Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Monitor sběrnice Profibus s HW podporou - SW část

Plzeň 2013 Jan Krahulec

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 13. května 2013

Jan Krahulec

Abstract

The Profibus Monitor with the HW Support - the SW Part
This diploma thesis deals with the question of current fieldbuses with an emphasis on the Profibus (especially its PA variant).

The modern fieldbuses are introduced and briefly described in the first part of the thesis. Profibus PA is described more in detail because it is an important part of the practical realization.

The applied hardware is closely described in the next part. Two units are used: FintT410 from the Norwegian company Fieldbus International AS (it implements the interface between protocols Ethernet and Profibus PA) and FNL (Fieldbus Network Link) from German company COMSOFT (it implements the interface between protocols Profibus PA and Modbus).

The very realization of the practical part follows. At first, the communication chain was created from the units mentioned above. At second, the application, which captures data from Profibus and filters this data appropriately, was implemented. The graphical user interface with the option of some parameters configuration is the part of the application.

Obsah

1 Úvod

V současné době jsou průmyslové sběrnice nedílnou součástí systémů pro sběr, distribuci a vyhodnocení dat různého informačního charakteru. S průmyslovými sběrnicemi se můžeme setkat prakticky ve všech technických odvětvích – automobilový průmysl, strojírenství, energetika a mnoho dalších. Oproti starým analogovým rozvodům mají průmyslové sběrnice několik výhod - mezi ty hlavní patří především možnost využití starých rozvodů (ušetření nákladů) a také schopnost přenášet mnohonásobně více dat po jednom vedení. Od počátku 80. let, kdy začal vývoj průmyslových sběrnic, až do dnešní doby vzniklo mnoho průmyslových standardů, které byly vytvářeny s ohledem na prostředí, v němž měly být uplatněny.

Tato diplomová práce se zabývá moderními průmyslovými sběrnicemi, především pak sběrnicí Profibus a její variantou PA.

Prvním cílem této práce je nastudování potřebných informací o sběrnici Profibus PA, především pak základní vlastnosti komunikačního protokolu.

Další fáze této práce spočívá v sestavení a oživení komunikačního řetězce z dostupného hardwaru, jehož hlavní část využívá pro komunikaci sběrnici Profibus PA. Po důkladném zvážení byly jako hlavní součásti řetězce pořízeny jednotky FintT410 od norské společnosti Fieldbus International AS a Fieldbus Network Link od německé společnosti COMSOFT.

Hlavním tématem poslední, ale nejdůležitější části práce je vytvoření aplikace, která dokáže monitorovat data přenášená po sběrnici Profibus PA. Kromě vlastního zachytávání dat je v aplikaci možné nastavovat základní komunikační parametry tak, aby byl výsledný program co nejvíc univerzální, čili aby byl kompatibilní i s jinými zařízeními, než jaká byla použita v této práci. Součástí vytvořené aplikace je také uživatelské rozhraní, jež umožňuje uživateli jednoduchý a intuitivní přístup ke konfiguraci a vlastním přenášeným datům.

2 Průmyslové sběrnice

Průmyslové sběrnice jsou digitální sériové sběrnice, které zajišťují síťovou komunikaci v průmyslových provozech, především pak slouží ke komunikaci mezi inteligentními zařízeními. Začátek vývoje průmyslových sběrnic sahá do raných 80. let, kdy vznikla potřeba nahradit starší analogové rozvody, u kterých byly vysoké nároky na režii. [8]

V následujícím textu budou stručně popsány současné nejpoužívanější sběrnice, v dalším textu jsou pak podrobně popsány sběrnice Modbus a Profibus, neboť jejich podrobnější popis je pro tuto práci zcela zásadní.

Mezi současné nejpoužívanější sběrnice patří:

- **FOUNDATION Fieldbus**: Protokol byl vyvíjen od roku 1970, jako standard byl přijat o 30 let později. Jedná se o číslicovou sběrnici s poloduplexním sériovým přenosem, která byla navržena tak, aby byly zachovány základní výhody analogových systémů s proudovou smyčkou 4 - 20 mA, čili aby bylo možné při instalaci zachovat původní kabeláž. Pro pomalý přenos dat v segmentech H1 je jako médium použita kroucená dvojlinka, přenosová rychlost je 31,25 kb/s. Vysokorychlostní přenos je realizován na bázi rychlého Ethernetu, přenosová rychlost je 100 Mb/s.[8][12]

- **ControlNET:** Sběrnice, která umožňuje uživatelům sdílet data prostřednictvím protokolu CIP (Common Industrial Protocol) - tento protokol využívají též sítě EtherNet/IP, DeviceNet a CompoNet. ControlNet umožňuje realizovat vysokorychlostní přenos časově kritických vstupních a výstupních dat a současně i upload a download programů, přenos nastavení parametrů a zasílání zpráv peer-to-peer jedním jednoduchým nebo zdvojeným (redundantním) fyzikálním médiem. V dnešní době je součástí sdružení ODVA (Open DeviceNet Vendors Association)[9]

- **HART** (Highway Addressable Remote Transducer): Sběrnice HART umožňuje digitální přenos dat po analogové proudové smyčce 4 - 20 mA, čili současný přenos digitálních a analogových dat. Každé zařízení, které implementuje protokol HART může kromě vlastní naměřené hodnoty poskytovat také např. statistické vyhodnocení efektivity výrobního procesu, plánování údržby, dálkovou diagnostiku závad, nastavení a kalibraci přístrojů. Komunikace mezi nadřazenými systémy

a procesními zařízeními je typu master - slave, data jsou přenášena konstantní rychlostí 1200 b/s.[10]

- **CAN** (Controller Area Network): CAN je sériový komunikační protokol, vyvinutý firmou Bosh (původně pro nasazení v automobilech), který umožňuje distribuované řízení systémů v reálném čase s přenosovou rychlostí do 1Mbit/s a vysokým stupněm zabezpečení přenosu proti chybám. Jedná se o protokol typu multi-master, kdy každá stanice může převzít řídící úlohu na sběrnici (komunikace je tak velmi bezpečná, protože po výpadku jedné stanice může převzít řízení jiná). Komunikace mezi zařízeními probíhá pomocí zpráv, které jsou přijímány všemi uzly na sběrnici (neobsahují adresu, pouze identifikátor důležitosti zprávy). Jednotlivé uzly mohou zprávy filtrovat podle identifikátoru.[11]

2.1 Modbus

Protože je protokol `Modbus` implementován v jednotce `T410`, která je součástí systému použitého v této diplomové práci, je určitě na místě uvést základní informace o tomto protokolu.

2.1.1 Obecné informace

`Modbus` je otevřený protokol, který funguje na principu komunikace klient (slave) - server (master). Na sběrnici je vždy přítomen jeden server a jeden nebo více klientů.

Protokol byl vyvinut firmou `Modicon` v roce 1979 pro využití v programovatelných logických automatech (PLC). Jednalo se o otevřený standard, který definoval pouze strukturu přenosu zpráv, fyzickou vrstvu bylo možné dodefinovat v závislosti na podmínkách, v kterých měl být protokol využit.

Mezi hlavní přednosti standardu `Modbus` patří jistě jeho jednoduchost, s čímž je spojená i možnost snadné implementace.[13][14]

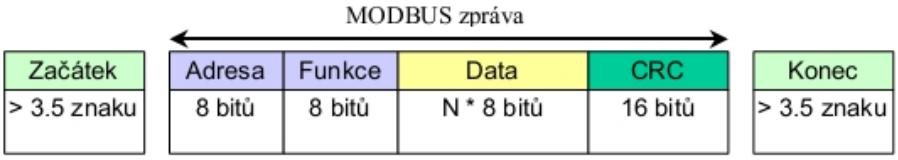
2.1.2 Vysílací režimy

Protokol `Modbus` definuje dva základní vysílací režimy:

- `Modbus RTU`,
- `Modbus ASCII`

Modbus RTU

Pokud je zařízení nastaveno do módu `RTU` (`Remote Terminal Unit`), každých 8 bitů zprávy obsahuje 2 4-bitové znaky. Oproti ASCII módu jsou přenášená data více nahuštěna a tím pádem dochází k přenesení většího objemu dat za stejné rychlosti (baudrate). Na obrázku 2.1 je znázorněna struktura `Modbus RTU` rámce, v tabulce 2.2 jsou pak jednotlivé položky popsány.[13][14]



Obrázek 2.1: Struktura Modbus RTU rámce

Název	Význam
Začátek, Konec	Indikace začátku i konce rámce je v režimu RTU stejná a jde o sekvenci klidu na sběrnici, která musí mít délku minimálně 3,5 znaku.
Adresa	Adresa cílového zařízení (0: Broadcast, 1 - 247: Adresa konkrétního zařízení, 248 - 255: Rezervováno).
Funkce	Kód použité funkce (čtení, zápis, viz dále).
Data	Požadovaná data (v závislosti na použité funkci, mohou být i prázdná).
CRC	Kontrolní CRC součet na bázi polynomu $x^{16}+x^{15}+x^2+1$.

Tabulka 2.1: Struktura Modbus RTU rámce - význam položek

Modbus ASCII

Na rozdíl od módu RTU je osmice bitů zprávy dekódována jako dvojice ASCII znaků. Výhoda tohoto přístupu je v tom, že mezi odvysílanými znaky může být až 1 sekundu dlouhá prodleva. Obrázek 2.2 znázorňuje strukturu rámce, tabulka popis jednotlivých položek.[13][14]

Začátek	Adresa	Funkce	Data	LRC	Konec
znak „:“	2 znaky	2 znaky	0 až 2*252 znaků	2 znaky	2 znaky CR, LF

Obrázek 2.2: Struktura Modbus ASCII rámce

Název	Význam
Začátek, Konec	Začátek je indikován znakem ':', konec pak dvojicí řídicích znaků CR a LF, což umožní delší prodlevy mezi vysílanými znaky.
Adresa	Adresa cílového zařízení (0: Broadcast, 1 - 247: Adresa konkrétního zařízení, 248 - 255: Rezervováno).
Funkce	Kód použité funkce (čtení, zápis, viz dále).
Data	Požadovaná data (v závislosti na použité funkci, mohou být i prázdná).
LRC	Kontrolní LRC (Longitudinal Redundancy Check) součet.

Tabulka 2.2: Struktura Modbus ASCII rámce - význam položek

2.1.3 Typy dat a základní funkce

Každá skupina dat je podle významu zařazena do jedné ze čtyř tabulek, které mají svůj definovaný adresní prostor (prostory se mohou překrývat, ale z důvodu zpětné kompatibility se obvykle volí oddělené prostory, každý o velikosti 10000 položek). V tabulce 2.3 jsou uvedeny názvy tabulek spolu s typickým rozsahem adres a stručným popisem.[13][14]

Tabulka	Rozsah adres	Popis
Cívky (*Coils*)	0 ÷ 9999	Jeden bit (boolean) určený pro čtení i zápis.
Diskrétní vstupy (*Discrete Inputs*)	10000 ÷ 19999	Jeden bit (boolean) určený pouze ke čtení.
Vstupní registry (*Input Registers*)	30000 ÷ 39999	16-bitové registry určené pouze pro čtení. Typické využití je reprezentace analogových vstupních dat.
Uchovávací registry (*Holding Registers*)	40000 ÷ 49999	16-bitové registry určené pouze čtení i zápis.

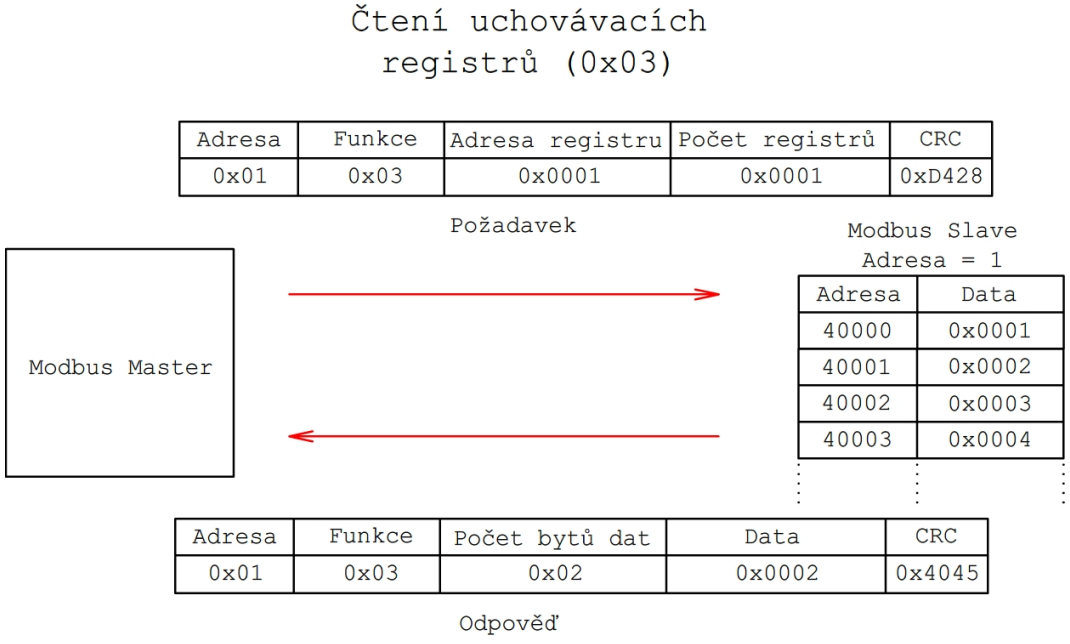
Tabulka 2.3: Modbus - typ dat

Pro čtení a zápis dat jsou definovány základní funkce, které jsou součástí rámce požadavku. Kromě těchto funkcí mají uživatelé možnost také nadefinovat své vlastní funkce. V tabulce 2.4 je uveden výčet všech základních funkcí spolu s jejich kódem a popisem:

Název funkce	Kód	Popis
`Read Coils`	01	Čtení cívek
`Read Discrete Inputs`	02	Čtení diskrétních vstupů
`Read Holding Registers`	03	Čtení uchovávacích registrů
`Read Input Registers`	04	Čtení vstupních registrů
`Write Single Coil`	05	Zápis jedné cívky
`Write Single Register`	06	Zápis jednoho registru
`Write Multiple Coils`	15	Zápis více cívek
`Write Multiple Registers`	16	Zápis více registrů

Tabulka 2.4: Modbus - funkce

Na obrázku 2.3 je pro ilustraci znázorněno přečtení dat funkcí Čtení uchovávacích registrů, jež probíhá na principu požadavek → odpověď:



Obrázek 2.3: Modbus - čtení dat

2.2 Profibus

V předcházející kapitole týkající se Modbusu bylo řečeno, že je tento protokol implementován v jednotce T410, která se chová jako Modbus master. Jednotka T410 ovšem také implementuje protokol Profibus PA, s jehož využitím komunikuje s jednotkou FNL (Fieldbus Network Link).

Na následujících několika stránkách je nejprve obecně popsán protokol Profibus a poté poněkud konkrétněji Profibus PA spolu s obecnými principy cyklické a acyklické komunikace, neboť ty jsou pro tuto práci vpravdě zásadní.

2.2.1 Obecné informace

Na úvod je třeba zmínit, že Profibus (Process Field Bus) není jeden komunikační systém, jedná se spíše o množinu protokolů, které jsou postaveny na stejném základu technologie Fieldbus. Výhodou tohoto přístupu je, že každá konkrétní aplikace může zmíněné protokoly kombinovat na základě svých specifických požadavků.

Historie

Historie Profibusu sahá do konce osmdesátých let minulého století, kdy byla v Německu vysoká poptávka ze strany průmyslových společností a vlády po standardu pro průmyslovou automatizaci. V roce 1987 tak spojilo 21 německých společností a institucí síly, vznikla skupina Centrální asociace pro elektrotechnický průmysl (ZVEI) a výsledkem jejich snahy bylo vytvoření standardu Profibus FMS (Fieldbus Message Specification). V roce 1993 představila skupina ZVEI nový standard, Profibus DP (Decentralized Periphery). Hlavní vylepšení nové verze byla větší jednoduchost, možnosti snadnější konfigurace a rychlejší komunikace.

Standardy Profibusu jsou udržovány (a zlepšovány) několika organizacemi. V roce 1989 založili uživatelé skupinu PROFIBUS User Organization (PNO). Cílem této nekomerční organizace je především podpora a vzdělávání, což zahrnuje například uvolňování technické dokumentace tak, aby uživatelé mohli využívat stávající technologii pro své specifické potřeby. Další organizace vznikla v roce 1995 pod názvem PROFIBUS International (PI),

která je nyní největší Fieldbus asociací na světě. Cíle PI jsou podobné jako u PNO, zabývá se ovšem také ověřováním kvality a tvorbou nových `Profibus` technologií.[1]

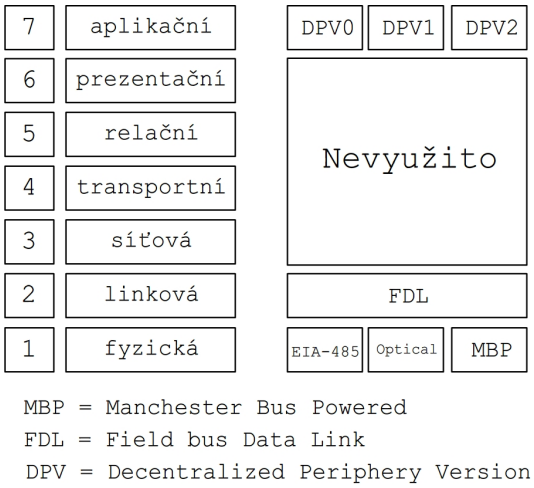
Verze protokolu

V současné době existují 3 varianty protokolu Profibus[2]:

- **Profibus-DP pro automatizaci výroby:**
  - přenosová rychlost až 12 Mb/s,
  - linka RS-485 nebo optické vlákno, s určením pro rychlou výměnu relativně malých objemů dat
- **Profibus-FMS:**
  - přenosová rychlost do 500 kb/s,
  - linka RS-485 nebo optické vlákno, s určením pro větší objemy dat – v hierarchii řízení představuje úroveň nad Profibus-DP.
- **Profibus-PA:**
  - konstantní přenosová rychlost 31,25 kb/s,
  - fyzická vrstva podle IEC 61158 (napět'ový režim, jiskrová bezpečnost).

Architektura

Norma `Profibus` vychází z modelu `ISO/OSI` (viz obrázek 2.4) s tím, že z důvodu časové optimalizace definuje pouze 3 vrstvy - fyzickou, linkovou a aplikační.

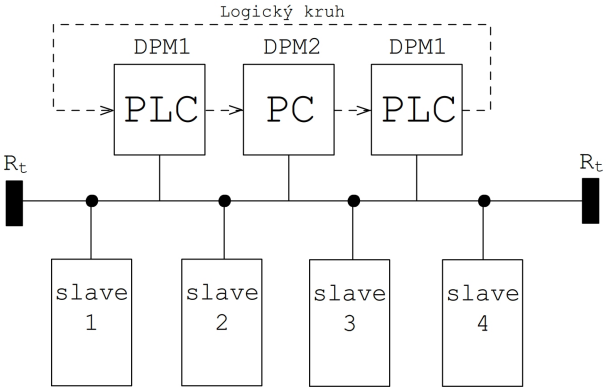


Obrázek 2.4: Profibus - ISO/OSI model

Fyzická vrstva se liší podle použité varianty protokolu `Profibus`[3]:

1. **RS-485(H2)** - Profibus DP/FMS,
2. **Optické vlákno** - Profibus DP/FMS,
3. **EC 1158-2 (H1)** - Profibus PA.

Linková vrstva, jež je označována jako FDL (Fieldbus Data Link) je u všech variant téměř shodná. Přístup na sběrnici je hybridní, jinými slovy kombinuje klasické uspořádání master→slave s distribuovaným řízením přístupu prostřednictvím předávání pověření (viz obrázek 2.5).



Obrázek 2.5: Profibus - přístup na sběrnici

2.2.2 Profibus PA

`Profibus PA` je ze zmiňovaných variant nejmladším standardem (v roce 1996 byl začleněn do evropského standardu `EN 50170 Volume 2`[4]), ve větší míře se začal používat až v posledních letech. V současné době je aktuální verze 3.02. Mezi nejzajímavější realizované aplikace využívající `Profibus PA` patří:

- Automatizace chemičky Wacker Chemie v Kolíně nad Rýnem,
- Automatizace skladovacích tanků firmy Shell v Hamburgu,
- Aplikace měření hustoty pohonných hmot na letišti v Ostravě.

Fyzická vrstva

Fyzická vrstva protokolu `Profibus PA` je navržena v souladu s normou `IEC 1158-2` a má tyto vlastnosti:

1. Stíněná nebo nestíněná kroucená dvojlinka zakončená sériovým RC článkem ($R_t = 100\Omega, C_t = 1\mu F$),
2. Napájení stanic přímo ze sběrnice. Napájecí napětí v rozmezí 9V – 32V (v prostředí s nebezpečím výbuchu maximálně 15 V),
3. Volitelná jiskrová bezpečnost – vysílající stanice má pasivní charakter, jinými slovy nedodává na sběrnici proud, pouze mění svůj odběr,

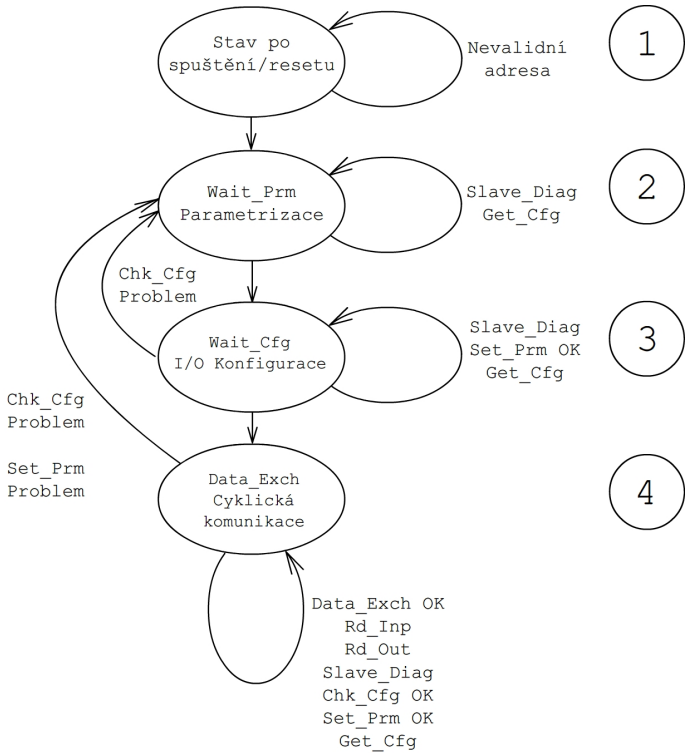
4. Synchronní přenos s konstantní rychlostí 31,25 kbit/s, kódování Manchester,
5. Délka segmentu může být maximálně 1900 m, na jeden segment je možné připojit nanejvýš 32 stanic (v případě jiskrové bezpečnosti nejvýše 10 stanic)

V praxi většinou nebývá `Profibus PA` nasazován samostatně, ale jako prodloužení `Profibusu DP` do míst, kde hrozí nebezpečí výbuchu. Existují 2 možnosti jejich propojení:

1. **Vazební člen (DP/PA Coupler)** – provádí pouze převod mezi odlišnými fyzickými vrstvami a převádí data přenášená asynchronně rychlostí 45,45 kbit/s na data přenášená synchronně rychlostí 31,25 kbit/s. Nevýhodou je skutečnost, že komunikace na sběrnici DP je zpomalena.
2. **Spojový člen (DP/PA Link)** – vůči zařízením na sběrnici DP se chová jako podřízená stanice (slave), vůči zařízením na PA segmentu pak jako řídící stanice (master).

Cyklická komunikace

Cyklická komunikace je u protokolu `Profibus PA` (stejně tak i DP) realizována mezi řídící stanicí třídy 1 (DPM1) a řízeným zařízením. Před tím, než začne cyklická výměna dat (`Data Exchange`), je třeba provést parametrizaci a konfiguraci řízené stanice. Stanice při těchto činnostech prochází několika stavy, což znázorňuje diagram na obrázku 2.6:



Obrázek 2.6: Profibus - stavový diagram

1. **Stav po zapnutí nebo resetu zařízení** - pro přechod do dalšího stavu je nutné, aby měla stanice validní adresu (0-125), kterou nastavuje Master telegramem `Set_Slave_Add`.

2. **Stav Parametrizace** - v tomto stavu čeká zařízení na jeho parametrizaci, kterou provede Master telegramem `Set_Prm`. Ostatní telegramy, kromě dotazu na diagnostiku (`Slave_Diag`) a konfiguraci (`Get_Cfg`) jsou ignorovány.

3. **Stav Konfigurace** - po parametrizaci je nutné specifikovat počet vstupních a výstupních bytů datových telegramů, které budou součástí datové výměny mezi řídící a řízenou stanicí. Pro konfiguraci slouží telegram `Chk_Cfg`.

4. **Cyklická komunikace** - v tomto okamžiku je již možné zahájit cyklickou komunikaci mezi stanicemi. Komunikace je velmi jednoduchá, na principu požadavek-odpověď, kdy řídící stanice pošle řízené stanici telegram `Data_Exch.req` se svými výstupními daty a řízená stanice odpoví telegramem `Data_Exch.resp`, jenž obsahuje její výstupní data. Tento

způsob komunikace se nazývá `Send and Request Data with Reply (SRD)`. Na obrázku 2.7 je stručně znázorněno, jak vypadá cyklická komunikace včetně formátu telegramu.


```
SYN: Synchronizační čas
SD2: Oddělovač typu rámce, hodnota určuje typ telegramu
LE:  Počet přenášených bytů
LEr: Opakování hodnoty LE
DA:  Cílová adresa
SA:  Zdrojová adresa
FC:  Kód funkce (požadavek, odpověď, chyba, atd.)
DU:  Přenášená data (1-246 bytů)
CRC: 16-bitový CRC kód
ED:  Koncový oddělovač, vždy 0x16
```

Obrázek 2.7: Profibus PA - přenos dat

Acyklická komunikace

Acyklická komunikace patří mezi rozšířené funkce DPV1. Tento typ komunikace se dále dělí na:

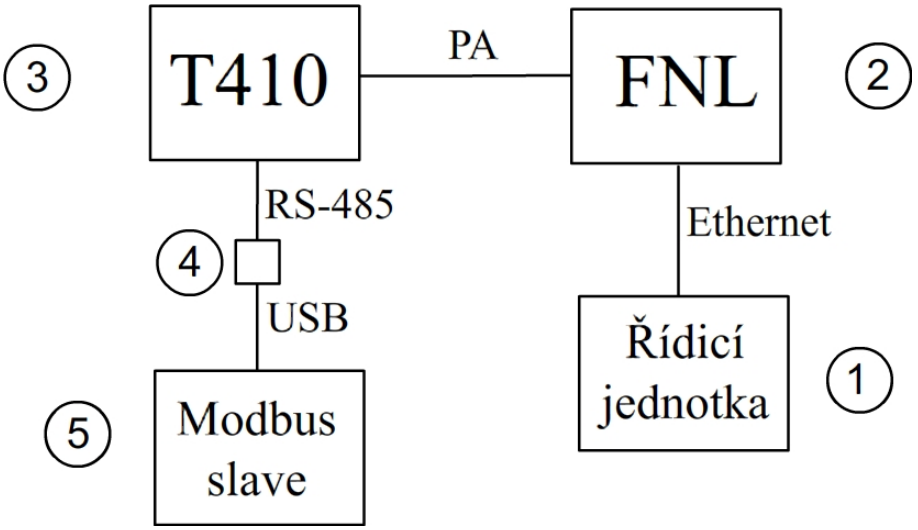
- **MSAC1 (Master/Slave Acyclic C1)** - probíhá mezi řízenou stanicí a DPM1 (DP master class 1), což je řídící stanice sloužící k cyklické výměně dat,
- **MSAC2 (Master/Slave Acyclic C2)** - probíhá mezi řízenou stanicí a DPM2 (DP master class 2), což je řídící stanice realizující diagnostické a monitorovací funkce.

Pro zahájení acyklické komunikace je nejprve nutné vytvořit a otevřít acyklický komunikační kanál (v případě vytvoření cyklického spojení je vytvářen automaticky). Parametry, které mají být přenášeny acyklickou komunikací, jsou identifikovány jednoznačnou dvojicí slot a index. Podrobnější informace o čtení parametrů je možné nalézt dále v textu v sekci 4.3.5

Další informace o protokolu `Profibus PA` lze dohledat ve zdrojích [2], [4], [5] a [6].

3 Hardwarové vybavení

V této kapitole bude nejprve obecně popsáno, jak celý hardwarový systém, který byl pro účely diplomové práce sestaven, funguje a následně budou důkladněji popsány jeho jednotlivé součásti. Na obrázku 3.1 je znázorněno stručné schéma použitého systému:

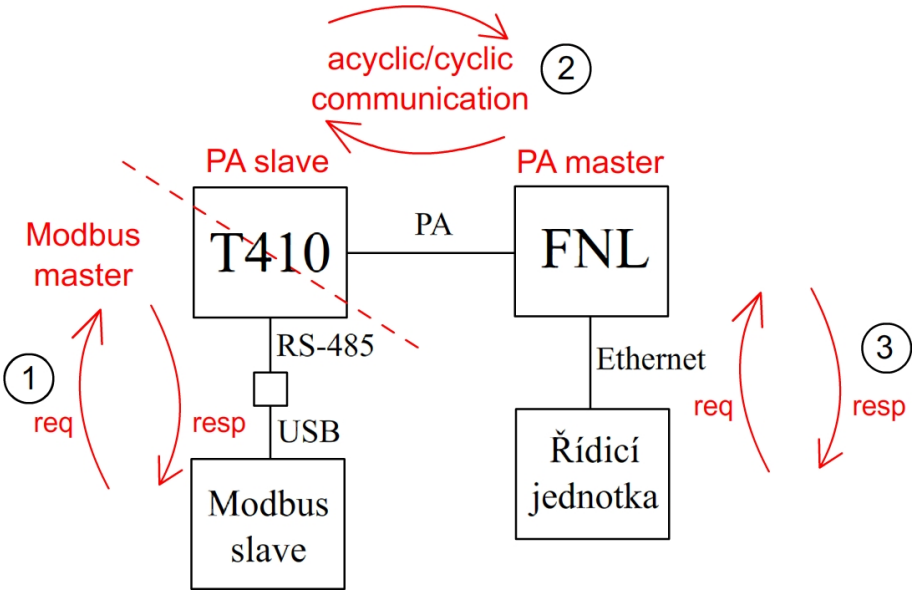


Obrázek 3.1: Schema systému

1. **Řídící jednotka** – slouží ke konfiguraci komunikace, zachytává a ukládá data přečtená z jednotky č. 5 (`Modbus slave`). V této práci je jako řídící jednotka použit stolní počítač,
2. **FNL (Fieldbus Network Link)** – jednotka pracuje jako rozhraní mezi protokolem `Profibus` a protokolem `Ethernet`. Uživateli jsou k dispozici rozličné funkce, které je možné z řídící jednotky volat,
3. **FintPROFI T410** – jednotka sloužící jako rozhraní mezí protokolem `Profibus` a `Modbus`,
4. **Převodník RS485/USB**,
5. **Modbus slave** – simulátor `Modbus` slave běžící na stolním počítači (použita shareware verze programu MOD_RSsim)

3.1 Popis komunikace mezi moduly

Pro úplnou představu je třeba zmínit, jak celé zařízení funguje, čili jak jednotlivá zařízení mezi sebou komunikují. Blokové schema komunikace je znázorněno na obrázku 3.2:



Obrázek 3.2: Schema komunikace

1. Na straně `Modbus` slave jsou v tzv. uchovávacích registrech (Read Holding Registers) uložena požadovaná data. Jednotka `T410`, která v této komunikaci funguje jako `Modbus` slave, periodicky čte a ukládá data,

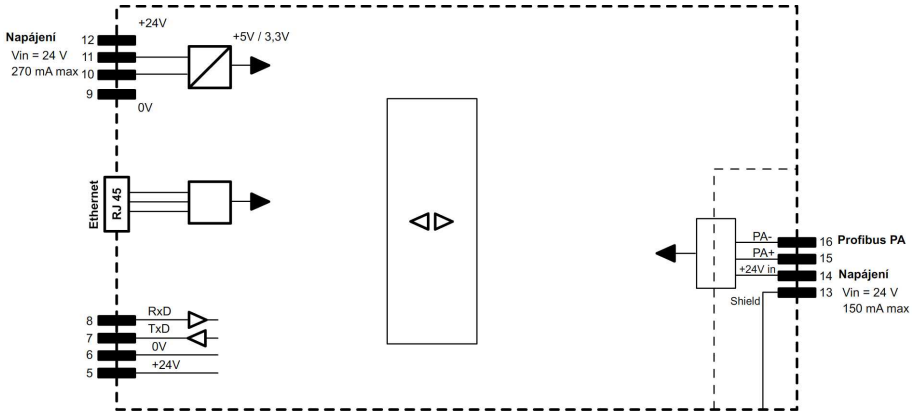
2. Aby výše zmíněná komunikace probíhala podle představ uživatele, je nutné ji nějakým způsobem parametrizovat (adresa `Modbus` slave, timeout mezi jednotlivými dotazy, atd.). Kýžené parametry jsou přenášeny acyklickou komunikací mezi jednotkou `FNL` a `T410`. Dále je nutné data z jednotky `T410` nahrát do paměti jednotky `FNL`, k čemuž slouží tzv. cyklická komunikace (data je ovšem možné přečíst též acyklicky),

3. Komunikace mezi řídící jednotkou a `FNL` je vpravdě zásadní, neboť slouží ke konfiguraci veškeré komunikace, tj. nastavení parametrů sběrnice `Profibus`, jednotky `T410`, atd. a také k řízení komunikace mezi `FNL` a jednotkou `T410` (pro tyto účely je k dispozici obsáhlá `API`).

V dalších kapitolách budou podrobněji popsány obě jednotky – `FNL` a `Fint-PROFI T410`.

3.2 Fieldbus Network Link (FNL)

`FNL` je zařízení, které umožňuje propojení sítí typu ethernet (10/100 Mbit/s) a `Profibus`, v tomto případě verze PA (existuje ovšem také verze DP). Blokové schéma této jednotky, obsahující informace o napájení a použitých rozhraních, je zobrazeno na následujícím obrázku (viz obr. 3.3).



Obrázek 3.3: Schema jednotky FNL

Jednotka `FNL` je nejdůležitější součástí systému z hlediska programátora, neboť je v ní implementováno uživatelské rozhraní, přes které je možné volat různé služby sloužící k parametrizaci sítě a přenosu dat do počítače. Nutno podotknout, že komunikace mezi počítačem (či jinou řídící jednotkou) probíhá přes protokol `TCP/IP` (jednotka má přidělenu statickou IP adresu, v tomto případě `192.168.1.75`)

Uživatelské rozhraní je navrženo jako `Dual-Port RAM (DPR)`, což je, jak již název napovídá, typ paměti, který umožňuje v jeden okamžik zapisovat nebo číst data z obou portů.[15]

3.2.1 Rozhraní Dual-Port RAM

Rozhraní lze rozdělit na dvě samostatné části:

1. **Command Interface** - přenos funkcí nebo služeb s požadovanými parametry,
2. **Process Data Image** – obsahuje uživatelská data a stavové informace o jednotlivých zařízení.

Struktura DPE je znázorněna v tabulce 3.4:

DPR-Interface	Command interface					Process data image		
DPR-Offest	0	0x100	0x200	0x300	0x3FE	0x400	0x480	0x484
Byte number	256	256	256	256	2	128	4	n
	Send-Box 1	Receive-Box 1	Receive-Box 2	Send-Box 2	Control-Flags	SlaveStatus[126]	Semaphore: UserLockInputs: 0x480 FWLockInputs: 0x481 UserLockOutputs: 0x482 FWLockOutputs: 0x483	DP-Inputs/Outputs: DPData
	DPE firmware							
	MSAC_C2-links Master-Master links					Class -1-Master- and Slave kernel		

Obrázek 3.4: Struktura DPE

Command Interface

Jak již bylo zmíněno, `Command Interface` je rozhraní, které vzájemně propojuje firmware `FNL` (jež implementuje `Profibus PA` profil) s uživatelem. Pro zvýšení rychlosti přenášení dat jsou k dispozici dva paměťové prostory pro odesílání (Send-Box 1 a 2) a příjem (Receive-Box 1 a 2) dat.[15]

Dostupné služby lze rozdělit do několika skupin jako příkazy pro:

- lokální inicializaci a konfiguraci firmware,
- acyklickou komunikaci,
- cyklickou komunikaci,

- komunikaci typu Master-Master,
- příjem lokálních indikací.

Příkazy lze dále dělit na potvrzované resp. nepotvrzované a příkazy inicializované uživatelem (odesílány jako požadavek na firmware) a inicializované firmwarem (odesílány uživateli jako indikace).

Příkazy jsou odesílány a přijímány v tzv. mailboxech. Každý mailbox se skládá z hlavičky (mailbox header) a těla příkazu (mailbox-body), viz tabulka 3.1:

	Byte	Vstup
Mailbox header	0	Service
	1	Group
	2	Result
	3	CommandID
Mailbox body	4 ... n	...

Tabulka 3.1: Mailbox - struktura

Service
Hodnota představuje kód volané služby, např. `0x0A` pro funkci `load_bus_par`

Group
Veškeré volané služby lze rozdělit celkem do čtyř skupin:

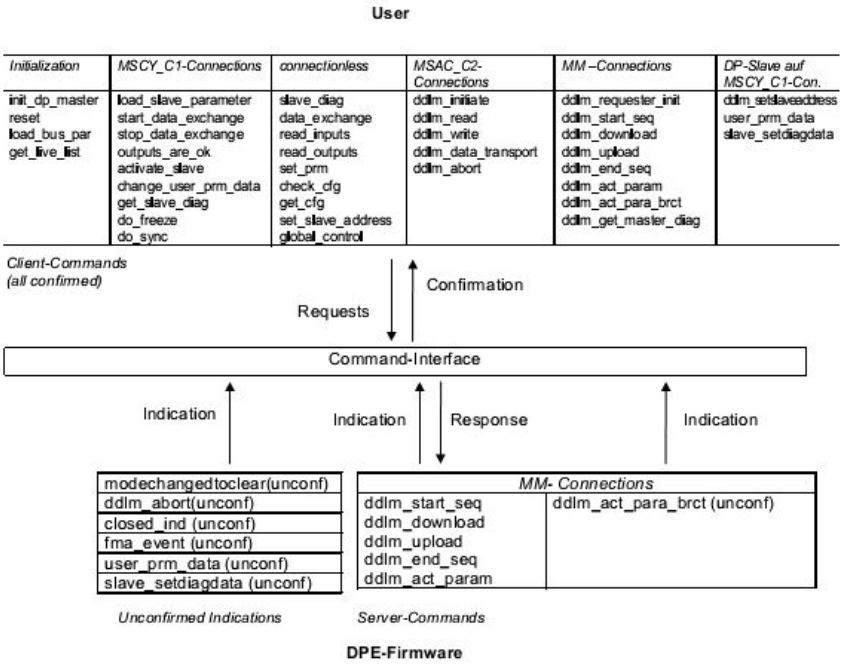
- inicializační (`0x04`)
  - např. `restart_fdl`
- komunikační (`0x08`)
  - např. `init_dp_master`
- lokální (`0x07`)
  - např. `get_live_list` (jediná služba v této skupině)
- asynchronní události vyvolané firmwarem (`0x42`)

Poslední bit tohoto parametru (MSB) určuje směr odesílání požadavku:

- 0: směr uživatel → firmware
- 1: směr firmware → uživatel

Result
Výsledek volání služby uložený firmwarem po zpracování požadavku. V případě odpovědi na požadavek obsahuje informaci, zda volání funkce proběhlo bez problémů nebo došlo k nějaké chybě. V případě, že firmware odešle indikaci nějaké události, obsahuje kód tzv. `status message` (viz příloha, kde jsou všechny tyto zprávy popsány)

Command ID
Tento parametr není pro firmware důležitý, tudíž je z hlediska uživatele nepodstatný. Je možné do něj uložit např. pořadí volání služeb, čímž se program stane přehlednějším. Při potvrzování má parametr stejnou hodnotu jako při volání funkce.



Obrázek 3.5: Přehled komunikace v rámci FNL

Obrázek 3.5 přehledně ilustruje, jak probíhá komunikace mezi uživatelem, Command Interface a firmwarem.

Process Data Image

Druhá část DPR se nazývá Process Data Image a obsahuje:

- **Slave status** - pole o velikosti 126 bytů, kde každý byte odpovídá jednomu koncovému zařízení (slave),
- **Semafory** - semafory jsou nezbytné pro zachování konzistence vstupních resp. výstupních dat. Rozlišují se čtyři druhy semaforů:
  - `UserLockInputs` – uzamčení přístupu k vstupním datům uživatelem,
  - `FWLockInputs` – uzamčení přístupu k vstupním datům firmwarem,
  - `UserLockOutputs` – uzamčení přístupu k výstupním datům uživatelem,
  - `FWLockOutputs` – uzamčení přístupu k výstupním datům firmwarem.
- **Uživatelská data** - data, která jsou přenášena během cyklické komunikace (`MSCY_C1`). Data jsou ukládána od adresy `0x484` s tím, že jejich přesné umístění je specifikováno offsety, jež jsou uživatelem zvoleny při volání služby `load_slave_parameters`.

V dalších částech této kapitoly nebudou popisovány jednotlivé služby, neboť vysvětlení části s nich i s konkrétními parametry je součástí praktické části této diplomové práce. Úplný seznam všech služeb včetně jejich kódů lze nalézt v [15].

3.2.2 Knihovní funkce

Pro programátora je vskutku zásadní otázka, jakým způsobem se připojit k jednotce `FNL`, jak odesílat resp. přijímat vybrané služby, atd. Pro tyto účely dává výrobce k dispozici knihovnu `TMGnet.dll`. S pomocí funkcí z této

knihovny je možné odesílat příkazy `Command Interface` a přistupovat k `Process Data Image`. V tabulce 3.2 jsou uvedeny všechny funkce spolu se stručným popisem jejich činnosti, v dalším textu jsou pak tyto funkce popsány poněkud podrobněji.[15]

Název	Popis činnosti
`ConnectBoard`	Připojení k jednotce
`ResetBoard`	Reset jednotky
`DisconnectBoard`	Odpojení od jednotky
`GetRequestBlock a SendRequestBlock`	Odeslání příkazu jednotce
`GetReceiveBlock a FreeReceiveBlock`	Příjem dat z jednotky
`PutProcessData`	Uložení dat do `Process Data Image`
`GetProcessData`	Čtení dat z `Process Data Image`
`TransferProcessData`	Uložení/čtení dat do/z `Process Data Image`
`GetTMGErrorMessage`	Textová podoba chybových kódů

Tabulka 3.2: Knihovní funkce - stručný popis

ConnectBoard

Prototyp

```
short USER_FUNCTION ConnectBoard (const char *BoardName, short ProtocolID);
```

Parametry

`char *BoardName`	:	Jméno přidělené jednotce `FNL` (jméno je zaregistrováno v aplikaci `TMG Net Series`)
`short ProtocolID`	:	Indentifikátor použitého protokolu

Popis

Funkce `ConnectBoard` slouží k připojení aplikace k `FNL` a musí být vždy volána jako první. Zároveň je po úspěšném připojení zaručeno, že žádná další aplikace nebude mít k jednotce přístup (funkce vrátí chybový status `CONNECT_BOARDINUSE`). Aplikace zůstává připojena do té doby, než je zavolána funkce `DisconnectBoard` (viz dále) nebo dojde k ukončení aplikace (DLL tuto událost rozpozná a automaticky se odpojí).

Návratové hodnoty

`SUCCESS (0x00)`	:	Aplikace byla úspěšně připojena k `FNL`
`CONNECT_BOARDINUSE (0x01)`	:	Jednotka je používána jinou aplikací
`CONNECT_UNKNOWNBOARDNAME (0x02)`	:	Jméno registrované aplikací TMG Net Series nesouhlasí se jménem zadaným jako parametr funkce ConnectBoard
`PROTOCOL_NOTSUPPORTED (0x0A)`	:	Jednotka nepodporuje zadaný typ protokolu
`LOAD_BOARDNOTFOUND (0x14)`	:	Jednotka neexistuje (pravděpodobně nebyla korektně zaregistrována v systému)
`LOAD_TIMEOUT (0x15)`	:	Při nahrávání firmwaru došlo k vypršení časového limitu
`LOAD_FILENOTFOUND (0x16)`	:	Firmware neexistuje
`LOAD_FILECORRUPT (0x17)`	:	Firmware je poškozený
`APP_ALREADYCONNECTED (0x1F)`	:	Aplikace je již připojena k jinému zařízení
`SYSTEM_DLLBLOCKED (0x28)`	:	DLL knihovna je zablokována některým z uživatelů
`SYSTEM_DEVICEDRIVERFAILED (0x29)`	:	Řadič jednotky selhal
`SYSTEM_REGISTRYCORRUPTED (0x2A)`	:	Poškozené systémové registry

ResetBoard

Prototyp

```
short USER_FUNCTION ResetBoard (void);
```

Parametry

Bez parametrů.

Popis

Vyresetování firmwaru jednotky `FNL`.

Návratové hodnoty

`SUCCESS (0x00)`	:	Aplikace byla úspěšně připojena k `FNL`
`LOAD_BOARDNOTFOUND (0x14)`	:	Jednotka neexistuje (pravděpodobně nebyla korektně zaregistrována v systému)
`LOAD_TIMEOUT (0x15)`	:	Při nahrávání firmwaru došlo k vypršení časového limitu
`LOAD_FILECORRUPT (0x17)`	:	Firmware je poškozený
`APPL_NOT_CONNECTED (0x1E)`	:	Aplikace není připojena k jednotce `FNL`

DisconnectBoard

Prototyp

```
short USER_FUNCTION DisconnectBoard(void);
```

Parametry

Bez parametrů.

Popis

Odpojení aplikace od jednotky `FNL`.

Návratové hodnoty

`SUCCESS (0x00)`	:	Aplikace byla úspěšně odpojena od `FNL`
`APPL_NOT_CONNECTED (0x1E)`	:	Aplikace není připojena k jednotce `FNL`

GetRequestBlock a SendRequestBlock

S využitím těchto funkcí je aplikace schopna odeslat příkaz firmwaru přes Command Interface. Příkazy jsou přenášeny ve třech krocích:

1. V prvním kroku je nutné získat ukazatel na prázdný mailbox, čehož se dosáhne (v případě, že je nějaký volný) zavoláním funkce:

   ```
   short USER_FUNCTION GetRequestBlock (PTCommand *PSendBox);
   ```

   kde `PTCommand *PSendBox` je ukazatel na strukturu, jež bude po úspěšném dokončení funkce `GetRequestBlock` k dispozici pro uložení odpovídajících parametrů

2. V případě, že funkce nevrátila nulový ukazatel, je struktura naplněna daty, což je ukázáno v následující ukázce funkce `get_slave_diag`:

   ```
   PTCommand m;
   m->GetSlaveDiag.Header.Service = 0x09;
   m->GetSlaveDiag.Header.Group = 0x08;
   m->GetSlaveDiag.SlaveAddress = 2;
   ```

3. V posledním kroku dochází k odeslání dat `Command Interface` funkcí:

   ```
   short USER_FUNCTION SendRequestBlock(PTCommand SendBox);
   ```

kde `PTCommand SendBox` je struktura (jež odpovídá struktuře mailboxu, viz výše) obsahující konkrétní parametry.

GetReceiveBlock a FreeReceiveBlock

Tyto dvě funkce, na rozdíl od předešlých dvou, slouží k získání dat, které dává k dispozici firmware. Operace je opět provedena ve třech krocích:

1. Pro získání ukazatele na mailbox naplněný firmwarem je třeba zavolat funkci:

   ```
   short USER_FUNCTION GetReceiveBlock(PTCommand *PreceiveBox);
   ```

   kde `PTCommand *PreceiveBox` je ukazatel na strukturu, jež bude po úspěšném dokončení funkce `GetReceiveBlock` naplněna daty.

2. Dekódování přijatých dat, jinými slovy podle hlavičky, která obsahuje kód skupiny a funkce, poznáme, o jaký příkaz se jedná:

   ```
   short result;
      if (pm->GetSlaveDiag.Header.Group = CM_DP &&
            pm->GetSlaveDiag.Header.Service = GET_SLAVE_DIAG) {
               result = pm->GetSlaveDiag.Header.Result;
       }
   ```

3. Na závěr je třeba navrátit přijatý mailbox zpátky firmwaru funkcí:

   ```
   short USER_FUNCTION FreeReceiveBlock(void);
   ```

PutProcessData

Prototyp

```
short USER_FUNCTION PutProcessData (short DataOffset, short DataLen, BYTE *Pro-
cessDataPtr, short UserLock );
```

Parametry

Popis

Funkce sloužící k uložení dat (`Output Data`) do `Process Data Image`.

`short DataOffset`	:	Offset adresy pro uložení dat (0x484 + DataOffset)
`short DataLen`	:	Délka přenášených dat (v bytech)
`BYTE *ProcessDataPtr`	:	Ukazatel na data k přenesení
`short UserLock`	:	Uzamčení přístupu k výstupním datům v Process Data Image

GetProcessData

Prototyp

```
short USER_FUNCTION GetProcessData (short DataOffset, short DataLen, BYTE *ProcessDataPtr, short UserLock);
```

Parametry

`short DataOffset`	:	Offset adresy pro uložení dat (`0x484 + DataOffset`)
`short DataLen`	:	Délka přenášených dat (v bytech)
`BYTE *ProcessDataPtr`	:	Ukazatel na místo v paměti, kam se mají uložit data z `Process Data Image`
`short UserLock`	:	Uzamčení přístupu k vstupním datům v `Process Data Image`

Popis

Funkce sloužící k přečtení dat (`Input Data`) z `Process Data Image`.

TransferProcessData

Prototyp

```
short USER_FUNCTION TransferProcessData (TPDTransferPrm* PDPrm);
```

Parametry

`TPDTransferPrm* PDPrm` – data, která mají následující strukturu:

```
typedef {
   short DataOffset;
   short DataLen;
   BYTE *ProcessDataPtr;
   short UserLock;
} TPDPrm;

typedef struct {
   short SlaveAdd;
   short SlaveCount;
   BYTE *Pstatus;
   TPDPrm Write;
   TPDPrm Read;
 TPDTransferPrm;
```

Popis

Funkce, která de facto kombinuje funkce `GetProcessData` a `PutProcessData` (struktura `TPDPrm` odpovídá parametrům těchto funkcí), navíc je také přijat Status stanice specifikované položkou `SlaveAdd`.

GetTMGErrorMessage

Prototyp

```
const char * USER_FUNCTION GetTMGErrorMessage( short ErrorCode, short Lang-
uage);
```

Parametry

`short ErrorCode`	:	Kód návratové hodnoty některé z výše zmíněných funkcí.
`short Language`	:	Lokalizace (0 – němčina, 1 – angličtina)

Popis

Funkce vracející textovou podobu chyb, které jsou reprezentovány kódem `ErrorCode`.

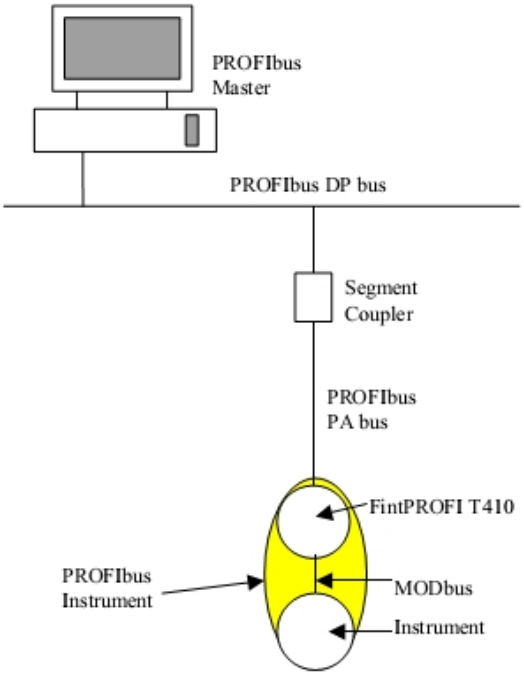
Další informace o jednotce `FNL` je možné nalézt v [15].

3.3 FintPROFI T410

Druhý modul, jenž byl použit pro vytvoření komunikačního řetězce, je jednotka `T410` od norské společnosti Fint (Fieldbus International AS).

V modulu jsou implementovány dva protokoly - `Modbus` a `Profibus PA`, jedná se tedy o jakési rozhraní mezi těmito protokoly. Na sběrnici se zařízení chová na straně protokolu `Modbus` jako master, na straně protokolu `Profibus PA` jako slave.

Na obrázku 3.6 je znázorněno, jak vypadá zapojení jednotky `T410` do komunikačního řetězce.



Obrázek 3.6: T410 - zapojení do komunikačního řetězce

3.3.1 Popis

Každé `Profibus PA` zařízení obsahuje bloky, které obsahují různé parametry. Jedná se o bloky:

- **fyzický (Physical, PB)** - zastřešující blok,
- **převodníkový (Transducer, TB)** - zajišťuje kontakt s měřicím čidlem, provádí například linearizaci,
- **funkční (Function, FB)** - zprostředkovává naměřenou hodnotu (získána z převodníkového bloku) na připojenou sběrnici, při překročení nějakého limitu generuje alarmy.

Jednotka `T410` obsahuje 1 fyzický blok, 4 funkční a 4 převodníkové bloky.

V převodníkovém bloku jsou v případě jednotky `T410` kromě parametrů daných standardem také parametry, kterými se nastavuje chování komunikace `Modbusu`. Jedná se o tyto parametry:

Parametr	Význam
PV Scaling	Přečtenou hodnotu (PV = Primary Variable, dynamická proměnná, hodnota čteného `Modbus` registru) je možné uložit s ohledem na počet desetinných míst, např. pro uložení na jedno desetinné místo bude hodnota parametru 0.1.
PV Unit	Jednotka přečtené hodnoty.
PV's Modbus register	Ukazatel na `Modbus` registr, z kterého má být PV čtena.
PV type	Datový typ proměnné, např. pro Float s bytovým pořadím 1 2 3 4 je hodnota parametru 3.
Periodic Measurement	Perioda čtení PV, udává se v sekundách.
Modbus address	Adresa `Modbus` zařízení.
Modbus inbound data package	Prostor, do něhož jsou zaznamenávány hodnoty z `Modbus` registrů (získané acyklickým čtením).
Modbus inbound data package register address	Ukazatel na první `Modbus` registr pro acyklické čtení.
No of registers for inbound data	Počet `Modbus` registrů, které se mají přečíst při acyklickém čtení.
Modbus outbound data package	Prostor, přes který je možné acyklicky zapsat data do `Modbus` zařízení.
Modbus outbound data package register address	Ukazatel na první `Modbus` registr pro acyklický zápis.
No of registers for outbound data	Počet `Modbus` registrů, do nichž se mají zapsat data při acyklickém zápisu.

Tabulka 3.3: Parametry převodníkového bloku

Ve fyzickém bloku je pak možné nastavovat tyto parametry:

- Počet stop bitů,
- Baudrate,
- Pořadí bytů v CRC součtu,
- Paritu.

Další informace o jednotce `T410` je možné nalézt v [7].

4 Realizace monitoru sběrnice Profibus PA

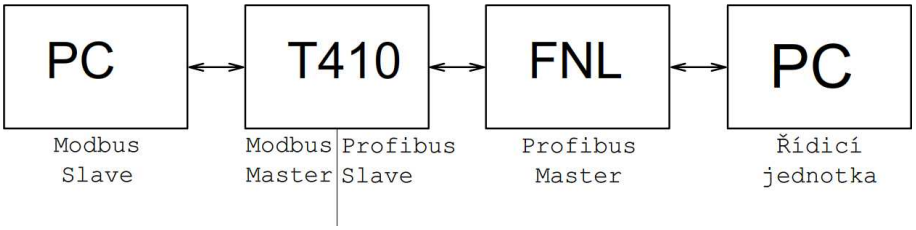
4.1 Úvod

Po získání všech potřebných informací o hardwarovém vybavení a použitých protokolech bylo dalším krokem vytvoření návrhu monitoru přenášených dat a jeho následná realizace. Na začátku je jistě na místě zmínit, jaké prostředky byly pro aplikaci použity. Pro naprogramování vlastní logiky aplikace (komunikace s moduly) byl použit jazyk `ANSI C` a to hlavně z toho důvodu, že volání knihovních funkcí je tomto jazyce poměrně jednoduchá záležitost. Další součástí aplikace je grafické uživatelské rozhraní (GUI) – pro jeho vytvoření byla využita knihovna `QT` verze 4.8.4. (potřebné informace byly čerpány z [16]) Aplikace byla vyvíjena v prostředí `NetBeans` verze 7.1.2.

Na dalších řádcích bude postupně popsáno, jakým způsobem aplikace funguje. Zvláštní důraz pak bude kladen na čtení parametrů z jednotlivých bloků jednotky `T410`, neboť bylo této problematice věnováno značné množství času a jedná se bez nadsázky o základní kámen vytvořené aplikace. Zdrojové soubory jsou k dispozici na přiloženém CD této práce.

4.2 Návrh aplikace

V kapitole 3 byl popsán systém, na jehož začátku je počítač (či jiné zařízení) sloužící jako řídící jednotka a na jehož konci je zařízení, které na sběrnici funguje jako `Modbus` Slave. Na obrázku 4.1 je pro připomenutí tento systém znovu znázorněn:

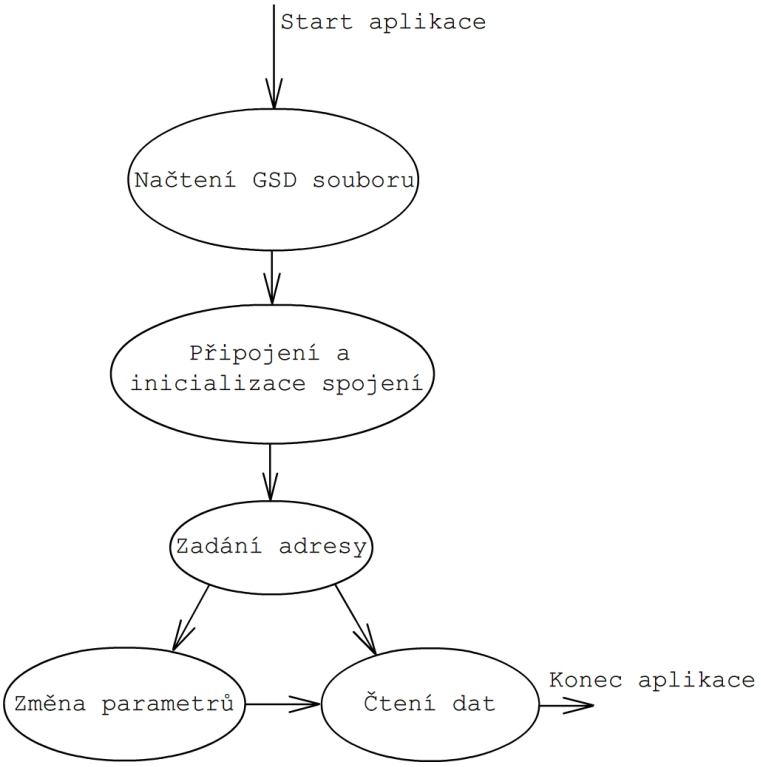


Obrázek 4.1: Schema systému

Z obrázku 4.1 je patrná hlavní funkce vytvářené aplikace – řídící jednotka (na níž monitor běží) by měla umět číst a interpretovat data, které odesílá `Modbus` Slave. Další věc, která z obrázku také vyplývá, je to, že získat tato data není záležitost triviální. Prvním úkolem při návrhu aplikace bylo zjištění, jakým způsobem se vůbec bude možné do celého systému připojovat. Jako základní prostředník pro přístup do systému byla z logických důvodů zvolena jednotka `FNL`, s jejímž firmwarem je možné komunikovat přes dostupné `API` (více informací o jednotce `FNL` a knihovních funkcích je možné zjistit z kapitol 3.2 a 3.2.2). Dostupné knihovní funkce se staly tedy kamenem úrazu celé aplikace – jejich vhodným voláním a parametrizací by na konci mělo být možné číst data z jednotky `Modbus` Slave, což bylo hlavním úkolem této diplomové práce (spolu s vytvořením vhodné filtrace těchto dat).

V další části bude nejprve ukázáno schéma popisující chování navržené aplikace v jednotlivých krocích, v dalším textu budou pak tyto kroky popsány důkladněji.

4.3 Aplikace - řešení

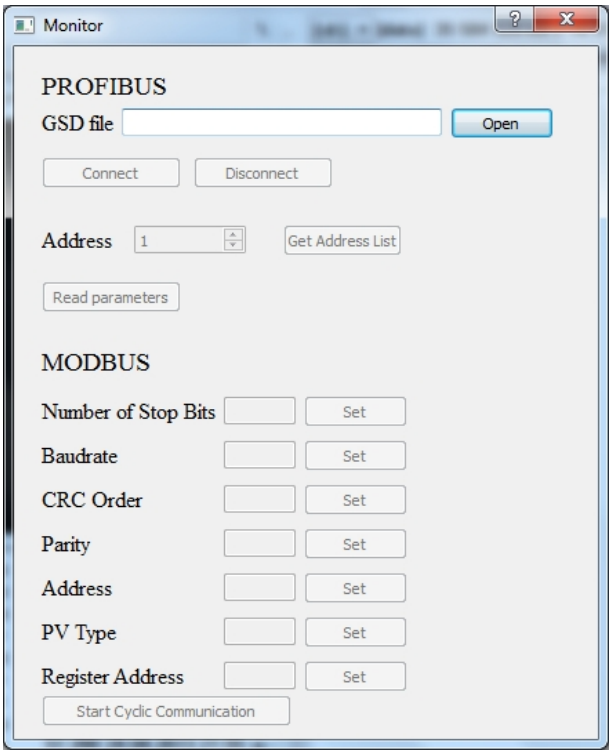


Obrázek 4.2: Životní cyklus aplikace

Obrázek 4.2 znázorňuje, jakými stavy aplikace prochází v průběhu jejího životního cyklu. Jednotlivé stavy budou nyní podrobně popsány (část týkající se změn parametrů je pak rozšířena o popis čtení parametrů z bloků jednotky `T410`).

4.3.1 Start aplikace

Po spuštění aplikace je uživateli nabídnuto okno (viz obr. 4.3), kde jedinou dostupnou možností je nalezení konkrétního `GSD` souboru. Uživatelský manuál aplikace je k dispozici v příloze tohoto dokumentu.



Obrázek 4.3: Vstupní okno aplikace

4.3.2 Načtení GSD souboru

První úkon, který je nutné udělat před samotným připojením k jednotce `FNL`, je přečtení a zpracování tzv. `GSD` souboru.

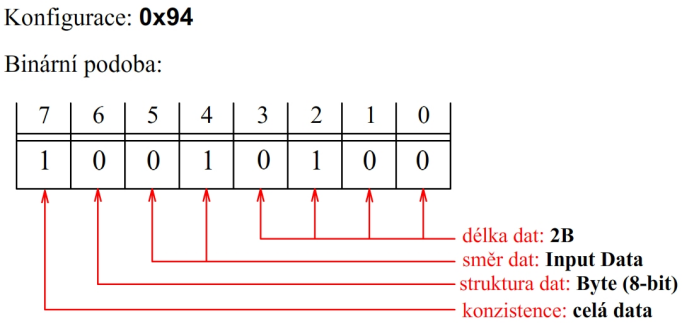
`General Station Description (GSD)` je soubor obsahující informace o každé jednotce, která vystupuje v síti `Profibus`. Soubor je výrobcem standardně dodáván ke každému zařízení a má standardizovaný formát. Vznik `GSD` souboru měl ryze praktické důvody – každé zařízení má své specifické parametry (Baudrate, adresa, časové konstanty, atd.), které jsou dostupné v manuálu. Aby byla práce se zařízením co nejjednodušší, obsahuje `GSD` tyto parametry, které pak stačí s využitím nějakého nástroje načíst a zpracovat.[6]

Každý `GSD` soubor má následující strukturu:

- Hlavička
  - např.: `#Profibus_DP`
- Parametry ve formátu `<klíčové slovo> = <hodnota>`
  - např.: `Ident_Number = 0xa002`
- Konfigurace ve formátu:
  ```
  Module = <nazev_modulu> <hodnota>
  <poradi>
  EndModule
  ```
  - např.:
    ```
    ;Modules for Analog Input
    Module = "Analog Input (AI)short"0x94
    2
    EndModule
    ```

Konfigurace specifikuje množství a uspořádání cyklicky přenášených dat. Je určena přes tzv. identifikátor a vyskytuje se ve dvou formátech - jednoduchý (velikost 1 byte) a speciální (velikost minimálně 2 byty).

Z `GSD` souboru pro jednotku `T410` lze vyčíst, že jsou pro ni k dispozici 2 konfigurace - `0x94` (jednoduchý formát) a `0x42 0x84 0x08 0x05` (speciální formát). Pro zajímavost si na zmíněném jednoduchém formátu dále popíšeme, jak tyto hodnoty interpretovat (viz obr. 4.4). Další informace o konfiguraci a `GSD` souborech lze najít v [6].



Obrázek 4.4: Konfigurace - jednoduchý formát

Zpracování `GSD` souboru ve vyvíjené aplikaci je poměrně jednoduché. Po otevření je soubor procházen řádek po řádku. Z každého řádku jsou odstraněny komentáře (text za znakem `';'`) a mezery. Protože známe přesnou strukturu dokumentu, není složité následně získat klíčové slovo a odpovídající hodnotu v případě parametrů a název modulu s identifikátorem v případě konfigurace. Každá tato dvojice je uložena do struktury a ta je následně uložena do pole:

```
struct info
char name[80];
char value[80];
object_info;
```

Kompletní obsah `GSD` souboru pro jednotku `T410` je možné nalézt v příloze v kapitole A.2, samotný soubor pak na přiloženém CD.

4.3.3 Připojení a inicializace spojení

Po načtení `GSD` souboru je uživateli zpřístupněno tlačítko pro připojení k jednotce `FNL`. Po jeho stisknutí dojde sekvenčně k vykonání těchto operací:

1. připojení k jednotce `FNL` funkcí `ConnectBoard` (popsána v kapitole 3.2.2),
2. nahrání parametrů sběrnice příkazem `load_bus_parameters`,
3. inicializace firmwaru příkazem `init_dp_master` tak, že poté bude firmware na sběrnici pracovat jako `Profibus Master`.

V následujícím textu bude nejprve popsán krok 2 spolu s vysvětlením, jak vypadá vykonání určitého příkazu, následně bude již stručněji popsán krok 3, protože princip práce s příkazy je totožný.

Parametry sběrnice

Pro správné fungování celé aplikace je nutné zajistit korektní chování komunikace na sběrnici, což je zajištěno její vhodnou parametrizací. Některé z para-

metrů byly v předchozím kroku přečteny z `GSD` souboru jednotky `T410` a staticky uloženy v řídící jednotce. Následně je třeba parametry předat dál jednotce `FNL`. K odeslání parametrů sběrnice slouží příkaz `load_bus_parameters`. V tabulce 4.1 jsou shrnuty odesílané parametry včetně jejich stručného popisu.

Parametr	Popis
Baudrate	Číslo na stupnici od 0 do 10 určuje hodnotu Baudrate (viz tab. 4.2).
Slot-Time	Časový úsek, během něhož čeká Master na odpověď od Slave předtím, než odešle další telegram.
Minimum Tsdr	Minimální doba, po kterou Slave musí čekat s odesláním odpovědi. Hodnota musí být menší než maximum Tsdr.
Maximum Tsdr	Maximální doba, po kterou Slave může čekat s odesláním odpovědi. Hodnota musí být menší než Slot-time.
Maximum Tsdr	Maximální doba, po kterou Slave může čekat s odesláním odpovědi. Hodnota musí být menší než Slot-time.
Quiet Time	Časový úsek, po který musí vysílající stanice po odeslání dat čekat, než bude moci data přijímat.
Setup Time	Doba mezi příchodem události (např. přerušení) a reakcí na ni.
GAP Factor	Hodnota specifikuje počet cyklů, během niž jsou hledána nová zařízení snažící se připojit do systému.
HSA	Nejvyšší adresa zařízení, jež operuje na segmentu.
Max-Retry-Limit	Specifikuje počet opakování telegramu, pokud adresovaná stanice neodpovídá.

Tabulka 4.1: Parametry sběrnice

Hodnota	0	1	2	3	4	5	6	7	8	9	10
Baud Rate[kBaud]	9,6	19,2	93,75	187,5	500	1500	3000	6000	12000	45,45	31,25

Tabulka 4.2: Mapování čísel na Baud Rate

S využitím funkcí, jež byly popsány v kapitole 3.2.2, je nyní možné odeslat příkaz firmwaru. Postup odesílání příkazu je ve vytvořené aplikaci následující:

1. Zavolání aplikační funkce `GetSendBox:`
   Ve smyčce je nejprve volána knihovní funkce `GetReceiveBlock`, která zjišťuje, zda nedošlo na sběrnici k nějaké asynchronní události (např. time out, indikace duplicitní adresy, atd.) Pokud je událost zachycena, je následně volána funkce `FreeReceiveBlock` pro navrácení přijatého mailboxu zpátky firmwaru. Před otáčkou cyklu je nakonec zavolána funkce `GetRequestBlock`, která v případě úspěchu vrátí nenulový ukazatel(`m`) na mailbox, čímž ukončí smyčku.

2. Naplnění mailboxu daty:
   Pro uložení dat jsou použity struktury, a to takovým způsobem, aby odpovídaly uspořádání mailboxu (viz 3.2.1). Pro příkaz `load_bus_parameters` vypadá naplnění dat takto:

```
m->LoadBusParameter.Header.Service = LOAD_BUS_PARAMETER; /*0x0A*/
m->LoadBusParameter.Header.Group = CM_DP; /*0x08*/
m->LoadBusParameter.Baudrate = 10; /*31,25 kBaud*/
m->LoadBusParameter.BusParameter.SlotTime = 4094;
m->LoadBusParameter.BusParameter.MinTsdr = 22;
m->LoadBusParameter.BusParameter.MaxTsdr = max_tsdr; /*přečteno z GSD
souboru*/
m->LoadBusParameter.BusParameter.QuietTime = 0;
m->LoadBusParameter.BusParameter.SetTime = 55;
m->LoadBusParameter.BusParameter.GapFactor = 10;
m->LoadBusParameter.BusParameter.HighestStationAdress = 126;
m->LoadBusParameter.BusParameter.MaxRetryLimit = 1;
```

3. Odeslání dat funkcí `sendRequestBlock`,

4. Po odeslání dat je třeba počkat na reakci firmwaru. Čekání probíhá opět ve smyčce, kdy se opakovaně volá funkce `GetReceiveBlock`. Znovu se testuje, zda nedošlo k asynchronní události, a také to, zda odpověď, kterou v tento okamžik odeslal firmware, odpovídá odeslaným datům (častý případ, kdy například firmware odpoví příkazem `ddlm_abort`). Pokud funkce vrátí správnou odpověď (parametry `Service` a `Group` odpovídají odeslaným datům), skončí smyčka a dojde k otestování parametru `Result`, zda příjem dat na straně firmwaru skončil úspěchem.

Inicializace firmwaru

Po nahrání parametrů sběrnice je již možné, aby došlo k inicializaci tzv. `Class-1-master` jádra, jinými slovy, aby se firmware jako master připojil do sítě `Profibus`. Do okamžiku zavolání tohoto příkazu bylo zmíněné jádro ve stavu `OFFLINE`, po úspěšném zavolání inicializačního příkazu `init_dp_master` přejde do stavu `STOP`.

4.3.4 Zadání adresy

Firmware už v tomto okamžiku pracuje na sběrnici jako master, ale nemá žádné informace o okolních stanicích, které se na sběrnici mohou vyskytovat. Pro navázání spojení s některým zařízením potřebuje master znát jeho adresu, což ovšem není informace, která by se vyskytovala v `GSD` souboru, a proto je třeba využít příkazu `get_live_list`. V případě, že skončí volání příkazu úspěchem, je k dispozici pole o 127 bytech. Indexy položek v poli odpovídají jednotlivým adresám a data, která tyto položky obsahují, se interpretují takto:

- 0 → stanice je slave,
- 3 → stanice je master,
- 4 → žádná stanice s touto adresou není na sběrnici přítomna.

4.3.5 Změna parametrů

Na sběrnici je možné připojit mnoho různých zařízení s mnoha různými parametry. Od začátku byla snaha vytvořit aplikaci, která bude navýsost univerzální, a proto je možné ještě před začátkem cyklické komunikace některé důležité parametry změnit. Jedná se o tento výběr:

- Počet stop bitů,
- Baudrate,
- CRC pořadí,

- Parita,
- Adresa `Modbus` zařízení,
- Typ přenášené proměnné (např. float, integer, atd.),
- Adresa `Modbus` registru.

Čtení bloků z jednotky T410

V úvodu kapitoly 4 bylo zmíněno, že v sekci týkající se čtení parametrů bude popsán postup čtení bloků z jednotky `T410`. Toto zařazení se ukázalo nakonec jako jediné smysluplné, protože změna parametrů je v celé aplikaci jediné místo, kde ke čtení resp. zápisu do bloků dochází.

Každé `Profibus PA` zařízení obsahuje jeden fyzický blok (`Physical block`) a alespoň jeden blok funkční (`Function`) a převodníkový (`Transducer`). Přístup k těmto blokům je zajištěn přes dvojici `slot-index`. Každé konkrétní zařízení má tyto bloky jinak adresované, a proto je nejprve nutné zjistit, kde se tyto bloky v adresním prostoru nachází.

Jednotka `Fint T410` obsahuje 1 fyzický blok (PB), 4 funkční (FB) a 4 převodníkové (TB) bloky.

Prvním krokem je přečtení tzv. adresáře objektů. Adresář se skládá z hlavičky (`Directory Object Header`), souhrnných informací o blocích (`Composite List Directory`) a z ukazatelů na jednotlivé bloky (`Composite Directory`).

Hlavičku adresáře objektů je možné nalézt u každého `Profibus PA` zařízení na pozici slot = 1 a index = 0 a má tuto strukturu (viz tabulka 4.3, která obsahuje parametry přečtené z jednotky `T410`):

2B	2B	2B
DIR_ID	DIR_REV_NO	NO_DIR_OBJ
0	1	1
NO_DIR_ENTRIES	FIRST_COMP_LIST_DIR_ENTRY	NO_COMP_LIST_DIR_ENTRY
12	1	3

Tabulka 4.3: Hlavička adresáře objektů

Jednotlivé položky pak mají tento význam:

- `DIR_ID`: Identifikační číslo adresáře,
- `DIR_REV_NO`: Číslo revize,
- `NO_DIR_OBJ`: Počet dalších indexů použitých ze slotu číslo 1,
- `NO_DIR_ENTRIES`: Celkový počet položek v `Composite List Directory` a v `Composite Directory`,
- `FIRST_COMP_LIST_DIR_ENTRY`: Index první položky `Composite List Directory`, jež se nachází vždy ve slotu číslo 1,
- `NO_COMP_LIST_DIR_ENTRY`: Počet položek v `Composite List Directory` (položkou se myslí trojice Index, Offset a NO_PB, čili celkem 4B, viz dále).

Po přečtení hlavičky je možné přečíst zároveň `Composite List Directory` a `Composite Directory`, neboť hodnoty obou skupin se nachází ve slotu 1 a na indexu přečteném z tabulky (pro jednotku `T410` je tento index 1). Po přečtení celé této skupiny dat je nutné nejprve zjistit hodnoty obsažené v adresáři `Composite List Directory`, který má tuto strukturu (viz 4.4):

1B	1B	2B	1B	1B	2B	1B	1B	2B
Index PB		NO_PB	Index TB		NO_TB	Index FB		NO_FB
Index	Offset		Index	Offset		Index	Offset	

Tabulka 4.4: Composite List Directory

Pro každý typ bloku jsou udány tyto parametry:

- `Index`: Pozice začátku dat bloku,
- `Offset`: Pozice dat na daném indexu (Offset × 4B),
- `No_PB/TB/FB`: Počet bloků daného typu.

Pro jednotku `Fint T410` vypadá `Composite List Directory` takto (viz tabulka 4.5):

1B	1B	2B	1B	1B	2B	1B	1B	2B
Index PB		NO_PB	Index TB		NO_TB	Index FB		NO_FB
Index	Offset		Index	Offset		Index	Offset	
1	4	1	1	5	4	1	9	4

Tabulka 4.5: T410 - Composite List Directory

Nyní je možné přečíst konkrétní informace o všech blocích (`Composite Directory`), které se nachází ve stejném slotu (1) a na indexu a offsetu přečteném z `Composite List Directory`. Pokud chceme například zjistit, kde se nachází všechny informace o fyzickém bloku, přečteme data ze slotu 1, indexu 1 a ukazatel na data zvýšíme o 4B. Tabulka 4.6 znázorňuje, jak vypadá struktura `Composite Directory` fyzického bloku (pro ostatní bloky je struktura stejná):

1B	1B	2B
Physical Block		
Slot	Index	Číslo

Tabulka 4.6: Composite Directory

Pro fyzický blok jednotky `T410` vypadá obsah `Composite Directory` takto: `Index` = 0, `slot` = 0, počet = 38.

Nyní jsou k dispozici potřebné údaje k tomu, aby bylo možné přečíst požadované informace z jednotlivých bloků. Pro čtení resp. zápis dat jsou k dispozici příkazy `ddlm_read` resp. `ddlm_write` s těmito parametry (viz tabulka 4.7):

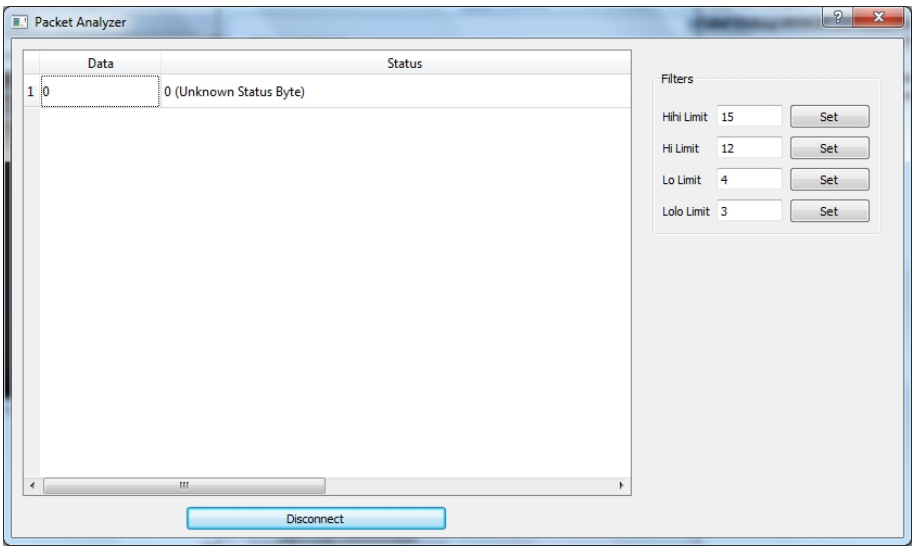
Parametr	Popis
Handle	Identifikace spojení, slouží pouze k informativním účelům
Slot number	Hodnota slotu (číslo od 0 do 255)
Index	Hodnota indexu (číslo od 0 do 255)
Length	Počet bytů dat k přečtení
Data (pro ddlm_write)	data, která mají být zapsána na dané umístění

Tabulka 4.7: Parametry příkazů ddlm_read a ddlm_write

V případě příkazu `ddlm_read` obsahuje odpověď firmwaru kýžená data (parametr odpovědi `Data`).

4.3.6 Čtení dat

Po připojení a inicializaci spojení, zadání adresy a případné změně parametrů je dalším krokem čtení dat, která jsou k dispozici v zařízení vystupující na sběrnici jako `Modbus` Slave. V této části se také změní vzhled aplikace, což demonstruje obrázek 4.5



Obrázek 4.5: Aplikace - zachytávání dat

Proces čtení dat je doslova kapitolou sama pro sebe. Po stisknutí tlačítka pro zahájení cyklické komunikace dojde opět k vykonání sekvence příkazů:

- načtení dalších parametrů (konkrétní informace o slave, čili o jednotce `T410`) příkazem `load_slave_parameters`,
- zahájení cyklické komunikace příkazem `start_data_exchange`,
- čtení `Modbus` dat z `Process Data Image`.

V následujícím textu budou stručně popsány první zmíněné kroky (jedná se o standardní volání příkazů), větší důraz bude kladen na čtení dat z `Process Data Image`, protože to bylo zatím zmíněno pouze v teoretické části v kapitole 3.2.1.

Parametry zařízení Modbus slave

GSD soubor obsahuje kromě informací o sběrnici také parametry, které se přímo vztahují k hardwaru dané jednotky. Před začátkem cyklické komunikace je tedy nezbytné dodat tyto parametry jednotce `Modbus Master` (inicializovaný firmware jednotky `FNL`). Tabulka 4.8 obsahuje kromě názvu také konkrétní hodnoty pro jednotku `T410` (spolu s vysvětlením jejich významu).

Parametr: `Slave Address`	**Hodnota:** `Address`
Význam: Proměnná `address` obsahuje adresu `Modbus Slave` zařízení, jež byla přečtena v kroku čtení parametrů.	
Parametr: `Sl_Flag`	**Hodnota:** `0x80`
Význam: Zařízení bude aktivováno po zapsání parametrů, čili se bude moci zúčastnit cyklické komunikace.	
Parametr: `WDFact1, WDFact2`	**Hodnota:** `0x00; 0x00`
Význam: Watchdog je deaktivován.	
Parametr: `Identification Number`	**Hodnota:** `0xA002`
Význam: Identifikační číslo zařízení (přečteno z GSD).	
Parametr: `Freeze Requested`	**Hodnota:** `0x00`
Význam: Zařízení nepodporuje tzv. Freeze Mode (zamrznutí hodnoty na vstupu).	
Parametr: `Sync Requested`	**Hodnota:** `0x00`
Význam: Zařízení nepodporuje tzv. Sync Mode (zamrznutí hodnoty na výstupu).	
Parametr: `User Parameters Length`	**Hodnota:** `0x03`
Význam: Délka `User_Prm_Data` v bytech (viz dále).	
Parametr: `Minimum Slave Interval`	**Hodnota:** `0x7D0`
Význam: Čas (200μs) mezi dvěma čtecími cykly (Master→Slave).	
Parametr: `Group Identification`	**Hodnota:** `0x00`
Význam: Slave není přiřazen do žádné skupiny Slave zařízení.	
Parametr: `Group Identification`	**Hodnota:** `0x00`
Význam: Slave není přiřazen do žádné skupiny Slave zařízení.	
Parametr: `Maximum Diagnostic Data Length`	**Hodnota:** `0x0E`
Význam: Maximální délka diagnostických dat, které odesílá slave masteru.	
Parametr: `Offset Inputs`	**Hodnota:** `0x2`
Význam: Offset v `Process Data Image` - od tohoto offsetu budou ukládána příchozí `Modbus` data při cyklické komunikaci.	

Parametr: `Offset Outputs`	**Hodnota:** `0x1A`
Význam: Offset v `Process Data Image` - od tohoto offsetu budou ukládána odchozí `Modbus` data při cyklické komunikaci.	
Parametr: `Config Length`	**Hodnota:** `0x01`
Význam: Délka konfiguračních dat (viz kapitola A.2).	
Parametr: `Config And User Parameter Data`	**Hodnota:** `0x94;0x00;0x00;0x00`
Význam: Podle parametru `Config Length` je první byte konfigurační, ostatní byty jsou poté uživatelskými parametry (přečtené z GSD souboru).	

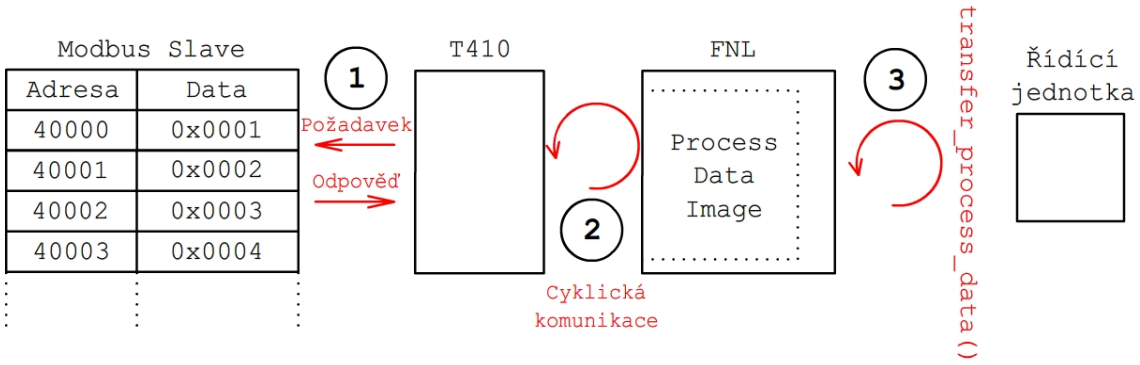
Tabulka 4.8: Parametry příkazu `load_slave_parameters`

Zahájení cyklické komunikace

Po načtení všech potřebných parametrů je již možné zahájit cyklickou komunikaci Master→Slave, k čemuž slouží příkaz `start_data_exchange`. Na následujících řádcích je pro ilustraci uveden přepis metody, v níž je příkaz `start_data_exchange` volán:

```
void MainWindow::startDataExchange() {
   short result;
   PTCommand m;
   bool success;
   m = Communication::GetSendBox();
   m->StartDataExchange.Header.Group = CM_DP;
   m->StartDataExchange.Header.Service = START_DATA_EXCHANGE;
   result = Communication::sendRequestBlock(m);
   success = Communication::testResult("StartDataExchange", result);
   if (success) {
      m = Communication::WaitRecBlock(CM_DP | CM_READY, START_DATA_EXCHANGE);
      result = m->StartDataExchange.Header.Result;
      success = Communication::testResult("StartDataExchange", result);
   }
   if (success) {
      result = Communication::freeReceiveBlock();
      success = Communication::testResult("StartDataExchange", result);
   }
}
```

Čtení dat z Process Data Image



Obrázek 4.6: Schema procesu přenosu `Modbus` dat k řídící jednotce

Na obrázku 4.6 je přehledně znázorněno, jak vypadá komunikace napříč celým systémem. Konkrétně se dá komunikace rozdělit na tyto části:

1. **Přenos dat mezi `Modbus Slave` a `Modbus Master` (T410)** - data jsou čtena z `Modbus` zařízení, jehož adresa a adresa čteného registru byly nastaveny v první části aplikace (viz kapitola 4.3.5),
2. **Cyklická komunikace** - cyklická komunikace byla zahájena příkazem `start_data_exchange`, jenž byl popsán v předchozí kapitole. Při této komunikaci dochází k přenosu tzv. primární proměnné (hodnota výše uvedeného `Modbus` registru) mezi jednotkou `T410` (pracující jako `Profibus Slave`) a jednotkou `FNL` (jejíž firmware funguje jako `Profibus Master`). Data jsou na straně `FNL` ukládány do `Process Data Image`, konkrétně do místa, jež určuje parametr `Offset Inputs` definovaný v příkazu `load_slave_parameters`.
3. **Čtení dat z Process Data Image** - tato část je z hlediska implementace nejzajímavější, a proto bude na následujících řádcích blíže popsána.

Pro přenos dat z `Process Data Image` je k dispozici několik příkazů, jež byly popsány v kapitole 3.2.2. V aplikace je pro tento účel použit příkaz `get_process_data`, protože pro účel aplikace je potřebná pouze funkce čtení. Vstupními parametry jsou:

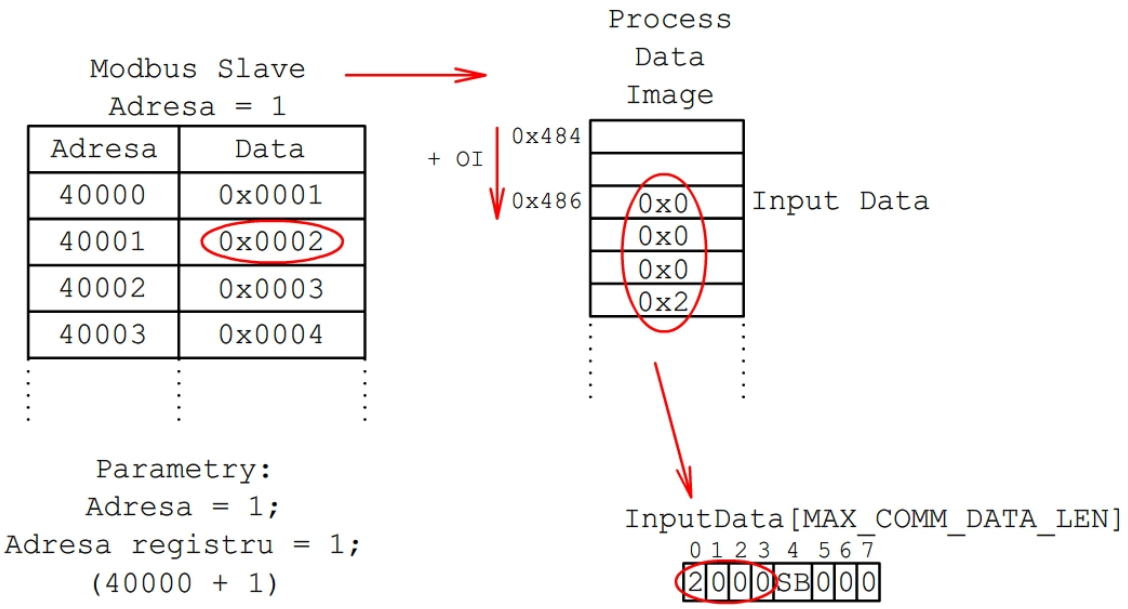
- `data_offset` - offset do adresního prostoru `Process Data Image`. Musí být shodný s parametrem `Offset Inputs` příkazu `load_slave_parameters`,

- `data_len` - počet přenášených dat (v bytech). V aplikaci je nastavena na 24 bytů (konstanta `MAX_COMM_DATA_LEN`), ovšem kýženou primární proměnnou lze najít na prvních pěti bytech,
- `process_data_ptr` - ukazatel na pole bytů, do něhož budou data uložena (má tedy statickou velikost `MAX_COMM_DATA_LEN`).

Ve zdrojovém kódu vypadá volání příkazu takto:

```
void MainWindow::getProcessData() {
   BYTE InputData[MAX_COMM_DATA_LEN];
    /* inicializace pole na samé nuly */
   memset(&InputData, 0x00, sizeof (InputData));
   /* INPUT_OFFSET = 0x02, MAX_COMM_DATA_LEN = 0x18 */
   short result = GetProcessData (INPUT_OFFSET, MAX_COMM_DATA_LEN,
                                  InputData, TRUE);
}
```

Do pole `InputData` jsou po úspěšném zavolání funkce `GetProcessData` nakopírována data z `Process Data Image`, jak ilustruje i obrázek 4.7.



Obrázek 4.7: Průchod dat z registrů `Modbus Slave` do pole v řídící jednotce

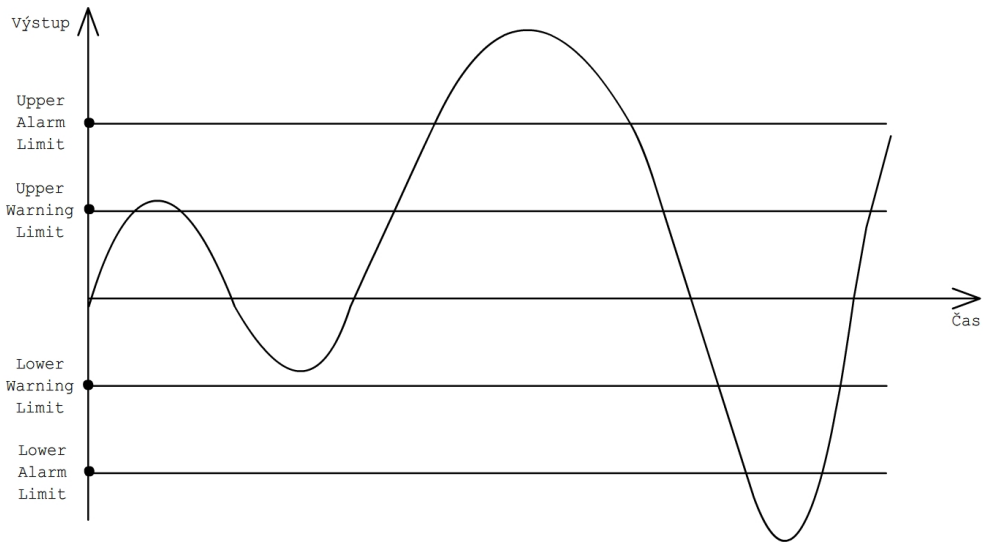
Spolu s daty je z `Process data Image` přečten také tzv. `Status Byte (SB)`.

Pro pochopení významu SB je nejprve nutné pochopit, jak u protokolu `Profibus PA` fungují tzv. kontroly limitů.

Princip kontroly limitů je poměrně jednoduchý - každé zařízení má přednastaveny limity, po jejichž překročení dojde k nastavení odpovídající proměnné. Celkem jsou v `PA` zařízeních rozlišovány 4 druhy limitů:

- `Upper Alarm Limit (HI_HI_LIM)`,
- `Upper Warning Limit(HI_LIM)`,
- `Lower Warning Limit (LO_LIM)`,
- `Lower Alarm Limit (LO_LO_LIM)`

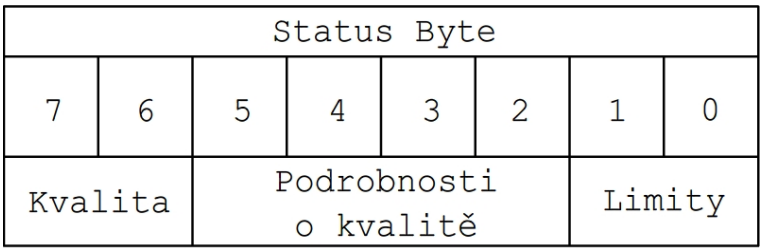
Význam limitů je demonstrován na obrázku 4.8:



Obrázek 4.8: Limity

Vytvořená aplikace umožňuje uživateli nastavovat tyto limity, což je patrné z obrázku 4.5. Limity jsou nastavovány standardním procesem čtení a zápisu do funkčního bloku.

Nyní je již možné vysvětlit význam `Status Bytu`. `Status Byte` je přiřazen ke každé naměřené hodnotě a má následující formát (viz obr. 4.9):



Obrázek 4.9: Status Byte

- **Kvalita** - informace o naměřené hodnotě,
- **Podrobnosti o kvalitě** - rozšiřující informace o naměřené hodnotě,
- **Limity** - informace, zda byl překročen některý z limitů. Nabývá těchto hodnot:
  - `00` - naměřená hodnota je validní, nepřekračuje žádný limit,
  - `01` - naměřená hodnota je menší než dolní limit,
  - `11` - naměřená hodnota je vyšší než horní limit.

Ve vytvořené aplikaci je každý `Status Byte` zaznamenáván do zvláštního sloupce. Jednotlivé statusy jsou interpretovány takto:

Status Byte	Interpretace
`0x80`	**Quality:** Good, **Quality Status:** OK, **Limits:** Not Limited
`0x89`	**Quality:** Good, **Quality Status:** Active Advisory Alarm, **Limits:** Low limited
`0x8A`	**Quality:** Good, **Quality Status:** Active Advisory Alarm, **Limits:** High limited
`0x8D`	**Quality:** Good, **Quality Status:** Active Critical Alarm, **Limits:** Low limited
`0x8E`	**Quality:** Good, **Quality Status:** Active Critical Alarm, **Limits:** High limited

0x44	**Quality:** Uncertain, **Quality Status:** Last Usable Value, **Limits:** Not Limited
0x51	**Quality:** Uncertain, **Quality Status:** Sensor Conversion not Accurate, **Limits:** Low limited
0x52	**Quality:** Uncertain, **Quality Status:** Sensor Conversion not Accurate, **Limits:** High limited
0x08, 0x09, 0x0A	**Quality:** Bad, **Quality Status:** Device Failure
0x10, 0x11, 0x12	**Quality:** Bad, **Quality Status:** Sensor Failure
0x14, 0x15, 0x16	**Quality:** Bad, **Quality Status:** No Communication

4.3.7 Otestování aplikace

Aplikace bohužel nebyla otestována na žádném konkrétním `Modbus` zařízení, k základním testům byla využita shareware verze programu `MOD_RSsim`, která plnohodnotně nahrazuje kýžený hardware.

5 Závěr

Tato diplomová práce se zabývá průmyslovými sběrnicemi, především pak sběrnicí Profibus a její variantou PA. Hlavním cílem bylo s využitím dostupného hardwaru vytvořit aplikaci, která umí zachytávat přenášená data, vhodně tato data filtrovat a zobrazovat je uživateli.

V první, teoretické části jsou nejprve stručně popsány moderní průmyslové sběrnice s důrazem na sběrnice typu Modbus a Profibus PA, neboť tyto dvě sběrnice jsou hlavní součástí komunikačního řetězce, jenž byl pro účely této diplomové práce sestaven.

Druhá část práce se zabývá zmiňovaným komunikačním řetězcem. Jako základní stavební kameny řetězce byly vybrány 2 jednotky – FintT410 (implementující rozhraní mezi protokoly Ethernet a Profibus PA) a Fieldbus Network Link (implementující rozhraní mezi protokoly Profibus PA a Modbus). V této části jsou tedy popsány obě jednotky, především pak jednotka Fieldbus Network Link, která skrz své API poskytuje přístup na sběrnici Profibus PA.

Dalším krokem byla již vlastní praktická část, čili vytvoření monitoru přenášených dat. Nejprve však bylo nutné naučit se pracovat s dostupným API, které je součástí firmwaru jednotky Fieldbus Network Link. S využitím dostupných manuálů byla jako první vytvořena aplikace, která uměla číst data z paměťových bloků jednotky T410. Tento program, jež byl následně podle požadavku přepsán z programovacího jazyka ANSI C do jazyka C#, byl zařazen do projektu Výzkum a vývoj vícepaprskových ultrazvukových průtokoměrů kapalin (FR-TI1/137). V další fázi vývoje pak byla vytvořena aplikace, která umí monitorovat data, která jsou přenášena mezi jednotkami v rámci cyklické komunikace, čili dokáže cyklicky číst data z registrů Modbus zařízení. Součástí aplikace je také možnost konfigurace, která umožňuje měnit parametry sběrnice s ohledem na připojená zařízení. Pro pohodlnou manipulaci s programem bylo vytvořeno jednoduché uživatelské rozhraní.

V další fázi vývoje by bylo vhodné rozšířit aplikaci o nové funkčnosti jako je například manuální nastavování adresy nebo získávání diagnostických dat připojených zařízení. Další užitečné vylepšení by mohlo například být ukládání přenášených dat do databáze pro možnost zpětného vyhodnocování.

6 Přehled zkratek

CIP	Common Industrial Protocol
ODVA	Open DeviceNet Vendors Association
HART	Highway Addressable Remote Transducer
CAN	Controller Area Network
RTU	Remote Terminal Unit
CRC	Cyclic Redundancy Check
ASCII	American Standard Code for Information Interchange
LRC	Longitudinal Redundancy Check
PROFIBUS	Process Field Bus
FNL	Fieldbus Network Link
ZVEI	Zentralverband Elektrotechnik und Elektronikindustrie
FMS	Fieldbus Message Specification
DP	Decentralized Periphery
PA	Process Automation
PNO	PROFIBUS User Organization
PI	PROFIBUS International
MBP	Manchester Bus Powered
FDL	Fieldbus Data Link
DPV	Decentralized Periphery Version
FDL	Fieldbus Data Link
IEC	International Electrotechnical Commission
DPM1	DP Master Class 1
SRD	Send and Request Data with Reply
MSAC1	Master/Slave Acyclic C1
MSAC2	Master/Slave Acyclic C2
API	Application Programming Interface
RAM	Random Access Memory
DPR	Dual-Port RAM
MSB	Most Significant Bit
PV	Primary Variable
GUI	Graphical User Interface
GSD	General Station Description
HSA	Highest Station Address
SB	Status Byte

Tabulka 6.1: Přehled zkratek

Literatura

[1] *Real Time Automation* [online]. 2010 [cit. 2013-05-01]. Dostupné z: <http://www.rtaautomation.com/profibus/>

[2] BURGET, Petr. Profibus-PA – řešení pro automatizaci procesů. *AUTOMA*. 2001, č. 1. [online] [cit. 1. 5. 2013] Dostupné z: <http://www.odbornecasopisy.cz/index.php?id_document=33429>

[3] HANZÁLEK, Zdeněk a Pavel BURGET. *Hw.cz* [online]. 2004 [cit. 2013-05-01]. Dostupné z: <http://www.hw.cz/navrh-obvodu/rozhrani/prumyslova-sbernice-profibus.html>

[4] PROFIBUS-PA: Part 4 Communications [online]. 1999 [cit. 2013-05-01]. Dostupné z: <http://www.samson.de/pdf_en/l453en.pdf>

[5] HLAVA, Jaroslav. Prostředky automatického řízení II: analogové a číslicové regulátory, elektrické pohony, průmyslové komunikační systémy [online]. Vyd. 1. Praha: České vysoké učení technické, 2000 [cit. 2013-05-01]. ISBN 978-800-1022-214.

[6] DIEDRICH, Christian a Thomas BANGEMANN. Profibus PA: Instrumentierungstechnologie für die Verfahrenstechnik. 2. Aufl. München: Oldenbourg Industrieverl, 2006. ISBN 978-383-5630-567.

[7] LIAN, Odd. FintPROFI T410 module specification. 2009.

[8] BÖHM, Pavel. *Komunikační modul pro průmyslovou sběrnici Foundation Fieldbus*. Plzeň, 2012. Diplomová práce. Západočeská univerzita v Plzni, Fakulta Aplikovaných věd, Katedra informatiky a výpočetní techniky.

[9] ControlNet je plnou součástí ODVA. *AUTOMA* [online]. 2008, č. 8 [cit. 2013-05-05]. Dostupné z: <http://www.odbornecasopisy.cz/res/pdf/37699.pdf>

[10] HART – jednoduchá a spolehlivá komunikace. *AUTOMA* [online]. 2004, č. 7 [cit. 2013-05-05]. Dostupné z: <http://www.odbornecasopisy.cz/index.php?id_document=32435>

[11] MAHALIK, N. *Fieldbus technology: industrial network standards for real-time distributed control*. New York: Springer, 2003, xxv, 590 p. ISBN 35-404-0183-0.

[12] HANZÁLEK, Zdeněk a Pavel BRONEC. Profibus a Foundation Fieldbus: konkurenti v oblasti průmyslových sběrnic. *AUTOMA* [online]. 2000, č. 2 [cit. 2013-05-05]. Dostupné z: <http://www.odbornecasopisy.cz/index.php?id_document=27615>

[13] RONEŠOVÁ, Andrea. Přehled protokolu MODBUS. [online]. 2005 [cit. 2013-04-28]. Dostupné z: <home.zcu.cz/∼ronesova/bastl/files/modbus.pdf>

[14] MODICON, Inc., Industrial Automation Systems. *Modicon Modbus Protocol Reference Guide*. 1996. Dostupné z: <http://modbus.org/docs/PI_MBUS_300.pdf>

[15] TMG I-TEC GMBH. *PROFIBUS FNLDPE Interface*. 2003.

[16] *QT Project* [online]. 2013 [cit. 2013-04-01]. Dostupné z: <http://qt-project.org/>

A Přílohy

A.1 Uživatelská příručka

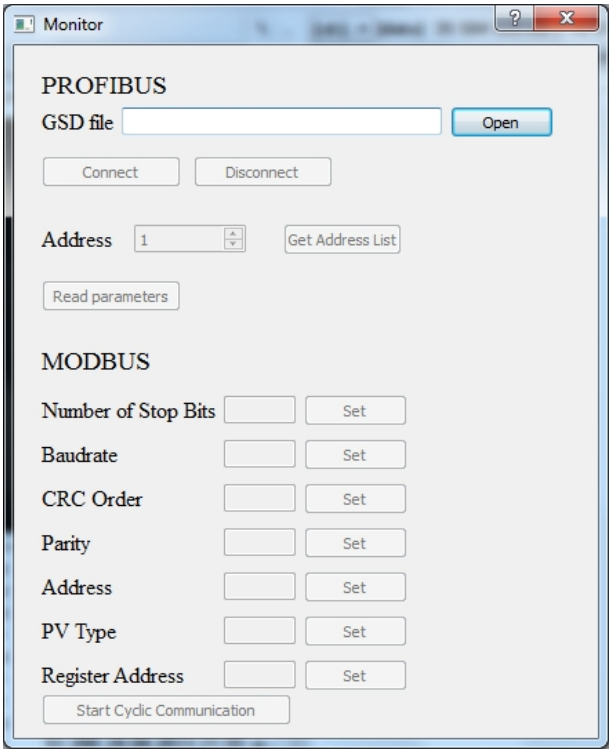
Program je dostupný na přiloženém CD, k dispozici je jak zkompilovaná aplikace, tak zdrojové soubory.

Pro start aplikace stačí spustit soubor `monitor.exe`, který se nachází ve složce `/Program/Bin/`

Uživatel má také možnost vlastního sestavení programu. Součástí přiloženého CD (ve složce `/Program/Src/`) je kompletní projekt vytvořený ve vývojovém prostředí `NetBeans` verze 7.1.2.

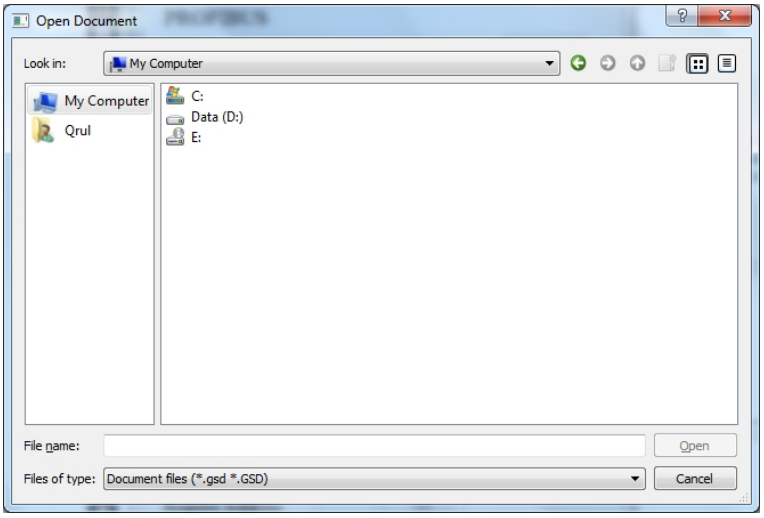
A.1.1 Spuštění programu a načtení GSD souboru

Po spuštění programu je uživateli zobrazeno úvodní okno aplikace `Monitor`, viz obrázek A.1.



Obrázek A.1: Monitor - úvodní okno

Z obrázku A.1 je patrné, že uživatel nemá prozatím přístupné téměř žádné funkce, k dispozici je pouze čtení `GSD` souboru. Po stisknutí tlačítka `Open` je uživateli zobrazeno okno pro výběr souboru (soubory jsou filtrovány podle přípony `.GSD` nebo `.gsd`), viz obrázek A.2

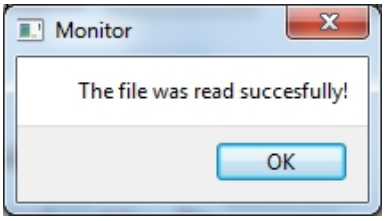


Obrázek A.2: Monitor - načtení souboru

Po nalezení souboru a potvrzení okna pro výběr mohou nastat 2 možnosti:

1. Soubor není validní GSD soubor, uživateli je zobrazena hláška viz obr. A.3 a),
2. Soubor je validní GSD soubor, uživateli je zobrazena hláška viz obr. A.3 b)

(a) Nevalidní GSD soubor

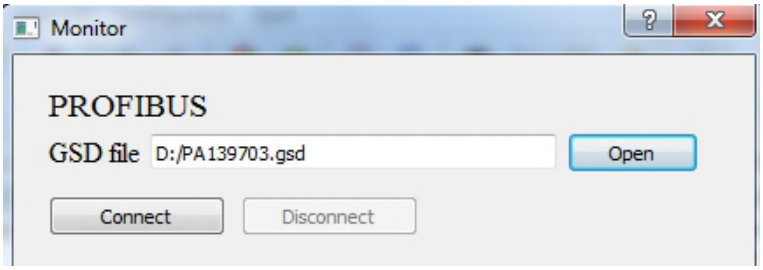


(b) Validní GSD soubor

Obrázek A.3: Monitor - validace GSD souboru

A.1.2 Připojení k jednotce a načtení adresy

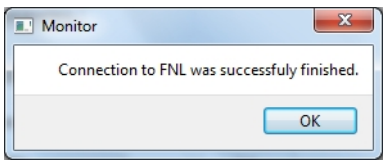
Po načtení validního `GSD` souboru je uživateli zpřístupněno tlačítko `Connect` pro připojení k jednotce `FNL` (viz obr. A.4)



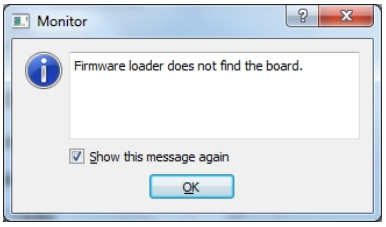
Obrázek A.4: Monitor - připojení

Po stisknutí tlačítka `Connect` mohou nastat 2 situace:

1. Došlo k úspěšnému připojení k jednotce, uživateli je zobrazena hláška viz obr. A.5 a),
2. Jednotka není připojena k počítači (či má odpojené napájení), uživateli je zobrazena hláška viz obr. A.5 b) Aplikace se nijak nezmění, uživatel má možnost pokus o připojení znovu opakovat.



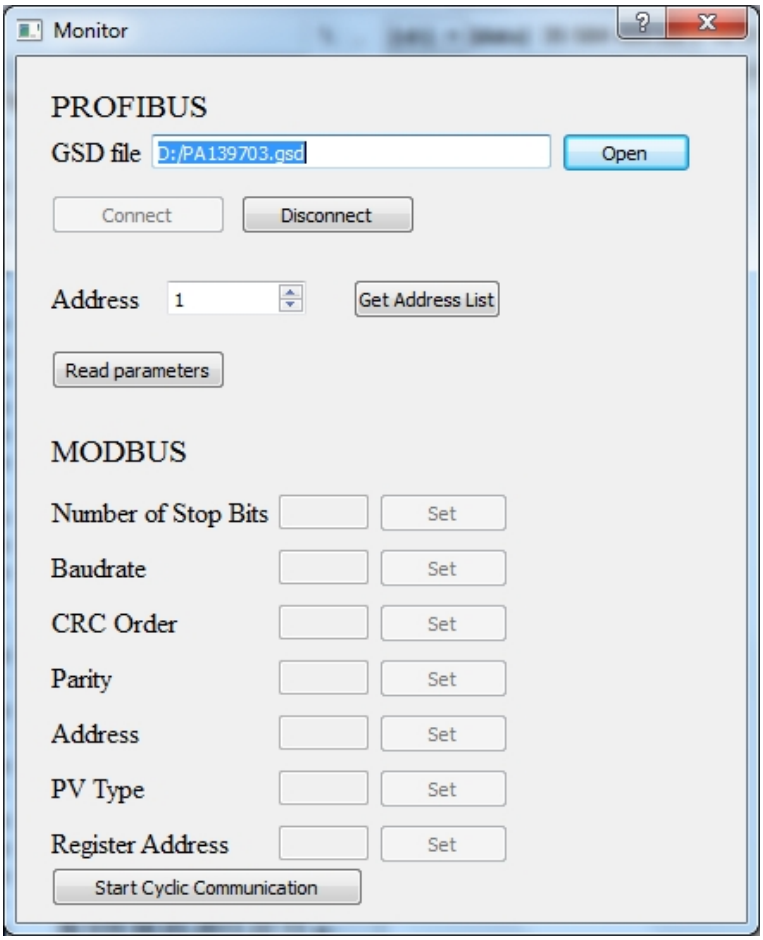
(a) Korektní připojení



(b) Problém s připojením

Obrázek A.5: Monitor - hlášky po pokusu o připojení

Po úspěšném připojení jsou uživateli zpřístupněny další funkce (viz obr. A.6).



Obrázek A.6: Monitor - stav po úspěšném připojení

Po připojení je samozřejmě k dispozici možnost odpojení, k čemuž slouží tlačítko `Disconnect`. Po jeho stisknutí jsou opět deaktivována všechna pole, která byla aktivována po úspěšném připojení.

Další funkcí je určení adresy připojeného `Profibus PA` zařízení. Uživatel má 3 možnosti, jak s tou funkcí naložit:

1. Uživatel zná adresu zařízení, může ji tedy ručně zadat do připraveného pole (defaultní možnost),
2. Uživatel nezná adresu zařízení, po stisknutí tlačítka `Get Address List` je zavolána funkce, která nalezne všechna připojená zařízení a jejich adresy přidá do rozbalovacího seznamu. Nutno podotknout, že volání funkce nějakou dobu trvá (maximálně 1 minutu),

3. Adresa zařízení se shoduje s tou přednastavenou (`1`), uživatel může nastavování adresy ignorovat a pokračovat postupem ze sekce A.1.3, eventuálně ze sekce A.1.4.

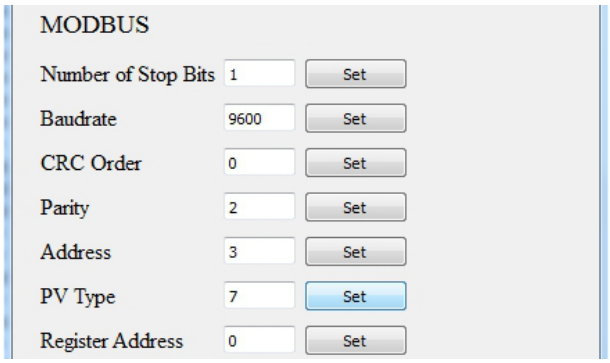
A.1.3 Nastavování parametrů

Uživatel má možnost nastavit několik parametrů, které jsou důležité pro korektní chování při cyklické komunikaci, jedná se o tyto parametry:

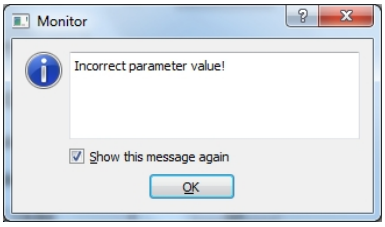
- Počet stop bitů (`Number of Stop Bits`) - defaultní hodnota: `1`,
- Baudrate (`Baudrate`) - defaultní hodnota: `9600`,
- Pořadí bytů v CRC součtu (`CRC Order`) - defaultní hodnota: `0`,
- Parita (`Parity`) - defaultní hodnota: `2`,
- Adresa Modbus zařízení (`Address`) - defaultní hodnota: `3`,
- Datový typ primární proměnné (`PV Type`) - defaultní hodnota: `7`,
- Adresa Modbus registru (`Register Address`) - defaultní hodnota: `0`.

Jedná se o nepovinnou možnost, pokud uživatel nechce nastavovat parametry, může pokračovat postupem, jenž je popsán v sekci A.1.4. Pokud se tak stane, jsou pro parametrizaci použity defaultní hodnoty.

Pokud uživatel chce nějakým způsobem změnit paremetry, musí stisknout tlačítko `Read Parameters`. Po jeho stisknutí jsou aktivována jednotlivá pole pro změnu konkrétních parametrů (viz obr. A.7). Každý konkrétní parametr se aktualizuje až po stisknutí tlačítka `Set`. V případě, že uživatel nezadá validní hodnotu, je zobrazen hláška viz obr. A.8.

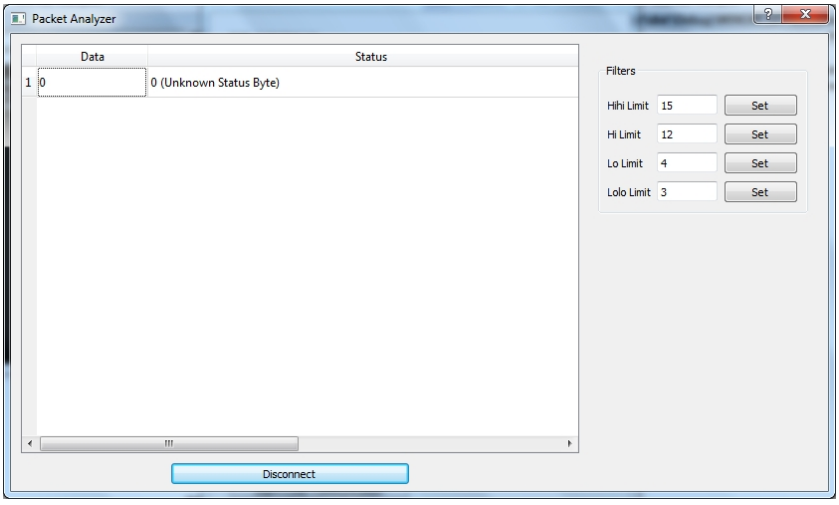


Obrázek A.7: Monitor - nastavování parametrů



Obrázek A.8: Monitor - nevalidní hodnota parametru

A.1.4 Cyklická komunikace



Obrázek A.9: Monitor - okno cyklické komunikace

Po stisknutí tlačítka `Start Cyclic Communication` je původní okno deaktivováno a zároveň je zobrazeno okno nové, viz. obr. A.9.

Data jsou zaznamenávána do tabulky, kde v prvním sloupci jsou konkrétní data, v druhém sloupci pak `Status Byte` spolu s jeho interpretací. V pravé části je možné nastavovat hodnoty alarmů a to stejným způsobem, jako se nastavovali parametry v sekci A.1.3.

V případě, že byly správně nastaveny parametry a zároveň je na konci řetězce připojeno `Modbus` zařízení, začnou se cyklicky přenášet data (do tabulky je zaznamenána každá změna). Obrázek A.10 znázorňuje několik cyklicky přenesených hodnot.

	Data	Status
1	0	0 (Unknown Status Byte)
2	1	8d (Quality: Good, Quality Status: Active Critical Alarm, Limits: Low limited)
3	2	89 (Quality: Good, Quality Status: Active Advisory Alarm, Limits: Low limited)
4	4	89 (Quality: Good, Quality Status: Active Advisory Alarm, Limits: Low limited)
5	23	8e (Quality: Good, Quality Status: Active Critical Alarm, Limits: High limited)

Obrázek A.10: Monitor - přenesená data

Po stisknutí tlačítka `Disconnect` je aplikace ukončena.

A.2 GSD soubor

```
#Profibus_DP GSD_Revision = 3
Vendor_Name = "Fieldbus International AS"
Model_Name = "T410"
Revision = "1.0"
Ident_Number = 0xa002
Protocol_Ident = 0
Station_Type = 0
FMS_supp = 0
Hardware_Release = "T410 1.01"
Software_Release = "T410 1.01.00"
Bitmap_Device = "T410"
31.25_supp = 1
45.45_supp = 1
93.75_supp = 1
MaxTsdr_31.25 = 100
MaxTsdr_45.45 = 250
MaxTsdr_93.75 = 1000
Redundancy = 0
Repeater_Ctrl_Sig = 0
24V_Pins = 0
Freeze_Mode_supp = 0
Sync_Mode_supp = 0
Set_Slave_Add_supp = 1
Min_Slave_Intervall = 250
Modular_Station = 1
Max_Module = 4
Max_Input_Len = 20 ; maximum Input Length
Max_Output_Len = 0 ; maximum Output Length
Max_Data_Len = 20 ; maximum In-Output Length
Slave_Family = 12
Max_Diag_Data_Len = 14
Max_User_Prm_Data_Len = 3
Ext_User_Prm_Data_Const(0) = 0x00, 0x00, 0x00
;------- Description of extended DP features: ---------------
;
DPV1_Slave = 1
C2_Read_Write_supp = 1
C2_Max_Data_Len = 128
```

```
C2_Read_Write_required = 1
C2_Max_Count_Channels = 1
Max_Initiate_PDU_Length = 52
C2_Response_Timeout = 4000
DPV1_Data_Types = 1
;-- Description of physical interface for async. and sync. transmis-
sion: --
Physical_Interface = 0 ; RS-485 Standard Copper
Transmission_Delay_45.45 = 0
Reaction_Delay_45.45 = 0
Transmission_Delay_93.75 = 0
Reaction_Delay_93.75 = 0
End_Physical_Interface
;
Physical_Interface = 1 ; IEC61158-2
Transmission_Delay_31.25 = 0
Reaction_Delay_31.25 = 0
End_Physical_Interface
;------- Description of device related diagnosis: --------------

;
Unit_Diag_Bit(16) = "Error appears"
Unit_Diag_Bit(17) = "Error disappears"
```