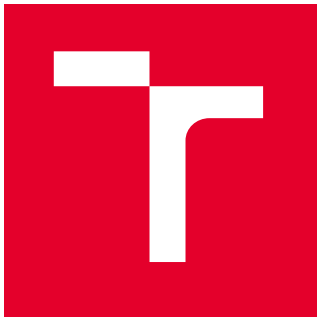


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

KRYPTOGRAFICKÉ ALGORITMY NA PLATFORMĚ FPGA

CRYPTOGRAPHIC ALGORITHMS ON FPGA

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan Broda**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jan Hajný, Ph.D.**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Jan Broda ***ID:*** 203697

Ročník: 2 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Kryptografické algoritmy na platformě FPGA

POKYNY PRO VYPRACOVÁNÍ:

Téma je zaměřeno na implementaci prostředí pro demonstraci kryptografických algoritmů na platformě FPGA, konkrétně s využitím vysokorychlostních síťových karet pracujících rychlostí 100 Gbps. Výstupem práce je demonstrátor schopný přenášet obecná data mezi FPGA a OS i mezi dvěmi síťovými kartami přes síťové rozhraní. Demonstrátor se bude skládat z implementace v jazyce VHDL na straně FPGA a z počítačové aplikace provádějící čtení a zápis dat na straně OS. Demonstrátor bude navržen s ohledem na snadnou modifikovatelnost za účelem vložení kryptografického algoritmu pracujícího nad přenášenými daty.

DOPORUČENÁ LITERATURA:

[1] Menezes, Alfred, Van Oorschot, Paul C. a VANSTONE, Scott A.. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[2] Ashenden, Peter J. The Designer's Guide to VHDL (2nd. ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 978-0120887859.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 24.5.2022

Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato diplomová práce je zaměřena na vytvoření demonstrátoru, který je schopný přenášet data jak mezi operačním systémem a síťovou kartou s FPGA čipem UltraScale+, tak i mezi dvěma síťovými kartami. V teoretické části práce pojednává o programovatelných hradlových polích, vývojem na FPGA, využívanými programovacími jazyky a vývojovém prostředí Vivado Design Suite. Demonstrátor se skládá ze dvou aplikací, vyvíjených v jazyce C, pro komunikaci mezi operačním systémem a síťovou kartou a dvou komponent, vyvíjených v jazyce VHDL, pro komunikaci přes síťové rozhraní na FPGA síťové kartě. Demonstrátor umožňuje vložení kryptografického algoritmu, který by pracoval s přenášenými daty. Pro vývoj na síťové kartě s FPGA čipem byl využit Network Development Kit od týmu Liberouter ze sdružení CESNET.

KLÍČOVÁ SLOVA

fb4CGg3, FPGA, NDK, síťové karty s FPGA čipem, UltraScale+, VHDL, Vivado Design Suite

ABSTRACT

The master thesis is focused on developing a demonstrator which is able to transmit data not only between operating system and network FPGA card with a UltraScale+ chip but also between two network FPGA cards. The theoretical part of the master thesis describes FPGA, developing on FPGA, programming languges that are used and develoment enviroment Vivado Design Suite. The demonstrator consists of two applications developed in C language which are used for communication between operating system and the network FPGA card and two components developed in VHDL langague which are used for communication throught a network module on the network FPGA card. The demonstrator allows inserting cryptographic algorithm which would work with transmitted data. For developing on the network FPGA card was used a Network Development Kit provided by a Liberouter team from CESNET association.

KEYWORDS

fb4CGg3, FPGA, FPGA network cards, NDK, UltraScale+, VHDL, Vivado Design Suite

BRODA, Jan. *Kryptografické algoritmy na platformě FPGA*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 52 s. Diplomová práce. Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Jan Broda
VUT ID autora:	203697
Typ práce:	Diplomová práce
Akademický rok:	2021/22
Téma závěrečné práce:	Kryptografické algoritmy na platformě FPGA

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu semestrální práce panu doc. Ing. Janu Hajnému, Ph.D. za odborné vedení, konzultace a podnětné návrhy k práci. Rovněž bych chtěl poděkovat panu Ing. Petru Jedličkovi za trpělivost, ochotu a cenné rady k praktické části práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Tato diplomová práce se zabývá problematikou spojenou s vysokorychlostními síťovými kartami s FPGA (Field Programmable Gate Array) čipem, které je možné využívat jako hardwarový akcelerátor pro kryptografické algoritmy.

První kapitola práce se věnuje úvodu do programovatelných hradlových polí. Je zde popsána základní architektura FPGA čipů a její jednotlivé prvky. V závěru kapitoly jsou popsány možné alternativy k FPGA čipům.

Druhá kapitola se zabývá vývojem na FPGA zařízeních. Jsou zde popsány a porovnány základní programovací jazyky využívané při vývoji na FPGA. Další část kapitoly se věnuje jazyku VHDL (Very High Speed Integrated Circuit Hardware Description Language), který je využíván pro programování komponent v praktické části práce. Rovněž jsou v této kapitole popsány jednotlivé fáze, ze kterých je vývoj složen.

Třetí kapitola popisuje vývojové prostředí Vivado Design Suite od společnosti Xilinx včetně jeho komponent a edicí.

Čtvrtá kapitola je věnována specifikaci vysokorychlostní síťové karty fb4CGg3. Obsahuje popis parametrů karty, pamětí a síťového rozhraní.

V páté kapitole je popsán vývojový nástroj NDK (Network Development Kit) od týmu Liberouter ze sdružení CESNET. Je zde popsána jeho struktura a moduly, které jsou důležité pro vývoj na síťových kartách s FPGA čipem. Dále je popsán protokol FrameLink Unaligned a jeho signály. V závěru kapitoly jsou vypsány užitečné nástroje pro komunikaci s NDK.

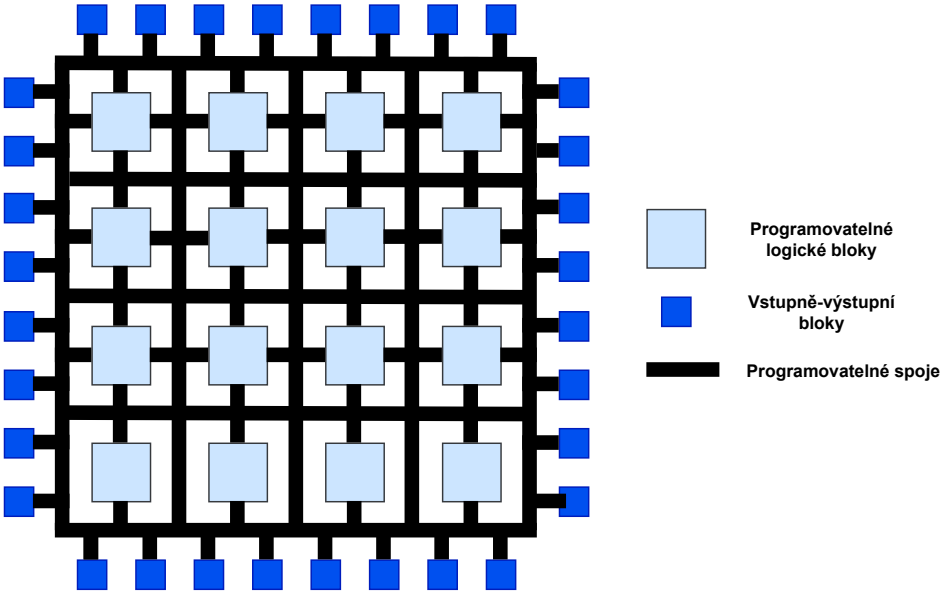
Poslední šestá kapitola je věnována praktické implementaci. V první části kapitoly je popsáno zprovoznění NDK na FPGA kartě a předpoklady, které musí být pro zprovoznění splněny. Dále jsou zde popsány aplikace pro komunikaci mezi operačním systém a síťovou kartou. Rovněž jsou zde popsány komponenty pro přenos dat přes síťové rozhraní. Následně je zde část věnující se otestovaní vytvořené implementace na síťových kartách. Tato část obsahuje názorné obrázky a výpisy, které znázorňují funkčnost celého demonstrátoru. V závěru kapitoly je uvedena využitelnost zdrojů a omezení.

1 Programovatelná hradlová pole

Programovatelná hradlová pole neboli FPGA patří do skupiny polovodičových integrovaných obvodů, které jako první představila v roce 1984 společnost Xilinx. Hlavní předností FPGA čipů je jejich přeprogramovatelnost, kdy uživatel je schopen kdykoli změnit konfiguraci čipů, a tím i jejich funkcionalitu. Díky této skutečnosti nachází uplatnění v různých odvětvích technologického průmyslu.

1.1 Architektura

FPGA čipy jsou složeny ze tří částí, a to vstupně-výstupních bloků, programovatelných logických bloků a programovatelných spojů [1]. Architektura FPGA čipu je znázorněna na obr. 1.1.

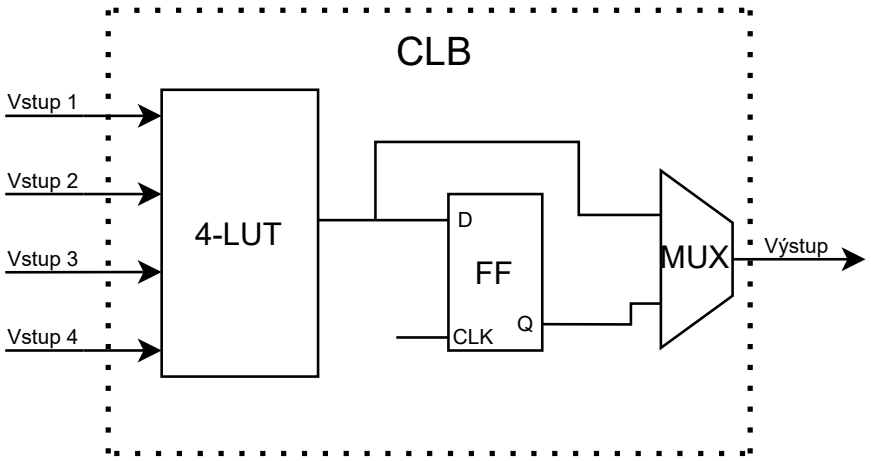


Obr. 1.1: Architektura FPGA čipu

1.1.1 CLB

CLB (Configurable Logic Blocks) jsou programovatelné logické bloky, které po naprogramování realizují určitou logickou funkci. CLB jsou tvořeny z několika Slice. Každý Slice se skládá z vyhledávacích tabulek LUT (Lookup Tables), multiplexoru (MUX) a klopného obvodu typu D (Flip-Flop) řízeného hodinovým signálem. Počet Slice je rozdílný v závislosti na použitém čipu. Slice lze rozdělit do dvou rozdílných druhů, a to SliceL a SliceM. Vyhledávací tabulky u SliceM lze nakonfigurovat jako posuvný registr či RAM (Random Access Memory) paměť. Pro příklad každý

Slice FPGA čipu Xilinx UltraScale+ obsahuje osm šestivstupových LUT tabulek a šestnáct klopných obvodů. Na obr. 1.2 je znázorněna základní podoba CLB [1][2].



Obr. 1.2: Základní CLB

LUT

Vyhledávací tabulky slouží k realizaci logické funkce, kdy na základě zadaného vstupu vygenerují patřičný výstup. Prakticky se jedná o RAM paměť, kde vstupy slouží jako adresa odkazující na paměťovou buňku. Počet vstupů LUT tabulky může být u různých čipů odlišný, avšak ve většině případů jsou LUT tabulky čtyřvstupové a šestivstupové [2].

Klopné obvody

Klopné obvody jsou společně s LUT základním stavebním prvkem CLB. Nejčastějším typem klopného obvodu v FPGA čipech je klopný obvod typu D. Klopný obvod se skládá z několika pinů, kdy pin D slouží jako vstup, pin Q jako výstup, CLK jako vstup hodinového signálů a R jako reset. Klopné obvody mohou být řízeny jak staticky, tak dynamicky. V převážné většině případů je ve FPGA zařízeních klopný obvod řízený dynamicky, tj. náběžnou hranou hodinového signálu. Ve výsledku se jedná o vnitřní paměť, která je schopná uchovat informaci po dobu jednoho cyklu hodinového signálu [2][3].

Multiplexory

Další součástí CLB jsou multiplexory, které jsou v různých velikostech používány ve všech FPGA čipech. Multiplexor je obvod, který na základě řídícího signálů volí, jaký ze vstupních signálů bude přiveden na výstup [2].

1.1.2 Programovatelné logické spoje

Programovatelné spoje poskytují propojení vstupně-výstupních bloků a konfigurovatelných logických bloků. Rovněž mezi sebou propojují jednotlivé CLB bloky. Jsou realizovány pomocí multiplexorů, třístavové vyrovnávací paměti a tranzistorů [1].

1.1.3 Vstupně-výstupní bloky

IOB (Input-Output Blocks) neboli vstupně-výstupní bloky poskytují rozhraní pro propojení vnějšího prostředí s vnitřní FPGA architekturou. IOB obsahují například zpožďovací prvky, vstupní a výstupní registry, zesilovač či linku pro reset [1].

1.2 Alternativy k FPGA

Alternativou ke FPGA čipům jsou čipy CPLD (Complex Programmable Logic Device) a ASIC (Application Specific Integrated Circuit). Čipy CPLD jsou tvořeny z programovatelných buněk, které jsou propojeny programovatelnými spoji. Jedná se o podobné pojetí architektury jako u čipů FPGA. CPLD čipy využívají integrovanou nevolatilní paměť, a tudíž oproti FPGA dokáží udržet konfiguraci i po odpojení od zdroje energie. Další rozdíl je v realizaci logických funkcí, kdy FPGA buňka je založena na LUT tabulkách, zatímco CPLD využívá SoP (Sum of Products). CPLD obvody je vhodné používat pro realizaci jednodušších logických funkcí, přičemž při realizaci komplexnějších funkcí je vhodné využít FPGA, které mají oproti CPLD daleko více logických a paměťových prostředků. [4].

ASIC jsou zákaznické integrované obvody, které jsou vyráběny pro specifické aplikace. Zásadním rozdílem, v porovnání s FPGA, je nemožnost jejich opětovného překonfigurování. Dalším rozdílem jsou vysoké jednorázové předvýrobní náklady, které zahrnují především náklady na návrh a vývoj. Z tohoto důvodu jsou ASIC obvody určeny převážně pro velkovýrobu. Jejich výhodou je větší energetická úspornost a vyšší rychlost [5].

2 Vývoj na FPGA

Pro popis chování FPGA zařízení slouží HDL (Hardware Description Language) programovací jazyky. Vývojová prostředí pro FPGA vývoj většinou umožňují syntézu a implementaci vytvořeného návrhu. Rovněž poskytují možnost odsimulovat vytvořený návrh v samotném vývojovém prostředí ještě před finálním nahráním na samotný FPGA čip, což je pro vývoj velice nápomocné a důležité. Převážně u ASIC čipů by absence simulace při vývoji znamenala problém. Pokud by výsledný návrh nebyl správně odsimulován, tak by se kvůli nemožnosti překonfigurování ASIC čipů musel, v případě chybného návrhu, vyrobit zcela nový čip.

Běžně využívanými HDL programovacími jazyky jsou VHDL a Verilog. Mezi oběma jazyky je několik rozdílů. VHDL není citlivý na velká písmena, vychází z jazyku Pascal a Ada, používá různé datové typy (včetně uživatelsky vytvořených), je silně typovaný a je složitější na pochopení.

Verilog je citlivý na velká písmena, vychází z jazyka C, má pouze pár základních datových typů, je slabě typovaný a je jednodušší na pochopení [6].

2.1 VHDL

VHDL patří mezi HDL programovací jazyky, které slouží pro popis struktury, chování a simulaci hardwaru např. FPGA, CPLD či ASIC čipů. Instrukce psané v jazyce VHDL jsou prováděny paralelně.

2.1.1 Základní struktura kódu

Základní struktura VHDL kódu je tvořena deklarací knihoven, entity a popisu architektury [7].

Knihovny a balíčky

Knihovna ve VHDL je soubor balíčků, které definují datové typy, funkce, procedury, uživatelem specifikované datové typy či konstanty. Knihovna je deklarována pomocí příkazu `library` a balíček pomocí příkazu `use`.

Entity

Blok entity popisuje komponentu pomocí vstupu a výstupů, které jsou brány jako porty. Každému z deklarovaných portů je nutné přiřadit jméno a datový typ, se kterým bude port pracovat. Rovněž je důležité přiřadit portu jeden z operačních módů. Mezi tyto módy patří:

- In – vstupní port, určený pouze pro čtení dat,
- Out – výstupní port,
- InOut – vstupně-výstupní port, slouží pro obousměrný tok dat.

Architektura

Blok architektury slouží pro popis chování čipu. Popis lze realizovat třemi různými způsoby [8]:

- Strukturální popis – Při strukturálním popisu se propojují již existující dílčí entity, které se nacházejí například v knihovnách. Představuje popis samotného zapojení entit v obvodu, kdy lze jednotlivé entity samostatně odladit. Nevýhodou strukturálního popisu je náročnost na implementaci a malá flexibilita. Výhodou je vysoká kontrola nad syntézou.
- Behaviorální popis – Pro popis chování čipu slouží behaviorální způsob. Tento způsob je často užíván při tvorbě simulací. Oproti strukturálnímu popisu není známo zapojení obvodu, ale jsou známy pouze vstupy a výstupy (tzv. black box). Chování je při tomto způsobu popsáno jako proces, ve kterém jsou instrukce vykonávány sekvenčně. Díky tomu se tvorba behaviorálního postupu přibližuje tradičnímu způsobu programování. Běžně jsou využívány různé typy příkazů a konstrukcí jako například `if - else`, `case`, `wait` či `loop`. Mezi výhody behaviorálního postupu patří jednoduchost implementace a velká flexibilita. Nevýhodou je nízká kontrola nad syntézou.
- RTL (Register Transfer Level) popis - RTL popis je způsob popisu činnosti čipu, při kterém je funkce obvodu sestavena pomocí Booleovy logiky. Popisuje tok dat mezi registry a jejich změnu způsobenou logickými hradly. Stejně jako u behaviorálního popisu, tak i u RTL popisu není známo zapojení obvodu. Rozdílem je skutečnost, že je známo alespoň zapojení logických hradel.

V praxi je zcela běžná kombinace všech těchto způsobů popisu v rámci jednoho zdrojového kódu tak, aby se dosáhlo co nejvyšší efektivity.

2.2 Postup při vývoji na FPGA

Vývoj na FPGA zařízeních se skládá ze čtyř kroků, a to z návrhu, syntézy, implementace a finálního nahrání programu na FPGA.

2.2.1 Návrh

Prvním krokem vývoje je návrh samotného systému, který se bude na FPGA implementovat. Návrh může být zhotoven jak schematicky, tak pomocí HDL jazyků, případně kombinací obou postupů. Schematický návrh poskytuje větší kontrolu nad

návrhem, ale z důvodu časové náročnosti je vhodné jej využívat pro méně komplexní návrhy. Naopak pro tvorbu složitějších návrhů je vhodnější zvolit způsob s využitím HDL jazyků, pomocí kterých je realizace návrhů podstatně rychlejší [9].

2.2.2 Syntéza

Syntézou se rozumí transformace vytvořeného návrhu na reálný obvod pomocí syntetizéru. Výsledkem syntézy je seznam (tzv. netlist) obsahující všechny prvky a jejich propojení, která jsou potřebná pro realizaci požadovaného systému. Syntéza rovněž zajišťuje verifikaci správné syntaxe a optimalizaci navrženého obvodu. Syntéza může být uskutečněna pomocí různého softwaru, jakým je například Vivado Design Suite od firmy Xilinx [9].

2.2.3 Implementace

Proces implementace je zprostředkován pomocí nástrojů, které poskytují výrobci daného FPGA. Zahrnuje překlad vytvořených netlistů a uživatelem zadaných omezení do jednoho souboru. Výsledný soubor je typu NGD (Native Generic Database) a je v něm například uloženo přiřazení pinů nebo perioda vstupního hodinového signálu. Následně je obvod rozdělen do několika menších bloků, které je možné přiřadit do logických bloků daného FPGA. Snahou nástrojů pro implementaci je optimalizace rozmístění každého z přiřazovaných bloků tak, aby výsledné propojení bylo co nejefektivnější [9].

2.2.4 Nahrání programu

Závěrečným krokem je samotné nahrání výsledného návrhu na FPGA. Pro nahrání je obvykle potřeba převést výsledný návrh na bitový proud (tzv. bitstream), který již lze nahrát na samotné FPGA.

3 Xilinx - Vivado Design Suite

Vivado Design Suite od společnosti Xilinx je nástupcem zastaralého a již nevyužívaného vývojového prostředí Xilinx ISE (Integrated Synthesis Environment). Vivado je efektivní vývojové prostředí pro programovatelné hradlové pole. Umožňuje kompletní vývoj programovatelných polí včetně návrhu v programovacích jazycích VHDL či Verilog, syntézy vytvořeného kódu i samotné implementace na fyzické zařízení. Rovněž je možné simulovat funkčnost návrhu pomocí zabudovaného simulátoru ISim. Interakce s Vivadem je možná jak pomocí grafického rozhraní, tak i s využitím TCL (Tool Command Language) nebo případně i kombinací obou možností [10].

3.1 Komponenty

Vivado Design Suite obsahuje různé vývojové komponenty, mezi něž patří Vivado HLS (High-Level Synthesis), Vivado Intellectual Property Integrator a Vivado Simulator ISim.

3.1.1 Vivado HLS

Vivado HLS umožňuje konverzi návrhů psaných v jazyce C/C++ či SystemC na podobu vhodnou pro syntézu. Díky této možnosti je vývoj složitějších systému jednodušší a rychlejší než za použití standardních HDL programovacích jazyků. Nevýhodou je nízká kontrola nad samotnou syntézou[11].

3.1.2 Vivado Intellectual Property Integrator

Poskytuje možnost vkládat do návrhu již existující IP bloky, což jsou bloky, které implementují určitou logickou funkci. Seznam již vytvořených IP bloků lze nalézt ve Vivado IP katalogu [12].

3.1.3 Vivado Simulator ISim

Simulátor dovoluje ověřit funkčnost vytvořeného návrhu před finální implementací na hardware. V grafickém výstupu simulace lze zobrazit všechny v návrhu definované signály a ověřit, zda plní očekávanou funkci. ISim podporuje různé programovací jazyky a rovněž i TCL skripty [13].

3.2 Edice Vivada

Aktuálně Xilinx poskytuje Vivado Design Suite ve dvou edicích, a to Vivado ML Standard a Vivado ML Enterprise.

Edice Standard je volně dostupná verze vývojového prostředí Vivado. Obsahuje všechny základní komponenty a funkcionality potřebné pro vývoj na určitých Xilinx zařízeních.

Edice Enterprise je placená verze vývojového prostředí Vivado, jejíž cena začíná na necelých třech tisících dolarů. Rozdílem oproti volně dostupné verzi je podpora všech Xilinx zařízení.

4 Vysokorychlostní síťová karta fb4CGg3

Karta fb4CGg3 [14] společnosti Silicom Denmark je vysokorychlostní síťová karta se zabudovaným FPGA čipem pracující rychlostí až 100 Gbit/s skrze každý ze čtyř QSFP28 (Quad Small Form-factor Pluggable) portů. Kartu lze osadit FPGA čipem Virtex UltraScale+ nebo popřípadě i starším čipem Virtex UltraScale. Čip UltraScale+ podporuje sběrnici PCI (Peripheral Component Interconnect) Express 3.0 x16 (UltraScale podporuje pouze PCI-express 3.0 x8) a RS-FEC (Reed Solomon Forward Error Correction) pro opravu chyb při síťovém přenosu. Základní hodinový signál, který karta podporuje, má frekvenci 50 MHz.

Kartu lze využít u všech systémů, které vyžadující nízkou latenci a vysoký výkon. Příkladem použití může být hardwarový akcelerátor pro výpočetně náročné kryptografické algoritmy, streamování videa či kompresi. Podoba karty je vyobrazena na obr. 4.1.

Obr. 4.1: Vysokorychlostní síťová karta fb4CGg3 [14]

4.1 Paměť

Karta je osazena dvěma integrovanými 4 GB paměťovými moduly DDR4 (Double Data Rate), které pracují rychlostí 2400 MT/s. Dále karta disponuje dvěma sokety SODIMM (Small Outline Dual In-line Memory Module), které lze osadit dalšími DDR4 moduly, případně i moduly QDRII+ (Quad Data Rate). Rovněž obsahuje konfigurovatelnou 8 MB flash paměť sloužící jakožto uživatelské úložiště a 256 MB

konfigurovatelnou flash paměť pro zavaděč systému. Paměťová rozhraní pracují s hodinovým signálem s frekvencí 266,66 MHz.

4.2 Síťové rozhraní

Karta poskytuje síťové připojení o rychlosti až 100 Gbit/s na každém z QSFP28 portů. U každého ze čtyř portů lze nastavit jednu z podporovaných rychlostí, a to 4x 10 Gbit/s, 4x 25 Gbit/s, 1x 40 Gbit/s, 2x 50 Gbit/s a 1x 100 Gbit/s. Všechny porty lze spravovat pomocí I2C (Inter-Integrated Circuit) kontroléru, kdy každý port je na kontrolér napojený pomocí I2C sběrnice. Každý z portů pracuje s hodinovým signálem o frekvenci 161,13 MHz.

4.3 FPGA čipy

Ve standardní konfiguraci je síťová karta osazena FPGA čipem Virtex UltraScale+ VU09P. Tato konfigurace byla rovněž využita v praktické části práce. Síťové karty je možné osadit i jinými verzemi FPGA čipů. Jedná se zejména o verze VU07P, VU190, VU125 a VU80. Srovnání zdrojů jednotlivých využívaných čipů je uvedeno v tab. 4.1.

Tab. 4.1: Zdroje využívaných FPGA čipů

Čipy	**VU80**	**VU125**	**VU190**	**VU07P**	**VU09P**
Flip-Flop	891424	1432320	2148480	1576320	2364480
LUT	445712	716160	1074240	788160	1182240
Bloky BRAM	1421	2520	3780	1440	2160
BRAM (Mb)	50	88,6	132,9	50,6	75,9
I/O	780	780	650	832	832

4.4 Ostatní prvky

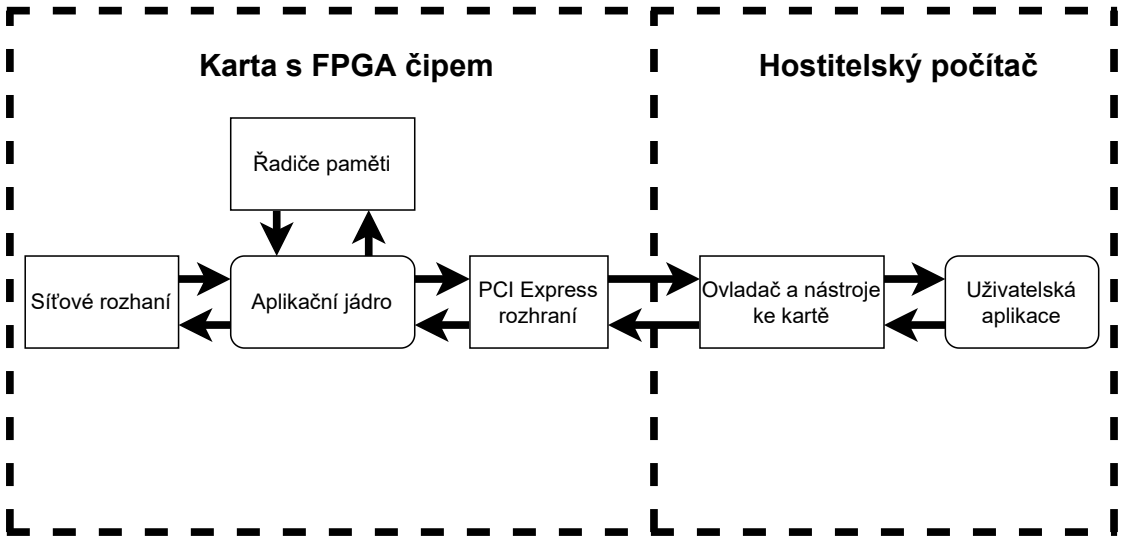
Mezi další prvky, kterými karta disponuje, patří například kontrolér konfigurovatelných LED (Light-Emitting Diode) diod, teplotní senzory, monitoring větráků či jednoznačný identifikátor karty.

5 Network Development Kit

Network Development Kit (NDK) je vývojový nástroj vytvořený týmem Liberouter ze sdružení CESNET. Umožňuje uživateli jednoduchý, rychlý a efektivní vývoj na vysokorychlostních síťových kartách s FPGA čipem, které podporují rychlosti až 400 Gbit/s. Struktura NDK je rozdělena na část určenou pro FPGA a část určenou pro hostitelský počítač. Komunikace mezi FPGA síťovou kartou a hostitelským počítačem je zprostředkována pomocí sběrnice PCI Express Gen5 x16 s propustností až 400 Gbit/s. Hlavní části NDK frameworku jsou [15]:

- Vstupně-výstupní rozhraní - slouží pro přenos a příjem dat. Použitá technologie a šířka pásma se může u jednotlivých rozhraní lišit.
- Přístup z hostitelské aplikace do jádra - poskytuje podporu pro zápis a čtení dat z jádra FPGA čipu přes sběrnici PCI Express.
- Přenos dat mezi hostitelskou aplikací a jádrem FPGA čipu - umožňuje vysokorychlostní přenos dat přes DMA (Direct Memory Access) modul. K datům lze přistoupit například pomocí aplikačního rozhraní PCAP (Packet Capturing), které slouží pro odchytávání síťové komunikace.

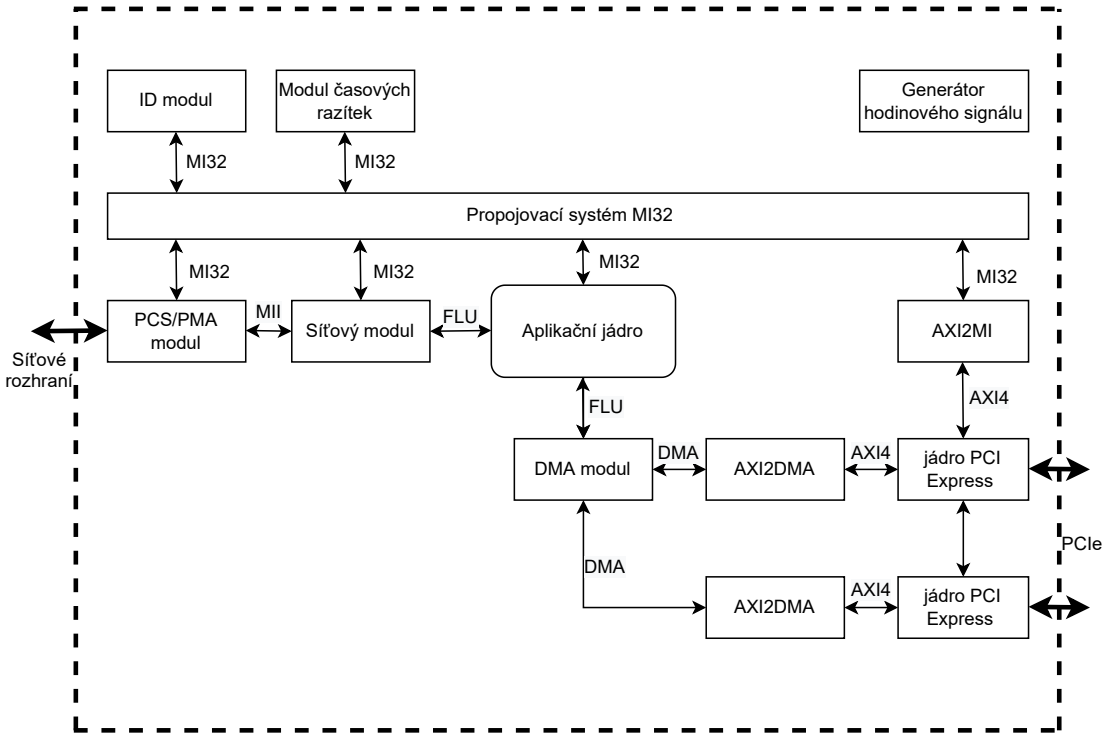
Struktura NDK je znázorněna na obr 5.1.



Obr. 5.1: Struktura NDK

5.1 Hlavní NDK moduly

NDK je složeno z dílčích modulů (např. DMA modul, modul časových razítek či síťový modul) [15]. Moduly jsou vytvářeny buď pomocí HDL jazyků, nebo pomocí IP generátoru. Na obr 5.2 jsou znázorněny NDK moduly a jejich propojení.



Obr. 5.2: NDK moduly

5.1.1 Aplikační jádro

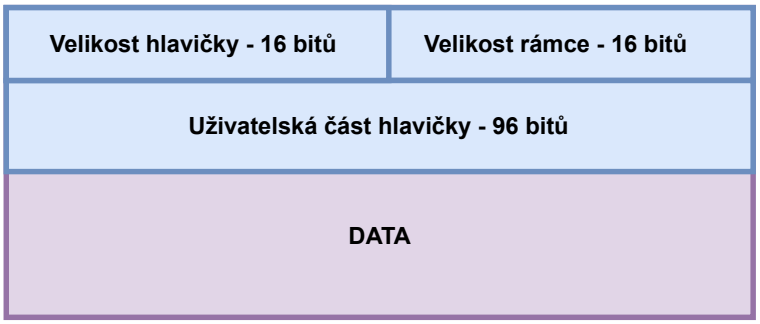
Pro vývoj pomocí NDK uživateli stačí upravit, z celé NDK architektury, pouze aplikační jádro, kam vloží vytvořenou komponentu. Pro odesílání a příjem dat je v aplikačním jádře umístěn síťový a DMA modul. Řízení aplikačního jádra z hostitelského počítače je realizováno pomocí MI32 (Memory Interface 32-bit) propojovacího systému. Propojovací systém je s aplikačním jádrem propojen pomocí sběrnice MI32, po které se přenáší adresy registrů. Jedná se převážně o stavové a řídící registry. Registry jsou přístupné skrze jejich adresy a dle požadavků hostitelského počítače lze provést operaci čtení či zápisu.

Při zasílání dat z hostitelského počítače do FPGA karty jsou data do aplikačního jádra přijaty z DMA modulu přes FLU (Frame Link Unaligned) sběrnici. Pokud je potřeba příchozí data editovat, tak k FLU rozhraní lze připojit komponentu, která bude s příchozími daty provádět požadovanou operaci.

Data přicházející z operačního systému přes DMA modul neobsahují Ethernetovou hlavičku. Při zasílání dat přes síťový modul je potřeba tuto hlavičku k datům připojit. Ethernetová hlavička je ve formátu - cílová MAC (Media Access Control) adresa (6 B) + zdrojová MAC adresa (6 B) + EtherType (2 B). Pole EtherType určuje typ přenášených dat, např. pro IPv4 (Internet Protocol version 4) by EtherType odpovídal hodnotě 0x0800. Celková velikost hlavičky je 14 bajtů. Společně

s daty je síťovému modulu odesláno i číslo síťového rozhraní, které bude pro přenos využito.

Data přicházející ze síťového rozhraní přes síťový modul jsou zaslána do aplikačního jádra. K datům je v síťovém modulu vytvořena tzv. NDP (Netcope Data Plane) hlavička, která poskytuje rozšiřující informace o přijatých datech. Celková velikost NDP hlavičky je 128 bitů. Prvních 16 bitů je označováno jako `Segment size`, který specifikuje velikost celého rámce v bajtech. Dalších 16 bitů určuje velikost NDP hlavičky v bajtech a je označováno jako `Hardware Size`. Zbylých 96 bitů tvoří uživatelská část hlavičky, která obsahuje číslo DMA rozhraní, číslo Ethernetového rozhraní a časovou známku. NDP hlavička je ze síťového modulu zaslána do aplikačního jádra, kde je k datům připojena pomocí komponenty HINS_PLUS. Struktura přenášených dat ze síťového rozhraní je znázorněna na obr. 5.3.



Obr. 5.3: Struktura NDK rámce

5.1.2 Síťový modul

Síťový modul posílá data přijaté z PCS/PMA (modul, který zasílá data ze síťového rozhraní do síťového modulu) do aplikačního jádra přes FLU rozhraní. Síťový modul je připojen sběrnicí MI32 k propojovacímu modulu, který řídí tok dat a rovněž zprostředkovává statistiku přijatých a odeslaných dat.

5.1.3 DMA

DMA modul umožňuje vysokorychlostní oboustranný přenos dat mezi hostitelským počítačem a FPGA kartou. Rovněž jej lze využít i pro vzájemnou komunikaci mezi kartami. Vysoká rychlost přenosu spočívá v přímém přístupu do paměti, kdy data neprocházejí přes CPU (Central Processor Unit). Data jsou přenášena mezi vyrovnávacími paměťmi, kdy jedna je umístěna v RAM hostitelského počítače a druhá na kartě v FPGA čipu. Maximální propustnost DMA modulu je závislá na verzi použité PCI sběrnice.

5.2 FrameLinkUnaligned protokol

FLU je komunikační protokol využívány převážně pro realizaci sběrnic. V NDK je FLU sběrnice použita pro komunikaci mezi aplikačním jádrem a síťovým modulem a také pro komunikaci mezi aplikačním jádrem a DMA modulem. Pro řízení komunikace slouží sada signálů, které jsou blíže popsány v tab. 5.1.

Tab. 5.1: Signály FLU protokolu

Signál	Strana	Popis signálu
SOP	odesílatel	Signál Start Of Packet slouží k indikaci začátku dat.
EOP	odesílatel	Signál End Of Packet slouží k indikaci konce dat.
SOP_POS	odesílatel	Signál Start Of Packet Position je tříbitový signál, který indikuje první bajt přenášených dat.
EOP_POS	odesílatel	Signál End Of Packet Position je šestibitový signál, který indikuje poslední bajt přenášených dat.
SRC_RDY	odesílatel	Signál Source Ready indikuje, zda je odesílatel připraven vysílat.
DST_RDY	příjemce	Signál Destination Ready indikuje, zda je příjemce připraven přijímat.
DATA	odesílatel	Signál DATA reprezentuje přenášená data. Obvyklá délka signálu DATA je 512 bitů.

5.3 Nástroje

Nástroje pro komunikaci s NDK skrze CLI (Command Line Interface) jsou společně s ovladačem pro kartu umístěny v balíčku s názvem `netcope-common`. Tabulka 5.2 obsahuje výčet NDK nástrojů a jejich popis.

Tab. 5.2: Nástroje NDK

Nástroj	**Popis**
`nfb-info`	Slouží pro výpis informací o kartě a použitém firmwaru.
`nfb-bus`	Umožňuje čtení a zápis do registrů (převážně pro ladění).
`nfb-boot`	Slouží pro zápis vygenerovaného bitstreamu na FPGA kartu.
`nfb-eth`	Slouží pro konfiguraci síťových rozhraní, konfiguraci minimální a maximální délky dat a sledování statistiky síťového provozu.
`nfb-dma`	Umožňuje zobrazit statistiku provozu v DMA modulu.
`ndp-read`	Slouží pro příjem dat z karty přes DMA.
`ndp-receive`	Stejně jako ndp-read slouží pro příjem dat z karty s tím rozdílem, že data lze ukládat do PCAP souboru.
`ndp-generate`	Generuje data s hodnotou 0, která jsou následně pomocí DMA zaslána na kartu.
`ndp-transmit`	Umožňuje posílat data specifikovaná v PCAP souboru na kartu.
`ndp-loopback`	Umožňuje posílat data ve smyčce, tj. z OS do FPGA a zpět.

6 Praktická implementace

V praktické části diplomové práce je řešeno vypracování demonstrátoru, který je schopen přenášet data mezi operačním systémem a FPGA kartou a rovněž i mezi dvěma FPGA kartami přes síťové rozhraní. Pro komunikaci mezi operačním systémem a FPGA kartou slouží aplikace pro čtení a zápis dat, které jsou implementovány v jazyce C. Pro komunikaci přes síťové rozhraní byly vytvořeny komponenty v jazyce VHDL.

6.1 Zprovoznění NDK na FPGA kartě

6.1.1 Předpoklady

Předpokladem pro zavedení NDK na kartu je úspěšné vygenerování bitstreamu. K jeho vygenerování lze využít vývojové prostředí Vivado. Překladové servery sdružení CESNET využívají Vivado ve verzi 2019.1, kdy generování bitstreamu pomocí jiné verze než 2019.1 může skončit chybou. Například u verze 2020.2, která je společně s verzí 2019.1 dostupná na školním Vivado serveru, generování skončí chybou kvůli problému se signály v souboru `cmac_wrapper.vhd`. Vivado 2019.1 využívá v souboru `cmac_wrapper .vhd` jednobitové signály, zatímco Vivado 2020.2 tyto signály používá ve formě vektoru. Z tohoto důvodu je potřeba pro úspěšné vygenerování bitstreamu ve verzi 2020.2 z příslušných jednobitových signálů vytvořit vektor.

Druhým předpokladem je nainstalování ovladače a NDK nástrojů na kartu. Ovladač je obsažen v balíčku `netcope-common`, který je dostupný jak pro CentOS, tak i pro Debian-based systémy. Pro správnou funkci nástrojů je důležité, aby karta již disponovala NDK firmwarem, který NDK nástroje podporuje. Pokud tomu tak není, je možné NDK firmware nahrát do flash paměti karty manuálně pomocí JTAG (Joint Test Action Group) portu.

6.1.2 Zavedení NDK na FPGA kartu

Prvním krokem při zavádění NDK na kartu je naklonování příslušných repozitářů z Liberouter GitLabu na kartu. Jedná se zejména o hlavní repozitář `fwbase` a jeho submodul `OFM`. Repozitáře lze z GitLabu na server s Vivadem naklonovat pomocí příkazu `git clone`. Následně již lze přistoupit do příslušného adresáře dané síťové karty a příkazem `make` spustit generování bitstreamu. Pro kartu fb4CGg3 se jedná o adresář `fwbase/applications/nic/mango`. Po dokončení překladu se ve stejné

složce objeví soubor `fb1cgg_nic.nfw`, což je `tar.gz` archív, který obsahuje příslušný bitstream a konfiguraci frameworku. Na závěr stačí vygenerovaný soubor `fb1cgg_nic.nfw` nahrát pomocí nástroje `nfb-boot` na kartu. Úspěšné nahrání firmwaru lze zkontrolovat pomocí nástroje `nfb-info`, jehož výstup je znázorněn ve výpisu 6.1.

Výpis 6.1: Výpis terminálu - nástroj nfb-info

```
server@fpga2:~$ nfb-info
---------------------------------------- Board info ----
Card name                    : FB4CGG3
Serial number                : 785
Network interfaces           : 2
------------------------------------ Firmware info ----
Project name                 : NIC_FB1CGG_100GE
Built at                     : 2022-05-04 10:21:07
Build tool                   : Vivado v2019.1 (64-bit)
RX queues                    : 2
TX queues                    : 2
ETH channels                 : 2
-------------------------------------- System info ----
PCI slot                     : 0000:47:00.0
NUMA node                    : 0
```

6.2 Komunikace mezi operačním systémem a FPGA kartou

Ke komunikaci mezi operačním systémem a FPGA kartou slouží aplikace pro zápis a čtení dat realizované v jazyce C. Aplikace využívají knihovnu `libnfb`, která poskytuje všechny funkce potřebné pro sestavení komunikace. Jedná se zejména o funkce:

- `nfb_open(const char *path)` – inicializační funkce, která musí být volána jako první před ostatními funkcemi knihovny,
- `nfb_close(struct nfb_device *dev)` – volá se po dokončení práce se zařízením,
- `ndp_open_rx_queue(struct nfb_device *nfb, unsigned queue_id)` – slouží k otevření přijímací (RX) fronty,
- `ndp_open_tx_queue(struct nfb_device *nfb, unsigned queue_id)` – slouží k otevření vysílací (TX) fronty,

- `ndp_close_rx_queue(ndp_rx_queue_t *queue)` – slouží k uzavření přijímací (RX) fronty,
- `ndp_close_tx_queue(ndp_tx_queue_t *queue)` – slouží k uzavření vysílací (TX) fronty,
- `ndp_queue_start(struct ndp_queue *queue)` – slouží k zahájení přenosu dat skrze danou frontu,
- `ndp_queue_stop(struct ndp_queue *queue)` – slouží k ukončení přenosu dat skrze danou frontu,
- `ndp_tx_burst_get(ndp_tx_queue_t *queue, struct ndp_packet *packets, unsigned count)` – pomocí této funkce je alokováno místo pro data ve vysílací (TX) frontě,
- `ndp_tx_burst_flush(ndp_tx_queue_t *queue)` – slouží pro zapsání dat do vysílací (TX) fronty,
- `ndp_rx_burst_get(ndp_rx_queue_t *queue, struct ndp_packet *packets, unsigned count)` – přečte data z přijímací (RX) fronty,
- `ndp_rx_burst_put(ndp_rx_queue_t *queue)` - ukončení čtení z přijímací (RX) fronty.

Přenášená data jsou ukládána do předem definované struktury. Tato struktura obsahuje položky, kterými jsou:

- délka dat (`data_length`),
- data (`*data`),
- délka hlavičky (`header_length`),
- hlavička (`*header`).

6.2.1 Aplikace pro odesílání dat na FPGA kartu

Pro odesílaní dat na FPGA kartu slouží aplikace `transmit.c`. Uživatel, který chce zaslat určitá data na síťovou kartu, vloží tato data do textového souboru v hexadecimální podobě. Data je možné v textovém souboru, pro větší přehlednost, rozdělit na několik řádků. Jelikož jsou data v textovém souboru reprezentována pomocí ASCII (American Standard Code for Information Interchange) znaků, je potřeba tato data převést do takové podoby, aby odpovídala požadovaným hexadecimálním hodnotám. K tomuto slouží vytvořená funkce `ascii_to_bin`, která daný ASCII znak převede na znak odpovídající požadované hexadecimální hodnotě. Pokud nejsou data uživatelem zadána v hexadecimálním formátu, není přenos možný. Funkce pro převod dat na hexadecimální hodnoty je uvedena ve výpisu 6.2.

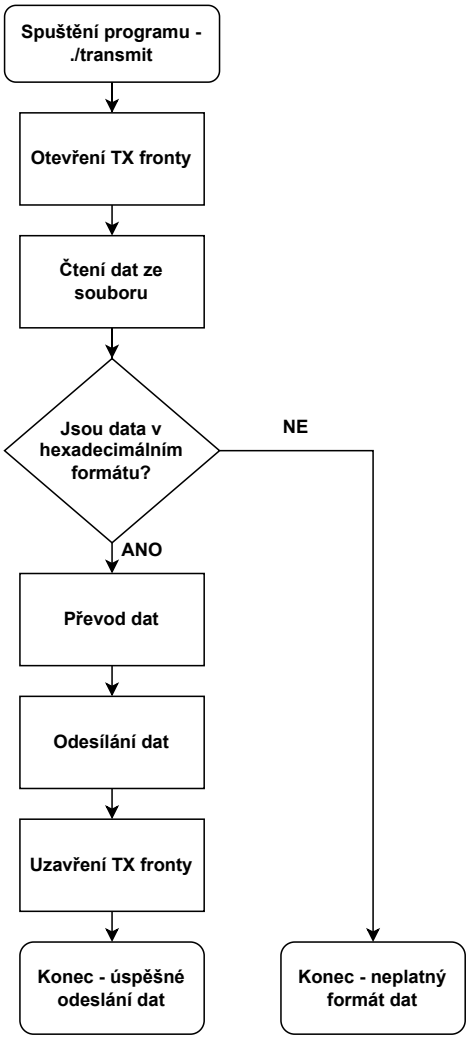
Výpis 6.2: Funkce pro převod přenášených dat

```
unsigned char ascii_to_bin(unsigned char in)
{
```

```
    if ((in >= '0') && (in <= '9')) {
        return in - '0';
    }
    if ((in >= 'a') && (in <= 'f')) {
        return (in - 'a') + 10;
    }
    if ((in >= 'A') && (in <= 'F')) {
        return (in - 'A') + 10;
    }
    return 0;
}
```

Pokud by pro příklad byl vstupem funkce pro převod znak `f`, který je v ASCII podobě reprezentován decimální hodnotou 70, je potřeba jej převést na znak, který bude odpovídat decimální hodnotě 15, tzn. hodnotě `f` v hexadecimálním tvaru. Toho lze docílit odečtením hodnoty znaku `a`, tj. hodnoty 65 od vstupní hodnoty, tj. hodnoty 70. K výslednému rozdílu se následně přičte konstanta 10, a tím se vytvoří požadovaný převod.

Aplikace pro odesílání dat na síťovou kartu používá funkce dostupné v knihovně `libnfb`. Tyto funkce jsou deklarovány v hlavičkových souborech `<nfb/ndp.h>` a `<nfb/nfb.h>`. Popis jednotlivých funkcí dostupných v daných hlavičkových souborech je uveden v kapitole 6.2. Vývojový diagram aplikace pro odeslání dat je zobrazen na obr. 6.1.



Obr. 6.1: Vývojový diagram - odeslání dat z FPGA karty

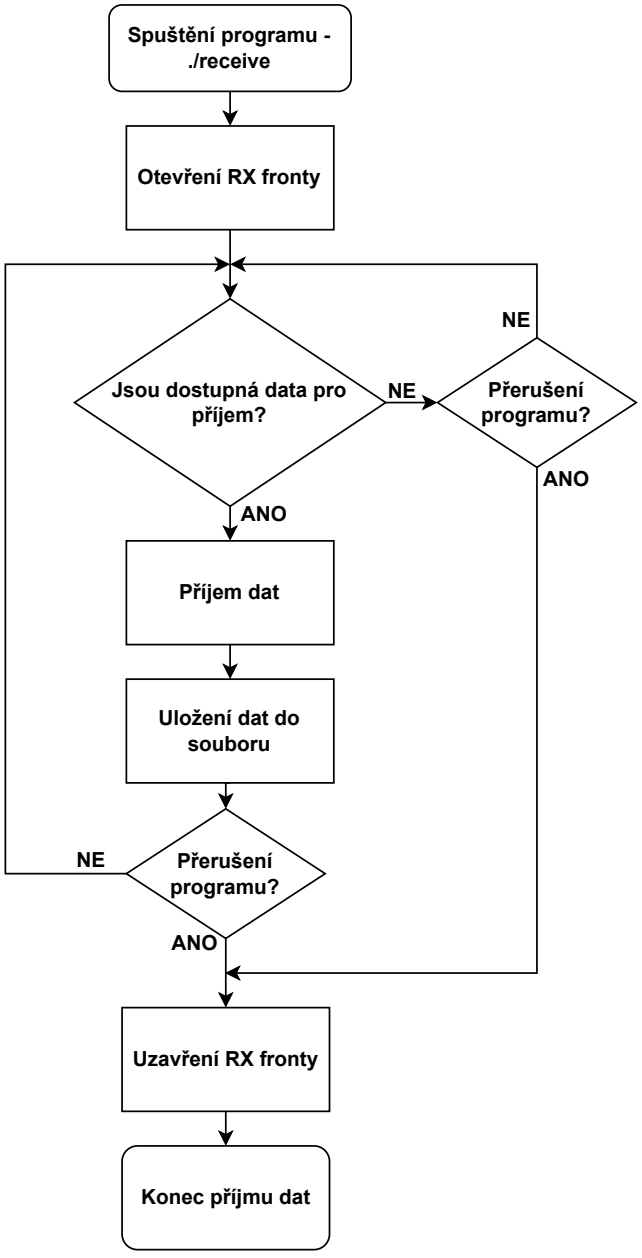
Spuštění aplikace pro přenos se provede příkazem `./transmit nazev_souboru cislo_dma_kanalu cislo_sitove_karty`. Uživatel při spouštění aplikace musí zadat tři argumenty. První argument specifikuje název textového souboru, ze kterého se budou data zasílat. Druhým argumentem uživatel určí číslo DMA kanálu pro přenos dat, kdy lze volit mezi kanálem číslo 0 a kanálem číslo 1. Třetím argumentem je číslo síťové karty, na kterou se data budou odesílat. Jelikož jsou dostupné dvě síťové karty, tak číslo síťové karty lze volit mezi hodnotami 0 a 1. Při úspěšném přenesení dat je pro kontrolu vypsána délka přenesených dat. Úspěšné přenesení dat mezi operačním systémem a FPGA kartou je zobrazeno ve výpisu 6.3.

Výpis 6.3: Výpis terminálu - odeslání dat

```
server@fpga2:~$ ./transmit input.txt 0 0
Length of transmitted data is 282 B
server@fpga2:~$
```

6.2.2 Aplikace pro příjem dat z FPGA karty

Pro příjem dat ze síťové karty slouží aplikace `receive.c`. Stejně jako aplikace pro odesílání dat, tak i tato aplikace využívá funkce z hlavičkových souborů `<nfb/ndp.h>` a `<nfb/nfb.h>`. Vývojový diagram aplikace pro příjem dat je zobrazen na obr. 6.2.



Obr. 6.2: Vývojový diagram - příjem dat z FPGA karty

Spuštění aplikace pro příjem dat se provede obdobně jako u aplikace pro odesílání dat, tj. příkazem `./receive nazev_souboru cislo_kanalu cislo_sitove_karty`. Opět jsou zde tři argumenty, které musí uživatel při spouštění zadat. Prvním argumentem je název souboru, do kterého se budou přijatá data v hexadecimální formě

zapisovat. Druhým argumentem je číslo DMA kanálu, přes který budou data z karty přicházet. Třetím argumentem je číslo síťové karty, na které bude příjem dat uskutečněn. Jakmile je aplikace spuštěna, přijímá data z daného kanálu, dokud aplikaci uživatel neukončí pomocí klávesové zkratky `CTRL + C`. Po ukončení aplikace je vypsán celkový objem přenesených dat a počet přijatých datových rámců. Úspěšný příjem dat je zobrazen ve výpisu 6.4.

Výpis 6.4: Výpis terminálu - příjem dat

```
server@fpga2:~$ ./receive output.txt 0 0
^C
Total length of received data is 282 B
Received frames: 1
server@fpga2:~$
```

Jestliže uživatel potřebuje překompilovat zdrojový kód dané aplikace, musí použít přepínač `-lnfb`, který umožní zkompilovat kód s `libnfb` knihovnou. Příkaz pro překompilování aplikace pro příjem by byl `gcc receive.c -lnfb -o receive`.

6.3 Komunikace přes síťové rozhraní

Síťová karta fb4CGg3 disponuje čtyřmi síťovými rozhraními, avšak kvůli skutečnosti, že NDK firmware podporuje pro daný druh karty pouze dvě rozhraní, je možné pracovat pouze ze síťovým rozhraním 0 a 1. Výsledný demonstrátor je schopný přenášet data mezi operačním systémem a FPGA kartou a rovněž mezi jejími síťovými rozhraními. Komponenta (NDK_app_core_TX_side.vhd) běžící na vysílací (TX) straně má za úkol zpracovat přijatá data, připojit Ethernetovou hlavičku a zaslat data na síťové rozhraní. Na přijímací (RX) straně pracuje komponenta (NDK_app_core_RX_side.vhd), která má za úkol odstranit Ethernetovou hlavičku a poslat data do DMA modulu, ze kterého jsou data přijímána do operačního systému.

6.3.1 Odesílání dat přes síťové rozhraní

Příchozí data z DMA modulu jsou ukládána do vstupní (RX) FIFO (First In, First Out) fronty. Data přicházejí po FLU sběrnici, kde je přenos dat řízen na základě signálů popsaných v tab 5.2. Příjem dat je možný pouze za předpokladu, kdy signály `SRC_RDY` a `DST_RDY` jsou aktivní, tj. v logické '1'. Tímto je signalizováno, že vysílací i přijímací komponenty jsou připraveny k vysílaní popř. příjmu dat. FLU sběrnice vykazuje při komunikaci specifickou vlastnost, kdy začátek prvního bloku dat nemusí začínat na pozici `DATA (7 downto 0)`, ale např. na pozici `DATA(71 downto 64)`.

Tuto skutečnost je potřeba na základě řídících signálů ošetřit a data naskládat do FIFO fronty tak, aby vždy začínala na dané pozici FIFO fronty na nulté pozici. Tímto je usnadněna následná práce s daty.

Řízení zápisu či čtení dat ze vstupní i výstupní FIFO fronty je řízeno pomocnými registry. Ve výpisu 6.5 jsou vypsány všechny registry, které řídí zápis či čtení dat ze vstupní FIFO fronty.

Výpis 6.5: Registry pro řízení zápisu a čtení dat ze vstupní FIFO fronty

```
1 for i in 0 to 8 - 1 loop
2     if RX_data_FIFO_valid_reg_set(i) = '1' then
3         RX_data_FIFO_valid_reg(i) <= '1';
4     end if;
5
6     if RX_data_FIFO_valid_reg_clear(i) = '1' then
7         RX_data_FIFO_valid_reg(i) <= '0';
8     end if;
9 end loop;
```

Pokud jsou na aktuální pozici FIFO fronty zapsána data, je na základě pomocného registru `RX_data_FIFO_valid_reg_set` uložena do řídícího registru `RX_data_FIFO_valid_reg` hodnota '1', která indikuje, že na dané pozici FIFO fronty jsou dostupná data. Jakmile se data z této pozice vyčtou, je pomocí pomocného registru `RX_data_FIFO_valid_reg_clear` uložena do řídícího registru `RX_data_FIFO_valid_reg` hodnota '0', která indikuje, že aktuální pozice FIFO fronty je volná pro další zápis dat. Stejným principem je řízen zápis a čtení dat i z výstupní (TX) FIFO fronty.

S daty uloženými ve vstupní FIFO frontě je možné provést jakoukoliv požadovanou operaci. V tomto případě je k datům přidávána Ethernetová hlavička. Síťová karta ve výchozím módu (Mode 0) pracuje v promiskuitním režimu. To znamená, že přijímá všechna data bez ohledu na MAC adresu. Je zde však i mód (Mode 1), který umožňuje, na základě tabulky povolených MAC adres, příchozí data filtrovat. Blok s Ethernetovou hlavičkou připojován k datům je ve tvaru cílová MAC adresa (6 B) + zdrojová MAC adresa (6 B) + EtherType (2 B) + 50 B nul (pro zarovnání FLU signálu `DATA` na 64 B). Pole CRC (Cyclic Redundancy Check) není potřeba k datům přidávat, jelikož je vypočítáno a připojeno v síťovém modulu. Ve výpisu 6.6 je znázorněna část kódu, která připojuje Ethernetovou hlavičku.

Výpis 6.6: Přidání Ethernetové hlavičky

```
1 if ETH_header_w = '0' then
2  if TX_data_FIFO_valid_reg(to_integer(TX_data_FIFO_write_addr + 1)) =
       '0' then
3     RX_data_FIFO_valid_reg_clear(to_integer(RX_data_FIFO_read_addr)) <=
          '1';
```

```
4       TX_data_FIFO_valid_reg_set(to_integer(TX_data_FIFO_write_addr)) <=
            '1';
5       TX_data_FIFO_valid_reg_set(to_integer(TX_data_FIFO_write_addr + 1))
            <= '1';
6       TX_data_FIFO(to_integer(TX_data_FIFO_write_addr)) <= ETH_header;
7       TX_data_FIFO(to_integer(TX_data_FIFO_write_addr + 1)) <=
            RX_data_FIFO(to_integer(RX_data_FIFO_read_addr));
8       ETH_header_w <= '1';
9       TX_data_FIFO_SOP(to_integer(TX_data_FIFO_write_addr)) <= '1';
10
11      TX_data_FIFO_EOP(to_integer(TX_data_FIFO_write_addr)) <= '0' ;
12      TX_data_FIFO_EOP(to_integer(TX_data_FIFO_write_addr + 1)) <= '0' ;
13      TX_data_FIFO_SOP(to_integer(TX_data_FIFO_write_addr + 1)) <= '0' ;
14
15      RX_data_FIFO_read_addr <= RX_data_FIFO_read_addr + 1;
16      TX_data_FIFO_write_addr <= TX_data_FIFO_write_addr + 2;
17
18      if RX_data_FIFO_EOP(to_integer(RX_data_FIFO_read_addr)) = '1' then
19          TX_data_FIFO_EOP(to_integer(TX_data_FIFO_write_addr + 1)) <= '1
                ' ;
20          TX_data_FIFO_EOP_POS(to_integer(TX_data_FIFO_write_addr + 1))
                <= RX_data_FIFO_valid_bytes(to_integer(
                RX_data_FIFO_read_addr)) ;
21          ETH_header_w <= '0';
22          end if;
23      end if;
```

Přidávání hlavičky je řízeno pomocným signálem `ETH_header_w`, který určuje, zda hlavička byla zapsána, či nikoliv. Pokud je signál `ETH_header_w` nastaven na hodnotu '0', je možné zapsat Ethernetovou hlavičku. Pomocí řídícího registru `TX_data_FIFO_valid_reg` se na dané zapisovací adrese zkontroluje, zda je ve výstupní (TX) FIFO frontě místo. Pokud ano, jsou do řídících registrů nastaveny odpovídající hodnoty o uvolnění popř. o obsazenosti míst ve vstupní a výstupní FIFO frontě. Následně je na aktuální volnou pozici výstupní fronty vložen blok s Ethernetovou hlavičkou a na další volnou pozici je uložen první blok dat ze vstupní `RX_data_FIFO` fronty. Po vložení hlavičky je nastaven pomocný signál `ETH_header_w` do hodnoty '1'. V této hodnotě setrvá do doby, dokud se do výstupní fronty neuloží blok dat, ve kterém přenášená data končí, tj. signál EOP je v '1'.

Data s připojenou hlavičkou jsou uložena ve výstupní frontě `TX_data_FIFO`, ze které jsou následně skrze FLU sběrnici poslána do síťového modulu a poté na síťové rozhraní. Souběžně s odesílanými daty je síťovému modulu zaslána i 128 bitová NDP hlavička, která v sobě nese číslo využitého síťového rozhraní. Výpis 6.7 znázorňuje nastavení výstupních signálů a hlavičky.

Výpis 6.7: Nastavení výstupních signálů a hlavičky

```
1
2  if (TX_SRC_RDY_sig = '0' or TX_DST_RDY = '1') then
3    if TX_data_FIFO_valid_reg(to_integer(TX_data_FIFO_read_addr)) = '1'
         then
4      TX_DATA_sig <= TX_data_FIFO(to_integer(TX_data_FIFO_read_addr));
5      TX_HEAD_DATA_sig <= (others => '0');
6      TX_SRC_RDY_sig <= '1';
7      TX_SOP_POS_sig <= (others => '0');
8      TX_EOP_POS_sig <= TX_data_FIFO_EOP_POS(to_integer(
           TX_data_FIFO_read_addr));
9      TX_SOP_sig <= TX_data_FIFO_SOP(to_integer(TX_data_FIFO_read_addr));
10     TX_EOP_sig <= TX_data_FIFO_EOP(to_integer(TX_data_FIFO_read_addr));
11     TX_data_FIFO_valid_reg_clear(to_integer(TX_data_FIFO_read_addr)) <=
           '1';
12     TX_data_FIFO_read_addr <= TX_data_FIFO_read_addr + 1;
13   end if;
14 end if;
```

Pokud jsou ve výstupní frontě data připravená k odeslání, jsou z fronty zaslána na výstup, tj. do `TX_DATA_SIG`. Hlavička se nastaví dle požadavků na použité síťové rozhraní. Signály FLU sběrnice jsou nastaveny podle délky přenášených dat. Po odeslání je pomocný registr pro výstupní frontu na dané čtecí adrese vynulován. Čtecí adresa je na závěr inkrementována o jedna tak, aby bylo následně možné číst z další pozice výstupní FIFO fronty. Data jsou odeslána do síťového modulu, kde je k datům přidáno CRC. Ze síťového modulu jsou data následně zaslána na dané síťové rozhraní.

6.3.2 Příjem dat přes síťové rozhraní

K datům přijatým přes síťové rozhraní je v síťovém modulu vygenerována odpovídající NDP hlavička ve tvaru, který je popisován v kapitole 5.1.1. Data a NDP hlavička jsou následně odeslána do komponenty pro příjem. Tato komponenta má obdobnou funkcionalitu jako komponenta pro odesílání dat. Přijatá data uloží do vstupní FIFO fronty, odstraní Ethernetovou hlavičku a zašle je na výstup. Odstranění hlavičky je založeno na podobném principu jako přidání hlavičky popisované v předchozí kapitole. V přijaté NDP hlavičce je upravena část `Segment size`, od které je odečtena velikost odstraněné Ethernetové hlavičky. Data a NDP hlavička jsou následně z komponenty pro příjem zaslána do komponenty `HINS_PLUS`, která slouží pro připojení NDP hlavičky k odpovídajícím datům. Komponenta `HINS_PLUS` využívá pro řízení přenosu hlaviček signály `HEAD_NEXT` a `HEAD_READY`. Signál `HEAD_NEXT` slouží jako ukazatel, že komponenta `HINS_PLUS` je připravena přijmout další hlavičku. Signál

HEAD_READY indikuje, že hlavička je dostupná a připravena k odeslání. Výsledná data s připojenou NDP hlavičkou jsou zaslána do DMA modulu, ze kterého již lze, pomocí aplikace pro příjem, vyčíst a zobrazit přijatá data.

6.3.3 Připojení komponent do aplikačního jádra

Po odsimulování funkčnosti je možné komponenty připojit k aplikačnímu jádru NDK. Aplikační jádro se nachází v souboru `application.vhd`. V prvním kroku je potřeba přidat komponenty pro odesílání a příjem do NDK jako modul. Toho lze docílit vložením příkazu `set MOD "$MOD $ENTITY_BASE/"nazev_komponenty.vhd"` do souboru `Modules.tcl`. Následně již lze do aplikačního jádra připojit samotné komponenty. Vstupní rozhraní (`RX_*`) u komponenty pro odesílání dat je připojeno k FLU rozhraní `DMA_TX_PIPE_*`, ze kterého budou přicházet uživatelská data. K výstupnímu (`TX_*`) rozhraní komponenty pro odesílání dat jsou přiřazeny pomocné signály `add_header`, které se připojí na vstupní rozhraní komponenty `eth_tx_slr_i`, která data zašle do síťového modulu. Ve výpisu 6.8 je znázorněno připojení komponenty pro odesílání dat do aplikačního jádra.

Výpis 6.8: Připojení komponenty pro odesílání dat do aplikačního jádra NDK

```
1  add_header : entity work.NDK_app_core_TX_side
2      port map(
3          RX_DATA => dma_tx_pipe(0).data,
4          RX_SOP_POS => dma_tx_pipe(0).sop_pos,
5          RX_EOP_POS => dma_tx_pipe(0).eop_pos,
6          RX_SOP => dma_tx_pipe(0).sop,
7          RX_EOP => dma_tx_pipe(0).eop,
8          RX_SRC_RDY => dma_tx_pipe(0).src_rdy,
9          RX_DST_RDY => dma_tx_pipe(0).dst_rdy,
10         TX_DATA => add_header.data,
11         TX_SOP_POS => add_header.sop_pos,
12         TX_EOP_POS => add_header.eop_pos,
13         TX_SOP => add_header.sop,
14         TX_EOP => add_header.eop,
15         TX_SRC_RDY => add_header.src_rdy,
16         TX_DST_RDY => add_header.dst_rdy,
17         TX_HEAD_DATA => add_header.head,
18         CLK => dma_clk_int,
19         SRST => dma_reset_int(0)
20     );
```

U komponenty pro příjem dat je vstupní rozhraní (`RX_*`) připojeno ke komponentám `eth_rx_data_slr_i` a `eth_rx_head_slr_i`, ze kterých přicházejí do aplikačního jádra data a NDP hlavička. K výstupnímu (`TX_*`) rozhraní komponenty pro příjem dat jsou přiřazeny pomocné signály `remove_header`, které se následně

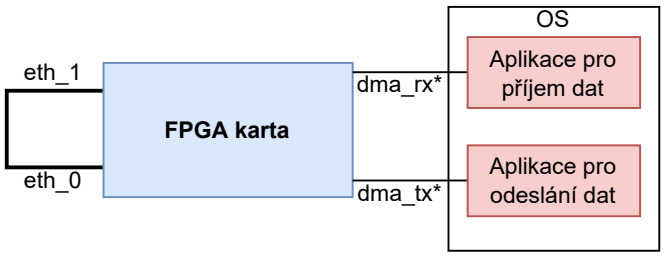
připojí na vstupní rozhraní komponenty `hins_i`, která k datům přidá NDP hlavičku a zašle je na výstupní FLU rozhraní `DMA_RX_*`.

6.4 Otestování implementace

Po připojení všech komponent do aplikačního jádra je možné implementaci otestovat. Otestování proběhlo jak v rámci jedné síťové karty, tak i mezi dvěma síťovými kartami.

6.4.1 Otestování implementace v rámci jedné karty

Testování v rámci jedné síťové karty probíhalo na základě propojení dvou síťových rozhraní, kdy byl vytvořen tzv. loopback. Výsledné zapojení pro otestování v rámci jedné karty je znázorněno na obr. 6.3.



Obr. 6.3: Zapojení pro otestování na jedné kartě

K síťové kartě je připojena sonda, která umožňuje pomocí ILA (Integrated Logic Analyzer) bloků zobrazit chování jednotlivých signálů přímo na síťové kartě, a tímto otestovat funkcionalitu nebo případně odladit chyby a nedostatky.

ILA blok je dostupný ve Vivadu jakožto IP Core. Do projektu je ILA blok přidán skrze nabídku `IP Catalog`, kde pod položkou Debug & Verification se nachází záložka Debug a v ní je dostupný ILA blok. V konfiguraci ILA bloku je nastaven název ILA bloku, počet sledovaných signálů a jejich bitová šířka. Zbylé parametry jsou ponechány ve výchozím stavu.

Stejně jak u předchozích komponent, tak i ILA blok je potřeba přidat do souboru `Modules.tcl`. Konkrétně je zde potřeba přidat soubor s příponou .xci. Příkaz pro přidání tohoto souboru je stejný jako v kapitole 6.3.4. Po přidání ILA bloku do souboru `Modules.tcl` je důležité jej připojit i do aplikačního jádra. Na vstup ILA bloku jsou připojeny odpovídající signály, které mají být zobrazeny. K aplikačnímu jádru byly připojeny 4 ILA bloky, které slouží pro zobrazení vstupních a výstupních signálů komponenty pro odesílání dat a komponenty pro příjem dat. Připojení

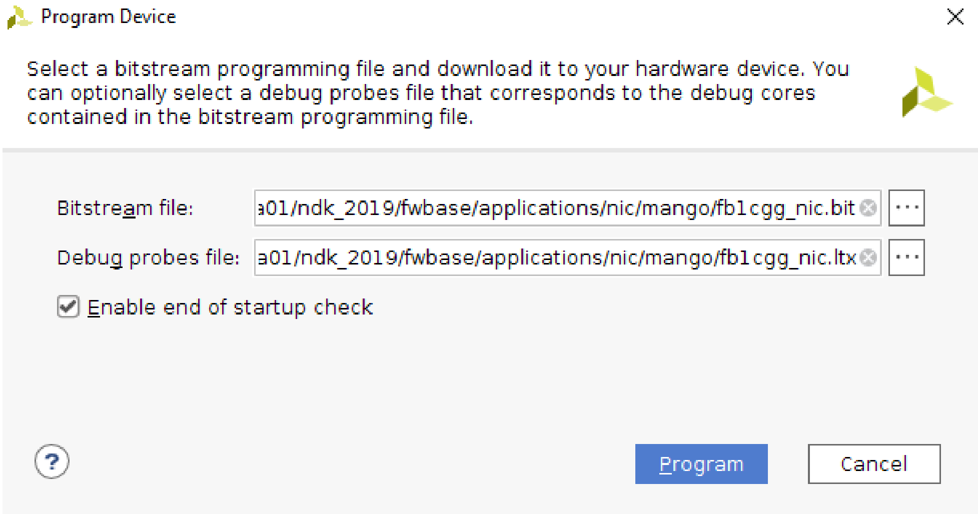
ILA bloku do aplikačního jádra pro zobrazení výstupního rozhraní komponenty pro odesílání dat je uvedeno ve výpisu 6.9.

Výpis 6.9: Připojení ILA bloku do aplikačního jádra

```
debug_ila_TX_side_TX: entity work.ila_TX_side_TX
    port map(
        probe0 => add_tx_header(0).data,
        probe1(0) => add_tx_header(0).sop,
        probe2(0) => add_tx_header(0).eop,
        probe3 => add_tx_header(0).sop_pos,
        probe4 => add_tx_header(0).eop_pos,
        probe5(0) => add_tx_header(0).src_rdy,
        probe6(0) => add_tx_header(0).dst_rdy,
        probe7 => add_tx_header(0).head,
        clk => dma_clk_int
        );
```

Po připojení komponent a ILA bloků k aplikačnímu jádru je možné přejít ke generování bitstreamu. Vygenerování bitstreamu je provedeno spuštěním příkazu `make` ve složce `fwbase/applications/nic/mango`. Příkazem `make` je rovněž provedena syntéza a implementace návrhu. Přibližně po třech a půl hodinách je vygenerován bitstream `fb1ccg_nic.bit`.

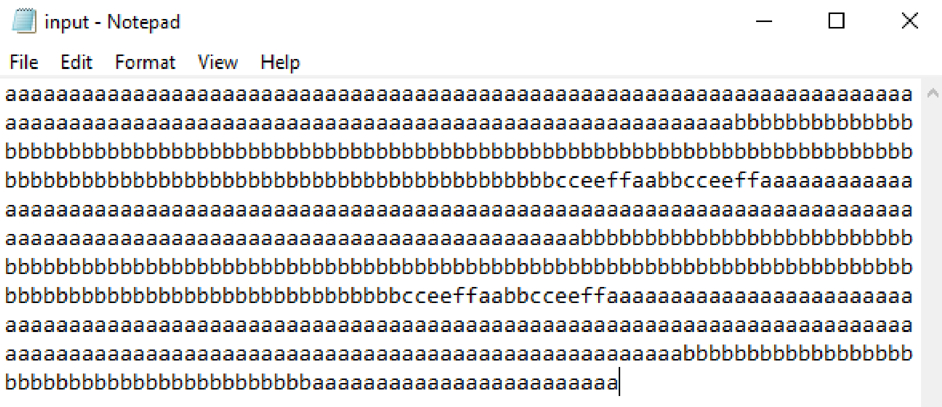
Po úspěšném vygenerování bitstreamu je možné přejít k otestování. V prvním kroku je potřeba vygenerovat soubor s příponou .ltx, který obsahuje informace důležité pro správné ladění. Tento soubor je vygenerován ve Vivadu v části Implemetation - Open Implemented Design. Zde stačí zadat do Tcl konzole příkaz `writ_debug_probes fb1cgg_nic.ltx`. Následně je důležité spustit na FPGA serveru Hardware server, na který se bude z Vivado serveru přistupovat. Hardware server se na FPGA serveru spustí ve složce `/tools/Xilinx/HWSRVR/2019.1/bin` příkazem ./hw_server. Po spuštění Hardware serveru je možné se na něj skrze Vivado připojit. Toho lze docílit otevřením položky Open Hardware Manager - Open Target - Open New Target, kde je po zadání odpovídající IP adresy a portu Hardware serveru vytvořeno spojení. Dalším krokem je, pomocí položky Program Device, nahrání patřičného bitstreamu a .ltx souboru na samotnou síťovou kartu. Nahrání bitstreamu a .ltx souboru skrze Vivado je znázorněno na obr. 6.4.



Obr. 6.4: Nahrání bitstreamu skrze Vivado

Po nahrání bitstreamu je nezbytné provést restart FPGA serveru. Posledním krokem, po restartu, je přidání spouštěče (Trigger), na který bude ILA blok reagovat. Spouštěč může reagovat na sestupnou, náběžnou či libovolnou hranu hodinového signálu. K tomuto účelu poslouží libovolná hrana signálu SOP.

Po provedení předchozích kroků je možné otestovat, zda implementace funguje správně. Před samotným odesíláním a příjmem dat je důležité povolit síťová rozhraní. Toho lze docílit příkazem `nfb-eth -e 1`. Pomocí aplikace pro odeslání dat je na kartu možné zaslat požadovaná data. Pro otestování byla na kartu odeslána data o velikosti 379 B ze souboru `input.txt`, jehož obsah je zobrazen na obr. 6.5.



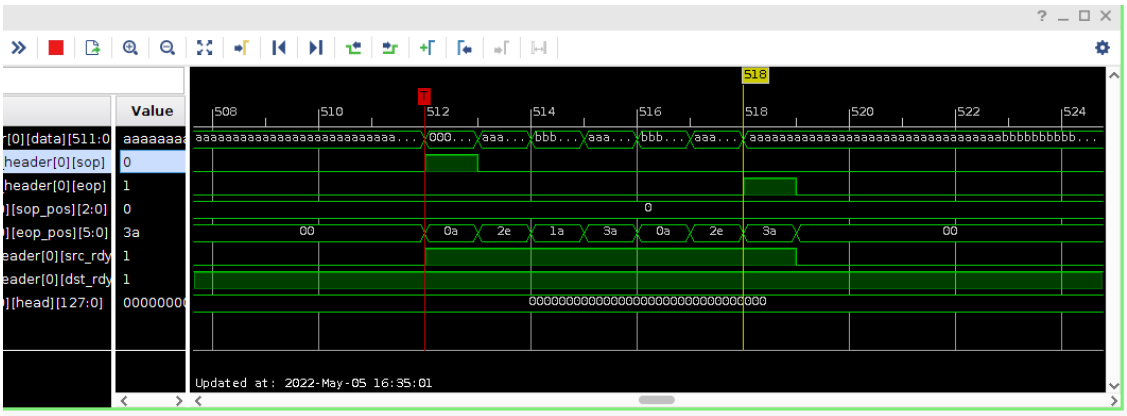
Obr. 6.5: Podoba odesílaných dat na síťovou kartu

Ve výpisu 6.10 je znázorněno odeslání požadovaných dat na síťovou kartu skrze odesílací aplikaci.

Výpis 6.10: Odeslání dat na síťovou kartu

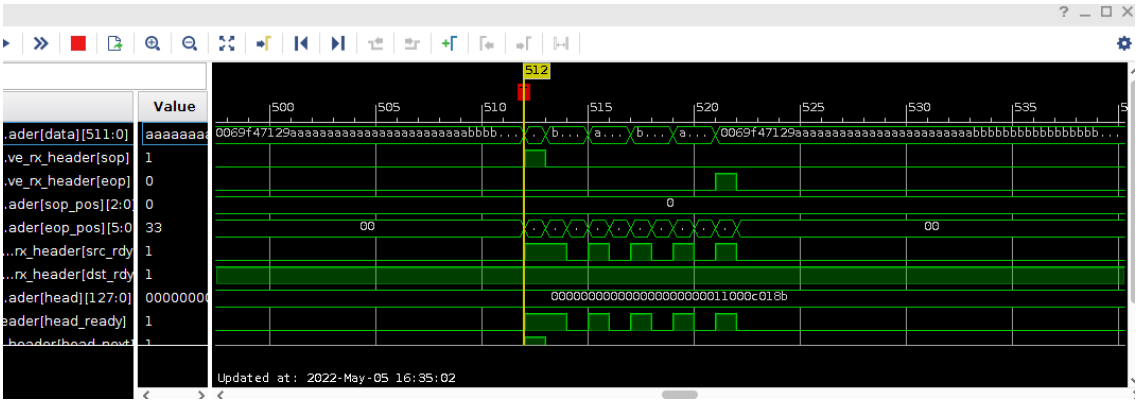
```
server@fpga2:~$ ./transmit input.txt 0 0
Length of transmitted data is 379 B
```

Na obr. 6.6 je pomocí ILA bloku zobrazeno výstupní rozhraní komponenty pro odeslání. Jsou zde vidět 512 bitové bloky dat FLU sběrnice a odpovídající řídící signály. Tímto je ověřeno, že komponenta pro odesílání dat funguje správně a že data z DMA modulu jsou zaslána na síťové rozhraní ve správné podobě.



Obr. 6.6: Podoba výstupního rozhraní komponenty pro odeslání dat

Na obr. 6.7 je pomocí ILA bloku zobrazeno výstupní rozhraní komponenty pro příjem. Rovněž jsou zde vidět 512 bitové bloky dat FLU sběrnice, NDP hlavička a odpovídající řídící signály. Tímto je ověřeno, že komponenta pro příjem dat funguje správně a že data a NDP hlavička přijatá ze síťového rozhraní jsou do DMA modulu posílána ve správném tvaru.



Obr. 6.7: Podoba výstupního rozhraní komponenty pro příjem dat

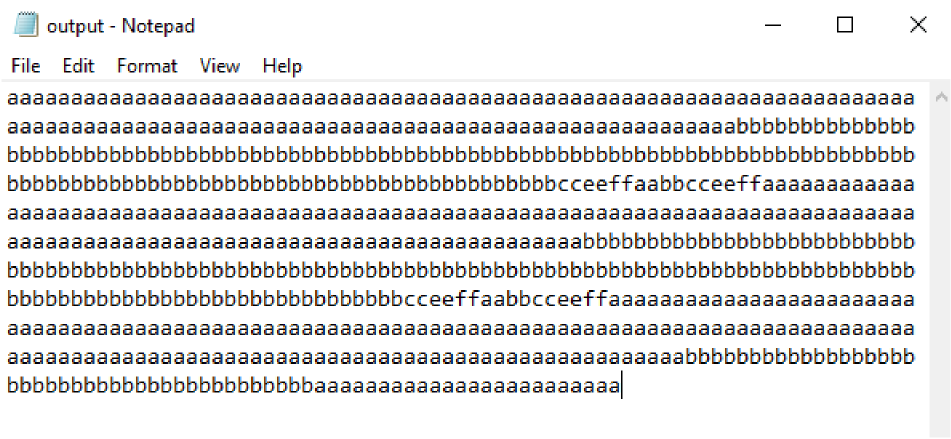
Příjem dat v operačním systému je proveden pomocí aplikace pro příjem. Přijatá data jsou v hexadecimálním tvaru uložena do souboru `output.txt`. Příjem dat

pomocí aplikace pro příjem je znázorněn ve výpisu 6.11.

Výpis 6.11: Příjem dat ze síťové karty

```
server@fpga2:~$ ./receive output.txt 0 0
^C
Total length of received data is 379 B
Received frames: 1
```

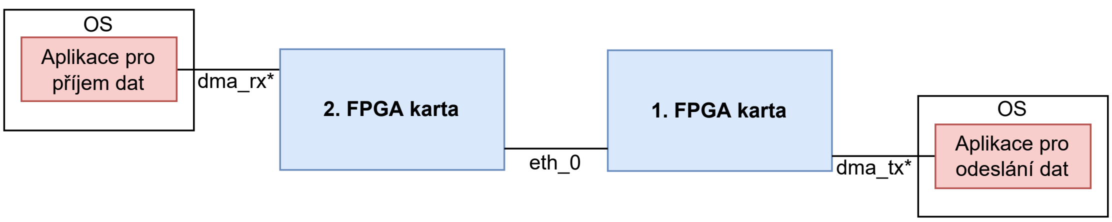
Na obr. 6.8 je vyobrazena podoba textového souboru `output.txt`, který obsahuje data přijatá ze síťového rozhraní. Výsledná data se shodují s daty ze souboru `input.txt`, která byla na kartu odeslána.



Obr. 6.8: Podoba přijatých dat ze síťové karty

6.4.2 Otestování implementace mezi dvěma kartami

Po otestování a odladění implementace v rámci jedné karty je možné provést otestování komunikace i mezi dvěma kartami. Zapojení pro otestování mezi dvěma kartami je zobrazeno na obr. 6.9.

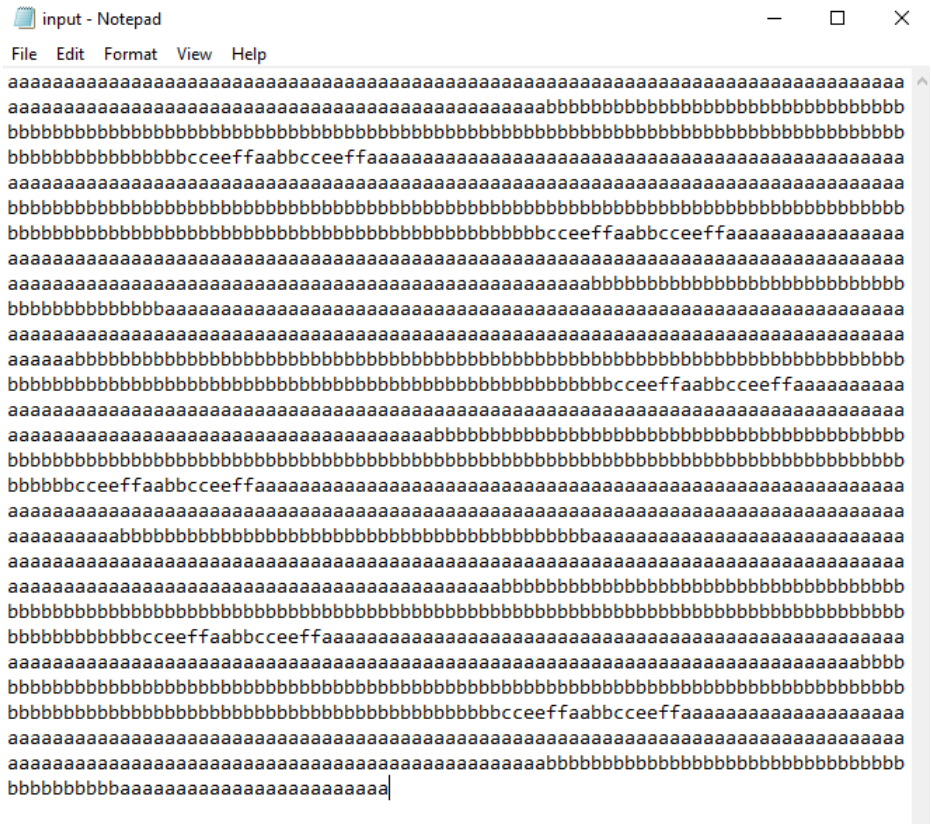


Obr. 6.9: Zapojení pro otestování na dvou kartách

Jelikož implementace již byla odladěna v rámci první karty, tak vytvořený bitstream již není potřeba zavádět na kartu skrze Vivado a zobrazovat průchozí komunikaci pomocí ILA bloků. Do paměti síťové karty lze bitstream vložit pomocí

NDK nástroje `nfb-boot`. Pro uložení jsou vyhrazeny dva sloty, kdy první slouží pro uživatelskou implementaci a druhý pro zálohu. Nahrání bitstreamu je možné provést příkazem `nfb-boot -f 0 fb1ccg_nic.bit`, kde parametr `-f` uloží bitstream do prvního slotu (slot 0) a zároveň provede jeho zavedení.

Po úspěšném zavedení bitstreamu je možné implementaci otestovat. Pomocí příkazu `nfb-eth -e 1` se aktivují síťová rozhraní a aplikací pro odesílání dat se na první kartu zašlou požadována data. Na obr. 6.9 jsou zobrazena data o velikosti 1137 B zasílaná z textového souboru `input.txt`.



Obr. 6.10: Podoba odesílaných dat na síťovou kartu

Ve výpisu 6.12 je znázorněn výpis aplikace pro odesílání.

Výpis 6.12: Odeslání dat na síťovou kartu

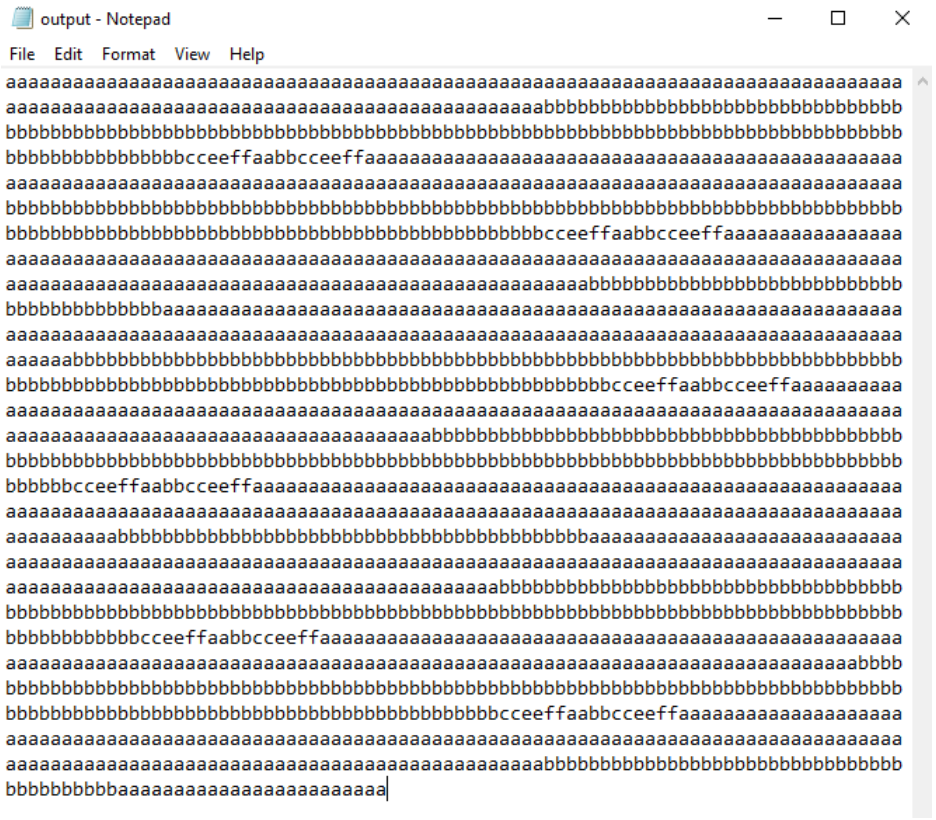
```
server@fpga2:~$ ./transmit input.txt 0 0
Length of transmitted data is 1137 B
```

Na druhé kartě je spuštěna aplikace pro příjem, u které je při spuštění potřeba upravit třetí argument (číslo síťové karty) z 0 na 1. Tímto je zajištěno, že aplikace bude přijímat data, která přišla na síťové rozhraní druhé karty. Přijatá data jsou aplikací uložena do textového souboru `output.txt`. Ve výpisu 6.13 je znázorněno přijetí dat na druhé kartu.

Výpis 6.13: Příjem dat ze síťové karty

```
server@fpga2:~$ ./receive output.txt 0 1
^C
Total length of received data is 1137 B
Received frames: 1
```

Z obr. 6.10 je patrné, že data přijatá na druhé kartě se shodují s daty odeslanými ze souboru `input.txt`.



Obr. 6.11: Podoba přijatých dat ze síťové karty

6.5 Využitelnost zdrojů a omezení

Síťová karta disponuje čipem Virtex UltraScale+, a to konkrétně typem xcvu9p-flgb2104-2-i. Pracovní frekvence je ve výchozím stavu nastavena na 200 MHz. Využitelnost zdrojů na tomto čipu u obou komponent a celé implementace je uvedeno v tab 6.1.

Tab. 6.1: Využitelnost zdrojů

Zdroj	Odesílání	Příjem	Celková impl.	Dostupné	Celkem [%]
LUT	9545	10035	92979	1182240	7,86
Flip Flop	9514	9627	140032	2364480	5,92

Komunikace přes síťové rozhraní je omezena velikostí MTU (Maximum Transmission Unit), které je ve výchozím stavu nastaveno na hodnotu 1500 B. MTU lze změnit nástrojem `nfb-eth`, kde pomocí parametru `-L` je možné nastavit velikost MTU až na 16 kB. Příkaz pro nastavení MTU, například na hodnotu 2000 B, by byl `nfb-eth -L 2000`.

6.6 Akcelerace kryptografických algoritmů

Síťové karty s FPGA čipem lze využít jako hardwarový akcelerátor pro kryptografické algoritmy. Demonstrátor je možné rozšířit o kryptografický algoritmus (například pro šifrování či podepisování dat), který by pracoval nad přenášenými daty. Akcelerovat lze všechny výpočetně náročné kryptografické algoritmy, u kterých je požadováno zrychlení výpočtů. Lze zmínit například šifrovací algoritmus AES-GCM (Advanced Encryption Standard - Galois Counter Mode) nebo postkvantové schéma CRYSTAL-Dilithium pro podepisování dat.

Závěr

Původní zaměření práce zahrnovalo implementaci šifrovacího algoritmu AES-GCM, avšak po dohodě s vedoucím práce byl cíl práce pozměněn na vytvoření demonstrátoru pro přenos dat mezi operačním systémem a síťovou kartou s FPGA čipem i mezi dvěma síťovými kartami. S tímto bylo spojeno nastudování FPGA problematiky, nastudování vývoje na FPGA zařízeních a seznámení se s vývojem na síťových kartách s FPGA čipem pomocí NDK od týmu Liberouter ze sdružení CESNET.

Pro účely přenosu dat mezi operačním systémem a síťovou kartou byly vytvořeny aplikace v jazyce C pro odesílání a přenos dat. Pro komunikaci přes síťové rozhraní byly vytvořeny komponenty pro odesílání a příjem dat v jazyce VHDL. Funkčnost komunikace byla otestována v rámci jedné karty na základě propojení dvou Ethernetových rozhraní a rovněž byla otestována i mezi dvěma kartami. Ladění vytvořené implementace na síťové kartě bylo provedeno pomocí ILA bloků, které pomohly odhalit a opravit řadu chyb. Výsledný demonstrátor je schopný přenést uživatelem specifikována data z textového souboru na síťovou kartu a rovněž je zaslat na druhou síťovou kartu přes Ethernetové rozhraní. Na druhé kartě je schopen data odchytit a uložit do textového souboru pro zobrazení. Demonstrátor lze případně rozšířit o kryptografický algoritmus, který by pracoval nad přenášenými daty.

Literatura

[1] *The Ultimate Guide to FPGA Design Flow* [online].[cit. 2021-11-29]. Dostupné z URL: <https://hardwarebee.com/the-ultimate-guide-to-fpga-architecture/>.

[2] *FPGA – Configurable Logic Block* [online]. 16-6-2021 [cit. 2021-11-29]. Dostupné z URL: <https://digilent.com/blog/fpga-configurable-logic-block/>.

[3] *Tutorial - How Flip-Flops Work in FPGAs* [online]. [cit. 2021-11-29]. Dostupné z URL: <https://www.nandland.com/articles/flip-flop-register-component-in-fp>.

[4] *CPLD vs FPGA: Differences between them and which one to use?* [online]. 29-11-2017 [cit. 2021-11-29]. Dostupné z URL: <https://numato.com/kb/cpld-vs-fpga-differences-one-use/>.

[5] *ASIC vs. FPGA: What's The Difference?* [online]. 10-10-2020 [cit. 2021-11-29]. Dostupné z URL: <https://www.asicnorth.com/blog/asic-vs-fpga-difference/>.

[6] *Hardware Description Languages: VHDL vs Verilog, and Their Functional Uses* [online]. [cit. 2021-11-29]. Dostupné z URL: <https://resources.pcb.cadence.com/blog/2020-hardware-description-languages-vhdl-vs-verilog-and-their-functional-uses>.

[7] KUBÍČEK, Michal. *Úvod do problematiky obvodů FPGA pro integrovanou výuku VUT a VŠB-TUO* [online]. Brno: Vysoké učení technické v Brně, 2014 [cit. 2021-11-29]. ISBN 978-80-214-5069-1.

[8] *Způsoby popisu logických obvodů v jazyce VHDL* [online]. [cit. 2021-11-29]. Dostupné z URL: <https://www.vovcr.cz/odz/tech/561/page03.html#heading4>.

[9] *How does using FPGAs impact the design process?* [online]. 17-2-2021 [cit. 2021-11-29]. Dostupné z URL: <https://www.microcontrollertips.com/how-does-using-fpgas-impact-the-design-process-faq/>.

[10] Xilinx. *Vivado Design Suite User Guide: Getting Started* [online]. 28-1-2021 [cit. 2021-11-29]. Dostupné z URL: <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug910-vivado-getting-started.pdf>.

[11] Xilinx. *Vivado Design Suite User Guide: High-Level Synthesis* [online]. 4-5-2021 [cit. 2021-11-29]. Dostupné z URL: <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug902-vivado-high-level-synthesis.pdf>.

[12] Xilinx. *Vivado Design Suite User Guide:Designing with IP* [online]. 18-12-2012 [cit. 2021-11-29]. Dostupné z URL: <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_4/ug896-vivado-ip.pdf>.

[13] Xilinx. *ISim User Guide* [online]. 24-4-2012 [cit. 2021-11-29]. Dostupné z URL: <https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx14_1/plugin_ism.pdf>.

[14] Silicom Denmark A/S. *FbBASIC for FPGA Card fb4CGg3* [online]. Denmark, 2017 [cit. 2021-11-29]. Dostupné z URL: <https://www.silicom.dk/wp-content/uploads/2019/08/fbBASIC-fb4CGg3.pdf>.

[15] Netcope Technologies, a.s. *NETCOPE DEVELOPMENT KIT:Firmware Developer's Manual* [online]. Czech Republic [cit. 2021-11-29].

Seznam symbolů a zkratek

AES-GCM Advanced Encryption Standard - Galois Counter Mode

ASIC Application Specific Integrated Circuit

CPLD Complex programmable logic device

CLB Configurable Logic Blocks

CLI Command Line Interface

CLU Central Processor Unit

DDR DoubleData Rate

DMA Direct Memory Access

GbE Gigabit Ethernet

HDL Hardware Description Language

FLU Frame Link Unaligned

FPGA Field Programmable Gate Array

ILA Integrated Logic Analyzer

ISE Integrated Synthesis Environment

IOB Input-Output block

IP Intellectual Property

I2C Inter-Integrated Circuit

LED Light-Emitting Diode)

LUT Lookup Tables

MI32 Memory interface 32-bit

NDK Network Development Kit

NDK Netcope Data Plane

NGD Native Generic Database

MTU Maximum Transmission Unit

PCAP	Packet Capturing
PCI	Peripheral Component Interconnect
RAM	Random Access Memory
RS-FEC	Reed Solomon Forward Error Correction
RTL	Register Transfer Level
SODIMM	Small Outline Dual In-line Memory Module
TCL	Tool Command Language
QDR	Quad Data Rate).
QSFP	QuadSmall Form-factor Pluggable
VHDL	Very High Speed Integrated Circuit HardwareDescription Language

A Obsah elektronické přílohy

- `/` ..... kořenový adresář přiloženého archivu
  - `applications` ..... zdrojové soubory aplikací pro komunikaci mezi OS a kartou
    - `receive.c`
    - `transmit.c`
  - `TX_side` ..... zdrojové kódy komponenty pro odesílání dat
    - `NDK_app_core_TX_side.vhd`
    - `TX_side_sim.vhd`
  - `RX_side` ..... zdrojové kódy komponenty pro příjem dat
    - `NDK_app_core_RX_side.vhd`
    - `RX_side_sim.vhd`
  - `ILAs` ..... adresář obsahuje využité ILA bloky
    - `ila_RX_side_RX.xci`
    - `ila_RX_side_TX.xci`
    - `ila_TX_side_RX.xci`
    - `ila_TX_side_TX.xci`