TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B 2612 – Elektrotechnika a informatika

Studijní obor: Informatika a logistika

Software pro ovládání analogové I/O měřící karty PCI-1711

Software for control analog I/O data acquisition control PCI-1711

Bakalářská práce

Autor: **Jan Macháček**

Vedoucí práce: Ing. Lukáš Hubka

V Liberci 19. 5. 2010

Prohlášení

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

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

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

Datum: 19. 5. 2010

Podpis: .....

Abstrakt

Cílem bakalářské práce je vytvoření softwaru pro multifunkční měřící kartu s využitím dynamických knihoven výrobce karty a správným zpracováním přijímaných dat, s jejich následným zobrazením a ukládáním. Teoretická část je zaměřena na popis vlastností měřících karet a na dostupný software obstarávající jejich obsluhu. Kapitola také obsahuje informace o používané kartě s jejím přesnějším popisem. Část práce „tvorba vlastní aplikace“ popisuje tvorbu jednotlivých částí aplikace s ukázkami zdrojového kódu. Obsahuje popsaný vývoj aplikace a použití dodaných dynamických knihoven. V části test aplikace je popsána metodika testování jednotlivých částí karty s ukázkami načtených dat v grafu, popřípadě po jejich uložení zobrazena v matlabu.

Klíčová slova

Měření dat, měřící karta, Visual C++ .NET, software, dynamické knihovny, DLL

Abstract

Aim of this bachelor work is creating software for data acquisition control with use dynamic library from card manufacturer with correct process of reading data and their drawing to a graph and saving to a file. Theoretical part is localized to description about parameters data acquisition control and software which provide their work. Chapter so include information about our card with specific description. Part work which is called “creation of own application” description create of individual parts application with previews of source code. So includes progression of application and using dynamic library. In part called test application is write a method of testing individual parts with preview of read data in graph or show this save data in matlab.

Keywords

Data acquisition, DAQ, Visual C++ .NET, software, dynamic library, DLL

Obsah

1 Úvod

Měřící karty se využívají především v průmyslu, kde slouží pro získávání hodnot z různých snímačů, kde získaná data vyhodnotí počítač, který následně může ovlivňovat akční členy připojené na výstup karty. Z tohoto pohledu se může jejich práce zdát podobná například programovatelným automatům. Oproti nim jsou však rychlejší a data z nich můžeme dále zpracovávat a uchovávat v počítači.

Cílem práce je vytvoření aplikace pro multifunkční měřící kartu, která bude využívat možnosti dodávaných dynamických knihoven od výrobce měřící karty. Hlavní pozornost by měla být věnována čtení analogových dat, kde by bylo výhodné zprovoznění čtení ze zásobníku karty.

Teoretická část pojednává o měřících kartách. Je zde popsáno k čemu karty slouží a jaké mají základní funkce. Jednotlivé funkce jsou dále rozepsány a jsou zde uvedeny základní parametry měřících karet.

V praktické části se zabýváme tvorbou vlastní aplikace. Zabýváme se tvorbou a vývojem jednotlivých sekcí, do kterých je aplikace rozdělena. Následně popisujeme postupný vývoj a práci s dodanými dynamickými knihovnami. V části testování aplikace testujeme možnosti karty a hodnotíme přesnost měření a naměřená data zobrazujeme.

2 Teoretická část

2.1 Měřící karty

2.1.1 Funkce měřících karet

Měřící karty[1] neboli karty pro sběr dat (data acquisition - DAQ), slouží pro měření, popř. vytváření signálu počítačem a jsou vlastně rozhraním mezi signálem a počítačem. Při získávání dat karta vzorkuje signál a převádí ho tak do digitální podoby, abychom ho mohli dále využít, popřípadě vytváří z digitálních dat signál analogový.

Měřící karty většinou obsahují jeden či několik z těchto typů funkcí:

- Analogové vstupy
- Analogové výstupy
- Digitální linky
- Čítače nebo časovače
- A další…

Karty s více než jedním z uvedených typů funkcí se označují jako multifunkční.

2.1.2 Rozdělení karet podle sběrnice

Vzhledem k rozšíření osobních počítačů se sběrnicí PCI (např. multifunkční měřící karta Advantech PCI-1711) převládá tato sběrnice i u měřících karet. U výkonnějších systémů je možné vidět sběrnici PXI (např. měřící karta pro analogové čtení National Instruments PXIe-4498), která díky zvláštním vodičům pro přenos časování umožňuje synchronizovat několik měřících karet a tím rozšířit počet kanálů nebo vytvářet větší systémy se současným řízením pohybu či snímáním obrazu. Postupně se objevují i nové způsoby připojení, například bezdrátové připojení a nově také přes sběrnici PCI – Express (např. multifunkční měřící karta National Instruments PCIe-6251). Stále se vyrábějí a prodávají karty i pro starší sběrnici ISA, a to zejména pro počítače v průmyslovém provedení. Pro mobilní použití jsou vhodné karty ve formátu PCMIA, použitelné v notebooku. V dnešní době se vyskytují i externí zařízení připojovaná k počítači prostřednictvím sběrnice USB (např. multifunkční měřící karta Advantech USB-4716).

2.1.3 Analogové vstupy

Základní parametry analogových vstupů jsou jejich počet, rozlišení, maximální vzorkovací frekvence a napěťové rozsahy.

Rozlišení vstupu je dáno použitým A/D převodníkem, ale ve výsledku mohou přesnost ovlivnit i další zařazené obvody a to jak v dobrém, tak ve špatném. V dnešní době se používají dvanáctibitové až šestnáctibitové převodníky. S nižším rozlišením se lze dnes setkat jen u nejrychlejších (osciloskopických) karet.

Vzorkovací frekvence bývají v rozsahu 10 kS/s u nejpomalejších karet připojovaných přes USB sběrnici až 30 MS/s u nejrychlejších karet připojovaných přes sběrnici PCI. U levnějších karet bývá problém, že se všechny vstupy přepínají do jednoho A/D převodníku a tak podle počtu vstupů klesá rychlost vzorkování a vzniká zpoždění mezi jednotlivými kanály. Těmito nedostatky netrpí karty vybavené plnohodnotným řetězcem, což znamená, že každý vstup má svůj vlastní vstupní zesilovač a A/D převodník.

Karty mívají napěťové rozsahy bipolární a unipolární. Bipolární využíváme při měření signálů, které mění polaritu a unipolární jsou pro měření signálu se stejnou polaritou. Většinou mají karty pro unipolární signál volitelný rozsah přibližně od 0 až ±0,625V do 0 až ±10V a pro bipolární mají rozsah od 0 až 0,125V do 0 až 10V při možnosti zvolit přibližně 4 mezistupně. Můžeme se setkat i s kartami, které mají rozšířený rozsah, kde je poté veliká přesnost až ± 0,005V. Je dobré měřit v rozsahu, který odpovídá signálu, abychom dosahovali co nejmenšího kvantizačního šumu.

2.1.4 Analogové výstupy

U analogových výstupů hodnotíme stejné funkce jako u analogových vstupů. Některé karty mají výstupy pouze statické a lze na nich vysílat vždy pouze jednu hodnotu, kde je rychlost změny dána pouze chováním ovládajícího softwaru. To s sebou nese velikou nejistotu v přesnosti časování signálu. Naopak karty s hardwarovým časováním a vyrovnávací pamětí dokážou celkem přesně generovat i rychlejší průběhy. Napěťové rozsahy většinou mívají méně možností oproti analogovým vstupům, pravidlem zde bývá rozsah 0 až ±10V.

2.1.5 Digitální linky

Digitální linky můžeme najít na většině měřících karet. Linky mohou být obousměrné nebo je směr pevně dán, zejména u karet s izolovanými linkami. Neizolované linky používají úrovně TTL a CMOS, opticky izolované umožňují spínat a snímat napětí do 60V, popř. i více. U levnějších karet bývají vstupy a výstupy statické, dovolují přečíst nebo vysílat vždy jediný stav. Některé levnější karty ale mají více možností a nabízejí detekci změny stavu, což nám ušetří testování v časových smyčkách. Dále je možné se setkat s programovatelnými filtry pro omezení zákmitů nebo rušivých impulsů.

Pro rychlé delší záznamy nebo generování posloupností jsou určeny karty s hardwarově časovanými linkami a vyrovnávací pamětí. Takové karty lze použít jako analyzátor logických stavů nebo rychlý binární generátor.

2.1.6 Čítače a časovače

Používají se k počítání impulsů, dělení frekvencí nebo vytváření signálů s požadovanou frekvencí a pro připojení inkrementačních čidel. U čítačů je rozhodující délka registru, tj. maximální hodnota registru, po jejímž dosažení čítač přeteče a začne čítat znovu od nuly. Dále se uvádí maximální frekvence, kterou dokáže čítač zpracovat.

2.2 Měřící karta Advantech PCI-1711

Jedná se o základní typ multifunkční měřící karty od společnosti Advantech do slotu PCI (viz. Obrázek 2-1).

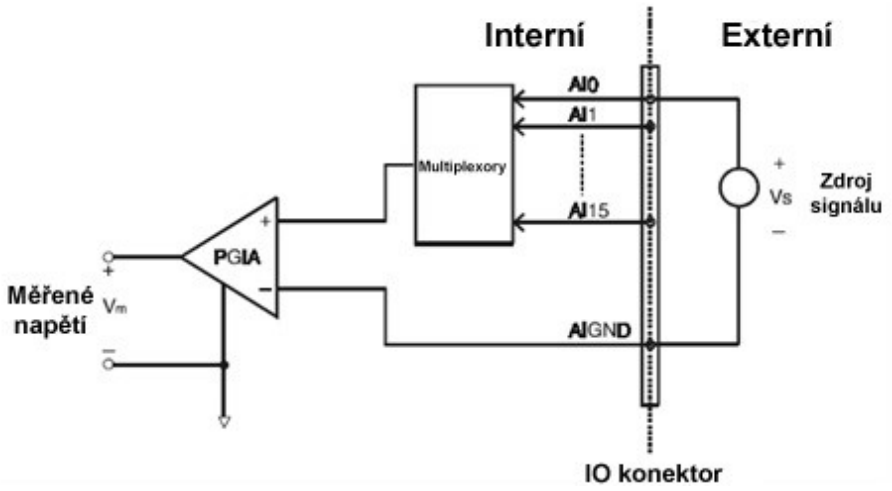
Obrázek 2-1- Měřící karta Advantech PCI-1711

Informace o kartě [2][3]:

16 analogových vstupů:

- Rozlišení - 12 bitů
- Rychlost vzorkování - 100 kS/s
- Napěťové rozsahy - ±10V, ±5V, ±2,5V, ±1,25V, ±0,625V – pro každý kanál je možné zvolit odlišný napěťový rozsah.
- Ochrana proti přepětí do 30Vp-p

Karta obsahuje pouze normální vstupy (viz. Obrázek 2-2), oproti pokročilejším kartám kde je možné nalézt rozdílové vstupy, které dokážou měřit rozdíl mezi dvěma kanály.

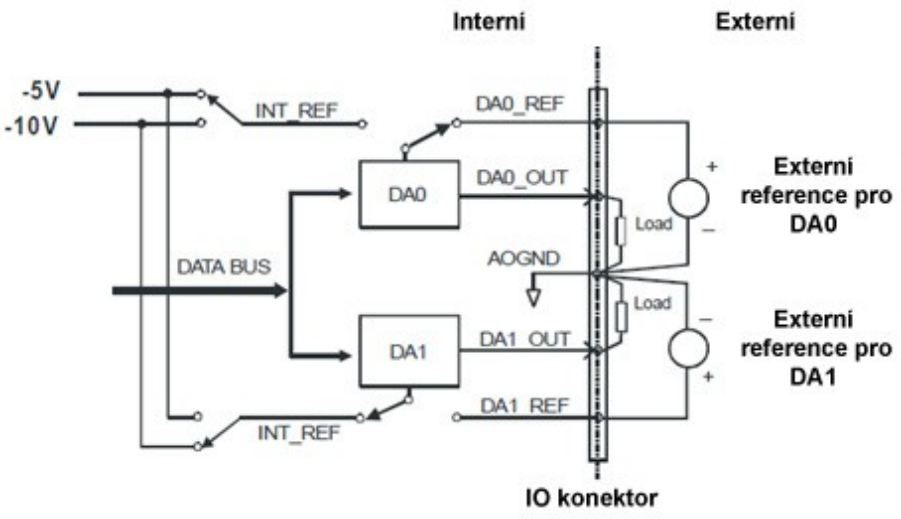


Obrázek 2-2 – Schéma zapojení analogového vstupu

2 analogové výstupy:

- Rozlišení – 12 bitů
- Napěťový rozsah:
  - Interní reference - 0 ~ +5V, 0 ~ +10V
  - Externí reference – 0 ~ +x V(-x V)
    při velikosti referenčního napětí $-10 \leq x \leq 10$V

Výstupy u této karty jsou řešeny oproti vyšším verzím pouze statickým vysíláním hodnoty na výstup a tak je vysílání realizováno pouze obslužným softwarem, který generuje signál a postupně hodnoty zapisuje na výstup karty. Vyšší modely dokážou při vysílání signálu použít zásobník a tak lze vysílat i rychlé průběhy.



Obrázek 2-3 – Schéma zapojení analogového výstupu

16 digitálních vstupů:

- Kompatibilita s 5V/TTL
- Vstupní napětí:
  - Logická „0“: max. 0,8V
  - Logická „1“: min. 2,0V

16 digitálních výstupů:

- Kompatibilita s 5V/TTL
- Výstupní napětí:
  - Logická „0“: 0,8V
  - Logická „1“: 2,0V

Vyrovnávací zásobník:

Karta je osazena vyrovnávacím zásobníkem pro uchování 1024 vzorků. U zásobníku je možné povolit přerušení. To zprovozní funkci, kdy při každém novém vzorku nebo jen když je zásobník zaplněn z poloviny, nastane přerušení a my můžeme hodnoty přečíst. To umožňuje průběžné rychlé čtení dat s menšími nároky na výkon počítače.

Karta je připojena na terminál (viz. Obrázek 2-4) společnosti Advantech, který slouží pro připojení sledovaných vodičů. V našem případě je použit terminál PCLD – 8710, který je ke kartě připojen přes stíněný komunikační kabel se SCSI konektorem a je možné jej připevnit na nosnou DIN-lištu. Terminál obsahuje šroubovací svorkovnice pro připojení vodičů a dva konektory pro připojení dvaceti pinových plochých kabelů pro digitální vstup/výstup. Pokud bychom chtěli vstupní analogový signál filtrovat, jsou na plošném spoji terminálu volné pozice pro součástky, ze kterých můžeme vytvořit dané filtry. Také zde nalezneme propojky, kterými je možné nastavit, zda budou vstupy klasické nebo rozdílové.

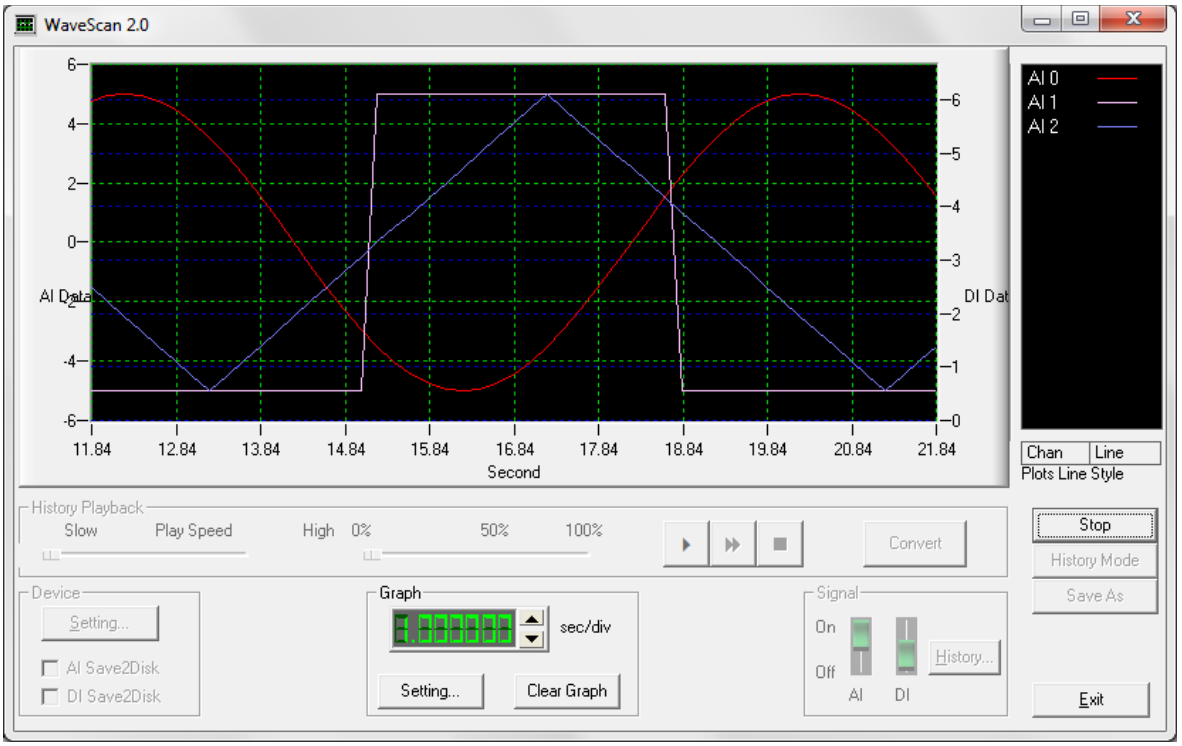
Obrázek 2-4- Terminál Advantech PCLD-8710

2.3 Dostupný software pro měřící karty

Základní software (viz. Obrázek 3-3), který je dostupný pro karty Advantech je aplikace obsažená přímo v softwaru ke kartě. Jedná se o jednoduchou aplikaci, která ukazuje hodnoty na analogových vstupech, kde si můžeme zvolit maximálně rychlost čtení

vzorků. Dále zvládá vysílat základní tvary signálů nebo jen určitou hodnotu na výstup s možnou volbou úrovně napětí a také se změnou délky periody vysílaného signálu. Digitální vstup je řešen barevnými indikátory, které zobrazují stav na vstupu. Digitální výstup se ovládá tlačítky.

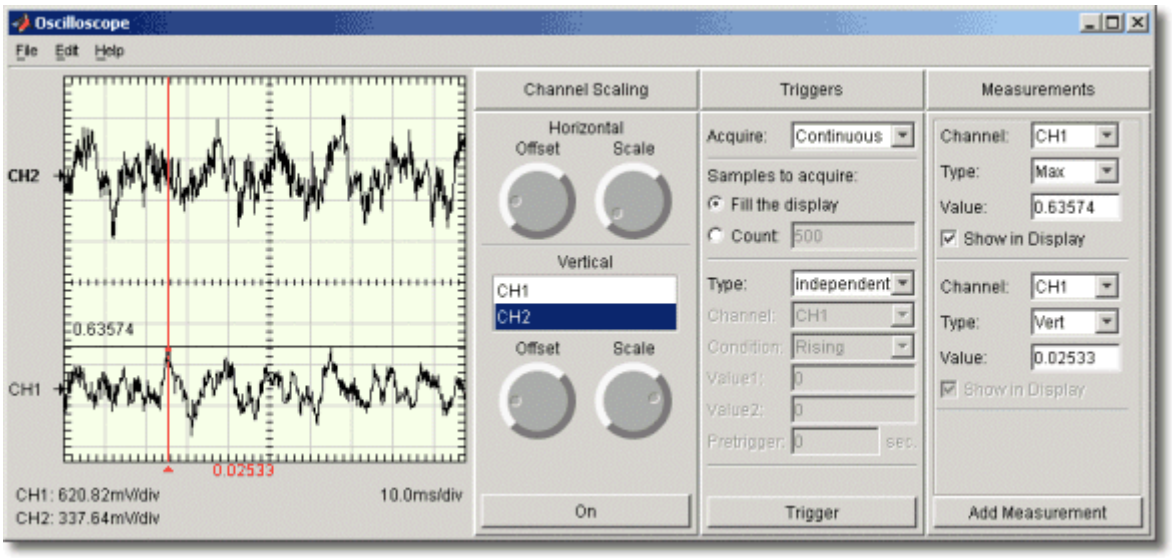
Společnost Advantech také nabízí pokročilou aplikaci WaveScan.2.0 (viz Obrázek 2-5) stažitelnou zdarma, která nabízí i kreslení do grafu. Je zde problém, že graf vykresluje všechny zvolené kanály najednou a nelze je jednotlivě vypínat a tak se graf stává nepřehledným. Dále je možné měnit velikost dílků u vykreslení, ale graf si pamatuje pouze hodnoty v právě viditelné oblasti, takže již nelze vidět předchozí průběh. Pokud někdo žádá vykreslování průběhu digitálních vstupů, tak je dobré, že i tyto data lze vykreslovat v plovoucím grafu. Výstupy zde však nenajdeme, aplikace se zaměřuje pouze na čtení signálů. Aplikace nabízí také uložení dat, ale data se mi z aplikace uložit nepodařilo.



Obrázek 2-5 – Aplikace Advantech WaveScan 2.0

Dále se můžeme obrátit na placený software, například na toolbox pro matlab jménem Data Acquisition Toolbox (viz. Obrázek 2-6). Část pro čtení vypadá podobně jako reálný osciloskop. Můžeme si zde volit posun jednotlivých kanálů a také signál různě zvětšovat popřípadě zmenšovat. Výhoda je v dobré provázanosti s matlabem a simulinkem,

kde můžeme rovnou využívat naměřených hodnot. Tento toolbox obsahuje kromě osciloskopu pro čtení také část pojmenovanou generátor, kde můžeme signály generovat a vysílat je na výstup karty.



Obrázek 2-6 – Osciloskop z Data Acquistion Toolbox

Obdobný je také toolbox pro matlab od společnosti Humusoft s názvem Real Time Toolbox. Tento toolbox stejně jako Data Acquistion toolbox umožňuje čtení analogových a digitálních dat z měřících karet v reálném čase. Má také provázání na simulink a tím pádem také možnost navrhovat řídicí systémy s použitím měřící karty.

2.4 Knihovny

Části programu, kde mohou být procedury, funkce, konstanty, definice datových typů a tříd, které mohou být sdíleny více programy.

Typy knihoven:

- Statické knihovny
- Dynamické knihovny (dynamic linking library – DLL)

Statické knihovny

Statické knihovny jsou spojovány linkerem v době stavění programu. Linker do výsledného spustitelného souboru vloží volané funkce. Statická knihovna je vlastně archivem jednoho nebo více objektových souborů.

Výsledkem je tedy jeden spustitelný soubor, který v sobě obsahuje část statické knihovny, která je nezbytná pro chod daného programu.

Dynamické knihovny

Při linkování programu s dynamickou knihovnou se do výsledného spustitelného souboru ukládají pouze tabulky odkazů na symboly definované v dynamické knihovně. Pro chod programu je pak nutno kromě vlastního programu mít na počítači nainstalovanou i příslušnou dynamickou knihovnu. Při spuštění programu pak operační systém provádí tzv. dynamické linkování. Během tohoto procesu operační systém načítá do operační paměti jak kód vlastního programu (spustitelný soubor), tak i kód dynamické knihovny, kterou program vyžaduje ke své činnosti.

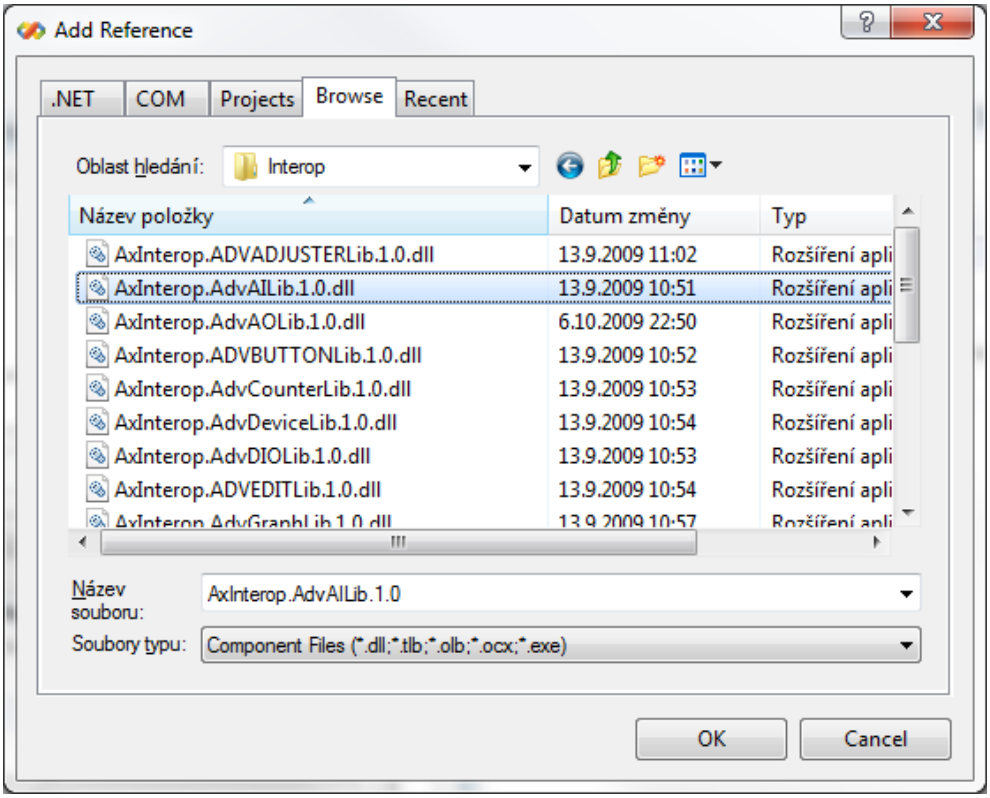
Dynamické knihovny je možné používat jako moduly k programu, kde bude každý modul v samostatné knihovně a poté v programu načítán. To, že je program rozdělen na moduly je výhodné i pro rychlost načítání programu, neboť daný modul můžeme načítat až v době, kdy je skutečně potřebován. Dále použití knihoven pomáhá odstranit duplicitní kód a tím můžeme ušetřit místo na disku. Při použití knihoven můžeme ulehčit případnou jednodušší aktualizaci pouze přepsáním staré knihovny verzí novou a není potřeba měnit kód programu.

3 Praktická část

3.1 Práce s DLL knihovnami

Knihovny společnosti Advantech jsou oddělené pro jednotlivé úkony. Zvlášť je knihovna pro analogový vstup, analogový výstup a poté je zde knihovna pro ovládání digitálních vstupů a výstupů. To je změna nových verzí oproti dřívějším, kde bylo ovládání analogových vstupů a výstupů v jedné knihovně.

Dodávané knihovny lze k projektu jednoduše připojit přes vizuální vývojové prostředí Visual Studia. Při otevřeném projektu si v okně najdeme Solution Explorer, kde vidíme zobrazeny jednotlivé soubory, které jsou součástí našeho projektu. Při kliknutím pravým tlačítkem myši na název projektu v tomto okně se zobrazí kontextové menu, v kterém nás zajímá položka References. Po otevření nového okna zvolíme možnost Add New Reference, kde si poté můžeme zvolit z několika možností. Nám se hodí možnost Browse, kde můžeme použít referenci na danou DLL knihovnu viz. Obrázek 3-1.



Obrázek 3-1 – Přidání reference na knihovnu

Knihovny od společnosti Advantech potřebují být zaregistrovány v systému, díky tomu může nastat problém, kdy je na počítači nainstalována starší verze knihoven. To může způsobit, že se při instalaci nové verze knihovny nezaregistrují. Při spuštění aplikace na jiném počítači než je ten vývojový pak může nastat problém s nenalezením knihoven a následnou chybou aplikace. Z tohoto důvodu je dobré udělat pro aplikaci instalační balíček, kde po instalaci zajistíme registraci potřebných knihoven do systému.

3.1.1 Funkce knihovny pro analogový vstup

Knihovna pro práci s analogovými vstupy má název AdvAILib.

Tabulka 3-1 – Základní atributy knihovny pro analogový vstup

Popis	**Název atributu**
Číslo zařízení	DeviceNumber
Název zařízení	DeviceName
Rychlost čtení	DataSampleRate
Počet vstupních kanálů	ChannelMaxNumber
Počet čtených kanálů	ChannelScanCount
Zvolený kanál	ChannelNow
Hodnota dat na vstupu	DataAnalog, DataDigital

Tabulka 3-2 – základní metody knihovny pro analogový vstup

Popis	**Název metody**
Výběr zařízení	SelectDevice
Čtení dat ze zásobníku	AcquireBulkData
Napěťové rozsahy	GetValuRange

3.1.2 Funkce knihovny pro analogový výstup

Knihovna obsahující prostředky pro práci s analogovým výstupem je pod názvem AdvAOLib a strukturou je velmi podobná knihovně pro analogový vstup.

Tabulka 3-3 – Základní atributy knihovny pro analogový výstup

Popis	**Název atributu**
Číslo zařízení	DeviceNumber
Název zařízení	DeviceName
Zvolený kanál	ChannelNow
Počet výstupních kanálů	ChannelMaxNumber
Počet kanálů, které mají vysílat	ChannelExportCount
Hodnota vysílaných dat	DataAnalog, DataDigital

Metody jsou u analogového výstupu obdobné jako u vstupu viz. Tabulka 3-2. U analogového výstupu je jen změna při výstupu dat přes zásobník, kde se metoda nazývá TransferBulkData.

3.1.3 Funkce knihovny pro digitální vstup a výstup

Knihovna obsluhující digitální linky v sobě zahrnuje jak obsluhu digitálních vstupů, tak výstupů. Knihovnu najdeme pod názvem AdvDIOLib.

Tabulka 3-4 – Základní atributy pro digitální vstup/výstup

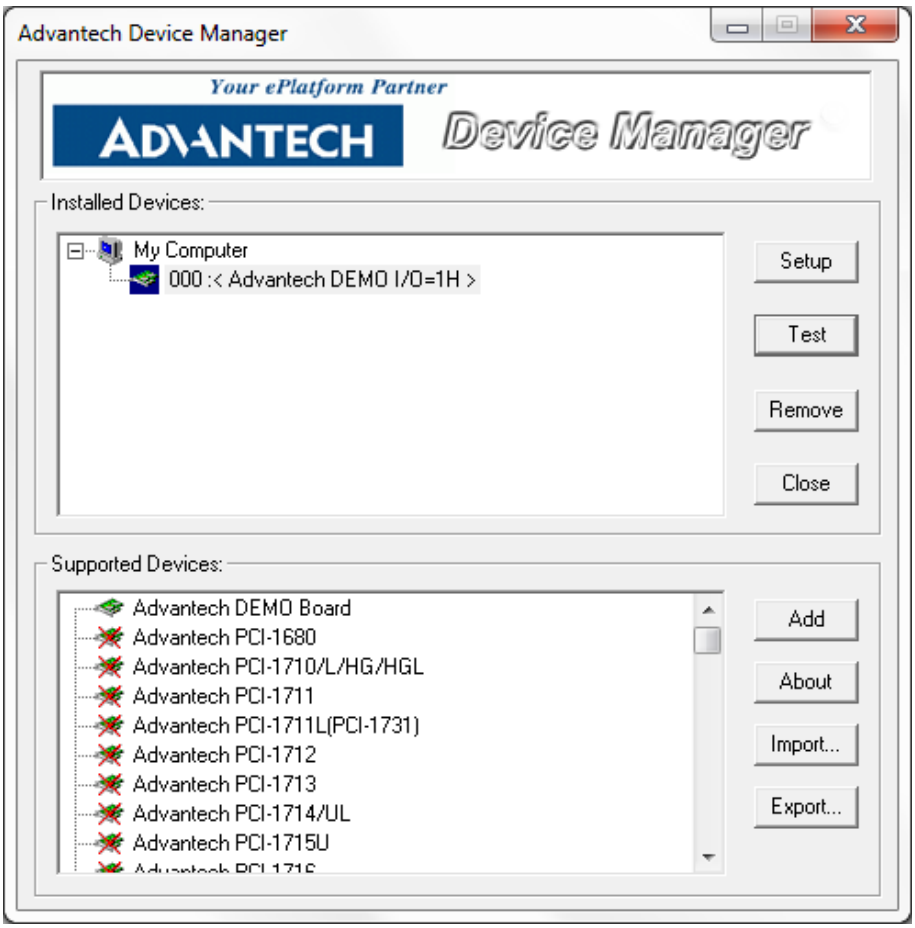
Popis	**Název atributu**
Číslo zařízení	DeviceNumber
Název zařízení	DeviceName
Počet vstupních portů	DiPortCount
Počet vstupních linek	DiChannelCount
Počet výstupních portů	DoPortCount
Počet výstupních linek	DoChannelCount

Tabulka 3-5 – Základní metody pro digitální vstup/výstup

Popis	**Název metody**
Výběr zařízení	SelectDevice
Čtení vstupní linky	ReadDiChannel
Čtení vstupních portů	ReadDiPorts
Čtení výstupní linky	ReadDoChannel
Čtení výstupních portů	ReadDoPorts
Zápis na výstupní linku	WriteDoChannel
Zápis na výstupní porty	WriteDoPorts

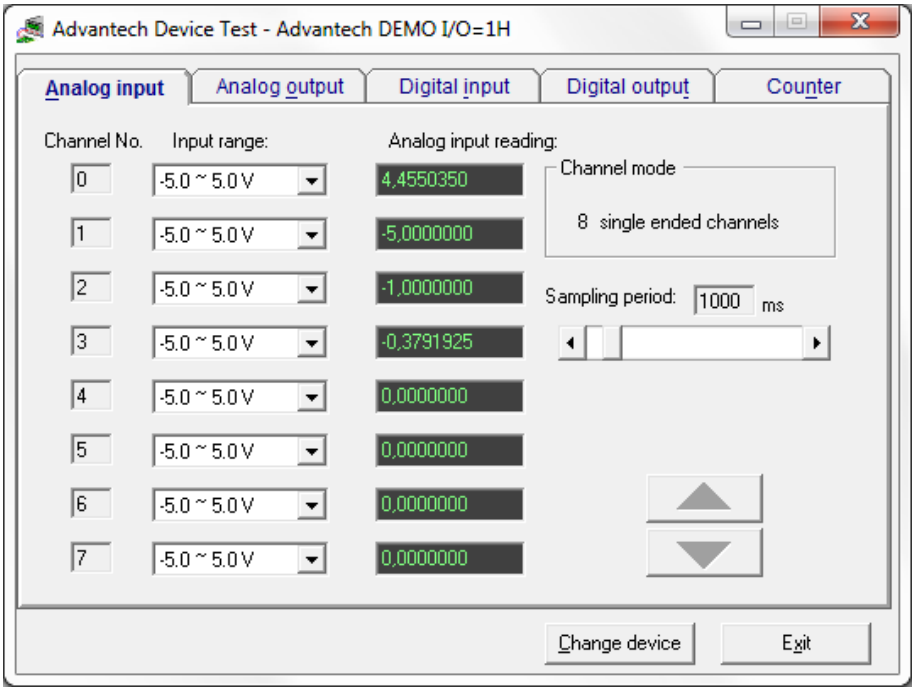
3.2 Potřebné softwarové vybavení pro běh aplikace

Před spuštěním aplikace je nutné nainstalovat na počítač potřebný software od společnosti Advantech. Jedná se o driver karty, který obsahuje také knihovny pro chod naprogramovaných aplikací. Dále můžeme nainstalovat balíček s Device managerem (Obrázek 3-2), kde vidíme karty, které jsou nainstalované v počítači. V mém případě je na počítači pouze DEMO karta, která slouží pro vývoj aplikace. Nepodporuje sice pokročilejší funkce, ale pro jednodušší programy je dostačující. Také si zde můžeme spustit jednoduchou kontrolní aplikaci (Obrázek 3-3). V ní se lze podívat na hodnoty vstupů a vysílat základní signály. Poté ještě můžeme nainstalovat balíček, jenž obsahuje různé ovládací komponenty, jako jsou tlačítka, displeje a podobné komponenty pro vývoj aplikací a ukázkové kódy jednoduchých programů. Ty nám ukazují jak nejlépe využívat dodávaných knihoven. Pro běh samotného programu je pak vyžadován .NET framework 3.5.



Obrázek 3-2 – Device manager od společnosti Advantech

Pokud nemáme v počítači nainstalovanou měřící kartu, máme k vývoji dostupnou virtuální DEMO kartu, kde jsou první čtyři vstupy generované signály (sinus, obdélník, pila a šum) a další čtyři vstupy slouží pro zobrazování hodnot, které zasíláme na výstup karty. Pokud stiskneme tlačítko pro vysílání digitální hodnoty, tak je i zde propojení s digitálním vstupem.

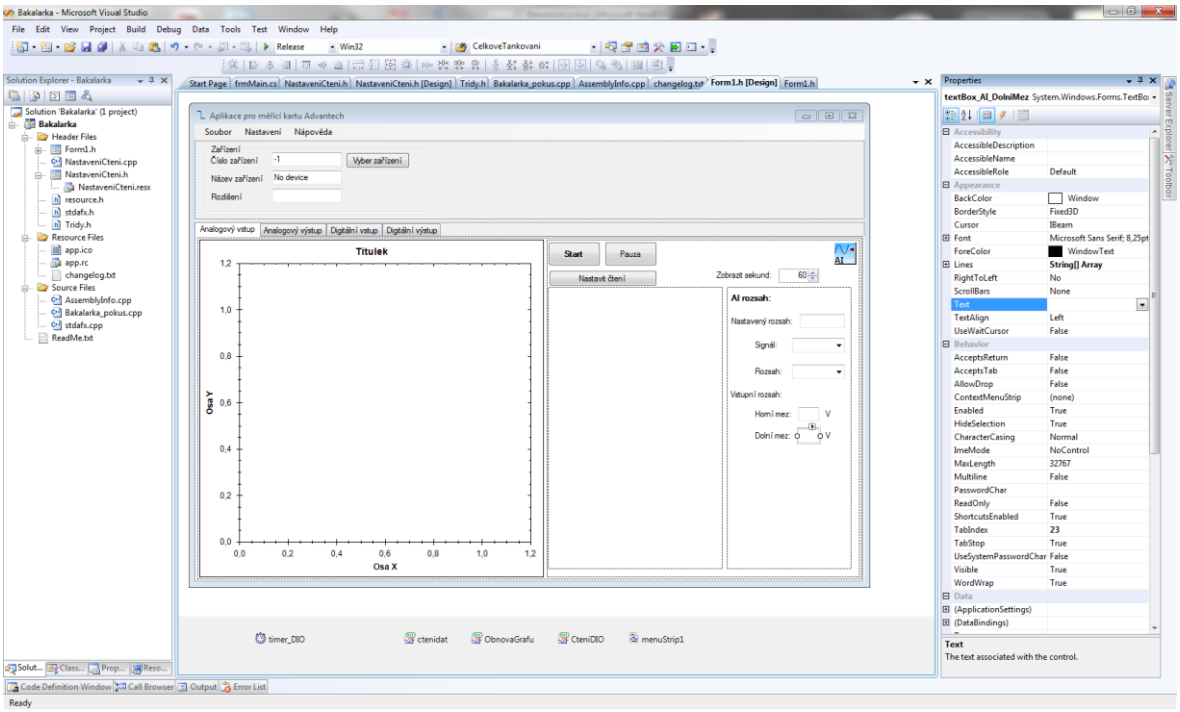


Obrázek 3-3 – Kontrolní aplikace

3.3 Vývojové prostředí

Jako prostředí pro vývoj mojí aplikace jsem zvolil Visual Studio 2008 od společnosti Microsoft. Programovací jazyk jsem zvolil dle dřívějších zkušeností Visual C++ .NET, který plně vyhovuje pro tvorbu aplikace a jeho syntaxe je velmi podobná jazyku C#, díky tomu, že oba jazyky využívají jako základ .NET. Také implementace ukázkových kódů nebyla díky podobnosti programovacích jazyků složitá.

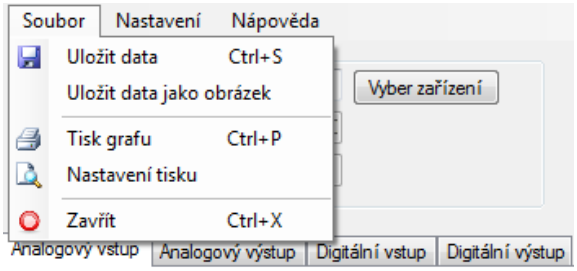
Pro moji aplikaci jsem zvolil projekt typu Windows Forms, kde mohu přehledně pracovat s komponentami a dalšími formuláři. Je zde jednoduchá správa komponent a dobrý přehled nad všemi součástmi programu. Komponenty lze jednoduše přidávat a měnit jejich parametry a vytvářet pro ně události, které mohou vyvolat.



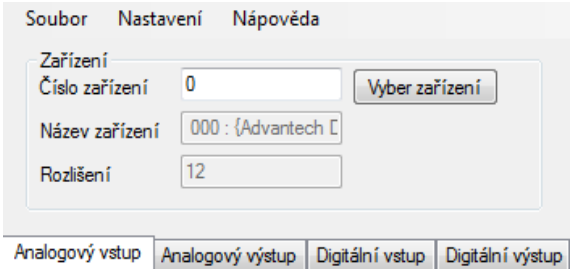
Obrázek 3-4 – Vývojové prostředí aplikace – Visual Studio 2008

3.4 Tvorba vlastní aplikace

Před začátkem psaní programu jsem nejdříve přemýšlel jak aplikaci vytvořit přehlednou a srozumitelnou pro uživatele. Rozhodl jsem se, že udělám čtyři záložky, kde v každé bude umístěna jedna z oblastí komunikace, tj. analogový vstup, analogový výstup, digitální vstup a digitální výstup. Nad těmito záložkami je umístěna informace o kartě, se kterou pracujeme a poté menu, které obsahuje základní funkce, jako uložení dat, uložení grafu jako obrázku, tisk grafu a podobné.



Obrázek 3-5 – Ukázka položek menu



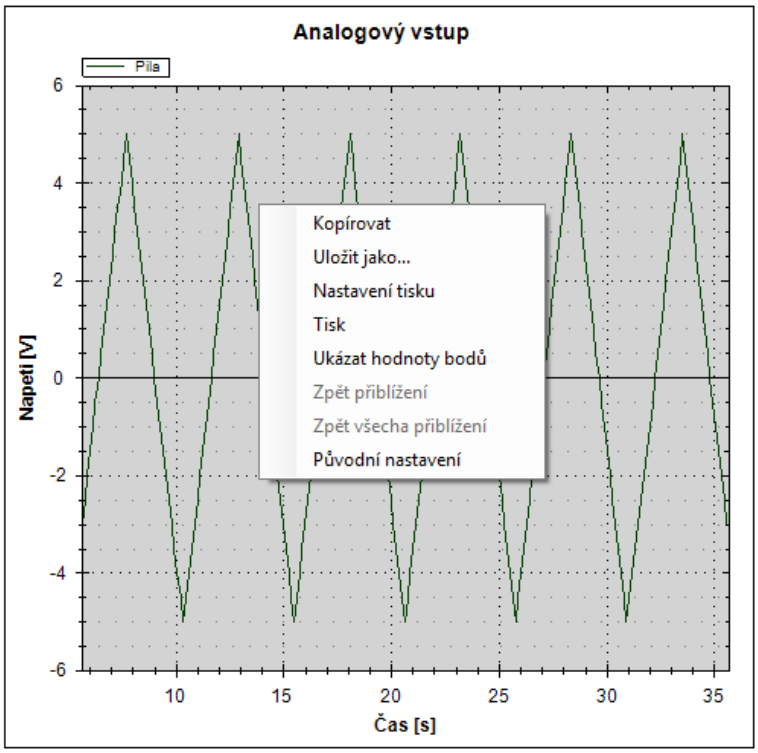
Obrázek 3-6 – Zobrazení používané karty

Když jsem měl vymyšlený návrh aplikace, bylo ještě potřeba rozmyslet její naprogramování. Nechtěl jsem, aby byla aplikace použitelná jen na kartě Advantech PCI-1711, když jsou knihovny univerzální. Udělal jsem proto načítání prvků dynamické podle typu karty. To si vyžádalo, že každý kanál analogového vstupu a výstupu bude tvořen vlastní třídou a podle počtu kanálů karty vytvořím potřebný počet instancí dané třídy. Každý analogový vstup obsahuje pole hodnot načtených ze vstupu a křivku, která je poté kreslena do grafu, nastavené rozsahy si není potřeba ukládat, neboť tento údaj jde získat pro každý kanál jednoduše z knihoven pro analogový vstup. Analogový výstup obsahuje informace o zvoleném typu výstupního signálu, pole bodů, které tvoří tvar vlastní funkce, hodnotu horní a spodní meze pro vysílané signály, délku periody vysílaného signálu a také informaci o tom, jestli chceme daný signál vyslat pouze jednou nebo jestli chceme jeho nepřetržité vysílání. Digitální vstupy a výstupy jsou načítané jednoduše přímo z knihoven a není důležité si u nich pamatovat nějaké informace a tím pro ně není potřeba tvořit samostatné třídy.

Pro zobrazování naměřených hodnot je použit volně dostupný projekt ZedGraph[4] (viz. Obrázek 3-7), se kterým mám již dřívější dobré zkušenosti. Popisky pro graf jsou původně v anglickém jazyce a tak jsem si stáhl zdrojové kódy a doplnil je o češtinu, aby

byla práce s grafem uživatelsky příjemnější. Pro moji aplikaci je výhodný v tom, že nabízí i jednoduše nastavitelný plovoucí graf, který je přesně tím, co je potřeba pro zobrazování analogového vstupu v reálném čase. Nabízí dále funkce uložení grafu jako obrázku, tisk grafu, přibližování v grafu výběrem myší a další.

```
//NASTAVENÍ GRAFU

//viditelnost jednotlivych os
zedGraphControl1->GraphPane->XAxis->MinorGrid->IsVisible = false;
zedGraphControl1->GraphPane->XAxis->MajorGrid->IsVisible = true;
zedGraphControl1->GraphPane->YAxis->MinorGrid->IsVisible = true;
zedGraphControl1->GraphPane->YAxis->MajorGrid->IsVisible = true;

//nastavení jednotné barvy pozadí
zedGraphControl1->GraphPane->Chart->Fill->Type = FillType::Solid;
zedGraphControl1->GraphPane->Chart->Fill->Color = Color::LightGray;

//popisky os grafu
zedGraphControl1->GraphPane->XAxis->Title->Text = "Čas [s]";
zedGraphControl1->GraphPane->YAxis->Title->Text = "Napeti [V]";
zedGraphControl1->GraphPane->Title->Text = "Analogový vstup";
```

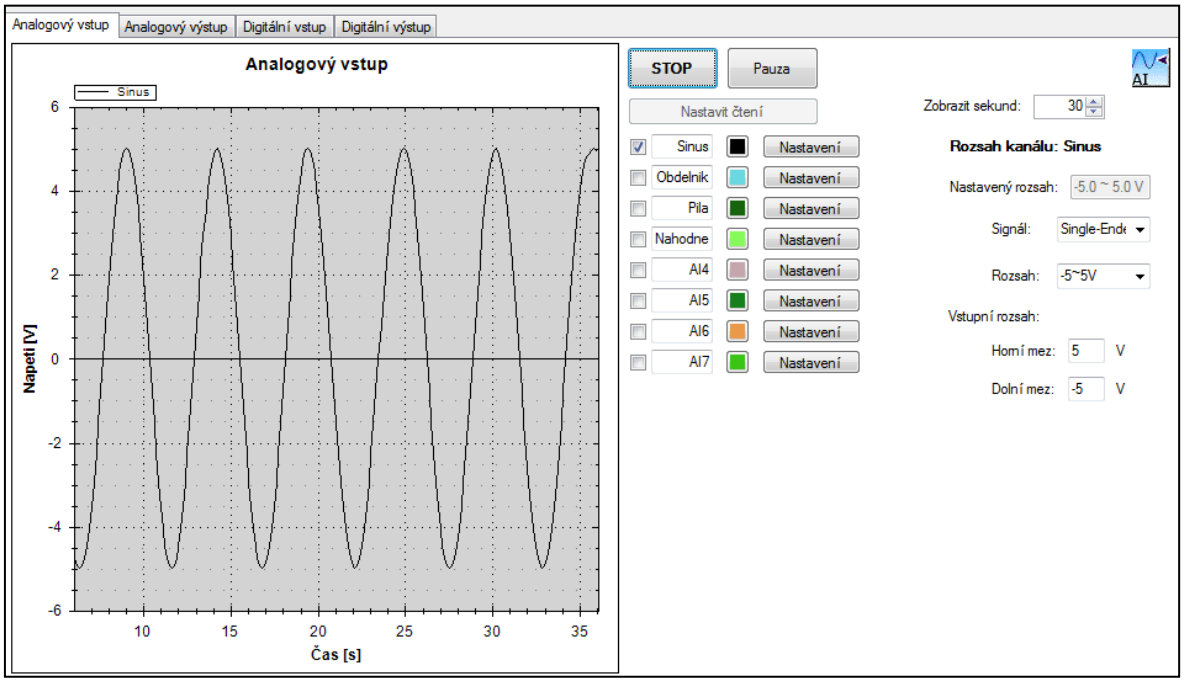


Obrázek 3-7 – Ukázka grafu a jeho možnosti

```
//NASTAVENÍ MAXIMÁLNÍ A MINIMÁLNÍ HODNOTY X PRO PLYNULÝ POHYB

//aktuální čas od začátku čtení dat
double time = (Environment::TickCount - tickStart) / 1000.0;

//změna maximální a minimální hodnoty na ose X
if ( time > zedGraphControl1->GraphPane->XAxis->Scale->Max)
{
      zedGraphControl1->GraphPane->XAxis->Scale->Max = time;
      zedGraphControl1->GraphPane->XAxis->Scale->Min = zedGraphControl1-
>GraphPane->XAxis->Scale->Max - pocetsekund;
}

//přizpůsobím Y osu datům
 zedGraphControl1->AxisChange();

//překreslení grafu
zedGraphControl1->Invalidate();
```

3.4.1 Analogové vstupy

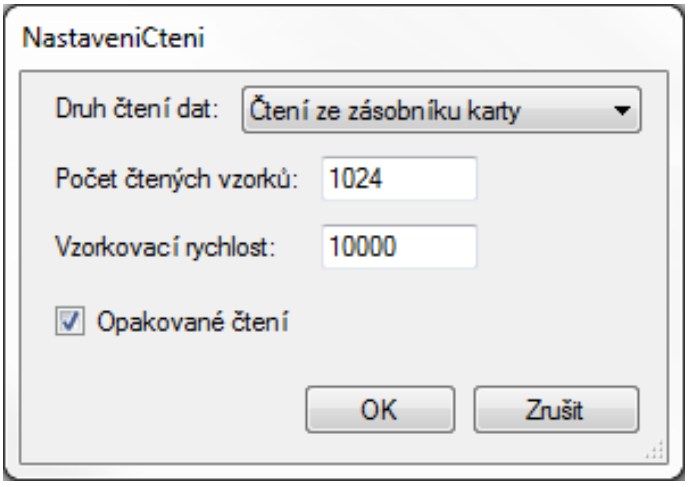
Jak bylo zmíněno výše, každý kanál z analogového vstupu je řešen svou instancí třídy AIkanal. Tato třída obsahuje pole naměřených hodnot a také svojí křivku, které je vykreslována do grafu, to mi umožňuje jednoduše zapínat a vypínat zobrazení naměřených hodnot pro jednotlivé vstupy.



Obrázek 3-8 – Ukázka analogového vstupu

Na obrázku Obrázek 3-8 je vidět celé uživatelské rozhraní pro analogový vstup. Vidíme zde graf, na kterém je vidět průběh signálu na prvním kanálu. Délku signálu, jenž bude vidět, lze jednoduše nastavit v poli „Zobrazit sekund“. Maximálně lze vidět pět set tisíc vzorků, poté se začnou hodnoty postupně přepisovat. Pro každý kanál si můžeme nastavit libovolnou barvu a název, který bude po přidání do grafu vidět v legendě grafu. Při stisku tlačítka Nastavení u kanálu se nám zobrazí nastavení rozsahu daného kanálu. Možnosti rozsahu zjišťuji přímo z měřící karty a tak je toto nastavení nezávislé na daném typu karty.

```
//ZÍSKÁNÍ ROZSAHU Z KARTY

//nastavení daného kanálu
Analog->ChannelNow = index;

//aktualizace textu zobrazujícího rozsah pro aktuální kanál
textBox_rozsah->Text = Analog->DataValueRange;

//získání rozsahů z karty, data se ukládají do Dolnimez a Hornimez
Analog->GetValueRange(index, Dolnimez, Hornimez, kodRozsahu);

//aktualizace hodnot pro dolní a horní mez
textBox_AI_HorniMez->Text = Hornimez.ToString();
textBox_AI_DolniMez->Text = Dolnimez.ToString();
```



Obrázek 3-9 – Nastavení čtení dat

Další možností, kterou při nastavování vstupu máme, je nastavení typu čtení dat. Můžeme si vybrat mezi jednoduchým čtením, kde načítám data knihovny pomocí DataAnalog v časových intervalech. Čtení probíhá pomocí vlákna pracujícího na pozadí a

přerušovaného na daný okamžik. Stejným způsobem v dalším vlákně po daných časových okamžicích, které určuji pomocí příkazu Sleep(časový interval v ms) překresluji graf. Druhou možností čtení je čtení ze zásobníku karty, to můžeme vidět na obrázku Obrázek 3-9. Lze zde nastavit počet načítaných vzorků. Při rychlosti vzorkování nižší než dva tisíce vzorků za sekundu je možné nastavit počet čtených vzorků od hodnoty 32 a při vyšší rychlosti čtení je minimální hodnota 1024 vzorků. Dalším nastavitelným parametrem je vzorkovací rychlost, která lze volit podle daného typu karty. V mém případu lze volit až sto tisíc vzorků za vteřinu. Při tomto nastavení je možné, při provozování karty na slabším počítači, vysoké zatížení systému. Poslední volbou je zaškrtávací možnost opakovaného čtení, které zajistí opakované měření, zatímco při vypnutém opakování se načte jen počet vzorků uvedený v nastavení.

```
//ČTENÍ DAT ZE ZÁSOBNÍKU

//nastavení rychlosti čtení
Analog->DataSampleRate = RychlostCteniAI;

//nastavení, která data budu číst, v mém případě Analogová data
Analog->DataReturnType = 2;

//vyprázdnění zásobníku na naměřená data
BufferCteniAI = nullptr;

//spuštění čtení dat ze zásobníku, bude se číst PocetDatAI do
BufferCteniAI
PocetDatAI = Analog->AcquireBulkDataToMemory(PocetDatAI, BufferCteniAI, -
1, OpakovaniCteniAI, false);
```

Při začátku čtení vždy vynuluji všechny předchozí načtené hodnoty a poté ukládám načtené hodnoty do polí, která jsou v daných AI kanálech. Ukládání se liší podle vybraného způsobu čtení dat. Při jednoduchém čtení ukládám data v daných časových intervalech, zatímco při čtení dat ze zásobníku ukládám data při události, jenž je vyvolána při polovičním popřípadě úplném zaplnění zásobníku.

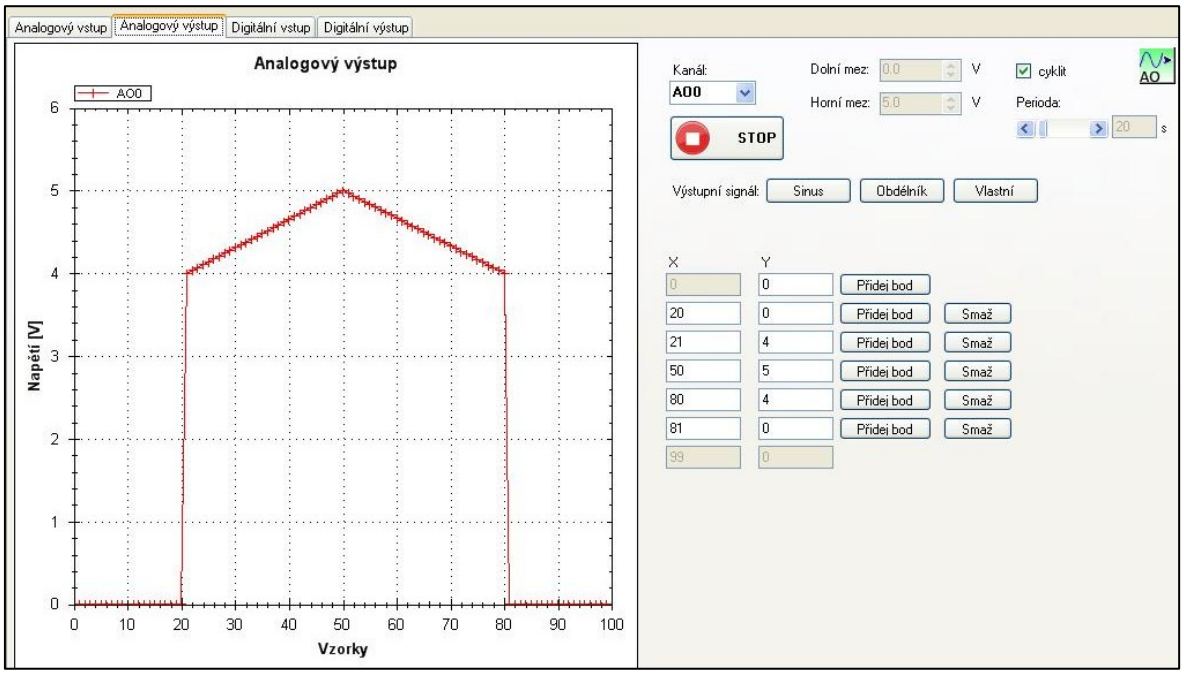
```
//ZÍSKÁVÁNÍ DAT PŘI JEDNODUCHÉM ČTENÍ

//pro všechny kanály ukládám jejich hodnotu a daný čas
for(int i=0; i<Analog->ChannelMaxNumber; i++){
      //nastavení aktuálního kanálu
      Analog->ChannelNow = i;
      //uložení dat do daného AI kanálu
      AIkanaly[i].DejList()->Add(time , Analog->DataAnalog);
}
```

```
//PŘI POLOVIČNÍM ZAPLNĚNÍ ZÁSOBNÍKU ULOŽÍM DATA DO AI KANÁLU

//při zavolání události získávám pomocí parametru e základní informace o
zásobníku
long PocetDat = e->dataCount;
//získáme hodnoty ze zásobníku
Object^ PoleDatBufferu = e->analogArray;

//jestli není pole prázný načteme hodnoty
if ( PoleDatBufferu != nullptr)
...
//aktuální čas od začátku čtení dat
double time = (Environment::TickCount - tickStart) / 1000.0;

//uložení hodnoty do daného AI kanálu
for (int i=0 ; i<Analog->ChannelMaxNumber ; i++ )
{
      AIkanaly[i].DejList()->Add(time, hodnoty[i]);
}
```

3.4.2 Analogové výstupy

Při tvorbě analogových výstupů jsem se řídil zadáním a vytvořil jsem prostředí, kde je možné vybrat si ze dvou předdefinovaných signálů, které jsou sinus a obdélník. Třetí volbou je možnost vytvořit si vlastní libovolný signál zadáním bodů signálu (Obrázek 3-10). Zvolený signál se vykresluje v grafu, který je součástí záložky s analogovým výstupem. Pokud si zvolíme předdefinovaný signál, můžeme si zde navolit horní a dolní mez signálu, opakování signálu a periodu. Výstupní signál je tvořen sto vzorky, které se vysílají po dobu určenou periodou a případně se opakují stále dokola. Při zvolení vlastního signálu se vytvoří první a poslední bod signálu, čímž můžeme vytvořit konstantní hodnotu nebo můžeme pomocí tlačítka přidat další body, kde zadáme hodnotu x a y pro bod našeho signálu.

Při vývoji jsem používal virtuální DEMO kartu ze softwaru Advantech, kde jsou výstupy navedeny na vstup a tak lze jednoduše vidět, jestli je generovaný signál podle našich představ.



Obrázek 3-10 – Analogový výstup s ukázkou libovolného signálu

Vytvořený signál je při změně ukládán do pole daného kanálu. Vysílání samotného signálu je řešeno pomocí komponenty backgroundworker, ve které testuji, zdali je nějaký kanál určen k vysílání a zdali ano, tak je podle jeho periody signál uložený v poli vysílán na výstup karty, kde je po vyslání následně testováno, zdali se má vysílat stále dokola.

```
//VYSILANI AO DAT

...
//jesli je jiz cas na vyslani dat, tak vyslu
Analogvyst->DataAnalog = AOkanaly[i].DejBody()[pocetVyslanych[i]].DejY();
...
//pokud je vyslan cely signal kontrola jestli opakovat
...
vysilaneAOkanaly[i] = false;
//pokud jiz ne, zmenim tlacitko
button->Text = "Vysílat";
button->Image =  imageList->Images[0]; //zobrazim tlacitko play
```

3.4.3 Digitální vstupy/výstupy

Digitální vstup a výstup je řešen velmi podobným způsobem. Podle údajů z knihovny pro digitální vstup a výstup načítám počet portů a daný počet kanálů pro daný port. Načítání je tedy dynamické a načítání portů a kanálů probíhá podle dané karty.

Obrázek 3-11 – Digitální vstup a výstup

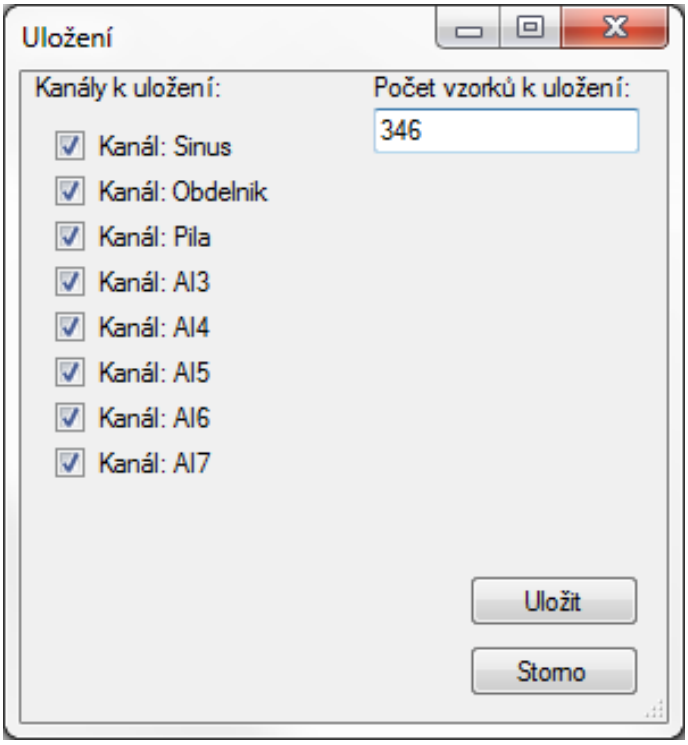
Jednotlivé kanály jsou reprezentovány tlačítky, v případě vstupu jsou použity jen jako zobrazovací prvek a nejsou aktivní. Jako indikaci jednotlivých stavů měním barvu na zelenou v případě logické jedničky nebo na červenou při logické nule. Výstup je realizován stejným způsobem, jen jsou tlačítka aktivní a můžeme jimi ovlivnit výstupní signál. V mém případě je realizace tlačítky dostačující, program je zaměřen hlavně na čtení analogových hodnot a tak nebylo potřeba dodělávat například grafické průběhy vstupních signálů.

Při čtení dat jsem chtěl nejdříve využít potenciálu knihoven, tím že bych využil události, které jsou volány při změně stavu na portu. Jedná se o událost onDiStatusChange, která reaguje na změnu v rámci celého portu. Aby mohla takováto událost nastat, musíme ji povolit metodou EnableDiStatusChange, která je obsažena v knihovně pro digitální vstup a výstup. Jenže při pokusu zprovoznit tuto funkci na kartě se ukázalo, že karta Advantech PCI-1711 touto funkcí nedisponuje a bylo nutné řešit čtení jiným způsobem. Bylo tedy nutné přistoupit k testování portů v časové smyčce. Zjišťuji tedy stavy jednotlivých kanálů a podle jejich hodnoty nastavuji barvu tlačítek přibližně třikrát za vteřinu.

```
//ZMĚNA BARVY DI TLAČÍTEK PODLE ZMĚNY NA DIGITÁLNÍM VSTUPU

//pro všechny DI kanály čtu jejich stav
for(int i=0; i<Digital->DiChannelCount; i++){
result = Digital->ReadDiChannel(i);
//zjistím pozici prvku, který má na zobrazuje daný kanál
j = i/8; //řádek
k = i%8; //sloupec

//podle řádku a sloupce najdu daný prvek a nastavím barvu
...
//podle stavu kanálu nastavím barvu
if(result == 1){
      ctrl->BackColor = BackColor.Green;
}else if(result == 0){
      ctrl->BackColor = BackColor.Red;
}
```

3.4.4 Ukládání dat do souboru

Pokud chceme načtená data uložit, musíme zvolit danou volbu v hlavním menu. Po kliknutí se zobrazí nový formulář (Obrázek 3-12). V tomto formuláři mohu vybírat, z jakých kanálů se mi uloží data do souboru a poté kolik vzorků chci uložit. Jako typ souboru si mohu zvolit typ CSV (comma-separated values – hodnoty oddělené středníkem), který je vhodný tím, že je možné jej lehce otevřít a pracovat s ním v excelu a podobných tabulkových procesorech. Další volbou je uložení dat to obyčejného textového souboru, kde jsou data oddělená tabulátorem a poslední možností je také uložení do textového souboru. Data jsou ale prezentována s desetinou tečkou, aby šla importovat do prostředí matlabu a dále s nimi pracovat. Této změny jsem dosáhl změnou prostředí na „en-US“ ve funkci CultureInfo().



Obrázek 3-12 – Možnosti uložení dat

Počet vzorků pro uložení je nastaven na aktuální počet vzorků v paměti a maximálně může tato hodnota dosahovat hodnoty 500 tisíc vzorků. Názvy kanálů jsou zobrazovány a ukládány podle nastavených názvů v záložce s analogovými vstupy.

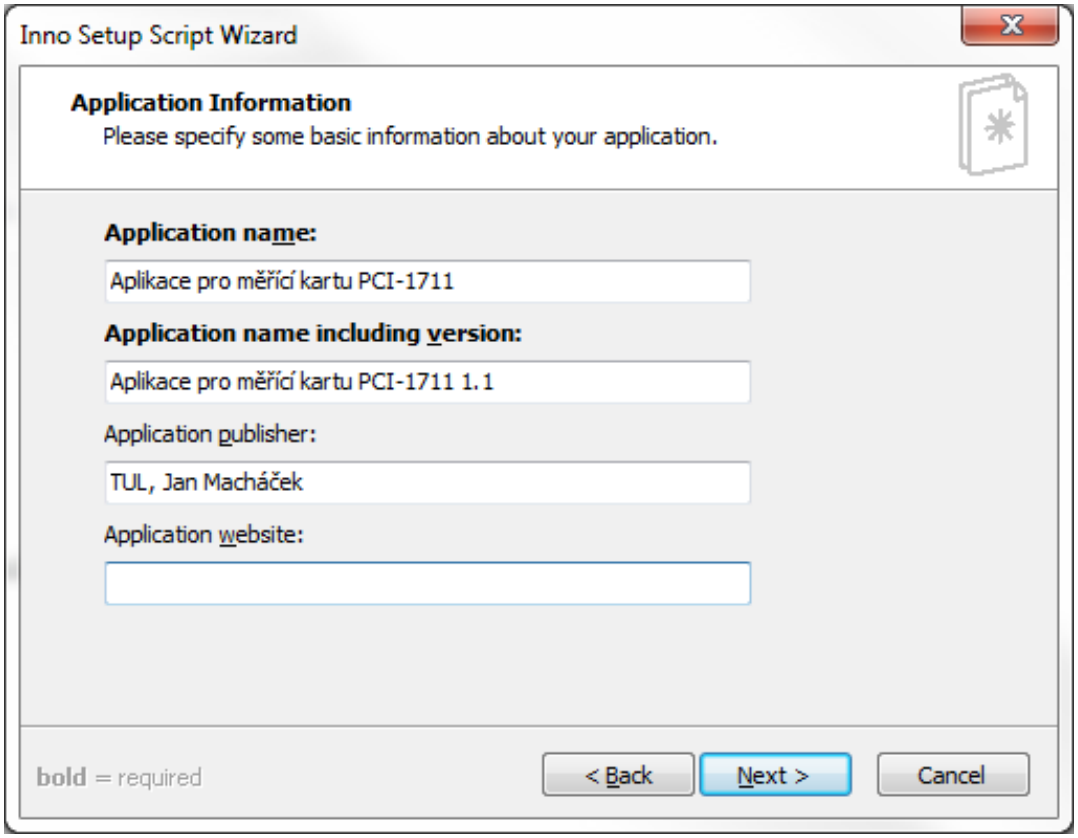
```
//NASTAVENÍ DIALOGU PRO ULOŽENÍ DAT

//dialog pro uložení dat do souboru
SaveFileDialog^ uloz = gcnew SaveFileDialog();

//nastavení parametrů pro ukládání
uloz->Filter = "CSV soubor (*.csv)|*.csv|Textový soubor
(*.txt)|*.txt|Textový soubor pro MATLAB (*.txt)|*.txt|Všechny soubory
(*.*)|*.*";
uloz->FilterIndex = 1;
uloz->RestoreDirectory = true;
```

Samotný zápis probíhá pomocí souborového proudu StreamWriter, kde poté používám funkce pro normální zápis tj. metoda Write() a pro zápis řádku tj. metoda WriteLine(). Abychom mohli data dále používat, je dobré mít s jednotlivými hodnotami uložen také název daných sloupců a časový údaj, kdy byly hodnoty pořízeny.

```
//UKLÁDÁNÍ JEDNOTLIVÝCH HODNOT Z VYBRANÝCH KANÁLŮ DO SOUBORU

//zápis názvů pro jednotlivé sloupce
zapis->Write("Cas"+oddelovac);
for(int j=0; j<PoleKanalu->Count; j++){

//získání všech kanálů, které jsem v předchozím formuláři označil k
uložení
...
 //zápis jednotlivých názvů s oddělovačem dat, který je dán typem souboru
zapis->Write(PoleKanalu[j].DejKrivku()->Label->Text+""+oddelovac);
...
//zapsani zalomení řádku
zapis->WriteLine("");

//zapsání počtu hodnot, které jsme uvedli v předchozím formuláři
for(int i=0; i<pocethodnot; i++){
//zapsání času dané hodnoty
zapis->Write(PoleKanalu[0].DejList()[i]->X+oddelovac);
for(int j=0; j<PoleKanalu->Count; j++){

//získání všech kanálů u kterých chci uložit data
...
//zapsání hodnoty daného kanálu + daný oddělovač podle typu souboru
zapis->Write(PoleKanalu[j].DejList()[i]->Y+oddelovac);
...
//zalomení žádku
zapis->WriteLine("");

...

//zapsání hodnoty daného kanálu při výběru uložení dat pro matlab

zapis->Write(PoleKanalu[j].DejList()[i]->Y.ToString(gcnew

CultureInfo("en-US"))+oddelovac);
```

3.5 Tvorba instalace

Pro tvorbu instalace jsem zvolil volně stažitelné prostředí Inno setup. Prostředí slouží k tvorbě instalačních průvodců pro operační systémy Microsoft Windows. Pokud neovládáme skriptovací jazyk pro vytvoření instalace, lze si spustit přehledného průvodce tvorbou instalace, kde postupně nastavujeme potřebné položky.



Obrázek 3-13 – Instalační průvodce Inno setup

V průvodci jdou nastavit základní postačující vlastnosti, pokud ale chceme například ověřit nainstalování potřebných aplikací, knihoven a podobně, musíme si toto ověření pomocí skriptu dopsat. Takové ověření je dobré například pro ověření nainstalovaného .NET framework balíčku, bez kterého bychom naší aplikaci nespustili. Je vhodné integrovat do instalačního balíčku také knihovny potřebné pro běh programu napsaného v jazyce visual C++. Tyto knihovny ve formě balíčku nabízí přímo společnost Microsoft. V mém případě je také potřeba zajistit registraci používaných knihoven společnosti Advantech, aby byly obsazeny v systému a aplikace s nimi mohla pracovat.

```
// Kontrola jestli je v pocitaci .NET framework ve verzi 3.5
isInstalled := 0;
//zjistovani zdali je .NET framework nainstalovan
netFrameworkInstalled := RegQueryDWordValue(HKLM, 'SOFTWARE\Microsoft\NET
Framework Setup\NDP\v3.5', 'Install', isInstalled);

if ((netFrameworkInstalled)  and (isInstalled <> 1)) then
netFrameworkInstalled := false;

if netFrameworkInstalled = false then   //pokud neni nainstalovan,
zobrazim informaci o jeho nepritomnosti a moznosti k jeho stazeni
  begin
    if (MsgBox(ExpandConstant('{cm:dotnetmissing}'),
        mbConfirmation, MB_YESNO) = idYes) then
    begin
//pokud souhlasim se stazenim, tak se odkaze na tuto adresu s .NET
framework 3.5
ShellExec('open',' adresa na .NET 3.5 ','','',...
```

```
//Registace a přidání knihoven do systému

Source: "C:\...\AdvAI.dll"; DestDir:"{sys}"; Flags: restartreplace
sharedfile regserver
Source: "C:\...\AdvAO.dll"; DestDir:"{sys}"; Flags: restartreplace
sharedfile regserver
Source: "C:\...\AdvDIO.dll"; DestDir: "{sys}"; Flags: restartreplace
sharedfile regserver
```

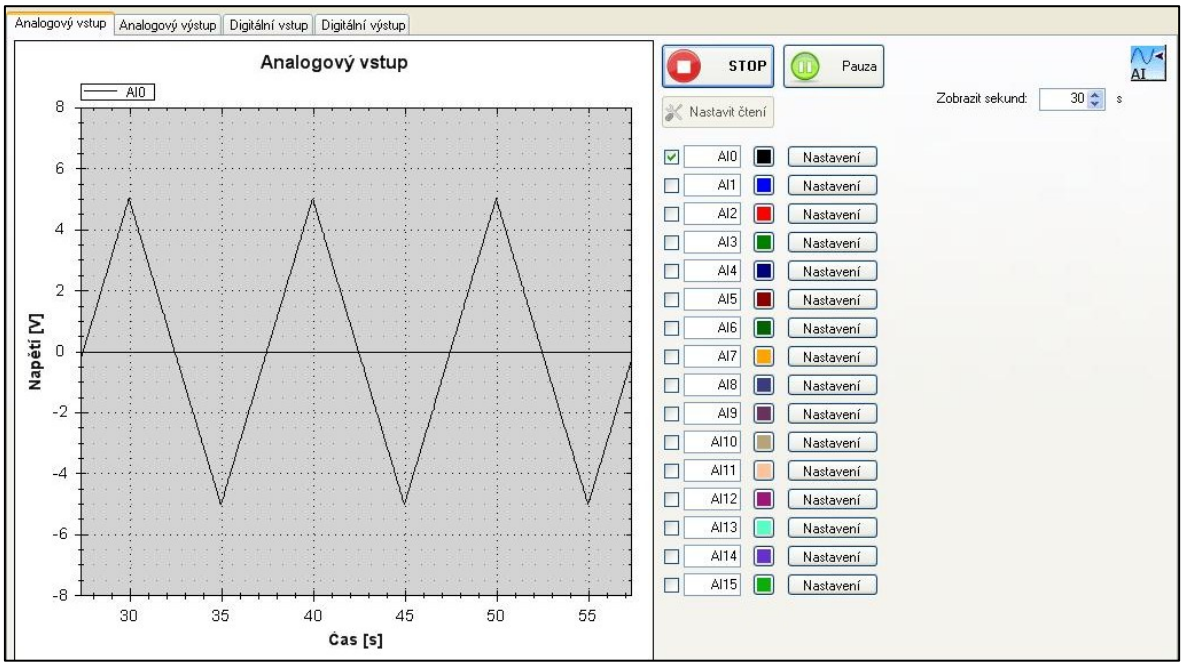
Ve výsledku získáme instalační soubor spustitelný na všech verzích Microsoft Windows od verze 95, který můžeme dále publikovat bez dalších obtíží. Instalace také podporuje i následné odebrání aplikace ze systému.

3.6 Test aplikace

Po vytvoření aplikace bylo potřeba zjistit, jestli software získává z karty správné údaje a zdali nějakým způsobem nedeformuje přijímaný signál. Pro tento test mi posloužil generátor signálů HP33120A který jsem připojil přes terminál karty (viz. Obrázek 2-4). Zkoušel jsem různá nastavení generátoru pro ověření funkčnosti karty v celém jejím rozsahu. Měnil jsem frekvenci, velikost napětí a při tom měnil druhy výstupních signálů. Dostupné tvary byly sinus, obdélník, trojúhelník a pila. Napětí bylo voleno v rozmezí 50mv až 10V, což je zároveň maximální napětí pro analogový vstup karty. Frekvence byla volena v rozsahu 0,1Hz až 10kHz.

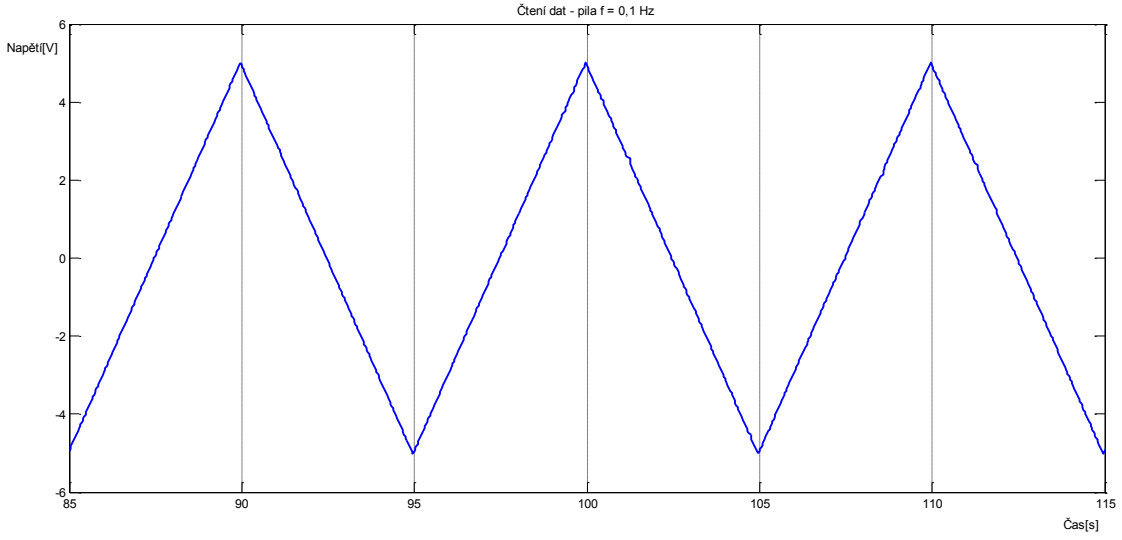
Aplikaci jsem nejdříve testoval při čtení jednoduchém, kdy je hodnota na vstupu získávána pomocí knihovny analogového vstupu, který se v časové smyčce čte. Při jednoduchém čtení jsem musel nastavit rychlost čtení na 50 vzorků za vteřinu, aby aplikace zvládala zobrazit správně i rychlejší průběhy, ale v tomto případě si aplikace vyžádala mnoho systémových zdrojů a tak byla časová základna nestabilní a pokud systém potřeboval vykonávat i jinou činnost, byl signál deformován. Pokud bychom chtěli měřit pomalejší průběhy a nastavili bychom pomalejší čtení dat, tak je možné toto jednodušší měření využít.

Díky problémům s časovou základnou a náročností rychlejšího čtení u jednoduchého čtení, bylo hlavní prioritou, aby aplikace uměla číst data i ze zásobníku umístěného na kartě. Po úspěšné implementaci čtení ze zásobníku jsme toto měření otestovali také pomocí generátoru signálů jako při čtení pomocí časové smyčky a bylo vidět výrazné zlepšení v časové přesnosti a v malém zatížení systému. Tento způsob měření zobrazuje data velmi přesně a plně vyhovuje pro náročnější měření. Při testu odchylky rychlosti vzorkování jsme nastavili rychlost čtení na 2 kHz a na generátoru jsme nastavovali frekvenci takovou, abychom dosáhli vodorovného průběhu signálu (ideálně také 2kHz). Toho jsme dosáhli při odchylce 2,7 mHz. Pokud chceme mít výstup co nejpřesnější, doporučuji však zobrazovat průběh jen potřebných kanálů, neboť vykreslování více vstupních kanálů v grafu zároveň, zatěžuje systém.



Obrázek 3-14 – Načtený signál z analogového vstupu

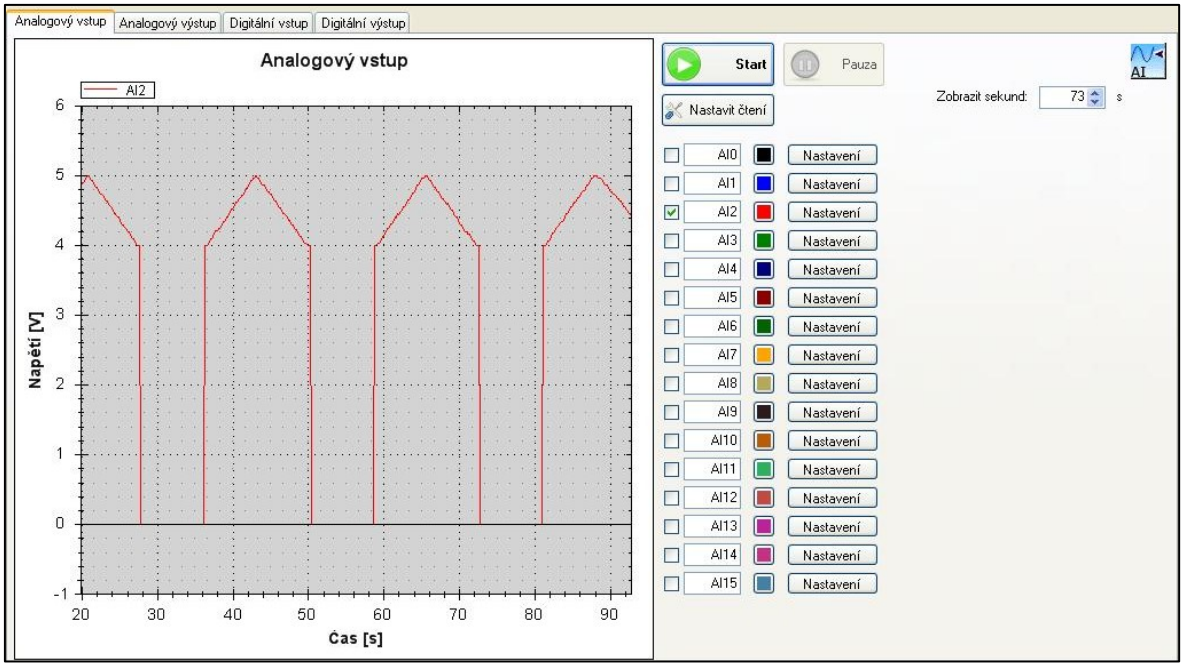
Načtený signál jsem uložil do souboru pro matlab a vykreslil ho viz. Obrázek 3-15, pro kontrolu přesnosti čtení dat. Z grafu je dobře patrné, že časová základna je přesná a signál odpovídá parametrům nastaveným na generátoru (U = $5V_{p-p}$ a f = 0,1 Hz).



Obrázek 3-15 – Načtený signál zobrazený pomocí matlabu

Analogový výstup jsem testoval propojením výstupní svorky terminálu s analogovým vstupem a tak jsem mohl rovnou pozorovat generovaný signál v aplikaci

viz. Obrázek 3-16. Generovaný signál má přesný tvar, ale jeho časová základna je díky vysílání signálu v daných časových intervalech nestabilní a tak nastavená perioda pro vysílaný signál přesně nesouhlasí s vysílaným signálem.



Obrázek 3-16 – Zobrazení vysílaného signálu

Protože je na digitálních vstupech hodnota logické jedničky, je proto nutné vstupy uzemňovat. Takže jsem si při testování vstupů vystačil s propojovacím drátkem, kde byl jeden konec připojen do svorkovnice a uzemněn a druhý jsem připojoval na jednotlivé vstupy, tím je uzemňoval a kontroloval zobrazování stavu v aplikaci.

Digitální výstup jsem testoval připojením LED diody do svorkovnice, která je umístěna na terminálu (viz. Obrázek 2-4) připojeném k měřící kartě. Tím, že karta pro digitální výstup používá úroveň TTL, je možné k testu použít klasickou LED diodu. Diodu jsem tedy připojoval do svorkovnice na digitální výstupy a poté jsem zapínal jednotlivé kanály v aplikaci a pozoroval rozsvěcení diody. Tím jsem zjistil správnou funkčnost digitálního výstupu.

4 Závěr

Cílem práce bylo vytvoření aplikace pro měřící kartu, konkrétně pro kartu společnosti Advantech, model PCI-1711. Aplikace měla využívat možností knihoven dodávaných od výrobce karty. Důraz byl kladen hlavně na provedení analogových vstupů a výstupů.

Při realizaci práce nastalo pár problémů, nejdříve byl problém s provozem aplikace na jiném počítači, kvůli registraci knihoven a dalším problémem bylo správné zprovoznění čtení dat ze zásobníku karty. Vše se nakonec podařilo správně vyřešit. Registrace knihoven je nyní zajištěna v instalaci aplikace a čtení ze zásobníku karty funguje velmi dobře a tím je měření dat výrazně přesnější nežli s měřením v časové smyčce.

Aplikaci jako takovou považuji za ucelenou. Pokud by ale bylo potřeba, šlo by dále rozšířit možnosti u digitálních vstupů. Také by šlo vytvořit graf a mohli bychom mít jakýsi logický analyzátor, kde bychom viděli průběhy jednotlivých digitálních kanálů.

Pokud bychom chtěli mít všechny moduly přesné jako analogové čtení přes zásobník, bylo by potřeba mít k dispozici lepší měřící kartu, takovou, která by zvládala vysílat signál na výstup karty také pomocí zásobníku a měla by detekci změny stavu na digitálních linkách. Poté by se již nemuselo používat čtení, popř. vysílání dat v časových smyčkách a měření a vysílání dat by se stalo přesnější a méně náročnější na systém.

Seznam literatury

[1] ŠTEFAN, Radim. Měřící karty - jak správně vybírat. *Automa.* [Online] Č. 7 2004. [Citace: 2. duben 2010.] Dostupný z WWW: <http://www.odbornecasopisy.cz/index.php?id_document=32427>.

[2] Datasheet PCI-1711. *www.advantech.cz.* [Online] [Citace: 28. březen 2010.] Dostupný z WWW: <http://origindownload.advantech.com//ProductFile/1-32A8K9/PCI-1711_DS.pdf>.

[3] *PCI - 1711/1731 User's Manual.* 1999. str. 60.

[4] ZedGraph. [Online] 29. listopad 2007. [Citace: 8. říjen 2009.] Dostupný z WWW: <http://zedgraph.org>.

[5] Microsoft Developer Network. *MSDN.* [Online] [Citace: 26. březen 2010.] Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/default.aspx>.