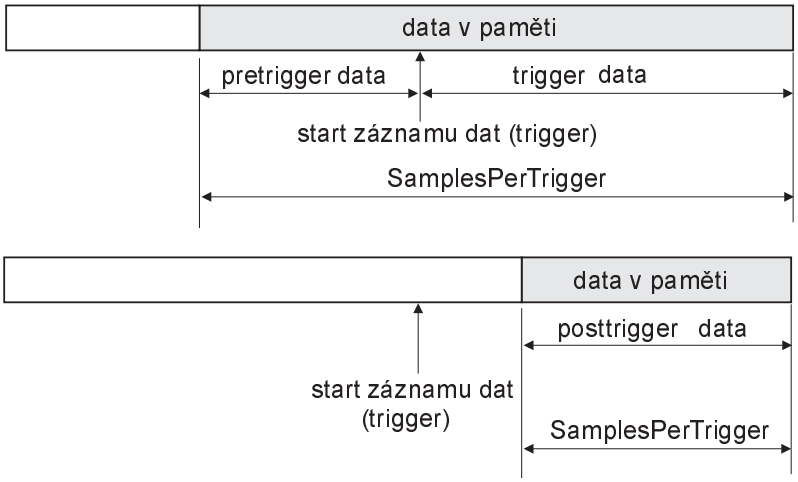
a spustíme příkazem trigger(AI) za podmínky, že předcházel příkaz start(AI).

DAQ toolbox dává možnost nastavení nebo čtení **vlastností startu záznamu dat**:
* InitialTriggerTime - indikuje absolutní čas prvního startu,
* TriggerAction - provede po startu specifikovanou M-funkci,
* TriggerChannel - specifikuje kanály sloužící jako zdroje startovacího signálu,
* TriggerCondition - specifikuje podmínku startu,
* TriggerConditionValue - specifikuje jednu nebo několik napěťových úrovní, jejichž dosažení je podmínkou startu,
* TriggerDelay - specifikuje zpoždění startu,
* TriggerDelayUnits - specifikuje jednotky v nichž je zpoždění měřeno,
* TriggerRepeat - specifikuje počet opakování startu,
* TriggersExecuted - indikuje počet provedených startů.

Podmínka TriggerCondition, nutná pro programový start záznamu dat, nabízí následující možnosti:
* Rising - start na vzestupnou část signálu,
* Falling - start na sestupnou část signálu,
* Leaving - start v okamžiku, kdy signál opustí specifikované pásmo hodnot,
* Entering - start v okamžiku, kdy signál vstoupí do specifikovaného pásma hodnot.

V souvislosti se zpožděním startu záznamu dat (TriggerDelay) si vysvětlíme pojmy **pretriggering** a **posttriggering**, obr. 4.14. Je-li hodnota zpoždění kladná, dojde k vlastnímu záznamu dat až specifikovanou hodnotu po startu (posttriggering). Je-li hodnota zpoždění záporná, dojde k vlastnímu záznamu dat již specifikovanou hodnotu před startem (pretriggering). Pretriggering se užívá k zachování záznamu situace před nějakým jevem, který spouští start záznamu dat. Tato zdánlivá nelogičnost se snadno vysvětlí tím, že vzorky jsou průběžně ukládány v paměťovém bloku určité velikosti a periodicky přepisovány. V okamžiku příchodu startu ukládání se zablokuje přepis specifikovaného počtu již před tím zapsaných vzorků. Velikost paměťového bloku potom určuje maximální

hodnotu pro pretriggering i posttriggering.



Obrázek 4.14: Pretriggering a posttriggering

Počet vstupů a **vzorkovací kmitočet** nastavíme takto

```
AI=analoginput('winsound');
addchannel(AI, 1:2);
set(AI,'SampleRate',16000);
```

Zvolili jsme zvukovou kartu stereo (t.j. 2 kanály) a vzorkovací kmitočet 16 kHz. Je nutno si uvědomit, že

max.vzorkovací kmitočet kanálu = max.vzork.kmit.karty/počet čtených kanálů

Takže v našem případě bude každý kanál čten max.vzorkovacím kmitočtem 8 kHz.

Typ analogového vstupu multifunkční karty může být (viz obr. 3.3):

* Differential
* SingleEnded
* NonReferencedSingleEnded

Většinou se typ vstupu nastaví spínači na kartě a je nutno o tom vyrozumět program příkazem např.

```
set(AI, 'InputType', 'SingleEnded')
```

Běžné multifunkční karty užívané pro sběr dat mají pouze jeden A/Č převodník, na který jsou multiplexovány jednotlivé vstupní kanály. Postupně je každý analogový vstup vzorkován a převeden na číslo. Protože kanály nelze vzorkovat současně, vzniká mezi jejich zpracováním určitá časová prodleva, která je v DAQ toolboxu označena ChannelSkew. Pomocí veličiny ChannelSkewMode lze specifikovat, jak má být tato hodnota stanovena [26]. Samozřejmě pro karty s několika vzorkovacími jednotkami, které umožňují simultánní vzorkování, tyto veličiny nemají smysl.

Použití uvedených příkazů snadněji pochopíme na **příkladu**:
Vytvoříme analogový vstupní objekt pro zvukovou kartu - potřebné informace o kartě získáme příkazem daqhwinfo. Data budeme snímat jedním kanálem, který pojmenujeme chan po dobu dvou sekund vzorkovacím kmitočtem 44,1 kHz. Z dat tohoto kanálu odvodíme start ukládání dat při vzestupném signálu a překročení úrovně 0,2 V. Před tímto okamžikem

chceme zachovat pretrigger 500 vzorků. Napíšeme následující program:

```
AI = analoginput('winsound');       %vytvorime vstupni analogovy
                                    %objekt pro zvukovou kartu
chan = addchanel(AI, 1);
set(AI, 'SampleRate', 44100);
duration = 2;                       %ulozime dobu trvani
konst = get(AI, 'SampleRate');      %konst = SampleRate
set(AI, 'SamplesPerTrigger', konst*duration);
set(AI, 'TriggerChannel', chan);
set(AI, 'TriggerType', 'Software');
set(AI, 'TriggerCondition', 'Rising');
set(AI, 'TriggerConditionValue', 0.2);
set(AI, 'TriggerDelayUnits', 'Samples');
set(AI, 'TriggerDelay', -500);

start(AI);
while(AI.Running, 'On')
end
[d, t] = getdata(AI, 1000);         %vyjmi z jadra 1000 vzorku
plot(t, d);                         %zobraz
xlabel('Cas [s]');
ylabel('Signal [V]');
grid on;
```

Prohlížení dat objektu v činnosti zajišťuje příkaz

data = peekdata(obj, samples)

kde obj je analogový vstupní objekt,
samples je počet vzorků,
data je matice m x n (m je počet vzorků, n je počet kanálů).
Funkci peekdata lze použít až po spuštění činnosti objektu funkcí start. Data zůstávají v jádře toolboxu. Např. zobrazení posledních 100 načtených vzorků zajistíme takto

```
data = peekdata(AI, 100);
plot(data)
```

Potřebujeme-li však data zpracovávat ihned Matlabem musíme je **vyjmout z jádra** toolboxu pomocí funkce getdata. Obecný zápis příkazu vypadá následovně

[data, time, abstime, events] = getdata(obj, samples, type)

kde obj je analogový vstupní objekt,
samples je volitelný parametr udávající počet vzorků k vyjmutí,
type je volitelný parametr specifikující formát vzorků,
data je matice m x n vzorků (m je počet vzorků, n je počet kanálů),
time je matice m x 1 s časovými údaji pro všech m vzorků,
abstime je absolutní čas prvního startu záznamu vzorků,
events je seznam **událostí**, které proběhly do spuštení funkce getdata.

V předešlém textu jsme již uvedli řadu příkladů na použití této funkce. Chceme-li získat data s údaji **relativního času** (od příchodu startu ukládání dat) použijeme příkaz

[data,time] = getdata(AI).

Informaci o **absolutním čase** startu ukládání dat získáme příkazem

[data,time,abstime] = getdata(AI);

Absolutní čas se po zapsání abstime do příkazového řádku Matlabu vypíše jako vektor ve tvaru

[rok měsíc den hodina minuta sekunda]

Informace o uložených datech můžeme získat pomocí následujících parametrů:
* SamplesAcquired - indikuje počet vzorků nabraných jedním kanálem,
* SamplesAvailable - indikuje počet vzorků dostupných v jádře (na jeden kanál),
* SamplesPerTriger - specifikuje počet vzorků na kanál a start záznamu dat.

Např. instrukci typu while AI.SamplesAcquired $<$ *konstanta* lze využít pro větvení programu.

Vyžádáme-li si **informace o událostech** postupujeme následovně. Zapsáním názvu proměnné events do příkazového řádku Matlabu dostaneme informaci, že se skládá z polí Type a Data. Zadáme-li events.Type, vypíše se seznam **typů událostí** :
* Data missed - data neuložena do jádra toolboxu,
* Input overrange - vstupní signál přesahuje nastavený vstupní rozsah karty,
* Run-time error - chyba v provádění programu (zahrnuje i hardwarové chyby),
* Samples acquired - definovaný počet vzorků byl přečten a uložen do jádra toolboxu,
* Start - byl vydán příkaz start,
* Stop - čtení dat zastaveno,
* Timer - uběhl definovaný časový úsek,
* Trigger - došlo ke startu záznamu dat.

Zapíšeme-li events.Data, vypíše se informace o jednotlivých událostech obsahující především údaj o jejich absolutním čase, údaj o počtu odebraných vzorků aj. podle typu události. **Vlastnosti údálostí** mají definované následující názvy z nichž je patrný i jejich význam: DataMissedAction, InputOverRangeAction, RutimeErrorAction, SamplesAcquiredAction, SampledAcquieredActionCount, StartAction, StopAction, TimerAction, TimerPeriod, TriggerAction.

Jestliže některá z událostí nastane, lze spustit tzv. **akční funkci** příkazem

set(obj, 'PropertyName', 'actionfcn');

kde PropertyName je název události z předešlého seznamu,
actionfcn je M-file nebo start, stop, nebo trigger funkce toolboxu. Akční funkce se vytváří příkazem

function actionfcn(obj, events)

Další důležitou vlastností DAQ je **nastavitelnost měřítka**. Nejlépe si to objasníme na **příkladu**:

Akcelerometr dává signál o napětí ±1,5 V. Analogový vstup multifunkční karty firmy National Instruments má nejbližší vhodný rozsah ±2,5 V. Rozsah karty by tedy normálně byl využit pouze na 60 %. Navíc signál ±1,5 V odpovídá zrychlení $\pm 5\,g$ ($1g = 9{,}81$ m/s^2). Sestavíme následující program:

```
AI = analoginput('nidaq', 1);       %nazev ovladace a ID karty
                                    %zjistime prikazem daqhwinfo
chan = addchannel(AI, 1);
set(chan,'SenzorRange',[-0.6 0.6]);%nastavi rozsah senzoru tak,
```

```
                                    %aby plne vyuzil vstupni rozsah karty
set(chan,'InputRange',[-2.5 2.5]); %analogovy vstupni rozsah kanalu
set(chan,'UnitsRange',[-2 2]);     %z 2,5 udela 5 v jadre toolboxu
set(chan,'Units','g''s (1g=9.81m/s/s)');  %jednotce v jadre priradi g

start(AI);                         %dalsi cast programu je jiz znama
while strcmp(AI.Running, 'On')     %cekej dokud AI neskonci
end
data = getdata(AI);
plot(data)
delete(AI)
```

DAQ umožňuje **ukládat do diskového souboru**:
∗ čtená data,
∗ informace o událostech,
∗ informace o objektu a kanálech,
∗ informace o technickém vybavení. Nejstručnější výklad v tomto případě je příklad: Použijeme zvukovou kartu pro ukládání dat dvěma kanály po dobu dvou sekund po každém startu záznamu dat. Start záznamu dat se bude opakovat třikrát vzorkovacím kmitočtem 8 kHz. Data a události uložíme do souboru file00.daq. Sestavíme následující program:

```
AI = analoginput('winsound');
chan = addchannel(AI, 1:2);
duration = 2;
set(AI, 'SampleRate', 8000);
konst = get(AI, 'SampleRate');          %konst = SampleRate
set(AI, 'SamplesPerTrigger', duration*konst);
set(AI, 'TriggerRepeat', 3);            %opakovani zaznamu 3krat
set(AI, 'LogFileName', 'file00.daq');  %jmeno souboru pro zaznam
set(AI, 'LogToDiskMode', 'Overwrite'); %nastavi rezim prepisu dat
set(AI, 'LoggingMode', 'Disk&Memory'); %misto ulozeni

start(AI)
while strcmp(AI.Running, 'On');
end
[data, time] = daqread('file00.daq');   %cteni ze souboru
```

Místo pro uložení informace volíme parametrem LoggingMode, který může být buď Disk nebo Disk&Memory.

Parametrem LogToDiskMode určujeme, zda vytvoříme jeden opakovaně přepisovaný soubor (Overwrite) nebo zvláštní soubor pro každý start záznamu dat (Index).

Obecný zápis instrukce pro **čtení dat ze souboru** vypadá následovně
[data, time, abstime, events, daqinfo] = daqread('file','P1',V1,'P2',V2,...)
kde je pro nás nový pojem daqinfo. Je to struktura v níž jsou uloženy informace o objektu, kanálech a technickém vybavení ve dvou polích: ObjInfo a HwInfo. Informace o tech-

nickém vybavení uložená v poli HwInfo odpovídá informaci, kterou získáme příkazem daqhwinfo(obj).

Parametry typu P1 a V1 jsou volitelné dvojice veličin *jméno vlastnosti* a *hodnota vlastnosti*. Veličina *jméno vlastnosti* může být:
* Samples - specifikuje počet vzorků,
* Time - specifikuje rozsah relativního času,
* Triggers - specifikuje počet startů záznamu dat,
* Channels - specifikuje použité kanály,
* DataFormat - specifikuje formát dat,
* TimeFormat - specifikuje formát času.

4.2.2 Ovládání analogových výstupů

Úvodem potodýkáme, že význam většiny dále uvedených pojmů lze najít v odst. 4.2.1. Ve významu některých pojmů je pouze činnost ukládání dat nahrazena činností vysílání dat.

Analogový výstup mění číslicově uložená data na reálný analogový signál. Je to v podstatě inverzní činnost k činnosti analogových vstupních subsystémů. Typická multifunkční karta má 2 výstupní analogové kanály s 12bitovým rozlišením.

Registrace ovládače karty proběhne automaticky. Dále vytvoříme analogový výstupní objekt, nazvaný např. AO, příkazem

AO = analogoutput('driver', ID);

kde 'driver' je název ovládače použité karty zjistitelný příkazem daqhwinfo. ID je identifikační číslo karty, které je pro zvukovou kartu implicitně 0 (nemusí se uvést) a pro jiné karty se dá zjistit příkazem daqhwinfo('driver'). Programový objekt lze zrušit příkazem

delete(AO)

Je-li objekt nainstalován lze o něm získat informace příkazy

get(AO) whos AO

Funkcí set lze nastavit **vlastnosti výstupních kanálů** a to: Channel, SampleRate, TriggerType.

Vlastnosti **startu vysílání dat** jsou následující: InitialTriggerTime, TriggerAction, TriggersExecuted, TriggerType.

Typ startu vysílání dat (TriggerType) může být Immediate a Manual. Kromě toho existují specifické příkazy spouštění činností desek podle jednotlivých výrobců (National Instruments, Hewlett-Packard), jejichž popis je nad rámec tohoto textu [26].

Typy událostí (events) jsou následující: Run-time error, Start, Stop, Timer, Trigger a navíc Samples output, který specifikuje počet vzorků vyslaných z jádra toolboxu. Události mají následující názvy vlastností (PropertyName) z nichž plyne i jejich význam: RunTimeErrorAction, SamplesOutputAction, SamplesOutputActionCount, StartAction, StopAction, TimerAction, TimerPeriod, TriggerAction.

Jestliže některá z událostí nastane, lze spustit tzv. **akční funkci** příkazem

set(obj, 'PropertyName', 'actionfcn');

kde PropertyName je název události z předešlého seznamu,
actionfcn je M-file nebo start, stop, nebo trigger funkce toolboxu. Akční funkce se vytváří

příkazem

function actionfcn(obj, events)

Výstupní data je nutno připravit v jádře toolboxu funkcí

putdata(obj, data)

kde obj je analogový výstupní objekt,
data je označení zdrojového souboru dat uspořádaného do matice m x n, kde m je počet vzorků a n je počet kanálů.

Vlastní vyslání dat spouští příkaz start(obj). V případě že TriggerType je Manual dojde ke startu vysílání dat až následujícím příkazem trigger(obj).

Pro vytvoření a použití akčních funkcí platí totéž co v odst. 4.2.1.

Uvedeme si opět **příklad**:
Vytvoříme analogový výstupní objekt pro zvukovou kartu se dvěma kanály. Vysílání signálu se má opakovat čtyřikrát. Po vyslání 8000 vzorků se má spustit akční funkce označená 'afce'. Sestavíme následující program:

```
AO = analogoutput('winsound');
ch = addchannel(AO, 1:2);
set(AO, 'SampleRate', 8000);
konst = get(AO, 'SampleRate');
set(AO, 'RepeatOutput', 4);
set(AO, 'SamplesOutputActionCount', 8000);
frekv = get(AO, 'SamplesOutputActionCount');
set(AO, 'SamplesOutputAction', 'afce');
data = sin(linspace(0, 800*pi, 3*frekv))';
putdata(AO, [data, data]);

AO                          %zobrazi informaci o AO
start(AO);
pause(0.1)
```

Akční funkce je definována následovně

```
function afce(obj, event)
samp = obj.SamplesOutput;
disp([num2str(samp), 'vzorky byly vyslany'])
```

Parametry umožňující **organizovat výstup dat** jsou:
* MaxSamplesQueued - indikuje maximální počet vzorků, které lze řadit v jádře toolboxu,
* RepeatOutput - specifikuje počet opakování výstupu dat,
* TimeOut - specifikuje čekací dobu řazení dat.

Nastavitelnost měřítek opět umožňují parametry Units, UnitsRange a navíc OutputRange, který odpovídá nastavenému analogovému výstupnímu rozsahu karty.

DAQ umožňuje vytvořit a spouštět několik vstupních i výstupních objektů současně. Pak některé instrukce vypadají např. takto:
set([AI AO], 'TriggerType', 'Manual', 'SampleRate', 8000);

.....
start([AI AO]);
trigger([AI AO]);
stop([AI AO])

4.2.3 Ovládání číslicových vstupů/výstupů

Číslicové subsystémy počítače jsou určeny k příjmu a vysílání dat v číslicové formě. Vstupy a výstupy (dále I/O) mohou tvořit jednotlivé **linky** (vodiče) nebo skupiny obyčejně 8 linek, nebo jejich násobku, označované jako **porty**.

Registrace ovládače karty proběhne automaticky. Číslicový I/O objekt vytvoříme příkazem

DIO = digitalio('driver', ID);

kde 'driver' je název ovládače použité karty zjistitelný příkazem daqhwinfo. ID je identifikační číslo karty, které se dá zjistit příkazem daqhwinfo('driver'). Programový objekt lze zrušit příkazem

delete(DIO)

Je-li objekt nainstalován lze o něm získat informace příkazy

get(DIO) whos DIO

Číslicovému I/O objektu **přiřadíme linky** příkazem

lines = addline(obj, hwline, port, 'direction', 'names');

kde obj je číslicový I/O objekt,
hwline je ID linek,
port (volitelný parametr) je ID číslicového I/O portu,
'direction' určuje **směr** In nebo Out,
names (volitelný parametr) umožňuje linky pojmenovat,
lines (volitelný parametr) je vektor obsahující číslicové I/O linky.

Nastavené **parametry lze číst** příkazy např.:
hwinfo=daqhwinfo(DIO) hwinfo.Port.LineIDs hwinfo.Port(ID) hwinfo.Port.Direction
Uvedeme si **příklady**:
1) Vyvoříme číslicový I/O objekt a přiřadíme mu 24 výstupních linek

```
DIO = digitalio('nidaq', 1);    %karta National Instruments
addline(DIO, 0:7, 'out');       %nulty port
addline(DIO, 0:7, 1, 'out');    %prvni port
addline(DIO, 0:7, 2, 'out');    %druhy port
```

nebo jednou instrukcí
addline(DIO, 0:23, 'out');

2) DIO přiřadíme 4 vstupní a 4 výstupní linky portu 0 příkazem
addline(DIO, 0:7,'in','in','in','in','out','out','out','out');

3) DIO přiřadíme první linku portu 0 a portu 2 pro vstup a druhou linku stejných portů pro výstup příkazem
addline(DIO, [0 1 8 9] 'in','out','in','out');
nebo dvěma příkazy

```
addline(DIO, 0, [0 2], 'in');
addline(DIO, 2, [0 2], 'out');
```

Číslicová data **posíláme na linky** příkazem

```
data = 23;
putvalue(DIO, data);
```

nebo přímo na jednotlivé linky příkazem

```
putvalue(DIO.Line(1:8), data);
```

Desítkové vyjádření dat lze použít do 32 linek. Nad tento počet je nutno data zadávat ve dvojkovém tvaru, např.: $23 = [1\ 1\ 1\ 0\ 1] = 2^0 + 2^1 + 2^2 + 2^4$. První element vektoru je LSB a poslední element je MSB.

Desítkově vyjádřené číslo převedeme na dvojkový tvar příkazem

```
out = dec2binvec(dec, bits)
```

kde dec je dekadická hodnota,
bits je požadovaný počet bitů,
out je výsledek.
Naopak dvojkový tvar čísla převedeme na desítkový příkazem

```
out = binvec2dec(bin)
```

kde bin je dvojkový tvar čísla (viz výše).

Čtení dat z číslicových linek zajistíme příkazem

```
out = getvalue(obj)
out = getvalue(obj.Line(index))
```

Např. první čtyři linky čteme a uložíme do souboru out příkazem

```
out = getvalue(DIO.Line(1:4))
```

Číslicový objekt **uvedeme do činnosti** příkazem

```
start(DIO)
```

a **zastavíme** příkazem

```
stop(DIO)
```

Vlastnosti časovače číslicového objektu jsou:
* Running - indikuje aktivní stav objektu,
* TimerAction - specifikuje akční funkci, která se spustí po proběhnutí definovaného časového úseku,
* TimerPeriod - specifikuje časový úsek mezi událostmi.

Příklad:

```
DIO = digitalio('nidaq'), 1);
addline(DIO, 0:7, 'in');
set(DIO, 'TimerAction', 'akcnifce');
set(DIO, 'TimePeriod', 5.0);
start(DIO)
delete(DIO)
```