SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-51347

ZBER ÚDAJOV Z REÁLNEJ SÚSTAVY V OPEN SOURCE OS

2010

Dávid Nemčok

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

ZBER ÚDAJOV Z REÁLNEJ SÚSTAVY V OPEN SOURCE OS

Bakalárska práca

Študijný program:	aplikovaná informatika
Študijný odbor:	9.2.9 aplikovaná informatika
Školiace pracovisko:	Ústav riadenia a priemyselnej informatiky (FEI STU)
Vedúci práce:	Ing. Michal Sedlák

Bratislava 2010
Dávid Nemčok

Abstrakt

Slovenská technická univerzita v Bratislave
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program: APLIKOVANÁ INFORMATIKA

Autor: Dávid Nemčok
Názov bakalárskej práce: Zber údajov z reálnej sústavy v open source OS
Vedúci bakalárskej práce: Ing. Michal Sedlák
Termín odovzdania: Máj 2010
Kľúčové slová: zber údajov, Linux, Scilab, komunikačný modul

Práca sa zaoberá možnosťami zberu údajov v slobodnom softvéry pod slobodným operačným systémom. V úvode približuje problematiku slobodného softvéru. Stručne opisuje vstupno-výstupný subsystém operačného systému Linux. Vysvetľuje základné princípy a spôsoby zberu údajov a prezentuje prehľad slobodného softvéru na zber údajov. Analyzuje vytvorenie komunikačného modulu pre dostupnú reálnu sústavu. Následne opisuje implementáciu komunikačného modulu v Scilabe použitím knižnice/ovládača hudaqlib. Ďalej popisuje testovanie komunikačného modulu pri riadení magnetickej levitácie a analyzuje vlastnosti navrhnutého riešenia. Výsledky tejto práce môžu byť použité pri zbere údajov a riadení systémov pomocou Xcosu, pri použití meracej karty MF624 od Humusoftu. Všetok použitý softvér je voľne dostupný na internete.

Abstract

Slovak University of Technology in Bratislava
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Degree Course: Applied Informatics

Author: Dávid Nemčok
Title of the bachelor theses: Data acquisition from real system in open source OS
Supervisor: Ing. Michal Sedlák
Submission time: May, 2010
Key words: data acquisition, Linux, Scilab, communication module

This paper discusses the data acquisition in the free software under free operating system. In the beginning presents the free and open source software. Briefly describes the I/O subsystem of the Linux operating system. It explains the basic principles and methods of data acquisition and presents a survey of free software for data acquisition. It discusses the development of a communication module for the available real system. And then describes the implementation of a communication module in Scilab, using the hudaqlib library/driver. Also contents testing of the communication module and describes the control of magnetic levitation and analyze the characteristics of the proposed solutions. The results of this work may be used in data acquisition and control, using Xcos and the data acquisition board MF624 from Humusoft. All needed software is available on the web and can be freely downloaded.

Čestné vyhlásenie

Vyhlasujem, že som bakalársku prácu vypracoval samostatne s využitím uvedených zdrojov literatúry.

.....

vlastnoručný podpis

V Bratislave, dňa .....

Poďakovanie

Týmto by som chcel poďakovať Ing. Michalovi Sedlákovi za odbornú pomoc pri tvorbe tejto bakalárskej práce.

Zoznam použitých skratiek

GUI	Graphical User Interface
PCI	Peripheral Component Interconnect
OS	operačný systém
DAQ	data acquisition – zber údajov
I/O	vstup/výstup, vstupno-výstupný
USB	univerzálna sériová zbernica
GPL	všeobecná verejná licencia
ISA	zbernica pre IBM kompatibilné počítače
FSF	Free software foundation
OSI	Open source initiative
HIL	Hardware in the loop
FOSS	Free and open source software
POSIX	Portable Operating System Interface for Unix
MS	Microsoft

Obsah

Úvod

Slobodný softvér je voľne dostupný všetkým používateľom, tento softvér je možné nie len slobodne používať, ale ak autor uvoľnil aj zdrojový kód (pod slobodnou licenciou), tak ho používatelia môžu aj upravovať a vylepšovať. Slobodný softvér poskytuje dostatok možností pre zber údajov [7][8], prácu s reálnym časom [11], analýzu a testovanie údajov [12], takisto poskytuje aj simulačné a vizualizačné nástroje [10][12].

Zber údajov (data aquisition) je proces spracovávania veličín reálneho sveta na digitálne hodnoty, ktoré môžu byť spracované počítačmi. Rozhranie medzi počítačom a spracovávaným signálom sa označuje DAQ hardvér. Existujú rôzne druhy DAQ zariadení pre PC. K dispozícii sú PCI, PCI Express, PCMCIA, USB, bezdrôtové a ethernetové zariadenia pre meranie, testovanie a automatické riadenie. Medzi funkcie DAQ hardvéru patria:

- Analógové vstupy/výstupy
- Digitálne vstupy/výstupy
- Počítadlá/časovače
- Enkódery – inkrementálne snímače
- Kombinácie týchto funkcií v jednom zariadení

Pre plnohodnotné využitie DAQ hardvéru je potrebný softvérový ovládač pre OS a aplikačný softvér. Softvérový ovládač je vrstva softvéru, ktorá sprostredkúva komunikáciu s DAQ zariadením. Ovládač tvorí strednú vrstvu medzi hardvérom a aplikačným softvérom. Ovládač taktiež uľahčuje prácu programátorovi, ktorý nemusí používať zložité príkazy na prístup k zariadeniu. Aplikačný softvér pridáva k ovládaču analytické a prezentačné funkcie. Softvér mení PC a DAQ zariadenie na analytický a prezentačný nástroj. V tejto práci budú popísané tieto súčasti DAQ systémov.

Cieľom tejto práce je preskúmať možnosti zberu údajov z reálneho zariadenia s využitím slobodného softvéru, a implementovať komunikačný modul do aplikačného softvéru.

1 Teória

V tejto časti bude vysvetlené čo je to slobodný softvér, popísaný vstupno-výstupný subsystém operačného systému Linux a základné vlastnosti rozhrania PCI.

1.1 Slobodný software a open source

Definícia slobodného softvéru (*free software*) pochádza od FSF, ktorá pre jeho podporu vytvorila GPL. Cieľom GPL nie je obmedzovať použitie licencovaného softvéru, ale naopak chrániť právo používateľa a vývojárov na prístup k zdrojovým kódom. GPL dovoľuje robiť zmeny v slobodnom softvéry vytvorenom pod touto licenciou. Napríklad je možné opraviť chybu vo veľkej aplikácií akou je Open Office. Problémom je, že anglické slovo „free“ má nielen význam slova „slobodný“, ale aj slova „zdarma“. Veľa ľudí si preto mýli koncept slobodného softvéru so softvérom, ktorý je zdarma. V praxi je to tak, že väčšina slobodného softvéru je zdarma, ale existujú organizácie, ktoré zarábajú peniaze na poskytovaní technickej podpory a ďalších služieb k softvéru licencovanému pod GPL.

Ďalšia organizácia, ktorá sa zaoberá voľným šírením softvéru je OSI, ktorá zaviedla pojem „open source“, čo v preklade znamená otvorený zdroj (zdrojový kód). V prístupe FSF a OSI sú viaceré rozdiely, ktoré však nepotrebujeme podrobne rozoberať. Pretože na zníženie chaosu okolo pojmov „free software“ a „open source“ bol vytvorený zjednocujúci pojem FOSS/FLOSS, z anglického *„free/libre open source software“*, čo v preklade znamená približne: „voľný slobodný otvorený softvér“. Pojem FOSS sa používa bez zaujatosti voči nejakému z týchto prístupov FSF a OSI. Takýto softvér sa nezameriava len na GPL licenciu, ale kladie dôraz aj na softvér, ktorý poskytuje svoje zdrojové kódy, aj keď tie nemusia byť pod licenciou GPL.

1.1.1 Slobodné operačné systémy

Slobodné operačné systémy patria tiež medzi FOSS. Medzi najznámejšie slobodné OS patria:

- GNU/Linux
- FreeBSD, NetBSD, OpenBSD
- OpenSolaris

Každý z týchto OS môžno označiť ako Unix-like, lebo spĺňajú špecifikáciu POSIX, takže majú spoločné vlastnosti. V prvom rade sme si pre našu úlohu potrebovali zvoliť vhodný slobodný operačný systém. Najvhodnejší OS pre našu úlohu bol GNU/Linux, pretože je z nich najrozšírenejší a je pre neho dostupný slobodný softvér na zber údajov.

1.2 Vstupno-výstupný subsystém operačného systému Linux

Podľa informácií uvedených v [1]. GNU/Linux a iné systémy vychádzajúce z Unixu sú závislé na základnej abstrakcií súboru, ktorá je používaná k reprezentácii mnohých rôznych funkcii operačného systému a zariadení. Ako sa často hovorí: „Všetko je súbor“. V skutočnosti je to tak, že väčšina zariadení v Linuxe je reprezentovaná súbormi v adresári */dev*. Pomocou týchto súborov je možné otvárať dostupné zariadenia a používať obvyklé I/O funkcie pre interakciu s hardvérom. V súčastnosti sa pre zariadenia používajú rôzny démoni ako *udev*, ktoré za behu vytvárajú nové uzly zariadení, vždy keď je detekované nové zariadenie a tiež zabezpečujú načítanie vhodných ovládačov pre tieto zariadenia.

Linux delí zariadenia na 3 typy:

a) Znakové zariadenia
b) Blokové zariadeniam
c) Sieťové zariadenia

a) Znakové zariadenia - zahŕňajú všetky zariadenia, do ktorých sa vždy zapisuje alebo číta sekvenčným spôsobom (tzn. jeden znak v daný okamžik). Do týchto zariadení môžeme zapisovať rovno celý zásobník dát, ale operácie na pozadí budú vždy prebiehať sekvenčne. Za typické znakové zariadenia môžeme označiť klávesnicu a myš, pretože vždy reprezentujú súčasný stav hardvéru.

b) Blokové zariadenia - je možné adresovať náhodne, ako napríklad ovládač disku. Je úplne jedno, ktoré dáta chceme z disku čítať, pretože môžeme špecifikovať, odkiaľ chceme čítať. Pri vytváraní ovládača pre blokové zariadenie sa netreba zaoberať priamo používateľskými I/O operáciami ako u znakového zariadenia, lebo existujú funkcie, ktoré umožňujú jadru efektívne vykonávať rôzne I/O operácie, pričom jadro sa už samo stará o prezentáciu zariadenia používateľom.

c) Sieťové zariadenia - sú iné ako znakové a blokové zariadenia, lebo nepoužívajú abstrakciu súboru. Používateľ nemôže priamo prenášať dáta na sieťové zariadenia. Tie komunikujú nepriamo, cez špeciálne rozhranie a špeciálne systémové volania.

Existuje veľa rôznych druhov periférnych zariadení a zberníc. Naša reálna sústava však komunikuje cez rozhranie PCI, preto v ďalšej podkapitole popíšeme toto rozhranie.

1.2.1 Rozhranie PCI

PCI architektúra bola vytvorená ako náhrada za ISA štandard, s tromi hlavnými požiadavkami:

- zvýšenie výkonu pri prenose dát medzi počítačom a jeho periférami
- čo najväčšia nezávislosť od platformy
- zjednodušenie pridávania a odoberania periférnych zariadení v systéme

V literatúre [2] je uvedené: „Najdôležitejšie pre vývoj ovládačov je automatická detekcia PCI zariadenia. PCI zariadenia nepoužívajú žiadne prepojky (narozdiel od starších zariadení). Každá základná doska s PCI je vybavená PCI firmvérom, zvaným BIOS, NV-RAM, PROM. Tento firmvér ponúka prístup k adresnému priestoru zariadenia, a to čítaním a zapisovaním registrov cez radič PCI. Počas bootovania systému firmvér vykoná konfiguráciu každého PCI zariadenia, aby bezpečne alokoval jeho pamäťový priestor. V čase keď ovládač zariadenia pristupuje k zariadeniu, jeho pamäť aj I/O oblasti sú už namapované do adresového priestoru procesora.

Každé PCI zariadenie je identifikované číslom zbernice, číslom zariadenia a číslom funkcie. PCI špecifikácia dovoľuje jednému systému mať až 256 zberníc, ale keďže 256 zberníc nie je dostatočný počet pre obrovské systémy, Linux podporuje PCI domény. Každá PCI doména môže mať až 256 zberníc, každá zbernica môže mať až 32 zariadení a každé zariadenie môže mať 8 funkcií.“

1.3 DAQ systémy

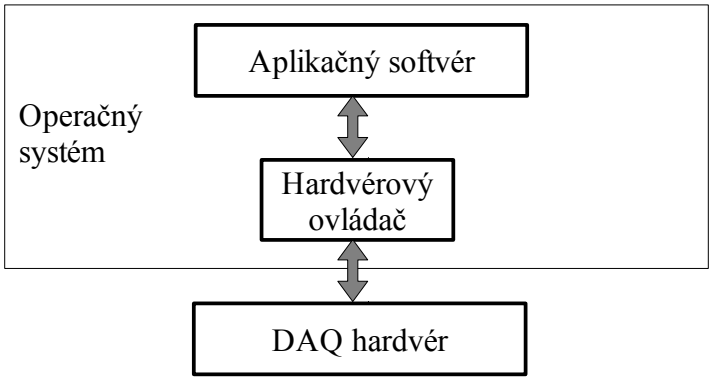
DAQ systémy sú zariadenia, ktoré sa používajú na meranie veličín z prostredia, ktoré ich obklopuje. Väčšina DAQ systémov získava dáta z rôznych druhov snímačov, ktoré produkujú analógové signály. Pre spracovanie týchto signálov je potrebné ich skonvertovať do digitálnej podoby. Na konvertovanie z analógových na digitálne signály sa používajú

A/D prevodníky. DAQ zariadenia môžu pracovať s jednou alebo s viacerými nezávislými veličinami - kanálmi.

Podľa [3] sa DAQ systémy dajú rozdeliť na tri hlavné kategórie:

- počítačové
- embedded
- FPGA (Field Programmable Gate Array)

Systémy založené na počítačoch využívajú výkon počítačov na spracovanie dát, vizualizáciu, ukladanie, simuláciu. Pre komunikáciu s DAQ zariadením je potrebný softvérový ovládač pre OS a pre simuláciu a vizualizáciu je potrebný aplikačný softvér. Ovládač tvorí strednú vrstvu medzi hardvérom a aplikačným softvérom. Toto prepojenie znázorňuje obrázok č. 1.



Obrázok 1: Komunikácia hardvéru a softvéru

Zariadenia v tejto kategórii sa dajú ešte rozdeliť na interné a externé. Interné zariadenia sú rozširujúce karty do PC (PCI, ISA, PCI Express, PCMCIA). Tieto zariadenia, sú závislé na hardvérovom vybavení počítača. Externé DAQ zariadenia sú moduly, ktoré sa pripájajú do počítačových portov (USB, paralelné, sériové, sieťové). Nevýhody počítačových DAQ systémov sú vysoká cena, veľké rozmery, veľká spotreba energie a závislosť na konkrétnom hardvéry.

Embedded systémy sú kompaktné zariadenia riadené mikroprocesorom. Používajú sa v zdravotníctve, automobilovom priemysle,pri monitorovaní životného prostredia. Ich výhodou je, že sú malé, prenosné a výkonné. Nevýhodu je, že ich hardvér je pevne daný, nedá sa jednoducho vymeniť, pri zmene požiadaviek je potrebné vymeniť celé zariadenie.

FPGA systémy sú integrované obvody navrhnuté tak, aby sa po výrobe dali konfigurovať a programovať zákazníkom. Moderné FPGA systémy dovoľujú integrovanie niekoľ-

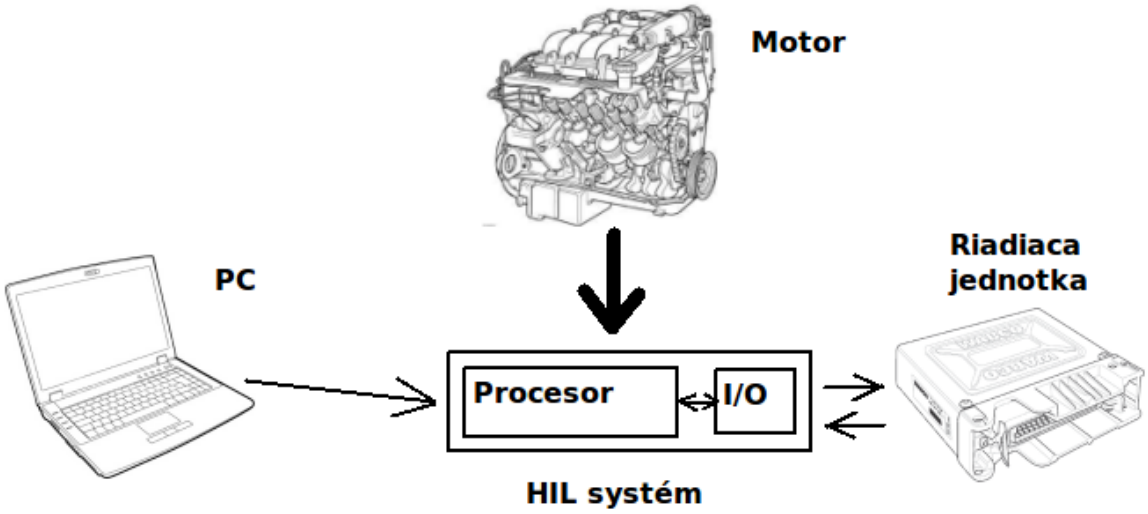
kých komponentov na jednom čipe. Takže jeden čip je schopný spracovávať signály, ukladať dáta a riadiť vstupno-výstupnú činnosť.

2 Analýza problému

V tejto kapitole sú popísané rôzne spôsoby zberu dát a pristupovania k DAQ hardvéru, ako aj konkrétne slobodné aplikácie na zber dát.

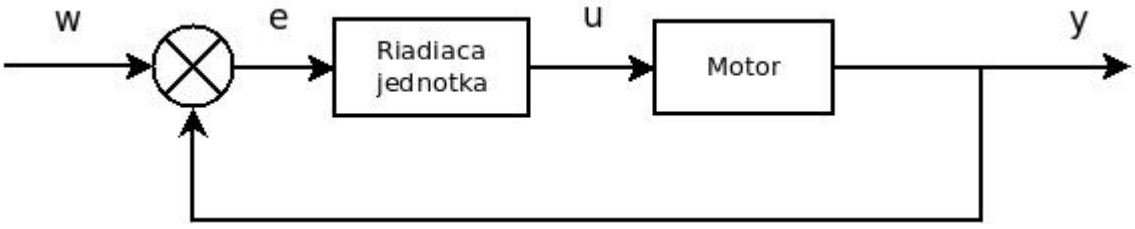
2.1 Hardware in the loop

Hardware in the loop je formou simulácie v reálnom čase. Pre HIL je charakteristické pridanie skutočného komponentu do slučky. O tzv. Hardware-in-the-Loop simulácii hovoríme vtedy, ak je hardware pre simuláciu v reálnom čase vybavený DAQ kartami, pomocou ktorých je spojený s reálnym zariadením. Príklad HIL systému z automobilového priemyslu je znázornený na obrázku č. 2. Motor je nahradený simuláciou, ale jeho riadiaca jednotka je skutočná. HIL systém simuluje riadiacej jednotke I/O signály, ktorá pracuje akoby komunikovala so skutočným motorom.



Obrázok 2: Príklad Hardware in the loop systému

Na HIL je potrebná spätná väzba, ktorá je zapojená do slučky. Princíp slučky je znázornený na blokovom obrázku č. 3.



Obrázok 3: Bloková schéma HIL

2.2 Comedi projekt

Comedi projekt je slobodný softvér, ktorý obsahuje ovládače, nástroje a knižnice pre zber údajov v OS Linux. Na stránke [4] je uvedené: „Zdrojový kód projektu je distribuovaný v dvoch balíkoch, comedi a comedilib. Comedi je súbor ovládačov pre bežné DAQ zariadenia. Ovládače sú realizované ako kombinácia jedného linuxového modulu jadra (zvaného „comedi“) a nízkoúrovňového modulu jadra pre dané zariadenie. Comedilib je samostatne distribuovaný balík obsahujúci užívateľskú knižnicu, ktorá poskytuje vývojárom jednoduché rozhranie k DAQ zariadeniam. Kcomedilib poskytuje rovnaké rozhranie ako comedilib, je to však modul jadra, tento modul je vhodný pre úlohy v reálnom čase.“

2.3 RTAI

Podľa [11] je RTAI rozšírenie Linuxového jadra o real time rozhranie. Skratka RTAI je z anglického *„ RealTime Application Interface“*. RTAI bolo vytvorené ako prostredie pre lacnú implementáciu DAQ systémov. Tento softvér je distribuovaný pod GNU/GPL licenciou pre časť jadra a pod LGPL pre užívateľskú časť. RTAI pozostáva z dvoch hlavných častí:

- patch linuxového jadra, ktorý zavádza hardvérovú abstraktnú vrstvu
- široká škála služieb, ktoré uľahčujú prácu programátorov v reálnom čase

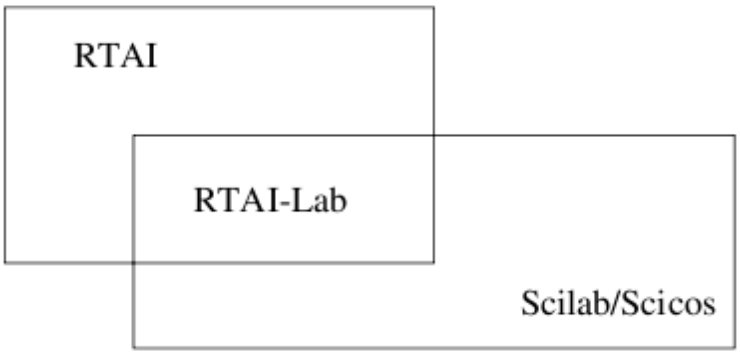
V súčasnosti rôzne výskumné, ale aj komerčné projekty využívajú prostredie RTAI [7][8].

Jednou z najdôležitejších funkcií RTAI je schopnosť používať kód v reálnom čase aj v priestore jadra aj v používateľskom priestore, použitím jedného RTAI plánovača.

2.3.1 RTAI-Lab

Podľa [11] poskytuje projekt RTAI-Lab balík nástrojov na vývoj blokových diagramov, ktoré môžu byť skompilované a spustené na RTAI Linux operačnom systéme. RTAI-Lab je súčasťou RTAI distribúcie. Blokové diagramy môžu byť vytvorené buď v slobodnom softvéry – Scilab/Scicos/Xcos alebo v komerčnom softvéry – Matlab/Simulink/RealTime-Workshop.

Spojenie RTAI a Scilabu/Scicosu znázorňuje obrázok č. 4



Obrázok 4: Spojenie RTAI so Scilabom/Scicosom

Hlavné funkcie RTAI-Lab:

- pridanie RTAI-Lib palety s RTAI blokmi do Scicos-u, ktoré slúžia na vývoj blokových diagramov
- umožňuje hosťovskému a cieľovému počítaču komunikovať cez sieťový protokol NET-RPC
- monitorovacia aplikácia s virtuálnym osciloskopom – *xrtailab*, ktorá umožňuje pracovať s úlohami v reálnom čase
- automatické generovanie real-time kódu zo Scilabu/Scicosu
- možnosť importovať projekty z Matlabu/Simulinku/RealTime-Workshopu
- poskytuje rozhranie k DAQ zariadeniam, ktoré sú podporované v Comedi

2.4 Scilab,Scicos/Xcos

Táto podkapitola bola spracovaná na základe infomácií z [10]. Scilab je vedecký softvérový balík pre numerické výpočty, spracovanie signálov, štatistické analýzy, modelovanie a simulácie. Je to slobodný softvér s GPL kompatibilnou licenciou. Scilab je dostupný nie len pre Linux, ale aj pre MS Windows a Mac OSX. Poskytuje výkonné výpočtové prostredie pre inžinierske a vedecké aplikácie. Scilab má kompletnú sadu nástrojov pre modelovanie, simuláciu a tvorbu designu hybridných riadiacich systémov. Od roku 1994 bol voľne distribuovaný cez internet spolu so zdrojovým kódom. V súčasnosti sa používa vo vzdelávacích a priemyselných odvetviach po celom svete. Scilab obsahuje stovky matematických funkcií s možnosťou pridať interaktívne programy v rôznych jazykoch (Fortran, C, C+ +, Java). Scilab obsahuje sofistikované dátové štruktúry (vrátane zoznamov, polynó-

mov, racionálnych funkcií, lineárnych systémov) a programovací jazyk na vysokej úrovni - Scilab Language, ktorého syntax je založená na syntaxe jazyka MATLAB-u. Scilab obsahoval do verzie 5.2 Scicos, balík pre modelovanie a simuláciu spojitých aj diskrétnych dynamických systémov. Scilab od verzie 5.2 obsahuje balík s názvom Xcos (založený na Scicos-e).

2.4.1 Scicos

Scicos je aplikácia na grafické modelovanie a simulácie dynamických systémov. Scicos sa používa pre aplikácie v riadiacich systémoch, spracovaní signálov, systémoch hromadnej obsluhy, a za účelom výskumu fyzikálnych a biologických systémov. V grafickom editore Scicos-u je možné umiestňovať, konfigurovať a spájať bloky, a tým vytvárať blokové diagramy na modelovanie hybridných dynamických systémov. Taktiež je možné simulovať tieto namodelované systémy. Väčšina grafického používateľského rozhrania Scicos-u je napísaná v jazyku Scilab-u, pre kompletnú integráciu so Scilab-om, kompatibilitu a maximálnu flexibilitu.

Niektoré hlavné funkcie Scicosu:

- graficky modelovať a simulovať dynamické systémy
- kombinovať spojité a diskrétne správanie v jednom modeli
- programovať nové bloky v C, Fotrane alebo v jazyku Scilabu
- spúšťanie simulácií v dávkovom režime z prostredia Scilabu
- generovanie C kódu zo Scicos modelu použitím generátora kódu
- spúšťanie simulácií v reálnom čase
- použitie blokov vytvorených v Modelice
- pridávanie rôznych toolboxov

2.4.2 Xcos

Xcos 1.0 je založený na Scicose 4.2 a v Scilabe od verzie 5.2 nahrádza Scicos. Prináša nové ergonomické vlastnosti a prepojenie so Scilabom zabezpečované Scilab konzorciom. Xcos ponúka stabilné a efektívne riešenie pre priemyselné a akademické potreby. Napríklad, poskytuje funkcie pre modelovanie mechanických systémov (automobilový priemysel, letectvo, ...), hydraulické okruhy (priehrady, modelovanie potrubí), riadiace sys-

témy, atď. Xcos je úplne kompatibilný so Scicos-om a ponúka nielen jeho funkcionalitu, ale aj ďalšie možnosti. Medzi ďalšie možnosti patria:

- úplne nové grafické užívateľské rozhranie založené na JgraphX
- nový štandard výmeny dát medzi Scilabom a Xcos editorom
- slobodný kompilátor Modelici, ktorý umožňuje simuláciu jej diagramov
- Zlepšenie softvérových funkcií a výkonu, ako sú reorganizácia palety, obohatenie dokumentácie

2.5 Komerčný softvér

Do komerčného softvéru na zber údajov, simulácie a modelovanie patria:

- Labview
- Matlab/Simulink
- Real time toolbox pre Matlab
- Real-Time Workshop

Týmto softvérom sa my však zaoberať nebudeme, pretože našou úlohou je používať FOSS.

3 Popis sústavy

V predošlej kapitole sme všeobecne analyzovali aké sú možnosti komunikácie s DAQ hardvérom. V tejto kapitole sa budeme venovať konkrétnemu hardvéru,ktorý sme mali k dispozícii a opisu konkrétneho riešenia komunikačného modulu pre daný hardvér.

K dispozícii sme mali meraciu PCI kartu od firmy Humusoft, konkrétne model Humusoft MF624. Ďalej sme mali k dispozícii sústavu magnetickej levitácie, čo je učebná pomôcka takisto od firmy Humusoft.

3.1 Meracia karta MF624

Niektoré pre nás dôležité vlastnosti udávané výrobcom [6] sú zhrnuté v tabuľke č. 1. Podrobnejší prehľad parametrov je na stránkach výrobcu [6].

Analógové vstupy	8 kanálov, vstupný rozsah +-10V, doba prevodu 10 μs
Analógové výstupy	8 kanálov, výstupný rozsah +-10V, doba ustálenia 31 μs
Digitálne vstupy	8, s úrovňami TTL
Digitálne výstupy	8, s úrovňami TTL
Časovač	4 kanály, rozlíšenie 32 bitov, 20ns

Tabuľka 1: Vlastnosti MF624

Výrobca dodáva k tejto karte tieto ovládače:

- Ovládač pre Real Time Toolbox pre MATLAB
- Ovládač pre Real-Time Windows Target
- Ovládač pre xPC Target

Podporované sú tieto operačné systémy:

- MS Windows 2000, XP, Server 2003, Vista, Server 2008, MS Windows 7
- MS Windows XP 64-bit, Server 2003 64-bit, Vista 64-bit, Server 2008 64-bit, MS Windows 7 64-bit

Ovládače pre OS Linux ani pre iný slobodný OS výrobca neponúka.

3.2 Sústava magnetickej levitácie

Použitý model magnetickej levitácie bol Humusoft CE 152. Sústava magnetickej levitácie je zhotovená z medenej cievky umiestnenej vertikálne nad kovovou guličkou, vertikálna pozícia guličky je snímaná indukčným senzorom.

Vstupný rozsah je 0 až +5V, výstupný rozsah senzora je takisto 0 až +5V. Na komunikáciu s kartou stačí použiť 2 analógové kanály, 1 vstupný a 1 výstupný.

4 Návrh riešenia

Našou úlohou bolo implementovať komunikačný modul pre túto kartu. Možností ako vytvoriť komunikačný modul je viacero, mohol by to byť modul jadra, comedi driver, dynamická knižnica alebo modul do existujúceho softvéru. Prvotným plánom bolo upraviť ovládač určený pre MS Windows, aby fungoval v OS Linux. Teda vytvoriť nízko-úrovňovú knižnicu, ktorá by priamo pristupovala k hardvéru.

Avšak prehľadávaním aj iných než oficiálnych zdrojov sme našli hotový ovládač pre Linux, tzv. hudaqlib. Ovládač je napísaný v jazyku C. Porovnaním hlavičkového súboru, tohto ovládača s ovládačom, ktorý je určený pre MS Windows sme zistili, že Linuxový ovládač má rovnaké funkcie ako ten pre MS Windows. Rozhodli sme sa využiť tento ovládač a pomocou jeho funkcií vytvoriť komunikačný modul do iného softvéru na zber údajov. Ako aplikačný softvér sme si vybrali Scilab a Scicos (neskôr Xcos). V Scicose sme plánovali vytvoriť vlastný modul, ktorý by mal formu vlastného bloku a používal by funkcie z hudaqlib a s týmto blokom sme plánovali namodelovať simuláciu pre reálny systém.

4.1 Hudaqlib

Hudaqlib je ovládač a knižnica vytvorená Jaroslavom Fojtikom pre zariadenia od firmy Humusoft. Jaroslav Fojtik síce pracoval pre Humusoft, ale tento ovládač nie je produktom Humusoftu a nemá od nich ani podporu. K ovládaču sa dá pristupovať pomocou dynamickej knižnice napísanej v jazyku C. Ovládač je určený pre 32 bitové aj 64 bitové OS. K dispozícii sú všetky zdrojové kódy, makefile aj príklady použitia a kompilácie. Kompiláciou zdrojových kódov sa vytvorí dynamická knižnica, súbor s príponou .so (v Linuxe po anglicky označovaná ako „shared object“). Pre použitie funkcií z dynamickej knižnice je potrebné vkladať do svojich programov hlavičkový súbor hudaqlib.h a pri kompilácií zadať cestu k hudaqlib.so. Súbor hudaqlib.h obsahuje prototypy funkcií, ktoré sa nachádzajú v hudaqlib.so.

Pri programovaní s použitím knižnice je každé zariadenie reprezentované svojim deskriptorom**,** takže pred pristupovaním k zariadeniu je potrebné najprv otvoriť jeho deskriptor. Každé zariadenie má niekoľko subsystémov, analógové/digitálne, vstupy/výstupy. Pre jednotlivé subsystémy sa používajú rôzne funkcie. Každý subsystém má niekoľko kanálov, ku kanálom sa pristupuje pomocou čísel od 0 do 7 (pre 8 kanálov). Pred ukon-

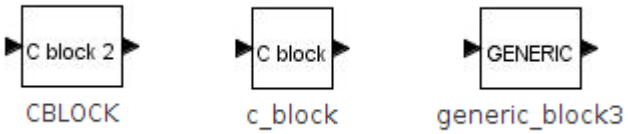
čením programu je potrebné ukončiť komunikáciu so zariadením. Podrobnejší popis knižnice v manuále [13].

4.2 Možnosti vytvorenia modulu v Xcose

Scicos/Xcos ponúka niekoľko blokov na používateľom definované funkcie, medzi inými aj blok pre fortran, Modelicu a C bloky. Pre nás sú zaujímavé bloky, ktoré vedia pracovať aj s jazykom C. V Scilabe sa používajú na prácu s jazykom C tieto bloky:

- c_block (C block)
- CBLOCK (C block 2)
- generic_block3 (GENERIC)

Ikony blokov v palete Xcosu sú znázornené na obrázku č. 5:



Obrázok 5: Bloky podporujúce jazyk C

Používateľom definované funkcie sa skladajú z dvoch častí. Z funkcie rozhrania (interfacing function) a výpočtovej funkcie (computation function). Funkcia rozhrania je napísaná v jazyku Scilabu a slúži na definovanie vzhľadu a správania sa grafických elementov bloku v editore. Všetky tri bloky majú preddefinovanú funkciu rozhrania, takže my sme ju nemuseli vytvárať. Výpočtová funkcia definuje samotnú činnosť bloku, v našom prípade bude vracať namerané hodnoty z DAQ karty a zapisovať žiadané hodnoty na DAQ kartu.

4.2.1 Popis blokov využívajúcich jazyk C

c_block (C block) – tento blok vytvára kostru výpočtovej funkcie a taktiež vytvára dynamickú knižnicu a objektový súbor. Zdrojový kód funkcie je dostupný priamo z Xcosu, po otvorení vlastností bloku. Po každej zmene v tomto kóde sa tento automaticky skompiluje a zlinkuje. Zistili sme, že zdrojový kód sa nedá editovať mimo Scilabu, lebo Scilab vždy tento súbor prepíše. Makefile je generovaný automaticky a pri jeho ručnej úprave sa taktiež vždy prepíše. Nastaviť parametre kompilácie a linkovania, ani upraviť makefile, nie

je jednoduché, lebo v dostupnej dokumentácii sa to nenachádza, kvôli tomu sme nemohli výpočtovú funkciu zlinkovať s hudaqlib ovládačom. Preto sme si vybrali iný blok, kde je vlastná kompilácia jednoduchšia.

Funkcia rozhrania: *SCI/modules/scicos_blocks/macros/Misc/c_block.sci*

CBLOCK (C block 2) – tiež vytvára výpočtovú funkciu, dynamickú knižnicu a objektový súbor. Má oveľa viac parametrov ako c_block, inú funkciu rozhrania a prepracovanejšiu kostru výpočtovej funkcie. Ale editovanie, kompilácia a linkovanie funguje rovnako ako v c_block-u. Tento blok je takisto obtiažne zlinkovať s hudaqlib knižnicou. Preto sme si vybrali iný blok. Oba bloky sú dobre použiteľné na funkcie, ktoré nevyužívajú externé dynamické knižnice. Ak by bolo možné editovať alebo generovať vlastný makefile sú použiteľné aj s externými knižnicami.

Funkcia rozhrania: *SCI/modules/scicos_blocks/macros/Misc/CBLOCK.sci*

generic_block3 (GENERIC) – vytvára všeobecnú funkciu rozhrania, ale výpočtová funkcia musí byť definovaná oddelene. Výpočtová funkcia môže byť ako Scilab funkcia, Fortran funkcia alebo C funkcia. Možnosť mať výpočtovú funkciu v externom súbore a možnosť vlastnej kompilácie boli pre nás výhodou. Na našu úlohu postačoval tento všeobecný blok. Parametre tohto bloku popíšem podrobnejšie neskôr, lebo z týchto troch blokov sme používali tento.

Funkcia rozhrania: *SCI/modules/scicos_blocks/macros/Misc/generic_block3.sci*

5 Implementácia komunikačného modulu

Rozhodli sme sa implementovať modul do Scilabu, konkrétne do programu, ktorý je jeho súčasťou – Xcosu. Keď sme začínali s prostredím Scilabu, ten bol vtedy vo verzii 5.1.1 a nebol tam ešte Xcos, ale Scicos 4.2. Používali sme OS Ubuntu 9.04, grafické prostredie Gnome 2.26 bez kompozitného manažéra. Na tejto platforme bolo prostredie Scilabu a Scicosu veľmi nestabilné, Scicos mal problémy s prekresľovaním obsahu okien, veľmi často padal, paleta nástrojov bola neprehľadná, celé prostredie malo pomalú odozvu a neintuitívne ovládanie. Práca s takýmto prostredím bola obtiažna a zdĺhavá. S novou verziou Scilabu 5.2 sa namiesto Scicosu 4.2 objavil Xcos 1.0, ktorý vychádza so Scicosu. Xcos je stabilnejší, má prepracovanejšie GUI, lepšiu paletu nástrojov a rýchlejšiu odozvu. Má však tiež problémy s prekresľovaním obsahu okien. Inak je veľmi podobný Scicosu a nie je v ňom problém otvoriť schému zo Scicosu. Svoj modul som testoval konkrétne so Scilabom 5.2.1. a Xcos-om 1.0, ale rovnakým spôsobom by sa mal dať takýto modul vytvoriť aj v Scicose 4.2.

Z troch používateľom definovateľných blokov sme použili GENERIC blok. Do výpočtovej funkcie bloku bolo treba vložiť funkcie na komunikáciu s MF624 z hudaqlib. Ďalej skompilovať a nalinkovať výpočtovú funkciu do Scilabu spolu s knižnicou libhudaqlib.so. Potom sme v Xcose modelovali diagram pre ovládanie reálnej sústavy a po úspešnej realizácii magnetickej levitácie sme testovali vlastnosti nášho riešenia.

5.1 GENERIC blok

Základné použité parametre bloku,ako meno, typ,vstupy/výstupy funkcie:
Všetky parametre je možné nájsť v dokumentácii [5].
Simulation function – názov výpočtovej funkcie, funkcia musí byť pred simuláciou nalinkovaná do Scilabu, my sme použili názov *readwrite*
Function type – typ funkcie, C block je číslo 4
Input port sizes – rozmery vstupných portov, parametrom sú rozmery vstupných matíc. Pri práci s viacerými napríklad tromi vstupnými kanálmi nie je dobré použiť maticu [1,3] alebo [3,1]. Lepšie je použiť 3 samostatné matice [1,1; 1,1; 1,1]. Zo samostatnou maticou pre každý vstupný kanál sa pracuje jednoduchšie v iných blokoch, ktoré spracúvajú tieto hodnoty. Nie každý blok vie pracovať s viacrozmernou maticou, preto by bolo potrebné

použiť multiplexory a demultiplexory. My sme používali iba jeden kanál, takže matica bola definovaná [1,1], všetkých 8 kanálov by sa definovalo takto: „[1,1; 1,1; 1,1; 1,1; 1,1; 1,1; 1,1; 1,1]“

Input ports type – dátové typy vstupných portov, pre každý vstupný port treba definovať dátový typ samostatne, my sme používali reálny typ = 1, čo zodpovedá typu *double* v jazyku C. Keby sme použili 8 kanálov tak tento parameter bude vyzerať takto: “1 1 1 1 1 1 1 1 ”

Output port sizes – rozmery výstupných portov, platí to isté ako pri vstupných portoch. Naša matica bola [1,1].

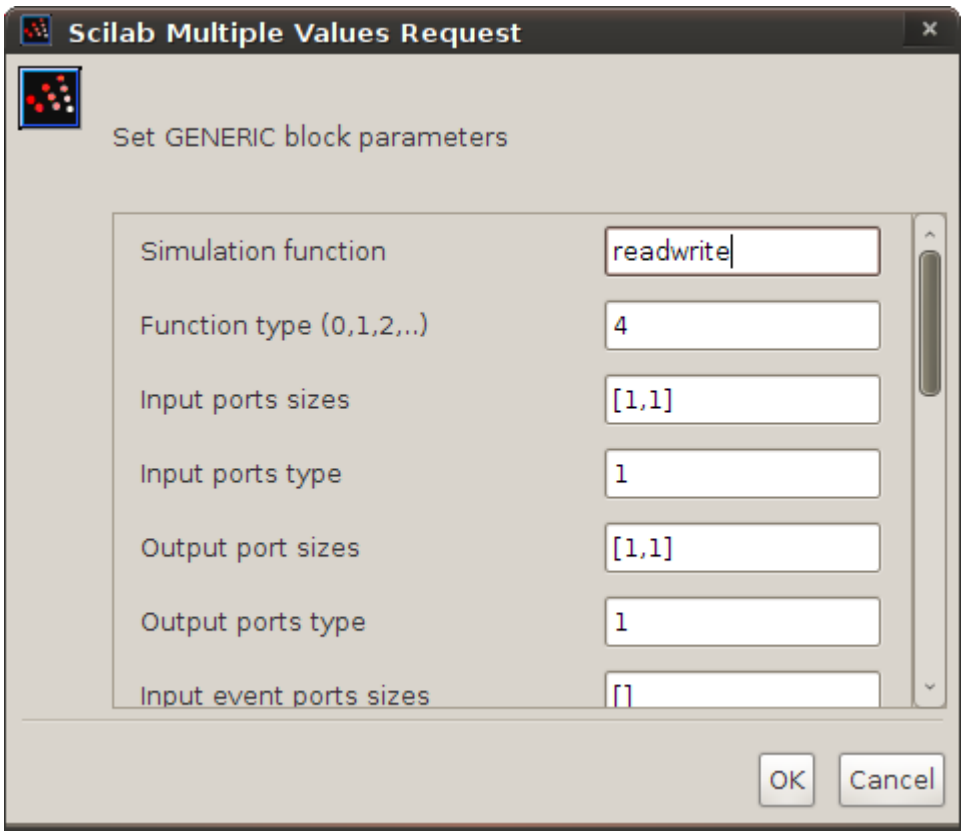
Output ports type – dátové typy výstupných portov, platí to isté ako pri vstupných portoch. Typ Double je 1.

Input event ports sizes - rozmery vstupných portov pre udalosti, hlavne rôzne časovače,ktoré sú v Xcos editore označované červenou farbou. Event porty sme nepotrebovali, pretože sme použili jeden časovač pre celý blokový diagram.

Output events ports sizes - rozmery výstupných portov pre udalosti. Tiež sme nepotrebovali.

Time dependence – špecifikuje či výstup bloku je časovo závislý, môže byť buď závislý „y“ alebo nezávislý „n“. Dôležitý parameter, lebo keď je nastavený na „n“ tak výpočtová funkcia sa zavolá iba raz pre celý beh simulácie. Pre opakované volanie funkcie podľa vzorkovacej frekvencie (časovača) je nutné nastaviť parameter na „y“.

Okno s parametrami bloku je znázornené na obrázku č. 6.



Obrázok 6: Parametre GENERIC bloku

5.2 Štruktúra Scicos_block

Táto štruktúra je vstupným parametrom výpočtovej funkcie generic bloku aj C bloku 2. Štruktúra obsahuje všetky potrebné parametre na prístup k bloku, vstupy, výstupy, parametre, registre, stavy, módy. Táto Štruktúra je definovaná v súbore *INSTALL_DIR/include/scilab/scicos_block4.h* a tento hlavičkový súbor musí byť použitý v každej výpočtovej funkcii: *#include <scicos_block.h>*

Dátové polia sú prístupné cez smerník *block, spôsobom *block->pole.*

Stručný popis parametrov, ktoré som používal, viac v dokumentácii [5]

block->nin – integer, ktorý udáva počet vstupných portov bloku

block->inptr – pole smerníkov, ktoré umožňuje priamy prístup k dátam vo vstupných maticiach

block->nout – integer, ktorý udáva počet výstupných portov bloku

block->outptr – pole smerníkov, ktoré umožňuje priame zapisovanie dát vo výstupných maticiach

5.3 Použitie Hudaqlib v GENERIC bloku

Používal som hudaqlib verzie 2.3.2. Prototyp výpočtovej funkcie vyzerá takto: *void readwrite(scicos_block *block, int flag)*. Prvým parametrom je štruktúra *scicos_block,* tá bola popísaná vyššie. Druhým parametrom je *flag*, tento parameter nastavuje Xcos podľa toho kedy v simulácii je volaná funkcia. Ak je funkcia volaná prvý krát tak flag==4, vtedy sa vykonávajú inicializačné operácie. Pri poslednom volaní funkcie flag==5, vtedy sa vykonávajú ukončovacie operácie. Pri ostatných volaniach funkcie flag==1, vtedy sa vykonáva samotné čítanie/zapisovanie hodnôt zo/do sústavy.

Pre použitie hudaqlib ovládača treba vložiť hlavičkový súbor *hudaqlib.h.* Potom je potrebné zavolať deskriptor pre kartu takto: *HUDAQHANDLE h;* a otvoriť komunikáciu s MF624 zariadením takto: *h = HudaqOpenDevice("MF624", 1, 0);* Prvý parameter je názov zariadenia, druhý parameter je poradie zariadenia (pre prípad ak by sme mali viac rovnakých zariadení v počítači), tretí parameter je vždy 0. Teraz môžeme používať premennú *h* ako deskriptor k nášmu zariadeniu. Na konci programu je zase potrebné zavrieť komunikáciu so zariadením: *HudaqCloseDevice(h);*

Keď máme takto inicializované zariadenie, môžeme už volať funkcie na čítanie alebo zapisovanie hodnôt. Napríklad takto: *block->outptr[6][0] = HudaqAIRead(h,6);*. Kde „h“ je deskriptor na našu kartu a „6“ číslo kanálu. Týmto sa z karty prečíta v poradí siedmi analógový kanál a hodnota sa priradí do (v poradí siedmej) výstupnej matice (za predpokladu, že používame rozmery matice [1,1]). Príklad zapísania hodnôt na kartu: *HudaqAOWrite(h, 6, block->inptr[6][0]);* , týmto sa zapíše hodnota siedmeho vstupu do GENERIC bloku na siedmi analógový vstup karty.

Podobne sa pracuje aj s digitálnymi vstupmi/výstupmi. Napr. *block->outptr[1][0] = HudaqDIRead(h,1);* prečíta digitálny výstup ako celok (8 bitový výstup vráti ako celé číslo) a zase priradí na výstup bloku. Je možné čítať aj jednotlivé bity digitálneho výstupu. Funkcia *HudaqDOWrite(h, 1, (unsigned) block->inptr[1][0]);* zapíše na druhý digitálny vstup karty celé číslo (pre prípad zle nastaveného vstupu do bloku pretypovanie na *unsigned*). Takisto je možné zapisovať bity jednotlivo.

Práca s enkóderom (inkrementálnym snímačom) je zložitejšia, pretože má niekoľko módov, rôzne podmienky spúšťania, rôzne možnosti resetovania a filtrovanie vstupov. My sme použili iba jednoduché čítanie počtu impulzov *HudaqEncRead(h,cislo_kanala)*, pretože primárne sme sa zameriavali na analógové vstupy/výstupy (magnetickú levitáciu). Ak by sa niekto chcel hlbšie venovať enkóderu, tak si môže doprogramovať ďalšie jeho funkcie podľa dokumentácie [13].

V našom riešení riadenia magnetickej levitácie sme použili jeden blok na čítanie aj zapisovanie hodnôt. Pri každom volaní výpočtovej funkcie sa načítajú hodnoty zo sústavy a hneď sa aj zapíšu požadované hodnoty na sústavu. Je však možné čítať toľko výstupných kanálov z karty koľko má GENERIC blok definovaných výstupov a takisto sa aj zapisovať toľko vstupných kanálov koľko má blok definovaných vstupov (stačí pridať porty v Xcose editore). Pri našej sústave stačilo čítať jeden vstupný kanál a zapisovať jeden výstupný kanál.

5.4 Kompilácia a linkovanie výpočtovej funkcie

Každú výpočtovú funkciu GENERIC bloku, je potrebné skompilovať a zlinkovať do Scilabu. Existuje niekoľko rôznych funkcií na kompilovanie a linkovanie funkcií.
Zoznam niektorých dostupných funkcií, podrobnejší popis je v manuáli [9]:

- ilib_build – vytvára dynamické knižnice
- ilib_gen_Make – generuje Makefile, pre ilib_build
- ilib_compile – spúšťa Makefile vytvorený pomocou ilib_gen_Make
- ilib_gen_loader – generuje súbor pre načítanie knižnice
- link – dynamické linkovanie
- ilib_for_link – vytvára dynamické knižnice a generuje súbor pre načítanie knižníc

Užitočná je aj funkcia *haveacompiler,* ktorá overuje či je dostupný kompiler jazyka C. Z uvedených funkcií sme používali *ilib_for_link*, jej výhodou je, že dokáže vykonať viac akcií. Nie je potrebné volať zvlášť funkciu na kompilovanie, zvlášť na linkovanie a zvlášť na generovanie súboru na načítanie knižnice. Funkcia skompiluje zdrojový kód, vytvorí dynamickú knižnicu a aj vygeneruje súbor na načítanie knižnice do Scilabu.

Pretože sme používali funkciu *ilib_for_link*, tak jej parametre popíšem podrobnejšie.

ilib_for_link *(name, file, libs, flag [,makename [,loadername [,libname [,ldflags [,cflags [,fflags [,cc]]]]]]])*

name – meno funkcie, ktorú chceme skompilovať a zlinkovať

file – názov objektového súboru, ktorý sa vytvorí zo zdrojového súboru rovnakého mena

libs – extra knižnice potrebné na vytvorenie dynamickej knižnice, zadávané bez *.so*

flag – indikátor, či ide o C alebo Fortran funkciu („c“ alebo „f“)

makename – cesta k Makefile súboru, tento parameter je nepotrebný od Scilabu 5.0

loadername – cesta k súboru na načítanie knižnice (predvolená hodnota je *loader.sce)*

libname – nepovinný parameter, meno dynamickej knižnice, ktorá bude vytvorená

ldflags - nepovinný parameter, slúži na pridanie ďalších linkovacích parametrov do vygenerovaného Makefilu

cflags - nepovinný parameter, slúži na pridanie ďalších kompilovacích parametrov pre C do vygenerovaného Makefilu

fflags - nepovinný parameter, slúži na pridanie ďalších kompilovacích parametrov pre Fortran do vygenerovaného Makefilu

cc - nepovinný parameter, slúži na definovanie kompilátora pre jazyk C

Príklad nášho použitia: *ilib_for_link('write', 'write_block.o' , ["/lib/libhudaqlib"],"c")*

Týmto príkazom sa skompiluje súbor *write_block.c*, vytvorí sa dynamická knižnica *libwrite.so*, súbor na načítanie tejto knižnice do Scilabu *loader.sce* a súbor na odstránenie tejto knižnice zo Scilabu *cleaner.sce*.

Pre načítanie vytvorenej dynamickej knižnice je potrebné ešte spustiť *loader.sce*, a to spustením príkazu: *exec loader.sce*

Na odstránenie dynamickej knižnice by sa zase spustil príkaz *exec cleaner.sce*

5.5 Načítanie knižníc pri štarte Scilabu

Keď už máme v Scilabe načítanú našu knižnicu môžeme s ňou pracovať a až do vypnutia (pádu) Scilabu. Po opätovnom zapnutí Scilabu je zase potrebné ručne načítať našu knižnicu (môžme aj prekompilovať náš kód). Pohodlnejšie by bolo keby sa naša knižnica pri štarte Scilabu načítala automaticky. Môžeme nato použiť súbor *scilab.ini*, tento súbor Scilab vykonáva pri štarte, je to vlastne skript napísaný v jazyku Scilabu. Po inštalácií Scilabu však tento súbor nemusí byť vytvorený. Súbor *scilab.ini* sa musí nachádzať v domácom adresári Scilabu (nie je totožný s inštalačným adresárom). Cesta domáceho adresára

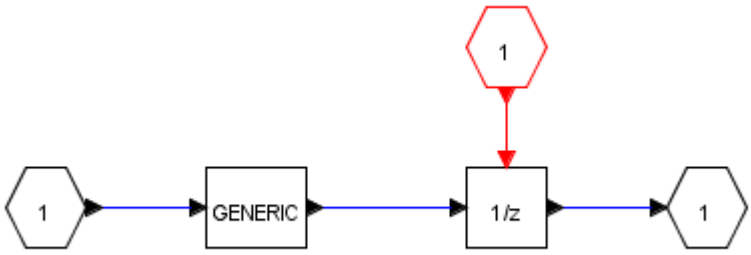
Scilabu je uložená v premennej *SCIHOME* (pre zistenie cesty stačí v Scilabe napísať SCI-HOME). Väčšinou to býva skrytý adresár *.Scilab* v domácom adresáry používateľa, ktorý inštaloval Scilab. Napr. u mňa: */home/david/.Scilab/scilab-5.2.1*

Do *scilab.ini* vložíme volanie nášho súboru na načítanie knižnice *loader.sce*, Takto: *exec '/CESTA/loader.sce'*. Potom sa nám pri každom štarte Scilabu zavolá *loader.sce*, ktorý načíta našu dynamickú knižnicu (v Scilabe budeme vidieť normálny výpis ako keby sme ho volali ručne).

Pre jednoduchšiu manipuláciu so súborom *scilab.ini*, sme vytvorili skript, ktorý pridáva riadok *exec '/CESTA/loader.sce'* do súboru *scilab.ini* (prípadne ak *scilab.ini* neexistuje tak ho vytvorí). Tento skript sa nachádza v prílohe A.

5.6 Pridanie vlastného bloku do palety

Aby sme nemuseli vždy otvárať xcos súbor, keď chceme vytvoriť schému s našim blokom, pridali sme si náš blok do palety. Najprv sme náš GENERIC blok presunuli do superblocku, čo je blok ktorý vlastne združuje rôzne bloky. V superblocku je potrebné vytvoriť vstupné porty (blok *IN_f*), výstupné porty (blok *OUT_f*) a vstup pre časovanie (blok *CLKINV_f*). Superblock ešte obsahuje blok pre oneskorenie signálu (bude vysvetlené neskôr). Zostava nášho superblocku je znázornená na obrázku č. 7.

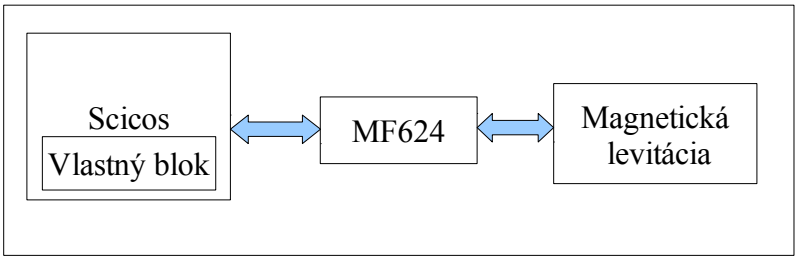


Obrázok 7: Superblock s našim komunikačným blokom

Keď sme mali vytvorený superblock uložili sme si ho na disk ako xcos schému, túto schému sme následne otvorili v palete Xcosu. Týmto sa nám do sekcie *User-defined* v palete pridal vytvorený superblock, ktorý tam ostane aj po reštarte Scilabu (môžeme ho samozrejme ručne odstrániť).

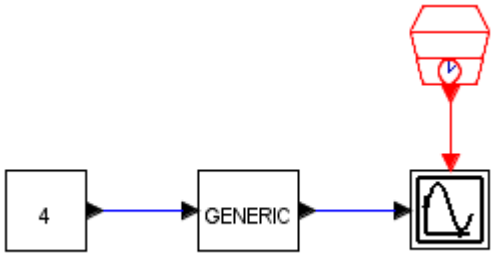
5.7 Testovanie riešenia

Testovanie riešenia je dôležitou časťou implementácie, pretože sme potrebovali vedieť či našim prístupom je možné pracovať v reálnom čase, a ak áno tak sme potrebovali zistiť aké sú limity takéhoto riešenia (čas odobratia). Keď už sme mali vytvorený blok, ktorý dokázal využívať hudaqlib ovládač, mohli sme ho vyskúšať s reálnou sústavou. Bloková schéma zapojenia sústavy je na obrázku č. 8



Obrázok 8: Blokové zapojenie reálnej sústavy

Pre overenie funkčnosti bloku sme museli v Xcose vytvoriť blokový diagram. Pre vyskúšanie základnej komunikácie postačovala jednoduchá schéma ako je na obrázku č. 9:

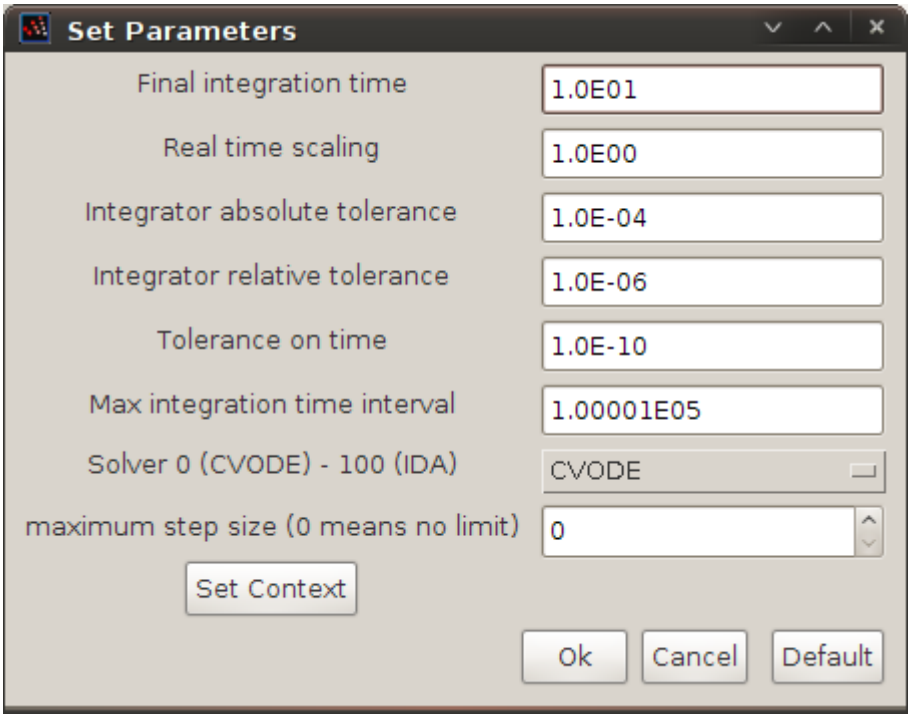


Obrázok 9: Schéma na overenie funkčnosti

Kde blok na ľavej strane je konštantná hodnota. *GENERIC* blok obsahuje našu výpočtovú funkciu na komunikáciu s MF624. Blok na pravej strane je *CSCOPE*, tento blok vykresľuje graf hodnôt v závislosti od času. Blok v hornej časti je časovač *SampleCLK*, tento blok udáva vzorkovaciu frekvenciu (pre tento účel bola použitá hodnota 0.1 to je 100ms).

Pred spustením simulácie bolo potrebné ešte nastaviť parametre simulácie (v menu *simulation -> setup*).

Okno nastavenia parametrov zobrazuje obrázok č. 10:



Obrázok 10: Nastavenie parametrov simulácie

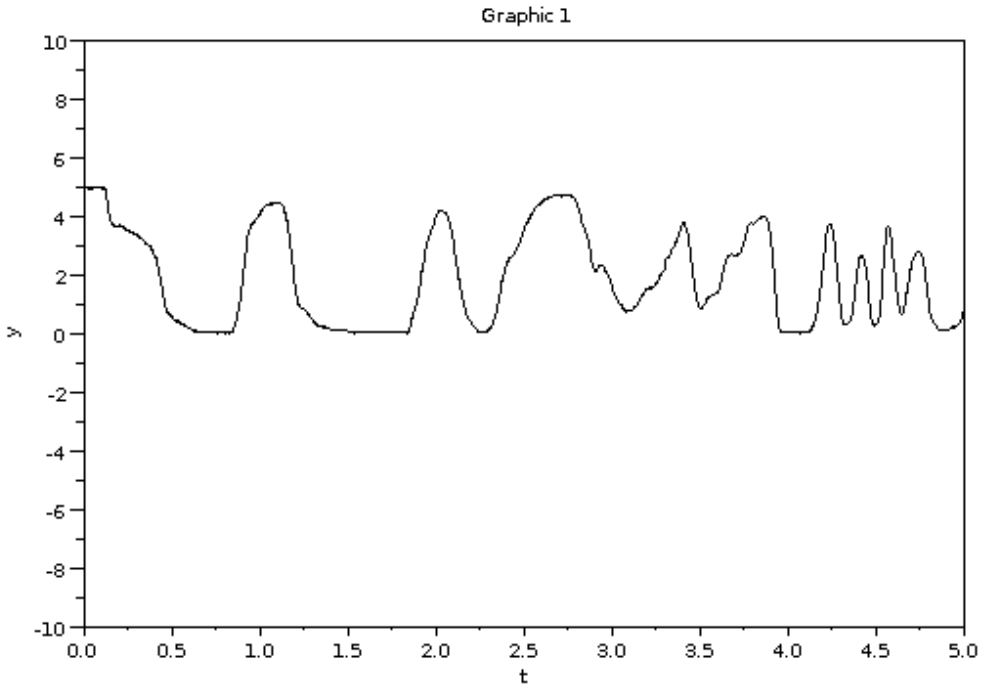
Final integration time – doba trvania simulácie

Real time scaling – definuje relatívnu rýchlosť simulácie v porovnaní so skutočným časom, my sme používali hodnotu 1 (1 sekunda v simulácie je skutočne 1 sekunda).

Ostatné parametre sme ponechali na predvolených hodnotách.

Po spustení simulácie bolo možné ručným pohybovaním guličky voči magnetu overiť či náš blok naozaj meria hodnoty z tejto sústavy. Výsledkom bol graf vykresľovaný v reálnom čase podľa vertikálnej polohy guličky, ktorý je znázornený na obrázku č. 11

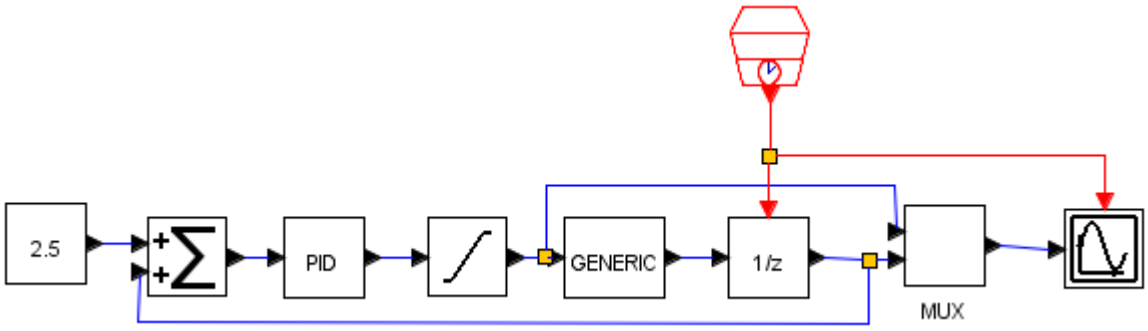
Funkčnosť zapisovania hodnôt bolo možné overiť nastavením vstupnej hodnoty rovnej 5 (teda 5V), pretože vtedy bola gulička úplne pritiahnutá k magnetu. Po úspešnom overení komunikácie so sústavou sme mohli pokračovať v skutočnom riadení magnetickej levitácie a testovaní minimálnej vzorkovacej frekvencie.



Obrázok 11: Graf manuálneho pohybu guličky

5.7.1 Riadenie magnetickej levitácie

Na riadenie magnetickej levitácie sme použili schému podľa obrázku č. 12:



Obrázok 12: Schéma na riadenie magnetickej levitácie

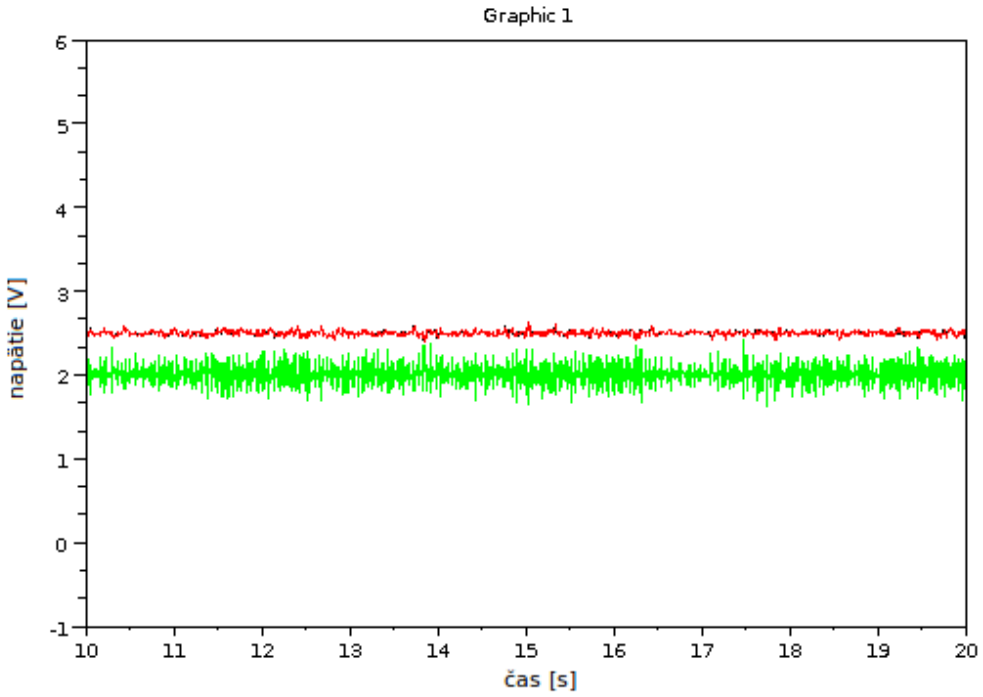
Stručný popis blokov zľava:

CONST_m – konštanta, požadovaná hodnota (výška)

BIGSOM_f – suma, v našom prípade odčítanie skutočnej hodnoty od požadovanej, nastavené: [1;-1]

PID – PID regulátor, nastavené hodnoty sme menili v závislosti od vzorkovacej frekvencie
SATURATION – saturácia, upravuje hornú a dolnú hranicu signálu, nastavené: dolná = 0, horná = 5 (rozsah sústavy magnetickej levitácie vo Voltoch)
generic_block3 – blok s našou funkciou na čítanie/zapisovanie dát z MF624
DOLLAR_f – operátor oneskorenia, výstup musí byť o jednu vzorku oneskorený, aby nevznikla algebraická slučka, nastavenie: *initial condition* = 0, *inherit* = 0
MUX – multiplexor, použité na spojenie vstupný a výstupných hodnôt do jedného grafu
CSCOPE – vykreslenie grafu
SampleCLK – časovanie vzorkovacej frekvencie (na schéme umiestnené hore)

Nastavenie simulácie bolo rovnaké ako v predošlom prípade. Pre jasnejšie zobrazenie nie je použitý superblock. Pre zvýšenie rýchlosti sme zrušili všetky kontrolné výpisy. Testovanie sme začínali s dobou odberu vzorky 0,005 (5ms). Po spustení simulácie sa vyťaženie procesora držalo na 100%, ale aj tak sa nám podarilo vyladiť PID regulátor tak, aby sme dosiahli magnetickú levitáciu. Jeden z grafov priebehu magnetickej levitácie je na obrázku č.13. Vrchná krivka (červená) je čítaná (nameraná) hodnota napätia zo sústavy, spodná krivka (zelená) je zapisovaná hodnota napätia.



Obrázok 13: Priebeh magnetickej levitácie

Otváranie a zatváranie komunikácie v každom cykle	
8 kanálov	1988,2 μs
1 kanál	1993,8 μs
Otvorenie a zatvorenie komunikácie iba raz	
8 kanálov	31,1 μs
1 kanál	3,7 μs

Tabuľka 2: Doba prečítania a zapísania hodnoty (mimo Xcosu)

Vzhľadom k takýmto výsledkom, sme upravili kód nášho bloku. Deskriptor zariadenia sme dali ako globálnu premennú, komunikáciu sme otvorili iba pri inicializačnom volaní výpočtovej funkcie a zatvorili pri poslednom volaní funkcie.

Po týchto úpravách sme magnetickú levitáciu mohli riadiť s dobou odberu vzorky 1ms, s vyťažením procesora 10 - 20 %.

Aby sme zistili presné možnosti simulácie v Xcose doplnili sme do výpočtovej funkcie meranie času prečítania a zapísania hodnoty (napätia) každej jednej vzorky. Tieto hodnoty sme po ukončení simulácie uložili do textového súboru. Tento test sme prevádzali priamo pri reálnom riadení levitácie počas 10 sekundovej simulácie. Výsledky tohto testu sú zhrnuté v nasledujúcej kapitole.

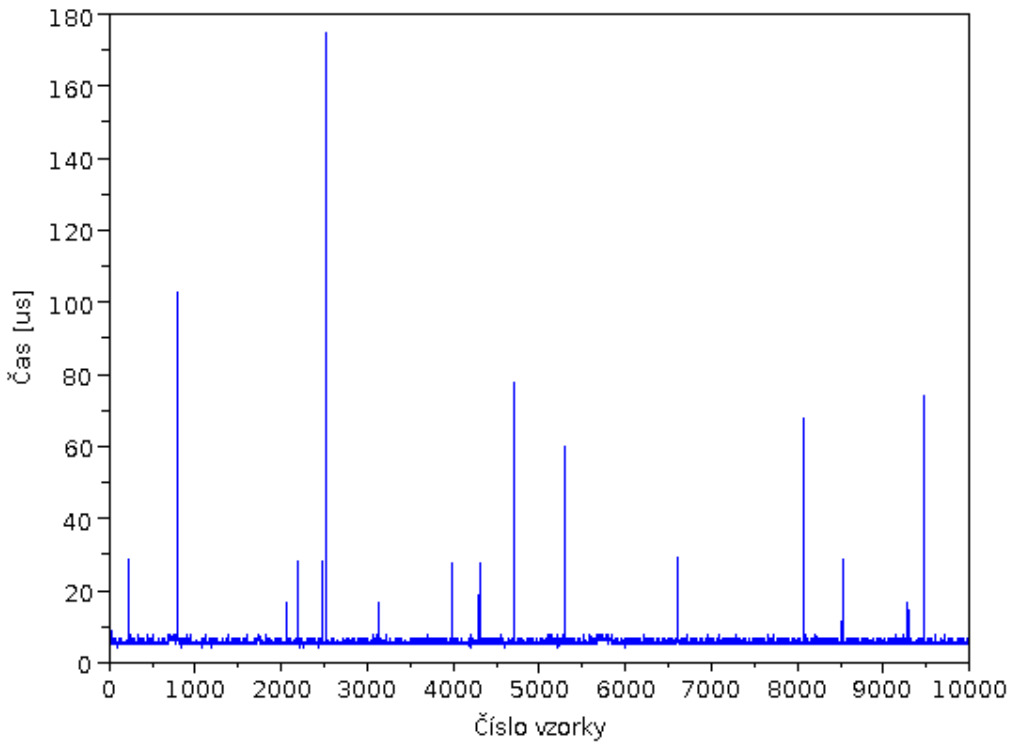
6 Zhodnotenie výsledkov

Výsledky testovacej simulácie s meraním času (prečítania a zapísania hodnoty) sú zhrnuté v tabuľke č. 3

	Hodnota [µs]
Vzorkovacia frekvencia	1000
Priemerná hodnota	5,64
Minimum	4
Maximum	175
	Počet	Podiel v %
Vzoriek	10002	100,00%
Väčších ako 7 µs	83	0,83%
Väčších ako 10 µs	17	0,17%
Väčších ako 100 µs	2	0,02%

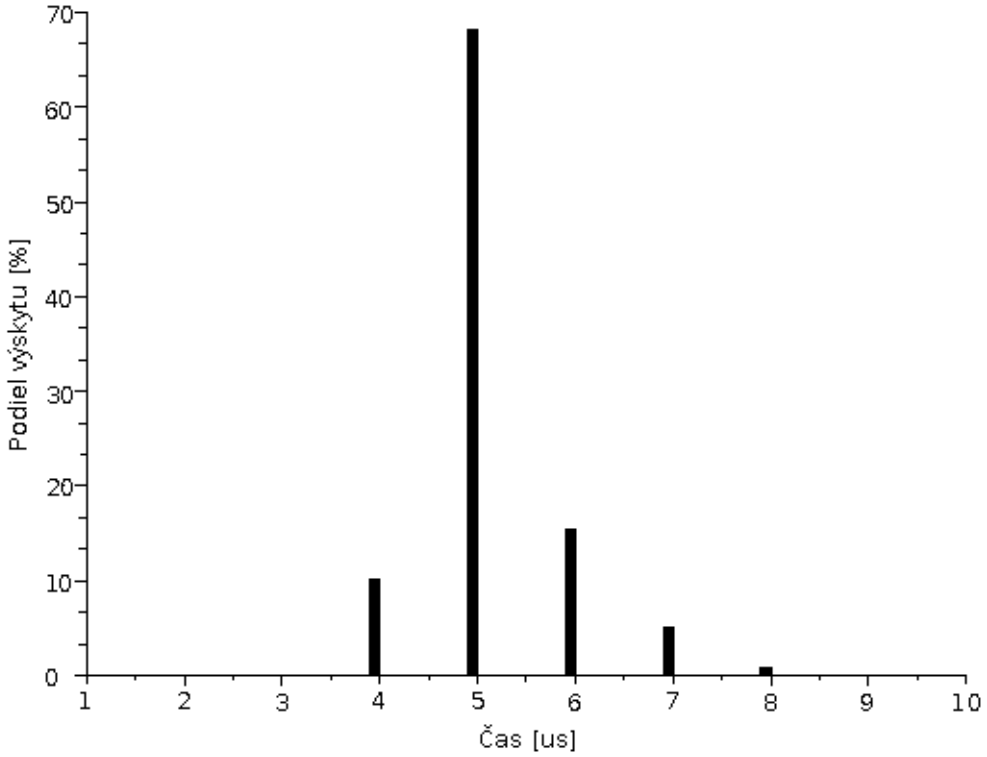
Tabuľka 3: Štatistické výsledky

Výsledky ukazujú že priemerná doba prečítania+zapísania hodnoty je **5,64 µs**, čo je o 1,94 µs viac ako pri meraní mimo Xcosu. Namerané hodnoty sme vložili do Scilabu a vykreslili sme priebeh prečítania+zapísania hodnoty vzhľadom na číslo (poradie) vzorky. Graf je znázornený na obrázku č. 14.



Obrázok 14: Priebeh simulácie

Čas v grafe priebehu simulácie je zobrazený iba do 10 µs, pretože vzoriek ktorých doba čítania+zapisovania bola dlhšia ako 10 µs je iba 0,17%. Pre lepší štatistický prehľad sme vytvorili aj histogram, znázornený na obrázku č. 15. Z histogramu vyplýva, že takmer 70% vzoriek bolo urobených za 5 µs. Doba čítania+zapisovania bola dlhšia ako 8 µs len pri veľmi malom počte vzoriek.



Obrázok 15: Histogram

Z vyhodnotenia výsledkov môžme odhadnúť, že najmenší vzorkovací čas nášho bloku je približne na úrovni 8 µs. S takýmto vzorkovacím časom samozrejme nemôže pracovať celá simulácia magnetickej levitácie, pretože okrem komunikácie so zariadením je v každom cykle potrebné vyhodnotiť získané hodnoty a vypočítať požadovanú hodnotu (PID regulátor). Na našom počítači bolo možné ovládať magnetickú levitáciu s časom odberu vzorky **100 µs** (aby procesor nebol vyťažený na 100%). Takéto hodnoty sú dostatočné a naše riešenie je použiteľné pri riadení reálnych systémov.

7 Záver

Táto práca opisuje možnosti zberu dát vo FOSS. V prvej časti sme analyzovali možnosti zberu dát použitím FOSS a na základe tejto analýzy sme si zvolili softvér, do ktorého sme implementovali vlastný komunikačný modul. Na implementáciu sme využili existujúci hardvérový ovládač Hudaqlib. S týmto ovládačom sa nám podarilo vytvoriť funkčný komunikačný modul do Xcosu. Pomocou nášho komunikačného modulu sa nám podarilo riadiť sústavu magnetickej levitácie. Zamerali sme sa hlavne na analógový modul, pretože k nemu sme mali najlepšie podmienky na reálne testovanie. Dosiahnutá hodnota vzorkovacieho času bola v rádoch 10-tok až 100-viek μs, pred implemetáciou sme očakávali tento čas v rádoch jednotiek až desiatok ms. Vzorkovacia frekvencia bola teda vyššia ako sme očakávali.

Ďalšia práca v tejto oblasti sa môže venovať testovaniu ďalších funkcií meracej karty MF624, ktoré sme nemali možnosť poriadne vyskúšať. Pokračovaním práce by mohlo byť aj vytvorenie Comedi ovládača z ovládača/knižnice Hudaqlib.

8 Literatúra

[1] MASTERS, Jon – BLUM, Richard: Linux profesionálne – programovaní aplikací. Brno: Zoner Press, 2008. 539 s. ISBN 978-80-86815-71-8.

[2] CORBET, Jonathan – RUBINI, Alessandro – KROAH-HARTMAN, Greg: Linux Device Drivers, Third Edition. United States of America: O'Reilly Media, Inc., 2005. 615 s. ISBN 0-596-00590-3.

[3] ABDALLAH, M.- ELKEELANY, O. A Survey on Data Acquisition Systems DAQ. [pdf]. Fullerton, CA: International Conference on Computing, Engineering and Information, 4-4-2009. [21-4-2010]. <http://ieeexplore.ieee.org/>

[4] SCHLEEF, David – HESS, Frank – BRUYNINCKX, Herman: The Control and Measurement Device Interface handbook. [online]. 2003. [19-4-2010]. <http://come-di.org/doc/index.html>

[5] Scicos Team. Constructing new blocks in Scicos. [pdf]. 3-3-2009. [21-4-2010]. <http://www.scicos.org/Formation_scicos_mars_2008.pdf>

[6] <http://www.humusoft.com/>

[7] MANNORI, Simone – NIKOUKHAH, Ramine – STEER, Serge: Free and Open Source Software for Industrial Process Control Systems. [pdf]. France: INRIA-Rocquencourt. [17-4-2010]. <http://www.scicos.org/ScicosHIL/angers2006eng.pdf>

[8] BUCHER, Roberto – BALEMI, Silvano: Scilab/Scicos and Linux RTAI - A unified approach. [pdf]. Toronto, Canada: Dept. of Innovative Technol., Univ. of Appl. Sci. of Southern Switzerland, Lugano-Manno. 31-8-2005. [17-4-2010]. <http://ieeexplore.ieee.org/>

[9] Scilab manual. [online]. [21-4-2010]. <http://www.scilab.org/product/man/>

[10] <http://www.scilab.org/>

[11] <https://www.rtai.org/>

[12] <http://www.gnu.org/software/octave/>

[13] Humusoft Data Acquisition Library Reference Manual [pdf].29-6-2007 [15-4-2010]. <http://www.penguin.cz/~fojtik/hudaqlib/hudaqlib.pdf>

9 Zoznam príloh

Príloha A: CD médium

Príloha B: Techniká dokumentácia

Príloha B: Technická dokumentácia

V tejto časti bude stručne popísaný postup inštalácie potrebného softvéru a spôsob použitia nášho bloku v Scilabe, technickou dokumentáciou sú v podstate aj kapitoly 3, 4 a 5, preto tu sa nachádza len stručné zhrnutie.

Je potrebné mať nainštalovanú nejakú modernú linuxovú distribúciu (my sme používali Ubuntu 9.04). Najprv je potrebné stiahnuť a nainštalovať balík Scilab 5.2.1 prípadne novší, buď z oficiálnej stránky [10] alebo z repozitárov (balík štandardne obsahuje aj Xcos). Potom je potrebné si nakopírovať balík *HuLinux-2.3.2.tgz* z CD alebo stiahnuť z http://www.penguin.cz/~fojtik/hudaqlib/hudaqlib.html. My sme používali Hudaqlib 2.3.2, fungovať by mali aj novšie tie sme však netestovali. Potom niekam tento balík rozbaliť(názov adresára je *Hulinux-2.3.2*, predpokladá sa že toto sa nebude meniť), skompilovať a nainštalovať (návod je súčasťou Hudaqlib v readme.txt). V adresáry *Hulinux-2.3.2* sa nachádzajú všetky potrebné súbory, v závislosti od distribúcie sa mohla pri inštalácií dynamická knižnica *libhudaqlib.so* nakopírovať aj do niektorého adresára s knižnicami, napr. */lib*. My predpokladáme použitie hlavičkového súboru *hudaqlib.h* a knižnice *libhudaqlib.so* priamo z adresára *Hulinux-2.3.2*. Možno bude treba zmeniť práva k zariadeniu MF624, inak treba spúšťať Scilab s administrátorskými právami, aby mohol pristupovať k tomuto zariadeniu.

Keď máme takto pripravený Scilab a ovládač s knižnicou pre MF624. Môžeme pokračovať s vložením nášho komunikačného modulu. Najprv si nakopírujeme zdrojové súbory a *install_script.sce* do ľubovoľného pracovného adresára. V zdrojových kódoch je potrebné zmeniť cestu k hlavičkovému súboru *hudaqlib.h*, podľa toho kde máme *Hulinux-2.3.2*. Spustíme Scilab a pomocou príkazu *cd '/CESTA K PRACOVNÉMU ADRESÁRU/'* sa dostaneme do tohto nášho pracovného adresára. Príkazom *ls* môžeme overiť či sa tu nachádzajú potrebné súbory. Ak je všetko v poriadku tak v súbore *install_script.sce,* nastavíme, ktorú funkciu chceme použiť a nastavíme cestu k *libhudaqlib.so* napr. takto *ilib_for_link('readwrite','read-write_block.o',["/CESTA/libhudaqlib"],"c").* Parametre tejto funkcie boli popísané v kapitole 5.4. Ak všetko prebehlo bez problémov môžeme funkciu načítať príkazom *exec loader.sce.* (pri ďalšom štarte sa už bude funkcia načítavať automaticky).Môžeme do palety pridať náš blok ako bolo popísané v kapitole 5.6 alebo otvoriť nejakú zo schém z prílohy A.