VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

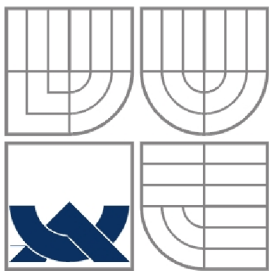
JEDNOTKA PRO ŘÍZENÍ PROTOKOLU PCI EXPRESS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

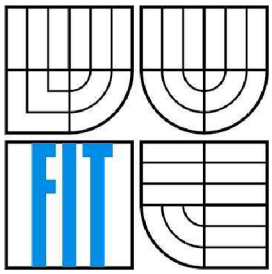
AUTOR PRÁCE
AUTHOR

BC. PAVOL KORČEK

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

JEDNOTKA PRO ŘÍZENÍ PROTOKOLU PCI EXPRESS

PCI EXPRESS BRIDGE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. PAVOL KORČEK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. TOMÁŠ MARTÍNEK

BRNO 2009

Zadání

Jednotka pro řízení protokolu PCI Express

PCI Express Bridge

Vedoucí:

Martínek Tomáš, Ing., UPSY FIT VUT

Oponent:

Kořenek Jan, Ing., UPSY FIT VUT

Přihlášen:

Korček Pavol, Bc.

Zadání:

1. Seznamte se s technologií programovatelných hradlových polí FPGA a dostupnými nástroji pro syntézu a implementaci obvodů do FPGA čipů.
2. Seznamte se s komunikačními protokoly PCI, PCI-X a PCI-Express a způsoby přenosu dat mezi CPU a adaptérem periferního zařízení.
3. Navrhněte řídící jednotku, která bude propojovat interní komponenty uvnitř FPGA čipu s rozhraním PCI Express.
4. Proveďte implementaci navržené jednotky v jazyce VHDL nebo HandelC a její funkčnost ověřte simulací.
5. Funkční prototyp řídicí jednotky a implementujte na kartě COMBO2 nebo ML555. Vhodně ověřte správnou činnost prototypu.
6. V závěru práce diskutujte vlastnosti Vaší implementace a možnosti dalšího pokračování projektu.

Část požadovaná pro obhajobu SP:

Splnění bodů 1-3 zadání.

Kategorie:

Počítačová architektura

Implementační jazyk:

VHDL, HandelC

Operační systém:

Linux

Literatura:

- Budruk Ravi, Anderson Don, Shanley Tom: "PCI Express System Architecture", Mindshare Inc., September, 2003, ISBN 978-0321156303

Abstrakt

Cílem diplomové práce bylo navrhnout a implementovat jednotku pro řízení protokolu sběrnice PCI Express. Jednotka má za úkol výrazným způsobem zjednodušit práci uživatelům – aplikačním inženýrům, kteří pracují na vývoji rozličných akcelerátorů pro čipy FPGA. Navržená jednotka transformuje komplexní rozhraní sběrnice PCI Express a nabízí uživateli obecnější a snadno škálovatelné rozhraní interní sběrnice pro připojení vnitřních komponent čipu. To umožňuje uživateli soustředit se pouze na vývoj cílové aplikace. Jednotka byla implementována v jazyce VHDL, dále byla provedená syntéza do hradlových polí s technologií Virtex-5 a zároveň byla otestovaná přímo na kartách ML555 a COMBOv2. Dosažené výsledky ukazují schopnost pracovat na maximální možné propustnosti, tedy na 7Gb/s.

Abstract

The aim of this thesis was to design and implement PCI Express Bridge. The main purpose of this unit is to help application engineers who develop various FPGA based accelerators. The implemented unit transforms complex PCI Express based system bus interface to more common and scalable interface of internal bus for on-chip components interconnection. This allows engineers to focus on the development of their target applications, not on a complicated communication protocol. The unit was implemented in the VHDL language, synthesized for Virtex-5 based FPGAs as well as completely tested on ML555 and COMBOv2 cards. The acquired results show that the component reaches the throughput of 7 Gb/s, which is the theoretical limitation of underlying protocols.

Klíčová slova

PCI Express, FPGA, Virtex-5, Interní sběrnice, COMBOv2, ML555

Keywords

PCI Express, FPGA, Virtex-5, Internal Bus, COMBOv2, ML555

Citace

Korček, P.: Jednotka pro řízení protokolu PCI Express, diplomová práce,
Brno, FIT VUT v Brně, 2009.

Jednotka pro řízení protokolu PCI Express

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Tomáše Martínka.
Další informace mi poskytli členové týmu projektu Liberouter.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Pavol Korček
1. května 2009

Poděkování

Na tomto místě bych rád poděkoval vedoucímu své diplomové práce Ing.Tomáši Martínkovi za jeho odborné vedení a čas věnovaný konzultacím. Dále bych chtěl poděkovat kolegům z projektu *Liberouter*, jmenovitě Ing. Petru Kobierskému za jeho pomoc a cenné rady.

Obsah

1 Úvod

Poslednou dobou nastáva prudký rozvoj v oblasti informačných technológií. Je to tým, že výpočtová technika začala byť stále viac nasadzovaná do rôznych sfér ľudskej činnosti tak, aby prispela k efektívnemu spracovaniu informácií a pomáhala v riešení najrôznejších problémov. Počítače sa stavajú obecným výpočtovým prostriedkom. S rozširovaním počítačov však súvisí i postupné zvyšovanie výkonnostných požiadavkou, ktoré sú na ne kladené. Obmedzenia založené na platforme PC začínajú v niektorých prípadoch narážať na obmedzenia spôsobené predovšetkým nedostatočným výkonom univerzálneho procesora. Týka sa to napríklad počítačových sietí, počítačovej grafiky ale i rôznych vedeckých výpočtov. Tu sa ponúka riešenie presunúť implementáciu takýchto počítačových systémov celých, alebo v prípade hardwarovej akcelerácie iba ich častí, na úroveň aplikačne špecifických obvodov ASIC alebo programovateľných hradlových polí FPGA. Pre vybrané problémy potom obe z uvedených technológií poskytujú potenciálne vyššiu výkonnosť než obecné procesory.

Technológia ASIC *(Application specific integrated circuit)* predstavuje integrovaný obvod špeciálne navrhnutý pre konkrétnu aplikáciu a jeho štruktúra je napevno vytvorená už pri výrobe. Riešenia založené na tejto technológií umožňujú dosahovať vysokú výkonnosť, majú nízku spotrebu elektrickej energie a pri veľkom počte vyrobených kusov majú i nízku cenu. Na druhú stranu je však pre výrobu takého obvodu nutné navrhnúť a vytvoriť tzv. masku, ktorej cena je pomerne dosť vysoká. Každá zmena návrhu teda prináša ďalšie nezanedbateľné náklady.

Oproti tomu obsahuje FPGA *(Field programmable gate array)* pole programovateľných štruktúr a vstavaných blokov, ktorých konfiguráciu môžme dodatočne i meniť. Vďaka tejto rekonfigurácií (či už plnej alebo parciálnej) ponúka technológia FPGA čipov kompromis medzi výkonom obvodov ASIC a flexibilnosťou univerzálnych procesorov. Uplatnenie takýchto čipov narastá najviac v oblasti komunikácií, no ich podiel na trhu sa stále zvyšuje.

S rastúcou zložitosťou výpočtového sýtemu na ľubovoľnej úrovni sa zvyšuje i nutnosť navrhnúť takýto systém dostatočne modulárne a pokiaľ možno čo najobecnejšie, aby bol dobre spravovateľný a udržiavateľný. Jednotlivé moduly si potom medzi sebou definovaným spôsobom predávajú dáta a musia tak byť navzájom prepojené. Toto platí ale i pre systémy, ktoré sú založené na platforme FPGA. Komunikovať medzi sebou musia jednak komponenty, ktoré sú na čipe samotnom, no v prípade že je FPGA hradlové pole súčasťou vývojového kitu (ML555 [15]), či rozširujúcej karty (COMBOv2 [8]), tak musí byť zaistená i komunikácia s procesorom a obecne s ostanými prvkami počítača. To zabezpečí systémová zbernica, ktorých sa dodnes vystriedalo niekoľko generácií. Posledná, zatiaľ mnohými parametrami neprekonateľná, ponúka vysokú priepustnosť a škálovateľnosť ale pritom i softvérovú podporu starších systémov. Predstaviteľom spomínanej generácie je PCI Express, na ktorej sú založené mnohé dnes dostupné aplikačne

špecifické rozširujúce karty systémových zberníc. Tento typ obecných kariet obsahuje v sebe čipy FPGA, ktorých naprogramovaním zvolíme vlastnú funkciu (funkcie) karty. Navyše výrobcovia čipov dodávajú riešenia v podobe IP blokov, ktoré v sebe zahŕňajú rôznu funkcionalitu. Jedným z takých blokov môže byť i blok nízkoúrovňového riadenia zbernice konkrétneho typu – PCI Express. Využitím týchto blokov môžeme výrazne zjednodušiť vývoj ďalších komponent čipu.

Táto práca sa zaoberá návrhom jednotky riadiacej tok na zbernici PCI Express, ktorá komunikuje jednak s hostiteľským systémom a jednak i s komponentmi na čipe FPGA pomocou internej zbernice. Takáto jednotka výrazne zjednoduší prácu ostatným návrhárom. Tí sa môžu totiž sústrediť už na vývoj svojej konkrétnej aplikácie, a vôbec sa nestarať o problematiku zložitého protokolu zbernice. To zabezpečí práve navrhovaná jednotka. Cieľom je pokryť typické funkčné a výkonnostné požiadavky, čiže zaistiť komunikáciu medzi prvkami umiestnenými na čipe a prvkami, ktoré sú mapované do pamäte hostiteľského systému, a to všetko na maximálnej možnej rýchlosti. Architektúra jednotky je rozčlenená do niekoľkých funkčných podkomponent pre jednoduchší návrh a záverečnú implementáciu. Vďaka tomu bolo možné vytvoriť časť jednotky ktorá je systémovo závislá (na rozhraní systémovej zbernice – PCI Express), a časť ktorá je úplne platformovo nezávislá. Jednotka bola vyvíjaná v rámci projektu *Liberouter,* ktorý je súčasťou výskumného zámeru CESNET – Programovateľný hardware [8]. Celý systém je implementovaný v jazyku VHDL a konkrétnym hardvérovým prostriedkom pre realizáciu návrhu sú karty COMBOv2 a ML555 [8,15] s rozhraním na PCI Express [16].

Práca nadväzuje na semestrálny projekt, ktorý tvoril nevyhnutný teoretický rozbor pred samotným návrhom a implementáciou jednotky. Dokument je logický členený do niekoľkých častí. V úvode je čitateľ zasvätený do základnej problematiky, ktorou sa práca zaoberá. Rozhranie systémovej zbernice, ich generácie a vývoj a taktiež základné princípy prenosu dát na zbernici samotnej nasledujú v kapitole 2. Kapitola 3 pojednáva konkrétne o zbernici typu PCI Express. Tá je v navrhovanej jednotke i požadovanou systémovou zbernicou pre riadenie. Ďalšia z kapitol, kapitola 4, popisuje komerčný blok, ktorý sa bežne dodáva s modernými čipmi FPGA pre nízkoúrovňové riadenie protokolu PCI Express. Kapitola zaoberajúca sa internou zbernicou, je kapitolou popisujúcou druhú stranu jednotky. Stručne zobrazuje jednotlivé podporované typy transakcií na čipe. Napokon nasleduje kapitola návrhu architektúry jednotky, ktorá obsahuje časti venujúce sa samotnej implementácií. V poslednej z kapitol je naznačený systém verifikácií, simulácií a testovania. Na záver sú zhrnuté výsledky práce a ďalšie možnosti pokračovania.

2 Rozhranie systémovej zbernice

Obecne sa výpočtový systém (založený na platforme PC) môže skladať z rôznych typov zberníc – od jednoduchých až po veľmi komplexné. Tieto sú dedikovaných pre rôzne účely. Jednou z najpodstatnejších v celej takejto hierarchií je systémová zbernica. Ak systém túto obsahuje, stáva sa základom celého počítača, pretože jej funkcie sú pre výpočtový systém veľmi dôležité. Bez systémovej zbernice by sme totiž nedokázali k procesoru, ako základnej jednotke výpočtového systému, pripojiť žiadnu ďalšiu externú (klávesnica, monitor, myš, ...) alebo internú (grafická karta, systémová RAM, ...) periférii. To by ale vo výsledku však znamenalo, že okrem toho, že by systém nemal čo počítať, nemal by taktiež možnosť žiadnej interakcie s užívateľom.

Medzi základné funkcie systémovej zbernice patrí prenos informácie medzi prvkami (registre, pamäť), to znamená prenosy dát a adries prvkov, ďalej prenos synchronizácie a jej prípadné využitie pri realizácií prenosov. Okrem tohto by mala systémová zbernica ponúkať prostriedky pre zisťovanie a nastavovanie stavovej logiky klientov zbernice, prostriedky pre generovanie žiadosti o pridelenie zbernice, prostriedky pre možnosť generovania žiadosti o prerušenie a napokon i napr. prostriedky pre testovanie celého systému (nový pohľad uplatnený prvýkrát u zberníc typu PCI).

Fyzický je zbernica realizovaná ako rozvod po systémovej doske počítača a je privedená do konektoru systémovej zbernice, cez ktorý komunikuje s ďalšími komponentmi (radič periférneho zaradenia) – s klientmi. Náročnosť fyzických rozvodov spočíva v obmedzení z hľadiska rýchlosti (paralelne vedené rozvody, fyzická realizácia konektorov, ...) a u starších typov zberníc i v tzv. *clock skew* (posunutie informácie na dátových vodičoch oproti synchronizácií) a pod. [7]. V súvislosti s rapídnym rozvojom systémových zberníc v poslednej dobe hovoríme o tzv. *generáciách systémových zberníc* popísaných v nasledujúcej kapitole.

2.1 Generácie systémových zberníc

2.1.1 1.generácia zberníc

Do prvej generácie systémových zberníc možno zaradiť zbernicu ISA [6]. Ide o paralelnú, 16 bitovú asynchrónnu zbernicu, ktorá je z pohľadu neskorších nástupcov veľmi pomalá (synchronizácia 8 až 10 MHz) so šírkou pásma 10 MB/s. Na zbernici tohto typu prebieha len jednoduchá komunikácia v podobe prenosov dát z prvkov na strane procesoru (pamäť, registre) do iných prvkov (registre fyzický prítomné v iných komponentách, napr. radičoch periférnych zariadení). Už táto prvá generácia mala zabudované mechanizmy na generovanie žiadosti o prerušenie a priamy prístup do pamäte *(Direct Memory Access - DMA)*. Existoval však ale iba obmedzený počet prvkov, ktoré

mohli riadiť zbernicu. Išlo o samotný radič zbernice (po rozpoznaní inštrukcie typu IN/OUT) a radič priameho prístupu do pamäte (po generovaní žiadosti o DMA prenos).

2.1.2 2.generácia zberníc

Štruktúra zbernice ISA a jej mechanické a elektrické vlastnosti nezaznamenali výrazný pokrok, i keď niektoré princípy prechádzali do 2.generácie zberníc. V druhej generácii sa jednalo o zbernice typu EISA, MCA a neskôr boli hlavnými predstaviteľmi tejto generácie zbernice PCI a PCI-X. Opäť išlo o paralelnú, tentokrát už ale synchrónnu zbernicu (PCI, PCI-X), kde prenosové rýchlosti dosahovali rádovo do jednotiek GB/s (viď Tabuľka 2.1 nižšie). Hierarchická štruktúra bola z dôvodov rozdielnych rýchlostí komunikácie tvorená pomocou *North Bridge* a *South Bridge* (na Obrázku 2.1 tmavšou farbou). Zakiaľ čo North Bridge realizuje rýchlu komunikáciu medzi procesorom, systémovou pamäťou a grafickým adaptérom, tak South Bridge sa chová ako samostatné PCI zariadenie, ktoré pripojuje pomalé periférie a realizuje ich radiče (napr. IDE, USB, ...). South Bridge môže obsahovať z dôvodu spätnej kompatibility napr. i bridge na pomalú ISA zbernicu, no dnes sa už toto riešenie skoro vôbec nevyskytuje. Viac informácii napr. i v [6].

Typ	Frekvencia [MHz]	Maximálna priepustnosť [MB/s]	Počet slotov na zbernici
PCI 32-bit	33	133	4-5
PCI 32-bit	66	266	1-2
PCI-X 32 bit	66	266	4
PCI-X 32 bit	133	533	1-2
PCI-X 32 bit	266	1066	1
PCI-X 32 bit	533	2131	1

Tabuľka 2.1: ***Priepustnosť a počet slotov jednotlivých typov zberníc. Pre zbernice šírky 64 bitov je výkonnosť dvojnásobná.***

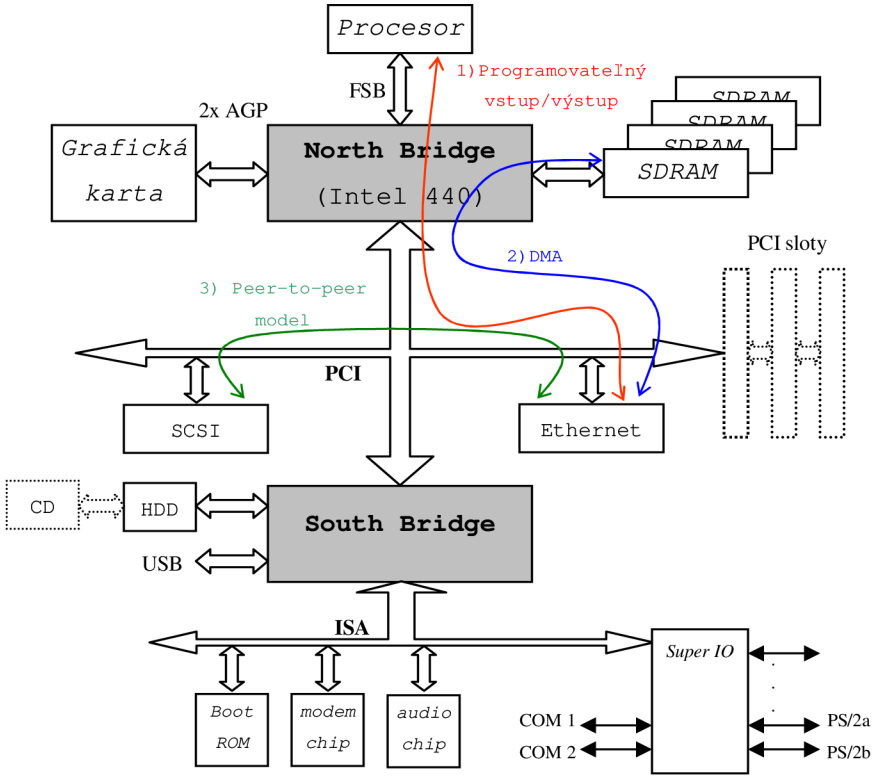
2.1.2.1 Modely prenosu na zberniciach typu PCI

Programovateľný vstup/výstup

Programovateľný vstup/výstup (IO) je jedným z možných spôsobov, ako komunikovať so zariadením na zbernici. Predpokladajme príklad, kedy procesor komunikuje s PCI perifériou akou je napríklad Ethernet sieťový adaptér. Transakcia č.1, ako je vidieť na Obrázku 2.1 (červenou farbou), ktorá je iniciovaná procesorom a jej cieľom je periférne zriadenie je označovaná transakciou cez programovateľný IO. Softvérová inštrukcia spôsobí, že procesor zahájí pamäťový alebo vstupne/výstupný zápisový/čítací zbernicový cyklus na zbernici s adresou odkazujúcou do pamäťového priestoru mapovaného do PCI adresového priestoru. North Bridge začne požadovať prístup k PCI zbernici, a keď vyhrá (súčasne môže žiadať viacero zariadení o prístup), začne generovať príslušný cyklus. Počas prvého cyklu zbernice (fáza adresy, kedy sa na zbernicu vystavuje adresa požadovaného zariadenia), všetky cieľové zariadenia na zbernici dekódujú adresu, ktorú

vystavil iniciátor prenosu (tzv. *master,* konkrétne North Bridge). Len jedno zo zariadení (v našom prípade Ethernet sieťový adaptér) po dekódovaní adresy zistí, že je adresované a potvrdí transakciu. Iniciátor prenosu (a súčasne i zariadenie riadiace prenos – *master,* konkrétne North Bridge) komunikuje s požadovaným cieľom. Dáta sú prenášané v ďalších prenosových cykloch zbernice po adresovom cykle. V každom takte sa prenáša buďto 4 byty alebo 8 bytov dát v závislosti na šírke zbernice PCI (32 vs. 64). Prenos po zbernici nazývame burstovým v tom prípade, pokiaľ nasleduje po sebe niekoľko dátových cyklov. Burstové prenosy sú najefektívnejším spôsobom prenosu.

Pokiaľ máme ku príkladu zbernicu taktovanú na 33 MHz so šírkou 32 bitov (4 byty), maximálna dosiahnuteľná priepustnosť je 4 byty násobené frekvenciou 33 MHz, čo je 133 MB/s (viď Tabuľka 2.1). Maximálna priepustnosť na 64 bitovej zbernici je teda pri danej frekvencii 266 MB/s.



Obrázok 2.1: ***Architektúra systému so zbernicou PCI.***

Samotný dátový prenos zbernice PCI je i tak efektívny iba na približne 50%. Efektívnosť je v tomto prípade definovaná ako stonásobok podielu počtu hodinových taktov zbernice kedy sú dáta prenášané s celkovým počtom hodinových taktov zbernice (arbitráž, adresácia, ...). Strata vo výkone je zapríčinená prázdnym hodinovým taktom zbernice *(idle time),* ďalej určitý čas zaberá prístup na zbernicu *(arbitration time),* čas stratený adresáciou (adresový cyklus zbernice), čakacie stavy počas dátových prenosov, zdržania spôsobené opakovaním transakcií a taktiež v nemalej miere i latencie spôsobené prenosom cez PCI North/South Bridge.

Priamy prístup do pamäte (DMA)

Najviac efektívnym spôsobom prenosu dát medzi PCI zariadením a systémovou RAM je metóda priameho prístupu k pamäti DMA *(Direct Memory Access).* Tento typ prenosu je zobrazený na Obrázku 2.1 (modrou farbou) ako transakcia č.2, kde sa PCI zariadenie stáva *bus master-om,* čiže zariadením schopným riadiť zbernicu PCI. Na základe príkazu z aplikácie (program procesora) alebo na základe hardvéru periférie samotnej, môže PCI zariadenie začať zbernicový cyklus, ktorým prečíta/zapíše dáta z/do systémovej RAM. PCI bus master zariadenie (v našom prípade radič SCSI disku) žiada o pridelenie zbernice, a v prípade získania zbernice zaháji (pamäťový) zbernicový cyklus. V dátovej fáze sú dáta na zbernici prenášané medzi koncovým zariadením a North Bridge. Je to preto, že práve North Bridge sa rozpoznal ako cieľ transakcie pomocou svojej adresy, no ale i preto, že práve North Bridge je prostredníkom medzi RAM a systémovou zbernicou. Preto i on generuje zápisové/čítacie cykly pre zbernicu, na ktorej je pripojená DRAM a komunikuje tak s ňou. Na záver PCI periféria generuje prerušenie, ktorým informuje systémový softvér o tom, že prenos bol ukončený. Takáto bus master alebo DMA metóda prenosu dát je z doposiaľ spomínaných metód najviac efektívna, pretože procesor nie je vôbec zaťažovaný pri komunikácii dvoch zariadení, no a taktiež je pre zápis bloku dát generovaný iba jeden jediný burst cyklus (obecne ľubovoľnej dĺžky). Viac o DMA prenosoch napr. v [2].

Peer-to-Peer model

Posledným zo spôsobov komunikácie zariadení na zbernici PCI je model per-to-peer (rovný s rovným). Je možný práve preto, že na rozdiel od predchodcov zbernice PCI (ISA), sa môže ktorékoľvek zariadenie na zbernici chovať ako bus master (zariadenie schopné riadiť zbernicu). Transakcia takéhoto typu je vyobrazená na Obrázku 2.1 (zelenou farbou) ako transakcia č.3. Ako je vidieť, ide o priamu komunikáciu dvoch zariadení medzi sebou. Iniciátor, ktorý chce komunikovať s iným, cieľovým zariadením, inicializuje prenos, zažiada o zbernicu, a v prípade pridelenia zbernice začína samotnú transakciu adresovým cyklom zbernice. Takto sa teda iniciátor stal obdobne ako v druhom prípade bus master-om. Cieľové PCI zariadenie (SCSI disk) rozpozná svoju adresu na zbernici. Nasleduje dátový cyklus. V čítacom dátovom cykle sú dáta prenášané z cieľového zariadenia (SCSI disk) do iniciátora prenosu (Ethernet sieťový adaptér). Naopak v zápisovom cykle sú dáta prenášané z iniciátora prenosu do cieľového zariadenia.

2.1.3 3.generácia zberníc

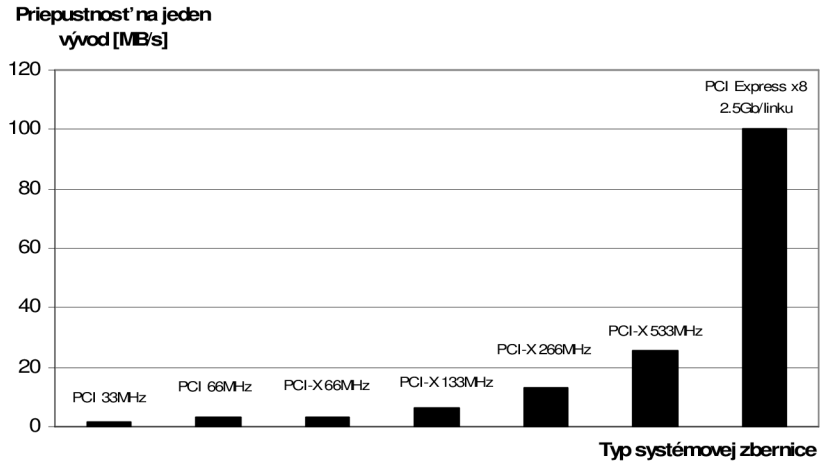
Využívanie novších generácií zberníc je následkom požiadavky na ich vyššiu dátovú priepustnosť. Tá sa doposiaľ dosahovala okrem zvyšovania počtu dátových vodičov i zvyšovaním základnej frekvencie a taktiež realizovaním viacerých prenosov v rámci základnej frekvencie (u PCI-X266 prenos realizovaný 2x, u PCI-X533 prenos realizovaný 4x za jednu periódu základnej frekvencie).

Vysoký počet dátových vodičov spolu s celkom vysokým napájacím napätím viedol ale k prílišnému šumu na základnej doske počítača. Ďalším krokom bolo teda znižovanie napájacieho napätia zberníc, ktoré skracuje dobu prepnutia medzi stavmi aktívnych prvkov. Toto má ale naopak neblahý vplyv na rozširovaní zbernice z pohľadu počtu pripojiteľných zariadení. Napríklad u zberníc typu PCI je z 32 maximálne možných pripojiteľných PCI zariadení možné pripojiť zhruba iba 10 až 12.

Toto všetko tlačilo k zavedeniu celkom nového konceptu. Týmto má byť zbernica založená na princípe, kedy dva koncové body komunikujú už pomocou plne duplexných sériových nízkonapäťových liniek. Predstaviteľom takejto zbernice je i PCI Express [16]. Nejde však o úplne nový koncept, čo sa týka typu prenosov, konfigurácie a inicializácie koncových zariadení a pod. PCI Express zachováva totiž pôvodný koncept PCI, a teda nie je potrebné meniť programové vybavenie počítačov, čo je určite podstatná a veľmi príjemná vlastnosť. Zbernica tohto typu ďalej spĺňa požiadavku rozšíriteľnosti a škálovateľnosti (čo sa priepustnosti týka), efektívnejšieho využitia spoja s mnohonásobne vyššou priepustnosťou (2,5 Gb/s na jedinú linku v jednom smere) oproti základným zberniciam PCI (či PCI-X), efektívnou a vylepšenou kontrolou chýb na viacerých úrovniach a pod. Je však zbernicou natoľko komplexnou, že sa jej venuje ďalšia zo samostatných kapitol (Kapitola 3).

2.1.4 Porovnanie systémových zberníc na báze PCI

Jednou z možností ako porovnávať výkonnosť systémových zberníc je zhodnotenie výkonnosti k počtu využitých vývodov dosky zbernice [1]. Na Obrázku 2.2 je vidieť takýto graf.



Obrázok 2.2: ***Priepustnosť systémových zberníc v závislosti na počte vývodov.***

Prvých sedem zberníc na grafe je typu PCI a PCI-X s 84 vývodmi vždy na jedno zariadenie. Vývody pozostávajú z 46 signálnych a prerušovacích, ďalej z vývodov pre riadenie spotreby, riadenie chýb a zvyšné pozostávajú z vývodov pre napájanie. Poslednou zbernicou v grafe je PCI Express v konfigurácií x8 (viď kapitola 3 o PCI Express). 40 vývodov na tejto zbernici pozostáva 32 signálnych a zvyšok tvorí napájanie.

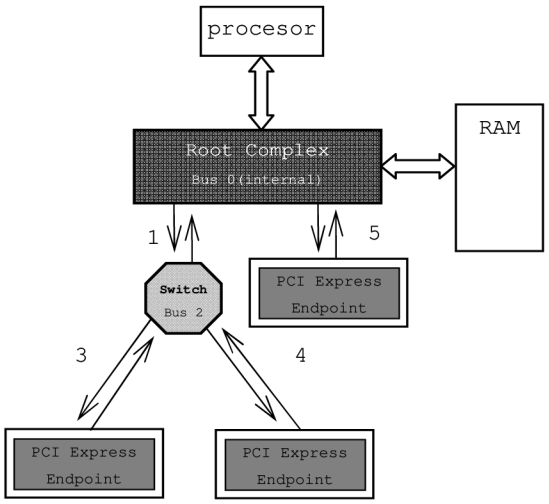
3 PCI Express

PCI Express je v súčasnej dobe poslednou a najmodernejšou zbernicou z rodiny PCI. Oproti predchádzajúcim typom PCI a PCI-X, tato zbernica nie je založená na širokých paralelných zberniciach, ale na vysokorychlostních plne-duplexných sériových linkách. V jeden okamžik je teda možné komunikovať na rýchlosti 2.5 Gb/s, oboma smermi je to teda celkom 5 Gb/s. Tieto plne duplexné linky je navyše možné radiť paralelne vedľa seba a vytvárať postupne zbernicu s vyššou prenosovou kapacitou (škálovateľnosť). Linky sa postupne, podľa toho aké sú široké, označujú *x1, x2, x4, x8, x16 a x32* s priepustnosťou *0.5, 1, 2, 4, 6, 8 a 16 GB/s* (hodnoty sú uvedené už bez réžie kódovania 8/10 – viď Kapitola 3.1.3.1 o fyzickej vrstve).

Všetka komunikácia na zbernici PCI Express prebieha pomocou paketov. Pokiaľ je spoj medzi dvoma zariadeniami tvorený viac ako jednou linkou, sú dáta paketu rovnomerne rozosielané do jednotlivých liniek. V okamžiku keď sa PCI Express zariadenie pripojí do systému, musí sa s druhou stranou komunikačného bodu dohodnúť na spôsobe komunikácie, tzn. počtu liniek, rýchlosti a pod. Z toho dôvodu musí každé PCI Express zariadenie povinne podporovať aspoň komunikáciu po jedinej linke.

3.1.1 Architektúra PCI Express

Základná architektúra systémovej zbernice PCI Express je znázornená na Obrázku 3.1. Architektúra má stromovú topológiu, kde koreňovým uzlom je *Root Complex*. Tento prvok zastáva funkciu North a South Bridge a taktiež prepojuje prvky s požiadavkou na veľmi vysokú priepustnosť ako je procesor a RAM. Prvky v listových uzloch stromu sú označované ako *Endpoint* a reprezentujú jednotlivé koncové zariadenia. Poslednou časťou topológie sú prepínače, tzv. *Switch* komponenty, ktoré tvoria hierarchiu stromu a smerujú transakčné pakety do jednotlivých vetiev.



Obrázok 3.1: ***Architektúra systému so zbernicou PCI Express.***

Každé zariadenie má v rámci celkovej topológie priradenú jednoznačnú identifikáciu rovnakým spôsobom ako majú tiež zariadenia na zbernici PCI a PCI-X. Táto identifikácia je tvorená trojicou číslic:

1. *Bus Number:* udáva číslo zbernice v systému PCI. Zbernicou sa pritom rozumie vždy samostatný segment, ktorý je oddelený cez *Bridge* alebo *Switch*. Zbernice typu PCI podporujú maximálna 256 rôznych zberníc.
2. *Device Number:* udáva číslo zariadenia pripojené v rámci jednej zbernice. Keďže sú na zbernici PCI Express iba point-to-point spoje, tak na každej linke sú iba dve zariadenia s označením 0 a 1.
3. *Function Number:* udáva číslo subsystému v rámci jedného PCI zariadenia. Môžu napríklad existovať zariadenia, ktoré majú dva subsystémy, jeden pre spracovanie obrazu a druhý v podobe sieťového rozhrania. Každé zariadenie môže mať najviac 8 funkcií.

Priradenie tejto jednoznačnej identifikácie prebieha vždy v inicializačnej fáze zbernice po resetu alebo zapnutí systému a samotné číslovanie prebieha smerom do hĺbky stromu zľava. Zbernica číslo 0 je umiestnená vnútri *Root Complex* prvku. Prvá linka zľava je označená ako 1. Vnútri *Switch* komponenty je započítaná vždy jedna virtuálna zbernica, v príklade na Obrázku 3.1 je to zbernica číslo 2. Najľavejšia linka komponenty *Switch* je označená hodnotou 3 atď. až pokiaľ nie je ohodnotená celá topológia.

3.1.2 Typy transakcií na zbernici

Každá transakcia na zbernici PCI Express reprezentuje tok dát zložený z jedného alebo viacerých paketov o maximálnej veľkosti 4kB. Uzol, ktorý transakciu inicializuje nazývame *Requester,* oproti tomu uzol, ktorý transakciu prijíma nazývame *Completer.* Toto názvoslovie je zhodné s PCI-X, kde bol taktiež ako i tu implementovaný model rozdelených transakcií *(tzv. split transaction).* Komunikácia môže prebiehať buďto medzi *Root* a *Endpointom*, alebo medzi dvoma *Endpointmi* navzájom (peer-to-peer). Všetky transakcie je možné rozdeliť do dvoch základných typov:

1. *Posted* – Ide o zápisové transakcie, v ktorých *Completer* nezasiela *Requesteru* žiadne potvrdenie o vykonaní transakcie. Ak nastane prípad kedy by *Completer* nemohol prijať zasielané dáta, *Requester* sa toto bez generovania ďalšej transakcie nedozvie.
2. *Non-posted* – Naopak v tomto prípade ide o transakcie, kedy *Requester* požiada o čítanie alebo i zápis dát a *Completer* mu vždy vráti odpoveď minimálne o stavu vykonanej transakcie – tzv. *Competition transakciu.* Tato transakcia môže ale nemusí obsahovať nejaké dáta. Aby *Requester* mohol rozlíšiť, ktorá odpoveď patrí ktorému požiadavku, je vždy súčasťou požiadavky i odpovede položka s označením *TAG*

(viď Kapitola 3.1.3.3 o transakčnej vrstve). Pokiaľ nastala chyba, bude v odpovedi zaslaný i príznak chyby.

Existuje niekoľko typov transakcii, ktoré sú *Posted* i *Non-Posted* typu. Zoznam všetkých typov transakcií je v Tabuľke 3.1. Oproti predchádzajúcim typom zberníc nám pribudla transakcia typu *Message,* ktorá slúži k zasielaniu rôznych správ. Ich hlavnou výhodou je teda to, že už nie je potrebné mať mnoho riadiacich signálov na zbernici napr. pre signalizáciu prerušenia, pre riadenie spotreby a pod. Namiesto toho je možné jednoducho zaslať správu s príslušnou informáciou.

Transakcie typu Posted sú iba operácie typu zápisu do pamäte a správy typu Message. Ostatné typy operácií ú vždy Non-Posted vrátane IO operácii Write a Configuration Write. U posledne menovaných sa predpokladá, že menia chovanie zariadenia, a preto je požadovaná odpoveď o výsledku operácie.

Názov transakcie	Typ
Memory Read	Non-Posted
Memory Write	Posted
I/O Read	Non-Posted
I/O Write	Non-Posted
Configuration Read	Non-Posted
Configuration Write	Non-Posted
Message	Posted

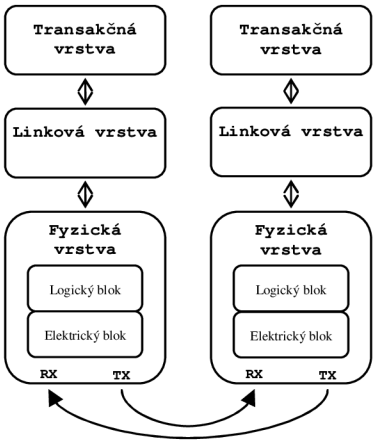
Tabuľka 3.1: ***Typy transakcií PCI Express a ich rozdelenie.***

3.1.3 Vrstvový model

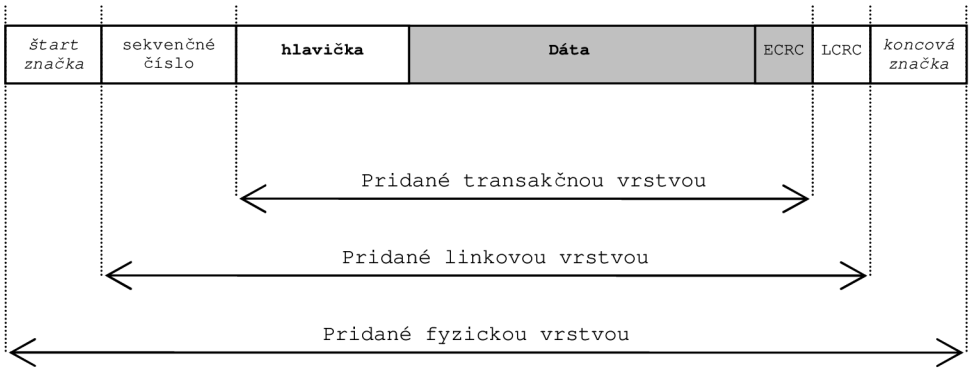
Architektúra každého PCI Express zariadenia je logicky členená do niekoľkých vrstiev podobne ako je tomu napríklad i u zariadení podľa modelu ISO/OSI. Každá z vrstiev je zodpovedná za určitú funkciu a medzi vrstvami je presne definované komunikačne rozhranie [1]. Činnosť každej vrstvy je možné rozdeliť do dvoch osobitných častí (TX – vysielanie a RX – príjem).

Vrstvový model PCI Express (Obrázok 3.2) sa skladá z:

1. *Fyzická vrstva* – realizuje prenos dát po linke na najnižšej úrovni. Zahŕňa digitálnu aj analógovú (elektrickú) časť.
2. *Linková vrstva* – stará sa o spoľahlivý prenos paketov medzi dvoma susednými bodmi zbernice.
3. *Transakčná vrstva* – riadi prenos paketov medzi ľubovoľnými uzlami na najvyššej úrovni.



Obrázok 3.2: ***Vrstvový model PCI Express.***



Obrázok 3.3: ***Paket na zbernici PCI Express (tmavšie časti sú voliteľné).***

3.1.3.1 Fyzická vrstva

Realizuje prenos paketov na fyzickej vrstve (logická i elektrická – analógová časť). Podstatnou funkciou fyzickej vrstvy je i inicializácia a trénovanie linky, ktoré prebieha automatiky bez účasti softvéru alebo jádra PCI zariadenia.

Inicializácia a trénovanie linky

Proces inicializácie linky zahŕňa detekciu počtu liniek, potrebnú na to, aby obe strany vedeli na koľkých linkách môžu a majú vysielať. Ďalej ma inicializácia na starosti detekciu prenosovej rýchlosti (2.5, 5 alebo 10 Gb/s), detekciu polarity diferenciálneho spoja a prípadnú reverzáciu liniek. Ďalej sa nastavuje i tzv. *Bit. lock* a tzv. *Symbol lock* [1] alebo i zamedzenie posunutia fáze medzi jednotlivými linkami *(tzv. Lane-to-Lane De-skew).* Po úspešnom nainicializovaní a natrénovaní linky je možné zahájiť vysielanie/príjem.

TX – vysielanie

Pakety linkovej vrstvy sú po prijatí z linkovej vrstvy uložené do vysielacieho pamäťového buffera. Za pomoci multiplexora je týmto paketom pridaná štartovacia a koncová značka (viď Obrázok 3.3). Tieto značky sú potrebné na to, aby prijímacia strana korektne detekovala začiatok a koniec prijímaného paketu.

Takto označkovaný paket je následne zaslaný do logiky *(byte striping logic),* ktorá multiplexuje jednotlivé byty paketu na samostatné linky. Prvý byte paketu je poslaný cez prvú linku, druhý cez druhú linku atď. cez všetky dostupné linky.

Ďalej nasleduje jednotka *(tzv. Scrambler)*, ktorá využitím algoritmu (jednoduché *LFSR* [4] s polynómom $p(x) = x^{16} + x^5 + x^4 + x^3 + 1$) pseudonáhodne zakóduje každý byte paketu (okrem štartovacej a koncovej značky). Algoritmus eliminuje opakujúce sa značky v bitovom prúde, ktorý tvoria dáta paketu. Opakujúce sa značky totiž znamenajú veľké množstvo energie sústredené v diskrétnych frekvenciách, čo spôsobuje generovanie značného elektromagnetického šumu (EMI). Pomocou uvedeného postupu rozprestrieme energiu cez celé frekvenčné pásmo, čím sa výsledný šum redukuje.

Takto upravené byty (8 bitov) sú následne zakódované do 10 bitov pomocou 8/10 kodéra. Stratíme síce 25% prenosového výkonu, avšak dôvod pre toto kódovanie je vytvorenie dostatočného počtu 1-0 a 0-1 prechodov výstupného dátového prúdu tak, aby prijímajúca strana mohla jednoducho obnoviť hodinový signál z takto prijímaných dát (pomocou *tzv. Phase Lock Loop – PLL,* obvodu fázového závesu). Takýmto spôsobom teda nie je potrebné dedikovať na hodinový signál zbernice osobitný vodič, a je možné sa napr. vyhnúť už i skôr spomínanému posunu hodinového signálu oproti dátam *(tzv. clock skew).*

Napokon sú dáta (vždy po 10 bitoch z výstupu predchádzajúceho kódera) prevedené do sériového dátového prúdu pomocou parelelno-sériového prevodníku (tzv. serializácia). Dátový prúd, časovaný na 2,5 GHz je zaslaný do elektrického sub-bloku fyzickej vrstvy, ktorý vyšle pakety pomocou diferenciálneho signálu.

RX – príjem

Na druhej strane je dátový prúd je prijímaný elektrickým sub-blokom a obvod PLL je synchronizovaný na frekvenciu s akou boli dáta zaslané vysielacou stranou. Prechody v prichádzajúcom prúde sú teda použité na zosynchronizovanie PLL, ktoré slúži na obnovu hodín. Následne je týmito hodinami taktovaný sériovo-paralalelný prevodník, ktorý prevádza 10 vstupných sériových bitov do paralelnej podoby (tzv. paralelizácia).

10 bitové symboly sú ukladané do elastických bufferov. Tieto slúžia na kompenzáciu hodín (obnovené hodiny vs. lokálne hodiny výstupu bufferu).

Symboly sú tentokrát prevedené späť na 8 bitové znaky pomocou 8/10 dekodéra. Tento zároveň sleduje chyby v prijatom 10 bitovom symbole (v prípade objavenia nesprávneho symbolu alebo chýbajúcej značky začiatku/konca). Pri odosielaní vložená štartovacia a koncová značka sú tentokrát odstránené.

Dáta sú následne inverzným algoritmom aký bol vo vysielači *(Scrambler),* prevedené do pôvodnej formy (inicializačná hodnota algoritmu je na oboch stranách FF_h). Byty z každej z liniek sú na záver spätne prevedené do správneho poradia, a v takejto podobe sú vložené do primajúceho buffera linkovej vrstvy.

3.1.3.2 Linková vrstva

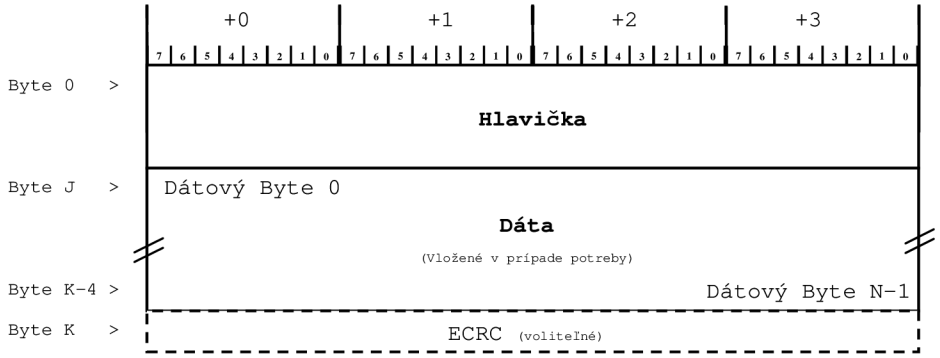
Hlavnou úlohou tejto vrstvy je zaistiť integritu dát prenesených cez jednu linku. Tá sa zaisťuje tým, že k dátam paketu, ktoré prišli z transakčnej vrstvy, pripojuje sekvenčné číslo a nad počíta CRC kód *(tzv. LCRC).* Sekvenčné číslo sa pripája na začiatok a LCRC na koniec paketu ako je znázornené na Obrázku 3.3. Paket je v takejto podobe zaslaný do fyzickej vrstvy, no zároveň je uložený i do tzv. *Reply Bufferu.*

Na druhej strane komunikačného spoja je paket po prijatí skontrolovaný pomocou uloženého LCRC a sekvenčného čísla. Pakety okrem toho, že nesmú byť porušené, nesmú zároveň prichádzať mimo poradia. Pokiaľ je toto v poriadku, vysiela sa odosielateľovi ACK *(Acknowledge)* paket s rovnakým sekvenčným číslom, ako potvrdenie o správnom príjme, a vysielacia strana tak vie, že je možné paket s daným číslom uvoľniť z *Reply Bufferu.* Pokiaľ je však zistená chyba (nesprávne LCRC alebo mimo poradia), odosiela sa NACK *(No Acknowledge)* paket. V tomto prípade vysielacia strana odosiela opätovne paket, tentokrát už z *Reply Bufferu.* Uvedeným spôsobom je možné paket preposielať nanajvýš 4 krát.

Linková vrstva teda okrem bežných dátových paketov generuje i pakety riadiacich informácií *(tzv. DLLP – Data Link Layer Packet)*. Tieto sú výrazne jednoduchšie (neobsahujú adresy) a slúžia pre potvrdzovací protokol *(ACK, NACK),* ďalej pre riadenie toku *(Flow Control* – informácie o dostupnom mieste v pamäťových bufferoch, ktorú si vymieňajú jednotlivé strany medzi sebou) a taktiež pre riadenie spotreby zariadenia *(Power Management).* Tieto pakety sú platné len v rámci jednej linky a teda nie sú smerované cez prepínače. Generujú sa automaticky a nezávisle od softvéru a jádra PCI zariadenia.

3.1.3.3 Transakčná vrstva

Transakčná vrstva sa primárne stará o zabezpečenie prenosu dát v podobe transakčných paketov *(tzv. TLP – Transaction Layer Packet)* medzi jednotlivými PCI zariadeniami. Táto vrstva taktiež zabezpečuje smerovanie TLP cez prepínače na zbernici.



Obrázok 3.4: ***Generické schéma paketu transakčnej vrstvy (TLP).***

Na základe informácií z jádra PCI zariadenia (identifikácia cieľa a typu transakcie, veľkosti dát a pod.) zostavuje hlavičku paketu, ktorá môže nadobúdať veľkosti 3 DW[1] (12 bytov) alebo 4 DW (16 bytov). Formát hlavičky závisí od typu transakcie, avšak obecne definuje vždy typ transakcie, adresu prijímateľa (v rôznej podobe v závislosti od typu), veľkosť dát (ak sú prítomné), atribút udávajúci typ použitého modelu usporiadania (viď ďalej), atribút cache koherencie a traffic class. Ďalej vkladá dáta. Dáta v TLP sú voliteľné a závisia od typu transakcie. Ich veľkosť sa pohybuje teda už od 0 do 1024 DW. Na záver sa môže pridať do TLP voliteľné 1 DW (32 bitov) dlhé CRC *(tzv. ECRC – End-to-End CRC,* niekedy označované ako *TLP digest).* Toto zabezpečenie sa vkladá v prípade vyšších nárokov na spoľahlivosť a pokiaľ má zariadenie povolenú kontrolu ECRC, musí generovať/kontrolovať každý odchádzajúci/prichádzajúci paket. Prepínače túto voliteľnú kontrolu preposielajú bez porušenia. Kontroluje sa až v koncovom uzle. Na Obrázku 3.4 je vidieť generické schéma typického TLP.

Popis položiek hlavičky u generického TLP nasleduje v Tabuľke 3.2 [1], konkrétne typy paketov potom detailne v Tabuľke 3.3. Usporiadanie položiek v hlavičke je navrhnuté tak, aby dôležité informácie boli obsiahnuté hneď na začiatku.

[1] 1 DW – dvojslovo = 4 Byty

Položka hlavičky	Pozícia v hlavičke paketu	Požitie položky
Length [9:0]	Byte 3, Bit 0-7 Byte 2, Bit 0-1	Veľkosť dát TLP v DW. Kódovanie: 00 0000 0001_b = 1DW 00 0000 0010_b = 2DW … 11 1111 1111_b = 1023 DW 00 0000 0000_b = 1024 DW
Attr *(Attributes)*	Byte 5, Bit 4-5	Bit 5 = *Relaxed Ordering* Ak je nastavené, využíva sa model usporiadaných transakcií podľa PCI-X. Ak nastavené nie je, používa sa striktný model usporiadania transakcii podľa PCI. Bit 6 = *No Snoop* Ak je nastavené, systém nie je nútený dodržiavať cache koherenciu. Inak dodržiava prísne model koherencie PCI.
EP *(Poisoned Data)*	Byte 2, Bit 6	Ak je nastavené, indikuje, že dáta v pakete nie sú v poriadku, i keď transakciu možno ukončiť normálne.
TD *(TLP Digest Field* Present)	Byte 2, Bit 7	Ak je nastavené, indikuje prítomnosť tzv. *TLP digest (ECRC).*
TC *(Traffic Class)*	Byte 1, Bit 4-6	Určuje Traffic Class: 000_b = *TC0* (predvolené) ... 111_b = *TC7*
Type [4:0]	Byte 0, Bit 0-4	Spolu s *Fmt* určuje typ transakcie v TLP (viď tabuľka č.4)
Fmt [1:0] *(Format)*	Byte 0, Bit 5-6	Určuje veľkosť hlavičky, informuje či sú súčasťou paketu dáta a doplňuje typ transakcie (viď tabuľka č.4). 00_b = hlavička o veľkosti 3DW, TLP bez dát 01_b = hlavička o veľkosti 4DW, TLP bez dát 10_b = hlavička o veľkosti 3DW,TLP s dátami 11_b = hlavička o veľkosti 4DW,TLP s dátami
First DW **Byte Enable**s	Byte 7, Bit 0-3	Bitová mapa platnosti prvého dátového bytu paketu. Hodnoty: Bit 3 = 1, byte 3 v prvom DW je platný Bit 2 = 1, byte 2 v prvom DW je platný Bit 1 = 1, byte 1 v prvom DW je platný Bit 0 = 1, byte 0 v prvom DW je platný
Last DW **Byte Enable**s	Byte 7, Bit 4-7	Bitová mapa platnosti posledného dátového bytu paketu. Hodnoty: Bit 3 = 1, byte 3 v poslednom DW je platný Bit 2 = 1, byte 2 v poslednom DW je platný Bit 1 = 1, byte 1 v poslednom DW je platný Bit 0 = 1, byte 0 v poslednom DW je platný

Tabuľka 3.2: ***Generické položky hlavičky TLP. (Nastaviť znamená zapísať hodnotu log. 1)***

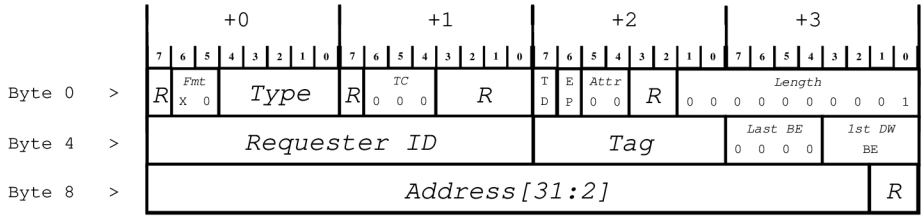
TLP	Fmt[1:0]	Type[4:0]
Memory Read Request (MRd)	00 = 3DW, no data 01 = 4DW, no data	0 0000
Memory Read Lock Request (MRdLk)	00 = 3DW, no data 01 = 4DW, no data	0 0001
Memory Write Request (MWr)	10 = 3DW, w/ data 11 = 4DW, w/ data	0 0000
IO Read Request (IORd)	00 = 3DW, no data	00010
IO Write Request (IOWr)	10 = 3DW, w/ data	0 0010
Config Type 0 Read Request (CfgRd0)	00 = 3DW, no data	0 0100
Config Type 0 Write Request (CfgWr0)	10 = 3DW, w/ data	0 0100
Config Type 1 Read Request (CfgRd1)	00 = 3DW, no data	0 0101
Config Type 1 Write Request (CfgWr1)	10 = 3DW, w/ data	0 0101
Message Request (Msg)	01 = 4DW, no data	1 0rrr*
Message Request W/Data (MsgD)	11 = 4DW, w/ data	1 0rrr*
Completion (Cpl)	00 = 3DW, no data	0 1010
Completion W/Data (CplD)	10 = 3DW, w/ data	0 1010
Completion-Locked (CplLk)	00 = 3DW, no data	0 1011
Completion W/Data (CplDLk)	10 = 3DW, w/ data	0 1011

Tabuľka 3.3: ***Veľkosť hlavičky a typ TLP.***
(* hodnota závisí od použitého smerovania [1]).

IO transakcie

Zatiaľ čo PCI Express špecifikácia neodporúča použitie IO transakcií (z dôvodov neskôr uvedených), možnosť generovať transakcie tohto typu stále existuje pre zariadenia, ktoré uprednostňujú mapovanie do IO priestoru pred mapovaním do globálneho pamäťového priestoru. Zakiaľ čo IO transakcie môžu technicky pristupovať do 32 bitového IO rozsahu, v skutočnosti veľa systémov obmedzuje tento prístup iba na spodných 16 bitov (64 kB).

Formát hlavičky IO paketu transakčnej vrstvy zobrazuje nasledujúci Obrázok 3.5:

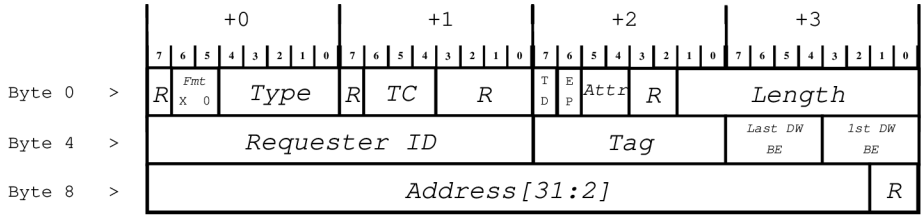


Obrázok 3.5: ***Formát hlavičky IO TLP.***

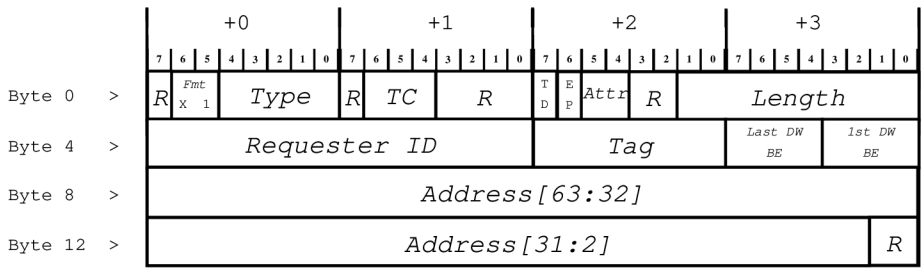
Paket tohto typu má okrem generických hodnôt hlavičky i položku *Requester ID.* Táto jednoznačne určuje *Requestera,* a hodnota môže byť zároveň použitá pri tvorbe *Completion* transakcie. Pozostáva z *Bus Number* (byte 4, bit 0-7), *Device Number* (byte 5, bit 3-7) a *Function Number* (byte 5, bit 0-2). Ďalším z položiek tejto hlavičky je 8 bitový *TAG* (byte 6, bit 0-7). Jeho hodnota jednoznačne identifikuje konkrétnu požiadavku. V ďalších *Non-Posted* transakciách sa používaja pre hodnotu TAGu nasledujúce čísla idúce sekvenčne za sebou. Používané sú však len bity 0 až 4, z čoho plynie, že je možné generovať 32 transakcií v jeden čas. Pokiaľ sa nastaví *tzv. Extended Tag bit* v kontrolnom registri PCI Express zariadenia, tak je možné použiť všetkých 8 bitov (až 256 transakcií). *„R“* označujú rezervované políčka, ktoré musia byť vždy nulové!

Pamäťové transakcie

PCI Expres pamäťové transakcie zahŕňajú dve triedy: *Read Request/Completion* a *Write Request*. Na Obrázku 3.6 a Obrázku 3.7 je vidieť rozdielne použitie týchto transakcií s 3 DW a 4 DW hlavičkou. TLP s 3DW hlavičkou (32 bitová adresa) musí byť použitý striktne pre pamäťové operácie zasahujúce do 4 GB pamäťového priestoru. 4 DW hlavičky sú naopak použité u transakcií presahujúcich 4 GB (je použitá 64 bitová adresa).

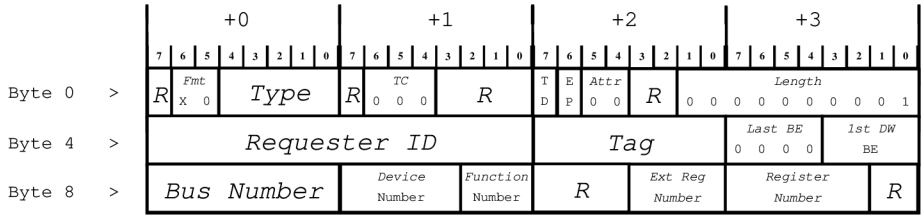


Obrázok 3.6: ***Formát hlavičky pamäťovej 3DW transakcie (do 4 GB).***



Obrázok 3.7: ***Formát hlavičky pamäťovej 4DW transakcie (nad 4 GB).***

Konfiguračné transakcie

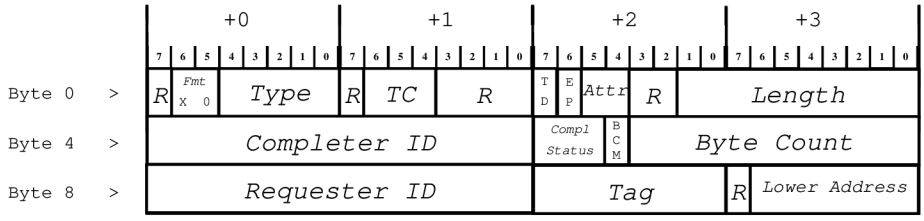


Obrázok 3.8: ***Formát hlavičky konfiguračnej transakcie.***

Konfiguračná transakcie obsahujú identifikáciu *Completera* opäť v podobe jeho ID, zloženého z *Bus Number* (byte 8, bit 0-7), *Device Number* (byte 9, bit 3-7) a *Function Number* (byte 9, bit 0-2). Ďalšími položkami hlavičky tejto transakcie je *Register Number* (byte 11, bit 2-7), ktorý udáva spodných 6 bitov ofsetu do konfiguračného priestoru PCI. Používa sa v spojení s *Ext Register Number* (byte 10, bit 0-3), ktorý udáva naopak horné 4 bity ofsetu do konfiguračného priestoru (pre plných 10 bitov adresy do konfiguračného priestoru PCI Express, inak nastavený na nuly).

Competition transakcie

Mnoho polí hlavičky TLP v transakcii typu *Competition* musí mať rovnaké hodnoty ako mala požiadavka. Sú to položky *Traffic Class, Attributes* a pôvodné *Requester ID,* ktoré je použité pre smerovanie odpovede. Formát hlavičky paketu je na Obrázku 3.9:



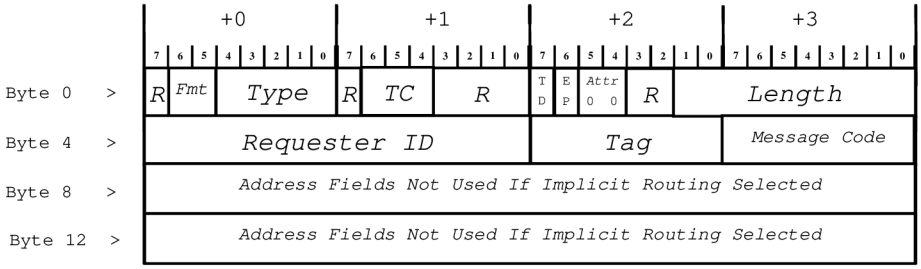
Obrázok 3.9: ***Formát hlavičky Competition transakcie.***

Ako je na tomto obrázku vidieť, pribudlo mnoho doposiaľ nedefinovaných položiek. Prvou z nich je 3 bitový *Compl. Status* (byte 6, bit 5-7), ktorý informuje o stave výsledku žiadosti. Jedinou správnou návratovou hodnotou je 000_b *(SC - Successful Completion).* Pre ostatné kódy viď napr. [1]. Ďalšou novou položkou je *Byte Count* (byte 7, bit 0-7 a byte 6, bit 0-3), definujúci počet zostávajúcich bytov do dokončenia celej odpovede na požiadavku. Odpoveď na ľubovoľnú *Non-Posted* transakciu totiž nemusí prísť v jednej samostatnej transakcii. Políčko *BCM* (byte 6, bit 4) je nastavované iba od PCI-X zariadení a indikuje, že predchádzajúca položka počtu zostávajúcich bytov reflektuje prvé prijaté dáta, a nie počet zostávajúcich, ešte neprenesených bytov transakcie. Posledným ešte nespomenutým políčkom je *Lower Address* (byte 11, bit 0-6), ktoré udáva

spodných 7 bitov adresy dát navrátených čítaním. Hodnota je používaná pre zistenie adresy ďalšej odpovede.

Message transakcie

Transakcie typu Message nahradili v novej PCI Express zbernici mnoho z prerušovacích, chybových a signálov riadenia spotreby u skorších zberníc. Všetky transakcie tohto typu používajú 4 DW formát hlavičky. Ako bude vysvetlené, smerované sú na základe adresy, ID alebo implicitne. Od toho sa odvíja i význam jednotlivých položiek zobrazených na Obrázku 3.10:



Obrázok 3.10: ***Formát hlavičky transakcie typu Message.***

Spodné 3 bity políčka typu paketu definuje spôsob smerovania. Toto môže byť smerovanie do *Root Complex* (000_b), smerovanie na základe adresy (001_b), smerovanie podľa ID (010_b), *Broadcast* (011_b), lokálna správa končiaca u prijímača (100_b) alebo zber a smerovanie k *Root Complexu* (101_b). Ostatné hodnoty sú rezervované pre budúce využitie. Je jasné, že položky adresy sú využité iba u smerovania na základe adresy.

Úplne novým políčkom tu je 8 bitové *Message Code* (byte 7, bit 0-7). Existujú správy typu *Unlock Message* ($0000\ 0000_b$), *Power Mgmt Message* ($0001\ xxxx_b$), *INTx Message* ($0010\ 0xxxb$) definujíce prerušenia ($0010\ 0000_b$ vyvolá INTa, $0010\ 0001_b$ vyvolá INTb apod.), *Error Message* ($0011\ 00xx_b$) informujúce o chybách, *Hot Plug Message* ($0100\ xxxx_b$), *Slot Power Message* ($0101\ 0000_b$), *Vendor Type 0 Message* ($0111\ 111x_b$) a *Vendor Type 1 Message* ($0111\ 1111_b$). Podrobnejší popis je potom možné nájsť i v [1,5].

Ďalšie funkcie transakčnej vrstvy

Riadenie toku (Flow Control)

Flow control je mechanizmus akým si dve susedné PCI zariadenia vymieňajú informácie o tom, koľko majú vo svojich pamäťových buferoch miesta pre príjem nového paketu. [5]. Vysielacia strana nesmie nikdy zaslať paket pokiaľ primajúca strana nemá dostatok miesta v svojich vstupných bufferoch, poprípade pokiaľ si tuto informáciu uzly ešte nevymenili. Ako bolo spomenuté, informácie o veľkosti týchto bufferov si koncové body zasielajú pomocou špeciálneho typu paketu na linkovej vrstve (DLLP). Dôvodom, prečo je táto funkcionalita umiestnená na transakčnej vrstve súvisí s tým,

že buffery pre uloženie paketu sú umiestnené práve na tejto vrstve a veľmi úzko súvisí s ďalšou dôležitou funkcionalitou transakčnej vrstvy popísanej v nasledujúcej kapitole.

Zaistenie kvality služieb (QoS)

Zaistenie kvality služieb znamená schopnosť systému riadiť prioritu prenosu v rámci jednotlivých služieb [5]. Toto je ďalšou noviniek, z pomedzi mnohých už spomínaných, na poli systémových zberníc.

Zbernica PCI Express má mechanizmus zaistenia priority už integrovaný. Ako bolo už spomenuté, každý prenášaný paket obsahuje položku *Traffic Class (TC)*, ktorá identifikuje jeho prioritu. Táto položka nadobúda hodnôt *TC0* až *TC7*, kde posledná hodnota označuje najvyššiu prioritu. Riadenie priorít sa však komplikuje zavedením tzv. *virtuálnych kanálov (VC – Virtual Channels).* Skutočná komunikácia cez fyzickú linku totiž prebieha cez tieto kanály, ktorých môže byť opäť až osem *(VC0 – VC7)*. Práve tieto kanály majú na transakčnej vrstve umiestnený každá svoj vlastný buffer. Zariadenie však nemusí podporovať všetkých osem kanálov (z dôvodu zníženia ceny realizácie môžu obsahovať napr. štyri kanály).

Pretože sa pakety pred prenosom zhromažďujú vždy do bufferov jednotlivých virtuálnych kanálov, je priorita riadená práve podľa týchto virtuálnych kanálov. Pred samotným uložením paketu do bufferu sa preto vykonáva mapovanie čísla TC na VC. Najjednoduchšie je samozrejme mapovanie 1:1. Toto ale nie je vždy dosiahnuteľné. Pokiaľ však PCI Express Endpoint zariadenie podporuje napríklad len 4 virtuálne kanály, je možné očakávať, že pakety s TC0 a TC1 budú mapované do VC0, TC2 a TC3 do VC1 atď. Napokon je dobré podotknúť to, že u každého zariadenia sa môže počet virtuálnych kanálov líšiť. Taktiež z dôvodu podpory všetkými zariadení je mapovanie TC0 na VC0 pevné a nemožno ho meniť.

U komponenty typu prepínač je riadenie priority o niečo zložitejšie. Priorita sa riadi ako aj na základe virtuálneho kanálu, tak i s ohľadom na číslo portu, z ktorého paket prichádza. Riadenie priority s ohľadom na porty je dôležité najmä preto, aby nedochádzalo k uprednostňovaniu prenosu z jedného portu a naopak dochádzalo tak k starnutiu komunikácie na porte druhom. V koncovom zariadení i v prepínači je potom možné vybrať spôsob, akým bude priorita vyhodnotená. Je možné použiť napríklad *prioritu statickú, Round-Robin, váhovaný Round-Robin* alebo časovaný *Round-Robin.*

Napokon *Requester* (Endpoint) môže generovať požiadavky iba s TC0 [1]. Pokiaľ by sa tak nestalo, môže byť paket označený ako poškodený a odfiltrovaný. Naopak *Completer* musí byť schopný akceptovať i požiadavky s TC iným ako je TC0. Odpoveď od *Completera* musí mať rovnaké TC ako bolo v požiadavke.

3.1.4 Smerovanie na zbernici

Aby jednotlivé pakety vždy dorazili do požadovaného koncového uzlu na zbernici, je potrebné v priebehu ich cesty vykonávať smerovanie tak, ako je tomu napr. i u Internetu. Funkcia smerovania je hlavnou úlohou prepínača (alebo i *Bridgu* pokiaľ je v architektúre umiestnený).

Zbernica PCI Express poskytuje tri typy smerovania na základe:

1. *Adresy* – v pakete je zadaná adresa (64 alebo 32 bitová – viď predchádzajúca Kapitola 3.1.3.3 o transakčnej vrstve) do globálneho pamäťového priestoru.
2. *Identifikácie* – zadaná je identifikácia PCI zariadenia *(Bus Number, Device Number, Function Number)*
3. *Implicitne* – v pakete je uložená špeciálna značka, ktorá jasne definuje cieľ transakcie.

Typ transakcie	Typ smerovania
Memory Read/Write	Adresa
I/O Read/Write	Adresa
Configuration Read/Write	Identifikácia
Completion	Identifikácia
Message	Adresa, identifikácia alebo implicitne

Tabuľka 3.4: ***Rozdelenie transakcii podľa typu smerovania. (Pre typ transakcie viď. kapitola o formáte TLP).***

Smerovanie na základe adresy je kompatibilné s PCI a PCI-X. Prepínače v prípade príchodu paketu najprv zistia, či nie je paket určený pre samotný prepínač. Ak nie, a paket prišiel primárnym rozhraním, tak skontrolujú konfiguráciu jednotlivých svojich sekundárnych portov, a podľa toho preposielajú daný paket na daný port alebo (v prípade ak by adresa nespadala do rozsahu) paket obslúžia ako nepodporovanú požiadavku. Pokiaľ prichádza paket z jedného zo sekundárnych rozhraní a adresa nespadá do žiadneho z rozsahov ostatných sekundárnych portov, potom je paket smerovaný na primárny port. Koncové zariadenie príjme paket a porovná adresu zo všetkými svojimi bázovými adresami vrátane kontroly jej veľkosti. Ak vyhovuje zariadeniu paket spracuje, inak paket zahodí.

Smerovanie na základe identifikácie využíva identifikáciu zariadení PCI. Pokiaľ sa identifikácia zhoduje, je paket pre dané zariadenie, inak sa paket odstráni. Prepínače kontrolujú, či paket nespadá do rozsahu niektorého z portov. Obslúženie je potom podobné smerovaniu na základe adresy. Identifikáciu si každé zo zariadenie alebo i prepínač (ten má navyše i rozsahy) pamätá, i keď nie je súčasťou jeho konfiguračnej štruktúry. Po resete alebo zapnutí systému sa identifikácie obnovuje pri inicializačnej fáze zbernice (viď Kapitola 3.1.1 o architektúre zbernice).

Implicitné smerovanie je špeciálnym typom smerovania využívaných správami typu *Message*. Ide o spôsob ako je možné komunikovať smerom k susednému uzlu, smerom ku koncovému uzlu, ku koreňovému uzlu *(Root Complex)* alebo zasielať všesmerové správy typu

Broadcast. Koncové zariadenia pritom dokážu spracovávať len správy tohto typu *(Broadcast)* a správy určené pre prijímajúci uzol. Prepínače samozrejme správy, ktoré sú smerované implicitne smeruje tak, že *Broadcast* rozosiela iba na sekundárne porty (kvôli obmedzeniu zahltenia zbernice) a správy pre *Root Complex* zas na primárny port. Ostatné správy nespracováva a teda ani nikam nesmeruje.

3.1.5 Inicializácia a konfigurácia zaradení

Zbernica PCI Express zachováva rovnakú štruktúru pamäťového priestoru ako zbernica PCI alebo PCI-X. Rozlišuje globálny adresový priestor o veľkosti 4 GB alebo 16 EB. Prvá časť pamäti je fyzická RAM umiestnená v systému. Nad touto pamäťou sú mapované pamäťové priestory periférnych zariadení, ktorá sa delí na *Prefetchable* a *Non-Prefetchable* [5]. PCI Express podporuje taktiež z dôvodu spätnej kompatibility aj IO pamäťový priestor. Tento sa však už neodporúča používať, pretože je možné ho nahradiť mapovaním pamäti do globálneho pamäťového priestoru.

U PCI Express je taktiež podporovaný konfiguračný pamäťový priestor, ktorý je plne kompatibilný s PCI a PCI-X. Každé zariadenie má tak alokovanú preddefinovanú dátovú štruktúru o veľkosti 256 bajtov pre každú svoju funkciu. U zariadení PCI Express je tento priestor rozšírený až na 4 kB o ďalšie možnosti (viď Obrázok 3.11). Týmito sú napríklad i rozšírená možnosť chybových hlásení, správa virtuálnych kanálov, ovládanie riadenia spotreby a iné [1,6,5]. Prístup do takéhoto konfiguračného priestoru bol u predchádzajúcej technológie PCI realizovaný pomocou dvoch registrov v IO priestore. Ako bolo ale spomenuté, v súčasných systémoch je i tento priestor mapovaný do globálneho adresového priestoru systému.

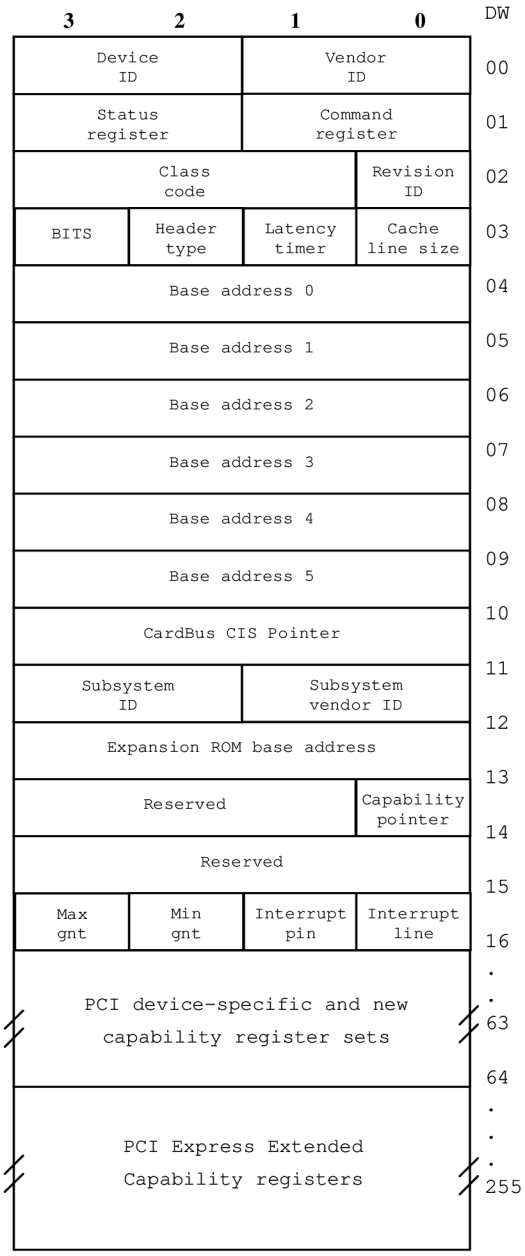
Z pohľadu konfiguračnej štruktúry o veľkosti *256 bajtov* sú rozlišované dva základné typy:

1. *Typ 0* – je určený pre koncové uzly a ďalej sa s ním budeme zaoberať,
2. *Typ 1* – je určený pre komponenty typu *Switch* (alebo *Bridge*). Štruktúra tohto typu je vyhradená pre každý samostatný port, a hlavným rozdielom je to, že obsahuje najmä limity adries na ktoré bude smerovať pakety [1,5,16].

3.1.5.1 Konfigurácia typu 0

Konfiguračná štruktúra typu 0 je určená pre koncové zariadenia na zberniciach typu PCI. Jej štruktúra zodpovedá vrchnej časti na Obrázku 3.11 (DW 0 až DW 15). Najvýznamnejšie položky sú:

- *Vendor ID* – identifikačné číslo výrobcu. Každý výrobca musí požiadať združenie PCI-SIG [17] o pridelenie jednoznačného identifikátoru v rámci všetkých výrobcov.
- *Device ID* – identifikácia PCI zariadenia. Jedná sa o identifikáciu, ktorú si pre jednotlivé typy zariadenia udáva sám výrobca.
- *Revision ID* – identifikácia revízie. každý výrobca si môže odlíšiť rôzne verzie zariadenia.



Obrázok 3.11: ***PCI Express konfiguračná štruktúra typu 0.***

- *Class Code* – označuje skupinu, do ktorej je zariadenie zaradené, napr. multimediálne, sieťové, wifi zariadenie a pod. Typy týchto skupín sú definované centrálne.
- *Subsystem Vendor ID* a *Subsystem ID* – obsahuje identifikáciu prípadného podsystému v rámci daného PCI zariadenia.
- *Base Address (BAR)* – jedná sa o šesticu bázových registrov, pomocou ktorých PCI zariadenie špecifikuje požiadavky na globálny alebo IO pamäťový priestor. Na základe týchto registrov prebieha proces alokácie pamäti pre PCI zariadenie. Registre môžu byť použité ako 6x bázový register pre 32 bitový adresový priestor, alebo 3x bázový register pre 64 bitový adresový priestor. Alebo môžu taktiež tvoriť prípustnú kombináciu oboch typov.

Postup alokácie pamäti na PCI

Každé PCI zariadenie, ktoré sprístupňuje určitú internú pamäť alebo riadiace registre, musí cez BAR registre v svojej konfiguračnej štruktúre, požiadať o pridelenie potrebného pamäťového priestoru v rámci celého systému. Výrobca PCI zariadenia špecifikuje svoj požiadavku na pamäť tak, že napevno nastaví spodné bity registra BAR [5]. Najnižšie bity vyjadrujú typ požiadavku, ďalej nasleduje oblasť, v ktorej je špecifikovaná veľkosť požadovanej pamäti. Posledná časť BAR registra zostáva nenastavená a slúži pre zápis adresy prideleného priestoru zo strany systémového softvéru.

Postup pri obsluhe požiadavku na alokáciu prebieha v nasledujúcich krokoch:

1. Systémový softvér najprv zapíše do neinicializovaného BAR registra samé jedničky. Bity, ktoré sú napevno nastavené výrobcom, nebudú samozrejme týmto zápisom ovplyvnené.
2. Systémový softvér prečíta hodnotu BAR registra, a zistí tak požadovaný typ a množstvo pamäte, ktoré adaptér potrebuje. Po získaní všetkých požiadavkou i od ostatných zariadení napokon rozdelí pamäťový priestor.
3. V poslednom kroky vykoná systémový softvér zápis hodnoty do BAR registra, ktorý odpovedá bázovej adrese prideleného priestoru.

4 Využitie Endpoint Block Plus

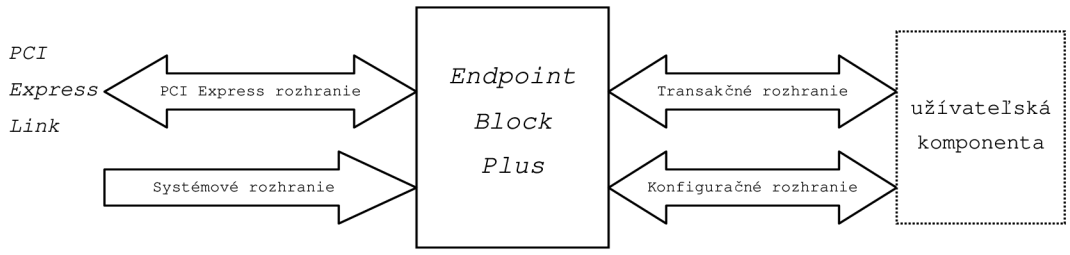
Ako bolo podrobne vysvetlené v predošlej kapitole, dva koncové body PCI Express zbernice komunikujú medzi sebou pomocou sériovej paketovej komunikácie. Sériové rozhranie je v takejto podobe privedené i na samotné vývody čipu FPGA. Aby sa však hardvérovým vývojárom pracujúcim s rozhraním systémovej zbernice priamo na tomto čipe zjednodušilo spracovanie dát v tejto podobe, ponúka Xilinx, ktorý je taktiež i výrobcom cieľových FPGA, nadstavbu nad samotným PCI Express rozhraním v podobe *Endpoint Block Plus* (v1.6) [12].

Endpoint Block Plus (ďalej ako *endpoint blok*) pre PCI Express je samostatný výstavbový blok (tzv. LogicCORE™ IP soft-core), ktorý interne inštancuje vstavaný integrovaný blok rozhrania PCI Express (tzv. *hard-core*) v moderných FPGA čipoch rodiny Virtex-5 (LXT/SXT/FXT) [14]. Medzi najväčšie výhody tohto vzájomného spojenia hard-core a soft-core patrí, že nám poskytuje nasledujúce:

- vysoko priepustný, flexibilný a rozšíriteľný IO blok kompatibilný
    - s *PCI Express Base Specification v1.1 [11]*
    - s konvenčným PCI softvérovým modelom
- PCI konfiguračný priestor typu 0 s rozšíreniami pre PCI Express [1,4]
- garanciu časovania (Xilinx Smart-IP™) [14]
- vyžitie RocketIO GTP (Virtex-5 LXT/SXT) alebo RocketIO GTX (Virtex-5 FXT) [14] zabezpečujúce
    - rýchlosť jednej linky 2,5Gbps
    - podporu pre x1, x4 alebo x8 konfiguráciu
    - elastické bufery
    - automatickú obnovu hodín
- dekódovanie a kódovanie 8b/10b
- podporu pre ošetrenie poškodených paketov a zotavenie po chybách
- kompatibilitu s PCI/PCI Express systémom riadenia spotreby
- podporu pre maximálne *512 bytov* v jednej transakcii
- plnú podporu s pravidlami usporiadania transakcií podľa PCI Express špecifikácie
- štandardizované užívateľské rozhranie, ponúkajúce
    - paketovo založený protokol
    - full-duplex prenos
    - flow control na vysielajúcej (Rx) i prijímajúcej (Tx) strane

Najmä posledná kategória podpory štandardného užívateľského rozhrania je z pohľadu vývojára veľmi zaujímavou, pretože poskytuje jednoduchšie rozhranie transakčnej vrstvy protokolu zbernice

PCI Express. Preto sa i popisu tohto rozhrania venuje niekoľko ďalších kapitol, ktoré prehľadne zhrňujú jednotlivé samostatné časti. Obrázok 4.1 názorne ukazuje zapojenie jednotlivých rozhraní endpoint bloku. Užívateľská komponenta využíva konfiguračné a transakčné rozhranie, PCI Expres rozhranie je napevno pripojené na čipe a systémové rozhranie je potrebné zapojiť podľa špecifikácie (viď ďalej).



Obrázok 4.1: ***Endpoint Block Plus so všetkými rozhraniami.***

4.1 Systémové a PCI Express rozhranie

Systémové rozhranie pozostáva z dvoch signálov. Prvým z nich je systémový reset *(sys_reset_n).* Ide o asynchrónny reset aktívny v nízkej úrovni a jeho aktivácia spôsobí tvrdý reset celého endpoint bloku spolu s RocketIO GTP (alebo RocketIO GTX) vysielačmi/prijímačmi. Ďalším zo skupiny systémového rozhrania je signál hodín *(sys_clk)*. Tieto hodiny sú použité po tvrdom resete na časovanie logiky endpoint bloku, no zároveň i pre RocketIO. Hodiny nesmú byť však výstupom z DCM. Viac informácií je možné nájsť napr. i v [12].

PCI Express rozhranie u endpoint bloku pozostáva z diferenciálnych vysielacích a prijímacích párov zoskupovaných do spoločných signálov, liniek pripojených priamo na systémovú zbernicu. Každá linka sa skladá teda z páru vysielacích diferenciálnych signálov *(pci_exp_txp, pci_exp_txn)* a páru prijímacích diferenciálnych signálov *(pci_exp_rxp, pci_exp_rxn).* Endpoint blok pre 1 linku (x1) má vo výsledku len linku č.0, ďalej endpoint blok pre 4 linky (x4) má linku č.0 až linku č.3 a na koniec konfigurácia endpoint bloku pre 8 liniek (x8) podporuje linky č.0 až linku č.7.

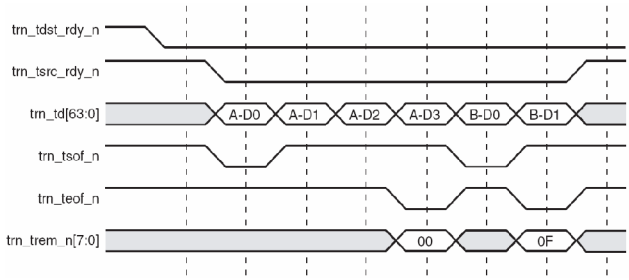
4.2 Konfiguračné rozhranie

Konfiguračné rozhranie nám poskytuje spôsob ako získať informácie o stave endpoint bloku z pohľadu užívateľskej komponenty (v našom prípade je užívateľskou komponentou navrhovaná jednotka). Užívateľ definuje 10 bitovú konfiguračnú adresu, ktorá zabezpečí adresovanie jedného z 1024 DW. Endpoint bok následne navráti hodnotu vybraného registra cez 32 bitový výstupný port. Konfiguračné rozhranie však neposkytuje len tento jediný prístup ku konfiguračnému priestoru.

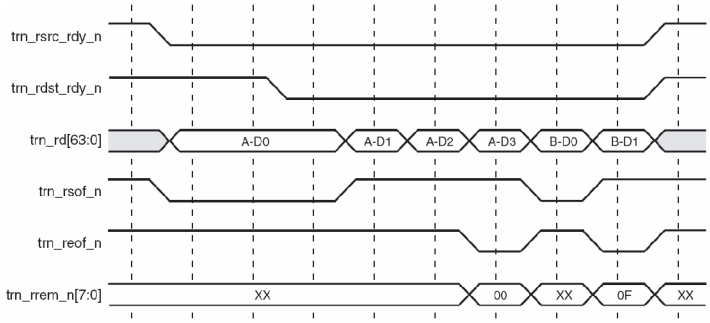
Niektoré z dôležitých informácií (napr. *Bus Number, Device Number, Function Number*) alebo dôležitých signálov (najmä napr. prerušenie) sú vyvedené priamo ako osobitné IO porty endpoint bloku. Pre výpis signálov konfiguračného rozhrania endpoint bloku je možné nazrieť do Príloh A a D.

4.3 Transakčné rozhranie

Transakčné rozhranie nám dáva možnosť pohodlne generovať a prijímať TLP po 64 bitovej dátovej zbernici *(trn_td – vysielanie, trn_rd – príjem).* Ako je už u dát zrejmé, rozhranie pozostáva zo signálov, ktoré sú vždy osobitne pre vysielanie (TX – „t“) a príjem (RX – „r“) pre zachovanie plne-duplexného prenosu. Základné rozhranie a princíp prenosu je veľmi podobné rozhraniu internej zbernice popísanej v Kapitole 5. Okrem dát sú v rozhraní signály pre signalizáciu začiatku prenosu paketu *(tnr_tsof_n, trn_rsof_n),* ďalej signály pre označenie konca prenosu paketu *(trn_teof_n, trn_reof_n)* a signály pre riadenie (pozastavovanie) toku dát z oboch strán *(trn_tsrc_rdy_n a trn_tdst_rdy_n, trn_rsrs_rdy_n a trn_rdst_rdy_n).* Nakoniec je v každom smere i 8 bitový tzv. *data reminder (trn_trem_n, trn_rrem_n)* udávajúci platnosť dát v poslednom prenášanom 64 bitovom slove. Hodnota reminder-u je platná vždy pokiaľ je zároveň aktívne trn_tdst_rdy_n a trn_teof_n (alebo v opačnom smere trn_rsrc_rdy_n, trn_rdst_rdy_n a trn_reof_n). Platné hodnoty sú 00_h u potvrdenia platnosti všetkých prenášaných dát (bity 0 až 63) alebo $0F_h$ u potvrdenia platnosti iba hornej polovice prenášaných dát (bity 63 až 32). Príklady časových diagramov komunikácie pri ktorej boli odoslané (prijaté) 2 pakety sú na Obrázku 4.2 pre TX a nezávisle na Obrázku 4.3 pre RX smer.



Obrázok 4.2: ***Priebeh dvoch transakcií na TX transakčnom rozhraní [12].***



Obrázok 4.3: ***Priebeh dvoch transakcií na RX transakčnom rozhraní [12].***

4.3.1 Rozšírenia vstupného transakčného rozhrania

Okrem spomínaných portov poskytuje vstupné rozhranie i 3 bitový signál udávajúci dostupnosť pamäťových front *(trn_tbuf_av).* Ide síce o port vstupný, avšak označujúci stav výstupného rozhrania endpoint bloku. Údaje o dostupnosti front sú veľmi dôležité pri prenose rôznych typov transakcií a aktívna hodnota (log. 1) na danej pozícií značí, že je možné zaslať do endpoint bloku ešte prinajmenšom jednu transakciu daného typu. Naopak hodnota neaktívna (log. 0) značí nedostupnosť pamäťových prostriedkov, a v tomto prípade treba pozastaviť odosielanie daného typu transakcie. Ignorovanie týchto hodnôt môže spôsobiť nepredvídateľné chovanie endpoint bloku.

Význam jednotlivých položiek zhrňuje nasledujúci popis:

tnr_tbuf_av[0] → *Non Posted transakcie*

tnr_tbuf_av[1] → *Posted transakcie*

tnr_tbuf_av[2] → *Completion transakcie*

Vstupné rozhranie ďalej poskytuje možnosť prerušenia vysielania kedykoľvek pri prenose paketu. V tomto prípade sa nemyslí pozastavenie pomocou už spomenutého trn_tsrc_rdy_n ale cez *tzv. transmit desrination discontiue (trn_tsrc_dsc_n).* Je možné ho aktivovať kedykoľvek medzi začiatkom a koncom prenosu paketu.

4.3.2 Rozšírenia výstupného transakčného rozhrania

Výstupné transakčné rozhranie poskytuje okrem už spomínaných signálov pre riadenie a prenos paketov taktiež niekoľko špecifických. Patrí tu signál riadenia chýb *(trn_rerrfwd_n),* ktorý môže byť aktivovaný kedykoľvek počas príjmu, a označuje tak chybu v práve prichádzajúcom pakete. Na aplikácii potom ostáva vyriešenie tohto problému, napr. zahodením paketu.

Ďalším dôležitým výstupom rozhrania je označenie BARu, do ktorého patrí aktuálna transakcia. Toto nám poslúži na premapovanie adries do lokálneho priestoru adries. Endpoint blok poskytuje 7 bitový vektor hodnôt *(trn_rbar_hit_n),* ktorého pozície bitov udávajú príslušný BAR.

Nasledujúci popis udáva usporiadanie položiek vo vektore:

trn_rbar_hit[0] → *BAR0*

trn_rbar_hit[1] → *BAR1*

trn_rbar_hit[2] → *BAR2*

trn_rbar_hit[3] → *BAR3*

trn_rbar_hit[4] → *BAR4*

trn_rbar_hit[5] → *BAR5*

trn_rbar_hit[6] → *Expansion ROM*

Pokiaľ je niektorá z hodnôt aktívna (log. 0), znamená to zásah do príslušného BARu. V prípade, že máme dva BARy konfigurované do jednej 64 bitovej adresy, tak pri príslušnosti transakcie do tohto priestoru sú naraz aktivované oba príslušné signály.

4.3.3 Obecné transakčné rozhranie

Pod obecné rozhranie endpoint bloku možno zaradiť hlavne výstupný transakčný reset *(trn_reset_n),* ktorého aktivácia musí spôsobiť nastavenie aplikačnej jednotky do východzieho stavu. Ide o synchrónny signál, ktorý je časovaný hodinami. Tieto hodiny *(trn_clk)* sú taktiež ďalším výstupným signálom z obecného rozhrania. Transakčné hodiny sú nedostupné v prípade aktívneho, už v predchádzajúcej kapitole spomínaného, systémového resetu *(sys_reset_n)* a môžu pracovať v jednom z predvolených režimov. Režimy zobrazuje Tabuľka 4.1 nižšie:

Produkt	Odporúčaná frekvencia [MHz]	Voliteľná frekvencia [MHz]
1-lane Endpoint Blok (x1)	62,5	125
4-lane Endpoint Blok (x4)	125	250
8-lane Endpoint Blok (x8)	250	125

Tabuľka 4.1: ***Rozpis frekvencií pre Endpoint Block Plus v1.6.***

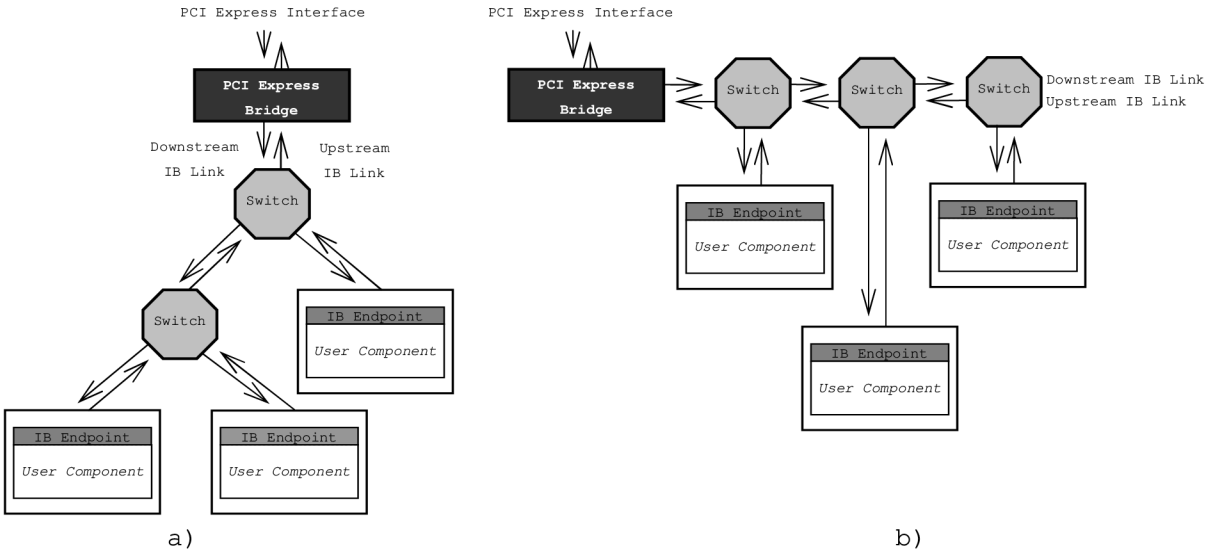
Na tomto mieste by bolo dobre poznamenať, že konfigurácia x1 s frekvenciou 125MHz alebo konfigurácia x4 s frekvenciou 250MHz je zbytočná a môže viesť na extrémne zložitú architektúru jednotiek, pretože oproti použitiu s nižšími frekvenciami nedosiahneme nikdy vyššiu priepustnosť. Navyše môžu vznikať problémy v správe vnútorných pamätí endpoint bloku. Naopak konfigurácia x8 s frekvenciou 125 znamená výrazné zníženie priepustnosti, a to až na polovicu maximálnej možnej.

Posledným signálom obecného rozhrania je stav linky *(trn_lnk_up_n).* Tento udáva v prípade aktívneho stavu pripravenosť oboch strán (endpoint core a druhá strany linky – najčastejšie *Bridge*) na prenos dát. V prípade ak by bol tento signál neaktívny, nie je možné prenášať dáta a pri pokuse o prenos budú všetky pakety navždy stratené.

5 Protokol internej zbernice

Interná zbernica alebo i *Internal Bus (IB)* je vysokopriepustnou, plne-duplexnou zbernicou určenou pre prenos dát v rámci komponent čipu FPGA ale i medzi jednotlivými komponentami a rozhraním PCI, čiže obecne pamäťou RAM hostiteľského počítača (pomocou jednotky riadiacej tok systémovej zbernice). Typickým príkladom komponenty pripojenej k internej zbernici môže byť DRAM radič, *PowerPC* procesor a pod.

Ako názorne ukazuje Obrázok 5.1 (a, b), zbernica využíva stromovú štruktúru zloženú z koncových tzv. *Endpoint* jednotiek, *Switch* jednotiek a z jednotky riadiacej prístup k rozhraní PCI *(Bridge, Root). Endpointy* majú na starosti pripojenie ďalších užívateľsky špecifických komponent. *Switche* prepínajú jednotlivé vetve a vhodne tak smerujú transakčné pakety internej zbernice. A konečne jednotkou riadiacou prístup k rozhraní PCI môže byť jednotka PCI, PCI-X alebo v našom prípade konkrétne jednotka PCI Express *(PCI Express Bridge).* Táto je i predmetom ďalšej práce. Pre bližší popis ostatných častí zbernice je možné nazrieť do špecifikácie internej zbernice [8].

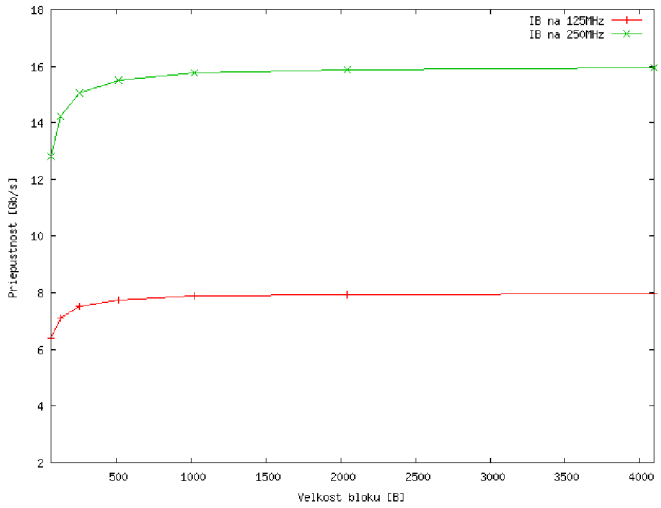


Obrázok 5.1: ***Architektúra internej zbernice (Internal Bus - IB).***

Stromová topológia dovoľuje komunikovať koncovým *Endpoint* jednotkám navzájom v rámci oddelenej vetve zbernice, a tým sa zbytočne neznižuje priepustnosť na vyžlích vrstvách zbernice. Architektúra môže byť však chápaná ale i ako zbernicová architektúra (Obrázok 5.1b), v ktorej sú *Switche* prepojené ako stupne pipelinového reťazca. Tento prístup umožňuje napr. výrazné zjednodušenie procesu *Place&Route* v prípade, kedy budú dve užívateľské komponenty pripojené na rovnakú internú zbernicu avšak situované na opačnej strane FPGA čipu.

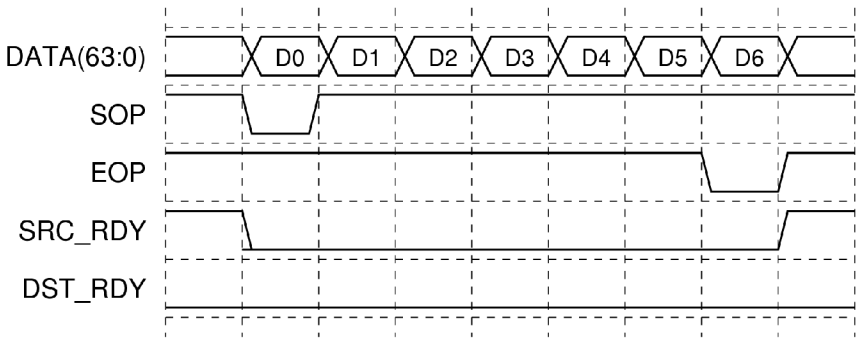
Interná zbernica môže byť 64 alebo 128 bitov široká, taktovaná na 100/125/250 MHz v závislosti od typu FPGA a od potrieb užívateľa. Každá z liniek je plne-duplexnou, čiže dáta smú byť prenášané

nezávisle v oboch smeroch. Maximálna teoretická priepustnosť zbernice (v Gb/s) pre frekvencie 125 MHz a 250 MHz pri dátovej šírke 64 bitov v závislosti od veľkosti prenášaných blokov je vyobrazená na Obrázku 5.2:

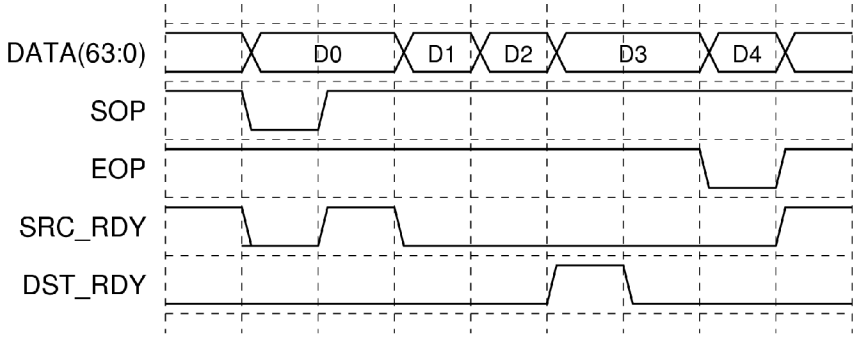


Obrázok 5.2: ***Priepustnosť 64 bitovej internej zbernice na frekvencii 125 MHz.***

Komunikácia na internej zbernici je založená na princípe paketového prenosu. Každý z paketov pozostáva z hlavičky obsahujúcej kontrolné informácie a voliteľne z prenášaných dát. Časový diagram komunikačného protokolu je vyobrazený na Obrázku 5.3 a na Obrázku 5.4. SOP signál (*Start Of Packet* - aktívny v nízkej úrovni) indikuje začiatok prenosu paketu, zatiaľ čo signál EOP (*End Of Packet* - aktívny v nízkej úrovni) indukuje posledné prenášané slovo. Komunikácia medzi dvoma komponentami na zbernici je riadená pomocou signálov SRC_RDY (*Source Ready* – aktívny v nízkej úrovni) a DST_RDY (*Destination Ready* – aktívny v nízkej úrovni). Použitím týchto dvoch signálov je možné pozastaviť komunikáciu v ľubovoľnom čase od prijímajúcej alebo vysielajúcej strany. Výhodou protokolu internej zbernice je jeho kompatibilita s protokolom *Xilinx Local Link* [13]. Práve pre túto kompatibilitu je možné pripojiť jednotky Xilinx IP Core priamo k internej zbernici. Príkladom takejto jednotky je Aurora IP Core, ktorá poskytuje komunikáciu pomocou multigigabitových prijímačov/vysielačov [11].



Obrázok 5.3: ***Priebeh transakcie na internej zbernici.***



Obrázok 5.4: ***Priebeh pozastavovanej transakcie na internej zbernici.***

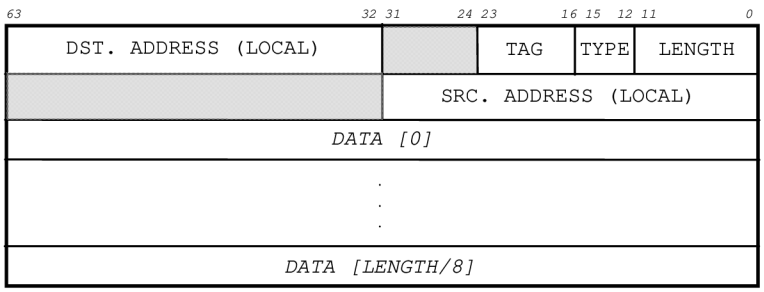
5.1 Typy transakcií internej zbernice

Existujú dva základné typy transakcií na internej zbernici – lokálne transakcie a globálne transakcie. Lokálne transakcie označujú tie transakcie, ktorých prenosy sa uskutočňujú v rámci komponent na FPGA čipe. Na druhú stranu globálne označujú transakcie, ktoré sú mimo lokálneho adresového priestoru i v rozsahu globálneho adresového priestoru (pamäť hostiteľského PC), čiže smerujú skrz PCI bridge.

5.1.1 Lokálne transakcie

5.1.1.1 Local to Local zápis

Local to local zápisová transakcia *(L2LW)* znamená zápisovú operáciu v rozsahu lokálneho adresového priestoru. Príkladom takejto transakcie môže byť zápis dát z jednej komponenty v FPGA do druhej. Na tomto mieste je potrebné poznamenať, že zápisová operácia vyvolaná hostiteľským PC (napr. zápisový prístup do kontrolného registra) je transformovaná na Local to local zápisovú transakciu. Vykonaná je ako lokálny zápis medzi jednotkou PCI bridge a cieľovou komponentou. PCI bridge teda vykoná všetky potrebné transformácie danej transakcie a skryje tak globálnu povahu pôvodného zápisu.



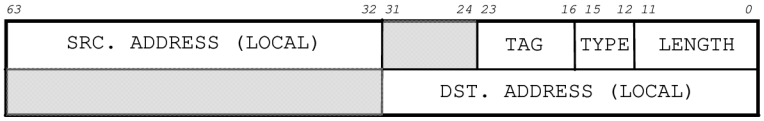
Obrázok 5.5: ***Paket transakcie L2LW.***

Pole paketu	Popis
DST. ADDRESS (LOCAL)	Cieľová adresa v lokálnom adresovom priestore. Je to adresa cieľa kam budú dáta zapísané.
SRC. ADDRESS (LOCAL)	Zdrojová adresa v lokálnom adresovom priestore. Je to adresa komponenty, ktorá zapisuje dáta.
LENGTH	Počet dátových bytov na zápis.
TYPE	Typ : 0000_b
TAG	Identifikácia transakcie.
DATA	Zarovnané dáta.

Tabuľka 5.1: ***Význam položiek paketu L2LW transakcie.***

5.1.1.2 Local to Local čítanie

Local to local čítacia transakcia *(L2LR)* znamená čítaciu operáciu v rozsahu lokálneho adresového priestoru. Príkladom takejto transakcie môže byť čítanie dát z jednej komponenty v FPGA do druhej. Na tomto mieste je opäť potrebné podotknúť, že čítacia operácia vyvolaná hostiteľským PC (napr. čítanie obsahu kontrolného registra) je transformovaná na Local to local čítaciu transakciu. Vykonaná je ako lokálne čítanie medzi jednotkou PCI bridge a cieľovou komponentou. PCI bridge teda vykoná všetky potrebné transformácie danej transakcie a skryje tak globálnu povahu pôvodného čítania.
Local to local čítacie transakcie sú realizované ako tzv. rozdelené transakcie (split transactions). Ako prvé je vygenerovaná príslušná čítacia transakcia s danou identifikáciou (položka TAG). Akonáhle sú požadované dáta nachystané, tak je generovaná dokončovacia (completion) transakcia s príslušnou identifikáciou, ktorá je zhodná s identifikáciou čítacej transakcie. Takto nedochádza k zbytočnému čakaniu na zbernici v prípade, že komponenta z ktorej sa dáta čítajú nemá dáta okamžite pripravené.



Obrázok 5.6: ***Paket transakcie L2LR.***

Pole paketu	Popis
SRC. ADDRESS (LOCAL)	Zdrojová adresa v lokálnom adresovom priestore. Je to adresa komponenty z ktorej sú dáta čítané.
DST. ADDRESS (LOCAL)	Cieľová adresa v lokálnom adresovom priestore. Je to adresa komponenty, ktorá požaduje dáta. Dokončovacia (completion) transakcia bude dáta zasielať práve na túto adresu.
LENGTH	Počet dátových bytov ktoré budú prečítané zo zdrojovej adresy.
TYPE	Typ : 0001_b
TAG	Identifikácia transakcie.

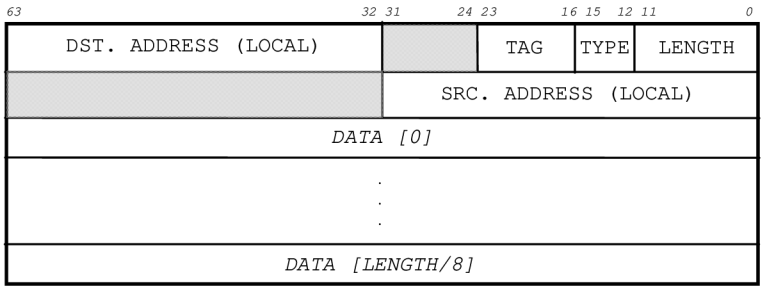
Tabuľka 5.2: ***Význam položiek paketu L2LR transakcie.***

5.1.1.3 Completion

Dokončovacia *(completion)* transakcia je generovaná ako odpoveď na Local to local čítaciu (L2LR) transakciu.

Podporované sú dva typy dokončovacej transakcie:

- *With last fragment bit* – znamená, že ide o poslednú dokončovaciu transakciu.
- *Without last fragment bit* – znamená, že po tejto transakcii bude nasledovať ďalšia dokončovacia transakcia s rovnakou identifikáciou (rovnaký TAG).



Obrázok 5.7: ***Paket Completion transakcie.***

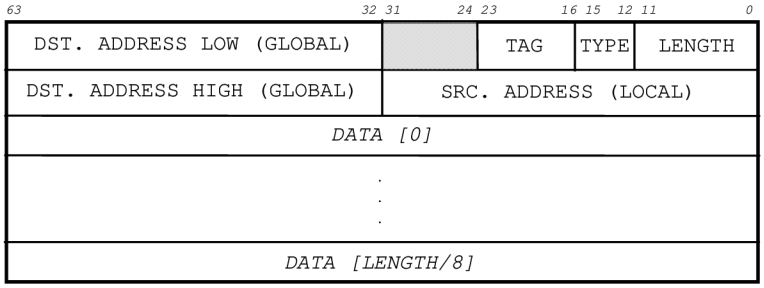
Pole paketu	Popis
DST. ADDRESS (LOCAL)	Cieľová adresa v lokálnom adresovom priestore. Je to adresa cieľa kam budú dáta smerované.
SRC. ADDRESS (LOCAL)	Zdrojová adresa v lokálnom adresovom priestore. Je to adresa komponenty, ktorá odosiela transakciu.
LENGTH	Počet čítaných dátových bytov.
TYPE	Čítacia odpoveď s *„last fragment“* : 1101_b Čítacia odpoveď bez *„last fragment“* : 0101_b
TAG	Identifikácia transakcie.
DATA	Zarovnané dáta.

Tabuľka 5.3: ***Význam položiek paketu Completion transakcie.***

5.1.2 Globálne transakcie

5.1.2.1 Local to Global zápis

Local to global zápisová transakcia (L2GW) znamená zápis z lokálneho adresového priestoru do globálneho priestoru. Príkladom takejto transakcie môže byť prípad, kedy *bus master* jednotka (napr. endpoint na internej zbernici) chce zapísať dáta do pamäte RAM (napr. prijaté pakety).



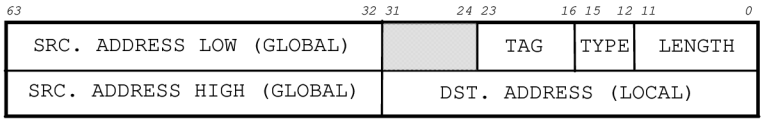
Obrázok 5.8: ***Paket transakcie L2GW.***

Pole paketu	Popis
DST. ADDRESS (GLOBAL)	Cieľová adresa v globálnom adresovom priestore. Je to adresa cieľa kam budú dáta zapísané.
SRC. ADDRESS (LOCAL)	Zdrojová adresa v lokálnom adresovom priestore. Je to adresa komponenty, ktorá zasiela dáta na zápis.
LENGTH	Počet dátových bytov na zápis.
TYPE	Typ : 0010_b
TAG	Identifikácia transakcie.
DATA	Zarovnané dáta.

Tabuľka 5.4: ***Význam položiek paketu L2GW transakcie.***

5.1.2.2 Global to Local čítanie

Global to local čítacia transakcia (G2LR) znamená čítanie z globálneho adresového priestoru do lokálneho adresového priestoru. Príkladom takejto transakcie môže byť prípad, kedy *bus master* jednotka chce čítať dáta z RAM.



Obrázok 5.9: ***Paket transakcie G2LR.***

Global to local čítacie transakcie sú realizované opäť (ako u L2LR) pomocou rozdelených transakcií. Ako prvé je vygenerovaná príslušná čítacia transakcia s danou identifikáciou. Hneď ako sú dáta pripravené je generovaná dokončovacia transakcia s rovnakou identifikáciou.

Pole paketu	Popis
SRC. ADDRESS (GLOBAL)	Zdrojová adresa z globálneho adresového priestoru odkiaľ budú dáta čítane.
DST. ADDRESS (LOCAL)	Cieľová adresa v lokálnom adresovom priestore kam budú dáta zapísané.
LENGTH	Počet dátových bytov na zápis.
TYPE	Typ : 0011_b
TAG	Identifikácia transakcie.

Tabuľka 5.5: ***Význam položiek paketu G2LR transakcie.***

6 Návrh architektúry jednotky

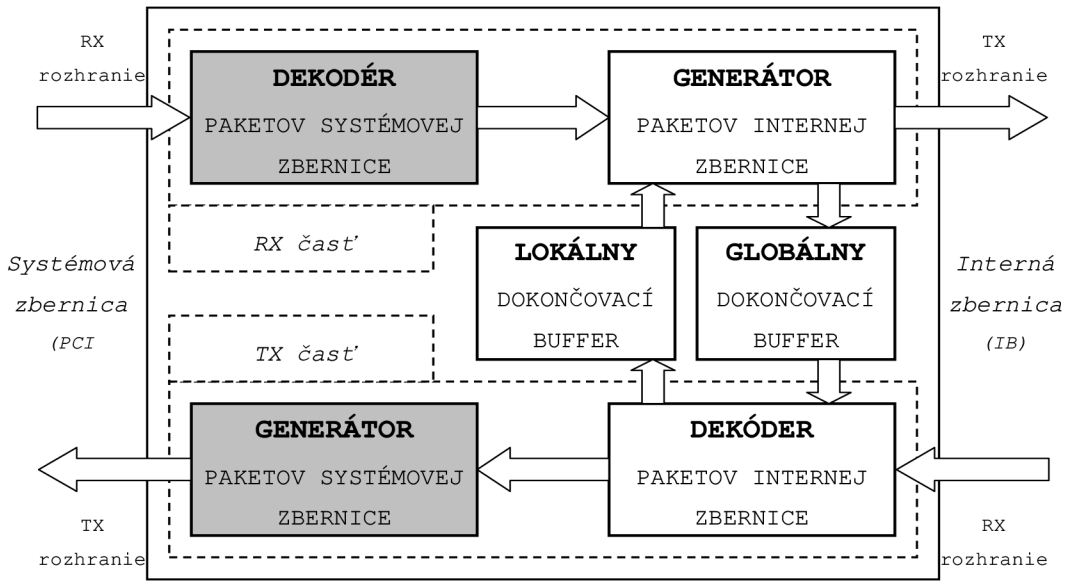
Najvyšším uzlom stromovej architektúry internej zbernice (Kapitola 5) je jednotka riadenia toku systémovej zbernice, ktorá prepojuje systém interných zberníc s nadradeným hostiteľským systémom. Ich prostredníkom je potom už skôr spomínaná jednotka (Kapitola 4), ktorá riadi komplexný nízkoúrovňový komunikačný protokol systémovej zbernice a sama poskytuje jednoduchšie rozhranie na úrovní transakcií, na ktoré sa potom pripojí navrhovaná jednotka riadenia toku. Na základe analýzy dostupných komerčných modulov, ktoré by mohli byť pre tento účet využité, a vzhľadom k požadovanej priepustnosti bola stanovená dátová šírka internej zbernice vedúcej z/do jednotky riadenia toku na fixnú hodnotu 64 bitov.

Hlavnou úlohou jednotky bude obojsmerný preklad medzi transakciami internej zbernice a nadriadeného systému. Z hostiteľskej strany sú prijímané pakety systémovej zbernice. Z hlavičky tohto paketu sa extrahujú jednotlivé riadiace informácie a transformujú sa do podoby, ktorá odpovedá položkám hlavičky internej zbernice. Vykoná sa napr. preklad globálnych adries na lokálne adresy komponent čipu alebo adaptácia dĺžky paketu a pod. Nakoniec sa podľa formátu interných transakcií vyskladá samotný výsledný paket, ktorý bude odoslaný na internú zbernicu. V druhom smere jednotky sú zo strany prepojovacieho systému prijímané pakety internej zbernice. Tieto budú obdobným spôsobom ako v predchádzajúcom prípade transformované na pakety systémovej zbernice. Z ohľadom na požiadavky a účel konkrétnej aplikácie je ďalej možnosť implementovať napríklad i generovanie prerušenia, či iné kontrolné a servisné mechanizmy.

Návrh základnej architektúry komponenty je znázornený na Obrázku 6.1. Skladá sa teda z prijímacej (RX) časti a vysielacej (TX) časti pre dosiahnutie plne-duplexného prenosu a špeciálnych dokončovacích pamätí (tzv. dokončovacie buffery). Pokiaľ má byť možné pripojiť prepojovací systém interných zberníc pomocou jednotky riadenia toku k ľubovoľnej zbernici hostiteľského systému (napr. PCI, PCI-X, či v našom prípade konkrétne PCI Express), musia byť niektoré časti jednotky platformovo závislé. Tieto časti sú na obrázku vyobrazené tmavšou farbou. Jedná sa o jednotky, ktoré sú zodpovedné za rozkladanie a zostavovanie paketov systémovej zbernice. V našom prípade budú platformové závislé časti jednotky implementované pre prácu s paketmi PCI Express. Vďaka tomu bude možné celý systém testovať napr. na karte ML555 [15] alebo karte COMBOv2 [8], ktoré obe disponujú konektorom PCI Express.

Bloky ktoré slúžia k rozkladaniu a zostavovaniu paketov internej zbernice, patria do platformovo nezávislej časti jednotky. Tieto komunikujú s tzv. globálnou a lokálnou (podľa smeru) pamäťou (dokončovacím bufferom), ktorá predstavuje komunikačný bod pre prijímajúcu a vysielaciu časť. Takýto buffer bude slúžiť k ukladaniu identifikačných informácií, ktoré patria jednotlivým čítacím transakciám tej alebo onej strany. V okamžiku keď príde príslušná dokončovacia transakcia, tak sa na základe jedinečnej identifikácie – *tagu* uložené informácie vyzdvihnú a použijú

sa pri preklade medzi transakciami internej zbernice a hostiteľského systému a naopak. Tento princíp bude použitý ako aj pri čítaní z lokálneho, tak i z globálneho adresového priestoru.



Obrázok 6.1: ***Návrh architektúry jednotky.***

6.1 Princíp činnosti v RX smere

Činnosť jednotky riadiacej tok PCI Express v RX smere je možné popísať niekoľkými základnými krokmi:

1. Hlavička prichádzajúceho PCI Express paketu sa rozloží na jednotlivé jej časti, ktorými je adresa, dĺžka, typ a pod.
2. Výstupom dekodéru paketu systémovej zbernice potom bude žiadosť signalizujúca, že bol práve prijatý paket a zároveň budú k dispozícii všetky dostupné riadiace informácie hlavičky. Architektúra tohto dekodéru je závislá na zbernici pripojenej systémovej PCI Express, no rozhranie smerom ku generátoru paketov systémovej zbernice ostane vždy rovnaké.
3. Prijatú požiadavku prevezme generátor paketov systémovej zbernice a vygeneruje príslušnú hlavičku paketu internej zbernice.
4. Za týmto procesom nasleduje vlastný prenos dát. Formát dát uložených v pakete PCI Express je však odlišný od formátu použitého u internej zbernice, preto je potrebné vykonávať príslušné kroky.

RX smer bude podporovať, kvôli kompaktnosti a taktiež kvôli nerozpoznaniu rôznych typov transakcií na internej zbernici, len niekoľko zo všetkých možných transakcií na PCI Express. Ide o jednoduché pamäťové čítacie alebo zápisové operácie v krátkom adresovom formáte (32 bitová adresa) alebo dlhom adresovom formáte (64 bitová adresa). Ďalej bude jednotka samozrejme

podporovať dokončovacie (tzv. completion) transakcie, ktoré vznikajú ako odpoveď na operáciu čítania. Všetky ostatné transakcie (IO a správy) bude jednotka vždy zahadzovať. Do budúcna je možno podporu pre tieto ďalšie typy doimplementovať. Konfiguračné čítanie a zápisy spracováva endpoint blok, ktorý oznamuje zmenu stavu na konfiguračnom rozhraní (viď Kapitola 4.2).

Princíp činnosti RX smeru pre podporované typy transakcií je nasledujúci:

- V prípade najjednoduchšej, zápisovej operácie (v ľubovoľnom adresovom formáte) budú transakcie iba jednoducho transformované a prerovnané na formát internej zbernice. Nevyužíva sa žiadny z dokončovacích pamäťových bufferov, pretože transakcia nie je odpoveďou na žiadnu žiadosť.
- U čítacej transakcie (opäť v ľubovoľnom adresovom formáte) sa do globálneho dokončovacieho buffera ukladajú nutné riadiace informácie, na základe ktorých sa bude po príchodu prečítaných dát v TX smere zostavovať príslušná odpoveď (PCI Express completion).
- V prípade samotnej transakcie čítacej odpovede (PCI Express completion) sa z lokálneho dokončovacieho buffera prečítajú riadiace informácie (uložila ich tam TX transakcia žiadosti), ktoré sa použijú k zostaveniu hlavičky príslušnej transakcie odpovede na internej zbernici (viď ďalej).

6.1.1 Dekodér paketov systémovej zbernice

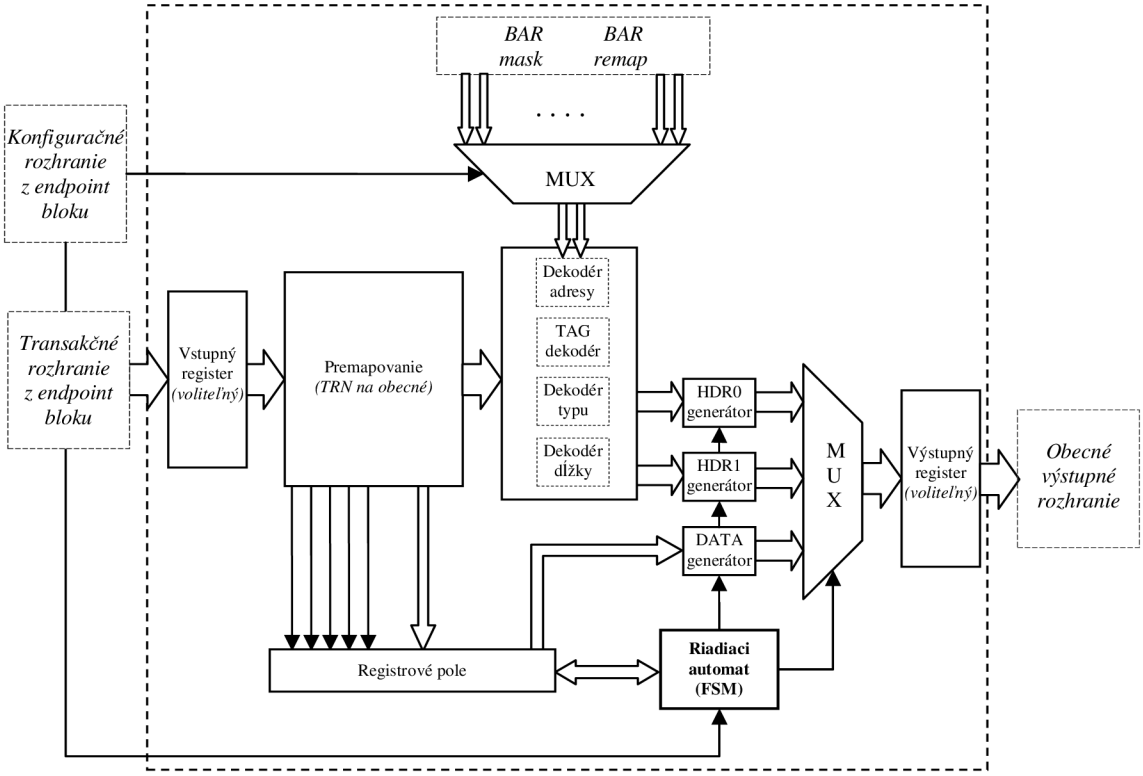
Úlohou tejto jednotky je prevádzať pakety systémovej zbernice na obecné rozhranie pozostávajúce z jednotlivých položiek hlavičky ako je adresa, typ, dĺžka dát a napokon i samotných 64 bitov dát (ak sú v pakete obsiahnuté). S ohľadom na spotrebované zdroje bude jednotka obsahovať z pohľadu registrov iba jednu 64 bitovú pipeline, do ktorej bude uložená v prvom kroku hlavička paketu. Na výstupné rozhranie, obsahujúce jednotlivé riadiace signály, tak budú zavedené signály z tohto registru. Vzhľadom k tomu, že hlavička má celkom dve 64 bitové slová (resp. podľa formátu 3 až 4 DW), tak budú v daný takt vždy platné len niektoré položky výstupného rozhrania.

Do vstupného registra sa okrem hlavičky budú ukladať i dáta. Tie je dôležité vhodným spôsobom upraviť na formát PCI Express dát, čo v podstate znamená iba zmenu poradia bytov. V dekodéri je taktiež potrebné vykonávať prepočet dĺžky dát samotných z formátu v DW a za pomoci hodnôt bitov platnosti (First/Last DW Enable) do formátu internej zbernice, teda v počte bytov. Hodnota dĺžky v počte bytov je viac vyhovujúca pre pripojené komponenty na čipe zariadenia, ktoré takto nemusia obsahovať už dekodér prepočtu hodnoty z dĺžky v DW. V prípade dokončovacej transakcie potom ešte závisí výpočet dĺžky aj od toho, či ide o poslednú časť odpovede na jednu čítaciu požiadavku, pretože špecifikácia dovoľuje aby odpoveď na jedno čítanie mohla prísť rozdelená do viacerých menších paketov.

Ďalšou úlohou dekodéra je premapovanie adresy na základe BARov. Keďže máme z konfiguračného rozhrania k dispozícií informáciu o tom do ktorého BARu transakcia smeruje, je možné podľa aktuálnej konfigurácie (generické parametre jednotky) ľahko zistiť bázovú adresu *(BARx_REMAP)* a masku *(BARx_MASK).* Výsledná, cieľová adresa sa potom určí na základe adresy v hlavičke paketu, tejto bázovej adresy a masky.

6.1.1.1 Architektúra dekodéru paketov systémovej zbernice

Na Obrázku 6.2 je zobrazená základná architektúra dekodéra paketov systémovej zbernice. Vstupom do tejto komponenty je jednak transakčné rozhranie z endpoint bloku (Kapitola 4.3), odkiaľ dostáva komponenta prísun paketov typu PCI Express, no taktiež i konfiguračné rozhranie (Kapitola 4.2), ktoré sprístupňuje niektoré potrebné riadiace a stavové informácie zbernice. Viac v Prílohe A s rozhraním komponenty. Riadiaca cesta je na obrázkoch zobrazená plnými úzkymi šípkami a dátová je naopak naznačená šípkami nevyplnenými.



Obrázok 6.2: ***Bloková architektúra dekodéra paketov systémovej zbernice.***

Na vstupe a zároveň na výstupe komponenty sú voliteľné oddeľovacie registre, ktoré v ďalších fázach môžu pomôcť pri zvyšovaní pracovnej frekvencie komponenty. Za vstupným registrom nasleduje blok premapovania, ktorý mapuje jednotlivé signály do registrového poľa alebo do ostatných dekodérov.

Registrové pole uchováva stavové informácie práve prebiehajúcej transakcie, ktoré slúžia pre riadiaci automat, no taktiež registruje vstupné dáta a hodnotu REQUESTER_ID (Bus Number, Device Number, Function Number). Medzi stavové informácie patrí informácia o formáte hlavičky

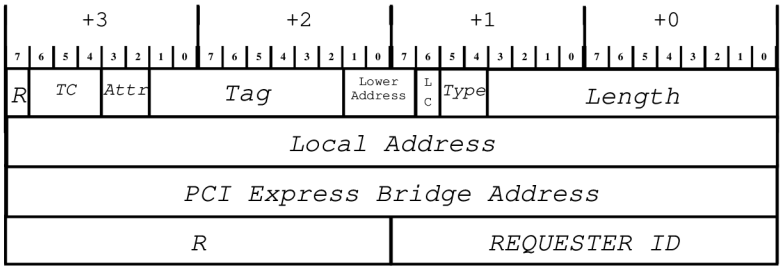
(3/4 DW), informácia o transakcii bez dát (NO_DATA, LAST_DATA) a informácia o potrebe posuvu dát o 32 bitov.

Dekodérov komponenta obsahuje hneď niekoľko. Tým najdôležitejším je dekodér typu transakcie, ktorý na základe položiek FMT a TYPE z paketov systémovej zbernice určí aktuálny typ transakcie. Taktiež označí, či je daný typ podporovaný alebo nie (UNSUPPORTED). Túto informáciu potrebuje opäť riadiaci automat. Dekodér dĺžky prepočítava z dĺžky udanej v DW (vynásobením s hodnotou 4 – obvodovo riešené posuvom), ďalej z platnosti prvého a posledného dátového slova (FIRST_BE, LAST_BE), počtu bytov u odpovedi na čítanie (BYTE_CNT) a spodných bytov adresy (LOWER_ADDR) dĺžku v bytoch pre internú zbernicu. Toto je však v závislosti od práve spracovávanej transakcie, pretože nie všetky položky sú prístupné vo všetkých typoch transakcií. Dekodér adresy vypočíta adresu lokálneho adresového priestoru podľa aktuálneho BARu, do ktorého transakcia patrí a podľa masky tohto BARu ako:

$$Výsledná\_adresa = (Adresa\_ z\_PCI\_Express\_hlavičky \ \mathbf{AND} \ BARx\_MASK) + BARx\_REMAP$$

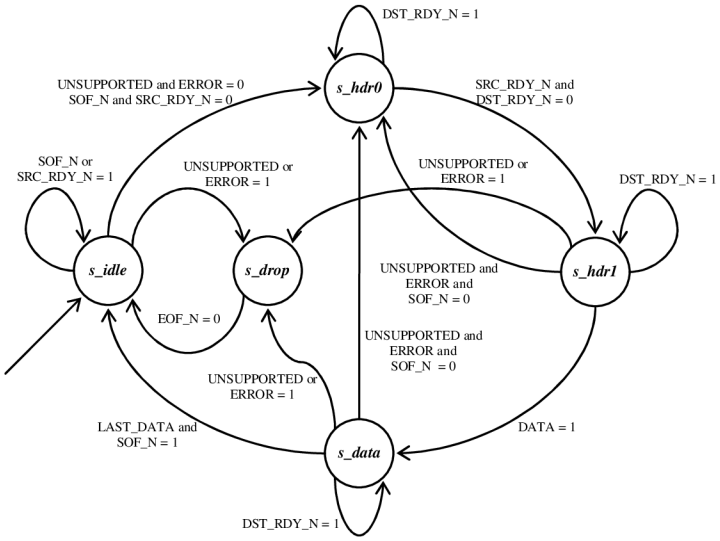
Navyše pokiaľ je nastavená i položka FIRST_BE inak ako „1111“, dekodér toto do adresy taktiež započíta a vhodne adresu upraví jej posuvom. Napokon najjednoduchší dekodér identifikácie transakcie na základe už dekódovaného typu prepína správny TAG. Buď zápisový, čítací alebo TAG odpovede na čítanie, ktoré sú v paketoch umiestnené opäť na iných pozíciách.

Generátory hlavičiek (HDR0, HDR1) a generátor dát (DATA) zostavujú položky obecných paketov smerujúcich na výstup komponenty. Predstavujú v podstate už spomínané záchytné registre, ktoré uchovávajú jednotlivé položky novo vznikajúceho paketu, pretože nie všetky položky by boli bez registrovania v daný okamžik platné. Predpis takého obecného paketu je uvedený na Obrázku 6.3:



Obrázok 6.3: ***Formát hlavičky obecného paketu v RX smere.***

O prepínanie jednotlivých častí hlavičky (HDR0, HDR1) a preusporiadaných dát (DATA) sa stará multiplexor (MUX), ktorý je tak ako i iné podkomponenty riadený riadiacim automatom. Činnosť automatu zobrazuje nasledujúci Obrázok 6.4. Princíp jeho činnosti je možné popísať v niekoľkých krokoch:



Obrázok 6.4: ***Riadiaci automat dekodéra paketov systémovej zbernice.***

- Automat sa na počiatku nachádza v stave ***s_idle*****.** Tu čaká na začiatok podporovaného typu paketu, ktorý musí byť navyše bez chyby.
- Nepodporované typy sa zahadzujú v stave ***s_drop***. V tomto stave automat riadi i zahadzovanie paketu, ktorý bol endpoint blokom označený ako chybný, a čaká sa tu až do ukončenia prenosu identifikovaného koncovou značkou.
- Pri korektnom príme prvej časti paketu sa prechádza do stavu ***s_hdr0,*** odkiaľ sa riadi vysielanie prvej časti hlavičky na obecné rozhranie a automat prechádza do ďalšieho stavu.
- V stave ***s_hdr1*** sa riadi odosielanie druhej časti hlavičky paketu. Automat následne rozhoduje, či paket obsahuje dáta:
    - Ak obsahuje, prechádza do stavu ***s_data,*** odkiaľ sa riadi korektné odosielanie dát na obecné rozhranie.
    - Pokiaľ dáta nenasledujú, môže sa detekovať nový prichádzajúci paket (s_hdr0) alebo práve prichádza ďalší nepodporovaný či poškodený paket (s_drop) alebo sa na zbernici práve nič nedeje (s_idle).
    - Zo stavu kedy automat odoslal poslednú časť dát (s_data) sa prechod do nových stavov riadi obdobne ako zo stavu odosielania druhej hlavičky (s_hdr1) avšak bez dát.

6.1.2 Generátor paketov internej zbernice

Generátor paketov internej zbernice má za úlohu zostaviť hlavičku paketu internej zbernice na základe poskytnutých riadiacich informácií rozhrania, ďalej komunikovať s dokončovacím bufferom a napokon, v prípade dokončovacej transakcie, prerovnávať dáta zo zdrojovej adresy na cieľovú.

Prerovnanie dát zo zdrojovej na cieľovú adresu u dokončovacích transakcií vykonávame preto, lebo u tohto typu transakcie sú dáta obsiahnuté v pakete prerovnané na zdrojovú (odkiaľ boli dáta čítané) a nie na cieľovú (kam budú dáta zapísané) adresu. Špecifikácia internej zbernice to ale vyžaduje presne naopak. Keďže prerovnávacia jednotka znamená nemalé zdroje navyše, bude ju možno v jednotke generátora povoliť alebo zakázať (generický parameter, ktorý inštancuje prerovnávaciu jednotku). Prerovnávanie nie je potrebné ani v tom prípade, keď sa číta zo zarovnaných adries (na hranicu 4 bytov) alebo pokiaľ sa cieľová a zdrojová adresa na spodných 3 bitoch zhodujú (dáta sú zarovnané rovnako).

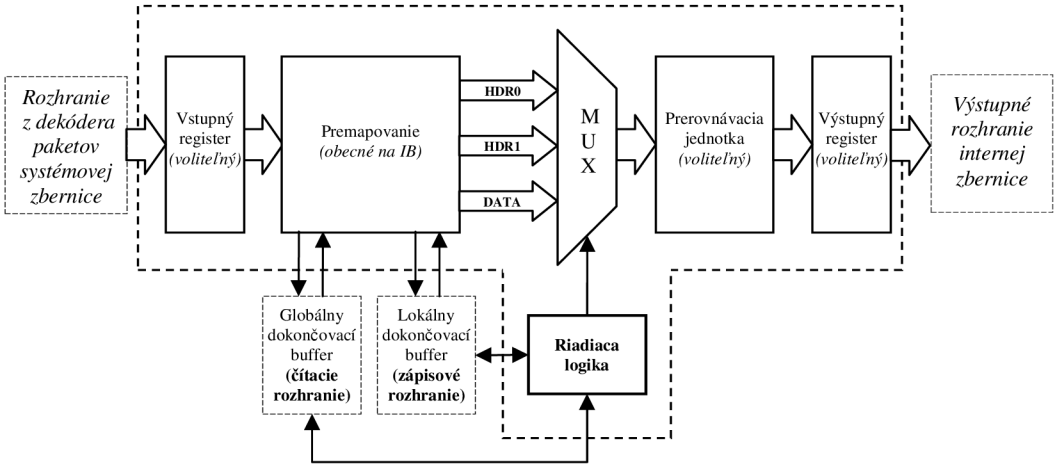
Čo sa týka detailnejšej komunikácie s dokončovacím bufferom, tak v prípade zápisovej transakcie nie je potrebné vôbec k nemu pristupovať. V prípade čítacej transakcie je potrebné v generátore ukladať do lokálneho dokončovacieho bufferu informácie, ktoré sú potrebné pre odpoveď. Sú to *TC, ATTR, TAG, BUS_NUMBER, DEVICE_NUMBER* a *FUNC_NUMBER*. Tieto informácie sa potom čítajú v dekodéri paketov internej zbernice pri tvorbe transakcie odpovede (viď ďalej).

V prípade dokončovacej transakcie je komunikácia s lokálnym dokončovacím bufferom zložitejšia. V bufferi budú uložené informácie o lokálnej adrese a tagu transakcie, ktoré tam uložil dekodér paketov internej zbernice pri generovaní čítacej požiadavky do globálneho adresového priestoru (TX smer). Ako bolo už spomenuté, odpovedí na čítaciu požiadavku môže prísť viacero (viacero dokončovacích transakcií), ktoré na seba nadväzujú a je teda potrebné upravovať uloženú lokálnu adresu. Hodnota sa upravuje podľa dĺžky prijatých dát až do poslednej transakcie, kedy sa hodnota z buffera definitívne odstráni a položka sa tak uvoľní. Toto môže predstavovať úzke miesto časovania z pohľadu návrhu jednotky. Posledný odosielaný paket na internú zbernicu sa zároveň označí ako *Last Fragment* (viď Kapitola 5).

6.1.2.1 Architektúra generátora paketov internej zbernice

Tak ako predchádzajúca komponenta dekodéra paketov systémovej zbernice je i nasledujúca na vstupe a výstupe oddelená vstupným a výstupným oddeľovancím registrom. Tieto sú samozrejme voliteľné. Základná architektúra generátora je vyobrazená na Obrázku 6.5.

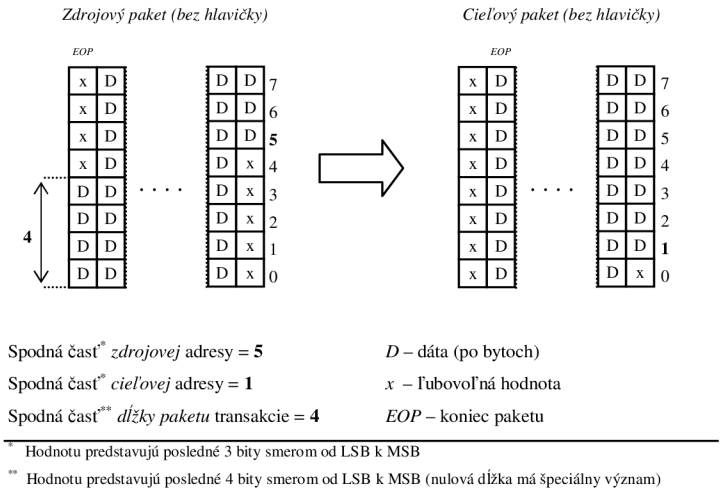
Vstupom komponenty je obecný formát uvedený na Obrázku 6.3 vyššie. Z tohto sa premapovaním jednotlivých položiek pristupuje do dokončovacích bufferov. U globálneho dokončovacieho buffera cez čítacie rozhranie a do lokálneho dokončovacieho buffera naopak cez zápisové rozhranie. Konkrétne signály je možno nájsť v Prílohe B. Premapovanie zároveň mapuje položky obecného rozhrania na formát prvej a druhej hlavičky paketu internej zbernice. Taktiež mapuje dáta paketu na výstup.



Obrázok 6.5: ***Bloková architektúra generátora paketov internej zbernice.***

O riadenie celej komponenty sa v tomto prípade stará jednoduchá riadiaca logika, ktorá detekuje začiatok paketu (prvá hlavička), prenos po začiatku paketu (druhá hlavička) a zvyšný prenos až po koniec paketu (samotné dáta). Podľa toho riadi prepínanie výstupu z multiplexora (MUX), zápis do bufferov a napokon ovláda i rozhranie na vstupe i výstupe.

Za multiplexorom, ktorý prepína položky paketu je zaradená i voliteľná prerovnávacia jednotka, ktorá zistí typ transakcie (potrebuje odpoveď na čítanie – PCI Express completion) a podľa dĺžky, zdrojovej adresy a cieľovej adresy prerovná na cieľovú adresu dáta obsiahnuté v pakete. Princíp prerovnávania názorne ilustruje Obrázok 6.6:



Obrázok 6.6: ***Princíp prerovnávania dát v generátore paketov internej zbernice.***

Prerovnávanie sa obvodovo rieši pomocou posuvných záchytných registrov *(tzv. barrel shifter),* zložených z po bytoch adresovateľných LUT pamätí (adresuje vždy platné dáta v celej dvojstupňovej linke) a z multiplexora, ktorý platné dáta korektne prepína na cieľovú pozíciu. Môže tak nastať napríklad tá situácia, ktorú zobrazuje obrázok, kedy sa na začiatku prenosu musí čakať na

nasunutie dostatočného množstva dát do pamäte. Na konci naopak kvôli malej dĺžke a zarovnaní odpadne jedno prenášané slovo. Je preto potrebné vhodne riadiť logiku signalizácie začiatku (pre detekciu pripravenosti prerovnávača) a konca prenášaného paketu. Samotný prerovnávací obvod je teda schopný vyčítať a uložiť si z hlavičky daného paketu potrebné hodnoty adries a dĺžok, hlavičku poslať na výstup a následné dáta podľa získaných hodnôt vhodne prerovnávať a zasielať na výstup.

6.2 Princíp činnosti v TX smere

Základný princíp navrhovanej jednotky riadiacej tok PCI Express je opäť možné, tak ako u RX smeru popísať niekoľkými krokmi. Keďže na oboch stranách (RX i TX) navrhovanej jednotky sú podobné typy protokolov, obecný princíp bude opäť obdobný:

1. Hlavička prichádzajúceho paketu internej zbernice sa rozloží na jednotlivé svoje časti, a práve prijatá časť paketu sa vhodne indikuje do generátora paketov.
2. Výstupom dekodéra paketu internej zbernice bude teda žiadosť, signalizujúca prijatie nového paketu, pričom všetky momentálne dostupné informácie budú vždy aktuálne a k dispozícii.
3. Generátor paketov systémovej zbernice preberá túto žiadosť a na základe poskytnutých informácii o type, adrese, dĺžke a pod. zostavuje príslušnú hlavičku PCI Express transakcie, ktorú následne zasiela na systémovú zbernicu.
4. Napokon v prípade existencie dát v transakcii, prichádza ich samotný prenos. Dáta budú prerovnané tak, ako to vyžaduje štandard PCI Express.

Tak ako aj RX smer nepodporuje prijímanie všetkých typov PCI Expres transakcií, tak i TX smer nebude tieto nepodporované typy generovať. Princíp činnosti, opäť s ohľadom na využitie dokončovacích bufferov je nasledujúci:

- v prípade zápisovej transakcie zo strany internej zbernice, je táto jednoducho prevedená na transakciu systémovej zbernice bez použitia vnútorných pamätí v podobe dokončovacích bufferov.
- Pokiaľ však z internej zbernice vystavíme žiadosť o čítanie, sú potrebné informácie (lokálna adresa a pod.) uložené do lokálneho dokončovacieho buffera. Pri dokončovacej transakcii v RX smere sa táto hodnota následne vyberá (prípadne upravuje, ako bolo spomenuté).
- Pre odpoveď na čítaciu transakciu zo strany systémovej zbernice musíme využiť naopak hodnoty uložené v globálnom dokončovacom bufferi a pomocou nich zostaviť príslušnú transakciu na systémovej zbernici (PCI Express completion).

6.2.1 Dekodér paketov internej zbernice

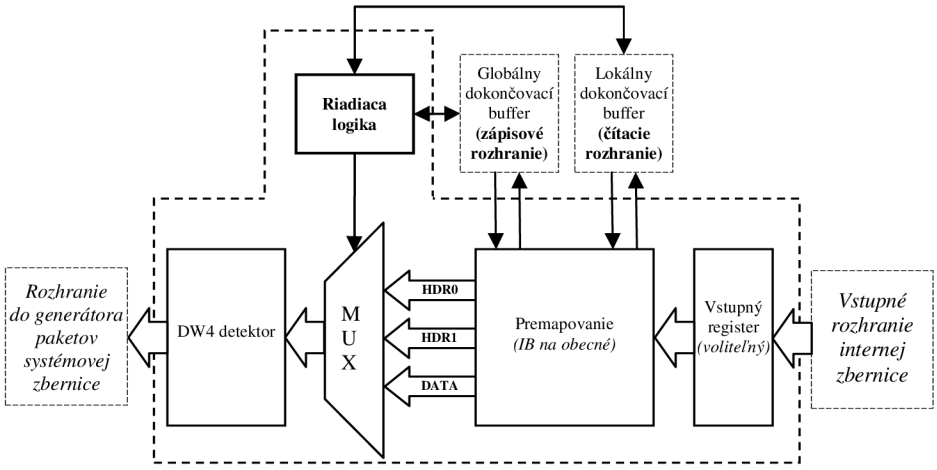
Základnou úlohou tejto jednotky je prevádzať pakety internej zbernice na obecné rozhranie pozostávajúce z jednotlivých položiek hlavičky ako je adresa, prevedený typ, dĺžka a samozrejme 64 bitov dát. V tejto jednotke sa nevykonáva žiadne rozšírené dekódovanie, pretože to je potrebné vykonať až v generátore paketov systémovej zbernice práve kvôli platformovej závislosti ďalšieho spracovania.

Jednotka bude s ohľadom na spotrebované zdroje obsahovať jedinú 64 bitovú pipeline, do ktorej sa budú ukladať postupne dve slova hlavičky a následne samotné dáta. Tu však nemusia byť platné hneď obe 32 bitové dátové slová, ale napr. pri dĺžke 4 byty bude platné iba jediné z nich. PCI Express ale nepodporuje prenos prázdneho prvého slova v dátach, a preto je potrebné prepínať medzi jednotlivými slovami a tak ich prípadne preusporiadať.

Ako bolo spomenuté, v prípade čítacej transakcie ukladá táto jednotka do lokálneho dokončovacieho buffera lokálnu adresu a tag pre dokončovaciu transakciu. Taktiež v prípade dokončovacej transakcie vyberá z globálneho dokončovacieho buffera údaje o identifikácii cieľa *(BUS_NUMBER, DEVICE_NUMBER, FUNCTION_NUMBER)* a ostatné informácie *(TC, ATTR, TAG)* pre správnu identifikáciu transakcie.

6.2.1.1 Architektúra dekodéra paketov internej zbernice

Do tejto komponenty vstupuje rozhranie internej zbernice, ktoré je voliteľne registrované. Nasleduje blok premapovania, ktorý tentokrát s lokálnym dokončovacím bufferom komunikuje cez čítacie rozhranie a s globálnym dokončovacím bufferom cez rozhranie zápisové. Konkrétne signály je možno nájsť v Prílohe C.



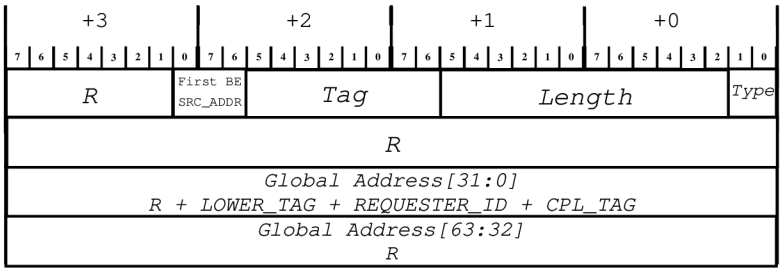
Obrázok 6.7: ***Bloková architektúra dekodéra paketov internej zbernice.***

Blok premapovania sa taktiež stará o správne nastavené položky prvej (HDR0) a druhej hlavičky (HDR1) obecného formátu. Taktiež privádza zatiaľ neupravené dáta, ktoré sú prerovnávané

a preusporiadané až v generátore paketov systémovej zbernice. O správny súbeh a riadenie sa stará jednoduchá riadiaca logika, zložená z detektora prvej a druhej hlavičky spolu s detektorom konca prenosu. Tak ako i u generátora paketov na internej zbernici riadi táto logika prepínanie výstupu multiplexoru (MUX), riadi zápis do bufferov a riadi i vstupné a výstupné rozhranie komponenty.

Na výstupe jednotky je zaradený jednoduchý tzv. DW4 detektor. Ten okrem toho že registruje vstup zisťuje, či sa bude na výstupe generovať hlavička formátu DW3 alebo DW4. Toto je potrebné pre ďalšiu komponentu generátora paketov systémovej zbernice, ktorému sa týmto ušetrí jeden stupeň linky. Obsahuje v sebe komparátor na nulovú hodnotu adresy [63:32].

Obecný formát paketov, ktorý jednotka vytvára je vidieť na Obrázku 6.8. Je vidieť, že pripravuje paket tak, aby bol čím najlepšie spracovateľný generátorom paketov systémovej zbernice, pričom už v prvej hlavičke pripraví informáciu o spodnej časti zdrojovej adresy (u odpovede na čítanie) alebo FIRST_BE (ostatné typy), ktoré je odvodené taktiež vždy z adresy. Podľa typu transakcie taktiež paket v druhej hlavičke obsahuje buďto globálnu adresu (zápis, čítanie), alebo riadiace informácie odpovede na čítanie, ktoré sú získane z dokončovacieho buffera.



Obrázok 6.8: ***Formát hlavičky obecného paketu v TX smere.***

6.2.2 Generátor paketov systémovej zbernice

Jednotka generovania paketov systémovej zbernice má za úlohu skladať paket (zložený z hlavičky a dát) z obecných riadiacich položiek dekodéra paketov internej zbernice a posielať tieto na systémovú zbernicu (transakčné rozhranie endpoint bloku). Rovnako ako i v RX smere je potrebné dáta preformátovať na typ použitý na zbernici PCI Express, čo znamená opäť zmeniť poradie vysielaných bytov opačným spôsobom ako v smere RX, prípadne posunúť dáta na vhodnú pozíciu podľa zarovnania. Na vysielacej strane generátora nie je potrebné sa zaoberať (na rozdiel od RX) BARmi a premapovávaním lokálnych adries, pretože ako adresa sa použije priamo tá, ktorá príde z rozhrania dekodéru. Naopak je potrebné sa venovať výpočtu Last BE hodnôt (z dĺžky po prerovnaní dát a prepočte na dĺžku v DW) a fragmentácii príliš veľkých dátových paketov.

Paket internej zbernice môže mať obecne veľkosť 1B až 4kB. Maximálna veľkosť paketu zbernice PCI Express závisí od konfigurácie endpoint bloku. Presnú hodnotu je možné zistiť pomocou konfiguračného rozhrania (väčšinou je rovná hodnote 128 B). Z toho ale vyplýva, že dátové zápisové transakcie a dokončovacie transakcie smerujúce do globálneho adresového priestoru (zápis

a odpoveď na čítanie) musia byť fragmentované (rozdeľované) na menšia časti o maximálne takej veľkosti, ktorá je podporovaná. Zápisové transakcie síce môžu byť podľa špecifikácie [9] fragmetnované podľa potreby rôzne, avšak pre zjednodušenie celého návrhu a výslednej logiky obvodu budú oba typy rozdeľované rovnako.

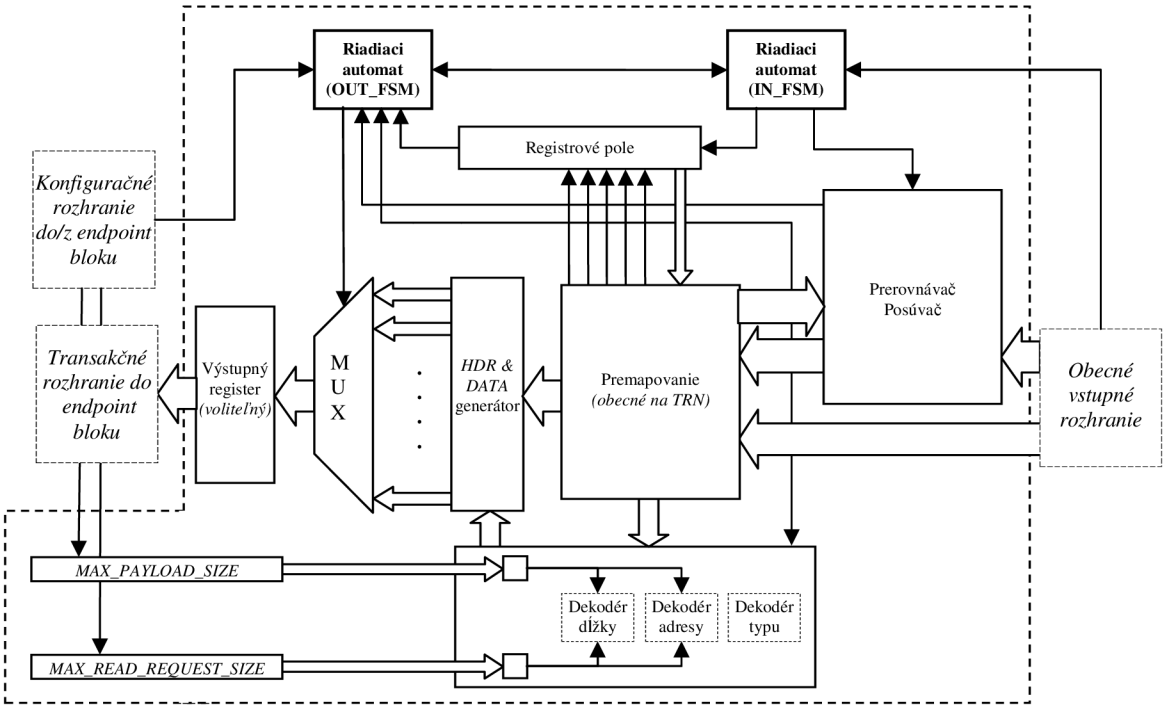
Pretože každý paket môže byť teoretický väčší ako je povolená hranica, musia byť súčasťou jednotky generátora i registre pre uloženie jednotlivých riadiacich informácií hlavičky, ktoré budú k dispozícii po celú dobu priebehu transakcie.

Pre každý fragment jednej zápisovej transakcie smerujúcej z internej zbernice sa musí prepočítať adresa, na ktorú paket smeruje. Taktiež je potrebné upravovať dĺžky paketu v DW presne tak, ako sú fragmentované pakety odosielané.

Operácie pre paket dokončovacej transakcie sú o niečo málo zložitejšie. Je potrebné totiž udržiavať tzv. *Lower addres*, ktorá je v prvom kroku rovná vstupnej zdrojovej adrese a v ďalších krokoch rovná nule, ďalej je potrebné prepočítavať dĺžku v DW, no a napokon prepočítavať aktuálnu hodnotu počtu bytov, ktoré je potrebné ešte preniesť. Naopak ale nie je potrebné prepočítavať adresy (používa sa smerovanie na základe ID).

6.2.2.1 Architektúra generátora paketov systémovej zbernice

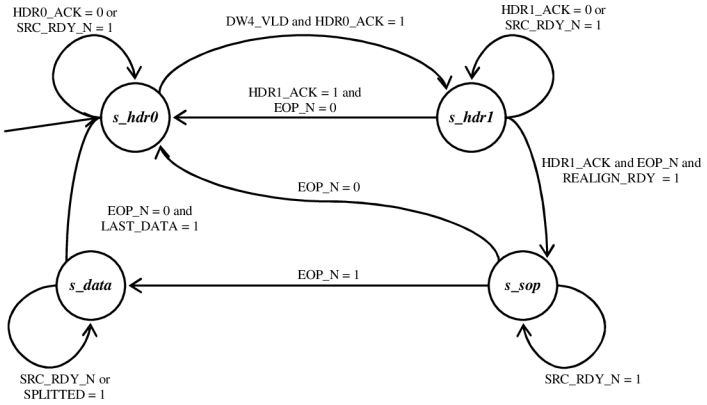
Architektúra zdrojovo najnáročnejšej komponenty generátora paketov systémovej zbernice je blokovo naznačená na Obrázku 6.9.



Obrázok 6.9: ***Bloková architektúra generátora paketov systémovej zbernice.***

Vstupom komponenty je tentokrát obecné rozhranie, ktoré poskytuje prevedené pakety internej zbernice. Výstupom je naopak cez voliteľný oddeľovací register transakčné rozhranie do endpoint bloku (Kapitola 4.3). Okrem týchto je jedným z rozhraní komponenty opäť i konfiguračné rozhranie (Kapitola 4.3). V Prílohe D je možné nájsť kompletné rozhranie celej komponenty.

Zo vstupu smerujú pakety do posúvača alebo prerovnávača, ktorý má na starosti premapovať dáta na PCI Express formát a podľa získaných hodnôt v prípade potreby prerovnávať dáta skrz neho prechádzajúce (spätne na zdrojovú adresu). To sa deje na základe hodnôt (FIRST_BE a LOWER_ADDR) zavedenej spätnej väzby od bloku premapovania. Tento blok získal hodnoty taktiež priamo zo vstupu. Okrem toho, že hodnoty sprístupňuje prerovnávaču, zavádza ich taktiež do registrového poľa pre ďalšie využitie. Okrem už spomínaných sa registrujú i hodnoty typu prebiehajúcej transakcie, formáte hlavičky, LAST_BE, REQUESTER_ID a iné. Celý popisovaný vstupný blok riadi vstupný riadiaci automat zobrazený na Obrázku 6.10.



Obrázok 6.10: ***Vstupný riadiaci automat generátora paketov systémovej zbernice.***

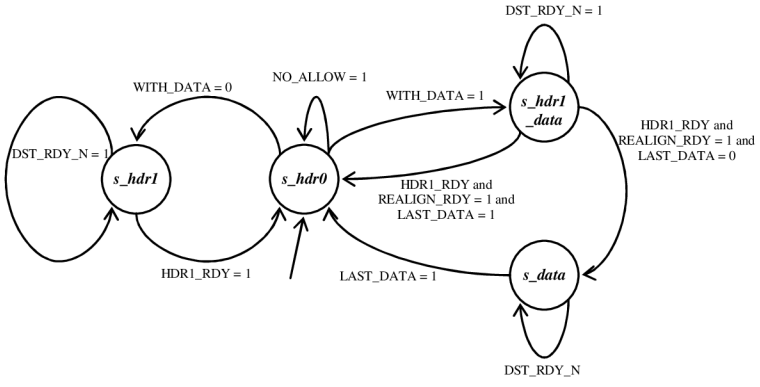
Vstupný riadiaci automat riadi spracovanie paketov zo vstupného obecného rozhrania. Okrem registrového poľa a prerovnávacej jednotky komunikuje taktiež s výstupným automatom. S týmto si predáva informácie o pripravenosti jednotlivých položiek (HDR0-1_RDY, HDR0-1_ACK).

- Na počiatku sa čaká na hlavičku prichádzajúceho paketu v stave ***s_hdr0***. Pokiaľ sa všetky potrebné hodnoty uložili do registrov a zároveň výstupný automat prebral tieto dáta (HDR0_ACK), je možné prejsť do stavu pre spracovanie druhej časti hlavičky.
- Ak je v druhom stave ***s_hdr1*** zo vstupu detekovaný koniec paketu a dáta boli odobrané, prechádza sa do počiatočného stavu s_hdr0, kde sa opäť čaká na ďalší paket. Pokiaľ koniec paketu nebol detekovaný (EOP_N), je potrebné počkať až kým sa nepreusporiadajú vstupné dáta z prerovnávača resp. posúvača.
- Ak toto nastane a je teda detekovaný začiatok dát ***(s_sop),*** dáta sa korektne prenesú na ďalšie spracovanie a opäť sa čaká na koniec paketu.
    - Ak koniec paketu nastal, prechádza sa ako v predošlom prípade do stavu čakania na nový paket (s_hdr0).

- Ak však koniec paketu ešte stále nebol detekovaný, a pritom prerovnávacia linka už je naplnená, je možné prejsť do stavu ***s_data,*** v ktorom sa dokončí prenos dát. V tomto stave sa taktiež čaká v prípade, že je potrebné výstupný paket rozdeliť. Po prenose nasleduje prechod do počiatočného stavu (s_hdr0).

Ďalšou dôležitou súčasťou komponenty sú dekodér dĺžky, adresy a typu. Dekodér typu určuje správny typ a sprístupňuje po prekódovaní túto informáciu nielen výstupnému riadiacemu automatu ale, ako i ostatné, generátoru hlavičiek. Adresový dekodér udržuje aktuálnu informáciu o cieľovej adrese transakcie, a zároveň generuje FIRST_BE, LAST_BE a BYTE_CNT u dokončovacej transakcie. Dekodér dĺžky generuje dĺžku v DW pre systémovú zbernicu. Oba dva posledné spomínané dekodéri musia navyše ale pracovať s položkami získanými z konfiguračného rozhrania endpoint bloku (MAX_READ_REQUEST_SIZE a MAX_PAYLOAD_SIZE), podľa ktorých si pri povele od výstupného automatu (INCREMENT) upravia aktuálne udržiavané položky. Tieto naopak sprístupňujú informáciu o posledných prenášaných dátach (LAST_DATA) a to i v prípade konca fragmentovanej časti. Oba pracujú obdobne – dekodér adresy inkrementuje a dekodér dĺžky naopak dekrementuje o danú, preformátovanú hodnotu. Toto sa deje vždy až pokiaľ nedôjde k odoslaniu posledného fragmentu paketu. Rozdelené takto nemusia byť len dátové pakety, no ale i pakety ktoré sú bez dát.

O správne generovanie položiek paketu sa stará spomínaný generátor hlavičiek/dát. Ide v podstate len o logiku, ktorá zozbiera všetky potrebné informácie, z ktorých následne zloží príslušnú časť paketu. Keďže v tomto smere môže byť typov paketu viac (32 a 64 bitový formát je rozdielný), obsahuje i výstupný multiplexor viac ako 3 možnosti. Sám je ovládaný jednak na základe uloženého typu transakcie a jednak na základe riadiacej informácie od výstupného automatu.



Obrázok 6.11: ***Výstupný riadiaci automat generátora paketov systémovej zbernice.***

Výstupný riadiaci automat na rozdiel od vstupného sleduje výstupné rozhranie komponenty, a podľa toho riadi odosielanie príslušných hlavičiek paketov na systémovú zbernicu. V prípade, že sú obsiahnuté dáta, riadi odosielanie aj týchto. Toto samozrejme v spolupráci so vstupným automatom, ktorý mu indikuje pripravenosť jednotlivých položiek. Okrem toho musí taktiež sledovať stav

pamäťových bufferov v endpoint bloku, a podľa toho riadiť i samotné vysielanie. Nemôže totiž začať vysielať ak endpoint blok nemá dostatočné miesto pre daný typ transakcie (sledované na signále TRN_TBUF_AV).

- Začiatočným stavom je pre automat stav ***s_hdr0.*** V tomto sa riadi vysielanie prvej časti hlavičky, pokiaľ to dovoľuje stav na zbernici. Z tohto stavu môže prejsť do dvoch stavov.
- Jeden z ďalších stavov je určený výhradne pre čítacie požiadavky. Je ním stav ***s_hdr1*** a automat do tohto prechádza teda v prípade, že transakcia neobsahuje dáta. Po odoslaní druhej hlavičky prechádza z tohto stavu opäť do počiatočného stavu (s_hdr0).
- Z počiatočného stavu je možné taktiež prejsť do stavu ***s_hdr1_data*****,** kedy sa odosielajú i dáta. Tieto dáta môžu však stále vojsť do druhej hlavičky, no a preto opäť existuje v takomto prípade možnosť návratu do počiatočného stavu. Samozrejme len v tom prípade, že boli dáta korektne prerovnané.
- Pokiaľ je dát viac, prechádza sa do stavu ***s_data,*** odkiaľ sa riadi vysielanie dát. Pokiaľ nastane odoslanie poslednej časti dát, prechádza sa do počiatočného stavu. Do tohto stavu pre odosielanie prvej časti hlavičky sa prechádza i v prípade rozdelenia transakcií. Logika pre výpočet dĺžky totiž zabezpečí, korektne nastavovanie informácie o dátach (LAST_DATA). Navyše vstupný riadiaci automat čaká v stave spracovania všetkých dát, a preto ponecháva všetky potrebné stavové registre k dispozícii pre odosielanie nového paketu.

6.3 Dokončovací buffer

Dokončovací, alebo inak i *Completion buffer*, slúži k ukladaniu riadiacich informácií pri čítacej transakcii, ktoré sa využívajú neskôr pri tvorbe príslušnej dokončovacej transakcie.

Čítanie môže byť lokálne, kedy sa číta z lokálneho adresového priestoru internej zbernice, a potom sa využíva *lokálny dokončovací buffer.* Do tohto buffera sa zapisuje v prípade príjmu čítacej PCI Express transakcie z generátora paketov internej zbernice a číta sa z neho v prípade tvorby príslušnej odpovedi v dekodéru paketov internej zbernice.

Čítanie môže byť taktiež i globálne, kedy sa číta z globálneho adresového priestoru (čiže mimo FPGA), a potom sa využíva *globálny dokončovací buffer.* Do tohto sa zapisuje v dekodéru paketov internej zbernice a číta sa z neho v generátore paketov internej zbernice pri tvorbe odpovede na čítanie. Zároveň ale je potrebné brať ohľad na to, že z tejto strany generátora paketov internej zbernice sa do globálneho dokončovacieho buffera bude taktiež zapisovať, a to z dôvodov, popísaných vyššie.

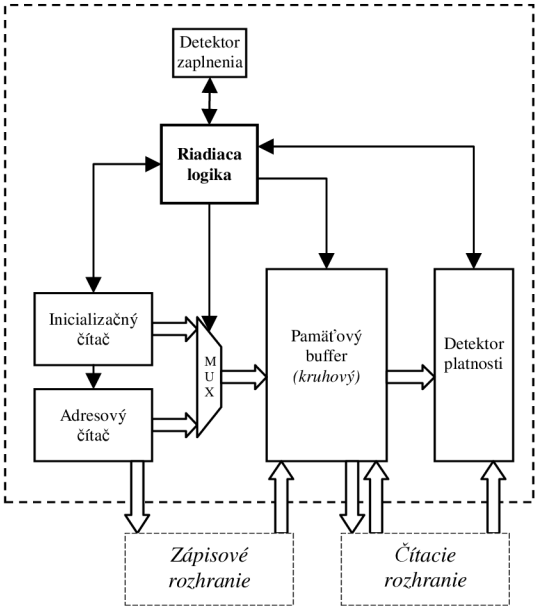
Pri vkladaní informácií do dokončovacieho buffera nám tento navráti novú hodnotu tagu (identifikácie). Na tomto mieste je potrebné si uvedomiť, že existujú dva základné druhy tagov:

- *tag* identifikujúci transakciu *na internej zbernici* - taktiež lokálny tag a
- *tag* identifikujúci transakciu na *PCI Express systémovej zbernici* – taktiež globálny tag.

Oba sú navzájom nezameniteľné. Pri príjme PCI Express paketu sa tag tejto čítacej transakcie ukladá do lokálneho dokončovacieho buffera, zatiaľ čo je generovaný nový (lokálny) tag internej zbernice. Naopak pri príjme transakcie z internej zbernice sa tag čítacej transakcie ukladá do globálneho dokončovacieho buffera a vygenerovaný je nový (globálny) tag použitý do paketu pre transakciu na zbernici PCI Express. Tagy sú generované inkrementáciou čítača naposledy použitého tagu. Takýto čítač tvorí vlastne aktuálnu adresu voľného miesta v bufferi.

6.3.1.1 Architektúra dokončovacieho buffera

Pre jednoduchší návrh bola komponenta dokončovacieho buffera rozdelená na globálny a lokálny buffer. Oba z nich sú však čo sa týka riadenia a podkomponet z ktorých sa skladajú veľmi podobné, preto je v nasledujúcom texte popísaná iba ich obecná architektúra. Globálny buffer sa líši tým, že má tzv. LAST_FRAGMENT, ktorým sa riadi úprava adries spätne ukladaných do buffera. Je to preto, lebo odpoveď na čítaciu požiadavku do globálneho priestoru môže prísť fragmentovaná a je teda potrebné adresy s prichádzajúcimi transakciami upravovať. Obrázok 6.12 ukazuje obecnú blokovú architektúru.



Obrázok 6.12: ***Bloková architektúra dokončovacích bufferov.***

Hlavným prvkom komponenty je samotný pamäťový buffer, do ktorého sa ukladajú potrebné dáta. Okrem neho predstavuje určitú pamäť i tzv. detektor platnosti. Ten obsahuje jednobitovú informáciu ku každému položke pamäťového buffera, ktorá indikuje použitie danej položky v buffri. Táto doplňujúca logika je potrebná kvôli tomu, že odpovede na čítania môžu prísť v ľubovoľnom poradí a je potrebné zaistiť aby nedošlo k prepisu už raz uloženej položky. Taktiež slúži pre adresový čítač, aby ten vždy vedel, ktoré položky má vynechať pre zápis. Takto sa jednoduchým spôsobom

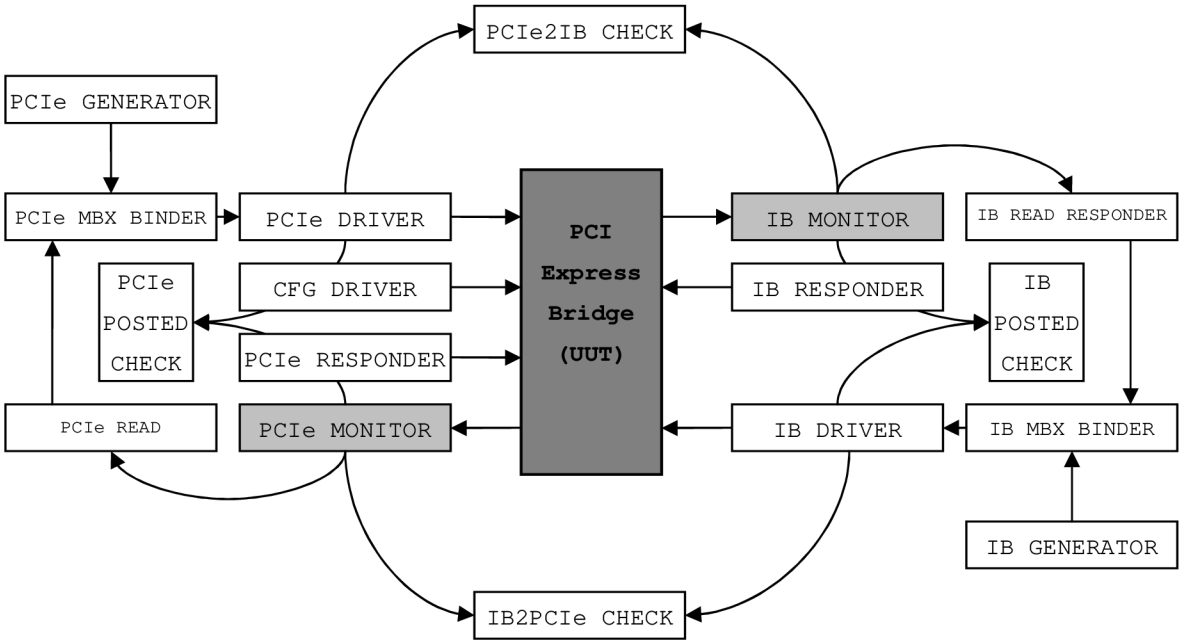
zabezpečí hodnota adresového čítača, ktorá bude vždy korektná a použiteľná pre jedinečnú identifikáciu transakcie (TAG). Okrem adresového čítača, ktorý teda adresuje určité miesto v pamäti, je v komponente ešte tzv. inicializačný čítač, ktorý napr. po resete správne odznačí použite všetkých doteraz použitých položiek. Posledným z čítačov je tzv. detektor zaplnenia, ktorý informuje o zaplnení pamäte dostatočne dopredu podľa nastavených parametrov (napr. 4 transakcie pred zaplnením). Takto sa propaguje dostatočne dopredu informácia o zaplneniu bufferov pričom na transakcie, ktoré sú ešte v linke zostane voľné miesto. Celú komponentu riadi opäť riadiaca logika, ktorá prepína medzi inicializačnou fázou a fázou bežnej činnosti a zároveň riadi korektný zápis alebo čítanie z bufferov.

7 Testovanie

7.1 Simulačné prostredie

Návrh komplexných firmwarových modulov akým je i jednotka riadenia protokolu PCI Express, sa v dnešnej dobe nezaobíde bez simulácií. Preto boli vytvorené pokročilé objektové simulačné modely a testbenche v jazyku SystemVerilog [10]. Vytvorený firmware je overený s využitím princípov *tzv. constraint random testing*. To znamená, že do testovanej komponenty sú vysielané náhodné (random) avšak parametrizovateľné (constraint) transakcie a sú skúmané odpovede systému oproti očakávaným. V rámci behu sa meria funkčné pokrytie *(functional coverage)* a pokrytie kódu *(code coverage)* [3]. Zatiaľ čo pokrytie kódu sleduje aká časť zdrojového kódu bola prevedená, funkčné pokrytie sleduje sofistikovanejšie parametre. Medzi ne patrí napr. aká časť kľúčových funkcií bola vykonaná, do akej miery boli overené okrajové situácie, aké konfigurácie návrhu boli testované alebo aká časť povolených kombinácií signálov na rozhraniach bola počas simulácie nastavená (tzv. command coverage).

Celé prostredie poslúži v prvých fázach pre jednoduchšiu simuláciu celku (dodávané simulačné modely sú mnohokrát ťažkopádne a simulácia s nimi zaberá množstvo času), no neskôr i na celkové overenie správnosti návrhu. Popis prostredia pre jednotku riadenia toku PCI Express *(*v tomto prípade už *UUT – Unit Under Test)* názorne ukazuje Obrázok 7.1:



Obrázok 7.1: ***Verifikačné/simulačné prostredie jednotky pre riadenie protokolu PCI Express.***

Prostredie pracuje na vysokej úrovni abstrakcie, a i preto je výhodnejšie si ho rozdeliť na viac funkčných blokov, ktoré plnia špecifické funkcie:

PCIe Generator, IB Generator – obecné bloky, ktoré majú na starosti generovanie náhodných, avšak dobre parametrizovateľných transakcií typu PCI Express alebo IB v podobe abstraktných tried.

PCIe MBX Binder, IB MBX Binder – bloky zhromažďujúce na oboch stranách transakcie z generátora transakcií a z bloku pre generovanie odpovedí na čítacie požiadavky.

PCIe Driver, IB Driver – prijíma transakcie v podobe abstraktnej triedy a transformuje ich do konkrétnych signálov na pripojenej zbernici rozhrania testovanej jednotky.

CFG Driver – simuluje konfiguračné rozhranie endpoint bloku.

PCIe Monitor, IB Monitor – komponenty sledujú rozhranie PCI Express Bridge a vytvárajú abstraktnú triedu. Po jej vygenerovaní ju zasielajú do komponenty pre kontrolu prevodu transakcií *(tzv. Scoreboard),* do komponenty pre kontrolu tzv. Posted transakcií a v prípade čítacej požiadavky do komponenty pre generovanie čítacej odpovede.

PCIe Read Responder, IB Read Responder – na základe stanovených parametrov ovládajú signály DST_RDY_N.

PCIe Posted Check, IB Posted Check – predstavujú komponenty, ktoré sa starajú o kontrolu tzv. Posted transakcií internej zbernice alebo zbernice PCI Express.

PCIe2IB Check, IB2PCIe Check – predstavuje ***Scoreboard.*** Do týchto modulov sa z jednej strany (z generátora) transakcie ukladajú a z druhej strany (z monitora) korešpondujúce transakcie mažú. Na konci úspešného testu musí byť teda Scoreboard vždy prázdny.

Výsledkom behu simulačného prostredia je počet správne prevedených náhodných transakcií (vyžaduje sa 100% správnosť) a veľkosť pokrytia. Počet transakcií je samozrejme parametrizovateľný, a pre hodnotu 100 miliónov transakcií sa dosahuje 100% pokrytia kódu a 98,6% funkčného pokrytia. Pre úplne pokrytie by bolo potrebné upraviť testy vhodným spôsobom, ale je potrebné podotknúť i to, že pre takýto počet transakcií je potrebné si na výsledok počkať zhruba 32 hodín na výkonnejšom počítači[2]. Výhodou parametrizovateľnosti prostredia je však to, že je možné si

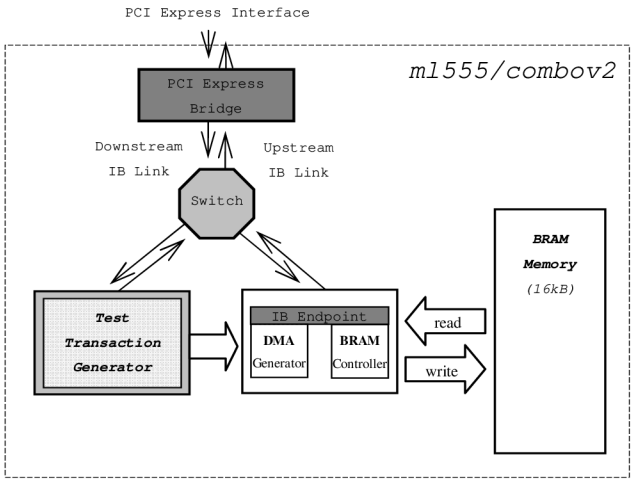
[2] 2 procesorový (každý procesor so 4 jadrami) *Xeon@2.33GHz*, 4GB operačnej pamäti

taktiež vytvoriť test pre konkrétne (napr. problémové) typy transakcií, a takto veľmi rýchlo ladiť v priebehu niekoľkých málo minút.

7.2 Testovanie v reálnom prostredí

Po úspešnej simulácií bolo potrebné funkcionalitu obvodu overiť i v reálnom prostredí, čiže priamo v FPGA. Na toto poslúžia karty ML555 alebo COMBOv2, obe osadné čipmi Virtex-5, a teda s podporou PCI Express rozhrania v podobe spomínaného endpoint bloku (Kapitola 4). Pre samotné testovanie síce môže poslúžiť už existujúci firmware pre FPGA, ktorý je vyvíjaný v rámci projektu Liberouter *(NIC/NIFIC, FlowMon)* [8], avšak tieto sú natoľko komplexné, že je jednoduchšie si vytvoriť samostatný testovací obvod, ktorý bude oproti už spomínaným omnoho jednoduchší a najmä prehľadnejší. Takto sa zjednoduší nielen testovanie, ale i proces syntézy a napokon i routovania počas úvodných fáz testovania. Obrázok 7.2 názorne ukazuje takýto dizajn.

Hlavnou a testovanou komponentou je jednotka riadenia protokolu PCI Express (na obrázku vyznačený najtmavšou farbou). Táto je pripojená na jednej strane k endpoint bloku (PCI Express rozhranie) a na druhej strane k Switchu internej zbernice (viď Kapitola 5). Tento oddeľuje komunikáciu do tzv. generátora transakcií (*Test Transaction Generator* na obrázku) a do blokovej pamäte (*BRAM Memory* na obrázku), ktorej veľkosť môže byť nastaviteľná. Jednotkou, ktorá zabezpečuje komunikáciu s touto pamäťou a generátorom transakcií je IB Endpoint (viď Kapitola 5). Ten generuje na základe došlej IB transakcie priamo zo Switch jednotky (a teda priamo z PCI Express Bridge) alebo z generátora čítanie alebo zápis z/do pamäte. Pomocou spomínanej Switch komponenty sa jednoducho oddelia pamäťové priestory bez nutnosti použiť zložitejších dekodérov v Endpoint jednotke alebo v generátore transakcií.



Obrázok 7.2: ***Testovací dizajn pre karty ml555/combov2.***

Ako generátor transakcií sú implementované dve jednotky: jedna pracuje v r*ežime overenia korektnosti* a druhá v *režime merania maximálnej priepustnosti*. Obe z testovacích jednotiek sa musia

ale samozrejme vhodne inicializovať, čo vedie na komunikáciu v podobe jednoduchých (vždy zarovnaných s dĺžkou 4 B) zápisových transakcií cez testovanú jednotku (PCI Express Bridge). Stav testovacích jednotiek sa naopak zisťuje jednoduchými čítacími požiadavkami (opäť vždy zarovnané s dĺžkou 4 B), čo opäť vedie na komunikáciu testovanú jednotku, pričom v tomto prípade sa navyše generuje z testovacej jednotky odpoveď na čítanie.

7.2.1 Test korektnosti

Tento test prebiehal v nasledujúcich krokoch:

1. inicializácia generátora transakcií zo softvéru (globálna adresa, lokálna adresa, typ použitej transakcie – G2LR alebo L2GW a jej veľkosť)
2. spustenie generátora, ktorý danú transakciu vygeneruje
3. čakanie na ukončenie transakcie kontrolou stavového registra
4. kontrola správnosti transakcie vypísaním obsahov pamätí RAM a BRAM v FPGA
    a. v prípade chyby ukončenie testu
    b. v prípade úspešného ukončenia vrátenie sa k prvému kroku

Uvedeným spôsobom boli otestované všetky typy *32 bitových transakcií* [3], ktorých zoznam nasleduje:

- **čítanie z FPGA** priestoru **do RAM** o ľubovoľnej dĺžke
    - zarovnané zdrojové a cieľové adresy
    - nezarovnané zdrojové a cieľové adresy
- **zápis do FPGA** priestoru **z RAM** o ľubovoľnej dĺžke
    - zarovnané zdrojové a cieľové adresy
    - nezarovnané zdrojové a cieľové adresy
- **BM čítanie z RAM do FPGA** (G2LR)
    - nerozdelené transakcie
    - rozdelené transakcie
    - nezarovnané prístupy (rozdelené i nerozdelené)
- **BM zápis do RAM z FPGA** (L2GW)
    - nerozdelené transakcie
    - rozdelené transakcie
    - nezarovnané prístupy (rozdelené i nerozdelené)

Všetky typy vymenovaných transakcií boli popísaným spôsobom úspešné overené.

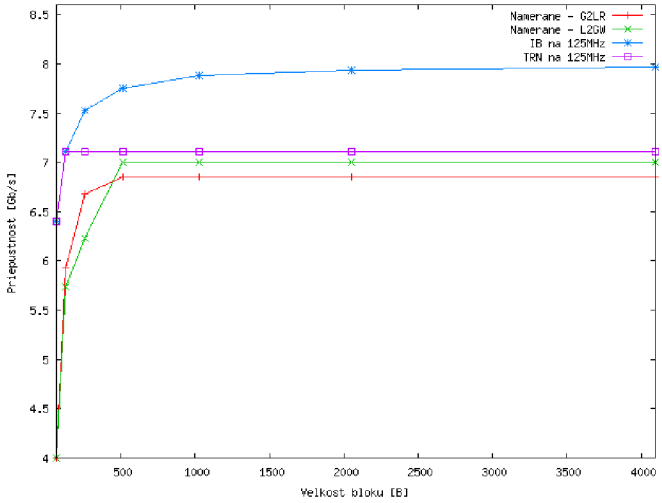
[3] Pre otestovanie 64 bitových transakcií je potrebné mať k dispozícii i 64 bitový operačný systém spolu s 64 bitovými ovládačmi rozširujúcej karty.

7.2.2 Test maximálnej priepustnosti

Tento test prebiehal v nasledujúcich krokoch:

1. inicializácia generátora transakcií zo softvéru (globálna adresa, lokálna adresa, typ – G2LR alebo L2GW, veľkosť bloku a celkový počet transakcií)
2. štart hardvérového (HW) čítača a generovania transakcií z testovacej jednotky
3. čakanie na ukončenie testu aktívnym čakaním (HW čítač po ukončení prenosu zastaví – nedochádza tak k skresleniu neskorších výpočtov)
4. otestovanie zo softvéru, či už dobehli všetky transakcie
    a. ak nie, nastala chyba (narušil sa počet prenesených transakcií cez testovanú jednotku) a test skončí
    b. ak dobehli, prečítanie hodnoty HW čítača pre zistenie doby prenosu v taktoch
5. výpočet prenosovej rýchlosti v jednotkách Gb/s, korektné ukončenie testu

Pomocou uvedeného postupu bola odmeraná zápisová (L2GW) a čítacia (G2LR) rýchlosť pre rôzne veľkosti blokov (dĺžky) prenesených paketov vždy pre dostatočný počet transakcií (tak aby výsledné množstvo prenesených dát bolo približne 40 GB). Rýchlosť u zápisu a čítania je odlišná z toho dôvodu, že jednotka IB Endpoint signalizuje iným spôsobom ukončenie transakcie, a tým sa i HW čítač pozastaví v iný okamžik. Toto chovanie nie je možné ale obmedziť. Nasledujúci Obrázok 7.3 zobrazuje v podobe grafu nameranú priepustnosť v závislosti od veľkosti blokov:



Obrázok 7.3: ***Nameraná priepustnosť jednotky pre riadenie protokolu PCI Express.***

Okrem nameraných hodnôt (Namerane - G2LR, Namerane - L2GW) je v grafe možné vidieť i maximálnu teoretickú priepustnosť internej zbernice (IB na 125 MHz) a zbernice na PCI Express rozhraní (TRN na 125 MHz). Pre toto transakčné rozhranie je maximálna teoreticky dosiahnuteľná

priepustnosť pre danú frekvenciu nemenná – 7,1 Gb/s. Je to z toho dôvodu, že na PCI Express rozhranie nie je možné zasielať pakety väčšie ako *tzv. MAX_PAYLOAD_SIZE* (viď Kapitola 6.2.2), ktorá je v tomto prípade rovná hodnote 128 B [4].Všetky konkrétne hodnoty nameranej (G2LR, L2GW) i vypočítanej (IB na 125 MHz, TRN na 125 MHz) priepustnosti sú zobrazené v Tabuľke 7.1:

Veľkosť bloku [B]	G2LR	L2GW	IB na 125 MHz	TRN na 125 MHz
4096	6.851179	7.005411	7.968872	7.111111
2048	6.851108	7.005309	7.937984	7.111111
1024	6.851179	7.004877	7.876923	7.111111
512	6.850003	7.000120	7.757576	7.111111
256	6.679127	6.230441	7.529412	7.111111
128	5.926788	5.735433	7.111111	7.111111
64	4.003636	4.000370	6.400000	6.400000

Tabuľka 7.1: ***Priepustnosť jednotky pre riadenie protokolu PCI Express.***

[4] Hodnota závislá od čipovej sady použitej v základnej doske počítača.V tomto prípade *Intel 5000.*

8 Záver

Cieľom tejto práce bolo navrhnúť jednotku riadenia toku PCI Express pre systém interných zberníc na čipe FPGA. Jednotka poskytuje komunikáciu s ostanými prvkami počítača mapovanými do hostiteľskej pamäti cez systémovú zbernicou a vytvára tak koreňový prvok celého prepojovacieho systému. Pri návrhu bol kladený dôraz na modularitu, využitie zdrojov ale i na priepustnosť. Je však bohužiaľ pravdou to, že posledné dva požiadavky idú výrazne proti sebe, čo sa prejavilo aj pri samotnom návrhu a nakoniec i vo výsledkoch po implementácií. Naopak zabezpečiť modularitu na základe dobre rozdeleného návrhu nepredstavovalo príliš veľký problém.

Pre pochopenie skúmanej problematiky boli naštudované potrebné materiály, ktoré sa zaberajú technológiami hradlových polí FPGA (Virtex-5), existujúcimi riešeniami systémov zberníc na týchto čipoch a systémom zberníc a prenosom dát v samotnom hostiteľskom počítači. Tu hlavným predmetom záujmu bolo transakčné spracovanie dátových prenosov na zbernici PCI Express pomocou paketov. Získané vedomosti boli zhrnuté v teoretickom rozbore úlohy a v kapitolách pojednávajúcich o danej problematike, z ktorých čitateľ získa veľmi kvalitný prehľad. Na základe tohto je možné následne pochopiť návrh, z ktorého vychádza i architektúra jednotky samotnej.

Výsledná implementovaná jednotka sa skladá zo 6 hlavných komponent popísaných v jazyku VHDL, ktoré po mapovaní na cieľovú technológiu Virtex-5 zaberajú spolu 1665 LUT. Je však taktiež potrebné podotknúť, že keďže endpoint blok je neodlučiteľnou súčasťou jednotky, neudáva toto číslo celkom výsledne obsadenie zdrojov systému pre komunikáciu so zbernicou PCI Express. Samotný endpoint blok zaberá zhruba 2100 LUT – a to ešte v závislosti na použitej verzii. Navyše verzia tohto hard+soft IP bloku od firmy Xilinx nehýbe len so zdrojmi (zatiaľ vždy nahor), ale i s funkcionalitou. Preto problémy, ktoré sa riešili v poslednom období spracovania práce, sa týkali výhradne problémov spojených s týmto blokom (za obdobie spracovania tejto práce sa vydali 3 verzie spolu s dohromady 8 záplatami), čo spôsobilo i výrazne spomalenie prechodu jednotky na 250 MHz. Spomalenie nezahŕňalo len výmenu a testovanie s novými endpoint blokmi, ale i napríklad komunikáciu so samotným Xilinx týmom. V súčasnej dobe je teda jednotku možné taktovať na frekvencii 158 MHz, avšak v prípade povolených frekvencií na transakčnom rozhraní endpoint bloku sa dostávame iba na nižšiu najbližšiu možnú, teda na 125 MHz. Z toho teda vyplýva i polovičná priepustnosť oproti použitiu transakčného rozhrania na frekvencii 250 MHz. Výsledná nameraná priepustnosť 7 Gb/s zodpovedá teda očakávaným predpokladom.

Pri záverečnom testovaní a ladení chýb po prvotnej implementácií jednotky výrazne pomohlo simulačné prostredie implementované v jazyku SystemVerilog. Pre jeho implementáciu ale bolo potrebných už i iných členov z týmu projektu Liberouter. Je to najmä z toho dôvodu, aby si implementátor nezaviedol rovnaké, či podobné návrhové chyby do simulačného prostredia i do testovacej jednotky kvôli napr. nesprávnemu pochopeniu štandardu. Samotné testovanie napokon

prináša možnosť v niekoľkých málo jednotkách minút overiť správnosť konkrétneho typu transakcií. Poskytuje teda výrazne zníženie časovej náročnosti simulácií oproti použitiu endpoint block simulačného bloku (od firmy Xilinx), kde iba preklad tohto prostredia trvá na rovnakom počítači 30 minút pre jednu jedinú transakciu. Po dôsledných simuláciách bolo možné jednotku testovať priamo na reálnych kartách. Tu sa doladili posledné nedostatky a hlavne problémy so spomínaným endpoint blokom za pomoci analyzátora PCI Express zbernice a iných pokročilých nástrojov. Napokon je celá jednotka v súčasnej dobe používaná v reálnych aplikáciách, v prostredí vysokorýchlostných sietí na kartách COMBOv2.

Ďalšia prácou, ktorá môže pokračovať je zrýchľovanie jednotky na vyššiu frekvenciu (250 MHz). Toto bude náročnou výzvou z pohľadu toho, že i keď síce limit technológie Virtex-5 je do 500 MHz, pri väčšom obsadení čipu nastáva problém s rozmiestnením jednotlivých častí tak, aby bolo časovanie splnené. Preto bude potrebné sa zaoberať napr. i ručným rozmiestňovaným jednotlivých prvkov na čipe samotnom.

Literatúra

[1] Anderson, D., Budruk, R., Shanley, T.: *PCI Express System Architecture.* MindShare Inc., Addison Wesley. September 2003. ISBN 0-321-15630-7. 1120s.

[2] Aspinwall, J.: *IRQ, DMA a I/O. Předcházení a řešení konfliktů v konfiguraci počítače.* Praha, Computer Press. 2000. ISBN 80-72-26264-5. 288s.

[3] Bergeron, J., et al.: *Verification Methodology Manual for SystemVerilog.* Springer. 2005. ISBN 978-0-387-25538-5. 510s.

[4] Korček, P.: *Generovanie pseudonáhodných čísel v FPGA.* [Bakalárska práca]. FIT VUT v Brně, 2007.

[5] Kořenek, J., Martínek T.: *Návrh externích adaptéru a vestavěných systému.* [Studijní materiály k předmětu NAV]. FIT VUT Brně, 2007.

[6] Kotásek, Z.: *Periferní zařízení.* [Studijní materiály k předmětu IPZ]. FIT VUT v Brně, 2007.

[7] Kotásek, Z.: *Technika personálních počítačů*. [Studijní materiály k předmětu ITP]. FIT VUT v Brně, 2006.

[8] *Liberouter project* [online]. [cit. december 2008]. URL: <http://www.liberouter.org>.

[9] PCI-SIG: *PCI Express Base Specification.* Revision 1.0a, April 2003.

[10] Spear, Ch.: *SystemVerilog for Verification.* A Guide to Learning the Testbench Language Features. Synopsys, Inc., Springer. 2006. ISBN 0-387-27036-1. 326s.

[11] Xilinx, Inc.: *Aurora.* Product Specification v3.0, September 2008.

[12] Xilinx, Inc.: *Endpoint Block Plus for PCI Express*. Product Specification v1.6, March 2004.

[13] Xilinx, Inc.: *LocalLink Interface Specification.* Product Specification v1.0, September 2006.

[14] Xilinx, Inc.: *Virtex-5 FPGA* [online]. User Guide v4.4, 2008 [cit. december 2008]. URL: <http://www.xilinx.com/support/documentation/user_guides/ug190.pdf>.

[15] Xilinx, Inc.: *Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs* [online]. User Guide v1.4, 2008 [cit. december 2008]. URL: <http://www.xilinx.com/support/documentation/boards_and_kits/ug201.pdf>.

[16] Wikipedia contributors: *PCI Express* [online]. Wikipedia, The Free Encyclopedia [cit. december 2008]. URL: <http://en.wikipedia.org/w/index.php?title=PCI_Express&oldid=261674900>.

Zoznam príloh

Príloha A

Rozhranie dekodéra paketov systémovej zbernice

Generické parametre	Popis
INPUT_REG_EN	*Zapnutie vstupného registrového stupňa.* Predvolené: FALSE
OUTPUT_REG_EN	*Zapnutie výstupného registrového stupňa.* Predvolené: FALSE
BAR0_REMAP[31:0]	*Bázová adresa transakcií z PCI Express do IB pre BAR0.*
BAR1_REMAP[31:0]	*Bázová adresa transakcií z PCI Express do IB pre BAR1.*
BAR2_REMAP[31:0]	*Bázová adresa transakcií z PCI Express do IB pre BAR2.*
BAR3_REMAP[31:0]	*Bázová adresa transakcií z PCI Express do IB pre BAR3.*
BAR4_REMAP[31:0]	*Bázová adresa transakcií z PCI Express do IB pre BAR4.*
BAR5_REMAP[31:0]	*Bázová adresa transakcií z PCI Express do IB pre BAR5.*
EXP_ROM_REMAP[31:0]	*Bázová adresa transakcií z PCI Express do IB pre oblasť rozšírenej pamäti ROM.*
BAR0_MASK[31:0]	*Maska transakcií z PCI Express do IB pre BAR0.*
BAR1_MASK[31:0]	*Maska transakcií z PCI Express do IB pre BAR1.*
BAR2_MASK[31:0]	*Maska transakcií z PCI Express do IB pre BAR2.*
BAR3_MASK[31:0]	*Maska transakcií z PCI Express do IB pre BAR3.*
BAR4_MASK[31:0]	*Maska transakcií z PCI Express do IB pre BAR4.*
BAR5_MASK[31:0]	*Maska transakcií z PCI Express do IB pre BAR5.*
EXP_ROM_MASK[31:0]	*Maska transakcií z PCI Express do IB pre oblasť rozšírenej ROM.*

Názov signálu vstupného rozhrania	Typ	Popis
TRN_RESET_N	Vstup	*Reset z transakčného rozhrania.*
TRN_CLK	Vstup	*Hodinový signál z transakčného rozhrania (fixne 125 MHz alebo 250 MHz)*
TRN_RBAR_HIT_N[6:0]	Vstup	*BAR aktuálnej transakcie ([0] – BAR0, [6] – Rozšírená ROM)*
TRN_RD[63:0]	Vstup	*Vstupné dáta*
TRN_RREM_N[7:0]	Vstup	*Určuje platnosť vstupného dátového slova (možnosť 00000000 – data platné na [63:0] alebo 00001111 platné na [63:32])*
TRN_RSOF_N	Vstup	*Signalizácia začiatku paketu*
TRN_REOF_N	Vstup	*Signalizácia konca paketu*
TRN_RSRC_RDY_N	Vstup	*Signalizácia pripravenosti vysielať dáta do komponenty*
TRN_RDST_RDY_N	Výstup	*Signalizácia pripravenosti prijímať dáta do komponenty*
TRN_RERRFWD_N	Vstup	*Signalizácia prítomnosti neopraviteľnej chyby v práve prijímanom pakete. Paket sa zahadzuje*
TRN_RNP_OK_N	Výstup	*Signalizácia schopnosti prijímať Non-Posted transakcie*
TRN_RCPL_STREAM_N	Výstup	*Povolenie tzv. streaming módu. Nevhodné pre bežné použitie!*
TRN_RSRC_DSC_N	Vstup	*Prerušenie prenosu v priebehu aktívnej činnosti*

TRN_RFC_PD_AV[11:0]	Vstup	*Dostupné Flow Control (FC) kredity pre dáta Posted trans.*
TRN_RFC_PH_AV[7:0]	Vstup	*Dostupné FC kredity pre hlavičky Posted transakcií*
TRN_RFC_NPD_AV[11:0]	Vstup	*Dostupné FC kredity pre dáta Non-Posted transakcií*
TRN_RFC_NPH_AV[7:0]	Vstup	*Dostupné FC kredity pre hlavičky Non-Posted transakcií*

Názov signálu výstupného rozhrania	Typ	Popis
DEC_DATA[63:0]	Výstup	*Výstupné dáta*
DEC_SOP_N	Výstup	*Signalizácia začiatku paketu*
DEC_EOP_N	Výstup	*Signalizácia konca paketu*
DEC_SRC_RDY_N	Výstup	*Signalizácia pripravenosti vysielať dáta ďalej*
DEC_DST_RDY_N	Vstup	*Signalizácia pripravenosti ďalšej komponenty prijímať dáta*
READ_TRANS_EN_N	Vstup	*Signalizácia povolenia prijímať čítacie požiadavky (závisí od stavu buffera jednotky)*

Príloha B

Rozhranie generátora paketov internej zbernice

Generické parametre	Popis
INPUT_REG_EN	*Zapnutie vstupného registrového stupňa.* Predvolené: FALSE
OUTPUT_REG_EN	*Zapnutie výstupného registrového stupňa.* Predvolené: TRUE
ENABLE_ALIGN_UNIT	*Zapnutie prerovnávacej jednotky*

Názov signálu vstupného rozhrania	Typ	Popis
TRN_RESET_N	Vstup	*Reset z transakčného rozhrania.*
TRN_CLK	Vstup	*Hodinový signál z transakčného rozhrania (fixne 125 MHz alebo 250 MHz)*
DEC_DATA[63:0]	Vstup	*Výstupné dáta*
DEC_SOP_N	Vstup	*Signalizácia začiatku paketu*
DEC_EOP_N	Vstup	*Signalizácia konca paketu*
DEC_SRC_RDY_N	Vstup	*Signalizácia pripravenosti vysielať dáta do komponenty*
DEC_DST_RDY_N	Výstup	*Signalizácia pripravenosti prijímať dáta do komponenty*

Názov signálu zápisového rozhrania LCB	Typ	Popis
LCB_TAG[7:0]	Výstup	*TAG transakcie PCI Express*
LCB_WR_0	Výstup	*Zápis č.1*
LCB_REQ_ID[15:0]	Výstup	*REQUESTER_ID (Bus, Device, Function)*
LCB_WR_1	Výstup	*Zápis č.2*
LCB_ALLOW	Vstup	*Povolenie zápisu*
LCB_LOCAL_TAG[7:0]	Vstup	*Vygenerovaný lokálny TAG (zápisová adresa)*
LCB_FULL	Vstup	*Lokálny buffer plný*
LCB_TRANS_EN_N	Vstup	*Povolenie transakcii*

Názov signálu čítacieho rozhrania GCB	Typ	Popis
GCB_LOCAL_ADDR[31:0]	Vstup	*Uložená lokálna adresa*
GCB_LOCAL_TAG[7:0]	Vstup	*Uložený lokálny TAG*
GCB_RD	Výstup	*Čítacia požiadavka*
GCB_GLOBAL_TAG	Výstup	*Globálny TAG (čítacia adresa)*
GCB_LAST_CPL	Výstup	*Identifikácia poslednej transakcie*
GCB_LEN_CPL	Výstup	*Dĺžka transakcie*

Názov signálu výstupného rozhrania	Typ	Popis
IB_DATA[63:0]	Výstup	*Výstupné dáta*
IB_SOP_N	Výstup	*Signalizácia začiatku paketu*
IB_EOP_N	Výstup	*Signalizácia konca paketu*
IB_SRC_RDY_N	Výstup	*Signalizácia pripravenosti vysielať dáta ďalej*
IB_DST_RDY_N	Vstup	*Signalizácia pripravenosti ďalšej komponenty prijímať dáta*

Príloha C

Rozhranie dekodéra paketov internej zbernice

Generické parametre	Popis
INPUT_REG_EN	*Zapnutie vstupného registrového stupňa.* Predvolené: FALSE

Názov signálu vstupného rozhrania	Typ	Popis
TRN_RESET_N	Vstup	*Reset z transakčného rozhrania*
TRN_CLK	Vstup	*Hodinový signál z transakčného rozhrania (fixne 125 MHz alebo 250 MHz)*
IB_DATA[63:0]	Vstup	*Vstupné dáta*
IB_SOP_N	Vstup	*Signalizácia začiatku paketu*
IB_EOP_N	Vstup	*Signalizácia konca paketu*
IB_SRC_RDY_N	Vstup	*Signalizácia pripravenosti vysielať dáta do komponenty*
IB_DST_RDY_N	Výstup	*Signalizácia pripravenosti prijímať dáta do komponenty*

Názov signálu zápisového rozhrania GCB	Typ	Popis
GCB_LOCAL_ADDR[31:0]	Výstup	*Lokálna adresa na uloženie*
GCB_LOCAL_TAG[7:0]	Výstup	*Lokálny TAG na uloženie*
GCB_WR	Výstup	*Žiadosť o zápis*
GCB_WR_ALLOW	Vstup	*Povolenie zápisu*
GCB_RTAG	Vstup	*Vygenerovaný globálny TAG (zápisová adresa)*

Názov signálu čítacieho rozhrania LCB	Typ	Popis
LCB_TAG[7:0]	Vstup	*TAG transakcie PCI Express*
LCB_REQ_ID[15:0]	Vtup	*REQUESTER_ID (Bus, Device, Function)*
LCB_RD	Výstup	*Čítacia požiadavka*
LCB_RTAG	Výstup	*Lokálny TAG (čítacia adresa)*

Názov signálu výstupného rozhrania	Typ	Popis
GEN_DATA[63:0]	Výstup	*Výstupné dáta*
GEN_SOP_N	Výstup	*Signalizácia začiatku paketu*
GEN_EOP_N	Výstup	*Signalizácia konca paketu*
GEN_SRC_RDY_N	Výstup	*Signalizácia pripravenosti vysielať dáta ďalej*
GEN_DST_RDY_N	Vstup	*Signalizácia pripravenosti ďalšej komponenty prijímať dáta*
GEN_DW4	Výstup	*Signalizácia prítomnosti DW3/DW4 hlavičky*
GEN_DW4_VLD	Výstup	*Platnosť stavu DW3/DW4 hlavičky*

Príloha D

Rozhranie generátora paketov systémovej zbernice

Generické parametre	Popis
OUTPUT_REG_EN	*Zapnutie výstupného registrového stupňa.* Predvolené: TRUE

Názov signálu vstupného rozhrania	Typ	Popis
TRN_RESET_N	Vstup	*Reset z transakčného rozhrania.*
TRN_CLK	Vstup	*Hodinový signál z transakčného rozhrania (fixne 125 MHz alebo 250 MHz)*
GEN_DATA[63:0]	Vstup	*Vstupné dáta*
GEN_SOP_N	Vstup	*Signalizácia začiatku paketu*
GEN_EOP_N	Vstup	*Signalizácia konca paketu*
GEN_SRC_RDY_N	Vstup	*Signalizácia pripravenosti vysielať dáta do komponenty*
GEN_DST_RDY_N	Výstup	*Signalizácia pripravenosti prijímať dáta do komponenty*
GEN_DW4	Vstup	*Signalizácia prítomnosti DW3/DW4 hlavičky*
GEN_DW4_VLD	Vstup	*Platnosť stavu DW3/DW4 hlavičky*
CFG_BUS_NUMBER[7:0]	Vstup	*Bus Number z konfiguračného rozhrania*
CFG_DEVICE_NUMBER[4:0]	Vstup	*Device Number z konfiguračného rozhrania*
CFG_FUNCTION_NUMBER[2:0]	Vstup	*Function Number z konfiguračného rozhrania*

Názov signálu výstupného rozhrania	Typ	Popis
TR_LNK_UP_N	Vstup	*PCI Express Linka je nakonfigurovaná a aktívna*
TRN_TD[63:0]	Výstup	*Výstupné dáta*
TRN_TREM_N[7:0]	Výstup	*Určuje platnosť vstupného dátového slova (možnosť 00000000 – data platné na [63:0] alebo 00001111 platné na [63:32])*
TRN_TSOF_N	Výstup	*Signalizácia začiatku paketu*
TRN_TEOF_N	Výstup	*Signalizácia konca paketu*
TRN_TSRC_RDY_N	Výstup	*Signalizácia pripravenosti vysielať dáta ďalej*
TRN_TDST_RDY_N	Vstup	*Signalizácia pripravenosti ďalšej komponenty prijímať dáta*
TRN_TERRFWD_N	Výstup	*Signalizácia prítomnosti neopraviteľnej chyby v práve vysielanom pakete*
TRN_RDST_DSC_N	Vstup	*Prerušenie prenosu v priebehu aktívnej činnosti*
TRN_TBUF_AV[2:0]	Vstup	*Dostupné vysielacie buffery ([0] – Non Posted Queue, [1] –Posted Queue, [2] – Completion Queue)*