VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ANALÝZA A EXTRAKCE POLOŽEK Z HLAVIČEK PAKETŮ V FPGA

PACKET PARSING AND HEADER FIELD EXTRACTION IN FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ROMAN SELECKÝ
AUTHOR

VEDOUCÍ PRÁCE Ing. JAN KOŘENEK, Ph.D.
SUPERVISOR

BRNO 2016

Zadání bakalářské práce/18099/2015/xselec01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Selecký Roman**
Obor: Informační technologie
Téma: **Analýza a extrakce položek z hlaviček paketů v FPGA**
Packet Parsing and Header Field Extraction in FPGA
Kategorie: Návrh číslicových systémů

Pokyny:

1. Seznamte se s jazykem P4, zejména s částí popisující analýzu a extrakci položek z hlaviček paketů.
2. Nastudujte architekturu FPGA Virtex 7, zaměřte se zejména na platformu Zynq.
3. Navrhněte vhodnou architekturu pro mapování procesu analýzy a extrakce položek z hlaviček paketů do FPGA.
4. Vytvořte program pro mapování popisu v jazyce P4 do navržené architektury.
5. Ověřte korektnost vytvořené implementace a zjistěte dosažené vlastnosti.
6. V závěru diskutujte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

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

Vedoucí: **Kořenek Jan, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

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

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Zariadenia, ktoré sa zúčastňujú komunikácie prostredníctvom počítačových sietí, potrebujú spracovávať pakety a získavať informácie z hlavičiek protokolov. S nárastom počtu protokolov a častými zmienami v ich definíciách, vzniká potreba zaistiť nielen vysokú rýchlosť, ale aj flexibilitu analyzátorov paketov. Preto je cieľom tejto práce vytvoriť návrh jednotky, ktorá umožní jednoducho definovať nové protokoly a generovať výkonnú hardwarovú architektúru. Využitie jazyka P4, popisujúceho hlavičky prokolov a spracovávanie paketov, v kombinácii s možnosťou rekonfigurovať hradlové polia FPGA, nám umožňuje vytvoriť flexibilný analyzátor paketov. Aby bolo možné dostatočne rýchlo premietať zmeny v popise tohto analyzátora, bol vyvynutý program mapujúci popis v jazyku P4 do navrhnutej architektúry.

Abstract

Network devices need to process packets and gather information from header fields. Packet parsers become outdated due to increasing number of protocols and frequent changes in their definitions. This thesis aims to create design of flexible and powerful packet parser. P4 language was designed to define packet processing. Flexible parsers can be constructed by combining potential of P4 with reconfigurable FPGA technology. Program mapping P4 language to designed architecture was implemented in order to promptly reflect changes in parser model.

Kľúčové slová

Analýza paketov, extrakcia položiek, P4, FPGA

Keywords

Packet parsing, header field extraction, P4, FPGA

Citácia

SELECKÝ, Roman. *Analýza a extrakce položek z hlaviček paketů v FPGA*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kořenek Jan.

Analýza a extrakce položek z hlaviček paketů v FPGA

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána doktora Jana Kořenka. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

. . . . . .
Roman Selecký
16. mája 2016

Poďakovanie

V prvom rade, by som rád poďakoval vedúcemu mojej baklárskej práce za všetok venovaný čas a neoceniteľné rady.

© Roman Selecký, 2016.
Táto práca vznikla ako školské dielo na FIT VUT v Brně. Práca je chránená autorským zákonom a jej využitie bez poskytnutia oprávnenia autorom je nezákonné, s výnimkou zákonne definovaných prípadov.

Obsah

1 Úvod **2**

2 Sieťové protokoly **4**
2.1 Vrstvové modely 4
2.2 Ethernet 5
2.3 IPv4 5
2.4 IPv6 6
2.5 TCP 7
2.6 UDP 8

3 Parsovanie hlavičiek protokolov **10**

4 Jazyk P4 **11**

5 Navrhnutá architektúra **14**
5.1 Modul FSM 15
5.2 Modul vstupnej pamäte 17
5.3 Modul výstupnej pamäte 19

6 Generovanie modulu FSM **21**
6.1 FSM_gen 22

7 Testovanie **25**
7.1 Návrh verifikáčného prostredia 26

8 Dosiahnuté výsledky **28**

9 Záver **30**

Literatúra **31**

Prílohy **34**
Zoznam príloh 35

A Simulačný diagram **36**

B Graf Protokolov **38**

C Obsah CD **40**

Kapitola 1

Úvod

Potreba vzájomnej komunikácie počítačov bola zrejmá už v ich prvopočiatkoch. Preto boli vymyslené prvé spôsoby komunikácie medzi zariadeniami. Prvé spôsoby prepojenia počítačov, ktoré pripomínali dnešné siete začali vznikať už v šesťdesiatych rokoch minulého storočia a položili základy celému odvetviu informatiky – sieťovým technológiám. Spočiatku išlo o spojenie štyroch univerzít v USA, známe ako *ARPANET* [21]. Postupom času začali vznikať ďalšie, nielen akademické a vojenské, ale aj súkromné siete. Avšak každá z týchto sietí používala vlastné sady pravidiel na komunikáciu po sieti, tzv. *protokoly*. Z tohto dôvodu nebolo možné siete prepájať a tak naplno využiť potenciál, ktorý tento rýchlo sa rozvíjajúci koncept poskytuje. Zmena nastala po uverejnení článku *"A Protocol for Packet Network Intercommunication"* [13] v roku 1974, ktorý popisoval návrh protokolu a filozofiu zdieľania zdrojov existujúcich v rôznych prepínaných sieťach. Ďalším dôležitým momentom bolo vytvorenie sedemvrstvového abstraktného modelu siete OSI, z angl. *Open Systems Interconnection model* [4], organizáciou ISO, z angl. *International Organization for Standardization*. Každá z vrstiev je definovaná svojou funkcionalitou a spôsobom komunikácie so susednými vrstvami, preto každý protokol spĺňajúci kritéria danej vrstvy, môže byť využitý bez toho aby, ovplyvnil funkčnosť komunikácie samotnej. Od tohto momentu sa začali počítačové siete ešte väčšmi prepájať, čím prilákali ešte väčšie množstvo užívateľov.

Zvýšenie počtu užívateľov, ale aj celkový technický pokrok na poli elektrotechniky, prispel ku zvyšovaniu nárokov kladených na počítačové siete. Tieto nároky sa netýkali len objemu dát, ale aj rýchlosti, akou bolo možné dáta po sieťach prenášať. K tomuto nárastu prispeli aj mnohé sieťové služby, ako napríklad zdieľanie súborov, elektronická pošta alebo rôzne multimediálne služby. Aby bol tento nárast zvládnuteľný a nespôsobil kolaps sietí, musí vývoj aktívnych sieťových prvkov, teda zariadení zabezpečujúcich komunikáciu, napredovať rovnako rýchlo [11]. Aby sa tieto zariadenia dostali k potrebným dátam a mohli tak vykonávať svoju úlohu, musia najprv spracovať hlavičky protokolov, ktoré zodpovedajú nielen za správne doručovanie, ale aj za bezpečnosť dát. Tento proces sa nazýva analýzou hlavičiek protokolov a jeho rýchla a bezchybná funkcionalita, je nevyhnutná pre správny chod každého sieťového zariadenia.

Pri analýzach paketov je možné využiť všeobecné procesory, ktorých inštrukčné sady sú dostatočne obecné, aby túto úlohu zvládli. Nie sú však dostatočne výkonné na to, aby boli schopné spracovávať väčšie dátové toky [ref]. Preto sa využíva hardwarové urýchlenie, ktoré môže realizované viacerími spôsobmi [19]. Jednou z možností je požiť špecializované obvody ASIC, angl. *Application-Specific Integrated Circuit*, ktorých funkcionalita je pevne daná pri výrobe, s čím súvisí aj problém s flexibilitou. Pri ich využití nie sme schopný dostatočne rýchlo reagavať na novovzniknuté protokoly. Alternatívu k ASIC predstavuje technológia

FPGA, angl. *Field Programmable Gate Array*, ktorá umožňuje meniť konfiguráciu obvodu realizujúceho urýchlenie.

Ďalším trendom je enormný nárast počtu protokolov, ktorý je dôsledkom vzniku nových služieb poskytovaných prostredníctvom siete Internet. Taktiež sa nedá vylúčiť ani možnosť zmien v špecifikácii, už existujúcich protokolov. Preto je potrebné prísť so všeobecným spôsobom analýz hlavičiek protokolov a tiež riešením ako meniť konfiguráciu analyzátorov, v už existujúcich zariadeniach.

Riešením je *protokolová nezávislosť* a *rekonfigurovateľnosť*. Protokolová nezávislosť je schopnosť vykonávať akcie, bez nutnosti dopredu vedieť, nad akými protokolmi budú dané akcie vykonávané. Rekonfigurovateľnosť dáva možnosť pridať, zmeniť alebo odobrať schopnosť rozpoznať protokol zariadením, po jeho nasadení.

Protokolová nezávislosť je jedným z pilierov návrhu jazyka P4 [26], slúžiaceho na popis dátových tokov a hlavičiek protokolov. V jazyku P4 sa dá popísať nielen štruktúra protokolu, ale aj vzájomná náväznosť hlavičiek protokolov. Ďalšou z výhod ktoré, jazyk P4 poskytuje je nezávislosť na cieľovom zariadení a technológii, pomocou ktorej je dané zariadenie implementované.

Na základe spomínaných trendov a možných spôsobov ich riešenia, sa ponúka využiť potenciál jazyka P4 v kombinácií s technológiou FPGA. V dnešnej dobe však neexistujú prostriedky, ktoré by umožnili prevod popisu v jazyku P4 na reprezentáciu, ktorú by bolo možné mapovať do hardwaru. Preto je cieľom tejto práce navrhnúť architektúru jednotky vykonávajúcej parsovanie protokolov a implementovať generátor z P4 do jazyka VHDL, ktorý sa využíva na popis hardwaru.

V kapitole 2 sú predstavené vrstvové modely ISO OSI, TCP/IP a štruktúra najpoužívanejších sieťových protokolov. Kapitola 3 je venovaná analýze hlavičiek protokolov a nárokom kladeným na jej implementáciu. Charakteristika jazyka P4 a popis syntaxi, jeho podmnožiny zodpovedajúcej za analýzu paketov, sa nachádza v kapitole 4. Kapitola 5 obsahuje popis architektúry navrhovanej jednotky a bližšie špecifikuje štruktúru a funkcionalitu modulov, z ktorých sa skladá. V kapitole 6 sa nachádza popis programového vybavenia vykonávajúceho preklad z jazyka P4 do jazyka VHDL. V kapitole 7 je popísaný spôsob testovania navrhovanej architektúry a nakoniec je kapitola 8 venovaná analýze dosiahnutých výsledkov.

Kapitola 2

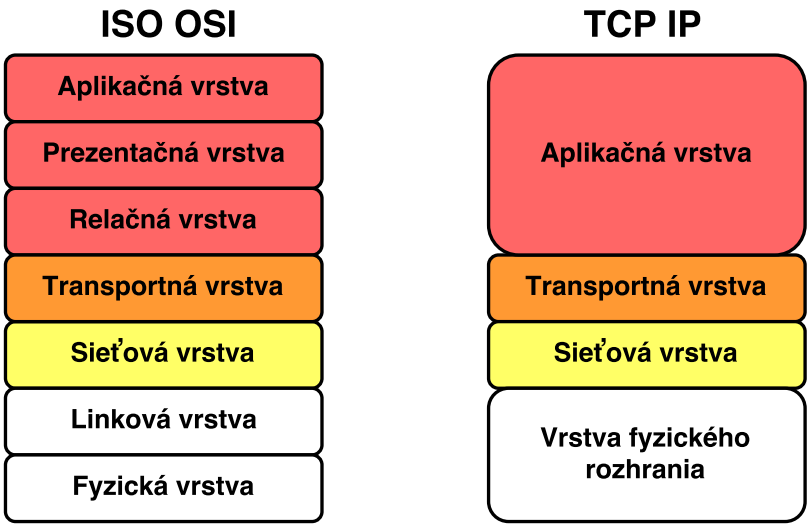
Sieťové protokoly

Protokol [25] je formálnym popisom komunikácie určujúci formát správ a pravidlá, ktoré musia byť dodržané, aby mohli sieťové zariadenia navzájom komunikovať. Formát správ definuje , aké položky sa v správe nachádzajú, aké môžu obsahovať hodnoty a aký význam tieto hodnoty reprezentujú. Pravidlá definujú systém adresovania, mechanizmus nadväzovania a ukončovania komunikácie a spôsob detekcie a spracovávania chýb.

Ak chceme prenášať dáta po sieti musíme ich tzv. *zapúzdriť protokolom*. Pred užitočné dáta, tak pridávame hlavičku, ktorá obsahuje informácie potrebné k správnej činnosti daného protokolu. Niektoré protokoly využívajú aj tzv. *pätičku*, ktorá sa pridáva za dáta a väčšinou slúži na detekciu chýb.

2.1 Vrstvové modely

Pri komunikácií sa nepoužíva iba jeden protokol. Viacero protokolov postupne zapúzdruje dáta podľa tzv. *vrstvových modelov*, čo umožňuje rozdelenie celej komunikácie medzi vrstvy tak, že každá vrstva má jasne definovanú funkcionalitu.



Obr. 2.1: Vrstvy modelov ISO OSI a TCP/IP

Existujú dva významné vrstvové modely. Abstraktný model ISO OSI [4], ktorého cieľom

je prepájať rôzne komunikujúce systémy, pomocou sady štandardných protokolov. Tento model pozostáva zo siedmich vrstiev, pričom žiadna nesmie byť vynechaná. Druhým modelom je TCP/IP [2], ktorý je v rámci celosvetovej siete Internet najvyužívanejší. Obsahuje štyri vrstvy, takže funkcionalita, ktorú tieto vrstvy definujú odpovedá funkcionalite jednej alebo viacerých vrstiev modelu ISO OSI. Na obrázku 2.1 sú znázornené vrstvy oboch modelov tak, že stupeň abstrakcie rastie zdola nahor. Vrstvy, ktoré majú navzájom medzi modelmi rovnakú farbu definujú totožnú funkcionalitu V obidvoch zmienených modeloch využíva každý protokol služby susednej nižšej vrstvy a svoje služby ponúka susednej vyššej vrstve.

Ďalej táto kapitola obsahuje popis najpoužívanejších protokolov, prvých troch vrstiev, z rodiny TCP/IP. Účelom nasledujúcich podkapitol je demonštrovať, na konkrétnych príkladoch, základné princípy uplatňované v protokoloch a štruktúru ich hlavičiek.

2.2 Ethernet

Úlohou protokolu *Ethernet* je adresovať zariadenia v rámci jednej siete a tým zabezpečiť doručovanie správ. Ako identifikátor slúži tzv. MAC adresa, ktorá je priradená sieťovému rozhraniu pri jeho výrobe. Ethernet [1] je najpoužívanejším protokolom vrstvy fyzického rozhrania. Na Obrázku 2.2 je možné vidieť štruktúru správy, ktorá je posielaná od odosielateľa k príjemcovi. Správy na vrstve fyzického rozhrania nazývame *rámce*. Veľkosť polí je uvedená v bajtoch.

Preamble	SFD	Destination address	Source address	Type	DATA	FCS
7	1	6	6	2	46 - 1500	4

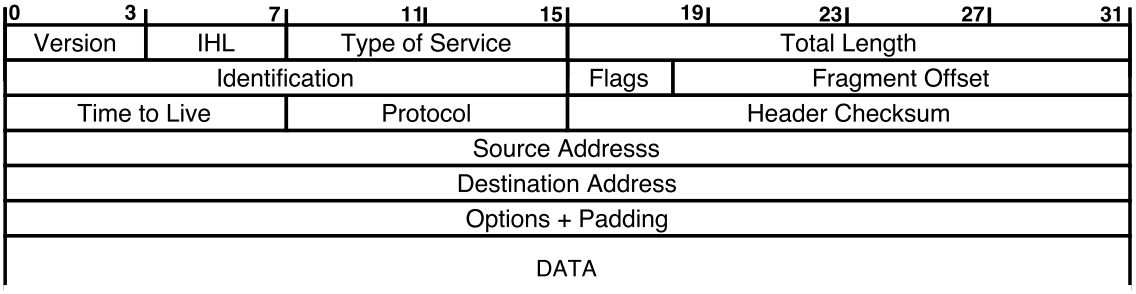
Obr. 2.2: Štruktúra Ethernet II rámca

- **Preable, SFD** obsahujú preddefinované hodnoty, ktoré signalizujú začiatok rámca.
- **Source, Destination address** sú MAC adresy odosielateľa a príjemcu ethernetového rámca.
- **Type** identifikuje zapúzdrený protokol vyššej susednej vrstvy.
- **Data** obsahuje nielen užitočné dáta, ale aj hlavičky protokolov vyšších vrstiev.
- **FCS** je hodnotou vypočítanou kontrolným súčtom a slúži sa na kontrolu správnosti prijatých dát.

2.3 IPv4

Internet Protocol version 4 [6] je protokolom sieťovej vrstvy, ktorá umožňuje existenciu a prepojenie viacerých sietí. Protokol IPv4 zabezpečuje doručovanie správ medzi sieťami. Správy sú doručované na základe IP adries, ktoré sú zariadeniam prideľované po pripojení do siete. IP adresy musia byť unikátne minimálne v rámci siete, pretože identifikujú uzol v sieti. Na Obrázku 2.3 je znázornená štruktúra polí v IPv4 datagrame.

- **Version** identifikuje verziu IP protokolu. V IPv4 hlavičkách obsahuje hodnotu 4.



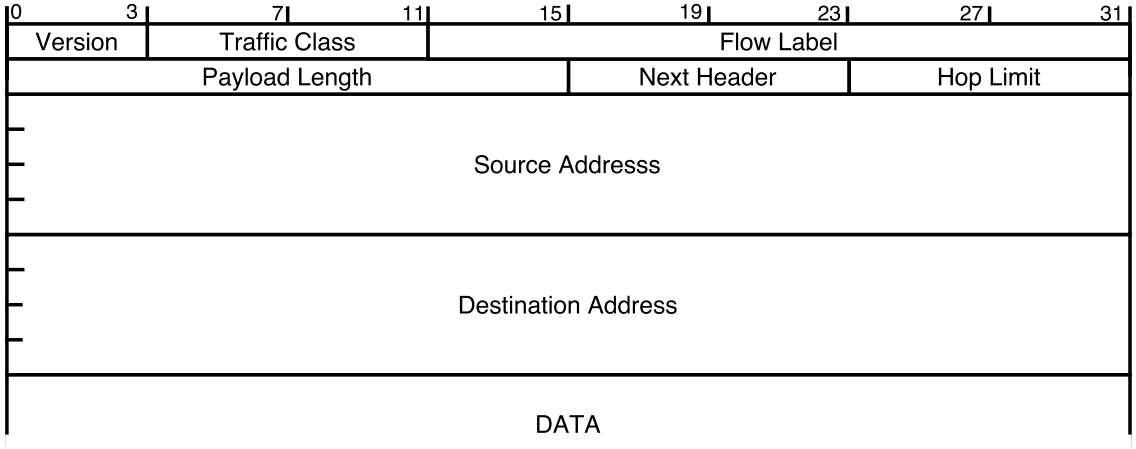
Obr. 2.3: Formát IPv4 hlavičky

- **IHL** určuje veľkosť hlavičky v slovách, 4 bajtových blokoch. Minimálna veľkosť IPv4 hlavičky je 20 bajtov, takže najmenšia hodnota tohto poľa je 5.
- **Type of Service** identifikuje typ služby a môže byť využitý pri zabezpečovaní kvality služieb.
- **Total Length** obsahuje veľkosť celého paketu v bajtoch.
- **Identification** identifikuje fragmenty datagramu.
- **Flags** sú príznaky využívané pri fragmentovaní datagramov.
- **Fragment Offset** určuje polohu fragmentu v datagrame.
- **Time to Live** obsahuje maximálny počet uzlov, ktoré môže k cieľu paket ešte prejsť. Hodnota tohto poľa je v každom uzle dekrementovaná a po dosiahnutí nuly je paket zahodený.
- **Protocol** identifikuje zapúzdrený protokol vyššej susednej vrstvy.
- **Header Checksum** určuje hodnotu kontrolného súčtu, vybraných polí IPv4 hlavičky.
- **Source Address** obsahuje IPv4 adresu odosielateľa datagramu.
- **Destination Address** obsahuje IPv4 adresu príjemcu datagramu.
- **Options** je pole s variabilnou šírkou. Nachádzajú sa v ňom nepovinné položky, ktorých veľkosť je vždy zarovnaná pomocou poľa **Padding** na 32 bitov.

Za poľami hlavičky nasledujú dáta, ktoré môžu byť zapúzdrené hlavičkami ďalších protokolov.

2.4 IPv6

Internet Protocol version 6 [14] je protokolom sieťovej vrstvy, z čoho vyplýva, že plní rovnakú funkciu ako protokol IPv4. Z dôvodu nedostatku IPv4 adries bola štandardizovaná nová verzia, protokol IPv6. Problém s počtom IPv4 adries bol vyriešený zväčšením šírky IPv6 adries na 128 bitov, čo však spôsobilo nekompatibilnosť týchto dvoch protokolov. Medzi ďalšie zmeny patrí pridanie mechanizmu bezstavovej konfigurácie adries, integrovanie *Multicastu* a zvýšenie maximálnej veľkosti paketov, tzv. *MTU* [3]. Tieto zmeny sa prejavili na štruktúre hlavičky protokolu, ktorá je zobrazená na obrázku 2.4.



Obr. 2.4: Formát IPv6 hlavičky

- **Version** identifikuje verziu IP protokolu. V IPv6 hlavičkách obsahuje hodnotu 6.
- **Traffic Class** obsahuje prioritu spracovávania paketu.
- **Flow Label** slúži na zaistenie kvality služieb.
- **Payload Length** určuje počet bajtov prenášaných v tele paketu.
- **Next Header** identifikuje ďalší zapúzdrený protokol. Toto pole odpovedá položke Protocol z IPv4 hlavičky.
- **Hop Limit** plní rovnakú funkciu ako pole Time to Live z IPv4 hlavičky.
- **Source Address** obsahuje IPv6 adresu odosielateľa datagramu.
- **Destination Address** obsahuje IPv6 adresu príjemcu datagramu.

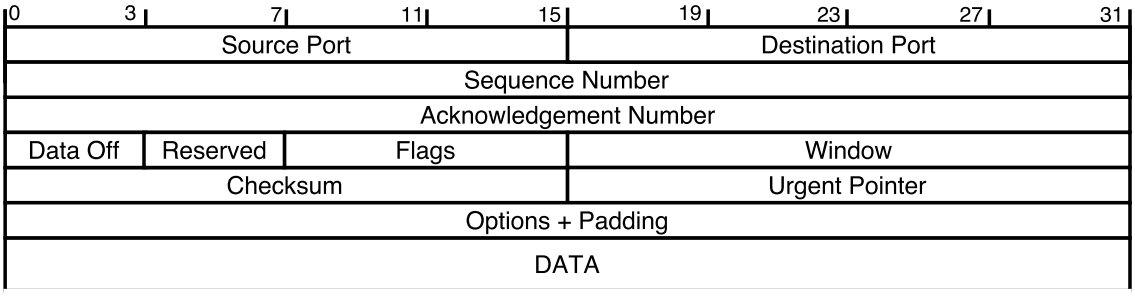
Za poľami hlavičky nasledujú dáta, ktoré môžu byť zapúzdrené hlavičkami ďalších protokolov.

2.5 TCP

Transmission Control Protocol [9] je protokolom transportnej vrstvy, ktorá zabezpečuje komunikáciu medzi aplikáciami zdrojového a cieľového zariadenia. Komunikujúcim aplikáciám sú priradené identifikátory, unikátne v každom zariadení. Na základe týchto identifikátorov, tzv. *portov*, sú prijaté dáta doručené správnej aplikácií.

Prenosu dát prostredníctvom TCP predchádza vytvorenie spojenia, preto je protokol označovaný za *spojovo orientovaný*. Tento protokol garantuje doručenie paketov v správnom poradí [7]. Spoľahlivé doručovanie funguje na základe *pozitívneho potvrdzovania*, tzn. že po prijatí paketu je odosielateľovi poslané potvrdenie o správnom doručení. Paket, o ktorého doručení odosielateľ nedostane potvrdenie, je odoslaný znova. Po odoslaní a správnom prijatí všetkých paketov je spojenie ukončené. Na Obrázku 2.5 je znázornená štruktúra TCP hlavičky.

- **Source Port** identifikuje zdrojovú aplikáciu



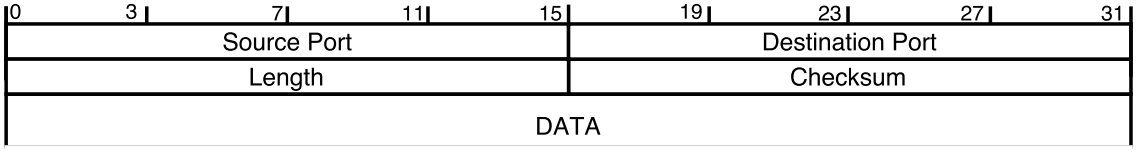
Obr. 2.5: Formát TCP hlavičky

- **Destination Port** identifikuje cieľovú aplikáciu
- **Sequence Number** je číslo identifikujúce pozíciu prenášaných dát, v rámci dátového toku, tohto TCP spojenia.
- **Acknowledgement Number** obsahuje číslo, ktoré odosielateľ paketu očakáva, v poli Sequence Number, v nasledujúcom pakete, ktorý príjme.
- **Data Offset** určuje veľkosť TCP hlavičky v slovách, 4 bajtových blokoch. Minimálna hodnota tohto poľa je 5.
- **Reserved** je zatiaľ nevyužité pole.
- **Flags** obsahuje príznaky signalizácie zahltenia siete.
- **Window** určuje veľkosť dát v bajtoch, ktoré môže odosielateľ paketu prijať.
- **Chechsum** obsahuje hodnotu kontrolného súčtu. Slúži na detekciu chýb vznikajúcich pri prenose paketov.
- **Urgent Pointer** obsahuje sekvenčné číslo posledných urgentných dát.
- **Options** je pole s variabilnou šírkou. Obsahuje nepovinné položky, ktorých veľkosť je vždy zarovnaná pomocou poľa **Padding** na 32 bitov.

Za poľami hlavičky nasledujú dáta, ktoré môžu byť zapúzdrené hlavičkami ďalších protokolov.

2.6 UDP

User Datagram Protocol [5] je protokolom transportnej vrstvy. Prenosu dát prostredníctvom UDP nepredchádza vytvorenie spojenia, preto je protokol označovaný za *nespojovaný*. Oproti protokolu TCP má výrazne menšiu hlavičku, ktorá obsahuje iba polia nevyhnutne potrebné pre doručovanie dát správnej aplikácii. Pri použití protokolu UDP hovoríme o tzv. *nespoľahlivej komunikácii*, pretože neobsahuje mechanizmy zaručujúce doručenie paketov. Pakety môžu byť doručené v nesprávnom poradí, niekoľkonásobne alebo nemusia byť doručené vôbec. Minimalistická hlavička predurčuje protokol UDP na využitie najmä aplikáciami, ktoré sú závislé na rýchlej komunikácií a s prípadnými chybami pri doručovaní paketov sa vedia vysporiadať, ako napr. streaming videí. Na Obrázku 2.6 je znázornená štruktúra UDP hlavičky.



Obr. 2.6: Formát UDP hlavičky

- **Source Port** identifikuje zdrojovú aplikáciu
- **Destination Port** identifikuje cieľovú aplikáciu
- **Length** určuje veľkosť hlavičky a dát, v bajtoch. Minimálna hodnota v tomto poli je 8.
- **Checksum** obsahuje hodnotu kontrolného súčtu.

Za poľami hlavičky nasledujú dáta, ktoré môžu byť zapúzdrené hlavičkami ďalších protokolov.

Kapitola 3

Parsovanie hlavičiek protokolov

Všetky sieťové zariadenia musia byť schopné získavať z paketov dáta, ktoré sú potrebné pre ich fungovanie. Smerovač na základe cieľovej IP adresy rozhodne, ktorým rozhraním odošle paket a osobný počítač môže aplikačné dáta interpretuje napr. ako webovú stránku. Potrebné dáta sa môžu nachádzať v hlavičkách protokolov alebo nimi môžu byť aplikačné dáta, ktoré boli pri odosielaním zapúzdrené.

Proces získavania potrebných dát je inverzný voči zapúzdrovaniu hlavičkami protokolov. Z toho vyplýva, že ako prvé sú spracovávané hlavičky, ktorými sa zapúzdrovalo nakoniec. Prvým krokom pri spracovávaní hlavičiek je *identifikácia protokolu*. Následne môžeme, na základe popisu štruktúry hlavičky, určiť pozíciu jednotlivých polí. Nakoniec keďže poznáme polohu potrebných dát môžeme ich *extrahovať*. Tento proces identifikovania a extrahovania položiek z hlavičiek protokolov sa nazýva *parsovanie* [15].

Návrh a implementácia *parsera*, modulu zodpovedného za spracovávanie paketov, musí spĺňať isté kritériá [15]. Tieto kritériá vyplývajú nielen zo štruktúry protokolov, ale aj z nárokov kladených na sieťové zariadenia, ako také.

- **Sekvenčná závislosť** protokolov vyplýva z ich štruktúry a vrstvového modelu, podľa ktorého sú protokoly zapúzdrované. V hlavičke sa väčšinou nachádza pole, ktoré identifikuje ďalší protokol, čo podporuje sekvenčné chovanie parserov.

- **Nejednoznačnosť** určenia nasledujúceho protokolu je skôr výnimkou, no objavuje sa aj pri často využívaných protokoloch ako je napr. *MPLS*. Ich hlavičky neobsahujú pole, ktoré by identifikovalo nasledujúci protokol. Preto je ďalší postup špekulatívny a využíva sa napr. metóda *lookahead* [20].

- **Premenlivá dĺžka polí** v hlavičkách robí architektúru parsera zložitejšou. Veľkosť týchto polí môže závisieť na hodnote v inom poli alebo na maximálnej dĺžke stanovenej protokolom [17].

- **Priepustnosť** je dôležitým parametrom sieťových zariadení. Keďže Parser je ich súčasťou, rýchlosť parsovania paketov musí odpovedať požiadavkám na priepustnosť. V opačnom prípade sa parser stane úzkym hrdlom celého zariadenia.

- **Flexibilita** je vlastnosť umožňujúca pridať, odobrať alebo zmeniť protokol, ktorý je možné analyzovať. Bez tejto možnosti sa môžu zariadenia rýchlo stať zastaranými a nepoužiteľnými.

Kapitola 4

Jazyk P4

Jazyk P4, celým názvom *"Programming Protocol-Independent Packet Processors"* [12]. Z názvu je možné usudzovať, že sa jedná o jazyk, ktorý slúži na popis protokolovo nezávislých sieťových zariadení. P4 je deklaratívny jazyk popisujúci metódy a pravidlá práce s paketmi, nedefinuje však spôsob vykonávania jednotlivých operácií. Spracovávanie paketov je popísané *abstraktným modelom* pozostávajúcim z parsera a sady pravidiel aplikovaných na pakety pri vstupe a výstupe. Program v jazyku P4 pozostáva z piatich základných logických častí [27]:

- **Headers** obsahuje popis štruktúry protokolov, postupnosť polí v danej hlavičke a ich veľkosť.
- **Parsers** špecifikuje akým spôsobom sú protokoly identifikované a v akom poradí môžu za sebou nasledovať v rámci paketu.
- **Tables** definujú aký typ vyhľadávania v poliach zvoliť a ktoré akcie v prípade zhody vykonať.
- **Acions** obsahuje definície akcií, ktoré môžu byť volané z rôznych častí programu.
- **Control programs** definujú poradie akcií a vyhľadávaní , ktoré budú vykonávané nad paketmi.

Pretože sa táto práca zaoberá analýzami paketov, budeme sa ďalej zaoberať časťami Headers a Parsers, pretože spolu definujú časť abstraktného modelu nazývanú *Parser*, ktorá je zodpovedná za rozpoznávanie hlavičiek prítomných v pakete.

```
header_type ethernet_t {
    fields {
        dstAddr   : 48;
        srcAddr   : 48;
        etherType: 16;
        checksum: 16; }
}
```

```
header_type udp_t {
    fields {
        srcPort : 16;
        dstPort : 16;
        length  : 16;
    }
}
```

Obr. 4.1: Definície hlavičiek protokolov Ethernet a UDP v jazyku P4

Špecifikácie hlavičiek protokolov sa nachádzajú v časti *Headers*. Ako je možné vidieť na obrázku 4.1 definícia hlavičky protokolu začína kľúčovým slovom `header_type`. Povinnou súčasťou tejto definície je zoradená postupnosť polí, uvedená v štruktúre `fields`. Za dvojbodkou, nasledujúcou za názvom každého poľa, sa nachádza číslo určujúce jeho veľkosť v bitoch. Jazyk P4 umožňuje v rámci protokolu označiť šírku jedného poľa, znakom "`*`", za variabilnú. V tom prípade za štruktúrou fields nasleduje výraz určujúci veľkosť celej hlavičky.

Časť *Parsers* obsahuje popis väzieb medzi protokolmi. Tieto väzby sú reprezentované funkciami, ktoré sa navzájom volajú. Na obrázku 4.2 je znázornený popis jednoduchého parsera, začínajúceho vo funkcii start. Definície funkcií začínajú kľúčovým slovom `parser`. Hlavičky všetkých protokolov, ktoré chceme analyzovať, musia byť inštancované pomocou kľúčového slova `header`. Aby bola hlavička prsovaná, v tele funkcie, musí byť s jej inštanciou volaná metóda `extract()`. Na konci funkcie je, za kľúčovým slovom `return`, určená funkcia, ktorá ďalej prevezme riadenia. V prípade, že výber nasledujúcej funkcie je ovplyvnený hodnotou niektorého poľa hlavičky, za kľúčovým slovom return nasleduje syntaktická konštrukcia `select`, tak ako je to vo funkcii parse_eth z obrázku 4.2. Hodnota poľa, s ktorým je select() volaný, bude porovnávaná s konštantami uvedenými pred dvojbodkami vo vnútri konštrukcie select. Prvá zhoda pri porovnaní určí funkciu, ktorá prevezme riadenie. Jej názov sa nachádza za dvojbodkou v select().

```
parser start {
    return parse_eth;
}

header ethernet_t eth;
parser parse_eth {
    extract(eth);
    return select(eth.etherType){
        0x0800 : parse_ipv4;
        0x86dd : parse_ipv6;
        default: end; }
}
```

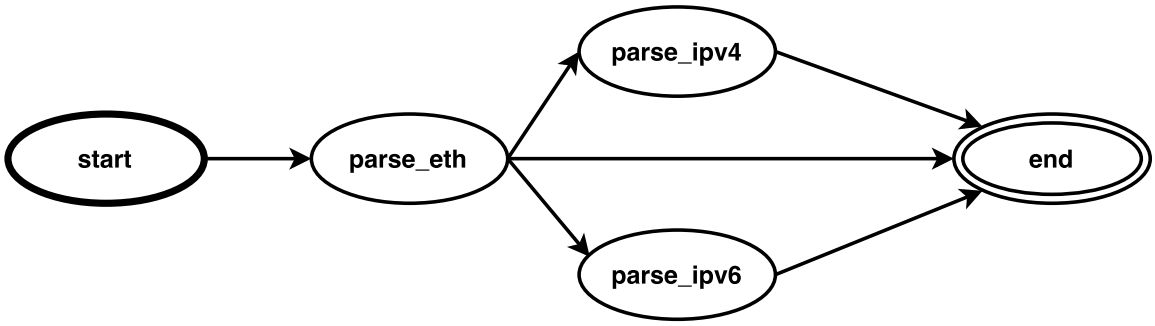
```
header ipv4_t ipv4;
parser parse_ipv4 {
    extract(ipv4);
    return end;
}

header ipv6_t ipv6;
parser parse_ipv6 {
    extract(ipv6);
    return end;
}
```

Obr. 4.2: Definícia funkcií parsera v jazyku P4

Jazyk P4 modeluje analyzátor protokolov ako *konečný automat* [22]. Stavy automatu reprezentujú funkcie, ktoré sú definované v časti Parsers. Konečný automat sa dá znázorniť ako tzv. graf protokolov, angl. *parse graph* [27]. Ide o orientovaný graf, v ktorom je každý stav konečného automatu reprezentovaný uzlom a prechody medzi uzlami sú značené orientovanými hranami. Obrázok 4.3 znázorňuje parse graph odpovedajúci popisu jednoduchého parseru z obrázku 4.2. Počiatočným stavom automatu je *start* a parsovanie paketu je ukončené v stave *end*.

Vzhľadom na potrebu dosiahnuť vysokú priepustnosť, je vhodné pri implementácii jednotky, vykonávajúcej analýzy hlavičiek protokolov a extrakcie polí, využiť hardwarovú akceleráciu. Túto voľbu podporuje aj fakt, že popis parseru v jazyku P4 je možné modelovať ako konečný automat, ktorý sa dá vhodne mapovať do hardwaru. Je však potrebné navrhnúť architektúru jednotky tak, aby výsledná implementácia bola funkčne ekvivaletná s popisom v jazyku P4.



Obr. 4.3: Graf protokolov reprezentujúci konečný automat

Kapitola 5

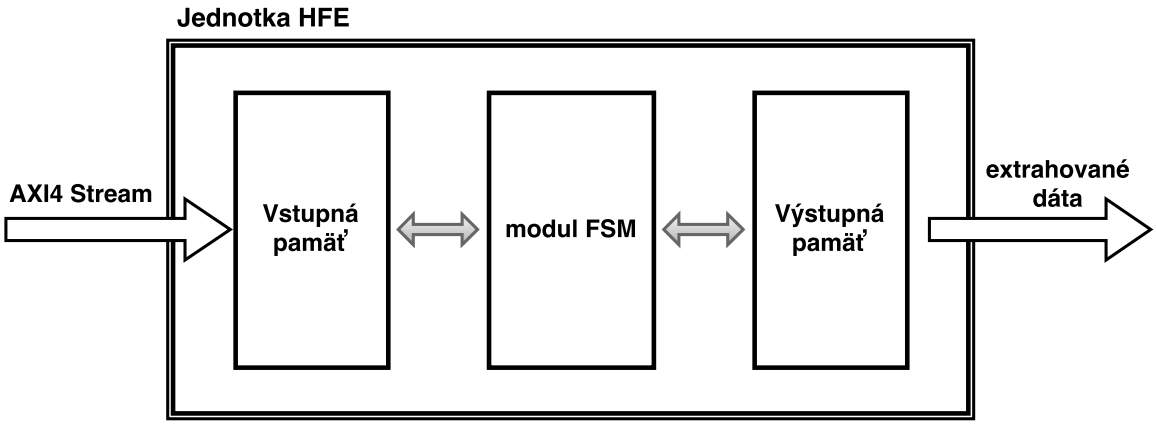
Navrhnutá architektúra

Hlavným cieľom navrhnutej architektúry je extrakcia polí hlavičiek protokolov. Poloha týchto polí, v pakete, však nie je v čase príchodu paketu známa. Analýzou hlavičiek protokolov zistíme polohu týchto polí a ich hodnotu môžeme uložiť do vopred definovanej dátovej štruktúry.

Vstupom jednotky je nepretržitý tok dát vo forme paketov. Pakety sú spracovávané po jednom, z čoho vyplýva, že počas spracovania paketu je HFE zaneprázdnené a nemôže vykonávať žiadnu inú činnosť. V tom prípade by pakety na vstup mohli prichádzať iba v čase, keď nie je spracovávaný iný paket. Preto bola do jednotky HFE zaradená vstupná pamäť, ktorá uchováva pakety prichádzajúce v čase činnosti parseru, kde čakajú na spracovanie. Vďaka tomu môžu pakety na vstup prichádzať nepretržite.

Výstupom jednotky HFE je štruktúra extrahovaných dát. Aby mohlo začať parsovanie ďalšieho paketu, výsledok predošlého spracovania musí opustiť jednotku. Táto skutočnosť by mohla spôsobovať čakanie jednotky HFE na prevzatie extrahovaných dát. Preto bola do jednotky zaradená výstupná pamäť, v ktorej budú výstupné dáta uložené dovtedy, pokiaľ nebudú prevzaté inou jednotkou. Ich preberanie už nebude spôsobovať čakanie medzi spracovávaním paketov.

Na obrázku 5.1 je zobrazené rozdelenie jednotky HFE na tri základné moduly. Pri rozhraniach sa nachádzajú vstupná a výstupná pamäť. Tretím modulom je FSM, ktorý je znázornený v strede jednotky.



Obr. 5.1: Jednotka HFE a moduly, z ktorých sa skladá

Pri prepojení dvoch jednotiek vstupná a výstupná pamäť vyrovnávajú rýchlosť prijí-

macej a odosielacej jednotky, preto je možné jednotku HFE použiť v rámci zreťazeného spracovávania, angl. *processing pipeline*.

Tieto pamäte taktiež tvoria vstupné a výstupné rozhranie navrhovanej jednotky. Keďže vstupným rozhraním prechádzajú dáta vo forme jednosmerného toku deleného na pakety je vhodné ako vstupné rozhranie použiť, v technológií FPGA štandardne používané, rozhranie *AXI4 Stream* navrhnuté pre paketovú komunikáciu. Prenos extrahovaných dát nevyžaduje zložitý komunikačný protokol, preto výstupné rozhranie jednotky pozostáva zo základných signálov potrebných pri čítaní blokov dát, ktoré sú identifikované adresou.

5.1 Modul FSM

Modul FSM je jadrom celej jednotky. Analyzuje hlavičky protokolov a extrahuje z nich položky. Tento modul je kompletne generovaný, na základe popisu protokolov a ich vzájomných prepojení, generátorom popísaným v kapitole 6.

Z pohľadu vzájomnej komunikácie FSM riadi interný chod jednotky pomocou signálov. Isté kombinácie signálov reprezentujú žiadosti o zaslanie dát zo vstupnej pamäte do modulu FSM a žiadosti o zapísanie dát z FSM do výstupnej pamäte. Pri parsovaní hlavičiek nie je nutné prechádzať celé pakety. Žiadosť posielaná vstupnej pamäti vždy obsahuje adresu, na ktorej sa požadované dáta nachádzajú.

Prevod vzťahov medzi protokolmi, na funkčne ekvivalentný konečný automat umožnil mapovať do hardwaru konečné automaty schopné spracovávať pakety. Pri využití konečných automatov je analýza paketu realizovaná prechodmi medzi stavmi automatu, kde stav reprezentuje parsovanie hlavičky protokolu. Spracovanie každého paketu začína počiatočným stavom a končí koncovým stavom automatu.

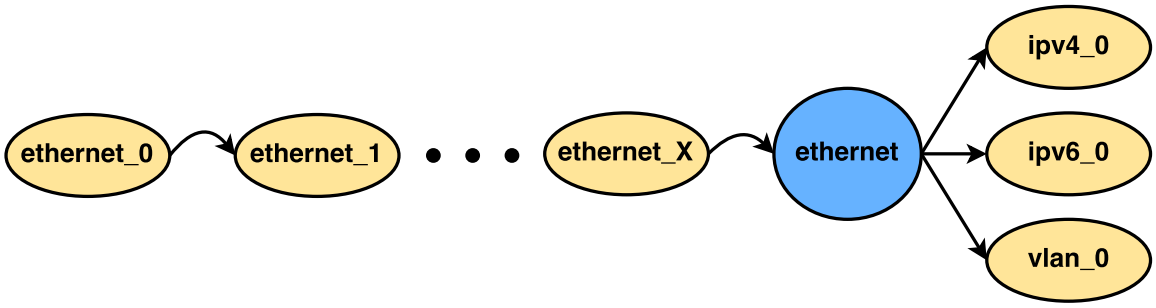
Po prevode konečného automatu do navrhovanej architektúry nemusí byť parsovanie hlavičky protokolu realizované jedným, ale celou postupnosťou stavov. Počet stavov v postupnosti závisí na štruktúre hlavičky a počte čítaní zo vstupnej pamäte, potrebných pri spracovaní hlavičky. Analýzu hlavičky protokolu je možné rozdeliť do dvoch fáz:

- **Fáza načítania dát** začína prvým a končí predposledným stavom postupnosti. V tejto fáze sú stavy prechádzané vo vopred stanovenom poradí. V každom stave fázy načítania dát sa posiela žiadosť vstupnej pamäti o dáta, ktoré sú prijaté a spracované v nasledujúcom stave. Ak prijatá položka obsahuje informácie využiteľné pri identifikácií nasledujúcej hlavičky, bude dočasne uložená v pomocných registroch. Ak sa žiadalo o hodnoty polí určených k extrakcii, odošlú sa do výstupnej pamäte.

- **Fáza vetvenia** prebehne vždy v poslednom stave postupnosti. Na základe porovnania dát uložených v pomocných registroch s hodnotami definovanými v zdrojových súboroch jazyka P4 sa rozhodne, ktorá hlavička nasleduje. Podľa výsledku porovnania sa začne analyzovať ďalšia hlavička. Ak sme práve ukončili spracovanie poslednej hlavičky v pakete, bude vykonaný prechod do koncového stavu automatu.

Obrázok 5.2 zobrazuje časť stavov konečného automatu, zodpovedajúcich za analýzu hlavičky protokolu Ethernet. Stavy znázornené žltým oválom patria do fázy načítania dát a stav reprezentovaný modrým kruhom odpovedá fáze vetvenia.

Konečný automat zodpovedajúci návrhu architektúry obsahuje aj stavy, ktoré sa nezúčastňujú na spracovaní hlavičiek. Tieto stavy riadia rutiny, ktoré sa vykonávajú vždy na začiatku a na konci analýzy paketu. Implementácia týchto stavov sa nemení. Patrí medzi ne počiatočný stav `start` a ukončovacie postupnosti stavov `fin` a `err`.



Obr. 5.2: Postupnosť stavov vykonávajúca parsovanie hlavičky protokolu Ethernet

- Počiatočný stav automatu, riadi inicializačnú rutinu analýz paketu. Keďže parsovanie paketu nie je závislé na predchádzajúcich spracovaniach, táto rutina nuluje hodnoty pomocných registrov. Ďalej zaisťuje, že vstupná pamäť nie je prázdna a výstupná pamäť naopak plná. Pokiaľ nie sú tieto podmienky splnené, konečný automat zotrváva v stave `start`.

- Postupnosť stavov `fin` implementuje vykonávanie rutiny, potom čo je ukončené parsovanie poslednej hlavičky v pakete. Táto rutina zabezpečuje, že do výstupnej pamäte sú uložené metadáta zhromažďované počas analýzy paketu, offset určujúci začiatok dát a bajt označujúci výstupy spracovania paketu za úplné. Nakoniec je zaslaná žiadosť vstupnej pamäti o posun na ďalší paket a výstupnej pamäti o pridelenie nového miesta na ukladanie výstupných dát.

- Druhú ukončovaciu rutinu reprezentuje postupnosť stavov `err`, ktorá je vykonaná v prípade výskytu chyby. Chyba je vyvolaná vstupnou pamäťou, ktorá signalizuje, že dáta na požadovanej adrese sú neplatné. Táto chyba môže nastať v prípade, že sa celý paket nezmestil do vstupnej pamäte, bol orezený a žiadame o dáta, ktoré nie sú uložené. Rovnako aj počas analýzy chybného paketu, kedy taktiež žiadame o dáta, ktoré sa vo vstupnej pamäti nenachádzajú. Spracovávanie paketu nepokračuje ďalej. Do výstupnej pamäti sú zapísané metadáta a offset platný v čase príchodu prerušenia. Ako posledný je zapísaný bajt označujúci výstupy analýz za neúplné. Nakoniec je zaslaná žiadosť vstupnej pamäti o posun na ďalší paket a výstupnej pamäti o pridelenie nového miesta na ukladanie výstupných dát.

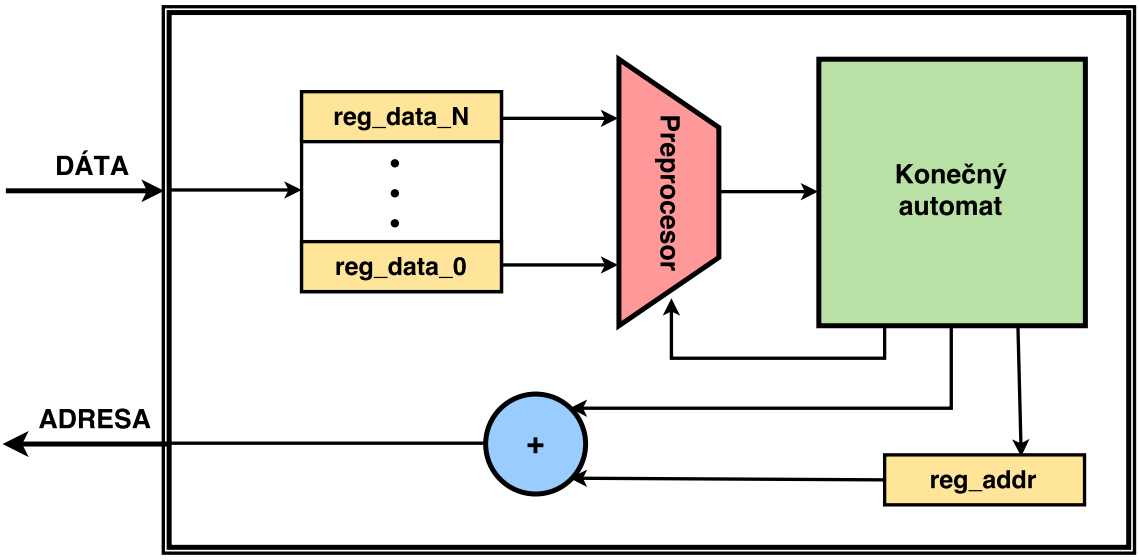
Vždy musí byť vykonaná práve jedna ukončovacia rutina, po ktorej nasleduje prechod do počiatočného stavu konečného automatu.

Súčasťou modulu FSM sú podporné obvody, ktorých úlohou je uľahčiť konečnému automatu prácu s adresami a dočasnými dátami. Na obrázku 5.3 je zobrazená schéma pomocných obvodov zabezpečujúcich komunikáciu cez vstupné rozhranie modulu.

Pri práci s adresami sa využíva register adresy reg_addr. V tomto registri sa uchováva odsadenie, tzv. *offset*, spracovávaného protokolu od začiatku paketu. Táto hodnota je inkrementovaná vždy na konci spracovávania hlavičky protokolu o jej veľkosť. Spracovávanie paketu začína vždy od začiatku, preto je register adresy nulovaný inicializačnou rutinou. Inkrementácia a nulovanie tohto registru je riadené konečným automatom. Adresa, posielaná so žiadosťou o dáta, je sumou hodnoty v registri adresy a offsetu poľa, ktoré chceme čítať. Offset poľa je nastavovaný konečným automatom.

Dáta potrebné pri identifikácií nasledujúcej hlavičky sú dočasne uložené v registroch dát, na obrázku 5.3 označených reg_data. Počet registrov reg_data je závislí na počte

čítaní potrebných dát. Hodnoty z registrov dát sú privedené do preprocesoru, ktorý na základe riadenia konečným automatom, vyberie potrebné časti dát, zoradí ich a nakoniec zkonkatenuje. Takto spracované dáta sú vedené do konečného automatu, ktorý na základe porovnania vo fáze vetvenia zmení svoj vnútorný stav.



Obr. 5.3: Schematické znázornenie modulu FSM

Ďalším typom dočasných dát, uchovávaných v pomocných registroch, sú metadáta ukladané vo forme *bitmapy*. Register, v ktorom sú uložené je nulovaný spolu s registrom adresy. Logická jednotka zapísaná v registri metadát signalizuje, že hlavička príslušného protokolu bola rozpoznaná. Ak by sme tieto informácie chceli zapisovať do výstupnej pamäte priebežne, zvýšil by sa počet zápisov a tým aj stavov automatu, pretože v jednom stave je možné zapisovať práve raz.

5.2 Modul vstupnej pamäte

Do vstupnej pamäte sú ukladané celé pakety, ktoré vstupujú cez vstupné rozhranie ako tok dát. Keďže modul FSM spracováva pakety ako nezávislé celky, sú uchovávané oddelene. Jednotka HFE môže byť súčasťou zreťazeného spracovania, ktoré vyžaduje spracovávanie položiek v poradí ich príchodu. Aj túto funkcionalitu musí zabezpečiť modul vstupnej pamäte. Vzhľadom na požiadavky je vstupná pamäť implementovaná ako dátová štruktúra FIFO, z angl. First In First Out. Pamäťový priestor je rozdelený na bunky rovnakej veľkosti, určené na uchovanie jedného paketu. Ak by bolo pre každý paket alokovanej príliš veľa pamäte, neviedlo by to nielen k jej neefktívnemu využívaniu, ale aj k možnosti útoku na analyzátor protokolov, veľmi komplikovanou kombináciou hlavičiek, čo by mohlo viezť k zníženiu rýchlosti nielen jednotky HFE. Preto bola veľkosť bunky stanovená na 512 bajtov. Pri parsovaní nás zaujímajú iba hlavičky protokolov, s aplikačnými dátami sa nepracuje. Preto ak sa celý paket nezmestí do bunky, jednoducho ignorujeme zvyšok jeho dát. Vstupná pamäť pracuje v režime prepojovania správ, angl. Strore and Forward. To znamená, že pakety môžu byť spracované, až po prijatí ich posledného bajtu.

Vstupná pamäť je konštruovaná z blokov pamätí, ktoré sa nazývajú Block RAM, skrátene BRAM. Tieto bloky sa využívajú v prípadoch, kedy je potrebné uchovávať väčšie ob-

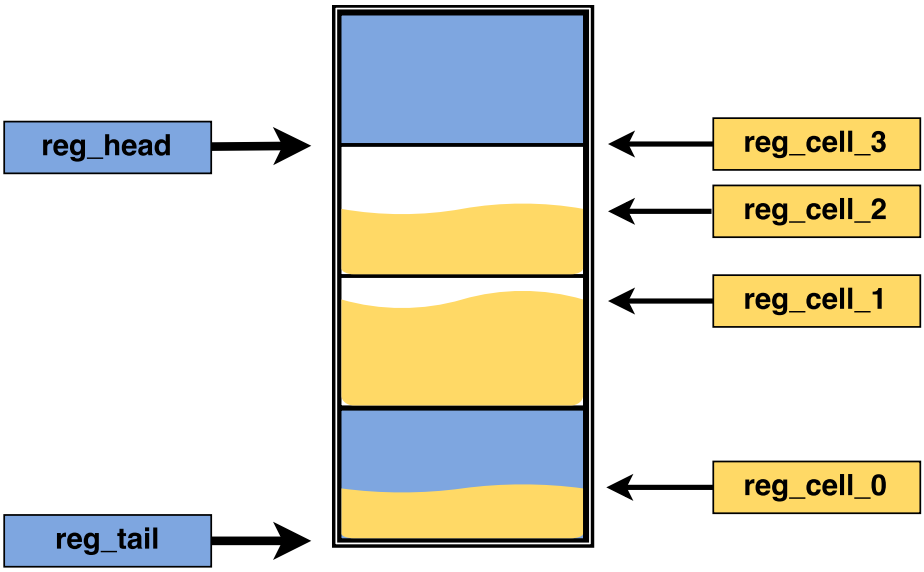
jemy dát. Špecifikom, ktoré prináša využitie tohto typu pamätí, je, že operácie s pamäťou zaberajú jeden hodinový takt. Ich využitie ovplyvňuje architektúru FSM tak, že načítavaná hodnota je dostupná, až v ďalšom stave konečného automatu.

Dáta paketu sú ukladané od začiatku bunky FIFA, v poradí akom prišli. Ukladanie do bunky je ukončené signalizáciou konca paketu.

Výstupné rozhranie je spojené so vstupným rozhraním modulu FSM. Ako už bolo spomenuté v kapitole 5.1, rôzne kombinácie signálov prepájajúce FSM so vstupnou pamäťou reprezentujú žiadosti. Prvým typom žiadostí sa žiada o sprístupnenie dát z konkrétnej adresy. Druhým typom žiadostí je riadené FIFO.

Fungovanie všetkých spomínaných operácií, zabezpečujú podporné obvody a pomocné registre. V registroch sa uchovávajú adresy a okolitá logika zabezpečuje správnu prácu s nimi.

Prvým typom adries, ktoré sa uchovávajú, sú adresy zápisu. Žltou farbou je na obrázku 5.4 vyznačené zaplnenie jednotlivých buniek FIFA a pomocné registre, ktorých obsah definuje mieru tohto zaplnenia. Pre každú bunku FIFA existuje jeden register reg_cell, obsahujúci adresu prvého voľného bloku pamäte v danej bunke. Pred prvým zápisom je obsah v registri vynulovaný, aby nedošlo k použitiu neplatných hodnôt. Pred každým zápisom sa hodnota registru, tzn. adresa, na ktorú sa bude zapisovať, porovná s hraničnou adresou. Hraničná adresa je najväčšia adresa, na ktorú je možné zapisovať dáta Ukladanie dát na väčšie adresy bunky by spôsobilo pretečenie registra reg_cell a ďalšie zápisy by prepisovali už uložené hodnoty. Pokus o zápis za hraničnú adresu je ignorovaný. Po každom úspešnom zápise je hodnota registru reg_cell inkrementovaná. Táto adresa však neslúži iba pri zápise. Pri čítaní je hodnota z registru porovnávaná s adresou, z ktorej modul FSM žiada dáta. Ak je hodnota v registri väčšia ako adresa zo žiadosti, znamená to, že dáta na žiadanej adrese sú platné. V opačnom prípade je informácia o tejto chybe poslaná modulu FSM.



Obr. 5.4: Vstupná FIFO pamäť s pomocnými registrami

Druhým typom adries, ktoré sa v moduly vstupnej pamäte uchovávajú, sú adresy zabezpečujúce správne fungovanie FIFA. Jedna adresa ukazuje na prvú voľnú bunku FIFA, tzv. hlavička, uchovávaná v registri reg_head. Druhá ukazuje na prvú bunku s platným obsahom, tzv. päta, uchovávaná v registri reg_tail. . Hlavička špecifikuje bunku, do ktorej

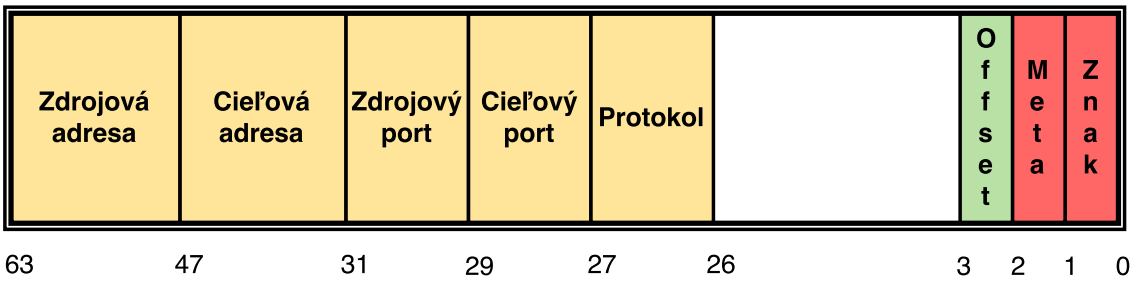
sa má zapisovať a päta bunku, z ktorej sa má čítať. Na obrázku 5.4 sú modrou farbou označené bunky, ktoré sú identifikované obsahom týchto dvoch registrov. Adresa zápisu a rovnako aj adresa posielaná z modulu HFE identifikujú pamäťové miesto v rámci bunky, preto ak chceme byť schopný adresovať celé FIFO, musíme ku adresám pripočítať hodnotu hlavičky alebo päty vynásobenú veľkosťou bunky FIFA.

5.3 Modul výstupnej pamäte

Výsledkom spracovania každého paketu, modulom HFE, sú extrahované položky z hlavičiek paketov. Výstupy spracovania paketov sú uložené oddelene. Zároveň mechanizmus práce s výstupnou pamäťou musí zachovávať poradie, v ktorom boli jednotlivé výstupy ukladané, tak aby boli čítané z pamäte v rovnakom poradí. Preto je výstupná pamäť implementovaná ako abstraktná dátová štruktúra FIFO.

Položky, ktoré sú výsledkom spracovávania paketu, sa delia do troch skupín. Položky prvej skupiny sú tvorené poliami extrahovanými počas analýz jednotlivých hlavičiek. Položky, ktoré budú extrahované je možné došpecifikovať, ale základnou množinou je tzv. pätica, obsahujúca päť položiek, podľa ktorých môžu byť identifikované dátové toky. Do druhej skupiny patrí položka Offset, ktorá označuje začiatok dát zaobalených treťou vrstvou a využíva sa napr. ako štartovacia pozícia analýz protokolov vyšších vrstiev. Položky tretej skupiny obsahujú pomocné dáta. Položka Metadáta obsahuje informácie o identifikovaných hlavičkách v pakete a položka Príznak určuje, či bolo parsovanie paketu úspešne dokončené.

Každá položka má v bunke výstupnej pamäte vyhradené miesto a na obrázku 5.5 je znázornená štruktúra uloženia týchto položiek. Žltou farbou sú označené prvky pätice a zelenou Offset. Červenou farbou sú znázornené položky tretej skupiny, pričom pole Meta označuje Metadata a Znak zastupuje položku Príznak. Veľkosť bunky FIFA, 64 bajtov, stačí na uloženie všetkých výstupných dát a necháva voľný priestor pre prípad extrakcií ďalších polí.



Obr. 5.5: Štruktúra dát uložených v bunke výstupnej pamäti

Pätica sa skladá z týchto položiek:

- **Zdrojová adresa:** Získava sa extrakciou poľa Source Address nachádzajúceho sa v hlavičkách protokolov sieťovej vrstvy, konkrétne IPv4 a IPv6.
- **Cieľová adresa:** Získava sa extrakciou poľa Destination Address nachádzajúceho sa v hlavičkách protokolov IPv4 a IPv6.
- **Zdrojový port:** Získava sa extrakciou poľa Source port nachádzajúceho sa v hlavičkách protokolov transportnej vrstvy, konkrétne TCP a UDP.

- **Cieľový port:** Získava sa extrakciou poľa Destination port nachádzajúceho sa v hlavičkách protokolov TCP a UDP.
- **Protokol:** Získava sa extrakciou poľa Protocol z hlavičiek protokolu IPv4 a poľa Next Header z hlavičiek protokolu IPv6.

V protokole IPv4 a IPv6 majú polia, zdrojová cieľová IP adresa, rôznu veľkosť. Vo výstupnej pamäti je alokované miesto pre adresy protokolu IPv6, ktoré sú väčšie. Ak sa extrahujú IPv4 adresy, tak sú uložené na začiatok alokovaného miesta, pričom zvyšok pamäti ostane nevyužitý. Z metadát je možné neskôr určiť, ktorý sieťový protokol bol rozpoznaný tj. akú veľkosť majú uložené IP adresy. Ostatné prvky pätice majú rovnakú veľkosť naprieč všetkými protokolmi, v ktorých sa vyskytujú.

Fyzicky je výstupná pamäť zložená z vyhľadávacích tabuliek angl. Look up tables, skr. LUT. Tieto prvky sa používajú najmä pri implementácii kombinačnej logiky, avšak každý takýto prvok môže slúžiť ako pamäť s veľkosťou 64 bitov. Pamäte zložené z vyhľadávacích tabuliek sa tiež nazývajú distribuované RAM a využívajú sa v prípadoch, kedy je potreba uchovávať menší objem dát. Výhodou využitia vyhľadávacích tabuliek, je že ide o asynchrónne pamäte. Preto sú dáta dostupné na výstupe v tom istom hodinovom takte, v ktorom bola odoslaná žiadosť o ne.

Keďže jedna vyhľadávacia tabuľka predstavuje najmenší prvok, z ktorého môže byť distribuovaná pamäť vyskladaná, v jednom okamihu je možné zapísať alebo čítať najviac jednu jej bunku, jeden bit. Ak teda chceme pristupovať k viacerým bitom naraz, napr. zapísať jeden bajt, musíme využiť osem vyhľadávacích tabuliek a zapísať každý bit do inej LuT.

Kapitola 6

Generovanie modulu FSM

Jedným z cieľov, tejto práce je dosiahnuť flexibilnosť implementovanej jednotky. Použitie technológie FPGA nám umožňuje, k splneniu tohto cieľa, využiť rekonfigurovateľnosť hradlových polí. Aby bolo možné, zmeny v zdrojových súboroch jazyka P4, dostatočne rýchlo premietnuť do konfigurácie jednotky, ako súčasť práce bol vyvinutý generátor popisu v jazyku VHDL.

Ako bolo spomenuté v kapitole 6, modul FSM riadi celú jednotku HFE, vykonáva analýzy hlavičiek a extrakcie dát. Preto zmeny, v popise parsovania alebo štruktúre hlavičiek protokolov ovplyvňujú iba tento modul. Z toho vyplýva, že implementácia vstupnej a výstupnej pamäti sa nemení a že je potrebné generovať iba modul FSM.

Implementovaný generátor na základe popisu parsovania a štruktúry hlavičiek protokolov v jazyku P4 vytvorí popis, odpovedajúci architektúre modulu FSM, v jazyku VHDL. Transformácia kódu z jedného jazyka do druhého sa nazýva preklad. Tento proces prebieha v štyroch fázach [18].

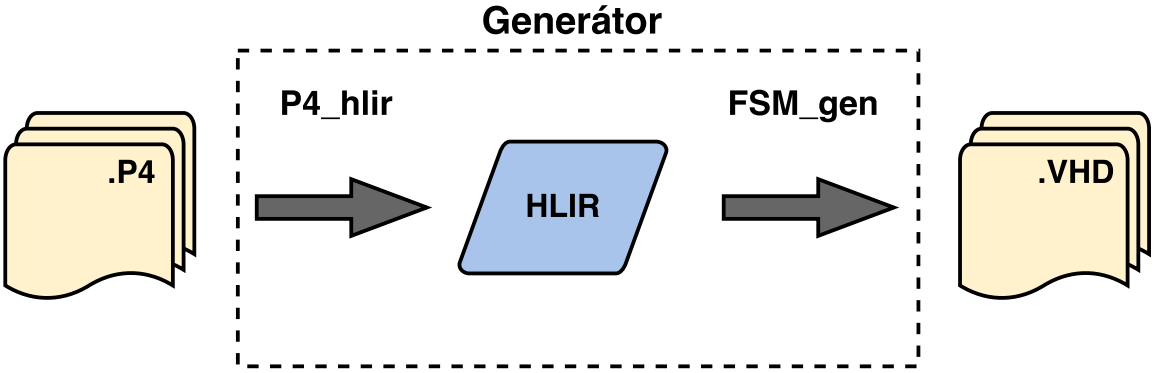
1. Lexikálna analýza
2. Syntaktická analýza
3. Sémantická analýza
4. Generovanie kódu v cieľovom jazyku

Proces prekladu je implementovaný generátorom, ktorý sa skladá z dvoch podprogramov: P4_hlir a FSM_gen. Na obrázku 6.1 sú reprezentované šípkami, ktorých orientácia rozlišuje vstupy a výstupy jednotlivých podprogramov. Súčasťou generátoru sú tiež objekty internej reprezentácie – HLIR, ktoré sú dočasne uložené v operačnej pamäti. Tieto objekty navzájom prepájajú oba podprogramy.

Organizácia P4 Language Consortium dohliada na Open source projekt P4 language, v rámci ktorého bol vyvinutý program P4_hlir [23]. Tento program vykonáva lexikálnu a syntaktickú analýzu súborov v jazyku P4. Analýzy určujú, či vstupné súbory odpovedajú špecifikácií jazyka P4 z lexikálneho a syntaktického hľadiska. Výstupom podprogramu P4_hlir je tzv. HLIR, z angl. High-Level Intermediate Representation [24]. Ide o internú reprezentáciu, ktorá je vo forme objektov dočasne uchovávaná v operačnej pamäti.

Druhou časťou generátora je podprogram FSM_gen, ktorý vykonáva sémantickú analýzu a generovanie kódu. Jeho vstupmi sú objekty HLIR, nad ktorými prebiehajú sémantické analýzy, ktoré zaisťujú, že všetky využité konštrukcie jazyka P4 je možné reprezentovať v

navrhnutej architektúre. Nakoniec prebehne generovanie kódu, z čoho vyplýva, že výstupom programu FSM_gen sú súbory v jazyku VHDL.



Obr. 6.1: Postup generovania VHDL z P4

Existuje niektoľko typov objektov HLIR, pričom každému odpovedá zoznam, v ktorom sú zoskupené všetky objekty daného typu. Typy objektov sú odvodené od dôležitých syntaktických konštrukcií jazyka P4, ktoré boli popísané v kapitole 3. Pri generovaní modulu FSM sú pre nás dôležité tieto štyri typy objektov:

- **p4_header** reprezentuje hlavičku protokolu definovanú v časti Headers. Objekty tohto typu obsahujú informácie o názve hlavičky, jej veľkosti a poliach, z ktorých pozostáva.
- **p4_header_instance** predstavuje inštanciu hlavičky vytvorenú v časti Parsers. Atribúty týchto objektov odkazujú na objekt p4_header, definujúci hlavičku, ktorá bola inštancovaná.
- **p4_field** zastupuje pole, ktoré je ako súčasť hlavičky definované v časti Headers. Objekt typu p4_field obsahuje informácie o názve poľa, jeho šírke, offsete v rámci hlavičky a inštancii hlavičky, ktorej je toto pole súčasťou.
- **p4_parse_state** reprezentuje funkciu parser, definovanú v časti Parsers. Objekty tohto typu odkazujú na inštanciu hlavičky analyzovanú v danej funkcii. Ďalej obsahujú zoznam závislých polí na základe hodnôt, ktorých sa vyhodnocuje volanie ďalšej funkcie. Taktiež aj zoznam dvojíc, objekt p4_parse_state a hodnota, ktoré určujú, aká hodnota musí byť v závislých poliach, aby riadenie bolo ďalej predané funkcii identifikovanej objektom p4_parse_state z danej dvojice.

6.1 FSM_gen

Podprogram FSM_gen spracúva objekty internej reprezentácie HLIR tak, aby bolo možné na základe získaných informácií generovať kód odpovedajúci návrhu architektúry. Postupnosť krokov, z ktorých sa prevod skladá, sa odráža v náväznosti tried, ktoré implementujú tieto kroky. Na obrázku 6.2 sú zobrazené jednotlivé kroky, ktoré predstavujú postup prekladu z jazyka P4 do VHDL. Každý krok charekterizuje forma, ktorú nadobúdajú jeho objekty.

Obr. 6.2:

Najprv je parsovanie popísané v jazyku P4 a skladá sa z funkcií parsera, ktoré spracovávajú jednotlivé protokoly. Preto sú v prvom kroku tieto funkcie analyzované a výsledkom sú im prislúchajúce objekty. Tieto objekty obsahujú informácie o postupe vykonávania analýz danou funkciou.

V ďalšom kroku prekladu sú funkcie parsera prevedené na graf protokolov, ktorý bol popísaný v kapitole 4. Uzly tohto orientovaného grafu reprezentujú parsovanie založené na inštanciách hlavičiek. V prípade, že je hlavička istého typu inštancovaná viackrát, tak funkcia spracúvajúca tieto inštancie je modelovaná v parse grafe ako viacero zreťazených uzlov. Každý z týchto uzlov vykonáva analýzy nad jednou inštanciou. Týmto spôsobom je možné obmedziť počet rovnakých hlavičiek nasledujúcich zasebou, napr. VLAN tagov. Tento krok je nevyhnutný, pretože prevádza funkcie parsera na funkčne ekvivalentný konečný automat, z ktorého vyhádza architektúra FSM.

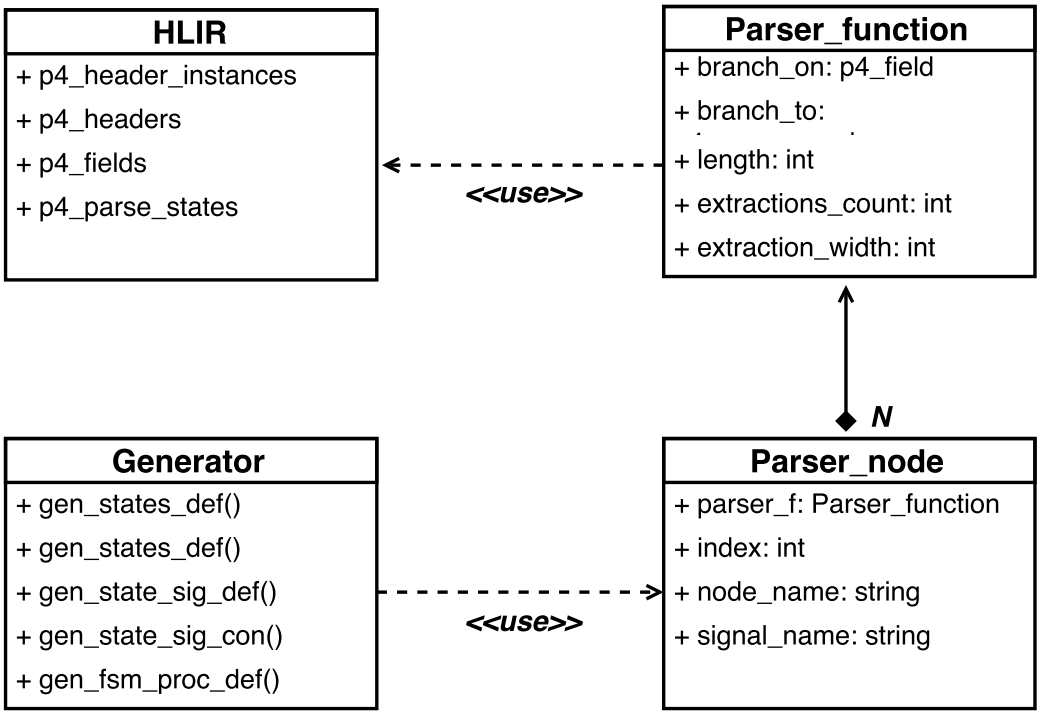
V treťom kroku je parse graf mapovaný do architektúry FSM. Abstrakcia uzlu parse grafu, analyzujúceho inštanciu hlavičky protokolu, je nahradená postupnosťou stavov, ktoré spolu tvoria presný postup parsovania tejto hlavičky. Zároveň v tomto kroku prebieha generovanie kódu, počas ktorého je popis činnosti v jednotlivých stavoch postupnosti vyjadrený v jazyku VHDL.

Každý z týchto troch krokov je implementovaný samostatnou triedou. Na obrázku 6.3 je zobrazený diagram tried, v ktorom sú označené vzťahy medzi triedami ako aj základné metódy a atribúty, ktoré tieto triedy obsahujú. Postupnosť krokov prekladu je reprezentovaná vzťahmi use na obrázku.

- **Parser_function** je triedou, ktorá reprezentuje funkcie parsera. Každá inštancia tejto triedy odpovedá jednému objektu typu p4_parse_state. Pomocou odkazov, medzi jednotlivými typmi objektov, sú zosumarizované informácie o každej funkcii parseru. Postupne sú získané a uložené informácie o veľkosti hlavičky, polohe potrebných dát a počte čítaní zo vstupnej pamäte.

- **Parser_node** predstavuje druhý krok prevodu. Inštancie tejto triedy zastupujú jednotlivé stavy grafu protokolov a obsahujú unikátne meno pre každý uzol parse grafu a obsahujú odkaz na inštanciu triedy Parser_function, z ktorej vychádzajú.

- **Generator** je triedou, ktorá je zodpovedá za mapovanie informácií získaných v prvých dvoch krokoch do navrhnutaj architektúry a následné generovanie VHDL kódu. Táto trieda už nie je inštancovaná, ale postupne prechádza inštanciami triedy Parser_node a do výstupného súboru vkladá odpovedajúce bloky kódu.



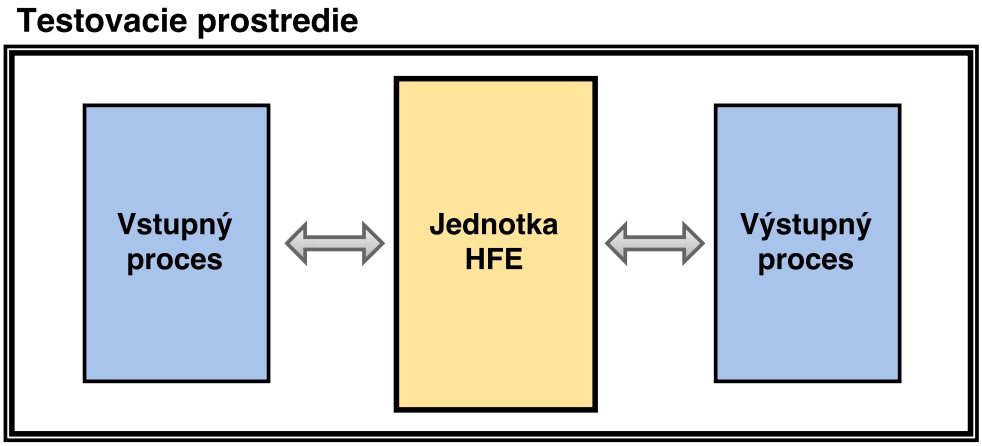
Obr. 6.3: Diagram tried podprogramu FSM_gen

Kapitola 7

Testovanie

Správnosť návrhu a implementácie generátoru z jazyka P4 do VHDL overíme testovaním jednotky HFE, ktorej súčasťou je vygenerovaný modul FSM. Testovanie prebieha formou simulácií v jazyku VHDL, ktoré overujú, že jednotka HFE dokáže spracovávať pakety a získavať z nich odpovedajúce výstupy. Simulácie sú vyhodnocované na základe porovnania extrahovaných informácií a očakávaných výstupov.

Simulačné prostredie samozrejme obsahuje jednotku HFE, ktorá je stimulovaná vstupným a výstupným procesom. Na obrázku 7.1 je znázornená schéma simulačného prostredia, ktoré obsahuje vstupný a výstupný proces, ktoré komunikujú s jednotkou HFE. Komunikácia, ktorá je na obrázku znázornená šípkami, prebieha prostredníctvom vstupného a výstupného rozhrania jednotky.



Obr. 7.1: Schéma simulačného prostredia

Za účelom testovania bola vytvorená sada simulačných prostredí, ktoré sa líšia od seba v implementácii vstupného procesu. Každá z týchto simulácií testuje istú situáciu alebo špecifickú kombináciu zapúzdrených protokolov. Sada simulačných prostredí sa zameriava na testovanie týchto prípadov:

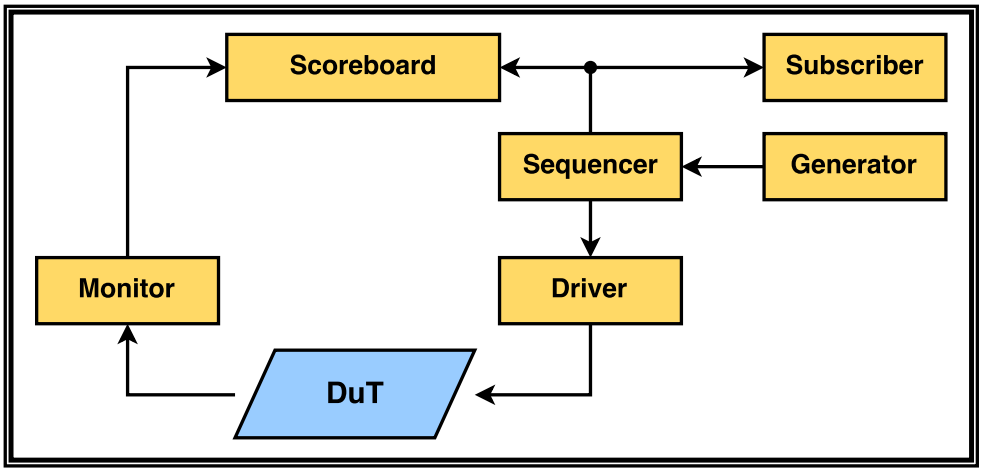
- **Základné protokoly** označujú množinu protokolov Ethernet, IPv4, IPv6, TCP, UDP, ICMP a IGMP. Vstupný proces vkladá do vstupnej pamäti pakety obsahujúce kombinácie hlavičiek týchto protokolov. Simulácie analýz základných protokolov overujú najjednoduchšiu funkcionalitu jednotky HFE.

- **Variabilná šírka** poľa v rámci protokolov IPv4, IPv6 a TCP robí proces analýz zložitejším, pretože najprv musíme zistiť jeho veľkosť. Z toho dôvodu je táto ojedinelá vlastnosť protolov testovaná samostatne.
- **VLAN, MPLS** sú špeciálnou skupinou protokolov, pretože ich hlavičky je možné reťaziť. Simulácie testujú potrebnú vlastnosť parsera obmedziť počet opakovaní hlavičiek týchto protokolov.
- **Chybné pakety** sú tiež predmetom testovania. Chyba v pakete je simulovaná vytvorením neúplného paketu alebo hlavičkou protokolu, ktorá nesprávne identifikuje nasledujúci protokol. Takýmto spôsobom je možné odtestovať, že jednotka HFE je schopná vysporiadať sa nielen s náhodnými chybami spôsobenými prenosom, ale aj so zámerným útokom s cieľom znefunkčniť implementovaný analyzátor.
- **Viacero paketov** je postupne odosielaných do jednotky HFE a výsledky spracovania sú priebežne preberané z výstupnej pamäte. Tento typ simulácii testuje vzájomnú komunikáciu komponent jednotky počas dlhodobejšej aktivity.

Simulácie boli vykonané v programe Vivado 2015.1 od firmy Xilinx. Priebeh simulácie je zobrazený v simulačnom diagrame, príklad ktorého sa nachádza v prílohe A.

7.1 Návrh verifikáčného prostredia

Dôkladnejším spôsobom ako overiť korektnosť implementácie, je funkčná verifikácia. Jej cieľom je určiť, či testovaný design odpovedá špecifikácii. Základ tvorí verifikačné prostredie, ktoré stimuluje testovanú komponentu, angl. *design under test*, skr. *DuT*. Narozdiel od simulácii, verifikačné prostredie náhodne generuje vstupné dáta a automaticky porovnáva hodnoty na výstupe s očakávanými výstupmi. Zároveň sú vedené záznamy o tzv. pokrytí, ktoré určuje aká časť všetkých kombinácií už bola odtestovaná. Tento prístup sa nazýva angl. *coverage-driven verification*, pretože jeho cieľom je dosiahnuť maximálne pokrytie [10].



Obr. 7.2: Schéma navrhnutého verifikačného prostredia

Na obrázku 7.2 je znázornená schéma, na ktorej je možné vidieť, že verifikačné postredie sa skladá z objektov a ich vzájomnej komunukácie. DuT predstavuje testovanú jednotku

HFE popísanú v jazyku VHDL. Ostatné objekty budú implementované v jazyku SystemVerilog [8].

Generator vytvára testovacie dáta, v našom prípade pakety. Pakety sú odoslané objektu *Sequencer*, ktorého úlohou je vytvoriť z nich sekvenciu. Sekvencia je postupnosťou tranzakcií a odosiela sa do objektov Subscriber, Scoreboard a Driver. *Subscriber* na základe informácií o štruktúre paketov z prijatých sekvencií zisťuje pokrytie kombinácií vygenerovaných paketov. Objekt *Scoreboard* obsahuje model DuT, na základe ktorého vie určiť očakávané výstupy spracovania paketov zo sekvencií. Objekt *Driver* je pripojený na testovanú jednotku a jeho úlohou je prevádzať pakety zo sekvencií na signály vstupného rozhrania. Opačnú funkcionalitu vykonáva *Monitor*, ktorý výstupy jednotky prevádza na tranzakcie odosielané do objektu Scoreboard, ktorý ich porovnáva s očakávanými výstupmi a tým overuje korektnosť implementácie jednotky HFE.

Kapitola 8

Dosiahnuté výsledky

Po overení funkcionality je potrebné zisiť, aké sú nároky návrhovanej architektúry na zdroje. Všetky potrebné informácie, ohľadom využitých zdrojov, sú získané po syntéze jednotky HFE, v našom prípade vykonávanej programom Xilinx ISE Design Suite 14.7.

Pojmom zdroje sú sumárne označené jednotlivé typy blokov, z ktorých sa skladá FPGA čip. Konkrétne sa jedná o vyhľadávacie tabuľky, označené ako *LUT-FF* a blokové pamäte, *BRAM*. V tabuľke 8.1 je množstvo spotrebovaných zdrojov navyše prepočítavané vzhľadom na celkový počet blokov zariadenia C7VX690T, z rodiny Virtex 7 od firmy Xilinx [16]. Ďalším údajom uvádzaným v tabuľke je počet stavov vygenerovaného konečného automatu. Veľmi významným parametrom je aj maximálna frekvencia hodinového signálu, ktorý zabezpečuje synchronizáciu syntetizovaného designu. Z tohto údaju je možné odvodiť maximálnu teoretickú priepustnosť jednotky. Túto hodnota je vypočítaná zo vzťahu 8.1.

$$P = f * w \tag{8.1}$$

Tento elementárny vzťah vyjadruje, že priepustnosť závisí na frekvencii a dátovej šírke, ktorá je označená w a rovná sa 8 bitom.

Aby bolo možné sledovať vývoj množstva využitých zdrojov a ďalších parametrov, vysyntetizovaných bolo niekoľko verzií jednotky HFE. Jednotlivé verzie sa odlišújú odseba konfiguráciou modulu FSM. Prvá konfigurácia, v tabuľke označená *Start* obsahuje iba pomocné stavy konečného automatu a nedokáže spracovať žiadnu hlavičku, takže každý paket prejde parserom bez zmeny. *Standard*, obsahuje protokoly Ethernet, IPv4, TCP, UDP, ICMP a IGMP. Graf protokolov tejto konfigurácie je súčasťou prílohy B. Ďalšia konfigurácia, *IPv6*, pridáva ku štandardnej, podporu protokolu IPv6 a jeho rozširujúcich hlavičiek, ktoré reprezentujú variabilnú šírku tohto protokolu. Posledná konfigugurácia pridáva ku všetkým spomínaným protokolom podporu *VLAN* a *MPLS*.

Podľa nárastu počtu stavov v konečnom automate môžeme odvodiť náročnosť parso-

Konfiguracia	Stavy	f	LUT-FF		BRAM		Priepustnost
	pocet	[MHz]	pary	%	pocet	%	[Gbps]
Start	5	330.682	315	0%	2	0%	2.645
Standard	35	232.870	445	0%	2	0%	1.863
IPv6	82	221.226	552	0%	2	0%	1.770
VLAN, MPLS	96	200.504	605	0%	2	0%	1.604

Tabuľka 8.1: Parametre jednotky HFE po syntéze rôznych konfigurácií

vania protokolov. Od počtu stavov závisí aj množstvo využitých zdrojov, najmä LUT-FF párov. Počet BRAM je vo všetkých konfiguráciách rovnaký, pretože sa využívajú v module vstupnej pamäti, ktorý množina spracovávaných protokolov neovplyvňuje. Konfigurácia najviac ovplyvňuje počet stavov konečného automatu a tým pádom zväčšuje kritickú cestu, ktorá prestavuje cestu, medzi vstupom a výstupom jednotky, s najväčším oneskorením. Z tohto oneskorenia sa odvodzuje frekvencia a v konečnom dôsledku aj teoretická priepustnosť.

Vynikajúcim výsledkom je, že žiadna konfigurácia nezaberá ani 1% zdrojov. Rovnako dobré výsledky dosiahla jednotka v oblasti výkonu, kde maximálna teoretická priepustnosť pri žiadnej z konfigurácií neklesla pod 1,6 Gbps.

Kapitola 9

Záver

Cieľom tejto práce bolo navrhnúť architektúru jednotky vykonávajúcej analýzu a extrakciu položiek z hlavičiek protokolov a implementovať program mapujúci popisy protokolov v jazyku P4 do navrhnutej architektúry. Oba tieto ciele boli splnené.

Návrhu a implementácii predchádzala príprava, počas ktorej bolo potrebné naštudovať jazyk P4. Na základe konceptu jazyka P4, ktorý reprezentuje parser ako graf protokolov, som sa rozhodol riadiť spracovávanie paketov konečným automatom. Ďalej som sa oboznámil s architektúru FPGA Virtex 7 a zameral som sa na jej nasadenie v rámci platformy Zynq.

Na základe získaných informácií, bolo možné prejsť k samotnému návrhu jadnotky HFE, ktorý bol ovplyvnený využitím technológie FPGA a črtami jazyka P4. Ako sa v kapitole 5 uvádza, v prvej fáze návrhu bola jednotka rozdelená na moduly. Ďalej bolo navrhnuté vstupné a výstupné rozhranie jednotky a následne aj rozhrania jednotlivých modulov. Nakoniec bol vytvorený návrh samotných modulov.

Jedným z nárokov kladených na analyzátor hlavičiek je flexibilnosť. Vďaka možnosti rekonfigurovať hradlové polia na FPGA čipe, môžeme flexibilnosť dosiahnuť bez toho, aby bol ovplyvnený samotný návrh jednotky. Preto bol implementovaný program generujúci z popisu v jazyku P4 nový popis vo VHDL.

Za účelom kontroly korektnosti implementácie jednotky HFE bola vytvorená sada simulačných prostredí. V simuláciách sú do jednotky HFE posielané testovacie dáta vo forme paketov, v ktorých sa nachádza 15 rôznych kombinácií hlavičiek protokolov. Taktiež je testovaná variabilná šírka polí v hlavičkách a určovanie nasledujúceho protokolu špekulatatívnym postupom *lookahead*. Súčasťou testovacej sady je aj dlhodobejší test počas ktorého je spracovaných 100 paketov.

Nakoniec boli vykonané merania dosiahnutých vlastností. Bolo vysyntetizovaných niekoľko variant jednotky s rôznymi modelmi analyzátoru paketov. Na základe výstupov syntéz bola vypočítaná teoretická priepustnosť jednotky a spotreba zdrojov každej z variánt.

Tieto výstupy ukázali, že aj pri variante s najväčším počtom protokolov je spotrba zdrojov veľmi nízka a žiadny meraný parameter nevyužil ani 1% dostupných zdrojov. Taktiež maximálna frekvencia pri týchto variantách neklesla pod hranicu 200 MHz, čo odpovedá priepustnosti 1,6 Gbps. Takáto priepustnosť, pri dátovej šírke 8 bitov a minimálna spotreba zdrojov, sú vynikajúcim výsledkom.

Veľký vplyv na vlastnosti jednotky má dátová šírka. Preto s cieľom dosiahnuť priepustnosť 10 Gbps budú v budúcnosti vytvorené ďalšie verzie generátora podporujúce dátovu šírku 32 a 64 bitov. Ďalším plánovaným rozšírením tejto práce je implementácia verifikačného prostredia, ktorého návrh bol prezentovaný v kapitole 7, s cieľom ešte dôkladnejšie testovať implementovaný generátor.

Literatúra

[1] *Ethernet*. Wikipedia: the free encyclopedia, [online; cit. 2016-03-04].
URL https://en.wikipedia.org/wiki/Ethernet

[2] *Internet protocol suite*. Wikipedia: the free encyclopedia, [online; cit. 2016-01-21].
URL https://en.wikipedia.org/wiki/Internet_protocol_suite

[3] *Maximum transmission unit*. Wikipedia: the free encyclopedia, [online; cit. 2016-04-10].
URL https://en.wikipedia.org/wiki/Maximum_transmission_unit

[4] *OSI model*. Wikipedia: the free encyclopedia, [online; cit. 2016-01-18].
URL https://en.wikipedia.org/wiki/OSI_model

[5] *RFC 768 User Datagram Protocol*. [online], Vytvorené v august 1980.
URL https://tools.ietf.org/html/rfc768

[6] *RFC 791 Internet Protocol*. [online], Vytvorené v septembri 1981.
URL http://tools.ietf.org/html/rfc791

[7] *RFC 793 Transmission Control Protocol*. [online], Vytvorené v septembri 1981.
URL https://tools.ietf.org/html/rfc793

[8] *SystemVerilog*. Wikipedia: the free encyclopedia, [online; cit. 2016-04-10].
URL https://en.wikipedia.org/wiki/SystemVerilog

[9] *Transmission Control Protocol*. Wikipedia: the free encyclopedia, [online; cit. 2016-03-01].
URL https://en.wikipedia.org/wiki/Transmission_Control_Protocol

[10] Accellera Systems Initiative: *Universal Verification Methodology (UVM) 1.2 Class Reference*. [online], Vytvorené v júni 2014 [cit. 2016-04-01].
URL http://accellera.org/images/downloads/standards/uvm/UVM_Class_Reference_Manual_1.2.pdf

[11] Attig, M.; Brebner, G.: *400 Gb/s Programmable Packet Parsing on a Single FPGA*. ANCS'11, *7th ACM/IEEE Symposium, [online], Vytvorené v októbri 2011.*
URL www.xilinx.com/programmable/about/research-labs/ANCS_final.pdf

[12] Bosshart, P.; Daly, D.; Gibb, G.; aj.: P4: Programming Protocol-Independent Packet Processors. Computer Communication Review, [online], Vytvorené v júly 2014.
URL www.sigcomm.org/sites/default/files/ccr/papers/2014/July/0000000-0000004.pdf

[13] *Cerf, V. G.; Kahn, R. E.: A Protocol for Packet Network Intercommunication. IEEE Transactions on Communications, ISSN 0090-6778, doi:10.1109/TCOM.1974.1092259, [online], Vytvorené v máji 1974.*
URL http://web.archive.org/web/20150723184900/http:/ece.ut.ac.ir/Classpages/F84/PrincipleofNetworkDesign/Papers/CK74.pdf

[14] *Deering, S.; Hinden, R.: RFC 2460 Internet Protocol, Version 6 (IPv6) Specification. [online], Vytvorené v decembri 1998.*
URL http://tools.ietf.org/html/rfc2460

[15] *Gibb, G.; Varghese, G.; Horowitz, M.; aj.: Design Principles for Packet Parsers. ANCS'13, ACM/IEEE Symposium, [online], Vytvorené v októbri 2013.*
URL http://yuba.stanford.edu/~nickm/papers/ancs48-gibb.pdf

[16] *Xilinx Inc.: Virtex-7 FPGAs. [online; cit. 2016-05-14].*
URL http://www.xilinx.com/support/documentation/selection-guides/virtex7-product-table.pdf

[17] *Kozanitis, C.; Huber, J.; Singh, S.; aj.: Leaping multiple headers in a single bound: wire-speed parsing using the Kangaroo system. INFOCOM 2010, [online], Vytvorené v marci 2010.*
URL http://cseweb.ucsd.edu/~ckozanit/files/infocom2010.pdf

[18] *Meduna, A.; Lukáš, R.: Formální jazyky a překladače: Studijní opora. [online; cit. 2016-02-17].*
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IFJ-IT/texts/OporaIFJ.pdf?cid=9970

[19] *Polčák, L.: Hardwarova akcelerace extrakce položek z hlaviček paketů. Diplomová práca, FIT VUT v Brně, 2010.*

[20] *Puš, V.; Kořenek, J.: Design Methodology of Configurable High Performance Packet Parser for FPGA. Design and Diagnostics of Electronic Circuits & Systems , [online], Vytvorené 2014.*
URL https://www.liberouter.org/wp-content/uploads/2013/02/hfe.pdf

[21] *Sunshine, C. A.: Computer Network Architectures and Protocols, kapitola A Brief History of Computer Networking. Springer US, první vydání, 1989, ISBN 978-1-4612-8093-4.*

[22] *The P4 Language Consortium: P4-GRAPHS. GitHub Inc., [online; cit. 2015-10-13].*
URL https://github.com/p4lang/p4c-graphs

[23] *The P4 Language Consortium: P4-HLIR. GitHub Inc., [online; cit. 2015-10-13].*
URL https://github.com/p4lang/p4-hlir

[24] *The P4 Language Consortium: P4 HLIR 0.9.22 Specification. [online], Vytvorené v apríli 2015 [cit. 2015-08-28].*
URL https://github.com/p4lang/p4-hlir/blob/master/HLIRSpec.pdf

[25] *WWW stránky: Communication Protocol. [online; navštívené 18.1.2016].*
URL www.techopedia.com/definition/25705/communication-protocol

[26] *WWW stránky: P4. [online; navštívené 24.9.2015].*
URL http://p4.org

[27] *The P4 Language Consortium: The P4 Language Specification, version 1.0.2. [online], Vytvorené v marci 2015 [cit. 2016-01-18].*
URL http://p4.org/wp-content/uploads/2015/04/p4-latest.pdf

Prílohy

Zoznam príloh

A Simulačný diagram **36**

B Graf Protokolov **38**

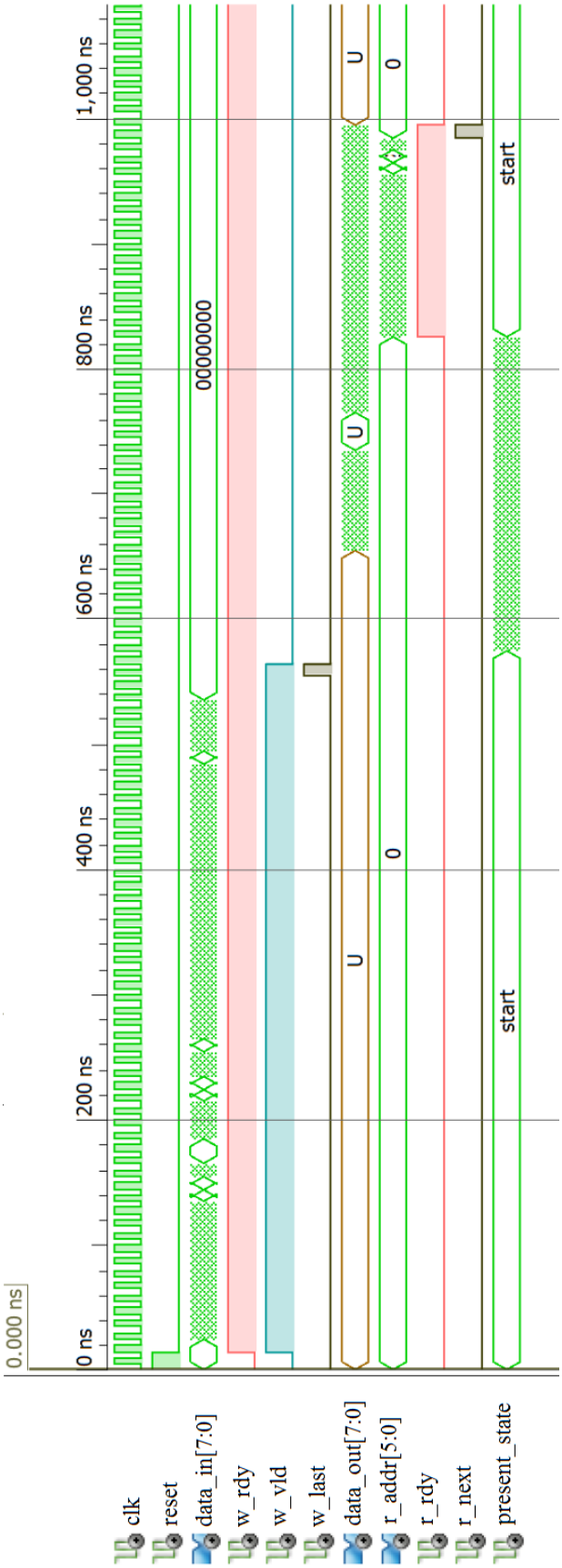
C Obsah CD **40**

Príloha A

Simulačný diagram

Simulačný diagram z tejto prílohy zobrazuje uloženie paketu do vstupnej pamäte, jeho spracovanie modulom FSM, uloženie výsledkov spracovania do výstupnej pamäte a prevzatie týchto dát.

Ukladanie paketu do vstupnej pamäte prebieha pokiaľ je nastavený signál *w_vld* v logickej 1. Na obrázku je signál w_vld označený modrou farbou. Uloženie posledného bajtu paketu signalizuje nastavenie signálu *w_last*, na obrázku znázorneného čiernou farbou. Spracovávanie modulom FSM oprebieha v čase zmien signálu *parse_state*. Počas tohto spracovania sú do výstupnej pamäte ukladané výsledky analýz. Tieto výsledky sú prevzané z výstupnej pamäte v čase zmien signálu *r_addr* a iba v tedy sú dáta vedené signálom *data_out* platné.

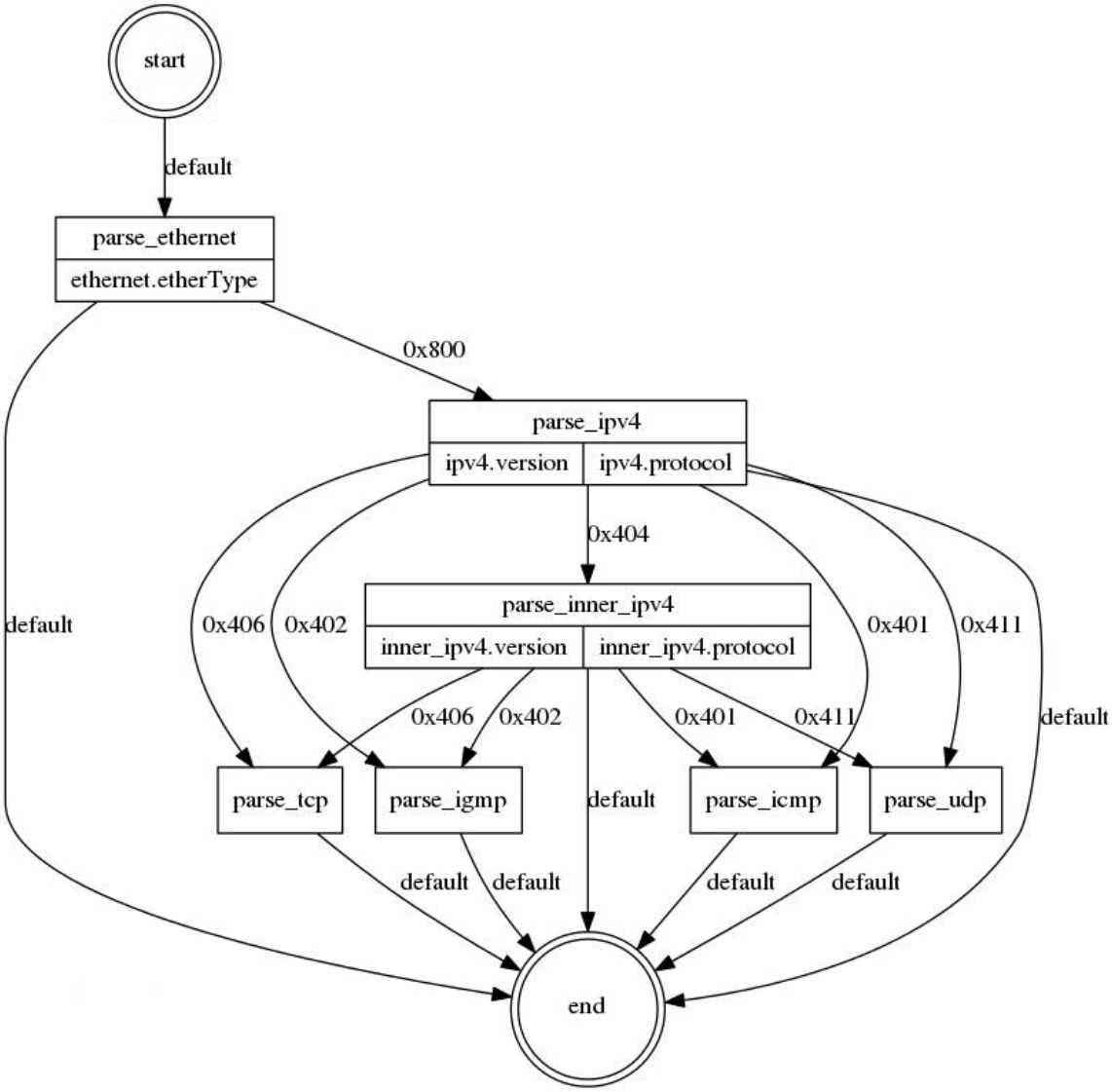

0.000 ns
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
clk
reset
data_in[7:0]
w_rdy
w_vld
w_last
data_out[7:0]
r_addr[5:0]
r_rdy
r_next
present_state
00000000
U
U
U
0
0
start
start

Príloha B

Graf Protokolov

Táto príloha obsahuje graf protokolov vygenerovaný pomocou programu P4-graphs [22] na základe konfigurácie *Standard*, ktorá podporuje protokoly Ethernet, IPv4, TCP, UDP, ICMP a IGMP.

Start je počiatočným a *end* koncovým uzlom grafu protokolov. V týchto dvoch uzloch neprebieha spracovávanie žiadneho protokolu. Ostatné uzly reprezentujú parsovanie hlavičiek protokolov. Tieto uzly sa na obrázku skladajú z dvoch častí. Vrchná časť obsahuje názov uzlu. Dolná časť je prítomná v prípade viacerých možností prechodu do nasledujúceho uzlu. Táto časť obsahuje názvy polí na základe ktorých je vykonané vetvenie. Prechody sú značené orientovanými hranami. Popisy jednotlivých hrán určujú hodnotu v hexadecimálnom tvare, pri ktorej je daný prechod vykonaný. Popis *default* označuje ostané hodnoty.


start
default
parse_ethernet
ethernet.etherType
0x800
parse_ipv4
ipv4.version
ipv4.protocol
0x404
parse_inner_ipv4
inner_ipv4.version
inner_ipv4.protocol
default
0x406
0x402
0x401
0x411
0x406
0x402
0x401
0x411
default
parse_tcp
parse_igmp
default
parse_icmp
parse_udp
default
default
default
default
end

Príloha C

Obsah CD

Priložené CD obsahuje:

- priečinok `p4_hlir` obsahujúci zdrojové súbori programu p4-hlir
- priečinok `src` obsahujúci zdrojové súbory generátora a jednotky HFE
- priečinok `p4` obsahujúci konfiguračné súbory jazyka P4
- priečinok `sim` obsahujúci sadu simulačných prostredí
- priečinok `doc` obsahujúci zdrojové súbory bakalárskej práce pre LaTeXa obrázky
- script `generate.sh` spúšťajúci generovanie modulu FSM
- súbor `README` s návodom na spustenie generátoru