VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2019 Bc. František Németh

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

PRIORITNÍ PAKETOVÉ FRONTY V FPGA
PRIORITY PACKET QUEUES IN FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. František Németh**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. David Smékal**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. František Németh ***ID:*** 164766
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Prioritní paketové fronty v FPGA

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je navrhnout, implementovat a ověřit způsob řízení front pro nasazení ve vysokorychlostních paketových sítích. Seznamte se s programovacím jazykem VHDL, FPGA kartami NFB, vývojovým frameworkem NDK a platformou Xilinx Vivado. Seznamte se s problematikou kvality služeb (QoS) a algoritmy řízení prioritních paketových front. Zvolte algoritmus vhodný pro obvodovou realizaci v FPGA. Implementujte zvolený algoritmus v jazyce VHDL. Ověřte funkčnost implementace v hardware a vyhodnoťte dosažené výsledky z pohledu rychlosti a obvodové náročnosti.

DOPORUČENÁ LITERATURA:

[1] WOLF, Wayne. FPGA based system design. New Jersey: Prentice-Hall, 2004, 530 s. ISBN 0-13-142461-0.

[2] PINKER, Jiří, Martin POUPA. Číslicové systémy a jazyk VHDL. 1. vyd. Praha: BEN - technická literatura, 2006, 349 s. ISBN 80-7300-198-5.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: Ing. David Smékal
Konzultant: Ing. Viktor Puš Ph.D. (Netcope Technologies, a.s.)

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Diplomová práca sa zaoberá problematikou riadenia paketových front a následným návrhom v jazyku VHDL pre možnosť nasadenia vo vysokorýchlostných paketových sietiach. V teoretickej časti práce sú podrobnejšie rozobraté fungavania mechanizmov zabezpečenia kvality služby. Teoretická časť ešte zahrňuje v sebe stručný popis programovacieho jazyku VHDL, FPGA čipov a vývojového prostredia Netcope Development Kit. Výstupnej praktickej časti je obmedzovač priepustnosti viacerých front založený na princípe Token Bucket. Výsledným testovaním komponentov sa overovala funkčnosť na simulačnej, syntéznej a aj reálnej úrovni. Všetky dosiahnuté výsledky a možnosti návrhu sú shrnuté v posledných troch kapitolách.

KĽÚČOVÉ SLOVÁ

Kvalita služby, tvarovanie priepustnosti, Tocken Bucket, RED, VHDL, FPGA, Xilinx, Netcope Development Kit, NFB-200G2QL.

ABSTRACT

Master thesis is dealing with issues and problems of packet queue management in high speed packet networks. Design implementation is made in VHDL hardware description language. In theoretical part of thesis are explained different types of mechanism used for providing quality of service in communication networks. Furthermore the brief description o VHDL, FPGA and framework Netcope Development Kit is a piece of theoretical part as well. The outcome of practical part contains a design, limiting packet queues based on Tocken Bucket mechanism. Design verification was made by simulations, synthesis and real implementation on smart NIC NFB-200G2QL. All kind of verificaion results are summerized in last three chapters.

KEYWORDS

Quality of Service, shaping througthput, Tocken Bucket, RED, VHDL, FPGA, XILINX, Netcope Development Kit, NFB-200G2QL.

NÉMETH, František. *Prioritní paketové fronty v FPGA*. Brno, 2019, 65 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. David Smékal

VYHLÁSENIE

Vyhlasujem, že som svoju diplomovú prácu na tému „Prioritní paketové fronty v FPGA“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád bych poděkoval vedoucímu diplomové práce panu Ing. David Smékal a konzultantovi z Netcope Technologies a.s. Ing. Viktor Puš,Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov

Úvod

S modernizáciou komunikačných technológií a postupným nárastom používatelov Internetu sa zvýšili aj výkonnostné požiadavky siete. Princíp best effort (rovnaká priorita pre všetky vstupujúce datové jednotky) sa stala nepostačujúcou pre udrženie kvality prenášaných datových tokov. Zvíšila sa doba zpracovávania, ktorá spôsobila meškanie, alebo strátu dôležitých dátových jednotiek. Ku eliminovaniu tohoto problému sa vyvynula sada technológií a mechanizmov zabezpečujúce kvalitu služby (Quality of Service - QoS).

Táto diplomová práca sa zaoberá s problematikou kvality služby a spôsobom riadenia paketových front. Cieľom diplomovej práce je navrhnúť a implementovať spôsob riadenia front pre nasadenie vo vysokorýchlostných paketových sietiach. Rýchly vývoj a usnadnenie práce so sieťovými kartami osadenými s FPGA čipmy otvára brány pre jejich využitie v tomto smeru sieťových aplikácií. Z dôvodu rýchlosti, výkonnosti a možnosť paralelizácie výpočtov sú FPGA čipy čoraz frekventovanejšie používané. Z týchto dôvodov pre realizovanie praktickej časti sa využije sieťová akceleračná karta NFB-200G2QL, osadená s výkonným FPGA čipom XILINX Virtex UltraScale+ (XCVU7P).

Na začiatku teoretickej časti práce je popísaný všeobecné zabezpečenie kvality služby. Následne sú uvedené jednotlivé mechanizmy zabezpečujúce rôzne úkoly pre udržanie kvality služby. Potom teoretická časť sa rozdelí na podrobnejší popis princípov rôzných typov paketových front kombinované plánovacími algoritmami, na mechanizmy aktívnej správy front a tvarovania dátového toku a na obecný popis jazyka VHDL, FPGA čipov a vývojového prostredia Netcope Development Kit.

V praktickej časti práce je popísaný vzniknutý návrh obmedzovača priepustností viacerých front na základe mechanizmu Token Bucket. Jednotlivé časti návrhu sú popísané v menších sekciách. Po všeobecnom popise sú uvedené dosiahnuté simulačné a syntetizačné výsledky hotového návrhu. Pomocou grafov je uvedená maximálna možná využitosť počtu front v obmedzovači. V posledných dvoch kapitolách sa práca viac zameriava na realné otestovanie vytvoreného obmedzovača na sieťovej karte. Overuje sa funkčnost výstupu pri rôzných konfiguráciach a nastaveniach obmedzovača priepustnosti. Popisujú sa vzniknuté neefektivity a jejich riešenia. Posledná kapitola sa venuje k popisu vytvoreného systému obmedzovača front kombinovaná s mechanizmom aktívnej správi front (Random Early Detection).

1 Kvalita služby

Kvalita služby (ang. Quality of Service) je sada mechanizmov, ktoré sú schopné poskytovať rôzné priority pre rôzne sieťové aplikácie [3]. Funkcie QoS sa primárne využívaju u Real-time multimedialnych aplikáciach ako internetové telefonovanie (VoIP), streamovanie videa či videokonferencií. Hlavnými parametrami kvality služby sú [7] :

- meškanie (delay),
  - potrebná doba k preneseniu paketu,
  - citlivé aplikácie :
    internetové telefonovanie VoIP
    Real-time aplikácie,
- kolísanie meškania (jitter),
  - popisuje aktuálnu hodnotu meškania,
  - dynamicky sa mení na základe vytíženia stroja,
  - citlivé aplikácie :
    VoIP,
    real-time multimediálne prenosy videa - menej citlevé kvôli používania väčšej vyrovnávacej pamäti,
- strátovosť paketov (packet loss),
  - pomer paketov, ktoré sa nedostali k cieľu,
  - strata paketov náhodne neznižuje kvalitu prenosu,
  - strata väčšieho bloku paketov vedie k zníženiu kvality prenosu,
- priepustnosť (throughput),
  - objem dát v bajtoch prenesené za jednotku času.

1.1 Mechanizmy kvality služieb

Zabezpečenie kvality služby (QoS, ang. Quality of Service) je možné rozdeliť na dve základné mechanizmy, IntServ (Integrované služby) a DiffServ (Diferencované služby). V nasledujúcich častiah sa najprv stručne popíše princíp chovania integrovaných služieb. Následne sa práca viac zameriava na differenciované služby a na súvisiace mechanizmy.

1.1.1 Integrované služby

Integrované služby sú známe ako „Hard QoS“ mechanizmy. Pred zahájením prenosu prevedia explicitnú rezerváciu zdrojov siete pre jednotlivé dátové toky, a rezervácia

potrvá až do konca prenosu [7]. K tomu sa využíva pomocný protokol Resource ReserVation Protocol (RSVP). Sice IntServ garantuje prenosové parametry jednotlivých datových tokov, v dnešnej dobe sa už menej využíva tento typ služby. To z toho dôvodu, že

- kladie veľké nároky na aktívne prvky,
- každý používaný prvok musí podporovať RSVP protocol,
- opakovaná rezervácia sieťových prostriedkov zaťažuje sieť.

1.2 Differenciované služby

Differenciované služby sú známe ako „Soft QoS“ mechanizmy. Negarantujú potrebné parametry pre jednotlivé dátové toky. Celý model je vybudovaný na pricípe triedenia [6]. Vstupné datové toky sú rozdelené do tried, pre ktoré sú nadefinované rôzne zachovacie princípy. Aj napriek tomu, že DiffServ nie je schopný garantovať potrebné parametry, je dobre škálovateľný, nekladie vysoké nároky na sieť a na procesor aktívnych prvkov. Je možné využívať s IP protokolom aj s inými protokolmi. V dnešnej dobe je najrozšírenejším mechanizmom pre dosiahnutie kvality služby.

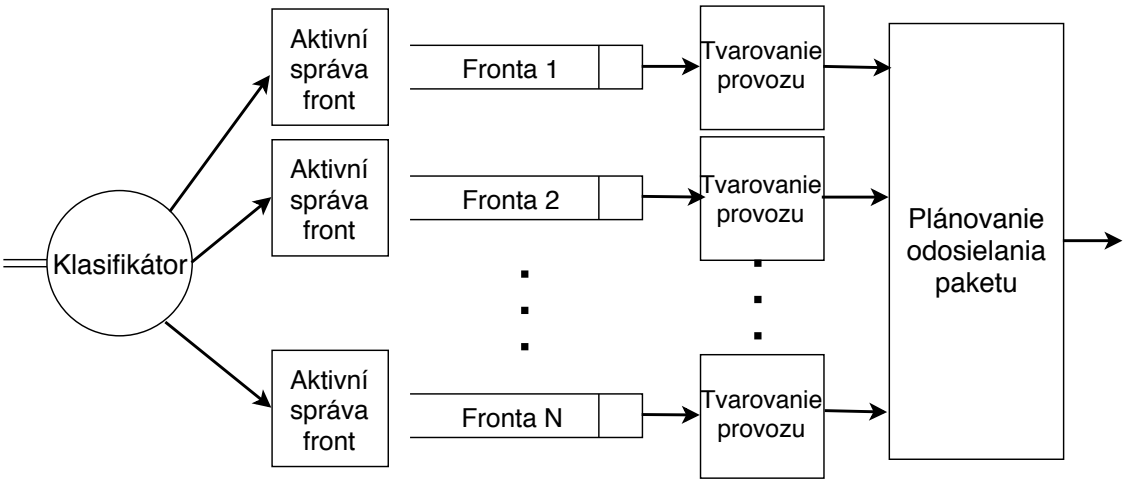
Každá technológia musí zabezpečit dve hlavné úkoly pre zabezpečenie kvality služby [3]

1. Triedenie dátových jednotiek podľa parametru služby,
    - klasifikácia (Clasification),
    - značkovanie (Marking).
2. Zabezpečenie rôzneho zachovania s definovanými triedami dátových tokov na základe,
    - dohladom nad datovým tokom (Meter),
    - aktívnej správy front (Congestion Avoidance),
    - plánovania odosielania paketov (Queuing and Scheduling),
    - tvarovania dátoveho toku (Traffic Shaping).

Na obrázku 1.1 je uvedená štruktúra architektúry modelu QoS. Ku správnemu chovaniu je potrebné aby navrhovaný systém sa skladal z časti, ktoré riešia úkoly popísane vyššie. Teda triedenie a označkovanie vstupujúcich dátových tokov na základe definovanej politiky systému (Klasifikátor). Následný dohľad a udržovanie stavov (Aktívna správa front) vyrovnávacej pamäti (fronta), aby nevznikli nežiadúce javy, ktoré by znížili kvalitu služby. Takýmto javom je napr. zaplnenie sa vyrovnávacej pamäti pri zápise dátových tokov. Výstupné prenosové rýchlosti jednotlivých front sú následne tvarované podľa politiky zabezpečenia kvality služby. Na konci systému je mechanizmus zabezpečujúci plánovanie a odosielanie triedených paketov.

V nasledujúcich kapitolách budú vysvetlené vybrané algoritmy aktívnej správy front (Random Early Detection, Weighted Random Early Detection), tvarovania

datového toku (Token Bucket) a plánovacie algoritmy pre odosielanie paketov (PQ, Weighted Round Robin, Deficit Round Robin).



Obr. 1.1: Obecná štruktúra systému kvality služieb

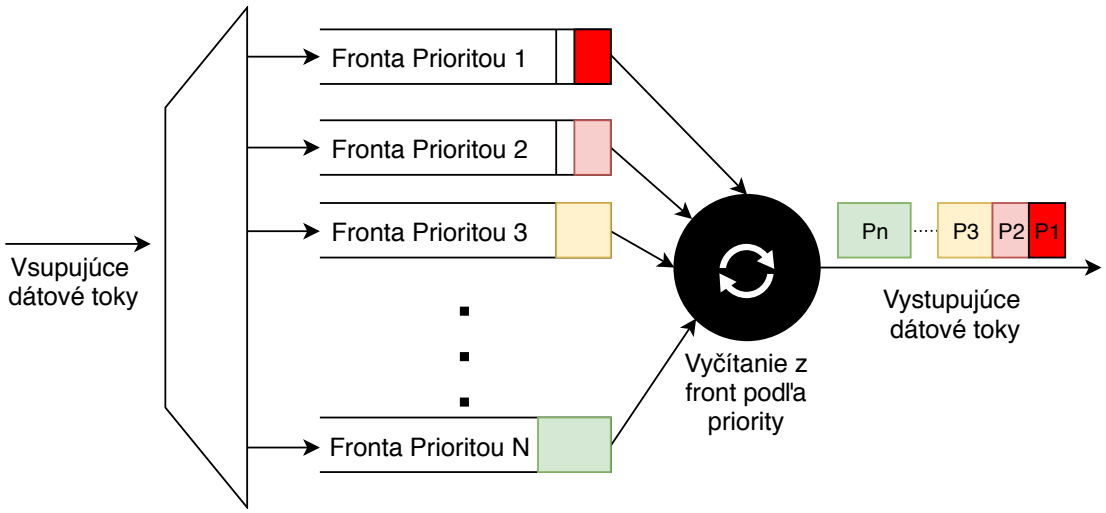
2 Paketové fronty

Pri správnom fungovania modelu QoS, klúčovú postavu hrajú fronty. Jedná sa o dátovú štruktúru typu FIFO (First In First Out). Inými slovami, vyrovnávacia pamäť. Zachováva poradie vstupných paketov na výstupe. Aby k jednotlivým prenosom bolo možné umožniť kvalitu služby, datové toky sa klasifikujú do tried a uložia sa do front tej triedy [6]. Následne je prevedené plánovanie a odosieľanie paketov z jednotlivých front, podľa definovanej konfigurácie. V nasledujúcej časti tejto sekcie budú vysvetlené plánovacie algoritmy:

- Priority Queuing,
- Weighted Round Robin,
- Deficit Round Robin.

2.1 Priority Queuing

Samotný algoritmus uprednostňuje vyčítanie z front s väčšiou prioritou. V prípade, keď fronta s najväčšiou prioritou je prázdna, vyčítanie sa presunie na frontu s menšou prioritou [3]. Princíp algoritmu je znázornený na obrázku 2.1.



Obr. 2.1: Schéma algoritmu Priority Queuing

Z obrázku plynie, že najprv je vyčítaný červený paket z fronty prioritou 1, ktorá má najväčšiu prioritu. Následne, keď fronta P_1 je prázdná vyčíta sa postupne z front P_2, P_3 a P_n.

Pri používania tohto algoritmu, treba dávať pozor aby nevznikli prioritné fronty s konštantným veľkým provozom [3]. V tom prípaďe dáta uložené vo frontách menšou prioritou sa zaplnia a začne sa zahazovanie vstupujúcich paketov danej triedy. Prípadne pozastavenie selekcie na vstupu až do tej doby, kým sa neuvolný dostatočný

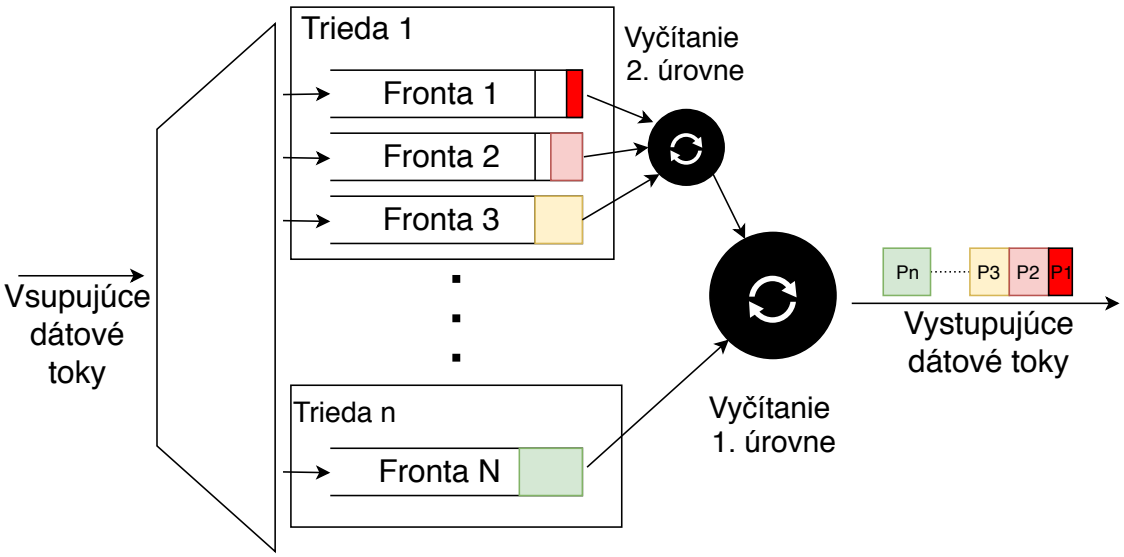
počet miest pre čakajúci paket. Existencia týchto javov je nepriepustné v systéme QoS.

Vo výsledku tento typ algoritmu je realizácia prenosu so striktnou prioritou (SP). Plánovanie a riadenie pomocou tohoto algoritmu sa využíva hlavne u paketov real-time aplikácií s nepravideľným príchodom do front.

2.2 Weighted Round Robin

Weighted Round Robin zapadá do skupiny algoritmov, ktoré zabezpečia váženú obsluhu front. Prichádzajúce pakety sú triedené na základe jejich priority a následne uložené do jednotlivých front patriacich k danej triede [5].

Šírka pásma výstupu je rozdelený medzi frontami na základe váhy jednotlivých tried. Podľa počty front v jednotlivých triedách, môže byť vyčítanie jedno alebo dvojúrovňové. Keď každá definovaná trieda vlastní práve jednu frontu, vyčítanie je jednoúrovňové. Pri existencie minimálne dvoch front v definovanej triede, vyčítací cyklus už je dvojúrovňové [3].
Algoritmus používa na oboch úrovniach cyklické plánovanie Round Robin. S tým rozdieľom, že na prvej úrovni doba jednoho vyčítacieho cyklu je rozdelená medzi jednotlivými triedami podľa jejich váhy, kým na druhej úrovni doba určená pre vyčítanie je ekvivalentná pre všetky fronty. Na nasledujúcom obrázku 2.2 je znázornený princíp algoritmu Weighted Round Robin.

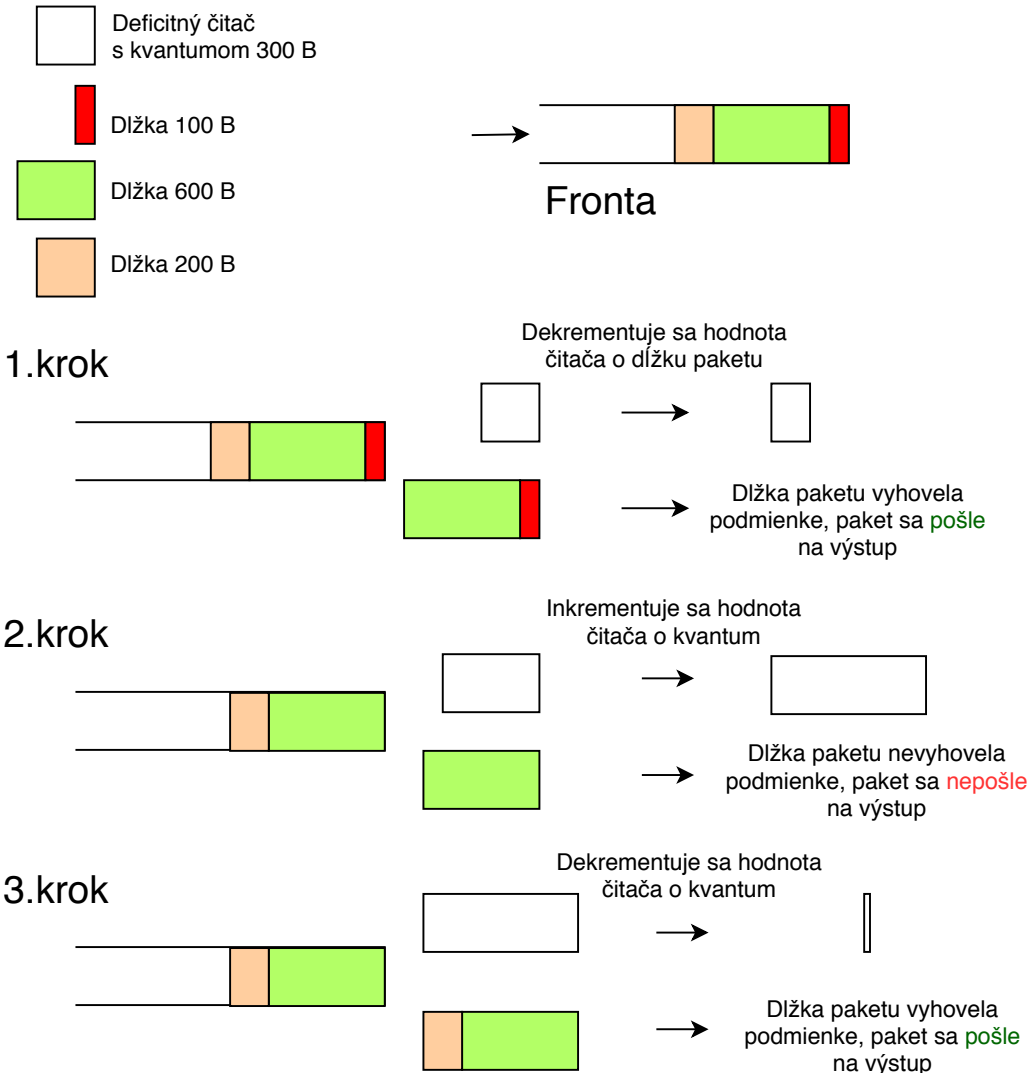


Obr. 2.2: Schéma algoritmu Weighted Round Robin

2.3 Deficit Round Robin

Algoritmus Defecit Round Robin je vylepšená verzia algoritmu Weighted Round Robin, ktorá vyriešila problém s dĺhými paketmi [4]. Spadá do skupiny algoritmov s váženou cyklickou obsluhou.

Pre každu jednu frontu sa používa deficitný čitač, ktorý sa inkrementuje pred každým vyčítaním o predom nastavenú hodnotu. Táto hodnota je tzv. kvantum. Môže byť rôzne definovaná pre jednotlive fronty. Pred každým vyčítaním sa overí či dĺžka paketu je menšia než hodnota deficitného čitača [4]. Keď je menšia vyčíta sa paket z fronty a dekrementuje sa hodnota čitača o dĺžku paketu. Tento postup sa opakuje dôvtedy kým hodnota nasledujúceho paketu je väčšia, než hodnota deficitnej čítačky. Vtedy vyčítanie z tej fronty sa skončí, a posunie sa na nasledujucú frontu. Princíp fungovania je znázornený na príklade z obrázka 2.3.



Obr. 2.3: Princíp algoritmu Deficit Round Robin

2.4 Aktívna správa front

Ako už bolo popísané v predchádzajúcich sekciách, strata väčších paketových blokov je nežiadúcim javom u systémoch zabezpečujúcich kvalitu služby. Veľká stráta spôsobuje mierný pokles kvality relácie, a v niektorých prípadoch aj ukončenie relácie. Stratu väčších paketových blokov môže zapríčiniť zaplnenie paketových front [7]. V tomto prípade každý jeden paket, ktorý sa už nezmestí do fronty je zahodený (tzv. tail drop). Aby sa predišlo takovým nežiadajúcim javom systému, boli vyvynuté pokročile mechanizmy pre detekovanie a reagovanie na blížiacieho zaplnenia front.

Náhodné zahodenie paketov nemá výrazní vplyv na pokles kvality služby. Na základe tohoto faktu boli vyvinuté mechanizmy ako Random Early Detection (RED) a Weighted Random Early Detection (WRED). V nasledujúcich častiach je stručný popis princípu týchto mechanizmov.

2.4.1 Random Early Detection

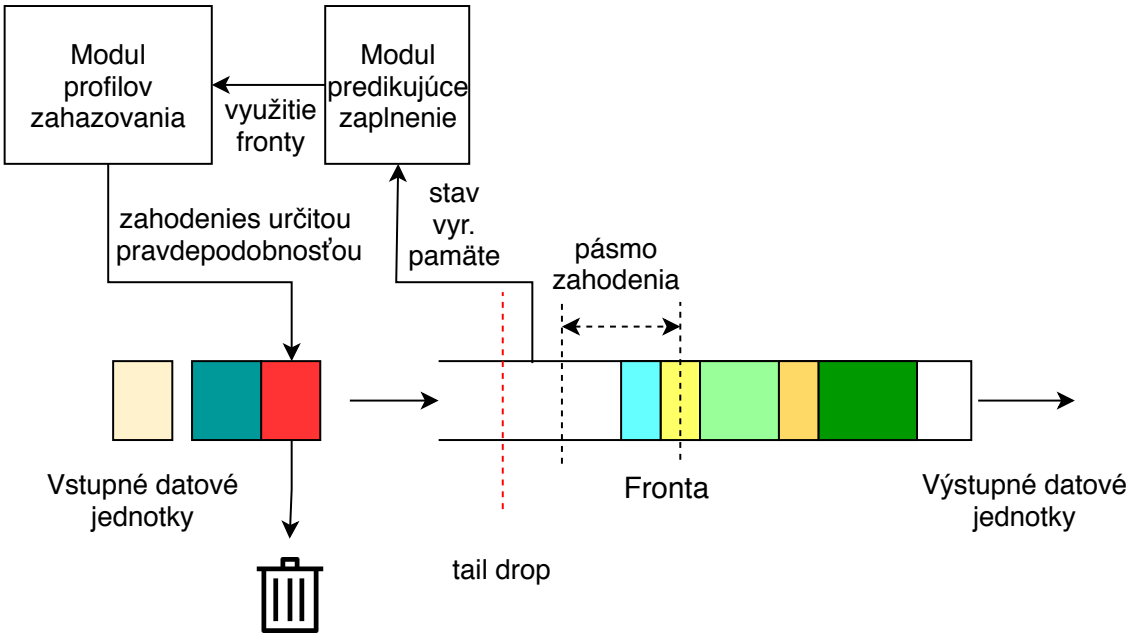
Mechanizmus RED sa skladá z [3]

- modulu predikujúcej zaplnenie vyrovnávacej pamäti,
- modulu profilov zahazovania.

Modul predikujúce zaplnenia fronty je možné používať rôzným spôsobom. Na základe počtu uložených paketov vo vyrovnávacej pamäti, je spočítané využitie fronty. Tento spôsob riešenia je z pohľadu výpočetnosti jednoduchšia. Následne vypočítaná hodnota je prinesená do modulu profilov. Na základe vstupnej hodnoty zaplnenia fronty a podľa nastaveného profilu je vypočítaná pravdepodobnosť zahodenia paketov na vstupu. Princíp fungovania je znázornená na obrázku 2.4. Pri využitia tejto metody sa možú nastať pripady, keď zaplnenie fronty:

- ostáva pod pásmom zahodenia,
  - neaplikuje sa náhodné zahodenie
- je v pásmu zahodenia,
  - aplikuje sa náhodé zahodenie s pravdepodobnosťou p
- prekročuje pásmo zahodenia a hranicu,
  - aplikuje sa tail drop.

Pokročilejšie predikujúce metody su založené na meraní charakteristiky zaplnenia fronty. Po dlhšej časovej analýze, modul je schopný predikovať priemerné zaplnenie fronty. Sice táto hodnota v danom prípade nie je aktualna, ale vyjadruje dĺhodobý priemer zaplnenia fronty.



Obr. 2.4: Schéma mechanizmu Random Early Detection [3]

2.4.2 Weighted Random Early Detection

Weighted Random Early Detection je rozšírená verzia algoritmu RED s podporou triedenia paketov pred zahodením [6]. Táto vlastnosť je dosiahnutá pomocou trojbitovej hodnoty IP Precedense z IP hlavičky. Toto pole definuje prioritu IP paketu pri spracovaní.

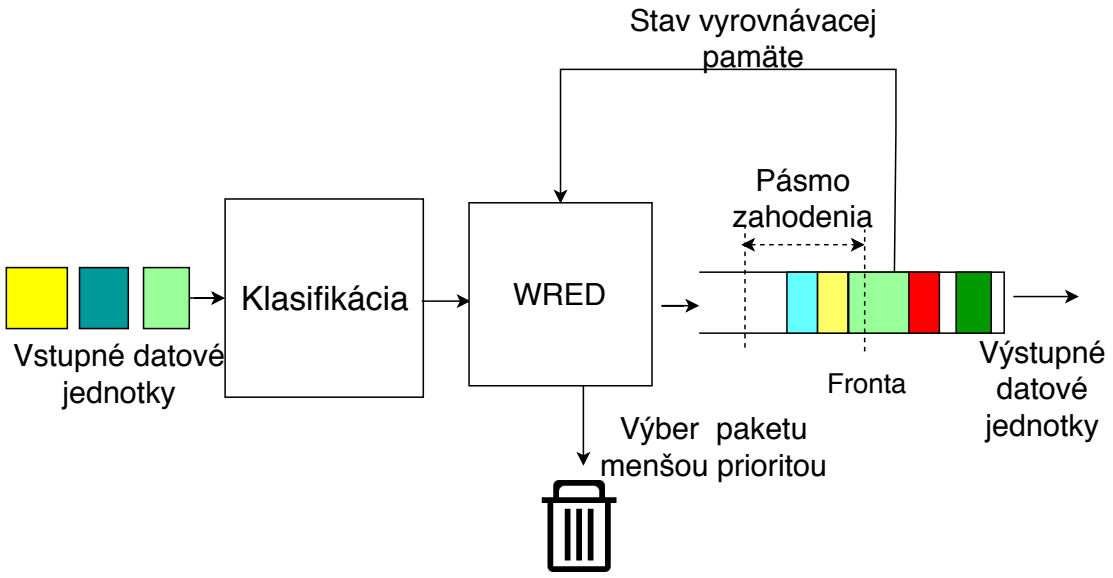
WRED je schopný rozdeliť pakety do rôznych tried, pre ktoré sú nadefinované rôzné politiky zahodenie paketov [6]. Je možné si nastaviť, z ktorých typov paketov bude prevedené náhodné zahodenie. Princíp fungovania je znázornený na obrázku 2.5

2.5 Tvarovanie datového toku

Tieto typy mechanizmov nám umožnia tvarovať rôzné kategórie dátových tokov. Vo vetšine prípadov sa to dosiahne pomocou limitovania prenosovej rýchlosti. Vstupujúce dáta sú uložené do front. Následne pomocou tvarovacích algoritmov sa stanovuje výsledná priepustnosť danej fronty. V tejto sekcie sú popísané dve tvarovacie algoritmy.

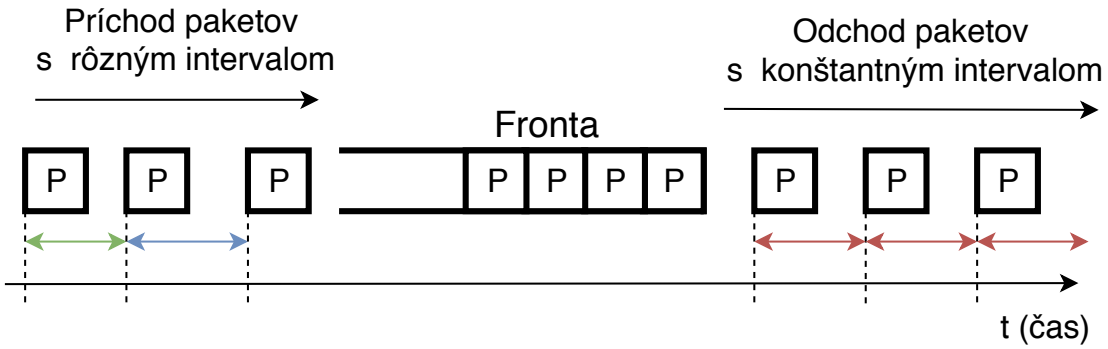
2.5.1 Leaky bucket

Je to jednoduchá varianta obmedzovania priepustnosti. Predstavme si jednoduchú frontu typu FIFO (First In First Out). Prichádzajúce pakety sú uložené do fronty



Obr. 2.5: Schéma algoritmu WRED

po poradí. Naopak vyčítanie z danej fronty je prevedený vždy po určitých intervaloch. Veľkosťou tohto intervalu si môžeme nastaviť rôzne priepustnosti fronty. Na obrázku 2.6 je uvedení princíp algoritmu.



Obr. 2.6: Princíp algoritmu Leaky bucket

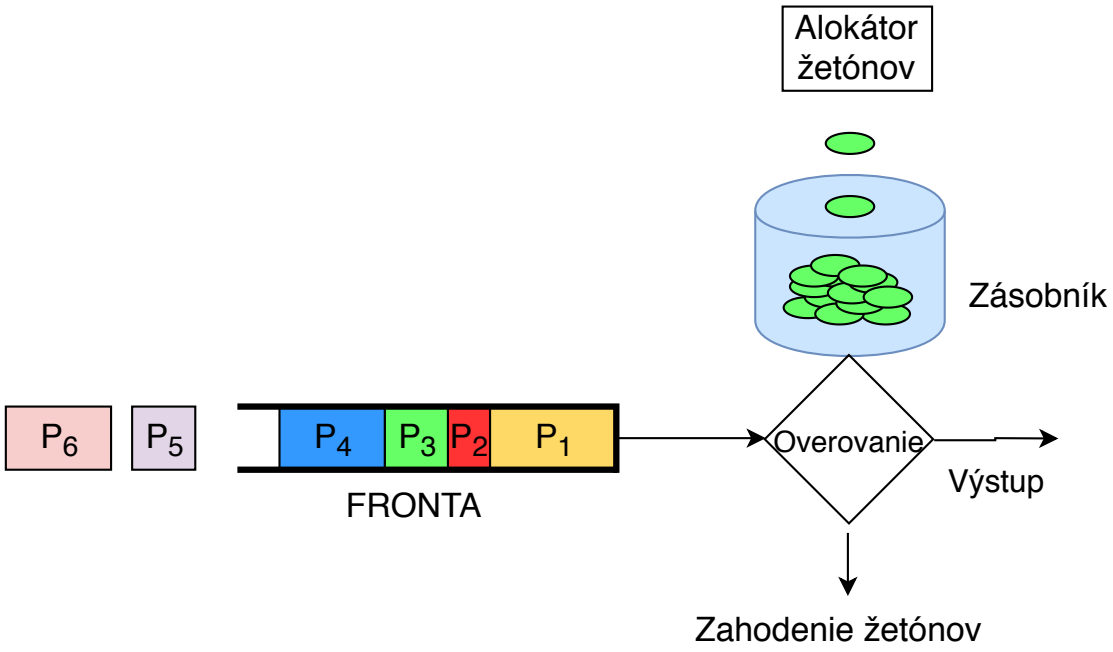
Keď stanovený interval vyčítania paketov z fronty je vetšia ako priemerný interval medzi prichodom paketov, tak výsledná pripustnosť fronty je dosiahnutá bez problémov. V opačnom prípade, keď priemerný interval medzi príchodom paketov je menší ako vyčítací interval, prejavujú nedostatky algoritmu.

- Všetky pakety, ktoré sa nezmestia do fronty, sú zahodené.
- Nepodporuje shluky dátového toku. Výstup fronty má konštantnú priepustnosť.

2.5.2 Token bucket

Vylepšená verzia algoritmu Leacky bucket je Token bucket. Tento algoritmus už podporuje shluky dátových tokov a aj znížila pravdepodobnosť zahodenia paketov na vstupe fronty, ale neodstranila tento problém. Pre odstranenie tohoto problému slúžia algoritmy z kapitoly 2.4 Aktívna správa front.

Algoritmus sa skladá z alokátora žetónov, zo zásobníka, kam sú uložené žetóni a z fronty. Kameňovým prvkom algoritmu sú žetóny (token). Jeden žetón vyjadruje n-tý násobok bitu. Zvolené číslo n závisý od užívateľa algoritmu. Samotný princíp algoritmu spočíva v tom, že alokátor pridáva žetóny do zásobníku s konštantnou rýchlosťou. Keď zásobník je plný žetónov, neprídávajú sa ďalšie. Súčasne prichádzajúci paket sa uloží do fronty. Následne sa overí, či existuje potrebný počet žetónov v zásobníku pre prenos. Keď áno, paket je vyčítaný z fronty a odpočíta sa potrebný počet žetónov zo zásobníka. V opačnom prípade paket počká vo fronte kým nepribudne potrebný počet žetónov. Na nasledujúcom obrázku 2.7 je znázornený princíp algoritmu.



Obr. 2.7: Princíp algoritmu Token bucket

3 VHDL jazyk, FPGA karty a vývojové prostredie

V tejto kapitole sa práca zaoberá s postupom návrhu digitálneho obvodu. V provom rade sa predstavuje jazyk pomocou ktorého sa popíše návrh a jeho chovanie v praktickej časti. V našom prípade tento jazyk je VHDL, ktorý jeden z dvoch najvýznamnejších hardvér popisujúcích jazykov. Následne sa kapitola venuje k obecnému popisu čipu FPGA, ktorý bude využívaný v praktickej časti. Na konci kapitoli je predstavený použitý nástroj [1]Vivado, pomocný framework [2] Netcope Development Kit a samotný proces vytvorenia návrhu na FPGA čip.

3.1 VHDL

Jazyk VHDL (Very High Speed Integrated Circuit Hardware Description Language) je programovací jazyk pre špeciálny popis číslicových obvodov [2].

V roku 1987 bol definovaný štandardom IEEE 1076, ale táto verzia mala ešte množstvo nedostatkov. Väčšinu týchto nedostatkov vyriešila revízia z roku 1993, ktorá je jedna z najpoužívanjších variant v dnešnej dobe. Drohou najpoužívanejšou variantou je VHDL 2008.

VHDL bol navrhnutý tak, aby podporovala všetky úrovne abstrakcie, ktoré sú bežne používane pri návrhovaní [8]. Umožnuje popísať obvod na úrovni

- hradlovej,
- RTL,
- algoritmickej.

Z toho plynie, že jazyk VHDL sa líši od bežných programovacých jazykov. Je vlastne nástroj, pomocou, ktorého sa jednoduchšie popisuje digitálny obvod a jeho chovanie. Výstupný VHDL popis je následovne ľahko zpracovatelný pre simulačné a syntetizačné nástroje.

VHDL kód sa skladá z popisu entity a architektúry [2]. Entita definuje názov a rozhranie danej komponenty. Na nasledujúcom výpisu je znázornená štruktúra entity.

```
entity názov_entity is
  generic(
          názov_generického_parametru : typ_parametru := hodnota;
          )
  port (
          názov_portu: smer_portu typ_portu
```

[1]Vivado, vývojový nástroj od firmy XILINX

[2]Netcope Development Kit, vývojový framework vytvorený spoločosťou Netcope Technologies

```
        );
end entity;
```

Výpis 3.1: Štruktúra entity

Popis entity sa skladá z dvoch sekcií. V sekci port sa definujú porty z ktorých sa skladá rozhranie. U každého porta je potrebné nadefinovať smer (vstupý, výstupný alebo aj-aj), a typ portu (napr. bit alebo bitový vektor). Sekcia generic je volitelná. Pomocou generického parametru definujeme konštantu (typ a implicitnú hodnotu). Následne pri inštancie danej entity si hodnotu generického parametra môžeme zmeniť.

Chovanie obvodu je popísaný v časti architektúra. Architektúra je návrhová jednotka, ktorá je závislá na entite. Každá entita musí mať aspoň jednu architektúru. Na nasledujúcom výpise je znázornená štruktúra architektúry.

```
architecture názov_architektúry of názov_entity
  deklarácia typov, konštant, signálov atď.;
begin
  príkazy;
end názov_architektúry;
```

Výpis 3.2: Štruktúra architektúry

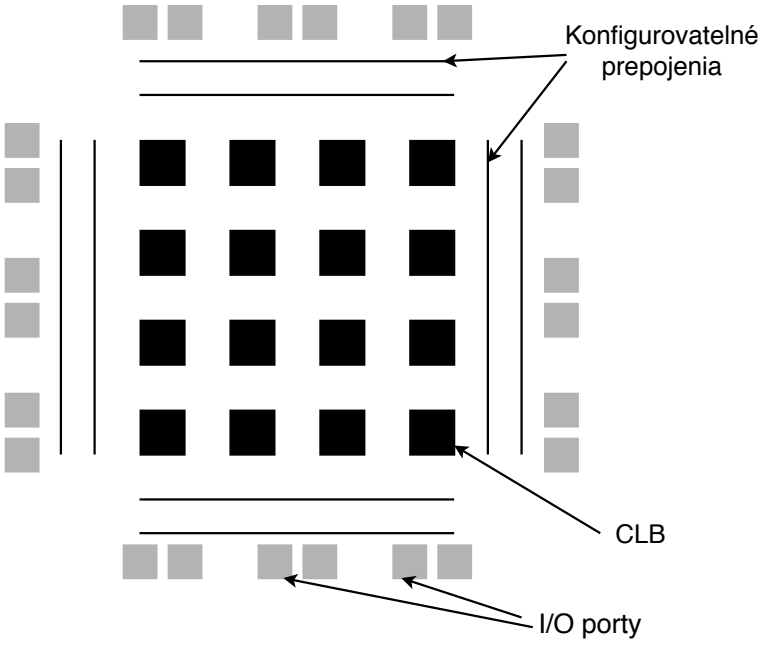
Popis architektúry sa skladá z deklaračnej, kde si môžeme deklarovať pomocné signáli, a z príkazovej časti. Táto časť popisuje logický vzťah medzi vstupnými a výstupnými portmi.

VHDL umožňuje vytváranie testovacej komponenty (testbench), kde si môžeme nadifinovať rôzne logické situácie. Tieto situácie potom odsimulovať na navrhnutej logike pomocou simulačného nástroja.

3.2 FPGA

FPGA (Field Programable Gate Array) je programovateľné hradlové pole. Na rozdieľ od ďalších hradlových polí, ktoré sa programujú metalicky počas výroby, FPGA čipy sú programované elektricky až po výrobe [8]. S tým je získaná jedna najvýznamnejšia vlastnosť týchto čipov, kvôli čomu sa uplatňujú čoraz v širšej škále aplikácií.

FPGA sa skladá z matice programovateľných logických blokov (CLB). Tieto logické bloky sú prepojené s konfigurovateľnými prepojeniami, ktoré potom zaisťujú vazbu na vstupné/výstupné buňky. Na obrázku 3.1 je znázornená zjednodušená štruktúra obvodu FPGA. Základná logická buňka CLB sa skladá z LUT (Look Up Table) a zo sekvenčného prvka [8]. LUTy majú zvyčajne 3 až 6 vstupov a jeden výstup. Po naprogramovania sú schopný realizovať ľubovoľné logické funkcie pre daný počet vstupov. Tieto logické buňky pracujú paralelne. Ako bolo zmienené vyššie



Obr. 3.1: Štruktúra FPGA obvodu

počet vstupov LUTiek nám limituje náročnosť definovanej kombinačnej logiky, avšak pri využití paralelizácie logických blokov môžeme náročnú kombinačnú logiku rozdeliť na menšie časti, a tak dosiahnúť výššie frekvencie a rýchlosť návrhu. Táto vlastnosť je druhou veľkou výhodou čipov FPGA.

Nevýhodou u týchto typov čipov je menšia efektivita využitia zdrojov čipu. Po naprogramovania sa môže stať, že návrhom zabrané zdroj zaberú iba malú časť dostupných zdrojov. Vtedy vznikne neefektivita zabraných zdrojov. To sa dá eliminovať zlúčením návrhov, ktoré môžu, ale nemusia byť spojené na seba. Tie jednotlivé návrhy sú schopné splňiť svoje úkoly navzájom bez toho, aby jeden na druhého čakal.

Samotné naprogramovanie logických blokov je prevedené pomocou špeciálného nástroja. V praktickej časti je používané Vivado 3.3.

3.3 Postup navrhovania a nástroj Vivado

Počas navrhovania je používaný sofistikovaný nástroj, ktorý v našom prípade je Vivado. Vivado nám umožnuje jednoduchší návrh a zrealizovanie danej logiky. Postup navrhovania pomocou nástroja je možné rozdeliť na nasledujúce fázy :

1. vytváranie VHDL popisu a simulácia,
2. syntéza,
3. implementácia,
4. generovanie designu.

3.3.1 Simulácia

Počas navrhovania logiky je veľmi dôležité aby naša logika bola správne otestovaná. V tom nám usnadňuje prácu špeciálny nástroj pomocou, ktorého sme schopný rôznými spôsobmi otestovať, ladiť a verifikovať funkčnost našej logiky. Ako bolo popísané vyššie, jazyk VHDL umožnuje nám napísať testovacie komponenty avšak existujú aj rôzné jazyky ako System Verilog pomocou, ktorých je možne napísať automatické testy. Taký typ testovania je použitá zvyčajne pri veľkých návrhoch, kde manuálne testovanie bolo zdlhavé a časovo neefektívné.

3.3.2 Syntéza

Syntéza je proces, ktorá spraví prevod z VHDL popisu na všeobecné elementárne buňky. V našom prípade výstupom syntézi je tzv. RTL schéma, ktorá je všeobecné zapojenie hradlov a klopných obvodov nezávislé na cieľovom obvode. Počas syntézi je prevedená nad výstupnou RTL schématou vyhľadávanie kritických ciest a optimalizácia logických funkcií. Aby syntetizér správne pochopil náš návrh, treba zachovať rôzné pravidlá pri popisu. Nie všetky konštrukcie kódov sú vysyntetizovateľné. S chybnou syntézou nie je možné návrh implementovať a práve preto je veľmi potrebné aby návrh bol syntetizovateľný bez chýb.

3.3.3 Implementácia

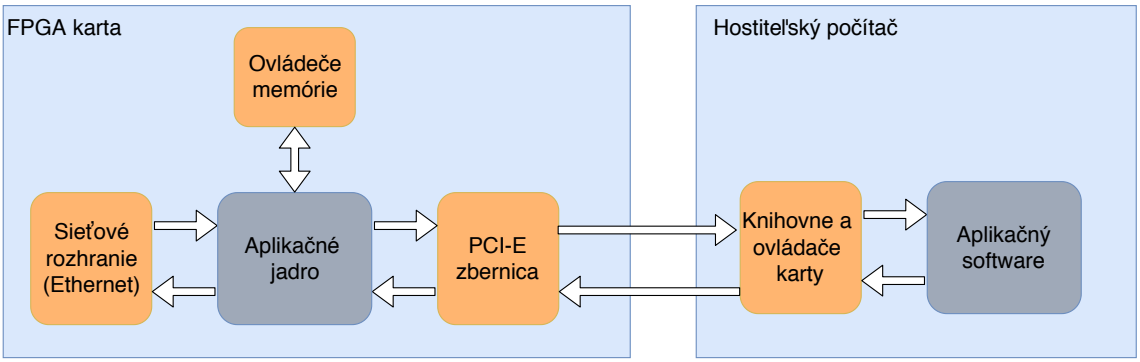
Po úspešnej syntéze je prevedená implementácia. Implementácia sa skladá z rôzných krokov. Prevedie sa najprv mapovanie všeobecnej štruktúry zo syntézi na reálne zdroje cieleného obvodu. Následne sa prevedie umiestenenie a prepojenie mapovaných zdrojov. Pred ukončením implementácie je prevedená dôkladná optimalizácia. Počas optimalizácie sa eliminujú vzniknuté kritické cesty, keď je možné, a optimalizuje sa využitie zdrojov čipu.

Po úspešnej implementácií sa vygeneruje bitový súbor pomocou, ktorého je možné naprogramovať FPGA.

3.4 Vývojový framework Netcope Development Kit

Netcope Development Kit (NDK) je konfigurovateľný framework od firmy Netcope, ktorý zjednodušuje prácu vývojárom pri vyvýjaní sieťových aplikacií na akceleračných kartách s FPGA [9].

Štruktúra frameworku je znázornená na nasledujúcom schéme nižšie. Skladá sa z firmvérovej a softvérovej časti.



Obr. 3.2: Štruktúra NDK [9]

Firmvérová časť je nahraná do karty, kým softvérová časť je nainštalovaný na hostiteľskom počítači. Vzájomná komunikácia je vyriešená cez PCI-Express rozhranie. Jádrom firmvérovej časti je User Application Modul (ďalej aplikačné jádro), kam si môže vývojár pripojiť svôj návrh. Komunikácia aplikačného jádra so sieťom a s hostiteľským počítačom už je vyriešené. Aby sme mohli správne napojiť náš návrh do aplikačného jádra je potrebné sa oboznámiť špeciálnymi komunikačnými rozhraniami Frame Link Unaligned (FLU) a MI32 firmy Netcope.

3.4.1 Rozhranie FLU a MI32

FLU je synchronné rozhranie s hodinami aplikačného jádra. Využíva sa pre prenos paketov zo sieťového modulu do aplikačného jádra, alebo pri penose dat cez DMA do hostiteľského počítača. Datové slovo môže mať šírku 256 bitov a viac. V nasledujúcej tabuľke sú znázornené jednotlivé signáli rozhrania a jejich popis pri 512 bitového šírke slova. Podrobnejšie je popísané v [9].

Názov signála	Posieľá strana	Popis
DATA	Vysielacia	Prenesené dátové slovo.
SOP	Vysielacia	Indikuje validný štart packetu v dátovom slove.
SOP_POS	Vysielacia	Trojbitový signál ukazuje pozíciu začiatku paketu. Pozícia je získaná vynásobením hodnoty signálu s 8 bytmi. Napr. pri hodnote "001" validný začiatok paketu by bola na pozicií DATA(127 downto 64).
EOP	Vysielacia	Indikuje validný koniec packetu v dátovom slove
EOP_POS	Vysielacia	Šesť bitový signál ukazuje pozíciu posledného bajtu paketu.
SRC_RDY	Vysielacia	Indukuje, že vysielacia strana je pripravená vyslať dáta.
DST_RDY	Prijímacia	Indukuje, že prijímacia strana je pripravená prijať dáta.

Tab. 3.1: Tabuľka rozhrania FLU [9]

MI32 rozhranie sa využíva pre konfiguráciu komponent z firmvérovej časti pomocou sofvéru. Komunikácia je vyriešená pomocou adresovej štruktúry. Každá komponenta má svoj adresný priestor, kde je dostupný. Rozhranie podporuje zápis i výpis. Šírka dátového slova je pevne nastavená na 32 bitov. Na nasledujúcej tabuľke sú popísane signáli rozhrania MI32. Detailnejší popis rozhrania je uvedený v [9].

Názov signála	Šírka (bit)	Posieľá strana	Operácia	Popis
MI_DWR	32	Software	Zápis	Dáta pre zápis.
MI_DRD	32	Firmware	Výpis	Dáta pre výpis.
MI_ADDR	32	Software	Zápis i výpis	Adresa registra.
MI_WR	1	Software	Zápis	Povel pre zápis.
MI_RD	1	Software	Výpis	Povel pre výpis.
MI_BE	4	Software	Zápis i výpis	Povolené byty.
MI_ARDY	1	Firmware	Zápis i výpis	Adresa je dostupná.
MI_DRDY	1	Firmware	Výpis	Dáta sú dostupné.

Tab. 3.2: Tabuľka rozhrania MI32 [9]

3.4.2 Sieťová akceleračná karta NFB200G2QL

Akceleračná karta je schopná kommunikovať so sieťom a s hostiteľským počítačom. Má dve sieťové rozhrania. Každá jedna je schopná vysieľať a prijať rýchlosťou 100 Gbits/s. Ako sieťové rohranie sú používané vysokorýchlostné transceivery QSFP28.

Obidve rozhrania sieťovej akceleračnej karty `NFB-200G2QL` podporujú nasledujúce módy:

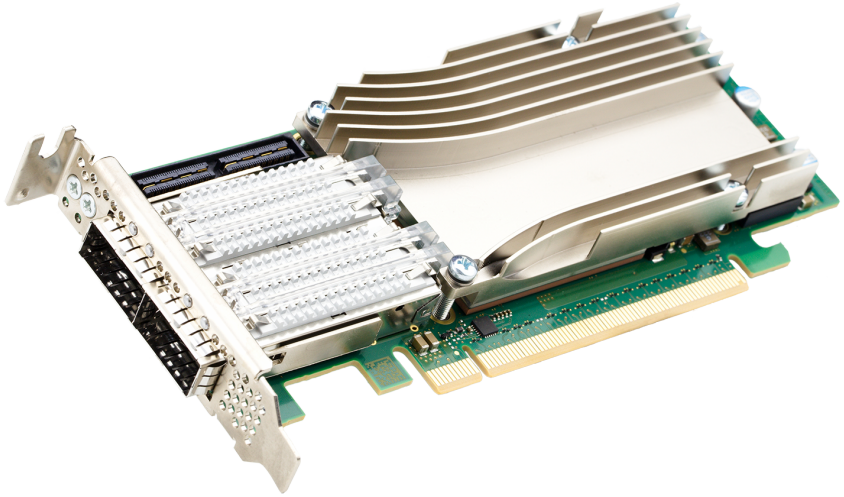
- 2×50G,
- 1×40G,
- 4×25G,
- 4×10G.

Prepojenie s hostiteľským počítačom je vyriešené pomocou zbernice PCI-E Gen3 x16 (x16). Cez túto zbernicu je možné vysielať rýchlosťou 128 Gb/s.

Sieťová karta je osadená s výkonným FPGA čipom Xilinx Virtex UltraScale+ (XCVU7P). Hlavné konfiguračné parametry FPGA čipu sú následovné.

- Konfiguračné bloky služiace pre
  - logiku (LUT) : 788160
  - pamäť (LUT_RAM) : 394560
  - registery (FF) : 1576320
- Blokramky (BRAM) : 1440

Na obrázku 3.3 je znázornená sieťová akceleračná karta `NFB-200G2QL`.

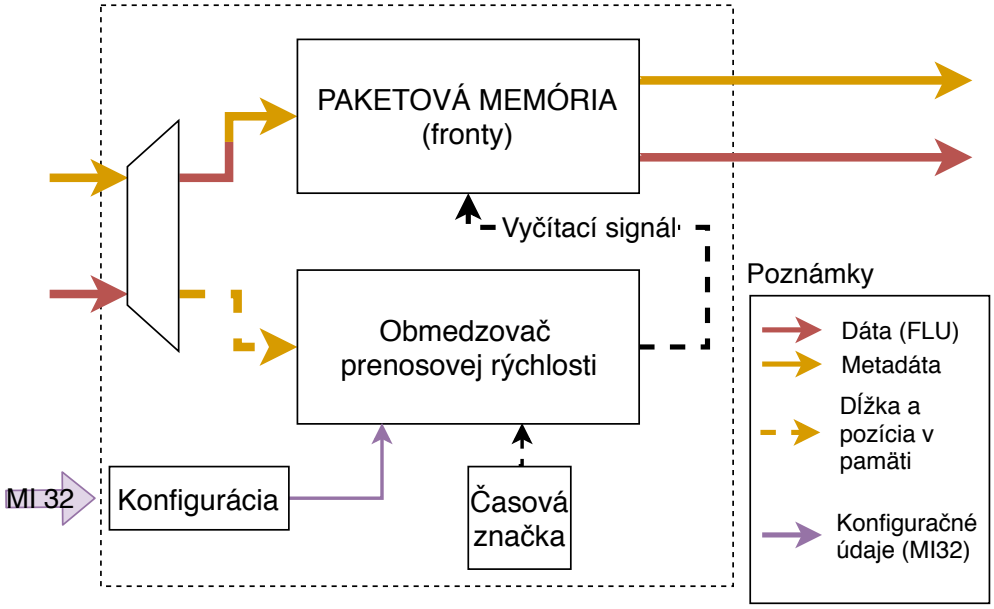


Obr. 3.3: Sieťová akceleračná karta NFB200G2QL

4 Obmedzovač priepustnosti viacerých front

Sieťová akceleračná karta NFB-200G2QL s FPGA čipom je schopná pri využití obidvoch aplikačných jadier spracovávať dátové toky na rýchlosti 2×100 Gb/s. Využitie tejto vlastnosti pri zabezpečenia dostatočnej kvality služby by výrazne zvýšila efektivitu daného systému. Z toho dôvodu praktická časť práce sa venovala ku vytvoreniu návrhu v jazyku VHDL, ktorá obmedzuje priepustnosti rôzných dátových tokov. Podmienkou návrhu bola možnosť kontrolovania a obmedzovanie priepustnosťi viacerých implementovaných front navzájom. Výsledná schéma komponenty je uvedená na obrázku 4.1. Komponenta používa tri rôzne vstupné rozhrania:

- metadátové,
- dátové (FLU viz. 3.4.1),
- konfiguračné (MI32 viz. 3.4.1).



Obr. 4.1: Všeobecné schéma navrhnutého obmedzovača priepustnosti

Ku každému paketu po spracovávaní v prostredí NDK pribudnú pomocné metadáta popisujúce dĺžku, časovú značku paketu a ešte niekoľko ďalších pomocných informácií nevyhnutných pre nasledujúce časti frameworku. Pre našu komponentu je najdôležitejšia dĺžka dát. Pre samotné dáta (pakety) sa používa FLU rozhranie. Posledné používané rozhranie je MI32, ktorá slúži pre konfiguráciu obmedzovača.

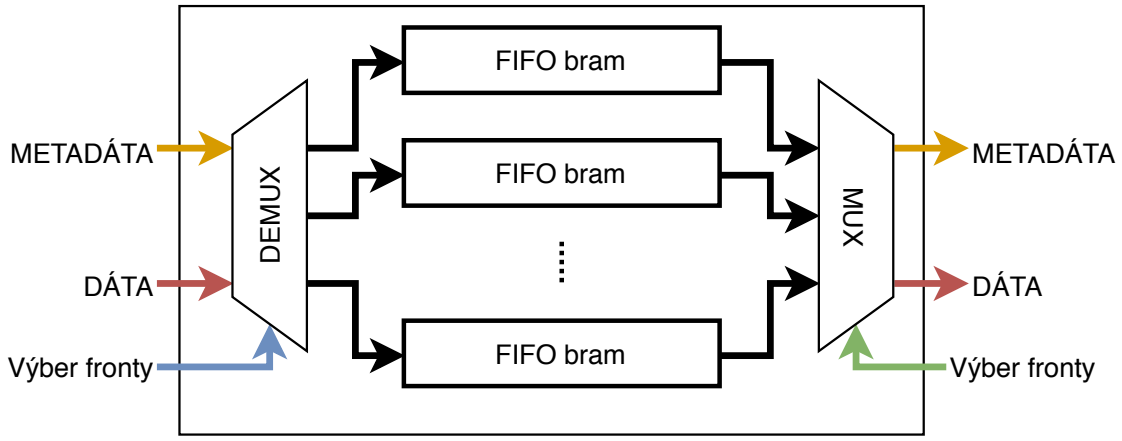
Pre lepšie znázornenie praktickej časti, návrh sa dá rozdeliť na

- dátovú (paketová memória - fronty),
- riadiacú (obmedzovač prenosovej rýchlosti),
- a konfiguračnú časť.

V nasledujúcich sekciách budú podrobnejšie popísané jednotlivé časti.

4.1 Dátová časť

V tejto časti sa vytvorila komponenta, ktorá generuje fronty pre vstupné dáta. Pre každu jednu frontu sa používa blokramkové FIFO (First in First Out). Počet inštancí front sa nastavý hodnotou generického parametra `FIFO_NUM`. Počet položiek a veľkosť fronty udáva hodnota parametrov `FIFO_ITEMS` a `SIG_WIDTH` (súčet dĺžky dát, metadát a FLU signálov). Pomocou týchto hodnôt je možne jednoducho zmeniť konfiguráciu front bez akejkoľvek zmeny v zdrojovom kóde. Schéma komponenty je znázornená na obrázku 4.2.



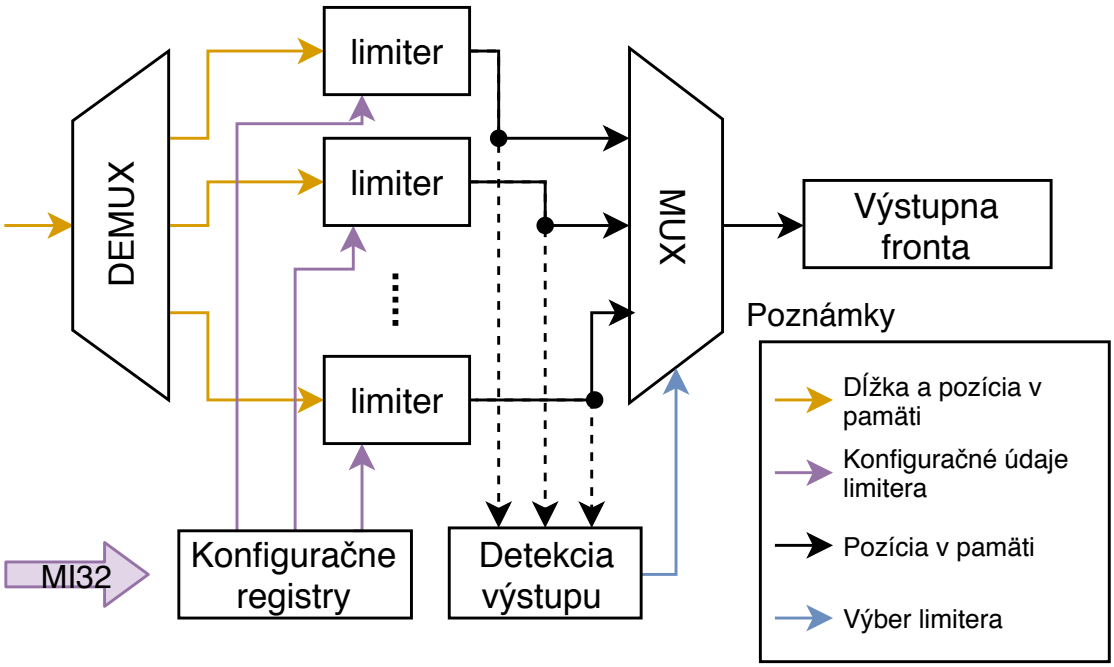
Obr. 4.2: Všeobecné schéma pamäťovej časti

Na vstup komponentov sú privedené dáta a metadáta súčasne. Následne pomocou demultiplexora sa uložia dáta a metadáta do fronty určením vstupom `SEL_WR_FIFO` (modrá šipka na obrázku 4.2). Pri vyčítania sa používa multiplexor, ktorý na základe vstupu `SEL_RD_FIFO` (zelená šipka na obrázku 4.2) vyčíta dáta z fronty a pošle na výstup.

4.2 Riadiaca časť

Doterajší text sa zaoberala so zapisovaním prichádzajúcich vstupných dát a metadát do jednotlivých front. V nasledujúcej časti sa podrobnejšie popíše princíp fungovania samotného limitovania jednotlivých front. Ako je aj znázornené na obrázku 4.1 riadiaca časť implementácie používa

- dĺžku vstupných dát (položka v metadátach),
- signál určujúci danú frontu pre zápis (pozícia v pamäti),
- konfiguračné parametry,
- a časovú značku.



Obr. 4.3: Všeobecné schéma riadiacej časti

V návrhu bolo stanovené, aby priepustnosť každej fronty bolo možné rôzne limitovať. Z toho dôvodu sa generuje rovnaký počet limitovacých komponent ako front. S týmto princípom sa docielí ktomu, aby jednotlivé limitery regulovali priepustnosť jednotlivých front. Všeobecné schéma tejto časti je uvedená na obrázku 4.3. Pozícia uloženého paketu v datovej časti a samotná dĺžka vstupuje do demultiplexora. Na základe pozície sa vyberie limiter, ktorý reguluje danú frontu. Pre priehľadnejšie vysvetlenie tejto sekcie si najprv podrobnejšie popíšeme fungovanie limitovacej komponenty jednej fronty.

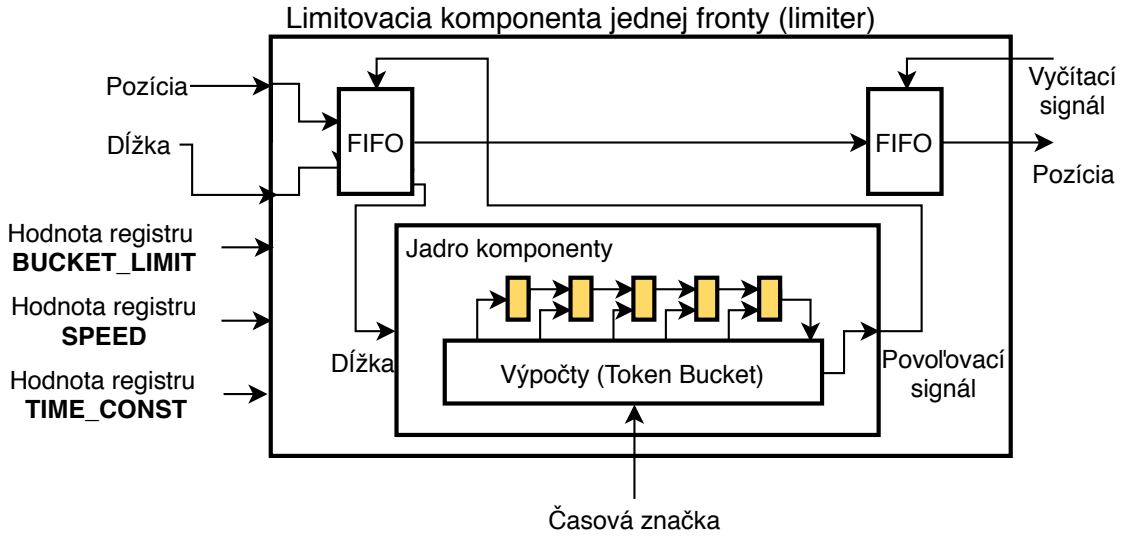
4.2.1 Limitovacia komponenta jednej fronty (limiter)

Táto komponenta uskutočňuje samotné limitovanie priepustnosti fronty. Skladá sa zo vstupnej fronty, limitovacieho jádra a výstupnej fronty. Vstupujúca dĺžka dát a pozícia v pamäti, rešp. fronty sa uloží do vstupnej fronty (FIFO). Následne dáta su napojené na samotné jádro limitovania.

Limitovacie jádro je založené na princípe algoritmu Token Bucket 2.5.2. Vlastný svôj zásobník žetonov. Pre správny výpočet jadro musí vedieť

- aktuálnu časovú značku (timestamp),
- čas posledného prepusteného paketu,
- stav zásobníka po poslednom prepustenom paketu,
- dĺžku aktuálnych dát,
- konfiguračné údaje :

– velkosť zásobníka v bajtoch (register `BUCKET_LIMIT`),
– hodnotu vyjadrujúcu mieru medzi žetónami a bajtmy (register `SPEED`),
– počet pridávaných žetonov s každým taktom hodín (register `TIME CONST`).



Obr. 4.4: Všeobecné schéma limitovacej komponenty jednej fronty

Princíp fungovania jádra je následovné. Namiesto toho aby jadro udržovala hodnotu zásobníka žetonov v každom jednom takte hodín, si vypočíta rozdieľ medzi aktuálnou časovou značkou a časovou značkou posledného prepusteného bajtu. S tým sa získa uplynulá doba medzi posledným limitovaním a príchodom nového paketu. Vynásobením tejto doby s registrom `TIME_CONST` je získaný počet pribudlých žetónov. Táto hodnota sa pridá k hodnote zásobníka po poslednom prepustenom pakete. Následne sa overí či nový počet žetónov nepresiahuje maximálnú možnú velkosť zásobníka. Pritom súčasne sa prevedie dĺžka paketov do podoby žetónov pomocou hodnoty registru `SPEED`. Algoritmus sa skončí s overením či je v zásobníku potrebný počet žetónov pre prenos. Keď ano, označí sa paket za prepustený, odpočíta sa potrebný počet žetónov zo zásobníka a aktualizuje čas a hodnotu zásobníka pre posledný prepustený paket. V opačnom prípade nič sa neaktualizuje a algoritmus sa začne znova. Aby časová značka bola v každom takte aktuálna, vytvoril sa pomocný čitač. S každým taktom hodín si inkrementuje svoju hodnotu o jedna.

Po uspešném výpočtu algoritmu, hodnota ukazujúca poradie fronty (pozícia), kde sa paket nachádza v pamäti je vyčítaný zo vstupnej fronty a uložený do výstupnej fronty limitovacej komponenty. Následne táto fronta je napojená na výstup komponenty.

4.2.2 Konfiguračná časť

Výsledná priepustnosť jednotlivých limiterov front sa nastavuje pomocou konfiguračných registrov `SPEED` a `TIME_CONST`.

Hodnota registru `SPEED` (token/B) udáva mieru prepočtu bajtov na žetóni. To s tým spôsobom, že po prepočtu jeden bajt sa rovná `SPEED` žetónov. Register `TIME_COSNT` vyjadruje počet žetónov pri každej alokáci. Celá komponenta je synchronná voči hlavnej časovej doméne. Z toho plynie, že hodnota registru `TIME_CONST` (token) vyjadruje počet alokovaných žetónov do zásobníka v každom jednom takte hodín. Nastavenie priepustnosti sa dá popísať pomocou rovnice 4.1. Kde R (B/s) je teoretická priepustnosť a F (Hz) je frekvencia časovej domény.

$$\frac{\mathrm{R}}{\mathrm{F}} = \frac{\mathrm{TIME\_CONSTANT}}{\mathrm{SPEED}} \tag{4.1}$$

Ľavá strana rovnice popisuje potrebný počet prenesených bajtov v každom jednom hodinovom takte pre očakávanú priepustnosť R. Nastavenie pravej strany už závisí na samotnom užívateli.

Veľkosť shlukov dát je popísané s hodnotou registru `BUCKET_LIMIT`, ktorá udáva veľkosť virtualného zásobníka v bajtoch a výsledním nastavením pravej strany rovnice 4.1. S väčšou hodnotou `SPEED` sa zvišuje aj veľkosť zásobníka.

Samotný zápis a výpis registrov je prevedené cez MI32 rozhranie. Pri limitovania viacerých front každá jedna komponenta (limiter) má svoje konfiguračné registry. Tieto registry sa nachádzajú na rôznych MI32 adresách.

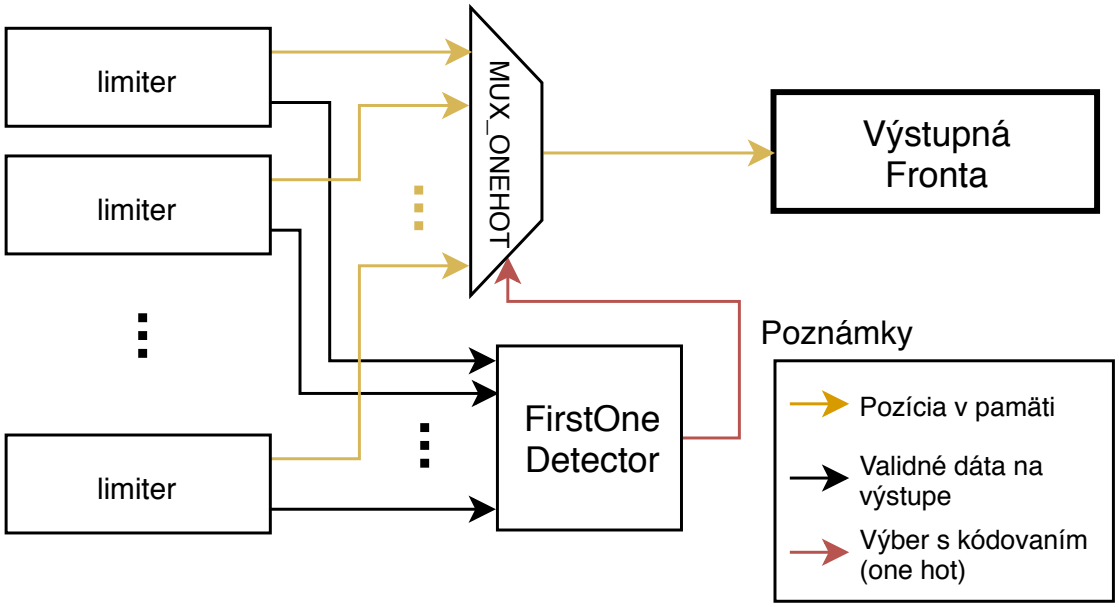
4.2.3 Detekcia výstupu limiterov

V tejto sekcie je popísaný princíp detekovania validných výstupov jednotlivých limiterov. Všeobecné schéma je znázornená na obrázku 4.5. Ako už bolo popísané vyššie, každý limiter dá na svoj výstup pozíciu paketu vo fronte a jednobitový signál určujúci validitu výstupu (`SRC_RDY`).

Jednotlivé pozície sú napojené na špeciálný multiplexer. Tento multiplexer prevedie výber vstupu na výstup na základe signála s one-hot kódovaním[1]. Pomocné signáli vstupujú do komponenty `FirstOneDetector`, kde sa určí prvý detekovaní limiter. Výstup má v podobe one hot kódovania. Práve kvôli tomu sa používa špeciálný multiplexer. Napríklad v prípade, keď stav výstpu komponenty `FirstOneDetector` je 0b0010, tak multiplexer si zvolý druhý vstup ako svôj výstup. V prípade 0b1000 si zvolí štvrtý vstup ako výstup.

Detekované hodnoty pozície v pamäti (poradie fronty) sú zapísané do výstupnej

[1]Typ kódovania, kde dĺžka vektora je rovná počtu stavov. Jednotlivé stavy sú určené s jednou logickou jedničkou kým ostatné prvky vektora sú nuly.



Obr. 4.5: Všeobecné schéma detekovania výstupov limiterov

fronty obmedzovacej časti. Z tejto fronty sa následne prebieha vyčítanie komponentom paketovej pamäti 4.1. Vyčítanie prebehne na základe validných pomocných signálov uložených do jednotlivých front sučasne s dátami. Pomocou poskytnutej pozície z obmedzovacej časti sa vybere fronta, kde sa paket nachádza. Následne vyčítanie trvá od validného začiatku (`SOP`) až po validné ukončenie dátové slova (`EOP`)

4.3 Testovanie a výsledky

Táto časť textu sa venuje k popisu testovania navrhnutej komponenty a dosiahnutým výsledkom. Samotné testovanie prebehlo s troma rôznymi spôsobmi. Ako prvé sa overila funkčnosť navrhnutej logiky v simulačnom prostredí. Následne sa testoval počet zaoberaných zdrojov na čipu s rôznymi nasteveniami obmedzovača. Ako posledné sa testovalo reálne chovanie komponenty zabudované do frameworku `Netcope Development Kit` na sieťovej akceleračnej karte `NFB_200G2QL`.

4.3.1 Overovanie funkčnosti pomocou simulácií

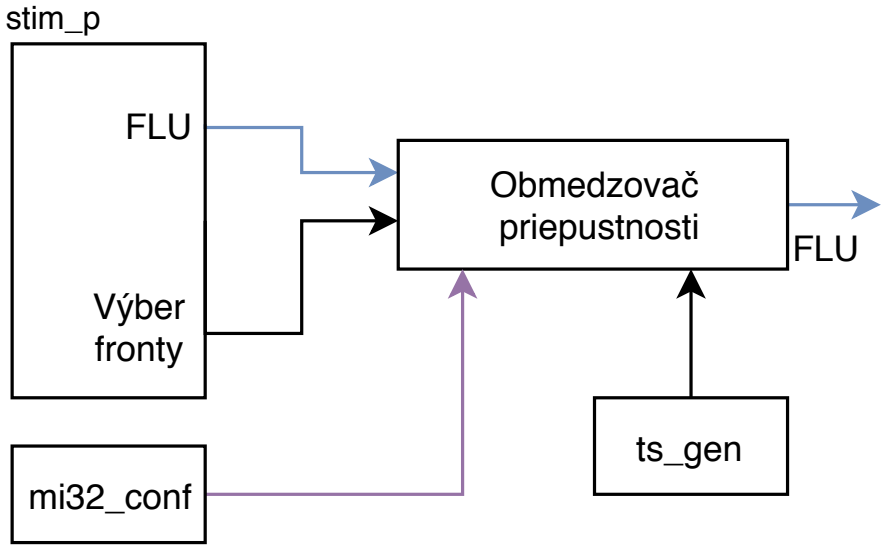
Pre vygenerovanie firmvérov sa používal sofistikovaný nástroj Vivado 3.3. Tento nástroj je možné využívať aj pre overovanie funkčnosti návrhu na úrovni simulácie. Avšak existujú konštrukcie kódov, ktoré nástroj je schopný vysyntetizovať, ale odsimulovať nie. V našom návrhu existujú komponenty pracujúce s takovými pomocnými

dátovými typmi, ktoré Vivado nie je schopný odsimulovať. Kvôli vzniknutému obmedzeniu pre overovanie funkčnosti sa používal nástroj ModelSim[2].

Simulácie boli hlavne zamerané pre overenie

- funkčnosti navrhnutej logiky (dosiahnutie nastavených priepustností),
- rovnakého počtu dát na výstupe ako na vstupe,
- eliminovaniu nežiadajúcich stavov (zaseknutie návrhu),
- správneho prenosu dát medzi jednotlivými komponentami,
- validnej konfigurácie cez MI32 rozhranie.

Pre zrealizovanie simulácie bolo potrebné navrhnúť testovací komponent. Štruktúra tejto komponenty je uvedená na obrázku 4.6.



Obr. 4.6: Všeobecné schéma testovacej komponenty

Celá štruktúra zahrňuje do seba navrhnutý obmedzovač front a pomocné procesy pre simuláciu. Pomocou procesu `stim_p` sa generuje `FLU` dátový tok (DMA alebo ETH), a s každým validným začiatkom generovaného toku aj signál slúžiaci pre výber fronty. Proces `mi32_conf` je navrhnutý na základe funkčnosti `MI32` rozhrania. Tento process sa využíva pre simulovanie konfiguračnej časti obmedzovača front. Pri každej simulácie, kde sa jedná o synchronný návrh je potrebné si nadefinovať hodinový a resetujúci signál.

Výsledná znázornená simulácia bola prevedená na menšej časti aplikačného jádra, kde samotný obmedzovač je používaný. S týmto krokom sa lepšie overovalo chovanie navrhnutej komponenty vo vývojovom prostredi `NDK` a nebolo potrebné zvlášť generovať metadáta, na základe ktorých obmedzovač pracuje. Pri všetkých testovacích prípadoch

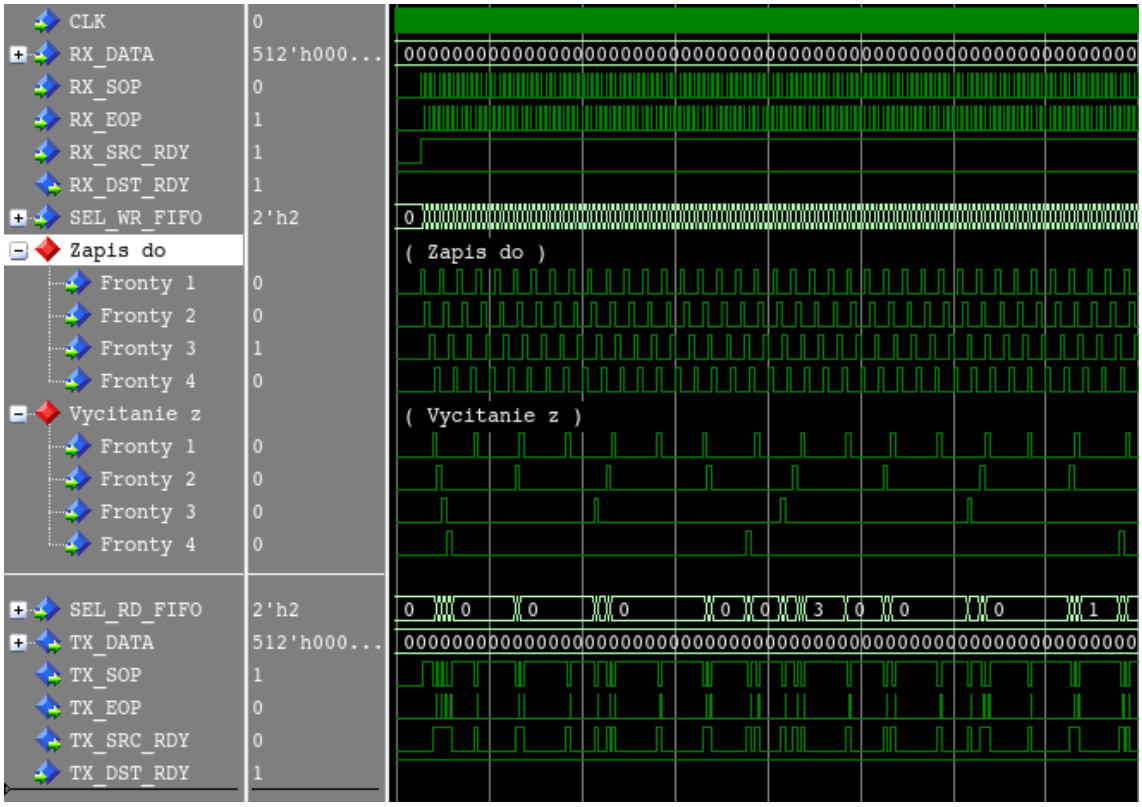
[2]ModelSim je simulačné prostredie pre hardvér popisujúce jazyky vyvíjané spoločnosťou Mentor Graphics [15].

- sa generovali dáta s dĺžkou 512 bajtov,
- vstupné dáta boli uložené do štyroch front na základe `Round Robin` selekcie,
- používala sa rovnaká hodnota
  - konštanty pre prevod bajtov na žetónov (`SPEED`).

Pre prvý testovací prípad, ktorý je znázornený na obrázku 4.7, sa zvolili rôzne počty pridávaných žetonov s každým taktom hodinového signála (`TIME_CONST`). Tieto nastavenia sú uvedené v tabuľke 4.1.

Fronta	TIME_CONST	Teoretická rýchlosť (Mb/s)	Počet bitov za takt hodín
1	128	10240	51,2
2	64	5120	25,6
3	32	2560	12,8
4	16	1280	6,4

Tab. 4.1: Tabuľka nastavenia alokácie žetónov jednotlivých front a predpokladané rýchlosti



Obr. 4.7: Výstup simulácie - Prvý testovací prípad

V hornej časti obrázku 4.7 sú znázornené prichádzajúce dáta do jednotlivých

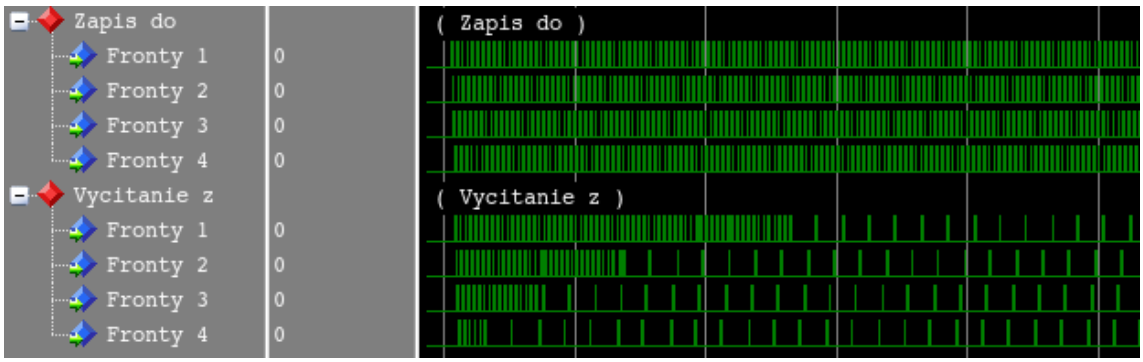
front. Do všetkých front sa zapisuje s rovnakou rýchlosťou. Významnou časťou výstupu tejto simulácie sú signáli určujúce vyčítanie z jednotlivých front. Z obrázku plynie, že najväčšiu pripustnosť ma prvá fronta. Pre lepšie overenie dosiahnutej rýchlosť prvej fronty sa namerala uplynulá doba medzi prvým a ôsmym vyčítaním. Táto doba odpovedala hodnote 800 nanosekund. Z tabuľky plynie, že pre dosiahnutie konfigurovanej priepustnosti je potrebné v každom takte vyčítať 51,2 bitov z fronty. Za nameranú dobu 800 nanosekund (ďalej d) bolo prevedené vyčítanie dáta osem krát z fronty ($4096\text{bit} = 8 \times 8 \times 64$). Nastavená perióda hodinového signála je nastavená na 10 nanosekund, takže počet uplynulých taktov za dobu d je 80. Výsledný výpočet rozloženia vyčítaných dát na jednotlivé takty sa určí pomocou podielu počtu prenesených dát s počtom uplynulých taktov za dobu d.

$$\frac{4096 \text{ bit}}{80 \text{ takt}} = 51,2$$

Vypočítaná hodnota priepustnosti je rovnaká ako stanovená teoretická hodnota. Pokročilejšie overenia dosiahnutých obmedzení neboli prevedené počas simulácie, ale pri realnej využitia na karte `NFB-200G2QL` v sekci 5.

Nasledujúci výstup simulácie sa zameriavali na znázornenie dátových shlukov front pri rôzných definovaných veľkostiach zásobníkov žetónov. Jednotlivé fronty majú nastavenú rovnakú priepustnosť. Na obrázku výstupu simulácie 4.8 sa používajú zásobníky s veľkosťou

- 16384 bajtov (327680 žetónov).
- 8192 bajtov (163840 žetónov).
- 4096 bajtov (81920 žetónov).
- 2048 bajtov (40960 žetónov).



Obr. 4.8: Výstup simulácie pri rôznych veľkostiach zásobníka

Na začiatku simulácie za istú dobu sa nevysielajú dáta do obmedzovača, aby jednotlivé zásobníky sa zaplnili s žetónmi. Z výstupu simulácie plynie, že začiatočné vyčítania z front sa uskutočňujú s väčšou frekvenciou, než by sa malo podľa stanovených rovnakých teoretických hodnotách. Táto doba udáva veľkosť podporovaných

shlukov dát. Z výstupných simulácií je zrejmé, že veľkosť shlukov dát (ďalej burst) závisí na veľkosti zásobníka. Pri prvej a štvrtej fronty z obrázku 4.8 je vidno veľký časový rozdieľ pretrvávania javu dátových burstov.

4.3.2 Zabrané zdroje čipu pri rôzných konfiguráciach

Sieťová akceleračná karta `NFB-200G2QL` je osadená s výkonným FPGA čipom Xilinx Virtex UltraScale+ (XCU7P). Počet využitelných zdrojov čipu (logické buňky (LUT), registery (FF), blokramky (BRAM)) sú zhrnuté v tabuľke 4.2.

Typ zdroja	Počet
LUT ako logika (LUT)	788160
LUT ako pamäť (LUT_RAM)	394560
Registery (FF)	1576320
Blokové ramky (BRAM)	1440

Tab. 4.2: Tabuľka zdrojov čipu Xilinx Virtex UltraScale+

Pomocou syntézneho nástroja VIVADO sa otestovali zabrané zdroje obmedzovača priepustnosti pri rôzných nastaveniach. Počas testovania sa zvyšovala počet generovaných front a počet položiek front v datovej a riadiacej časti navrhnutého obmedzovača. Použité nastavenia a výsledky syntézi sú zhrnuté v tabuľke 4.3. Pre lepšie znázornenie čerpané zdroje sú vyjadrené počtovo a percentuálne.

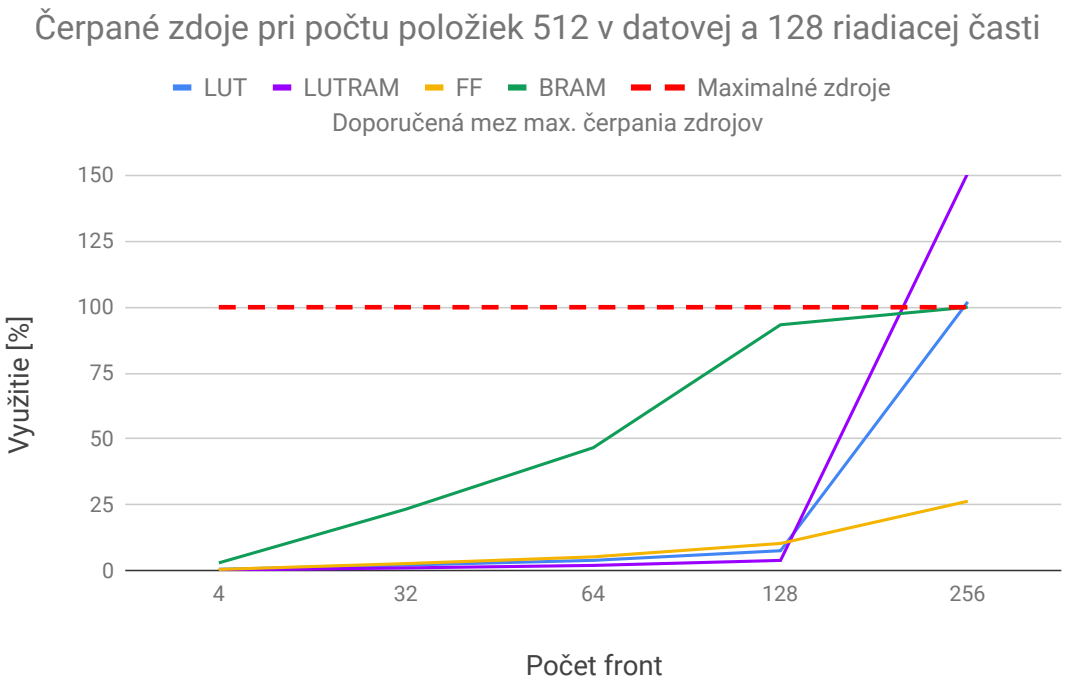
Pre začiatočné syntézy sa zvolila konštantný počet položiek front. V dátovej časti sú nastavené blokramkové fronty s 512 položkami. Kým v riadiacej časti z dôvodu menších pomocných dát sa zvolilo 128 položiek. Výsledky čerpaných zdrojov pre tieto nastavenia sú znázornené na grafu 4.9.

S postupným zvýšovaním počtu generovaných front sa lineárne zvyšovalo čerpanie logický buňiek a registrov na čipu. Ku zlomovému bodu linearity je dosiahnuto pri generovaní 128 front. V tomto prípaďe obmedzovač využije až 93 % možných blokramiek. Od tohoto bodu, je viditelný enormný nárast použitých logických buniek. Syntetizačný nástroj má definované, aby jednotlivé generované fronty čerpali blokramky. Avšak v našom prípade už neexistuje dostatočný počet pamäťových zdrojov, z toho dôvodu nástroj začne čerpať logické buňky slúžiace pre kobinačnú logiku a pamäť (`LUT`, `LUTRAM`). Tento jav sa nastane pri generovania 256 front, kde následne počet použitelných logických buňiek prekročuje mez využitelných zdrojov.

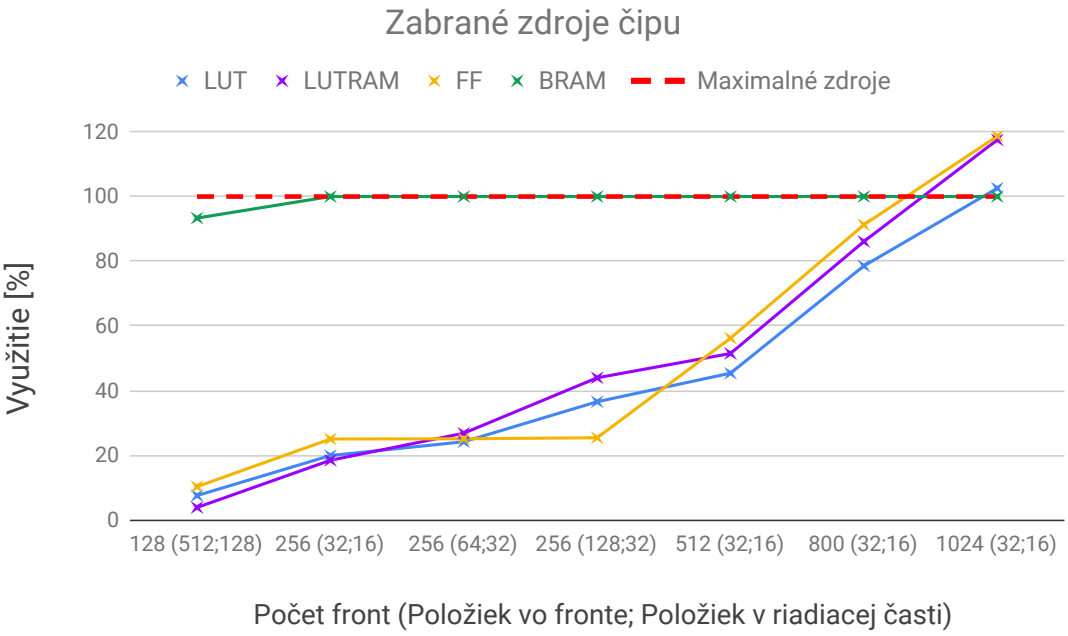
Pomocou zníženia počtu položiek vo frontách sa dá znížiť blokramkové nároky jednotlivých front. Avšak v dátovej časti fronty potrebujú mať dostatočný počet položiek pre uloženie dát pri nastavenej maximálnej prenosovej jednotky (`MTU`).

Počet front	Počet položiek (MEM)	Počet položiek (LIM)	LUT	LUTRAM	FF	BRAM
4	512	128	4018 (0,51%)	664 (0,17%)	6750 (0,43%)	42 (2,92%)
32	512	128	16459 (2,09%)	3976 (1,01%)	41913 (2,66%)	336 (23,34%)
64	512	128	30506 (3,88%)	7720 (1,96%)	82068 (5,21%)	672 (46,67%)
128	512	128	58701 (7,54%)	15040 (3,86%)	161789 (10,3%)	1344 (93,34%)
256	32	16	156925 (19,92%)	72768 (18,45%)	394487 (25,03%)	1440 (100%)
256	64	32	190561 (24,18%)	105792 (26,82%)	395517 (25,1%)	1440 (100%)
256	128	32	288051 (36,55%)	173376 (43,95%)	400647 (25,42%)	1440 (100%)
256	512	128	804140 (102,03%)	595432 (150,92%)	414896 (26,33%)	1440 (100%)
512	32	16	357615 (45,38%)	203072 (51,47%)	885900 (56,21%)	1440 (100%)
512	128	32	850378 (107,9%)	571968 (144,97%)	905972 (57,48%)	1440 (100%)
800	32	16	619166 (78,56%)	339664 (86,09%)	1438415 (91,26%)	1440 (100%)
1024	32	16	808336 (102,56%)	463680 (117,52%)	1870335 (118,66%)	1440 (100%)

Tab. 4.3: Tabuľka výsledkov syntézi obmedzovača priepustností



Obr. 4.9: Graf zabraných zdrojov pri konštantnej veľkosti front



Obr. 4.10: Graf zabraných zdrojov pri rôzných veľkostiach front

Ak dáta sa prenášajú cez `ETHERNET`, kde stanovená `MTU` je obvykle 1500 bajtov, tak je potrebné definovať minimálne 24 položiek u front. To sá dá odvodiť zo vzťahu $\frac{\text{MTU}}{\text{FLU\_WIDTH}}$, kde `FLU_WIDTH` je používaná šírka dátového rozhrania (512). Na grafu 4.10 sú uvedené výstupy syntézi pri rôznych veľkosiach front.

Je evidentný rozdieľ medzi grafmy 4.9 a 4.10 pri generovania 256 front s rôznymi hodnotami položiek. Kým nastavenie s väčšími frontami prečerpalo využitelné zdroje, tak s menšími frontami nebolo vyčerpaných ani poľovina zdrojov. Na základe tohoto princípu je možné si stanoviť z grafu 4.10 maximálny počet generovaných front zhruba na 800 s nastavenými 32 a 16 položkami. Tento odhad je plne teoretické a nereálne. Obmedzovač bez vývojového prostredia `NDK` nie je schopný pracovať. Z tohoto dôvodu pre reálnejšie odhady si musíme prirátať k dosiahnutým výsledkom čerpané zdroje vývojového prostredia. Prázdný framework využíva nasledujúce počty zdrojov.

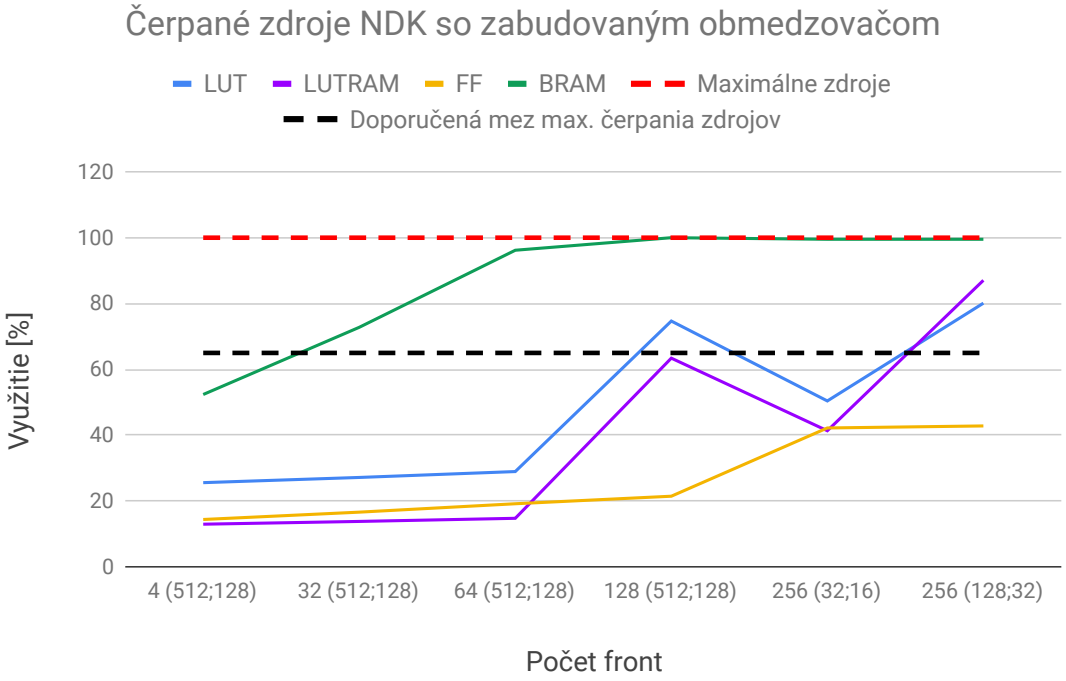
- LUT ako logika - 22,98 %
- LUT ako pamäť - 11,46 %
- registery (FF) - 12,93 %
- blokramky (BRAM) - 48,86 %

Počet front	Počet položiek (MEM)	Počet položiek (LIM)	LUT	LUTRAM	FF	BRAM
4	512	128	201673 (25,59%)	51152 (12,97%)	226884 (14,4%)	754 (52,37%)
32	512	128	214121 (27,17%)	54454 (13,81%)	262053 (16,63%)	1048 (72,78%)
64	512	128	228189 (28,96%)	58198 (14,76%)	302204 (19,18%)	1385 (96,19%)
128	512	128	588523 (74,68%)	250173 (63,41%)	338350 (21,47%)	1440 (100%)
256	32	16	397272 (50,41%)	163174 (41,36%)	665101 (42,2%)	1433 (99,52%)

Tab. 4.4: Tabuľka výsledkov syntézi obmedzovača priepustností zabudovanej do frameworku NDK

Je vidielné rýchlejšie vyčerpanie dostupných blokramiek ako v tabuľke 4.3. Pri generovania 128 front sa využili všetky blokramky a viac než dve tretiny kombinačnej logiky. Zhrnuté výsledky sú získané po syntéze, ktoré sú pravdepodobným odhadom použitého nástroja. Je pravdou, že po následnej implementácie realné využitie zdrojov bude menší. Toto je spôsobené pomocou rôznych optimalizácií počas implementačného procesu.

Neprečerpanie využitých zdrojov nie je jediným z dôležitých faktorov úspešnosti implementácie. Časová analýza udáva splnenie definovaného času medzi všetkými



Obr. 4.11: Graf čerpaných zdrojov NDK so zabudovaným obmedzovačom

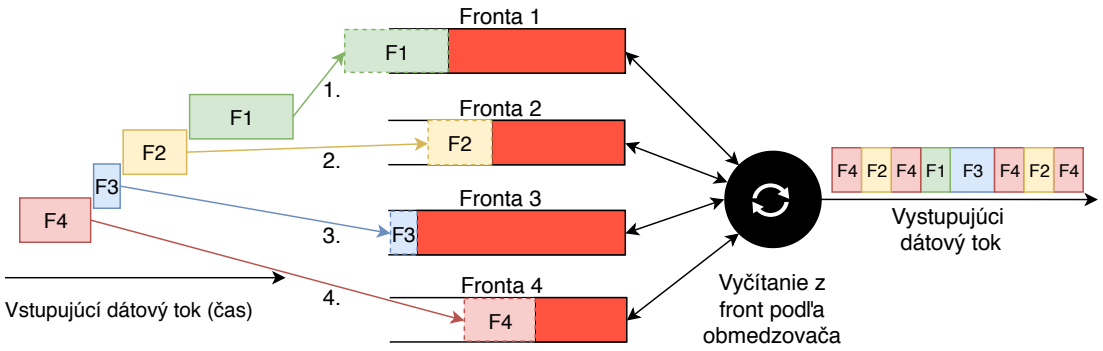
prepojenými zdrojmi. S väčším počtom využívaných zdrojov sa ťažšie splní časová analýza návrhu. Aplikačné jadro vývojové prostredie `NDK`, kam je zabudovaný obmedzovač priepustnosti beží na frekvencie 200 MHz. Z dôvodu dodržania stanovenej rýchlosti v praxi je veľmi ťažké vytvoriť úspešný návrh pri väššej čerpatelnosťi než 65-70 % zdrojov.

Z dosiahnutých výsledkov z tabuľky 4.4 a z grafu 4.11 plynie, že navrhnutá komponenta nie je vhodná pre limitovanie väčšej počty front (radovo tisíce). Maxmimálný možný počet front je približne 256 s 32 a 16 položkami v dátovej a riadiacej časti. Po zvýšenia hodnoty položiek syntézne výsledky už výrazne prekročili doporúčenú mez. Toto spôsobuje komponenta dátovej časti, kde sú uchováva veľké počty dát na FPGA čip. Takýto princíp využitia čipu je veľmi neefektívny a nepraktický. Z dôvodu rýchlej vyčerpanosti blokramiek sa rýchlo vyčerpajú logické buňky. Optimalizácia, ktorá by eliminoval nastávajúci problém, by bolo používanie externej pamäti pre vstupujúce dáta. S tým by klesla blokramková náročnosť navrhu, ale bolo by potrebné upraviť logiku rozdelovania vstupných dát a používané pomocné dáta v riadiacej časti. Realizácia obmedzovača s týmto spôsobom nebola cieľom diplomovej práce.

5 Testovanie obmedzovača na karte NFB-200G2QL

Táto sekcia sa venuje testovaniu návrhu obmedzovača na sieťovej akceleračnej karte `NFB-200G2QL`. Z textu vyššie vyplíva, že návrh v tejto forme nie je vhodná pre implementovanie veľkého počtu front. Z toho dôvodu testovanie sa venovalo hlavne k overeniu funkčnosti a správneho chovania obmedzovača priepustnosti pri menších počtov front. Jednotlivé fronty boli implementované s veľkostou 512 položiek v dátovej a 128 v riadiacej časti.

Zo začiatku navrhnutý komponent bol vytvorený tak, aby vstupujúce dáta neboli zahodené. Pri zachovaní tejto vlastnosti sa prejavil nežiadajúci jav zaplnenia front. Tento prípad je uvedení na obrázku 5.1.



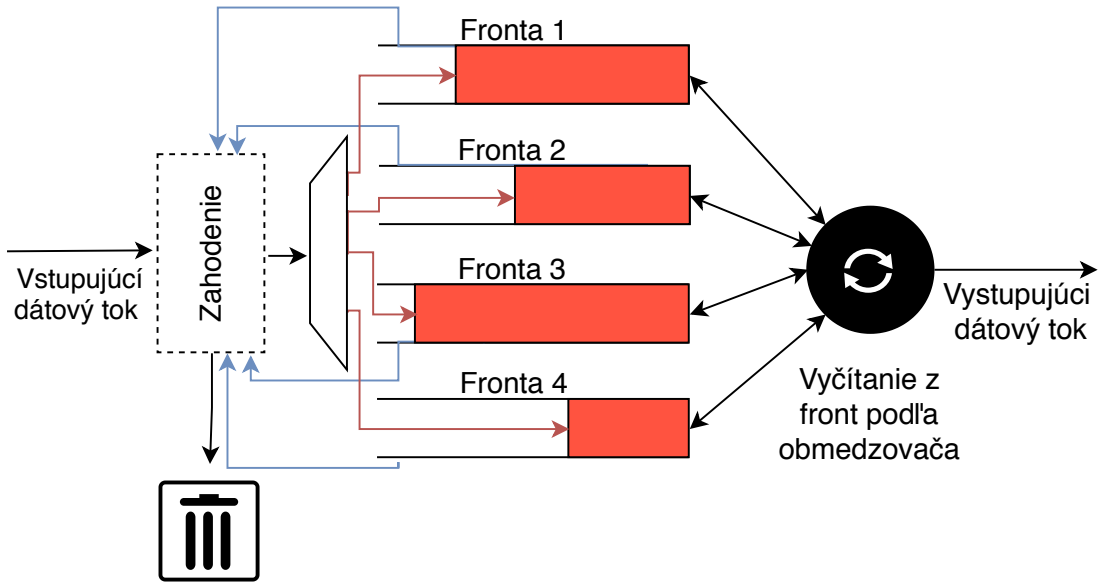
Obr. 5.1: Nežiadúci jav zaplnenia jednej fronty

Vstupujúce dáta z jednotlivých zdrojov prichádzajú ku rozdeľovacej logike s rovnakou pravdepodobnosťou. Pri rôzne nastavených priepustnostiach front, po istom čase fronta s najnižšou priepustnosťou sa zaplní a nedovolí spracovávanie ďalších dát. Vtedy sa nastane prípad, ako je aj znázornené na obrázku 5.1, keď pakety F2, F3 a F4 by sa vliezli do určených front, ale spracovávanie paketu F1 potrvá až do istej doby uvolňenia dostatočnej miesta vo fronte. V tomto prípade zapisovacia rýchlosť do fronty, bude odpovedať vyčítacej rýchlosti. Výsledkom bude nechcené obmedzenie všetkých front na najnižšiu definovanú priepustnosť. Tento jav sa aplikuje vtedy, keď prichádzajúce pakety majú väčšiu rýchlosť než stanovené priepustnosti jednotlivých front.

Z dôvodu eliminovania nežiadaného javu sa pridala zahadzovacia logika na základe jednoduchého mechanizmu aktívnej správy front. Táto logika pracuje podľa princípu Tail Drop[1]. Pred uložením vstupného paketu do určenej fronty sa overí, či existuje dostatočný počet stanovených miest vo fronte. Pri nesplnenia podmienky

[1] Pakety, ktoré sa nezmestia do fronty sú zahodené

paket sa zahodí. Overenie môže byť zrealizované na základ dvoch konfigurácií. Pri prvom sa využíva nakonfigurovaná hodnota cez MI32 rozhranie. Keď počet voľných položiek je menšia než stanovená hodnota, pakety sú zahodené. V druhom prípade sa využíva prepočet dĺžky paketu na potrebný počet položiek. V tomto prípade zahodenie paketov je dynamické. Upravené chovanie návrhu je uvedené na obrázku 5.2.



Obr. 5.2: Eliminovanie zaplnenia fronty pomocou logiky Tail Drop

Reálne testovanie návrhu bolo zrealizované na internom stroji firmy `Netcope Technologies`. Pre rôzné testovacie prípady sa používali dve verzie vygenerovaného obmedzovača priepustnosti. U oboch prípadoch testovanie sa zrealizovalo medzi akceleračnou sieťovou kartou a hostitelským počítačom (DMA kanály).

5.1 Testovanie obmedzovača s dvoma frontami

Prvá verzia obmedzovača generuje dve fronty. Počet položiek front je uvedené na začiatku sekcie. Rozdeľovacia logika navrhnutého obmedzovača už ráta stým, že vstupujúce dáta sú dopredu označkované (napr. číslo fronty). Bez procesu označkovania vstupujúcich dát nie je možné komponentu reálne používať. Z toho dôvodu sa pridala jednoduchá rozdeľovacia logika, ktorá rovnoměrne rozdelí vstupujúce dáta (pakety) do dvoch front. Výberový signál po každých ôsmych paketov si zmení určenú frontu pre zápis. Z toho plynie, že pakety prichádzajú do front s polovičnou rýchlosťou vstupného dátového toku.

Táto verzia obmedzovača slúži pre experimentálne overenie reálnej funkčnosti návrhu a správneho chovania limitovania front.

Pri testovania boli využívané nasledujúce aplikačné nástroje vývojového prostredia `NDK`.

- ndktool bus - Slúži pre konfigurovanie cez MI32
- ndktool fastwrite - Generovanie dát
- ndktool fastread - Vyčítanie dát

V tabuľke 5.1 sú znázornené konfigurácie front pre testovanie prvej verzie obmedzovača. Konfigurácia bola prevedená pomocou nástroja ndktool bus. Vyhradený adresný priestor pre konfiguráciu komponenty je od 0x2000000 do 0x207FFFF. Definované MI32 adresy konfiguračných registerov sú nasledovné.

- Fronta 1
  - `BUCKET LIMIT` : 0x2000004
  - `SPEED` : 0x2000008
  - `TIME CONST` : 0x200000C
- Fronta 2
  - `BUCKET LIMIT` : 0x2000014
  - `SPEED` : 0x2000018
  - `TIME CONST` : 0x200001C

Vysokorýchlostné generovanie dátových tokov bolo prevedené pomocou nástroja ndktool fastwrite. Na základe vstupného parametru sa dáju zvoliť konštantné dĺžky paketov v generovanom dátovom toku. Pre naše testovanie sa využili dátové toky s dĺžkami paketov 64, 128, 300, 500, 1000 a 1500 bajtov.

Test	Číslo fronty	Teoretická priepustnosť [Gb/s]	Konfiguracia	
			Speed	Time constant
1	1.	80	10	500
	2.	20		125
2	1.	40	10	250
	2.	32		200
3	1.	26	10	162
	2.	16		100
4	1.	8	10	50
	2.	1,6		10
5	1.	3,2	10	20
	2.	4		25
6	1.	1,6	10	10
	2.	1,44		9
7	1.	0,32	10	2
	2.	0,48		3

Tab. 5.1: Tabuľka konfigúracie pre prvú verziu obmedzovača

Aby bolo možné overiť priepustnosť jednotlivých front, využili sa drobné úpravy pri spracovávaní vstupných metadát v obmedzovači. Pre pakety vyčítané z prvej fronty sa definovala nultý výstupný DMA kanál. Pre pakety z druhej fronty prvý výstupný DMA kanál. Následne s využívaním nástroja ndktool fastread sa namerala rýchlosť prenosu na použitých DMA kanáloch. Na nasledujúcich tabuľkách 5.2 a 5.3 sú znázornené dosiahnuté výsledky jednotlivých meraní.

Dĺžka paketu	80 [Gb/s]	20 [Gb/s]	40 [Gb/s]	32 [Gb/s]	26 [Gb/s]	16 [Gb/s]
64	6,88	6,87	5,44	5,43	6,87	6,87
128	21,38	21,38	24,33	24,33	17,10	17,11
300	37,11	21,01	42,03	33,625	27,23	16,81
500	49,80	20,90	41,81	33,45	27,09	16,72
1000	50,47	20,82	41,64	33,31	26,98	16,65
1500	50,39	20,79	41,58	33,27	26,94	16,63

Tab. 5.2: Tabuľka výsledkov testov 1, 2 a 3

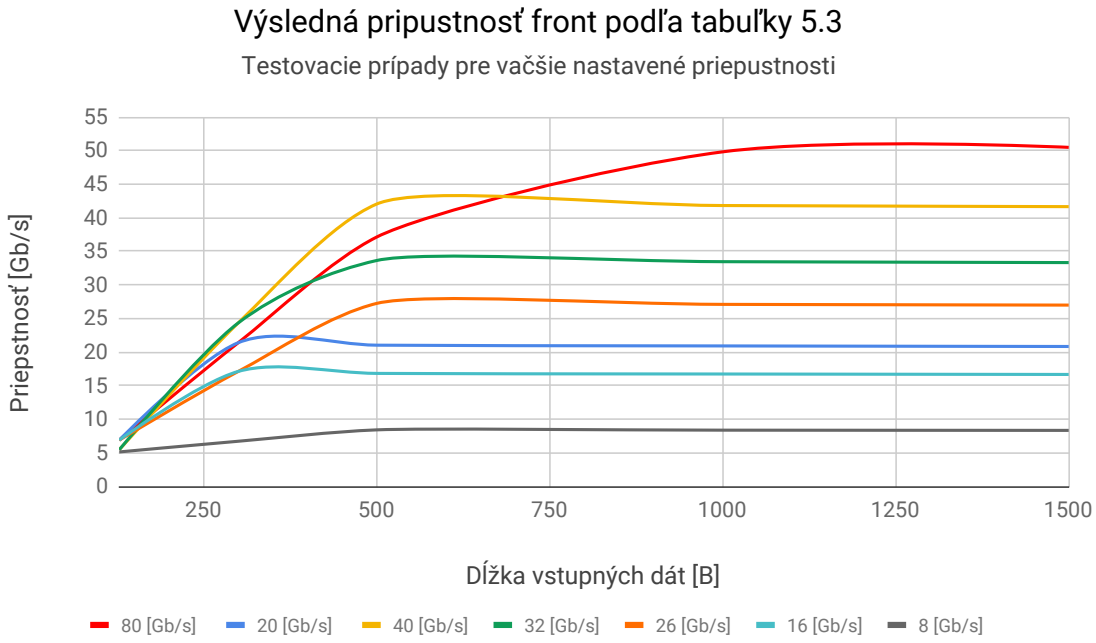
V prípade limitovania fronty na 80 Gb/s vznikla väčšia chyba medzi nameranými a teoretickými hodnotami. Toto avšak spôsobilo nedostačujúca zapisovacia rýchlosť do front. Pri generovania dátového toku s 100 Gb/s, maximálna zapisovacia rýchlosť do front bude 50 Gb/s. Táto vlastnosť tejto verzie bola popisaná v textu vyššie. Ostatné vzniknuté chyby pri krátkych paketoch (napr. 64) sú spôsobené z dôvodu pomalej rýchlosti generátora. Dosiahnuté výsledku sú znázornené na grafu 5.3

Dĺžka paketu	8000 [Mb/s]	1600 [Mb/s]	3200 [Mb/s]	4000 [Mb/s]	1440 [Mb/s]	320 [Mb/s]	480 [Mb/s]
64	5118,62	1247,98	1742,47	2121,64	1146,80	352,58	446,11
128	6711,01	1711,02	3422,09	3421,99	1539,91	342,19	460,21
300	8406,35	1681,28	3362,58	4203,20	1513,15	336,25	504,38
500	8362,17	1672,43	3344,87	4181,08	1505,18	334,48	501,73
1000	8328,95	1665,82	3331,56	4164,45	1499,22	333,15	499,74
1500	8317,89	1663,58	3327,13	4158,97	1497,21	332,71	499,07

Tab. 5.3: Tabuľka výsledkov testov 4, 5, 6 a 7

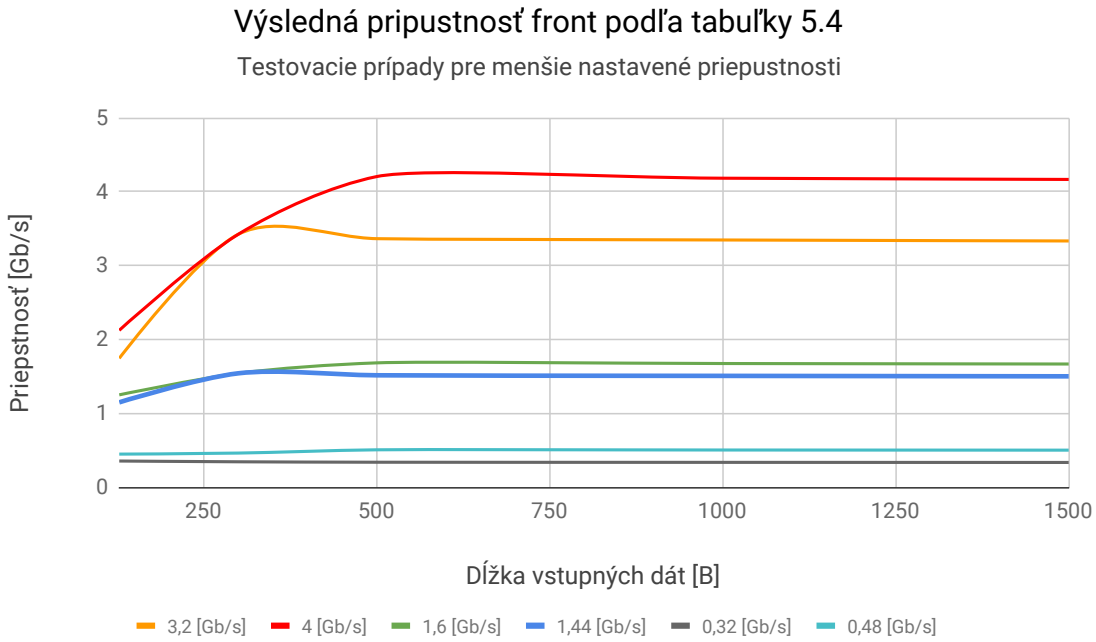
Vzniknuté chyby pri krátkych paketoch u menších nastavených priepustnostiach sú spôsobené s drobnou neefektivitou jádra obmedzovača. Vypočet algoritmu jádra trvá 5 hodinových taktov, kým prenos 64 bajtového paketu trvá jeden takt.

Z dosiahnutých výsledkov vyplýva, že obmedzovač priepustnosti pri konštantne



Obr. 5.3: Graf výsledkov testov 1, 2 a 3

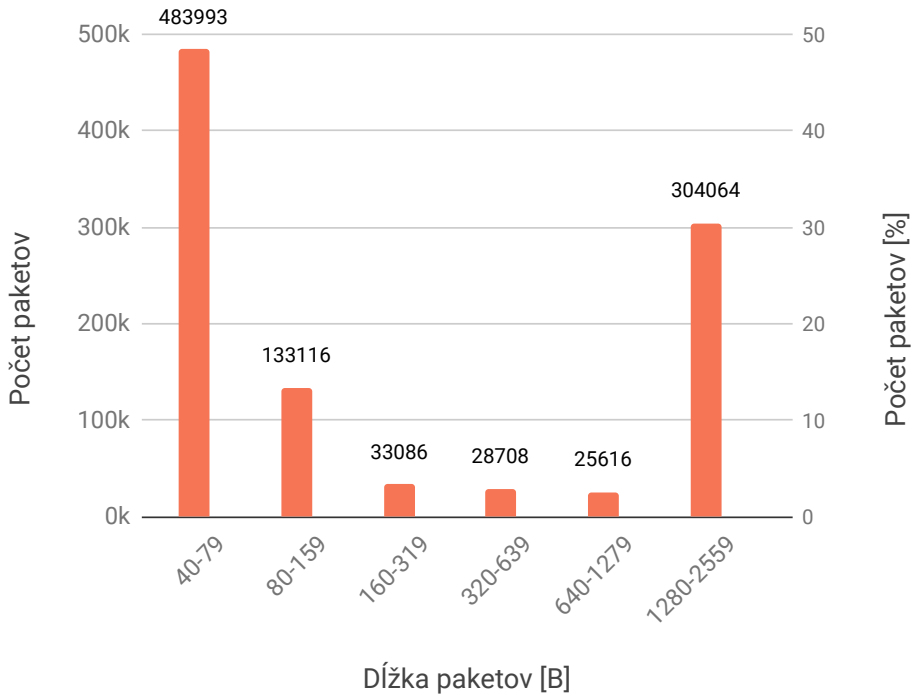
dlhých paketoch pracuje správne, avšak s istou mierou menšej odhýlky. Na grafu 5.4 sú znázornené namerané výsledky z tabuľky 5.3.



Obr. 5.4: Graf výsledkov testov 4, 5, 6 a 7

5.2 Testovanie obmedzovača so štyrmi frontami

V doterajších testoch sa využívali pakety s konštantnou veľkosťou. Pre vytvorenie realnej simulácie dátových tokov v sieti, použil sa zachytení provoz z PCAPu (Packet Capture) `fdump.pcap`. Tento súbor sa skladá z 1008583 paketov. Na nasledujúcom obrázku je znázornený histogram paketových dĺžok vyskytujúce v PCAPu.



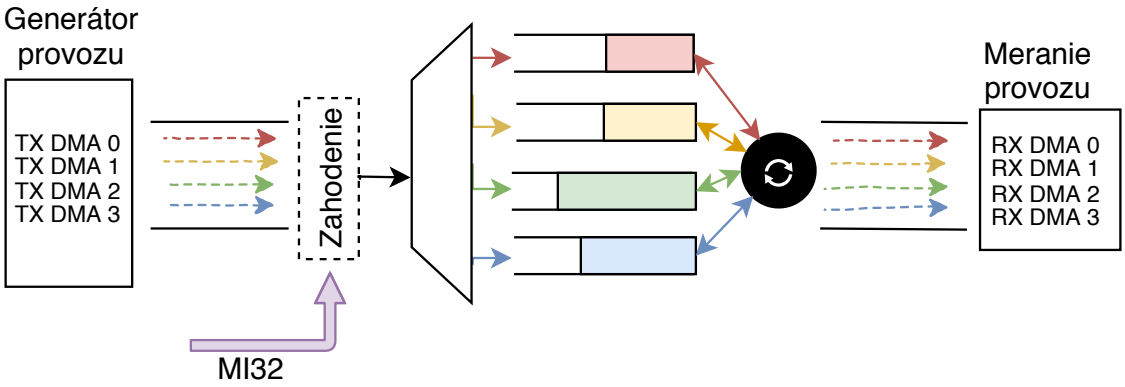
Obr. 5.5: Histogram testovacieho pcapu

Aby každý vyskytujúci paket sa dostal do každej fronty, spravili sa drobné úpravy naproti predošlej používanej verzie. Zmenila sa rozdelovacia, zahadzovacia logika a počet generovaných front. Samotné rozdelenie je prevedené na základe vstupújucého rozhrania paketu. Toto nám určuje položka v používaných metadátach. Dáta z nultého DMA kanála sú uložené do prvej fronty, z prvého DMA kanála do druhej fronty atď. Obdobným spôsobom sa zachadzuje s paketmy pri vyčítania. Počet používaných front je 4. Pre odstránenie zaplnenia jednotlivých front sa implementovali oba vyššie popísané spôsoby zahadzovania. Medzi jednotlivými spôsobmy sa dá prepínať na základe konfigurácie cez MI32. Zo 32 bitového slova rozhrania MI32 sú využívané 16 bity s najvyššou váhou (MSB - Most Significant Bit) a dva bity s najmenšou (LSB - Least Significant Bit).

- LSB 0
  - log. 1 : zapne sa zahadzovanie paketov
  - log. 0 : vypne sa zahadzovanie paketov

- LSB 1
  - log. 1 : overovanie zahadzovania na základe definovanej hodnoty
  - log. 0 : overovanie zahadzovania na základe dĺžky paketu
- MSB 16 bitov
  - definovaná hodnota pre zahadzovanie

Konfigurácia je prístupná na MI32 adrese 0x2000010. Výsledná schéma vytvoreného systému testovania je uvedené na obrázku 5.6.



Obr. 5.6: Schéma vytvoreného systému s obmedzovačom

Pred samotným testovaním konfigurácia sa správala rovnako ako u predchádzajúcej verzie. Konfiguračné registery nových implementovaných front sa nachádzajú na adresách 200002X a 200003X. Pri prvých troch testoch sa predviedli konfigurácie na základe hodnôt z tabuľky 5.4. (R_t - teoretická priep., R_m - nameraná priep.)

Test	Číslo fronty	R_t [Gb/s]	Konfiguracia		R_m [Gb/s]	Chyba [%]
			Speed	Time constant		
1	1.	8	10	50	8,13	1,65
	2.	4		25	4,18	4,75
	3.	2		12	2,01	0,9
	4.	1		6	1,01	1,3
2	1.	8	50	250	8,17	2,22
	2.	4		125	4,19	4,8
	3.	2		62	2,08	4,3
	4.	1		32	1,04	4,8
3	1.	2	10	6	2,021	1,05
	2.	2		6	2,021	1,05
	3.	2		6	2,022	1,1
	4.	2		6	2,023	1,15

Tab. 5.4: Tabuľka používaných konfigúracií a dosiahnutých výsledkov

Generovanie dátových tokov prebiehala pomocou nástroja ndp-tool transmit, ktorá je schopna paralelne posieľať obsah pcapu na viacerých DMA kanáloch. Nastroj

	Packets	Bytes	L1 Mbps	L2 Mbps	PCIe Mbps
Channel 0:	2400782	1290064718	10759	10376	10529
Channel 1:	2300750	1236975394	10316	9949	10096
Channel 2:	2400782	1290064718	10759	10376	10529
Channel 3:	2400782	1289646970	10756	10373	10526
Total:	9503096	5106751800	42593	41075	41682

Obr. 5.7: Snímka obrazovky nástroja ndp-tool transmit pri testu 1

ukazuje aktuálny počet preposieľaných paketov i bajtov a dosiahnuté rýchosti na rôzných vrstvách. Pri prvých troch testovacích prípadoch sa požívala zahadzovanie na základe dĺžky paketu. Pre meranie rýchlosti jednotlivých výstupných DMA kanálov sa použil nástroj ndp-tool read. Tento nástroj ukazuje rovnaké hodnoty ako ndp-tool transmit, iba na výstupných DMA kanáloch. Z tabuľky 5.4 je viditelné, že

	Packets	Bytes	L1 Mbps	L2 Mbps	PCIe Mbps
Channel 0:	2094269	1011214044	8466	8132	8265
Channel 1:	1417214	519604939	4415	4189	4280
Channel 2:	902587	249460037	2162	2018	2076
Channel 3:	576300	124758354	1105	1013	1050
Total:	4990370	1905037374	16150	15353	15672

Obr. 5.8: Snímka obrazovky nástroja ndp-tool read pri testu 1

vzniknuté odchýlky medzi nameranými a teoretickými hodnotami nepresiahajú 5%. Relatívne najväčšie odchýlky vznikli, v prípade používania väčších hodnôt konfiguračných registrov.

Doterajšie testy sa plne venovali k overovaniu nastavených priepustností. V nasledujúcej časti textu budú porovnané vlastnosti implementovaných metód zahadzovania a jejich vplyv na výsledné obmedzovanie. Podľa hodnôt tabuľky 5.5 sa definovali priepustnosti jednotlivých front na 1000, 800, 600 a 400 Mb/s. Pri prvom testu sa používala metóda zahadzovania na základe dĺžky vstupujúceho paketu. Dosiahnuté výsledky sú zhodné s teoretickými nastaveniami. Kým pri druhom móde zahadzovania, priepustnosť front je výrazne menšia než stanovený limit.

Pre podrobnejšiu analýzu, do každej jednej fronty sa poslal obsah testovacieho pcapu `fdump` iba raz. Následne vystupujúce dátové toky z jednotlivých front boli zachytené do PCAPov. S týmto spôsobom sa overili typy a dĺžky vstupujúcich paketov do jednotlivých front. Dosiahnuté výsledky sú zapísané do tabuliek 5.6, 5.7 na základe rozloženia paketov podľa dĺžky.

Číslo fronty	R_t [Gb/s]	Konfiguracia		Mód zahodenia		R_m [Gb/s]
		Speed	Time constant	Dĺžka	Konštanta	
1.	1	10	6	√	×	1,11
2.	0,8		5			0,81
3.	0,6		3			0,606
4.	0,4		2			0,405
1.	1	50	31	×	√	0,656
2.	0,8		25			0,470
3.	0,6		18			0,366
4.	0,4		12			0,29

Tab. 5.5: Tabuľka konfigúracie pre prvú verziu obmedzovača

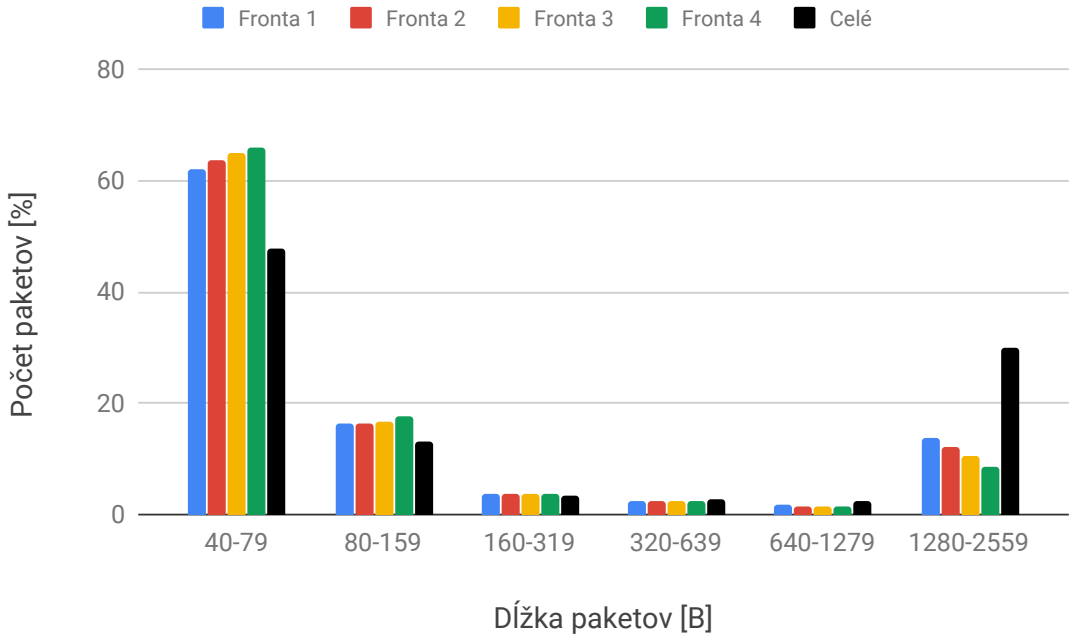
Dĺžky paketov	Fronta 1		Fronta 2		Fronta 3		Fronta 4	
	Počet	[%]	Počet	[%]	Počet	[%]	Počet	[%]
40-79	192406	62,14	171759	63,69	144021	65,01	108793	66,17
80-159	50433	16,28	44440	16,48	37319	16,84	28888	17,57
160-319	11259	3,63	9668	3,58	8154	3,68	6215	3,78
320-639	7910	2,55	6803	2,52	5461	2,46	4092	2,48
640-1279	5328	1,72	4323	1,60	3369	1,52	2335	1,42
1280-2559	42266	13,65	32661	12,11	23210	10,47	14081	8,56
Celkom	309602	-	269654	-	221534	-	164404	-

Tab. 5.6: Tabuľka výstupných paketov z front pri zahodenia na základe dĺžke

Dĺžky paketov	Fronta 1		Fronta 2		Fronta 3		Fronta 4	
	Počet	[%]	Počet	[%]	Počet	[%]	Počet	[%]
40-79	8862	46,53	8010	46,08	6268	46,5	5271	46,24
80-159	2437	12,8	2238	12,88	1722	12,77	1533	13,45
160-319	550	2,89	534	3,07	383	2,84	322	2,82
320-639	527	2,77	482	2,77	404	3	318	2,79
640-1279	483	2,54	417	2,4	326	2,42	277	2,43
1280-2559	6185	32,48	5701	32,8	4378	32,48	3678	32,27
Celkom	19044	-	17382	-	13481	-	11399	-

Tab. 5.7: Tabuľka výstupných paketov z front pri zahodenia na základe konštanty

Z tabuľky 5.6 sa dá zistiť, že limitované priepustnosti sú dosiahnuté, avšak na úkor vzniknutých parazitných vlastnosti. Po dlhšom čase krátke pakety su uprednostnené pred veľkými paketmi.



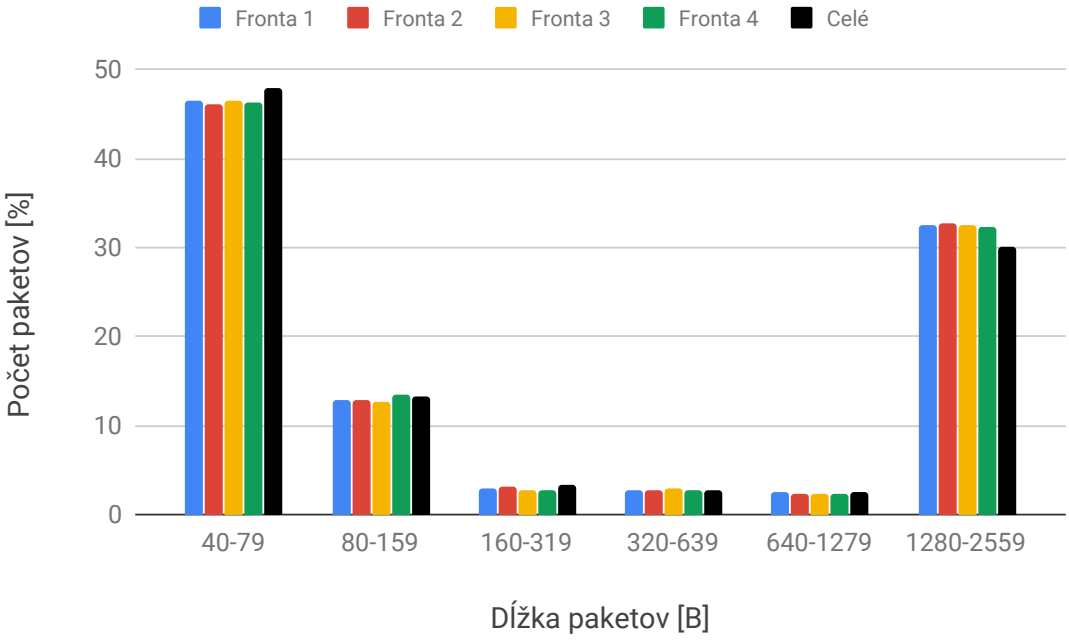
Obr. 5.9: Histogram vystupujúcich paketov z front podľa 5.6

Táto vlastnosť je znázornená na histogramu 5.9. Na nižších priepustnostiach sa zvyšuje počet prenesených malých paketov na úkor veľkých. Napr. pri limitovaní dátových tokov na 8 Gb/s, 62 % reálne limitovaných paketov má dĺžku v rozmezdzí 49 až 80 bajtov. Kým takéto dlhé pakety tvoria iba 48 % vysielaného PCAPu. Táto vlastnosť je istou neefektivitou tejto metódy zahadzovania. V prípade, keď dátové toky viacerých účastníkov sú napojoné do jednej fronty, jeden z učastníkov na základe znalosti tohoto javu si môže využiť celú povolenú priepustnosť ibe preseba. To s tým spôsobom, že svoje veľké dátové pakety si fragmentuje na mešie časti. Avšak pre eliminovanie zaplnenia pri extremných prípadoch zvyšenia sa vstupnej prenosovej rýchlosti dátových tokov je efektívna metoda.

Mód zahadzovania na základe definovanej hodnoty eliminuje tento problém. Nastavením tejto hodnoty na `MTU`, je docielená rovnaká pravdepodobnosť výskytu paketov vo frontách ako v prichádzajúcich dátových tokoch. Pri našom testovaní PCAP `fdump` obsahovala maximálne 1518 bajtové pakety, z toho dôvodu sme si definovali konštantnú hodnotu na 24. Táto hodnota sa vypočítala na základe nasledujúcej rovnice.

$$\text{Definovaná hodnota} = \frac{\text{Najväčšia vyskytujúca dĺžka}}{\text{Veľkosť položky}}$$

Na histogramu 5.10 sú znázornené dosiahnuté výsledky. Pri jednotlivých frontách rozloženie limitovaných paketov sa zhoduje s menšími odchýlkami s rozložením re-

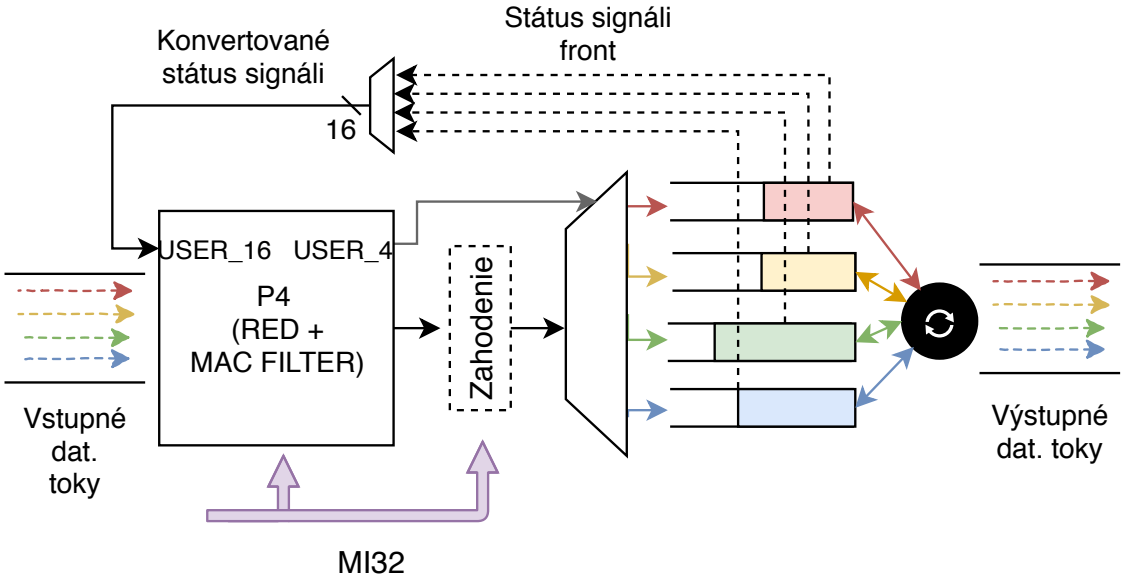


Obr. 5.10: Histogram vystupujúcich paketov z front podľa 5.7

álne vysielaných paketov. Eliminovaním parazitných javov sa enormne zvýšila zahadzovanie paketov a výsledné priepustnosti boli výrazne menšie než stanovený limit. Tieto metódy sú jednoduché na implementáciu, ale vzťahujú sa k nim menšie neefektivíty, na základe ktorých pri realnej využitia týchto metód potrebujeme používať aj pokrečilejší mechanizmus aktívnej správy front. V nasledujúcej kapitole je popísaná kombinácia obmedzovača front s mechanizmom Random Early Detection (RED), ktorý je pokročilejší algoritmus triedy mechanizmov aktívnej správy front.

6 Systém obmedzovača viacerých front kombinovaná s mechanizmom aktívnou správou

V tejto kapitole je popísaný vytvorený systém tvarovania dátoveho toku s aktívnou správou front pomocou mechanizmu RED (Random Early Detection). Obecné schéma vytvoreného systému je znázornené na obrazku 6.1.



Obr. 6.1: Obecné schéma vytvoreného systému

Novým prvkom systému je komponenta vygenerovaná z jazyka P4 (Programming Protocol-independent Packet Processors).

P4 je vysokoúrovňový jazyk pre programovanie protokolovo nezávislých sieťových procesorov. Jeho hlavnou úlohou je možnosť volne definovať spracovanie paketov v sieťových zariadeniach. Z dôvodu jednoduchosti tohto jazyka zdrojový kód je môžne kompilovať pre ľubovolné sieťové zariadenie. Firma `Netcope Technologies` vyvinula svôj vlastný kompilátor P4 jazyka na VHDL. S tým je dosiahnuté využitie P4-kých aplikácií na platforme FPGA. Pomocou nahratých pravidieľ je môžne nastaviť chovanie komponenty.

Používaná komponenta P4 (RED + MAC FILTER) je výstupom inej diplomovej práce a z toho dôvodu nie je popísaná detailnejší postup implementácie. Avšak pre porozumenie vytvoreného systému je nevyhnutné zmieniť o vstupných a výstupných parametroch tejto komponenty. Vytvorený P4-kový komponent používa rovnaké rozhranie ako navrhnutý obmedzovač front. Vstupujú do nej pomocné metadáta a `FLU` dátové rozhranie. Pre externú komunikáciu s jádrom komponenta je možné využiť

položky `USER_16` (16 bitové pole) a `USER_4` (4 bitové pole) v metadát.

V našom prípade sa využivá položka `USER_16` pre vstupujúce informácie do jadra a `USER_4` pre vystupujúce informácie z jadra. Aby mechanizmus RED mohol fungovať potrebuje vedieť úroveň zaplnenia jednotlivých front 2.4.1. Pri návrhu obmedzovača sa vytvoril výstupný port pre státus signál jednotlivých front. Tento vystup určuje počet voľných položiek vo fronte. Veľkosť státus signála záleží na počte položiek fronty. Pri využití väčších front nezmestia sa všetky státus signáli do 16 bitového vektora `USER_16` a z toho dôvodu je potrebné všetky státus signáli front prekonvertovať do podoby 16 bitového vektora.

Výpis 6.1: Pseudokód konvertovacej logiky pre státus signál jednej fronty

```
// n je počet bitov pre vyjedrenie úrovní
n = 16 / pocet_front;
// pocet roznych úrovní
k = 2^n;
// s je počet stavov vyjadrených s jedným úrovňom
s = pocet_poloziek / k;
//
if (status_fronty mod s = 0){
    conv_status = status_fronty div s;
else
    conv_status = (status_fronty div s) + 1;
}
```

Pri využívaní štyroch front, šírka signála vyjadrujúci úroveň zaplnenia sú štyri bity. Pri tomto nastevení sme schopný reprezentovať 16 rôzných úrovní zaplnenia. Každý jeden úroveň následne reprezentuje $\frac{\text{Počet položiek}}{\text{Počet úrovní}}$ stavov. Pri implementovania 512 položiek do front, jednotlivé úrovne reprezentujú nasledujúce skupiny stavov.

Úroveň	Počet zapĺneních položiek	
(4 bity)	Minimálni	Maximálni
1111	0	31
1110	32	63
1110	64	95
⋮	⋮	⋮
0001	448	479
0000	480	512

Tab. 6.1: Tabuľka vyjadreních jednotlivých stavov

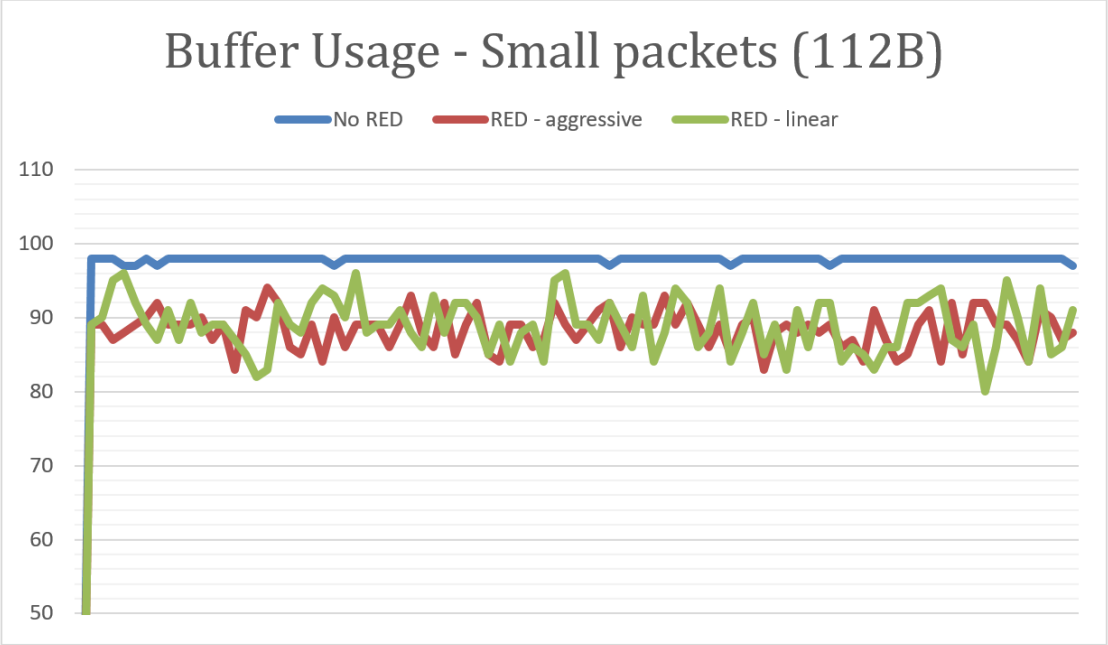
Štyri bity s najmenšou váhou patria k prvej fronte, následujúce štyri bity pre druhú

frontu atď. Komponenta vygenerovaná z jazyka P4 paralelne počas aktívnej správy front aj označkováva pakety. Označkovanie je prevedené na základe zdrojových MAC adries paketu. Pri konfigurácie jadra sa nahrajú aj pravidlá určujúce pakety s danou MAC adresou do nastavenej fronty. Vystupujúce pakety pri validnom začiatku dátového slova majú určené poradie fronty v položke `USER_4` v metadát. Na základe tejto hodnoty je prevedená selekcia obmedzovačom.

6.1 Výsledky

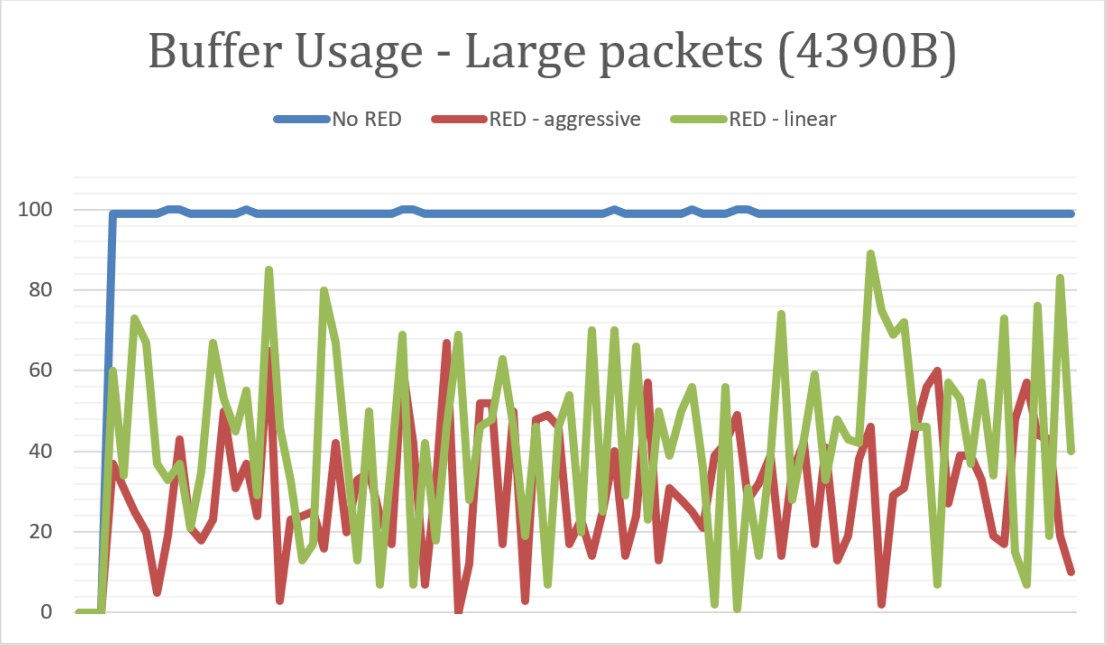
Práca v kapitole 5 sa zamerala hlavne na overovanie funkčnosti navrhnutého obmedzovača priepustností, pri využití dvoch jednoduchých mechanizmoch zahodenia paketov. Z toho dôvodu v tejto časti práce sa zameriavame na vyhodnotenie systému na základe funkčnosti používanej komponenty z diplomovej práce [11].

Táto komponenta je vygenerovaná z jazyka P4 a pracuje na základe pokročilejšieho mechanizmu RED (2.4.1). Na základe informácií o zaplnenia fronty zahadzuje na svojom vstupe prichádzajúce pakety náhodne. Aggresivitu zahodenia je možné definovať pomocou nahratých pravidieľ. Na obrazoch 6.2 a 6.3 sú znázornené využitie fronty pri aplikovania rôznych pravidieľ.



Obr. 6.2: Využitie fronty pri malých paketoch [11]

Z grafu 6.2 plynie, že pri využívania mechanizmu RED fronta sa nezaplní. Komponenta udržuje stav fronty, aby sa neprejavili nežiadajúce javy, ktoré boli popísané v kapitole 5. Pri aggresívnejších pravidlách je viditelné presnejšie udržovanie stavu.



Obr. 6.3: Využitie fronty pri veľkých paketoch [11]

fronty. Z grafu 6.3 je vidno, že pri väčších paketoch úroveň zaplnenia fronty je udržovaná na menších percentuálnych hodnotách.

Ako aj z teórie plyne, náhodné zahadzovanie paketov nespôsobuje výrazný pokles kvality služby. Naopak zahodenie väčšej skupiny posebeidúcich paketov kvalitu výrazne zníži. Z toho dôvodu navrhnutý systém, kde mechanizmus RED udržuje aktívnu správu front, navrhnutý obmedzovač môže limitovať priepustnosti front bez vzniknutých nežiadajúcich javov.

7 Záver

Cieľom teoretickej časti diplomovej práce bolo oboznámenia sa s problematikou kvality služby (Quality of Service) a riadenia paketových front. Z toho dôvodu v teoretickej časti práce boli najprv preštudované spôsoby zabezpečenia kvality služby a princípi použitelných mechanizmov. Zo začiatku teoretický rozbor bol venovaný paketovým frontám Priority Queue, Weighted Round Robin a Deficit Round Robin. Následne sa vysvetlil princíp mechanizmov aktívnej správy front Random Early Detection a Weighted Random Early Detection. Teoretická časť zaoberajúca s problematikou kvality služby sa skončila popisom mechanizmov používaných pre tvarovanie dátových tokov (Leaky Bucket, Token Bucket).

Cieľom praktickej časti diplomovej práce bolo navrhnúť a implementovať spôsob riadenia front pre nasadenie do sieťovej akceleračnej karty NFB-200G2QL vyvynutej spoločnostou Netcope Technologies. Aby praktická časť mohla byť zrealizovatelná, bolo potrebné sa oboznámiť s hardvér popisujúcim jazykom VHDL, s FPGA čipmy a s vývojovým prostredím Netcope Development Kit.

Sieťová karta s FPGA čipom je schopná spracovávať a vysielať dáta s 200 Gb/s. Kvôli dosiahnutelným rýchlostiam sú tieto typy kariet s FPGA čipmi čoraz frekventovanejšie využivané pri sieťových aplikáciach. Praktická časť práce sa venovala vytvoreniu návrhu v jazyku VHDL, pre obmedzovanie priepustnosi viacerých front. Výsledná komponenta je schopná paralelne limitovať priepustnosťi front na základe definovanej konfigurácie. Samotná rekonfigurácia je realizovatelná aj behom spracovávania dátových tokov.

Začiatočné simulácie boli prevedené pomocou simulačného nástroja ModelSim. Po odladení vzniknutých chýb sa overili maximálne zdrojové nároky vytvorenej komponenty. Z dosiahnutých výsledkov plynie, že maximálny počet limitovateľných front je zhruba 256 s menším počtom použitých položiek. Jednotlivé paketové fronty rýchlo vyčerpajú pamäťové zdroje čipu. S využitím externých pamäti by sa dalo eliminovať tento problém. Avšak toto riešenie bolo nadrámec zadania diplomovej práce. Následne sa testovala a overovala funkčnosť návrhu na sieťovej akceleračnej karte.

Prvé testy zo sekcie 5.1 boli prevedené nad dvoma frontami pri rôzných definovaných prenosových rýchlostiach. Tieto testy sa zameriavali na správne dosiahnuté obmedzené priepustnosti. Generovali sa pakety s konštantnými dĺžkami. Najpresnejšie výsledky boli získané pri malých priepustnostiach. Maximálne odchýlky z dôvodu nepresnosti návrhu nepresahovali zhruba 8-10%.

V sekcie 5.2 boli otestované návrhy so štyrymi frontami. Využívali sa pakety replikujúce reálne chovanie siete. Z dôvodu eliminovania nežiadajúceho javu zaplnenia fronty, sa implementovali dva typy zahadzovacieho mechanizmu. Zahodenie paketu na vstupe fronty v prípade, že sa paket nezmesti do fronty, a zahodenie pa-

ketu v prípade, keď vo fronte nie dostačujúci počet miest pre definovanú konštantnú hodnotu. Pomocou prvej metódy sa výrazne zvýšila presnosť nastavených priepustnosti. Dosiahnuté odchýlky nepresiahali 5%. Avšak tento typ zahadzovania po istom čase prenosu preferovala prepustenie malých paketov voči veľkým. Druhá metóda zahadzovania eliminovala tento problém. V poslednej kapitole sa vytvoril systém kombinovania obmedzovania priepsutnosti s mechanizmom aktívnej správy front (RED [11]). Tento typ systému je pre praktické využitie najefektívnejšie z uvedených metód. Znázornené jednoduché mechanizmy zahadzovania môžu byť efektívne pri nastania extrémnych podmienok.

Literatúra

[1] WOLF, Wayne. *FPGA based system design*. New Jersey: Prentice-Hall, 2004, 530 s. ISBN 0-13-142461-0.

[2] PINKER, Jiří, Martin POUPA. *Číslicové systémy a jazyk VHDL*. 1. vyd. Praha: BEN - technická literatura, 2006, 349 s. ISBN 80-7300-198-5.

[3] KOTON, J. *Moderní síťové technologie*. 2014. s. 1-191. ISBN: 978-80-214-5026-4.

[4] SHREEDHAR, M. a G. VARGHESE. *Efficient fair queuing using deficit round-robin*. IEEE/ACM Transactions on Networking. 4(3), 375-385. DOI: 10.1109/90.502236. ISSN 10636692. Dostupné z URL: <`http://ieeexplore.ieee.org/document/502236/`>

[5] WANG, J. a Y. LEVY. *Managing performance using weighted round-robin*. Proceedings ISCC 2000. Fifth IEEE Symposium on Computers and Communications. IEEE Comput. Soc, 2000, 4(3), 785-792. DOI: 10.1109/ISCC.2000.860739. ISBN 0-7695-0722-0. ISSN 10636692. Dostupné z URL: <`http://ieeexplore.ieee.org/document/860739/`>

[6] QoS Cisco Documentation. [online], 2018 [cit. 2015-13-10]. Dostupné z URL: <`http://docwiki.cisco.com/wiki/Quality_of_Service_Networking`>

[7] Cesnet Archív. [online], 2018 [cit. 2015-13-10]. Dostupné z URL: <`http://archiv.cesnet.cz/doc/techzpravy/2000-6/diffserv.html`

[8] KUBÍČEK, M. *Úvod do problematiky obvodů FPGA pro integrovanou výuku VUT a VŠB- TUO*. Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, Technická 12, 612 00 Brno: Vysoké učení technické v Brně, 2014. s. 1-120.

[9] NETCOPE TECHNOLOGIES a.s. *Netcope Development Kit: Firmware developers manual*, 2018. 51 stran

[10] TORASSA, Sergio. *IETF – MEF Token Bucket Comparison*. 2016, 05(01). DOI: 10.4172/2167-0919.1000128. ISSN 21670919. Dostupné z URL: <`http://www.omicsgroup.org/journals/ietf--mef-token-bucket-comparison-2167-0919-1000128.php?aid=70118`

[11] ELIÁŠ, Richard. *Rate limiting and traffic shaping*. Brno, 2019. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce prof. Ing. Václav Přenosil, CSc.

[12] XILINX, Inc *Vivado Design Suite User Guide: Logic Simulation*, 2018. 252 stran

[13] XILINX, Inc *Vivado Design Suite User Guide: Synthesis*, 2018. 285 stran

[14] XILINX, Inc *Vivado Design Suite User Guide: Implementation*, 2018. 185 stran

[15] Mentor Graphics, Siemens Group *ModelSim web page*, Dostupné z URL: <`https://www.mentor.com/company/higher_ed/modelsim-student-edition`

Zoznam symbolov, veličín a skratiek

QoS	quality of service
VoIP	Voice over IP
IntServ	Integrated Services
DiffServ	Differentiated services
RSVP	Resource reservation protocol
IP	Internet Protocol
PQ	Priority Queuing
WRR	Weighted Round Robin
DRR	Deficit Round Robin
FIFO	First In First Out
SP	Strict Priority
RED	Random Early Detection
WRED	Weighted Random Early Detection
VHDL	Very High Speed Integrated Circuit Hardware Description Language
FPGA	Field Programmable Gate Array
NDK	Netcope Development Kit
RTL	Register-transfer level
CLB	Configurable logical block
LUT	Look Up Table
LUTRAM	Look Up Table as Random Access Memory
BRAM	Block Random Access Memory
FLU	Frame Link Unaligned
FIFONUM	Počet inštancí front
ITEMS	Počet položiek vo fronte
SIGWIDTH	súčet dlžky dát, metadát a FLU signálov
MUX	multiplexor
DEMUX	demultiplexor
SELWRFIFO	výber fronty pri zápise
SELRDFIFO	výber fronty pri vyčitania
SRCRDY	source ready
SOP	start of position
ETH	Ethernet
DMA	Direct Memory Access
MTU	Maximum transmission unit
PCAP	Packet Capture
MSB	Most Significant Bit
LSB	Least Significant Bit

Zoznam príloh

A Obsah přiloženého CD

Z dôvodu firemnej tajomnosti nie sú zverejnené snami nevytvorené komponenty. V jednotlivých súboroch `Modules.tcl` sú vyditelné všetky potrebné komponenty pre funkčnost návrhu.

- `/CD_Németh` ..... Koreňový adresár
  - `readme.txt`
  - `sim` ..... testovací komponent celého návrhu
    - `testbench.vhd`
  - `src` ..... používané zdrojové súbory
    - `top` ..... složka hlavných zdrojových súborov
      - `n_limiter_top_v1.vhd` ..... verzia používaná v kapitole 5.1
      - `n_limiter_top_v2-1.vhd` ..... prvá verzia používaná v kapitole 5.2
      - `n_limiter_top_v2-2.vhd` ..... druhá verzia používaná v kapitole 5.2
      - `n_limiter_top_v3.vhd` ..... používaná v kapitole 6
      - `Modules.tcl`
    - `comp` ..... složka zdrojákov dátovej časti
      - `n_fifo`
        - `sim`
          - `testbench.vhd`
        - `n_fifo_aligned.vhd`
        - `Modules.tcl`
    - `limiter` ..... složka zdrojákov riadiacej časti
      - `sim`
        - `testbench.vhd`
      - `limiter_meta.vhd`
      - `Modules.tcl`
  - `firmwares` ..... vygenerované firmvéry jednotlivých verzí
    - `nfb_200g2ql_v1.bit`
    - `nfb_200g2ql_v2-1.bit`
    - `nfb_200g2ql_v2-2.bit`
    - `nfb_200g2ql_v3.bit`