ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ PRÁCA

Študijný program:

Aplikované sieťové inžinierstvo

Bc. Jakub Hrabovský

Technológia netFPGA a jej využitie v zariadeniach reálnych sietí

Vedúci práce: doc. Ing. Pavel Segeč, PhD.

Reg. číslo: 483/2014 Máj 2015

ŽILINSKÁ UNIVERZITA V ŽILINE, FAKULTA RIADENIA A INFORMATIKY.

ZADANIE TÉMY DIPLOMOVEJ PRÁCE.

Študijný program : Aplikované sieťové inžinierstvo

Zameranie: Sieťová infraštruktúra

Meno a priezvisko	**Osobné číslo**
Jakub Hrabovský	554158

Názov práce v slovenskom aj anglickom jazyku

Technológia netFPGA a jej využitie v zariadeniach reálnych sietí

The netFPGA technology and its real network utilization.

Zadanie úlohy, ciele, pokyny pre vypracovanie (Ak je málo miesta, použite opačnú stranu)

Cieľ diplomovej práce:
Cieľom práce je venovať sa z pohľadu výskumu katedry pilotne vybratej téme netFPGA. Práca bude pilotne skúmať technológiu netFPGA so zameraním na jej využitie v počítačových sieťach. Práca sa bude venovať architektúre, budovaniu experimentálnych HW platforiem, ako aj možnostiam programového rozšírenia funckionality netFPGA riešení.

Obsah:
Pri riešení práce sa zamerajte:

- Oboznámenie sa s technológiou FPGA so zameraním hlavne na netFPGA aplikáciu
- Analýza technológie, jej funkcionality, problematika nasadenia HW a SW komponentov s úlohou jej progamovania. Možnosti využitia NetFPGA a hw implementácie s jej využitím v sieťach vo forme sieťových a SDN zariadení.
- Stanovenie cieľov z oblasti jej možného využitia a nasadenia v reálnej počítačovej sieti. Analýza vzorových open source implementácií
- Výber z existujucich open-source projektov, ich implementácia, konfigurácia a testovanie funkčnosti
- Návrh a implementácia vlastného projektu so zameraním na SDN a testovanie jeho funkčnosti v reálnej sieti
- Zhodnotenie a zdokumentovanie.
- Vytýčenie ďalších smerov skúmania netFPGA

Témy z predmetov študijného zamerania 5SI041: 1, 6, 7
Meno a pracovisko vedúceho DP: doc. Ing. Pavel Segeč, PhD., KIS, ŽU
Meno a pracovisko tútora DP:

vedúci DP (dátum a podpis)	tútor (dátum a podpis)	vedúci katedry (dátum a podpis)	garant (dátum a podpis)
5.10.2014			

Zadanie zaregistrované dňa 7. 10. 2014 pod číslom 483/2014 podpis

ABSTRAKT

HRABOVSKÝ, Jakub: *Technológia netFPGA a jej využitie v zariadeniach reálnych sietí* [diplomová práca] – Žilinská univerzita v Žiline. Fakulta riadenia a informatiky; Katedra informačných sietí. – Vedúci: doc. Ing. Pavel Segeč, PhD. – Stupeň odbornej kvalifikácie: Inžinier v odbore Aplikované sieťové inžinierstvo. Žilina: FRI ŽU v Žiline, 2015. – 67 s.

Cieľom diplomovej práce je oboznámenie sa s technológiou NetFPGA a jej hlbšia analýza, ktorá je zameraná na jej funkcionalitu, využitie a problematiku nasadenia v oblasti počítačových sietí. V práci sú stručne zhrnuté základné znalosti z oblasti FPGA obvodov a popísané HW komponenty NetFPGA karty, v ktorej sú zvolené projekty implementované. Časť práce je venovaná popisu použitej architektúry a modulu, ktorý je jej základným stavebným prvkom. V samostatnej kapitole je podrobnejšie popísané rozhranie, cez ktoré modul komunikuje s okolím. Súčasťou práce je analýza, realizácia a testovanie východiskových projektov, ktoré predstavujú základné sieťové zariadenia. Práca uvádza popis použitých softvérových nástrojov a ich funkcií, ktoré sú súčasťou vývojového a testovacieho prostredia. V ďalšej časti uvádzame postup tvorby nového projektu pre NetFPGA kartu, ktorý je podrobne popísaný v jednotlivých krokoch. Na základe výsledkov realizácie a testovania projektov sú zhrnuté výhody a nevýhody použitia tejto technológie v oblasti počítačových sietí.

Kľúčové slová: NetFPGA, FPGA, HDL programovanie, modulárna architektúra, počítačová sieť.

ABSTRACT

HRABOVSKÝ, Jakub: *The netFPGA technology and its real network utilization* [Master thesis] – The University of Žilina. Faculty of Management Science and Informatics; Department of information networks. – Tutor: doc. Ing. Pavel Segeč, PhD. – Qualification level: Master in field Applied network engineering. Žilina: FRI ŽU in Žilina, 2015. – 67 pages.

An aim of master thesis is to familiarize ourselves with NetFPGA technology and its deeper analysis, which is targeted at its functionality, use and issues related to its implementation in the field of computer networks. In the thesis, background knowledge of the field of FPGA circuits is summarized in brief and HW components of NetFPGA card are described, in which selected projects are implemented. Part of thesis is dedicated to description of used architecture and module, which creates its core building element. In separate section, module interface is described in more detail, via which module communicates with its environment. Analysis, realization and testing of reference projects are presented in the next part of thesis. These projects represent basic network devices. Thesis presents description of used software tools and their functions, which are part of development and testing environment. In the next section, we present process of creation of new project designed for NetFPGA card. This process is described in more detail step by step. Based on results of project realization and testing, pros and cons of use of this technology in the field of computer networks are summarized.

Key words: NetFPGA, FPGA, HDL programming, modular architecture, computer network.

Prehlásenie

Prehlasujem, že som túto diplomovú prácu napísal samostatne pod odborným vedením vedúceho diplomovej práce, a že som uviedol všetky použité pramene a literatúru, z ktorých som čerpal.

V Žiline, 17.4.2015

.....

Jakub Hrabovský

Poďakovanie

Touto cestou by som sa chcel poďakovať vedúcemu diplomovej práce doc. Ing. Pavlovi Segečovi, PhD. za cenné rady a poskytnutú pomoc pri vypracovaní diplomovej práce.

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM POUŽITÝCH SKRATIEK

API - Application Programming Interface

ARP - Address Resolution Protocol

ASIC - Application Specific Integrated Circuit

AXI - Advanced eXtensible Interface

BPI - Byte Peripheral Interface

CAM - Content Addressable Memory

CLB - Configurable Logic Block

CLI - Command-Line Interface

CLK - Clock

CPU - Central Processing Unit

DCM - Digital Clock Manager

DMA - Direct Memory Acccess

DSP - Digital Signal Processor

EDK - Embedded Development Kit

FIFO - First-In First-Out

FMC - FPGA Mezzanine Card

FPGA - Field Programmable Gate Array

GPU - Graphics Processing Unit

GUI - Graphical User Interface

HDL - Hardware Description Language

ICMP - Internet Control Message Protocol

IDE - Integrated Development Environment

IOB - Input/Output Block

IP Core - Intellectual Property Core

ISE - Integrated Software Environment

JTAG - Joint Test Action Group

LAN	- Local Area Network
LC	- Logic Cell
LED	- Light-Emitting Diode
MAC	- Media Acccess Control
MDIO	- Management Data Input/Output
MIG	- Memory Interface Generator
NIC	- Network Interface Card
OTP	- One-Time Programmable
PAR	- Place and Route
PCIe	- Peripheral Component Interconnect Express
PME	- Power Management Event
PMOD	- Peripheral Module
RAM	- Random Access Memory
RGMII	- Reduced Gigabit Media Independent Interface
RKD	- Router Kit Daemon
RTL	- Register Transfer Level
SCONE	- Software Component Of NetFPGA
SDK	- Software Development Kit
TCAM	- Ternary Content Addressable Memory
TCP	- Transmission Control Protocol
UART	- Universal Asynchronous Receiver and Transmitter
UDP	- User Datagram Protocol
VLAN	- Virtual Local Area Network
XAUI	- 10 Gigabit Attachment Unit Interface
XPS	- Xilinx Platform Studio
XST	- Xilinx Synthesis Technology
XUP	- Xilinx University Program

ÚVOD

Cieľom diplomovej práce je oboznámenie sa s technológiou NetFPGA, ktorá je pilotne vybranou témou vhodnou pre výskum na katedre informačných sietí. Úlohou je získanie základných znalostí zameraných na túto technológiu, oboznámenie sa s potrebným vývojovým prostredím ako aj preskúmanie praktického použitia NetFPGA zariadení v reálnom sieťovom prostredí. Práca sa hlbšie zaoberá skúmaním technológie NetFPGA a jej hlavného prvku - FPGA obvodu - so zameraním na ich využitie v počítačových sieťach. Práca popisuje architektúru, budovanie experimentálnych HW platforiem a možnosti programového rozšírenia funkcionality NetFPGA v **HW** ako aj v **SW** oblasti. Hlavným zdrojom informácií je NetFPGA projekt a open-source komunita, ktorá sa aktívne podieľa na vývoji nových systémov pre toto zariadenie a prostredníctvom fóra pomáha novým programátorom v zoznamovaní sa s touto technológiou. Práca súčasne slúži ako východiskový dokument popisujúci základy pre študentov, ktorí sa rozhodnú venovať oblasti výskumu využitia NetFPGA, a motivuje ich, aby predstavili a realizovali svoje návrhy a nápady.

Práca pozostáva zo šiestich kapitol. Prvá kapitola popisuje dôvody pre vznik a použitie NetFPGA karty v oblasti počítačových sietí, uvádza všeobecný popis FPGA obvodu, ktorý je jej hlavným prvkom a príklady jeho využitia v reálnom prostredí. V druhej kapitole sa venujeme popisu HW špecifikácie nami používanej NetFPGA karty ako aj hostiteľského PC, do ktorého je karta vložená, zameriavame sa na možné využitie karty v súčasných počítačových sieťach a na spôsoby jej konfigurácie. Tretia kapitola popisuje vývojové a testovacie prostredie, ktoré je realizované v hostiteľskom PC a slúži pre implementáciu a testovanie jednotlivých projektov. V tejto kapitole uvádzame stručný popis použitých nástrojov ako aj NetFPGA repozitára, v ktorom sa nachádzajú všetky potrebné súbory. V štvrtej kapitole je podrobnejšie popísaná architektúra, na ktorej sú vybudované východiskové projekty. Kapitola popisuje modul, ktorý je základným stavebným prvkom každého systému, pričom sa zameriava na jeho rozhranie a vzájomnú komunikáciu s okolím. V piatej kapitole je uvedený postup tvorby vzorového projektu, ktorý je jasne a zrozumiteľne popísaný na jednotlivých priložených obrázkoch. V uvedenom postupe sa venujeme všetkým potrebným krokom od prípravy vhodného prostredia až po konfiguráciu NetFPGA karty. Analýza, realizácia a testovanie nami zvolených východiskových projektov sa nachádzajú v kapitole šesť. V zhodnotení použitej technológie uvádzame jej výhody a nevýhody, ktoré sme počas tvorby tejto práce zaznamenali. Prínosy zavedenia tejto technológie do oblasti počítačových sietí sú uvedené v závere práce.

1 ÚVOD DO NETFPGA

Jedným z problémov, ktoré dlhodobo pretrvávajú v oblasti sieťového inžinierstva, je potreba nástrojov pre vyučovanie sieťových počítačových systémov na vysokých školách ako aj rýchly návrh, implementácia a testovanie ich prototypov v oblasti výskumu. Problémom súčasného spôsobu vyučovania počítačových sietí je nedostatok praktických skúseností študentov na najnižších vrstvách OSI/ISO modelu (fyzická a linková vrstva). Dôvodmi pre tieto nedostatky sú zložitosť budovania vlastných hardvérových systémov (ich vytvorenie vyžaduje realizáciu viacerých zariadení, čo je cenovo aj časovo náročné) ako aj prioritné zameranie sa na mikroprocesory v predmetoch návrhu číslicových systémov.

Jedným z vhodných riešení odstraňujúcim spomínané problémy je netFPGA platforma, ktorá prináša nové možnosti v oblasti výskumu a štúdia. Táto platforma prešla postupným vývojom od počiatočných pokusov na Standfordskej univerzite až do súčasného stavu, ktorý rieši vyššie uvedené nedostatky a prináša flexibilné riešenie s minimálnymi nákladmi v porovnaní s alternatívami na trhu. NetFPGA umožňuje nie len vedcom ale aj študentom experimentovať s novými spôsobmi hardvérového spracovania paketov a svoje návrhy priamo implementovať vo forme sieťových zariadení. Realizované zariadenie je možné vložiť do reálnej počítačovej siete, testovať jeho funkcionalitu a spoluprácu s inými sieťovými prvkami, nájsť prípadné chyby, opraviť ich v návrhu a opäť vložiť do systému bez ďalších zložitejších krokov.

NetFPGA platforma je založená na modulárnom prístupe – tvorba systému pozostáva z prepojenia už existujúcich modulov s novými. Tento prístup prináša pružnosť do celého procesu návrhu systémov a zjednodušuje ho tým, že umožňuje vývojárom pokračovať v práci svojich predchodcov, nadviazať na nich a pridať nové funkcie. Princíp je rovnaký ako v prípade softvérových open-source projektov.

Na trhu existujú rôzne dosky obsahujúce ako svoje jadro FPGA obvod, ktoré sa líšia len použitými pripojenými perifériami. Práve tieto periférie určujú hlavné využitie danej dosky. Jednou z nich je aj NetFPGA doska. Tá pripomína bežnú grafickú kartu do PC a pozostáva z viacerých častí. Hlavnú časť tvorí tzv. **Field Programmable Gate Array** (**FPGA**) obvod, ktorý vykonáva práve vložený návrh – vnútornú logiku. Tento obvod sa nachádza priamo na karte vo forme čipu. Ku nemu sú pripojené ďalšie periférie, ktoré rozširujú možnosti jeho použitia tým, že prinášajú dodatočné (off-chip – umiestnené na doske

mimo FPGA obvod) RAM pamäte a slúžia ako vstupno-výstupné rozhrania. Cez tieto rozhrania komunikuje karta s inými zariadeniami.

Pre naše potreby sú podstatné sieťové ethernetové konektory, cez ktoré je možné pripojiť kartu do počítačovej siete, a tým umožniť vzájomnú komunikáciu s inými stanicami v LAN sieti. Druhým dôležitým rozhraním je PCI-Express konektor, cez ktorý komunikuje FPGA obvod priamo s hostiteľským počítačom, do ktorého je karta vložená.

NetFPGA kartu je možné použiť pre návrh a realizáciu ľubovoľného systému. Je ale zrejmé vzhľadom na jej štruktúru (použité periférie), že najväčší potenciál jej použitia sa skrýva v realizácii sieťových systémov, čo predstavuje jej pôvodný zámer.

Konkrétna **špecifikácia HW prvkov** NetFPGA karty a jej podrobnejšia analýza sú uvedené v kapitole 2.

1.1 ANALÝZA FPGA – FIELD PROGRAMMABLE GATE ARRAY

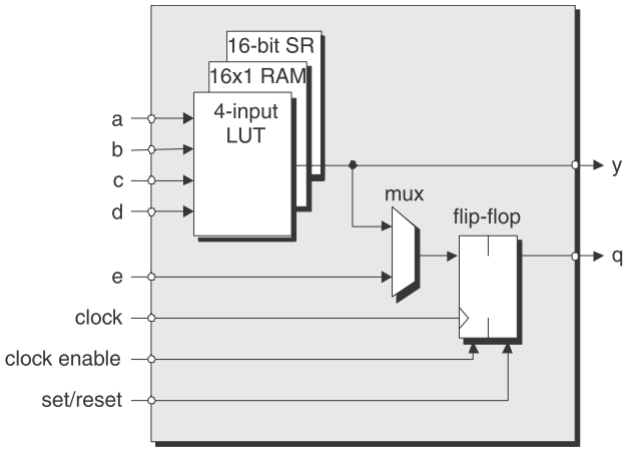
FPGA je číslicový integrovaný obvod známy ako programovateľné hradlové pole, ktorý obsahuje programovateľné logické bloky. Tie sú prepojené konfigurovateľnými prepojmi, ktoré tvoria maticu, čím je možné dosiahnuť vzájomné prepojenie všetkých blokov. V závislosti od použitej technológie pri implementácii FPGA obvodu môže byť tento obvod konfigurovateľný len raz, tzv. *one-time programmable* (OTP), alebo ľubovoľne veľakrát. V oboch prípadoch je však samotné naprogramovanie obvodu úlohou dizajnéra – používateľa. Ten zvolí návrh podľa vlastných požiadaviek a použije ho pre samotnú konfiguráciu bez ohľadu na výrobné nastavenia obvodu.

Vďaka FPGA obvodu je možné realizovať integrovaný obvod s ľubovoľnou funkcionalitou. Veľkosť a zložitosť návrhu je obmedzená počtom konfigurovateľných logických blokov v FPGA obvode. Súčasné FPGA obvody však umožňujú vytvorenie dostatočne komplexných návrhov s mnohými funkciami, ktoré sú obmedzené len schopnosťami a predstavivosťou samotného používateľa.

V nasledujúcich podkapitolách sú popísané prvky FPGA obvodu, ktoré umožňujú, aby FPGA obvod bol schopný fungovať ako aktuálne nastavený systém. Zdrojom informácií je [**1**].

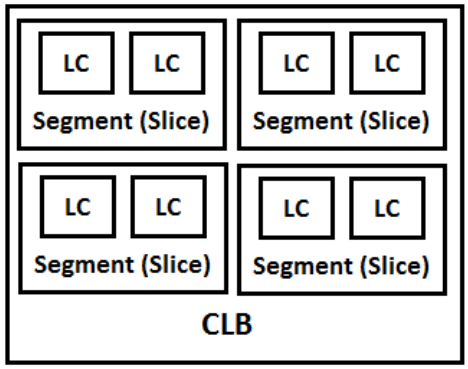
1.2 ARCHITEKTÚRA FPGA

Základným stavebným prvkom každého FPGA obvodu je logická bunka – *logic cell* (LC). Bunka pozostáva z niekoľkých navzájom prepojených elementárnych funkčných blokov (MUX, LUT, registre a iné), pomocou ktorých sme schopní realizovať ľubovoľné logické funkcie. Počet a štruktúra LC závisí od výrobcu a konkrétneho modelu obvodu.



OBRÁZOK Č. 1: Príklad LC bunky

Spojením viacerých LC dokopy dostaneme segment – *slice*. Všetky bunky v jednom segmente spoločne používajú CLK (hodinový signál), CE(Clock Enable) a S/R (Set/Reset) signály. Spojením viacerých segmentov dostaneme konfigurovateľný logický blok – *configurable logic block* (CLB). FPGA obvod pozostáva z veľkého množstva týchto blokov, v ktorých je implementovaná logika systému.



OBRÁZOK Č. 2: Príklad štruktúry CLB

1.2.1 Stavebné bloky FPGA

Okrem CLB sa na FPGA obvode nachádzajú ďalšie bloky, z ktorých každý prispieva špecifickou funkciou. Medzi ne patria:

- **Vstavané RAM pamäte** (**e-RAM**, **BRAM**) – tieto bloky sú umiestnené priamo na obvode a organizované v stĺpcoch. Každý z RAM blokov je nezávislý a teda môže byť použitý samostatne. Alternatívou využitia je spojenie viacerých blokov do jedného väčšieho bloku. Tieto bloky sú využívané ako RAM pamäte, FIFO (First-In First-Out), stavový automat a iné.

- **Hard IP jadrá** – pevne umiestnené moduly, ktoré rozširujú funkcionalitu obvodu a zjednodušujú návrh zložitejších systémov. Vďaka tomu programátor nemusí nanovo navrhnúť moduly, ktoré vykonávajú dané funkcie, ale priamo využije dostupnosť týchto blokov. Tieto bloky sú optimalizované na výkon aj veľkosť, aby v obvode zaberali čo najmenšiu plochu. Nevýhodou je fakt, že zaberajú plochu a spotrebujú energiu aj keď nie sú v konkrétnom návrhu používané. Vzorovým príkladom sú mikroprocesory. Zvyšovanie zložitosti systému vyžaduje od určitého bodu aj využitie mikroprocesoru, keďže realizácia takéhoto systému len v hardvérovej forme by bola príliš časovo aj návrhovo náročná. Z tohto dôvodu sú zložitejšie funkcie systému implementované v softvérovej forme a vykonávané mikroprocesorom.

- **Všeobecné vstupno-výstupné piny** – slúžia na komunikáciu obvodu s okolím. Sú umiestnené na okraji obvodu a zoskupené do blokov – *bánk*. Každý z týchto blokov je plne konfigurovateľný aby spĺňal používateľom zvolený štandard – elektrické aspekty signálov, ako napr. úroveň napätia pre log. 1 a log. 0. Vďaka tomu môže obvod komunikovať s rôznymi zariadeniami cez vhodne nastavené piny bez nutnosti riešiť každý zo štandardov zvlášť.

- **Manažér hodín** (**DCM**) – špeciálny číslicový vstavaný blok, ktorý je úplne konfigurovateľný. Na vstupe príjme externý hodinový signál (z externého zdroja mimo FPGA obvod), ten spracuje a na výstup pre každú CLK doménu (viď. nižšie) generuje upravený hodinový signál podľa aktuálnej konfigurácie. Tento blok poskytuje viaceré funkcie, cez ktoré je možné upravovať výstupný hodinový signál niektorej z CLK domén, napr. automatická korekcia oneskorenia (jitteru) a posunu (skew), zmena frekvencie a fázový posun. DCM je realizovaný ako *phase-locked loop* (**PLL**) alebo *digital delay-locked loops* (**DLL**).

- **Gigabit transceiver –** špeciálny pevný blok, ktorý slúži na rýchlu komunikáciu medzi zariadeniami (FPGA ←→ iné). Aby zabezpečil požadovanú rýchlosť

oboma smermi (čítanie/zápis), používa diferenčné páry signálov pre odosielanie (**TX**) a príjem (**RX**) samostatne. FPGA obvod obsahuje viacero takýchto blokov, ktoré sú využívané najmä pri rýchlych periférіach, napr. PCI-Express, 1G/10G ethernet a iné.

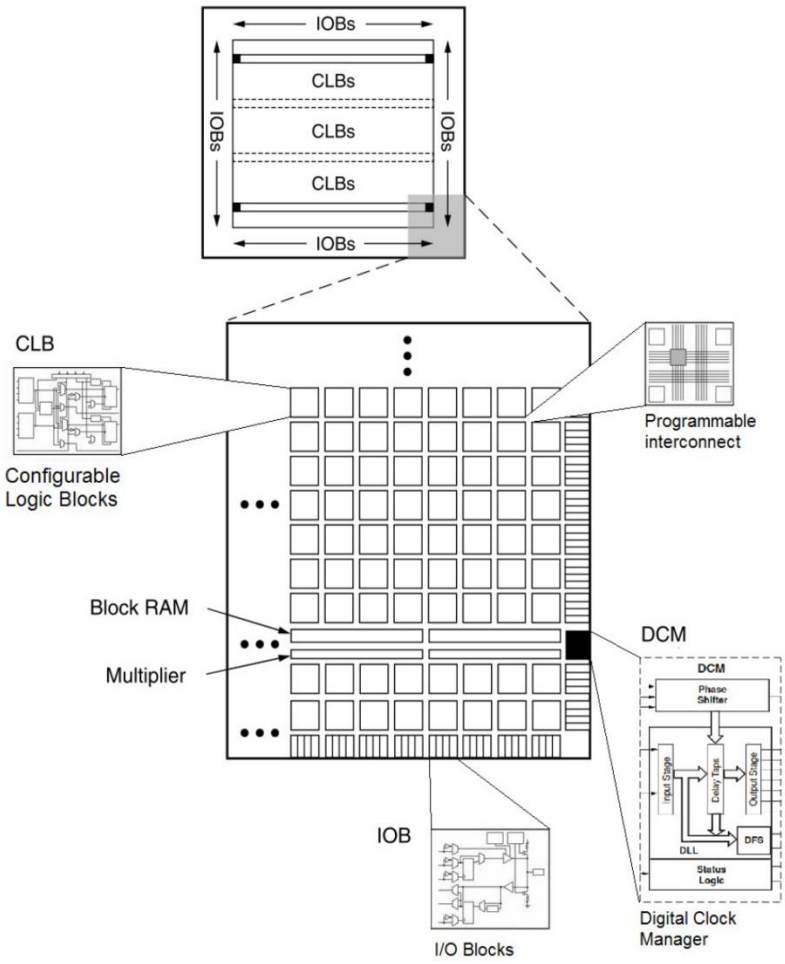
- **Vstavané násobičky**, **sčítačky** a iné – hlavnou úlohou dodatočných výrobcom vložených blokov je najmä zvýšenie výkonu obvodu a zjednodušenie práce programátorov tým, že do obvodu už vo výrobe vložia tie funkcie, ktoré sú používané v takmer každom systéme (napr. násobičky a sčítačky). Takto je značne znížený čas návrhu systému.

1.2.2 Konfigurovateľná matica prepojov

Väčšiu časť obvodu tvorí matica prepojov. Tá zabezpečuje, že jednotlivé CLB a iné špecifické bloky na obvode sú vzájomne prepojené a teda môžu navzájom komunikovať formou vstupných a výstupných signálov. Okrem všeobecných prepojov sú v každej LC použité špeciálne prenosové vedenia s vyššou rýchlosťou na miestach, kde je rýchlosť kľúčová - logické a aritmetické funkcie (počítadla, sčítačky).

1.2.3 Synchronizácia - hodinový signál a hodinový strom

Mnohé logické bloky použité v FPGA architektúre vyžadujú synchronizáciu prostredníctvom hodinového signálu (CLK) a preto je potrebné ku každému z týchto blokov CLK signál priviesť. Pre tento účel slúži tzv. **hodinový strom**. Jedná sa o strom spojov, ktorý sa postupne vetví. Na jeho konci sú listy – v tomto prípade vstupné CLK signály do jednotlivých blokov (napr. registre). Stromová štruktúra zabezpečí, že všetky elementy (listy stromu) príjmu signál v približne rovnakom časovom okamihu. V inom prípade by sa mohlo stať, že rôzne vzdialené elementy príjmu rôzne oneskorený signál, čo môže viesť k problémom a požadovaná synchronizácia nebude zabezpečená. Hodinový signál je do FPGA obvodu privedený z externého zariadenia cez špeciálne vstupné **hodinové piny**. V skutočnosti existujú v FPGA obvode viaceré hodinové stromy – **CLK domény**. DCM uvedený vyššie je schopný rôzne upravovať hodinový signál každej domény zvlášť podľa požiadaviek systému.



OBRÁZOK Č. 3: Architektúra FPGA

1.2.4 Vnútorná štruktúra obvodu - SRAM

Väčšina súčasných FPGA obvodov je založená na **SRAM** technológii, vrátane FPGA obvodu, ktorý sa nachádza na nami použitej NetFPGA karte (**NetFPGA-1G-CML**). Každý z vyššie uvedených blokov je realizovaný cez SRAM konfigurovateľné bunky.

Jednou z hlavných výhod SRAM je možnosť neustále rekonfigurovať obvod a teda zabezpečiť rýchlu implementáciu a testovanie nových štandardov a prototypov. Táto vlastnosť je aj hlavným dôvodom, prečo sú FPGA obvody schopné konkurovať ASIC a dokonca ich prevýšiť v počte používaných obvodov vo svete.

Neustály výskum a rozvoj SRAM technológie vo viacerých oblastiach (nie len v oblasti FPGA) vedie k rozvoju HW FPGA riešení problémov, ktoré bolo doposiaľ možné riešiť len softvérovo kvôli ich komplexnosti, t.j. s obmedzeným výkonom.

SRAM je volatile technológia a teda aktuálna konfigurácia obvodu trvá len počas jeho napájania elektrickou energiou. Po odpojení napájania sa daný stav stratí a obvod je potrebné opäť nakonfigurovať požadovaným návrhom. Proces konfigurácie trvá určitý čas, čo môže byť podstatné v systémoch, ktoré sú časovo náročné a vyžadujú stály chod. Tento problém možno vyriešiť ďalšou samostatnou non-volatile pamäťou (napr. FLASH), kam sa konfigurácia uloží a z tejto pamäte je následné načítaná do obvodu vždy, keď je to potrebné. Takéto riešenie však vyžaduje dodatočný priestor na príslušnej doske pre druhú pamäť.

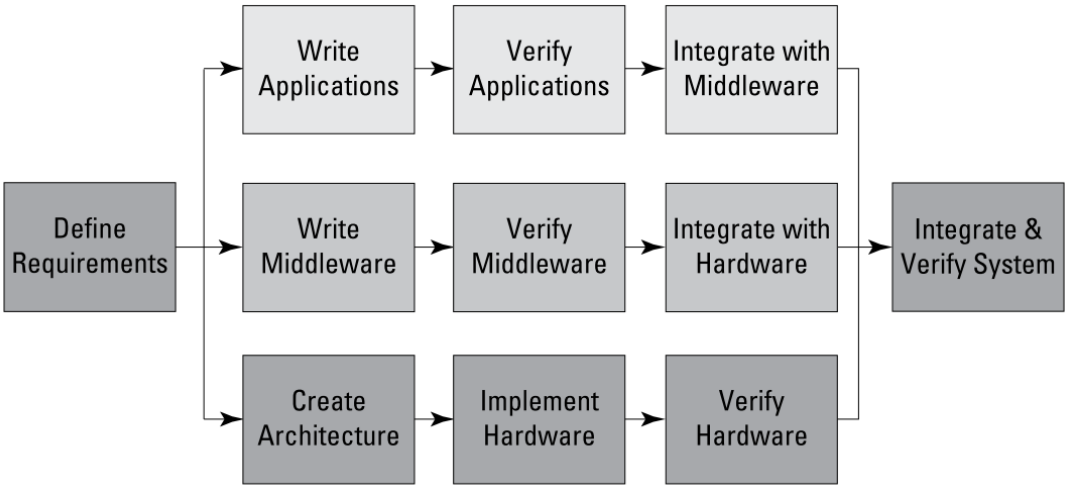
1.3 POSTUP PRI NÁVRHU SYSTÉMU

Návrh systému, ktorý má byť implementovaný do FPGA obvodu, sa veľmi podobá návrhu softvérovej aplikácie, kde postupujeme podľa všeobecne známych a overených krokov. Cieľom tejto časti je poskytnúť prehľadnú postupnosť bodov bez ich podrobnejších popisov. Podrobnejšie informácie o jednotlivých fázach sú uvedené v [**2**].

1. **Definovanie požiadaviek na systém** – na začiatok určíme, aké požiadavky sú na navrhovaný systém kladené a definujeme prislúchajúce funkcie, ktoré má systém poskytovať. Definované požiadavky slúžia aj pri testovaní systému ako východiskové očakávane hodnoty a jeho správanie.
2. **Vytvorenie architektúry** – rozdelíme systém do funkčných blokov, z ktorých každý poskytuje aspoň jednu z predtým definovaných funkcií. Tieto bloky je možné rozdeliť do dvoch skupín podľa toho, či sú realizované softvérovo ako aplikačné programy alebo hardvérovo cez logické moduly. Realizácia oboch skupín prebieha súbežne.
   2.1. **Napísanie, verifikácia a integrácia softvérových aplikácii** – SW aplikácie sú vytvárané (písané v programovacom jazyku) a overované samostatne, t.j. nezávisle od HW. Ak ich overenie prebehne úspešne, sú integrované do hardvérovej časti.
   2.2. **Návrh, implementácia a overenie hardvéru** – navrhneme bloky pre FPGA obvod. Ak je to možné, použijeme už existujúce dostupné IP jadrá s danou funkcionalitou , napr. CPU, DSP, radiče a iné (viď. Stavebné bloky FPGA) alebo vytvoríme (napíšeme) vlastné moduly v jednom z HDL jazykov. Použijeme syntézu a vložíme do FPGA obvodu. Bloky sú následne testované formou simulácie a testov na reálnom zariadení. Pre tento účel je vhodné použitie *testbench* (testovacieho prostredia).

2.3. **Napísanie, verifikácia a overenie middleware vrstvy –** vytvorenie jednotného rozhrania (napr. softvérové ovládače) medzi HW a SW vrstvou, ktoré umožňuje zmenu v jednej vrstve bez potrebnej dodatočnej úpravy druhej vrstvy.

3. **Integrácia a verifikácia systému** – zloženie všetkých častí do spoločného celku – ich integrácia do jedného systému. Následne je overená jeho funkcionalita, či sú splnené všetky požiadavky definované v prvom kroku. Ak systém spĺňa všetky požadované funkcie, jeho realizácia je úspešná. V inom prípade je nevyhnutné vrátiť sa a kroky opakovať.



OBRÁZOK Č. 4: Návrh systému

1.4 FPGA VS. MIKROPROCESOR

Jednou z kľúčových úloh programátora, resp. návrhára je definovanie všetkých častí – funkčných blokov, z ktorých bude budúci systém pozostávať, tak aby spĺňal všetky predtým špecifikované požiadavky (viď. vyššie). Pred realizáciou každého z blokov (prvý krok tvorby architektúry) je vhodné položiť si otázku, či bude konkrétny blok realizovaný ako softvérový program alebo ako hardvérová logická funkcia. Softvérové programy sú vykonávané mikroprocesorom ako postupnosti inštrukcií, hardvérové logické funkcie zase cez logické členy implementované v FPGA obvode.

Jednou z hlavných kritérií je požiadavka na rýchlosť, s akou má byť funkcionalita bloku vykonaná.

- **Veľmi rýchle (pikosekundy a nanosekundy)** – funkcie s takouto požadovanou rýchlosťou nie je možné realizovať softvérovo, keďže mikroprocesor nemá

požadovanú rýchlosť. Jedinou možnosťou pre takéto funkcie a bloky, ktoré ich majú zabezpečiť, je FPGA obvod.

- **Rýchle (mikrosekundy)** – pre implementáciu takýchto funkcií je možné zvoliť ľubovoľnú z vyššie uvedených možností (mikroprocesor aj FPGA). Je úlohou návrhára zhodnotiť, ktorá z nich to bude. Práve rozhodnutia a čas strávený pri nich odlišujú lepšie návrhy od tých horších.
- **Pomalé (milisekundy)** – funkcie s takouto požadovanou rýchlosťou je odporúčané implementovať softvérovo, keďže ich hardvérová realizácia je paradoxne zložitejšia – potrebné spomaliť vykonávanie funkcií, napr. blikanie LED diódy s prislúchajúcou frekvenciou, čo vedie k neefektívnemu využitiu HW zdrojov, ktoré máme k dispozícii.

Okrem rýchlosti rozhodujú aj špecifické požiadavky, ktoré úplne spĺňa len jedna z volieb, napr. potreba **čisto paralelného vykonávania** funkcie bloku (FPGA) alebo jej **komplexnosť** (CPU).

Na základe vyššie uvedeného sú preto v reálnych (zložitejších) systémoch použité obe metódy – HW aj SW. Jedným z príkladov je aj naša NetFPGA karta. FPGA obvod je priamo súčasťou karty. Funkcie mikroprocesora nám v tomto prípade poskytuje hosťujúci počítač (jeho CPU), ktorý vykonáva softvérové programy. Komunikácia medzi FPGA obvodom a CPU je zabezpečená cez PCI-Express zbernicu a prislúchajúci komunikačný protokol. To nám ale nebráni v tom, aby sme mikroprocesor vložili priamo do FPGA obvodu ako tzv. *soft microprocessor core* (napr. Xilinx MicroBlaze / PicoBlaze), ak je to potrebné pre náš systém.

1.5 VYUŽITIE FPGA V REÁLNOM PROSTREDÍ

FPGA obvody sú využívané vo viacerých oblastiach priemyslu a vývoja najmä vďaka ich nízkej cene, minimálnom čase potrebnom na vytvorenie návrhu systémov a vysokej flexibilite v porovnaní s alternatívami, napr. ASIC a DSP.

1.5.1 Prototypy pre ASIC

Počiatočným významom FPGA obvodov bol návrh prototypov, ktoré boli po úspešnej realizácii následne implementované do vtedy výkonnejších ASIC obvodov. Neustály vývoj umožnil vytvorenie oveľa výkonnejších FPGA obvodov, vďaka čomu je dnes možné nahradiť nimi ASIC obvody a použiť FPGA aj v iných odvetviach.

Jednou z hlavných oblastí využitia FPGA je automobilový priemysel, kde požadovaný výkon (spracovanie údajov z kamier a mnohých senzorov v reálnom čase a riadenie) a neustála aktualizácia systémov (nové štandardy) vyžadujú flexibilné HW riešenie – systém implementovaný do FPGA obvodu.

1.5.2 Telekomunikácie

Ďalšou veľkou oblasťou sú telekomunikácie a sieťové inžinierstvo, ktoré neustále riešia problém vysoko-rýchlostného spracovania veľkého množstva údajov, ktoré tečú medzi sieťovými uzlami ako aj stále dôležitejšou bezpečnosťou sietí. V ďalšej časti tejto práce sa zaoberáme analýzou, implementáciou a testovaním sieťových zariadení, ktoré spadajú do tejto oblasti výskumu. Trendom v počítačových sieťach je postupná náhrada softvérových riešení jednotlivých zariadení používaných v sieťach za oveľa výkonnejšie hardvérové riešenia, ktoré poskytujú okrem bežných funkcií typických pre SW riešenia aj mnohé ďalšie. Tie často nie je možné realizovať prostredníctvom mikroprocesorov z dôvodov požadovanej rýchlosti. Medzi tieto zariadenia patria šifrovacie karty, rôzne analyzátory sieťovej prevádzky, IDS, IPS, zariadenia bezdrôtového prenosu ale aj klasické smerovače a firewally.

1.5.3 Heterogénne spracovanie dát

Veľký rozvoj zaznamenáva aj oblasť heterogénneho spracovania údajov. Hlavnou myšlienkou je vzájomná spolupráca viacerých procesorov rôzneho typu bežiacich paralelne na rôznych platformách (CPU, GPU, FPGA, DSP a iné). Každý z týchto procesorov má k dispozícii určité výpočtové zdroje, ktoré ak využívané rozumne, môžu spracovať ohromný objem dát. Tento prístup je nevyhnutný najmä v dátových centrách, kde sa každodenne stretávame so spracovaním tzv. *big data*.

Jednou z metód je dátový paralelizmus, kde sa vstupné dáta rozdelia do viacerých blokov a tie môžu byť paralelne spracované rôznymi procesormi. Inou metódou je úlohový paralelizmus, kde sa samotná úloha (proces) rozloží na jednotlivé paralelne bežiace vlákna. Aby mohol daný proces bežať súčasne na rôznych platformách, je potrebné definovať jednotný programovací jazyk alebo štandard, ktorý je podporovaný každou platformou.

Príkladom je OpenCL ako framework, v ktorom je možné písať programy pre viacero platforiem. OpenCL je podporovaný aj FPGA platformou, ktorá predstavuje ideálneho kandidáta na paralelné vykonávanie. Do FPGA obvodu je možné vložiť ľubovoľný počet soft mikroprocesorových jadier (obmedzené počtom CLB, viď. architektúra FPGA), ktoré pracujú

úplne nezávisle – paralelne. Ďalšou výhodou OpenCL v FPGA je využitie programovacieho jazyka C, v ktorom sú jednotlivé programy písané, takže programátor nemusí rozumieť žiadnemu HDL jazyku. V súčasnosti väčšina výrobcov FPGA podporuje vo svojich SDK aj kompiláciu OpenCL programov.

Zdrojom údajov vyššie uvedených použití FPGA obvodov v reálnej prevádzke a pre veľké množstvo zákazníkov je [**2**].

2 ANALÝZA NETFPGA ZARIADENIA DOSTUPNÉHO NA KIS

V úvode práce je NetFPGA karta popísaná všeobecne, pričom sme sa venovali predovšetkým vysvetleniu FPGA. Tento obvod je totiž (ako už bolo uvedené v kapitole 1) hlavnou časťou karty. Práve v ňom prebieha celý proces návrhu a realizácie, a preto je naň kladený taký dôraz. V tejto časti sa zameriavame na podrobnejšiu analýzu karty ako celku a vypracujeme jej popis. Rozoberieme konkrétne prvky, z ktorých pozostáva, aké funkcie poskytujú a ako ich možno využiť pri realizácii sieťových zariadení. Všetky prvky NetFPGA karty, na ktoré sa v nasledujúcej časti odkazujeme, sú vyznačené na obrázku č. 5 uvedenom v závere tejto kapitoly.

2.1 HARDVÉROVÁ ŠPECIFIKÁCIA

Modelové označenie karty, ktorú máme k dispozícii, je **NetFPGA-1G-CML** od výrobcu Digilent. Karta obsahuje **Kintex-7 XC7K325T** FPGA obvod, ktorý vykonáva samotnú logiku systému. Pre sieťovú komunikáciu sú k dispozícii **štyri ethernetové porty** s podporovanou rýchlosťou až do 1Gb/s. Ako pamäťové úložisko dát nám slúži 512 MB **DDR3 RAM** a 4,5 MB **QDRII+ SRAM**. Okrem týchto dynamických pamätí môžeme použiť externú **SD kartu**, ktorú vložíme do NetFPGA karty cez príslušný slot. **BPI Flash** pamäť s veľkosťou 128 MB umožňuje uložiť konkrétnu konfiguráciu, následne slúži ako jeden zo zdrojov pre konfiguráciu FPGA obvodu. **PCIe x4 Gen 2** konektor umožňuje vložiť kartu do väčšiny súčasných matičných dosiek a prepojiť tak kartu s PC. Cez **FMC** a **PMOD** konektory môžeme pripojiť rôzne rozširiteľné karty, ktoré prinášajú dodatočnú funkcionalitu v oblasti komunikácie, merania a riadenia.

2.1.1 Zdroj napájania

NetFPGA karta vložená do PCIe slotu počítača nie je priamo napájaná cez túto zbernicu a teda vyžaduje samostatný zdroj napájania s parametrami 12V a 5A. K tomuto účelu môžeme použiť ľubovoľný ATX zdroj používaný v pevných počítačoch. Pre pripojenie napájania ku karte je použitý 6-pinový **molex konektor** rovnako ako pri výkonných grafických kartách. Následne nesmieme zabudnúť prepnúť mechanický prepínač na karte (vedľa molex konektora) do polohy **POWER ON**. Ak plánujeme využívať kartu v tzv. *standalone* režime (nezapojenú do PC cez PCIe), musíme v hlavnom 20-pinovom konektore zdroja použiť jumper na pinoch 15 a 16, aby sme povolili napájanie na ATX jednotke. Túto

následne pripojíme rovnako ako v predchádzajúcom prípade. Jednotlivé zariadenia môžeme bezpečne pripájať a odpájať aj za behu karty, t.j. keď je napájanie aktívne.

2.1.2 Oscilátory

Pre potreby synchronizácie prvkov FPGA obvodu a periférii sa na karte nachádzajú viaceré oscilátory, napr. 125 MHz oscilátor pre sieťové ethernetové radiče (**PHY**). Hlavné systémové hodiny FPGA obvodu sú riadené cez 200 MHz diferenčný oscilátor, ktorý je do FPGA obvodu privedený cez dva piny a riadi jednotlivé **DCM** (Digital Clock Manager) bloky.

2.1.3 FPGA vstavané pamäte

Súčasťou FPGA obvodu sú aj vstavané pamäte – **Block RAM (BRAM)** 36Kb pamäte o celkovej veľkosti 1,78 MB (445 * 36Kb) , z ktorých každá môže byť rozdelená do dvoch úplne nezávislých 18Kb RAM pamätí. Jedným z ich použití je FIFO štruktúra vhodná pre prenos údajov z jedného funkčného bloku do ďalšieho. Nastavenia BRAM pamätí v danom návrhu je možné meniť cez návrhové prostredie, napr. Xilinx ISE Design Suite.

2.1.4 DDR3 pamäť

NetFPGA karta využíva **Micron MT41K512M8 DDR3 SDRAM** pamäť o veľkosti 512 MB a frekvencii 800 MHz s 8-bitovou dátovou zbernicou. Jedným z využití tejto pamäte je dočasné ukladanie paketov do frontu s podporou vysokej priepustnosti (1600 MT/s). Konfiguráciu rozhrania medzi FPGA obvodom a pamäťou zabezpečuje nástroj - Xilinx **Memory Interface Generator** (MIG). Ten automaticky nastaví dané rozhranie tak, aby mohlo byť použité s AXI4-System zbernicou.

2.1.5 QDRII+ pamäť

Okrem DDR3 je k dispozícii aj 4,5 MB **Cypress CY7C2263KV18 QDRII+ SRAM** pamäť s frekvenciou 450 MHz (900 MT/s), ktorá je schopná súčasne prenášať údaje cez READ aj WRITE porty na oboch hranách hodinového signálu (3,6 Gb/s). Táto pamäť je používaná pre FIFO fronty a Look-up tabuľky, ktoré vyžadujú vysokú rýchlosť a nízke oneskorenie, napr. prepínacia alebo smerovacia tabuľka. Vytvorenie a konfigurácia rozhrania medzi pamäťou a FPGA obvodom sú aj v tomto prípade vykonané MIG nástrojom.

2.1.6 BPI-Flash pamäť

Ďalšou pamäťou je **Numonyx BPI (Byte Peripheral Interface) flash** pamäť o veľkosti 128MB. Táto pamäť je non-volatile, a preto je jej hlavným využitím vysoko-rýchlostná konfigurácia FPGA obvodu. Vďaka pomerne veľkej kapacite je možné uložiť do tejto pamäte viacero konfiguračných súborov súčasne a tie následne dynamicky riadiť (cez externé zariadenie, napr. mikroprocesor alebo CPLD) tak, aby bol FPGA obvod nakonfigurovaný vždy podľa aktuálnych potrieb. Po úspešnej konfigurácii je možné uložiť sem ľubovoľné údaje a využiť tak non-volatile charakter tejto pamäte – nastavenie BPI konfiguračných pinov ako bežné I/O piny.

2.1.7 SD karta

NetFPGA karta obsahuje slot pre vloženie SD karty a zabezpečuje jej podporu (všetky požiadavky fyzickej vrstvy pre SPI aj SD protokoly). Vložená SD karta následne slúži ako dodatočná non-volatile pamäť.

2.1.8 PCI-express rozhranie

Jedným z kľúčových rozhraní pre naše potreby je PCIe, ktorým možno pripojiť kartu k PC. Pre účely PCIe zbernice sú použité štyri vysoko-rýchlostné sériové GTX vysielače/prijímače (viď. prvá kapitola), ktoré tvoria štyri samostatné Gen 2 PCIe komunikačné linky. Tie slúžia na komunikáciu NetFPGA karty a jej FPGA obvodu s CPU hostiteľského počítača. PCIe rozhranie v karte je riadené vysoko-výkonným **PCI Express I/O** jadrom (Integrated Block for PCIe V2.0). Toto jadro je možné vložiť do návrhu a konfigurovať cez Xilinx ISE Coregen alebo Vivado Design Suite nástroje.

2.1.9 Ethernet PHY čipy

Ďalší spôsob komunikácie NetFPGA karty s okolím prinášajú štyri **Realtek RTL8211** ethernetové čipy (PHY), ktoré poskytujú rozhranie pre sieťovú komunikáciu cez **RJ-45** konektory. Súčasťou každého z nich sú dve LED diódy, ktorých správanie je predvolene nastavené tak, aby bolo rovnaké ako u bežných sieťových kariet (nanovo nastavené pri každom resete). PHY je možné programovať cez spoločnú **MDIO** (Management Data Input/Output) zbernicu. Jednotlivé konektory sú dostupné cez príslušné adresy **ETH1** až **ETH4**. Každý z PHY môže na základe vykonaného auto-negotiation procesu nezávisle prispôsobiť svoju dátovú rýchlosť na 10/100/1000 Mb/s. Na prenos údajov je použité **RGMII** (Reduced Gigabit Media Independent Interface) rozhranie používajúce frekvenciu 125 MHz.

Aby bolo možné použiť tieto sieťové konektory, je potrebné do návrhu pridať **Ethernet MAC Xilinx IP** jadro.

2.1.10 PIC Mikroradič

Súčasťou karty je aj 32-bitový PIC mikroradič. Ten je predvolene nastavený pre riadenie **USB OTG** (On-The-Go), kde môže vystupovať ako hosťujúce alebo hostiteľské zariadenie (Slave/Master). To mu umožňuje načítať konfiguračný súbor z externého USB flash disku. Okrem toho je zodpovedný za správu **Real-Time** hodín a **Secure Storage** (zabezpečenie bitstream súboru pred neoprávneným čítaním). Funkcionalita mikroradiča môže byť preprogramovaná používateľom, ak je to potrebné, napr. ak chce používateľ využiť aj iné komponenty pripojené k mikroradiču. Ten komunikuje s ostatnými komponentmi cez **I2C** (Inter-Integrated Ciruit) zbernicu použitím špecifických I2C adries.

2.1.11 LED diódy a tlačidlá

Programátor má prístup aj k štyrom tlačidlám a štyrom LED diódam, ktoré sú umiestnené na karte. Tie je možné ovládať cez GPIO piny. Okrem nich sú k dispozícii špeciálne červené tlačidlá **BTN4-RESET** (vyvolá reset celého návrhu) a **BTN5-PROG** (vyvolá konfiguračný proces, viď. nižšie).

2.1.12 PMOD a FMC rozširujúce konektory

Okrem komponentov, ktoré sú priamo súčasťou karty, je možné pripojiť ďalšie. Ich úlohou je pridanie dodatočnej funkcionality, ktorá je potrebná pre daný návrh. K tomu slúžia **PMOD** a **FMC** konektory umiestnené na karte. Príkladom PMOD karty je **PmodUSBUART** – konverzia z UART (Universal Asynchronous Receiver and Transmitter) na micro-USB. Táto karta je súčasťou balenia NetFPGA-1G-CML karty, ktorou disponuje KIS.

Podrobnejšie informácie k vyššie popísaným HW prvkom NetFPGA karty sú uvedené v [**3**].

2.2 CIELE VYUŽITIA V POČÍTAČOVEJ SIETI

Vzhľadom na vyššie uvedené HW prvky NetFPGA karty je našim cieľom v tejto práci preskúmať a vytvoriť také sieťové HW zariadenia (sieťové systémy), ktoré využívajú väčšinu týchto HW periférií. Prostredníctvom návrhu, vytvorenia týchto systémov a ich implementácie do NetFPGA karty sme schopní predviesť viaceré funkcie, ktoré nám karta ponúka a poukázať tak na možné využitie tejto technológie aj v oblasti počítačových sietí.

Realizované sú základné sieťové zariadenia, ktoré tvoria bežnú súčasť dnešných počítačových sietí. Vďaka tomu je ich požadovaná funkcionalita dobre známa, čo vedie k jednoduchšiemu testovaniu a verifikácii správania hotových systémov. Príkladmi sú NIC (Network Interface Controller) karta, Learning-CAM prepínač a smerovač (prípadne ich modifikácie). V kapitole 4 sa podrobnejšie venujeme spoločnej architektúre týchto zariadení a v kapitole 6 uvedieme ich špecifické funkcie.

2.3 INTEGRÁCIA DO PEVNÉHO STOLOVÉHO POČÍTAČA

Pre dosiahnutie jednotlivých cieľov uvedených vyššie, ktoré predstavujú HW projekty a ich implementáciu do NetFPGA karty, sme použili pevný stolový počítač. Tu je takisto realizované prostredie, na ktorom prebieha ich vývoj a testovanie. Hardvérová špecifikácia (zloženie počítača) a riešenia problémov, ktoré sa objavili pri vložení karty do PC, sú uvedené v tejto časti.

2.3.1 Hardvérová špecifikácia PC

Hlavnou časťou počítača je matičná doska s voľným PCIe slotom pre vloženie NetFPGA karty. Požadovaný PCIe slot musí podporovať minimálne **Gen 2** a mať rozmery aspoň **x8** (rozmery PCIe konektora NetFPGA karty sú x8, ale piny sú umiestnené len do polovice konektora - **x4**). Kartu je možné vložiť aj do slotu väčších rozmerov bez nebezpečenstva poškodenia. V našom prípade sme použili voľný **x16** slot umiestnený na matičnej doske. Táto doska podporuje NetFPGA kartu a je schopná rozpoznať ju ako PCIe zariadenie v prípade, že je vložená karta nakonfigurovaná funkčným návrhom. Model dosky je **Gigabyte GA-X79-UD3**. Procesor počítača s označením **Intel(R) Core (TM) i7-3820** komunikuje s kartou cez PCIe zbernicu, ak je použitý vhodný ovládač. Jeho úlohou je vykonávať softvérovú časť projektov – programy, ktoré riadia správanie karty, napr. zmena obsahu registrov. V počítači sú vložené Kingston RAM pamäte o celkovej veľkosti 12 GB. Napájanie počítača a NetFPGA karty zabezpečuje samostatný zdroj.

2.3.2 Vloženie karty do PC

Vloženie NetFPGA karty do počítača je jednoduché, nevyžaduje takmer žiadne špeciálne opatrenia a je porovnateľné s vložením bežnej grafickej karty. Jedinou požiadavkou je miesto na matičnej doske, kam kartu vložiť a voľný molex konektor vedúci zo zdroja napájania. Pred samotným vložením karty je vhodné prečítať si problémy, ktoré môžu nastať. Nižšie sú uvedené nami identifikované problémy a ich riešenia.

Postup vloženia karty:

1. Vložíme kartu do voľného PCIe slotu. Ten musí spĺňať parametre uvedené v HW špecifikácii PC.
2. Pripojíme molex konektor zo zdroja napájania a prepneme mechanický **POWER switch** na karte do polohy **POWER ON**.
3. Zapneme počítač a pracujeme s kartou podľa vlastných potrieb.

2.3.3 Riešenie HW problémov

Vložením karty do PCIe slotu sa objavili problémy – neočakávané správanie, ktoré bolo potrebné vyriešiť. Tu sú uvedené problémy, ktoré sme doposiaľ identifikovali a vyriešili.

- **Zatuhnutie PC** - vložením karty do PCIe slotu, pripojením napájacieho konektora zo zdroja a zapnutím počítača sa po určitej chvíli objavil problém v podobe jeho zatuhnutia, pričom sa obrazovka vypla a počítač nereagoval. Tento problém nastane, ak karta nie je nakonfigurovaná funkčným návrhom ešte pred zapnutím PC. Riešením je vloženie ľubovoľného funkčného konfiguračného súboru. My sme stiahli jeden (**reference_nic_nf1_cml.bit** – implementácia NIC karty) z GIT repozitára NetFPGA komunity [**4**] a ten sme podľa postupu uvedeného nižšie vložili cez USB flash disk do karty. Tento spôsob je najjednoduchší a v prípade novej rozbalenej karty, ktorá ešte neobsahuje konfiguračný súbor, je aj jedinou možnosťou, ak nepoužijeme **standalone** režim (obe zvyšné metódy vyžadujú bežiaci počítač, cez ktorý konfiguráciu vykonáme, čo je ale nemožné z dôvodu jeho zatuhnutia).
- **Opätovné zapnutie PC** – počítač opäť nabehne aj v prípade, že sme predtým vykonali jeho vypnutie. Problémom je **Wake-Up** pin PCIe karty, ktorý po vypnutí napájania okamžite zobudí počítač. Riešením tohto nevhodného správania je vypnutie **PME Wake-Up** funkcie v BIOS-e počítača. Táto voľba sa nachádza v *Power Management* časti BIOS menu. Ak tam z nejakých dôvodom nie je (náš prípad), je potrebné vypnúť celkovú funkcionalitu **PME Eventov** tým, že povolíme **ErP Support**.

2.4 PIORITA ZDROJOV PRE KONFIGURÁCIU KARTY

Ako už bolo uvedené v predchádzajúcej časti, pred samotným vložením karty do PCIe slotu, pripojením napájania a zapnutím počítača musí karta obsahovať funkčnú konfiguráciu –

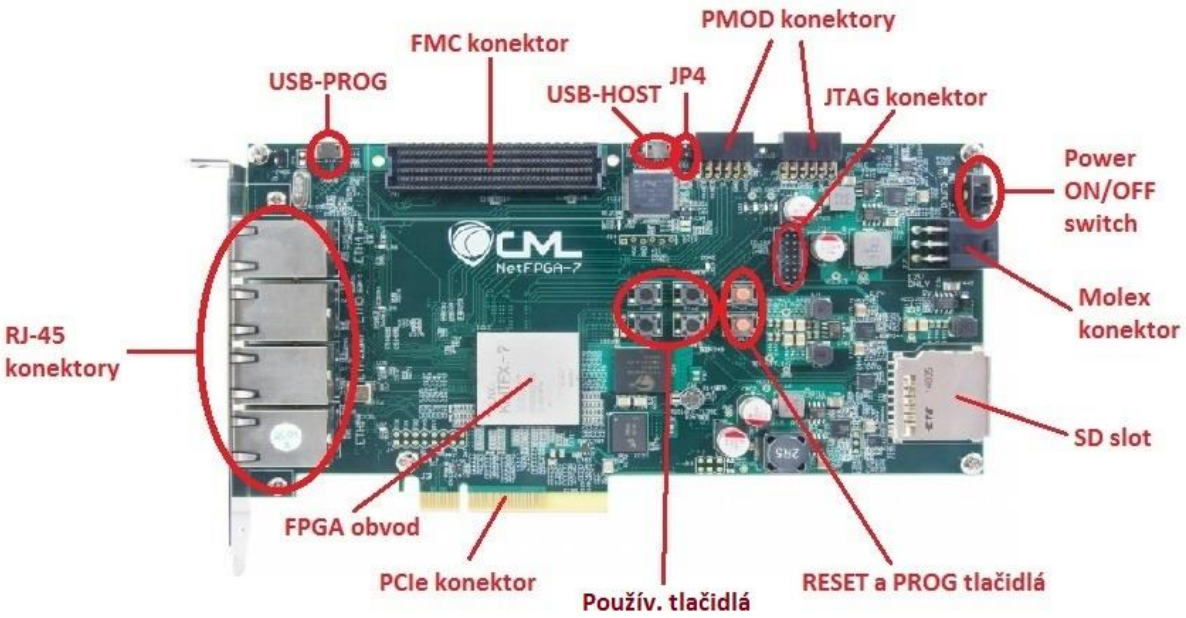
konfiguračný súbor s príponou **.bit** alebo **.msc** známy ako bitstream. To je možné dosiahnuť viacerými spôsobmi.

Podobne ako počítač môže použiť viacero zdrojov pre načítanie operačného systému, tzv. OS boot priority nastaviteľný v BIOS-e, aj karta má pevne definované poradie zdrojov, v akých hľadá konfiguračný súbor. Týmito zdrojmi sú externý USB flash disk pripojený ku karte cez vhodný konektor, BPI Flash pamäť a konfigurácia cez PC. Samotný proces bootovania je vykonaný vždy, keď pripojíme napájanie alebo použijeme červené tlačidlo **PROG** (BTN5) umiestnené priamo na karte. Poradie zdrojov je pevné a nemožno ho meniť:

1. **USB flash disk** – tento spôsob je vhodný pre občasnú konfiguráciu (najmä pre prvú konfiguráciu karty, kedy potrebujeme vložiť do FPGA obvodu ľubovoľný funkčný systém, aby sme s kartou mohli v PC pracovať). USB flash disk musí spĺňať určité podmienky – musí byť formátovaný ako **FAT** (nie FAT32) súborový systém a obsahuje jediný súbor **download.bit**, ktorý je funkčným konfiguračným súborom vhodným pre náš typ FPGA obvodu. USB flash disk pripojíme ku NetFPGA karte cez na to určený kábel (Micro-male to A-female USB adaptér) pripojený druhým koncom do **USB-HOST** konektora. Aby bol USB disk napájaný, je potrebné vložiť do **JP4** konektora jumper. Ak nie je splnený aspoň jeden z predchádzajúcich bodov, tento zdroj konfigurácie je ignorovaný.
2. **BPI-Flash** – tento spôsob umožňuje veľmi rýchlu konfiguráciu, čo je pri viacerých špecializovaných systémoch nevyhnutné (vzhľadom na časové obmedzenia). Hlavnou výhodou je fakt, že BPI Flash pamäť je **non-volatile**, a teda konfiguračný súbor zostáva v pamäti aj po odpojení napájania a môže byť použitý ľubovoľne veľakrát. Konfigurácia BPI-Flash pamäte prebieha cez USB kábel pripojený do **USB-PROG** konektora alebo JTAG kábel pripojený do **XILINX PROG CABLE** konektora. Samotný prenos údajov riadi **iMPACT** nástroj. Bitstream súbor má príponu **.msc**.
3. **PC cez USB/JTAG kábel** – ak ani jeden z predchádzajúcich zdrojov úspešne nenakonfiguroval kartu, tá zostane neaktívna, až kým nie je nakonfigurovaná cez PC. Tento postup je pomalší, ale umožňuje neustálu rekonfiguráciu bez nutnosti opakovať celý proces bootovania. Preto je oveľa častejšie používaný než predchádzajúce metódy najmä v procese návrhu prototypov a ich testovania. Ak už bola karta úspešne nakonfigurovaná jedným z predchádzajúcich zdrojov, týmto spôsobom môžeme jej aktuálnu konfiguráciu hocikedy prepísať. Samotný proces tejto konfigurácie je z pohľadu programátora rovnaký

ako pri programovaní BPI flash pamäte v predchádzajúcom bode, keďže sa o všetko postará iMPACT. Odlišný je len bitstream súbor, ktorý má v tomto prípade príponu **.bit**.

Pre prehľadnosť a jednoduché nájdenie požadovaného prvku na NetFPGA karte je nižšie uvedený obrázok. Na ňom sú jasne vyznačené všetky tie časti, na ktoré sa odkazujeme v tejto práci.



OBRÁZOK Č. 5: Popis NetFPGA karty

3 REALIZÁCIA VÝVOJOVÉHO A TESTOVACIEHO PROSTREDIA

V tejto kapitole je popísané nevyhnutné softvérové vybavenie prostredia PC, ktorého HW špecifikácia bola popísaná v predchádzajúcej kapitole. Toto PC bude slúžiť spolu s osadenou kartou ako vývojové a testovacie prostredie NetFPGA, v ktorom budú vykonávané všetky potrebné kroky počas testovania a tvorby jednotlivých sieťových systémov. Tie budú následne vložené do NetFPGA karty a funkčne pretestované. Súčasťou prostredia sú operačný systém, jednotlivé vývojové prostredia – softvérové nástroje určené na tvorbu návrhov pre FPGA obvody ako aj GIT repozitár NetFPGA projektu. Ten obsahuje úplnú súborovú štruktúru – zdrojové a konfiguračné súbory všetkých východiskových projektov, ktorým sa v práci venujeme. V tomto prostredí sme súčasne vytvorili testovacie prípady, ktoré dokazujú korektnú funkčnosť každého z realizovaných návrhov testovaných sieťových systémov.

3.1 POPIS PROSTREDIA

Základom prostredia je operačný systém. Pre vývoj bol zvolený operačný systém **LINUX**. Hlavným dôvodom tejto voľby je ovládač (modul jadra) PCIe zbernice našej NetFPGA karty, ktorý je v súčasnosti k dispozícii len pre Linux. Pre Windows existuje ovládač len pre starší model NetFPGA karty, ktorý nie je kompatibilný s naším modelom. Ďalším dôvodom je dostupnosť rôznych nástrojov ako aj prístup ku zdrojovým súborom systému v prípade ich potreby. V neposlednom rade sú to aj skúsenosti s vývojom softvérových aplikácii v tomto systéme v porovnaní s inými operačnými systémami ako napr. Windows, takže v prípade problémov sme schopní nájsť riešenie oveľa skôr.

V rámci Linuxu bola zvolená distribúcia **Fedora verzie 21**. Táto distribúcia (avšak v nižšej verzii) bola použitá pri vytvorení všetkých východiskových projektov v [**4**], odkiaľ boli získané a použité všetky zdrojové kódy testovaných projektov. Súčasne je táto distribúcia odporúčaná aj v [**5**], kde sú uvedené návody popisujúce jednotlivé kroky práce s NetFPGA kartou (dôležitý zdroj informácii, o ktoré sme sa do veľkej miery opierali aj pri tvorbe tejto práce). Okrem toho je vhodné používať rovnaké prostredie ako väčšina programátorov, ktorí sa venujú tejto oblasti – návrh sieťových systémov pre NetFPGA-1G-CML. To zaručuje väčšiu pravdepodobnosť, že v prípade problémov nájdeme ich riešenie priamo na uvedených odkazoch alebo nám bude riešenie v krátkom čase poskytnuté na príslušnom fóre.

Ako grafické pracovné prostredie sme zvolili **XFCE** pre jeho jednoduchosť a nenáročnosť, ale nie sú uvedené žiadne obmedzenia výberu, a teda programátor môže zvoliť ľubovoľné z existujúcich prostredí, ktoré mu vyhovuje.

Pre tvorbu softvérových aplikácií, ktoré sú súčasťou realizovaných sieťových systémov nad NetFPGA a sú vykonávané procesorom tohto počítača, sme zvolili integrované vývojové prostredie (IDE) **Eclipse** z dôvodov podpory vývoja C/C++ ako aj Java aplikácií. Z nášho pohľadu sa jedná o graficky nenáročné prostredie, ktoré poskytuje veľké množstvo dodatočných funkcií vrátane možnosti prehľadného ladenia.

Pre prístup ku zdrojovým súborom, ktoré sa nachádzajú v [**4**], bol použitý nástroj **git**. Je to nástroj pre prácu s GIT repozitármi, ktorý poskytuje rozhranie len vo forme príkazového riadku. Z dôvodu zložitosti NetFPGA repozitára sme preto použili aj jeho grafický front-end **qgit**. Ten sprehľadňuje prácu v rámci repozitára a umožňuje tak jednoduchšiu manipuláciu so súbormi, ktoré sa v ňom nachádzajú.

Jedným z rozhraní, ktoré môžeme použiť pre vzájomnú komunikáciu medzi PC a NetFPGA kartou, je UART rozhranie. Textovo-orientovaný nástroj, ktorý sme použili ako klienta pre riadenie prístupu k tomuto rozhraniu, je **minicom –** linuxová alternatíva HyperTerminálu známeho v OS Windows pre prístup a riadenie sériovej komunikácie portu PC.

Nevyhnutnou súčasťou práce v oblasti počítačových sietí je nástroj, ktorý je schopný zachytiť údaje – rámce, ktoré tečú sieťovými rozhraniami. Pre tento účel sme použili nástroje **tcpdump**, ktorý môžeme jednoducho ovládať priamo z príkazového riadku a **Wireshark**, ktorý nám poskytuje prehľadné grafické prostredie s veľkým množstvom funkcií. Pre účely testovania sme použili **hping3** a **ncat** nástroje, ktoré slúžia ako generátory prevádzky. Vhodným nastavením parametrov môžeme otestovať výkon jednotlivých NetFPGA projektov, ktorým sa v práci venujeme. Okrem tohto spôsobu testovania je potrebný **scapy -** program pre manipuláciu so sieťovou prevádzkou, ktorý je použitý v jednotlivých testovacích skriptoch umiestnených v popísanom NetFPGA repozitári.

Vyššie uvedené softvérové nástroje sú dostupné vo forme RPM balíčkov. Pre ich inštaláciu sme použili nástroj **yum**. Okrem nich je pre realizáciu projektov, ktoré používajú modul CAM (Content Addressable Memory) tabuľky, potrebná inštalácia **glibc.i686** a **zlib.i686** balíčkov.

Celý proces návrhu systémov pre FPGA obvody prebieha v niekoľkých nástrojoch, ktoré sú špeciálne navrhnuté pre tento účel. Spoločne tvoria IDE podobne ako v prípade softvérových aplikácii. Každý z výrobcov má svoje špecifické IDE. V našom prípade sme použili **Xilinx ISE (Integrated Software Environment) Design Suite 14.7**. Jeho súčasťou sú viaceré nástroje, ktoré potrebujeme v jednotlivých krokoch návrhu. Okrem tohto prostredia poskytuje Xilinx aj tzv. **Lab Tools**. Jedná sa o **EDK** (Embedded Development Kit) a **SDK** (Software Development Kit) sady nástrojov.

3.2 POPIS NÁSTROJOV POSKYTOVANÝCH SPOLOČNOSŤOU XILINX

Xilinx ISE Design Suite riadi všetky aspekty návrhu systému prostredníctvom rôznych špecifických nástrojov. V tejto časti v krokoch popíšeme postup realizácie systému a následne uvedieme stručný popis tých nástrojov, ktoré pri tom používame. Tie zahŕňajú **Project Navigator**, **Xilinx Platform Studio** a **iMPACT**. Nakoniec sú uvedené zdroje a licencia, ktoré sú potrebné pre prácu s týmito nástrojmi.

3.2.1 Postup tvorby návrhu v Xilinx ISE

V tejto časti sú uvedené jednotlivé kroky, ktoré je potrebné vykonať, aby sme z návrhu získali systém, ktorý môžeme následne vložiť do FPGA obvodu.

1. **Vytvorenie projektu a HDL súborov** – napísanie jednotlivých modulov v jednom z HDL jazykov a pridanie súboru s používateľskými obmedzeniami implementácie (definovanie časových obmedzení a priradenie vstupno-výstupných pinov návrhu fyzickým pinom FPGA obvodu) [ISE Text-editor].
2. **Vytvorenie súboru s pravidlami testovania v HDL jazyku a vykonanie RTL (Register Transfer Level) simulácie** – testovanie správnosti HDL kódu [**ISim**, **ModelSim**].
3. **Syntéza (resp. logická syntéza)** – softvérová transformácia HDL modulov na zoznamy použitých primitívnych logických hradiel a klopných obvodov [**XST - Xilinx Synthesis Technology**].
4. **Implementácia** – proces pozostávajúci z troch krokov: preklad, mapovanie a PAR (Place And Route).
   4.1. **Preklad** zlúči jednotlivé zoznamy syntézy do jedného spoločného **netlist** súboru.
   4.2. **Technologické mapovanie** mapuje prvky z netlist súboru do LC a IOB FPGA obvodu.

4.3. **PAR** umiestni tieto bunky, resp. bloky do fyzických pozícií FPGA obvodu a definuje prepoje medzi nimi. Na konci tohto procesu je vykonaná časová analýza, ktorej výsledkom sú rôzne časové parametre.

5. **Generovanie konfiguračného súboru** – vytvorenie bitstream súboru.

6. **Stiahnutie súboru do reálneho FPGA obvodu** [**iMPACT**] - vloženie konfiguračného súboru do FPGA obvodu a konfigurácia NetFPGA karty.

3.2.2 Project Navigator

Bežný front-end pre Xilinx ISE Design Suite je grafické rozhranie nazvané **Project Navigator**, ktoré umožňuje transparentný prístup ku všetkým potrebným softvérovým nástrojom z jedného miesta (používateľ nespozoruje použitie špecifických nástrojov). Toto rozhranie poskytuje vo forme zoznamu procesov všetky potrebné kroky, ktoré je nutné vykonať, aby sme prešli od napísania modulov v HDL jazyku až po vygenerovanie konfiguračného súboru a jeho stiahnutie do FPGA obvodu, tak ako je uvedené vyššie.

Project Navigator ako väčšina podobných nástrojov (napr. XPS) podporuje **auto-make** technológiu, ktorá zabezpečí, že pri výbere konkrétneho procesu z vyššie uvedeného zoznamu sú automaticky vykonané aj všetky predchádzajúce ešte nevykonané procesy. Vďaka tomu je vždy zabezpečené ich správne poradie. V [**6**] a [**7**] sú uvedené podrobnejšie informácie aj s popisom prostredia.

3.2.3 Xilinx Platform Studio

Jedným z kľúčových komponentov, ktoré sú súčasťou **EDK**, je **Xilinx Platform Studio (XPS)**. Tento front-end pomáha programátorom s návrhmi systémov, ktorých súčasťou je aj ľubovoľný mikroprocesor napr. Microblaze. Problémom, ktorý sťažuje návrh takýchto systémov, je samotná konfigurácia a zabezpečenie korektného prepojenia mikroprocesora s ostatnými modulmi systému. XPS je vhodným riešením, ktoré zjednodušuje tvorbu, prepojenie a konfiguráciu systémov so vstavaným mikroprocesorom. Súčasťou XPS sú rôzne grafické pohľady na systém, ktoré sprehľadňujú celý proces návrhu, ako aj funkcie jednoduchého vloženia ďalších modulov do systému a ich konfigurácie.

Po podrobnejšom preskúmaní jednotlivých projektov umiestnených v [**4**] sme zistili, že súčasťou každého z týchto projektov je aj Microblaze soft-procesor, a teda jedná sa o vstavané systémy, ktorých návrh je realizovaný v XPS. Preto pre prácu s danými návrhmi NetFPGA implementácií použijeme aj EDK.

3.2.4 iMPACT

Celý proces návrhu je zbytočný, ak nie sme schopní takto získaný konfiguračný súbor vložiť do FPGA obvodu a tak zabezpečiť, že sa FPGA obvod bude skutočne správať podľa nášho návrhu. Príkladom softvérového nástroja, ktorý slúži tomuto účelu, je **iMPACT**. Aj vďaka grafickému používateľskému prostrediu sa jedná o pomerne jednoduchý a používateľský prívetivý nástroj, ktorý nám umožňuje pracovať s FPGA obvodom cez pripojený programovací kábel (JTAG, USB). Okrem prehľadného grafického prostredia môžeme použiť rozhranie príkazového riadku vrátane použitia skriptov s popisom príkazov, ktoré má iMPACT vykonať (batch mode). IMPACT je schopný automaticky rozpoznať pripojený obvod, vykonať boundary scan (testovanie funkčnosti obvodu) a pre nás to najdôležitejšie – nakonfigurovať tento obvod vrátane vloženia konfiguračného súboru do BPI-Flash pamäte.

Výhodou je aj vzájomná spolupráca tohto nástroja z vyššie uvedenými prostrediami. Ako príklad uvedieme možnosť automatickej konfigurácie obvodu z ISE/XPS prostredia bez nutnosti spustiť iMPACT samostatne. Podrobnejšie informácie o tomto nástroji sú uvedené aj v [**8**].

3.2.5 Zdroje a licencia od spoločnosti Xilinx

Xilinx ISE Design Suite je spolu s Lab Tools dostupný na [**9**] ako jeden balík, ktorý je možné bezplatne stiahnuť. Jedinou podmienkou je vytvorenie účtu na Xilinx stránke. V inštalácii tohto prostredia je zahrnutá aj inštalácia jednotlivých nástrojov uvedených vyššie. Výhodou tohto prístupu je fakt, že programátor nemusí vyhľadávať a inštalovať jednotlivé nástroje zvlášť, keďže tie sú všetky súčasťou jedného balíka.

Inštalácia daného vývojového prostredia je neobmedzená. Aby sme ale mohli naplno využívať vyššie uvedené nástroje, musíme mať platnú licenciu. My sme využili Xilinx University Program (XUP) a v mene našej univerzity sme pre tento projekt získali licenciu bezplatne. Vďaka tomu môžeme pracovať s jednotlivými nástrojmi podľa potreby.

3.2.6 Inštalácia ovládačov pre programovacie káble

Pri inštalácii vývojových komponentov sa objavil problém, ktorý súvisí s inštaláciou ovládačov pre programovacie káble slúžiace na konfiguráciu FPGA obvodu cez iMPACT nástroj. Ovládače od Digilent, ktorý je výrobcom aj našej NetFPGA karty, nie sú automaticky

nainštalované spolu s Xilinx ISE, a preto je potrebné vykonať v termináli (bash) nasledujúce príkazy.

$ cd /opt/Xilinx/14.7/ISE_DS/ISE/bin/lin64/digilent/
$ sudo ./install_digilent.sh
$ sudo cp –r /root/.cse ~/
$ sudo chown –R <použív. meno>:users ~/.cse

Okrem týchto príkazov je potrebné spustiť *xilinx-usb-progammer-install.sh* skript. Ten je spolu s podrobnejším postupom dostupný na [**10**].

3.3 PROJEKT NETFPGA.ORG

V tejto podkapitole sú popísané zdroje už existujúcich projektov pre NetFPGA kartu, ktoré sú vyvíjané NetFPGA komunitou.

3.3.1 Prístup ku zdrojovým kódom

V práci využívame a nadväzujeme na už hotové riešenia NetFPGA projektu uvedeného v [**4**]. Tento projekt existuje vo forme GIT repozitárov, z ktorých každý je určený pre špecifický model NetFPGA karty. Repozitár pre model nami používanej karty je https://github.com/NetFPGA/NetFPGA-1G-CML-live. Aby sme získali priamy prístup do tohto repozitára, boli sme povinní registrovať sa na github.com a následne požiadať o registráciu na netfpga.org. Potvrdením registrácie sme získali úplný prístup, cez ktorý sme vyklonovali celý repozitár do nášho lokálneho počítača. Tento repozitár pozostáva z viacerých vetiev. Vývoj prebieha nad vetvou **develop**, ktorá obsahuje najnovšie verzie súborov. Preto sme ju hneď po vyklonovaní nastavili ako aktívnu vetvu. Takto môžeme robiť zmeny bez toho, aby sme poškodili súbory, ktoré sa nachádzajú na vzdialenom serveri. Tento prístup použitia GIT repozitárov je v súčasnosti bežný najmä v prípade vývoja voľne dostupných softvérových projektov, ku ktorým majú súčasne prístup mnohí vývojári. Pre operácie nad lokálnym repozitárom používame nástroj git. Podrobnejšie informácie pre prácu s týmto nástrojom sú uvedené v [**11**]. Pre používateľov, ktorí preferujú grafické nadstavby, je k dispozícii vyššie uvedený qgit. V takto získanom lokálnom repozitári (súborový adresár) sa nachádzajú všetky potrebné súbory pre rozbehanie referenčných systémov, ktorými sa zaoberáme v práci, a na ktoré sa priamo odkazujú návody a postupy uvedené v [**5**].

3.3.2 Popis štruktúry NetFPGA repozitára

Vyššie uvedený repozitár tvorí pomerne zložitú adresárovú štruktúru, v ktorej je dôležité mať prehľad najmä v prípade, keď hľadáme konkrétne súbory. V tejto časti je uvedený popis adresárovej štruktúry, ktorý použijeme aj pre jednoduchšie uvádzanie a odkazovanie sa v ďalších častiach práce. Hlavný adresár pozostáva z podadresárov **contrib-projects/**, **docs/**, **lib/**, **projects/** a **tools/**.

- **contrib-projects/** - obsahuje všetky práve vyvíjané projekty, ktoré sú určené pre rôzne modely NetFPGA kariet. Slúžia v prípadoch, kedy chce používateľ niektorý z nich implementovať aj do svojho modelu karty.
- **docs/** - tu je uvedená dokumentácia k modelu našej karty. V súčasnosti sa tu nachádzajú referenčný manuál a schéma karty.
- **lib/** - v tomto adresári sú uložené knižnice používané v zdrojových súboroch testov, ktoré sú napísané v jazyku python a nachádzajú sa v adresári **python/**. Okrem nich tu máme IP jadrá - jednotlivé moduly / zdroje používané v rôznych projektov. Môžeme ich chápať ako všeobecné moduly, ktoré sú spoločne používané viacerými projektmi. HW popis (zdrojové súbory napísané v jednom z HDL jazykov) týchto modulov je vložený do **hw/** podadresára a ich ovládače sú umiestnené v **sw/**. Tieto moduly sú rozdelené do troch skupín: **contrib** - moduly používané v contrib projektoch, **std** - štandardné moduly používané v projektoch pre náš model karty a **xilinx** - moduly použité v projektoch, ktoré implementujú Microblaze soft-procesor.
- **tools/** - v tomto adresári nájdeme nástroje, ktoré nie sú špecifické pre konkrétny projekt, t.j. sú všeobecne používané. Patria sem viaceré skripty, z ktorých každý poskytuje určitú funkciu (adresár **scripts/**). V adresári **bin/** sa nachádza skript **nf_test.py**, ktorý poskytuje spoločné rozhranie pre spúšťanie testov jednotlivých projektov. V adresári **lib/** sú umiestnené zdrojové súbory spolu so spoločne používanou knižnicou *nf10_lib.so*, ktorá je používaná pri HW testoch. Táto knižnica zabezpečuje operácie súvisiace so zápisom a čítaním registrov NetFPGA karty.
- **projects/** - tento adresár obsahuje všetky projekty určené pre náš model NetFPGA karty. Každý z projektov je uvedený v samostatnom adresári. Pre nové projekty je vhodné vytvoriť nový adresár, kam vložíme všetky súbory špecifické pre tento projekt. V projektoch je použitá jednotná štruktúra. Každý z nich obsahuje nasledujúce

adresáre (uvedené sú len tie, ktoré sú pre nás podstatné): **bitfiles/**, **hw/**, **lib/**, **sw/** a **test/**.

- **bitfiles/** - obsahuje vygenerovaný konfiguračný súbor projektu, ktorý je následne vložený do FPGA obvodu karty.
- **hw/** - obsahuje súbory, ktoré súvisia s HW časťou návrhu – projekt realizovaný v Xilinx ISE prostredí. Nachádzajú sa tu aj zdrojové súbory modulov, ktoré sú špecifické pre tento projekt.
- **lib/** - knižnice daného projektu. Prístup ku spoločným modulom (uvedené vyššie) je riešený formou odkazu v tomto adresári.
- **sw/** - obsahuje softvérovú časť projektu. Súbory aplikácie, ktorá je vložená do pamäte NetFPGA karty a vykonávaná Microblaze soft-procesorom, sa nachádzajú v **embedded/**. Softvérové aplikácie, ktoré sú určené pre procesor pevného počítača a riadia správanie karty, sú spolu s ich zdrojovými súbormi umiestnené v **host/**.
- **test/** - obsahuje jednotlivé testy projektu uložené v samostatných adresároch napísané ako skripty v jazyku python. Pre testy a ich tvorbu je špecifikované jednotné rozhranie. Podrobnejšie informácie k vykonávaniu testov sú uvedené v nasledujúcej časti.

3.3.3 Testovacie skripty

Pre účely testovania existujú v NetFPGA repozitári skripty napísané v jazyku python, ktoré sú umiestnené v **<projekt>/test/**. Každý test sa nachádza v samostatnom adresári a pozostáva z jedného súboru – **run.py**. V adresári **connections** sú umiestnené súbory, ktoré popisujú fyzické zapojenie sieťových rozhraní pre jednotlivé testy. Každá dvojica navzájom prepojených portov je špecifikovaná na samostatnom riadku. Formát zapojenia je **nfX:ethY**, kde ľavé rozhranie je port NetFPGA karty a pravé port sieťovej karty pripojenej do PC. Ak daný NetFPGA port nie je prepojený s iným rozhraním, pravá časť zostane prázdna. Adresár **global** obsahuje skript **setup**, v ktorom je pre každé sieťové rozhranie definovaná IP adresa, ktorá bude použitá v testoch. Pre názvy testov je definovaný špecifický formát – **TYPE_MAJOR_MINOR**. TYPE informuje o povolenom spôsobe použitia testu a môže byť **sim** (len simulácia), **hw** (len hardvérový test) alebo **both** (**sim** aj **hw**). Pred spustením testov musíme vykonať prípravu prostredia, ktorá je popísaná v kapitole 5. Konkrétny test spustíme

pomocou skriptu **nf_test.py**. Pred samotným spustením testu musíme vypnúť službu AVAHI, ktorá generuje vlastnú sieťovú prevádzku na jednotlivých rozhraniach, čo by viedlo ku falošným výsledkom. Ak túto službu nepoužívame, môžeme ju zakázať, aby sa pri ďalšom nabehnutí systému znovu nespustila.

$ systemctl [stop|disable] avahi-daemon.service
$ sudo $NF_ROOT/tools/bin/nf_test.py [sim|hw] --major <MAJOR> --minor <MINOR>

Podrobnejšie informácie o testovacích skriptoch a ich vytvorení sú uvedené v sekcii **Hardware-Tests** v [**5**].

4 JEDNOTNÁ ARCHITEKTÚRA SYSTÉMOV

Funkcionalita HW a SW riešenia konkrétneho sieťového zariadenia je v jej samotnej podstate rovnaká (prevažne presne špecifikovaná konkrétnym RFC dokumentom). Dôvodom, prečo sa zaoberať návrhom systému a realizovať ho ako HW zariadenie aj napriek tomu, že už existuje jeho SW verzia, je prenosová rýchlosť (resp. rýchlosť spracovania údajov prenášaných sieťou), ktorú môžeme dosiahnuť.

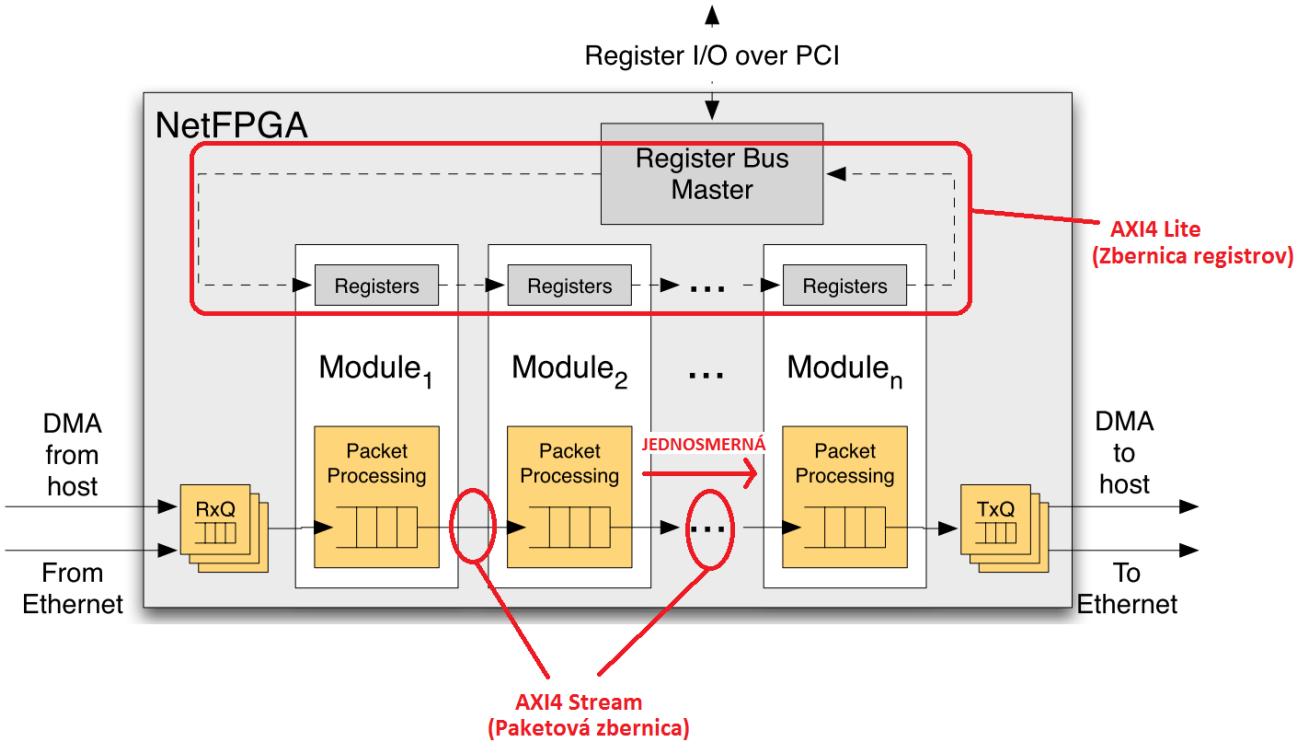
V projektoch, ktoré sú hlbšie rozobrané v praktickej časti tejto práce, sa zameriavame práve na analýzu a preskúšanie niektorých z dostupných sieťových NetFPGA implementácií vhodných pre náš typ NetFPGA karty. Kostra návrhu väčšiny sieťových uzlov je jasná - pakety prijaté na vstupných rozhraniach sú zlúčené do jedného frontu, následne spracované podľa požiadaviek systému a takto upravené sú odoslané cez výstupné rozhranie, príp. zahodené. Spracovanie bežných paketov (pakety bez explicitných požiadaviek, ako napr. ARP alebo IP options, či zložitejšie šifrovanie) je v riešeniach tejto práce vykonané hardvérovo priamo v FPGA obvode vo forme logických funkcií. Tým sa líšia od svojich SW alternatív, v ktorých je spracovanie špecifikované ako program vykonávaný mikroprocesorom vo forme postupnosti inštrukcií. Keďže hlavnou veličinou, na ktorú sa zameriavame, je rýchlosť, môžeme chápať FPGA riešenie ako HW akcelerátor daného SW riešenia (viditeľné najmä v smerovačoch, kde sú jednoznačne odlíšené funkcie FPGA obvodu od tých, ktoré ma za úlohu zabezpečiť mikroprocesor).

V tejto kapitole sa venujeme popisu jednotnej architektúry všetkých nami testovaných projektov. Súčasne je uvedený aj popis jednotlivých častí, z ktorých architektúra pozostáva vrátane ich rozhrania. Podrobnejšie informácie o architektúre projektov sú uvedené v [**12**].

4.1 POPIS ARCHITEKTÚRY

Každý NetFPGA systém, ktorému sa v práci venujeme, je založený na **modulárnej architektúre**, t.j. - návrh systému pozostáva zo skupiny modulov. Tieto moduly sú prepojené do jednej dlhej reťaze (obr. 6), cez ktorú prechádzajú pakety vo forme toku dát. Každý z modulov príjme na svojom vstupnom rozhraní daný paket, spracuje ho svojim špecifickým spôsobom (funkcionalita daného modulu) a takto upravený ho odošle cez svoj výstup ďalej na vstup susedného modulu. Susedné moduly komunikujú navzájom cez jednoduché **paketovo-založené jednosmerné FIFO fronty**. Okrem samotných údajov si moduly navzájom vymieňajú aj riadiace informácie, ako napr. stavový bit informujúci o zaplnení frontu daného

modulu smerujúci z modulu *i+1* do modulu *i*. Vďaka tomuto prístupu (použitie frontov medzi každou dvojicou susedných modulov) sme schopní ľubovoľne kombinovať poradie modulov a pridávať, resp. odoberať rôzne moduly tak, aby sme dosiahli požadovaný efekt a vytvorili tak návrh nových systémov. Vnútorná logika je tak pred používateľom skrytá. Jedinou podmienkou je dodržanie požadovaného rozhrania (vstupné a výstupné API) pre daný modul, ktorý chceme do návrhu pridať.



OBRÁZOK Č. 6: Modulárna architektúra systému

4.2 MODUL ARCHITEKTÚRY

Pod modulom rozumieme logický funkčný blok. Môžeme si ho predstaviť ako samostatný logický obvod, ktorý príjme vstupné signály, vykoná nad nimi rôzne operácie a výsledok svojej činnosti odošle vo forme výstupných signálov. Modul môžeme popísať rôznym spôsobom. V práci používame pre popis modulu a jeho správania HDL jazyk – Verilog. Následne teda každý z návrhov, ktorým sa v práci venujeme, pozostáva prevažne z modulov, ktoré sú popísané vo forme zdrojového kódu napísaného vo verilogu. V praxi sa ukázalo, že niektoré funkcionality (kľúčové bloky, napr. konverzia signálov z rôznych periférií) sú potrebné v takmer každom systéme, a preto boli vytvorené špecifické moduly, ktoré tieto funkcie zabezpečujú, nazývajú sa **IP (Intellectual property) jadrá**. Z pohľadu návrhára, ktorý sa rozhodne použiť dané IP jadro vo svojom návrhu, je tento prvok čiernou skrinkou, ktorej vnútornú logiku nepozná. Stačí mu len poznať rozhranie daného jadra, aby ho

mohol správne začleniť do svojho návrhu a vďaka tomu nemusí venovať čas tvorbe nového modulu s rovnakou funkcionalitou. Namiesto toho môže riešiť dôležitejšie otázky návrhu. IP jadrá sú dnes považované za služby, z ktorých niektoré sú poskytované zadarmo a iné je potrebné kúpiť.

V prvej kapitole sme uviedli popis hard IP jadier. Okrem nich existujú aj firm a soft IP jadrá. Od hard IP jadier sa líšia tým, ako sú v systéme realizované. Obidva tieto typy sú realizované cez CLB v FPGA obvode ako hociktorý iný používateľský modul (rozdiel oproti hard IP jadru, ktoré je realizované ako samostatný prvok). Firm sa od soft líši len tým, že kvôli optimalizácii výkonu môže byť umiestnený len na presne definovanom mieste FPGA obvodu, zatiaľ čo umiestnenie soft IP jadra nie je obmedzené. Soft IP jadro je v podstate len bežný používateľský modul, ktorého implementácia je pred používateľom skrytá. V ďalšej časti sa venujeme už len modulom, ktoré používame v našich systémoch.

4.3 ROZHRANIE MODULU

Rozhranie modulu tvoria vstupné a výstupné signály, cez ktoré modul komunikuje s okolím. Pre vzájomnú komunikáciu používajú moduly dve navzájom nezávislé komunikačné linky: **packet bus** a **register bus**, ktoré môžeme chápať ako dátovú a riadiacu/stavovú zbernicu.

Paketová zbernica zabezpečuje prenos paketov z jedného modulu do ďalšieho prostredníctvom synchrónneho frontu (uvedený vyššie) s frekvenciou 125 MHz. **Zbernica registrov** je samostatnou linkou, a teda nespotrebuje prenosové pásmo dátovej zbernice. Táto komunikačná linka umožňuje prenos údajov oboma smermi s použitou frekvenciou 100 MHz. Keďže slúži na prenos riadiacich a stavových informácií, nevyžaduje takú veľkú šírku pásma.

Vyššie uvedené komunikačné linky sú v systémoch určených pre NetFPGA kartu implementované cez komunikačný štandard **AXI4** pôvodne uvoľnený ARM výrobcom pod označením **AMBA4** a neskôr upravený Xilinx výrobcom pre potreby FPGA obvodov. Tento štandard definuje rôzne formy komunikačného protokolu, ktoré sa do veľkej miery líšia. Príkladom sú implementácie uvedených zberníc – **AXI4 Lite** rozhranie a **AXI4 Streaming** rozhranie.

4.3.1 Zbernica registrov

Táto zbernica poskytuje jednotné rozhranie, cez ktoré umožňuje softvérovej aplikácii vykonávanej vstavaným mikroprocesorom karty alebo procesorom hosťujúceho počítača

pristupovať a meniť hodnoty registrov. Tie sú definované v jednotlivých moduloch a špecifikujú ich vlastnosti – ovplyvňujú ich správanie (tabuľky, registre) alebo popisujú ich aktuálny stav (počítadlá). Takto je softvér schopný kontrolovať a riadiť správanie hardvérového systému. Mapovaním interných hardvérových registrov do pamäte sa tie javia pre softvér ako vstupno-výstupné registre, a teda sú dostupné cez vhodné **ioctl** volania. Tento mechanizmus riadi vhodný ovládač, ktorý je vo forme **kernel object (.ko)** súboru vložený do operačného systému. Ďalšie informácie k ovládaču sú uvedené v kapitole 4. Zbernica registrov spája jednotlivé moduly návrhu do spoločného kruhu. Komunikáciu zbernice riadi jeden z modulov, ktorý je zvolený ako **Master**. Jeho úlohou je iniciovať požiadavky prichádzajúce z PCIe zbernice (softvérová aplikácia) a odpovedať na ne. Vďaka kruhovej topológii môže modul pristúpiť k registrom ľubovoľného iného modulu. Táto zbernica poskytuje dva typy požiadaviek – **register READ** a **register WRITE**. Žiadosť postupne prechádza cez jednotlivé moduly. Daný modul môže odpovedať na požiadavku alebo ju poslať nezmenenú ďalej. V prvom z prípadov vykoná modul požadované zmeny a ak je potrebné, vloží požadovanú hodnotu na zbernicu. Cez **potvrdzovací signál ACK** informuje o vykonaní zmien. Tie sú následne cez PCIe zbernicu prenesené do hosťujúceho počítača.

V systéme je zbernica registrov implementovaná cez **AXI4 Lite** rozhranie odvodené od špecifikácie AMBA4 zbernice. Tento protokol je jednoduché read/write rozhranie s 32-bitovou adresou a 32-bitovými údajmi implementované cez päť FIFO kanálov: **read data**, **read address**, **write data**, **write address** a **write response**. Pre účely vzájomného prepojenia všetkých modulov, z ktorých každý obsahuje AXI4 Lite rozhranie, slúži samostatný špecifický modul s názvom AXI4 Lite Interconnect. Ten umiestni moduly do jedného spoločného adresného priestoru. Moduly prislúchajúce k PCIe zbernici a Microblaze procesoru vystupujú v tomto prostredí ako **Master**. Ostatné moduly vystupujú ako **Slave**. Adresy je možné manuálne zvoliť alebo nechať automaticky vygenerovať cez EDK, ktorý vytvorí celý adresný priestor pre všetky moduly pripojené do tejto siete. Pomenovanie signálov, ktoré prislúchajú k tomuto štandardu, je pre prehľadnosť zachované vo všetkých projektoch a má tvar - [**M|S]_AXI_<meno_signálu**>. Podrobnejší popis rozhrania a jeho signálov je uvedený v sekcii **Standard-IP-Interfaces** v [**5**]. Špecifikácia AXI4 Lite rozhrania je uvedená v [**13**].

4.3.2 Paketová zbernica

Táto zbernica spomenutá vyššie zabezpečuje prenos údajov medzi dvojicou susedných modulov. Údaje sú posielané vo forme **paketov**, tak aby nedošlo ku ich prekrytiu, t.j. v danom okamihu je prenesený práve jeden paket. Kvôli jednoduchosti nie sú v rámci jedného paketu žiadnym spôsobom oddelené hlavičky od samotných dát, keďže táto funkcionalita nie je potrebná. To ale nebráni jednotlivým modulom urobiť tak interne. Vďaka jednoduchosti tohto rozhrania je možné dosiahnuť požadovaného výkonu. Paketová zbernica pozostáva z dvoch samostatných interných zberníc – **dátovej** a **riadiacej**. Cez **READY** signál informuje modul *i+1* svojho suseda, že je pripravený prijať ďalšie údaje. Vďaka tomu nemôže dôjsť k zahlteniu kanála a tým je zabezpečené riadenie toku medzi modulmi. Ako reakciu na takto prijatý signál nastaví modul *i* riadiacu aj dátovú zbernicu a svojho suseda informuje o prenose cez **WRITE** signál. Pri prechode paketu systémom dochádza k jeho spracovaniu. Pri tomto procese vznikajú čiastočne výsledky, ktoré jeden modul vytvorí a iný modul neskôr následne použije. Tieto údaje sú vkladané do samostatného na to určeného signálu.

V systéme je paketová zbernica implementovaná cez **AXI4 Streaming** rozhranie, ktoré je rovnako ako AXI4 Lite odvodené od špecifikácie AMBA4 zbernice. Medzi každou dvojicou susedných modulov je vytvorené a špecifikované samostatné rozhranie, v ktorom jeden modul vystupuje ako **Master** a druhý ako **Slave**. Rozhranie pozostáva z viacerých signálov. **TDATA** s 256-bitovou šírkou obsahuje údaje paketu. **TREADY**, **TVALID** a **TLAST** zabezpečujú funkcie riadenia kanálu. **TUSER** so 128-bitovou šírkou obsahuje meta-data - používateľom doplnené údaje súvisiace s daným paketom, ktoré je možné neskôr použiť v ďalších moduloch. Tento signál obsahuje **preddefinované polia**: TUSER[15:0] – **dĺžka paketu** v bajtoch, TUSER[23:16] – **zdrojový port**, cez ktorý bol paket prijatý a TUSER[32:17] - **cieľový port**, cez ktorý má byť paket odoslaný. Porty uvedené v týchto poliach používajú **one-hot** kódovanie – pre každé sieťové rozhranie (štyri fyzické ethernetové sieťové porty a štyri virtuálne DMA porty) slúži jeden bit. Nepárne bity sú pridelené fyzickým portom a párne bity virtuálnym DMA portom. Zvyšnú časť tvoria **používateľom definované polia** rozdelené do šiestich 16-bitových slotov. Slot je rozdelený do dvoch častí: **typ** a **hodnota**, ktoré si užívateľ volí podľa seba. Podrobnejší popis AXI4 Streaming rozhrania je uvedený v [**4**] a [**13**].

4.4 MODULY VSTUPNÉHO A VÝSTUPNÉHO ROZHRANIA SYSTÉMU

Aby sme boli schopní priamo v systéme jednotne pracovať so signálmi prichádzajúcimi z rôznych periférií, potrebujeme špecifické moduly - konvertory. Úlohou týchto modulov je prevod vstupných signálov rôzneho typu (prijaté z periférií s rôznou charakteristikou) na jednotné FIFO rozhranie, ktoré je použité vo vnútri systému (viď. vyššie). Tieto moduly rovnako tak zabezpečujú spätný prevod signálov FIFO rozhrania na výstupné signály danej periférie. Tým poskytujú jednotné vstupné/výstupné rozhranie systému. Vďaka tomuto prístupu nemusíme ako programátori rozumieť danému typu I/O portov (danej periférii), aby sme ich pripojili do návrhu, ale jednoducho použijeme jeden z modulov.

V prípade ethernetových portov slúži k tomuto účelu **nf10_1G_interface** modul, kde pre každý fyzický port je vytvorená jedna inštancia tohto modulu. Ten kombinuje **Xilinx XAUI** a **1G MAC Xilinx** IP jadrá, ktoré sú pripojené k **AXI-Stream adaptéru**. Pakety prichádzajúce z externých 1G PHY portov sú cez XAUI rozhranie prevedené na **RGMII** (Reduced Gigabit Media Independent Interface) signály. Transformáciu zabezpečuje Xilinx XAUI jadro. Tieto signály následne číta Xilinx 1G MAC jadro a prevedie ich na AXI4-Stream, ktorý je používaný vo vnútri systému. Rovnaký proces len v opačnom poradí je vykonaný pre odosielajúce pakety.

Pakety prichádzajúce z PCIe zbernice vstupujú priamo do **DMA** modulu. Ten pozostáva zo **Xilinx PCIe** jadra a **AXI4-Lite** modulu. PCIe jadro poskytuje AXI4-Stream rozhranie pre komunikáciu s ostatnými modulmi v systéme.

IP jadrá spomenuté v tejto časti sú podrobne popísané v [**14**] a [**15**].

4.5 MODULY TVORIACE USER DATAPATH

Vyššie uvedené moduly sú pripojené k vnútornej časti systému – tzv. **user datapath**. Tú tvoria moduly, v ktorých je implementovaná prevažne používateľská logika. Výstupy zo všetkých nf10_1G_interface modulov a z DMA modulu sú pripojené cez FIFO fronty do **Input_arbiter** modulu. Jeho úlohou je zbieranie paketov zo všetkých vstupných rozhraní. Postupne prechádza neprázdnymi frontami a zakaždým vyberie jeden paket, ktorý je odoslaný ďalej do susedného modulu. Ako obslužný algoritmus je aktuálne použitý **round-robin**. Odoslaný paket je následne spracovaný postupnosťou modulov, ktoré určia jeho cieľový port, cez ktorý bude odoslaný von z NetFPGA karty. Potom paket smeruje do

Nf10_bram_output_queues modulu, ktorý ho podľa **cieľového portu** v TUSER vloží do jedného z frontov pridelených jednotlivým interface modulom vrátane DMA modulu. Ak je daný front zaplnený, resp. takmer zaplnený, paket je zahodený. Úlohou programátora je definovať logiku modulov, ktoré sú umiestnené medzi modulmi Input_arbiter a NF10_bram_output_queues. V nami analyzovaných projektoch je používateľská logika skrytá v module **output_port_lookup**, ktorý je umiestnený medzi vyššie uvedenými modulmi.

4.6 OVLÁDAČ PRE PCIE ZBERNICU NETFPGA KARTY

Aby bol operačný systém hostiteľského počítača (Linux Fedora), ku ktorému je karta pripojená, schopný pracovať s danou kartou cez PCIe zbernicu, potrebuje ovládač, ktorý riadi túto komunikáciu. Upozorňujeme, že pre operačný systém Windows v čase písania tejto práce ešte neexistuje ovládač PCIe zbernice pre NetFPGA-1G-CML model karty, dostupný je len linuxový ovládač vo forme modulu jadra.

Zdrojový kód tohto ovládača sa nachádza v NetFPGA repozitári ako súčasť referenčného NIC projektu **(reference_nic_nf1_cml/sw/host/driver)**. Tieto súbory použijeme pre kompiláciu modulu. Keďže sa jedná o kompiláciu modulu jadra, potrebujeme predtým ešte doinštalovať dodatočné balíky, ktoré obsahujú všetky potrebné hlavičkové a knižničné súbory. Sú to **kernel-headers.x86_64** a **kernel-devel.x86_64** balíky, pre ktorých inštaláciu môžeme použiť **yum** nástroj. Pri prechode na linuxové jadro 3.17 došlo ku zmene rozhrania niektorých systémových funkcií, a preto musíme pred kompiláciou upraviť jeden zo zdrojových súborov. Problémom je systémové makro **alloc_netdev**, do ktorého bol pridaný štvrtý parameter. V súbore **nf10iface.c** na riadku 160 pridáme vo volaní uvedeného makra medzi druhý a tretí parameter ešte **NET_NAME_UNKNOWN**. Po vykonaní tejto úpravy skompilujeme modul použitím nástroja **make**. Ako výsledok získame súbor **nf10.ko**, ktorý cez príkaz **insmod**, resp. **modprobe** môžeme následne nainštalovať do jadra systému. Inštaláciu ovládača musíme vykonať po každej rekonfigurácii karty ako aj po každom nabehnutí operačného systému.

Nainštalovaním tohto modulu je sprístupnený súbor **/dev/nf10**, cez ktorý môžeme v softvérovej aplikácii pristupovať k registrom NetFPGA karty (AXI4-Lite), a tak čiastočne riadiť jej správanie. Pre prístup do adresného priestoru AXI4-Lite FPGA obvodu slúžia **ioctl** volania vykonané nad vyššie uvedeným súborom. Odporúčame preštudovať použitie ioctl

volaní v už existujúcich softvérových aplikáciách, ktoré sú súčasťou jednotlivých projektov v NetFPGA repozitári.

Okrem toho sú sprístupnené štyri nové virtuálne sieťové rozhrania – **nf0**, **nf1**, **nf2** a **nf3.** Rámce, ktoré sú z NetFPGA karty odoslané cez jeden z virtuálnych DMA portov PCIe zbernice, sú vďaka ovládaču transformované na sieťovú prevádzku prijatú na týchto virtuálnych rozhraniach. Vďaka tomu sa celý proces komunikácie medzi NetFPGA kartou a počítačom javí, ako keby boli pripojené cez počítačovú sieť. Rovnako tak môžeme tento princíp použiť pre rámce vytvorené v počítači, ktoré odošleme cez jedno z virtuálnych rozhraní priamo do NetFPGA karty.

Vzhľadom na vyššie uvedené skutočnosti sme schopní v počítači sledovať prostredníctvom virtuálnych rozhraní len prevádzku DMA portov NetFPGA karty. Rámce, ktoré sú prijaté alebo odoslané cez fyzické porty NetFPGA karty, nie sú žiadnym spôsobom viditeľné v počítači. Jedinou možnosťou, ako sledovať takúto prevádzku je pripojenie jedného z fyzických portov počítača na fyzický port NetFPGA karty. Tento prístup sme zvolili aj my pri testovaní jednotlivých projektov. V nasledujúcej časti sú uvedené niektoré príklady, ako použiť ovládač pre zmenu hodnoty registrov v NetFPGA karte.

Aby sme mohli pristupovať k jednotlivým registrom NetFPGA karty, potrebujeme ich adresy. K tomuto účelu slúži hlavičkový súbor *reg_defines.h,* v ktorom sú uvedené adresy všetkých registrov daného projektu. Tento súbor získame použitím nasledujúcich príkazov.

$ cd $NF_DESIGN_DIR/hw
$ make regs

4.6.1 Modifikácia registrov z PC

Pre manipuláciu s registrami FPGA obvodu NetFPGA karty je v repozitári vytvorená spoločná knižnica, ktorá poskytuje funkcie pre prístup k týmto registrom cez ich adresu (adresy registrov definované v *reg_defines.h*). Vďaka nim nemusíme priamo používať *ioctl* volania. Práca s registrami je tak z pohľadu programátora vnímaná len ako volanie jednej z funkcií *reg_read* a *reg_write*. Knižnica sa nachádza v **$NF_ROOT/tools/lib/**. V tomto adresári sa nachádza aj zdrojový súbor, ktorý môžeme podľa potrieb zmeniť a knižnicu nanovo skompilovať. Táto knižnica je používaná aj pri HW testoch, ktoré sú napísané v jazyku python. V nasledujúcej kapitole sa podrobnejšie venujeme realizácii projektu a jeho vloženiu do karty. Celý proces podrobnejšie rozoberieme na konkrétnom príklade.

5 REALIZÁCIA PROJEKTOV PRE NETFPGA

Doteraz sme sa venovali prevažne teoretickým aspektom, ktoré sú pre nás nepodstatné, ak ich nevieme použiť v praxi. Ich aplikácia do praxe spočíva v použití už vytvorených projektov a v ich úprave ako aj vo vytvorení vlastných projektov. V tejto kapitole popíšeme postup vytvorenia projektu na jednoduchom príklade.

5.1 TVORBA NOVÉHO PROJEKTU

V tejto časti stručne uvádzame spôsob, ako vytvoriť nový projekt pre NetFPGA kartu vrátane všetkých potrebných krokov, ako v ňom vykonať zmeny ako napríklad tým, že pridáme nami vytvorený nový modul a ako následne z takto upraveného projektu vytvoríme konfiguračný súbor a vložíme ho do obvodu. Proces tvorby projektu je možné vykonať buď v grafickom prostredí nástroja XPS alebo použitím nástroja **make** v príkazovom riadku. Druhý z týchto spôsobov je uvedený až nakoniec.

5.1.1 Príprava prostredia

Pred začatím používania nástrojov je potrebné vykonať kroky, ktoré súvisia s nastaveniami prostredia operačného systému. Prvým krokom je aktualizácia **PATH** premennej prostredia – pridanie cesty ku binárnym súborom každého z nástrojov. Pre tento účel slúži skript poskytnutý priamo od Xilinx, ktorý je súčasťou inštalácie ISE - **<cesta_ku_Xilinx_ISE>/14.7/ISE_DS/settings64.sh**.

$ source <cesta_k_skriptu>/settings64.sh

Druhým krokom je editovanie **bashrc_addon_NetFPGA_10G** súboru v koreňovom adresári repozitára, kde je potrebné zmeniť hodnotu premennej prostredia **NF_ROOT** na cestu k danému repozitáru a **NF_DESIGN_DIR** na cestu k projektu, na ktorom práve pracujeme.

$ source <cesta_k_repozitaru>/bashrc_addon_NetFPGA_10G

Vykonaním vyššie uvedeného príkazu nastavíme systémové premenné, ktoré sú používané vo viacerých skriptoch NetFPGA repozitára. Tieto príkazy je potrebné vykonať pri každom spustení počítača/systému ako aj pri každej zmene projektu. To môžeme zabezpečiť ich pridaním do **.bash_profile**, ktorý sa nachádza v domovskom adresári používateľa.

Vytvorením lokálnej kópie NetFPGA repozitára uvedeného v kapitole 3 máme k dispozícii takmer všetky potrebné súbory. Kvôli minimalizácii jeho veľkosti sa v ňom

nachádzajú len zdrojové súbory a rôzne skripty. Aby sme získali jednotlivé jadrá, ktoré sú použité v projektoch ako aj knižnice pre ich testovanie, musíme vykonať nasledujúce príkazy.

$ cd $NF_ROOT
$ make cml_cores
$ make hwtestlib

Tieto príkazy je potrebné vykonať len prvýkrát ako súčasť prípravy prostredia pre nasledujúci vývoj a testovanie. Podrobnejšie informácie sú uvedené v [**5**].

5.1.2 Vytvorenie nového projektu

Pre vývoj systémov pre NetFPGA kartu používame XPS nástroj, ktorý je určený pre vstavané systémy obsahujúce mikroprocesor (v našom prípade Microblaze soft-procesor). Jednou z možností, ako začať, je vytvorenie nového prázdneho projektu. Tento krok je v XPS jednoduchý a pre používateľa intuitívny. Takto začíname úplne od začiatku a do projektu pridávame len tie moduly, ktoré potrebujeme.

Vzhľadom na zložitosť projektov, ktoré analyzujeme, je však lepšie pokračovať tam, kde iní skončili. Tento postup je odporúčaný NetFPGA komunitou, a preto sme ho zvolili aj my. Prvým krokom je vytvorenie nového adresára pre projekt v **projects** a skopírovanie jedného z referenčných projektov umiestnených v repozitári do tohto adresára. Projekt obsahuje súbory, ktoré popisujú jeho nastavenia v XPS:

- **hw/system.xmp** – hlavný súbor obsahujúci všeobecné nastavenia projektu,
- **hw/system.mhs** – definovanie vstavaného systému, architektúry rozhrania, periférií, procesora a ich vzájomného prepojenia – štruktúra projektu,
- **hw/data/system.ucf** – definovanie používateľských obmedzení – pridelenie I/O pinov systému a nastavenie časových požiadaviek.

Do tohto projektu môžeme následne pridávať nové moduly alebo meniť už existujúce. XPS nástroj sa automaticky postará o aktualizáciu týchto súborov.

5.1.3 Vytvorenie modulu a jeho vloženie do návrhu

Problémom vytvorenia nového modulu je definovanie jeho rozhrania tak, aby spĺňalo požiadavky komunikačného protokolu použitého v našom systéme (viď. vyššie) a aby modul mohol byť začlenený do projektu. Z popisu nami použitého rozhrania (AXI4-Lite a AXI4-

Stream) je zrejmé, že jeho definícia nie je jednoduchá (obsahuje veľa vstupných a výstupných portov rôznej šírky). Práve tu využijeme výhody, ktoré nám poskytuje XPS nástroj. Pomocou neho môžeme priamo v grafickom prostredí vo forme zaškrtávacieho okna vytvoriť šablónu nového modulu, ktorý už bude obsahovať požadované rozhranie. Inou možnosťou pre vytvorenie šablóny modulu je použitie skriptu, ktorý vytvorí modul zhodný so štruktúrou modulov jednotlivých projektov – **nf10_coregen.py**. V našom prípade odporúčame túto možnosť.

$ python <tools>/scripts/nf10_coregen.py --name <meno_noveho_modulu> --path \ $NF_DESIGN_DIR/hw/pcores

Výsledkom je skupina súborov, ktoré tvoria modul a sú umiestnené v samostatnom adresári (definovaný cez *--path* parameter vo vyššie uvedenom príkaze). Pre nás je dôležitý zdrojový súbor s koncovkou **.v** (Verilog). Našou úlohou je dopísať do tohto súboru logiku modulu – jeho správanie. Túto operáciu nie je možné vykonať priamo v XPS. Preto použijeme ISE Project Navigator, kde pomocou ISE Editora pridáme do zdrojového súboru potrebné bloky. Výhodou ISE je knižnica rôznych bežne používaných prvkov napísaných vo verilogu, ktoré môžeme do modulu vložiť. Okrem týchto vygenerovaných súborov je potrebné vytvoriť pre daný modul samostatný ovládač (viď. príklad).

Ak sme tento krok urobili, môžeme takto vytvorený a funkčný modul vložiť do projektu cez XPS. Po otvorení projektu sa nám zobrazí v IP katalógu náš nový modul, ktorý zvolíme a pridáme. Súčasťou tohto kroku je vyplnenie parametrov modulu ako napr. šírka komunikačnej zbernice. Vďaka XPS je modul následne priamo vložený do projektu a ak povolíme, aj priamo napojený na inštanciu Microblaze modulu. Posledným krokom je nastavenie zoznamu vstupných a výstupných portov modulu a adresného priestoru jeho AXI4 Lite rozhrania. Tieto operácie vykonáme v nasledujúcich tabuľkách. Cez **Bus Interfaces** tabuľku zvolíme pre každú zbernicu, ku čomu bude pripojená. Takto vložíme modul na správnu pozíciu v systéme – zvolíme jeho susedné moduly, s ktorými bude prepojený. V ďalšej tabuľke **Ports** pridelíme jednotlivým portom modulu, ako sú porty RESET a CLK prislúchajúce signály v systéme. V poslednej z tabuliek **Addresses** skontrolujeme a podľa potrieb zmeníme adresný priestor (AXI4-Lite) pridelený tomuto modulu.

5.1.4 Vytvorenie konfiguračného súboru

Ak sme spokojní s daným návrhom systému, môžeme prejsť až ku vytvoreniu konfiguračného súboru. Vďaka XPS sa nemusíme starať o všetky operácie, ktoré je nutné

vykonať, lebo XPS ich urobí automaticky. V hlavnom menu zvolíme **Hardware → Export Design**. Ak proces prebehne úspešne, výsledkom je bitstream súbor uložený v **sw/embedded/SDK_Workspace/hw/system.bit**. Použitím SDK nástrojov skompilujeme SDK projekt. Tak získame výsledný konfiguračný súbor **sw/embedded/result/download.bit,** ktorý môžeme následne vložiť do FPGA obvodu cez iMPACT.

5.2 PRÍKLAD REALIZÁCIE PROJEKTU

V tejto časti na príklade ukážeme vytvorenie nového projektu, pričom predpokladáme, že príprava prostredia uvedená vyššie už bola úspešne vykonaná a projekt, z ktorého vychádzame, je funkčný. Štruktúra repozitára je uvedená v kapitole 3.

Našim vzorovým príkladom je realizácia ethernetového LAN prepínača s jednoduchou podporou prístupových VLAN (Virtual LAN) sietí a ovplyvnenie prepínania, kedy okrem cieľovej MAC adresy sa bude analyzovať aj VLAN zdrojového a cieľového portu. Vzhľadom na zadanie použijeme ako východiskový projekt referenčný prepínač – **reference_switch_nf1_cml**, ktorý upravíme tak, aby poskytoval aj vyššie uvedené funkcie. V našom príklade zmeníme **output_port_lookup** modul tohto projektu tým, že do jeho zdrojových súborov pridáme funkcionalitu VLAN. Tento modul odoberieme z pôvodného návrhu, vykonáme jeho zmeny a späť pridáme už ako nový lokálny modul projektu. Zaujímavosťou je pridanie nových registrov do modulu, ktoré budeme schopní meniť z PC, čím riadime aktuálne správanie karty.

5.2.1 Príprava projektu

Pre nový projekt vytvoríme samostatný adresár. Takto dosiahneme prehľadnosť a zmenšíme riziko, že si rôznymi pokusmi poškodíme niektorý zo súborov repozitára. Do tohto adresára skopírujeme obsah východiskového projektu. Pre vykonanie týchto krokov použijeme príkazy:

$ cd $NF_ROOT/projects
$ cp –r reference_switch_nf1_cml switch_vlan
$ cd switch_vlan
$ make clean

Rovnako tak skopírujeme obsah modulu, ktorý ideme meniť. V našom prípade to je **nf10_switch_output_port_lookup_v1_10_a**. Keďže tento modul je spoločne používaný viacerými projektmi, jeho súbory sú umiestnené mimo adresár projektu. Nachádzajú sa v

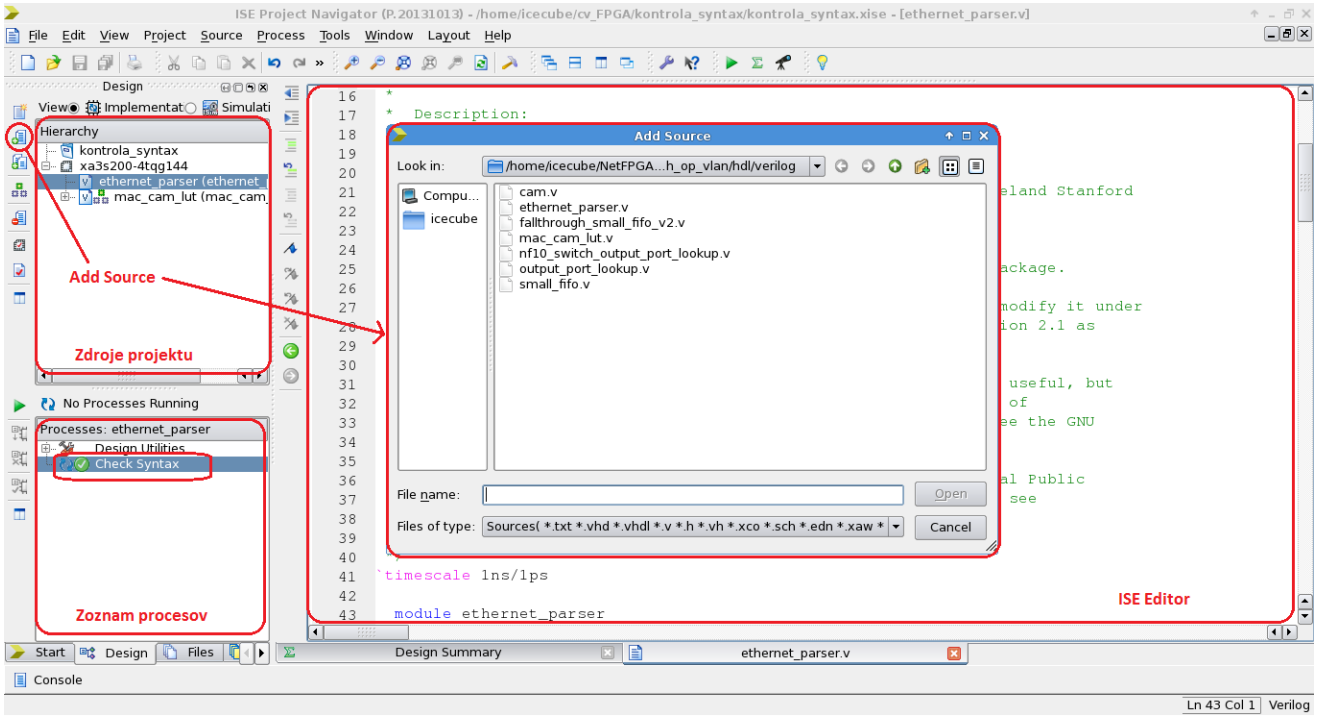
$NF_ROOT/lib/. Obsah tohto modulu skopírujeme na vhodné miesto v projekte. Modul sa tak stane lokálnym.

$ cd $NF_ROOT/lib/hw/std/pcores/
*$ cp –r nf10_switch_output_port_lookup_v1_10_a *
$NF_ROOT/projects/switch_vlan/hw/pcores/nf10_switch_op_vlan_v1_10_a

5.2.2 Modifikácia zdrojových súborov

Ak už máme vytvorené kópie, môžeme pokračovať ich modifikáciou. Najprv upravíme zdrojové súbory modulu napísané vo verilogu, ktoré sú umiestnené v **nf10_switch_op_vlan_v1_10_a/hdl/verilog/** (adresár vytvorený v predchádzajúcom kroku). Tieto zmeny je vhodné vykonať v ISE editore, do ktorého sa dostaneme cez ISE Project Navigator. Vďaka priamej podpore jazyka verilog môžeme využiť výhody tohto nástroja, napr. automatická kontrola syntaxe súborov. Táto kontrola je vhodná najmä, ak nemáme skúsenosti s HDL jazykmi. Aby sme túto funkcionalitu mohli použiť, musí byť daný súbor súčasťou nejakého projektu ako jeho zdroj. Preto je vhodné vytvoriť si v ISE Project Navigator nový prázdny projekt len pre potreby kontroly. Následne kliknutím na ikonu **Add Source** pridáme do projektu každý zo zdrojových súborov, ktoré chceme skontrolovať. Kontrolu spustíme tak, že označíme konkrétny súbor v zozname projektu a v zozname procesov, ktorý sa nachádza hneď pod nim, zvolíme **Check Syntax**. Tieto komponenty sú zvýraznené na obr. 7.

Aby sme mohli priamo z PC špecifikovať VLAN každého portu a následne dynamicky túto hodnotu meniť cez softvérový nástroj, použijeme pre každý fyzický port jeden 32-bitový register z adresného priestoru (AXI4-Lite) tohto modulu. V jeho nižších 12 bitoch bude uložená aktuálna VLAN, do ktorej daný port patrí. Predvolene je táto hodnota rovná 0.



OBRÁZOK Č. 7: ISE Project Navigator

V časti, kde sú z rámca vyčítané jeho zdrojová, cieľová MAC adresa a zdrojový port, z ktorého bol rámec prijatý, pridáme dodatočnú logiku. Na základe zdrojového portu (8-bitový vektor) získaného z TUSER (viď. predchádzajúca kapitola, popis paketovej zbernice) identifikujeme zdrojovú VLAN – VLAN zdrojového portu. Tú použijeme pre špecifikovanie **aktívnych portov** – porty patriace do spoločnej VLAN ako zdrojový port. Aktívne porty sú kódované vo forme 8-bitového vektora podobne ako zdrojový port, kde log. 1 majú aktívne porty a log. 0 neaktívne porty (porty, ktoré sú v inej VLAN ako zdrojový port). Vektor aktívnych portov použijeme nakoniec ako masku pri špecifikovaní cieľového portu, aby sme zabránili odoslaniu rámca portami, ktoré nepatria do VLAN zdrojového portu.

Táto implementácia slúži len ako ukážkový príklad. Zmenené zdrojové súbory modulu sa nachádzajú na priloženom CD disku ako súčasť prílohy A.

5.2.3 Modifikácia HW nastavení modulu

Keďže meníme parametre modulu, je potrebné upraviť niektoré súbory, ktoré sa nachádzajú v adresári **nf10_switch_op_vlan_v1_10_a/data/**. Najprv ich premenujeme tak, aby boli zhodné s menom modulu.

$ mv nf10_switch_output_port_lookup_v2_1_0.bbd nf10_switch_op_vlan_v2_1_0.bbd
$ mv nf10_switch_output_port_lookup_v2_1_0.pao nf10_switch_op_vlan_v2_1_0.pao

$ mv nf10_switch_output_port_lookup_v2_1_0.mpd nf10_switch_op_vlan_v2_1_0.mpd

V **nf10_switch_output_port_lookup_v2_1_0.pao** sú okrem iného uvedené všetky zdrojové súbory, ktoré tvoria modul. Keďže sme zmenili názov adresára modulu, je potrebné vykonať zmeny aj v tomto súbore a nahradiť všetky výskyty pôvodného mena novým. V **nf10_switch_output_port_lookup_v2_1_0.mpd** musíme upraviť názov modulu (za **BEGIN**) na **nf10_switch_op_vlan**. Súčasťou zmien je pridanie nových registrov.

Formát zápisu:

PARAMETER <meno>_OFFSET=<offset>, DT=<udajovy_typ>, BUS=S_AXI, ASSIGNMENT=CONSTANT, TYPE=NON_HDL

Príklad:

PARAMETER VLAN_ID_IF0_OFFSET=0x4, DT=std_logic_vector, BUS=S_AXI, ASSIGNMENT=CONSTANT, TYPE=NON_HDL

5.2.4 Modifikácia SW nastavení modulu – ovládače

Posledný krok, ktorý musíme vykonať v rámci tvorby nášho nového modulu je vytvorenie ovládača. Ovládače modulov sa nachádzajú v **$NF_ROOT/lib/sw/std/drivers/**. V tomto prípade použijeme pôvodný ovládač, ktorý skopírujeme do samostatného adresára a upravíme.

$ cd $NF_ROOT/lib/sw/std/drivers
$ cp –r nf10_switch_v1_00_a nf10_switch_vlan_v1_10_a
$ cd nf10_switch_vlan_v1_10_a/data
$ mv nf10_switch_v2_1_0.mdd nf10_switch_vlan_v2_1_0.mdd
$ mv nf10_switch_v2_1_0.tcl nf10_switch_vlan_v2_1_0.tcl

V **nf10_switch_vlan_v2_1_0.mdd** zmeníme riadok **supported peripherals** tak, aby zahŕňal aj náš nový modul. V **nf10_switch_vlan_v2_1_0.tcl** pridáme registre, pričom použijeme rovnaké názvy ako v **nf10_switch_output_port_lookup_v2_1_0.mpd** (vrátane _OFFSET). Tieto zmeny zabezpečia, že pridané registre budú súčasťou neskôr vygenerovaného *reg_defines.h* súboru (viď. kapitola 4).

5.2.5 Úprava návrhu v XPS

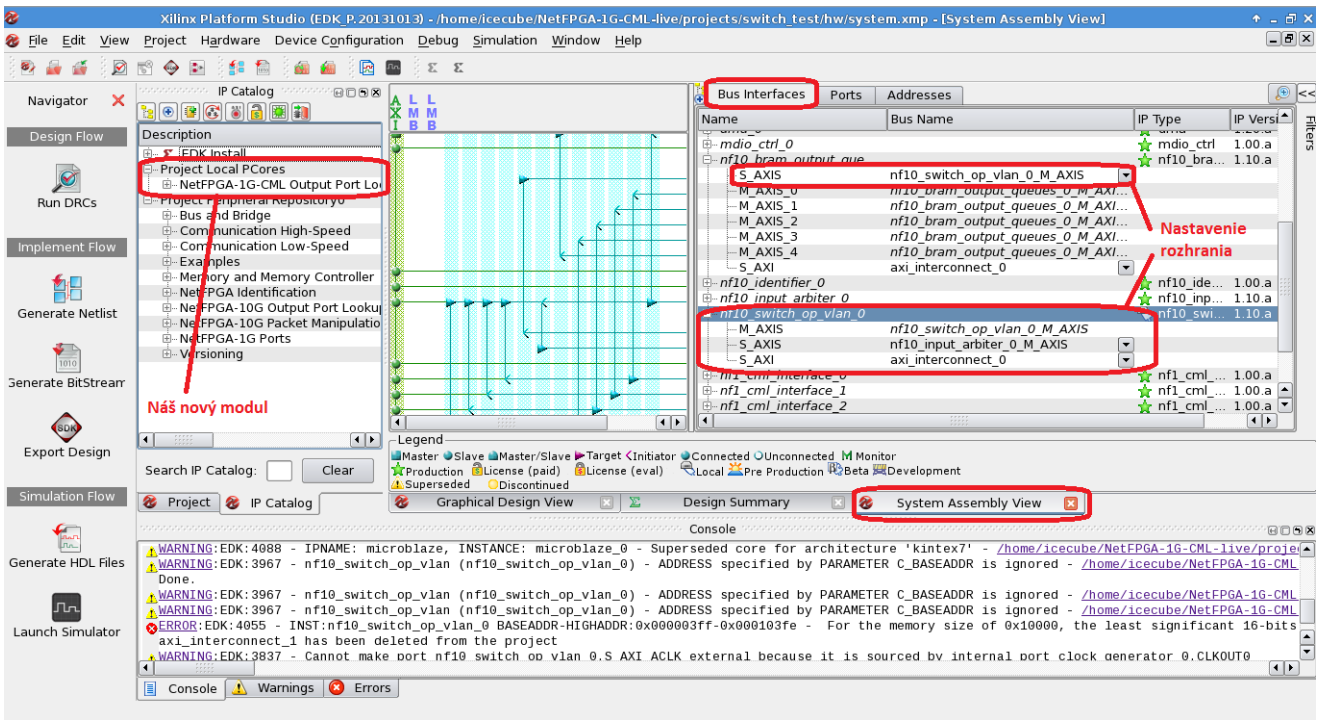
Po úspešných úpravách modulu sa môžeme zamerať na vykonanie zmien v projekte. Spustíme XPS a otvoríme náš nový projekt.

$ xps $NF_DESIGN_DIR/hw/system.xmp

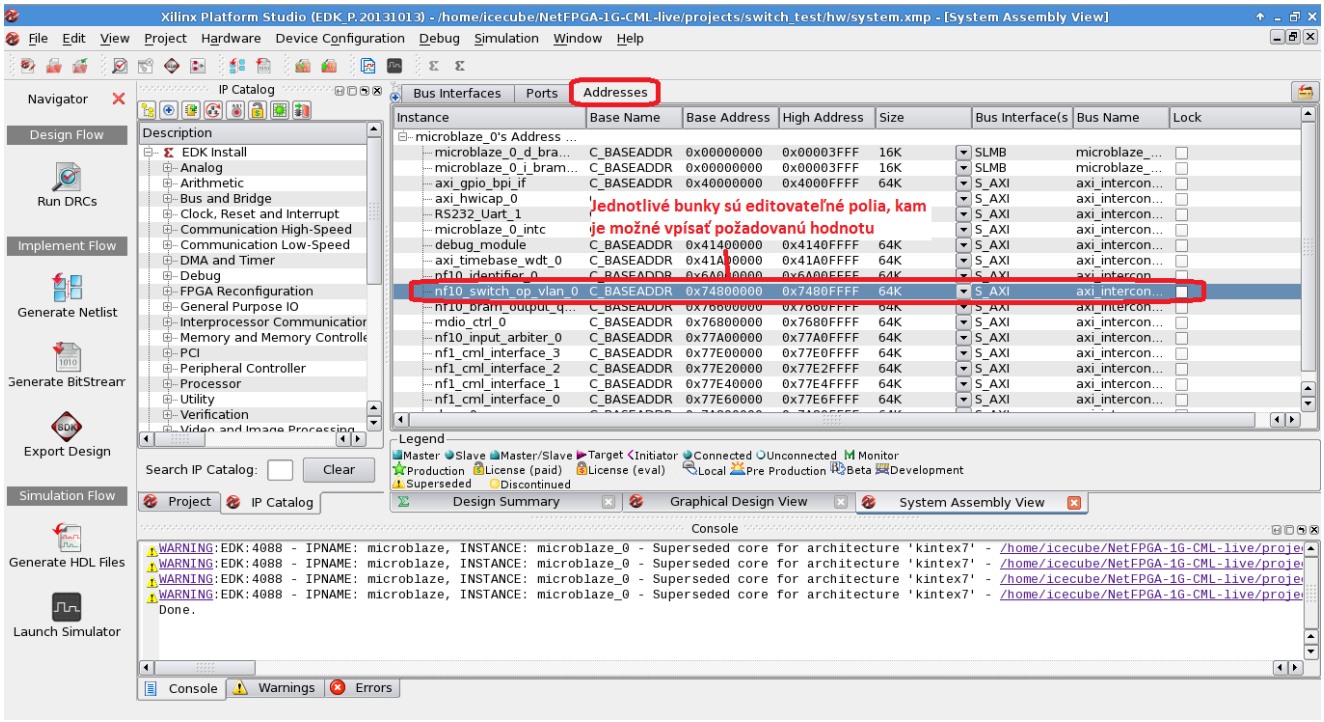
Prvým krokom je odstránenie inštancie pôvodného modulu z návrhu. V **System Assembly View** klikneme na inštanciu modulu **nf10_switch_output_port_lookup_0**, pravým kliknutím myši zvolíme možnosť **Delete Instance** a potvrdíme. Ak tento krok prebehol úspešne, daná inštancia modulu je preč a my môžeme pridať inštanciu nového modulu. V IP katalógu sa už nachádza náš modul uvedený ako lokálny. Klikneme naň, zvolíme **Add IP** a potvrdíme. Inštancia sa okamžite objaví v **System Assembly View**. Okrem tejto inštancie vzniknú aj dve ďalšie - axi_interconnect_1 a axi2axi_connector_1. Tieto inštancie môžeme okamžite zmazať. V tabuľke **Bus Interfaces** nastavíme rozhranie novej inštancie a zmeníme **S_AXIS** inštancie **nf10_bram_output_queues_0** (obr. 8). V tabuľke **Ports** nastavíme **S_AXI_ARESETN** na hodnotu **proc_sys_reset_0::Peripheral_aresetn**. V poslednej tabuľke **Addresses** nastavíme adresný priestor novej inštancie (obr. 9).

Všetky úpravy sú hotové a my môžeme vygenerovať konfiguračný súbor tak, ako je uvedené vyššie. Tento proces je časovo náročný (v našom prípade trval viac ako hodinu). Po úspešnom vykonaní predchádzajúceho procesu je potrebné vykonať nasledujúce príkazy, aby sme získali požadovaný súbor **$NF_DESIGN_DIR/sw/embedded/result/download.bit**. Pred spustením príkazov musíme zmeniť názov inštancie nášho nového modulu v súbore **$NF_DESIGN_DIR/sw/embedded/SDK_Workspace/bsp/system.mss**. V inom prípade by tieto príkazy skončili s chybou, ktorá informuje o neexistujúcom module.

$ cd $NF_DESIGN_DIR/sw/embedded/
$ make



OBRÁZOK Č. 8: XPS - Nastavenie rozhrania modulu



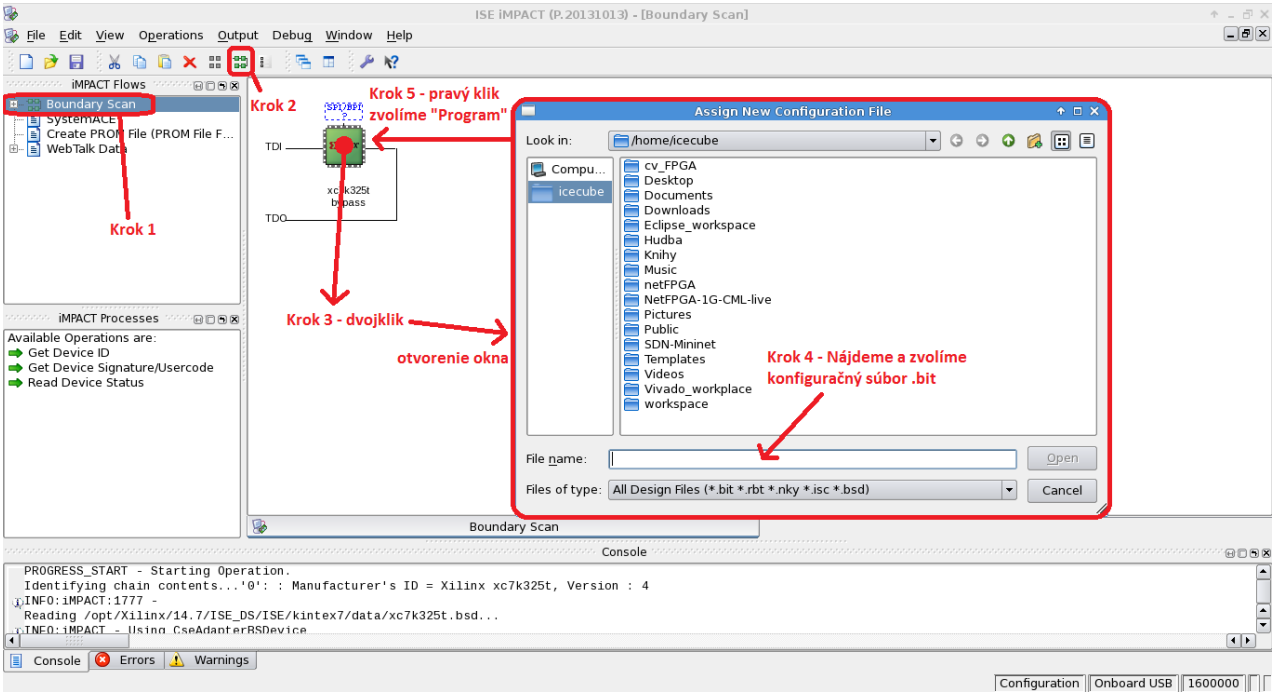
OBRÁZOK Č. 9: XPS - Nastavenie adresného priestoru

5.2.6 Vloženie konfiguračného súboru do FPGA obvodu

Pre vykonanie tejto operácie použijeme nástroj iMPACT. Postup je popísaný na nasledujúcom obrázku (obr. 10). Následne je potrebné obnoviť PCIe zariadenie v PC. K tomuto účelu môžeme použiť dva rôzne postupy: reštartovanie PC / vykonanie skriptu **$NF_ROOT/tools/scripts/pci_rescan_run.sh**. Nakoniec nainštalujeme **nf10.ko** ovládač a aktivujeme všetky sieťové rozhrania prislúchajúce k NetFPGA karte.

$ sudo insmod nf10.ko

$ for i in 0,1,2,3 do; sudo ifconfig nf$i up; done



OBRÁZOK Č. 10: IMPACT - Vloženie konfiguračného súboru

5.2.7 Použitie CLI (Command-Line Interface)

Operácie vytvorenia a vloženia konfiguračného súboru do FPGA obvodu je možné vykonať aj priamo z príkazového riadku s použitím nástroja **make**. Každý z referenčných projektov obsahuje *Makefile*, ktorý zabezpečí všetky potrebné kroky, ktoré súvisia s vytvorením konfiguračného súboru a jeho vložením do karty. Predpokladom je pripravený hotový projekt.

$ cd $NF_DESIGN_DIR

$ make

$ make download

6 NETFPGA IMPLEMENTÁCIE SIEŤOVÝCH ZARIADENÍ

V tejto kapitole rozanalyzujeme, popíšeme a otestujeme referenčné projekty, ktoré sú uvedené v NetFPGA repozitári ako východiskové projekty. Ich úpravou sme schopní realizovať vlastné projekty.

6.1 REFERENČNÝ NIC PROJEKT

Najjednoduchším referenčným projektom, ktorý sa nachádza v NetFPGA repozitári, je **reference_nic_nf1_cml** projekt. Jeho vložením do NetFPGA karty sa táto karta správa ako bežná NIC karta so štyrmi nezávislými 1G fyzickými rozhraniami. V tejto kapitole sa venujeme analýze tohto systém a popisu požiadaviek naň. Následne uvedieme teoretické východiská, na ktorých je systém založený a postup, ako ho implementovať. Aby sme si overili, že systém spĺňa dané požiadavky, na záver vykonáme viacero testov.

6.1.1 NIC - Analýza a požiadavky

Hlavnou požiadavkou na systém je regulácia sieťovej prevádzky tak, aby rámce prichádzajúce zo siete do fyzických portov, boli presmerované na virtuálne DMA porty, a tak odoslané cez PCIe do PC. Súčasne rámce prijaté z PC cez jeden z virtuálnych DMA portov musia byť presmerované a odoslané cez prislúchajúci fyzický port do siete. Aby sme dosiahli nezávislosti medzi jednotlivými fyzickými rozhraniami, musíme špecifikovať ďalšie požiadavky:

- rámec, ktorý je prijatý cez MACX (X-tý fyzický port karty), je do PC odoslaný cez NFX (X-tý virtuálny DMA port). Tento rámec je následne v PC prijatý cez sieťové rozhranie nfX**.**
- rámec, ktorý odošleme z PC cez rozhranie nfX, musí karta po prijatí odoslať cez fyzické rozhranie MACX.

Tieto požiadavky zabezpečia, že sa systém bude správať ako reálna sieťová karta.

6.1.2 NIC - Teoretické východiská

Referenčný NIC projekt je postavený na architektúre, ktorá je popísaná vyššie v kapitole 4. Projekt pozostáva z modulov, ktoré sú uvedené v tejto architektúre. Pre jednoduchosť neuvádzame v vedľajšie moduly, ktoré poskytujú podporné funkcie, ako napr. modul pre UART rozhranie a modul pre generátor hodinového signálu. Hlavná logika systému je implementovaná v module **nf10_nic_output_port_lookup**.

Zdrojový port každého rámca, ktorý je prijatý na jednom z rozhraní karty (fyzický port alebo PCIe), je uložený v TUSER[23:16] ako 8-bitový vektor. Formát vektora je popísaný v kapitole 4 pri špecifikácii paketovej zbernice. Každý nepárny bit vektora prislúcha jednému fyzickému portu a každý párny bit jednému virtuálnemu DMA portu. Ak bol rámec prijatý cez daný port, hodnota prislúchajúceho bitu v tomto vektore je rovná logickej jednotke, v inom prípade je rovná logickej nule. Okrem zdrojového portu je pre každý rámec špecifikovaný aj **cieľový port**. Jeho formát je rovnaký ako v prípade zdrojového portu, pričom logická jednotka bitu prislúchajúceho k danému portu znamená, že rámec bude z karty odoslaný týmto portom.

Aby sme splnili požiadavky špecifikované vyššie, použijeme nasledujúce rozhodovanie:

- Ak bol rámec prijatý z fyzického portu (aspoň jeden nepárny bit v zdrojovom porte je nenulový), musíme všetky bity zdrojového portu posunúť o jednu pozíciu vľavo a výsledok uložiť do cieľového portu. Takto sa logické jednotky fyzických portov dostanú na pozície bitov pre prislúchajúce virtuálne porty, cez ktoré bude rámec následne odoslaný - prvý bod v požiadavkách.
- Ak bol rámec prijatý z virtuálneho portu (aspoň jeden párny bit v zdrojovom porte je nenulový), musíme všetky bity zdrojového portu posunúť o jednu pozíciu vpravo a výsledok uložiť do cieľového portu. Princíp je analogický k predchádzajúcemu bodu a zabezpečuje druhý bod v požiadavkách.

Predpokladáme, že karta môže prijať daný rámec len cez jedno rozhranie, a tak bude zdrojový port pri prijatí rámca obsahovať len jeden jednotkový bit. Nemôže teda nastať prípad, že by pre konkrétny rámec nastali obe vyššie uvedené situácie súčasne. Implementácia je pomerne jednoduchá a nachádza sa v súbore **nf10_nic_output_port_lookup.v**, ktorý je súčasťou modulu.

6.1.3 NIC - Realizácia

Realizácia tohto projektu v zmysle jeho nastavení, vytvorenia konfiguračného súboru a vloženia do NetFPGA karty je rovnaká ako vo vzorovom príklade, ktorý je uvedený v predchádzajúcej kapitole. Projekt nevyžaduje žiadne dodatočné kroky, ktoré by bolo potrebné vykonať. V repozitári nie je vytvorená žiadna softvérová aplikácia určená pre tento projekt. Jednotlivé fyzické porty sú viditeľné z PC ako nf0,.., nf3. Spôsob nastavenia ich

parametrov (IP adresa, maska, brána, atď.) je rovnaký ako pre rozhranie bežnej sieťovej karty. Ďalšie informácie sú uvedené v [5].

6.1.4 NIC - Testovanie

Pre účely testovanie sme použili testovacie skripty, ktoré sú súčasťou tohto projektu. Všetky z testov, ktoré sme vykonali podľa postupu uvedeného v kapitola 3, prebehli úspešne. Okrem nich bol realizovaný fyzický test. Testovací prípad pozostáva z pripojenia jedného zo sieťových rozhraní NetFPGA karty do internetu. Pre získanie IP adresy a ďalších potrebných parametrov rozhrania sme použili protokol DHCP (Dynamic Host Configuration Protocol). Vďaka DHCP serveru prebehli nastavenia rozhrania bez problémov. Priradené parametre sme overili nástrojom **ifconfig**. Generovanie sieťovej prevádzky sme vyriešili použitím nástroja **ping**, ktorý je základným sieťovým nástrojom pre prípady testovania aktívneho spojenia. Vzájomná výmena ICMP správ bola úspešná (obr. 11). Aby sme otestovali aj protokoly vyššej vrstvy (UDP a TCP), použili sme nástroj **ncat** pre generovanie TCP a UDP správ. Oba typy prevádzky boli úspešne zaznamenané.

```
File  Edit  View  Terminal  Tabs  Help
[981][netfpga: scripts]$ sudo ifconfig nf2 up
[982][netfpga: scripts]$ sudo dhclient nf2
[983][netfpga: scripts]$ ifconfig nf2
nf2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 158.193.139.203  netmask 255.255.255.0  broadcast 158.193.139.255
        inet6 fe80::24e:46ff:fe31:3002  prefixlen 64  scopeid 0x20<link>
        ether 00:4e:46:31:30:02  txqueuelen 1000  (Ethernet)
        RX packets 1240  bytes 166385 (162.4 KiB)
        RX errors 0  dropped 8  overruns 0  frame 0
        TX packets 100  bytes 12171 (11.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 60

[984][netfpga: scripts]$ ping -c 4 google.sk
PING google.sk (173.194.116.111) 56(84) bytes of data.
64 bytes from fra02s27-in-f15.1e100.net (173.194.116.111): icmp_seq=1 ttl=56 time=20.8 ms
64 bytes from fra02s27-in-f15.1e100.net (173.194.116.111): icmp_seq=2 ttl=56 time=20.7 ms
64 bytes from fra02s27-in-f15.1e100.net (173.194.116.111): icmp_seq=3 ttl=56 time=20.8 ms
64 bytes from fra02s27-in-f15.1e100.net (173.194.116.111): icmp_seq=4 ttl=56 time=20.8 ms

--- google.sk ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 20.714/20.809/20.859/0.056 ms
[985][netfpga: scripts]$
```

OBRÁZOK Č. 11: NIC projekt - Priebeh testu

Výsledky získané z jednotlivých testov dokazujú, že tento systém spĺňa požiadavky, ktoré sú uvedené vyššie, a jeho správanie zodpovedá správaniu sieťovej NIC karty.

6.2 REFERENČNÝ PROJEKT SWITCH (PREPÍNAČ)

Ďalší z projektov v repozitári je **reference_switch_nf1_cml**. Jeho vložením do karty sa táto NetFPGA karta správa ako bežný prepínač so štyrmi 1G fyzickými portami, ktorý je schopný učiť sa, a tak aktualizovať údaje vo svojej prepínacej CAM (Content Addressable Memory) tabuľke. V tejto časti sa zameriame na analýzu a požiadavky tohto systému, uvedieme stručné východiská, na ktorých je vybudovaný, popíšeme postup jeho konfigurácie a overíme jeho korektnosť vykonaním viacerých testov.

6.2.1 SWITCH - Analýza a požiadavky

Úlohou prepínača v počítačovej sieti je efektívne spracovanie prijatých rámcov na jednom z fyzických portov a ich odoslanie len vhodnými fyzickými rozhraniami späť do siete. Tým sa prepínač líši od rozbočovača, ktorý odosiela prijaté rámce vždy všetkými portami okrem zdrojového portu. Prepínač je povinný udržiavať databázu naučených MAC adries a k nim priradených portov. Táto databáza je riešená formou **CAM** tabuľky, ktorá je pri prvotnom zapnutí prepínača prázdna. Na základe zdrojových adries a fyzických portov prijatých rámcov sú do CAM tabuľky vkladané nové záznamy. Vďaka tomuto procesu sa prepínač učí. Po prijatí rámca je povinnosťou prepínača rozhodnúť, ktorými fyzickými portami odošle rámec von. Tento proces závisí od aktuálneho stavu CAM tabuľky. Pre konkrétny rámec môžu nastať dve situácie, podľa ktorých je obslúžený:

- Ak cieľová MAC adresa prijatého rámca nie je uvedená v CAM tabuľke, rámec je odoslaný všetkými fyzickými portami von okrem zdrojového portu. Toto správanie sa nazýva **unknown unicast flooding**.
- Ak je cieľová MAC adresa prijatého rámca uvedená v CAM tabuľke, rámec je odoslaný len tým fyzickým portom, ktorý je uvedený v danom zázname.

Výnimkou je tzv. **broadcast MAC adresa** (ff:ff:ff:ff:ff:ff), pre ktorú tieto pravidlá neplatia. Rámec s takouto cieľovou adresou je spracovaný rovnako ako v prvom prípade.

Okrem cieľovej MAC adresy je v procese spracovania potrebná aj zdrojová MAC adresa prijatého rámca. Tá je použitá pre aktualizáciu CAM tabuľky. Prijatím rámca s danou zdrojovou adresou na konkrétnom fyzickom porte predpokladáme, že stanica s danou adresou sa nachádza za daným portom. Preto vložíme do CAM tabuľky nový záznam s danou zdrojovou MAC adresou a portom, prípadne aktualizujeme existujúci záznam, ak sa v tabuľke položka s danou MAC adresou už nachádza.

Tieto požiadavky popisujú základné správanie sieťového prepínača, pričom sme pre jednoduchosť neuviedli podrobnejšie operácie, ktoré musí prepínač v skutočnosti vykonať.

6.2.2 SWITCH - Teoretické východiská

Tento projekt je založený na architektúre popísanej v kapitole 4, a teda pozostáva z rovnakých modulov ako predchádzajúci referenčný NIC projekt. Preto sa ďalej venujeme len popisu modulu **nf10_switch_output_port_lookup**, v ktorom je implementovaná logika prepínača. Tento modul pozostáva z viacerých zdrojových súborov, v ktorých sú implementované jednotlivé časti logiky. Hlavnou úlohou modulu je špecifikovanie **cieľového portu** rámca podľa definovaných požiadaviek.

Prvý krok, ktorý je potrebné vykonať okamžite po prijatí rámca, spočíva v získaní cieľovej a zdrojovej MAC adresy ako aj zdrojového portu, cez ktorý bol rámec prijatý. Tieto údaje sú umiestnené v TDATA[47:0], TDATA[95:48] a TUSER[23:16]. Proces parsovania sa nachádza v **ethernet_parser.v**.

Získané údaje z hlavičky rámca použijeme ako vstupné údaje do CAM tabuľky. Pre jej implementáciu je použitý samostatný modul. Ten je stručne popísaný v realizácii tohto projektu. Podľa hodnoty výstupného signálu **cam_match** jednoznačne vieme, či bolo vyhľadávanie úspešné, a teda dokážeme určiť, ktorý z prípadov uvedených v popise požiadaviek nastal.

- Ak záznam s cieľovou MAC adresou prijatého rámca v tabuľke neexistuje, v **cieľovom porte** sú bity prislúchajúce k všetkým fyzickým portom s výnimkou zdrojového portu rovné logickej jednotke. Keďže používame len fyzické porty NetFPGA karty, všetky bity priradené k virtuálnym DMA portom zostanú rovné logickej nule.
- Ak je v tabuľke nájdený záznam, ktorý zodpovedá danej cieľovej MAC adrese prijatého rámca, na výstupe z modulu CAM tabuľky získame výstupný port ako 8-bitový vektor, ktorého kódovanie je rovnaké ako v prípade zdrojového portu. Bit prislúchajúci zdrojovému portu vymaskujeme a výsledný vektor použijeme ako **cieľový port**.

Prípad **broadcast** adresy riešime tak, že túto MAC adresu automaticky vložíme do CAM tabuľky ako posledný záznam, pri ktorom uvedieme ako výstupný port vektor použitý v prvom prípade. Vďaka tomu bude pre rámce s cieľovou broadcast adresou vždy nájdený

záznam v CAM tabuľke. Proces vyhľadávania v CAM tabuľke a priradenie cieľového portu sa nachádza v **mac_cam_lut.v**.

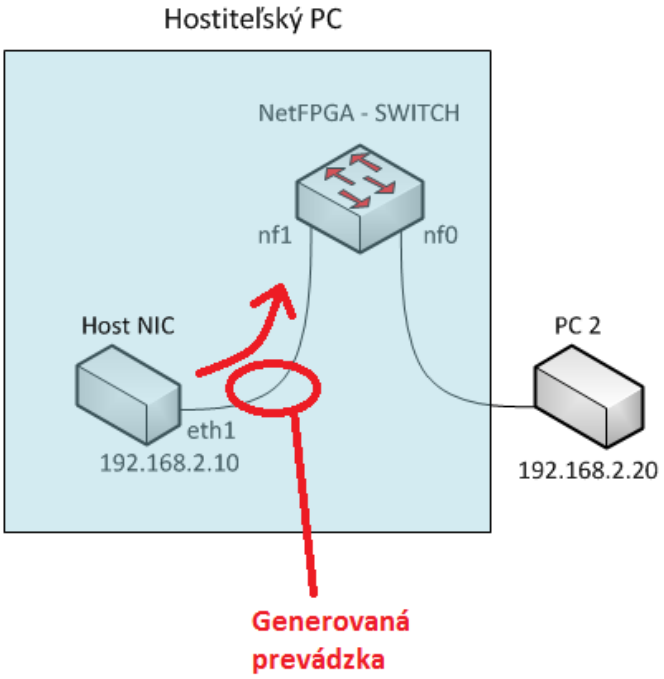
Keďže žiadna iná situácia pre prijatý rámec okrem vyššie uvedených nemôže nastať, pre každý rámec bude jednoznačne špecifikovaný cieľový port, ktorý spĺňa požiadavky na systém a rámec bude odoslaný cez požadované sieťové rozhrania.

6.2.3 SWITCH - Realizácia

Pre realizáciu tohto projektu je použitý takmer rovnaký postup ako vo vzorovom príklade. Projekt vyžaduje dodatočné kroky, ktoré je potrebné vykonať. Tieto kroky súvisia s vygenerovaním modulu pre CAM tabuľku, ktorá je v projekte použitá. Tento modul pochádza priamo od spoločnosti Xilinx a jeho zdrojové súbory sú voľne dostupné. Jeho konfigurácia a inštalácia sú popísané v README dokumente, ktorý je umiestnený v adresári projektu. Tento modul je možne použiť všade, kde je potrebná CAM, resp. TCAM tabuľka, pričom pri generovaní môžeme prostredníctvom rôznych parametrov nastaviť tabuľku podľa našich potrieb, napr. definovanie jej šírky a hĺbky. Podrobnejšie informácie o CAM module sú uvedené v [**16**]. V repozitári je vytvorená len jednoduchá softvérová aplikácia určená pre monitorovanie registrov projektu. Vďaka úplnej nezávislosti tohto systému od PC, môže takto nakonfigurovaná NetFPGA karta bežať aj bez pripojenia k PC v *standalone* režime. Ďalšie informácie sú uvedené v [**5**].

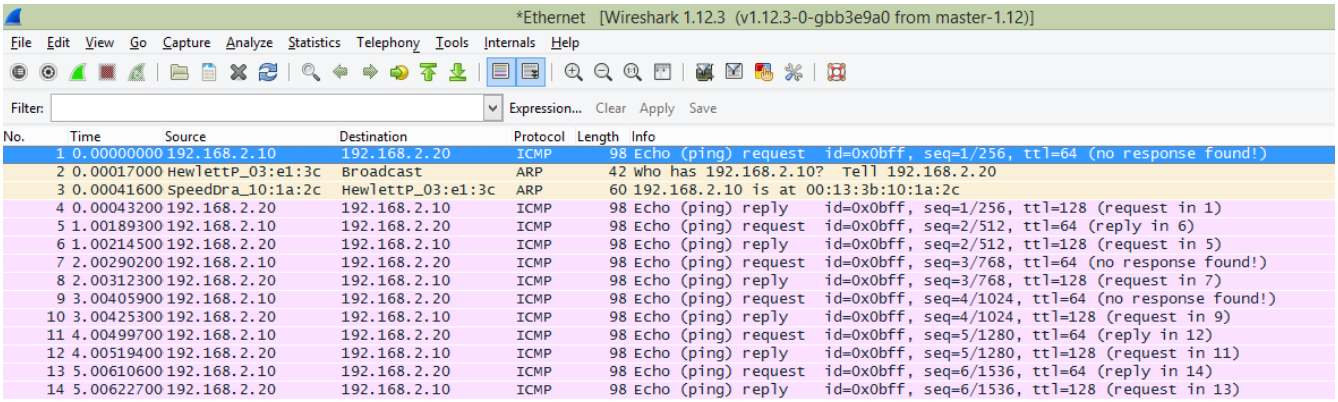
6.2.4 SWITCH - Testovanie

Pre overenie funkčnosti vyššie uvedeného projektu, ktorý predstavuje ethernetový prepínač, boli vykonané viaceré testy. Po realizácii projektu sme podobne ako v prípade NIC karty najprv vykonali všetky dostupné testy realizované vo forme testovacích skriptov popísaných v kapitole 3, ktoré sú pre tento projekt k dispozícii v NetFPGA repozitári. Všetky tieto testy dopadli úspešne. Okrem nich boli realizované aj fyzické testy. Aby sme reálne overili funkcie prepínača, ktoré má NetFPGA karta poskytovať, vytvorili sme jednoduché zapojenie. Jedno z fyzických rozhraní NetFPGA karty sme prepojili so sieťovou kartou, ktorá je súčasťou hostiteľského PC. Druhé rozhranie NetFPGA karty sme pripojili do rozhrania sieťovej karty umiestnenej v PC2. Parametre jednotlivých sieťových rozhraní sme nastavili podľa topológie, ktorá je uvedená nižšie (obr. 12).



OBRÁZOK Č. 12: SWITCH - Testovacia topológia

Pre otestovanie vzájomnej konektivity medzi stanicami (Host NIC a PC2) bol použitý nástroj **ping**. Pre generovanie TCP a UDP paketov sme použili nástroj **ncat**. Zdrojom prevádzky, kde bolo generovanie spustené, je rozhranie hostiteľského PC **eth1** (obr. 12). Na nižšie uvedenom obrázku sú viditeľné ICMP správy zachytené v PC2 (obr. 13). Na základe úspešnej výmeny správ medzi stanicami považujeme testy za úspešné.



OBRÁZOK Č. 13: SWITCH - Zachytená prevádzka v PC2

Na ostatných rozhraniach NetFPGA karty bol zaznamenaný len jeden rámec od každého zo zúčastnených PC. Tieto rámce predstavujú situáciu - **unknown unicast flooding**, kedy ešte nie je v CAM tabuľke uvedený záznam s danou cieľovou adresou, a teda rámec bol odoslaný na všetky porty okrem vstupného rozhrania. Prijatím týchto rámcov bol do CAM tabuľky vložený nový záznam s danou zdrojovou MAC adresou, a teda všetky nasledujúce rámce boli

prepnuté už len na požadované rozhranie **nf0**, resp. **nf1**. Na základe úspešnosti testov môžeme usúdiť, že daný projekt je funkčnou implementáciou ethernetového LAN prepínača.

6.3 REFERENČNÝ PROJEKT ROUTER (SMEROVAČ)

Posledným z nami analyzovaných projektov je **reference_router_nf1_cml**, ktorý je svojou štruktúrou zložitejší než predchádzajúce projekty. Použitím tohto projektu sme schopní nakonfigurovať kartu tak, aby sa v spolupráci s hostiteľským PC správala ako IPv4 smerovač so štyrmi 1G portami. V tejto časti popíšeme požiadavky na systém, uvedieme teoretické základy, na ktorých je postavená implementácia, popíšeme kroky potrebné pre realizáciu a otestujeme výsledný systém, aby sme sa uistili, že sú splnené všetky požiadavky.

6.3.1 ROUTER - Analýza a požiadavky

IPv4 smerovač je jedným z najznámejších elementov súčasných počítačových sietí. Jednou z hlavných úloh smerovača je smerovanie paketov podľa ich cieľových IP adries medzi sieťami, ktoré sú založené na rôznych prenosových technológiách. Vďaka tomu sú koncové stanice pripojené do týchto sieti schopné vzájomne komunikovať. Riešenie tohto problému bolo pôvodným impulzom pre vznik prvých smerovačov.

Smerovač pozostáva z **riadiacej** a **dátovej** roviny. V riadiacej rovine sú vykonávané zložitejšie algoritmy, ktoré okrem iného zabezpečujú spracovanie neobvyklých paketov, aktuálnosť jednotlivých tabuliek a celkovo riadia správanie smerovača. Úlohou dátovej roviny je rýchle smerovanie paketov zo vstupných rozhraní na výstupné rozhrania na základe aktuálnych údajov v jednotlivých tabuľkách. Súčasťou tohto procesu je kontrola a prípadná modifikácia niektorých položiek IP hlavičky paketu.

Všetky požiadavky, ktoré musí dané zariadenie zabezpečiť, aby bolo prijaté ako IPv4 smerovač, ako aj podrobný postup požadovaného spracovania paketov sú presne špecifikované v **RFC 1812** [**17**].

6.3.2 ROUTER - Teoretické východiská

Na základe požiadaviek uvedených vyššie v danom dokumente je každá z rovín riešená samostatne. Vzhľadom na funkcie, ktoré má riadiaca rovina poskytovať, a vzhľadom na ich zložitosť je táto rovina v našom projekte implementovaná vo forme softvérovej aplikácie, ktorá beží v hostiteľskom PC ako samostatný proces. Popis jednotlivých nástrojov,

ktoré poskytujú funkcionalitu tejto roviny, je uvedený v realizácii projektu. Operácie, ktoré tvoria dátovú rovinu, je možné implementovať priamo v NetFPGA karte.

Časť systému riešená v NetFPGA karte, ktorá vykonáva úlohy dátovej roviny, je aj v tomto prípade založená na architektúre z kapitoly 4, a teda pozostáva z rovnakých modulov ako predchádzajúce projekty. Od nich sa líši len logikou, ktorá je súčasťou **nf10_router_output_port_lookup** modulu. Hlavnou úlohou modulu je špecifikácia **cieľového portu** pre každý z prijatých rámcov tak, aby boli splnené dané požiadavky. Ďalej sa venujeme len popisu jednotlivých krokov, z ktorých pozostáva proces spracovania v tomto module. Tie sú umiestnené v samostatných zdrojových súboroch.

Po prijatí rámca na jednom zo vstupných rozhraní, musíme prečítať jeho cieľovú MAC adresu a overiť, že je zhodná s MAC adresou jedného zo sieťových rozhraní NetFPGA karty, a teda tento rámec je určený pre nás. Okrem MAC adresy prečítame hodnotu **ethertype**, ktorá špecifikuje typ protokolu použitého v tele rámca. Tieto operácie sú špecifikovaná v **eth_parser.v**.

V ďalších krokoch sa venujeme už len IP paketu, ktorý sa nachádza v tele prijatého rámca. Najprv skontrolujeme správnosť **IP checksum** a **TTL** (Time To Live) hodnôt v hlavičke. Ak sú korektné, znížime TTL o jednotku a vypočítame novú hodnotu IP checksum položky. Tieto kroky sú uvedené v **ip_checksum_ttl.v**.

Aby sme vedeli, kam daný paket smerovať, musíme pre cieľovú IP adresu vyhľadať prislúchajúcu **next-hop IP adresu**. K tomuto účelu slúži **smerovacia tabuľka**, ktorá je v systéme realizovaná ako **TCAM** (Ternary CAM) tabuľka. Ak je pri danej cieľovej IP adrese uvedená nulová IP adresa skoku, cieľová stanica sa nachádza v priamo pripojenej sieti. Okrem adresy skoku je pri každom zázname smerovacej tabuľky uvedený cieľový port, ktorým máme nakoniec odoslať paket von. Tento proces je uvedený v **ip_lpm.v**. Ak bolo vyhľadávanie úspešné a my máme IP adresu skoku, potrebujeme k nej získať prislúchajúcu MAC adresu. Tú vyhľadáme v **ARP tabuľke**, ktorá je v systéme realizovaná ako **CAM** tabuľka. Tento proces sa nachádza v **ip_arp.v**.

Okrem týchto dvoch tabuliek je ešte použitá **filtrovacia tabuľka**, ktorá je taktiež realizovaná v systéme ako **CAM** tabuľka. Pakety, ktorých cieľová IP adresa je uložená v tejto tabuľke, sú automaticky odosielané do PC. Tento filter slúži pre účely riadenia. Medzi takto

filtrované pakety patria napríklad pakety smerovacích protokolov a pakety, ktorých cieľovou stanicou je priamo NetFPGA karta. Tento proces je uvedený v **dest_ip_filter.v**.

Podľa hodnôt získaných z predchádzajúcich krokov rozhodneme, čo s prijatým rámcom vykonáme. Ak sa cieľová MAC adresa prijatého rámca nezhoduje s MAC adresou fyzického rozhrania, cez ktoré bol rámec prijatý, tento rámec je zahodený. To isté platí aj pre rámce, ktoré obsahujú IP paket s nesprávnou hodnotou položky IP checksum. V prípade rámcov, ktorých IP pakety úspešne prešli cez všetky kroky a ich cieľová IP adresa nie je filtrovaná, aktualizujeme zdrojovú a cieľovú MAC adresu a odošleme ich príslušným cieľovým portom von. V inom prípade, kedy rámec neobsahuje IP paket alebo aspoň jeden z predchádzajúcich krokov preň skončil neúspešne, je tento rámec odoslaný do PC na ďalšie spracovanie v riadiacej rovine. Proces rozhodovania sa nachádza v **op_lut_process_sm.v**. Jednotlivé časti sú navzájom prepojené do jedného celku v **output_port_lookup.v** a zabalené do výsledného modulu v **nf10_router_output_port_lokup.v**.

Vo vyššie uvedenom procese rozhodovania je presne špecifikovaná operácia pre každý z prípadov, ktoré môžu nastať. Jeho správnosť je overená formou testov, ktoré sú vykonané nad týmto systémom.

6.3.3 ROUTER - Realizácia

Realizácia tohto projektu je rozdelená do dvoch častí. Každá z nich prislúcha k jednej z rovín, z ktorých projekt pozostáva. Postup realizácie dátovej roviny je veľmi podobný postupu, použitému v predchádzajúcom projekt. Okrem bežných krokov uvedených vo vzorovom príklade je v tomto prípade potrebné vygenerovať moduly pre CAM a TCAM tabuľky, ktoré sú v systéme použité. Zdrojové súbory pre tieto tabuľky sú rovnaké ako v prípade prepínača. Modul každej z tabuliek je vygenerovaný samostatne. Podrobný popis krokov, ktoré je potrebné vykonať, je uvedený v README súbore, ktorý sa nachádza v adresári projektu.

Ako už bolo uvedené skôr, riadiaca rovina je realizovaná formou softvérových aplikácii, ktoré sú súčasťou projektu a sú spustené v hostiteľskom počítači. Tu uvedieme popis každej z použitých aplikácií a prípadné dodatočné kroky, ktoré musíme pre danú aplikáciu vykonať.

SCONE – Software Component of NetFPGA

Hlavnou aplikáciou, ktorá riadi NetFPGA kartu a zabezpečuje funkcie riadiacej roviny, je **SCONE**. Táto aplikácia je komplexným softvérom, v ktorom sú implementované jednoduché verzie IP, ICMP, TCP a UDP protokolov. Súčasťou je aj smerovací protokol PW-OSPF. Ten je založený na známom OSPF protokole, ale nie je s ním vzájomne kompatibilný. Úlohou nástroja je spracovanie správ týchto protokolov. Okrem toho poskytuje niektoré služby aplikačnej vrstvy ako napr. TELNET a HTTP (HyperText Transfer Protocol). SCONE si v operačnej pamäti uchováva všetky informácie ako sú napríklad smerovacia tabuľka a ARP tabuľka. Pri ich zmene sú tieto údaje okamžite odoslané na NetFPGA kartu a aktualizované. Aktuálne nastavenia SCONE je možné meniť cez TELNET vzdialeným pripojením na jeden z fyzických rozhraní karty použitím IP adresy daného rozhrania. Inou možnosťou je použitie CLI a GUI, ktoré sú popísané nižšie.

SCONE využíva knižnice **libnet** a **libpcap**, ktoré sme povinní nainštalovať ešte pred jeho kompiláciou. Tú môžeme vykonať až potom, ako sme vygenerovali *reg_defines.h* (viď. kapitola 4). Pre samotnú kompiláciu zdrojových súborov aplikácie, ktoré sa nachádzajú v NetFPGA repozitári, sme použili nástroj **make**.

```
$ sudo yum install libpcap libpcap-devel libnet-devel
$ cd $NF_DESIGN_DIR/sw/host/scone
$ make
```

Pri používaní SCONE sme objavili obmedzenie v podobe dĺžky používateľského meno (**hostname**), ktoré môže byť dlhé maximálne 32 znakov. Ak je aktuálny hostname dlhší, musíme vykonať jeho zmenu. V inom prípade sa pri spustení aplikácie objaví chybové upozornenie a program následne skončí. SCONE vyžaduje existenciu súborov cpuhw a rtable vo svojom adresári ešte pred spustením. Z týchto súborov načítava aplikácia inicializačné nastavenia smerovača.

```
$ sudo hostname <nove_meno_do_32_znakov>
$ cd $NF_DESIGN_DIR/sw/host/scone/
$ sudo ./scone
```

RKD – Router Kit Daemon

Ako náhradu za SCONE máme k dispozícii aplikáciu **RKD**, ktorá používa pre riadenie smerovača údaje priamo nastavené v operačnom systéme. RKD neustále monitoruje

smerovaciu tabuľku, ARP tabuľku a nastavenia sieťových rozhraní **nf0, .., nf3** v operačnom systéme. Pri zmene sú údaje, ktoré prislúchajú k jednému z vyššie uvedených rozhraní, okamžite posielané na NetFPGA kartu. Vďaka jednoduchému prístupu tejto aplikácie je možné jednoducho pomocou známych systémových nástrojov (ip, route, arp, atď.) pridávať záznamy, meniť tabuľky a modifikovať nastavenia jednotlivých rozhraní. Keďže nezáleží na spôsobe, ako sú tieto údaje zmenené, môžeme použiť ľubovoľný nástroj, ktorý sa stará o ich aktualizáciu. Príkladom sú smerovacie nástroje Zebra/Quagga a XoRP, s ktorými je schopný RKD nástroj vzájomne spolupracovať. Kompilácia tohto nástroja je rovnaká ako v prípade SCONE.

Java GUI (Graphical User Interface) a CLI (Command-Line Interface)

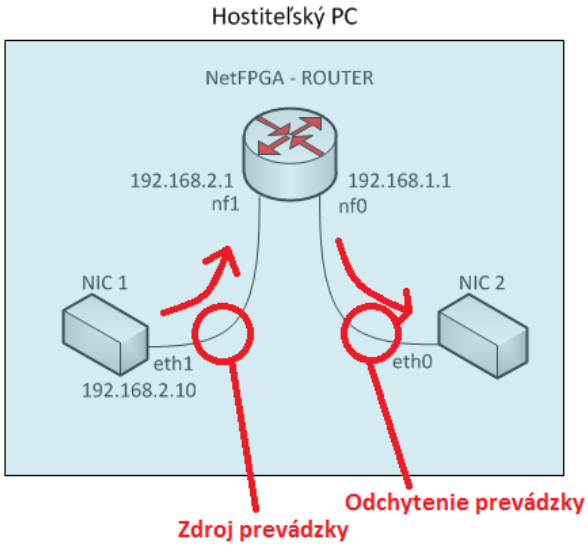
V spolupráci s vyššie uvedenými nástrojmi máme k dispozícii CLI a GUI aplikácie, pomocou ktorých môžeme sledovať a meniť aktuálne nastavenia nášho smerovača. **Java GUI** poskytuje používateľovi prehľadné grafické prostredie vrátane grafov, ktoré zobrazujú aktuálnu sieťovú prevádzku na jednotlivých sieťových rozhraniach. Táto aplikácia je napísaná v jazyku Java, a preto vyžaduje nainštalované **JKD** (Java Development Kit) v operačnom systéme. **CLI** je jednoduchý nástroj, ktorý poskytuje používateľovi rovnaké možnosti ako GUI prostredie len vo forme príkazového riadku. Kompilácia oboch týchto nástrojov je rovnaká ako v prípade SCONE.

Pre podrobnejšie informácie odporúčame preštudovať časti prislúchajúce k tomuto projektu v sekcii **Projects** v [**5**].

6.3.4 ROUTER - Testovanie

Aby sme overili, že systém spĺňa vyššie uvedené požiadavky, vykonali sme testy spojenia. Najprv boli vykonané testovacie skripty, ktoré sú určené pre tento projekt a nachádzajú sa v NetFPGA repozitári. Okrem nich sme realizovali fyzické zapojenie. V tomto zapojení sú použité obe sieťové karty hostiteľského počítača označené **NIC1** a **NIC2**. Sieťové rozhranie každej z nich je pripojené na jeden z portov NetFPGA karty. Rozhranie **eth1** slúži ako zdroj, ktorý generuje sieťovú prevádzku. Jeho IP adresa je uvedená v topológii. Rozhranie **eth0** slúži výlučne na odpočúvanie prevádzky, ktorá smeruje z rozhrania karty **nf0**. Pre účely odpočúvania sme použili nástroj **tcpdump**. Topológia zapojenia je zobrazená na obr. 14. Konfigurácia smerovača je zobrazená v konfiguračnom okne aplikácie Java GUI (obr. 18). Riadiaca rovina smerovača je vo všetkých testoch riešená

cez SCONE aplikáciu, ktorá je popísaná v realizácii. Pre vizuálnu kontrolu stavu smerovača a jeho dodatočnú konfiguráciu sme použili Java GUI aplikáciu. Realizované boli tri testy. Prvý test sa venuje vzájomnej komunikácii medzi stanicami, ktoré sú pripojené do **priamo pripojených sietí**. Teda smerovač je bránou pre obe stanice. V druhom teste sa stanica s cieľovou IP adresou nachádza vo **vzdialenej sieti**, teda jej bránou nie je naša karta. V týchto dvoch testoch je sieťová prevádzka generovaná cez nástroj **ncat.** V treťom teste je našim cieľom otestovať výkon karty ako smerovača, tým že sa pokúsime zahltiť kartu prostredníctvom nástroja **hping3**. Súčasne tak dokážeme, že operácie tohto smerovača sú priamo vykonávané samotnou kartou, keďže CPU počítača sa naplno venuje generovaniu prevádzky. Zdrojová stanica a jej IP adresa majú vo všetkých troch testoch rovnaké nastavenia. Cieľová IP adresa sa v jednotlivých testoch mení. Cieľom fyzických testov je overenie funkcie smerovania paketov medzi rôznymi sieťami. Test považujeme za úspešný, ak sú na danom výstupnom rozhraní odchytené očakávané pakety.



OBRÁZOK Č. 14: ROUTER - Testovacia topológia

Príkazy použité pri generovaní sieťovej prevádzky v jednotlivých testoch:

$ cat /dev/random | sudo ncat –u <cielova_IP> <cielovy_port>
$ sudo hping3 –c 1000000 –d 120 –S –w 64 –p <cielovy_port> --flood <cielova_IP>

TEST 1 – PRIAMO PRIPOJENÁ SIEŤ

Cieľom tohto testu je overenie funkčnosti karty ako smerovača v prípade priamo pripojených sietí. Pre stanice v týchto sieťach platí, že majú spoločnú bránu, ktorou je naša karta. Cieľovou IP adresou fiktívnej stanice, ktorú sme použili pre generované UDP pakety, je

192.168.1.10/24. Táto IP adresa je súčasťou siete (192.168.1.0/24), ktorá je priamo pripojená cez rozhranie **nf0**. Na obr. 15 sú zobrazené odchytené pakety, ktoré na danom rozhraní očakávame.

```
[1004][netfpga: ~]$ sudo tcpdump -i eth0 -c 5 -nn -q udp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:10:12.648741 IP 192.168.2.10.59398 > 192.168.1.10.1234: UDP, length 6
14:10:13.649745 IP 192.168.2.10.59398 > 192.168.1.10.1234: UDP, length 6
14:10:16.145510 IP 192.168.2.10.59398 > 192.168.1.10.1234: UDP, length 6
14:10:20.441165 IP 192.168.2.10.59398 > 192.168.1.10.1234: UDP, length 6
14:10:29.849363 IP 192.168.2.10.59398 > 192.168.1.10.1234: UDP, length 6
5 packets captured
5 packets received by filter
0 packets dropped by kernel
[1004][netfpga: ~]$
```

OBRÁZOK Č. 15: ROUTER - Test 1

TEST 2 – VZDIALENÁ SIEŤ

Tento test je veľmi podobný predchádzajúcemu testu, od ktorého sa líši len použitou cieľovou IP adresou - 192.168.50.10/24. Táto IP adresa je súčasťou siete (192.168.50.0/24), ktorá sa ale nenachádza priamo za nf0 rozhraním. K tejto sieti sa dostaneme cez fiktívny smerovač, ktorý je pripojený ku karte cez rozhranie **nf0**. Do karty sme pre túto sieť pridali nový smerovací záznam (obr. 18). Keďže smerovač, ktorý predstavuje **next-hop** pre danú sieť, nie je reálne pripojený, musíme do karty pridať nový ARP záznam, ktorý zabezpečí mapovanie next-hop IP adresy na fiktívnu MAC adresu. V inom prípade by karta žiadny z prijatých paketov nesmerovala ďalej z dôvodu neznámej cieľovej MAC adresy (nikto by neodpovedal na ARP žiadosti odosielané kartou). Na obr. 16 sú uvedené odchytené pakety, v ktorých je jasne viditeľná cieľová IP adresa.

```
[1004][netfpga: ~]$ sudo tcpdump -i eth0 -c 5 -nn -q udp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:11:30.305398 IP 192.168.2.10.37453 > 192.168.50.10.1234: UDP, length 39
14:11:31.546986 IP 192.168.2.10.37453 > 192.168.50.10.1234: UDP, length 6
14:11:33.162880 IP 192.168.2.10.37453 > 192.168.50.10.1234: UDP, length 6
14:11:34.034768 IP 192.168.2.10.37453 > 192.168.50.10.1234: UDP, length 6
14:11:35.034709 IP 192.168.2.10.37453 > 192.168.50.10.1234: UDP, length 6
5 packets captured
5 packets received by filter
0 packets dropped by kernel
[1004][netfpga: ~]$
```

OBRÁZOK Č. 16: ROUTER - Test 2

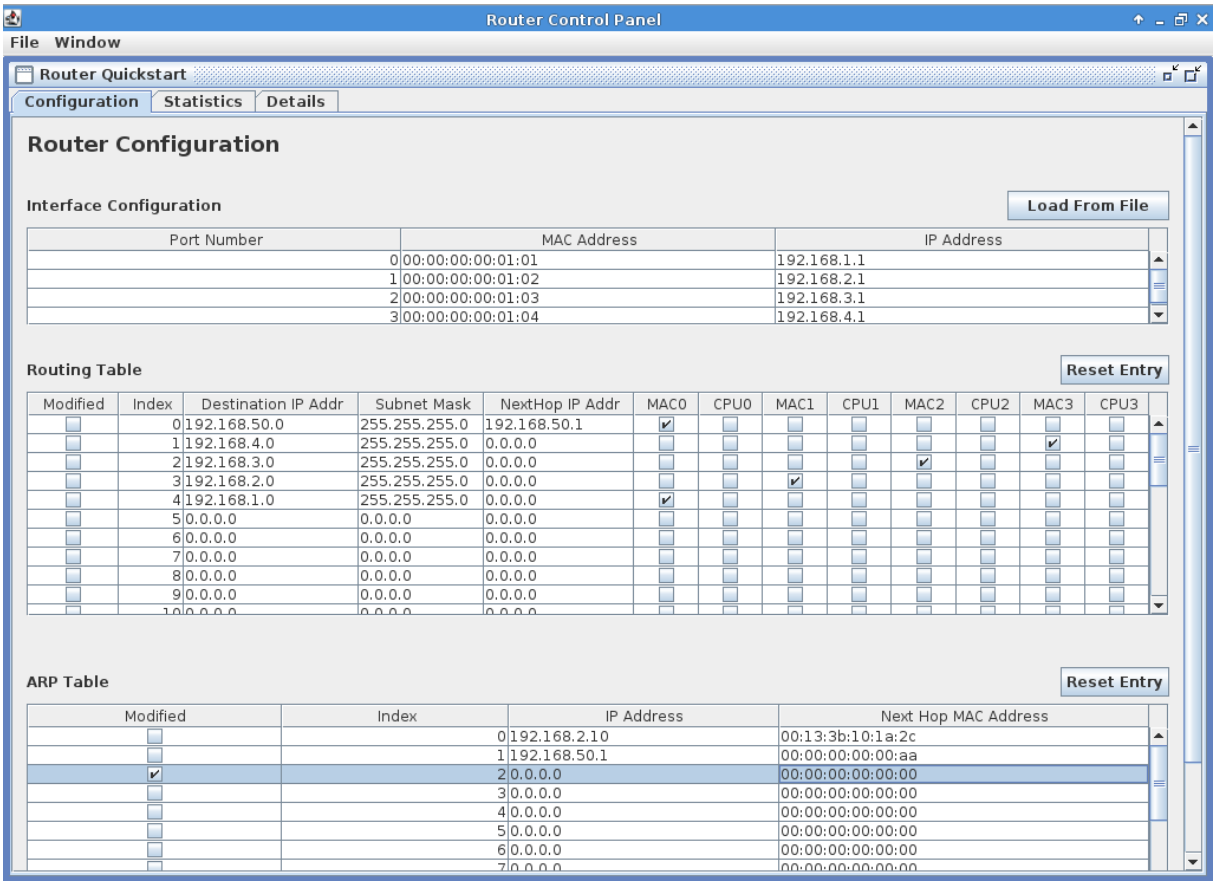
TEST 3 – ZÁPLAVA

V tomto teste sa zameriavame na výkon karty. Nastavenia a parametre (zdrojová a cieľová IP adresa) generovaných paketov sú totožné s predchádzajúcim testom. V tomto prípade sme namiesto nástroja **ncat** použili **hping3**, aby sme formou záplavy generovali veľké množstvo paketov. Použitím nástroja **top** vidíme, že CPU je plne zaťažený nástrojom hping3 (obr. 17), ale karta bez problémov stíha spracovať všetky prichádzajúce pakety (obr. 19 a 20).

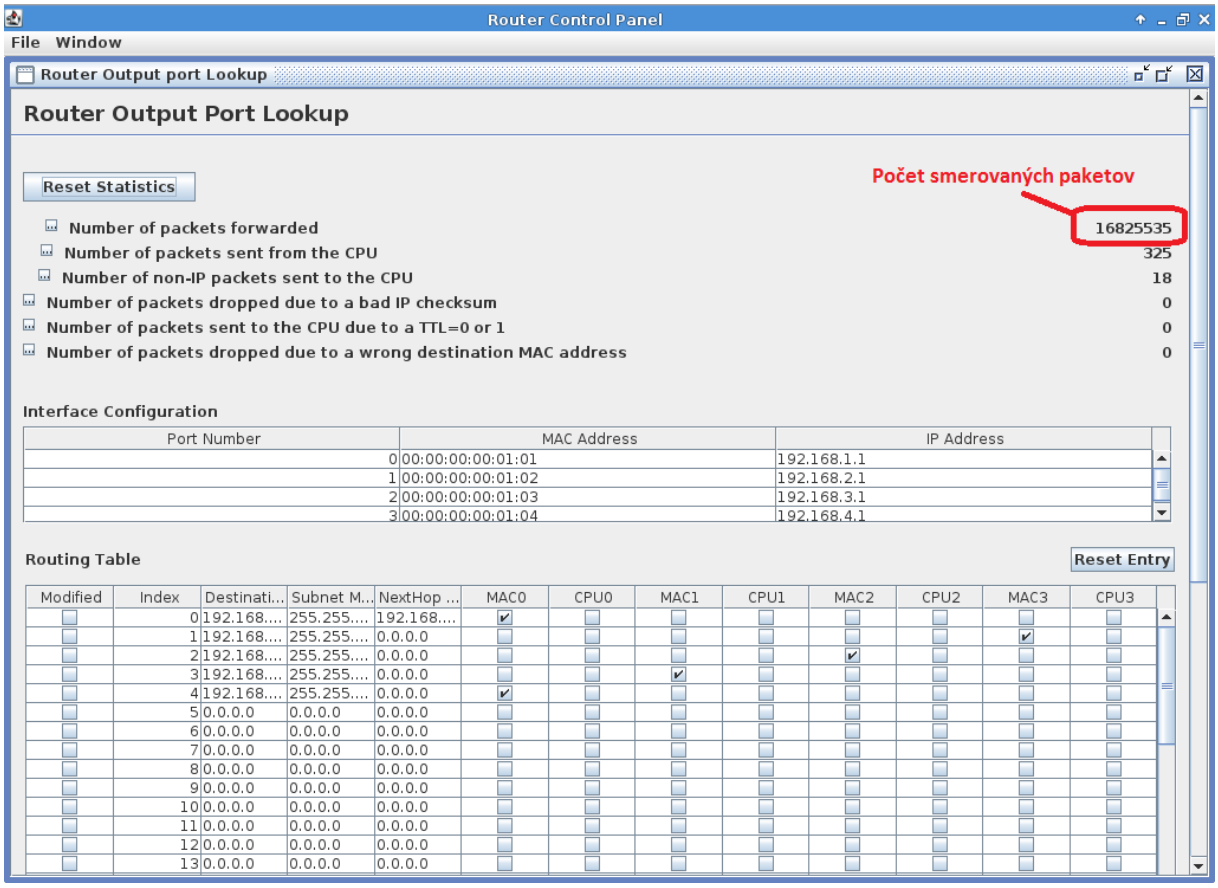
```
top - 10:36:13 up 38 min,  5 users,  load average: 0.47, 0.15, 0.08
Tasks: 222 total,   2 running, 220 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.8 us,  8.8 sy,  0.0 ni, 87.3 id,  0.1 wa,  0.0 hi,  1.0 si,  0.0 st
KiB Mem : 12253200 total, 10899996 free,   519112 used,   834092 buff/cache
KiB Swap:  6160380 total,  6160380 free,        0 used. 11436728 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4117 root      20   0   22644   1424   1224 R  90.4  0.0   0:32.77 hping3
 4002 root      20   0 6396820 153488  20324 S   5.0  1.3   0:26.82 java
 1207 root      20   0  227756  41496  23476 S   4.3  0.3   0:31.94 Xorg.bin
```

OBRÁZOK Č. 17: ROUTER (Top) - Výpis využitia CPU

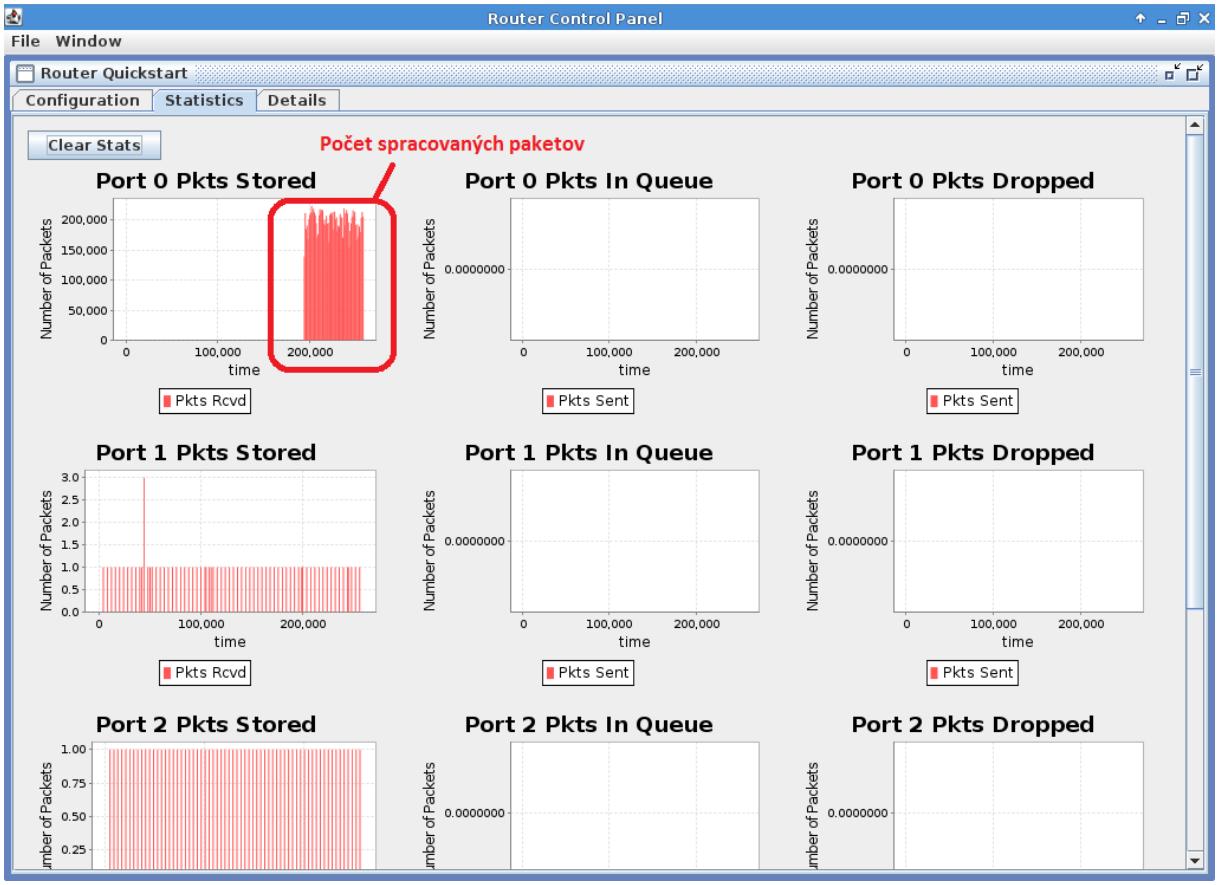
Všetky tri vyššie uvedené testy dopadli úspešne, keďže sme na danom rozhraní odchytili všetky očakávané pakety. Na základe týchto výsledkov môžeme tvrdiť, že správanie systému zodpovedá správaniu smerovača.



OBRÁZOK Č. 18: ROUTER - Konfigurácia karty



OBRÁZOK Č. 20: ROUTER - Štatistiky



OBRÁZOK Č. 19: ROUTER - Grafy

ZHODNOTENIE POUŽITEJ TECHNOLÓGIE

Technológia FPGA obvodov a konkrétne NetFPGA karta, ktorú sme použili pri realizácii jednotlivých systémov, prinášajú do oblasti počítačových sietí nové príležitosti. Pri používaní tejto technológie sme objavili niektoré jej výhody aj nevýhody.

Jednou z hlavných nevýhod je množstvo času, ktorý je potrebný pre vývoj systémov v FPGA. Keďže FPGA obvody sú v oblasti sietí nové, vyžadujú samostatné štúdium. Programátor sa musí oboznámiť s návrhom systémov pre FPGA obvody, ktoré sú dosť odlišné od návrhu softvérových aplikácií – implementácia konkrétnej funkcionality v FPGA obvode je oveľa náročnejšia než jej implementácia formou softvérovej aplikácie. Programátor musí súčasne poznať základné techniky návrhu číslicových systémov, keďže riešenia implementované formou FPGA obvodov sú v konečnom dôsledku porovnateľné s riešeniami realizovanými vo forme špecializovaných obvodov. Prelínajú sa tu tak viaceré oblasti vývoja HW a SW aplikácií - návrh hardvérového systému do FPGA obvodu ako aj softvérovej aplikácie pre jeho riadenie. Pri práci sme potrebovali rôzne softvérové nástroje, z ktorých každý má svoje špecifické prostredie, a teda vyžaduje čas pre jeho pochopenie. Tieto nástroje sú pomerne drahé a vyžadujú rôzne licencie. Avšak po prekonaní počiatočných problémov a získaní požadovaných vedomostí a praktických skúseností sa tieto nevýhody rýchlo vytratia.

Medzi výhody tejto technológie patria najmä rôznorodosť a rýchlosť, s akou tieto systémy v FPGA obvode pracujú v porovnaní s ich softvérovými alternatívami. Vďaka tomu sme schopní pracovať s oveľa väčším tokom dát, čím sa otvárajú nové možnosti, ktoré boli kvôli časovým a objemovým obmedzeniam doposiaľ nerealizovateľné. Pre realizáciu všetkých systémov, ktorým sa v práci venujeme, sme potrebovali len jednu NetFPGA kartu, ktorú sme podľa potrieb rekonfigurovali. Vďaka tomu môžeme neustále zdokonaľovať naše systémy, opätovne ich vkladať do NetFPGA karty a testovať priamo v reálnom prostredí.

V práci sme sa venovali len základným sieťovým zariadeniam. Vzhľadom na úspešnosť ich realizácie plánujeme túto kartu využiť aj v oblasti bezpečnosti počítačových sietí, kde je vzhľadom na vyššie uvedené výhody vhodným kandidátom. Príkladom je sieťová sonda, ktorá zaznamenáva prevádzku v počítačovej sieti a vyhodnocuje ju. Na základe získaných hodnôt tak môžeme rozpoznať prípadné útoky a vykonať protiopatrenia. Vzhľadom na aktuálny nárast využitia tejto technológie vo viacerých oblastiach výskumu môžeme predpokladať, že v blízkej budúcnosti nahradia NetFPGA karty viaceré súčasné riešenia, ktoré sú používané v počítačových sieťach.

ZÁVER

Cieľom diplomovej práce bolo oboznámenie sa s technológiou NetFPGA so zameraním na jej aplikáciu v počítačových sieťach. Keďže táto technológia je v danej oblasti nová, v práci sme sa najprv venovali teoretickému popisu NetFPGA karty a štruktúre jej hlavného prvku – FPGA obvodu. Súčasťou teoretickej časti bola analýza funkcionality a problematika nasadenia tejto technológie v reálnom prostredí, pričom sme uviedli možné využitia NetFPGA karty v oblasti počítačových sietí. Jednou z hlavných úloh, ktoré sme mali riešiť, je realizácia zvolených východiskových projektov. Aby sme boli schopní implementovať tieto projekty, získali sme prístup k NetFPGA repozitáru, kde sa nachádzajú všetky potrebné súbory. Okrem toho sme pripravili a popísali vývojové a testovacie prostredie vhodné pre realizáciu a testovanie zvolených projektov. Súčasťou tohto prostredia sú viaceré nástroje, ktorých funkciám sme sa v práci stručne venovali. Kvôli pochopeniu základných princípov sme podrobnejšie popísali použitú architektúru projektov, pričom sme sa venovali najmä modulu, ktorý je jej základným stavebným prvkom. Časť práce sme venovali podrobnejšiemu popisu rozhrania, cez ktoré modul komunikuje so svojím okolím. Na základe týchto znalostí a podľa jedného z východiskových projektov sme vytvorili vlastný projekt a jeho postup sme v práci krok po kroku rozpísali. Pre zrozumiteľnejšie vysvetlenie jednotlivých operácií sme priložili viaceré obrázky s ich popisom.

Po úspešnom vytvorení vlastného projektu sme sa zamerali na analýzu, realizáciu a testovanie zvolených projektov. Na základe výsledkov, ktoré sme získali počas testovania, môžeme tvrdiť, že implementácia projektov bola úspešná. Žiaľ, vzhľadom na časové obmedzenia a nedostatok vedomostí v danej oblasti sme neboli schopní realizovať projekt zameraný na SDN. Namiesto toho sme do projektu prepínača pridali podporu VLAN, a tak rozšírili jeho funkcionalitu. Počas tejto realizácie sme získali nové znalosti súvisiace s programovaním v jednom z HDL jazykov a hlbšie pochopili princípy návrhu systémov pre NetFPGA kartu.

V práci sme realizovali len východiskové projekty. V budúcnosti by sme chceli dokončiť realizáciu SDN projektu a otestovať jeho funkčnosť v reálnom prostredí. Keďže oblasť FPGA obvodov je pomerne zložitá ale o to zaujímavejšia, jedným z našich cieľov do budúcnosti je prehĺbenie teoretických znalostí a praktických skúseností v tejto oblasti. Preto plánujeme realizovať aj vlastné systémy, ktoré prinesú nové riešenia súčasných problémov v oblasti počítačových sietí, prípadne vylepšia už existujúce riešenia.

ZOZNAM POUŽITEJ LITERATÚRY

[1] C. MAXFIELD, *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*.: Elsevier, Inc., 2004.

[2] A. MOORE, *FPGA for Dummies, Altera Special Edition*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2014.

[3] *NetFPGA-1G-CML Board Reference Manual*.: Digilent, 2014.

[4] NetFPGA-projekt. (2015) NetFPGA-1G-CML-live. [Online]. https://github.com/NetFPGA/NetFPGA-1G-CML-live

[5] NetFPGA-1G-CML-wiki. [Online]. https://github.com/NetFPGA/NetFPGA-public/wiki/Home_NetFPGA-1G-CML

[6] P. P. CHU, *FPGA prototyping by verilog examples*. Hoboken: John Wiley & Sons, Inc., 2008.

[7] *ISE In-Depth Tutorial*.: Xilinx, 2012.

[8] B. KNOLL. www.hunteng.co.uk. [Online]. http://www.hunteng.co.uk/pdfs/manuals/impact.pdf

[9] Xilinx ISE Design Suite. [Online]. http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html

[10] Github - CMLAB. [Online]. https://github.com/cmlab/NetFPGA-1G-CML/wiki/Fixing-Windrvr-6-Error-In-Xilinx

[11] S. CHACON, *Pro Git*. Praha: CZ.NIC, z. s. p. o., 2009.

[12] J. NAOUS, G. GIBB, S. BOLOUKI, and N. MCKEOWN, *NetFPGA: Reusable Router Architecture for Experimental Research*. Seatle: ACM, 2008.

[13] (2011) Xilinx AXI Reference Guide. [Online]. http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

[14] Xilinx Tri-mode Ethernet MAC. [Online]. http://www.xilinx.com/support/documentation/ip_documentation/tri_mode_eth_mac/v5_4/pg051-tri-mode-eth-mac.pdf

[15] Xilinx 7-series PCIe core. [Online]. http://www.xilinx.com/support/documentation/ip_documentation/pcie_7x/v1_6/pg054-

7series-pcie.pdf

[16] K. LOCKE. (2011) Parameterizable Content-Addressable Memory. [Online]. http://www.xilinx.com/support/documentation/application_notes/xapp1151_Param_CAM.pdf

[17] F. BAKER, *RFC 1812 - Requirements for IP Version 4 Routers*., 1995.

ZOZNAM PRÍLOH

PRÍLOHA A: CD MÉDIUM – DIPLOMOVÁ PRÁCA V ELEKTRONICKEJ PODOBE, ZDROJOVÉ SÚBORY UPRAVENÉHO MODULU **SWITCH_OP_VLAN**.