Česká zemědělská univerzita v Praze

Technická fakulta

Katedra jakosti a spolehlivosti strojů

Návrh a realizace softwaru pro univerzální měřící jednotku

Diplomová práce

Vedoucí diplomové práce: Ing. Bohuslav Peterka, Ph.D.

Autor práce: Jaromír Holec

PRAHA 2014

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra jakosti a spol. strojů

Technická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Holec Jaromír

Informační a řídící technika v agropotravinářském komplexu

Název práce

Návrh a realizace softwaru pro univerzální měřící jednotku

Anglický název

Development of software for universal measurement device

Cíle práce

Student se detailně seznámí s kostrukcí, funkcemi a firmwarem měřící jednotky používané zejména pro měření výkonu akcelerační metodou. Následně navrhne a ve vhodně zvoleném vývojovém prostředí realizuje modul (knihovnu) pro komunikaci s jednotkou a sběr měřených dat a prověří jeho funkčnost.

Metodika

Student vyhledá odbornou literaturu a související informační zdroje k tématu práce. Student zpracuje literární rešerši, v které zmapuje současný stav poznání v oblasti zadané problematiky. Dále bude zvolen vhodný programovací jazyk a vývojové prostředí, v kterém bude realizována univerzální knihovna pro obsluhu měřícího zařízení. Navržené řešení bude řádně okomentováno a všechny implementované funkcionality budou řádně otestovány.
Student se seznámí s pokyny Technické fakulty pro vypracování a odevzdání dilomové práce. Student bude pracovat systematicky a průběžné výsledky bude konzultovat s vedoucím práce. Student vypracuje čistopis práce a ten v řádném termínu odevzdá na sekretariátu katedry.

Osnova práce

1. Úvod
2. Literární rešerše
3. Vlastní práce
4. Diskuze
5. Závěr
6. Literatura

Rozsah textové části

60-70 stran

Klíčová slova

měření, software, CRC, výkon, tenzometry

Doporučené zdroje informací

Manuál měřící jednotky

WIEGERS, Karl Eugene. Požadavky na software. Vyd. 1. Brno: Computer Press, 2008, 448 s. ISBN 978-80-251-1877-1.

GADDIS, Tony. Starting out with C: from control structures through objects. 7th ed., brief version. Boston: Pearson, c2012, xxx, 997 p. ISBN 978-013-2772-891.

DOS PASSOS, Waldemar. Numerical methods, algorithms, and tools in C?. Boca Raton, FL: CRC Press, c2010, xv, 580 p. ISBN 08-493-7479-0.

VIRIUS, Miroslav. C# 2010: hotová řešení. 1. vyd. Brno: Computer Press, 2012, 424 s. K okamžitému použití (Computer Press). ISBN 978-80-251-3730-7.

VACEK, Václav. Sériová komunikace ve WIN 32: hotová řešení. 1. vyd. Praha: BEN - technická literatura, 2003, 127 s. K okamžitému použití (Computer Press). ISBN 80-730-0086-5.

VLK, František. Diagnostika motorových vozidel: hotová řešení. 1. vyd. Brno: Prof.Ing.František Vlk,DrSc., nakladatelství a vydavatelství, 2006, 444 s. K okamžitému použití (Computer Press). ISBN 80-239-7064-X.

Vedoucí práce

Peterka Bohuslav, Ing., Ph.D.

Termín zadání

listopad 2012

Termín odevzdání

duben 2014

Elektronicky schváleno dne 11.2.2013

prof. Ing. Josef Pošta, CSc.

Vedoucí katedry

Elektronicky schváleno dne 11.2.2013

prof. Ing. Vladimír Jurča, CSc.

Děkan fakulty

Prohlášení

Prohlašuji, že jsem diplomovou práci na téma: „Návrh a realizace softwaru pro univerzální měřící jednotku“ vypracoval samostatně a použil jsem jen citovaných literárních pramenů.

V Praze dne 8.4.2014

Jaromír Holec

Poděkování

Tímto bych rád poděkoval Ing. Bohuslavu Peterkovi, Ph.D. za odborné vedení a pomoc při zpracování diplomové práce a doc. Ing. Martinu Pexovi, Ph.D. za ceněnou možnost otestování modulu pro komunikaci s měřící jednotkou na katedře jakosti a spolehlivosti strojů, Technické fakulty, České zemědělské univerzity v Praze.

Abstrakt: Cílem této diplomové práce bylo navrhnout univerzální modul pro komunikaci s měřící jednotkou, která byla za tímto účelem poskytnuta k testování a samotnému vývoji softwaru pro obsluhu a zpracování naměřených dat.

Jednotlivé kapitoly uvedou čtenáře do dané problematiky, zmapují současný stav poznání a budou čtenářům nápomocny při pochopení této práce.

Práce je rozdělena na dvě části. V první části jsou popsány základní fyzikální veličiny, které jsou svázány s měřením výkonu, dále jsou rozebrány možnosti měření těchto výkonových parametrů včetně senzorů pro pořizování těchto diagnostických signálu. Druhá část této práce přejde k jiné problematice a zabývá sériovou komunikací, kontrolními součty, stručně čtenáře seznámí s měřící jednotkou a hlavně se bude zabývat přímo vývojem softwaru, včetně dokumentace algoritmů použitých při vývoji.

V závěrečné části této práce jsou shrnuty jednotlivé kapitoly, rekapituluje problematiku, která byla v předchozích kapitolách probírána a popisuje, jak bylo postupováno při řešení jednotlivých problémů.

Klíčová slova: Měření, software, cyklický redundantní součet, výkon, inkrementální snímač, tenzometry

Abstract: The aim of this thesis was design a universal module for communication with the measurement device, which for this purpose has been provided for testing and development of software itself to communicate with her.

Individual chapters bring the reader into the issue, map the current state of knowledge and will assist the reader in undertanding this work.

This work is divided into two parts. In the first section describes the basic physical quantities that are linked to performace measurement, further are also discussed possibilities of measure these performance parameters including sensors for capturing these diagnostic signals. The second part of this work goes to othre issue and engaged in seriál communicatins, checksums, briefly acquiant the reader with the measurement device and mainly will deal directly with software development, including documentation of the algorithms used in the development.

The final section of theses thesis summarizes the individual chapters, summary of issues witch are discussed in them a description of how we procedded to solve individual problems.

Key words: Measurment, software, cyclic redundancy check, power, incremental encoder, strain gage

Obsah

1 Úvod

Provozním parametrem stroje je míněna každá běžně měřitelná fyzikální veličina, kterou jsme schopni charakterizovat činnost stroje při jeho práci. Mezi tyto základní fyzikální veličiny může patřit například výkon, výkonnost, účinnost, hospodárnost provozu a podobně. Velmi mnoho provozních parametrů lze výhodně využít přímo jako diagnostické signály. Výhoda a jednoduchost je zde především v tom, že tyto diagnostické signály jsou přímo vybuzeny provozní činností stroje, která lze obvykle snadno realizovat nebo alespoň imitovat v dílenských podmínkách. Budeme-li například mluvit o výkonu vozidla, je nutné si uvědomit, že při měření určitého výkonu se zcela logicky budeme ptát, s jakou hospodárností a s jakými případnými vedlejšími důsledky bylo tohoto výkonu dosaženo [1].

Měření výkonu je možné uskutečnit mnoha zařízeními. Tato práce se bude zabývat vývojem softwaru pro univerzální měřící jednotku, jejímž hlavním účelem je provést velmi přesná měření mnoha veličin současně, což je vysoce náročné na počítačový strojový čas. Použitím této jednotky sice dojde k odlehčení nadřazenému počítači, který navíc ani není schopen tak přesného měření, nicméně jednotka sama o sobě s uživatelem nijak nekomunikuje. Protože jednotka pracuje ve dvou režimech, kde v jednom režimu čistě pouze odpovídá na povely od uživatele a dostane-li správný pokyn, tak odesílá do nadřazeného počítače data v nezpracované formě, bylo nutné navrhnout modul, který bude s těmito daty schopen pracovat, a prezentovat je obsluze v určité čitelnější podobě [1] [2].

2 Cíle práce a metodika

Hlavním cílem této práce je vytvořit univerzální komunikační modul, knihovnu, pro sběr dat a obsluhu měřící jednotky válcové zkušebny vozidel. Nejprve bylo nutné uskutečnit rozbor současného stavu problematiky, poté zvolit vhodný programovací nástroj a také řádně modul okomentovat z důvodu přenositelnosti a snazšího porozumění algoritmu jednotlivých metod pro komunikaci.

V první kapitole literární rešerše si rozebereme výkon a točivý moment. Principu jejich výpočtu a stručnému shrnutí, aby čtenář této práce byl alespoň z části uveden do dané problematiky. V kapitole následující pak budou analyzovány metody, které jsou v současné době pro měření výkonu využívané. Pro měření výkonu je vždy nutný určitý diagnostický signál, proto tato práce obsahuje kapitolu, která popisuje snímače těchto diagnostických signálu, používané pro snímání otáček, nebo točivého momentu měřící jednotkou. Pro měření otáček je použit inkrementální snímač a ke zjištění točivého momentu můžeme využít tenzometrické snímače. Základní principy obou rozebere příslušná kapitola.

Další část práce bude věnována komunikačním protokolům a cyklickým redundantním součtům. Abychom byli schopni naměřená data číst, je nutné tato data předat do nadřazeného počítače. Proto budou v kapitole následující po rozboru výkonu a variant jeho měření popsány možnosti komunikace, principy jak komunikace mezi měřící jednotkou a nadřazeným počítačem probíhá, protože naměřená data nelze v jednotce uchovávat. Kontrolní součet, který je bezpochyby nutný k ověření správnosti přijatých dat, protože se může stát, například použitím nevhodného datového kabelu, že spoj mezi měřící jednotkou a počítačem bude rušený a z dat, která budeme chtít využít ke zpracování, nebude možné získat validní informace.

Abychom byli schopni komunikovat s měřící jednotku, musíme zmapovat jaké příkazy je měřící jednotka schopna akceptovat, a zároveň jakými příkazy jednotka komunikuje s uživatelem. Proto, než vznikly tyto řádky, bylo nutné seznámit se s postupem měřícího algoritmu a principem činnosti vůbec, abychom pochopili, jak vlastně jednotka data pořizuje a v jaké konkrétní formě data odesílá do nadřazeného počítače.

Jelikož se práce zabývá vývojem softwaru a může stát, že čtenářem nemusí být nutně programátor, byla přidána kapitola, ve které čtenáře seznámíme alespoň s hlavními klíčovými vlastnostmi programovacího jazyka a vývojových diagramů, aby pochopil, proč která metoda používá daný datový typ a aby bylo porozuměno modifikátorům přístupu, které se v metodách nacházejí. Tato kapitola jemně uvede čtenáře do programování a v podkapitolách následujících jsou již popisovány samotné metody, které je modul pro komunikaci s měřící jednotkou, schopen poskytnout uživateli, ať již bude ke knihovně přistupovat pomocí nějakého vývojového prostředí, nebo například softwaru pro simulaci měřící techniky, ve

kterém byl průběžně modul testován. Úvod této kapitoly je věnován koncepcí pojmenování a značení jednotlivých elementů, což by mělo usnadnit čitelnost a orientaci v této kapitole.

Kapitoly, které popisují jednotlivé třídy, jejich atributy a metody jsou sepsány a rozděleny tak, že v úvodu každé podkapitoly je čtenáři zobrazen návrh se stručným popisem a účele dané třídy a poté, je pomocí textových komentářů nebo vývojových diagramů popsán princip samotných metod. Jelikož vývojový diagram slouží ke grafickému znázornění jednotlivých kroků algoritmů nebo obecně nějakého procesu, není v nich zahrnut datový tok zpracovávaných údajů, protože v tomto případě není zapotřebí.

3 Měření výkonových parametrů

Výkon spalovací motoru je nesporně významným diagnostickým signálem, využitelným především k souhrnné diagnóze pístní skupiny, rozvodového ústrojí, palivové a též zapalovací soustavy. Musíme si ovšem uvědomit, že výkon sám ke stanovení diagnózy zpravidla nestačí [1].

Výkon motoru, nebo jak bychom správně měli nazývat „vnější otáčková charakteristika“, které bychom mohli rozumět jako průběh výkonu, točivého momentu a měrné spotřeby paliva při plně otevřené škrticí klapce [3], ve zkušebnictví motorových vozidel nejčastěji zjišťuje mechanický výkon rotačním pohybem na válcové stanici. Jelikož tento parametr nelze přímo měřit, musíme jej vypočítat [4].

Vstupním signálem u spalovacího motoru může být bezprostředně měřen výkon **P**. Například torzním dynamometrem lze získat elektrický napěťový signál, který je úměrný točivému momentu **M**, tachodynamem pak získáme napěťový signál, který je úměrný úhlové rychlosti **ω**. Teď již ale k rozboru jednotlivých diagnostických veličin [1].

3.1 Výkon motoru

Výkonem motoru vyjadřujeme práci, někdy nazývanou jako energii, kterou vykonává motor za jednotku času. Výkon motoru patří mezi základní diagnostické veličiny, kterými jsme schopni číselně vyjádřit hodnotu při daných otáčkách motoru, nebo také grafickou závislost zkoumaného výkonu v celém rozsahu otáček. Vypočítá se dle pomocí vztahu 1 a počítáme-li s výkonem jako s prací, kterou vykoná motor za 1 sekundu, můžeme jej vyjádřit v základních jednotkách – watech [5] [4].

$$P = M \cdot \omega \, [W] \tag{1}$$

kde	P	[W]	-	výkon motoru
	ω	$[\mathrm{rad \cdot s^{-1}}]$	-	úhlová rychlost
	M	[N·m]	-	točivý moment

3.2 Točivý moment motoru

Točivý moment **M** je roven součinu úhlového zrychlení **ε** a momentu setrvačnosti **I** veškerých rotujících hmot uvnitř motoru [1]. Vypočítá se podle

.

$$M = I \cdot \varepsilon \, [N \cdot m] \tag{2}$$

kde	M	[N·m]	-	točivý moment motoru
	I	$[\mathrm{kg \cdot m^2}]$	-	moment setrvačnosti

	ε	$[\text{rad}\cdot\text{s}^{-2}]$	-	úhlové zrychlení

Z výše uvedených poznatků pak vyplyne rovnice 3 vztahu pro výpočet výkonu **P**:

$$P = I \cdot \varepsilon \cdot \omega \; [W] \tag{3}$$

kde	P	[W]	-	výkon motoru
	I	$[\text{kg}\cdot\text{m}^2]$	-	moment setrvačnosti
	ε	$[\text{rad}\cdot\text{s}^{-2}]$	-	úhlové zrychlení
	ω	$[\text{rad}\cdot\text{s}^{-1}]$	-	úhlová rychlost

Pro měření točivého momentu můžeme použít deformačních členů. Toto měření funguje na principu, že měřící člen (hřídel s kruhovým průřezem) je krutem namáhán momentem síly, který převádí tento způsob namáhání na deformaci a měří pomocí tenzometrických snímačů nebo snímači výchylky. Jiné konstrukce využívají například změn magnetických vlastností deformačního členu [4].

3.3 Moment setrvačnosti

Moment setrvačnosti je fyzikální veličina, která vyjadřuje míru setrvačnosti tělesa při otáčivém pohybu. Její velikost závisí na rozložení hmoty v tělese vzhledem k ose otáčení. Body tělesa s větší hmotností a umístěné dál od osy otáčení mají větší moment setrvačnosti. Symbolem veličiny je **J**, někdy také **I** a v jednotkách SI se vyjadřuje jako $kg \cdot m^2$.

3.3.1 Získání momentu setrvačnosti

Pro měření dynamických provozních parametrů vozidel a jejich spalovacích motorů je nutné ve všech případech do výpočtu zahrnout správnou hodnotu momentu setrvačnosti motoru, nebo celého vozidla, protože při měření na samotném motoru má točivý moment zásadní vliv. Možností, jak získat hodnotu momentu setrvačnosti je několik [6]:

- *Získání údaje od výrobce* – U moderních motorů to již zpravidla problematické není z důvodu, že koncernové diagnostické přístroje přímo nabízejí možnost měření výkonu akcelerační metodou. Údaj o momentu setrvačnosti motoru by již měl být k dispozici v diagnostických přístrojích. Problém nastane tehdy, pokud není k dispozici koncernová diagnostika, podrobné informace o vozidle, nebo v případě staršího vozidla [6].
- *Výpočtem* – Moment setrvačnosti lze vypočítat z rozměrů a dalších údajů o jeho jednotlivých součástech. Jedná se o variantu velmi náročnou na přesnou znalost konstrukce a rozměrů všech součástí [6].

- *Přívažkem* - Při použití přívažku o známém momentu setrvačnosti lze provést dvě měření. Jedno s přívažkem a druhé bez něj. Porovnáním obou výsledků měření můžeme dopočítat moment setrvačnosti motoru [6].

$$I = \frac{F_a}{d_2 - d_1}[kg \cdot m^2] \tag{4}$$

kde	I	$[kg \cdot m^2]$	-	moment setrvačnosti
	F_A	$[kg \cdot m^2]$	-	přidaná externí zátěž
	d_1, d_2	$[rad \cdot s^{-2}]$	-	zpomaleni

- *Nový motor* - Pokud máme k dispozici nový motor, u kterého výrobce garantuje výkonové parametry, můžeme provést akcelerační měření. Zpětně pak lze stanovit hodnotu momentu setrvačnosti motoru tak, aby výkonové parametry odpovídaly tabulkovým hodnotám [6].

$$I = \frac{M_D}{\varepsilon_M}[kg \cdot m^2] \tag{5}$$

kde	I	$[kg \cdot m^2]$	-	moment setrvačnosti
	M_D	$[kg \cdot m^2]$	-	efektivní točivý moment měřený dynamometrem
	ε_M	$[rad \cdot s^{-2}]$	-	úhlové zrychlení

- *Dynamometrem* - Obdobně jako nový motor lze podobným způsobem využít dynamometr. Výsledným hodnotám akceleračního měření přiřadíme určený moment setrvačnosti tak, aby souhlasily s hodnotami naměřenými na dynamometru [6].
- *Průměrem* – U této metody předpokládáme měření rozsáhlého počtu vozidel se stejným motorem a postupně zpřesňujeme hodnoty momentu setrvačnosti [6].

3.4 Měření výkonových parametrů spalovacího motoru

Měřením výkonových parametrů motorového vozidla, tj. točivého momentu [**N·m**] a výkonu [**W**] v závislosti na frekvenci otáček motoru se v praxi používá několik metod s různou mírou přesnosti na náročnosti. Nejvýznamnější metody, které se dnes používají k měření výkonových parametrů, můžeme rozdělit podle tabulky 1 uvedené pod odstavcem [7].

Pro měření výkonu se používají výkonové brzdy. Označení brzdy je v tomto případě odvozeno z toho, že proti točivému momentu motoru působí moment brzdný o známé velikosti nebo takový moment, který můžeme měřit [8].

Tab. 1 - Přehled metod měření výkonových parametrů motoru [7]

<table>
<tr><th>Způsob zatížení</th><th>Umístění motoru</th><th>Výkon měřený na</th><th>Princip měřícího zařízení</th></tr>
<tr><td rowspan="4">STATICKÉ
(Stacionární) předvolené otáčky motoru jsou udržovány zatěžovacím momentem brzdy (u automobilových motorů), zatěžovací moment se volí nezávisle na otáčkách (u motorů s vlastní regulací)</td><td>Na zkušebním stanovišti</td><td>Klikové hřídeli nebo jiném srovnatelném místě</td><td rowspan="4">Absorpčními dynamometry:
- Elektromagnetické vířivé brzdy
- Hydraulické brzdy
- Mechanické frikční brzdy
- Vzduchové brzdy ()
- Tandemové brzdy
Univerzálními dynamometry:
- Elektro dynamické motorgenerátory na stejnosměrný nebo střídavý proud
Torzní dynamometry:
(nebrzdí)</td></tr>
<tr><td rowspan="3">Ve vozidle (v místě instalace)</td><td>Obvodu hnacích kol (válcové zkušebny)</td></tr>
<tr><td>Klikové hřídeli nebo jiném srovnatelném místě</td></tr>
<tr><td>Vývodové hřídeli (traktory a užitková vozidla</td></tr>
<tr><td rowspan="3">DYNAMICKÉ
urychlování setrvačných hmot zvoleným točivým momentem</td><td rowspan="3">Ve vozidle</td><td>Obvodu hnacích kol (válcové zkušebny)</td><td>Měření úhlového zrychlení setrvačných hmot (přídavné setrvačníky na válcích)</td></tr>
<tr><td>Klikové hřídeli nebo jiném srovnatelném místě</td><td>Měření úhlového zrychlení klikové hřídele samotného motoru (volná akcelerace) nebo s přídavnými setrvačnými hmotami při jízdě na určitý převodový stupeň</td></tr>
<tr><td>Přepočet výkonu na klikový hřídel</td><td>Měření přímočarého zrychlení celého vozidla</td></tr>
</table>

3.5 Měření statickým zatěžovacím momentem

Obvykle se statickým (stabilním) zatížením spalovacího motoru rozumí takové zatížení, které umožňuje nastavení předvolených otáček, které jsou v průběhu snímání jednotlivých vstupů a výstupů z motoru konstantní. K udržování příslušného zatížení slouží celá řada dynamometrů [8].

3.6 Měření dynamickým zatěžovacím momentem

Motor je krátkodobě zatížen odporem setrvačníků během jejich roztáčení. Výkon je v tomto případě stanoven výpočtem (pomocí momentu setrvačnosti, úhlové rychlosti a úhlového zrychlení). Zásadní roli tedy mají momenty setrvačnosti všech roztáčejících se částí. Pokud nejsou tyto hodnoty přesně známy, má hodnota dynamicky zjišťovaného momentu

pouze informativní charakter. Během této zkoušky nedojde k stabilizaci vnitřních teplot motoru (povrchové teploty spalovacího prostoru, plyny v sacím a výfukovém potrubí), kde nejdůležitější je teplota spalin ve výfukovém potrubí. Pokud není výfukové potrubí dostatečně prohřáté, dojde k posunutí maximálního výkonu do nižších otáček a celá výkonová křivka může být deformována [4].

3.7 Akcelerační měření výkonu

Výkon a točivý moment jsou úměrné uhlovému zrychlení klikové hřídele nezatíženého motoru, rozbíhající se ho při plném sešlápnutí akceleračního pedálu z volnoběžných otáček. Zároveň tak jsou rovny úhlovému zpomalení nepracujícího motoru [1].

Úhlové zrychlení lze v zásadě měřit buď u celého motoru, nebo u motoru, který pracuje na jednotlivé válce. Úhlové zpomalení však představuje při běžném měření vždy hodnotu, která je platná pro celý motor. Protože úhlové zrychlení a zpomalení lze měřit bez nákladných investičních zařízení poměrně jednoduchými, snadno přenosnými elektronickými přístroji, nabývá tato metoda stále většího praktického významu [1].

Metodika měření je založena na principu, kde se při režimu urychlování otáček vně nezatíženého motoru měří zároveň časové úseky mezi impulsy, které vzniknou vždy pootočením klikové hřídele motoru o konstantní úhel φ. Úhlové zrychlení ε klikové hřídele motoru v každém jednotlivém časovém úseku t_k, se vyhodnotí podle vztahu (6) [9].

$$\varepsilon_k = \omega_k \cdot \left(\frac{1}{t_{k+1} - t_k} - \frac{1}{t_k - t_{k-1}} \right) \left[rad \cdot s^{-2} \right] \tag{6}$$

kde	t_k	[s]	-	bod na časové ose v okamžiku náběžné hrany ze snímače otáček klikové hřídele
	ε_k	$\left[rad \cdot s^{-2} \right]$	-	úhlové zrychlení
	ω_k	$\left[rad \cdot s^{-1} \right]$	-	úhlová rychlost

Úhlovou rychlost v čase vzniku náběžné hrany ze snímače otáček klikové hřídele $\boldsymbol{\omega_k}$ vypočteme dle vztahu:

$$\omega_k = \frac{\pi \cdot n_k}{30} \left[rad \cdot s^{-1} \right] \tag{7}$$

kde	ω_k	$\left[rad \cdot s^{-1} \right]$	-	úhlová rychlost
	n_k	$\left[min^{-1} \right]$	-	okamžitá frekvence otáček

Respektive při konstantním pootočení klikové hřídele o jednu otáčku $\varphi=2\pi$, respektive $\varphi=4\pi$ u čtyřdobého motoru, přesněji podle vztahu [9]:

$$\omega_k = \frac{2\pi(4\pi)}{t_{k+1} - t_k}\ [rad \cdot s^{-1}] \tag{8}$$

kde	ω_k	$[\mathrm{rad} \cdot \mathrm{s}^{-1}]$	-	úhlová rychlost v k-tém časovém úseku měření, za jednu otáčku klikové hřídele
	t_k	[s]	-	bod na časové ose v okamžiku náběžné hrany ze snímače otáček klikové hřídele

Jádrem přesnosti akcelerační metody měření výkonových parametrů jsou vztahy vyjadřující úhlové zrychlení klikového hřídele primárně měřenými veličinami času. V našem případě je jádrem přesnosti výše uvedený vztah (6), vyjadřující úhlové zrychlení ω_k primárně měřenými veličinami času $\boldsymbol{t_k}$. Přesnost vztahu (6) je vždy třeba ověřit na modelu a vyvodit závěry o potřebě přesnosti měření časových úseků $\boldsymbol{t_k}$ odpovídajícím vzniku náběžné hrany signálu ze snímače otáček klikového hřídele, včetně požadavku na daný měřící systém [9].

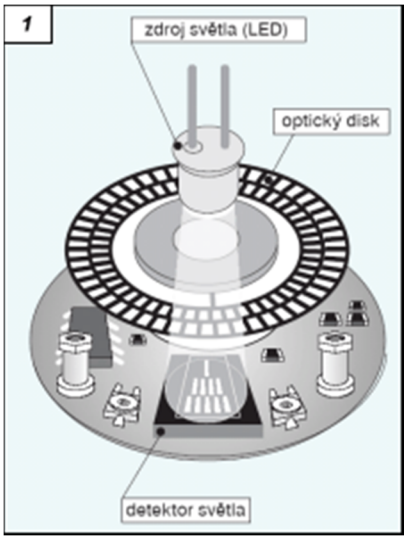
Výhodou vyhodnocování indikovaných parametrů spočívá zejména v tom, že jsou méně závislé na teplotě měřeného motoru, než je tomu u hodnot efektivních. Efektivní i ztrátové výkonové parametry se mění v celém rozsahu pracovních teplot motoru. Při zanedbání výrazně zhoršené kvality spalování, vlivem kondenzace paliva na studených stěnách válce, by byl součet jejich absolutních hodnot v celém průběhu teplot téměř konstantní. Provedené experimenty prokázaly, že téměř konstantní jsou indikované hodnoty teprve nad teplotou cca 50 °C (chladícího media). Význam pro diagnostickou praxi tedy spočívá v tom, že při použití indikovaných hodnot nemusíme měřit výkonové parametry až při dosažení předepsané provozní teploty, což významným způsobem sníží prostoje a spotřebu paliva při vlastním měření. Praktické využití se osvědčilo zejména při relativním měření výkonových parametrů za účelem zjištění účinku opravárenského úkonu nebo tzv. tuningové úpravy motoru, případně při jeho seřizování a ladění [9].

4 Snímače diagnostických signálů

4.1 Inkrementální rotační snímače

Při analýze vibrodiagnostického signálu u rotační soustavy jsou otáčky hřídelů jednou ze základních veličin. Pro tyto účely lze použít bezdotykových senzorů, které fungují na základě indukčnosti nebo jsou optoelektronické [10].

Principem optoelektronických snímačů je clonění světelného toku mezi zdrojem světla, kterým může být například světlo emitující dioda a fotocitlivým prvkem. Fotocitlivému prvku můžeme také říkat pravítko nebo jinak znám jako kotouč. Ten je rozdělen na úseky světlo propustné a nepropustné. Pro zjištění potřebných údajů pro výpočet informací o rychlosti otáčení stačí zjistit počet impulzů za určitý časový úsek. Pro zjišťování směru otáčení je pak nutno použít snímač, který obsahuje rotující kotouč se dvěma řadami otvorů, které jsou vůči sobě posunuty o poloviny šířky otvoru. Pro zjištění úhlu natočení pak má rotující kotouč ještě jeden otvor, který je určen pro generování nulového impulzu. Klasické uspořádání inkrementálního fotoelektrického snímače určeného je uvedeno na obrázku níže [12] [13].



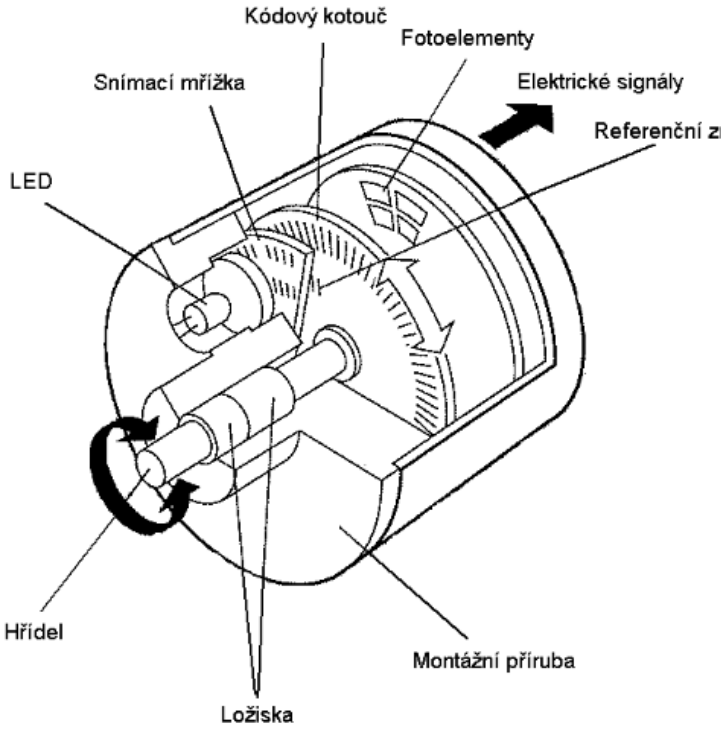
Obr. 1 - Princip inkrementálního snímače [11]

Rotační snímač polohy je elektromechanický převodník. Jeho výstup je možné používat k řízení rotačních nebo lineárních pohybů (jsou-li snímače opatřeny pastorkem a připojeny k ozubenému hřebenu nebo upevněny na pohybovém šroubu). Mezi hlavní aplikace rotačních snímačů patří obráběcí a tvářecí stroje, roboty a všeobecně elektropohony se zpětnou vazbou [11].

$$\omega = \frac{2\pi \cdot n}{n_{celk} \cdot t} \; [rad \cdot s^{-1}] \tag{9}$$

kde	ω	[rad·s^{-1}]	-	úhlová rychlost
	n	[-]	-	počet inpulsů
	n_{celk}	[-]	-	celkový počet dílků inkrementálního snímače
	t	[s]	-	doba pořízení vrozku

Jedním z nedostatků inkrementálních senzorů je skutečnost, že změna obsahu čítače případnými rušivými impulsy je opravitelná až po dosažení referenční značky. Proto se v kritických situacích užívají náročnější senzory s prostorovým kódem [12].



Obr. 2 - Inkrementální rotační snímač [13]

4.2 Senzory krouticího momentu

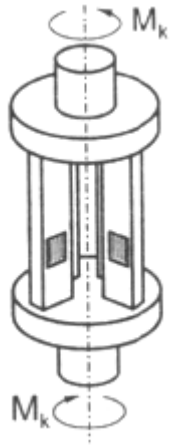
Budeme-li v této práci mluvit o senzorech točivého momentu. Můžeme v naší aplikaci mluvit o senzorech s odporovými tenzometry. Tenzometry určují buď přímo deformaci hřídele mezi motorem a zátěží, nebo je deformaci podroben vložený torzní pružný člen, speciálně tvarovaný pro tyto účely. Z rozboru mechanického napětí v jednoduchém případě hřídele nebo pružného členu ve tvaru plného i dutého válce vyplívá, že hlavní napětí jsou orientována pod úhly 45° k ose. Jde o napětí typu tah-tlak s nulovou smykovou složkou a s maximální hodnotou na povrchu válce [12]. Deformace ve směru hlavních napětí je dána rovnicí:

$$\varepsilon = \frac{r}{2 \cdot G \cdot J} \cdot M \tag{10}$$

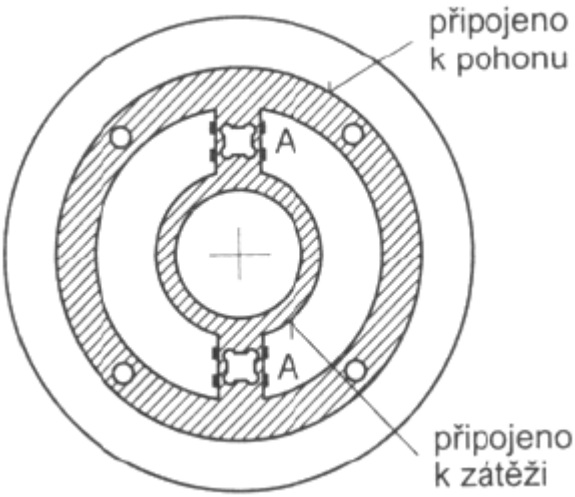
kde	G	[Pa]	-	modul pružnosti ve smyku
	J	$[kg \cdot s^2]$	-	moment setrvačnosti
	M	[N·m]	-	točivý moment motoru
	ε	[-]	-	deformace ve směru hlavních napětí

Největší citlivost a kompenzace vlivu teplotní diletace a ohybových deformací se dosáhne nalepením čtyř aktivních tenzometrů pod úhlem 45°, přičemž dva z nich jsou umístěny na opačných stranách pružného členu [12].

Návrh pružného členu je kompromisem mezi požadavkem citlivosti (G·J malé) a dynamických vlastností, vyžadujících velkou tuhost. Z tohoto pohledu je vhodný dutý válec, další užívané tvary jsou na Obr. 3 a Obr. 4. Výhodou křížového členu na Obr. 3 je zejména snadné lepení tenzometrů na ploché elementy. Odolností vůči parazitním deformacím vyniká člen na Obr. 4, v němž je střední prstenec spojen se zátěží a vnějším motorem. Oba prstence jsou spojeny pružnými členy, typu paralelogram s možností vytvořit z použitých tenzometrů dva plné můstky [12].



Obr. 3 - Křížový pružný člen [12]



Obr. 4 - Prstencový pružný člen [12]

5 Sériová linka RS-232

Komunikace definována standardem RS-232 se řadí mezi metody asynchronní sériové komunikace. Slovo sériové znamená, že informace se posílají bit po bitu najednou. Význam slova asynchronní říká, že pro informace, které posíláme přes rozhraní, nejsou předem definovány časové úseky. Přenos dat jako takový může začít v kterémkoliv okamžiku, a je tedy úkolem přijímací strany, detekovat začátek a konec zprávy. Tento způsob komunikace má zajisté své výhody, které spočívají v absenci hodinového signálu, a proto by komunikační kanál měl být levnější na provoz, ale zajisté má i své nevýhody, mezi které patří například to, že přijímací strana může začít vysílat v kterémkoliv, i nesprávném okamžiku, a tím dochází k desynchronizaci, mezi stranou přijímací a vysílající a tím i ke ztrátě dat. Mezi další nevýhody patří to, že k samotným datům musíme přidat bity, které budou uvozovat a ukončovat přijímaný datový rámec. [15]

Samotný obvod pro sériovou komunikaci zpravidla tvoří obvod UART 8250, pokud se jedná o starší typy PC, nebo obvod 82550 u typů novějších. Jádro celé komunikace může být obsaženo i v jiných čipech, které jsou použity pro řešení sérových i paralelních portů najednou, ale hovoříme-li o sériovém rozhraní COM, musíme jasně specifikovat vlastnosti obvodu, který bude sériovou komunikaci realizovat. [16]

Datové bit při sériové komunikaci mají předem stanovenou frekvenci a přenosovou rychlost. S tím souvisí, že obě strany, strana vysílače i přijímače, musí být naprogramovány tak, aby měli tyto parametry přenosu stejné. Základní parametry, pro nastavení komunikace sériové linky, patří tyto: název (označení) portu, rychlost přenosu, počet datových bitů, parita, počet stopbitů, možnost hardwarového řízení toku. Možnosti nastavení těchto parametrů jsou uvedeny v tabulce 2 níže. [16]

Tab. 2 - Parametry pro nastavení sériové komunikace. [13]

Parametry konfigurace RS-232	
Rychlost [bps]	110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 11520, 128000, 256000
Počet datových bytů	5, 6, 7, 8
Parita	Žádná, lichá, sudá
Počet stopbitů	1, 1.5, 2

5.1 Komunikační protokol SLIP

Rodina protokolů TCP/IP je funkční na spousty rozdílných nosných médiích. Jedná se o tato: IEEE 802.3 (běžně známý jako ethernet), 802.5 (token ring), linky X.25, satelitní spoje a sériové linky. Standardní zapouzdření je definováno pro mnohé z těchto typů přenosových medií až na sériové linky. SLIP, neboli Serial Line Internet Protocol, se v současnosti stal de facto standardem a je nejčastěji používán pro připojení typu bod-bod, kde na sériových spojení běží protokol TCP/IP. [17]

5.1.1 Princip funkce

SLIP protokol definuje dva speciální znaky. Jedná se o znak END a ESC. Zastoupení těchto znaků číselnou hodnotou vyjadřuje tabulka 3 níže.

Tab. 3 - Číselné zastoupení znaků SLIP [14]

Příkaz	Dekadicky	Hexadecimálně
END	192	C0
ESC	219	DB

Pokud chce hostitel odeslat datový paket, tak jednoduše začne vysílat. Jestliže v přijatém datovém paketu přijde bajt, který je totožný s koncovým charakterem, protokol SLIP jednoduše provede přidání zástupného znaku pro END sekvenci a to konkrétně znakem $(DC)_{HEX}$. Na přijímací straně toto musíme detekovat a příslušně zpracovat. Je-li datový bajt totožný s ESC charakterem, pak protokol SLIP provede nahrazení dvěma bajty $(DB)_{HEX}$ a $(DD)_{HEX}$. V našem případě je komunikační protokol SLIP implementován ve třídě *AccessSerialPort* a to konkrétně metodou **AplySlip()**. Bližší informace, jak tato metoda funguje, jsou uvedeny v příslušné kapitole. [17]

6 Cyklický redundantní součet

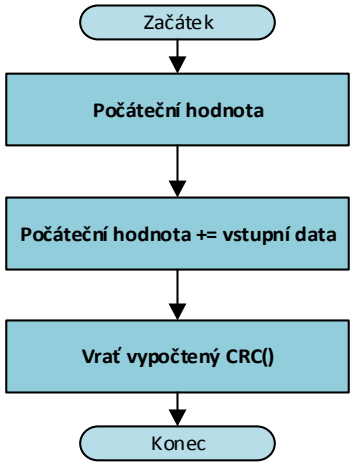
Kdykoliv začala být data ukládána nebo propojována, mohlo dojít k jejich poškození. Od začátku počítačových věd tedy začali lidé přemýšlet o možnostech, jak tyto problémy vyřešit. Pro data, která byla přenášena sériově, bylo prvotním řešením přidání paritního bytu, na každý odeslaný bajt (dále jen B). Tento jednoduchý detekční mechanizmus funguje pouze tehdy, když je v poslaném B lichý počet změněných bitů, ale ani počet záporných bitů v jednom bajtu nebude možno detekovat při kontrole parity. K překonání tohoto problému lidé hledali a vynalezli tak matematický mechanizmus, kterým detekovali více negativních bitů. Cesta, jak toto překonat, tedy byla vynalezení CRC (cyklický redundantní součet). Nyní využíváme výpočet CRC ve všech typech komunikací, s přenosem elektrických signály. Skrze síť jsou poslány všechny pakety určené k přenosu a jsou kontrolovány pomocí kontrolních součtů. Stejným způsobem je také kontrolován každý blok dat na pevném disku. V dnešní době se moderní počítačový věk bez těchto výpočtů kontrolních součtů neobejde, takže bychom se tedy v této kapitole podívali na to, proč jsou dnes kontrolní součty tak široce používány a jaké jsou známé způsoby (druhy) výpočtů kontrolních součtů. Nejsnazším objasněním proč jsou tyto kontrolní součty používány, je to, že jsou velmi schopné detekovat mnoho typů chyb a jejich výpočet je extrémně rychlý, zejména tehdy, jsou-li pro jejich výpočet používány dedikované hardwarové čipy. [18]

Někdo by si mohl myslet, že užitím kontrolního součtu lze nahradit vlastní výpočet CRC. Bylo by to jistě jednodušší, avšak z vypočteného kontrolního součtu nejsme schopni objevit všechny chyby v přijatých datech. Vezměme si příklad na textovém řetězci „*Diplomka*“**,** který převedeme na ASCII hodnoty. Po převedení dostaneme v ASCII hodnoty [68 105 112 108 111 109 107 97]. Pro tento řetězec by pak jeden bajt kontrolního součtu, který bychom vypočítali sečtením hodnot všech znaků, a udržením zbytku po celočíselném dělení hodnotou 256 byl roven hodnotě 49. [18]

Ve výše uvedeném příkladu jsme použili jeden B dlouhý kontrolní součet, který však nabízí pouze 256 možných kombinací. Použijeme-li kontrolní součet o délce 2B, budeme schopni vypočíst až 256^2=65 536 možných různých hodnot kontrolních součtů. Při použití 4B kontrolního součtu pak analogicky 256^4=4 294 967 296 možných hodnot pro kontrolní součet. Můžeme tedy předpokládat, že při 4 B kontrolního součtu je pravděpodobnost nezjištění chyby 1 ku 4 miliardám. Teoreticky se to zdá dobré, ale skutečnost je trochu jiná. Praktické zkušenosti říkají, že v průběhu komunikace se bity nemění náhodně. Často selžou v dávkách nebo z důvodu elektrických špiček. Představme si tedy, že namísto řetězce „*Diplomka*“ bychom odeslali řetězec „*Cjplomka*“ [67 106 112 108 111 109 107 97], pro tento řetězec je kontrolní součet totožný s předchozím, avšak ve výsledku je přijatý řetězec špatně, tudíž ani kontrolní součet nepřináší o moc větší ochranu proti paritnímu bytu, nezávisle na jeho délce [18].

Myšlenka pro výpočet kontrolní hodnoty je jednoduchá. Máme funkci crc, která k 1B vstupních dat přidá počáteční hodnotu kontrolního součtu a jako návratovou hodnotu vrací vypočteného kontrolního součtu. Toto opakuje pro každý bajt vstupního řetězce a pro jako hodnotu počáteční hodnoty vezme číslo 0. Takto popsaná funkce je uvedena na vývojovém diagramu níže na Obr. 5 [18].

Obr. 5 - Vývojový diagram funkce crc

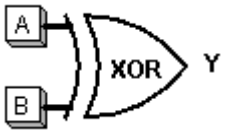


Samotný výpočet pak spočívá v aplikaci logické funkce XOR na generální polynom a každý bit hodnoty, ze které chceme CRC vypočítat.

6.1 Logická funkce XOR

Funkce XOR, jinak známa také jako exkluzivní disjunkce, nabývá pravdy, logické hodnoty 1, právě tehdy, když je hodnota obou vstupů různá. Názorná ukázka této logické funkce je uvedena v pravdivostní tabulce na Obr. 6.

Obr. 6 - Logická funkce XOR

Logická funkce	*A*	*B*	*Y*
	*0*	*0*	*0*
	0	*1*	*1*
	1	*0*	*1*
	1	*1*	*0*

6.2 Princip výpočtu CRC

Pokud mluvíme o cyklických redundantních součtech, nebo jak je známo CRC, vycházejícího z anglického Cyclic Redundancy Check, tak se jedná o obdobu hašovací funkce, která slouží k detekci chyb. Ačkoliv umí chyby přenosu detekovat, neopravují je,

pouze zobecňují paritní bit. CRC je odesílán spolu s daty a po přijetí znovu nezávisle vypočítán. Pokud se vypočtený CRC jakkoliv liší od přijatého, je zřejmé, že během přenosu došlo k chybě [19].

Princip výpočtu CRC je pohled na data jako na jedno velké binární číslo. Toto číslo se dělí konkrétní hodnotou a zbytek výpočtu se pak nazývá jako cylindrický redundantní součet [19].

Přenášená data lze interpretovat, jako generální polynom, někdy jen polynom, s binárními koeficienty. Význam a reprezentaci generálního polynomu nejlépe vysvětlí následující příklad, kde binární hodnotu $(110011001)_2$ můžeme zapsat ve formátu generálního polynomu a to jako $x^8+x^7+x^4+x^3+1$ cyklický redundantní součet se pak počítá jako zbytek po dělení dat generálním polynomem. Dělení při výpočtu CRC zpočátku stálo značnou část výpočetního výkonu, ale muže být velmi urychleno, použijeme-li metodu, který je podobná té, která se učí již na základní škole [19] [18].

Jako příklad si uvedeme výpočet zbytku pro znak ´. ´, který je binární notací zobrazen jako 101110 a budeme jej dělit hodnotou $15=(1111)_2$. Názorná ukázka je v Obr. 7. Berme na vědomí, že číslo 15 je liché, protože pro tento výpočet je lichá hodnota dělitele nezbytná jak uvidíme níže [18]. Jak je z Obr. 7 vidět, tak výsledek po dělení čísla 46 hodnotou 15 je roven 3 a zbytku 1.

Výpočet cyklického kontrolního součtu se odvíjí od generálního polynomu. Uvedeme si příklad v kapitole 7.4 Příklad výpočtu CRC-CCITT s počáteční hodnotou 0xFFFF je názorně ukázáno, jak takový výpočet probíhá. V kostce je zde určitá počáteční hodnota. Výpočet kontrolního součtu probíhá pro každý byte jednotlivě, proto mluvíme o cyklickém redundantním součtu. Nejprve musíme provést logickou operaci XOR s počáteční hodnotu a

Obr. 7 - Dělení 46/15

46:15=3 zbytek 1

Binárně pak:

```
  1  0  1  1  1  0  =  1  1
- 1  1  1  1  |  |
  1  1  1  1  1  |
- 1  0  0  0  0  0
   - 1  1  1  1  1
                  1
```

jednomu bytu čísla, ze kterého chceme CRC vypočítat. Pro nově vzniklý prvek poté provedeme znovu logickou funkci XOR nicméně pro každý bit, ale teď již s hodnotou generálního polynomu. Po každé operaci XOR musíme provézt bitový posuv vlevo o jeden bit. Pokud je hodnota posledního bitu aktuálně xorovaného řetězce rovna jedné, přeskakujeme

výpočet. Na stejném principu je postavena metoda **GetCrcCcitt()** a sní související metoda **CrcCcitt()**, které jsou členem třídy *Crc* a jejich vývojové diagramy jsou na Obr. 15 a Obr. 16.

6.3 Polynomiální funkce kontrolních součtů

V této kapitole shrneme nejčastější používané generální polynomy, které jsou v dnešní době používány. Tabulka 4 pod odstavcem ukazuje na pár nejpoužívanějších.

Tab. 4 - Hodnoty generálních polynomů [16]

Označení CRC	Hodnota GP	Funkční přepis GP	Poznámka
CRC-1	0x3	$x + 1$	Znám jako paritní bit
CRC-5-USB	0x25	x^5+x^2+1	Využívaný v USB paketech
CRC-16	0x8005	$x^{16}+x^{12}+x^2+1$	USB, Modbus
CRC-CCITT	0x1021	$x^{16}+x^{12}+x^5+1$	Aplikován v měřící jednotce
CRC-DNP	0x3D65	$x^{16}+x^{13}+x^{12}+x^{11}+x^{10}+$ $x^8+x^6+x^5+x^2+1$	M-Bus
CRC-32	0x04C11DB7	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+$ $x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$	Ethernet

6.4 Příklad výpočtu CRC-CCITT s počáteční hodnotou 0xFFFF

Níže uvedený Obr. 8 předvádí názornou ukázku výpočtu cyklického redundantního součtu pro řetězec $(AAFF)_{HEX}$. Výsledek praktické ukázky reprezentuje CRC o hodnotě $(F14A)_{HEX}$.

Výpočet CRC-CCITT pro hodnotu 0xAAFF

GP:	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
Řetězec:	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	
Počáteční hodnota:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	1	0	1	0	1	0	1	0									
0xAA:	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	0	0
		1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1
			0	1	0	0	0	1	1	1	1	1	0	1	1	1	0	1	0	0
				1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1
				0	0	0	0	0	1	1	1	1	0	1	0	1	0	1	0	0	0	0
								0xFF:	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
									0	0	0	0	1	0	1	0	0	1	0	1	0	0	0	0
													1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0
													1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1
													0	0	1	0	1	1	0	1	0	0	0	1	0	0	0	0	1
															1	0	1	1	0	1	0	0	0	1	0	0	0	0	1	0	0
															1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1
															0	0	1	1	1	1	0	0	0	1	0	1	0	0	1	0	1	0
																CRC:		**F**				**1**				**4**				**A**		

Obr. 8 - Příklad výpočtu CRC-CCITT

7 Měřící jednotka pro válcovou zkušebnu vozidel

Tato kapitola stručně popisuje nejdůležitější části, manuálu měřící jednotky. Jedná se pouze o výpis nejpodstatnějších segmentů z vybraných kapitol, protože podrobné, do hloubky vysvětlené, specifika měřícího zařízení jsou uvedena v návodu, se kterým byl dodána.

Obr. 9 - Měřící jednotka [2]

Účel, za kterým byla jednotka zkonstruována bylo vytvoření jednoúčelového měřícího zařízení, které by mělo snímat primární data z válcové zkušebny vozidel a posílat je pomocí sériové linky do nadřazeného počítač. Tím, že se o prvotní sběr dat stará měřící zařízení, nikoliv řídící počítač, dojde k enormnímu ušetření výpočetního výkonu nadřazeného počítače, kde jsou data zpracována. Jelikož měřící jednotka velmi přesně provádí měření mnoha veličin současně, tak tento proces je natolik náročná, že jej ani nadřazený počítač nebyl schopen zvládat. [2]

7.1 Příkazy a odpovědi

V této kapitole jsou sepsány příkazy a odpovědi, které jednotce můžeme zaslat, nebo naopak kterými jednotka odpovídá a interakuje s nadřazeným počítače. Kapitola popisuje pouze příkazy, kterými odpovídá měřící jednotka, proto bychom je neměli zaměňovat

s odpověďmi, které vrací knihovna, jakožto komunikační modul pro obsluhu jednotky válcové zkušebny vozidel.

7.1.1 Tabulka příkazů

Tab. 5 - Příkazy akceptovatelné měřící jednotkou [2]

Název příkazu HEX hodnota	Parametr	Popis
TestCmd 0xaa	1 bajt libovolné hodnoty, který jednotka vrací v odpovědi na tento příkaz	Testovací příkaz pro ověření spojení s měřicí jednotkou. Jednotka pouze odpoví, ale na její funkci to nemá žádný vliv.
StartMeasure 0x35	Žádný	Aktivuje měření, jednotka začne měřit a asynchronně posílat naměřená data.
StopMeasure 0x39	Žádný	Deaktivuje měření, jednotka přestane měřit a asynchronně posílat naměřená data.
SetInterval 0x30	2 bajtová hodnota v rozsahu 2 až 65535	Nastaví požadovaný interval měření. 1 dílek je 327,68µs (= 214/50MHz)
GetTorque 0x45	Žádný	Žádost o změření momentů všech kanálů. Příkaz funguje pouze ve stavu deaktivovaného měření a slouží pro kalibraci snímačů momentu.
GetTimeBase 0x5a	Žádný	Žádost o vzorek z časové osy. Příkaz funguje kdykoliv a lze jej využít pro diagnostické účely nebo pro synchronizaci jiných měření.
GetStatus 0x5c	Žádný	Žádost o zaslání stavového registru. Stavový registr je bajt, který obsahuje binární diagnostické informace.

Jak můžeme vidět, tak většina příkazu je složena z hexadecimální hodnoty, volitelně parametru o velikosti jednoho nebo dvou bajtů. Příkaz samotný je poté složen z dat, která jsou reprezentována příkazem a volitelnými data, pokud příkaz tuto možnost podporuje. Nedílná součást každého příkazu je CRC, který se počítán za samotných dat, nikoliv synchronizačních bajtů indikaci počátku a konce datového paketu [2].

7.1.2 Tabulka odpovědí

Dle odpovědí na přijatý příkaz jsme schopni detekovat, jestli jsou data v pořádku nebo ne. Toto platí při synchronní komunikaci, kdy jednotka odpoví pouze tehdy, vyzve ji k tomu uživatel odesláním nějakého povelu [2].

Tab. 6 - Odpovědi na příkazy poslané jednotkou [2]

Název odpovědi HEX hodnota	Data	Popis
ReplyOK 0xa5	Žádná	Odpověď OK, zadaný příkaz byl proveden
ReplyTest 0xa3	1 bajt z parametru testovacího příkazu	Odpověď na testovací příkaz, vrácení bajtu z jeho parametru
ReplyTorque 0xa7	4 dvoubajtové hodnoty naměřených momentů	Za odpovědí následují 4 dvoubajtové hodnoty naměřených momentů v pořadí kanál 1 až kanál 4
PackSubstError 0xf1	Žádná	Chyba zástupného znaku v přijatém paketu SLIP
PackShortError 0xf2	Žádná	Přijatý paket je kratší, než bylo očekáváno
PackLongError 0xf7	Žádná	Přijatý paket je delší, než bylo očekáváno
PackCrcError 0xf3	Žádná	CRC v přijatém paketu nesouhlasí s vypočteným
CmdError 0xf4	Žádná	Chyba formátu přijatého příkazu
CmdUnknown 0xf5	Žádná	Přijat neznámý příkaz
CmdBlocked 0xf6	Žádná	Přijatý příkaz je blokován, nyní ho nelze provést

7.2 Implementace CRC

Poslední dva B každého přijatého i odeslaného paketu obsahují kontrolní součet. Konkrétně se jedná o CRC-CCITT s výchozí hodnotou 0xFFFF. Následná operace mezi vypočteným CRC a konstantou se neprovádí [2].

7.3 Prvotní zpracování naměřených dat

Algoritmus pro prvotní zpracování dat v modulu pro měřící jednotku musí brát ohled na to, že měřící jednotka posílá informace o naměřených údajích kumulativně. V příchozí data neustále rostou, a když dojde k přeplnění bitového rozsahu v interních proměnných, tak dojde

k vynulování a tím počítání opět od nuly. Nedojde tím však ke ztrátě nesoucí informace, je totiž proveden přenos do vyššího bitu, které již ale není implementován ani zobrazen [2].

Modul musí umět detekovat přetečení, tj. náhlý pokles hodnoty téměř z konce měřícího rozsahu na téměř začátek, a správně tento stav ošetřit. Princip ošetření je uvede v deváté kapitole, v popisu metody **CheckTimes()** [2].

8 Modul pro komunikaci s měřící jednotkou

Tato kapitola rozebere podrobně vlastnosti a funkcionality samotného modulu, v našem případě dll knihovny vytvořené v programovacím jazyce C#.NET pomocí vývojového prostředí Microsoft Visual Studio 2010. Dynamicky linkovaná knihovna, z angličtiny jako dynamic link library, je soubor procedur, funkcí, metod a datových typů, které můžou být sdíleny více počítačovými programy.

Jednotlivé kapitoly postupně proberou základní struktury modulu, konkrétně z jakých s skládá tříd, včetně popisu jejich atribut a metod. Celá kapitola bude provázena vývojovými diagramy pro lepší pochopení dané problematiky a praktické ukázky, jak knihovnu použít v simulačním prostředí LabVIEW.

Celá knihovna byla používána a testována v softwaru LabVIEW 2012, který zároveň posloužil jako nástroj pro tvorbu ukázek použití dané knihovny. LabVIEW jako software sám o sobě je ideální pro všechny měřící systémy. Integruje v sobě většinu nástrojů, které inženýři a vědci potřebují, ale lze jej použít pro budování široké škály aplikací ve výrazně kratším čase, oproti tomu, který strávíme tvorbou samostatné aplikace. Je to nástroj, vývojové prostředí, pro řešení problémů a zrychlení produktivity.

8.1 Programovací jazyk C#

Visual C# je moderní, více vzorový, univerzální programovací jazyk vyšší úrovně pro stavební aplikace pomocí vývojového prostředí Visual Studio a. NET Framework. C# je navržen tak, aby byl jednoduchý, silný, typově bezpečný, a objektově-orientovaný. Mnoho inovací v C# umožňuje rychlý vývoj aplikací při zachování výraznost a elegance C-styl jazyků [20].

8.1.1 Koncepce pojmenování a značení

Pro lepší čitelnost, orientaci a pochopení kapitol, které budou této následovat, vznikal interní koncept značení a vysvětlivek, vyskytujících se v části vlastního přínosu této diplomové práce.

Textové popisy jsou formátovány následovně, jak je uvedeno v Tab. 7 - Konvence značení a formátování.

Vývojové diagramy jsou vždy koncipovány tak, že elipsovitý tvar na začátku a konci diagramu uvozuje jeho začátek, respektive konec, obdélníkový tvar znační určitou akci a kosočtverec značí větvení, nebo také rozhodující blok.

Tab. 7 - Konvence značení a formátování

Entity	Koncepce značení
Třídy	Všechny třídy, které se budou v programu nacházet, jsou značeny velkým počátečním písmenem a stylem *kurzivou*.
Atributy	Atributy (instanční proměnné) začínají malým písmene a jsou značeny stylem formátování ***tučně s kurzivou***.
Metody()	Metody v textu začínají názvem s velkým počátečním písmene a kulatými závorkami, do který můžeme psát případné parametry volání těchto metod. Styl formátování je **tučně**.

8.1.2 Datové typy a proměnné jazyka C#

Jakákoliv entita v programu C# je objekt, který je buď v zásobníku, nebo ve spravované skupině. Metody si definujeme pomocí deklarace tříd nebo struktur. V programovacím jazyce C# není možné vytvořit nic takového, jako jsou volné funkce, definované mimo deklaraci třídy nebo struktury jako je tomu v jazyce C++. Dokonce i vestavěné datové typy, jako jsou int, long, double atd., mají své metody, ke kterým můžeme přistupovat, jelikož jsou s nimi implicitně sdruženy. Nejznámější datové typy, které jsou součástí jazyka C# a které zároveň používáme v naprogramované knihovně, jsou uvedeny v tabulce 8 níže. [20]

Tab. 8 - Datové typy použité v knihovně [18]

Krátké označení	Třída .NET	Typ	Šířka	Rozsah
byte	Byte	Unsigned integer Neoznačený integer	8	0 až 255
int	Int32	Signed integer Označený integer	32	-2147483648 až 2147483647
uint	UInt32	Unsigned integer Neoznačený integer	32	0 až 4294967295
ushort	Int16	Unsigned integer Neoznačený integer	16	0 až 65535
long	Int64	Signed integer Označený integer	64	-9223372036854775808 až 9223372036854775807
ulong	UInt64	Unsigned integer Neoznačený integer	64	0 až 18446744073709551615
float	Single	Single-precision floating point type Jednodná presnost s plovoucí desetinnou čárkou	32	-3,402823e38 až 3,402823e38
double	Double	Double-precision floating point type Dvojitá přesnost s plovoucí desetinnou čárkou	64	-1,79769313486232e308 až 1,79769313486232e308
bool	Boolean	Logical Boolean type Logický typ boolean	8	TRUE nebo FALSE 0 nebo 1
object	Object	Base type of all other types Základní typ všech ostatních typů		
string	String	A sequence of characters Sekvence znaků		

8.1.3 Modifikátory jazyka C#

Modifikátor, nebo jak bychom správně měli nazývat modifikátor přístupu, se nachází před klíčovým slovem class a určuje viditelnost typu vně sestavení. Jejich používání může programátorům připadat intuitivní, pokud již má nějaké zkušenosti s jiným objektově orientovaným programovacím jazykem [20].

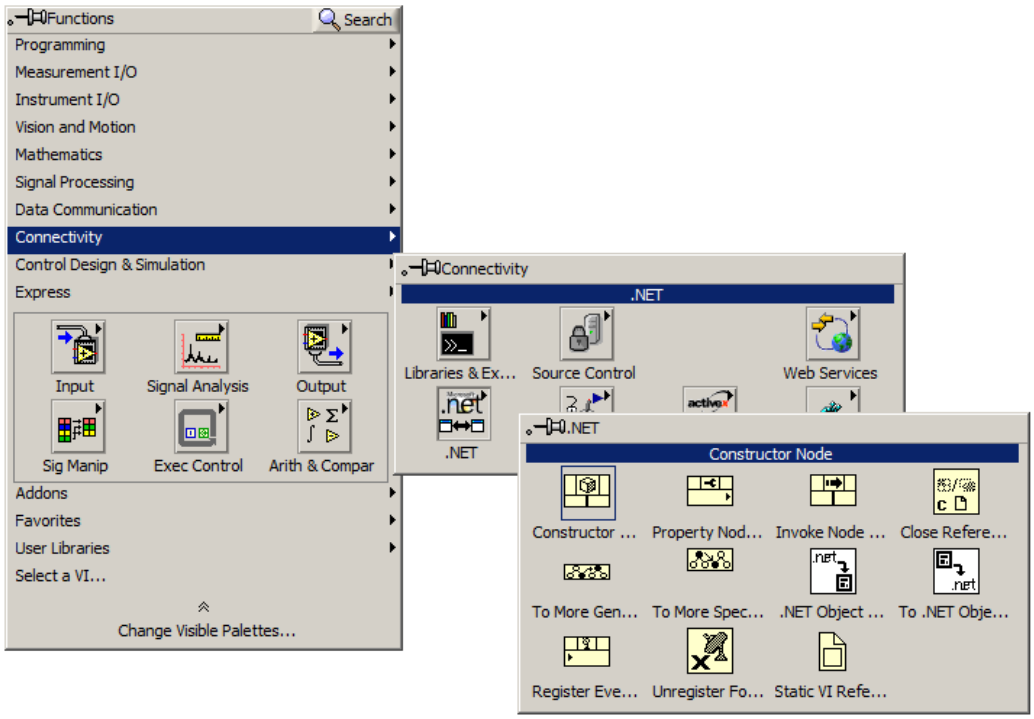
Modifikátor přístupu lze v zásadě použít na jakoukoliv entitu, která se nachází v programu C#, včetně tříd a jakýchkoliv jejich členů, jakými jsou atributy, metody, konstruktory a další. Modifikátor přístupu, který je na třídu aplikován, ovlivňuje viditelnost třídy vně sestavení, ve kterém se nachází. Níže uvedená tabulka obsahuje popis různých modifikátorů přístupu, které můžeme v programovacím jazyce C# použít a které jsou zároveň použity v modulu, který byl vytvořen pro měřící jednotku [20].

Tab. 9 - Modifikátory přístupu v jazyce C# [17]

Modifikátor přístupu	Význam
public	Člen je úplně viditelný vně definující oblasti platnosti a interní oblasti platnosti. Jinými slovy, přístup k veřejnému členu není nijak omezen.
protected	Člen je viditelný pouze definující třídě a kterékoliv třídě, která je od ní odvozená.
internal	Člen je viditelní uvnitř sestavení, které ho obsahuje. To zahrnuje definující třídu a všechny oblasti platnosti mimo ní, které jsou součástí sestavení.
protected internal	Člen je viditelný v rámci definující třídy a všude jinde v sestavení. Tento modifikátor, jak z názvu vypovídá, kombinuje modifikátory protected a internal prostřednictvím logické operace OR. Člen je viditelný pro jakoukoliv třídu, která je odvozená od třídy definující, ať už je ve stejném sestavení, nebo v jiném.
private	Člen je viditelný pouze v definující třídě, bez výjimky. Jedná se o nepřísnější formu přístupu a o výchozí nastavení přístupnosti všech členů třídy.

9 Použití knihovny v aplikaci LabVIEW

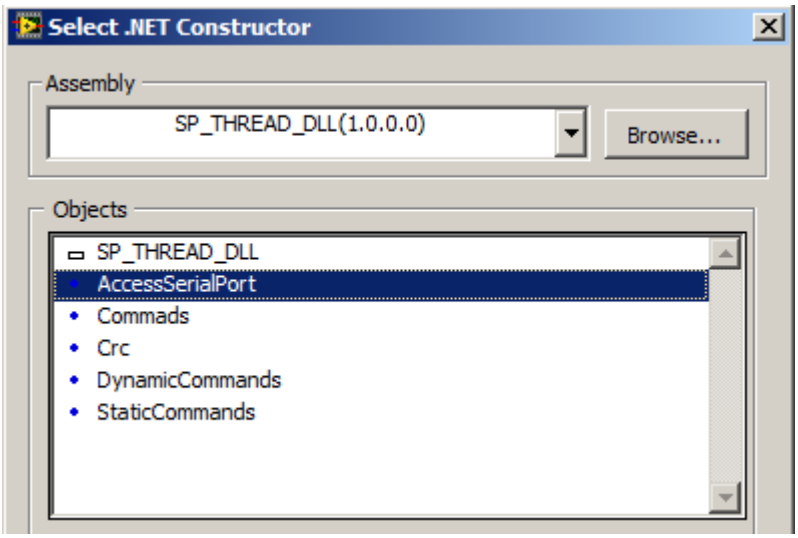
Pro přístup k metodám a instančním parametrům knihovny, je nutné knihovnu provázat s aplikací. Toto je možné provést v blokovém diagramu, do se přepneme stiskem klávesové zkratky CTRL+E. Pravým kliknutím na bílou plochu a vybráním bločku ze skupiny *„Conectivity/.NET“* vybereme konstruktor, pro vytvoření nové instance třídy. V LabVIEW se tato funkční část jmenuje *„Constructor Node“*. Názorné použití je uvedeno na obrázku pod odstavcem.



Obr. 10 - Přístup k funkčním bločkům pro užití knihovny

Po vybrání vytvořené knihovny jsme schopni zavolat námi zvolený konstruktor, pro komunikaci s jednotku slouží konstruktor třídy *AccessSerialPort* a dále pak přistupovat k jeho metodám a vlastnostem.

Pomocí bločku *„Invoke Node (.NET)“* jsme v LabVIEW schopni přistupovat k jednotlivým metodám třídy, u které jsme zavolali konstruktor a pomocí bločku *„Property Node (.NET)“* pak přistupujeme instančním proměnným, vlastnostem, ze kterých pak vyčítáme a jsme schopni dále zpracovávat jednotlivé návratové hodnoty zaslané měřící jednotkou.



Obr. 11 - Příklad vytvoření konstruktoru třídy

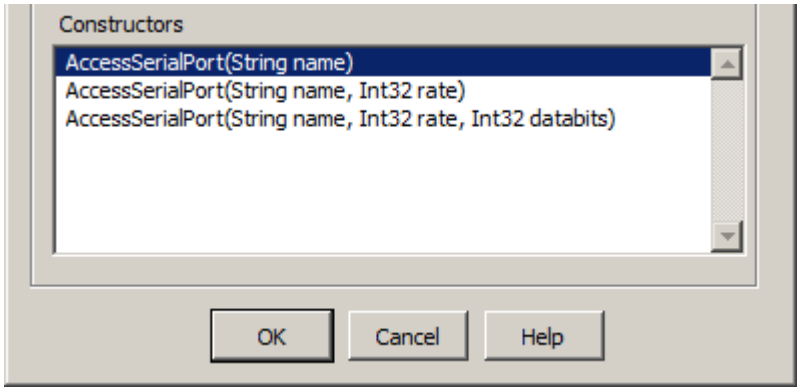
9.1 Základní popis knihovny

Celá knihovna jak je naprogramovaná, využívá systémové komponenty, které již jsou součásti operačního systému. Nejstěžejnější pro naši knihovnu je obor hodnot System.IO.Ports, ze kterého přebíráme třídu *SerialPort*, pomocí které je pak uživatel schopen komunikovat s měřící jednotkou. Třídu *SerialPort* můžeme použít pro řízení souboru prostředků sériového portu. Třída sama o sobě obsahuje událostmi řízené I/O, přístupy k pinům a hraničním stavům sériového zařízení. Pro komunikaci po sériovém rozhraní byla tedy vytvořena třída *AcessSerialPort*, kde se nacházejí veškeré atributy a metody pro komunikaci s měřící jednotkou. [21]

Další třídou, která se nachází v naší knihovně je třída *StaticSommands*. Po zavolání konstruktoru této třídy pak jsme schopni odeslat po sériové lince příkaz měřící jednoty. Příkazy, které je možné jednotce poslat, jsou popsány v kapitole Měřící jednotka pro válcovou zkušebnu vozidel. Veškeré vlastnosti třídy jsou statické, nijak se nemění a byli vytvořeny za účelem obhájení funkčnosti třídy *DynamicCommands*, kterou jsme schopni jednotce zaslat stejné příkazy. Při použití této knihovny se ovšem nejedná o statické pole B, které posíláme jednotce, ale o dynamicky vytvářená pole. Třída *Crc* je velice důležitá, pokud chceme, aby příkazy, které dynamicky generuje třídou *DynymicCommands* byli akceptovány měřící jednotkou. Jak je totiž uvedeno v kapitole o formátu rámce, který měřící jednotka akceptuje,

tak nebudeme-li počítat synchronizační bajty, tak poslední dva poslané B se skládají z kontrolního součtu, který se počítá z dat, která chceme jednotce poslat.

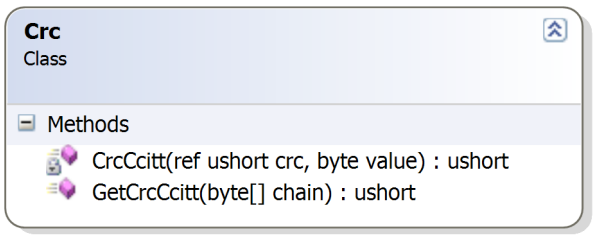
Pro možnost přístupu k metodám třídy *AccesSerilaPort* a ostatním, je nutné vytvořit instanci třídy podle toho, ke kterým metodám chceme přistupovat. Třída, kterou popisuji v této kapitole třída *AccessSerialPort*, obsahuje dva konstruktory, jak je názorně ukázáno na obrázku níže.



Obr. 12 - Volba konstruktoru třídy

V podkapitolách níže uvedených budeme rozebírat jednotlivé třídy, jejich atributy, metody a jak vlastně samotné metody fungují. Z důvodu úspory místa, u každé metody není uveden její zdrojový kód. Kompletní zdrojový kód programu bude uveden na přiloženém CD, včetně samotného projektu vývojového prostředí vytvořeného ve Visual Studiu 2010 a zkompilované knihovny.

9.2 Třída Crc



Obr. 13 - Návrh třídy Crc

9.2.1 Popis třídy

Ze všech tříd, které knihovna pro komunikaci s měřícím zařízení obsahuje, se skládá pouze ze dvou metod. Tyto dvě metody jsou však pro komunikaci a následnou akceptaci příkazu jednotkou nejdůležitější. Pokud chceme vypočítat CRC ze vstupních dat, musíme použít meto **GetCrcCcitt()**, která jako vstupní parametr potřebuje libovolně dlouhé pole

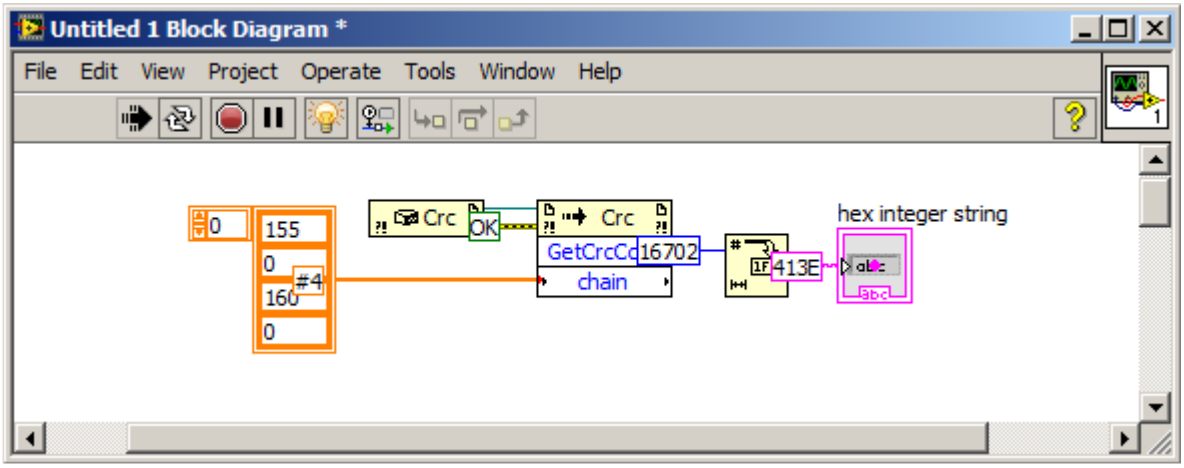
datové typu byte a jako návratovou hodnotu vrací datový typ **ushort**, z důvodu vymaskování bitů, které jsou dočasné proměnné pro výpočet cyklického kontrolního součtu ukládány. Metoda je v programu vytvořena takto:

```
public static ushort GetCrcCcitt(byte[] chain)
        {
            //Nastavení počáteční hodnoty
            ushort crc = 0xFFFF;
            //Cyklus pro opakované volání funkce výpočtu kontrolního součtu.
            for (int i = 0; i < chain.Length; i++)
            {
          //Uložení CRC do výstupní proměnné.
                crc = Crc.CrcCcitt(ref crc, chain[i]);
            }
            return crc;
        }
```

Jak je z kódu vidět, tak samotná metoda je nastavena jako public, tzn., že pokud zavoláme konstruktor třídy Crc, budeme moci přistupovat právě k těm metodám, které jsou takto nastaveny. Pro samotný výpočet se použije metoda **CrcCcitt()**, ve které proběhne výpočet, podle postupu, uvedeného v kapitole o kontrolních součtech. K této metodě není schopen uživatel přistupovat přímo, modifikátor této metody je nastaven jako private.

Pro názornou ukázku je na

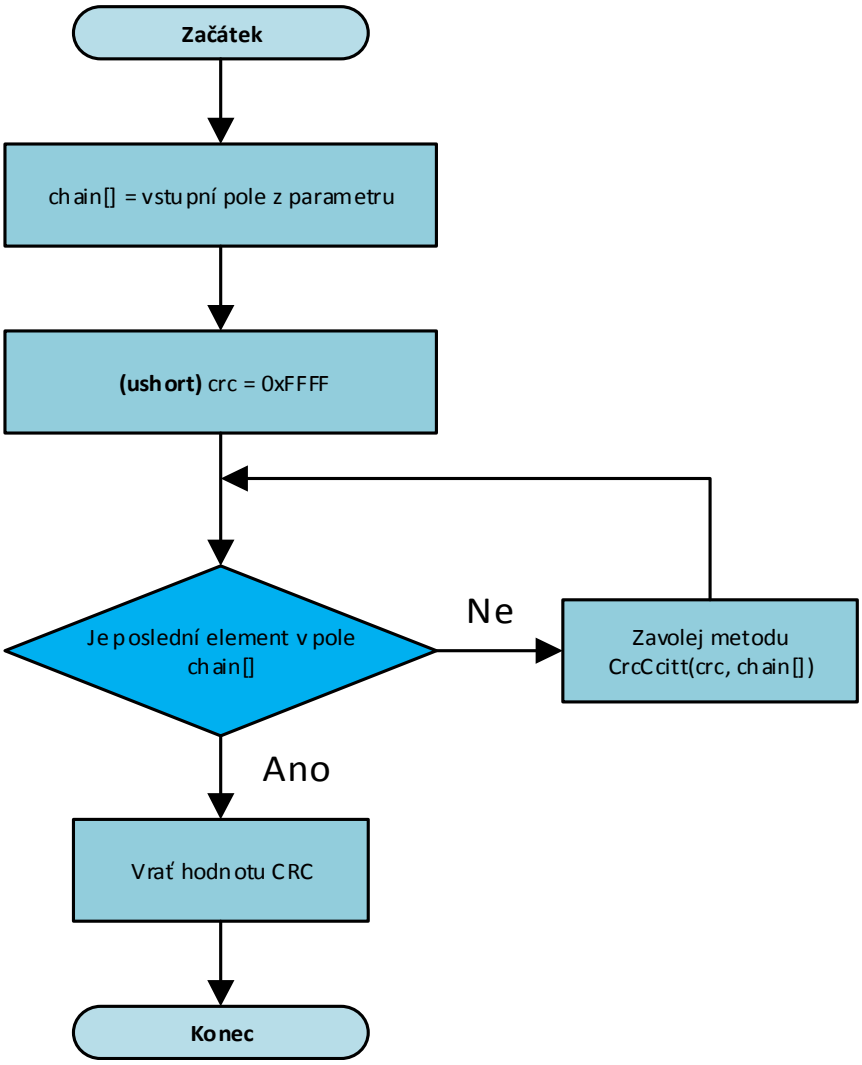
Obr. 14 je vidět praktická možnost použití třídy *Crc*, konkrétně metody **GetCrcCcitt()**, kde vstupním řetězcem bylo číslo $(9B00A000)_{HEX}$ a jako návratová hodnota byl vypočtený kontrolní součet, který byl roven hodnotě $(413E)_{HEX}$.



Obr. 14 - Praktické použití knihovny Crc v LabVIEW

Algoritmus samotného výpočtu je uveden na vývojových diagramech na Obr. 15 a Obr. 16, ze kterých čtenář pochopí, jak přesně metody pro výpočet fungují.

9.2.2 Metoda GetCrcCcitt(byte[] chain)

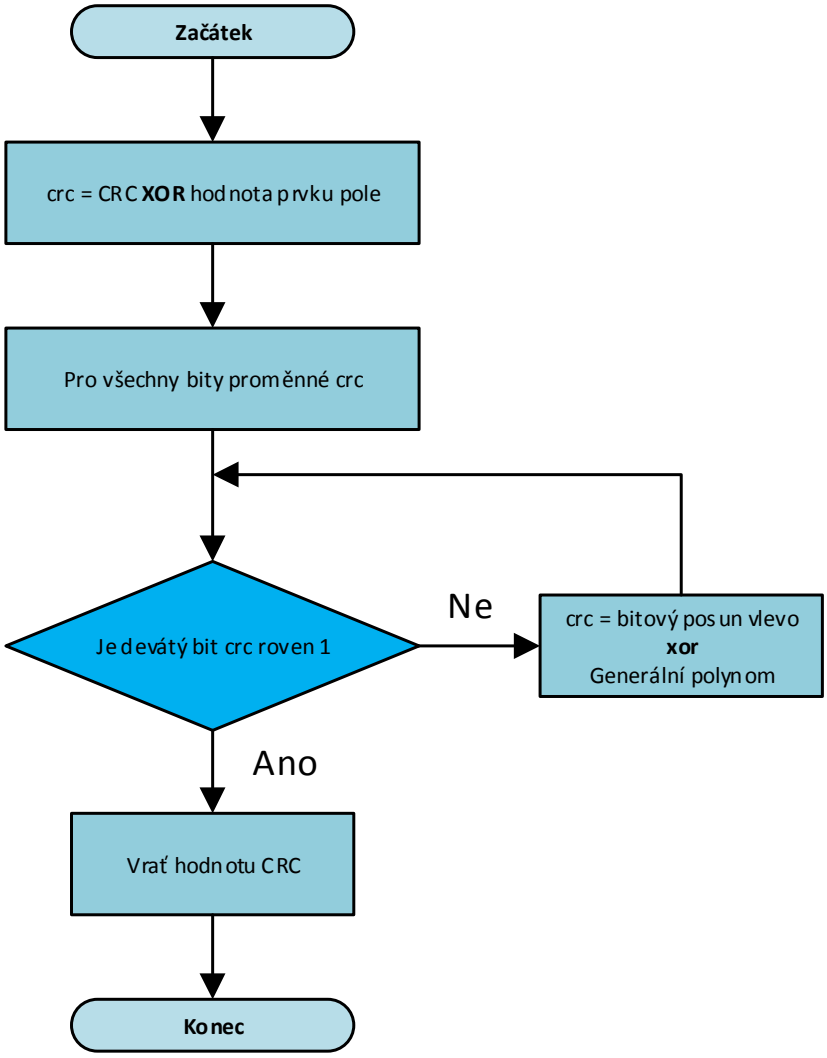


Obr. 15 - Vývojový diagram metody GetCrcCcitt()

9.2.2.1 Popis metody

Po zavolání metody **GetCrcCcitt()** se vstupním parametrem pole datového typu byte, je v paměti programu alokována počáteční hodnota cyklického redundantního součtu. Měřící jednotka používá CRC s počáteční hodnotou $(FFFF)_{HEX}$. Metoda kontroluje pomocí iteračního cyklu for, jestli je načten poslední prvek vstupního pole a volá metodu **CrcCcitt()** se vstupními parametry počáteční hodnoty a hodnoty prvku pole, dle čísla iterace cyklu.

9.2.3 Metoda CrcCcitt(ref ushort crc, byte value)

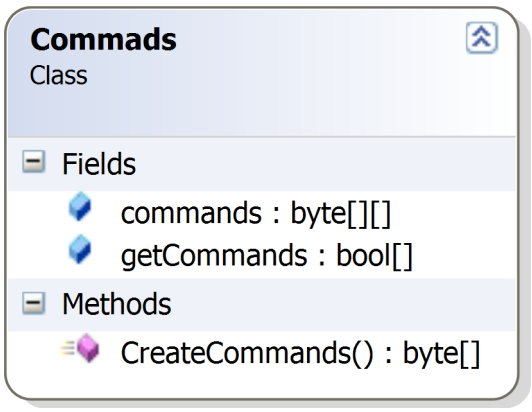


Obr. 16 - Vývojový diagram metody CrcCcitt()

9.2.3.1 Popis metody

Z proměnných načtených ze vstupních argumentů metody, je s proměnnou ***crc*** provedena logická operace XOR s hodnotou aktuálně načteného prvku pole. Pro všechny bity proměnné ***crc*** se kontroluje, jestli je na pozici devátého bitu logická jednička. Pokud není, provede opět funkce XOR dle generálního polynomu pro výpočet CRC, v našem případě je to hodnota $(11021)_{HEX}$. Jelikož kontrolujeme pozici devátého bitu počáteční podmínkou, provádí se logická funkce XOR pouze podle hodnoty $(1021)_{HEX}$.

9.3 Třída Commands

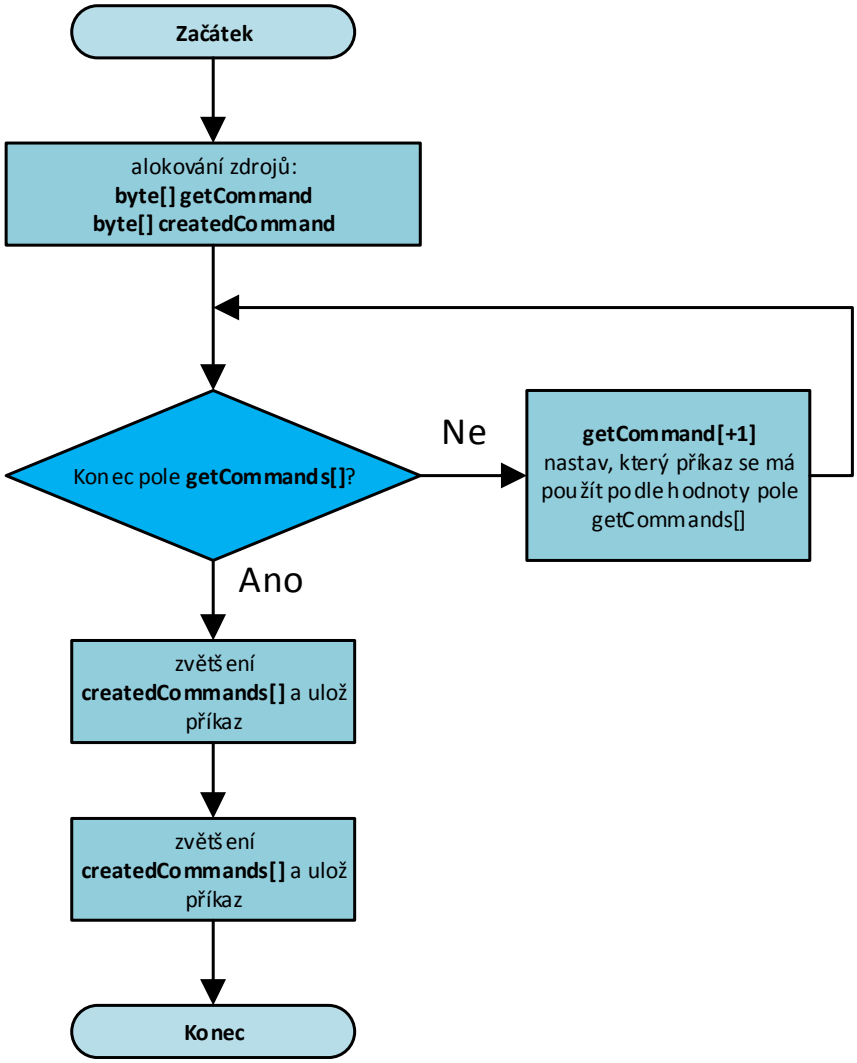


Obr. 17 - Návrh třídy Commands

9.3.1 Popis třídy

Třída *Commands* slouží jako mateřská (nadtřída) třída pro třídy *StaticCommands* a *DynamicCommands*. Obsahuje jednu metodu a dva atributy, které zároveň používají obě podtřídy. Atribut ***getCommands*** je pole datového typu bool, do kterého se v podtřídách ukládá nastavení z konstruktorů tříd. Atribut ***commands*** je pole polí datové typu byte, kde jsou uloženy atributy nebo metody podle toho, které podtřída tento atribut použije. Třída sice má veřejný konstruktor, ale jelikož jediná metoda, která je schopna vrátit vygenerovaný příkaz pro jednotku, reaguje pouze na zavolání z potomků třídy, tak pro uživatele v programu LabVIEW jeví jako metoda pouze ke čtení. V příloze na obrázku přílohy 4 je pak znázorněn vztah mezi třídami *StaticCommand*, *DynamicCommand* a třídou *Commands*.

9.3.2 Metoda CreateCommands()



Obr. 18 - Vývojový diagram metody CreateCommands

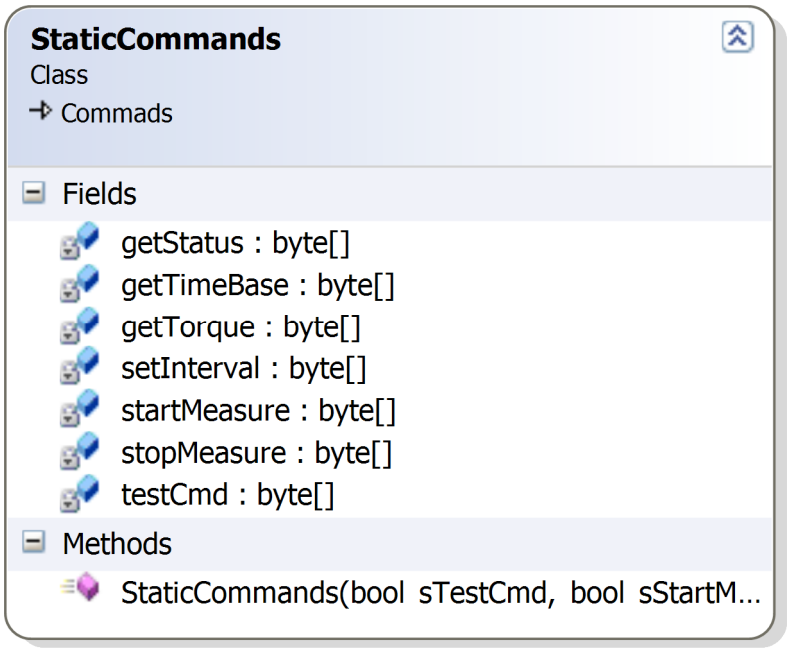
Metoda **CreateCommands()** byla vytvořena za účelem vytvoření příkazů, které vyplynou z konstruktorů tříd *StaticCommands* a *DynamicCommands*. Tyto dvě třídy mají jako vstupní parametry booleovské proměnné, kde podle jejich hodnot skládáme výsledný příkaz, který posíláme měřící jednotce. Zavoláním konstruktoru jedné z podtříd se to atributu ***getCommands*** uloží nastavení volaného konstruktoru, podle kterého určujeme, jaký příkaz pošleme jednotce. V proměnné ***commands***, kterou si pro zjednodušení můžeme představit jako tabulku, kde v první sloupec určuje, zdali máme příkaz použít a ve sloupci druhém, uložíme příkaz ve formátu akceptovatelném pro měřící jednotku. Pro usnadnění pochopení atributu je pod odstavcem uvedena Tab. 10, která zobrazuje strukturu proměnné.

Metoda **GetCommands()** pak zjistí dle prvního sloupce použití příslušného příkazu, příkazy si uloží do vnitřní proměnné, a když dojde k poslednímu řádku tabulky, tak vytvořenou proměnnou vrátí uživateli, který ji pak může pomocí Třídy *AccessSerialPort*, konkrétně metody **SendCommand()** odeslat měřící jednotce. Tento způsob byl volen pro usnadnění práce s vygenerováním povelu, na který by mohla jednotka odpovědět.

Tab. 10 - Složení příkazu

Použít příkaz	Příkaz
True/False	TestCmd
True/False	StartMeasure
True/False	StopMeasure
True/False	SetInterval
True/False	GetTimeBase
True/False	GetTorque
True/False	GetStatus

9.4 Třída StaticCommand



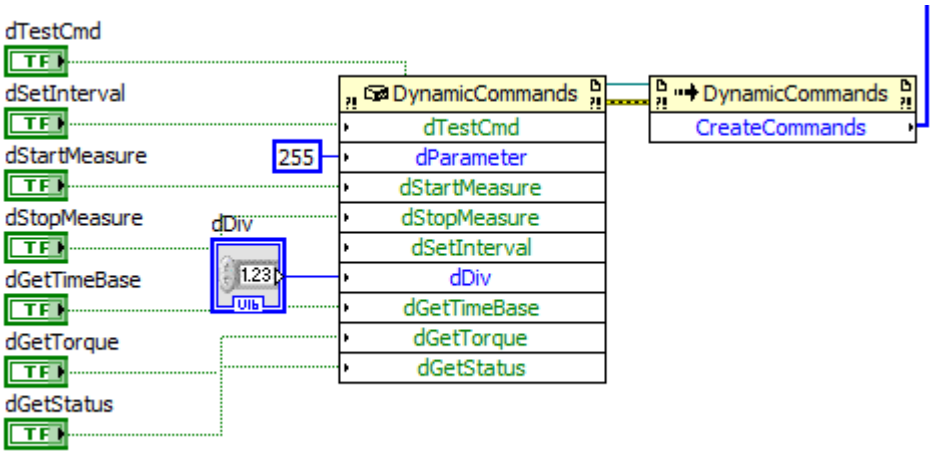
Obr. 19 - Návrh třídy StaticCommands

9.4.1 Popis třídy

Třída *StaticCommands* je specifická tím, že obsahuje pouze jeden konstruktor třídy, metodu zděděnou od třídy *Commands* a 7 atributů (instančních proměnných), která mají

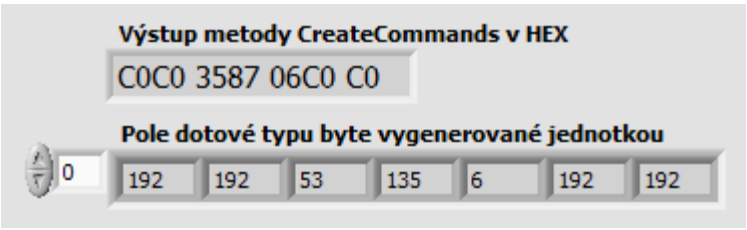
modifikátor private. Abychom mohly k těmto vlastnostem přistupovat, musíme zavolat, konstruktor třídy. Všechny atributy, které se v třídě nacházejí, jsou statické a nic nepočítají ani dynamicky negenerují. Tato třída byla vytvořen na základě tabulky příkazů, které je jednotka schopna akceptovat a také z důvodu ověření, zdali příkazy, které jsou dynamicky generované třídou *DynamicCommands* fungují, a jestli skutečně umíme cyklický redundantní součet vypočítat. Dle parametrů konstruktoru třídy (logických hodnot) je poté vygenerováno pole datového typu byte, ve formátu akceptovatelném měřící jednotkou.

9.4.2 Použití třídy v aplikace LabVIEW



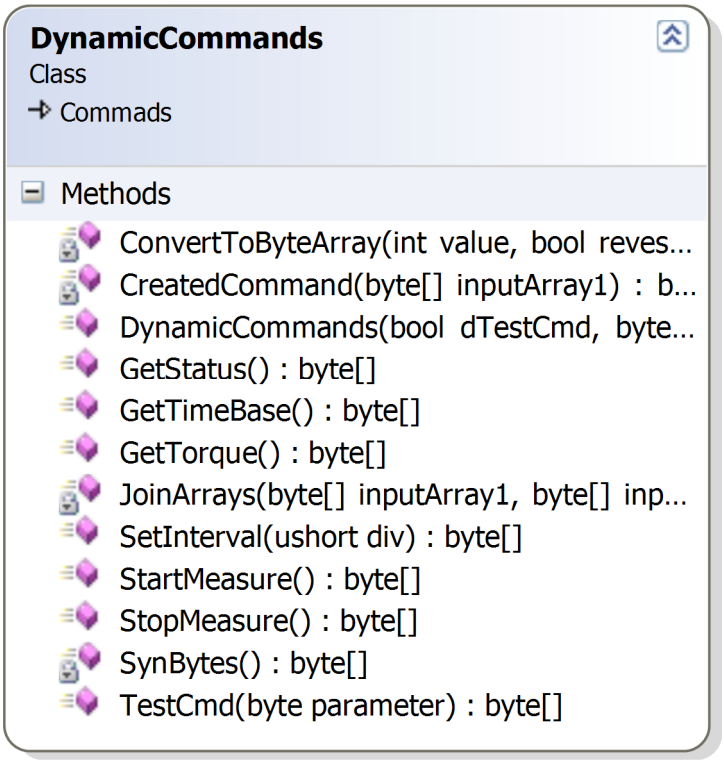
Obr. 20 - Použití třídy StaticCommands v aplikaci LabVIEW

Pro přístup k metodám třídy *StaticCommands* je nutné zavolat konstruktor dané třídy. Konstruktor třídy má 7 parametrů, každý parametr je logický funkce. Pole toho, kolika parametrů je nastavena logická hodnota true, je zavolána metoda **CreateCommands()**, které je zděděna třídou *Commands*. Tato metoda zároveň vygeneruje příkaz pro jednotku akceptovatelném formátu, příklad je uveden na obrázku 21 pod odstavcem.



Obr. 21 - Příkaz StartMeasure vygenerovaný pomocí třídy StaticCommands

9.5 Třída DynamicCommands



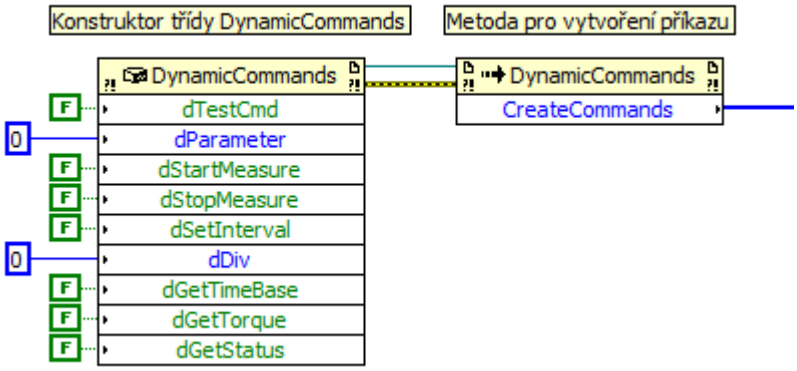
Obr. 22 - Návrh třídy DynamicCommands

9.5.1 Popis třídy

Třída *DynamicCommands*, co se návratových typů týká, je velmi podobná třídě *StaticCommands*, až na jeden významný rozdíl. Ve třídě *StaticCommands* nejsou žádné metody a její veškeré výstupy jsou ve formě statického pole datového typu byte. U této třídy je tomu jinak.

Třída neobsahuje atributy, ale je složena z mnoha metod. Metody mají modifikátory public i private. A uživatel, pokud k nim chce programově přistupovat, tak vidí pouze metody s modifikátorem public, jak je vidět na obrázku 24 níže. V podkapitolách si jednotlivé metody rozebereme, jak fungují, ukážeme si možné způsoby použití v aplikaci LabVIEW a hlavně jaké vracejí proměnné.

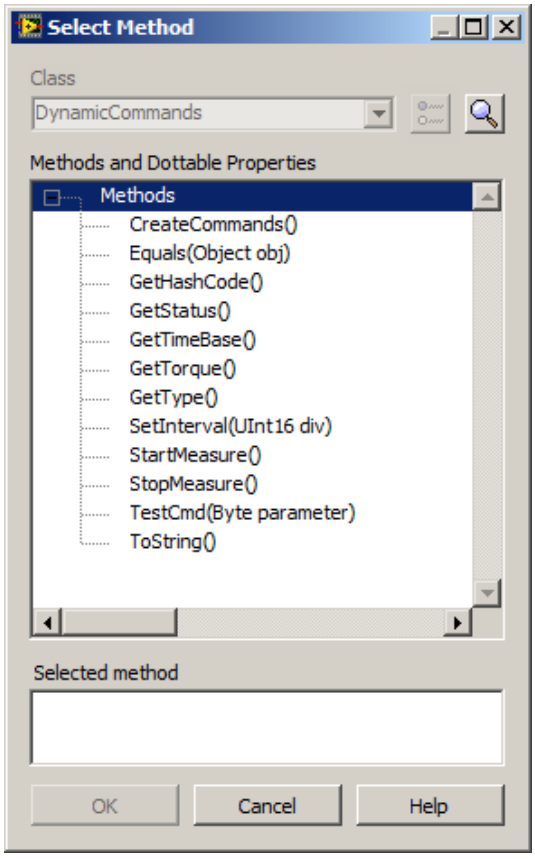
9.5.2 Použití třídy v aplikace LabVIEW



Obr. 23 - Použití třídy DynamicCommands v aplikaci LabVIEW

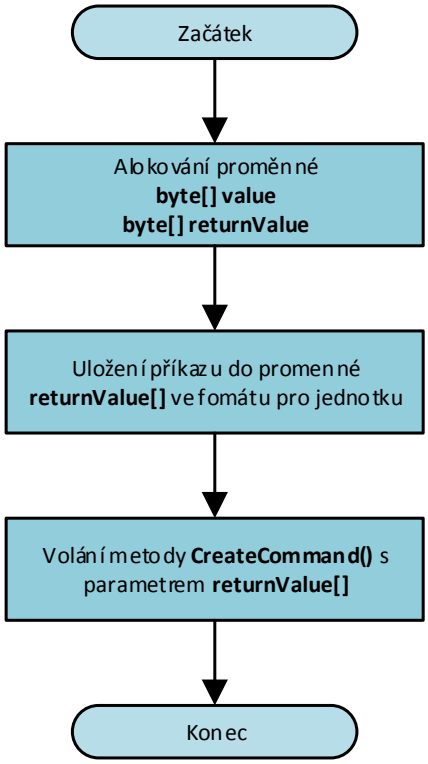
Pro přístup k metodám je nutné zavolání konstruktoru třídy. Pro laboratorní účely jsou u většiny metod ponechány modifikátory public.

Níže uvedený Obr. 24 zobrazuje názorný příklad, které metody jsou dostupné uživateli, když použije konstruktor třídy *DynamicCommands*. Na náhledu, který byl vytvořen pomocí simulačního prostředí LabVIEW, jsou vidět veškeré metody třídy, které mají nastaven modifikátor public.



Obr. 24 - Metody třídy DynamicCommands.

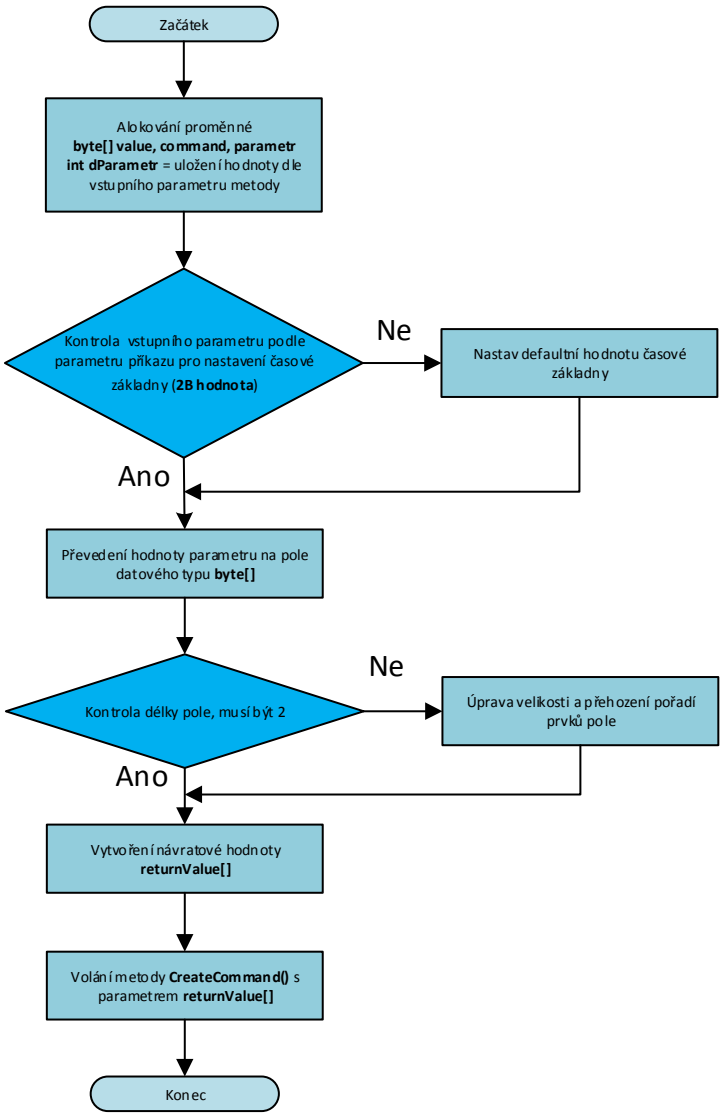
9.5.3 Metoda GetStatus()



Obr. 25 - Vývojový diagram metody GetStatus()

Výše uvedený vývojový diagram metody **GetStatus()** názorně ukazuje, jak metoda funguje. Metoda není nijak složitá, až na druhý krok, který ne, že by byl nijak zvlášť složitý, ale pro vytvoření příkazu ve formátu akceptovatelném jednotkou je potřeba volání dvou dalších metod, konkrétně metody **JoinArrays()** a **GetCrcCcitt()**. Princip těchto metod je pochopitelně také popsán, ale ne v tomto odstavci. Metody **GetTimeBase()**, **GetTorque()**, **StarMeasure()**, **StopMeasure()** fungují totožně, analogicky jako metoda **GetStatus()**, proto nejsou zahrnuty do popisu metod třídy *DynamicCommands*.

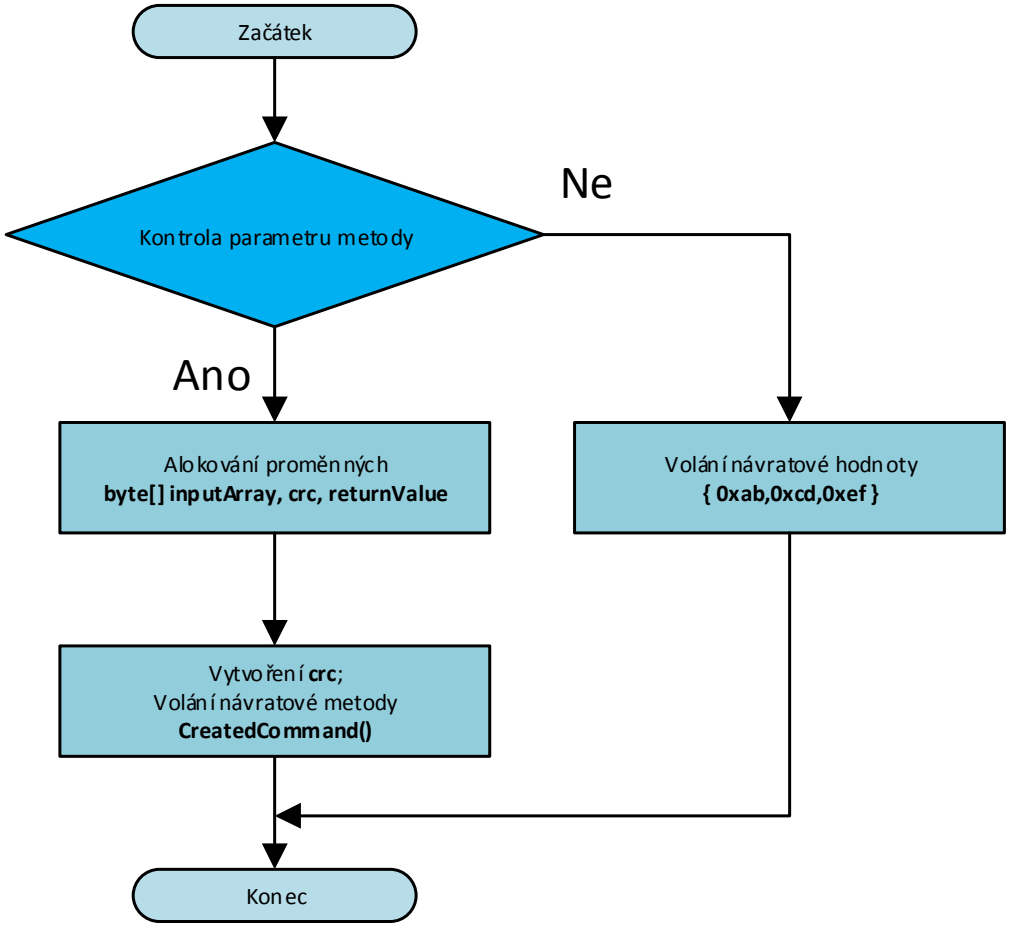
9.5.4 Metoda SetInterval(ushort div)



Obr. 26 - Vývojový diagram metody SetInterval()

Vývojový diagram popisuje princip metody **SetInterval()**. V metodě opět používáme metody pro spojení polí a vypočet kontrolního součtu. Z důvodu úspory místa je obrázek ve vyšším rozlišení v přílohách na příloze 1 stránka A.

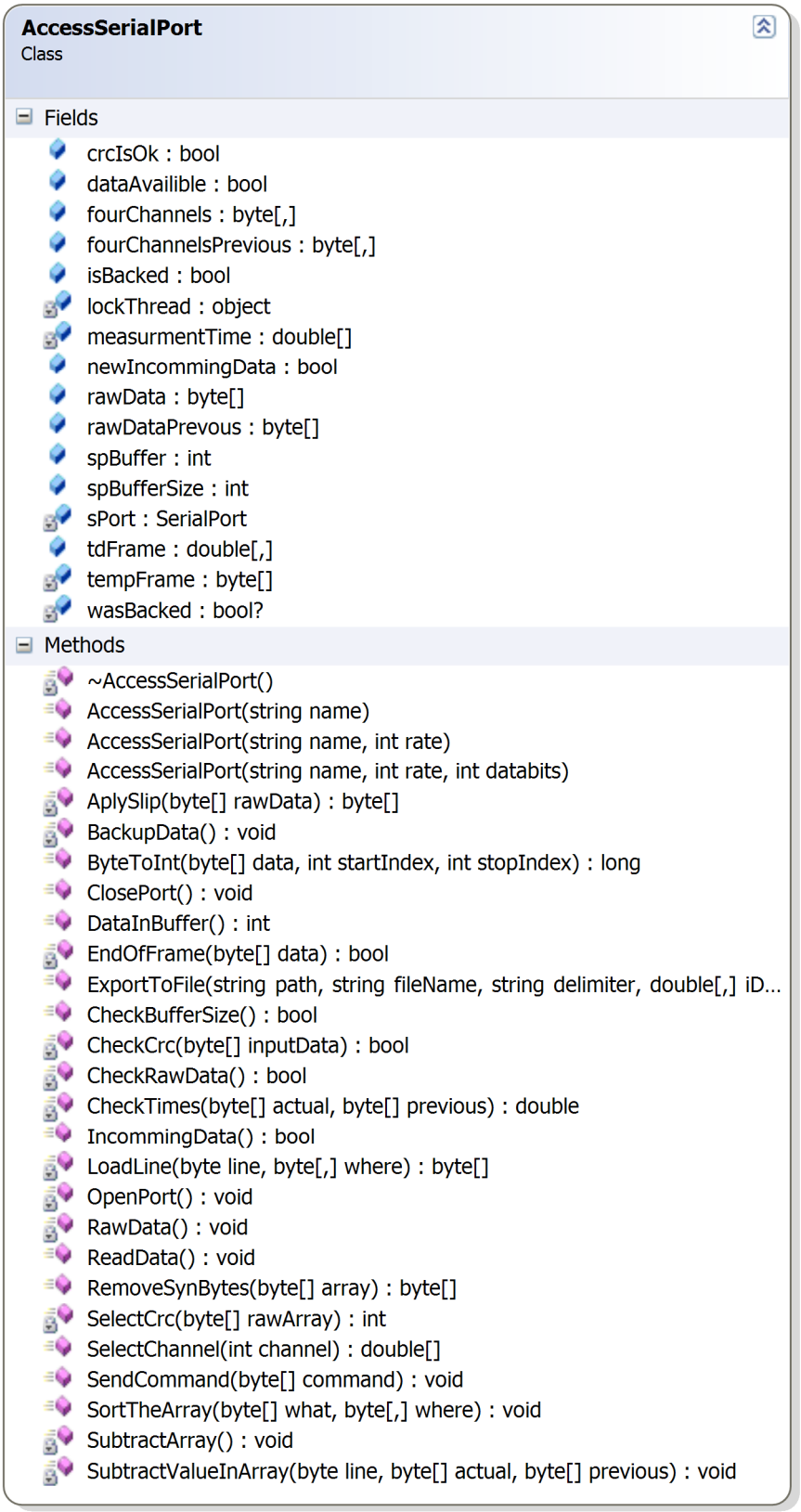
9.5.5 TestCmd(byte parameter)



Obr. 27 - Vývojový diagram metody TestCmd()

Metoda **TestCmd()** funguje, jak je popsáno na výše uvedeném vývojovém diagramu. Pokud by se ošetření vstupu je v tomto případě provedeno tak, že pokud by se uživateli podařilo zavolat metodu se vstupním parametrem mimo rozsah datového typu byte, metoda vrátí pole o třech prvcích s hodnotou $(0xAB,0xCD,0xEF)_{HEX}$.

9.6 Třída AccessSerialPort



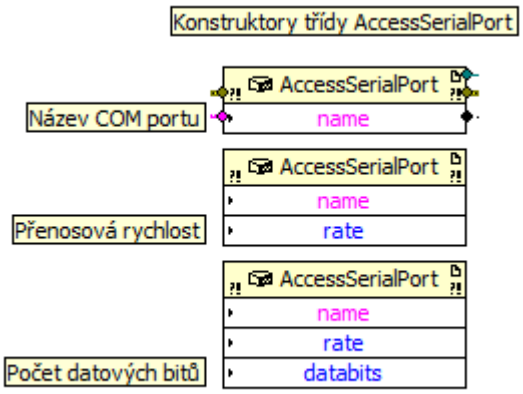
Obr. 28 - Návrh třídy AccessSerialPort

9.6.1 Popis třídy

Pro samotný přístup a obsluhu měřící jednotky je tato třída nejpodstatnější, protože osahuje všechny metody, konstruktory a návratové hodnoty, které jsou schopny reagovat na vstupy od uživatele.

Samotná komunikace s měřící jednotku závisí na zavolání příslušného konstruktoru třídy. Třída sama o sobě obsahuje tři konstruktory, kde každý nastaví sériový port, podle parametrů zadaných v konstruktoru třídy a otevře sériovou komunikaci. Rozdílnosti jednotlivých konstruktorů si probereme v následujících kapitolách.

9.6.2 Konstruktory třídy AccessSerialPort

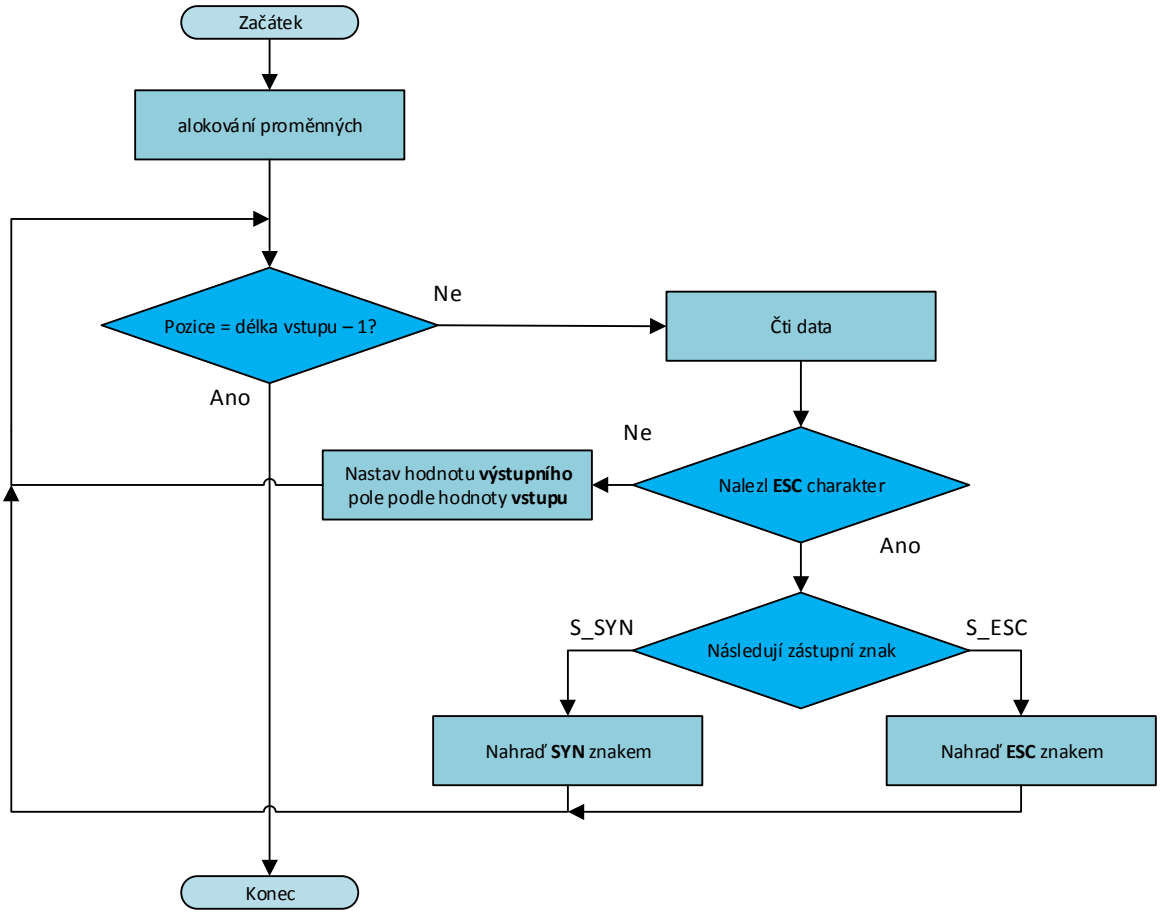


Obr. 29 - Konstruktory třídy AccessSerialPort

Nad odstavcem vidíme na obr. 29 výčet konstruktorů reprezentací v aplikaci LabVIEW. První konstruktor má jeden vstupní argument, a to název COM portu, na kterém je připojená měřící jednotka. Při použití tohoto konstruktoru je nastavena sériová linka na přenosovou rychlost 57600 bps, 8 datových bitů, bez parity a jeden stop bit. Každý konstruktor třídy *AccessSerialPort* na závěr zavolá metodu **OpenPort()**, která otevře komunikační kanál.

Použitím druhého konstruktoru je uživatel schopen ovlivnit přenosovou rychlost pro komunikaci s měřící jednotkou a třetím, posledním konstruktorem může nastavit jiný počet stop bitů.

9.6.3 Metoda AplySlip(byte[] rawData)



Obr. 30 - Metoda AplySlip()

Pro pochopení algoritmu je pod odstavcem uvedena tabulka 11 s překladem zástupných znaků. Algoritmus funguje tak, že čte data, které dostane jako vstupní argument. Vstupným argumentem je pole bytů. Data čte tak dlouho, dokud nedorazí na pozici předposledního charakteru vstupního pole, protože tím, že algoritmus kontroluje hodnotu následující položky v poli, tak tím, že by toto nebylo ošetřeno, tak by mohlo dojít k přetečení a tím i pádu běhu programu. Dle překladu zástupných znaků pro SLIP protokol již je algoritmus snadný na pochopení. Další popis principu protokolu SLIP je uveden v kapitole 5.1 Komunikační protokol SLIP.

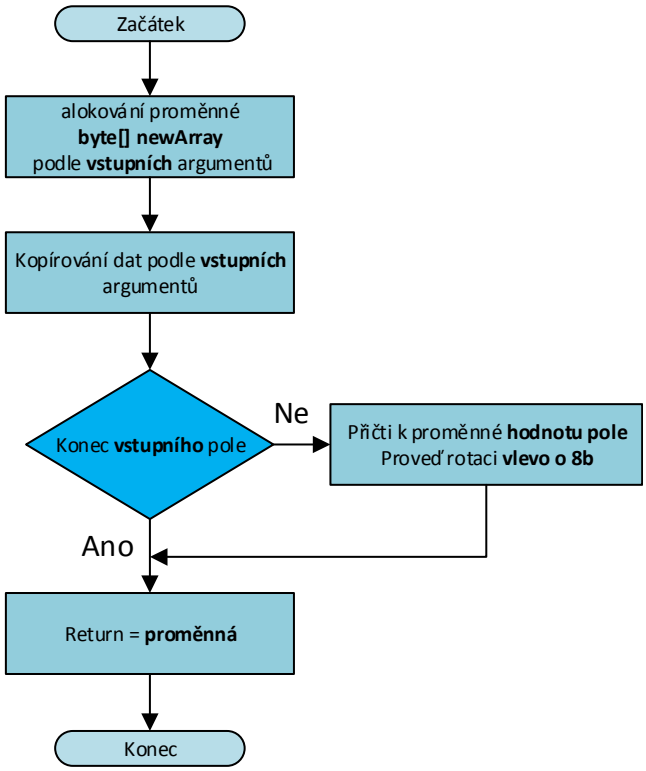
Tab. 11 - Zástupné znaky SLIP

Zástupný znak	Hodnota v HEX	Hodnota v DEC
ESC	DB	219
SYN	C0	192
S_ESC	DD	221
S_SYN	DC	220

9.6.4 Metoda BackupData()

Metoda pracuje na velmi jednoduchém principu, proto zde není uveden její vývojový diagram. Princip metody spočívá, že v prvním kroku vymaže předposlední přijatá data, na jejich místo uloží data poslední přijatá a volá metodu **SortTheAttay()** s parametry posledních a předposledních přijatých dat, která provede jejich odečtení a uloží je do instanční proměnné **tdFrame**. Výše popsaná metoda má privátní modifikátor, tudíž si ji uživatel není schopen mimo zdrojový kód knihovny zavolat.

9.6.5 Metoda ByteToInt(byte[] data, int startIndex, int stopIndex)



Obr. 31 - Metoda ByteToInt()

Metoda z libovolného pole datového typu byte vrací proměnnou datového typu integer. Princip metody spočívá v tom, že vždy od konce pole přebere hodnotu prvku, uloží si ji do proměnné typu integer a jakmile načte další prvek pole, tak dočasné číslo bitově posune vlevo o osm bitů a k nově vzniklé proměnné přičte hodnotu prvku pole.

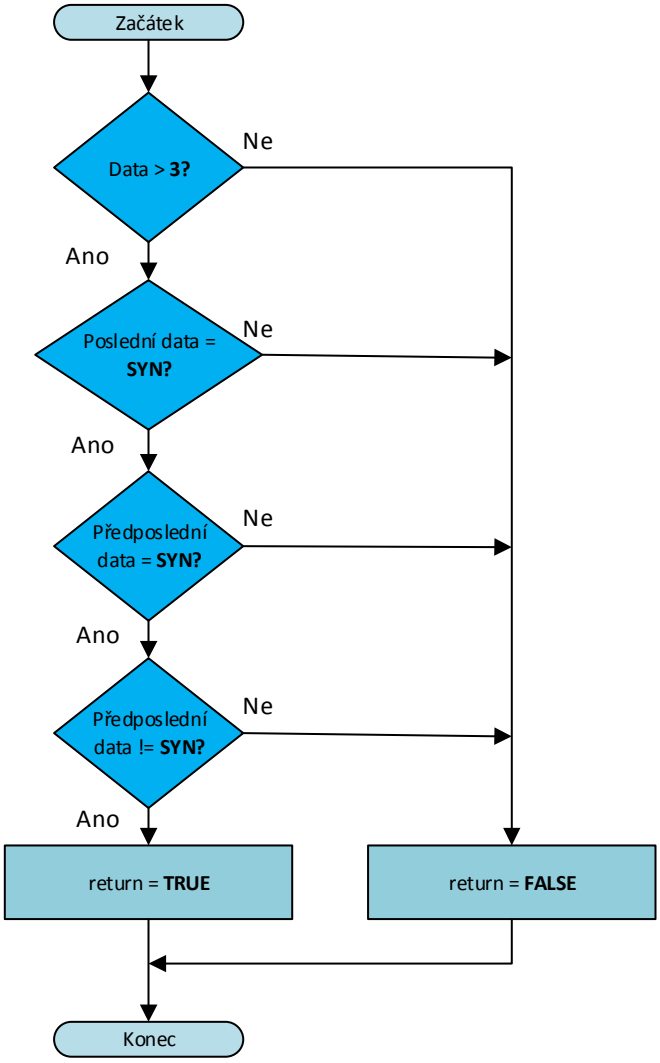
9.6.6 Metoda ClosePort()

Tato metoda uzavře komunikační kanál, mez měřící jednotkou a knihovnou. Nevrací žádnou datovou proměnnou, proto je její vývojový digram vynechán.

9.6.7 Metoda DataInBuffer()

Metoda si zjistí, kolik bajtů je ve vyrovnávací paměti sériového kanálu a tuto hodnotu reprezentuje uživateli. Schopnost zjištění velikosti bajtů v zásobníku je možná, protože využíváme instance třídy System.IO.Ports, kde účelem této třídy je přístup k periferiím počítače.

9.6.8 EndOfFrame(byte[] data)

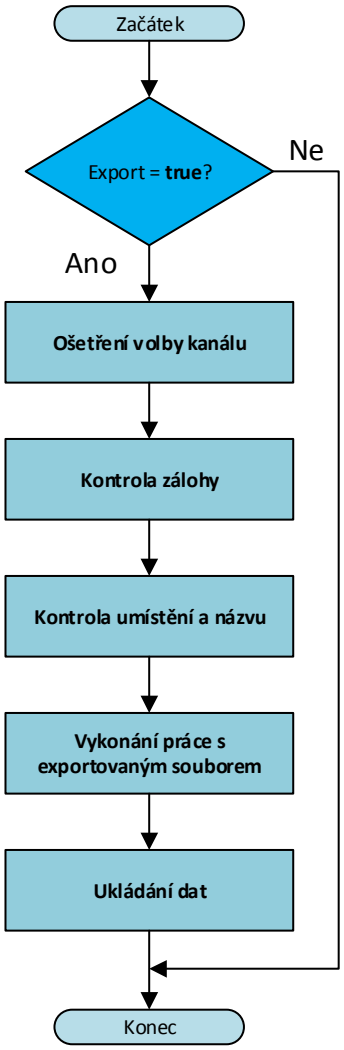


Obr. 32 - Metoda EndOfFrame()

Tato metoda je celkem zásadní pro běh programu. Je aplikována na všechna přijatá data, dokud vstupní parametry nesplňují všechny podmínky, tak vrací logickou hodnotu false. Na metodu jsou navázány další dílčí úkony algoritmu pro zpracování dat.

9.6.9 Metoda ExportToFile()

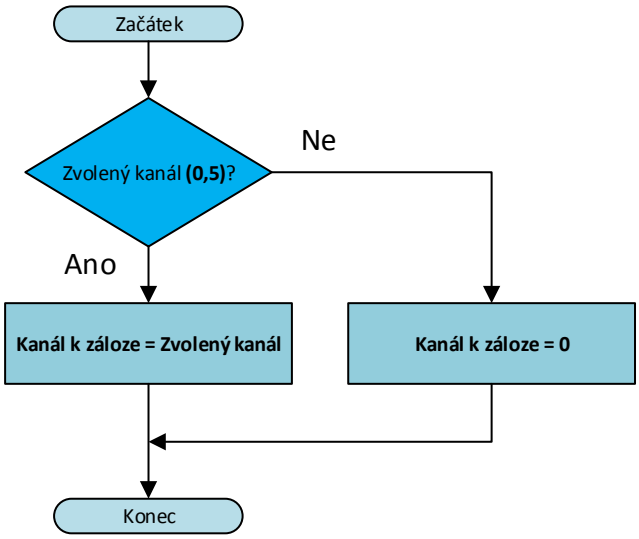
Z důvodu mnoha vstupů od uživatele a tím i nutnosti aplikace musí ošetřit spousty vstupů a obsahovat mnoho rozhodujících podmínek, bylo nutné vývojový diagram metody **ExportToFile()** rozdělit na několik dílčích vývojových diagramů, pro lepší pochopení a snazší orientaci. Principiálně je metoda zobrazena na diagramu níže.



Obr. 33 - Základní vývojový diagram exportu

Během běhu programu metody je prováděno několik kroků a ošetřování vstupů od uživatele. Jelikož bylo snahou potlačit co nejvíce výjimek, musí během použití metody několikrát zavolat rozhodujících funkcí typu if nebo switch.

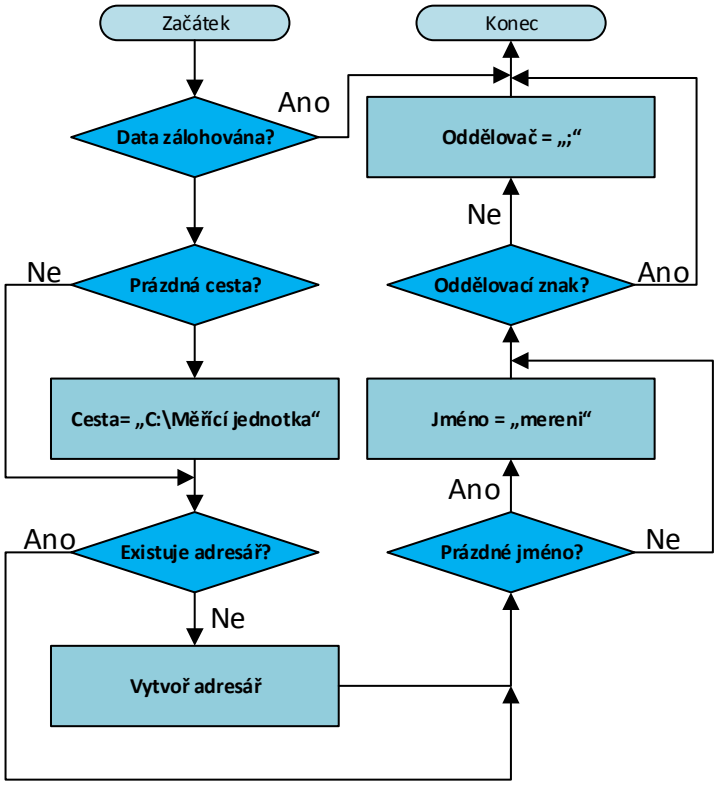
9.6.9.1 Ošetření volby kanálu



Obr. 34 - Ošetření kanálu

Jak se říká, nikdo není neomylný, proto se může stát, že ač má měřící jednotka čtyři kanály, může uživatel zadat kanál mimo rozsah. Pokud se tak stane, jsou automaticky preventivně zálohována všechna data, než aby došlo k jejich ztrátě.

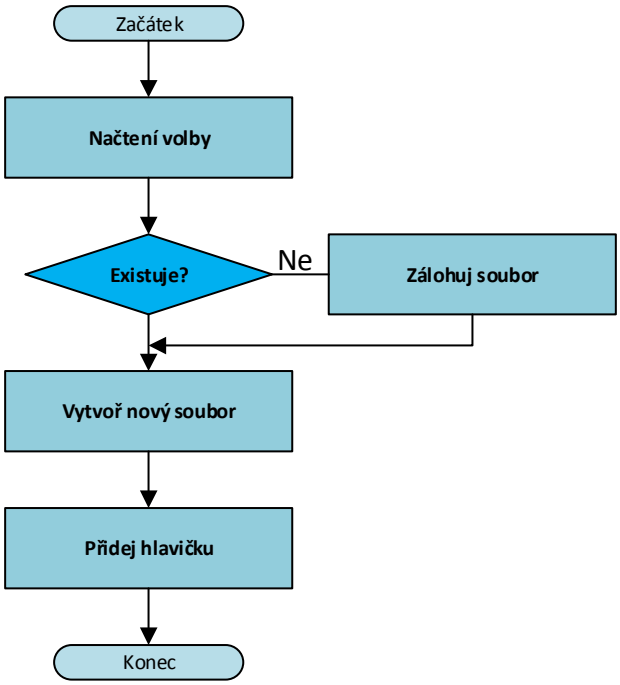
9.6.9.2 Kontrola zálohy, umístění a názvu



Obr. 35 - Kontrola zálohy, umístění a názvu

Uživatel se může vybírat vlastní název a umístění souboru. Tím může dojít například k chybné cestě, nebo uživatel může zapomenout vyplnit název souboru či oddělovací znak. Pokud se tak stane, metoda buď to automaticky vytvoří cestu, kam chtěl uživatel ukládat, nebo kontroluje, jestli již ve zvolené cestě neexistuje soubor, jako který bych chtěl uživatel ukládat. Stane-li se tak, je provedena záloha, která k uvedenému souboru přidá datum a čas vytvoření zálohy. Podobným způsobem oddělujeme i oddělovací znaky.

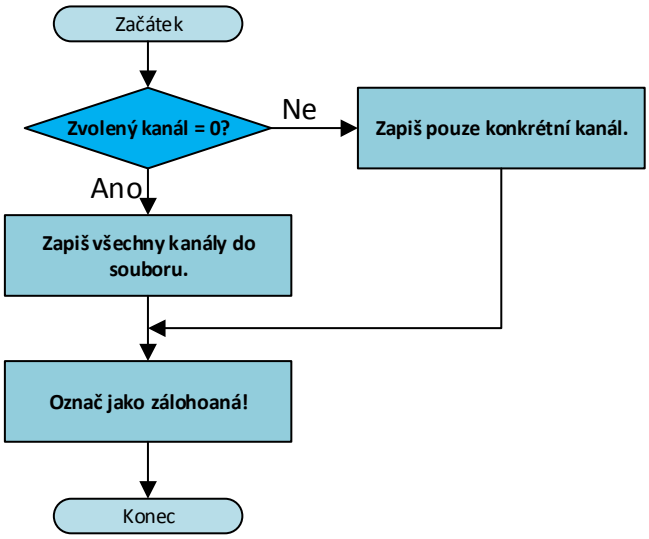
9.6.9.3 Vykonání práce se souborem



Obr. 36 - Práce se souborem

Podle reakce na volbu práce se souborem, metoda kontroluje existenci souboru, z důvodu lepší čitelnosti zpracovaných dat přidává hlavičku, kde první sloupec označuje číslo kanálu, poté čas měření, počet pulsů poslaných inkrementálním snímačem, točivý moment poslaný jednotkou a dobu, za kterou byl vzorek pořízen.

9.6.9.4 Ukládání dat

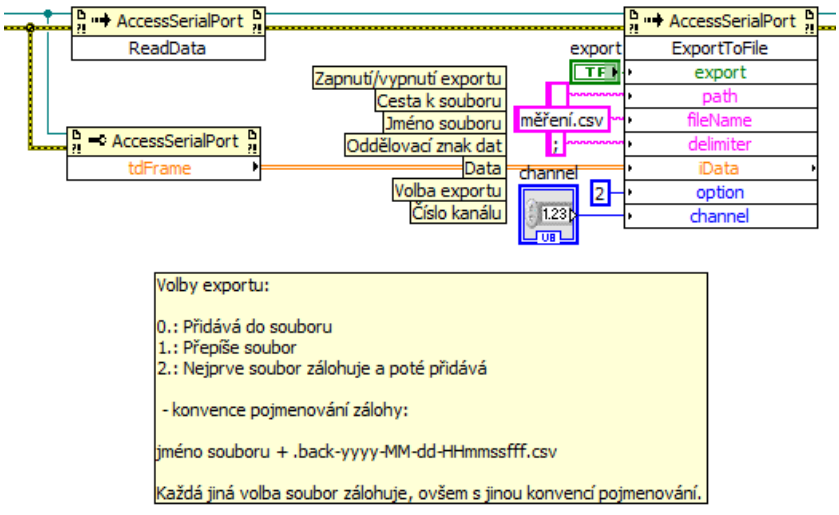


Obr. 37 - Ukládání dat do souboru

Samotné ukládání dat je již velmi jednoduché. Podle zvoleného kanálu metoda přistupuje k vícerozměrnému poli, kde jsou uložena data z aktuálního měření, vybere si, která konkrétní data potřebuje a zapíše je do souboru. Po tomto zápisu označí data jako zálohované, aby při opakovaném volání metody nedocházelo k duplicitě dat exportovaných do souboru.

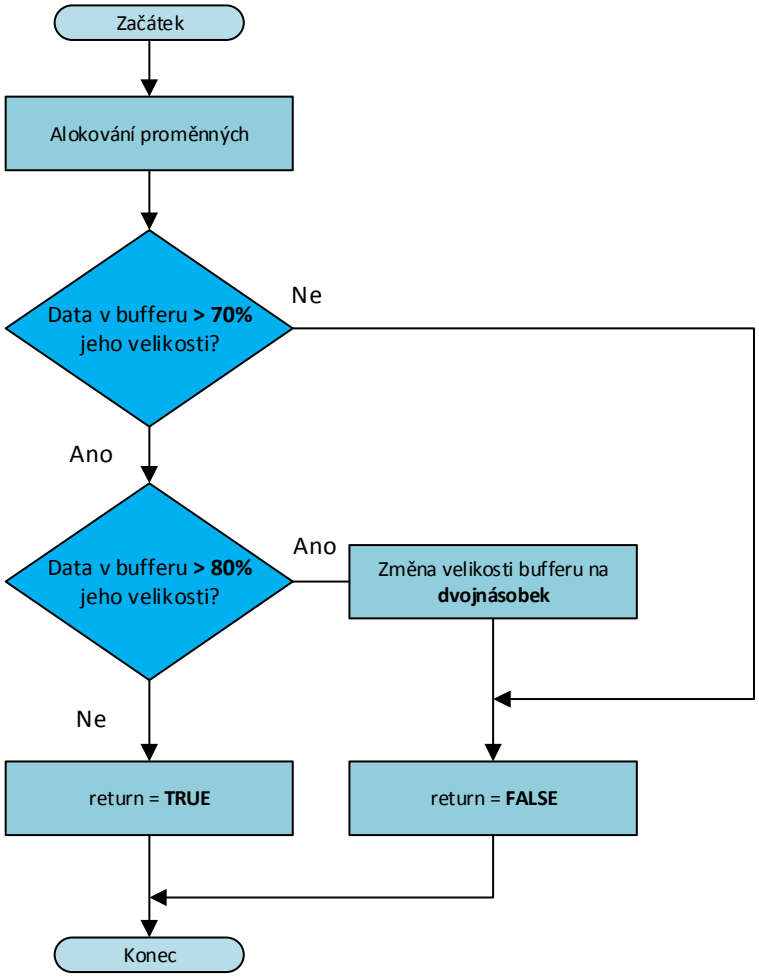
9.6.9.5 Příklad použití v aplikaci LabVIEW

Níže uvedený obrázek naznačuje, nejjednodušší cestu, jak výše popisovanou metodu používat. Příklad použití v LabVIEW je přiložen do příloh.



Obr. 38 - Příklad použití metody ExportToFile() v LabVIEW

9.6.10 CheckBufferSize()

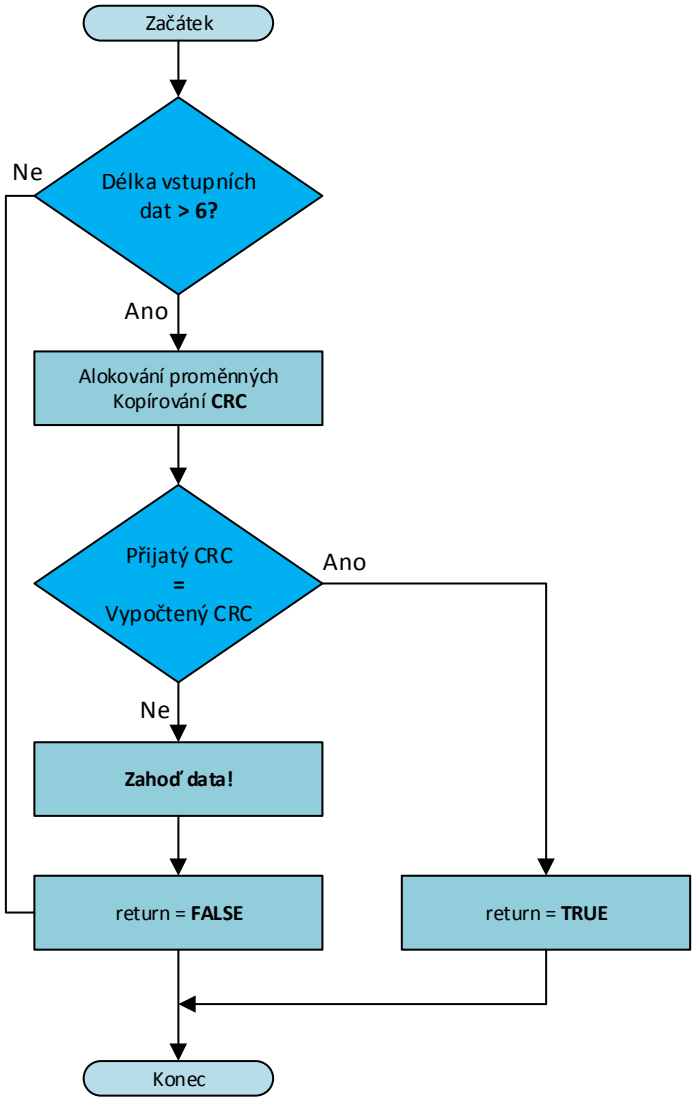


Obr. 39 - Metoda CheckBufferSize()

Metoda slouží ke kontrole příchozích dat, která jsou ukládány ve vyrovnávací paměti sériového kanálu. Při volání metody jsou alokovány dvě proměnné, do jedné se uloží kritická velikost, které se vypočítá tak, že od maximální aktuální možné velikosti bufferu odečteme počet dat, které se v něm momentálně nacházejí a vynásobí se hodnotou 0,2. Porovnáním této hodnoty s aktuální velikostí zásobníku kontrolujeme kritické naplnění nad 80% kapacity.

Totožným způsobem, akorát násobení jinou konstantou kontrolujeme naplnění na 70%. Dosáhne-li velikost bufferu této meze, je preventivně zvětšena kapacita mezipaměti pro ukládání dat na dvojnásobek, aby nedošlo k jejímu přetečení.

9.6.11 CheckCrc(byte[] inputData)



Obr. 40 - Metoda CheckCrc()

Jelikož každý přijatý rámec má stanovenou strukturu, které je uvedena v tabulce 12 pod textem,

Tab. 12 - Struktura datového rámce

Struktura datového rámce							
Pozice:	0	1	2 ÷ n-4	n-3	n-2	n-1	n
Označení:	SYN	SYN	DATA	CRC	CRC	SYN	SYN

kde **n** je celková délka datového rámce. Je celkem snadné najít v rámci bajty nesoucí informaci o cyklickém redundantním součtu. Metoda pracuje tak, že nejprve zkontroluje, jestli velikost datového rámce je delší šesti znaků, aby nedošlo k přetečení výpočtu, poté vybere data z pozice n-3 a n-2, uloží je do pole datového typu byte o dvou prvcích a volá metodu **ByteToInt()** s parametrem tohoto pole. Poté z přijatého rámce, konkrétně prvků pole

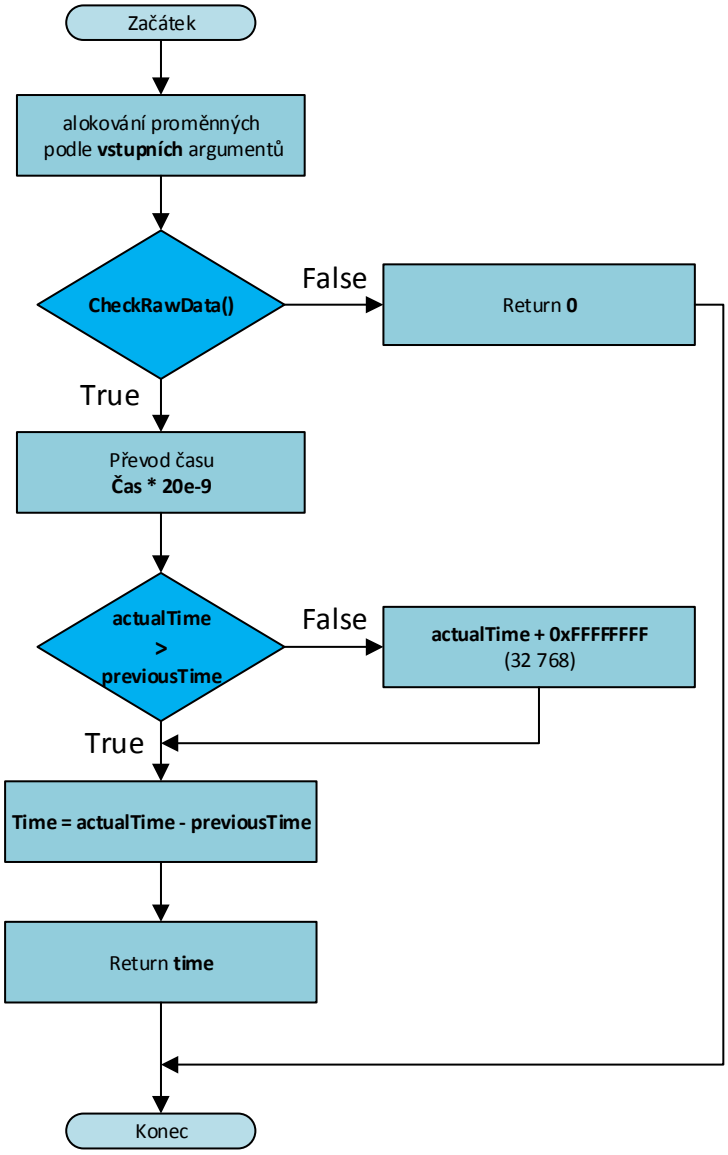
nesoucí samotná data, vypočte kontrolní součet a tuto vypočtenou hodnotu porovná s hodnotu poslanou. Souhlasí-li obě hodnoty, metoda vrací logickou jedničku, v opačném případě vrací logickou nulu.

Jelikož pro výpočet CRC v měřící jednotce je použit samostatný čip, je počet zahozených, datových rámců, které tato metoda vyhodnotí negativně, minimální, ba dokonce mizivý.

9.6.12 CheckRawData()

Princip metody spočívá v tom, že kontroluje surová data tak, aby odpovídala, alespoň délkou, datovému rámci, který musí mít po aplikaci metody protokolu SLIP 16 bajtů. Jelikož může nastat stav, ve kterém během asynchronní komunikace pošle například odpověď na příkaz, tak tato data nemůžeme počítat do datových rámců, které nesou informace z naměřených senzorů. Přijde-li tedy datový rámec kratší jak 16 bajtů, metoda vrací logickou hodnotu false, v opačném případě pak vrací hodnotu true.

9.6.13 CheckTimes(byte[] actual, byte[] previous)



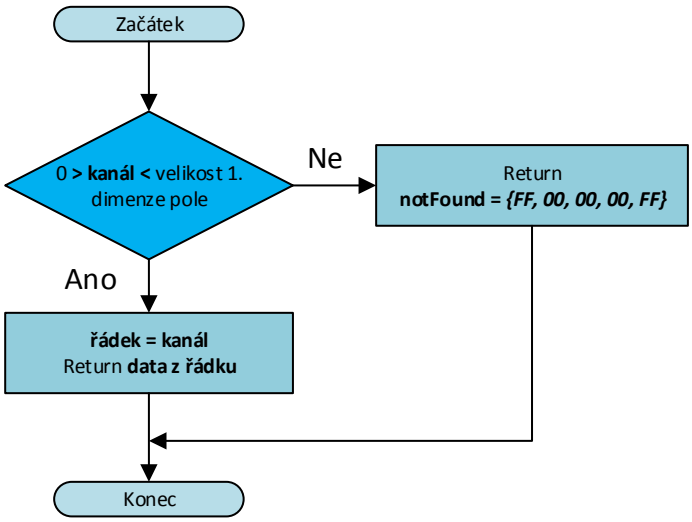
Obr. 41 - Metoda CheckTimes()

Princip metody je postaven, podle kapitoly kde se rozebírá popis prvotního zpracování naměřených dat, takto: časová základna, měřená jednotkou, je reprezentována 32 bitovým číslem, u kterého přesně víme jeho rozsah. Tím, že si modul uchovává informace o aktuálním a předchozím naměřeném údaji a ukládá je do datové proměnné, která má dvojnásobný rozsah. Časová proměnná je tedy ukládána v 64 bitové proměnné, a pokud provedeme porovnáním těchto dvou hodnot, je možné detekovat přetečení časové základny. Pokud je tedy rozhodující podmínka vyhodnocena negativně, jak je zobrazeno na vývojovém diagramu nad kapitolou, dojde k přičtení maximální možné hodnoty 32 bitového čísla k aktuální naměřené hodnotě a až poté dojde k odečtení těchto dvou údajů, ze kterých vypočteme dobu, mezi náběžnými hranami inkrementálního snímače. Tato hodnota je pomocí proměnné ***time***, dále zpracovávána.

9.6.14 IncommingData()

Metoda v reakci na příchozí data, které se nejprve ukládají v zásobníku sériové linky, vrací logickou hodnotu true/false, podle toho, jsou-li v zásobníku data nebo ne.

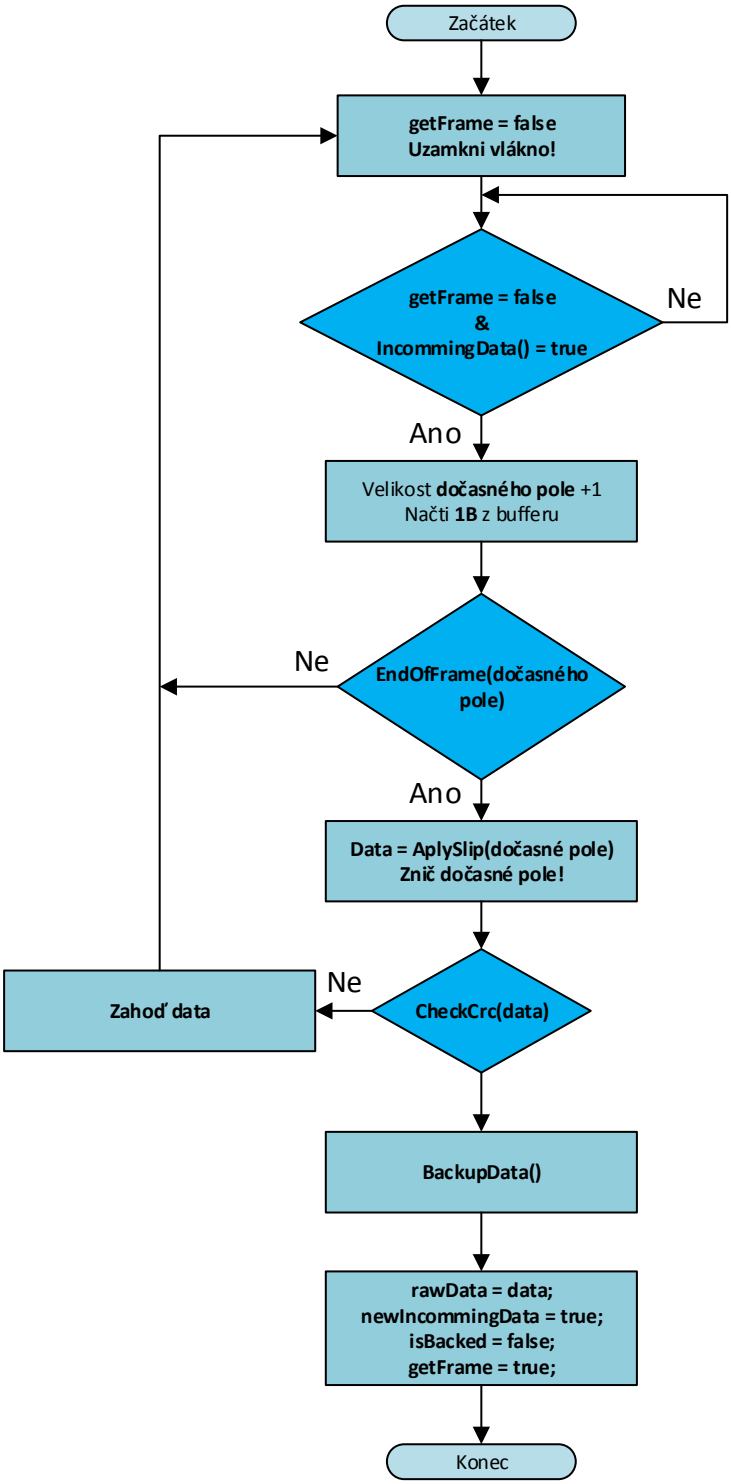
9.6.15 LoadLine(byte line, byte[,] where)



Obr. 42 - Metoda LoadLine()

Měřená data jsou uložena v dvoudimenzionálním poli, kde v první dimenzi je uložena informace o čísle kanálu pořízeného vzorku, a v druhé dimenzi jsou samotná změřená data. Metoda má dva vstupní argumenty, přičemž první argument je číslo kanálu a druhým argumentem je zdroj, odkud se mají data načítat. Číslo kanálu může nabývat hodnot v rozmezí 0 až 3, protože měřící jednotka může v naší hardwarové konfiguraci zpracovávat informace až ze čtyřech kanálů. Voláním metody s prvním parametrem čísla kanálu a druhým odkud vyčítat, jestli chceme měřené hodnoty aktuální nebo předchozí, dojde k navrácení pole datové typu byte, kde jsou informace z požadovaného kanálu. Pokud uživatel zadá číslo kanálu z většího, respektive z menšího rozsahu, je to ošetřující podmínkou negativně vyhodnoceno a uživateli reprezentováno návratovou hodnotou ***notFound***, jak naznačeno vývojovým diagramem metody **LoadLine()**. Pokud dojde k pozitivnímu vyhodnocení ošetřující podmínky, mohou být data načítána pomocí for cyklu, nebo zavoláním příslušné metody třídy *Array*.

9.6.16 RawData()



Obr. 43 - Metoda RawData()

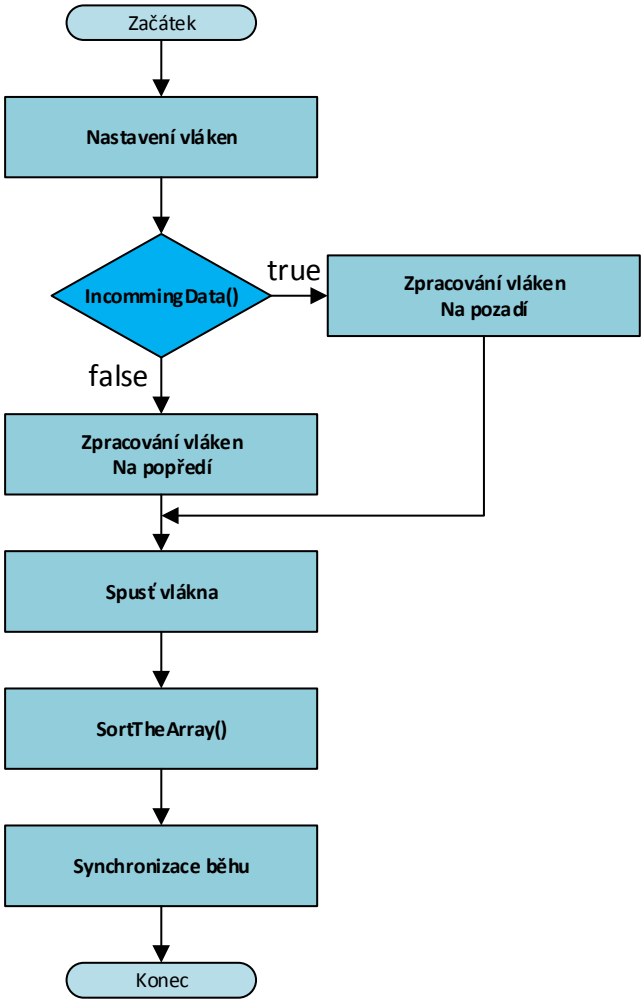
Metoda **RawData()** je jedna z nejdůležitějších metod celého zpracování měřených dat. Protože je toto zpracování prováděno paralelně k ostatním metodám, je nutno prvně uzamknout instanční proměnné, do kterých jsou měřená data ukládána. Při každém volání metody má nejprve booleovská proměnná ***getFrame*** nastavena logickou hodnotu false a

cyklus celého zpracování se vykonává, dokud není splněna počáteční podmínka, složena z metody **IncommingData()** a právě této proměnné.

Princip je založen tak, že bajt po bajtu vyčítáme a ukládáme hodnoty z vyrovnávací paměti sériové linky, kde jsou ukládána data poslané jednotkou, do dočasné proměnné a metodou **EndOfFrame()** kontrolujeme, příznak o konci datového rámce. Je-li nalezen konec datového rámce, dojde použitím metody **AplySlip()** na tento dočasný rámec, korekce poslaných dat, uložení do nové proměnné a zahození dočasného pole. U nově vzniklé proměnné zkontrolujeme příslušnou metodou CRC, a pokud je i toto pozitivně vyhodnoceno, uložíme metodou **BackupData()** do instanční proměnné ***rawData*** tato naměřené údaje v nezpracované formě, nastavíme proměnnou ***getFrame*** na hodnotu true a další, mezi které patří informace, o dispozici nově naměřených dat a zároveň o možnosti jejich uložení do souboru. Na závěr odemkneme vlákny zpracovávané instanční proměnné, abychom k nim mohli přistupovat z metod přístupných uživateli.

Tento princip byl volen z důvodu absence možnosti reakce na události, které simulační software LabVIEW není schopen detekovat a následně zpracovat.

9.6.17 ReadData()



Obr. 44 - Metoda ReadData()

ReadData() metoda je nutná, chceme-li získat ze snímačů připojených k měřící jednotce libovolná naměřená data. Princip metody je postaven tak, že v prvním kroku vyhodnocujeme metodu **IncommingData()** a nastavujeme tak umístění běhu paralelních vláken na pozadí, respektive popředí běhu algoritmu zpracování dat. Nejsou-li k dispozici žádná nová data a nastal by stav, kdy vlivem nadměrného zpomalení běhu cyklu čtení v aplikaci LabVIEW, při kterém dojde k nahromadění dat zasílaných jednotku v zásobníku sériového kanálu, začneme tato data zpracovávat v popředí běhu algoritmu a tím dojde k určitému zrychlení. Vlákna jsou prioritizované v následujícím pořadí od nejvyšší: čtení dat, třízení kanálů a zbytek, jako je například export dat do souboru. Na závěr metody voláme funkci pro synchronizaci běhu, kdy znovu nespustíme v novém vlákně další výčet dat, dokud nejsou předchozí data zpracované.

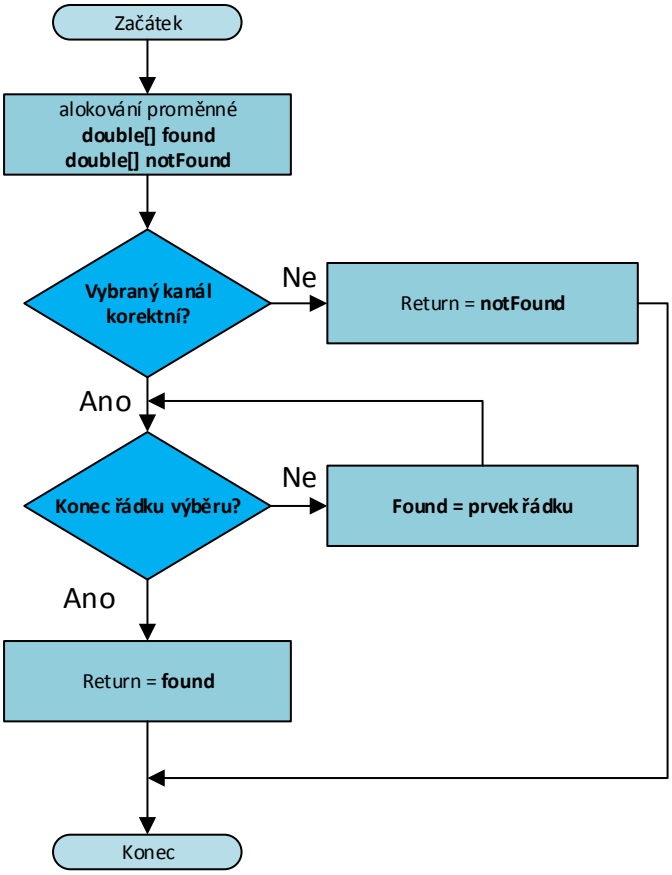
9.6.18 RemoveSynBytes(byte[] array)

Metoda pracuje tak, že ze vstupních dat, u kterých se jedná o pole datového typu byte, odebere ze začátku a z konce **2B**. Tyto konkrétní dva bajty jsou rovny hodnotě **SYN**. Pokud bychom metodu zavolali se špatným vstupní parametrem, nebo z daty, které neodpovídají minimální délce rámec, které je rovna **5B**, metoda vrací $(FF00FF00FF)_{HEX}$.

9.6.19 SelectCrc(byte[] rawArray)

Pokud parametry metody splní počáteční podmínku, které je nastavena tak, že délka vstupních dat musí být rovna délce nejkratšího rámce, který je jednotka možná poslat, provede tyto operace. Cyklický redundantní součet je vždy na n-4 a n-3 pozici, kde n je délka přijatého rámce. Tyto dvě hodnoty uloží do dočasného pole a volá metodu **ByteToInt()** s parametrem tohoto dočasného pole. Metoda převede pole na číslo datového typu integer a vrací jej uživateli. Pokud není splněna počáteční podmínka, metoda vrací nulu.

9.6.20 SelectChannel(int channel)



Obr. 45 - Metoda SelectChannel()

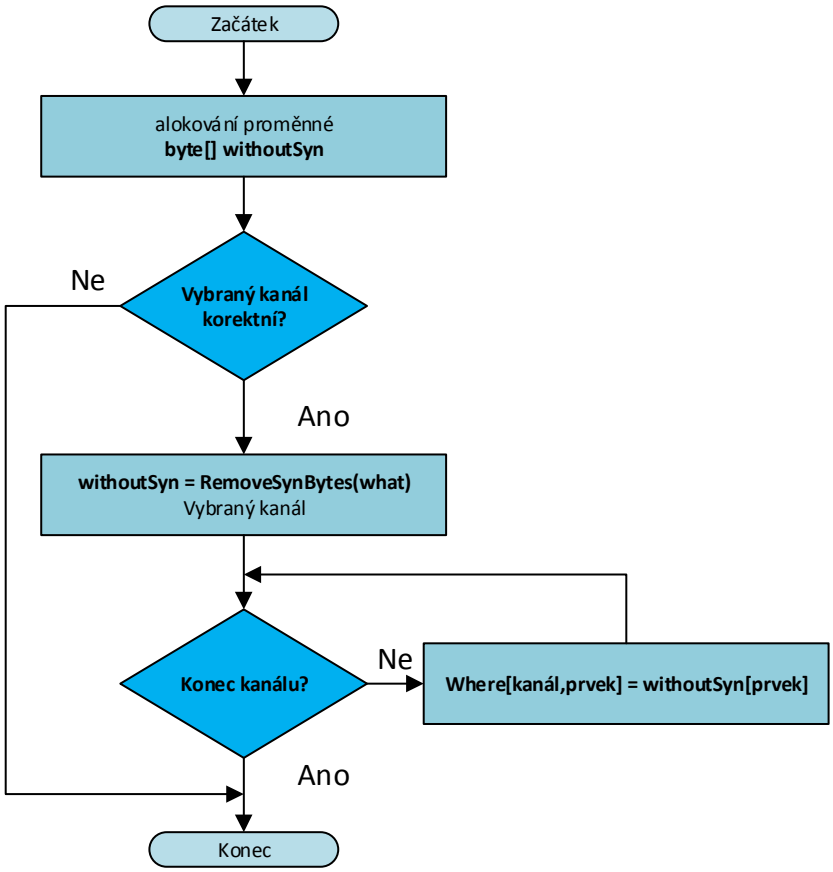
Metoda **SelectChannel()** vrací jako výstup pro uživatel pouze informace z požadovaného kanálu. Pokud by tedy uživatel chtěl pouze informace z druhého kanálu, tak

metoda bude vracet pouze rámce nesoucí informace z konkrétního požadovaného kanálu. Toto slouží pro grafický na obrazovku. Metoda má ošetřen vstupní argument, kterým je číslo kanálu, způsobem, že pokud dojde k volbě kanálu, který ani jednotka není schopna měřit, například kanál číslo 12, tak jelikož tento kanál neexistuje, vrátí uživateli pole pěti prvků, kde první a poslední hodnota prvku pole nabývá hodnotu $(FF)_{HEX}$. Správný kanál je pochopitelně reprezentován uživateli údaji, příslušné konkrétnímu kanálu.

9.6.21 SendCommand(byte[] command)

Metoda **SendCommand()** složí k zápisu dat na sérovou linku a tím i ke komunikaci s měřící jednotkou. Metoda má vstupní argument pole datového typu byte, které vezme a předá sériové lince a tím i měřící jednotce. Délka pole není omezená, pokud by uživatel poslal jednotce jiný příkaz, než ten, který je jednotkou akceptovatelný, měřící zařízení odpoví příslušnou návratovou hodnotu. Možné odpovědi, uvedené v Tab. 6, nalezneme v kapitole rozebírající měřící jednotku.

9.6.22 SortTheArray(byte[] what, byte[,] where)

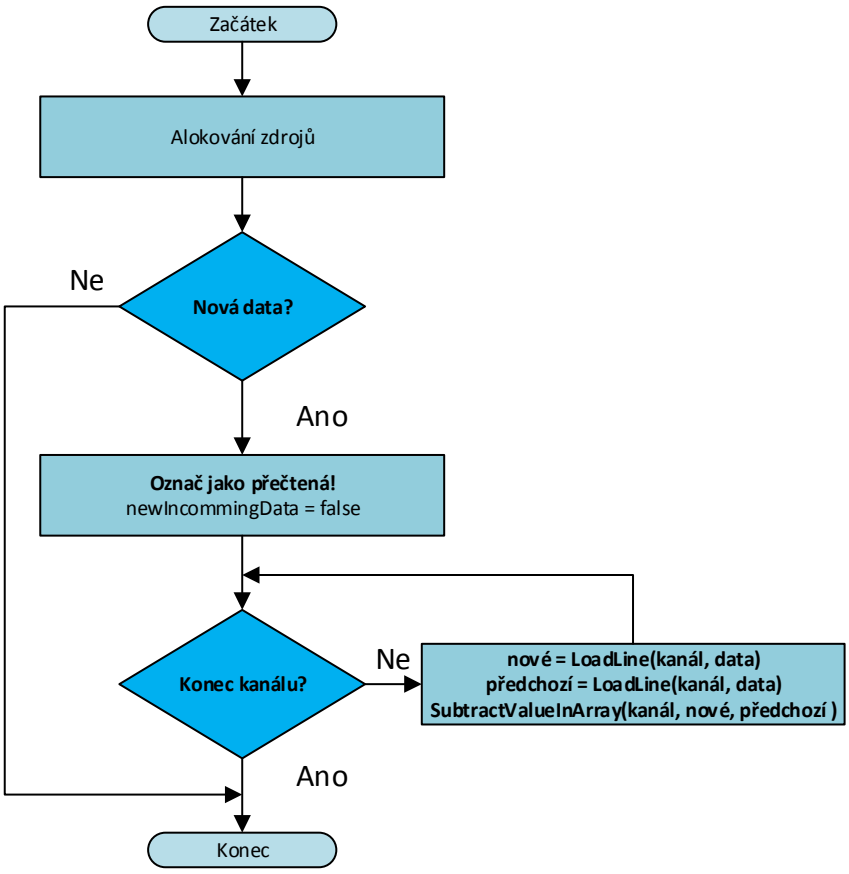


Obr. 46 - Metoda SortTheArray()

Metoda uvedena na Obr. 46 - Metoda **SortTheArray()** , třídí příchozí data. Jelikož má měřící jednotka 4 vstupy a tím i data, která jsou posílána, mají v příznaku označení kanálu

různé hodnoty, je nutné je roztřídit z důvodu pozdějšího zpracování. Kdybychom tento proces neprovedli, mohlo by dojít ke zpracování rámců, nesoucí informace pořízené z různých vstupních kanálu a tím, při jejich pozdějším zpracování, k nepoužitelnosti, protože bychom od sebe odečítali informace pořízené nestejnými datovými zdroji.

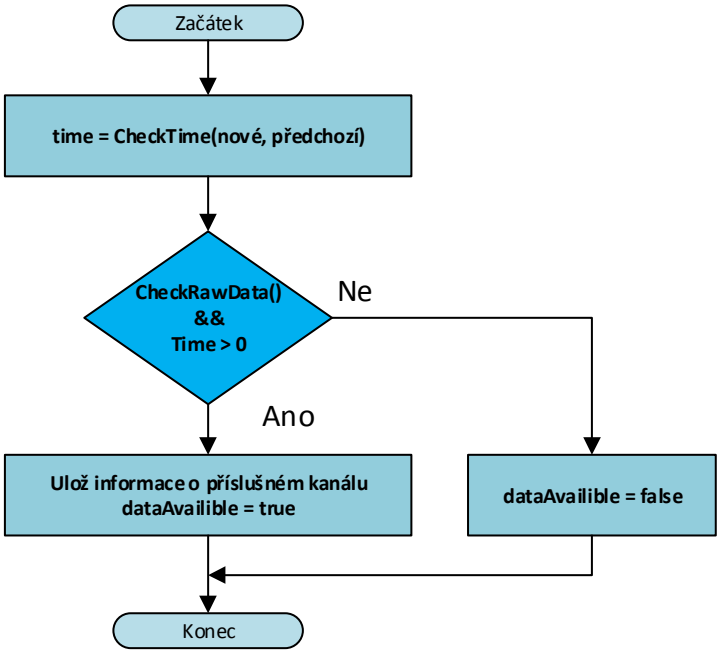
9.6.23 SubtractArray()



Obr. 47 - Metoda SubtractArray()

Metoda, vytvořená za účelem samotné reprezentace naměřených dat uživateli. V prvním kroku dojde a alokování vnitřních proměnných, které metoda používá pro volání funkce, která pak provádí jejich samotné odečtení, a kontrole instanční proměnné nesoucí informaci, jedná-li se o informace nově příchozí, které od sebe doposud nebyli odečteny. Pokud ano, jsou označena jako přečtená a pomocí metody **SubtractValueInArray()** provedeme samotnou diferenci pro všechny kanály současně. Metoda vznikala z důvodu absence možnosti spuštění metody s návratovým typem paralelně k běhu programu.

9.6.24 SubtractValueInArray(byte line, byte[] actual, byte[] previous)

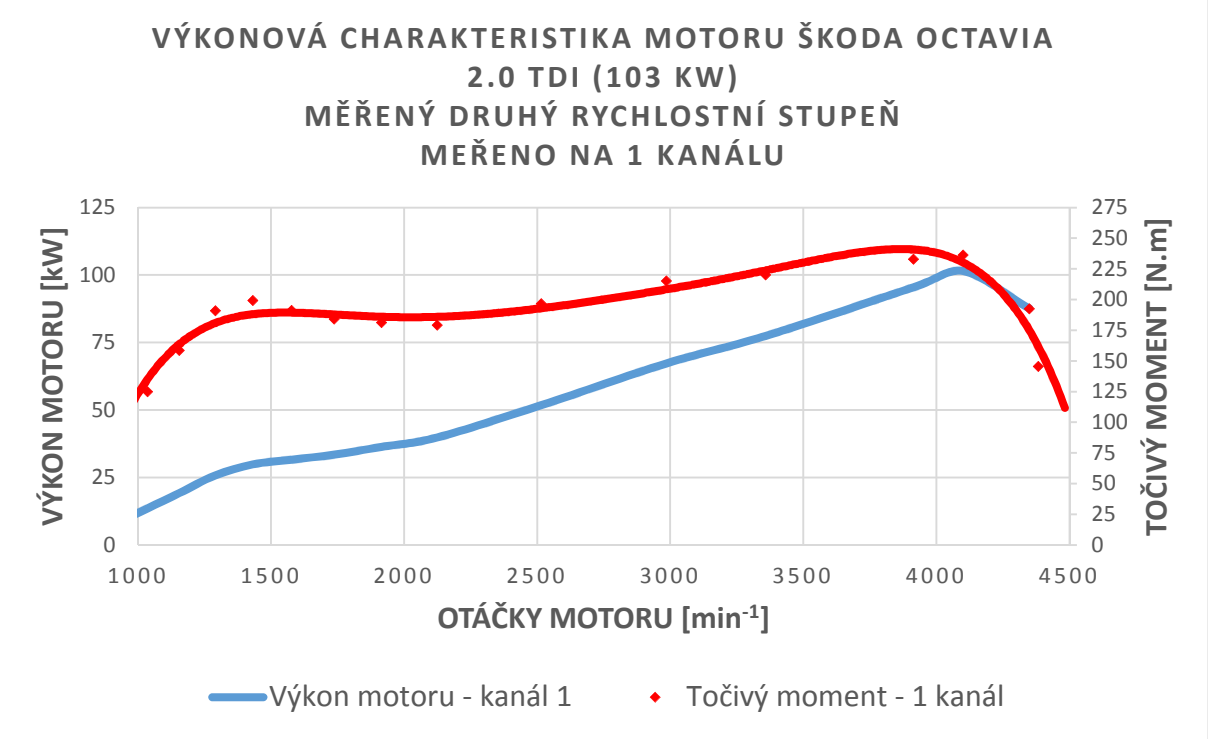


Obr. 48 - Metoda SubtractValueInArray()

Výše uvedený vývojový diagram metody **SubtractValueInArray()** odečítá aktuální a předchozí naměřené hodnoty, které nesou data ze stejných kanálů. Forma prezentace je formou dvourozměrného pole, kde první dimenze pole nese informaci o čísle kanálu a druhá dimenze samotná odečtená data. Uživateli jsou pak data reprezentována jako matice 4x5 prvků, kde každý řádek obsahuje informaci o aktuálním času měření, rozdílu počtu pulzů inkrementálních snímačů, údaje o naměřeném momentu a dobu, kterou trvalo pořízení samotného vzorku. Vyhoví-li data vstupní podmínce, a tím tedy k odečtení naměřených vzorků, metoda informuje uživatele nastavením instanční proměnné na logickou jedničku o dostupnosti změřených dat.

10 Praktické měření

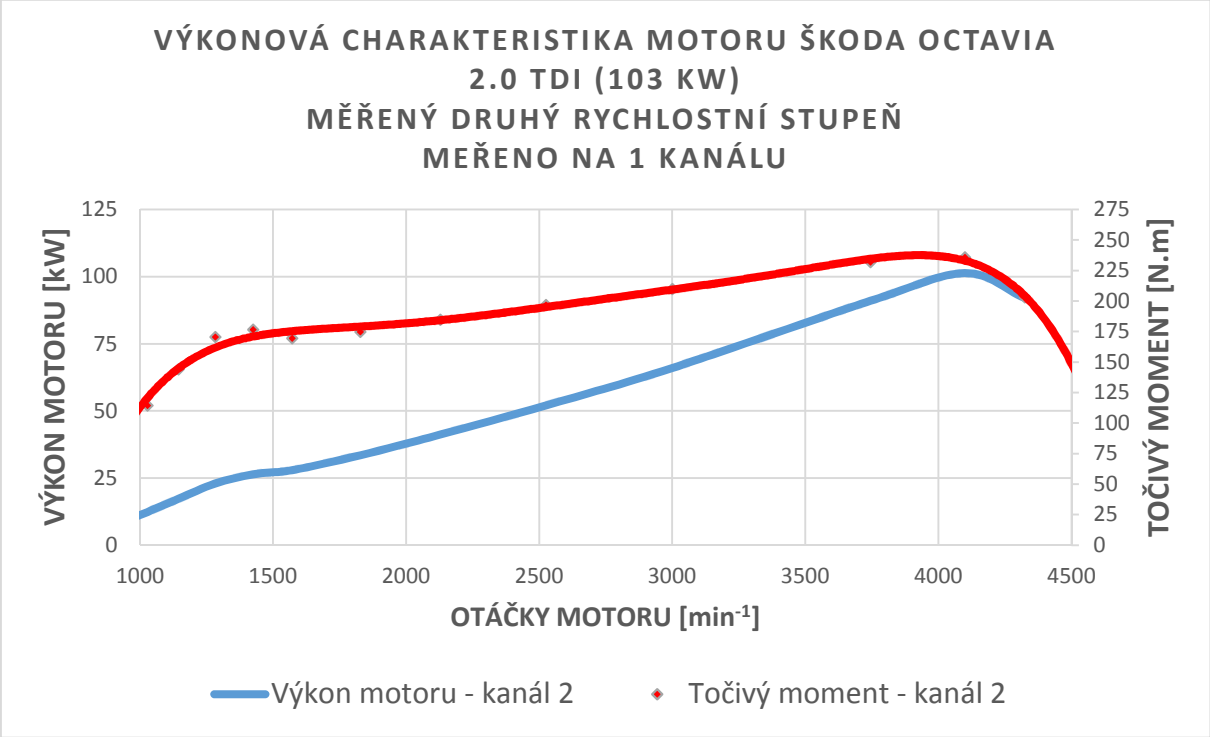
V této kapitole je ukázáno reálné použití knihovny pro měřící jednotku. Na válcové zkušebně Katedry jakosti a spolehlivosti stojů, byla knihovna otestována, zdali je program funkční v reálné aplikaci. K měřící jednotce byly připojeny dva kanály, první a druhý a na každém kanálu byl připojen jeden inkrementální snímač otáček. Válce zkušebny byly roztáčeny motorovým vozidlem značky Škoda Octavia II, z roku 2006 o obsahu a značení motoru 2.0 TDI. Katalogový výkon motoru je 103 kW. Test byl proveden na druhý převodový stupeň, akcelerační metodou.



Graf 1 - Výkonová charakteristika motoru měřená na 1 kanálu

Pomocí naprogramovaného softwaru byla vyexportována data, ze kterých podle vztahů a nabytých poznatků, rozebraných v kapitole 3.7, byl vypočten výkon motorového vozidla. Maximální naměřený výkon byl přibližně kolem 4050 otáček·min^{-1} a samotná hodnota naměřeného výkonu byla 101,5 kW z dat pořízených z prvního kanálu a 101,3 kW z druhého kanálu.

Z naměřených parametrů byl zároveň vypočítán točivý moment, kde jednotlivé hodnoty naměřeného momentu odpovídali maximální hodnotě 237 N·m při 4100 otáčkách·min^{-1}, měřeno na prvním kanálu a 236 N·m při 4050 otáčkách·min^{-1} na druhém kanálu.



Graf 2 - Výkonová charakteristika motoru měřená na 2 kanálu

11 Diskuse.

Cílem této práce bylo navrhnout univerzální komunikační modul, pro komunikaci, obsluhu a zpracování dat poslaných měřící jednotkou, válcové zkušebny vozidel. Výše napsané kapitoly postupně analyzují danou problematiku formou literární rešerše, kapitola s vlastním přínosem se poté věnuje samotné tvorbě softwaru.

Původní koncepce návrhu knihovny pro měřící jednotku byla situována jinak. Autor práce předpokládal, že bude knihovna testována v prostředí Microsoft Visual Studio, ve kterém zároveň modul vznikal a proto byla celá aplikace událostně orientovaná. To znamená, že jakmile jednotka poslala data, knihovna byla schopna tuto událost detekovat a poté provedla následná zpracování a veškeré dílčí úkony, které souvisely se zpracováním datového rámce. Když se však přešlo na testování ve vývojovém prostředí LabVIEW, jehož princip není schopen události detekovat, muselo se přejít k jiné cestě.

LabVIEW funguje tak, že veškeré funkční bločky, které umístíme do block diagramu, zavolá pouze jednou, respektive n-krát, použijeme-li cyklu while. Bez použití cyklu se aplikace spustila pouze jednou a poté, co došla na konec diagramu, se běh programu ukončil a knihovna již nemohla data odesílat, protože skončením programu se zároveň uzavřel komunikační kanál, mezi měřící jednotkou a nadřazeným počítačem. Ani použití cyklického spouštění nebylo řešením, protože stále docházelo k uzavírání komunikačního kanálu. Toto přispělo ke koncepční změně při programování tak, že knihovna pošle data pouze na vyžádání uživatele. S tím se mohl vyskytnout problém s přeplněním bufferu sériové linky. Pokud by obsluha nadřazeného počítače čela data ze sériové linky příliš pomalu, řekněme, že by interval cyklu byl nastaven na 2000 ms, mohlo by se stát, že by měřící jednotka posílala příliš velké množství informací, a tím došlo by k naplnění vyrovnávací paměti do jeho maximální velikosti. Výchozí velikost vyrovnávací paměti je nastavena na 4096 bajtů, není-li uživatelem změněno. Tento problém byl vyřešen metodou **IncommingData()**, která si pokaždé, jak dojde k jejímu zavolání, zjistí aktuální velikost vyrovnávací paměti, určí kolik je aktuálně dat v paměti a jakmile by mohlo dojít ke kritickému naplnění, které bylo definováno na 80% velikosti, dojde ke zvětšení vyrovnávací paměti na dvojnásobek.

Aby došlo k urychlení zpracování informací, je modul naprogramován tak, že zpracování každého kanálu se vykonává v samostatném vlákně. Metoda, která slouží ke čtení dat, vždy detekuje konec datového paketu a poté dalšími dílčími metodami třídí kanály, aplikuje algoritmus na ošetření SLIP protokolu a vždy ukládá aktuální a předchozí přijatý rámec. Tento proces ukládání je zaveden proto, že příchozí data jsou kumulativní. To znamená, že každý nově poslaný datový rámec obsahuje informace o času a počtu impulsu předchozího. Odečtením těchto dvou rámců je tedy možné získat aktuální přesnou informaci o počtu impulzů a době, mezi náběžnými hranami inkrementálního snímače.

Metoda, která musela ošetřit nejvíce vstupů od zadavatele je metoda **ExportToFile().** Metoda samotná funguje tak, že každému nově vzniklému souboru přidá hlavičku, pro lepší orientaci a čitelnost souboru a poté začne data do souboru zapisovat. Budeme-li uvažovat, že obsluha nechtěně zadá jméno souboru, které již existuje, může také naopak zadat soubor, který neexistuje, soubor který sice existuje, ale cesta k němu je špatně zvolená nebo například může zapomenout zadat oddělovací znak a po otevření souboru se bude jednat o směs neidentifikovatelných čísel. Všechny tyto nežádoucí vstupy musí umět metoda detekovat a případně je ošetřit. Příkladem bych uvedl použití metody ošetření zálohy souboru, kde abychom potlačili opětovné volání zálohy a tím museli vytvořit rekurzivní algoritmus detekování, zdali soubor nebyl někdy zálohován pod stejným jménem, došlo k řešení využitím vnitřních hodin operačního systému. Při tvorbě záložního jména souboru si metoda jednoduše načte systémový čas a přidá ho ke jménu zálohy. Tím by mělo být ošetřena jakákoliv ztráta předchozích zálohovaných dat.

Když došlo k samotnému testování, muselo dojít k drobné úpravě zálohování dat. Jelikož při domácím testování byl k dispozici pouze jeden inkrementální snímač, nepočítal jsem, že by jednotka mohla v jednom rámci poslat data z jednoho snímače a v dalším data z druhého snímače. Tím by mohlo dojít při ukládání k duplicitě ukládaných záznamů, kterou bychom museli ručně ošetřit. Byla proto přidána podmínka, která kontroluje, jestli již data z daného kanálu nebyla zálohována. Pokud zálohována byla, přeskakujeme proces zálohy.

Uživateli knihovny jsou zobrazeny veškeré metody, které mají nastaven modifikátor public. Protože každá definovaná třída automaticky dědí členy definované třídou object, které tudíž nejdou skrýt, má každá třída metody, které nepatří mezi autorovu tvorbu. Mezi metody o kterých mluvíme, patří tyto: **ToString()**, **GetType()**, **GetHashCode()** a **Equals()**. Tyto metody jsou implicitně zobrazeny každou třídou a jejich zobrazení nejde potlačit.

12 Závěr

Cílem této práce bylo vytvořit komunikační modul, knihovnu, pomocí které bychom mohli bez bližší znalosti měřící jednotky přistupovat k jejím funkcionalitám. Tomuto vývoji proto byla věnována velká část z ní. S tím souvisí to, že jedním z bodů, které měla práce splnit, bylo to, že veškerý kód musí být řádně okomentován a zdokumentován. Tímto nárokem na práci se kapitoly, které spadají do části vlastního přínosu, staly zároveň technickou dokumentací vytvořeného produktu, kterým je v tomto případě, komunikační modul.

Pro vznik této práce bylo nutné uvést jak autora, tak čtenáře do dané problematiky. Nejprve se autor musel seznámit, jak měřící jednotka funguje, jaké přijímá příkazy, jak uživateli odpovídá a jakých možností je schopna. Tato část je popsána v kapitole, které se přímo věnuje měřící jednotce.

Měřící jednotka slouží primárně k měření výkonu akcelerační metodou. S tím mohla vzniknout další kapitola, která je věnována definici výkonu a točivého momentu jako takového. Abychom mohli nahlédnout do dané problematiky, jsou v jednotlivých podkapitolách části věnující se výkonu motoru nastíněny možnosti samotného měření výkonu, včetně základního rozdělení jakým způsobem lze výkon měřit. Kapitola zároveň rozebere způsob měření výkonu akcelerační metodou, pro kterou byla měřící jednotka primárně zkonstruována.

Když už jsme se dozvěděli, jaké jsou možnosti v měření výkonu, nastala otázka čím tyto veličiny měřit. Protože manuál měřící jednotky přímo popisuje, že vstupy, které měřící jednotka má, jsou určeny k zapojení inkrementálních snímačů otáček a odporových tenzometrů, bylo více než vhodné se seznámit, jak vlastně tyto snímače fungují.

Měřící jednotka komunikuje s nadřazeným počítačem pomocí sériového portu, nebo pomocí emulovaného virtuálního sériového portu. Abychom vůbec mohli přistupovat a programovat tak aplikaci, která je schopna po sériovém rozhraní komunikovat se zařízením na druhé straně, rozebrala kapitola, věnující se sériové lince její vlastnosti a pro běh programu velmi důležitý komunikační protokole SLIP, který když bychom ignorovali a data zpracovávala v surové formě, jak je jednotka pošle, bylo by sice možné data zpracovat, ale získání relevantních validních informací by mělo velkou chybovost.

Jedním z klíčových slov této práce byl cyklický redundantní součet. Tato matematická operace je nutná abychom detekovali, správnost příchozích dat. Protože hardware měřící jednotky má přímo v sobě naprogramován, jaký kontrolním součet má k datům přikládat, bylo nutné se tímto principem výpočtu seznámit a vhodným způsobem jej implementovat i do vznikajícího komunikačního modulu. Principy kontrolních součtů, příklad výpočtu CRC a důvody vzniku rozebrala samostatná kapitola.

Vlastním přínosem této práce byla tvorba softwaru. Softwaru ve formě knihovny byla věnována většina této práce, hlavně pak popisu jednotlivých metod. Této části práce předchází kapitola, které probere alespoň nejnutnější základy programovacího jazyka, které jsou nutné k přiblížení popisu jednotlivých metod.

Posledním bodem cílů práce bylo testování, kterým se zabývala kapitola 100. Z této kapitoly mohlo vzejít to, že naprogramovaná aplikace je schopna komunikovat s měřící jednotkou, zpracovat přijaté rámce z více kanálu zároveň a také exportu přijatých dat do souboru. Z vyexportovaných dat byl sestaven graf, který ukazuje výkon naměřený na druhý převodový stupeň, pomocí měřící jednotky a vyvinutého softwaru. Změřená data byla ve formátu kde na prvním místě je označení měřeného kanálu, času měření, počtu impulsů z inkrementálního snímače otáček, točivý moment a doba mezi náběžnými hranami pulsů inkrementálního snímače. Z těchto hodnot poté bylo možné vypočítat úhlovou rychlost, z časových úseků dopočítat úhlové zrychlení a dosazeném vhodného momentu setrvačnosti byl vypočítán výkon motoru. Z vypočítaného výkonu motoru byl dopočítáván točivý moment. Vypočtené hodnoty zaneseny do grafů 1 a 2.

Veškerý zdrojový kód bude přiložen k práci a je řádně okomentován, navíc při použití této diplomové práce, konkrétně pak kapitol, které rozebírají jednotlivé metody, by neměl být problém modul používat.

13 Citovaná literatura

[1]. **PEJŠA, L.** *Technická diagnostika.* Praha : Česká zemědělská univerzita v Praze, 1995. stránky 45-53. 80-213-0249-6.

[2]. **MANUÁL.** Manuál. *Meřící jednotka pro válcovou zkušebnu vozidel.* Praha : autor neznámý. v 1.1.

[3]. **PAPOUŠEK, MIROSLAV A ŠTĚRBA, PAVEL.** *Diagnostika spalovacích motorů.* Brno : Computer Press, 2007. stránky 37-54. 978-80-251-1697-5.

[4]. **VLK, F.** *Zkoušení a diagnostika motorových vozidel.* Brno : vlk, 2001. stránky 48-53. 80-238-6573.

[5]. **FYZWEB.** Pokusy a materiály: Točivý monent a výkon motoru. *FyzWeb.* [Online] Katedra didaktiky fyziky, MFF UK v Praze. [Citace: 15. 3 2014.] http://fyzweb.cz/materialy/sily/moment/podr.php. 1803-4179.

[6]. **PEXA, M., POŠTA, J. A PETERKA, B.** Moment of inertia measurement of vehicle engine. Praha : Česká zemědělská univerzita v Praze, 2010. stránky 44-47. ISSN 1507-2711

[7]. **HROMÁDKO, J.** *Posuzování jakosti vozidel z hlediska jejich provozní spotřeby paliva a produkce emisí, Disertační práce.* Česká zemedělská univerzita v Praze, Praha : 2007.

[8]. **PEXA, M.** *Možnosti uplatnění dynamických měření pro diagnostice motorových vozidel, Disertační práce.* Česká zemědělsá univerzita v Praze, Praha : 2005.

[9]. **KADLEČEK, B.** Studijní materiály ke cvičení z předmětu diagnostika motorových vozidel. *Teoretické řešení algoritmů měření výkonových parametrů.* Praha : ČZU v Praze, 2004.

[10]. **KREIDL, M. A ŠMÍD, R.** *Technická diagnostika.* Praha : Ben - technická literatura, 2006. stránky 191-192. 80-73000-158-6.

[11]. **FALC, P.** Rotační inkrementální snímače polohy. *Automa.* Snímače a akční členy, řízení pohonů, 2005, 3. [Citace: 21.2.2014] http://www.odbornecasopisy.cz/index.php?id_document=30393

[12]. **RIPKA, P., A DALŠÍ.** *Senzory a převodníky.* Praha : Vydavatelství ČVUT, 2005. stránky 40-50. ISBN 80-01-03123-3

[13]. **TUL.** Snímače otáček a polohy. *Ústav mechatroniky a technické informatiky (MTI).* [Online] 27. 5 2011. [Citace: 17. 3 2014.] http://www.mti.tul.cz/files/svm/Snimace_polohy.pdf.

[14]. **PETRELLA, R., A DALŠÍ.** Speed Measurement Algorithms for Low-Resolution Incremental Encoder Equipped. *Azionamenti Elettrici II.* University of Udine, Faculty of Engineering, 2007. stránky 780-787. ISBN 978-1-4244-0891-7 [Online] Září 2007. [Citace:1.4.2014] http://web.diegm.uniud.it/petrella/Azionamenti%20Elettrici%20II/Tesine/Petrella%20et%20al.%20-%20Speed%20Measurement%20Algorithms%20for%20Low-

Resolution%20Incremental%20Encoder%20Equipped%20Drives_a%20Comparative%20Analysis.pdf.

[15]. **TEXAS INSTRUMENTS.** *Comparing Bus Solutions.* [PDF dokument] Dallas, Texas, USA : Texas Instruments, 10/2009. SLLA067B.

[16]. **VACEK, V.** *Sériová komunikace ve WIN 32.* Praha : BEN, 2003. stránky 6-15. 80-7300-086-5.

[17]. **ROMKEY, J.** The Internet Engineering Task Force. *Nonstandard for transmission of IP datagrams over serial lines: SLIP.* [Online] Červen 1988. [Citace: 15. Březen 2014.] http://www.rfc-editor.org/rfc/pdfrfc/rfc1055.txt.pdf. RFC 1055.

[18]. **LAMMERT, B.** Introduction on CRC calculations. *Lammert Bies.* [Online] [Citace: 20. 12 2014.] http://www.lammertbies.nl/comm/info/crc-calculation.html.

[19]. **NOVÁK, J.** Distribuované systémy a počítačové sítě. *Katedra měření.* [Online] [Citace: 22. 11 2013.] http://measure.feld.cvut.cz/system/files/files/cs/vyuka/predmety/A4B38DSP/uloha9.pdf.

[20]. **TREY, N.** *C# 2010.* Brno : Computer Press, 2010. stránky 73-99. 978-80-251-3034-6.

[21]. **MICROSOFT.** SerialPort – class. *Microsoft Developer Network.* [Online] [Citace: 20. 12 2013.] http://msdn.microsoft.com/enus/library/system.io.ports.serialport%28v=vs.100%29.aspx.

[22]. **MICROSOFT.** Data Types (C# vs. Java). *Microsoft Developer Network.* [Online] Microsoft. [Citace: 1. 3 2014.] http://msdn.microsoft.com/enus/library/ms228360%28v=vs.90%29.aspx.

14 Seznamy použitých zkratek, obrázků, tabulek, symbolů

14.1 Seznam použitých obrázků

14.2 Seznam grafů

14.3 Seznam tabulek

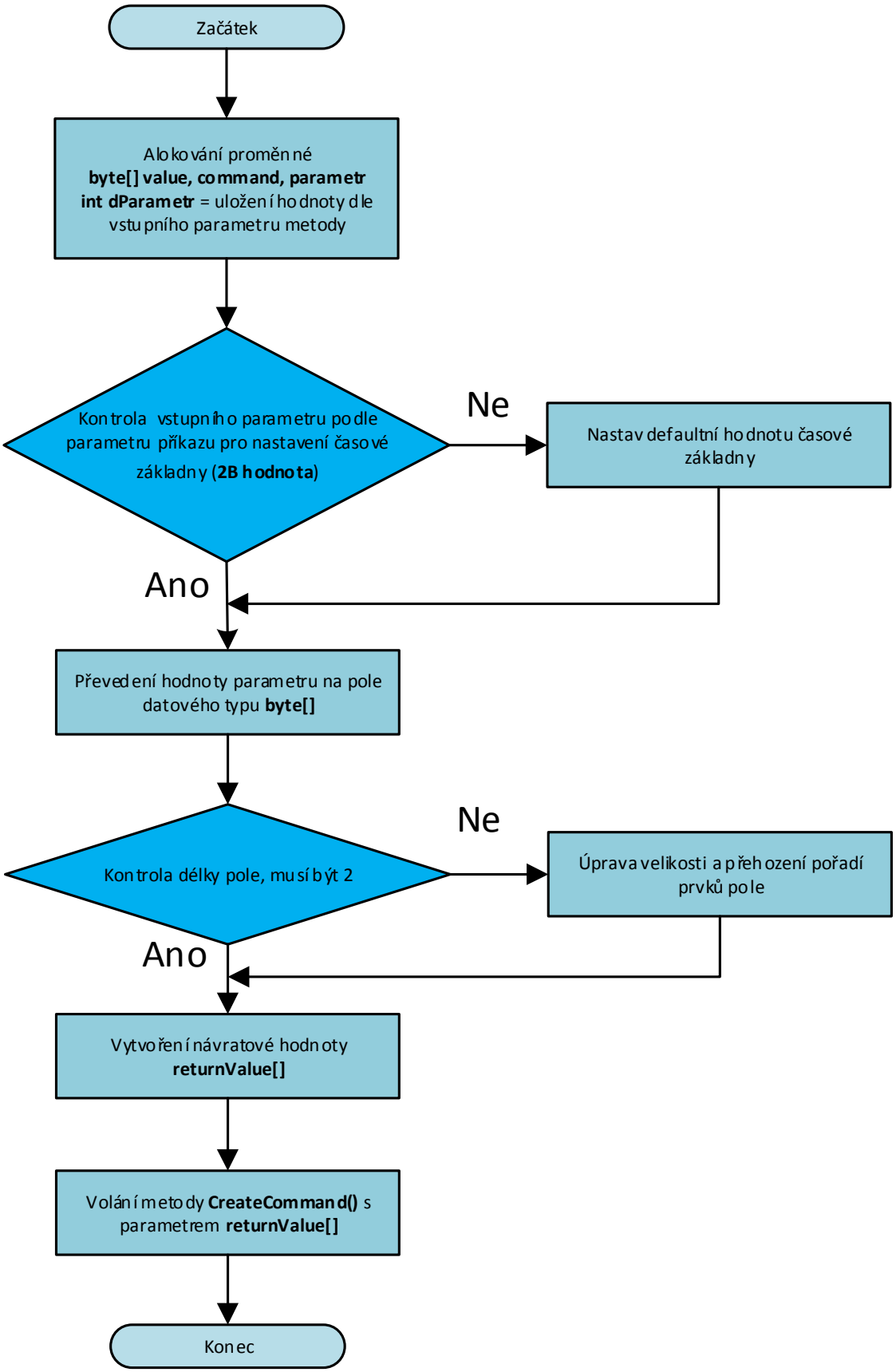
14.4 Seznam vzorců

14.5 Seznam použitých zkratek

ASCII	American Standard Code for Information Interchange - kódová tabulka, která definuje znaky anglické abecedy, a jiné znaky používané v informatice
B	Byte - 8 bitů
b	bit
COM	Komunikační port s rozhraním RS-232.
CRC	Cyklic redundancy check - cyklický redundantní součet
GP	Generální polynom
IEEE	Institute of Electrical and Electronics Engineers - mezinárodní nezisková profesní organizace usilující o vzestup technologie související s elektrotechnikou
Kanál	Sada signálů od jednoho válce zkušebny, která se vyhodnocuje společně. V současnosti se jedná o signál od snímače otáček a signál od snímače točivého momentu
PC	Osobní počítač - zkratka Personal Computer
RS-232	Sériový port
SLIP	Serial Line Interafce Protokol
TCP/IP	Transmission Control Protocol/Internet Protocol - sadu protokolů pro komunikaci v počítačové síti

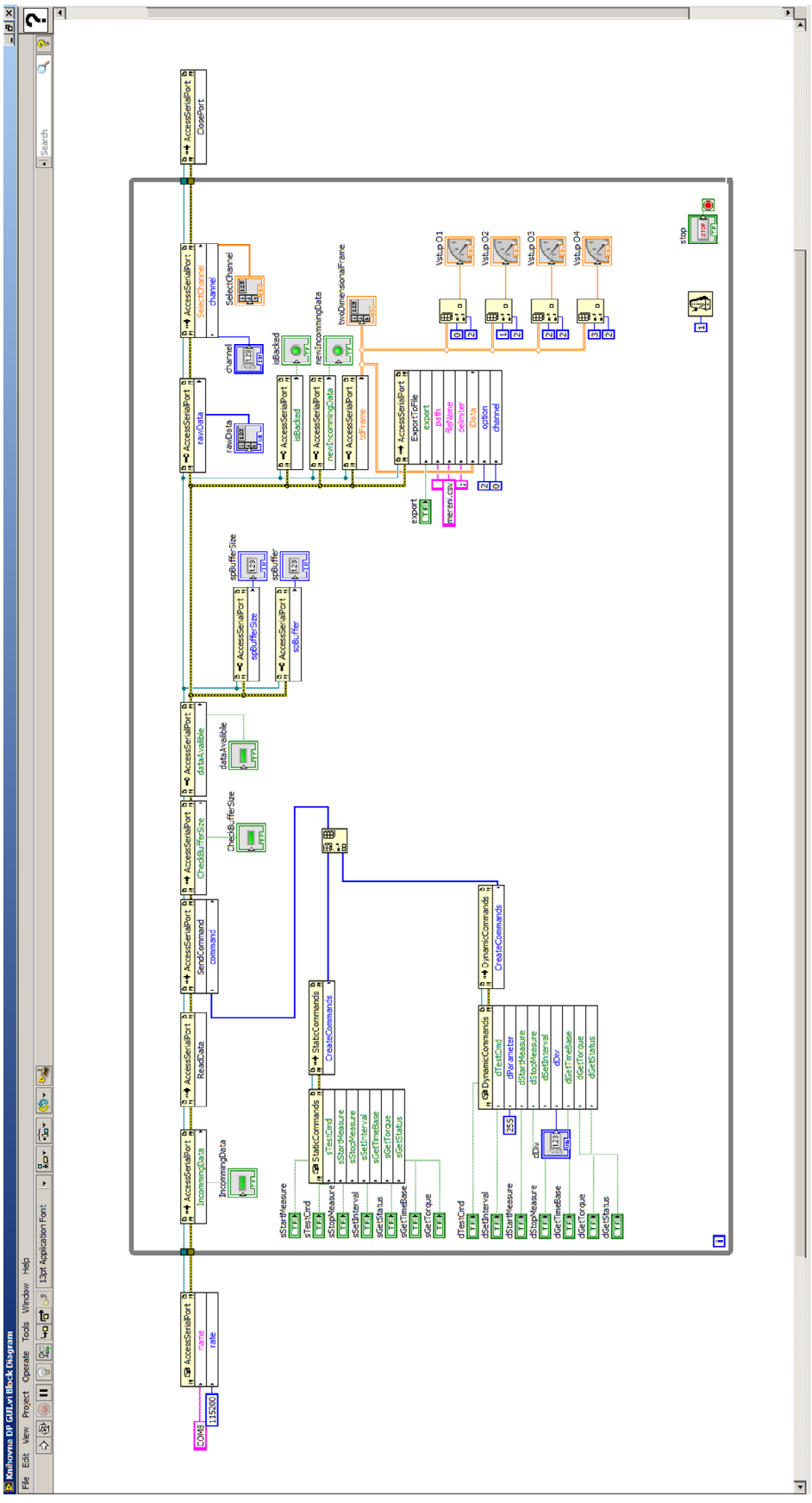
15 Přílohy

Seznam příloh

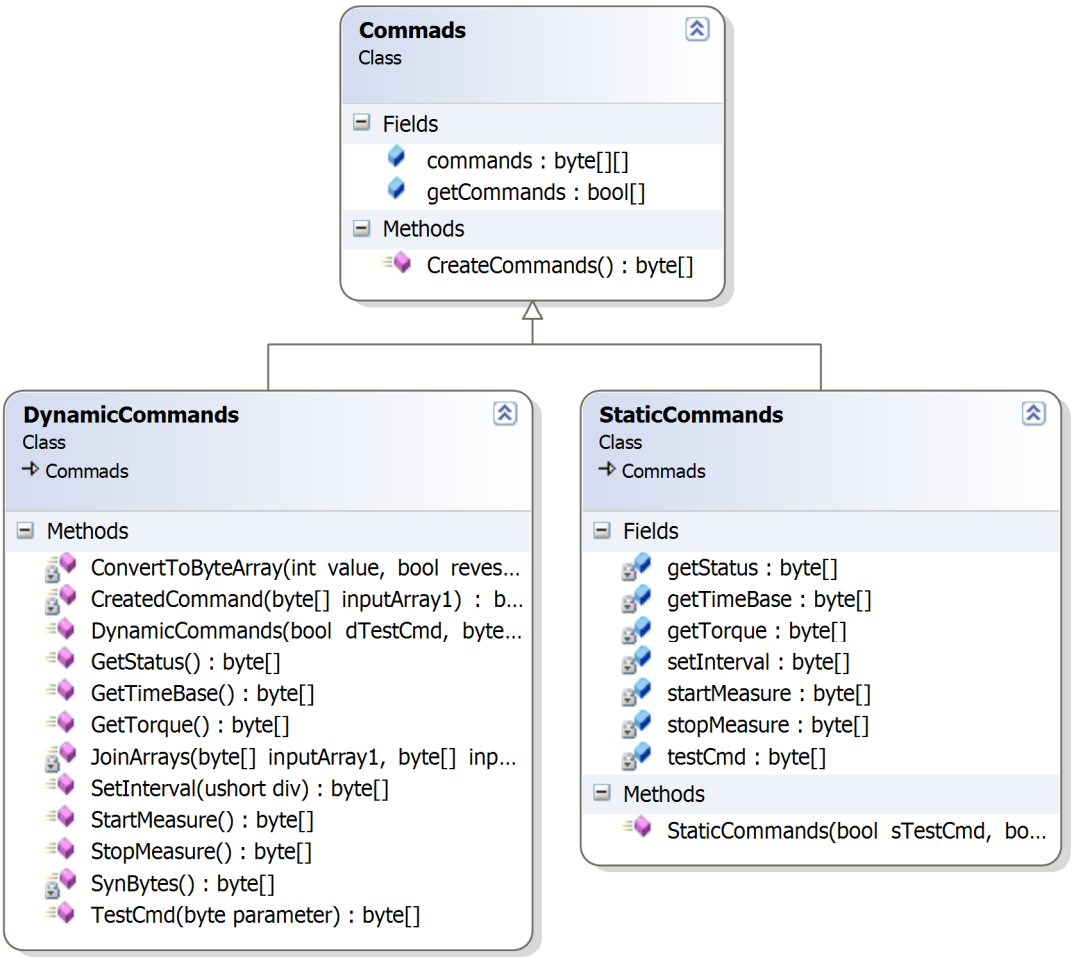


Příloha 1 - Vývojový diagram metody SetInterval()

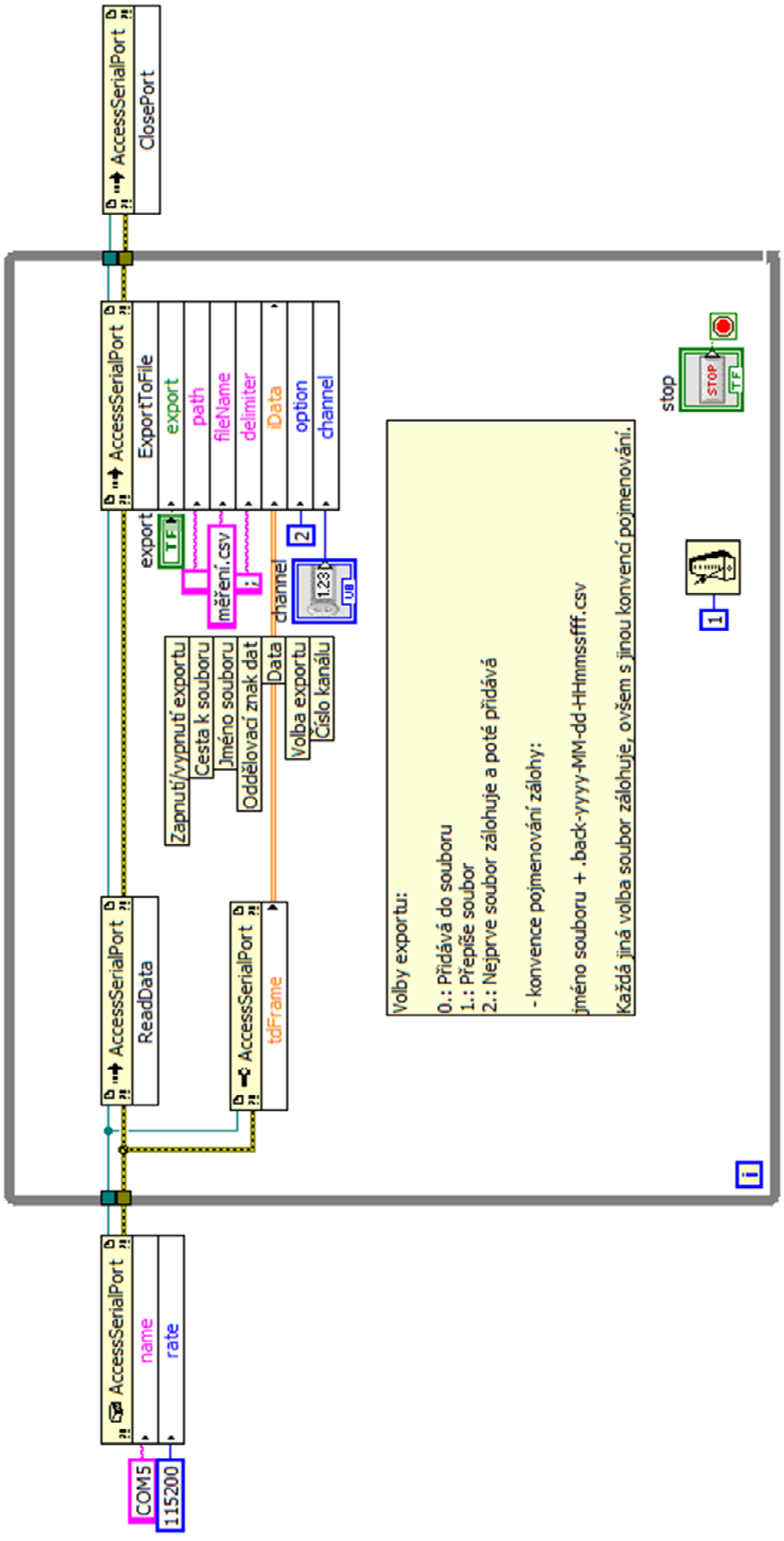
Příloha 2 - Front Panel příkladu použití v LabVIEW



Příloha 3 - Block Diagram příkladu použití knihovny



Příloha 4 - Vazby mezi třídami typu Commands



Příloha 5 - Metoda ExportToFile() v LabVIEW