VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

VYSOKORYCHLOSTNÍ PAKETOVÉ DMA PŘENOSY DO FPGA

HIGH-SPEED PACKET DATA DMA TRANSFERS TO FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **JAN KUBÁLEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MARTÍNEK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Kubálek Jan, Bc.**
Program: Informační technologie Obor: Počítačové a vestavěné systémy
Název: **Vysokorychlostní paketové DMA přenosy do FPGA**
High-Speed Packet Data DMA Transfers to FPGA
Kategorie: Počítačová architektura

Zadání:

1. Seznamte se s technologiemi FPGA, PCI-Express, XDP a DPDK.
2. Navrhněte firmwarový modul pro přenos paketových dat ze systémové paměti RAM do akcelerační karty s FPGA. Při návrhu se zaměřte na dosažení co nejvyšší přenosové rychlosti.
3. Navržený firmwarový modul implementujte.
4. Ověřte funkční a výkonové vlastnosti modulu v simulacích a v čipu FPGA.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Martínek Tomáš, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 20. května 2020
Datum schválení: 25. října 2019

Abstrakt

Tato práce se zabývá návrhem, implementací, testováním a měřením firmwarového modulu pro čip FPGA, který zajišťuje DMA přenosy síťových dat z RAM počítače do samotného čipu na síťové kartě. Tyto přenosy jsou prováděny přes sběrnici PCIe rychlostí až 100 Gb/s s možností podpory rychlostí 200 Gb/s a 400 Gb/s. Cílem této technologie je umožnit zpracování síťového provozu za účelem údržby páteřních uzlů sítě a datových center. Modul je současně navržen tak, aby jej bylo možné použít na různých typech FPGA čipů a to především od firem Xilinx a Intel.

Abstract

This thesis deals on the design, implementation, testing and measuring of a firmware module for FPGA chips, which enables DMA transfers of network data from computer RAM to the FPGA chip placed on a network interface card. These transfers are carried out using a PCIe bus on the speed of up to 100 Gbps with the possible support of speeds 200 Gbps and 400 Gbps. The goal of this technology is to allow network data processing for the purpose of maintenance of backbone network nodes and data centers. The module is designed so it can be used on different types of FPGA chips, mainly those produced by companies Xilinx and Intel.

Klíčová slova

FPGA, DMA, PCIe, Ultrascale+, Stratix10

Keywords

FPGA, DMA, PCIe, Ultrascale+, Stratix10

Citace

KUBÁLEK, Jan. *Vysokorychlostní paketové DMA přenosy do FPGA*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Martínek, Ph.D.

Vysokorychlostní paketové DMA přenosy do FPGA

Prohlášení

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

.....
Jan Kubálek
22. května 2020

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Tomáši Martínkovi Ph.D. za jeho pomoc a podporu při psaní této práce. Dále děkuji svým kolegům z výzkumného týmu Liberouter CESNET, zejména pak Ing. Martinu Špinlerovi a Ing. Jakubovi Cabalovi za jejich odbornou konzultaci při firmwarovém vývoji a Ing. Pavlovi Benáčkovi Ph.D. za konzultaci při psaní textu této práce.

Obsah

Kapitola 1

Úvod

V počítačových technologiích dochází v dnešní době k neustálému rozvoji směřujícímu ke zvyšování výpočetního výkonu a rychlosti zpracování dat. Jednou z oblastí, kde je rychlost zpracování dat kritická z důvodu přirozeného zvyšování jejich objemu, je oblast počítačových sítí. Protože od vzniku Internetu také neustále vzrůstá i komplexita sítí jako takových, rostou i nároky na operace potřebné pro jejich údržbu. Pro zajištění bezporuchového běhu důležitých uzlů i koncových stanic je zapotřebí zajišťovat průběžný monitoring stavu jejich propojení a počítat statistiky datových přenosů. Často jsou vyžadovány mechanismy pro ochranu před útoky a pro filtrování provozu. Na páteřních uzlech a v datových centrech dosahuje přitom již v dnešní době rychlost datových přenosů řádů stovek gigabitů za sekundu. Na takové rychlosti jsou výpočetně náročné už i úlohy potřebné pro běžný provoz sítě, jako je například směrování.

Zařízení, která jsou součástí počítačové sítě, musí být proto vysoce výkonná a navíc vyžadují, aby co největší část jejich práce byla hardwarově akcelerována. Sdružení CESNET, které zajišťuje síťové propojení mezi akademickými institucemi v České Republice, se z toho důvodu zabývá vývojem akceleračních síťových karet, které umožňují řešit výpočetně náročné úlohy spojené s provozem sítě na vysokých rychlostech. Tyto síťové karty využívají pro akceleraci výpočtů výkonné FPGA (*Field Programmable Gate Array*) čipy. Tato platforma umožňuje dosažení výkonu dostačujícího pro zpracování dat na vysokých rychlostech a zároveň poskytuje flexibilitu opakované rekonfigurace pro urychlení vývoje a pro podporu dodatečných funkcionalit pro specifické případy užití.

V rámci této práce je mým úkolem navrhnout, implementovat a otestovat DMA modul (*Direct Memory Access*) pro FPGA čip na síťové kartě, který bude zajišťovat vysokorychlostní přenosy síťových paketů z paměti počítače na čip. Jedná se tedy pouze o TX část celého DMA modulu. Část ve směru přijímání dat ze sítě do počítače (tedy ve směru RX) není součástí této práce.

Navržený DMA modul musí umět podporovat přenosovou rychlost 100 Gb/s s možností snadného rozšíření na 200 Gb/s a 400 Gb/s. K přenosu mezi síťovou kartou a pamětí počítače je přitom použito rozhraní PCIe Gen3, případně Gen4, (*Peripheral Component Interconnect Express Generation 3, Generation 4*), TX DMA modul proto bude komunikovat právě s tímto rozhraním. Dalším důležitým požadavkem je podpora vysokého počtu DMA kanálů. DMA kanál je virtuální kanál, který je z pohledu software viděn jako jedno síťové rozhraní. Účelem DMA kanálů je možnost rozdělit síťový provoz na síťové kartě na více částí. Každá tato část je pak v počítači zpracovávána na odděleném jádře procesoru nebo přímo na speciálním virtuálním stroji. Tím dochází k rozložení zátěže na procesorech.

Pro DMA modul to ale představuje komplikace, neboť každý DMA kanál musí být možno řídit ze software zcela nezávisle.

Rozbor technologií, kterých se tato práce týká je uveden v kapitole 2. V této kapitole jsem se také zaměřil na zhodnocení různých přístupů provádění DMA přenosů se snahou o dosažení co nejlepšího využití propustnosti PCIe rozhraní. Kapitola 3 krátce shrnuje aktuální stav technologií v této oblasti srovnatelných s touto prací. Kapitola 4 obsahuje hrubý popis navržené architektury TX DMA modulu a vysvětlení některých klíčových prvků. V kapitole 5 jsou následně uvedeny některé implementační detaily a podrobné návrhy vybraných komponent TX DMA modulu. Postup funkční verifikace a testování modulu jsou uvedeny v kapitole 6. Následné měření výsledného modulu z pohledu spotřeby zdrojů na FPGA a dosažitelné propustnosti je obsahem kapitoly 7.

Kapitola 2

Teoretický rozbor

Protože část ve směru RX není součástí zadání této práce, budu dále označovat termínem *DMA modul* právě TX směr DMA modulu, pokud nebude výslovně řečeno jinak.

Tato kapitola obsahuje popis technologií DMA a PCIe. Z těchto technologií jsou zde popsány především aspekty významné z pohledu DMA modulu, který provádí DMA přenosy z FPGA čipu na síťové kartě právě přes sběrnici PCIe. Dále jsou v rámci sekcí 2.2, 2.3, 2.4, 2.5 a 2.6 popsány některé přístupy používané pro přenos síťových dat pomocí DMA. U jednotlivých přístupů jsou popsány dopady na složitost řízení přenosů z pohledu software a hardware a na schopnost využití plné propustnosti sběrnice PCIe. Na závěr této kapitoly jsou v sekci 2.7 definována rozhraní používaná pro přenos dat mezi DMA modulem a okolními komponentami v FPGA.

DMA

DMA je technologie přenosu dat v počítači mezi periferním zařízením a RAM (*Random Access Memory*) samotného počítače bez nutnosti přímého řízení ze strany procesoru. Jde o významný prvek při snaze o urychlení přenosu dat mezi periferním zařízením a pamětí a zavedení autonomního vykonávání operací periferním zařízením. Detailnější popis této technologie lze najít například v [10].

Při přenosu dat z paměti RAM do periferního zařízení si nejprve software vyhradí prostor v paměti a následně do něj uloží data určená k odeslání. Poté předá „deskriptory“ popisující tento prostor řadiči periferního zařízení (DMA modulu) a spustí jej. Řadič periferního zařízení (v našem případě síťové karty) následně začne posílat po sběrnici, která jej spojuje s RAM, čtecí požadavky (transakce) obsahující žádosti o čtení dat. V odpovědi na žádosti získá zpátky transakce obsahující přečtená data, která musí správným způsobem interpretovat a zpracovat (v případě síťové karty odeslat jako pakety do sítě).

2.1 DMA přenosy přes PCIe

Síťové karty, které jsou cílovou platformou pro tento projekt, jsou připojeny k počítači přes sběrnici PCIe. Tato sběrnice umožňuje provozovat přenosy na několika paralelních linkách. Protokol přenosu je přitom rozdělen do několika vrstev podobně jako je tomu například u TCP/IP modelu. To vše za účelem dosažení co nejvyšší přenosové rychlosti. U PCIe Gen3 dosahuje tato propustnost na 16 linkách více než 100 Gb/s a pro PCIe Gen4 je to již přes 200 Gb/s. Dosažení tak vysoké propustnosti, ale není vždy zaručeno. Aby mohl DMA modul přenášet z RAM na FPGA data touto rychlostí musí se snažit minimalizovat

režii vznikající pro každou přenosovou transakci na sběrnici a také musí dodržovat omezení daná specifikací sběrnice. Specifikace z oblasti PCI jsou dostupné na webové stránce konsorcia PCI-SIG [9].

Při odesílání čtecích a zápisových požadavků na PCIe musí žadatel dodržovat dvě hlavní pravidla:

1. Požadavek nesmí překročit maximální délku čtených (nebo zapisovaných) dat stanovenou sběrnicí

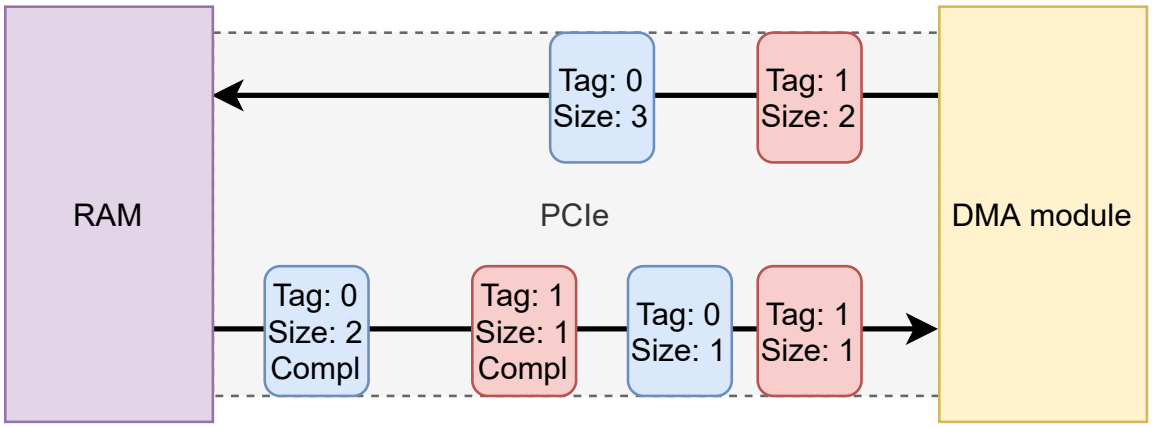
2. Požadavek směřující do RAM nesmí překračovat hranici dvou stránek v paměti

Maximální délka požadavku bývá obvykle 256 B pro zápis a 512 B pro čtení. Tyto hodnoty se obvykle označují jako *MPS* (*Maximum Payload Size*) a *MRRS* (*Maximum Read Request Size*). Omezení na překračování hranic stránek vychází z toho, že počítač může obsahovat více paměťových modulů. Protože tyto moduly jsou mapovány do jednoho adresového prostoru, tak požadavek směřující do dvou různých paměťových stránek může ve skutečnosti směřovat do dvou různých modulů. Sběrnice PCIe pak nedokáže takový požadavek zpracovat.

DMA modul musí také umět správně přijmout odpovědi na čtecí požadavky. Pokud chceme dosáhnout co nejvyšší propustnosti na PCIe, musíme odesílat čtecí požadavky v takzvaném „Relaxed“ režimu. V tomto režimu jsou čtecí požadavky vystavené na sběrnici vykonávány mimo pořadí (out-of-order), aby se maximalizovala rychlost přenosů. Odpověď na každý požadavek navíc nemusí dorazit zpět jako jedna transakce, ale může být rozdělena na více částí. Části jedné odpovědi přicházejí již vzájemně v pořadí (in-order) a poslední z nich je označena příznakem „Completed“ umístěným v hlavičce této transakce. Aby bylo možno od sebe rozlišit odpovědi na jednotlivé požadavky má každý požadavek při odeslání přidělenou hodnotu „Tag“. Stejná hodnota políčka Tag je pak v hlavičce všech částí jeho odpovědi. Nový požadavek se stejnou hodnotou políčka Tag lze znovu odeslat až po přijetí odpovědi s odpovídajícím Tagem a s nastaveným příznakem Completed, aby byla hodnota políčka Tag unikátní v rámci všech požadavků čekajících na odpověď. Tyto principy jsou demonstrovány v příkladu na obrázku 2.1.

Další omezení pro komponenty používající PCIe je zarovnání adresy a délky požadavků. Sběrnice PCIe pracuje vnitřně pouze s bloky dat o velikosti 4 B a adresami zarovnanými na 4 B. Adresa i délka požadavku musí proto vždy být zarovnána na 4 B. Pokud chce žadatel přečíst nebo zapsat data, která nejsou takto zarovnána, může využít speciální sady bitů v hlavičce požadavku „First Byte Enable“ a „Last Byte Enable“. Pokud například pošleme zápisový požadavek o délce 8 B z adresy *0x10* a hodnotu First Byte Enable nastavíme na *1000* a hodnotu Last Byte Enable na *0001*, pak ve skutečnosti provedeme zápis pouze dvou bajtů na adresu *0x13*. Po sběrnici se ale bude přenášet celých 8 B. U zápisových požadavků lze tímto způsobem zamezit přepsání jednotlivých bytů v paměti. Pro čtecí požadavky se informace o platnosti bajtů v rámci první a poslední čtveřice odráží v hodnotách Byte Enable v hlavičce odpovědi. Zde platí, že první část odpovědi na určitý čtecí požadavek bude mít stejnou hodnotu First Byte Enable, jako samotný požadavek. A podobně poslední část odpovědi bude mít zase hodnotu Last Byte Enable stejnou jako požadavek. Jde tedy pouze o doplňující informaci, kterou není třeba používat, pokud si přesnou adresu a délku pamatuje jednotka, která požadavek vytvořila.

Pro ovládání sběrnice PCIe poskytují výrobci FPGA čipů rozhraní, na která se lze připojit z komponent vytvořených v konfigurovatelné části čipu. Toto rozhraní se nazývá „PCIe Endpoint“ a slučuje zpravidla několik linek PCIe do jediného proudu datových rámců. PCIe



Obrázek 2.1: Demonstrace out-of-order vykonávání čtecích požadavků na sběrnici PCIe. V tomto případě nejprve DMA modul odešle čtecí požadavek s hodnotou políčka Tag 0 pro čtení dat o velikosti 3 a následně požadavek s hodnotou Tag 1 a velikostí 2. S určitým, předem neznámým zpožděním se zpět z RAM vrací odpovědi na tyto požadavky. Oba požadavky se ale rozdělili na 2 části, kde až ta druhá má nastavený příznak „Completed“. Zároveň došlo k náhodnému prohození jejich pořadí. Teprve po přijetí odpovědí s příznakem „Completed“ může DMA modul vyslat znovu čtecí požadavky s hodnotami Tagu 0 a 1.

Endpoint je přístupný přes rozhraní pro proudový přenos, které je specifické pro daného výrobce. U FPGA od firmy Xilinx je to nejčastěji rozhraní AXI (*Advanced eXtensible Interface*) [2]. U čipů vyráběných firmou Intel se jedná o sběrnici Avalon [1]. Konkrétní parametry rozhraní, jako je datová šířka, frekvence, zarovnání dat a jiné, se navíc mohou lišit pro jednotlivé modely FPGA čipů, jednotlivé generace PCIe sběrnice a jednotlivá nastavení PCIe Endpointu. Aby byl DMA modul od těchto odlišností odstíněn jsou pro naše síťové karty vytvořeny komponenty, které obsluhují jednotlivé typy těchto PCIe Endpointů a převádějí je na naše vlastní rozhraní, která jsou blíže popsána v sekci 2.7.

Požadavky cestující mezi PCIe a FPGA čipem se dělí na dvě skupiny:

1. Požadavky iniciované z FPGA čipu (čtení a zápis dat DMA modulem)
2. Požadavky iniciované z CPU (čtení a zápis vnitřních registrů v FPGA ovladačem síťové karty)

S požadavky z druhé skupiny se pracuje zcela jinak než s těmi z první skupiny, neboť se nejedná o DMA přenosy dat a proto zde nejsou nijak velké nároky na propustnost. Z toho důvodu se tyto dva typy požadavků často rozdělují do dvou toků a zpracovávají se nezávisle. Toto rozdělení podporuje v některý případech přímo PCIe Endpoint, ale není to pravidlem a proto s tím při návrhu obecného DMA modulu nelze počítat.

2.2 DMA přenosy síťových dat

Přenosy dat z počítače do síťové karty musejí být plně řízeny softwarovým ovladačem. Pro každý jeden DMA kanál je spuštěna jedna instance tohoto ovladače a to zpravidla na samostatném jádře procesoru (pro dosažení maximálního výkonu). Pro uživatele (aplikaci, která chce síťovou kartu použít) zařizuje ovladač detekci síťové karty na PCIe rozhraní

a vytvoření síťových rozhraní pro příjem a odesílání dat. Pro zařízení (DMA modul) zajišťuje ovladač spouštění a zastavování jednotlivých DMA kanálů, jejich nastavení a průběžné poskytování deskriptorů na data k odeslání. Fáze nastavení probíhá vždy před samotným spuštěním DMA kanálu. V této fázi zapisuje ovladač do registrů DMA modulu informace potřebné pro běh. Je to především adresa a velikost bufferu v paměti obsahujícího deskriptory. Každý z těchto zápisů do registrů v DMA modulu znamená odeslání požadavku z CPU do FPGA přes sběrnici PCIe. Ovladač má přitom adresový prostor všech registrů v FPGA na kartě mapovaný do adresového prostoru paměti, takže tyto operace provádí instrukcemi pro práci s pamětí.

To, které konkrétní parametry je třeba DMA modulu předat a jak bude probíhat průběžné řízení přenosů, záleží na přístupu k přenosům, který chceme v ovladači implementovat. Těch existuje více. Jednotlivé přístupy se výrazně liší výpočetní náročností v software a složitostí implementace v hardware. Zvýšená složitost v hardware v tomto případě znamená často i nižší dosažitelnou propustnost na PCIe sběrnici z důvodu jejích výše popsaných omezení. Tyto přístupy jsou popsány v následujících sekcích této kapitoly. První sekce 2.3 se přitom zabývá obecně prací se síťovými daty z pohledu software a kategorizací jednotlivých přístupů.

2.3 Zpracování síťového provozu

U klasických síťových aplikací, které nejsou příliš náročné z pohledu datové propustnosti a které jsou provozovány na běžných osobních počítačích, se přenosy paketů provádějí přes síťový stack (například TCP/IP) implementovaný částečně na síťové kartě a částečně v kernelu operačního systému. Tato implementace zpracovává pakety jeden po druhém a s pouze minimální – nebo žádnou – úrovní paralelizmu. Pro pohodlí vývojáře uživatelského programu (aby nemusel pracovat se složitým rozhraním pro sdílení paměti, DMA koherencí a podobně) jsou pakety zpracovávány v odděleném paměťovém prostoru a mezi aplikací a kernelem musejí být kopírovány.

Pokud chceme ale dosáhnout maximálního výkonu, je zpravidla užitečné, když dokážeme aplikací přijímat a odesílat pakety, aniž by musely procházet síťovým stackem operačního systému a rovněž chceme minimalizovat kopírování dat paketů v paměti. Řada konceptů pro vysokorychlostní přenosy proto umožňuje aplikacím pracovat se stejným prostorem v paměti, jako se kterým pracuje DMA modul na síťové kartě. Tímto způsobem pracují všechny systémy popsané níže v této kapitole. Pouze systém XDP (*eXpress Data Path*) v sekci 2.6 umožňuje některé pakety zpracovávat přes kernel a některé přímo v aplikaci.

Systémy pro odesílání a příjem síťových dat jsou děleny na „proudové“ a „paketové“. V případě proudových přenosů jsou pakety uloženy v jednom (virtuálně) souvislém bufferu v paměti. Díky tomu lze v jedné čtecí PCIe transakci přenášet data více paketů současně a tím se snižuje režie PCIe přenosů a zvyšuje se propustnost. Příkladem tohoto přístupu je systém NDP (*Netcope Data Plane*) popsaný v sekci 2.4.

U přenosů paketových mohou být jednotlivé pakety uloženy na různých místech v paměti. To znamená, že přenosy na PCIe se musejí provádět pro každý paket zvlášť a žádný požadavek nemůže číst data více než jednoho paketu. Tento přístup však může přinášet jisté výhody z pohledu software. Především z pohledu paralelizace zpracování paketů. Příklady systémů implementujících paketové přenosy jsou uvedeny v sekcích 2.5 a 2.6.

2.4 NDP

Princip NDP spočívá v proudovém přenášení dat paketů uložených v souvislém bufferu v paměti. Pakety musejí být do tohoto bufferu uloženy v těsném sledu za sebou (pouze se zarovnáním na 8 B). NDP poskytuje knihovnu, která umožňuje sdílet paměťový prostor tohoto bufferu mezi ovladačem síťové karty a aplikacemi, které přijímají či odesílají data. Díky tomu nedochází ke kopírování dat při přenosu mezi aplikací a ovladačem.

Nevýhodou však je, že při přijímání dat ze síťové karty (čtení paketů z datového bufferu aplikací) je pozice začátku každého paketu závislá na délce všech paketů předcházejících. Každá aplikace, která čte pakety uložené v bufferu musí nejprve přečíst délku jednoho paketu (délka je uložena v prvních 2 bajtech paketu) a teprve poté se dozví, kde leží začátek paketu následujícího. Z toho důvodu nelze proces zpracování jednotlivých paketů paralelizovat ani vektorizovat. Při odesílání paketů je tato režie ještě zvýšena tím, že několik aplikací vkládá současně pakety do jednoho bufferu a proto je třeba jejich vzájemný přístup synchronizovat, čímž dochází k serializaci odesílání napříč všemi aplikacemi. K této serializaci by sice stejně muselo dojít, neboť data z bufferu jsou čtena DMA modulem paket po paketu, ale na úrovni přístupu z aplikace do paměti vzniká navíc i režie se zamykáním a odemykáním zámků bufferu. Tato režie je vyšší, než kdyby například měla každá aplikace svůj vlastní buffer a DMA modul by se měl rozhodovat, ze které z aplikací odešle příští paket.

Navíc si lze představit, že proces vytvoření paketu aplikací přímo v prostoru bufferu může být výpočetně a časově náročnější, než samotné kopírování paketu z jiného místa v paměti nebo v cache. Toto časové zpoždění bude mít pak za následek snížení propustnosti i pro ostatní aplikace, které odesílají data na stejném rozhraní. Žádná další aplikace totiž nemůže začít vytvářet v bufferu svůj vlastní paket dokud není známo, jak velký prostor zabírá paket oné první aplikace.

Lze si to představit na následujícím příkladu:

Mějme 10 aplikací, kde každá chce odeslat 1 paket. Vytvoření jednoho paketu trvá 10 časových jednotek. Zkopírování paketu trvá ale pouze 1 časovou jednotku. Pokud použijeme přímý přístup do odesílacího bufferu, pak jednotlivé aplikace budou muset své pakety vytvářet postupně a celková doba odesílání bude $10 * 10 = 100$ časových jednotek. V případě, že si ale každá aplikace v první fázi vytvoří svůj paket ve svém vlastním bufferu a v další fázi odesílání jej pouze zkopíruje do odesílacího bufferu, pak první fázi mohou všechny aplikace provádět paralelně a sekvenční část bude obsahovat pouze kopírování. Celkově pak bude doba potřebná pro odesílání $10 + 10 * 1 = 20$, tedy 5krát méně. Jedinou nevýhodou budou větší paměťové nároky a o něco větší množství vykonané práce (kvůli kopírování, které se v první variantě nemuselo provádět).

Systém NDP byl vyvinut v rámci sdružení CESNET a to za účelem dosažení maximální propustnosti na PCIe bez ohledu na délku paketů (což se podařilo již předchozím verzím DMA modulu). Jeho použití v praxi je ale problematické z důvodu nemožnosti paralelizace zpracování jednotlivých paketů a absence dodatečné podpory u běžných softwarových nástrojů.

2.5 DPDK

DPDK (*Data Plane Developement Kit*) označuje sadu knihoven, umožňující uživatelské aplikaci a ovladači síťové karty pracovat se stejným paměťovým prostorem pro ukládání paketů. DPDK je na rozdíl od NDP systém pro přenosy paketové. Jednotlivé odesílané

pakety mohou být v rámci tohoto systému odesílány z náhodných míst v paměti. Některé pakety zde mohou být navíc rozděleny na více částí, pokud je uživatel takto vytvořil. V tom případě musí DMA modul číst každou tuto část zvlášť. To vede na výrazné snížení propustnosti na PCIe sběrnici, zvláště pokud jsou pakety rozděleny na části menší, než je MRRS. Ke snížení propustnosti dojde i v případě, kdy jsou pakety větší než MRRS a nejsou rozděleny na více částí, ale jejich délka není násobkem MRRS. Každý požadavek, který čte poslední část paketu, pak bude mít velikost menší než je MRRS.

Výhodou paketových přenosů v DPDK z pohledu používání PCIe je to, že DPDK používá paměťové adresy zarovnané na 4 B a pokud je paket rozdělen na více částí, pak i délka těchto částí je násobkem 4 B (kromě poslední části, kde délka závisí na délce paketu). Tím je zaručeno, že přečtená data přicházející z PCIe sběrnice do DMA modulu budou vždy zarovnaná, a díky tomu je lze snadněji zpracovávat.

Kompletní dokumentace k DPDK je dostupná na [8].

2.6 XDP

Systém XDP je primárně určen pro urychlení zpracování paketů přicházejících ze sítě do softwaru (směr RX). XDP umožňuje předložit kernelu jednoduchou kompilovanou aplikaci, které pro každý paket vyhodnotí, jakým způsobem s ním má být naloženo. Paket může být předán přímo uživateli, nebo může projít klasickou cestou přes síťový stack. Dále jej lze nechat přímo v kernelu zahodit, nebo přeposlat rovnou na síťový výstup (TX). Podobně jako u DPDK, i zde si mohou ovladač a aplikace předávat pakety bez kopírování dat. Příkladem systému využívajícího tento postup je eBPF (*extended Berkeley Packet Filter*), který je součástí aplikací jako například *tcpdump* a který slouží k filtrování paketů pomocí komplexních výrazů.

Další výhodou XDP je možnost připojení uživatelských aplikací přímo z virtuálního stroje bez režie způsobené virtualizací. Díky tomu lze síťová data zpracovávat ve virtuálním stroji bez potřeby jejich přenášení přes operační systém fyzického stroje. To je velmi žádoucí v celé řadě případů zpracování síťových dat. Distribuci dat na jednotlivé virtuální stroje zajišťuje klasifikační program XDP napsaný uživatelem. Podrobnější informace o technologiích XDP a eBPF lze najít v [3], [4] a [11].

Ze strany ovladače je komunikace s aplikací zajištěna zaregistrováním implementace specifických funkcí vyžadovaných v XDP. Po přijetí paketu do své vnitřní datové fronty v paměti předá ovladač ukazatel na data paketu XDP programu. Program paket vyhodnotí a následně vrátí ovladači instrukci, jak má s paketem naložit. V případě odesílání zpět na TX by opět – v ideálním případě – nemělo docházet k žádnému kopírování dat paketu.

DPDK může v některých aplikacích umožňovat oproti XDP vyšší rychlost zpracování a větší flexibilitu. Je to proto, že v případě DPDK je propojení aplikace a ovladače o něco více přímé a zpracování může být prováděno zcela bez zásahu kernelu. XDP bude ale rychlejší, pokud budou využívány instrukce pro kernel zajišťující zahazování a přeposílání ze vstupu na výstup, které by v případě DPDK museli být řešeny aplikací. Navíc DPDK neumožňuje přímé propojení aplikací ve virtuálním stroji a ovladače síťové karty.

Jednou z podmínek zpracování RX paketů v XDP je to, že každý přijatý paket musí být uložen na jiné stránce v paměti. To je z důvodu snazší práce s pamětí v software.

Když chce aplikace odeslat nové pakety do sítě, předá pouze ovladači ukazatel na tento paket (a jeho délku) a ovladač podle toho vytvoří deskriptor pro DMA modul k odeslání dat. Stejně jako v DPDK může i tady vytvořit paket rozdělený na několik částí (zpravidla, když vytváří jednotlivé hlavičky paketu zvlášť). Na rozdíl od DPDK zde už ale není brán žádný

ohled na zarovnání jednotlivých částí. DMA modul se tak může dostat do situace, kdy bude muset odeslat paket jehož každý byte je uložen na jiném místě v paměti a žádný z nich neleží na adrese zarovnané na 4 B. Tento krajní případ je samozřejmě velmi nepravděpodobný a není vyžadováno při něm udržet maximální propustnost, ale DMA modul i systém jeho řízení z ovladače by měli tento případ podporovat alespoň z hlediska funkčnosti.

Dosažitelná propustnost na sběrnici je zde ve většině případů závislá na konkrétní podobě dat vytvářených uživatelem, ale DMA modul by měl být schopen dosahovat maximální propustnosti alespoň v ideálním případě. Tedy v případě, kdy bude muset číst pouze části od délce násobku MRRS, které jsou zarovnané v paměti na 4 B. I v ostatních případech by se měl návrh modulu snažit zajistit, aby byla propustnost co možná nejvyšší.

2.7 Rozhraní

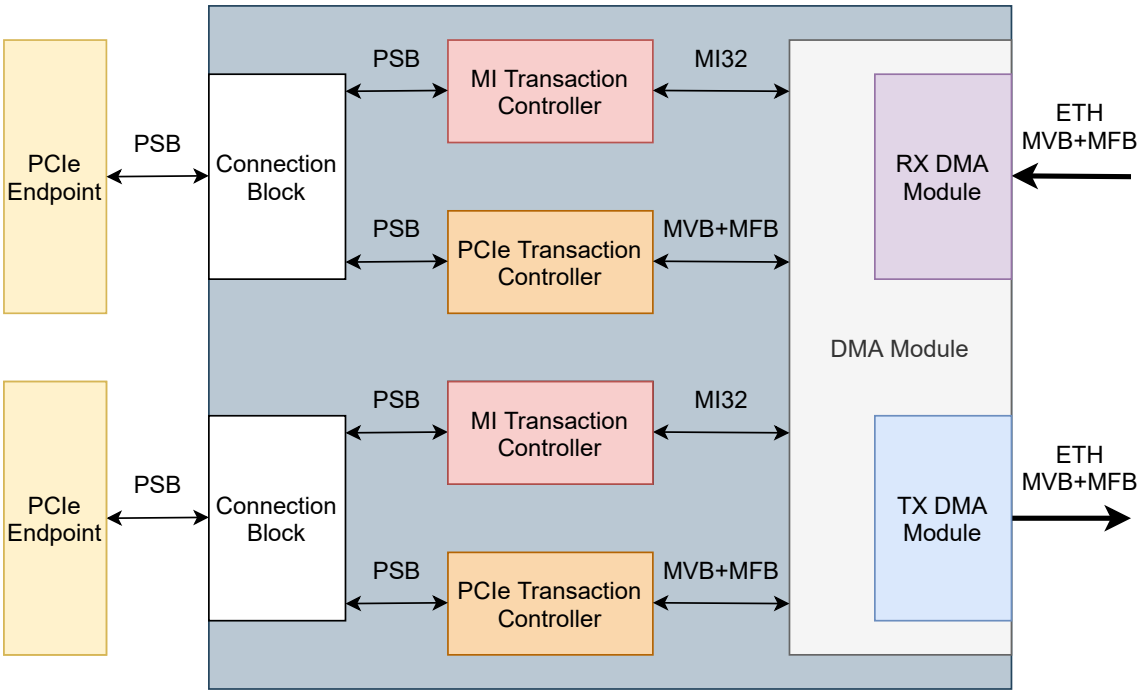
Jak bylo řečeno, přes PCIe Endpoint proudí do FPGA a z FPGA dva druhy požadavků. Ty, které posílá DMA modul do paměti, a ty, které posílá CPU do DMA modulu a jiných komponent v FPGA. Každý z těchto druhů se v FPGA zpracovává prostřednictvím jiné datové sběrnice. Transakce pro DMA přenosy používají sběrnici navrženou pro maximalizaci datové propustnosti a řídící transakce z CPU zase sběrnici navrženou pro jednoduché používání z pohledu hardware.

Protože DMA modul navržený v této práci má podporovat libovolnou platformu, využívá rozhraní používaná v rámci vývoje ve sdružení CESNET. Aby byla zajištěna kompatibilita s jednotlivými platformami, jsou součástí zapojení DMA modulu i jednotky převádějící tato rozhraní na rozhraní PCIe Endpointu.

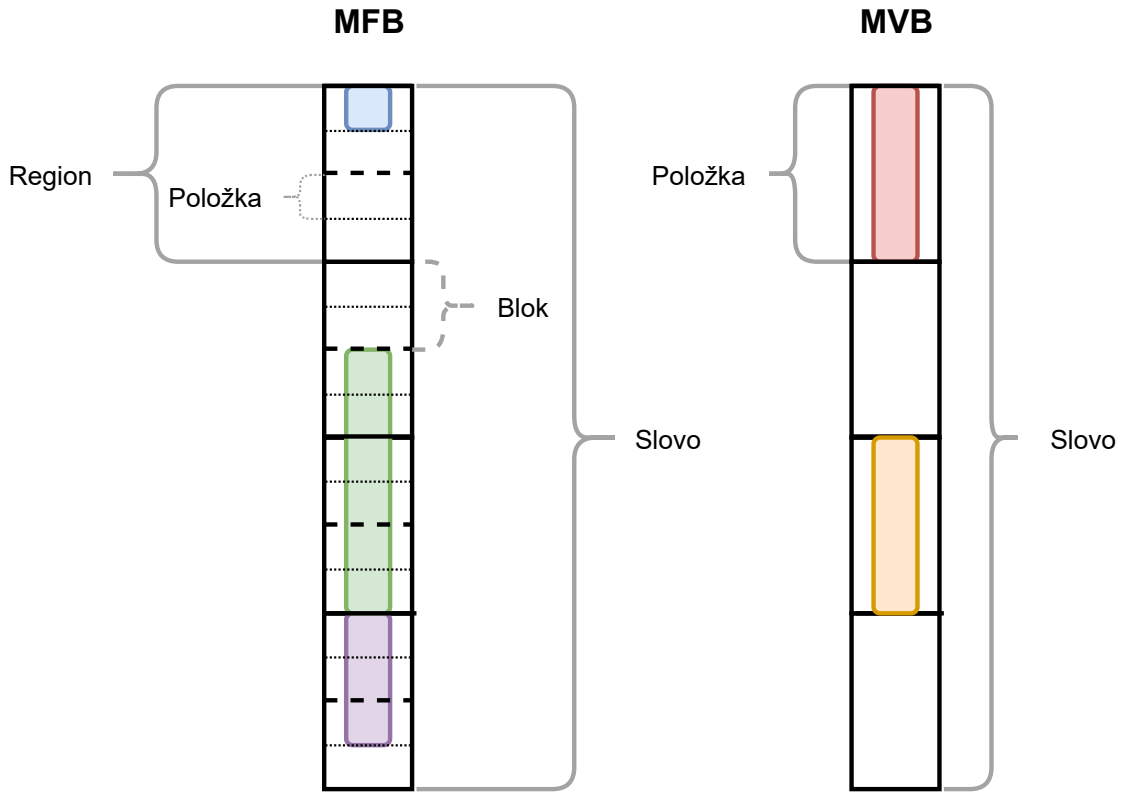
Pro vykonávání požadavků posílaných ovladačem síťové karty z CPU je používána sběrnice *MI32* (*Memory Interface*, adresová a datová šířka 32 b). Pro přenosy dat je to sběrnice *MVB+MFB* (*Multiple Value Bus* a *Multiple Frame Bus*). Sběrnice MVB+MFB je současně použita pro výstup dat paketů z TX DMA modulu směrem na ethernetové rozhraní (jde o odlišnou konfiguraci MVB+MFB než v případě PCIe přenosů). Příklad zapojení na FPGA čipu se dvěma PCIe Endpointy připojenými k jednomu DMA modulu je ukázán na obrázku 2.2.

Rozhraní MI32 slouží k obsluze požadavků přicházejících ze softwarového ovladače do FPGA přes PCIe Endpoint. Každý tento požadavek může být zápis do vnitřního registru některé komponenty v FPGA nebo jeho čtení. Cílovou komponentu a cílový registr přitom určuje adresa vystavená na adresné sběrnici. Adresový prostor je přitom zcela pod kontrolou adresových dekodérů v FPGA, případně autora cílové komponenty. Jak vyplývá z názvu sběrnice, pracuje se zde s daty o šířce 32 b. O řízení požadavků na sběrnici MI32 se stará centrálně komponenta MI Transaction Controller. Ta přijímá požadavky od PCIe Endpointu a vykonává je jeden po druhém. Vykonání každého požadavku může trvat více hodinových cyklů (v závislosti na délce cesty propagující signály k cílové komponentě a zpět a na chování komponenty samotné) a přenos proto není příliš rychlý. Protože se však jedná pouze o přenos konfiguračních informací v malém objemu (konfigurace registrů, čtení statistických čítačů a podobně), není nízká propustnost překážkou.

Sběrnice MVB+MFB se skládá ze sběrnic MVB a MFB, které jsou při samotném přenosu dat vzájemně nezávislé. Sběrnice MFB je koncept datové sběrnice pro přenos rámců s proměnnou délkou, u které lze zvolit parametry definující zarovnání dat, šířku sběrnice a maximální počet rámců přenášených v jednom hodinovém taktu. Parametry MFB pracují s pojmy „Region“, „Blok“ a „Položka“. Tyto pojmy jsou definovány v tabulce 2.1. Samotné parametry využívané při konfiguraci MFB jsou pak následující: *Bitová šířka Položky*, *Počet*



Obrázek 2.2: Schéma propojení DMA modulu a dvou PCIe Endpointů v FPGA. Každý z PCIe Endpointů je připojen prostřednictvím PSB (*Platform-Specific Bus*; Avalon, AXI nebo jiné) k jednotce Connection Block, která rozděluje tok dat na řídící požadavky z CPU a datové DMA přenosy. Řídící požadavky jsou prostředníctvím jednotky MTC (*MI Transaction Controller*) převedeny na sběrnici MI32. Datové požadavky jsou prostředníctvím jednotky PTC (*PCIe Transaction Controller*) převedeny na sběrnici MVB+MFB. Samotný DMA modul je rozdělen na část RX a část TX (která je předmětem této práce). Mimo tyto dvě části obsahuje DMA modul propojovací logiku, která umožňuje TX modulu i RX modulu posílat a přijímat požadavky z libovolného PCIe Endpointu. Směrem k Ethernetu má celý systém jedno vstupní a jedno výstupní rozhraní MVB+MFB nastavené v konfiguraci pro odpovídající Ethernetové rozhraní síťové karty.



Obrázek 2.3: Demonstrace konfigurovatelných částí na sběrnicích MFB a MVB. V případě MFB se jedná o konfiguraci $\langle x, 2, 2, 4\rangle$ (bitová šířka Položky zde není definována). V případě MVB jde o konfiguraci se čtyřmi Položkami ve slově a šířkou Položky shodnou s šířkou Regionu u zobrazeného MFB. Pro ilustraci obsahuje MFB slovo jeden z možných způsobů rozložení tří datových rámců a MVB slovo zase příklad rozložení dvou položek.

položek v jednom Bloku, *Počet Bloků v jednom Regionu* a *Počet Regionů v datovém slově*. Výsledná šířka sběrnice se pak spočte jako součin všech těchto hodnot. Tyto parametry mohou být pro zkrácení vyjadřovány jako seznam čtyř hodnot v ostrých závorkách v pořadí, ve kterém zde byly uvedeny (například $\langle 8, 4, 1, 2\rangle$ pro dva Regiony, každý s jedním Blokem, každý se čtyřmi Položkami o šířce 8 b). Sběrnice MVB naproti tomu pracuje pouze pojmem „Položka“ a jejími parametry jsou *Bitová šířka položky* a *Počet Položek v jednom slově*. Všechny tyto pojmy jsou ilustrovány na obrázku 2.3.

Pomocí sběrnice MFB lze vyjádřit nejrůznější druhy již definovaných sběrnic pro přenos datových rámců. Například sběrnice používaná pro přenos dat určených na Ethernet s rychlostí 100 Gb/s na datové šířce 512 b odpovídá nastavení MFB $\langle 8, 8, 8, 1\rangle$: Jednotlivé rámce se mohou lišit velikostí o jeden bajt, paket může v rámci slova začínat na 8 různých místech, v jednom slově může začínat anebo končit pouze jeden a celková šířka je 512 b. Tyto vlastnosti vyplývají ze specifikace pro Ehternet [5].

Signály obsažené na rozhraní se sběrnicí MFB jsou popsány v tabulce 2.2. Signál *MFB_DATA* obsahuje samotná data přenášená po sběrnici v rámci jednoho hodinového cyklu (jedno datové slovo). *MFB_SOF* (*Start of Frame*) a *MFB_EOF* (*End of Frame*) značí pro každý Region, zda je v něm obsažen začátek, respektive konec, rámce. Hodnoty *MFB_SOF_POS* (*Start of Frame Position*) a *MFB_EOF_POS* (*End of Frame Position*)

Název parametru	Popis
Region	Část šířky sběrnice, v rámci které se může nacházet jeden začátek rámce anebo jeden konec rámce. (Určuje maximální počet rámců v jednom hodinovém taktu.)
Blok	Část Regionu, na jejímž začátku se může objevit začátek rámce. (Určuje zarovnání začátků rámců.)
Položka	Část Bloku, která je nejmenší datovou jednotkou rámce. (Jednotlivé rámce se mohou lišit délkou pouze o násobek této jednotky.)

Tabulka 2.1: Popis pojmů, které se využívají při definici konkrétního nastavení u sběrnice MFB.

Název signálu	Směr	Šířka [b]
MFB_DATA	výstupní	Součin všech parametrů sběrnice
MFB_SOF	výstupní	1 pro každý Region
MFB_EOF	výstupní	1 pro každý Region
MFB_SOF_POS	výstupní	log_2(počet Bloků v Regionu) pro každý Region
MFB_EOF_POS	výstupní	log_2(počet Bloků v Regionu * počet Položek v Bloku) pro každý Region
MFB_SRC_RDY	výstupní	1
MFB_DST_RDY	vstupní	1

Tabulka 2.2: Výčet signálů přítomných na sběrnici MFB při použití MFB jako výstupu. Pokud je sběrnice vstupem, pak příznaky „vstupní“ a „výstupní“ budou u jednotlivých signálů prohozeny.

Název signálu	Směr	Šířka [b]
MVB_DATA	výstupní	Součin všech parametrů sběrnice
MVB_VLD	výstupní	1 pro každou položku
MVB_SRC_RDY	výstupní	1
MVB_DST_RDY	vstupní	1

Tabulka 2.3: Výčet signálů přítomných na sběrnici MVB při použití MVB jako výstupu. Pokud je sběrnice vstupem, pak příznaky „vstupní“ a „výstupní“ budou u jednotlivých signálů prohozeny.

pak pro odpovídající Region určují přesnou pozici tohoto začátku anebo konce. Pozice začátku rámce je udávána s přesností na Blok, zatímco pozice konce s přesností na Položku. Signály *MFB_SRC_RDY* (*Source Ready*) a *MFB_DST_RDY* (*Destination Ready*) tvoří společně handshake protokol.

Sběrnice MVB funguje podobným způsobem jako MFB. Rozdíl je v tom, že místo rámců o proměnné délce se po ní předávají pouze položky s konstantní velikostí. Můžeme si ji proto představit jako MFB sběrnici, kde každý rámec je přesně stejně velký jako Region a každý Region má právě jeden Blok. Z toho důvodu jsou signály *MFB_SOF*, *MFB_EOF*, *MFB_SOF_POS* a *MFB_EOF_POS* nahrazeny jediným signálem *MVB_VLD* označujícím platnost (nebo neplatnost) položky v daném „Regionu“. Kompletní výčet signálů na sběrnici MVB je uveden v tabulce 2.3. Sběrnice MVB má pouze dva parametry a to jsou *maximální počet položek předávaných v jenom hodinovém cyklu* a *bitová šířka jedné položky*.

Spojení sběrnic MVB a MFB do rozhraní MVB+MFB umožňuje po datové sběrnici MFB přenášet data a současně s nimi (nebo také dříve a nebo i později) přenášet k těmto datům i hlavičky obsahující dodatečná metadata. Tímto způsobem je MVB+MFB využita v DMA modulu při odesílání kompletních paketů ven z modulu. Sběrnice MFB v tomto případě musí obsahovat pouze data paketů, která mohou teoreticky být přivedena přímo do ethernetového výstupu FPGA. Pokud se ale cestou nacházejí ještě další komponenty, tak budou ke každému paketu potřebovat doplňující informace (například z jakého DMA kanálu paket pochází), ta jsou proto posílána po vedlejší sběrnici MVB.

Druhou výhodou tohoto spojení je, že po MVB můžeme odesílat i takové hlavičky, ke kterým žádná data nenáleží. Pokud na MFB přenášíme veliký datový rámec ve více hodinových taktech, může nám MVB zvýšit paralelizmus přenosu, neboť případné hlavičky bez rámců mohou být přenášeny současně. Takto je v DMA modulu sběrnice MVB+MFB využita na straně vedoucí k PCIe Endpointu, kde se po ní přenášejí požadavky na zápis (s daty) a požadavky na čtení (bez dat).

V rámci jedné dvojice MVB+MFB mají obě dílčí sběrnice své vlastní nezávislé handshake signály. Položky na MVB mohou proto při přenosu předbíhat rámce na MFB a naopak. Pokud je toto chování v nějakém případě nežádoucí, pak se musí sběrnice synchronizovat dodatečnou komponentou (zpravidla s využitím nezávislých FIFO front pro MVB a pro MFB). Jedinou přímou vazbou mezi daty na MVB a MFB je jejich pořadí (hlavičky na MVB, ke kterým náleží data, cestují ve stejném pořadí jako odpovídající rámce na MFB).

Kapitola 3

Současný stav

V oblasti vysokorychlostních síťových přenosů existuje v současnosti celá řada komerčních řešení. Tyto technologie jsou zajímavé především pro provozovatele datových center a páteřních síťových uzlů. V rámci uzlů a měřících bodů provozovaných sdružením CESNET se již několik let používají síťové karty vyvíjené na našem oddělení. Jedná se o síťové karty řady COMBO ([6]), z nichž nejnovější karta COMBO-200G2QL dosahuje propustnosti až 200 Gb/s. Této propustnosti je zde ale dosaženo přes dvě dvojice ethernetového rozhraní a PCIe rozhraní Gen3 x16, z nichž každá umožňuje dosáhnout propustnosti pouze 100 Gb/s. Nový DMA modul by měl podporovat i PCIe rozhraní Gen4 x16 s propustností až 200 Gb/s a ethernetové rozhraní s propustností až 400 Gb/s. Kromě těchto karet existuje v současnosti například již i síťová karta podporující přenosovou rychlost 200 Gb/s od firmy Mellanox s názvem ConnectX-6 ([7]).

Vysokorychlostní DMA přenosy řeší v rámci svého vývoje i výrobci FPGA čipů jako jsou Xilinx a Intel. Protože se však jedná o komerční řešení nejsou pro ně zpravidla dostupné příliš rozsáhlé informace popisující použité přístupy pro řízení přenosu dat ani omezení vyplývající z těchto přístupů.

Při navrhování nového DMA modulu jsem vycházel především z toho, čím byly nedostatečné předchozí verze DMA modulů používané v FPGA čipech na kartách vyvíjených v rámci našeho oddělení. DMA modul, který se v současnosti používá na těchto kartách byl vyvinut ještě pro první karty s podporou datové rychlosti 100 Gb/s a pro kartu COMBO-200G2QL byl použit tak, že ke každému PCIe Gen3 x16 rozhraní byla připojena jedna nezávislá instance. Hlavní nevýhodou tohoto modulu pro budoucí použití je podpora pouze malého počtu DMA kanálů (maximálně 64) vlivem nedostatků v návrhu, který nepředpokládal požadavek na více kanálů. Další nevýhodou je nemožnost rozšíření propustnosti DMA modulu nad 100 Gb/s.

Z toho důvodu začala před dvěma lety práce na nové verzi tohoto modulu, která měla tyto nedostatky odstranit. Nový DMA modul nazývaný dále v tomto textu „NDP DMA modul“ měl umožnit podporu až 1024 DMA kanálů a umožnit propojení několika PCIe rozhraní s propustností M*100 Gb/s s jedním ethernetovým rozhraním s propustností N*100 Gb/s.

Jak se později ukázalo, tak návrh tohoto modulu obsahoval rovněž jisté nedostatky. Tyto nedostatky vyplynuli částečně i z toho, že návrh probíhal v příliš dlouhém časovém rozmezí a zabýval se nejprve přenosy ve směru RX a teprve po dokončení práce ve směru RX začal návrh modulu ve směru TX. Tím vznikly značné návrhové rozdíly mezi těmito částmi, které zkomplikovaly a nakonec zastavily celý jeho vývoj. Navíc se obě tyto verze zabývaly DMA přenosy podle systému NDP, který je sice výhodný z pohledu propustnosti

přenosů na sběrnici PCIe, ale není výhodný z pohledu uživatelské aplikace, což omezuje jeho použitelnost.

Nový DMA modul, jehož TX část je předmětem této práce, má odstranit problémy týkající se současně používaného DMA modulu a má přitom primárně podporovat přenosy paketové a nikoliv proudové. Dále je žádoucí, aby modul umožňoval podporu co největšího počtu DMA kanálů. Podporovaný počet kanálů by se měl pohybovat na mocninách 2 od 128 výše, což je zapotřebí hlavně v aplikacích, kdy ke se každému DMA kanálu připojuje samostatný virtuální stroj. (Druhým případem využití vyššího počtu kanálů je situace, kdy je každý kanál zpracováván na jednom fyzickém jádře procesoru. Pro tento případ užití ale není zapotřebí tak vysoký počet kanálů, neboť ani počítače v současnosti většinou nemají tak vysoký počet jader.)

Návrh nového DMA modulu by měl vycházet z veliké části z vývoje NDP DMA modulu a opravit nedostatky, které byly u NDP DMA modulu zjištěny a které by mohly ohrozit i jeho vývoj.

Kapitola 4

Návrh

V této kapitole je v sekci 4.1 popsán návrh nového způsobu řízení paketových přenosů, který byl vyvinut pro tuto práci a který nese název NPP (*Netcope Packet Plane*). Dále je zde uveden stručný popis návrhu nového DMA modulu nazvaného „NPP DMA modul“ v sekci 4.3 a způsob jeho připojení ke sběrnici PCIe. Sekce 4.2 popisuje způsob, jakým je v návrhu řešena podpora co nejvyššího počtu DMA kanálů. Podrobnější popis jeho architektury je obsahem kapitoly 5.

4.1 NPP

Všechny tři přístupy k DMA přenosům síťových dat zmíněné v kapitole 2 (NDP, DPDK a XDP) mají svoje výhody a nevýhody. Proudový přenos paketů v NDP je výhodný z hlediska propustnosti na sběrnici a jednoduchosti implementace DMA modulu. Souběžný přístup k jedné datové frontě a nutnost zarovnaného a souvislého ukládání paketů v paměti ale znamená, že většina aplikací stejně nebude schopna plné propustnosti dosáhnout. Navíc zde neexistuje žádná širší podpora ze strany operačního systému. Paketový přenos přes DPDK již určitou podporu v software přináší, ale XDP je v tomto ohledu výhodnější, neboť je přímo součástí Linuxového jádra a navíc umožňuje připojení z virtuálního stroje.

Z toho důvodu bylo rozhodnuto, že nový DMA modul bude podporovat přenosy podle XDP. V rámci této práce jsem se proto podílel i na navržení nového protokolu NPP pro komunikaci mezi ovladačem a DMA modulem, který umožní tyto přenosy provádět. Tento protokol by měl teoreticky podporovat libovolné paketové přenosy (jako například DPDK) pouze s potřebou dodělání podpory v ovladači. Protokol obsahuje i některé prvky umožňující přidání podpory pro proudové přenosy NDP, které už ale budou vyžadovat zásah do některých částí DMA modulu. Protože tyto prvky jsou nad rámec zadání této práce nebudu je v této práci popisovat. Systém NPP řeší řízení přenosů pro RX i pro TX, ale zde budu vysvětlovat pouze směr TX a to především tu jeho část, která je důležitá z pohledu fungování DMA modulu. Nejdůležitější kroky při řízení přenosů jsou na konci sekce demonstrovány na příkladě na obrázcích 4.2, 4.3 a 4.4.

Jak již bylo zmíněno výše, DMA modul získává informaci o adrese a délce dat, která má číst z paměti, od ovladače ve formě deskriptorů. Při odesílání paketů o velikosti 512 B rychlostí 100 Gb/s je třeba, aby DMA modul získával deskriptory rychlostí skoro 25 milionů za sekundu. U paketů velikosti 64 B je to už přes 195 milionů za sekundu. Ovladač proto tyto deskriptory nezapisuje přímo do registrů v FPGA jeden po druhém. Místo toho je v paměti vytvořen souvislý kruhový buffer, ve kterém jsou uloženy deskriptory jeden

za druhým (každý deskriptor má velikost 8 B). Po uložení deskriptorů do bufferu v paměti, ovladač předá DMA modulu informaci o adrese a velikosti tohoto bufferu. Následně DMA modul spustí a DMA modul si začne číst deskriptory z bufferu v paměti sám, pokud možno po blocích o velikosti MRRS. DMA modul a ovladač se musejí průběžně informovat o tom, které deskriptory v bufferu jsou již platné a které jsou již použité (pakety jimi popisované jsou již odeslané). Proto obsahuje DMA modul pro každý DMA kanál i registry pro zápisový a čtecí ukazatel do tohoto bufferu. Na začátku mají oba ukazatele hodnotu 0 (v bufferu není žádný platný deskriptor). Když ovladač vloží do bufferu nové deskriptory, zvýší hodnotu zápisového ukazatele o počet nových deskriptorů. Jakmile DMA modul přečte tyto deskriptory a následně přečte i data s pakety, které tyto deskriptory popisují, zvýší zase odpovídajícím způsobem hodnotu čtecího ukazatele. Ovladač musí periodicky kontrolovat hodnotu čtecího ukazatele, aby se dozvěděl, které deskriptory může již přepsat a která data v paměti uvolnit. Tyto ukazatele jsou dále v této práci označovány jako „HDP“ (*Hardware Descriptor Pointer*, čtecí ukazatel nastavovaný DMA modulem) a „SDP“ (*Software Descriptor Pointer*, zápisový ukazatel nastavovaný ovladačem).

Pro spouštění a zastavování jednotlivých DMA kanálů obsahuje DMA modul registry „Control“ a „Status“. Do registru Control zapíše ovladač hodnotu 1 když chce tento kanál spustit (adresa a velikost deskriptorového bufferu musí v tu chvíli být již správně nastavena a nesmí se měnit po celou dobu běhu kanálu). Zapsáním hodnoty 0 do tohoto registru vydá příkaz k jeho zastavení. Registr Status nastavuje DMA modul na 0 u zastaveného kanálu a 1 u spuštěného. Protože při zastavení musí DMA modul nejdříve dokončit odesílání všech dat popisovaných právě platnými deskriptory, může zastavování trvat až několik stovek taktů hodinového signálu modulu.

Formát deskriptorů pro NPP

Velikost deskriptoru samotného je kritická z pohledu propustnosti, neboť s jeho velikostí vzrůstá režie komunikace po PCIe pro přenos každého paketu. U NPP byla velikost jednoho deskriptoru stanovena na 8 B. Pokud tedy budeme přenášet pakety o velikosti 512 B, každý popsaný jedním deskriptorem, musíme přes sběrnici PCIe přenést celkem $512 + 8 = 520\,B$. Tím se propustnost vlivem přenosu deskriptorů snižuje na $\frac{512}{520} = 98,5\%$. Rychlosti 100 Gb/s můžeme ale dosáhnout i tak, neboť sběrnice PCIe umožňuje ve skutečnosti celkovou propustnost o trochu vyšší než 100 Gb/s.

8 bajtů ale zabere jen samotná 64 bitová adresa do paměti a v deskriptoru by tak už nezbývalo místo na délku popisovaných dat. Jednou z možností by bylo číst pro každý paket deskriptory 2: jeden s adresou a jeden s délkou. To ale znamená, že při přenosu paketů o velikosti 64 B bude vždy nutno přečíst paměti $64 + 16 = 80\,B$ dat, což snižuje efektivitu přenosu malých paketů na $\frac{64}{80} = 80\ \%$.

Snížení průměrného počtu přenesených deskriptorů na paket lze dosáhnout zavedením stavového chování při jejich dekódování. Operační systém běžně nevyužívá všechny bity fyzické adresy v paměti. Nejvyšší bity zůstávají většinou stejné a mění se pouze vzácně. Pokud pošleme DMA modulu deskriptor obsahující hodnotu nejvyšších několika bitů adresy určité části paketu, pak můžeme předpokládat, že i následující data budou mít pravděpodobně u těchto bitů adresy stejnou hodnotu. Tuto informaci tedy pošleme pouze pro první z nich a následně budeme posílat deskriptory, kde každý bude obsahovat pouze nižší bity adresy, délku a případně další potřebné informace. Deskriptor s nejvyššími bity adresy odešleme znovu až když budou u některé části paketu odlišné od té předcházející.

Typ 00	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
bity 63:48	0	0	X													
bity 47:32	X														A 63:62	
bity 31:16	A 61:46															
bity 15:0	A 45:30															

Typ 10	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
bity 63:48	1	0	P	X	H											
bity 47:32	D															
bity 31:16	X		A 29:16													
bity 15:0	A 15:0															

Obrázek 4.1: Schéma formátu TX NPP deskriptorů. Formát je zobrazen pro deskriptory typu 00 (nahoře) a typu 10 (dole). Oba typy mají šířku 64 bitů, jejichž obsah je zobrazen na 4 řádcích po 16 bitech od nejvyššího bitu 63 (vlevo nahoře) až po bit 0 (vpravo dole). Významy jednotlivých políček jsou: *0* — Bit má vždy hodnotu 0. *1* — Bit má vždy hodnotu 1. *X* — Bit není využit a má mít vždy hodnotu 0. *A <rozsah>* — Bity náležící do dané části v adrese do paměti. *P* — Příznak pokračování (má hodnotu 1, pokud popisovaný paket pokračuje do dalšího deskriptoru; jinak má hodnotu 0). *H* — Bity obsahují uživatelem definovanou hlavičku (metadata). *D* — Bity vyjadřující délku prostoru v paměti v bajtech.

Obrázek 4.1 obsahuje podrobný popis významu všech bitů v deskriptorech používaných v TX NPP. Deskriptory jsou dvou různých typů: 00 a 10 (značeno binárně). Důvod, proč je typ deskriptoru vyjádřen na 2 bitech je ten, že existuje ještě typ 11, který je určen pro optimalizaci v případě podpory proudových přenosů NDP. Ten není součástí návrhu v této práci. Typ s označením 01 není využit a je rezervován.

Adresa dat je zde rozdělena na dolních 30 bitů, které jsou dodávány pro každý blok dat, a horních 34 bitů, které jsou předloženy pouze, pokud se změnili oproti předchozí adrese (a také úplně na začátku po spuštění DMA kanálu). Důvodem, proč není adresa rozdělena dvě části po 32 bitech, je dosažení podobnosti s formátem NPP deskriptorů pro směr RX. V RX směru je totiž zapotřebí umístit do 64 bitového deskriptoru spodní část dvou různých adres (2krát 30 bitů) a navíc 2 bity pro vyjádření typu deskriptoru a 2 bity kontrolních příznaků.

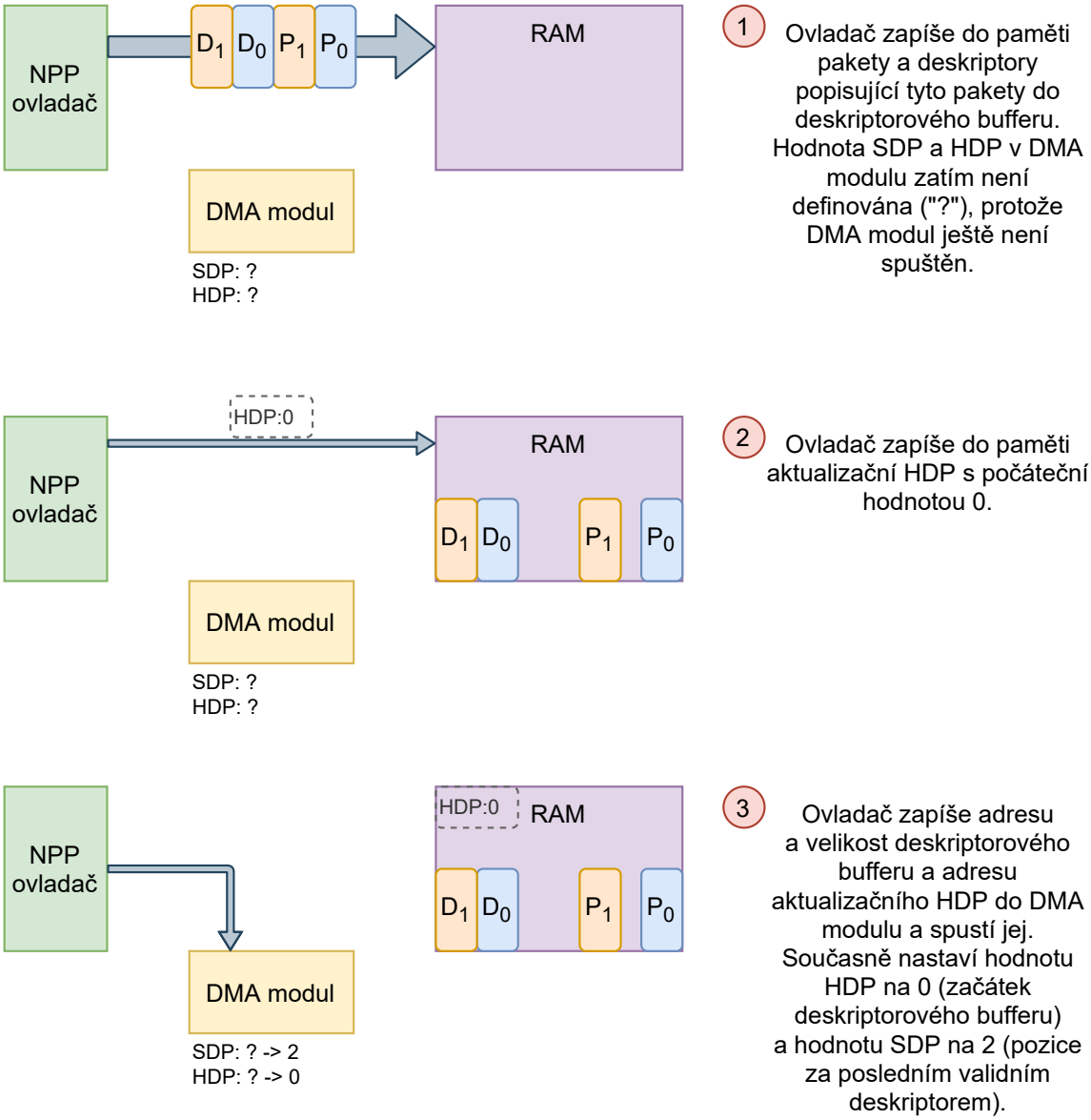
Kromě již zmíněné adresy a délky dat popisovaných deskriptorem, obsahuje formát také políčko s příznakem pokračování (na obrázku 4.1 označeno „P“). Hodnotou tohoto příznaku informuje ovladač DMA modul, zda na konci dat popisovaných tímto deskriptorem se nachází také konec paketu (hodnota příznaku 0), nebo jestli paket pokračuje v následujícím bloku dat (hodnota příznaku 1). Další dosud nezmíněné pole v deskriptoru je pole hlavičky (na obrázku 4.1 označeno „H“). Toto pole obsahuje 12 bitů, které si definuje tvůrce aplikace zpracovávající pakety v FPGA až poté, co odejdou z DMA modulu, a musejí být propagovány DMA modulem na výstup společně se samotným paketem. V případě rozdělení paketu do více deskriptorů je tato hlavička platná ve všech těchto deskriptorech.

Aktualizace deskriptorového ukazatele

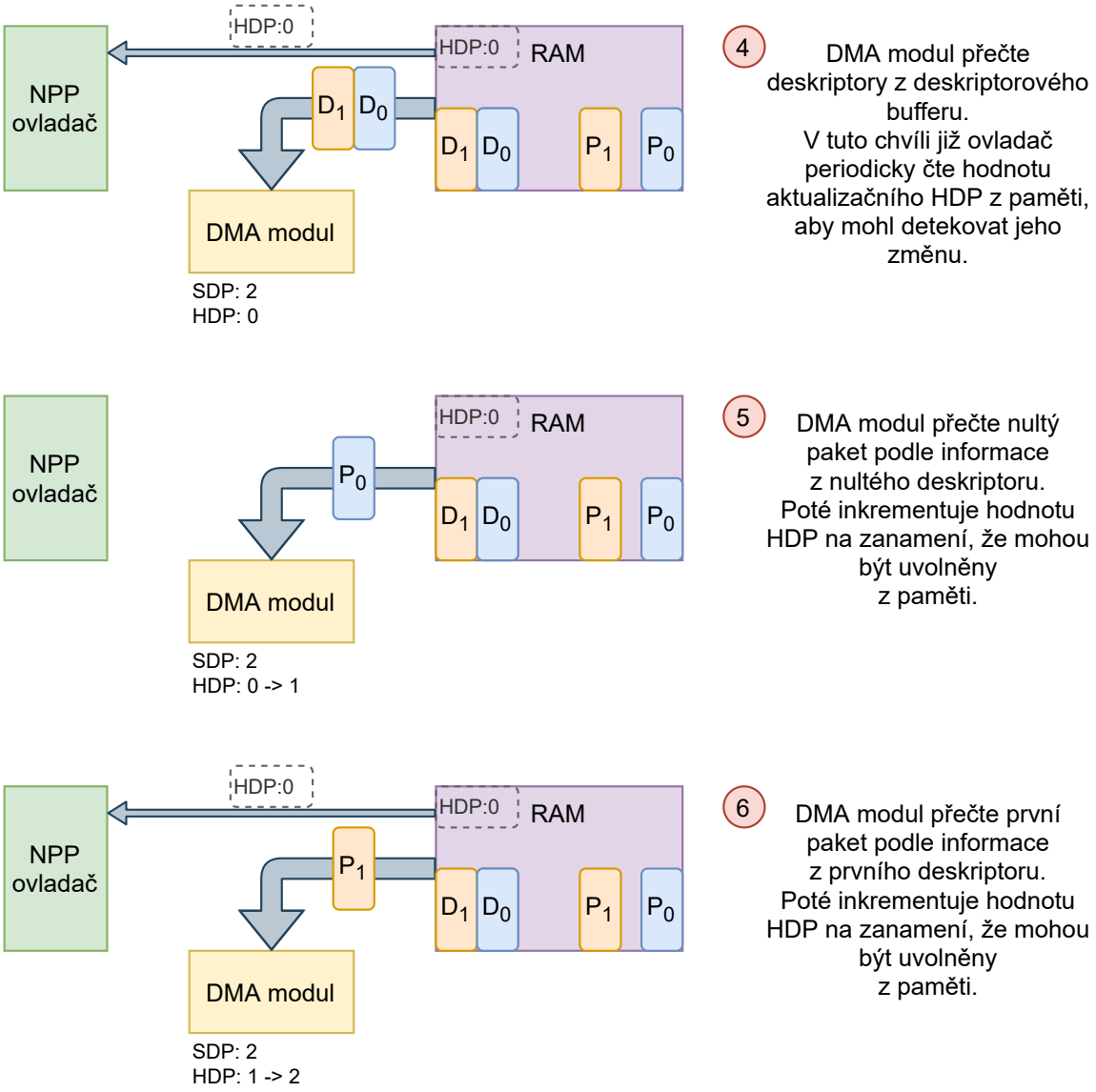
Jak je popsáno výše, musí NPP ovladač pracující nad jedním DMA kanálem periodicky kontrolovat aktuální hodnotu HDP, který je nastavován DMA modulem, aby se dozvěděl o uvolnění místa v deskriptorovém bufferu. Tento ukazatel je obsažen v registru DMA modulu a jeho přečtení proto pro ovladač znamená zaslání čtecího požadavku na PCIe, zpracování tohoto požadavku v FPGA a následné odeslání odpovědi zpět do CPU. Latence této operace může být pro vysokorychlostní přenosy příliš veliká. Při provádění této operace paralelně na několika stovkách DMA kanálů jde také o znatelné zatěžování sběrnice PCIe. Z toho důvodu DMA modul nejenom periodicky zapisuje aktualizovaný ukazatel do svého vnitřního registru, ale současně jej odesílá i jako zápisový požadavek do paměti. Cílovou adresu v paměti mu určí ovladač při spouštění kanálu. Ovladač si pak nemusí číst ukazatel z FPGA na síťové kartě, ale čte jej pouze z interní paměti počítače, což je výrazně rychlejší.

Režie odesílání zápisového požadavku z DMA modulu je nízká, protože požadavek se zasílá, pouze pokud se hodnota ukazatele skutečně změní a to navíc ne vždy, ale jen jednou za čas. Do budoucna se uvažuje zavedení podpory pro asynchronní přerušení odesílané z DMA modulu přes PCIe při změně hodnoty zápisového ukazatele. Díky tomu nebude muset CPU při čekání na změnu ukazatele vykonávat žádnou práci a bude automaticky „probuzen“ pouze ve chvíli, kdy se skutečně uvolní nové místo v jeho deskriptorovém bufferu.

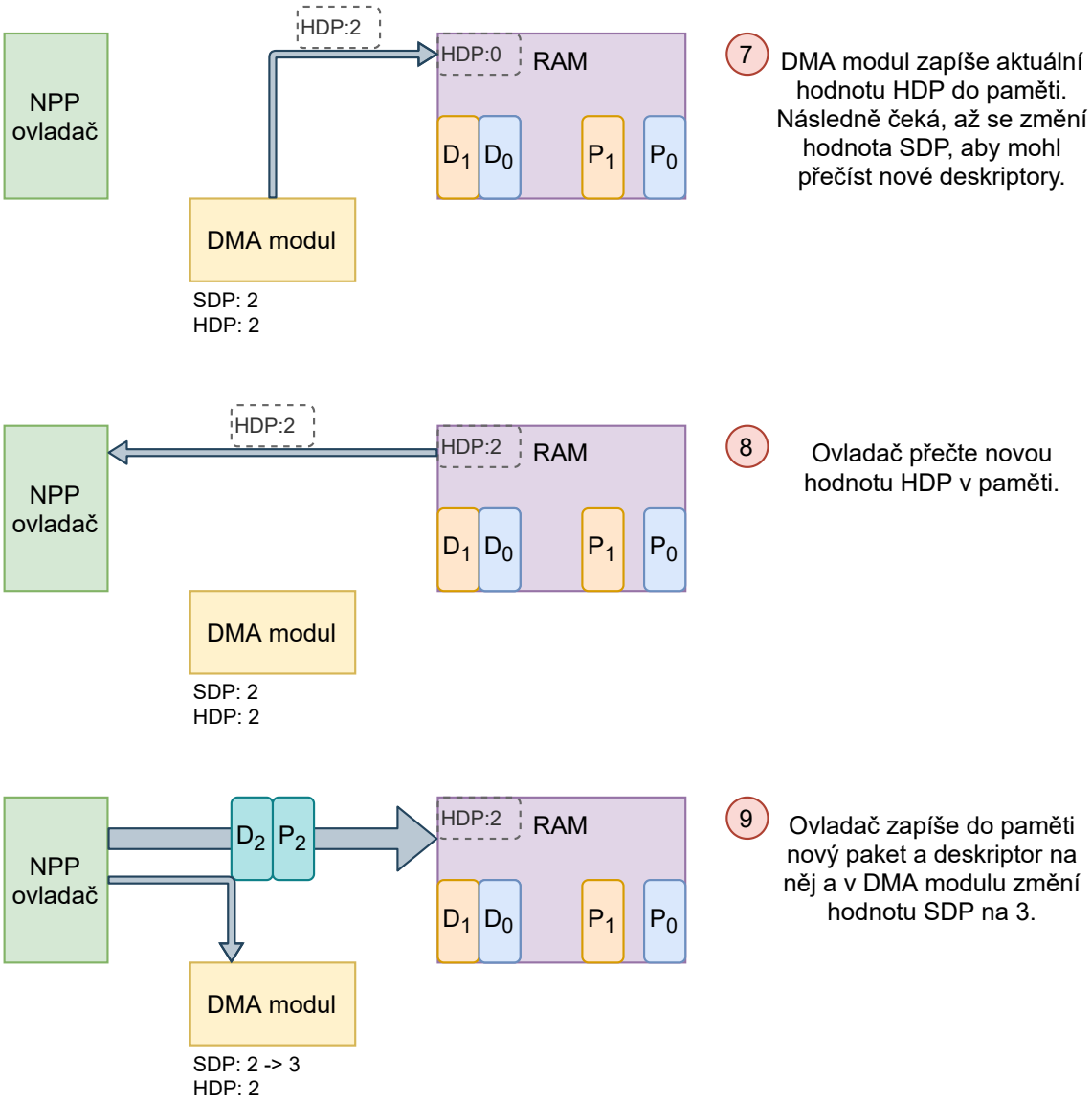
Pro TX DMA modul znamená zasílání aktualizací především to, že nebude generovat pouze čtecí požadavky pro čtení deskriptorů a dat, ale také zápisové požadavky pro zápis ukazatele.



Obrázek 4.2: Demonstrace řízení DMA přenosů podle systému NPP – kroky 1 až 3, inicializace přenosů ovladačem.



Obrázek 4.3: Demonstrace řízení DMA přenosů podle systému NPP – kroky 4 až 6, čtení paketů DMA modulem.



Obrázek 4.4: Demonstrace řízení DMA přenosů podle systému NPP – kroky 7 až 9, synchronizace DMA modulu s ovladačem a příprava na odeslání dalšího paketu.

4.2 Řešení podpory DMA kanálů

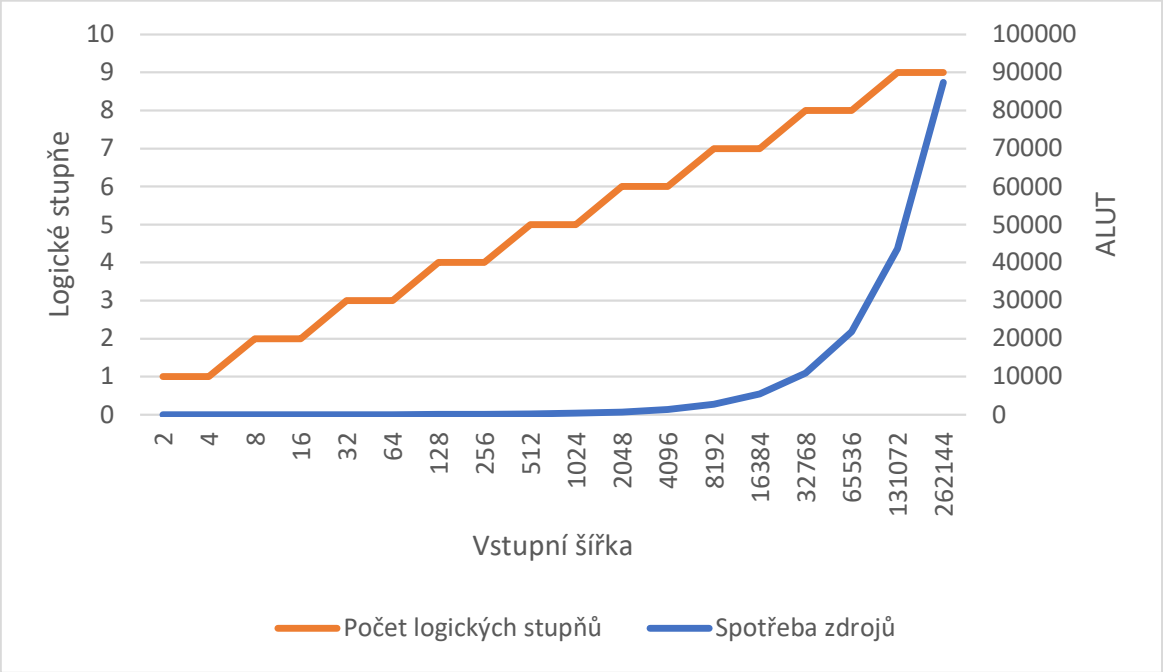
Od DMA modulu se očekává, že bude umožňovat podporu současného provozu vysokého počtu DMA kanálů. Celková propustnost DMA modulu, by přitom v ideálním případě neměla být ovlivněna počtem kanálů, které odesílají data. Počet podporovaných DMA kanálů je parametr modulu, který se nastavuje před jeho syntézou. Při samotném používání si uživatel může vybrat podmnožinu těchto kanálů jejich samostatným spouštěním a zastavováním.

Vyšší počet nastavených DMA kanálů, ale povede nutně ke zvyšování zdrojů spotřebovaných na FPGA a také k prodlužování některých logických cest. Řekněme, například, že bychom chtěli (pro zjednodušení implementace) ukládat v DMA modulu data přečtená z paměti v odděleném bufferu pro každý kanál. To by znamenalo lineární nárůst celkové velikosti těchto bufferů s rostoucím počtem kanálů. Vzhledem k tomu, že každý z nich by měl být schopen pojmout alespoň 2 pakety o maximální velikosti (pro zajištění plné propustnosti při odesílání velikých paketů), byla by spotřeba zdrojů enormní již pro 64 DMA kanálů. Tak tomu bylo u jedné ze starších verzí DMA modulu.

I pokud budeme používat buffer společný pro všechny DMA kanály, budou se pořád v modulu vyskytovat místa, jejichž zdrojový nárůst bude lineární k počtu DMA kanálů. Jedním z těchto míst budou například registry, ve kterých se uchovávají konfigurační údaje pro jednotlivé kanály zapisované ze software. Ty musejí být nutně přítomny v jedné instanci pro každý DMA kanál zvlášť a budou vždy zapotřebí všechny, aby mohl být podporován souběžný chod všech DMA kanálů. Protože ale tyto registry nebudou samy zabírat příliš zdrojů ani pro relativně vysoký počet kanálů, nemusíme se je snažit nijak redukovat. V každém případě ale musí být při návrhu zajištěno, aby jednotlivé DMA kanály mohly zdroje spotřebované na FPGA co nejlépe sdílet.

Stejně jako zdroje bude nepochybně s počtem kanálů vzrůstat i komplexnost některých logických operací prováděných v DMA modulu. Například obyčejné vybrání jedné položky z jednoho z DMA kanálů znamená vytvoření multiplexoru, jehož šířka je závislá na počtu DMA kanálů. Logická cesta u multiplexoru se přitom prodlužuje logaritmicky s jeho šířkou (pokud nebereme v úvahu navíc i omezenou hustotu spojů na FPGA). Spotřeba zdrojů na FPGA stoupá se zvyšující-se šířkou multiplexoru lineárně. Zároveň si ale musíme uvědomit, že počet kanálů nebudeme chtít zvyšovat lineárně, ale exponenciálně, neboť nás zajímají pouze počty kanálů, které jsou mocninami dvou. U určitého počtu DMA kanálů se tak vždy dostaneme do bodu, kdy ani obyčejný multiplexor nebude splňovat podmínky dané periodou hodinového signálu. Tento efekt je graficky znázorněn na výsledcích měření syntézy multiplexoru na FPGA Stratix10 na obrázku 4.5 V DMA modulu budeme muset provádět často i mnohem složitější operace než multiplexování, které budou mít složitost závislou na počtu kanálů.

Částečným řešením na problém nárůstu zdrojů a délky logických cest je využití takzvaných „Vybraných DMA kanálů“. Toto řešení bylo již úspěšně implementováno v předchozí verzi NDP DMA modulu. Vybrané DMA kanály jsou taková podmnožina DMA kanálů, pro kterou platí, že kanály v ní obsažené jsou v jeden okamžik současně aktivní (na těchto kanálech se právě provádějí DMA přenosy). Maximální počet Vybraných DMA kanálů nastavených v modulu může být přitom výrazně nižší než celkový počet kanálů a od určité hranice už jeho zvyšování ani nepřináší žádné výhody. Protože uživatel může vyžadovat souběžný provoz většího počtu kanálů, než kolik je Vybraných kanálů, musí DMA modul dynamicky přepínat mezi spuštěnými kanály a vybírat, které z nich budou v danou chvíli aktivní a které ne. Složitost komponent pracujících s daty DMA kanálů bude pak závislá pouze na počtu Vybraných kanálů, protože v daném okamžiku pracuje DMA modul vždy



Obrázek 4.5: Graf naměřených výsledků spotřeby zdrojů a délku logické cesty u multiplexoru v závislosti na jeho šířce na FPGA čipu Stratix10 od firmy Intel. Na vodorovné ose je šířka vstupu multiplexoru. Na svislé ose vlevo je délka logické cesty ze vstupu na výstup vyjádřená v počtu stupňů přes logické členy. Na svislé ose vpravo je množství spotřebovaných zdrojů na multiplexor vyjádřené v počtu logických členů ALUT. Vodorovná osa je vyjádřena v logaritmickém měřítku, což transformuje logaritmický nárůst počtu logických stupňů na lineární a lineární nárůst spotřeby zdrojů na exponenciální. Dále je třeba si uvědomit, že se vzrůstajícím počtem potřebných zdrojů stoupá i náročnost umístění logických cest na FPGA, což od určitého bodu vede k dramatickému prodlužování logických cest bez ohledu na počet logických stupňů.

pouze s daty těchto několika aktivních DMA kanálů. Tento systém dokáže zajistit maximální propustnost na všech DMA kanálech za předpokladu, že splňuje tyto čtyři podmínky:

1. Počet Vybraných DMA kanálů je tak vysoký, že DMA modul dokáže posíláním čtecích požadavků na tento počet kanálů plně vytížit propustnost připojené sběrnice PCIe.

2. Všechny spuštěné kanály se pravidelně střídají v roli aktivních kanálů a nedochází k blokování žádného z nich ani k jejich vyhladovění.

3. Režie nutná k deaktivaci jednoho kanálu a aktivaci nového kanálu není na tolik velká, aby během ní došlo k poklesu vytížení sběrnice PCIe. (Tuto podmínku lze nahradit tvrzením, že počet Vybraných DMA kanálů *mínus jedna* musí být dostatečně vysoký na plné vytížení sběrnice PCIe, což je jen mírné zpřísnění podmínky 1.)

4. Pokud k tomu není důvod, nemělo by z Vybraných DMA kanálů být neaktivní nikdy více než jeden, a to ten, který je právě přepínán. (Důvodem k deaktivaci více Vybraných DMA kanálů může být pouze to, že momentálně není spuštěno tolik DMA kanálů, aby zaplnili všechny pozice mezi Vybranými kanály.)

To, zda DMA modul dokáže plně vytížit propustnost sběrnice PCIe a tím dosáhnout maximální propustnosti, je ovlivněno mnoha faktory. Předpokládejme, že jednotka odesílající z DMA modulu požadavky na čtení dat je připojena k PCIe Endpointu s propustností 100 Gb/s sběrnicí o šířce 512 b (64 B) taktovanou na frekvenci 200 MHz. Pokud bude mít jednotka možnost číst data vždy po blocích o velikosti MRRS (512 B), pak po této sběrnici přijde do DMA modulu maximálně jeden požadavek každých 8 cyklů ($\frac{512}{64} = 8$). Pro plné vytížení bude proto stačit odeslat v průměru pouze jeden požadavek za 8 taktů. Dokonce i v případě, že budou odesílány pakety o velikosti pouhých 64 B – a za předpokladu, že nebudou rozděleny do více deskriptorů –, může DMA modul zajistit plné vytížení této sběrnice odesíláním jednoho požadavku za takt. Z toho teoreticky vyplývá, že DMA modulu stačí k plné propustnosti provozovat v každém okamžiku pouze jeden jediný kanál a počet Vybraných DMA kanálů by tak měl být 2. Existují ale ještě dva další důležité faktory, které vedou k tomu, že Vybraných kanálů by mělo být o něco více.

1. K tomu, aby DMA modul mohl neustále odesílat čtecí požadavky na jeden DMA kanál, musí mít neustále k dispozici deskriptory přečtené z paměti. Latence mezi uvolněním použitého deskriptoru z paměti, jeho nahrazení novým a přečtením nového deskriptoru DMA modulem, je ale dosti vysoká. Aby mohl DMA modul mít stále dostatek deskriptorů na odesílání čtecích požadavků na data, musel by je ovladač v software dostatečně rychle vytvářet a deskriptorový buffer by musel být hodně veliký, aby překryl zmíněnou latenci. To se od něj ale nedá vždy očekávat. Provoz se dvěma Vybranými kanály by proto pravděpodobně vypadal tak, že DMA modul by použil všechny deskriptory na jednom kanálu, následně se začal přepínat na nový a mezi tím by použil všechny deskriptory i na druhém Vybraném kanálu. Pokud by přepínání nebylo dost rychlé, mohl by v jednu chvíli vyčerpat deskriptorový buffer na obou těchto kanálech současně a neodesílal by žádné požadavky na data až do aktivace nového kanálu.

2. Pokud se DMA modul při odesílání čtecích požadavků bude periodicky přepínat mezi dvěma Vybranými kanály, potom musí být zajištěno, aby po uplynutí dvou taktů byly aktualizovány všechny vnitřní informace potřebné pro odeslání nového požadavku

na stejný kanál. To znamená, že zřetězená linka vyhodnocující toto odesílání by mohla mít maximálně 2 stupně. Kvůli komplexnosti operací, které se zde musejí vykonávat (především výpočet délky požadavku), je ale žádoucí, aby tato linka mohla být co nejdelší.

Vzhledem k těmto dvěma faktorům jsem se rozhodl, že cílený *minimální* počet Vybraných DMA kanálů bude 8. Vyšší hodnota by teoreticky mohla vést na lepší propustnost v některých okrajových situacích, ale pro účely snížení spotřeby zdrojů a zkrácení logických cest se bude počet Vybraných kanálů zvyšovat pouze v případě, že se objeví takový požadavek při pozdějším testování. Počet Vybraných kanálů bude nastavován jediným parametrem před syntézou DMA modulu a jeho změna proto bude z implementačního hlediska triviální (pokud to přímo nezpůsobí, že DMA modul přestane splňovat podmínky časování).

Řízení Vybraných DMA kanálů je v DMA modulu prováděno tak, že u aktivního Vybraného kanálu je zahájena deaktivace, pokud platí některá z následujících podmínek:

1. Kanál momentálně nemá žádná data k odeslání (jeho HDP a SDP se vzájemně rovnají).
2. Pro tento kanál přišel ze software příkaz na jeho zastavení. (V tomto případě se před deaktivací musí navíc dokončit odeslání všech paketů tohoto kanálu čekajících v paměti.)
3. Pro Vybraný kanál vypršel čítač maximální doby jeho aktivace. (Tento čítač se nastavuje u každého kanálu při aktivaci na předem danou konstantu a je u všech kanálů pravidelně dekrementován dokud nedosáhne hodnoty 0. Tím je zabráněno vyhladovění neaktivních DMA kanálů.)

Při samotné deaktivaci kanálu musí být nejprve dokončeno čtení celého paketu. Následně je zastaveno čtení dat i deskriptorů z paměti a je odeslána aktualizace HDP. Tento proces může trvat i stovky hodinových cyklů, což ale nevadí, neboť DMA modul dokáže po tuto dobu plně vytěžovat sběrnici požadavky na ostatních 7 Vybraných kanálech. Problém by nastal jenom v případě, že by během této doby vyčerpaly všechny ostatní kanály veškerá data k odeslání a museli čekat na posunutí SDP ze strany software. To, že by k tomu došlo u všech 7 kanálů je ale nepravděpodobné a je to známkou toho, že software odesílá data na těchto kanálech velmi pomalu.

Je třeba zdůraznit, že koncept Vybraných DMA kanálů lze využít pouze ve směru TX. Při odesílání dat z paměti si totiž může DMA modul sám vybrat, z kterých kanálů do něj budou v daný okamžik přicházet data. Ve směru RX, kdy na vstup DMA modulu přicházejí data ze sítě přiřazená do jednotlivých kanálů komponentou před ním, musí být DMA modul připraven kdykoli přijmout data pro kterýkoli kanál. Jedinou možností použití Vybraných DMA kanálů by zde bylo ukládání velikého počtu paketů na všech kanálech a jejich opožděné odesílání do paměti počítače na Vybraných kanálech. To by vedlo nejenom ke zvýšení latence, ale kladlo by vysoké paměťové nároky na FPGA čip. V případě TX je problém paměťových nároků přenesen na buffer ovladače v paměti.

4.3 NPP DMA modul

TX NPP DMA modul je navržen tak, aby podporoval přípojení většího počtu PCIe Endpointů. Data čtená z jednotlivých PCIe rozhraní jsou uvnitř modulu slučována do jediného

proudu paketů na výstupu. Aby mohl modul dosáhnout plné propustnosti na všech připojených PCIe Endpointech, je datová šířka výstupu rovna sumě šířek všech PCIe rozhraní. Protože DMA modul bude pracovat na frekvenci 200 MHz je pro propustnost N*100 Gb/s tato šířka N*512 b. Současně to znamená, že při odesílání paketů o velikosti 64 B (512 b) musí DMA modul zpracovat v některých částech N paketů v každém cyklu hodinového signálu.

Aby byl tento náročný úkol splnitelný v rámci FPGA platformy, byl návrh vytvořen tak, aby co největší část DMA modulu pracovala pouze s jedním PCIe Endpointem a pouze minimum komponent pracovalo s daty ze všech PCIe rozhraní současně. Části pracující jen s jedním PCIe Endpointem jsou následně zabaleny do komponenty nazvané „DMA Endpoint“ a v rámci modulu se vyskytují v několika kopiích podle potřeby. Části mimo DMA Endpoint slouží především ke sloučení dat ze všech DMA Endpointů do jednoho výstupu.

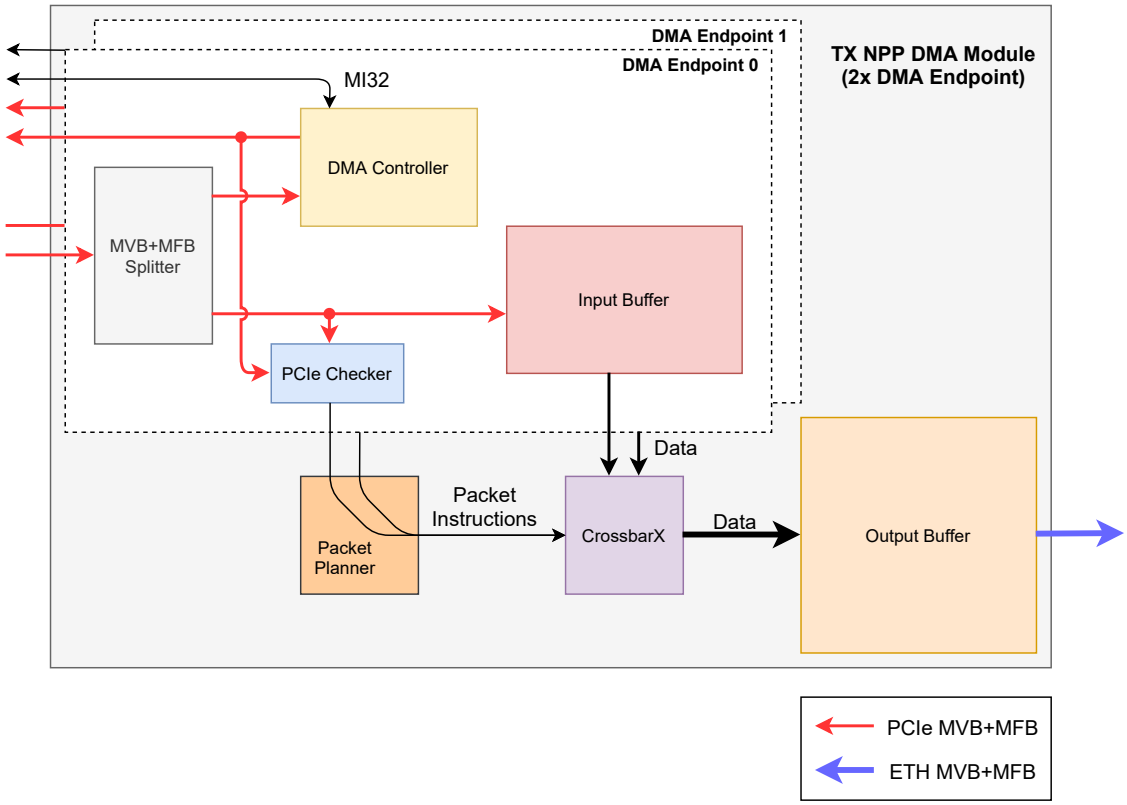
Hrubé schéma TX NPP DMA modulu (konkrétně verze se dvěma DMA Endpointy) je zobrazeno na obrázku 4.6. Podrobnější schéma lze nalézt v kapitole 5. Jediné komponenty, které se v DMA modulu nacházejí mimo obálku DMA Endpoint jsou Packet Planner, CrossbarX a Output Buffer. Všechny tyto tři komponenty se skutečně podílejí na slučování dat z Endpointů do jednoho výstupu. Není to však jejich jediná funkce. Například komponenta Packet Planner zajišťuje určování cílové adresy paketů v Output Bufferu. Současně však tyto adresy přiděluje tak, aby se mezi výstupními pakety nacházely správné mezipaketové mezery, jak je vyžadováno u datového provozu pro Ethernet [5].

DMA Endpoint obsahuje komponentu DMA Controller. Tato jednotka se stará o řízení samotných DMA přenosů. Spravuje komunikaci jednotlivých DMA kanálů s ovladačem přes sběrnici MI32, odesílá požadavky na čtení deskriptorů a přijímá a zpracovává tyto deskriptory v souladu s protokolem NPP. Dále odesílá (podle informací získaných z deskriptorů) požadavky na čtení dat paketů. Z toho, že tato jednotka obsahuje kompletní kontext pro řízení přenosů v jedné sadě DMA kanálů, vyplývá, že přenosy na jednom DMA kanálu mohou být prováděny pouze v jednom z DMA Endpointů, což limituje maximální propustnost jednoho DMA kanálu na propustnost jednoho PCIe Endpointu. DMA kanály jsou tak staticky rozděleny na jednotlivé PCIe Endpointy. (Například pokud máme 64 DMA kanálů a 4 PCIe Endpointy, pak přes PCIe Endpoint 0 jsou přenášena pouze data kanálů 0 až 15, přes PCIe Endpoint 1 kanálů 16 až 31 atd.) Ve skutečné aplikaci ale není třeba dosáhnout propustnosti přesahující propustnost PCIe Endpointu na jednom jediném DMA kanálu, neboť ani software není na jednom kanálu tak rychlý. Samotná jednotka DMA Cotroller v sobě obsahuje mechanismus pro Vybrané DMA kanály, takže jejich celkový počet se lineárně zvyšuje i s počtem DMA Endpointů. DMA Endpointů ale nebývá většinou více než 4.

Komponenta CrossbarX obsahuje jednotku Crossbar (přepínač), který přesouvá data mezi Input Bufferu a Output Bufferem. Kromě tohoto Crossbaru obsahuje také zřetězenou linku, která plánuje jednotlivé atomické přenosy dat na základě vstupních instrukcí o paketech.

Zpracování dat

Během průchodu DMA modulem jsou data postupně uložena v Input Bufferu a poté v Output Bufferu. Input Buffer obsahuje data jednotlivých transakcí přicházejících ze sběrnice PCIe. Střídají se zde data z různých Vybraných DMA kanálů podle toho, jak adresy do Input Bufferu přiděluje transakcím jednotka DMA Cotroller. Přidělování adres již při vytváření



Obrázek 4.6: Schéma TX NPP DMA modulu ve variantě se dvěma DMA Endpointy. Jednotka DMA Controller ovládaná ze software přes sběrnici MI32 odesílá požadavky na sběrnici PCIe MVB+MFB směrem na PCIe. Zpět se po sběrnici vracejí opačným směrem odpovědi na čtecí požadavky. Komponenta MVB+MFB Splitter je rozděluje na ty, které obsahují deskriptory (ty posílá zpět do jednotky DMA Controller), a na ty, které obsahují data paketů (ty jsou zapsány do Input Bufferu). Komponenta PCIe Checker sleduje odchozí i příchozí PCIe Transakce a podle nich určuje, kde se v Input Bufferu již nacházejí celé pakety. Tato informace je propagována z jednotlivých DMA Endpointů do jednotky Packet Planner. Packet Planner serializuje instrukce o paketech do jednoho toku a každému paketu určí cílovou pozici v Output Bufferu. Informace o pozici v Input Bufferu a v Output Bufferu je propagována do jednotky CrossbarX a ta podle ní provede přenos dat paketů. Výstup z Output Bufferu je propagován na výstup DMA modulu v podobě rozhraní ETH MVB+MFB.

nového čtecího požadavku je důležité proto, aby bylo zaručeno, že pro data přicházející z PCIe bude v Input Bufferu vždy dostatek místa.

Output Buffer obsahuje data jednotlivých paketů ze všech PCIe Endpointů uložených postupně za sebou a to ve formě, která je přípustná pro jejich odeslání na ethernetový výstup FPGA čipu podle standartu [5]. To znamená, že jednotlivé pakety mají mezi sebou dodrženou správnou mezi-paketovou mezeru a že je začátek každého z nich zarovnán na 8 B.

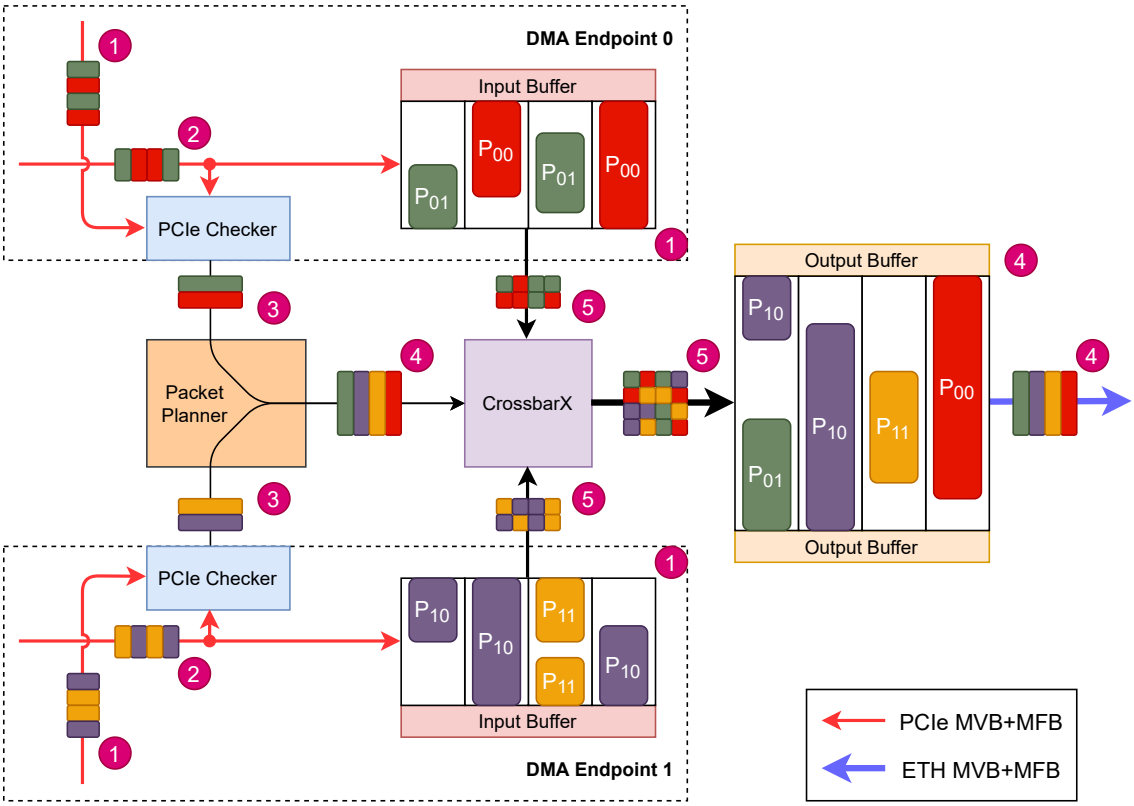
DMA modul zpracovává v jednotlivých bodech data a metadata paketů v různém pořadí. V některých případech je to způsobeno vnějšími vlivy (například že PCIe vykovává čtecí požadavky out-of-order) a v jiných zase návrhem samotným (například protože data se z jednotlivých DMA Endpointů musejí spojit do jednoho toku). Všechny různé způsoby seřazení jsou demonstrovány na příkladu na obrázku 4.7. Tento obrázek také demonstruje způsob uložení dat paketů v jednotlivých bufferech uvnitř DMA modulu.

Protože serializace dat paketů musí probíhat na datové šířce, která již umožní celkovou propustnost celého DMA modulu (pro 100 Gb/s je to 512 b, pro 400 Gb/s je to už 2048 b), musí být přesouvání dat na sběrnici prováděno nejjednodušším možným způsobem a bez jakýchkoliv výpočtů, které by prodlužovali kritické cesty. Cílem je přečíst data z každého z Input Bufferů a uložit je do Output Bufferu. Přitom ideálně v každém hodinovém cyklu zapsat celé jedno datové slovo bez ohledu na kolize při čtení a zápisu dat v bufferech. Ke kolizi dochází v případě, kdy chceme současně zapsat (nebo přečíst) části dvou rámců z jednoho bufferu, které ale ve zmíněném bufferu zasahují do stejného řádku. Tak je tomu například na obrázku 4.7 při snaze o současné čtení prvních částí paketů P_{00} a P_{10} z Input Bufferů a jejich zápisu do Output Buffer, kde tato data leží částečně ve stejné části slova, byť na jiné adrese.

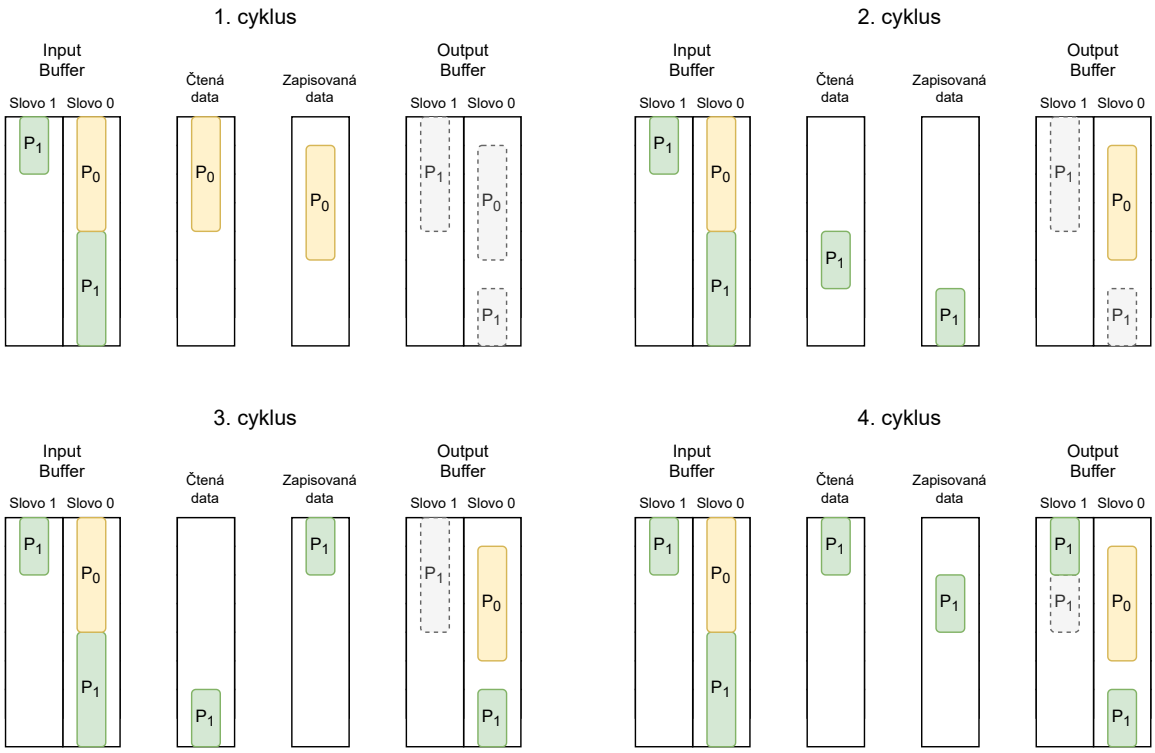
Jak Input Buffer tak Output Buffer je fyzicky rozdělen na buffery pro jednotlivé řádky, každý o šířce 8 B. Každý tento řádek má své vlastní zápisové a čtecí rozhraní s adresou nezávislou na ostatních řádcích. Díky tomu nejsme při přenosu dat mezi těmito buffery limitování na přenos po celých datových slov a proto můžeme dosáhnout vyšší propustnosti. Tento princip je demonstrován na obrázcích 4.8 a 4.9. Pro jednoduchost jsou zde přenosy ukázány pouze s jedním Input Bufferem se stejnou datovou šířkou jako má Output Buffer.

Přenos po 8-bajtových blocích byl použit již v dřívějším NDP DMA modulu. Díky tomu, že pakety na Ethernetu jsou zarovnávány na 8 bajtů a stejné zarovnávání dodržuje i systém NDP v paměti, byl tento způsob přenosů zcela dostačující pro všechny kombinace uložení dat ve zdrojovém i v cílovém bufferu. Současně šlo o značné zjednodušení oproti přenosu po jednotlivých bajtech, což vedlo ke zjednodušení multiplexorů přesouvajících data. U Input Bufferu v NPP DMA modulu se ale na zarovnání dat na 8 bajtů spolehnout nemůžeme, neboť ten obsahuje nezarovnaná data přečtená z paměti přes PCIe, který podporuje pouze transakce začínající na adresách zarovnaných na 4 bajty a data odesílaná v systému NPP žádné zarovnání nedodržují.

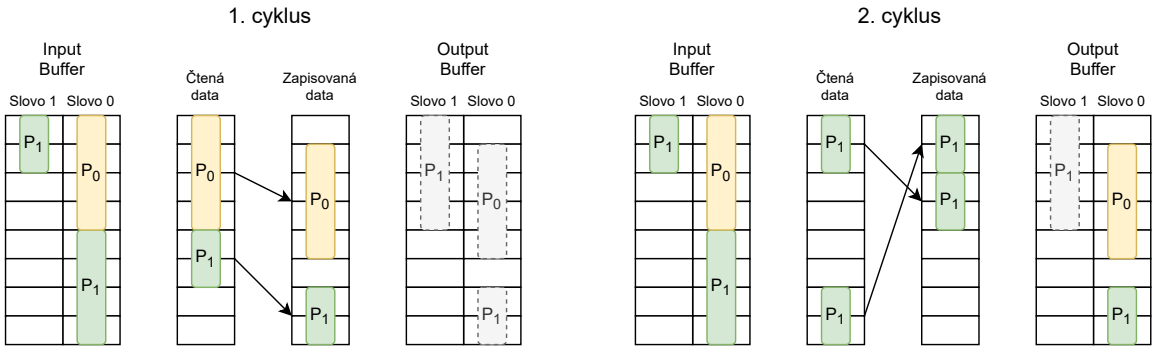
Crossbar uvnitř jednotky CrossbarX musí proto nově podporovat i přenosy datových rámců, které nejsou zarovnané na začátek 8 bajtového datového bloku. Jednou z možnosti by bylo zmenšit velikost datového bloku na jeden bajt, ale to by znamenalo až 8 násobné navýšení spotřeby zdrojů z důvodu zesložitění datových multiplexorů. Proto jsem se rozhodl aplikovat jiné řešení a to takové, které bude vyžadovat pouze přidání jednoduché funkcionality do jednotky CorssbarX, ale povede na snížení propustnosti při přenosu. K tomuto snížení bude však docházet pouze v případě, že mezi přenášenými daty se budou skutečně vyskytovat nezarovnané rámce. Tedy v případě, kdy uživatel poskytl na některém DMA kanálu data na adrese v paměti, která není zarovnaná na násobek 4 B a proto se v odpovědi



Obrázek 4.7: Příklad průchodu dat čtyř paketů (P_{00}, P_{01}, P_{10} a P_{11}) skrz NPP DMA modul. Každý z těchto paketů je rozdělen do dvou čtecích transakcí, které jsou uloženy v Input Bufferech. Pakety P_{00} a P_{01} jsou odesílány DMA Endpointem 0 a pakety P_{10} a P_{11} DMA Endpointem 1. Data paketů a transakcí (případě informace o nich) v tomto příkladě putují v pěti různých pořadích označených na schématu: 1 – Pořadí ve kterém jsou vytvořeny požadavky na čtení dat a ve kterém budou data těchto požadavků uložena v Input Bufferu. 2 – Pořadí ve kterém se přečtená data vrací ze sběrnice. (Zde by navíc mohlo dojít i k rozdělení jednotlivých odpovědí na více částí.) 3 – Pořadí ve kterém se uvnitř transakcí v Input Bufferu nacházejí konce paketů. 4 – Pořadí ve kterém jednotka Packet Planner sloučila příchozí instrukce do jednoho toku a ve kterém budou pakety uloženy v Output Bufferu a následně odeslány na výstup. 5 – Náhodné pořadí ve kterém jsou prováděny out-of-order přesuny datových bloků jednotkou CrossbarX.



Obrázek 4.8: Příklad přenosu dvou datových rámců (P_0 a P_1) mezi dvěma slovy v Input Bufferu a dvěma slovy v Output Bufferu při možnosti čtení a zápisu pouze po celých slovech. Při tomto přístupu má celý Input Buffer pouze jednu čtecí adresu vybírající celé datové slovo a stejně tak i Output Buffer pouze jednu zápisovou adresu. Kromě samotného přečtení a zapsání lze také přečtené datové slovo před zápisem rotovat a maskovat, aby mohla být data zapsána správně a nedošlo k přepsání již validních dat v cílovém slově. Jak je možné vidět, je v tomto případě k vygenerování 2 datových slov v Output Bufferu zapotřebí 4 hodinových cyklů, a proto dosažená efektivita přenosu činí pouhých 50 %.



Obrázek 4.9: Příklad přenosu dvou datových rámců (P_0 a P_1) mezi dvěma slovy v Input Bufferu a dvěma slovy v Output Bufferu při možnosti čtení a zápisu po jednotlivých blocích. Při tomto přístupu má Input Buffer jednu čtecí adresu pro každý řádek vybírající pouze blok v rámci tohoto řádku a stejně tak i Output Buffer má oddělené adresy pro řádky se stejnou velikostí. Díky tomu lze v každém cyklu zapisovat datové bloky do různých sloupců na jednotlivých řádcích a pro každý z nich vybrat datový blok čtený z jiného sloupce v PCIe bufferu. Pokud jsou přenosy datových bloků bezkolizní (žádné dva řádky Output Bufferu nepotřebují číst data ze stejného řádku v Input Bufferu), lze tyto přenosy realizovat sadou jednoduchých multiplexorů, které propojují každý cílový řádek s každým zdrojovým řádkem. Jak je možné vidět, vygenerování 2 datových slov v Output Bufferu tímto způsobem lze v tomto případě provést ve 2 hodinových cyklech, a proto dosažená efektivita přenosu činí plných 100 %.

na PCIe čtecí požadavek nebude začátek těchto dat nacházet přímo na začátku odpovědi. Místo toho se může nacházet až o jeden, dva nebo tři bajty dále.

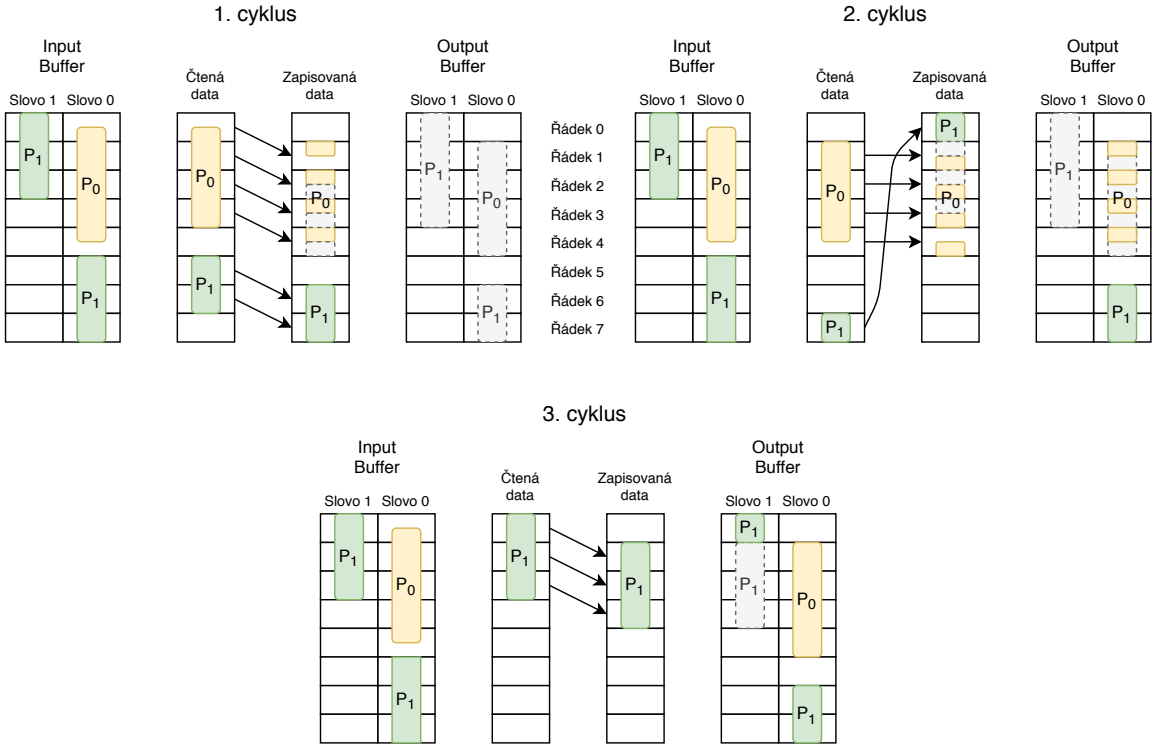
Pokud má umět CrossbarX zapsat do Output Bufferu 8-bajtový blok dat, který se v Input Bufferu nachází ve skutečnosti na hranici dvou po sobě jdoucích bloků, musí provést tento zápis ve dvou krocích jako dva datové přenosy. Navíc musí zajistit přerovnání dat v rámci tohoto bloku před jeho zápisem a také při druhém přenosu zamezit přepsání té části bloku v cílovém bufferu, která je již validní. Přerovnání lze zajistit zavedením rotace každého bloku o 0 až 3 bajty před jeho zápisem do Output Bufferu. Přepsání validních dat lze zamezit aplikováním masky na jednotlivé bajty bloku zapisované do paměti. Blokové paměti na FPGA čipech zpravidla použití masky při zápisu dat umožňují.

Můžeme si povšimnout, že oba tyto kroky se provádějí u jednotlivých přenášených bloků zcela nezávisle a proto nárůst spotřeby zdrojů nebude tak dramatický jako v případě zmenšení velikosti bloků. Příklad přenosu dat tímto způsobem je uveden na obrázku 4.10

Jedinou nevýhodou tohoto přístupu je snížení propustnosti v jednotce CorssbarX oproti prostým přenosům po blocích u dřívějšího NDP DMA modulu. Jak bylo ale již zmíněno, dojde k němu, pouze pokud se skutečně přenášejí nezarovnaná data. Pokud bude uživatelský program odesílající pakety přes kartu všechna data zarovnávat na 4 B (jako například při použití DPDK), bude propustnost stejná jako v případě prostého přenosu po blocích.

DMA Endpoint

Aby se z pohledu jednoduchosti návrhu a samotné logiky v FPGA vyplatilo rozdělení do DMA Endpointů co nejvíce, musíme se snažit, aby komponenty v rámci této obálky nemusely nikdy zpracovávat více než jeden paket (případně transakci nebo instrukci) v jed-



Obrázek 4.10: Příklad přenosu dvou datových rámců (P_0 a P_1) mezi dvěma slovy v Input Bufferu a dvěma slovy v Output Bufferu při možnosti čtení a zápisu po jednotlivých blocích a rotaci dat v rámci bloku. Přenos funguje podobně jako jednoduchý přenos po blocích zobrazený na obrázku 4.9. Je zde ale navíc i možnost rotace a maskování částí dat v rámci každého bloku před zápisem do cílového bufferu. V tomto případě, je paket P_0 posunut a jeho začátek není zarovnaný na začátek bloku. Z toho důvodu musí jeho přenos proběhnout ve dvou cyklech. V 1. cyklu jsou na řádky 1 až 4 v Output Bufferu zapisována data z řádků 0 až 3 z Input Bufferu, přičemž každý blok je rotován o polovinu své šířky a při zápisu je dolní polovina některých z nich maskována. Ve 2. cyklu se pak na řádky 1 až 4 v Output Bufferu zapisují data z řádků 1 až 4 v Input Bufferu. Jsou rotována stejným způsobem, ale maska při zápisu je u nich invertována, aby nedošlo k přepsání již validní poloviny bloku v Output Bufferu. Tím dochází ke snížení efektivity přenosu, neboť některé bloky jsme museli přenést dvakrát (řádky 1 až 3 ve slově 0 v Input Bufferu). Tato neefektivita se však dotkla pouze přenosu paketu P_0. Paket P_1 je zarovnaný a proto jeho přenos mohl proběhnout klasickým způsobem. Výsledná efektivita přenosu je proto přibližně 67 %.

nom hodinovém cyklu. Pokud je šířka sběrnice PCIe MFB 512 b a frekvence hodin 200 MHz, pak je tato podmínka splněna pro všechny jednotky uvnitř DMA Endpointu s výjimkou těch, které musejí pracovat s daty přicházejícími po MFB sběrnici s více Regiony. Pokud má totiž sběrnice PCIe MFB více než jeden Region (což platí pro většinu PCIe Endpointů podporujících propustnost 100 Gb/s a více), pak se v jednom slově mohou nacházet data několika různých transakcí. Input Buffer musí umět každou tuto část uložit na jiné místo a PCIe Checker musí zase umět zaznamenat a správně vyhodnotit přijetí všech těchto transakcí zároveň. Ve skutečnosti je práce s více-Regionovým MFB rozhraním zdrojově nejnáročnější operací v celém DMA Endpointu. Data přicházející po sběrnici v každém cyklu spolu totiž mohou kolidovat při zápisu do jednotlivých řádků v Input Bufferu a proto se musejí pro každý řádek předem bufferovat ve FIFO frontách. Transakce s deskriptory směřující do DMA Controlleru nemusejí být zpracovávány tak rychle a tudíž se u nich provádí redukce na jeden Region. I tato redukce ale znamená velikou spotřebu zdrojů.

Může ale nastat i taková situace, kdy rozhraní PCIe Endpointu bude obsahovat sběrnici s větší šířkou, nebo takovou, která je taktovaná na vyšší frekvenci. Například u sběrnice PCIe Gen4 x16 se očekává propustnost 200 Gb/s, což vede na zdvojnásobení šířky a nebo frekvence na každém Endpointu. Dvojnásobná frekvence by znamenala, že DMA modul musí být navržený a optimalizovaný na mnohem vyšší výkon, což je u něj na současných FPGA čipech velmi problematické. Dvojnásobná šířka sběrnice by zase znamenala, že modul musí umět pracovat se dvěma pakety za takt i v místech, kde dříve stačil jeden. Aby se tento problém vyřešil, bylo v návrhu stanoveno, že pracovní frekvence i šířka dat v rámci jednoho DMA Endpointu bude fixní, čímž se i jeho propustnost zafixuje na 100 Gb/s. V případě, že PCIe Endpoint bude mít propustnost vyšší, pak k tomuto PCIe Endpointu bude připojeno více DMA Endpointů a převod mezi jejich rozhraními bude implementován v rámci komponenty PTC (viz schéma zapojení DMA na obrázku 2.2).

DMA Controller

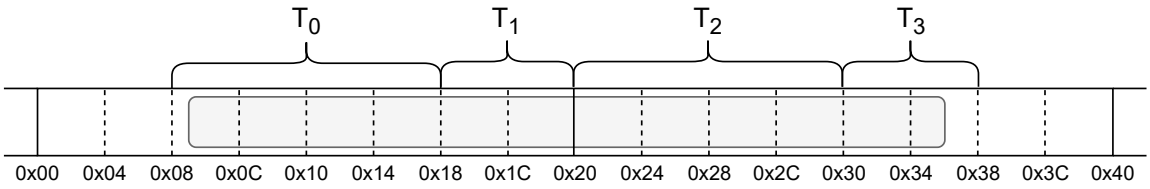
Komponenta DMA Controller má ve skutečnosti celou řadu pod-komponent:

1. Jednotka, která udržuje registry pro jednotlivé DMA kanály, které jsou přístupné přes MI32.
2. Jednotka, která dynamicky řídí přepínání, aktivace a deaktivace Vybraných DMA kanálů.
3. Jednotka pro čtení deskriptorů z deskriptorového bufferu v paměti a jejich ukládání v DMA modulu.
4. Jednotka řídící čtení dat na základě informací z uložených deskriptorů.
5. Jednotka odesílající pravidelné zápisové požadavky pro aktualizaci hodnoty HDP v paměti.

Z těchto jednotek pouze první dvě obsahují logiku závislou na celkovém počtu DMA kanálů. Ostatní části uvnitř jednotky DMA Controller (stejně jako ve zbytku DMA modulu) pracují výhradně s informacemi o aktuálních Vybraných DMA kanálech.

Odesílání čtecích požadavků na data

Jednotka, která zajišťuje odesílání čtecích požadavků na data řeší tři hlavní úkoly:



Obrázek 4.11: Příklad rozdělení čtecích požadavků při čtení jednoho souvislého bloku v paměti na transakce T_0, T_1, T_2 a T_3. V tomto příkladu se jedná o datový blok umístěný na hexadecimální adrese 0x09 s délkou 45 bajtů. Velikost stránky v paměti je zde pro jednoduchost stanovena na 32 bajtů (osm 4-bajtových bloků) a MRRS je 16 B. Transakce T_0 čte data o délce 16 B a to ze zarovnané adresy 0x08. Proto ve skutečnosti čte pouze 15 B validních dat. Transakce T_1 může číst pouze 8 B, protože jinak by došlo ke čtení ze dvou různých stránek v paměti. Transakce T_2 je jedinou transakcí v tomto příkladě, která skutečně čte celých 16 B dat. Transakce T_3 čte již úplný konec datového bloku od adresy 0x30 dál a obsahuje proto jenom 6 B validních dat.

1. Výpočet adresy a délky konkrétního čtecího požadavku
2. Rozhodnutí, zda požadavek vůbec může být odeslán
3. Přidělení adresy pro data požadavku v Input Bufferu

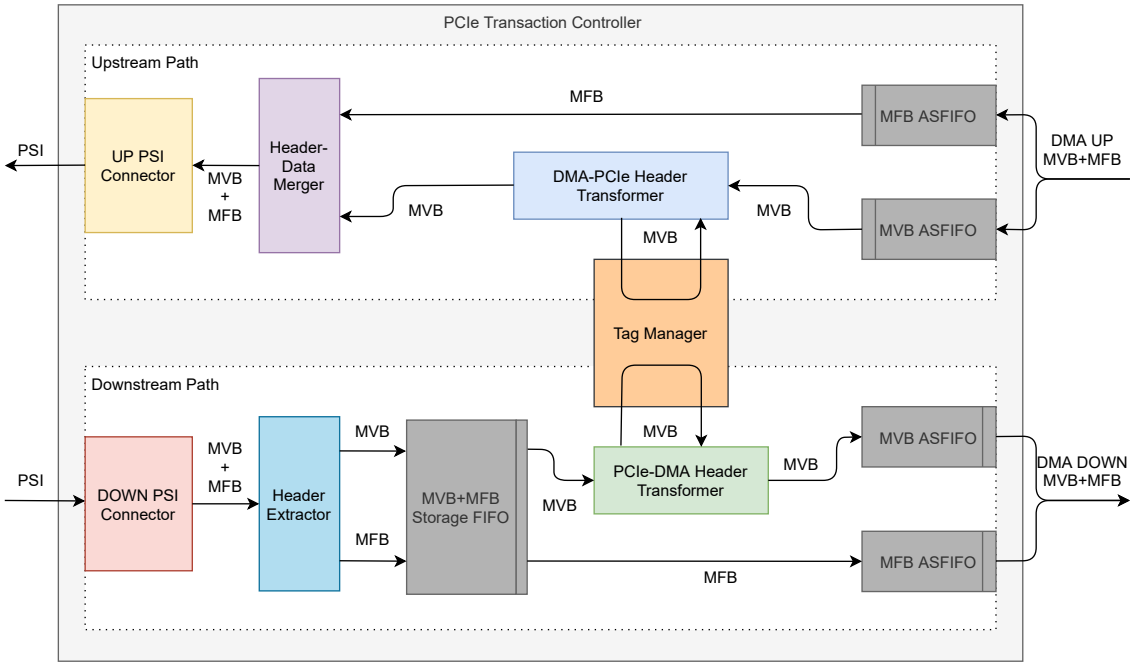
Výpočet adresy a délky požadavku je relativně složitý úkol a to hlavně kvůli omezením platícím pro požadavky na sběrnici PCIe. Při čtení dat popisovaných jedním deskriptorem, která jsou příliš veliká na jeden MRRS čtecí požadavek si DMA Controller udržuje aktuální posun v rámci prostoru tohoto deskriptoru. Na začátku deskriptoru můžou být data nezarovnaná a adresa čtení se proto musí zaokrouhlit dolů na násobek 4 B. Při výpočtu délky požadavku vybírá DMA modul minimum z těchto tří hodnot:

1. MRRS
2. Počet bajtů od aktuální adresy do konce dat popisovaných deskriptorem
3. Počet bajtů od aktuální adresy do konce stránky v paměti

Příklad rozdělení na čtecí požadavky při čtení dat popisovaných jedním deskriptorem je uveden na obrázku 4.11.

Při rozhodování, zda může být čtecí požadavek vygenerován hraje roli celá řada faktorů, jako jsou například: existence deskriptorů již stažených z paměti, probíhající deaktivace kanálu, dostatek prostoru v Input Bufferu a připravenost výstupního MVB rozhraní (hodnota signálu MVB_DST_RDY). Všechny tyto rozhodovací operace a výpočty představují relativně vysoké nároky pro FPGA platformu a jejich vykonávání v rámci jednotky DMA Controller musí proto být co nejlépe zřetězeno.

Transakcím ze všech Vybraných DMA kanálů jsou přidělovány adresy do jednoho společného prostoru v Input Bufferu. Díky tomu může být tento buffer efektivně dynamicky využíván menším nebo větším počtem kanálů podle toho, které právě odesílají více dat. Alternativní řešení, které bylo použito u dřívějšího NDP DMA modulu, je rozdělit Input Buffer staticky na skupiny slov pro každý Vybraný kanál. Pak je ale paměť mnohem hůře využita v případech, kdy se data odesílají například jen po jednom nebo dvou DMA kanálech.



Obrázek 4.12: Schéma jednotky PCIe Transaction Controller. Komponenta je připojena na jedné straně k DMA modulu přes rozhraní DMA UP MVB+MFB a DMA DOWN MVB+MFB. Z druhé strany se pak nachází PSI (*Platform-Specific Interface*) odpovídající rozhraní PCIe Endpointu na konkrétním FPGA čipu. Celá jednotka je rozdělena na část Upstream Path a Downstream Path. V Upstream části putují požadavky odesílané z DMA modulu na sběrnici PCIe. V Downstream části jsou přenášeny odpovědi na čtecí požadavky přicházející ze sběrnice zpět. Komponenta Tag Manager komunikuje s oběma těmito částmi.

PCIe Checker

Úkolem této jednotky je kontrolovat faktickou přítomnost dat uvnitř Input Bufferu, aby Packet Planner (a posléze CrossbarX) pracovali pouze s daty paketů, které již byly skutečně přijaty z PCIe. Dále také na základě informace z jednotky CorssbarX, která potvrzuje přenos každé kompletní transakce, zjišťovat, která data v Input Bufferu byla již přečtena a mohou být přepsána a informovat o této skutečnosti DMA Controller. (Tato komunikace není na obrázku 4.6 zobrazena z důvodu zjednodušení.) PCIe Checker vysílá na svůj výstup instrukce popisující pakety v pořadí, v jakém se v Input Bufferu nacházejí konce paketů. Toto pořadí nemusí být nutně stejné jako pořadí začátků paketů z důvodu rozdělení paketů na více transakcí. To je vidět například u paketů P_{10} a P_{11} na obrázku 4.7.

4.4 PCIe Transaction Controller

Jak už bylo zmíněno výše, probíhá přenos DMA požadavků mezi DMA modulem a PCIe Endpointem v FPGA přes jednotku PCIe Transaction Controller (PTC). Je to z toho důvodu, že DMA modul má být použitelný na libovolném FPGA čipu a potřebuje být nezávislý na formátu rozhraní poskytovaném PCIe Endpointem na specifické platformě. Tato komponenta řeší některé aspekty komunikace přes sběrnici PCIe. Součástí mojí práce při vývoji NPP DMA modulu byl návrh a spolupráce na implementaci i této komponenty.

Obrázek 4.12 obsahuje schéma vnitřního zapojení v komponentě PTC. Celé schéma je rozděleno na část „Upstream Path“, tedy datovou cestu směřující z DMA modulu na PCIe, a „Downstream Path“, tedy datovou cestu zpět.

PTC zajišťuje tyto tři hlavní úkony:

1. Adaptuje rozhraní PCIe Endpointu na rozhraní MVB+MFB.
2. Provádí mapování PCIe Tagů na čtecí požadavky.
3. Zajišťuje zaručené ukládání všech příchozích odpovědí.

Význam prvního bodu byl již zmíněn. Kromě odlišností v řízení sběrnice pro přenos dat se odstínění týká také formátu hlaviček v požadavcích. V rámci našich komponent připojených přes PTC používáme vlastní formát nazvaný „DMA hlavička“ a ten je převáděn uvnitř PTC na PCIe hlavičku v jednom směru a zpět ve směru opačném. Přesný formát PCIe hlavičky je závislý na typu PCIe Endpointu.

Druhý bod souvisí se zajištěním unikátnosti PCIe Tagu u všech čtecích požadavků, které v daný okamžik čekají na odpověď. Vyžadovat po všech komponentách, které vytvářejí čtecí požadavky, aby dodržovali tuto zásadu interně i vzájemně mezi sebou, by bylo komplikované. Jednou možností by bylo staticky rozdělit prostor hodnot Tagů na jednotlivé komponenty, což by ale zamezilo kterékoli z nich využít všechny hodnoty a mohlo by tak vést ke snížení propustnosti. Druhou možností je vytvoření centrální jednotky, kterou budou tyto komponenty žádat o přidělení jednotlivých Tagů. Tato komponenta by ale byla dosti komplexní a její složitost by výrazně rostla s počtem připojených jednotek žádajících o Tagy.

Aby se tento problém vyřešil, nachází se v DMA hlavičce kromě hodnoty Tag ještě pole „Unit ID“. Každá komponenta generující čtecí požadavky má přidělenu svou unikátní hodnotu Unit ID a tu nastavuje do hlavičky všech svých požadavků. Díky tomu může libovolná komponenta využívat všechny hodnoty Tagu, neboť její požadavky lze rozlišit od požadavků ostatních komponent právě podle políčka Unit ID. Prostor Tagů, který je u PCIe hlavičky globální je tedy u DMA hlavičky lokální pro každou komponentu.

Přitom celkový prostor všech kombinací Unit ID a Tagů může být podstatně větší, než prostor PCIe Tagů poskytovaných PCIe Endpointem. Větší může být dokonce i prostor Tagů poskytovaných pro jednu každou komponentu. Stačí, pokud bitová šířka Tagu v DMA hlavičce je vyšší než šířka Tagu v PCIe hlavičce. Mapování mezi těmito dvěma prostory hodnot jedním a druhým směrem je prováděno dynamicky právě uvnitř PTC a to jednotkou Tag Manager.

Poslední zmíněná úloha, kterou vykonává jednotka PTC je ukládání odpovědí na čtecí požadavky. U některých modelů FPGA čipů od firmy Xilinx je u PCIe Endpointu vyžadováno, aby komponenta přijímající odpovědi na požadavky přicházející ze sběrnice byla vždy schopna přijmout příchozí data. PCIe Endpoint v takovém případě nemá ve směru Downstream žádný vstupní signál odpovídající signálu DST_RDY u handshake protokolu, kterým by bylo možno příchozí tok dat zastavit. Z toho důvodu obsahuje PTC jednotku MVB+MFB Storage FIFO pro ukládání příchozích dat a je zajištěno, aby bylo v jednom okamžiku vystaveno pouze tolik čtecích požadavků, pro kolik je možno zajistit uložení jejich dat do této fronty bez jejího zaplnění. Toto Storage FIFO lze odstranit v případech, kdy PCIe Endpoint podporuje Downstream DST_RDY. Všechny komponenty, které odesílají čtecí požadavky (komponenta na čtení dat a komponenta na čtení deskriptorů v DMA modulu), obsahují totiž svoje vlastní mechanismy, které zajišťují odesílání pouze tolika

požadavků, kolik dat je možno uložit v lokáním bufferu. Tyto mechanismy mohou však způsobit i občasné shození hodnoty DST_RDY na 0.

Význam všech jednotlivých pod-komponent PTC je následující:

- MVB ASFIFO
  - Ukládá hlavičky přicházející po sběrnici MVB a umožňuje jejich asynchronní přenos mezi hodinovým signálem PCIe Endpointu a DMA modulu.
- MFB ASFIFO
  - Ukládá rámce přicházející po sběrnici MFB a umožňuje jejich asynchronní přenos mezi hodinovým signálem PCIe Endpointu a DMA modulu.
- DMA-PCIe Header Transformer
  - Provádí převod mezi formátem DMA hlavičky a formátem vyžadovaným PCIe Endpointem.
- Header-Data Merger
  - Vkládá hlavičky před jednotlivé rámce a spojuje tak MVB+MFB do jediného MFB rozhraní.
  - Pokud připojený PCIe Endpoint umožňuje odesílat hlavičky a data na oddělených rozhraních, není tato jednotka přítomna.
- UP PSI Connector
  - Provádí převod z rozhraní MFB (případně MVB+MFB, pokud není přítomen Header-Data Merger) na specifické rozhraní PCIe Endpointu.
- DOWN PSI Connector
  - Provádí převod ze specifického rozhraní PCIe Endpointu na rozhraní MFB (případně MVB+MFB, pokud není přítomen Header Extractor).
- Header Extractor
  - Extrahuje hlavičky nacházející se na začátku rámců na vstupním MFB rozhraní na výstupní MVB rozhraní.
  - Pokud připojený PCIe Endpoint propaguje hlavičky a data na oddělených rozhraních, není tato jednotka přítomna.
- MVB+MFB Storage FIFO
  - Ukládá hlavičky a data příchozích transakcí ve dvou oddělených FIFO frontách.
  - Uvolňuje hlavičky uložených transakcí až poté, co byla odpovídající data na MFB kompletně odebrána z FIFO fronty následující komponentou. Tím je zajištěno u každé hlavičky na výstupním MVB, že popisovaná transakce již nezabírá žádný prostor v této jednotce.
  - Pokud připojený PCIe Endpoint podporuje v Downstream signál DST_RDY není tato komponenta přítomna.

- PCIe-DMA Header Transformer
  - Provádí převod mezi formátem hlavičky PCIe Endpointu a formátem DMA hlavičky.
- Tag Manager
  - Provádí mapování hodnot DMA Tag a Unit ID u čtecích požadavků na hodnotu PCIe Tag.
  - Provádí zpětné mapování hodnoty PCIe Tag u odpovědí na čtecí požadavky na hodnoty DMA Tag a Unit ID.
  - Provádí kontrolu zaplnění jednotky Storage FIFO (pokud je přítomna) tak, že dovoluje odeslat pouze tolik čtecích požadavků, pro kolik je dostatek prostoru v komponentě Storage FIFO.

Je třeba si uvědomit, že všechny komponenty uvnitř PTC musejí být schopny pracovat pro libovolné nastavení sběrnic MVB a MFB, které bude odpovídat nastavení sběrnice na PCIe Endpointu. Většinou se jedná o nastavení, kde MVB obsahuje dvě hlavičky a MFB dva Regiony. Například na PCIe Endpointu pro Generaci 3 x16 u FPGA *Ultrascale+* je MFB nastaveno na $\langle 32, 8, 1, 2\rangle$ v Upstream a $\langle 32, 4, 1, 4\rangle$ v Downstream. (Zde si můžeme všimnout, že sběrnice má v obou případech šířku Položky nastavenou na 32 b, což odpovídá zarovnávání délek požadavků na sběrnici PCIe.)

Jednotka PTC obsahuje parametry, kterými lze před syntézou určit, pro který typ PCIe Endpointu jej chceme syntetizovat. Při přidávání podpory pro nový typ PCIe Endpointu dochází zpravidla k úpravám pouze v komponentách DMA-PCIe Header Transformer, PCIe-DMA Header Transformer, UP PCIe Interface Connector a DOWN PCIe Interface Connector.

Připojení více DMA Endpointů

Jak se uvádí v předchozí sekci 4.3, je u nového NPP DMA modulu vyžadována podpora připojení několika DMA Endpointů k jednomu PTC přes několik MVB+MFB rozhraní (Upstream i Downstream). Protože v současnosti není případ, kdy by bylo nutno připojovat k jednomu PCIe Endpointu více než dva DMA Endpointy, jsou zde uvedená řešení navržena právě pro tuto variantu.

Protože mezi jednotkou PTC a PCIe Endpointem se nevyskytuje žádný asynchronní přechod, musí celý modul PTC pracovat se stejným hodinovým signálem jako samotný PCIe Endpoint a tudíž i na stejné frekvenci. Teprve asynchronní FIFO fronty na rozhraní směrem k DMA modulu převádějí data na hodinový signál DMA modulu. Z toho vyplývá, že pokud bude PCIe Endpoint s propustností 200 Gb/s pracovat na frekvenci 400 MHz, musí na této frekvenci pracovat i jednotka PTC. To stejné platí i v případě, že zmíněný PCIe Endpoint bude mít frekvenci 200 MHz, ale datovou šířku 1024 b.

Aby mohla mít jednotka PTC dvojí nezávislá rozhraní MVB+MFB, dojde ke zdvojení počtu asynchronních FIFO front na těchto rozhraních. Každá z těchto front provádí implicitně převod z frekvence 400 MHz na frekvenci DMA modulu. Při nastavení datové šířky front na 1024 b (u MVB front na tolik Položek, kolik podporuje PCIe Endpoint) a přidáním logiky pro převod této šířky směrem k DMA modulu můžou tyto fronty provádět i úpravu v tomto směru.

Sloučení MVB+MFB sběrnic do jedné ve směru Upstream a naopak rozdělení do dvou ve směru Downstream se pak provede hned na opačné straně FIFO front a to komponentami MVB+MFB Merger a MVB+MFB Splitter, které jsou určeny právě k tomuto účelu.

Kapitola 5

Implementace

Tato kapitola obsahuje detailnější popis návrhu NPP DMA modulu, stejně jako některých jeho významných komponent. Vybrané komponenty jsou významné svojí kritičností z pohledu složitosti v závislosti na počtu DMA Endpointů. Některé z nich jsou důležité, protože se v návrhu vyskytují na více místech anebo je lze využít i pro úlohy mimo TX DMA modul.

5.1 Detailní popis NPP DMA modulu

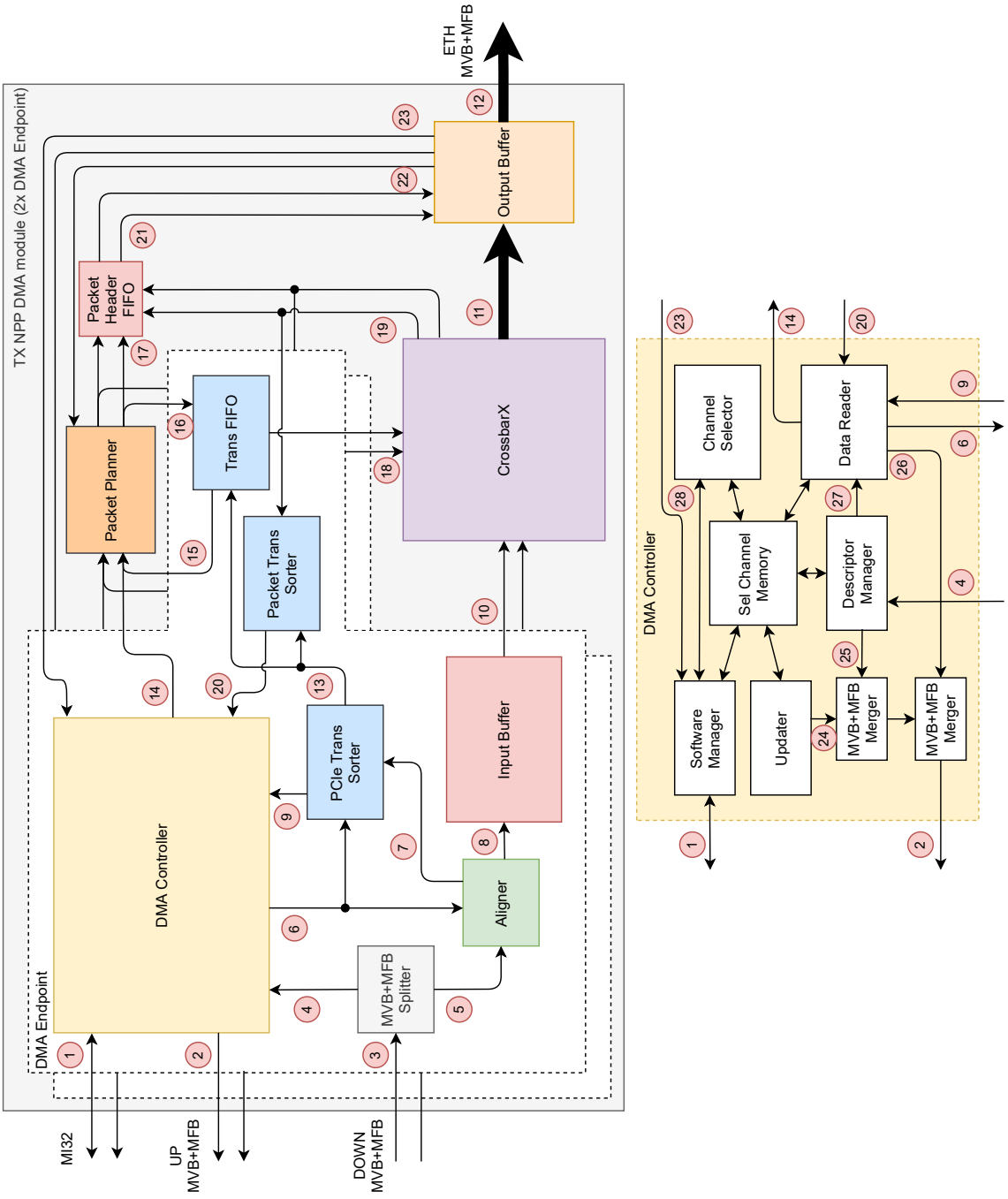
Při návrhu nové verze DMA modulu jsem se soustředil především na odstranění klíčových nedostatků návrhu předchozího. Mezi druhořadé cíle patřila snaha o znovupoužití – případně i vylepšení – prvků, které byly u předchozího návrhu výhodné a také o přizpůsobení nového DMA modulu pro podporu systému NPP. Celkové detailní schéma NPP TX DMA modulu je zobrazeno na obrázku 5.1. Stejně jako v předchozí kapitole na obrázku 4.6 se jedná o zapojení DMA modulu se dvěma DMA Endpointy.

Shodně s méně detailním schématem DMA modulu v předchozí kapitole se i zde nacházejí komponenty DMA Controller, MVB+MFB Splitter, Input Buffer, Packet Planner, CrossbarX a Output Buffer. Funkce komponenty PCIe Checker je zde ale nahrazena jednotkami PCIe Transaction Sorter, Packet Transaction Sorter a Transaction FIFO. A jsou zde i dvě nové komponenty Aligner a Packet Header FIFO.

Vnitřní propojení

Schéma na obrázku 5.1 obsahuje číselné značení pro jednotlivá propojení mezi vnitřními komponentami DMA modulu a pro vnější rozhraní. Zde je uveden seznam popisu jednotlivých propojení jako doplněk k popisu významu jednotlivých komponent uvedenému v dalších podsekcích.

1. MI32 rozhraní pro komunikaci s CPU přes PCIe
2. UP MVB+MFB rozhraní pro odesílání čtecích a zápisových požadavků do paměti přes PCIe
3. DOWN MVB+MFB rozhraní pro příjem odpovědí na čtecí požadavky z PCIe
4. MVB+MFB rozhraní obsahující odpovědi na čtecí požadavky s deskriptory
5. MVB+MFB rozhraní obsahující odpovědi na čtecí požadavky s daty paketů



Obrázek 5.1: Schéma TX NPP DMA modulu ve variantě se dvěma DMA Endpointy (nahoře) včetně detailnějšího schématu komponenty DMA Controller (dole). Podrobný popis jednotlivých očíslovaných propojení je uveden v textu této sekce.

6. Informace o transakcích pro čtení dat odesílaných na PCIe
7. Potvrzení o zapsání kompletní odpovědi na jednotlivé transakce do Input Bufferu (teoreticky v náhodném pořadí určeném vykonáním požadavků na PCIe)
8. Data transakcí s pakety zapisovaná na příslušnou adresu do Input Bufferu
9. Potvrzení o zapsání kompletní odpovědi na jednotlivé transakce do Input Bufferu (ve stejném pořadí jako na rozhraní 6)
10. Data transakcí čtená za účelem přesunu do Output Bufferu
11. Data paketů zapisovaná do Output Bufferu
12. ETH MVB+MFB obsahující souvislý proud paketů ze všech DMA Endpointů
13. Informace o transakcích prokazatelně uložených v Input Bufferu (ve stejném pořadí jako na rozhraních 6 a 9)
14. Metadata určená k propagování s každým paketem na ETH MVB rozhraní (obsahují uživatelskou hlavičku extrahovanou z deskriptoru uvedenou v sekci 4.1)
15. Informace o délce jednotlivých paketů
16. Informace o adrese v Output Bufferu určené pro jednotlivé pakety
17. Metadata určená k propagování s každým paketem na ETH MVB rozhraní společně s adresou paketu v Output Bufferu
18. Instrukce pro vykonání přenosu dat z Input Bufferu do Output Bufferu
19. Potvrzení o vykonání instrukce pro přenos dat (ve stejném pořadí jako na rozhraní 18)
20. Potvrzení o uvolnění dat transakce z Input Bufferu (ve stejném pořadí jako na rozhraních 6, 9 a 13)
21. Metadata určená k propagování s každým paketem na ETH MVB rozhraní společně s adresou paketu v Output Bufferu pro již potvrzené pakety (ve stejném pořadí jako na rozhraní 17)
22. Čtecí ukazatel v Output Bufferu (k určování zaplnění bufferu)
23. Potvrzení o kompletním odeslání paketu z DMA modulu (pro inkrementaci statistických čítačů odeslaných paketů a odeslaných bajtů)
24. MVB+MFB požadavky pro zápis aktualizace HDP v paměti
25. MVB požadavky pro čtení deskriptorů z paměti
26. MVB požadavky pro čtení dat paketů z paměti
27. Deskriptory přečtené z paměti přes PCIe
28. Rozhraní pro řízení spouštění a zastavování DMA kanálů

Rozhraní připojená ke komponentě Selected Channel Memory zde popsaná nejsou, neboť se jedná o rozsáhlý výčet zápisových a čtecích rozhraní pro paměť. Blíže jsou popsána dále v popisu samotné komponenty Selected Channel Memory.

Komponenty NPP DMA modulu

TX NPP DMA modul na obrázku 5.1 obsahuje kromě komponent uvedených v předchozí kapitole ještě tyto:

1. Aligner
   - Zajišťuje, aby data příchozích PCIe transakce byla zapsána na správnou pozici v Input Bufferu.
   - Pro řešení kolizí při zápisu dat ze dvou rámců nacházejících se v jednom vstupním slově, obsahuje vyrovnávací paměť, která umožňuje několikrát zapsat současně do každého řádku Input Bufferu. Pokud se tato paměť zaplní, je zastaven vstupní proud dat.
   - Odesílá informaci o kompletním zapsání každé transakce do Input Bufferu do jednotky PCIe Transaction Sorter.
2. PCIe Transaction Sorter
   - Přijímá a ukládá informaci o transakcích odeslaných na PCIe a ve stejném pořadí odesílá tuto informaci na výstup až poté, co jsou jejich data uložena v Input Bufferu.
   - Odesílá uvolněné hodnoty DMA Tagu zpět jednotce generující požadavky.
3. Packet Transaction Sorter
   - Přijímá a ukládá informaci o transakcích uložených v Input Bufferu a ve stejném pořadí odesílá tuto informaci na výstup až poté, co jsou jejich data zapsána do Output Bufferu.
4. Transaction FIFO
   - Ukládá infromaci o příchozích transakcích.
   - Akumuluje délku paketů rozdělených do více transakcí.
   - Informaci o délce paketů odesílá do jednotky Packet Planner.
   - Po získání cílové adresy paketu v Output Bufferu z jednotky Packet Planner, odesílá informace o transakcích tvořících tento paket do jednotky CrossbarX.
5. Packet Header FIFO
   - Přijímá hlavičky paketů extrahované z deskriptorů v tom pořadí, v jakém budou pakety uloženy v Output Bufferu a odesílá je v tomto pořadí až poté, co byla jejich data skutečně zapsána do Output Bufferu.

Komponenty jednotky DMA Controller

DMA Controller se v rámci jednoho DMA Endpointu stará o komunikaci s ovladačem, o čtení a zpracování deskriptorů a o odesílání požadavků na čtení dat. Odpovědi na tyto požadavky obsahující data paketů již do DMA Controlleru nepřicházejí. Místo toho jsou přesměrovány do komponenty Aligner, která dostane z DMA Controlleru informaci o tom, které odpovědi má očekávat, při odesílání samotného požadavku. Samotný DMA Controller se skládá z těchto hlavních pod-komponent:

1. Selected Channel Memory

    - Paměť obsahující položky potřebné pro práci a vzájemnou komunikaci mezi ostatními komponentami DMA Controlleru pro každý Vybraný DMA kanál. Jedná se o položky jako například: index DMA kanálu mapovaného na daný Vybraný kanál, hodnoty SDP a HDP, informaci o právě probíhající deaktivaci Vybraného kanálu. Tyto položky jsou propagovány přes čtecí a zápisová rozhraní okolním jednotkám.

2. Software Manager

    - Obsahuje registry pro jednotlivé DMA kanály umožňující komunikaci s ovladačem podle protokolu NPP přes rozhraní MI32.
    - Propaguje informace nutné ke spouštění a zastavování DMA kanálů do jednotky Channel Selector.
    - Zajišťuje propagaci aktuální hodnoty MI registru s hodnotou SDP do Selected Channel Memory a propagaci aktuální hodnoty HDP opačným směrem.

3. Channel Selector

    - Řídí dynamické mapování DMA kanálů na Vybrané DMA kanály. Průběžnou aktivací a deaktivací Vybraných DMA kanálů.

4. Descriptor Manager

    - Odesílá požadavky na čtení deskriptorů z deskriptorového bufferu v paměti hostitelského počítače.
    - Zpracovává formát přijatých deskriptorů podle protokolu NPP.
    - Ukládá deskriptory a umožňuje přístup k nim jednotce Data Reader.

5. Data Reader

    - Odesílá čtecí požadavky na čtení dat paketů z paměti hostitelského počítače na základě deskriptorů dostupných z jednotky Descriptor Manager.
    - Před odesláním každého požadavku kontroluje dostupnost volného prostoru v Input Bufferu a přiděluje čtecí transakci adresu do něj.
    - Informaci o každém odeslaném požadavku posílá do jednotek Aligner a PCIe Transaction Sorter.
    - Ukládá informaci o uživatelských hlavičkách paketů extrahovaných z deskriptorů a poskytuje ji jednotce Packet Planner.

6. Updater

    - Pravidelně kontroluje změnu hodnoty HDP v Selected Channel Memory a v určitých intervalech odesílá aktualizovanou hodnotu do paměti podle protokolu NPP.

7. MVB+MFB Merger

    - Slučuje dvě vstupní sběrnice MVB+MFB do jedné výstupní.

5.2 FIFOX

FIFOX je název komponenty, která byla vytvořena v rámci vývoje na našem oddělení ve sdružení CESNET. Jedná se o komponentu, která zaobaluje nejrůznější varianty implementace FIFO fronty na různých FPGA čipech. Jejím hlavním účelem je poskytnout uživateli (jinému vývojáři) univerzální jednoduché rozhraní pro použití FIFO fronty, při současném zajištění maximální kompatibility a efektivity využití zdrojů na libovolném cílovém FPGA. Komponenta dokáže, mimo jiné, při syntéze automaticky vybrat na základě požadovaného počtu položek ve FIFO frontě a modelu cílového čipu optimální variantu implementace vnitřní paměti. Možné varianty jsou paměťové LUT (*Look-Up Table*), BRAM (*Block Random Access Memory*) anebo URAM (*Ultra Random Access Memory*, existuje pouze pro určité modely FPGA od firmy Xilinx). Z pohledu této práce je důležité, že se jedná o komponentu implementující univerzální FIFO frontu s jednotným rozhraním obsahujícím handshake na vstupu i na výstupu (zde pomocí signálů *WRITE* a *FULL* na zápisové straně a *EMPTY* a *READ* na čtecí straně).

5.3 FIFOX Multi

Na základě pozitivní zkušenosti s komponentou FIFOX jsem navrhl a implementoval komponentu nazvanou FIFOX Multi. FIFOX Multi implementuje FIFO frontu, u které lze provádět současný zápis a současné čtení několika položek v každém cyklu. Počet zapisovaných a čtených položek se přitom může lišit v jednotlivých cyklech a je pouze shora omezen parametrem zadaným při zapojení komponenty.

Jediným dalším omezením je, že při čtení položek na výstupním rozhraní musí uživatel číst položky postupně a nesmí se snažit žádnou vynechávat. Pokud například nastavíme u FIFOX Multi počet zápisových portů na 5 a počet čtecích portů na 6, potom můžeme v jednom cyklu zapsat současně na libovolnou kombinaci ze vstupních 5 portů (například můžeme zapsat na port s indexem 1 a port s indexem 4 a na ostatních portech nic nezapisovat). Při čtení však musí platit podmínka, že pokud čteme položku z portu n, kde $n \in \{0, 1, 2, 3, 4, 5\}$, potom musíme současně provádět čtení i všech položek, které se nacházejí ve frontě před touto položkou. Totiž těch, které jsou aktuálně vystaveny na portech i , kde $i \in \{0, 1, 2, 3, 4, 5\} \wedge i < n$.

Jednotka FIFOX Multi je v DMA modulu využita na mnoha místech. Například v komponentách Descriptor Manager, Data Reader, Aligner, Packet Planner, Packet Header FIFO, Output Buffer a také v rámci jednotky CrossbarX. Je velmi užitečná hlavně při zpracovávání většího počtu položek v jednom cyklu na sběrnicích MVB a MFB. Pochopení jejího chování je důležité pro pochopení implementace některých dalších komponent popsaných v této kapitole.

Rozhraní komponenty FIFOX Multi je popsáno v tabulce 5.1. Zde je možno si povšimnout, že zatímco čtecí rozhraní poskytuje jeden bit signálu *EMPTY* pro každou položku, zápisové rozhraní obsahuje pouze jeden bit široký signál *FULL* pro celý vstup. To je proto, že pokud nastavený počet čtecích portů je vyšší než aktuální počet položek ve FIFOX Multi, musí být uživatel schopen zjistit, že některé z položek, které vidí na výstupu, jsou validní – u nich bude signál *EMPTY* mít hodnotu 0 – a některé ne – u nich bude *EMPTY* s hodnotou 1. Na zápisové straně FIFOX Multi stačí uživateli informace o tom, jestli byl jeho aktuálně nastavený vstup přijat do fronty nebo ne. Komponenta FIFOX Multi nastaví signál *FULL* na 0 v okamžiku, kdy je schopna přijmout maximální možný počet položek (tolik, kolik je zápisových portů). Jakmile není komponenta schopna přijmout tolik no-

Název signálu	Směr	Šířka [b]
DI	vstupní	Počet zápisových portů * Šířka datové položky
WR	vstupní	Počet zápisových portů
FULL	výstupní	1
AFULL	výstupní	1
DO	výstupní	Počet čtecích portů * Šířka datové položky
RD	vstupní	Počet čtecích portů
EMPTY	výstupní	Počet čtecích portů
AEMPTY	výstupní	1

Tabulka 5.1: Výčet signálů přítomných rozhraní FIFOX Multi.

vých položek, kolik je zápisových portů, pak nastaví signál *FULL* na 1 a přestane přijímat veškerý nový vstup bez ohledu na to, kolik položek chce uživatel v danou chvíli skutečně zapsat. Tento přístup slouží ke zjednodušení logiky určující hodnotu signálu *FULL* a tím ke zkrácení logických cest, což je důležité, protože uživatel může teoreticky signálem *FULL* řídit ještě další komplexní operace, které tuto cestu déle prodlužují.

Signály *AFULL* (*Almost Full*) a *AEMPTY* (*Almost Empty*) slouží ke zjištění, nakolik je daná fronta prázdná případně plná. Uživatel může u FIFOX Multi nastavit parametry „*AFULL Offset*“ a „*AEMPTY Offset*“. Pokud počet volných míst ve FIFOX Multi klesne pod hodnotu *AFULL Offset*, nastaví se signál *AFULL* na 1. Pokud počet platných položek ve FIFOX Multi klesne pod hodnotu *AEMPTY Offset*, nastaví se signál *AEMPTY* na 1.

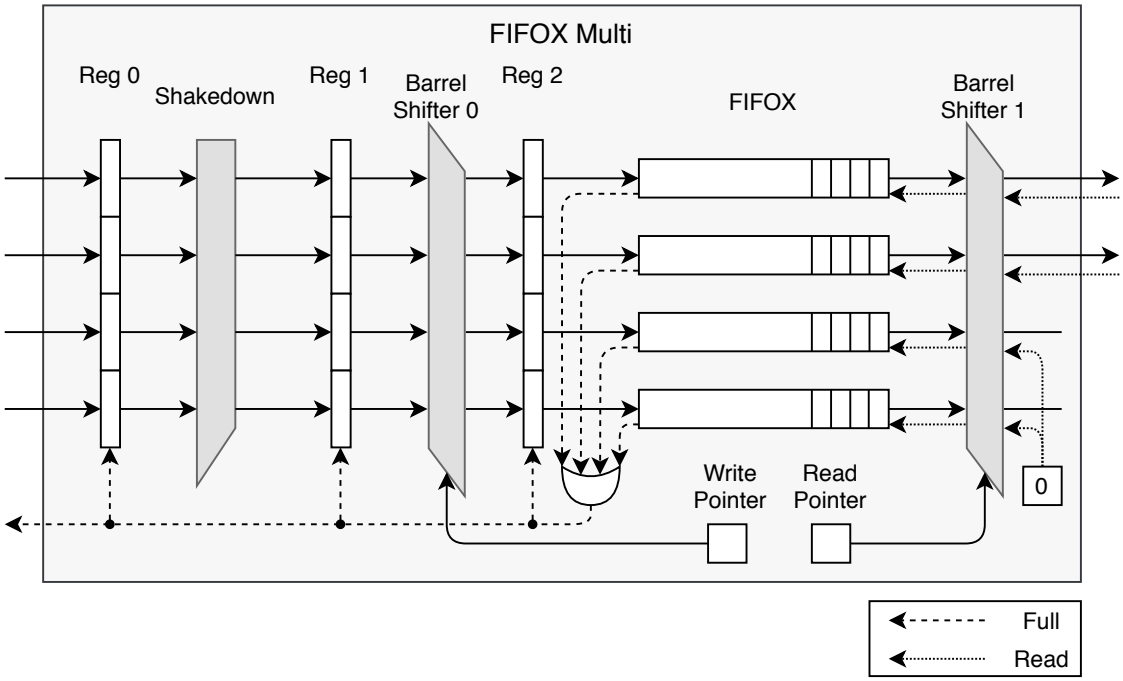
Schéma komponenty FIFOX Multi je vyobrazeno na obrázku 5.2. Hlavní součástí komponenty FIFOX Multi je sada FIFOX front, které jsou použity pro samotné ukládání datových položek. Vstupní data jsou propagována přes komponentu „Shakedown“. Tato jednotka zajistí, že pokud se na jejím vstupu objeví libovolná kombinace platných a neplatných položek, pak na jejím výstupu budou všechny platné položky umístěny na portech s nejnižším indexem a ostatní výstupní položky budou neplatné. Pokud například bude mít v případě na obrázku 5.2 registr Reg0 platné položky na pozicích 1 a 3, pak do registru Reg1 se tyto položky uloží na pozice 0 a 1 a pozice 2 a 3 budou neplatné. Jednotka Shakedown je vnitřně popsána zcela behaviorálně. Ve skutečnosti se její struktura skládá z řady multiplexorů řízených enkodéry a dalšími logickými funkcemi pracujícími se signály označujícími platnost jednotlivých vstupů. Pro vysoký počet portů může tato komponenta obsahovat velmi komplexní logiku a z toho důvodu je umístěna mezi dva registrové stupně odděleně od ostatní logiky.

V případě, že je používán signál *AFULL* nebo *AEMPTY*, obsahuje komponenta FIFOX Multi také registr, který počítá aktuální počet položek uložených uvnitř pro správné nastavování těchto signálů.

5.4 Transaction Sorter

Komponenta Transaction Sorter se v NPP DMA modulu vyskytuje celkem třikrát. Jednou jako PCIe Transaction Sorter, dále jako Packet Transaction Sorter, a nakonec jako podkomponenta uvnitř jednotky CrossbarX (popsáno v sekci 5.5).

Jak bylo již řečeno v popisu aplikací této komponenty v DMA modulu, je úkolem této komponenty přijímat „Transakce“ a potvrzení na tyto Transakce a následně potvrzené Transakce opět uvolňovat. Výstupní Transakce mají přitom stejné pořadí jako ty vstupní,



Obrázek 5.2: Schéma komponenty FIFOX Multi v nastavení se čtyřmi zápisovými a dvěma čtecími porty. Data ze vstupních rozhraní (vlevo) jsou uložena do registru „Reg0“. Následně jsou jednotkou „Shakedown“ shromážděny pouze skutečně validní položky na nejnižší indexy registru „Reg1“. Takto uspořádané položky jsou zapsány do registru „Reg2“ a následně do oddělených FIFOX front skrz Barrel Shifter 0. Ten zajistí, aby se nová položka s nejnižším indexem zapsala do fronty, do které bylo naposledy zapisováno před nejdelší dobou. (Nové položky zapsané na port s nižším indexem se tak budou ve FIFOX Multi nacházet před novými položkami zapsanými na port vyšším indexem.) Zápis veškerých vstupních dat je pozastaven jakmile kterákoliv z FIFOX front nastaví signál *FULL* na 1. Na výstupu jsou data z čel FIFOX front propagována přes Barrel Shifter 1, který zde zajišťuje, aby na indexu 0 byl přítomen prvek z té FIFOX fronty, ze které bylo naposledy čteno před nejdelší dobou (ten nejstarší, tedy ten na čele FIFOX Multi). Společně s daty jsou propagovány i signáy *EMPTY* jednotlivých FIFOX front. Čtecí signály jsou propagovány přes Barrel Shifter 1 nazpět (ve skutečnosti přes jiný Barrel Shifter, který rotuje data opačným směrem a má společný řídící signál s Barrel Shifterem 1). Protože FIFOX Multi je nastaveno na 2 čtecí porty, nejsou výstupy 2 a 3 nikam zapojeny a čtecí signál na těchto výstupech je nastaven pevně na 0. Tím dojde v syntézním nástroji k optimalizaci části logiky Barrel Shifteru 1. Registry „Write Pointer“ a „Read Pointer“ obsahují v tomto případě 2-bitový čítač řídící Barrel Shifter 0, respektive Barrel Shifter 1. Registr Write pointer se při zápisu do FIFOX front inkrementuje přesně o počet validních zapsaných položek. Registr Read pointer se v každém cyklu inkrementuje o počet přečtených položek.

ale příchozí potvrzení mohou být obecně náhodně seřazeny. Jednotlivé Transakce se odlišují identifikátory (dále „Transakční ID“ nebo „ID“). Počet vstupních Transakcí v jednom cyklu může být přitom libovolné přirozené číslo, stejně jako počet výstupních Transakcí a i počet příchozích potvrzení.

Základem této komponenty je FIFOX Multi a registrové pole příznaků. Ve FIFOX Multi jsou uloženy všechny Transakce, které čekají na to, než budou moci být vyslány na výstup.

Registrové pole obsahuje 1-bitový příznak „potvrzení“ pro každou hodnotu ID. Při příchodu nové Transakce se příznak odpovídající hodnotě jejího ID nastaví na 0. Tento příznak se nastaví zpět na 1, až když dorazí potvrzení s touto hodnotou ID. Transakce na čele FIFOX Multi jsou propagovány na výstup, až když jim odpovídající příznak v registrovém poli má hodnotu 1.

Kritickou částí této komponenty je práce s registrovým polem. Toto pole obsahuje jedno nezávislé zápisové rozhraní pro každý vstup – ať už Transakční nebo potvrzovací – a jedno čtecí rozhraní pro každý výstup. Navíc si musíme uvědomit, že čtecí rozhraní je řízeno (adresováno) hodnotou, která vychází z jednotky FIFOX Multi, a že hodnotou přečtenou na tomto rozhraní je řízen signál pro čtení z tohoto FIFOX Multi. Tím vzniká logická cesta začínající v registru Read Pointer FIFOX Multi a končící opět v tomto registru. Zároveň, jak je ukázáno na obrázku 5.2, samotný výstup FIFOX Multi i signál pro čtení jsou propagovány skrze Barrel Shifter, který přidává další stupně logiky. Při vyšším počtu čtecích rozhraní ve FIFOX Multi také platí, že položka na určité pozic může být z fronty odebrána pouze tehdy, pokud odebrány mohou být i všechny položky na pozicích předcházejících. (Jinak by bylo porušeno pořadí odchozích Transakcí.) Další faktor, který silně ovlivňuje délku této logické cesty, je také počet položek v registrovém poli – tedy počet různých hodnot ID.

Všechny parametry ovlivňující kritičnost logických cest v jednotce Transaction Sorter (počet vstupních Transakcí, počet vstupních potvrzení, počet výstupních Transakcí a počet různých hodnot ID) jsou nastavovány uživatelem této komponenty a je pouze na něm, aby dokázal zredukovat jejich hodnoty natolik, že komponenta bude splňovat požadovanou frekvenci hodinového signálu.

Na ukládání Transakcí by bylo možno využít i jednoduché FIFOX, do kterého bychom vložili vždy slovo obsahující všechny vstupní instrukce společně s příznakem, který by označoval, které jsou skutečně platné a které ne. Obyčejná komponenta FIFOX by ale oproti FIFOX Multi měla důležitou nevýhodu: Obsahovala by totiž i nevalidní Transakce, což by vedlo na méně efektivní využití paměti a menší očekávanou kapacitu v případech, kdy do fronty nevkládáme vždy slova se všemi Transakcemi validními. Vzhledem k tomu, že u jednotky Transaction Sorter očekává, že se bude často zaplňovat, ale neměla by se v ideálním případě nikdy zaplnit zcela – to může platit, pokud potvrzení na Transakce přicházejí v průměru stejně často, jako nové Transakce, byť s velkým zpožděním vůči nim – je schopnost pojmout maximální počet Transakcí bez ohledu na momentální využití jednotlivých vstupních portů významná.

Naštěstí existuje způsob, jakým lze logiku v této jednotce výrazně zjednodušit, a to zavedením používání neunikátních ID pro Transakce. Pokud vložíme do Transaction Sorteru skupinu Transakcí se shodnou hodnotou ID, a následně vložíme potvrzení pro toto ID, pak toto potvrzení bude platné pro všechny tyto Transakce. Jakmile se totiž tyto Transakce objeví na výstupu z vnitřního FIFOX Multi, tak všechny přečtou stejnou hodnotu příznaku potvrzení a u všech bude vyhodnoceno, že mohou být odeslány na výstup. Jediné, co musíme zajistit je, aby na vstup komponenty nepřišla nová Transakce s touto hodnotou ID dříve, než budou všechny tyto potvrzené Transakce odstraněny. Příchod nové Transakce by totiž způsobil opětné vynulování příznaku a tím odstranil informaci o jejich potvrzení. Pokud

ale budeme shodnou hodnotu ID dávat pouze těm transakcím, které jdou přímo po sobě (například vložíme skupinu Transakcí s ID 0 a poté skupinu s ID 1 atd.), pak můžeme snadno detekovat, že z jednotky jsme již odebrali všechny Transakce s určitou hodnotou ID podle toho, že na výstupu se právě nachází Transakce s hodnotou ID, která následovala po ní.

Tímto způsobem lze počet různých hodnot ID zredukovat až na 2, vyjádřené v jediným bitem. Cenou za toto opatření je zvýšená latence potvrzování Transakcí skrze tuto jednotku (první Transakce s určitou hodnotou ID může být potvrzena nejdříve až po vložení poslední Transakce s tímto ID), což současně vede na nutnost zvýšení velikosti vnitřní FIFOX Multi fronty.

5.5 CrossbarX

Základní princip jednotky CrossbarX spočívá v tom, že tato jednotka je připojena na čtecí rozhraní jednoho bufferu a zápisové rozhraní druhého bufferu, přijímá „Transakce“ definující požadované přenosy mezi těmito buffery a vykonává je. Jak zdrojový, tak cílový buffer je přitom rozdělen na řádky, kde každý řádek má své vlastní nezávislé rozhraní.

Jednotka CrossbarX původně vznikla zaobalením několika komponent obsažených v NDP DMA modulu. Došel jsem ale k názoru, že oproti původní implementaci je žádoucí, aby se existující funkcionality zobecnily a některé další i přidaly. Z toho důvodu jsem – po důkladném nastudování implementace předchozích verzí – provedl celý návrh a implementaci od základu.

CrossbarX rozlišuje dvojí označení bufferů, které jsou k němu připojeny.

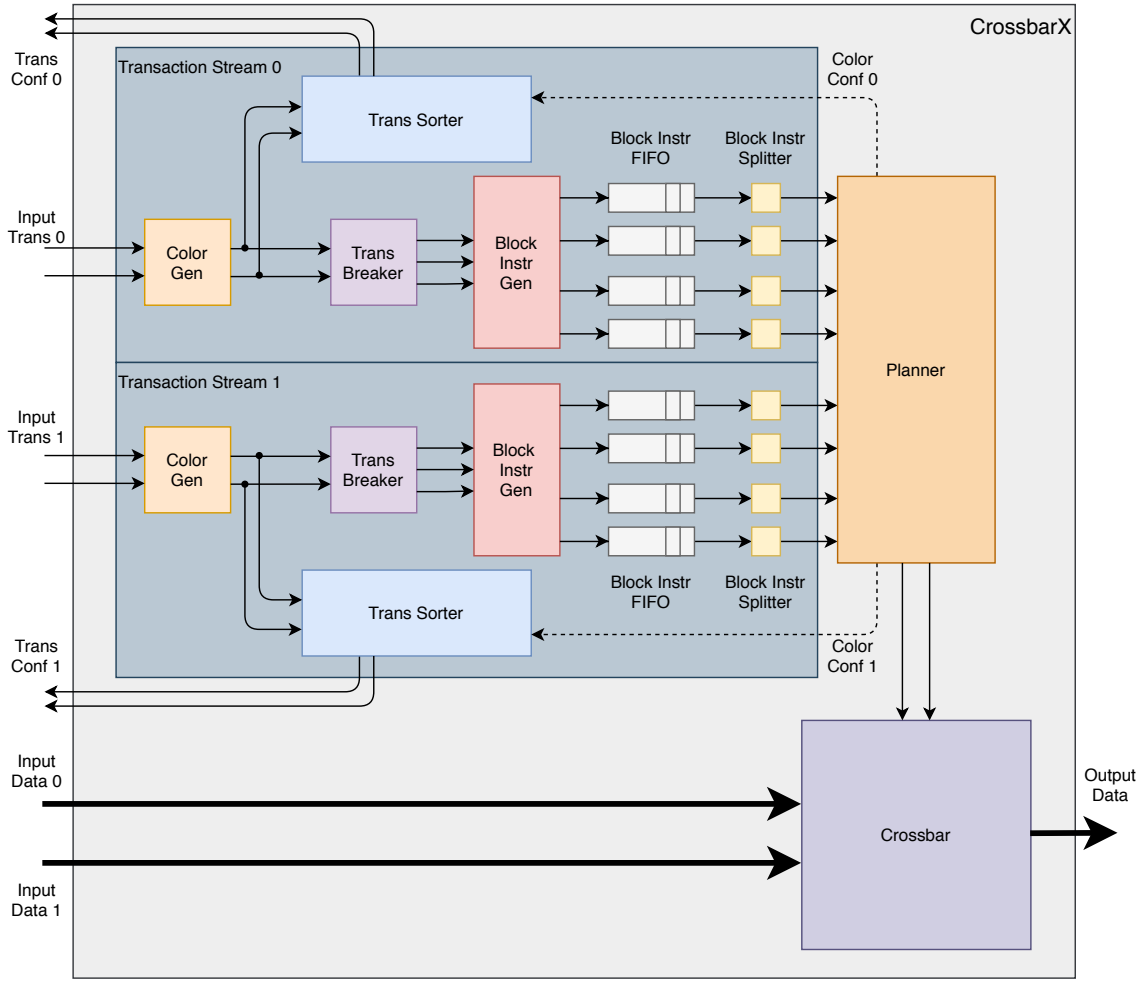
1. Buffery označuje jako „zdrojový“ a „cílový“, podle toho, ze kterého bufferu jsou data čtena a do kterého bufferu jsou zapisována.

2. Buffery označuje jako „Buffer A“ a „Buffer B“, podle toho, pro který buffer jsou kolize řešeny rozdělením Transakcí na jednotlivá slova (Buffer A) a pro který přímou detekcí kolizí a dynamickým plánováním (Buffer B). U jednotky CorssbarX pak říkáme, že plánuje přenosy dat „pro jednotlivá slova v Bufferu A“.

Uživatel může u této jednotky parametrem zvolit, který z připojených bufferů (zdrojový či cílový) má být Buffer A a který Buffer B. Hlavní rozdíl spočívá v tom, že pro Buffer A dovoluje CrossbarX rozdělit řádky do několika skupin, kde každá skupina může být chápána jako nezávislý buffer, který obsahuje data oddělená od dat ostatních skupin. Tento princip je využit právě v případě NPP DMA modulu, kde zdrojový buffer (komponenta Input Buffer v DMA modulu) je nastaven jako Buffer A a díky tomu může existovat v několika nezávislých kopiích, každá v rámci jednoho DMA Endpointu. CrossbarX pak pracuje se všemi těmito Input Buffery současně a chápe je pouze jako skupiny řádků jednoho bufferu.

Díky této vlastnosti může být CrossbarX využit například na slučování dat z několika bufferů do jednoho nebo naopak rozdělování dat z jednoho bufferu do několika výstupních bufferů. To dokáže provádět s téměř stoprocentní propustností a na vysoké datové šířce (která by u jiných řešení znamenala vznik příliš dlouhých kritických cest).

Na obrázku 5.3 je vyobrazeno schéma jednotky CrossbarX v jednom konkrétním nastavení. V tomto nastavení je jako Buffer A určen zdrojový buffer. Pro každou část Bufferu A vstupuje do jednotky jeden nezávislý proud Transakcí (v tomto případě až dvě Transakce v každém cyklu). Jak si lze všimnout, tak velká část komponent uvnitř jednotky CrossbarX



Obrázek 5.3: Schéma komponenty CrossbarX v nastavení se dvěma Transakčními proudy, každý o dvou Transakcích a čtyřech řádcích bufferu. Jako Buffer A je v tomto případě nastaven zdrojový buffer. Přes rozhraní „Input Trans 0“ a „Input Trans 1“ vstupují do jednotky dva nezávislé proudy Transakcí, každá popisující datový rámec v jednom ze zdrojových bufferů a jeho cílové umístnění v cílovém bufferu. Rozhraní „Input Data 0“, „Input Data 1“ a „Output Data“ představují čtecí, respektive zápisové, rozhraní bufferů. Výstupní rozhraní „Trans Conf 0“ a „Trans Conf 1“ obsahují Transakce, pro které byl již dokončen přenos dat. Potvrzení na výstupu Trans Conf 0 odcházejí ve stejném pořadí, jako byly Transakce na vstupu Input Trans 0. To stejné platí i pro Trans Conf 1 a Input Trans 1.

pracuje pouze s Transakcemi v rámci jednoho proudu, takže zvyšování počtu proudů Transakcí nevede v této části k žádnému výraznému zvyšování složitosti logických cest.

Jak již bylo řečeno, pracuje Crossbar uvnitř této jednotky s datovými bloky. Každý tento blok představuje data přečtená z jedné adresy jednoho řádku ve zdrojovém bufferu. V případě NPP DMA modulu je šířka tohoto bloku 8 B a každý z Input Bufferů obsahuje 8 těchto řádků. Transakce předávané do jednotky CrossbarX ale popisují rámce o obecné délce vyjádřené v bajtech. CrossbarX proto musí jednak rozdělit tyto transakce na takzvané „Blokové Instrukce“, které popisují přenos dat o maximální délce jednoho bloku a dále musí zajistit, aby do Crossbaru v jednom cyklu byla předána pouze taková sada Blokových Instrukcí, kde žádné dvě Instrukce nepracují se stejným řádkem ve zdrojovém ani v cílovém bufferu. To by totiž znamenalo kolizi při práci s buffery, kterou jednotka Crossbar neumí vyřešit. Aby bylo dosaženo co nejvyšší propustnosti, je jednotka Crossbar taktována hodinovým signálem o dvojnásobné frekvenci oproti zbytku DMA modulu a v každém hodinovém cyklu „pomalých“ hodin dostává na vstup dvě sady Blokových Instrukcí.

O rozdělení Transakcí na Blokové Instrukce se stará komponenta Block Instruction Generator. Odstranění kolizí mezi Instrukcemi zajišťují komponenty Transaction Breaker a Planner.

Každá příchozí Transakce je v komponentě Transaction Breaker rozdělena na části, kde každá popisuje pouze část rámce, která se nachází uvnitř jednoho slova v Bufferu A. Na vstupu jednotky Transaction Breaker je přijata další Transakce teprve poté, co ta předchozí byla zcela zpracována. Tím dojde u výstupních Transakcí k odstranění kolizí mezi řádky Bufferu A. (Jednotlivé Transakce se totiž nesmějí překrývat a pokud určitá sada Transakcí popisuje data jen v jednom slově, tak nemohou zasahovat do stejných řádků.) Současně tím nedochází ke snížení propustnosti, protože CrossbarX stejně potřebuje pro plnou propustnost zpracovat pouze jedno datové slovo v jednom cyklu. Transaction Breaker obsahuje na výstupu jednu Transakci navíc, neboť v rámci jednoho slova se můžou nacházet části obou Transakcí, které byly právě na vstupu komponenty, a současně i poslední část Transakce, která začínala v jednom z dřívějších slov.

Komponenta Block Instruction Generator následně tyto Transakce rozdělí podle hranic řádků v rámci Bufferu A. Na výstupu této komponenty se proto nachází jedna Bloková Instrukce pro každý řádek Bufferu A náležící do tohoto Transakčního proudu.

Instrukce pro jednotlivé řádky jsou uloženy do oddělených FIFO front. Z těchto front jsou odebírány nerovnoměrně podle toho, jaké se mezi nimi vyskytují kolize řádků v Bufferu B. Tyto fronty ale zajišťují bufferování Instrukcí na jednotlivých řádcích bez ohledu na adresu slova v Bufferu A. Díky tomu dochází k automatickému zvýšení propustnosti v případě, že datový tok obsahuje mezery mezi rámci. V případě mezery se totiž do dané fronty nevloží žádná Instrukce a fronta se tak může začít vyprazdňovat. Dokud nedochází k úplnému zaplnění žádné z těchto front – a tím zastavování příjmu vstupních Transakcí – můžeme tvrdit, že CrossbarX pracuje s propustností sto procent.

Předtím, než mohou být Blokové Instrukce naplánovány do jednotky Crossbar, musí u nich být zajištěno, že žádná z nich nepřesahuje hranici dvou řádků v Bufferu B (pro řádky v Bufferu A to již zajištěno je). K tomu může dojít, pokud přenosy nejsou zarovnány na celé bloky, jako je tomu v případě přenosů v NPP DMA modulu. Přesněji řečeno k tomu dojde, pokud adresa v uvnitř jednoho řádku v Bufferu A se liší od adresy uvnitř řádku v Bufferu B. Tento případ je detekován jednotkou Block Instruction Splitter a Instrukce je v takovém případě rozdělena na dvě části, kde každá pracuje pouze s daty v jednom řádku v Bufferu A i v Bufferu B. Tím dochází ke snížení propustnosti popsanému na obrázku 4.10,

ale dochází k němu pouze a jedině u těch datových bloků, které to skutečně vyžadují. Celý proces rozdělování Transakcí na finální Blokové Instrukce je uveden na obrázku 5.4.

Komponenta Planner má na svém vstupu jednu Blokovou Instrukci pro každý řádek Bufferu A. Jejím úkolem je vybrat z těchto Instrukcí takovou podmnožinu, kde žádné dvě Instrukce nepracují se stejným řádkem Bufferu B (to že nepracují se stejným řádkem Bufferu A je již zde zajištěno). Tato podmnožina tvoří jeden bezkolizní plán, který lze vložit na vstup jednotky Crossbar a ta následně provede přenos dat. Protože Crossbar je taktován na dvojnásovné frekvenci, může Planner v jednom svém cyklu vygenerovat dva bezkolizní plány. Protože ale i sjednocení těchto dvou plánů je pouhou podmnožinou vstupních Instrukcí a těch je maximálně tolik, kolik je je dat v jednom celém slově všech Bufferu A dohromady, nemůže v tomto místě propustnost překročit 100 % a pouze se bude této hodnotě blížit. Jediný důvod, proč může CrossbarX i přesto mít plnou propustnost dat je ten, že mezi daty v bufferech se nacházejí mezery, takže ne v každém cyklu se zapisuje do všech Block Instruction FIFO front a nemusí se tedy ani vždy ze všech číst. Jednotka Planner již není součástí obálky Transaction Stream a pracuje s Instrukcemi ze všech proudů. To je proto, že na řádku v Bufferu B spolu mohou kolidovat i Instrukce z různých Transakčních vstupů.

Jednotka Planner představuje jedno z nejvíce problémových míst z pohledu splnění časování v celém DMA modulu. Je to proto, že pro zjištění kolizí mezi vstupními Instrukcemi se zde musí provádět porovnávání každý s každým. To je operace, která má kvadratickou složitost v závislosti na počtu řádků v bufferech a tedy na celkové datové šířce DMA modulu.

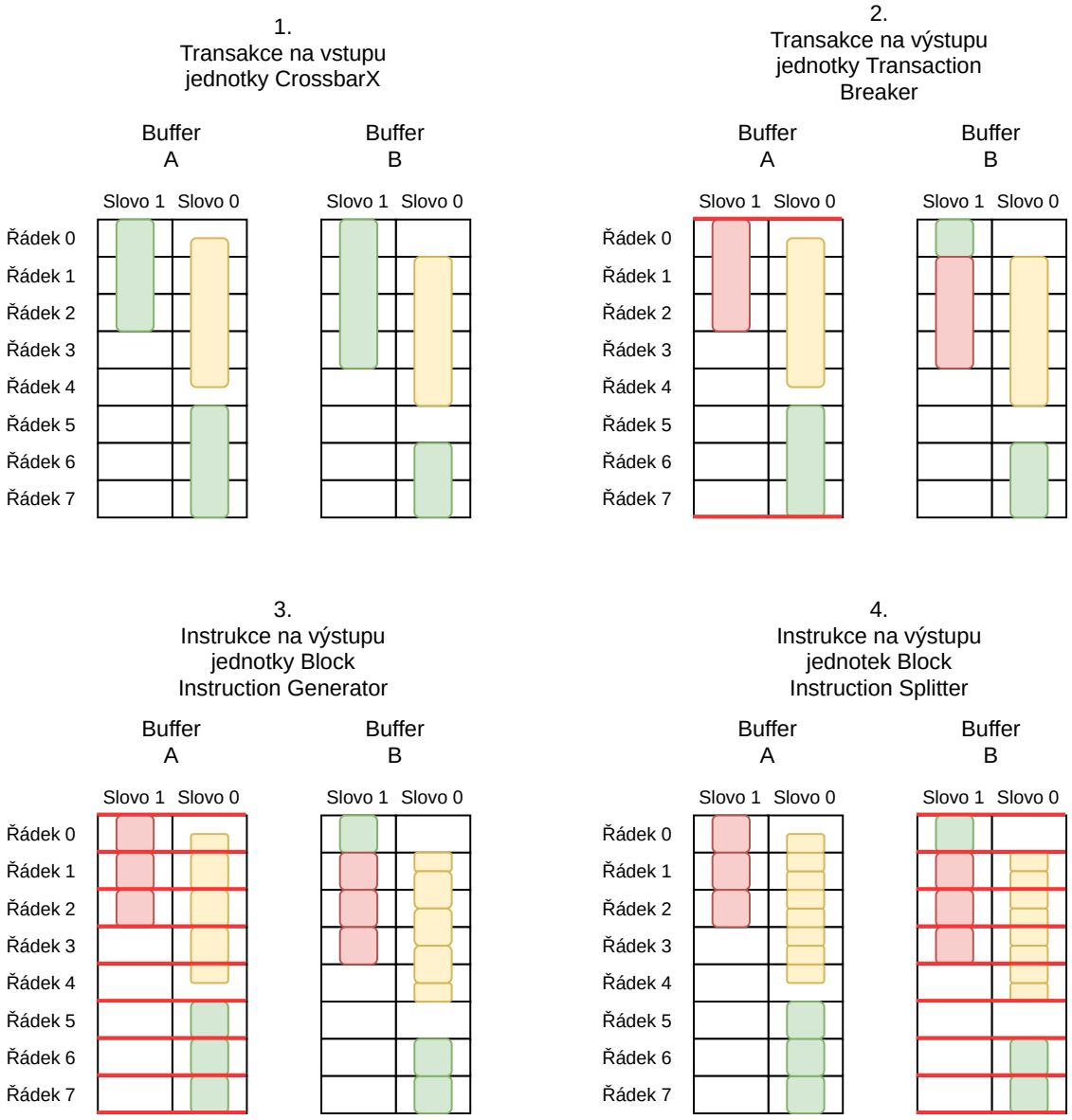
Jak je ukázáno na obrázku 5.3, obsahuje každý Transaction Stream také jednotku Transaction Sorter. Díky tomu, jak Planner vybírá jednotlivé Blokové Instrukce k provedení, probíhá vykonávání Transakcí vstupujících do jednotky CrossbarX out-of-order. Jednotka Transaction Sorter zde slouží k odesílání potvrzení o dokončení přenosu Transakcí a to ve stejném pořadí, jaké měly Transakce na vstupu. Transakční ID v tomto případě není unikátní a je vyjádřeno pouze jedním bitem. Hodnotu tohoto ID přiřazuje každé Transakci jednotka Color Generator umístěná hned na vstupu do jednotky CrossbarX. Jednotka Planner pak odesílá potvrzení o tom, že pro určitý Transakční proud zpracovala celou dávku Instrukcí s určitou hodnotou ID (rozhraní Color Conf na obrázku 5.3).

5.6 Packet Planner

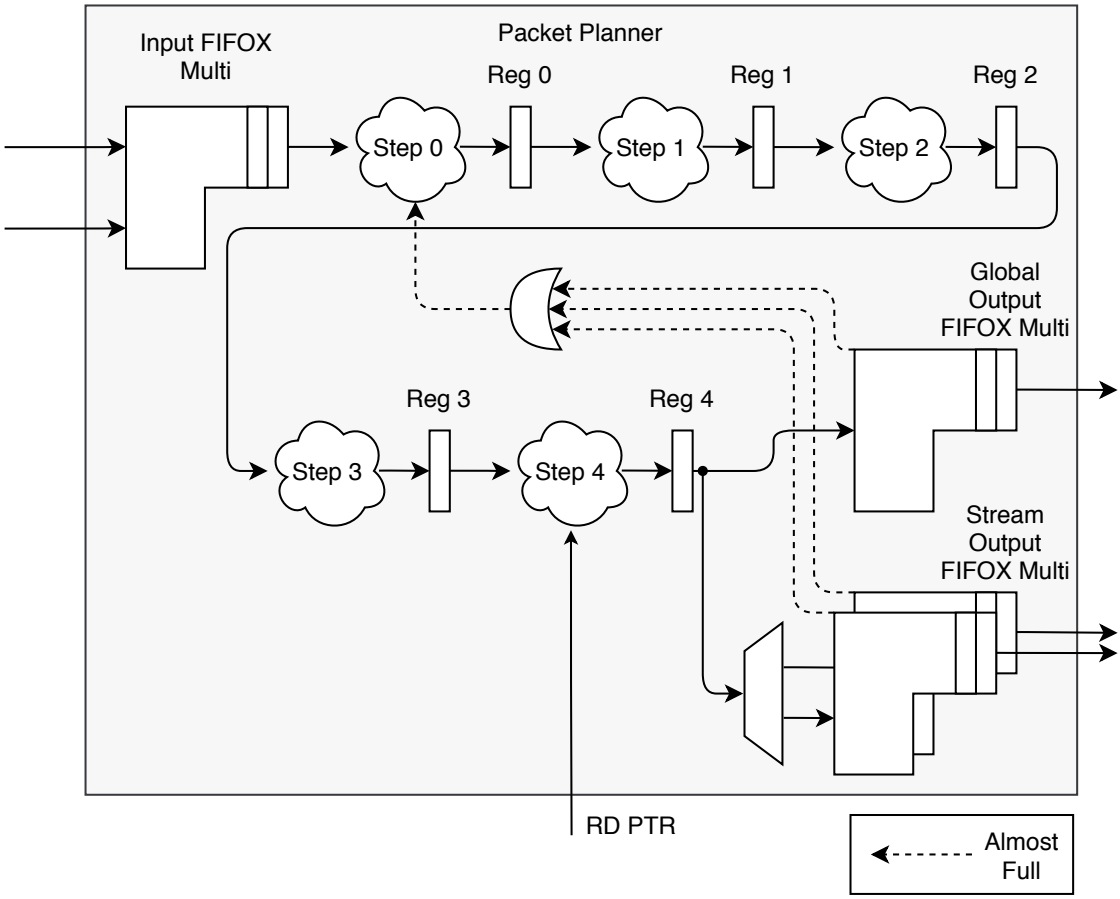
Jednou z komponent v NPP DMA modulu, jejichž funkci nelze rozdělit do jednotlivých DMA Endpointů je komponenta Packet Planner. Účelem této jednotky je určit, na jakých adresách v Output Bufferu budou uloženy jednotlivé odesílané pakety. Tím, že paketům z jednotlivých Endpointů jsou přidělovány postupně vzrůstající adresy do jediného souvislého prostoru (prostoru adres v Output Bufferu) dochází k jejich serializaci a určení jejich globálního pořadí pro výstup z DMA modulu.

Kromě serializace paketů ze všech DMA Endpointů zajišťuje Packet Planner ještě dodržení správných mezi-paketových mezer v Output Bufferu a kontrolu zaplnění Output Bufferu. Aby mohl tuto kontrolu provádět, je do něj z Output Bufferu propagována hodnota aktuálního čtecího ukazatele v tomto bufferu.

Pro určování adres jednotlivých paketů potřebuje Packet Planner pouze informaci o jejich délce. Z praktických důvodu (konkrétně protože v právě této komponentě je určováno globální pořadí paketů) jsou však v DMA modulu skrz tuto komponentu propagovány i informace o DMA kanálu a uživatelské hlavičce, které jsou na výstupu posílány do jednotky Packet Header FIFO.



Obrázek 5.4: Demonstrace zpracování Transakcí uvnitř jedné obálky Transaction Stream v jednotce CrossbarX. Na vstupu se nacházejí dvě Transakce, které popisují určité umístění dat v Bufferu A a Bufferu B zobrazené vlevo nahoře. Jedná se o stejné rozložení jako na obrázku 4.10 s demonstrací nezalovnaného přenosu dat. Jednotka Transaction Breaker rozdělí Transakce podle hranic jednotlivých slov Bufferu A (vpravo nahoře). Následně jsou vzniklé Transakce v jednotce Block Instruction Generator rozděleny na části popisující bloky v Bufferu A (vlevo dole). Jednotky Block Instruction Splitter rozdělí Blokové Instrukce podle hranic bloků v Bufferu B, pokud je to zapotřebí. Ve výsledku jsou dvě vstupní Transakce rozděleny celkem na 14 Blokových Instrukcí. Každá z těchto finálních Blokových Instrukcí pracuje pouze s jedním datovým blokem v Bufferu A a zároveň pouze s jedním datovým blokem v Bufferu B. Tyto Instrukce mohou být následně provedeny tak, jak je uvedeno na obrázku 4.10.



Obrázek 5.5: Schéma komponenty Packet Planner v nastavení se dvěma paketovými proudy (*Streams*). Komponenta obsahuje vstupní FIFOX Multi, pět stupňů zřetězené linky a následně jedno globální výstupní rozhraní a jedno proudové výstupní rozhraní, kde každé je opět opatřeno frontou FIFOX Multi. Vstupní FIFOX Multi je schopno naráz přimout jednu sadu vstupů ze všech proudů současně. Tím dochází automaticky k serializaci příchozích paketů. Pakety následně procházejí jednotlivými kroky, přičemž dojde k přidělení adresy každému paketu. Na výstupním rozhraní je informace o každém paketu vložena do Globálního FIFOX Multi, které obsahuje seřazené pakety ze všech vstupů. Současně je každý paket vložen i do jednoho z Proudových FIFOX Multi a to na základě jeho původního proudu. Každá z těchto FIFOX Multi front proto obsahuje pakety v tom pořadí, v jakém byly vloženy do Packet Planneru na odpovídajícím proudu. V případě, že se některá z těchto front začne blížit zaplnění, aktivuje se její signál Almost Full a ze vstupního FIFOX Multi se přestanou odebírat položky.

Obrázek 5.5 obsahuje nákres schématu komponenty Packet Planner. Protože tato komponenta je v DMA modulu umístěna globálně a její složitost poroste s počtem připojených DMA Endpointů, snažil jsem se při jejím návrhu dosáhnout rozdělení funkcí do co nejmenších částí, aby logické cesty v jednotce byly co nejkratší. Z toho důvodu je rozhodovací logika rozdělena do celkem pěti kroků a jednotlivé kroky jsou odděleny registry, čímž tvoří zřetězenou linku. Kromě těchto kroků obsahuje komponenta především vstupní a výstupní FIFOX Multi fronty. Díky těmto frontám lze parametrizovat počet paketů se kterými se pracuje v jednom hodinovém cyklu a to nezávisle pro část vstupní, pro část vnitřní a pro části výstupní.

Například pokud bychom zjistili, že potřebujeme přijmout na vstupu informaci o až 8 paketech současně, ale vnitřní zřetězená linka nesplňuje pro tak vysoký počet souběžných paketů podmínky časování, potom můžeme parametrem nastavit, aby se ve vnitřní lince pracovalo pouze se 4 pakety současně. Tím umožníme, aby komponenta splnila časování a nezměnila se přitom její funkčnost. Nevýhodou však bude, že jednotka již nebude schopná přijímat plný počet vstupních paketů neustále, což povede ke snížení propustnosti u malých paketů, kdy informace o nových paketech přicházejí nejčastěji.

V některých případech nám to ale vadit nemusí. Můžeme si představit situaci, kdy máme v DMA modulu zapojeny 2 DMA Endpointy a každý je připojen na jeden vstupní proud Packet Planneru. Nicméně výstupní rozhraní DMA modulu ETH MFB je nastaveno tak, že obsahuje pouze jeden Region. Neexistuje proto důvod, abychom na výstup Packet Planneru propagovali v průměru více než jeden paket každý cyklus a proto můžeme i šířku interní linky nastavit na jeden paket za cyklus beze strachu, že by to mohlo negativně ovlivnit propustnost.

Jednotlivé kroky uvnitř zřetězené linky jednotky Packet Planner vykonávají tyto funkce:

0. V tomto kroku se detekuje zaplnění některé z výstupních front pomocí signálu Almost Full. K tomu může dojít, pokud uživatel neodebírá dostatečně rychle data z některého z výstupních rozhraní. V případě, že tato situace nastane, je v tomto kroku zastaveno čtení dalších položek ze vstupní fronty. AFULL Offset je přitom nastaven tak, aby výstupní fronty dokázali i po jeho nastavení akceptovat všechny pakety, které se momentálně mohou nacházet v registrech Reg 0 až Reg 4.

1. Dochází ke spočítání velikosti mezi-paketové mezery, která se bude nacházet za každým validním paketem. Mezera je vypočtena na základě délky paketu a hodnoty „Deficit Idle Count“, která je závislá na zpracování předchozího paketu. Přesný postup je definován ve standartu pro Ethernet [5].

2. Zde je každému paketu přidělena cílová adresa. Pro určení této adresy si zde jednotka udržuje registr se zápisovým ukazatelem do Output Bufferu. Toto místo obsahuje zpětnou smyčku z tohoto registru přes všechny pakety a zpět do tohoto registru, což může znamenat problémy s časováním při plánování pro vysoký počet paketů současně.

3. V tomto kroku je spočítáno celkové množství místa, které dané pakety obsadí v Output Bufferu (od začátku prvního paketů do konce mezery za posledním paketem).

4. Na základě výsledku sumy spočtené v předchozím kroku a aktuálním množství volného místa v Output Bufferu je detekováno, zda se pakety tak, jak byly naplánovány, vejdou do Output Bufferu. Pokud ne, nejsou propuštěny dále a dojde k zastavení zřetězené linky. Aktuální množství volného místa je udržováno v registru. Hodnota

v tomto registru je snížena s každou sadou paketů vloženou do Reg 4 a zvýšena vždy, když se změní hodnota RD PTR propagovaná z Output Bufferu.

Kapitola 6

Verifikace a testování

Pro odladění návrhu a implementace NPP DMA modulu byla využita funkční verifikace implementovaná v jazyce *SystemVerilog*. V rámci této verifikace probíhalo několik desítek testů zaměřených na různé konfigurace a okrajové situace při běhu DMA modulu. Testovací zapojení obsahovalo také komponentu PTC (případně více jejích instancí v závislosti na počtu PCIe Endpointů). Jednotka PTC má sice svojí vlastní verifikaci, ale tímto způsobem bylo možno lépe nasimulovat reálné chování PCIe strany rozhraní k DMA modulu. Zde je uvedeno několik příkladů těchto testů:

- Různý počet PCIe Endpointů.
- Různý počet DMA Endpointů.
- Různá šířka ETH MFB rozhraní. (Díky jednotce CrossbarX by měl DMA modul podporovat různé šířky bez ohledu na počet DMA Endpointů.)
- Různý počet DMA kanálů.
- Různý rozsah velikosti paketů. (Speciální test pro pakety s minimální podporovanou velikostí a s maximální podporovanou velikostí.)
- Různý rozsah velikosti deskriptorů.
- Různý rozsah velikosti deskriptorového bufferu.
- Různá frekvence spouštění a zastavování DMA kanálů.

Tyto testy postupně pomohly odladit problémy v implementaci modulu a odhalily také pár drobných nedostatků v návrhu. Například to, že pokud data popisovaná deskriptorem začínají na adrese, která je zarovnaná na 4 B, ale není zarovnaná na 8 B, a zároveň přesahuje hranici RCB, potom může dojít k problému při zápisu do Input Bufferu. Pokud například budeme mít u PCIe velikost RCB 64 B a budeme číst 12 B na adrese 60 v RAM, může dojít k tomu, že odpověď se nám vrátí jako dvě části: 4 B a 8 B. Tato transakce má ale přiřazenou adresu na začátku 8-bajtového bloku v Input Bufferu. Druhá příchozí část by proto musela být zapsána na rozhraní mezi dvěma řádky v Input Bufferu. To by od jednotky Aligner vyžadovalo dodatečnou rotaci dat a ještě větší zesložitění již tak složité logiky v této komponentě. Tento problém jsem vyřešil tak, že v jednotce Data Reader, která odesílá čtecí požadavek se detekuje, že takový případ může nastat. Nejprve se přečte část dat po hranici RCB a teprve poté se přečte zbytek dat. Každému z těchto čtecích požadavků je přiřazena

oddělená adresa v Input Bufferu zarovnaná na začátek datového bloku. V uvedeném případě by tedy byl nejprve odeslán požadavek na první 4 B a poté na zbylých 8 B. Obě tyto transakce by se uložily v Input Bufferu do zcela oddělených datových bloků a to zarovnaně na 8 B.

Následně po provedení verifikace a souběžně s některými optimalizacemi proběhlo testování DMA modulu v hardware. Konkrétně na jedné z našich již zmíněných vysokorychlostních karet COMBO-200G2QL, která obsahuje FPGA čip Xilinx Ultrascale+. Na této kartě je DMA modul připojen pouze k jednomu rozhraní PCIe Gen3 a proto testuje pouze jeden DMA Endpoint s propustností maximálně 100 Gb/s. Tyto testy byli současně také prvními testy pro softwarový ovladač NPP, který byl na našem oddělená implementován současně se samotným DMA modulem.

Kapitola 7

Měření výsledků

Po funkčním odladění DMA modulu jsem přešel k měření důležitých charakteristik výsledného modulu. Jednalo se především o spotřebu zdrojů na FPGA čipu v závislosti na počtu DMA kanálů a o datovou propustnost při odesílání paketů. Propustnost jsem měřil na stejné kartě, kterou jsem použil pro testy funkčnosti. Spotřebu zdrojů jsem zjišťoval jak pro čip Xilinx Ultrascale+, tak také pro Intel Stratix10. Tuto spotřebu jsem v obou případech porovnal se spotřebou starší verze DMA modulu pro přenosy NDP, která se na našich kartách používá v současnosti.

7.1 Spotřeba zdrojů

Množství zdrojů spotřebovaných DMA modulem na FPGA čipu je významné ze dvou hledisek:

1. Z hlediska jejich růstu v závislosti na počtu DMA kanálů, který se u DMA modulu nastaví aby podporoval.
2. Z hlediska porovnání tohoto růstu s předchozí verzí modulu.

Tabulky 7.1 a 7.2 obsahují konkrétní hodnoty spotřebovaných zdrojů NPP TX DMA modulu na zmíněných FPGA čipech Stratix10 a Ultrascale+. V obou případěch je uvedena spotřeba LUT (*Look-Up Table*), které byly použity pro kombinační logiku a těch, které byly použity jako distribuovaná paměť. Dále spotřebu jednotek implementujících blokovou paměť a DSP *Digital Signal Processing*. Blokové paměti využívá DMA modul pro implementaci Input Bufferu a Output Bufferu. DSP bloky jsou využity v jednotce DMA Controller pro implementaci statistických čítačů počtu odeslaných paketů a odeslaných bajtů.

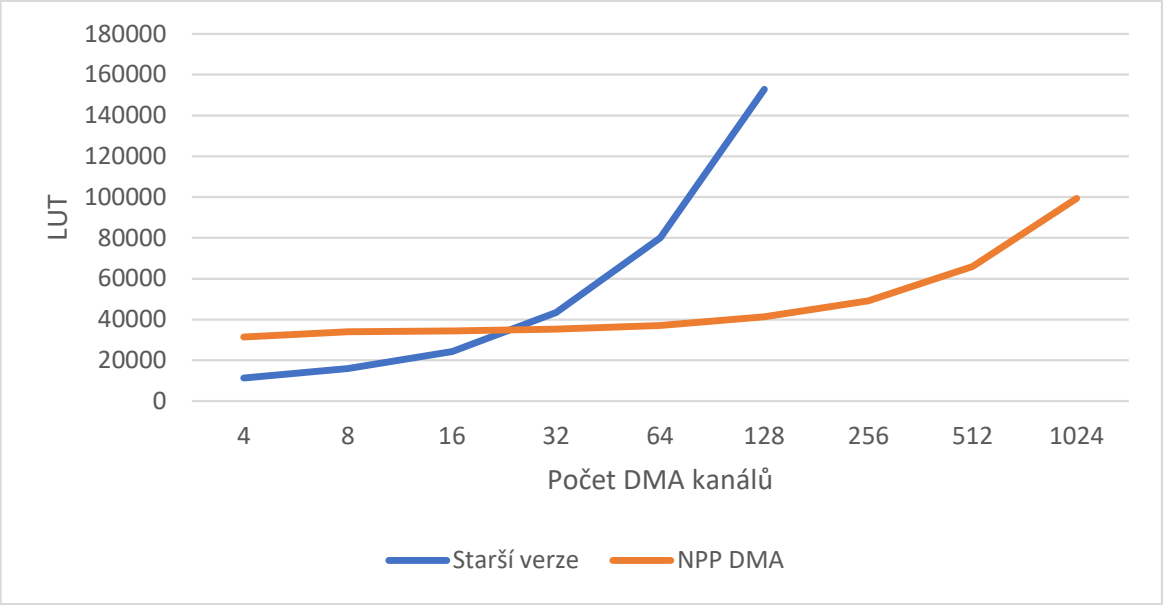
Jak v případě tabulky pro Stratix10, tak v případě tabulky pro Ultrascale+ můžeme pozorovat stejné chování nárůstu hodnot u jednotlivých sloupců. Počet spotřebovaných blokových pamětí není nijak závislý na počtu DMA kanálů a je proto konstantní. Tento počet lze měnit parametrem DMA modulu pro hloubku Input Bufferu a Output Bufferu. Naopak počet DSP bloků roste podle očekávání lineárně s počtem kanálů. Pro každý DMA kanál jsou totiž potřeba přesně 2 DSP bloky pro statistické čítače. To vede k tomu, že pro 1024 DMA kanálů spotřebuje DMA modul přibližně polovinu dostupných DSP bloků na čipu (Stratix10 i Ultrascale+). Spotřebu DSP lze pro vyšší počty DMA kanálů omezit tím, že některé DMA kanály budou svoje statistické čítače uchovávat v registrech a jejich inkrementace bude realizována přímo v logice FPGA pomocí LUT.

Počet DMA kanálů	l-ALUT (z 93 312 dostupných)	m-ALUT (z 93 312 dostupných)	M20K (z 11 721 dostupných)	DSP (z 5 760 dostupných)
4	44 579	9 026	1 977	8
8	47 130	9 555	1 977	16
16	47 781	9 560	1 977	32
32	49 276	9 565	1 977	64
64	51 563	10 151	1 977	128
128	54 940	11 416	1 977	256
256	63 022	13 941	1 977	512
512	79 273	18 986	1 977	1 024
1024	109 757	29 071	1 977	2 048

Tabulka 7.1: Spotřeba zdrojů NPP TX DMA modulu v závislosti na počtu DMA kanálů na FPGA čipu Stratix10. Spotřebované zdroje jsou uvedeny pro tyto jednotky: l-ALUT – Logické bloky použité pro kombinační logiku. m-ALUT – Logické bloky použité jako paměť. M20K – Bloková paměť. DSP – *Digital Signal Processing* jednotky.

Počet DMA kanálů	l-LUT (z 788 160 dostupných)	m-LUT (z 394 560 dostupných)	BRAM (z 1 440 dostupných)	DSP (z 4 560 dostupných)
4	27 448	3 962	1 054	8
8	29 700	4 286	1 054	16
16	30 150	4 337	1 054	32
32	30 928	4 348	1 054	64
64	32 448	4 660	1 054	128
128	36 032	5 237	1 054	256
256	42 549	6 612	1 054	512
512	55 509	10 495	1 054	1 024
1024	81 866	17 507	1 054	2 048

Tabulka 7.2: Spotřeba zdrojů NPP TX DMA modulu v závislosti na počtu DMA kanálů na FPGA čipu Ultrascale+. Spotřebované zdroje jsou uvedeny pro tyto jednotky: l-LUT – Logické bloky použité pro kombinační logiku. m-LUT – Logické bloky použité jako paměť. BRAM – Bloková paměť. DSP – *Digital Signal Processing* jednotky.



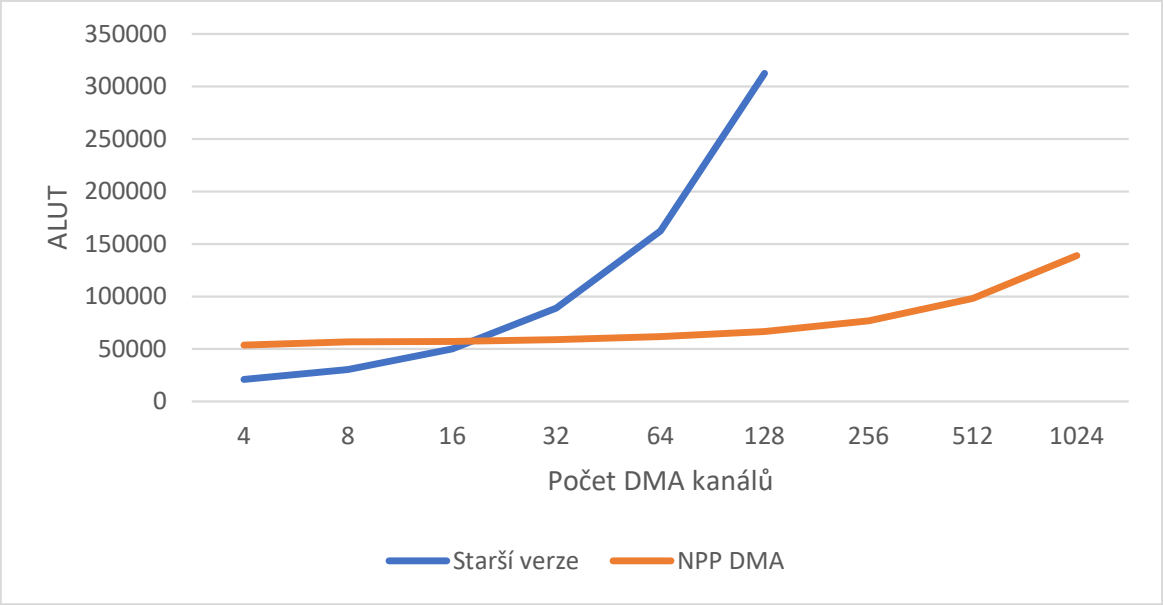
Obrázek 7.1: Graf porovnání spotřeby LUT na čipu Ultrascale+ mezi starší verzí NDP DMA modulu a novým NPP DMA modulem. U obou verzí je patrný lineární růst vzhledem k počtu DMA kanálů (na grafu exponenciální růst kvůli logaritmickému měřítku vodorovné osy). Hodnoty pro starší verzi jsou zobrazeny pouze po 128 DMA kanálů, neboť pro vyšší počet se již syntetizovaný modul nevejde na cílový čip. Díky použití metody Vybraných DMA kanálů je ale u nového modulu pouze minimum komponent skutečně závislých na celkovém počtu DMA kanálů a nárůst se tak začíná výrazněji projevovat až od 128 DMA kanálů výš. Současně je vidět, že pro počet kanálů 16 a méně, má NPP DMA modul spotřebu vyšší než ten starý. Důvodem je to, že NPP modul implementuje některé funkcionality, které starší modul nepodporuje (například podporu připojení na více DMA Endpointů), a také je navržen na přenosy s vyšší propustností a proto je komplexnější. Navíc podporuje paketové přenosy NPP, což v porovnání s proudovými přenosy NDP vyžaduje složitější zapojení.

U počtu spotřebovaných LUT je pro prvních několik řádků hodnota téměř konstantní. Teprve od 128 DMA kanálů začínají převládat svojí spotřebou ty jednotky, které rostou v závislosti na počtu kanálů a růst začíná být lineární. Například verze DMA modulu na čipu Ultrascale+, která podporuje 1024 DMA kanálů, zabírá 81 866 logických LUT jednotek. Téměř polovinu z toho přitom činí komponenta Software Manager, která obsahuje MI32 registry pro každý DMA kanál zvlášť. I tak se ale celková spotřeba LUT pro 1024 DMA kanálů pohybuje u obou čipu jenom kolem pouhých 11 %.

Obrázky 7.1 a 7.2 obsahují grafy porovnávající spotřebu zdrojů mezi novým NPP DMA modulem a aktuálně používanou verzí NDP DMA modulu. Na těchto grafech je porovnávána pouze spotřeba LUT, která je v tomto případě nejvíce relevantní.

7.2 Datová Propustnost

Stejně jako funkčnost i propustnost jsem měřil na síťové kartě COMBO-200G2QL, která, při zapojení pouze jednoho DMA moulu k jednomu PCIe Gen3 portu, podporuje maximální dosažitelnou propustnost 100 Gb/s. Propustnost byla měřena pro jednotlivé paketové délky

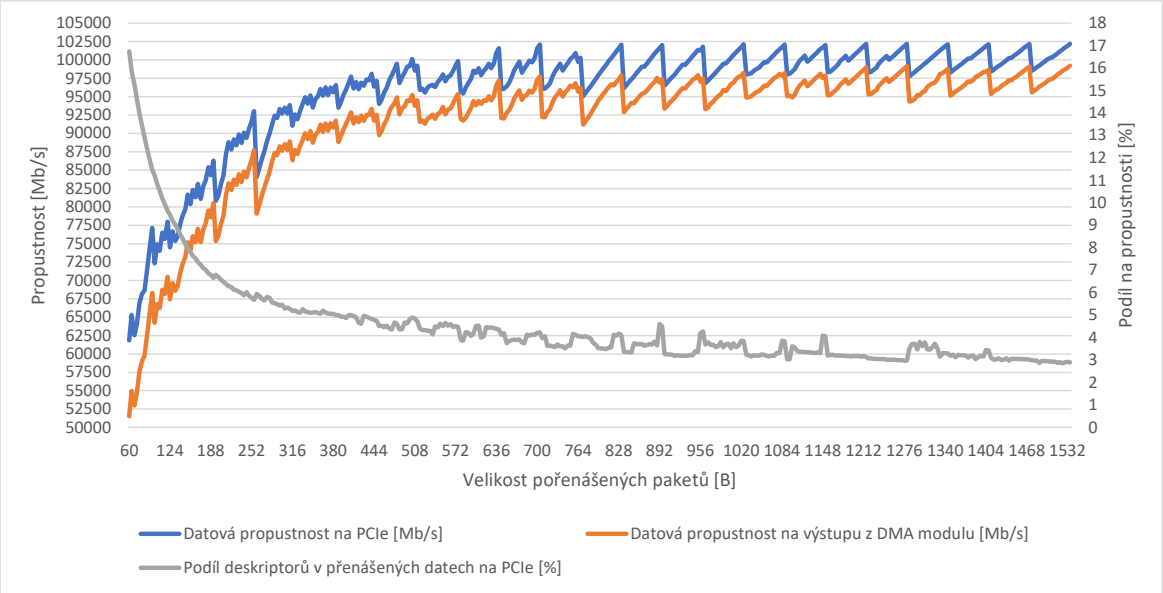


Obrázek 7.2: Graf porovnání spotřeby ALUT na čipu Stratix10 mezi starší verzí NDP DMA modulu a novým NPP DMA modulem. Graf vykazuje podobný výsledek porovnání jako graf na obrázku 7.1. Tento graf je ale méně vypovídající, neboť starší verze DMA modulu nebyla nikdy určena pro čipy od firmy Intel a proto ani není nijak optimalizována pro tuto technologii. Z toho důvodu vychází v tomto případě v porovnání s ním NPP DMA modul o něco lépe než na grafu pro Ultrascale+.

od 60 B do 1536 B s krokem 4 B. Hodnota 60 B byla zvolena, protože minimální velikost rámce na Ethernetu je 64 B včetně CRC (*Cyclic Redundancy Check*) kódu, ale protože CRC se na síťové kartě vkládá do paketů až za TX DMA modulem, je minimální očekávaná velikost paketů přenášených DMA modulem právě 60 B. Horní hranice 1536 B je zvolena, protože se jedná o trojnásobek hodnoty MRRS (která je 512 B). Při měření byla data odesílána na 16 DMA kanálech současně. Příliš malý počet DMA kanálů by totiž mohl způsobit, že pro krátké pakety bude propustnost omezována CPU. Současně se tím otestovalo, jak dobře dokáže DMA modul zatěžovat sběrnici PCIe, když musí dynamicky přepínat mezi Vybranými DMA kanály, kterých bylo v testovaném DMA modulu nastaveno 8. Aby byla dosažená propustnost co nejvyšší, byl každý odesílaný paket uložen v souvislém prostoru v paměti popsaném jedním deskriptorem a měl adresu zarovnanou na 64 B (jinou variantu v době měření ani softwarový ovladač ještě nepodporoval).

Aby byly výsledky měření co nejpřesnější, použil jsem firmwarovou komponentu, která dokáže změřit aktuální propustnost na MFB rozhraní uvnitř FPGA. Výsledky jejího měření jsou následně dostupné přes MI32 rozhraní a lze je tedy získat přímo v příkazové řádce v průběhu odesílání paketů. Tuto komponentu jsem v testovaném modulu zapojil na tři různá místa:

1. Na vstupní DOWN MFB rozhraní DMA modulu.
2. Na vstupní MFB rozhraní komponenty Aligner. (zde obsahuje MFB již pouze rámce s daty paketů a nikoliv rámce s deskriptory.)
3. Na výstupní ETH MFB rozhraní DMA modulu. (Toto rozhraní obsahuje přímo jednotlivé pakety. Propustnost naměřená zde musí být logicky v průměru stejná jako



Obrázek 7.3: Graf naměřené propustnosti TX NPP DMA modulu na síťové kartě COMBO-200G2QL. Graf ukazuje propustnost v gigabitech za sekundu naměřenou na vstupním MFB rozhraní do DMA modulu z jednotky PTC. (V tomto místě obsahuje datový tok i transakce s deskriptory.) Dále také propustnost dat naměřenou na výstupu z DMA modulu. (Zde se již jedná přímo o data paketů.) Graf také ukazuje naměřený relativní podíl deskriptorů v transakcích přicházejících do DMA modulu (hodnoty na pravé svislé ose).

propustnost naměřená na vstupu do jednotky Aligner, neboť DMA modul do proudu paketů žádná data nepřidává ani z něj žádná neodebírá. Jedná se proto spíš o kontrolu, že všechna data opravdu vystupují z DMA modulu.)

Pro každou paketovou délku jsem – pro vyšší přesnost – provedl 20 měření a jako výsledek jsem zaznamenal propustnost na vstupu do DMA modulu, propustnost na výstupu z DMA modulu a dále procentuální pokles propustnosti mezi vstupem do DMA modulu a vstupem do jednotky Aligner, který byl způsobený odebráním deskriptorů z MFB proudu. Výsledky tohoto měření jsou vyobrazeny na obrázku 7.3. Jak tento graf ukazuje, vliv přítomnosti deskriptorů v datech na propustnost klesá se vzrůstající délkou paketů. To je očekávané chování, neboť pro větší pakety je potřeba přenést méně bajtů deskriptorů v přepočtu na bajty paketů. Největší je tento vliv pro pakety o délce 60 B, kde činí 16,74 %. Nejméně je to pro délku paketů 1536 B, kde je neměřený vliv 2,90 %.

U hodnot naměřené propustnosti lze pozorovat tyto dva nejvýraznější jevy:

1. Propustnost postupně roste a přibližuje se horní hranici 102,4 Gb/s. (To je reálná maximální propustnost MFB sběrnice o šířce 512 b taktované na frekvenci 200 MHz a také maximální propustnost použité sběrnice PCIe.)

2. Propustnost vykazuje výrazný periodický propad a to s periodou 64 B.

První bod souvisí částečně s poklesem procentuálního podílu deskriptorů v datech čtených z paměti. Částečně je to způsobeno tím, že čím větší pakety přenášíme, tím větší část čtecích požadavků má velikost MRRS. Tím se snižuje režie přenosu po sběrnici PCIe a stoupá propustnost. K periodickému poklesu propustnosti dochází u každé délky, která následuje po násobku 64 B (68 B, 132 B, 196 B atd.). Tento pokles je přirozený pro datové

přenosy na sběrnici PCIe a souvisí s hodnotou RCB, která je v tomto případě právě 64 B. Další, méně nápadný periodický pokles propustnosti je zde vidět pro periody 512 B. Tento pokles je způsoben hranicí 512 B MRRS. Pokud například odesíláme pakety s délkou 516 B, pak pro každý paket musí DMA modul vygenerovat dva čtecí požadavky, což znamená dvojnásobnou režii na sběrnici PCIe, oproti paketům s délkou 512 B, kde stačí vygenerovat pouze jeden požadavek.

Pro pakety s velikostí 60 B dosáhl DMA modul na PCIe propustnosti 61,8 Gb/s. Vlivem vysokého podílu deskriptorů to ale znamená pouhých 51,5 Gb/s pro přenos paketů, což byla rovněž vůbec nejnižší naměřená hodnota. Nejvyšší propustnost byla naopak naměřena pro pakety s délkou 1280 B a to 102,2 Gb/s s deskriptory a 99,1 Gb/s bez deskriptorů. Později jsem zkoušel měřit rychlost i pro větší délky paketů (až do délky 4096 B). Vývoj propustnosti zde byl přibližně stejný jako, který lze pozorovat na obrázku 7.3 pro délky od 1300 B výš. Nejvyšší propustnost jsem v tomto rozsahu naměřilo pro délku paketů 3776 B. Bylo to 102,2 Gb/s na PCIe a 99,9 Gb/s na výstupu z DMA modulu. Podíl deskriptorů na propustnosti činil pro tuto paketovou délku 2,25 %. Během tohoto měření neklesla propustnost na PCIe pod hodnotu 95 Gb/s pro žádnou paketovou délku od 460 B výš.

Celkově lze z těchto měření odvodit, že nový DMA modul úspěšně zvládá přenosy přes PCIe Gen3 na požadované rychlosti 100 Gb/s v případech, které jsou příznivé pro paketové DMA přenosy. K tomu, abych mohl otestovat propustnost i pro případy, které naopak budou pro paketové přenosy nepříznivé, jsem neměl v době měření ještě dostatečnou podporu v NPP ovladači. To jsou například případy, kdy budou pakety rozděleny do více deskriptorů a kdy budou adresy jednotlivých částí paketů v RAM nezarovnané. Dalším krokem bude také otestování propustnosti pro vyšší rychlosti 200 Gb/s a 400 Gb/s. Pro rychlost 400 Gb/s nemám v současnosti připravený dostupný hardware. Rychlost 200 Gb/s mám v plánu otestovat na kartě COMBO-200G2QL, kde však bude potřeba pro tuto variantu udělat rozsáhlejší úpravy firmware.

Kapitola 8

Závěr

Cílem této práce bylo navržení, implementace a otestování firmwarového modulu pro čip FPGA zajišťujícího vysokorychlostní přenosy síťových paketů z paměti počítače do síťové karty a to přes sběrnici PCIe. Cílem tohoto modulu je podporovat přenosovou rychlost pohybující se v řádu stovek gigabitů za sekundu.

Součástí práce bylo nastudování možných přístupů k řízení těchto přenosů a zhodnocení jejich kladů a záporů z pohledu dosažitelné propustnosti, složitosti návrhu a použitelnosti v praxi. Na základě tohoto zhodnocení byl navržen systém NPP pro řízení paketového přenosu dat, který tento nový DMA modul implementuje. Cílem systému NPP je dosažení maximální flexibility a použitelnosti v software (zejména pokud jde o možnost paralelizace zpracování paketů) a zároveň optimalizace komunikačního protokolu mezi software a hardware za účelem snížení režie přenosů a tím dosažení maximální propustnosti na sběrnici PCIe. Protože pro systém NPP byl implementován i softwarový ovladač síťové karty a protože byl tento systém navržen tak, aby byl flexibilní z pohledu software, je možné jej používat pro přenosy různými již existujícími systémy (jako například DPDK) pouze s potřebou dodání podpory v ovladači. Firmwarový DMA modul se v takovém případě nemusí modifikovat.

Kromě samotného přenosu síťových paketů podporuje tento NPP DMA modul také nastavitelný počet DMA kanálů, které umožňují distribuci zátěže při vysokorychlostním odesílání na více procesorových jader a rychlé odesílání dat z virtuálních strojů s minimální režií způsobenou virtualizací. Pro výkonné FPGA čipy jako jsou Intel Stratix10 a Xilinx Ultrascale+ může podle výsledků naměřených v této práci NPP DMA modul podporovat přenos až na 1024 DMA kanálech při současné spotřebě logických zdrojů pohybující se okolo pouhých 11 %. To je výrazné zlepšení oproti předchozím verzím DMA modulu, které mohly z důvodu vysoké spotřeby zdrojů podporovat nejvýše 64 DMA kanálů.

Součástí práce na DMA modulu bylo i vytvoření komponenty, která jej umožňuje propojit s PCIe rozhraním různých FPGA čipů a to jak od firmy Xilinx tak od firmy Intel. Tato komponenta umožňuje relativně snadno přidávat podporu pro nejrůznější modely FPGA čipů a nejrůznější konfigurace PCIe rozhraní bez nutnosti zasahování do architektury samotného DMA modulu.

Další vlastností navrženého DMA modulu je schopnost řídit přenosy přes několik rozhraní PCIe Generace 3 anebo Generace 4 zároveň, při současném slučování paketů ze všech těchto rozhraní do jednoho výstupního proudu dat. Tato vlastnost umožňuje teoreticky DMA modulu přenést do výstupního ethernetového rozhraní FPGA, které podporuje rychlost 400 Gb/s, provoz ze čtyř rozhraní PCIe Generace 3 nebo ze dvou rozhraní PCIe Generace 4. Z pohledu architektury také nic nebrání tomu, aby byl tento stejný DMA modul

použit i pro přenosovou rychlost vyšší než je 400 Gb/s za předpokladu, že jej cílový FPGA čip dokáže v takovém nastavení provozovat na požadované frekvenci 200 MHz.

V rámci této práce byly otestovány přenosy paketů do FPGA po jedné sběrnici PCIe Generace 3 x16 s maximální dosažitelnou propustností 102,4 Gb/s. Během těchto měření bylo ověřeno, že navržený DMA modul dokáže při odesílání paketů s vhodnou délkou skutečně využít na připojené sběrnici propustnost přes 100 Gb/s a díky nízké režii u systému NPP přitom dosáhne propustnosti síťových paketů přesahující 99,9 Gb/s. Při přenášení paketů s délkou 460 B a více dokázal během měření DMA modul dosáhnout na sběrnici PCIe vždy propustnosti nad 95 Gb/s. Bude-li do budoucna dostuná platforma s FPGA, která umožní odladit ovládání sběrnice PCIe i pro generace 4 a 5, měl by tento modul skutečně dovolovat přenášet pakety z RAM do FPGA rychlostí blížící se až 400 Gb/s. To je také zamýšlený směr pro tuto práci do budoucna.

Literatura

[1] *Avalon Interface Specifications*. [cit. 2020-2-16]. Dostupné z: https://www.xilinx.com/support/documentation/user_guides/ug576-ultrascale-gth-transceivers.pdf.

[2] *AXI Reference Guide*. [cit. 2020-2-16]. Dostupné z: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf.

[3] *XDP - eXpress Data Pathlabs*. Jesper Dangaard Brouer [cit. 2020-2-23]. Dostupné z: https://prototype-kernel.readthedocs.io/en/latest/networking/XDP/index.html.

[4] *EBPF - extended Berkeley Packet Filter* [online]. The kernel development community, 2016 [cit. 2020-03-11]. Dostupné z: https://prototype-kernel.readthedocs.io/en/latest/bpf/.

[5] IEEE Standard Ethernet. *IEEE Std. 8023-2018*. 2018.

[6] *Cards* [online]. Liberouter CESNET, 2020 [cit. 2020-04-18]. Dostupné z: https://www.liberouter.org/technologies/cards/.

[7] *ConnectX® Ethernet Adapters* [online]. Mellanox Technologies, 2020 [cit. 2020-04-18]. Dostupné z: https://www.mellanox.com/products/ethernet/connectx-smartnic.

[8] *Documentation* [online]. DPDK Project, 2020 [cit. 2020-02-23]. Dostupné z: http://core.dpdk.org/doc/.

[9] *Specifications* [online]. PCI-SIG, 2020 [cit. 2020-02-23]. Dostupné z: https://pcisig.com/specifications.

[10] J., A. *IRQ, DMA a I/O Předcházení a řešení konfliktů v konfiguraci počítače*. 1. vyd. Praha: Computer Press, 2000. ISBN 80-7226-264-5.

[11] Steven McCanne, V. J. The BSD Packet Filter: A New Architecture for User-level Packet Capture. *Winter USENIX conference*. 1992, January 25 1993.