VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF MICROELECTRONICS

AKCELERACE IDENTIFIKACE HTTP HLAVIČEK V OBVODECH FPGA

ACCELERATION UNIT FOR HTTP HEADERS IDENTIFICATION IN FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — IVAN BRYNDZA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARIÁN PRISTACH
SUPERVISOR

KONZULTANT — Ing. VIKTOR PUŠ, Ph.D.
CONSULTANT

BRNO 2015

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav mikroelektroniky

Bakalářská práce

bakalářský studijní obor
Mikroelektronika a technologie

Student: Ivan Bryndza
Ročník: 3

ID: 154684
Akademický rok: 2014/15

NÁZEV TÉMATU:

Akcelerace identifikace HTTP hlaviček v obvodech FPGA

POKYNY PRO VYPRACOVÁNÍ:

V rámci bakalářské práce se seznamte s protokolem HTTP, akcelerační kartou COMBO a systémem HANIC vyvíjeným v rámci projektu Liberouter. Navrhněte architekturu hardwarové jednotky pro identifikaci HTTP hlaviček v paketech pro obvody FPGA. V rámci práce proveďte implementaci navržené jednotky a ověřte implementaci v simulaci a syntézu pro FPGA. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování práce.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 10. 2. 2015

Termín odevzdání: 4.6.2015

Vedoucí práce: Ing. Marián Pristach
Konzultanti bakalářské práce:

doc. Ing. Jiří Háze, Ph.D.
předseda oborové rady

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta elektrotechniky a komunikačních technologií Ústav mikroelektroniky

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Abstrakt

Táto bakalárska práca sa zaoberá hardvérovou akceleráciou identifikácie hlavičiek HTTP protokolu, ktorý je na internete veľmi rozšírený. Cieľom je navrhnúť a implementovať hardvérovú architektúru, ktorá bude slúžiť na detekciu prítomnosti HTTP protokolu v pakete a bude dosahovať priepustnosť potrebnú k monitorovaniu na 100-gigabitových sieťach. V architektúre bol využitý nedeterministický stavový automat a vysoký stupeň paralelizmu na detekciu regulárnych výrazov.

Abstract

The bachelor thesis deals with hardware accelerated identification of HTTP protocol headers, since HTTP is the most used protocol on the Internet. The goal is to design and implement a hardware architecture which will be used for detection of HTTP header in packet, and to achieve the throughput needed for monitoring of 100 Gbps networks. Nondeterministic finite automata and massive parallelism has been used for pattern match detection.

Kľúčové slová

HTTP, nedeterministický konečný stavový automat, BRAM, programovateľné hradlové pole, VHDL

Keywords

HTTP, Nondeterministic Finite Automata (NFA), BRAM, Field Programmable Gate Array (FPGA), VHDL

BRYNDZA, I. *Akcelerace identifikace HTTP hlaviček v obvodech FPGA*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav mikroelektroniky, 2015. 35 s., 2 s. příloh. Bakalářská práce. Vedoucí práce: Ing. Marián Pristach.

Prehlásenie

Prehlasujem, že svoju bakalársku prácu na tému **„Akcelerácia identifikácie HTTP hlavičiek v obvodoch FPGA“** som vypracoval samostatne pod vedením vedúceho bakalárskej práce a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, predovšetkým som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si úplne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich zákona 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 zákonníka č. 40/2009 Sb.

V Brne dňa: ……..

.....
podpis autora

Poďakovanie

Ďakujem vedúcemu bakalárskej práce Ing. Mariánovi Pristachovi za účinnú metodickú a pedagogickú pomoc pri spracovaní bakalárskej práce. Ďalej ďakujem Ing. Viktorovi Pušovi, Ph.D. za výbornú odbornú pomoc a za ďalšie cenné rady pri tvorbe praktickej časti práce. Chcel by som tiež poďakovať mojim rodičom a mojej priateľke, ktorí ma podporovali počas celého štúdia a tak umožnili vznik tejto bakalárskej práce.

V Brne dňa: ……..

.....
podpis autora

Obsah

Úvod

Rýchly rozvoj technológií a sním súvisiaci rozvoj sietí prináša nárast bezpečnostných hrozieb. S pribúdajúcim množstvom zariadení, ktoré sú schopné internetového pripojenia, pribúda aj množstvo používateľov, ktorí internetové pripojenie využívajú. Stabilita, bezpečnosť, dostupnosť a rýchlosť prenosových liniek sú požiadavky, na ktoré sú kladené čoraz väčšie nároky. Tieto faktory vedú k potrebe zlepšovania a urýchľovania systémov pre monitorovanie sieťovej prevádzky.

Najrozšírenejším a najpoužívanejším protokolom na Internete je HTTP protokol. Poskytuje veľmi dobrý a rýchly spôsob pre výmenu rôznych informácií. Prináša však aj veľa hrozieb, pretože je náročné ho analyzovať a spracovávať firewallmi alebo inými monitorovacími prvkami. Monitorovacie systémy sú väčšinou riešené softvérovými aplikáciami, ktoré nie sú schopné dostatočne rýchlo spracovávať množstvo dátových tokov. Tieto problémy vedú k potrebe vývoja urýchľovania detekcie HTTP protokolu. Táto práca je zameraná na hardvérové urýchľovanie s využitím technológie FPGA.

Práca popisuje návrh architektúry, ktorá umožňuje vyhľadávať v pakete hlavičky HTTP protokolu, s rýchlosťou 100 Gb/s. Na ich vyhľadávanie je využitý nedeterministický konečný stavový automat. Jednoduchosť automatu a zároveň rýchlosť vyhľadávania je zabezpečená využitím paralelne pracujúcich automatov. Návrh je popísaný v jazyku VHDL. Funkcia dizajnu bola overená simuláciou v programe ModelSim SE 10.0 a množstvo použitých zdrojov bolo zistených pomocou syntézy v programe Vivado 2015.1 pre FPGA typu Virtex-7 H580T.

Text práce je členený na 3 kapitoly. V kapitole 1 je popísaný teoretický rozbor protokolu HTTP, základné informácie o FPGA, o jazyku VHDL, o protokole FLU a o nedeterministickom konečnom stavovom automate. V kapitole 2 je popísaný návrh a funkcia architektúry pre detekciu HTTP. V kapitole 3 sú ukážky simulácie niektorých jednotiek a výsledky syntézy implementovanej architektúry. V závere je zhrnutie obsahu práce a dosiahnutých výsledkov.

1 Teoretický rozbor

1.1 TCP/IP protokoly

Agentúra amerického ministerstva obrany DARPA začala s vývojom sady internetových štandardov, ktoré špecifikovali detaily počítačovej komunikácie, pravidlá pre prepojovanie sietí a smerovanie komunikácie [**1**]. Tieto štandardy sú oficiálne pomenované ako balík internetových protokolov (*angl. Internet Protocol Suite*) často označovaný ako TCP/IP, podľa dvoch hlavných protokolov v tomto štandarde. Sú to TCP (*Transmission Control Protocol*) a IP (*Internet Protocol*). TCP/IP model je protokolový model, pretože popisuje funkcie protokolov na každej vrstve. Na rozdiel od OSI (*Open systems interconnection*) modelu, ktorý je referenčným modelom a bol navrhnutý pre lepšie pochopenie funkcií a procesov komunikácie [**2**].

TCP/IP bol pre svoju zložitosť navrhnutý ako vrstvovitý model, kde každá vrstva využíva služby nižšej vrstvy a poskytuje služby vrstve nad ňou. Protokoly týchto vrstiev ďalej obsahujú špecifikáciu formátovania, adresovania, odosielania, smerovania a prijímania dát.

1.1.1 Architektúra TCP/IP

Predpoklady pre systémy implementujúce TCP/IP sú popísané v RFC 1122 [**3**] a RFC 1123 [**4**]. TCP/IP model definuje štyri komunikačné abstraktné vrstvy. Sú to vrstvy aplikačná, transportná, sieťová a linková [**5**] [**6**].

Linková vrstva zabezpečuje komunikáciu zariadení na lokálnej sieti, tzv. linke. Je to najnižšie položená vrstva v modeli. Takáto sieť je ohraničená smerovačmi. Linková vrstva prijíma pakety od sieťovej vrstvy a zapuzdruje ich do rámcov. Každý rámec obsahuje hlavičku, zapuzdrený paket a prívesok.

Funkcie protokolov linkovej vrstvy:

- prenos jednotiek sieťovej vrstvy, paketov,
- adresovanie zariadení na linke,
- príprava dát na prenos fyzickým médiom,
- determinizácia okrajov rámcov,
- detekcia chýb.

Charakteristika TCP/IP popisuje spôsoby prekladu sieťovej adresy na linkovú. Linkovou adresou je napríklad MAC adresa.

Protokoly sieťovej vrstvy, ktorá sa tiež nazýva ako internetová vrstva, majú za úlohu smerovať dáta (pakety) z počiatočnej siete do cieľovej. V balíku protokolov TCP/IP ide dnes najmä o protokoly IPv4 a IPv6.

Funkcie IP protokolov:

- adresovanie klientov a sietí IP adresou,
- smerovanie paketov medzi sieťami.

Medzi počiatočnou a cieľovou stanicou sa typicky nachádza mnoho intermediálnych prepojení (tzv. *hops*). Smerovače postupne posielajú pakety ďalším smerovačom (tzv. *next-hop router*). Sieťová vrstva poskytuje takzvané doručenie s najlepším úsilím a nezaručuje spoľahlivý prenos dát. Spoľahlivý prenos musí v prípade potreby zabezpečiť niektorý z protokolov vyššej vrstvy.

Transportná vrstva vytvára základný komunikačný kanál medzi procesmi aplikácií s definovaným stupňom spoľahlivosti, ktorý je nezávislý na aktuálne použitom fyzickom prenosovom médiu. Najčastejšie využívané protokoly sú TCP a UDP. V prípade TCP sú prenášané jednotky segmenty a v prípade UDP sú to datagramy.

TCP je spojovo orientovaný protokol zabezpečujúci spoľahlivosť spojenia. Zahŕňa to zoraďovanie, potvrdzovanie, retransmisiu a riadenie toku paketov, podľa stavu sieťového spojenia.

UDP je nespojovaný protokol poskytujúci prenos dát bez zriadenia spojenia. Okrem kontrolného súčtu a identifikácie procesov neposkytuje ďalšie služby.

Aplikačná vrstva je najvyššia vrstva modelu TCP/IP. Zahŕňa protokoly používané aplikáciami poskytujúcimi služby ich používateľom. Rozlišujú sa dve kategórie protokolov aplikačnej vrstvy: užívateľské protokoly poskytujúce služby priamo používateľovi a systémové protokoly, ktoré vykonávajú základné systémové funkcie. Užívateľský protokol je napríklad Telnet, FPT, SMTP, HTTP atď. Príkladom systémového protokolu je DNS, SNMP, BOOTP, RARP atď.

1.2 HTTP protokol

HTTP (*Hypertext Transfer Protocol*) je protokol aplikačnej vrstvy [**7**] [**8**]. Dnes slúži pre distribuované, kolaboratívne, hypermediálne informačné systémy ale pôvodne bol navrhnutý pre výmenu hypertextových dokumentov (Hypertext Markup Language, HTML) Tento protokol sa používa v rámci World-Wide Web od roku 1990.

Jeho prvá verzia, HTTP/0.9, bola veľmi jednoduchá, a dnes sa využíva len zriedka [**9**]. HTTP/1.0 je vylepšená verzia protokolu, ktorá podporuje hlavičky vo formáte MIME (Multipurpose Internet Mail Extensions). Hlavičky obsahujú metainformácie o prenášaných dátach a modifikátoroch sémantiky požiadaviek a odpovedí. Verzia HTTP/1.1 rozširuje tieto metainformácie zohľadňujúc hierarchické proxy, potrebu prezistentných spojení a virtuálne servery. Tieto dve verzie prinášajú tiež niekoľko nových metód pre požiadavky. Novinkou je HTTP/2.0 [**10**], ktorým sa už práca nezaoberá.

HTTP je protokol typu požiadavka/odpoveď v architektúre typu klient/server. Klient pošle požiadavku serveru obsahujúcu URI (Uniform Resource Indentifier), verziu protokolu, nasleduje správa vo formáte MIME obsahujúca modifikátory požiadavky, informáciu o klientovi a metainformácie o tele správy. Server odpovie pomocou stavového riadku, ktorý zahŕňa verziu protokolu správy, a kódom o správnosti údajov. Ďalej v odpovedi nasleduje správa vo formáte MIME obsahujúca informácie o serveri a metainformácie o tele správy. Komunikácia je väčšinou inicializovaná klientom. Od verzie protokolu HTTP/1.1 môže byť jedno spojenie využité pre odoslanie či prijatie viacerých požiadaviek/odpovedí.

Požiadavka začína riadkom označovaným ako Request-Line. Riadok začína použitou metódou, následne URI oddelený medzerami a reťazec obsahujúci verziu protokolu. Riadok je ukončený dvojicou znakov \r\n. Metóda určuje metódu vykonanú na zdroji identifikovanom pomocou URI. HTTP server musí podporovať minimálne metódy GET a HEAD. Metóda GET požaduje od serveru, aby poslal webovú stránku. HEAD zase požaduje iba hlavičku správy. URI identifikuje zdroj, nad ktorým má byť vykonaná požiadavka. Príklad riadku Request-Line by mohol vyzerať takto:

```
GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1\r\n
```

V prípade, že nie je známa celá cesta, URI nesmie ostať prázdny. Vyplní sa ako "/" čo znamená server.

Za prvým riadkom nasleduje niekoľko ďalších riadkov hlavičky. Tie obsahujú doplňujúce informácie o požiadavke, ako aj o samotnom klientovi. Slúžia ako modifikátory požiadavky. Sú to: hostiteľ dokumentu, identifikácia klienta, požiadavky na kódovanie, dĺžka obsahu v tele správy a iné. Celá hlavička je ukončená reťazcom \r\n. Príklad HTTP požiadavky:

```
GET / HTTP/1.1\r\n
Host: example.com\r\n
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:37.0) Firefox /37.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5,en-us;q=0.5,en;q=0.3\r\n
Accept-Encoding: gzip, deflate\r\n
DNT: 1\r\n
Connection: keep-alive\r\n
\r\n
```

Prvý riadok odpovede sa nazýva Status-Line. Obsahuje verziu protokolu, numerický kód stavu a stav popísaný textovým reťazcom. Príklad riadku Status-Line:

```
HTTP/1.1 200 OK\r\n
```

Účel a syntax hlavičky odpovede sa v podstate nelíši od hlavičky požiadavky. Odlišujú sa množinou definovaných modifikátorov. Odpoveď obsahuje najmä metainformácie tela správy, ktorá nasleduje za hlavičkou. Telom odpovede je HTML dokument. V skrátenom príklade sú uvedené jeho prvé dva riadky. Príklad HTTP odpovede:

```
HTTP/1.1 200 OK\r\n
Date: Wed, 27 May 2015 8:16:58 GMT\r\n
Content-Type: text/html; charset=utf-8\r\n
Expires: Thu, 26 May 2016 8:16:58 GMT\r\n
Last-Modified: Wed, 04 Mar 2015 12:32:33 GMT\r\n
Server: nginx\r\n
Cache-control: no-cache, no-store, must-revalidate\r\n
Content-Length: 293\r\n
\r\n
<!doctype html>\n
<html>\n
```

všetkých prenášaných dát [**11**]. Taktiež nie je zanedbateľné jeho využívanie pre tunelovanie, tzn. zapuzdrenie inej komunikácie a jej prenos prostredníctvom HTTP protokolu.

1.3 Regulárny výraz

Regulárny výraz je špeciálny textový reťazec, ktorý popisuje hľadaný vzor. Najjednoduchší regulárny výraz môže pozostávať z jedného obyčajného písmena abecedy. Napríklad písmeno *„e“*. Takýto regulárny výraz nájde prvý výskyt tohto písmena v reťazci. Ak by bol reťazec *„Dnes je pekne“*, výraz nájde písmeno *„e“* v slove *„Dnes“*. Podobne regulárny výraz *„bude“* nájde slovo *„bude“* v reťazci *„Zajtra bude rovnako“*.

Pre rozšírenie možností regulárnych výrazov sú vymedzené špeciálne znaky s rôznym významom. Sú to napríklad: hranaté zátvorky, spätná lomka, zvislá lomka, hviezdička, bodka, dolár, strieška, otáznik, plus a mnoho ďalších. Napríklad regulárny výraz *„[GPH]ET“* nájde kdekoľvek v reťazci slovo *„GET“* ale aj *„PET“* či *„HET“*. Regulárny výraz *„GET|HEAD“* nájde slovo *„GET“* alebo slovo *„HEAD“*. Bodka *„.“* znamená ľubovoľný znak. Regulárny výraz *„ab*“* znamená, že písmeno *„b“* sa môže v reťazci nachádzať nula až viackrát. Detekoval by sa vzor *„a“*, *„ab“*, *„abb“*, *„abbb“* atď. Reulárny výraz *„ab+“* znamená, že písmeno *„b“* sa môže v reťazci nachádzať raz alebo viackrát. Ak by bolo potrebné hľadať vzor napríklad *„1+1=2“*, odpovedajúci regulárny výraz by bol *„1\+1=2“*. Spätná lomka zmení špeciálny znak regulárneho výrazu na obyčajný vyhľadávaný znak.

Vyššie uvedené príklady regulárnych výrazov predstavujú len niekoľko príkladov. Pomocou regulárnych výrazov sa dá detekovať prakticky ľubovoľný textový reťazec. Podrobnejší popis je možné nájsť v [**12**].

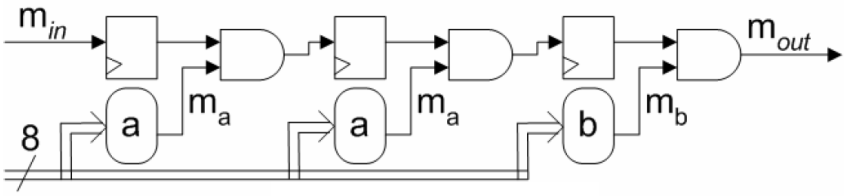
1.4 Nedeterministický konečný stavový automat pre detekciu výrazov

Kľúčovou a zároveň veľmi výpočtovo náročnou operáciou používanou v oblasti monitorovania a bezpečnosti počítačových sietí je hľadanie vzorov v dátach paketov [**13**]. Vyhľadávanie vzorov je taktiež vhodné pre detekciu a spracovanie HTTP protokolu, pričom je pre detekciu útokov potrebné hľadať rádovo tisíce vzorov na gigabitových rýchlostiach, čo súčasné softvérové riešenia neumožňujú. Aby bolo zaistené vyhľadávanie na vysokých rýchlostiach používajú sa rôzne spôsoby hardvérovej akcelerácie. Pán Vern Paxon ukázal [**14**], že pre identifikáciu nebezpečnej sieťovej komunikácie sú omnoho účinnejšie regulárne výrazy než reťazce. V posledných rokoch boli preto vytvorené hardvérové architektúry, založené na technológii FPGA, ktoré umožňujú hľadanie regulárnych výrazov v gigabitových linkách. Jeden z prvých prístupov umožňujúci hľadať množiny regulárnych výrazov, vychádza z mapovania nedeterministického konečného stavového automatu (*angl. nondeterministic finite automata*) (NFA) do technológie FPGA.

Obvod implementujúci NFA pre detekciu výrazov pozostáva zo zreťazených jednotiek porovnávajúcich znak. Výhoda NFA je, že počet stavov narastá lineárne s dĺžkou regulárneho výrazu. Veľkosť pamäti pre uloženie automatu tak približne odpovedá popisu množiny regulárnych výrazov, takže k uloženiu celej tabuľky prechodov je potrebné omnoho menej pamäti, než v prípade deterministického automatu (DFA). Nevýhoda NFA spočíva v tom, že nemajú lineárnu časovú zložitosť, čo môže viesť k zhoršeniu priepustnosti. Preto architektúry na báze NFA vyžadujú pre garantovanie priepustnosti vysoký stupeň paralelného spracovania.

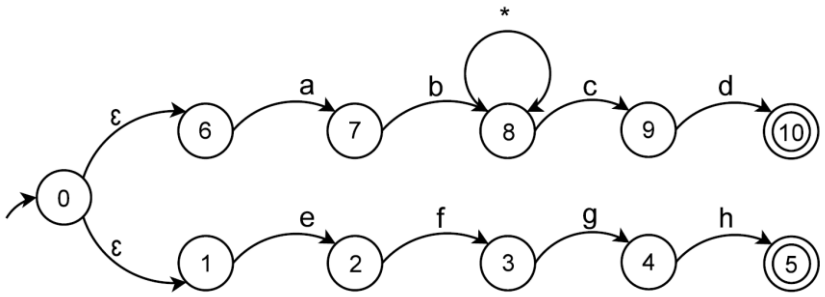
Pri mapovaní automatu do FPGA je použité one-hot kódovanie. Každý stav má vytvorený jednobitový klopný obvod. Pretože každý klopný obvod je možné nezávisle nastaviť do logickej jednotky (aktívny stav) alebo logickej nuly, je možné súčasne aktivovať viac než jeden stav a prechádzať tak v automate všetky nedeterministické cesty.

Prechody medzi cestami sú riešené pomocou kombinačnej logickej funkcie. Ku každému prechodu je vytvorený komparátor, ktorý porovnáva vstupný znak a stav s definíciou prechodu. Vytvára sa tak signál indikujúci či je prechod do aktívneho stavu možný. Príklad takého automatu pre detekciu výrazu *„aab“* je na obr. 1.1.



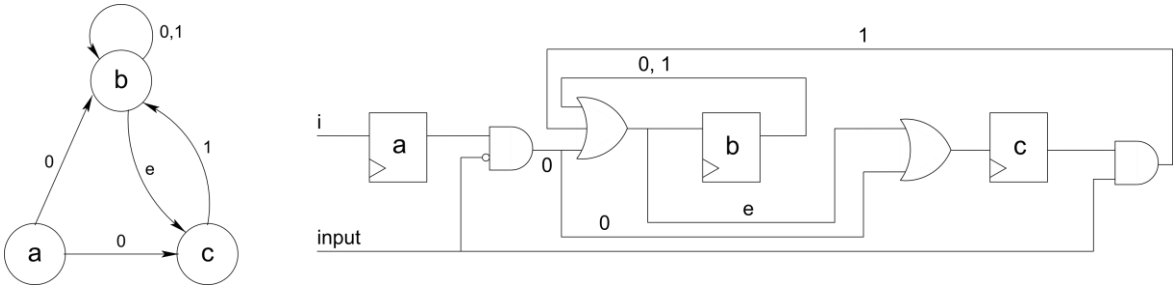
*Obr. 1.1: Nedeterministický konečný stavový automat s distribuovanými komparátormi [**15**]*

Na obr. 1.2 je príklad nedeterministického konečného stavového automatu pre regulárne výrazy *„ab.*cd“* a *„efgh“*. Konštrukcia *„.*“* popisuje ľubovoľný reťazec. Symbol *„ε“* značí prázdny reťazec.



*Obr. 1.2: Nedeterministický konečný stavový automat pre regulárne výrazy „ab.*cd“ a „efgh“ **[13]***

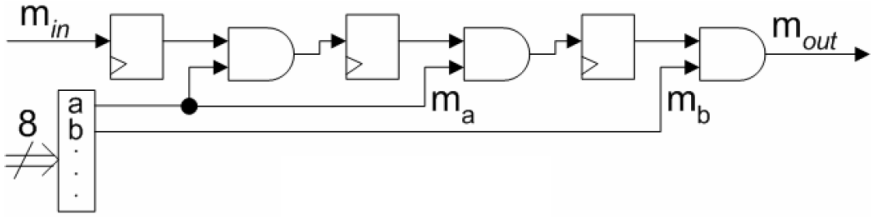
Na obr. 1.3 je príklad architektúry pre jednoduchý automat s tromi stavmi. V ľavej časti je znázornený automat a v pravej časti jeho mapovanie na hardvérovú architektúru. Jednotlivým stavom *a, b, c* odpovedajú rovnako pomenované klopné obvody. Jednotka v registri znamená, že stav je aktívny. Realizácia jednotlivých prechodov je znázornená rovnakým spôsobom ako v automate.



*Obr. 1.3: Príklad mapovania nedeterministického automatu do FPGA **[13]***

Prístup porovnávania s distribuovanými komparátormi má viacero nevýhod. Jednou z nich je, že množstvo použitých hardvérových zdrojov s každým ďalším reťazcom alebo regulárnym výrazom veľmi rýchlo narastá. Je to spôsobené narastajúcim množstvom komparátorov, pretože je pre každý prechod použitý jeden. Ďalšou nevýhodou je nízka frekvencia, pretože do výpočtu nasledujúceho stavu je nutné započítať aj porovnávanie vstupného znaku.

Mapovanie NFA do FPGA rozšíril Clark o zdieľaný dekodér znakov [**15**]. Kľúčom pozorovania vedúcemu k zjednodušeniu je, že 8 bitové porovnávanie nemusí byť vykonávané každou jednotkou. V prípade výrazu obsahujúceho niekoľko tisíc znakov je vysoko pravdepodobné, že sa vo výraze nachádzajú stovky rovnakých znakov. Znamená to stovky rovnakých komparátorov. Tie zbytočne zaberajú cenné logické a prepojovacie zdroje. Zbytočne zabrané zdroje je možné ušetriť pomocou dekodéru 8-na-256 (obr. 1.4). Namiesto vysielania 8 bitového signálu každej jednotke sa v tejto optimalizácii posiela už iba 1 bitový signál z dekodéra, zodpovedajúci hodnote znaku cieľovej jednotky.



*Obr. 1.4: Nedeterministický konečný stavový automat so zdieľaným dekodérom **[15]***

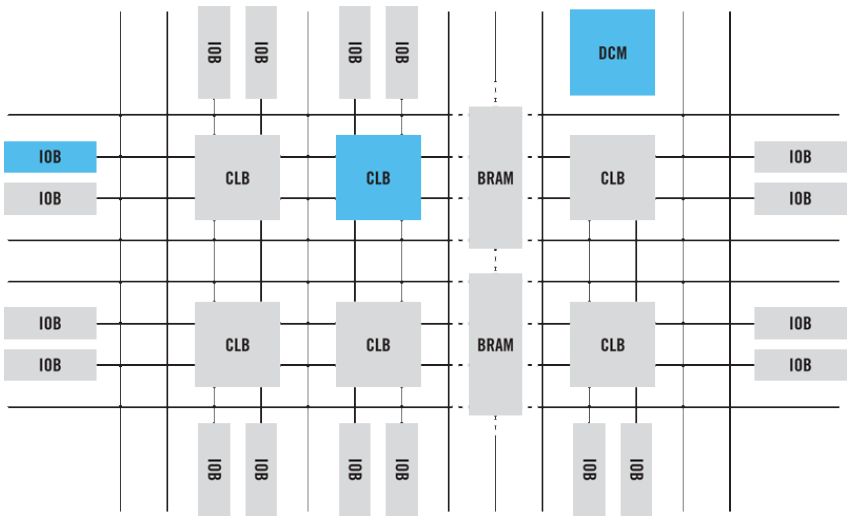
Tento prístup viac než zdvojnásobuje maximálnu kapacitu výrazov na danom programovateľnom logickom obvode. Taktiež šetrí prepojovacie zdroje. Zatiaľ čo prístup s distribuovanými komparátormi zabral 8*n prepojení zo vstupu k jednotke porovnávajúcej znak, prístup riešenia so zdieľaným dekodérom zaberie iba n prepojení pričom n je počet jednotiek porovnávajúcich znak.

NFA je popísaný tak, že spracováva 8 bitov v jednom hodinovom takte. Prenosová rýchlosť tohto NFA je pri reálne dosiahnuteľnej frekvencii FPGA 200 MHz iba 1,6 Gb/s. Aby sa dosiahla žiadúca prenosová rýchlosť 100 Gb/s pri pracovnej frekvencii FPGA 200 MHz je potrebná zbernica so šírkou 512 bitov. Znamenalo by to zväčšenie vstupného zdieľaného dekodéru v NFA z 256 znakov na 2^{512} znakov, čo je exponenciálny nárast, nehovoriac o zložitosti NFA a počte jeho stavov, ktoré by bolo nutné popísať. Výsledný dizajn by zabral veľmi veľké množstvo zdrojov a nebolo by ho možné realizovať na FPGA.

Výhodnejší spôsob, ako zabezpečiť prenosovú rýchlosť 100 Gb/s je použiť 64 jednoduchých NFA, ktoré spracovávajú 8 bitov v jednom takte. Automaty by spracovali 8*64 bitov paralelne v jednom takte. Výsledkom by bol iba lineárny nárast veľkosti NFA automatov a zachovanie jednoduchosti NFA.

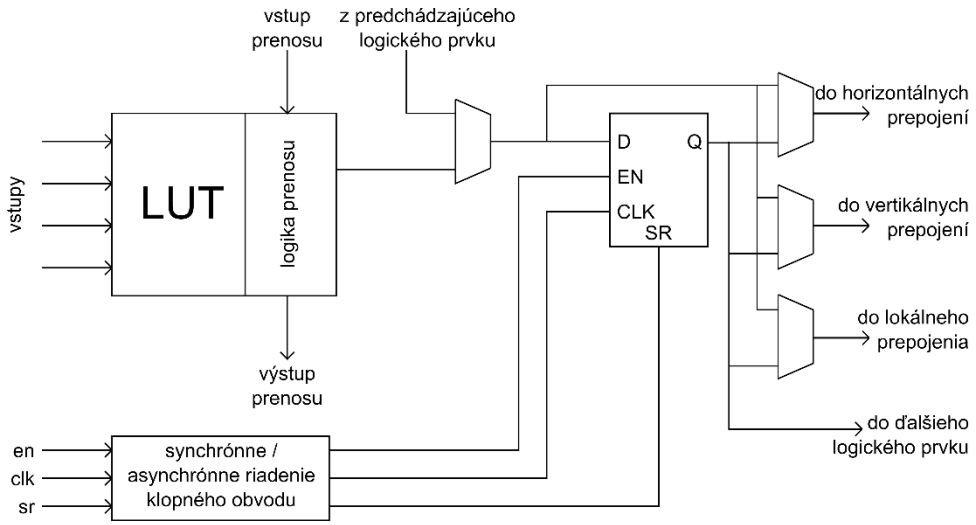
1.5 Programovateľné hradlové pole FPGA

FPGA (*Field Programmable Gate Array*) je polovodičový čip. Jeho základom sú programovateľné logické bloky (CLB), ktoré sa skladajú zo SLICE blokov (obr. 1.6). Jeden SLICE blok je schopný realizovať kombinačný alebo sekvenčný obvod [**16**]. Obsahuje generátor logickej funkcie pomocou pamätí (LUT tabuľku), podpornú logiku pre vstup a výstup aritmetického prenosu, klopný obvod typu D, blok asynchrónneho či synchrónneho riadenia klopného obvodu a multiplexory na prepojovanie s ďalšími prvkami [**17**]. V minulosti sa používali LUT tabuľky so 4 vstupmi a jedným výstupom. Dnes už väčšina FPGA obvodov používa 6 vstupové LUT s dvomi výstupmi.



*Obr. 1.5: Štruktúra blokov v FPGA [**18**]*

Ďalej sa v FPGA nachádzajú vstupno-výstupné bloky (IOB), ktoré majú podporu mnohých I/O štandardov. Na prepojenie všetkých komponentov slúžia programovateľné horizontálne a vertikálne prepojenia. Pre rozvod hodinového signálu sa však používajú špeciálne prenosové cesty, ktoré vykazujú veľmi malé oneskorenie.



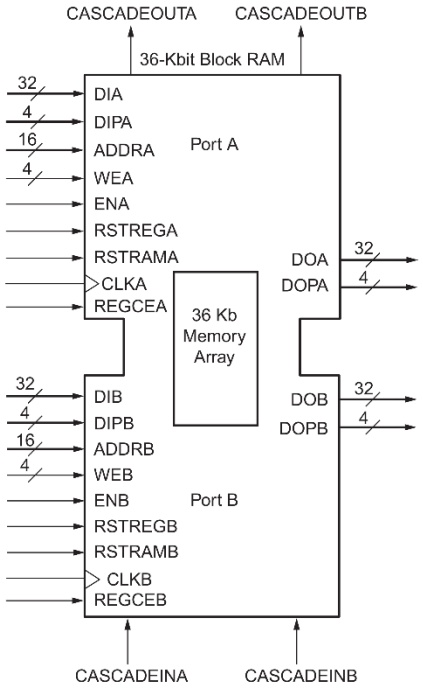
*Obr. 1.6: Zjednodušená bloková schéma logického prvku (SLICE) [**17**]*

K popísaným blokom postupným vývojom pribúdali ďalšie špecializované bloky. Vďaka nim je možné minimalizovať spotrebu zdrojov a zároveň zvýšiť frekvenciu. Ako prvé pribudli vstavané bloky pamätí (BRAM). Tieto pamäťové bloky sú umiestené v FPGA v stĺpci a je možné ich nakonfigurovať do rôznych pamäťových funkcií ako napr. ROM, jedno alebo dvojbránové RAM alebo ako FIFO. Podrobnejší popis BRAM sa nachádza v podkapitole 1.6. Ďalšími blokmi sú napríklad: Digital Clock Manager (DCM), ktorý sa používa pre zmenu frekvencie, fázy alebo korekciu hodinového signálu, vstavané násobičky a iné. Dostupné vstavané bloky sa líšia podľa daného typu obvodu FPGA.

V dnešnej dobe sú FPGA obvody využívané najmä vďaka ich veľkej flexibilite a možnosti realizovať zmeny priamo u zákazníka. Sú ľahko programovateľné a programovanie obvodu je možné opakovať bez obmedzení, pretože stavebné prvky FPGA udržujú svoju konfiguráciu väčšinou v statickej RAM pamäti (SRAM). Programovanie prebieha buď automaticky po zapnutí napájania z externe pripojenej pamäti EEPROM, FLASH alebo pomocou rozhrania počítača.

1.6 Synchrónna dvoj-portová bloková pamäť RAM

FPGA typu Virtex-7 XC7VH580T, použité v tejto práci, obsahuje 940 BRAM pamätí, z toho každá má veľkosť 36 kbit [**19**]. Pamäť má dva nezávislé prístupové porty, A a B. Štruktúra pamäte je symetrická a tak sú porty ľubovoľne zameniteľné. Bloková schéma pamäte je na obr. 1.7. Popis vstupných a výstupných portov je v tab. 1.1. Dáta môžu byť zapisované do jedného alebo oboch portov naraz a takisto čítané z jedného alebo oboch portov naraz. Operácie čítania a zápisu sú synchrónne a reagujú na hranu hodinového signálu.



*Obr. 1.7: Dvoj-portová pamäť BRAM a jej signály [**20**]*

Pri zápise a čítaní naraz z jednej adresy prostredníctvom jedného portu existujú tri módy. WRITE_FIRST, READ_FIRST a NO_CHANGE. V móde WRITE_FIRST sú vstupné dáta zapísané do pamäte a zároveň vyvedené na výstupný port. V móde READ_FIRST sú vopred uložené dáta vyvedené na výstupný port, zatiaľ čo sú vstupné dáta zapisované do pamäte. NO_CHANGE mód znamená, že v prípade zapisovania dát ostanú na výstupe dáta, ktoré sa prečítali pred začiatkom zápisu.

Ak sa však pristupuje z oboch portov na jednu adresu, je potrebné si uvedomiť dve možné situácie. Buď majú oba porty rovnaký hodinový signál (synchrónne časovanie), alebo má každý port odlišný hodinový signál (asynchrónne časovanie).

Synchrónne časovanie je práve záležitosť dizajnu v tejto práci. V tomto prípade je potrebné zaistiť, aby oba porty nezapisovali naraz na jednu adresu pamäte pokiaľ nezapisujú rovnaké dáta. Dáta na danej adrese by potom boli nedefinované a nebolo by možné určiť aké informácie sa na adresu zapíšu. V prípade čítania a zápisu naraz z jednej adresy je možný mód READ_FIRST. V prípade módu WRITE_FIRST alebo NO_CHANGE by výstupné dáta nemuseli byť správne.

V prípade asynchrónneho časovania je potrebné sa vyhnúť módu READ_FRST, pretože neexistuje záruka, že sa čítajú pôvodné dáta pred zápisom nových. Naopak mód WRITE_FIRST je odporúčaný mód v prípade simultánneho zápisu a čítania z jednej adresy. V prípade ignorovania týchto pravidiel, čítanie alebo zápis môžu spôsobiť nedefinované chovanie [**20**].

Tab. 1.1: Popis vstupných a výstupných portov BRAM

Port	**Funkcia**
DI(A/B)	Zbernica vstupných dát
DIP(A/B)	Zbernica vstupných paritných dát. Môže byť použitá pre vstup dodatočných dát.
ADDR(A/B)	Zbernica s adresou
WE(A/B)	Zbernica pre povolenie zápisu (po bajtoch)
EN(A/B)	Ak neaktívny, nezapíšu sa dáta do BRAM a výstupné dáta ostanú nezmenené.
RSTREG(A/B)	Synchrónny Set/Reset výstupných registrov
RSTRAM(A/B)	Synchrónny Set/Reset výstupných dátových latch registrov
CLK(A/B)	Vstup pre hodinový signál
DO(A/B)	Zbernica výstupných dát
DOP(A/B)	Zbernica výstupných paritných dát. Môže byť použitá pre výstup dodatočných dát.
REGCE(A/B)	Povoľovací signál pre výstupný register.
CASCADEIN(A/B)	Kaskádny vstup
CASCADEOUT(A/B)	Kaskádny výstup

1.7 Jazyk VHDL

K programovaniu FPGA obvodov sa používajú HDL (*Hardware Description Language*) jazyky [**17**]. Vývoj jazyka VHDL začal v roku 1981 v rámci výskumného projektu VHSIC (*Very High Speed Integrated Circuits*). Vlastný vývoj jazyka bol zadaný firmám, ktoré sa podieľali na projekte v roku 1983. Tieto firmy vytvorili do roku 1985 základ definície jazyka VHDL. Syntax jazyka vychádzala z jazyka ADA. Skratka VHDL vznikla ako akronym z názvu *VHSIC Hardware Description Language*. V roku 1987 bol organizáciou IEEE (*Institute of Electrical and Electronics Engineers*) prvý raz publikovaný štandard jazyka VHDL pod označením *IEEE Standards VHDL Language Reference Manual (IEEE Std 1076-1987)*, často v literatúre označovaný ako VHDL-87. Dodnes bolo publikovaných viacero revízií tohto štandardu. Zatiaľ ako posledná je revízia publikovaná v roku 2009 (*IEEE Std 1076-2008*) [**21**].

Programovací jazyk VHDL poskytuje možnosti pre popis logických štruktúr a funkcií digitálnych systémov v niekoľkých úrovniach abstrakcie, od hradlovej úrovne až po algoritmickú a architektonickú. Je zamýšľaný, okrem iného, ako modelovací jazyk pre simuláciu. Môže byť použitý aj pre hardvérovú syntézu, ak sa využije iba istá časť jazyka, ktorú je možné automaticky preložiť do hardvéru. Je navrhnutý tak, aby splnil množstvo potrieb spojených s procesom návrhu. Jazyk VHDL umožňuje:

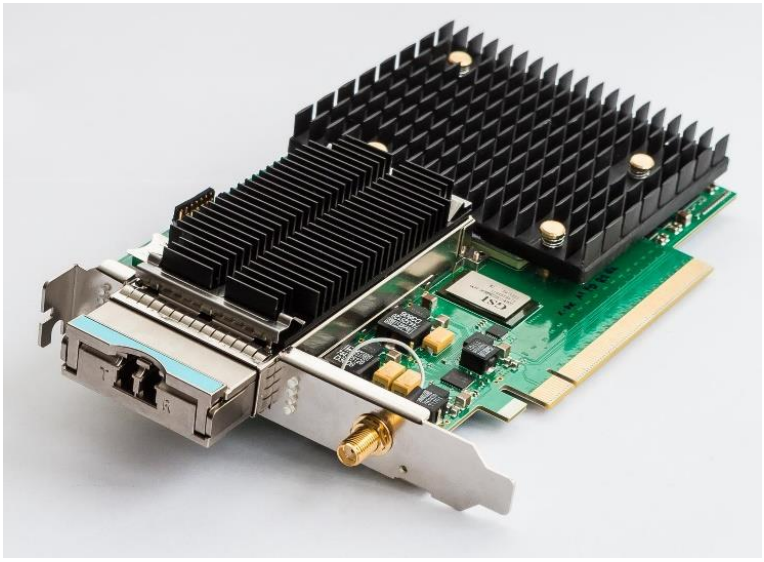
- popis štruktúry systému, z akých podsystémov sa skladá a ako sú tieto podsystémy prepojené,
- špecifikovať funkciu systému pomocou známych foriem programovacích jazykov,
- výsledný návrh systému simulovať a tak overiť správnosť funkcie ešte pred výrobou,
- syntézu detailnej štruktúry návrhu z abstraktnej špecifikácie,
- určiť elektrické aspekty obvodu ako sú oneskorenia cez hradlá či vysokú impedanciu,
- popis paralelného chovania systému [**22**].

Okrem VHDL existujú aj iné HDL jazyky. Napríklad ďalšie dva jazyky štandardizované organizáciou IEEE sú Verilog a SystemC.

1.8 Rodina kariet COMBO

Tieto karty boli vytvorené organizáciou CESNET [**23**] pre projekt hardvérovej akcelerácie IPv6 smerovania (Liberouter [**24**]). Neskôr sa projekt zameral na vývoj FPGA kariet pre 10, 40 a 100 Gb siete a hardvérovú akceleráciu bezpečnosti sietí a monitorovacích nástrojov. Počas dlhej histórie projektu Liberouter sa výskumný tím zúčastnil mnohých EU projektov, získal mnoho skúseností v oblasti sietí a vytvoril mnoho hardvérovo akcelerovaných nástrojov založených na FPGA kartách. Mnoho týchto nástrojov bolo nasadených v sieti CESNET a boli aj základom niekoľkých komerčných produktov.

Základom každej karty je FPGA čip doplnený pamäťami, PCI-Express zbernica, konektory rozhrania a zdroje energie, pre lepšiu flexibilitu FPGA čipov. Funkcionalita COMBO kariet tak môže byť za veľmi krátku dobu zmenená nahratím novej konfigurácie do FPGA [**25**].



Obr. 1.8: Karta COMBO-100G [25]

V tejto práci je dizajn navrhovaný pre kartu COMBO-100G (obr. 1.8), ktorá je vyvinutá organizáciou CESNET v spolupráci s firmou INVEA-TECH [**26**].

Karta obsahuje:

- 1x CFP2 klietku optického modulu,
- PCI-Express 3 generácie, 16 linkový,
- FPGA Virtex-7 H580T,
- 3x QDR-IIIe SRAM 500 MHz, 72 Mib každá,
- 8x DDR3 DRAM, 800 MHz, 4Gib každá,
- precízny vstup pre signál, ktorý slúži na generovanie presných časových značiek prichádzajúcim paketom (napríklad GPS signál).

1.9 FLU protokol

Protokol FLU (*FrameLinkUnaligned*) je jeden zo základných zbernicových protokolov v projekte Liberouter [**24**], ktorý je možné využívať na prepojenie komponentov. Protokol prenáša dáta vo forme paketu. Používa sa v FPGA dizajnoch v rámci projektu Liberouter. Vznikol začiatkom roku 2012, pretože starší FrameLink (FL) už nedosahoval

potrebné rýchlosti [**16**]. Hlavným dôvodom vzniku FLU bolo neefektívne využívanie širších zberníc (256 bitov a viac) protokolom FL, kde každý začiatok nasledujúceho paketu bol zarovnaný na LSB bit celého slova. Dochádzalo tak k nevyužívaniu miesta na zbernici.

FLU umožňuje v jednom zbernicovom slove definovať koniec paketu a zároveň začiatok ďalšieho. Zvýši sa tak celková efektivita prenosu a celková dátová priepustnosť.

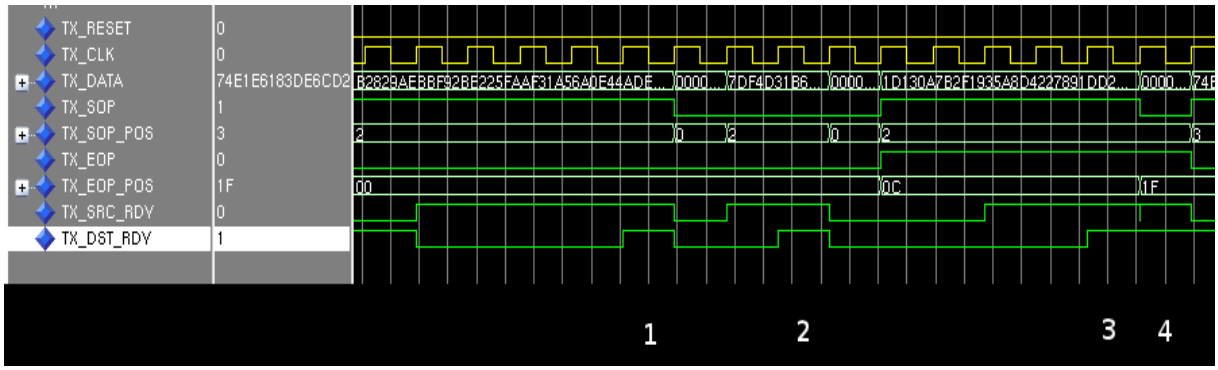
Pre potvrdenie platnosti všetkých signálov je vyžadovaná aktívna úroveň signálov SRC_RDY a DST_RDY. Pre platnosť signálu EOP_POS respektíve SOP_POS je vyžadovaná aktívna úroveň signálu EOP respektíve SOP. Medzi generické parametre patrí šírka zbernice prenášaného slova (DATA_WIDTH) a šírka zbernice SOP_POS (SOP_POS_WIDTH). Vďaka týmto parametrom dokážeme definovať začiatok rámca kdekoľvek na zbernici. Signály protokolu FLU a ich význam je uvedený v tab. 1.2.

Tab. 1.2: Signály protokolu FLU a ich popis

Signál	**Šírka**	**Popis**
DATA	DATA_WIDTH	Prenášané dáta
SOP_POS	SOP_POS_WIDTH	**Start of packet position** – indikuje pozíciu začiatku nového paketu
EOP_POS	$\log_2$(DATA_WIDTH/8)	**End of packet position** – indikuje pozíciu konca predchádzajúceho paketu
SOP	1	**Start of packet** – indikuje začiatok paketu
EOP	1	**End of packet** – indikuje koniec paketu
SRC_RDY	1	**Source ready** – indikuje, že zdroj je pripravený odosielať
DST_RDY	1	**Destination ready** – indikuje, že je pripravený príjemca

1.9.1 Ukážka komunikácie

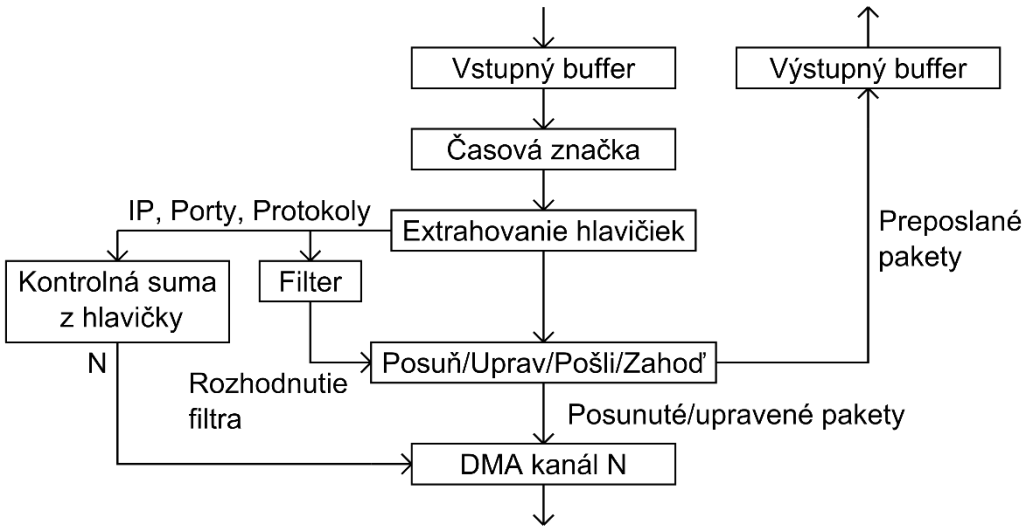
V ukážke na obr. 1.9 je hodnota DATA_WIDTH = 512 a SOP_POS_WIDTH = 3. V prvom cykle prenášaných dát začína paket P1. Hodnota SOP_POS = 2 takže prvý bajt paketu P1 sa nachádza na 128. bajte slova (*SOP_POS*DATA_WIDTH/2^SOP_POS_WIDTH = 2*512/2^3 = 128*). V prvom cykle je nevyužitých 16 a prenesených 48 bajtov. V druhom cykle pokračuje prenos paketu P1, kde je prenesených 64 bajtov. V treťom cykle končí paket P1. Posledný bajt paketu je na 12. bajte slova (*EOP_POS = 0x0C = 12*). Paket P1 má *48 + 64 + 13 = 125* bajtov. V treťom cykle sa zároveň začína prenos paketu P2. SOP_POS = 2. takže prvý bajt paketu P2 je na 128. bajte slova. Vo štvrtom cykle končí paket P2 EOP_POS = 0x1F = 31, takže posledný bajt paketu P2 je na 31. bajte slova. Signál SOP je v nule, nevyužitých je 33 bajtov slova.



Obr. 1.9: Ukážka komunikácie FLU

1.10 Systém HANIC

Hardware Accelerated Network Interface Card (HANIC) [**27**] obsahuje softvér aj firmware, ktorý priniesol ďalšie funkcionality rodine kariet COMBO. HANIC je sieťová karta s hardvérovo akcelerovanými funkciami. Karta má buď dva 40 Gb/s porty alebo osem 10 Gb/s portov alebo jeden 100 Gb/s port. Hlavnou funkciou je rozdeľovanie prichádzajúcich paketov medzi 8 softvérových rozhraní (DMA kanálov), pričom dáta z každého rozhrania môžu byť spracované na jednom CPU jadre. Distribúcia paketov je založená na kontrolných sumách vypočítaných z extrahovaných paketových hlavičiek.



Obr. 1.10: Firmware HANIC (zjednodušená bloková schéma)

Okrem distribúcie paketov do DMA kanálov, firmware HANIC tiež podporuje:

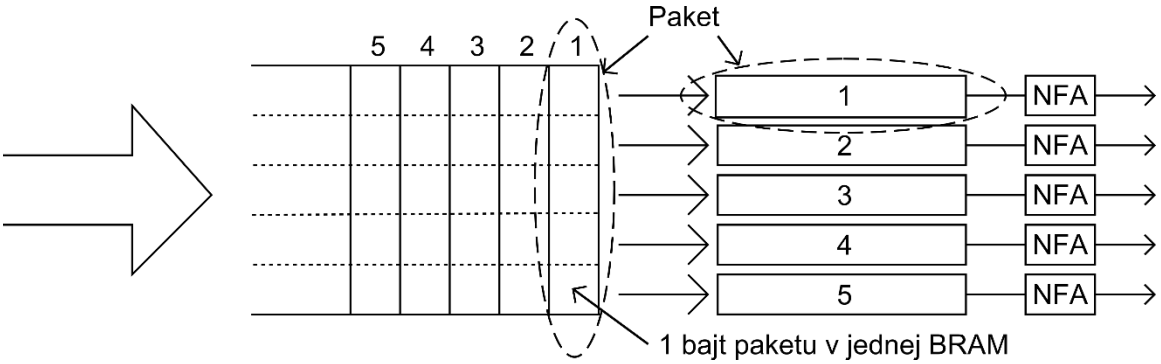
- vzorkovanie paketov 1:N,
- prepínanie zdrojových a cieľových adries a portov pre výpočet kontrolných súm (pre monitorovanie oboch smerov komunikácie na jedom CPU jadre),
- round-robin distribúciu paketov,
- orezávanie paketov pre lepšie využitie PCI zbernice,
- precízne časové značky,
- základné filtrovanie paketov pomocou určených pravidiel,
- základný firewall,
- exportovanie oddelených paketových hlavičiek do softvéru (namiesto celých paketov).

HANIC je v tejto práci rozšírený o identifikáciu HTTP hlavičiek.

2 Architektúra pre detekciu HTTP hlavičiek

Architektúra má za úlohu vyhľadávať v paketoch regulárne výrazy a tak detekovať HTTP hlavičky. Nedeterministický konečný stavový automat (NFA) so zdieľaným dekodérom, využívaný na detekovanie výrazov, vie spracovávať v jednom takte iba 8 bitov. Zvýšenie priepustnosti jednoduchým rozšírením zbernice by spôsobilo exponenciálny nárast zdieľaného dekodéru a značne skomplikovalo a zväčšilo stavový automat.

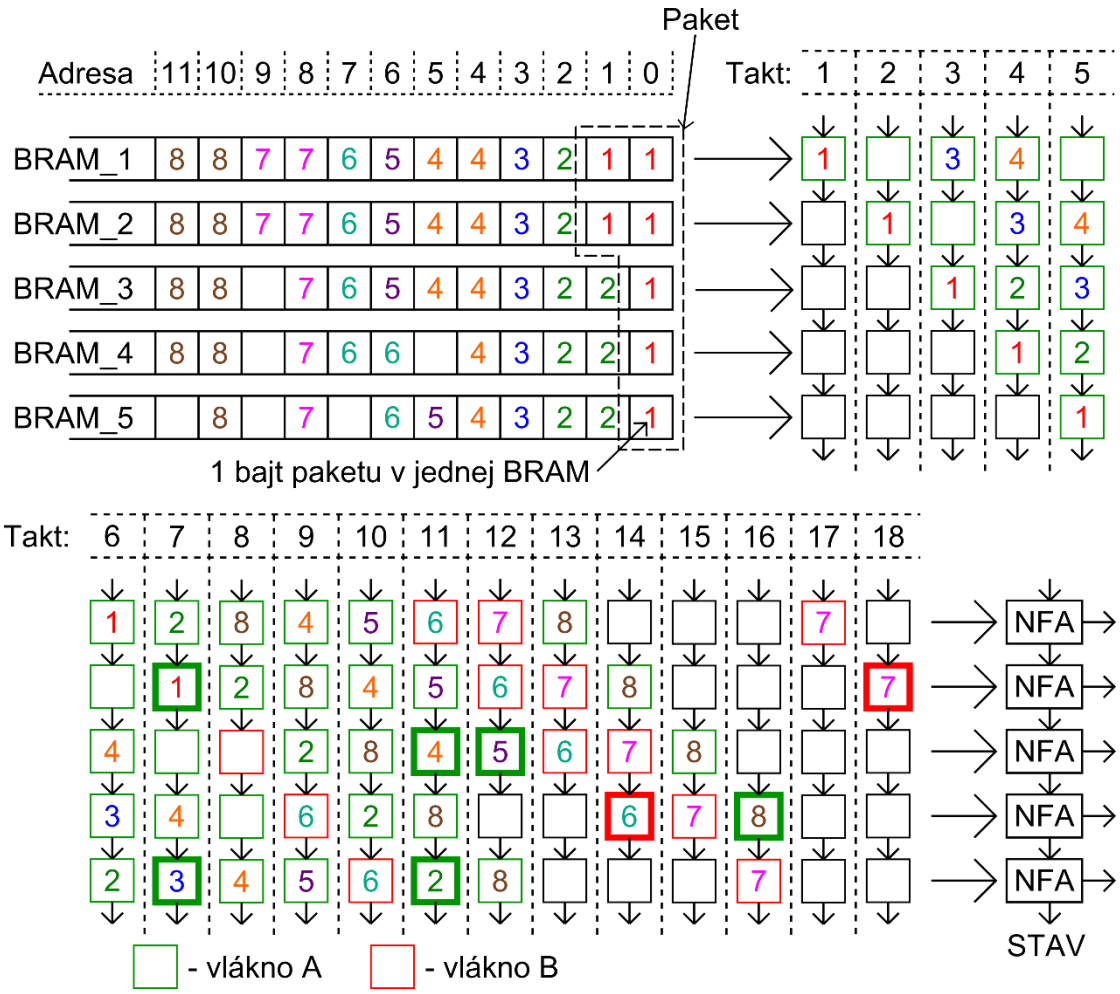
Pre zrýchlenie toku dát je potrebné použiť viac paralelne pracujúcich jednotiek, ktoré budú hľadať výrazy v paketoch. Princíp funkcie je naznačený na obr. 2.1, kde je uložených 5 paketov po jednom bajte v 5 BRAM pamätiach. Veľký tok dát zo vstupu sa ukladá do týchto BRAM pamätí slúžiacich ako vyrovnávacia pamäť, pričom sa rozdeľuje po jednom bajte do každej BRAM. Z nich sa potom číta a naraz spracováva 5 paketov v jednotkách, ktoré hľadajú regulárne výrazy (NFA). Tak sa zabezpečí väčšia priepustnosť a len lineárny nárast počtu NFA.



Obr. 2.1: Zjednodušený princíp spracovávania paketov v architektúre pre detekciu HTTP hlavičiek

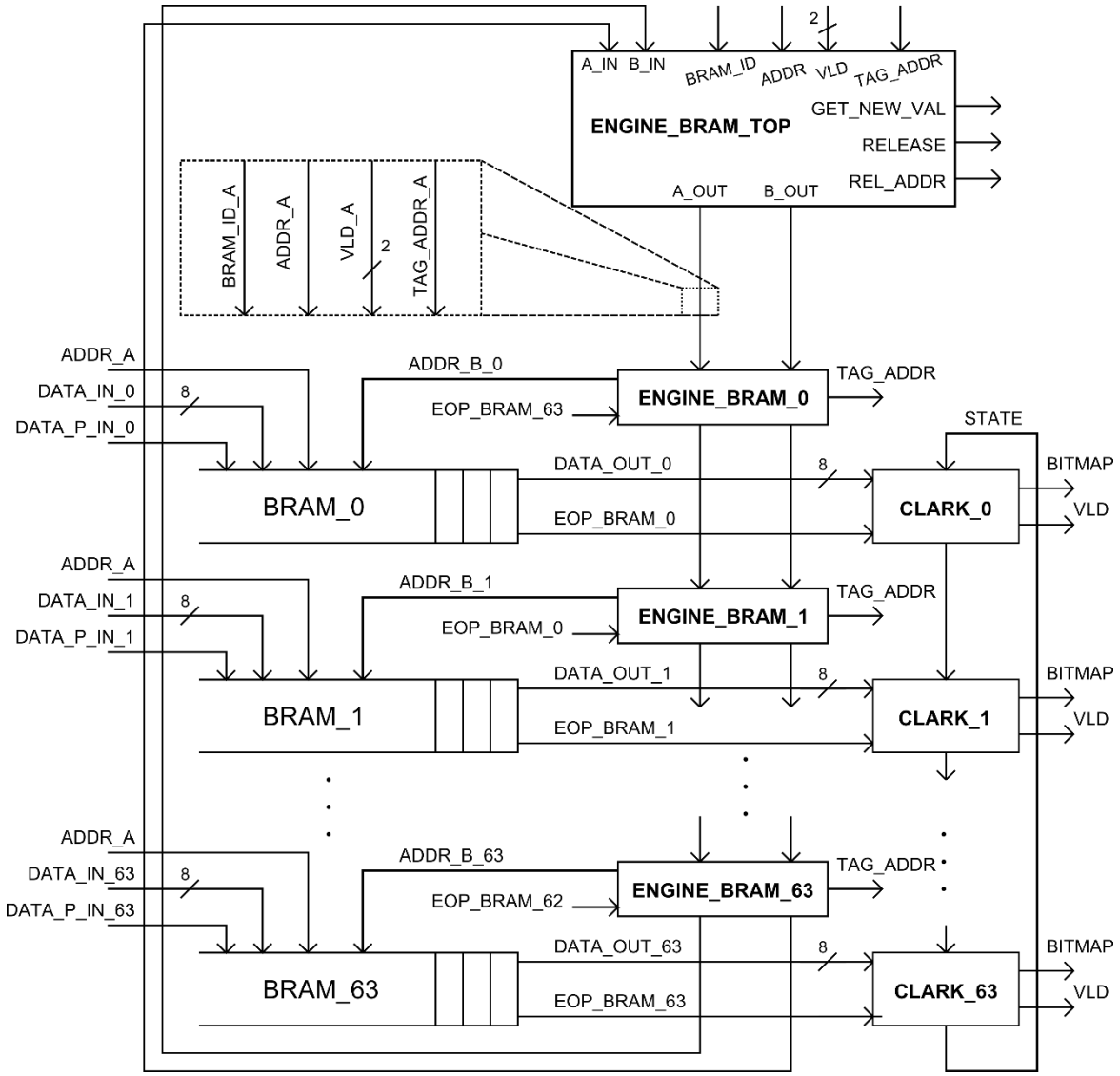
V skutočnosti je funkcia celej architektúry o niečo zložitejšia. Príklad na obr. 2.1 má mnoho nedostatkov. Ak by mala jedna jednotka NFA sama spracovať jeden paket, znamenalo by to množstvo prepojení z každej BRAM do každej jednotky NFA. Navyše by sa musela kontrolovať situácia, aby dve alebo viac jednotiek NFA nečítali naraz z jednej BRAM z rôznych adries.

Riešenie problémov popísaných v predchádzajúcom odstavci poskytuje schéma na obr. 2.2. V tomto prípade jednotky NFA stále pracujú paralelne. Zmena nastala v ich vzájomnom vertikálnom prepojení. Prostredníctvom týchto prepojení si môžu posielať informácie o aktuálnom stave. Umožní sa tak postupné čítanie a spracovávanie paketov s minimom prepojení. Zároveň nemôže nastať situácia, aby viac jednotiek NFA čítalo naraz z jednej BRAM. Na obr. 2.2 vidieť, že sa v jednom takte číta stále iba z jednej adresy danej BRAM. Bajty paketu sa postupne čítajú a posielajú do jednotiek pre detekciu výrazov. Prvý NFA automat v prvom takte spracováva prvý bajt prvého paketu. Následne pošle informáciu o svojom stave do druhého NFA a v druhom takte už spracováva prvý bajt druhého paketu atď. Jednotka, ktorá spracuje posledný bajt paketu pošle informáciu o detekovaných reťazcoch na ďalšie spracovanie. Obrázok je navyše doplnený o možnosti protokolu FLU a o funkciu dvoch vlákien. Pravidlá FLU hovoria, že začiatok alebo koniec paketu môže byť definovaný kdekoľvek v strede zbernicového slova. Funkcia vlákien A a B je vysvetlená v podkapitole 2.3.



Obr. 2.2: Spracovávanie paketov v architektúre pre detekciu HTTP hlavičiek

Na detekciu výrazov v pakete a teda detekciu HTTP hlavičiek sa v dizajne využíva už spomenutý nedeterministický konečný stavový automat so zdieľaným dekodérom. Keďže riešenie zdieľaného dekodéru a poslednú úpravu NFA vymyslel pán Clark, jednotka pre detekciu výrazov sa v dizajne nachádza pod názvom CLARK. Pre docielenie priepustnosti 100 Gb/s je potrebná šírka zbernice 512 bitov. V dizajne je preto použitých 64 BRAM pamätí (obr. 2.3) na ukladanie paketov pri šírke slova zbernice 512 bitov. Blokové diagramy s anglickým pomenovaním blokov a signálov zodpovedajú popisu v zdrojových kódoch. Na obrázku sa pri pamätiach nachádzajú ďalšie riadiace jednotky, ktorých funkcia je vysvetlená v podkapitole 2.2. Zapojenie výsledného dizajnu sa nachádza v prílohe A. Každá pamäť slúži na uloženie jedného bajtu z celého 512 bitového slova. To umožňuje v každom takte čítať a spracovávať naraz 64 bajtov v 64 jednotkách CLARK (obr. 2.3). Priepustnosť dát je tak dostatočná a zložitosť jednotky CLARK sa nezvýšila. Šírka slova je v dizajne určená pomocou generickej konštanty. Pri zmene šírky slova sa mení aj počet použitých BRAM pamätí a jednotiek CLARK.



Obr. 2.3: Pamäte BRAM v architektúre spolu s riadiacimi jednotkami a jednotkami CLARK

2.1 Zápis dát do pamätí

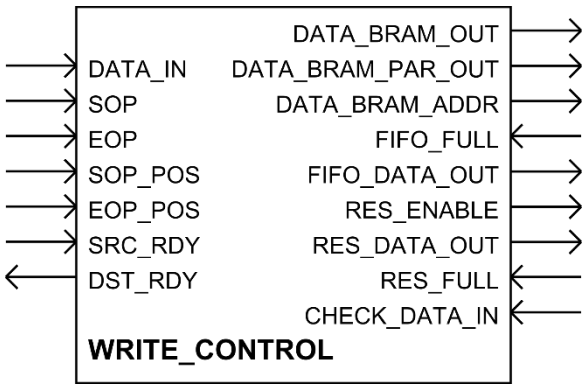
Riadenie zápisu paketov do pamätí BRAM má na starosti jednotka WRITE_CONTROL (obr. 2.4). Pakety prichádzajú pod protokolom FLU. Z jeho rozhrania je možné určiť príchod dát na vstup (signál SRC_RDY je v aktívnej úrovni). Jednotka pri zápise slova do BRAM určuje adresu, na ktorú sa má slovo zapísať. Adresa sa pri zápisoch slov stále zvyšuje až do pretečenia.

Ak je pri zapisovanom slove signál EOP v aktívnej úrovni, zapíše sa do paritných dát jednej konkrétnej BRAM pamäte bit, informujúci o ukončení paketu. BRAM, do ktorej sa paritný bit zapíše je určená pomocou signálu EOP_POS. Tento signál nesie informáciu o presnej pozícii, kde v slove paket končí.

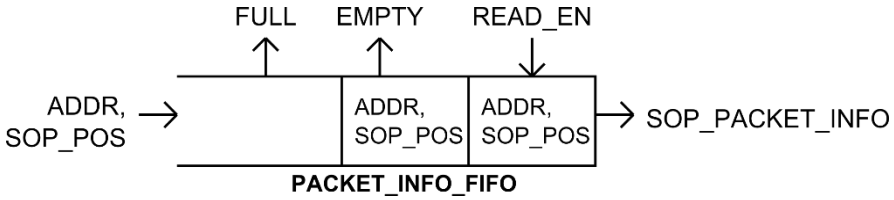
V prípade aktívnej úrovne signálu SOP sa cez port FIFO_DATA_OUT uloží do pamäte PACKET_INFO_FIFO (obr. 2.5) aktuálna adresa zapisovaného slova spolu s informáciou o presnej pozícii začiatku paketu v slove. Signál SOP v aktívnej úrovni znamená, že sa v danom slove nachádza začiatok paketu. Signál SOP_POS informuje o presnej pozícii, kde v slove začína paket.

Adresa s aktívnou úrovňou signálu SOP z rozhrania FLU sa rovnako zapíše do jednotky RESERVATION_UNIT (obr. 2.6) cez port RES_DATA_OUT. V tom istom

hodinovom takte sa nastaví signál RES_ENABLE do aktívnej úrovne. Nastavením tohto signálu sa k adrese zapíše jednobitová informácia o využití tejto adresy v BRAM pamätiach.



Obr. 2.4: Jednotka pre kontrolu zápisu paketov do BRAM pamätí



Obr. 2.5: Pamäť FIFO pre uloženie adresy začatia paketu v BRAM

Signál DST_RDY v aktívnej úrovni znamená povolenie zápisu a prijatie dát jednotkou WRITE_CONTROL. Nastavenie tohto signálu do neaktívnej úrovne pozastaví prenos dát. Na vstupe FLU rozhrania počkajú signály až do obnovenia aktívnej úrovne signálu DST_RDY.

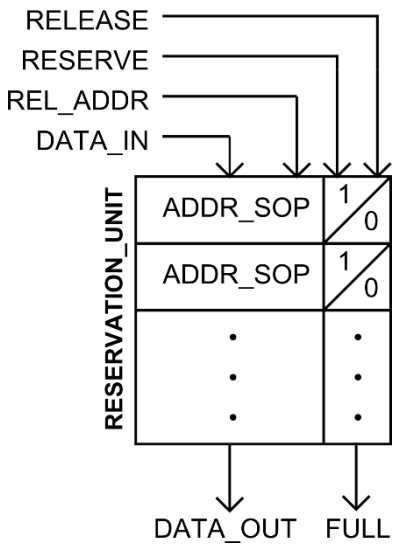
Pozastavenie prenosu zo vstupného rozhrania FLU môžu spôsobiť 3 situácie:

- pamäte BRAM sú plné,
- jednotka RESERVATION_UNIT je plná,
- jednotka PACKET_INFO_FIFO je plná.

O zaplnení dátových BRAM pamätí informuje signál na porte CHECK_DATA_IN, ktorý nesie informáciu o najbližšej využitej adrese v BRAM pamätiach. Tento signál nastavuje jednotka RESERVATION_UNIT, do ktorej sa zapisujú adresy so začiatkom paketov. Ak sa adresa na porte CHCECK_DATA_IN rovná práve zapisovanej adrese (CHECK_DATA_IN = DATA_BRAM_ADDR), na túto adresu sa slovo nezapíše a prenos sa pozastaví.

Signály na portoch RES_FULL a FIFO_FULL informujú o zaplnení jednotky RESERVATION_UNIT a jednotky PACKET_INFO_FIFO. Signál v aktívnej úrovni znamená, že jednotka je plne využitá. Prenos sa pozastaví pri najbližšom štarte nového paketu (SOP v aktívnej úrovni). Do týchto jednotiek sa zapisujú iba začiatočné adresy paketov a tak nie je potrebné prenos pozastavovať v priebehu zápisu paketu ale iba na jeho začiatku.

Údaje v jednotkách PACKET_INFO_FIFO a RESERVATION_UNIT zabezpečujú, okrem iného, neprepisovanie ešte nespracovaných paketov v BRAM pamätiach a zároveň kontrolujú svoje zaplnenie.



Obr. 2.6: Rezervačná jednotka pre rezervovanie adries v BRAM pamätiach

2.2 Čítanie dát z pamätí

Čítanie dát z pamätí BRAM je zložitejšie než zápis. Pri zápise sa jedno 512 bitové slovo zapíše v jednom takte. Na jeho prečítanie a spracovanie je potrebných 64 taktov, pretože toto slovo spracovávajú jednotky CLARK po 8 bitov. V dizajne je však možné naraz spracovávať 64 slov zo 64 paketov. Na zaistenie bezchybnosti spracovania je preto potrebných niekoľko riadiacich jednotiek.

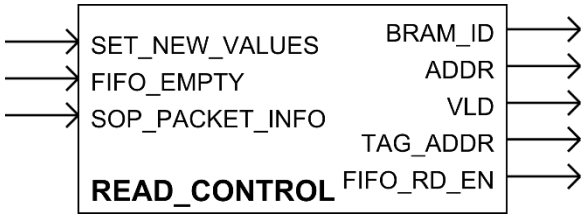
ENGINE_BRAM_TOP na obr. 2.3 má za úlohu:

- preposielať signály vo vláknach A a B ešte nespracovaných paketov,
- nastavovať signály RELEASE a REL_ADDR pre rezervačnú jednotku (RESERVATION_UNIT) na uvoľnenie pamäti už spracovaných paketov,
- prijímať nové riadiace signály BRAM_ID, ADDR, VLD a TAG_ADDR pre spracovanie nových paketov a posielať ich do vlákna A alebo B v prípade nevyužitého taktu,
- určovať príchod nových riadiacich signálov pomocou portu GET_NEW_VAL.

Signály vo vláknach z portov A_IN a B_IN na porty A_OUT a B_OUT nie sú v tejto jednotke oneskorené o hodinový takt, aby nenarušili plynulosť spracovania paketov.

ENGINE_BRAM jednotky na obr. 2.3 sa nachádzajú pri každej BRAM pamäti. Nastavujú adresu čítania z pamätí. O tom, aká adresa bude na výstupe jednotky, rozhoduje signál ADDR vo vláknach. V prípade signálu EOP z paritných dát BRAM pamäte, jednotka ENGINE_BRAM ukončí nastavovanie adresy pre čítanie z BRAM a prepíše signál VLD vo vlákne do stavu DONE (presná funkcia signálu VLD je vysvetlená v podkapitole 2.2.1). Všetky signály vo vlákne spolu s prepísaným signálom VLD postupne prepošlú jednotky ENGINE_BRAM až do jednotky ENGINE_BRAM_TOP bez čítania z BRAM pamätí. ENGINE_BRAM_TOP rozhodne na základe VLD signálu o uvoľnení pamäte v rezervačnej jednotke.

Funkciou READ_CONTROL (obr. 2.6) je sprostredkovať komunikáciu medzi jednotkami PACKET_INFO_FIFO a ENGINE_BRAM_TOP. Jednoducho uložená informácia v pamäti FIFO je spracovaná v READ_CONTROL na 4 riadiace signály poslané do jednotky ENGINE_BRAM_TOP. Jednotka READ_CONTROL prijíma informácie o vyprázdnení pamäte FIFO portom FIFO_EMPTY a príkazy z ENGINE_BRAM_TOP na nastavenie nových riadiacich signálov portom SET_NEW_VALUES. Jednotka si sama nastavuje signál FIFO_RD_EN pre povolenie čítania z FIFO pamäte.



Obr. 2.7: Jednotka nastavujúca signály pre čítanie

2.2.1 Riadiace signály

Riadiace signály rozhodujú o správnom čítaní z BRAM pamätí, uvoľňovaní miesta v rezervačnej jednotke a o stave spracovania paketu. Sú to signály: BRAM_ID, ADDR, VLD a TAG_ADDR. Niektoré z týchto signálov sa počas spracovania paketu prepisujú a menia. O hodnote a stave týchto signálov rozhodujú už vyššie vysvetlené riadiace jednotky. Nové signály do obehu nastavuje jednotka ENGINE_BRAM_TOP. Táto jednotka signály z obehu aj vyradzuje/nahradzuje po spracovaní celého paketu.

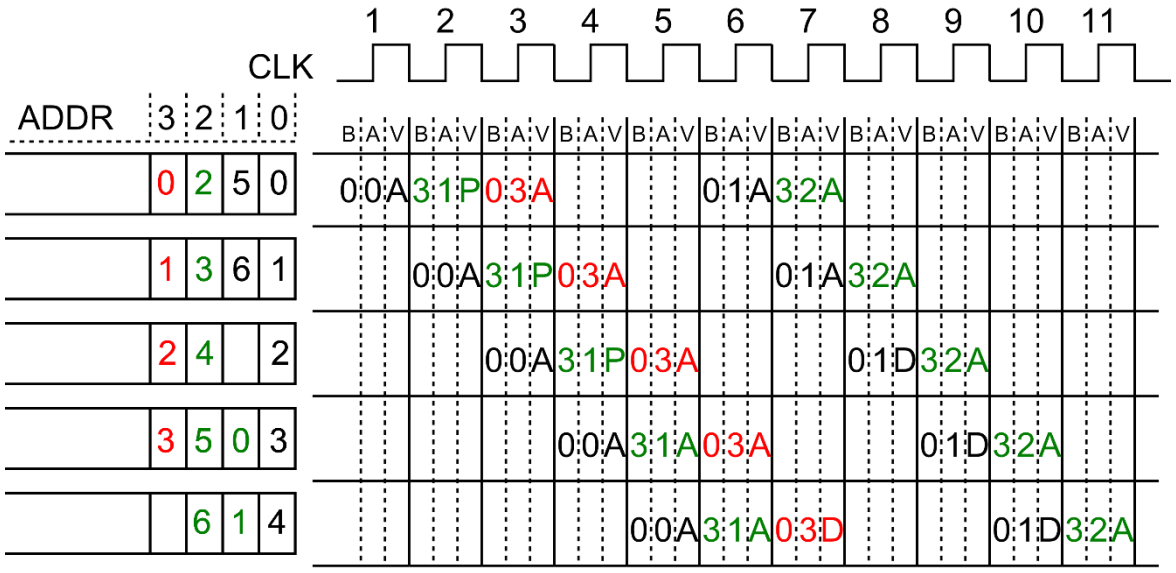
Signál BRAM_ID predstavuje číslo BRAM pamäti, v ktorej začína paket. Signál ADDR určuje adresu pamäte, z ktorej sa bude čítať. Signál TAG_ADDR určuje adresu v rezervačnej jednotke, na ktorej je informácia o využití pamäte daným paketom a signál VLD poskytuje informáciu o aktuálnom stave spracovaní paketu.

Signál VLD má 4 stavy:

- VLD = 0 => NOP
  - nenastavené riadiace signály,
  - z pamäti sa nečíta.
- VLD = 1 => PENDING
  - nastavené riadiace signály,
  - z pamäti sa nečíta,
  - paket čaká na spracovanie,
  - adresa sa pri posielaní od poslednej BRAM k prvej nezvyšuje,
  - ak BRAM_ID = index BRAM prepína sa do stavu ACTIVE.
- VLD = 2 => ACTIVE
  - nastavené riadiace signály,
  - z pamäti sa číta,
  - paket sa spracováva,
  - adresa sa pri posielaní od poslednej BRAM k prvej zvyšuje o jedna,
  - ak v BRAM je v paritných dátach informácia o EOP prepína sa do stavu DONE.
- VLD = 3 => DONE
  - adresa z predošlého stavu ostáva nastavená,
  - z pamäti sa nečíta,
  - paket je spracovaný,
  - signály sa preposielajú až k poslednej BRAM, potom sa v ENGINE_BRAM_TOP prepína do stavu NOP alebo sa nastavia nové hodnoty signálov.

2.2.2 Príklad čítania dát

Príklad čítania paketov z pamätí spolu s riadiacimi signálmi je na obr. 2.8. Na obrázku je naznačených 5 pamätí BRAM. Každá má vyznačené 4 adresy (0 až 3). Tieto adresy sú zaplnené tromi paketmi (čierny, zelený, červený). Napríklad nultý bajt čierneho paketu sa nachádza na adrese 0 v prvej BRAM pamäti, prvý bajt čierneho paketu sa nachádza na adrese 0 v druhej BRAM pamäti, štvrtý bajt zeleného paketu sa nachádza na adrese 2 v tretej BRAM pamäti atď. V tabuľke je ukázané ako sa tieto bajty paketov čítajú. Každý hrubo vyznačený stĺpec tabuľky znamená jeden hodinový takt. V stĺpcoch sú signály B = BRAM_ID, A = ADDR a V = VLD. Keďže signál TAG_ADDR je podstatný iba pre uvoľnenie rezervácie v rezervačnej jednotke a za celú dobu spracovania paketu sa nemení, nie je v tabuľke vyznačený.



Obr. 2.8: Príklad čítania paketov z BRAM pamätí

V prvom takte sa teda nastavil signál BRAM_ID = 0, ADDR = 0 a VLD = **A**CTIVE. To znamená, že čítanie paketu začína v prvej BRAM na adrese nula a spracovanie paketu je aktívne. V druhom takte sa tieto signály pošlú k druhej BRAM a z prvej BRAM je už možné čítať dáta ďalšieho paketu. Ďalší v poradí je zelený paket. Nastavia sa teda signály pre čítanie tohto paketu. Prvý bajt zeleného paketu sa nachádza vo štvrtej BRAM (BRAM_ID = 3) na adrese 1 (ADDR = 1). Signál VLD sa nastaví do stavu **P**ENDING, keďže spracovanie paketu sa má spustiť až vo štvrtej BRAM. Tieto 3 signály sa rovnako posielajú každý takt k ďalšej BRAM. Ak sa signál BRAM_ID rovná indexu BRAM (začiatok paketu) signál VLD sa nastaví do stavu **A**CTIVE a začne sa čítať.

Čierny paket zaplnil všetky BRAM pamäte s adresou 0 a zasahuje aj do adresy 1. Signál ADDR sa musí pri posielaní od poslednej BRAM k prvej zväčšiť o jedna. Signál ADDR zvyšuje svoju hodnotu iba v prípade, že VLD je v stave **A**CTIVE. O zmene adresy rozhoduje ENGINE_BRAM_TOP na základe signálu VLD. Pri spracovaní čierneho paketu budú pokračovať signály BRAM_ID = 0, ADDR = 1 a VLD = **A**CTIVE.

V BRAM, v ktorej paket končí, je v paritných bitoch uložená informácia o ukončení paketu. Tam sa signál VLD nastaví do stavu DONE. Signály sa posielajú postupne až k poslednej BRAM a následne do jednotky ENGINE_BRAM_TOP. Signál VLD v stave **D**ONE znamená, že paket je spracovaný. Jednotka vyšle signál RELEASE na príslušnú adresu REL_ADDR do jednotky RESERVATION_UNIT. Adresu v rezervačnej jednotke,

v ktorej je informácia o rezervovaní miesta v BRAM pamätiach pre paket, nesie signál TAG_ADDR. Tak sa do rezervačnej jednotky zaznačí, že je daný paket spracovaný a môže sa v pamätiach BRAM prepísať.

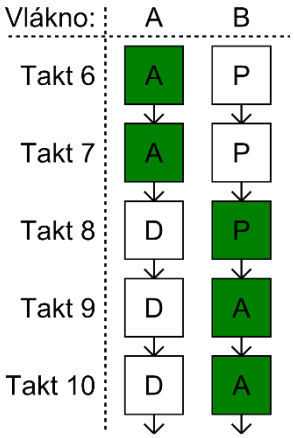
2.3 Systém dvoch vlákien

Na spomalenie priepustnosti má veľký vplyv „nábeh“ a „dobeh“ spracovania skupiny paketov spôsobený zreťazením spracovania. Vtedy sa takty nevyužívajú naplno. Príkladom je prvý takt na obr. 2.2, kedy sa spracováva iba jeden bajt namiesto piatich bajtov naraz, ako je to v deviatom takte. Reálna komunikácia je však značne dlhšia a tak je vplyv tohto faktoru zanedbateľný. Ďalšie spomalenie spôsobujú rôzne dĺžky paketov a medzery medzi nimi. Na zmiernenie tohto vplyvu a urýchlenie spracovania sa používajú dve vlákna (A a B).

Všetky nové riadiace signály nastavuje jednotka ENGINE_BRAM_TOP, ktorá tiež tieto signály odstraňuje z obehu. Ak spracovanie paketu skončí v strede slova, riadiace signály slúžiace pre správne čítanie sa musia postupne preposlať až k jednotke ENGINE_BRAM_TOP, ktorá ich nahradí. Takty, v ktorých sa preposielali riadiace signály už spracovaného paketu, sú v prípade použitia jedného vlákna nevyužité. Preto riadiace signály BRAM_ID, ADDR, VLD a TAG_ADDR kolujú medzi jednotkami ENGINE_BRAM v dvoch vláknach A a B. Zatiaľ čo vlákno A preposiela riadiace signály už spracovaného paketu, riadiace signály vo vlákne B môžu začať spracovanie ďalšieho paketu.

Príkladom urýchlenia je takt 6 na obr. 2.2. V tomto takte sa z prvej BRAM číta bajt prvého paketu, ktorého spracovanie začalo už v prvom takte. Nové riadiace signály by sa v prípade jedného vlákna nemohli nastaviť. Takty 8, 9 a 10 by boli po spracovaní prvého paketu nevyužité a šiesty paket by sa začal spracovávať až v jedenástom takte. V prípade využitia dvoch vlákien sa nové signály nastavia do vlákna B. Spracovanie šiesteho paketu sa spustí už v deviatom takte a jedenásty takt sa môže využiť pre spracovanie iného paketu. Na obr. 2.9 je popisovaná situácia znázornená pomocou stavu signálu VLD vo vláknach. Zelenou je vyznačené aktívne vlákno, používané na čítanie z BRAM pamätí.

Vlákno A je primárne a uprednostňované pred vláknom B. Ak by v oboch vláknach čakali pakety na spracovanie (VLD = **P**ENDING), ako prvý by sa spracoval paket vo vlákne A. Jednotka ENGINE_BRAM_TOP preto v prípade signálu VLD v stave **D**ONE vo vlákne A a signálu VLD v stave **P**ENDING vo vlákne B nenastaví nové signály do vlákna A ale preradí riadiace signály z vlákna B do A a nové signály nastaví do vlákna B. Tak sa zabezpečí, že po spracovaní paketu riadeného vláknom A sa začnú spracovávať signály vo vlákne B a naopak.



Obr. 2.9: Príklad princípu dvoch vlákien

Celý tento systém je dôležitejší a účinnejší s použitím 64 BRAM pamätí. Ak by paket skončil už v druhej BRAM pamäti (EOP_POS = 2), vlákno musí túto informáciu postupne preposlať až do poslednej BRAM kde to zaznamená jednotka ENGINE_BRAM_TOP a nastaví signál RELEASE a REL_ADDR. Počas tejto doby sa signály vo vlákne nemôžu prepísať a vlákno by v tomto takte bolo nevyužité. Nevyužitých by teda bolo 61 taktov čo by malo za následok spomalenie spracovania paketov, pretože 61 taktov by sa spracovávalo o jeden paket menej ako je možné. Spracovávalo by sa 63 paketov v jednom takte namiesto teoreticky možných 64.

2.4 Rezervačná jednotka

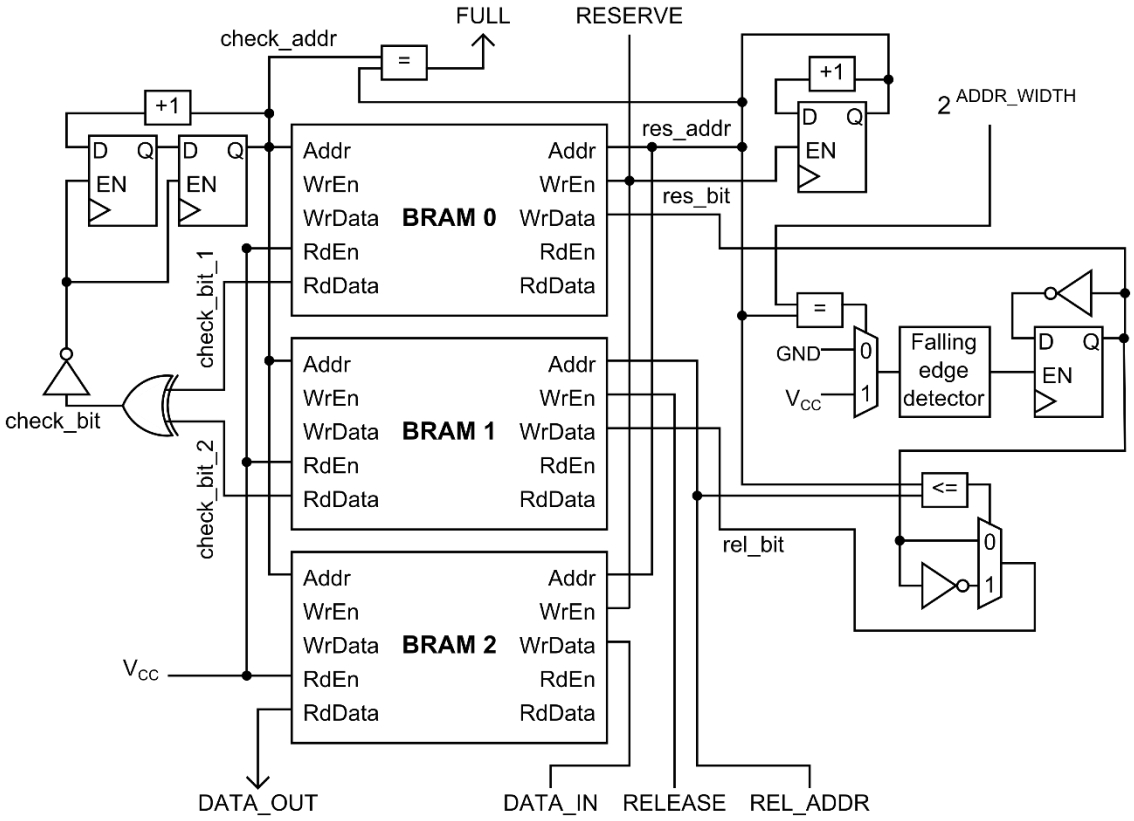
Rezervačná jednotka RESERVATION_UNIT sa skladá z 3 BRAM pamätí. Jedna (BRAM 2) slúži na uloženie adresy dátových BRAM, v ktorej začína paket. Zvyšné 2 slúžia na určenie, či je daná adresa v dátových BRAM využitá, alebo sa paket už spracoval a adresy sa môžu prepísať. Použité sú 3 BRAM pamäte, pretože jednotka musí mať schopnosť v každom takte uskutočniť jednu kontrolu (čítanie), jednu rezerváciu (zápis) a jedno uvoľnenie (zápis). Keďže neexistuje troj-portová pamäť, musí sa použiť zložitejšie zapojenie (obr. 2.10).

Rezervácia každého pamäťového miesta je reprezentovaná dvomi bitmi, každý v inej BRAM (BRAM 0 a BRAM 1). Na začiatku sú všetky bity vynulované. Pri kontrole sa čítajú oba bity naraz. Ak sú rovnaké (0 a 0 alebo 1 a 1), je pamäťové miesto voľné. Kontrola prebieha neustále od adresy 0 až po poslednú adresu. Ak sa natrafí na rezervované pamäťové miesto kontrola sa pozastaví a na výstupe jednotky je pomocou portu DATA_OUT informácia o najbližšie rezervovanej adrese v dátových BRAM.

Rezervácie sú uskutočňované do pamäťových prvkov na daných adresách určené signálom *res_addr*. Signál RESERVE je povoľovací signál čítača, zvyšujúceho adresu na ktorú sa zapisuje. Spolu s týmto signálom v aktívnej úrovni, musí byť vždy na porte DATA_IN hodnota rezervovanej adresy. Rezervácia je uskutočnená zmenou hodnoty bitu v BRAM 0. Rezervačný bit má na začiatku hodnotu logickej jednotky, keďže sú všetky bity v BRAM na začiatku v logickej nule. Po pretečení čítača adresy sa rezervačný bit zmení na logickú nulu. Hodnoty v pamäťových prvkoch sa budú rezervačným bitom prepisovať zase na logické nuly.

Uvoľňovanie rezervácie je uskutočnené zápisom bitu *rel_bit* do BRAM 1, ktorý je rovnaký ako *res_bit*. Hodnoty bitov tak budú v pamätiach opäť rovnaké. Zápis uvoľňovacieho bitu *rel_bit* sa povolí signálom RELEASE a adresa uvoľnenia sa určí signálom REL_ADDR.

Jednotka nemá kontrolu prepisovania už rezervovaných adries. Informuje však inú jednotku signálom FULL. Ak je kontrolná/rezervovaná adresa rovnaká, ako adresa na ktorú sa rezervuje, signál FULL je v aktívnej úrovni. Externá jednotka musí na základe tohto signálu rozhodnúť o pozastavení ďalšieho rezervovania.



Obr. 2.10: Schéma zapojenia rezervačnej jednotky

2.5 Zoradenie paketov na výstupe

Pakety, ktoré prišli na vstup pod protokolom FLU v istom poradí sa počas spracovania rôzne pomiešajú. Spôsobujú to rôzne dĺžky paketov. Na výstupe jednotiek CLARK sú informácie o detekovaných reťazcoch, ktoré je potrebné zoradiť a určiť ku ktorému paketu daná informácia patrí. Návrh jednotky už nie je súčasťou zadania tejto práce a slúži len ako teoretické doplnenie architektúry.

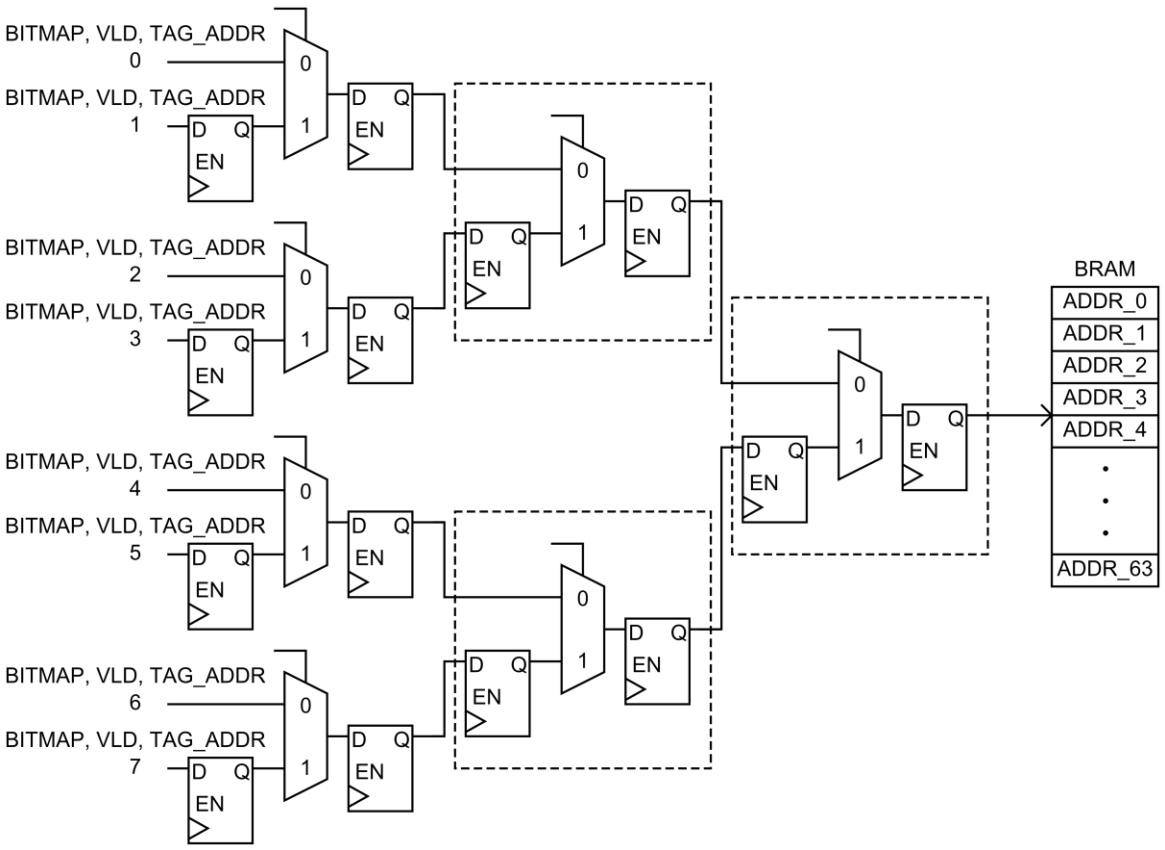
Pre správne zoradenie dát sa využíva signál TAG_ADDR z jednotiek ENGINE_BRAM. Tento signál nesie informáciu o adrese v rezervačnej jednotke, na ktorej je uložená rezervácia pre daný paket. Do rezervačnej jednotky sa ukladajú rezervácie vždy postupne po adresách. Rezervácia sa uskutoční pri začiatku nového paketu. Signál TAG_ADDR teda možno použiť ako signál pre určenie poradia paketov, respektíve na priradenie poradia signálu BITMAP z jednotky CLARK.

S veľkou pravdepodobnosťou môže nastať situácia, kedy sa ukončí spracovanie viacerých paketov v jednom hodinovom takte. V najhoršom prípade sa naraz ukončí spracovanie všetkých paketov. Viacero jednotiek CLARK tak poskytne informáciu o detekovaných reťazcoch v jednom hodinovom takte. Všetky tieto informácie z každej jednotky CLARK je potrebné v tomto takte prevziať ďalšou jednotkou. Následne ich zoradiť a prenechať na ďalšie spracovanie. Architektúra na obr. 2.11 umožňuje splniť túto úlohu.

Základným stavebným prvkom v architektúre na obr. 2.11 je multiplexor s dvoma klopnými obvodmi riadenými povoľovacími signálmi EN. Z tohto stavebného prvku je vytvorený multiplexorový strom. V každom multiplexore je vždy uprednostnený vstup 0. Pokiaľ je na vstupe 0 signál VLD v neaktívnom stave, vyberie sa vstup 1, ktorý obsahuje register pre zachovanie dát na dlhšiu dobu než je jeden hodinový takt. Na konci stromu sa nachádza BRAM pamäť, do ktorej sa ukladajú signály BITMAP na adresy podľa signálu

TAG_ADDR. V pamäti sú zoradené informácie od adresy 0 až po adresu 63 (pri 512 bitovej zbernici). Z týchto adries je čítané pomocou druhého portu postupne po adresách.

Táto architektúra je založená na generických parametroch. Je tak jednoduché upravovať počet vstupov podľa potreby. Architektúra síce spôsobuje ďalšie zvýšenie latencie, no zabezpečuje efektívne spracovanie a zoradenie informácii na výstup bez straty dát. Zapojenie architektúry do výsledného dizajnu sa nachádza v prílohe A.2.



Obr. 2.11: Návrh architektúry pre zoraďovanie paketov na výstupe (príklad pre 8 vstupov)

3 Výsledky návrhu a využitie zdrojov

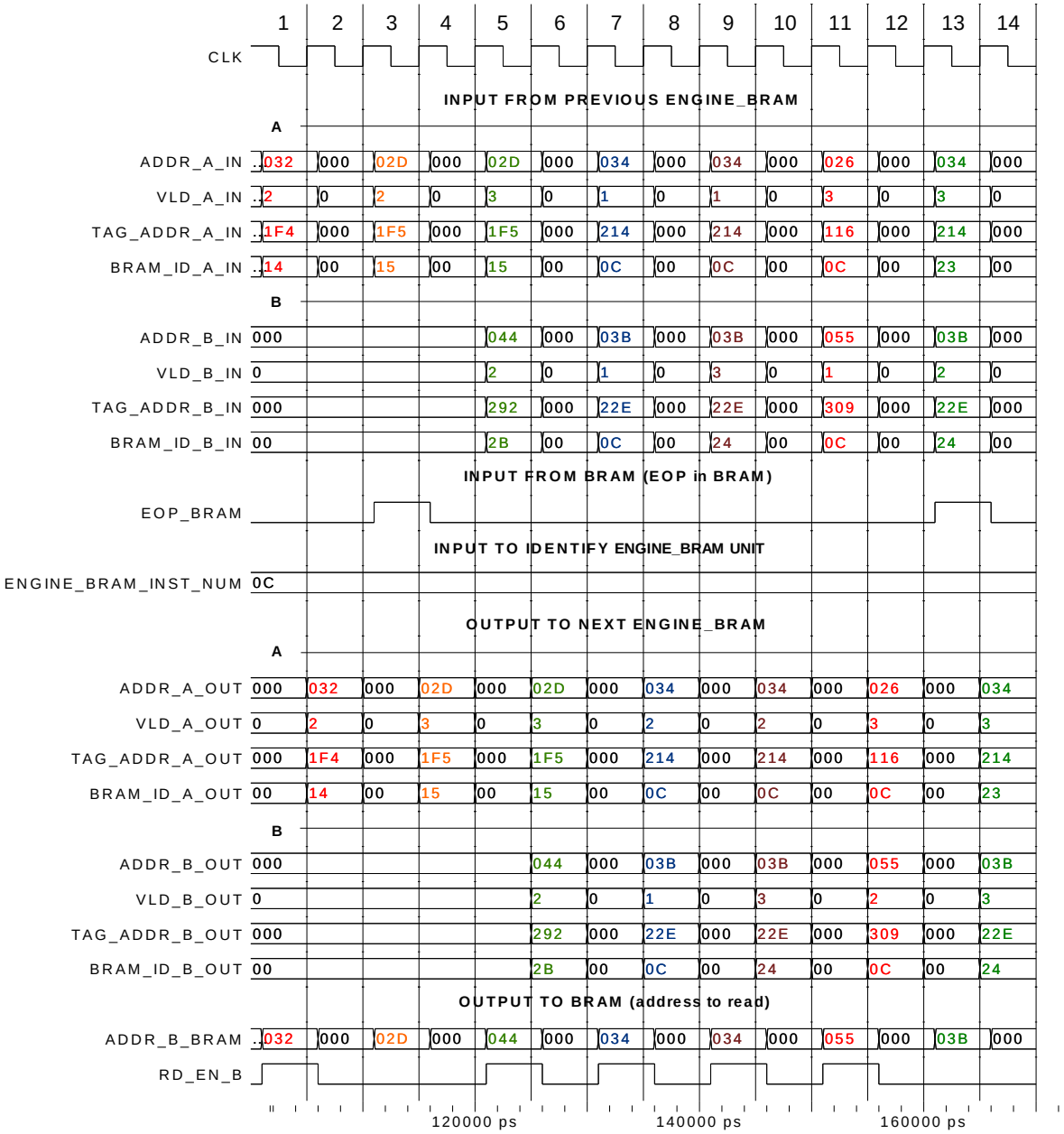
3.1 Simulácia

Po implementácii architektúry bola uskutočnená simulácia pre všetky jednotky samostatne a následne aj simulácia výsledného dizajnu. V tomto dokumente sú ukázané simulácie dvoch najzaujímavejších jednotiek. Sú to jednotky ENGINE_BRAM a RESERVATION_UNIT. Zvyšné simulácie sa nachádzajú na priloženom dátovom CD, kde sú pripravené súbory pre simuláciu v programe ModelSim SE 10.0.

3.1.1 Jednotka ENGINE_BRAM

Na obr. 3.1 sa nachádza zjednodušená simulácia jednotky ENGINE_BRAM. V hornej časti sa nachádzajú vstupné riadiace signály vo vlákne A aj vo vlákne B. Tieto signály spravidla prichádzajú z predchádzajúcej jednotky ENGINE_BRAM, alebo z jednotky ENGINE_BRAM_TOP v závislosti na umiestnení konkrétnej jednotky. Nasleduje vstup z pamäti BRAM, ktorý signalizuje ukončenie paketu (EOP) v danej BRAM pamäti. Signál ENGINE_BRAM_INST_NUM určuje poradové číslo jednotky ENGINE_BRAM. V spodnej časti obrázka sa nachádzajú výstupné riadiace signály vo vlákne A aj vo vlákne B. Tieto signály sú vstupom pre ďalšiu jednotku ENGINE_BRAM. Výstup do BRAM pamäte zastupuje signál ADDR_B_BRAM nastavujúci adresu čítania a signál RD_EN_B povoľujúci čítanie z BRAM. Simulácia na obr. 3.1 nie je kontinuálna. Jednotlivé simulované situácie, ktoré môžu nastať, sú oddelené nulovými signálmi a zvýraznené rôznymi farbami.

O úkone, ktorý jednotka vykoná, rozhoduje signál VLD. V prvom prípade (takt 1) je signál VLD vo vlákne A v stave ACTIVE (hodnota v simulácii 2). Paket je v priebehu spracovania a tak sa číta z BRAM adresy 32. Pri nábežnej hrane hodinového signálu (CLK) sa všetky riadiace signály prepošlú na výstup jednotky nezmenené. V druhom prípade (takt 3) je situácia signálov podobná. Rozdiel je v signále EOP, ktorý je v logickej jednotke. Adresa na čítanie z BRAM je nastavená ale z pamäti sa nečíta, pretože signál RD_EN_B je v logickej nule. Na výstup sa riadiace signály prepošlú so zmenou signálu VLD, ktorý sa nastaví do stavu DONE (hodnota v simulácii 3). V takte 6 je simulovaná situácia, kedy sú riadiace signály v oboch vláknach. Aktívne je vlákno B. V 7. takte je simulovaný prípad uprednostnenia vlákna A pred vláknom B. V oboch vláknach je signál VLD v stave PENDING (hodnota v simulácii 1) a pakety čakajú na spracovanie. Obe vlákna majú rovnaký signál BRAM_ID_IN (0C), čo znamená, že pakety začínajú v rovnakej BRAM ale na iných adresách. Uprednostní sa vlákno A, začne sa čítať z adresy nastavenej v tomto vlákne a signál VLD sa na výstupe zmení do stavu ACTIVE (hodnota v simulácii 2). Signály vo vlákne B ostávajú nezmenené. V ďalších prípadoch sú znázornené rôzne kombinácie signálov VLD vo vláknach.



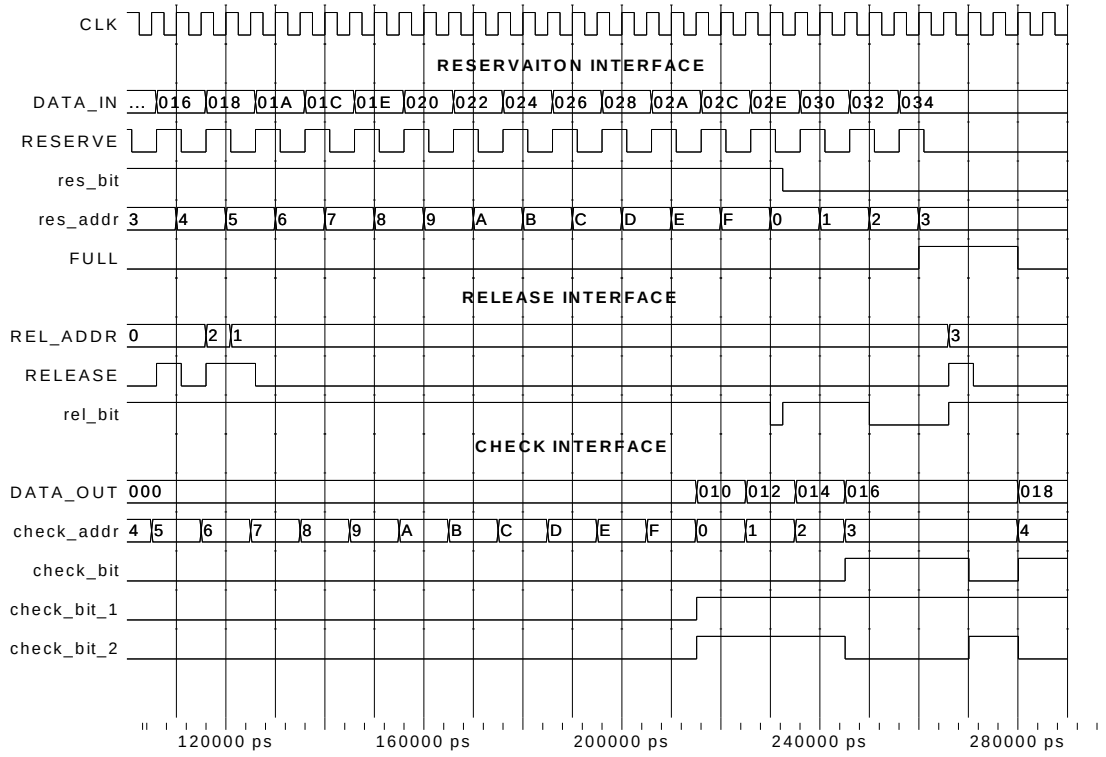
Obr. 3.1: Upravený výstup simulácie jednotky ENGINE_BRAM z programu ModelSim SE 10.0

3.1.2 Jednotka RESERVATION_UNIT

Simulácia rezervačnej jednotky sa nachádza na obr. 3.2. Signály v simulácii sú rozdelené na rezervačné rozhranie spolu so signálom FULL, rozhranie odrezervovania a kontrolné rozhranie. Názvy signálov a portov sa zhodujú so schémou zapojenia na obr. 2.10.

Pri rezervácii sa automaticky nastavuje rezervačný bit (*res_bit*) a rezervačná adresa (*res_addr*). Rezervácia prebieha nastavením vstupných dát (DATA_IN) a povolením rezervácie signálom RESERVE. Pri odrezervovaní je automaticky nastavený signál *rel_bit* na základe signálu *res_bit* a adresy REL_ADDR. Odrezervovanie sa uskutoční nastavením adresy (REL_ADDR), ktorá má byť uvoľnená a povolením uvoľnenia signálom RELEASE. Kontrola zaplnenia prebieha automatickým nastavovaním kontrolnej adresy (*check_addr*). Výsledkom porovnávania signálov check_bit_1 a check_bit_2 je signál *check_bit*. Ak sú signály rovné, potom *check_bit* má nulovú hodnotu. V simulácii prebehlo odrezervovanie až po adresu 2. Kontrolná adresa sa zastavila na adrese 3, kedy je signál *check_bit* v logickej

jednotke. Na výstupe DATA_OUT je hodnota na danej rezervovanej adrese. V tomto dizajne sa tam nachádza adresa začiatku paketu v dátových BRAM pamätiach. Rezervačná adresa, ktorá sa dostala znovu na hodnotu 3 spôsobila zmenu stavu signálu FULL. Po tom, čo sa adresa 3 uvoľní, signál FULL bude znovu v logickej nule a kontrolná adresa pokročí na ďalšiu hodnotu.



Obr. 3.2: Upravený výstup simulácie jednotky RESERVATION_UNIT z programu ModelSim SE 10.0

3.2 Syntéza

Dizajn bol navrhovaný tak, aby zabral čo najmenej zdrojov na čipe FPGA a zároveň poskytol potrebnú prenosovú rýchlosť. Preto sa využili jednoduché nedeterministické stavové automaty pracujúce paralelne. Dáta sú posielané do NFA automatov zo vstupnej vyrovnávacej pamäte pozostávajúcej z BRAM pamätí. Každý zápis, čítanie a posielanie dát má na starosti 6 riadiacich jednotiek. Výsledkom je dizajn s malou veľkosťou na čipe FPGA a s teoretickou prenosovou rýchlosťou až 128 Gb/s. Všetky syntézy boli uskutočnené v programe Vivado 2015.1 pre FPGA typu Virtex-7 H580T.

Tab. 3.1 Využitie zdrojov pre vyrovnávaciu pamäť spolu s kontrolnými jednotkami po syntéze so šírkou zbernice 512 bitov

ZDROJ	VÝSLEDOK SYNTÉZY	K DISPOZÍCII	VYUŽITIE %
LUT	1 338	362 800	0,37
Klopné obvody	2 044	725 600	0,28
Pamäťové LUT	92	141 600	0,06
BRAM	68	940	7,23

V tab. 3.1 je množstvo použitých zdrojov pre pamäte BRAM a riadiace jednotky v prípade použitia 512 bitovej zbernice. Počet použitých LUT tabuliek, klopných obvodov a pamäťových LUT tabuliek je percentuálne veľmi málo. Tieto časti čipu sú využité riadiacimi jednotkami. Najväčšie percento zaberajú BRAM pamäte z ktorých 64 slúži na ukladanie paketov, 3 pamäte využíva rezervačná jednotka a 1 pamäť je použitá v jednotke PACKET_INFO_FIFO.

Tab. 3.2: Využitie zdrojov 64 NFA automatov pre základnú množinu regulárnych výrazov na detekciu protokolov na 100 Gb/s sieti

ZDROJ	VÝSLEDOK SYNTÉZY	K DISPOZÍCII	VYUŽITIE %
LUT	12 494	362 800	3,44
Klopné obvody	9 984	725 600	1,38

Množstvo zdrojov FPGA čipu použitých pre nedeterministické stavové automaty (jednotky CLARK) sa líši podľa množstva a veľkosti použitých regulárnych výrazov. V tab. 3.2 je využitie zdrojov pri použití 64 nedeterministických konečných stavových automatov a základnej množiny šiestich regulárnych výrazov pre detekciu protokolov na 100 Gb/s sieti. Pomocou týchto regulárnych výrazov je možné detekovať HTTP ale aj iné protokoly. Jednotky CLARK sú automaticky generované na základe regulárnych výrazov pomocou skriptu napísaného v jazyku Python. Generovanie jednotiek už nie je predmetom tejto práce. Regulárne výrazy použité na vygenerovanie jednotiek CLARK sa nachádzajú v textovom súbore na priloženom dátovom CD. Príklad použitého regulárneho výrazu, ktorý detekuje hlavičku HTTP požiadavky:

```
/^(OPTIONS|GET|HEAD|POST|PUT|DELETE|TRACE|CONNECT|PATCH) [^\r\n]+
HTTP\/[0-9]\.[0-9]\r\n/
```

V tab. 3.3 sa nachádza využitie zdrojov celej implementovanej architektúry a jej dosiahnuteľná frekvencia. Použitá šírka zbernice je 512 bitov.

Uskutočnená časová analýza v programe Vivado 2015.1 ukázala, že dizajn je schopný pracovať na frekvencii 250 MHz. Z toho vyplýva teoretická prenosová rýchlosť pri šírke zbernice 512 bitov až 128 Gb/s ($250M * 512 = 128\ Gb/s$).

Tab. 3.3: Využitie zdrojov implementovanej architektúry po syntéze so šírkou zbernice 512 bitov a jej dosiahnuteľná frekvencia

ZDROJ	VÝSLEDOK SYNTÉZY	K DISPOZÍCII	VYUŽITIE %
LUT	13 658	362 800	3,76
Klopné obvody	11 998	725 600	1,65
Pamäťové LUT	92	141 600	0,06
BRAM	68	940	7,23
Frekvencia			250 MHz

V budúcnosti je možné zväčšiť dátovú priepustnosť rozšírením vstupnej zbernice alebo zmenšiť dátovú priepustnosť zmenšením vstupnej zbernice pre potreby úspory zdrojov na čipe pokiaľ nie je využívaná prenosová rýchlosť. Umožňuje to zmena jedného generického parametru. Dizajn sa prispôsobí a použije sa potrebný počet BRAM pamätí

a jednotiek ENGINE_BRAM. V tab. 3.4 je množstvo použitých zdrojov pre implementovanú architektúru pri použití rôznych šírok zberníc. Tabuľka ukazuje iba lineárny nárast zabratia zdrojov. Pri zdvojnásobení veľkosti zbernice sa zdvojnásobí aj množstvo použitých zdrojov.

Tab. 3.4: Využitie zdrojov implementovanej architektúry po syntéze pre rôzne šírky zbernice spolu s teoretickou prenosovou rýchlosťou

Šírka zbernice [bit]	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
Teoretická rýchlosť [Gb/s]	**4**	**8**	**16**	**32**	**64**	**128**	**256**
LUT	612	1 043	1 916	3 589	6 977	13 658	26 378
Klopné obvody	500	860	1 616	3 080	6 074	11 998	23 846
Pamäťové LUT	0	22	22	38	70	92	136
BRAM	6	8	12	20	36	68	132

Záver

Táto práca je zameraná na návrh a implementáciu hardvérovo akcelerovanej detekcie HTTP hlavičiek v paketoch. Cieľom je využiť súčasné technológie a vytvoriť hardvérovú architektúru pre detekciu HTTP protokolu v pakete. Dizajn má dosiahnuť priepustnosť dostačujúcu pre vysokorýchlostné siete s rýchlosťou 100 Gb/s.

Pre potreby návrhu architektúry na detekciu HTTP hlavičiek som sa zoznámil s protokolom HTTP a s jeho hlavičkami. Ďalej som si naštudoval princíp funkcie nedeterministického konečného stavového automatu pre detekciu výrazov v pakete, ktorý predstavoval najlepšiu voľbu pre detekciu výrazov z doposiaľ známych možností. Výsledkom je návrh architektúry, ktorý využíva nedeterministický konečný stavový automat so zdieľaným dekodérom a niekoľko BRAM pamätí pre ukladanie paketov a pre riadenie funkcie celého dizajnu.

Paralelne pracujúcimi nedeterministickými stavovými automatmi (NFA) sa dosiahla veľká prenosová rýchlosť. Súčasne sa takýmto zapojením zachovala jednoduchosť NFA a iba lineárny nárast veľkosti zdieľaného dekodéru, ktorý je súčasťou NFA. Ako vyrovnávacia pamäť pred vstupom do NFA jednotiek sú použité BRAM pamäte, pričom počet BRAM pamätí je rovný počtu NFA. Zápis a čítanie z týchto pamätí riadi 6 typov jednotiek. Nastavovaním signálov pre čítanie paketov vždy iba k prvej BRAM pamäti a ich automatickým preposielaním v každom takte k ďalším pamätiam sa ušetrilo množstvo prepojení na čipe FPGA. Dizajn je založený na generických parametroch pre budúce zvyšovanie priepustnosti, alebo prípadné znižovanie priepustnosti, pre potrebu úspory zdrojov na čipe v prípade, že nie je prenosová rýchlosť využívaná.

Jazyk VHDL je použitý pre implementáciu jednotky. Tento jazyk umožnil jednoducho popísať paralelne pracujúci systém. Pomocou simulácie v programe ModelSim SE 10.0 je funkčnosť jednotky overená. Výsledky syntézy z programu Vivado 2015.1 pre čip FPGA typu Virtex-7 H580T ukázali pri šírke zbernice 512 bitov len malú spotrebu zdrojov pre vyrovnávaciu pamäť spolu s kontrolnými jednotkami. Použitie tejto metódy prináša výraznú úsporu zdrojov využitých nedeterministickými konečnými stavovými automatmi, ktoré v dizajne pracujú paralelne. Bolo použitých 64 NFA a 6 regulárnych výrazov pre detekciu protokolov na 100 Gb/s sieti. Následná časová analýza potvrdila schopnosť dizajnu pracovať na frekvencii 250 MHz. Pri použití zbernice so šírkou 512 bitov je dosiahnutá teoretická prenosová rýchlosť 128 Gb/s. Všetky zdrojové kódy dizajnu a kódy pre simuláciu a syntézu sa nachádzajú na priloženom dátovom CD.

V budúcnosti je možné prácu rozšíriť o implementáciu jednotky pre opätovné zoradenie paketov na výstupe. Tiež je vhodné uskutočniť funkčnú verifikáciu a testy na reálnej komunikácii.

Zoznam použitej literatúry

1. CORMER, D. E. *Internetworking with TCP/IP: Principles, Protocols, and Architecture*. 5th edition. Upper Saddle River (New Jersey, USA): Pearson Prentice Hall, c2006. ISBN 0-13-187671-6.
2. DYE, MARK A. MCDONALD, R. RUFI, ANTOON W. *Network Fundamentals: CNA Exploration Companion Guide*. Indianapolis (Indiana, USA): Cisco Press, c2008. ISBN 1-58713-208-7.
3. BARDEN, R. *RFC 1122: Requirements for Intertnet Hosts - Communication Layers* [online]. Internet Engineering Task Force, 1989. Dostupné také z: http://www.rfc-editor.org/pdfrfc/rfc1122.txt.pdf
4. BARDEN, R. *RFC 1123: Requirements for Internet Hosts -- Application and Support* [online]. Internet Engineering Task Force, 1989. Dostupné také z: http://www.rfc-editor.org/pdfrfc/rfc1123.txt.pdf
5. MATTHEWS, J. *Computer Networking: Internet Protocols in Action*. John Wiley & Sons, Inc. 2005. ISBN 0-471-66186-4.
6. ANDREW S. TANENBAUM AND DAVID J. WETHERALL. *Computer Networks*. 5th edition. New Jersey (USA): Prentice Hall, 2011. ISBN 0-13-212695-8.
7. BERNES-LEE, T. ET AL. *RFC 1945: Hypertext Transfer Protocol - HTTP/1.0* [online]. Internet Engineering Task Force, 1996. Dostupné také z: http:// www.ietf.org/rfc/rfc1945.txt
8. FIELDING, ET AL. *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1* [online]. Internet Engineering Task Force, 1999. Dostupné také z: http://www.ietf.org/rfc/ rfc2616.txt
9. KRISHNAMURTHY, BALACHANDER, JEFFREY C. MOGUL, DAVID M. KRISTOL. *Key Differences between HTTP/1.0 and HTTP/1.1* [online]. Dostupné také z: http://www8.org/w8-papers/5c-protocols/key/key.html
10. BELSHE ET AL. *RFC 7540: Hypertext Transfer Protocol version 2* [online]. Internet Engineering Task Force, 2015 [cit. 2015-05-19].
11. BUDINSKÝ, J. *Akcelerace analýzy HTTP provozu*. bakalářská práce. Brno: FIT VUT, 2014.
12. GOYVAERTS, J. *Regular Expressions: The Complete Tutorial*. Lulu.com, 2006. ISBN 1-4116-7760-9.
13. VÝZKUMNÁ SKUPINA ANT AT FIT. *Rychlé hledání regulárních výrazů: Technická zpráva*. Brno: FIT VUT, 2010.
14. V. PAXSON, K. ASANOVIČ, S. DHARMAPURIKAR, J. LOCKWOOD, R. PANG, R. SOMMER, AND N. WEAVER. *Rethinking hardware support for network analysis and intrusion prevention*. Berkeley (CA, USA): USENIX Association, 2006.
15. CHRISTOPHER R. CLARK AND DAVID E. SCHIMMEL. Efficient Reconfigurable Logic Circuits for Matching Complex. In: *Field Programmable Logic and Application, 13th International Conference*. Lisbon (Portugal): 2003.
16. BENÁČEK, P. *Ethernetový tester pro vysokorychlostní sítě*. diplomová práce. Praha: Fakulta informačních technológií ČVUT, 2012.
17. JIŘÍ PINKER, MARTIN POUPA. *Číslicové systémy a jazyk VHDL*.. Praha: BEN - technická literatura, 2006. ISBN 80-7300-198-5.

18. XILINX. *What is a FPGA?* [online]. 2014 [cit. 2014-12-14]. Dostupné z: http://www.xilinx.com/fpga/index.htm
19. *Virtex-7 FPGAS* [online]. XILINX, c2010-2014 [cit. 2014-12-14]. Dostupné z: http://www.xilinx.com/publications/prod_mktg/Virtex7-Product-Table.pdf
20. *7 Series FPGAs Memory Resources: User Guide* [online]. XILINX, 2014 [cit. 2014-12-14]. Dostupné z: http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf
21. *1076-2008 - IEEE Standard VHDL Language Reference Manual* [online]. New York (USA): IEEE Computer Society, 2009 [cit. 2014-12-14]. ISBN 978-0-7381-6853-1. Dostupné z: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4772740&queryText%3DIEEE+Std+1076-2008+.LB.Revision+Of+IEEE+Std+1076-2002.RB.
22. ASBENDEN, P. E. *The Designer's Guide to VHDL*. Third Edition. Burlington: Morgan Kaufmann Publishers, 2008. ISBN 978-0-12-088785-9.
23. *Stránky organizácie CESNET* [online]. [cit. 2014-12-14]. Dostupné z: http://www.cesnet.cz/
24. *Stránky projektu Liberouter* [online]. [cit. 2014-12-14]. Dostupné z: https://www.liberouter.org/
25. CARDS. *Liberouter* [online]. [cit. 2014-12-14]. Dostupné z: https://www.liberouter.org/technologies/cards/
26. *Stránky INVEA-TECH* [online]. [cit. 2014-12-14]. Dostupné z: https://www.invea.com/en
27. Firmware HANIC. *Liberouter* [online]. [cit. 2014-12-14]. Dostupné z: https://www.liberouter.org/technologies/hanic/

Zoznam použitých skratiek

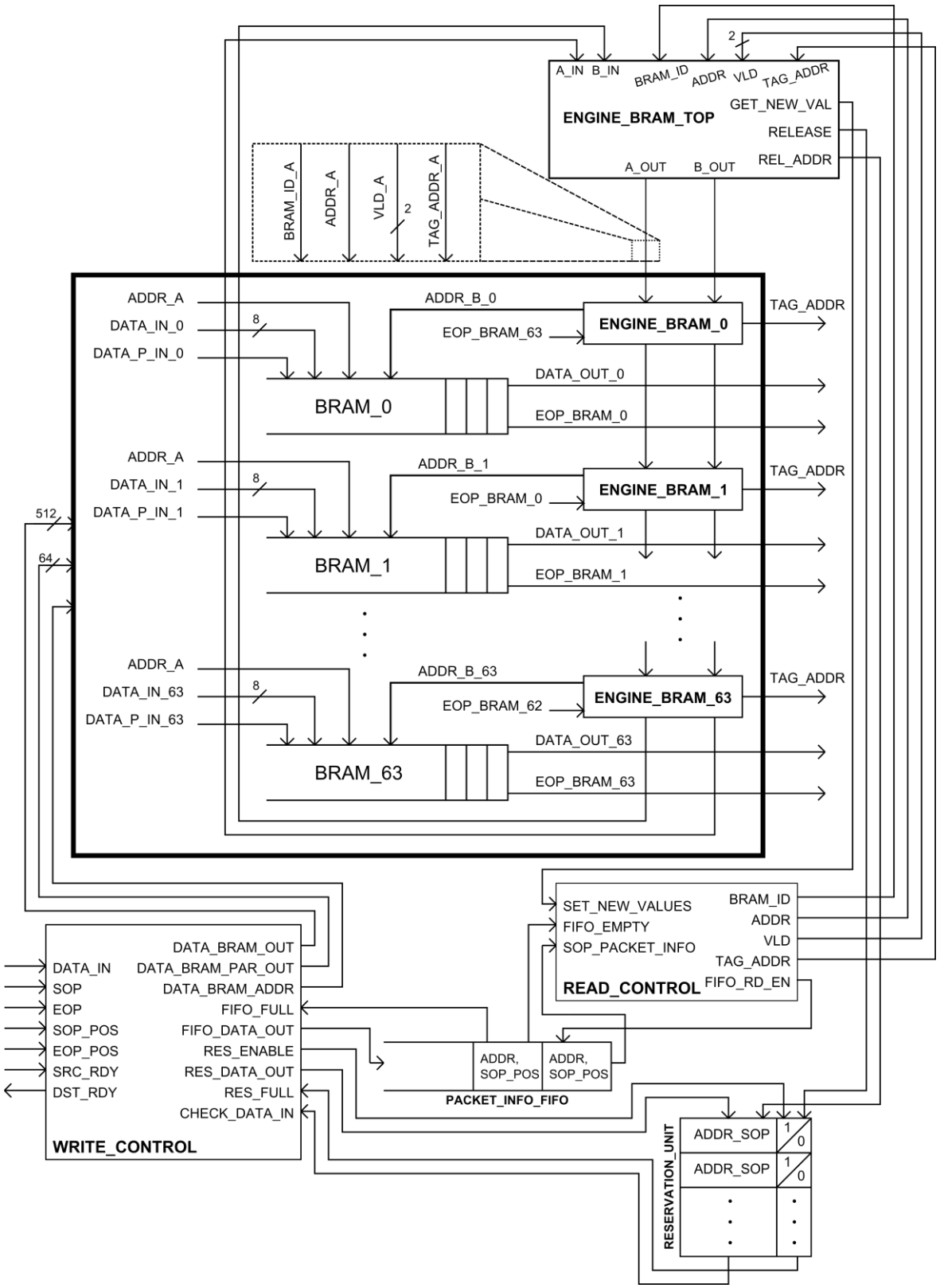
BOOTP	klient/server protokol používaný pre dynamické priradenie rôznych parametrov zo servera v čase načítania **(**Bootstrap Protocol)
BRAM	bloková pamäť RAM používaná v FPGA **(**Block RAM)
CLB	konfigurovateľný logický blok **(**Configurable Logic Block)
DARPA	agentúra pre výskum pokročilých obranných projektov (Defence Advanced Research Projects Agency)
DFA	deterministický konečný automat (Deterministic Finite Automata)
DNS	systém doménových mien (Domain Name System)
EEPROM	elektricky zmazateľná pamäť ROM (Electrically Erasable Programmable Read-Only Memory)
FIFO	vyrovnávacia pamäť **(**First In – First Out)
FLASH	vylepšená pamäť typu EEPROM
FLU	jeden zo základných zbernicových protokolov v projekte Liberouter **(**Frame Link Unaligned)
FPGA	programovateľné hradlové pole (Field Programmable Gate Array)
FTP	protokol určený pre prenos súborov **(**File Transfer Protocol)
HANIC	hardvérovo akcelerovaná sieťová karta **(**Hardware Accelerated Network Interface Card)
HDL	jazyk popisujúci hardvér **(**Hardware Description Language)
HTML	jazyk určený na vytváranie webových stránok (HyperText Markup Language)
HTTP	protokol pre výmenu hypertextových dokumentov vo formáte HTML (Hypertext Transfer Protocol)
IEEE	inštitút pre elektrotechnické a elektronické inžinierstvo **(**Institute of Electrical and Electronics Engineers)
IP	internetový protokol (Internet Protocol)
LSB	najmenej významný bit, bit s najnižšou hodnotou v binárnom vyjadrení čísla (Least Significant Bit)
LUT	náhľadová tabuľka (Look Up Table)

MIME	viacúčelové rozšírenie internetovej pošty (Multipurpose Internet Mail Extension)
NFA	nedeterministický konečný automat (Nondeterministic Finite Automata)
OSI	abstraktný model siete (Open Systems Interconnection)
RAM	pamäť s priamym prístupom (Random-Access Memory)
RARP	protokol slúžiaci pre získanie IP adresy na základe MAC adresy (Reverse Address Resolution Protocol)
ROM	permanentná pamäť (Read-Only Memory)
SMTP	protokol určený pre prenos elektronickej pošty (Simple Mail Transfer Protocol)
SNMP	protokol sieťe pre spravovanie (Simple Network Management Protocol)
SRAM	statická RAM (Static Random Access Memory)
TCP	spojovo orientovaný, spoľahlivý komunikačný protokol transportnej vrstvy (Transmission Control Protocol)
UDP	nespojovaný protokol poskytujúci prenos dát bez zriadenia spojenia (User Datagram Protocol)
URI	jednotný identifikátor zdroja (Uniform Resource Identifier)
VHDL	programovací jazyk popisujúci hardvér (VHSIC Hardware Description Language)
VHSIC	veľmi rýchle integrované obvody (Very High Speed Integrated Circuits)

Zoznam príloh

A Výsledná architektúra

A.1 Vstupná vyrovnávacia pamäť spolu s kontrolnými jednotkami



A.2 Jednotky Clark spolu s architektúrou pre zoraďovanie detekovaných reťazcov