VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PROCESNÍ JEDNOTKA PRO ANALÝZU A EDITACI SÍŤOVÉHO PROVOZU V FPGA

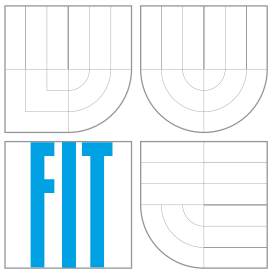
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN PAZDERA

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PROCESNÍ JEDNOTKA PRO ANALÝZU A EDITACI SÍŤOVÉHO PROVOZU V FPGA

PROCESSING UNIT FOR ANALYSIS AND MODIFICATION OF NETWORK TRAFFIC

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JAN PAZDERA
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MARTÍNEK
SUPERVISOR

BRNO 2007

Abstrakt

Tato práce se zabývá návrhem a implementací Procesní jednotky pro analýzu a editaci síťového provozu. Jejím úkolem je analyzovat příchozí síťový tok a provádět editace hlaviček paketů nezbytné pro jejich správné doručení. Navržená architektura má následující vlastnosti. Vychází z konceptu proudových procesorů, který umožňuje paralelní zpracování nezávislých elementů proudu (paketů). Dovoluje použít více proudových klientů pracujících nad stejným proudem, čímž umožňuje provádět několik výpočtů zároveň. Proudoví klienti mohou fungovat buď autonomně, nebo mohou být řízeni programem. Pakety jsou zpracovávány na základě vstupních metadat a po úpravě posílány na výstup. Implementace je provedena v jazyce VHDL. Cílovou technologií je programovatelné hradlové pole (FPGA).

Klíčová slova

Proudový procesor, Liberouter, Ethernet, IPv4, IPv6, FPGA, VHDL

Abstract

This paper deals with the design and implementation of the Processing Unit for Analysis and Modification of Network Traffic. The proposed unit is intended to analyse an incoming network traffic and perform packet header editations to provide the proper packet delivery. The designed architecture has the following characteristics. It is based on the stream processor concept which allows to process independent stream elements (i.e. packets) in parallel. Multiply stream clients can be used to process the same stream data concurrently. The stream clients can be driven either autonomously or by program. The packets are processed according to the incoming metadata and transmited to the output. The Processing Unit has been implemented in VHDL language. The target technology is Field Programmable Gate Array (FPGA).

Keywords

Stream processor, Liberouter, Ethernet, IPv4, IPv6, FPGA, VHDL

Citace

Jan Pazdera: Procesní jednotka pro analýzu a editaci síťového provozu v FPGA, diplomová práce, Brno, FIT VUT v Brně, 2007

Procesní jednotka pro analýzu a editaci síťového provozu v FPGA

Prohlášení

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

.....
Jan Pazdera
18. května 2007

Zadání diplomové práce/4741/2006/xpazde08

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů — Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Pazdera Jan, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Procesní jednotka pro analýzu a editaci síťového provozu v FPGA**
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s technologií programovatelných hradlových polí FPGA a dostupnými nástroji pro syntézu a implementaci obvodů do FPGA hradlových polí.
2. Seznamte se se strukturou síťových protokolů IPv4, IPv6 a způsoby jejich editace při průchodu sítí.
3. Navrhněte architekturu procesní jednotky pro analýzu a editaci síťového provozu s ohledem na implementaci v FPGA. Při návrhu uvažujte obecné použití této jednotky pro různé síťové aplikace.
4. Proveďte implementaci navrženého řešení v jazyce VHDL nebo HandleC a jeho funkci ověřte simulací.
5. Realizujte funkční prototyp procesní jednotky a ověřte její správnost na kartě COMBO6.
6. V závěru diskutujte vlastnosti vytvořené implementace a možnosti dalšího pokračování projektu.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1, 2 a 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Martínek Tomáš, Ing.**, UPSY FIT VUT
Datum zadání: 28. února 2006
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Bc. Jan Pazdera**
Id studenta: 49296
Bytem: Komenského 180, 742 01 Suchdol nad Odrou
Narozen: 22. 10. 1982, Přerov
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Procesní jednotka pro analýzu a editaci síťového provozu v FPGA
Vedoucí/školitel VŠKP: Martínek Tomáš, Ing.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....
Nabyvatel

.....
Autor

Obsah

Kapitola 1

Úvod

V posledním desetiletí jsme svědky výrazného rozvoje Internetu. Objem přenášených dat každým dnem roste a to společně se zvyšujícím se počtem uživatelů přináší stále vyšší nároky na systémy zabezpečující chod celosvětové sítě (dále jen uzly).

Data jsou na Internetu přenášena ve formě paketů, které kromě samotných dat nesou i kontrolní údaje, sloužící uzlům k jejich správné interpretaci a zpracování. Především se jedná o údaje nutné k správnému, bezpečnému a bezchybnému přenosu paketů od zdroje k cíli. To zahrnuje zejména směrování, řízení toku, opravné mechanismy při výskytu chyby a kódování/dekódování dat.

Struktura paketů je popsána různými standardy, které popisují komunikační protokoly mezi dvěma a více uzly a to v různých vrstvách (ISO/OSI). Dominantními protokoly v dnešním Internetu jsou zejména Ethernet v linkové vrstvě L2, IP v síťové vrstvě L3 (ve dvou verzích) a TCP/UDP v transportní vrstvě L4. K základním vlastnostem těchto protokolů patří především jejich robustnost a odolnost vůči výpadkům jednotlivých uzlů. To lze nejspolehlivěji zajistit tím, že každý uzel na základě aktuálního stavu svých výstupních linek zvolí tu, která je nejvhodnější pro přenos paketu. Důsledkem tohoto přístupu je nedeterministická cesta paketu, kterou není možné dopředu naplánovat v odesílacím uzlu. Každý uzel musí být proto vybaven mechanismem pro analýzu příchozího paketu a jeho modifikaci danou příslušnými protokoly.

Mezi jednotlivými uzly mohou být přenášená data interpretována různými kódy. V současné době je u některých aplikací vyžadován zabezpečený přenos, a proto musí být příslušné uzly (např. na vstupu a výstupu ze zabezpečené zóny) schopny příchozí data šifrovat a dešifrovat. V případě malé propustnosti linky mohou být odchozí data komprimována a tím sníženy nároky na šířku pásma.

Aby nebyla narušena plynulost toku dat na síti, je nezbytné, aby analýza a editace paketů v uzlech byla prováděna v reálném čase. To lze realizovat různým metodami. Jedním ze způsobů je postup, kdy se celé zařízení implementuje softwarově s využitím univerzálního procesoru PC. Tento způsob však naráží na značná omezení způsobená limitující propustností sběrnic PC a nedostatečný výkon univerzálního procesoru. Z tohoto důvodu stále více nabývá na důležitosti snaha přesouvat implementaci na úroveň aplikačně specifických obvodů (ASIC) nebo *programovatelných hradlových polí* (FPGA), které jsou schopny řídit činnost celého zařízení s dostatečnou rychlostí a tento systém tak nebude úzkým hrdlem sítě. Tento specializovaný hardware může být realizován jako samostatné zařízení (např. velké

komerční směrovače nebo přepojovače), nebo jako akcelerátor v podobě přídavné karty do PC. V obou případech lze k implementaci výkonného firmwaru využít programovatelného hradlového pole, které oproti jiným technologiím (ASIC), umožňuje průběžně vylepšovat vlastnosti systému.

Jednotlivé pakety jsou na sobě nezávislé, a proto mohou být rozděleny do proudů a zpracovávány paralelně stejným programem, který se bude opakovat pro každý paket. Před zpracováním budou pakety uchovávány ve společném úložišti. Vzhledem k charakteru příchozích dat je pro jejich hardwarové zpracování vhodné použít proudový procesor.

Tato práce se zabývá návrhem a implementací procesní jednotky pro analýzu a editaci síťového provozu v FPGA. Je realizována jako proudový procesor vybavený volitelným počtem programovatelných 16-bitových výpočetních jader Gena [8] s architekturou RISC a řadiči jednotlivých proudů realizujících jejich řízení a komunikaci s externím úložištěm paketů. Každé výpočetní jádro může být vybaveno volitelným počtem různých koprocesorů, které mohou autonomně vykonávat některé výpočetně náročné operace. Komunikace jádra s proudovým řadičem a koprocesory je realizována prostřednictvím interní paměti.

Procesní jednotka pro analýzu a editaci síťového provozu (dále jen Procesní jednotka) je implementována v rámci projektu Liberouter [7], jehož cílem je tvorba IPv6 směrovače a je součástí výzkumného záměru CESNETu Programovatelný hardware. Celý projekt je realizován na kartách COMBO6X [21], které jsou rozšířeny o přídavné karty pro různá síťová rozhraní (RJ45 1Gb ethernet, SFP 1Gb ethernet, XFP 10Gb ethernet).

Dokument je členěn do několika částí. Za úvodem následuje pasáž, ve které je čtenář seznámen s referenčním modelem ISO/OSI, formátem nejběžnějších protokolů linkové a síťové vrstvy (Ethernet, IPv4 a IPv6) a příklady nejčastějších editací. Na teoretický úvod navazuje kapitola zabývající se rozborem současného stavu řešené problematiky. Je zde nastíněna typická architektura proudových procesorů. Největší část práce je vyhrazena popisu realizace *Procesní jednotky*. Čtenář je nejprve seznámen s cílovým hardwarem a celkovou blokovou strukturou projektu Liberouter, v jehož rámci je komponenta vyvíjena. Následuje podrobný popis celé architektury *Procesní jednotky* a všech jejich částí. V samém závěru jsou zhodnoceny dosažené cíle a vlastnosti návrhu.

Kapitola 2

Síťové modely a protokoly

2.1 Referenční model ISO/OSI

Model ISO/OSI je referenční komunikační model označený zkratkou slovního spojení „International Standards Organization / Open System Interconnection“ (Mezinárodní organizace pro normalizaci / propojení otevřených systémů) [4, 5]. Jedná se o doporučený model definovaný organizací ISO v roce 1983, který rozděluje vzájemnou komunikaci mezi počítači do sedmi souvisejících vrstev. Zmíněné vrstvy jsou též známé pod označením Sada vrstev protokolu.

Úkolem každé vrstvy je poskytovat služby následující vyšší vrstvě a nezatěžovat ji detaily o tom, jak je služba ve skutečnosti realizována. Než se data přesunou z jedné vrstvy do druhé, rozdělí se do paketů. V každé vrstvě se pak k paketu přidávají další doplňkové informace (formátování, adresa), které jsou nezbytné pro úspěšný přenos po síti. Uvedený model je zobrazen na obrázku 2.1 a obsahuje následující vrstvy:

7. **Aplikační vrstva** – Tvoří uživatelské rozhraní pro komunikaci po síti. Definuje způsob, jakým komunikují se sítí aplikace, například databázové systémy, elektronická pošta nebo programy pro emulaci terminálů. Používá služby nižších vrstev a díky tomu je izolována od problémů síťových technických prostředků.

6. **Prezentační vrstva** – Převádí data do jednotné podoby srozumitelné aplikační vrstvě a naopak. Řeší například problém kódování přenášených dat, kdy různé platformy používají různý formát pro reprezentaci čísel, znaků a řetězců. Další problematikou je například komprese a šifrování dat.

5. **Relační vrstva** – Zajišťuje relaci mezi dvěma uzly. Ustavuje, udržuje a ukončuje spojení mezi lokální a vzdálenou aplikací. Je odpovědná za zabezpečovací, přihlašovací a správní funkce. V případě chyby obstará zotavení relace na základě bodů návratu, které vytváří.

4. **Transportní vrstva** – Zajišťuje transparetní přenos dat mezi dvěma uzly a odstiňuje vyšší vrstvy od záležitostí spojených se spolehlivým přenosem dat. Transportní vrstva řídí přenos dat pomocí řízení toku (flow control), segmentace a kontroly chyb. Některé z protokolů této vrstvy jsou spojově orientované, tzn. že sledují jednotlivé datové toky a pokud se v nich vyskytne chyba, zajistí opětovné odvysílání chybného paketu. Nejznámější protokoly čtvrté vrstvy jsou *Transmission Control Protocol* (TCP) nebo *User Datagram Protocol* (UDP).

3. **Síťová vrstva** – Rozděluje libovolně dlouhé datové sekvence do paketů a přenáší je ze zdroje do cíle přes jednu nebo více sítí. Hlavním úkolem síťové vrstvy je směrování paketů od zdroje k cíli po mezilehlých uzlech, dále zajišťuje kvalitu služeb (QoS). Nejznámějším protokolem třetí vrstvy je *Internet Protocol* (IP). Na této vrstvě pracují směrovače.

2. **Linková vrstva** – Zajišťuje přenos dat mezi dvěma sousedními uzly sítě a odhaluje (příp. opravuje) chyby, které vznikly na fyzické vrstvě. Řeší přístup ke sdílenému médiu a provádí serializaci a deserializaci dat fyzické vrstvy. Nejznámějším protokolem druhé vrstvy je *Ethernet*. Na této vrstvě pracují přepojovače, mosty a síťové karty.

1. **Fyzická vrstva** – Definuje elektrické a fyzické vlastnosti uzlu. Udává rozložení nožiček, napěťové úrovně a druh kabeláže. Převádí data z digitální podoby na odpovídající signál přenášený médiem a zajišťuje jejich přenos po fyzickém médiu. Na této vrstvě pracují např. opakovače a rozbočovače.

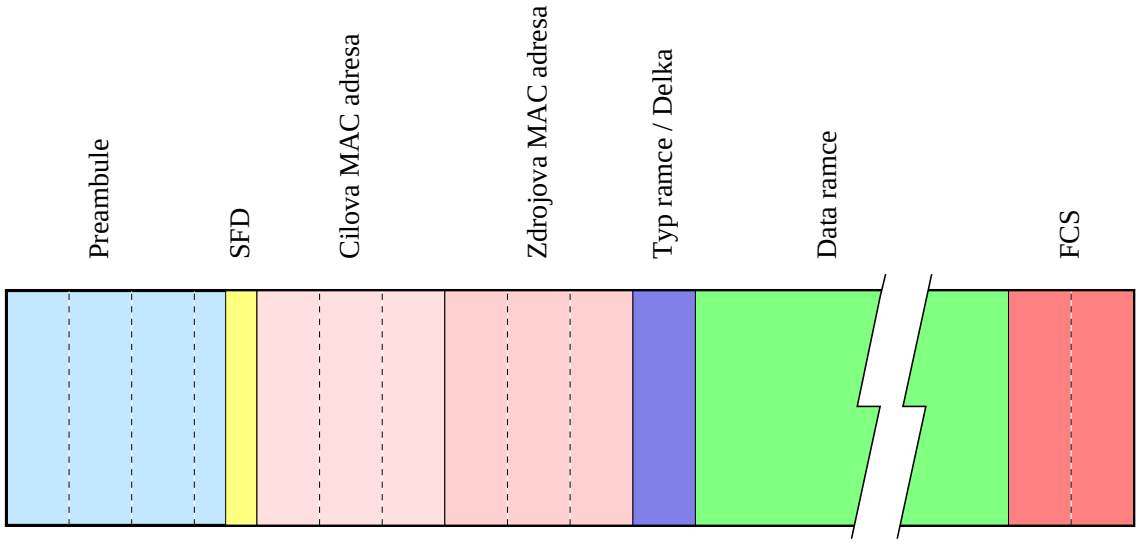
OSI Model			
	Datove jednotky	Vrstva	Funkce
Vrstvy orientovane na podporu aplikaci	Data	**Aplikacni**	Zpristupnuje data uzivateli
		Prezentacni	Reprezentace dat a sifrovani
		Relacni	Meziuzlova komunikace
	Segmenty	**Transportni**	Dvoubodove spojeni a spolehlivy prenos
Vrstvy media	Pakety	**Sitova**	Smerovani a logicke adresovani (IP)
	Ramce	**Linkova**	Fyzicke adresovani (MAC & LLC)
	Bity	**Fyzicka**	Prenosove medium, signal a prenos bitu

Obrázek 2.1: Referenční model ISO/OSI

2.2 Ethernetový rámec a jeho editace

Dominantním protokolem na linkové vrstvě je Ethernet (až 80% všech instalací v lokálních sítích). Je specifikován normou IEEE 802.3 [1]. Zajišťuje komunikaci mezi dvěma sousedními uzly na sdíleném médiu, definuje adresování v rámci tohoto média a řešení kolizí při společném přístupu. Norma 802.3 specifikuje také kontrolu přijatých dat na základě kontrolního součtu (Frame Check Sequence - FCS). Dále definuje způsob, jakým zajistit znovuzaslání chybných rámců; to se však v praxi příliš nerozšířilo a tato starost je ponechána Transportní vrstvě.

Norma definuje několik formátů Ethernetového rámce. Vedle standartního formátu se používá ještě formát definovaný normou IEEE 802.1Q obsahující VLAN tag. Tento typ rámců vznikl s vytvořením konceptu tzv. virtuálních lokálních sítí, který byl později standardizován. Konstrukce virtuálních sítí umožnuje vytvoření několika logických lokálních sítí v rámci jedné fyzické. Hlavní výhodou je pak snadná rekonfigurace a údržba. Další modifikací je formát definovaný normou 802.1P, který byl vytvořen pro zajištění kvality služeb (Quality of Service - QoS). Rozděluje rámce do prioritních tříd, přičemž rámce s vyšší prioritou jsou odesílány přednostně. Formát standartního ethernetového rámce je na obrázku 2.2.



Obrázek 2.2: Stadnartní ethernetový rámec

Preambule (až 7 bajtů) – Slouží k synchronizaci příjímače a je tvořena alternující posloupností nul a jedniček začínající jedničkou. Bajty preambule mají tedy hodnotu 0x55. Obvykle mívá délku sedmi bajtů, ale její velikost může být menší až nulová.

SFD (1 bajt) – Starting Frame Delimiter označuje začátek dat. Je tvořen alternující posloupností nul a jedniček stejně jako preambule, ale poslední nula je nahrazena jedničkou. Hodnota pole SFD je tedy hodnota 0xD5.

Cílová MAC adresa (6 bajtů) – Udává fyzickou (MAC) adresu cílového uzlu. Vzhledem k tomu, že je Ethernetová linka sdílená, obdrží tento rámec všechny uzly sdílející linku s cílovým uzlem. Každý uzel proto zkontroluje hodnotu tohoto pole a pokud se shoduje s jeho MAC adresou, pošle rámec vyšší vrstvě, jinak jej zahodí. Výjimku tvoří broadcastové a multicastové rámce. Broadcastový je přijat každým uzlem, který ho obdrží a multicastový pouze těmi uzly, které jsou příhlášené do odpovídající multicastové skupiny.

Zdrojová MAC adresa (6 bajtů) – Udává fyzickou (MAC) adresu odesílajícího uzlu.

Typ rámce / Délka (2 bajty) – Tato položka udává buď délku rámce, nebo typ dat vyšší vrsvy (u protokolu Ethernet II). Význam pole je dán jeho hodnotou. Je-li větší než 0x600, pak se jedná o rámec typu Ethernet II. V opačném případě udává délku rámce.

Data rámce (46-1500 bajtů) – V tomto poli jsou přenášena samotná data rámce rozšířená o hlavičky vyšších vrstev. Pokud je velikost dat rámce menší než 46 bajtů, jsou doplněna libovolnou posloupností (tzv. Pad) na minimální požadovanou velikost. Minimální požadovaná velikost rámce hraje důležitou při odhalování kolizí na sdíleném médiu.

FCS (4 bajty) – Pole Frame Check Sequence nese kontrolní součet počítaný z cílové a zdrojové MAC adresy, typu rámce a dat rámce. Slouží pro detekci chyb při přenosu po fyzickém médiu. Odesílající uzel vypočítá hodnotu FCS a připojí ji na konec rámce. Cílový uzel vypočítá nad rámcem hodnotu FCS podle stejného algoritmu a porovná ji s hodnotou FCS na konci rámce. Pokud se neshoduje, rámec zahodí a může požádat o jeho znovuzaslání.

Z pohledu linkové vrstvy musí uzel při odesílání/přeposílání rámce provést následující editace:

1. Správně vyplnit cílovou a zdrojovou MAC adresu.
2. Vyplnit délku rámce, příp. jeho typ.
3. Vložit do rámce data vyšších vrstev. Pokud jsou příliš krátká, doplnit je Padem.
4. Spočítat a vyplnit kontrolní součet FCS.

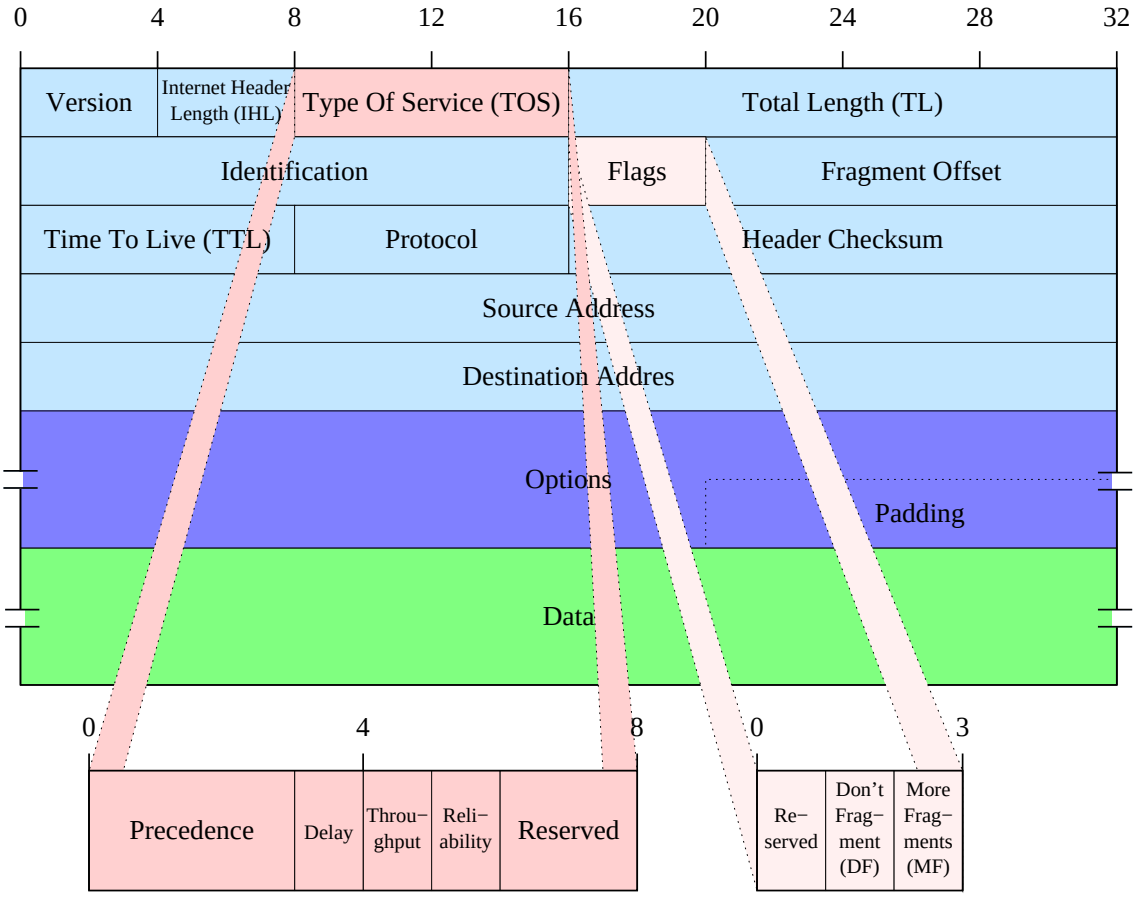
2.3 IP paket a jeho editace

Dominantním protokolem síťové vrstvy je Internet Protocol a tvoří základ Internetu. Je specifikován normou RFC 791 [6]. Zabezpečuje doručování paketů ze zdrojového uzlu k cílovému, přičemž se oba mohou nacházet v jiných sítích. Zajišťuje tedy *směrování* paketu po mezilehlých uzlech. IP protokol v doručování datagramů poskytuje nespolehlivou službu, označuje se také jako best effort - „nejlepší úsilí“; tj. všechny stroje na trase se datagram snaží podle svých možností poslat blíže k cíli, ale nezaručují prakticky nic. Datagram vůbec nemusí dorazit, může být naopak doručen několikrát a neručí se ani za pořadí doručených paketů. Pokud aplikace potřebuje splehlivost, je potřeba ji implementovat v jiné vrstvě síťové architektury, typicky protokoly bezprostředně nad IP (obvykle TCP).

Každé síťové rozhraní komunikující prostřednictvím protokolu IP má přiřazeno jednoznačný identifikátor, tzv. *IP adresu*. V každém paketu je pak uvedena IP adresa odesílatele i příjemce. Na základě IP adresy příjemce pak každý uzel na trase provádí rozhodnutí, jakým směrem paket odeslat. Tato činnost se nazývá *směrování* (routing).

2.3.1 IP verze 4

Starší verzí IP protokolu je IP verze 4. Tento protokol tvoří základní kámen Iternetu a navzdory svým nevýhodám je stále domintatní. Na obrázku 2.3 je jeho struktura.



Obrázek 2.3: IP paket verze 4

Version (4 bity) – Určuje verzi IP paketu. U verze IPv4 obsahuje číslo 4. Smyslem tohoto pole je zajistit kompatibilitu mezi systémy pracujícími nad různými verzemi IP protokolu. Obecně zařízení pracující nad starší verzí bude odmítat novější verze IP paketů.

IHL (4 bity) – Udává délku hlavičky IP paketu v 32-bitových slovech. Zahrnuje délku všech volitelných polí (Options) a zarovnání (Pad). Obvyklá hodnota tohoto pole je 5, pokud nejsou použita žádná volitelná pole.

TOS (1 bajt) – Pole určené především pro zajištění kvality služeb, zejména prioritní doručování. Ke svému původnímu účelu nebylo nidky příliš používáno a jeho význam se postupně změnil pro použití s technikou *Differentiated Services (DS)*. Viz níže.

TL (2 bajty) – Udává celkovou délku IP paketu v bajtech. Maximální délka IP paketu je tedy 65535 bajtů, obvykle však bývají mnohem menší.

Identification (2 bajty) – Toto pole nese hodnotu vlastní každému fragmentu konkrétní zprávy. Je vyplněno i u nefragmentovaných paketů, takže může být použitu, pokud musí směrovač paket během přenosu fragmentovat. Přijímající strana hodnotu v tomto poli použije pro sestavení paketu z fragmentů, protože ty mohou dorazit mimo pořadí.

Fragment Offset (13 bitů) – Udává pozici fragmentu ve zprávě v násobku 8 bajtů (64 bitů). První fragment má offset 0.

Flags (3 bity) – Pole tří příznaků. Dva z nich jsou použity pro řízení fragmentace, jeden je rezervován.

Reserved – Rezervováno.

DF – Don't Fragment, nefragmentovat. Je-li nastaven na 1, paket by neměl být fragmentován. Vzhledem k tomu, že proces fragmentace není pro vyšší vrstvy viditelný, používá se obvykle pouze ke zjištění hodnoty MTU linky.

MF – More Fragments, další fragmenty. Je-li nastaven na 0, pak jedná o poslední fragment zprávy. V opačném případě budou následovat ještě další fragmenty. Pokud není paket fragmentován, je tento příznak nastaven na 0.

Time To Live (1 bajt) – Udává počet směrovačů, kterými paket projde, než bude zahozen. Každý směrovač při průchodu paketu sníží hodnotu pole o jedna. Tento mechanismus má význam pro odstraňování zbloudilých paketů ze sítě.

Protocol (1 bajt) – Určuje protokol vyšší vrstvy - obvykle protokol transportní vrstvy nebo zapouzdřený síťový protokol - přenášený v paketu. Hodnoty pole jsou původně definovány standardem „Assigned Numbers" organizace IETF, dokumentem RFC1700 a v současnosti jsou spravovány organizací IANA.

Header Checksum (2 bajty) – Kontrolní součet vypočítaný z polí hlavičky. Slouží pro detekci chyby při přenosu. Jedná se o jednoduchou verzi kontrolního součtu, která vzniká pouhým sčítáním hodnot polí hlavičky.

Source Address (4 bajty) – 32-bitová adresa uzlu, který data paketu vytvořil. Během cesty paketu se nemění.

Destination Address (4 bajty) – 32-bitová adresa uzlu, pro který jsou data paketu určena. Během cesty se obvykle nemění.

Options (proměnná délka) – Vyskytuje se u některých typů IP hlaviček.

Padding (proměnná délka) – Vyskytuje-li se v hlavičce pole Options, pak je zarovnáno Padem na délku násobku 32 bitů (4 bajtů).

Data (proměnná délka) – Obsahuje data paketu společně s hlavičkami vyšších vrstev.

2.3.2 IP verze 6

Novou verzí IP protokolu je IP verze 6. Byla vytvořena za účelem náhrady IP verze 4, která již svým adresovým prostorem nedostačuje a nesplňuje a nevyhovuje již požadavkům současného Internetu. Hlavní rozdíly oproti verzi 4 jsou:

Rozšiřující hlavičky – Některá méně často používaná pole jsou přesunuta z hlavní hlavičky do volitelných rozšiřujících hlaviček a nepoužívaná pole byla odstraněna. Tento model poskytuje lepší flexibilitu, protože umožňuje v případě potřeby přidat další informace v podobě rozšiřující hlavičky.

Přejmenovaná pole – Názvy některých polí byly změněny tak, aby lépe reflektovaly jejich využití v současných systémech.

Eliminace výpočtu CRC – Pro IPv6 hlavičku se nepočítá kontrolní součet. To zjednodušuje zpracování paketu a ušetří místo v hlavičce.

Lepší podpora pro zajištění kvality služeb – V IPv6 hlavičce je definováno nové pole *Flow Label* pro lepší podporu kvality služeb.

Základní koncept IPv6 definuje obecnou strukturu začínající povinnou hlavní hlavičkou délky 40 bajtů, která může být následována volitelným počtem rozšiřujících hlaviček a daty paketu různé délky. To umožňuje přidat k paketu další informace, pokud jsou potřeba. Formát hlavní hlavičky je na zobrazen na obrázku 2.4.

Version (4 bity) – Určuje verzi IP paketu. U verze IPv6 obsahuje číslo 6. Smyslem tohoto pole je zajistit kompatibilitu mezi systémy pracujícími nad různými verzemi IP protokolu. Obecně zařízení pracující nad starší verzí bude odmítat novější verze IP paketů.

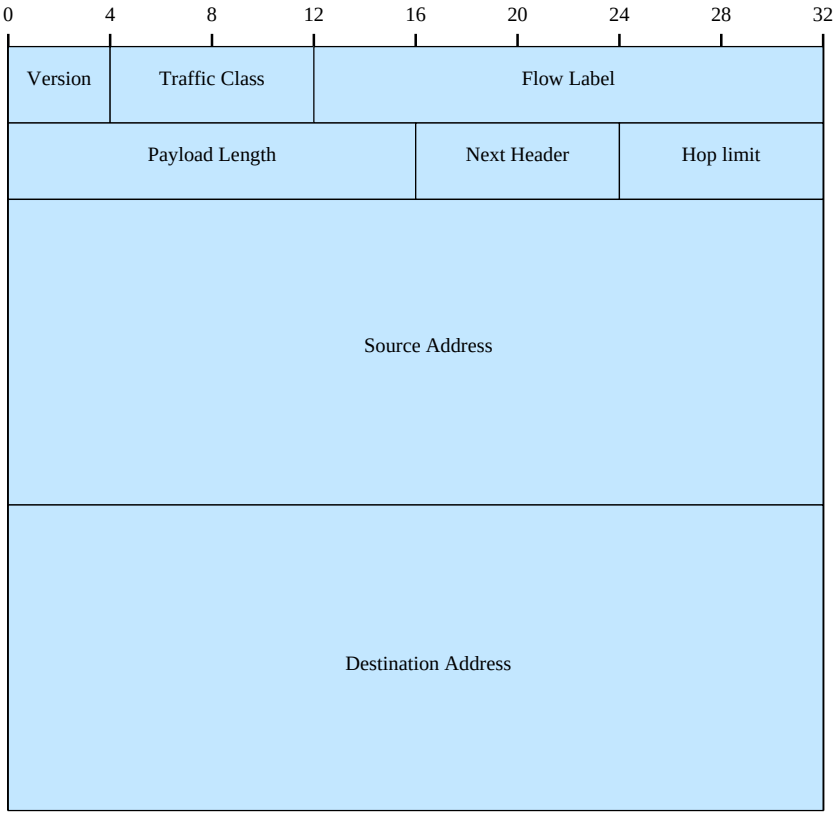
Traffic Class (1 bajt) – Tato položka nahrazuje pole *Type of Service* v IPv4 hlavičce. Je využívána službou *Differentiated Services* definovanou normou RFC 2474 pro IPv4 i IPv6.

Flow Label (2,5 bajtu) – Toto pole se používá pro podporu doručování paketů v reálném čase a požadavků pro zajištění kvality služeb (QoS). Pojem *toku (flow)* je definován normou RFC 2460 jako posloupnost paketů odeslaných z nějakého zdroje do jednoho nebo více cílových uzlů. Unikátní *jmenovka toku (Flow Label)* se používá k rozlišení konkrétních toků, takže směrovače mohou pakety patřící do stejného toku zpracovat stejným způsobem. Tím se ušetří čas potřebný k určení cesty a zkrátí se doba potřebná k doručení paketu.

Payload Length (2 bajty) – Nahrazuje pole *Total Length* v IPv4 hlavičce. Udává délku dat paketu včetně rozšiřujících hlaviček. Nezahrnuje délku hlavní hlavičky.

Next Header (1 bajt) – Nahrazuje pole *Protocol* v IPv4 hlavičce. Pokud paket obsahuje rozšiřující hlavičky, hodnota tohoto pole udává typ první z nich. Pokud paket žádně rozšiřující hlavičky neobsahuje, je význam tohoto pole shodný s polem Protocol.

Hop Limit (1 bajt) – Nahrazuje pole *TTL* v IPv4 hlavičce. Název Hop Limit lépe vystihuje způsob, jakým je TTL používáno v moderních sítích.



Obrázek 2.4: IP paket verze 6

Source Address (16 bajtů) – 128-bitová adresa uzlu, který data paketu vytvořil. Během cesty paketu se nemění.

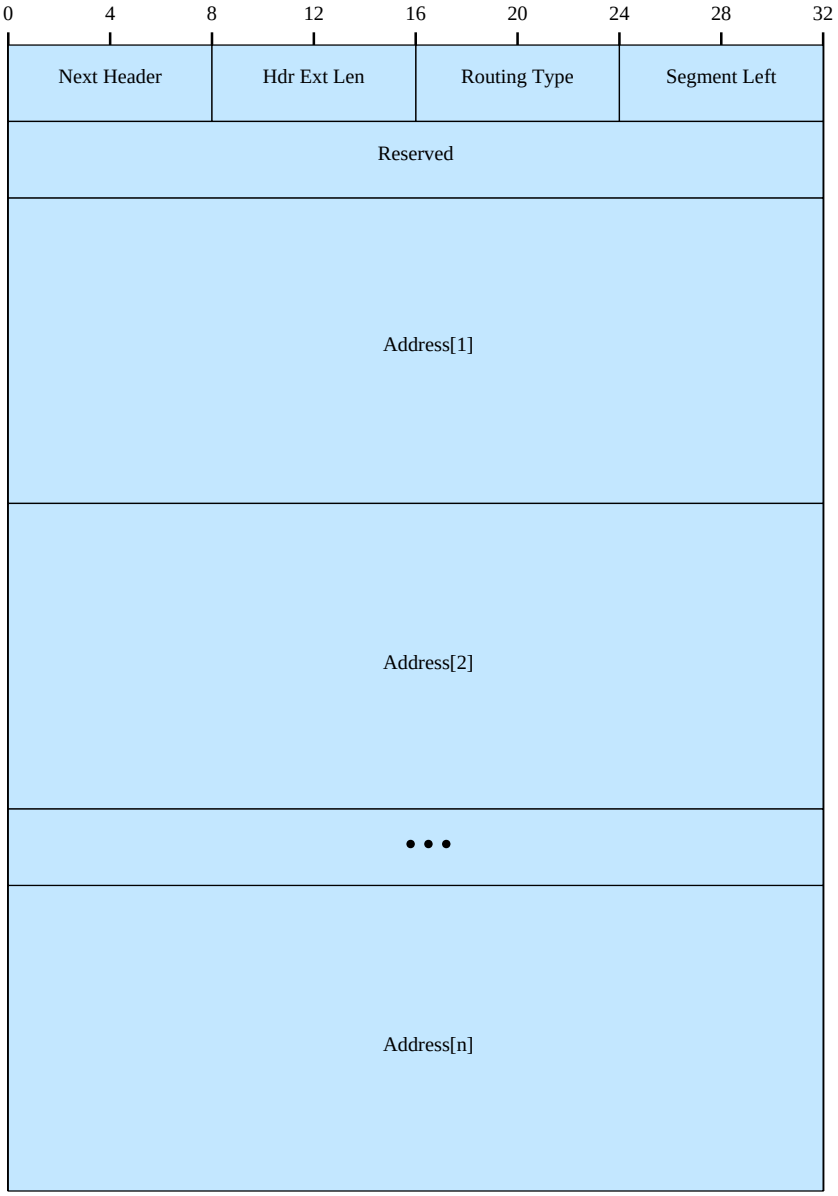
Destination Address (16 bajtů) – 128-bitová adresa uzlu, pro který jsou data paketu určena. Během cesty se obvykle nemění.

2.3.3 Snížení hodnoty Hop Limit (TTL)

Vlivem poruch na jednotlivých segmentech sítě může docházet k situacím, kdy není možné paket doručit do cílového uzlu. Například při výpadku směrovače. Snahou směrovacího mechanismu je v tomto případě nalézt cestu jinou a nefunkční bod obejít. Tato cesta však nemusí vždy existovat. Aby nedocházelo k nekonečnému zacyklení paketu v síti nebo k přenosu po dlouhých a nesprávných cestách, je použit mechanismus dekrementace položky Hop Limit uložené v hlavičce IPv6 paketu (TTL pro IPv4). Položka Hop Limit je nastavena vysílačem paketu a všechny uzly (směrovače), kterými v průběhu své cesty paket projde, jsou povinny tuto položku snížit o jedničku. Pokud položka dosáhne nulové hodnoty, je paket ze sítě odstraněn.

2.3.4 Zpracování Směrovací hlavičky

Směrovací hlavička (*Routing Header*) je volitelná rozšiřující hlavička IPv6 paketu. Každý směrovač by měl být schopen tuto hlavičku rozpoznat a zpracovat. Směrovací hlavička slouží pro tzv. source routing, kdy vysílač z nějakého důvodu potřebuje předem určit cestu paketu k cílovému uzlu. Hlavní součástí Směrovací hlavičky je seznam IPv6 adres odpovídajících uzlům na cestě, kterými paket musí postupně projít (pole Address). Posledním prvkem seznamu je adresa cílového uzlu. Formát Směrovací hlavičky je uveden na obrázku 2.5.



Obrázek 2.5: Formát Směrovací hlavičky

Next Header (1 bajt) – Udává typ následující rozšiřující hlavičky.

Hdr. Ext. Len (1 bajt) – Udává délku Směrovací hlavičky v násobcích osmi bajtů (bez prvních osmi bajtů).

Routing Type (1 bajt) – Určuje typ Směrovací hlavičky.

Segment Left (1 bajt) – Nese počet nezpracovaných položek pole Address (tedy počet uzlů uvedených v Address, kterými paket ještě neprošel).

Address (N x 16 bajtů) – Obsahuje IPv6 adresy uzlů, které má paket po řadě projít.

Zpracování Směrovací hlavičky probíhá ve třech krocích:

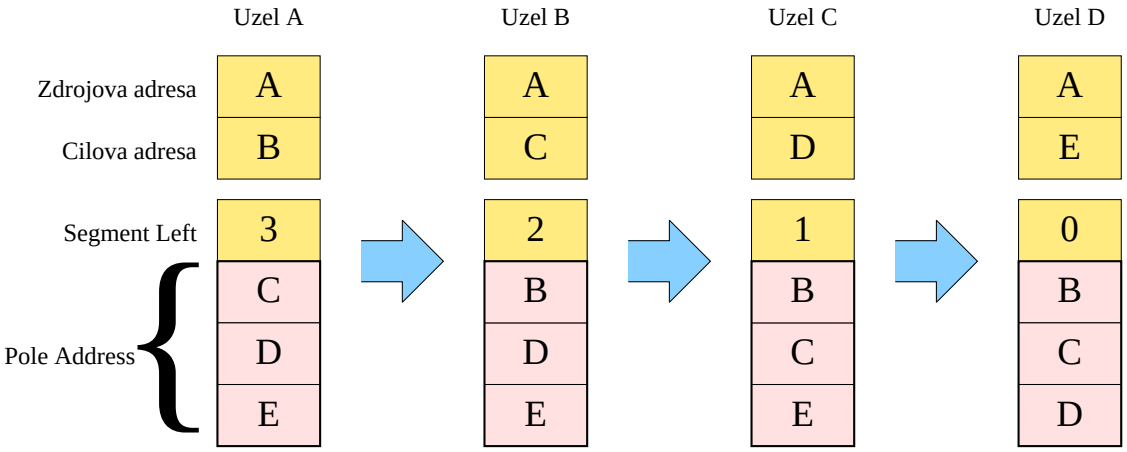
1. Pokud cílová adresa (DA) IPv6 paketu odpovídá adrese současného uzlu (směrovače), nalezne novou cílovou adresu (NewDA) v poli Address takto:

$$NewDA = Address[(HdrExtLen/2) - SegmentLeft + 1]$$

2. Provede výměnu současné DA za NewDA.

3. Sníží hodnotu pole SegmentLeft o jedničku.

Na obrázku 2.6 je uveden příklad zpracování Směrovací hlavičky. Uzel A posílá paket přes uzly B, C, D do uzlu E.



Obrázek 2.6: Příklad zpracování Směrovací hlavičky

2.3.5 Výpočet CRC hlavičky IPv4 paketu

Při směrování paketu dochází ke změnám položek v jeho hlavičce, proto je nezbytné přepočítat po úpravách její CRC. Jelikož se nachází uvnitř hlavičky a ne na jejím konci, je nutné vypočítat kontrolní součet dopředu, ještě před odesláním hlavičky. Výpočet 16-bitového kontrolního součtu je velmi zjednodušený. Data hlavičky se rozdělí na 16-bitová slova a ta se sčítají, příčemž případné přenosy se zahazují.

Kapitola 3

Současný stav řešené problematiky

V minulé kapitole byly popsány hlavičky nejrozšířenějších internetových protokolů a způsoby jejich editací. Při přenosových rychlostech dosahovaných na páteřních sítích není možné tyto úpravy provádět softwarově. Vhodným řešením je použití hardwarového zařízení, konkrétně *proudového procesoru* [2, 3].

Proudový procesor je vhodný pro aplikace, které lze hustě paralelizovat a vyžadují minimální globální komunikaci. Typicky je to zpracování médií nebo síťového toku. Základní ideou proudového zpracování je rozdělit aplikaci do *proudů (streams)* a *jader (kernels)*.

Proudy jsou datové toky různých délek skládající se z posloupnosti elementů stejného typu. Proudové elementy mohou být jednoduché, např. číslo, nebo složité, např. souřadnice trojúhelníka v 3D prostoru, nebo paket.

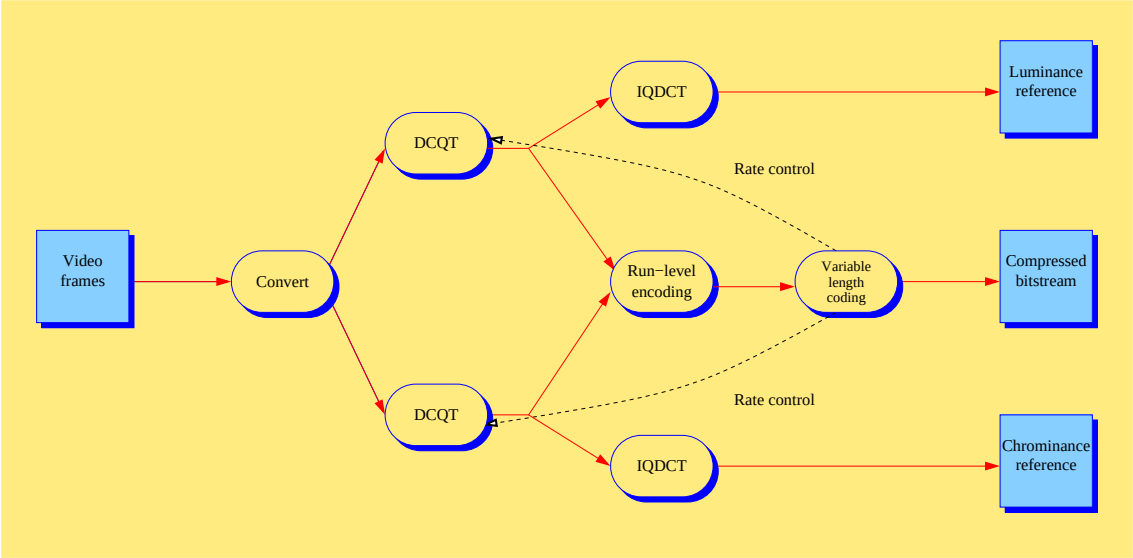
Jádra realizují výpočetní operace nad elementy a mohou mít jeden nebo více vstupních a výstupních proudů (počet vstupů a výstupů může být navzájem různý). Každý element je zpracováván stejným způsobem. Jádra mohou být realizována jako úzce specializované moduly, nebo jako programovatelné procesory. Obvykle se združují do jednotek nazývaných *Kernel Execution Unit (KEU)*.

Typický proudový procesor se skládá z několika proudů a jader, které každé řeší konkrétní podúlohu aplikace. Příkladem proudového procesoru je kodér videa formátu MPEG-2. Na obrázku 3.1 je zobrazeno schéma části MPEG-2 kodéru, které realizuje proudové zpracování I-rámců. Tato aplikace přijímá proud elementů z video vstupu, který jde do prvního jádra Convert. To ho zpracuje a rozdělí na dva proudy. Následují jádra DCTQ nad nimi provádějí diskrétní kosinovou transformaci. Odtud jednotlivé proudy pokračují dále a vstupují do dalších jader. Jakmile je zpracování dokončeno, jsou jednotlivé proudy ukládány do globálních úložišť k dalšímu zpracování (viz dále).

V předchozím příkladu jsou názorně vidět dva základní rysy proudových procesorů: mnohonásobný paralelismus a lokalita zpracovávaných dat. Tomu je uzpůsobena paměťová hierarchie proudového procesoru, která má tři úrovně:

1. **Lokální registrová pole (Local Register Files - LRFs)** – každé jádro má své LRF, které slouží k uchování mezivýsledků nebo operandů aritemtických operací.

2: **Proudové registrové pole (Stream Register File - SRF)** – každý proud má své SRF, které slouží pro přenos výsledků z jednotlivých jader. Umožňuje efektivní přenos dat mezi jednotlivými LRF.



Obrázek 3.1: MPEG-2 I-frame kodér

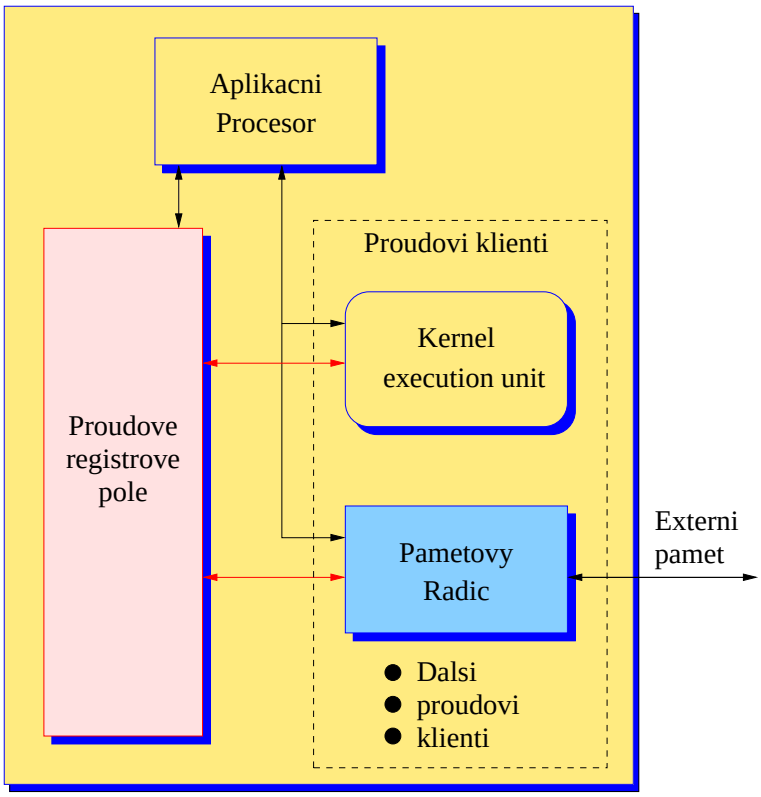
3. Globální úložiště – je společné pro všechny proudy a slouží k uložení globálních dat, především nezpracovaných elementů, které jsou odtud přidělovány do proudů a zpracovaných elementů, které čekají na vyčtení/odeslání. Obvykle se k němu přistupuje přes specializované I/O rozhraní s velkou latencí.

Proudové procesory se dělí na *úzce specializované* a *programovatelné*. Úzce specializované proudové procesory mají pevně danou funkci, kterou nelze změnit. Jejich jádra jsou realizována jako logika napevno zadrátovaná v čipu. Příkladem jsou staré grafické karty. Naproti tomu programovatelné proudové procesory mají některá jádra realizována jako jednoduché procesory s vlastní instrukční sadou a pamětí. Mezi ně patří většina moderních grafických karet vybavených programovatelnými pixel shadery a vertex shadery.

Na obrázku 3.2 je základní schéma programovatelného proudového procesoru, skládající se z *aplikačního procesoru*, SRF a *proudových klientů*.

Aplikační procesor zpracovává kód aplikace. Pracuje nad instrukční sadou typu RISC rozšířenou o *proudové instrukce* pro kontrolu datových proudů a tyto instrukce rozesílá příslušným proudovým klientům.

Mezi proudové klienty patří KEU (zde programovatelná) a řadič externí paměti. K vzájemné komunikaci a předávání dat využívají SRF. KEU spouští jednotlivá jádra a zprostředkovává lokální komunikaci pro operace v rámci jader, zatímco paměťový řadič poskytuje přístup do globálního úložiště. Zmíněná architektura může obsahovat i další proudové klienty včetně I/O rozhraní, rozhraní k externí propojovací síti nebo specializovaných KEU, např. kodér Huffmanova kódu. Jednotliví klienti se mohou v systému vyskytovat ve více instancích.



Obrázek 3.2: Základní architektura proudového procesoru

Každý programovatelný proudový klient má svou instrukční sadu vybavenou běžnými instrukcemi, v některých případech rozšířenou o proudové instrukce. Řadič paměti podporuje dvě proudové instrukce - *load_stream* a *store_stream*, které zajistí přenos celých proudů mezi externí pamětí a SRF. KEU podporuje instrukci *load_kernel*, která nahraje zkompilovaný kód pro jádro do instrukční paměti uvnitř KEU. To se obvykle děje pouze jednou při prvním spuštění jádra. Při dalším spuštění se provádí tentýž kód, který je již v instrukční paměti přítomen. Další obvyklá proudová instrukce KEU je *run_kernel*, která spustí vykonávání programu jádra.

Kapitola 4

Návrh a implementace Procesní jednotky

4.1 Technické prostředky

Procesní jednotka pro analýzu a editaci síťového provozu (dále jen Procesní jednotka) je vyvíjena v rámci projektu Liberouter, jehož cílem je tvorba IPv4 a IPv6 směrovacího akcelerátoru. Tento akcelerátor je schopen autonomně provádět směrování a filtraci multigigabitového provozu podle pravidel zadaných uživatelem. Samotné PC pak řeší pouze konfiguraci směrovače, případně zpracování paketu, jehož obsluha by v hardwaru byla velmi složitá.

V rámci projektu Liberouter byl vyvinut speciální hardware poskytující potřebné technické prostředky. Základním stavebním prvkem je karta COMBO6, která v současnosti existuje v několika provedeních, určených pro připojení k různým systémovým sběrnicím PC. Jsou to sběrnice PCI, PCI-X a PCI Express:

1. Klasická karta COMBO6 je osazena technologií FPGA Xilinx VIRTEX II XC2V3000 [16], ve kterém je nahrán firmware akcelerátoru. Ten ke své činnosti může využít třech statických pamětí SSRAM [19] a asociativní paměti CAM. Navíc je karta vybavena jedním konektorem pro dynamickou paměť DDR o velikosti až 2GB. Tato karta využívá ke komunikaci s PC sběrnice PCI.

2. Novějšími typy jsou karty COMBO6X resp. COMBO6E, které jsou určeny pro sběrnice PCI-X resp. PCI Express. Jsou osazeny čipem Xilinx VIRTEX II Pro XC2VP50 [17], ve kterém je nahrán firmware směrovače. Ke komunikaci se systémovou sběrnicí slouží druhý čip FPGA Xilinx VIRTEX II Pro XC2VP4 s jádrem pro obsluhu PCI-X resp. PCI Express. Další vybavení je obdobné jako u karty COMBO6.

Ke kartě COMBO6 lze připojit přídavnou kartu, kterých bylo vyvinuto více typů pro různá fyzická síťová rozhraní a pro různé účely. V současnosti existují tyto typy:

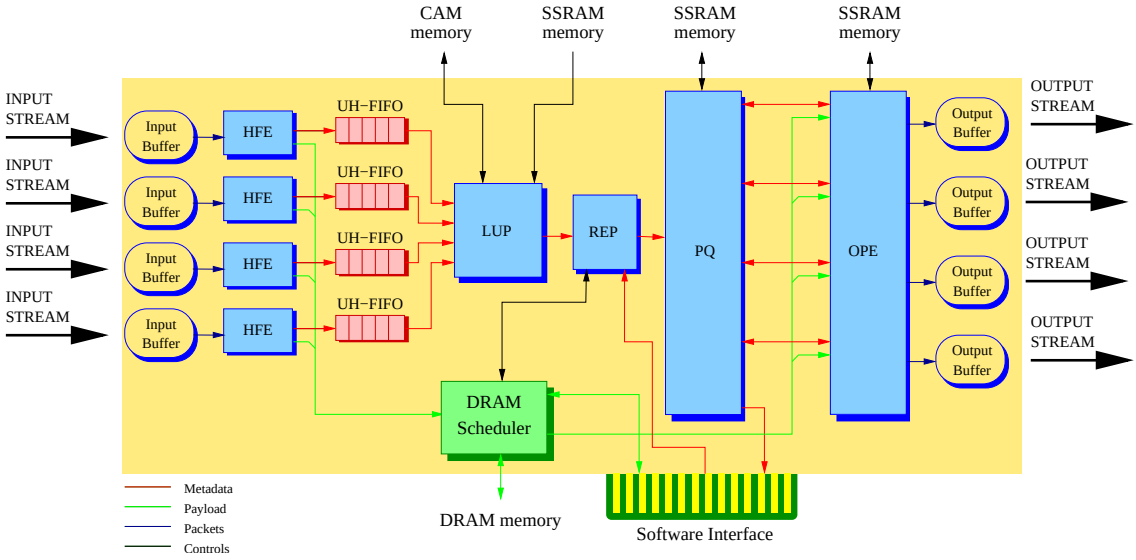
1. COMBO-4MTX je nejstarším typem a je osazena čtyřmi měděnými porty RJ45 gigabitového ethernetu (GE). Jádro karty tvoří FPGA Xilinx VIRTEX II. Firmware, který je v něm nahrán, může ke své činnosti využít dvou statických pamětí o kapacitách 2MB.

2. COMBO-4SFP je osazena čtyřmi porty SFP optického rozhraní pro GE. Jádro karty tvoří opět FPGA Xilinx VIRTEX II. Jako paměťová média slouží dvě statické paměti o kapacitách 2MB.

3. COMBO-2XFP je osazena dvěma porty XFP optického rozhraní pro desetigigabitový ethernet (10GE). Jádro této karty tvoří Xilinx VIRTEX II Pro. Další výbava je obdobná kartě SFP. Oproti ní však navíc disponuje asociativní pamětí CAM.

4.2 Schéma návrhu směrovače

K pochopení úlohy procesní jednotky je třeba osvětlit celkovou strukturu a funkci směrovače. Jeho základní schéma je na obrázku 4.1.



Obrázek 4.1: Základní schéma projektu Liberouter

Do systému vstupují pakety ze čtyř GMII rozhraní, tvoří tedy čtyři nezávislé proudy. Každý z nich je přiveden do **vstupního bufferu** (IBUF - Input Buffer) [12], odkud si je odebírá **Analyzátor paketů** (HFE - Header Field Extractor) [9]. Jedná se o jednoduchý procesor typu RISC s vlastní instrukční sadou. HFE má za úkol analyzovat hlavičku paketu a zkontrolovat, zda se v ní nevyskytují chyby. Po analýze rozdělí vstupní tok na dva - tok dat paketu a tok metadat. Metadata jsou uložena do speciální fronty: UH FIFA, odkud si je odebírá *Vyhledávací procesor* (LUP). Data paketu jsou uložena do dynamické paměti, ve které setrvají po celou dobu zpracování. Společně s nimi jsou uloženy parametry paketu potřebné pro Procesní jednotku.

Řadič dynamické paměti (DRAM scheduler) má na starosti koordinaci komunikace dynamické paměti se čtyřmi HFE, Procesní jednotkou a softwarovým rozhraním. Jeho důležitou činností je udržovat jednotlivé datové bloky v DRAM a počet odkazů na ně. Pokud na dílčí blok už neexistuje žádný odkaz, Řadič DRAM ho odstraní.

Vyhledávací procesor (LUP - LookUp Processor) [10] tvoří jádro celého směrovače. Zajišťuje samotné směrování a filtrování a to na základě zadaných pravidel. Tato pravidla jsou uspořádána do vhodné stromové struktury, uložené v asociativní paměti CAM, ve které se podle údajů z UH provede dohledání. Jeho výsledkem je adresa do statické paměti, kde je uložen mikrokód. Jeho provedením vznikne Replikační záznam (RR - Replicator Record),

jenž je předán Replikátoru. Jeho součástí je identifikace paketu a seznam výstupních rozhraní, na která se má paket odeslat. Dále obsahuje údaje potřebné pro zpracování paketu ve Výstupním paketovém editoru.

Replikátor (REP - Replicator) má za úkol zajistit zreplikování paketu pro potřeby multicastu. Zajistí tedy replikaci RR do příslušných prioritních front a současně sdělí Řadiči DRAM, aby odpovídajícím způsobem upravil počet odkazů na jednotlivé datové bloky paketu.

Prioritní fronty (PQ - Priority Queues) [14] zajišťují odesílání paketů v pořadí podle jejich priorit. Jsou složeny ze 16 front typu FIFO rozdělených do pěti skupin, z nichž každá náleží jednomu výstupnímu proudu. V rámci těchto skupin mají fronty přiděleny své jednoznačné priority. Z té, která má nejvyšší prioritu, se odbere RR a předá se příslušnému OPE ke zpracování. V případě, že se jedná o RR, který byl zreplikován Replikátorem, provede OPE pro každou zreplikovanou kopii *zpětný zápis* (writeback - WB) do fronty, ze které jej přetím vyčetl. Veškerý obsah prioritních front je fyzicky uložen v externí statické paměti.

Procesní jednotka pro analýzu a editaci paketu (OPE - Output Packet Editor) je předmětem této práce a má na starosti analýzu a editaci hlavičky paketu, jeho celkové sestavení a odvysílání. Na vstup má přivedeny čtyři proudy metadat a čtyři proudy dat paketů, z nichž podle svého programu vygeneruje čtyři proudy paketů, jeden pro každé výstupní rozhraní. Zpracování paketu začíná obdržením metadat - replikačního záznamu - z PQ, který obsahuje odkaz na blok dynamické paměti (ve kterém jsou uložena data paketu a jeho parametry) a ukazatel do paměti Procesní jednotky, ve které se nachází blok editačních parametrů, tvořený mikrokódem. Tímto mikrokódem se OPE řídí při editaci paketu. V případě, že byl tento paket zreplikován Replikátorem, upraví komponenta OPE jeho replikační záznam a uloží ho zpět na *začátek* fronty, ze které ho předtím obdržel (tzv. writeback). Tím se zajistí opakované odvysílání tohoto paketu s různými hlavičkami. Po zpracování vysílá OPE data čtyřmi výstupními proudy do Výstupních bufferů.

Výstupní buffer (OBUF - Output Buffer) [13] zajistí odvysílání dat z výstupního proudu na čtyři GMII rozhraní.

4.3 Zdroje vstupních dat

Základním úkolem Procesní jednotky je vytvořit hlavičku nového paketu, k ní připojit odpovídající data a takto vytvořený paket odeslat do výstupního proudu. Celý proces začíná zpracováním elementu ze vstupního proudu metadat, dále jen metadata. Tento proud je generován v prioritních frontách a obsahuje dvě důležité informace [11].

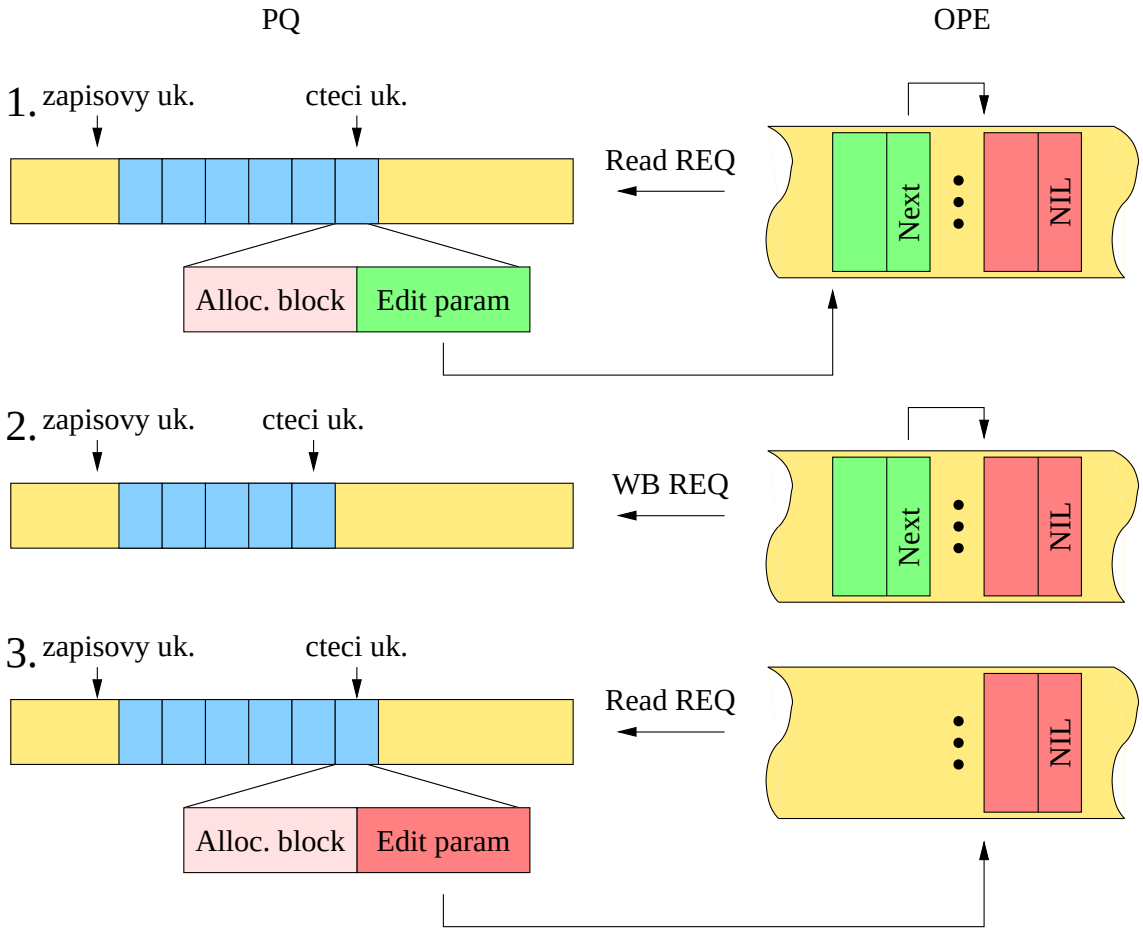
První z nich je adresa alokačního bloku dynamické paměti, kde jsou uložena data paketu společně s parametry získanými analyzátorem paketu (HFE). *Parametry paketu (Packet parameters - PP)* jsou vždy umístěny na začátku alokačního bloku a obsahují důležité informace pro zpracování. Například jeho délku nebo seznam ukazatelů na položky hlavičky, které mohou být v závislosti na typu paketu Procesní jednotkou změněny.

Druhou důležitou informací je adresa do paměti Procesní jednotky, realizované externí statickou pamětí. Tato paměť obsahuje bloky *Editačních parametrů (Edit Parameters - EP)*. Tyto bloky obsahují mikrokód, který definuje způsob, jakým bude paket zpracován. Např. zda se provede standardní zpracování hlavičky, tunelování paketu, zda se zpracuje Routing Header atd.

V praxi může nastat případ, kdy je potřeba daný paket odeslat konkrétním výstupním rozhraním ve více kopiích a každou z těchto kopií zpracovat obecně různým způsobem.

V tomto případě jsou Editační parametry zřetězeny v podobě lineárního seznamu. Procesní jednotka při zpracování prochází tímto seznamem a zpracovává vstupní paket postupně podle všech Editačních parametrů, dokud nenarazí na koncovou značku.

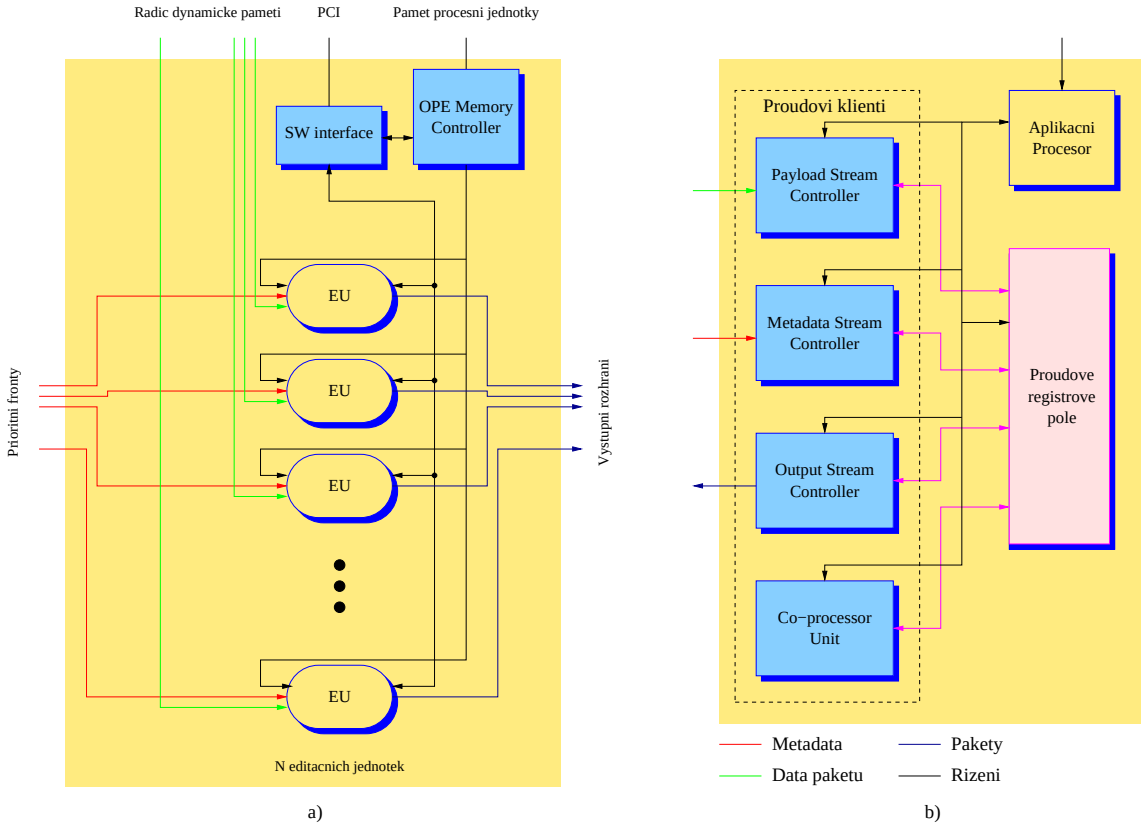
Při tomto způsobu zpracování je nutné brát ohled na příchozí pakety s vyšší prioritou. Může nastat situace, kdy OPE začne zpracovávat paket s relativně nízkou prioritou a s řetězenými editačními parametry a v průběhu zpracování přijde paket, který je potřeba odeslat na stejné výstupní rozhraní, ovšem přednostně. V této situaci není možné, aby prioritní paket čekal, než se zpracuje celý seznam Editačních parametrů paketu s nižší prioritou. Proto se vždy zpracuje pouze jeden blok seznamu a po té se provede tzv. *zpětný zápis - writeback*. Při zpětném zápisu OPE aktualizuje metadata tak, že místo původní adresy na Editační parametry vloží novou adresu na Editační parametry umístěné na následující pozici v seznamu. Takto upravená metada vloží zpět na čelo prioritní fronty. Procesní jednotka pak běžným způsobem načte další metadata z prioritních front a pokud existuje paket s vyšší prioritou, zpracuje jej, jinak obdrží předchozí metadata s upravenou adresou na Editační parametry. Proces zpětného zápisu je znázorněn na obrázku 4.2.



Obrázek 4.2: Proces zpětného zápisu

4.4 Architektura

Architektura Procesní jednotky vychází z konceptu proudových procesorů. OPE zpracovává pakety z N vstupních rozhraní, které jsou uspořádány v N-krát dvou proudech - proudu metadat a proudu dat paketů. Proud metadat je generován v prioritních frontách a proud dat paketu řadičem dynamické paměti na základě adresy alokačního bloku v metadatech. OPE se skládá z N *editačních jednotek (Editation Unit - EU)*, jednoho *řadiče paměti procesní jednotky (OPE Memory Controller - OMC)* a *softwarového řadiče (SW Interface)*. Každá editační jednotka zpracovává pakety z jednoho vstupního rozhraní a je realizována jako proudový procesor. Architektura OPE a EU je znázorněna na obrázku 4.3.



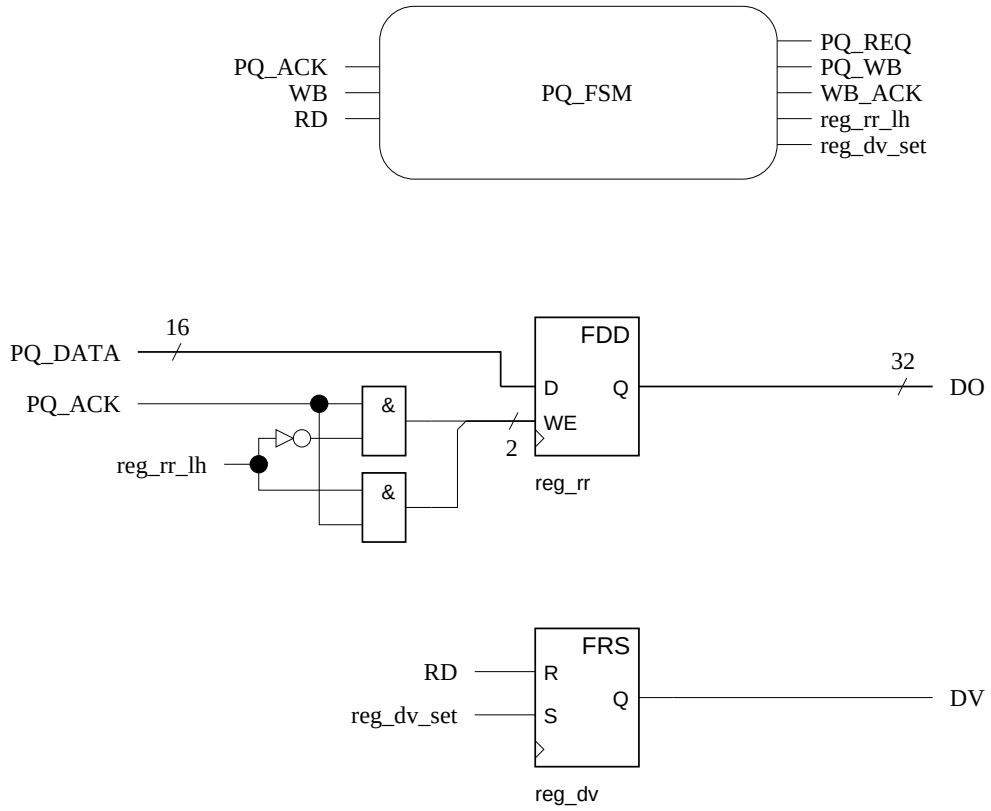
Obrázek 4.3: a) Architektura Procesní jednotky b) Architektura Editační jednotky

Procesní jednotka je navržena tak, aby načítání dalšího elementu (paketu) vstupního proudu mohlo probíhat paralelně se zpracováním elementu předchozího. Proto jsou paměti některých jednotek rozděleny dvou banků. V jednom banku jsou data zpracovávaného paketu a do druhého se načítají data paketu následujícího. Po dokončení zpracování paketu dojde k přepnutí těchto banků a zpracování nového paketu může začít bez prodlení. Následující kapitoly obsahují podrobný popis jednotlivých podkomponent.

4.4.1 Řadič proudu metadat (Metadata Stream Controller, MSC)

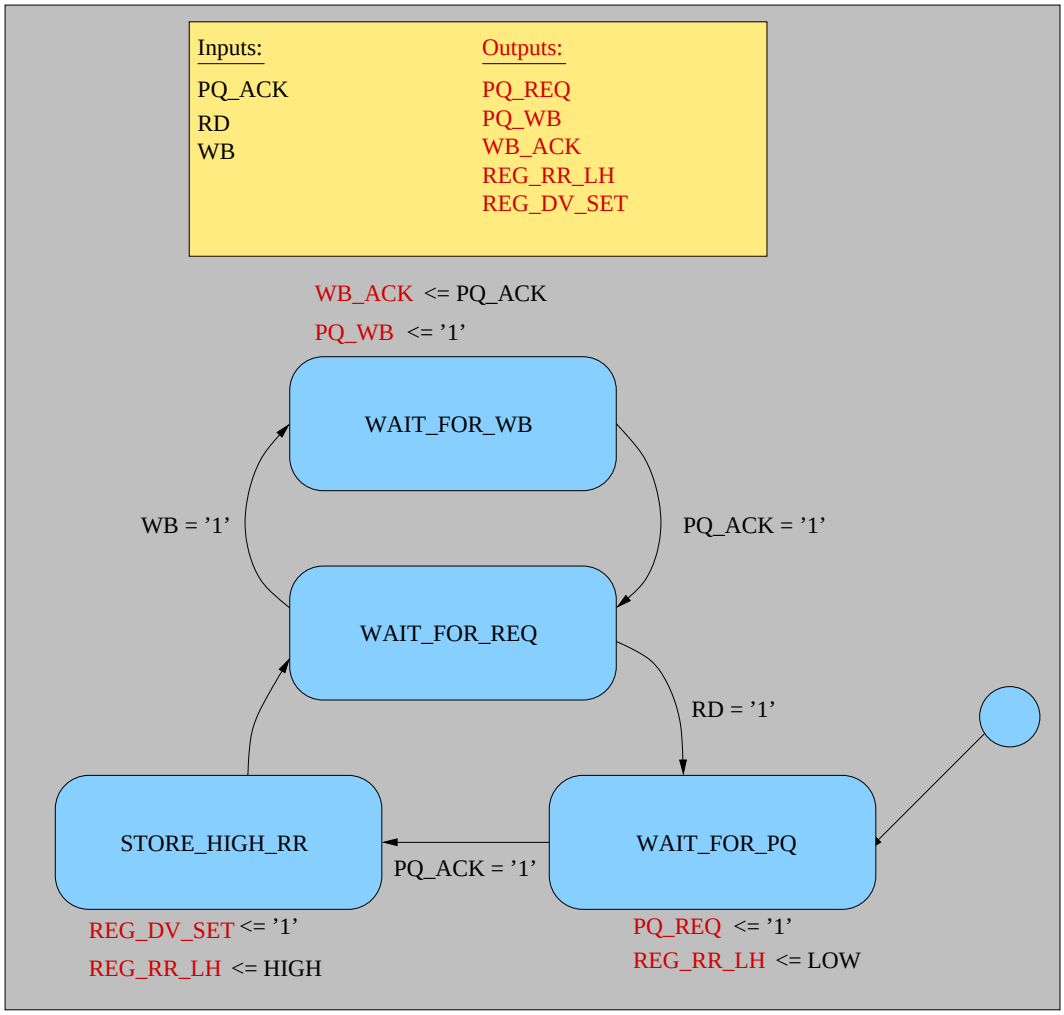
Řadič proudu metadat (MSC) má na starosti komunikaci s Prioritními frontami - načítaní elementů metadat a realizaci zpětného zápisu. Je tvořen *Řadičem priotních front (PQ Controller)*. Tento řadič si na pokyn Hlavního řadiče vyžádá záznam z Prioritních front tvořený metadaty. Metadata jsou nutná pouze během fáze načítání paketu, takže během jeho zpracování mohou být přepsána metadaty dalšího načítaného paketu. Činnost řadiče je následující:

1. Čeká na pokyn Hlavního řadiče.
2. Nastaví žádost o metadata, nebo o provedení zpětného zápisu.
3. Čeká, až Prioritní fronty potvrdí vykonání požadované operace.
4. Žádal-li Hlavní řadič o provedení zpětného zápisu, vrátí se řadič do bodu 1. Žádal-li o metadata, přejde do následujícího bodu.
5. Řadič vyčte první slovo metadat.
6. Řadič vyčte druhé slovo metadat.
7. Řadič vystaví načtená metadata a přejde do bodu 1.



Obrázek 4.4: Architektura Řadiče prioritních front

Architektura Řadiče prioritních front je znázorněna na obrázku 4.4. Je tvořena automatem PQ_FSM a dvěma registry reg_dv a reg_rr. Automat zajišťuje komunikační protokol mezi Prioritními frontami a Hlavním řadičem aplikačního procesoru (viz kapitola 4.4.4) a bude podrobně popsán níže. Registr reg_rr slouží k uložení tzv. Replikačního záznamu, který tvoří jeden element metadat a je dlouhý 32 bitů. Prioritní fronty ho posílají portem PQ_DATA ve dvou částech po 16-bitech. Platnost dat je signalizována na portu PQ_ACK a na základě signálu reg_rr_lh je uložena buď dolní, nebo horní část Replikačního záznamu. Výstup registru je přiveden na port DO. Registr reg_dv typu RS je nastavován automatem a nulován požadavkem na čtení Hlavního řadiče. Jeho výstup (port DV) potvrzuje platnost dat na portu DO.



Obrázek 4.5: Stavový diagram automatu PQ_FSM

Stavový diagram automatu PQ_FSM a seznam jeho vstupů a výstupů je znázorněn na obrázku 4.5. Po resetu se automat nachází ve výchozím stavu *WAIT_FOR_PQ*. V tomto stavu je nastaven požadavek na čtení záznamu z Prioritních front (port PQ_REQ) a je aktivní zápis do dolní části registru reg_rr (signál reg_rr_lh). Jakmile se na portu PQ_DO objeví platná data, je nastaven port PQ_ACK. Na základě něj přejde automat do stavu *STORE_HIGH_RR*.

V tomto stavu se uloží horní část Replikačního záznamu a je nastaven registr reg_dv. V dalším taktu přejde automat do stavu *WAIT_FOR_REQ*.

V tomto stavu čeká automat na požadavek od Hlavního řadiče Aplikačního procesoru. Ten může žádat buď o čtení dalšího Replikačního záznamu (nastavením portu RD) nebo o provedení zpětného zápisu (nastavením portu WB). V prvním případě přejde automat zpět do výchozího stavu *WAIT_FOR_PQ*. V druhém případě přejde do stavu *WAIT_FOR_WB*.

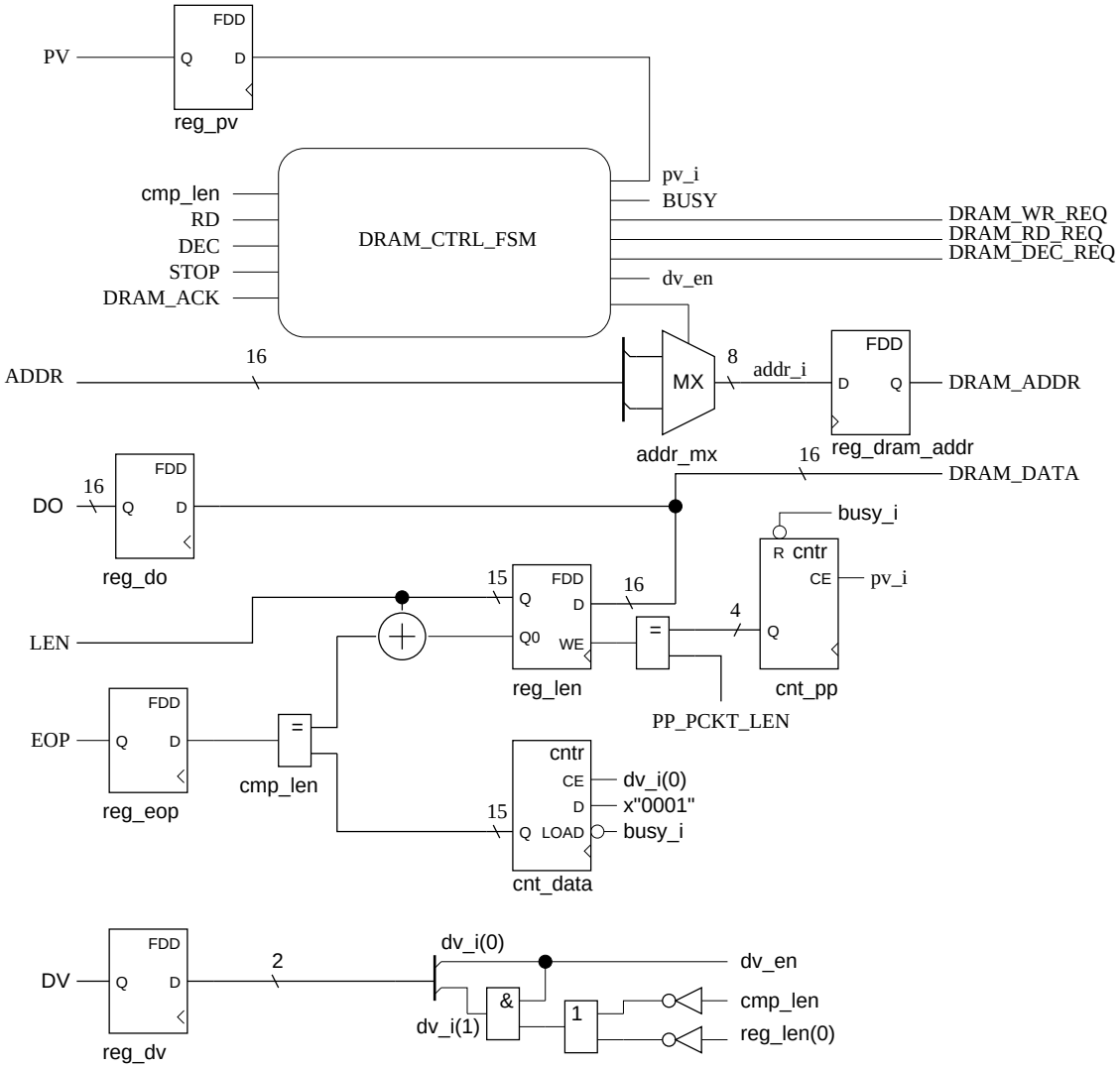
Ve stavu *WAIT_FOR_WB* nastaví automat žádost Prioritním frontám o zpětný zápis (port PQ_WB) a čeká na jeho potvrzení na portu PQ_ACK. Jakmile je jeho provedení potvrzeno, vyrozumí o tom automat Hlavní řadič nastavením portu WB_ACK a přejde zpět do stavu *WAIT_FOR_REQ*.

4.4.2 Řadič proudu dat paketů (Payload Stream Controller, PSC)

Řadič proudu dat paketů (PSC) má na starosti generování a správu proudu dat paketů. Skládá se z *Řadiče externí paměti (Memory Controller)* a *Vstupního bufferu (Input Stream Buffer - IStB)*. Řadič externí paměti zajišťuje komunikaci s externí dynamickou pamětí, v níž jsou uložena data paketu a Paketové parametry. Řadič externí paměti z nich vygeneruje proud dat paketu. Na začátku každého elementu tohoto proudu se nacházeji Paketové parametry, které se automaticky uloží do Paměti paketových parametrů a jsou z proudu odstraněny. Program k nim tedy může přistupovat pouze prostřednictvím Paměti paketových parametrů. Následuje popis činnosti Řadiče externí paměti.

1. Čeká na data z Prioritních front.
2. Uloží si adresu paketu z metadat.
3. Čeká na pokyn Hlavního řadiče.
4. Hlavní řadič žádá přečtení dat paketu.
5. Řadič zapíše adresu do řadiče dynamické paměti - nejprve dolní část, pak horní část.
6. Řadič čeká na potvrzení adresy.
7. Řadič čeká na vyčtení prvního bloku dat - paketové parametry.
8. Řadič vyčte blok paketových parametrů.
9. Řadič čeká na další blok.
10. Řadič vyčte blok dat paketu.
11. Žádá-li Hlavní řadič další data, pokračuje řadič bodem 9, jinak se vrátí do bodu 1.

Následuje popis architektury Řadiče externí paměti, zobrazené na obrázku 4.6. Řadič externí paměti je spojen se Vstupním bufferem (porty na levém okraji schématu) a s řadičem dynamické paměti (porty na pravé straně schématu s názvy DRAM_xxx). Architektura řadiče je rozdělena do tří bloků. První blok obsahuje automat DRAM_CTRL_FSM, registry reg_pv a reg_dram_addr. Úkolem automatu je řídit řadič dynamické paměti pomocí portů DRAM_RD_REQ, DRAM_WR_REQ A DRAM_DEC_REQ. Prvním portem signalizuje požadavek na čtení z adresy, kterou je nutné předtím do řadiče dynamické paměti zapsat. Zápis 16-bitové adresy se provádí po osmi bitech a je signalizován nastavením portu



Obrázek 4.6: Architektura Řadiče externí paměti

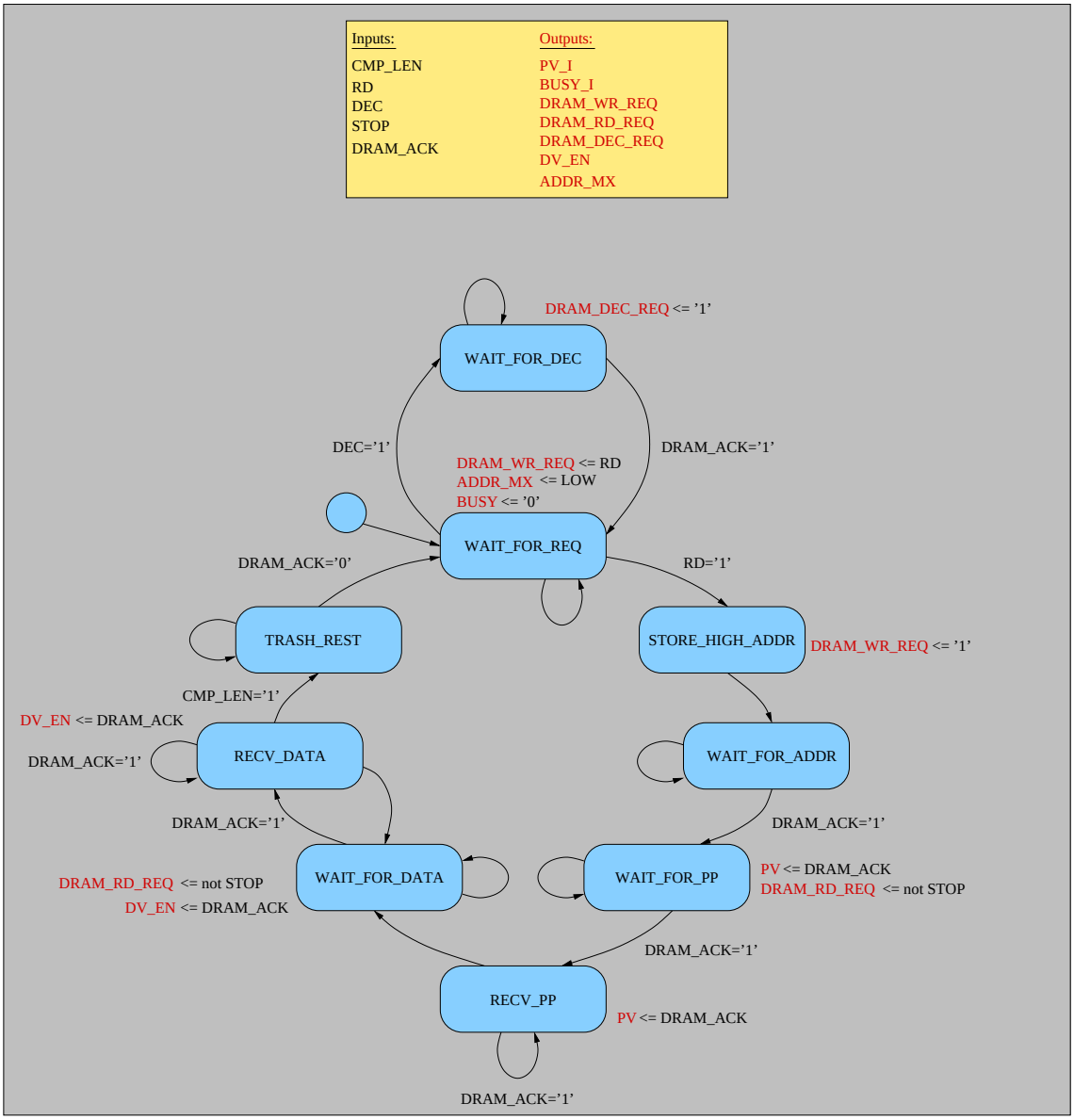
DRAM_WR_REQ. Řadič dynamické paměti uchovává pro každý paket v paměti počet aktivních odkazů, proto když dojde k odeslání paketu, musí být tento čítač snížen. To se provádí pulsem na portu DRAM_DEC_REQ. U multicastových (replikovaných) paketů se počet odkazů nesnižuje. Adresa bloku dynamické paměti je 16-bitová a řadiči dynamické paměti se posílá po osmi bitech (viz multplexor addr_mx). Na portu PV signalizuje automat Vstupnímu bufferu, že na portu DO jsou platné paketové parametry (viz popis následujícího bloku). Podrobný popis automatu bude uveden později.

Druhý blok obsahuje logiku, která zprostředkovává data vyčtená z dynamické paměti Vstupnímu bufferu. Na port DO jsou přiváděna data čteného paketu. Jeho délka je obsažena v paketových parametrech na pozici PP_PCKT_LEN. Čítač cnt_pp počítá přijatá slova paketových parametrů a na pozici obsahující délku paketu se nastaví výstup komparátoru povolující zápis do registru reg_len. V tomto registru je tedy uložena délka paketu v bajtech. Čítač cnt_data počítá přijatá data paketu (v 16-bitových slovech) a komparátor cmp_len je porovnává s hodnotou v registru reg_len dělenou dvěma a zvýšenou o zbytek. Výstup

komparátoru je přiveden na port EOP, kterým Řadič externí paměti oznamuje Vstupnímu bufferu, že přijal poslední slovo paketu.

Třetí blok slouží ke generování dvoubitového signálu DV (Data Valid - na portu DO jsou platná data paketu). Každý bit portu DV odpovídá jednomu bajtu portu DO. U paketů s lichou délkou je u posledního slova nastaven pouze dolní bit DV.

Následuje podrobný popis automatu DRAM_CTRL_FSM, jehož stavový diagram a seznam vstupních a výstupních portů je na obrázku 4.7.



Obrázek 4.7: Stavový diagram automatu DRAM_CTRL_FSM

Po resetu se automat nachází ve stavu *WAIT_FOR_REQ*, kde čeká na žádost od Vstupního bufferu. Požádá-li Vstupní buffer o snížení počtu odkazů na paket (pulsem na portu DEC), přejde automat do stavu WAIT_FOR_DEC. Bude-li Vstupní buffer žádat čtení z paměti (pulsem na portu RD), zapíše automat do řadiče dynamické paměti první bajt adresy

(nastaví DRAM_WR_REQ a multiplexor ADDR_MX ponechá v nule) a přejde do stavu *STORE_HIGH_ADDR*.

Ve stavu *WAIT_FOR_DEC* čeká automat, až bude provedeno snížení počtu odkazů na paket. To je signalizováno řadičem dynamické paměti pulsem na portu DRAM_ACK a automat se vrátí do výchozího stavu.

Ve stavu *STORE_HIGH_ADDR* se provede zápis horní části adresy (nastaví DRAM_WR_REQ a ADDR_MX je nastaven do jedničky) a automat v přístím taktu přejde do stavu *WAIT_FOR_ADDR*.

V tomto stavu čeká automat na potvrzení přijaté adresy (pulsem na portu DRAM_ACK). Poté přejde do následujícího stavu.

Ve stavu *WAIT_FOR_PP* nastaví automat požadavek na čtení (port DRAM_RD_REQ) a čeká na příjem prvního bloku dat z dynamické paměti. Řadič dynamické paměti posílá data vždy v 32 bajtových blocích (16-krát 16 bitů). První blok obsahuje Paketové parametry. Jakmile se objeví na výstupu (port DRAM_ACK je nastaven), nastaví automat port PV a přejde do dalšího stavu.

Ve stavu *RECV_PP* probíhá příjem paketových parametrů. Ukončení prvního bloku je signalizováno nulováním portu DRAM_ACK a automat přejde do stavu *WAIT_FOR_DATA*.

V tomto stavu žádá automat o vyčtení dalšího bloku. Není-li ve Vstupní bufferu volné místo, nastaví port STOP a čtení dalšího bloku se odloží. Platná data jsou signalizována na portu DV_EN. Automat přejde do stavu *RECV_DATA*.

V tomto stavu probíhá příjem bloku dat. Jedná-li se o poslední blok, pak se při příjmu posledního slova nastaví port CMP_LEN a automat přejde do stavu *TRASH_REST*. Není-li to poslední blok, vrátí se do stavu *WAIT_FOR DATA*.

Ve stavu *TRASH_REST* počká automat, až skončí příjem posledního bloku a vrátí se do výchozího stavu.

Následuje popis Vstupního bufferu, do něhož se ukládají data z proudu. IStB je rozdělen do dvou banků, přičemž v jednom banku jsou data zpracovávaného paketu (aktivní bank) a do druhého se načítají data dalšího paketu. Je-li druhý bank prázdný, je možné pracovat i nad daty, která se teprve načítají. V tomto případě se také jedná o aktivní bank. To je nutné z toho důvodu, že se data načítaného paketu nemusejí celá vlést do svého banku. V tom případě se načítaní zastaví do doby, než se místo v banku uvolní.

Data v aktivním banku tvoří *okno* v datech paketu. K datům v okně může přistupovat Aplikační procesor, který může okno posouvat směrem ke konci paketu. Je-li okno posunuto, jsou data před oknem uvolněna z bufferu a na jejich místo se načtou nová data. Z tohoto důvodu nelze oknem pohybovat zpět.

S oknem lze manipulovat pomocí operací vstupního proudu a souboru 16-bitových registrů v Proudovém registrovém poli. Registry vstupního proudu jsou pouze pro čtení. Následuje jejich seznam:

- ISR - čtecí ukazatel
- ISB - pozice v paketu, kde je začátek okna
- ISE - pozice v paketu, kde je konec okna
- ISD - data na pozici odkazované ISR

Registry ISR, ISB a ISE odkazují na data zarovnaná na osm bitů. Hodnoty v těchto registrech se počítají od nuly vzhledem k počátku paketu.

Následuje seznam operací vstupního proudu. Operand operace se uloží na adresu v Příkazovém bloku Datové paměti odpovídající žádané operaci. Operace se provede po zapsání operandu:

cmd_rd – Čti data z proudu. Realizuje se standartní instrukcí MOV Aplikačního procesoru, kde jako zdrojový operand je použit registr ISD. Tato operace má dva režimy v závislosti na registru rd_mode:

- rd_mode je vynulován. Po přečtení dat z ISD registru je hodnota registru ISR automaticky zvýšena o dva (adresuje se po bajtech).
- rd_mode je nastaven. Po přečtení dat z ISD registru je zvýšena o dvě také hodnota registru ISE. Takto lze zároveň číst a posouvat okno.

cmd_set_ISR – Přesuň ISR na absolutní pozici v paketu. Operand udává novou absolutní pozici.

cmd_set_ISRE – Přesuň ISR na absolutní pozici v paketu a nastav ISE <- ISR. Operand udává novou absolutní pozici. Jako operand lze použít hodnotu registru ISR - pak se změní pouze hodnota registru ISE.

cmd_sndtopos_ISR – Odešle data do výstupního proudu od aktuální pozice ISR po abs. pozici (nevčetně) a posouvá ISR. Operand udává novou absolutní pozici.

cmd_sndtopos_ISRE – Odešle data od aktuální pozice ISR po abs. pozici (nevčetně) a posouvá ISR a provádí ISE <- ISR. Operand udává novou absolutní pozici.

cmd_sndnfwrd_ISR – Odešle určitý počet slov od aktuální pozice ISR (včetně) a posouvá ISR. Operand udává počet 16-bitových slov k odeslání.

cmd_sndnfwrd_ISRE – Odešle určitý počet slov od aktuální pozice ISR (včetně) a posouvá ISR a provádí ISE <- ISR. Operand udává počet 16-bitových slov k odeslání.

cmd_sndrest_ISRE – Odešle vše od ISR až po konec paketu a vyprázdní okno. Operace se provede po uložení jakékoli hodnoty operandu.

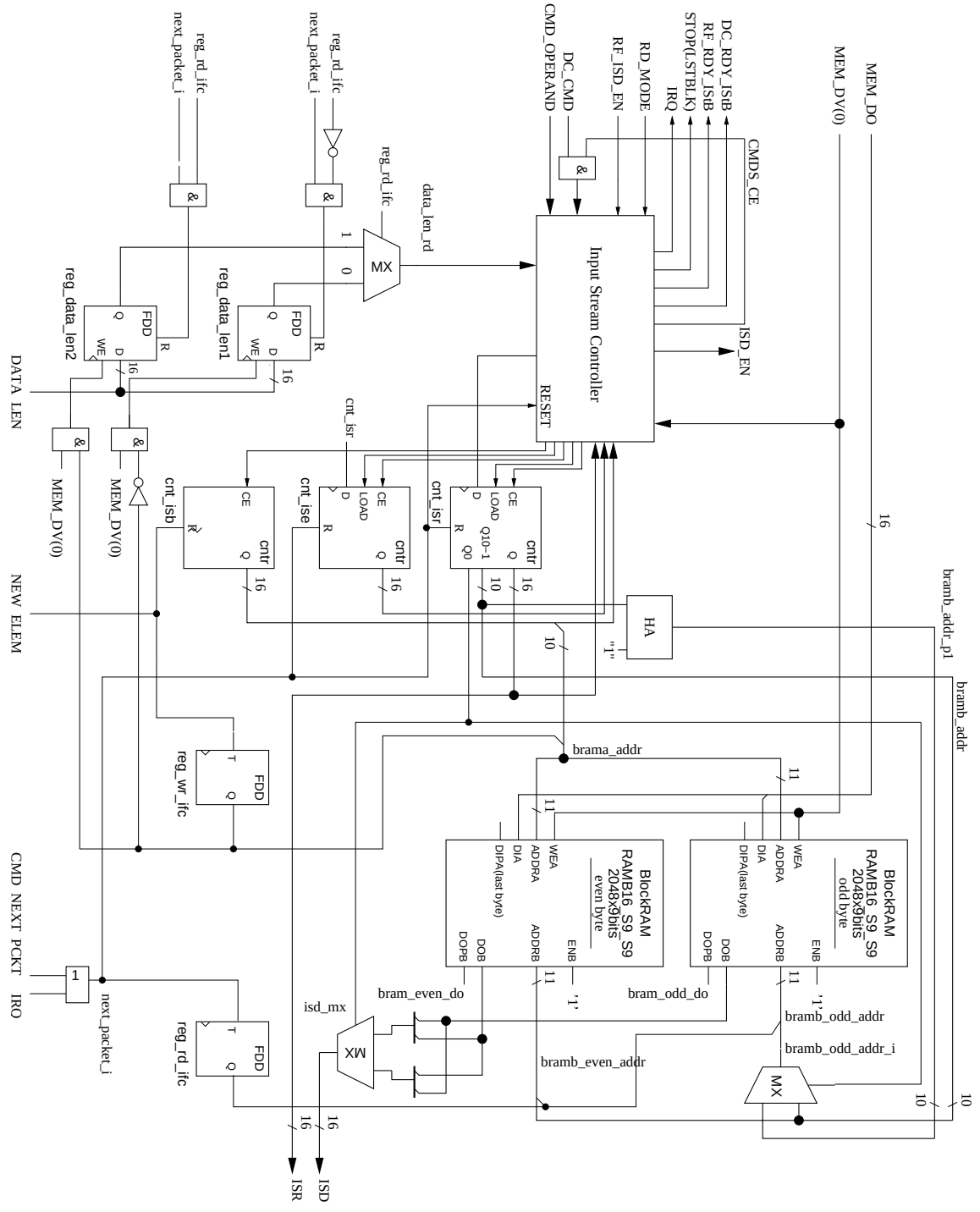
Operace cmd_sndtopos_ISR, cmd_sndtopos_ISRE, cmd_sndnfwrd_ISR, cmd_sndnfwrd_ISRE a cmd_sndrest_ISRE ovládají také Řadič výstupního proudu.

Pokud dojde k neošetřitelné situaci, je vyvoláno přerušení. Zdroje přerušení jsou tyto:

- ISR skočil za fyzickou velikost paketu.
- ISR skočil na pozici zpět, která je menší než ISE.
- ISR skočil dopředu na pozici, která nemůže být nikdy načtena vlivem malého okna.

Následuje popis činnosti uložení paketu do Vstupního bufferu.

1. Buffer čeká na data z proudu.
2. Data z proudu ukládá do zápisového banku.
3. Jakmile se do banku zapíší poslední bajty paketu, přepne se zápisový bank a Buffer přejde do bodu 1.



Obrázek 4.8: Architektura Vstupního bufferu

Následuje popis činnosti čtení paketu z aktivního banku Vstupního bufferu.

1. Aplikační procesor čeká, až budou v aktivním banku připravena data.
2. Aplikační procesor čte data z aktivního banku. Nejsou-li požadovaná data připravena, přejde do bodu 1.
3. Dojde-li k neošetřitelné situaci, je vyvoláno přerušení, dojde k přepnutí aktivního banku a Aplikační procesor přejde do bodu 1.
4. Aplikační procesor dokončil zpracování paketu - žádá nový paket.
5. Dojde k přepnutí aktivního banku. Pokračuje bodem 1.

Architektura Vstupního bufferu je znázorněna na obrázku 4.8. Jádrem komponenty je Řadič vstupního bufferu (Input Stream Controller), který dekóduje operace vstupního bufferu na sekvenci signálů realizujících tuto operaci a v případě chyby zajistí generování přerušení (port IRQ). V závislosti na prováděné operaci generuje READY status pro Proudové registrové pole (port RF_RDY_IStB) a datovou paměť aplikačního procesoru (port DC_RDY_IStB) - viz kapitola 4.4.4. To je nutné pro případ, kdy by program vyžadoval čtení z některého registru vstupního proudu, jehož hodnota není ještě platná. Např. program procesní jednotky obsahuje posloupnost instrukcí, kdy nejprve vyvolá operaci cmd_sndnfwrd_ISR a v přístí instrukci čte z registru ISD. Čtecí operace musí být proto pozdržena do doby, než skončí operace vstupního proudu a hodnota v registru ISD bude platná. Podobně může nastat, že program žádá provedení dvou po sobě jdoucích operací vstupního proudu. Vyvolání druhé operace musí být pozdrženo, než se dokončí operace první. Toho je docíleno tak, že se procesorové jádro při zápisu do příkazové paměti pozastaví.

Důležitou součástí Vstupního bufferu jsou dvě dvouportové paměti typu BlockRAM (bram), které realizují samotný buffer. Prvním portem (port A) se zapisují data paketu přicházející z Řadiče externí paměti (porty MEM_DO a MEM_DV) a druhý port (port B) slouží pro čtení. Tím je zajištěno, že se mohou data do bufferu nahrávat a zároveň s nimi může Aplikační procesor pracovat.

Buffer je rozdělen do dvou banků. Hodnota aktuálního čtecího banku (tedy aktivního banku) je uložena v registru reg_rd_ifc a hodnota aktuálního zápisového banku je v registru reg_wr_ifc. Hodnoty těchto registrů tvoří horní bit adresy čtecího (signály bramb_odd_addr a bramb_even_addr), resp. zápisového portu (signál brama_addr).

Data v bufferu jsou 16-bitová, zarovnaná na 8 bitů, proto se do jedné bram ukládají liché bajty a do druhé sudé bajty. Čítače cnt_isr, cnt_isb a cnt_ise se nazývají *adresové čítače* a generují hodnoty registrů ISR, ISB a ISE a jsou řízeny Řadičem vstupního bufferu. Velikosti čítačů jsou 16 bitů, tzn. že délka paketu může být 65536 bajtů. Velikost 1 banku bram paměti je pouze 1kB a proto se k jejich adresování používá jen dolních deset bitů čítačů. Je-li paket delší než 1kB, je nutné pozastavovat přísun dat z Řadiče externí paměti (portem STOP), aby bank nepřetekl.

Kód operace vstupního bufferu je přítomen na portu DC_CMD a její operand na portu CMD_OPERAND. Hodnota na port RF_ISD_EN je generována Proudovým registrovým polem a puls na tomto portu znamená, že program procesní jednotky přečetl hodnotu registru ISD, tzn. že se ukazatel ISR musí posunout na další pozici. V závislosti na hodnotě portu RD_MODE bude jeho hodnota zvýšena buď o jedna, nebo o dvě. Pulsem na portu ISD_EN oznamuje Řadič vstupního proudu Řadiči výstupního proudu (viz kapitola 4.4.5), aby odeslal na výstup data z registru ISD. To je využíváno při provádění operací autonomního odesílání dat.

Hodnota registru ISD je generována z výstupů druhých portů obou bram pamětí. V případě, že ukazatel ISR ukazuje na lichou pozici v bufferu, nejsou data v registru ISD zarovnána na 16 bitů. V tomto případě je čtecí adresa liché bram paměti zvýšena o jedničku (signál bramb_addr_p1) a na výstupu jsou lichý a sudý bajt prohozeny pomocí multiplexoru isd_mx.

Délky paketů v obou bancích jsou uloženy v registrech reg_data_len1 a reg_data_len2. Délka načítaného paketu je přivedena na port DATA_LEN Řadičem externí paměti a je zapsána do odpovídajícího registru.

Porty NEW_ELEM A CMD_NEXT_PCKT jsou řízeny Hlavním řadičem Aplikačního procesoru (viz kapitola 4.4.4). Puls na portu NEW_ELEM signalizuje příjem nového paketu, proto při jeho příchodu dojde změně hodnoty registru reg_wr_ifc (bude se zapisovat do druhého banku) a vynuluje se hodnota čítače cnt_isb, jehož hodnota slouží jako adresa pro zápisový port bufferu. Původní hodnota čítače je uložena do registru a slouží jako hodnota registru ISB druhého banku. Puls na portu CMD_NEXT_PCKT znamená vyvolání operace hlavního řadiče cmd_next_pckt. To znamená že se aktivní bank přepne na následující paket, tzn. že dojde k přehození čtecího banku. To se provede změnou hodnoty v registru reg_rd_ifc a vynulováním čítačů cnt_isr a cnt_ise. Dále se na vstup registru ISB přivede odpovídající hodnota. Následuje popis činnosti Řadiče vstupního bufferu.

1. Čeká na žádost o provedení operace vstupního bufferu.

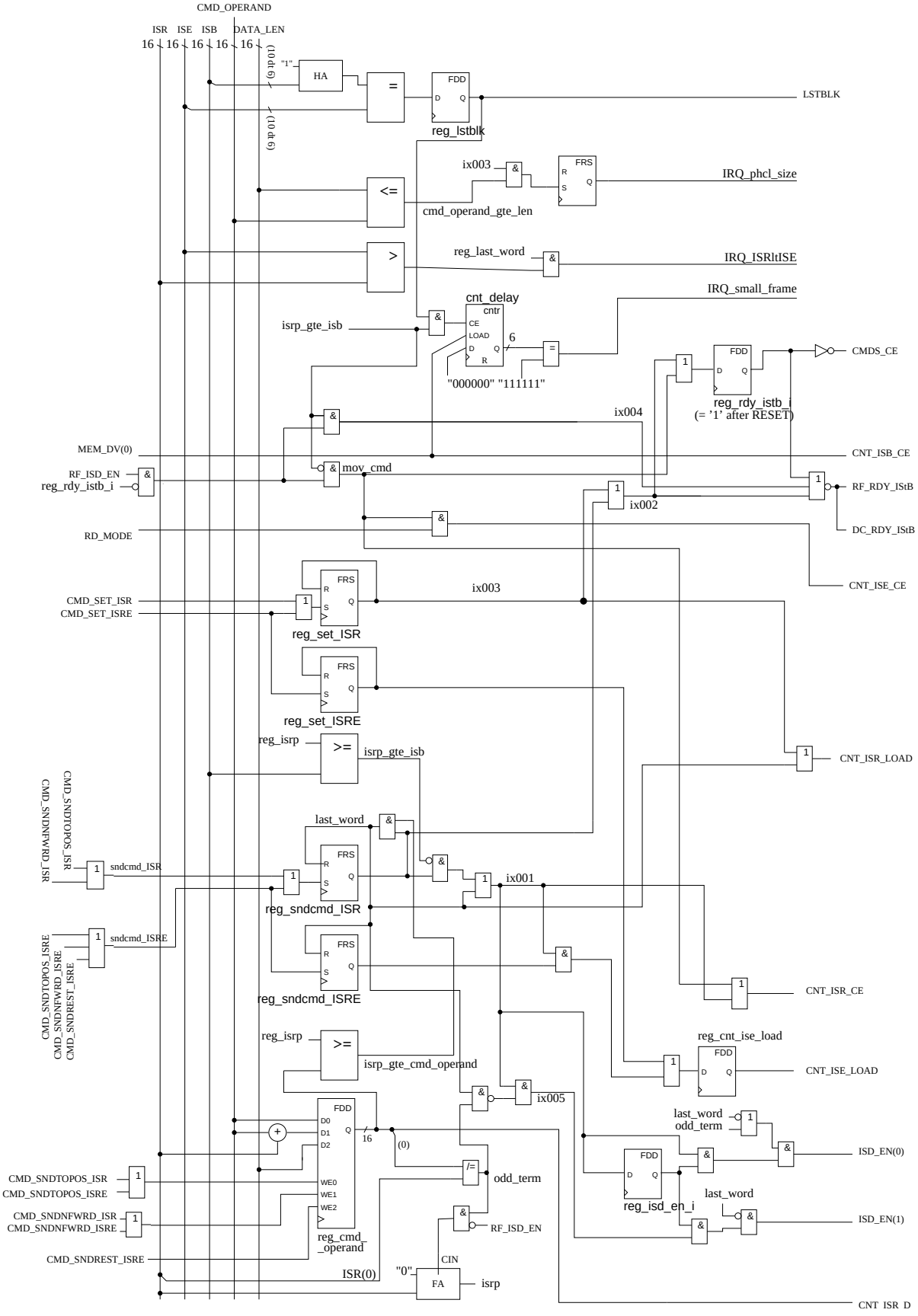
2. Provádí operaci vstupního bufferu. Je nulován ready status Proudového registrového pole a Datové paměti. Průběžně jsou aktualizovány ukazatele ISR a ISE. Generuje signály pro Řadič výstupního proudu, aby odesílal data na pozici dané ukazatelem ISR.

3. Operace byla dokončena. Ready status je nastaven a hodnoty ISR a ISE odpovídají novému stavu. Přejde do stavu 1.

Následuje podrobný popis architektury Řadiče vstupního proudu, která je zobrana na obrázku 4.9. V levém horním rohu jsou na vstup přivedeny hodnoty registrů vstupního proudu, operand operace vstupního proudu a délka paketu. Na levé straně schématu se nacházejí ostaní vstupy, především jednotlivé dekódované operace. Na pravé straně schématu jsou výstupy řadiče.

Port LSTBLK signalizuje, že je zápisový bank téměř zaplněn. Je generován komparátorem, na jehož vstupy je přivedeno horních 5 bitů registru ISE a horních pět bitů registru ISB zvýšených o 1.

Port IRQ_phcl_size signalizuje přerušení způsebené skokem za fyzickou velikost paketu. Je-li hodnota operandu větší než délka paketu, je v případě volání operace cmd_set_isr nebo cmd_set_isre (indikováno signálem ix003) vyvoláno přerušení. Port IRQ_ISRltISE signalizuje přerušení způseobené posunem ukazatele ISR před ukazatel ISE, tedy před okno. Tento příznak není kontrolován v průběhu provádění operace vstupního proudu, což je zajištěno logickou funkcí AND se signálem reg_last_word. Port IRQ_small_frame signalizuje přerušení v důsledku skoku na pozici, která nemůže být nikdy načtena vlivem malého okna. Hodnota tohoto portu je generována příznakem, kdy je ISR větší než ISB. Může nastat případ, kdy ISR skočí na pozici větší než ISB, avšak v banku je stále volné místo a tak se načítají další data a ISB se zvětšuje. Nakonec může hodnota ISB „přerůst“ ISR. Proto je toto přerušení generováno až se zpožděním, které vylučuje falešný poplach. Zpoždění je zajištěno čítačem cnt_delay.



Obrázek 4.9: Architektura Řadiče vstupního bufferu

Port CMDS_CE povoluje na vstupu dekódované operace, aby během vykonávání jedné operace nemohla přijít další. Jádro je sice v tomto případě pozastaveno, avšak zapisovaná operace je aktivní na výstupu z příkazové paměti.

Porty CNT_ISB_CE, CNT_ISR_CE, CNT_ISR_LOAD, CNT_ISR_D, CNT_ISE_CE a CNT_ISE_LOAD slouží k řízení adresových čítačů.

Pulsem na dvoubitovém portu ISD_EN oznamuje Řadič vstupního proudu Řadiči výstupního proudu (viz kapitola 4.4.5), aby odeslal na výstup data z registru ISD. Je-li puls přítomen pouze na dolním bitu, odešle OSC pouze dolních 8 bitů registru ISD. Je-li přítomen i na horním bitu, pak odešle všech 16 bitů.

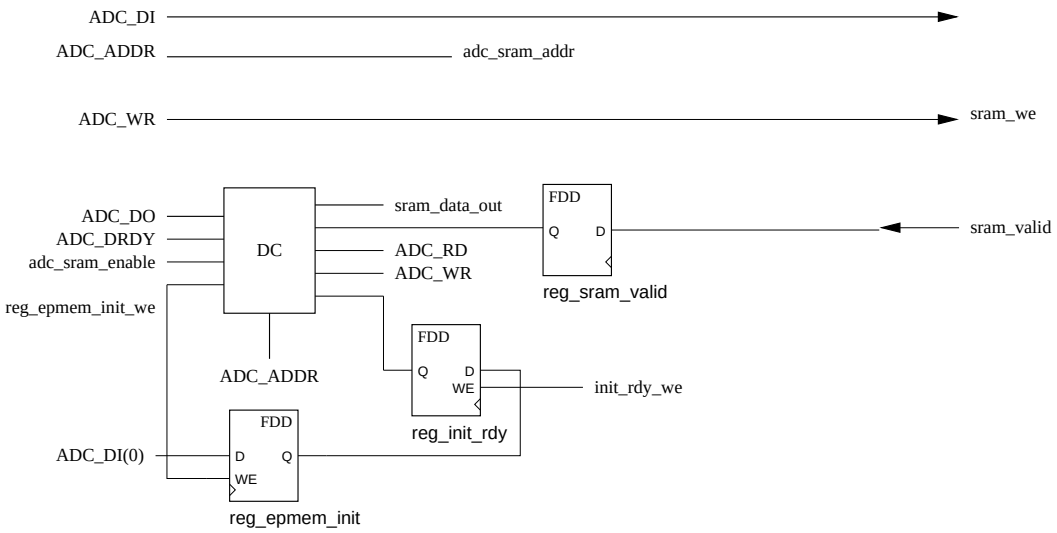
Registr reg_cmd_operand je vícevstupový a uchovává upravenou hodnotu operandu. Většina operací vstupního proudu je de-facto modifikovanou operací cmd_sndtopos_xxx. Stačí jen příslušným způsobem upravit operand tak, aby ukazoval na cílovou pozici ukazatele ISR, pak ho postupně inkrementovat odesílat jednotlivá slova. Jedná-li se o variantu s modifikací ukazatele ISE (cmd_xxx_ISRE), pak se hodnota registru ISR přenese i do ISE. Přítomnost tohoto druhu operace je signalizována registrem reg_sndcmd_ISR, resp. reg_sndcmd_ISRE.

Operace cmd_set_xxx je samostatným druhem operace a její přítomnost je signalizována registrem reg_set_ISR, resp. reg_set_ISRE.

Přečtení hodnoty registru ISD vyvolá poslední z možných operací vstupního bufferu, kdy se hodnota ukazatele ISR zvýší o jedna, resp. o dvě (v závislosti na RD_MODE). Přítomnost této operace je značena signálem mov_cmd.

4.4.3 Řadič paměti procesní jednotky (OPE Memory Controller, OMC)

Řadič paměti procesní jednotky slouží k získání Editačních parametrů z Paměti procesní jednotky a to pro více Editačních jednotek zároveň. Obsahuje arbitr typu Round-Robin, který určuje pořadí, v jakém budou žádosti z Editačních jednotek uspokojovány. OMC je tvořen dvěma řadiči: Edit Parameters Memory Controllerem (EPMC) a SSRAM Controllerem. První řadič funguje jako arbitr jednotlivých EU, druhý řadič realizuje komunikační protokol s Pamětí procesní jednotky, která je tvořena synchronní statickou pamětí [19].



Obrázek 4.10: Architektura Řadiče paměti procesní jednotky - Software interface part

Žádá-li EU o Editační parametry, nasune pomocí posuvného registru do EPMC adresu požadovaného bloku (porty EPMC_ADDR a EPMC_AV) a nastaví příznak žádosti (EPMC_REQ). EU poté čeká na vyhovění žádosti. Jakmile se na ni dostane řada, dá EPMC příkaz SSRAM Controlleru, který načte postupně celý blok z Paměti procesní jednotky. EPMC vystavuje tato data na portu EPMC_DO a jejich platnost signalizuje na portu EPMC_ACK.

OMC je dále napojen na Softwarové rozhraní, které zpřístupňuje Paměť procesní jednotky softwarovým ovladačům, jež mají za úkol paměť inicializovat. Během inicializace je SSRAM Controller řízen Softwarovým rozhraním a vyřizování žádostí Editačních jednotek je pozastaveno. Následuje popis činnosti Řadiče paměti procesní jednotky.

1. Čeká na žádost od některé EU.
2. Přišla žádost. EU nasune do řadiče adresu.
3. Řadič cyklicky kontroluje příznak žádosti z jednotlivých EU.
4. Je-li příznak aktivní, provede vyčtení požadovaného bloku z paměti.
5. Jakmile jsou data z paměti platná, oznámí to řadič příslušné EU a čeká, než se přečte poslední bajt bloku.
6. Řadič přejde do bodu 3.

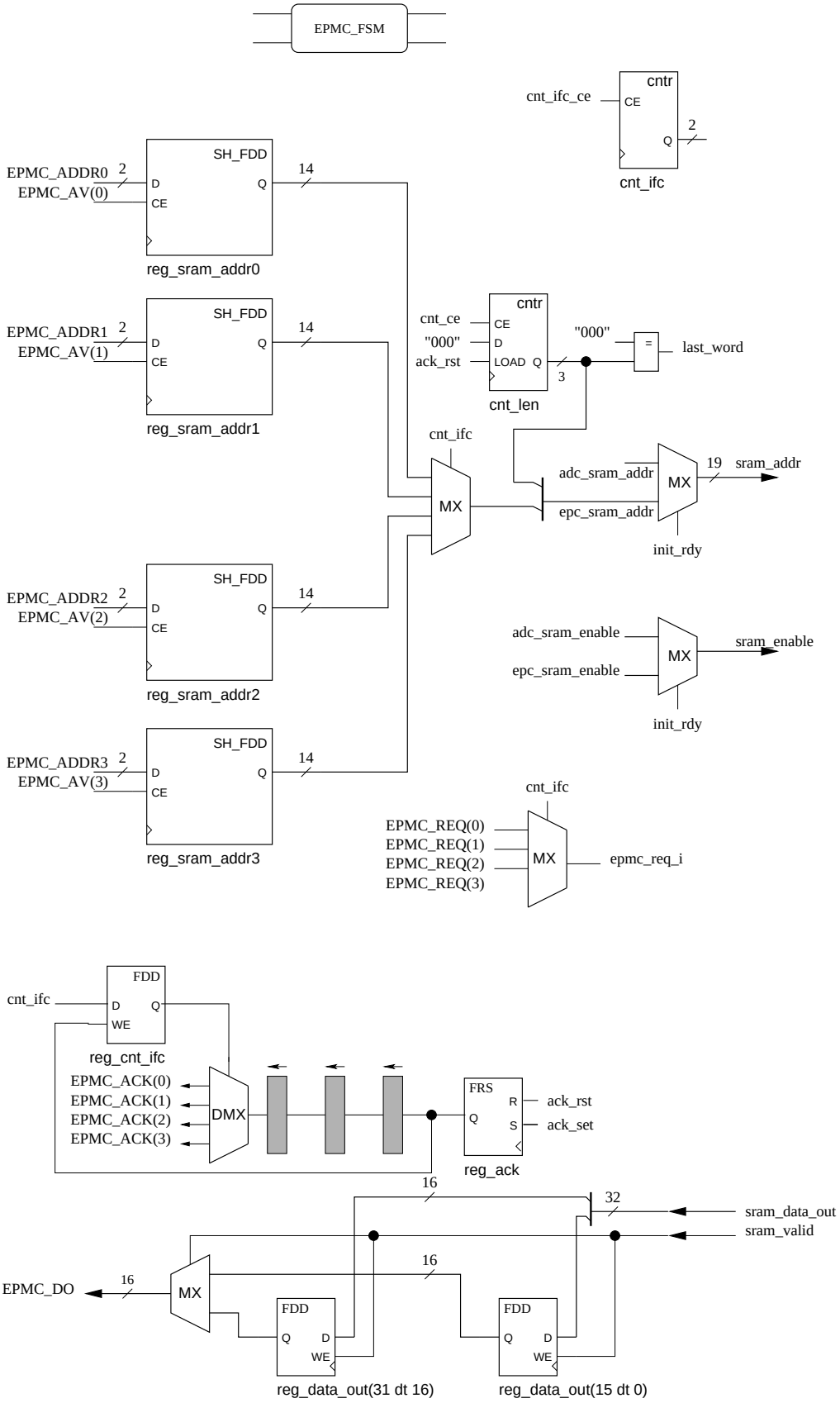
Architektura EPMC je rozdělena do dvou částí - první (tzv. EPC part) je propojena se 4 Editačními jednotkami (konkrétně s Řadičem editačních parametrů EPC - viz následující kapitola) porty EPMC_xxx a druhá část (Software interface part) pak se Softwarovým rozhraním porty ADC_xxx. Obě části jsou napojeny na SSRAM Conroller porty sram_xxx.

Software interface part je znázorněna na obrázku 4.10. Je tvořena adresovým dekodérem ADC_ADDR a dále třemi registry reg_epmem_init, reg_init_rdy a reg_sram_valid. První dva registry slouží k přepínání režimů inicializováno(1)/neinicializováno(0). V režimu neinicializováno jsou data v Paměti editačních parametrů neplatná a ovládání SSRAM Controlleru je napojeno na Softwarové rozhraní.

O přepnutí do režimu neinicializováno zažádá softwarový ovladač zápisem nuly do registru reg_epmem_init. Ke změně režimu však může dojít pouze ve chvíli, kdy není vyřizována žádná žádost od některé z EU. To je signalizováno signálem init_rdy_we, který povolí přenos hodnoty z registru reg_epmem_init do registru reg_init_rdy. Přepnutí do režimu inicializováno probíhá analogicky, pouze se do registru reg_epmem_init zapíše jednička.

EPC part je znázorněna na obrázku 4.11. Je tvořena třemi bloky. První blok obsahuje automat EPMC_FSM a čítač cnt_ifc. Tento čítač cyklicky mění index aktivního rozhraní EU. Žádost z aktivního rozhraní je vyřízena, hodnota čítače se zvýší a proces se opakuje. V případě, že aktivní rozhraní nežádá o čtení, je hodnota čítače zvýšena neprodleně. Automat EPMC_FSM má na starost generování řídicích signálů a řízení SSRAM Controlleru v režimu inicializováno. Jeho detailní popis následuje níže.

Druhý blok slouží k předání žádosti z aktivního rozhraní SSRAM Controlleru. Je tvořen čtyřmi posuvnými registry reg_sram_addrx. Do těchto registrů nasouvají EU adresu požadovaného bloku. Jejich výstupy vedou do multiplexoru, který přepíná na výstup registr z aktivního rozhraní (je řízen čítačem cnt_ifc). Výstup multiplexoru tvoří horní část adresy. Dolní část je generována čítačem cnt_len, který je řízen automatem. Obě části tvoří adresu epc_sram_addr. Ta je přivedena na vstup druhého multiplexoru spolu s adresou

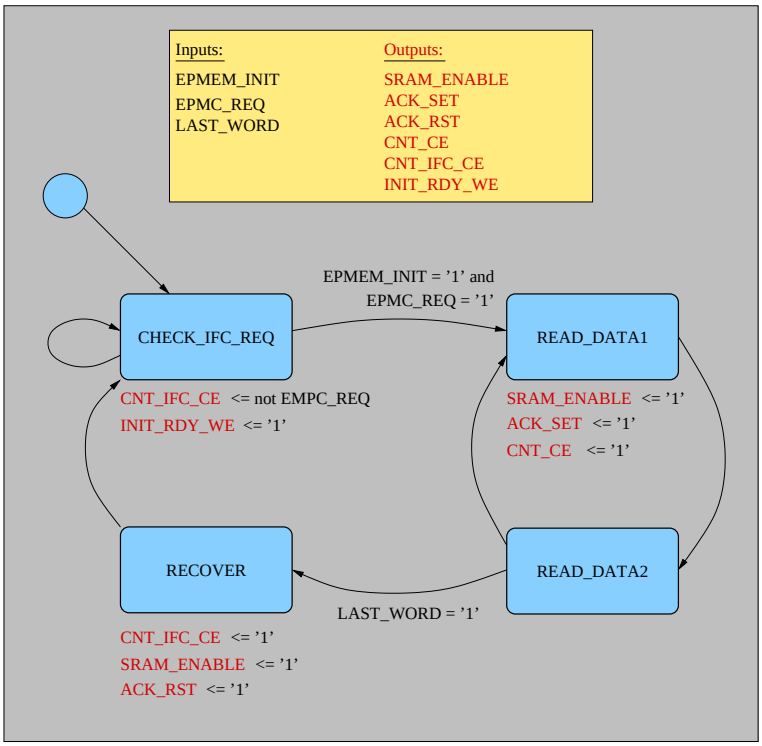


Obrázek 4.11: Architektura Řadiče paměti procesní jednotky - EPC part

adc_sram_addr generovanou Softwarovým rozhraním. Obě adresy jsou přepínány hodnotou v registru init_rdy, tedy v režimu neinicializováno je to adresa adc_sram_addr a v režimu inicializováno adresa epc_sram_addr. Na výstupu čítače cnt_len je komparátor, který oznamuje automatu, že se jedná o poslední adresu bloku Editačních parametrů. Ve druhém bloku se nacházejí ještě dva multiplexory. První přivádí na vstup automatu žádost o čtení z aktivního rozhraní (signál empc_req_i) a druhý v závislosti na režimu přepíná vstupy pro port sram_enable.

Třetí blok slouží k předání vyčtených dat z SSRAM Controlleru (port sram_do) aktivnímu rozhraní (port EPMC_DO). 32-bitová data z portu sram_do jsou uložena do registru reg_data_out. Port EPMC_DO je 16-bitový a jsou na něj postupně vystaveny obě poloviny registru reg_data_out. Platnost dat na portu EPMC_DO je signalizována na aktivním portu EPMC_ACK. Porty EPMC_ACK pro jednotlivá rozhraní jsou napojeny na demultiplexor řízený registrem reg_cnt_ifc uchovávajícím adresu rozhraní, kterému náleží vyčtená data. Na vstup demultplexoru je přivedena hodnota registru reg_ack, která je nastavována automatem.

Následuje popis automatu EPMC_FSM. Jeho stavový diagram a seznam vstupních a výstupních signálů je na obrázku 4.12.



Obrázek 4.12: Stavový diagram automatu EPMC_FSM

Po resetu se automat nachází ve stavu *CHECK_IFC_REQ*, kdy zkontroluje, zda aktivní rozhraní nežádá o čtení (signál EPMC_REQ). V tomto stavu je povolena změna režimu incializováno/neinicializováno nastavením portu INIT_RDY_EN. Je-li aktivní režim inicializováno a je-li přítomna žádost o čtení, povolí automat inkrementaci čítače cnt_ifc (signál CNT_IFC_CE) a přejde do stavu *READ_DATA1*.

V tomto stavu je generován čtecí požadavek na SSRAM Controller (signál SRAM_EN) a je povolena inkrementace čítače cnt_len (signál CNT_CE). Automat dále nastaví registr reg_ack (signál reg_ack_set). V dalším taktu hodin přejde automat do stavu *READ_DATA2*.

V tomto stavu je čtení pozastaveno a čeká se, až se data z portu ssram_do vyčtou portem EPMC_DO. Jedná-li se o poslední slovo (port LAST_WORD je nastaven), pak přejde automat do stavu *RECOVER*, jinak se vrátí zpět do stavu READ_DATA1.

Ve stavu *RECOVER* se povolí inkrementace čítače cnt_ifc, nuluje se registr reg_ack a automat přejde do výchozího stavu *CHECK_IFC_REQ*.

4.4.4 Aplikační procesor (Application Processor, AP)

Aplikační procesor vykonává program procesní jednotky a řídí všechny proudové klienty. Skládá se z generického procesorového jádra GENA [8], Instrukční paměti (Instruction Cache - IC) a Datové paměti (Data Cache - DC). K autonomnímu (tedy ne programem) řízení proudových klientů je Aplikační procesor vybaven Hlavním řadičem (Main Controller - MNC). Dále je vybaven Jednotkou pro načítání editačních parametrů (Edit Parameters Controller - EPC). Architektura Aplikačního procesoru je znázorněna na obrázku 4.13.

Procesorové jádro GENA je 16 bitové a typu RISC. Obsahuje běžné aritmetické a logické instrukce, instrukce posuvu a rotací a skokové instrukce.

Instrukční paměť je realizována jako dvouportová 32-bitová paměť velikosti 2 kB. Je napojena na Řadič softwarového rozhraní, který ji zpřístupňuje systémovým ovladačům.

Lokální registrové pole je realizována sadou 32-bitových registrů, které jsou k dispozici programu pro libovolné využití.

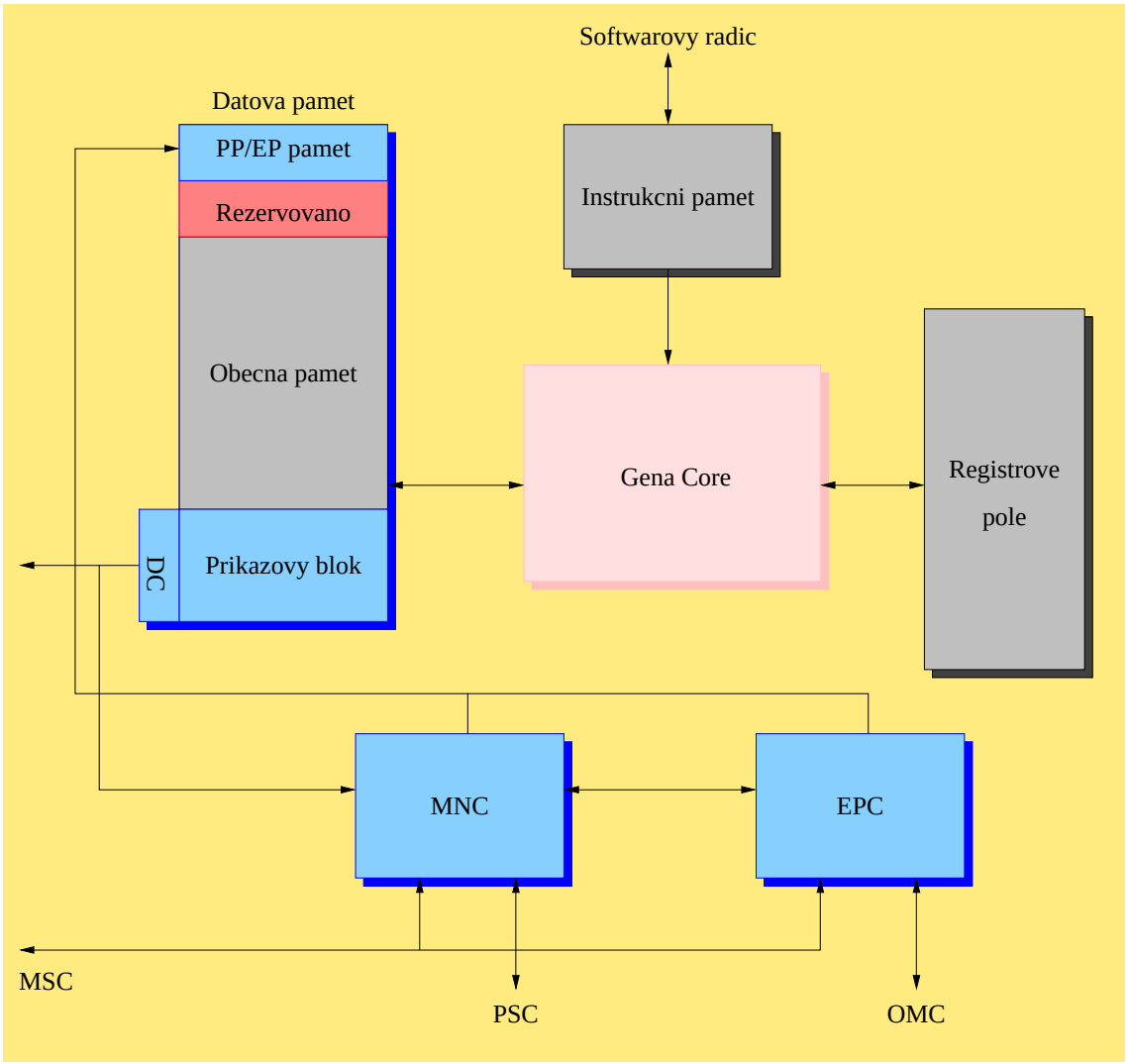
Datová paměť je realizována jako dvouportová 16-bitová paměť velikosti 4 kB. Prvním portem k ní přistupuje procesorové jádro a druhý port je sdílen řadiči MNC a EPC, které ho využívají pro nahrávání paketových a editačních parametrů. Adresový prostor je rozdělen do několika hlavních bloků:

0x000 - 0x00F: Paměť paketových parametrů – obsahuje paketové parametry zpracovávaného paketu. Jsou načítany autonomně Řadičem proudu dat paketů. Paměť paketových parametrů je rozdělena do dvou banků - pro zpracovávaný paket a načítaný paket. Pokud není ani v jednom z banků kompletně načtena sada paketových parametrů, je Datová paměť v režimu *nepřipravena*.

0x010 - 0x01F: Paměť editačních parametrů – obsahuje editační parametry zpracovávaného paketu. Jsou načítany autonomně Jednotkou pro načítání editačních parametrů (EPC). Paměť editačních parametrů je rozdělena do dvou banků - pro zpracovávaný paket a načítaný paket. Pokud není ani v jednom z banků kompletně načtena sada editačních parametrů, je Datová paměť v režimu *nepřipravena*.

0x020 - 0x03F: Rezervováno – vyhrazeno pro neaktivní banky Paměti paketových a editačních parametrů.

0x040 - 0x3FF: Obecná paměť – k dispozici programu pro ukládání libovolných dat.



Obrázek 4.13: Architektura aplikačního procesoru

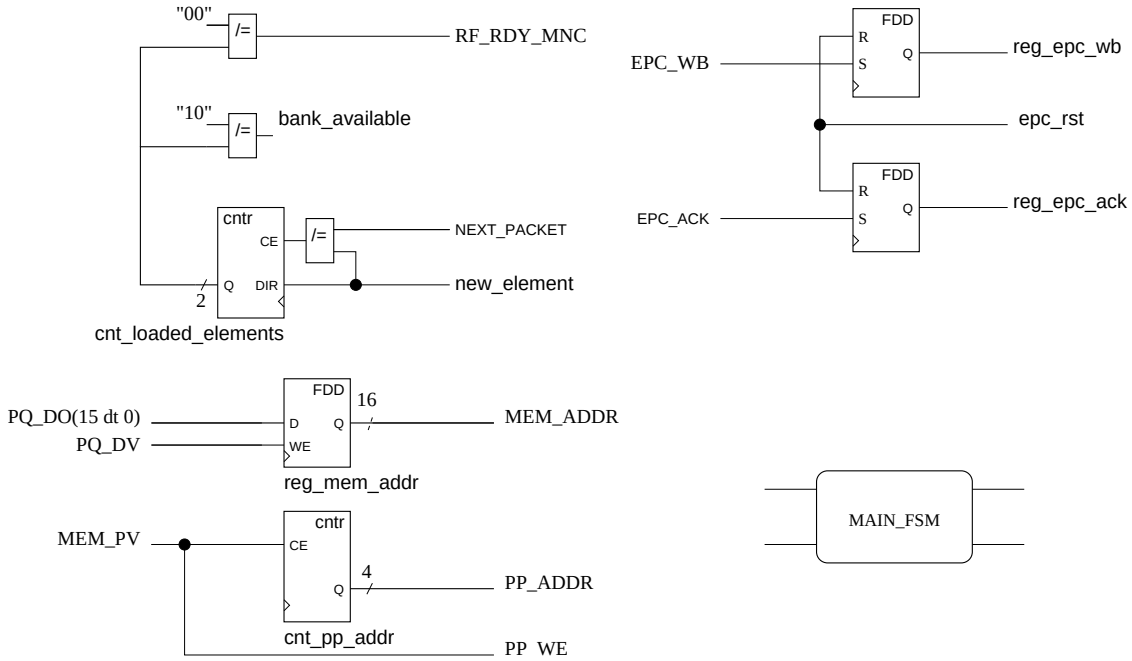
0x400 - 0x7FF: Příkazový blok – je určen k zpřístupnění operací proudových klientů programu. Každá adresa Příkazového bloku odpovídá konkrétní operaci některého proudového klienta. Jedná-li se o operaci s parametrem, je operace vyvolána uložením hodnoty parametru na adresu operace. Jedná-li se o bezparametrickou operaci, je operace vykonána po zápisu libovolné hodnoty na adresu operace.

Datová paměť je vybavena dekodérem operací, zapisovaných do příkazového bloku. Výstupem dekodéru jsou povely pro jednotlivé proudové klienty a Hlavní řadič Aplikačního procesoru. Během provádění operace konkrétním klientem není možné zapsat nový příkaz pro téhož klienta. Žádá-li jádro o takový zápis, přepne se Datová paměť do stavu nepřipravena a činnost jádra je pozastavena do doby, než klient dokončí předchozí operaci. Kompilátor programu pro procesní jednotku by měl zajistit, aby k těmto situacím docházelo co nejméně.

Hlavní řadič je určen k řízení autonomní činnosti proudových klientů. Stará se především o načítání dat z obou proudů a přepínání banků v některých pamětech. Přepínání banků

je vyvoláno programem pomocí operace hlavního řadiče cmd_next_pckt. Hlavní řadič dá pokyn zúčastněným jednotkám, aby uvolnily aktuálně zpracovávaný paket z paměti a přeply aktivní banky, aby mohlo začít zpracování dalšího paketu. Zároveň dá pokyn Řadiči výstupního proudu, aby zakončil aktuální vysílaný paket. Po té pověří MSC zíkáním nových metadat. Na jejich základě dá příkaz PSC, aby z externí paměti začal načítat data nového paketu. Zajistí, aby se prvních 16 bajtů těchto dat uložilo do Paměti paketových parametrů. Následuje popis činnosti Hlavního řadiče v bodech.

1. Čeká na uvolnění aktivního banku (např. operací cmd_next_pckt).
2. Dá příkaz MSC, aby získal nová metadata.
3. Dá příkaz PSC, aby začal nahrávat do aktivního banku nový paket.
4. Jakmile jsou na výstupu PSC platné Paketové parametry, zapíše je Hlavní řadič do Paměti paketových parametrů.
5. Dá příkaz EPC, aby zajistil nahrání Editačních parametrů.
6. Čeká, až EPC dokončí práci.
7. Žádá-li EPC provedení zpětného zápisu, přikáže Hlavní řadič PSC, aby ho provedl.
8. Pokračuje bodem 1.



Obrázek 4.14: Architektura Hlavního řadiče

Schéma Hlavního řadiče je na obrázku 4.14 a je rozděleno do 4 hlavních bloků. První blok obsahuje čítač cnt_loaded_elements, jehož hodnota se zvýší s každým novým paketem (signál new_element) a sníží s každou operací cmd_next_pckt (signál NEXT_PACKET). Výstupem tohoto bloku jsou signály bank_available a RF_RDY_MNC. Signál bank_available je určen

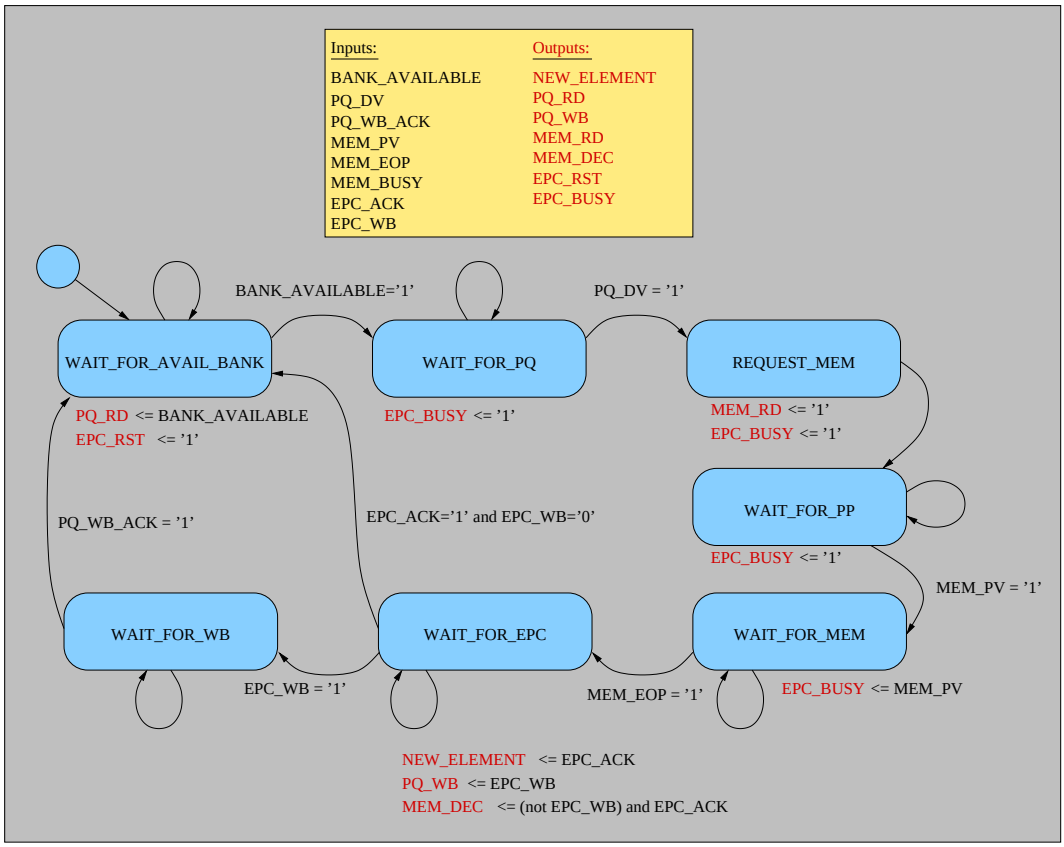
pro automat MAIN_FSM (viz dále) a signalizuje, že je alespoň jeden volný bank (lze načíst další paket). Signál RF_MNC_RDY, je určen pro proudové registrové pole a je nastaven, je-li v aktivním banku nějaký paket. Pokusí-li se nějaká komponenta číst z registrů vstupního proudu, bude proudové registrové pole nepřipraveno a dotyčná komponenta bude čekat. Signál RF_MNC_RDY je určen také pro PSC. Pokud procesorové jádro požádá o provedení některé operace vstupního proudu a signál RF_MNC_RDY nebude nastaven, bude PSC čekat.

Druhý blok je tvořen automatem MAIN_FSM. Jeho úkolem je řídit proudové klienty MSC a PSC a zajistit tak přísun paketů do volných banků. Jeho detailní popis následuje níže.

Třetí blok obsahuje registr reg_mem_addr a čítač cnt_pp_addr. Registr je napojen na datový výstup MSC a slouží k uchovávní adresy paketu v dynamické paměti. Povolovací vstup čítače je na pojem na signál MEM_PV, kterým PSC oznamuje, že jsou na jeho datovém výstupu platné Paketové paramtry. Samotný čítač slouží ke generování adresy do Paměti paketových parametrů.

Poslední blok je tvořen dvěma registry reg_epc_wb a reg_epc_ack a uchovávají poslední stav signálů z řadiče EPC. Pulsem na portu EPC_ACK oznamuje že dokončil svou práci. Pokud je zároveň generován puls i na portu EPC_WB, žádá tím řadič EPC o provedení zpětného zápisu. Výstupy obou registrů jsou přivedeny na vstup automatu MAIN_FSM.

Následuje popis automatu MAIN_FSM. Jeho stavový diagram a seznam vstupních a výstupních signálů je na obrázku 4.15.



Obrázek 4.15: Stavový diagram automatu MAIN_FSM

Po resetu se automat nachází ve stavu *WAIT_FOR_AVAIL_BANK* a čeká až bude nastaven signál bank_available. V tento okamžik zažádá Řadič proudu metadat o získání metadat z Prioritních front. Zároveň resetuje Řadič editačních parametrů a přejde do stavu *WAIT_FOR_PQ*.

V tomto stavu čeká, až budou připravena nová metadata. Řadiči EPC je nastaven signál BUSY, který značí, že druhý port Datové paměti je právě využíván řadičem MNC. Jakmile jsou metada připravena (nastaví se signál PQ_DV), přejde automat do stavu *REQUEST_MEM*.

Ve stavu *REQUEST_MEM* zažádá automat PSC o zaslaní dat paketu (generováním pulsu na portu MEM_RD) a přejde do následujícího stavu.

Ve stavu *WAIT_FOR_PP* čeká automat, až se na výstupu z PSC objeví paketové parametry. PSC tento stav indikuje na portu MEM_PV. Nastavení tohoto portu povolí přechod do stavu *WAIT_FOR_MEM*.

V tomto stavu probíhá příjem dat z PSC. Jakmile je dokončen příjem paketových parametrů (hodnota na port MEM_PV se změní do nuly), je nulován signál EPC_BUSY a Řadič editačních parametrů může začít s nahráváním EP do Paměti editačních parametrů. Konec paketu je indikován PSC nastavením portu MEM_EOP a automat přejde do následujícího stavu.

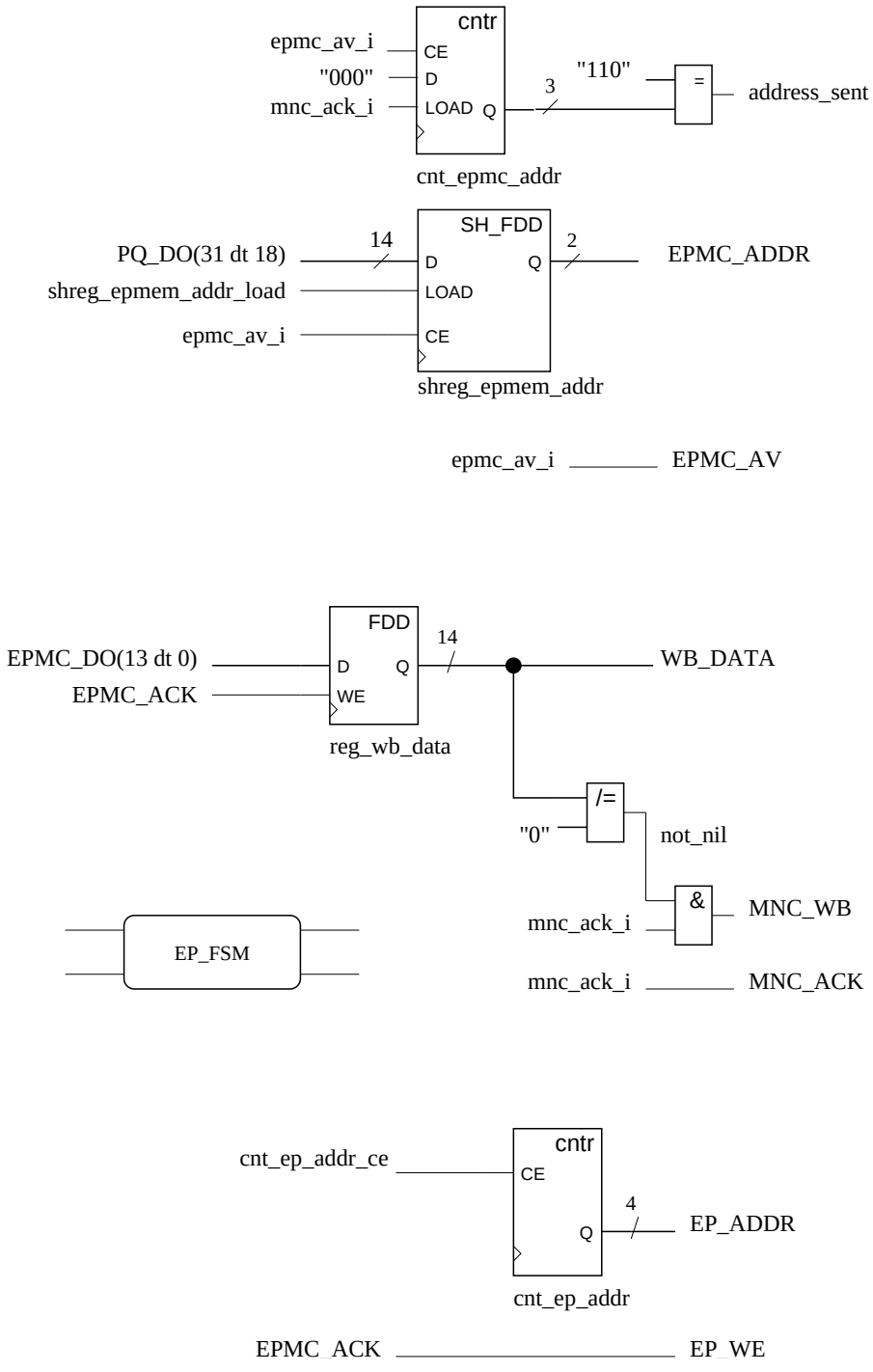
Ve stavu *WAIT_FOR_EPC* čeká automat, až Řadič editačních parametrů skončí svoji práci. To je indikováno pulsem na portu EPC_ACK, který vyvolá nastavení signálu new_element na jeden takt. Žádá-li EPC o provedení zpětného zápisu, generuje zároveň s pulsem na portu EPC_ACK puls i na portu EPC_WB. Na základě toho potom automat buď zažádá, nebo nezažádá PSC o snížení počtu odkazů na paket. Byla-li provedena žádost o zpětný zápis, přejde automat do stavu *WAIT_FOR_WB*. V opačném případě se vrátí do výchozího stavu *WAIT_FOR_AVAIL_BANK*.

Ve stavu *WAIT_FOR_WB* automat počká, až Prioritní fronty potvrdí provedení zpětného zápisu a přejde do stavu *WAIT_FOR_AVAIL_BANK*.

Řadič editačních parametrů je řízen Hlavním řadičem a zajišťuje autonomní načítání Editačních parametrů zprostředkovaných OMC do neaktivního banku Paměti editačních parametrů. Adresu editačních parametrů získá z metadat a následně ji s pomocí posuvného registru „nasouvá“ do EPMC. Ten posléze vrátí odpovídající Editační parametry. Na základě jejich analýzy může Řadič editačních parametrů zažádat o provední zpětného zápisu. Následuje popis činnosti Řadiče editačních parametrů v bodech.

1. Čeká na platná metadata.
2. Zapíše adresu Editačních parametrů do EPMC.
3. Čeká na pokyn Hlavního řadiče.
4. Nastaví žádost o čtení Editačních parametrů a čeká, až je EPMC vyčte.
5. Uloží Editační parametry do Paměti Editačních parametrů.
6. Přejde do bodu 1.

Architektura řadiče je znázorněna na obrázku 4.16. Schéma EPC je rozděleno do 4 bloků. Nejdůležitějším blokem je automat EP_FSM, který řídí ostatní bloky a obstarává komunikační protokol s EPMC. Jeho detailní popis následuje později.



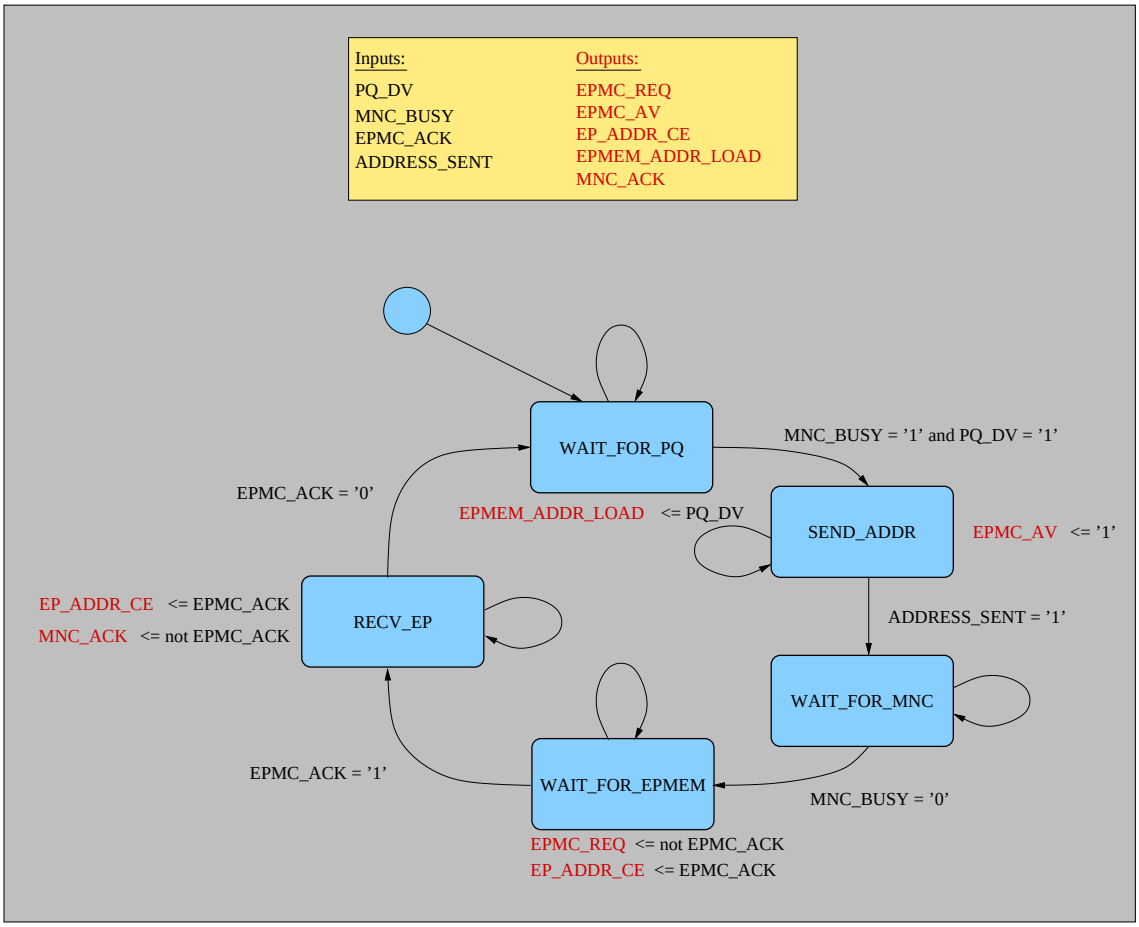
Obrázek 4.16: Architektura Řadiče editačních parametrů

Druhý blok je určen ke generování adresy pro EPMC a obsahuje čítač cnt_epmc_addr a posuvný registr shreg_epmem_addr. Posuvný registr se na pokyn automatu (signál shreg_epmem_addr_load) inicializuje na adresu požadovaného bloku Editačních paratemtrů. Tato adresa je získána z metadat zprostředkovaných Prioritními frontami (port PQ_DO). Signál epmc_av_i je nastavován automatem a signalizuje platnost adresy na portu EPMC_ADDR. Zároveň povoluje inkrementaci čítače a posouvá registr shreg_epmem_addr. Na výstupu čítače je komparátor, který nastaví signál address_sent v okamžiku odesllání poslední části adresy.

Třetí blok obsahuje čítač cnt_ep_addr, který slouží ke generování adresy do Paměti editačních parametrů. Zároveň generuje signál povolující zápis do této paměti.

Poslední blok obsahuje logiku, která zjišťuje, zda za přijatým blokem Editačních parametrů nenásleduje adresa na další blok. Na základě těchto dat generuje žádost o zpětný zápis. K tomu využívá registru reg_wb_data, ve kterém se uchovává adresa následujícího bloku (vyskytuje se jako poslední slovo Editačních parametrů). Je-li nenulová (komparátor not_nil), pak se generuje žádost o zpětný zápis.

Následuje popis automatu EP_FSM. Jeho stavový diagram a seznam vstupů a výstupů je na obrázku 4.17.



Obrázek 4.17: Stavový diagram automatu EP_FSM

Po resetu se automat nachází ve stavu *WAIT_FOR_PQ*, kde čeká na metadata z Prioritních front. Jejich platnost je signalizována na portu PQ_DV. Nastavením tohoto portu a portu MNC_BUSY přejde automat do následujícího stavu.

Ve stavu *SEND_ADDR* dá automat pokyn k odeslání adresy řadiči EPMC signálem EPMC_AV a čeká na signál ADDRESS_SENT. Na jeho základě přejde do stavu *WAIT_FOR_MNC*.

V tomto stavu čeká, až řadič MNC dokončí příjem Paketových parametrů. To je oznámeno nulováním portu MNC_BUSY. Pak automat přejde do následujícího stavu.

Ve stavu *WAIT_FOR_EPMEM* zažádá automat řadič EPMC o zaslání Editačních parametrů (port EPMC_REQ) a čeká na odpověď. Ta je signalizována na portu EPMC_ACK.

Ve stavu *RECV_EP* probíhá příjem Editačních parametrů. Automat povolí generování adresy do Paměti editačních parametrů (signál EP_ADDR_CE). Po přijetí posledního bajtu oznámí automat Hlavnímu řadiči, že je hotov (signál MNC_ACK) a přechází zpět do výchozího stavu *WAIT_FOR_PQ*.

4.4.5 Řadič výstupního proudu (Output Stream Controller, OSC)

Řadič výstupního proudu slouží ke generování výstupního proudu. Je řízen některými operacemi vstupního proudu, operací cmd_next_pckt Hlavního řadiče a také procesorovým jádrem skrze 16-bitový registr OSD proudového registrového pole. Hodnota zapsaná do tohoto registru se odešle do výstupního proudu. Tato operace má dva režimy přepínané příkazem dc_os_cmd:

Režim 16b: Do příkazové paměti na adresu dc_os_cmd se nezapisovalo. Odešle se celých 16 bitů zapsaných do registru OSD.

Režim 8b: Na adresu dc_os_cmd byla zapsána libovolná hodnota. Z přístích 16 bitů zapsaných do registru OSD se odešle pouze dolních 8 bitů - po té dojde automaticky k návratu do režimu 16b.

Řadič výstupního proudu odesílá data pomocí tzv. *Command protokolu*, který byl vyvinut v rámci projektu Liberouter. Jeho hlavním cílem je přenášet paketová data uvnitř FPGA nezávisle na jejich šířce a přenášet s nimi zároveň i kontrolní data. Struktura kontrolních dat je závislá na jednotlivých komponentách a může být během přenosu měněna. Především ale znamená jednotný přenosový protokol pro všechny jednotky, což usnadňuje spojování jednotlivých komponent.

Data jsou v Command protocolu uspořádána do pruhů, kde každý pruh nese 8 bitů dat rozšířených o jeden kontrolní bit udávající jejich význam. Je-li kontrolní bit nulový, pak se jedná o data, v opačném případě o řídicí slovo. Řídicí slova mohou např. vyznačovat začátek paketu, konec paketu atd. Následující seznam shrnuje nejčastěji používaná řídicí slova.

Idle – Žádná data, představuje mezeru v datech

SOP – Začátek paketu (Start Of Packet)

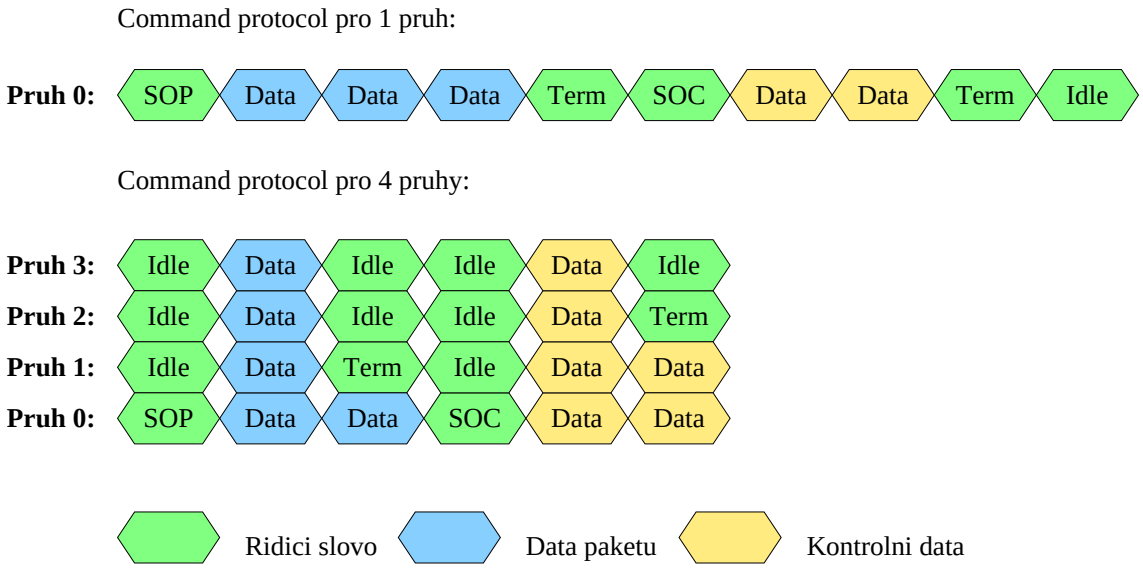
SOC – Začátek kontrolních dat (Start Of Control section)

Term – Ukončující řídicí slovo

Pravidla a omezení Command protokolu:

- Každý paket začína řídicím slovem SOP a končí řídicím slovem Term.
- Každý paket může volitelně obsahovat kontrolní data, která musí obdobně začínat řídicím slovem SOC a končit řídicím slovem Term.
- Každé řídicí slovo s výjimkou Term a Idle se musí vyskytovat v prvním pruhu dat. Ostatní pruhy (jsou-li nějaké) musí být vyplněny řídicími slovy Idle.
- Řídicí slovo Term se může vyskytovat na libovolné pozici v datech a určuje tak jejich zarovnání. Ostaní pruhy následující za Termem musí být vyplněny řídicími slovy Idle.
- Řídicí slovo Idle se nesmí vyskytovat uprostřed dat, např uspořádání „Data-Data-Idle-Data“ pro 32-bitová data není dovoleno.

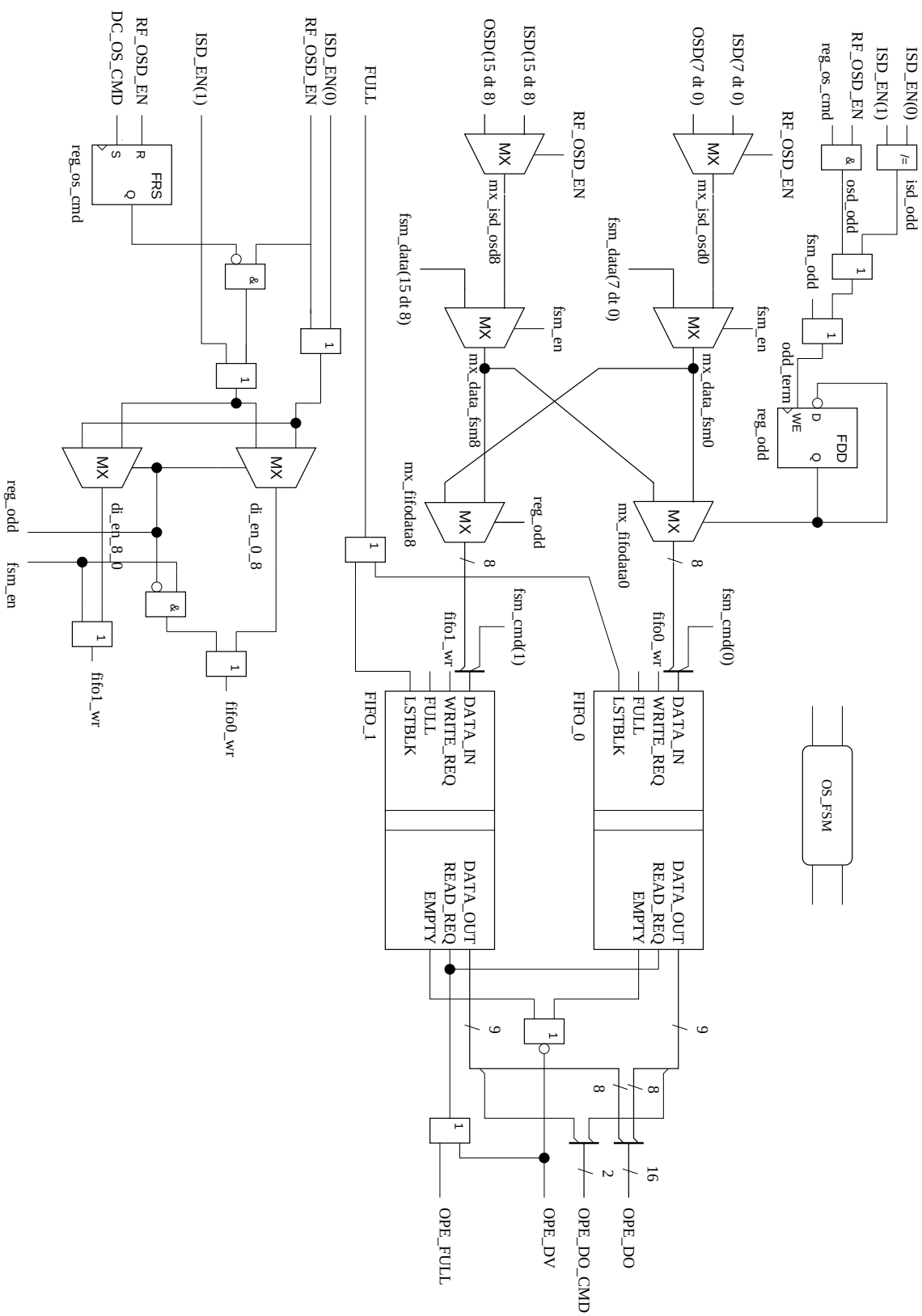
Na obrázku 4.18 jsou znázorněny dva příklady struktury Command protocolu pro jeden pruh a 4 pruhy.



Obrázek 4.18: Příklad Command protocolu

Následuje popis činnosti Řadiče výstupního proudu v bodech.

1. Odešle SOP.
2. Na výstup odesílá data z Aplikačního procesoru, nebo Vstupního bufferu a čeká na operaci cmd_next_pckt, příp. na přerušení.
3. Zakončí data paketu odesláním TERM.
4. Vygeneruje a odešle kontrolní data.
5. Přejde do bodu 1.



Obrázek 4.19: Architektura Řadiče výstupního proudu

Následuje popis architektury Řadiče výstupního proudu znázorněné na obrázku 4.19. Jeho jádrem je automat OS_FSM a dvě FIFA, do kterých jsou zapisována výstupní data. Automat má na starosti zaobalení odesílaných dat do Command protokolu a generování kontrolních dat. Jelikož jsou data do proudu odesílána průběžně, nelze předem určit, zda nedojde během zpracování paketu v Řadiči výstupního proudu k přerušení (to je signalizováno portem IRQ). Proto nesou kontrolní data údaj o tom, zda byl paket zpracován korektně. Tím je informován OBUF, zda takovýto paket zpracovat, nebo ne.

Fifa FIFO_0 a FIFO_1 slouží jako buffer pro odesílaná data. Do FIFO_0 se zapisují data, která budou odeslána jako dolních osm bitů a do FIFO_1 se zapisují data, která budou odeslána jako horních osm bitů. Data mohou do obou fif přicházet buď z Proudového registrového pole z registrů ISD a OSD, nebo z automatu OS_FSM (signál fsm_data). Platnost dat na portu ISD je indikována dvoubitovým signálem ISD_EN. Ten může nabývat hodnot

00 – data nejsou platná

01 – platných dolních osm bitů

11 – platných všech šestnáct bitů

Platnost dat na portu OSD je indikována portem RF_OSD_EN, generovaným proudovým registrovým polem, který signalizuje zápis do registru OSD. Stav operace dc_os_cmd je zachycen v registru reg_os_cmd, který se nastaví během zápisu na adresu této operace (žádost o režim 8b), což je signalizováno na portu DC_OS_CMD, generovaném Datovou pamětí. Po zápisu do registru OSD je hodnota registru nulována, což odpovídá režimu 16b.

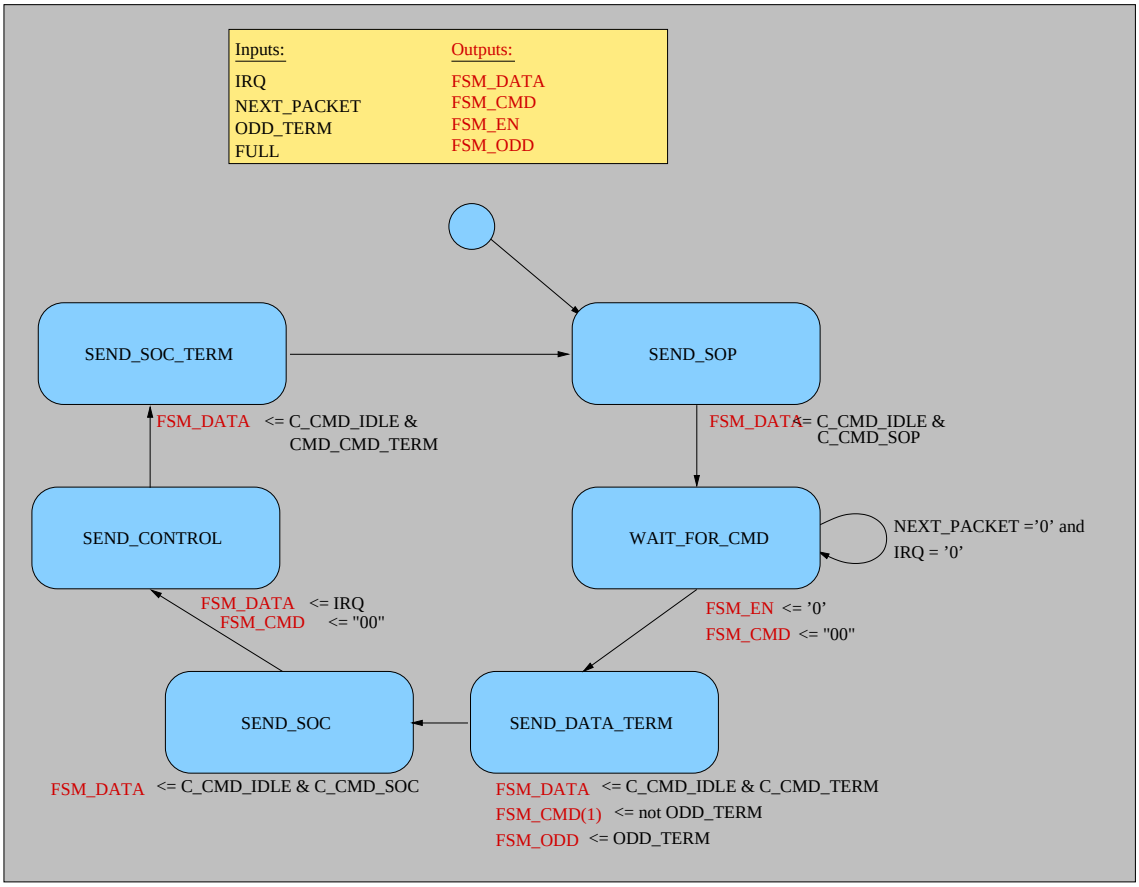
Data na signálu fsm_data jsou generována automatem a jejich platnost je indikována signálem fsm_en. Tato data jsou tvořena řídicími slovy Command protokolu a obsahují také kontrolní data.

Jelikož do Řadiče výstupního proudu mohou přicházet data šestnáctibitová i osmibitová, je nutné je přerovnat na šířku výstupního portu, která je 16 bitů. To je zajištěno tak, že když přijde pouze osm bitů, je změněna hodnota registru reg_odd, který řídí mutliplexory mx_fifodata0 a mx_fifodata8. To zajistí přehození vstupů do obou fif. Zároveň je však potřeba správně generovat hodnoty signálů fifo0_wr a fifo1_wr, které povolují zápis do FIFO_0, resp. FIFO_1.

Přítomnost osmibitových dat je indikována signály isd_odd (port ISD), osd_odd (port OSD) a fsm_odd (signál fsm_data).

Data jsou na vstupu do fifa sdružena s kontrolním bitem, který je na výstupu přiveden na portu OPE_DO_CMD. Datové bity jsou přivedeny na port OPE_DO. Platnost dat na tomto portu je signalizována portem OPE_DV, který je nastaven pouze v případě, kdy jsou obě fifa neprázdná. Port OPE_FULL signalizuje stav výstupního proudu. Je-li plný, je další odesílání dat pozastaveno.

Následuje popis automatu OS_FSM. Jeho stavový diagram je na obrázku 4.20. Po resetu se automat nachází ve stavu *SEND_SOP*, kdy je do výstupního proudu odesláno řídicí slovo SOP doplněné o Idle dle pravidel Command protokolu. V dalším taktu přejde automat do stavu *WAIT_FOR_CMD*.



Obrázek 4.20: Stavový diagram automatu OS_FSM

V tomto stavu jsou odesílána data paketu z portů ISD a OSD (signál fsm_en je v nule a kontrolní bity také). Automat čeká v tomto stavu na vyvolání operace NEXT_PACKET, nebo na přerušení IRQ, aby mohl přejít do následujícího stavu.

Ve stavu *SEND_DATA_TERM* provede automat zakončení dat paketu řídicím slovem Term. To musí být umístěno hned za posledním bajtem dat. Výstup registru reg_odd je přiveden na port ODD_TERM automatu. Na základě něj generuje kontrolní bity (port FSM_CMD). Je-li ODD_TERM rovno nule, pak jsou data zarovnána na 16 bitů a budou zakončena sekvencí dvou řídicích slov Term a Idle (FSM_CMD bude rovno „11“). V opačném případě bude data zakončovat je jedno řídicí slovo Term, proto musí být nastaven signál FSM_ODD a FSM_CMD bude rovno „10“. Tím je zajištěno zakončení dat a jejich zarovnání na 16 bitů. V následujícím taktu přejde automat do stavu *SEND_SOC*.

Ve stavu *SEND_SOC* odešle automat řídicí slovo SOC a přejde do následujícího stavu.

Ve stavu *SEND_CONTROL* odešle automat jedno slovo kontrolních dat. Jeho hodnota je závislá na stavu signálu IRQ. V následujícím taktu přejde do stavu *SEND_SOC_TERM*.

V tomto stavu zakončí automat kontrolní data sekvencí Term a Idle a vrátí se zpět do výchozího stavu *SEND_SOP*.

4.4.6 Koprocesorová jednotka (Co-processor Unit)

Koprocesorová jednotka obsahuje pomocné koprocesory Aplikačního procesoru. V současné aplikaci je využit pouze jeden koprocesor - Checksum Computer (CSC).

CSC zajišťuje výpočet 16b kontrolního součtu nad vybranými daty. Průběžný výsledek se ukládá do registru checksum_result. Program ovládá CSC prostřednictvím registru checksum_ctrl. Následuje seznam operací pro CSC:

START – vynuluje obsah registru checksum_result a spustí CSC

STOP – činnost CSC se pozastaví - obsah checksum_result se nebude měnit

CONTINUE – spustí CSC, přičemž obsah checksum_result zůstane zachován

Je-li CSC spuštěno, pak počítá kontrolní součet ze všech slov uložených do OSD (z 16 nebo 8 bitů, podle dc_os_cmd) nebo CSC_data registru a průběžný výsledek ukládá do registru checksum_result.

Kapitola 5

Závěr

Cílem této práce bylo navrhnout, implementovat a v praxi otestovat Procesní jednotku pro analýzu a editaci síťového provozu v FPGA. Vzhledem k charakteru zpracovávaných dat vychází návrh Procesní jednotky z konceptu proudových procesorů [2, 3]. Návrh byl proveden s ohledem na vysokou míru paralelismu zpracování vstupního proudu dat, kdy lze souběžně zpracovávat několik nezávislých elementů proudu. Architektura umožňuje jednoduše přídávat a odebírat proudové klienty tak, aby bylo dosaženo co nejvyšší efektivity zpracování. Jednotliví proudoví klienti mohou plnit různé úkoly a jsou řízeni buď autonomně, nebo programem Procesní jednotky.

Komponenty byla implementována a testována v programovatelném hradlovém poli VIRTEX-II Pro. Pro tento účel byly nastudovány materiály pojednávající o architektuře, vlastnostech [17] a použití komponent dostupných ve zmíněném čipu [18].

Dále bylo zapotřebí seznámit se s cílovým hardwarem, zahrnujícím výše zmíněné hradlové pole a další potřebné prostředky (především externí paměti). Za tímto účelem byla nastudována schémata a specifikace karet rodiny COMBO [21].

Nezbytné bylo také naučit se jazyku VHDL [20], ve kterém byla implementace provedena. Bylo zapotřebí nastudovat především pravidla pro tvorbu VHDL kódu, který by byl snadno a efektivně syntetizovatelný pro použití v cílovém FPGA [15]. V tomto případě jsem čerpal také z rad zkušených kolegů z projektu Liberouter.

Komponenta byla testována na taktovací frekvenci 100MHz, při které byla schopna zpracovat síťový provoz o rychlosti v jednotkách gigabitů za sekundu. Konrétní propustnost je závislá na složitosti prováděného programu a rozložení síťového provozu. Nejnižší propustnosti dosahuje komponenta na krátkých paketech. To je dáno charakterem zpracování, které se zaměřuje především na hlavičky paketů. Další snížení propustnosti vyvolá velké množství složitě zpracovávaných paketů (např. odeslání tunelem), kdy musí procesor provést na jednom paketu velké množství operací.

Při testech, kdy byla jednotka úmyslně zahlcena velkým množstvím krátkých paketů délky 64 bajtů, přesáhla propustnost Procesní jednotky 1Gb/s. Při stejném testu, ale s pakety délky 1500 bajtů, dosáhla propustnost těsně pod hranici 3Gb/s. Vzhledem k snadné škálovatelnosti a replikovatelnosti navržené architektury je možné propustnost dále zvýšit replikováním jednotlivých Editačních jednotek, mezi něž se datový tok rozdělí.

Literatura

[1] Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specificationss, norma IEEE 802.3, IEEE Computer Society, 2002

[2] Kapasi, U., J., Rixner, S., Dally, W., J., Khailany, B., Ahn, J., H., Mattson, P., Owens, J., D.: Programmable Stream Processors, Published by the IEEE Computer Society, Srpen 2003

[3] Khailany, B., Dally, W., J., Rixner, S., Kapasi, U., J., Owens, J., D., Towles, B.: Exploring the VLSI Scalability of Stream Processors, Stanford University a Rice University

[4] Zimmermann, H.: OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection, IEEE Transactions on Communications, vol. 28, no. 4, 1980

[5] International Standart 7498-1, ISO/IEC, druhé vydání, 1994

[6] Internet Protocol, norma RFC791, Information Sciences Institute, University of Southern California, 1981

[7] The Liberouter Webpage, 2006. Dokument dostupný na http://www.liberouter.org

[8] Mikušek, P.: Generic Nanoprocessor (GENA), 2006. Dokument dostupný na http://www.liberouter.org/vhdl_design/generated/HEAD_gena_info.php.

[9] Mikušek, P.: Návrh a implementace procesní jednotky pro analýzu vstupních paketů, FIT VUT Brno, 2005.

[10] Kořenek, J.: Návrh a implementace vyhledávacího procesoru, FIT VUT Brno, 2003.

[11] Martínek, T.: Návrh a implementace editoru paketů, FIT VUT Brno, 2003.

[12] Pazdera, J.: Input buffer for GMII interface, Dokument dostupný na http://www.liberouter.org/vhdl_design/generated/HEAD_ibuf_info.php.

[13] Pazdera, J.: Output buffer for GMII interface, Dokument dostupný na http://www.liberouter.org/vhdl_design/generated/HEAD_obuf_info.php.

[14] Pazdera, J.: Implementace systému prioritních front v FPGA, FIT VUT Brno, 2005.

[15] Mastretti, M., Busi, M., L., Sarvello R., Sturlesi. M., Tomasello S.: VHDL quality: synthesizability, complexity and efficiency evaluation, 7th IEEE International Workshop on Rapid System Prototyping, 1996

[16] Xilinx. DS031 Virtex-II 1.5V Field-Programable-Gate-Arrays, září 2002. Dokument dostupný na http://direct.xilinx.com/bvdocs/publications/ds031.pdf.

[17] Xilinx. Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet, verze (4.1), listopad 2004. Dokument dostupný na http://direct.xilinx.com/bvdocs/publications/ds083.pdf.

[18] Xilinx. Virtex-II Pro and Virtex-II Pro X FPGA User Guide, verze 3.0, srpen 2004. Dokument dostupný na http://www.xilinx.com/bvdocs/userguides/ug012.pdf.

[19] Samsung. 512Kx36/x32 & 1Mx18-Bit Synchronous Burst SRAM, květen 2002. Dokument dostupný na http://www.datasheetcatalog.com/catalog/p118480.shtml.

[20] Ashenden, P., J: The VHDL Cookbook, First Edition, Červenec 1990

[21] Novotný, J., Bardas, R.: Schematics and PCB of COMBO6X, CESNET, z. s. p. o.

Dodatek A

Příklady programů

A.1 Program 1

Tento program změní linkové adresy a odešle rámec jako Ethernet broadcast rámec.

```
#define TMP R00 ; temporary data register

init:
MOVI %R00, 0xff

loop:
MOV %R00, %ISD ; read old destination MAC address - word 0.
MOV %R00, %ISD ; read old destination MAC address - word 1.
MOV %R00, %ISD ; read old destination MAC address - word 2.
MOV %R00, %ISD ; read old source MAC address - word 0.
MOV %R00, %ISD ; read old source MAC address - word 1.
MOV %R00, %ISD ; read old source MAC address - word 2.
MOVI %OSD, 0xffff ; send packet as ethernet broadcast
MOVI %OSD, 0xffff ; send packet as ethernet broadcast
MOVI %OSD, 0xffff ; send packet as ethernet broadcast
MOVI %OSD, 0x0011 ; send word 0. of my MAC address
MOVI %OSD, 0x1700 ; send word 1. of my MAC address
MOVI %OSD, 0x0001 ; send word 2. of my MAC address
MOV %OSD, %ISD ; copy ethernet type to output
SW %R00, CMD_SNDREST_ISRE ; send rest of packet without editing
NOP
SW %R00, CMD_NEXT_PCKT ; request a new packet to input stream
JMP loop
```

A.2 Program 2

Tento program vygeneruje odpověď ping reply na příchozí paket typu ping request.

```
loop:
    ; swap MAC addresses
    MOV     %R00, %ISD              ; store destination MAC address
    MOV     %R01, %ISD
    MOV     %R02, %ISD
    MOVI    %R03, 6

    NOP
    SW      %R03, CMD_SNDNFWRD_ISRE ; transmit the source MAC address
                                    ; instead of the destination one

    MOV     %OSD, %R00              ; transmit the destination MAC address
    MOV     %OSD, %R01              ; instead of the source one
    MOV     %OSD, %R02
    MOVI    %R00, 14
    SW      %R00, CMD_SNDNFWRD_ISRE ; transmit until src IP address

    ; swap IP addresses
    MOVI    %R03, 4
    NOP
    MOV     %R00, %ISD              ; store src IP address
    MOV     %R01, %ISD
    NOP
    SW      %R03, CMD_SNDNFWRD_ISRE ; transmit dst IP address as src
    MOV     %OSD, %R00              ; transmit the src IP address as dst
    MOV     %OSD, %R01

    ; change ICMP type from 'ping request' to 'ping reply'
    MOV     %R00, %ISD              ; read type 8 - request
    MOVI    %OSD, 0                 ; send type 0 - reply

    ; transmit the rest of the frame
    SW      %R00, CMD_SNDREST_ISRE
    NOP
    ; gimme next packet
    SW      %R00, CMD_NEXT_PCKT
    NOP

    ; repeat
    JMP     loop
```

Dodatek B

Seznam použitých zkratek

B.1 Komponenty směrovače

HFE – Header Field Extractor

IBUF – Input Buffer

LUP – Look Up Processor

PQ – Priority Queues

OPE – Output Packet Editor

REP – Replicator

UH FIFO – Unified Header FIFO

B.2 Komponenty Procesní jednotky

AP – Application Processor

DC – Data Cache

EPC – Edit Parameters Controller

EPMC – Edit Parameters Memory Controller

EU – Editation Unit

GENA – Generic Nanoprocessor

IC – Instruction Cache

IStB – Input Stream Buffer

MNC – Main Controller

MSC – Metadata Stream Controller

OMC – OPE Memory Controller

OSC – Output Stream Controller

PSC – Payload Stream Controller

B.3 Registry proudového registrového pole

ISB – Input Stream - Begin

ISD – Input Stream - Data

ISE – Input Stream - End

ISR – Input Stream - Read

OSD – Output Stream - Data

B.4 Ostatní zkratky

EP – Edit Parameters

PP – Packet Parameters

WB – Write-Back