VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AUTOMATICKÁ KONFIGURACE OBSLUŽNÝCH NÁSTROJŮ PRO FPGA FIRMWARE

AUTOMATIC CONFIGURATION OF UTILITY TOOLS FOR FPGA FIRMWARE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN PEREŠÍNI
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KUČERA
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19925/2016/xperes00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Perešíni Martin**
Obor: Informační technologie
Téma: **Automatická konfigurace obslužných nástrojů pro FPGA firmware**
Automatic Configuration of Utility Tools for FPGA Firmware
Kategorie: Vestavěné systémy

Pokyny:

1. Seznamte se s firmwarovými projekty NIC, HANIC a SDM a se systémem DeviceTree.
2. Identifikujte problémy spojené s obsluhou vysoce flexibilních zařízení, zejm. na bázi FPGA.
3. Navrhněte jednotný systém pro správu konfiguračních dat a implementaci obslužných nástrojů pro danou třídu zařízení.
4. Navržený systém implementujte a ověřte jeho funkčnost na nástrojích ve vybraných projektech.
5. Diskutujte dosažené výsledky a možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

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

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

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

Vedoucí: **Kučera Jan, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Táto bakalárska práca sa zaoberá návrhom automatickej konfigurácie obslužných nástrojov pre FPGA firmvér. Zadanie práce je riešené v rámci výskumnej aktivity združenia CESNET, ktoré sa venuje vývoju hardvérovo akcelerovaných sieťových kariet postavených na technológií FPGA. Cieľom práce je náhrada súčasného neflexibilného systému pre popis štruktúry firmvéru, ktorý je použitý združením v projektoch NIC, HANIC a SDM. Pôvodný systém je založený na popise firmvéru samostatným XML súborom, ktorý je vytváraný ručne pre každú konfiguráciu. Na základe dôkladnej analýzy problémov systému bol vytvorený návrh. Následne bol tento systém nahradený Device Tree popisom, ktorý dovoľuje automatické generovanie popisu konfigurácie zmenou v prekladovom systéme platformy NetCOPE. Vygenerovaný popis je priamo distribuovaný spolu s hardvérovou konfiguráciou. Softvérové nástroje pracujúce s firmvérom využijú popis pre prácu s firmvérom. V práci bol návrh systému implementovaný a následne bola úspešne overená jeho funkčnosť na nástroji *ethctl*. Na záver práce sú spomenuté ďalšie možné rozšírenia systému.

Abstract

This bachelor's thesis is about designing an automatic configuration of utility tools for FPGA firmware. The assignment is solved within CESNET research activity, which is devoted on the developement of hardware-accelerated network interface cards based on FPGA technology. The aim of the thesis is to replace current inflexible system for describing the firmware structure used by NIC, HANIC and SDM projects. The system was based on a firmware description by XML file, which was created manually for each configuration. Based on negative aspects of system is created new design, which is using Device Tree. Device Tree is opening possibility to change NetCOPE build system for automatic generating firmware description. Description of harware is distributed together with firmware. In the thesis, the design of the system was implemented and then verified by testing functionality on the ethctl tool. At the end of the work are mentioned possible system features and extensions for future.

Kľúčové slová

sieťové karty COMBO, technológia FPGA, hardvérová akcelerácia, vysokorýchlostné siete, fyzická sieťová vrstva, NetCOPE, HANIC, SDM, 100 Gb/s, Device Tree, knižnica libcombo, nástroj ethctl, kompilátor dtc, knižnica libfdt.

Keywords

COMBO network cards, FPGA technology, hardware acceleration, high-speed networks, Physical network layer, NetCOPE, HANIC, SDM, 100 Gb/s, Device Tree, libcombo library, ethctl tool, dtc compiler, libfdt library.

Citácia

PEREŠÍNI, Martin. *Automatická konfigurace obslužných nástrojů pro FPGA firmware.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kučera

Automatická konfigurace obslužných nástrojů pro FPGA firmware

Prehlásenie

Vyhlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Jana Kučeru. Ďalšie informácie mi poskytol konzultant Ing. Martin Špinler zo združenia CESNET, pracujúceho na projekte Liberouter. Dodatočné informácie a znalosti mi taktiež poskytli ľudia z projektu Liberouter. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

. . . . . .
Martin Perešíni
28. júla 2017

Poďakovanie

Chcel by som sa poďakovať môjmu vedúcemu Ing. Janovi Kučerovi za ochotu, trpezlivosť a ústretovosť pri konzultáciách. Moja veľká vďaka patrí taktiež Ing. Martinovi Špinlerovi za jeho odborné znalosti v danej problematike s ktorými sa s ochotou podelil. Poďakovanie patrí aj celému tímu ľudí z projektu Liberouter. V neposlednej rade poďakovanie patrí taktiež mojej rodine, ktorá ma podporovala počas celého obdobia bakalárskeho štúdia.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

S nárastom veľkého množstva dát, ktoré sa ročne vyprodukujú sa taktiež vyvíjajú nové technológie, ktoré dokážu dáta spracovať efektívne a rýchlo. Nové technológie sa neustále zrýchľujú a hardvérové obvody pre výpočty sú čoraz zložitejšie. Okrem aplikačne špecifických integrovaných obvodov (ASIC) a obecných výpočtových procesorov (CPU) sa v dnešnej dobe ukazuje ako veľmi výhodná aj technológia FPGA. FPGA dokáže poskytnúť vysoký výpočtový výkon, ale taktiež vysokú mieru možnosti rekonfigurácie pre danú aplikáciu. Výkonnostná priepasť medzi ASIC čipmi a FPGA čipmi sa v posledných rokoch zmenšuje a zároveň výroba FPGA začína napredovať vo väčšom počte. FPGA čipy začínajú byť rentabilné a bežne sa nasadzujú do praktických aplikácií, napríklad vysokorýchlostné spracovanie sieťových dát. S nasadením FPGA v aplikáciách ale vzniká problém konfigurácie softvérových nástrojov pre obsluhu premennej štruktúry firmvéru. Softvérové nástroje by mali dokázať reagovať na zmenu konfigurácie automaticky. Mali by sa prispôsobiť danému hardvéru a na základe zmeny vybrať vhodný typ prístupu k daným komponentom karty, alebo vypisovať iné údaje, ktoré sú relevantné pre danú kartu a jej konfiguráciu. Ak by bol vzťah nástroja k firmvéru pevný, neprinieslo by to výhody používania spojené s premennou konfiguráciou hardvéru. Ako príklad môže byť vytvorená nová konfigurácia s komponentami na iných adresách. Nástroj by sa musel zakaždým ručne upraviť pre konkrétne adresy. Takýto nástroj by nepriniesol žiadne uľahčenie práce a vychádzal by zo zlého konceptuálneho návrhu. Preto dáva zmysel vytvoriť určitú abstrakciu (popis štruktúry) firmvéru, ktorá by bola vstupom do obecného nástroja pre obsluhu. Na základe popisu by nástroj dokázal automaticky reagovať zmenou svojich vlastností pre prístup ku komponentom. Samotný nástroj by bol nemenný, ale popis danej konfigurácie by sa dynamicky menil s každou zmenou.

Zadanie práce je riešené v rámci výskumnej aktivity združenia CESNET, v projektovom tíme Liberouter. Cieľom výskumnej skupiny je vývoj hardvérovo akcelerovaných sieťových kariet na báze FPGA, tvorba nástrojov pre hardvérovú akceleráciu a sieťovú bezpečnosť. Vyvíjané sieťové karty sú z rodiny COMBO a podporujú technológiu 10G až 100G Ethernet. Vlastná bakalárska práca sa následne zaoberá návrhom a implementáciou systému pre automatickú konfiguráciu obslužných nástrojov práve pre akceleračné sieťové karty COMBO. Tvorený systém upravuje firmvérové projekty NIC, HANIC a SDM vyvíjané nad týmito kartami. Hlavným cieľom práce je nahradenie súčasného neflexibilného systém pre popis štruktúry firmvéru, založeného na XML, novým systémom založenom na Device Tree popise. Myšlienkou práce je upraviť súčasný prekladový systém platformy NetCOPE, tak aby umožnil automatizáciu generovania Device Tree popisu firmvéru. Vygenerovaný popis by

sa následne spropagoval do softvérových nástrojov, ktoré by z poskytnutého popisu vedeli pracovať s firmvérom.

Práca je rozdelená do viacerých kapitol. V kapitole 2 je popísaný teoretický rozbor danej problematiky. Teoretický rozbor slúži k oboznámeniu čitateľa s technológiou FPGA a platformou NetCOPE. Ďalej sú v kapitole popísané projekty NIC, HANIC a SDM vytvorené nad touto platformou a analýza problémov súčasného systému konfigurácie softvérových nástrojov pre dané projekty. Na záver kapitoly je popísaná Device Tree štruktúra, ktorá sa používa ako existujúce riešenie v linuxovom jadre pre automatické rozpoznanie hardvérovej platformy. Ďalšia kapitola 3 obsahuje postup a návrh riešenia zadaného problému. Na základe analýzy problémov (v teoretickom rozbore) sú určené požiadavky na nový systém a tie sú zakomponované v následnom návrhu. Implementácia vychádzajúca z návrhu sa nachádza v kapitole 4. V samotnej implementácií musel byť zohľadnený celý súčasný systém od prekladu firmvéru (vytvorenie platnej firmvérovej konfigurácie) až po softvérový nástroj pracujúci s firmvérom. Kapitola 5 sa skladá z overenia funkčnosti dosiahnutej implementácie na ukážkovom nástroji *ethctl*. Posledná kapitola 6 je venovaná celkovému zhodnoteniu riešenia, vyzdvihnutiu výsledkov práce a možným vylepšeniam do budúcnosti.

Kapitola 2

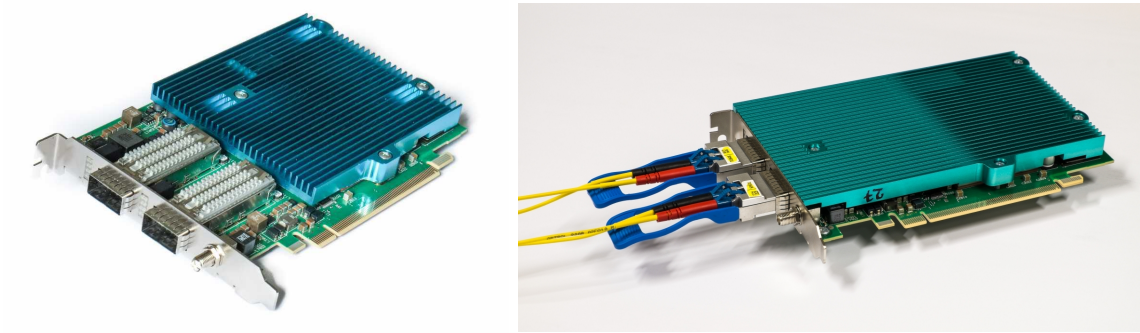
Teoretický rozbor

V tejto kapitole sa nachádzajú teoretické poznatky, ktoré tvoria základ pre praktickú časť bakalárskej práce. Na úvod pre zoznámenie čitateľa s rodinou COMBO kariet, pre ktoré je vyvíjaný systém automatickej konfigurácie primárnym cieľom, je venovaná podkapitola 2.1. Bližší rozbor technológie FPGA, na ktorej sú COMBO karty založené, sa nachádza v podkapitole 2.2. Následne podkapitola 2.3 obsahuje popis platformy NetCOPE a firmvérové projekty NIC, HANIC a SDM postavené nad danou platformou. Podkapitola 2.4 obsahuje informácie o softvérovej časti NetCOPE popisom jednotlivých softvérových vrstiev počínajúc od ovládačov až po vybraný nástroj *ethctl*. Na konci podkapitoly sa nachádzajú poznatky o fyzickej sieťovej vrstve, z dôvodu bližšieho pochopenia problematiky práce súvisiaceho s vybraným nástrojom. Rozbor a analýza problémov spojených s konfiguráciou obslužných nástrojov pre firmvérové komponenty založené na technológií FPGA aplikovaných na platformu NetCOPE je riešená v podkapitole 2.5. Záver samotnej kapitoly, v poslednej podkapitole 2.6, zoznamuje čitateľa s existujúcou štruktúrou pre popis hardvéru Device Tree, ktorá je dnes bežne nasadzovaná v linuxových systémoch.

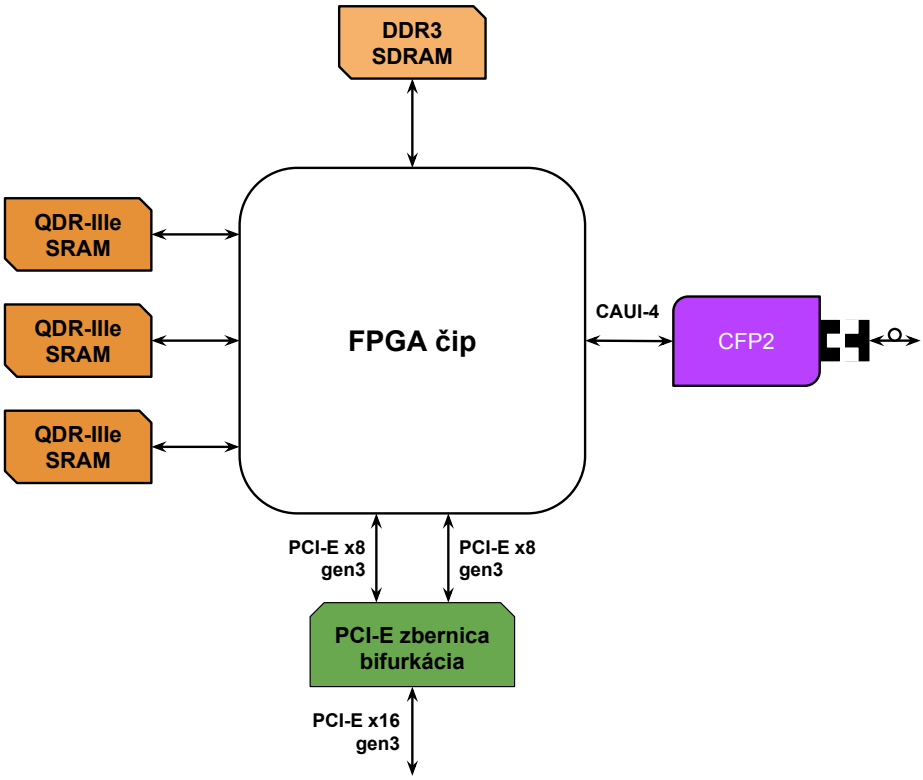
2.1 Rodina kariet COMBO

Karty rodiny COMBO sú prídavné sieťové karty založené na technológií FPGA. Dokážu spracovávať a prenášať dáta až do 100 Gb/s. Hlavnou úlohou kariet je spracovanie sieťových dát na vysoko-rýchlostných linkách v sieťovej infraštruktúre. Aktuálne je vyvíjaná tretia generácia kariet, označená ako COMBOv3. V minulosti existovali aj staršie generácie COMBOv1 a COMBOv2 karty. V súčasnosti sú plne podporované len konfigurácie COMBOv3 kariet, z tohoto dôvodu sa práca bude ďalej zaoberať iba najnovšou generáciou kariet. Pre predstavu čitateľa ako vyzerá vyrobená COMBO karta slúži obrázok 2.1. Vyrobené karty sú pripravené na nasadenie do prevádzky a na obrázku je možné vidieť mohutné pasívne chladenie karty, 2 vstupné porty pre sieťové optické moduly na boku karty a PCI-Express adaptér na spodku karty.

Architektúra kariet generácie COMBOv3, konkrétne karta COMBO-100G1, je znázornená na obrázku 2.2 a vychádza z [3]. Jadrom karty je FPGA čip (v strede obrázku), ku ktorému sú pripojené ďalšie komponenty. Karta sa zasúva do bežného PCI-Express slotu, pre prenos dát medzi kartou a hostiteľským systémom je tak použitá rýchla sériová zbernica PCI-Express. Preto sa na karte nachádzajú celkovo dve PCI-Express rozhrania 3. generácie s 8 dátovými linkami (dole v obrázku). Na karte sú taktiež umiestnené 3 moduly statických pamätí QDR-IIIe (vľavo na obrázku), jedna dynamická pamäť (SDRAM) typu DDR3 (hore



Obr. 2.1: Fotografia COMBO-100G2 a COMBO-100G2Q karty



Obr. 2.2: Schéma COMBO-100G1 karty generácie COMBOv3

v obrázku) a optické sieťové rozhrania (angl. transceiver, znázornené napravo v obrázku). Pre účel zjednodušeného vývoja hardverových akcelerovaných nástrojov postavených nad COMBO kartami, bola vytvorená platforma NetCOPE.

Vytváraný systém konfigurácie obslužných nástrojov pre firmvérové projekty, musí podporovať sieťové karty COMBO tretej generácie s rôznorodými režimami sieťových rozhraní a typmi firmvérových konfigurácií. Preto pre lepšiu predstavu rôznorodosti použitých kariet a ich najpoužívanejších firmvérových konfigurácií sieťových rozhraní slúži nasledujúci súpis.

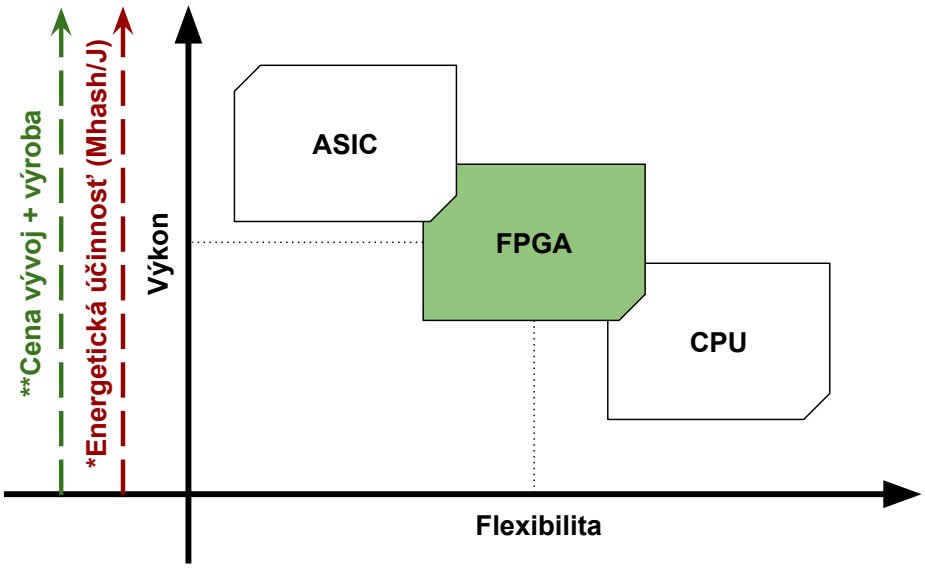
- **COMBO-40G2**, QSFP, 2 porty (1.port | 2.port), dve používané konfigurácie karty pre rýchlosť 80 Gb/s
  - (4x10 | 4x10) – **10GE** (8x10)
  - (4x10 | 1x40)
  - (1x40 | 1x40) – **40GE** (2x40)
  - (1x40 | 4x10)
- **COMBO-100G2**, CFP4, 2 porty (1.port | 2.port), rôzne konfigurácie
  - (1x100 | none)
  - (1x100 | 1x100)
  - (4x 10 | none)
- **COMBO-100G1**, CFP2, 1 port, tri konfigurácie karty pre rýchlosť 100 Gb/s
  - **10GE**, 10x10 Gb/s (nezávislé 10G linky)
  - **100GE LR4**, 4x25 Gb/s
  - **100GE SR10**, 10x10 Gb/s (spojenie liniek do 100G)
- **COMBO-100G2Q**, QSFP, 2 porty (1.port | 2.port), rôzne konfigurácie
  - (1x100 | none)
  - (1x100 | 4x 10)
  - (4x 10 | none)
  - (4x 10 | 4x 10)

V súpise sú zahrnuté len najpoužívanejšie platné konfigurácie. Súčasne sú vyvíjané štyri karty, jedna jednoportová a ostatné dvojportové. Najzložitejšie kombinácie rozhraní majú COMBO-100G2 a COMBO-100G2Q. Zo zoznamu je vidieť, že každý port môže byť nakonfigurovaný buď ako 4x10G alebo 100G. Rozdiel medzi 100G2 a 100G2Q je v použití rozdielnych optických modulov CFP4 alebo QSFP a v hardvérovej konfigurácií možného počtu sieťových liniek. V prípade 100G2 sú maximálne podporované len 4 linky a preto nie je možné napríklad nakonfigurovať 1x100G+4x10G ktorá by obsahovala 5 liniek. Z teoretického hľadiska nič nezabraňuje implementovať napríklad konfiguráciu, ktorá by používala 3x10G+1x100G, ale v praxi táto konfigurácia ešte nebola implementovaná (nebol po nej dopyt). Taktiež je možné porty mať v konfigurácií nie ako 10G, 100G ale napríklad ako 25G alebo 50G (pri použití správnych sieťových modulov). Tým pádom sa celkové možné kombinácie portov zväčšujú. Z praktického hľadiska sú ale používané iba určité platné kombinácie.

Pre zmenu režimu sieťového portu je potrebné typicky vymeniť optický modul nachádzajúci sa na karte (transceiver) a rekofigurovať FPGA. Rekonfigurácia karty prebieha načítaním uloženej konfigurácie z flash pamäte karty alebo je karta priamo preprogramovaná počas behu pomocou JTAG rozhrania. FPGA čip ktorý je umiestnený na karte umožňuje túto zmenu funkcionality karty. Technológiou FPGA sa detailnejšie zaoberá následujúca podkapitola.

2.2 Technológia FPGA

Programovateľné hradlové polia, označované skratkou FPGA (z angl. Field Programmable Gate Array), sú typom logických integrovaných obvodov. Z praktického hľadiska je možné s ich pomocou vytvoriť ľubovoľný digitálny obvod. Technológia FPGA vznikla ako kompromis medzi drahými ASIC čipmi a nevýkonnými CPU procesormi. V grafe 2.3 vidíme porovnanie technológie FPGA s inými technológiami. Graf je zameraný na porovnanie medzi výkonom (zvislá Y os) a flexibilitou (vodorovná X os) daných technológií. Okrajovo sú

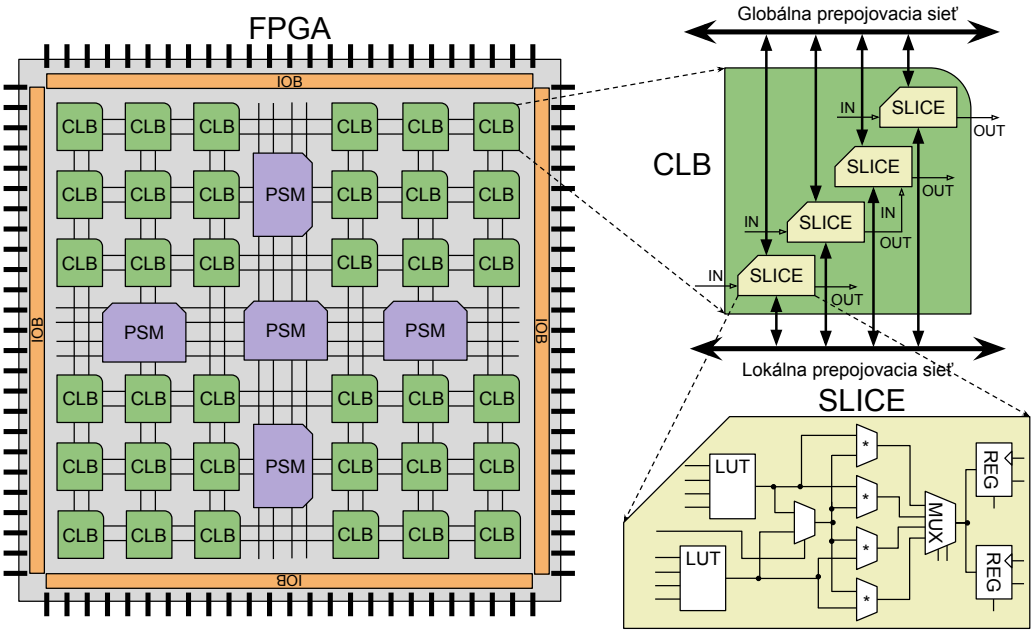


Obr. 2.3: Porovnanie FPGA voči ostatným technológiám, [10]

znázornené aj parametre ako energetická účinnosť a celková cena vývoja a výroby danej technológie.

Z grafu môžeme vyčítať, že ASIC (z angl. Application Specific Integrated Circuits) čipy sú najmenej flexibilnejšie, tzn. pri výrobe a vývoji sú navrhnuté na jednu špecifickú úlohu (aplikáciu) a následne, už nie je možné rozširovať alebo modifikovať aplikačnú triedu obvodu. Vyplýva to z faktu, že pri zmene alebo oprave danej aplikácie je treba celý hardvérový čip znova navrhnúť a vyrobiť. Tým pádom celý vývoj a výroba daného čipu vyžaduje nesmierne úsilie a finančné náklady a celá technológia sa stáva drahá. Výhodou tejto technológie je vykonávať výpočty oveľa rýchlejšie ako obdobné softvérové riešenia na obecných procesoroch. S časti je to spôsobené aj masívnou paralelizáciou výpočtu.

Obecné procesory (CPU, z angl. Central Processing Unit) sú opakom ASIC čipov a majú najvyššiu možnú flexibilitu. CPU je hardvérový obvod ktorý spracúva programy, ktoré sú zložené z elementárnych strojových inštrukcií. Inštrukcie programu sú usporiadané sekvenčne za sebou a sú vykonávané procesorom postupne jedna za druhou. Procesor dokáže spracovať rozdielne programy. Programy sa všeobecne odlišujú, nie sú špecificky zamerané na tú istú aplikačnú úlohu. Procesor je tým pádom generický nástroj pre rôzne druhy výpočtov. Nevýhodou tohto riešenia je nízky výpočtový výkon. Komplexnejšie programy musia byť zapísané väčším počtom inštrukcií a samotný procesor dokáže vykonávať jednotlivé inštrukcie len krok za krokom. Rýchlosť spracovania inštrukcií je udávaná v jednotkách IPS (z angl. Instructions per second). Jedným z parametrov udávajúcich IPS je frekvencia na ktorej procesor pracuje. V súčasnosti nie je tvrdenie, že moderné procesory vykonávajú inštrukcie sekvenčne za sebou, úplne pravdivé. V skutočnosti sú inštrukcie zreťazené a vykonávajú sa paralelne. Dnešné moderné procesory už nie sú postavené na samostatnej SISD architektúre (z angl. Single Instruction stream Single Data stream), ale ako hybrid skalárnej alebo superskalárnej paralelnej architektúry (SIMD prípadne MIMD architektúra, podľa



Obr. 2.4: Hierarchická štruktúra FPGA obvodu

Flynnovej klasifikácie paralelných systémov). To znamená spracovanie viac rozpracovaných inštrukcií zároveň a dokončenie viacerých inštrukcií v jednom takte procesora. CPU sa vyrábajú v sériách veľkých počtov kusov, čo prispieva k znižovaniu ceny výrobných nákladov. Navyše sa procesory nachádzajú vo veľkom počte zariadení každodenného života a dopyt po nich je veľký, ceny sú nižšie oproti komplikovanému návrhu a výrobe ASIC čipov.

ASIC obvody sú náchylné na chyby v procese návrhu a ich oprava je po výrobe takmer nemožná. Procesory sú výkonovo príliš „pomalé“, stále ide o sekvenčné spracovávanie. Preto na trhu existuje ešte ďalšie možné riešenie v podobe FPGA. FPGA je kompromis medzi flexibilitou, ktorú preberajú zo softvérového riešenia (CPU) a výkonom blízkemu ASIC obvodom. Ako už bolo spomenuté FPGA sú rekonfigurovateľné hradlové polia a môžu byť užívateľom naprogramované tak, aby realizovali ľubovolné užívateľské aplikácie. Nesmiernou výhodou je jednoduchšie a rýchlejšie prototypovanie užívateľských aplikácií, z čoho vyplýva, že oprava chýb sa odstraňuje s menším úsilím ako pri ASIC čipoch. Taktiež sa do užívateľskej aplikácie môže jednoducho pridávať nová funkcionalita a to rekonfiguráciou obvodu. Nie je nutné vymieňať celý nový čip. Výkonnostný rozdiel medzi ASIC a FPGA sa znižuje. FPGA v posledných rokoch napreduje z technologického hľadiska. Celkový výkon záleží od počtu dostupných zdrojov (počet LUT a logických blokov) nachádzajúcich sa na čipe a od rýchlosti daného čipu.

Obrázok 2.4 zobrazuje architektúru FPGA čipu. Základ štruktúry FPGA (ľavá časť obrázku) je tvorený programovateľnou maticou PSM (z angl. Programmable Switch Matrix) a konfigurovateľnými logickými blokmi CLB (z angl. Configurable Logical Block). Vstup a výstup do FPGA majú na starosti IOB (Input Output Block) bloky. PSM matica slúži na rôzne kombinácie prepojenia CLB blokov. CLB bloky sú tvorené viacerými menšími blokmi, ktoré sú označované ako *slices* (horná pravá časť obrázku). Prepojenie jednotlivých

slice je taktiež rekonfigurovateľné pomocou prepojovacej matice. Samotný *slice* je tvorený viacerými hardvérovými funkčnými obvodmi. Tieto obvody sú logické hradlá, multiplexory, registre, LUT (z angl. Look-Up Table) a iné konfigurovateľné súčasti. LUT sú realizované ako n-vstupé vyhľadávacie tabuľky a tvoria základnú jednotku kombinačnej logiky. Viac o LUT sa dá dočítať v [2] v sekcií Building Look-up Tables.

Okrem spomínaných CLB blokov a IOB blokov sa na obvode FPGA nachádzajú aj samostatné obvody rozvodu hodinového a resetovacieho signálu, pre správne zaistenie funkčnosti či už synchrónnych alebo asynchrónnych sekvenčných obvodov. FPGA taktiež obsahuje blokové pamäte nazývané BRAM (z angl. Block Random Access Memory). FPGA obsahuje aj komponenty DSP (z angl. Digital Signal Processing) čo sú jednotky pre spracovanie signálov a aj rôzne ďalšie pomocné jednotky ako napríklad rýchle sériové spoje PCI-Express. Celý popis architektúry FPGA, ktorý bol prezentovaný vychádza z [14].

V dnešnej dobe sa pre návrh užívateľských aplikácií nad FPGA využívajú špecializované jazyky, ktoré slúžia na popis hardvéru. Tieto jazyky sa radia do samostatnej skupiny programovacích jazykov HDL (z angl. Hardware Description Language). Návrh užívateľskej aplikácie končí vytvorením konfigurácie FPGA. Zisk konfigurácie je podmienený úspešnou syntézou a implementáciou samotného popisu v programovacom jazyku. Samotná konfigurácia FPGA, ktorá určuje funkciu každého logického bloku, je nazývaná bitstream.

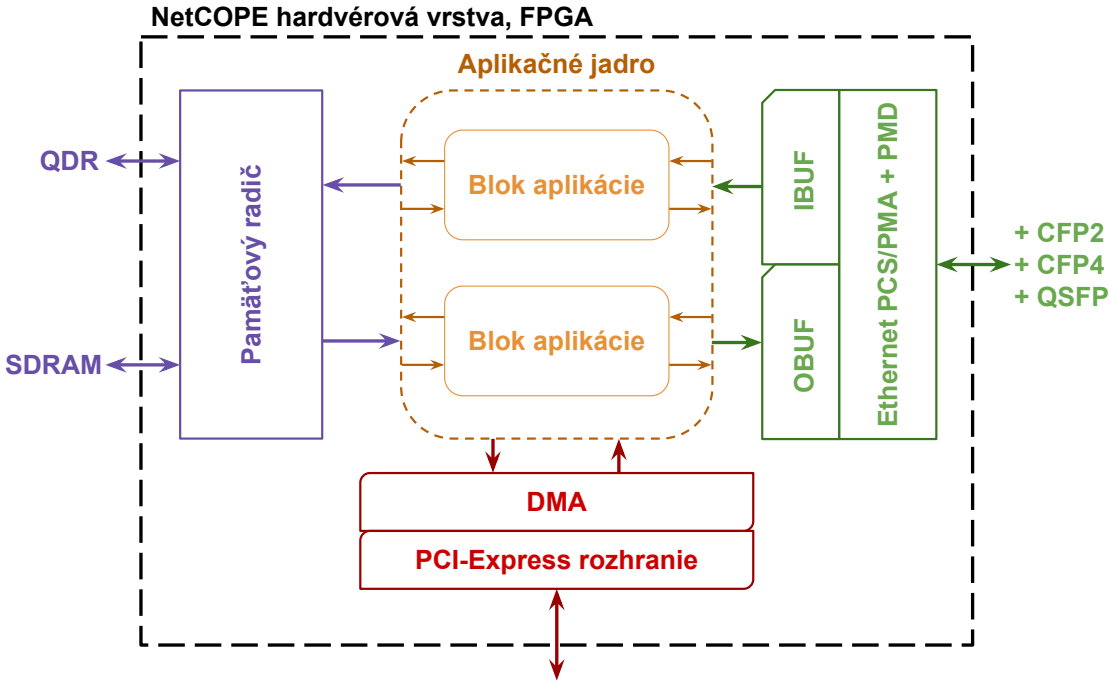
Pre užívateľa je samostatný FPGA čip s užívateľskou aplikáciou v mnohých prípadoch nevyužiteľný. Preto sa vo väčšine prípadoch FPGA čip navrhne pre zostavenie v podobe karty, ktorá sa vloží do hostiteľského systému, aby samostatný procesor mohol spolupracovať s užívateľskou aplikáciou FPGA čipu. Jedným zo spôsobov zjednodušeného návrhu číslicových obvodov je využitie tzv. IP cores, knižnice s hotovými komponentami. Príkladom takéhoto prístupu môže byť platforma NetCOPE, ktorá okrem infraštruktúry a predpripravených komponentov pre vývoj aplikačného jadra pre COMBO karty, poskytuje aj softvérové nástroje pre komunikáciu s firmvérom FPGA. Platforma NetCOPE je popísaná v nasledujúcej podkapitole.

2.3 NetCOPE – firmvérové projekty

NetCOPE je vysoko výkonná škálovateľná platforma pre rýchly vývoj sieťových aplikácií použiteľných na akceleračných kartách využívajúcich technológiou FPGA. NetCOPE ako celok poskytuje vývojové prostredie s radou firmvérových ale aj softvérových knižníc, nástrojov a komponentov. Platforma bola pôvodne vyvinutá združením CESNET pre rodinu COMBO kariet [11], ale taktiež bola úspešne portovaná na akceleračné karty NetFPGA-10G vyvíjané na Standford University [7].

Hlavným účelom NetCOPE je uľahčiť prácu vývojárovi pri tvorbe samotnej aplikácie a odbremeniť vývojára od nízko-úrovňovej problematiky práce s jednotlivými hardvérovými prvkami. Tieto prvky sú v platforme implementované ako samostatné komponenty a NetCOPE poskytuje jednotne komunikačné rozhranie medzi danými komponentami. Tým pádom sa vývojár nemusí sústrediť na implementáciu komunikačných rozhraní pre prenos dát medzi aplikáciou a infraštruktúrou NetCOPE. Navyše platforma NetCOPE využíva rovnaké komunikačné rozhrania na rôznych akceleračných kartách a umožňuje prenositeľnosť užívateľskej aplikácie.

Platforma je zložená z softvérovej a hardvérovej (firmvér) časti. Na obrázku 2.5 je zobrazená modulárna hardvérová časť. Komunikácia s užívateľskou aplikáciou v rámci NetCOPE je založená na protokoloch s rozhraním MI32 a FLU (z angl. FrameLink Unaligned). Hlavné komponenty z ktorých je platforma NetCOPE zložená je pamäťový radič na ktorý je pripo-



Obr. 2.5: Platforma NetCOPE

jená externá pamäť typu QDR (z angl. Quad Data Rate) alebo klasická SDRAM (z angl. Synchronous Dynamic Random Access Memory). Ďalej sa v platforme nachádza radič priameho prístupu do operačnej pamäte DMA (z angl. Direct Memory Access), používaný na rýchle prenosy dát medzi pamäťou počítača, spolu s kruhovými vyrovnávacími pamäťami. Komunikácia a prenos dát medzi FPGA a samotným procesorom je zabezpečená prostredníctvom PCI-Express zbernice. Dôležitými prvkami architektúry sú aj vstupné a výstupné sieťové bloky slúžiace na príjem a vysielanie dát cez sieťové Ethernet rozhranie. Navyše sú dostupné aj podporné komponenty pre identifikáciu, monitorovanie teploty, monitorovanie napájacieho napätia a jednotky pristupujúce k stavovým a konfiguračným registrom danej karty.

Základná myšlienka platformy je taktiež poskytnúť predpripravené základné komponenty pre tvorbu užívateľskej aplikácie. Implementované sú napríklad pamäte CAM (z angl. Content-Addressable Memory), pamäte FIFO (z angl. First In First Out), Barrel shifter jednotka pre posun dát v pamäti, komponenty pre prácu s viacerými komunikačnými protokolmi, ale aj komponenty k riadeniu prvkov mimo FPGA, či modul pre príjem časových značiek zo systému GPS alebo z interných hodín. Platforma obsahuje aj komponenty pre spracovanie sieťovej prevádzky ako napríklad výpočty a verifikácia CRC kontrolných súčtov alebo získavanie hlavičiek prichádzajúcich sieťových paketov. Vytvárané projekty pre rodinu COMBO kariet sú založené na 3 základných pilieroch: sieťová vrstva, DMA prenosy a samotné aplikačné jadro.

Sieťové rozhranie je tvorené fyzickou sieťovou vrstvou PHY (z angl. Physical Layer). Vrstva je delená na podvrstvy PCS/PMA (z angl. Physical Coding Sublayer/Physical Medium Attachment) a PMD (z angl. Physical Medium Dependent). Nad vrstvou PCS/PMA sa nachádzajú vstupné sieťové vyrovnávacie pamäte (IBUF, Input BUF-

fer) a výstupné sieťové vyrovnávacie pamäte (OBUF, Output BUFfer), táto vrstva sa v NetCOPE nazýva MAC vrstvou. Tieto vyrovnávacie pamäte sú prístupné z aplikačného jadra rozhraním FLU. Rozhranie má prístup k CGMII (100 Gigabit Media Independent Interface), XLGMII (40 Gigabit Media Independent Interface) alebo XGMII (10 Gigabit Media Independent Interface). Samotné rozhrania sú definované v sieťovom štandarde IEEE 802.3. Rozhrania by mali mať nezávislé hodiny pre vysielanie (TX) a príjem (RX). V NetCOPE platforme sa všetky dané rozhrania modulov nachádzajú pod *network_mod* a *eth_phy* komponentov. Komponent definuje sieťový modul príslušnej konfigurácie a to PMD, PCS/PMA a MAC vrstvu.

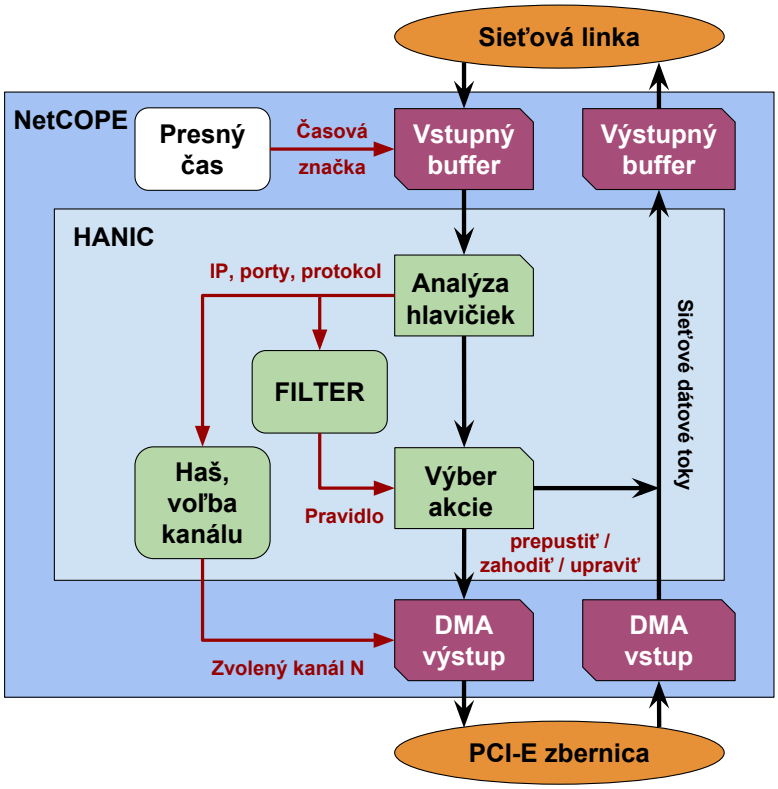
DMA prenosy medzi aplikačným jadrom hardvéru a hostiteľskou RAM pamäťou reprezentujú jednu z najdôležitejších vlastností NetCOPE platformy. Vďaka neustálemu zlepšovaniu rýchlych prenosov dát medzi aplikačným jadrom a RAM je možné v dnešnej dobe v určitých situáciách dosiahnuť 100 Gb/s paketové prenosy do procesora hostiteľského systému. Implementácia DMA je postavená na prenosoch medzi dvoma kruhovými zásobníkmi. Jeden zásobník je v hostiteľskej RAM pamäti a je používaný pre prístup softvérovej aplikácie, prípadne vlákien aplikácie. Druhý zásobník sa nachádza na FPGA čipe a sprístupňuje dáta pre aplikačné jadro. Existujú 3 základné softvérové rozhrania pre rýchle DMA prenosy v platforme NetCOPE. Jeden spôsob je použitie libsze2 knižnice pomocou jednoduchého aplikačného rozhrania. Iný spôsob môže byť využitím DPDK (z angl. Data Plane Development Kit, [15]), alebo použitím upravenej libpcap knižnice pre podporu PCAP.

Aplikačné jadro platformy je užívateľská aplikácia v ktorej je implementované pokročilé spracovanie prichádzajúcich alebo odchádzajúcich sieťových dát. Na rozdiel od sieťového rozhrania a DMA prenosov, ktoré sú súčasťou NetCOPE platformy, aplikačné jadro sa pre každý projekt líši. Spracovanie môže byť vo forme extrakcie dát (predspracovanie hlavičiek protokolov), filtrácia na základe zadaných pravidiel, výpočet štatistík sieťovej prevádzky, bežný IPv4/IPv6 smerovač, generátor sieťovej prevádzky, kryptografický nástroj a iné. Pokiaľ sa aplikačné jadro v platforme NetCOPE nevyužije vznikne najjednoduchší projekt vo forme NIC (z angl. Network Interface Controller). NIC dokáže prijímať a odosielať dáta ako každá bežná sieťová karta, avšak na väčších dátových tokoch, až do 100 Gb/s. Ďalšie projekty s vlastným užívateľským jadrom sú napríklad HANIC a SDM, detailnejšie popísané v nasledujúcich podkapitolách.

Súčasť NetCOPE platformy je taktiež prekladový systém. Prekladový systém slúži k automatizácií práce so syntézou a simuláciou projektov vytvorených z NetCOPE. Preklad využíva skripty napísane v jazyku Tcl a súbory Makefile, ktoré obsahujú pravidlá pre tvorbu automatického zostavenia projektu.

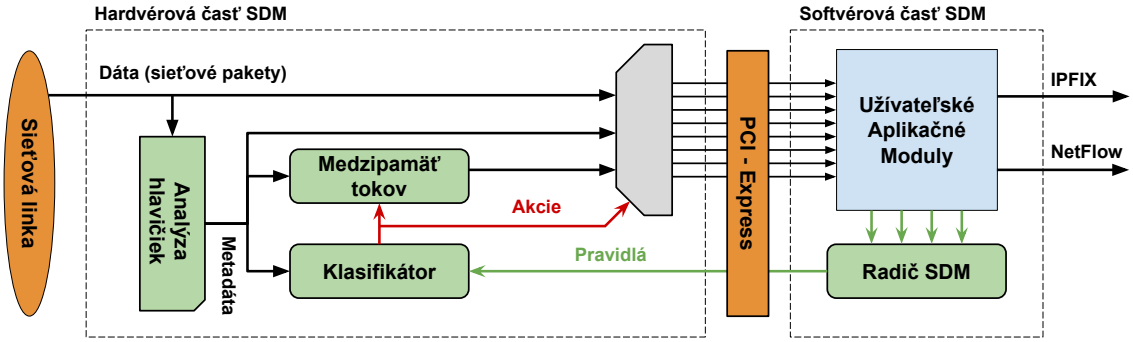
2.3.1 HANIC

Názov HANIC predstavuje skratku celého názvu firmvérového jadra pre COMBO karty. Názov projektu v celom znení je hardvérovo akcelerovaná sieťová karta (Hardware Accelerated Network Interface Card) alebo hašovacia sieťová karta (HAshing Network Interface Card), [13]. Rozširuje klasickú funkcionalitu NIC platformy nad COMBO kartami. Základná schéma HANIC platformy je zobrazená v obrázku 2.6. HANIC podporuje spracovávanie paketov v plnej rýchlosti pripojených sieťových liniek (horná časť obrázku) a následne prenos paketov do softvéru cez PCI-Express zbernicu pomocou rýchlych DMA prenosov (dolná



Obr. 2.6: Základná schéma HANIC štruktúry

časť obrázku) s tým že jeho hlavná funkcionalita spočíva v schopnosti deliť vstupný tok dát medzi niekoľko softvérových rozhraní. Každé softvérové rozhranie je spracovávané na samostatnom procesorovom jadre. Záťaž sa efektívne rozdelí úmerne počtu jadier procesora. Táto funkcionalita je zabezpečená pomocou implementácie rozdeľovania prichádzajúcich paketov na základe ich haš hodnoty (angl. hash), ktorá je vypočítaná z položiek hlavičky paketov (stred obrázku, označené ako analýza hlavičiek). Distribúcia na základe haše paketov prináša výhodu spracovania rovnakého sieťového toku tým istým jadrom procesora. Pakety s rovnakou haš hodnotou sú zaradené do rovnakého kanálu (stred obrázku, označené ako voľba kanálu). HANIC podporuje aj filtráciu (v obrázku blok filter) na základe pravidiel, podmienky za ktorých sa má paket preposlať do softvéru. Filter udáva akcie napríklad na prepustenie daných paketov alebo zahadzovanie. Pravidlá určujú z ktorých položiek v hlavičke paketu sa majú vyberať hodnoty do výpočtu. Položky z ktorých sa môže vyberať sú napríklad verzia IP protokolu (IPv4 alebo IPv6), zdrojová a cieľová IP adresa, protokol transportnej vrstvy (TCP, UDP), číslo zdrojového a cieľového portu transportného protokolu a iné. Okrem hlavnej funkcionality HANIC taktiež podporuje vzorkovanie spracovávaných paketov, každý i-tý paket je prepustený do softvérového jadra. V rámci NetCOPE platformy HANIC dokáže taktiež pridávať časové pečiatky (angl. timestamps) prichádzajúcim paketom s presnosťou na nanosekundy (v obrázku komponent presný čas). Softvérová časť HANIC platformy obsahuje nástroje a skripty pre ovládanie, nahrávanie pravidiel do filtra a nástroje pre konfiguráciu hašovania a vzorkovania dát.

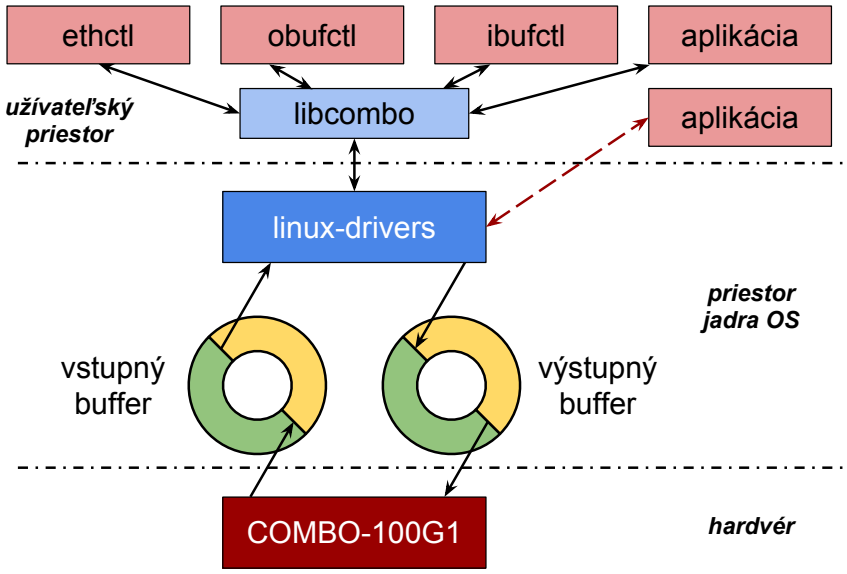


Obr. 2.7: Základná schéma funkcionality SDM systému

2.3.2 SDM

Softvérovo definované monitorovanie siete, skratka SDM (z angl. Software Defined Monitoring) je koncept softvérového riadenia spracovávania sieťových dát. Celá časť venovaná SDM vychádza z bakalárskej práce [8] a diplomovej práce [6]. Pri vyšších dátových prenosových rýchlostiach nad 10 Gb/s cez sieť zväčša dáta nestíhajú byť včasne spracované softvérom (pokiaľ sa jedná o analýzu aplikačných protokolov vyšších ako 4 vrstva referenčného modelu ISO/OSI, protokoly ako HTTP, DNS a iné). Pokiaľ nie sú dáta spracované nemôže byť zachytená detekcia incidentu, pretože systém nestíha, a následne vykonaná príslušná akcia súvisiaca s incidentom. Tento fakt vzniká kvôli nedostatočnému výpočtovému výkonu procesora na vykonanie potrebných výpočtov a taktiež kvôli nedostatočnej priepustnosti zbernice medzi sieťovou kartou a samotným procesorom. Z tohoto dôvodu vznikol projekt SDM nad COMBO kartami s myšlienkou hardvérového predspracovania prichádzajúcich sieťových dát, ktoré budú riadené softvérom. Výhodou harvérového spracovania dát je zníženie toku prenosov cez systémovú zbernicu, prenášajú sa len užitočné dáta. Softvérom sú vykonávané komplexnejšie úlohy rozhodovania a spracovania dát, ktoré by nebolo možné realizovať v hardvéri karty.

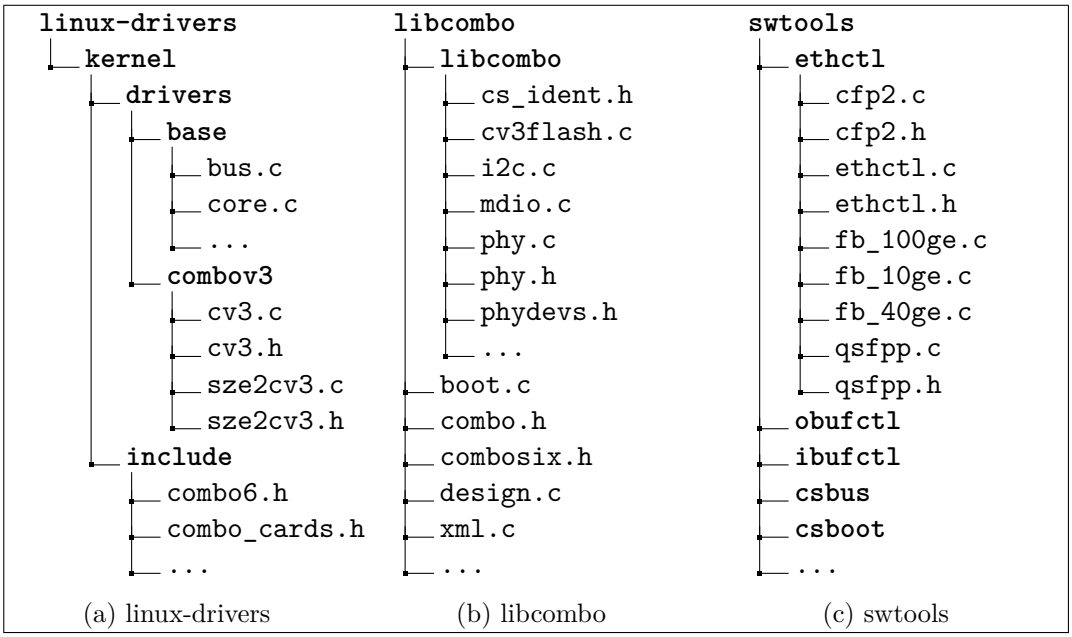
Obrázok 2.7 zobrazuje základnú schému konceptu SDM. Koncept sa skladá z hardvérovej a softvérovej časti. Tieto dve časti sú prepojené PCI-Express zbernicou, slúžiacou ako komunikačný kanál. Hardvérová časť (COMBO karta) sa skladá z analyzátora hlavičiek sieťových paketov (modul vľavo na obrázku), medzipamäťou sieťových tokov a klasifikátorom dát (v strede obrázku). Analyzátor extrahuje potrebné dáta (hlavičky) zo sieťových paketov a získané metadáta preposiela do medzipamäte a klasifikátora. Extrahované dáta slúžia ku klasifikácií paketov na základe pravidiel. Pravidlá sú definované radičom SDM (na obrázku dole vpravo) ktorý je riadený softvérom užívateľských aplikácií. Každé pravidlo určuje, aké akcie majú byť vykonané s prichádzajúcimi paketmi rozpoznaného toku. Prenos dát medzi hardvérom do softvéru môže byť vo forme prijatých paketov bez zmeny alebo vo forme extrahovaných dát. SDM rovnako ako HANIC umožnuje vzorkovanie dát podľa paketov alebo tokov (flow). Taktiež ako HANIC aj SDM obsahuje softvérové moduly, skripty a nástroje na konfiguráciu hašovania a vzorkovania dát alebo nástoje na vyčítanie štatistiky prenosov.



Obr. 2.8: Zobrazenie NetCOPE softvérovej architektúry

2.4 NetCOPE – softvérová architektúra

Ako bolo spomenuté, platforma NetCOPE sa taktiež skladá zo softvérovej časti. Príklad všeobecného konceptu softvérovej časti a znázornenie súčastí pre konkrétnu užívateľskú aplikáciu je zobrazené v obrázku 2.8. Hardvérová časť (v obrázku dole) je niektorá z rodiny COMBO kariet, ktoré sú popísané v samostatnej sekcií 2.1 a 2.3. Karta je zasunutá v PCI-Express slote v hostiteľskom systéme. Nad hardvérovou vrstvou začína samotná softvérová vrstva, ktorá slúži na prenášanie dát medzi kartou a hostiteľským systémom. Na prenos dát slúžia vstupné a výstupné vyrovnávacie pamäte. Ovládač operačného systému má priamy prístup do adresného priestoru vyrovnávacích pamätí a vytvára jednotné aplikačné rozhranie pre užívateľské aplikácie, ktoré potrebujú prenášať dáta z karty. V NetCOPE platforme tvorí ovládač operačného systému pre podporu COMBO kariet balíček *linux-drivers*. Medzivrstva medzi samotnou aplikáciou a ovládačom karty je tvorená knižnicou *libcombo*. Knižnica poskytuje rovnaké rozhranie všetkým aplikáciám pre prístup ku karte. Nevylučuje sa však aj použitie aplikácie, ktorá komunikuje priamo s ovládačom samostatne bez prídavnej vrstvy. Všetky užívateľské nástroje na zisk prevádzkových parametrov karty, pre konfiguráciu a správu karty a prenos dát medzi kartou a systémom sú obsiahnuté v NetCOPE balíčku *swtools*. Obrázok 2.9 zobrazuje zjednodušenú štruktúru súborov a priečinkov nachádzajúcich sa v softvérovej časti NetCOPE v balíčkoch *linux-drivers*, *libcombo* a *swtools*. Priečinok `./swtools` obsahuje napríklad nástroje pre správu sieťových vyrovnávacích pamätí *ibufctl* a *obufctl* (nástroje zobrazujú stavové informácie o pamätiach ako napríklad stav povolený alebo zakázaný, rýchlosť a štatistiky ako počet prepustených, zahodených alebo porušených paketov), alebo nástroj *csboot* na uloženie dizajnu do flash pamäte karty pre následnú rekonfiguráciu karty, všeobecný nástroj *csbus* pre priamy prístup ku karte a zisku dát na príslušnej adrese, nástroj *csid* pre identifikáciu karty a použitého dizajnu na hostiteľskom systéme a iné nástroje. Medzi nástrojmi sa nachádza aj nástroj *ethctl*, ktorý bol v rámci tejto práce upravovaný. Viac o nástroji je v samostatnej podkapitole 2.4.3.



(a) linux-drivers (b) libcombo (c) swtools

Obr. 2.9: Hierarchická štruktúra balíčkov softvérovej časti NetCOPE platformy

2.4.1 Ovládač linux-drivers

Na obrázku 2.9a je možné vidieť názornú ukážku adresárovej štruktúry ovládačov vyvíjaných pre COMBO karty pod Linux operačným systémom. Ovládače sú kompatibilné s Linux kernel verziou 2.6. Súbory definujú štruktúry a funkcie pre prácu s kartou. Ovládače taktiež obsahujú pravidlá pre dynamické pridávanie zariadení do `/dev` (combo.udev.rules[1]), Makefile súbory (pre úspešné zostavenie binárnej formy ovládačov automatickými nástrojmi) a rôzne iné pomocné skripty a definície vlastností kariet. V adresári `./kernel/drivers/base` sa nachádzajú súbory definujúce základnú prácu s kartou ako komunikácia s hostiteľským systémom pomocou PCI-Express zbernice. Hlavné jadro ovládačov COMBOv3 generácie kariet sa nachádza v adresári `./kernel/drivers/combov3`. V jadre sú implementované potrebné funkcie na správnu identifikáciu karty, identifikovanie verzie firmvérového dizajnu, rozpoznanie flash pamäte, výber vhodnej metódy pre prístup ku karte a iné. Okrem iného v jadre sa nachádza `sze2cv3` ktorého obsahom je implementácia rozhrania SZE2 pre rýchle DMA prenosy medzi hostiteľským systémom a kartou. Viac o tejto problematike je v [15]. Jadro taktiež inicializuje interné dátové štruktúry s ktorými pracujú samotné užívateľské nástroje. Adresár `./kernel/include` je zameraný na komplexné definície konštánt kárt ako veľkosti adresných priestorov jednotlivých komponentov a informácie o dizajne kariet.

2.4.2 Knižnica libcombo

Ovládač *linux-drivers* poskytuje základné nízko-úrovňové funkcie na manipuláciu s kartou a *libcombo* zabaľuje tieto funkcie do vlastných rozšírených funkcií a dátových štruktúr vhodnejších pre užívateľské aplikácie. Knižnica vytvára akúsi medzivrstvu medzi priamym prístupom užívateľskej aplikácie ku karte cez ovládače. Zároveň funkcie z knižnice majú zabrániť používateľovi pristupovať do celého adresného priestoru karty. Dôvod tohoto opatrenia je odtieniť užívateľa od nízko-úrovňových funkcií, ktoré by mohli spôsobiť pád apliká-

[1] Writing udev rules http://www.reactivated.net/writing_udev_rules.html

cie, prípadne pád celého hostiteľského systému. V obrázku 2.9b je možné vidieť adresárovú štruktúru knižnice. Funkcie pre základnú prácu s COMBO kartami ako je načítanie dát zo zadanej adresy, zápis dát na zadanú adresu, identifikácia karty, identifikácia dizajnu, uloženie dizajnu do karty a rekonfigurácia dizajnu v karte sú implementované v súboroch najvyššej úrovne adresára. Podpriečinok `./libcombo/libcombo` obsahuje súbory implementujúce prácu s komponentami karty cez rôzne rozhrania. Napríklad práca s MDIO rozhraním, s I^2C rozhraním alebo s rozhraním pre prístup do flash pamäte nachádzajúcej sa na karte. V podpriečinku sa taktiež nachádzajú súbory obsahujúce implementáciu funkcií pre prácu s fyzickou sieťovou vrstvou karty (PHY vrstva). Pokiaľ je celá *libcombo* knižnica úspešne skompilovaná vzniknú dynamicky zdielané objekty. Binárna podoba knižnice má názvom libcombo.so.X (.so, z angl. Shared Objects) kde X vyjadruje verziu knižnice. Takto vzniknutá dynamická knižnica sa pripojí pri zostavovaní danej užívateľskej aplikácie.

Ako bolo opísané v kapitole 2.1 je niekoľko typov COMBO kariet a každý typ môže mať rôzne firmvérové konfigurácie. Dizajn karty je pomenovanie konkrétnej konfigurácie karty. Z princípu návrhu sa každý dizajn môže jednoducho odlišovať. V rôznych dizajnoch sú napríklad rozdielne adresné priestory pre komponenty, iný počet vstupných alebo výstupných sieťových vyrovnávacích pamätí alebo rovnaký počet pamätí ale na rôznych adresách a iné. Pre správnu údržbu dizajnov a zisku prehľadu, čo ktorý dizajn obsahuje vznikol koncept *design.xml*.

design.xml je súbor zapísaný v štandardnom XML značkovacom jazyku (z angl. eXtensible Markup Language) s definovanou štruktúrou. Súbor je používaný softvérovými užívateľskými nástrojmi a obsahuje popis adresných priestorov pre každý nachádzajúci sa komponent vo firmvérovej konfigurácií karty. Pre každý dizajn karty existuje samostatný *design.xml*. Návrhár firmvéru karty musí vždy *design.xml* ručne upraviť, aby zodpovedal upravovanej konfigurácií. Ak by tak nevykonal, mohol by vzniknúť konflikt. Pri konflikte by konfigurácia bola prakticky nepoužiteľná so softvérovými nástrojmi. Nástroje by nevedeli aké komponenty sa nachádzajú na karte a aké adresy z adresného priestoru sú použité pre prístup ku komponentom. Ukážkový príklad *design.xml* je v prílohe B. Táto bakalárska práca sa zaoberá odstránením a náhradou *design.xml* flexibilnejším a zároveň automatizovanejším systémom.

2.4.3 Nástroj ethctl

Pre potreby práce je nutné vybrať nástroj na ktorom by bola ukázaná funkcionalita celkového riešenia problematiky. Vybraný nástroj *ethctl* slúži ako dôkaz overenia navrhnutého systému a zároveň je v ňom ukázaná zbavená závislosť na *design.xml*. Dôvody výberu úpravy tohoto nástroja v práci sú, že tento nástroj je komplexný a vyžaduje podporu nemalého počtu rôznych konfigurácií jednotlivých kariet. Z tejto kombinácie vznikla perfektná príležitosť pre vytvorenie nástroja podporujúceho automatickú konfiguráciu FPGA dizajnu a zároveň upravený nástroj poslúži ako vzor pre iné nástroje pri ich refaktorizácií v budúcnosti. *ethctl* využíva aplikačné rozhranie knižnice *libcombo*. Pri riešení problému práce bude na nástroji ukázaná zbytočná duplicita kódu, ktorá bola využitá v doterajšej implementácií. Samotný nástroj slúži k zisteniu prevádzkových informácií o fyzickej sieťovej vrstve nachádzajúcej sa na karte. Obrázok 2.9c stručne zobrazuje adresárovú štruktúru nástroja. Pre správnu identifikáciu sieťových vrstiev PMD a PCS/PMA nachádzajúcich sa na karte vyžaduje aplikácia *design.xml*. Nástroj je tvorený z viacerých modulov:

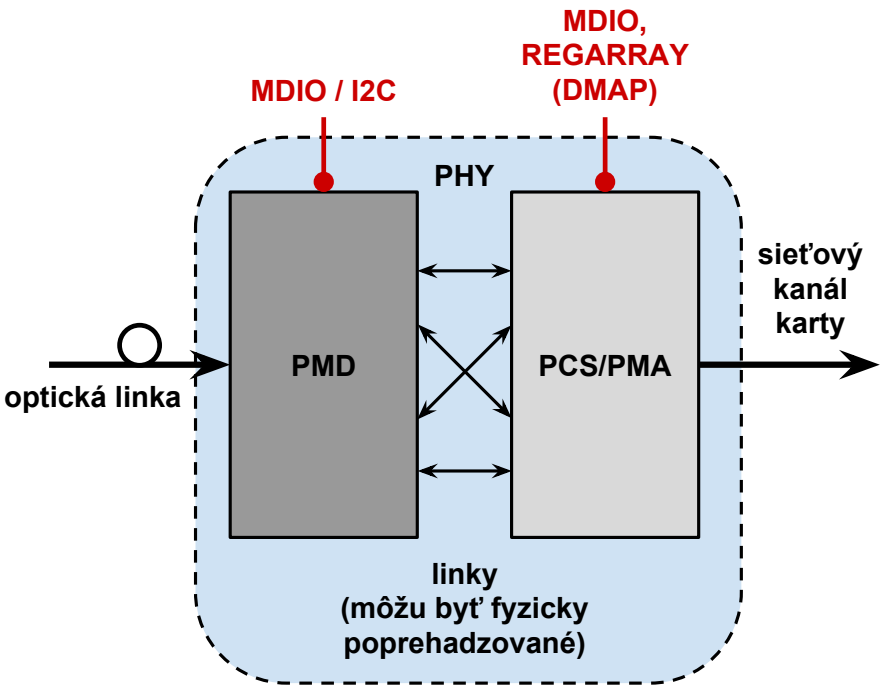
- `ethctl` – jadro celého nástroja, obsahuje funkcie na rozpoznanie dizajnu a následné volanie príslušných funkcií.

- `fb_100ge`, `fb_40ge`, `fb_10ge` – moduly pre rovnomenné typy kariet, definujú jedinečné konštanty a funkcie pre každý typ, rovnako implementujú predpripravenú formu výpisu informácií pre daný typ karty (veľká časť kódu je duplicitná).
- `cfp2`, `qsfpp` – súbory obsahujú konštanty a funkcie pre rozdielny prístup k rozličným optickým modulom sieťovej vrstvy.

Fyzická sieťová vrstva

Pre pochopenie činnosti nástroja *ethctl* je nutné dôkladne porozumieť štruktúre fyzickej sieťovej vrstvy, ktorú nástroj konfiguruje, preto je v tejto časti textu detailnejšie popísaný rozbor najnižšej fyzickej sieťovej vrstvy. Na kartách rodiny COMBO sa nachádza optické sieťové rozhranie podporujúce technológiu 100G Ethernet. FPGA má prístup k sieťovému rozhraniu cez CAUI-4 (4-linkové) elektrické rozhranie alebo CAUI-10 (10-linkové, v prípade určitých konfigurácií COMBO-100G1). Do rozhrania je zapojený optický modul (klietka pre vysielač). Optický modul môže podporovať vysielač (angl. transceiver) typu CFP2, CFP4 alebo QSFP. Moduly pre príslušné vysielače sú si navzájom nekompatibilné, elektricky a konštrukčne pre dané rozhranie. Napríklad do CFP2 kompatibilného modulu nemôžeme vsunúť QSFP vysielač. Vysielače sa následne ešte delia na varianty pre Long Range (LR) a Short Range (SR). Podľa použitého vysielača a jeho varianty LR alebo SR a iných parametrov sa vyberie daná konfigurácia FPGA. Podľa použitej konfigurácie FPGA musí daný nástroj použiť iný spôsob prístupu k adresám vrstvy, ktorá prislúcha danej karte.

Obrázok 2.10 zobrazuje zjednodušenú schému PMD a PCS/PMA vrstvy na karte. PMD vrstva (v obrázku naľavo) je súčasťou celej ethernetovej fyzickej vrstvy (v obrázku celá



Obr. 2.10: Zjednodušená schéma PMD a PCS/PMA vrstvy na COMBO karte

modrá vrstva), taktiež označovanej ako PHY. Fyzická sieťová vrstva štandardného modelu OSI popisuje fyzické a elektrické parametre prenosu dát, rozloženie pinov na konektoroch, napätia a impedancie prenosu a špecifikácie káblov. Súčasťou PHY vrstvy okrem PMD vrstvy je aj PCS/PMA vrstva. Táto vrstva (v obrázku napravo) zodpovedná za kódovanie/dekódovanie prenosu a synchronizácií dát. V COMBO karte sa pre prístup k stavovým a riadiacim registrom PMD a PCS/PMA vrstvy optického modulu (horná časť obrázku) používa MDIO (z angl. Management Data Input/Output) alebo I^2C (z angl. Inter-Integrated Circuit) zbernica, prípadne priamy prístup k registrom pomocou DMAP (direct map, priamy prístup do registrov cez namapované adresy v hostiteľskom počítači). Tieto registre napríklad obsahujú informácie o teplote vysielača, o stave konkrétnej linky či je zapnutá alebo vypnutá, počet BIP chýb v prijatých dátach na linke (z angl. Bit-Interleaved Parity Error Rate) a iné. Celý princíp fyzickej sieťovej vrstvy je na začiatku vstupom optickej linky do PMD (optický sieťový kabel zapojený do vysielača na karte), následne je optický signál transformovaný do elektrického signálu, ten je ďalej modulovaný a dekódovaný PCS/PMA vrstvou a na výstupe je sieťový optický kanál.

2.5 Analýza problémov

Najväčším problémom spojeným s aktuálnym systémom rekonfigurácie nástrojov pri zmene firmvéru FPGA COMBO kariet v použitých projektoch je využitie *design.xml*. Nasledujúci text zhrňuje niekoľko dôvodov prečo je *design.xml* nedostačujúce riešenie a treba nájsť náhradu. Aktuálny vývojový cyklus firmvéru spojeného s *design.xml* vyzerá približne nasledovne. Návrhár firmvéru vytvorí dizajn v ktorom sú napríklad zmenené adresy komponentov (príklad, sieťové vyrovnávacie pamäte IBUFy sa nachádzajú na iných adresách ako v predchádzajúcom dizajne tej istej konfigurácie). Keďže je rozpoznanie správnych adries pre užívateľské nástroje odkázané na popis hardvéru vo forme *design.xml*, návrhár musí taktiež zmeniť ručne popis adries komponent v súbore *design.xml*. Zmena dizajnu môže ovplyvniť viacero variant dizajnov tej istej karty a pre každú variantu treba *design.xml* opraviť (Príklad: zmena adresy 100G IBUFu, musí zodpovedať zmene *design.xml* pre LR4, SR10 variantu 100G1 karty. Následne sa celá zmena musí prejaviť aj v projektoch NIC, HANIC a SDM ktoré túto kartu 100G1 s 100G IBUFom používajú). Vzniká vysoká pravdepodobnosť, že návrhár zabudne upraviť modifikáciu na viacerých miestach.

V prípade že by návrhár zabudol pozmeniť daný *design.xml* a použil by nástroj na prácu s daným komponentom, vyčítané dáta by boli neplatné. V najhoršom prípade by mohlo dôjsť k prístupu na nepovolenú adresu a pádu celého hostiteľského systému (ak by táto situácia nebola ošetrená v jadre OS). Ako bolo čiastočne naznačené, ďalší problém spočíva v tom, že pre danú rovnakú hardvérovú konfiguráciu by mohli byť zmenené len minoritné vlastnosti firmvéru (pozmenené adresy, rôzne verzie použitých komponentov a iné). Pre každú konfiguráciu treba mať vytvorený príslušný *design.xml*, ktorý sa musí distribuovať spolu s konfiguráciou. Problémom *design.xml* je aj v jeho „plochom“ dizajne štruktúry, ktorý nedovoľuje definovať hierarchickú štruktúru a vzťahy medzi komponentami (referencie kde jeden komponent odkazuje na druhý). Ako príklad môže byť, že na najvyššej úrovni FPGA sa nachádzajú sieťové porty. Porty sa skladajú z viacerých sieťových liniek a ku každej linke existuje príslušná vyrovnávacia pamäť IBUF, OBUF. Túto štruktúru je takmer nemožné popísať v *design.xml* jedna k jednej. Vyjadrovacia schopnosť súčasného XML popisu je nedostačujúca. Posledný problém je spojený so samotnými nástrojmi. Napríklad projekt HANIC a SDM majú paketový filter, ktorý je konfigurovateľný nástrojom. Tieto filtre sú firmvérovo implementačne totožné, avšak odlišujú sa generikami VHDL (každý filter má

inú adresu, iné veľkosti blokov a podobne). Samotný nástroj na správu konfigurácie filtra je v podstate tým pádom rozdvojený (duplikovaný) pre HANIC a pre SDM. Nástroj pristupuje rovnako k obom filtrom ale potrebuje rozpoznať iné hardvérové komponenty. Tento problém zovšeobecnenia (nástroj nie je obecný) vyplýva z historickéj implementácie ale je taktiež spojený s využitím *design.xml* štruktúry. Problém duplicity sa vyskytuje aj v nástroji *ethctl*, viac o riešení bude v implementačnej časti práce.

Zhrunutie problémov v piatich bodoch:

1. Nutná ručná úprava *design.xml* zároveň so zmenou vykonanou pre firmvér karty.

2. Existuje viacero verzií *design.xml*.

3. Problém distribúcie *design.xml* spolu s konfiguráciou.

4. Neexistujúca hierarchická štruktúra v *design.xml*.

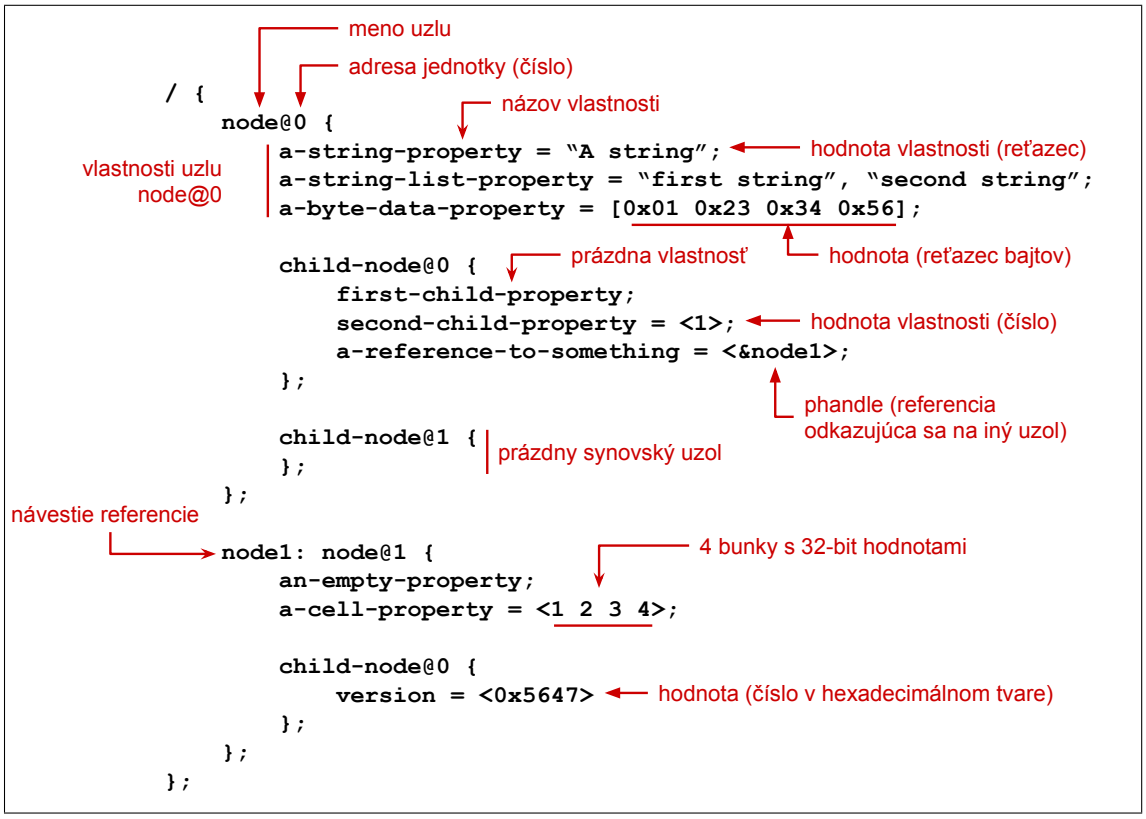
5. Duplicita kódu niektorých užívateľských nástrojov pre prácu s kartou.

Na základe vyššie spomenutých problémov vyplynulo, že sa treba zamerať na odstránenie *design.xml* v ktorom tkvie slabina doterajšieho systému rekonfigurácie nástrojov v daných projektoch NIC, HANIC a SDM. Pre riešenie podobného problému existuje v linuxovom jadre systém Device Tree, ktorý je využívaný pre automatické rozpoznanie hardvérovej platformy a príslušných periférií. Systém slúži napr. pri rozpoznaní a využití správnych modulov na prácu s ARM platformou alebo PowerPC. Nasledujúca podkapitola sa preto venuje rozboru Device Tree štruktúry.

2.6 Device Tree

Device Tree (slovensky, strom zariadení) je dátová štruktúra popisujúca dostupný hardvér, ktorý môže jadro operačného systému plnohodnotne využiť. Štruktúra bola navrhnutá ako rozšírenie Open Firmware [4]. Špecifikáciu Device Tree zastrešuje samotná linuxová komunita a organizácia DeviceTree.org[2]. Pred špecifikovaním a nasadením Device Tree do prevádzky, ktoré sa uskutočnilo v posledných rokoch boli všetky potrebné informácie o hardvéri obsiahnuté v samotnom linuxovom jadre. To znamená, že jadro malo informácie o adresných priestoroch, pamäťových miestach, obslúh prerušení, či informácie o periférnych zariadeniach na čipoch ktoré sa nachádzali v platforme. Tento prístup v podstate fungoval len v plne podporovaných platformách a zriedka jadro úspešne rozpoznalo všetky dostupné komponenty platformy. Vzhľadom na skutočnosť, že popis každej hardvérovej platformy bol vložený do jadra, tak bootloader (zavádzač) mohol povedať jadru nad ktorou platformou je spustený. Jadro potom vnútorne vyhľadalo vhodné parametre platformy a mohlo ich použiť na zaistenie plného využitia dostupného hardvéru. Toto riešenie ale nebolo dostatočné. Prvým problémom je, že v poslednej dobe došlo k stále narastajúcemu počtu malých mikropočítačových dosiek, každá s vlastnou sadou hardvéru. Príkladom môžu byť dosky Beaglebone Black[3] alebo Raspberry Pi[4]. Jadro muselo obsahovať informácie o všetkých platformách a samotné jadro začalo naberať (nabobtnať) na veľkosti. Druhý problém je spojený s tým, že linuxové jadro je centrálne udržiavané a vydáva sa postupne vo verziách. Pre správcov jadra sa tempo ustavičného pridávania špecifikácií platforiem stalo neúnosné. Oba

[2] https://www.devicetree.org
[3] https://beagleboard.org
[4] https://www.raspberrypi.org



Obr. 2.11: Popis základnej štruktúry Device Tree

tieto problémy dali za vznik komunity, ktorá mala navrhnúť riešenie problémov. Vzniklo Device Tree, ktoré je v súčasnosti existujúce funkčné riešenie a používa sa napríklad pri nasadení ARM procesorov v systémoch alebo v PowerPC architektúre.

Nasledujúca časť textu je založená na poznatkoch zo špecifikácie Device Tree [9] a dokumentov venujúcim sa Device Tree [12] a [1]. Základná štruktúra Device Tree s legendou je zobrazená na obrázku 2.11. Štruktúra je zložená z uzlov (v podstate strom uzlov). Jednotlivé uzly môžu byť do seba vnorené, čím vytvárajú hierarchickú podobu. V princípe každý uzol zodpovedá nejakej hardvérovej časti platformy. Uzol musí mať svoje unikátne meno. Ak by mali dva uzly rovnaké meno vznikol by konflikt a jadro by sa nevedelo rozhodnúť, ktoré parametre sú platné pre danú časť hardvéru. Uzly obsahujú vlastnosti (parametre). Samotná vlastnosť sa skladá z mena a prislúchajúcej hodnoty. Hodnoty vlastností môžu byť reťazce, číselné hodnoty v rôznych formách zápisu, bunky hodnôt, atď.

Existujú 3 špeciálne vlastnosti. Prvá vlastnosť je takzvaný **phandle**. V podstate sa jedná o referenciu, ktorá sa odkazuje na iný uzol v hierarchickom popise. Ďalšia špeciálna vlastnosť, ktorá sa môže nachádzať v uzle je **reg**. Vlastnosť popisuje prístup do pamäte, vždy sa skladá z adresy a veľkosti adresného priestoru pamäte. Posledná špeciálna vlastnosť vyjadruje **compatible**. Táto vlastnosť vyjadruje kompatibilitu jednotlivých častí hardvéru s daným systémom. Obsahuje textový reťazec vo forme „`<výrobca>,<model>`“. Je dôležité špecifikovať presné zariadenie a zahrnúť meno výrobcu a model aby sa zabránilo kolíziám v mennom priestore jadra operačného systému. Jadro použije túto vlastnosť o kompatibilite na rozhodnutie ako má byť daný hardvér obsluhovaný.

Doteraz sme popisovali Device Tree ako textový reťazec parametrov (koncovka súboru `.dts`) ale samotný Device Tree popis, ktorý dokáže linuxové jadro spracovať je binárny súbor, typicky končiaci s koncovkou `.dtb`. Binárny súbor obsahuje bajt kód (podobný Java alebo .NET), ktorý jadro dokáže interne spracovať vlastným interpretom. Skutočnosť, že Device Tree je v binárnej forme implikuje fakt, že musí existovať kompilátor, ktorý dokáže skonvertovať zadanú konfiguráciu Device Tree do binárnej podoby. Takýto kompilátor existuje avšak nie všetky linuxové distribúcie ho obsahujú v základe. Verejne dostupný oficiálny kompilátor sa nachádza na GitHube[5]. Kompilátor *dtc* (z angl. Device Tree Compiler) je napísaný v jazyku C a dokáže konvertovať Device Tree z textovej formy do binárnej podoby a naopak. Okrem kompilátora je taktiež dostupná knižnica *libfdt*, ktorá slúži na čítanie a manipuláciu s binárnou formou Device Tree (parsovanie a získanie parametrov jednotlivých uzlov).

[5] https://github.com/dgibson/dtc

Kapitola 3

Návrh systému

V tejto kapitole sa nachádza návrh nového systému pre dosiahnutie automatickej konfigurácie obslužných nástrojov, ktorý vychádza z teoretických poznatkov popísaných v samostatnej kapitole 2. Návrh systému je určený ako náhrada starého neflexibilného systému, ktorý sa používa v projektoch NIC, HANIC a SDM. Hlavným cieľom návrhu je zbaviť sa zastaralého *design.xml*, ktorý musí vývojár vždy ručne upravovať pri zmene firmvéru. Náhradou *design.xml* je Device Tree popis, ktorý odstraňuje tieto nedostatky.

Kapitola je rozdelená do troch podkapitol. V prvej podkapitole 3.1 je rozbor požiadaviek na nový systém, ktorý vychádza z analýzy problémov vznikajúcich v doterajšom systéme. Podkapitola 3.2 je zameraná na popis štruktúry firmvérovej konfigurácie a ako je vôbec možné abstraktne popísať tento firmvér. V podkapitole je taktiež opis súčasnej štruktúry *design.xml* a vytvorenie návrhu Device Tree, ktorý by mal zachovať tento popis a logiku firmvérovej konfigurácie. Posledná podkapitola 3.3 následne nadväzuje na požiadavky systému a zameriava sa na vyriešenie problému zakomponovania Device Tree popisu do súčasného prekladového systému NetCOPE.

3.1 Požiadavky na nový systém

Z podkapitoly 2.5 vychádza päť základných problémov súčasného systému. Prvá veľmi negatívna vlastnosť systému je, že každá prejavená zmena v konfigurácií hardvéru sa musí ručne zapísať do popisu hardvéru (*design.xml*). Požiadavka na nový systém, vyplývajúca z tohoto problému, by mala byť snaha o čo najväčšiu integráciu automatického generovania popisu na základe vykonaných zmien a uľahčiť tak prácu návrhárovi firmvéru. Nové riešenie vyžaduje úpravu v prekladovom systéme NetCOPE, tak, aby sa popis firmvéru generoval automaticky. Pokiaľ vývojár určí adresu komponentov v rámci štruktúry firmvéru, v prekladovom systéme je možné túto informáciu využiť. Okrem vytvorenia samotného firmvéru sa taktiež vygeneruje popis firmvéru. Návrhár nemusí informáciu o adresách duplicitne uvádzať na viacerých miestach.

Ďalším problémom aktuálneho systému je, že existuje viacero verzií popisu hardvéru (viacero *design.xml*). Tieto *design.xml* sú v základe totožné, popisujú ten istý typ karty, ale odlišujú sa v aplikácií pre použité projekty NIC, HANIC a SDM. Cieľom nového systému by malo byť dosiahnutie jednotného popisu pre všetky karty. Riešenie opäť odstraňuje duplicitu. Tým, že popis rovnakých častí firmvérového projektu bude jednotný a na jednom mieste. Odstráni sa problém, že v súčasnom systéme sa pri každej zmene spoločnej časti firmvérových projektov museli modifikovať jednotlivé hardvérové popisy projektov. Popis

musí byť taktiež hierarchický. Príkladom môže byť skutočnosť, že na karte sa nachádza sieťové rozhranie. Rozhranie je tvorené z niekoľkých sieťových liniek. Linky sa zase skladajú zo štyroch základných vrstiev: vstupná vyrovnávacia pamäť (IBUF), výstupná vyrovnávacia pamäť (OBUF), fyzická sieťová vrstva linky (PCS/PMA) a fyzická sieťová vrstva prenosu (hardvérový modul optického vysielača [transceiver], PMD vrstva). Prístup k fyzickej sieťovej vrstve je tvorený pomocou špecifikovaného radiča. Takéto delenie komponentov by mohlo ďalej pokračovať do ešte menších podkomponentov z ktorých sa skladajú hlavné komponenty a ich vrstvy atď. Jednotlivé komponenty v popise majú logické súvislosti medzi sebou a navzájom na seba nadväzujú. V súčasnom systéme je ale možnosť takéhoto hierarchického popisu neuskutočniteľná. Nový systém by mal mať možnosť zapísať hierarchiu komponentov.

Nedostatkom súčasného systému je taktiež nutnosť distribuovať popis *design.xml* spolu s firmvérom ako dva samostatné súbory (*design.xml* a bitstream konfigurácia). S takouto distribúciou vznikajú problémy nekompatibility, ak by súbory neboli totožné, tj. *design.xml* popisuje inú hardvérovú konfiguráciu. Táto skutočnosť by mala byť taktiež riešená v novom systéme ako priame zakomponovanie popisu do firmvéru. Ako posledný bod analýzy súčasného systému je, že v softvérovom jadre projektov sa nachádzajú nástroje, ktoré majú časť kódu duplicitnú. Odstránenie duplicity sa zabezpečí prepísaním nástrojov, ktoré budú fungovať na novom systéme. Navrhnutý systém by mal byť overený pre zhodnotenie celkovej funkčnosti. Overenie systému začínajúc od samotného prekladu a vytvorenia popisu a konfigurácie dizajnu, až po otestovanie nástroja a jeho kompatibility s vytvoreným popisom.

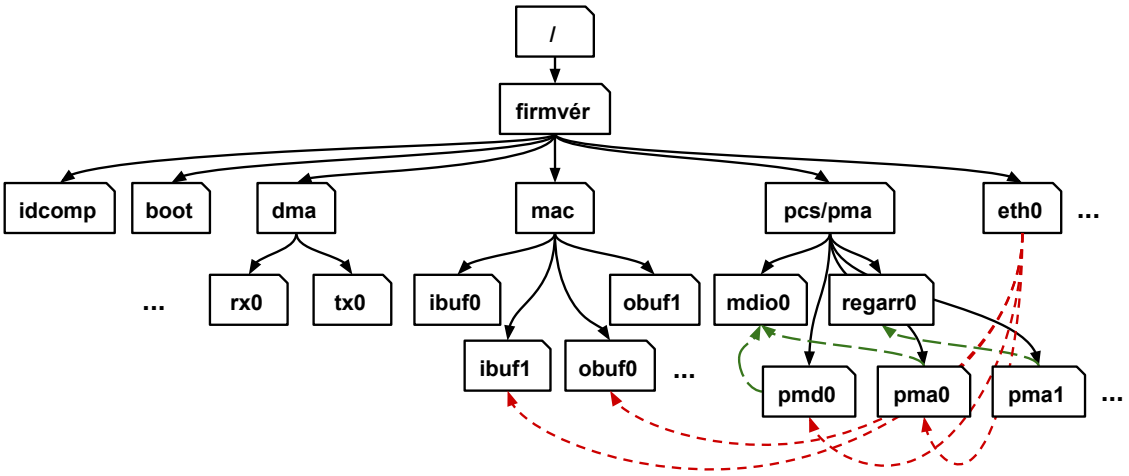
Zhrnutie požiadaviek, ktoré by mal nový systém spĺnať v piatich bodoch:

1. Automatické generovanie popisu na základe firmvéru karty.
2. Jednotný popis pre všetky karty (štruktúra totožná pre všetky karty a projekty).
3. Distribuovať popis priamo s firmvérom (popis priamo zakomponovaný v konfigurácií).
4. Hierarchický popis komponentov.
5. Overenie funkčnosti systému na ukážkovom nástroji *ethctl*.

Z vyššie popísaných požiadaviek vychádza návrh celého systému. Najprv návrhom architektúry a potom návrhom implementácie, ktoré sa zaoberajú súčasným systémom a štruktúrov *design.xml*. Z analýzy systému sa vyvodia potrebné zmeny v systéme, ktoré budú musieť byť vykonané pri samotnej implementácií.

3.2 Návrh architektúry

Návrh architektúry nového systému je založený na rozbore štruktúry súčasného popisu hardvéru (*design.xml*) a na myšlienkach vytvorenia popisu ktorý bude zodpovedať abstraktnému modelu karty a jej komponentov. Ako bolo naznačené vyššie v požiadavkách na nový systém, popis firmvérovej konfigurácie kariet nie je triviálny problém, pretože karta obsahuje viaceré komponenty. Komponenty sa môžu deliť na podkomponenty a tie sa následne môžu ďalej deliť na menšie logické celky atď. Usporiadanie komponentov je hierarchické a tomu by mal zodpovedať aj daný popis ku karte. Podoba súčasného popisu vo forme *design.xml* sa nachádza v prílohe B. Je zrejmé, že popis bol navrhnutý pre svoju jednoduchosť a účel popisu komponentov. Na začiatku súboru sa nachádza záhlavie, ktoré informuje o názve dizajnu, informácie o autorovi dizajnu a typ karty s FPGA čipom pre ktorú bol dizajn



Obr. 3.1: Hierarchický strom popisujúci komponenty firmvérových dizajnov

vytvorený. Po záhlaví sa nachádza firmvérový blok. V tomto bloku sú zahrnuté jednotlivé komponenty. Základom všetkých dizajnov projektov je NetCOPE platforma a tá sa skladá z *fwbase* jadra (firmware base). Vo *fwbase* sú implementované všetky potrebné komponenty z ktorý sa skladá karta. Prekladový systém zariadi, aby boli tieto komponenty použité v dizajne pre určenú kartu. Každý dizajn má v základe použité potrebné komponenty zhodné pre daný typ karty. Čo odlišuje funkcionalitu dizajnu (samotné projekty NIC, HANIC a SDM) je, že každý komponent je inicializovaný s inými parametrami. Komponenty sú napríklad na iných adresách, majú iné počty vyrovnávacích pamätí, rôzny počet sieťových kanálov, atď. Jednotlivé projekty taktiež používajú rozdielne prídavné komponenty. V obrázku 3.1 je zobrazená základná štruktúra komponentov *fwbase* v hierarchickom strome.

Dizajn (konfigurácia) každej karty je tvorená v podstate niekoľkými hlavnými súčasťami:

- **idcomp** komponent slúžiaci pre základnú identifikáciu karty, v komponente sú zaznačené verzie karty, verzie dizajnu a iné.

- **boot** komponent (bootfpga), implementovaný radič na úspešnú rekonfiguráciu a spustenie samotnej karty a načítanie dizajnu.

- **dma** komponent (dma_bus) má na starosti DMA prenosy medzi kartou a hostiteľským systémom (radič DMA). Tento komponent je zložený z viacerých blokov, ktoré popisujú jednotlivé DMA kanály. Napríklad *rx0* značí nultý vstupný (receive) DMA kanál a *tx0* nultý výstupný (transmit) DMA kanál. Karta má viacero DMA kanálov.

- **mac** komponent (network_module), MAC vrstva karty. Sú v nej implementované vstupné a výstupné sieťové vyrovnávacie pamäte (IBUFy a OBUFy)

- **pcs/pma** komponent (eth_phy), obsahuje implementáciu fyzickej sieťovej vrstvy pre jednotlivé sieťové kanály. Táto vrstva využíva komponenty, ktoré implementujú radiče pre prístup do pamäte napr. MDIO radič, I^2C radič alebo priame mapovanie registrov do adresného priestoru. Tieto radiče pristupujú do vrstiev, ktoré reprezentujú samotné vrstvy PMD, PMA a PCS.

- **iné** okrem popísaných komponentov sa v dizajne nachádzajú aj iné dôležité súčasti tvoriace dizajn karty. Napríklad implementujú rôzne radiče pre rozdelenie adresného priestoru karty, jednotka generujúca časové značky a iné. Okrem hlavných komponentov sú vo *fwbase* implementované aj najzákladnejšie podkomponenty z ktorých sú tvorené vyššie abstraktnejšie komponenty. Napríklad rôzne implementácie prístupových radičov, zásobníkov, atď.

Na základe získaných poznatkov zo štruktúry *design.xml* a z faktov, ktoré sa javia ako vhodné pre popis firmvéru a jeho hierarchického rozloženia sa použije pre návrh nového popisu už existujúci a funkčný systém popisu Device Tree. Device Tree riešenie poskytuje možnosti aplikovania všetkých požiadaviek na nový systém. Dovoľuje predovšetkým automatické generovanie popisu až po hierarchický popis komponentov. Časť vzniknutej Device Tree štruktúry je zobrazená na obrázku 3.2. V obrázku je naznačená základná kostra popisu karty. Každý komponent je reprezentovaným samostatným uzlom Device

```
/dts-v1/;
/ {
    firmware {
        design-name = "NIC";
        build-time = <0x59634386>;

        mac: mac_module {
            ibuf0: ibuf0 {
                compatible = "netcope,ibuf";
                type = "xgmii";
                version = <0x1>;
                reg = <0x8000 0x200>;
                mtu = <0xfe0>;
                linux,phandle = <0x1>;
                phandle = <0x1>;
            };
        };

        eth0 {
            compatible = "netcope,eth";
            pmd-name = <&pmd0>;
            pmd-params {
                lines = <3 1 2 0>;
            };
            pcs = <&pcspma0>;
            ibuf = <&ibuf0>;
            obuf = <&obuf0>;
        };
    };
};
```

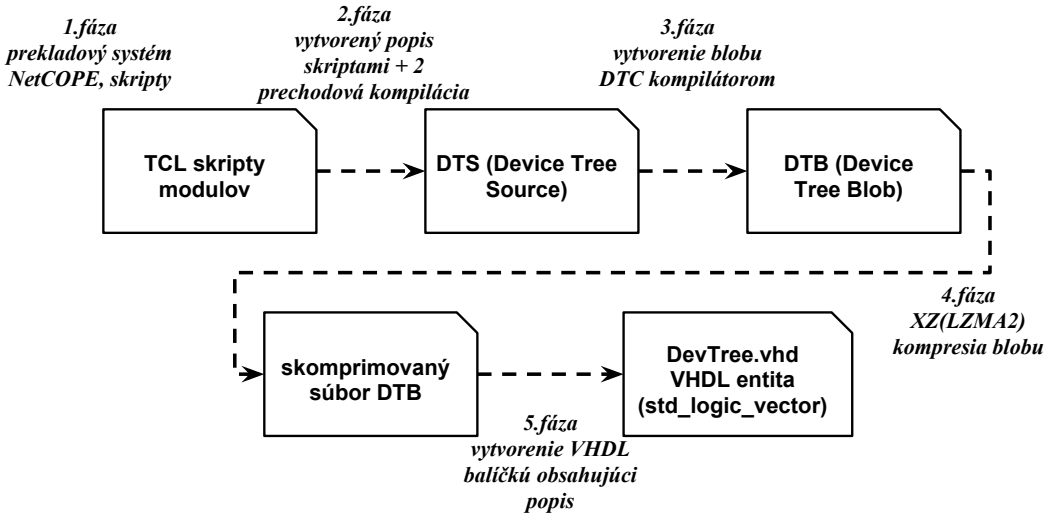
Obr. 3.2: Časť štruktúry Device Tree popisujúca komponenty NIC projektu

Tree štruktúry. V obrázku sa popis skladá z hlavného uzla (*firmware*), ktorý obsahuje všetky ostatné komponenty karty (poduzly). Popis čiastočne vychádza aj z používaného *design.xml*. Prvý zobrazený komponent je *mac* vrstva s jednou vstupnou vyrovnávacou pamäťou (IBUF). Tento popis sedí na spomenutý hierarchický strom a je teda zrejmé, že je možné dosiahnuť hierarchické usporiadanie komponentov v popise. Ďalšou výhodou popisu je možnosť odkazovať komponenty na iné komponenty. Špecifikácia Device Tree povoľuje odkazovanie sa na uzly pomocou *phandle*. Príkladom takéhoto chovania môže byť na karte napríklad sieťový kanál (v obrázku *eth0*). Sieťový kanál sa skladá zo štyroch vrstiev (komponentov). Vrstvy PMD (optický sieťový modul, transceiver), PCS/PMA a MAC vrstva skladajúca sa z sieťových vyrovnávacích pamätí. Sieťový kanál v popise je uzol obsahujúci štyri vlastnosti a tie sa odkazujú na príslušné uzly vrstiev. V obrázku je červenou farbou označená vlastnosť *compatible*. Táto vlastnosť je dôležitá z pohľadu kompatibility softvérových nástrojov. Vlastnosť popisuje komponent, ktorý je súčasťou NetCOPE platformy a v konkrétnom prípade na obrázku sa jedna o vstupnú sieťovú vyrovnávaciu pamäť (IBUF). Použitie vlastnosti spočíva v rozpoznaní správneho komponentu, ktorý je vyžadovaná pre prácu s nástrojom. Vývoj finálnej podoby štruktúry bol iteračný proces. Prvotná podoba popisu PCS/PMA vrstvy jednotlivých kariet sa bližšie nachádza ako samostatný súbor **karty_navrh_popis_pcspma.dts** priložený na pamäťovom médiu. Finálna podoba Device Tree popisu pre konkrétnu COMBO kartu 100G1 v konfigurácií 100G1_LR4 je v prílohe C. Popisy sú vytvorené aj pre iné COMBO karty (v prílohe je uvedená len táto jedna konfigurácia). Ostatné firmvérové popisy pre karty sa nachádzajú v priloženom pamäťovom médiu.

3.3 Návrh implementácie

Štruktúra navrhnutého popisu je vytvorená v podobe Device Tree. Ďalším krokom, ktorý vychádza z požiadaviek je zakomponovanie samotného popisu do firmvéru. Súčasný *design.xml* nie je žiadnym spôsobom vytváraný automatizovane. Popis je jednoducho vytvorený ručne návrhárom ako textový XML súbor podľa špecifikácie. Pre zaručenie jednotného popisu všetkých kariet a ich konfigurácií je tento spôsob ale nevhodný. Spôsob ako zaintegrovať popis do firmvéru je pridanie samotného popisu už do fázy tvorby samotnej konfigurácie karty. Táto fáza je zabezpečená v prekladovom systéme NetCOPE. Prekladový systém NetCOPE je tvorený Tcl skriptami, ktoré sa iteratívne volajú medzi sebou a postupne vyberajú a inicializujú komponenty, ktoré sa použijú pri FPGA syntéze. Následne sú všetky zahrnuté komponenty, z ktorých sa má vytvoriť konfigurácia karty, vysyntetizované nástrojom Xilinx Vivado. Nástroj Vivado po úspešnej syntéze a optimalizáciách ciest vytvorí samotnú finálnu konfiguráciu v podobe bitstream súboru.

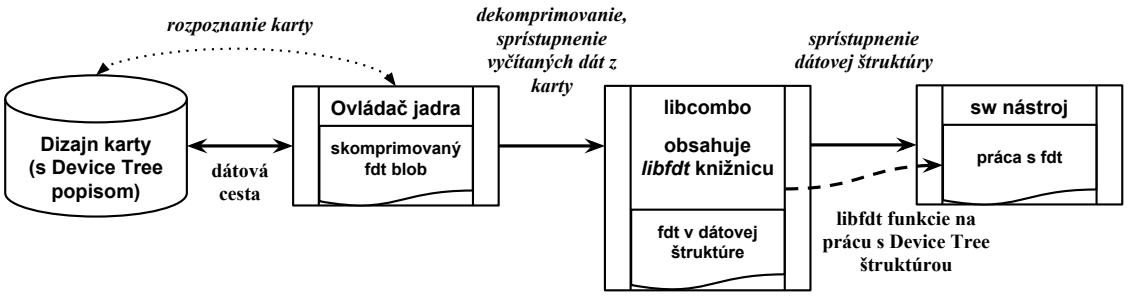
Obrázok 3.3 zobrazuje návrh postupu generovania Device Tree popisu priamo zakomponovaného v konfigurácií karty. Celý návrh spočíva v úpravách prekladového systému NetCOPE a jeho súčastí. Postup je rozdelený do piatich fáz. Najprv sa v prekladovom systéme upravia Tcl skripty jednotlivých modulov (komponenty z ktorých je tvorená karta). Tcl skripty modulov budú tvoriť celky Device Tree popisu. Hlavný modul tieto celky spojí do samostatného Device Tree popisu v textovej podobe. Následne sa použije kompilátor Device Tree *dtc* pre upravenie formy popisu a overenie platnosti samotného popisu. Textová forma Device Tree (.dts) sa kompilátorom prevedie do binárnej podoby Device Tree blob (.dtb). Na blob podobu sa použije kompresný algoritmus LZMA/LZMA2 pre ušetrenie pamäťového miesta. Výsledkom celého generovania je vytvorenie VHDL súboru, ktorý obsahuje binárnu podobu Device Tree popisu. Súbor je pridaný do firmvérovej konfigurácie



Obr. 3.3: Postup generovania Device Tree popisu

a je spolu vysyntetizovaný s inými modulmi tvoriacimi kartu. Tým pádom je zabezpečené, že samotná konfigurácia obsahuje aj popis. Popis je prístupný na zvolenej adrese vo forme pamäťového miesta obsahujúceho dáta.

V obrázku 3.4 je zobrazená štruktúra propagácie vytvoreného dizajnu s popisom do vyšších softvérových vrstiev. Ovládač deteguje kartu (ľavá časť obrázku) a zistí, že obsahuje platný Device Tree popis. Tento popis vyčíta zo zvolenej adresy pamäťového miesta dizajnu. Vyčítaný popis Device Tree je skomprimovaný vo forme binárneho blobu (fdt blob). Ovládač ho najprv musí dekomprimovať a následne sprostredkuje príslušné dáta užívateľskému priestoru softvéru. Knižnica *libcombo*, ktorá slúži ako medzivrstva medzi jadrom a softvérovými nástrojmi, popis zahrnie do internej softvérovej dátovej štruktúry. Dátová štruktúra vyjadruje COMBO kartu a jej súčasti. Knižnica taktiež zaobaľuje knižnicu *libfdt*, ktorá poskytuje funkcie na prácu s Device Tree popisom. Samotný softvérový nástroj si daný popis sprístupní pomocou položky v dátovej štruktúre a môže využívať funkcie na prácu Device Tree. Zmeny propagácie popisu museli byť implementované v softvérovej časti NetCOPE platformy.



Obr. 3.4: Propagácia popisu do softvérových vrstiev systému

Kapitola 4

Implementácia

Kapitola popisuje postup samotnej implementácie celého systému, ktorá vychádza z predchádzajúcej kapitoly venujúcej sa návrhu systému. V kapitole sú zahrnuté zmeny, ktoré boli implementované do prekladového systému NetCOPE. Zmeny slúžia pre vytvorenie funkčného dizajnu s daným Device Tree popisom firmvéru. Samotné modifikácie boli implementované postupne v poradí od najnižšej vrstvy (tvorba dizajnu) až po úpravu softvérového nástroja v súlade s hierarchiou systému. Kapitola je tak rozdelená do štyroch podkapitol. Najprv začínajúc popisom modifikácií a pridaním Tcl skriptov do prekladového systému – podkapitola 4.1, následne podporou Device Tree v samotnom ovládači pre jadro OS – podkapitola 4.2, potom sprístupnením Device Tree popisu užívateľským aplikáciám pomocou *libcombo* knižnice – podkapitola 4.3 a na záver úpravou ukážkového nástroja *ethctl* pre overenie funkčnosti a názornú ukážku funkčnosti systému – podkapitola 4.4.

4.1 Prekladový systém, zaintegrovanie Device Tree popisu

Táto podkapitola rozširuje a vychádza z informácií zahrnutých v podkapitole 3.3. Do prekladového systému NetCOPE bolo implementované automatické generovanie Device Tree popisu, ktorý je priamo zakomponovaný do príslušnej hardvérovej konfigurácie. Pre dosiahnutie tohoto cieľa sa museli pridať a modifikovať existujúce Tcl skripty prekladového systému. Celý prekladový systém NetCOPE platformy so všetkými potrebnými súčasťami a komponentami na vytvorenie funkčného dizajnu sa nachádza v priečinku *fwbase* (firmware base). V obrázku 4.1 je naznačená štruktúra priečinku *fwbase*. Celý priečinok je príliš komplexný a rozvetvený, preto z tohoto dôvodu je zobrazená len určitá časť na vytvorenie hrubej predstavy o prekladovom systéme NetCOPE. V obrázku sú farebne označené súčasti, ktoré boli pridané a modifikované v systéme. Súbory označené červenou farbou existovali v systéme, boli len mierne pozmenené. Tieto súbory sú *Modules.tcl*. *Modules.tcl* majú na starosti zahrnúť a inicializovať všetky potrebné podkomponenty, ktoré tvoria celý dizajn karty, celý komponent atď. Súbory sú previazané a volajú sa iteračne, vnorením hlbšie do hierarchie štruktúr komponentov. Hlavný *Modules.tcl* pre konkrétny projekt a konkrétnu kartu sa nachádza napríklad v `/fwbase/application/nic/40g2/top/`. Tento skript následne volá postupne všetky *Modules.tcl* podkoponentov. Zmena *Modules.tcl* spočíva v pridaní riadku:

```
source $PROJECT_PKG_BASE/DevTree.tcl
```

ktorý vraví že sa má vo fáze prekladu daneho komponentu alebo väčšieho celku, taktiež zahrnúť skript *DevTree.tcl*. Príkaz sa čiastočne odlišuje podľa umiestnenia komponentov.

```
fwbase ..................................................... základ firmvéru kariet
├── application ........................................ aplikačné jadro dizajnov
│   ├── nic ........................... jadrá pre jednotlivé karty 40G2, 100G1, 100G2
│   │   ├── 40g2
│   │   │   ├── pkg ................ v podpriečinku sú užívateľské konštanty konfigurácie
│   │   │   │   ├── combo40g2_user_const.vhd
│   │   │   │   ├── combo10g8_user_const.vhd
│   │   │   │   └── DevTree.tcl
│   │   │   └── top ........ hlavná časť TOP z ktorej sa volá *make* pre vytvorenie dizajnu
│   │   │       ├── Makefile
│   │   │       ├── Vivado.tcl
│   │   │       ├── Modules.tcl
│   │   │       ├── DevTree.dts ..................... vygenerované počas tvorby dizajnu
│   │   │       ├── DevTree.vhd ..................... vygenerované počas tvorby dizajnu
│   │   │       └── ...
│   │   └── 100g1 ........................................ totožná štruktúra ako v 40g2
│   └── hanic ..... okrem NIC existuje aj HANIC a SDM (logická štruktúra je rovnaká)
└── ndk ............... implementované komponenty nachádzajúce sa v dizajnoch kariet
    ├── 40g2
    │   └── src
    │       ├── Modules.tcl
    │       ├── DevTree.tcl
    │       └── comp ..................................... jednotlivé komponenty kariet
    │           ├── netcope_adc
    │           │   ├── Modules.tcl
    │           │   ├── DevTree.tcl
    │           │   ├── netcope_adc.vhd
    │           │   └── ...
    │           ├── dma_bus .... komponenty obsahujú Tcl skripty a implementáciu VHDL
    │           │   ├── Modules.tcl
    │           │   ├── DevTree.tcl
    │           │   └── ...
    │           ├── network_mod
    │           └── eth_phy
    ├── 100g1 ............................ logický princíp štruktúry ako pri karte 40g2
    │   └── ...
    └── common .......... spoločné komponenty, ktoré sú zdieľané naprieč konfiguráciami
        ├── comp
        │   └── nic ........ základná sieťová vrstva kariet, obsahuje viacero komponentov
        │       └── cgmii ........................... CGMII media independent interface
        │           ├── ibuf
        │           │   ├── Modules.tcl
        │           │   ├── DevTree.tcl
        │           │   └── ...
        └── build
            └── DevTree.tcl
```

Obr. 4.1: Priečinok fwbase NetCOPE platformy

DevTree.tcl súbory boli pridané do prekladového systému a sú označené zelenou farbou. Tieto súbory sú Tcl skripty, ktoré generujú samotný Device Tree popis pre daný komponent (uzol Device Tree štruktúry) v ktorom sa nachádzajú. Skripty obsahujú rôzne funkcie s premenným počtom parametrov, napríklad:

```
proc dts_build_netcope {}{};
proc dts_network_mod {pcs_pma base ibufE obufE mtuI mtuO}{};
proc dts_pcs_pma_100g1 {pcs_pma eth_base}{};
proc dts_eth_chnls {pcs_pma}{};
proc dts_ibuf_cgmii {no base mtu}{};
...
```

Každá funkcia generuje vlastnú časť popisu Device Tree. Funkcie generujú popis v textovej podobe a návratové hodnoty funkcií sú textové reťazce. Textové reťazce sa v hlavnom module spoja vo funkcií `dts_build_netcope{}` a vytvoria celý finálny popis tvorený všetkými uzlami. Takto vytvorený popis má podobu ako je znázornené v prílohe C. Do prekladového systému sa teda zaintegrovalo paralelné generovanie popisu súčasne s tvorbou a syntézou firmvéru, už s využitím predpripravenej hierarchie volania komponentov a podkomponentov pomocou súčasného systému NetCOPE (*Modules.tcl*).

Takto vytvorený Device Tree popis ale nie je zahrnutý priamo vo firmvéri konfigurácie. Vytvorený je len textový súbor *DevTree.dts*, v obrázku označený olivovo zelenou farbou. Ako vyplýva z návrhu v obrázku 3.3, tento súbor je prevedený z textovej podoby (.dts) pomocou *dtc* kompilátora do binárnej podoby (blob, .dtb). Na vytvorený Device Tree blob sa použije kompresný algoritmus LZMA/LZMA2. Čisté surové skomprimované dáta sú následne uložené vo VHDL entite v súbore *DevTree.vhd* v std_logic_vector dátovej štruktúre. Celý proces troch fáz od vytvorenia binárneho blobu, cez kompresiu dát po vytvorenie VHDL entity, ktorá obsahuje dáta popisu a je inicializovaná a zakomponovaná do samotnej firmvérovej konfigurácie je implementovaný v Tcl skripte ktorý sa nachádza v `/fwbase/ndk/common/build/DevTree.tcl`. Všetky vykonané úpravy skriptov a súborov prekladového systému NetCOPE sú priložené v pamäťovom médiu v priečinku *fwbase*.

4.2 Ovládač linux-drivers

Tak ako je znázornené v návrhu na obrázku 3.4 dizajn karty obsahuje Device Tree popis. Ovládač musí rozpoznať kartu v hostiteľskom systéme a získať informácie o dostupnosti popisu. Následne ovládač popis sprístupní do softvérového užívateľského priestoru. Samotná modifikácia ovládača *linux-drivers* pre podporu Device Tree popisu hardvéru spočíva v úprave štruktúry *combo6*. Táto štruktúra obsahuje všetky potrebné položky popisujúce samotnú kartu zo softvérového hľadiska. V štruktúre sa nachádzajú napríklad informácie o PCI-Express linke, ktorú karta využíva (asociácia PCI zariadenia s jadrom OS, IRQ čísla prerušení, fyzické a virtuálne adresy pre PCI). V štruktúre sú taktiež informácie a čísla pre identifikáciu karty a adresné priestory namapovaných hardvérových radičov a komponentov karty. Úprava bola vykonaná pridaním údajov o veľkosti Device Tree štruktúry a ukazovateľ na Device Tree štruktúru:

```
int  fdt_size
void     *fdt
```

Okrem úpravy v ovládači musel byť taktiež upravený dizajn karty, konkrétne časť PCI-Express radiča. Do radiča bola pridaná špecifikácia **extended capability** obsahujúca in-

formácie, že dané PCI zariadenie podporuje Device Tree. Vo všeobecnosti pri PCI zariadeniach **capabilities** popisujú PCI konfiguračný priestor určený pre rozšírené schopnosti a špecifiká dodávateľa PCI zariadenia. Funkcie v ovládači, ktoré zabezpečia správne rozpoznanie vlastnosti a následné načítanie dát cez PCI zbernicu z karty do privátneho priestoru ovládača:

```
static int combov3_find_vsec (struct combo6 * combo, u32 vsec_header)
static int combov3_init_fdt  (struct combo6 * combo)
```

Kedže v dizajne je zapísaný samotný Device Tree popis skomprimovaný, funkcie musia dáta dekomprimovať pomocou LZMA/LZMA2 kompresného algoritmu. Device Tree popis je sprístupnený ako súbor na čítanie nachádzajúci sa v systémovej ceste. Do ovládača boli pridané funkcie, ktoré umožňujú toto sprístupnenie:

```
loff_t  combo6_llseek (struct file *file, loff_t off, int whence)
ssize_t combo6_read   (struct file *file, char __user *buffer,
                       size_t length, loff_t *offset)
```

Funkcie dovoľujú prístup k súboru a zaobchádzanie s ním. Napríklad Device Tree popis môže užívateľ vypísať priamo do terminálu pomocou príkazu „`cat /dev/combosix/0`“.

4.3 Knižnica libcombo

Ako bolo napísané, ovládač sprístupňuje Device Tree popis do softvérového užívateľského priestoru. Softvérový nástroj by mohol využiť prístup k popisu priamo bez využitia nejakej abstraktnej vrstvy. Tento spôsob je možný, avšak nevyužívaný kvôli tomu, že každý nástroj by si musel sám ošetrovať všetky možné problémy vznikajúce pri prístupe k popisu a k samotnej nízko-úrovňovej funkcionalite karty. Z tohoto dôvodu je medzi ovládačom a softvérovým nástrojom zavedená medzivrstva poskytujúca jednotné funkcie pre prístup ku karte a k jej popisu. Medzivrstva je vo forme knižnice *libcombo*. Modifikácia knižnice v tejto práci spočívala v zmenách v súboroch `bus.c`, `combosix.h` a `cs_local.h`. Prvá zmena sa týkala úpravy štruktúry `cs_device_t`. Štruktúra slúži na popis samotného dizajnu COMBO karty, jeho identifikácia, výrobca karty, namapované adresného priestoru karty a iné. Do štruktúry sa pridal ukazovateľ na dátový priestor Device Tree popisu.

```
void  *fdt;
```

Tento ukazovateľ sa naplní dátami Device Tree popisu, ktoré sú vyčítané z ovládača. Celá inicializácia COMBO karty (naplnenie dát štruktúry `cs_device_t`, už aj s popisom dizajnu) je implementovaná vo funkcíí, ktorá využíva systémové volania:

```
static int cs_attach_real (cs_device_t **, const char *, int);
```

Popis firmvéru je dostupný v štruktúre `cs_device_t`, ale ako privátna položka. Softvérové nástroje používajúce knižnicu *libcombo* majú prístup k štruktúre `cs_device_t` a pomocou funkcie:

```
void* cs_fdt (cs_device_t *);
```

získajú adresu popisu. Softvérový nástroj následne pracuje s vráteným ukazovateľom (adresa) a má plný prístup k vytvorenému Device Tree popisu pri preklade dizajnu. Knižnica

libcombo okrem sprístupnenia Device Tree popisu softvérovým nástrojom taktiež zaobaľuje základné funkcie na prácu s Device Tree štruktúrou. Tieto funkcie sú implementované v knižnici *libfdt*, ktorá je súčasťou Device Tree kompilátora *dtc*. Do knižnice *libcombo* boli zahrnuté zdrojové a hlavičkové súbory tejto *libfdt* vrstvy. V kapitole návrh implementácie v obrázku 3.4 je táto skutočnosť naznačená.

4.4 Ukážkový nástroj ethctl

V tejto fáze implementácie je vytvorený hardvérový popis dostupný softvérovým nástrojom v dátovej štruktúre inicializovanej v knižnici *libcombo*. Zostáva už len overiť, či navrhnutý popis je dostačujúci k práci so softvérovými nástrojmi. Nasledujúca časť textu je venovaná popisu modifikácie vybraného softvérového nástroja *ethctl*. Na pozmenenom nástroji by sa mala vyskúšať reálna práca s Device Tree popisom.

Samotný nástroj slúži na výpis vybraných informácií vyčítaných z fyzickej sieťovej vrstvy karty. Na začiatku bude popísaná predošlá verzia nástroja. Pred vykonanou zmenou bol nástroj postavený na XML popise *design.xml*. V tomto popise karty nástroj našiel komponenty spojené s PCS/PMA a PMD vrstvou. Na základe nájdených komponentov a prednastavených konštánt, ktoré boli zapísané priamo v zdrojovom kóde, určil nástroj o akú platnú konfiguráciu karty sa jedná a zavolal príslušnú funkciu na inicializáciu interných dátových štruktúr pre prácu s kartou:

```
cs_phyter_t *ph_10ge_attach_by_xml  (cs_device_t *, char *, char *xmlfile);
cs_phyter_t *ph_10g10_attach_by_xml (cs_device_t *, char *, char *xmlfile);
cs_phyter_t *ph_40ge_attach_by_xml  (cs_device_t *, char *, char *xmlfile);
cs_phyter_t *ph_100ge_attach_by_xml (cs_device_t *, char *, char *xmlfile);
cs_phyter_t *ph_100g2_attach_by_xml (cs_device_t *, char *, char *xmlfile);
```

Nevýhodou takéhoto prístupu je, že všetky použité funkcie majú veľkú časť zdrojového kódu duplicitnú. Funkcie sú vcelku totožné, odlišujú sa len počtom a rôznym spôsobom inicializácie komponentov pre prístup k registrom sieťovej vrstvy (radiče MDIO, I^2C). Výstupom funkcií je štruktúra obsahujúca tieto inicializované komponenty. Po úspešnej inicializácií sa volajú funkcie na výpis informácií. Napríklad:

```
void ph_100g2_print (cs_phyter_t *, int ifc, uint8_t, uint8_t transceiver);
void ph_40ge_print_ifc (cs_device_t *ifc, cs_space_t *ifc_space)
void ph_40ge_print (cs_phyter_t *, int ifc, uint8_t, uint8_t transceiver);
void ph_10g10_print (cs_phyter_t *, int ifc, uint8_t, uint8_t transceiver);
void ph_10ge_print_ifc (cs_device_t *mdio, cs_space_t *space, int ifc);
void ph_10ge_print_common (cs_device_t *, cs_space_t *, int ifc, int base);
void ph_10ge_print_speed (uint32_t reg, uint8_t pma);
...
```

Ako vidno zo zoznamu funkcií, funkcie sa opakujú. Zmenené sú len na základe názvov prislúchajúcich konfigurácií 10ge, 40ge, 10g10, 100g, 100g2. Veľká časť kódu funkcií je taktiež duplicitná. Navyše nástroj nedokáže rozoznať iné typy kariet (dizajnov), pokiaľ ich nemá zaznačené vo svojom internom zozname, pre ktoré ma vytvorené potrebné konštanty. Nástroj nie je jednoducho dostatočne obecný pre všeobecnú manipuláciu s kartami a výpisom potrebných dát. Účelom tejto práce bolo taktiež nástroj vyčistiť a zmeniť nepotrebné funkcie.

Vo vlastnej implementácií nástroja *ethctl* sú všetky vyššie popísané funkcie odstránené a nahradené viacerými generickými funkciami, ktoré nevyžadujú prednastavené konštanty (tieto funkcie sa nachádzali v súboroch (`fb_10ge.c`, `fb_40ge.c`, `fb_100ge.c`, súbory sa zrušili). Princíp výpisu sieťových liniek a ich vrstiev v novom systéme postavenom na Device Tree popise spočíva v priechode štruktúry a hľadaní potrebných informácií. V hardvérovom popise majú primárne komponenty vlastnosť *compatible*. Sieťové linky, pre ktoré má tento nástroj vypísať informácie, majú túto vlastnosť vygenerovanú s hodnotu „**netcope,eth**". V nástroji sú použité funkcie z knižnice *libfdt* pre prácu s Device Tree štruktúrou. Jednoduchým cyklom sa prechádza cez všetky uzly popisu a porovnáva sa vlastnosť *compatible*.

```
#define for_each_compatible_node(fdt, node, compatible)              \
for ( node = fdt_node_offset_by_compatible(fdt, node, compatible);  \
      node >= 0;                                                     \
      node = fdt_node_offset_by_compatible(fdt, node, compatible) )
```

Pokiaľ je porovnávaná hodnota totožná s hľadanou hodnotou, získame ofset uzlu, ktorý popisuje sieťovú linku. Z návrhu popisu sieťovej linky (obrázok 3.2 vieme že uzol sa skladá z vlastností, ktoré obsahujú odkazy (phandle) na podkomponenty z ktorých sa skladá sieťová vrstva. Jednoducho získame odkazy na tieto uzly a presunieme interné ukazateľe na daný uzol.

```
phandle = fdt_getprop(fdt,eth_node,PCSPMA,&len);
pcspma_node =
    fdt_node_offset_by_phandle(fdt,fdt32_to_cpu(*(uint32_t *)phandle));
```

Tento postup sa opakuje (napr. vrstva PCS/PMA má prístup k registrom pomocou MDIO radiča alebo priamym prístupom, v samotnom uzle PCS/PMA sa nachádza odkaz na danú prístupovú vrstvu) až kým sa dostaneme k uzlom obsahujúcim informácie o adresnom priestore komponentov zaznačených vo vlastnosti **reg**. Vyčítané hodnoty vyjadrujú základnú (bázovú) adresu pre prístup k danej vrstve a veľkosť adresného priestoru. Okrem toho sa získa informácia aká metóda prístupu k vyčítaniu registrov sa má zvoliť. Získané dáta sa pošlú do všeobecnej funkcie, ktorá vypíše informácie o danej vrstve.

```
// nízko-úrovňové metódy prístupu k registrom priamo na karte
uint16_t cs_pcspma_mdio_read (cs_device_t *,cs_space_t *,u_int,u_int,u_int);
uint16_t cs_pcspma_dmap_read (cs_device_t *,cs_space_t *,u_int,u_int,u_int);

// získanie hodnôt bázovej adresy a veľkosti adresného priestoru
property = fdt_getprop(fdt,ctrl_node,REG,&len);
base = fdt32_to_cpu((*(fdt32_t *)property));
size = fdt32_to_cpu((*(fdt32_t *)(property+4)));

// voľba metódy prístupu
readptr = &cs_pcspma_mdio_read;

// volanie funkcie na výpis informácií o vrstve
print_pcspma_common((*readptr),dev,space,prtdev);
```

Podobný princíp je použitý aj pre výpis údajov optického sieťového modulu (PMD vrstva, transceiver). Samotná funkcia `print_pcspma_common()` vypisuje dáta vrstvy. Informácie,

ktoré sú možné vyčítať z jednotlivých registrov vrstiev pochádzajú zo špecifikácie IEEE Ethernet 802.3 [5] a z [16]. Funkcia vyčíta obsah registrov a následne vymaskuje príslušné bity. Potom tieto informácie vypíše na terminál vo vhodnom textovom formáte (namiesto konkrétnych hodnôt jednotlivých bitov registra). Počas implementácie sa prišlo na to, že niektoré informácie v popise o adresných priestoroch dizajnov nesedia. Device Tree popis sa potom čiastočne upravil, aby zodpovedal príslušnej špecifikácií a implementovanej skutočnosti. Z tohoto dôvodu sa ďalej pracuje na implementácií podpory ďalších konfigurácií COMBO kariet. Implementovaný nástroj v súčasnosti dokáže rozpoznať a vypísať informácie pre konfigurácie COMBO kariet 40G2 a 100G1. Na nástroji bola ukázaná úspešná podpora Device Tree popisu.

Kapitola 5

Overenie funkčnosti

Implementácia celého systému je dokončená. V systéme boli implementované súčasti od automatického generovania popisu konfigurácie FPGA až po ukážkový nástroj pracujúci s vytvoreným popisom. Následne, aby snaha, ktorá bola vynaložená pri návrhu, tvorbe a implementácií mala zmysel, je nutné overiť celú funkčnosť systému. Taktiež je nutné overiť, či sú splnené všetky požiadavky, ktoré boli predložené na začiatku tvorby návrhu nového systému.

Systém bol postupne testovaný po menších častiach, tak ako bol implementovaný. Prvotné otestovanie prebiehalo pri implementácií zakomponovania Device Tree popisu do prekladového systému NetCOPE. Overenie tvorby textového Device Tree popisu (.dts) mohlo byť ešte pred vykonaním samotnej syntézy (Tcl skripty sa inicializujú pred fázou syntézy). Tým pádom chyby spojené so zlým návrhom popisu konfigurácie mohli byť odhalené relatívne jednoducho hneď po spustení prekladu. Celý proces tvorby správnej štruktúry popisu bol iteračný. Trvalo niekoľko verzií vytvorených popisov, kým sa ustálila finálna podoba súčasného popisu. Ukážka finálnej podoby sa nachádza v prílohe C. Táto podoba nie je definitívna a možno sa časom ukážu nedostatky, ktoré by mohli byť upravené. V súčasnej podobe popisu sa nájdu určité odlišnosti voči prvotného návrhu.

Ďalšia fáza testovania systému spočívala v úspešnom dobehnutí prekladu a vytvorení samotného dizajnu s popisom, ktorý sa môže nahrať do COMBO karty. Paralelne s touto fázou sa vyvíjal ovládač, aby bolo možné vyčítať popis z dizajnu. Tento krok sa taktiež podaril a ovládač dokáže úspešne rozpoznať dizajn, ktorý obsahuje Device Tree popis, a následne ho sprístupniť softvéru. Keďže popis je dostupný v softvéri, ďalším logickým krokom je sprístupnenie firmverového popisu knižnici. Overenie funkčnosti knižnice sa mohlo prejaviť len pri implementácií samotného ukážkového nástroja kvôli tomu, že knižnica nepracuje priamo s popisom, iba ho sprístupňuje. Posledným krokom pre úplne overenie funkčnosti firmvérového popisu je implementovanie ukážkového nástroja.

Vybraný nástroj *ethctl* bol implementovaný a mohlo sa vyskúšať, či dokáže rozpoznať dizajn karty a následne vypísať údaje o fyzickej sieťovej vrstve portov, ktoré sa nachádzajú v konfigurácií. Skutočný výpis, ktorý nástroj zobrazuje je v prílohe D. Nástroj bol odskúšaný na dvoch COMBO kartách v dvoch rôznych konfiguráciách s rôznym prístupom k fyzickej sieťovej vrstve (využitie MDIO radiča alebo priamy prístupu k registrom PMA vrstvy). Konkrétne testy prebiehali na kartách 40G2 v konfigurácií 8x10G a 100G v konfigurácií 1x100G (LR4). Vo výpise, ktorý poskytuje nástroj je možne vidieť a následne overiť, že napríklad pri karte 40G2 nebol fyzický zasunutý optický sieťový modul do druhého sieťového portu. Vo výpise (posledná strana, Transceiver pmd1) sa toto prejavilo ako vyčítanie samých 1 (stav vysokej impedancie) z registrov prislúchajúcej vrstvy. Dáta sa následne

previedli do textovej podoby. Je teda dokázané, že vybraný nástroj *ethctl* dokáže pracovať s vygenerovaným popisom Device Tree a získať užitočné informácie.

Všetky požiadavky, ktoré boli zadané pri návrhu nového systému sú úspešne splnené. Súčasný systém dokáže automaticky generovať jednotný popis založený na základe vytvorenej konfigurácie karty (kapitola 3.2 a 4.1). V popise je zachovaná hierarchia komponentov a sú naznačené vzťahy medzi komponentami (viď. príloha C a kapitola 3.2). Vytvorený popis je taktiež distribuovaný priamo s konfiguráciou ako súčasť firmvérovej konfigurácie (kapitola 3.3 a 4.1). Celý systém je teda otestovaný a vyzerá ako úspešná náhrada za zastaralý *design.xml*.

Kapitola 6

Záver

Cieľom bakalárskej práce bol návrh a vytvorenie automatickej konfigurácie obslužných nástrojov pre FPGA firmvér. V práci bol navrhnutý a úspešne implementovaný systém, ktorý dokáže automaticky generovať popisy firmvérových konfigurácií COMBO kariet. Navrhnutý popis vychádza z Device Tree štruktúry. Device Tree je existujúce riešenie popisu hardvéru a používa sa už v linuxovom jadre. Takto vytvorený popis dokáže vhodne zachytiť hierarchiu jednotlivých komponentov z ktorých je tvorený samotný celok firmvéru. V práci bola navrhnutá vhodná štruktúra popisu pre existujúce COMBO karty. Následne bol tento Device Tree popis kariet integrovaný do prekladového systému NetCOPE tak, aby sa automaticky vygeneroval pri tvorbe samotnej konfigurácie dizajnu. Úpravou bolo taktiež dosiahnuté, že popis je distribuovaný priamo v samotnej konfigurácií pre FPGA. Okrem úpravy prekladového systému pre podporu nového Device Tree riešenia sa museli modifikovať aj všetky vrstvy softvérovej časti platformy NetCOPE: ovládač, knižnica *libcombo* a vybraný nástroj *ethctl*. Po modifikácií, má daný ovládač linuxového jadra možnosť rozpoznať popis nachádzajúci sa v dizajne. Ten dokáže sprístupniť do softvérového užívateľského priestoru. Následne knižnica *libcombo* inicializuje interné dátové štruktúry pre prácu s kartou, ktoré obsahujú aj daný popis. Štruktúry (s popisom) knižnica spropaguje softvérovým nástrojom, aby s ním mohli pracovať. Knižnica *libcombo* je taktiež rozšírená o pomocnú knižnicu *libfdt*, ktorá dovoľuje prácu s Device Tree štruktúrou na ktorej je popis založený. V práci bol modifikovaný ukážkový nástroj *ethctl*. Samotný nástroj *ethctl* slúži na výpis údajov o fyzickej sieťovej vrstve nachádzajúcej sa zvolenej karte. V nástroji bola odstránená duplicita kódu a kompletne prerobená logika práce s popisom. Nástroj si postupne nájde všetky potrebné údaje v popise a potom pracuje priamo s kartou cez nízko-úrovňové funkcie. Pomocou týchto funkcií vyčíta údaje a tie vypíše na terminál.

Súčasťou práce bolo taktiež dôkladné overenie funkčnosti celého systému a práce softvérového nástroja s popisom. Pre špeciálne vytvorenú konfiguráciu FPGA všetko funguje podľa očakávania. Počas testovania bola prítomnosť Device Tree popisu v dizajne karty správne rozpoznaná linuxovým ovládačom. Popis bol ovládačom dekódovaný a predaný knižnici *libcombo*. Úspešne bolo overené aj sprístupnenie Device Tree popisu prostredníctvom knižnice do softvérového nástroja *ethctl*. Ten následne plne podporuje prácu s popisom a dokáže vyhľadávať potrebné dáta v popise. Nástroj tak nie je závislý na zastaranom popise *design.xml*. Do nástroja sa úspešne podarilo integrovať plnú podporu práce s COMBO kartami 40G2 a 100G1 na ktorých sa nástroj reálne testoval. Testovanie dopadlo zdarne a nástroj dokáže vyprodukovať platný výpis, príloha D. Pridanie podpory ďalších kariet je jednoduchá rutina, pretože základný vzor pre prácu s kartami bol úspešne implementovaný. Rozšírenie podpory kariet je jedno z možných pokračovaní práce.

Nástroj bol implementovaný ako ukážka slúžiaca pre budúcu reimplementáciu ostatných softvérových nástrojov pracujúcich s kartami, ktoré budú postavené na novom systéme popisu Device Tree. V ostatných nástrojoch je stále používaná závislosť na zastaranom popise *design.xml* a do budúcnosti sa počíta s úplným odstránením tejto závislosti. Aktuálne z dôvodov spätnej kompatibility knižnica *libcombo* podporuje oba systémy popisov (nový Device Tree ale aj zastaraný *design.xml*). V momente keď sa prepíšu všetky nástroje na podporu nového systému Device Tree, bude možné túto závislosť definitívne z knižnice odstrániť. V najbližšej dobe sa taktiež počíta s prechodom NetCOPE prekladového systému na nový modulárny preklad. Modulárny preklad umožní predsyntetizovať jednotlivé komponenty samostatne a urýchli tak tvorbu finálnej konfigurácie. S nasadením nového systému bude spojená aj čiastočná zmena v aktuálnej implementácií Device Tree. Samotný systém Device Tree bol pri návrhu na túto skutočnosť dopredu pripravený. Jedným z možných rozšírení pri prechode na nový preklad môže byť vytváranie samostatných častí Device Tree uzlov komponentov (forma .dtsi častí, ktoré ako celok tvoria samotný popis .dts). Pri modifikácií linuxového ovládača bude možné tieto časti priamo meniť už za prevádzky (angl. hot plug, prípade hot swapping) priamo v načítanom Device Tree popise z karty. Vytvorená podoba Device Tree štruktúry popisu (príloha C) nie je definitívna a môže sa časom meniť. Stále sa objavujú nové skutočnosti a parametre komponentov, ktoré by mohlo byť vhodné zakomponovať do popisu a ktoré pôvodný *design.xml* nepodporoval alebo to neumožňoval. To ukazuje, že nový systém je rozšíriteľný a otvára tak nové možnosti pre lepší popis konfigurácie FPGA. Na práci sa dá tak ďalej pokračovať a vytvorený systém vylepšovať.

Literatúra

[1] *About the Device Tree*. [Online; cit. 05.07.2017].
URL http://www.ofitselfso.com/BeagleNotes/AboutTheDeviceTree.pdf

[2] Altera: *White Paper - FPGA Architecture*. [Online; cit. 24.06.2017].
URL https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01003.pdf

[3] Friedl, Š.; Puš, V.; Matoušek, J.; aj.: *Designing a Card for 100 Gb/s Network Monitoring*. [Online; cit. 29.06.2017].
URL https://www.cesnet.cz/wp-content/uploads/2014/02/card.pdf

[4] IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices. *IEEE Std 1275-1994*, 1994: s. 1–262, doi:10.1109/IEEESTD.1994.89427.

[5] IEEE Standard for Ethernet. *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)*, December 2012: s. 1–3747, doi:10.1109/IEEESTD.2012.6419735.

[6] Kekely, L.: *Hardwarová akcelerace aplikací pro monitorování a bezpečnost vysokorychlostních sítí*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2013.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=15490

[7] Košař, V.; Korček, P.; Žádník, M.; aj.: Hacking NetCOPE to Run on NetFPGA-10G. In *Proceedings of the 7th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2011, s. 217–218, doi:10.1109/ANCS.2011.40.

[8] Kučera, J.: *Aplikačně specifický procesor pro stavové zpracování síťových dat*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2014.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=16018

[9] Linaro: *Devicetree Specification, Release 0.1*. [Online; cit. 05.07.2017].
URL http://www.devicetree.org/specifications-pdf

[10] Martínek, T.: *Návrh číslicových systémů (INC) – Technologie FPGA*. [Online; cit. 04.07.2017].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FINC-IT%2Flectures%2Ftechnologie_fpga.pdf

[11] Martínek, T.; .Košek, M.: NetCOPE: Platform for Rapid Development of Network Applications. In *2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, Apríl 2008, s. 1–6, doi:10.1109/DDECS.2008.4538789.

[12] Petazzoni, T.: *Device Tree for Dummies*. [Online; cit. 05.07.2017].
URL https://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf

[13] Puš, V.; Kekely, L.; Špinler, M.; aj.: *HANIC 100G: Hardware accelerator for 100 Gbps network traffic monitoring*. [Online; cit. 08.07.2017].
URL https://www.cesnet.cz/wp-content/uploads/2015/01/hanic-100g.pdf

[14] Sekanina, L.; Vašíček, Z.; Růžička, R.; aj.: *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Edice Gerstner, Academia, 2009, ISBN 978-80-200-1729-1, 328 s.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9123

[15] Vido, M.: *DPDK nad síťovými kartami COMBO*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2016.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=18312

[16] Xilinx: *10G Ethernet PCS/PMA v6.0 - LogiCORE IP Product Guide*. [Online; cit. 16.07.2017].
URL https://www.xilinx.com/support/documentation/ip_documentation/ten_gig_eth_pcs_pma/v6_0/pg068-ten-gig-eth-pcs-pma.pdf

Príloha A

Obsah priloženého pamäťového média

```
/
├── source/
│   ├── fwbase/ ..................................... prekladový systém NetCOPE
│   ├── linux-drivers/ ............................... ovládač jadra
│   ├── libcombo/ ....................................... knižnica
│   │   └── libfdt/ ................................ pomocná Device Tree knižnica
│   ├── swtools/
│   │   └── ethctl/ ................................ upravovaný softvérový nástroj
│   └── karty_navrh_popis_pcspma.dts ..... návrh Device Tree popisu pre rôzne karty
├── latex/
├── bp-xperes00.pdf
└── README.txt
```

Adresár `source/` obsahuje všetky zdrojové súbory. V priečinku sa nachádza implementácia systému automatického generovania Device Tree popisu firmvéru (podadresár *fwbase/*, štruktúra ako v obrázku 4.1), úpravy vykonané v ovládači (podadresár *linux-drivers/*, štruktúra ako v obrázku 2.9a), úpravy v knižnici *libcombo* (podadresár *libcombo/*, štruktúra ako v obrázku 2.9b) a úpravy na vybranom softvérovom nástroji *ethctl* (podadresár *swtools/ethctl/*, štruktúra ako v obrázku 2.9c). Priložené súbory sú samostatne nepreložiteľné, pretože celá implementácia prebiehala v privátnych repozitároch CESNET výskumnej aktivity. Pre zostavenie projektu a dizajnu kariet je nutnosť prístupu k privátnemu repozitáru. V priečinku sa taktiež nachádza súbor **karty_navrh_popis_pcspma.dts** v ktorom je zobrazená prvotná kostra návrhu Device Tree popisu pre jednotlivé karty.

Adresár `latex/` obsahuje LaTeX zdrojové súbory pre sadzbu bakalárskej práce. V adresári sa taktiež nachádzajú obrázky, ktoré boli použité v práci.

Súbor `bp-xperes00.pdf` je vo formáte PDF a obsahuje finálnu podobu textu práce, ktorá bol vytvorená z LaTeX zdrojových súborov

Súbor `README.txt` obsahuje informácie o adresárovej štruktúre priloženého pamäťového média. V súbore sú poskytnuté informácie pre úspešné vytvorenie funkčných dizajnov a preloženie softvérových nástrojov, ktoré boli implementované v práci.

Príloha B

Ukážka Design XML

```
<?xml version="1.0" encoding="ASCII"?>
<combo6design dversion="1.0" version="1.01">
   <comment>Network Interface Card firmware at Fiberblaze FB1CG
    card.</comment>
   <utctimestamp></utctimestamp>
   <author email="xperes00@stud.fit.vutbr.cz">Martin Peresini</author>
   <copyright>CESNET</copyright>
   <firmware fversion="1.0" id="0x41c10701"
             strid="NIC_FB1CG"
             chip="xc7vh580t" location="0" base="0">
      <comment>NIC firmware for FB1CG</comment>
      <fwdata url="file://fb1cg.bit"/>
      <component cversion="1.0" version="1.6"
                 name="ID32" index="0" base="0x00000000" size="0x100">
         <comment>ID component</comment>
      </component>
      <component cversion="1.0" version="1.0"
                 name="BOOT" index="0" base="0x00002000" size="0x08">
         <comment>BootFPGA component</comment>
      </component>
      <component cversion="1.0" version="2.3"
                 name="TSU_GEN" index="0" base="0x00004000"
    size="0x030">
         <comment>Timestamp unit using PPS pulse from GPS - needs sw
    regulation.</comment>
      </component>
      <component cversion="1.0" version="2.4"
                 name="IBUF" index="0" base="0x00008000" size="0x200">
         <comment>Input Buffer 0</comment>
      </component>
      <component cversion="1.0" version="1.0"
                 name="OBUF" index="0" base="0x00009000" size="0x100">
         <comment>Output Buffer 0</comment>
      </component>
      <component cversion="1.0" version="1.0"
           name="RX_DMA" index="0" base="0xC000" size="0x1000">
         <comment>RX DMA controllers</comment>
      </component>
```

```
      <component cversion="1.0" version="1.0"
            name="TX_DMA" index="0" base="0xD000" size="0x1000">
         <comment>TX DMA controllers</comment>
      </component>
      <component cversion="1.0" version="1.0"
            name="DMA_STATS" index="0" base="0xE000" size="0x1000">
         <comment>DMA Stats registers</comment>
      </component>
      <component cversion="1.0" version="1.0"
            name="DMA_CONF" index="0" base="0xF000" size="0x1000">
         <comment>DMA global configuration registers</comment>
      </component>
      <component cversion="1.0" version="1.0"
                  name="PHYTERMDIO" index="0" base="0x00800000"
   size="0x10000">
         <comment>PCS/PMA layers MDIO controller</comment>
      </component>
      <component cversion="1.0" version="1.0"
                  name="CFP2MDIO" index="0" base="0x04400000" size="0x8">
         <comment>CFP2 transceiver MDIO controller</comment>
      </component>
      <component cversion="1.0" version="1.0"
                  name="USER" index="0" base="0x02000000"
   size="0x02000000">
         <comment>User defined component</comment>
      </component>
   </firmware>
</combo6design>
```

Príloha C

Ukážka Device Tree popisu

```
/dts-v1/;

/ {
    firmware {
        build-tool = "Vivado v2017.2 (64-bit)";
        build-author = "xperes00@stud.fit.vutbr.cz";
        build-revision = "9381786";
        build-time = <0x5970b8ad>;
        card-name = "NFB-100G1";

        idcomp: idcomp {
            compatible = "netcope,idcomp";
            reg = <0x0 0x100>;
            version = <0x10003>;
        };

        boot: boot_controller {
            compatible = "netcope,boot_controller";
            reg = <0x2000 0x8>;
            type = <0x1>;
            version = <0x1>;
        };

        tsu: tsucomp {
            compatible = "netcope,tsucomp";
            reg = <0x4000 0x30>;
            type = <0x1>;
            version = <0x1>;
        };

        dma: dma_module {
            dma_ctrl_sze_rx0 {
                compatible = "netcope,dma_ctrl_sze_rx";
                reg = <0xc000 0x40>;
```

```
                version = <0x10001>;
            };
a
            dma_ctrl_sze_rx1 {
                compatible = "netcope,dma_ctrl_sze_rx";
                reg = <0xc040 0x40>;
                version = <0x10001>;
            };

            dma_ctrl_sze_tx0 {
                compatible = "netcope,dma_ctrl_sze_tx";
                reg = <0xd000 0x40>;
                version = <0x10001>;
            };

            dma_ctrl_sze_tx1 {
                compatible = "netcope,dma_ctrl_sze_tx";
                reg = <0xd040 0x40>;
                version = <0x10001>;
            };
        };

        mac: mac_module {
            ibuf0: ibuf0 {
                compatible = "netcope,ibuf";
                type = "cgmii";
                version = <0x1>;
                reg = <0x8000 0x200>;
                mtu = <0xfe0>;
                linux,phandle = <0x5>;
                phandle = <0x5>;
            };

            obuf0: obuf0 {
                compatible = "netcope,obuf";
                type = "cgmii";
                version = <0x1>;
                reg = <0x9000 0x200>;
                mtu = <0xfe0>;
                linux,phandle = <0x6>;
                phandle = <0x6>;
            };
        };

        pcs_pma: pcs_pma_100g1_100G1 {
            type = <0x1>;
            version = <0x1>;
```

```
            mdio0: mdio0 {
                compatible = "netcope,mdio";
                reg = <0x880000 0x10>;
                linux,phandle = <0x1>;
                phandle = <0x1>;
            };

            regarr0: regarr0 {
                compatible = "netcope,pcsregs";
                reg = <0x800000 0x40000>;
                linux,phandle = <0x2>;
                phandle = <0x2>;
            };

            pmd0: pmd0 {
                compatible = "netcope,transceiver";
                type = "CFP2";
                control = <0x1>;
                linux,phandle = <0x3>;
                phandle = <0x3>;
                control-param {
                    dev = <0x0>;
                };
            };

            pcspma0: pcspma0 {
                type = "100G";
                control = <0x2>;
                linux,phandle = <0x4>;
                phandle = <0x4>;
                pma-control-param {
                    offset = <0x10000>;
                };
                pcs-control-param {
                    offset = <0x30000>;
                };
            };
        };

        eth0 {
            compatible = "netcope,eth";
            pmd = <0x3>;
            pcspma = <0x4>;
            ibuf = <0x5>;
            obuf = <0x6>;
        };
    };
};
```

Príloha D

Ukážka výpisu ethctl nástroja

Výpis D.1: COMBO 100G1 (1x100)

```
------ Interface eth0 --------
------   PMA regs   -----------
Link status:                DOWN
Speed:                  100 Gb/s
PMA type:           100GBASE-LR4
Transmit Fault:               No
Receive Fault:               Yes
------   PCS regs   -----------
Link status:                DOWN
Speed:                  100 Gb/s
Transmit Fault:               No
Receive Fault:                No

---  Transceiver  pmd0 -------
Temperature:            32.37 C
Vendor name:      FINISAR CORP.
Vendor PN:         FTLC8221RFNM
Compliance:       100GBASE-SR10
Connector:                  MPO
HW spec. rev.              1.00
Managenent ifc. spec.:     2.20
RX input power
channel  1: 0.60uW (-32.22dBm)
channel  2: 0.00uW (-inf  dBm)
channel  3: 0.00uW (-inf  dBm)
channel  4: 0.00uW (-inf  dBm)
channel  5: 0.50uW (-33.01dBm)
channel  6: 0.10uW (-40.00dBm)
channel  7: 1.00uW (-30.00dBm)
channel  8: 0.00uW (-inf  dBm)
channel  9: 0.00uW (-inf  dBm)
channel 10: 0.40uW (-33.98dBm)
```

Výpis D.2: COMBO 40G2 (8x10)

```
------ Interface eth0 --------
------   PMA regs   -----------
Link status:                DOWN
Speed:                   10 Gb/s
PMA type:            10GBASE-ER
Transmit Fault:               No
Receive Fault:               Yes
------   PCS regs   -----------
Link status:                DOWN
Speed:                   10 Gb/s
Transmit Fault:              Yes
Receive Fault:                No


------ Interface eth1 --------
------   PMA regs   -----------
Link status:                  UP
Speed:                   10 Gb/s
PMA type:            10GBASE-ER
Transmit Fault:               No
Receive Fault:               Yes
------   PCS regs   -----------
Link status:                DOWN
Speed:                   10 Gb/s
Transmit Fault:              Yes
Receive Fault:                No


------ Interface eth2 --------
------   PMA regs   -----------
Link status:                  UP
Speed:                   10 Gb/s
PMA type:            10GBASE-ER
```

```
Transmit Fault:                 No
Receive Fault:                 Yes
------   PCS regs   -----------
Link status:                 DOWN
Speed:                   10 Gb/s
Transmit Fault:                Yes
Receive Fault:                  No


------ Interface eth3 --------
------   PMA regs   -----------
Link status:                   UP
Speed:                   10 Gb/s
PMA type:             10GBASE-ER
Transmit Fault:                 No
Receive Fault:                 Yes
------   PCS regs   -----------
Link status:                 DOWN
Speed:                   10 Gb/s
Transmit Fault:                Yes
Receive Fault:                  No


------ Interface eth4 --------
------   PMA regs   -----------
Link status:                 DOWN
Speed:                   10 Gb/s
PMA type:             10GBASE-ER
Transmit Fault:                 No
Receive Fault:                 Yes
------   PCS regs   -----------
Link status:                 DOWN
Speed:                   10 Gb/s
Transmit Fault:                Yes
Receive Fault:                  No


------ Interface eth5 --------
------   PMA regs   -----------
Link status:                   UP
Speed:                   10 Gb/s
PMA type:             10GBASE-ER
Transmit Fault:                 No
Receive Fault:                 Yes
------   PCS regs   -----------
Link status:                 DOWN
Speed:                   10 Gb/s
Transmit Fault:                Yes
Receive Fault:                  No


------ Interface eth6 --------
------   PMA regs   -----------
Link status:                   UP
Speed:                   10 Gb/s
PMA type:             10GBASE-ER
Transmit Fault:                 No
Receive Fault:                 Yes
------   PCS regs   -----------
Link status:                 DOWN
Speed:                   10 Gb/s
Transmit Fault:                Yes
Receive Fault:                  No


------ Interface eth7 --------
------   PMA regs   -----------
Link status:                   UP
Speed:                   10 Gb/s
PMA type:             10GBASE-ER
Transmit Fault:                 No
Receive Fault:                 Yes
------   PCS regs   -----------
Link status:                 DOWN
Speed:                   10 Gb/s
Transmit Fault:                Yes
Receive Fault:                  No




---  Transceiver  pmd0 -------
Temperature:             40.54 C
Vendor name:        FINISAR CORP
Vendor PN:            FTL410QE2C
Compliance:          40GBASE-SR4
Connector:                   MPO
Revision:
Wavelength: 850.00nm +-10.00nm

RX input power
channel  1: 0.00uW (-inf  dBm)
channel  2: 0.00uW (-inf  dBm)
channel  3: 0.00uW (-inf  dBm)
channel  4: 0.00uW (-inf  dBm)
```

```
Software TX disable
channel  1:           inactive
channel  2:           inactive
channel  3:           inactive
channel  4:           inactive


---  Transceiver  pmd1 -------
Temperature:          256.00 C
Vendor name:      XXXXXXXXXXXX
Vendor PN:        XXXXXXXXXXXX
Compliance: 40G Active (XLPPI)
Connector: Unknown or Unspecif
Revision:                 XXXX
Wavelength:3276.75nm +-327.5nm

RX input power
channel  1: 6.55mW (18.80 dBm)
channel  2: 6.55mW (18.80 dBm)
channel  3: 6.55mW (18.80 dBm)
channel  4: 6.55mW (18.80 dBm)

Software TX disable
channel  1:             active
channel  2:             active
channel  3:             active
channel  4:             active
```