TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: 2612T – Elektrotechnika a informatika

Studijní obor: 3902T005 – Automatické řízení a inženýrská informatika

Systém pro komunikaci s měřící kartou (v C#)

Communication System for I/O Boards

Diplomová práce

Autor: Pavel Vašát

Vedoucí DP : Doc. Ing. Osvald Modrlák, CSc.

Konzultant: Ing. Lukáš Hubka

V Liberci dne 2.1.2007Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **souhlasím** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

V Liberci dne 4. 1. 2007.

....

Pavel Vašát

Anotace

Systém pro komunikaci s měřící kartou (v C#)

Cílem diplomové práce je seznámit se s možnostmi využití softwarové platformy .NET a programovacího jazyka C# při komunikaci s měřícími kartami v PC, sledování průběhů dynamických systémů a jejich regulaci. Navržený měřící systém následně realizovat a ověřit na reálné soustavě v laboratoři.

Během programování bylo vyzkoušeno několik přístupů a možných implementací kódu. Výsledný systém je stabilní a nabízí univerzalitu v ovládání měřících karet firmy Advantech. Narozdíl od podobných produktů nabízí možnost okamžitého sledování systémů a jejich reakcí na akční zásahy i studentům bez znalosti programovacích algoritmů. Pro ověření poznatků byla použita soustava tachodynamo (řízení otáček tachometru propojeného pružnou spojkou s DC-motorem) z laboratoře KŘT. Kvalita získaných průběhů signálu sledovaných vstupů byla velmi dobrá a plně odpovídala totožné situaci sledované v aplikaci Matlab Simulink. Systém ControlSystem se osvědčil jako dobrý základ pro budoucí rozšíření v komplexní nástroj, pomáhající studentům při výuce.

Abstract

Communication system for I/O Boards

The aim of thesis is to acquaint with capabilities of .NET platform and programming language C# for usage in communication with I/O boards inserted in PC, acquiring dynamic system outputs and controlling them by active inputs. Designed System is to be tested on real system in laboratory.

There were few different ingresses to try, during the design of application. Final system is stable running and gives versatility in user control over I/O boards made by Advantech. In contrast to similar software solutions, ControlSystem enables immediate output monitoring and system control even to students without programming skills. Application was tested on system tachodynamo (rotation speed control of tachometer connected to DC-motor with elastic clutch) from KRT laboratory. Acquired results were good and fully comparable with results from Matlab Simulink.

Obsah

TITULNÍ STRANA 1
Zadání diplomové práce 2
Prohlášení 3
Anotace 4
Abstract 4
Obsah 5
1. Úvod 7
    1.1 Problém 7
    1.2 Cíl 7
2. Resumé stávajících řešení 8
    2.1 Advantech Device Test Utility 8
    2.2 Advantech Studio 9
    2.3 Matlab 10
    2.4 LabView 11
    2.5 PCI-1710HG 12
    2.6 REXControls 13
3. Použitý software 13
    3.1 Jazyk C# a platforma .NET 13
4. Hardware 14
5. Komunikace s kartou v jazyce C# 16
6. Běh Aplikace 18
    6.1 Struktura Aplikace 19
    6.2 Vzhled Aplikace 20
    6.3 Navázání komunikace s kartou 20
    6.4 Dynamická tvorba komponent za běhu programu 22
    6.5 Konfigurace portů 23
    6.6 Měření 24
    6.7 Spuštění měřícího procesu 25
    6.8 Zastavení měřícího procesu 26
    6.9 Metoda GetData 26
    6.10 Nastavení hodnot výstupních portů 27
    6.11 Programové zpracování výstupů 28
    6.12 Definice periodických výstupních funkcí 29
        6.12.1 Sinusoida 29
        6.12.2 Obdélníkový signál 29
        6.12.3 Inkrementální funkce 30
7. Časování 31
    7.1 Výpočetní náročnost časování 33
8. Zobrazovací funkce 34
    8.1 Frekvence a přesnost zobrazování 36

9. Datové výstupy 40
9.1 Textový soubor 41
9.2 Soubor CSV 41
9.3 Soubor XML 42
10. Datové vstupy 43
11. PID Regulace 45
11.1 Proporcionální složka 46
11.2 Integrační složka 46
11.3 Derivační složka 47
11.4 Diskretizace přenosu PID regulátoru 48
11.5 Aplikace PID regulace v ControlSystemu 49
11.5.1 Parametry P,I a D použité v aplikaci 49
11.5.1 Text psaný uživatelem 50
11.5.2 Součást kódu aplikace 50
11.5.3 Interní metoda aplikace 50
11.5.4 Interní třída aplikace 50
11.5.5 Externí třída aplikace 51
12. Výsledky měření na reálné soustavě 52
12.1 Postup měření 53
Závěr 55
Použitá literatura 56

1. Úvod

1.1 Problém

Během výuky na fakultě mechatroniky se student seznamuje s řadou aplikací, které slouží k získávání dat a sledování chování modelových systémů. Používané programy jsou komerční a tudíž finančně náročné. Dalším problémem je nutnost seznámení studenta s používáním programu. Tyto aplikace jsou ve většině případů velmi komplexní a práce s nimi není pro začátečníka jednoduchá. Tedy příprava studenta může být časově náročné v případě, že nebyl během předchozích studií seznámen s programovacími algoritmy či podobným softwarem. Řada studentů, studujících jiné obory, nebude práci s podobnými programy v budoucnu potřebovat, ale je třeba je rychle a efektivně seznámit s průběhy jednotlivých složek sledovaného systému a reakcí na akční zásahy.

Proto vznikl požadavek na aplikaci, která by umožnila všem studentům rychlý a intuitivní přístup k měřícímu a regulačnímu procesu bez předchozích znalostí programování či grafického návrhu modelového procesu. Je třeba nalézt optimální poměr mezi rychlostí programu a jeho výpočetní náročností, stejně jako ošetření neobvyklých událostí během chodu. Grafický vzhled aplikace a ovládací prvky musí být umístěny logicky a umožňovat rychlý zásah uživatele do probíhajícího děje. Po skončení měření musí mít uživatel možnost daný průběh uložit do paměti počítače v požadovaném formátu, případně jej v budoucnu nahrát zpět do aplikace.

1.2 Cíl

Cílem je realizovat aplikaci, která by odstranila negativní stránky komerčních aplikací, byla velmi jednoduchá na ovládání a nevyžadovala žádné externí znalosti studentů o programování, nevyžadovala hardwarové zásahy do systému pro svou funkci a byla co možná nejvíce univerzální. Proto by měl program fungovat na všech kartách Advantech dostupných v laboratořích KŘT, bez jakýchkoli dalších nastavení. Důležitá je i možnost rozšiřování aplikace v budoucnu požadovanými algoritmy či potřebnými dynamickými knihovnami. Naměřené průběhy by měly být porovnány s výsledky zavedených komerčních produktů (např. Matlab), pro ověření důvěryhodnosti měření. Během realizace je třeba se soustředit na rychlost a efektivitu celého kódu pro dosažení optimálních výsledků během používání vysokých vzorkovacích frekvencí pro sledování systému.

2. Resumé stávajících řešení

Před započetím práce bylo třeba seznámit se s podobnými produkty, dostupnými na trhu. Naprostou většinu představují aplikace typu Matlab, které umožňují podporu měřících karet pomocí volitelných rozšiřujících modulů. Problémem těchto aplikací je jejich cena – která i ve školních licencích nebývá nejnižší. Proto je snahou nalézt bezplatné freewarové aplikace, které alespoň zčásti nahradí profesionální programy. Po pečlivém prohledání zdrojů na internetu bylo shledáno, že podobných aplikacích je velmi poskrovnu. Navíc jsou převážně vždy zpoplatněné a tedy pro běžného uživatele leckdy nedostupné. Licence pro jejich používání stojí desítky až stovky tisíc korun. Výjimku představují tzv. studentské licence, které nabízejí omezenou funkčnost avšak za nulovou cenu. Nezbývá tedy, než se pokusit o naprogramování vlastní aplikace, která bude splňovat naše požadavky.

Firma Advantech ve svém manuálu k dynamické knihovně Adsapi32.dll zmiňuje podporu následujících programovacích jazyků:

Microsoft Visual C++, Microsoft Visual Basic

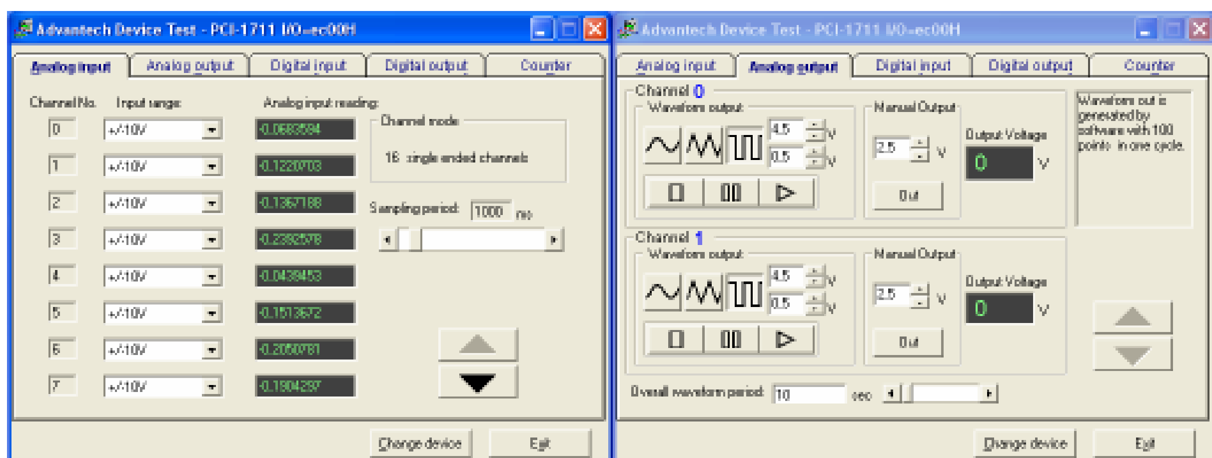
Borland Delphi

Borland C++ / C++ Builder

Pro tyto nabízí knihovny zkompilované pro příslušný jazyk. Ostatní programovací jazyky nejsou nativně podporovány. Avšak díky koncepci platformy .NET lze při psaní např. v jazyce C# používat knihovny psané v jazyce C++ a naopak. To nám umožňuje používat tyto ovladače i přes fakt, že jazyk C# není podporován.

Nyní se seznámíme se stručným přehledem několika dostupných aplikací:

2.1 Advantech Device Test utility



Obrázek č.1 Vzhled aplikace Advantech Device Test

Aplikace dodávaná firmou Advantech společně s ovladači ke kartě – lze získat i samostatně z internetu - zadarmo. Po instalaci a spuštění detekuje nainstalované karty Advantech v PC a umožní jejich otestování. V programu lze nastavovat hodnoty a průběhy výstupních signálů, rozsahy vstupních signálů, kontrolovat digitální vstupy / výstupy a čítač. Samotná aplikace je velmi strohá a má spíše informativní charakter, díky němuž lze otestovat funkčnost karty. Pro měření je naprosto nevhodná z několika důvodů:

- Nejnižší možná vzorkovací perioda je 200ms – tedy 5Hz – vhodné jen pro nejpomalejší systémy.
- Absence jakékoli vizualizace – jediné co uživatel vidí jsou periodicky se měnící hodnoty vstupních signálů. Zobrazení do grafů nelze.
- Nemožnost ukládání naměřených hodnot na disk – program nezaznamenává žádnou formou průběh naměřených signálů

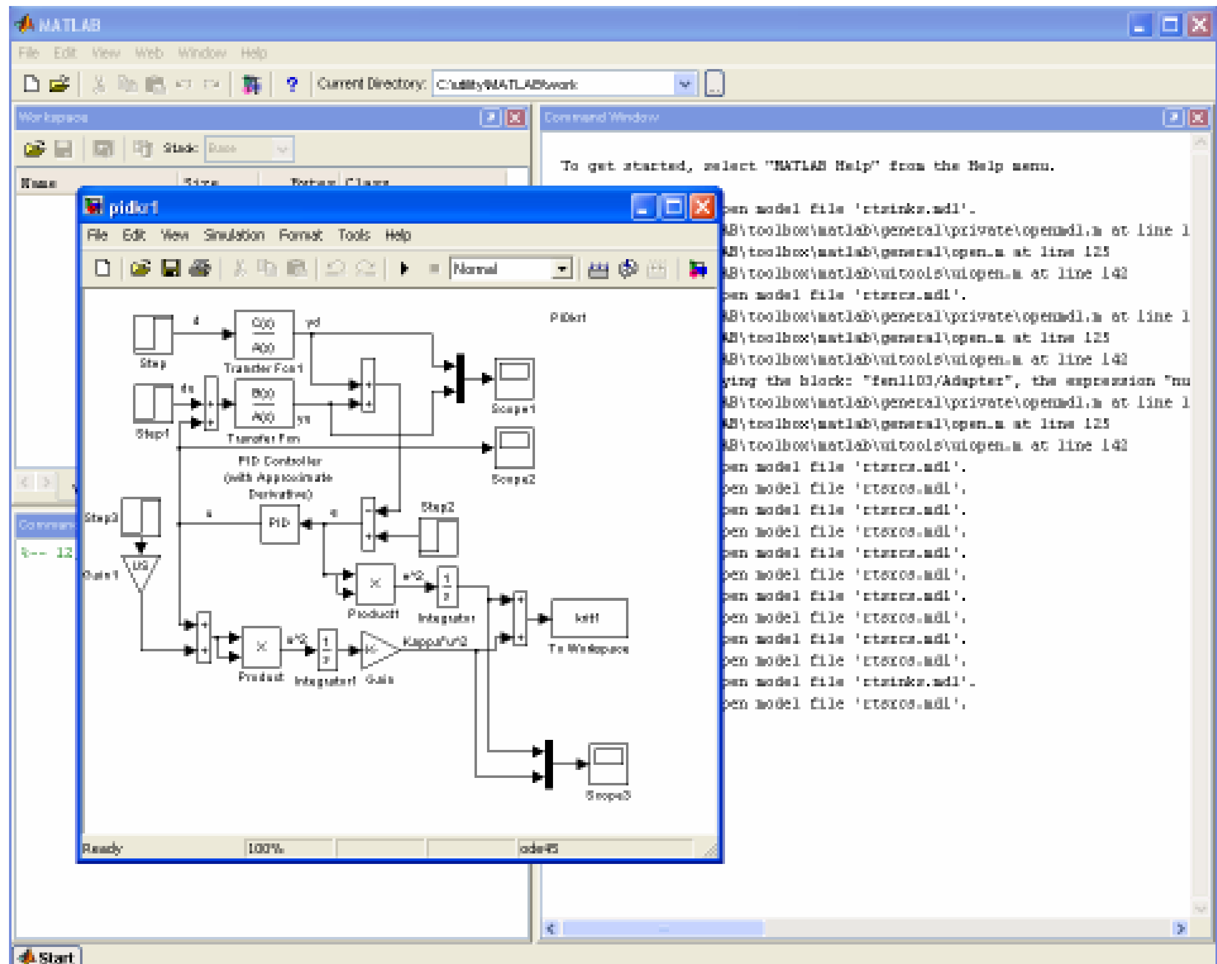
2.2 Advantech Studio

Obrázek č.2 Vzhled aplikace Advantech Studio

Grafické rozhraní pro komunikaci s kartami Advantech – nahrazuje předchozí Advantech Genie SW. Programování je založeno na sestavování grafických bloků – představující jednotlivé objekty. Po „naprogramování“ měřícího / regulačního schématu je systém funkční v plně automatickém režimu. Ovládání je intuitivní a jednoduché. Standardní vzorkovací frekvence je 200Hz, maximální frekvence je

20kHz – což je dostatečné i pro velmi rychlé děje. Průběh veličin může být zobrazován v real-time grafech Nevýhodou tohoto rozhraní je placená licence.

2.3 Matlab

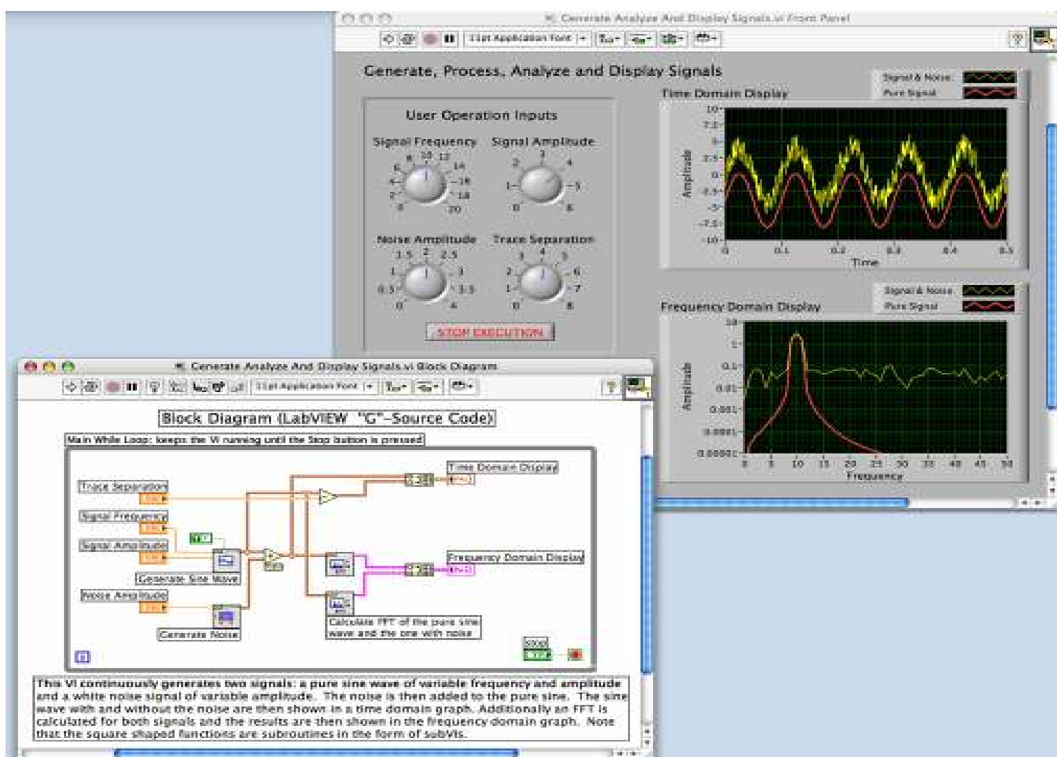


Obrázek č.3 Vzhled aplikace MathWoks Matlab 6.5

Firma MathWorks (producent Matlabu) a Advantech oficiálně uzavřeli „spojenectví“ na poli podpory měřících karet. Proto jsou nativně podporovány v samotných ovladačích programu Matlab. Podporuje všechny typy karet Advantech. Komunikace probíhá přes Real Time Toolbox komponentu aplikace Matlab Simulink. Matlab je velmi komplexní nástroj a umožňuje veškeré myslitelné operace s kartou. Získaná data lze jednoduše sledovat v reálném čase na grafech, ukládat do libovolných formátů a dále zpracovávat. Samozřejmostí je i real-time regulace procesů na základě získaných dat. Jedná se o jeden z nepoužívanějších softwarů na Katedře Řídící Techniky fakulty Mechatroniky. Nevýhodou programu Matlab je jeho cena.

Kompenzována je možností zakoupit pouze zvolené moduly, avšak i jejich cena je extrémně vysoká. Matlab Simulink pro průmyslové využití stojí cca 630 000 Kč.

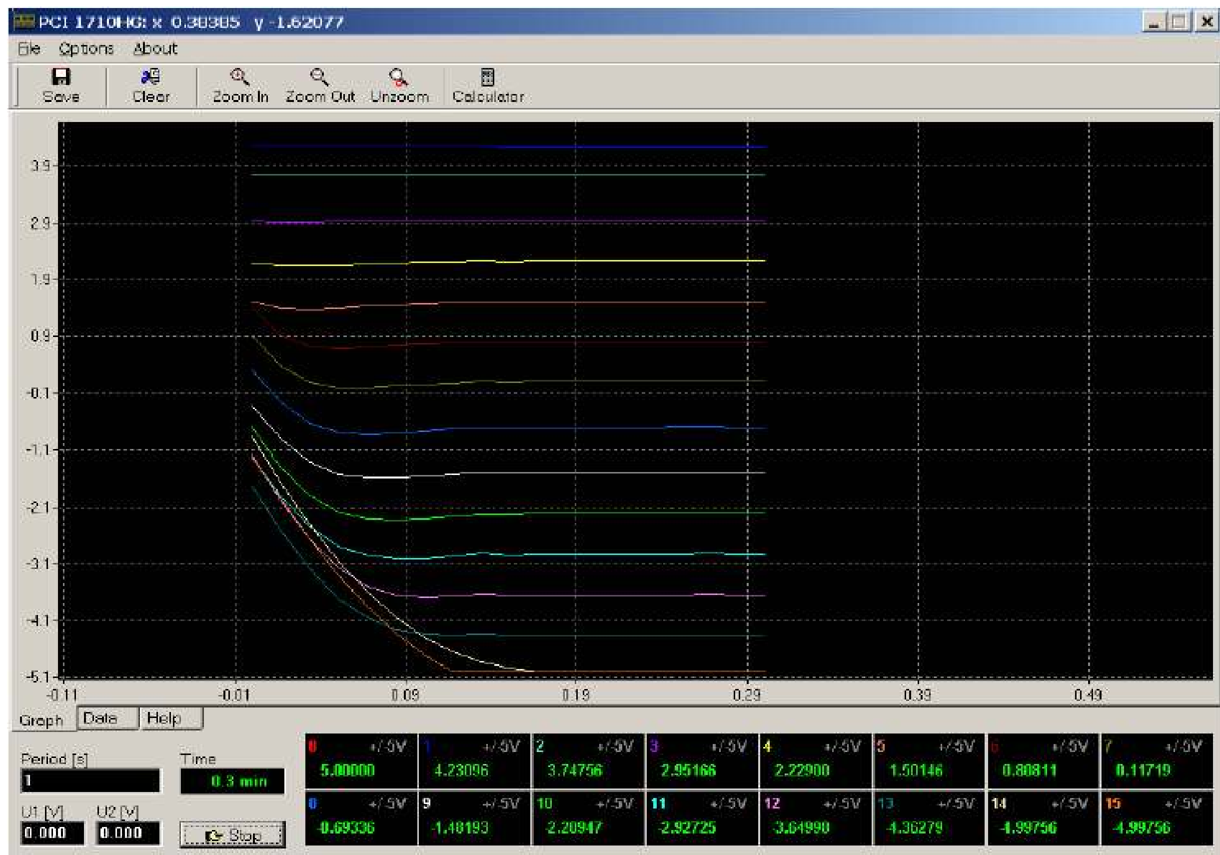
2.4 Labview



Obrázek č.4 Vzhled aplikace LabView

Firma Advantech nativně podporuje software LabView - má k dispozici speciální ovladač svých karet pro aplikace Labview – Advantech DA&C LabView driver. Podporuje všechny funkce karet – jedná se o nadstavbu nad základním ovladačem adsapi32.dll. Tato Aplikace, podobně jako Matlab, umožňuje komplexní správu vstupně výstupních portů, sledování průběhů na uživatelem definovaných grafech, regulaci a modelování systémů a následné ukládání či načítání průběhů z datových souborů. Grafický návrh měřících systémů (typu Drag&Drop) umožňuje práci i pracovníkům bez znalosti programovacích jazyků. Umožňuje tvorbu nezávislých aplikací, běžících na systémech bez nainstalovaného softwaru Labview. Při návrhu této aplikace byl kladen velký důraz na nízké hardwarové požadavky systému. Nevýhodou je opět placená licence – s výjimkou studentské verze, která je poskytována zdarma. Výhodou je nezávislost na operačním systému a platformě, na které hardware běží.

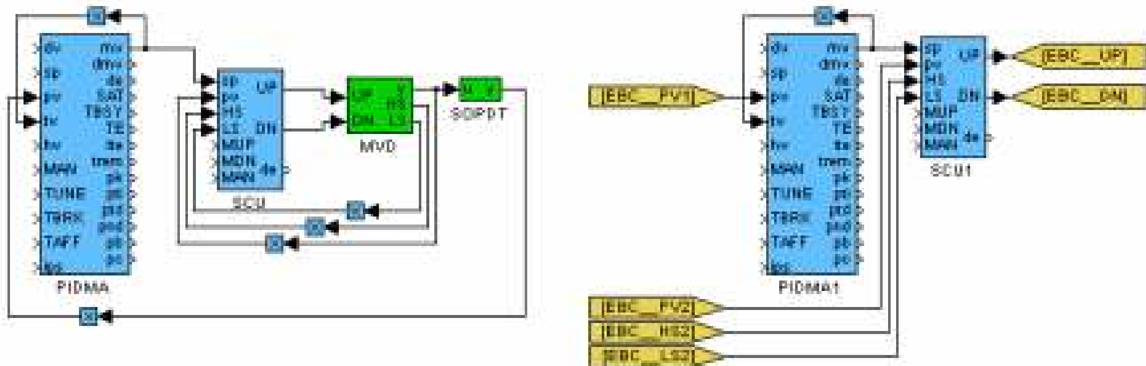
2.5 PCI – 1710 HG



Obrázek č.5 Vzhled aplikace PCI-1710HG

Aplikace vyvinutá na fakultě fyziky Masarykovy univerzity v Brně. Je naprogramována v Borland Delphi 5.0 a slouží primárně k zobrazování a ukládání naměřených dat. Svou koncepcí a účelem se nejvíce podobá aplikace ControlSystem, o které pojednává tato práce. Program PCI-1710HG umožňuje volit počet a rozsahy měřených výstupních kanálů, měnit rozsahy vstupních kanálů. Nevýhodou této aplikace je fakt, že získávání vzorků probíhá na základě událostí komponenty *timer*. Ta je silně nevyhovující a nelze zaručit ekvidistantnost vzorkování s dostatečnou přesností. Taktéž nejnižší nastavitelný interval mezi měřeními je 0,1s – tedy vzorkovací frekvence 10 Hz – což je použitelné jen na nejpomalejší systémy – např. úloha Fén. Tato aplikace umožňuje i programování řídícího algoritmu karty – pro nastavování adekvátních hodnot výstupu, jakožto reakce na změnu vstupu. Psaní vkládaného kódu vyžaduje znalost jazyka *Delphi*, proto nebude tato možnost dostupná všem uživatelům. Ostatně stejný problém byl řešen i v ControlSystemu.

2.6 REXControls



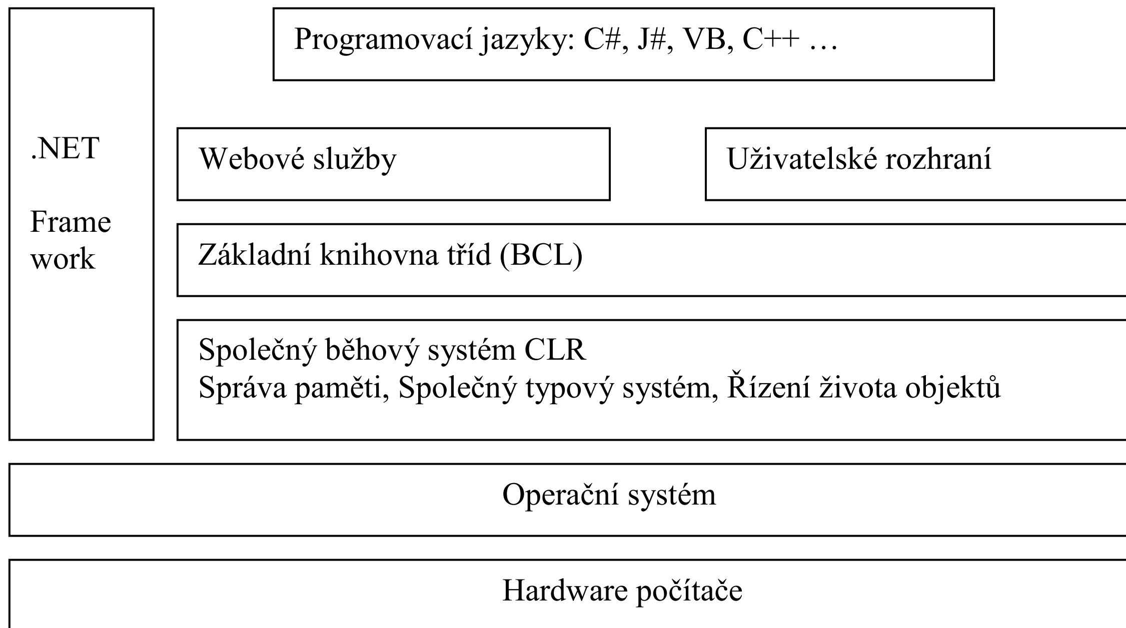
Obrázek č.6 Vzhled aplikace REXControl

Multiplatformový řídící systém – kompatibilní se systémem Matlab-Simulink, naprogramován v jazyce C/C++. Koncovému uživateli se jeví jako Simulink, avšak k jeho běhu není třeba mít licenci Simulinku. REX ve skutečnosti obsahuje mnoho funkcí navíc a je vhodnější pro řízení systémů – například v lepší zpětné kontrole funkce PID regulátorů.

3. Použitý Software

3.1 Jazyk C# a platforma .NET:

Jedná se o poměrně novou platformu, otevírající nový přístup k programování. Jádrem systému je společné běhové prostředí (common language runtime – CLR), zajišťující běh programů, přeložených z různých jazyků do mezijazyka MSIL. CLR umožňuje jejich vzájemnou spolupráci – takže jednotlivé části programu mohou být napsány v různých jazycích. To nám dává možnost pracovat s týmem programátorů, aniž by psali aplikace ve stejném jazyce. Tímto způsobem je implementována knihovna *Adsapi32.dll* (napsaná v jazyce C++) do knihovny *AdvantechDAC.dll* (psané v jazyce C#). Naopak velkou nevýhodou je nutnost přítomnosti .NET Frameworku na počítači, kde chceme zvolenou aplikaci spouštět. Taktéž hardwarová náročnost tohoto řešení je vyšší než v případě aplikací naprogramovaných mimo .NET.



Obrázek č.7 Diagram platformy .NET

CLR se též stará o automatickou správu paměti, řízení doby života objektů a další nezbytné věci. Tím se snižují nároky na programátora a může se věnovat konkrétnímu řešení daného problému. Během programování aplikace však bylo zjištěno, že tato automatická správa nefunguje ve všech situacích tak jak by měla, proto se na ni nelze spolehnout a některé situace musíme ošetřit ručně.

Samotný jazyk C# je zvláštní směsicí jazyků C++ a Java. První zmínky o něm pronikly na veřejnost v roce 2000. Oficiálně byl uveden na trh v roce 2002 jakožto součást Visual Studia. Jeho nespornou výhodou je dostupnost ve freeware verzi studia – Expres. V této verzi byla naprogramována celá aplikace ControlSystem.

4. Hardware

Testování naprogramovaných skriptů probíhalo na jednom z nejrozšířenějších typů karet Advantech na katedře řízení – a sice modelu Advantech PCI 1711. Jedná se o model, který obsahuje analogové výstupy i vstupy – bylo tedy možné otestovat funkčnost všech programových bloků.

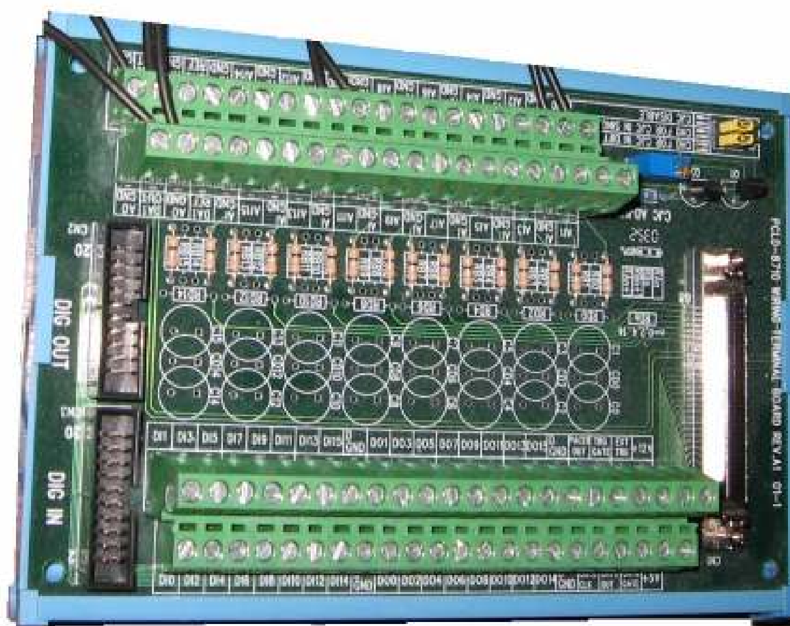
Obrázek č.8 Karta Advantech PCI 1711

Jedná se o 12-bitovou kartu určenou pro instalaci do PCI slotu počítačů. Je to střední model řady 17, navržený s ohledem na dostatečný výkon při zachování nízké ceny. Karta patří do kategorie Low Cost – tedy nízkonákladových. Její specifikace jsou následující:

- PCI sběrnice pro přenos dat
- 16 Analogových vstupů – typu Single-Ended
- 12 bitový A/D převodník s vzorkovací frekvencí 100kHz
- Programovatelný rozsah vstupu pro každý vstupní kanál
- FIFO buffer pro 1000 naměřených vzorků
- 2 kanálový D/A výstup
- 16 kanálový digitální vstup
- 16 kanálový digitální výstup
- Programovatelný časovač/čítač
- Automatický scan rozsahů kanálů

Karta dále nabízí funkci Plug & Play – tedy snadnou instalaci do PC. Vyžadující pouze instalaci potřebných ovladačů. Není třeba nastavovat žádné jumpery či DIP switche.
Uživatel může libovolně měnit rozsahy vstupních kanálů dle jeho potřeb. Tato konfigurace je uložena v paměti SRAM. Flexibilní design umožňuje více kanálové měření s vysokou vzorkovací periodou. Programování aplikace probíhalo na stolním PC s procesorem AMD

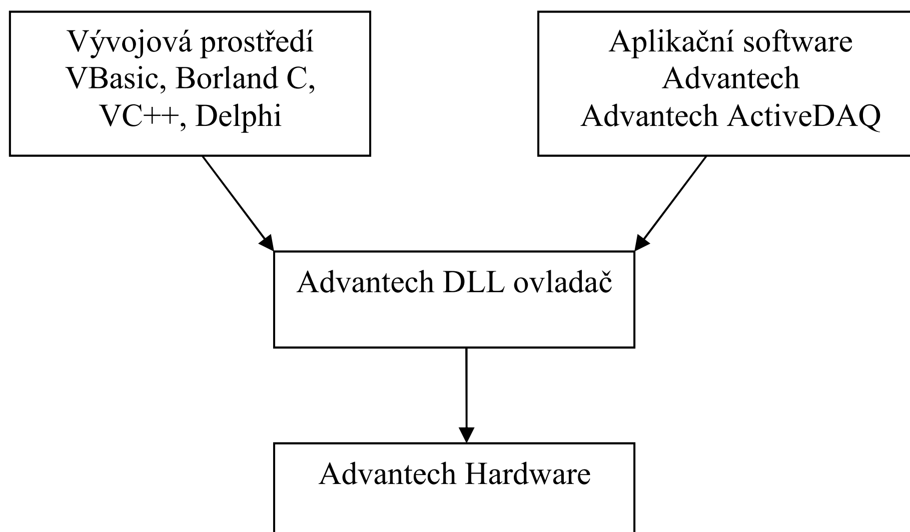
Athlon 64 a systému Windows XP. Druhý systém, který byl používán pro vývoj aplikace byl notebook s procesorem Intel Core 2 Duo opět běžící na systému Windows XP. Přídavné testy časování se uskutečnily na dvou testovacích stavech HIL vybavených průmyslovými PC – jedno s procesorem Penium M, druhé s procesorem Pentium 4 D, opět na Windows XP. Měřící karta byla spojena kabelem se vstupně výstupním přípravkem pro připojování externích systémů PCLD-8710 Wiring Terminal Board Rev.A1 01-1 též od firmy Advantech.



Obrázek č.9 přípravek PCLD-8710

5. Komunikace s kartou v jazyce C#

Dnešní karty firmy Advantech se v převážné většině případů osazují do PCI slotů základních desek počítačů. Starší verze se osazují i do ISA slotů, které jsou stále přítomné např. ve starších průmyslových počítačích ve výrobních linkách. Po vsunutí karty do slotu je nutné doinstalovat příslušné ovladače ke kartě. Ovladače jsou univerzální pro téměř všechny DAC karty firmy Advantech. Jejich základem je dynamická knihovna *adsapi32.dll*. Díky ní není zapotřebí používat specifické příkazy registru pro různé typy hardwaru. Dává nám možnost programování API v různých programovacích prostředích a jazycích. Pro námi použitou kartu bylo třeba doinstalovat knihovnu *Ads1711.dll*.



Obrázek č.10 Diagram komunikace mezi kartou a PC

Firma Advantech poskytuje nativní podporu knihoven pro následující vývojová prostředí:

Microsoft Visual C++

Microsoft Visual Basic

Borland Delphi

Borland C++, C++ Builder

Ostatní jazyky musí provést konverzi *adsapi32.dll* do svého prostředí. Aplikace Control System používá knihovnu AdvantechDAC.dll, která konvertuje adsapi32.dll do jazyka C#. Je třeba nadefinovat nové metody, obsluhující hardware a provést přetypování na datové typy používané v jazyce C#. Nové metody jsou zabalené v třídě AdsApi32Wrapper – implementování vypadá následovně – uvedeno na příkladu metody obsluhující čtení z analogového vstupního kanálu:

```
[DllImport("adsapi32.dll")]
public static extern uint DRV_AIBinaryIn(int DriverHandle, ref
PT_AIBinaryIn lpAIBinaryIn);
```

nová struktura *PT_AIBinaryIn* je nadefinována následovně:

```
[StructLayout(LayoutKind.Sequential, Pack=4)]
public struct PT_AIBinaryIn
{
        public ushort chan;
        public ushort TrigMode;
```

```
		public IntPtr reading; // USHORT far *reading;
	}
```

nakonec nadefinujeme funkci pro čtení z příslušného kanálu – ve zkrácené formě, bez ošetření výjimek:

```
	public void AnalogInputRead(int startChannel, int channelCount, bool
	externalTrigger, short[] data)
	{
		AdsApi32Wrapper.PT_MAIBinaryIn analogInputBinary;
		analogInputBinary.StartChan = (ushort)startChannel;
		analogInputBinary.NumChan = (ushort)channelCount;
		analogInputBinary.TrigMode = externalTrigger ? (ushort)1 :
		(ushort)0;
		uint lastError = AdsApi32Wrapper.DRV_MAIBinaryIn(driverHandle,
		ref analogInputBinary);
	}
```

V aplikaci Control System provedeme volání metody pro načtení hodnoty z analogového vstupu kanálu 0 následovně

DataAcquisitionAndControl.AnalogInputRead(0,1,false,new short[0])

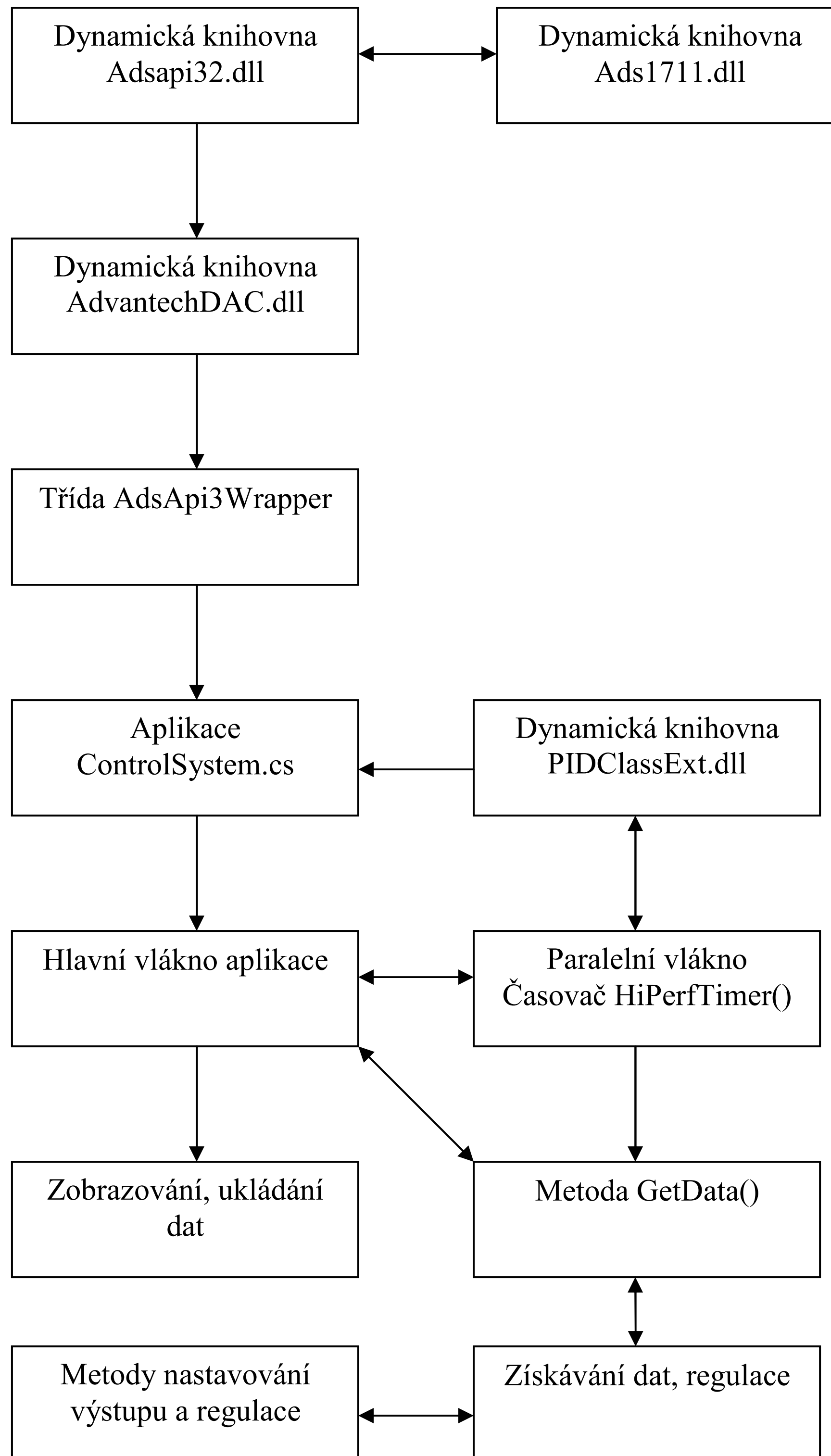
Data jsou načtená do datového typu *short* a poté zpracována.

6. Běh aplikace

Aplikace se spouští souborem *ControlSystem.exe*, s nímž jsou v příslušném adresáři umístěny i všechny potřebné dynamické knihovny. Pro správný běh programu je zapotřebí mít nainstalovaný .NET Framework ve verzi 2.0. Aplikace totiž využívá některé vlastnosti této verze – a není zpětně spustitelná na Frameworku verze 1.1 a nižší. Program během spouštění importuje všechny potřebné knihovny a inicializuje proměnné. Pro další běh je nejdůležitější definice proměnné typu *DataAcquisitionAndControl (dále jen DAC)*, která nám umožňuje komunikovat s kartou. Ta během události *OnLoad* hlavního formuláře získá informace o přítomných kartách Advantech v PC pomocí příkazu *DAC.GetDeviceList()*. Je nutné mít nainstalovánu knihovnu *Adsapi32.dll* – i bez příslušného hardwaru. Tato knihovna se během inicializace nahrává a přes knihovnu *AdvantechDAC.dl*l se transformují její metody do jazyka C#. V případě, že se v počítači nenachází žádná měřící karta, proměnná obsahující informace o kartě zůstane neinicializována a uživateli nenabídne možnost připojit se k vybranému hardwaru. V opačném případě se vytvoří list všech přítomných karet a provede se jejich uskladnění do rolovacího *comboBoxu*.

6.1 Struktura aplikace

V přehledovém diagramu si zobrazíme strukturu aplikace, používaných knihoven a metody které aplikace volá.



Obrázek č.11 Struktura aplikace

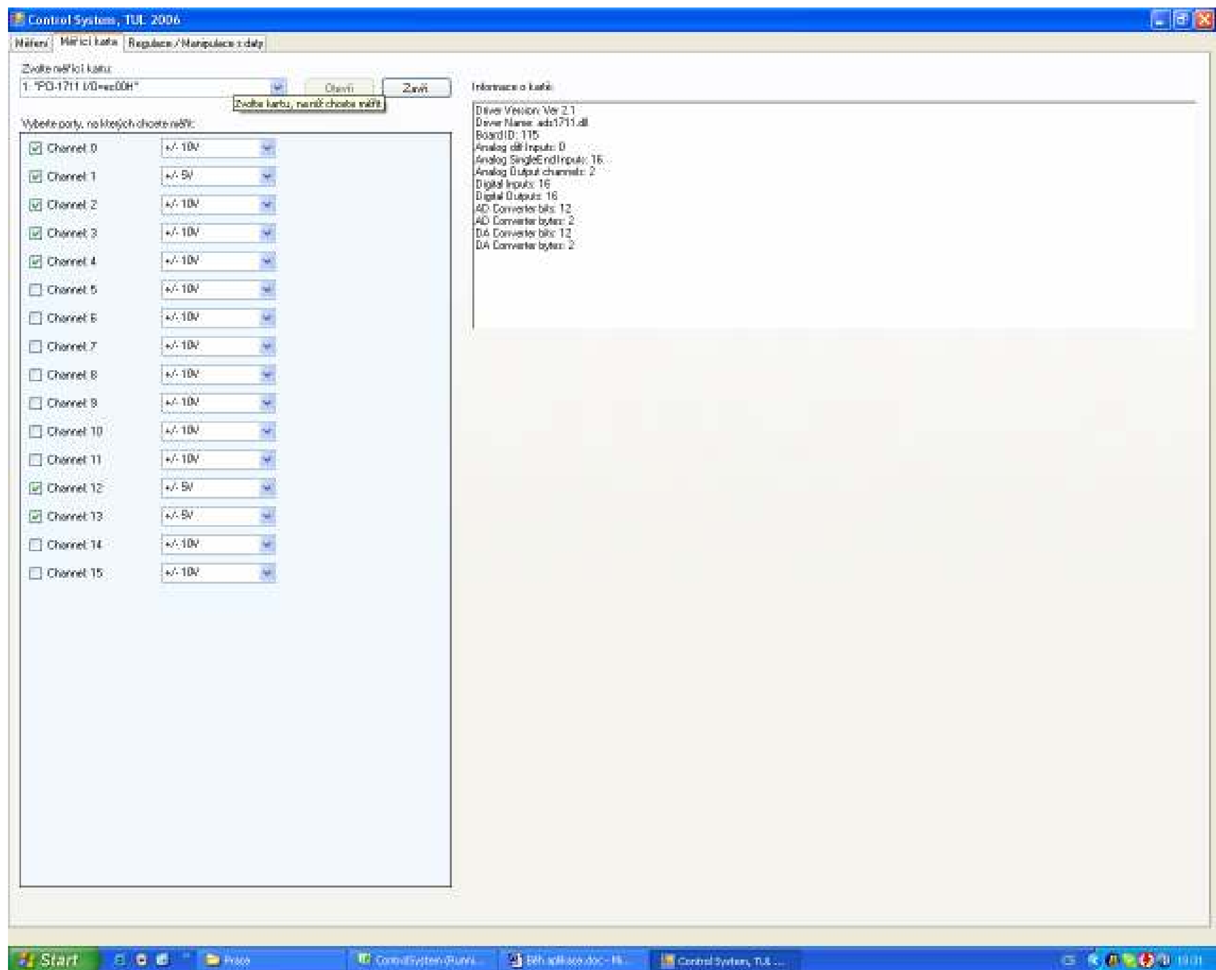
6.2 Vzhled aplikace

Aplikace se spouští v jednom formuláři, který má několik přepínatelných pod-oken pomocí komponenty *tabControl*. Během návrhu bylo odzkoušeny různé varianty grafického uživatelského rozhraní a nakonec bylo zvoleno toto jako nejvhodnější. Zde je stručné porovnání vyzkoušených variant:

- **Několik nezávislých formulářů:** rychlé a intuitivní – možnost přepínání pomocí klávesové kombinace alt+tab, lze sledovat více oken najednou, nevýhodou je zvýšená spotřeba operační paměti počítače, v případě že se vykresluje na více oken bylo patrné i vysoké zatížení procesoru a zpomalení celé aplikace
- **Skrytá okna v hlavním formuláři:** pomalejší řešení – okna přístupná buď přes zobrazovací položku lišty Menu, případně přes příslušnou klávesovou kombinaci, nelze sledovat více oken najednou, nenáročné na paměť počítače
- **Listy (záložky) komponenty *tabControl*:** nakonec zvoleno, jakožto nejideálnější kompromis mezi uživatelskou příjemností a hardwarovou náročností. Okna dostupná z hlavního pomocí jednoho kliknutí, všechna dostupná v kterýkoli okamžik, nelze však sledovat více oken najednou, snížené nároky na paměť počítače

6.3 Navázání komunikace s kartou

Probíhá na záložce „*Měřící karta*“. Uživateli se nabídne soupis všech karet fyzicky přítomných v PC v rolovacím menu. Po vybrání příslušné karty a stisku tlačítka *Otevři* se naváže komunikace s kartou. V aplikaci ControlSystem nelze komunikovat s více kartami najednou – lze to však obejít vícenásobným spuštěním programu. V tomto případě si však musíme uvědomit dopad na výkon celé aplikace. Přesné časování, byť běžící v paralelním threadu je velmi náročné a při spuštění více podobných časovačů může dojít k nechtěným konfliktům o systémové prostředky – a tím pádem ke snížení přesnosti měření. Proto se doporučuje spouštět pouze jednu aplikaci a komunikovat jen s jednou kartou, pokud chceme dosáhnout požadované přesnosti při vysokých vzorkovacích frekvencích. Jiná situace nastává v systémech s více jádrovými procesory, které se na našem trhu začínají mohutně rozvíjet. Tam je situace jiná a umožňuje efektivní práci více současně spuštěných aplikací. Jediným problémem zůstává fakt, že operační systém Windows někdy jedná nesmyslně a nevyužívá potenciál více jader.



Obrázek č.12 ControlSystem – záložka „Měřící karta

Vraťme se však k samotné aplikaci a jejímu běhu. Po stisku tlačítka *Otevři* se komunikace naváže příkazem *DAC.DeviceOpen()* – pro jeho bezchybný chod je potřeba nadefinovat proměnnou *DAC.DeviceNumber* – nebo-li identifikační číslo kanálu měřící karty. Tato hodnota není implicitně nastavena! Proto musí dojít k její inicializaci - ta probíhá již při detekci nainstalovaných zařízení – indexy se přiřazují dle pořadí karet v PCI slotech počítače. Dle tohoto klíče je vyplněno i výběrové menu s kartami – karta ve slotu s nejnižším pořadovým číslem bude zobrazena jako první. Ihned po navázání komunikace s kartou se v poli *richTextBox* zobrazí základní informace o kartě

- verze a jméno ovladačů
- identifikační číslo desky
- počty analogových a digitálních vstupů a výstupů
- specifikace A/D a D/A převodníků

Současně se v panelu po výběrovým menu s nabídkou karet dynamicky vytvoří příslušný počet *checkBoxů* a *comboBoxů* – dle počtu analogových vstupů. Tato dynamická tvorba je složitější, a proto ji věnujeme následující odstavec.

6.4 Dynamická tvorba komponent za běhu programu

Narozdíl od klasického přístupu programování ve Visual Studiu – WYSIWYG (What You See Is What You Get) – nebo-li – co vidíme, to dostaneme – se někdy musí programátor vypořádat s tvorbou komponent až za běhu programu, na základě uživatelových podnětů.

Stejný problém byl řešen i zde. Aplikace je již od začátku koncipována jakožto univerzální pro všechny karty firmy Advantech, využívající dynamické knihovny adsapi32.dll. Tyto karty se však výrazně liší ve svých hardwarových návrzích – hlavně v počtu vstupních kanálů. Proto nebylo možné předem nadefinovat jejich pole. Počet kanálů se získá za běhu programu příkazem *DAC.Features.AnalogInputSingleEndedChannels* – pokud se nám jedná o tento druh vstupů. Po otevření karty se inicializuje datové *data[,]* pole v rozměru počet „kanálů x počet vzorků“. Počet vzorků byl stanoven dle zamýšleného účelu aplikace na maximálních 360 000 – o tom později v sekci „Měření“.

Stejným způsobem se vytvoří i příslušný počet dynamických komponent. Jedná se o již zmiňovaný pár *checkBox* a *comboBox*. První slouží k zaškrtnutí – aktivaci – žádaného kanálu, jehož průběh chce uživatel sledovat. Druhý slouží k nastavení rozsahu daného kanálu – v naprosté většině karet se jedná o volbu mezi +- 5V nebo +- 10V. Bylo by jednodušší implicitně navolit rozsah +-10V, čímž bychom si zjednodušili definici jen na jednu komponentu, ale pro některé aplikace je vyžadována vyšší přesnost měření – a té dosáhneme na nižším rozsahu, rozděleném na stejný počet úrovní jako vyšší rozsah.

Samotnou tvorbu komponenty v jazyce C# si ukážeme na konkrétním příkladu – tvorba *checkBoxu*:

```
CheckBox checkBox = new CheckBox();
checkBox.Top = topMost;
checkBox.Left = 10;
checkBox.Text = "Channel: " + i.ToString();
checkBox.Name = "ch" + i.ToString();
checkBox.CheckedChanged += new EventHandler(checkBox_CheckedChanged);
panel.Controls.Add(checkBox);
```

V prvním řádku se vytvoří nový objekt typu CheckBox. V dalších řádcích se nastaví jeho vizuální vlastnosti – rozměry, umístění, případně barva, doprovodný text a podobné. Důležité je přiřadit správné jméno komponentě, aby nevznikla při běhu programu duplicita. To by mělo za následek kolaps celé aplikace. Na předposledním řádku nadefinujeme i případné ošetření události – zde událost typu „změna zaškrtnutí" . A na posledním řádku komponentu přidáme její na formulář – zde na komponentu typu panel. Nyní si ukážeme definici události, ošetřující chování dynamické komponenty:

```
private void checkBox_CheckedChanged(object sender, EventArgs e)
        {
                string port;
                short portNumber;
                port = ((CheckBox)sender).Name.ToString();
                portNumber = Convert.ToInt16(port.Remove(0, 2));
                if (((CheckBox)sender).Checked)
                {
                    cardConfig[0, portNumber] = 1;
                }
                else
                {
                    cardConfig[0, portNumber] = 0; } }
```

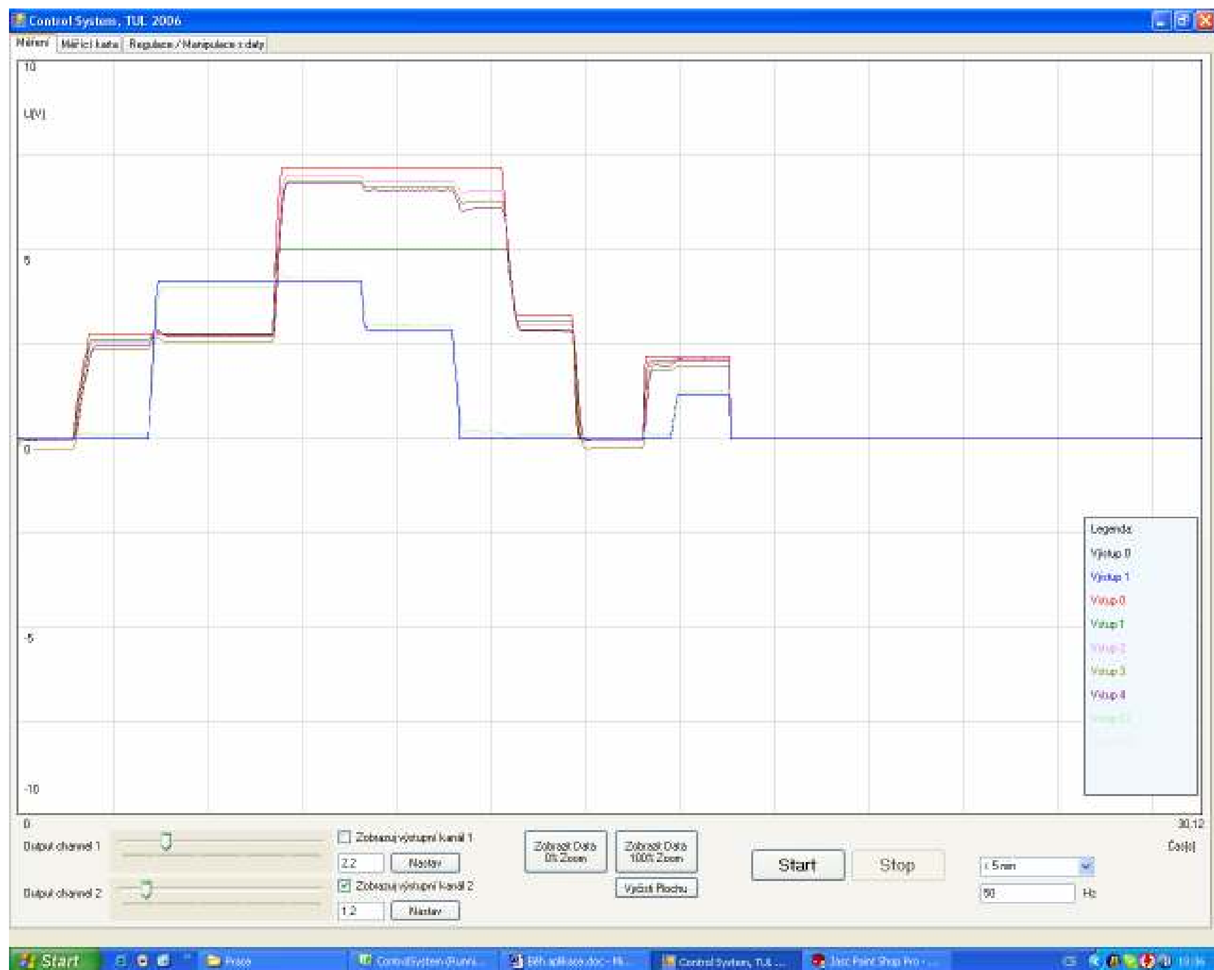
na prvním řádku definujeme typ metody – a sice že se jedná o privátní metodu a že nevrací žádnou hodnotu. Důležitá je formulace (CheckBox)sender – toto je jediný způsob, jak zjistit, která komponenta vyvolala tuto událost. Vzhledem k tomu, že komponenty nejsou při návrhu aplikace viditelné a nemají své jméno, nemůžeme přistupovat k jejich atributům. Zde bylo třeba zjistit, který checkBox byl zaškrtnut – tedy který kanál chce uživatel sledovat. Jedinou možností jak toto zjistit, je právě přes objekt sender. V jeho atributu Name je uloženo jméno komponenty. V našem případě toto jméno obsahuje číslo zvoleného kanálu – číslo vyextrahujeme metodou string.Remove(). Poté nám již nic nebrání v konfiguraci příslušného portu.

6.5 Konfigurace portů

Zvolená konfigurace portů se ukládá do dvourozměrného pole cardConfig velikosti počet kanálů x 2. V první buňce jsou nastaveny údaje o aktivitě portů – zda je bude aplikace aktivně sledovat či nikoli – konkrétně: 0 – neaktivní, 1 – aktivní. Druhá buňka obsahuje zvolený rozsah daného kanálu: 0 - +-10V, 1 - +-5V. Jakmile je karta nakonfigurována, může uživatel spustit měřící a regulační proces.

6.6 Měření

Samotný proces a sledování měření je kontrolováno z prvního panelu formuláře – „Měření“. Největší plochu zabírá komponenta panel na níž se vykresluje graf. Uvnitř této je přítomen další panel, se zobrazením barevné legendy ke grafu. Nápisy příslušných vstupů / výstupů jsou tvořeny dynamicky a barevně odpovídají příslušnému průběhu na grafu. Tento panel lze po kliknutí přemístit, pokud překáží v zobrazení grafu.



Obrázek č.13 ControlSystem – záložka „Měření“

Pod grafem jsou přítomny tlačítka pro ovládání měřícího procesu. Nejdůležitější z hlediska správného měření je volba předpokládané délky měření a příslušná vzorkovací frekvence. Po zvážení a analýze přítomných úloh v laboratoři TK4 byly zvoleny následující časové intervaly a s tím související maximální povolené hodnoty vzorkovacích frekvencí:

< 5 minut ... 1 - 1000 Hz

5-10 minut ... 1 - 500Hz

10-60 minut ... 1-10Hz

Nejvyšší limit 60ti minut je dostatečný i pro nejpomalejší úlohy typu průtokový ohřívač či fén. Taktéž vzorkovací frekvence 10Hz umožní sledovat chování systému s dostatečnou přesností. Pro záznam těchto dějů stačí i vzorkování jednou za vteřinu.

Naopak nejrychlejší děje – motory propojené pružným členem – ani zdaleka nevyužijí možností aplikace. Po praktických zkouškách bylo dosaženo uspokojujících výsledků měření s frekvencemi kolem 50Hz. Z tohoto důvodu je nejvyšší nastavitelná frekvence vzorkování 1 kHz a nikoli například 10 kHz – i když by na to hardware počítače stále stačil. Snížila by se tím přesnost měření – a mohlo by dojít k porušení ekvidistantnosti vzorků. Proto se nedoporučuje sledovat systém zbytečně vysokými frekvencemi. Uveďme si příklad podobné nepraktičnosti, která s sebou nese řadu dalších na první pohled nezřetelných aspektů. Po dobu 5ti minut byl systém sledován vzorkovací frekvencí 1kHz ze všech kanálů – konkrétně 16ti. Počet získaných vzorků se rovnal téměř 300 000. V nejúspornějším formátu csv zabíral soubor na pevném disku přes 40 MB. Program MS Excel nebyl schopen načíst více než prvních 65 536 hodnot. Získaná informativní hodnota byla rovna měření se vzorkovací frekvencí 50 Hz. Proto je na uživateli, aby zvážil a vhodně nastavil ideální poměr mezi délkou měření a počtem získaných vzorků.

6.7 Spuštění měřícího procesu

Máme-li otevřený komunikační kanál s měřící kartou, nastavené sledované porty a vybranou vzorkovací frekvenci, můžeme začít se sledováním systému. Začíná se tlačítkem Start. Po jeho stisku dojde k nastavení atributu Enabled několika komponent na hodnotu false. Aplikace se tím chrání před nevhodnými zásahy uživatele během měření, které by mohly způsobit pád aplikace. V dalším kroku se nakonfigurují porty na kartě, dle uživatelem specifikovaného konfiguračního pole cardConfig. Nastaví se též všechny proměnné potřebné pro správné vykreslování grafů. Ze zvoleného časového úseku a vzorkovací frekvence se vyhodnotí předpokládaný počet vzorků. Pokud tento překračuje šířku vykreslované plochy, vypočte se poměrové číslo, díky kterému se bude vykreslovat jen n-tý vzorek.

Poté dojde ke spuštění nového threadu, který zpracovává metodu *GetData* v přesných časových intervalech, odpovídajících vzorkovací periodě. Poslední iniciací je spuštění komponenty *timer*, která zajišťuje vykreslování grafů během měření. Toto vykreslování má informativní charakter a velmi zatěžuje systémové prostředky počítače. Původně bylo zamýšleno vykreslování v reálném čase zcela vypustit, ale je třeba dát uživateli vizuální kontrolu nad probíhajícím dějem a případný zásah do něj.

6.8 Zastavení měření

Uživatel může kdykoli přerušit měření stiskem tlačítka Stop. Tato vlastnost byla hlavním podnětem k tvorbě více vlákenné aplikace. Před tímto přístupem byl přesný časovač spuštěn v hlavním vlákně aplikace. To bohužel zcela zahltilo veškeré systémové prostředky – a znemožnilo ošetřování veškerých událostí nad ostatními komponentami. Narozdíl od časovače Windows neprobíhalo časování automaticky v externím vlákně. Tedy ani tlačítko Stop nešlo zmáčknout a aplikace se sama zastavila až po naměření všech hodnot. Podobné chování bylo neakceptovatelné – ošetření zmáčknutí tlačítek za běhu přesného časovače bylo vitální pro použitelnost celé aplikace.

Po stisku tlačítka dojde ke zpětné aktivaci několika komponent, které byly z bezpečnostních důvodů deaktivovány. Následovně se zkontroluje, zda je aktivní paralelní vlákno a dojde k jeho ukončení. Taktéž se ukončí časovač Windows, určený pro vykreslování grafu v reálném čase.

Manuální ukončení vlákna s přesným časovačem bylo nutné kvůli špatné práci garbage collectoru platformy .NET. Po stisku tlačítka nedošlo k automatickému uvolnění proměnných z paměti počítače a vlákno dále běželo – využívajíce veškeré systémové prostředky znemožňovalo další práci.

6.9 Metoda GetData

Jedná se o stěžejní metodu programu, ve které dochází k načítání dat ze vstupů karty a nastavování jejích výstupů. Metoda se spouští v paralelním threadu, pro dosažení požadované přesnosti vzorkování. Spouštěcí příklad vypadá následovně:

```
thread = new Thread(GetData);
```

Samotná metoda je implementována takto (jedná se jen o princip – není zde uveden celý kód metody):

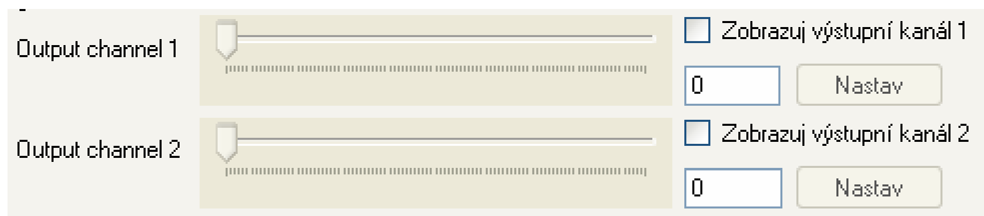
```
public void GetData()
  {
    int periodCount = 0;
      while (!terminate)
        {
if (numOfSamples <= data.GetLength(1))
{
              perfTimer.Start();
              dac.AnalogInputRead(0, 16, false, actual);
              outData[0, numOfSamples] = output1 * 0.002445;
              outData[1, numOfSamples] = output2 * 0.002445;
```

```
for (int j = 0; j < dac.Features.AnalogInputSingleEndChannels; j++)
  {
    if (cardConfig[0, j] == 1)
      {
        if (cardConfig[1, j] == 0)
         {
       data[j, numOfSamples] = (double)(actual[j] - 2048) * 0.00489;
         }
       else
        {
      data[j, numOfSamples] = (double)(actual[j] - 2048) * 0.002445;
        }
     }
  }
 numOfSamples += 1;
 perfTimer.Stop();
 while (perfTimer.cycleTime < (perfTimer.period - 0.00001))
   {
      perfTimer.Stop();
   }
 }
 else
  {
    terminate = true;
    thread.Abort();     }}}
```

Celý cyklus probíhá tak dlouho, dokud není ukončovací booleovská proměnná *terminate* nastavena na true. V dalším kroku se zkontroluje, zda již nedošlo k přetečení datového pole a nastartuje se časovač. Přečte se hodnota ze všech kanálů karty v jednom kroku – díky funkci *MAIRead()* – multiple analog read - knihovny *AdvantechDAC.dll*. Následně se uloží naměřené hodnoty do datového pole – na základě konfiguračního vektoru karty – neukládají se vstupy, o něž nemá uživatel zájem. Nakonec dojde k ukončení časové smyčky, opakováním příkazu tak dlouho, dokud není hodnota časového intervalu (periody) rovna požadovanému. Kromě čtení dat naměřených na zvolených vstupních portech dochází k nastavování výstupních hodnot dle zvolené funkce.

6.10 Nastavení hodnot výstupních portů

Poněvadž některé z karet Advantech obsahují i výstupní porty, bylo potřeba zpracovat jejich ovládání a správné nastavování. Po prozkoumání dostupných karet Advantech bylo zjištěno, že většina obsahuje buď právě dva výstupní porty nebo žádné. Konkrétně z PCI karet obsahují 2 analogové výstupy jen karty PCI-1711 a PCI-1712. Nejjednodušší nastavování výstupního napětí probíhá na listu „*Měření*" aplikace ControlSystem. Jedná se o přímou manuální kontrolu nad hodnotou výstupního napětí.



Obrázek č.14 Panel manuálního nastavování výstupního napětí

Rychlé změny výstupního napětí může uživatel provádět pomocí šoupátek na liště. Přesnost nastavení je 1/10 V. Nevýhodou je fakt, že nastavování napětí probíhá během ošetřování události *trackBar.Scroll*. Při pomalejším posunu šoupátka dojde k vícenásobnému zpracování události a tím pádem k několikanásobnému nastavení výstupu. Nemůžeme tedy zaručit, že dojde ke skokové změně a nikoli postupnému nárůstu výstupního napětí. Chceme-li tohoto docílit a zároveň zvýšit přesnost nastavení výstupního napětí, provedeme tak přes *textBox* a tlačítko *Nastav*. Položky „*Zobrazuj výstupní kanál*"slouží pouze k vizualizaci průběhů výstupních kanálů na panelu grafu. Nesouvisí nijak s ukládáním výstupních dat do datového pole.

6.11 Programové zpracování výstupů

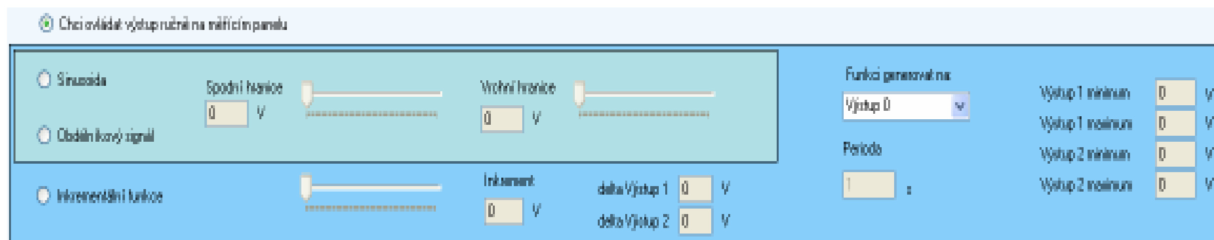
Již během navázání komunikace s kartou dojde ke konfiguraci obou výstupních portů příkazem *DAC.AnalogOutputConfig()*. Předávanými parametry jsou číslo výstupního kanálu, booleovská proměnná specifikující interní / externí referenci, minimální a maximální hodnotu referenčního napětí. Pokud na kartě nejsou výstupní kanály přítomny, k jejich nastavení samozřejmě nedojde a ovládání nastavovacích komponent výstupu nebude mít žádný vliv. Při události, která vyvolá změnu nastavení výstupního napětí dojde k přečtení požadované hodnoty v dekadické formě – buď z polohy šoupátka na liště nebo z hodnoty v textovém poli. Tato hodnota se musí převést na hodnotu D/A převodníku. Vzhledem k faktu, že se jedná o 12ti bitový převodník máme možnost nastavit výstup na 4096 možných úrovní. Převod a zápis probíhá v tomto skriptu:

```
output1 = (short)(((double)sliderOut1.Value / 10)/ 0.002445);
dac.AnalogOutputWrite(0, output1);
```

Konstanta 0.002445 je poměrové číslo 1:4096. Příkaz *DAC.AnalogOutputWrite()* provede zápis hodnoty na specifikovaný výstupní port.

6.12 Definice periodických výstupních funkcí

Uživatel má i jinou možnost specifikace výstupů, než jen pomocí manuálních zásahů do aplikace. Pro volbu výstupních funkcí se přesuneme na záložku „*Regulace / Ukládání dat*“, kde bude naším středem zájmu panel se čtyřmi přepínatelnými komponentami typu *radioButton.*



Obrázek č.15 Panel pro nastavování výstupních funkcí

Pro ruční ovládání na panelu „*Měření*“ necháme označenou první položku. V opačném případě si uživatel může vybrat ze tří nabízených funkcí – Sinusoida, Obdélníkový signál či Inkrementální funkce. Funkce bude generována na výstupu, který uživatel zvolí v rolovacím menu – k dispozici jsou buď oba kanály samostatně, nebo oba najednou. Po volbě jedné z nabízených funkcí je zablokována možnost ručního ovládání výstupu. Nyní se již zaměřme na jednotlivé výstupní funkce.

6.11.1 Sinusoida – známá, často používaná goniometrická funkce. Výstup je definován následovně:

```
output1=(short)(2048+409.6*Math.Sin(numOfSamples*2*Math.PI/periodSamples));
dac.AnalogOutputWrite(0, output1);
```

Zápis je prováděn v metodě *GetData*. Úhel funkce sinus je určen poměrem současného počtu vzorků *numOfSamples* ku počtu vzorků na jednu uživatelem definovanou periodu *periodSamples*. Perioda sinu se ručně nastaví v textovém poli – perioda se udává v sekundách. Aplikace přepočítá čas periody na počet cyklů časovače po stisku tlačítka *Start*. Nastavením šoupátek lišt „*Spodní hranice*“ a „Vrchní hranice“ stanoví uživatel amplitudu a posun signálu do oblasti, kde si přeje signál generovat.

6.11.2 Obdélníkový signál – užitečný signál, určený pro sledování reakcí systému na skokové změny. Generuje se obdobně jako sinusoida v metodě *GetData*. Uživatelem definovaná perioda je opět převedena na počet cyklů časovače vzhledem k vzorkovací periodě.

Nastavením spodní a vrchní hranice pomocí šoupátek zvolíme hodnoty, mezi kterými bude signál oscilovat. Skript generující funkci je v principu velmi jednoduchý:

```
if (periodCount >= periodSamples)
{  periodCount = 0;
   if ((output1 == 0) || (output1 == output1Min))
   {output1 = output1Max;
    dac.AnalogOutputWrite(0, output1);}
  else
   {output1 = output1Min;
    dac.AnalogOutputWrite(0, output1);}
  if ((output2 == 0) || (output2 == output2Min))
   {output2 = output2Max;
    dac.AnalogOutputWrite(1, output2);}
  else
   {output2 = output2Min;
    dac.AnalogOutputWrite(1, output2);}
}
else
{periodCount += 2;}
```

V polovině periody nastaví hodnotu výstupního napětí na opačnou hranici, než na které se aktuálně nachází. Tím je vygenerován jeden cyklus obdélníkového signálu v jedné periodě. Lze nastavit různé hranice pro oba výstupy a poté zároveň generovat průběhy na obou kanálech.

6.11.3 Inkrementální funkce – Jednoduchá funkce periodicky zvyšující výstupní napětí o malý skok. Inkrement ve voltech zvolí uživatel buď šoupátkem na liště, nebo vepsáním hodnoty do textového pole. Lze nastavit rozdílné inkrementy pro oba kanály. Perioda, po které dojde k navýšení napětí o schodek se nastavuje ve stejném poli, jako perioda funkcí *Sinusoida* a *Obdélník*. Po dosažení maximálního výstupního napětí +10 Voltů dojde k deaktivaci funkce a hodnota se dále nezvyšuje. Skript je nenáročný na zpracování – nezpomaluje tedy běh aplikace:

```
if (periodCount >= periodSamples)
  {periodCount = 0;
   output1 = (short)(output1 + out1Increment);
   if (output1 > 4089)
     { output1 = 4089;}
   dac.AnalogOutputWrite(0, output1);
   output2 = (short)(output2 + out2Increment);
   if (output2 > 4089)
     {output2 = 4089;}
   dac.AnalogOutputWrite(1, output2);
  }
else
  { periodCount += 1;}
```

Veškeré nastavování výstupních hodnot a jejich zápis na výstupní porty probíhá na začátku metody *GetData*, kde se pomocí příkazů *if* rozhoduje o funkci, která bude vygenerována. V celé aplikaci bylo dbáno na rychlost zpracování – proto dochází k ohlídání veškerých důležitých událostí podmínkami *if* a nikoli pomocí dnes velmi propagovaných výjimek *exceptions*. Skrytou vadou tohoto přístupu je jeho rychlost. Ošetření výjimek je totiž extrém pomalé. Na platformě .NET je bohužel řada operací velmi málo transparentní, proto nelze s určitostí říci, proč jsou tyto operace natolik pomalé. Při normálním běhu programu si uživatel ani nevšimne, že byly použito výjimek a nikoli například příkazů *if*. Ovšem v aplikaci ControlSystem došlo k desynchronizaci měření při použití vysokých vzorkovacích frekvencí.

7. Časování

Pro účely aplikace musela být vyřešena otázka přesného časování v operačním systému MS Windows. Vzhledem k očekávanému použití vysokých vzorkovacích frekvencí (až 1 kHz) v programu nebylo možné použít standardní časovač systému Windows.

Standardní časovač Windows ,vytvořený funkcí *setTimer*, periodicky zasílá zprávu *WM_Timer* proceduře systému, nebo přímo volá specializovanou funkci *TimerProc.* Tyto časovače jsou velmi jednoduché k používání, ale jejich rozlišovací schopnost je pro účely této práce nedostatečná a opakovaná přesnost je zcela nevyhovující. V ideálním případě lze dosáhnout rozlišení 55ms, tedy lehce nad 18 Hz. Ve většině případů se však nedostaneme pod 100ms, což odpovídá frekvenci 10 Hz – to je 100x méně, než je zapotřebí pro přesné měření probíhajících dějů v očekávaných systémech, na kterých bude aplikace užívána.

Přesnější verzí těchto časovačů je tzv. multimediální časovač. S jeho využitím lze dosáhnout periody volání 10ms při zachování vysoké přesnosti. Pod tuto hranici se lze dostat – teoreticky až na 1ms, avšak na úkor snížené přesnosti.

Žádný z těchto přístupů nelze použít, chceme-li zachovat ekvidistantnost měřených vzorků. Proto bylo potřeba zvolit zcela jiný přístup. Windows obsahuje vysoce výkonný časovač s rozlišením v řádu desítek nanosekund. V reálných podmínkách dnešních procesorů lze dosáhnout přesnosti v řádu jednotek mikrosekund. Jedná se o API funkci *QueryPerformanceCounter* a Q*ueryPerformanceFrequency*. Zavoláním příkazu Q*ueryPerformanceCounter* získáme aktuální hodnotu přesného časovače ve formátu 64bit

integeru. Voláním QueryPerformanceFrequency získáme hodnotu 3579545 – z čehož vypočteme, že nejkratší měřitelný okamžik je 1/3579545 – tedy 0.28 mikrosekundy – toto však není prakticky dosažitelné – kvůli času, který zabere samotné volání funkce časovače. Bylo provedeno hledání původu tohoto zvláštního čísla 3579545. Příkaz QueryPerformanceFrequency byl spuštěn na celkem 5ti počítačích s rozdílnými procesory – konkrétně se jednalo o desktop PC s procesorem AMD Athlon XP@2500MHz, notebook s procesorem Pentium M 740@1730MHz, notebook s procesorem Core Duo T2400@1830MHz a nakonec na dvou průmyslových počítačích – jeden s procesorem Pentium M 780@2260MHz a druhý s procesorem Pentium 4 D 945. Na všech bylo systémem vráceno právě číslo 3579545. Z toho bylo usouzeno na fakt, že toto číslo naprosto nesouvisí s typem a frekvencí daného procesoru, jak by se na první pohled mohlo zdát. Ve skutečnosti se jedná o frekvenci ACPI (Advanced configuration and Power Interface) čítače. Otevřená specifikace, která definuje řízení spotřeby u celé řady mobilních a stolních počítačů, serverů a periferních zařízení. Rozhraní ACPI je základem iniciativy OnNow, která umožňuje vyrábět počítače, jež lze spustit pomocí tlačítka na klávesnici. Použití rozhraní ACPI je nezbytné k plnému využití funkcí řízení spotřeby a systému Plug and Play. Z toho plyne, že ACPI je k dispozici i při spánkovém módu počítače a stav procesoru nemá vliv na jeho přesnost.

Samotné měření nejkratšího dosažitelného času bylo prováděno na dvou platformách. V prvním případě se jedná o desktop PC s procesorem AMD Athlon XP@2500MHz – zde bylo dosaženo nejkratšího času 5 mikrosekund. V druhém případě šlo o dvoujádrový procesor Intel CoreDuo T2400@1830MHz – nejkratší dosažený čas osciloval mezi 1,5 a 2,5 mikrosekundy. Dosažené časy se výrazně neměnily ani při plném vytížení procesorů externími aplikacemi. Nebyl problém udržet opakovanou přesnost do +- 5 mikrosekund.

Pro bližší představu je princip přesné časové smyčky následující:

```
using System.Runtime.InteropServices;
long startTime, endTime;
double ellapsedTime;
double requiredTime; //požadovaná délka smyčky
QueryPerformanceFrequency(out freq);
QueryPerformanceTimer(out startTime);
        // samotný kód běžící v přesné časové smyčce – v našem případě se jedná o
        čtení / zapisování vstupů a výstupů.
QueryPerformanceCounter(out endTime);
```

```
ellapsedTime = (endTime – startTime)/freq;//výsledný čas je v sekundách
while (ellapsedTime <= requiredTime)
        {       QueryPerformanceCounter(out endTime);
                ellapsedTime = (endTime – startTime)/freq;}
```

Tato funkce je pro dané účely optimální – z hlediska poměru náročnost/přesnost. Podle nápovědy MSDN by měl tento způsob časování být obsažen v třídě *stopWatch* v jmenném prostoru *System.Diagnostics* Avšak po důkladném proměření nebyla použita. Z neznámých důvodů trvala smyčka přes 17 mikrosekund – zřejmě je to způsobeno voláním přídavných funkcí a kontrolami v třídě *stopWatch*.

Poslední možností jak měřit přesný čas je nejpřesnější a též nejkomplikovanější. Jedná se o čítač procesorových časových známek (time stamp) získaný provedením procesorové instrukce RDTSC. Vrácená hodnota je počet proběhlých period procesorových hodin. Problémem této metody je zjištění přesné pracovní frekvence procesoru a z toho plynoucí potíž s výpočtem uplynulé doby v měřitelném čase.

7.1 Výpočetní náročnost časování

Po realizaci přesného časovače se vyskytl problém s extrémní náročností tohoto procesu – a tedy jeho nepoužitelnost. Po spuštění časovače bylo 100% procesorového času věnováno dodržování přesné časové smyčky. Systém nereagoval na žádné vnější podněty (mačkání tlačítek obsluhy) až do konce měření. To nebylo přípustné – uživatel musí mít možnost proces kdykoli přerušit či zastavit. Proto bylo nutné zpracovávat v paralelním vlákně aplikace. Aplikace po zmáčknutí tlačítka START založí nové vlákno, ve kterém probíhá zpracování vstupně-výstupních dat v přesné časové smyčce. Nové vlákno běží na pozadí aplikace a nárokuje si výpočetní čas procesoru stejně jako hlavní vlákno aplikace. Uživateli se jeví, že vše probíhá současně. Na vícejádrovém systému je nové vlákno spuštěno na druhém nevyužitém jádře a jeho výkon není negativně ovlivněn vláknem hlavní aplikace. Aplikace byla testována na běžném jednojádrovém procesoru s podporou Hyperthreading technologie. Vliv běhu více vláken na přesnost časování nebyl měřitelný.
Ve zjednodušené formě vypadá mechanismus následovně:

```
using System.Threading;
Thread thread;
```

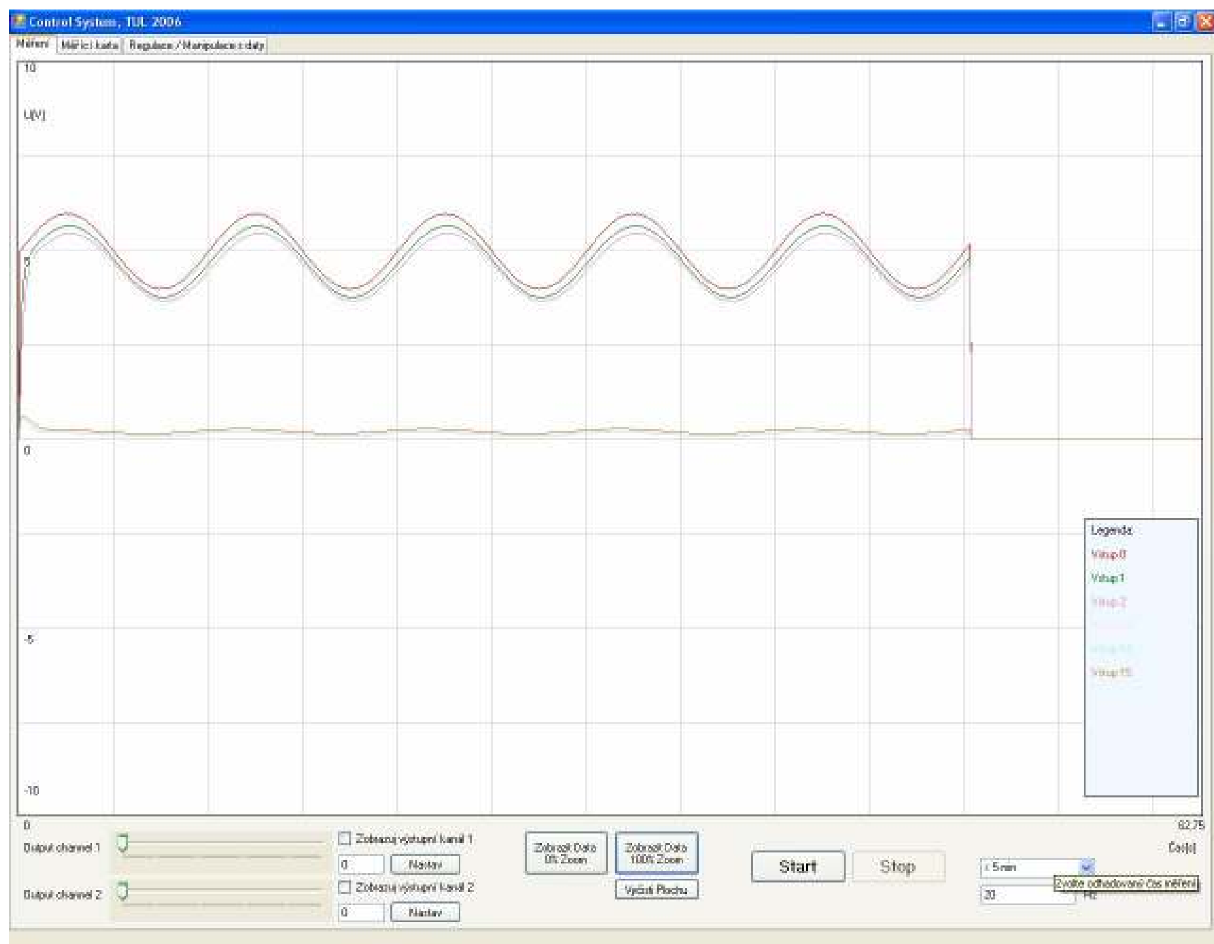
```
private void GetData()
        //spuštění časovače, načítání dat po uplynutí periody
private void startButton_click(object sender, EventArgs e)
        {       thread = new thread(GetData);
                thread.Start();}
private void stopButton_click(object sender, EventArgs e)
        {       if (thread.isAlive)
                thread.Abortt();}
```

Během programování se vyskytl problém s nesprávně fungujícím garbage collectorem platformy .NET. Správné fungování by mělo spočívat v tom, že po ukončení běhu aplikace se automaticky ukončí všechna běžící vlákna a z paměti se odstraní veškeré nadefinované proměnné. Právě běžící vlákno časovače však garbage collectorem nebylo ukončeno a zůstalo v plném běhu na pozadí. Vzhledem k jeho charakteru si ihned po ukončení aplikace začalo nárokovat plných 100% procesorového času a celý systém se výrazně zpomalil. Proto bylo nutné přidat do aplikace interní kontrolu běhu vláken – booleovská proměnná *thread.isAlive* – a jejich korektní ukončování.

8. Zobrazovací funkce

Vykreslování v jazyce C# působí na první pohled poněkud těžkopádně, avšak po zjištění základních zákonitostí není problém jakékoli grafy zobrazovat.

Pokud se nebudeme zabývat vykreslováním pomocí *OpenGL* knihovny, musíme použít třídu *PaintEventArgs.Graphics* a její metody pro zobrazení námi definovaných tvarů. V našem případě budeme vykreslovat jen krátké úsečky příkazem *Graphics.drawLine*(). Atributy předávané této funkci jsou 2 objekty typu *Point* a jeden typu *Pen*. Objekt typu *Point* lze nahradit dvěma *Integery*. *Point* je struktura obsahující dvě proměnné typu *integer* – x-ovou a y-ovou souřadnici bodu. *Pen* je též struktura obsahující objekt typu *Color* udávající barvu „tužky“ a integer udávající tloušťku vykreslované čáry. Logicky jsou předávanými atributy počáteční bod, koncový bod a typ čáry, kterou se mají body spojit. V aplikaci ControlSystem probíhá vykreslování na prvním listu „*Měření*“ komponenty *tabControl*.



Obrázek č.16 ControlSystem – záložka „Měření“

Největší plochu zabírá panel, na nějž se vykresluje průběh sledovaného děje. Pomocný panel *Legenda* slouží k lepší orientaci v grafu, je-li přítomno více průběhů. Vykreslování probíhá během události Windows *timer.Tick*. Na začátku se naposledy uložený index vzorku dat *numOfSamoples* uloží do lokální proměnné. Tímto se zabrání možným konfliktům při použití vyšších vzorkovacích frekvencí. Vzhledem k tomu, že vykreslování je hardwarově náročné a trvá určitý časový interval, hrozila by situace, že během vykreslování dojde k získání a uložení dalších vzorků dat. Tedy pro prvních několik kanálů bychom vykreslili graf v čase *t*, a pro další vzorky až v čase *t+k*. Uložením do lokální proměnné, která se změní až při příští události *timer.Tick* tomuto zamezíme. Před začátkem vykreslování je podstatné ještě nastavení poměrového čísla vykreslování, uloženého v proměnné *ratio*. To ze zvolené frekvence a času měření vypočte, zda bude celkový počet vzorků menší či větší než je šířka vykreslované plochy. V prvním případě je číslo ponecháno na hodnotě 1 a vykresluje se každá naměřená hodnota. V případě opačném dojde k výpočtu čísla následovně:

```
measureLength = 300 * Convert.ToInt32(selectedPeriod.Text);
ratio = Math.Round((double)(measureLength / drawPanel.Width));
```

Hodnota *measureLength* závisí na zvolené délce měření a frekvenci. Poměr *ratio* je dán vydělením tohoto čísla šířkou vykreslovací plochy. Vzhledem k tomu, že se obě konstanty po dobu měření nemění, inicializují se již po stisku tlačítka *Start*. Je tedy zbytečné, aby se znovu počítaly při každé nové události *timer.Tick*.

Vykreslení všech uživatelem zvolených průběhů je provedeno v následujícím skriptu:

```
for (int j = 0; j < dac.Features.AnalogInputSingleEndChannels; j++)
{
if (cardConfig[0, j] == 1)
{
drawPen = new Pen(drawColor[j + 2], 1);
startPoint = startPtCol[j];
endPoint = startPoint;
endPoint.X = endPoint.X + 1;
endPoint.Y = 400 - (int)(data[j, drawNumberLast + (int)(ratio * i)] * 40);
startPtCol[j] = endPoint;
drawGraph.DrawLine(drawPen, startPoint, endPoint);}}
```

Pokud je kanál zvolen uživatelem, zvolí se příslušná barva z předem definované kolekce barev *drawColor*, nastaví se počáteční *startPoint* a koncový *endPoint* bod a provede se vykreslení *DrawLine(drawPen, startPoint, endPoint).* Problém je v dynamicky zjišťovaném počtu vstupních kanálů a sekvenčním přístupu k vykreslování. Chceme v jednom kroku vykreslit data z více kanálů – musíme tedy opakovaně zavolat funkci *drawLine* s rozdílnými hodnotami *startPoint, endPoint a drawPen*. Jeden možný přístup je v definici například 32 různých proměnných typu *Point* pro uchovávání souřadnic bodů rozdílných kanálů se jmény rozlišenými např. pořadovými čísly na konci proměnné. Z programátorského hlediska je tato cesta nepřípustná a extrémně náročná na údržbu a ošetřování za běhu programu. Proto byl zvolen jiný přístup. V kolekci typů *Point* pojmenované *startPtCol* se vytvoří daný počet položek dle počtu kanálů dynamicky. Každá uchovává stav objektu typu *Point* z minulého kroku, tedy máme uchovány výchozí body pro kreslení v kroku následujícím. Na konci cyklu příslušné položce opět přiřadíme aktuální stav – tedy bod *endPoint.* Přístup k jednotlivým objektům se provádí přes indexy, odpovídající číslu vstupního kanálu.

8.1 Frekvence a přesnost zobrazování

Frekvence časovače určeného pro zobrazování během měření je implicitně nastavena na 2Hz – tedy 2 obnovení obrazu za vteřinu. Byl zvolen kompromis mezi výpočetní

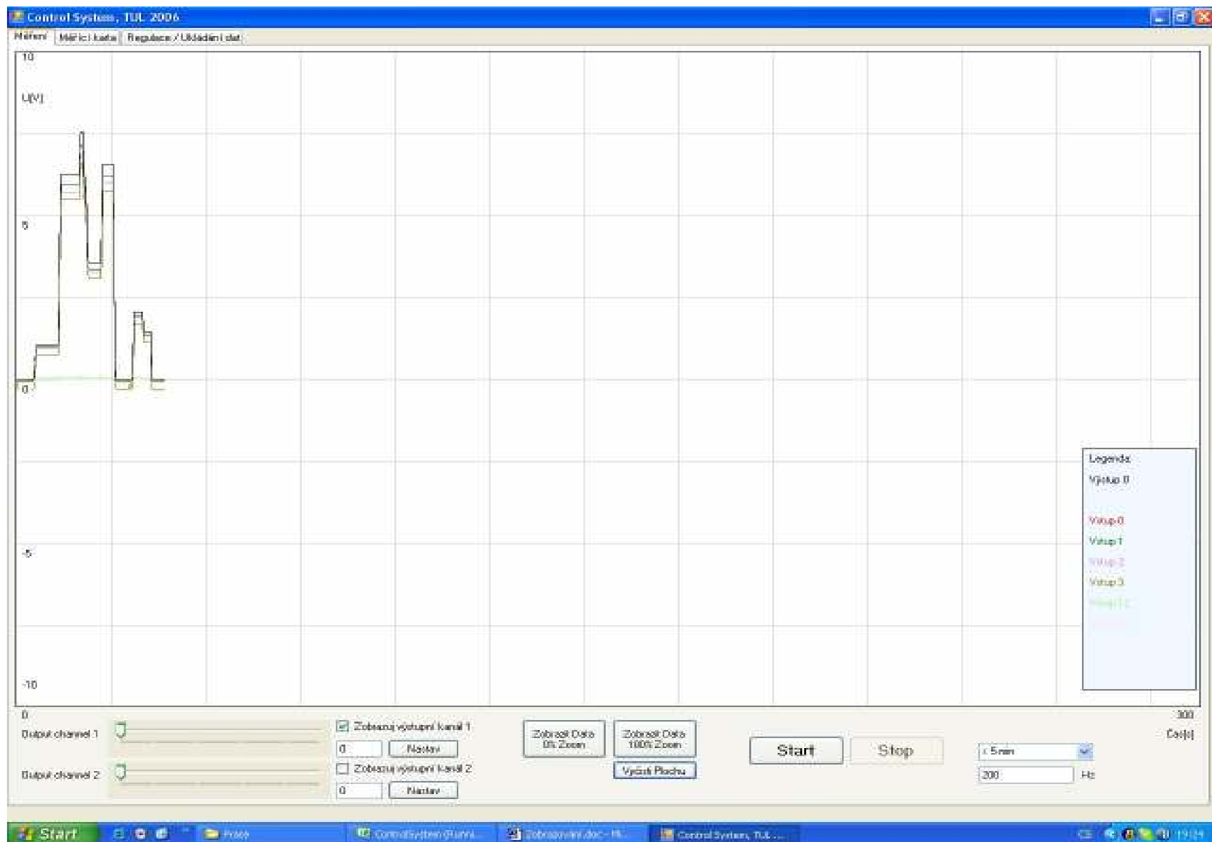
náročností a praktickou použitelností aplikace. Tato byť nízká frekvence stačí i pro sledování rychlých dějů a změn a nenamáhá zbytečně procesor.

Vykreslování během měřícího procesu je pouze informativního charakteru – nevykreslují se všechny vzorky, pokud je předpokládaný počet vzorků vyšší než šířka panelu v pixelech. Během události *timer.Tick* dojde k extrakci naposled změřených vzorků a vykreslí se úsečka mezi těmito a koncovými body naposled provedeného vykreslování. Z hlediska přesnosti by jistě bylo zapotřebí provést výpočet průměrné hodnoty ze všech vzorků naměřených během periody vykreslovacího časovače. Avšak to je výpočetně náročné a má to negativní vliv na přesnost paralelně běžícího časovače měření. Proto byla zvolena metoda méně přesná, ale zároveň nezatěžující procesor počítače.

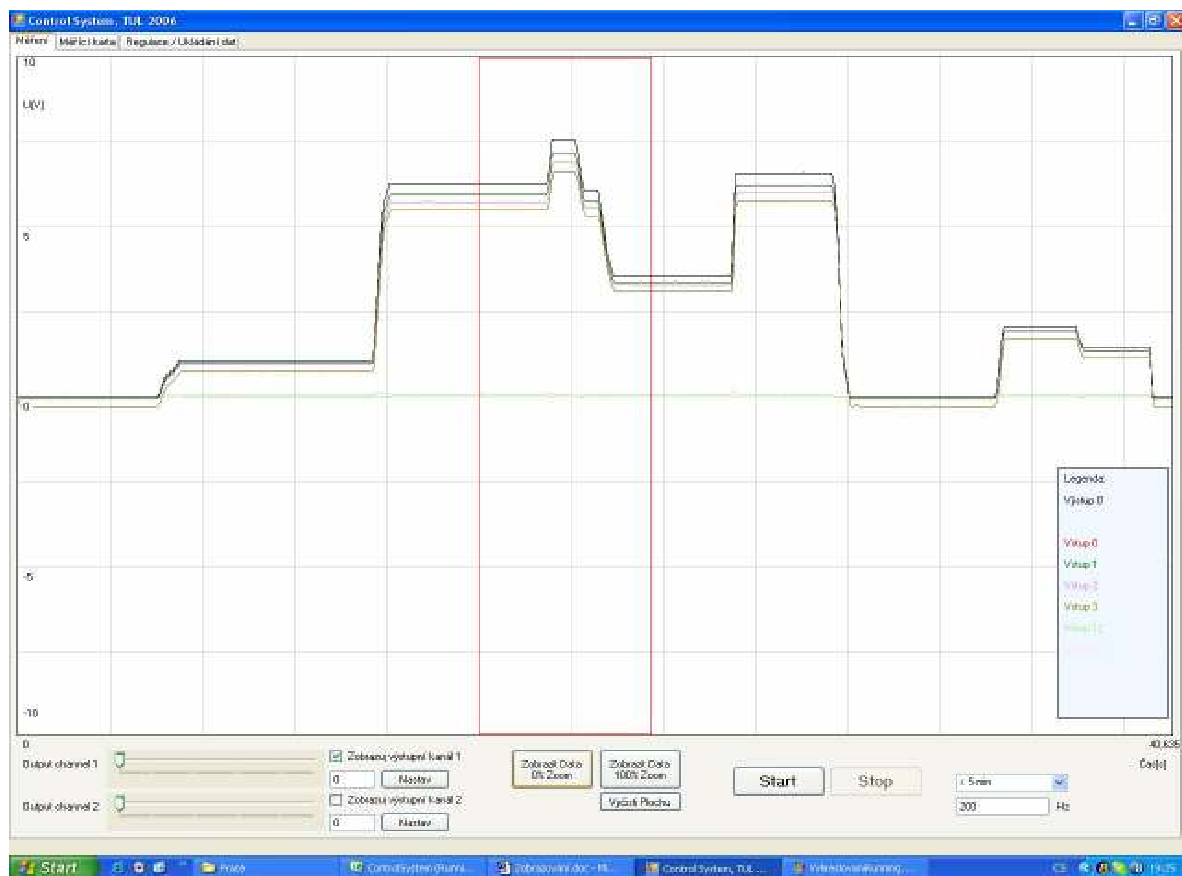
Po ukončení měření má uživatel možnost vykreslit data ve skutečném měřítku bez vynechaných hodnot. Pod panelem grafu se nachází dvě tlačítka určená pro opětovné vykreslení datových průběhů. První nese označení „*Zobrazit data 0% Zoom*“. Slouží k zobrazení všech naměřených dat na jedné obrazovce – pro celkový přehled průběhu měření. Jak je patrno, i zde dojde k vykreslení jen části dat, pokud je počet naměřených dat vyšší než šířka vykreslovacího panelu. Poměr zkreslení však může být nižší než u vykreslování během měření – poměr se vypočítává ze skutečného počtu naměřených dat a nikoli jen odhadovaného. Taktéž se změní popisy os – aby byl zachován poměr mezi vykreslovanými vzorky a časovou osou grafu.

Daleko zajímavější je druhá možnost – a sice vykreslení všech vzorků, bod po bodu. Slouží pro to tlačítko „Zobrazit data 100% Zoom“. K docílení tohoto však bylo potřeba upravit samotný vykreslovací panel *drawGraph*. Jeho atribut *AutoScroll* byl nastaven na hodnotu *true*, pro zobrazení posuvné lišty v případě, že komponenty daného kontejneru (v našem případě panelu) přesáhnou jeho plochu. Během testování však bylo zjištěno, že vykreslování grafu pomocí metody G*raphics.drawLine* nepřidává komponenty typu *Point*, ale jen vykresluje statický obraz. Proto k docílení posuvu muselo být použitu triku s neviditelnou komponentou – konkrétně typu *label*. Po stisku tlačítka se poměr vykreslování nastaví na hodnotu 1 – aby došlo k vykreslení všech bodů. Poté se dynamicky vytvoří komponenta typu *label* na souřadnicích odpovídajících počtu vzorků mínus šířka samotného labelu. Tím dojde k automatické inicializaci a zobrazení posuvné lišty na grafu. Při jejím posuvu je vyvolána událost *panel.Paint*, ve které dojde ke zpracování data a vykreslení příslušné části grafu. Byla

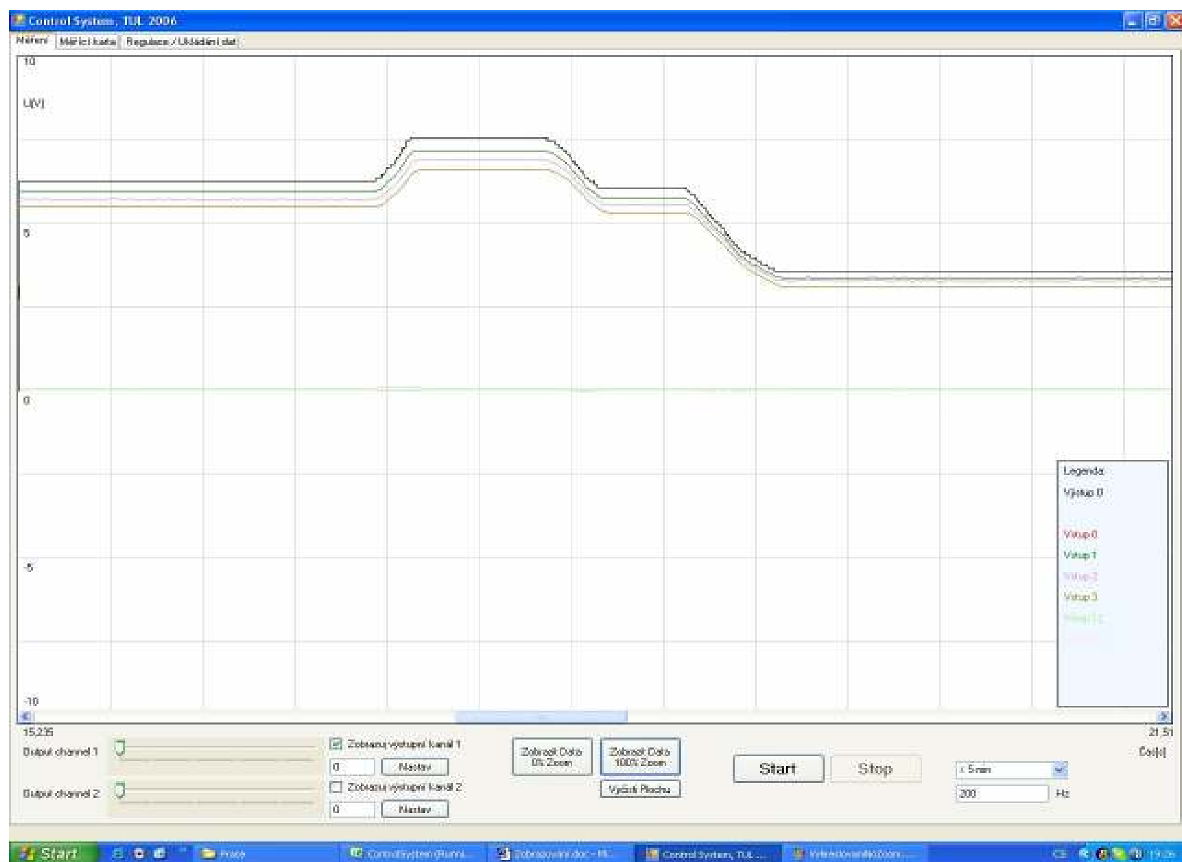
vyzkoušena i možnost vykreslování grafu na událost *panel.Scroll*, avšak docházelo k většímu zpomalování systému, než při události *Paint* – událost byla zpracovávána častěji než při *Paint* a tím pádem od toho bylo ustoupeno. Ani již zvolená událost není zcela ideální a při posunu posuvné lišty dochází k problikávání grafu a je vidět jeho postupné dokreslování. Není to však závažný problém a práce s grafem je i přesto pohodlná. Při posuvu lišty dochází ke změně atributu *panel.AutoScroll.Position* – obsahující dvě souřadnice – X a Y. Ty se mění v závislosti na tom, se kterou lištou posouváme. Vzhledem k naší aplikaci posuny v ose Y nejsou zapotřebí, a proto máme umožněny jen posuny v ose X. Po skončení posuvu lišty dojde k vygenerování události *panel.Paint*. Aplikace načte současnou polohu v pixelech a nastaví první vykreslovaný vzorek právě na tuto polohu. Jsme-li posunuti 2000 pixelů od začátku, dojde k vykreslení grafu od 2000 vzorku až do hodnoty 2000 + šířka okna grafu. Stejným způsobem dojde i ke změně popisu osy X. Hodnota posunu v pixelech – tedy pořadové číslo prvního zobrazeného vzorku - se vydělí zvolenou periodou a získáme čas nového začátku časové osy. Obdobně získáme i údaj o konci časového úseku, který uživatel právě sleduje. Nyní si v praxi ukážeme, rozdíly mezi jednotlivými zobrazeními.



Obrázek č.17 Graf zachycený během měření



Obrázek č.18 Zobrazení všech naměřených dat na jedné obrazovce

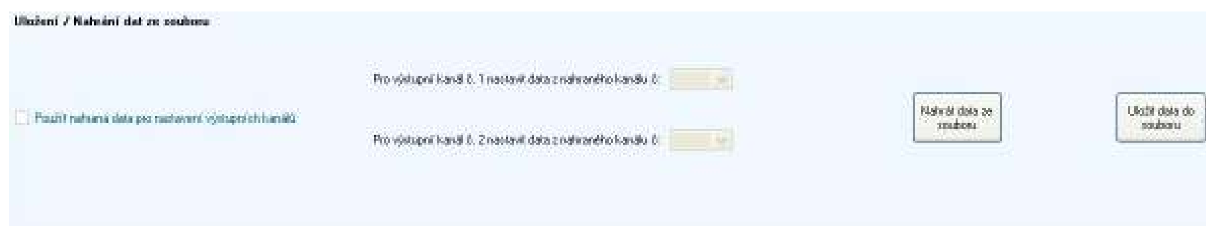


Obrázek č.19 Detailní zobrazení naměřených dat (červená oblast z předchozího grafu)

Graf znázorňuje data zaznamenávaná po dobu cca 40ti sekund se vzorkovací frekvencí 200Hz. Na prvním obrázku vidíme data zobrazená během měření. Graf je velmi malý a postrádá detail – je to způsobeno tím, že předpokládaná délka měření byla 300 sekund. Při daném vzorkování byl vykreslován jen každý 48mý vzorek naměřených dat. Druhý graf již zobrazuje kompletní naměřený průběh na jedné obrazovce. Bylo naměřeno cca 8100 vzorků. Vzhledem k šířce grafu v pixelech (cca 1200) je vykreslován každý 7mý vzorek – informativní hodnota grafu je již 6x vyšší. Na posledním obrázku vidíme detail oblasti grafu mezi 15ou a 21ní vteřinou děje. Zde je již vykreslován každý naměřený vzorek – pod grafem se objevila posouvací lišta umožňující prohlédnutí každého požadovaného úseku.

9. Datové výstupy

Ukládání a zpětné nahrávání dat probíhá na záložce „*Regulace / Manipulace s daty*“



Obrázek č.20 Panel pro ukládání / nahrávání dat ze souboru

Aplikace Control System ukládá naměřená data do dvourozměrného pole typu *double*. Data získaná z vstupních kanálů karty jsou ve formátu *short* – výstup z AD převodníku. To je však z hlediska dalšího uchování nevhodné. Ztrácíme tím informaci o zvoleném měřícím rozsahu – maximální hodnota vrácená z převodníku je rovna 4096 – tedy 2^{12} hodnot – odpovídající 12 bitovému převodníku. Chybí nám však doprovodná informace o maximální hodnotě naměřeného napětí, které toto číslo představuje. Z toho důvodu převádíme data ihned po naměření na typ *double* – a násobíme ho příslušným poměrem dle zvoleného měřícího rozsahu. Do pole ukládáme již hodnotu konkrétního napětí ve voltech v desetinném tvaru – např. 1,25 voltu.

Takto upravená data je možné po skončení měření uložit na pevný disk počítače. Do každého souboru jsou uloženy informace ze všech kanálů – i těch, na kterých nebylo prováděno měření. Důvod je prostý – nabízí nám plnou zpětnou kontrolu nad proběhlým dějem. V případě nezvyklého chování lze zkontrolovat všechny kanály, zda se na nějakém

nevyskytly anomálie, které by mohly mít za následek změnu chování pozorovaného děje. Po analýze dostupných typů výstupních souborů byly vybrány následující typy:

9.1 Textový soubor

Soubor s příponou *.txt* – typický soubor, lze přečíst ve všech textových editorech. Data jsou ukládána po řádcích. První řádek je informativní a udává strukturu sloupců pod ním – tím, že jim nadřadí konkrétní název. Druhý řádek již obsahuje první naměřené hodnoty. Na jednom řádku jsou obsažena veškerá data naměřená během jedné měřící periody. Pořadí je následující – čas měření – ve vteřinách, výstupní kanál 1, 2 a následují všechny vstupní kanály, přítomné na kartě. Jednotlivé položky jsou kvůli přehlednosti odděleny znakem *tab* – tedy v jazyce C# oddělovačem */t*. Jednotlivé řádky jsou odděleny odřádkováním *newline*. Poměrná velikost souboru se umístila mezi souborem *csv* a *xml*. Soubor s 14 400 vzorky zabírá 2 581 kB. V ideálním případě by měl soubor *txt* být velký stejně jako *csv*. Avšak vzhledem k faktu, že v textových editorech nejsou přesně definované buňky (narozdíl od tabulkových editorů typu Excel) – bylo třeba formátovat text s ohledem na větší přehlednost. Právě tyto znaky mají na svědomí zvětšení velikostí souboru oproti typu *csv*. Po komprimaci softwarem WinRar jsme získali soubor o velikosti 116 kB. Soubor byl tedy zkomprimován na 4,49 % své původní velikosti.

9.2 Soubor CSV

Soubor s příponou *.csv*. Lehce zobrazitelný v tabulkových editorech – např. MS Excel. Jedná se o oddělovaný datový formát, který má jednotlivé datové buňky odděleny specifickým oddělovačem – v našem případě znakem středníku “;“ V anglicky mluvících zemích je tento oddělovač změněn na znak desetinná tečka “.” Záleží na nastavení znakové sady v operačním systému MS Windows. Samotná struktura souboru je totožná s textovým souborem – tedy data z jedné měřící periody v jednom řádku, oddělená středníkem. Nová perioda je na dalším řádku, odděleném oddělovačem *newline*. Narozdíl od textového souboru, kde plně přebíráme kontrolu nad formou zobrazení je v tomto případě zobrazení ponecháno na automatické funkci daného tabulkového editoru. Proto se můžeme setkat s nechtěnými anomáliemi. Obsahuje-li daná buňka znak „nový řádek“ či středník – případně tečkou – editor by si to mylně vyložil a zobrazil část buňky za tímto znakem v nové buňce nebo na novém řádku. Tomu však lze předejít, pokud celý obsah jedné buňky vložíme do dvojitých uvozovek.

Vzhledem k naší struktuře a zamýšlenému místu použití programu není potřeba více oddělovačů než jednoho středníku mezi buňkami. Proto je výsledná velikost souboru nepatrně

menší než u předchozího textového souboru. Konkrétně při 14 400 vzorcích jsme získali soubor o velikosti 2088 kB. Po zkomprimování nabyl velikosti 113 kB – tedy 5,4% původní velikosti.

9.3 Soubor XML

Dnes velmi rozšířený a oblíbený formát pro ukládání všemožně strukturovaných dat. Problémem XML dokumentů je podmínka jejich správného naformátování. Bez správného formátování nejsou data parserována a nedojde k jejich korektnímu načtení v prohlížeči. Naštěstí je v MS Visual Studiu velmi silná podpora tohoto formátu.

Třída *System.XML* slouží ke správném načítání a ukládání dokumentů ve formátu XML. Pro zápis se používá třída *XmlWriter*, pro čtení *XmlReader*. Jedná se o rychlý streamový zápis XML dat, nepoužívající cache mezipaměť. Následně se provede konfigurace struktury dat – například hodnota odsazení vnořených elementů. Poté se zapíše značka začátku dokumentu – *WriteStartDocument*. Všechna data z jedné měřící periody jsou zapsána v jednom elementu dokumentu. Jednotlivá data jsou uložena jako atributy elementů. Každý element i atribut má své jméno, které je zobrazeno v dokumentu. Je nutné zapsat i koncové znaky elementů a samotného dokumentu. Bez toho je dokument nečitelný. Důvod pro ukládání dat do atributů komponent a nikoli mezi značky počátečního a konečného elementu je ve velikosti souboru. Tímto způsobem ušetříme zbytečný zápis koncových značek elementu. Jak je patrné ze struktury tohoto typu souboru, je zde mnoho informací navíc, oproti předchozím typům. Tomu odpovídá i velikost souboru – 14 400 vzorků uložených ve struktuře XML zabírá na disku 3917 kB, což je téměř dvojnásobek ve srovnáním s CSV souborem. Po komprimaci zabírá soubor 124 kB – tedy 3,16% původní velikosti. To je dobrý výsledek a je způsoben faktem, že v souboru je většina znaků „mezera“, tedy lze dobře zkomprimovat.

Původně aplikace obsahovala ještě možnost přímého zobrazení dat v programu MS Excel. Po testech byla tato funkce odstraněna pro svou praktickou nepoužitelnost. V programu se vytvoří reference na dynamickou knihovnu souboru programů Office. Následně se vytvoří objekt typu Excel a s ním je již možná manipulace podobně jako s otevřenou aplikací Excel. Problémem je celková pomalost tohoto řešení. Maximální rychlost zápisu dat do tabulky Excelu je zhruba 20-30 buněk za vteřinu. Tato rychlost je téměř nezávislá na výkonu hardwaru. Proto není možné tento proces urychlit a stává se v praxi nepoužitelným. Pokus o otevření naměřeného pole dat s několika tisíci vzorky skončil

nezdarem. Celý systém věnoval veškeré dostupné prostředky načítání dat a nebyla možná jakákoli jiná aktivita. Po pročtení zdrojů a diskuzí na internetu bylo shledáno, že se jedná o obecně známou věc a jedinou cestou jak tento proces urychlit je uložení dat do souboru *csv* a jeho následné otevření v aplikaci Excel.

10. Datové vstupy

Aplikace samozřejmě umožňuje i zpětné nahrání dat z uloženého souboru. Pokud má program v datovém poli uložena nějaká data, dotáže se aplikace uživatele, zda si skutečně přeje pokračovat v dané operaci. Poněvadž dojde k předefinování datových polí a tím pádem ztrátě informace. Pokud si uživatel skutečně přeje pokračovat, otevře se komponenta *openFileDialog* s nabídkou 3 možných typů souborů k otevření – tedy XML, TXT a CSV.

Načítaní XML – probíhá velmi jednoduše a intuitivně. Vytvoří se komponenta typu *XMLTextReader*. Následná orientace v souboru je jednoduchá. Struktura uloženého souboru vypadá následovně:

```
<tick xmlns="0,1" />
<s time="0,1" Out1="0" Out2="0" In0="-0,00978" In1="-0,05868" In2="0" In3="0"
In4="0" In5="0" In6="0" In7="0" In8="0" In9="0" In10="0" In11="0" In12="0"
In13="0" In14="0" In15="0,02445" />
```

Úvodní element „tick“ uchovává informaci o vzorkovací periodě se kterou byla data zaznamenána. Přístup k tomuto údaji probíhá přes příkaz *XMLTextReader.ReadToFollowing(“tick”)* – předávaným atributem je jméno vyhledávaného elementu. Hodnotu atributu vyextrahujeme příkazem *XMLTextReader.GetAttribute(„xmlns“)*. Poté již cyklicky čteme hodnoty jednotlivých vstupů a výstupů.

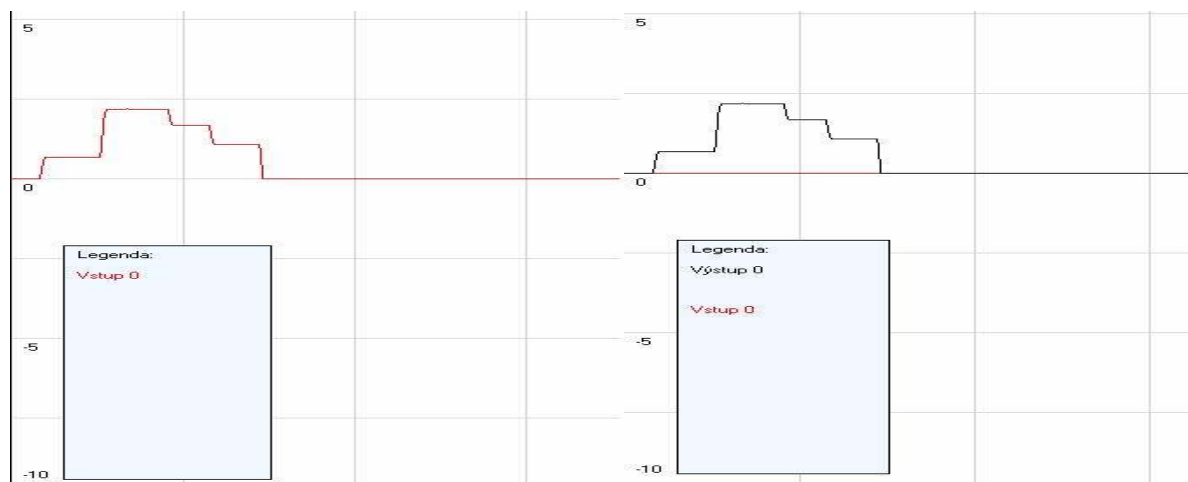
```
while (sr.ReadToFollowing("s"))
{  for (int i = 0; i < x; i++)
     {  attribute = "In" + i.ToString();
     data[i,line] = Convert.ToDouble(sr.GetAttribute(attribute));
     }
     for (int j = 1; j <= 2; j++)
     {  attributeOut = "Out" + j.ToString();
       outData[j-1, line] = Convert.ToDouble(sr.GetAttribute(attributeOut));
     }
     line += 1;
}
```

Tímto zaplníme obě datová pole příslušnými hodnotami a můžeme s nimi nadále pracovat. Nejdříve se však krátce zmíníme o načítání CSV a TXT souborů.

Načítání TXT a CSV – Probíhá přes třídu *StreamReader* a příkaz *StreamReader.ReadLine()*. Cyklicky se načítá jeden řádek za druhým do objektu typu string. Následně dochází k jeho analýze – extrahují se substringy mezi jednotlivými oddělovači – buď '\t' nebo ‚;' dle typu souboru. Skript vypadá takto:

```
while (!sr.EndOfStream)
  {  word = sr.ReadLine();
     for (int i = 0; i < word.Length; i++)
     {  if ((word[i] == '\t') || (i == word.Length - 1))
        {  specimen = specimen + 1;
           subWord = word.Substring(last, i - last);
           last = i + 1;
           if (specimen >= 4)
           {  data[(specimen - 4), line] = Convert.ToDouble(subWord);}}}
     specimen = 0;
     last = 0;
     line += 1;}
```

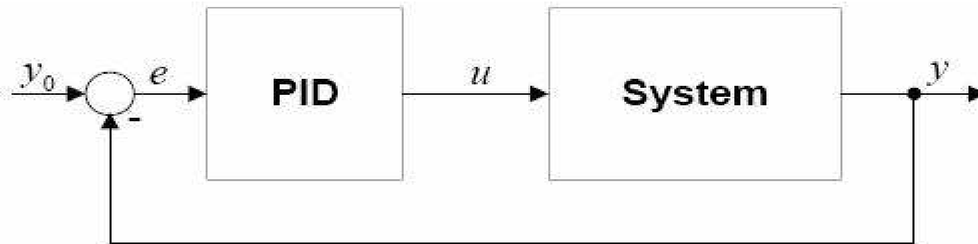
Po nahrání dat do datových polí má uživatel možnost si je prohlédnout na grafu – ovšem pro jejich zobrazení je třeba mít otevřenou komunikaci s kartou a zaškrtnout příslušné kanály, které chce vidět. Proto byla naprogramována funkce, která umožňuje použít nahraná data jako výstup pro další měření. Zaškrtne-li uživatel možnost „*Použít nahraná data pro nastavení výstupních kanálů*" – dojde k aktivaci dvou rolovacích menu s čísly nahraných kanálů. Jejich výběrem přiřadíme data ze souboru do pole *outPut*. Též vzorkovací frekvence se nastaví, dle vstupních dat. Nyní, po otevření komunikačního kanálu na kartě a stisku tlačítka „*Start*" dojde ke kopírování průběhu nahrané funkce na výstupním kanálu – případně kanálech, dle volby uživatele. Během tohoto procesu nejsou výstupní data ukládána – docházelo by k přepisu těch samých hodnot.



Obrázek č.21 Použití nahrané vstupní fce pro budoucí výstupní

11. PID Regulace

Potřebujeme-li nějakým způsobem regulovat výstupní veličinu ze systému na základě odchylky této veličiny od žádané hodnoty, můžeme použít PID regulátory k ovládání akční veličiny. Základní schéma regulované soustavy vypadá následovně:



Obrázek č.22 Schéma regulované soustavy

Jednotlivá písmena představují tyto veličiny:

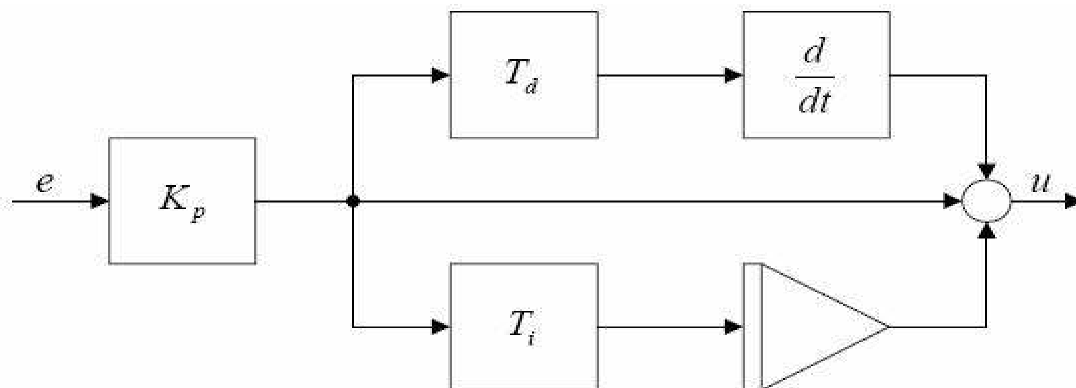
y_0 - žádaná hodnota

e - regulační odchylka (chyba)

u - akční veličina

y - výstupní veličina ze systému

PID regulátor porovnává hodnotu výstupu systému s hodnotou žádanou a dle jejich rozdílu nastavuje akční veličinu. Chyba (rozdíl hodnoty naměřené a žádané) je zpracována třemi způsoby – přes proporcionální, integrační a derivační složku PID regulátoru. Ukážeme si schéma regulátoru na diagramu:



Obrázek č.23 Schéma PID regulátoru

Kp – Proporcionální složka

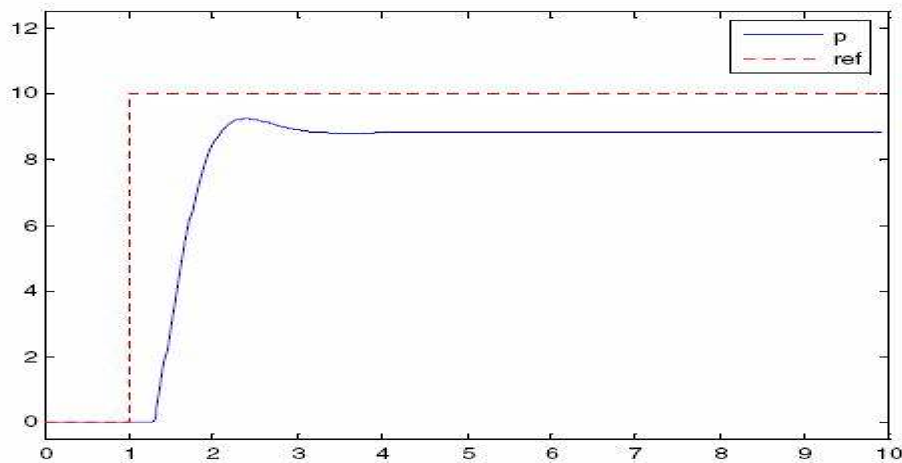
Td - Derivační časová konstanta

Ti - Integrační časová konstanta

Popíšeme si vliv jednotlivých složek na chování systému

11.1 Proporcionální složka

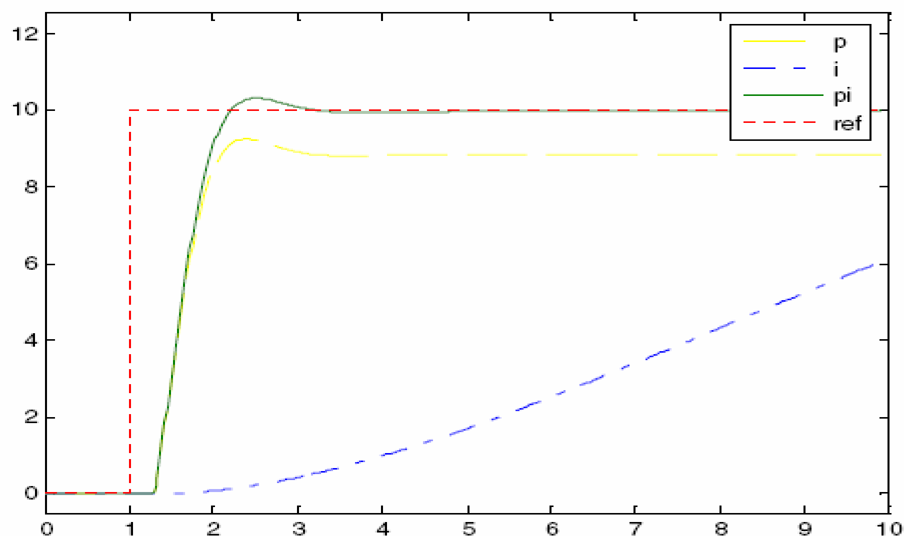
Při použití pouze složky P daného regulátoru docílíme stacionární regulační odchylky ve všech případech – s výjimkou pro případy, kdy je vstup do systému roven nule. Na obrázku vidíme stacionární odchylku od žádané hodnoty. Příliš vysoká proporcionální složka destabilizuje systém.



Obrázek č.24 Vliv P složky na chování systému

11.2 Integrační složka

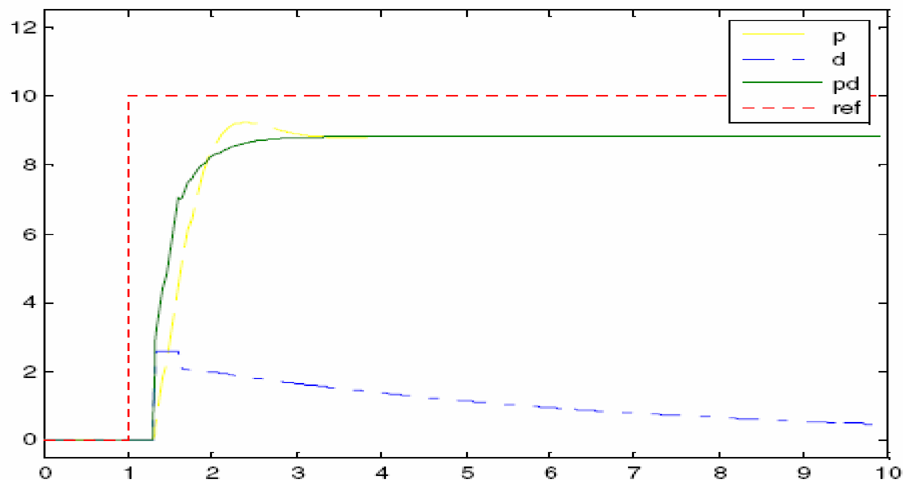
Přidává aditivní hodnotu k akční veličině – založenou na součtu předchozích chyb. Sčítání chyb pokračuje až do doby, než se výstupní hodnota systému vyrovná požadované. Nejčastěji se I složka používá společně s P složkou – tedy PI regulátor. Použití samotné I složky způsobí zpomalení odezvy systému a někdy i jeho oscilaci. PI regulátor nemá žádnou stacionární odchylku.



Obrázek č.25 Vliv I složky na chování systému

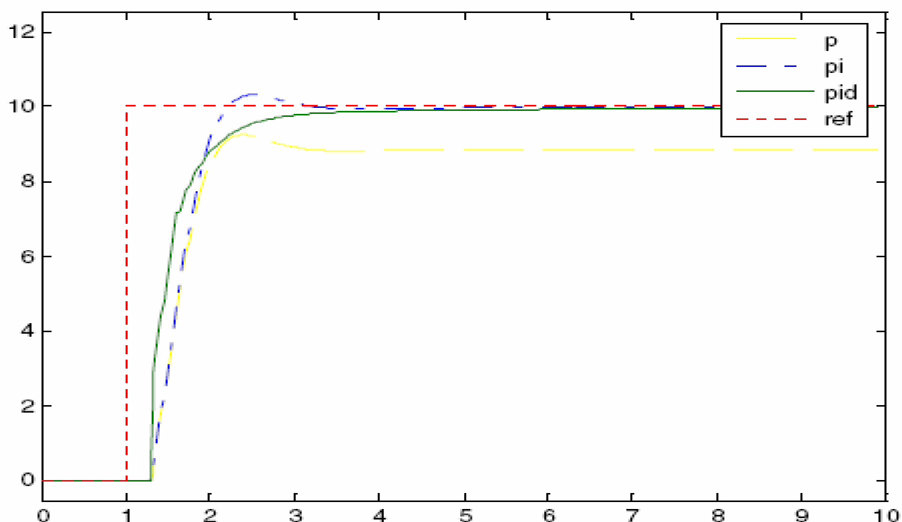
11.3 Derivační složka

Hodnota, kterou přidává D složka regulátoru závisí na četnosti změn odchylky (chyby) výstupu systému vůči požadované hodnotě. Rychlá změna chyby způsobí aditivní změnu akční veličiny. Účelem je zlepšení odezvy systému na rychlé a náhlé změny stavu či referenční hodnoty. Příliš veliká složka D destabilizuje systém. D složka se používá společně se složkou P (PD regulátor) nebo PI (PID regulátor). Zrychluje odezvu systému více než samotná P složka. Navíc se složka D chová jako filtr typu horní propust chybových signálů – tedy dělá systém náchylnější k aditivnímu šumovému signálu.



Obrázek č.26 Vliv D složky na chování systému

Ve většině případů dosáhneme nejlepších výsledků s použitím všech tří složek PID regulátoru.



Obrázek č.27 Vliv PID regulátoru na chování systému

11.4 Diskretizace přenosu PID regulátoru

Vzorce klasického PID regulátoru se uvádí ve spojité časové formě, poněvadž se předpokládá regulace spojitého procesu:

$$u(t) = K_c e(t) + \frac{K_c}{T_i}\int_0^t e(t)dt + K_c T_d \frac{de(t)}{dt} + u_o \tag{1}$$

Avšak to nás staví před problém, kterak tento algoritmus naprogramovat do programu, který pracuje diskrétně. Proto musíme aproximovat spojité operace derivace a integrace v diskrétní formě.

Aproximace derivace: $$\frac{de(t)}{dt} \approx \frac{e(t) - e(t-1)}{T_s} \tag{2}$$

Aproximace integrace: $$\int_0^t e(t)dt \approx T_s \sum_0^t e(i) \tag{3}$$

Tedy po dosazení do vzorce dostaneme následující rovnici:

$$u(t) = K_0 e(t) + \frac{K_c T_s}{T_i}\sum_{i=0}^{t} e(i) + \frac{K_c T_d (e(t) - e(t-1))}{T_s} + u_0 \tag{4}$$

Dostali jsme formu, která je již naprogramovatelná v jakémkoli programovacím jazyce. Tato forma se nazývá „*poziční*“, kvůli faktu, že akční veličina je vypočítávána s referencí na základní úroveň u_0. Existuje ještě jiný přístup k diskretizaci – a sice ze sériové struktury PID regulátoru. Jedná se o odvození diskrétní formy přenosu PID regulátoru, od přenosu uvedeného v Laplaceově transformaci.

$$\frac{U(s)}{E(s)} = K_c\left[1 + \frac{1}{T_i s} + T_d s\right] \tag{5}$$

Zpětnou diferencí nebo bilineární transformací dostaneme ekvivalent této rovnice v diskrétní formě.

$$\frac{U(s)}{E(s)} = K_c\left[1 + \frac{1}{T_i s} + T_d s\right] \xrightarrow{backward-difference} u(t) = e(t) K_c\left[1 + \frac{T_s}{T_i(1 - z^{-1})} + T_d \frac{(1 - z^{-1})}{T_s}\right] \tag{6}$$

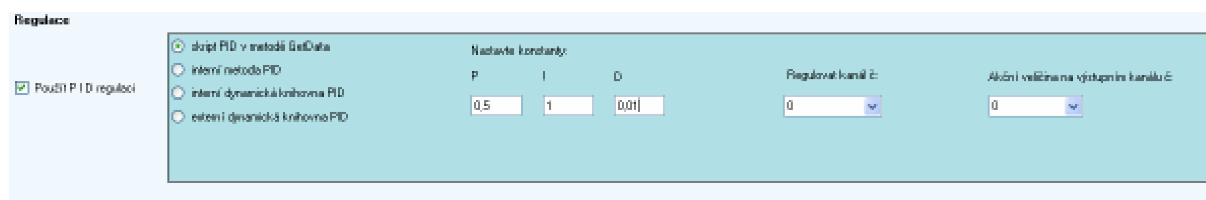
Po zjednodušení dostaneme následující vzorec:

$$u(t) = u(t-1) + K_c\left[e(t) - e(t-1)\right] + \frac{K_c T_s}{T_i} e(t) + \frac{K_c T_d}{T_s}\left[e(t) - 2e(t-1) + e(t-2)\right] \tag{7}$$

Tato forma se nazývá „*rychlostní*“ nebo též „*sériová*“. Narozdíl od předchozího příkladu kde se pro výpočet akční veličiny používala referenční hladina u_0, používáme zde pro výpočet předchozí hodnotu akční veličiny.

11.5 Aplikace PID regulace v ControlSystemu

Nastavení typu regulace, konstant a způsobu implementace regulačního skriptu probíhá na záložce „*Regulace / Manipulace s daty*“.



Obrázek č.28 Panel pro ovládání a nastavování PID regulace

Panel s názvem „*Regulace*“ není přístupný a ovladatelný až do okamžiku, kdy je otevřen komunikační kanál s kartou. ControlSystem neumí simulovat průběh dějů, proto je offline simulace nepoužitelná a tudíž zbytečná. Pokud máme již panel zpřístupněný, můžeme zaškrtávacím checkboxem zvolit možnost, že chceme na výstupní kanál posílat vypočítané hodnoty z PID regulátoru. V další nabídce je volba typu implementace PID skriptu. Po analýze byly zvoleny čtyři možnosti implementace, pátá byla vynechána. O důvodu vynechání této možnosti se budeme nyní krátce věnovat.

11.5.1 Parametry P, I a D použité v aplikaci

Vzhledem k použité sériové struktuře PID regulátoru je třeba si uvědomit, v jaké formě budeme zadávat parametry P, I a D do textových polí aplikace. Uveďme si vzorec sériové formy regulátoru:

$$u(t) = K * \left[e(t) + \frac{1}{T_i} \int e(t) + T_d \frac{de(t)}{dt} \right] \tag{8}$$

Pro jednotlivé parametry dostaneme následující hodnoty:

$$P = K \rightarrow K = P \tag{9}$$

$$I = \frac{K}{T_i} \rightarrow T_i = \frac{K}{I} \tag{10}$$

$$D = K * T_d \rightarrow T_d = \frac{D}{K} \tag{11}$$

Do textových polí zadáváme hodnoty K, Ti a Td – je třeba na toto pamatovat a nezadávat hodnoty P, I a D! V opačném případě nebude regulátor korektně nastaven a nebude funkční dle předpokladů.

11.5.2 Text psaný uživatelem - do textového pole a následně včleněný do kódu – Jedná se o možnost plné kontroly nad probíhajícím skriptem (např. PID regulace) uživatelem. Vepsaný text – například do komponenty *textBox* nebo *richTextBox* – se přes třídu *CSharpCodeProvider* zkompiluje a začlení do kódu aplikace. Jedná se sice o velmi zajímavou možnost vkládání vlastního kódu, ale negativa převažují. Prvním velmi důležitým aspektem je nutnost ovládat programovací jazyk C#. Pravděpodobnost, že ho bude znát a ovládat některý ze studentů, potencionálně užívající aplikaci ControlSystem během výuky, je velmi malá. Další nevýhodou je nutnost znalosti kódu samotné aplikace – v případě, že by byl vkládaný skript nevhodně naprogramován, mohlo by dojít k narušení přesného časování, případně k nějaké nekonečné smyčce a pádu celé aplikace. Z těchto důvodů byl tento způsob ovládání/implementace PID regulace (či jiných) zcela vynechán.

11.5.3 Součást kódu aplikace – nejjednodušší a nejrychlejší možnost implementace. Jedná se o vložení cca 10ti řádků kódu do metody *GetData*, počítajících hodnotu vstupního kanálu pro regulaci systému na základě hodnot naměřených na výstupním kanálu. Odpadá definice nových proměnných, implementace používá pouze lokální proměnné celé aplikace. Velkou nevýhodou je neflexibilita tohoto řešení. Budeme-li chtít použít kód na více místech aplikace, budeme ho muset celý napsat na příslušné místo. Taktéž budoucí úpravy jsou velmi náročné na pozornost programátora, aby nezapomněl změnit text ve všech implementacích.

11.5.4 Interní metoda aplikace – klasická funkce, která je definována v aplikaci jen jednou a poté volána z různých míst se zadanými parametry. Výhodou zůstává možnost použít lokální proměnné. Případnou změnu provádíme jen v definici kódu, odpadá negativum předchozího typu implementace. Metoda je typu *double*, parametry posílané do metody jsou – číslo posledního naměřeného vzorku, hodnoty P, I a D regulátoru, vstupní a výstupní kontrolovaný kanál.

11.5.5 Interní třída aplikace – definovaná ve stejném souboru, kde se nachází samotná aplikace. Stejná implementace jako u třídy *HiPerfTimer*, ovládající časování. Problémem je nemožnost použití lokálních proměnných – musíme tedy posílat více parametrů a definovat nové proměnné. Kromě výše zmíněných parametrů posíláme navíc parametry: hodnota PID výstupu z minulého kroku, regulační odchylky až do 2 kroku do minulosti.

11.5.6 Externí třída aplikace – kód naprogramovaný mimo samotnou aplikaci – ve zcela nezávislém souboru, zkompilováno do typu **.dll*. Přiřazeno do aplikace ControlSystem pomocí reference na daný soubor. Jediné co potřebujeme znát je jmenný prostor dané třídy. Jakmile nadefinujeme nový objekt této třídy, máme přístup ke všem metodám, které třída obsahuje. Posílané parametry se zobrazují ve vyskakujícím okně během psaní příkazu – práce ve Visual Studiu je velmi příjemná díky podobným vlastnostem.

Nyní se budeme věnovat samotné implementaci kódu PID regulátoru do jazyka C#. Jedná se o jednoduchý a rychlý skript, určený především k odzkoušení možností implementace samotného kódu, než k optimální regulaci systému. To není předmětem této aplikace, spíše hlavní možností jejího rozšíření do budoucna.

```
error = wantedValue[setPIDToOut] - data[readPIDFromIn, numOfSamples];
errorT_1 = wantedValue[setPIDToOut]-data[readPIDFromIn,(numOfSamples - 1)];
errorT_2 = wantedValue[setPIDToOut]-data[readPIDFromIn,(numOfSamples - 2)];
outData[setPIDToOut,numOfSamples]=outData[setPIDToOut,numOfSamples1)]
+P*(error-errorT_1 + I*error + D*(error - (2*errorT_1) + errorT_2));
  if (outData[setPIDToOut, numOfSamples] > 10)
  {  outData[setPIDToOut, numOfSamples] = 10;  }
  if (outData[setPIDToOut, numOfSamples] < 0)
  {  outData[setPIDToOut, numOfSamples] = 0;   }
output1 = (short)(outData[setPIDToOut, numOfSamples] / 0.002445);
dac.AnalogOutputWrite(setPIDToOut, output1);
```

Je vidno, že samotný výpočet výstupní hodnoty je prováděn na jediném řádku. Poté se převede na hodnotu typu *short* v proměnné *output*, ta je zapsána na výstupní port přes D/A převodník.

12. Výsledky měření na reálné soustavě

Pro ověření funkčnosti programu byla použita soustava tachodynamo (řízení otáček tachometru propojeného pružnou spojkou s DC-motorem) z laboratoře TK4 KŘT. Proběhla dvě měření (bez regulace / s regulací) v aplikaci ControlSystem a ta samá v aplikaci Matlab Simulink. Záměrem bylo porovnání získaných průběhů a vyhodnocení pravdivosti získaných vzorků v aplikaci ControlSystem. Úloha tachodynamo byla vybrána z důvodu toho, že se jedná o jednu z nejrychlejších úloh v laboratoři. Proto je zcela vitální dodržení vysokých vzorkovacích frekvencí a velmi rychlá regulace. V Matlabu se ke konci prvního měření vyskytla závada způsobená frekvencí vzorkování, která je na hranici jeho možností a počítač

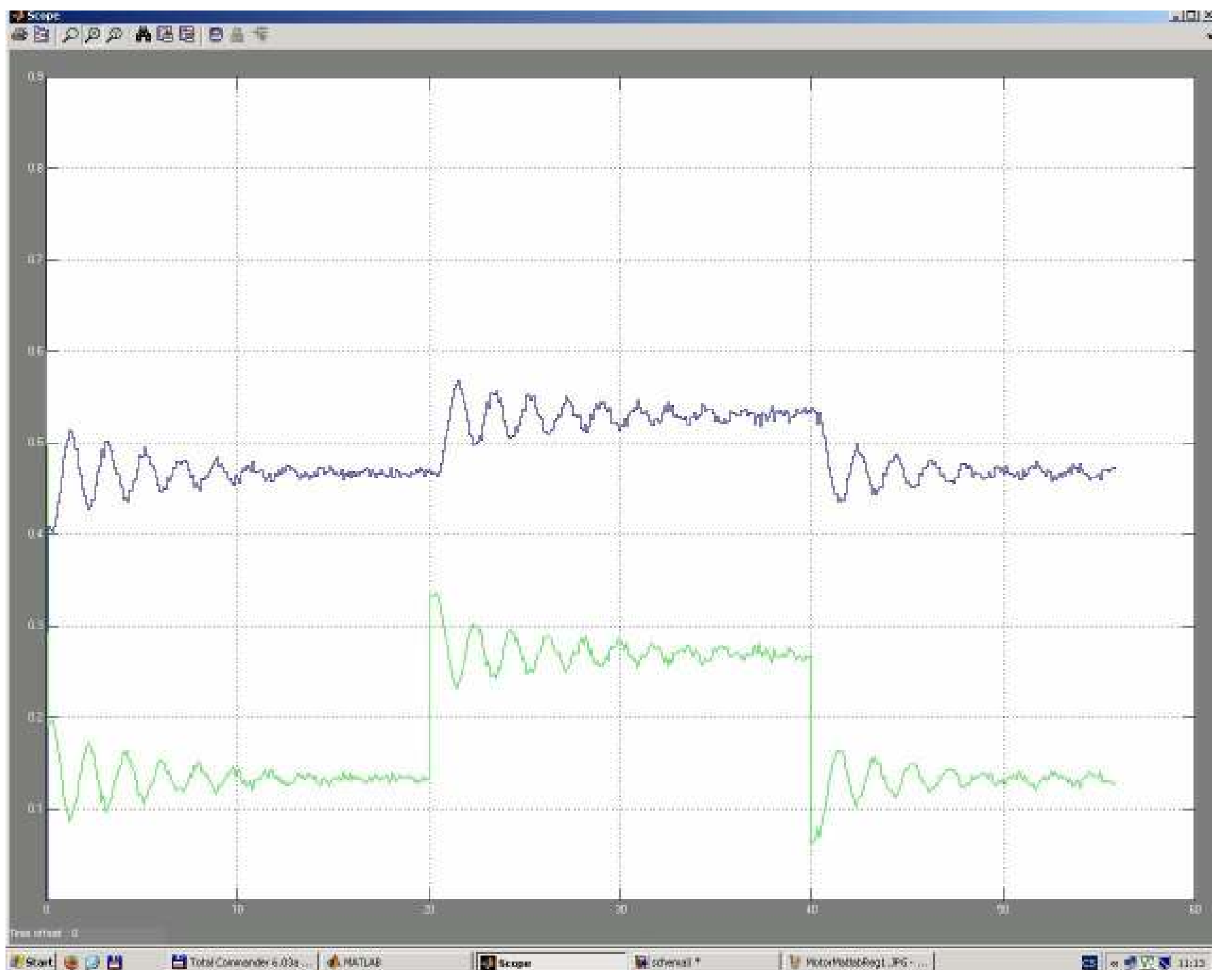
musel být restartován. Dle informací konzultanta je na vině fakt, že RealTime toolbox v Simulinku je z předchozí verze programu a není zajištěna dostatečná kompatibilita. Měření v ControlSystemu proběhlo bez závad.

12.1 Postup měření

Měření probíhalo po dobu 1 minuty se vzorkovací frekvencí 100Hz. Během prvního záznamu dat se akční veličina z nulové hodnoty nastavila na hodnotu 2V, po 20ti vteřinách na hodnotu 4V a po dalších 20ti vteřinách zpět na hodnotu 2V. Soustava nebyla regulována pomocí PID regulátoru. Při druhém pokusu byl do nastavování výstupní hodnoty zapojen PI regulátor s následujícím nastavením:

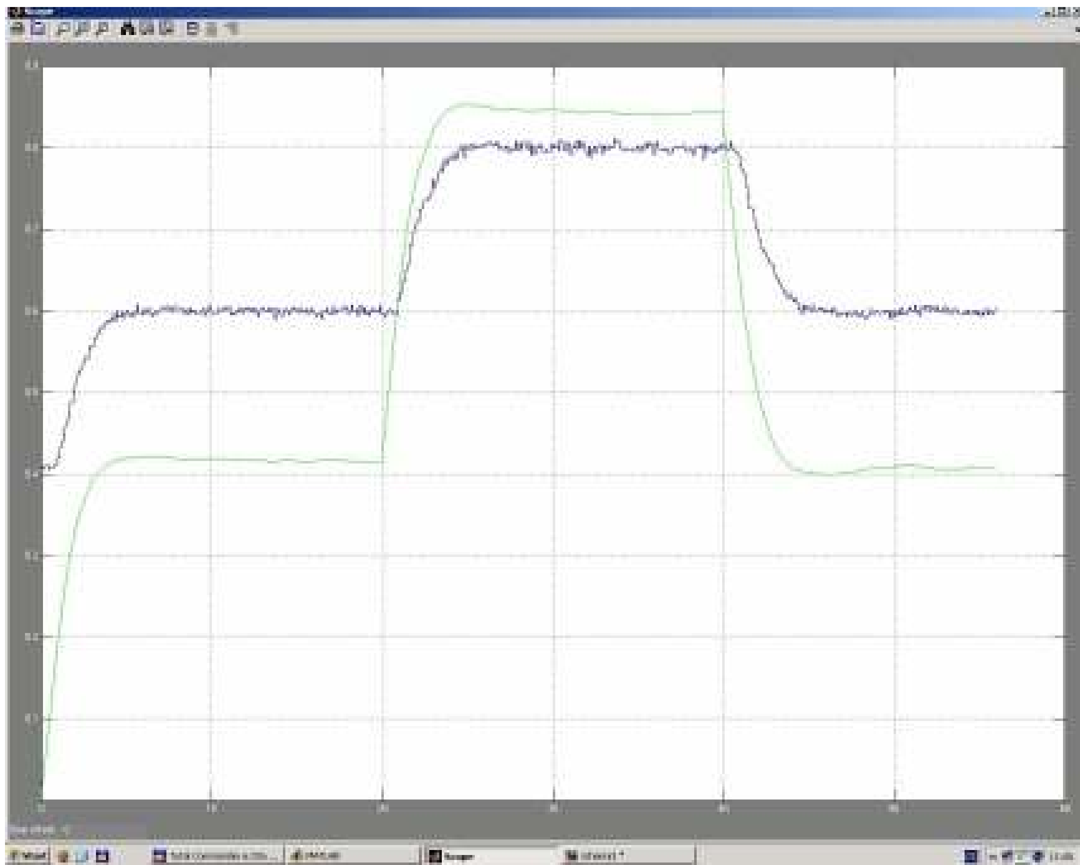
Matlab:	P = 0.0590	I = 0.990
ControlSystem:	P = 0.0590	Ti = 0.0595

Odlišnost hodnot je způsobena přepočtem parametrů, jak je uvedeno v podkapitole 6.1 kapitoly PID Regulace.

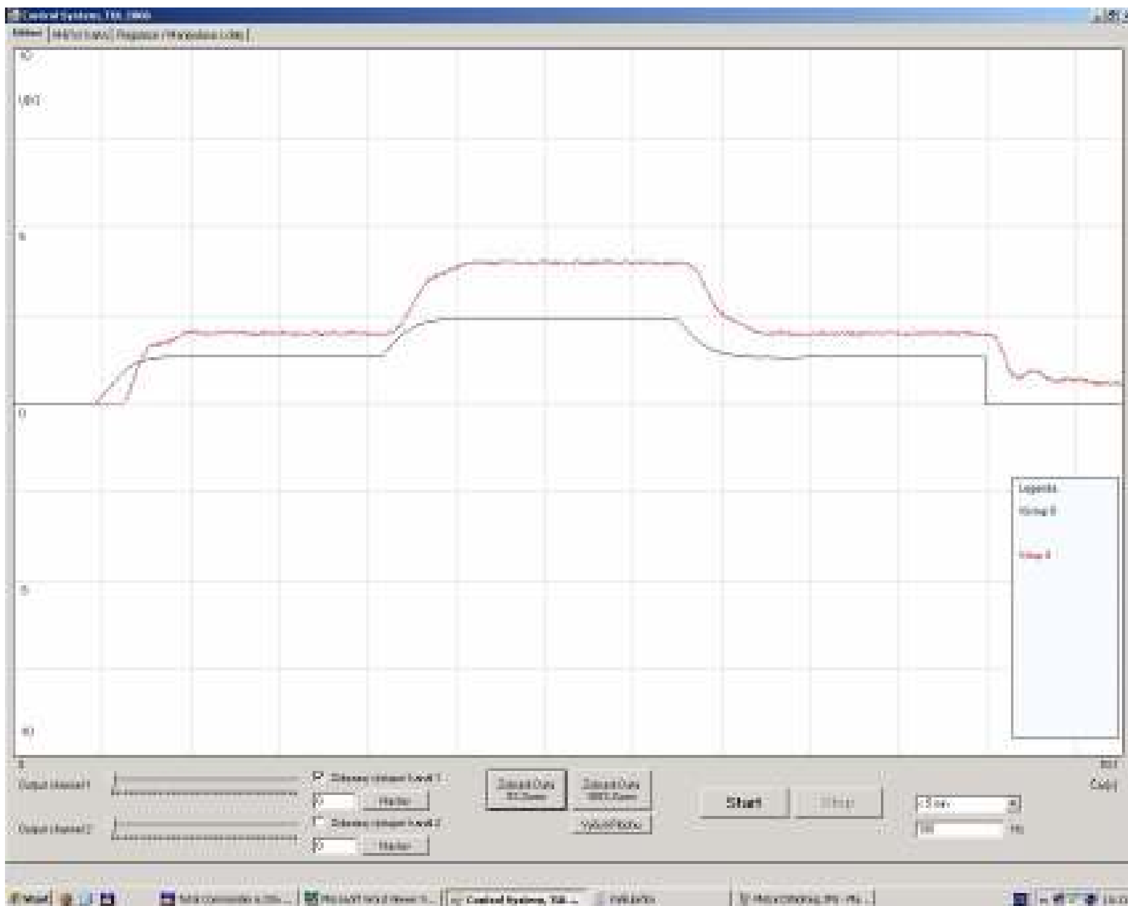


Obrázek č.29 Neregulovaná soustava – Matlab

Obrázek č.30 Neregulovaná soustava – ControlSystem



Obrázek č.31 Regulovaná soustava – Matlab



Obrázek č.32 Regulovaná soustava – ControlSystem

Jak je dobře patrné i přes jiný poměr vykreslování, obě měření si plně odpovídají. ControlSystem tedy naměřil důvěryhodná data a dokázal soustavu přesvědčivě regulovat. Vzhledem k nevhodné struktuře použitého PID regulátoru téměř není možné použít D složku k regulaci zašuměných systémů v aplikaci ControlSystem. Vzhledem k tomu, že samotný návrh optimálního regulátoru nebyl cílem práce a sloužil jen jako ukázka způsobu implementace kódu do aplikace v budoucnu, lze tento fakt akceptovat. Implementace nového vhodného regulátoru je prvním uvažovaným rozšířením ControlSystemu do budoucna.

Závěr

Tato práce ukazuje možnost odklonu od používání komerčních aplikací k měření a regulaci systémů. Pokud by se v budoucnosti katedra řídící techniky soustředila na rozvoj této aplikace do komplexního nástroje, mohla by s její pomocí alespoň zčásti nahradit stávající programy. Studentům je dána možnost porovnat práci ve více programech a následně si zvolit uživatelsky nejpříjemnější. Program ControlSystem je ze všech alternativ nejjednodušší na pochopení a ovládání. Umožňuje okamžitou a pohodlnou práci i studentům bez jakýchkoli znalostí podobných řešení.

Jak bylo ukázáno ve výsledných měřeních, je aplikace plně konkurence schopná co se kvality naměřených vzorků týče. Její obsluha je velmi jednoduchá a nevyžaduje programovací znalosti uživatele. Kód aplikace je dostatečně rychlý a umožňuje vzorkování systému až 10x vyššími frekvencemi než Matlab. V případě potřeby lze zvýšit nastavení vzorkovací frekvence až k 10 KHz – avšak zde se již potýkáme s otázkou, zda to má pro relativně pomalé laboratorní úlohy smysl. I nejrychlejší používané systémy si vystačí s frekvencí vzorkování 100Hz, tedy 10x menší, než program umožňuje.

Regulace soustavy je též na velmi dobré úrovni, pokud nezahrneme D složku regulátoru. Zde je hlavní směr pro zlepšení stávajících algoritmů – náhrada / přidání jiných struktur regulátorů, pro dosažení optimálních výsledků. Další rozšíření je třeba provést v otázce kompatibility se stávajícími aplikacemi – například export dat do souboru typu *.mat, použitelným v Matlabu.

Shrňme tedy přínosy této práce do několika bodů:

- vytvoření aplikace vhodné pro použití během výuky
- snadnější ovládání než konkurenční aplikace
- možnost okamžitého zahájení měření, bez nutnosti psát program či sestavovat model systému
- možnost sledování systému vysokými vzorkovacími frekvencemi
- použití moderní programovací platformy umožňující dobré rozšíření programu o požadované funkce v budoucnu

Použitá literatura

[1] Pirkl, Josef. Řešené příklady v C# aneb C# skutečně prakticky. České Budějovice: KOPP, 2005. ISBN: 80-7232-265-6

[2] Press, H., William & team. Numerical Recipes in C – The art of scientific computing. 2nd ed., Cambridge university press, Cambridge, 1998

[3] Modrlák, Osvald. Učební texty [on-line]. [cit. květen 2003]. KŘT TUL. Dostupné na World Wide Web: http://www.fm.vslib.cz/~krt/krt_cz/vyuka/text.htm

[4] Virius, Miroslav. C# pro zelenáče. Praha: Neocortex, 2002. ISBN: 80-86330-11-7

[5] Precision Timing Under Windows Operating System. Dostupné na World Wide Web: http://www.grahamwideman.com/gw/tech/dataacq/wintiming.htm

[6] Control Information from John Shaw. Dostupné na World Wide Web: http://learncontrol.com/pid/

[7] Advantech product guide & software support. Dostupné na World Wide Web: http://www.advantech.com

[8] C# solved coding examples, miscelanous dynamic libraries. Dostupné na World Wide Web: http://www.thecodeproject.com