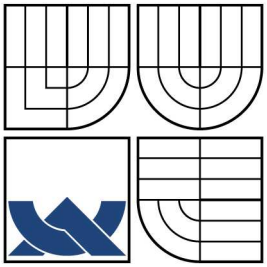
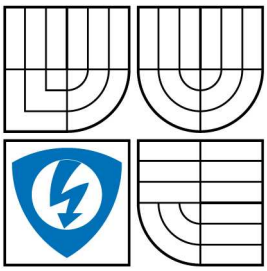


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘÍCÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

DATALOGGER PRO SBĚRNICI TYPU PROFIBUS

DATALOGGER FOR PROFIBUS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
BC. RADIM BRÁBLÍK

VEDOUCÍ PRÁCE
SUPERVISOR
DOC. ING. ZDENĚK BRADÁČ, PH.D.

BRNO 2010

ANOTACE A KLÍČOVÁ SLOVA

ANOTACE

Tato diplomové práce se zabývá vytvořením aplikace Datalogger pro sběrnici typu PROFIBUS. Popisuje komunikační standard Profibus, komunikaci PLC a PC přes sběrnici Profibus, API rozhraní k použité Profibus-PCI kartě, použitý programovací jazyk a vývojové prostředí, popis vytvořené aplikace Datalogger po grafické a programové části a na závěr popisuje testování vytvořené aplikace Datalogger.

KLÍČOVÁ SLOVA

Datalogger, Profibus, komunikační sběrnice, komunikace mezi počítačem a PLC, sběr a záznam dat, Visual Basic .Net, MS Visual Studio, PAPI rozhraní, Profibus-PCI karta, zpracování dat, Postmort

ANOTATION AND KEY WORDS

ANOTATION

This thesis deals with creating application Datalogger for PROFIBUS. Describes communications standard Profibus, PLC and PC communication via PROFIBUS, API used for Profibus-PCI card, used programming language and development tool, a description of graphic and software parts of developed application Datalogger and finally describes the testing of developed applications datalogger.

KEY WORDS

Datalogger, Profibus, communication bus, communication between the computer and the PLC, data collection and data recording, Visual Basic. NET, MS Visual Studio, PAPI interface, Profibus PCI card, data processing, Postmort

BIBLIOGRAFICKÁ CITACE

BRÁBLÍK RADIM.: Datalogger pro sběrnici typu profibus. Diplomová práce. FEKT VUT v Brně, 2010.

PROHLÁŠENÍ

„Prohlašuji, že svou diplomovou práci na téma Datalogger pro sběrnici typu Profibus jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne: **21. května 2010** ……

podpis autora

PODĚKOVÁNÍ

Děkuji vedoucímu semestrální práce Ing. Zdeňku Bradáčovi Ph. D. a konzultantům Ing. Radku Cabejškovi a Ing. Ondřeji Ondruškovi ze společnosti ABB za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **21. května 2010**

……
podpis autora

OBSAH

SEZNAM OBRÁZKŮ A TABULEK

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

ÚVOD

Tématem této diplomové práce je vytvoření aplikace Datalogger pro sběrnici typu PROFIBUS. Téma vzniklo ze zadání společnosti ABB, jejímž požadavkem bylo právě vytvoření dataloggeru komunikujícího s PLC této společnosti přes sběrnici Profibus.

Účelem úvodních částí je zejména seznámení se standardem sběrnice Profibus, jeho vlastnostmi a parametry sběrnice Profibus. Dále by měly objasnit a odůvodnit volbu komunikační sběrnice Profibus pro komunikaci PLC a počítače v aplikaci dataloggeru.

Další části mají za účel zejména prozkoumání a popsání komunikace mezi Profibus-PCI kartou a aplikací v prostředí MS Windows. A dále vybrání vhodného programovacího jazyka a vývojového prostředí pro vývoj aplikace dataloggeru.

Předposlední část se pak zabývá samotnou realizací aplikace Datalogger, popisem jejího grafického interface a programových částí.

Poslední část je určena testování aplikace a vyhodnocení provedených testů.

1. DATALOGGER

Tato kapitola je převzata z SP1 [1].

Datalogger je hardwarové zařízení či softwarová aplikace, která slouží k získávání a ukládání dat z automatizačního, výrobního, řídícího a jiného procesu.

Jednodušší verze dataloggerů umí z procesu pouze získávat data, jako jsou hodnoty ze snímače teploty nebo snímače otáček a tato data jsou schopny ukládat do paměti či souboru. Zpracování a vyhodnocení uložených dat se provádí až později jiným, k tomu určeným softwarem.

Pokročilejší verze dataloggerů jsou schopny data získaná z procesu nejen jednoduše ukládat, ale jsou schopny data také rovnou zpracovávat, analyzovat, vyhodnocovat a vytvářet z nich statistiky, grafy a trendy.

Jedním z příkladů analýzy a vyhodnocení dat je funkce POSTMORT (dodatečný rozbor). Tato funkce vyhodnocuje hodnotu sledované veličiny a v případě například překročení stanovených mezí nebo výskytu alarmu označí, případně uloží do samostatného souboru, sérii dat odpovídající určitému časovému úseku před a po události, která postmort vyvolala. Přitom časový úsek dat získaných po dané události může být získán s kratší periodou, což znamená, že data budou v časovém úseku po události zhuštěnější. Při následné analýze dat získaných postmortem lze zjistit jaké chování sledovaného systému předcházelo dané, často kritické, události a jak se systém choval po jejím výskytu. Při takovémto vyhodnocování je samozřejmě důležité, aby časový interval mezi jednotlivými hodnotami byl co nejkratší.

Data z procesu nejčastěji získáváme pomocí PLC (Programmable Logic Controller – Programovatelný logický kontroler) a k nim připojeným snímačům. PLC však většinou nedisponují dostatečnou pamětí aby mohly vykonávat funkci dataloggeru, proto je potřeba k nim datalogger připojit externě, což sebou ovšem přináší určité nároky, zejména na rychlost komunikace. Požadavek na rychlost komunikace je dán zejména tím, že PLC, konkrétně například mikrokontroler řady AC 800M zadavatelské společnosti ABB, nedisponuje pamětí potřebnou pro samotnou funkci dataloggeru, ale podle informací od společnosti ABB

ani bufferem, který by data po krátký časový interval shromažďoval a následně v jednom paketu odeslal externímu dataloggeru. PLC data z procesu odesílá, s určitou vzorkovací periodou, na nastavený komunikační port a pokud data nejsou jiným zařízením z toto portu vyčtena, jsou ztracena.

Z tohoto důvodu například zadavatelská společnost ABB zamítla pro komunikaci mezi PLC a dataloggerem použít Ethernet, který díky své nerealtimovosti nezajišťuje rychlé cyklické vyčítání dat z PLC. Pro rychlé cyklické vyčítání dat se však jako komunikační sběrnice hodí Profibus, který je přímo určen k rychlé cyklické komunikaci dat mezi účastníky komunikace.

2. PROFIBUS

Tato kapitola je převzata z SP1 [1].

2.1 HISTORIE SBĚRNICE PROFIBUS

[4][5] PROFIBUS (**PRO**ces **FI**eld **BUS**) je standardizovaná průmyslová komunikační sběrnice se standardizovaným komunikačním protokolem. Vzniku standardu Profibus předcházela snaha o vytvoření mezinárodního otevřeného modelu fieldbusu, tedy průmyslové komunikační sítě pro real-time distribuované řízení. Tato snaha selhávala jak v Americe u IEC (International Electrotechnical Commision) a ISA (Instrument Society of America), tak v Evropě, kde byla tato snaha podporována například DEK (Deutche Elektrotechnische Kommision) a to zejména díky odporu mnoha firem prosazujících svá vlastní řešení průmyslových sítí.

Proto se od roku 1985 začíná v Německu intenzivně pracovat na standardu Profibusu. Na jeho vývoji se podílelo také několik významných německých firem pod koordinací DBFT (Deuthe Bundesminstreum für Forschung und Technik). Práce na vývoji byly ukončeny v letech 1989/90 a Profibus se stal německou normou DIN 19245.

Díky vzniku standardu v Německu je Profibus velmi populární zejména v německy mluvících zemích, ale i v České republice je to jedna z nejpoužívanějších sběrnic v mnoha aplikacích. Jednou z firem, které Profibus podporují nejvíce je například Siemens, celkově se však o sběrnici Profibus stará organizace Profibus Foundation.

2.2 ZÁKLADNÍ VLASTNOSTI PROFIBUSU

[4][5][6] Profibus je sériová průmyslová komunikační sběrnice určená pro oblasti průmyslové automatizace a je založena na komunikačním modelu ISO/OSI (Open Systems Interconnection Reference model model – Referenční model propojování otevřených systémů). Standard Profibus je však upraven tak, že z modelu ISO/OSI využívá pouze 1., 2. a 7. vrstvu. Profibus se dělí na tři varianty,

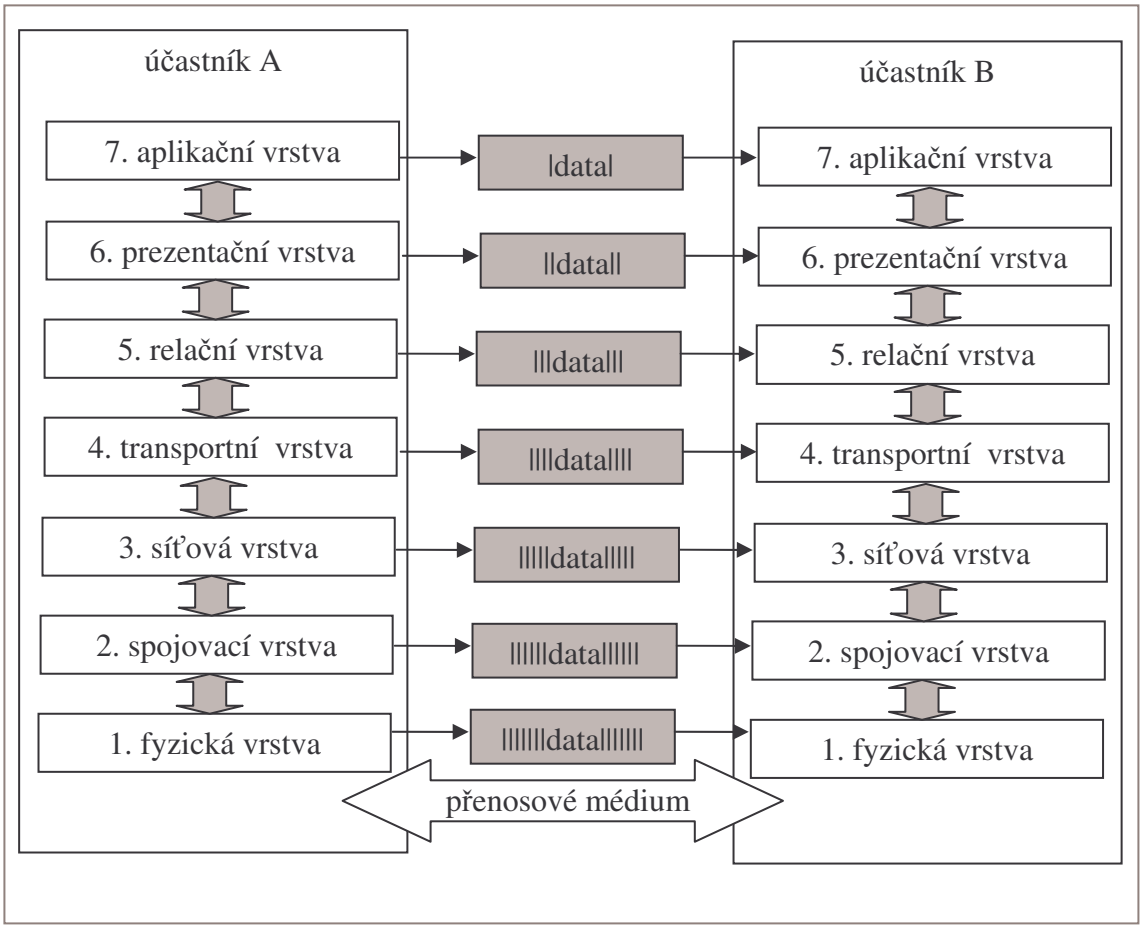
podle druhu využívané komunikace a to na Profibus-DP, Profibus-PA a Profibus-FMS.

Jako přenosová technologie se využívá stíněný kroucený pár s rozhraním RS-485, jehož maximální délka může být až 1200 m, případně optická vlákna, pro variantu používanou v prostředí s nebezpečím výbuchu se pak používá proudová smyčka. Topologie sítě Profibus jsou sběrnice, strom, hvězda a kruh, přičemž nejvíce preferovaná je první zmíněná: sběrnice. Přenosové rychlosti pro Profibus jsou od 9,6 kbit/s po 12Mbit/s pro varianty Profibus-DP a Profibus-FMS, varianta Profibus-PA pak využívá pouze rychlost 31,25 kbit/s, která je dána normou IEC 1158-2. Pomocí sítě Profibus lze propojit až 32 jednotlivých účastníků na jednom segmentu. Propojením více segmentů sítě, lze počet účastníků zvýšit až na 126. Zároveň lze propojením segmentů prodloužit maximální délku sítě 1200 m až na délku 10 000 m.

Účastníci, kteří jsou k síti Profibus připojeni se dělí na aktivní a pasivní stanice. Komunikace na sběrnici je pak řízena pomocí předávání tokenu mezi aktivními stanicemi, kdy pouze stanice vlastnící token může vysílat. Pasivní stanice může předávat svá data, pouze pokud je adresována aktivní stanicí vlastnící token.

2.3 REFERENČNÍ MODEL ISO/OSI A PROFIBUS

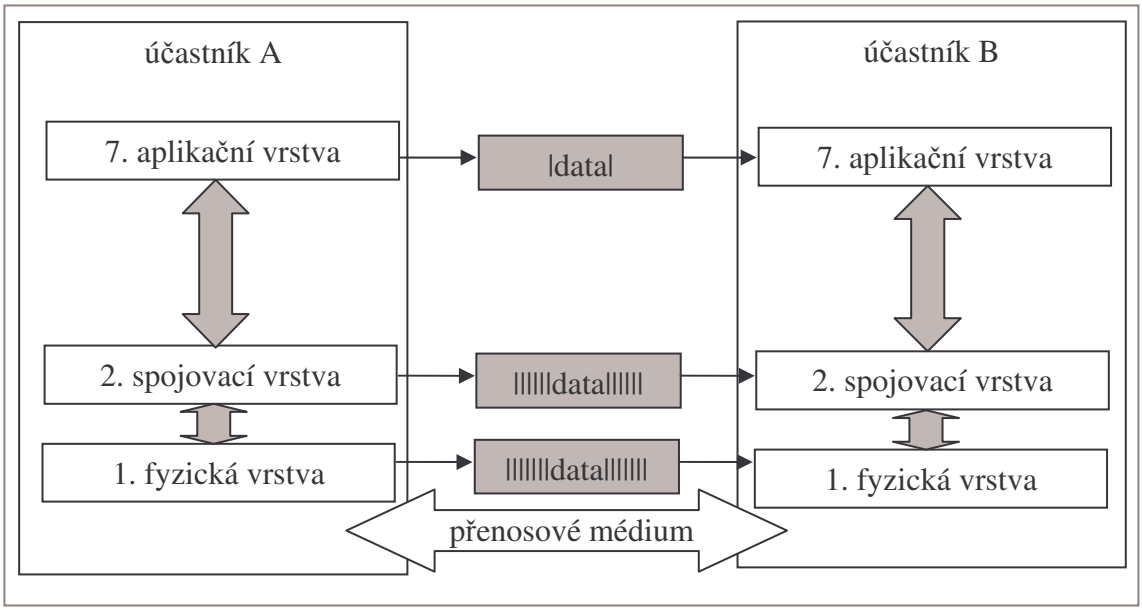
[7] Aby mohl být Profibus nasazen v heterogenním prostředí, tedy takovém prostředí kde se například na automatizaci nějakého procesu podílejí prostředky a zařízení různých výrobců, kdy řídící počítače dodal jeden výrobce, čidla výrobce druhý, akční členy další výrobce a ostatní prvky jsou rovněž od jiných výrobců, je Profibus postaven na referenčním modelu ISO/OSI. Což je model otevřené komunikace (OSI - Open System Interconnection – propojování otevřených systémů) vytvořený organizací ISO na počátku 80. let 20. století jako standard ISO 7498 určený k propojování právě heterogenních počítačových systémů.



Obr. 1 Referenční model OSI/OSI [7]

Model ISO/OSI je rozdělen na sedm vrstev, jak je možno vidět na Obr. 1 Referenční model OSI/OSI [7], z nichž každá má svou přesně definovanou funkci a služby. Zpráva, kterou si dva účastníci předávají mezi sebou, zpravidla vzniká v 7. aplikační vrstvě jednoho účastníka a sestupuje postupně po nižších vrstvách, kde se k původní zprávě nabalují data příslušných vrstev až k nejnižší 1. fyzické vrstvě, kde je pak přes přenosové médium zpráva předána 1. fyzické vrstvě druhého účastníka. U druhého účastníka zpráva stoupá přes jednotlivé vrstvy modelu, jsou z ní odstraňována data náležící příslušným vrstvám až v 7. aplikační vrstvě zůstane původní zpráva. Přestože k reálnému přenosu dat dochází pouze na úrovni 1. fyzické vrstvy, účastníci si mezi sebou na úrovni jednotlivých vrstev vytvářejí virtuální spoje. To umožňuje, aby spolu příslušné vrstvy komunikovaly aniž by něco věděly

o funkcích ostatních vrstev. Každá vrstva má přitom definovány dvě základní skupiny funkcí a to služby dané vrstvy a protokol pro komunikaci mezi účastníky přenosu. Funkce jsou však definovány pouze pro nejbližší nižší a nejbližší vyšší vrstvu účastníka a pak na stejnou vrstvu jiného účastníka.



Obr. 2 Model ISO/OSI standardu Profibus

Profibus má ale implementovány pouze vrstvy 1., 2. a 7. a to z toho důvodu, že pro potřeby sítí jako je Profibus nejsou ostatní vrstvy, tedy 3., 4., 5. a 6. potřeba. Chybějící funkce vynechaných vrstev jsou realizovány na 7. aplikační vrstvě.

2.3.1 Fyzická vrstva Profibusu

[4][5][6][8] Fyzická vrstva definuje fyzické spoje mezi účastníky, přenosové médium po stránce mechanických i elektrických vlastností, topologii sítě a rozhraní.

Jak již bylo zmíněno dříve, z hlediska topologie Profibus využívá čtyři typy topologií.

Topologie sítě pro Profibus:

- sběrnice
- strom
- hvězda
- kruh

Přičemž nejpreferovanější je topologie typu sběrnice.

Profibus v současnosti využívá tři **přenosové technologie**:

- RS-485 s krouceným párem pro Profibus DP a FMS
- Optické vlákno pro Profibus DP a FMS
- Proudovou smyčku pro Profibus PA

2.3.1.1 RS-485 s krouceným párem

Stíněný krouceny pár s rozhraním RS-485 je nejčastěji využívaným fyzickým médiem v Profibusu. Jeho výhodou je vysoká přenosová rychlost, která je vyžadována v real-time aplikacích ve kterých je Profibus hojně nasazován. Přenosová rychlost je definována od 9,6kbit/s po 12Mbit/s a je nepřímo úměrně závislá na délce vedení jak je vidět z Tab. 1 Závislost rychlosti na délce.

rychlost [kbit/s]	9,6	19,2	93,75	187,5	500	1500	12000
délka segmentu [m]	1200			1000	400	200	100

Tab. 1 Závislost rychlosti na délce

Tabulka udává rychlosti sítě pro jeden segment sítě, ten může mít maximální délku 1200 m. Na jednom segmentu může být připojeno maximálně 32 účastníků, což je maximum připojených stanic jaké dokáže vysílač budit. Pomocí opakovače lze k síti připojit další segment, na kterém může být připojených dalších 32 stanic. Opakovač je zařízení, které přijímá signál z jednoho segmentu sítě, zesiluje ho a posílá do druhého segmentu sítě. Tímto způsobem lze nejen prodloužit celkovou délku sítě až na 10 000 m, ale také zvýšit počet účastníků až na 126. Omezením je

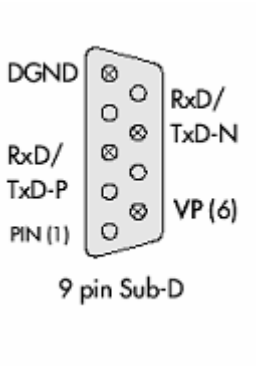
použití maximálně deseti opakovačů a omezení standardu Profibus na sedmibitovou adresu stanice, což nám dává rozsah adres 0 až 125 pro účastníky, dvě nejvyšší adresy jsou vyhrazeny pro speciální použití.

Jako vedení se, z na trhu nabízených typů kabelů A až D, pro RS-485 používá kabel typu A **Chyba! Nenalezen zdroj odkazů.**, stíněný kroucený pár, zakončený devíti-pinovým konektorem Sub-D, na Obr. 4 RS-485 zapojení konektoru Sub-D [4].

Základní parametry kabelu typu A:

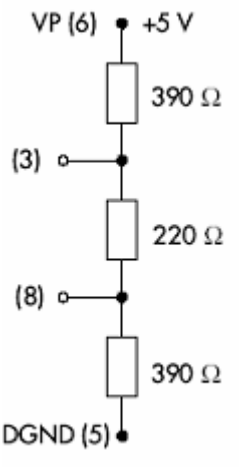
- impedance: 135 až 165 Ω
- kapacita: <30 pF/m
- odpor smyčky: 110 Ω/km
- Průřez vodiče: >0,34 mm^2

Obr. 3 Kabel typu A [8]



Obr. 4 RS-485 zapojení konektoru Sub-D [4]

Pro standard RS-485 je definována vstupní impedance obvodů R = 12 kΩ. Jelikož na vedení dochází k obousměrnému přenosu dat a na koncích vedení by mohlo docházet k odrazům je potřeba oba konce vedení zakončit aktivními terminátory, na Obr. 5 Aktivní terminátor [8]. Dalším důvodem pro zakončení vedení terminátory je definování úrovně na sběrnici ve chvíli kdy žádný z účastníků nevysílá a podmínka minimálního zatížení vysílače.



Obr. 5 Aktivní terminátor [8]

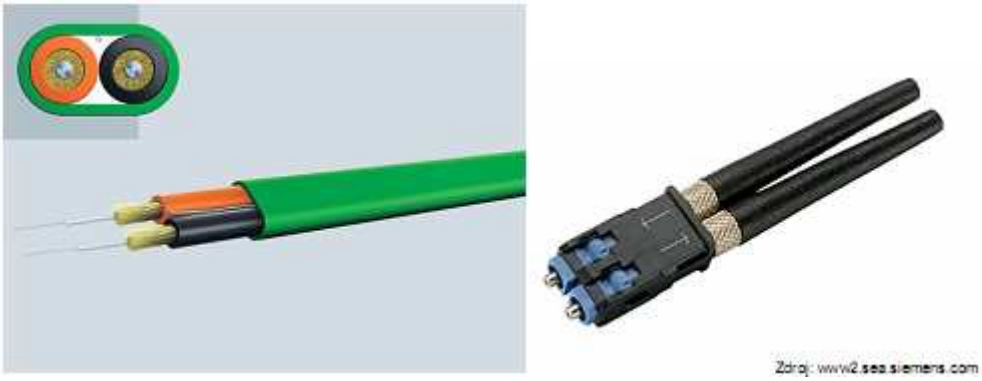
Sběrnice Profibus využívá asynchronního přenosu dat. Vlastní kódování je typu NRZ (Non Return to Zero – bez návratu k nule). Přenos probíhá po jedenácti-bitových znacích typu UART, který má jeden start bit, jeden stop bit, osm datových bitů a jeden bit sudé parity. Řazení bitů ve znaku je na Tab. 2 Řazení bitů ve znaku typu UART [6]

Start bit	8 datových bitů	bit sudé parity	stop bit

Tab. 2 Řazení bitů ve znaku typu UART [6]

2.3.1.2 Optické vlákno

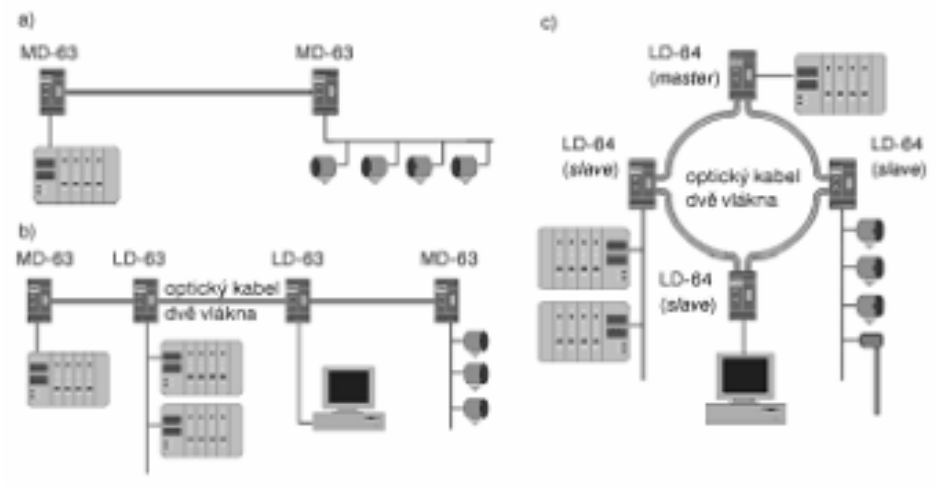
Optické vlákno se u Profibusu používá jako vhodná alternativa využitelná v prostředích se silným elektromagnetickým rušením, případně v případech, kdy je potřeba data přenášet vysokou rychlostí na větší vzdálenosti než umožňuje RS-485. Pomocí optického vlákna lze také vytvořit redundantní propojení, ve kterém RS-485 slouží jako záloha přenosu.



Obr. 6 Příklad optického kabelu [8]

Délka jednoho segmentu, tedy vzdálenost mezi dvěma optickými moduly, na kterou je optické vlákno možné využít je pro mnohavidové skleněné vlákno s průměrem jádra 62,5 mm a průměrem pláště 125 mm, 2 až 3 km, pro jednovidové skleněné vlákno s průměrem jádra 9 mm a průměrem pláště 125 mm, 15 km a pro umělohmotné jádro o průměru 0,98 mm a průměrem pláště 1mm až 90 km. Pro vlákno HCS (Hard Clad Silica – speciální vlákno na bázi Si), což je speciální optické vlákno na bázi křemíku s průměrem jádra 200 mm a průměrem pláště 300 mm je velikost segmentu 500 m.

Optická vlákna se používají v topologiích bod – bod, sběrnice, hvězda a kruh. Použitím převodníku mezi optickým vláknem a RS-485 lze realizovat různé varianty sítě, například optickou páteřní síť s metalickými odbočkami jako na části b) na Obr. 7 Topologie optických sítí [6].



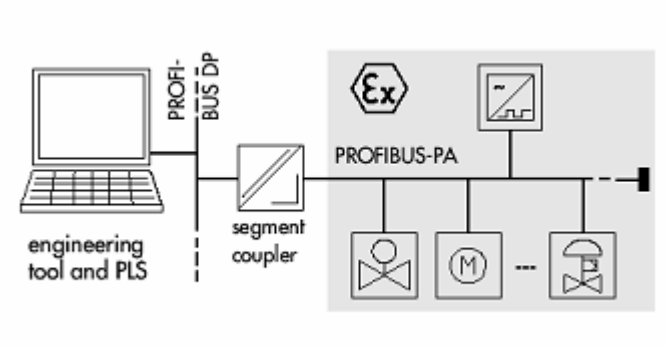
Obr. 7 Topologie optických sítí [6]

2.3.1.3 Proudová smyčka

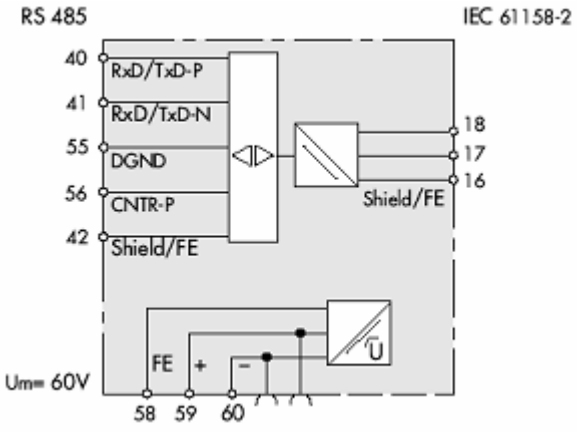
Proudová smyčka Profibusu je vytvořena na normě IEC 1158-2 pro prostředí s nebezpečím výbuchu. Stejně jako RS-485 využívá jako přenosové médium stíněný kroucený pár, ale rychlost je normou přesně stanovena na 31,25 kbit/s.

Stejně jako u RS-485 lze k jednomu segmentu připojit 32 účastníků, ale délka jednoho segmentu je omezena na 1900 m. Díky napětí od 9 V do 32 V, v prostředí s nebezpečím výbuchu pak do 15 V, lze prvky s nízkým odběrem napájet přímo ze sběrnice i při zachování jiskrové bezpečnosti, avšak počet účastníků, které lze napájet ze sběrnice se podle typu prostředí snižuje na 6 až 10. Vysílající účastník do sběrnice proud nedodává, ale při vysílání mění svůj proudový odběr.

Pokud se celá síť Profibus nenachází v prostředí s nebezpečím výbuchu, lze zbývající část sítě, tedy tu, která se nenachází v prostředí s nebezpečím výbuchu realizovat pomocí Profibus-DP, tedy pomocí technologie RS-485 (Obr. 8 Propojení RS-485 a proud. smyč. [8]). Propojení mezi těmito dvěmi přenosovými technologiemi se řeší pomocí vazební jednotky typu brána na Obr. 9 Vazební jednotka typu brána [8], která převádí RS-485 na jiskrově bezpečnou proudovou smyčku.



Obr. 8 Propojení RS-485 a proud. smyč. [8]



Obr. 9 Vazební jednotka typu brána [8]

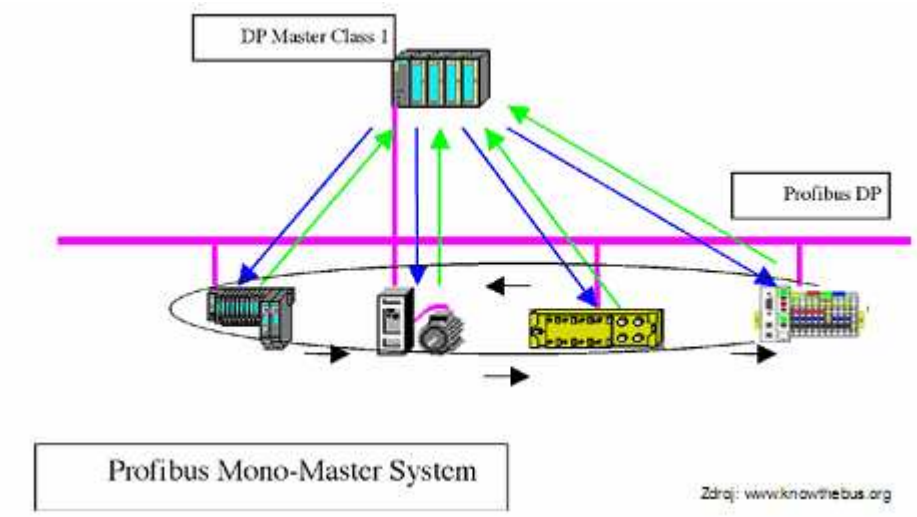
2.3.2 Linková vrstva Profibusu

[4][5][6][8] Linková vrstva Profibusu obsahuje protokol FDL (Field Bus Data Link), který zajišťuje jednotný přístup k přenosovému médiu a řízení komunikace mezi účastníky na sběrnici. Linková vrstva zajišťuje vytvoření zprávy do podoby bitového řetězce a kontrolní mechanismy přenosu dat.

Jelikož je Profibus sériová sběrnice, může v jeden okamžik vysílat vždy pouze jeden účastník, jinak by došlo ke kolizi dat. Řízení komunikace usnadňuje fakt, že účastníci na sběrnici jsou rozděleni na zařízení typu master a typu slave. Zařízení typu master pak řídí nebo se podílí na řízení komunikace na sběrnici,

zařízení typu slave čekají dokud nejsou adresována zařízením typu master a pak s ním komunikují data. K řízení kdy muže který účastník vysílat slouží dva způsoby.

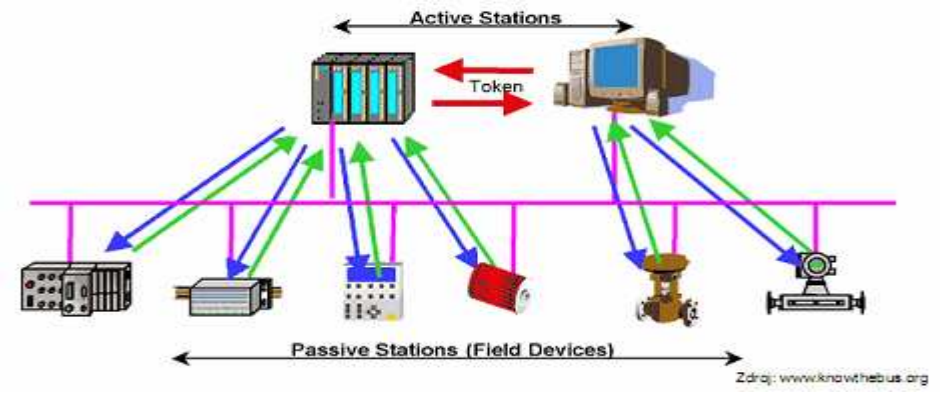
Prvním způsobem je komunikace master – slave, na Obr. 10 Komunikace Master - Slave [8]. U tohoto typu komunikace je na sběrnici připojeno pouze jedno zařízení typu master neboli single-master, například PLC a několik zařízení typu slave, například decentralizované vstupy a výstupy, tedy snímače a akční členy. Master řídí komunikaci a adresuje jednotlivá zařízení slave, s těmi pak komunikuje data. Tento způsob komunikace zajišťuje jednoduchý a rychlý cyklický přenos dat.



Obr. 10 Komunikace Master - Slave [8]

Druhým způsobem komunikace je Token Passing. Tento způsob komunikace se využívá je-li na sběrnici připojeno více zařízení typu master, neboli při multi-master komunikaci, kdy je potřeba zajistit aby vždy vysílala pouze jedna stanice. To zajišťuje protokol MAC (Medium Acces Control – řízení přístupu k mediu), který řídí přístup stanic typu master na sběrnici tak, že ve startovací fázi přiřazením adres účastníkům vytvoří logický kruh (Token Ring) podle kterého pak stanice typu master komunikují. Kromě toho MAC zajišťuje detekci poruch fyzického média, detekci chybných či vypnutých účastníků, přidávání nových účastníků v průběhu komunikace, předávání pověření ke komunikaci (token), chyby v pověření

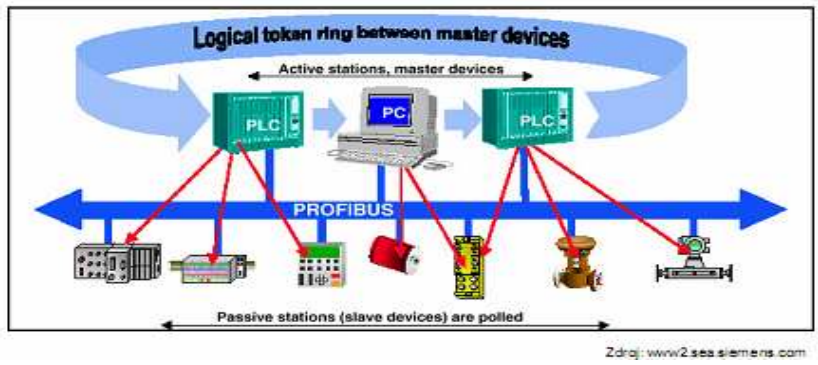
a podobně. Díky těmto službám lze za chodu sítě přidávat a odebírat účastníky, master i slave, MAC po odebrání nebo přidání tohoto účastníka detekuje a automaticky překonfiguruje logický kruh.



Obr. 11 Komunikace Multi - Master [8]

Předávání pověření ke komunikaci, tedy tokenu v logickém kruhu probíhá mezi stanicemi typu master podle jejich adresy a to vždy od nejnižší adresy směrem nahoru. Každý master zná adresy svých sousedů a tedy přesně ví, od koho token dostane a komu má po ukončení své komunikace token předat. V okamžiku kdy master dostane token, může zahájit komunikaci a vysílat či přijímat zprávy od stanic typu slave, ale i od ostatních masterů.

Časový interval pro vlastnění tokenu je možné nastavit, aby každý účastník měl dostatečný čas k provedení svých komunikačních úkolů. Čas, po který může master vlastnit token, a tím i maximální doba oběhu tokenu, je však striktně vymezena a to z důvodu zachování rychlého cyklického přenosu dat na sběrnici.



Obr. 12 Předávání tokenu v Profibusu [8]

Linková vrstva také nabízí služby vyšším vrstvám. A to tři asynchronní: SRD, SDN a SDA a jednu cyklickou: CSRD.

SRD (Send and Request Data with acknowledge) – Vyslání a vyžádání dat s potvrzením slouží pro výměnu dat mezi masterem a slavem. Master pošle data a slave potvrdí jejich přijetí potvrzovací zprávou. Pokud měl data připravená, pošle je v rámci potvrzovací zprávy, pokud data neměl, pošle v potvrzovací zprávě pouze indikaci o nedostupnosti dat. Toto celé je provedeno v rámci jednoho komunikačního cyklu.

SDN (Send Data with No acknowledge) – Vyslání dat bez potvrzení se používá v případě vysílání dat vybrané skupině (multicast) nebo všem účastníkům (broadcast). Potvrzení o přijetí dat se v tomto případě neposílá.

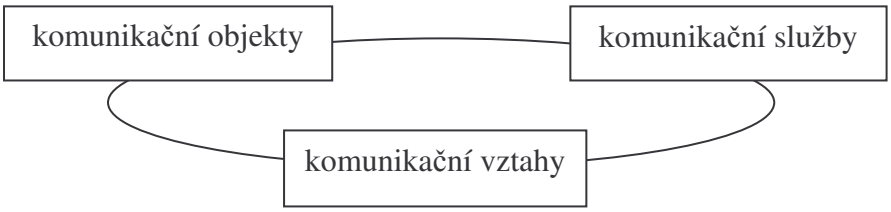
SDA (Send Data with Acknowledge) – Vyslání dat s potvrzením je podobné jako SRD, s tím rozdílem, že jsou vyslána data a zpět je posláno pouze potvrzení bez dat.

CSRD (Cyclic Send and Request Data) – Cyklické vysílání a vyžádání dat je jedinou cyklickou službou poskytovanou linkovou vrstvou. V podstatě se jedná o cyklickou variantu SRD, kdy aktivní stanice vysílá a přijímá data k a od stanic postupně podle daného seznamu.

2.3.3 Aplikační vrstva Profibusu

[4][5][6] Aplikační vrstva je nejvyšší vrstvou modelu ISO/OSI. Jelikož ve standardu Profibusu nejsou zastoupeny vrstvy 3. - 6. modelu ISO/OSI, je potřeba aby 7. aplikační vrstva mohla přistupovat přímo ke 2. linkové vrstvě. To umožňuje podvrstva LLI (Low Layer Interface – nízko úrovňové rozhraní), nižší ze dvou podvrstev vestavěných do 7. aplikační vrstvy, která promítá funkce 7. vrstvy do 2. linkové vrstvy a zároveň zastupuje potřebné funkce chybějících vrstev.

U Profibusu vzniká malý rozpor v omezeném rozsahu funkcí na nejnižší straně řízení, tedy u čidel a akčních členů a poměrně širokým rozsahem funkcí potřebných pro komunikaci mezi řídícími stanicemi, PLC, případně i PC (Personal Computer – osobní počítač) na vyšších vrstvách hierarchické struktury řízení. Tento rozpor řeší vyšší ze dvou podvrstev vestavěných do 7. aplikační vrstvy, FMS (Fieldbus Message Specification – vrstva specifikace zprávy). FMS je objektově orientovaná a reálné objekty jako hodnotu snímače otáček motoru nebo polohu otevření ventilu převádí na objekty komunikační, nad kterými pak definuje různé komunikační služby, díky kterým mohou tyto komunikační objekty využívat účastníci komunikace. FMS dále definuje komunikační vztahy, které účastníkům určují, které komunikační služby a objekty ostatních účastníků mohou využívat. Jak jsou komunikační objekty, komunikační služby a komunikační vztahy provázány ukazuje Obr. 13 Vazby mezi komunikačními objekty [4].



Obr. 13 Vazby mezi komunikačními objekty [4]

Převod dat na komunikační objekty dovoluje po Profibusu přenášet nejen data, ale i programy což umožňuje, pokud tedy dané zařízení tuto možnost podporuje, programovat a následně konfigurovat zařízení na dálku.

Ke komunikaci mezi sedmou aplikační vrstvou a samotnou aplikací, která přes sběrnici Profibus komunikuje, slouží API (Application Programming Interface - Aplikační programátorské rozhraní), což je soubor definující principy, datové struktury a protokoly, které usnadňují propojení mezi aplikací a aplikační vrstvou.

2.4 VARIANTY PROFIBUSU

[4][5][6][8] Profibus se dělí na tři základní varianty:

- Profibus – DP
- Profibus – FMS
- Profibus – PA

Tyto varianty se odlišují podle speciálního použití pro které jsou určeny.

2.4.1 Profibus – DP

Varianta Profibus – DP (Decentralized Periphery – decentralizované periférie) je nejjednodušší a nejrozšířenější variantou Profibusu. Je orientována hlavně na rychlost, může dosahovat maximální rychlosti komunikace 12Mbit/s, proto se využívá hlavně v automatizaci na nižší úrovni řízení pro rychlý přenos signálů z procesů od decentralizovaných jednotek k řídícím zařízením, kde je kladen velký důraz na krátkou dobu odezvy a zároveň nejsou potřebné složité komunikační funkce. Profibus – DP se používá hlavně při komunikaci master – slave nebo také single master, aby rychlost komunikace mezi účastníky nebyla omezována čekáním na opětovné přidělení povolení k přístupu na sběrnici.

Jak už bylo řečeno v jedné z předchozích kapitol, Profibus – DP využívá jako přenosové médium stíněný kroucený pár s rozhraním RS-485 nebo optické vlákno.

Protokol řídící komunikaci přes Profibus se postupně vyvíjel a v současnosti existuje ve verzích DP-V0, DP-V1, DP-V2.

- DP – V0 – má základní funkce Profibusu, tedy cyklickou komunikaci dat mezi masterem a slavem. Oprávnění pro komunikaci mezi více mastery pomocí tokenu.
- DP – V1 – je rozšířen o acyklickou komunikaci dat, přenos alarmů a fail-safe komunikaci.
- DP – V2 – umožňuje přímý přenos dat mezi zařízeními typu slave, časovou synchronizaci a asynchronní zahájení přenosu dat a možnost redundance zařízení i linek včetně kombinací.

2.4.2 Profibus – FMS

Varianta Profibus – FMS je určena zejména pro komunikaci složitějších systémů na vyšších úrovních řízení. K tomu poskytuje širokou skupinu služeb pro práci s daty, alarmy a programy. Větší rozsah a složitost funkcí omezuje přenosovou rychlost na 500 kbit/s, proto se tato varianta používá v aplikacích, které nejsou tak časově kritické. Používá se v komunikaci typu master – slave i token passing. Přenosové médium je shodné jako u varianty Profibus – DP, tedy stíněný kroucený pár a optické vlákno.

2.4.3 Profibus - PA

Varianta Profibus – PA (Process Automatization – automatizace procesů) byla vyvinuta a je speciálně určena pro automatizaci v prostředí s nebezpečím výbuchu. V podstatě se jedná o mírně rozšířenou variantu Profibus – DP pro jednodušší aplikace a řízení pomalých procesů. Nízká rychlost je dána použitím proudové smyčky na komunikačním médiu, ta sice odpovídá požadavkům na jiskrovou bezpečnost, za to však výrazně snižuje komunikační rychlost na konstantních 31,25 kbit/s.

3. VYUŽITÍ SBĚRNICE PROFIBUS PŘI SBĚRU A ZÁZNAMU DAT

Tato kapitola je převzata z SP1 [1]

3.1 DŮVODY POUŽITÍ SBĚRNICE PROFIBUS

Pokud si upřesníme požadavky naznačené v kapitole 1, tedy nejen rychlé cyklické vyčítání dat, ale i jednoduchost připojení do sledovaného procesu a široké rozšíření sběrnice v průmyslu, je z popisu v kapitole 2 zjistíme, že sběrnice Profibus všechny požadavky splňuje.

Pro aplikaci dataloggeru je naprosto postačující varianta sběrnice Profibus – DP. Potřebujeme totiž po sběrnici přenášet pouze data hodnot sledovaných veličin, vyšší funkce poskytované variantou Profibus – FMS v aplikaci dataloggeru nevyužijeme. Varianta Profibus – DP nám navíc umožňuje při propojení na vzdálenost do 100 m využít maximální rychlost sběrnice 12Mbit/s, navíc můžeme touto rychlostí data z PLC vyčítat cyklicky a tím minimalizujeme riziko ztráty dat jejich nevyčtením.

Protože varianta Profibus – DP jako přenosové médium využívá stíněný kroucený pár se Sub-D konektorem je zapojení do sítě a procesu velmi snadné. Implementovaný redukovaný referenční model ISO/OSI a jeho druhá linková vrstva s protokolem MAC pak dovoluje datalogger připojit na sběrnici Profibus i za chodu běžícího procesu.

Dalším argumentem pro použití Profibusu je jeho rozšíření a to nejen v německy mluvících zemích, ale v Evropě a i světě vůbec. Sběrnice Profibus se využívá v automatizaci, na výrobních linkách, v oblasti procesní automatizace, v řízení výroby i v distribuci energie, což nabízí široké možnosti uplatnění dataloggeru využívajícího komunikaci po této sběrnici.

3.2 PROPOJENÍ PLC A PC SE SBĚRNICÍ PROFIBUS

Vzhledem k tomu, že hardwarově realizovaný datalogger je v celku jednostranně zaměřené zařízení, jehož připojení na sběrnici, která je omezena na 32 účastníků na jeden segment sítě a 126 účastníků na celou síť, by v podstatě zbytečně zabralo místo pro jiného, možná důležitějšího, účastníka sítě. Proto je lepší ke sledování procesu využít softwarově realizovaný datalogger, který může běžet na počítači připojeném do procesu, přičemž na tomto počítači souběžně poběží i jiné aplikace.

Připojit počítač ke sběrnici Profibus lze jak nepřímo, tak přímo.

Nepřímá varianta je založena například na použití převodníku Profibus - DP / Ethernet. Tento převodník je přes sběrnici Profibus připojen k PLC a Ethernetem pak k počítači. Výhodou tohoto převodníku je, že lze sběrnici Profibus propojit přímo do sítě LAN (Local area network – lokální síť). Datalogger pak nemusí běžet na jednom konkrétním počítači, ale může běžet na kterémkoliv počítači v síti LAN. Nevýhodou tohoto převodníku je zpomalení komunikace samotným převodníkem a následná komunikace po Ethernetu sráží výhody sběrnice Profibus.

Přímá varianta je postavena na instalaci komunikační Profibus PCI karty přímo do počítače, takže celá komunikace mezi počítačem a PLC probíhá pouze přes sběrnici Profibus. To samozřejmě dovoluje plně využít vlastností sběrnice Profibus, přednostně cyklické komunikace dat rychlostí 12Mbit/s. Pro aplikaci dataloggeru byla zvolena právě tato varianta.

4. PROFIBUS-PCI KARTA

Tato kapitola je převzata z SP2 [2].

4.1 PROFIBUS-PCI KARTA

Pro projekt dataloggeru byla vybrána Profibus-PCI komunikační karta společnosti Softing s typovým označením PB-IF-1MS (dále označovaná pouze jako Profibus-PCI karta). Karta obsahuje jeden Profibus komunikační kanál, podporuje protokoly PROFIBUS DP - MASTER, PROFIBUS DP - SLAVE, PROFIBUS FMS, PROFIBUS FDL a maximální přenosovou rychlost 12 Mbit/s. Další parametry této Profibus-PCI karty jsou uvedeny v následující tabulce Tab. 3 Parametry Profibus karty [11].

	PROFIboard PCI (1 Kanál)
Protokol	
PROFIBUS-DP Master	ano
PROFIBUS-FMS	ano
PROFIBUS-FDL	ano
PROFIBUS-DP Slave	ano
Konektor	
Typ konektoru	9pin, Sub-D zástrčka, RS-485, galvanicky odděleno
Počet kanálů	1
Řadič	ASPC2 + SPC3
Přenosová rychlost	9,6; 19,2; 45,45; 93,75; 187,5; 500; 1500; 3000; 6000; 12000 KBit/s
Indikátory	LED diody (Aktivní kanál Master, datový tok na kanálu Slave)
Připojení k PC	
Typ připojení	PCI
Dual-Port-Memory	16 KByte
Přerušení	Plug and Play
Technické parametry	
Pracovní teplotní rozsah	0 °C ... +55 °C
Skladovací teplota	-20 °C ... + 70 °C
Vlhkost	< 90 %
Rozměry (mm)	174 x 107

Napájení	
Napájecí napětí	5V (+-5%)
Proud (mA)	700
Homologace	
CE	ano
FCC	ano
Podporované operační systémy	
Windows XP	ano
Windows 2000	ano
Windows NT 4.0	ano
Windows 9x	ano
VenturCom RTX	ano
LINUX	na vyžádání

Tab. 3 Parametry Profibus karty [11]

4.2 SDK

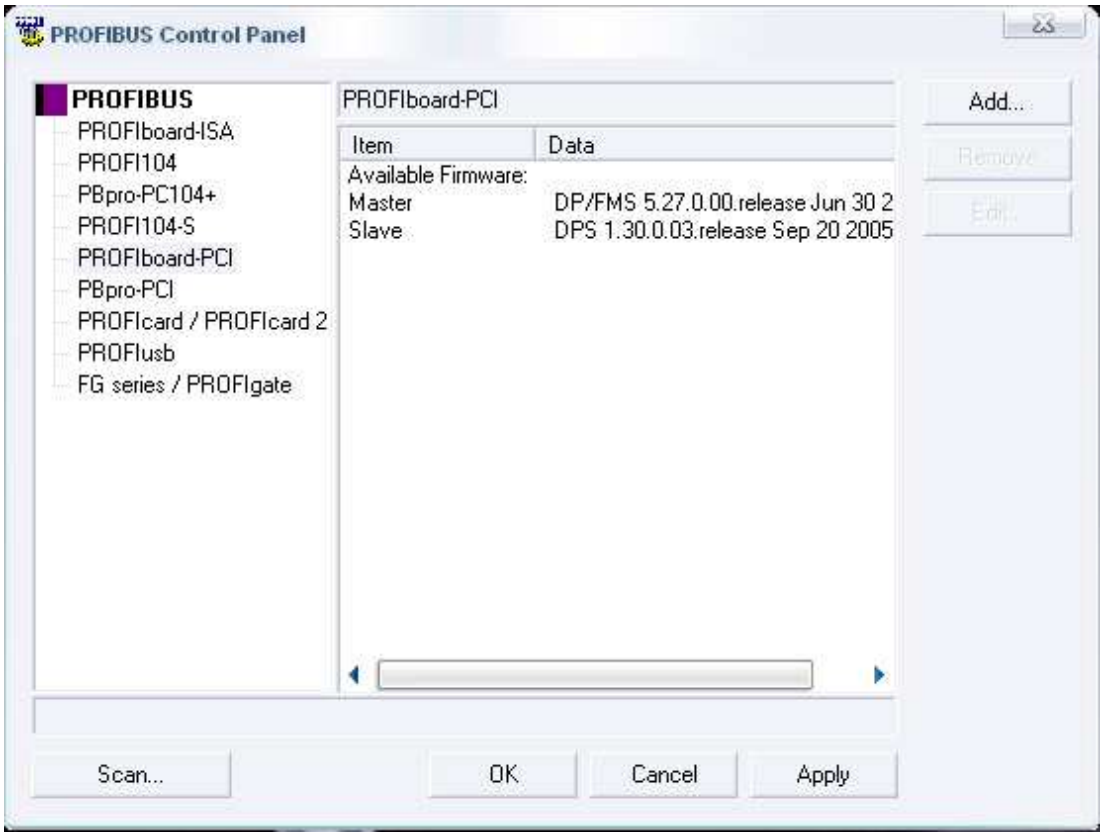
Společnost Softing k této kartě dodává také SDK (Software Development Kit – Nástroj pro vývoj softwaru), který obsahuje ovladač, manuál, ukázky programového kódu pro Win 2000, Xp, Vista32, RTX a PAPI ve verzi V5.44. Ukázky programového kódu ukazují jak využívat funkce a služby, které poskytuje PAPI rozhraní a Profibus-PCI karta.

4.3 DUAL-PORT PAMĚŤ (DP-RAM)

[10] Výměna dat mezi Profibus-PCI kartou a počítačem probíhá přes dual-port RAM (Random Access Memory – Paměť s náhodným přístupem), která je při načtení hardwarového ovladače Profibus-PCI karty automaticky namapována v 32 - bitovém adresním prostoru počítače.

4.4 ZÁKLADNÍ KONFIGURACE KARTY A OVLADAČE

[10] K základní konfiguraci Profibus-PCI karty slouží „PROFIBUS Control Panel“ jenž je instalován společně s ovladačem Profibus-PCI karty a jehož základní obrazovku můžeme vidět na Obr. 14 Printscreen Profibus Control panelu. Podrobnější popis „PROFIBUS Control Panelu“ je uveden v [10].



Obr. 14 Printscreen Profibus Control panelu

Při první konfiguraci, tedy po přidání karty do počítače a nainstalování ovladače, klikneme na tlačítko „Scan...“, tímto „PROFIBUS Control Panel“ vyhledá všechny profibusové karty nainstalované v počítači. Použitá Profibus-PCI karta odpovídá typu „PROFIboard-PCI“. Po nalezení Profibus-PCI karty se pomocí ovladače automaticky nakonfigurují nezbytné prostředky, jako například výše

zmíněná dual-port paměť, I/O (Input / Output – Vstupně / Výstupní) porty, přerušení, atd.

Po té co je „PROFIBUS Control Panel“-em automatický vybrán typ instalované karty, se „PROFIBUS Control Panel“ dotáže na symbolický „node“ (node – uzel) název, který aplikacím slouží k přístupu ke kartě.

Dále jsme dotázáni, zda má Profibus-PCI karta pracovat v režimu „DP/FMS Master“ nebo v režimu „DP Slave“. Datalogger bude k Profibus síti přepojen jako slave (slave – otrok), zvolíme tedy konfiguraci „DP Slave“.

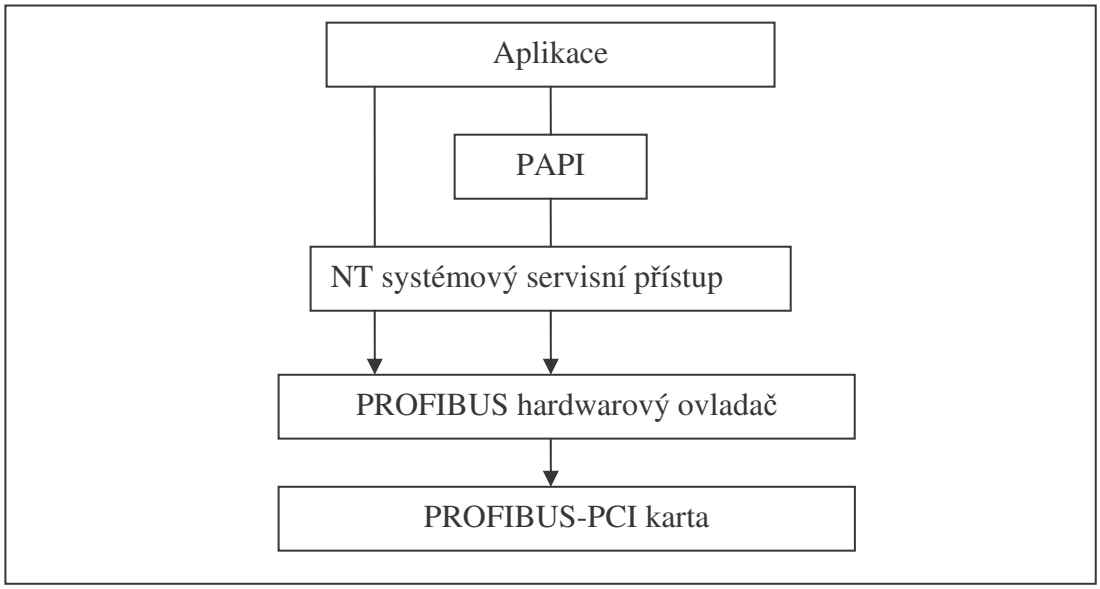
Nakonec jsme dotázání na číslo PCI (Peripheral Component Interconnect – Připojení periferních komponent) sběrnice a číslo PCI slotu ke kterému je karta připojena, tyto volby můžeme ponechat na automatické volbě. Ještě jsme dotázání na číslo kanálu, to je ovšem podstatné pouze pro dvoukanálové karty, proto také ponecháme na automatické volbě.

Tím je dokončena konfigurace Profibus-PCI karty v režimu „DP Slave“, která bude sloužit pro komunikaci dataloggeru a řídícího systému, přes sběrnici PROFIBUS. Podrobnější konfiguraci opět nalezneme v [10].

5. PAPI ROZHRANÍ

Tato kapitola je převzata z SP2 [2].

[10] PAPI (PROFIBUS Application Program Interface – Rozhraní pro programování aplikací [22]) rozhraní slouží jako rozhraní pro přístup ke všem funkcím Profibus-PCI karty a je kompletně obsaženo v PAPI dll (dynamic link library – dynamicky připojená knihovna) knihovně. Základní struktura přístupu je patrná na Obr. 15 Základní struktura přístupu aplikace ke kartě [10].



Obr. 15 Základní struktura přístupu aplikace ke kartě [10]

Z obrázku je patrné, že k funkcím Profibus-PCI karty lze přistupovat dvěma způsoby.

5.1 NÍZKO-ÚROVŇOVÝ PŘÍSTUP

[10] Jedním je nízko-úrovňový přístup, tedy přístup přímo přes I/O funkce operačního systému Windows jako k logickým zařízením. Tyto logická zařízení vytvoří ovladač Profibus-PCI karty automaticky při startu operačního systému.

Před použitím každého logického zařízení musí být toto zařízení nejprve otevřeno pro čtení nebo zápis a pak může být přistupováno přes čtecí, zapisovací

nebo řídící funkce. Po použití logického zařízení by mělo být pro čtení nebo zápis zase uzavřeno. Při otevření zařízení pro čtením, lze ze zařízení pouze číst, při otevření pro zápis lze pouze zapisovat, u řídících funkcí závisí na typu dané funkce.

Výhody tohoto přístupu jsou v tom, že nízko-úrovňový přístup poskytuje kompletní funkčnost ovladače zařízení a je možné provádět volání funkcí pro čtení a zápis asynchronně. Nevýhodou je, že je tento přístup mnohem složitější a tudíž náročnější na programování a výsledný program je daleko rozsáhlejší.

5.2 PŘÍSTUP PŘES PAPI FUNKCE

[10] Druhým přístupem je využití funkcí implementovaných v PAPI rozhraní. Tyto PAPI funkce za nás řeší všechny úkony a problémy, které bychom museli řešit při použití nízko-úrovňového přístupu, jako jsou otevření logického zařízení pro čtení a zápis, jednotlivé přistupující funkce a zavření logického zařízení.

PAPI funkce jsou rozděleny na dva mechanismy výměny dat mezi aplikací, Profibus-PCI kartou a kontrolerem. Prvním je send / receive (send / receive - pošli / přijmi) rozhraní, které využívá bloky požadavků pro servisně orientovanou výměnu dat, druhé je datové rozhraní umožňující rychlou cyklickou výměnu dat.

Výhodou tohoto druhého přístupu, tedy využití funkcí implementovaných v PAPI rozhraní je, že používá stejné funkce, jaké byly používány ve starším softwaru pro Profibus, proto se doporučuje při použití staršího software. Dále nabízí rozšířenou funkčnost, například dokáže vypočítat délku rámce. Nevýhodou je, že jeden proces dokáže obsluhovat pouze jednu kartu.

Pro datalogger jsem zvolil druhý přístup, tedy využití PAPI funkcí.

5.3 PAPI FUNKCE

[10] V PAPI rozhraní jsou implementovány následující funkce:

f_1	**profi_init**	Inicializuj rozhraní
f_2	**profi_end**	Ukonči rozhraní
f_3	**profi_snd_req_res**	Pošli servisní rámec
f_4	**profi_rcv_con_ind**	Přijmi servisní rámec
f_5	**profi_set_data**	Zapiš data
f_6	**profi_get_data**	Přečti data
f_7	**profi_set_dps_input_data**	Zapiš DP-Slave vstupní data
f_8	**profi_get_dps_input_data**	Přečti DP-Slave vstupní data
f_9	**profi_get_dps_output_data**	Přečti DP-Slave výstupní data
f_10	**profi_get_versions**	Přečti informace o verzi
f_11	**profi_get_serial_device_number**	Přečti sériové číslo zařízení
f_12	**profi_get_last_error**	Navrací poslední kód chyby rozhraní

5.3.1 Inicializace a ukončení

Logická zařízení sloužící k přístupu k Profibus-PCI kartě je potřeba nejprve inicializovat, respektive otevřít pro čtení nebo pro zápis aby k němu bylo možné přistupovat PAPI funkcemi. K tomu slouží funkce profi_init. Tato funkce musí být zavolána před ostatními PAPI funkcemi.

Hlavička této funkce je:

```
INT16 profi_init
        (
        IN UNSIGN8 Board,
        IN UNSIGN32 ReadTimeout,
        IN UNSIGN32 WriteTimeout
        );
```

Podrobněji je tato funkce popsána v [10].

V případě, že již nejsou PAPI funkce potřeba, je vhodné uvolnit logické zařízení pro další použití. K tomu slouží funkce profi_end.

Hlavička této funkce je:

```
INT16 profi_end
        (
        VOID
        );
```

Podrobněji je tato funkce popsána v [10].

5.3.2 Odesílání a příjem

K předávání řídících požadavků mezi Profibus-PCI kartou a programem zajišťují PAPI funkce pro odesílání a příjem servisních nebo řídících dat.

Funkce pro odesílání požadavků a odpovědí je profi_snd_req_res.

Hlavička této funkce je:

```
INT16 profi_snd_req_res
        (
        IN T_PROFI_SERVICE_DESCR* pSdb,
        IN VOID* pData,
        IN PB_BOOL Dummy
        );
```

Podrobněji je tato funkce popsána v [10].

Funkce, která slouží pro příjem potvrzení a indikování je profi_rcv_con_ind.

Hlavička této funkce je:

```
INT16 profi_rcv_con_ind
        (
        IN T_PROFI_SERVICE_DESCR* pSdb,
```

```
IN VOID* pData,
INOUT UNSIGN16* pDataLength
);
```

Podrobněji je tato funkce popsána v [10].

5.3.3 Data

Pro rychlou cyklickou výměnu dat mezi kontrolerem, Profibus-PCI kartou a aplikací slouží datové struktury umístěné v DP-RAM. Přístup k těmto strukturám opět zajišťují PAPI funkce. Funkce pro zápis nebo modifikaci dat v DP-RAM je profi-set-data.

Hlavička této funkce je:

```
INT16 profi-set-data
    (
    IN UNSIGN8 DataId,
    IN UNSIGN16 Offeset,
    IN UNSIGN16 DataSize,
    IN VOID* pData
    );
```

Podrobněji je tato funkce popsána v [10].

Funkce pro čtení dat z DP-RAM je profi_get_data.

Hlavička této funkce je:

```
INT16 profi_get_data
    (
    IN UNSIGN8 DataId,
    IN UNSIGN16 Offeset,
    INOUT UNSIGN16* pDataSize,
    OUT VOID* pData
    );
```

Podrobněji je tato funkce popsána v [10].

Funkce, která zapíše vstupní data přímo do struktury dané pro vstupní (*) data je profi_set_dps_input_data.

Hlavička této funkce je:

```
INT16 profi_set_dps_input_data
        (
        IN UNSIGN8 DataId,
        IN UNSIGN16 Offeset,
        INOUT UNSIGN16* pDataSize,
        OUT VOID* pData
        );
```

Podrobněji je tato funkce popsána v [10].

Funkce profi_get_dps_input_data čte aktuálně nastavená vstupní (*) data a s nimi spojený stav „DP-Slaveu“.

Hlavička této funkce je:

```
INT16 profi_get_dps_input_data
        (
        OUT UNSIGN8* pData,
        INOUT UNSIGN8* pDataLength,
        OUT UNSIGN8* pState
        );
```

Podrobněji je tato funkce popsána v [10].

Ke čtení aktuálních výstupních (*) dat slouží funkce profi_get_dps_output_data.

Hlavička této funkce je:

```
INT16 profi_get_dps_output_data
```

```
(
OUT UNSIGN8* pData,
INOUT UNSIGN8* pDataLength,
OUT UNSIGN8* pState
);
```

Podrobněji je tato funkce popsána v [10].

(*) Zde je potřeba zmínit jednu neočekávanou skutečnost. V hlavičkách datových funkcí a jejich popisu je označení pro vstupní a výstupní data, tedy „input_data“ a „output_data“. Standardně zavedená konvence je, že data, která vstupují do zařízení, se označují jako vstupní, tedy „input“ a data, která ze zařízení vystupují, se označují jako výstupní, tedy „output“. V případě výše uvedených funkcí je ovšem označení opačné. Data, která jsou z aplikace předávána kartě a dále řídícímu systému a jsou tedy výstupní, jsou označena jako „input“ a data, která jsou z kontroleru předávána kartě a dále aplikaci, jsou označena jakou „output“.

Tato záměna značení způsobila značné časové zpoždění projektu, jelikož se projevila už při konfiguraci komunikace (kapitola 6), kdy je potřeba nastavit jednotlivé rámce pro vstupní a výstupní data. Tyto rámce musí být nastaveny shodně jak na „masteru“ (kontroler), tak na „slavu“ (karta + aplikace), přičemž záleží i na pořadí jednotlivých rámců. Jelikož byly na „slavu“ nastaveny reálně vstupní data jako výstupní a reálně výstupní data jako vstupní, konfigurace „masteru“ a „slavu“ si neodpovídala, „slave“ tedy nepotvrdil konfigurační data a nedošlo tak k úspěšnému ukončení konfigurace komunikace.

5.3.4 Přídavné funkce

Funkce profi_get_versions slouží ke zjištění aktuální verze PAPI knihovny a verzi firmaware Profibus-PCI karty.

Hlavička této funkce je:

```
INT16 profi_get:versions
(
```

```
    OUT char* pPapiVersion,
    OUT char* pFirmwareVersion
    );
```

Podrobněji je tato funkce popsána v [10]

Funkce profi_get_serial_device_number slouží ke zjištění sériového čísla Profibus-PCI karty.

Hlavička této funkce je:

```
INT16 profi_get:versions
    (
    OUT UNSIGN32* pSerialDeviceNumber
    );
```

Podrobněji je tato funkce popsána v [10]

Funkce profi_get_last_error pouze navrací poslední chybové hlášení dříve volané PAPI funkce.

Hlavička této funkce je:

```
INT16 profi_get_last_error
    (
    VOID
    );
```

Podrobněji je tato funkce popsána v [10]

5.4 DP SLAVE SLUŽBY

[10] Služby jsou v podstatě datové struktury, které se předávají jako parametry při volání funkcí profi_snd_req_res a profi_rcv_con_ind. Protože bude datalogger k řídícímu systému připojen jako „slave“, ze služeb nabízených PAPI rozhraním se použijí služby pro „DP-Slave“. Tyto služby slouží k inicializaci, konfiguraci a ukončení „DP-Slave“ a k zjištění stavu a konfigurace „DP-Slave“.

5.4.1 Výběr služby

Službu, kterou chceme použít definujeme přes strukturu PROFI_SERVICE_DESCR [10]

```
struct PROFI_SERVICE_DESCR
        {
        USIGN16 comm_ref;
        USIGN8 layer;
        USIGN8 service;
        USIGN8 primitive;
        INT8 invoke_id;
        INT16 result;
        };
```

Její konfigurace a popis je v [10].

5.4.2 Inicializace

K inicializaci „DP-Slave“ slouží služba DPS_INIT_SLAVE [10]. Tuto službu nakonfigurujeme pomocí struktury DPS_INIT_SLAVE_REQ [10].

```
struct DPS_INIT_SLAVE_REQ
        {
        USIGN8 slave_add;
        USIGN8 min_tsdr;
        PB_BOOL auto_cfg_response;
        PB_BOOL auto_prm_response;
```

```
PB_BOOL auto_startup_inputs;
PB_BOOL sync_mode_supported;
PB_BOOL freeze_mode_supported;
PB_BOOL set_slave_add_supported;
USIGN8 max_input_data_len;
USIGN8 max_output_data_len;
USIGN8 max_cfg_data_len;
USIGN8 max_usr_prm_data_len;
USIGN8 max_ext_diag_data_len;
USIGN8 max_address_data_len;
USIGN16 ident_number;
USIGN16 user_watchdog_timeout;
USIGN8 reserved [4];
USIGN8 cfg_data_len;
USIGN8 enhanced_init_data_len;
USIGN8 cfg_data [DP_MAX_CFG_DATA_LEN];
USIGN8 enhanced_init_data [DPS_MIN_SERVICE_IF_LEN -
DP_MAX_CFG_DATA_LEN - 24];
};
```

Tato datová struktura určuje, jak má být konfigurován „DP-Slave“. Podrobný popis struktury a možnosti její konfigurace jsou popsány v [10].

5.4.3 Ukončení

Pokud již „DP-Slave“ není potřeba nebo je nutné jej například při úpravě konfigurace reinicializovat, je nutné „DP-Slave“ ukončit. K tomu je určena služba DPS_EXIT_SLAVE [10]. Volbu služby opět určuje struktura PROFI_SERVICE_DESCR [10]. Strukturou určující ukončení „DP-Slave“ je DPS_EXIT_SLAVE_REQ [10].

```
VOID T_DPS_EXIT_SLAVE_REQ;
```

Na této struktuře je zajímavé, že nemá žádné parametry. U této služby je rovněž možné vyžádat potvrzení o ukončení „DP-Slave“ a to opět pomocí struktury PROFI_SERVICE_DESCR [10] a struktury DPS_EXIT_SLAVE_CON [10].

```
struct _T_DPS_EXIT_SLAVE_CON
        {
        USIGN16 status;
        };
```

Popis a konfigurace této struktury jsou opět v [10].

5.4.4 Status

Velmi užitečnou službou je DPS_GET_STATUS [10]. Díky této službě můžeme zjistit operační stav Profibus-PCI karty, přiřazené identifikační číslo, celkovou velikost rámců pro vstupní a výstupní data, adresu přiřazeného „masteru“ i adresu „slavu“ samotného, přenosovou rychlost a další užitečné informace. Službu opět volíme přes strukturu PROFI_SERVICE_DESCR [10], kterou musíme nejprve nakonfigurovat jako požadavek na získání statusu, nastavení v [10] a k tomuto požadavku připojit strukturu DPS_GET_STATUS_REQ [10].

```
VOID    T_DPS_GET_STATUS_REQ;
```

Tímto jsme rozhraní informovali o požadavku na status, a nyní je potřeba status získat pomocí konfigurace struktury PROFI_SERVICE_DESCR [10] na požadavek potvrzení nebo indikace a vysláním tohoto požadavku společně se strukturou DPS_GET_STATUS_CON_IND [10].

```
struct DPS_GET_STATUS_CON_IND
        {
        USIGN16 status;
        USIGN8 slave_state;
        USIGN8 diag_state;
        USIGN16 ident_number;
```

USIGN8 number_inputs;
USIGN8 number_outputs;
USIGN8 slave_add;
USIGN8 non_volatile_slave_add;
USIGN8 master_add;
USIGN8 baud_rate;
PB_BOOL sync_enabled;
PB_BOOL freeze_enabled;
PB_BOOL clear_data;
PB_BOOL prm_await_response;
PB_BOOL cfg_await_response;
PB_BOOL await_startup_inputs;
USIGN8 reserved [16];
};

Výsledek tohoto druhého požadavku může být jak pozitivní, tak negativní, od čehož se odvíjí, jak bude po požadavku nastavena struktura DPS_GET_STATUS_CON_IND, jejíž podrobný popis je opět v [10].

„DP-Slave“ obsahuje ještě další služby, ty však nebudou do programu dataloggeru implementovány ani při testování, proto je nebudu uvádět, jejich popis je ovšem v [10].

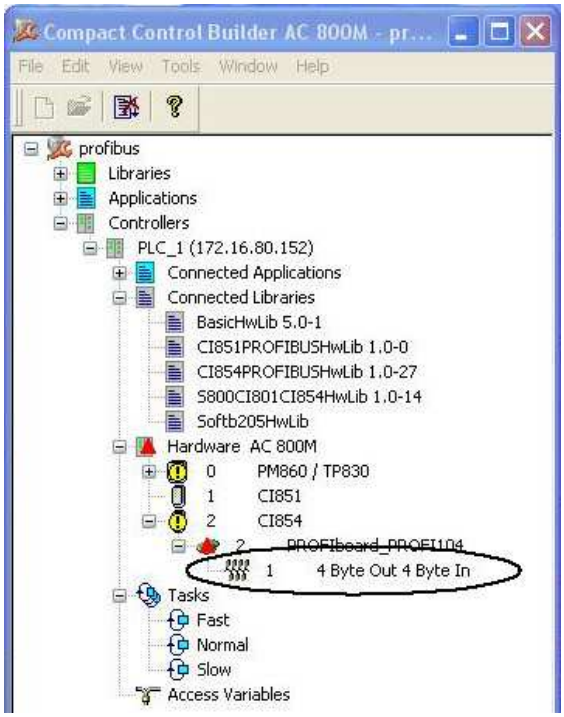
6. KOMUNIKACE DAT Z KONTROLERU DO APLIKACE

Tato kapitola je převzata z SP2 [2].

6.1 KONFIGURACE KONTROLERU

Aby byl kontroler schopný komunikovat s Profibus-PCI kartou, musí se také nakonfigurovat. Ke konfiguraci kontroleru slouží konfigurační soubory s příponou dsg, které jsou dodávány společně s Profibus-PCI kartou. Pro typ karty PROFIboard-PCI v režimu „DP-Slave“ se kontroler konfiguruje pomocí souboru „soft205.dsg“. Z tohoto souboru se pak přes program k nastavení kontroleru vytvoří soubor se základní hardwareovou konfigurací, pomocí níž se pak kontroler nakonfiguruje.

Při vytváření hardwareové konfigurace se nadefinují datové rámce, kterými se budou přenášet data. Jeden datový rámec může obsahovat 1-16 bytů nebo 1-16 wordů vstupních nebo výstupních dat. Byty/wordy se pak ještě spojí do jednotlivých proměnných určitých datových typů. Tyto rámce se pak v programu kontroleru připojí jako logická zařízení, kde se k jednotlivým proměnným přiřadí veličiny z procesu.



Obr. 16 Datový rámec připojený jako logické zařízení

Na Obr. 16 Datový rámec připojený jako logické zařízení vidíme na pozici 1 připojený jeden rámec (v černém kroužku) skládající se ze 4 výstupních bytových proměnných. Další rámce by se připojily na následující pozice.

6.2 KONFIGURACE PROFIBUS-PCI KARTY A PAPI ROZHRANÍ

Jak bylo napsáno v kapitole 5.3.1 Logická zařízení je potřeba nejprve inicializovat. Základní inicializace se provede pomocí funkce f_1. Poté je potřeba nakonfigurovat strukturu, 5.4.1 tato struktura slouží jako první parametr, čímž určíme, že pomocí funkce f_3 budeme provádět konfiguraci „DP-Slave“ a dokončovat tak jeho inicializaci. Druhým parametrem této funkce je struktura 5.4.2 při jejíž konfiguraci musíme dát pozor zejména na nastavení rámců. Ty totiž musí být nastaveny stejně a ve stejném pořadí jako rámce konfigurované v kontroleru, jinak se inicializace nedokončí a skončí chybovým hlášením.

Po dokončení je vhodné aktivovat časovač aplikace, který bude cyklicky zjišťovat status a konfiguraci karty a posílat a vyčítat data.

6.3 KOMUNIKACE DAT

Zjištění statusu a konfigurace karty se provádí cyklickým voláním funkce f_4 s vynulovanými strukturami 5.4.1 a 5.4.4 – potvrzení v parametrech. Pokud funkce proběhne v pořádku a byly k dispozici potvrzovací nebo oznamovací data, která jsou Profibus-PCI kartou generována automaticky, potvrzení nebo oznámení bylo zapsáno do struktury 5.4.4 – potvrzení, a jeho typ byl zapsán do struktury 5.4.1, tyto struktury byly před voláním funkce vynulovány a je tedy možné z nich status dekódovat.

Potvrzovací nebo oznamovací data je možné vyžádat i ve chvíli kdy nejsou vygenerována automaticky zavoláním funkce f_3 s parametrem - strukturou 5.4.1 nastavenou jako požadavek na status.

Posílat a vyčítat data je možné pomocí funkcí f_5, f_6, f_7, f_8 a f_9. U funkcí, které data posílají je samozřejmě nutné data nejdříve nastavit do pole proměnných, které se posílá jako parametr. U funkcí, které data přijímají je potřeba nastavit délku pole proměnných neboli bufferu, do kterého se mají data zapsat.

Pro ukončení logického zařízení je potřeba zavolat funkci f_2, která logické zařízení ukončí.

7. VÝVOJOVÉ PROSTŘEDÍ A PROGRAMOVACÍ JAZYKY

Tato kapitola je převzata z SP2 [2].

7.1 POŽADAVKY ABB NA VÝVOJOVÉ PROSTŘEDÍ

Zadavatelská společnost ABB měla několik požadavků na vývojové prostředí a programovací jazyk použitý k vytvoření dataloggeru..

Jedním z požadavků je, aby aplikace nebyla příliš závislá na verzi operačního systému, což je problém většiny starších programů, které na novějších verzích operačních systémů nefungují. Tento požadavek tedy klade nároky na použitý překladač programového kódu. Protože společnost ABB plánuje datalogger používat dlouhodobě, bylo by nepraktické a hlavně finančně velmi náročné zajišťovat kompatibilitu s novými verzemi operačních systémů nákupem nových překladačů. Čímž se dostáváme k dalšímu požadavku a tím je cena, respektive co nejnižší cena.

7.2 MS VISUAL BASIC STUDIO

7.2.1 Visual Basic .Net

[23][24][25] Volba programovacího jazyka nakonec padla na Visual basic .Net, což je nová generace programovacího jazyku Visual Basic, která je postavená na platformě .NET Framework. Visual Basic .Net jsem si zvolil proto, protože svůj předchozí program dataloggeru, který jsem vytvořil v rámci své bakalářské práce, byl zvětší části vytvořen v jazyce Visual Basic ve verzi 6, tudíž jsem měl s programováním ve Visual Basicu větší zkušenosti.

Dalším důvodem pro volbu Visual Basic .Net bylo, že datalogger je vyvíjen primárně pro operační systém Microsoft Windows XP a společnost Microsoft k jazyku Visual Basic .Net poskytuje vývojové prostředí MS Visual Basic .NET Studio [27] ve verzi Express Edition zdarma a to jak pro nekomerční, tak pro komerční použití.

7.2.2 .Net Framework

[24] Jak jsem uvedl výše Visual Basic .NET je postaven na platformě .NET Framework. Aplikace napsané nad touto platformou se při překladu předkompilují do jazyka MSIL, častěji se používá zkratka CIL (Common Intermediate Language – obecný přechodný jazyk), což je jazyk podobný assembleru, tedy strojovému kódu, a zabalí do takzvaného assembly (podrobně popsáno v [25]), což je soubor s příponou exe. Teprve spuštěním na klientském počítači se provede kompilace do strojového kódu a samostatně spustitelné aplikace. Strojový kód se přitom optimalizuje pro procesor a operační systém klientského počítače. Díky .NET Frameworku je tak aplikace nezávislá na systému (Windows XP, Windows 7, Linux).

Díky optimalizaci je rychlost takové aplikace srovnatelná s kompilovanou aplikací psanou v jazyce C/C++, na rozdíl od Visual Basicu 6 a ostatních skriptovacích jazyků, které pro svůj běh vyžadují interpretor, který kód čte řádek po řádku a překompilovává program až za jeho chodu.

Nevýhodou může být, že ke spuštění aplikace psané ve Visual Basic .Net je potřeba platforma .NET Framework, ta je ovšem dostupná zdarma v aktualizacích pro Windows.

7.3 MS VISUAL C++ STUDIO

Později se při vytváření aplikace zjistilo, že přímé propojení aplikace psané ve Visaul Basic .Net a PAPI dll knihovny, která zajišťuje přístup k Profibus-PCI kartě, pouhým voláním funkcí není možné. Popis tohoto problému je uveden v kapitole 8.1.

Protože jsem nechtěl opouštět zvolenou platformu .NET Framework a zvolený programovací jazyk Visual Basic .Net, bylo nutné najít cestu jak aplikaci a PAPI knihovnu propojit.

Jako řešení jsem nakonec zvolil vytvoření vlastní dll knihovny v programovacím jazyce C++, která se vloží mezi aplikaci a PAPI knihovnu, a která zapouzdřuje funkce PAPI knihovny a umožňuje je tak přes vloženou dll knihovnu

volat z Visual Basic .Net aplikace. Úpravy volání které bylo nutné udělat jsou popsány v kapitole 8.2.

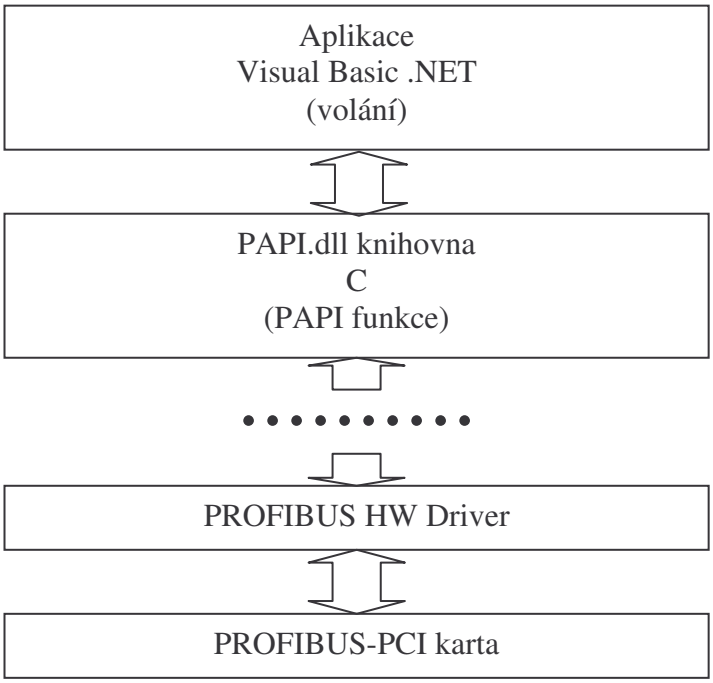
Programovací jazyk C++ jsem zvolil z podobných důvodů z jakých jsem zvolil jazyk Visual Basic .Net. Hlavním důvodem bylo, že vývojové prostředí MS Visual C++ Studio [28] existuje i ve verzi Express Edition, která je opět jak pro nekomerční, tak pro komerční účely společností Microsoft poskytována zdarma. Dalším důvodem bylo, že problém s propojením aplikace a PAPI knihovny popsaný v kapitole 8.1 se u C++ dll knihovny nevyskytne a propojení mezi aplikací ve Visual Basic .Net a C++ dll knihovnou půjde vyřešit vhodnou úpravou hlaviček funkcí PAPI dll knihovny popsanou v kapitole 8.2. Což se po následném odzkoušení potvrdilo.

8. PROPOJENÍ VISUAL BASIC .NET A PAPI DLL KNIHOVNY

Tato kapitola je převzata z SP2 [2].

8.1 PROPOJENÍ – VARIANTA 1

Původní myšlenkou bylo propojit aplikaci ve Visual Basic .Net s PAPI dll knihovnou dodanou k Profibus-PCI kartě přímo. Tedy do aplikace přilinkovat PAPI dll knihovnu a s pomocí deklarovaných hlaviček, funkce z dll knihovny jednoduše volat, stejně jako je tomu u demo příkladů, psaných v jazyce C, které jsou dodávány společně s ovladačem Profibus-PCI karty. Struktura na Obr. 17 Původně zamýšlená struktura programu.



Obr. 17 Původně zamýšlená struktura programu

Jak již bylo naznačeno v kapitole 7.3, toto jednoduché propojení nefungovalo tak jak by dle očekávání mělo. Funkce, které jsou z PAPI dll knihovny volány jsou

popsány v kapitole 5.3 a většina z jejich parametrů jsou struktury popsané v kapitole 5.4. Volání některých funkcí jako například profi_init a profi_end proběhlo v pořádku a úspěšně i s pozitivním výsledkem, které funkce při úspěšném vykonání navrací.

Problém však nastal u volání funkcí, které měly jako parametr strukturu, která obsahovala pole proměnných. To se projevilo už při inicializační konfiguraci „DP-Slave“ kdy je volána funkce profi_snd_req_res se strukturou DPS_INIT_SLAVE_REQ, která obsahuje pole proměnných pro konfiguraci rámců vstupních a výstupních dat. Tato funkce vždy vracela hodnotu /15/, která podle [10] znamená „nedostatečná velikost datového bloku“.

Obrátil jsem se tedy na společnost Softing AG, která je výrobcem Profibus-PCI karty s popisem problému a dotazem na příčinu této chyby. Po krátké emailové komunikaci mi ze Softing AG přišla následující odpověď [Email 1].

> Dear Mr. Radim Bráblík,
>
> I have talked with our product manager and he said that Visual Basic .NET is currently not supported and not tested with the PAPI interface.
> We plan to support VB.NET with the PAPI interface in the near feature.
> You can solve the problem by yourself if you know how to create native C code with your VB.NET application.
> If you have further questions let me know.
>
> Best regards
> Julia Sagi
> Technical Support

Email 1 Závěr emailové komunikace ohledně chybové návratové hodnoty

Z emailové komunikace jsem tedy usoudil, že přesně nevědí, v čem by mohl být problém, nicméně mi vnukli myšlenku zkontrolovat, jak vypadá parametr poté co jej obdrží volaná funkce v PAPI dll knihovně. Vytvořil jsem tedy malou dll knihovnu v C++, která měla stejnou hlavičku jako funkce v PAPI dll knihovně, ale předané

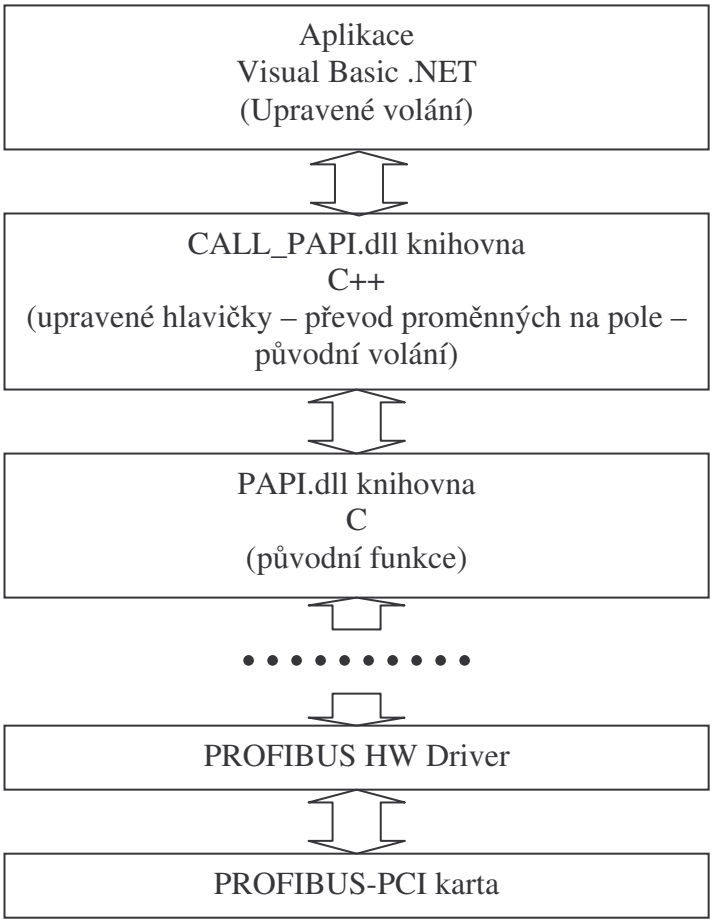
parametry pouze tiskla do textového souboru. Při následné kontrole těchto parametrů jsem zjistil, že hodnoty v poli proměnných v datové struktuře neodpovídají hodnotám, které byly do pole vloženy před voláním. Následně jsem ještě provedl pokus s vytvořenou funkcí, která měla v parametru pouze pole proměnných, a tyto hodnoty opět neodpovídaly.

Problém s neodpovídajícími si hodnotami v poli proměnných, předávaném jako parametr mezi Visual Basic .Net a C/C++ knihovnami dle mého názoru vzniká v rozdílném přístupu jednotlivých těchto jazyků k paměti. Zatím co C/C++ umožňují k paměti přistupovat přímo, Visual Basic .Net si paměť a její správu hlídá sám (podrobně popsáno v [26]), v čemž vzniká konflikt a adresa paměti předaná Visual Basic .Netem v C/C++ odkazuje na jiné místo v paměti než by měla.

Variantu přímého volání funkcí jsem tedy musel zavrhnout.

8.2 PROPOJENÍ – VARIANTA 2

K řešení problému popsaného v předchozí kapitole 8.1 mě trochu navedl email [Email 1] , který jsem obdržel od společnosti Softing AG při řešení nefunkčnosti propojení s PAPI dll knihovnou. Napadlo mě vytvořit vlastní dll knihovnu v jazyce C++, která by zapouzdřovala funkce PAPI dll knihovny. Struktura tohoto přístupu je na Obr. 18 Struktura programu po vložení zapouzdřovací C++ knihovny.



Obr. 18 Struktura programu po vložení zapouzdřovací C++ knihovny

C++ dll knihovna (dále už jen CALL_PAPI knihovna) vložená do struktury programu podle Obr. 18 Struktura programu po vložení zapouzdřovací C++ knihovny obsahuje funkce a modifikované struktury, jejichž vzory jsou v kapitolách 5.3, 5.4.

Jak bylo popsáno v předchozí kapitole 8.1 některé funkce původní PAPI dll knihovny používají struktury, jejichž součástí jsou pole proměnných. Tyto struktury bylo nutné modifikovat tak aby byla zachována funkčnost, ale při tom již struktury neobsahovaly pole. Jelikož bylo možné struktury modifikovat jak na straně Visual Basic .Net, tak na straně PAPI_CALL knihovny, byly v těchto modifikovaných strukturách pole proměnných vyměněny za jednotlivé proměnné.

Ve funkcích CALL_PAPI knihovny pak stačí před voláním původní funkce z PAPI dll knihovny překopírovat hodnoty z modifikované struktury do původní struktury a po vykonaném volání původní funkce opět překopírovat hodnoty z původní struktury do modifikované struktury.

9. APLIKACE DATALOGGER

9.1 POŽADAVKY ABB NA APLIKACI

Hlavními požadavky zadavatelské společnosti ABB na aplikaci dataloggeru byly schopnost záznamu velkého množství dat, co možná nejkratší perioda vyčítání dat z kontroleru a jednoduché vyhodnocení pro postmort funkci. Tyto požadavky odpovídají jednodušší variantě dataloggeru zmiňované v kapitole 1.

Rozhodl jsem se tedy při vývoji dataloggeru neimplementovat do něj funkce pro podrobnější zpracování, analýzu a vyhodnocení dat. Jednak k tomuto účelu velmi dobře poslouží software jako například Matlab nebo MS Excel, které jsou pro zpracování dat přímo určeny a na jejichž vývoji se podílí celé týmy čítající desítky až stovky lidí a jejichž vývoj trvá již řadu let a jednak by implementování všech potřebných funkcí výrazně protáhlo celý vývoj aplikace dataloggeru.

9.2 VLASTNOSTI APLIKACE

9.2.1 Data

9.2.1.1 Datové typy

Datalogger je schopný v jednom vyčítacím cyklu přijmout až 244 bytů dat. Data mohou mít strukturu byte nebo word a lze použít tyto datové typy:

1 Byte to 8 Bool
1 Byte to 1 DInt
2 Byte to 1 DInt unsigned
4 Byte to 1 DInt
1 Byte to 1 DWord
2 Byte to 1 DWord
4 Byte to 1 Real
1 Byte to 1 DInt signed

2 Byte to 1 DInt signed

nebo

1 Word to 16 Bool
1 Word to 1 DInt unsigned
2 Word to 1 DInt
1 Word to 1 DWord
2 Word to 1 Real
1 Word to 1 DInt signed

(datové typy a skládání z byte a word do těchto datových typů jsou převzaty z aplikace Control Builder společnosti ABB).

9.2.1.2 Datové rámce

Data se konfigurují do datových rámců po jednom až šestnácti bytech. Rámců může být maximálně 32. Do jednoho rámce se dá nakonfigurovat pouze jeden datový typ, výjimkou je ovšem případ, kdy například do bloku o velikosti 15 bytů chceme nastavit 4 bytový typ Real, pak se blok automaticky nakonfiguruje na 3x Real (4 byte) + 1x DWord (2 byte) + 1x DWord (1 byte) = 15 byte.

9.2.1.3 Počet sledovaných proměnných/signálů

Na konfiguraci závisí maximální počet sledovaných proměnných signálů. Počet sledovaných proměnných/signálů lze snadno vypočítat tak, že velikost přenášených dat v bytech v jednom cyklu vydělíme velikostí datového typu. Například tedy:

- pro Real (4 byte) je možné sledovat 61 signálů
- pro DInt (1 byte) je možné sledovat 244 signálů
- pro Bool (8x1 bit = 1 byte) je možné sledovat 1952 signálů

Bloky datových typů lze samozřejmě kombinovat, například lze nastavit tři bloky po 12 bytech datového typu Real, dva bloky po 1 byte datového typu Bool a čtyři bloky po 1 byte datového typu DInt, pak bychom sledovali 29 proměnných/signálů.

Při konfiguraci ovšem vždy záleží na pořadí bloků v konfiguraci.

9.2.1.4 Perioda vyčítání

Periodu vyčítání dat lze nastavit libovolně. Testování ukázalo, že nejlépe Datalogger pracuje s periodou k-násobku 125 ms (tedy periodou 125 ms, 250 ms, 500 ms, 750 ms, 1000 ms, 1125 ms, ...), pokud se perioda liší od k-násobku 125, perioda se prodlužuje o 3 až 15 ms. Prodloužení periody je způsobeno implementovaným vnitřním časovačem a vliv má také výkonová konfigurace PC.

U některých funkcí dataloggeru je omezení na minimální periodu vyčítání dat.

- pro zobrazení průběhů v grafu je to 100 ms
- pro postmort „Online postmort marks“ je to 50 ms
- pro postmort „Online postmort“ je to 1000 ms

9.2.1.5 Datové soubory

Data je možné ukládat ve dvou souborových formátech:

- **xls** – datový formát aplikace MS Excel, data oddělována tabulátorem
- **txt** – textový soubor, data oddělována středníkem

Formát xls má omezení na 65535 řádků, to znamená 63533 záznamů plus dva řádky hlavička. Aplikace si sama hlídá počet řádků, při naplnění souboru automaticky vytvoří nový soubor a pokračuje v ukládání dat do tohoto nového souboru. Maximální počet takto vytvořených souborů je 2^64, tedy 18446744073709551616.

9.2.2 Postmort

9.2.2.1 Typy postmortu

V aplikaci Dataloggeru jsou čtyři typy postmortu.

- **Postmort off** – Postmort je vypnutý, vyčítaná data se průběžně zaznamenávají do souboru, přičemž se ukládají všechna vyčtená data. Perioda vyčítání není omezena.
- **Turbo postmort** – do souboru se ukládají pouze postmort data, tedy pouze časový úsek vymezený před a po výskytu události (splnění postmort funkce). Data se ukládají až ve chvíli, kdy je vyčteno potřebné množství dat. Po dobu ukládání dat se vyčítání pozastaví, aby ukládání proběhlo rychleji a nedošlo při něm k poškození ukládaných dat.
- **Online postmort marks** – data se ukládají stejně jako v případě „Postmort off“, tedy všechny, při výskytu události se do souboru na konec příslušného řádku (do nového sloupce), na kterém došlo k výskytu události, zapíše číslo řádku/proměnné/signálu identifikující proměnnou/signál u které došlo k výskytu události. Minimální perioda vyčítání je 50 ms.
- **Online postmort** – ukládá do souboru všechny data stejně jako „Postmort off“ a zároveň do extra souboru ukládá časový úsek vymezený před a po výskytu události. Minimální perioda vyčítání je 1000 ms.

9.2.2.2 Ukládaný časový úsek

Maximální délka časového úseku dat, který lze postmortem uložit se určí ze vzorce (1).

$$x = \frac{T_{be} * 1000}{T_{per}} + \frac{T_{ae} * 1000}{T_{per}} - 2 \qquad \textbf{(1)}$$

x – počet záznamů, $\mathbf{T_{be}}$ – čas před událostí [s], $\mathbf{T_{ae}}$ – čas po události [s], $\mathbf{T_{per}}$ – perioda vyčítání [ms]

Přičemž maximální počet záznamů je $x_{max} = (2^{64} - 1)$.

9.2.2.3 Datové soubory postmort

Data je možné ukládat ve dvou souborových formátech:

- **xls** – datový formát aplikace MS Excel, data oddělována tabulátorem
- **txt** – textový soubor, data oddělována středníkem

Časový úsek dat se ukládá vždy do samostatného souboru.

9.2.2.4 Vyhodnocovací funkce

Proměnné/signály lze vyhodnocovat pomocí těchto funkcí

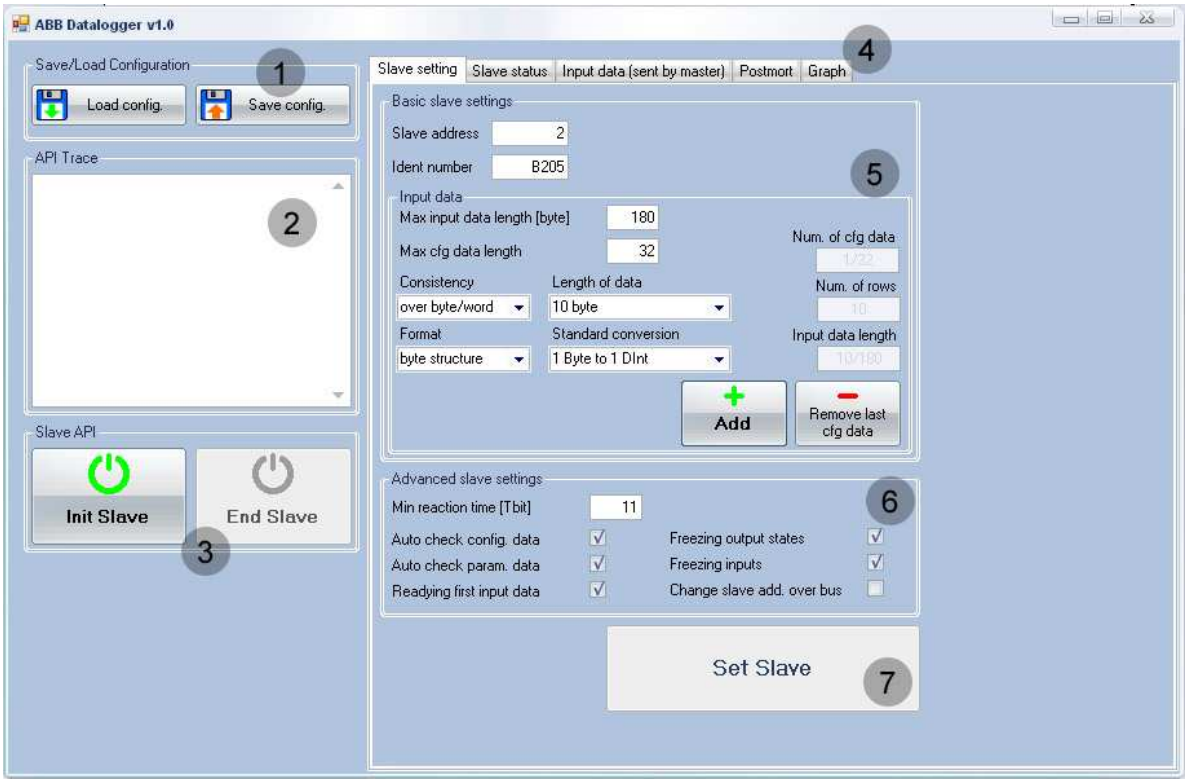
- **equal (=)** – vyhodnotí kdy je signál roven zadané hodnotě
- **equal with tolerance (=)** – vyhodnotí kdy je signál roven zadané hodnotě se zadanou tolerancí
- **less then (<)** – vyhodnotí kdy je signál menší než zadaná hodnota
- **more then (>)** – vyhodnotí kdy je signál větší než zadaná hodnota
- **in range (< , >)** – vyhodnotí kdy je signál uvnitř zadaného rozsahu
- **out of range (< , >)** – vyhodnotí kdy je signál mimo zadaný rozsah

9.2.3 Graf

Graf je velice zjednodušený a slouží pouze k orientační vizuální kontrole přenášených dat. Maximální počet zobrazených proměnných/signálů v grafu je 20. Minimální perioda vyčítání dat pro sledování dat v grafu je 100 ms.

9.3 VZHLED APLIKACE

9.3.1 Hlavní okno aplikace a záložka „Slave setting“



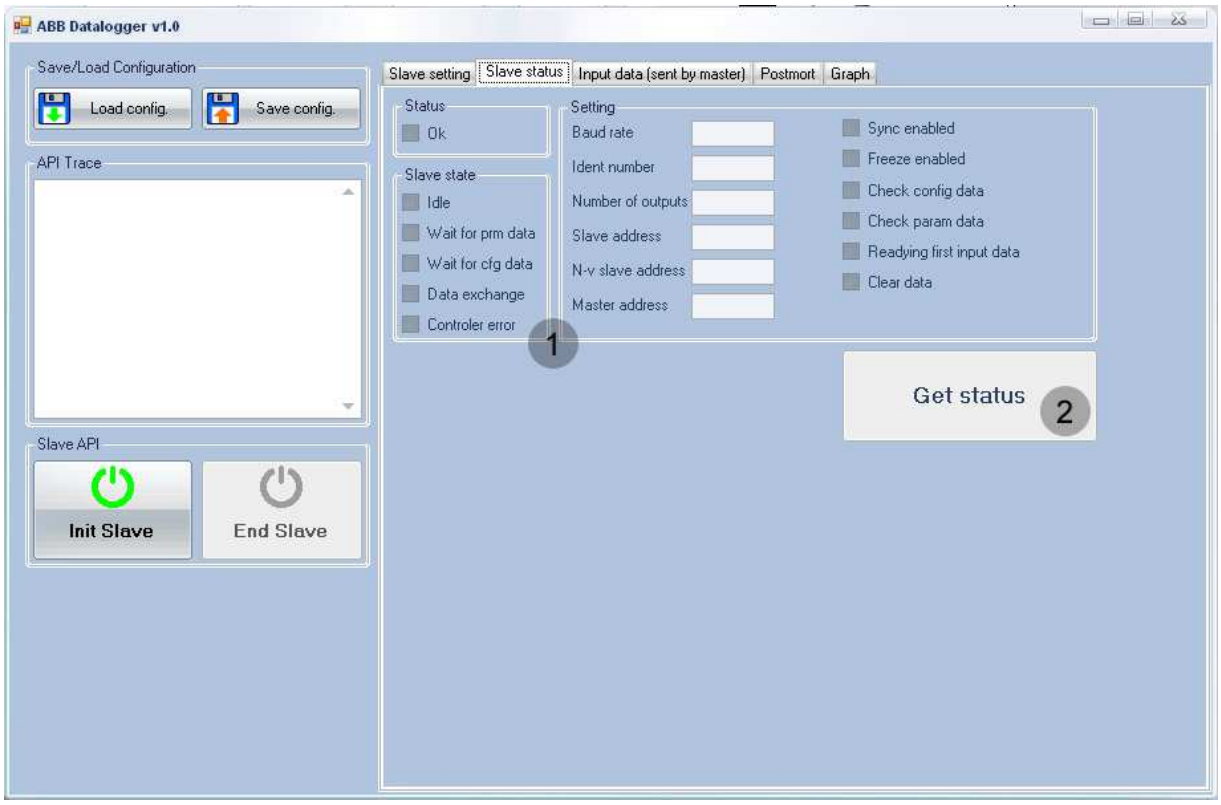
Obr. 19 Hlavní okno aplikace + záložka „Slave setting“

Na Obr. 19 Hlavní okno aplikace + záložka „Slave setting“ je vidět hlavní okno aplikace Datalogger. Hlavní okno aplikace je v podstatě vertikálně rozděleno v poměru přibližně 1:2. V levé části aplikace jsou dvě tlačítka „Load config.“ A „Save config.“(na obrázku označená 1), která slouží k nahrání nebo uložení konfigurace celé aplikace. Pod nimi se nachází textové pole „API Trace“ (na obrázku označené 2) do kterého se zapisují informace ohledně PAPI. Pod tímto polem se nachází dvě tlačítka „Init Slave“ a „End Slave“ (na obrázku označená 3), ta slouží pro inicializaci a ukončení PAPI.

V pravé části se nachází panel (na obrázku označený 4) rozdělený do pěti záložek „Slave setting“, „Slave status“, „Input data (sent by master)“, „Postmort“ a „Graph“. Na obrázku je zobrazena první záložka „Slave setting“. V horní části záložky jsou ovládací prvky pro základní nastavení (na obrázku označené 5), jako

například adresa Slave v síti Profibus, identifikační číslo gsd souboru a nastavení datových rámců, které bude ovšem popsáno později. V dolní části jsou ovládací prvky pro pokročilé nastavení Slave (na obrázku označené 6) a tlačítko „Set Slave“ (na obrázku označené 7) potvrzující provedená nastavení.

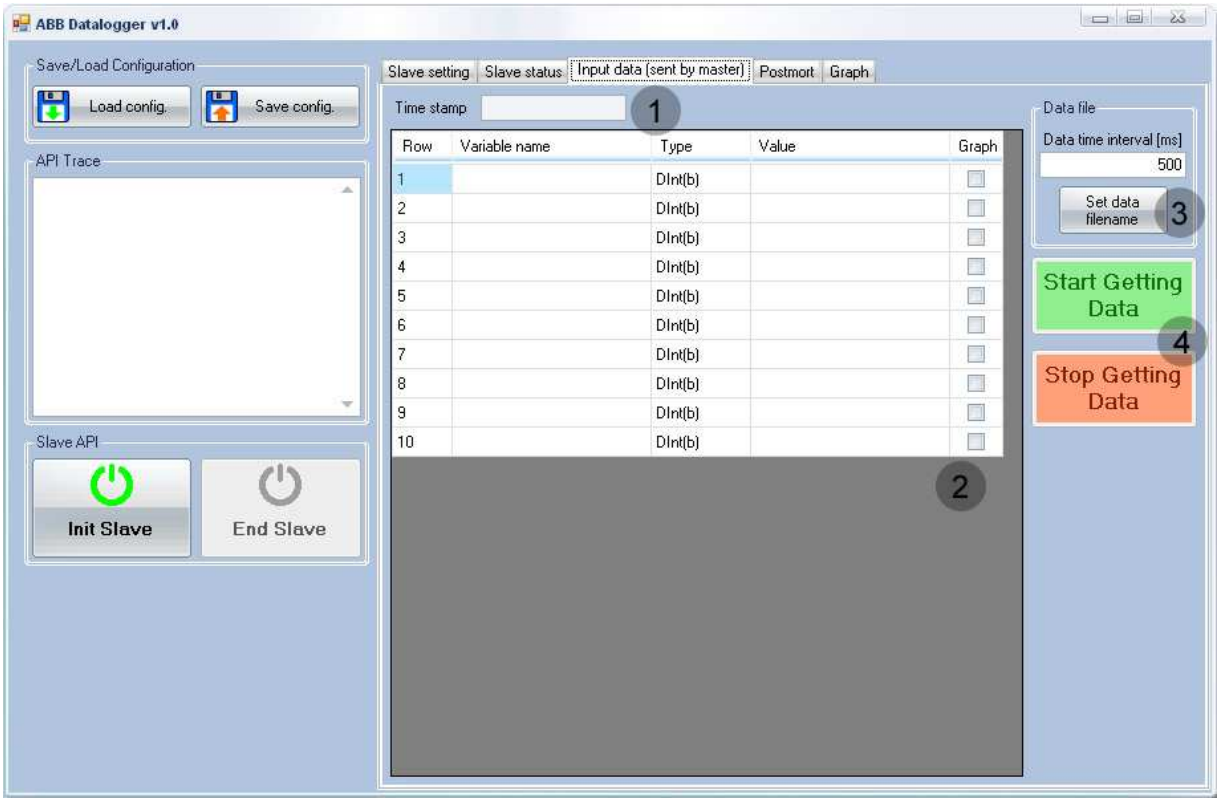
9.3.2 Záložka „Slave status“



Obr. 20 Záložka „Slave status“

Záložka „Slave status“ (Obr. 20 Záložka „Slave status“) slouží k zobrazení aktuálního stavu „Slave“. K zobrazení stavu slouží textová pole a zatrhávací políčka (na obrázku označena 1). „Status“ zobrazuje základní stav. „Slave state“ zobrazuje průběh konfigurace Slave, pro vyčítání dat z Masteru je potřeba aby se Slave dostal do stavu „Data Excange“. „Setting“ zobrazuje parametry, které byly nastaveny při konfiguraci Slave a potvrzeny, například adresu Slave v síti Profibus, počet výstupů, ale také adresu Masteru. Dále je na této záložce tlačítko „Get status“ (na obrázku označeno 2), které slouží k získání nebo obnovení stavu.

9.3.3 Záložka „Input data (sent by master)“



Obr. 21 Záložka „Input data (sent by master)“

Tato záložka (Obr. 21 Záložka „Input data (sent by master)“) slouží hlavně k zobrazení aktuálně vyčtených dat a k spouštění samotného vyčítání. V levém horním rohu této záložky je textové pole „Time stamp“ (na obrázku označené 1), které zobrazuje časovou značku aktuálně získaných dat, respektive systémový čas okamžiku, kdy byla data vyčtena z Masteru.

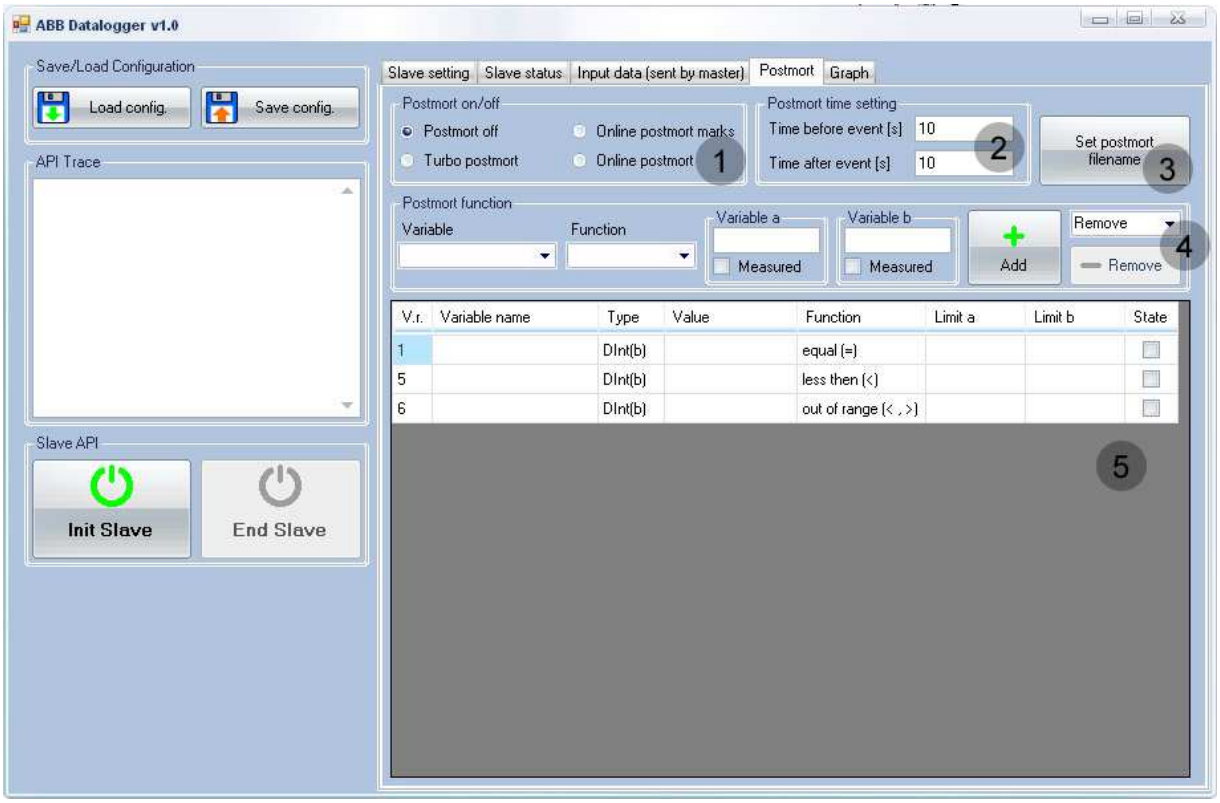
Pod tímto textovým polem je tabulka (na obrázku označena 2) pro zobrazení aktuálně vyčtených dat. Ta je rozdělena na pět sloupců. V prvním sloupci „Row“ je číslo řádku, které zlepšuje orientaci v tabulce a v proměnných při větším počtu sledovaných signálů. Druhý sloupec „Variable name“je určen pro názvy jednotlivých proměnných čili signálů. Ačkoliv se v podstatě celá tabulka konfiguruje automaticky při nastavování datových rámců na záložce „Slave setting“, názvy proměnných je potřeba zadat ručně. Ve třetím sloupci „Type“ jsou datové typy jednotlivých

proměnných. Písmeno v závorce na konci datového typu označuje, zda má daná proměnná strukturu byte nebo word. Čtvrtý sloupec „Value“ slouží k zobrazení aktuálně vyčtených hodnot jednotlivých signálů. Poslední, pátý sloupec „Graph“ obsahuje zatrhávací pole, jejichž zatržením se průběh proměnné zobrazuje v jednoduchém grafu na záložce „Graph“.

Dále je na této záložce textové pole „Data time interval [ms]“ a tlačítko „Set data filename“ (na obrázku označené 3). Textové pole slouží k nastavení periody vyčítání dat a tlačítko slouží k nastavení názvu a cesty k souboru či souborům do kterých se data budou ukládat.

Posledními prvky na této záložce jsou dvě tlačítka „Start Getting Data“ a „Stop Getting Data“ (na obrázku označena 4). Tato tlačítka slouží ke spuštění a zastavení vyčítání dat.

9.3.4 Záložka „Postmort“



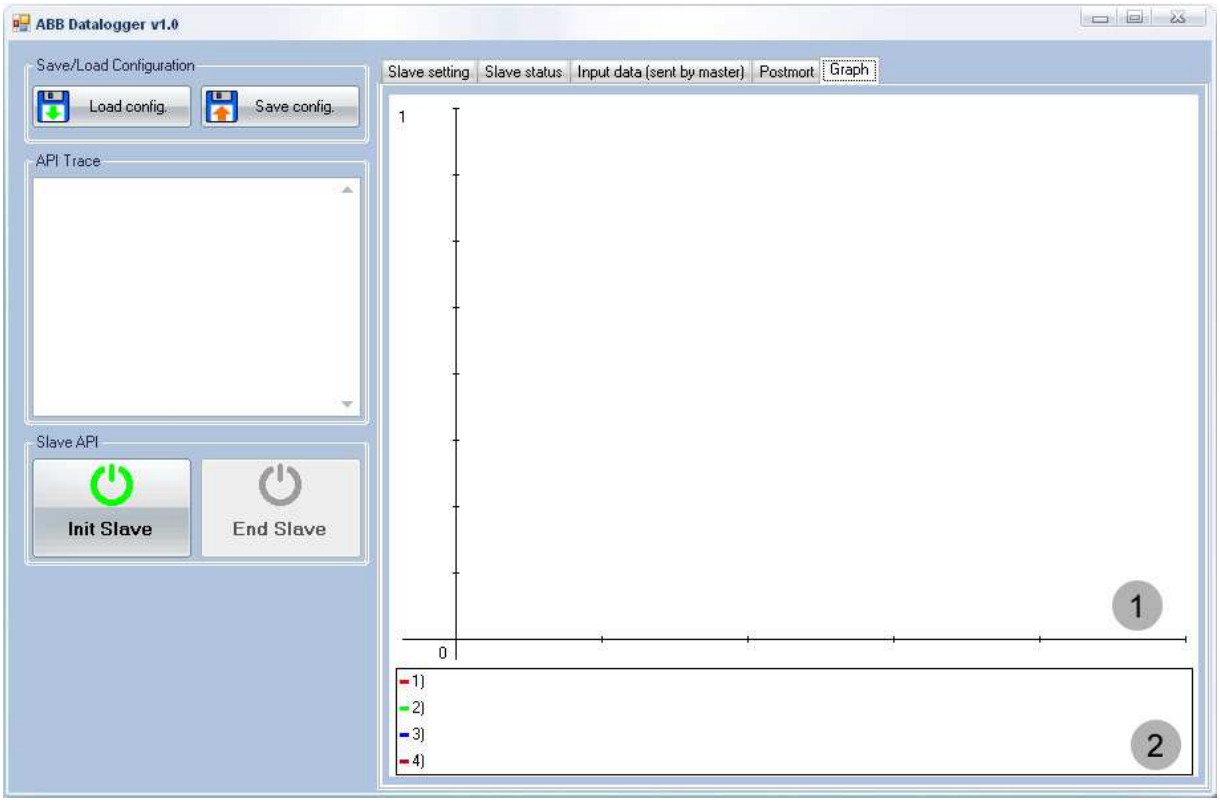
Obr. 22 Záložka „Postmort“

Záložka „Postmort“ (Obr. 22 Záložka „Postmort“) slouží k nastavení funkce Postmort. V levém horním rohu je možné jaký typ funkce Postmort chceme použít (na obrázku označené 1). Vedle je možné nastavit dobu záznamu před a po výskytu postmort události „Postmort time setting“ (na obrázku označené 2). „Time before event [s]“ nastavuje časový úsek před výskytem události, který se má uložit a „Time after event [s]“ nastavuje časový úsek, který se má uložit po výskytu události. V pravém horním rohu záložky se nachází tlačítko „Set postmort file“ pro nastavení cesty a názvu souborů do kterých se budou postmort data ukládat.

Pod těmito prvky se nachází sada ovládacích prvků „Postmort function“ (na obrázku označené 4). Tyto prvky slouží k nastavení postmort funkcí. „Variable“ slouží k výběru, která z proměnných se bude funkcí postmort vyhodnocovat. „Function“ slouží k výběru postmort funkce. „Variable a“ slouží k nastavení hodnoty, se kterou se bude signál pomocí postmort funkce porovnávat. Po zatržení „Measured“ je možné místo pevné hodnoty zvolit jednu ze sledovaných proměnných. „Variable b“ má v podstatě stejnou funkci jako „Variable a“, slouží například k nastavení tolerance, případně druhé meze intervalu.

V dolní části záložky je se pak nachází tabulka nakonfigurovaných postmort funkcí (na obrázku označena 5). Tato tabulka se skládá z osmi sloupců. V prvním sloupci „V.r.“ je číslo řádku sledované proměnné z tabulky na záložce „Input data (sent by master)“ pro zlepšení orientace v proměnných nebo pro případ, že by u dané proměnné nebyl napsán název proměnné. Ve druhém sloupci „Variable name“ je název sledované proměnné, který se při nastavení funkce automaticky zkopíruje z tabulky „Input data (sent by master)“. Ve třetím sloupci je datový typ sledované proměnné, opět automaticky kopírovaný. Do čtvrtého sloupce „Value“ se zapisuje aktuální hodnota sledované proměnné. V pátém sloupci „Function“ je zvolená postmort funkce. V šestém „Limit a“ a sedmém „Limit b“ sloupci jsou hodnoty, případně názvy proměnných se kterými se bude sledované hodnota porovnávat. V posledním sloupci je pak zatrhávací pole, které indikuje zda byla podmínka funkce splněna.

9.3.5 Záložka „Graph“



Obr. 23 Záložka „Graph“

Na této poslední záložce „Graph“ (Obr. 23 Záložka „Graph“) je zjednodušený graf (na obrázku označen 1), sloužící k zobrazení průběhu sledovaných hodnot. Graf zobrazuje posledních 550 hodnot sledovaných veličin a maximum y-ové osy odpovídá nejvyšší načtené hodnotě od počátku měření.

Pod grafem se nachází legenda (na obrázku označena 2). Číslo před názvem odpovídá číslu řádku, na kterém se proměnná nachází v tabulce „Input data (sent by master)“. Pro přehlednost se názvy proměnných v legendě zkracují na deset znaků.

9.4 PROGRAM

9.4.1 Knihovna call_papi.dll

Vzhledem k problému, který jsem popsal v kapitole 8, bylo potřeba vytvořit knihovnu v jazyce C++, která by tvořila jakýsi most mezi PAPI knihovnou a samotnou aplikací. Pro správnou funkci Dataloggeru je potřeba pouze pět funkcí z PAPI rozhraní. Funkce *profi_init* pro inicializaci PAPI rozhraní, funkce *profi_end* pro jeho ukončení, funkce *profi_snd_req_res* pro odesílání požadavků na PAPI rozhraní, funkce *profi_rcv_con_ind* pro příjem potvrzení a výsledků požadavků a funkci *profi_get_dps_output_data* pro vyčítání sledovaných signálů.

Pro tyto funkce jsem vytvořil zapouzdřovací funkce aby je bylo možné po úpravě předávaných parametrů volat z aplikace Dataloggeru a vytvořil z nich dll knihovnu call_papi.dll. Hlavičky zapouzdřovacích funkcí obsažených v call_papi.dll knihovně vypadají následovně.

short call_profi_init();

short call_init_slave_req(_T_DPS_INIT_SLAVE_REQ_modified Isr);*

short call_get_status_req(void);

short call_profi_rcv_con_ind(T_PROFI_SERVICE_DESCR pSdb, mi_buffer* pmi_buffer, unsigned short* pDataLength);*

short call_get_dps_output_data(mi_buffer pOutData, unsigned char* pOutDataLength, unsigned char* pOutState, data_time_stamp* GetDataTime);*

short call_profi_end(void);

Jak je vidět, pro přehlednou orientaci ve funkcích jsem zachoval názvy původních funkcí, s tím, že jsem do názvu zapouzdřovací funkce na začátek přidal „call_“. Výjimku tvoří funkce *call_init_slave_req* a *call_get_status_req*, které obě zapouzdřují funkci *profi_snd_req_res*, ale každá s jinými parametry.

9.4.1.1 Funkce call_profi_init

Tato funkce zapouzdřuje funkci *profi_init*, která inicializuje PAPI rozhraní. K zapouzdření této funkce nebylo potřeba žádných úprav, proto zapouzdřovací

funkce obsahuje pouze volání funkce *profi_init*, jejíž návratová hodnota je předána dále opět jako návratová hodnota zapouzdřovací funkce.

9.4.1.2 Funkce call_init_slave_req

Tato funkce zapouzdřuje funkci *profi_snd_req_res*, která slouží k odesílání požadavků na PAPI rozhraní, přičemž význam jednotlivých požadavků závisí na službách (kapitola 5.4) a jejich nastavení, které funkci předáme. V tomto případě je funkce použita k nastavení parametrů Slave, jako například adresa Slave v síti Profibus, maximální délka vstupních a výstupních dat, nastavení datových rámců a podobně, pomocí příslušného servisního požadavku (kapitola 5.4.1) a služby inicializace (kapitola 5.4.2). A právě například nastavení datových rámců je ve službě tvořené datovou strukturou uloženo v datovém poli, což je problém popsaný v kapitole 8. Parametry pro nastavení Slave jsou tedy zapouzdřovací funkci *call_init_slave_req* z aplikace Dataloggeru předány v modifikované datové struktuře, která neobsahuje pole, ale parametry má uloženy v jednotlivých proměnných. Zapouzdřovací funkce nejprve z modifikované struktury parametry pro Slave překopíruje do nemodifikované struktury a pak provede volání funkce *profi_snd_req_res* jejímž parametrem je nemodifikovaná struktura s překopírovaným nastavením pro Slave. Návratová hodnota funkce *profi_snd_req_res* je opět předána jako návratová hodnota zapouzdřovací funkce.

9.4.1.3 Funkce call_get_status_req

Tato funkce stejně jako předchozí (kapitola 9.4.1.2) zapouzdřuje funkci *profi_snd_req_res*. V tomto případě je ovšem potřeba vyvolat požadavek na zjištění stavu PAPI rozhraní, což opět provedeme nastavení vhodného servisního požadavku (kapitola 5.4.1). Servisní požadavek nakonfigurujeme přímo v zapouzdřovací funkci, je totiž pokaždé stejný a proto nemusíme předávat jeho parametry z aplikace Dataloggeru. Tentokrát ovšem není potřeba funkci předat žádnou službu, proto jí v zapouzdřovací funkci předáme pouze prázdnou strukturu. Návratová hodnota funkce *profi_snd_req_res* je opět předána jako návratová hodnota zapouzdřovací funkce.

9.4.1.4 Funkce call_profi_rcv_con_ind

Tato funkce zapouzdřuje funkci *profi_rcv_con_ind*, která slouží k příjmu potvrzení a výsledků požadavků vyvolaných funkcí *profi_snd_req_res*. Funkce *profi_rcv_con_ind* se volá s anulovanou datovou strukturou servisního požadavku a anulovanou strukturou místo struktury služby. Tyto struktury se ovšem předávají odkazem a nikoli hodnotou, takže funkce *profi_rcv_con_ind* může jejich obsah změnit. Zapouzdřovací funkce *call_rvc_con_ind* tady nejprve datové struktury anuluje, následně zavolá funkci *profi_rcv_con_ind*, čímž se naplní datová struktura servisního požadavku a datová struktura obsahující nastavení služby Status (kapitola 5.4.4). Protože funkce *profi_rcv_con_ind* pracuje se strukturou obsahující datové pole je potřeba data z této struktury překopírovat do modifikované struktury bez datového pole, kterou je možné bezproblémově předat aplikaci Dataloggeru. Návratová hodnota funkce *profi_rcv_con_ind* je opět předána jako návratová hodnota zapouzdřovací funkce.

9.4.1.5 Funkce call_get_dps_output_data

Tato funkce zapouzdřuje funkci *profi_get_dps_output_data*, která slouží k vyčítání dat z DP-RAM popsané v kapitole 4.3 a tedy dat z řídícího systému. Funkce *profi_get_dps_output_data* vrací vyčtená data přes byte-ové datové pole předávané odkazem. Ihned po zavolání této funkce pro získání dat z DP-RAM je volána funkce *GetSystemTime*, která vrací strukturu, v níž je uložen aktuální systémový čas včetně milisekund. Tím získáme časovou značku pro aktuálně vyčtená data v nejkratším možném čase od okamžiku jejich vyčtení. Protože jsou vyčtená data uložena v bytovém poli, je potřeba je přesunout do datové struktury, kterou je možné předat aplikaci Dataloggeru. Návratová hodnota funkce *profi_get_dps_output_data* je opět předána jako návratová hodnota zapouzdřovací funkce.

9.4.1.6 Funkce call_profi_end

Tato funkce zapouzdřuje funkci *profi_end*, která ukončuje PAPI rozhraní. Při zapouzdření této funkce nebylo potřeba, stejně jako u funkce *profi_init*, žádných

úprav, proto tato funkce obsahuje pouze volání funkce *profi_end*. Návratová hodnota funkce *profi_end* je opět předána jako návratová hodnota zapouzdřovací funkce.

9.4.2 Hlavní program

Hlavní program se skládá z necelých čtyř desítek sub-procedur, což jsou metody, které nemají návratovou hodnotu. Většina metod v programu slouží jako handle (handle – mít na starosti) pro události prováděné s ovládacími prvky v uživatelském rozhraní programu. Tedy například stisknutí tlačítka pro inicializaci PAPI rozhraní se díky handle vyvolá příslušná metoda, která vykoná příslušný kód pro inicializaci PAPI rozhraní. Některé metody jsem vytvořil k zapouzdření často vykonávaného kódu, který by se opakoval ve více metodách, případně k rozdělení příliš dlouhého a nepřehledného kód k jeho zpřehlednění.

9.4.2.1 Funkce inicializace PAPI

Tato metoda *Button_init* se volá po stisknutí tlačítka „Init Slave“. Volá knihovní funkci *call_profi_init* (kapitola 9.4.1.1). Výsledek vykonání funkce *call_profi_init* se vypíše do „API Trace“ (kapitola 9.3.1), zároveň pokud se tato vykoná úspěšně aktivují se některá tlačítka v interface a tlačítko „Init Slave“ se deaktivuje.

9.4.2.2 Funkce ukončení PAPI

Tato metoda *Button_end* se volá po stisknutí tlačítka „End Slave“. Volá knihovní funkci *call_profi_end* (kapitola 9.4.1.6). Výsledek vykonání funkce *call_profi_end* se vypíše do „API Trace“ (kapitola 9.3.1), zároveň pokud se tato vykoná úspěšně deaktivují se některá tlačítka v interface, naopak se aktivuje tlačítko „Init Slave“ a „Add“ pro nastavení datových rámců.

9.4.2.3 Funkce nastavení parametrů Slave

Tato metoda *Button_set_slave* se volá po stisknutí tlačítka „Set Slave“. Podle nastavení naplní datovou strukturu pro službu inicializace Slave, přičemž jsou nastavené hodnoty nejprve zkontrolovány a ověřeny na správnost. Poté je volána

funkce *call_init_slave_req* (kapitola 9.4.1.2), výsledek jejího vykonání se vypíše do „API Trace“ (kapitola 9.3.1). Při úspěšném vykonání funkce *call_init_slave_req* se aktivuje časovač.

9.4.2.4 Funkce časovač

Tato metoda *Timer1_Tick* je volána automaticky s nastavenou periodou. Perioda se nastavuje buď při nastavení parametrů Slave (kapitola 9.4.2.3) nebo při spuštění vyčítání dat (kapitola 9.4.2.5). Metoda volá metodu *handle_con_ind* (kapitola 9.4.2.14) pokud není aktivní vyčítání dat, pokud je vyčítání dat aktivní volá se metoda *get_output_data* (kapitola 9.4.2.7). Pokud je aktivní „Turbo postmort“ volá se místo metody *get_output_data* (kapitola 9.4.2.7) metoda *turbo_postmort* (kapitola 9.4.2.8).

9.4.2.5 Funkce datalogging start

Tato metoda *Button_datalog_start* se volá po stisknutí tlačítka „Start Getting Data“. Nejprve se provede ověření správnosti všech nastavených parametrů. Následně se ověří, zda neexistuje soubor se zadaným názvem a cestou, pokud existuje, inkrementuje se číslo, které se automaticky vkládá do názvu souboru a vytvoří se nový soubor se zadaným názvem. Nastaví se perioda pro časovač, deaktivuje se většina ovládacích prvků, programové proměnné se nastaví na příslušné hodnoty a přenastaví se velikost datových polí pro data.

9.4.2.6 Funkce datalogging stop

Tato metoda *Button_datalog_stop* se volá po stisknutí tlačítka „Stop Getting Data“. Nastaví příslušné programové proměnné a aktivuje ovládací prvky deaktivované funkcí *Button_datalog_start* (kapitola 9.4.2.5).

9.4.2.7 Funkce Get data from card

Tato metoda *get_output_data* je volána časovačem. Nejprve volá knihovní funkci *call_get_dps_output_data* (kapitola 9.4.1.5), které jako parametr předá datovou strukturu pro data vyčtená z karty. Po vykonání funkce se zavolá metoda *CopyDataToArray* (kapitola 9.4.2.22) která data vyčtená z karty přesune z datové

struktury do datového pole se kterým se lépe pracuje. Pokud se funkce *call_get_dps_output_data* (kapitola 9.4.1.5) vykonala úspěšně, metoda pokračuje zpracováním vyčtených dat.

Pokud je výstupní soubor nastaven na souborový typ xls, provede se ověření na množství dat v souboru a v případě potřeby se vytvoří nový soubor.

Do „Time stamp“ v uživatelském rozhraní se zapíše časová značka pro data, kterou vytvoří funkce *call_get_dps_output_data* (kapitola 9.4.1.5) a zároveň se časová značka zapíše do souboru, pokaždé na nový řádek.

Data jsou z DP RAM vyčtena v bytovém datovém poli kde jsou jednotlivé proměnné/signály rozděleny na byty a je potřeba je složit zpět. Složení probíhá tak, že se v cyklu postupně projdou jednotlivé řádky v tabulce v záložce „Input data (sent by master)“ (kapitola 9.3.3), jeden řádek tabulky odpovídá jedné proměnné/signálu, a zjišťuje se hodnota ve sloupci „Type“, který určuje datový typ proměnné/signálu. Z datového typu se pak identifikuje, kolik prvků z datového pole náleží pro danou proměnnou. Jednotlivé byty se buď matematicky, nebo pomocí speciální datové struktury složí dohromady a uloží se do datové proměnné odpovídajícího datového typu, která se následně uloží do příslušné buňky na příslušný řádek v tabulce v záložce „Input data (sent by master)“ (kapitola 9.3.3). Index datového pole se pak navýší o počet zpracovaných bytů, takže následující indexovaný byte odpovídá další proměnné. Hodnota zapsaná do tabulky je zároveň zapsána do externího souboru. V jedné iteraci cyklu se takto zpracuje vždy jeden řádek tabulky. Pokud je aktivní funkce Postmort, volá se na konci iterace metoda *postmort_marks_run* (kapitola 9.4.2.9).

Poté co jsou do tabulky zapsány hodnoty všech proměnných/signálů a funkce Postmort je aktivní zapíší se na konec posledního řádku externího souboru čísla proměnných/signálu, tedy v podstatě řádků tabulky, které vyhodnotila funkce Postmort.

Pokud je aktivní „Postmort online“ (kapitola 9.2.2.1) a je splněna příslušná podmínka, zavolá se metoda *postmort_full_function* (kapitola 9.4.2.10).

Pokud bylo u některých proměnných/signálů nastaveno vykreslení do grafu v záložce „Graph“ (kapitola 9.3.5) zavolá se metoda *draw_graph_function*.

9.4.2.8 Funkce turbo postmort

Tato metoda *turbo_postmort* je modifikovaná metoda *get_output_data* (kapitola 9.4.2.7). Rozdíl mezi těmito metodami spočívá v tom, že tato metoda neukládá hodnoty proměnných do externího souboru, ale pouze na vnitřní zásobník, který má velikost danou vzorcem

(2).

$$x = \frac{T_{be} * 1000}{T_{per}} + \frac{T_{ae} * 1000}{T_{per}} \qquad \textbf{(2)}$$

kde:

x – počet záznamů

T_{be} – čas před událostí [s]

T_{ae} – čas po události [s]

T_{per} – perioda vyčítání [ms]

Přičemž maximální velikost zásobníku je je $x_{max} = (2^{64} - 1)$. Data jsou do tabulky ukládána stejně jako u metody *get_output_data*, pouze na konci každé iterace cyklu je volána metoda *turbo_postmort_evaluate* (kapitola 9.4.2.11).

Následuje podmínka, jejíž splnění umožní uložení postmort dat. Při jejím splnění se deaktivuje časovač (kapitola 9.4.2.4), aby nedošlo ke změně dat v zásobníku během ukládání. Poté se začnou data ze zásobníku ukládat do externího souboru, přičemž záleží na nastaveném typu externího souboru, kdy se používá jiný oddělovač a u souboru s příponou xls je omezení množství dat na jeden soubor.

Ukládání dat do externího souboru je provedeno tak, že se v cyklu prochází řádky tabulky v záložce „Input data (sent by master)“ stejně jako u metody *get_output_data* (kapitola 9.4.2.7), ovšem jednotlivé byty nejsou vyčítány z datového pole, ale ze zásobníku a složené proměnné/signály nejsou ukládány do tabulky, ale přímo do externího souboru pro postmort data.

Poté co jsou data z celého zásobníku uložena do externího souboru je opět aktivován časovač a je zvýšeno číslo automaticky vkládané do názvu externího souboru pro postmort data, aby při dalším ukládání nedošlo k jeho přepsání.

Pokud bylo u některých proměnných/signálů nastaveno vykreslení do grafu v záložce „Graph“ (kapitola 9.3.5) zavolá se metoda *draw_graph_function*.

9.4.2.9 Funkce postmort marks

Tato metoda *postmort_marks_run* vyhodnocuje aktuálně vyčtená data z DP RAM (4.3) vůči nastaveným Postmort funkcím. Metoda v cyklu prochází řádky tabulky v záložce „Postmort“ (kapitola 9.3.4) a na základě postmort funkce ve sloupci „Function“ porovnává příslušnou hodnotu proměnné/signálu z tabulky „Input data (send by master)“ s hodnotou ve sloupci „Limit a“, případně „Limit b“.

Pokud je postmort funkce vyhodnocena kladně, projeví se to ve sloupci „State“ v tabulce v záložce „Postmort“ (kapitola 9.3.4) a do pomocné datové proměnné se uloží číslo řádku, tedy proměnné/signálu, která splnila podmínky postmort funkce.

9.4.2.10 Funkce postmort full

Tato metoda *postmort_full_run* ukládá postmort data do externího souboru pro postmort data. Metoda si ukládá řadu časových značek do rotujícího zásobníku. Ve chvíli kdy je vyvoláno uložení postmort dat, otevře se externí soubor s již uloženými vyčtenými daty a vyhledá se časová značka odpovídající časové značce na první pozici v rotujícím zásobníku. Od této časové značky se pak uložená zkopírují do externího souboru pro postmort data.

9.4.2.11 Funkce turbo postmort evaluate

Tato metoda *turbo_postmort_evaluate* je v podstatě shodná s funkcí *postmort_marks_run* (kapitola 9.4.2.9), ovšem tato funkce neukládá číslo řádku, tedy proměnné/signálu, která splnila postmort funkci.

9.4.2.12 Funkce graph data run

Tato metoda *graph_data_function* převádí hodnoty sledovaných proměnných/signálů, které se mají zobrazovat v grafu v záložce „Graph“ (kapitola 9.3.5) na y-ové souřadnice, které přepočítává v poměru k dosud největší zobrazené hodnotě a ukládá je do speciálního zásobníku. Metoda rozlišuje, zda již byl graf

vykreslen v celém rozsahu x-ové osy a pokud ano, zavolá metodu *rotate_graph_data* (kapitola 9.4.2.35), která data v zásobníku posune tak, že první uložená hodnota je ztracena. Na poslední, uvolněné místo se pak uloží hodnota nová. Nakonec metoda zavolá metodu vnitřní *refresh* nad objektem Panel1, která graf vykreslí.

9.4.2.13 Funkce get status

Tato metoda *Button_status* se volá po stisknutí tlačítka „Get status“ v záložce „Slave status“ (kapitola 9.3.2). Metoda volá knihovní funkci *call_get_status_req* (kapitola 9.4.1.3). Výsledek vykonání funkce *call_get_status_req* se vypíše do „API Trace“ (kapitola 9.3.1).

9.4.2.14 Funkce handle con_ind

Tato metoda *handle_con_ind* je volána časovačem. Metoda volá knihovní funkci *call_profi_rcv_con_ind* (kapitola 9.4.1.4), které předá dvě prázdné struktury. Po vykonání funkce *call_profi_rcv_con_ind* je v těchto dvou strukturách uloženo potvrzení od PAPI rozhraní, případně indikace stavu PAPI rozhraní. Metoda jednu ze struktur dekóduje sama a informace vypíše do „API Trace“ (kapitola 9.3.1). Pro dekódování druhé struktury zavolá metodu *decode_con_ind_status* (kapitola 9.4.2.15), případně metodu *decode_dps_prm_data* (kapitola 9.4.2.16).

9.4.2.15 Funkce decode con_ind

Tato metoda *decode_con_ind_status* je volána metodou *handle_con_ind* (kapitola 9.4.2.14) a slouží k dekódovaní druhé struktury s uloženou informací o potvrzení od PAPI rozhraní, případně indikací stavu PAPI rozhraní. Podle dekódované informace metoda nastaví jednotlivé prvky v záložce „Slave status“ (kapitola 9.3.2).

9.4.2.16 Funkce decode prm data

Tato metoda *decode_dps_prm_data* je volána metodou *handle_con_ind* (kapitola 9.4.2.14) a také slouží k dekódovaní druhé struktury s uloženou informací o potvrzení od PAPI rozhraní, případně indikací stavu PAPI rozhraní. Podle

dekódované informace metoda nastaví jednotlivé prvky v záložce „Slave status“ (kapitola 9.3.2).

9.4.2.17 Funkce add cfg data

Tato metoda *Button_add_cfg* je volána po stisku tlačítka „Add“ (kapitola 9.3.1) a slouží k nastavení datových rámců. Metoda nejprve ověří správnost nastavení všech hodnot pro vytvoření datového rámce a pak podle nich vypočte hodnotu ve které je zakódován formát datového rámce, jeho délka a informace, že se jedná o rámec vstupních dat. Zároveň jsou při výpočtu hodnoty do tabulky v záložce „Input data (sent by master)“ (kapitola 9.3.3) přidávány řádky pro jednotlivé proměnné/signály, jejichž počet je dán velikostí vytvářeného datového rámce a velikostí datového typu proměnných/signálů náležících do tohoto datového rámce. Při přidání každého jednotlivého řádku je v závislosti na formátu vytvářeného datového rámce volána metoda *sel_data_type_byte* (kapitola 9.4.2.18) nebo metoda *sel_data_type_word* (kapitola 9.4.2.19), která do nově vytvořeného řádku přidá do sloupce „Row“ číslo řádku a do sloupce „Type“ typ dané proměnné/signálu.

9.4.2.18 Funkce selecting data type – byte

Tato metoda *sel_data_type_byte* je volána metodou *Button_add_cfg* (kapitola 9.4.2.17). Do sloupce „Row“ v tabulce v záložce „Input data (sent by master)“ (kapitola 9.3.3) nastavuje podle aktuálního indexu číslo příslušného řádku a pro datový typ byte nastavuje do sloupce „Type“ datový typ odpovídající proměnné/signálu.

9.4.2.19 Funkce selecting data type - word

Tato metoda *sel_data_type_word* je volána metodou *Button_add_cfg* (kapitola 9.4.2.17). Do sloupce „Row“ v tabulce v záložce „Input data (sent by master)“ (kapitola 9.3.3) nastavuje podle aktuálního indexu číslo příslušného řádku a pro datový typ word nastavuje do sloupce „Type“ datový typ odpovídající proměnné/signálu.

9.4.2.20 Funkce remove cfg data

Tato metoda *Button_rem_cfg* je volána po stisknutí tlačítka „Remove last cfg data“ (kapitola 9.3.1). Metoda slouží k odebrání a smazání posledního nakonfigurovaného rámce, přičemž zároveň z tabulky „Input data (sent by master)“ (kapitola 9.3.3) odstraní řádky s proměnnými, které do odstraněného datového rámce náležely.

9.4.2.21 Funkce selecting data format

Tato metoda *Combobox_select_format* je volána automaticky při vybrání položky z rozbalovací lišty „Format“ (kapitola 9.3.1) pro výběr datového formátu datového rámce. Metoda zviditelní rozbalovací lišty „Length of data“ a „Standard conversion“ (kapitola 9.3.1) s vybraným formátem a zneviditelní rozbalovací lišty s druhým datovým formátem.

9.4.2.22 Funkce update data

Tato metoda *CopyDataToArray* je volána metodou *get_output_data* (kapitola 9.4.2.7) a slouží k překopírování dat vyčtených z DP RAM (kapitola 4.3) z datové struktury do datového pole se kterým se lépe pracuje.

9.4.2.23Funkce update postmort data

Tato metoda *CopyDataToPostmortArray* je volána metodou *turbo_postmort* (kapitola 9.4.2.8) a slouží k překopírování dat vyčtených z DP RAM (kapitola 4.3) z datové struktury do speciálního datového pole, které je v podstatě zásobník v paměti počítače a které udržuje všechny vyčtené hodnoty spadající do časového intervalu daného Postmort funkcí.

9.4.2.24 Funkce set name and path to data file

Tato metoda *Button_data_file* je volána po stisknutí tlačítka „Set data filename“ (kapitola 9.3.3). Metoda vyvolá a zobrazí *SaveFileDialog1*, která umožní nastavit název externího souboru a cestu k němu. Název a cesta jsou pak upraveny a uloženy.

9.4.2.25 Funkce reload items on tabpage show

Tato metoda *TabControl_reload* je automaticky volána pokud je zobrazena některá ze záložek (kapitola 9.3.1) a jejím úkolem je aktualizovat položky v rozbalovacích lištách v záložce „Postmort“ (kapitola 9.3.4).

9.4.2.26 Funkce constant or measured variable

Metody *Checkbox_measured1* a *Checkbox_measured2* jsou volány při zatrhnutí prvku „Measured“ v záložce „Postmort“ (kapitola 9.3.4). Při nastavovaní Postmort funkce se při zatrhnutí prvků se zneviditelní textová pole pro zadání konstantních hodnot a místo nich se zviditelní rozbalovací lišty pro výběr sledované proměnné/signálu.

9.4.2.27 Funkce add postmort item

Tato metoda *Button_add_pmi* je volána po stisknutí tlačítka „Add“ v záložce „Postmort“ (kapitola 9.3.4). Metoda přidá nový řádek do tabulky v záložce „Postmort“ (kapitola 9.3.4) a vloží do něj nastavení postmort funkce, tedy název sledované proměnné/signálu, datový typ sledované proměnné/signálu, typ postmort funkce a limity postmort funkce.

9.4.2.28 Funkceremove postmort item

Tato metoda *Button_rem_pmi* je volána po stisknutí tlačítka „Remove“ v záložce „Postmort“ (kapitola 9.3.4). Metoda z tabulky postmort funkcí v záložce „Postmort“ (kapitola 9.3.4) odstraní řádek se zvolenou sledovanou proměnnou/signálem. Sledovaná proměnná/signál se zvolí pomocí rozbalovací lišty „Remove“ nad tlačítkem „Remove“ v záložce „Postmort“ (kapitola 9.3.4).

9.4.2.29 Funkce save settings

Tato metoda *Button_save_set* je volána po stisknutí tlačítka „Save config.“ v hlavním okně aplikace (kapitola 9.3.1). Metoda vyvolá a zobrazí *SaveFileDialog1*, který umožňuje nastavit název externího souboru pro uložení nastavení a cestu k němu. Pak do tohoto souboru uloží nastavení většiny prvků aplikace datalogger a obsah některých programových proměnných.

9.4.2.30 Funkce load settings

Tato metoda *Button_load_set* je volána po stisknutí tlačítka „Load config.“ v hlavním okně aplikace(kapitola 9.3.1). Metoda vyvolá a zobrazí *OpenFileDialog1*, který umožňuje zvolit cestu a název externího souboru s uloženým nastavením. Poté metoda ze souboru načte všechna uložená nastavení a obsah programových proměnných.

9.4.2.31 Funkce postmort type choice

Tato metoda *RadioButton_pm_type* je volána pokaždé, když je změněn typ Postmort funkce „Postmort on/off“ v záložce „Postmort“ (kapitola 9.3.4). Metoda nastaví, která z postmort funkcí bude aktivní.

9.4.2.32 Funkce enable postmort item remove button

Tato metoda *ComboBox_enable_rem_pmi* je volána při výběru položky z rozbalovací lišty „Remove“ v záložce „Postrmort“ (kapitola 9.3.4). Metoda povolí tlačítko „Remove“ v záložce „Postmort“ (kapitola 9.3.4) pokud je vybrána některá z položek rozbalovací lišty „Remove“.

9.4.2.33 Funkce set name and path to postmort file

Tato metoda *Button_pm_file* je volána při stisknutí tlačítka „Set postmort filename“ v záložce „Postmort“ (kapitola 9.3.4). Metoda vyvolá a zobrazí *SaveFileDialog1*, která umožní nastavit název externího souboru pro postmort data a cestu k němu. Název a cesta jsou pak upraveny a uloženy.

9.4.2.34 Funkce graph drawing

Tato metoda *Panel1_paint* je automaticky volána pokaždé, když má být vykreslen nebo překreslen graf v záložce „Graph“ (kapitola 9.3.5). Metoda nejprve vykreslí osy grafu a legendu a následně vykreslí vybrané průběhy sledovaných proměnných/signálů, přičemž automaticky mění barvu vykreslované čáry.

9.4.2.35 Funkce rotate graph buffer data

Tato metoda *rotate_graph_data* je volána metodou *graph_data_function* (kapitola 9.4.2.12) a slouží k posunutí hodnot v zásobníku y-ových souřadnic pro vykreslení grafu v záložce „Graph“ (kapitola 9.3.5) tak, že každá hodnota je přepsána hodnotou s indexem o jedna vyšším, takže první hodnota, nejstarší, je ztracena a poslední místo v zásobníku je uvolněno pro novou hodnotu. To stejné se provede se zásobníkem pro časové značky, které slouží jako hodnoty pro x-ovou osu.

9.4.2.36 Funkce graph item adding and removing

Tato metoda *DataGridView_graph_items* je volána pokaždé, kdy je uživatelsky editována jakákoliv buňka v tabulce v záložce „Input data (sent by master)“ (kapitola 9.3.3). Metoda prochází sloupec „Graph“ v tabulce v záložce „Input data (sent by master)“ (kapitola 9.3.3) a zjišťuje, zda není zatržen prvek označující, že má být daná proměnná/signál zobrazena v grafu v záložce „Graph“ (kapitola 9.3.5). Pokud zjistí, že nějaký prvek zatržen je, ověří zda, má uložen aktuální řádek. V případě že ho již uložen má neděje se nic, v případě že ho uložen nemá, řádek uloží. Zároveň metoda prochází uložené řádky a zjišťuje, zda jsou opravdu všechny zatrženy, když zjistí, že již nějaký zatržen není, odstraní ho z uložených. Pro přehlednost metoda zatržené řádky označuje změnou barvy písma v buňce „Variable name“ na barvu jakou je daná proměnná/signál vykreslen v grafu v záložce „Graph“ (kapitola 9.3.5). Metoda také hlídá počet vykreslovaných proměnných/signálů v grafu a periodu vyčítání dat.

9.4.2.37 Funkce start button color a stop button color

Tyto dvě metody *Button_start_color* a *Button_stop_color* se volají automaticky, pokud dojde ke změně stavu tlačítek „Start Getting Data“ a Stop Getting Data“ v záložce „Input data (sent by master) (kapitola 9.3.3) a mění jejich barvu v závislost na jejich stavu aktivní/neaktivní.

10. TESTOVÁNÍ APLIKACE DATALOGGER

10.1 PRINCIP TESTU

S aplikací dataloggeru byla provedena sada testů, kdy bylo při různých nastaveních typu postmortu, periody vyčítání dat a vykreslování grafu sledováno zatížení CPU testovacího PC a dodržování nastavené periody vyčítání dat. Délka jednotlivých testů byla přibližně 10 a 5 minut. Testy byly provedeny jeden po druhém bez restartování PC a aplikace dataloggeru, čímž byl částečně simulován dlouhodobější běh aplikace.

Testovací PC mělo následující výkonové parametry:

- Procesor : Intel Pentium 4
- Frekvence CPU: 3,2 GHz
- Paměť RAM: 1,99 GB
- Otáčky HDD: 7200 ot/min
- OS: Windows XP Professional SP2 32bit

V průběhu celého testování byly na PC spuštěny tyto programy:

- Compact Control Builder
- Windows Commander
- systémový Správce úloh
- Datum a čas – vlastnosti
- složka – windows explorer
- Mezi testy byl spouštěn Poznámkový blok

Aplikace dataloggeru byla přes Profibus PCI kartu a Profibus sběrnici spojena s kontrolerem AC 800M společnosti ABB, ve kterém byla nahrána testovací aplikace. Testovací aplikace obsahovala 23 signálů typu real a 16 signálů typu bool napojených na různé generátory hodnot.

Přehled testovaných nastavení aplikace dataloggeru je v tabulce Tab. 4.

nastavení	perioda	délka testu
Postmort off	10 ms	10 min
Postmort off	50 ms	10 min
Postmort off	100 ms	10 min
Postmort off	125 ms	10 min
Postmort off	250 ms	10 min
Postmort off	1000 ms	10 min
Online postmort marks	50 ms	10 min
Online postmort marks	100 ms	10 min
Online postmort marks	125 ms	10 min
Online postmort	1000 ms	10 min
Turbo postmort	10 ms	10 min
Turbo postmort	50 ms	10 min
Turbo postmort	100 ms	10 min
Turbo postmort	125 ms	10 min
Turbo postmort	250 ms	10 min
Turbo postmort	1000 ms	10 min
Postmort off + graf	100 ms	5 min
Postmort off + graf	125 ms	5 min
Turbo postmort + graf	100 ms	5 min
Turbo postmort + graf	125 ms	5 min
Postmort off + zatížení	100 ms	5 min
Postmort off + zatížení	125 ms	5 min
Turbo postmort + zatížení	100 ms	5 min
Turbo postmort + zatížení	125 ms	5 min

Tab. 4 Přehled testovaných nastavení

Postmort měl dvě vyhodnocovací funkce. Obě vyhodnocovaly stejnou proměnnou/signál, která nabývala hodnot 0 až 59 a její hodnota byla každou sekundu inkrementována o 1. První postmort funkce porovnávala proměnnou/signál na hodnotu rovno (=) 0, druhá postmort funkce porovnával proměnnou/signál na hodnotu rovno (=) 30. Ukládaný časový úsek před výskytem události a ukládaný časový úsek po výskytu události byli shodně nastaveny na 10 sekund.

Graf byl nastaven na zobrazování průběhů prvních deseti realových signálů.

Zatížení bylo simulováno tak, že bylo přes Windows Commander kopírováno, z pevného disku PC na USB flash disk a zpět, větší množství souborů o celkové velikosti kolem 2 GB.

Doba mezi jednotlivými záznamy byla zjištěna tak, že časová značka záznamu ve formátu „hh:mm:ss:sss“ byla převedena na čas dne v milisekundách

a tato hodnota byla odečtena od stejně převedené časové značky záznamu po ní následujícího.

Při vyhodnocování byly u některých postmort souborů odstraněny nulové řádky, které vznikly, takže došlo k výskytu události dříve, než od spuštění vyčítání uběhl nastavený časový úsek „Time before event [ms]“ (kapitola 9.3.4).

10.2 VYTÍŽENÍ CPU

V průběhu všech testů bylo pomocí systémového správce úloh sledováno vytížení CPU testovaného PC. Sledováno bylo přibližné průměrné vytížení CPU, minimální a maximální hodnota vytížení CPU na začátku (v první minutě) testu a minimální a maximální hodnota vytížení CPU na konci (v poslední minutě) testu. Sledované hodnoty CPU testovaného PC jsou uvedeny v tabulce Tab. 5.

TEST nastavení	perioda	délka testu	CPU průměr	CPU min - zač.	CPU max - zač	CPU min - kon.	CPU max - kon
Postmort off	10 ms	10 min	5	3	9	4	12
Postmort off	50 ms	10 min	4	2	8	3	8
Postmort off	100 ms	10 min	4	1	6	1	9
Postmort off	125 ms	10 min	4	1	7	3	7
Postmort off	250 ms	10 min	4	2	8	3	7
Postmort off	1000 ms	10 min	4	1	8	2	6
Online postmort marks	50 ms	10 min	5	1	7	3	8
Online postmort marks	100 ms	10 min	4	1	8	1	6
Online postmort marks	125 ms	10 min	4	2	7	2	7
Online postmort	1000 ms	10 min	4	2	10	2	8
Turbo postmort	10 ms	10 min	5	4	54	4	51
Turbo postmort	50 ms	10 min	4	2	18	1	19
Turbo postmort	100 ms	10 min	4	1	12	2	11
Turbo postmort	125 ms	10 min	4	1	11	2	10
Turbo postmort	250 ms	10 min	4	2	11	3	8
Turbo postmort	1000 ms	10 min	4	1	8	4	8
Postmort off + graf	100 ms	5 min	5	3	11	3	10
Postmort off + graf	125 ms	5 min	4	3	6	3	8
Turbo postmort + graf	100 ms	5 min	7	2	10	3	11
Turbo postmort + graf	125 ms	5 min	8	4	16	6	13
Postmort off + zatížení	100 ms	5 min	7	4	26	5	8
Postmort off + zatížení	125 ms	5 min	20	15	53	10	52
Turbo postmort + zatížení	100 ms	5 min	50	6	73	8	50
Turbo postmort + zatížení	125 ms	5 min	11	9	23	47	68
Zatížení			15	12	18		

Tab. 5 Hodnoty sledovaného zatížení CPU

Z tabulky je patrné, že běh aplikace datalogger se spuštěným vyčítáním dat nemělo na průměrné vytížení CPU výraznější vliv, hodnoty odpovídají běžnému vytížení CPU v klidovém stavu PC a operačního systému. Zvýšení průměrného vytížení CPU je patrné spuštěného grafu, protože se graf v každém cyklu vyčítání dat celý překresluje. Významné zvýšení průměrného vytížení CPU je pak patrné při simulovaném zatížení kopírováním dat, to je ovšem způsobené zejména samotným kopírováním dat jak je vidět z posledního řádku tabulky, kde jsou hodnoty zjištěné při samotném kopírování dat.

Vysoké maximální hodnoty vytížení CPU u „Turbo postmort“ jsou způsobeny dávkovým ukládáním většího množství vyčtených dat na pevný disk. U ostatních typů postmortu jsou data ukládány průběžně, tudíž se vytížení při ukládání na pevný disk rozloží v čase. Samozřejmě ovšem závisí na množství ukládaných dat.

10.3 TEST 1.: POSTMORT OFF

Popisované grafy jsou v příloze Příloha 1: Grafy Testů.

Z grafů je patrné, že pro některé periody vyčítání dat má aplikace problém s dodržením požadované periody. U periody 10 ms se perioda zvýšila na 15 - 16 ms, místy až na hodnotu 31 ms. U periody 50 ms došlo k prodloužení periody na 62 - 63 ms, avšak na rozdíl od periody 10 ms je prodloužený interval konstantní po celou dobu vyčítání. U periody 100 ms je situace stejná jako u periody 50 ms, avšak prodloužení periody je 109 - 110 ms. U periody 125 ms k žádnému prodloužení periody nedošlo a perioda je konstantní po celou dobu vyčítání dat, stejně tak u periody 250 ms a periody 1000 ms nedošlo k žádnému prodloužení periody. Je tedy patrné, že aplikaci datalogger nejvíce vyhovují periody vyčítání dat rovnající se k-násobku 125 ms. Extrémní jednorázové zvýšení periody projevující se v grafech 10 ms, 50 ms a 250 ms je způsobeno krátkodobým vytížením CPU.

10.4 TEST 2.: ONLINE POSTMORT MARKS

Popisované grafy jsou v příloze Příloha 1: Grafy Testů.

Stejně jako u předchozího testu je z grafů patrné, že pro některé periody vyčítání dat má aplikace problém s dodržením požadované periody. Prodloužení period je stejné jako u shodných požadovaných period z předešlého testu, z čehož můžeme usoudit, že vyhodnocení postmortu a ukládání značky identifikující výskyt události do souboru nezpůsobuje prodloužení periody vyčítání dat. Extrémní jednorázové zvýšení periody projevující se v grafu 50 ms je způsobeno krátkodobým vytížením CPU.

10.5 TEST 3.: ONLINE POSTMORT

Popisovaný graf je v příloze Příloha 1: Grafy Testů.

Z grafu je patrné, že aplikace dodržuje zvolenou periodu vyčítání dat. V dané periodě tedy zvládá průběžně ukládat data i dávkově ukládat postmort data. Kratší periodu pro tento typ postmortu není možné v aplikaci nastavit. Extrémní jednorázové zvýšení periody projevující se v grafu je způsobeno krátkodobým vytížením CPU.

10.6 TEST 4.: TURBO POSTMORT

Popisované grafy jsou v příloze Příloha 1: Grafy Testů.

Z grafů je patrné, že má aplikace problém s dodržením požadované periody pro některé periody vyčítání dat stejně jako u ostatních typů postmortu. Z grafů je patrné stejné prodloužení period jako u předchozích testů. Periodicky se opakující extrémní jednorázové zvýšení periody není způsobeno krátkodobým vytížením CPU, ale tím, že se ukládají pouze požadované časové úseky před a po výskytu události a také tím, že je vyčítání dat v průběhu ukládání dat do souboru pozastaveno. Krátkodobým vytížením CPU je způsobeno jednorázové neperiodické extrémní zvýšení periody.

10.7 TEST 5.: POSTMORT OFF + GRAF

Popisované grafy jsou v příloze Příloha 1: Grafy Testů.

Z grafů je patrné opět stejné prodloužení periody pro shodné periody vyčítání jako u předchozích testů. Zároveň vidíme, že vykreslování grafu nezpůsobuje

prodloužení periody vyčítání dat. Extrémní jednorázové zvýšení periody projevující se v grafu je způsobeno krátkodobým vytížením CPU.

10.8 TEST 6.: TURBO POSTMORT + GRAF

Popisované grafy jsou v příloze Příloha 1: Grafy Testů.

Z grafů je patrná stejná situace jako u „Turbo postmort“ bez grafu, stejné prodloužení periody jako u předchozích testů a žádný vliv vykreslovaní grafu na prodloužení periody vyčítání dat jako u předchozího testu. Krátkodobým vytížením CPU je způsobeno jednorázové neperiodické extrémní zvýšení periody.

10.9 TEST 7.: POSTMORT OFF + ZATÍŽENÍ

Popisované grafy jsou v příloze Příloha 1: Grafy Testů.

Z grafu je patrné, že prodloužení periody je téměř stejné jako v předchozích testech, vyšší vytížení CPU a pevného disku zatíženého kopírováním dat způsobuje častější jednorázové extrémní zvýšení periody.

10.10 TEST 8.: TURBO POSTMORT + ZATÍŽENÍ

Popisované grafy jsou v příloze Příloha 1: Grafy Testů.

Z grafů je patrné, že zatížení CPU a pevného disku nemá na periodu vyčítání výraznější vliv, protože v průběhu vyčítání dat není pevný disk zatěžován, ten je zatěžován až v průběhu ukládání, kdy je ovšem vyčítání dat pozastaveno.

10.11 POROVNÁNÍ

Popisované grafy jsou v příloze Příloha 1: Grafy Testů.

Tyto grafy umožňují bližší porovnání změřených period vyčítání dat. Porovnávány jsou vždy stejné nastavené periody vyčítání. Pro přehlednost je zobrazen pouze malý výřez. Z grafů je patrné, že průběhy period jsou v podstatě stejné, posunutí v y-ové ose u nízkých period je pravděpodobně způsobeno jiným časem spuštění vyčítání nebo momentálním vytížením CPU.

10.12 SHRNUTÍ VÝSLEDKŮ TESTŮ

Jak bylo již uvedeno v kapitole 10.2 samotná aplikace Datalogger nezpůsobuje výrazné zatížení CPU počítače na kterém je aplikace spuštěna. Asi nejvýznamnějším faktem vyplývajícím z testování je, že aplikace nejlépe pracuje s periodou vyčítání dat, která je rovna k-násobku 125 milisekund. Pokud je zvolena jiná perioda dochází k prodloužení periody vyčítání dat o zhruba 3 -15 milisekund. Vzhledem k tomu, že prodloužená perioda je jinak víceméně konstantní a to i pro různé varianty nastavení, usuzuji, že prodloužení periody nezpůsobuje chyba v programu, ale časovač implementovaný z vývojového prostředí MS Visual Basic .NET. V průběhu testů se také občas objevilo náhodné jednorázové významnější prodloužení periody, které bylo pravděpodobně způsobeno dočasným vytížením CPU a implementaci na nerealtimovém operačním systému. Minimální dosažená perioda vyčítání dat je 15 milisekund.

Všechny testy proběhly úspěšně a s uspokojivými výsledky.

ZÁVĚR

Hlavním cílem této diplomové práce bylo vytvořit pro společnost ABB aplikaci Datalogger, která bude schopna přijímat data z PLC přes sběrnici PROFIBUS a přijmutá data zpracovávat, vyhodnocovat a ukládat.

V první kapitole (Datalogger) je stručně popsáno co je to datalogger, k čemu slouží a je zde naznačeno z jakého důvodu byl jako komunikační standard vybrán standard Profibus.

Druhá kapitola (Profibus) ve které je popsána sběrnice Profibus a komunikační standard. V kapitole je zmíněna historie tohoto standardu, základní vlastnosti standardu Profibus. Jedna z podkapitol je věnována referenčnímu modelu ISO/OSI, kde jsou společně s jednotlivými vrstvami modelu ISO/OSI využitými ve standardu Profibus podrobněji popsány vlastnosti Profibusu související s danými vrstvami. V poslední části této kapitoly je rozdělení komunikačního standardu Profibusu na jednotlivé typy podle jejich použití.

Třetí kapitola (Využití sběrnice Profibus při sběru a záznamu dat) zdůvodňuje požadavek na použití sběrnice Profibus pro komunikaci s PLC. Hlavními argumenty jsou rychlost a cyklická komunikace dat. Druhá podkapitola popisuje dva způsoby propojení PLC a PC pomocí této sběrnice.

Čtvrtá kapitola (PROFIBUS-PCI karta) je věnována Profibus-PCI kartě. Tato karta umožňuje připojení PC ke sběrnici Profibus a aplikace Datalogger využívá její Profibus API, které je k této kartě distribuováno jejím výrobcem. Kapitola popisuje vlastnosti samotné karty, částečně popisuje SDK sloužící k vývoji aplikací pro tuto kartu, způsob výměny dat mezi kartou a aplikací, a také popisuje základní konfiguraci karty a jejího ovladače.

Pátá kapitola (PAPI rozhraní) popisuje PAPI rozhraní, což je v podstatě komunikační rozhraní, které umožňuje aplikaci přistupovat k Profibus-PCI kartě a k datům touto kartou komunikovaným. Dále popisuje nízko-úrovňový přístup k funkcím „DP-Slave“ a způsob jak k těmto funkcím přistupovat přes PAPI funkce obsažené v PAPI dll knihovně. Kapitola také obsahuje výčet a stručný popis PAPI funkcí, které umožňují vyhnout se úkonům a problémům spojených s nízko-

úrovňovým přístupem. Poslední část této kapitoly obsahuje popis datových struktur, které slouží k nastavení služeb požadovaných na „DP-Slave“.

Šestá kapitola (Komunikace dat z kontroleru do aplikace) popisuje komunikaci dat z kontroleru do aplikace. Nejprve je uvedena konfigurace kontroleru, která je nutná pro komunikaci dat s Profibus-PCI kartou a PAPI rozhraním. Následuje konfigurace, kterou je nutné provést na straně Profibus-PCI karty a PAPI rozhraní. A na konec komunikace dat mezi Profibus-PCI kartou a PAPI rozhraním.

V sedmé kapitole (Vývojové prostředí a programovací jazyky) je popsáno zvolené vývojové prostředí a programovací jazyky s přihlédnutím na požadavky, které měla společnost ABB na tuto část projektu.

V osmé kapitole (Propojení Visual Basic .NET a PAPI dll knihovny) jsou popsány uvažované varianty propojení aplikace ve Visual Basic .Net a PAPI dll knihovny, přičemž 1. varianta byla nakonec zavržena kvůli náročnosti a chybám ke kterým při komunikaci docházelo. U 2. varianty sice k těmto chybám také docházelo, ale pomocí úprav je možné těmto chybám předejít.

Devátá kapitola (Aplikace Datalogger) se věnuje již samotné aplikaci Datalogger. Nejprve jsou uvedeny požadavky, které měla společnost ABB. Následně jsou popsány vlastnosti aplikace Datalogger a to z hlediska komunikovaných dat a sledovaných signálů, periody vyčítání dat, datových souborů a Postmortu. Dále je popsán grafický interface aplikace. Kapitola pokračuje popisem programové části aplikace, která se dělí na knihovnu, kterou bylo potřeba vytvořit aby aplikace mohla využívat funkce Profibus API a hlavní program aplikace Datalogger. V části věnované vytvořené knihovně jsou zároveň popsány i jednotlivé funkce knihovny. V části věnované hlavnímu programu jsou popsány jednotlivé části programu.

Poslední desátá kapitola (Testování aplikace Datalogger) popisuje testování aplikace Datalogger. Na začátku této kapitoly je uvedeno co a jak bylo testováno, zbytek kapitoly obsahuje vyhodnocení jednotlivých testů, pomocí grafů umístěných v příloze této diplomové práce a shrnutí výsledků testů. Testování ukázalo, že aplikace nejlépe pracuje s periodou vyčítání dat, která je rovna k-násobku 125 milisekund. Pokud je zvolena jiná perioda dochází k prodloužení periody vyčítání dat o zhruba 3 -15 milisekund. Toto prodloužení periody způsobuje časovač

implementovaný z vývojového prostředí MS Visual Basic, naneštěstí se mi nepodařilo získat žádné další informace, které by danou skutečnost vysvětlovaly.

Během testů aplikace pracovala správně, neobjevily se žádné programové chyby, aplikace významně nezatěžovala CPU testovacího počítače a výsledky testů byly celkově uspokojivé.

Aplikace Datalogger splňuje všechny požadavky společnosti ABB a v době odevzdání této diplomové práce je plánováno nasazení aplikace Datalogger v některých projektech společnosti ABB.

SEZNAM ZDROJŮ

LITERATURA

[1] Bc. Radim Bráblík: Datalogger pro sběrnici typu PROFIBUS, VUT Brno 2009, semestrální práce č.1, pdf

[2] Bc. Radim Bráblík: Datalogger pro sběrnici typu PROFIBUS, VUT Brno 2009, semestrální práce č.2, pdf

[3] Pavel Herout,: Učebnice jazyka C, KOPP, 2004, IV. Přepracované vydání, ISBN 80-7232-220-6

[4] Doc. Ing. František Zezulka - Csc.; Ing. Petr Fiedler - Ing. Petr Vaňous - Ing. Petr Cach: Průmyslové komunikační sítě, skriptum VUT [pdf], ÚAMT FEI VUT v Brně, 2000

[5] Dr. Zdeněk Hanzálek, FEL ČVUT Praha – Ing. Pavel Bronec, CSc., Fisher-Rosemount: Profibus a Foundation Fieldbus: konkurenti v oblasti průmyslových sběrnic, časopis AUTOMA, 2000, číslo 2

[6] Otakar Vlasák: Průmyslová sběrnice Profibus, semestrální práce, ČVUT v Praze, 2004/2005

[7] Prof. Ing. František Zezulka, CSc. – Ing. Ondřej Hynčica: Průmyslový Ethernet II: Referenční model ISO/OSI, časopis AUTOMA, 3/2007

[8] Ing. Pavel Kučera Ph.D.: Profibus, elektronické podklady k přednášce na VUT Brno

[9] PROFIBUS PCI karta PROFIboard Master/Slave (1 Kanál), http://www.foxon.cz/profibus-pci-karta-profiboard-masterslave-1-kanal-p-157.html?cPath=6_42

[10] Softing AG: PROFIBUS Application Program Interface User manual, verze 5.4, 30.10.2007, [pdf]

[11] PROFIBUS PCI karta PROFIboard Master/Slave (1 Kanál), http://www.foxon.cz/profibus-pci-karta-profiboard-masterslave-1-kanal-p-157.html?cPath=6_42

DOPLŇUJÍCÍ INFORMAČNÍ ZDROJE

[12] www.profibus.com Stránky společnosti Profibus Foundation věnované Profibusu

[13] www.profibus.cz Česká mutace stránek věnovaných Profibusu

[14] www.softing.com Stránky výrobce vybrané karty

[15] Redakce HW serveru: Průmyslová sběrnice Profibus, online článek, http://hw.cz/Rozhrani/ART1028-Prumyslova-sbernice-Profibus.html , 17. Leden 2004

[16] Wikipedia: Profibus, http://cs.wikipedia.org/wiki/Profibus

[17] www.profibus.com Stránky společnosti Profibus Foundation věnované Profibusu

[18] www.profibus.cz Česká mutace stránek věnovaných Profibusu

[19] www.softing.com Stránky výrobce vybrané karty

[20] Redakce HW serveru: Průmyslová sběrnice Profibus, online článek, http://hw.cz/Rozhrani/ART1028-Prumyslova-sbernice-Profibus.html , 17. Leden 2004

[21] Wikipedia: Profibus, http://cs.wikipedia.org/wiki/Profibus

[22] Wikipedia, http://cs.wikipedia.org

[23] Visual Basic .Net, http://cs.wikipedia.org/wiki/Visual_Basic_.NET

[24] Herceg Tomáš, Úvod do .Net frameworku, 3.4.2009, online článek, http://www.vbnet.cz/clanek--125-net_framework_od_zacatku_dil_1_uvod_do_net_frameworku.aspx

[25] Herceg Tomáš, Základní elementy VB.NET a C#, 18.4.2009, online článek, http://www.vbnet.cz/clanek--126-net_framework_od_zacatku_dil_2_zakladni_elementy_vb_net_a_c_.aspx

[26] Herceg Tomáš, Datové typy, 5.5.2009, online článek, http://www.vbnet.cz/clanek--127-net_framework_od_zacatku_dil_3_datove_typy.aspx

[27] MS Visual Basic .Net Studio Express Edition, http://www.microsoft.com/express/vb/default.aspx

[28] MS Visual C++ Studio Express Edition,
http://www.microsoft.com/express/vc/Default.aspx

SEZNAM ZKRATEK

PLC (Programmable Logic Controller – Programovatelný logický kontroler)

ISO/OSI (Open Systems Interconnection Reference model – Referenční model propojování otevřených systémů)

OSI - Open System Interconnection – propojování otevřených systémů

NRZ (Non Return to Zero – bez návratu k nule)

HCS (Hard Clad Silica – speciální vlákno na bázi Si)

SRD (Send and Request Data with acknowledge – Odeslání a požadavek na data s potvrzením)

SDA (Send Data with Acknowledge – odesílání dat s potvrzením)

SDN (Send Data with No acknowledge – odešli data bez potvrzení)

CSRD (Cyclic Send and Request Data – cyklické odesílání dat a požadavků)

LLI (Low Layer Interface – nízko úrovňové rozhraní)

PC (Personal Computer – osobní počítač)

FMS (Fieldbus Message Specification – vrstva specifikace zprávy)

API (Application Programming Interface - Aplikační programátorské rozhraní)

DP (Decentralized Periphery – decentralizované periférie)

PA (Process Automatization – automatizace procesů)

LAN (Local area network – lokální síť)

SDK (Software Development Kit – Nástroj pro vývoj softwaru)

RAM (Random Access Memory – Paměť s náhodným přístupem)

I/O (Input/Output – Vstupně/Výstupní)

PCI (Peripheral Component Interconnect – Připojení periferních komponent)

PAPI (PROFIBUS Application Program Interface – Rozhraní pro programování aplikací)

dll (dynamic link library – dynamicky připojená knihovna)

CIL (Common Intermediate Language – obecný přechodný jazyk)