TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Parametrizace elektrických pohonů Siemens pomocí komunikačního protokolu USS

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 3906T001 – Mechatronika

Autor práce: **Bc. Jiří Ešpandr**
Vedoucí práce: Ing. Martin Diblík, Ph.D.

Liberec 2018

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Program with USS communication protocol for Siemens electric drives comissioning

Master thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 3906T001 – Mechatronics

Author: **Bc. Jiří Ešpandr**
Supervisor: Ing. Martin Diblík, Ph.D.

Liberec 2018

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jméno a příjmení:	**Bc. Jiří Ešpandr**
Název práce:	**Parametrizace elektrických pohonů Siemens pomocí komunikačního protokolu USS**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**
Vedoucí práce:	**Ing. Martin Diblík Ph.D.**
Rozsah práce:	**40—50 stran**
Konzultant:	**Dr. Jan Podrapský, Ph.D. (Siemens s.r.o. Praha)**

Zásady pro vypracování:

1. Prostudujte strukturu a způsob použití komunikačního protokolu USS (Universal Serial Schnittstelle) pro parametrizaci pohonů Siemens dle dostupné dokumentace.
2. Pro zvolený typ frekvenčního měniče Siemens navrhněte program umožňující zobrazovat a měnit hodnoty vybraných parametrů.
3. Ověřte funkcionalitu programu s použitím reálného měniče, k programu vytvořte uživatelskou příručku a nápovědu.

Seznam odborné literatury:

[1] SIEMENS A.G. Universal Serial Interface Protocol USS: Specification. 09.94. 1994. E20125-D0001-S302-A1-7600

[2] Siemens A.G. Sinamics G110 120W - 3kW Seznam parametrů, příručka uživatele. Vydání 04/03. 2004.

[3] BAYER, Jürgen. C# 2005: velká kniha řešení. Přeložil Jiří. KOLÁŘ. Brno: Computer Press, 2007. ISBN 978-80-251-1620-3.

V Liberci dne .....

.....
Ing. Martin Diblík Ph.D.

Prohlášení

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

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

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

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

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Abstrakt

Práce se zabývá problematikou parametrizace frekvenčních měničů prostřednictvím USS (Universal Serial Schnittstelle) protokolu přenášeného sériovým portem. Obsahuje základní úvod do problematiky, popis co je to frekvenční měnič a k čemu slouží. Navazuje kapitola o sériové komunikaci, zavádí se základní pojmy. Poté se práce věnuje USS protokolu, jeho složení, jaké jsou zákonitosti pro jeho užívání a jak správně sestavit telegram pro komunikaci.

Je formulován seznam požadavků na novou aplikaci, která bude sloužit jako alternativa k již existující aplikaci od firmy Siemens. Z těchto požadavků vzniká aplikace, jejíž vývoj je zde popsán. Navazuje popis finální aplikace, všech jejich funkcí a uživatelského rozhraní. Je zde uvedeno i několik zajímavých událostí, s nimiž se autor setkal.

Další kapitola se věnuje samotnému programátorskému kódu a všem klíčovým částem, ze kterých se aplikace skládá. Jsou zde popsány vybrané třídy a vysvětleny funkce důležitých metod a funkcí. Blíže se zde specifikuje členění aplikace a je zde popsán způsob realizace cyklického vyčítání dat a ukládání do souboru. Nakonec je popsána nápověda k tomuto programu, jak vznikla a v jakém prostředí byla zrealizována.

Na závěr jsou shrnuty přínosy této práce z praktického hlediska.

Klíčová slova

USS (Universal Serial Schnittstelle) protokol, Cyklické vyčítání parametrů, Parametrizace, návrh aplikace, práce s daty, frekvenční měnič, komunikace po RS-232, uživatelské rozhraní

Abstract

The thesis deals with frequency inverter parameterization using the serial port transmitted by the USS (Universal Serial Schnittstelle) protocol. It contains a basic introduction to the problem, a description of what a frequency inverter is and what it is used for. The chapter on serial communication follows, introducing the basic concepts. Afterwards, the paper deals with the USS protocol, its composition, the rules for its use, and how to properly compose a telegram for communication.

A list of requirements for a new application is formulated to serve as an alternative to an existing application from Siemens. The application is created from these requirements, which development is described here. It follows the description of the final application, all of its functions and the user interface. Here are some interesting events that the author has met.

The next chapter deals with the programmer code itself and with all the key parts that make up the application. Selected classes are described and parts of important methods and functions explained. Closer here is a specification of the application and describes a method of realizing the cyclic reading of data and storing them in a file. Finally, the help for this program is described, how it was created and in what environment it was implemented.

Finally, the benefits of this work are summarized from a practical point of view.

Keywords

USS (Universal Serial Schnittstelle) Protocol, Cyclic Parameter Reading, Parameterization, Application Design, Data Processing, Frequency Converter, RS-232 Communication, User Interface

Obsah

Seznam obrázků

Tabulky

Zdrojové kódy

Použité zkratky

ADR	adresa
BCC	kontrolní znak kontroly bloku
IND	dílčí index
LGE	délka
PKE	identifikační číslo parametru
PKW	hodnota identifikačního čísla parametru
PWE	hodnota parametru
PZD	procesní data
STX	začátek textu
USS	univerzální sériový protokol
ID	číslo řádku

Úvod

Firma Siemens se zabývá mimo jiné vývojem a výrobou frekvenčních měničů, PLC automatů. Je jednou z největších světových firem v tomto oboru a jejich výrobky dosáhly světového věhlasu. Své produkty distribuuje do celého světa. Tato práce se věnuje především frekvenčním měničům od této firmy a jejich parametrizací.

Pro sériovou komunikaci s frekvenčními měniči a s řídícími jednotkami elektromotorů se v případě firmy Siemens využívá USS protokolu (Universal Serial Schnittstelle), který může být přenášen přes rozhraní RS-232, RS-485 nebo ethernet. USS Protokol vytvořila firma Siemens, jiné firmy tento protokol nepodporují, není tedy možné parametrizovat nebo řídit zařízení od firmy Siemens produktem od konkurenční firmy za použití USS protokolu. Firma Siemens nabízí i PC aplikace, pomocí kterých je možné jejich měniče řídit. Ovšem pro nový měnič Sinamics V20 jež je nástupcem G110 neexistuje funkční a zároveň bezplatná aplikace pro parametrizaci. V důsledku toho vzešel požadavek na tvorbu nezávislé aplikace, jež tyto parametry splní.

Cílem této práce je porozumět způsobu komunikace s vybraným frekvenčním měničem od firmy Siemens AG prostřednictvím USS protokolu přes rozhraní RS-232. Navrhnout aplikaci, která bude schopná komunikace s frekvenčním měničem prostřednictvím USS protokolu. A pomocí níž bude možné zobrazovat a měnit hodnoty vybraných parametrů. Následně má ověřit funkcionalitu navržené aplikace na vybraném frekvenčním měniči.

1 Co je to frekvenční měnič

Na úvod bude vysvětlen pojem frekvenční měnič, k čemu slouží a jaký význam má v této práci. Budou zde také vysvětleny základní pojmy, které se tohoto tématu týkají. Je zde představen a popsán frekvenční měnič vybraný pro tuto práci, způsob komunikace a parametrizace po USS protokolu.

Frekvenční měnič je zařízení sloužící k řízení třífázových elektromotorů. Slouží ke změně vstupní frekvence napájecího napětí motoru. Toho se využívá pro změnu rychlosti otáčení anebo přímo pro řízení výstupního momentu na hřídeli elektromotoru. Změnou vstupní frekvence napájecího napětí lze také docílit zlepšení rozběhových momentů motoru a efektivnějšího chodu při dynamických dějích (především v případě vektorového řízení). Změna otáček motoru se využívá všude tam, kde je požadavek na jiné otáčky motoru, než jsou nominální otáčky dané konstrukcí motoru a síťovou frekvencí napájecího napětí.

1.1 Zvolený frekvenční měnič Sinamics G110

Pro analýzu USS protokolu a následné reálné otestování funkčnosti aplikace byl zvolen frekvenční měnič Siemens Sinamics G110. Jedná se o základní modelovou řadu od tohoto výrobce pro asynchronní motory, s podporou parametrizace prostřednictvím USS protokolu přenášeného v tomto případě rozhraním RS-232. Je napájený z 1f soustavy TN-S. Způsob řízení je skalární, bez zpětné vazby. Příkon měniče jsou 3 kW a maximální výkon motoru připojeného k měniči činí 2 kW. Výhodou je nízká cena a kompaktnost.

Obr. 1 Frekvenční měnič Sinamics G110

Uživatelské rozhraní je realizováno pomocí vyměnitelného uživatelského panelu *Obr. 2,* na němž se nachází tlačítka a dobře čitelný displej. Pomocí něj je možné vyčítat hodnoty parametru a následně je upravovat prostřednictvím tlačítek. Pro potřeby této práce je nutné uživatelský panel vyměnit za panel s rozhraním RS-232 *Obr. 3.* Ten umožní řídit měnič pomocí USS protokolu. Před tím je ale nezbytné ručně nastavit některé parametry fr. měniče tak, aby odpovídaly požadavkům pro řízení prostřednictvím USS protokolu.

Obr. 2 Uživatelský panel frekvenčního měniče Sinamics G110

Pro nastavení „rychlosti“ komunikace po sériové lince se zavádí pojem *Baud rate* jež vyjadřuje počet změn přenosového média (změna z bin 0 na 1) za jednu sekundu, zkratka Bd. V tomto případě si lze zvolit ze standardních hodnot, jež měnič podporuje 1200, 2400, 4800, 9600, 19200, 38400 a 57600 Bd. To je jeden z parametrů, které je nutné nastavit pro správnou komunikaci prostřednictvím sériové linky. Číslo parametru v paměti měniče je P2010.

Obr. 3 Panel pro USS komunikaci s konektorem RS-232

Další parametry, které je potřeba nastavit pro správnou komunikaci prostřednictvím USS protokolu jsou parametry P2012 a P2013. Parametr P2012 nastavuje délku procesních dat zkráceně PZD část sériové linky USS. Parametr P2013 udává délku hodnoty identifikačního čísla parametru zkráceně PKW.

PZD část přenáší řídící slovo a žádanou hodnotu (master → slave), nebo stavové slovo a skutečnou hodnotu prvku (slave → master). V praxi to funguje tak, že master zařízení pošle slave zařízení (v tomto případě frekvenční měnič) řídící slovo (zapnout/vypnout atd.) a požadovanou hodnotu. Na to je mu odpovězeno stavovým slovem (zapnuto/vypnuto atd.) a skutečnou aktuální hodnotou. Délka PZD části může být 0 až 4 wordy. Toto číslo ovlivňuje kolik, 16bitových slov bude mít PZD část. Pro hodnotu P2012 2 budou přenášeny dvě 16bitové části neboli čtyři bajty.

Délka PKW části lze nastavit parametrem P2013. Tento parametr může nabývat hodnot 0, 3, 4 a 127. Hodnoty parametru 0, 3 a 4 udávají, kolik bude 16bitových slov obsahovat PKW část. Pokud je ale parametr P2013 nastaven na hodnotu 127, znamená to, že PKW část bude mít proměnlivou délku. Délka je vždy upravována dle potřeby. Délka telegramu od master → slave může být pro jeden dotaz jiná, než slave → master. Této vlastnosti se dá dobře využít při vyčítání popisu jednotlivých parametrů, nebo pro vyčítání indexovaného parametru. V tom případě je možné, vyčíst všechny hodnoty parametru a jeho indexů v jednom telegramu.

2 Komunikační rozhraní

Tato kapitola bude pojednávat o komunikačních rozhraních a rozhraní fyzické vrstvy, která byla použita v této práci. Nejprve budou vysvětleny základní pojmy jako přenosová média, její typy, sériová a paralelní komunikace poté zde bude podrobněji rozebrán sériový port RS-232, který je pro tuto práci využit a umožňuje přenos USS protokolu. Komunikaci s vybraným zařízením (frekvenční měničem) je možné provádět i prostřednictvím RS-485. Tato komunikace má tu výhodu, že umožňuje mít na jedné sériové lince více *slave* zařízení (v případě *USS protokolu* až 32 měničů), se kterými je možné pomocí změny parametru *Adresa* nezávisle komunikovat/ parametrizovat. Další výhodou je i to, že toto rozhraní umožnuje přenos na vyšší vzdálenost a obecně vyšší přenosovou frekvenci (*Baud rate*) díky diferenciálnímu přenosu dat. Ovšem pro účely této práce je vybráno rozhraní RS-232 na kterém lze komunikaci také ladit a zároveň jím daný měnič disponuje.

Obecně se komunikace dělí na dva základní typy a to na paralelní a sériovou komunikaci. Dále ji lze řadit do pododdílů podle toho, po jakém médiu se daná komunikace přenáší. Lze je rozdělit do těchto skupin přenosových médií: metalické, optické anebo bezdrátové.

Paralelní přenos je přenos informace, kdy se najedou přenese daný kus slova nebo celé slovo. Při tomto přenosu se používá více vodičů, které umožňují převést větší objem dat, než při sériové komunikaci je-li stejná frekvence přenosu (*Baud rate*). Nevýhodou je nižší odolnost proti rušení a nižší přenosové vzdálenosti. Proto je tento přenos v kontextu dnešní doby spíše na ústupu.

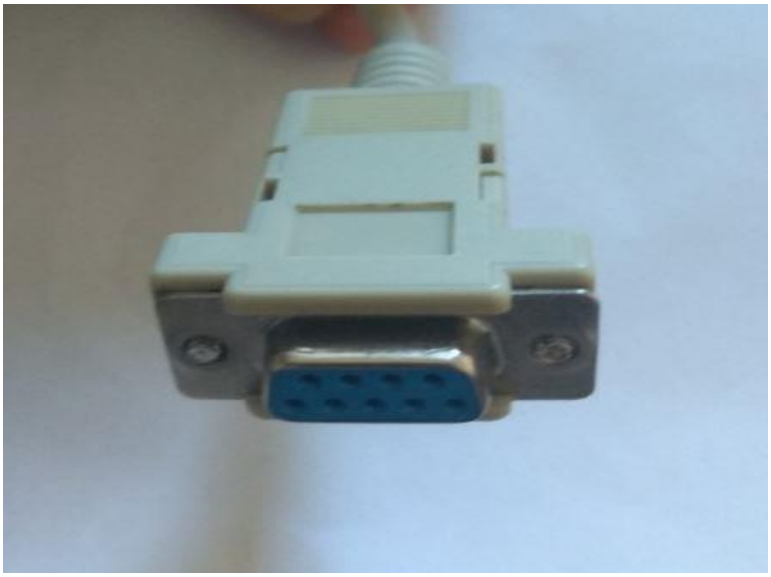
Sériová komunikace oproti tomu je postavena na opačném principu a to na principu, že informace se nepřenáší nejednou ale sekvenčně. Výhodou této komunikace je možnost používat vyšší přenosovou frekvenci a díky tomu dosáhnout vyšších přenosových rychlostí při větších vzdálenostech. Proto umožňuje dosah na větší vzdálenosti než paralelní komunikace. Lze zde lépe ošetřit odolnost proti rušením pomocí diferenčního přenosu informace popřípadě přídavného stínění. Nevýhodnou je nutnost použití převodníků ze sériové komunikace do paralelní podoby z důvodu následného

zpracování dat. V této práci je komunikační protokol přenášen metalickým médiem. Typ komunikace je sériový.

Diferenciální přenos signálu, je takový přenos, při němž se informace přenáší dvěma samostatnými vodiči, na těchto vodičích je stejný signál ale s opačnou polaritou. Přijímač pak nevyhodnocuje napěťový potenciál proti společné zemi, ale rozdíl mezi dvojicí vodičů. Díky tomu má takový přenos vyšší odolnost proti vnějšímu rušení, jelikož se vnější rušení naindukuje na oba vodiče ve stejné míře a jelikož přijímač vyhodnocuje pouze diferenciál mezi těmito vodiči (nikoliv potenciál proti zemi), tak nedojde ke zkomolení přenášené informace. Sériové rozhraní, které tuto technologii využívá je například RS-422, RS-485 a ethernet po kroucené dvojlince.

2.1 Sériový port RS-232

Standard RS-232 též nazývaný sériový port či sériová linka je komunikační rozhraní vytvořené původně pro komunikaci dvou zařízení do vzdálenosti 20 m. Port podporuje přenos v režimu halfduplex a fullduplex, spojení mezi zařízeními je tzv. point to point, je tedy možné propojovat vždy jen dvě zařízení a to jeden přijímač a jeden vysílač (modem a terminál).
Na straně přijímače se signál měří vždy proti společné zemi, nejedná se tedy o diferenciální přenos. Pro tuto práci je výhradně využíván tento typ komunikace.



Obr. 4 Konektor D-Sub DE-9 F

Přenos informací probíhá asynchronně, pomocí pevně nastavené přenosové rychlosti a synchronizace následně probíhá prostřednictvím sestupné hrany uvozovacího znaku. Jako konektor pro tento port se nejčastěji používají 9 pinoví D-Sub DE-9 M (male - samec) nebo F (female - samice). Pro větší odolnost proti rušení je informace po přenosovém médiu přenášena vyšším napětím, než je 5 V.

Logická úroveň	Vysílač	Přijímač
Logická 0	+5 V až +15 V	+3 V až +15 (+25) V
Logická 1	-5 V až -15 V	-3 V až -15 (-25) V
Nedefinováno	---	-3 V až +3 V

Tab. 1: Tabulka logických úrovní RS-232

Z tabulky si lze všimnout, že tento přenos respektive napěťové úrovně jsou symetrické, tzn. rozsah napětí v kladném potenciálu je i v záporném. Přenos informace probíhá od nejméně významného bitu (LSB) po nejvíce významný bit (MSB). Počet datových bitů je volitelný, obvykle se používá 8 bitů, ale lze se také setkat se 7 bity. Pro přenos dat se používají základní tři vodiče a ty jsou: TxD, RxD a GND.

TxD nebo-li Transmision Data je tok dat z modemu (DCE) do terminálu (DTE). RxD nebo-li Receive Data je tok dat z terminálu (DTE) do modemu (DCE). GND je společná zem.

Pro zabezpečení přenosu dat bez nároků na výpočetní výkon se používá prvku parity. Funguje tak, že se ve vysílacím zařízení sečte počet jedničkových bitů a doplní se paritním bitem tak, aby byla zachována předem dohodnutá podmínka.

SUDÁ PARITA – Počet jedničkových bitů + paritní bit = SUDÉ ČÍSLO

LICHÁ PARITA – Počet jedničkových bitů + paritní bit = LICHÉ ČÍSLO

SPACE PARITY – Tzv. nulová parita – paritní bit je vždy v log. 0, používá se například při komunikaci s 7-bitového zařízení s 8-bitovým, kdy paritní bit nahrazuje tvrdou log. 0 poslední bit v byte, tím je zachována kompatibilita s 8-bitovým přenosem.

MARK PARITY - Paritní bit je nastaven tvrdě na log. 1. Při kompenzaci 7-bitového provozu je třeba jej na přijímací straně nulovat, jinak není kompatibilní s ASCII.

Pro uvození a ukončení datového řetězce komunikace mezi vysílačem a přijímačem slouží tzv. START a STOP bity. STOP bit zároveň zajišťuje určitou časovou prodlevu pro přijímač. Právě v době příjmu STOP bitu většina zařízení zpracovává přijatý byte. Pro komunikaci s pomalejšími zařízeními k zajištění doběhu přijaté zprávy se STOP bity zdvojují. Toto řešení se používá při komunikaci na 110 Bd.

V této práci k propojení mezi frekvenčním měničem a PC, na kterém běží program pro komunikaci, byl využit kabel s konektory DF-9 F na obou koncích. A redukce z RS-232 na USB kde na jednom konci je konektor DF-9 M a na druhém USB Type A, který se zapojí do USB portu PC. Nastavena je sudá parita a počet datových bytů je standardních osm.

3 USS protokol a jeho struktura

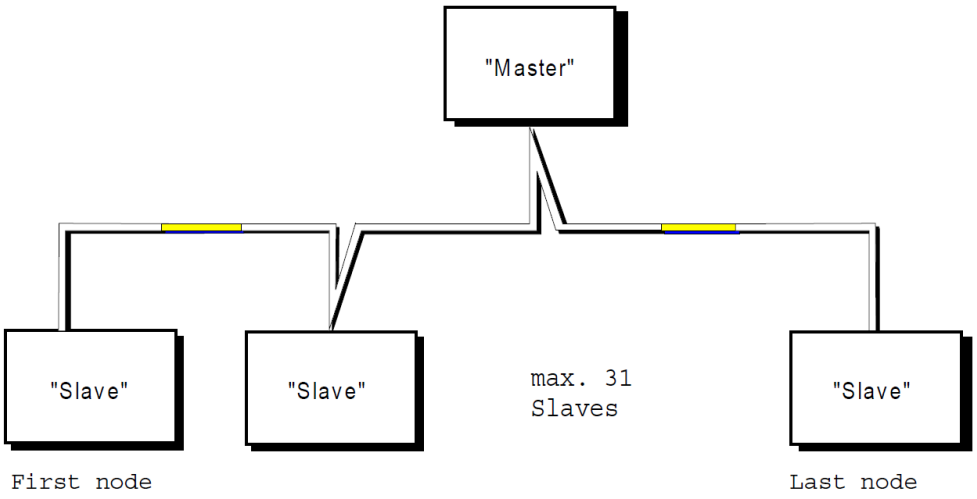
Kapitola pojednává o USS protokolu, který slouží pro přenos informace mezi zařízeními. Na začátek bude přestaveno jeho složení a nastíněno jak daná komunikace probíhá. Následně budou vysvětleny klíčové prvky protokolu a jejich funkce.

Universal Serial Schmittstelle protokol slouží ke komunikaci a parametrizaci vybraného zařízení po sériové lince. Lze pomocí něj vyčítat aktuální hodnoty měniče či jeho stav. Kromě čtení hodnot či parametrů ať jednorázového či cyklického lze i vybrané parametry měnit či jinak modifikovat. Příkladem může být parametr P0305 *„jmenovitý proud motoru“*, který lze měnit v rozmezí minimální hodnoty 0,1 až po maximální hodnotu 10 000, v tomto případě je jednotka [A] ampér. Vybrané parametry lze měnit i za chodu motoru (respektive frekvenčního měniče).

V této práci je pro komunikaci s měničem vybráno rozhraní RS-232. V závislosti na typu provedení měniče lze USS protokol přenášet i po sériovém portu RS-485 u čtyřvodičového zapojení. Díky tomu je možné připojit několik zařízení na jednu sériovou linku, kdy je využita architektura *master*/*slave*. Maximální počet takto připojených zařízení je 31 *slave* a jedno *master* zařízení.

Master je takové zařízení, které řídí komunikaci na sériové lince a dotazuje se *slave* zařízení na jejich stav či další informace a hodnoty. *Master* jednotka inicializuje komunikaci. *Slave* není schopné inicializovat komunikaci ani ji nijak řídit. Čeká, až bude master zařízením dotázáno a v závislosti na dotazu odpoví, popřípadě provede, příslušnou činnost. Pokud si *slave* jednotky chtějí předat informace, musí to udělat prostřednictvím *master* zařízení.

Aby bylo možné jednotlivé zařízení připojené na sériovou linku rozlišit, má každé svoji vlastní adresu. Tak *slave* jednotka pozná, že daná zpráva je určena jemu a má tedy poslouchat. Oproti tomu *master* zařízení pozná, které *slave* zařízení mu právě odpovídá. Zamezí se tak přeslechům atp.



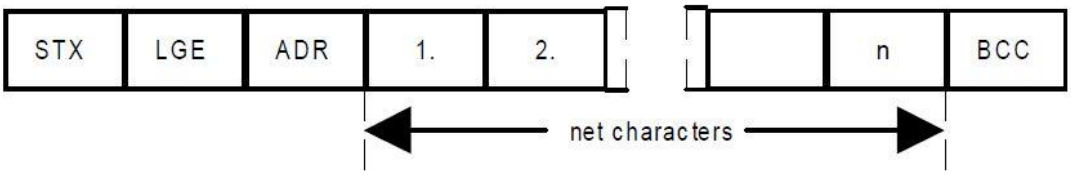
Obr. 5 Topologie zapojení [1]

Přenosovou rychlost telegramu lze měnit v závislosti na typu komunikace a koncovém zařízení. Pro fr. měnič Siemens G110 jsou možné přenosové rychlosti 1200, 2400, 4800, 9600, 19200, 38400 a 57600 Bd. Obecně se ale rozsah pohybuje od 300 do 187 500 Bd.

3.1 Složení USS protokolu

Zpráva USS protokolu se skládá z jednotlivých slov posílaných v předepsaném pořadí a některé i s předepsanou hodnotou či základní strukturou. Naproti tomu jsou zde i části zprávy, které se při určitém typu komunikace nepřenášejí vůbec nebo v omezeném rozsahu. Délka jednoho slova je osm bitů, tedy jeden bajt. Celý telegram se může skládat až z 256 bajtů. Tato délka telegramu může být buď fixní anebo pevně daná v závislosti na druhu a nastavení komunikace.

Na začátku každé zprávy, říkejme telegramu, je vždy uvozovací slovo. To je předepsané a nikdy nesmí být jiné. Pokud ano stává se celý telegram neplatným a je zahozen. Tento uvozovací znak má v hexadecimální reprezentaci podobu 0x02. Po tomto uvození následuje vždy délka zprávy. Ta reprezentuje počet bajtůkterá zpráva obsahuje. Tato hodnota může být v rozmezí 1 až 254. Jelikož se první dvě slova v telegramu (tj. uvozovací znak a délka zprávy) nepočítají do celkové délky, je skutečná celková délka vždy o dva vyšší.



Obr. 6 USS telegram [1]

Dalším nezbytným prvkem telegramu je adresa. Tento prvek slouží pro identifikaci zařízení pro příjem telegramu (směr komunikace *master → slave)* či pro identifikaci *slave* zařízení při opačném směru komunikace. Nicméně tento prvek nepřenáší jen informaci o adrese ale i další specifické informace uložené v nejvyšších třech bitech této proměnné.

Nejvyšší bit nese informaci o tom, zda je přijímaný telegram standardní, či se jedná o tzv. *„Special telegram“.* Pokud je tento bit nastaven na hodnotu jedna, tak se jedná o speciální telegram a ostatní slave zařízení, která nejsou nastavena na příjem speciálního telegramu, tento telegram odmítnou. Toho se využívá pro komunikaci s odlišnou strukturou dat, než je definována v USS protokolu.

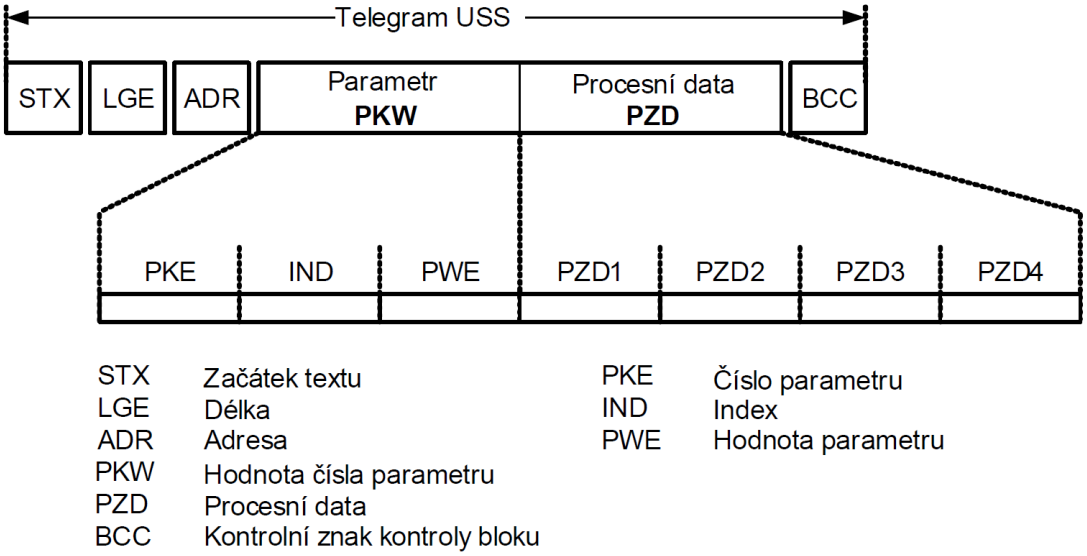
Druhý nejvyšší bit nese informaci o tom, zda je telegram tzv. *„Mirror“* pro hodnotu jedna, či nikoliv pro hodnotu nula. Tento zrcadlový telegram může *master* zařízení vyžadovat po *slave* zařízení a na to *slave* reaguje tak, že zrcadlí příchozí telegram bez toho, aby v něm provedl jakékoliv změny. Toho lze využít při odstraňování problémů na sériové lince, nebo při detekci vadného *slave* zařízení.

Třetí nejvyšší bit v sobě skrývá informaci, zda je daný telegram tzv. *broadcastový* pro hodnotu bitu jedna. Tedy že ho přijímají všechny *slave* zařízení bez výjimky. V tomto případě se pak bity 0 až 4 ignorují. V opačném případě nejsou tyto byty ignorovány a je v nich uložena informace o adrese, ta je v rozmezí 0 až 31.

Poté následují nepovinné části *„Hodnota čísla parametru“* (*PKW*), kde se přenášejí čísla parametrů a jejich hodnoty, a *„Procesní data“* (*PZD*), kde je aktuální stav měniče. Ale o těch až v následujících kapitolách, kde budou podrobněji představeny.

Celý telegram je ukončen tzv. kontrolním součtem nebo-li *BCC (Block Check Character).* Ten, jak už název napovídá, slouží pro kontrolu správnosti přijatého

telegramu. Kontrolní součet vznikne tak, že se provede funkce *XOR* na všechny bajty telegramu. Výsledný bajt je hledaným kontrolním součtem. Jelikož při přenosu telegramu může dojít ke zkomolení, slouží tento mechanismus jako základní kontrola správnosti přijatého telegramu. Po přijetí musí souhlasit *BCC* v přijatém telegramu s vypočteným *BCC* z přijetého telegramu bez *BCC*, pokud tomu tak není, je telegram zahozen.

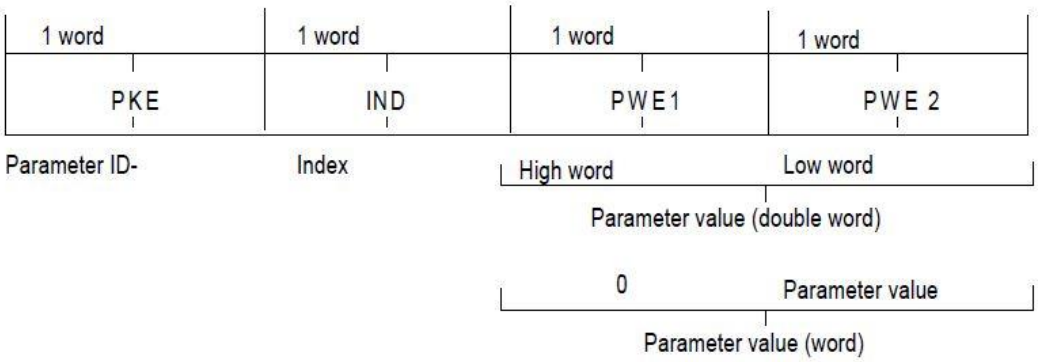


Obr. 7 Složení USS telegramu s popisem [2]

3.2 Parameter ID value

Nebo-li *hodnota čísla parametru* (zkráceně PKW) je nepovinná část telegramu sloužící k vyčítání a nastavení hodnot parametrů a indexů parametrů. Toto vyčítání probíhá formou dotazu na stav či hodnotu daného parametru, jehož číslo je přenášeno v těle PKW části.

Postup je takový, že se *master* zařízení prostřednictvím USS protokolu respektive PKW části dotáže na hodnotu parametru *X*. Odpovědí mu je zpráva, kde se v PKW části objevuje číslo dotazovaného parametru a jeho stav, popřípadě pokud má parametr indexy, tak stav dotazovaného indexu parametru.



Obr. 8 Složení PKW části telegramu [1]

U indexovaných parametrů se není možné dotazovat na stavy všech indexů najednou a je nutné stavy jednotlivých indexů vyčítat postupně. Jelikož se jedná o nepovinnou část tak tato část může být buď nulová anebo nepřítomná vůbec (v závislosti na nastaveném typu komunikace).

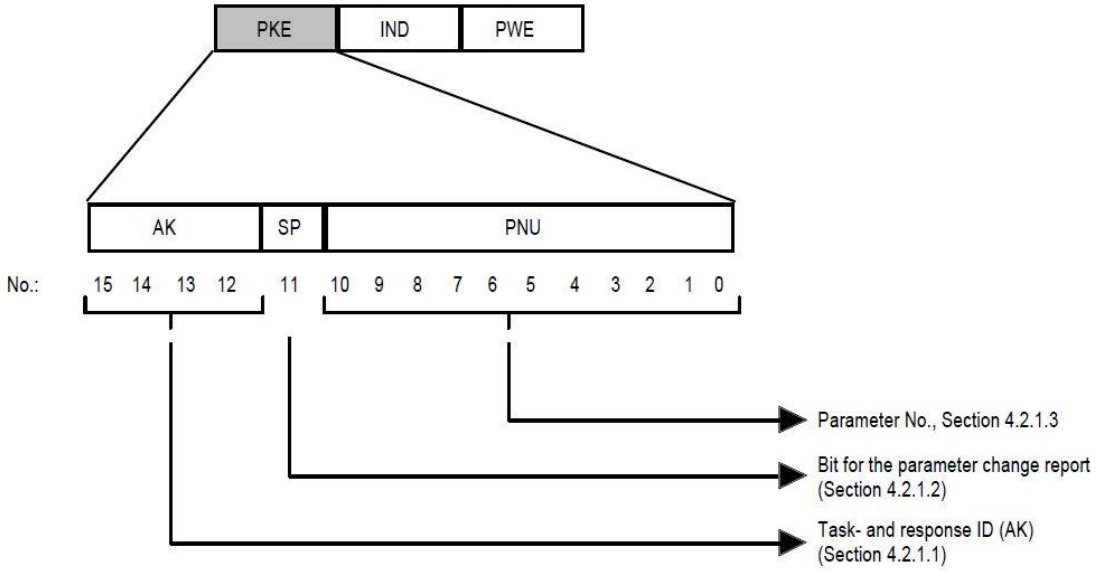
Délka PKW části je proměnlivá a ve spojení s měničem *Sinamics G110* ji lze nastavit na délku 0, 6, 8 bajtů, anebo lze nastavit proměnlivou délku této části. V takovém případě může nabývat délky nula, až 252 bajtů, to závisí ještě na nastavení PZD části, ale o té později.

Možnost nastavit proměnlivou délku se jeví jako velice praktická, jelikož se přenáší takový počet bajtů, který je potřebný a nedochází tak ke zbytečným prodlevám. Zároveň je možné díky tomuto nastavení bezpečně vyčítat všechny hodnoty, parametry a komunikace není limitovaná omezenou délkou PKW části.

3.2.1 Složení Parameter ID value

Samotný blok PKW se skládá ze tří základních částí. Ty jsou *Parameter ID (*zkráceně *PKE), Sub-index* (zkráceně *IND*) *a* nakonec *Parameter Value* (zkráceně *PWE*).

První v pořadí je část PKE, která má pevnou délku a tou jsou dva bajty, nebo-li jeden word. Spodních 11 bitů (0 – 10-tý bit) je vyhrazeno pro číslo parametru (zkratka *PNU*). Z toho lze určit maximální počet parametrů, který činní 2048. Následující jedenáctý bit slouží jako příznak požadavku potvrzení, kdy *slave* zařízení vyžaduje od *master* zařízení potvrzení na zaslaný požadavek (zkratka *SP*).



Tab. 2: Složení PKE části telegramu [1]

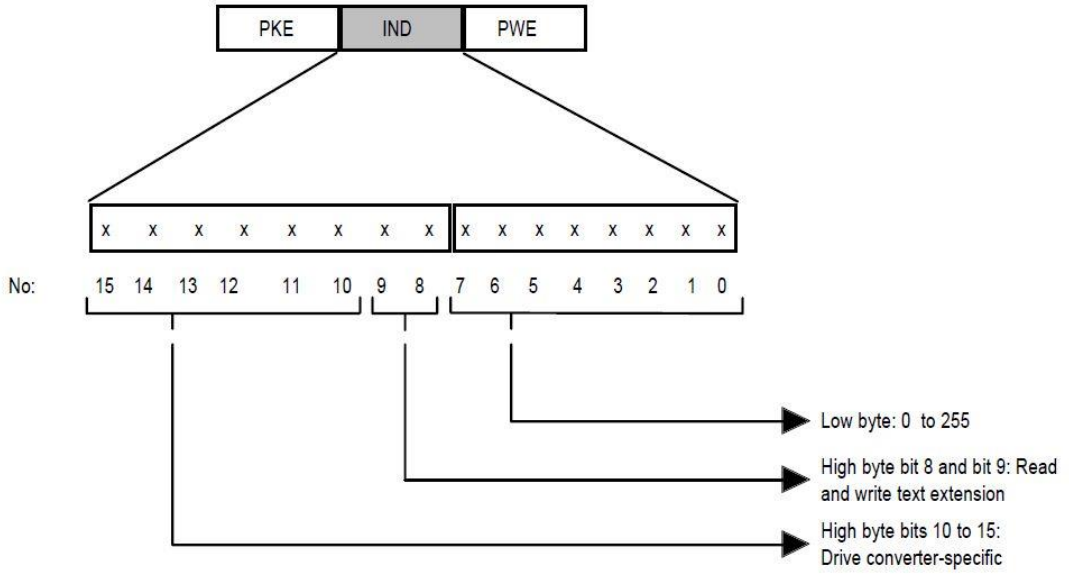
Zbývající čtyři bity (12 – 15-tý bit) slouží jako tzv. *Task and response ID* (zkratka *AK*). Tento segment nese informaci o typu *PEK* požadavku. Zda se jedná o čtení či zápis do parametru (v případě, že telegram odesílá *master* zařízení) a počtu bytů proměnné (zda 16 či 32 bitové).

V případě, že telegram odesílá *slave* zařízení a příchozí hodnota parametru je 16 nebo 32 bitová, tak je zde uložena informace o jakou z těchto hodnot se jedná. V tomto segmentu se také nastavuje typ zápisu do paměti. Jestli je požadován zápis jen do RAM paměti anebo do EEPROM paměti.

Dojde-li k zápisu pouze do RAM paměti, tak při odpojení napájení a následné obnově napájení dojde ke znovunačtení hodnot parametrů z EEPROM paměti. Což má za následek, že změny parametrů, které byly provedeny pouze v RAM paměti, jsou ztraceny.

Následuje segment *Sub-index* (dále už jen jako *IND*), ten má také pevnou délku a to stejnou jako předchozí část, tedy jeden word. U parametrů jež mají dílčí indexy, slouží tento segment k výběru daného indexu a umožňuje s ním následně pracovat, jakoby se jednalo o samostatný parametr s jistými specifiky.

Složení *IND* části je následující, dolních osm bitů slouží pro výběr indexu. Z toho plyne, že rozsah indexů je 0 až 255. Následující dva vyšší bity slouží pro nastavení čtení či zápisu rozšiřujícího textu pro daný parametr a index. Posledních 6 bitů je specifických pro každý měnič a v případě měniče *G110* nebylo zjištěno, že by je jakkoliv využíval. Jsou tedy s největší pravděpodobností rezervovány pro vyšší řady měničů pro specifické funkce.



Obr. 9 Složení IND části telegramu [1]

Nicméně tento fakt nikterak neovlivňuje schopnost vyčítat dané parametry či indexy parametrů. Pro lepší představu o pojmu indexech parametrů uveďme příklad. Nechť je parametr *r0207 „Jmenovitý proud měniče“*. Tento parametr lze pouze číst, proto klíčové označení *r* jako read. Parametr *r0207 s* indexem nula udává *„Jmenovitý proud měniče“*, s indexem jedna udává *„Jmenovitý proud při proměnném točivém momentu“ a s* indexem dva udává *„Jmenovitý proud při konstantním točivém momentu“* [2].

Posledním segmentem, který tvoří PKW část, je *Parameter Value* (dále už jen *PWE*). V tomto segmentu jsou ukládány hodnoty parametru v případě, že *master* zařízení požaduje změnu hodnoty parametru a nebo *slave* zařízení odpovídá na dotaz hodnoty parametru. V případě, že *master* se pouze dotazuje na hodnotu parametru, tak není hodnota *PWE* části brána v potaz a *slave* zařízení ji automaticky ignoruje, je-li přítomna, což pro proměnlivou délku *PKW* nemusí být.

Délka *PWE* části závisí na vybraném typu délky *PKW* části, nicméně pro většinu vyčítaných hodnot je dostačující délka dva až čtyři bajty. Základní délka *PWE* segmentu je jeden word, ale pro vyčítání 32 bitových nebo float proměnných je potřebná délka dva wordy.

3.3 Process data

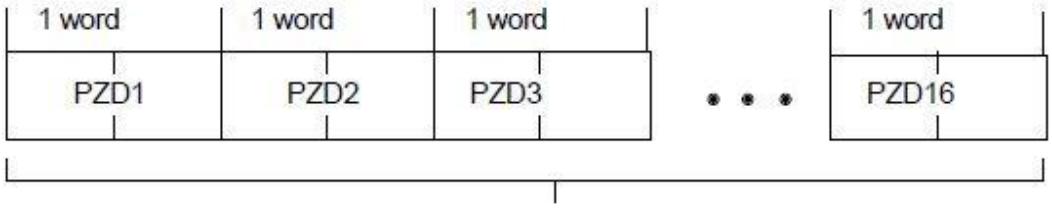
Procesní data (zkráceně *PZD*) jsou částí USS protokolu, která slouží k přenášení řídícího slova a žádané hodnoty, nebo stavového slova a skutečné hodnoty. Jeho pomocí je možné číst a nastavovat stav měniče.

Nejedná se o nezbytnou součást USS telegramu a je tedy možné, aby se tato část při vybraném typu komunikace vůbec nepřenášela anebo byla nulová, je-li to třeba.

Délka *PZD* části je opět volitelná podobně jako u *PKW* části ale s tím, že zde není možné volit proměnlivou délku bloku. Maximální možná délka této části je až 16 wordů tedy 32 bajtů. Nicméně měnič *G110* umožňuje nastavit délku jen 0, 2 nebo 4 wordy.

Příkladem využití této části je požadavek na zapnutí měniče (0x04 0x7F) a nastavení výstupní frekvence na 50 Hz (0x40 0x00). Takto vypadá *PZD* část pro zapnutí měniče a nastavení výstupní frekvence s nastavenou délkou čtyř bajtů.

Pokud by byla délka nastavena na osm bajtů, byl by zbytek doplněn nulami. Celý *PZD* řetězec by pak pro představu vypadal: *0x04, 0x7F, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00*. V případě měniče *G110* je tak možné přenášet až čtyři řídící slova o délce jeden word.



Obr. 10 Složení PZD části telegramu [1]

3.3.1 Složení PZD části

PZD se skládá z jednotlivých PZDx paketů o velikosti jeden word. Ty mohou představovat řídící slovo nebo žádanou hodnotu. Její rozsah je specifický v závislosti na řídícím slově.

Dotazuje-li se *master* zařízení, tak první *PZD* slovo vždy obsahuje tzv. *control word* neboli řídící slovo, ve kterém mu sdělí, jaký stav je od něj vyžadován (např.: zapnutí či vypnutí popřípadě aktivace výstupních svorek atp.). Poté následuje ještě slovo, které udává, jaké hodnoty jsou požadovány. Jako příklad může posloužit již příklad předchozí, kdy se v *control word* slově nastaví požadavek na zapnutí měniče a v *main setpoint* je informace, jaká výstupní frekvence je požadována.

Jeli telegram od zařízení typu *slave*, tak v prvním *PZD* slovo nese tzv. *status word*, tedy informaci v jakém stavu se dané zařízení nachází (popřípadě do jakého stavu přešlo vzhledem k požadavku od *master* zařízení) a aktuální hodnotu dotazovaného stavu.

Pokud je zapotřebí více *main setpoint* slov, tak se slova řadí za sebe. Odpovědí na takový telegram je standardní úvod, kdy první *PZD* slovo obsahuje *status word* a následují *main actual value* je v takovém pořadí, v jakém se *master* zařízení dotázalo.

Jednotlivé bity řídícího slova *PZD* komunikace mají tento význam pro frekvenční měnič *Siemens G110*. Uvádí se to zde jen pro představu, z čeho se řídící slovo může skládat.

Bit 0: zapíná/vypíná frekvenční měnič
Bit 1: povoluje tzv. *„ALL OFF 2“* elektrické vypnutí
Bit 2: povoluje tzv. *„ALL OFF 3“* rychlé vypnutí
Bit 3: povoluje pulsy.
Bit 4: zapnutí rampového generátoru
Bit 5: start rampového generátoru
Bit 6: zapnutí žádané hodnoty
Bit 7: nulování poruchy
Bit 8: krokování vpravo
Bit 9: krokování vlevo
Bit 10: požadavek řízení z řídicího systému

Bit 11: reverzace (změna chodu)

Bit 12: nedefinováno

Bit 13: motor potenciometr zvýšit

Bit 14: motor potenciometr snížit

Bit 15: místní ovládání/dálkové ovládání

Pro účely této práce je využívána pouze *PKW* část. Část *PZD* ve složení telegramu využívající tato práce není přenášena. Je zde uvedena pro doplnění celistvosti informací. S touto částí se autor podrobněji seznámil při realizaci diplomového projektu, kde v rámci zadání nastavoval stavy měniče atd.

4 Aplikace

V této kapitole je popsáno, jak vznikla výsledná aplikace pro komunikaci s frekvenčním měničem. Jaké byly požadavky a jak se v průběhu realizace modifikovaly. Bude nastíněn postup realizace aplikace a kroky vedoucí k finální verzi programu. Následně bude popsána celá aplikace, její jednotlivé části a jejich funkce.

K tomuto úkolu bylo zvoleno vývojové prostředí Visual Studio 2013, .NET framework v jazyce C#, jedná se tedy o WindowsFormsApplication. To bylo zvoleno z několika důvodů. Tím hlavním je dostatečná podpora a veliká programová vybavenost. Dále dobrá znalost tohoto prostředí a zkušenost s prací v něm během studia. Dalším důvodem bylo i to, že k této aplikaci jsou pro studenty dostupné licence, které zpřístupní využití všech funkcí a komponent, které toto prostředí nabízí.

4.1 Požadavky na aplikaci

Mezi hlavní cíle této práce patří tvorba nové aplikace pro komunikaci s měniči a to především s Sinamics V20 a G110, na kterém je aplikace vyvíjena. Pro první zmíněný měnič není v současné době dostupná bezplatná aplikace pro parametrizaci a vyčítání hodnot proměnných. Alternativa v podobě placené aplikace je možná, ovšem její programová vybavenost a rozsáhlé možnosti funkcí mají za následek složitější uživatelské rozháraní. Zdá se být tedy zbytečně složitá na běžnou práci s měničem.

Tato práce si klade za cíl vytvořit takovou aplikaci, pomocí které bude možné parametrizovat a vyčítat hodnoty parametrů měniče a zároveň je bude možné v čitelné podobě ukládat do souboru. Díky cílenému vývoji aplikace pro tyto účely bude výsledné aplikace disponovat přívětivějším a přehlednějším ovládáním než je tomu u placené verze programu od Siemensu. Zároveň ji bude možné spustit na počítačích s vybavením .NET 4.5.

Jako hlavní požadavky, co do funkcí programu, se ukázaly funkce cyklického vyčítání parametrů měniče. Vyčítané parametry by se měli zadávat do tabulky vyčítaných parametrů, kde k nim bude možné přidat poznámku. V této tabulce bude zobrazován i stav vyčítání parametru. Možnost jejich nastavování a ukládání v podobě textových

dokumentů s hodnotami parametrů v čitelné podobě. S textovým dokumentem se pojí i požadavek na schopnost vyčítání již existujících listů a jejich modifikace, jako například přidávání a odebírání položek atp. Podoba ukládaných textových dokumentů by měla být v takové formě, kterou by byl uživatel schopný editovat bez použití programu, tedy přímo v textovém souboru.

Při realizaci této práce přibyl i požadavek na přehledné a smysluplné zobrazovaní stavu komunikace a nastavených parametrů komunikace ve status panelu. Dále pak alespoň základní kontrola zadávaných dat a parametrů s přehlednou zpětnou vazbou o případných chybách anebo možných nastaveních respektive mezí, ve kterých se daná veličina může pohybovat.

4.2 Stručný popis postupu práce

Díky návaznosti této práce na diplomový projekt byla schopnost cyklického vyčítání a ukládání do souboru zvolena jako ústřední část celého programu. Zároveň se navrhovaná struktura nemohla zbytečně komplikovat, jelikož by došlo k nesplnění jednoho z cílů práce, a to návrhu jednoduché a intuitivní aplikace s co možná nejpřívětivějším uživatelským rozhraním.

Prvním úkolem bylo podrobně analyzovat komunikaci po USS protokolu a zjistit její zákonitosti a specifika. Skladbu protokolu, jeho klíčové prvky a k čemu slouží popřípadě další možné variace telegramu. Došlo k upřesnění, v jaké podobě bude s měničem probíhat komunikace, jaké prvky USS protokolu budou přenášeny a které nikoliv v základní komunikaci.

Po této analýze autor postoupil k realizaci jednoduché form aplikace, jejíž účelem bylo po ručním zadání podoby telegramu, jej odeslat a přijmout odpověď s tím, že základní prvky telegramu (uvozovací znak, adresu a na konci telegramu kontrolní součet *BCC*) program doplní automaticky. Komunikace s měničem probíhá prostřednictvím sériového portu RS-232, který je v programu ovládán za pomoci komponenty *Serial port* a k němu nezbytné nastavení, které bude uživatel zadávat. Komunikace probíhá pomocí převodníku z RS-232 na USB rozhraní, které je na straně *masteru* (PC) a na straně *slave* je již výše zmíněné rozhraní RS-232.

Následovala realizace jednoduchého uživatelského rozhraní, kde po zadání nezbytných informací program sestaví celý telegram automaticky ze vstupních dat a následně ho odešle jako dotaz měniči. Současně byl tento návrh schopen zpracovat příchozí zprávu a zobrazovat příchozí data uživateli prostřednictvím jednoduchého zobrazovacího rozhraní.

Dalším krokem byly první testy s nastavováním hodnot parametrů a s cyklickým vyčítáním omezeného počtu parametrů. Pro zobrazování a zadávání hodnot parametrů byl zvolen *textBox*. Každý z parametrů má svoje pole text boxů, ve kterých jsou zobrazeny všechny potřebné informace. Pro účely vývoje se ukázalo jako vhodné použít velké textové okno, do kterého se zobrazují všechny přijaté zprávy. To se ukázalo jako velice praktické hlavně při hledání chyb při komunikaci či zpracování telegramu.

Po úspěšném zvládnutí takové komunikace přešel autor k řešení toho, jak hodnoty parametrů smysluplně cyklicky vyčítat a ukládat do souborů. K tomu se jako nejvhodnější ukázalo zavedení další záložky s tabulkovým rozhraním, ve kterém se celý tento proces bude odehrávat. Zároveň se v této fázi řešila i podoba textového souboru respektive formát dat, ve kterém se budou dané hodnoty ukládat tak, aby byly čitelné i při otevření samotného textového souboru a bylo je tak možné i měnit.

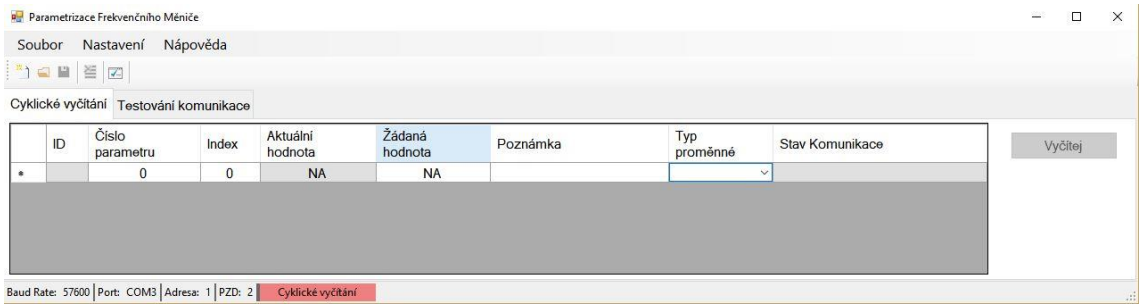
Dále se řešila podoba uživatelského rozhraní, jak přidávat nové řádky do souborů, popřípadě je odebírat. Jak tvořit nový soubor prostřednictvím aplikace. A také jak předejít nechtěným pádům aplikace v důsledku chybového načtení textového souboru, poškozených dat, nebo po zásahu uživatele.

Podoba uživatelského rozhraní dospěla k jistým změnám, až se nakonec ustálila na současné podobě v jednom formuláři a volbou podoby cyklické komunikace. Buď tabulkou anebo jednoduší verzí, která má sloužit hlavně k testování a analýze přijatých či odeslaných telegramů.

V průběhu navrhování aplikace probíhalo také testování aplikace, jak ošetřit zadávání nesprávných dat do příslušných parametrů. Díky tomu bylo odhaleno několik nedokonalostí v chování aplikace. Na závěr byla vytvořena uživatelská nápověda v programu HelpNDoc 4 [5].

4.3 Vzhled aplikace

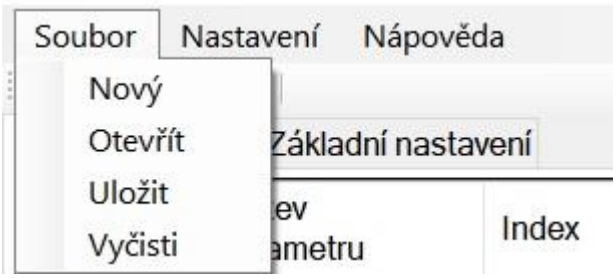
Na Obr. *14* můžete vidět současný vzhled aplikace. Jedná se o základní uživatelské okno, které se zobrazí po spuštění aplikace. Vzhled aplikace lze rozdělit do těchto tří základních částí. Horní část se základními ovládacími prvky. Střední část, kde lze podle výběru záložky realizovat cyklické vyčítání parametrů nebo testovací okno komunikace. A dolní část s informacemi o nastavení a stavu komunikace.



Obr. 11 Úvodní formulář aplikace

V horní části (viz Obr. 13) je umístěno menu s položkami *Soubor*, *Nastavení* a *Nápověda*. Nachází se zde volby pro nastavení parametrů komunikace, dále pak ovládací prvky sloužící pro práci s tabulkou určenou pro cyklické vyčítání. Pomocí nich lze ukládat a číst textové soubory sloužící jako databáze parametrů.

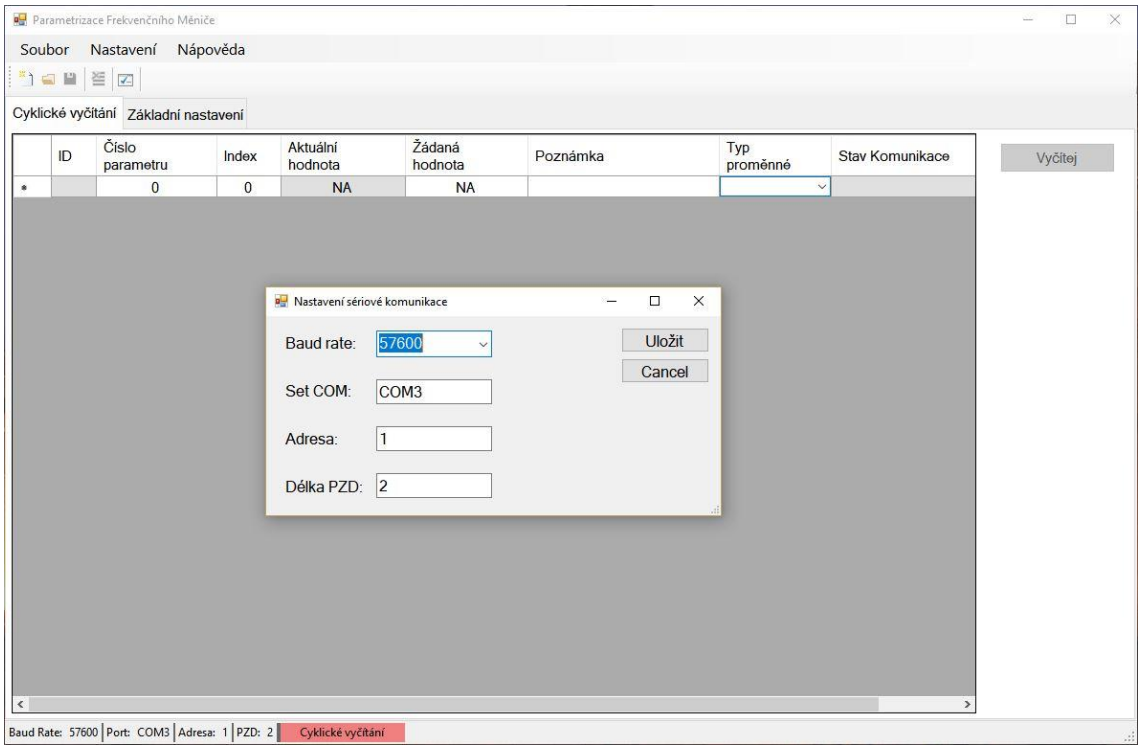
Při stisku volby *Soubor* se zobrazí okno se stejnými možnostmi jako toto rychlé menu. Je zde hlavně z důvodu standardního vzhledu a ovládání aplikace pomocí textových položek.



Obr. 12 Detail na položku Soubor v menu

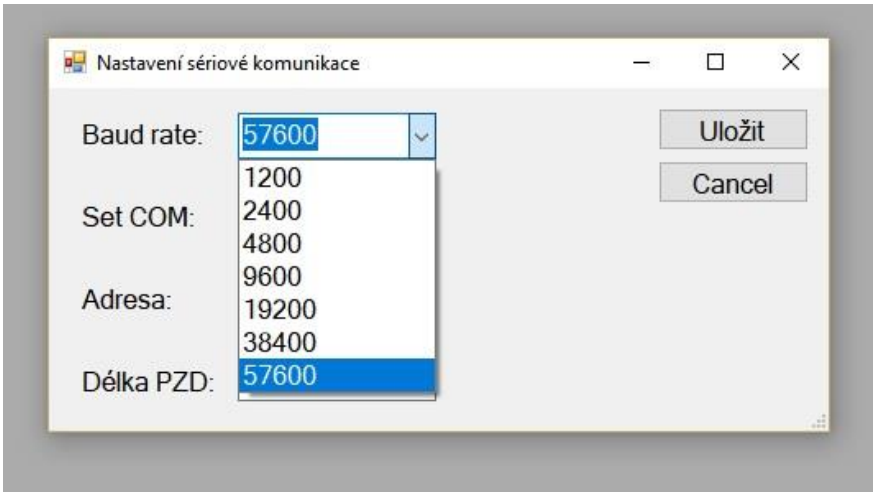
Po stisknutí menu *Nastavení* se zobrazí okno pro nastavení parametrů komunikace. Zde lze nastavit *Baud rate*, příslušný COM port, Adresa koncového zařízení a délka PZD

části. Poslední dva parametry vychází z parametrů komunikace prostřednictvím USS protokolu. Hodnoty parametrů jsou při zadávání kontrolovány a případné špatné hodnoty parametru vyvolají okno s příslušným upozorněním a radou.



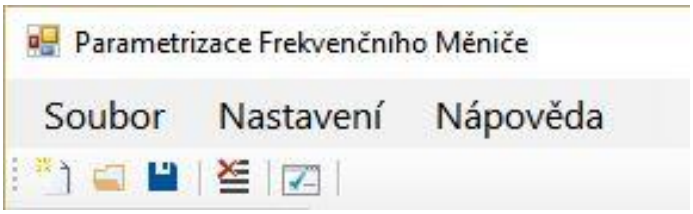
Obr. 13 Nastavení sériového portu

Položka *Baud rate* obsahuje předem definované komunikační rychlosti, pomocí kterých je možné vést komunikaci – možnost výběru koresponduje s podporovanými rychlostmi měniče.



Obr. 14 Detail nastavení sériového portu

Poslední položka *Nápověda* slouží k zobrazení nápovědy k programu, ale o té až později. Nastavení sériové komunikace lze otevřít i pomocí předem připravených tlačítek v menu pro rychlejší přístup.

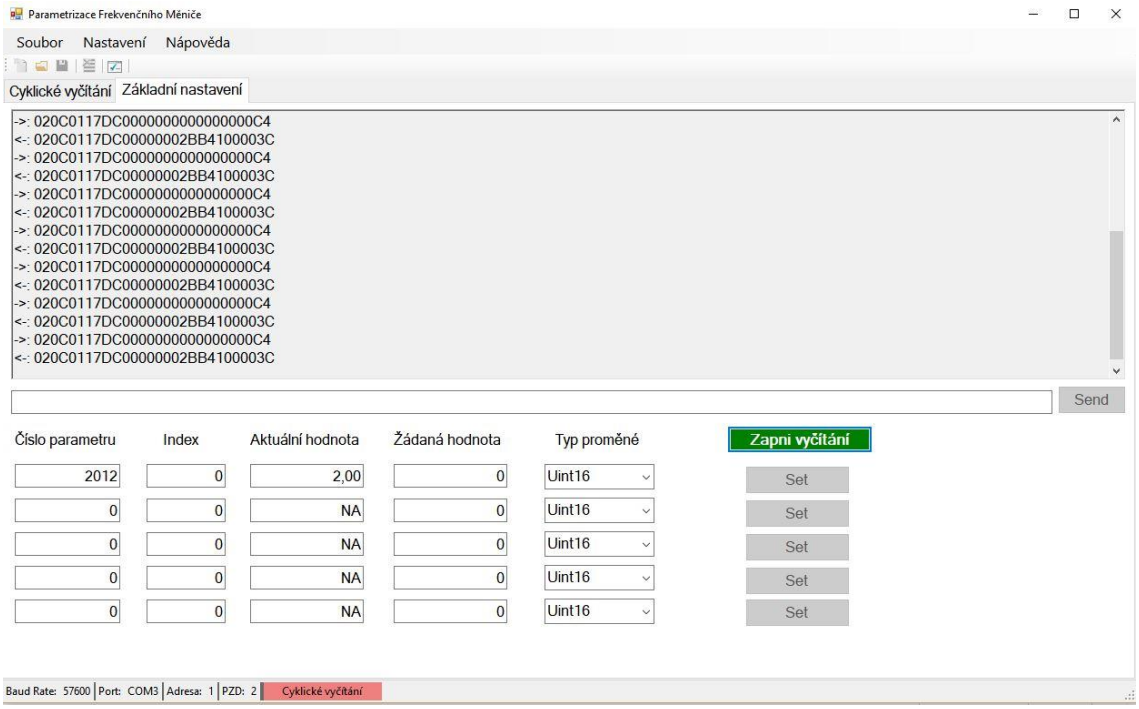


Obr. 15 Uživatelské rozhraní

První ikona zleva slouží pro vytvoření nového seznamu parametrů. Druhá umožňuje načtení již vytvořeného textového souboru s parametry. K uložení aktuálního seznamu parametrů je pak ikona v podobě diskety. Další v pořadí je volba smazat aktuální seznam parametrů. Pomocí poslední ikony je možné otevřít dialogové okno pro nastavení parametrů komunikace. Během těchto úkonů je uživatel dotázán, zda chce opravdu provést tuto akci a zda má uloženou práci.

Všechny tyto možnosti jsou za běhu programu povolovány či zakazovány v závislosti na aktuálním stavu programu. Není žádoucí, aby uživatel mohl měnit například parametry komunikace, když současně provádí cyklické vyčítání parametrů a komunikace tedy běží.

Ve střední části aplikace se nachází oblast pro komunikaci s měničem. Typ komunikace se volí pomocí záložek s názvem *Cyklické vyčítání* a *Testování komunikace*. Druhá v pořadí slouží, jak už název napovídá pro testování komunikace a detekci chyb při ní, je tedy detailnější ve výpisu. Tato záložka původně vznikla jen pro účely testování při vývoji aplikace. Ukázala se však natolik užitečná, že byla zahrnuta i do finální verze programu.



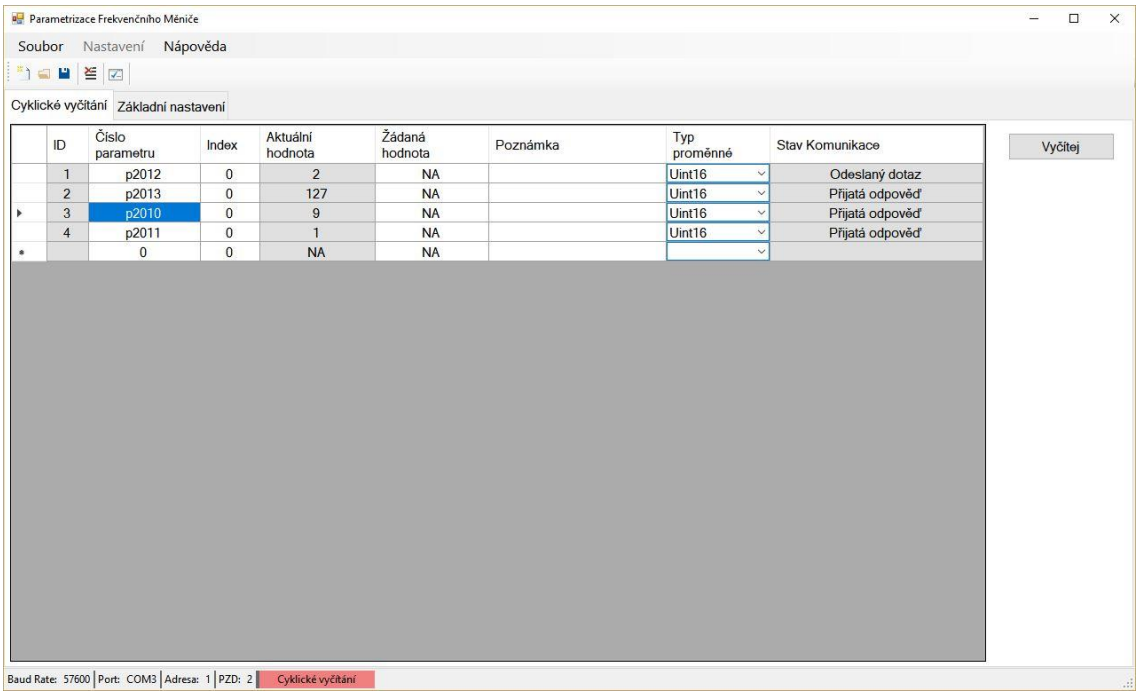
Obr. 16 Záložka pro testování komuniakce

Jak je vidět na Obr. 18, v textovém okně je zobrazena podoba vysílaného i přijímaného telegramu v hexadecimální podobě. Dále je možné ručně sestavit telegram za pomoci řádku umístěného pod textovým oknem a stisknutím tlačítka *Send* jej lze odeslat. Takto sestavený telegram bude doplněn o uvozovací znak 0x02 a kontrolní BCC součet.

Ve spodní části této záložky se nachází oblast pro cyklické vyčítání parametrů a jejich nastavování. Najednou je možné vyčítat až pět hodnot, nastavení parametru na žádanou hodnotu je možné pomocí tlačítka *Set*.

Pro zahájení této komunikace je nezbytné správně vyplnit políčka *číslo parametru*, *index* a *žádaná hodnota*. Pro nastavení parametru na žádanou hodnotu nesmí běžet cyklické vyčítání parametrů (oproti první záložce kde je tato funkce podporována). Aktivace cyklického vyčítání se děje pomocí tlačítka *Zapni vyčítání*, jehož pozadí se po úspěšném zahájení procesu změní z červené na zelenou s nápisem *Vypni vyčítání*.

Stěžejním prvkem této práce je první záložka s názvem *Cyklické vyčítání*.



Obr. 17 Záložka Cyklické vyčítání hodnot parametrů

Slouží pro práci s parametry, k jejich vyčítání, ukládání a editování. K prezentaci hodnot parametrů slouží komponenta *dataGridView*. Pomocí ní je i definován formát dat a jejich podoba v souboru (pořadí, ve kterém se ukládají).

Finální podoba byla podrobena testování a prošla rozsáhlými změnami oproti původnímu návrhu. Tím, co se vyvíjelo, byl obsah samotné tabulky parametrů, editovatelnost a uživatelská přístupnost.

Při prvním návrhu bylo nutné otevírat samostatným tlačítkem SPI port a dalším zapnout cyklické vyčítání parametrů. To se ukázalo jako neelegantní řešení. Což vedlo autora ke sloučení těchto funkcí v jedno tlačítko.

Samotný obsah sloupců v tabulce procházel také změnami, až se nakonec ustálil na současné podobě. Obsahuje osm sloupců, počet řádků je dán počtem parametrů buď ručně vytvořených, anebo z nahraného textového souboru. Sloupce jsou v pořadí: *ID* (číslo řádku), *Číslo parametru, Index parametru, Aktuální hodnota, Žádaná hodnota, Poznámka, Typ proměnné, Stav komunikace*.

V tabulce se objevují i sloupce, do kterých uživatel nemá přístup např. jako *Index řádku*. Tyto uživatelsky nepřístupné sloupce jsou rozlišeny šedým podbarvením pole.

Prvním sloupcem je parametr *ID*, který je vyplňován automaticky vždy po potvrzení řádku. Nese v sobě informaci, na kterém řádku se daný parametr nachází. Barva podbarvení pole je šedá, není možné ji editovat.

Následuje *Číslo parametru*, které uživatel vyplňuje. Slouží pro výběr vyčítaného parametru. Je daný formát zadávání, kde prvním znakem je znak „P“ nebo „R“ v závislosti na tom zda se dá parametr editovat či nikoliv. Čísla parametrů se uživatel dozví v uživatelské příručce k danému měniči. Při nesprávném formátu parametru se objeví okno s informací a chybném zadání a je zde uveden přiklad správného formátu. Za úvodním znakem následuje samotné číslo parametru, které může mít maximálně čtyři znaky a nesmí být záporné, pohybuje se v rozsahu 0 až 2047. Při jakémkoliv nesprávném zadání se opět objevují příslušná okna s hlášením a radou. Toto platí pro celou tabulku parametrů. Tímto se autor snaží usnadnit uživateli seznámení s programem i fr. měničem a ulehčit práci s ním.

Po *Čísle parametru* je možné vybrat i *Index,* disponuje-li jím daný parametr. Mnoho parametrů tento index nemá a v takovém případě postačí ponechat defaultní hodnotu a to nulu. Rozsah této proměnné je 0 až 255.

Navazuje *Aktuální hodnota* parametru, jež uživatel nevyplňuje a je aktualizována při vyčítání hodnot. Při započetí vyčítání je tato hodnota nastavena na NA, aby se posléze rozlišily výstupní hodnoty, které se daří úspěšně vyčítat a které nikoliv. Je to jeden z prvků kontroly kvality komunikace.

Žádaná hodnota parametru slouží pro změnu aktuální hodnoty parametru na uživatelem vybranou. Jsou zde kontrolní mechanismy, aby se pro daný datový typ parametru

nezadávala neplatná čísla (např. pro datový typ Uint16 záporná nebo mimo rozsah). Nastavení této hodnoty probíhá v cyklech, kde se postupně prochází všechny řádky, a porovnává se pole *Aktuální hodnoty* a *Žádané hodnoty*, jsou-li ve shodě, provede se standardní dotaz na hodnotu parametru dle datového typu. V opačném případě je formulován a odeslán telegram s žádostí o změnu parametru na danou hodnotu.

V prvních verzích programu nebylo možné měnit hodnotu parametru při zapnutém cyklickém vyčítání. Což se ukázalo jako uživatelsky nevyhovující a přistoupilo se k editovatelnému konceptu.

Sloupec *Poznámka* byl do programu přidán až jako poslední, jelikož požadavek na něj se začal rýsovat až při finálních fázích vývoje programu. Slouží, jak již název napovídá, k zadání uživatelské poznámky, kterou je pak možné spolu s ostatními parametry uložit do souboru, popřípadě je zpětně načíst.

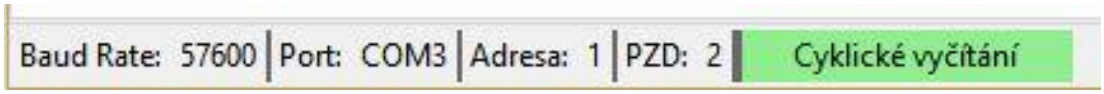
Důležitým parametrem pro správnou komunikaci s měničem a vyčítání parametrů z něj je *Typ proměnné* datového typu parametru. Jelikož se pro různé datové typy používají jiné typy telegramů (délka telegramu a podoba v jaké je hodnota parametru přenášená), je nezbytné, aby uživatel toto pole vyplnil správně. Informaci o datovém typu daného parametru jsou dostupné v manuálu k danému měniči. V tomto případě je možné zvolit jeden z následujících typů: *Uint16, Uint32* a *Float*.

Pro informování uživatele o stavu komunikace slouží sloupec *Stav Komunikace*. Uživatel jej nevyplňuje (má šedé pozadí) a jsou do něj zobrazovány stavy komunikace, kdy byl dotaz na hodnotu odeslán, přijatá odpověď, popřípadě se zde zobrazuje informace uživateli o nesprávně zadaném datovém typu.

Rozlišují se dva typy dotazovacího telegramu a to dotaz na hodnotu, nebo žádost o změnu parametru. Zadá-li uživatel například neplatnou hodnotu parametru, která je ovšem přijatelná v rámci datového typu a nikoliv v rámci parametru, tak nedojde ke změně *Aktuální hodnoty* a při každém dotazovacím cyklu jsou odesílány telegramy typu žádosti o změnu. To je jeden z prvků jak uživatel pozná, že je něco v nepořádku a měl by překontrolovat zadaná data.

Je-li zadán nevhodný datový typ parametru, je to opět z průběhu komunikace rozpoznáno a uživatel je informován o špatném nastavení. Rozpoznání probíhá tak, že je odeslán telegram s dotazem na hodnotu parametru a odpovědí mu je hodnota parametru spolu s informací, o jaký typ se jedná. Při zpracování telegramu je tato informace porovnána s hodnotou zadanou uživatelem a následně vyhodnocena. Takto je možné poměrně rychle a efektivně odhalit chyby v komunikaci.

Nastavení stavu komunikace je zobrazováno ve spodní části aplikace prostřednictvím status panelu. Zde jsou zobrazeny klíčové parametry komunikace jako je *Baud rate*, nastavený COM port, nastavená adresa měniče, délka PZD části a zda je cyklické vyčítání aktivní či nikoliv, respektive jestli komunikace probíhá. Zda je komunikace aktivní je indikováno polem *Cyklické vyčítání* se zeleným či červeným pozadím v závislosti na stavu komunikace, Obr. 20.



Obr. 18 Status panel aplikace

4.4 Struktura programu

Program se skládá z jednoho hlavního formuláře, na němž jsou umístěny všechny potřebné prvky a vytváří vzhled celého programu. Nastavování určitých parametrů se děje pomocí dialogových oken. Program je rozdělen do tří tříd. Jednou z hlavních tříd je `FileProcess`, která ve formuláři pro práci se soubory zpracovává podobu dat tabulky parametrů (formát dat tabulky, přidávání, odebírání parametrů, ukládání a vyčítání souborů).

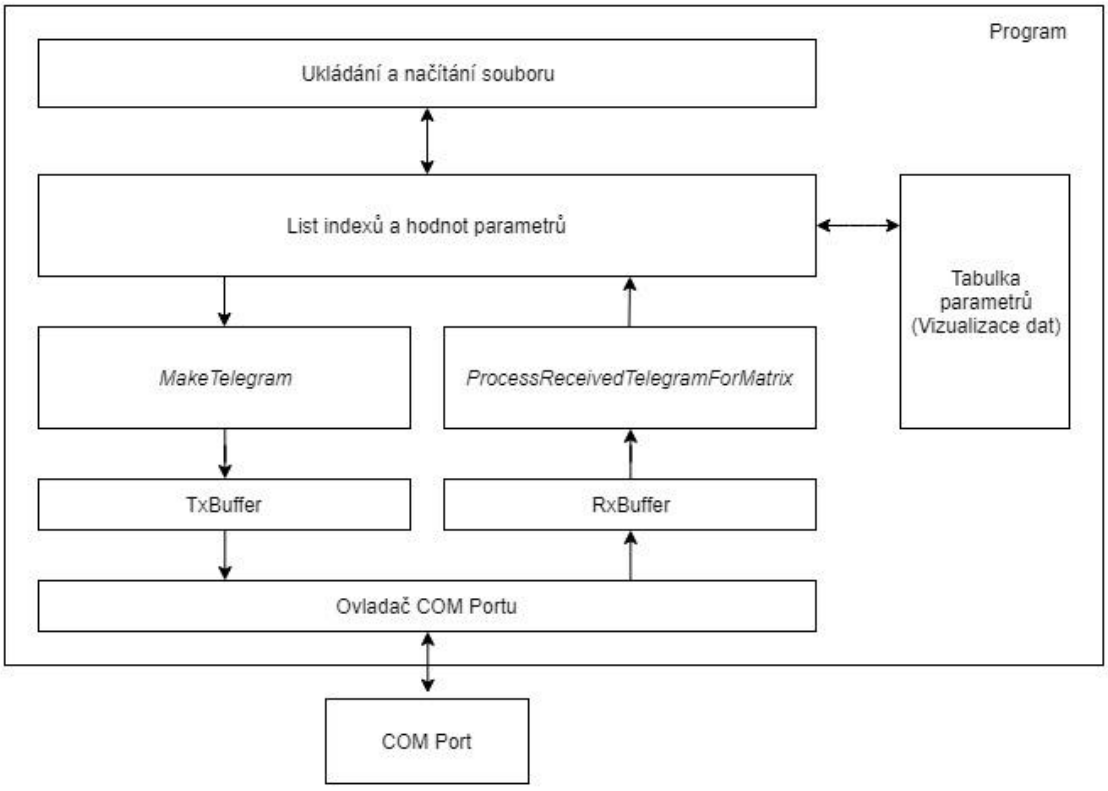
Další třída s názvem `MakeTelegram` slouží k sestavení komunikačního telegramu. Pro získávání potřebných informací z příchozího telegramu byla založena třída `TelegtramInfo`. Velká část kódu se nachází v hlavním formuláři, kde je přístup ke všem komponentám a periferiím.

V menu v horní části formuláře, které je tvořeno komponentami *ToolStripMenu* a *ToolStripButton, je umístěno* nastavení komunikace s vybraným měničem, to je možné

za pomoci dialogového okna s názvem *DlgSettings*. V něm se nastavují parametry komunikace jako např.: komunikační kanál, rychlost komunikace a typ USS protokolu (jakou podobu bude mít PZD část). Tyto parametry jsou posléze zobrazovány ve *StatusBaru* v dolní části formuláře, kde se zobrazuje i aktuální stav komunikace.

V prostřední části formuláře je umístěné záložkové menu, kde se volí druh komunikace s měničem. Tedy cyklické vyčítání, anebo testování komunikace. Tabulka pro cyklické vyčítání je realizovaná pomocí komponenty *DataGridView, z* ní jsou vybrané hodnoty načítány do listu parametrů a ten je posléze použit k cyklickému vyčítání.

Druhá záložka slouží pro testování komunikace a pro ruční sestavení telegramu. Jedná se spíše o pomocný prvek, který vznikl při vývoji této aplikace. Zde jsou v komponentě *RichTextBox* zobrazovány všechny telegramy s příslušným rozlišením, zda se jedná o odesílaný nebo přijímaný telegram.



Obr. 19 Diagram zpracování dat

Na Obr. 21 vidíte diagram, který znázorňuje strukturu zpracování dat a komunikaci s fr. měničem. Týká se hlavní části aplikace, která slouží pro cyklické vyčítání parametrů

a jejich zobrazování do hlavní tabulky. COM Port je ovládán pomocí komponenty *SerialPort*. Detailnější popis jednotlivých částí je v další kapitole.

4.5 Jednotlivé části programu

Tato kapitola se věnuje detailněji samotnému návrhu jednotlivých částí aplikace a postupu při jejich řešení. Budou zde vysvětleny vybrané třídy, funkce a metody, popřípadě zdůvodněno jejich použití. Jsou zde uvedeny části zdrojového kódu. V popisech jsou používána skutečná jména proměnných a dalších datových struktur použitých přímo v kódu.

4.5.1 Vlastnosti nastavení sériového portu

Pro komunikaci po sériové lince (v případě měniče *G110* se jedná o RS-232) je použit sériový port, pomocí kterého je vedena komunikace. Ten je součástí formuláře *Form1*. Je inicializován a defaultně nastaven při otevírání formuláře, ale samotnou komunikaci musí uživatel nastavit prostřednictvím menu v horní části formuláře.

```
static SerialPort serialPortA;
...
        public Form1()
        {
            ...
            serialPortA = new SerialPort();
            ...
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            SetSerialPort();
            serialPortA.DataReceived += serialPortA_DataReceived;
            ...
        }

        public void SetSerialPort()
        {
            serialPortA.BaudRate = 57600;
            serialPortA.PortName = "COM3";
            serialPortA.Parity = Parity.Even;
            serialPortA.DataBits = 8;
            serialPortA.ReadTimeout = 250;
            serialPortA.WriteTimeout = 250;
        }
```

Zdrojový kód 1: Inicializace sériového portu

Pro nastavení parametrů komunikace slouží dialogové okno *DlgSettings*, to ukládá hodnoty nastavené uživatelem. V tomto okně dochází i ke kontrole zadávaných hodnot parametrů po stisknutí tlačítka *Uložit*.

```
private void btnSave_Click(object sender, EventArgs e)
{
    baudRate = int.Parse(comboBox1.SelectedItem.ToString());
    tmpByte = (byte)textBox26.Text.ToString().Length;
    if(tmpByte < 4)
    {
        MessageBox.Show("Zadána neplatná hodnota COM portu – správný
        formát zadávání je COM a číslo portu - neplatný počet znaků",
        "Error");
        return;
    }
    if (!(textBox26.Text.ToString().Substring(0,3).ToUpper() == "COM"))
    {
        MessageBox.Show("Zadána neplatná hodnota COM portu – správný
        formát zadávání je COM a číslo portu", "Error");
        return;
    }
    if (!(byte.TryParse(textBox26.Text.ToString().Substring(3),
    out tmpByte)))
    {
        MessageBox.Show("Zadána neplatná hodnota COM portu – správný
        formát zadávání je COM a číslo portu - číslo portu musí být celé
        nezáportné číslo menší než 256", "Error");
        return;
    }
    ...
    DialogResult = DialogResult.OK;
}
```

Zdrojový kód 2: Testování parametrů sériového portu zadaných uživatelem

Hodnoty parametrů jsou uloženy do příslušných proměnných, pokud jsou správně zadány. V opačném případě se zobrazí okno s adekvátním upozorněním na špatně zadané hodnoty, případně i s informací, v jakém rozmezí se má daná veličina pohybovat, popřípadě jaký formát zadávaní je požadován.

```
private void DlgSettings_Load(object sender, EventArgs e)
        {
            comboBox1.SelectedIndex = baudrateIndex;
            textBox26.Text = setCom;
            textBox23.Text = address.ToString();
            textBox25.Text = pZD.ToString();
        }
```

Zdrojový kód 3: Předávání aktuálních hodnot do dialogového okna

Aby bylo zajištěno aktuálních hodnot v dialogovém okně, při jeho otevírání jsou hodnoty načteny z hlavních proměnných ve *Form1*.

Po úspěšném uložení těchto parametrů se povolí možnost komunikace s měničem tím, že se v první záložce povolí tlačítko s názvem *Vyčítej*. Při jeho stisknutí dochází k nastavení sériového portu na zadané parametry a k otevření portu. Slouží k tomu metoda s názvem *OpenComButton*.

Odeslání telegramu po sériové lince je řešeno funkcí *serialPortA.Write*, do té se jako parametry předávají byte řetězec se zprávou, počáteční index a délka telegramu.

```
public byte[] buffDataSet = new byte[256];

...
serialPortA.Write(buffDataSet, 0, (byte)(buffDataSet[1] + 2));
...
```

Zdrojový kód 4: Funkce pro zápis dat do sériového portu

O zbytek se již stará komponenta sériového portu. Pro čtení ze sériového portu je také předem definovaná funkce s názvem *serialPortA.Read*. Čtení ze sériového portu probíhá pomocí události, kdy když je přijat telegram na sériové lince, tak se vyvolá událost a program přejde k jejímu řešení. Zajímavostí může být to, že událost příchozí zprávy běží ve vlastním programovém vlákně. Díky tomu je možné obsluhovat sériový port, aniž by byl narušen běh zbytku programu. Toto je ovšem spojené se skutečností, že chceme-li volat metodu nebo funkci v tomto vlákně, která je ovšem realizována např. ve formuláři *Form1* je nezbytné použít klíčový příkaz „`this.Invoke((MethodInvoker)delegate`“, to umožní volání funkcí a metod z formulářů v příslušném vlákně. Příklad takového použití je na *Zdrojovém kódu 5*.

```
public byte[] buffDataR = new byte[256];

void serialPortA_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    ...
    this.Invoke((MethodInvoker)delegate
    {
    ...
    });
    while ((serialPortA.BytesToRead <= 1) && !overTime_timer3)
    {
        if (overTime_timer3)
```

```
            break;
    }
    if (overTime_timer3 == false)
    {
        serialPortA.Read(buffDataR, 0, 2);
        while ((serialPortA.BytesToRead < buffDataR[1]))
        {
            if (overTime_timer3)
                break;
        }
        timer3.Stop();
        if (overTime_timer3 == false)
        {
            serialPortA.Read(buffDataR, 2, buffDataR[1]);
            this.Invoke((MethodInvoker)delegate
            {
                if (buffDataR[0] == 0x02)
                    if (buffDataR[Get_Length(buffDataR) - 1] ==
                        BCC(buffDataR, 0, (Byte)(Get_Length(buffDataR) - 2)))
                        {
     ...
}
```

Zdrojový kód 5: Funkce pro příjem telegramu a jeho testování

Zde je uveden příklad přijmutí zprávy spolu se základním ošetřením a pomocnými metodami. Metoda *serialPortA.Read* pracuje podobně jako *serialPortA.Write* jen s tím rozdílem, že se jako první dává prázdný byte řetězec. Další dva parametry mají stejnou funkci.

V této funkci je zahrnuto i testování přijatého telegramu na správnost. V telegramu se kontroluje správnost uvozovacího znaku (0x02), správnost kontrolního součtu *BCC* a délka telegramu.

```
public byte Bcc(Byte[] help, Byte zacatek, Byte delka)
        {
            tmpUI8 = help[zacatek];
            for (Byte i = (Byte)(zacatek + 1); i < (zacatek + delka); i++)
            {
                tmpUI8 = (Byte)(tmpUI8 ^ help[i]);
            }
            return tmpUI8;
        }
```

Zdrojový kód 6: Funkce pro vytvoření kontrolního součtu

K testování zprávy již při jejím přijetí se autor rozhodl z toho důvodu, aby šetřil strojový čas. Pokud by zpráva byla nesprávná, je možné přerušit čtení zprávy, a tím ukončit komunikaci a předejít dalším časovým prodlevám v dalších funkcích.

Při testování správnosti všech prvků (tedy uvozovacího znaku a *BCC* znaku) je spojené i testování odpovídající délky telegramu. Pokud by byla nesprávná, vyšel by i kontrolní součet nesprávně.

```
if (buffDataR[0] == 0x02)
    if (buffDataR[Get_Length(buffDataR) - 1] ==
        BCC(buffDataR, 0, (Byte)(Get_Length(buffDataR) - 2)))
        ...
```

Zdrojový kód 7: Podmínky pro vytvoření kontrolního součtu

Po úspěšném přijetí telegramu probíhá jeho zpracovávání, to bude popsáno v následující kapitole.

Při vyčítání dat ze sériové linky je nutné ošetřit situaci, kdy náhle uživatel ukončí komunikaci nebo zavře celý program. Při neošetření dochází k pádům aplikace popřípadě k nesprávné funkci sériového portu při opětovném započetí komunikace. Řešení tohoto problému vyžadovalo ze strany autora důkladné prostudování problematiky, kde přesně vzniká problém a jak ho co nejlépe ošetřit.

Problém vzniká tehdy, je-li náhle ukončena komunikace. Tedy pokud dojde k zavření sériového portu, ale program i přes to čte ze sériového portu (respektive plní *buffer* příchozích správ). Tato situace nastává v důsledku toho, že pro obsluhu sériového portu je vyhrazeno vlastní vlákno, které běží nezávisle na dění v hlavním vlákně aplikace. Kdežto komponenta sériového portu je spouštěna a nastavována z hlavního vlákna.

Prvotní myšlenka autora, jak tuto situaci ošetřit, byla přidat pomocné proměnné jako příznaky a testování těchto příznaků v průběhu čtení ze sériového portu. To se po mnoha testováních ukázalo jako nedostatečně účinné a to i navzdory tomu, že autor tyto příznaky doplnil ještě časovači, které měly obě tato vlákna alespoň částečně synchronizovat (při přetečení časovače – přesáhnutí stanoveného času pro vyčítání z portu by došlo k jeho ukončení).

To, co se ovšem ukázalo být efektivní variantou, bylo ošetření místa, kde by mohlo dojít k pádu aplikace, pomocí příkazů `try-catch`. V případě vzniku nežádoucího chování bylo nutné celý port zavřít v navazující metodě `catch` v kombinaci s odebíráním události příchodu zprávy z komponenty sériového portu. Díky tomu program nepřejde

do obsluhy pro vyčítání přijaté zprávy a nemůže v důsledku zavřeného sériového portu dojít k pádu aplikace. Tato podoba ošetření se ukazuje být funkční a již nedochází k pádům aplikace.

```
public void StopComunication()
{
    try
    {
        serialPortA.DataReceived -= serialPortA_DataReceived;
        Thread.Sleep(100);
        serialPortA.DiscardInBuffer();
        serialPortA.DiscardOutBuffer();
        serialPortA.Dispose();
        serialPortA.Close();
    }
    catch
    {
        Thread.Sleep(40);
        serialPortA.Close();
    }
}
```

Zdrojový kód 8: Ošetření ukončování sériového portu

4.5.2 Zpracování příchozího telegramu

Pro zpracování zprávy byly vytvořeny dvě metody s názvy *ProcessReceivedTelegramForMatrix*, *ProcessReceivedTelegramForTable*. První jmenovaná slouží ke zpracování příchozího telegramu, je-li komunikace vedena z druhého formuláře. Druhá metoda je pro vedení komunikace z první záložky pro cyklické vyčítání.

Před samotným zpracováním telegramu probíhá ovšem v metodě pro obsluhu události příchozího telegramu testování, zda vůbec přijatý telegram obsahuje základní prvky, které musí nezbytně obsahovat. Testuje se zde tedy délka telegramu vzhledem k počtu přijatých bajtů.

```
...
serialPortA.Read(buffDataR, 0, 2);
while ((serialPortA.BytesToRead < buffDataR[1]))
{
    if (overTime_timer3)
        break;
}
timer3.Stop();
if (overTime_timer3 == false)
{
    serialPortA.Read(buffDataR, 2, buffDataR[1]);
```

```
    this.Invoke((MethodInvoker)delegate
    {
        if (buffDataR[0] == 0x02)
            if (buffDataR[Get_Length(buffDataR) - 1] ==
                BCC(buffDataR, 0, (Byte)(Get_Length(buffDataR) - 2)))
            ...
```

Zdrojový kód 9: Základní testování příchozího telegramu

Začátek testování začíná zapnutím časovače s názvem *timer3*, ten slouží pro ochranu situace, kdy by nebyla odvysílána celá zpráva. Následně se nahrají první dva bajty pomocí metody `serialPortA.Read(buffDataR, 0, 2)`. Je zjištěna uvažovaná délka telegramu a nahrán zbytek telegramu s tím, že kdyby došlo k výraznému zdržení při načítání telegramu, je celá akce ukončena kvůli překročení časové konstanty, kterou hlídá *timer3*.

Po úspěšném přijetí celého telegramu probíhá testování, zda uvozovací znak má požadovanou podobu (0x02) a souhlasí kontrolní součet. Samotné rozhodnutí, které metodě bude telegram předán ke zpracování, značí pomocná proměnná `signOfRead`, která podle hodnoty do ní uložené vybere správnou metodu.

```
sigReceiveOK = true;
if (signOfRead == 10) // pro vyčítání z druhé záložky - datagrid
{
    if (dataGridView1 != null)
    {
        ProcessReceivedTelegramForTable(buffDataR);
    }
}
else
{
    ProcessReceivedTelegramForMatrix(buffDataR);
    buttonZapMen.Enabled = false;
    WriteRTB(buffDataR, 0, (byte)(buffDataR[1] + 2), "<-: ");
    buttonZapMen.Enabled = true;
    sigReceiveError = false;
}
```

Zdrojový kód 10: Náhled kódu pro výběr metody ke zpracování

Pokud je vybrána metoda `ProcessReceivedTelegramForTable` pro zpracování telegramu v tabulce, je přijatý telegram zpracován. Náhled metody viz *Zdrojový kód 11*.

Pomocí třídy `TelegramInfo` respektive její instance a funkce `Telegram_Number()` se zjistí číslo parametru z telegramu. Získané číslo parametru se použije jako

vyhledávací klíč pro nalezení hodnoty v seznamu. Není-li dané číslo parametru nalezeno, jsou následující kroky přeskočeny, jelikož se daný parametr vyhodnotí jako nedotazovaný (částečně i ochrana proti několikanásobnému zkomolení telegramu). Index parametru v seznamu hodnot se uloží a využije se následně jako klíč pro zapisování hodnoty parametru. Níže náhled funkce (Zdrojový kód 12) získání čísla parametru.

```
private void ProcessReceivedTelegramForTable(byte[] telegram)
{
    ushort tmpUI16_3 = new ushort();
    tmpUI16_2 = 0xffff;
    tmpUI16 = telegramInfo.telegram_Number(telegram);

    for (UInt16 tmp = 0; tmp < dataGridView1.RowCount; tmp++)
    {
        UInt16.TryParse(dataGridView1.Rows[tmp].Cells[1].Value.ToString().
                                                    Sub-string(1),outtmpUI16_3);
        if (tmpUI16_3 == tmpUI16)
        {
            tmpUI16_2 = tmp;
            break;
        }
    }
}
```

Zdrojový kód 11: Metoda pro zpracování telegramu - Tabulka

```
public UInt16 telegram_Number(byte[] retez)
{
     tmpUI16 = retez[3];
     tmpUI16 = (UInt16)((tmpUI16 << 8) + retez[4]);
     tmpUI16 = (UInt16)(tmpUI16 & 0x7FF);
     return tmpUI16;
}
```

Zdrojový kód 12: Funkce pro získání čísla parametru

V těle metody probíhá sestavení prvku adresa (délka 1 *word*, první dva řádky) následně se adresa demaskuje (maska 0x7FF) a prostřednictvím klíčového slova *return* je vrácena jako návratová hodnota metody.

Program zjistí, zda daný řetězec přijal 16 bitovou nebo 32 bitovou proměnnou a podle toho je testován na datový typ a posléze ve správném formátu zapsán.

```
if ((tmpUI16_2 != 0xffff))
{
    switch (GetPKE_ID(telegram))
    {
    case 1:
        if (dataGridView1.Rows[tmpUI16_2].Cells[6].Value.ToString() =="Uint16")
        {
            dataGridView1.Rows[tmpUI16_2].Cells[3].Value =
                TelegramInfo.telegram_ValueUI16(telegram).ToString();
            dataGridView1.Rows[tmpUI16_2].Cells[7].Value = "Přijatá odpo-věď";
        }
        else
            dataGridView1.Rows[tmpUI16_2].Cells[7].Value = "Špatný datový typ –
            Uint32/Float";
        break;
    case 2:
        ...
    default:
        break;
    }
}
```

Zdrojový kód 13: Funkce pro získání hodnoty parametru

K získání správné hodnoty parametru z telegramu jsou určené funkce *telegram_ValueUI16*, *telegram_ValueUI32* a *telegram_ValueFloat* nacházející se ve třídě *TelegramInfo*. První dvě funkce jsou si podobné, jediný rozdíl je různý počet byte proměnných s nimiž pracují a návratová hodnota. Tělo funkcí je uvedeno na *Zdrojovém kódu 14*.

```
public UInt16 telegram_ValueUI16(byte[] retez)
{
    tmpUI16 = retez[7];
    tmpUI16 = (UInt16)((tmpUI16 << 8) + retez[8]);
    return tmpUI16;
}

public UInt32 telegram_ValueUI32(byte[] retez)
{
    tmpUI32 = retez[7];
    tmpUI32 = (UInt32)((tmpUI32 << 8) + retez[8]);
    tmpUI32 = (UInt32)((tmpUI32 << 8) + retez[9]);
    tmpUI32 = (UInt32)((tmpUI32 << 8) + retez[10]);
    return tmpUI32;
}
```

Zdrojový kód 14: Funkce pro získání hodnoty parametru Uint16

Funkce *telegram_ValueFloat* se již více liší od předchozích, jelikož příchozí telegram obsahuje *Float* číslo v bitové reprezentaci. Je tedy nezbytné takovou proměnnou správně identifikovat a následně zpracovat do standardní podoby. K tomu již tedy slouží tato metoda. Další zajímavosti je i to, že toto *Float* číslo přijde rozdělené do bytů (stejně jako předchozí proměnné typu *UInt16 nebo UInt32*) ale v přeházeném pořadí. Je tedy nezbytné prohodit pořadí bytů tak, aby korespondovalo s bitovou *Float* reprezentací. K tomu slouží funkce *TwistByteArray*, do které se jako vstupní parametry dávají vstupní pole bajtů, výstupní pole bajtů, počáteční a koncový index.

```
public void TwistByteArray(byte[] retez, byte[] vystupniretez, byte index, byte
delka)
{
    for (byte i = 0; i < (delka); i++)
    {
        vystupniretez[i] = retez[(index + delka - 1) - i];
    }
}
```

Zdrojový kód 15: Funkce pro prohození pořadí bajtů

Po této úpravě vstupního telegramu je již možné zpracovat bitovou *Float* proměnnou na standardní *Float* proměnnou, se kterou je možné standardně pracovat. K tomuto převodu je použita třida `BitConverter` s funkcí `ToSingle` do které se vkládají jako vstupní parametry řetězec bajtů a počáteční index. Od počátečního indexu se vždy vezmou následující čtyři bajty, ze kterých je sestaveno *Float* číslo.

```
public float telegram_ValueFloat(byte[] retez) {
    TwistByteArray(retez, tmpByteS4, 7, 4);
    tmpFloat = BitConverter.ToSingle(tmpByteS4, 0);
    return tmpFloat;
}
```

Zdrojový kód 16: Funkce pro získání hodnoty parametru Float

Následně je možné aktuální hodnoty parametrů zapisovat do tabulky parametrů. O zapsání aktuální hodnoty daného parametru je uživatel informován změnou sloupce *Stav komunikace* na příslušném řádku v tabulce parametrů, tam se zobrazí informace „*Přijatá odpověď*“ (více je vysvětleno v kapitole *tabulka parametrů*).

4.5.3 Sestavení telegramu pro komunikaci

Pro sestavení dotazovacího telegramu, který se přenáší prostřednictvím sériové komunikace je realizována třída s názvem *MakeTeLegram* s funkcí stejného jména. Pomocí této funkce lze sestavovat dva druhy telegramů, a to telegram s dotazem na hodnotu parametru a s příkazem ke změně aktuální hodnoty parametru. To, zda se jedná o tzv. „čtecí“ či „zapisovací“ telegram, rozhoduje parametr vkládaný při volání funkce. Návratová hodnota funkce je nastavena jako základní rozlišení, jestli se povedlo sestavit telegram nebo ne, je tedy typu *bool*.

```
bool MakeTelegram(byte adresa, byte index, UInt16 numberOfParameter, UInt32 value,
Form1.writeOrReadt writeorread, Form1.dataType dataType,byte[] retez, byte PZD)
```

Zdrojový kód 17: Funkce pro sestavení telegramu 1/3

Další parametry, které se při volání funkce předávají, jsou adresa měniče, index daného řádku v tabulce parametrů, číslo parametru, hodnota parametru (důležitá jen když ji chceme zapisovat), typ telegramu (čtecí či zapisovací), datový typ, řetězec bytů do kterého se složený telegram uloží a délka PZD části. Hlavička funkce je na *Zdrojovém kódu 17*.

```
retez[0] = 0x02;//uvozovací znak
retez[2] = adresa;//adresa
retez[1] = (byte)((PZD * 2) + 2); retez[1] = (byte)(retez[1] + (2 * 2));/
retez[3] = (byte)(numberOfParameter >> 8);
retez[4] = (byte)numberOfParameter;
retez[5] = (byte)((index >> 8));
retez[6] = (byte)index;
```

Zdrojový kód 18: Funkce pro sestavení telegramu 2/3

Telegram se skládá z uvozovacího znaku (0x02), následuje informace pro kterou adresu měniče je telegram určen. Následně se částečně vyplní délka telegramu, ta ale bude

zpřesněna v průběhu sestavování telegramu. Doplní se číslo proměnné a index, náhled je ve *Zdrojovém kódu 18*.

Poté následuje část PKW, pomocí které se realizuje čtení a zapisování hodnot parametrů. Je zde voleno, zda se jedná pouze o dotaz na hodnotu parametru či o nastavení parametru na příslušnou hodnotu. Jedná-li se pouze o dotaz na hodnotu parametru, tak se pro bajt s indexem 3 provede logický součet s hodnotou 0x10, tím se nastaví jako dotazovací. A upraví se délka telegramu (bajt s indexem 1), přičtou se k němu dvě (za přenášenou hodnotu).

```
if (writeorread == Form1.writeOrReadt.Read) {
    retez[3] = (byte)(retez[3] | 0x10);
    retez[7] = 0x00
    retez[8] = 0x00;
    retez[1] = (byte)(retez[1] + 2);
}
```

Zdrojový kód 19: Funkce pro sestavení telegramu 3/3

Je-li požadavek na zápis, je nezbytné vědět, o jakou proměnou se jedná, aby mohl být telegram správně sestaven. To je rozdíl oproti dotazovacímu telegramu, kde toto není potřeba znát.

```
else if (writeorread == Form1.writeOrReadt.Write)
{
    switch (dataType)
    {
        case Form1.dataType.UInt16:
            {
                retez[7] = (byte)(value >> 8);
                retez[8] = (byte)(value);
                retez[3] = (byte)(retez[3] | 0x20);
                retez[1] = (byte)(retez[1] + 2);
            }
            break;
        case Form1.dataType.UInt32:
            {
                retez[7] = (byte)((value >> 24));
                retez[8] = (byte)((value >> 16));
                retez[9] = (byte)((value >> 8));
                retez[10] = (byte)value;
                retez[3] = (byte)(retez[3] | 0x30);
                retez[1] = (byte)(retez[1] + 4);
            }
            break;
        case Form1.dataType.Float:

                retez[7] = (byte)((value >> 24));
```

```
                retez[8] = (byte)((value >> 16));
                retez[9] = (byte)((value >> 8));
                retez[10] = (byte)value;
                retez[3] = (byte)(retez[3] | 0x30);
                retez[1] = (byte)(retez[1] + 4);

            break;
        default:
            return false;
    }
}
```

Zdrojový kód 20: Funkce pro sestavení zapisovacího telegramu

Zápis 32 bytových proměnných se neliší, nicméně autor se rozhodl toto rozlišovat kvůli možné detekci chyb a vnitřní programové čistotě. Kód pro vytvoření zapisovacího telegramu je vidět níže. Nakonec se přidají prázdné bajty, aby odpovídali délce PZD části.

```
for (byte i = 0; i < (PZD * 2); i++)
{
    retez[retez[1] + 3 + i] = 0x00;
}
```

Zdrojový kód 21: Doplnění telegramu do správné délky

Celý telegram je vždy ukončen kontrolním součtem nebo-li BCC bajtem, který se nesestavuje v této funkci, ale je pro něj napsána vlastní funkce. Kontrolní součet je definován přímo příručkou pro USS protokol. Jeho sestavení spočívá v provedení funkce XOR na všechny bajty celého telegramu. Ukázka kódu viz *Zdrojový kód 22*.

```
public byte BCC(Byte[] help, Byte zacatek, Byte delka)
{
    tmpUI8 = help[zacatek];
    for (Byte i = (Byte)(zacatek + 1); i < (zacatek + delka); i++)
    {
        tmpUI8 = (Byte)(tmpUI8 ^ help[i]);
    }
    return tmpUI8;
}
```

Zdrojový kód 22: Funkce pro vytvoření BCC součtu

4.5.4 Cyklické vyčítání hodnot parametrů

Schopnost cyklického vyčítání hodnot parametrů z měniče za pomoci souboru parametrů je jedním z klíčových prvků této práce. Aktivace této funkce je prostřednictvím stisknutí tlačítka *Vyčítej*, kde se vyvolá událost mající za následek sestavení všech potřebných telegramů pro dotázání se na hodnotu parametru, které jsou následně uloženy do listu polí bajtů.

Aktivací komponenty `timer` se v předem definovaných časových intervalech odesílají dotazy na hodnoty parametrů, které jsou uloženy v listu polí bajtů s názvem proměnné `byteList`. Kód zrealizovaný pro toto je na *Zdrojovém kódu 23.*

```
private void btnRead_Click(object sender, EventArgs e)
{
    if (btnRead.Text == "Vyčítej")
    {
        byteList.Clear();
        for (byte i = 0; i < (dataGridView1.RowCount); i++)
        {
            MakeTelegram(i, writeOrReadt.Read, tmpByteS, adresa);
            tmpByteS[tmpByteS[1] + 1] = BCC(tmpByteS, 0, (byte)(tmpByteS[1] + 1));
            byteList.Add((byte[])tmpByteS.Clone());
        }
        timer1.Enabled = true;
        btnRead.Text = "Stop";
        timer1.Enabled = false;
        btnSet.Enabled = false;
        acutalRow = 0;
        prijatyTelegram = 0;
        chybnyTelegram = 0;
        textBox1.Text = 0.ToString();
        textBox2.Text = 0.ToString();
    }
    else
    {
        timer1.Enabled = false;
        btnRead.Text = "Vyčítej";
        btnSet.Enabled = true;
    }
}
```

Zdrojový kód 23: Funkce pro cyklické vyčítání hodnot parametrů

V kódu je vidět i příklad doplnění kontrolního součtu pro každý jednotlivý telegram. Zajímavostí může být to, že při vkládání pole bajtů do `byteListu` pomocí metody `Add(x)` se ukládali pouze ukazatele na toto pole a nikoliv obraz celého pole. To mělo za následek, že všechny pozice v listu byly obsazeny posledním sestaveným a přidaným

polem bajtů, tedy telegramem (pole pointerů na jeden a ten samý prvek). Řešením bylo použití příkazu *Clone*, který celý prvek zkopíruje do listu a zamezí se tak nežádoucímu chování.

Akce stisknutí tlačítka je svázána i se změnou textu tlačítka tak aby vše bylo pro uživatele co možná nejjednodušší. Jsou zde i potlačeny funkce některých tlačítek pro ochranu správného běhu komunikace, potažmo programu.

Cyklické vyčítání je realizováno pomocí časovače *timer2*, který je v této metodě aktivován. Při vzniku události, kdy *timer2* dosáhl intervalu, přejde program obsloužit toto přerušení. Pomocí něho se cyklicky odesílá vždy jeden telegram jako dotaz na hodnotu parametru. Kód k vidění na *Zdrojovém kódu 24*.

```
private void timer2_Tick(object sender, EventArgs e)
{
    timer2.Enabled = false;
    serialPortA.Write(byteList[acutalRow], 0, (byte)(byteList[acutalRow][1] + 2));
    acutalRow++;
    if (!(acutalRow < dataGridView1.RowCount))
        acutalRow = 0;
    timer2.Enabled = true;
}
```

Zdrojový kód 24: Časovač pro cyklické vyčítání

První co se vykoná je zastavení *timeru2*, následuje odeslání telegramu prostřednictvím metody *serialPortA.Write,* poté se inkrementuje ukazatel na řádek parametru. Následuje testování, zda neukazuje mimo tabulku, pokud ano je vynulován tím je zajištěn cyklus. Nakonec se aktivuje časovač *timer2*.

4.5.5 Práce se soubory

Pro ukládání dat parametrů do souboru a jeho čtení byla vytvořena třída *FileProces*. V ní jsou definovány funkce *SaveDataToFile* pro ukládání dat a čtení dat *ReadDataFile*.

```
public bool ReadDataFile(string directoryPath)
{
    string line; int cnt = 0; bool errorTryPrase = false;
    try
    {
```

```
        using (StreamReader sr = new StreamReader(directoryPath))
        {
            while ((line = sr.ReadLine()) != null)
            {
                errorTryPrase = false;
                if (cnt == 0)
                {
                    cnt++;
                    continue;
                }
                string[] data = line.Split(';');
                DataStruct newLine = new DataStruct();
                for (UInt16 tmp = 0; tmp < data.Length; tmp++)
                {
                    switch (tmp)
                    {
                        case 0:
                            if (!UInt16.TryParse(data[tmp], out newLine.ID))
                            {
                                errorTryPrase = true;
                                continue;
                            }
                            break;
                        case 1:
                            if (!UInt16.TryParse(data[tmp], out newLi
                                ne.parameterNUmber))
                            {
                                errorTryPrase = true;
                                continue;
                            }
                            break;
                        case 2:
                   ...

                        default:
                            errorTryPrase = true;
                            break;
                    }
                }
                if (!errorTryPrase)
                {
                    listDataStr.Add(newLine);
                    cnt++;
                }
        ...
}
```

Zdrojový kód 25: Funkce pro čtení souboru

Po výběru souboru následuje jeho načtení. To probíhá tak, že se první řádek přeskočí, jelikož je vyhrazen pro popis dat kvůli čitelnosti pro případnou ruční editaci či kontrolu. Načte se první platný řádek a použije se na něj metoda *Split(';')*. Následně jsou data načítána do příslušných sloupců v daném řádku. Byla zde ošetřena možnost načtení zkomoleného znaku či řádku. V takovém případě bude uživatel informován o této chybě

MessageBoxem, ve kterém se zobrazí informace o chybě. Kód je možné vidět na *Zdrojovém kódu 25.*

Pro ukládání vytvořené či načtené a upravené tabulky hodnot parametrů slouží již výše zmíněná metoda `SaveDataToFile`. Stejně tak jako u funkce pro čtení souboru se jako vstupní parametr metody udává cesta, kde má být soubor uložen. Zde je i definováno jakou pobudu budou mít ukládaná data. Pro ošetření nepředvídatelných chyb je funkce ošetřena pomocí `try` *a* `catch` metody a opět je případná chyba zobrazována uživateli prostřednictvím *MessageBoxu.* Kód je možné vidět na *Zdrojovém kódu 26.*

```
public bool SaveDataToFile(string directoryPath)
{
    try
    {
        using (StreamWriter sw = new StreamWriter(directoryPath))
        {
            sw.WriteLine("ID;ParameterNumber;Index;ActualValue;WantedValue;
                          VariableType;ReadOnly");
            foreach (DataStruct row in listDataStr)
            {
                sw.WriteLine(String.Format("{0};{1};{2};{3};{4};{5};{6}", row.ID,
                  row.parameterNUmber, row.Index, row.actualValue, row.wantedValue,
                  row.variableType, row.readOnly));

            }
         return true;
        }
    }
    catch (Exception e)
    {
        MessageBox.Show("Chyba při zápisu do txt souboru" + e, "Error");
         return false;
    }
}
```

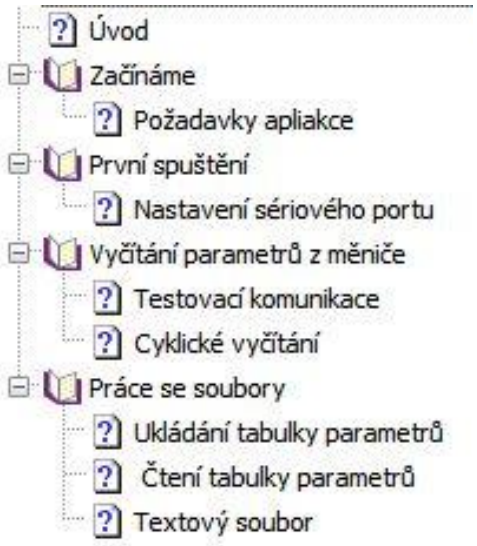
Zdrojový kód 26: Funkce pro ukládání souboru

Data jsou ukládána do řádku v pořadí *ID* parametru (číslo řádku), *Číslo parametru*, *Index* (parametru), *Aktuální hodnota*, *Žádaná hodnota*, *Poznámka*, *Typ proměnné*. Dojde-li k chybě, je dále pronášena pomocí návratové hodnoty a program příslušně reaguje.

5 Uživatelská nápověda

Jeden z důležitých požadavků na aplikaci byla také uživatelská přívětivost a možnost otevření nápovědy. Z těchto důvodů jsou v celé aplikaci umístěny nápovědní okénka ve formě *ToolTipu,* která jsou umístěna nejen na tlačítkách, ale i na některých polích určených k vyplňování, či pouze ke čtení.

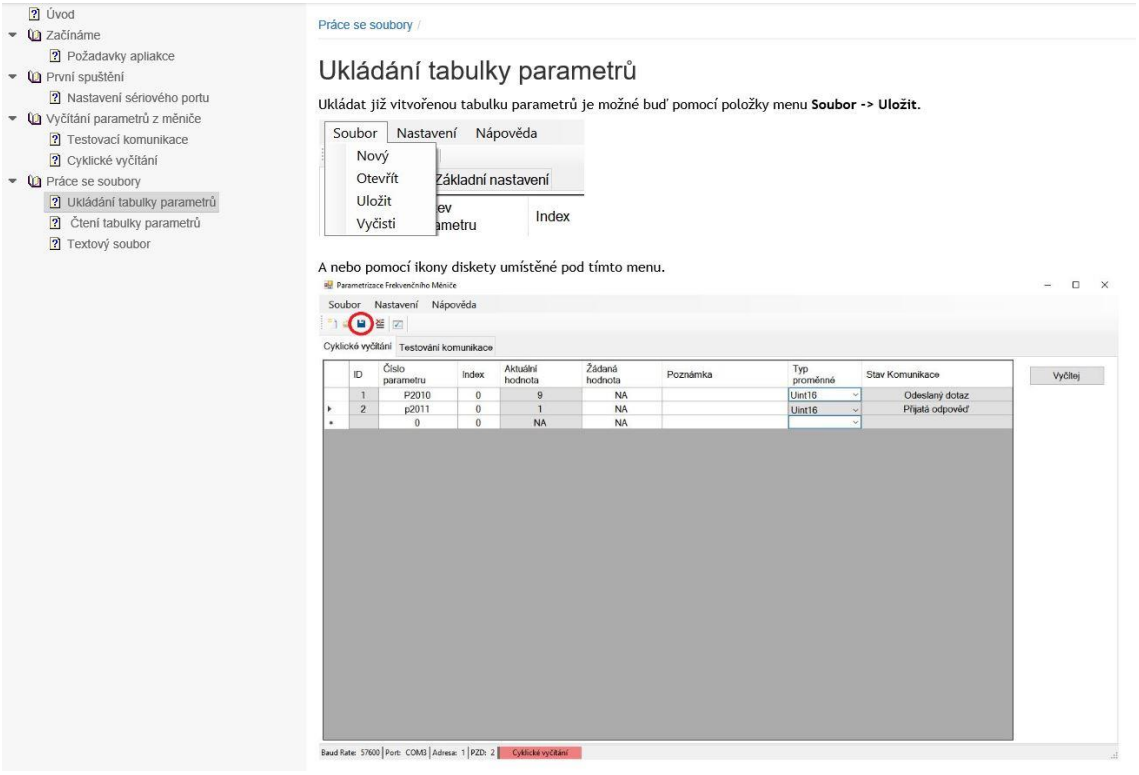
Další kompletnější forma nápovědy je k dispozici v položce horního menu, která nápovědu otevře. Tato nápověda byla vytvořena v programu HelpNDoc 4 [5], který je volně použitelný pro nekomerční účely. Jako výsledný formát nápovědy byl zvolen HTML soubor, ve kterém ve výsledku nebyly problémy s českou diakritikou, oproti klasickému formátu CHM, užívaného běžně pro Microsoft nápovědu. Nápovědu lze otevřít v prohlížeči InternetExplorer. Vytváření nápovědy je v tomto programu velice jednoduché a intuitivní. Prostředí připomíná velice zjednodušený textový editor s možností vkládání odkazů do textu, organizací stromové struktury témat a vytváření klíčových slov pro téma.



Obr. 20 Stromová struktura výsledné nápovědy

Tato nápověda má jednoduchou stromovou strukturu uvozenou několika výchozími položkami – Úvod, Začínáme, První spuštění, Vyčítání parametrů z měniče a Práce se soubory. Celá struktura je vidět na *Obr. 22* z běžící nápovědy. Jednotlivá témata nápovědy jsou doplněna o veliké množství obrázků, znázorňujících jednotlivé funkce programu, jejich umístění a příklady jejich použití včetně vysvětlení některých pravidel.

Protože některé funkce jsou velice podobné jako v předchozí verzi aplikace, nejsou zde do detailu všechny popsány. Na *Obr. 23* je vidět malý náhled otevřeného tématu v této nápovědě.



Obr. 21 Ukázka nápovědy

6 Závěr

Tato diplomová práce řeší reálný požadavek odborné veřejnosti na zrealizování aplikace, která by byla alternativou k již existující aplikaci od firmy Siemens. Výsledná aplikace bude používána pro vyčítání a parametrizaci frekvenčních měničů od této firmy, umožňuje ukládání tabulky parametrů do souboru, kde je lze ručně editovat. Během vývoje byl kladen důraz na uživatelskou přívětivost grafického prostředí a hardwarovou nenáročnost spolu s možností program spustit na operačních systémech Microsoft Windows.

I když se jedná o „alternativu“ k současně používané aplikaci, nebyl z původní aplikace přenesen žádný kód a její vývoj probíhal odděleně. K tomu přispívá i zvolené vývojové prostředí a jazyk programu, jímž je Visual studio 2013, framework .NET 4.5, jazyk C#. Nové prostředí bylo zvoleno také díky předem určené cílového skupině operačních systémů Microsoft Windows 7 a vyšší.

Během postupu práce bylo řešeno několik problémů, včetně prvotního, kterým bylo prostudování USS protokolu, všech jeho parametrů a součástí, dále jeho využití při komunikaci s cílovým zařízením. Po prostudování protokolu práce přešla do fáze návrhu a vývoje funkce pro sestavení telegramu pro vyčítání parametrů. Následně bylo vyčítání jednotlivých parametrů modifikováno na cyklické vyčítání vybraných parametrů z tabulky a umožnění jejich editace. Další vybrané problémy a jejich řešení jsou uvedeny v textu práce. Autor zaznamenal během programování veliký vývoj v oblasti návrhu a realizace funkcí, metod a tříd. Po závěrečné kontrole kódu došlo v mnoha funkcích k několika úpravám, které měli za následek snížení nároků na systém a zvýšení efektivity spolu s úsporou potřebné paměti systému.

Největší přínos této aplikace je shledán v možnosti ukládat a dále editovat tabulku vyčítaných parametrů v programu i v textovém souboru. Oproti aplikaci od Siemensu má výsledná aplikace jednoduché a přehledné uživatelské rozhraní, pomocí kterého je možné parametrizovat měnič Sinamics G110 tak i V20. Aplikace byla testována na měniči Sinamics V20 českým zastoupením Siemensu v České Republice. I díky jejich poznatkům byla aplikace dále formována do současné podoby, a tak splnila všechna počáteční kritéria.

Použitá literatura

[1] SIEMENS A.G. Universal Serial Interface Protocol USS: Specification. 09.94. 1994. E20125-D0001-S302- A1-7600

[2] Siemens A.G. Sinamics G110 120W - 3kW Seznam parametrů, příručka uživatele. Vydání 04/03. 2004

[3] MICROSOFT. 2015. *Developer network: Visual C#* [online]. [cit. 2015-05-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/kx37x362.aspx

[4] BAYER, Jürgen. C# 2005: velká kniha řešení. Přeložil Jiří. KOLÁŘ. Brno: Computer Press, 2007. ISBN 978-80-251-1620-3.

[5] *HelpNDoc* [online]. 2016 [cit. 2016-12-28]. Dostupné z: http://www.helpndoc.com/

[6] Create multiple threads and wait of them to complete. *Stack Overflow* [online]. [cit. 2018-07-18]. Dostupné z: https://stackoverflow.com/questions/4190949/create-multiple-threads-and-wait-all-of-them-to-complete

[7] Sériový port RS232. *Papouch* [online]. [cit. 2018-07-18]. Dostupné z: https://www.papouch.com/cz/website/mainmenu/clanky/jak-na-to/rs232/

[8] SerialPort Class. *DOCS Microsoft* [online]. [cit. 2018-07-18]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/api/system.io.ports.serialport?view=netframework-4.7.2

[9] C# - Pro pokročilé. *IT Network* [online]. [cit. 2018-07-18]. Dostupné z: https://www.itnetwork.cz/csharp/pokrocile

[10] EŠPANDR, Jiří. *Použití USS protokolu pro parametrizaci pohonů Siemens*. Liberec, 2016. Semestrální projekt. Technická univerzita v Liberci. Vedoucí práce Ing. Martin Diblík, Ph.D.