VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

JEDNOTKA PRO ŘÍZENÍ RYCHLÝCH DMA PŘENOSŮ S GENERICKÝM POČTEM KANÁLŮ

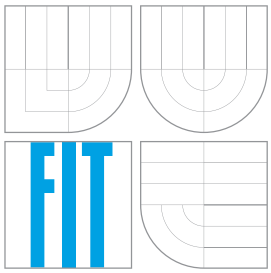
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN ŠPINLER

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

JEDNOTKA PRO ŘÍZENÍ RYCHLÝCH DMA PŘENOSŮ S GENERICKÝM POČTEM KANÁLŮ

DMA CONTROLLER WITH GENERIC NUMBER OF COMMUNICATION CHANNELS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN ŠPINLER
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MARTÍNEK
SUPERVISOR

BRNO 2009

Abstrakt

Práce se zabývá tvorbou hardwarové jednotky, která realizuje DMA přenosy mezi periferním zařízením a pamětí RAM. Jednotka je vyvíjena na platformě NetCOPE, vytvořené pro karty Combo s programovatelným hradlovým polem. I když jde o kartu primárně určenou k akceleraci zpracování síťového provozu, jednotku je možné použít univerzálně.

Abstract

Work has been developing hardware unit that implements DMA transfers between peripherals and RAM. Unit is being developed on the platform NetCOPE created for Combo cards with programmable gate array. Even if the card is primarily intended for acceleration of processing network traffic, the unit can be used universally.

Klíčová slova

přímý přístup do paměti, NetCOPE, programovatelný hardware, kruhový buffer, Linux, ovladač zařízení

Keywords

direct memory access, NetCOPE, programmable hardware, ring buffer, Linux, device driver

Citace

Martin Špinler: Jednotka pro řízení rychlých DMA přenosů s generickým počtem kanálů, bakalářská práce, Brno, FIT VUT v Brně, 2009

Jednotka pro řízení rychlých DMA přenosů s generickým počtem kanálů

Prohlášení

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

.....
Martin Špinler
20. května 2009

Poděkování

Chtěl bych poděkovat Ing. Tomáši Martínkovi za trpělivost a vstřícnost při konzultacích. Také bych rád poděkoval projektu Liberouter, který mi umožnil využít prostředí k vytvoření této práce.

Obsah

Kapitola 1

Úvod

Jedním ze základních prvků dnešního světa jsou možnosti komunikace – v moderním světě si lidé vyměňují informace, znalosti či myšlenky. Nejinak je to i v oblasti digitální techniky, komunikace počítačů s okolím je jejich neodmyslitelnou součástí. Počítač pro zařízení, jako je například hudební přehrávač nebo digitální fotoaparát, představuje úschovnu dat. Bez něho by tato zařízení byla téměř nepoužitelná. Paradoxem přitom je, že u předchůdců uvedených dvou zařízení (tedy walkman a klasický fotoaparát) by připojení k počítači ani nebylo realizovatelné.

Aby bylo možné propojit počítač s externím zařízením a přenášet mezi nimi data, oba musí mít stejné rozhraní. Je tedy nutné vytvořit jakýsi most mezi hlavní sběrnicí počítače a elektronickými obvody zařízení. Existují univerzální rozhraní, které se snaží sjednotit protokoly a fyzické spojení (např. USB), ale nelze je použít vždy. Univerzální sběrnice totiž nejsou schopny plně pokrýt všechny požadavky, nebo jinak nevyhovují konkrétnímu problému. Příkladem může být počítačová síť – od síťových zařízení se požaduje vysoká přenosová kapacita, minimální časová prodleva a jiné parametry. Proto vznikají specifické periferní moduly zvané I/O karty, typickým příkladem je zvuková, grafická nebo síťová karta. Tyto karty nám vytvářejí datovou cestu mezi počítačem a zařízením.

Protože objemy některých multimediálních či síťových dat jsou dnes natolik velké, že by procesor počítače byl plně vytížen jejich zpracováním, používáme různé techniky, jak přenést část zátěže procesoru jinam. Nejrozšířeným způsobem, jak snížit zátěž procesoru při přenosu dat mezi periferním zařízením a pamětí počítače, je technika DMA, při níž karta sama provádí přenos dat téměř bez vědomí procesoru. Na multimediálních kartách dále najdeme různé signálové procesory pro zpracování zvuku a videa. Existují i hardwarově akcelerované síťové karty, které dokáží například filtrovat síťový provoz. Jednou takovou je karta *Combo* s obvodem FPGA, jenž je vyvíjena výzkumným projektem *Liberouter*.

V této práci se budeme zabývat tvorbou hardwarové DMA jednotky, která bude řídit datové přenosy mezi pamětí počítače a periferním zařízením. V první části si vysvětlíme důležité pojmy. Řekneme si, co je to periferní operace a vysvětlíme si princip DMA přenosu. Objasníme si pojem stránkování paměti a virtuální a fyzická paměť. Předvedeme si platformu, na které budeme DMA jednotku vyvíjet. Z těchto znalostí budeme moci ve druhé části navrhnout řadič DMA pro architekturu zmíněné karty *Combo*. V další části se podíváme podrobněji na implementaci řadiče. A nakonec zjistíme, jak jsme při tvorbě nové jednotky byli úspěšní.

Kapitola 2

Seznámení s prostředky

Abychom správně navrhli a implementovali nějakou komponentu, musíme vědět, jaké chování od ní očekáváme. Stejně tak je pro nás důležité znát rozhraní, ke kterému ji připojíme. Potřebné informace si shrneme v této kapitole.

2.1 Periferní operace

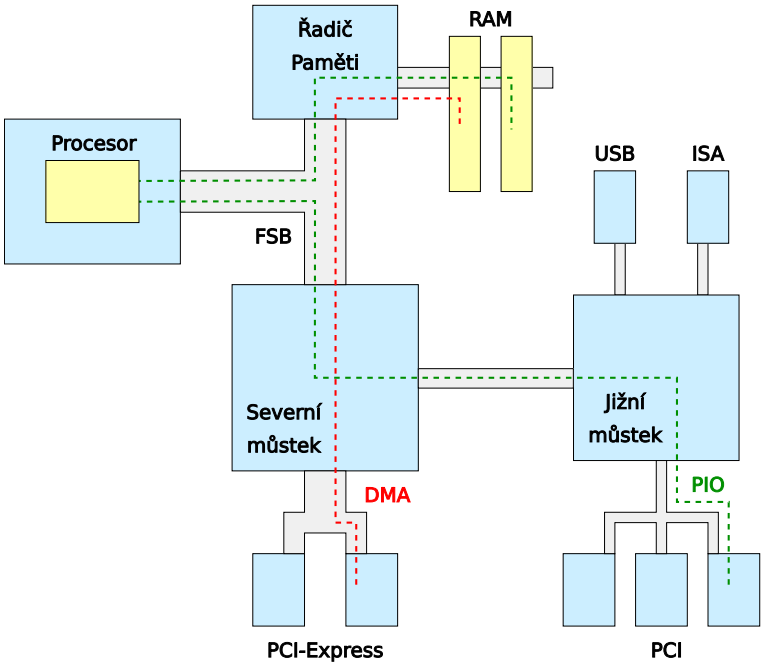
Periferní, nebo také vstupně-výstupní (V/V, z angl. I/O, input/output), operace je sled událostí za účelem komunikace počítače s periferním zařízením (dále jen PZ). Při periferní operaci se přenesou data mezi PZ připojeným k systémové sběrnici a pamětí počítače nebo procesorem. Na zjednodušeném blokovém schématu počítače 2.1 jsou vyznačeny dva typy periferních operací, které si představíme. Jejich průběh můžeme přibližně shrnout do těchto základních kroků:

1. Zajištění sběrnice a nastavení jejích signálů.
2. Provedení přenosu dat.
3. Ukončení práce se sběrnicí.

2.1.1 Programovaný vstup a výstup

V procesoru existují I/O instrukce, které umožňují přímo zapsat nebo přečíst jednu hodnotu registru v PZ. Při provedení těchto instrukcí jsou nastaveny adresové, řídicí a datové signály systémové sběrnice. PZ z adresy na sběrnici rozpozná svůj adresový prostor. Zjistí, zda je na sběrnici požadavek pro zápis nebo čtení a podle něho data přečte, nebo vystaví na datovou sběrnici. Data se po sběrnici přenesou z nebo do univerzálního registru procesoru a periferní operace je dokončena.

V PIO (Programmed I/O) režimu procesor čte či zapisuje data z PZ vykonáním jedné I/O instrukce právě tehdy, když o to program žádá. Pokud je potřeba přenést blok dat, I/O instrukce musí být použita vícekrát. Většinou přitom po každé čtecí I/O instrukci přenáší přijatá data z procesoru do paměti pro pozdější zpracování. Při zápisu bloku dat do PZ je přenos z paměti do procesoru před každou instrukcí samozřejmostí.



Obrázek 2.1: Periferní operace PIO a DMA.

2.1.2 Přímý přístup do paměti

V situacích, kdy je potřeba přenášet do paměti velké množství dat, přestává být PIO režim efektivní, protože značně zatěžuje procesor. Proto byla navržena technika, která má v těchto situacích odlehčit procesoru. DMA (Direct Memory Access) je režim, kdy přenos dat aktivně nevykonává procesor, ale jiné zařízení. Procesor jej na začátku pouze inicializuje a po dokončení přenosu data zpracuje. V průběhu přenosu je odpojen od sběrnice a může dál pokračovat ve vykonávání programu.

Tento režim má určitá specifika pro různé sběrnice.

Sběrnice ISA

Pro řízení přímého přístupu do paměti zde slouží jeden nebo dva DMA řadiče. Řadič může obsluhovat až čtyři kanály, přičemž každý z kanálů má vlastní adresový registr a čítač. Kanály lze sdílet, takže je možno mít více PZ používajících jeden kanál.

Pro zahájení DMA přenosu ovladač PZ zapíše do DMA řadiče korektní hodnoty. Je nutné nastavit směr a délku přenosu a zdrojovou i cílovou adresu paměti. Řadič posléze převezme kontrolu nad sběrnicí a provede přenos dat sám. Procesor získá zpět řízení sběrnice v okamžiku, kdy je přenos dokončen.

Sběrnice PCI

Na sběrnicích typu PCI (PCI-X, PCI-Express) neexistuje DMA řadič jako samostatná jednotka. Odpadá tedy sdílení kanálů a nastavování hodnot ze strany ovladače zařízení při každém přenosu. Na druhou stranu se konstrukce PZ stává složitější, neboť každé PCI zařízení musí umět být *Bus Master* (tzn. musí umět samo iniciovat přenos a řídit sběrnici). Uveďme typické schéma přenosů na sběrnici PCI.

1. Pro zahájení DMA přenosu PZ nejprve požádá arbitr o přidělení sběrnice.
2. Jakmile procesor nebo jiné PZ dokončí přenos dat po sběrnici, arbitr ji přidělí žádajícímu PZ. Je-li žádajících PZ více, arbitr sběrnici přidělí PZ s nejvyšší prioritou[1].
3. Poté je procesor od sběrnice odpojen a veškeré její řízení a přenos dat zastává PZ.
4. Po dokončení přenosu PZ sběrnici uvolní a procesor se opět ujme jejího řízení.

2.1.3 Stav periferního zařízení

Jsou dva důležité případy, kdy program potřebuje znát stav PZ.

- Program odešle nějaký příkaz do PZ a musí vědět, kdy byl tento příkaz dokončen. Příkladem může být požadavek na nastavení hlavičky disku na určitou stopu.
- Nastala nějaká událost, kterou PZ zaznamenalo a připravilo data pro zpracování. Zde si můžeme uvést například stisknutí klávesy, nebo pohyb myší.

Polling

Základní způsob jak zjistit stav PZ je opakované čtení jeho stavového registru. PZ nastaví tento registr na určitou hodnotu, jakmile zpracuje data. Procesor si informaci o stavu při dalším čtení vyzvedne.

Dnes se však dává přednost druhému způsobu, protože polling zbytečně zpomaluje vykonávání programu.

Přerušení

Přerušení je systém, jakým PZ samovolně oznámí nějakou událost. Každé PZ má přiřazen sdílený nebo vlastní signál na sběrnici (způsoby zapojení se liší se dle dané sběrnice)[2]. Po dokončení operace PZ asynchronně vyšle tento signál směrem k procesoru. Procesor signál přijme a provede speciální kód, který se nazývá ISR (Interrupt Service Routine). Program obslouží přerušení například přečtením zpracovaných dat a vrátí se k původnímu programu.

Systém přerušení je ve většině případů výhodnější než polling, procesor totiž aktivně pracuje po celou dobu operace PZ.

2.2 Správa paměti

Toto téma je značně rozsáhlé a vydá na samostatnou studii – každá počítačová architektura používá vlastní koncept, o operačních systémech nemluvě. Obecný výklad lze nalézt v [5]. Omezíme se tedy na architekturu x86 a operační systém s jádrem Linux.

[1] Ve skutečnosti specifikace PCI neurčuje, jakým způsobem arbitr má přidělovat sběrnici, takže je možné implementovat různé algoritmy podle konkrétních požadavků, více podrobností najdeme v [3].

[2] Na sběrnicích PCI lze přerušení vygenerovat i pomocí MSI (Message Signal Interrupt[8]) zápisem na speciální adresu. Pro sběrnici PCI-Express je toto dokonce jedinou možností.

2.2.1 Adresové prostory

I/O prostor

Komunikace s PZ pomocí I/O prostoru (neboli přes porty) je poměrně stará a dnes se přestává používat. Nicméně nalezneme tu stále registry důležitých jednotek počítače, jako je řadič DMA, časovač nebo sériový port. Nejčastěji slouží k řízení nebo získání stavu jednotky použitím zmíňených I/O instrukcí. V tabulce 2.1 je ukázka některých komponent a jejich portů.

Paměťový prostor

Paměť všech komponent se mapuje do jediného fyzického paměťového prostrou. Největším představitelem je samozřejmě systémová RAM. Dále je zde například Video RAM/ROM a prostor FLASH paměti obsahující BIOS. Značnou výhodou je možnost do fyzického paměťového prostoru mapovat i pamět a řídící registry jiných zařízení a pracovat s nimi jako s obyčejnou pamětí. Je to rychlejší oproti I/O instrukcím a nejsme tolik limitováni velikostí I/O prostoru.

2.2.2 Virtuální paměť a stránkování

Jak se operační systémy vyvíjely, jednoduchý *Flat Memory* model (viz [1]) přestával být použitelný. Mohl nastat problém, kdy byl program větší než fyzická paměť. Vznikl tedy virtuální adresovatelný prostor nezávislý na velikosti fyzické paměti. I když má také svoje limity dané architekturou (32 bitová platforma x86 může adresovat maximálně 4GiB), pro většinu programů to dostačuje. Pro ty ostatní existují techniky, jak tento limit obejít (např. *mmap*).

Samotná virtuální paměť by nám nepřinesla velké výhody, je zde ještě jedna důležitá technika. Virtuální paměť je rozdělena na malé úseky stejné velikosti (např. 4KiB) nazývané stránky (*pages*). Každou stránku je pak možné namapovat na blok fyzické paměti zvaný rámec (*frame*), nebo ji uložit do odklácího prostoru (*swap*).

Díky těmto dvěma technikám je možné spustit jakýkoliv program, aniž by narušil běh ostatních programů či jádra operačního systému. Každý program má totiž svůj vlastní souvislý prostor, bez ohledu na to, jak jsou data rozmístěna ve fyzické paměti. Tím se zvýší bezpečnost, která je podpořena i hardwarovou jednotkou MMU (Memory Management Unit) provádějící za pomoci jádra překlady virtuálních adres na fyzické.

Začátek	Konec	Popis
`0x0000`	`0x001F`	DMA řadič 1
`0x0020`	`0x0021`	PIC (Programovatelný řadič přerušení)
`0x0040`	`0x0043`	Časovač 0
`0x0050`	`0x0053`	Časovač 1
`0x0070`	`0x0077`	RTC0 (Obvod reálného času)
`0x02F8`	`0x02FF`	Sériový port 1
`0x0378`	`0x037A`	Paralelní port 1
`0x1000`	`0x107F`	ACPI

Tabulka 2.1: Ukázka obsazení I/O prostoru

Na obrázku A.1 můžeme vidět mapování zařízení do fyzické paměti, překlady adres a některé problémy s nimi spojené. Více informací lze nalézt v [4].

2.3 Platforma NetCOPE

NetCOPE poskytuje prostředí pro rychlý vývoj aplikací postavených na akcelerované síťové kartě Combo. Využijeme tuto platformu pro vývoj DMA jednotky a zhruba si popíšeme hotové jednotky, jaké použijeme při implementaci.

2.3.1 Vývojový cyklus

Tvorba komponenty probíhá v několika krocích. Každý z nich si rozebereme, a uvedeme si používané nástroje.

Návrh

Při návrhu komponenty je důležité si co nejpodrobněji probrat její požadovanou funkci, a navrhnout rozhraní. Nejvíce práce zastane tužka a papír, ale někdy se nám může hodit například podrobná technická specifikace elektronického obvodu, se kterým pracujeme.

Implementace

Většina zdrojových souborů platformy NetCOPE je psána v jazyce VHDL, který patří do rodiny jazyků HDL pro popis hardwaru (hardware description language). Používá se pro implementaci a simulaci číslicových obvodů – programovatelných hradlových polí (CPLD a FPGA) nebo jednoúčelových ASIC obvodů. Ve VHDL je komponenta popisována téměř na úrovni hradel (registry, sčítačky...), lze jej tedy částečně přirovnat k jazyku assembler, používaného procesory.

Někdy je však potřeba vytvořit složitou jednotku, u které by implementace ve VHDL byla náročná. Proto existují i jiné jazyky pro popis hardwaru na vyšší úrovni, jedním z nich je Handel C. Podobá se klasickému jazyku C, avšak má výrazné odlišnosti. Hlavní změnou je možnost příkazy provádět nejen sekvenčně, ale také je paralelizovat. Dále například dovoluje vytvářet proměnné s přesně určenou datovou šířkou, nebo přistupovat pouze k jejich určitým bitům. Bohužel jako každý vyšší programovací jazyk i Handel C s sebou nese jistá omezení. Například neumožňuje používat generické rozhraní, Tento problém lze sice částečně obejít, ale je nutné pokaždé zdrojový soubor přeložit s novými parametry. Pro další kroky tedy potřebujeme kompilátor, který přeloží zdrojové kódy z Handel C do VHDL

Simulace a verifikace

Po vytvoření jednotky je nutné zkontrolovat její funkčnost. Slouží k tomu simulátory číslicových obvodů, nejpoužívanějším je program ModelSim. K vytvořené komponentě se připojí naprogramovaný generátor signálů (může to být opět jednotka napsaná ve VHDL), spustí se časová simulace a sledují se výstupní signály. Proběhne základní test funkčnosti a případně se opraví viditelné chyby.

V simulaci jednotka sice může vypadat jako funkční, nelze ale zaručit její bezchybnost. Tu kontrolujeme verifikací – procesem, při kterém je komponenta testována pro všechny vstupní kombinace. Lze ji tedy přirovnat k automatizované komplexní simulaci. Teprve pokud komponenta projde verifikací, lze ji téměř považovat za bezchybnou.

Syntéza a testování

Abychom jednotku mohli používat přímo v obvodu, je nutné zdrojový kód přeložit do formátu, kterému obvod rozumí. K tomu slouží nástroje pro syntézu, např. XST. Jejich prací je nejprve ze zdrojových kódů získat model obvodu – vytvořit funkční tabulky pro LUT, rozpoznat registry, paměti BlockRam... Ve druhé fázi pak rozmístí bloky po konkrétním FPGA a propojí je (place and route). Po nahrání výsledného *bitstreamu* do FPGA otestujeme a doladíme funkčnost jednotky v reálném prostředí.

2.3.2 Sběrnice

Ukažme si sběrnice používané na kartě Combo. Podrobnější popis najdeme na [7].

Internal Bus

Toto je hlavní sběrnice na kartě, kudy projdou všechna softwarem přijaté a odeslané data a řídicí příkazy. Internal Bus tvoří několik komponent, jenž se skládají do stromové struktury.

První a nejdůležitější *IB Root* odesílá a přijímá transakce a je úzce spojena s PCI-Express bridgem. Na kartě existuje vždy jen jediná instance.

Další komponenta *IB Switch* inteligentně (podle adresy) rozděluje jednu větev stromu sběrnice na dvě.

Poslední komponenta *IB Endpoint* propojuje nějakou jinou komponentu se sběrnicí. Jejím výstupem jsou dvě oddělená rozhraní (zápisové a čtecí). Existuje varianta *IB Endpointu* doplněná o *Bus Master* rozhraní, které dovoluje komponentě generovat požadavky na přenos dat mezi pamětí RAM.

Local Bus

Sběrnice Local Bus se hodí pro řízení komponent nebo pro přenos malých objemů dat. Stejně jako Internal Bus ji lze sestavit do stromové struktury pomocí podobných komponent: *LB Root* (převádí Internal Bus na Local Bus), *LB Switch* a *LB Endpoint*. Poslední jmenovaná má na výstupu zjednodušené paměťové rozhraní MI32.

Frame Link

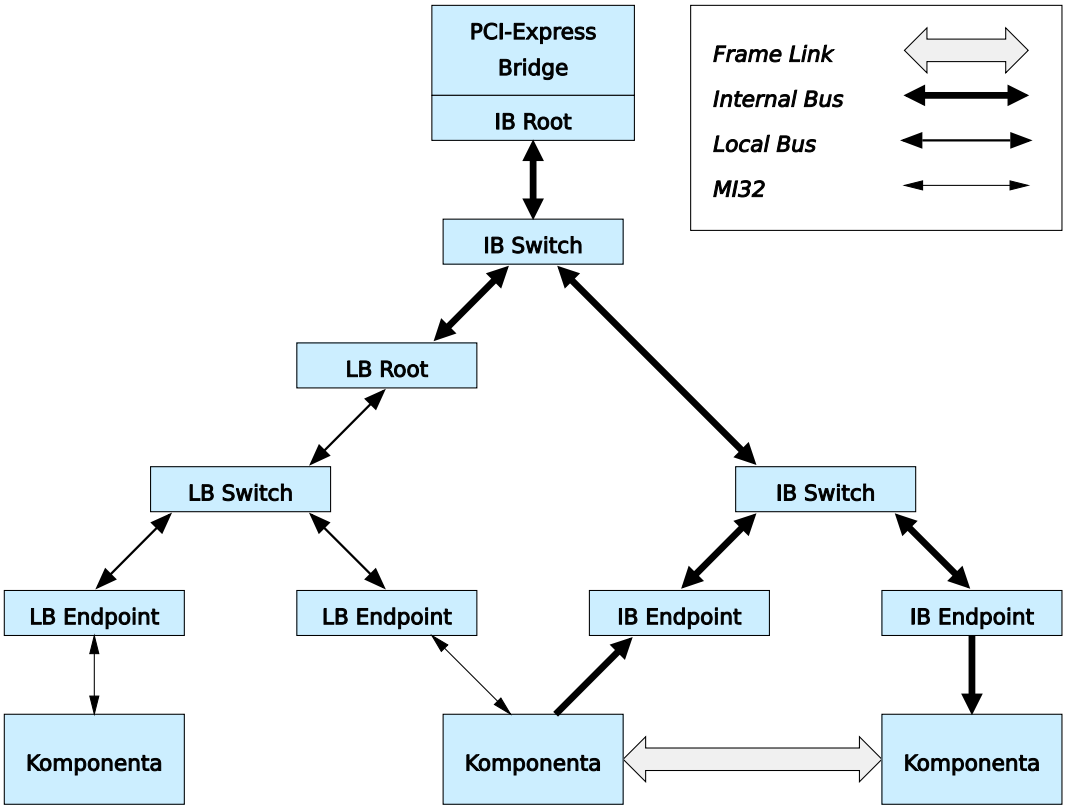
Spíše než sběrnice je to dedikovaný datový spoj, inspirovaný protokolem *Local Link* firmy Xilinx. Propojuje vždy dvě jednotky, které s daty nějakým způsobem pracují. Přenáší se po něm čistě užitečná data, budeme je dále nazývat datové pakety.

2.3.3 Komponenty

SW Buffery

Softwarové buffery jsou dvě podobné nezávislé komponenty. Název „SW Buffer“ je odvozen z jejich funkce přenášet data mezi softwarem. Používají princip kruhových bufferů, který bude vysvětlen v kapitole 3.1. Jejich chování lze na jedné straně přirovnat k paměti, na druhé straně k frontě FIFO.

První komponenta přijímá (příchozí směr bude dále značen jako Rx) data z rozhraní FrameLink. Po přijetí celého datového paketu informuje připojenou jednotku o příchozích datech signálem `NEWLEN`. Ta vydá požadavek na čtení dat z bufferu skrz čtecí rozhraní



Obrázek 2.2: Sběrnice platformy NetCOPE

InternalBus. Data zůstávají v bufferu do doby, než je z něj komponenta po dokončení přenosu uvolní signálem `RELLEN`.

Druhá komponenta analogicky odesílá (odchozí směr bude dále značen jako Tx) data na rozhraní FrameLink. Jakmile je do ní zapsán datový paket přes zapisovací rozhraní InternalBus, připojená jednotka ji informuje o velikosti nových dat k odeslání signálem `NEWLEN`. Po odeslání dat buffer oznámí velikost odeslaného datového paketu připojené jednotce pomocí signálu `RELLEN`.

Zmiňme ještě jednu důležitou informaci. Aby nebylo nutné při čtení nebo zápisu dat procházejících přes hranice bufferu generovat dva požadavky, adresový prostor bufferu je zdvojnásoben a data v něm jsou klonovány.

Descriptor Manager

Abychom pochopili význam Descriptor Manageru, nejprve trochu zapřemýšlejme nad funkcí DMA jednotky. Jelikož má generovat požadavky na přenos dat z nebo do paměti RAM, musí mít nějakou informaci o tom, na jakou adresu má přistupovat.

Uvažujme 64 MiB prostor v RAM pro přenášená data. Pokud by tento prostor byl v jednom souvislém bloku paměti, stačilo by do registru PZ zapsat informaci o jeho začátku a velikosti. Ale není to zrovna malý kus paměti, takže kdybychom chtěli alokovat celý prostor jako jeden souvislý blok paměti, nemuseli bychom uspět. Operační systém totiž s pamětí nakládá, jak potřebuje a časem dochází ke fragmentaci, podobně jako u dat na pevném disku. Ovladač proto naalokuje požadovanou paměť, která nemusí být souvislá a zjistí ukazatele na začátek každé její stránky. Tyto ukazatele, které obsahují fyzickou

adresu stránky, nazýváme deskriptory.

Už předem je zamítnuta možnost připravit prostor pro všechny deskriptory v FPGA a při alokaci kruhového bufferu je naplnil platnými daty. Pro uvažovanou velikost prostoru v RAM by bylo zapotřebí 16 384 deskriptorů o velikosti 64 bitů. Takové upotřebení zdrojů je nemyslitelné, nehledě na ztrátu možnosti změnit velikost prostoru bez rekonfigurace FPGA.

Byla vytvořena komponenta Descriptor Manager, jejíž funkcí je automaticky číst připravené deskriptory z paměti RAM a uchovávat je, dokud si je nevyzvednou DMA řadiče. Nejprve je nutné komponentu správně inicializovat, toho je docíleno zápisem prvního deskriptoru pro každý kanál na speciální adresu pomocí zápisového rozhraní Internal Bus. Jakmile je po inicializaci aktivován některý z kanálů (signálem `ENABLE` z DMA řadiče), Descriptor Manager automaticky začne generovat DMA požadavky na stažení deskriptorů.

Existují dva typy deskriptorů:

1. typ 0 - deskriptor ukazující na začátek stránky v RAM pro přenášená data.
2. typ 1 - deskriptor ukazující na stránku v RAM, kde jsou umístěny další deskriptory. Bývá posledním deskriptorem ve stránce, ale může se vyskytnout i dříve.

Kapitola 3

Návrh

Teď už známe všechny potřebné principy a jednotky, z nichž můžeme navrhnout kompletní DMA jednotku. Aby byla jednotka univerzální, nebudeme se zajímat o obsah dat, zajistíme pouze přenos ze vstupního rozhraní do zpracovávající aplikace. Analogicky zajistíme přenos dat z aplikace generující data na výstupní rozhraní.

3.1 Kruhové buffery

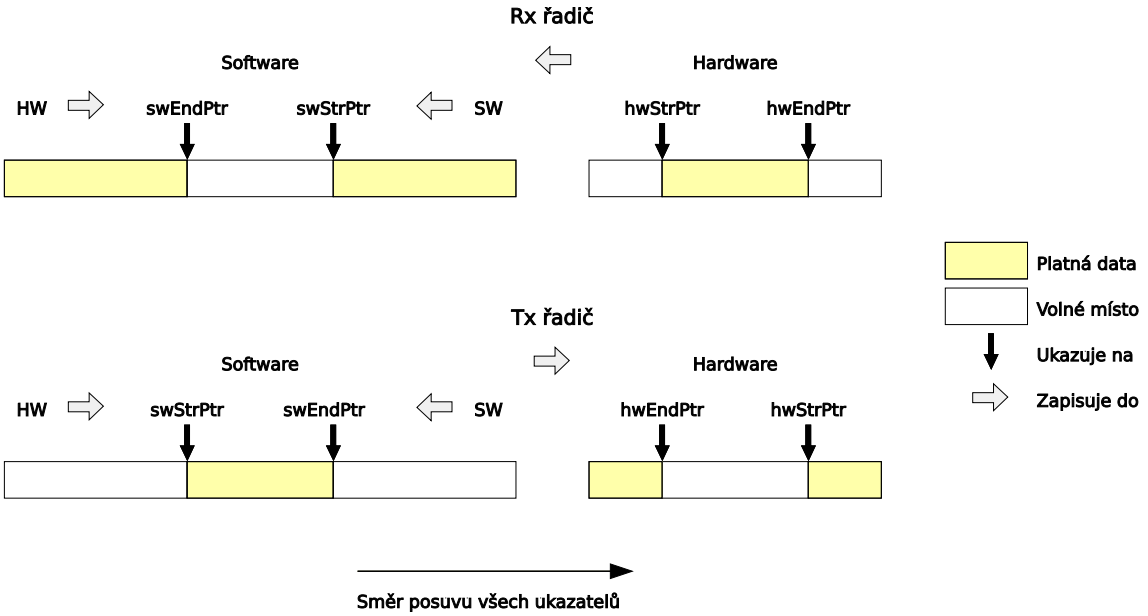
Prvním bodem návrhu je systém uložení dat v počítači. Jako nejpoužitelnější se jeví metoda kruhových bufferů v paměti RAM, můžeme ji přirovnat k nekonečné frontě FIFO. Není u ní potřeba obsah paměti nijak přesouvat či mazat po jejím zpracování. Orientaci v následujícím konceptu nám usnadní obrázek 3.1.

Při příjmu paketu PZ zapíše příchozí data na začátek volného místa v kruhovém bufferu a označí toto místo jako zaplněné. Jakmile se software dozví, že v kruhovém bufferu jsou platná data, zpracuje je a následně označí toto místo jako volné. Podobně je to i pro odesílání: software nejprve zapíše odchozí data na začátek volného místa, kde si je hardware posléze vyzvedne.

Označování volného a obsazeného místa probíhá posouváním ukazatelů na začátek a konec bufferu (StrPtr, EndPtr). Přiřadíme si je následovně: StrPtr bude ukazovat na začátek místa, kde jsou data, naproti tomu EndPtr bude ukazovat na místo, kde platná data končí. Protože SW Buffery v PZ také používají metodu kruhových bufferů, budeme potřebovat ukazatele i na ně. V řadiči ukazatele rozlišíme prefixem hw a sw.

Nastává zde problém jak zjistit, zda je buffer plný nebo volný, při těchto stavech totiž oba ukazatele mají stejnou hodnotu. U každého směru tento stav budeme řešit zvlášť, vrátíme se k tomu tedy při implementaci. Uveďme si alespoň pár základních způsobů, jakými lze problém řešit:

- Vždy nechat volný jeden byte.
- Použít počítadlo obsazených dat.
- Uchovávat si počet zapsaných a přečtených bytů
- Ukazatelům přidat jeden adresový bit navíc.



Obrázek 3.1: Princip kruhových bufferů pro DMA řadič.

3.2 Řadiče DMA

Nejprve zapracujme do návrhu požadavek generického počtu kanálů, který můžeme splnit dvěma způsoby. Protože Descriptor Manager a SW Buffery mají signály pro každý kanál zvlášť, nabízí se možnost pro každý kanál instancovat jednu jednotku. Ovšem toto je zaprvé nepřehledné a zadruhé náročné na zdroje FPGA. Zvolíme tedy druhý, implementačně náročnější způsob: vytvoříme jediný řadič pro každý směr, který bude mít společnou logiku pro více kanálů. Obsloužení všech kanálů pak dosáhneme jejich postupným procházením.

I když budou řadiče pro Rx a Tx směr do jisté míry podobné (blokové schéma obou řadičů je na obr. 3.2), jejich odlišnosti mohou být matoucí a proto nejprve uvedeme Rx a potom si probereme rozdíly Tx směru.

3.2.1 Rx řadič

Shrňme si činnosti, které má řadič vykonávat:

1. Získávat informaci o délce dat přijatých do Rx Bufferu.
2. Generovat požadavek na DMA přenos obsahující směr a délku transakce, zdrojovou adresu v Rx Bufferu a cílovou adresu v paměti RAM.
3. Přijímat informace o dokončení přenosů a podle toho uvolňovat data z Rx Bufferu.
4. Generovat přerušení po překročení nastaveného časového limitu nebo určeného množství přenesených dat.

Z toho vyjdeme a vytvoříme čtyři nezávislé bloky vykonávající požadovanou funkci.

Blok zpracovávající nová data

Tento blok bude jednoduše přičítat délku nových dat k registru. Registr ukazuje na místo, kde končí platná data, bude to tedy `hwEndPtr`.

Blok generující DMA požadavek

První fází je zjištění nových dat – pokud žádná nová data nezjistíme, přejdeme na další kanál a provedeme to samé.

Ve druhé fázi musíme zjistit, zda můžeme všechna nová data přenést v jednom požadavku. Místo v aktuální stránce paměti, do které přenášíme, totiž nemusí být dostatečně velké pro celý blok dat. Proto ořízneme data na velikost volného místa ve stránce.

Úkolem další fáze je zjistit, kolik je volného místa v softwarovém bufferu, abychom nepřepsali data, které ještě software nezpracoval. Pokud je softwarový buffer zcela zaplněn, vygenerování požadavku se neprovede a místo toho pokračujeme na dalším kanálu zase od začátku.

Jestliže jsme se dostali až do poslední fáze, nic nám nebrání vygenerovat sestavený požadavek. Následně aktualizujeme patřičné ukazatele (`hwStrPtr` a lokální `swEndPtr`, který bude vysvětlen dále) a pokud jsme zaplnili celou stránku, vydáme požadavek na přečtení nového deskriptoru.

Blok zpracovávající dokončení přenosů

Jakmile PCI-Express bridge dokončí přenos vygenerovaného požadavku do paměti RAM, oznámí to signálem. Tento signál neobsahuje délku přenesených dat, takže si ji budeme muset zapamatovat již v předchozím bloku. Současně s ní budeme potřebovat vědět, na jakém kanálu byl přenos dokončen. Využijeme alespoň toho, že bridge obsluhuje požadavky v pořadí ve kterém přišly, a propojíme tento a předchozí blok frontou FIFO. V opačném případě by zde bylo nutné použít složitějších mechanizmů, jak identifikovat dokončený požadavek.

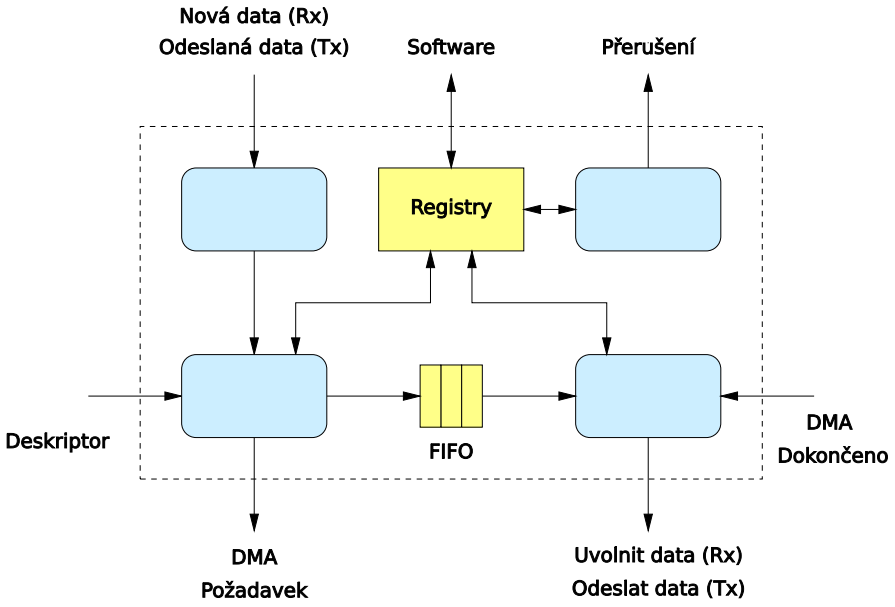
Nejprve tedy vyzvedneme informace o přenosu z fronty FIFO. Následně uvolníme z Rx Bufferu přenesená data, a zároveň jejich délku přičteme k obsahu registru `swEndPtr`.

Blok generující přerušení

Nejdříve si upřesněme, kdy má řadič přerušení vyvolat. Přerušení je tu pro informování ovladače, že v paměti RAM jsou platná data čekající na zpracování. Můžeme generovat přerušení po každém paketu, ovšem v některých případech by to mělo negativní dopad na vytížení procesoru. Zvolíme raději způsob, při kterém budeme informovat ovladač po přenesení několika paketů, resp. určité délky dat. Ideální bude možnost tuto hodnotu nastavovat ze softwaru, protože každá aplikace potřebuje data zpracovat jinak rychle.

Nyní uvažujme případ, kdy aplikace nastaví tuto hodnotu poměrně velikou. Může se stát, že přijde jen určité množství paketů, které ještě mezní hranici nepřekročí. Data pak sice budou v RAM, ale software o nich nebude vědět. Proto zapojíme další mechanizmus vyvolání přerušení a to časový limit (timeout). Hned po přenosu dat nastavíme sestupný čítač na určitou hodnotu (opět nastavitelnou softwarem), a jakmile dojde do nuly, vyvoláme přerušení. Jestliže se v průběhu čítání dokončí přenos dalších dat, čítač pouze nastavíme na původní hodnotu (vynulujeme) a necháme ho volně odčítat dál.

Prvním krokem tedy bude zjistit, zda ukazatel `swEndPtr` překročil nastavený limit. Jestliže je tato hodnota překročena a zároveň je povoleno přerušení, vygenerujeme ho a veškerá další zablokujeme. Jinak v dalším kroku buď vynulujeme nebo dekrementujeme čítač, v závislosti na tom, zda byla přenesena nová data, či nikoliv. Pokud čítač dosáhne nulové hodnoty, stejně jako v prvním kroku vygenerujeme přerušení a další zakážeme.



Obrázek 3.2: Blokové schéma Rx a Tx řadiče.

3.2.2 Tx řadič

Opět si ujasněme, jak má řadič pracovat:

1. Po zjištění nových dat v RAM generovat požadavek na přenos. Ten bude opět obsahovat směr a délku, zdrojovou adresu v paměti RAM a cílovou adresu v Tx Bufferu.
2. Po dokončení přenosu oznámit velikost nových dat Tx Bufferu.
3. Čekat oznámení o odeslání dat od Tx Bufferu a podle toho počítat volné místo.
4. Generovat přerušení po překročení nastaveného časového limitu nebo přenesení určitého množství dat.

Je vidět že funkce Rx a Tx řadiče jsou dosti podobné, tomu samozřejmě odpovídá i návrh bloků. Blok generující přerušení má dokonce naprosto stejnu funkčnost.

Popíšeme si jen změny v hlavním bloku, protože v ostatních se týkají například jen použitých ukazatelů.

Blok generující DMA požadavek

Nejprve porovnáme, zda se liší `swStrPtr` a `swEndPtr`. Pokud se shodují, žádná nová data zde nejsou, takže pokračujeme dalším kanálem. Ve druhém kroku určíme délku přenosu jako minimum z délky nových dat a volného místa v bufferu. Dále zarovnáme tuto délku na hranice stránky, abychom přenesli pouze platná data. V následujícím kroku můžeme sestavit a vygenerovat požadavek. Nakonec aktualizujeme patřičné ukazatele (`hwEndPtr` a lokální `swStrPtr`) a případně požádáme o nový deskriptor.

3.2.3 Problémy Rx i Tx řadiče

Lokální ukazatele

Aby byl ovladač správně informován o nových datech, ukazatele `swEndPtr` u Rx a `swStrPtr` u Tx řadiče musí být aktualizovány až po dokončení přenosu. Nicméně blok generující požadavek potřebuje vědět, které místo použijí vygenerované, avšak ještě nedokončené požadavky. Toto místo už nesmí být použito pro vygenerování dalších požadavků. Proto u obou řadičů bude tento blok obsahovat vlastní ukazatel.

Přenos přes stránku

S problémem lokálního `swEndPtr` u Rx řadiče částečně souvisí i přenos přes hranici stránky. Představme si situaci, kdy ovladač přečte tento registr hned po dokončení přenosu první části dat. Přečetl by sice platnou informaci, ale nijak by nepoznal, že datový paket v RAM není kompletní. Musel by přímo z dat získat velikost datového paketu, a podle toho dopočítat, zda je v paměti celý. Může ale nastat tak extrémní situace, že přijde jen první byte dat, takže na dalších bytech nejsou platná data. V tom případě software není schopen korektně zjistit velikost dat. Z toho plyne, že ovladač nesmí získat nový stav ukazatele dříve, než se dokončí přenos celého paketu. U Rx směru tedy musíme do řadiče implementovat logiku, která aktualizuje tento registr až po přenesení celého datového paketu. Informaci o přenosu přes stránku budeme získávat pomocí příznaku Page Break (`PB_flag`).

U Tx směru tento problém sice nastává také, ale poněkud jiným způsobem. Při nesprávném ukončení přenosu dat může být přenesena jen první část paketu. Druhá část už nikdy nedojde, a při dalším spuštění nová data téměř jistě zmatou Tx Buffer. Takže ovladač při zastavení řadiče musí zajistit, aby DMA jednotka vždy obdržela celý paket.

3.3 Ovladač

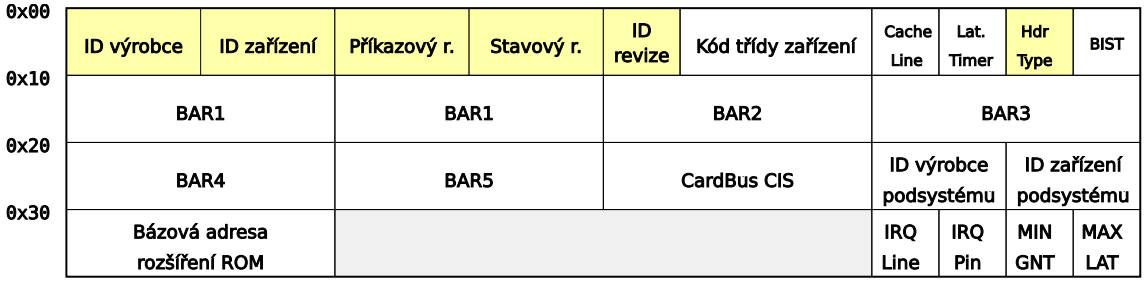
Ovladače jsou nedílnou součástí operačního systému. Je to vrstva mezi hardwarem a softwarem, která uživateli dovoluje jednodušeji přistupovat k hardwaru.

V Linuxu existují tři druhy ovladačů, každý z nich poskytuje jiné komunikační rozhraní pro přístup k zařízení[2, s.7]:

- **Znaková zařízení**. Patří sem seriové porty, tiskárny, terminály a jiná zařízení pracující s datovými toky.
- **Bloková zařízení**. Typicky jsou to úložná zařízení – pevné disky, výměnná média (diskety, CD, DVD) a paměti (RAM, ROM, flash atd.). Zpracovávají data po celých blocích a většinou je na nich umístěn nějaký souborový systém.
- **Síťová zařízení**. Mezi ně počítáme vše, co komunikuje po počítačové síti, včetně *loopbacku*[1]. Mají něco málo společného s typy uvedenými výše, ale hlavně mají spoustu vlastností, které u předchozích dvou nenajdeme.

Přestože vyvíjíme řadič na platformě síťové karty, chceme, aby ho bylo možné využít i v jiných aplikacích. Proto nebudeme navrhovat ovladač pro síťové, ale pro znakové zařízení.

[1] Loopback ovladač není vázán na žádnou hardwarovou kartu. Pouze vytváří virtuální síťové rozhraní, které všechna odeslaná data pošle zpět do systému



Obrázek 3.3: Konfigurační registry PCI. Žlutě jsou znázorněny povinné položky.

3.3.1 Přístup k PCI zařízení

Abychom mohli komunikovat se zařízením, nejprve ho musíme identifikovat. Slouží k tomu identifikátory v zařízení, které přečte subsystém PCI. Ovladač tyto identifikátory musí obsahovat také, aby operační systém mohl připojit zařízení ke konkrétnímu ovladači.

Samotná komunikace se zařízením pak probíhá pomocí zmíněných portů a paměťových registrů[6, s.413-421]. Jediná věc, kterou ještě potřebuje ovladač zjistit, je adresový prostor zařízení. Popis prostoru se stejně jako identifikátory nalézá v konfiguračních registrech, jak je vidět na obrázku 3.3. U každého PCI zařízení můžeme používat 6 různých oblastí (popsaných v BAR – Base Address Register) v jednom fyzickém adresovém prostoru.

3.3.2 Koherence paměti

Koherentní (přesněji *cache-koherentní*) paměť je ta, kde se obsah hlavní paměti shoduje s cache v procesoru. Když tedy procesor provede instrukci zapisující do hlavní paměti, obsah cache bude mít stejnou hodnotu. Architektura x86 nám naštěstí vždy zajistí, že paměťový prostor pro DMA přenos bude vždy koherentní.

Pro úplnost ještě uveďme, že existuje nekoheretní paměť. Při použití takové paměti je nutné ji explicitně synchronizovat s cache za použití např. *proudového mapování*.

3.3.3 Shrnutí činnosti

Ukázali jsme si několik aspektů, které uplatníme při tvorbě ovladače pro DMA jednotku. Shrňme si hlavní činnost ovladače:

1. Připojit se ke správnému zařízení. Naplnit strukturu podporovaných zařízení.
2. Alokovat paměť. Každý kanál a směr bude mít svůj paměťový prostor.
3. Inicializovat deskriptory. Je to místo v paměti RAM, kam bude Descriptor Manager směrovat svoje požadavky.
4. Spustit řadiče. Správně inicializovat, spustit a řídit DMA řadiče. Předávat data mezi aplikací a hardwarem.
5. Ukončit činnost řadičů. Korektně zastavit přenosy.

Kapitola 4

Implementace

Jednotku DMA jsme implementovali ve čtyřech zdrojových souborech. Dva z nich jsou napsané v jazyce Handel C, provádějí veškeré řízení jednotky. Využívají hlavičkový soubor, který obsahuje konstanty a některé společné definice. Poslední je psaný ve VHDL, který instancuje a propojuje komponenty. Tím zastřešuje celou jednotku.

4.1 Hlavičkový soubor

Hlavičkový soubor `common.hch` obsahuje důležité konstanty a definice společné pro Rx i Tx řadič. Nejprve kontroluje platnost konstanty `DMA_IFC_COUNT`, kterou je nutno definovat pomocí parametru kompilátoru. Následují měnitelné konstanty, jako je velikost hardwarového bufferu, stránky a šířka rozhraní pro generování DMA požadavku. Dále jsou zde konstanty, které měnit nelze, a konstanty vypočítané z obou předešlých typů. Jsou to především výpočty datových šířek pomocí dvojkového logaritmu. Definici konstant zakončuje výčet možných stavů řadiče.

Druhá část souboru obsahuje definici rozhraní, které jsou společná pro oba řadiče. Nejprve musíme definovat výstupní proměnné, pro snadnou orientaci názvům přidáme prefix `ifc_`. A protože Handel C proměnné definované jako `signal` vrací na inicializační hodnotu automaticky v dalším hodinovém taktu, využijeme to pro signál přerušení, či žádosti o DMA přenos. Po definici vstupů hodinových taktů a resetu jsou zapsány `interface`. Ty výstupní mají přiřazen výše definované proměnné. A abychom nemuseli ke vstupním rozhraním vždy přistupovat pomocí operátoru tečka (přímý přístup ke členu struktury), definujeme si druhé jméno, opět s prefixem `ifc_`.

4.2 Rx řadič

Implementaci komponenty najdeme v souboru `rx_dma_ctrl.hcc`. Nejprve vkládáme hlavičkový soubor `common.hch`, a hned potom si dodefinujeme specifická rozhraní.

Následuje definice globálních proměnných. Bude potřeba, aby dva různé bloky mohly přistupovat k jedné proměnné. Přitom ve většině případech bude postačovat možnost v jednom bloku zapisovat i číst a ve druhém jen číst. Obyčejná LUT paměť nám sice neumožňuje požadované chování, naštěstí je ale možné vygenerovat víceportovou paměť. Vygenerují se přitom dvě LUT pro každý bit, namísto jedné u jednoportové paměti. Tímto způsobem deklarujeme všechny ukazatele do kruhových bufferů (kromě `hwStrPtr` k němuž přistupujeme jen v jednom bloku) a řídicí a stavový registr.

4.2.1 NewData proces

Dostáváme se k samotné implementaci. Z pohledu návrhu byl sloučen první a druhý blok, takže vznikl jen jeden proces, který zastává funkci obou bloků. Nicméně z hlediska přehlednosti a podpoření myšlenky paralelizmu byl tento blok rozdělen do několika nezávislých funkcí `main`.

Přijmutí nových dat

První je zmíněná triviální akutalizace registru `hwEndPtr`. V okamžiku příchodu nových dat se k němu přičte jejich délka.

Vygenerování DMA požadavku

Ve druhé funkci je implementována nejsložitější a nejdůležitější část řadiče – sestavení DMA požadavku. Aby k tomu mohlo dojít, je nutné splnit určité podmínky.

Nejprve zjistíme stav aktuálního kanálu:

- Kanál je zastaven. Jsou zakázána přerušení a stahování deskriptorů pro tento kanál.
- Kanál je pouze pozastaven. Další požadavky řadič zatím negeneruje, ale je možné kdykoliv pokračovat.
- Kanál je spuštěn. Sestaví se a vygeneruje DMA požadavek.

První dva stavy nevykonávají v podstatě nic. Takže zbývá poslední stav, jímž se budeme podrobně zabývat v následujících krocích:

1. Povolíme stahování deskriptorů, pokud tomu již tak není. Jestliže řadič nemá platný deskriptor (dojde k tomu při spuštění kanálu nebo zaplnění stránky), pokusí se ho získat. Pokud ale Descriptor Manager neobsahuje žádný deskriptor pro tento kanál, řadič pokračuje s obsloužením dalšího kanálu. Bez platného deskriptoru totiž nemůže postoupit dál.
2. Zjistíme, zda jsou v Rx bufferu neodeslaná data. To provedeme porovnáním hodnoty `hwStrPtr` a `hwEndPtr`. Pokud se shodují, není co přenášet a řadič opět pokračuje obsluhou dalšího kanálu.
3. Nyní vybereme pouze data, která nepřesahují přes hranici stránky. Volné místo ve stránce získáme jako rozdíl konstanty velikosti stránky a části deskriptoru. Délka přenosu bude minimum z délky nových dat a volného místa ve stránce.
4. Dále musíme zkontrolovat, zda buffer v paměti RAM není zcela zaplněn. Jak již víme, může zde nastat problém při zjištění stavu kruhového bufferu. V tomto případě jsme ho vyřešili dostatečně velkými 32 bitovými ukazateli. Nepředpokládáme totiž, že by jeden softwarový buffer mohl mít velikot 4 GiB a více.
5. Teď zbývá sestavit požadavek. Použijeme k tomu paměť, do které zapíšeme celý deskriptor, adresu v Rx Bufferu a délku dat. Vygenerujeme požadavek a vyčkáme, až si ho připojená komponenta převezme.

6. K deskriptoru přičteme délku přenášených dat, abychom mohli příště přenášet data na novou adresu v RAM. Pokud bude mít deskriptor několik prvních bitů nulových (počet závisí na velikosti stránky), znamená to, že jsme zaplnili stránku a zneplatníme deskriptor. K ukazatelům do kruhových bufferů také přičteme délku přenášených dat a do fronty FIFO umístíme informace o přenosu, které potřebuje proces ReleaseData.

Čtení DMA požadavku

Další samostatná funkce vystavuje na datový signál DMA část vygenerovaného požadavku podle žádané adresy.

Počítadlo rozpracovaných požadavků

Aby mohl řadič správně signalizovat svůj stav, musí vědět, kolik požadavků je ještě rozpracovaných. Až dokončí všechny požadavky na jednom kanálu, teprve pak může signalizovat zastavení. V této funkci inkrementuje čítač požadavků při vygenerování a dekrementuje ho při dokončení přenosu.

4.2.2 ReleaseData proces

V okamžiku dokončení DMA přenosu přečteme jednu položku z FIFO a uvolníme pomocí ní délku dat z konkrétního kanálu Rx Bufferu. Následně aktualizujeme `swEndPtr` tímto způsobem:

1. Jestliže není aktivní předchozí `PB_flag`, přenos byl dokončen a začínáme znovu. Inicializujeme tedy pomocnou proměnnou na hodnotu v `swEndPtr`.
2. K pomocné proměnné přičteme délku právě uvolněných dat.
3. Zapamatujeme si hodnotu současného příznaku `PB_flag`.
4. Jestliže tento DMA požadavek nepřesahoval hranici stránky, přepíšeme `swEndPtr` hodnotou dočasné proměnné.

Tento způsob byl implementován z důvodu úspory logiky, protože nevyžaduje použití dvou sčítaček. Nesmí ale dojít ke změně rozhraní při přenosu přes hranici stránky, to nám již zajistil NewData proces.

4.2.3 Interrupt proces

Podobně jako v NewData procesu, i zde procházíme dokola všechny kanály. Jestliže `swEndPtr` překročil hodnotu nastavenou v `intEndPtr`, vygenerujeme přerušení, je-li povoleno a zakážeme jakákoliv další.

V dalším kroku (po kontrole povolení přerušení) pracujeme s časovým limitem, mohou nastat tři možnosti:

- Přišla nová data. V tom případě je čítač nastaven (vynulován) na hodnotu registru `timeout` nastavenou ovladačem.
- Vypršel časový limit. Vygenerujeme přerušení a další zakážeme.
- Nic z toho se nestalo. Pouze dekrementujeme čítač.

4.2.4 Adresový dekodér

Je zde ještě jedna funkce, o které jsme se nezmínili v návrhu – aby bylo možné z ovladače přistupovat k registrům řadiče pomocí rozhraní *MI32*, je nutné implementovat adresový dekodér. Tvoří ho dva podobné bloky *switch-case*, první je proveden při čtení a druhý při zápisu. Adresovaný registr se určí podle prvních 6 bitů adresy, a pro indexaci konkrétního kanálu se použije zbylá část adresy[1]. Zápis má specifický průběh pro některé registry:

- Stavový registr je jen pro čtení
- Do `swEndPtr` nelze zapisovat za běhu řadiče.
- Při zápisu do registrů `intEndPtr` a `timeout` je nutné inicializovat i další pomocné registry přerušení.

4.3 Tx řadič

Podobně jako při návrhu, i v implementaci si uvedeme pouze hlavní rozdíly oproti Rx řadiči.

První část v souboru `tx_dma_ctrl.hcc` – definice globálních proměnných – se téměř shoduje. Tentokrát však i registr `hwStrPtr` je dvouportový.

4.3.1 NewData proces

Data byla odeslána

Opět je zde aktualizován registr, ke kterému se přičte délka nových dat. Registr `hwStrPtr` navíc musíme na začátku inicializovat na velikost bufferu, abychom rozpoznali, zda je kruhový buffer plný nebo prázdný.

Vygenerování DMA požadavku

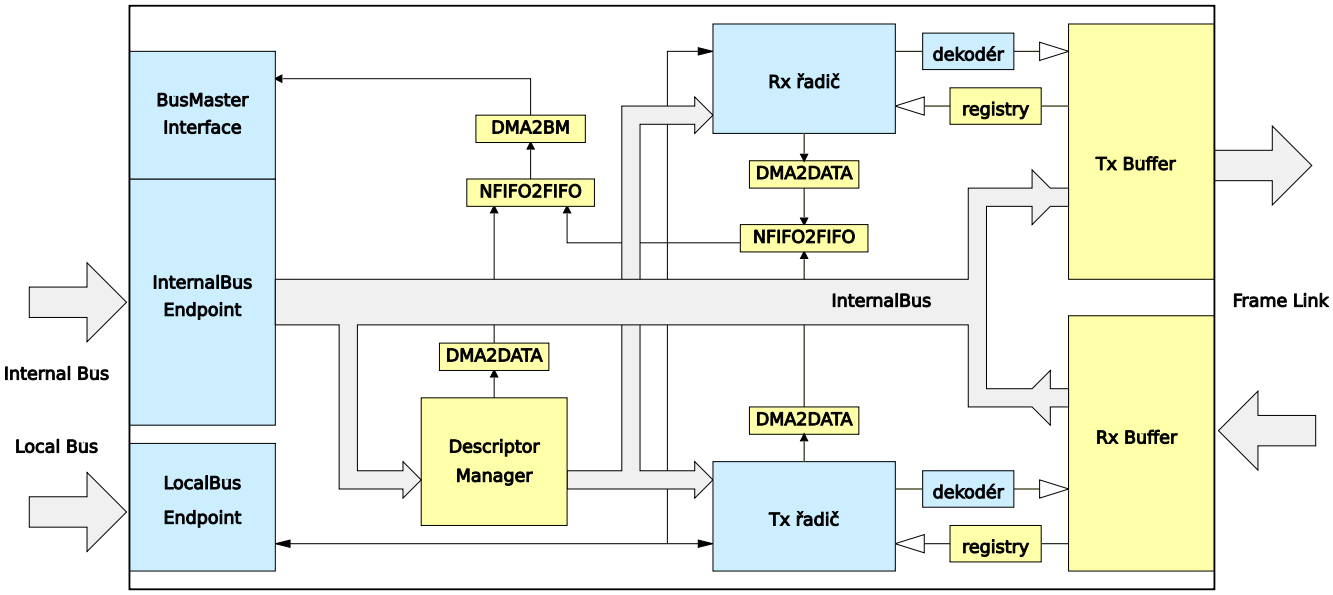
Jako v případě Rx řadiče, i zde budeme stav kanálu obsluhovat podobně. Rozdíl bude jen v sestavení DMA požadavku.

- Zjistíme, zda jsou v RAM data čekající na odeslání. To provedeme porovnáním hodnoty `swStrPtr` a `swEndPtr`.
- Vybereme pouze data, pro které je v Tx Bufferu místo. To získáme jako rozdíl `hwStrPtr` a `hwEndPtr`. Případně data ještě ořízneme tak, aby přenos nepřesahoval stránku.
- Nemusíme rozlišovat, kdy proběhne přenos přes hranici stránky. Odpadne tedy několik kontrol a frotna FIFO nebude obsahovat `PB_flag`.

4.3.2 DataSend proces

Stejně jak tomu je v NewData procesu, i zde odpadá rozlišování rozdělených přenosů. Stačí tedy při dokončení DMA přenosu přečíst jednu položku z FIFO, informovat konkrétní kanál Tx Bufferu o nových datech a aktualizovat `swStrPtr`.

[1] Sedmý bit určuje, zda se jedná o Rx či Tx směr, abychom tyto kanály mohli prokládat.



Obrázek 4.1: DMA jednotka.

4.4 DMA jednotka

Modul tvoří pět hlavních komponent, jak je vidět na blokovém schématu 4.1. Srdcem celé jednotky jsou samozřejmě dva *řadiče DMA*, které jsou propojené se *SW Buffery* a s *Descriptor Managerem*. Aby bylo možné propojit modul s okolím, jsou zde také instance komponent *IB Endpoint* a *LB Endpoint*. A protože jednotlivé komponenty nelze mezi sebou propojit přímo, musíme jejich rozhraní částečně přizpůsobit.

4.4.1 Descriptor Manager

Descriptor Manager je společný pro oba řadiče a nemá dva nezávislé výstupy. Je tedy nutné nějakým způsobem sdílet mezi řadiči nejen datový výstup, ale i řídicí signály.

Signál `ENABLE` je vyveden pro každý kanál zvlášť jak u Descriptor Manageru tak i u obou řadičů. Stačí ho tedy propojit střídavě k Rx a k Tx řadiči. V případě signálu `EMPTY` je ze signálu z Descriptor Manageru multiplexorem vybrán jeden konkrétní bit podle čísla kanálu pro každý řadič.

Druhou částí zajišťující přepínání řadičů je v podstatě volně běžící jednobitový čítač. Podle jeho hodnoty jsou přepínány multiplexory, které přepojují signály buď z Rx nebo Tx řadiče. Na začátek adresového signálu je ještě přidán jeden bit pro rozlišení řadičů.

Problém nastává v situaci, kdy o deskriptor zažádá řadič, který zrovna není připojen. V tom případě si musíme žádost zapamatovat a obsloužit ji v dalším taktu. K tomu slouží dva R-S klopné obvody. Signál `RESET` je aktivován vždy při obsluze řadiče, `SET` když řadič požádá o deskriptor. Aby nedošlo ke ztrátě informace, má signál `SET` vyšší prioritu.

Je zde ještě jedna specialita – protože platný deskriptor přijde přesně o jeden takt později, kdy už je připojený druhý řadič, signál platnosti dat (`DESC_DVLD`) je z multiplexoru připojen k řadičům naopak.

4.4.2 SW buffery

Podobně jako u propojení s Descriptor Managerem, i zde musíme spojit dvě rozdílná rozhraní. SW Buffery mají signály pro každé rozhraní zvlášť, narozdíl od DMA řadičů, kde jsou vždy datové a řídící signály doplněny signálem identifikujícím kanál.

Pro signály `RELLEN` u Rx řadiče a `NEWLEN` u Tx řadiče (oba jsou výstupní) lze propojení realizovat poměrně snadno. Stačí signál platnosti dat vždy připojit na správný kanál. To lze zařídit pomocí *dekodéru 1 z n*.

U signálů jdoucích do řadiče (`NEWLEN` pro Rx a `RELLEN` pro Tx) je situace komplikovanější. Data mohou přijít najednou a každý kanál oznámí nová data současně. Musíme tedy délku dat dočasně přičítat do registrů a podobně jako v řadiči je postupně obcházet a nulovat.

4.4.3 IB a LB Endpoint

I tato rozhraní jsou sdílena některými komponentami:

- Zápisové rozhraní je sdíleno komponentami Descriptor Manager a Tx Buffer.
- Na čtecí rozhraní je připojen Rx Buffer a stavové registry, zaznamenávající přerušení.
- Rozhraní BusMaster sdílí dva DMA řadiče a Descriptor Manager.
- LB Endpoint zpřístupňuje pole registrů Rx i Tx řadiče.

Pro zápisové a čtecí rozhraní je v modulu sada multiplexorů a každý má na starost přepínání určitého signálu. Podle adresy (resp. určitých jejích bitů) musí být IB Endpointu zpřístupněna správná komponenta.

Ke sdílení rozhraní BusMaster je použito několik komponent. Nejprve převedeme DMA rozhraní řadičů pomocí komponenty `DMA2DATA` tak, aby dva tyto datové toky mohla komponenta `NFIFO2FIFO` spojit do jednoho. Podobně jej propojíme i s rozhraním Descriptor Manageru a datový tok, který vznikne spojením všech třech DMA požadavků, připojíme komponentou `DATA2BM` k BusMaster rozhraní IB Endpointu.

Podrobné obsazení adresových prostorů je v tabulce 4.1 a 4.2.

Adresa	Popis	Směr
`0x00000000`	Rx Buffer	↑
`0x00100000`	Tx Buffer	↓
`0x00200000`	Desc. Manager	↓
`0x00280000`	Stavový registr Rx	↑
`0x00280008`	Stavový registr Tx	↑

Tabulka 4.1: Adresový prostor jednotky na sběrnici Internal Bus.

Adresa	Popis	Směr
`0x0000xx00`	Řídící registr	↓↑
`0x0000xx04`	Stavový registr	↑
`0x0000xx08`	`swStrPtr`	↓↑
`0x0000xx0C`	`swEndPtr`	↓↑
`0x0000xx10`	`swBufferMask`	↓↑
`0x0000xx14`	`intEndPtr`	↓↑
`0x0000xx18`	`timeout`	↓↑

Tabulka 4.2: Adresový prostor jednotky na sběrnici Local Bus.

Kapitola 5

Testování

Dokončili jsme implementaci DMA řadiče a celé jednotky. Aby mělo naše snažení nějaký smysl, musíme ověřit, zda jsme jednotku navrhli a implementovali správně. Zbývá tedy otestovat, jestli je jednotka fuknčí a pracuje tak, jak byla navržena.

5.1 Ověření funkčnosti

Simulace probíhala současně s oživením DMA jednotky ve třech částech.

Nejprve byla vytvořena základní kostra Rx řadiče, obsahovala pouze proces NewData pro generování požadavků. K ní byla připojena jednoduchá komponenta, která řadič inicializovala a vygenerovala příchozí data. Následně byl řadič upravován do té doby, než byl schopen data přijmout, korektně zpracovat a vytvořit požadavek na DMA přenos. Pomocná komponenta byla potom pozměněna tak, aby hned po vygenerování požadavku oznámila řadiči dokončení přenosu, takže jsme mohli rovnou odladit proces ReleaseData. Tento stav můžeme vidět na časovém průběhu B.1.

V tomto okamžiku, kdy již Rx řadič vykazoval nějakou aktivitu, bylo zapotřebí vytvořit složitější simulační model, abychom mohli jednoduše generovat a kontrolovat různé transakce po sběrnici. Sestavili jsme jednotku DMA, zatím bez funkčního Tx řadiče. Následně jsme tuto jednotku instancovali v pokročilém simulačním modelu. Ten se skládá z několika generátorů a monitorů pro rozhraní *Frame Link* a hlavní komponenty `IB_BFM`, která simuluje globální prostor, tedy paměť RAM. Umožňuje nám používat jednoduché funkce pro generování zápisu či čtení.

Takto vypadá průběh komplexního testu, podobně by měl pracovat i ovladač:

1. Inicializujeme prostor deskriptorů. Deskriptory jsou čtené z připraveného souboru.
2. Zaplníme prostor pro data k odeslání pakety přečtenými ze soubou.
3. Inicializujeme kanály Descriptor Manageru zápisem na speciální adresu.
4. Povolíme testované kanály řadiče. Zapíšeme do registru `swBufferMask` velikost softwarového bufferu, a pomocí zápisu do řídícího registru kanál spustíme.
5. Spustíme test – vyzkoušíme přenosy přes stránku, generování přerušení oběma způsoby, pozastavení a opětovné spuštění řadiče.

Jak je vidět na časovém průběhu B.2, přijímací část jednotky vypadá funkční.

Přišel na řadu Tx řadič, postup byl obdobný jako u Rx směru. Nejprve jsme zapojili jednoduchou komponentu a ladili základní funkcionalitu. Pak jsme Tx řadič zapojili do DMA jednotky a prováděli komplexnější testy. Nakonec se nám podařilo zprovoznit celou jednotku B.3.

5.2 Předběžná syntéza

Syntéza nám ukázala, nakolik je jednotka náročná pro FPGA. Pro vyjádření náročnosti se používají tyto hodnoty – počet spotřebovaných registrů a LUT a maxiální frekvence obvodu.

Nejprve jsme vysyntetizovali samotné řadiče. V tabulce 5.1 je ukázka využití zdrojů Rx řadičů. Abychom si lépe dokázali představit, kolik to je, na poslední řádek tabulky jsme přidali zdroje vysyntetizované celé jednotky. Problémem bylo vytvoření řadiče pro jedno rozhraní. Tam je zapotřebí speciálních podmíněných bloků.

Další informací, kterou se dozvíme z nástrojů, je maximální možná frekvence jednotky. Hodnota, kterou nám sdělila první syntéza, byla mimořádně nízká. Řadiče DMA totiž nemohly běžet na frekvenci větší než 90 MHz. Spolu s touto informací také syntézní nástroje vypsaly nejpomalejší cestu, takže jsme mohli přibližně zjistit, která ji část kódu vygeneruje. Bohužel kompilátor Handel C vytváří při překladu spoustu proměnných, jejichž jména nevolí úplně nejlépe, takže je potřeba si zdrojový kód a výpisy důkladně projít. Po několika neúspěšných pokusech byl nalezen systém, jakým můžeme navyšovat rychlost obvodu. Dokonce to mělo navíc pozitivní dopad na využité zdroje, což bylo trochu překvapující, očekávali jsme totiž opačný vývoj využití zdrojů při zvyšování frekvence. Nakonec jsme maximální frekvenci zvýšili až na 175 MHz, což je jistě velký úspěch.

Počet kan.	FFD	LUT jako logika	LUT jako paměť
2	248	854	594
4	263	288	596
8	288	995	598
16	327	1136	600
32	414	1381	602
4+4	3198	4996	2139

Tabulka 5.1: Využití zdrojů.

Kapitola 6

Závěr

V rámci této práce byla vytvořena hardwarová jednotka, která uskutečňuje přenosy dat mezi pamětí počítače a periferním zařízením. Uvedli jsme si zde možné způsoby přenosu dat. Dále jsme si představili cílovou platformu NetCOPE a její komponenty důležité pro sestavení celé jednotky. Provedli jsme návrh řadičů, a seznámili se se základními principy tvorby ovladačů periferních zařízení v operačním systému Linux. Nejvíce jsme se zabývali implementací dvou řadičů v jazyce Handel C, které jsme následně umístili do modulu spolu s dalšími potřebnými komponentami. Simulací jsme ověřili správnou funkčnost jak samotného řadiče, tak celé jednotky. Nakonec jsme syntézou zjistili hlavní parametry jednotky.

Vzhledem k tomu, že vytvořená jednotka byla otestována pouze jednoduchou simulací, měl by nyní následovat verifikační proces, který odhalí případné doposud skryté chyby. Je potřeba řadiče upravit tak, aby bylo možné vygenerovat jednotku i s jedním rozhraním. Dále je možno jednotku více zoptimalizovat. I když je zde vidět jasný pokrok v optimalizaci rychlosti (z původních 90 MHz na 175 MHz), přesto je možné dále snižovat využití zdrojů.

Literatura

[1] Anon.: Flat memory model. [online].
URL http://en.wikipedia.org/wiki/Flat_memory_model

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

[3] Gibbons, T.: The Arbitration and Data Transfer Mechanisms. Technická zpráva, 2002.
URL http://www1.pacific.edu/~mgibbons/PCI/index.html

[4] Gorman, M.: *Understanding the Linux Virtual Memory Manager*. Upper Saddle River, New Jersey: Pearson Education, Inc., 2004, ISBN 0-13-145348-3.

[5] Hennesy, J.; Patterson, D.; Goldberg, D.: *Computer Architecture - A Quantitative Apporach*. Morgan Kaufmann, 2004, ISBN 978-1558605961.

[6] Jelínek, L.: *Jádro systému linux*. Brno: Computer Press, 2008, ISBN 978-80-251-2084-2.

[7] Kobiersky, P.: Interconnection system project. [online].
URL http://www.liberouter.org/vhdl_design/gen-svn/ICS_info.php

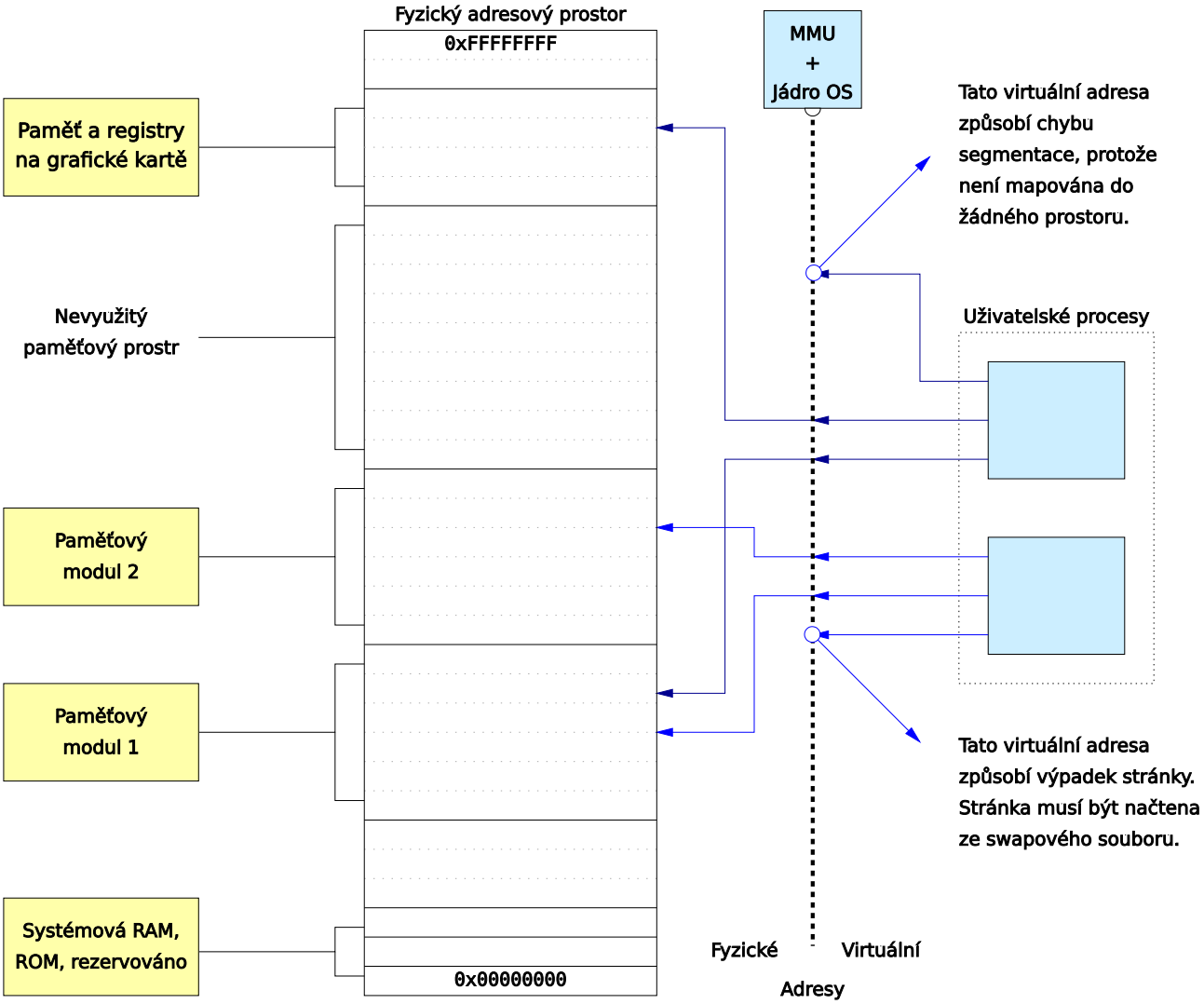
[8] Messmer, H.-P.; Dembowski, K.: *Velká kniha hardware*. Brno: Computer Press, 2005, ISBN 80-251-0416-8.

Seznam příloh

- Příloha A. Schémata a nákresy.
- Příloha B. Časové průběhy signálů.
- Příloha C. CD se zdrojovými kódy a elektronickou verzí této práce.

Příloha A

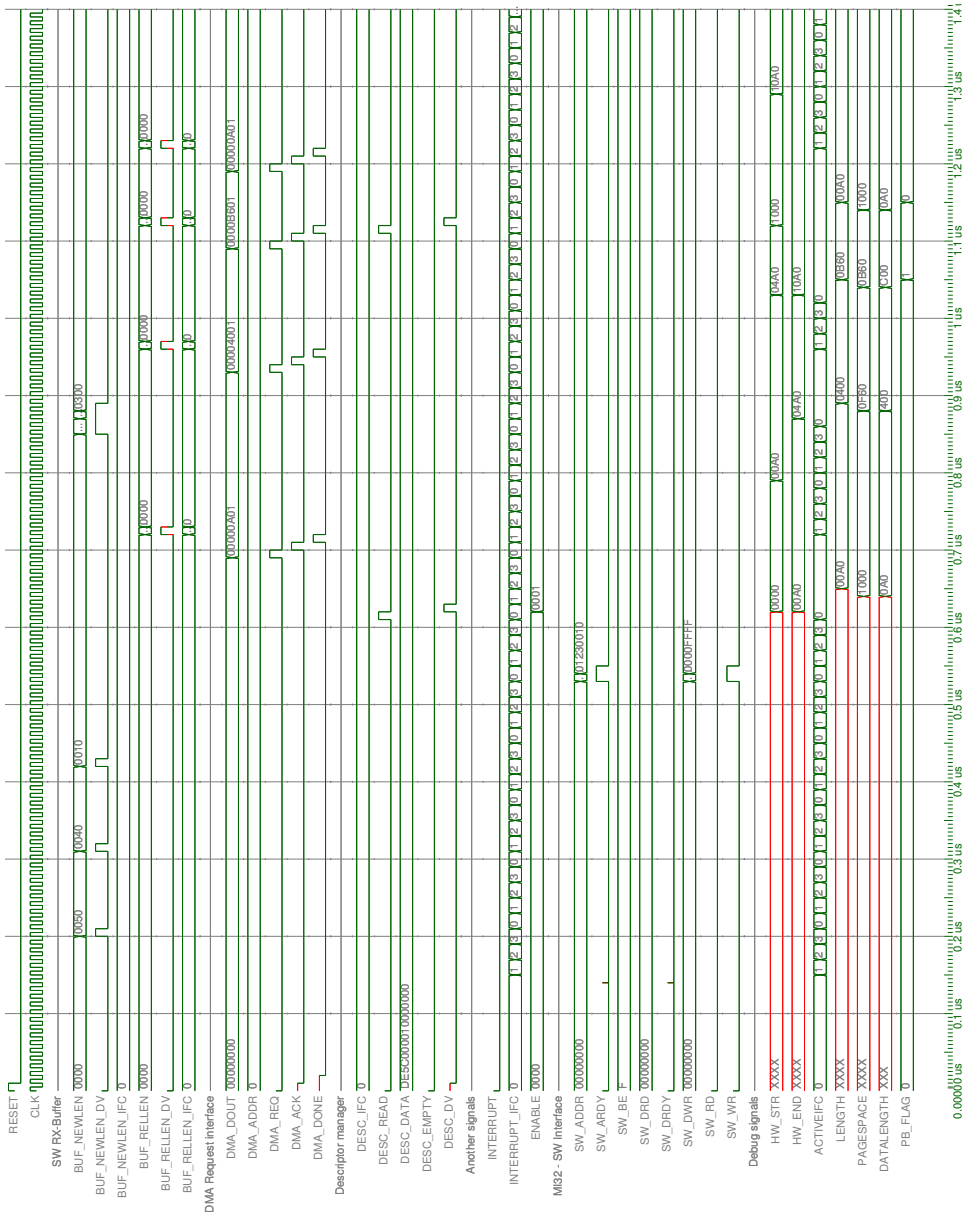
Schémata a nákresy



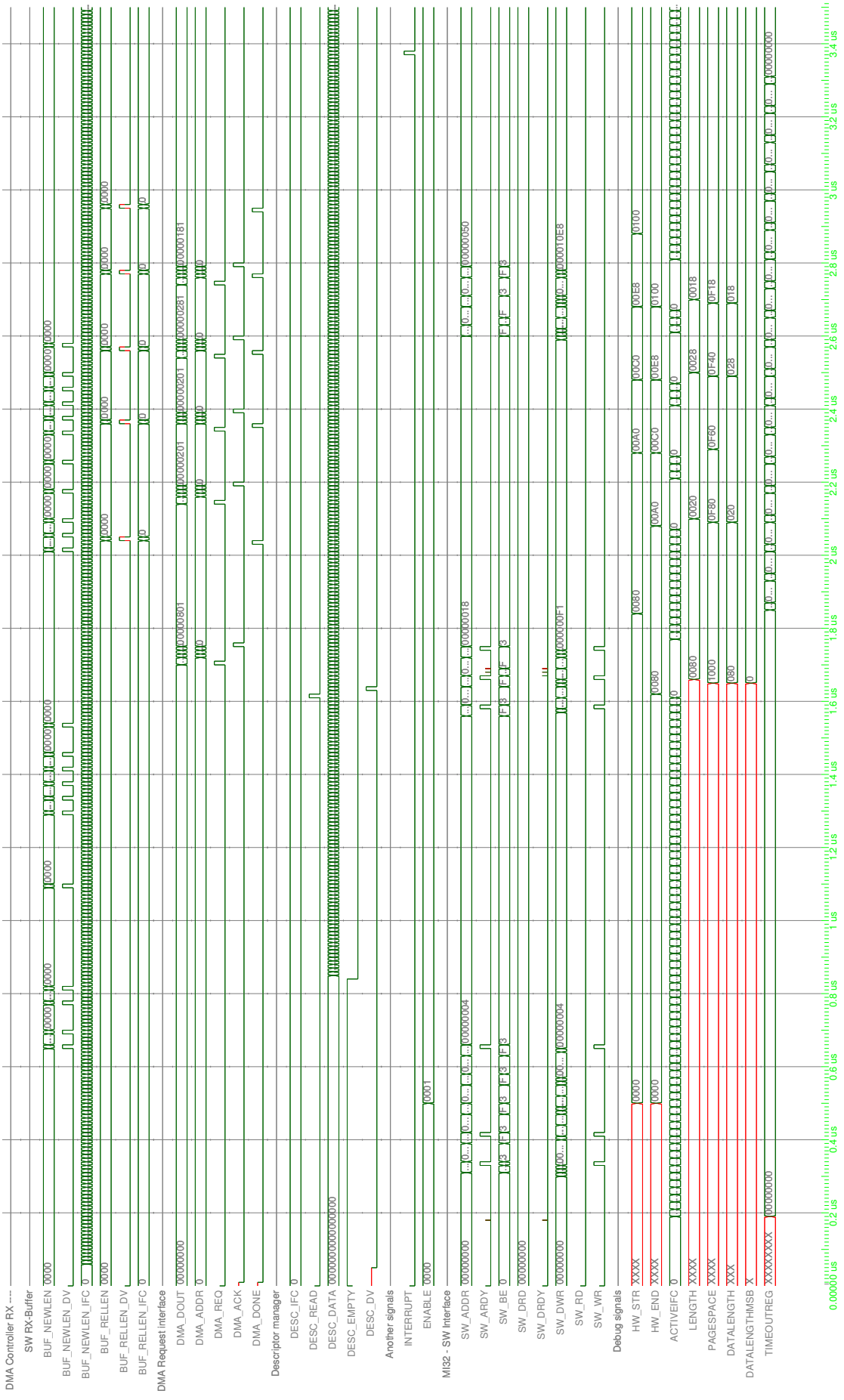
Obrázek A.1: Paměťový model platformy x86 a moderního operačního systému.

Příloha B

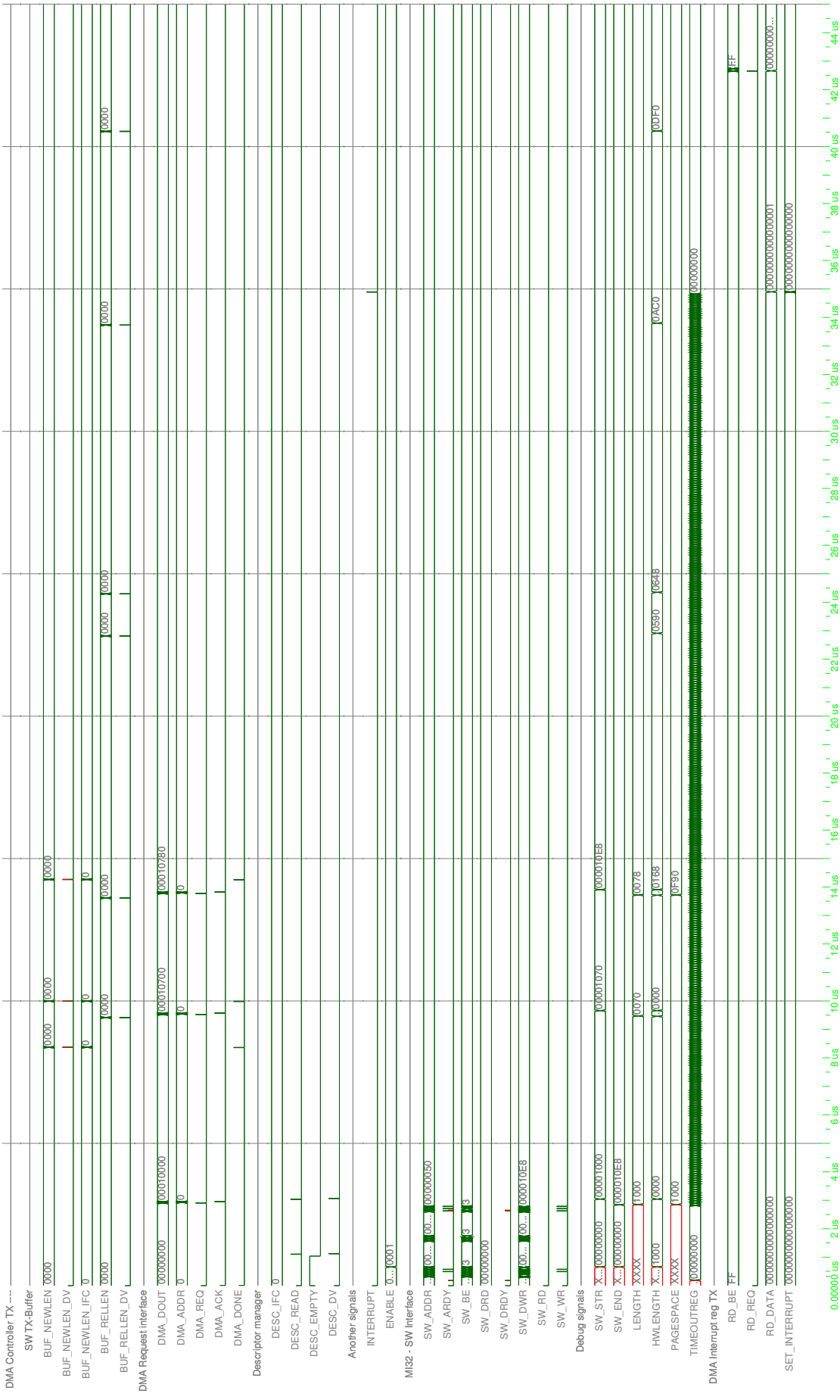
Časové průběhy signálů



Obrázek B.1: Průběhy signálů při simulaci Rx řadiče.



Obrázek B.2: Průběhy signálů při simulaci celého DMA modulu, Rx řadič.



Obrázek B.3: Průběhy signálů při simulaci celého DMA modulu, Tx řadič.