VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

MĚŘENÍ PARAMETRŮ KOMUNIKACE PŘES PCI EXPRESS

MEASURING PARAMETERS OF COMMUNICATION OVER PCI EXPRESS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN MATĚJKA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JIŘÍ MATOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Matějka Martin**
Program: Informační technologie
Název: **Měření parametrů komunikace přes PCI Express**
Measuring Parameters of Communication over PCI Express
Kategorie: Návrh číslicových systémů

Zadání:

1. Nastudujte specifikaci PCI Express verze 3.0. Zaměřte se především na parametry ovlivňující maximální propustnost.
2. Navrhněte jednotku pro měření parametrů komunikace přes PCI Express, která bude určena pro čip FPGA na akcelerační kartě s PCI Express rozhraním.
3. Navrhněte uživatelskou aplikaci pro práci s navrženou jednotkou v systému Linux.
4. Navrženou jednotku implementujte v jazyce VHDL; navržený software implementujte v jazyce C.
5. S využitím implementované jednotky a vytvořeného software proveďte měření parametrů komunikace přes PCI Express.
6. Zhodnoťte naměřené výsledky a diskutujte možnosti dalších rozšíření.

Literatura:

- *PCI Express Base Specification Revision 3.0*. PCI-SIG, November 10, 2010.
- Jason Lawley: *Understanding Performance of PCI Express Systems*. Xilinx White Paper WP350 (v1.2), October 28, 2014.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

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

Vedoucí práce: **Matoušek Jiří, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 29. října 2021

Abstrakt

Táto bakalárska práca sa zaoberá vytvorením jednotky, ktorá je schopná merať parametre komunikácie na zbernicovom štandarde PCI Express 3.0. Navrhnutá jednotka je určená pre čip FPGA umiestnený na akceleračnej karte s PCI Express rozhraním. Zároveň opisuje softvér implementovaný v jazyku C, určený na jej ovládanie. V poslednej časti tejto práce sú ukázané experimenty s navrhnutou a implementovanou jednotkou.

Abstract

This bachelor thesis describes designing a unit capable of measuring throughput on a PCI Express bus. The described unit is designed for an FPGA chip on an acceleration card with PCI Express 3.0 interface. At the same time, the thesis describes software implemented in C language used for its control. In the last section of this bachelor thesis are experiments conducted with the designed and implemented unit.

Kľúčové slová

PCI Express 3.0, FPGA, meranie priepustnosti, zbernica, maximum payload size, maximum read request size, relaxed ordering

Keywords

PCI Express 3.0, VHDL, throughput measurement, bus, maximum payload size, maximum read request size, relaxed ordering

Citácia

MATĚJKA, Martin. *Měření parametrů komunikace přes PCI Express*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Matoušek, Ph.D.

Měření parametrů komunikace přes PCI Express

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jiřího Matouška, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Martin Matějka
17. mája 2022

Poďakovanie

Veľké poďakovanie patrí vedúcemu práce, pánovi Ing. Jiřímu Matouškovi, Ph.D., za cenné rady, pripomienky a pomoc, ktoré prácu posunuli dopredu. Čas, ktorý práci venoval si veľmi vážim. Rád by som sa ešte menovite poďakoval pánovi Ing. Martinovi Špinlerovi zo združenia CESNET, ktorý práci poskytol potrebný ovládač a v neposlednom rade mojej rodine, ktorá ma veľmi podporovala.

Obsah

Kapitola 1

Úvod

Zavedenie PCI (Peripheral Component Interconnect) zbernice – na začiatku deväťdesiatych rokov minulého storočia – spoločnosťou Intel prinieslo zjednocujúci účinok. PCI zbernica začala postupne nahradzovať existujúce zbernice ako boli ISA (Industry Standard Architecture), VESA (Video Electronics Standards Association) alebo EISA (Extended Industry Standard Architecture). Táto zbernica ponúkla veľa nových vylepšení oproti existujúcim riešeniam, vrátane nezávislosti procesora, izoláciu vyrovnávacej pamäte, ovládanie zbernice a pravé plug-and-play operácie. Izolácia vyrovnávacej pamäte v zásade oddelila ako elektricky, tak aj po stránke domén hodinového signálu lokálnu zbernicu procesora od PCI zbernice. Oddelenie zlepšilo výkon umožnením behu súbežných cyklov na takto izolovaných zberniciach PCI a procesora. Výhodou bola aj možnosť zvýšenia frekvencie na zbernici procesora, a to nezávislo od rýchlosti PCI zbernice. [9]

Postupom času však dopyt po rýchlejších riešeniach a nových funkciách potrebných pre nové generácie I/O zariadení spôsobil, že PCI zbernica a jej deriváty ako napríklad PCI-X, ktoré zväčša priniesli zlepšenie len v zvýšení frekvencií, boli nahradené novým zbernicovým štandardom PCI Express [9]. Ten – na rozdiel od pôvodného paralelného štandardu – prenášané dáta serializuje, čo umožnilo zvýšiť taktovaciu frekvenciu bez parazitných javov, vyskytujúcich sa v paralelných prenosoch, a v konečnom dôsledku zvýšiť priepustnosť zbernice.

Táto práca by mala priniesť finančne dostupnejšie riešenie pre meranie rýchlosti PCIe zbernice, implementované na pomerne lacnej akceleračnej karte s FPGA čipom, ktorá je pripojená k PCIe rozhraniu. Pri existujúcich riešeniach ako napríklad prístroje (analyzátory protokolu) od spoločnosti Teledyne Technologies [1], sa cena pohybuje v rádoch státisícov českých korún, čo má na dostupnosť negatívny vplyv.

V mojej práci sa v prvých dvoch kapitolách (2 a 3) zaoberám práve treťou generáciou štandardu PCI Express (ďalej PCIe) a parametrami, ktoré ovplyvňujú jeho priepustnosť. Ďalej v práci v kapitole 5 opisujem navrhnutú jednotku pre meranie parametrov komunikácie cez túto zbernicu, ktorá je implementovaná na akceleračnej karte s FPGA (Field-Programmable Gate Array) čipom, obsahujúcej PCI Express rozhranie. Rovnako tak popíšem software implementovaný v jazyku C, ktorý komunikuje s akceleračnou kartou a umožňuje merať priepustnosť zbernice.

Kapitola 2

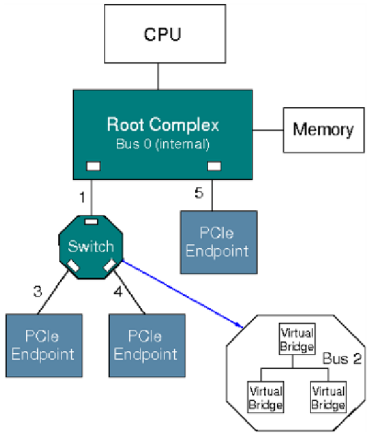
PCI Express

PCI Express je technológia umožňujúca sériovú point-to-point komunikáciu. Ponúka vysokú prenosovú rýchlosť a vysokú škálovateľnosť [9]. Táto kapitola s istou úrovňou abstrakcie opisuje, na akých princípoch technológia PCIe pracuje.

Na začiatok si je potrebné uvedomiť, že PCIe funguje viacej ako sieť, než ako zbernica. Predchádzajúce verzie PCI, vrátane PCI-X, sú naozajstné zbernice, pretože ich tvoria paralelné vodiče, ktorými sú pripájané periférie k procesoru. Na rozdiel od klasických zberníc, v PCIe sú dáta serializované a prenášané vo forme paketov, používajúc rôzne mechanizmy – počnúc riadením toku, až po detekciu chýb či retransmisiu.

2.1 Topológia

Zariadenia v PCIe sú prepojené do tzv. *„stromovej topológie“*. Na obrázku 2.1 je možno vidieť príklad stromovej topológie. Koreňom tejto topológie je *„root complex“* – zariadenie, ktoré prepája samotný procesor a pamäťový podsystém s PCIe zbernicou. Root complex generuje transakcie v mene procesora. Procesor tak môže komunikovať s PCIe zariadeniami pomocou vlastnej zbernice, implementovanej medzi ním a root complexom. Keďže na každej linke môžu byť pripojené maximálne dve zariadenia, dôležitým prvkom v topológii je switch (prepínač). Ten prepína a smeruje komunikáciu medzi viacerými linkami. Listové prvky tejto topológie tvoria PCIe koncové body, resp. PCIe periférie. Všetky zariadenia sú vrámci topológie jednoznačne identifikované pomocou trojice hodnôt – Bus Number, Device Number a Function Number. [3]



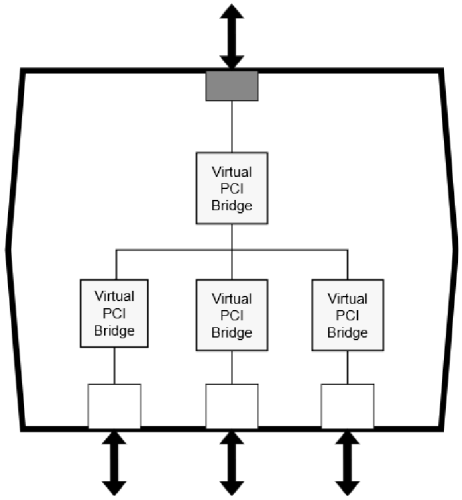
Obr. 2.1: Stromová topológia PCIe [3]

2.1.1 Root complex

Rozhranie medzi procesorom a PCIe zbernicou môže obsahovať niekoľko komponentov, rozhraní a čipov. Toto rozhranie sa súhrnne označuje ako *„root complex“*. Špecifikácia neobsahuje presný opis toho, čo všetko patrí pod root complex, avšak presne opisuje jeho úlohy. Root complex obsahuje jeden alebo viacero PCI Express portov. Každý takýto port vytvára novú samostatnú hierarchiu, ktorá môže byť zložená z jedného PCIe endpointu alebo v zapojení spolu so switchom viacerých endpointov. Schopnosť point-to-point smerovania komunikácie medzi jednotlivými hierarchiami je voliteľná. Jednou z možností ako takéto smerovanie zaistiť, je integrovanie switchu priamo do root complexu. ([14], str. 41)

2.1.2 Switch

Switch je označenie zariadenia, ktoré obsahuje dve či viacero PCIe-to-PCIe premostení. Na obrázku 2.2 je príklad switchu so štyrmi portami, ktorý obsahuje štyri virtuálne premostenia. Tie sú vnútorne pospájané nešpecifikovanou zbernicou. Každé takéto virtuálne premostenie obsahuje konfiguračnú hlavičku, vďaka ktorej switch určuje miesto smerovania. Port smerujúci k root complexu (na obrázku sivý) sa označuje ako *„upstream“*, ostatné porty ako *„downstream“*. Úlohou switchu je odoslať všetky transakcie z ľubovoľného vstupného portu na iný ľubovoľný výstupný port. Transakcie sú smerované pomocou jedného z troch smerovacích mechanizmov. Dva z týchto mechanizmov (ID routing a address routing) sú prenesené z PCI zbernice, zatiaľ čo ten tretí priniesol samotný PCIe a nazýva sa implicitné smerovanie (z angl. *implicit routing*) [2].



Obr. 2.2: Príklad switchu PCIe, inšpirované z [3]

2.1.3 PCIe Endpoint

Ako PCIe koncové zariadenia sa označujú tie zariadenia, ktoré nie sú switch alebo root complex, ale zároveň sa správajú ako žiadateľ alebo realizátor (viď. sekcia 2.4) [2]. Štandardne sú to zariadenia ako USB a sieťová alebo grafická karta.

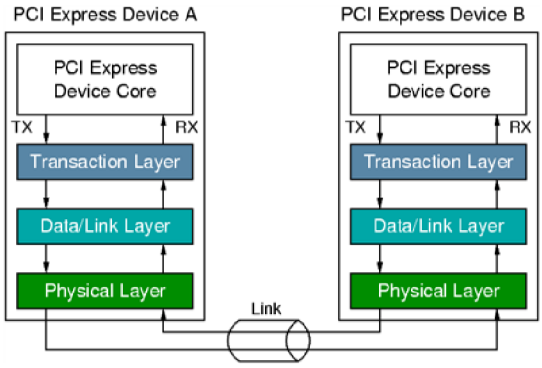
Špeciálnym druhom endpointu je tzv. *„legacy endpoint“*. Ak sa endpoint identifikuje systému ako *„legacy“*, môže podľa špecifikácie ([14], str. 42) podporovať staršie typy transakcií (`IO Read`, `IO Write`, `Memory Read Lock`), generovanie prerušovacích signálov MSI (Message Signaled Interrupt), implementovaných v PCIe kvôli spätnej kompatibilite. Takýto endpoint taktiež nie je povinný ani dodržovať 64-bitové adresovanie. Endpointy určené priamo pre PCIe zbernicu sa označujú ako natívne (z angl. *native*). Natívnym endpointom je zakázané používať zastaralé mechanizmy určené pre legacy endpointy, ako napríklad generovať `IO` transakcie a `Lock` požiadavky.

2.1.4 PCIe-PCI(-X) bridge

Zbernica PCI Express bola navrhnutá tak, aby na nej mohli pracovať aj spomínané legacy zariadenia, ktoré boli určené pre staršiu zbernicu PCI, resp. PCI-X. To najmä z dôvodu, že väčšina takýchto zariadení bola príliš drahých na to, aby pre spoločnosti bolo finančne možné ich všetky nahradiť. Preto sa v niektorých topológiách vyskytuje zariadenie označené ako *„Bridge“* resp. *„PCIe-to-PCI(-X) Bridge“*, ktoré je pripojené k root complexu. Toto zariadenie je určené na premostenie PCI(-X) zariadení do systému PCIe zbernice.

2.2 Vrstvy PCI Express

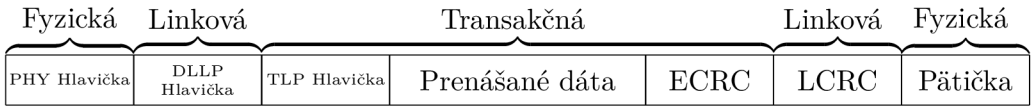
PCI Express je protokol zložený z troch vrstiev – fyzickej, linkovej (angl. *data link*) a transakčnej vrstvy, tie sú zobrazené na obrázku 2.3.



Obr. 2.3: Prehľad vrstiev PCIe [3]

Dáta sú po PCI Express prenášané vo forme TLP (Transaction Layer Packet). Pakety sa tvoria postupne po vrstvách, pri posielaní paketu najskôr transakčná vrstva pridá k prenášaným dátam TLP hlavičku a na koniec ECRC (end-to-end cyclic redundancy checksum), ktorého použitie je voliteľné. K týmto dátam sa následne pribalí hlavička linkovej vrstvy a LCRC (link-layer cyclic redundancy checksum). Nakoniec sa na začiatok vloží hlavička fyzickej vrstvy a paket je pripravený k odoslaniu. ECRC a LCRC sú kontrolné súčty, ktoré zabezpečujú maximálnu spoľahlivosť, že v prípade chyby pri prenose bude táto chyba ihneď detegovaná. Fyzická vrstva pridá na začiatok informáciu o začiatku a konci paketu [9].

Na obrázku 2.4 je zobrazené obecné postupné zapuzdrenie prenášaných dát opísané vyššie.



Obr. 2.4: Zapuzdrenie jednotlivých vrstiev PCIe paketu

2.2.1 Transakčná vrstva

Transakčná vrstva vytvára z dodaných dát TLP – paket transakčnej vrstvy, ktorý následne posunie linkovej vrstve. V opačnom smere prijaté dáta odovzdá aplikačnému softvéru zariadenia. Do paketu vkladá hlavičku, samotné prenášané dáta a voliteľný kontrolný súčet.

PCI Express tiež na transakčnej vrstve implementuje riadenie toku založené na kreditovom systéme. Zariadenie na začiatku prijímania pošle informáciu o tom, aký je počiatočný kredit podľa voľného miesta vo vyrovnávacej pamäti. Podľa tejto informácie odosielajúce zariadenie vie, koľko paketov môže poslať. Po spracovaní paketu z vyrovnávacej pamäte, prijímajúce zariadenie vyšle signál odosielajúcemu zariadeniu, ktoré zvýši stav kreditu.

Quality of Service (QoS)

Niektoré pakety, napríklad také, ktoré obsahujú dáta z videokamery pripojenej pomocou špeciálnej karty, je potrebné doručiť skôr, ako iné. PCIe preto implementuje zabezpečenie kvality služieb (QoS). Tento mechanizmus má zabezpečiť predvídateľný bandwidth a odozvu. Operačný systém v spolupráci s aplikačným ovládačom priradí vyslaným paketom

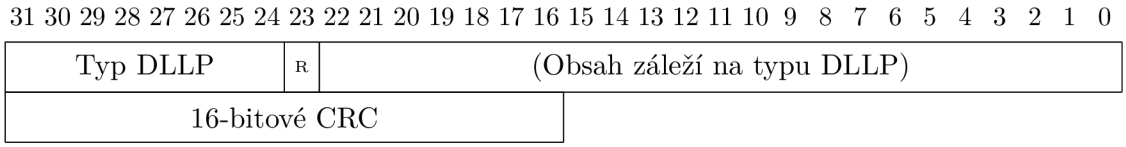
v TLP hlavičke príslušnú prenosovú triedu (TC7 - TC0) ([3], str. 82). Pakety s vyššou prenosovou triedou tak majú vyššiu prioritu pri prenose. QoS však môže byť využité len vtedy, keď operačný systém ako aj aplikačný softvér si je vedomý toho, že využíva PCIe zbernicu.

2.2.2 Linková vrstva

Linková vrstva plní tri dôležité úlohy v PCIe protokole a z dodaných dát vytvára DLLP (Data link layer packet) – paket linkovej vrstvy.

1. Určuje poradie paketov z vyššej transakčnej vrstvy pri odoslaní.
2. Zabezpečuje spoľahlivý prenos pomocou protokolu potvrdenia príjmu ACK (acknowledgment) a NAK (negative-acknowledgment).
3. Inicializuje a zabezpečuje kredit pre riadenie toku.

Linková vrstva generuje ku každému vyslanému paketu sekvenčné číslo, ktoré slúži ako unikátny identifikátor pre daný paket a je vložené do hlavičky odchádzajúceho paketu. Ďalej vygeneruje kontrolný súčet s názvom *„link cyclic redundancy code“*, skrátene LCRC, ktorý je vložený na koniec DLLP. Pakety z tejto vrstvy však môžu byť vysielané aj samostatne – bez payload-u (TLP). Na obrázku 2.5 je zobrazený formát takéhoto paketu. Podľa zakódovania typu paketu v prvom bajte je potom obsah príslušne spracovaný. Samostatný DLLP paket môže obsahovať dáta pre zachovanie integrity prenosu (ACK, NAK), dáta slúžiace správe napájania linku alebo správy týkajúce sa riadenia toku.



Obr. 2.5: Generický DLLP

Replay buffer

Na linkovej vrstve je implementovaný aj *replay buffer*, nazývaný aj *retry buffer*. Tento buffer ukladá odoslané pakety spolu s ich sekvenčných číslom a kontrolným súčtom linkovej vrstvy (LCRC) v poradí ich vysielania. Po informácií o prijatí prostredníctvom ACK signálu od prijímacej strany je paket z bufferu vymazaný.

Návrh replay bufferu umožňuje zmazať z bufferu viacero paketov jediným ACK signálom, a tým zmenšiť premávku na zbernici. Pri prijatí ACK signálu označujúceho, že paket bol úspešne prijatý, všetky uložené pakety s nižším sekvenčným číslom sú zmazané. V prípade prijatia NAK signálu, systém stále pozná posledný správny paket, čo mu opäť umožňuje zmazať pakety s nižším sekvenčným číslom začínajúc posledným správnym paketom. ([10], str. 321)

2.2.3 Fyzická vrstva

Zariadenia komunikujú pomocou prepojenia, ktoré sa nazýva interconnect alebo link. Link je tvorený z jednej alebo viacero ciest (angl. *lane*). Každá cesta obsahuje 4 vodiče - dva páry s diferenciálnou signalizáciou (pre každý smer vysielania jeden pár). Takéto spojenie teda umožňuje v jeden moment preniesť jeden bit oboma smermi. Niektoré požiadavky a

prerušenia už nové zariadenia nepoužívajú, avšak v návrhu sú zachované vedome pre to, aby bolo možné migrovať staršie PCI zariadenia do PCIe, a to bez nutnosti zmeny software v staršom zariadení a/alebo vytvoriť premostenie k pripojeniu PCI zariadení, bez stratenia ich funkcionalít. Podobne ako v iných vysokorýchlostných sériových protokoloch, hodinový signál je vložený do odosielaných dát, ako to je popísané v ďalšej kapitole v sekcii 3.1.

Správa napájania linku

Fyzická vrstva poskytuje správu napájania linku (angl. *Link Power Management)*, ktorá umožňuje znížiť spotrebu energie na nevyhnutné minimum. Bežný stav, v ktorom zariadenie pracuje, sa nazýva L0 stav. V momente, keď žiadne pakety nie sú vysielané a na linku nie je aktivita, zariadenie prejde do L0**s** (L0 Standby) stavu. Vstup a navrátenie z tohoto stavu je najkratšie, pretože k tomu zariadenie nepotrebuje softvér. Ďalšie dva stavy L1 a L2 majú nižšiu spotrebu, ale navrátenie z týchto stavov do stavu L0 trvá dlhšie, čím zvyšuje latenciu. Posledný stav L3 je úplné vypnutie linku, z ktorého zariadenie nemôže vygenerovať tzv. budiaci signál (angl. *wake-up signal*). ([3], str. 95)

2.3 Typy spojení

Tak ako bolo v sekcii 2.2.3 naznačené, spojenie (angl. *link*) môže byť tvorené z jednej či viacerých ciest. Link tvorený z jednej cesty sa označuje ako x1 link, z dvoch ciest sa nazýva x2 link, atď. PCI Express 3.0 podporuje vytvorenie spojenia typu x1, x2, x4, x8, x12, x16 alebo x32 ([10], str. 40), no výrobcovia bežne umiestňujú konektory vhodné len pre x1, x4, x8 alebo x16 typy spojení. Pridaním počtu ciest sa priamoúmerne zvýši rýchlosť. Často je však fyzicky umožnené vložiť kartu do konektoru s vyšším počtom ciest, toto sa označuje anglickým termínom *„up-plugging“*. Opačný prípad – vkladanie karty určenej pre väčší počet ciest do konektoru s menším počtom ciest, sa označuje ako *„down-plugging“* ([3], str. 686). *Down-plugging* je síce možný, avšak v praxi tomu často fyzicky zabraňujú konektory, ktoré na konci (kde by mohla byť karta nepripojená) obsahujú zarážku, čím znemožňujú jej pripojenie do menšieho konektoru.

2.4 Transakcie

„Transakcie sú definované ako séria jedného alebo viacerých paketových prenosov potrebných na dokončenie prenosu informácií medzi žiadateľom a realizátorom.“ [3]

V každej transakcií vystupujú dve strany, **žiadateľ** (angl. *requester*) a **realizátor** (angl. *completer*). Žiadateľ je zariadenie, ktoré iniciuje transakciu. Realizátor má za úlohu transakciu obslúžiť a *„kompletizovať“*. Transakcie sa delia na `posted` (bez odpovedi) a `non-posted` (s odpoveďou). V transakciách typu `posted` žiadateľ zasiela realizátorovi TLP, na ktorý realizátor neodpovie žiadnym *„completion“* paketom a to aj v prípade, že z nejakého dôvodu tieto dáta nemôže prijať. V transakciách typu `non-posted` žiadateľ dostane od realizátora odpoveď spoločne s dátami, alebo bez nich a spolu s tým obdrží identifikátor danej požiadavky. V prípade chyby bude v odpovedi nastavený príznak chyby. V tabuľke 2.6 sú všeobecne zobrazené TLP transakcie, ktoré na PCIe zbernici môžu prebehnúť, a ich príslušné typy.

Názov transakcie	Typ transakcie
Memory Read	Non-Posted
Memory Write	Posted
Memory Read Lock	Non-Posted
IO Read	Non-Posted
IO Write	Non-Posted
Configuration Read (Typ 0 a Typ 1)	Non-Posted
Configuration Write (Typ 0 a Typ 1)	Non-Posted
Message	Posted

Obr. 2.6: Názvy a typy transakcií

2.4.1 Memory Read

Jedna z najzákladnejších transakcií na PCIe zbernici je transakcia typu `Memory Read`. Legacy zariadenia zostrojené pre PCI zbernicu využívajú namiesto transakcie typu `Memory Read Request` transakciu `IO Read`. Patrí medzi transakcie, ktoré sú typu *Non-Posted*. To znamená, že po zbernici budú odvysielané minimálne dva pakety. Jeden paket, ktorý predstavuje požiadavku na čítanie (`Memory Read Request`) z určenej adresy z pamäte a druhý (prípadne viacero) ako completion paket. V pakete, v ktorom zariadenie žiada o čítanie, je v hlavičke vyplnená hodnota o veľkosti jeden bajt s názvom *Tag*. Táto hodnota bude obsiahnutá aj v hlavičke completion paketov, vďaka čomu môže zariadenie identifikovať prichádzajúci paket a priradiť ho k požiadavke, ktorú odoslalo.

2.4.2 Memory Write

S veľkým počtom zariadení PCIe, je potrebné komunikovať aj zápisom do ich pamäte. Na to typicky slúži transakcia typu `Memory Write`. Táto transakcia je typu *Posted* čo znamená, že prijímajúce zariadenie na ňu neodpovedá completion paketom. Na rozdiel od transakcie `Memory Read Request`, hodnota `Tag` bude nepoužitá.

2.4.3 Konfiguračné transakcie

Konfiguračné transakcie sú využívané v konfiguračnom procese, opísanom neskôr v sekcii 2.8. Tieto transakcie sú nevyhnutné pre správne fungovanie celej PCIe hierarchie. Jediný prvok, ktorý môže konfiguračné transakcie vytvárať, je root complex ([3], str. 718). Rovnako ako pri doposiaľ spomenutých transakciách, rozlišujeme `Configuration Read` a `Configuration Write` transakciu, pre čítanie resp. zápis. Všetky konfiguračné transakcie sú typu *Non-Posted*, teda aj pri konfiguračnom zápise je vyžadovaný completion paket bez dát ako odpoveď, čo predstavuje potvrdenie konfigurácie.

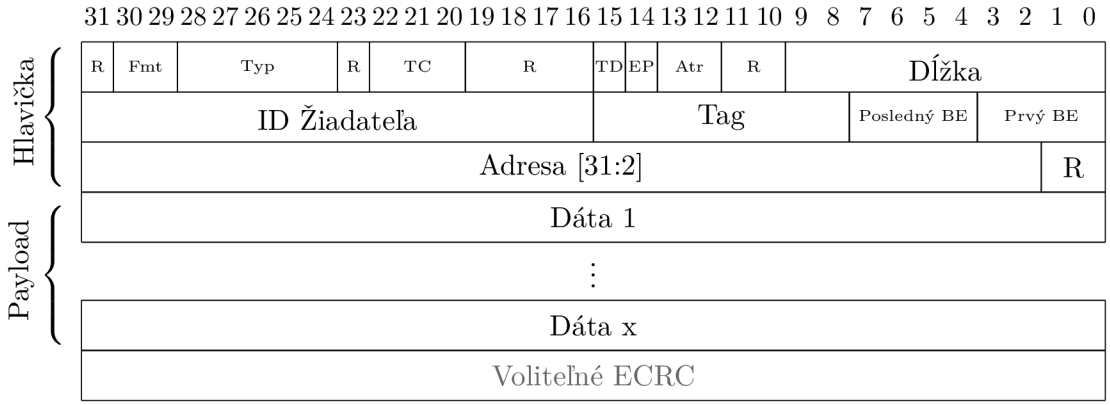
2.4.4 Locked transakcie

Špeciálnym typom transakcií sú tzv. locked požiadavky. Tie sú využívané vtedy, keď procesor potrebuje výhradný prístup ku koncovému zariadeniu. Takáto požiadavka môže byť vytvorená výhradne len root complexom a jej completer môže byť iba legacy zariadenie. Po vyslaní `Memory read Locked` transakcie (MRdLk) je transakcia smerovaná pomocou medziľahlých bodov (switch). Celá cesta z root complexu až po koncový bod je uzamknutá, vrátane cieľového a zdrojového portu. Po doručení paketu cieľovému zariadeniu môže zariadenie poslať požadované dáta buď pomocou jedného alebo viacero completion locked

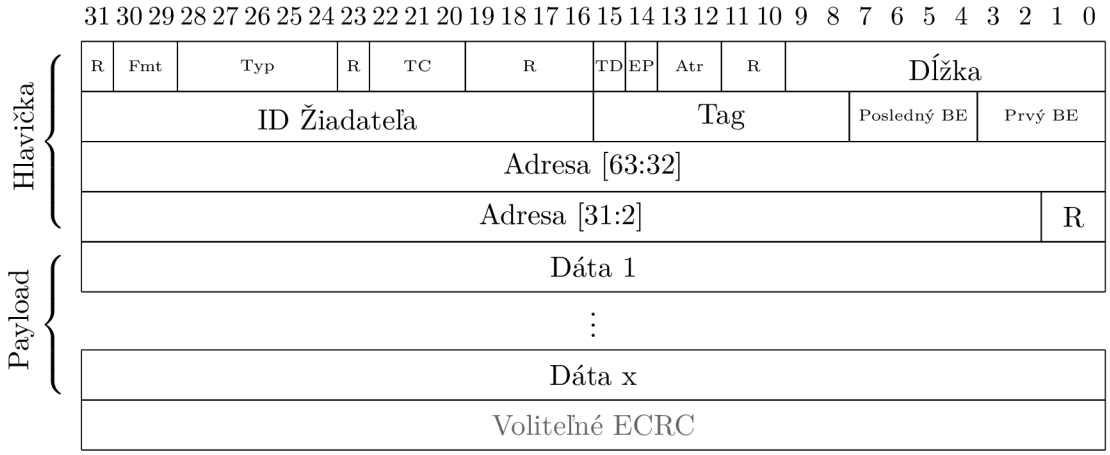
paketov s dátami (CplDLk), alebo môže vrátiť chybu pomocou completion locked paketu bez dát (CplLk). Uzamknutie cesty zotrvá až do chvíle, pokiaľ root complex neodošle tzv. *unlock* správu koncovému bodu. ([3], str. 59)

2.5 Formát paketu

Formát paketu závisí od vlastností ako sú typ transakcie, či použité adresovanie (64-bitové, 32-bitové).



Obr. 2.7: Formát paketu pri 32-bitovom adresovaní



Obr. 2.8: Formát paketu pri 64-bitovom adresovaní

Na obrázkoch 2.7 a 2.8 je zobrazené zloženie paketov, v závislosti od použitej veľkosti adresovania. V pakete sú definované parametre ako jeho `dĺžka`, `identifikátor žiadateľa` vrámci PCIe stromu, `tag` na identifikovanie požiadavky, `adresa` do pamäte a v neposlednom rade `dáta`. Pri niektorých typoch požiadaviek ako je napríklad `memory read`, takýto paket dáta neobsahuje a odosiela sa iba hlavička. V nasledujúcich podsekciách sú opísané najdôležitejšie polia z paketov na obrázkoch 2.7 a 2.8.

Formát a typ

Pri vytváraní a čítaní paketu, je pre určenie niektorých hodnôt v pakete potrebné poznať správne kódovanie jednotlivých hodnôt polí. Samotný typ paketu určuje dvojica hodnôt `Fmt` a `Typ`. V tabuľke 2.1 sú zobrazené niektoré (najčastejšie) kombinácie hodnôt, ktoré spolu určujú spomínaný typ paketu.

Typ TLP	**Formát**	**Typ**	**Popis**
MR	000 / 001	0 0000	Memory Read Request
MRL	000 / 001	0 0001	Memory Read Request Locked
MW	010	0 0000	Memory Write Request
IOR	000	0 0010	I/O Read Request
IOW	010	0 0010	I/O Write Request
CR0	000	0 0100	Čítanie konfigurácie typu 0
CW0	010	0 0100	Zápis konfigurácie typu 0
CR1	000	0 0101	Čítanie konfigurácie typu 1
CW1	010	0 0101	Zápis konfigurácie typu 1

Tabuľka 2.1: Kódovanie dvojíc FMT - Typ na konkrétne pakety, ktorým odpovedajú

Dĺžka

V poli `Dĺžka` sa udáva počet 32-bitových doublewordov (z angl. *double word*), ktoré určuje veľkosť samotného payloadu v TLP. V tabuľke 2.2 je naznačený spôsob kódovania tejto hodnoty.

Dĺžka [9:0]	**Veľkosť Payloadu v TLP**
00 0000 0001	1 DW
00 0000 0010	2 DW
⋮	
11 1111 1111	1023 DW
00 0000 0000	1024 DW

Tabuľka 2.2: Kódovanie poľa dĺžka v pakete

ID Žiadateľa

Na identifikáciu odosielateľa v rámci topológie sa používa 16-bitová hodnota na obrázkoch 2.7 a 2.8 označená ako `ID Žiadateľa`. Ako bolo v sekcii 2.1 spomenuté, každé zariadenie je v topológii identifikované pomocou trojice hodnôt – číslo zbernice (angl. *Bus Number*), číslo zariadenia (angl. *Device Number*) a číslo funkcie (angl. *Function Number*). 16 bitov `ID Žiadateľa` sa preto skladá z týchto troch čísel, určených v *enumeračnom* procese, opísanom neskôr v sekcii 2.8. Tieto tri hodnoty sa do 16tich bitov mapujú nasledovne:

- Bity [15:8] sú číslo zbernice.
- Bity [7:3] sú číslo zariadenia.
- Bity [2:0] sú číslo funkcie.

Tag

Pri transakciách zahrňujúcich completion pakety je v pakete dôležité pole `tag`. Completion pakety rôznych transakcií nemusia prijímateľovi príjsť v poradí, v akom o nich prijímateľ požiadal, a to napríklad kvôli mechanizmom fungujúcim v PCIe, ako je *relaxed ordering* (viď. 3.6) alebo *QoS*. Takýto mechanizmus môže v medziľahlých bodoch ako sú switche – z dôvodu vyššej priepustnosti – preskladať poradie vysielaných paketov, resp. prioritizovať rôzne pakety. Ďalším dôvodom môže byť napríklad rôzna doba prístupu k požadovaným dátam medzi transakciami. Preto k priradeniu completion paketov k vyslanej požiadavke slúži bajtová hodnota `Tag`. Realizátor ju do completion paketov vkladá na základe rovnakej hodnoty, prijatej v požiadavke.

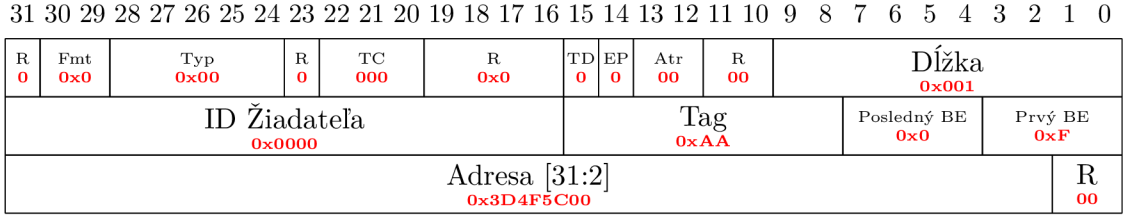
Adresa

Adresa v pakete nefunguje len ako samotný ukazovateľ na pamäťový blok v konkrétnom zariadení, ale v niektorých transakciách (napr. `memory read request`, `memory write`) funguje aj ako identifikátor cieľového zariadenia, kam má byť paket odoslaný. Tento spôsob smerovania je nazývaný *„address routing“*. Toto smerovanie funguje vďaka mapovaniu systémovej pamäte ako celku a nie len samostatne vrámci zariadení. Samotná adresa môže v pakete zaberať buď 32 alebo 64 bitov, v závislosti od adresovania zvoleného systémom.

2.6 Príklady transakcií

2.6.1 Príklad požiadavky na čítanie

Na obrázku 2.9 je zobrazený príklad paketu s doplnenými dátami, vyznačenými červenou farbou.

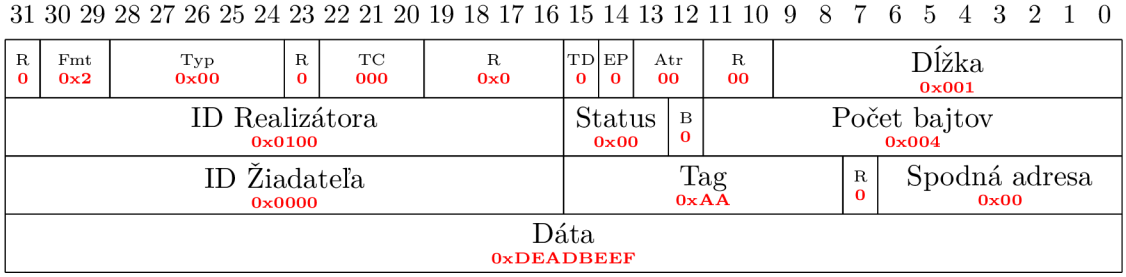


Obr. 2.9: Príklad Read Request TLP paketu

Ako prvé je v pakete na obrázku 2.9 možno vidieť polia `Fmt` a `Typ`. Tieto dve hodnoty spolu udávajú dĺžku paketu a zároveň presne definujú typ paketu. Pole `Dĺžka` s hodnotou `0x001` hovorí, že požadovaný obsah dát na prečítanie činí jeden `doubleword` (32 bitov). Pri požiadavke na čítanie treba venovať zvýšenú pozornosť poliam `ID Žiadateľa` (angl. *Requester ID*) a `Tag`. Pre completera je dôležité vedieť správne informácie pre úspešné odoslanie completion paketu, resp. jeho prijatie vyžiadaným zariadením. Podľa hodnoty v poli adresa je zrejmé, že paket bude smerovať na adresu `0xF53D7000` (keďže uvedená hodnota v poli je len pre rozsah bitov [31:2]).

2.6.2 Príklad completion paketu

Uvažujme o tomto príklade ako o odpovedi na predchádzajúcu požiadavku na čítanie. V prípade, že zariadenie obdrží požiadavku na čítanie, musí na ňu odpovedať completion paketom a to aj vtedy, ak túto požiadavku nie je schopné obslúžiť. V tomto príklade je však zobrazený prípad, kedy je zariadenie schopné obdržanú požiadavku uspokojiť. Na obrázku 2.10 sú do completion paketu červenou farbou doplnené dáta.



Obr. 2.10: Príklad Memory Write TLP paketu

Opäť ako pri predošlom pakete, polia `Fmt` a `Typ` udávajú typ paketu. V poli `Dĺžka` sa znova nachádza veľkosť TLP payload-u a to napriek tomu, že zariadenie vie, o koľko bajtov dát požiadalo. Pri jedinom completion pakete toto pole stráca na dôležitosti, avšak môže sa stať, že prijímacia strana prijala viacero completion paketov, ako odpoveď na rôzne transakcie, čím táto hodnota nadobúda v pakete dôležitosť. Takisto ako pri predošlom príklade, paket obsahuje pole `ID Žiadateľa`, ale tentokrát pribudlo aj pole `ID Realizátora`. V pakete je preto možné vidieť, že completion paket prichádza zo zariadenia identifikujúceho sa hodnotou `0x0100` a je určené pre zariadenie s hodnotou `0x0000` (Root Complex). Pri jedinom completion pakete opäť stráca dôležitosť `Počet bajtov`, ale v zásade ide o pole, ktoré udáva počet bajtov zostávajúcich na odoslanie, a to aj vrátane tých, ktoré sú obsiahnuté v danom pakete. Hodnota `Tag` činí `0xAA`, keďže, ako bolo v úvode príkladu spomenuté, tento completion paket je odpoveďou na predošlú požiadavku na čítanie. To znamená, že do paketu je vložený rovnaký `Tag`, ako mu bol zaslaný v požiadavke. Pole `Status` je nastavené na nulu, čo indikuje, že nevznikli žiadne problémy a teda, že požiadavka je kompletovaná. Bit `B` je nastavený vždy na log. 0 okrem prípadu, že paket pochádza od zariadenia, ktoré je premostené pomocou PCI-X-to-PCI bridge-u. `Spodná adresa` obsahuje spodných 7 bitov z adresy, ktorá bola čítaná (v tomto prípade bol požiadavok o čítanie z adresy `0xF53D7000`) a nakoniec paket obsahuje samotné prečítané dáta z adresy s hodnotou `0xDEADBEEF`.

2.7 Konfiguračný priestor

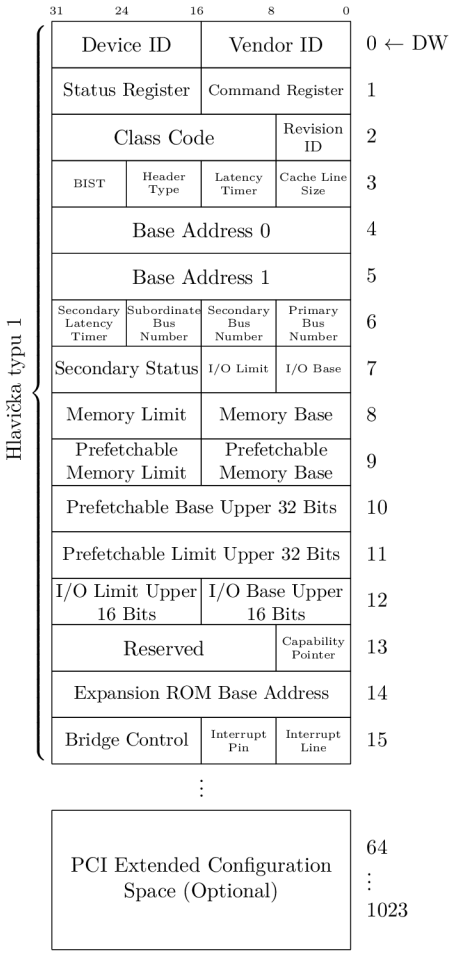
Konfiguračný priestor predstavuje základný spôsob pre PCIe, na vykonanie automatickej konfigurácie zariadení. Vďaka automatickej konfigurácii je možné, že PCIe obsahuje už spomínanú „*Plug and Play*“ funkcionalitu. Konfiguračný priestor sa rozdeľuje na **PCI konfiguračný priestor** a **PCI rozšírený konfiguračný priestor**. Ten rozšírený nie je v PCIe podporovaný staršími (legacy) zariadeniami. Pre nastavovanie tohoto priestoru sa využívajú špeciálne zápisové a čítacie transakcie spomenuté v sekcii 2.4.3.

2.7.1 PCI Konfiguračný priestor

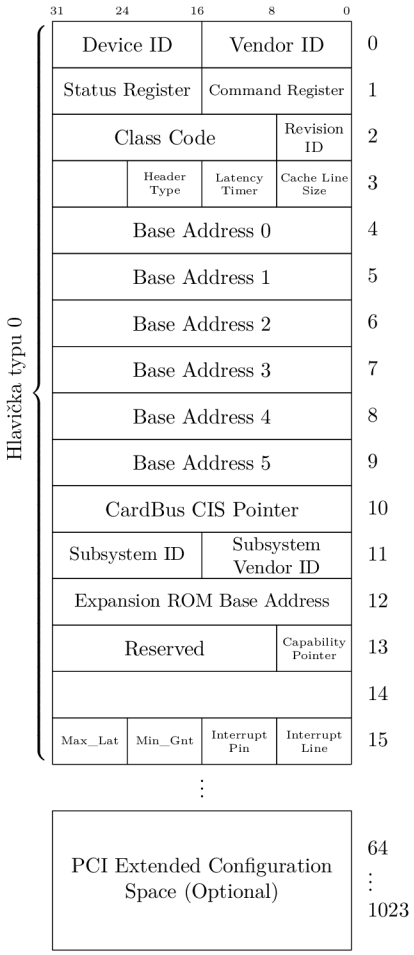
Základný PCI konfiguračný priestor zaberá prvých 64 doublewordov, jeho veľkosť je teda 256 bajtov. Rozlišujú sa tu dva možné typy hlavičiek konfiguračného priestoru, **typ 0** a **typ 1**. Typ 0 využívajú pre konfiguráciu koncové body, zatiaľ čo typ 1 sa využíva na konfiguráciu switchov a root complexu. Podľa využívaného typu sa mení hlavička konfiguračného priestoru a spôsob, akým je do neho potrebné nahrať konfiguračné dáta. Oba typy však obsahujú niekoľko **bázových registrov** (z angl. *Base Address Register*), ktoré sa označujú skratkou BAR, niekoľko štandardných registrov a miesta pre ďalšie konfiguračné informácie ([3], str. 715). Na obrázkoch 2.11 a 2.12 je naznačený konfiguračný priestor, konkrétne rozdielne hlavičky typu 1 a typu 0.

V prvých 64 bajtoch konfiguračného priestoru, je obsiahnutá hlavička typu 1 resp. typu 0. V nasledujúcich 192 bajtoch sa nachádzajú ďalšie sety registrov, ktoré sú špecifické pre dané zariadenie, a *„capability“* registrov, v ktorých sú napríklad informácie o systémových parametroch, ktoré zariadenie podporuje. Medzi týmito registrami sa nachádzajú aj informácie o linku.

Za týmto priestorom, taktiež nazývaným aj *„PCI kompatibilným“*, sa nachádza voliteľný rozšírený konfiguračný priestor, typický pre novšie zariadenia (legacy zariadenia ho nepodporujú). Celý konfiguračný priestor teda môže tvoriť veľkosť až **4kB**.



Obr. 2.11: Konfiguračný priestor s hlavičkou typu 1



Obr. 2.12: Konfiguračný priestor s hlavičkou typu 0

Bázové registre

Bázové registre fungujú ako komunikačný kanál so zariadením, udržujú v sebe rozpätie adries (začiatočnú adresu a dĺžku), na ktoré dané zariadenie môže odpovedať ako `completer` ([3], str. 136). Samotné bázové registre sú pri *enumeračnom* procese preskúmané systémovým softvérom, ten následne po ich preskúmaní alokuje blok pamäte a nahrá do nich tzv. bázovú adresu (adresa ktorou sa alokovaný blok začína). Pri použití jediného bázového registra môže rozpätie adries ukazovať na blok maximálnej veľkosti 4GB. Preto PCIe umožňuje spojiť dva registre a umožniť tým 64-bitové adresovanie.

2.8 Vyjednanie parametrov

Pri štarte systému s implementovanou PCI Express zbernicou je konfigurácia a topológia neznáma. Konfiguračný softvér musí prehľadať fyzický priestor a objaviť prípadné PCIe zariadenia pripojené do zbernice. Tento proces prehľadávania priestoru je označovaný ako *enumeračný* proces ([3], str. 714). S veľkou mierou abstrakcie sa dá povedať, že tento proces očísluje linky pomocou implementácie klasického vyhľadávacieho algoritmu hľadania do

hĺbky (z angl. *Depth-first search*) a očísluje aj vnútorné zbernice každého switchu. Tým dokáže určiť topológiu systému a prečítaním všetkých konfiguračných priestorov zariadení vykoná ďalšie konfiguračné kroky, aby ich uspokojil. Z tohoto číslovania vychádza aj trojica čísel Bus Number, Device Number a Function Number, ktoré – ako bolo už povedané – jednoznačne identifikujú každé zariadenie na zbernici.

2.9 Vývoj rýchlostí

Prvá verzia PCI Express bola predstavená v roku 2003 a priepustnosť x1 linku v jednom smere predstavovala 250 MB/s pričom dáta boli odosielané rýchlosťou 2.5 GHz [7]. Pri ďalšej verzií 2.0, predstavenej v roku 2007, sa rýchlosť odosielania zdvojnásobila na 5 GHz, čo zvýšilo priepustnosť x1 linku v jednom smere na 500 MB/s [11]. Iná zmena, ktorou nebolo najmä zvýšenie rýchlosti, prišla v roku 2010 s treťou revíziou PCIe štandardu. Prenosová rýchlosť sa prvýkrát nezdvojnásobila – zvýšila na 8 GHz, ale nové zavedené technológie ako napríklad nové kódovanie znakov 128b/130b (z pôvodného 8b/10b) opísané v sekcii 3.1 spôsobili, že priepustnosť dosiahla skoro dvojnásobok (985 MB/s).

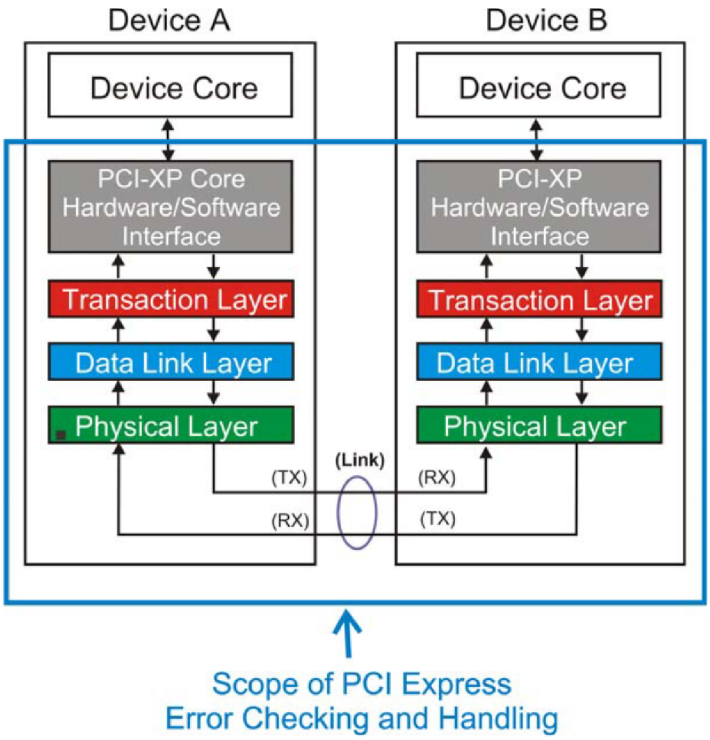
2.10 Hot Plug

Pojmom hot plug sa označuje možnosť zapojenia zariadenia za behu systému. PCIe podporuje nie len samotný hot plug ale aj hot unplug – čo je odpojenie zariadenia za chodu systému. V PCI bola táto funkcionalita zabezpečená ústredným kontrolérom, ktorý takéto odpojenie resp. pripojenie sledoval. V PCIe je táto funkcionalita zabezpečovaná priamo samotnými portami, do ktorých je zariadenie pripájané, pomocou konkrétnych pinov (PRSNT2#). Tie pri pripojení resp. odpojení odošlú tzv. *interrupt* správu root complexu, ktorou aktivujú hot plug softvér ([3], str. 46).

2.11 Detekcia chýb

PCI Express obsahuje množstvo mechanizmov, ktorými je tento štandard schopný chyby vzniknuté počas prenosu detegovať, informovať o nich a častokrát aj opraviť. V nasledujúcej sekcii sú priblížené najčastejšie chyby, ktoré môžu vzniknúť, a mechanizmy, ktoré ich detegujú a vyriešia. Počet vzniknutých chýb počas prenosu ale aj ich typ, má nesporný dopad na celkový výkon zbernice. Na obrázku 2.13 je modrou farbou vyznačený rozsah, pokiaľ tieto mechanizmy siahajú a môžu detegovať vzniknuté chyby. Tieto chyby PCIe rozdeľuje do troch kategórií; opraviteľné (angl. *correctable*), nefatálne (angl. *non-fatal*) a fatálne (angl. *fatal*) ([3], str. 357).

Opraviteľné chyby sú také, ktoré majú vplyv na priepustnosť zbernice prípadne jej odozvu, ale žiadne dáta sa nestratili a chyby môžu byť hardvérom opravené. Do kategórie neopraviteľných chýb spadajú tie, ktoré sú nefatálne, a aj tie, ktoré sú fatálne. Pri oboch typoch sú prenášané dáta stratené. Rozdiel medzi nimi je taký, že s nefatálnymi chybami si dokáže poradiť hardvér, pri tých fatálnych je už potrebný zásah systémového softvéru, pretože majú vplyv na integritu samotnej PCIe zbernice.



Obr. 2.13: Obrázok zobrazujúci rozsah hľadania a oznamovania chýb [3]

2.11.1 Kontrola chýb na transakčnej vrstve

Na transakčnej vrstve kontrolujú chyby mechanizmy implementované v žiadateľovi a realizátorovi. Počas prenosu cez medziľahlé switche sa táto vrstva nekontroluje (okrem ECRC) ([3], str. 358).

Kontrola ECRC súčtu

ECRC je kontrolný súčet, ktorý je v pakete voliteľný a samotné zariadenie musí oznámiť softvéru, že má túto funkcionalitu implementovanú. Switche musia doručiť TLP s nezmeneným ECRC z vysielacieho až do cieľového portu zariadenia. Switche môžu vykonať kontrolu ECRC súčtu počas toho, ako ho vysielajú ďalej, a v prípade chyby ju môžu oznámiť. Nemôžu však brániť paketu dostať sa do koncového bodu ([14], str. 146). 32-bitový kontrolný súčet sa vygeneruje z celého TLP (hlavička a payload) pomocou polynómu a vloží sa na koniec TLP. Kontrola paketu potom prebieha rovnako, zariadenie opäť vygeneruje ECRC (vynechá samotné ECRC na konci paketu) a porovná ho. V prípade, že sa nezhodujú, koncové zariadenie oznámi chybu ako *„ECRC Error“*. Po oznámení chyby je nutné paket poslať znova, teda sa táto chyba radí do kategórie neopraviteľných *„non-fatal“* chýb. Špecifikácia tiež uvádza, že identifikačné údaje tohoto paketu by mali byť zaznamenané.

Deformácia, poškodené dáta, nepodporovaná alebo neočakávaná požiadavka

Príjem deformovaného paketu so zlým formátom (*„malformed TLP“*), poukazuje na možnú chybu v systéme, preto sa radí do fatálnych errorov, kedy je nutné informovať systémový softvér. Ďalšie štyri prípady sa radia do nefatálnych chýb. Ak má paket nastavený tzv.

„otrávený bit“ (z angl. *poisoned bit*), môže to značiť práve poškodené dáta. Takýto paket sa označuje ako *„poisoned TLP“*. Keďže PCIe nemá nástroje na to zistiť, ktoré dáta môžu byť poškodené, je nutné paket vyslať znovu. Rovnako je chyba, keď zariadenie prijme požiadavku, ktorú nepodporuje alebo obdrží completion paket ktorý nepožadovalo. Takúto chybu by mali detegovať všetky zariadenia. Zariadenia odosielajúce Non-Posted požiadavky taktiež implementujú mechanizmus, ktorý po určitom čase čakania na completion paket bez odpovede vyhlásia *„completion time-out“* chybu.

Voliteľné kontroly

Okrem ECRC sú k dispozícii ďalšie tri mechanizmy, ktorých implementácia na transakčnej vrstve je voliteľná:

- Kontrola chýb vo Flow Control protokole
- Prerušenie zo strany realizátora CA (*„Completer Abort“*)
- Pretečenie príjímača (*„Receiver Overflow“*)

Zatiaľ čo chyba CA môže byť spôsobená zlou manipuláciou s prostriedkami zariadenia, chyby vo flow control protokole či pretečenie prijímača indikujú vážnejšie chyby v systéme, ktoré je potrebné oznámiť softvéru.

2.11.2 Kontrola chýb na linkovej vrstve

Linková vrstva robí nie len kontroly, ale taktiež sú to práve jej pakety (DLLP), ktoré spravujú oznamovanie chýb. Následujúce chyby opísané v tejto sekcii, sú skontrolované žiadateľom, realizátorom a aj medziľahlými bodmi (switch).

Kontrola LCRC súčtu a sekvenčného čísla

Tak ako pri transakčnej vrstve, aj linková vrstva obsahuje v zásade rovnaký kontrolný súčet ako transakčná vrstva. Tento kontrolný súčet je však povinný. Pri detekcii zlého LCRC alebo sekvenčného čísla je paket zahodený ([14], str. 180). Zatiaľ čo LCRC chyba pri kontrole DLLP paketu je považovaná za nefatálnu, pri kontrole TLP s možnosťou využiť replay buffer (viď. 2.2.2) sa chyba môže považovať za opraviteľnú.

Ostatné chyby linkovej vrstvy

Ďalšie možné chyby zahŕňajú replay buffer. Ten implementuje časovač, ktorý počíta čas od posledného vyslaného znaku hocakého TLP. Časovač sa zastaví vtedy, keď prijímajúce zariadenie potvrdí príjem pomocou signálu ACK/NAK. Pri prekročení určeného časového limitu bez odpovede vyvolá chybu *„Replay Time-out“*. Naopak chyba, pri ktorej bol určitý TLP vyslaný už príliš veľakrát, sa označuje pojmom *„Replay Number Rollover“*.

2.11.3 Kontrola chýb na fyzickej vrstve

Chyby na fyzickej vrstve taktiež kontrolujú všetky zariadenia, ktorými paket prejde. Tieto chyby sa už však týkajú samotného fyzického prenosu. Keď zariadenie deteguje chybu v prenose, indikuje ju linkovej vrstve. Chyby na fyzickej vrstve ako napríklad *„Receiver Error“* sa považujú za opraviteľné typy chýb.

Kapitola 3

Parametre ovplyvňujúce priepustnosť PCI Express zbernice

Maximálna prenosová rýchlosť tretej generácie PCI Express je skoro 8 Gb/s. Táto rýchlosť predstavuje počet bitov prenesených po jednom páre vodičov v jednom smere za jednu sekundu. Naozajstná rýchlosť prenosu dát je však menšia, a to najmä kvôli bitom, ktoré sú potrebné k samotnému prenosu, ako aj kvôli rôznym kompromisom v návrhu tejto technológie. Táto kapitola vychádza z [12] a opisuje najdôležitejšie parametre, ktoré majú vplyv na výslednú rýchlosť a priepustnosť PCIe zbernice.

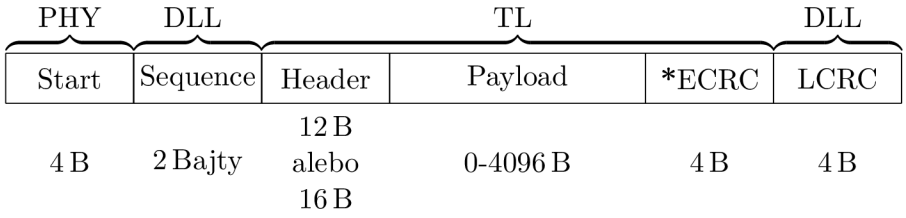
3.1 Kódovanie znakov

Na fyzický prenos bitov sa používa 128b/130b schéma kódovania znakov. To znamená, že pre každých 128 prenesených bitov je potrebné pridať ďalšie dva. Táto schéma zabezpečuje, že zariadenie prijímajúce dáta si je schopné zo signálu obnoviť svoj hodinový signál a zabezpečuje čo najnižší tzv. „*DC Balance*“. Pod pojmom „*DC Balance*“ sa rozumie rozdiel medzi počtom vyslaných log. núl a jednotiek. Zariadenia musia tento rozdiel aktívne sledovať a musia byť schopné detegovať rozdiel najmenej 511, pričom pri prekročení tejto hodnoty im nesmie pretiecť pamäťové miesto, kde túto hodnotu ukladajú. Už pri hodnote vyššej ako 15, sú do prenosu vkladané znaky na zníženie celkového „*DC Balance*“ ([14], str. 227).

Keďže pre každých prenesených 128 bitov je potrebné pridať dva navyše, vyplýva z toho, že strata na fyzickom prenose všetkých odoslaných dát tvorí takmer 2 %. Toto percento je veľkým pokrokom oproti starším generáciám PCI Express, ktoré používali 8b/10b schému kódovania znakov, ktorá spôsobovala stratu 20 % v prenose dát.

3.2 Hlavičky PCI vrstiev

Všetky dáta, ktoré sú cez PCIe zbernicu odoslané musia obsahovať hlavičky jednotlivých vrstiev, ktoré špecifikujú kľúčové vlastnosti paketu. Tieto hlavičky majú pri každom pakete rovnakú veľkosť, čo má pri posielaní malých blokov dát negatívny vplyv na celkovú priepustnosť. Zloženie paketu z pohľadu jednotlivých vrstiev je zobrazené na obrázku 3.1. Súčet všetkých hlavičiek, a teda celková veľkosť hlavičky odosielaného paketu sa môže pohybovať v rozmedzí 22 až 30 bajtov.



* ECRC je voliteľné

Obr. 3.1: Zloženie PCIe paketu

3.3 Transakčná vrstva

Veľkosť hlavičky transakčnej vrstvy je závislá najmä od faktorov ako použitie 32-bitového alebo 64-bitového adresovania, či vloženie nepovinného kontrolného súčtu (ECRC). Pokiaľ je použité 32-bitové adresovanie, veľkosť hlavičky je 12 bajtov, narozdiel od 16 bajtov pre 64-bitové adresovanie. Nepovinný kontrolný súčet ECRC zaberá 4 bajty. Okrem hlavičky majú veľký vplyv na celkovú priepustnosť aj koncové zariadenia, a to najmä spôsob, ako s PCIe systémom pracujú. Ak koncové zariadenia neefektívne využívajú zdroje a zbytočne zahlcujú zbernicu, pochopiteľne sa jej priepustnosť v smere k root complexu zníži.

3.4 Linková vrstva

Linková vrstva pridáva hlavičku o veľkosti dvoch bajtov a kontrolný súčet zaberajúci 4 bajty. Samotná hlavička a pätička však nie sú jediné elementy, ktoré na linkovej vrstve ovplyvňujú celkovú priepustnosť. Totižto, pre každý TLP poslaný z jedného zariadenia do druhého, je nutné vrátiť ACK či NAK, a to kvôli indikácii úspešnosti alebo neúspešnosti doručenia paketu. Odosielajúce zariadenie musí paket držať vo svojej vyrovnávacej pamäti, pokým nie je úspešne doručený. Tento systém zaisťuje vysokú spoľahlivosť doručenia dát príjemcovi. Pri vysokom počte prenosov je taktiež generované veľké množstvo DLLP, čo vplýva negatívne na priepustnosť zbernice.

Avšak, protokol umožňuje spojiť viacero DLLP, ktoré čakajú na prenos do jedného DLLP. Toto riešenie so sebou prináša výhody v podobe zmenšenia záťaže na zbernicu, ale aj nevýhody. Hlavná nevýhoda je v podobe situácie, keď sa ACK signalizácia neposiela dostatočne často. Vysielajúce zariadenie vtedy môže obmedziť rýchlosť odosielania paketov, a to pokým odoslané pakety nebudú z vyrovnávacej pamäte uvoľnené. Algoritmus implementujúci toto spájanie je ponechaný na dizajne konkrétneho zariadenia.

Ďalším elementom ovplyvňujúcim celkovú priepustnosť je riadenie toku. Ako bolo popísané v kapitole 2.2.1, PCIe používa riadenie toku založené na kreditovom systéme. Tento systém eliminuje vznik rizika pretečenia vyrovnávacej pamäte prijímajúcej strany. Zariadenie informuje o svojom stave kreditu pomocou špeciálnych FC (flow control) DLLP. Po uvoľnení vyrovnávacej pamäte, prijímajúce zariadenie pošle FC DLLP vysielajúcej strane, ktorým ho informuje o zmene stavu pridelených kreditov. Efektivita a rýchlosť s ktorou dokáže zariadenie tieto pakety odosielať má dopad na priepustnosť daného linku.

3.5 Vyrovnávanie rýchlostí liniek

Pre vyrovnanie rozdielov v rýchlosti jednotlivých liniek, predstavuje PCIe riešenie v podobe tzv. *„skip-ordered“* setov. Toto riešenie je implementované ako na fyzickej, tak aj na linkovej vrstve protokolu. Tieto sety majú typickú dĺžku 16 bajtov a musia byť vložené do komunikácie každých 370 až 375 jednotiek symbolového času. Symbolový čas je jednotka, ktorá udáva rýchlosť odoslania jedného bajtu (symbolu) dát do sériového spojenia. Špecifikácia PCI Expressu neumožňuje vkladať tieto sety, a ani iné pakety do TLP, to znamená, že *„skip-ordered“* a ostatné pakety spravujúce link môžu byť odoslané len medzi odoslanými TLP [12]. To je jeden z dôvodov, prečo zvýšenie MPS (Maximum Payload Size) – maximálnej veľkosti prenášaných dát – nezvýši efektívnosť priepustnosti linku.

3.6 Relaxed ordering

Relaxed ordering je mechanizmus, pôvodne predstavený v zbernicovom štandarde PCI-X. Avšak, PCIe tento mechanizmus prijalo s istými zmenami. Princíp spočíva v umožnení switchom, ktoré ležia medzi žiadateľom a realizátorom, zmeniť poradie niektorým transakciám, ktoré boli prijaté neskôr a predbehnúť tak transakcie čakajúce v rade na odoslanie, čo je v bežných prípadoch zakázané. Táto funkcionalita je paketu povolená samotným zariadením, ktoré paket odosiela, a to konkrétne v hlavičke paketu, kde sa nachádza bit, ktorý po uvedení do log. jednotky túto funkcionalitu paketu sprístupní ([3], str. 319). Relaxed ordering umožňuje flexibilitu v poradí transakcií a podľa spoločnosti NVIDIA, zvyšuje výkon zbernice až štvornásobne [13].

3.7 Systémové parametre ovplyvňujúce priepustnosť PCI Express zbernice

Okrem hlavičiek samotného protokolu, mechanizmom, ktoré na jednotlivých vrstvách pracujú a spôsobu, akým koncové body zbernicu využívajú, priepustnosť ovplyvňujú aj parametre jednotlivých zariadení pracujúcich na PCIe zbernici.

3.7.1 Maxmium payload size

Veľkosť užitočných dát prenášaných v TLP sa môže pohybovať od 0 do 4096 bajtov. MPS (maximal payload size) je konštanta udávajúca maximálnu veľkosť dát, ktoré je možné vložiť do TL paketu a je definovaná v každom uzle PCIe hierarchie. Špecifikácia hovorí, že softvér každého zariadenia musí zabezpečiť, aby jeho veľkosť MPS neprevyšovala MPS iného zariadenia v hierarchií. Celá zbernica sa preto musí prispôsobiť veľkosťou svojho MPS zariadeniu s najmenším MPS. Táto konštanta je nastavená pri konfiguračnom procese spomenutom v sekcii 2.8, každé zariadenie v hierarchii vloží svoj údaj MPS do svojho *„capability“* registra, ktorý sa nachádza v jeho konfiguračnom priestore. Zariadenia tak môžu zistiť MPS hodnotu každého zariadenia a prispôsobia ju k najmenšej zistenej hodnote. [12]

MPS konštanta ma priamy vplyv na to, koľko TLP je treba na odoslanie určitého objemu dát. So zvyšujúcou sa veľkosťou MPS konštanty sa zmenšuje počet potrebných TLP.

Rovnica definujúca efektívnosť paketu:

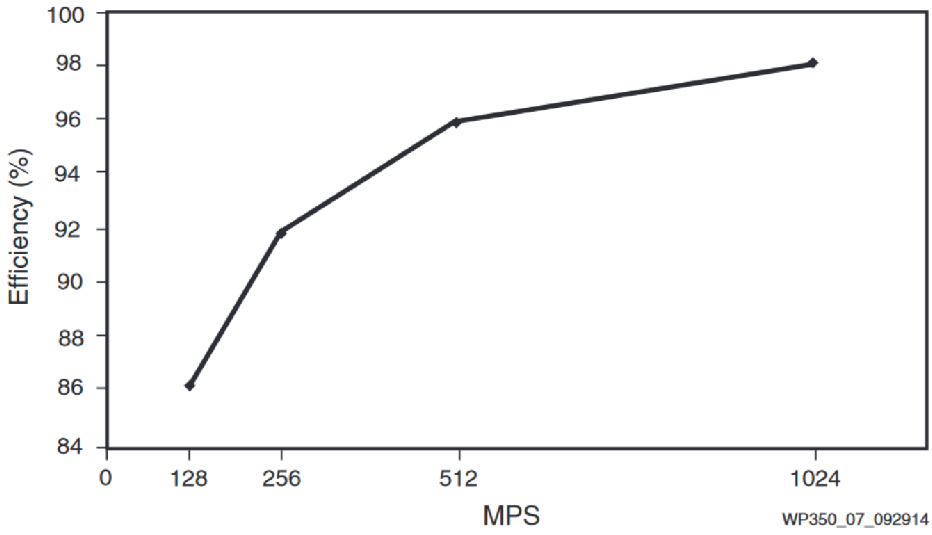
$$E = \frac{M}{M + H}$$

kde E predstavuje efektívnosť paketu, M predstavuje konštantu MPS a H je veľkosť hlavičky.

MPS (bajty)	Efektívnosť paketu (%)
128	86
256	92
512	96
1024	98

Tabuľka 3.1: Tabuľka efektívnosti paketu pre rôzne hodnoty MPS

Z tabuľky 3.1 môžeme vidieť, že zvýšenie efektivity pri nízkej hodnote MPS je signifikantné, no pri vyššej hodnote MPS zvýšenie efektivity s pribúdajúcou veľkosťou MPS klesá. Tento trend je možné sledovať aj na grafe zobrazenom na obrázku 3.2.



Obr. 3.2: Graf funkcie zobrazujúci efektivitu [12]

3.7.2 Maximum read request size

MRRS (maximum read request size) je konštanta určujúca najväčšiu veľkosť dát, ktorú môže požadovať požiadavka na čítanie zaslaná danému zariadeniu. Zariadenie si túto konštantu uloží podobne ako MPS do „capability“ registra. Táto hodnota môže byť väčšia ako hodnota MPS, v takom prípade je odpoveď rozdelená do viacerých TL paketov.

Systém používa túto konštantu na alokáciu priepustnosti v topológií. Limitácia objemu dát, ktoré môže zariadenie čítať v jednej požiadavke, predchádza obsadeniu celej priepustnosti zbernice jediným zariadením.

MRRS má významný vplyv na priepustnosť PCIe zbernice pretože určuje, koľko požiadaviek na čítanie musí byť zaslaných na prečítanie určitého objemu dát. Požiadavky na čítanie sa skladajú len z hlavičky a ak je ich potrebné poslať viac, znižujú tým priepustnosť celej zbernice.

Kapitola 4

Použité technológie

Kapitola sa zameriava na existujúce technológie, ktoré boli pri implementácií použité. Opisuje fungovanie zberníc MFB (4.2), MVB (4.3) a MI32 (4.4). Informácie o popisovaných zberniciach vychádzajú z OFM (Open FPGA Modules) dokumentácie [6].

4.1 Field-programmable gate array

FPGA (z angl. Field-Programmable Gate Array) je prefabrikované zariadenie, ktoré môže byť elektronicky naprogramované tak, aby reprezentovalo skoro ľubovoľný obvod alebo systém. Pre nízku až strednú produkciu sú tieto zariadenia výhodnejšie ako ASIC (z angl. Application Specific Integrated Circuit) obvody. [8]

Jednotka je navrhnutá pre akceleračnú kartu, na ktorej sa nachádza čip práve s touto technológiou.

4.2 Zbernica MFB

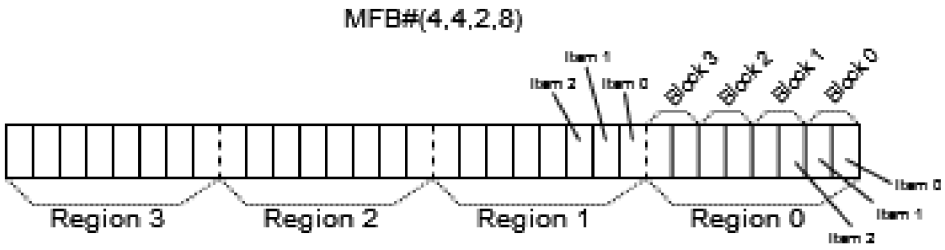
`Multi-Frame Bus` – skrátene MFB, je vysokorýchlostný typ zbernice, vyvíjaný združením CESNET, určený na prenos viacero rámcov v jednom takte. Jednotka túto zbernicu využíva na prenášanie dát medzi PCIe modulom. Tieto dáta sú následne PCIe modulom umiestnené do payload-u odoslaného PCIe paketu.

4.2.1 Princíp fungovania

Dáta sú v jednom takte prenášané v jednotke označovanej ako **slovo**. Tá je hierarchicky rozdelená na menšie jednotky, konkrétne **regióny**, **bloky** a **položky**. Počet regiónov, blokov a položiek sa môže prispôsobiť konkrétnemu využitiu zbernice. Veľkosť slova je teda možné vypočítať ako

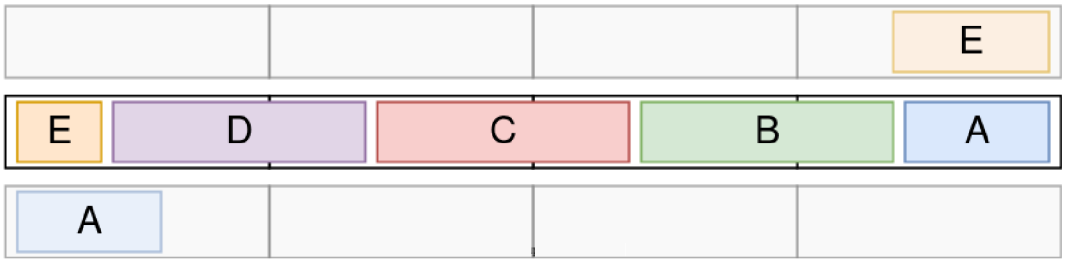
$$W = R \cdot B \cdot I \cdot S$$

kde **R** je počet regiónov, **B** je počet blokov v jednom regióne, **I** je počet položiek v jednom bloku a **S** označuje veľkosť jednej položky v bitoch. Začiatok rámca musí byť zarovnaný na začiatok bloku a jeho koniec na ľubovoľnú položku. Na obrázku 4.1 je zobrazená konfigurácia zbernice o štyroch regiónoch, štyroch blokov v jednom regióne, dvoch položiek v každom bloku a ôsmich bitoch v každej položke – skrátene MFB#(4,4,2,8).



Obr. 4.1: Príklad jedného slova zbernice

Obrázok 4.2 je príkladom odoslania piatich rámcov v rovnakej konfigurácií – (4,4,2,8), v jednom slove. Rámce **A** a **B** však nie sú v danom slove kompletné.



Obr. 4.2: Príklad rozdelenia piatich rámcov do troch slov

4.2.2 Signály zbernice

Zbernica využíva šesť kontrolných signálov a jeden signál na prenos samotných dát. Tabuľka 4.1 zobrazuje ich prehľad spoločne s výpočtom ich veľkostí, podľa zvolenej konfigurácie zbernice. Signály `SRC_RDY` a `DST_RDY` spolu tvoria akýsi *handshake* oboch strán a dáta sú platné len v prípade, že oba signály sú v logickej jednotke.

Signály `SOF` a `EOF` sa používajú na indikáciu, v ktorom regióne začína resp. končí odosielaný frame. V každom regióne môže začínať resp. končiť jediný frame, to znamená, že ak by celé slovo tvorilo iba jeden región, nebolo by možné preniesť v jednom takte dva frame-y. Signál `SOF_POS` slúži ako ukazovateľ na bloky, v ktorých začína každý frame a `EOF_POS` má od neho o čosi väčšiu veľkosť, pretože slúži ako ukazovateľ na konkrétne položky, v ktorých frame-y končia. Tieto ukazovatele obsahujú indexy bloku, v ktorom frame začína resp. index položky, v ktorom frame končí tak, ako to je naznačené v obrázku 4.2.

Názov	**Veľkosť signálu** (*bit*)	**Smer signálu**
DATA	REGIONS · REGION_SIZE · BLOCK_SIZE · ITEM_WIDTH	OUT
SOF	REGIONS	OUT
EOF	REGIONS	OUT
SOF_POS	REGIONS · $log2$(REGION_SIZE)	OUT
EOF_POS	REGIONS · $log2$(REGION_SIZE · BLOCK_SIZE)	OUT
SRC_RDY	1	OUT
DST_RDY	1	IN

Tabuľka 4.1: Tabuľka signálov zbernice MFB

4.3 Zbernica MVB

`Multi-Value Bus` skrátene MVB je typ zbernice, vyvíjaný združením CESNET, ktorý je v implementácií použitý ako komplementárny, k zbernici typu MFB. Na prenos dát používa rozdelenie do položiek a jej účel je najmä v prenášaní metadát o pevnej veľkosti.

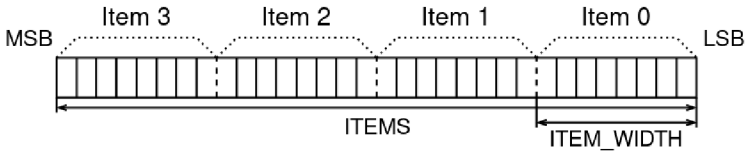
4.3.1 Princíp fungovania a signály

Na rozdiel od zbernice MFB, sa po zbernici prenášajú položky pevnej dĺžky. Prenášané dáta musia byť umiestnené do položky a ich veľkosť nemôže byť vyššia, ako veľkosť položky – táto podmienka závisí najmä od konkrétnej implementácie a použití zbernice. Počet položiek je závislý na jej konfigurácii. MVB využíva štyri signály zobrazené aj s ich veľkosťou v tabuľke 4.2. Opäť sa tu využívajú dva signály `SRC_RDY` a `DST_RDY`, ktoré predstavujú *handshake* oboch strán. Vyslané dáta sa považujú za platné len vtedy, keď obe strany majú tento signál v logickej jednotke. Samotné dáta sa vložia do ľubovoľnej položky a v signáli logického vektoru `VLD` sa nastaví logická jednotka bitu s indexom rovnakým, ako index položky, do ktorej boli tieto dáta vložené.

Názov	**Veľkosť signálu** (*bit*)	**Smer signálu**
DATA	ITEMS · ITEM_WIDTH	OUT
VLD	ITEMS	OUT
SRC_RDY	1	OUT
DST_RDY	1	IN

Tabuľka 4.2: Tabuľka signálov zbernice MFB

Na obrázku 4.3 je zobrazený príklad zbernice o štyroch položkách. Jednotka cez zbernicu typu MVB prenáša hlavičky odosielaných paketov.



Obr. 4.3: Príklad MVB zbernice o štyroch položkách

4.4 Zbernica MI32

Zbernica s nízkou priepustnosťou MI (*Memory Interface*), realizuje komunikáciu medzi softvérom a firmvérovými komponentami. Tieto komponenty (obvykle nejaký ovládač či register) môžu byť nakonfigurované zápisovými požiadavkami, alebo je ich hodnotu možné vyčítať požiadavkami na čítanie. Komponenty sú sprístupnené ich adresou, ktorá je zaslaná spoločne so žiadosťou. Predvolená bitová šírka, ktorá je v návrhu použitá a ktorá sa bežne používa je 32 bitov – preto je často používané označenie MI32.

4.4.1 Princíp fungovania a signály

V tabuľke 4.3 sú uvedené signály, ktoré MI zbernica používa. Smer signálu na tabuľke je platný pre zapojenie ako slave. Signál `ADDR` nesie adresu, z ktorej sa má vyčítať resp. do ktorej sa má zapísať obsah. Pri požiadavke na zápis potom signál `DWR` nesie samotné dáta k zápisu. `MWR` je v zbernici voliteľné a nemá dopad na fungovanie zbernice. `Byte enable` potom určuje, ktoré bajty sú v zapisovaných dátach platné. Signály `RD` a `WR` signalizujú požiadavku na čítanie a zápis.

Pri bežnej zápisovej operácií, odosielajúca strana nastaví adresu, do ktorej chce zapisovať (`ADDR`), dáta na zápis (`DWR`) a nastaví signál `WR` do log. jednotky. Toto nastavenie signálov musí byť vysielané dovtedy, pokým prijímajúce zariadenie nepotvrdí zápis signálom `ARDY`. Naopak pri čítaní stačí odosielajúcej strane nastaviť signály `ADDR` a `RD` v jedinom takte a čakať do chvíle, kedy prijímajúce zariadenie nastaví signál `DRDY` do log. jednotky, čím potvrdí platnosť dát v `DRD` signáli. V obidvoch prípadoch môže byť platnosť jednotlivých bajtov upravená signálom `BE`.

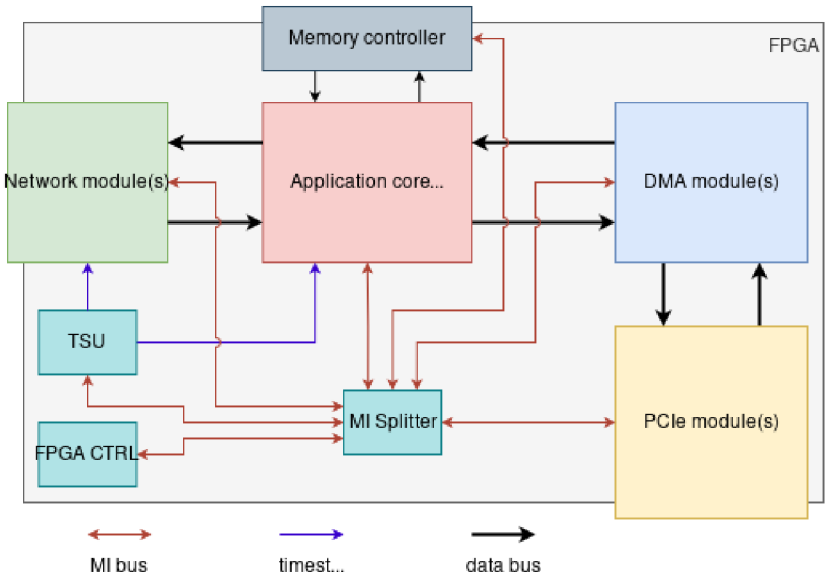
Názov	**Veľkosť signálu** (*bit*)	**Smer signálu**
ADDR (address)	ADDRESS WIDTH	IN
DWR (data write)	DATA WIDTH	IN
MWR (metadata write)	DATA WIDTH	IN
BE (byte enable)	DATA WIDTH / 8	IN
WR (write)	1	IN
RD (read)	1	IN
ARDY (address ready)	1	OUT
DRDY (data ready)	1	OUT
DRD (data read)	DATA WIDTH	OUT

Tabuľka 4.3: Tabuľka signálov zbernice MI

4.5 NDK

Jednotka bola vyvíjaná tak, aby mohla byť integrovaná pomocou nástrojov, ktoré poskytuje NDK (Network Development Kit) [4], vyvíjaný združením CESNET. Tieto nástroje majú mať za úlohu zjednodušiť vývoj vysokorýchlostných aplikácií, implementovaných na akceleračných kartách s FPGA čipom (najmä) od spoločnosti Xilinx. NDK poskytuje nástroje a knižnice na komunikáciu s kartou a jej vnútornými komponentami.

4.5.1 Architektúra NDK

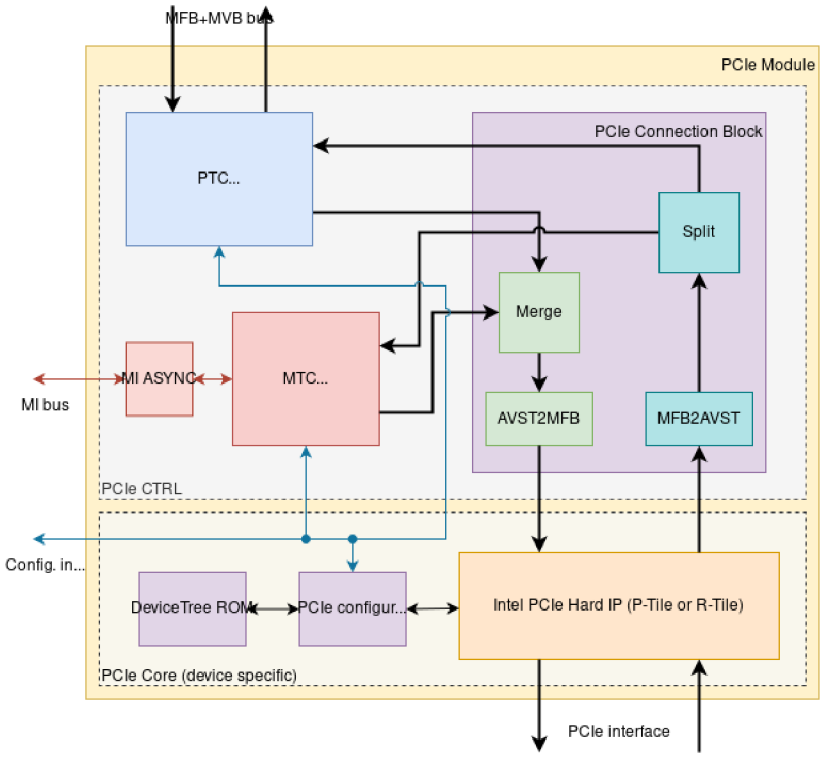


Obr. 4.4: Architektúra NDK [4]

Na obrázku 4.4 je zobrazená architektúra nahraná do FPGA čipu na akceleračnej karte. Bežná prax je vytvorenie aplikácie, ktorá bude vložená do aplikačného jadra karty, to má prístup k signálom z MFB zbernice, ktoré pochádzajú z DMA (Direct memory access) modulu. Tento modul sa stará o rozdeľovanie zaslaných dát podľa požiadavok PCIe modulu, vytváranie hlavičiek a vytvára akúsi abstrakciu pre aplikačné jadro nad PCIe rozhraním.

PCIe modul

Na obrázku 4.5 je zobrazený blok PCIe modul z obrázku 4.4 popísaného v predošlej sekcii. Práve s týmto blokom navrhnutá jednotka komunikuje, namiesto DMA modulu. Tento modul je zložený z dvoch častí, spodná časť modulu označená ako *„PCIe Core“* je špecifická pre každý typ akceleračnej karty. V tejto časti sa nachádza implementácia konfiguračného bloku, ROM (Read Only Memory) bloku, ktorý umožňuje čítanie firmvérových informácií a PCIe Hard IP bloku, ktorý realizuje komunikáciu s PCIe rozhraním. V druhej časti sa nachádza abstrakcia pre prekladanie DMA transakcií (PTC blok), nasledovaný PCIe connection blokom, ktorý prekladá MFB zbernicu na zbernicu typu Avalon-ST a naopak [4].



Obr. 4.5: Architektúra PCIe modulu [4]

4.5.2 Softvérová výbava NDK

Okrem samotnej architektúry ponúka NDK platforma aj softvérové nástroje, nazývané tiež *„NFB Tools“* [5]. Medzi tieto nástroje patria programy ako `nfb-info` na zistenie základných informácií o karte, `nfb-boot` umožňujúci nahranie novej konfigurácie do PCIe karty za chodu, či programy zobrazujúce štatistiky z karty ako `nfb-dma`. Najdôležitejším prvkom však je knižnica `libnfb`, ktorá poskytuje abstrakciu nad čítacími a zápisovými operáciami do komponentov karty, a to pomocou MI zbernice.

Kapitola 5

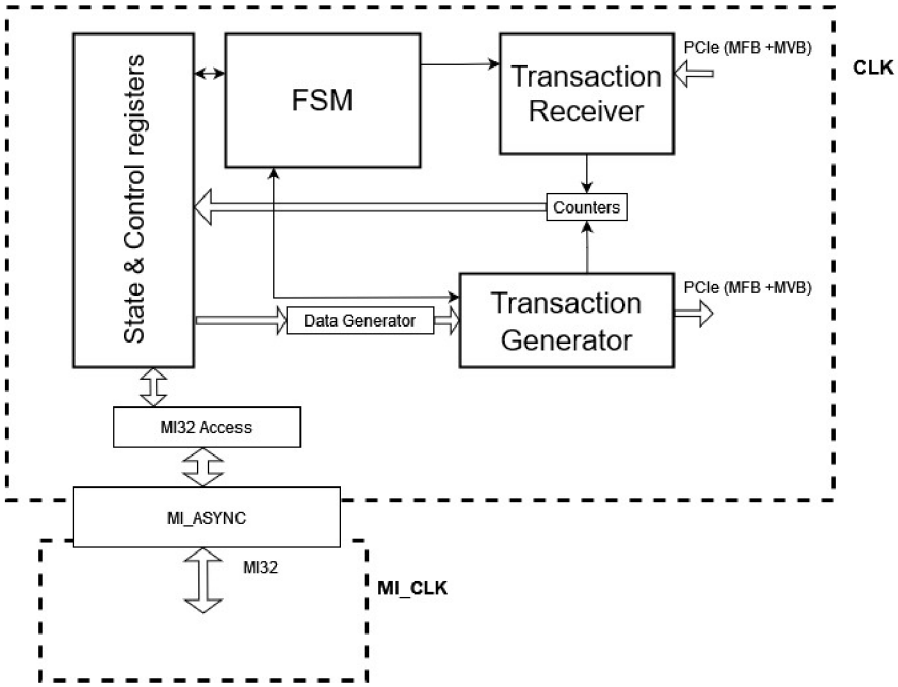
Architektúra jednotky

Táto kapitola podrobne pojednáva o navrhnutej a implementovanej architektúre jednotky a bližšie opisuje jednotlivé prvky, z ktorých sa skladá. Okrem toho opisuje, akým spôsobom sú použité technológie, ktoré boli spomenuté v kapitole 4.

5.1 Navrhnuté zariadenie

Zariadenie musí byť schopné merať rýchlosť zbernice v oboch smeroch. Samotné zariadenie bude vložené do akceleračnej karty, a teda bude merať rýchlosť medzi akceleračnou kartou a systémom, do ktorého je vložená. Je veľmi dôležité, aby zariadenie čo najoptimálnejšie pri generovaní využívalo každý jeden takt, a bolo tak schopné vyslať veľké množstvo dát na zbernicu.

Na obrázku 5.1 je zobrazená bloková schéma navrhnutého zariadenia. Riadiacu časť predstavuje konečný automat, v schéme vyznačený ako FSM (Finite State Machine). Po nahratí správnych dát a príznaku začatia merania cez rozhranie MI32 začne jednotka generovať pakety na MFB a MVB zbernice. Tieto pakety sú spracované PCIe rozhraním, ktoré ich následne odošle. Pri meraní typu TX (z karty do systému) sa posielajú `memory write` pakety, zatiaľ čo počítadlo meria čas, za ktoré tieto pakety sú schopné opustiť kartu až do momentu, kedy je karta schopná znovu odosielať ďalšie dáta. V smere RX karta začne generovať `memory read` pakety, ktoré budú vyžadovať čítanie z dopredu systémom alokovaného miesta. Merať sa bude čas od momentu poslania prvého `memory read` paketu až po moment prijatia všetkých očakávaných dát. Do kontrolného registra bude možné uložiť veľkosť prenosu a počet požadovaných prenosov.



Obr. 5.1: Bloková schéma

5.1.1 Stavové a kontrolné registre

Na uchovávanie informácií o meraní a na samotné ovládanie merania sú v návrhu určené registre, v blokovej schéme 5.1 označené ako blok `State & Control registers`. Tento blok predstavuje systém registrov, z ktorých budú tie najpodstatnejšie samostatne opísané v tejto sekcii. Do všetkých registrov (mimo registru určeného pre počítanie taktov) je možné nahrať ľubovoľnú hodnotu pomocou zbernice MI32, popísanej v sekcii 4.4. Cez túto zbernicu funguje aj vyčítanie obsahu všetkých registrov. Prehľad registrov je zobrazený v tabuľke 5.1.

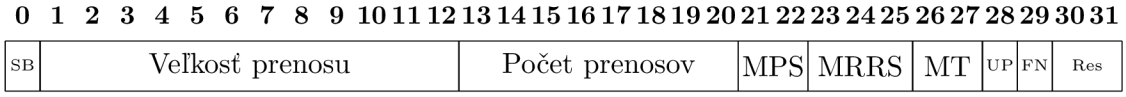
Adresa	**Typ registra**	**Povolený režim** (Read / Write)
0x00	Kontrolný register	`R/W`
0x04	Adresový register 1	`R/W`
0x08	Adresový register 2	`R/W`
0x0C	Counter register	`R`
0x10	Pattern register	`R/W`

Tabuľka 5.1: Typy implementovaných registrov a ich režim

Kontrolný register

Kontrolný register je zobrazený na obrázku 5.2, má veľkosť 4 bajty a začína na adrese **0x00**. Prvý bit `SB` je príznak na začatie merania, spolu s ním sa nastavuje do log. jednotky aj bit `FN`. Bit `FN` je po skončení merania vynulovaný. Ďalšími dvanástimi bitmi sa definuje veľkosť prenosu v bajtoch. Nutné je špecifikovať počet prenosov. Dôležité je tiež, aby softvér poznal parameter Maximum Payload Size a Maximum Read Request Size a nahral ho do kontrolného registra. Veľkosť prenosu a počet prenosov sú očakávané ako celé čísla bez znamienka, hodnoty očakávané na miestach MPS, MRRS a MT sú popísané v tabuľke 5.2.

Bit `UP` (skr. pre *use pattern*) predstavuje možnosť aktivácie 32-bitového vzoru, ktorý bude prenášaný v paketoch. Tento register obsahuje navyše dvojicu signálov (`fsm_reg1_cl_sel` a `fsm_reg1_cl`) – 32-bitovú masku a aktivačný signál, ktorými môže konečný automat (opísaný neskôr v sekcii 5.1.2) nulovať jednotlivé bity v kontrolnom registri.



Obr. 5.2: Obsah kontrolného registra

MPS	00b: 128 bajtov 01b: 256 bajtov 10b: 512 bajtov 11b: 1024 bajtov
MRRS	000b: 128 bajtov 001b: 256 bajtov 010b: 512 bajtov 011b: 1024 bajtov 100b: 2048 bajtov 101b: 4096 bajtov
MT	01b: Meranie v smere TX smerom zo zariadenia 10b: Meranie v smere RX smerom k zariadeniu

Tabuľka 5.2: Nastavenie parametrov kontrolného registra

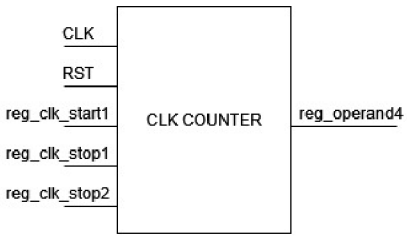
Adresový register

Registre na adresách **0x04** a **0x08** spolu tvoria úložisko pre 64-bitovú adresu do pamäte RAM, na ktorej je softvérom alokované miesto pre zápis vygenerovaných transakcií jednotkou, alebo z ktorého jednotka požaduje dáta (pri RX smere).

Counter register

Špeciálny register je umiestnený na adrese **0x0C**, pretože je upravený tak, aby fungoval ako počítadlo taktov. Obsahuje prídavné signály `reg_clk_start1`, `reg_clk_stop1` a `reg_clk_stop2`. Toto počítadlo je aktivované signálom z bloku FSM pri štarte merania a môže byť zastavené jedným z dvoch signálov – signálom z bloku FSM `reg_clk_stop1` alebo signálom z bloku Transaction Receiver `reg_clk_stop2`. Tieto signály sú vybudené do logickej jednotky iba po dobu jedného taktu, čo znamená, že register si musí uchovať svoj vnútorný stav – či svoj obsah má alebo nemá s nábežnou hranou hodinového signálu zväčšovať. Register ukončí počítanie len vtedy, ak bol jeden z logických signálov `reg_clk_stop1` a `reg_clk_stop2` uvedený aspoň v jednom takte do logickej jednotky a **zároveň** je MFB zbernica (opísaná neskôr v 4.2) v TX smere pripravená prijímať ďalšie pakety. Ak bol jeden zo signálov uvedený do logickej jednotky ale zbernica nebola schopná prijímať ďalšie pakety, register v počítaní zotrvá až do chvíle kedy je táto podmienka splnená. Tento register spolu

s vedomosťou o taktovacej frekvencií čipu, umožňuje určiť trvanie vykonaného merania. Na obrázku 5.3 je tento register zobrazený, vľavo sú vstupné signály a vpravo výstupné.



Obr. 5.3: Register počítajúci takty

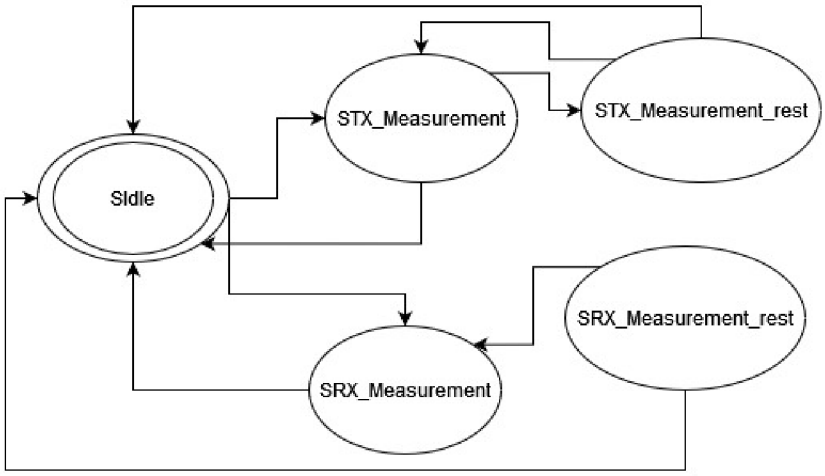
Pattern register

Pattern register umiestnený na adrese **0x10** slúži na uloženie 32-bitového vzoru, ktorým majú byť naplnené dáta paketov. Pre aktiváciu tohoto vzoru slúži `UP` bit v kontrolnom registri spomenutý v sekcií o kontrolnom registri vyššie.

5.1.2 Konečný automat

Konečný automat (na obr. 5.1 FSM) tvorí akýsi mozog celej jednotky. Jeho úloha je zotrvať v nečinnom stave, v ktorom sleduje obsah kontrolného registra. Po nahratí príznaku začatia merania vyhodnotí parametre vložené do kontrolného registra. Veľkosť transakcie v TX nemôže presiahnuť konštantu `MPS`. Ak ju požadovaná veľkosť transakcie presahuje, úlohou automatu je rozdeliť požiadavky na dielčie kúsky, ktoré bude možné po zbernici preniesť. Táto úloha je rovnaká aj pri meraní smeru RX, pri ktorom jednotlivé transakcie nemôžu presiahnuť konštantu `MRRS`.

Konečný automat využíva stavy (`SIdle`, `STX_measurement`, `STX_measurement_rest`, `SRX_measurement` a `SRX_measurement_rest`), pomocou ktorých zadáva požiadavky na generovanie paketov generátoru. Na obrázku 5.4 je jednoduchý náčrt jeho stavov, určený pre ľahšiu vizualizáciu. Pri požiadavkách, ktoré nepresahujú veľkosť `MPS` resp. `MRRS`, na zadávanie požiadaviek používa výhradne stav `STX_measurement` resp. `SRX_measurement`. V prípade však, že transakcia nejde rozdeliť na dielčie transakcie rovnakej maximálnej veľkosti, automat využije stav `STX_measurement_rest` resp. `SRX_measurement_rest`, na odoslanie dielčej transakcie so zostatkovými bajtami. Tieto dva stavy sú pridané najmä kvôli prehľadnosti kódu. Konečný automat taktiež využíva signálov ako `reg_clk_start1` a `reg_clk_stop1` na zapnutie a vypnutie počítadla taktov, či `fsm_reg1_cl` a `fsm_reg1_cl_sel` pre zmazanie príznaku na začatie merania `SB`.



Obr. 5.4: Vizualizácia stavov konečného automatu

V zhrnutí, princíp jeho fungovania spočíva najmä v monitorovaní kontrolného registra, zadávaní požiadavkov na generovanie paketov a následnom čakaní na signál dokončenia tohoto generovania.

5.1.3 Generátor transakcií

Generátor transakcií (na obr. 5.1 označený ako Transaction Generator) je najdôležitejším prvkom tejto jednotky. Pomocou signálov `FSM_gen_packet` resp. `FSM_gen_rr_packet`, `FSM_gen_packet_size` a `FSM_gen_packet_cnt` dostáva príkazy na generovanie paketov určitého typu, veľkosti a počtu. Vo vnútri generátoru sa nachádza konečný automat s piatimi stavmi – `SIdle`,`SSof`,`SEof`, `SMiddle` a `SSofeof`. Keďže šírka MFB zbernice (opísanej neskôr v sekcii 4.2), je voliteľná a môže byť menšia ako samotný prenášaný paket, generátor musí byť schopný na túto zbernicu vyslať pakety rôznej dĺžky. Na začiatok je dôležité povedať, že generátor na zbernicu vysiela dáta len vtedy, ak je signál MFB zbernice `TX_DST_RDY` v logickej jednotke, čo značí, že prijímacia strana (v tomto prípade PCIe modul) je schopná dáta prijímať. Samotný automat potom prechádza medzi stavmi podľa toho, koľko už odoslal paketov o zadanej veľkosti, pokým nevygeneruje požadovaný počet. Dáta, ktoré sú odoslané po MFB zbernici, budú po spracovaní PCIe modulom predstavovať payload paketu. Hlavička je odosielaná v stavoch `SSof` alebo `SSofeof` pomocou MVB zbernice. Samotná hlavička a fungovanie MVB zbernice je opísané v sekcii 4.3. Akonáhle generátor dokončí zadané generovanie, upozorní FSM blok o dokončení požiadavky pomocou signálu `FSM_gen_completed`.

5.1.4 Prijímač transakcií

Prijímač transakcií (na obr. 5.1 Transaction Receiver) ma za úlohu prijímať dáta, o ktoré vrámci merania RX smeru zažiadal konečný automat. Využíva k tomu MVB zbernicu, v ktorej je umiestnená hlavička dát prichádzajúcich po MFB zbernici. Z hlavičky následne vyčíta identifikátor transakcie a veľkosť transakcie. Blok je o počte očakávaných dát informovaný signálom z konečného automatu pri začatí merania. Po obdržaní požadovaného počtu dát zastaví počítanie taktov pomocou signálu `reg_clk_stop2`. Blok si uchováva počet obdržaných bajtov vo svojej pamäti. Jeho vnútorný stav a pamäť je vynulovaná signálom z

konečného automatu, pred začiatkom merania. Okrem spracovaní hlavičiek s dátami nijak inak nepracuje a prijaté dáta zahadzuje.

5.1.5 Generátor dát

Generátor dát (na obr. 5.1 označený ako *Data generator*), funguje z veľkej časti ako multiplexor, ktorý je ovládaný pomocou bitu `UP`, umiestneného v kontrolnom registri. Ma za úlohu dodávať dáta generátoru paketov, ktorý ich počas vysielania vkladá na zbernicu MFB. Ak je kontrolný bit `UP` nastavený v logickej jednotke, data generátor rozšíri vzor uložený v pattern registri na celú šírku MFB zbernice. Takto rozšírený vzor je potom výstupom tohoto bloku, v podobe výstupného signálu `pattern_data`, použitý v generátore paketov. V prípade logickej nuly na kontrolnom bite `UP` bude výstupný signál nastavený na samé logické jednotky. Tento blok nie je synchronizovaný s hodinovým signálom, jeho logika je teda čisto kombinačná.

5.1.6 MI32 Access

Blok `MI32 Access` zabezpečuje zápis a čítanie zo stavových a kontrolných registrov pomocou zbernice MI32 opísanej v sekcii 4.4. Skladá sa z viacerých logických obvodov. Každý register obsahuje trojicu signálov (`reg_operandX_we`, `reg_operandX_cs` a `reg_operandX`), ktoré tento blok ovláda. Kombinačný obvod adresového dekodéru uvádza jeden z *chip select* signálov registrov do logickej jednotky, a to podľa adresy zadanej MI32 zbernicou. Každý register si na nábežnej hrane hodinového signálu sleduje svoj *chip select* signál a v prípade, že je spolu s týmto signálom uvedený do logickej jednotky aj signál `MI_WR`, register si uloží hodnotu vloženú do `MI_DWR` signálu. Pri čítaní dát je multiplexorom vložená hodnota v adresou vybranom registri do výstupu `MX_ADC_DO`. V nasledujúcom takte je obsah tohoto signálu prekopírovaný na zbernicu MI32 s príslušnou signalizáciou.

5.1.7 MI ASYNC

Do navrhnutej jednotky bolo nutné zaviesť modul `MI ASYNC`, ktorý je dostupný ako nástroj z NDK [4]. Keďže MI zbernica využíva vlastný hodinový a reset signál, je nutné (v tomto prípade) pomocou asynchrónnej FIFO (first in first out) pamäte vytvoriť prechod medzi hodinovými doménami. Tento modul teda oddeľuje dve hodinové domény, a tak umožňuje naďalej využívať len hodinový signál zariadenia `CLK`.

5.2 Funkcia MFB a MVB zbernice v jednotke

Jednu z dôležitých častí práce zohrávajú zbernice MFB a MVB. Zatiaľ čo MFB zbernica je používaná na prenos payloadu, funkcia zbernice MVB je trochu komplexnejšia. Ako to bolo už spomenuté, práve pomocou MVB zbernice sú prenášané hlavičky jednotlivých paketov. Tieto hlavičky sú prenášané v tzv. DMA formáte. Čo ale ešte nebolo povedané je fakt, že v oboch smeroch sa generické parametre zberníc v cieľovom umiestnení karty líšia. Pri zbernici MFB je tento fakt zanedbateľný, pretože syntax a sémantika prenášaných dát zostáva rovnaká. To isté však neplatí pre MVB zbernicu, ktorej sémantika hlavičiek sa medzi TX a RX smerom zásadne líši.

5.2.1 MVB TX

Na MVB zbernici sa v smere TX (zo zariadenia do systému) vkladajú DMA UP hlavičky. V tabuľke 5.3 je zobrazený formát DMA UP hlavičky, ktorú implementovaná jednotka vypĺňuje.

Bity	Popis
[0:10]	Počet 32b odoslaných slov v pakete
11	Typ transakcie, 0 = čítanie z RAM, 1 = zápis do RAM
[12:13]	Rezervované
14	Povolenie Relaxed Ordering
15	Rozlíšenie PCIe core
[16:23]	Tag transakcie
[24:31]	ID Jednotky
[32:95]	Adresa dat v RAM

Tabuľka 5.3: Formát DMA UP hlavičky

5.2.2 MVB RX

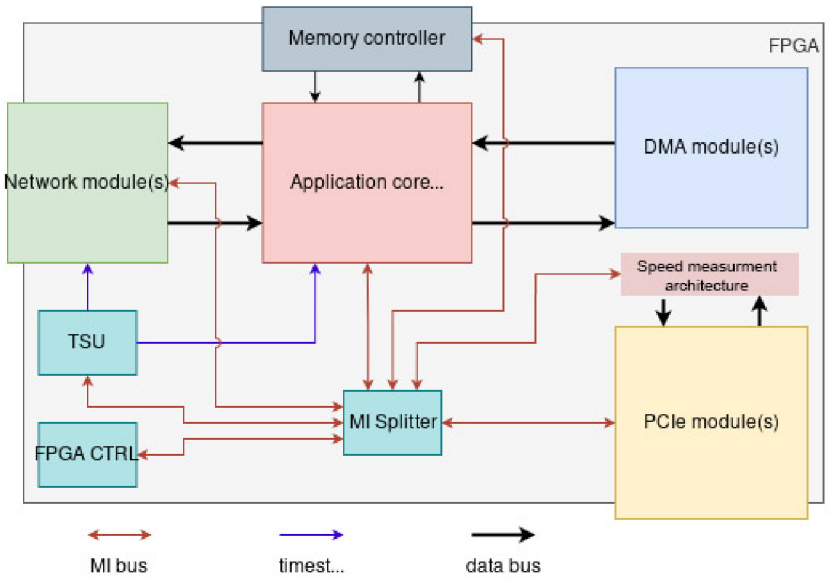
V RX smere (zo systému do jednotky) zariadenie prijíma cez MVB zbernicu DMA DOWN hlavičky. Ich formát je zobrazený v tabuľke 5.4.

Bity	Popis
[0:10]	Počet 32b prijatých slov v pakete
[11:15]	Typ transakcie, 0 = čítanie z RAM, 1 = zápis do RAM
[16:23]	Tag transakcie
[24:31]	ID jednotky

Tabuľka 5.4: Formát DMA DOWN hlavičiek

5.3 Umiestnenie na cieľovej karte

Na obrázku 5.5 je zobrazené cieľové umiestnenie implementovanej jednotky (stred pravého boku obrázka). V tomto mieste má jednotka väčšiu kontrolu nad vytváraním a zasielaním paketov na PCIe interface. DMA modul a aplikačné jadro teda zostane odpojené v tejto architektúre, pretože v tejto konfigurácii nie sú potrebné. Implementovaná jednotka nahrádza a simuluje základné funkcie DMA modulu, vďaka čomu má väčšiu kontrolu nad odoslanými a prijatými dátami, resp. je tým bližšie k PCIe rozhraniu.



Obr. 5.5: NDK architektúra a umiestnenie novej jednotky [4]

5.4 Softvérová aplikácia

Softvérová aplikácia ma za úlohu komunikovať s registrami navrhnutej jednotky pomocou MI zbernice a vypočítať výslednú nameranú rýchlosť. Aplikácia je implementovaná v jazyku C a využíva štandardné knižnice jazyka C a na spracovanie argumentov knižnicu `getopt`. Na implementáciu samotnej komunikácie s akceleračnou kartou bola použitá knižnica `libnfb` z NDK platformy. Táto knižnica poskytuje abstrakciu nad pripojením ku karte resp. k cieľovej komponente a čítania z jej registrov. Po načítaní cieľovej karty a cieľovej komponenty (implementovanej jednotky) z DeviceTree ROM (viď. 4.5.1) je možné vykonávať transakcie cez MI zbernicu pomocou funkcií `nfb_comp_write32(*comp, address, value)` a `nfb_comp_read32(comp, address)`.

5.4.1 Použitie aplikácie

K prekladu aplikácie je určený priložený `Makefile`. Po preložení aplikácie sa vytvorí spustiteľný súbor s názvom `perf-app`. Tento súbor ide spustiť s nasledovnými parametrami:
`./perf-app [-h] -type <RX|TX> -trans-size <4,8192> -trans-count <1,20>`
`(-MPS [0,1,2,3] | -MRRS [0,1,2,3,4,5]) [-up <32bit hex value>]` .

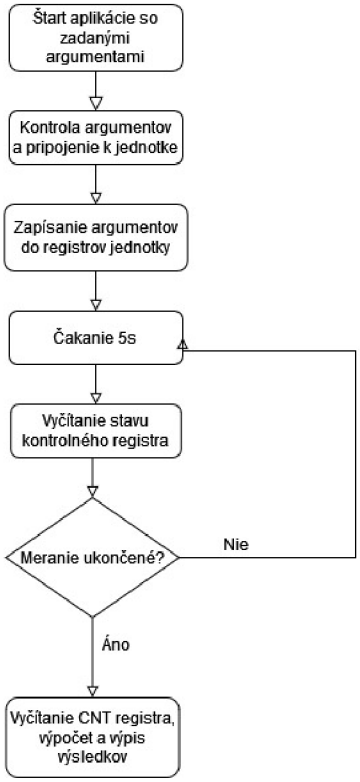
5.4.2 Popis fungovania

Aplikácia najskôr skontroluje všetky argumenty, následne sa pripojí ku cieľovej karte a ďalej k cieľovej komponente. Keď sa toto pripojenie úspešne podarí, zapíše do adresových registrov adresu alokovaného miesta zadanú v argumente. V prípade využitia argumentu –up (use pattern) nasledovaného 32-bitovým hexadecimálnym číslom naplní aj register s patternom. Následne zaplní kontrolný register jednotky, čím odštartuje meranie, zatiaľ čo užívateľovi vypíše hlášku „*Measuring...*“. Po dokončení merania vypočíta priepustnosť zber-

nice pomocou frekvencie FPGA čipu a prečítaním taktov z počítacieho registra. Na obrázku 5.6 je zobrazený vývojový diagram aplikácie.

Vzorec, ktorým aplikácia počíta priepustnosť:

$$\frac{\text{Veľkosť transakcie} \cdot \text{Počet transakcií}}{\text{Počet taktov / Frekvencia FPGA čipu}} = \text{Priepustnosť}\,[B/s]$$



Obr. 5.6: Vývojový diagram aplikácie

5.4.3 Zoznam a popis argumentov aplikácie

V tejto sekcii sú podrobne vypísané argumenty programu:

- `-h --help` vypíše legendu programu
- `-t --type` je **povinný** argument, ktorý je nasledovaný jedným zo smerov merania v podobe dvoch znakov RX (rx), pre smer čítania zo systému, alebo TX (tx), smer pre zápis z karty do systému
- `-s --trans-size` je **povinný** argument nasledovaný veľkosťou transakcie, deliteľnou číslom 4 z intervalu $\langle 4, 8192 \rangle$
- `-c --trans-count` je **povinný** argument nasledovaný počtom opakovaní požadovanej transakcie, tento počet je možné vybrať z intervalu $\langle 1, 20 \rangle$
- `-a --address` je **povinný** argument nasledovaný hexadecimálnou hodnotou adresy, z (do) ktorej môže zaradenie čítať (písať).
- `-p --MPS` je **povinný** argument, ak je vybraný smer merania TX, a je nasledovaný jedným zo štyroch kódov pre maximum payload size **0** (128B), **1** (256B), **2** (512B) alebo **3** (1024B)
- `-r --MRRS` je **povinný** argument, ak je vybraný smer merania RX, a je nasledovaný jedným zo šiestich kódov pre maximum read request size **0** (128B), **1** (256B), **2** (512B), **3** (1024B), **4** (2048B) alebo **5** (4096B)
- `-u --up` je voliteľný argument, ktorý za sebou očakáva 32 bitové číslo v hexadecimálnom tvare, ktoré bude slúžiť ako pattern v prenášaných dátach

5.4.4 Nástroje potrebné k aplikácii

K aplikácií, konkrétne v argumente `-a`, je nutné vedieť, na akej systémovej adrese je alokovaný dostatočne veľký blok pre zápisové a čítacie operácie navrhnutej jednotky. Vrámci tejto práce je táto adresa zabezpečovaná upraveným ovládačom karty, ktorý alokoval požadovanú veľkosť miesta a do logu operačného systému vložil informáciu o adrese, na ktorom tento blok začína. Ovládač bol upravený a dodaný Ing. M. Špinlerom.

Kapitola 6

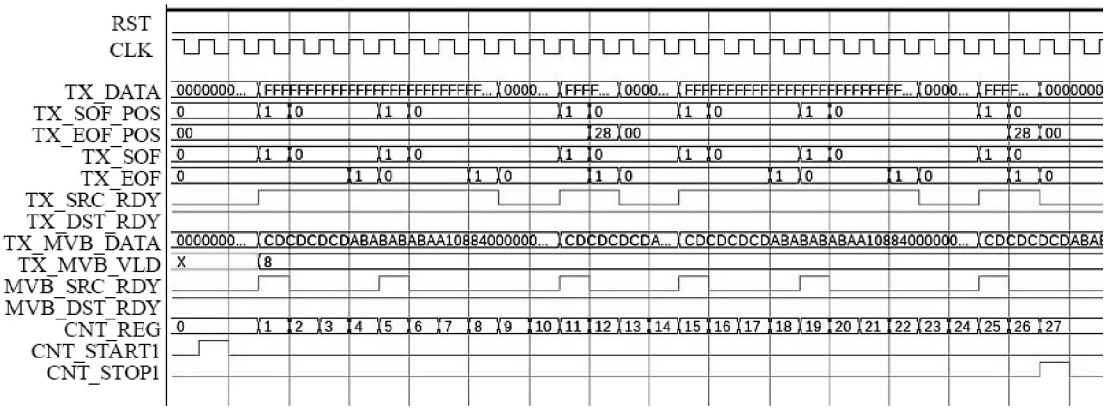
Výsledky a simulácia

V predošlej kapitole bola opísaná implementovaná jednotka. Jednotku bolo najskôr nutné pomocou softvéru ModelSim správne odladiť. Simulácie sú popísané na začiatku tejto kapitoly, v sekciách 6.1 a 6.2. Jednotka bola následne syntetizovaná – prevedená syntéznymi nástrojmi do podoby konfigurácie pre FPGA čip. Avšak, vykonať merania v hardvéri možné nebolo. Zdôvodnenie je opísané v sekcii 6.3.

6.1 Simulácia TX smeru

Nasledujúca simulácia bola vykonaná pri konfigurácií MFB zbernice MFB#(1,1,64,8). To znamená, že šírka MFB zbernice je 512 bitov. Simulačná frekvencia je nastavená na 100 MHz. Do registrov pre adresu bola zapísaná adresa `0xABABABABCDCDCDCD` a do kontrolného registra bola zapísaná hodnota `0x042044B1`. Inými slovami, do kontrolného registra boli zapísané tieto parametre:

- Štartovací bit = 1
- Veľkosť transakcie = 600
- Počet prenosov = 2
- MPS = 1 (256 bajtov)
- MT = 1 (meranie TX smeru)



Obr. 6.1: Simulácia TX smeru

Na obrázku 6.1 s časovým diagramom simulácie je možno vidieť, ako boli transakcie vysielané na MFB zbernicu po nahraní hodnôt do kontrolného registra. Jednotka musela 600 bajtovú zápisovú transakciu rozdeliť podľa parametru MPS, preto bolo treba transakciu rozdeliť a vysielať 2·256 bajtov a dodatočné 1·88 bajtov. Samotné prenášanie bolo nutné taktiež rozdeliť, keďže v simulácií bola použitá konfigurácia zbernice MFB#(1,1,64,8). To znamená, že na zbernicu sa zmesti v jednom takte len 64 bajtov. Preto je nutné jeden 256 bajtový paket vysielať 4 takty. Na tejto simulácií je vidieť aj funkciu stavu `STX_Measurement_rest`, ktorý zadá generátoru vytvoriť zvyšok transakcie $(600 - 2 \cdot 256)$. Táto transakcia sa zopakovala dvakrát, ako bolo zadané. Počítací register `CNT_REG` napočítal 27 taktov. Táto rýchlosť je však dosiahnutá len teoreticky, pretože signál indikujúci voľnosť MFB zbernice (`TX_DST_RDY`) bol celú dobu v stave log. 1.

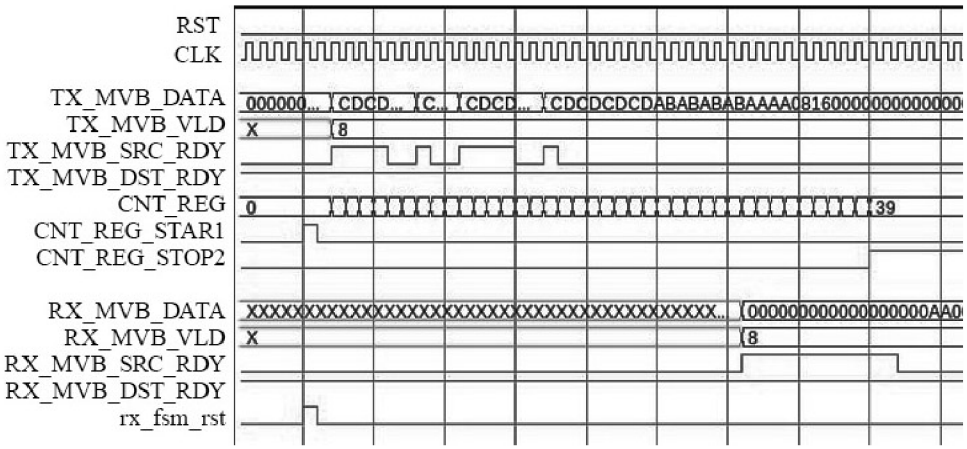
6.2 Simulácia RX smeru

V ďalšej simulácii na obrázku 6.2 je jednotka uvedená do situácie, kedy má čítať 600 bajtov opäť z adresy `0xABABABABCDCDCDCD`. Dáta vložené do kontrolného registra:

- Štartovací bit = 1
- Veľkosť transakcie = 600
- Počet prenosov = 2
- MRRS = 0 (128 bajtov)
- MT = 2 (meranie RX smeru)

V tejto situácií je jednotka opäť uvedená do stavu, kedy má vyčítať až 600 bajtov, ale parameter MRRS (Maximum Read Request Size) je nastavený na 128 bajtov. To znamená, že v jednej „*transakcii*“ musí poslať 4 read requesty na 128 bajtov a jeden na 88 bajtov. Pozor, pojem transakcia v spojení s implementovanou jednotkou v tejto práci značí úlohu vyčítať či poslať zadaný počet bajtov, nejedná sa teda o pomenovanie jednej PCIe transakcie. Jednotka tak vytvorí dokopy 10 požiadaviek na čítanie a čaká na dáta. Ich veľkosť vyčíta z hlavičky a po dodaní všetkých dát zastaví počítanie taktov. Prijímač transakcií je pred každým meraním resetovaný signálom `rx_fsm_rst`. Na obrázku simulácie je teda vidno, že jednotka vyšle za sebou 4 požiadavky na čítanie 128 bajtov a následne jednu požiadavku o čítanie 88 bajtov. Keďže jej boli zadané dve takéto transakcie, túto operáciu zopakuje dvakrát.

Už od momentu vysielania prvej požiadavky medzičasom jednotka čaká odpoveď na takéto požiadavky. V tomto prípade jej prišlo 10 completion paketov až po odoslaní všetkých požiadaviek, každý po 120 bajtoch, pre jednoduchšiu demonštráciu. V reálnom svete je očakávané, že už počas generovania nových požiadaviek môžu prichádzať completion pakety z predošlých požiadaviek. Po prijatí posledného completion paketu – alebo ako nad tým jednotka rozmýšľa $2 \cdot 600$ bajtoch – sa počítanie zastaví.



Obr. 6.2: Simulácia RX smeru

6.3 Integrovanie jednotky do FPGA dizajnu cieľovej karty

Po úspešnom preložení jednotky bol DMA modul z NDK platformy úspešne nahradený touto jednotkou. Syntéza s konečným dizajnom prebehla bez problémov a design tak mohol byť nahratý na cieľovú kartu. V tomto bode sa však začali prejavovať prvé problémy. Po nahratí dizajnu sa cieľový server „zasekol“. Počas hľadania chyby bolo odhalené, že tento problém nenastane v dvoch situáciách. Ak sa k ovládaču nepripojí NFB ovládač, čo však spôsobí, že ovládač nevykoná v operačnej pamäti alokáciu miesta pre potreby DMA prenosov cez PCIe. V druhom prípade problém nenastane, ak sa po nahratí nového dizajnu nespustí rescan PCIe zbernice. Toto však spôsobí, že cieľová jednotka sa nezobrazí na PCIe zbernici.

Problém so zaseknutím servera sa dial aj po nahratí nového (skoro prázdneho) dizajnu, takmer identického s tým funkčným. Ovládač bol zmenený taktiež len minimálne. Jednotka bola zapojená nielen namiesto spomínaného DMA modulu, ale aj do viacero miest cieľového dizajnu, no napriek tomu problém pretrvával.

6.4 Možné vylepšenia jednotky

Ako ďalšie možné vylepšenia nastávajúcej jednotky boli navrhované:

- Na porovnanie predošlých meraní by bolo vhodné v jednotke implementovať pamäť, ktorá by ukladala posledné merania.
- Pripojiť k jednotke signály z hard IP bloku, ktoré zahrnujú informácie o dohodnutom MPS a MRRS.
- Vrámci pridania pamäte by bolo vhodné implementovať priestor na nahrávanie už hotových transakcií, ktoré by mohli byť odoslané.

Kapitola 7

Záver

Cieľom prvej časti práce bolo oboznámiť čitateľa so zbernicovým štandardom PCI Express verzie 3.0. Čitateľ bol oboznámený so základnými vlastnosťami a mechanizmami, ktoré pomáhajú dosiahnuť štandardu vysokú prenosovú rýchlosť. V niektorých prípadoch boli prezentované rozdiely tretej verzie tohoto štandardu od predošlých verzií. Samostatná kapitola bola venovaná parametrom, ktoré ovplyvňujú priepustnosť PCIe zbernice. V teoretickej časti boli predstavené aj zbernicové typy ako MFB, MVB a MI. Tieto poznatky boli potom použité pri implementácii navrhnutej jednotky.

Praktickú časť práce tvoril návrh a implementácia zariadenia, schopného merať priepustnosť PCIe zbernice. Toto zariadenie bolo navrhnuté a opísané pomocou jazyka VHDL, ktorý umožnil jeho integráciu do čipu s technológiou FPGA. Zariadenie bolo navrhnuté a implementované tak, aby bolo vhodné pre NDK platformu vyvíjanou združením CESNET. K tomuto zariadeniu bol taktiež implementovaný softvér v jazyku C, ktorý umožňuje komunikáciu s navrhnutou jednotkou.

Jednotka bola úspešne simulovaná a syntézou prevedená do logického obvodu, ale po nahratí designu na cieľový server sa prejavovalo „zaseknutie“ servera. Napriek všetkej snahe sa tak do termínu odovzdania nepodarilo splniť piaty bod zadania a vykonať merania s navrhnutou jednotkou. Avšak, jednotka je navrhnutá a plne implementovaná spolu so softvérom, ktorý ju ovláda. Pre jej použitie v cieľovej karte je už len potrebné správnym spôsobom odladiť zapojenie do cieľového dizajnu karty, či použiť iný open-source ovládač na nahratie cieľového dizajnu. Tento krok bude naďalej predmetom skúmania, aby riešenie bolo plne použiteľné.

Diskutované vylepšenia tohoto riešenia zahŕňajú umožnenie ukladania niekoľko posledných meraní, pridanie možnosti odoslať niekoľko vlastných transkacií, predom nahratých do pamäte jednotky. Ako komplexné rozšírenie vhodné pre diplomovú prácu, by bolo odstránenie závislosti jednotky od použitia NDK platformy, čo by vyžadovalo aj implementáciu ovládača.

Literatúra

[1] *Protocol Analyzers*. Dostupné z: https://teledynelecroy.com/protocolanalyzer/pci-express/.

[2] Budruk, R. *An Introduction to PCI Express*. Dostupné z: https://www.mindshare.com/files/resources/MindShare_Intro_to_PCIe.pdf.

[3] Budruk, R., Anderson, D. a Shanley, T. *PCI Express System Architecture*. 1. vyd. Addison-Wesley Professional, 2003. ISBN 0321156307.

[4] CESNET z.s.p.o. *Documentation of Minimal NDK Application*. Dostupné z: https://cesnet.github.io/ndk-app-minimal/.

[5] CESNET z.s.p.o. *NFB Software User Guide*. Dostupné z: https://cesnet.github.io/ndk-sw.

[6] CESNET z.s.p.o. *OFM user guide*. Dostupné z: https://cesnet.github.io/ofm/.

[7] Dhawan, S. Introduction to PCI Express-a new high speed serial data bus. In: *IEEE Nuclear Science Symposium Conference Record, 2005*. 2005, sv. 2, s. 687–691. DOI: 10.1109/NSSMIC.2005.1596352. Dostupné z: https://doi.org/10.1109/NSSMIC.2005.1596352.

[8] Farooq, U., Marrakchi, Z. a Mehrez, H. *Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*. 1. vyd. Springer New York, 2012. 7–48 s. ISBN 978-1-4614-3594-5. Dostupné z: https://doi.org/10.1007/978-1-4614-3594-5_2.

[9] Fountain, T., McCarthy, A., Peng, F. et al. PCI express: An overview of PCI express, cabled PCI express and PXI express. In: *10th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems*. 2005. Dostupné z: https://accelconf.web.cern.ch/ica05/proceedings/pdf/I3_001.pdf.

[10] Jackson, M., Budruk, R., Winkles, J. a Anderson, D. *PCI Express Technology 3.0*. Mindshare Press, 2012. ISBN 0977087867.

[11] Koop, M. J., Huang, W., Gopalakrishnan, K. a Panda, D. K. Performance Analysis and Evaluation of PCIe 2.0 and Quad-Data Rate InfiniBand. In: *2008 16th IEEE Symposium on High Performance Interconnects*. 2008, s. 85–92. DOI: 10.1109/HOTI.2008.26. Dostupné z: https://doi.org/10.1109/HOTI.2008.26.

[12] Lawley, J. *Understanding Performance of PCI Express Systems*. 2014. Dostupné z: https://docs.xilinx.com/v/u/en-US/wp350.

[13] NVIDIA Corporation. *MLNX_EN Documentation Rev 5.1-1.0.4.0*. Dostupné z: https://docs.nvidia.com/networking/display/MLNXENv511040.

[14] PCI-SIG, ed. *PCI Express Base Specification Revision 3.0*. PCI-SIG, 2010.

Príloha A

Obsah pamäťového média

1. `xmatej55.pdf` je tento dokument vo formáte PDF
2. `xmatej55_latex.zip` sú zdrojové súbory technickej správy
3. `xmatej55_source.zip` je archív so zdrojovými súbormi tejto práce

Príloha B

Manuál k integrácii zdrojových súborov

Jednotka je vyvíjaná pre kartu s názvom Tivoli v git repozitári `fwbase`, ktorý nie je verejný a je predmetom výskumu v združení CESNET. Na integráciu zdrojových súborov je nutné najskôr získať prístup k tomuto repozitáru a následne aplikovať patch zo zdrojových súborov z archívu `xmatej55_source.zip`.

Patch je možno aplikovať s pomocou softvéru git tak, že súbor `apply_patch.patch` sa prekopíruje do koreňovej zložky repozitára `fwbase` a spustí sa príkaz
`git apply --reject --whitespace=fix apply_patch.patch`.

Po tomto kroku je nutné prekopírovať zložku s aplikáciou (zložku `performance_app`) do repozitára do zložky `fwbase/cards/tivoli`. To znamená, že po prekopírovaní sa súbory aplikácie budú nachádzať v zložke `fwbase/cards/tivoli/performance_app`.