České vysoké učení technické v Praze

Fakulta elektrotechnická

# Bakalářská práce

# Profibus FDL vrstva pro Linux

Vypracoval : Tran Duy Khanh
Vedoucí práce : Ing. Petr Smolík

Praha, 2006

Zadání

## Anotace

Tato práce se zabývá problémem připojení osobního počítače k populární průmyslové sběrnici Profibus. Komerční řešení pro připojení osobního počítače k Profibusu využívají speciální rozšiřující karty, ke kterým většinou technické dokumenty ani zdrojové kódy nejsou k dispozici běžným uživatelům. Navíc pořízení takových karet se softwarem je dosti nákladné, a proto není dostupné každému. Tato práce se nesnaží konkurovat těmto řešením, naopak se snaží jít svou cestou, nabídnout pokud možno nejjednodušší a nejekonomičtější řešení a přitom dosáhnout vysokých výkonů. Snahou bylo také položit základ, aby v budoucnu bylo možné použít toto řešení i na jiných operačních systémech ve světě volně šířitelných softwarů a stát se její součástí.

Hardwarové řešení spočívá ve využití osobního počítače s převodníkem z RS-232 na standard RS-485, užívaný sběrnicí Profibus, a to buď ve formě s jednoduchým převodníkem využívající standardní sériové rozhraní, nebo pomocí jednoduchých zásuvných karet s obvody UART.

Softwarově přestavuje ovladač do jádra implementující činnosti mastera, který je součástí řídicích jednotek v Profibusu. Prozatím je implementován na Linuxu. V budoucnu bych chtěl v tom pokračovat a rozšířit tuto práci i na jiné volně šířitelné operační systémy jako např. RTLinux a FreeBSD.

## Anotation

PBMaster is a solution to connect common personal computer to Profibus, the most popular fieldbus used in factory automation. There are many solutions with hardware support from commercial companies. However the implementations of their hardware and software are usually well protected and not readily available. PBMaster doesn`t want to compete with them, but aims to offer a cheap solution with open code.

Hardware is based on cooperation between personal computer and convertor of RS-232 to RS-485, the standard commonly used by Profibus. It is possible in two ways. Using expansion cards with integrated UART chips or a simple convertor of standard RS-232 used by personal computers. The first one is more expensive, but it is able to achieve higher speeds of communication. The latter is very simple and cheap, but its highest attainable speed is only 19 200 baud.

Software is based on device driver implementing functions of master station of Profibus. The driver has been developed for Linux, but in the future I would like to port this driver to other free operating systems such as RTLinux and FreeBSD.

## Prohlášení

*Prohlášuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.*

*Nemám závážný důvod proti užití tohoto školního díla ve smyslu zákona 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).*

*V Praze dne 25. srpna 2006* .....

*podpis*

# Poděkování

*Chtěl bych poděkovat svému vedoucímu bakalářké práce Ing. Petru Smolíkovi za jeho ochotu a vstřícnost při konzultacích a za veškerou pomoc a rad, které mi věnoval. Dále bych chtěl poděkovat Ing. Pavlovi Píšovi a Ing. Pavlovi Trnkovi za jejich cenné rady a teoretické řešení některých problémů, bez kterých by nebylo možné uskutečnit tuto práci.*

*Velké poděkování patří také mé rodině a mé přítelkyni Martině. Děkuji za podporu a energii, kterou jste mi neustále dodávali. Děkuji také svým kamarádům z Kostariky a Nicaragua, kteří mě srdečně přivítali a pohostili ve svých zemích během 5 měsíců psaní části této práce ve Střední Americe.*

# Profibus FDL
# vrstva pro Linux

*Tuto práci věnuji svým rodičům, své sestře a Martině.*

# Obsah

1 Úvod....10
1.1 Seznámení s Profibusem....10
1.2 Cílem práce....10
1.3 Struktura práce....11
2 Profibus DP....12
2.1 Vrstvená architektura Profibus DP....12
2.2 Fyzická vrstva....13
2.3 Linková vrstva....13
2.4 Aplikační vrstva....13
2.5 Popis rámců....14
Typy rámců....14
Adresové pole....16
Pole uživatelských dat....17
Znak FCS (Frame Control Sequence)....17
Znak FC (Frame Control)....17
2.6 Stavový automat stanice typu master....19
Offline....19
Listen_Token....20
Active_Idle....20
Claim_Token....21
Use_Token....21
Await_Data_Response....22
Check_Access_Time....22
Pass_Token....22
Check_Token_Pass....23
Await_Status_Response....23
2.7 Časovače a čítače FDL vrstvy....24
Synchronizační čas....24
Token Rotation Timer....24
Idle Timer....24
Slot Timer....24
Time-out Timer....25
Syn Interval Timer....25
GAP Update Timer....25
2.8 Řešení události na sběrnici....26
Připojení stanice typu master na sběrnici....26

Ztráta tokenu....26
Odpojení stanice typu master ze sběrnice....26
Dotazovaná stanice neodpovídá....26
Chyba při příjmu odpovědi na dotaz....27
Duplicita adres na sběrnici....27
3 Sériová komunikace....28
3.1 Sériový přenos dat....28
Synchronní přenos....28
Asynchronní přenos....28
3.2 Výhody a nevýhody synchronního a asynchronního přenosu dat....29
Synchronní přenos....29
Asynchronní přenos....29
3.3 Popis standardu RS-232....29
3.4 Standard RS-485....30
4 Realizace fyzické vrstvy....32
4.1 Použití standardního sériového portu....32
4.2 Použití rozšiřujících zásuvných karet....34
Obvod OX16PCI954....36
5 Tvorba ovladačů pro operační systém Linux....37
5.1 Kernel space a user space....37
Kernel space....38
User space....38
Ovladač zařízení....38
Moduly....39
make(1) a Makefile....39
První modul....40
Obsluha přerušení (Interrupt Service Routines)....41
Psaní PCI modulu....45
6 Struktura a provedení PBMaster....47
6.1 Struktura PBMaster....47
6.2 Pobrobnější popis jednotlivých částí....47
Modul „pbmcore“....47
Modul „pbm_950pci“ a „pbm_8250“....49
Modul „pbm_fma“....50
6.3 Nároky Profibusu....50
6.4 Ovládání převodníku a jeho využití k časování....50
Časování pomocí jednoduchého převodníku RS485/RS232....50
Vysílání znaků....51
Příjem znaků....51
Časování se současným příjmem znaků....51

Synchronizační mezera....52
Časování s obvodem 16C950....52
7 Použití PBMaster....54
7.1 Kompilace a instalace ovladače....54
Ukázkový příklad....55
8 Závěr....56
8.1 Další vývoj PBMaster....56
Seznam použité literatury....57
Zdroje z internetu....57
Seznam obrázků....58
Seznam tabulek....59
Seznam výpisů....60
Obsah přiloženého CD....61

# 1 Úvod

## 1.1 Seznámení s Profibusem

Profibus je nejznámější typ průmyslové sběrnice s více než 10-ti milióny uzlů (z roku 2004) používané po celém světě. Je to výsledkem prací německého výzkumného programu více jak 15-ti firem a institucí. V Evropě dominuje Profibus s více než 60-ti procent trhu v automatizačních továrnách.

Existuje 3 verze Profibusu:

- PROFIBUS-FMS (Fieldbus Message Specification), ve kterém je používán model Klient - Server pro komunikaci mezi automatizačními jednotkami,
- **PROFIBUS-DP** (Decentralized Peripherals), který je používán pro rychlé ovládání vstupu a výstupu, k připojení senzorů a akčních členů k řídicím jednotkám,
- PROFIBUS-PA (Process Automation), který je používán pro připojení průmyslových zařízení a čidel k řídicím jednotkám. Umožňuje bezpečný přenos a zapnutí linky.

Standard definuje navíc funkce pro bezpečné použití (PROFIsafe), pro řízení pohybu a další typy zařízení.

Profibus byl definován v roce 1991/1993 jako norma DIN 19245, následně v roce 1996 byl nově definován EN 50170 a od roku 1999 je zahrnut i pod IEC jako standard IEC 61158/IEC 61784.

Standard Profibus je spravován neziskovou organizací Profibus International, řízená z Karlsruhe, Německo. Ačkoliv je standard otevřený, není jednoduché získat jeho kopie. Je třeba být členem této organizace, pokud je mi známo.

V této práci se budu zabývat pouze verzí Profibus DP (Decentralized Peripherals). Je to nejpoužívanější varianta průmyslové sběrnice Profibus, která je určena zejména pro komunikaci mezi řídicími a decentralizovanými jednotkami. Obvykle jedním komunikačním kanálem nahrazuje mnohavodičové spojení řídicích jednotek se snímači a akčními členy, kde použitím Profibusu je současně zajištěna spolehlivá a rychlá výměna dat.

## 1.2 Cílem práce

Jak již bylo zmíněné v anotaci, cílem práce je vyřešit problém připojení osobího počítače k průmyslové sběrnici Profibus DP. Tato práce má navázat na práce Ing. Pavla Trnky „Profibus DP Master na PC“, který svým dílem odvedl velký kus práce. Nemusel jsem proto

řešit problémy, co se týče hardwaru. Mým hlavním úkolem tedy bylo vytvořit Linuxový ovladač, který by implementoval funkce řídicí jednotky mastera na sběrnici Profibus. Dále vytvořit dobrý základ, aby bylo možné práce v budoucnu přenést i na jiné operační systémy.

Tato práce měla být původně přepsaní ProfiMu do operačního systému Linux a na základě toho využít už funkčního základu z předchozí práce. Po zralé úvaze jsem se ale rozhodl pro napsaní ovladače od začátku. Opíral jsem se o některá řešení složitých problémů z ProfiMu, hlavně co se týče přepínání toku dat, které je velmi důležité pro správnou funkčnost ovladače. Snažil jsem se také o to, aby použití a rozšíření této práce v budoucnosti bylo možná co nejjednodušší. Proto struktura PBMaster je velmi odlišná od ProfiMu.

## 1.3 Struktura práce

V příští kapitole se podíváme více na vrstvenou architekturu Profibusu DP. Stručně porovnáme jeho model s modelem ISO/OSI, ze kterého vychází. Dále následuje podrobnější popis linkové vrstvy, zejména část FDL, která nás bude nejvíce zajímat. Popíšeme si funkce stavového automatu FDL vrstvy včetně jeho chování a přechody mezi stavy. Nezapomeneme ani na popis, jak řeší FDL vrstva při událostech na sběrnici.

Kapitola o sériové komunikaci se zabývá popisem standardů RS-232 a RS-485. Přiblíží rozdíly a výhody, resp. nevýhody těchto dvou standardů, které souvisejí s touto prací. Dále na tuto kapitolu navazuje popis realizace fyzické vrstvy, která byla převzata z ProfiMu.

Jelikož psaní ovladače představovalo podstatnou část této bakalářské práce, popíši později některé zkušenosti a poznatky při psání modulu pro Linux. Názorně si to předvedeme také na jednoduchých příkladech. Tato kapitola se pokusí seznámit čtenáře s programováním ovladače v Linuxu, avšak nemůže nahradit knihy zabývající se touto problématikou.

Následně se už podíváme podrobněji na strukturu a provedení PBMastera. Popíši některé řešení a jeho důvody, proč jsem to udělal. Nakonec popíši postup při instalaci ovladače a jeho použití si ukážeme na příkladech.

# 2 Profibus DP

V této kapitole se více seznámíme s touto variantou Profibusu. Popíšeme si postupně 3 vrstvy ISO/OSI modelu užívané u Profibusu, u kterých se soustředíme hlavně na popis Linkové vrstvy, která je pro pochopení Profibusu nejdůležitější. Podíváme se také na to, jak spolu stanice na sběrnici spolu komunikují, jak řeší různé situace a uvedeme si také některé požadavky pro správnou funkci sběrnice, hlavně co se týče časování.

## 2.1 Vrstvená architektura Profibus DP

Na obrázku vidíme model Profibus DP, který má vrstvenou architekturu postavenou podle modelu ISO/OSI. Profibus DP z něj definuje pouze fyzickou, linkovou a aplikační vrstvu. Zbylé čtyři vrstvy jsou částečně zahrnuty v ostatních vrstvách.

```
                        ISO/OSI                 PROFIBUS User
                +------------------+-----------------------------------+
   Layer        ! Application, App !               FMA7                !
     7          !                  !                                   !
  -----------   +------------------+-----------------------------------+
   Layer        !    Presentation  !           ^                ^      !
     6          !                  !           !                !      !
                +------------------+           !                !      !
   Layer        !      Session     !           !                !      !
     5          !                  !           !                !      !
                +------------------+           !                !      !
   Layer        !     Transport    !           !                !      !
     4          !                  !           !                !      !
                +------------------+           !                !      !
   Layer        !      Network     !           v                v      !
     3          !                  !        FDL-User        FMA1/2 User!
  -----------   +------------------+-----------------------+-----------+
                !                  !   Interface Services  !    M      !
   Layer        !                  !                       !    A      !
     2          !       Link       ! Medium Access Control !    N      !
                !                  !                       !    A      !
                !                  ! Transmission Protocol !    GM     !
                !                  !          (FDL)        !    EE     !
  -----------   +------------------+-----------------------+     N     !
   Layer        !                  !                       !     T     !
     1          !  Physical Layer  ! Physical Layer (PHY)  !           !
                !                  !                       !  FMA1/2   !
  -----------   +----------+---+---+----------+---+--------+-----------+
                -----------+   +--------------+   +---------------------
    "Layer 0"                     Transmission Medium
                -------------------------------------------------------

   FMA7  : Fieldbus Management Layer 7
   FMA1/2: Fieldbus Management Layers 1 and 2
   FDL   : Fieldbus Data Link
   PHY   : Physical
```

*Obrázek 1: ISO/OSI model a Profibus DP*

## 2.2 Fyzická vrstva

Fyzická vrstva definuje požadavky na vlastnosti přenosového kanálu, které jsou přenosové médium, popis topologie sítě, možné rychlosti komunikace a modulační kódování dat na sběrnici. Přenosovým médiem pro Profibus DP je **sběrnice RS-485** nebo **optické vlákno**. Tím jsou určeny fyzické vlastnosti kanálu a kódování dat na sběrnici.

Důležitou vlastností, která je normou pro fyzickou vrstvu definována, je rychlost komunikace. Pro Profibus DP jsou definovány rychlosti **9,6 kbaud**, **19,2 kbaud**, **187,5 kbaud**, **375 kbaud**, **750 kbaud**, **1,5 Mbaud**, **3 Mbaud**, **6 Mbaud**, **12 Mbaud**. Aby bylo možné přijmout bezchybně celý rámec, norma pro ni definuje maximální přípustnou odchylku **±0,3 %**, která může v praxi znemožňovat dosažení některých komunikačních rychlostí. Například pokud má UART nevhodnou základní hodinovou frekvenci a pouze hrubé možnosti dělení této frekvence. Proto je důležitá správná volba základní hodinové frekvence pro vysílací a přijímací obvody. Praxe však ukazuje, že přijatelná odchylka rychlosti je až 1%.

## 2.3 Linková vrstva

Úkolem linkové vrstvy je řídit přístup na sběrnici – MAC (Medium Access Control), sestavovat vysílané rámce, dekódovat příchozí rámce a poskytovat vyšší vrstvě služby datové výměny se vzdálenými stanicemi a služby umožňující řízení a správu linkové vrstvy. Linková vrstva je tvořena částí **FDL** (Fieldbus Data Link), která zajišťuje její hlavní funkce a částí **FMA** (Fieldbus Management), zajišťující její řízení. Ačkoliv to není úplně přesné, je celá linková vrstva často označována jako FDL vrstva.

Rozhraní linkové vrstvy pro vrstvu aplikační definuje norma ve dvou částech. Jedna část popisuje rozhraní FDL vrstvy, která aplikační vrstvě poskytuje „služby datové výměny“. Druhá část popisuje rozhraní FMA1/2 vrstvy, která aplikační vrstvě poskytuje „služby správy vrstev FDL a fyzické“. Ve většině služeb je vrstva FMA1/2 pouze prostředníkem mezi aplikační vrstvou a FDL, případně fyzickou vrstvou.

## 2.4 Aplikační vrstva

Aplikační vrstva je užívána uživatelem. Představuje většinou knihovny, které poskytují uživateli pohodlnější způsob komunikace s linkovou vrstvou. Využívá FDL vrstvu pro řízení přístupu na sběrnici a vrstvu FMA1/2 pro provádění služeb a předávání parametrů pro správu linkové a fyzické vrstvy.

## 2.5 Popis rámců

### Typy rámců

Základním stavebním kamenem rámců je jeden znak (Octet), který je na sběrnici vysílán v podobě **11-bitového UART znaku** (UART character) podle norem ISO 11777 a ISO 2022 pro asynchronní komunikaci.

```
Transmitted
Bit Sequence 1st 2   3   4   5   6   7   8   9  10  11th
                 20                          27
Information
sig. Bits       LSB                         MSB
        ---+   +---+---+---+---+---+---+---+---+---+---+---
           ! 0 ! b1! b2! b3! b4! b5! b6! b7! b8! P ! 1 !
           +---+---+---+---+---+---+---+---+---+---+
Start bit----! !                                ! !   !---Stop bit
(ST)           !             Octet               ! !        (SP)
               +<----------------------------->+ !
                                                  !-----Parity bit
                                                        even
```

*Obrázek 2: Formát znaku Profibusu*

UART znak se skládá ze start bitu, který má vždy logickou úrověň *„0“*, osmi datových bitů (LSB – bit s nejnižší váhou je vysílán jako první), sudého paritního bitu a jednoho stop bitů. Protože komunikace mezi stanicemi na sběrnici probíhá asynchronně, je třeba, aby byl vysílač i přijímač (popřípadě více přijímačů) synchronizován. K synchronizaci v rámci dochází se sestupnou hranou start bitů. K zajištění správné synchronizace na začátku rámce platí na Profibusu dvě důležitá pravidla. Před začátkem vysílání každého aktivního rámce (požadavek či předání pověření – Token) musí předcházet **33 $t_{BIT}$ů**, kdy je na sběrnici stav *log. „1“*. Neboli před vysíláním musí sběrnice být na dobu **33 $t_{BIT}$ů** v klidovém stavu. Druhým důležitým pravidlem je to, že jednotlivé znaky rámce musí na sebe přímo navazovat, to znamená, že hned za stop bitem jednoho znaku následuje start bit dalšího znaku a není mezi nimi žádná mezera. Tento požadavek se může zdát banální, ale také se může stát problémem v případě, pokud použitý vysílač sériového kanálu nemá FIFO paměť a nemáme možnost pokaždé dostatečně rychle obsluhovat příchozí přerušení.

Komunikační rámce standardu Profibus jsou prostředkem, kterým spolu komunikují stanice na sběrnici. Pomocí rámců jsou přenášeny nejen samotná uživatelská data (např. hodnoty vstupů a výstupů řízené technologie), ale i služební příkazy, konfigurační a diagnostické údaje a v neposlední řadě i předávání pověření v logickém kruhu. Mimo datových a řídicích mechanismů obsahují rámce i kontrolní mechanismy, kterými je kontrolována jejich bezchybnost a které dále zamezují například nechtěnému opakování již zaslaných zpráv, či naopak jejich ztrátám při chybě příjmu.

Každý rámec protokolu začíná znakem **Start Delimiter**, a kromě rámce pověření a rámce krátkého potvrzení (**SC** – Short Acknowledgement) končí ukončovacím znakem **ED** (End Delimiter). Rámec s proměnnou délkou datového pole navíc obsahuje znaky **LE** a **LEr**, které udávají délku informačního pole rámce. Součástí rámce je informační pole. Informační pole obsahuje adresu cílové stanice **DA** (Destination Address), stanice odesílatele **SA** (Source Address), znak **FC** (Frame Control), který má řídící a kontrolní funkci, a blok uživatelských dat. Za informačním polem následuje znak **FCS** (Frame Check Sequence), který má funkci kontrolního součtu rámce. Odlišnou strukturu má rámec pověření (Token) a rámec krátkého potvrzení SC.

Profibus má celkem 5 typů rámců:

```
Rámec bez dat
+----+----+----+----+----+----+
!SD1 ! DA ! SA ! FC !FCS ! ED !
+----+----+----+----+----+----+
     !       L      !
     +<------------>+
```

*Obrázek 3: Formát rámce bez dat*

```
+----+----+----+----+----+----+----+----/ /----+----+----+
!SD2 ! LE ! LEr!SD2 ! DA ! SA ! FC ! DATA_UNIT !FCS ! ED !
+----+----+----+----+----+----+----+----/ /----+----+----+
                    !             L            !
                    +<------------------------>+
```

*Obrázek 4: Formát rámce s proměnnou délkou dat*

```
+----+----+----+----+-----------+----+----+
!SD3 ! DA ! SA ! FC ! DATA_UNIT !FCS ! ED !
+----+----+----+----+-----------+----+----+
     !             L            !
     +<------------------------>+
```

*Obrázek 5: Formát rámce s pevnou délkou dat*

```
+----+----+----+
!SD4 ! DA ! SA !
+----+----+----+
```

*Obrázek 6: Formát rámce pověření (Token)*

```
+----+
! SC !
+----+
```

*Obrázek 7: Formát rámce krátkého potvrzení*

Všech pět znaků pro start delimiter je pečlivě voleno tak, aby Hemmingova vzdálenost těchto znaků byla 4. To znamená, že každé dva znaky se liší minimálně ve 4 bitech. Aby došlo k chybnému přijetí Start Delimiteru, muselo by při přenosu dojít ke 4 chybám v jediném znaku. Tímto je hlavička rámce velmi bezpečně kódována. V případě rámce s proměnnou délkou dat, kde se jeho celá délka může měnit, je ze stejného důvodu opakován znak znamenající délku informačnícho pole rámce.

### Adresové pole

```
   b8                          b1
   +---+---+---+---+---+---+---+---+
   !EXT! 2⁶!                   ! 2⁰!
   +---+---+---+---+---+---+---+---+
       !          Address          !
       +<------------------------->+
      DA = 0 to 127 ;  SA = 0 to 126
```

*Obrázek 8: Pole DA a SA*

Znaky SA a DA rámce určují jeho zdrojovou a cílovou adresu (0 až 126). Cílová adresa může mít případně hodnotu 127, která signalizuje, že jde o **Broadcast** či **Multicast** zprávu. Broadcast zprávy přijmou všechny stanice na sběrnici, Multicast zprávy přijmou jen stanice, které mají aktivovaný daný přístupový bod **SAP** (Service Access Point). Zprávy Multicast a Broadcast jsou přístupné pouze pro službu **SDN** (Send Data with No acknowledge).

```
        EXT=1
              EXT=0
   -----+----+----+----+----+--------------/ /-------------+-----
        ! DA ! SA ! FC !DAE !                              !
   -----+----+----+----+----+--------------/ /-------------+-----
                        !             DATA_UNIT            !
                        +<--------------------------------->+
         EXT=0
              EXT=1
   -----+----+----+----+----+-------------/ /--------------+-----
        ! DA ! SA ! FC !SAE !                              !
   -----+----+----+----+----+-------------/ /--------------+-----
                        !             DATA_UNIT            !
                        +<--------------------------------->+
         EXT=1
              EXT=1
   -----+----+----+----+----+----+--------/ /--------------+-----
        ! DA ! SA ! FC !DAE !SAE !                         !
   -----+----+----+----+----+----+--------/ /--------------+-----
                        !             DATA_UNIT            !
                        +<--------------------------------->+
```

*Obrázek 9: Formát rámce při použití Address Extension*

Aktivní a pasivní stanice umožňují definovat LSAP (či jen SAP – Link Service Access Point). Jsou to přístupové body, pomocí kterých probíhá přenos dat mezi stanicemi. Pokud vysílaný rámec obsahuje data, mohou v něm být vyslány též tzv. **adresní rozšíření** (Address Extension), do kterých se uvádějí SAP adresu zdrojovou, cílovou nebo obě. Hodnoty přístupových bodů SSAP a DSAP (Source/Destination SAP) jsou předány linkové vrstvě aplikační vrstvou při požadavku služby SDA, SDN nebo SRD. Hodnoty 0 až 62 definují konkrétní SAP. Hodnota 63 definuje **globální přístupový bod** (zpráva je přijata, i když služba není aktivována), který je povolen pouze pro služby SDA a SDN (v případě SDN je používána též v kombinaci s DA=127 pro zasílání Broadcast). V případě použití DAE a SAE je nejvýznamnější bit DA či SA, nastaven na *log. „1“* a DAE a/nebo SAE, které jsou uvedeny na začátku bloku uživatelských dat.

### Pole uživatelských dat

Pole uživatelských dat rámce může mít celkově maximální délku 246 znaků. Mimo vlastních dat může obsahovat maximálně 4 znaky s adresovými extensemi (SAP, Region/Segment adresy). Pro rámce služeb FDL Ident with Reply a LSAP Status with Reply je normou předepsán formát přenášených dat v tomto poli.

### Znak FCS (Frame Control Sequence)

Znak kontrolního součtu rámce FCS vždy předchází znaku End Delimiter. Jeho hodnota je počítána vždy ze součtu znaků informačního pole rámce takto :

$$FCS = (DA + SA + FC + \Sigma UD) \bmod 256$$

kde UD (User Data) je pole uživatelských dat. Počítá ho jak odesílatel, tak příjemce. Pokud u příjemce nesouhlasí vypočítaný FCS s přijatým, je rámec prohlášen za neplatný.

### Znak FC (Frame Control)

Pole určuje typ přenášených zpráv a nese kontrolní informaci, která zabraňuje ztrátám či zbytečnému opakování. Význam jednotlivých bitů jsou uvedené níže.

```
 b8    b7    b6    b5    b4                b1
+-----+-----+-----+-----+-----+-----+-----+-----+
!     !  1  ! FCB ! FCV !  2³               2⁰ !
! Res !Frame!-----+-----+--     Function      --!
!     !  0  !  Stn-Type !                       !
+-----+-----+-----+-----+-----+-----+-----+-----+
Res: Reserved   (the sender shall be set binary "0",
             the receiver does not have to interpret)
Frame Type:  1    Request, Send/Request Frame
             0    Acknowledgement, Response Frame
b7 = 1:
FCB     Frame Count Bit: 0/1, alternating
FCV     Frame Count Bit valid:
        0     alternating Function of FCB is invalid
        1     alternating Function of FCB is valid
b7 = 0:
Stn-Type: Station Type and FDL Status
b6 b5   <-- Bit Position
0  0     Slave Station
0  1     Master Station not ready to enter logical token ring
1  0     Master Station ready to enter logical token ring
1  1     Master Station in logical token ring
```

*Obrázek 10: Pole FC (Frame Control) a jeho příznaky*

```
+------+--------------------------------+--------------------------------+
! Code !                          Function (b1-b4)                        !
!  No  !--------------------------------!--------------------------------!
!      !        Frame Type b7 = 1       !        Frame Type b7 = 0       !
+------+--------------------------------+--------------------------------+
!   0  ! Reserved                       ! ACKnowledgement positive   (OK) !
!   1  ! Reserved                       ! ACK negative                   !
!      !                                ! FDL/FMA1/2 User Error      (UE) !
!   2  ! Reserved                       ! ACK negative                   !
!      !                                ! no Resource for Send Data      !
!      !                                ! (& no Response FDL Data)   (RR) !
!   3  ! Send Data with Acknowledge low ! ACK negative                   !
!      !                                ! no Service activated       (RS) !
!   4  ! Send Data with No Acknow. low  ! Reserved                       !
!   5  ! Send Data with Acknowledge high ! Reserved                      !
!   6  ! Send Data with No Acknow. high ! Reserved                       !
!   7  ! Reserved (Req. Diagnosis Data) ! Reserved                       !
!   8  ! Reserved                       ! Response FDL/FMA1/2 Data low   !
!      !                                ! (& Send Data ok)           (DL) !
!   9  ! Request FDL Status with Reply  ! ACK negative                   !
!      !                                ! no Response FDL/FMA1/2 Data,   !
!      !                                ! (& Send Data ok)           (NR) !
!  10  ! Reserved                       ! Response FDL Data high,        !
!      !                                ! (& Send Data ok)           (DH) !
!  11  ! Reserved                       ! Reserved                       !
!  12  ! Send and Request Data low      ! Response FDL Data low,         !
!      !                                ! no Resource for Send Data (RDL) !
!  13  ! Send and Request Data high     ! Response FDL Data high,        !
!      !                                ! no Resource for Send Data (RDH) !
!  14  ! Request Ident with Reply       ! Reserved                       !
!  15  ! Request LSAP Status with Reply ! Reserved                       !
!      ! (Code No 14 and 15: FMA1/2)    !                                !
+------+--------------------------------+--------------------------------+
```

*Obrázek 11: Pole FC (Frame Control) a význam funkcí*

## 2.6 Stavový automat stanice typu master

```
                                +---------+
                +-------------->!Claim_Tok!------------------------+
                !               !    3    !---------+              !
                !               +---------+         !              !
     PON        !                    ^              !              !
      !         !                    !              !              !
      !         ! +-+            +-+ !              !     +-+      !
      V         ! ! V            ! V !              V     ! V      !
   +-----+    +-----+         +-----------+     +------------+    !
   ! Off !--->!Li_To!-------->!  A_Idle   !---->! Use_Token  !<-+ !
   !  0  !<---!  1  !<--------!     2     !<----!     4      !  ! !
   +-----+    +-----+         +-----------+     +------------+  ! !
    ! ^ ^        ^              ^   ^   ^        ! ^     ! ^    ! !
    ! ! !        !              !   !   !        ! !     ! !    ! !
    ! ! !        !              !   !   !        V !     V !    ! !
    V ! +-----+  !              !   !   !      +-----+ +-----+  ! !
   +------+   !  !      +-------+   !   +------!Await! !Check!  ! !
   !P_Idle!   !  !      !           !          !  5  ! !  6  !  ! !
   !  10  !   !  !      !           !          +-----+ +-----+  ! !
   +------+   !  !      !        +-----+         ! ^     ^ ^    ! !
     !  ^     !  !      !     +->!Ch_To!         +-+     ! !    ! !
     +--+     !  !      !     +--!  8  !                 ! !    ! !
              !  !      !        +-----+                 ! !    ! !
              !  !      !          ! ^                   ! !    ! !
              !  !      !          ! !                   ! !    ! !
              !  !      !          V !                   ! !    ! !
              !  !   +-----+     +-----+                 ! !    ! !
              !  !   !Aw_St!<----!Pass_!-----------------+ !    ! !
              !  !   !  9  !---->!Token!<------------------+    ! !
              !  !   +-----+     !  7  !------------------------+ !
              !  !     ! ^       !     !<-------------------------+
              !  !     +-+       +-----+
              !  !                 ! !
              !  +-----------------+ !
              +----------------------+

   PON  Power On/Reset FDL

   0: Offline                5: Await_Data_Response
   1: Listen_Token           6: Check_Access_Time
   2: Active_Idle            7: Pass_Token
   3: Claim_Token            8: Check_Token_Pass
   4: Use_Token              9: Await_Status_Response
                            10: Passive_Idle
```

*Obrázek 12: Stavový automat FDL vrstvy*

### Offline

Do stavu vstupuje master ihned po zapnutí napájení. V tomto stavu jsou inicializovány pracovní parametry a je proveden self-test. Při inicializaci parametrů je načtena konfigurace a jsou inicializovány datové struktury FDL vrstvy. Provedení self-testu je závislé na konkrétní implementaci mastera a obvykle zahrnuje test obvodů vysílače a přijímače. Aktivní stanice v tomto stavu nepřijímá, ani nevysílá žádné rámce na sběrnici. Ve stavu odpojení od sběrnice je uzavřen vnitřní loop-back a vyzkoušen testovací přenos dat. Po úspěšné inicializaci a self-testu přechází master do stavu „Listen_Token“.

Další důvod pro přechod do stavu může nastat tehdy, pokud je na sběrnici detekována stanice se stejnou adresou nebo pokud je zjistěna hardwarová porucha v přístupu na sběrnici - porucha vysílacích nebo přijímacích obvodů.

**Listen_Token**

Pokud proběhla inicializace úspěšně, je stanice připravena pro vstup do logického kruhu a přechází do stavu „Listen_Token“. V tomto stavu je zatím stále pasivní a pouze sleduje provoz na sběrnici, aby se zjistilo jaké stanice jsou aktivní. To zjišťuje tak, že přijímá a analyzuje všechny rámce typu *„Token Frame“*, pomocí kterých je mezi aktivními stanicemi v logickém okruhu předáváno pověření. Získáváním adres z rámců si stanice vytváří seznam aktivních stanic - LAS (**List of Active Stations**).

Pokud se masteru podaří při vytváření seznamu LAS odposlouchat dvakrát identický oběh tokenu mezi aktivními stanicemi, nastaví svůj stav na *„ready to enter logical token ring“* a dále pokračuje ve sledování provozu na sběrnici a aktualizaci seznamu LAS. Z vytvořeného seznamu LAS master určí svého předchůdce (PS – **Previous Station**) a následovníka (NS – **Next Station**) v logickém okruhu. V tomto stavu zůstává až do příchodu požadavku „Request FDL Status“, který je adresován jemu. Jako odpověď master odesílá svůj stav *„ready to enter logical token ring“* a s bezprostředně následovaným příchodem pověření přechází do stavu „Active_Idle“. Takto je průběh popsán v normě, ale po příchodu rámce pověření se jedná spíše o současný přechod přes „Active_Idle“ rovnou do stavu „Use_Token“.

Se vstupem do tohoto stavu spouští FDL časovač time-outu s periodou $T_{TO}$. Jestliže na sběrnici není do vypršení tohoto časovače detekován žádný provoz, potom FDL předpokládá, že je potřeba inicializovat nebo obnovit logický okruh. Za tímto účelem přechází master do stavu „Claim_Token“.

Při odposloucháváni rámců pověření se může stát, že přijde rámec, jehož adresa odesílatele (SA) je shodná s naší adresou (TS – **This Station**). Jsou-li přijaty dva takové rámce, potom zřejmě stanice s naší adresou na sběrnici již existuje a je zařazena do logického kruhu. Master přechází do stavu „Offline“ a uvědomuje vrstvu FMA1/2 o této chybě.

**Active_Idle**

„Active_Idle“ je stavem, ve kterém je master plně zařazen do logického kruhu, avšak právě nemá pověření k vysílání (token). Sleduje provoz na sběrnici a odpovídá na požadavky, které mu jsou adresovány, případně pouze přijímá data ze sběrnice, které nevyžadují odpověď.

Přijde-li rámec pověření, adresovaný pro nás (DA=TS), přechází master do stavu „Use_Token“. Detekuje-li master, že byl při předávání pověření v logickém kruhu přeskočen

(např. stanice PS předala token stanici NS), přechází do stavu „Listen_Token“. Do tohoto stavu také přechází, pokud zjistí, že na sběrnici vysílá stanice s duplicitní adresou (rámec přijatý z vnějšku má SA=TS).

Se vstupem do tohoto stavu spouští FDL časovač time-outu s periodou $T_{TO}$. Jestliže na sběrnici není do vypršení tohoto časovače detekován žádný provoz, potom FDL předpokládá, že došlo k rozpadu logického kruhu a je potřeba jej obnovit. Za tímto účelem přechází master do stavu „Claim_Token“.

**Claim_Token**

Do tohoto stavu přechází master v důsledku vypršení časovače pro sledování aktivity na sběrnici, a to buď ze stavu „Listen_Token“, nebo „Active_Idle“. Znamená to, že došlo k rozpadu logického kruhu předávání pověření nebo naše stanice je jediný master na sběrnici. Došlo-li k rozpadu logického kruhu, máme již k dispozici seznamy GAPL a LAS a není nutno je znovu vytvářet, proto rovnou přecházíme do stavu „Use_Token“. Aby nedošlo k souběhu současným přivlastněním si tokenu několika mastery, je v hodnotě time-outu $T_{TO}$ zohledněna adresa TS podle vzorce:

$$T_{TO} = 6.T_{SL} + 2.TS.T_{SL}$$

kde $T_{SL}$ je interval časového slotu - **Time Slot**.

Díky tomu se jako první pokouší o znovuobnovení logického kruhu stanice master s nejnižší adresou.

Je-li potřeba logický kruh inicializovat, je nejprve na sběrnici dvakrát vyslán rámec pověření s adresami SA=TS a DA=TS, přesněji se toto provádí ve stavu „Pass_Token“. Je to nutné k vytvoření záznamu s naší adresou v LAS listu ostatních masterů.

**Use_Token**

„Use_Token“ je stav, ve kterém je master vlastníkem oprávnění, a tudíž může inicializovat vysílání na sběrnici. Při vstupu do tohoto stavu nejprve stanice zjišťuje, kolik má času pro využití oprávnění. Z časovače zjistí, jak dlouho skutečně trval tokenu oběh logického kruhu $T_{RR}$ (**Real Rotation time**). Porovnáním s požadovanou dobou oběhu $T_{TR}$ (**Target Rotation time**) je určena maximální doba pro držení tokenu $T_{TH}$ (**Token Holding time**) jako:

$$T_{TH} = T_{TR} - T_{RR}$$

Zbývá-li čas ($T_{TH} > 0$), vybere master z fronty požadavků rámec k vyslání. Přednostně si vybírá první požadavek, který je na řadě, z fronty s vysokou prioritou, anebo je-li prázdná,

potom vybere požadavek z nízkoprioritní fronty. Po vyslání rámce, na kterém je očekávána odpověď, přechází se do stavu čekání „Await_Data_Resp“ a zároveň je spuštěn časovač s hodnotou $T_{SL}$, který zajišťuje vyvolání time-outu, pokud dotazovaná stanice neodpovídá. Po návratu z „Await_Data_Resp“ nastává přechod do stavu „Check_Access_Time“.

Je-li požadavek z fronty takového typu, který nevyžaduje vysílání na sběrnici (např. požadavek na aktivaci SAP), je zpracován, odpověď je poslána vyšší vrstvě a master přechází do stavu „Check_Access_Time“.

Je-li při vstupu do stavu čas ($T_{TH}$) vyčerpán, tj. ($T_{TH}$ <= 0), může master ještě zpracovat jeden požadavek s vysokou prioritou. Není-li žádný požadavek s vysokou prioritou ke zpracování, master přechází do stavu „Check_Access_Time“.

**Await_Data_Response**

„Await_Data_Response“ je stav, ve kterém čekáme na odpověď po vyslání rámce požadavku. Jestliže vyslaný rámec byl typu SDN (Send Data with No Acknowledge), nečekáme na odpověď a ihned se navracíme do stavu „Use Token“. Při čekání můžeme přijmout tyto tři typy rámců:

1) platné potvrzení nebo odpověď od stanice, kterou jsme adresovali . V tomto případě odpověď zpracujeme a přecházíme zpět do stavu „Use_Token“,
2) platný rámec, avšak nečekaného typu (např. rámec pověření) nebo od jiné stanice než očekáváme. Zřejmě někde nastala chyba, přecházíme tím do stavu „Active_Idle“,
3) neplatný rámec, tj. například rámec s chybným Start Delimiterem, End Delimiterem, chybným kontrolním součtem nebo chybnou paritou některého znaku. Přijde-li tato odpověď nebo vyprší časovač „Slot Timer“, je požadavek opakovaně vyslán. Neodpovídá-li stanice ani na opakované požadavky, je o tom uvědoměna vyšší vrstva a master přechází do stavu „Use_Token“.

**Check_Access_Time**

V tomto stavu master kontroluje, jestli mu ještě zbývá čas k držení oprávnění. Zbývá-li času dostatek ($T_{TH}$ > 0), vrací se do stavu „Use_Token“. Je-li všechen čas vyčerpán, přechází do stavu „Use_Token“.

**Pass_Token**

V tomto stavu se FDL pokouší předat pověření následující stanici v logickém okruhu (NS). Zároveň s vysíláním rámce pověření by měl master současným příposlechem ověřit, jestli

se rámec skutečně daří vyslat. Nepřichází-li z příposlechu nazpět stejný rámec, je zřejmě chyba ve vysílacích obvodech a FDL přechází do stavu „Offline“.

Jestliže ještě před vysláním rámce pověření vyprší „**GAP Update Time**“, je obnoven nejstarší záznam v seznamu GAP tak, že master vyšle testované stanici z jemu příslušného GAPu rámec typu „FDL Request Status“ a přejde do stavu „Await_Status_Response“. Na požadavek mohou přijít dvě odpovědi:

1) je přijata odpověď od stanice typu slave nebo na požadavek nepřišla žádná odpověď (stanice s danou adresou na sběrnici neexistuje nebo nekomunikuje), záznam v seznamu GAP je aktualizován a po vyslání pověření pro NS přechází master do stavu „Check_Token_Pass“,
2) je přijata odpověď od stanice typu master, která je připravena vstoupit do logického kruhu. V tomto případě odpovídající master je označen jako NS (Next Station) a seznam GAP je příslušně zkrácen. Současně je tomuto masteru odeslán rámec pověření a FDL přechází do stavu „Check_Token_Pass“.

### Check_Token_Pass

Master přechází do tohoto stavu, aby počkal po dobu $T_{SL}$ na ověření úspěšného předání pověření. Pokud master přijme platný začátek rámce v této čekací době, přepokládá se, že pověření bylo přijato a následující stanice začala vysílat. Master přechází do stavu „Active_Idle“. Do stavu „Active_Idle“ přejde i v případě, že master přijme neplatný začátek rámce, protože se předpokládá, že začala vysílat jiná stanice. Pokud není přijat žádný začátek rámce než $T_{SL}$ vyprší, vrací se master zpět do stavu „Pass_Token“.

### Await_Status_Response

Do tohoto stavu přechází master po vyslání požadavku ke zjištění stavu stanice na dané adrese. Při vstupu do tohoto stavu je spuštěn časovač s intervalem $T_{SL}$. Přijde-li odpověď nebo vyprší-li časovač (stanice na dané adrese není nebo neodpovídá), přechází FDL do stavu „Pass_Token“, kde je výsledek zpracován.

Další možností je, že při čekání na odpověď přijde rámec jiného typu, než jaký je očekáván, potom FDL předpokládá, že jiná stanice je aktivní a přechází do stavu „Active_Idle“.

## 2.7 Časovače a čítače FDL vrstvy

K měření doby rotace povění v logickém kruhu, synchronizaci vysílače a přijímače stanic na sběrnici, měření doby pro odpověď příjemce, či k včasnému zjistění chyb sběrnice, je nezbytné realizovat následující časovače a výpočty s tím souvisejících časů. Jednotkou času na Profibusu je Bit Time, který udává délku trvání jednoho bitu na sběrnici.

$$1t_{BIT} = 1/(\text{přenosová rychlost})\ [\text{baud}^{-1}]$$

### Synchronizační čas

Synchronizační čas $T_{SYN}$ (Syn Time) je minimální časový interval, který musí uplynout před začátkem vysílání akčního rámce od okamžiku posledního bitu předchozího rámce na sběrnici. Po dobu $T_{SYN}$ by měla být na sběrnici trvalá logická úroveň *„1“* nebo klid.

$$T_{SYN} = 33\ t_{BIT}$$

Přijímač by měl kontrolovat platnost tohoto intervalu, jinak by měl přijatá data označit za neplatná.

### Token Rotation Timer

Po prvním přijetí pověření aktivní stanicí začíná měření doby oběhu pověření v logickém kruhu, kdy je časovač naplněn hodnotou $T_{TR}$ (Target Rotation Time). Měření oběhu pověření končí při jeho následném přijetí touto stanicí, kdy změřená doba odpovídá hodnotě $T_{RR}$ (Real Rotation Time), a je vypočten rozdíl $T_{TH} = T_{TR} - T_{RR}$. Časovač je následně restartován s hodnotou $T_{TR}$ a začíná nové měření následujícího oběhu. Hodnota $T_{TH}$ (Token Holding Time) je doba, po kterou může master držet pověření, pak ho musí předat dál.

### Idle Timer

Časovač Idle Timer sleduje klidový stav sběrnice. Aktivní master (ve stavu Use Token) nesmí vyslat další rámec, dokud tento časovač nevyprší. Tento časovač měří dobu $T_{ID1}$, pokud poslední vyslaný rámec na sběrnici byl rámec odpovědi či rámec pověření. V případě rámce služby SDN se nečeká na odpověď, tehdy tento časovač měří dobu $T_{ID2}$, od vyslání posledního bitu akčního rámce služby SDN. Během doby $T_{ID1}/T_{ID2}$ musí být na sběrnici *log. „1“*, jinak je časovač stále restartován. Výpočet hodnot $T_{ID1}$, $T_{ID2}$ je uveden ve normě.

### Slot Timer

Slot Timer $T_{SL}$ (Slot Time) je maximální doba, kterou iniciátor akčního rámce čeká od vyslání posledního bitu do přijetí prvního znaku rámce odpovědi. Dále je $T_{SL}$ maximální

doba, po kterou iniciátor rámce povění čeká na příjem prvního znaku jakéhokoliv rámce (očekává se, že následující stanice master - NS, bylo pověření předáno). Pokud doba $T_{SL}$ vyprší ještě před příjmem prvních znaků zmiňovaných rámců, je vyhodnoceno jako chyba a pokud počet opakování akčních rámců jestě nebyl vyčerpán iniciátorem, jsou tyto rámce znovu vyslány.

### Time-out Timer

Tímto časovačem sledují všechny stanice aktivitu na sběrnici. Po vyslání, či příjmu posledního bitu jakéhokoli rámce je časovač naplněn hodnotou $T_{TO}$ a dekrementován s časem až do příjmu prvního bitu následujícího rámce. Pokud časovač $T_{TO}$ vyprší, znamená to pro aktivní stanici reinicializaci logického kruhu. Tato událost je oznámena indikační zprávou uživatelské vrstvě.

$$T_{TO} = 6 \cdot T_{SL} + 2 \cdot n \cdot T_{SL}$$

kde n, je adresa aktivní stanice na sběrnici a $T_{TO}$ tedy vyprší nejdříve u stanice s nejnižší adresou.

### Syn Interval Timer

Aktivní a pasivní stanice používají tento časovač ke sledování sběrnice, zda je zaznamenána synchronizační doba $T_{SYN}$. V okamžiku synchronizace přijímače je časovač naplněn hodnotou $T_{SYNI}$ a s prvním zaznamenaným Start bitem je nastartován. Proces se opakuje, pokud je zaznamenána další synchronizační doba $T_{SYN}$. Pokud časovač vyprší, je to vyhodnoceno jako chyba na sběrnici (zkrat sběrnice - trvalá *log. „0“*, trvale střídající se hrany logických hodnot *„1“* a *„0“*) a je oznámeno uživatelské vrstvě službou Event FMA1/2. Doba $T_{SYNI}$ je definována jako délka dvou cyklů zpráv o maximální délce včetně mezery $T_{SYN}$, tedy:

$$T_{SYNI} = 2 \cdot (2 \cdot (33 + 255 \cdot 11)) + 33 = 11\ 385\ t_{BIT}$$

### GAP Update Timer

GAP Update Timer $T_{GUD}$ (GAP Uptade Time) používá pouze aktivní stanice. Vypršení toho časovače zahajuje občerstvovací cyklus seznamu GAPL. Po ukončení tohoto cyklu je časovač $T_{GUD}$ restartován.

$T_{GUD} = G \cdot T_{TR}$, kde G je tzv. **GAP Update faktor.**

Často je tento časovač realizován pouze jako čítač počtu přijatých rámců pověření a aktualizace GAPL seznamu je provedena tehdy, když čítač dosáhne hodnoty faktoru G. Tato realizace je jednodušší a přitom i přínosnější, protože pokud je na sběrnici malý provoz, pak

zvýšený počet „FDL Status“ rámců nevadí, naopak dosáhneme rychlejšího zapojení nových stanic master do logického kruhu.

## 2.8 Řešení události na sběrnici

Silnou stránkou Profibusu jsou dobře propracované postupy k řešení výjimečných událostí na sběrnici. Většina z nich vyplývá z popsaného stavového automatu, ale některé ukážeme konkrétně.

### Připojení stanice typu master na sběrnici

Po připojení stanice master na sběrnici se stanice chová pasivně a pouze sleduje provoz na sběrnici. Z rámců posílaných po sběrnici si vytváří seznam LAS, a tímto způsobem se zjišťuje adresy dalších stanic typu master. Po úspěšném vytvoření seznamu čeká master na přijetí požadavku od stanice master, v jejímž adresním GAP prostoru leží. Po příchodu tohoto požadavku odesílá odpověď, ve které oznamuje, že je připraven. To způsobí, že je mu vzápětí poslán token, čímž vstupuje do logického ringu a může začít aktivně komunikovat.

### Ztráta tokenu

Dojde-li z nějaké příčiny ke ztrátě tokenu, ustane provoz na sběrnici. Tento stav je stanicemi master detekován a po vypršení time-outu si stanice master s nejnižší adresou přivlastní token a reinicalizuje logický okruh. Délka time-outu je určena podle adresy stanice, proto master s nejnižší adresou se jako první pokusí o reinicializaci okruhu.

### Odpojení stanice typu master ze sběrnice

Je-li stanice typu master náhle odpojena ze sběrnice, mohou nastat tyto dva případy:

1) v okamžiku odpojení vlastnila token jiná stanice master. Až přijde řada na odpojenou stanici, bude se jí předchozí stanice (PS) snažit předat token. Po opakovaném neúspěšném pokusu o předání tokenu odpojené stanici, bude její adresa v seznamu LAS označena jako pasivní a zároveň bude z tohoto seznamu určena následující stanice z logického okruhu, které bude předán token,
2) držela-li odpojená stanice master v okamžiku odpojení token, potom odpojením dojde k jeho ztrátě. Tento stav je řešen podle popisu z předchozího odstavce.

### Dotazovaná stanice neodpovídá

Vyšle-li master požadavek, potom s posledním vyslaným bytem rámce spouští časovač time-out s periodou $T_{SL}$. Nezačne-li do jeho vypršení přicházet odpověď, potom master požadavek opakuje. Počet opakování je určen parametrem sběrnice **retry_ctr**. Neodpovídá-li

stanice ani na opakované pokusy, je označena jako neaktivní.

### Chyba při příjmu odpovědi na dotaz

Povinností každé stanice na sběrnici je udržovat v paměti rámec odpovědi na poslední požadavek ze sběrnice. Dojde-li totiž ke ztrátě nebo poškození této odpovědi, potom je stanicí master požadavek vyslán opakovaně. To, že se jedná o opakovaný požadavek, se pozná podle nezměněného bitu FCB, který je součástí řídícího znaku rámce (FC - Frame Control). Tento bit je s každým novým požadavkem invertován. Dojde-li tedy požadavek se stejným FCB jako ten, který měl požadavek předchozí, opakuje stanice poslední odpověď.

### Duplicita adres na sběrnici

Detekuje-li stanice odposlechem provozu na sběrnici rámce vyslané od stanice se stejnou adresou, potom po přijetí dvou takových rámců přechází do stavu „Offline“ a signalizuje chybu na sběrnici.

# 3 Sériová komunikace

## 3.1 Sériový přenos dat

Jako sériový přenos označujeme takový, kdy se signálové prvky téhož datového proudu předávají za sebou – sériově. Při přenosu informace musíme zabezpečit, aby přijímač správně vyhodnotil okamžiky platnosti jednotlivých značek generovaných vysílačem. Proto vysílač a přijímač spolu musí být nějakým způsobem časově synchronizovány. Právě podle druhu synchronizace rozlišujeme sériový přenos na synchronní a asynchronní.

### Synchronní přenos

Synchronní přenos se děje pomocí izochronního signálu, tedy takového, kde odstup dvou libovolných charakteristických okamžiků (např. začátků a konců jednotlivých značek) je celistvým násobkem určitého (apriorně daného) jednotkového intervalu. Komunikační kanál je tedy taktován společným hodinovým signálem (vedeným zvlášť nebo obsaženém v datovém signálu), který vymezuje intervaly platnosti jednotlivých značek.

Synchronní přenos se nejčastěji používá u bitově orientovaných protokolů, kde se informace seskupuje do rámců. V datových komunikacích se používá zejména pro přenos větších objemů dat.

### Asynchronní přenos

Asynchronní (správněji arytmický) přenos, je druh přenosu, kdy vysílač a přijímač nemají společný hodinový signál, který by vymezoval intervaly platnosti značek. Namísto toho mají obě strany své vlastní hodiny, dostatečně přesné, aby se po fázovém zasynchronizování mohly po několik značkových intervalech považovat za izochronní. Jelikož je třeba hodiny pravidelně synchronizovat, používá se tento způsob nejčastěji pro přenos krátkých bitových posloupností, znaků (5, 6, 7 nebo 8 bitů). Synchronizace probíhá před každým znakem, kdy před prvním významovým bitem vždy předchází tzv. startbit, který je reprezentován opačnou hodnotou signálu, než je klidová úroveň na lince a trvá stejnou dobu jako následující bitové intervaly. Po ukončení sekvence je další příjem opět synchronizován startovní sekvencí. Arytmický přenos ze své podstaty vhodný např. pro komunikaci s počítačem prostřednictvím terminálu.

Profibus DP využívá tento způsob přenosu informace. V případě Profibusu tyto značky přestavuji jeden start bit, jeden stop bit a jeden paritní bit.

## 3.2 Výhody a nevýhody synchronního a asynchronního přenosu dat

**Synchronní přenos**

- Výhodné pro velké objemy dat, přenášené po více vodičích.
- Nutno jednoznačně určit, kdo vysílá synchronizační impulzy.
- Možno použít spojitě proměnnou rychlost přenosu, například podle poměru chybovosti.
- Nutnost synchronizačního vodiče „navíc“ – v podstatě „nepřenáší žádnou informaci“.
- Na straně zařízení nepotřebuje nijak složitou elektroniku.

**Asynchronní přenos**

- Nevýhodné pro velké objemy dat, ale vhodné pro dlouhá vedení, na nichž by synchronizační vodič činil nezanedbatelné finanční náklady.
- Lze použít pro komunikaci mezi mnoha zařízeními.
- Nutno definovat jednoznačně přenosové rychlosti, změnu rychlosti je třeba ošetřit softwarovou sekvencí, která přiměje počítač změnit hardwarově přenosovou rychlost.
- Celkem složitá a drahá elektronika, nutno použít krystalové oscilátory.
- Až o 20 % menší přenosová rychlost užitečných dat při stejné rychlosti komunikace, vzhledem k nutnosti startovacích a paritních bitů.

## 3.3 Popis standardu RS-232

RS-232 je standard pro sériové binární připojení mezi DTE (Data Terminal Equipment) a DCE (Data Communication Equipment). V počítačovém světě bývá DTE počítačový terminál a DCE modem. Konektory DTE mají většinou kolíky a DCE mívají díry (tedy samec na straně DTE a samice na straně DCE). V současné době RS-232 je stále ještě nejvíce používanou normou pro sériovou komunikaci v osobních počítačích.

Osobní počítače kompatibilní s IBM PC používají integrované obvody UART (Universal Asynchronous Receiver Transmitter) různých typů. Ve starých počítačích byly integrované UART čipy zvané 8250. Od 486 byly postupně integrovány obvody 16450, které byly nepatrně rychlejší než 8250. Od Pentia jsou naintegrované do základních desek novější obvody 16550.

UART 8250 a 16450 mají pouze jednubytovou vyrovnávací paměť. To znamená, že tyto obvody generují přerušení po každém vyslaném či příjmajícím bytu. Obvody 16550 byly významně vylepšené oproti 8250 a 16450, neboť nabízí vyrovnávací pamět o velikosti 16 bytů.

V osobním počítači seriové porty bývají označené jako COM1 až COM4. Na tato zařízení lze přístupovat přes standardně definované porty a přerušení uvedená níže.

| | ***IRQ*** | ***Číslo portu*** |
|---|---|---|
| COM1 | IRQ4 | 0x3F8 |
| COM2 | IRQ3 | 0x2F8 |
| COM3 | IRQ4 | 0x3E8 |
| COM4 | IRQ3 | 0x2E8 |

*Tabulka 1: Standardní UART IRQ a I/O porty*

Signály RS-232 jsou reprezentovány napěťovými úrovněmi vzhledem k zemi. Pro každý signál existuje jeden vodič a navíc exituje společná zemní vodič, vůči kterému jsou napěťové úrovně vztaženy. Tento způsob je vhodný pro point-to-point komunikace jako například přípojení myši s počítačem, modem s počítačem atd. Zapojení signálů vyžaduje společnou zem, tudíž se dostáváme k omezené délce vodičů, maximálně do 60m. Hlavním důvodem omezení je odpor dlouhého vedení a snadná narušitelnost. Lze tedy říci, že RS-232 bylo vyvinuto pro komunikaci lokálních zařízení a podporuje jeden vysílač a jeden přijímač.

Signál přerušení z obvodu 8250 nevede v PC do řadiče přerušení (8259) přímo, ale je hradlován signálem OUT2 (signál obecného použití, výstup 8250). Pokud tedy má dojít k přerušení procesoru při události na 8250, je třeba přerušení povolit na třech místech:

1) na 8250 - odmaskovat bity událostí, od kterých se má přerušení generovat (register IER),
2) na 8250 - nastavit signál OUT2 tak, aby signál o přerušení prošel hradlem na řadič přerušení (register MCR),
3) na řadiči přerušení (8259) - povolit IRQ 4, resp. IRQ 3.

## 3.4 Standard RS-485

RS-485 je standard pro sériový přenos dat, který se používá hlavně v průmyslovém prostředí a v systémech pro řízení a přenos dat.

RS-422 a RS-485 používá poněkud odlišný princip: pro každý signál používá jedno twistedpair (TP) vedení, tj. 2 vodiče smotané kolem sebe. Z elektrického hlediska mluvíme o tzv. Balanced data transmision, nebo také Differential voltage transmission. Laicky řečeno, označíme-li si jeden vodič z TP jako A, druhý jako B, pak je-li signál neaktivní, je napětí na A záporné a na B kladné. V opačném případě, signál je aktivní, je A kladné a B záporné. Pochopitelně se jedná o rozdíl (diferenci) mezi vodiči A a B. Pro RS-422 a RS-485 může délka vedení dosahovat až 4000 stop (cca 1200 metrů) a běžně vyráběné obvody dosahují přenosových

rychlostí až 2.5 MB/s.

RS-485 se používá pro multipoint komunikaci, u které více zařízení může být připojeno na jedno signálové vedení. Většina RS-485 systémů používá Master/Slave architekturu, kde má každá slave jednotka svojí unikátní adresu a odpovídá pouze na jí určené pakety. Tyto pakety generuje Master (například PC) a periodicky obesílá všechny připojené slave jednotky.

RS-485 existuje ve dvou variantách:

- **Single TwistedPair RS-485**
  V této verzi jsou všechna zařízení připojená na jeden TwistedPair. Všechna tudíž musí mít třístavové budiče (včetně Master). Komunikace probíhá po tomtéž vedení v obou směrech. Je tedy důležité, aby nezačalo vysílat více zařízení najednou, což je softwarová záležitost.
- **Double TwistedPair RS-485**
  V tomto zapojení Master nemusí mít třístavový výstup, protože Slave zařízení vysílají do druhého twistedpairu, určeného pro komunikaci od slave zařízení k masteru. Toto řešení často umožňuje implementovat multipoint řešení v systémech, které byly původně (jak HW, tak i SW) určeny pro RS-232. Samozřejmá je ovšem nutnost úpravy Master software tak, aby obesílal všechna Slave zařízení dotazovacími pakety. Zřejmé je také zvýšení datové propustnosti při větších objemech dat.

# 4 Realizace fyzické vrstvy

Řešení realizace fyzické vrstvy jsem nemusel řešit, neboť tento nemalý problém byl už vyřešený v diplomové práci Pavla Trnky, „Profibus DP Master na PC“. Pro pochopení a použití PBMaster je však velmi důležité, a proto zde bude uvedené toto řešení předkládané už v ProfiMu. Programové řešení bude uvedené v další kapitole.

Nejjednodušší možností připojení s minimálními náklady je použití standardního sériového portu PC s obvodem kompatibilním s UART 8250 a s jednoduchým převodníkem RS 232/485 bez externího napájení. To nám umožní plnohodnotné připojení na Profibus pro přenosové rychlosti 9600 bps a 19200 bps. Vyšších rychlostí na standardním sériovém portu nelze dosáhnout, pokud porovnáme rychlosti podporované RS-232 a Profibusu v následující tabulce.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| PC RS232 | … | **9600** | **19200** | 38400 | | 57600 | | 111520 | |
| Profibus | | **9600** | **19200** | | 45450 | | 93750 | | 187500 |

*Tabulka 2: Porovnání přenosových rychlostí RS-232 v PC a Profibusu [baud]*

Jinou možností je použít rozšiřující zásuvné PCI karty s obvodem UART (Universal Asynchronous Receiver/Transmitter), který umožňuje posunout rychlostní limit mnohem dále a s vhodným typem obvodu a dostatečně výkonným procesorem lze dosáhnout i momentálně maximální přenosové rychlosti Profibusu 12 Mbps.

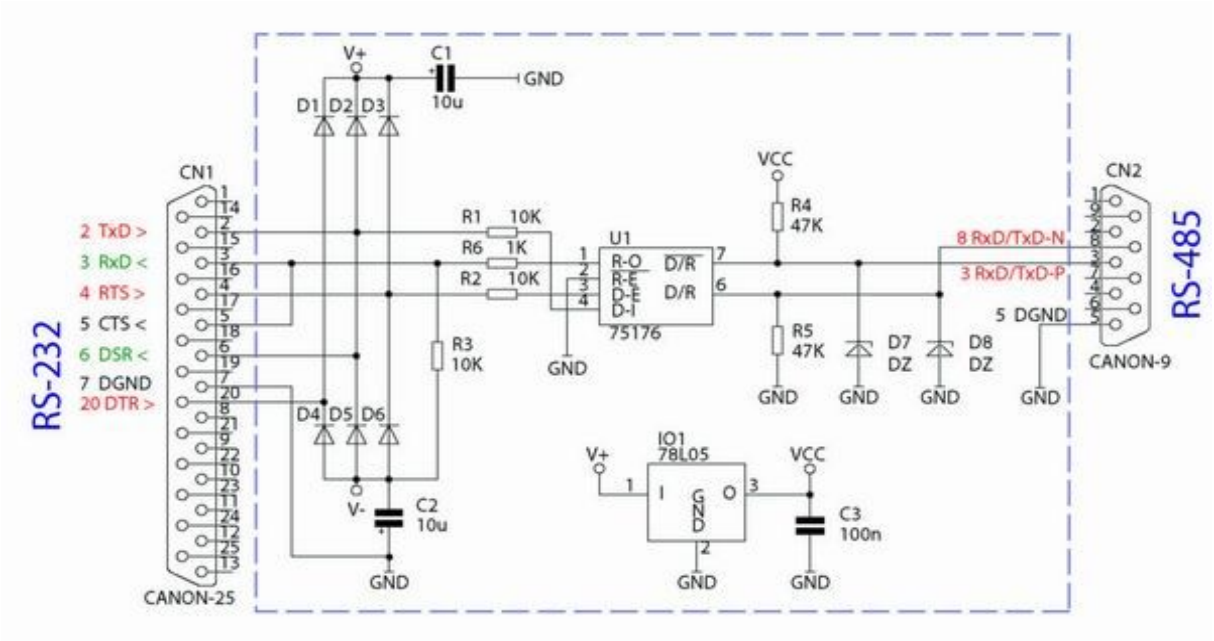
## 4.1 Použití standardního sériového portu

K připojení Profibusu na sériový port PC je třeba použít převodník RS 232/485. Profibus totiž pracuje na standardu RS-485 a sériový port PC na RS-232. Rozdíly mezi těmito standardy byly popsané v kapitole o sériové komunikaci.

Možné schéma převodníku je na obrázku. Jedná se o jednoduchý převodník postavený na obvodu 75176, který ke své činnosti nepotřebuje externí napájení. Napájen je přímo ze sériového portu nastavením výstupu DTR na logickou jedničku. Tím se zjednodušuje použití převodníku, avšak je tím také omezeno celkové maximální zatížení budiče sběrnice. To nám omezuje maximální počet připojitelných zařízení, délku vedení a neumožňuje použití ukončovacích odporů vedení sběrnice nutných pro vyšší přenosové rychlosti.

Potřebujeme-li převodník s lepšími vlastnostmi, nezbyde než použít převodník s externím napájením. Ten kromě výkonového budiče linky může také nabídnout galvanické oddělení, které je vzhledem k obvyklému nasazení standardu RS-485 v průmyslu často používané.

Nezbytnou součástí převodníku je přepínání směru toku dat. K tomu slouží signál RTS.




*Obrázek 13: Jednoduchý interface RS-232/RS-485*

Pro RTS = 0 jsou výstupy budiče linky ve stavu vysoké impedance a převodník je tak přepnut na příjem, pro RTS = 1 jsou budiče sběrnice aktivovány a převodník může vysílat.

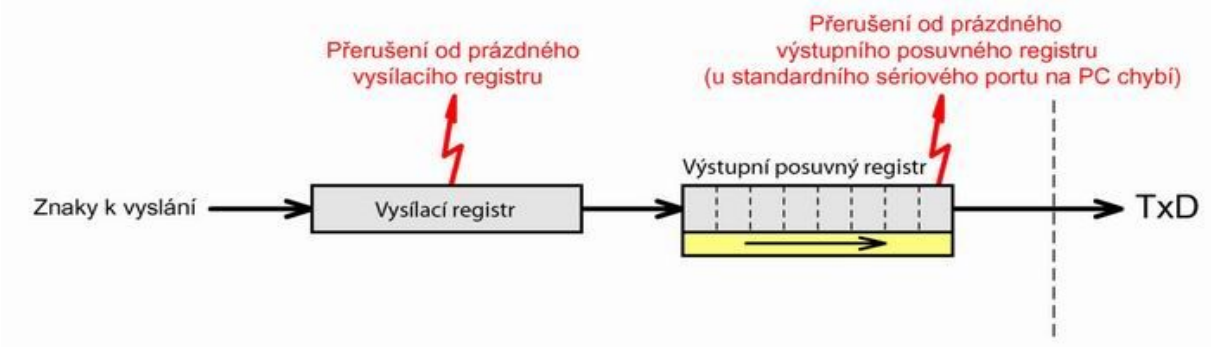
Na tomto místě se hodí poznamenat, že existují převodníky s automatickým přepínáním směru. Princip jejich činnosti spočívá v použití monostabilního klopného obvodu pro řízení směru toku dat. Tento klopný obvod přepíná na vysílání s příchodem sestupné hrany start bitu na začátku každého vysílaného znaku. Perioda klopného obvodu je nastavena na délku jednoho znaku, což znamená, že pro každou přenosovou rychlost je potřeba tuto délku periody změnit. K využití pro komunikaci na Profibusu jej však v našem případě použít nemůžeme, protože potřebujeme pomocí převodníku zajišťovat navíc ještě časování, ke kterému je potřeba softwarové ovládání směru toku dat (o tom bude popsáno později).

Další vlastností tohoto zapojení převodníku je hardwarový loopback. Je-li převodník přepnut na vysílání (RTS = 1), potom vše, co je vysláno do vstupu převodníku signálem TxD se vnitřní smyčkou, se vrací do RxD. Jinými slovy vše, co ze sériového portu PC přes převodník vyšleme se také ihned na vstup sériového portu vrátí. Tuto vlastnost lze využít pro přepínání směru toku dat.

Důležitou částí převodníku, která je PBMasterem také využívána, je propojení mezi výstupem TxD a vstupem DSR. Použijeme-li s PBMasterem jiný převodník než takový, jako je na obrázku, je potřeba, aby měl tuto propojku a hardwarový loop-back pro vysílaná data.

Velkou potíží při připojování sběrnice používající standard RS-485 přes převodník RS 232/485 na většinu obvodů UART s RS-232 je nemožnost generovat přerušení od prázdného **výstupního posuvného registru** (viz. obrázek). Chceme-li totiž použít převodník RS 232/485, potřebujeme softwarově řídit směr přenosu dat. To znamená před každým vysíláním přepnout směr převodníku na vysílání a po vyslání posledního bitu posledního znaku přepnout zpět na příjem. Přepnutí směru na vysílání není problém, to lze provést před vysláním prvního bytu vysílaných dat, avšak přepnutí nazpátek již tak jednoduché není a to právě díky tomu, že většina standardních UART obvodů (viz. tabulka) nenabízí přerušení od prázdného **výstupního posuvného registru**.

Jediné přerušení, které u takovýchto obvodů můžeme při vysílání dostat pouze říká, že je prázdný **vysílací registr**, který tímto můžeme znovu naplnit, avšak **výstupní posuvný registr** ještě prázdný není a teprve dokončuje vysouvání znaku, to znamená, že znak ještě nebyl z UARTu úplně vyslán. Přepneme-li tedy směr po příchodu takového přerušení, dojde k chybě na sběrnici, neboť směr vysílání je změněn uprostřed vysílaného znaku a proto je na sběrnici vyslána pouze jeho část.




*Obrázek 14: Přerušení vysílače UARTu*

Jediný způsob, jak u těchto obvodů zjistit úplné vyslání znaků je přečtení Link Status Registru (LSR), který obsahuje bit indikující prázdný **výstupní posuvný registr**. Ale jelikož od tohoto bitu nemůžeme generovat přerušení, je potom nutné k čekání použít nekonečné smyčky, který není přijatelný způsob, pokud píšeme ovladač a potřebujeme toto čekání provádět v rutině obsluhy přerušení (ISR). Tento problém se však podařilo vyřešit a jeho řešení bude popsáno později.

## 4.2 Použití rozšiřujících zásuvných karet

Pokud srovnáme přípustné přenosové rychlosti, které nabízí standardní RS-232 na PC s možnými přenosovými rychlostmi Profibusu, zjistíme, že do tolerance požadované Profibusem 0,3 % se vejdeme pouze pro rychlosti 9600 bps a 19200 bps. Pro dosažení vyšších rychlostí nám proto nezbývá, než použít zásuvných karet s obvody UART, které mají vyšší základní hodinové

frekvence s dobrými možnostmi jejího dělení. Ty nám umožní nejen dosahovat potřebných rychlostí přenosu (při použití vhodného krystalu základní hodinové frekvence), ale také přinášejí nové možnosti oproti obvodům UART standardně používaný v PC, které jsou kompatibilní s obvodem 16C450.

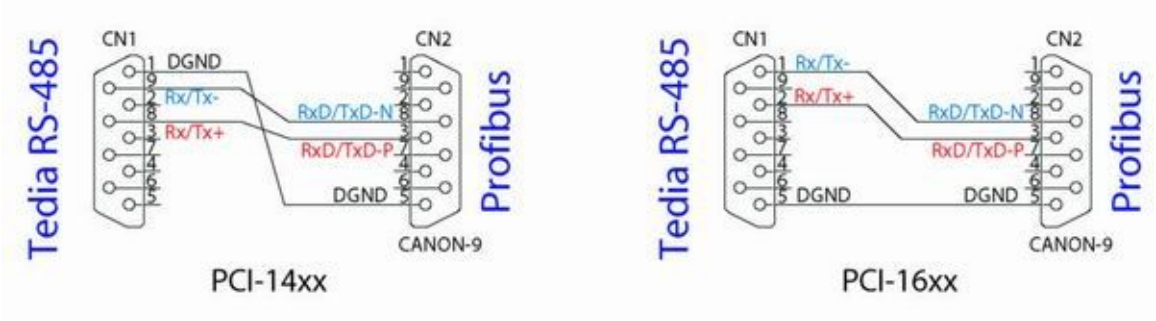
Některé novější obvody UART nabízejí možnost generovat přerušení i od vyprázdnění výstupního posuvného registru. Pokud bychom tuto možnost měli, zjednodušila by se nám tím programová obsluha přístupu na sběrnici a také by se zjednodušil způsob, jakým je sériový port využíván k časování. V následující tabulce jsou uvedeny některé novější obvody UART spolu se základními vlastnostmi. Je vidět, že obvodem, který takové přerušení nabízí, je 16C950. Navíc nové obvody nabízejí velkou vyrovnávací FIFO paměť, která umožňuje snížit zátěž procesoru při vysokých přenosových rychlostech.

| ***Typ UART obvodu*** | ***Velikost FIFO paměti [Byte]*** | ***Přerušení od prázdného posuvného vysílacího registru*** |
|---|---|---|
| 450 | 1 | Ne |
| 550 | 16 | Ne |
| 650 | 128 | Ne |
| 750 | 128 | Ne |
| 950 | 128 | Ano |

*Tabulka 3: Vlastnost UART obvodů*

Jako další možnost hardwarového řešení proto byla vybrána karta pro PCI sběrnici PCI- 482 od firmy Tedia, tato karta je postavena na obvodu OX16PCI954, který je s 16C950 kompatibilní. Stejně tak je možné použít od dané firmy libovolnou kartu z řady PCI-14xx nebo PCI-16xx. Porty na kartě je potřeba pomocí přepínačů přepnout do režimu a pomocí propojky nastavit režim. Dále je potřeba použít jednoduchou redukci, neboť výstupní konektor karty má jiné rozložení signálů než konektor Profibusu.

Použijeme-li jinou kartu s obvodem UART, potom nejlepší je taková, která je postavena také na obvodu kompatibilním s 16C950. Je-li použita karta s obvodem nižšího typu 450 až 750, je zapotřebí, aby měla alespoň možnost datového loop-backu a propojení mezi výstupem TxD a vstupem DSR. Programová obsluha takové karty by pak byla téměř stejná, jako při použití převodníku RS 232/485.




*Obrázek 15: Redukce pro připojení karet Tedia na Profibus*

**Obvod OX16PCI954**

Je to obvod vyráběný firmou Oxford Semiconductor, určený pro použití na rozšiřujících zásuvných kartách pro PCI sběrnici. Jedná se o složitý obvod, který kromě sériových kanálů nabízí i jiné funkce, jako paralelní port, podporu IrDA atd. Pro nás je důležité to, že tento obvod integruje čtyři vysokorychlostní UART kanály, kde každý je kompatibilní s obvodem 16C950 a má 128B FIFO paměti pro vysílač i přijímač.

Podle rychlosti použitého krystalu základní hodinové frekvence může být maximální dosažitelná přenosová rychlost až 15 Mbps pro asynchronní režim. Dobrou vlastností tohoto obvodu je možnost neceločíselného dělení hodinové frekvence, která lépe umožňuje současně dosáhnout několika netypických přenosových rychlostí pro jeden hodinový krystal.

Dále je tento obvod připraven pro připojení na sběrnici PCI a plně integruje podporu Plug and Play s možností načíst konfiguraci z paměti EEPROM. V neposlední řadě umožňuje přerušovat od prázdného vysílacího posuvného registru.

# 5 Tvorba ovladačů pro operační systém Linux

Jelikož psaní ovladače mi zabrala velkou část této práce, chtěl bych v této kapitole ve stručnosti popsat některé poznatky a zajímavé informace, které jsem získal při její psaní. V této kapitole se seznámíme se tvorbou ovladače a ukážeme si to na jednoduchém příkladu. Podíváme se také na to, co je potřeba udělat, aby ovladač mohl komunikovat s opravdovým hardwarem, jak ho řídit a spravovat. Vytvoříme si také jednoduché ukázky obsluhy přerušení a registrace PCI zařízení. Příklad jak by mohl vypadat znakové zařízení, přes které může probíhat komunikace mezi ovladačem a uživatelem, nebude zde uvedený avšak bude dodán s touto prací ve zdrojovém souboru. Rovněž to bude v případě jednoduché knihovny, která bude umožňovat jednodušší práce s ovladačem. Předem bych chtěl upozornit na to, že tato práce pouze seznamuje s psaním ovladače. Pro podrobné pochopení je potřeba si toho vysvětlit mnohem více, ale to už přesahuje rámce této práce. Na to existuje mnoho knih a návodů mnohem lépe napsaných. Jedním z nich je kniha „Linux Device Drivers“, kterou mohu doporučit.

Napsat jednoduchý modul do Linuxu není složité. Psaní ovladače pro fyzické zařízení už představuje nesnadný úkol, aspoň to tak bylo v mém případě. Jelikož jsem nikdy žádný ovladač pro Linux nepsal, zabralo mi hodně času studováním a koumáním nad tímto problémem. Využil jsem k tomu své zkušenosti z programování v jazyku C díky tomu, že jsem už několik let nadšeným uživatelem volně šířitelných operačních systémů a programů.

Tuto práci a příklady k ní dodané jsem vyvíjel na Linuxu s jádrem 2.6.14.3. Bez problému by měl běžet i pod jádrem 2.4. K psaní jsem použil svůj oblíbený textový editor VIM a ke kompilaci standardně dodávaný kompilátor GCC (v mém případě GCC verze 3.3.5).

## 5.1 Kernel space a user space

```
+--------------+
!  User Space  !
+--------------+
    ^      ^
    !      !
 (system calls)
    !      !
    v      v
+--------------+
! Kernel Space !
+--------------+
    ^      ^
    !      !
    v      v
+--------------+
+   Hardware   !
+--------------+
```

*Obrázek 16: Kernel space a user space*

Operační systémy mají většinou paměť rozdělenou na oblasti. Není to tomu jinak ani v případě operačního systému Linux. Při psaní ovladače pro Linux je potřeba si toho uvědomit a držet se tím. Operační paměť v Linuxu je rozdělena do dvou odlišných oblasti:

**Kernel space**

Je to oblast, kam uživatelské procesy mohou přistupovat pouze pomocí volání jádra (system calls). Jsou to služby poskytnuté jádrem, jako například: vstupní a výstupní operace, vytvoření adresáře, změna práv souboru atd. Tyto služby jsou volané aktivními procesy. Aktivní proces je proces, který právě má právo používat CPU. Zde se nachází každá část podprogramu, která je součástí jádra (např. moduly a ovladače zařízení). Je potřeba si uvědomit, že v kernel space nelze používat funkce běžně obsažené ve standardních knihovnách. Lze použít pouze to, co je nabídnuté jádrem nebo jinými moduly.

**User space**

Je část paměti, kde uživatelské procesy běží (všechno ostatní, jen ne jádro). Procesy jsou běžicí programy. Jednou z úkolů jádra je spravovat tyto uživatelské procesy a zabraňovat jim zasahovat do jiných procesů. Uživatelské procesy mohou rovněž přistupovat na fyzická zařízení, ovšem pouze pomocí funkce nabídnuté jádrem, tzn. *system calls*.

**Ovladač zařízení**

Ovladač je program, který přistupuje na hardwarové zařízení, používá a spravuje jeho prostředky. Je vytvořen, aby poskytnul obecnější rozhraní mezi uživatelskými programy a fyzickým hardwarem. Vytvořené obecné rozhraní poskytuje uživatelským programům jednodušší přístup k zařízením. Ovladač je součástí kernelu, to znamená, že sdílí s jádrem paměťový prostor, sdílí s ním kernel space. V Linux/Unixovém světě ovladač lze rozdělit do tří skupin:

1) *znakové zařízení* – je zařízením, kde data jsou přenášena znak po znaku. Softwarová implementace představuje nebufferovaný vstupní a výstupní operace. Každý znak je čten nebo zapisován okamžitě. Je využíván většinou v zařízeních se sériovou komunikací jako např. klávesnice, terminály, tiskárny, myši atd.,
2) *blokové zařízení* – je zařízením, kde data jsou přenášena po blocích. Na rozdíl od znakového zařízení využívá buffer ke vstupním a výstupním operacím. Je často využíván v paralelní komunikaci jako např. pevné disky, magnetické pásky, CD-ROM atd.,
3) *síťové zařízení* – je zařízením pro obsluhu síťových karet. Z programových a výkonových důvodů nespadá do souborových abstrakcí.

### Moduly

Modul je část kódu, který se zaregistruje v kernelu jako ovladač zařízení. Je volán jádrem, pokud je potřeba, aby komunikoval se zařízením. Naopak volá funkce nabídnuté jádrem, aby tyto úkoly provedl. Modul je zkompilovaný kód ve formě objektu a je zaveden do jádra v případě potřeby.

### *make(1)* a Makefile

*make(1)* je nástrojem pro usnadnění kompilace zdrojových souborů a jejich následné slinkování. Tento nástroj je používán i pro překlad jádra a ovladače v Linuxu. Soubor pro nastavení chování programu make je obvykle pojmenován *Makefile* nebo *makefile*.

V souboru Makefile je definován závislosti při překladu zdrojových souborů a reakce na příkazu make*(1)*. *Makefile* se vytváří ručně, pokud jde o jednoduché závislosti. Ve větších projektech je lepší použít některé běžně dostupné nástroje, jako je např. Autoconf.

Použití Makefilu si ukážeme na následujícím jednoduchém příkladu. K tomu jsem si vytvořil 3 zdrojové soubory: *main.c*, *chao.c* a *hola.c*. Hlavní soubor *main.c* volá funkce ze dvou zbývajících souborů. Pro překlad a slinkování těchto souborů je velmi výhodné použít program *make(1)*, neboť bychom po každé museli zadávat ručně dlouhé příkazy pro překlad a slinkování.

```
int main()
{
        chao();
        hola();

        return 0;
}
```

*Výpis 1: main.c*

```
void hola()
{
        printf("Hola, Mundo!\n");
}
```

*Výpis 2: hola.c*

```
void chao()
{
        printf("Chao The gioi!\n");
}
```

*Výpis 3: chao.c*

V tomto souboru *Makefile* jsou 5 záznamů. Každý obsahuje řádek závislosti, který definuje, jak má být soubor sestaven. V prvním řádku se definuje, že *greeting* (které je jméno výstupního programu, uvedené před dvojtečkou) má být sestaven ze tří cílových kódů *main.o*, *chao.o* a *hola.o* (která jsou jména za dvojtečkou). Zároveň tento řádek specifikuje, že příkaz na následujícím řádku se má provést pokaždé, kdy se změní jeden z uvedených cílových kódů. Velmi důležitým pravidlem je, že řádky obsahující příkazy musí začínat minimálně dvěma tabulátory (nesmějí zde být mezery), jinak dostanete hlášení o chybějících oddělovačích. Po spustění příkazu *make(1)* v adresáři, kde se nachází *Makefile* a zdrojové

soubory, se *make* postará o přeložení potřebných souborů, a poté z nich sestaví program s jménem *greeting*.

```
greeting:  main.o chao.o hola.o
        gcc -o greeting main.o chao.o hola.o

main.o: main.c
        gcc -c main.c

chao.o: chao.c
        gcc -c chao.c

hola.o: hola.c
        gcc -c hola.c

clean:
        @rm -f *.o
```

*Výpis 4: první Makefile*

*Make* spustí příkaz *gcc* pouze v případě, že ani jeden ze souborů *main.o*, *chao.o* a *hola.o* neexistuje, anebo kdy soubor *greeting* sice exituje, ale jeden z cílových kódů byl změněn. Zde představuje cílový objekt spustitelný soubor *greeting* a závislosti, neboli nezbytné předpoklady jsou soubory uvedené za dvojtečkou. Poslední cíl *clean* nemá žádnou závislost, a proto je proveden vždy. Ukazuje také, jak lze zavolat příkazy příkazového interpretu.

Tyto znalosti o souboru *Makefile* nám umožňuje pochopit základní chování programu *make* a využít je k psaní modulu. *Makefile* nabízí mnohem více možnosti, které lze najít v mnoha dostupných dokumentech, avšak pro pochopení později uvedených jednoduchých příkladů není potřebné.

**První modul**

Nyní se už podíváme na jednoduchý modul, který vypíše tradiční „Hello World!“. Modul se jmenuje *hello* a je uvedený níže. Tento modul definuje dvě funkce. Jedna je volána při zavedení modulu do jádra (*hello_init*) a druhá při odstranění modulu z jádra (*hello_exit*).

*module_init* a *module_exit* jsou speciální makra definovaná v jádře a říká, jaké funkce má jádro použít při zavedení resp. odstranění modulu. Funkce *printk* je definovaná v jádře a slouží k výpisu, podobně jako *printf* ve standardních knihovnách jazyku C. Důvodem, proč nemůžeme použít *printf* je, že není definován v jádře, ale v knihovnách, které nejsou součástí jádra (neboli není v *kernel space*).

K překladu použijeme níže uvedený soubor *Makefile*. Do seznamu závislostí je přidán náš modul v prvním řádku. K zavedení modulu do jádra se používá příkaz *insmod(8)*. Pro jeho následné odstranění existuje příkaz *rmmod(8)*. Chceme-li zobrazit moduly zavedené v jádře, použijeme příkaz *lsmod*(8). Modul přeložíme a zavedeme pomocí těchto příkazů. Ve výpisu

*lsmod(8)* bychom měli vidět náš nově zavedený modul. V logovacím souboru */var/log/messages* bychom měli vidět výpis „Hello World!“, případně „Good bye!“ při zavedení modulu do jádra, resp. odstranění modulu z jádra. Nesmíme také zapomenout na to, že příkazy pro zavedení a odstranění modulu ovlivňuje funkce jádra a tím i stabilitu celého systému. Proto je dovoleno pouze uživateli *root* (superuser).

```
#include <linux/kernel.h>
#include <linux/module.h>

static int __init hello_init(void)
{
        printk("Hello World!\n");
        return 0;
}

static void __exit hello_exit(void)
{
        printk("Good bye!\n");
}

module_exit(hello_exit);
module_init(hello_init);
```

*Výpis 5: hello.c*

```
obj-m += hello.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

*Výpis 6: Makefile pro hello.c*

```
# make
# insmod hello.ko
# lsmod
Module                  Size  Used by
hello                   1072  0
# rmmod hello
```

*Výpis 7: Zavedení modulu hello*

## Obsluha přerušení (Interrupt Service Routines)

V počítači jednotlivé části většinou nepracují stejnou rychlostí. Vnější zařízení a přídavné karty většinou pracují mnohem pomaleji než procesor. Proto zařízení využívají přerušení, aby signalizovalo procesoru, že se něco změnilo na hardwaru. Díky tomu například procesor nemusí po každém zápisu čekat, než všechny znaky odejdou na vnější sběrnici a může mezitím zpracovávat jiné požadavky. Zrychlí se tím zpracování jiných procesů a zvýší se efektivitu a využití všech prostředků.

Obsluha přerušení je podprogram, který je volán jádrem, pokud přijde přerušení od zařízení, pro které je obsluha přerušení zaregistrován. Činnost obsluhy přerušení spočívá ve zjistění, o jaké přerušení šlo, a provedení následné reakce na tuto událost. Obsluha by měla

být pokud možno rychlá a efektivní, neboť v tomto stavu je procesor blokován pro jiné činnosti. Běžně se to dělá tak, že se v ISR zamaskuje přerušení pro dané zařízení, naplánuje se na později zpracování vstupních a výstupních požadavků a ukončí obsluhu přerušení. Provedení naplánovaných úkolů je ponechán na procesoru, aby je provedl, kdy to uzná za vhodné. Tím lze dosáhnout větší plynulost celého systému. U Profibusu se však vyžaduje mnohem přesnější časování, proto bylo potřeba u PBMaster provádět hlavní úkoly v obsluze přerušení.

Přerušení v Linuxu se zaregistruje příkazem *requeset_irq()* definovaným v *<linux/interrupt.h>*

```
int request_irq (unsigned int irq,
                 irqreturn_t (*handler)(int, void *, struct pt_regs *),
                 unsigned long flags,
                 const char *dev_name,
                 void *dev_id);
```

*Výpis 8: request_irq*

První parametr je číslo přerušení, které chceme zaregistrovat. Další parametr je ukazatel na funkce obsluhy přerušení. Třetí parametr je příznak přerušení. Tímto příznakem lze určit, zda chceme zaregistrovat např. rychlé přerušení (*SA_INTERRUPT*) anebo sdílené přerušení (*SA_SHIRQ*). Další parametr udává jméno vlastníka zaregistrovaného přerušení (toto jméno bude umístěné v */proc/interrupts*). Posledním parametrem se obvykle předává struktura zařízení a pomocí něho lze určit, komu je přerušení adresováno při sdíleném přerušení.

Na následujícím příkladě si ukážeme registrace obsluhy přerušení pro obvody kompatibilní s UART 8250. Je třeba si uvědomit, že signál přerušení z obvodu 8250 nevede v PC do řadiče přerušení (8259) přímo, ale je hradlován signálem OUT2 (signál obecného použití, výstup 8250). Pokud tedy má dojít k přerušení procesoru při události na 8250, je třeba přerušení povolit na třech místech:

1) na 8250 - odmaskovat bity událostí, od kterých se má přerušení generovat (register IER),
2) na 8250 - nastavit signál OUT2 tak, aby signál o přerušení prošel hradlem na řadič přerušení (register MCR),
3) na řadiči přerušení (8259) - povolit IRQ 4, resp. IRQ 3.

V příkladu je uveden jednoduchý modul, který využívá sériový port běžně označovaný jako COM1. Jeho vstupní a výstupní port je na adrese uvedené makrem *U8250_PORT*. Číslo přerušení je definované makrem *U8250_IRQ*. Zařízení má jednoduchou datovou strukturu se dvěma proměnnými. Jedním je čítač přerušení, ve druhém je uložena poslední hodnota registru ISR (Interrupt Status Register).

Pro vstupní a výstupní operace jsou v příkladu používané funkce *inb()* a *outb()*. Jsou to

funkce umožňující čtení a zápis pomocí tzv. *I/O portů*. Stručně řečeno jsou to čísla, která představuje brány umožňující vstupní a výstupní operace s fyzickým zařízením. Kromě přístupu pomocích I/O portů lze vstupní a výstupní operace realizovat také pomocí I/O paměťi. Tento způsob je rychlejší a pokud je to možné, měl by být preferován. Nevýhodou je, že ne každý hardware tento přítup podporuje a navíc užitím I/O paměti způsobí, že ovladač bude závislý na operačním systému, neboť před použitím je potřeba požadující oblast paměti zaregistrovat.

Modul při zavedení nastavuje přenosou rychlost sériové linky. Poté provede všechny kroky, které byly již výše uvedené a které jsou nutné pro povolení přerušení. Poslední krok je registrace obsluhy přerušení. Je rovněž nezbytné přečíst si vstupní buffer, jinak to může zabránit další přerušení. Obsluha spočívá ve výpisu počtů přerušení vyvolané hardwarem. V tomto případě je přerušení vyvolané pravděpodobně chybou na vedení. Po zavedení modulu bychom měli vidět naše zaregistrované přerušení v souboru */proc/interrupts*. Výpisy z obsluhy přerušení lze najít v logovacím souboru */var/log/messages*.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/interrupt.h>
#include <asm/io.h>
#define U8250_PORT              0x3f8
#define U8250_IRQ               4
#define U8250_IER               0x01
#define U8250_IIR               0x02
#define U8250_IIR_NO_INT        0x01
#define U8250_LCR               0x03
#define U8250_DLL               0x00
#define U8250_DLM               0x01
#define U8250_MCR               0x04
#define U8250_RX                0x00

struct uart8250_priv {
        int irq_cnt;
        uint8_t iir;
};
struct uart8250_priv priv;

static irqreturn_t uart8250_isr(int irq, void *data, struct pt_regs *regs)
{
        struct uart8250_priv *priv = (struct uart8250_priv *)data;
        priv->irq_cnt++;

        priv->iir = inb(U8250_PORT + U8250_IIR);
        /* the interrupt isnt for us */
        if (priv->iir & U8250_IIR_NO_INT)
                return IRQ_NONE;
        /* interrupts off */
        outb(0x00, U8250_PORT + U8250_IER);
        printk("I`m ISR: %d\n", priv->irq_cnt);
        /* we dont want more than 10 interrupts */
        if (priv->irq_cnt < 10)
                outb(0x0f, U8250_PORT + U8250_IER);

        return IRQ_HANDLED;
}

static int __init uart8250_init(void)
{
        priv.irq_cnt = 0;
        /* baud rate */
        outb(0x80, U8250_PORT + U8250_LCR);
        outb(0x06, U8250_PORT + U8250_DLL); /* 0x06 = 19,200 BPS */
        outb(0x00, U8250_PORT + U8250_DLM);
        outb(0x1b, U8250_PORT + U8250_LCR);
        /* enable interrupts */
        outb(0x09, U8250_PORT + U8250_MCR);
        outb(0x0f, U8250_PORT + U8250_IER);
        request_irq(U8250_IRQ, uart8250_isr, SA_SHIRQ,
                                        "example_uart8250", &priv);
        inb(U8250_PORT + U8250_RX);
        return 0;
}

static void __exit uart8250_exit(void)
{
        free_irq(U8250_IRQ, &priv);
}

module_exit(uart8250_exit);
module_init(uart8250_init);
```

*Výpis 9: uart8250.c*

### Psaní PCI modulu

Zásuvné karty podporující PCI standard jsou velmi oblíbené mezi rozšiřujícími kartami užívanými v počítači. Proto si ve stručnosti uvedeme jednoduchý příklad registrace PCI zařízení.

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/pci.h>

/* Supported PCI device IDs */
static struct pci_device_id simplepci_card_ids[] = {
        {0x1415, 0x950A, PCI_ANY_ID, PCI_ANY_ID, 0, 0, 0x16954000},
        /* required last entry */
        {0,}
};
MODULE_DEVICE_TABLE(pci, simplepci_card_ids);

static int simplepci_pci_probe(struct pci_dev *pdev,
                                const struct pci_device_id *ent)
{
        printk("PCI device found at slot %s (IRQ=%d)\n", pci_name(pdev), pdev->irq);
        return 0;
}

static void simplepci_pci_remove(struct pci_dev *pdev)
{
        printk("device removed\n");
}

static struct pci_driver simplepci_driver = {
        .name           = "simple pci module",
        .id_table       = simplepci_card_ids,
        .probe          = simplepci_pci_probe,
        .remove         = __devexit_p(simplepci_pci_remove)
};

static int __init simplepci_init(void)
{
        int retval=0;

        retval = pci_register_driver(&simplepci_driver);
        if (retval)
                printk("Unable to register PCI module!\n");

        return retval;
}
static void __exit simplepci_exit(void)
{
        pci_unregister_driver(&simplepci_driver);
}
module_exit(simplepci_exit);
module_init(simplepci_init);
```

*Výpis 10: simplepci.c*

Pro registraci PCI zařízení potřebujeme nadefinovat, o jaké zařízení se zajímáme (zařízení, které chceme používat, pokud se najde při detekci). K tomu potřebujeme identifikační číslo výrobce karty a id označení samotné karty, popřípadě další idenfikační čísla. Tato čísla jsou přidělovaná mezinárodní organizací a výrobce by je měly dodržovat. Jejich hodnoty lze najít v souboru */usr/share/misc/pci.id.* V našem případě jde o kartu s čipem od *Oxford*

*Semiconductor* (id výrobce je *0x1415*) a s integrovaným čipem, který má id označení *0x950A*. Speciálním makrem *MODULE_DEVICE_TABLE* ovladač oznamuje, jaká konkrétní zařízení může podporovat.

Dále je potřeba nadefinovat datovou strukturu *struct pci_driver*, ve kterém se uvede jméno zařízení, tabulka PCI zařízení, o kterých se zajímáme, funkce, které se mají zavolat pokud se najde podporované zařízení (funkce *probe()*), případně funkce vyvolané při odstranění modulu s již zaregistrovaným hardwarem (funkce *remove()*). Podrobnější popis o PCI zařízeních a jejich datových struktur lze najít v souboru *Documentation/pci.txt* ve zdrojácích jádra Linuxu.

Funkce *probe()* se zavolá, pokud při detekci byla nalezená zařízení. Ve funkci *probe()* se obvykle inicializují datové struktury, registruje obsluhu přerušení, inicializuje frontu pro naplánových úkolů, registruje se oblasti I/O paměti atd. Ve funkci *remove()* se naopak uvolňuje držené zdroje, neboť je vyvolána při odstranění ovladače.

# 6 Struktura a provedení PBMaster

V této kapitole se seznámíme podrobněji s ovladačem. Popíši funkce a činnosti jednotlivých modulů. Později v kapitole bude popsán způsob využití převodníku k časování, převzaté z ProfiMu.

## 6.1 Struktura PBMaster

Při návrhu jsem dbal hlavně na to, aby projekt umožňoval jednoduché rozšíření do budoucna, a to jak co se týče nových hardwarů, tak i nových funkcí. Funkce celého programu jsem proto rozděloval do jednotlivých modulů. Z nichž hlavní modul zajišťuje funkce stavového automatu FDL vrstvy. Další částí představují moduly pro operace s vlastním čipem. Nakonec jsem vytvořil modul, který přestavuje rozhraní mezi aplikační vrstvou a ovladačem. Struktura PBMastera je uvedena na obrázku.

Současná verze PBMasteru obsahuje celkem čtyři moduly, z nichž jsou dva čipové:

1) *pbmcore* – je hlavní část PBMaster. Implementuje FDL vrstvu Profibusu. Ke komunikaci s hardwarem a k časování používá operace nabídnuté čipovými moduly. Přijímá požadavky od vyšší vrstvy (pbm_fma) a po zpracování a přijetí odpovědi od nižší vrstvy je vrací zpět vyšší vrstvě,
2) *pbm_950pci* a *pbm_8250* – implementují operace s hardwarem. Nabízí vyšší vrstvě operace pro nastavení čipu, časování, čtení a zápis vstupních a výstupních dat,
3) *pbm_fma* – představuje rozhraní mezi aplikační vrstvou a FDL vrstvou. Je realizované jako znakové zařízení, do kterých uživatelský program zapisuje rámce požadavků a z něj čte došlé rámce odpovědi.

## 6.2 Pobrobnější popis jednotlivých částí

### Modul „pbmcore“

Hlavní modul se jmenuje *pbmcore* a implementuje FDL vrstvu Profibusu. Stará se o běh stavového automatu FDL vrstvy. Implementuje funkce obsluhy přerušení. Samotná registrace obsluhy přerušení se však provádí v čipových modulech. Ukazatel na tuto funkci je předán pomocí datové struktury zařízení. Důvodem tohoto řešení bylo, že registrace přerušení je hardwarově závislá operace, a proto by měla být oddělená od hlavního modulu, které nezajímá, o jaké skutečné zařízení jde, ale pouze o některé funkce nabídnuté tímto čipovým modulem. O tom bude podrobněji uvedeno později.

```
      pbmlib                                +--------+ +--------+ +--------+
                                            ! User 1 ! ! User 2 ! ! User 3 !
   (USER SPACE)                             +--------+ +--------+ +--------+
  -------------------------------------------------------+  +---------------
  -------------------------------------------------------+  +---------------
   (KERNEL SPACE)                                          !  !
                 +-------------------------+        +-----+  +------+
   (MODULE NAME) ! UART DEVICE SHARED DATA !        ! /dev/pbmaster !
                 +-------------------------+        +---------------+
      pbm_fma    ! +---------------------+ !   +---- pbm_copy_from_fma()
                 ! ! Queue of Responses  ! !   ! +-- pbm_copy_to_fma()
                 ! +------+      +-------+ !   ! !
                 !        ! !  ^ !         !   v v
  ---------------+--------+ !  ! +---------+--------------+  +---------------
                 !        ! v  ! !         !
                 ! +------+      +-------+ !
                 ! !    High Priority    ! !
                 ! !  Queue of Requests  ! !
                 ! +   ----------------  + !
                 ! !    Low Priority     ! !
                 ! !  Queue of Requests  ! !
                 ! +---------------------+ !       +-------------------+
                 !  +-------------------+  !       !                   !
      pbmcore    !  ! Interrupt Service !  !       ! FDL State Machine !
                 !  !      Routines     !  !       !                   !
                 !  +-------------------+  !       +-------------------+
                 !     +-----+ +------+    !
                 !     ! LAS ! ! GAPL !    !
                 !     +-----+ +------+    !   +---- read()
                 !      +-----------+      !   !+--- write()
                 !      ! RX Queue  !      !   !!+-- timer()
                 !      +--+     +--+      !   !!!   ....
                 !         ! ! ^ !         !   vvv
  ---------------+---------+ ! ! +---------+--------------+  +---------------
                 !         ! v ! !         !
                 !      +--+     +--+      !
                 !      ! TX Queue  !      !
                 !      +-----------+      !
                 ! +---------------------+ !
                 ! !        Master       ! !
                 ! !    Configuration    ! !
                 ! +---------------------+ !
                 !   +-----------------+   !     +-----------------------+
      pbm_8250   !   ! Chip Operations !   !     !                       !
                 !   +-----------------+   !     ! Device I/O Operations !
      pbm_950pci !   ! read()          !   !     !                       !
                 !   ! write()         !   !     +-----------------------+
                 !   ! int_on()        !   !
                 !   ! int_off()       !   !
                 !   ! timer()         !   !
                 !   ! init()          !   !
                 !   ! ...             !   !
                 !   +-----------------+   !
                 +-------------------------+

  --------------------------+   +------------+   +---------------------------
                            +---+---+---+     +--+---+---+
                            ! UART 8250 !     ! OX16C950 !
                            +---+---+---+     +--+---+---+
  --------------------------+   +------------+   +---------------------------
                             PROFIBUS (Transmission Medium)
  ---------------------------------------------------------------------------
```

*Obrázek 17: Struktura PBMaster*

Jak již bylo uvedeno dříve, hlavní část PBMaster běží v obsluze přerušení. Důvodem jsou vysoké požadavky na časování u Profibusu. Proto pokud by stavový automat FDL vrstvy běžel mimo obsluhy přerušení, neměli bychom jistotu, že budou dodrženy přísné podmínky časování, neboť může mezitím přijít jiné přerušení, a tím bychom ztratili CPU. Tato situace je velmi pravděpodobná, například když na počítači pracují najednou více zařízení vykonávající vstupní a výstupní operace. K realizaci stavového automatu jsem použil velmi chytrý způsob volání a plánování funkcí pomocí zásobníku, který jsem převzal z práce *„ulan – RS-485 communication driver“* od Ing. Pavla Píšy a Ing. Petra Smolíka. Díky tomuto způsobu je možné zdrojový kód napsat velmi přehledně a umožňuje také efektivní plánování a volání požadovaných funkcí.

K časování a ke komunikaci s vlastním hardwarem používá funkce nabídnuté čipovým modulem. Tyto operace využívá prostřednictvím datové struktury *struct pbm_chip_operations*. K tomu se vracíme později.

Dále jsou v něm obsažené funkce pro kódování a dekódování rámců, funkce pro sestavení LAS a GAPL listů, funkce pro práce s frontami zpráv, funkce pro práce s vysílajícími a příjmajícími fronty, a v neposlední řadě také funkce pro registrace čipových modulu.

Modul **pbmcore** sdílí s nižší vrstvou (s čipovými moduly) vysílající a příjmající frontu. Chce-li pbmcore vysílat na sběrnici, zapíše do vysílající fronty byty rámce, které chce vyslat, a poté dá nižší vrstvě vědět, že má vysílat prostřednictvím čipovové operace *write()*. Naopak v případě, že chce přijímat data ze sběrnice, zavolá operace *read()*. Čipový modul načte ze sběrnice a zapíše je do přijímající fronty.

Pro časování se volá operace *timer()*. Předtím je potřeba čítače časovačů nastavit na požadovanou hodnotu.

### Modul „pbm_950pci“ a „pbm_8250“

Tato část implementuje operace s vlastním hardwarem, jako je inicializace hardwaru, vysílání, příjmání a rovněž časování pro vyšší vrstvu (*pbmcore*). Povinné funkce, které modul musí implementovat, předává vyšší vrstvě jako ukazatel na tyto funkce prostřednictvím datové struktury *struct pbm_chip_operations*. Mezi těmito operacemi patří např. čtení z přijímajícího, resp. zápis do vysílajícího bufferu, časování, nastavení přerušení, přepínání směru toku dat, atd.

V této vrstvě se definují vlastnosti stanice, jako adresa mastera (TS), nejvyšší adresa v jeho adresovém prostoru (HSA), hodnoty time-outu atd. Důvodem, proč toto nastavení není v pbmcore, je proto, že by každé fyzické zařízení mohl fungovat jako master s vlastním nastavením. Tzn. že na jednom počítači může běžet např. dvě stanice master a přitom každá bude běžet v jiné síti s jinou přenosovou rychlostí.

**Modul „pbm_fma“**

Tento modul představuje rozhraní mezi aplikační vrstvou a FDL vrstvou. Je realizované jako znakové zařízení, do kterých uživatelský program zapisuje rámce požadavku a z něj čte došlé rámce odpovědi. K předávání požadavku na nižší vrstvu používá funkci *pbm_copy_from_fma()*. Naopak pro čtení odpovědi používá *pbm_copy_to_fma()*. Obě funkce jsou nabídnuté *pbmcore*.

## 6.3 Nároky Profibusu

Pro umožnění práce s Profibusem potřebujeme obecně zajistit plnohodnotnou schopnost komunikovat po sběrnici s RS-485 a především mít možnost časování s krátkými periodami. Pro představu délek těchto period, které potřebujeme dosáhnout, si vezmeme například Profibusu běžícího na rychlosti 1,5 Mbps. Všechny časové intervaly pro Profibus, jako délky timeoutů, maximální zpoždění a nezbytné časové prodlevy, jsou udávány s jednotkou $t_{BIT}$. Ta je definována jako interval daný délkou přenosu jednoho bitu po sběrnici:

$$1t_{BIT} = 1 / (\text{přenosová rychlost v bps})$$

Pro rychlost 1,5 Mbps vychází $t_{BIT}$ = 0,67µs. Z praktického hlediska vzhledem k délkám časovaných period stačí přesnost rozlišení asi desetkrát menší, to znamená pro rychlost 1,5 Mbps potřebujeme dosáhnout minimálních period asi 6 µs, která je z pohledu user módu Linuxu težkodosažitelné. Tam je minimální dosažitelná perioda v řádech milisekund.

K časování tedy nelze použít služby operačního systému. Lze však použít pro sledování časovačů systémový čas, který Linux umožňuje zjistit s vysokým rozlišením. Systémový čas je dán datovou strukturou timeval a představuje počet microsekund od restartu počítače. Jeho přesnos je tedy kolem 1 µs, což je pro naše potřeby dostatečné. Struktura timeval je definována v hlavičkovém souboru *<bits/time.h>*

## 6.4 Ovládání převodníku a jeho využití k časování

**Časování pomocí jednoduchého převodníku RS485/RS232**

Jak již bylo zmíněno, u standardního sériového portu na PC nemáme k dispozici přerušení od prázdného výstupního posuvného registru. Musíme proto použít hardwarových vlastností převodníku, jako je loop-back vracející všechna vyslaná data výstupem TxD na vstup RxD (při přepnutém převodníku na vysílání) a propojení mezi výstupem TxD a vstupem DSR.

Využívání sériového portu spolu s převodníkem, který zajišťuje přístup na RS-485 a zároveň dostatečně rychlé časování, řeší následující módy činnosti:

**Vysílání znaků**

Před začátkem vysílání rámce (bloku dat) přepneme převodník na vysílání (RTS = 1). Díky loop-backu v převodníku bude každý vyslaný znak zpětně také přijat. S příchodem každého přerušení indikujícího příjem znaku zjišťujeme, jestli se nejedná o poslední znak rámce. S jeho příchodem můžeme přepnout převodník zpět na příjem (RTS = 0). Proto k přepnutí nemůže dojít v nevhodnou chvíli (uprostřed posledního vysílaného znaku), neboť ho provádíme až po příjmu přerušení indikujícího zpětný příjem posledního znaku, kdy znak už musel být celý vyslán a nikoliv po přerušení indikujícím prázdný vysílací registr.

**Příjem znaků**

V této situaci je RTS v logické nule, čímž jsou výstupy budičů sběrnice ve stavu vysoké impedanci a sériový kanál tak může přijímat znaky ze sběrnice. Tento stav se ovšem při práci téměř vůbec nevyužívá, neboť pokud nevysíláme, pak pokaždé běží nějaký time-out, například time-out při čekání na odpověď, čemuž odpovídá následující mód.

**Časování se současným příjmem znaků**

Je to nejnáročnější a také nejdůležitější mód. V něm je převodník přepnutý na příjem (RTS = 0), proto smyčka loop-backu je přerušena a převodník tak může přijímat znaky ze sběrnice. Zároveň je však požadováno odměření nějakého časového intervalu (většinou time-out).

Časováním od sériového portu získáme rozlišení přibližně 11 $t_{BIT}$, které odpovídá intervalu potřebnému k vyslání jednoho znaku, který má kromě osmi datových bitů ještě jeden start bit, jeden stop bit a jeden sudý paritní bit. Požadovaný interval v jednotkách $t_{BIT}$ tak vydělíme hodnotou 11 a výsledná hodnota pak určuje, kolik znaků je potřeba vyslat pro odměření požadovaného intervalu. Jelikož je smyčka loop-backu přerušena, můžeme začít do převodníku vysílat libovolné znaky - pro naše účely je, jak uvidíme později, vysílán znak 0x00. Ty se nám tímto nedostanou na vnější sběrnici a ani nijak nenarušují příjem znaků.

Opět zde musíme vyřešit problém týkající se přepínání směru toku dat na převodníku. V tomto módu sice na vnejší sběrnici nevysíláme, ale po příjmu požadovaných dat nebo vypršení time-outu musíme zajistit korektní přepnutí na vysílání, které obvykle následuje.

Dokončení vysílání každého znaku je indikováno přerušením oznamující prázdný vysílací registr, tím dostáváme signál k vyslání dalšího znaku. Když nám přijde přerušení oznamující vyslání posledního znaku, máme problém. Vysílací registr je sice prázdný, ale znak je stále vysouván z výstupního posuvného registru. Řešení pomocí loop-backu z módu vysílání znaků je zde nepoužitelné, neboť loop-back je rozpojen. Okamžité přepnutí směru s příchodem

posledního znaku by způsobilo to, že se časovací znak dostane na sběrnici, což je nepřijatelné.

K řešení tohoto problému je v zapojení převodníku propojka mezi výstupem TxD a vstupem DSR. Tento vstup nám umožňuje generovat tzv. *„přerušení od modemu“*. Toho využijeme následujícím způsobem. Po příjmu přerušení od posledního vysílaného znaku povolíme přerušení od modemu. Toto přerušení, díky tomu, že vysílaný časovací znak má hodnotu 0x00, přijde v okamžiku, kdy je na převodník vysílána náběžná hrana stop bitu. To už si můžeme dovolit přepnout směr na vysílání. Jedničková hodnota stop bitu znamená klidovou hodnotu na sběrnici se standardem RS-485. Po přepnutí ještě zakážeme přerušení od modemu, aby nám další vysílaný znak nezpůsobil přerušení.

**Synchronizační mezera**

Daná mezera se využívá, pokud před vysílaným datovým rámcem potřebujeme vložit časovou prodlevu. Například pro každý rámec dotazu je normou Profibusu vyžadována prodleva o délce 33 $t_{BIT}$. Funguje podobně jako předchozí mód činnosti, pouze neočekává příjem žádných znaků.

Ve skutečnosti ProfiM funguje tak, že na převodník jsou neustále vysílány znaky. Buď se jedná o data určená pro vnější sběrnici RS-485, tedy Profibus, nebo jsou to znaky zajišťující časování, které na vnější sběrnici neprojdou.

Použijeme-li rozšiřující kartu s obvodem UART typu 16C450 až 16C750, potom způsob přístupu na sběrnici Profibus a způsob časování je stejný. Rozdíl je pouze v tom, že některé karty již mají výstup ve standardu RS-485, avšak nutnost softwarově ovládat směr zůstává (kvůli možnosti časování).

**Časování s obvodem 16C950**

S tímto obvodem je práce výrazně jednodušší, jelikož se jedná o obvod, který dává k dispozici přerušení od prázdného výstupního posuvného registru. Tím nám odpadají všechny složité programové konstrukce, které se snažily tento nedostatek obejít.

Na druhou stranu směr toku dat na sběrnici s RS-485 je stále řízen programově a zůstávají i principy použité pro časování. Pouze není potřeba hardwarový loop-back a propojka mezi výstupem TxD a vstupem DSR, jež byly hardwarové prostředky převodníku, které umožnily nahradit přerušení od prázdného výstupního posuvného registru.

Použití tohoto obvodu je velice spolehlivé a rychlostní limit přenosové rychlosti je pouze otázkou výkonu procesoru a maximální přípustné míry zátěže procesoru.

# 7 Použití PBMaster

V této kapitole si popíšeme instalace ovladače a poté si ukážeme jeho použití na jednoduchém příkladu, který bude komunikovat s jednotkou slave Siemens ET200B. Program pošle stanici parametrizaci a konfiguraci. Nakonec změní hodnotu výstupu jednotky.

## 7.1 Kompilace a instalace ovladače

Po rozbalení programu je potřeba ovladače zkompilovat. Ke kompilaci je potřeba pouze standardně dodávané nástroje. Před instalaci je třeba zjistit, zda nejsou už nainstalované jiné ovladače pro zařízení, které chceme použít pro PBMaster. Pokud je všechno v pořádku, můžeme pokračovat v instalaci modulů. Zavedeme nejdříve *pbmcore* a poté čipový modul, v tomto případě je to *pbm_950pci*. Vlastnosti stanice můžeme změnit předáním parametrů při zavedení modulu. Výpis všech nabídnutých parametrů se vypisuje příkazem *modinfo()*. Nakonec zavedeme modul pro komunikaci s aplikační vrstvou. Do logovacího souboru modul *pbm_fma* vypisuje, pod jakým major číslem byl zaregistrován. Toto číslo je důležité pro vytvoření uzlu (znakového zařízení), skrz kterého můžeme komunikovat. Uzel se vytváří příkazem *mknod(1)*. Jako první parametr se předává jméno znakového souboru, které chceme vyvořit. Další parametr určuje, že jde o znakové zařízení. Třetí parametr je již zmíněné major číslo a poslední je minor číslo, které můžeme nastavit na 0. Nyní už můžeme s našim masterem komunikovat a posílat mu požadavky do znakového zařízení a číst z něj odpovědi. Pokud bychom měli např. soubor *framedump*, ve kterém jsou rámce uložené v binární podobě, můžeme poslat požadavky pouhým příkazem *cat(1)* s jménem souboru jako parametr.

```
# make
# insmod pbmcore.ko
# modinfo pbm_950pci
parm:           param_tgapg:GAP G - GAP Update Time (int)
parm:           param_tsl:Tsl - Slot Timer (int)
parm:           param_tto:Tto - Time-out Timer (int)
parm:           param_ttr:Ttr - Token Rotation Timer (int)
parm:           param_retry:Number of retry if no reply received (int)
parm:           param_hsa:Highest station address (int)
parm:           param_ts:Master address (int)
parm:           param_baud_rate:Baud rate (int)

# insmod pbm_950pci.ko param_ts=100 param_baud_rate=1500000
# insmod pbm_fma.ko

# tail /var/log/messages
pbmaster: Allocated pbmaster with major number 254 and minor number 0

# mknod /dev/pbmaster c 254 0
# ls -la /dev/pbmaster
crw-r--r--  1 root root 254, 0 2006-08-17 19:21 /dev/pbmaster

# cat framedump > /dev/pbmaster
```

*Výpis 11: Kompilace a instalace PBMaster*

# Ukázkový příklad

```
#include <stdio.h>

#define MASTERADDR      100
#define SLAVEADDR       8

/* Set parameters of Siemens ET200B */
void et200b_set_conf(FILE *file)
{
        /* parametrization */
        unsigned char param[] = {       0x68, 0x11, 0x11, 0x68,
                                        0x80 | SLAVEADDR, 0x80 | MASTERADDR,
                                        0x4d,   61,   62,
                                        0xb0, 0x08, 0x09, 0x0b, 0x00, 0x0f,
                                        0,    0,    0,    0,    0,    0,
                                        0xff, 0x16 };
        /* configuration */
        unsigned char conf[] = {        0x68, 0x07, 0x07, 0x68,
                                        0x80 | SLAVEADDR, 0x80 | MASTERADDR,
                                        0x4d,   62,   62,
                                        0x20, 0x12,
                                        0xff, 0x16 };
        int i;

        /* send parametrization */
        for (i=0; i<23; i++)
                putc(param[i], file);

        /* send configuration */
        for (i=0; i<13; i++)
                putc(conf[i], file);
}
/* Turn on digital output */
void et200b_on(FILE *file, int output_num)
{
        unsigned char q[] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
        unsigned char conf[] = {        0x68, 0x04, 0x04, 0x68,
                                        SLAVEADDR, MASTERADDR,
                                        0x5d, q[output_num],
                                        0xff, 0x16 };
        int i;

        /* send configuration */
        for (i=0; i<10; i++)
                putc(conf[i], file);
}

int main()
{
        FILE *file;

        file = fopen("/dev/pbmaster", "rw+");

        /* set parameters */
        et200b_set_conf(file);
        /* turn on required output */
        et200b_on(file, 2);

        fclose(file);

        return 0;
}
```

*Výpis 12: test1.c (jednoduchý příklad použití PBMaster)*

# 8 Závěr

Cílem práce na začátku bylo naimplementovat všechny vrstvy od linkové až po aplikační tak, jak to bylo dosažené v ProfiMu. Z časových důvodů se nepodařilo uskutečnit všechny tyto cíle. Podařilo se však dosáhnout funčního ovladače, který umožňuje připojit osobního počítače k průmyslové sběrnici Profibus DP. Byl vytvořen podle mého názoru dobrý základ k tomu, aby se v budoucnu v této práci mohlo pokračovat. Struktura PBMaster je rozdělena do jednotlivých vrstev, které usnadňují rozšíření funkčnosti ovladače, a to jak po stránce podpory nových hardwarů, tak po stránce chování hlavního modulu, který zajištuje funkce FDL vrstvy.

PBMaster umožňuje připojit Profibus přes sériový port s přenosovými rychlostmi 9600 bps a 19200 bps. Vyšší přenosové rychlosti lze dosáhnout díky použití PCI karty s obvodem UART 16C950. S použitím rozšiřující karty se podařilo dosáhnout maximální rychlosti 1,5 Mbps. To bylo omezené především proto, že tolerance rychlosti karty při vyšších přenosových rychlostech již přesahuje tolerance definované Profibusem. Myslím, že zvolením karty s jinou hodinovou frekvencí lze dosáhnout vyšších přenosových rychlostí.

Podařilo se dosáhnout dobrého výkonu ovladače, neboť ani při maximální dosažené přenosové rychlosti zatížení procesoru nepřesahuje 5 % (bylo vyvíjeno a testováno na AMD Athlon(tm) 64 Processor 2800+ s 512 MB operační paměti). Pro ukázku funkčnosti ovladače byly vytvořeny jednoduché příklady, které komunikují s jednotkami na sběnici.

## 8.1 Další vývoj PBMaster

I když ovladač funguje celkem dobře, je potřeba na něm ještě hodně pracovat, aby mohl být využíván v praxi. Přístími kroky bych viděl především v dolaďování a odstranění chyb z tohoto ovladače. Poté by bylo vhodné ovladač přeorganizovat, aby byl oddělený na části závislé na operačním systému a na ty části, který lze používat i v jiných operačních systémech. Dále by se pracovalo na aplikačním rozhraní, které by umožňovalo příjemnější komunikaci se sběrnici. Udělat rozhraní kompatibilní se rozhraní od firmy Siemens tak, jak to bylo vyřešené v ProfiMu, mi přijde velmi rozumné.

Mohlo by se vytvořit i nějaké nástroje, které by umožňovaly analyzovat rámce na sběrnici, a tím nabídnout volně šířitelného analyzátora Profibusu DP. V současné implementaci je velmi jednoduché předělat modul pbmcore tak, aby pouze naslouchával to, co se děje na sběrnici.

# Seznam použité literatury

(1) Trnka, Pavel - „Profibus DP master na PC“, Diplomová práce, Katedra řídicí techniky, ČVUT Praha, 2004.

(2) Krikl, Pavel - „Master Profibus DP v FPGA“, Diplomová práce, Katedra řídicí techniky, ČVUT Praha, 2004.

(3) Pavel Píša, Petr Smolík - „uLan RS-485 Communication Driver“, Praha, 1996-2004.

(4) „Profibus specifikace, verze 1.0“, Karlsruhe, Profibus International, 1998.

(5) Jonathan Corbet, Alessandro Rubini & Greg Kroah-Hartman - „Linux Device Drivers 3rd edition“, O`Reilly, 2005.

(6) Matt Welsh, Matthias Kalle Dalheimer, Terry Dawson, Lar Kaufman - „Používáme Linux třetí vydání“, O`Reilly, 2003

(7) David A. Rusling - „The Linux Kernel“, 1996-1999.

(8) Herout, Pavel - „Učebnice jazyka C třetí vydání“, Nakladatelství Kopp, 2001.

(9) „OX16C954 Data Sheet, revision 1.3“, Oxfordshire: Oxford Semiconductor Ltd., 1999.

### Zdroje z internetu

(10) http://rs232.hw.cz/

(11) http://en.wikibooks.org/wiki/Serial_Programming:8250_UART_Programming

(12) http://en.wikipedia.org/wiki/RS-232

(13) http://docs.linux.cz/linux/LDP/nag2/x-087-2-serial.hardware.html

(14) http://www.hw.cz/Dokumentace/Teorie-a-praxe/ART821-RS-485-%26amp%3B-22.html

(15) http://www.cs.vsb.cz/grygarek/PS/sem/sercomm.html

(16) http://www.linuxjournal.com

# Seznam obrázků

Obrázek 1: ISO/OSI model a Profibus DP....12
Obrázek 2: Formát znaku Profibusu....14
Obrázek 3: Formát rámce bez dat....15
Obrázek 4: Formát rámce s proměnnou délkou dat....15
Obrázek 5: Formát rámce s pevnou délkou dat....15
Obrázek 6: Formát rámce pověření (Token)....15
Obrázek 7: Formát rámce krátkého potvrzení....15
Obrázek 8: Pole DA a SA....16
Obrázek 9: Formát rámce při použití Address Extension....16
Obrázek 10: Pole FC (Frame Control) a jeho příznaky....17
Obrázek 11: Pole FC (Frame Control) a význam funkcí....18
Obrázek 12: Stavový automat FDL vrstvy....19
Obrázek 13: Jednoduchý interface RS-232/RS-485....33
Obrázek 14: Přerušení vysílače UARTu....34
Obrázek 15: Redukce pro připojení karet Tedia na Profibus....36
Obrázek 16: Kernel space a user space....37
Obrázek 17: Struktura PBMaster....48

# Seznam tabulek

Tabulka 1: Standardní UART IRQ a I/O porty....29
Tabulka 2: Porovnání přenosových rychlostí RS-232 v PC a Profibusu [baud]....31
Tabulka 3: Vlastnost UART obvodů....34

# Seznam výpisů

Výpis 1: main.c....39
Výpis 2: hola.c....39
Výpis 3: chao.c....39
Výpis 4: první Makefile....40
Výpis 5: hello.c....41
Výpis 6: Makefile pro hello.c....41
Výpis 7: Zavedení modulu hello....41
Výpis 8: request_irq....42
Výpis 9: uart8250.c....44
Výpis 10: simplepci.c....45
Výpis 11: Kompilace a instalace PBMaster....53
Výpis 12: test1.c (jednoduchý příklad použití PBMaster)....54

# Obsah přiloženého CD

### /docs

veškerá dokumentace v elektronické podobě (specifikace Profibusu, RS-485, tištěné spoje potřebné pro výrobu převodníku, manuály a tutoriály).

### /linux

zdrojové soubory jádra Linuxu 2.6.14.3.

### /pbmaster/src

zdrojové kódy PBMaster a všechny příklady uvedené v této práci.

### /pbmaster/docs

bakalářská práce v elektronické podobě.