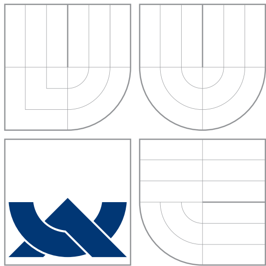
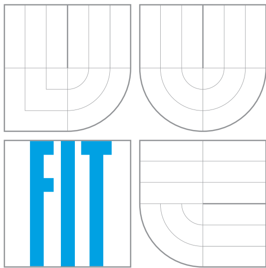


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DMA ŘADIČ A OVLADAČ SÍŤOVÉ KARTY PRO PLATFORMU COMBO2

DMA CONTROLLER AND NETWORK INTERFACE CARD DRIVER FOR COMBO2 PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETR KAŠTOVSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MARTÍNEK
SUPERVISOR

BRNO 2009

Abstrakt

V rámci výzkumné aktivity *Liberouter* společnosti *CESNET* je vyvíjena rodina *COMBO* karet určená pro akceleraci zpracování síťového provozu. Tyto karty disponují programovatelným hradlovým polem firmy *Xilinx*. Aby bylo možné s kartami používat i klasické nástroje pro správu a monitorování sítí, nejen aplikačně specifické nástroje, je nezbytné pro tuto platformu implementovat síťovou kartu, která realizuje příjem a odesílání paketů skrze standardní rozhraní linuxového jádra. Tato práce obsahuje popis návrhu a realizace dvou klíčových komponent síťové karty. DMA řadiče a ovladače zařízení pro Linux.

Klíčová slova

Liberouter, počítačová síť, programovatelný hardware, Linux, paket, přímý přístup do paměti, ovladač zařízení.

Abstract

There is a family of *COMBO* cards used for netork monitoring acceleration being developed on the *Liberouter* project, which is the *CESNET's* research activity. These cards are equipped with *Xilinx's* programmable field array. To enable usage of classic tools for network monitoring and management, not only application specific tools, it is necessary to implement network interface card on the platform, that realizes packet reception and transmission through the standard Linux kernel interface. This thesis describes the design and implementation of network interface card's key components. Those are DMA controller and Linux device driver.

Keywords

Liberouter, computer network, programmable hardware, Linux, packet, direct memory access, device driver.

Citace

Petr Kaštovský: DMA řadič a ovladač síťové karty pro platformu COMBO2, diplomová práce, Brno, FIT VUT v Brně, 2009

DMA řadič a ovladač síťové karty pro platformu COMBO2

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Tomáše Martínka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . .
Petr Kaštovský
22. května 2009

Poděkování

Za odborné vedení, vstřícnost při konzultacích a spoustu dobrých rad bych chtěl poděkovat Ing. Tomáši Martínkovi.

© Petr Kaštovský, 2009.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **4**

2 Teoretický základ **5**

2.1 Počítačová síť a model ISO/OSI 5

2.1.1 Typy sítí 6

2.2 Periferní operace 8

2.2.1 Metody komunikace 8

2.2.2 Pokročilé varianty DMA 9

2.2.3 Řízení periferní operace 10

2.2.4 Typy transakcí 12

2.3 Ovladače zařízení 12

2.3.1 Ovladače zařízení v Linuxu 13

2.3.2 Správa paměti 13

2.3.3 Obsluha přerušení 15

2.3.4 Síťové ovladače 16

3 Popis cílové platformy **20**

3.1 Karta COMBOv2 20

3.1.1 Sběrnice PCI-Express 21

3.2 Platforma NetCOPE 21

3.2.1 Připojení k DMA bufferům 22

3.2.2 Připojení k systémové sběrnici 22

3.3 Rozhraní linuxového jádra 25

3.3.1 Životní cyklus modulu 25

3.3.2 Ovladače PCI zařízení 25

3.3.3 Registrace a obsluha přerušení 26

3.3.4 Síťové rozhraní 26

3.3.5 NAPI: bližší pohled 27

3.3.6 Alokace DMA paměti 28

3.3.7 Operace se strukturou `sk_buff` 28

3.4 Možnosti implementace síťových zařízení 29

3.5 Architektura ovladačů *COMBO* 30

3.5.1 Modul *combo6core* 31

3.5.2 Modul *combov2* 31

3.5.3 Modul *combo-cv2eth* 31

4 Návrh **32**
4.1 Komunikace ovladač - DMA řadič . . . 32
4.1.1 Velikost DMA bufferů . . . 33
4.1.2 Deskriptor . . . 34
4.1.3 Princip činnosti . . . 35
4.1.4 Analýza propustnosti . . . 37
4.2 Ovladač . . . 39
4.2.1 Mapování DMA paměti . . . 40
4.3 Blok řízení DMA přenosů . . . 40
4.3.1 Descriptor download manager . . . 41
4.3.2 Status update manager . . . 41
4.3.3 Rx DMA controller . . . 42
4.3.4 Tx DMA controller . . . 42

5 Popis realizace **43**
5.1 Software . . . 43
5.1.1 Organizace datových struktur . . . 43
5.2 Hardware . . . 44
5.3 Rozbor implementace . . . 46
5.3.1 Software . . . 46
5.3.2 Hardware . . . 46

6 Ověření funkčnosti **51**
6.1 Software . . . 51
6.2 Hardware . . . 53

7 Závěr **54**

A Sekvenční diagram modulu **56**

B Diagram případů užití **57**
B.1 USE CASE diagram . . . 57
B.2 Popis případů užití . . . 58
B.2.1 UC00_insert_module . . . 58
B.2.2 UC01_open . . . 58
B.2.3 UC02_stop . . . 58
B.2.4 UC03_remove_module . . . 58
B.2.5 UC04_hard_start_xmit . . . 59
B.2.6 UC05_tx_timeout . . . 59
B.2.7 UC06_net_device_stats . . . 60
B.2.8 UC07_poll . . . 60
B.2.9 UC08_change_mtu . . . 60
B.2.10 UC09_interrupt_rx . . . 61
B.2.11 UC09_interrupt_tx . . . 61

C Bloková schémata **62**
C.1 Descriptor download manager . . . 62
C.2 Status update manager . . . 63
C.3 Rx DMA controller . . . 64

C.4 Tx DMA controller 65

Kapitola 1

Úvod

V dnešní době snadno dostupného připojení k Internetu a zvyšujícího se počtu zařízení vybavených schopností komunikace po Internetu roste potřeba zvyšovat přenosové rychlosti. Tato potřeba není způsobená jen zvyšujícím se počtem přístrojů, ale také rozšiřováním nabídky poskytovaných služeb. Mezi pokročilé služby kladoucí vyšší požadavky na rychlost výměny dat patří IP televize *IPTV*, IP telefonie *VoIP*, video konference, počítačové hry a celá řada dalších, především multimediálních, aplikací. Rostoucí požadavky potom vytváří takové nároky na infrastrukturu, aby bylo možné obsloužit více klientů či kvalitnější služby.

Podstatnou vlastností sítí využívajících protokol IP, z anglického *Internet protocol*, je, že souvislý tok dat je rozdělen na krátké zprávy, které nezávisle putují sítí. Samotný počítač bez dodatečného vybavení však není schopen zprávy do sítě předat a musí být doplněn o síťovou kartu.

Pokud chceme síť nejen využívat, ale také sledovat či spravovat použitím prostředků jako jsou pasivní monitorování toků nebo detekce vzorů napadení, není běžně dostupný počítač vybavený síťovou kartou postačující řešení.

Pro urychlení uvedených úloh vznikla v rámci výzkumné aktivity *Liberouter* společnosti *CESNET* rodina karet *COMBO*. Tyto karty obsahují programovatelná hradlová pole, zkráceně *FPGA*, sloužící k akceleraci výpočetně náročné části úlohy. Programovatelné pole umožňuje snadno a rychle změnit svou funkci. Počítač vybavený takovouto kartou potom dokáže dynamicky měnit svou roli v rámci sítě.

Významným přínosem síťové karty realizované na akcelerační platformě *COMBO* je nejen možnost dynamické změny funkce, ale také možnost modifikace aplikačního jádra na čipu FPGA. Síťová karta tak může být doplněna například o filtraci paketů prováděnou na plné rychlosti linky. Návrh a implementace dvou klíčových částí právě takové síťové karty postavené na rodině karet *COMBO*, konkrétně *COMBOv2*, je popsána v této práci.

Diplomová práce v první části obsahuje vysvětlení základních pojmů a nezbytný úvod do oblasti počítačových sítí, periferních operací a ovladačů zařízení. Ve druhé části je shrnut popis prostředí, ve kterém budou ovladač a DMA řadiče implementovány. Prostředí a rozhraní jsou obzvlášť důležitá, neboť je nezbytné podřídit jim samotný návrh. Návrhem se pak zabývá následující část. V páté kapitole je popsána implementace. Dále následuje kapitola shrnující ověření funkčnosti implementovaných komponent a ovladače. V poslední kapitole je podáno zhodnocení práce a náměty na možný budoucí postup.

Kapitola 2

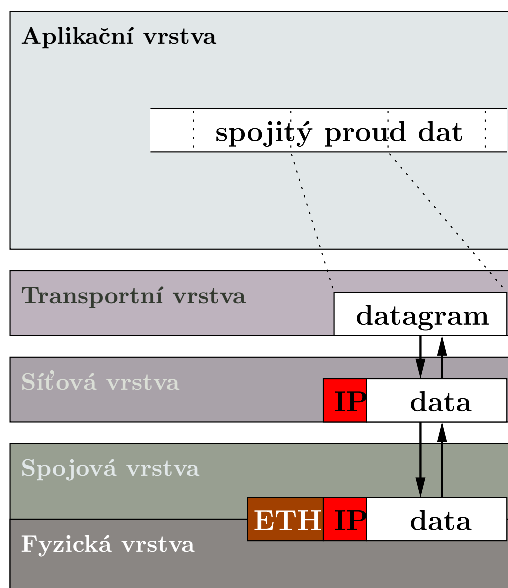
Teoretický základ

Pro objasnění základních pojmů a sjednocení terminologie je uvedeno několik definic, na které bude v dalším textu odkazováno.

2.1 Počítačová síť a model ISO/OSI

Počítačová síť [3] je souhrnné označení pro technické prostředky, které realizují spojení a výměnu informací mezi počítači. Umožňují tedy uživatelům komunikaci podle určitých pravidel za účelem sdílení využívání společných zdrojů nebo výměny zpráv.

Obrázek 2.1: Vrstvy ISO/OSI



Obrázek 2.2: Internetový model

Pro zjednodušení komunikace a jejího řízení byl zaveden vrstvový model ISO/OSI. Problém pod souhrnným označením síťová architektura rozkládá na problémy menší složitosti - vrstvy. Každá vrstva představuje samostatnou úlohu či službu řízenou protokolem vrstvy. Standardizovaný model ISO/OSI je na obrázku 2.1.

Model ISO/OSI se prakticky neujal a slouží pouze jako reference. Namísto toho se používá Internetový model (viz obrázek 2.2), který zachovává čtyři nejnižší vrstvy modelu

ISO/OSI. Zbylé jsou sloučeny do aplikační vrstvy. Internetový model se také často označuje podle dominantně používaných protokolů TCP/IP.

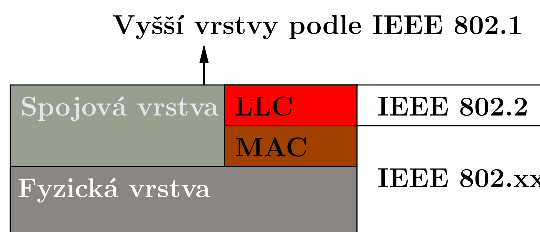
Uživatelský program při komunikaci prostřednictvím sítě využívá aplikační vrstvy. Mezi sebou si pak jednotlivé vrstvy data vyměňují v definovaném formátu. V případě Internetového modelu je na transportní vrstvě rozdělen spojitý proud dat na bloky označované datagramy. Síťová vrstva připojí k datagramům informace důležité pro směrování, jako jsou adresa odesílatele a adresa příjemce. Datagram doplněný údaji síťové vrstvy se označuje *IP paket*. Nižší vrstvy následně doplní paket o hlavičky tak, aby bylo možné bezchybně přenést a korektně sestavit na straně přijímače původní datový tok.

2.1.1 Typy sítí

Každá vrstva síťové architektury, ať už podle modelu ISO/OSI, nebo Internetového modelu, umožňuje alternativní implementace požadované funkcionality. Na nejnižší, fyzické, vrstvě se jedná o různé druhy přenosových medií, ať už metalických, optických či rádiových. Na spojové vrstvě lze nalézt řadu řešení jak přistupovat k médiu a řídit logické spojení nezávisle na fyzické vrstvě. Používají se:

- *Ethernet*,
- *FDDI*,
- *Wi-Fi* a další.

Všechny uvedené typy sítí [8] a mnohé další jsou standardizovány organizacemi *IEEE* (z angl. *Institute of Electrical and Electronics Engineers*) a *ANSI* (z angl. *American National Standardisation Institute*). Standardy IEEE pokrývají fyzickou vrstvu a spojovou vrstvu rozdělenou na samostatné podvrstvy: *MAC* a *LLC*. Organizace IEEE pro tyto vrstvy vypracovala množinu norem *IEEE 802.xx* (viz obrázek 2.3), které převzala i mezinárodní normalizační organizace *ISO* (z angl. *Industrial Standard Organization*) pod označením *ISO 8802*.



Obrázek 2.3: Rozdělení spojové vrstvy a příslušné normy

Linková vrstva, jak už bylo uvedeno, je v normách IEEE rozdělena na následující dvě podvrstvy:

- **LLC** (logical link control) - podvrstva řízení logického spoje,
- **MAC** (medium access control) - podvrstva řízení přístupu na médium.

Předcházející rozdělení je dáno nezávislostí vrstvy LLC a závislostí vrstvy MAC na fyzické vrstvě sítě. Vztah mezi LLC a MAC lze také chápat jako vztah mezi softwarovým a hardwarovým vybavením počítače. Software provádí transformaci dat pro přenos a hardware pro danou síť aplikuje příslušnou přístupovou metodu.

Podvrstva řízení logického spoje tvoří vrchní část spojové vrstvy modelu ISO/OSI (viz obrázek 2.3). Svým horním rozhraním komunikuje se síťovou vrstvou. Je nezávislá na fyzické interpretaci sítě a řídí spoj, tj. vytváří, ruší a kontroluje spojení. Obsahuje též funkce, jež rozpoznávají chyby přijatých dat a řídí jejich výměnu mezi uzly sítě. Vrstva tedy zajišťuje bezpečný přenos dat mezi dvěma uzly sítě bez jejich přímého fyzického propojení. Podvrstva LLC je nezávislá na použité přístupové metodě a popsána normou IEEE 802.2.

Podvrstva řízení přístupu na médium tvoří spodní část linkové vrstvy, má společné rozhraní s fyzickou vrstvou. Proto zabezpečuje hlavně ty funkce linkové vrstvy, které jsou závislé na topologii sítě a použité přístupové metodě. Tato vrstva řídí přístup na médium, určuje časový multiplex, kontroluje správnost přenášených rámců, hodnotí blokování sítě, využívání jinými účastníky apod. Vrstva inicializuje vysílání a příjem dat pro fyzickou vrstvu. Odlišné typy sítí se různí v algoritmech a typech rámců použitých v podvrstvě MAC.

Ethernet: bližší pohled

Pod sítí Ethernet se rozumí lokální počítačová síť, která spojuje jednotlivé počítače prostřednictvím společně využívaného komunikačního média.

Každý počítač v síti pracuje samostatně, nezávisle na ostatních, takže neexistuje centrální prvek. Signály od vysílajícího počítače jsou přenášeny k ostatním uzlům, přičemž v daném okamžiku médium využívá jen jediný vysílající počítač a může začít s vysíláním až po uvolnění média. Přístupová metoda zabezpečující přístup na médium se nazývá kolizní, protože předpokládá možnost současného vysílání stanic na médium. V případě kolize následuje přerušení vysílání a uvolnění média. Síť Ethernet specifikuje norma IEEE 802.3.

V síti Ethernet se používají dva typy rámců podvrstvy MAC. Liší se velikostí a významem jednotlivých bitových polí. Síťové karty musí být nastaveny tak, aby rozpoznávaly zvolený typ rámce. Oba typy rámců zachycují obrázky 2.4 a 2.5.

P	**CA**	**ZA**	**TYP**	**DATA**	**FCS**
8B	**6B**	**6B**	**2B**	**46...1500B**	**4B**

Obrázek 2.4: Formát rámce podle normy ETHERNET II

P	**SFD**	**CA**	**ZA**	**D**	**DATA**	**PAD**	**FCS**
7B	**1B**	**6B**	**6B**	**2B**	**DATA+PAD=46...1500B**		**4B**

Obrázek 2.5: Formát rámce podle normy IEEE 802.3

Kde:

- **P** - preambule, pole pro synchronizaci přijímací stanice,

- **SFD** - *start frame delimiter*, příznak začátku rámce (pouze u IEEE 802.3),
- **CA** - MAC adresa cílové stanice,
- **ZA** - MAC adresa zdrojové stanice,
- **D** - délka dat v bajtech (pouze u IEEE 802.3),
- **TYP** - identifikátor protokolu vyšší vrstvy (pouze u ETHERNET II),
- **DATA** - datová část rámce,
- **PAD** - výplň pro dosažení minimální délky rámce (pouze u IEEE 802.3) a
- **FCS** - *frame check status*, zabezpečení rámce proti chybám cyklickým kódem *CRC*.

V součtu oba typy rámců dosahují minimální délky 72 bajtů a maximální délky 1526 bajtů. Navíc mohou volitelně obsahovat identifikaci virtuální sítě *VLAN* podle standardu IEEE 802.1Q o velikosti 4 bajty. Odečtením délky úvodní sekvence sloužící pouze pro hardwarovou detekci začátku rámce vychází výsledná minimální délka dat v sítích typu Ethernet 64 bajtů a maximální 1522 bajtů včetně identifikátoru VLAN sítě. Tyto hodnoty jsou důležité pro pozdější výpočty.

Je vhodné poznamenat, že výjimku tvoří tzv. *jumbo frames*, které mohou prakticky dosahovat délky 9000 bajtů, teoreticky 64 kB. Praktické omezení je dáno efektivitou výpočtu kontrolního součtu CRC. Teoretické omezení diktuje velikost pole hlavičky udávající délku paketu v bajtech.

2.2 Periferní operace

Periferní, nebo také vstupně-výstupní, zkráceně V/V, operace je souhrnné označení akcí, které je třeba provést v rámci komunikace mezi procesorem a periferním zařízením (obrázek 2.6). Komunikace však také může probíhat mezi periferním zařízením a hlavní pamětí bez účasti procesoru (obrázek 2.7). Příklady periferních zařízení jsou pevný disk, grafická karta či síťová karta.

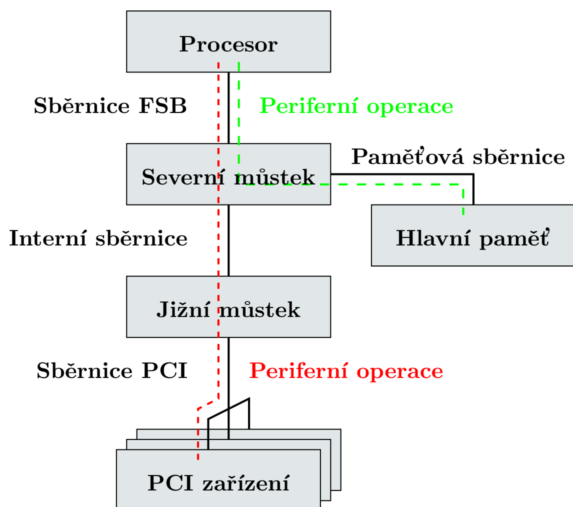
Jednotlivé fáze periferní operace:

1. Zahájení periferní operace. Nastavení příslušných signálů sběrnice. Žádost o sběrnici.
2. Přenos dat. Probíhá komunikace po sběrnici.
3. Ukončení periferní operace. Zjištění stavu periferního zařízení.

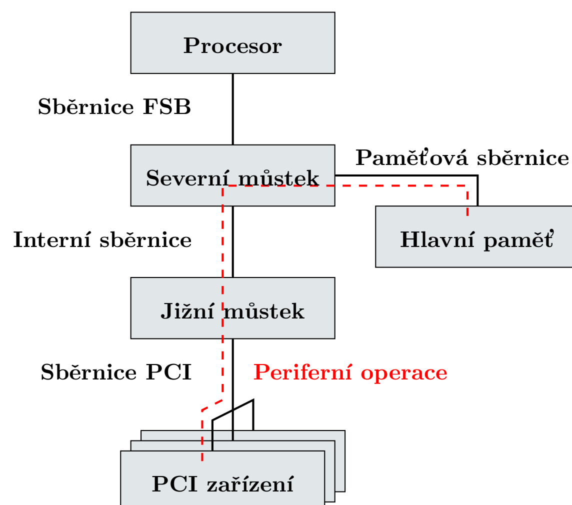
Fáze mohou být dále děleny v závislosti na způsobu připojení periferního zařízení. Pro základní orientaci však postačuje uvedené rozdělení.

2.2.1 Metody komunikace

Jak je znázorněno na obrázcích 2.6 a 2.7, existují dvě metody komunikace mezi hlavní pamětí a periferním zařízením realizující druhou fázi periferní operace, tj. přenos dat. Pro demonstraci principu je na obrázcích naznačena architektura typu severní-jižní můstek, kterou lze stále nalézt ve velkém množství počítačů. Nicméně, postupně je vytlačována modernějšími architekturami. Například *IHA* firmy *Intel*. I přes odlišný způsob komunikace



Obrázek 2.6: Procesorem řízená V/V operace



Obrázek 2.7: DMA

mezi prvky čipové sady lze nadále v počítači provozovat systémové sběrnice, jako jsou například PCI či moderní PCI-express, a tudíž i zařízení pro tyto sběrnice určená. Základní principy komunikace proto zůstávají stejné.

Procesorem řízený přenos

Pro přenos dat z periferního zařízení do hlavní paměti se musí provést dva kroky. Nejprve přenos z periferního zařízení do univerzálního registru procesoru a následně z tohoto registru do hlavní paměti. Oba dva kroky představují samostatnou periferní operaci, která odpovídá instrukci daného procesoru. Vstup-výstupní instrukce jsou prováděny sekvenčně a následující je zahájena až po dokončení předchozí. Nelze tedy využít ani řetězení operací, někdy také nazývané *burst*. Celý přenos vyžaduje účast procesoru a výrazně konzumuje výpočetní výkon.

Přímý přístup do paměti

Základním požadavkem pro efektivní zvládnutí periferních operací je jejich autonomnost. V případě přímého přístupu do paměti, z anglického *direct memory access* zkráceně *DMA*, možnost komunikace bez účasti procesoru.

Konkrétní implementace závisí na použité systémové sběrnici. Pro všechny realizace však platí, že v okamžiku přímého přístupu do systémové paměti je procesor odpojen od sběrnice a může pokračovat ve své činnosti. Nedochází ke spotřebě výpočetního výkonu na přenos dat.

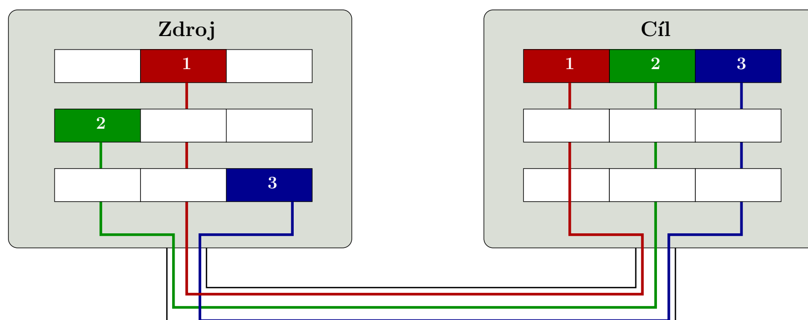
Další nespornou výhodou DMA přenosů je, že se obvykle odehrávají po celých blocích, což vede ke snížení režie přenosů a zvýšení propustnosti sběrnice či celého systému.

2.2.2 Pokročilé varianty DMA

Jelikož je základní funkcí DMA přenos dat, lze zároveň využít k sestavení nesouvislých bloků dat, nebo naopak k rozdělení souvislých dat do nespojitých bloků. Tyto operace se nazývají *gather* a *scatter*.

Operace sestavení

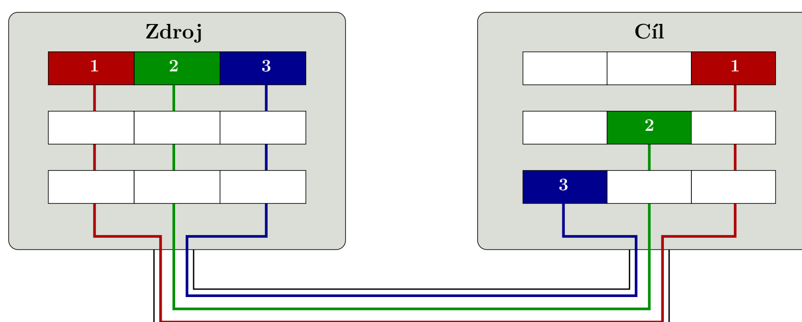
Operace sestavení, nebo-li gather, urychluje práci počítače tak, že odesílatel nemusí napřed data sestavit a následně odeslat, ale postačí provést několik přenosů na navazující adresy. Princip operace sestavení ukazuje obrázek 2.8.



Obrázek 2.8: Princip operace gather

Operace rozdělení

Obdobně operace rozdělení, nebo-li scatter, urychluje práci počítače tak, že odesílatel nemusí napřed data rozdělit a následně odeslat, ale postačí mu provést několik přenosů dat na požadované adresy. Princip operace rozdělení ukazuje obrázek 2.9.



Obrázek 2.9: Princip operace scatter

2.2.3 Řízení periferní operace

První fáze periferní operace, zahájení, se obvykle provádí jako součást vstup-výstupní instrukce procesoru. Může však být iniciována i periferním zařízením reagujícím na externí událost. Například pohyb myši či příchod paketu.

Poslední fází periferní operace je její ukončení a zjištění stavu. Existují dvě techniky: zjištění stavu dotazováním a přerušením.

Dotazování

Opakované dotazování, anglicky *polling*, se provádí tak, že procesor cyklicky čte stavový registr zařízení. Pokud zařízení ukončí rozpracovanou operaci, nastaví příslušné hodnoty do tohoto registru a procesor si informaci vyzvedne.

Velkou nevýhodou je neustálý režijní provoz na sběrnici a také čerpání strojových cyklů procesoru.

Někdy naopak lze výhodně použít pro zjišťování stavu periferního zařízení. Především u sběrnic s přerušením spouštěným úrovní (viz dále). Postupuje se pak tak, že se přerušení generovaná zařízením zakážou a dotazování je prováděno v pravidelných časových intervalech určovaných časovačem. Redukuje se režie spojená s obsluhou přerušení. Vysvětlení této problematiky lze nalézt v následujícím textu.

Přerušení

Požadavek autonomnosti periferní operace a existence spousty různě rychlých periferních zařízení dal vzniknout technice asynchronního oznámení o dokončení operace. Přerušení, anglicky *interrupt*, lze realizovat několika způsoby a je závislé na použité systémové sběrnici.

Po zahájení periferní operace je tato kompletně v režii řadiče periferního zařízení. Po dokončení řadič asynchronně vyšle signál přerušení směrem k procesoru. Z uvedeného vyplývá, že po celou dobu provádění operace může procesor pokračovat ve své činnosti. Mechanizmus přerušení je výhodný zejména u pomalých zařízení či zařízení reagujících na externí události.

V praxi se používají následující implementace:

- *Přerušení spouštěná hranou.* Pro každé zařízení existuje na systémové sběrnici oddělený vodič, přes který zařízení signalizuje žádost o přerušení. Obsluha přerušení je spouštěna s nástupnou případně sestupnou hranou signálu. Díky jednoznačnosti vodiče náležícího zařízení je okamžitě znám i vektor přerušení, který je vždy právě jeden a unikátní pro zařízení a ke každému zařízení tak existuje právě jedna obslužná procedura. Obsluha přerušení hranou je velmi rychlá. Není třeba zjišťovat žádné další informace a přerušovací signál je veden po samostatných vodičích systémové sběrnice. To také omezuje maximální množství připojitelných zařízení a umožňuje přerušení předběhnout data putující po sběrnici. Implementováno ve sběrnici ISA.

- *Přerušení spouštěná úrovní.* Všechna zařízení sdílejí jeden nebo několik málo vodičů systémové sběrnice určených pro signalizaci přerušení. Zařízení, které signalizuje přerušení, uzemní takovýto vodič. Tím je vyvoláno přerušení. Všechna zařízení připojená ke stejnému vodiči sdílí přerušovací vektor a tedy i přerušovací obsluhu. Prvním krokem obsluhy přerušení pak musí být zjištění zdroje, tj. signalizujícího zařízení. To se provede například kontrolou stavových registrů. Počet připojitelných zařízení není omezen, ale obsluha je výrazně pomalejší než v předchozím případě. Tento mechanismus je implementován ve sběrnici PCI.

- *MSI.* Zprávou signalizované přerušení (z angl. *message signaled interrupt*) je nejmodernější způsob žádosti o přerušení využívaný u sběrnic podporujících zasílání zpráv. V závislosti na formátu zprávy dokáže jedno zařízení zaslat několik různých vektorů přerušení. V okamžiku přijetí zprávy tedy obslužná rutina ví, které zařízení bylo zdrojem signálu, navíc může zpráva obsahovat doplňující informace a každé zařízení,

pak může jednoznačně signalizovat různé události bez potřeby dodatečného zjišťování typu události.

Odpadá tak problém s omezeným počtem připojitelných zařízení a také nutností vyhledávat konkrétní zdroj přerušení. Nevýhodou je šíření zprávy stejnou cestou jako data, které způsobí delší časovou odezvu, ale na druhou stranu zamezí rozpadu synchronizace mezi přerušením a komunikací probíhající po sběrnici.

Takzvané předbíhání přerušení po oddělených vodičích vytváří problém zejména při použití DMA přenosů, kdy typicky po obdržení přerušení musí následovat synchronizační čtení, které zaručí dokončení rozpracovaných přenosů. Signalizaci přerušení zasíláním zpráv implementuje sběrnice PCI-Express, ale také PCI v revizi 2.2, viz [11] a [12].

2.2.4 Typy transakcí

V předchozím textu byly popsány způsoby řízení periferní operace. Nyní budou vysvětleny jednotlivé atomické kroky prováděné během periferní operace. Těmi jsou čtení a zápis jakožto instrukce procesoru. Existují dva typy paměťových operací podle způsobu jejich provedení. Překlad názvů obou operací není jednoznačný, a jsou proto použity anglické ekvivalenty.

Posted

Typ paměťové transakce, kdy iniciátor neočekává potvrzení o provedení operace. Z pohledu procesoru to znamená významný výkonnostní přínos. Pokud například lze zápis provést jako *posted* transakci, může procesor po provedení instrukce okamžitě pokračovat v činnosti. Nastává například při zápisu do paměťově mapovaného vstup-výstupu. Výsledkem mohou být dávkové, nebo taky *burst* přenosy.

Non-posted

Tento typ paměťové transakce by se dal označit za transakci s potvrzením. Iniciátor zahájí operaci a následně očekává informaci o jejím dokončení. Informace o dokončení se často označuje *completition* transakce. Na straně procesoru nastává problém, je-li nutné čekat na dokončení operace před dalším postupem v provádění programu. To v důsledku vede na delší dobu provádění. Non-posted transakce se používají při přístupu k vstupně-výstupním portům.

2.3 Ovladače zařízení

Ovladač zařízení je nedílná součást jádra operačního systému. Zprostředkovává zbytku jádra a uživatelským procesům nízkoúrovňový přístup k funkcím periferního zařízení skrze abstraktní rozhraní. Tato rozhraní jsou unifikována tak, aby v rámci operačního systému existovalo omezené množství typů zařízení, kde se zařízeními stejného typu lze pracovat identickým způsobem.

Pro implementaci softwarové části práce byl vybrán volně dostupný a dobře dokumentovaný operační systém (někdy jen OS) Linux. Proto budou v následujícím textu uvažovány ovladače a jejich implementace jen v tomto operačním systému.

2.3.1 Ovladače zařízení v Linuxu

Jádro OS Linux rozlišuje tři druhy zařízení a jim odpovídajících ovladačů. Liší se poskytnutým rozhraním a základní datovou jednotkou, kterou jsou schopny zpracovat. Tvorbu ovladačů pro OS Linux popisuje [5].

- Znaková zařízení. Do této skupiny patří zařízení produkující respektive zpracovávající proud dat v čase po jednotlivých bajtech. Příkladem může být myš, klávesnice nebo také zvuková karta.
- Bloková zařízení. Bloková zařízení komunikují po celých blocích dat. Blok dat je nejmenší datovou jednotkou, kterou jsou schopny zpracovat. Mezi nejvýznamnější zařízení této kategorie patří pevný disk a hlavní paměť.
- Síťová zařízení. Ty tvoří výjimku oproti předchozím dvěma typům, neboť je v operačním systému nereprezentuje žádný soubor. Naopak jsou vzájemně rozlišeny jednoznačným identifikátorem rozhraní, například *eth0*.

  Z uživatelského pohledu se síťová komunikace jeví jako spojitý proud dat přijímaný či odesílaný skrze *socket*. Tento proud dat však jádro zpracuje a rozdělí do paketů. Z pohledu ovladače síťového zařízení je proto základní datovou jednotkou komunikace paket.

  socket je označení koncového bodu obousměrného spojení mezi dvěma procesy. Pro práci se sockety existuje rozhraní pro programování aplikací umožňující komunikaci mezi procesy.

Zároveň je důležité si uvědomit, že rozdělení zařízení do uvedených skupin není dogmaticky dáno. Často lze vytvářet adaptéry rozhraní umožňující pracovat s diskem jako znakovým zařízením nebo obdobně se síťovou kartou. Opačným přístup, tj. převod znakového zařízení na síťové, není běžný a z principu dokonce nevhodný.

Roli ovladače v operačním systému přibližuje obrázek 2.10.

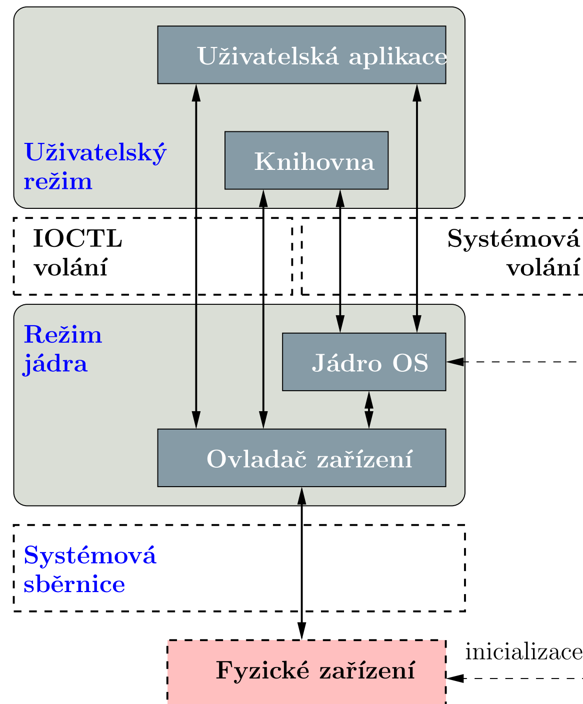
Modulární vs. monolitické jádro Sestavením všech částí operačního systému včetně ovladačů zařízení do jednoho nedělitelného programu vzniká uspořádání zvané monolitické jádro, kde jsou všechny poskytované služby těsně svázány a běží ve stejné oblasti paměti. Lze tedy bez omezení a efektivně přistupovat k hardware.

Pokud bude třeba přidat do monolitického jádra novou funkcionalitu, například ovladač nového zařízení, je nezbytné celé jádro znovu sestavit a po restartování počítače jádro zavést.

Poněkud složitější řešení z pohledu návrhu celého systému představuje modulární jádro. To umožňuje jednotlivé ovladače sestavit jako *moduly*, odtud i název, a podle potřeby je do jádra zavádět bez nutnosti restartování systému. Modulární architekturu adaptoval i OS Linux.

2.3.2 Správa paměti

Linux patří mezi operační systémy s virtuálním paměťovým systémem. To znamená, že adresy používané v uživatelském prostoru nejsou rovny fyzickým adresám. Navíc virtualizace paměti umožňuje oddělení adresových prostorů jednotlivých procesů a je základním předpokladem pro realizaci moderního operačního systému poskytujícího ochranu paměti.



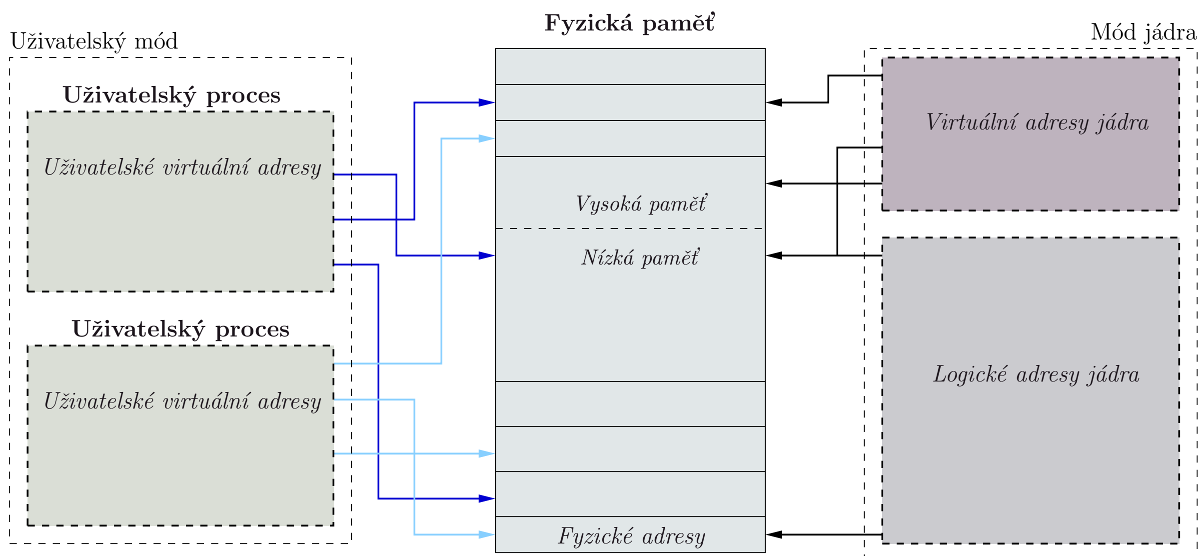
Obrázek 2.10: Role ovladače v OS Linux

Taktéž je možné s použitím virtualizace paměti zpřístupnit uživatelskému procesu fyzickou paměť zařízení.

Linux rozlišuje následující typy adres:

- Uživatelská virtuální adresa. Jedná se o adresy, ke kterým přistupují uživatelské procesy.
- Fyzická adresa. Fyzická adresa slouží pro přístup k hlavní paměti ze strany procesoru.
- Sběrnicová adresa. Používá se mezi periferními zařízeními a hlavní pamětí. Často odpovídá fyzické adrese používané procesorem. Toto je ovšem závislé na konkrétní architektuře.
- Logická adresa jádra. Tyto adresy popisují část hlavní paměti a představují běžný adresový prostor jádra. Na většině architektur přímo odpovídají fyzickým adresám. Pokud je velikost hlavní paměti celá adresovatelná ze strany procesoru, odpovídá virtuální adresový prostor jádra fyzickým adresám.
- Virtuální adresa jádra. Představuje mapování mezi fyzickou adresou a adresou prostoru jádra. Přičemž platí, že každá logická adresa jádra je virtuální adresou jádra, ale naopak ne. Slouží k mapování fyzické paměti, která není dostupná přes logické adresy jádra.

Vztah uvedených typů adres ukazuje obrázek 2.11. V obrázku je hlavní paměť rozdělena na vysokou a nízkou. Význam toho rozdělení je, že nízká paměť odpovídá logickému adresovému prostoru jádra, naopak vysoká paměť je ta část virtuálního adresového prostoru jádra, pro které neexistuje logická adresa jádra.



Obrázek 2.11: Typy adres v Linuxu a jejich vztah

Pro realizaci DMA přenosů jsou důležité sběrnicové adresy, neboť vystavované požadavky na přímý přístup do paměti musí používat právě je. Při komunikaci s periferním zařízením tak musí ovladač převádět virtuální adresy jádra na adresy sběrnicové. Pro tyto převody existuje sada pomůcek v podobě maker definovaných ve zdrojových kódech jádra.

Paměť pro přímý přístup

Podle způsobu využití paměti pro přímý přístup lze rozlišit dvě varianty mapování:

- *Koherentní*, nebo také synchronní či konzistentní. Paměť se mapuje do oblasti koherence s vyrovnávací pamětí. Do takové oblasti je možný souběžný přístup jak z periferního zařízení, tak z procesoru. Koherenci zajišťuje hardware transparentně.
- *Proudové*, nebo také asynchronní či nekonzistentní. Na rozdíl od předchozího, není paměť v oblasti koherence s vyrovnávací pamětí. Pro korektní přístup k proudově mapované paměti musí být provedena explicitní synchronizace jejího obsahu s vyrovnávací pamětí.

Podrobnější popis lze nalézt v [1] v souboru *Documentation/DMA-mapping.txt* nebo také v [5] a v [6].

2.3.3 Obsluha přerušení

Z pohledu operačního systému je přerušení událost, která změní sekvenci prováděných instrukcí. Jak už bylo uvedeno v části 2.2.3, může to být následkem nastavení hodnoty příslušného vodiče či zasláním zprávy. Moderní hardware je vybaven programovatelnými řadiči přerušení, které koncentrují potenciální zdroje a předávají signály procesoru, či více procesorům u multiprocesorového systému, jednotným způsobem.

Linux rozlišuje dva typy přerušení podle způsobu vyvolání respektive jejich zdroje:

- *Synchronní.* Požadavek na přerušení je vyvolán na základě prováděné instrukce. Je tedy předvídatelný okamžik příchodu a lze jednoznačně určit, který proces obsahuje danou instrukci, a proto i obsluha může být provedena v rámci času vyhrazeného pro proces. Synchronní přerušení jsou označována jako *výjimky*. Obsluha výjimek obvykle spočívá v zaslání příslušného signálu aktuálnímu procesu, např. *SIGSEGV*. Výjimky se také používají pro ladění programů.

- *Asynchronní.* Požadavek na přerušení je vyvolán obvykle externím zdrojem jedním ze způsobů popsaných v části 2.2.3. Předem nelze určit okamžik příchodu signálu a není tedy ani jasné, komu by měl být zaslán případný signál. Obsluha ve výchozím stavu spočívá v zaznamenání neobsloužené, dokonce i neočekávané, události.

  Každé zařízení schopné generovat přerušení by nemělo signalizovat události do doby, než je pro něj zaveden odpovídající ovladač, který zaregistruje obslužnou proceduru v datových strukturách jádra. Teprve po zavedení ovladače by tento měl povolit signalizaci událostí a každou příchozí korektně obsloužit.

Vzhledem k tomu, že synchronní události, výjimky, způsobí pouze odeslání signálu příslušnému procesu, není zpracování výjimky časově kritické. Zbylé výpočty se provedou v době provádění daného procesu. V kontextu uživatelského procesu.

Obdobně je snahou při zpracování asynchronní události, přerušení, aby obsluha netrvala dlouho dobu, neboť výrazně ovlivňuje latenci celého systému. Obsluha typicky spočívá v určení zdroje přerušení, zjištění nových informací, tj. proč událost nastala a následném návratu do místa původního běhu. Zpracování nových informací se potom provádí mimo kontext přerušení a nezhoršuje odezvu systému.

Nechť jako příklad poslouží příjem paketů, pak samotná obsluha přerušení pouze poznamená množství nově přijatých dat. Ve vhodném okamžiku se vyvolá prvotní zpracování dat, například nastavení časových značek či ověření kontrolního součtu, a data se předají uživatelským procesům, které provedou výpočetně nejintenzivnější část zpracování dat.

Problematickou částí tohoto přístupu se stává určení zdroje přerušení. V případě hranou spouštěného nebo zprávou signalizovaného přerušení nevzniká komplikace. Naopak u úrovní spouštěného přerušení, jak již bylo zmíněno, je třeba projít všechna zařízení připojená k danému signálu a přesně určit zdroj události. V terminologii linuxového jádra se používá označení *sdílení požadavku přerušení* (z angl. *interrupt request sharring*). Tento mechanizmus v praxi znamená upozornit všechny ovladače zařízení sdílejících jeden požadavek o příchodu signálu. Každý ovladač pak prověří, jestli jemu příslušející zařízení nevyvolalo událost. Typicky přečtením stavového registru. Se zvyšujícím se počtem zařízení sdílejících přerušovací vektor roste počet upozorněných ovladačů, a tím i režie obsluhy.

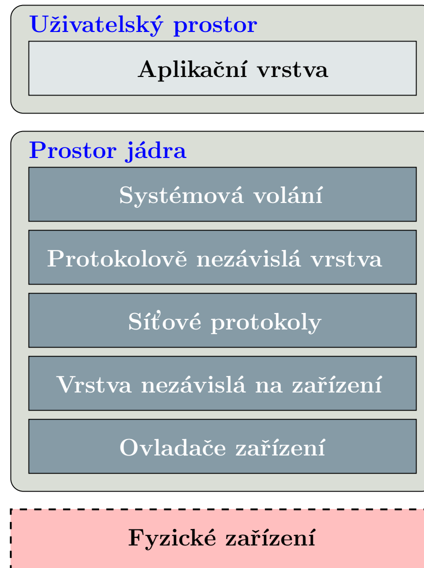
Je tedy snaha redukovat množství vznikajících přerušení, především, ale ne pouze, u zařízení signalizujících přerušení úrovní. Jak je tomu u sběrnice PCI. Vhodným prostředkem se jeví rozhraní NAPI, viz dále.

2.3.4 Síťové ovladače

Jedním z důvodů volby Linuxu jako cílového operačního systému je orientace na práci v počítačových sítí. Od samého počátku obsahuje systém vestavěnou podporu pro síťovou komunikaci. Za dobu své existence také posloužil mnoha výzkumným kolektivům z oblasti počítačových sítí. K dispozici tak je celá řada studií, analýz a dokumentačních zdrojů popisujících práci s pakety uvnitř jádra Linuxu.

Anatomie síťového zásobníku

Stejně jako je problém síťové komunikace rozdělen modelem ISO/OSI (viz část 2.1), je i problém zpracování paketů uvnitř linuxového jádra rozdělen do několika vrstev. Toto rozdělení ukazuje obrázek 2.12.



Obrázek 2.12: Anatomie síťového zásobníku

Uživatelská aplikace v průběhu své činnosti využívá vrstvy systémových volání pro příjem či odesílání dat. Proto existují systémová volání *read* a *write*. Pokud by síťové zařízení poskytovalo pouze tyto dvě služby, bylo by možné přistupovat k síťovým kartám stejně jako k jiným znakovým zařízením.

Při práci se síťovým rozhraním je však třeba provádět řadu operací, které jsou výlučně spojeny s komunikací v počítačových sítí a to bez ohledu na konkrétní použitou kartu. Například aktivování či deaktivování síťového rozhraní, nastavení IP adresy, zjištění hardwarové adresy nebo také vyčítání statistik. Řadu uvedených operací obsluhuje samotný ovladač zařízení, naopak jiné jsou realizovány vyššími vrstvami síťového zásobníku.

Společným prvkem všech vrstev prostoru jádra je datová struktura sloužící pro reprezentaci paketu. Struktura se nazývá `sk_buff`, přičemž základní operace, které musí ovladač vykonat s touto strukturou jsou následující:

- Odeslání paketu.

  1. Převzetí paketu od jádra v okamžiku vyvolání funkce pro odeslání paketu a předání do zařízení. Od jádra je paket přijat v podobě zmíněné struktury `sk_buff`. Předání paketu do zařízení je implementačně závislé, avšak ovladač v obvyklých případech nemodifikuje obsah paketu.
  2. Uvolnění struktury `sk_buff` po úspěšném odeslání paketu. Jednou ze základních zodpovědností ovladače je zajistit, aby každý mu svěřený paket byl korektně odeslán po síti. Ovladač tedy musí udržovat vazbu na paket do té doby, než je si jist, že byl paket odeslán do sítě. O odeslání paketu je typicky informován

přerušením (viz část 2.2.3), po jehož přijetí může strukturu uvolnit k dalšímu použití.

- Příjem paketu.
  1. Vytvoření struktury `sk_buff` pro příjem paketu. Pro každý přijatý paket je nezbytné vyhradit v hlavní paměti místo, kde bude později uložen. Způsobů, kdy vytvářet a jak předávat instance struktury `sk_buff` do zařízení, je několik. Dvě varianty jsou popsány v části 3.3.4.
  2. Předání přijatého paketu jádru. Obdobně jako u odeslání, je naplnění struktury `sk_buff` přijatým paketem implementačně závislé a stejně tak platí, že samotný ovladač nemodifikuje obsah paketu. Okamžik, kdy je paket připraven v hlavní paměti pro další zpracování, se typicky signalizuje přerušením (viz část 2.2.3).

Samotná struktura `sk_buff` obsahuje rozsáhlé množství položek, které nesou informace o paketu. Nechybí délka dat, čas přijetí, ukazatele na hlavičky protokolů jednotlivých vrstev (viz část 2.1), ukazatel na předchozí/následující paket a další (viz [1] soubor *include/linux/skbuff.h*).

Zrychlené odesílání

Pro pochopení popisované techniky je důležité si uvědomit, jak vzniká odesílaný paket. Na počátku vzniku paketu stojí aplikace, která si přeje komunikovat prostřednictvím sítě. Ta vytvoří blok dat, který bude odeslán, a předá jej standardnímu socketovému rozhraní. Socketové rozhraní vytváří abstrakci na úrovni transportního protokolu, jenž přenáší data mezi dvěma konkrétními aplikacemi - procesy běžícími na vzdálených strojích. Aplikaci identifikuje číslo portu, stroj IP adresa (v případě rodiny protokolů TCP/IP).

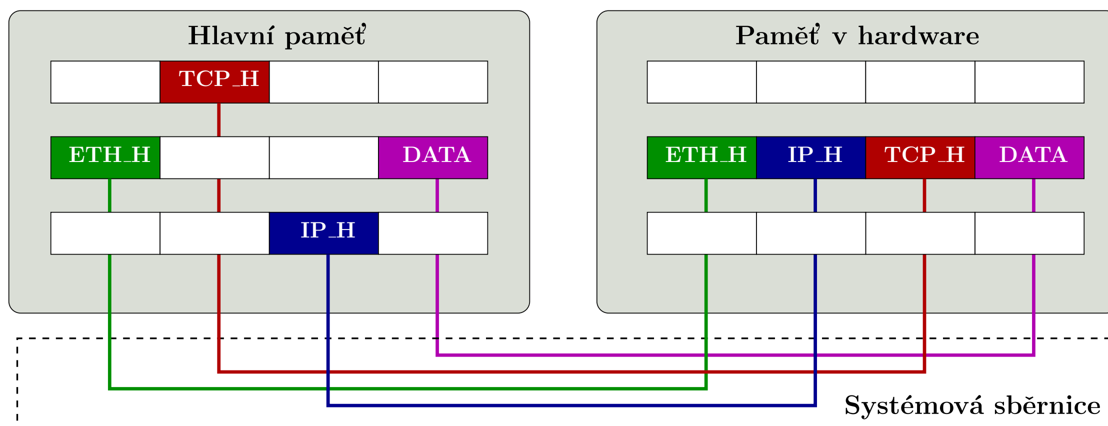
Transportní vrstva rozdělí spojitý proud dat na pakety, označí hlavičkou a předá nižší vrstvě, síťové. Ta opět přidá svou hlavičku. Vzniká tedy postupně blok dat, na jehož začátek jsou přidávána data. Efektivní přístup k realizaci popsaného postupu by byl vyhradit před daty dostatečný úsek paměti pro pozdější vložení hlaviček. Nemusí však být dopředu jasné kolik místa bude potřeba. Druhá možnost je neustále data posouvat, což vede k plýtvání výpočetním časem. Třetí možnost je ponechat data odděleně. Avšak při odeslání na fyzické vedení musí být blok dat spojitý.

Pro sestavení nespojitého bloku dat byl představen prostředek zvaný DMA přenos typu gather (viz část 2.2.2). Ten samozřejmě vyžaduje podporu, jak ze strany ovladače zařízení, tak samotného hardwarového zařízení. Jednotlivé části paketu, postupně přidávané hlavičky, tedy vznikají odděleně a není třeba je přesouvat. K jejich spojení do souvislého bloku dat dojde až v okamžiku přenosu do hardware. Situaci zachycuje obrázek 2.13.

Rozhraní NAPI

V době rozšíření víceprocesorových systémů byla provedena úprava stávající implementace síťového zásobníku uvnitř linuxového jádra. Nová implementace podporující symetrický multiprocesing, zvaná *softnet*, se však po studii provedené společností *Mindcraft* ukázala jako nedostatečná. Po rozboru [13] slabin této implementace, navrhli autoři nové rozhraní označované *New API*, zkráceně NAPI.

NAPI odstraňuje problémy zjištěné u předchozí implementace. Především přeuspořádání paketů při souběžném použití více procesorů, což u protokolu TCP



Obrázek 2.13: Sestavení paketu operací gather

způsobuje žádosti o znovu zaslání paketů vedoucí k degradaci přenosové kapacity. Pak také řeší snížení počtu přerušení vyvolaných během vysoké zátěže a předchází zbytečnému zpracování následně zahozených paketů.

Velikost paketu v B	Počet přijatých paketů za 1s	Počet signalizovaných přerušení
60	890 000	17
128	758 150	21
256	445 632	42
512	232 666	241 292
1024	119 061	872 519

Tabulka 2.1: Snížení počtu přerušení využitím NAPI

Z provedených měření [2], jejichž výsledky ukazuje tabulka 2.1, lze vyčíst výrazné snížení počtu generovaných přerušení vedoucího k snížení zátěže procesoru při příjmu krátkých paketů. Naopak pro dlouhé pakety se počet vyvolaných přerušení blíží počtu přijatých paketů.

Uvedené rozhraní je podle naměřených hodnot vhodné pro vysokorychlostní příjem paketů, a proto bude použito pro implementaci ovladače.

Kapitola 3

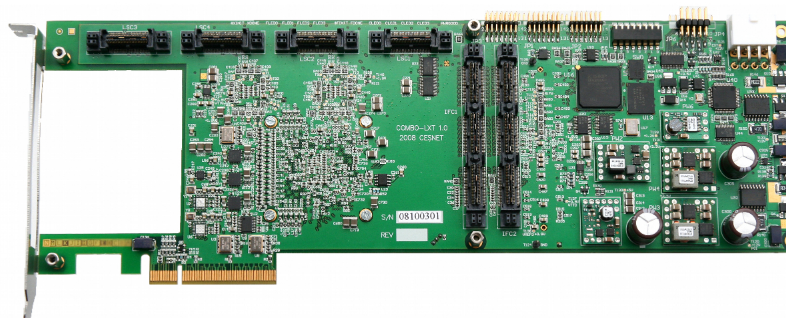
Popis cílové platformy

Aby bylo možné navrhnout efektivní řešení, je nezbytná detailní znalost prostředí, ve kterém budou ovladač a DMA řadiče implementovány a následně používány.

3.1 Karta COMBOv2

Návrh dma řadičů musí být nezávislý na konkrétní použité kartě, ale je třeba znát alespoň rozhraní a špičkovou rychlost jakou karta komunikuje se zbytkem počítače. Pro implementaci prototypu poslouží karta COMBOv2, proto zde budou uvedeny její základní technické parametry. Nezávislost implementace na konkrétní kartě zajistí platforma NetCOPE, viz část 3.2.

Jedná se o kombinaci nejnovější verze mateřské karty s rozšiřující kartou rozhraní vyvinutých v rámci výzkumného projektu *Liberouter*. Mateřskou kartu spojuje se zbytkem počítače sběrnice *PCI-Express x8*. Nese označení *COMBO-LXT*, na obrázku 3.1, a je osazena programovatelným hradlovým polem rodiny *Virtex 5* vyráběným firmou *Xilinx*.



Obrázek 3.1: Ilustrační snímek karty *COMBO-LXT*

Rozšiřující karty rozhraní slouží k připojení různých druhů vedení počítačových sítí. Ke kartám lze připojit jak metalická, tak optická vedení. V případě optického vedení se jedná o jednovidová, mnohavidová vlákna či vlákna s přenosem metodou *CWDM* (viz například [4]). U metalických vedení lze použít rychlostí 1 Gb/s a 10 Gb/s. Rozšiřující karty se liší počtem fyzických rozhraní a rychlostí, na kterých můžou tato rozhraní pracovat. V současnosti existují dvě varianty :

- COMBOI-1G4. Obsahuje čtyři rozhraní pracující až na rychlosti 1 Gb/s.

- COMBOI-10G2. Obsahuje dvě rozhraní pracující až na rychlosti 10 Gb/s.

3.1.1 Sběrnice PCI-Express

Duplexní sběrnice se stromovou topologií realizující spojení typu bod-bod. Pro přenos používá paketovou komunikaci. U paměťových transakcí podporuje jak 32bitovou, tak 64bitovou velikost adresy. Ve verzi 1.0 přenáší data špičkovou rychlostí 2,5 Gb/s v každém směru jedním spojem. Celková propustnost karty COMBOv2 tak v jednom směru činí $8 * 2,5 = 20$ Gb/s.

Identifikace PCI zařízení

Stejně jako PCI a PCI-X používá sběrnice PCI-Express pro identifikaci zařízení registry ve svém konfiguračním prostoru. To umožňuje operačnímu systému a ovladačům rozpoznat, zda dokáží dané zařízení obsluhovat. Z pohledu identifikace jsou nejdůležitější registry *Vendor ID*, *Device ID*, *Class Code*, *Subsystem Vendor ID* a *Subsystem Device ID.* Jako praktický příklad následuje identifikace karty COMBOv2:

Vendor ID = 0x18EC (CESNET)
Device ID = 0xC032 (COMBOv2)
Subsystem Vendor ID = 0x18EC
Subsystem Device ID = 0x0100

Signalizace přerušení

Podle revize 1.1 specifikace sběrnice (viz [12]) PCI-Express podporuje jak signalizaci přerušení úrovní, zavedenou ve sběrnici PCI, tak signalizaci přerušení zprávou. A to ve variantách MSI a MSI-X.

Vzhledem k tomu, že sběrnice PCI-Express neobsahuje žádné vyhrazené vodiče pro signalizaci přerušení, je signalizace úrovní emulována prostřednictvím dvojice vyhrazených zpráv. A to nastavení a nulování příslušného přerušení. Pro dodržení zpětné kompatibility, pak musí hardware obsahovat i stejnou sadu konfiguračních registrů.

Signalizace použitím MSI umožňuje generovat až 32 (viz [1] soubor *Documentation/PCI/MSI-HOWTO.txt*) různých přerušovacích vektorů. MSI-X až 2048. Jak již bylo uvedeno v části 2.2.3, zprávy putují stejnou cestou jako data a nemohou se předbíhat, což zaručuje synchronizaci transakcí provedených před vyvoláním přerušení. Navíc není třeba dále zjišťovat zdroj události.

3.2 Platforma NetCOPE

Základním cílem platformy NetCOPE je vytvoření konfigurovatelné vrstvy, která usnadní a urychlí vývoj síťových aplikací akcelerovaných pomocí technologie FPGA. Platforma vytváří abstrakci nad kartou a odstiňuje tak nízkoúrovňové hardwarově závislé vlastnosti. Navíc poskytuje snadný přístup ke zdrojům karty.

Určujícím z hlediska návrhu jsou rozhraní, přes která budou DMA řadiče přijímat informace o přijatých respektive odeslaných paketech, vystavovat požadavky na samotné DMA přenosy a také sledovat stav dokončení vyvolaných přenosů. Z tohoto pohledu lze identifikovat dvě rozhraní:

- rozhraní pro komunikaci po systémové sběrnici a
- rozhraní pro signalizaci přijatých či odeslaných paketů.

Názorněji vše ukazuje obrázek 3.2. Jednotlivé stavební bloky mají následující význam:

- Blok DMA přenosů. Obsahuje samotné řadiče DMA a paměť, ve které jsou pakety uloženy před jejich odesláním do sítě, respektive po jejich přijetí ze sítě před přesunem do hlavní paměti. Tato paměť je složena z oddělených bloků vyhrazených pro jednotlivá fyzická rozhraní a označena *DMA buffery*. Pakety jsou z/do hlavní paměti kopírovány prostřednictvím sběrnice *PCI-Express*, jejíž připojení v rámci čipu FPGA zprostředkovává interní sběrnice, anglicky *Internal Bus*.

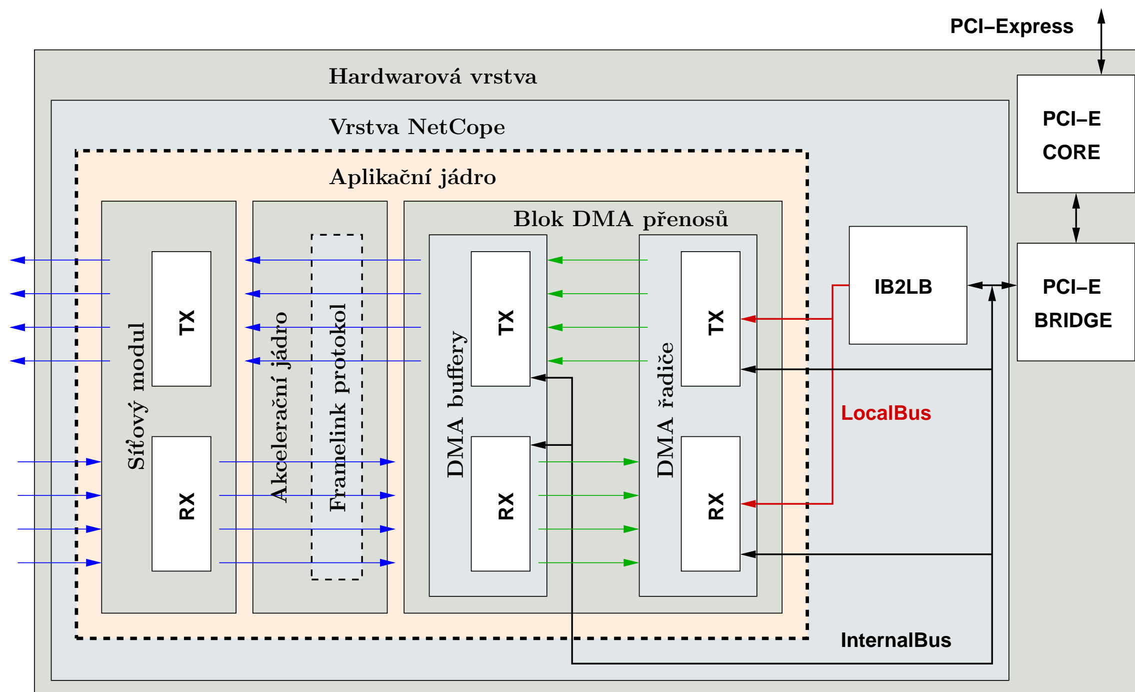
  Navrhované DMA řadiče zastupuje stejnojmenný blok.
- Framelink protokol. Označuje protokol použitý pro výměnu dat mezi DMA buffery a síťovým modulem. Je inspirován protokolem *LocalLink* společnosti *Xilinx* a pro přenos dat používá *framelinkové* pakety.
- Síťový modul obsahuje obvody zpracovávající pakety na spojové vrstvě. Jedná se o proprietární implementaci společnosti *Xilinx*.
- Blok *IB2LB* je podstatnou součástí vrstvy NetCOPE. Slouží jako most pro připojení jednoduché lokální sběrnice, označené *Local Bus*, k interní sběrnici. Lokální sběrnice dosahuje nižší propustnosti a má také menší datovou šířku než interní sběrnice, typicky 8 nebo 16 bitů. Na druhou stranu přináší úsporu v množství logiky potřebné k jejímu zpracování. Čehož můžeme využít k připojení registrů, které jsou zapisovány nebo čteny jen zřídka.
- Bloky *PCI-E core* a *PCI-E bridge* jsou hardwarově závislé na konkrétním typu karty a zpřístupňují systémovou sběrnici obvodům uvnitř čipu FPGA prostřednictvím interní sběrnice.

3.2.1 Připojení k DMA bufferům

Na obrázku 3.2 je toto rozhraní znázorněno šipkami mezi DMA buffery a DMA řadiči. Samotné pakety putují po interní sběrnici. Je však nezbytné, aby se DMA řadič dozvěděl o příchodu nového paketu ještě před tím, než vydá povel k jeho přesunu do hlavní paměti. Analogicky u vysílání paketů, DMA řadič musí informovat konkrétní DMA buffer, že má připravena data k odeslání do sítě.

3.2.2 Připojení k systémové sběrnici

Prostřednictvím interní a lokální sběrnice přistupuje DMA řadič ke respektive je přístupný ze systémové sběrnice. Na obrázku 3.2 označeno šipkami vedoucími z vrstvy NetCOPE do bloku DMA přenosů. Pomocí protokolu interní sběrnice pak vydává jednotlivé povely k provedení DMA přenosů. Lokální sběrnice slouží k inicializaci či zastavení činnosti řadičů.



Obrázek 3.2: Blokové schéma síťové karty na platformě NetCOPE

Interní sběrnice

Jedná se o duplexní spojení stromové topologie o šířce 64bitů. Plně duplexní schopnost komunikace u interní sběrnice zaručují dva oddělené kanály označené

- *uplink*, pro přenosy směrem z FPGA čipu ke systémové sběrnici, a
- *downlink*, pro opačný směr.

Příklad typického zapojení ukazuje obrázek 3.3.

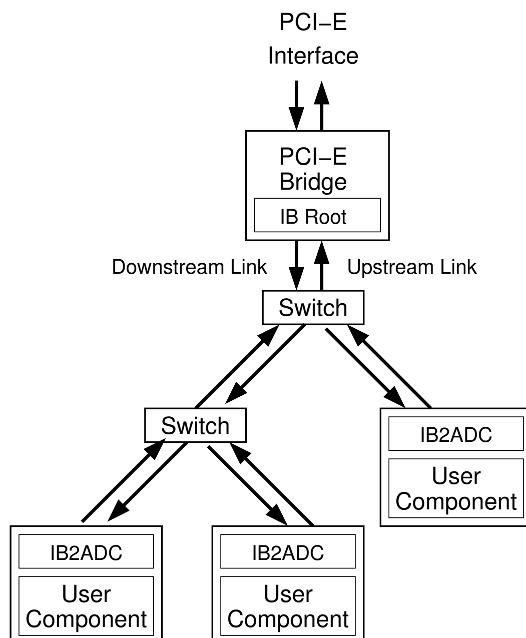
Pro řízení provozu na interní sběrnici je nezbytné znát použitý komunikační protokol. Protože se jedná o sběrnici přenášející data v podobě paketů, je činnost řízena položkami v hlavičce paketu. Pro jednoznačné rozlišení jsou pakety interní sběrnice označeny jako *IB pakety*. Formát IB paketu lze vidět na obrázku 3.4.

Velikost hlavičky je vždy 128 bitů. Význam jednotlivých polí pak závisí na typu transakce. Fixní pozici a význam mají položky:

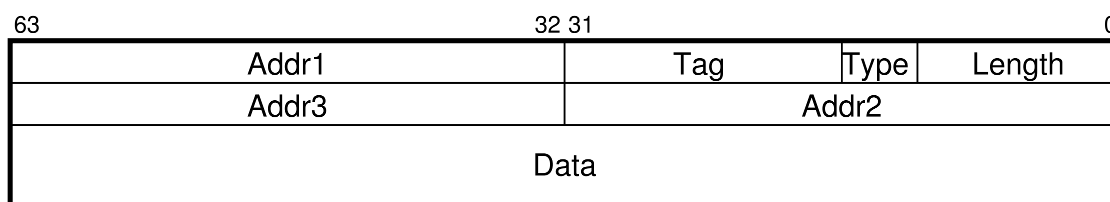
- *Length*. 12 bitů udávajících délku prováděné transakce.
- *Type*. 4 bity rozlišující typ prováděné transakce.
- *Tag*. 16 bitů identifikujících transakci, ke které paket náleží.

Zbylá pole obsahují adresu zdroje a cíle, případně jsou nevyužita. Detailnější popis IB paketů lze nalézt v [9].

V rámci jedné transakce může být poslán jeden nebo dva IB pakety. Příkladem dvou paketů na jednu transakci jsou *non-posted* (viz 2.2.4) transakce, které nastávají při přenosu



Obrázek 3.3: Typické zapojení interní sběrnice



Obrázek 3.4: Formát paketu interní sběrnice

dat z hlavní paměti do prostoru karty iniciovaného DMA řadičem a vykonávaného PCI-E bridgem, tzv. *global to local read*. Nejprve DMA řadič odešle požadavek v podobě paketu neobsahujícího žádná data, ale položkou *length* nastavenou na množství čtených dat. PCI-E bridge pak vykoná čtecí operaci na systémové sběrnici a na downlink interní sběrnice vloží paket se stejnou položkou *length*, *tag*, obsahující požadovaná data a s typem transakce nastaveným na *read completition*.

Příkladem transakcí sestávajících z pouze jednoho paketu jsou zápis do hlavní paměti nebo také zápis ze strany procesoru.

DMA požadavek

Požadavek na přímý přístup do paměti se na interní sběrnici předává prostřednictvím *bus master* rozhraní koncové komponenty a má následující položky:

1. Globální adresa. Adresa je chápána z pohledu DMA řadiče. Označuje sběrnicovou adresu místa v operační paměti počítače. Velikost adresy je 64 bitů.

2. Lokální adresa. Stejně jako globální adresa je chápána z pohledu DMA řadiče.

Označuje adresové místo v prostoru karty. Jinak také adresa v rámci čipu FPGA o velikosti 32 bitů.

3. Značka. Anglicky *tag*. Jelikož interní sběrnice podporuje *non-posted* transakce, je nezbytné transakce navzájem odlišit tak, aby při jejich dokončení a odeslání *completition* transakce bylo možné identifikovat původní zdroj/cíl. Pro tento účel každý DMA požadavek obsahuje značku o velikosti 16 bitů.
4. Délka. Určuje počet bajtů, které mají být přesunuty při přenosu. Velikost položky délka je 12 bitů a omezuje maximální délku přenosu na 4 kB.
5. Typ transakce. Udává jakým směrem bude přenos proveden. Zda-li ze zařízení do hlavní paměti *FPGA2RAM*, či naopak *RAM2FPGA*. Položka má velikost 2 bity.

Jak je možné si všimnout, odpovídá DMA požadavek velikostí svých položek paketu interní sběrnice na obrázku 3.4.

3.3 Rozhraní linuxového jádra

Linuxové jádro představuje vysoce paralelní, událostmi řízený program. Jednotlivé události putují mezi částmi jádra prostřednictvím definovaných rozhraní a stejné tvrzení platí i v případě zpracování paketů. Samotný paket se dá rovněž chápat jako příchozí/odchozí událost.

3.3.1 Životní cyklus modulu

Pro názornější popis událostí nastávajících v průběhu životního cyklu modulu obsahuje příloha A sekvenční diagram jazyka *UML*. V diagramu je znázorněna časová návaznost níže uvedených funkcí.

První událost v průběhu života ovladače vytvořeného jako modul nastává v okamžiku zavedení do jádra. Opačnou událostí je pak odstranění modulu. K obsluze slouží funkce `module_init` a `module_exit`. V okamžiku zavedení se modul zaregistruje v rámci jádra operačního systému a oznámí, jaké služby či funkce poskytuje. Naopak v době odstranění zruší veškeré záznamy a vazby.

3.3.2 Ovladače PCI zařízení

Protože je vytvářen ovladač síťové karty připojené ke sběrnici PCI, konkrétně PCI-Express, nicméně na všechny varianty sběrnice PCI je v jádře pohlíženo stejně, bude popsán i způsob obsluhy PCI zařízení v linuxovém jádře.

Nejprve se musí spárovat zařízení s odpovídajícím modulem. Ovladač zveřejní seznam PCI zařízení, které je schopen obsluhovat. Seznam se zveřejňuje prostřednictvím struktury `pci_device_id` v době sestavení modulu. Na základě poskytnuté identifikace se pak v době startu systému nebo v okamžiku připojení PCI zařízení zavede odpovídající modul. Pro úplnost, struktura `pci_device_id` obsahuje položky odpovídající konfiguračním registrům uvedeným v části 3.1.1.

V inicializační metodě se zaregistruje ovladač v datových strukturách jádra voláním `pci_register_driver`. Analogicky se v době odstranění odebere záznam o ovladači prostřednictvím `pci_unregister_driver`. Při registraci zařízení získá jádro odkazy na funkce, které musí implementovat každý ovladač PCI zařízení, `probe` a `remove`. Během

své registrace dostane ovladač možnost obsluhovat všechna neobsazená zařízení, která odpovídají zveřejněným identifikačním údajům (viz část 3.1.1), vyvoláním právě metody `probe`. Pokud ovladač zařízení rozpozná, oživí jej funkcí `pci_enable_device`. Detailnější popis uvedených funkcí lze nalézt v [5].

3.3.3 Registrace a obsluha přerušení

Nalezení přerušovacího vektoru, který má ovladač obsloužit je obecně netriviální věc. Ale vzhledem k tomu, že se implementuje ovladač zařízení pro sběrnici PCI, celá situace se výrazně zjednodušuje a nebude zde tedy rozebírán způsob detekce přerušovacího vektoru. Podrobnosti lze nalézt v [5].

Přidělení vektoru přerušení pro PCI zařízení proběhne již v době zavedení počítače a jedinou zodpovědností ovladače PCI zařízení je vyčíst tuto hodnotu z konfiguračního prostoru, obvykle pomocí `pci_read_config_byte`. Po získání hodnoty vektoru přerušení se registruje obsluha pomocí funkce jádra `request_irq`. Zde je vhodné poznamenat, že pro správnou činnost sdílených přerušení (viz část 2.3.3) musí být nastaven bit `SA_SHIRQ` bitové masky `flags`. Pro uvolnění registrovaného přerušovacího vektoru slouží volání `free_irq`.

Dále je důležité dodržet několik omezení a požadavků kladených na funkci provádějící obsluhu přerušení. Jelikož neběží v kontextu procesu, nesmí přenášet data z uživatelského prostoru. Dále nesmí provádět žádnou operaci, která by mohla vést k pozastavení provádění, jako například čekat na událost, alokovat paměť jinak než s parametrem `GFP_ATOMIC`, nebo zamykat semafor. Naopak by měla informovat zařízení o zpracování vyvolaného přerušení.

3.3.4 Síťové rozhraní

Inicializace a ukončení

Správný okamžik k vytvoření síťového rozhraní nastává v době rozpoznání periferního zařízení, tj. v době volání funkce `probe`. Síťové zařízení se vytváří ve dvou krocích. Nejprve se alokuje struktura `net_device` voláním `alloc_netdev` a nastaví se položky struktury, nebo lze využít funkce `alloc_etherdev`, která provede oba kroky v případě, že je vytvářen ovladač zařízení pro sítě typu *Ethernet*.

Síťové rozhraní se registruje pomocí `register_netdev`. Od tohoto okamžiku může být ovladač kdykoliv požádán o odeslání paketu.

Duální operace volané v opačném pořadí sloužící k odstranění síťového rozhraní se nazývají `unregister_netdev` a `free_netdev`.

Operace se zařízením

Jakmile je rozhraní registrováno v rámci systému, může být použito pro změnu nastavení, příjem a odesílání paketů. Před tím než uživatel začne posílat a přijímat samotná data, musí aktivovat síťové rozhraní, například programem *ifconfig*. Událost aktivace se šíří až k samotnému ovladači zařízení. Systém vyvolá metodu ovladače `open`. Ta zajistí aktivaci fronty pro odesílání paketů funkcí `netif_start_queue`.

Analogický postup nastane v případě uzavření rozhraní. Jádro vyvolá metodu `stop`, která zastaví frontu pro odesílání paketů použitím funkce `netif_stop_queue`.

K samotnému odeslání paketu slouží `hard_start_xmit`. Pokud vznikne potřeba pozastavit odesílání paketů, například kvůli vyčerpání paměti na síťové kartě, lze odesílání

pozastavit funkcí `netif_stop_queue` a v okamžiku obnovení dostatečného množství zdrojů znovu nastartovat funkcí `netif_wake_queue`.

Na včasné odeslání paketů dohlíží časovač. Pokud dojde k překročení nastaveného intervalu, vyvolá jádro funkci ovladače `tx_timeout`, která se postará o odstranění případných problémů a úspěšné odeslání rozpracovaných paketů.

Pokud síťová karta disponuje schopností scatter/gather přenosů (viz část 2.2.2), musí ovladač při odesílání paketu projít lineární seznam všech fragmentů paketu a provést jejich odeslání.

Příjem paketů závisí na externí události přerušení, kterou periferní zařízení informuje ovladač o přijatém paketu. Existuje několik variant realizace příjmu paketů lišících se ve způsobu alokace a plnění struktury `sk_buff`.

Varianta 1 Po obdržení přerušení ovladač vytvoří instanci struktury a sdělí zařízení adresu. Na tuto adresu zařízení přenese přijatý paket a dokončení přenosu signalizuje přerušením. Takovýto přístup je značně neefektivní, neboť vzniknou dvě přerušení na jeden paket. Což může vést k vyčerpání značného množství strojového času jen na obsluhu přerušení, nehledě na prodlevu způsobenou výměnou informací mezi zařízením a ovladačem.

Varianta 2 V současnosti používaný způsob (viz [1] soubor *drivers/net/e100.c*) spočívá v dopředné alokaci paměti pro uložení paketů. Adresy jednotlivých paměťových míst jsou uloženy v zařízení a v okamžiku přijetí paketu se tento přesune na připravené místo. Teprve pak dojde k vyvolání přerušení a paket může být dále zpracován bez zbytečného čekání na realizaci přenosu. Ukazuje se však, že i poměr jedno přerušení na paket je příliš vysoký.

3.3.5 NAPI: bližší pohled

Jak bylo uvedeno v části 2.3.4, používá NAPI k dosažení vyšší propustnosti správu množství signalizovaných přerušení a zahazování paketů již v hardware, obojí v případě vysoké zátěže.

Předpoklady pro realizaci ovladače využívajícího NAPI jsou:

1. dostatečná paměť pro dočasné uložení paketů před jejich zpracováním,
2. možnost atomicky pozastavit generování přerušení v hardware a
3. schopnost korektně detekovat novou práci respektive nově příchozí pakety.

První požadavek je odůvodněn potřebou uschovat pakety po dobu, než je provedeno zpracování paketů, které není přímou reakcí na příchod přerušení. Výhoda rozhraní NAPI pak vychází z odbourání režie spojené s obsluhou přerušení pro každý přijatý paket. K tomu je nezbytný druhý požadavek, tj. možnost zablokovat signalizaci přerušení. S okamžikem znovu-povolení přerušení se váže potenciální ztráta přerušení a následné stárnutí paketu, proto existuje třetí požadavek na korektní detekci nově přijatých paketů (viz [7] příloha 2).

Redukce množství přerušení

Pokud je každý příchod paketu, potažmo okamžik, kdy je paket nakopírován do hlavní paměti, signalizován přerušením, dochází ke konzumaci velké části procesorového času jenom na obsluhu samotného signálu přerušení a nezbývá prostor ke zpracování paketů. NAPI proto zavádí funkce jádra a pravidla, při jejichž správném použití nedojde k popsanému efektu.

První krok spočívá v úpravě obsluhy přerušení. V okamžiku vyvolání funkce pro obsluhu přerušení se zablokuje v hardware generování dalších signálů informujících o příchodu paketu a naplánuje se obsluha pomocí funkce `poll`, která je opakovaně aktivována jádrem. Plánování se provádí voláním `netif_rx_schedule`.

Metoda `poll` provádí činnost původně vykonávanou v obsluze přerušení. S odlišností, že k předání paketu vyšším vrstvám se použije jiná funkce, konkrétně `netif_receive_skb`. Odlišná funkce se zavádí z důvodu kontroly počtu zpracovaných paketů v rámci jedné aktivace metody `poll`. Pokud se podaří zpracovat všechny přijaté pakety je ukončena opakovaná obsluha funkcí `netif_rx_complete` a aktivuje se přerušení. Ve výsledku vzniká samoregulující se obsluha, která v době vysoké zátěže zabraňuje vzniku velkého množství přerušení. Detailnější popis lze nalézt v [2].

Zahazování paketů v hardware

Pokud je použita pro příjem paketů uvedená metodika, kdy se přímo v zařízení ukládají adresy paměťových míst, lze detekovat okamžik vyčerpání dostupných zdrojů již v hardware a je možné zahazovat pakety bez zatěžování jak interní sběrnice, tak sběrnice systémové a následně i procesoru.

3.3.6 Alokace DMA paměti

Jak bylo uvedeno v části 2.3.2, existují dva druhy mapování DMA paměti. V jádře Linuxu lze pro mapování DMA paměti použít buďto obecné DMA rozhraní nebo rozhraní specifické pro použitou systémovou sběrnici, v tomto případě PCI. Pro vytvářený ovladač lépe vyhovuje specifická sada funkcí, neboť pokrývá funkcionalitu nabízenou sběrnicí PCI, která není dostupná u jiných sběrnic a naopak.

Pro přidělení a uvolnění konzistentní paměti slouží funkce `pci_alloc_consistent` a `pci_free_consistent`. Pro úseky velikosti menší než jedna stránka se používá `pci_pool_create`, `pci_pool_alloc`, `pci_pool_free` a `pci_pool_destroy`. Tyto funkce provádějí souběžně jak alokaci, tak mapování paměti.

U proudového mapování se nejprve alokuje paměť s použitím běžných funkcí jádra a následně nastaví mapování, u kterého je třeba vždy uvést směr, ve kterém budou data přenášena. Buďto `PCI_DMA_BIDIRECTIONAL` pro obousměrný přenos, přenos do zařízení `PCI_DMA_TODEVICE` nebo přenos ze zařízení `PCI_DMA_FROMDEVICE`. Mezi funkce sloužící k nastavení proudového mapování patří `pci_map_single` a `pci_unmap_single`.

Podrobnější popis lze nalézt v [1] v souboru *Documentation/DMA-mapping.txt* nebo také v [5].

3.3.7 Operace se strukturou `sk_buff`

Jádro Linuxu obsahuje řadu funkcí zjednodušujících práci se strukturou `sk_buff`. Popis nejdůležitějších následuje, zbylé lze nalézt v [5].

- `alloc_skb` vytvoří instanci struktury typu `sk_buff`. Zároveň inicializuje prvky `data` a `tail` této struktury na hodnotu prvku `head`. Toto je výchozí stav a odpovídá prázdné struktuře nenesoucí žádný paket.

- `dev_kfree_skb` se postará o bezpečné uvolnění instance alokované například předchozí funkcí.

- `skb_put` aktualizuje patřičné prvky struktury tak, aby bylo možné zapsat na získanou návratovou adresu požadovaná data. Je však nezbytné, aby množství vkládaných dat nepřekročilo velikost vytvořeného bufferu.
- `skb_push` rovněž modifikuje prvky struktury `sk_buff`, tentokrát však za účelem vložení dat na začátek bufferu. Obvykle je využívána pro vložení hardwarové hlavičky na začátek paketu před jeho předáním do zařízení.
- `skb_tailroom` slouží ke zjištění množství dostupného volného místa v bufferu. V praxi značí použití této funkce spíše problém na straně návrhu, ale například pro ladící účely je tato funkce vhodná.
- `skb_headroom` zjistí množství volného alokovaného místa před samotnými daty. O použití platí stejné tvrzení jako u předchozí funkce.
- `skb_reserve` lze využít k vyhrazení místa v bufferu pro pozdější účely. Například pro pozdější vložení hardwarové hlavičky obsahující řídící informace důležité ke zpracování v hardware.
- `skb_pull` je zde uvedena pouze pro úplnost. Umožňuje modifikací prvků struktury zrušit dříve vyhrazené místo.
- `skb_is_nonlinear` vrací jednobitovou logickou informaci o tom, zda-li je paket uložen ve více fragmentech. Tato informace je důležitá především pro realizaci scatter/gather přenosů.

3.4 Možnosti implementace síťových zařízení

V oblasti zpracování vysokorychlostních paketových přenosů v zásadě existují dva hlavní přístupy k tvorbě síťové karty. Každý přístup má pochopitelně své výhody i nevýhody a výběr mezi nimi určuje cílová aplikace.

Standardní síťové rozhraní

Jedná se o rozhraní popsané v předchozí části textu. Jak už bylo uvedeno, základní datovou jednotkou je paket a prostupuje celým zpracováním od software až k hardware. Nad standardním rozhraním implementovaným v jádře Linuxu pracuje široká škála programů pro správu nastavení, monitorování a v neposlední řadě komunikaci. Hodí se tedy jak běžným uživatelům, tak správcům sítě či odborníkům, kteří studují vzory síťového provozu.

Obecnost použití však sebou přináší větší množství operací prováděných s každým paketem. Zejména pak standardní operace a statistiky počítané v jádře Linuxu, které nemusí být pro jisté aplikace užitečné.

Speciální síťová rozhraní

Hodilo-li se všeobecné zpracování paketů pro výše zmíněné uživatele, pro účely vysokorychlostního monitorování je tento přístup nevhodný. Speciální síťová rozhraní se zaměřují na poskytnutí maximální propustnosti a na ponechání volnosti ve volbě operací prováděných při zpracování paketu na cílové aplikaci.

Příkladem speciálního rozhraní je rozhraní *szedata* a přímý nástupce *szedataII*, jejichž cílem je efektivní způsob předání dat přijatých z počítačové sítě respektive odesílaných

do počítačové sítě. Na paket je pohlíženo jako na obecný blok dat, který není vnitřně nijak interpretován. Rozhraní bylo navrženo a implementováno v rámci výzkumné aktivity *Liberouter* společnosti *CESNET*. Stejně jako v případě této diplomové práce se jednalo o realizaci softwarové a hardwarové části.

Přínosem tohoto přístupu je velice rychlá možnost výměny dat. Nevýhodou je však nutnost implementovat veškerou funkcionalitu odděleně. Obvykle bez možnosti využití obecných nástrojů, funkcí a knihoven. Rychlostní přínos ale převažuje nad nedostatkem komfortu při tvorbě rozhraní a koncových aplikací.

3.5 Architektura ovladačů *COMBO*

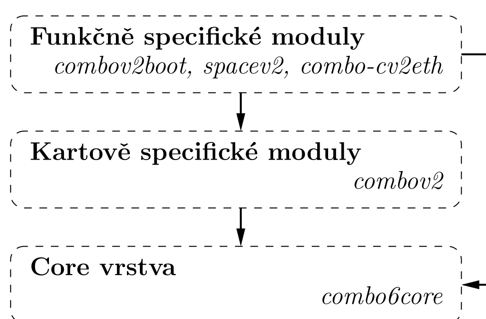
Stejně jako v hardware existuje připravená platforma pro snadnější realizaci síťové či jiné karty v podobě platformy NetCOPE, i na straně software lze využít již hotovou infrastrukturu pro jednoduší vytvoření ovladače.

Jaderné moduly COMBO lze rozdělit do tří vrstev. Každá vrstva plní odlišnou funkci a není vždy žádoucí ji implementovat znovu. Obrázek 3.5 graficky doplňuje následující text.

1. *Core vrstva.* Tato vrstva je tvořena jediným jaderným modulem s názvem *combo6core.* Vytváří spojovací článek všech ostatních modulů, které se u něj registrují. Poskytuje informace o dostupných kartách a ovladačích pro tyto karty určených. Spravuje virtuální souborová zařízení.

2. *Kartově specifické moduly.* Moduly jsou určeny pro nízkoúrovňovou obsluhu karet, tak aby moduly vyšších vrstev mohli jednoduše pracovat s danou kartou. V současné době existují tři kartově specifické moduly: *combo6, combo6x, combov2.* Názvy odpovídají hardwarovým kartám.

3. *Funkčně specifické moduly.* Tyto moduly se vážou na aktuálně naprogramovanou funkci použité karty. Mohou například realizovat rychlé přenosy mezi hardware a software, obsluhovat hardwarově akcelerované výpočty nebo poskytovat standardní síťové rozhraní OS Linux.



Obrázek 3.5: Vazba mezi moduly *COMBO*

Náplní této práce je vytvořit ovladač síťové karty pro platformu COMBOv2. To odpovídá použití modulu *combo6core*, dále pak kartově specifického modulu *combov2* a zbývá doplnit funkčně specifický modul, který poskytne standardní síťové rozhraní.

3.5.1 Modul *combo6core*

Základní datovou strukturou, která spojuje jednotlivé vrstvy k sobě, je struktura `struct combo6`. Ke každé hardwarové kartě existuje jedna instance této struktury obsahující odkaz na kartově specifický modul, přidělené PCI zařízení, typ karty, adresové prostory karty. Dále pak údaje o rozšiřujících kartách, počtu rozhraní, vektoru přerušení, použitelných zařízeních a další. Pod pojmem použitelná zařízení se rozumí množina funkčně specifických modulů, které jsou schopny obsluhovat aktuálně naprogramovanou hardwarovou funkci. Pro ještě jemnější rozlišení aktuálního designu se používá identifikační jednotka, která svým obsahem rozhodne, který z množiny modulů bude použit.

Modul *combo6core* poskytuje sadu funkcí pro alokaci a uvolnění, přidání a odebrání struktury `combo6`. Dále pak vytváří, ruší a obsluhuje speciální soubor zařízení umístěný v adresáři */dev*.

3.5.2 Modul *combov2*

Jeden z kartově specifických modulů. Využívá konkrétní znalosti použité karty pro její registraci prostřednictvím funkcí poskytnutých předchozím modulem.

Důležitou úlohou, která se v závislosti na použité kartě různí, je zavedení designu či funkce. Jinými slovy se jedná o naprogramování karty. Stejně tak se podle typu karty liší podporovaný způsob signalizace přerušení. Vzhledem k tomu, že karta COMBOv2 je připojitelná skrze rozhraní PCI-Express, nabízí se možnost použít úrovní signalizované přerušení, nebo MSI či MSI-X. Z důvodu chybějící podpory MSI-X ze strany hardwarového PCI-Express bloku použitého v platformě NetCOPE, zbývá pouze alternativa MSI, nebo úrovní signalizované přerušení. Použití MSI omezuje aktuální stav jádra OS Linux, kdy podpora využití více přerušovacích vektorů byla přidána teprve v jádře verze 2.6.30 (viz [1]).

3.5.3 Modul *combo-cv2eth*

Stejně jako ostatní funkčně specifické moduly musí implementovat základní operace pro registraci, přidání a uvolnění modulu, obsluhu přerušení a další. K tomu využije funkcí poskytnutých ovladačem *combov2* a jeho prostřednictvím funkcí modulu *combo6core*.

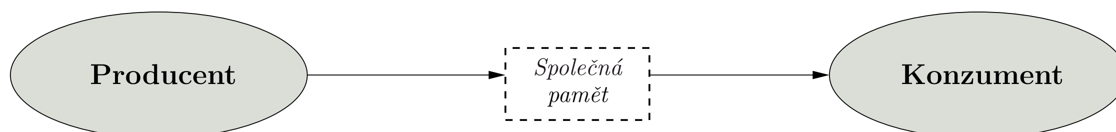
Kapitola 4

Návrh

Prozkoumáním cílové platformy byl získán základ, který bude dále rozšířen v návrh stavebních bloků. Jelikož se jedná o aplikaci zabývající se vysokorychlostním zpracováním paketů, je klíčová především propustnost celého systému a jako součást návrhu bude provedena její analýza.

4.1 Komunikace ovladač - DMA řadič

Realizace ovladače a DMA řadiče vyžaduje jednoznačnou specifikaci rozhraní či protokolu, jakým komunikace probíhá. Z obecného hlediska se jedná o problém typu *producent-konzument*, kdy do paměti omezené velikosti zapisuje producent data, která jsou určena konzumentovi. Je zřejmé, že pokud by tyto entity mezi sebou neudržovali žádnou informaci o aktuálním stavu paměti, mohlo by dojít k přepisování dat producentem ještě předtím, než měl konzument možnost tato data zpracovat. Při odesílání paketů produkuje data operační systém, respektive jej využívající běžící uživatelské programy. Konzumentem je síťová karta. Příjem paketů probíhá v právě opačném smyslu úlohy producent-konzument.



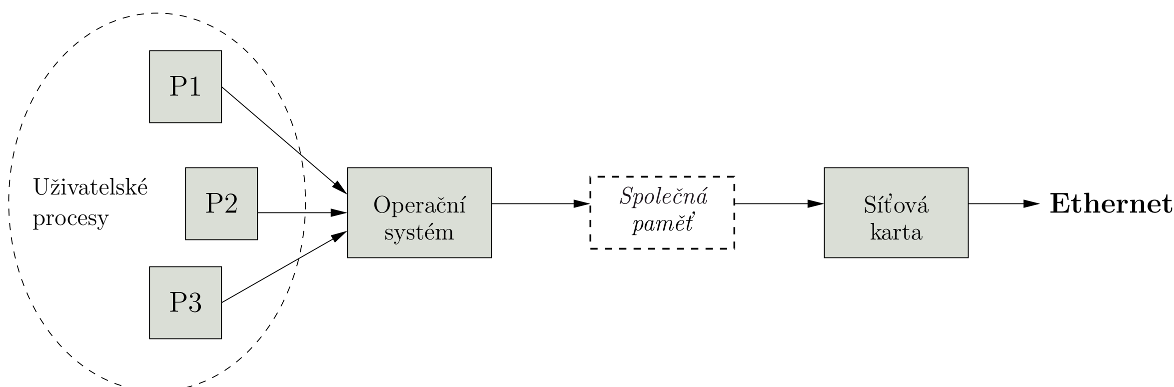
Obrázek 4.1: Model producent-konzument

Společnou paměť pro výměnu dat reprezentuje v tomto případě část operační paměti počítače určená pro DMA přenosy. Situaci zachycuje obrázek 4.1. Pro správnou činnost je nezbytné, aby při zaplnění této paměti producent, ať už operační systém při odesílání, či síťová karta při příjmu, ustal ve své činnosti a dal možnost konzumentovi zpracovat všechna nashromážděná data.

Logickým důsledkem čekání producenta na konzumenta je potenciální ztráta dat. Pokud totiž síťová karta neustále přijímá ze sítě nové pakety a má k dispozici pouze omezené množství vyrovnávací paměti, dojde k zaplnění jak vyrovnávací paměti, tak společné paměti pro výměnu dat a nezbývá než následující pakety zahazovat. Problém je tedy rychlost producenta.

V opačném případě, tj. při odesílání dat, nenastává problém s rychlostí producenta, anebo lze účinně řešit na úrovni operačního systému. Doplněním obecného obrázku 4.1

o konkrétní producenty a konzumenty dat lze získat návod na řešení problému rychlosti producenta při odesílání dat. Z obrázku 4.2 je patrné, že pozastavováním procesů *P1, P2, P3* se dosáhne regulace rychlosti producenta a nemusí dojít k zahazování.



Obrázek 4.2: Doplněný model producent-konzument

Jako poznámku je vhodné dodat, že se zahazováním paketů, ať už z důvodu nedostatečné rychlosti konzumenta, či výpadku na vedení, počítají běžně používané protokoly, jako například *TCP*.

Pro úplnou analýzu a zvládnutí problému producent-konzument zbývá odpovědět na dvě otázky:

- Jak bude prováděna správa společné paměti? Správou paměti se rozumí udržování informace o stavu zabrané, respektive dostupné paměti. V tomto ohledu je myšlenkově i implementačně nejjednodušší správa paměti prostřednictvím kruhových bufferů. Kruhový buffer je taková paměťová organizace běžného jednorozměrného pole či vektoru, u které je následníkem posledního prvku prvek první. A naopak předchůdcem prvního prvku prvek poslední.

- Jak signalizovat zpracování či dostupnost nových dat za použití dané správy paměti? Producent a konzument se musí shodnout na hodnotách ukazovátek či indexů označujících začátek a konec obsazených dat. Pokud se dodrží pravidlo, že dostupnost nových dat ze strany producenta (zpracovanost dat ze strany konzumenta) je signalizována protistraně až v okamžiku, kdy jsou data v bufferu platná (data z bufferu zpracována), nedojde ke čtení (uvolnění) dosud neplatných dat.

  Producent a konzument si tedy vzájemně zasílají zprávy o změně stavu indexů. Čekání na tyto zprávy se v software typicky realizuje čekáním na asynchronní příchod přerušení, případně periodickým testováním na základě události časovače. Naopak v hardware se obvykle čeká v aktivní smyčce.

4.1.1 Velikost DMA bufferů

Pro další postup v návrhu je nezbytný alespoň teoretický odhad velikosti bufferů v hlavní paměti. Tyto buffery slouží jako vyrovnávací paměť pro uložení přijatých paketů v okamžiku, kdy procesor zpracovává jiné úlohy než příjem paketů. Při všech analýzách bude uvažován vždy nejhorší případ, který představují nejkratší pakety. Z pohledu bufferů toto tvrzení platí rovněž.

Efektivní způsob komunikace reprezentuje druhá varianta popsaná v části 3.3.4, tj. dopředná alokace paměti pro příjem paketů. V takovéto situaci se alokují bloky paměti schopné pojmout největší možné pakety. Pokud ale budou přijímány nejkratší pakety, zůstane podstatná část přidělené paměti nevyužita. Cílem přesto zůstává dosažení maximální propustnosti, proto neúplné využití paměti nevadí.

Rychlost linky	10 Gb/s
PPS	14,8 milionů paketů
Mezera systémového plánovače	0,05 s
MTU	1522 B

Tabulka 4.1: Základní údaje pro odvození velikosti bufferů

Výchozí údaje shrnuje tabulka 4.1. MTU znamená maximální délku IP paketu v síti typu *Ethernet* (viz část 2.1.1) a PPS počet paketů za sekundu. Výpočet PPS vychází z minimální délky IP paketu 64 B, mezipaketové mezery 12 B, preambule ethernetového rámce 7 B a omezovače počátku rámce 1 B (anglicky *starting frame delimiter*, *SFD*), dohromady tedy 84 B. PPS se potom vypočte ze vztahu $\frac{rychlost_linky}{delka_paketu}$. Dále musí alokovaná paměť pokrýt nejen okamžik, kdy procesor zpracovává jiné úlohy, ale také dobu, po kterou se provádí samotné zpracování paketů, tedy dvojnásobek délky mezery systémového plánovače.

Výsledná velikost V se vypočte:

$$V = 2 * MSP * PPS * MTU$$

Po dosazení hodnot z tabulky:

$$V = 2 * 0,05 * 14,8 * 10^6 * 1522 = 2,25\,GB$$

Pro úplnost je uvedena tabulka 4.2 shrnující velikost bufferů pro různé rychlosti linky. *Rx* označuje příjem, *Tx* odesílání paketů.

Rychlost linky	Velikost Rx paměti	Velikost Tx paměti
1 Gb/s	0,225 GB	12,5 MB
10 Gb/s	2,25 GB	125 MB
40 Gb/s	9,0 GB	500 MB

Tabulka 4.2: Velikost bufferů pro příjem při různých rychlostech sítě

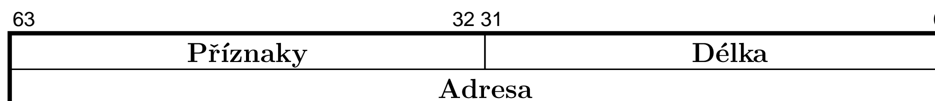
Horní odhad množství potřebné paměti pro odesílání paketů je nižší, neboť nedochází k fragmentaci paměti alokací bufferů pro pakety dopředu neznámé velikosti. Velikost potřebné paměti lze vypočítat jako množství dat odeslaných na dané přenosové rychlosti za dvojnásobnou dobu mezery systémového plánovače, tedy podle vztahu:

$$V = 2 * MSP * Rychlost_linky$$

4.1.2 Deskriptor

Pro popis paměťového místa sloužícího k uložení paketu v hlavní paměti se zavádí označení *deskriptor*, neboť *popisuje* konkrétní instanci struktury `sk_buff`.

Formát deskriptoru lze vidět na obrázku 4.3. Význam a velikost polí je následující:



Obrázek 4.3: Formát deskriptoru

- Délka. 32bitová hodnota udávající velikost paměťového místa. Velikost položky vychází z velikosti prvku `data_len` struktury `sk_buff`, která je typu `unsigned int` (viz [1] soubor *include/linux/skbuff.h*). Uvedený datový typ má velikost 32bitů, jak na 32bitové architektuře *i386* či *i686* (viz [5]), tak na 64bitové architektuře označované *amd64* (viz [10] a [5]).
- Příznaky. 32bitová hodnota obsahující pomocné příznaky, například příznak přerušení nebo posledního segmentu paketu u scatter/gather přenosu. Velikost je stanovena tak, aby velikost celého deskriptoru byla zarovnaná na 64 bitů.
- Adresa. 64bitová adresa paměťového místa. Velikost adresy vyplývá z potřeby použít velký logický adresový prostor jádra, aby bylo možné alokovat dostatečné buffery (viz část 4.1.1). Tento prostor poskytuje 64bitová architektura, která používá právě 64bitové adresy.

4.1.3 Princip činnosti

Nyní bude popsán základní pracovní cyklus při odesílání a příjmu paketů, každý odděleně.

Odesílání paketů

Softwarová část

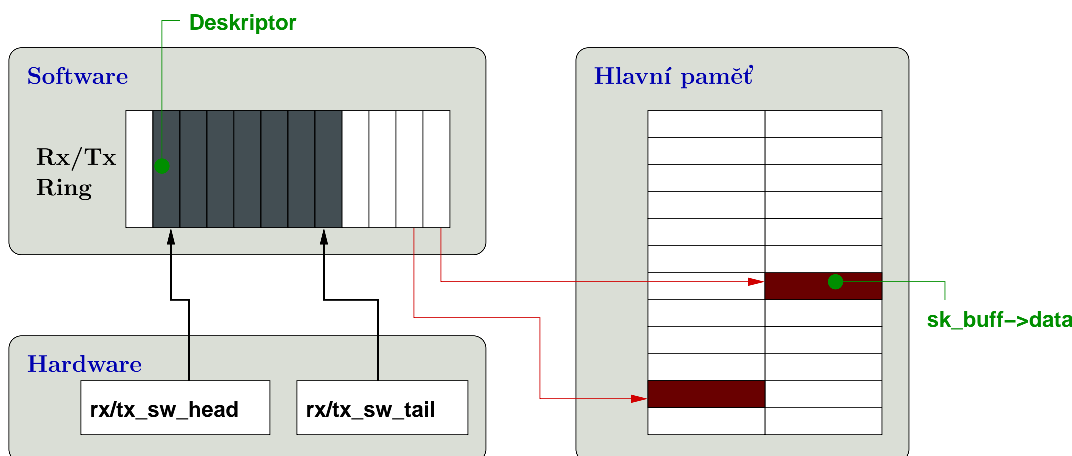
1. V okamžiku získání PCI zařízení, tj. v okamžiku volání metody `probe` (viz 3.3.2), alokuje ovladač kruhový buffer sloužící pro uložení deskriptorů, pracovně označen *Tx Ring*. Předá zařízení počáteční adresu.
2. Jakmile je aktivováno síťové rozhraní, aktivuje ovladač odesílání paketů v hardware a očekává příchod paketů z jádra.

   Pro zjištění stavu odeslaných paketů vytvoří ovladač paměťové místo určené pro zápis ze strany zařízení. Adresu nastaví do příslušného registru v hardware.
3. Pokud existuje paket k odeslání, jádro vyvolá metodu `hard_start_xmit`, ovladač vytvoří deskriptor, který vloží do příslušného Tx Ringu a následně zapíše informaci o připraveném paketu do vyhrazeného řídícího registru v hardware označeného **sw_tx_tail**.
4. Se signálem přerušení synchronizuje oblast obsahující stav odeslaných paketů pro procesor a přečtením hodnot zjistí, kolik paketů bylo odesláno. Podle toho upraví *Tx Ring* a uvolní obsah deskriptorů.
5. Při uzavření rozhraní zastaví činnost hardware.
6. Při odebrání zařízení uvolní *Tx Ring*.

Hardwarová část

1. Po aktivaci čeká zařízení na zápis ze strany ovladače.
2. S příchodem zápisu do registru **sw_tx_tail** vystaví požadavek na stažení připravených deskriptorů. Po jejich příchodu postupně prochází jeden po druhém a vytváří požadavky na přenos dat z paměťových míst a o délkách uvedených v deskriptorech. S každým požadavkem aktualizuje vnitřní registr **sw_tx_head**. Deskriptory naplánované ke zpracování tedy leží v *Tx Ringu* mezi indexy **sw_tx_head** a **sw_tx_tail**.

   Situaci přehledně ukazuje obrázek 4.4.



Obrázek 4.4: Organizace paměti a *Rx/Tx Ringu*

3. Po dokončení DMA přenosu informuje řadič příslušný hardwarový DMA buffer, označen **tx_dma_buffer**, který paket odešle do síťového modulu a následně přes fyzické rozhraní do sítě. Informaci o odeslaném paketu uchovává DMA řadič pro pozdější předání ovladači. Toto předání proběhne buď na základě příznaku přerušení v deskriptoru, nebo po vypršení časového intervalu. V obou případech se signalizuje přerušení.

Odesílání paketů lze tedy chápat jako dvojí realizaci modelu producent-konzument. Nejprve je třeba na základě signalizace ovladače stáhnout deskriptory a následně tyto zpracovat. Což odpovídá odeslání paketových dat. V obou případech je shodně producentem software a konzumentem hardware.

Příjem paketů

Softwarová část

1. Analogicky k předchozí části, k vytvoření *Rx Ringu* dojde v době získání PCI zařízení. Zařízení obdrží počáteční adresu *Rx Ringu*.
2. Na aktivaci rozhraní reaguje ovladač tak, že naplní *Rx Ring* deskriptory, tj. alokuje struktury `sk_buff`, a aktivuje příjem paketů v hardware.

3. Následně ovladač čeká na vyvolání přerušení. Po vyvolání přerušení zpracuje přenesené pakety podle aktualizovaného obsahu *Rx Ringu* a předá je jádru. Namísto použitých deskriptorů naplní nové a informuje hardware o množství nově dostupných deskriptorů aktualizací registru **sw_rx_head**.

4. Při uzavření rozhraní zastaví činnost hardware a uvolní alokované deskriptory.

5. Při odebrání zařízení uvolní *Rx Ring*.

Hardwarová část

1. Po aktivaci zadá DMA přenos pro stažení deskriptorů a čeká na příchod dat od **rx_dma_bufferu**.

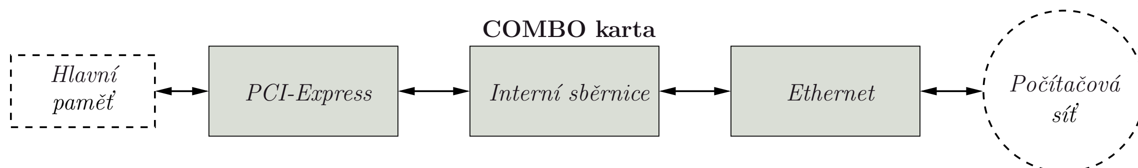
2. Pro přijatý paket vystaví požadavek na přenos do hlavní paměti. S každým požadavkem upraví hodnotu registru **sw_rx_tail**. Aktualizuje deskriptor délkou přijatého paketu a zařadí jej do fronty pro upload deskriptorů zpět do *Rx Ringu*. Po dokončení přenosu, pokud byl v jednom z deskriptorů nastaven příznak přerušení, signalizuje příchod paketu(ů) ovladači přerušením. Funkci popsaných registrů a organizaci paměti ukazuje obrázek 4.4.

3. Po ukončení přenosu paketu do hlavní paměti oznámí příslušnému **rx_dma_bufferu**, že byl přenos úspěšně proveden a paket může být uvolněn z paměti na kartě.

Příjem paketů má tedy, stejně jako odesílání, charakter dvojí realizace modelu producent-konzument. Opět se jedná o správu deskriptorů a následné přenosy paketových dat. Avšak role software a hardware je v obou realizacích odlišná. Pro správu deskriptorů zaujímá software roli producenta a hardware konzumenta. Co se týká paketových dat, je tomu přesně naopak.

4.1.4 Analýza propustnosti

Nyní, po objasnění způsobu, jakým bude probíhat komunikace, co všechno poputuje z karty do hlavní paměti a naopak, lze určit vytížení a teoretickou propustnost pro celý systém, který znázorňuje obrázek 4.5. Pro výpočet je důležitá část 3.2.2 a [9].



Obrázek 4.5: Propojení komunikačního řetězce

Při analýze propustnosti se zkoumá obousměrný provoz na interní sběrnici, který představuje nejhorší případ. V průběhu analýzy je rozlišeno jestli se provoz odehrává na kanále *uplink* či *downlink* interní sběrnice. Z výpočtu jsou vypuštěny zápisy do registrů informující o nově dostupných deskriptorech, jelikož neprobíhají pro každý paket a celkovou propustnost ovlivní minimálně.

1. Stažení deskriptorů pro Rx. Generuje se jeden požadavek na stažení bloku paketů kanálem uplink, ale protože se tato režie amortizuje na celém bloku paketů, nebude uvažována. Pro každý přijatý paket se kanálem downlink přenese jeden deskriptor.
2. Stažení deskriptorů pro Tx. Platí stejný případ jako u Rx.
3. Přenos přijatých paketů z karty. Zátěž pro uplink, kdy s každým paketem putuje jedna hlavička IB paketu.
4. Přenos aktualizovaných deskriptorů z karty. Zátěž pro uplink, kdy po přijetí paketu musí být přenesen aktualizovaný deskriptor. Opět přenášen po interní sběrnici ve formě IB paketu s hlavičkou obsahujícího blok deskriptorů. Zátěž se, podobně jako v prvním případě, amortizuje.
5. Přenos odesílaných paketů do karty. Z karty se odešle požadavek na čtení z hlavní paměti. Kanálem uplink jeden IB paket obsahující pouze hlavičku. Odpověď kanálem downlink, odesílaný paket jako IB paket s hlavičkou.
6. Aktualizace počtu odeslaných paketů. Odesílá se jednou pro blok paketů, proto se tato zátěž neuvažuje.

Nechť existuje obecné označení

počet Rx paketů	cRx
počet Tx paketů	cTx
délka Rx paketu	sRx
délka Tx paketu	sTx
velikost deskriptoru	sd
velikost hlavičky IB paketu	sh

S využitím právě zavedeného značení lze psát:

Operace	downlink	uplink
1	$cRx * sd$	-
2	$cTx * sd$	-
3	-	$cRx * (sRx + sh)$
4	-	$cRx * sd$
5	$cTx * (sTx + sh)$	$cTx * sh$

Tabulka 4.3: Zatížení interní sběrnice

Interní sběrnice je plně duplexní. Nechť $IBbps$ označuje rychlost v jednom směru. V součtu pro downlink tedy platí:

$$IBbps = cRx * sd + cTx * (sTx + sd + sh)$$

a pro uplink:

$$IBbps = cRx * (sRx + sd + sh) + cTx * sh. \quad (4.1)$$

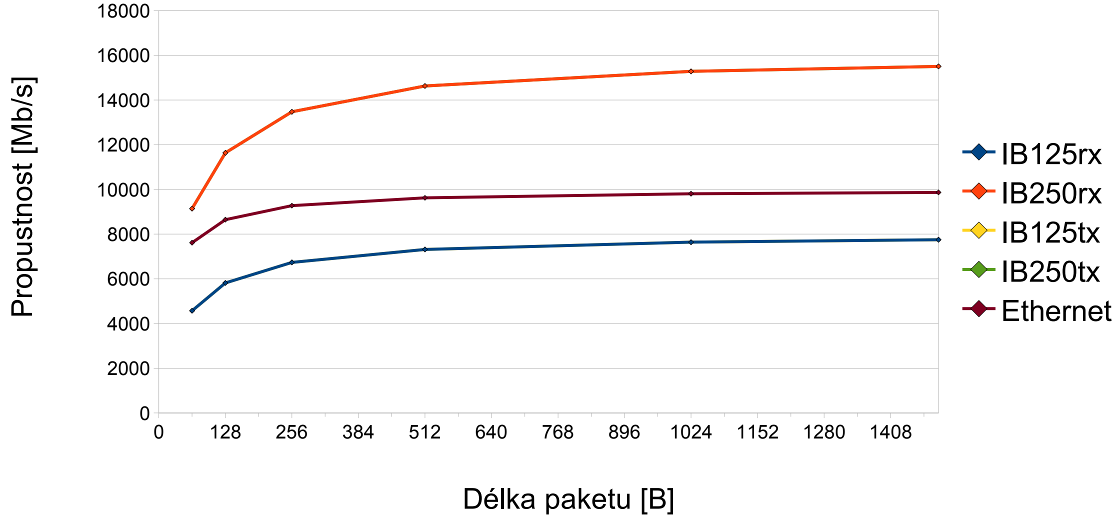
Po úpravě:

$$\frac{cTx}{cRx} = \frac{sRx + sh}{sTx + sd}. \quad (4.2)$$

Velikost deskriptoru i hlavičky paketu jsou konstantní a obě shodně velké 16 B. Nejhorší případ nastává pro nejkratší pakety. Za sRx a sTx je tedy shodně dosazeno 64 B:

$$\frac{cTx}{cRx} = \frac{sRx + sh}{sTx + sd} = \frac{80}{80} = 1. \tag{4.3}$$

Jak lze vidět, je dosaženo identického zatížení obou kanálů.



Obrázek 4.6: Graf propustnosti

Závislost počtu přijatých paketů na délce paketu a rychlosti sběrnice získaná dosazením z rovnice 4.2 do 4.1:

$$cRx = \frac{IBbps}{sRx + sd + sh * (1 + \frac{sRx+sh}{sTx+sd})}. \tag{4.4}$$

S využitím vztahu 4.4 s levou stranou rozšířenou o sRx

$$sRx * cRx = \frac{IBbps}{sRx + sd + sh * (1 + \frac{sRx+sh}{sTx+sd})}$$

lze sestavit graf na obrázku 4.6. Prostřední křivka reprezentuje špičkovou propustnost Ethernetu počítanou se zátěží 20 B na jeden paket (viz část 4.1.1). Spodní křivka představuje propustnost Rx a Tx směru při použití interní sběrnice na frekvenci 125 MHz, horní na frekvenci 250 MHz. Z toho plyne, že pro zpracování plného 10 Gb/s toku je třeba interní sběrnice alespoň na frekvenci 250 MHz.

Křivky pro Rx a Tx se v grafu překrývají, to plyne z rovnice 4.3.

K ověření propustnosti celého komunikačního řetězce (viz obrázek 4.5) je třeba uvážit i propustnost systémové sběrnice, tj. PCI-Express. Špičková propustnost použité varianty této sběrnice je 20 Gb/s (viz část 3.1.1), což je dostačující.

4.2 Ovladač

Softwarová část práce je do značné míry dána rozhraním jádra. Základní popis činnosti uvedený v předchozí kapitole bude nyní doplněn informacemi o způsobu alokace DMA paměti.

Pro podrobnější popis návrhu ovladače je použit jazyk *UML*. Konkrétně jde o diagram případů užití (viz příloha B), anglicky *USE CASE diagram*, vytvořený v nástroji *bouml*.

4.2.1 Mapování DMA paměti

V hlavní paměti budou používány následující datové struktury - v jejich seznamu je zároveň uveden typ alokace s krátkým zdůvodněním volby:

- Rx/Tx Ring. Použije se koherentní mapování, neboť jsou vytvářeny v době zavedení modulu, respektive získání zařízení, a přetrvávají v paměti po celý život modulu jádra.
- `sk_buff` pro příjem paketů. Vhodné je proudové mapování se směrem `PCI_DMA_FROMDEVICE`. Instance jsou alokovány jen dočasně pro jednosměrný přenos z karty do hlavní paměti.
- `sk_buff` pro odeslání paketů. Analogicky k předchozímu, proudové mapování se směrem `PCI_DMA_TODEVICE`. Instance jsou alokovány jen dočasně pro jednosměrný přenos z hlavní paměti do karty.
- Aktualizace počtu odeslaných paketů. Proudové mapování se směrem `PCI_DMA_FROMDEVICE`. Přenosy se dějí pouze ve směru z karty do hlavní paměti. O okamžiku změny obsahu je ovladač informován přerušením a může tak vždy provést synchronizaci obsahu paměti pro procesor.

4.3 Blok řízení DMA přenosů

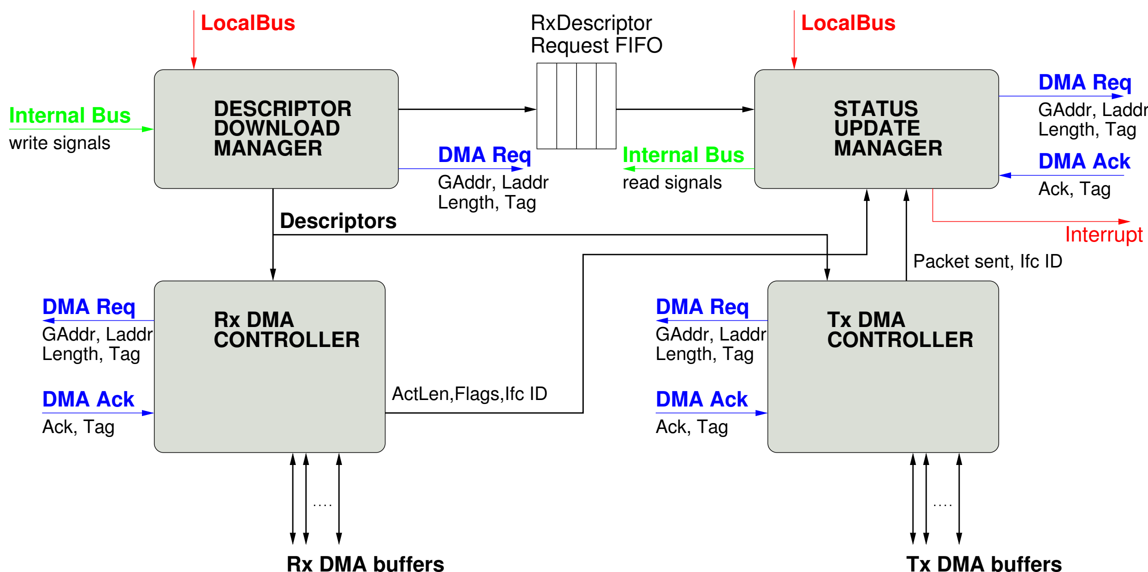
Návrh obvodů pro řízení DMA přenosů do značné míry vychází ze zkušeností získaných při tvorbě speciálního rozhraní pro přenos dat (viz část 3.4). Jako zbytečné plýtvání omezenými zdroji čipu FPGA se ukázala násobná replikace bloků realizujících řízení DMA přenosů. Paralelně pracující vzájemně nezávislé bloky sice dosahují vysoké teoretické propustnosti, ta je ale v konečném důsledku limitována propustností interní sběrnice (viz část 4.1.4).

Jádro návrhu se opírá o replikaci datových položek potřebných pro zpracování jednoho síťového rozhraní a vytvoření společné řídící logiky, která periodicky obsluhuje jednotlivá rozhraní. Jedná se o časový multiplex na místo prostorového. I přes časový multiplex lze stále dosáhnout dostatečné propustnosti pro plné vytížení interní sběrnice. Nesporným přínosem je snížení množství zabrané logiky FPGA čipu.

Navrhovaný hardware, na obrázku 3.2 se jedná o blok označený DMA řadiče, bude dále rozdělen na čtyři spolupracující komponenty. Rozdělení přehledně ukazuje obrázek 4.7.

Hlavní komponenty budou popsány dále, nyní následuje popis bloku na obrázku 4.7 označeného **RxDescriptor Request FIFO**. V obrázku 4.7 byly zavedeny anglické názvy pro jednotlivé bloky. Ty vystihují funkci komponent, a proto budou použity v dalším textu.

RxDescriptor Request FIFO Pro každý přijatý paket se přenáší do paměti aktualizovaný deskriptor. Abychom znali správnou adresu, kam deskriptor nakopírovat, jsou poslední vygenerované požadavky na stažení těchto deskriptorů uloženy do dočasné paměti, odkud jsou v okamžiku potřeby vyzvednuty a je dopočítána cílová adresa v hlavní paměti.



Obrázek 4.7: Rozdělení bloku řízení DMA přenosů do subkomponent

4.3.1 Descriptor download manager

Komponenta (viz obrázek C.1) obstarává přísun deskriptorů Rx a Tx DMA řadičům. Činnost kontroluje ovladač pomocí sady registrů vyhrazených pro každý kanál. Na obrázku označeny **CONTROL**. Pokud je daný kanál aktivní, konečný automat obsažený v bloku **NEXT_DESC** cyklicky sleduje stav deskriptorů. Signalizuje-li blok **NFIFO** nedostatek deskriptorů a zároveň obsah registrů **HEAD** a **TAIL** indikuje dostupnost deskriptorů v příslušném **Ringu**, vydá povel na jejich stažení.

Na průběh zápisu nových deskriptorů dohlíží blok **WE_LOGIC**. Konec zápisu bloku deskriptorů vyvolá změnu obsahu registrového pole **NEXT_DESC** a zároveň zápis do **RxDescriptor Request FIFO**.

NEXT_DESC blok obsluhuje zámek bránící vytvoření opakovaného DMA požadavku pro jeden kanál, zatímco se předchozí provádí, a také ukládá adresu, odkud budou stahovány následující deskriptory.

4.3.2 Status update manager

Po dokončení příjmu či odeslání paketu musí být aktualizován stav **Ringů**. V případě Rx práci obstarává z části hardware a z části ovladač. Hardware aktualizuje použité deskriptory. K tomu slouží komponenta **NFIFO** (viz obrázek C.2), ve které jsou uloženy skutečné délky přijatých paketů. **RxDESC_UPDATE** cyklicky kontroluje obsah zmíněné komponenty a generuje požadavky na přenos do hlavní paměti. Cílovou adresu získá z **RxDescriptor Request FIFO**.

Pokud obsahoval kopírovaný deskriptor příznak přerušení, poznačí se tento fakt do *tagu* DMA požadavku. **Int_logic** pak zpracovává jak Rx, tak Tx potvrzení o DMA a na základě *tagu* signalizuje přerušení.

Pro směr Tx se udržují pouze počty odeslaných paketů a příznak přerušení. Pokud se změní hodnota některého z čítačů odeslaných paketů v bloku **TxStatus** a je nastaven příznak přerušení, vystaví **TxStatus_UPDATE** DMA požadavek. Adresa v hlavní paměti

zůstává fixní a nastavuje ji v době inicializace ovladač do registru **TxGAddr**. Stejně jako u Rx, v případě přítomnosti se příznak přerušení poznačí v *tagu* DMA požadavku.

Spolu se způsobem aktualizace byl popsán i mechanismus tzv. synchronního přerušení, kdy si ovladač určí, se kterým deskriptorem si přeje být upozorněn. Druhý důvod pro vyvolání přerušení nastává při vypršení časového intervalu. Ten se zavádí, aby nedocházelo ke stárnutí paketů, ale obecně i jiných informací, v hardware. Kontrolu intervalu provádí volně běžící čítače, pro každý kanál jeden, a sada registrů, rovněž vždy jeden pro kanál. Obsah registrů nastavuje ovladač. Pokud volně běžící čítač dosáhne hodnoty nastavené v registru, je signalizováno přerušení. Nulování volně běžících čítačů nastane vždy v okamžiku přijetí nebo odeslání paketu.

4.3.3 Rx DMA controller

Úkolem Rx DMA řadiče (viz obrázek C.3) je vystavovat požadavky na přenos přijatých paketů z odpovídajícího DMA bufferu do hlavní paměti. Činnost řídí konečný automat v bloku označeném **P1**. Cyklicky zpracovává vstupy od DMA bufferů a pokud má pro daný kanál k dispozici deskriptor a zároveň paket, vystaví DMA požadavek. Po jeho odeslání předá parametry požadavku do paměti typu FIFO směrem k bloku **P2** a aktualizuje obsah registru **HwEndPtr**. Tento registr reprezentuje index do DMA bufferu, odkud budou přenášeny data.

P2 čeká na potvrzení DMA přenosu. V okamžiku přijetí potvrzení vyzvedne údaj z fronty od **P1**, vydá povel pro uvolnění paketu z DMA bufferu a odešle potřebné informace do **status update manageru**.

4.3.4 Tx DMA controller

Blok **P1** Tx DMA řadiče (viz obrázek C.4), podobně jako u Rx, cyklicky ověřuje dostupnost deskriptorů. Pokud jsou dostupné, znamená to, že existují pakety, které se mají odeslat. Zkontroluje množství místa v DMA bufferu a vystaví DMA požadavek na přenos. Souběžně předá údaje do fronty, tj. paměti typu FIFO, směrem k **P2** a aktualizuje registr **HwEndPtr**. **HwEndPtr** udává index, kam bude do Tx DMA bufferu zapsán následující paket.

Po přijetí potvrzení o DMA přenosu vyzvedne **P2** údaje z fronty, informuje DMA buffer o připraveném paketu a předá informaci o odeslaném paketu do **status update manageru**. DMA buffer po uvolnění paketu z paměti informuje řadič, což se projeví aktualizací registru **HwStrPtr**.

Kapitola 5

Popis realizace

Následující kapitola obsahuje popis realizace jak softwarové, tak hardwarové části diplomové práce. Vysvětluje volby provedené v době implementace práce a zároveň jsou odvozeny důsledky těchto rozhodnutí. Nezbytnou součástí popisu realizace je zhodnocení výkonnosti ve formě rozboru implementace, kde je nejen podán rozbor, ale i náměty pro možná budoucí vylepšení.

5.1 Software

Pro implementaci ovladače je důležité jednoznačně stanovit PCI identifikaci zařízení (viz část 3.1.1). Pro síťovou kartu na platformě COMBOv2 jsou použity hodnoty uvedené v příkladu v části 3.1.1.

Implementační jazyk jádra operačního systému Linux je jazyk C, stejně tak musí být použit pro implementaci ovladače. Ovladač je vytvořen jako samostatný modul, který pro svou správnou funkci vyžaduje, aby současně s ním byly zavedeny moduly *combo6core* a *combov2*, jak je uvedeno v části 3.5.

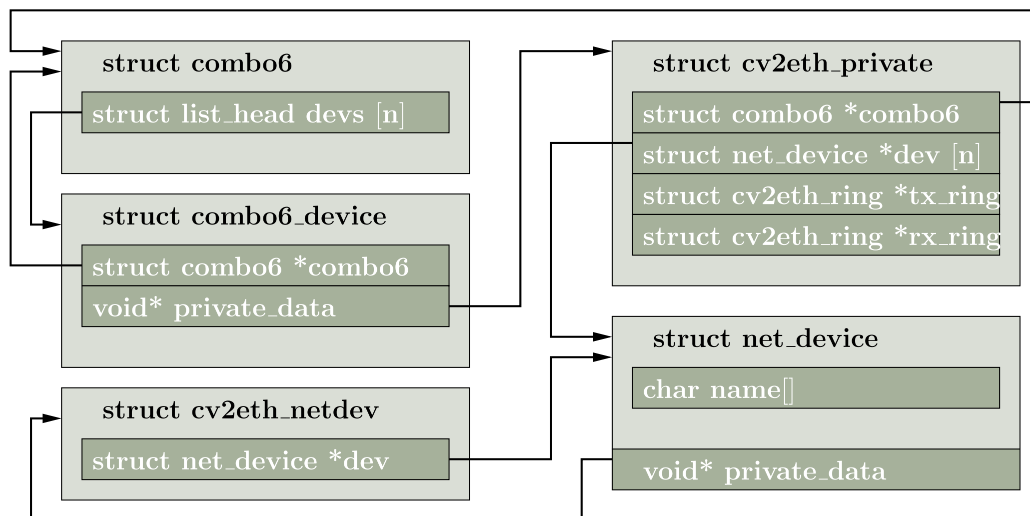
5.1.1 Organizace datových struktur

V předchozí kapitole byl popsán návrh organizace paměti a také činnost ovladačem vykonávaná. Během realizace musí být návrh převeden do programové podoby. U organizace paměti to znamená navrhnout takovou strukturu datových typů, aby efektivně pokrývala důležitou funkcionalitu. Vytvořené struktury a jejich vztah k datovým typům definovaným v core vrstvě je znázorněn na obrázku 5.1.

Na obrázku 5.1 jsou uvedeny klíčové datové struktury a vybrané prvky těchto struktur. Struktury `combo6` a `combo6device` jsou definovány v core vrstě ovladačů *COMBO*. Struktura `net_device` je definována v jádře OS Linux. Zbylé dvě struktury jsou definovány ve vytvářeném modulu:

- `cv2eth_private` udržuje společné informace potřebné pro implementaci síťového ovladače,
- `cv2eth_netdev` udržuje informace specifické pro konkrétní rozhraní síťové karty realizované na platformě COMBOv2.

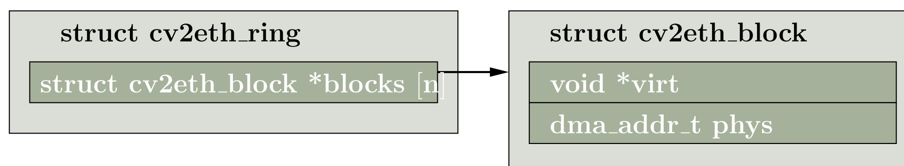
Poznámka k obrázku 5.1: **[n]** označuje lineární seznam, či pole prvků.



Obrázek 5.1: Vztah datových struktur s vrstvou core

Důvodem proč jsou ukazatele na kruhové buffery umístěny ve struktuře se společnými údaji je, že alokaci prostoru pro popis bufferů je paměťově efektivnější provést hromadnou alokací. Pokud by byl prostor alokován ve více krocích a velikost alokovaného místa nebyla zarovnána na velikost stránky, došlo by k nenávratné ztrátě paměti.

Kruhové buffery jsou reprezentovány dvojicí datových struktur zachycených na obrázku 5.2. Celý buffer se skládá z jednotlivých bloků svázaných do lineárního seznamu. Každý blok je reprezentován instancí struktury `cv2eth_block`. Ta, jak je vidět na obrázku, obsahuje dvě položky. Jsou to virtuální adresa bloku a fyzická adresa téhož bloku. Právě tato fyzická adresa se předává do zařízení.

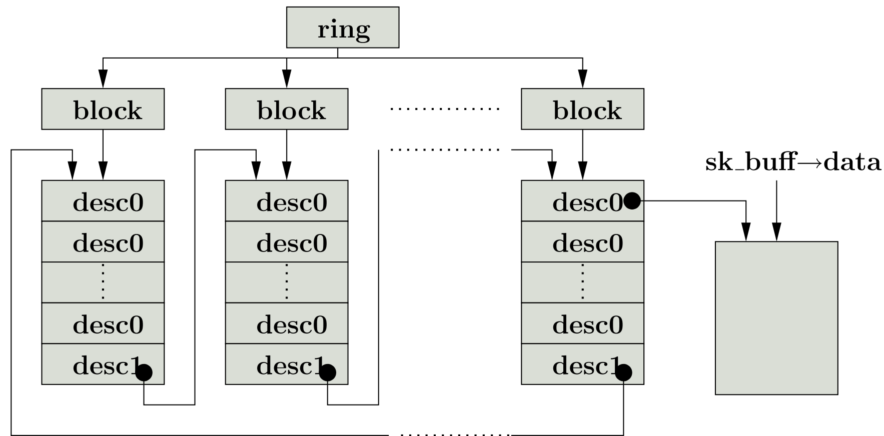


Obrázek 5.2: Datové struktury tvořící *Ring*

Výsledná datová struktura má podobu jako na obrázku 5.3. Každý blok kruhového bufferu obsahuje deskriptory ukazující na paketové buffery (**desc0**) a také deskriptory ukazující na další blok kruhového bufferu (**desc1**).

5.2 Hardware

V zadání diplomové práce jsou nabídnuty dvě možnosti volby implementačního jazyka pro popis hardware: Handel-C a VHDL. Z předchozích zkušeností při tvorbě řadičů pro speciální síťové rozhraní *szedataII* vyplynulo, že Handel-C je jazyk vhodný pro rychlé prototypování a tvorbu složitých řídících automatů. Nevýhodou tohoto jazyka je nižší úroveň kontroly nad výsledným hardware.



Obrázek 5.3: Výsledná organizace *Ringu*

Naopak jazyk VHDL je nízkoúrovňový a umožňuje vysokou kontrolu nad výsledkem procesu syntézy. Tato kontrola je vykoupena větším množství textu potřebného k popisu stejné funkcionality jako v jazyce Handel-C.

Dalším důležitým faktem pro zvolení implementačního jazyka je, že zbytek platformy NetCOPE je realizován v jazyce VHDL. I přesto, že je možné oba jazyky provázat a komponenty spojit do jednoho celku, vznikají na rozhraní mezi takovýmito komponentami problémy. Konkrétním příkladem jsou generické parametry jazyka VHDL sloužící pro parametrizaci entit. Handel-C naopak pro podobné účely používá direktivy preprocesoru, například `#define`. Problémem je, že neexistuje standardní přenositelné rozhraní umožňující převod mezi těmito vyjadřovacími prostředky jednotlivých jazyků.

Pro implementaci hardware byl tedy zvolen jazyk VHDL. Přínos je zřejmý především v přesné kontrole nad časováním jednotlivých operací a kritickými cestami. Toto je samozřejmě možné i v jazyce Handel-C, ale je potřeba mít na paměti, jak programátor zapisuje jednotlivé konstrukce. Obdobně jako je tomu u běžných programovacích jazyků. Analogií jsou jazyk C a jazyk symbolických instrukcí.

Pozitivní dopad na výsledné množství spotřebované logiky má použití předpřipravených VHDL komponent, které buďto přímo využívají základních stavebních prvků na čipu, nebo jsou na tyto prvky mapovány optimálním způsobem.

Výsledky procesu syntézy bloku řízení DMA přenosů jsou shrnuty v tabulce 5.1. Je nezbytné dodat, že pro syntézu byl jako cílový zvolen FPGA čip firmy *Xilinx Virtex5* v provedení *VLX110T*. Generickým parametrem byly zvoleny čtyři síťová rozhraní a syntéza byla provedena programem *XST* ve verzi 10.1.03 pro Linux.

Typ zdroje	Zabrané množství	Celkové množství	Procentuální vyjádření
registry	1669	69 120	2%
LUT	4623	69 120	6%

Tabulka 5.1: Množství zdrojů zabraných blokem pro řízení DMA přenosů

Zkratka LUT, z anglického *look-up table*, označuje paměťovou strukturu adresovanou vstupními signály, která se dá použít buďto jako paměť s jednobitovým výstupem, nebo k realizaci funkce kombinační logiky.

Maximální dosažitelná pracovní frekvence samotného bloku řízení byla nástrojem *XST* stanovena na 159,792 MHz. Toto je výborný výsledek zejména při srovnání s obdobným hardware vytvořeným v jazyce Handel-C, kde se maximální pracovní frekvence pohybuje pod hranicí 130 MHz.

Výsledky syntézy celé síťové karty se čtyřmi rozhraními obsahuje tabulka 5.2.

Typ zdroje	Zabrané množství	Celkové množství	Procentuální vyjádření
registry	17 435	69 120	25%
LUT	20 163	69 120	29%
Block RAM	30	148	20%

Tabulka 5.2: Množství zdrojů zabraných síťovou kartou

5.3 Rozbor implementace

Celá práce se zabývá návrhem a implementací vysokorychlostní síťové karty, proto tato část obsahuje rozbor výkonnosti softwarové i hardwarové části.

5.3.1 Software

V případě software se jedná o událostmi řízený ovladač, jehož činnost do značné míry závisí na použitém počítači, verzi jádra operačního systému a jeho nastavení. Lze tedy jen obtížně provést přesný rozbor časové náročnosti jednotlivých operací prováděných při zpracování paketu. Ať už vysílaného nebo přijímaného.

Vždy však postačí provést určení časové složitosti jednotlivých použitých algoritmů či funkcí. Důležité je si uvědomit, že úkolem ovladače zařízení je zprostředkovat přístup k hardwarovým zdrojům a jako takový by tedy neměl provádět s daty žádné složité výpočty. Naopak množina úkolů připadajících ovladači musí být minimální. Každá operace z této množiny v ideálním případě nepřekročí konstantní časovou složitost. V nejhorším případě lineární.

Tvrzení o mezích časové složitosti vyplývá z úvahy nad typem prováděných operací. Rozlišit se dají dva typy:

1. Inicializace. Sem patří veškeré operace týkající se registrace ovladače, vytvoření síťových rozhraní atd. Děje se omezeně krát bez závislosti na množství přijatých, odeslaných paketů. Očekávaná časová složitost je konstantní.

2. Zpracování paketu. Jedná se především o vytvoření a zrušení datových míst pro uložení paketu. Každá jednotlivá operace této skupiny se provádí s konstantní časovou složitostí, ale s každým paketem. V součtu lineární časová složitost.

5.3.2 Hardware

Z pohledu hardware se jedná o hodinami řízený výpočetní stroj implementovaný vždy v konkrétním FPGA čipu. Tady lze prozkoumat časovou náročnost až do úrovně počtu potřebných hodinových impulzů, které zkonzumuje daná operace. Zde může nastat, zejména v případě pomalého software, zpomalení až úplné zablokování. Význam má tedy analyzovat pouze ideální dosažitelný stav.

Bloková schémata v příloze C zachycují tok dat v hardware. Chybí zde však informace v jakých okamžicích a za jakých okolností jsou data mezi jednotlivými entitami předávána. Běh, výměnu a transformaci dat řídí konečné automaty. Ve schématech označeny FSM, z angl. *finite state machine*. Každá ze čtyř základních komponent obsahuje vždy alespoň jeden konečný automat, které budou nyní zevrubně popsány spolu s analýzou časové náročnosti prováděných operací a jejich dopadem na celkovou výkonnost systému.

V každé z níže popsaných komponent slouží alespoň jeden konečný automat ke generování DMA požadavků. Způsob odesílání požadavků je společný a je optimalizován tak, aby zabíral co nejméně zdrojů čipu, ale zároveň byl dostatečně rychlý. Celý požadavek má velikost 128 bitů. Nejprve je tedy ve dvou krocích zapsán do paměti o datové šířce 64 bitů, odkud je odeslán dál ke zpracování. Zápis DMA požadavku je řízen automaty a vždy zabere dva takty.

Při rozboru výkonosti konečných automatů není uvažováno zpoždění způsobené nedostatečnou propustností interní sběrnice. Uvažuje se tedy sběrnice o neomezené rychlosti. Analýza slouží ke zjištění maximálních možných rychlostí a dílčích parametrů potřebných k dosažení této rychlosti. Případně ukáže způsoby a prostředky jak výkonnost zvýšit.

Komponenta DDM

Komponenta DDM, nebo-li **D**escriptor **D**ownload **M**anager, obsahuje dva konečné automaty:

1. **WE_LOGIC** - jak již bylo uvedeno v části 4.3, stará se o korektní zápis nově příchozích deskriptorů a aktualizuje potřebné synchronizační příznaky a ukazatel na následující deskriptor.

   Konečný automat nevkládá žádné čekací stavy a všechny příchozí zápisové transakce obsluhuje v témže taktu. Výkonnost tedy nijak neomezuje.

2. **NEXT_DESC** - jednoduchý konečný automat. Pokud jsou splněny všechny přechodové podmínky, sestaví v jednom kroku DMA požadavek. Zápis do výše zmiňované paměti trvá dva takty. Po zapsání čeká na převzetí požadavku.

   Jeden běh automatu pro vytvoření požadavku trvá osm taktů. Jeden takt výpočet, dva takty zápis do paměti, pět taktů převzetí. Doba potřebná k převzetí požadavku závisí na datové šířce, kterou je požadavek vyčítán z paměti. Uvažována bude datová šířka 32 bitů, neboť odpovídá rozdělení datové šířky 128 bitů pro DMA požadavek mezi čtyři hlavní komponenty. Doba převzetí je tedy 128/32 plus jeden režijní takt.

   Tyto DMA požadavky slouží ke stažení deskriptorů, které probíhají po blocích. Bude-li jeden blok obsahovat čtyři deskriptory, je celková velikost přenášeného bloku deskriptorů $4 * 16 = 64\,\mathrm{B}$.

   Jeden požadavek na deskriptory z předchozího příkladu obslouží čtyři pakety. Doba potřebná pro vytvoření požadavku je 8 taktů. Při pracovní frekvenci 125 MHz a počtu čtyř deskriptorů na blok lze vygenerovat dostatek požadavků pro obsloužení 62,5 milionů paketů za sekundu, zkráceně $Mpaket/s$. Zvětšením počtu deskriptorů v bloku může být rychlost dále vylepšována podle předpisu:

$$N = \frac{pdb * f}{pt} \tag{5.1}$$

Kde N je požadovaná výkonnost v počtu paketů za sekundu, pdb počet deskriptorů přenášených v jednom bloku, f pracovní frekvence a pt počet hodinových taktů potřebných k vystavení požadavku, tj. 8.

Důležité je poznamenat, že celý výpočet uvažuje pouze jeden kanál jedním směrem. Pokud bude použito více síťových rozhraní, rozdělí se výkonnost rovným dílem mezi všechna rozhraní a směry. To vyplývá z principu časového přepínání obsluhy mezi rozhraními.

Komponenta SUM

Komponenta SUM, nebo-li **S**tatus **U**pdate **M**anager, obsahuje rovněž dva konečné automaty. Vzájemný vztah těchto automatů je však komplikovanější než v předchozím případě. Oba slouží k vytváření DMA požadavků. Pro pochopení funkce je důležité uvědomit si rozdíl mezi příjmem a odesíláním paketů popsaným v části 4.1.4. Ve zkratce, pro příjem paketů je nezbytné posílat aktualizované deskriptory zpět do hlavní paměti, pro vysílání postačí aktualizovat počet odeslaných paketů.

1. **RxDESC_UPDATE** - vytváří DMA požadavky pro přenos aktualizovaných deskriptorů zpět do hlavní paměti. Prakticky se jedná o identickou činnost jako v případě automatu **NEXT_DESC**. Rozdílný je směr přenosu.

   Drobná odlišnost je i v době potřebné k vytvoření DMA požadavku, která spočívá v přidaném přechodu, který slouží k vyzvednutí informací o cílové adrese v hlavní paměti z komponenty **RxDescriptor Request FIFO** (viz obrázek 4.7). Doba provedení se tedy může v závislosti na připravenosti dat v uvedené komponentě prodloužit.

   Jednoduchou úvahou lze ověřit, že data budou v paměti fifo vždy připravena a přidané zpoždění bude právě jeden takt, během něhož jsou data z paměti vyčítána. Úvaha je taková, že data se zapisují do paměti fifo právě v okamžiku odeslání požadavku automatem **NEXT_DESC**. Paměť fifo pak potřebuje alespoň jeden takt, než budou data připravena na výstupu. Než-li však informace o přijatém paketu dorazí do komponenty SUM, musí být vytvořen požadavek na přenos paketu v Rx dma řadiči. Součástí vystavení požadavku, jak bylo popsáno v úvodu této části, jsou vždy dva takty, kdy je kopírován do paměti.

   Vztah pro propustnost je identický (viz rovnice 5.1), až na hodnotu jmenovatele pt, která je rovna 9. Propustnost dosažitelná za shodných parametrů je tedy 55,5 Mpaket/s.

   Pokud bude použito souběžně více rozhraní, provádí se časové přepínání obsluhy, a proto se propustnost rovnoměrně rozdělí mezi počet rozhraní. Ne však mezi oba směry. Pokud tedy bude mít síťová karta 2 rozhraní, propustnost se sníží dvakrát. Důvodem je asymetričnost směru Tx, kde se přenáší pouze počet paketů.

2. **TxSTATUS_UPDATE** - vytváří požadavky pro přenos počtu odeslaných paketů do hlavní paměti. Pro každé rozhraní je k dispozici 32-bitový čítač, jehož hodnota musí být zkopírována. Parametry pro DMA požadavek jsou dány předem. Globální adresa nastavena v registru, délka určena v době překladu, stejně tak lokální adresa, jako i směr přenosu a značka. Navíc s ohledem na použití čítačů je výsledné chování agregující, tj. pro přenos je nedůležité jestli se předává informace o jednom odeslaném paketu, nebo o tisících.

Pokud lze ukázat, že automat má možnost pravidelně vystavovat DMA požadavky, není třeba dále analyzovat propustnost, právě s ohledem na agregaci počtu paketů. Nestárnutí, tj. možnost odesílat data, je zřejmá ze způsobu přidělování času obsluhy. Jedná se o implementaci cyklické obsluhy s výzvou. Po řadě dostává možnost pracovat automat **RxDesc_Update** pro každé rozhraní a následně je jedna výzva přidělena pro **TxSTATUS_UPDATE**. Podíl přiděleného času je tedy počet rozhraní ku jedné ve prospěch Rx, které přenáší výrazně větší objem dat a nemá možnost agregace.

Pro úplnost příklad propustnosti, kterou je schopen automat **TxSTATUS_UPDATE** obsloužit. Počet taktů pro vystavení požadavku je 8, pro výpočet lze použít upravenou rovnici 5.1:

$$N = \frac{ppp * f}{pt} \tag{5.2}$$

Význam proměnných je zachován, počet deskriptorů v bloku je nahrazen proměnnou ppp udávající počet přenesených paketů. Tato proměnná může nabývat hodnot od 1 do $2^{32}-1$. Tomu tedy odpovídá i propustnost od 15,6 Mpaket/s až řádově 10^{16} Mpaket/s. Je evidentní, že nijak analyzovaný systém neomezuje.

Komponenta Rx DMA controller

Rx DMA řadič obsahuje jeden konečný automat, který vytváří požadavky na přenos přijatých paketů do hlavní paměti. Tento automat, pracovně označen **NEW_PACKET**, spadá do bloku **P1** (viz obrázek C.3). Automat řídí výpočet potřebných údajů pro DMA požadavek, který sestavuje do pomocné paměti. Opět tedy celková doba sestává z výpočtu délky, sestavení v paměti a následného předání zbylé části designu. Celý cyklus trvá 9 taktů.

Výpočet výkonnost, které dokáže konečný automat v aktuálním zapojení dosáhnout, se provede podle vzorce:

$$N = \frac{f}{pt} \tag{5.3}$$

Pracovní frekvence bude uvažována stejná jako v předcházejícím textu, tj. 125 MHz a počet taktů pt je 9. Výsledná výkonnost $N = 13,88$ Mpaket/s. Pro dosažení vyšší rychlosti lze zkrátit dobu potřebou pro předání požadavku. Samotný výpočet potřebných údajů trvá pouhé 3 takty. Pokud by byl požadavek předán v celé své datové šířce paralelně, zmenšil by se počet potřených taktů a dosažitelná výkonnost se ztrojnásobila.

Komponenta Tx DMA controller

Tx DMA řadič je přímou analogií Rx řadiče a rovněž obsahuje jeden konečný automat, který vytváří požadavky na přenos, tentokrát ovšem odesílaných paketů z hlavní paměti do hardwarového bufferu karty. Opět je automat pracovně označen **NEX_PACKET** a je realizován v bloku **P1** (viz obrázek C.4). Pro odeslání DMA požadavku je nezbytné vypočítat potřebné údaje, jako jsou délka přenášených dat a lokální adresa určující pozici v hardwarovém bufferu.

Globální adresa je dána pevně obsahem deskriptoru, směr přenosu a značka jsou konstanty známé již v době překladu.

Doba sestavení a odeslání DMA požadavku je opět 9 taktů a pro výpočet tedy platí vztah 5.3. Dosažitelná výkonnost s hodnotami shodnými s předchozím výpočtem je rovněž $P = 13,88$ Mpaket/s.

K dosažení vyšší propustnosti lze opět zkrátit dobu potřebnou k odeslání DMA požadavku, neboť samotný výpočet potřebných údajů trvá pouhé tři takty. V optimálním případě lze tedy dosáhnout trojnásobné propustnosti při zachování pracovní frekvence.

Poznatky získané analýzou

Komponenty řídící přenos deskriptorů, tj. DDM a SUM, nejsou, podle očekávání, kritickým místem. Propustnost dosažitelná v rámci omezení kladených těmito komponentami je snadno škálovatelná úpravami popsanými výše.

Komponenty řídící samotný přenos paketů jsou výkonnostně kritické a určují maximální dosažitelnou propustnost. Zvýšení propustnosti je možné pouze za cenu změny hardwarové realizace, nárůstu spotřebovaných hardwarových zdrojů a zvýšení složitosti designu.

Kapitola 6

Ověření funkčnosti

Po provedení implementace je nezbytné ověřit její správnost. Ověření funkčnosti je proto věnována tato kapitola.

6.1 Software

Prvním krokem je otestovat funkcionalitu ovladače vztahující se k registraci zařízení a přepínání designu. Toto je umožněno díky PCI identifikaci karty, podle níž se zavádějí potřebné moduly do jádra operačního systému.

V reálu bylo tedy úspěšně odzkoušeno zavádění a odstraňování modulu z jádra. Při zavádění modulu se provádí registrace PCI zařízení a bylo tedy ověřeno, že tato obsluha rovněž pracuje správně, neboť ovladač získal přístup ke kartě. V tomto kroku se také provádí registrace funkčně specifického ovladače u modulu core vrstvy, tj. *combo6core*. Ta rovněž pracuje správně a pro ověření slouží následující výpis programu `csid -s`:

```
1.  Board     : combov2
2.  Addon     : unknown
3.  Chip      : unknown
4.  LAN ports: 4/4 (RX/TX)
5.  Firmware : ok
6.  SW        : 0x41c10600
7.  HW        : 0x00050000
8.  Text      : NIC_NetCOPE
9.
10.  Device [combo6] cv2eth
11.   (0x41c10600-0x41c106ff) {}: active
```

Testování bylo provedeno na stroji *medoc.liberouter.org* vybaveného kartou *COMBOv2*. Pro ověření správnosti registrace ovladače jsou důležité zejména řádky 10 a 11 předchozího výpisu. Ty ukazují, jaký funkčně specifický modul je k dispozici, a také jestli je schopen pracovat s aktuálním hardware (položka *active*).

Součástí zaváděného designu je i PCI identifikace, která je v případě testovaného hardware vidět na následujícím výpisu příkazu `lspci -v -s 08:00 -n`:

```
1. 08:00.0 Class 0200: 18ec:c032
2.         Subsystem: 18ec:0100
```

```
3.         Flags: bus master, fast devsel, latency 0, IRQ 222
4.         Memory at dc000000 (64-bit, non-prefetchable) [size=1M]
5.         Memory at d8000000 (64-bit, non-prefetchable) [size=64M]
6.         Memory at dc100000 (32-bit, non-prefetchable) [size=1M]
7.         [virtual] Expansion ROM at dc300000 [disabled] [size=1M]
8.         Capabilities: <available only to root>
```

Identifikaci ukazují řádky 1 a 2 a lze vidět, že odpovídá příkladu uvedenému v části 3.1.1. Stejné údaje je možné extrahovat i ze souborů umístěných v adresářové struktuře s kořenem */sys/bus/pci/devices*.

Finálním krokem je ověření dostupnosti vytvářených síťových rozhraní a test odeslání a příjmu paketů. Nová rozhraní ukazuje výstup programu `ifconfig -a`:

```
cv2eth00  Link encap:Ethernet  HWaddr 00:11:17:FF:FF:00
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

cv2eth01  Link encap:Ethernet  HWaddr 00:11:17:FF:FF:01
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

cv2eth02  Link encap:Ethernet  HWaddr 00:11:17:FF:FF:02
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

cv2eth03  Link encap:Ethernet  HWaddr 00:11:17:FF:FF:03
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

Pro ověření funkčnosti odesílání a příjmu paketů posloužil program `tcpreplay`. Na kombinaci testovacích strojů *medoc.liberouter.org* a *morgon.liberouter.org* zapojených proti sobě. Výsledek testovacích přenosů ukazuje opět výpis příkazu `ifconfig`:

```
cv2eth00  Link encap:Ethernet  HWaddr 00:11:17:FF:FF:00
          inet addr:10.0.0.1  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::211:17ff:feff:ff00/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```
RX packets:5 errors:0 dropped:0 overruns:0 frame:0
TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:378 (378.0 b)  TX bytes:468 (468.0 b)
```

Z výpisu je patrné, že bylo odesláno pět paketů a přijato šest.

6.2 Hardware

Prvním krokem při ověřování správnosti hardware je simulace popisu v jazyce VHDL. Pro tento účel byl použit program *ModelSim SE* ve verzi 6.5 od společnosti *Mentor Graphics*. Simulace proběhly jak nad jednotlivými stavebními bloky DMA řadičů, tak nad celou síťovou kartou. Simulace odhalily několik implementačních chyb, následně odstraněných, a prokázaly funkčnost implementace. Stejně tak posloužily k ověření výsledků analýzy výkonnosti popsané v části 5.3.2.

Problémem funkční simulace je časová náročnost jejího provedení a obvykle tedy slouží pouze pro ověření základní funkčnosti. Pokročilejší prostředkem pro ověření správnosti implementace před jejím převedením do hardware je funkční verifikace. Obvykle vytvořená tak, že realizuje řadu možných průběhů činnosti testovaného obvodu. Výhodou je autonomnost provádění testu, které řídí samotné verifikační prostředí a dokáže pokrýt více přípustných stavů, než pouhá funkční simulace.

Verifikace implementovaných řadičů je ve fázi příprav.

Ověření funkčnosti v reálném provozu bylo provedeno společně s testem software. Zátěžové testy prozatím neprošly, nicméně na nalezení a odstranění chyb se pracuje. Také je důležité poznamenat, že síťovou kartu netvoří pouze komponenty a ovladač implementovaný v rámci diplomové práce, ale celá řada dalších hardwarových a softwarových komponent. Ty samozřejmě rovněž tvoří potenciální zdroj problémů.

Kapitola 7

Závěr

Cílem diplomové práce bylo navrhnout a realizovat softwarové a hardwarové prostředky pro přenos paketů prostřednictvím standardního síťového rozhraní jádra operačního systému Linux. Proto bylo nejprve třeba nastudovat problematiku počítačových sítí, metodiku implementace hardware a ovladačů zařízení. Teoretické poznatky získané v průběhu přípravy na realizaci diplomové práce doplnit znalostmi konkrétní platformy, pro niž jsou ovladač a DMA řadiče určeny. Tyto poznatky jsou shrnuty v úvodních kapitolách práce.

Na teoretických základech je postaven návrh, jehož cílem bylo dosažení špičkové propustnosti 10 Gb/s. V průběhu návrhu byla také provedena analýza propustnosti systému, která ukázala, že pro dosažení této rychlosti je nezbytná dostatečně kvalitní infrastruktura na čipu FPGA, především pak interní sběrnice. Podle návrhu byla následně provedena implementace, jejíž rozbor lze nalézt v textu. Výsledky rozboru ukázaly, že současná implementace je schopná zpracovat více než 13 milionů paketů za vteřinu! Z analýzy také vyplynula další možná vylepšení propustnosti.

Posledním krokem bylo ověření funkčnosti celé práce na cílové platformě, které odhalilo problémy při komunikaci mezi hardware a software na vyšších rychlostech. Avšak po odstranění současných nedostatků a vyladění zbývajících parametrů systému bude možné provést výkonnostní měření v reálném provozu.

Závěrem je důležité říct, že cíle práce byly dosaženy a prototyp byl úspěšně otestován na kartě *COMBOv2*.

Literatura

[1] Linux kernel source code. WWW, [Online],[ver. 2.6.30-rc4],[cit. 2009-05-06].
URL <http://www.kernel.org/pub/linux/kernel>

[2] Net:NAPI. WWW, [Online],[cit. 2008-12-18].
URL <http://www.linuxfoundation.org/en/Net:NAPI>

[3] Počítačová síť. WWW, [Online],[cit. 2008-12-12].
URL <http://cs.wikipedia.org/wiki/Pocitacova_sit>

[4] Wavelength-divison multiplexing. WWW, [Online],[cit. 2008-12-12].
URL <http://en.wikipedia.org/wiki/Wavelength-division_multiplexing>

[5] Corbet, J.; Rubini, A.; Kroah-Hartman, G.: *Linux Device Drivers*. O'Reilly, třetí vydání, Únor 2005, ISBN 0-569-00590-3.

[6] Dovet, D. P.; Cesati, M.: *Understanding the Linux kernel*. O'Reilly, třetí vydání, Únor 2008, ISBN 0-569-00565-2.

[7] Kuznetsov, A.; Salim, J. H.; R., O.: NAPI Howto. WWW, [Online],[cit. 2009-01-01].
URL <http://www.cookinglinux.org/pub/netdev_docs/napi-howto.php3>

[8] Kállay, F.; Peniak, P.: *Počítačové sítě a jejich aplikace*. Praha: Grada, první vydání, 1999, ISBN 80-7169-407-X.

[9] Martínek, T.: Specifikace interní sběrnice, interní dokumentace projektu Liberouter, [cit. 2008-12-16].

[10] Matz, M.; Hubička, J.; Jaeger, A.; aj.: *System V Application Binary Interface: AMD64 Architecture Processor Suplement*. [Online],[ver. 0.99],[cit. 2008-12-20].
URL <http://www.x86-64.org/documentation/abi-0.99.pdf>

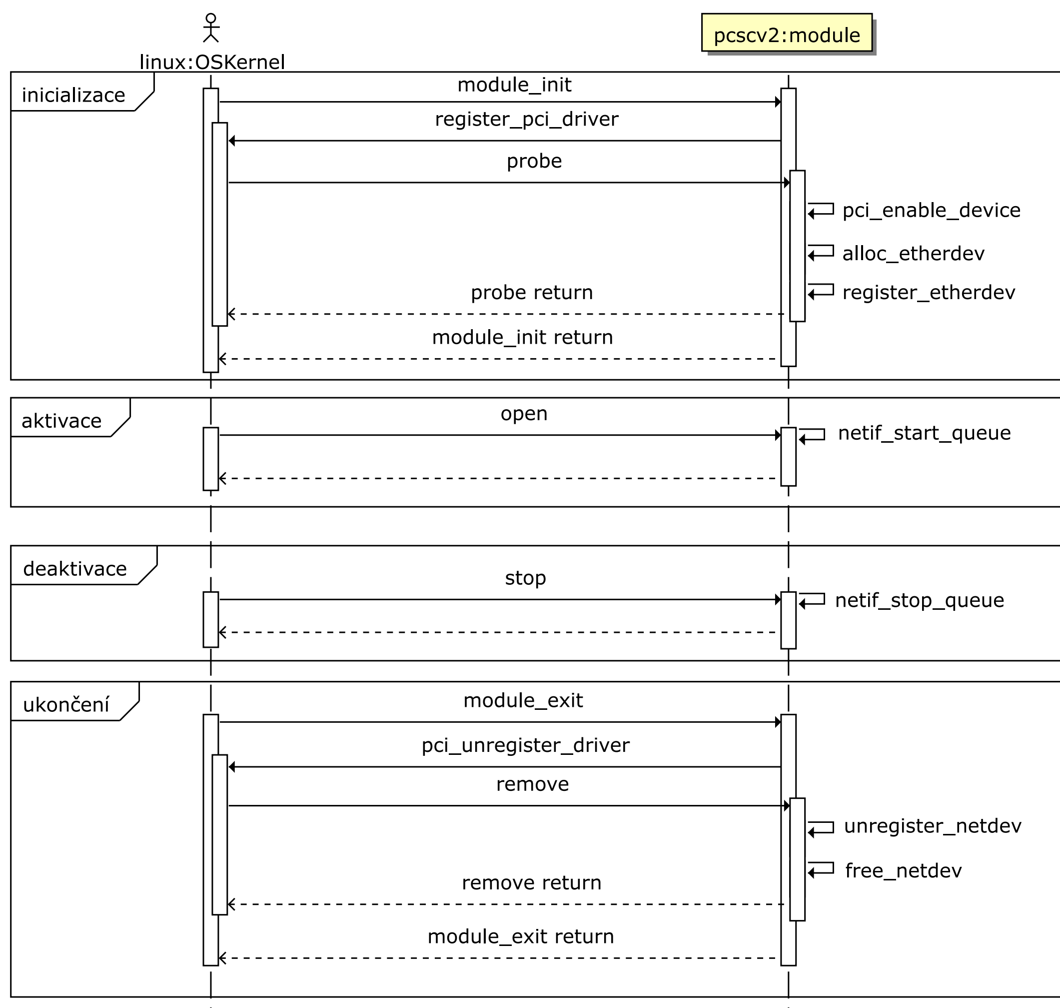
[11] PCI-SIG: *PCI Local Bus Specification*. PCI Special Interest Group, Prosinec 1998, [rev. 2.2].

[12] PCI-SIG: *PCI Express Base Specification*. PCI Special Interest Group, Březen 2005, [rev. 1.1].

[13] Salim, J. H.; Olsson, R.; Kuznetsov, A.: *Beyond Softnet*. USENIX Association, 2001, [Online],[cit. 2008-12-21].
URL <https://www.usenix.net/publications/library/proceedings/als01/full_papers/jamal/jamal.pdf>

Příloha A

Sekvenční diagram modulu

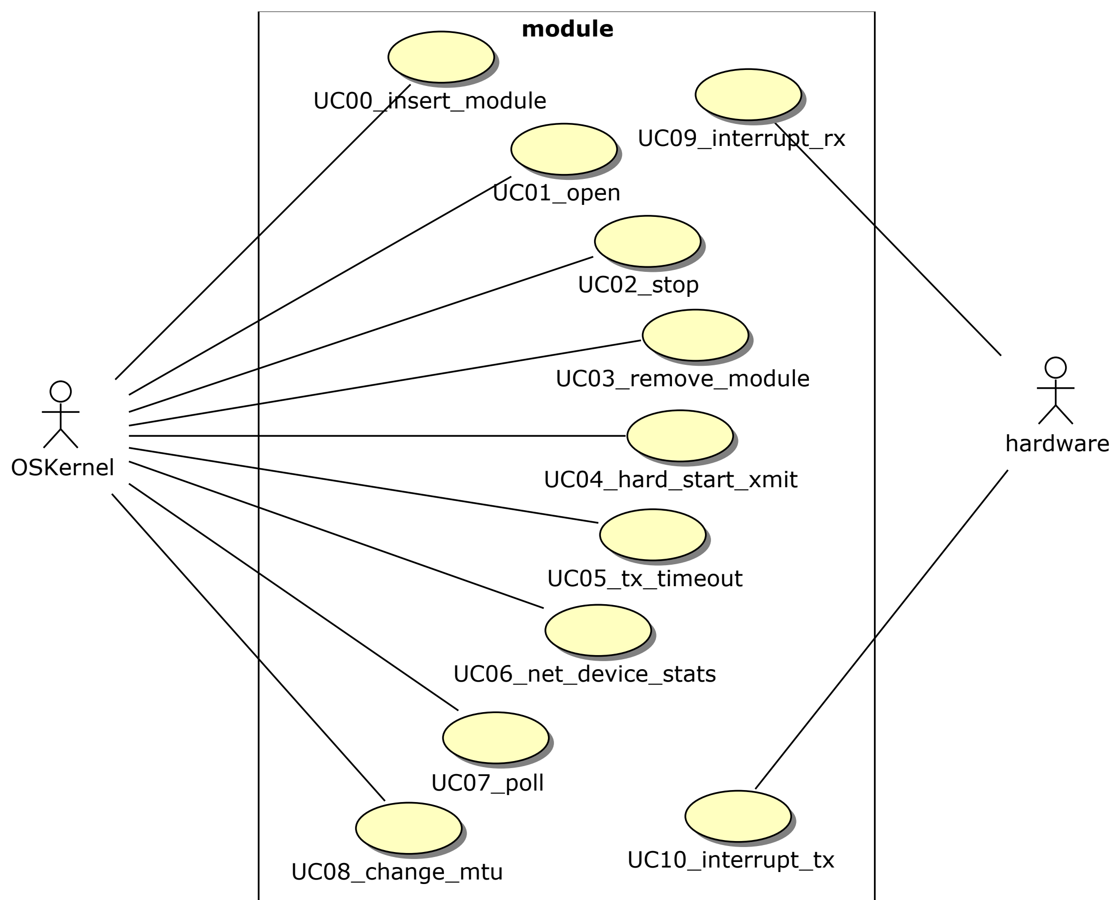


Obrázek A.1: Sekvenční diagram základního životního cyklu modulu jádra

Příloha B

Diagram případů užití

B.1 USE CASE diagram



Obrázek B.1: Diagram případů užití

B.2 Popis případů užití

B.2.1 UC00_insert_module

Podmínka provedení V počítač je připojena odpovídající karta.

Stav po provedení V jádře OS je registrován odpovídající počet síťových rozhraní.

Prováděné kroky

1. Do jádra OS je zaveden modul ovladače.
2. Pro každé rozhraní se alokují **Rx/Tx Ringy**.
3. Rozhraní se zaregistrují v rámci operačního systému.

B.2.2 UC01_open

Podmínka provedení Rozhraní je neaktivní.

Stav po provedení Rozhraní je připraveno pro příjem a odesílání paketů.

Prováděné kroky

1. Je aktivováno vybrané rozhraní.
2. Pro příjem paketů jsou alokovány deskriptory.
3. Nastaví se registry hardware pro dané rozhraní.
4. Spustí se hardware pro dané rozhraní.

B.2.3 UC02_stop

Podmínka provedení Rozhraní je připraveno pro příjem a odesílání paketů.

Stav po provedení Rozhraní je neaktivní.

Prováděné kroky

1. Je vypnuto vybrané rozhraní.
2. Zastaví se hardware pro dané rozhraní.
3. Počká se na dokončení rozpracovaných přenosů.
4. Uvolní se deskriptory pro příjem paketů.

B.2.4 UC03_remove_module

Podmínka provedení Všechna rozhraní jsou neaktivní.

Stav po provedení V jádře nejsou nadále dostupná žádná odpovídající síťová rozhraní.

Prováděné kroky

1. Registrovaná rozhraní se odejmou z jádra OS.
2. Uvolní se alokované **Rx/Tx Ringy**.
3. Z jádra OS je odstraněn modul ovladače.

B.2.5 UC04_hard_start_xmit

Podmínka provedení V **Tx Ringu** je dostupný volný deskriptor.

Stav po provedení Paket je předán hardware k odeslání.

Prováděné kroky

1. Pro `sk_buff` je vytvořeno DMA mapování z hlavní paměti do zařízení.
2. Je vytvořen odpovídající deskriptor.
3. Deskriptor je zařazen na konec **Tx Ringu**.
4. Hardware je informován o novém deskriptoru.

Alternativní provedení

Podmínka provedení Zařízení je schopné provádět SG přenosy a odesílaný paket je uložen v několika nespojitých blocích.

Stav po provedení Paket je předán hardware k odeslání.

Prováděné kroky

1. Zkontroluje se dostupnost odpovídajícího množství deskriptorů v **Tx Ringu**.
2. Pro každý `sk_buff` je vytvořeno DMA mapování z hlavní paměti do zařízení.
3. Jsou vytvořeny odpovídající deskriptory.
4. Poslední deskriptor je označen příznakem konce paketu.
5. Deskriptory jsou zařazeny na konec **Tx Ringu**.
6. Hardware je informován o nových deskriptorech.

B.2.6 UC05_tx_timeout

Podmínka provedení Signalizace o zpracování paketu nedorazí v nastaveném časovém intervalu.

Stav po provedení Paket je zpracován.

Prováděné kroky

1. Je provedena kontrola stavu přerušení v registrech hardware.
2. Provede se kontrola počtu odeslaných paketů.
3. Uvolní se alokovaný deskriptor.
4. Pro `sk_buff` je zrušeno DMA mapování.
5. `sk_buffy` jsou uvolněny.
6. Obsah hardwarových registrů je aktualizován.
7. Pokud byla zastavena, aktivuje se fronta pro odesílání paketů.

B.2.7 UC06_net_device_stats

Podmínka provedení Rozhraní je registrováno v jádře OS.

Stav po provedení Jádro OS získalo statistiky.

Prováděné kroky

1. Provede se předání shromážděných statistik.

B.2.8 UC07_poll

Podmínka provedení Rozhraní je zařazeno v seznamu obsluhy výzvou (*poll*).

Stav po provedení Přijaté pakety jsou předány jádru.

Prováděné kroky

1. Provede se kontrola **Rx Ringu**.
2. Modifikované deskriptory jsou zpracovány a odpovídající `sk_buffy` jsou předány jádru.
3. Alokují se nové `sk_buffy`.
4. Jsou vytvořeny odpovídající deskriptory a zařazeny do **Rx Ringu**.
5. Provede se aktualizace registrů hardware obsahujících indexy do **Rx Ringu**.
6. Pokud byly zpracovány všechny přijaté pakety, povolí se v hardware přerušení pro příjem paketů.

B.2.9 UC08_change_mtu

Podmínka provedení Rozhraní je neaktivní.

Stav po provedení Nastavená nová hodnota *MTU*.

Prováděné kroky

1. Zkontroluje se, zda je rozhraní neaktivní.
2. Nastaví se nová hodnota *MTU*.

B.2.10 UC09_interrupt_rx

Podmínka provedení Hardware je spuštěn, signalizováno Rx přerušení.

Stav po provedení Aktivována obsluha příjmu paketů výzvou.

Prováděné kroky

1. Přečte se vektor přerušení.
2. Zablokuje se Rx přerušení v hardware pro všechny signalizující rozhraní.
3. Všechna signalizující rozhraní jsou zařazena do seznamu obsluhy výzvou.

B.2.11 UC09_interrupt_tx

Podmínka provedení Hardware je spuštěn, signalizováno Tx přerušení.

Stav po provedení Struktury odpovídající odeslaným paketům jsou uvolněny.

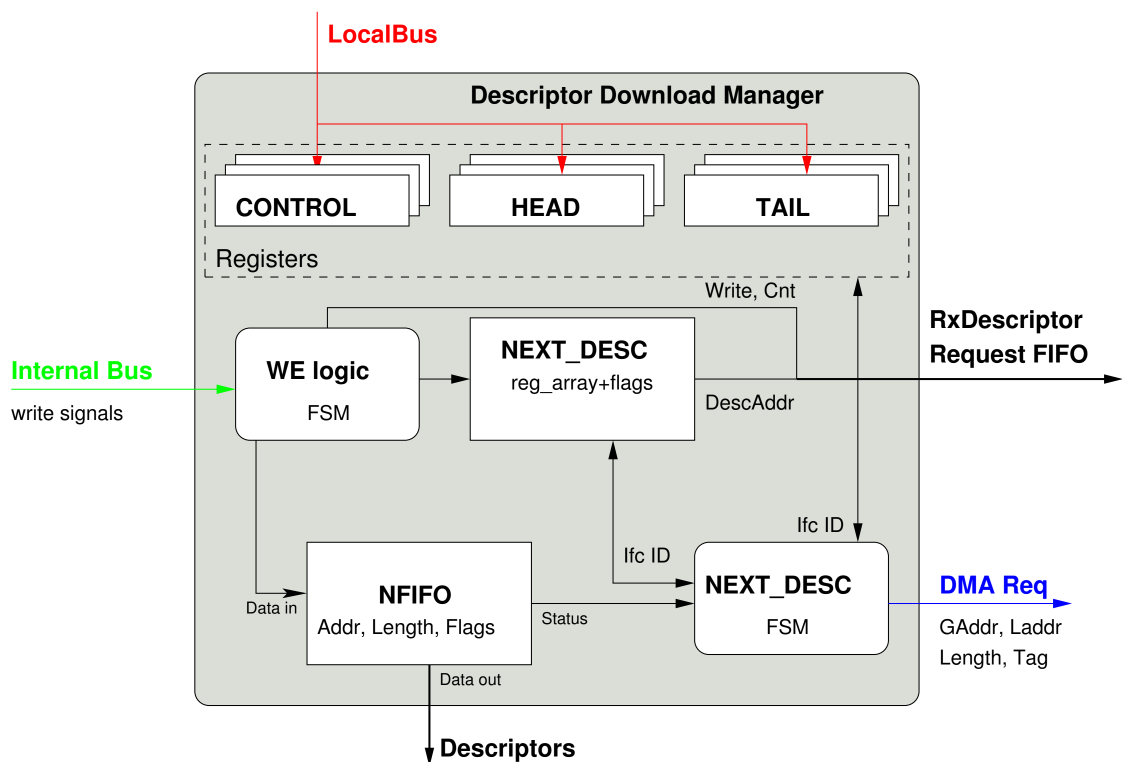
Prováděné kroky

1. Pro `sk_buffy` je zrušeno DMA mapování.
2. `sk_buffy` jsou uvolněny.
3. Jsou uvolněny všechny deskriptory odpovídající odeslaným paketům z **Tx Ringu**.
4. Jádro je informováno o odeslání paketů.
5. Obsah hardwarových registrů je aktualizován.

Příloha C

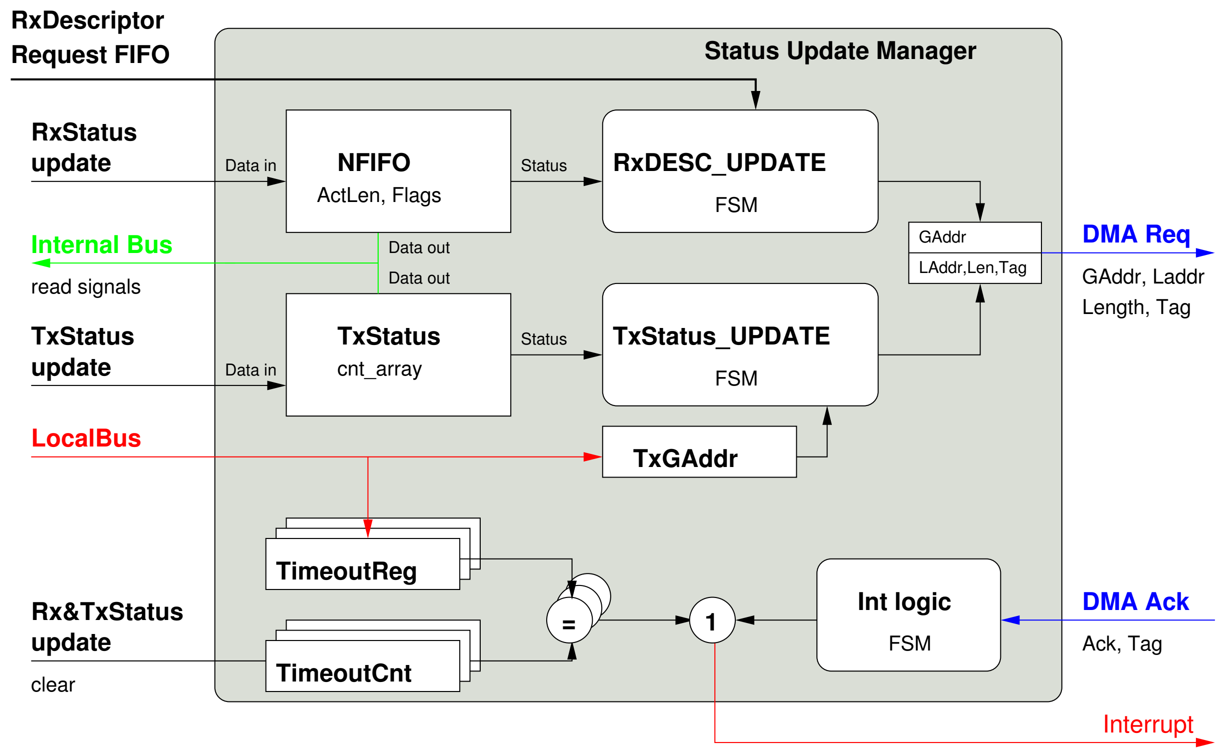
Bloková schémata

C.1 Descriptor download manager



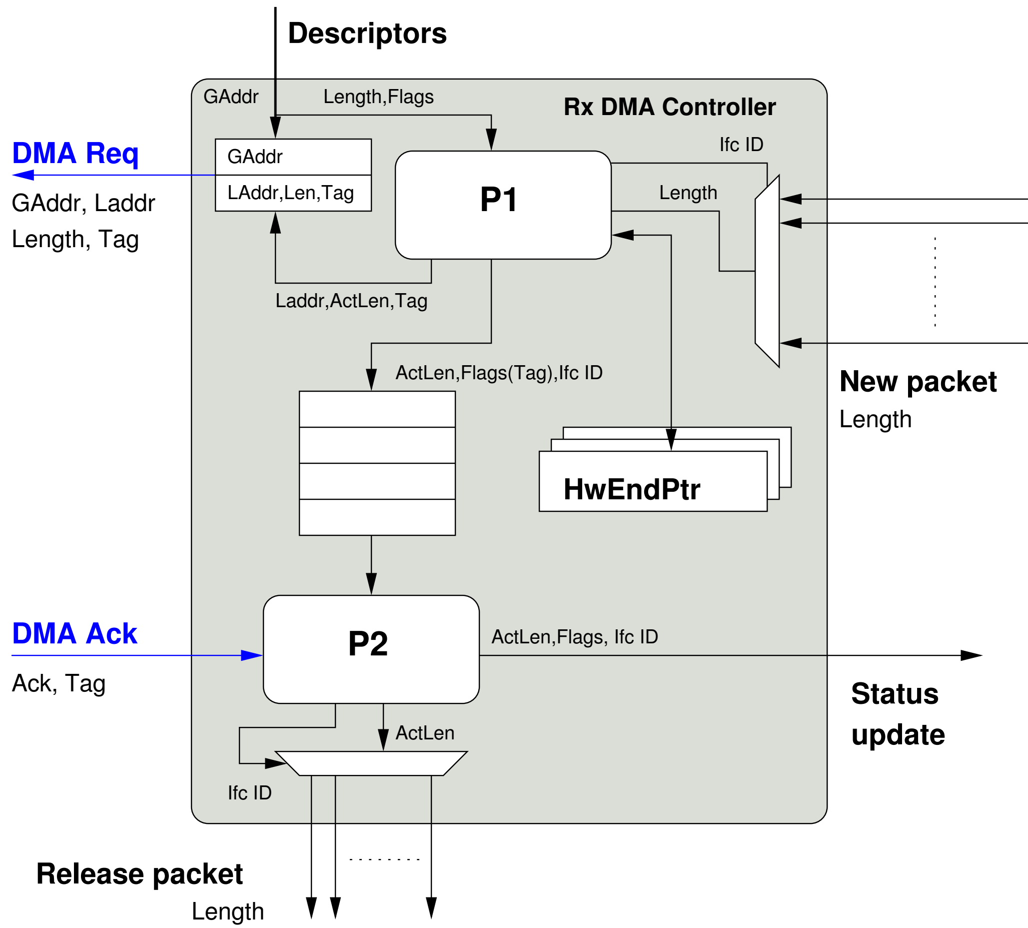
Obrázek C.1: Komponenta descriptor download manager

C.2 Status update manager



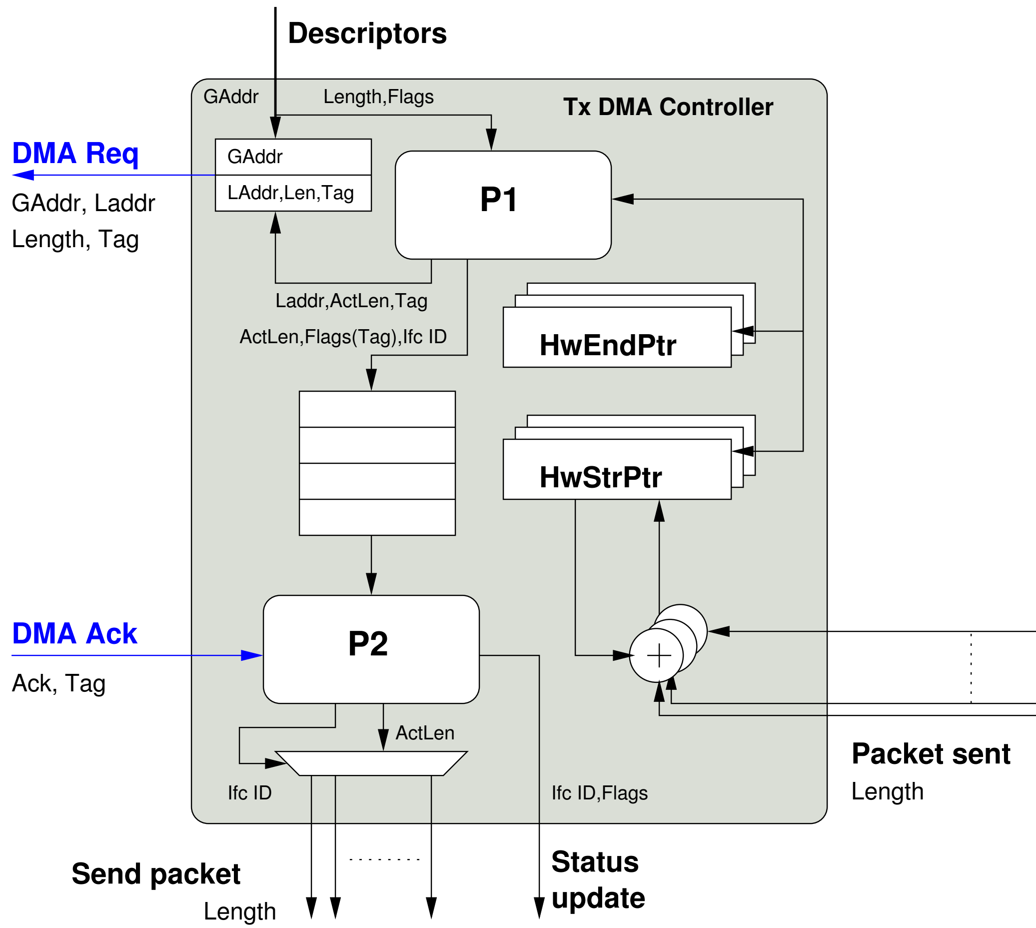
Obrázek C.2: Komponenta status update manager

C.3 Rx DMA controller



Obrázek C.3: Komponenta Rx DMA controller

C.4 Tx DMA controller



Obrázek C.4: Komponenta Tx DMA controller