VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

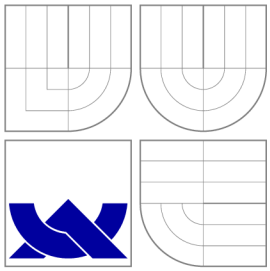
PŘEVOD PLATFORMY NETCOPE DO EDK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

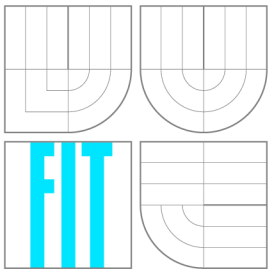
AUTOR PRÁCE
AUTHOR

JAN PALIČKA

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PŘEVOD PLATFORMY NETCOPE DO EDK

PORTING NETCOPE PLATFORM TO EDK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN PALIČKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN VIKTORIN
SUPERVISOR

BRNO 2014

Abstrakt

Tato bakalářská práce se zaobírá převodem platformy NetCOPE do Xilinx Embedded Development Kit (EDK). Hlavním úkolem je vytvoření anotace hardware platformy NetCOPE pro její použití v prostředí EDK. Před samotnou implementací anotace je nutné nastudovat technologii FPGA, platformu NetCOPE a formát PSF pro anotaci IP-core platformy NetCOPE.

Abstract

This bachelor's thesis deals with porting of NetCOPE to Xilinx Embedded Development Kit (EDK). Main task is to create annotation of NetCOPE hardware for using in EDK. Before implementation of annotation itself, it is necessary both to study FPGA technology, possibilities of FPGA progamming, NetCOPE platform and PSF for anotation of NetCOPE's IP-core.

Klíčová slova

převod, hardware, FPGA, IP-core, NetCOPE ,VHDL, EDK, XPS, PSF, COMBOv2, PAO, BBD, MPD, MHS

Keywords

porting, hardware, FPGA, IP-core, NetCOPE ,VHDL, EDK, XPS, PSF, COMBOv2, PAO, BBD, MPD, MHS

Citace

Jan Palička: Převod platformy NetCOPE do EDK, bakalářská práce, Brno, FIT VUT v Brně, 2014

Převod platformy NetCOPE do EDK

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Viktorina a za odborné spolupráce pánů Ing. Viktora Puše ze společnosti CESNET, z. s. p. o. a Ing. Pavla Korčeka

.....
Jan Palička
19. května 2014

Poděkování

Na tomto místě bych rád poděkoval zejména vedoucímu práce panu Ing. Janu Viktorinovi, dále pak konzultantovi panu Ing. Viktoru Pušovi ze společnosti CESNET, z. s. p. o., panu Ing. Pavlu Korčekovi za pomoc a odborné rady a v neposlední řadě také všem, kteří mě během vypracovávání práce podporovali.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Návrh číslicových obvodů je rychle se rozvíjejícím odvětvím informačních technologií. Množství dostupných vývojových platforem umožňuje snazší vytváření nových hardwarových projektů.

NetCOPE je konfigurovatelná platforma pro rychlý a snadný vývoj aplikací na čipech FPGA. Tato platforma vznikla v projektu Liberouter ve spolupráci se sdružením Cesnet z. s. p .o. NetCOPE vytváří mezivrstvu mezi FPGA a samotnou aplikací. Nezatěžuje hardwarového návrháře nutností znát přesná specifika hardware, na kterém chce NetCOPE provozovat.

Cílem této bakalářské práce je vytvoření snadnějšího způsobu pro výstavbu nových hardwarových projektů nad NetCOPE. anotací částí NetCOPE tak, aby ho bylo v budoucnu možné použít v nástroji Xilinx Embedded Development Kit. Tento nástroj umožňuje jednoduché použití požadovaných komponent. Návrhář hardware pouze vybere, které části chce ve výsledku použít a nástroj sám vytvoří všechny potřebné součásti pro překladový systém.

Tato bakalářská práce je rozdělena do 6 kapitol. Kapitola 2 se zaměřuje na technologii FPGA a na možnosti jejího programování. Programovací jazyky pro FPGA jsou v této kapitole rozděleny na HDL jazyky a vyšší programovací jazyky.

V kapitole 3 jsou popsány čipy FPGA firmy Xilinx a vývojovému prostředí Xilinx Embedded Development Kit. Kapitola obsahuje popis překladového systému firmy Xilinx. Dále popisuje, jakým způsobem je realizován popis hardware pomocí IP-core, následuje část věnující se Xilinx Embedded Development Kit. Zde je popsán formát anotace IP-core pro použití v EDK – Platform Specific Format. V závěrečné části této kapitoly je popsán způsob anotace hardware pomocí PSF.

Ve 4. kapitole je popsán samotný objekt převodu – platforma NetCOPE. V úvodu této kapitoly je popsána obecně platforma NetCOPE společně s hlavní hardwarovou platformou, na níž je NetCOPE používán. V následující části je podrobněji popsána platforma NetCOPE.

Stěžejní kapitolou je kapitola 5, kde je popsán návrh převodu platformy NetCOPE do EDK. Počínaje převodem NetCOPE jako celku, přes převod jednotlivých komponent, až po integraci mikroprocesoru MicroBlaze do výsledného designu karty. Cílem této bakalářské práce, který je popsán v druhé části této práce, je převod NetCOPE jako celku.

V závěrečné kapitole 6 na straně 46 je uvedeno shrnutí celé práce. Navíc jsou specifikovány možnosti dalšího budoucího vývoje.

Kapitola 2

Technologie FPGA

V následující části jsou popsány dva způsoby výroby číslicových obvodů. Konkrétně se jedná o typu čipů ASIC a FPGA. Jedinou vývojovou platformou použitou v rámci této práce jsou čipy FPGA. Čipy FPGA se v některých případech používají pro prototypování a až po detailním ověření funkčnosti číslicového obvodu se daný obvod vyrobí jako čip ASIC.

2.1 Hradlová pole FPGA

Programovatelná hradlová pole, neboli zkráceně FPGA (*z angl. Field Programable Gate Array*) je typ číslicových integrovaných logických obvodů. FPGA je vnitřně složeno z různě složitých logických bloků (*logické bloky, vstupně-výstupní bloky, LUT, BRAM, DSP, ...*), které jsou propojeny konfigurovatelnou maticí propojů.[9, s. 35]. Hlavní rozdíl oproti čipům ASIC je možnost rekonfigurace (změny funkce) čipu FPGA. Existují dva typy čipů FPGA dle uložení konfigurace, a to:

- *FPGA s volatilním uložením konfigurace*[9, s. 43] – používá pro její uložení paměti typu SRAM. Tyto paměti umožňují snadnější změnu konfigurace, ale kód je v paměti méně chráněn a kód není ochráněn před zásahem třetí strany. Navíc uložení konfigurace v paměti SRAM vyžaduje, aby byla při spuštění čipu tato konfigurace do paměti SRAM nejprve nahrána a až potom může dojít ke spuštění funkce čipu. Informace o konfiguraci, pokud mají přežít odpojení čipu od napájení, musí být uloženy v některém nevolatilním typu paměti, což vyžaduje dodatečné místo na desce plošných spojů. Samotné nahrání konfigurace do paměti SRAM také spotřebuje určitý čas, během něhož je čip neaktivní.
- *Nevolatilní uložení konfigurace*[9, s. 43] – používá pro uložení konfigurace paměti typu EEPROM. Před zápisem nově konfigurace je nutné tuto paměť vymazat. V současné době se od použití pamětí EEPROM upouští. Jsou nahrazovány pamětmi typu Flash.

2.2 Programování FPGA

K návrhu číslicových obvodů pro čipy FPGA se používá řada jazyků. Přehled některých z nich přináší následující část.

2.2.1 Jazyky HDL

Jazyky HDL (neboli *Hardware Description Language*) jsou jazyky pro popis struktury a funkce číslicových systémů. Jazyky HDL umožňují přesnou definici elektronického obvodu. Tato definice umožňuje simulaci a testování daného číslicového obvodu. Programy vytvořené v jazyce HDL se obvykle kopilují do nižších úrovní popisu hardwaru (*RTL, ...*).

Jazyky HDL na rozdíl od běžných aplikačních programovacích jazyků (jako je například jazyk C) umožňují zaznamenávat přesné chování algoritmu v čase. Mezi nejznámější zástupce jazyků HDL patří jazyky VHDL a Verilog.

```
entity test_bench is
end entity test_bench;

architecture test_reg4 of test_bench is
  signal en, clk;
begin
  stimulus : process is
  begin
    wait for 10 ns;
    en <= '1'; wait for 10 ns;
    clk = '1', '0' after 10 ns; wait for 20 ns;
  end process stimulus;
end architecture test_reg4;
```

Zdrojový kód 2.1: Ukázka časování v kódu VHDL

Jazyk VHDL

Jazyk VHDL[9, s. 47] je standardizován standardizačním institutem IEEE od roku 1987, revize jazyka proběhla v roce 1998. Samotné VHDL obsahuje prostředky pro popis paralelismu, konektivity a explicitního vyjádření času. Jazyk VHDL patří do skupiny typových jazyků.

Výhodou jazyka VHDL je jeho flexibilita. Původně jazyk VHDL vznikl jako jazyk pro dokumentaci chování obvodů v čipech typu ASIC [9, s. 47]. Později se vyvinula technika, která umožňovala spustit simulaci právě na základě dokumentace chování. Nakonec se část jazyka VHDL oddělila a stala se syntetizovatelnou.

Jazyk VHDL používá pro popis číslicových zařízení (hardwaru) dvojici [9, s. 54]:

- *entita*: Definuje rozhraní komponenty (vstupní a výstupní porty a piny).
- *architektura*: Určuje chování číslicového obvodu.

Pro jednu entitu může v jazyce VHDL existovat více architektur, může tedy existovat více implementací dané entity. Tato vlastnost však není vždy plně podporována syntézními nástroji.

Rozhraní hardwaru je v jazyce VHDL popsáno pomocí portů, ty jsou definovány názvem, propustným směrem a datovým typem. Nejpoužívanější propustné směry jsou [34, s. 2-5]:

- ***In**put port:* Port je určen pouze pro vyčítání dat.
- ***Out**put port:* Port je určen pouze pro příjem dat.

- ***InOut** port (dvousměrný port):* Port je určen pro příjem i vyčítání dat. Tento typ portu se používá výhradně na rozhraní čipu, jelikož vyžaduje třístavové budiče.

Základními datovými typy v jazyce VHDL jsou: Enumeration, Array, Integer, bit, a bit_vector [34, s. 2-11].

V následující ukázce zdrojového kódu je uvedena jednoduchá jednobitová násobička v jazyce VHDL[29, s. 88].

```
-- import std_logic z knihovny IEEE
library IEEE;
use IEEE.std_logic_1164.all;

-- definice entity
entity my_and is
  port (IN1, IN2 : in std_logic; OUT1: out std_logic);
end entity;

-- definice architektury
architecture example of my_and is
begin
  OUT1 <= IN1 and IN2;
end example;
```

Zdrojový kód 2.2: Jednoduchá násobička ve VHDL

Jazyk Verilog

Jazyk Verilog je programovací jazyk určený pro popis hardware, konkrétně pro popis analogových, digitálních či smíšených číslicových obvodů realizovaných v rámci programovatelných hradlových polí, jako jsou pole typu CPLD, FPGA. Jazyk Verilog je rovněž možné použít pro popis uživatelských obvodů postavených na technologii ASIC.

Jazyk Verilog [33, s. 1-1] byl standardizován institucí IEEE pod číslem 1394-1995, proto je možné se setkat s označením Verilog-95. Další verzí, která byla standardizována konzorciem IEEE byla verze Verilog 2001 (standard IEEE 1394-2001). Dosud poslední revizí jazyka Verilog byla verze Verilog-2005 (standardizována jako IEEE 1394-2005).

Na rozdíl od jazyka VHDL nerozděluje jazyk Verilog návrh hardware na architekturu a entitu. Naopak obě tyto části sdružuje do *modulů*. V rámci jednoho modulu je uvedena jak funkčnost, tak rozhraní [33, s. 2-1].

Rozhraní je popsáno pomocí portů a registrů, ty mohou mít různý datový směr (stejně jako v případě VHDL). Porty mohou být: vstupní (*Input port*), výstupní (*Output port*) a vstupně-výstupní (*InOut port*). Datový typ register slouží pro uložení hodnoty vzniklé přiřazením matematického nebo logického výpočtu.

Pro logické spojení dvou strukturních entit, jako jsou například hradla slouží datový typ *Net* [33, s. 3-1]. Do proměnné datového typu *Net* nemůže být zapsána hodnota. Datový typ *Net* proto musí být řízen elementy, které propojuje, například hradly.

Následuje příklad syntaxe jazyka Verilog, jde o jednobitovou binární násobičku[32].

```
‘timescale 1ns / 1ps

module AND2gate(A, B, F);

// definice vstupu a výstupu
   input A;
   input B;
   output F;

// definice použití registru
   reg F;

   always @ (A or B)
   begin
      F <= A & B;
   end

endmodule
```

Zdrojový kód 2.3: Jednoduchá násobička v jazyce Verilog

2.3 Vysokoúrovňové jazyky

Vysokoúrovňové jazyky sloužící pro popis hardware jsou většinou deriváty jazyka C nebo C++. Tyto jazyky (C, C++, ...) nebyly primárně vytvořeny pro popis hardware, proto bývají doplněny o konstrukce doplňující tuto chybějící funkcionalitu.

Velkou výhodou použití vysokoúrovňových jazyků při popisu a vytváření hardwaru je právě jejich podobnost s běžně používanými programovacími jazyky. Programátor tudíž nepotřebuje tak velký čas k přechodu na „jiný“ programovací jazyk. Ale i při použití vysokoúrovňových jazyků při popisu hardwaru platí, že programátor musí navrhovat algoritmy inteligentně a dodržovat principy obvyklé pro návrh hardware. Jinak hrozí, že syntetizační vytvoří nefunkční hardware, nebo se podaří vytvořit funkční hardware, který však spotřebovává příliš velké množství zdrojů. Nejpoužívanějšími vysokoúrovňovými jazyky jsou *Handle-C*, *SystemC* a *„čisté“ C*.

Handle-C

Jazyk Handle-C byl vytvořen na Oxfordské univerzitě v roce 1996 v rámci výzkumné skupiny zabývající se hardwarem. Na základě tohoto výzkumu vznikl *spin-of* nazvaný *Embedded Solutions Limited*. Společnost byla v roce 2000 přejmenována na *Celoxica*, ta se věnuje vývoji jazyka Handle-C dodnes.

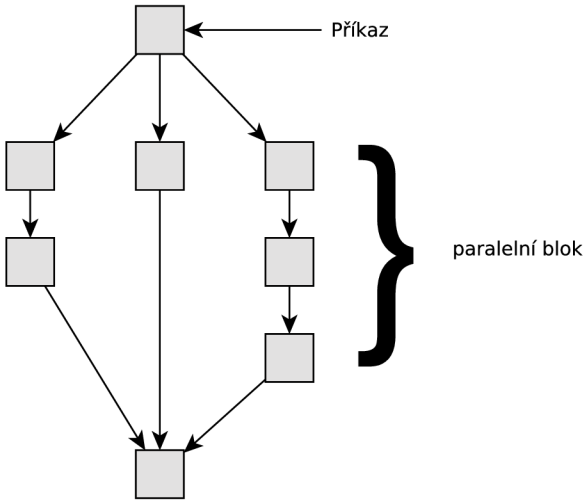
Jazyk Handle-C [13, s. 12] vychází z programovacího jazyka C, ke kterému přidává podporu pro:

- Paralelní programování.
- Komunikace pomocí kanálů (mezi paralelními procesy).

Běžný blok kódu je v jazyce Handle-C syntetizován jako sekvenční obvod. Pokud návrhář požaduje, aby byl blok kódu vykonán paralelně, uvede ho do operátoru *par*. Viz následující příklad [13, s. 13]:

```
par {
  ++c;
  a = d + e;
  b = d + e;
}
```

Při syntéze paralelního bloku je provádění rozděleno na několik částí, podle počtu paralelních příkazů (v předešlém příkladě by to byly tři paralelní větve) a každá tato větev by byla vykonána. Po dokončení všech paralelních větví se paralelní bloky opět spojí do jednoho. Paralelní bloky, které jsou vykonány dříve musí před spojením čekat na bloky pomalejší, viz obrázek 2.1.



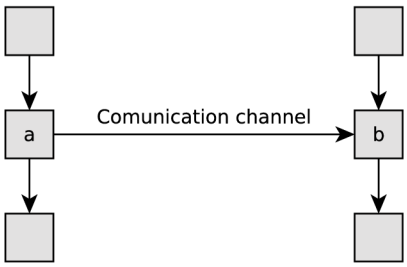
Obrázek 2.1: Paralelismus v Handle-C [13, s. 13]

Další vlastností, kterou přináší jazyk Handle-C je komunikace pomocí kanálů. Tato vlastnost je určena pro komunikaci mezi paralelně prováděnými procesy. Existují dvě varianty komunikace, a to komunikace se synchronizací a bez synchronizace [13, s. 13].

Komunikace pomocí kanálů se synchronizací [13, s. 13] využívá dvou principů, jak může být kanál mezi procesy vytvořen. První možností synchronní komunikace je konstrukce komunikačního kanálu jako FIFO fronty. Pokud je FIFO fronta plná, je procesu zablokována možnost zápisu do FIFO fronty do doby, než dojde k uvolnění dostatečného množství místa ve frontě. Naopak, pokud chce jeden z procesů z fronty data číst a fronta je momentálně prázdná, je operace čtení zablokována do doby, než budou ve frontě dostupná nějaká data.

Pokud je vyžadována synchronní komunikace, ale není využita FIFO fronta [13, s. 13], musí komunikační kanál zajistit synchronizaci mezi větvemi paralelně prováděného procesu. Pokud tedy chtějí procesy komunikovat, musí být obě strany připraveny.

Obrázek 2.2 ukazuje realizaci synchronní komunikace bez FIFO fronty. Pokud levá větev dosáhne místa `a`, ale pravá větev ještě nedosáhla místa `b`, musí levá větev čekat před místem `a`.

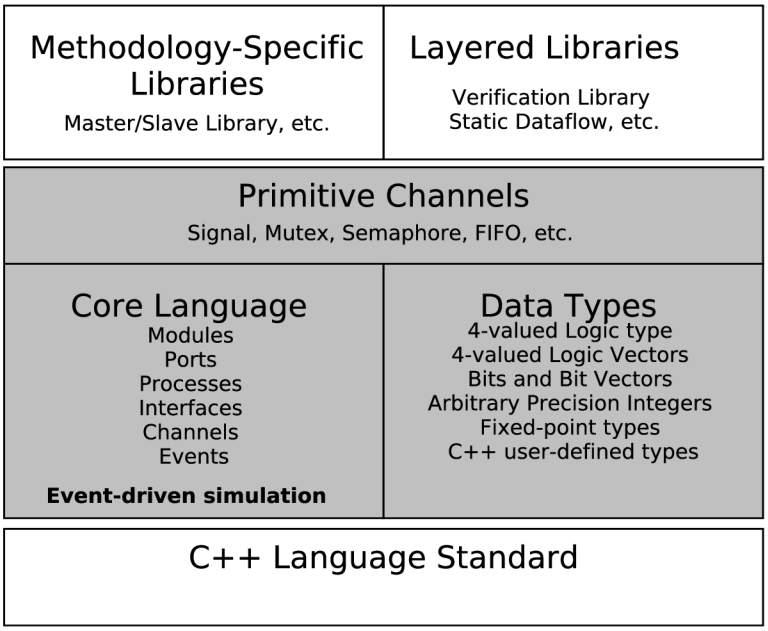


Obrázek 2.2: Synchronní komunikace bez FIFO fronty [13, s. 14]

V případě, že je zvolen komunikační kanál bez synchronizace, je také použita FIFO fronta [13, s. 14]. Data jsou do fronty zapisována každou časovou jednotku do doby, než je fronta zcela zaplněna. Na druhé straně paralelní větev, jenž data z fronty čte provádí čtení opět každou časovou jednotku, až do doby, dokud není fronta prázdná. Zarovnání vstupně-výstupních operací s frontou podle časových jednotek zajišťuje synchronní chování.

SystemC

Programovací jazyk SystemC je určen pro popis, simulaci a modelování číslicových systémů. SystemC byl vyvinut asociací Open SystemC Initiative a později standardizován konzorciem IEEE jako IEEE 1666-2005 [14].



Obrázek 2.3: Architektura jazyka SystemC [31, s. 2]

Na obrázku 2.3 je znázorněna architektura jazyka SystemC. Ten vychází z jazyka C++, konstrukce přidané jazykem SystemC nad standard C++ jsou na obrázku 2.3 podbarveny

šedou barvou. Součásti, podporované jazykem, ale neuvedené ve standardu jazyka SystemC, jako například verifikační knihovny, knihovny definující práci s časem atd., jsou též znázorněny na obrázku 2.3. SystemC je do jisté míry ovlivněn i jazyky pro popis hardware VHDL a Verilogem.

Jazyk SystemC je dělen do modulů [31, s. 2]. Jednotlivé moduly komunikují pomocí portů, rozhraní a kanálů. Port je v jazyce SystemC chápán stejně, jako v jazycích VHDL či Verilog, tedy jako spojení vnitřní části modulu s jeho okolím. Modul pomocí portů komunikuje s rozhraním komunikačního kanálu.

Komunikační kanál definuje, jak jsou implementovány metody (funkce) rozhraní modulů [31, s. 3]. Komunikační kanály mohou být implementovány formou zásobníku, fronty FIFO, pomocí semaforů, atd.

Následující příklad ukazuje násobičku popsanou jazykem SystemC [30]:

```
#include "systemc.h"

SC_MODULE(adder)           // module (class) declaration
{
  sc_in<int> a, b;         // ports
  sc_out<int> sum;

  void do_add()            // process
  {
    sum.write(a.read() + b.read()); //or just sum = a + b
  }

  SC_CTOR(adder)           // constructor
  {
    SC_METHOD(do_add);     // register do_add to kernel
    sensitive << a << b;   // sensitivity list of do_add
  }
};
```

Zdrojový kód 2.4: Jednoduchá násobička v jazyce SystemC

„Čisté“ C

Proces programování číslicových obvodů pomocí „čistého“ jazyka C zahrnuje zejména překlad kódu do formátu HDL nebo RTL. Nejdůležitějším aspektem je tedy kvalita syntezátoru *C-to-HDL*.

Při návrhu číslicového obvodu v běžném programovacím jazyce (C, C++) a jeho překlad pro použití v číslicových systémech je nutné dodržovat jisté zásady. Špatným návrhem programu může při překladu dojít k problémům se splněním časových omezení nebo k problémům s nedostakem programovatelných logických členů (LUT, Flip-Flops, ...).

Mezi aplikace, které umožňují programování číslicových obvodů pomocí jazyka C patří:

- Catapult C od firmy *Mentor Graphics*.
- Handle-C od firmy *Celoxica*.
- Impulse C od firmy *Impulse Accelerated Technologies*.

Kapitola 3

Xilinx FPGA a EDK

Tato kapitola popisuje technologii FPGA z pohledu společnosti Xilinx. V úvodní části je popsán překladový systém využívaný v prostředí Xilinx Embedded Development Kit. Základní stavební jednotka projektů v EDK – IP-core je popsána v části 3.2.1. Pro anotaci IP-core se využívá Platform Specification Format (*PSF*), který je popsán v závěrečné části.

3.1 Proces překladu

V následující části bude popsán překlad souborů HDL (VHDL a Verilog) do konfiguračního řetězce (označován jako Bitstreamu, dle firmy Xilinx).

Obrázek 3.1 ukazuje jednotlivé fáze překladu HDL souborů do Bitstreamu, jsou to:

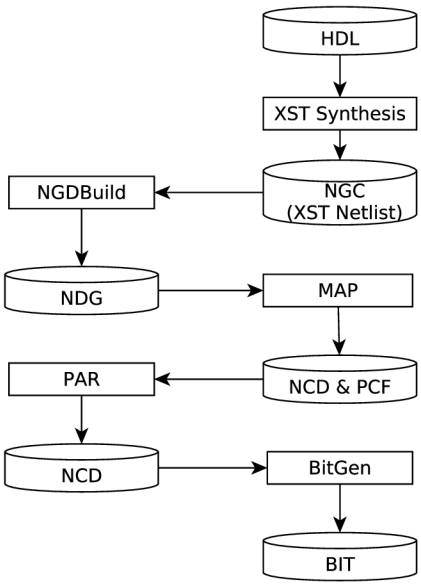
- syntézu HDL souborů,
- proces překladu do logických bloků FPGA (NGDBuild),
- mapování HDL reprezentace na logické bloky FPGA,
- proces umístění a propojení designu v FPGA (*Place and Route*),
- proces generování výsledného Bitstreamu.

3.1.1 XST syntéza

XST (*Xilinx Synthesis Technology*) syntézu je možné spustit nad uživatelskými VHDL soubory například z prostředí Xilinx ISE. Tento proces je možné spustit i dávkově z prostředí terminálu.

Na obrázku 3.2 jsou znázorněny vstupy, průchody a výstupy XST syntézy. Vstupem XST syntézy jsou soubory VHDL a Verilog. Dále vstupují do syntézy informace o omezeních výsledného hardwaru. Tato omezení mohou definovat například minimální frekvenci, na níž budou jednotlivé části hardware komunikovat nebo definují propojení jednotlivých bloků. Pomocí omezení lze nastavit, které piny či porty budou vyvedeny na rozhraní čipu FPGA [37, s. 305].

Společně se souborem omezení (*UCF*) vstupují do XST syntézy informace o použité technologii, jako je konkrétní typ FPGA, typ pouzdra FPGA a informace o časování FPGA [37, s. 307]. Podle těchto informací potom syntetizační nástroj provede syntézu.



Obrázek 3.1: Xilinx Software Design Flow (čipy FPGA)

Výstupem jsou tři soubory [37, s. 253], konkrétně jde o soubory NGR, ten obsahuje rozložení jednotlivých logických kompenent v rámci bloku. Dalším souborem je soubor NGC. Tento soubor obsahuje logický design i omezení (constraints) a je dále v práci označován jako Netlist.

3.1.2 Překlad do logických bloků FPGA (NGDBuild)

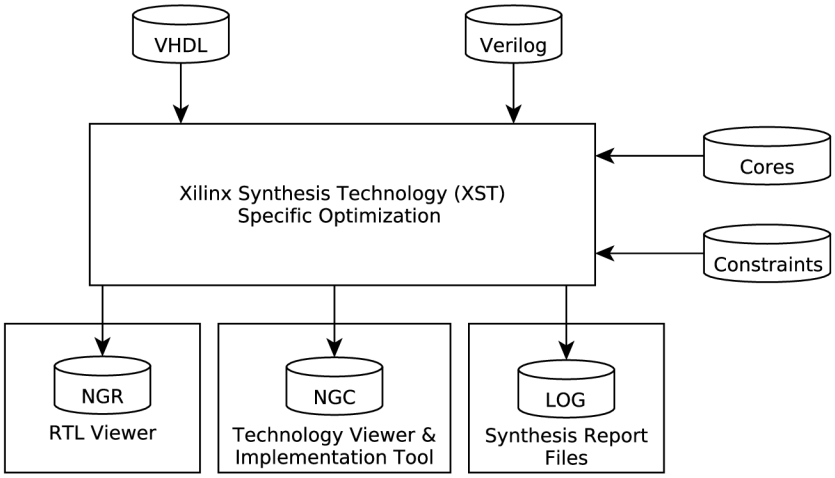
Vstupem NGDBuildu jsou netlisty v souborech NGC nebo EDIF 2.0.0 [3, s. 147]. Netlisty vznikají jako výstup XST syntézy (viz sekce 3.1.1).

Dalším elementem vstupujícím do procesu NGDBuild je soubor UCF (*User Constraints File*) [3, s. 147]. Tento ASCII soubor popisuje uživatelská omezení, a to buď časová nebo omezení týkající se rozložení komponent na čipu. Informace ze souboru omezení (UCF) jsou přidána do výsledného NGD souboru, jež je procesem NGDBuildu vygenerován. Soubor omezení může být napsán ručně programátorem nebo je možné jej vytvořit pomocí nástroje *Constraint Editor*.

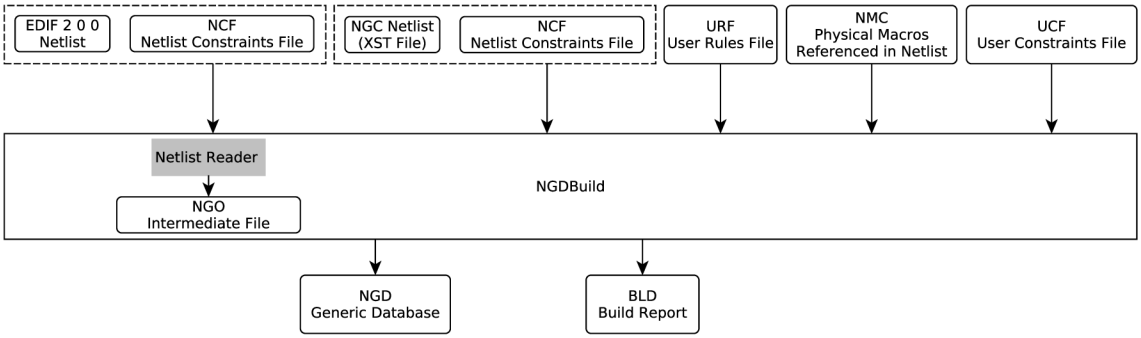
Další vstupní informací, která je v rámci volání NGDBuild využívána, jsou omezení vztažená k celému netlistu [3, s. 148]. Tato sada omezení je uložena v souboru NCF (*Netlist Constraint File*). Načítání omezení ze souboru (NCF) je vyvoláno procesem *Netlist Reader* (viz obr. 3.3) při zpracovávání vstupního netlistu. Informace ze souboru NCF jsou uloženy dočasného souboru NGO a do výsledného NGD souboru.

Dalším faktorem ovlivňujícím výslednou podobu souboru NGD je soubor s uživatelskými pravidly pro překlad [3, s. 148]. Jedná se o soubor URF (*User Rules File*). Ten obsahuje uživatelská pravidla, jež ovlivňují výslednou podobu designu, který vystupuje z procesu NGDBuildu. V rámci souboru uživatelských pravidel mohou být též definovány nástroje třetích stran, které budou použity během vytváření výstupu.

Dalším vstupním souborem je také výstupní soubor předešlého kroku překladu hardware, a to soubor NGC – výstup XST syntézy (viz sekce 3.1.1). Obsah tohoto binárního



Obrázek 3.2: Průchody a vstupy XST syntézy [36]



Obrázek 3.3: Vstupy a výstupy procesu NGDBuild [3, s. 146]

souboru byl již popsán dříve.

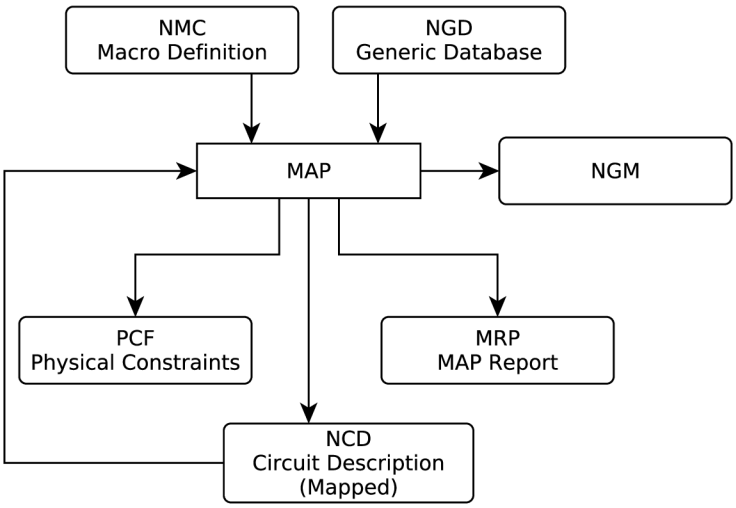
Uvnitř souboru NGC jsou instanciována fyzická makra (multiplexory, násobičky, děličky, atd.). Jejich implementaci obsahuje soubor NMC [3, s. 149]. NGDBuild tento soubor dostává na vstupu a jeho výstupem je funkcionální simulace těchto maker.

Hlavním výstupem procesu NGDBuild je soubor NGD. Jedná se o binární soubor obsahující logický popis hardware [3, s. 149]. Tento popis je v souboru NGD obsažen jednak ve formě hierarchie získané z předchozího kroku, ale také ve formě NGD primitiv, na něž byl vstupní design převeden.

Druhým výstupem NGDBuildu je soubor BLD. Jedná se o logovací soubor obsahující informace o provádění procesu NGDBuild, o podprocesech volaných v rámci NGDBuild (například o těch, jež byly volány pro splnění pravidel specifikovaných v souboru URF) [3, s. 149].

3.1.3 MAP

V následující části je popsán proces mapování popisu hardwaru na jednotlivé fyzické bloky obsažené uvnitř FPGA [3, s. 157]. Vstupem procesu map je soubor NGD, který vznikl v předešlém kroku překladu (viz sekce 3.1.2).



Obrázek 3.4: Vstupy a výstupy procesu MAP [3, s. 158]

Map nejprve provede DRC (*Design Rule Check*) [3, s. 172]. DRC se provádí nad designem obsaženým uvnitř NGD souboru. Po skončení DRC provede proces MAP mapování logického designu na fyzické komponenty obsažené uvnitř FPGA (*logické bloky, vstupně-výstupní bloky, ...*).

Výstupem procesu MAP je soubor NCD (*Netlist Circuit Description*) [3, s. 159]. Ten obsahuje fyzickou reprezentaci designu namapovaného na konkrétní elementy zvoleného FPGA (vstupně-výstupní buňky, LUT, BRAM, atd.). Soubor NCD je dále předán procesu `Place & Route` (viz sekce 3.1.4).

Vstupy procesu MAP

Nejdůležitějším souborem, jež vstupuje do procesu MAP je výstup předešlé části překladu, tedy soubor NGD (viz sekce 3.1.2).

Na vstupu očekává proces MAP soubor NMC. Tento soubor obsahuje definici fyzických maker [3, s. 158]. Tato fyzická makra jsou instanciována uvnitř souboru NGD a soubor NMC slouží jako skladiště jejich definic. Pro každý typ makra obsažený v NGC souboru existuje jeden NMC soubor.

Do procesu MAP vstupuje také soubor NGM obsahující informace o fyzickém rozložení komponent ustanoveném během předešlé iterace procesu MAP.

Výstupy procesu MAP

Hlavním výstupem procesu MAP je soubor NCD (*Native Circuit Description*), ten obsahuje informace o rozložení designu pomocí vnitřních bloků FPGA [3, s. 159].

Dalším produkovaným výstupem je soubor NGM, jenž obsahuje všechny informace o designu ze vstupního NGD souboru společně s informací o fyzickém rozložení tohoto designu vyprodukovaného v rámci procesu MAP .

Během procesu MAP vzniká ASCII soubor PCF (*Physical Constraints File*) obsahující omezení jednotlivých fyzických komponent na čipu.

Posledními soubory vystupujícími z procesu MAP jsou informační soubory MRP a MAP [3, s. 159].První zmíněný obsahuje soubor varování a chyb, které byly zachyceny během průchodu procesu MAP. Dále tento soubor obsahuje specifikaci atributů daného designu a v neposlední řadě také informuje o tom, jaká logika byla do designu přidána či odebrána, či jakým způsobem byly namapovány signály z designu na signály logických elementů FPGA. Druhý jmenovaný logovací soubor (soubor MAP) obsahuje informace o procesu MAP.

Design Rule Check

V rámci Design Rule Check (*DRC*) mohou být ověřeny různé problémy, které vzniknou během procesu MAP. Existuje několik typů kontrol [26, s. 197]:

- *Kontrola pinů* – tato kontrola ověřuje jeden nebo více připojených nebo nepřipojených signálů. Při tomto ověřování oznamuje DRC chyby s nesouhlasnými počty pinů, problémy s připojením pinů na třístavové buffery na vstupně-výstupních pinech, připojením plovoucích segmentů, atd.
- *Kontrola bloků* – ověřuje jeden nebo více připojených nebo nepřipojených bloků a oznamuje chyby spojené s logickou výstavbou bloku, s napojením pinů na blok nebo problémy s programováním daného bloku.
- *Kontrola čipu* – speciální typ kontroly ověřuje bloky a signály, ve smyslu celého čipu.

Testování mohou být volána samostatně nebo je lze v rámci DRC spojit do jednoho volání.

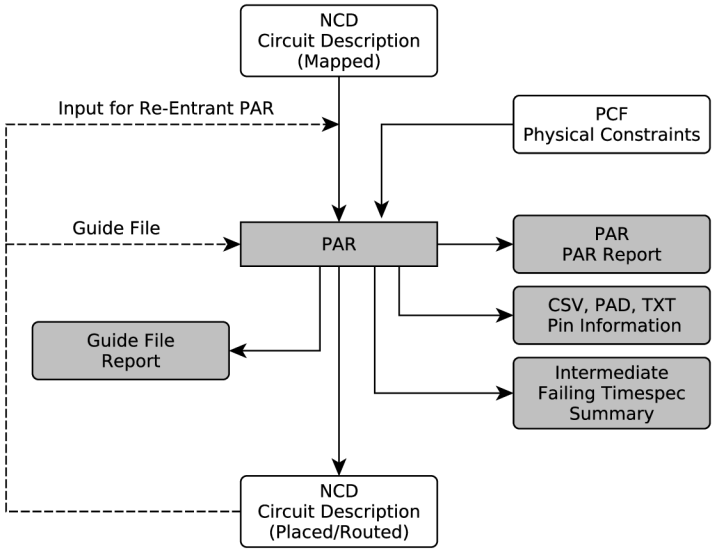
3.1.4 Place & Route

Proces Place & Route (*PAR*) je předposlední částí tvorby výsledného designu. Na svém vstupu vyžaduje soubor s informací o rozložení jednotlivých komponent designu (viz sekce 3.1.3). Provádění procesu PAR se skládá ze dvou podprocesů, `Place` a `Route`. Vstupní a výstupní soubory procesu PAR jsou znázorněny na obrázku 3.5.

Placing

Proces Placing provádí umístění jednotlivých komponent (vnitřních bloků) v designu s ohledem na zvolenou výslednou platformu. Při jednom běhu procesu Place se opakovaně spouští Placer a jeho výstup je postupně optimalizován pro co nejlepší využití plochy FPGA. Výsledný NCD soubor je zapsán až po dokončení všech běhů Placeru [3, s. 193].

Během umisťování designu Placerem do FPGA jsou zohledňována různá omezení vyplývající zejména ze souboru PCF, z délky jednotlivých propojů (zejména z důvodu splnění časových požadavků) a také omezení vyplývající z dostupnosti různých zdrojů v rámci různých částí FPGA.



Obrázek 3.5: PAR flow [3, s. 192]

Routing

Dalším krokem je spuštění procesu Routing (*propojení*). Ten opět v několika bězích optimalizuje komponenty uvnitř FPGA, ale nyní z hlediska splnění časových omezení [3, s. 193]. Provádí tedy „sbližování“ jednotlivých bloků tak, aby byla právě tato časová omezení splněna.

Po dokončení všech fází PAR je výsledek zapsán do souboru NCD [3, s. 194], ve kterém jsou jednotlivé bloky designu finálně umístěny a propojeny).

Vstupní soubory

Do procesu PAR vstupuje zejména soubor NCD z předchozí fáze překladu, (viz sekce 3.1.3). Jedná se o NCD soubor, který obsahuje pouze namapovaný design. Dalším vstupem je soubor PCF (*Physical Constraint File*) [3, s. 195], který obsahuje omezení fyzických prvků, z nich je namapován design. Tato omezení určují faktory upravující možnosti umístění daného prvku uvnitř designu.

Volitelně je možné vložit na vstup procesu PAR již jednou vygenerovaný NCD soubor vygenerovaný předchozím průchodem procesu PAR. Ten může být v dalším běhu lépe optimalizován.

Výstupní soubory

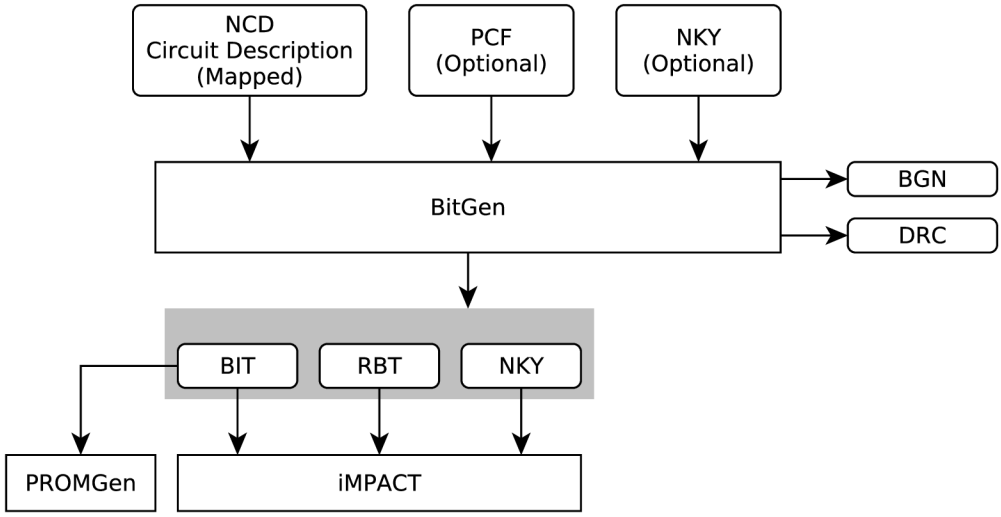
Z procesu PAR vystupuje řada souborů. Nejdůležitější z nich je upravený soubor NCD (obsahuje umístěný a propojený design). Ostatní výstupní soubory jsou již spíše informačního charakteru [3, s. 195]:

- Soubor PAR – logovací soubor s informacemi o jednotlivých instancích procesu PAR,
- Soubor PAD – parsovatelný soubor s informacemi o I/O pinech,

- Soubor CSV – soubor s informacemi o I/O pinech (pro tabulkové procesory),
- Soubor TXT – soubor s informacemi o I/O pinech (pro textové editory),
- Soubor XRPT – XML soubor obsahující informace o jednotlivých průchodech procesu PAR.

3.1.5 BitGen

BitGen je řádkový nástroj, který produkuje soubory potřebné pro konfiguraci čipu FPGA. Produkuje bitstream. Jedná se o konečnou fázi překladu. Vstupní a výstupní soubory nástroje BitGen shrnuje obrázek 3.6.



Obrázek 3.6: BitGen flow [3, s. 308]

Na svém vstupu očekává BitGen plně umístěný a propojený design (v souboru NCD). Soubor NCD je výstupem procesu PAR (viz sekce 3.1.4). Volitelně je možné na vstup procesu BitGen přivést soubory PCF (obsahuje fyzická omezení bloků FPGA) a soubor NKY (obsahuje šifrovací klíč). Při použití šifrovacího klíče je možné vytvořit šifrovaný bitsream, kde využívá 256 bitový AES klíč [35, s. 102].

Výstupem procesu BitGen je řada logovacích souborů. Nejdůležitějším souborem je soubor obsahující výsledný bitstream (`.bit`). Stejné informace jako soubor BIT obsahuje i soubor RBT (na rozdíl od souboru BIT ve formátu ASCII) [3, s. 308]. ASCII reprezentace bitstreamu je vhodná v případě, kdy programování FPGA provádí mikroprocesor.

Pokud byl na vstup procesu BitGen přiveden soubor NKY, je informace o dešifrování uložena ve výstupním souboru NKY [3, s. 308].

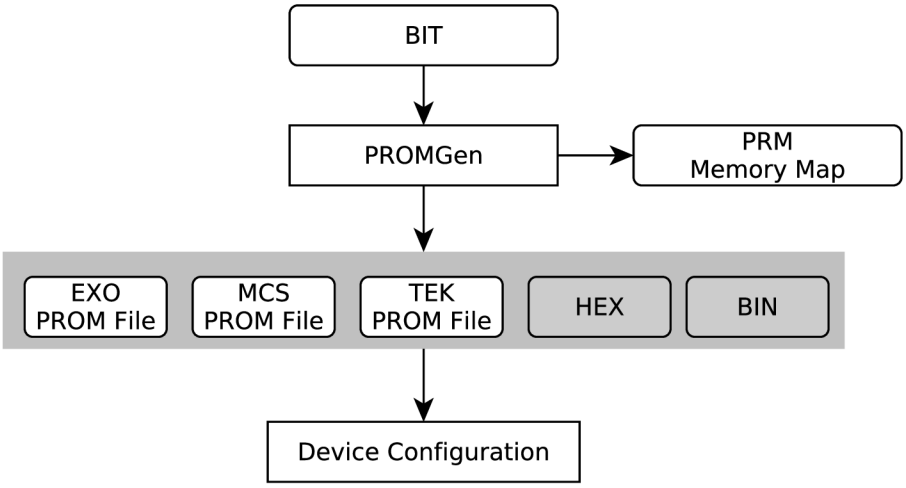
Mezi logovací soubory, které jsou BitGenem vytvořeny, patří soubor BGN obsahující informace o volání BitGen. Dále sem patří soubor DRC, který obsahuje informace o procesu Design Rule Check, který je prováděn během volání BitGen (viz sekce 3.1.3).

3.1.6 PROMGen

PROMGen transformuje bitstream generovaný nástrojem BitGen (viz 3.1.5) do formátu PROM, který obsahuje konfigurační řetězec pro jednu z těchto platforem [28, s. 319]:

- MCS-86 (*Intel*),
- EXORMAX (*Motorola*),
- TEKHEX (*Tektronix*).

Mimo tyto tři formáty je možné generovat výslednou konfiguraci do binárního nebo hexadecimálního formátu. Ten je vhodný v případě, kdy programování FPGA provádí mikroprocesor (stejně jako v případě ASCII u procesu BitGen).



Obrázek 3.7: PromGen flow [3, s. 345]

Vstupem procesu PROMGen je bitstream ve formátu BIT (viz obrázek 3.7). Výstupem potom soubory konfigurace ve formátu dle zvolené cílové platformy nebo v binárním či hexadecimálním formátu.

3.2 Návrh obvodů pomocí IP-core

Tato část popisuje IP core, základní jednotku, se kterou se v rámci EDK[1] pracuje. Je kladen důraz zejména na popis vnitřní struktury a rozdělení dle typů IP core.

3.2.1 Co je to IP core

Název IP core [16] vychází z anglického slovního výrazu *intelectual property core* (*intelektuální vlastnictví*) je základní stavební jednotkou hardware. V ideálním případě by dané IP core mělo být přenositelné a snadno znovu použitelné v jiných projektech. IP core je možné rozdělit do tří tříd podle možností konfigurace v cílovém systému:

[1] Embedded Development Kit

- *Hard Core*,
- *Firm Core*,
- *Soft Core*.

IP core typu *Hard Core* obsahuje hardwarovou komponentu s určitou specifickou funkcí a nastavením. Je tedy dále nekonfigurovatelný. Tento typ je nejvýhodnější pro rychlé prototypování aplikací, jelikož jej návrhář jen vloží do připravovaného hardware a nemusí dále pracovat na jeho dodatečném nastavení. Příkladem je NetCOPE převáděný v této práci.

Naopak *Firm Core* (popřípadě *Semi-Hard Core*) neobsahuje takovou míru uzavření a jako takový nabízí jistou míru konfigurovatelnosti. Příkladem je vstupní buffer (IBUF) využívaný v síťovém modulu NetCOPE, kde je možné nastavit velikosti vstupních front.

A konečně IP core typu *Soft Core* je, co se týče míry konfigurovatelnosti, pro návrháře nejflexibilnější a nabízí také nejvyšší míru konfigurovatelnosti. V reálných systémech bývají IP core typu *Soft Core* nejčastěji reprezentovány HDL soubory. Naopak IP core typů *Hard Core* a *Firm Core* jsou nejčastěji reprezentovány soubory *NGC*. Příkladem Soft Core IP-core jsou korporátní IP-core, například IP-core DMA modulu NetCOPE.

3.2.2 Implementace IP-core

IP-core lze vnitřně reprezentovat více způsoby, které jsou založené na použitém programovacím jazyce. Je možné, aby bylo IP-core reprezentováno jazyky HDL (viz 2.2.1). Lze však použít i vysokoúrovňové jazyky (viz 2.3).

3.2.3 Manuální integrace IP-core

Pro manuální integraci IP-core do číslicového obvodu je nutná znalost rohraní přidávaného IP-core. S touto znalostí je následně nutné specifikovat napojení IP-core do číslicového obvodu.

Většinou je také nutné vyřešit samostatně překlad tohoto systému (viz 3.1.1) a splnit všechny požadavky, které vyžaduje přidávaný IP-core. Pokud například navzájem neodpovídají programovací jazyky číslicového systému a přidávaného IP-core, je nutné ručně tento problém vyřešit.

3.2.4 Integrace IP-core pomocí IDE

V případě, že je pro integraci IP-core do číslicového obvodu použito IDE, odpadá nutnost samostatného řešení překladu obvodu. Tuto činnost IDE řeší samostatně. Pokud využijeme příklad z předchozí části, pokud používáme IP-core, které využívá reprezentaci rozdílnou od námi navrhovaného obvodu, řesí tyto problémy překladový systém IDE.

Naopak je nutné anotovat hardware v IP-core pro použití v daném IDE. V případě této práce se využívá software Xilinx Embedded Development Kit (viz 3.3), kde se pro anotaci hardware IP-core používá Platform Specification Format (viz 3.3.1).

Většinou také existují omezení ohledně vnitřní reprezentace IP-core. Anotace IP-core pro použití v určitém IDE. Anotační formát obvykle dovoluje použití jen určité množiny programovacích jazyků, jimiž může být IP-core vnitřně reprezentováno.

3.3 Software Xilinx Embedded Development Kit

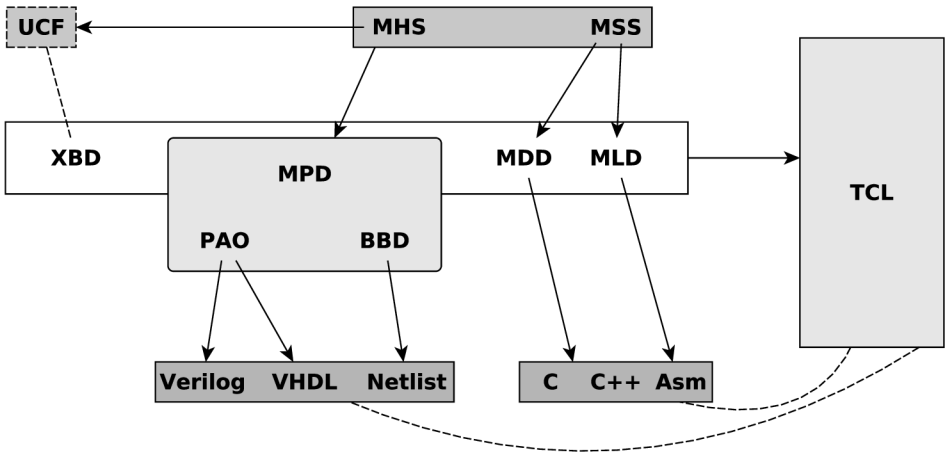
Vývojové prostředí Xilinx Embedded Development Kit obsahuje sadu nástrojů a předpřipravených definic IP-core. Pomocí nástrojů obsažených v prostředí je možný celkový návrh, implementace a zprovoznění hardware pro čipy typu FPGA.

Software Xilinx Embedded Development Kit je možné rozdělit na dvě části [4, s. 25]:

- Překladový systém a Platform Specification Format,
- Programové vybavení pro vytváření projektů pomocí definic IP-core – Xilinx Platform Studio (XPS).

3.3.1 Platform Specification Format (PSF)

Platform Specification Format slouží pro reprezentaci metadat, na základě kterých provede překladový systém syntézu výsledného hardware. PSF také popisuje, jak bude k výslednému hardwaru připojen software, jaké ovladače se použijí pro obsluhu periferií a jaké knihovny je třeba mít k dispozici. Na základě těchto metadat probíhá syntéza designu programem `platgen` a překlad softwaru programem `libgen` [5, s. 1]. Tato metadata jsou obsažena v souborech MSS (*Microprocessor Software Specification*) a MHS (*Microprocessor Hardware Specification*).



Obrázek 3.8: Schéma Platform Specification Format [5, s. 1]

Dalšími soubory tvořícími formát PSF jsou: MPD (*Microprocessor Peripheral Definition*), MLD (*Microprocessor Library Definition*), MDD (*Microprocessor Driver Definition*), BBD (*Black Box Definition*), PAO (*Peripheral Analyze Order*), XBD (*Xilinx Board Description*) a TCL (*Tool Command Language*).

3.3.2 Procesor MicroBlaze

Jedná se o 32-bitový RISC mikroprocesor s harvardskou architekturou [21]. Je určen pro použití v FPGA.

Mikroprocesor MikroBlaze nabízí přes 70 uživatelsky definovaných nastavení, která umožňují flexibilní použití tohoto mikroprocesoru. Od mikroprocesoru, který podporuje

jednoduchá stavová zařízení po mikroprocesorové systémy podporující řízení systémem na bázi Linuxu. Srovnání výkonnosti jednotlivých konfigurací MicroBlaze nabízí tabulka 3.1. Nastavení výkonu a plochy mikroprocesoru se provádí v rámci průvodce při vkládání mikroprocesoru do výsledného designu.

Device Family	**Performance Optimized MicroBlazewith branch optimizations (5-stage pipeline) 1.38 DMIPs/MHz**	**Performance Optimized MicroBlaze (5-stage pipeline) 1.30 DMIPs/MHz**	**Area Optimized MicroBlaze (3-state pipeline) 1.03 DMIPS/MHz**
Zynq-7000 SoC (-3)	228DMIPs	259DMIPs	196DMIPs
Virtex-7 FPGA (-3)	293DMIPs	393DMIPs	264DMIPs
Kintex-7 FPGA (-3)	317DMIPs	408DMIPs	264DMIPs
Virtex-6 FPGA (-3)	306DMIPs	384DMIPs	246DMIPs
Spartan-6 FPGA (-4)	166DMIPs	209DMIPs	152DMIPs

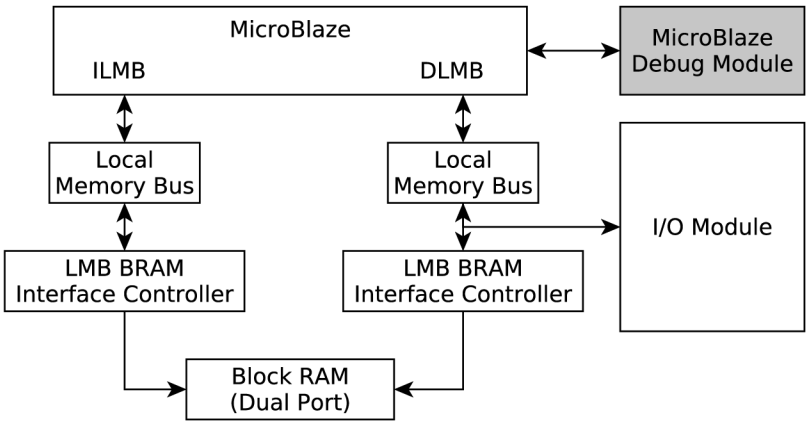
Tabulka 3.1: Možnosti nastavení výkonnosti mikroprocesoru MicroBlaze [21]

Procesor MicroBlaze se k ostatním komponentám v FPGA připojuje jednou ze sběrnic [1, 20, s. 6, s 1]:

- AXI – pro FPGA řady Virtex6 a vyšší,
- PLB – pro FPGA řady Virtex5 a nižší.

3.3.3 MicroBlaze Micro Controller System

MicroBlaze Micro Controller System (MCS) je samostatný systém pro řízení aplikací, jehož součástí je i mikroprocesor MicroBlaze. Schéma MCS je znázorněno na obrázku 3.9.

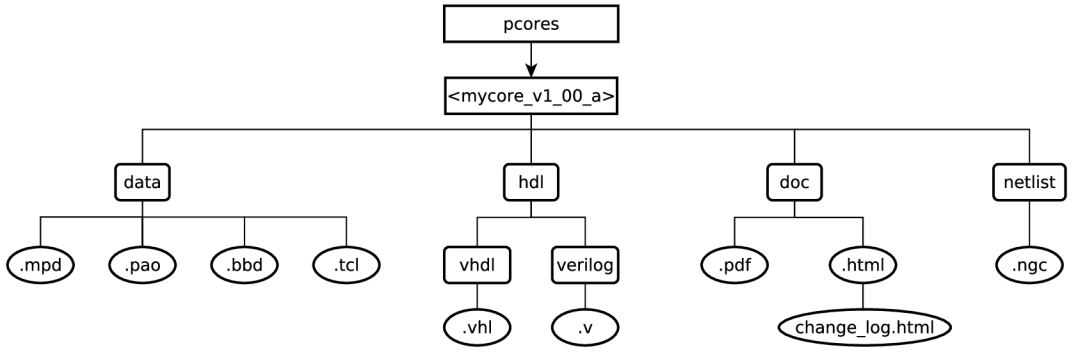


Obrázek 3.9: Vnitřní uspořádání MicroBlaze MCS [19, s. 2]

Data společně s programem jsou uloženy v lokální paměti, ladění je volitelnou součástí MCS a obstarává ho MicroBlaze Debug Module. Součástí MCS je také sada periferií pro obsluhu přerušení, poskytující rozhraní UART, atd [19, s. 1].

3.4 Anotace hardware IP-core pomocí PSF

V následující části bude popsán postup anotace hardware popsaného v jazyce VHDL do formátu použitelného v rámci EDK, tedy do formátu PSF. Budou zde popsány dva možné způsoby, jakým může být výsledný IP-core specifikován.



Obrázek 3.10: Struktura formátu PSF [27, s. 10]

3.4.1 Syntéza VHDL souborů

Překlad souborů VHDL do souborů NGC je vhodný zejména tam, kde je plánován export výsledných IP-core třetím stranám a kde nechceme poskytovat koncovému uživateli zdrojové kódy hardware (viz *Hard Core* v části 3.2.1).

Pro překlad souborů VHDL do souborů NGC je nutné, aby byla nad soubory VHDL spuštěna XST syntéza. Jak je popsáno v kapitole 3.1.2 je výstupem XST syntézy soubor NGC. Ten pak může být využit v rámci IP-core, fyzicky je uložen v adresáři `netlist` ve struktuře IP-core (viz obrázek 3.10).

3.4.2 Použití čistého HDL v IP-core

Další možností reprezentace specifikace hardware v IP-core je vložení souborů VHDL do adresáře `hdl/vhdl` [27, s. 11] (viz obrázek 3.10). Zde jsou soubory popisu hardwaru v plain-text podobě, proto je tento způsob vhodný, pokud chceme koncovému uživateli poskytnout zdrojové kódy hardware.

3.4.3 Obsah jednotlivých složek anotace IP-core

Pro vytvoření nové anotace IP-core je nutné vytvořit sadu souborů a složek v předem dané struktuře a s daným obsahem (viz [27, s. 10]). Struktura anotačních metadat IP-core je znázorněna na obrázku 3.10.

Soubor `MPD`

Soubor MPD (*Microprocessor Peripheral Definition*) je uložen ve složce `data`. Obsahuje informace o rozhraní anotovaného IP-core [27, s. 27].

Ke každému souboru MPD musí ve slože `hdl/vhdl` nebo `hdl/verilog` existovat HDL. Struktura tohoto souboru a informace o něm jsou popsány v kapitole 3.4.3.

Soubor MPD definuje:

- Seznam portů, jejich typ a výchozí připojení na sběrnice,
- Seznam parametrů a jejich výchozích hodnot.

Struktura souboru MPD je následující [27, s. 28]:

```
BEGIN jmeno_periferie

## Nastaveni periferie
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
OPTION STYLE = BLACKBOX
OPTION LAST_UPDATED = 9.2
OPTION DESC = Popis periferie
OPTION LONG_DESC = Detailni popis periferie
OPTION IP_GROUP = Skupina v katalogu XPS
OPTION ARCH_SUPPORT_MAP = ( virtex5lx=PREFERRED, others=
 DEPRECATED)

## Porty
PORT PCLK250_P = "", DIR = I
PORT PET_N = "", DIR = O, VEC = [7:0]
PORT PCLK0_P = "", DIR = IO, THREE_STATE=FALSE
...
END
```

Zdrojový kód 3.1: Ukázková struktura souboru MPD

Jak je vidět ze zdrojového kódu 3.1, je soubor MPD uveden direktivou `BEGIN` následovanou názvem periferie definované v daném souboru.

Následuje část, v níž jsou uvedena nastavení konkrétní periferie. Jedná se například o typ periferie (`IPTYPE`) [27, s. 38], zde je možné nastavit, zda se jedná o periferii adresivatelnou na sběrnici (`PERIPHERAL`) nebo půjde o komponentu ovládající sběrnici (`BUS`), a nebo zda bude popisován mikroprocesor (`CPU`).

Dalším parametrem ovlivňujícím výslednou periferii je nastavení označené jako `IMP_NETLIST` [27, s. 37]. To definuje, zda je hardware popsán pomocí HDL, nebo je definován pomocí netlistů. Přiřaditelná hodnota je tedy `TRUE` nebo `FALSE`.

Tento parametr je následován specifikací jazyka HDL (parametr `HDL`) [27, s. 37]. Možné jsou tři hodnoty a to `VHDL`, `Verilog` a nebo `MIXED`.

Parametr `STYLE` [27, s. 40] definuje, jakým způsobem je popsána periferie. Existují tři možnosti, které jsou uvedeny v tabulce 3.2:

Hodnota *STYLE*	**Popis**
HDL	Popsán jen HDL soubory (výchozí možnost)
BLACKBOX	Popsán jen pomocí netlistů
MIX	Popsán jak pomocí HDL, tak pomocí netlistů

Tabulka 3.2: Možnosti parametru *STYLE* [27]

Následuje část obsahující informace o periferii, které se zobrazí v IP katalogu programu Xilinx Platform Studio. Konkrétně jde o možnosti označené jako `DESC`, `LONG_DESC` a `IP_GROUP`.

V poslední částí pro nastavení možností IP-core jde o nastavení podporovaných rodin a typů FPGA. V části `ARCH_SUPPORT_MAP` je uveden seznam zařízení společně s atributem, který určuje míru kompatibility IP-core pro danou programovatelnou logiku. Přehled všech možností a jejich význam je uveden v tabulce 3.3.

Hodnota *ARCH_SUPPORT_MAP*	**Popis**
PREFERRED	IP-core je plně dostupný pro dané FPGA
AVAILABLE	IP-core je dostupný, ale uživateli je zobrazena informace o možné nedostatečné kompatibilitě
BETA	IP-core je v beta verzi
DEPRECATED	IP-core je možné použít, ale EDK informuje o zastarání
DEVELOPMENT	IP-core je ve vývoji a je syntetizován při každém novém překladu
EARLY_ACCESS	IP-core je pouze ukázkovou verzí a EDK informuje o možných problémech
OBSOLETE	IP-core je zastaralé (vyšší priorita než DEPRECATED) a EDK zobrazí chybu

Tabulka 3.3: Význam jednotlivých hodnot parametru *ARCH_SUPPORT_MAP* [27, s. 35]

Za částí obsahující nastavení parametrů IP-core následuje výčet všech portů vyvedených na rozhraní dané periferie (viz kód 3.2). Každý port je uveden klíčovým slovem `PORT` následovaným názvem daného portu, označením sběrnice, na niž je port připojen. Dále je specifikován směr portu a v případě vstupně-výstupních portů je uvedeno, zda má být použit třístavový buffer pro řízení vstupně-výstupních pinů.

```
PORT PCLK250_P = "", DIR = IO, THREE_STATE=FALSE
```

Zdrojový kód 3.2: Informace o jednom portu

Soubor `BBD`

Soubor BBD (*Black Box Definition*) obsahuje seznam netlistů, které jsou používány v rámci daného IP-core (a volitelně i pro danou rodinu FPGA). Jedná se o look-up tabulku, kde je uveden seznam souborů potřebný pro syntézu daného designu.
Formát takového souboru BBD je následující [27, s. 82]:

```
FILES
blackbox1.ngc, blackbox2.ngc, blackbox3.edn
```

Jak již bylo uvedeno výše, je možné vztáhnout seznamy netlistů ke konkrétní rodině čipů FPGA. Ukázka takového souboru následuje [27, s. 82]:

```
C_FAMILY    C_BUS_CONFIG          FILES
virtex           1                virtex/ip1.edf
virtex           2                virtex/ip2.edf
virtexe          1                virtex/ip1.edf
```

```
virtexe          2                virtex/ip2.edf
virtex2          1                virtex2/ip1.edf
virtex2          2                virtex2/ip2.edf
virtex2p         1                virtex2/ip1.edf
virtex2p         2                virtex2/ip2.edf
```

To zda budou použity soubory BBD ovlivňuje parametr STYLE v souboru MPD. Pokud je u parametru STYLE v souboru MPD uvedena hodnota MIX nebo BLACKBOX, jsou soubory BBD využívány.

Soubor `PAO`

Soubor PAO (*Peripheral Analyze Order*) obsahuje seznam souborů, které jsou nutné pro syntézu výsledného designu. Pokud je v souboru MPD u parametru STYLE uvedena hodnota MIX nebo VHDL, jsou využívány i soubory PAO. Formát souboru PAO je následující [27, s. 74]:

```
tooltarget libraryname <relative path from library>/filename[.v |.vhd] hdlang
```

Kde význam jednotlivých částí je následující:

- *tooltarget* – specifikuje, zda je IP-core použit pro syntézu (hodnota *synlib*), nebo je IP-core použit pro simulaci (hodnota *simlib*) a nebo pro oba procesy (hodnote *lib*).
- *libraryname* – specifikuje název IP-core ve tvaru libraryname_v1_00_a,
- *filename* – specifikuje název souboru s popisem hardwaru (nebo cestu k obálce dané periferie). Nemusí obsahovat příponu pro daný typ jazyka.
- *hdllang* – specifikuje jazyk v němž byl hardware popsán (tato informace je vyžadována, pokud je v souboru MPD nastaven parametr HDL na hodnotu MIXED.

Zdrojový kód 3.3 [27, s. 74] ukazuje soubor PAO popisující design složený pouze z VHDL komponent. Naopak zdrojový kód 3.4 [27, s. 74] ukazuje soubor PAO s různou hodnotou parametru *tooltarget*. Navíc každý typ knihovny je vytvořen v jiném HDL jazyce (*VHDL* a *Verilog*).

```
lib common_v1_00_a common_types_pkg.vhd
lib common_v1_00_a pselect.vhd
lib opb_gpio_v1_00_a gpio_core
lib opb_gpio_v1_00_a opb_gpio
```

Zdrojový kód 3.3: Ukázka souboru PAO popisující jen VHDL komponenty

```
lib ipname_v1_00_a file3.vhd vhdl
simlib ipname_v1_00_a simfile.v verilog
synlib ipname_v1_00_a synfile.vhd vhdl
```

Zdrojový kód 3.4: Ukázka souboru PAO obsahující simulační a syntezační části design

Soubor `TCL`

Soubor TCL [3, s. 53] obsahuje uživatelské skripty popsané pomocí jazyka *Tool Command Language*. Pomocí skriptů jazyka TCL je možné ovládat činnosti překladu dávkově, bez použití grafického uživatelského rozhraní. TCL skripty také mohou rozšířit základní funkcionalitu grafického uživatelského rozhraní o další funkce.

Soubory HDL

Soubory HDL popisují strukturu dané periferie ve zvoleném HDL jazyce [27, s. 11]. Informace o tomto jazyce musí být totožná s informací uvedenou u parametru `HDL` v souboru MPD (viz 3.4.3). Název souboru musí také odpovídat hodnotě uvedené v souboru PAO (viz 3.4.3). HDL soubory mohou být uloženy ve složce `hdl/vhdl` respektive `hdl/verilog`.

V případě, že je periferie vnitřně specifikována pomocí souborů netlistů (NGC), je v této složce uložena HDL obálka s popisem rozhraní této komponenty. Ukázka HDL obálky je ve zdrojovém kódu 3.4.3.

```
library ieee;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;

entity combov2_core_top is
port(
  --Seznam portů entity
  PCLK250_P       : in std_logic
);

end entity;

architecture use_core of combov2_core_top is
component combov2_core is
port(
  --Seznam portů architektury
  PCLK250_P       : in std_logic
);

end component;

begin
combov2_core_i: combov2_core
port map (
  --Mapování portů entity na porty uvedené v popisu
 architektury
  PCLK250_P => PCLK250_P
);

end architecture;
```

Je nezbytně nutné, aby název entity, v našem případě `combov2_core_top` odpovídal s názvem periferie uvedeným v souboru MPD.

Soubory dokumentace

Soubory dokumentace jsou volitelnou součástí IP-core periferie. Pokud mají být v rámci IP-core použity, jsou uloženy ve složce `doc`. Pokud se návrhář IP-core rozhodne dokumentaci do IP-core nezahrnovat, nemusí se složka `doc` vytvářet.

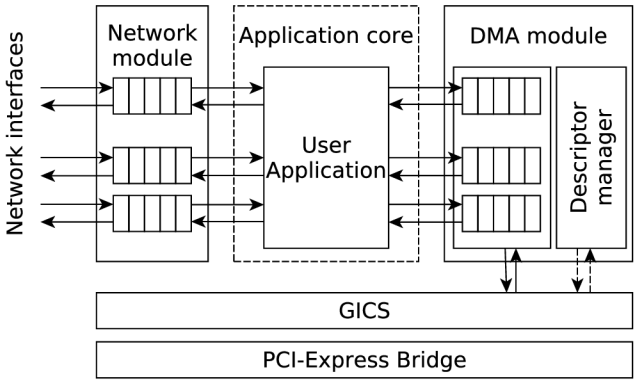
Kapitola 4

Platforma NetCOPE

V následující kapitole se věnuji platformě NetCOPE, jejímu popisu a specifikaci jejího použití. Také následuje přehled akceleračních karet, v nichž je platforma NetCOPE používána.

4.1 Popis NetCOPE

NetCOPE je konfigurovatelný vývojový framework navržený pro rychlé prototypování vysokorychlostních aplikací akcelerovaných pomocí čipů FPGA [24]. Samotný framework NetCOPE se skládá ze dvou částí, a to z části softwarové (ta pro tuto bakalářskou práci není podstatná) a z části hardwarové. Tato část je v této práci převáděna do EDK. Architektura platformy NetCOPE je znázorněna na obrázku 4.1.



Obrázek 4.1: Vnitřní uspořádání hardwarové vrstvy platformy NetCOPE [12]

Platforma NetCOPE vytváří abstraktní vrstvu nad čipem FPGA, a tak umožňuje snadnější prototypování bez znalostí konkrétních specifikací dané rodiny čipů FPGA. Tato vlastnost dává programátorovi výhodu koncentrovat se primárně na vývoj aplikací.

Hardwarová část platformy NetCOPE obsahuje tyto části [24]:

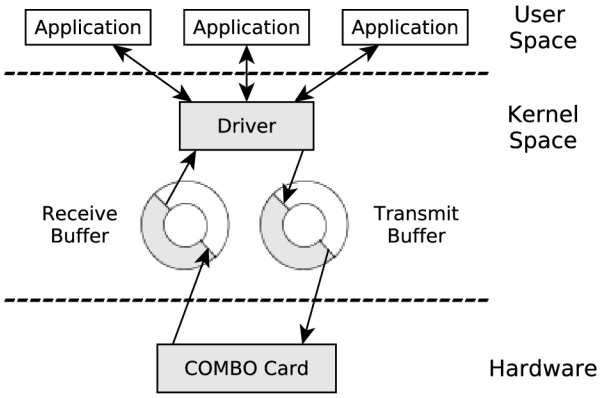
- I/O bloky (Network module) viz 4.1.2,
- Generický propojovací systém (*GICS*) [7],
- Rychlý DMA řadič (DMA module).

Následující seznam nastiňuje možnosti aplikačního využití karet s platformou NetCOPE [38]:

- síťová karta s hardwarovou filtrací,
- aktivní/pasivní síťový monitoring,
- IPv4/IPv6 směrovač,
- kryptografie,
- generování paketů, ...

4.1.1 Realizace DMA přenosů v NetCOPE

Rychlé DMA přenosy jsou realizovány mezi akceleračním jádrem v komponentě COMBOv2 Core a mezi RAM pamětí v PC [24]. Realizace rychlých DMA přenosů využívá dvou kruhových bufferů (viz obrázek 4.2). Jeden z nich je umístěn v RAM paměti PC, používá se k přístupu k datům ze strany uživatelské aplikace [24]. Druhý kruhový buffer je umístěn v FPGA, kde zpřístupňuje data akcelerační jednotce uvnitř komponenty COMBOv2 Core. Paměť bufferu je fyzicky rozdělena na stránky o velikosti 4 kB a v softwaru je neustále mapována do adresového prostoru uživatelské aplikace.



Obrázek 4.2: Znázornění kruhových bufferů DMA přenosů v architektuře NetCOPE [24]

4.1.2 Network module

Network module tvoří spojení mezi vnějším síťovým rozhraním a vnitřními bloky NetCOPE. Každé síťové rozhraní je osazeno vstupním a výstupním bufferem.

Pro komunikaci mezi kartou a okolní sítí je karta osazena čtyřmi 1 GB nebo dvěma 10 GB transceivery. Je tak podporován přenos jak rychlostí 1 Gb/s tak 10 Gb/s. Pro výměnu paketů mezi hardwarem COMBOv2 karty a aplikací vytvořenou v kartě je použita sběrnice FrameLink.

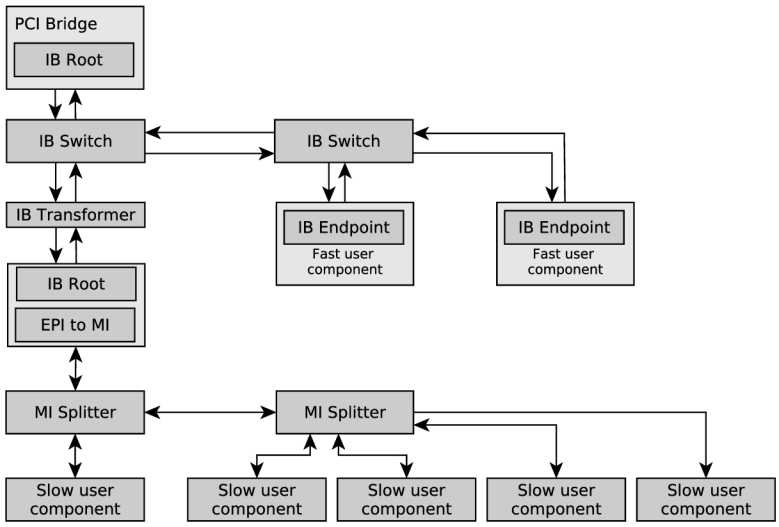
Bloky vstupního síťového rozhraní implementují příjem paketů podle standardu IEEE 802.3 [38]. Každému tomuto bloku je možné nastavit různé parametry, jako jsou:

- rychlost síťového rozhraní,
- maximální a minimální velikost přijímaného paketu,
- parametr pro zastavení přenosu.

Všechny tyto parametry je možné nastavit za běhu aplikace na COMBOv2 kartě. O nastavování těchto parametrů se starají softwarové nástroje.

4.1.3 Propojovací systém

Propojovací systém (*GICS*) slouží pro komunikaci mezi softwarem v počítači a COMBOv2 kartou [38]. Respektive pro komunikace mezi komponentami umístěnými v FPGA na COMBOv2 kartě a sběrnicí na systémové desce (PCI nebo PCI-Express). Vnitřní architektura propojovací systému je uvedena na obrázku 4.3.



Obrázek 4.3: Architektura propojovacího systému [22]

Při přenosu informací ze systémové sběrnice do COMBOv2 karty dokáže propojovací systém distribuovat dotaz konkrétní komponentě v FPGA (*Slave mode*). Naopak v případě, kdy dochází k přenosu dat z čipu FPGA na systémovou sběrnici, slouží propojovací systém jako prostředník a řídí činnost na sběrnici (*Bus Master mode*).

4.2 Sběrnice na platformě NetCOPE

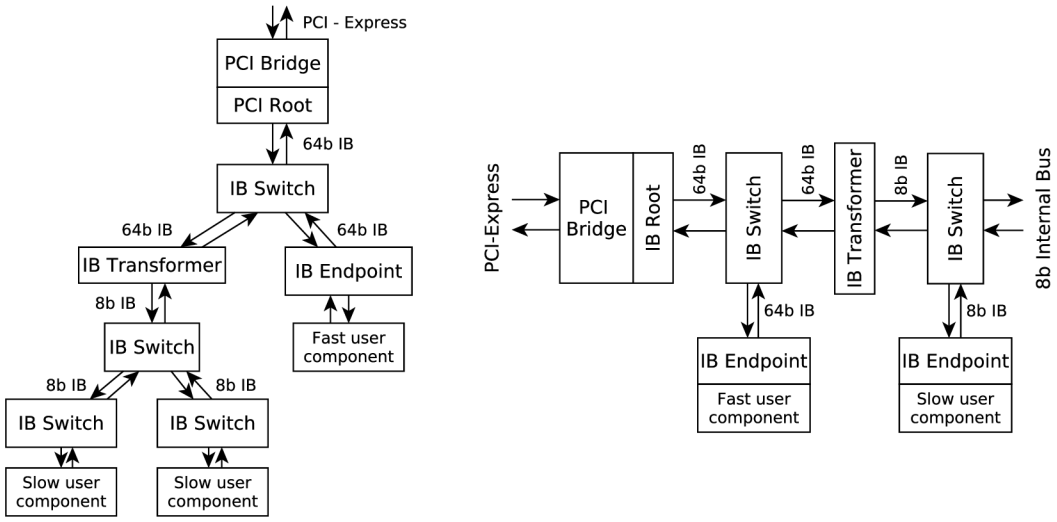
Platforma NetCOPE využívá několika druhů interních sběrnic dle jejich funkce. Následující kapitola shrnuje informace o těchto sběrnicích.

4.2.1 Sběrnice TRN

Sběrnice TRN (*Transaction*) umožňuje platformě NetCOPE komunikovat přes rozhraní PCI-Express. Sběrnice TRN je korporátní sběrnicí společnosti Xilinx [18, s. 1].

4.2.2 Sběrnice InternalBus

Sběrnice InternalBus je vysokorychlostní konfigurovatelná sběrnice s velkou propustností. Je určena pro rychlé přenosy mezi interními komponentami na FPGA, stejně jako pro přenosy mezi FPGA a rozhraním PCI Express.



Obrázek 4.4: Architektura sběrnice InternalBus [23]

Jak ukazuje obrázek 4.4, sběrnice InternalBus je složena z komponent *root*, *switch*, *transformer* a *endpoint*.

Komponenta *root* je částí *Host Bus Bridge (PCI Bridge)* sběrnice, která vytváří spojení mezi sběrnicí InternalBus a rozhraním PCI [23].

Komponenta *switch* propojuje jednotlivé větve a směřuje jednotlivé transakce mezi komponentami nebo FPGA a rozhraním [23]. Pro spojení dvou větví s rozdílnou datovou šířkou jsou využity komponenty *transformer*.

Komponenta *endpoint* slouží pro připojení uživatelských komponent k platformě NetCOPE.

4.2.3 Sběrnice MI32

Sběrnice MI32 je určena pro pomalejší komunikaci. Pomocí sběrnice MI32 je možné konfigurovat jak uživatelskou část aplikace v platformě NetCOPE, tak Network module a DMA module.

4.2.4 Sběrnice FrameLink

FrameLink [11] je sběrnice pro přenos dat ve formě paketů. Vychází ze sběrnice LocalLink od společnosti Xilinx [17], avšak obsahuje několik rozdílů. Např. data jsou zarovnána na

hranici slova (v jednom slově nemůžou být dvě části paketu), signály SOP a EOP ohraničují každou část rámce, atd. Tabulka 4.1 shrnuje popis signálů sběrnice FrameLink.

Signál	**Popis signálu**
CLK	Hodinový signál rozhraní.
SOF_N	Start of Frame indikuje hodinový cyklus, od kterého jsou na sběrnici vysílána data jednoho rámce. Společně s tímto signálem musí být nastaven i signál SOP_N.
SOP_N	Start Of Part indikuje hodinový cyklus obsahující první části dat. Jeden rámec (frame) může obsahovat více část dat, která musí být ohraničena signály SOP_N a EOP_N.
EOP_N	End Of Part indikuje hodinový cyklus obsahující poslední část dat.
EOF_N	End Of Frame indikuje hodinový cyklus, kdy je ukončeno vysílání jednoho rámce na sběrnici. Společně s ním musí být nastaven i signál EOP_N.
SRC_RDY_N	Source Ready indikuje, že je odesilatel připraven k odeslání dat
DST_RDY_N	Destination Ready indikuje, že je příjemce připraven pro příjem dat
DATA	Přenášené datové slovo libovolné datové šířky. V platformě NetCOPE jsou používány datové šířky o základu druhé mocniny. Nejběžněji 16, 32, 64 a 128 bitové šířky.
DREM	Data Reminder. Tento signál je čten příjemcem, když je nastaven signál EOP_N. Signál DREM obsahuje informaci o posledním platném datovém bitu aktuální části. Pozice posledního platného bitu je binárně kódována v signálu DREM (šířka signálu DREM je log2(n)). Pokud DREM = 0, potom pouze LSB je platný.

Tabulka 4.1: Přehled signálů sběrnice FrameLink [11]

4.3 Rodina karet COMBOv2

Rodina karet COMBOv2 vznikla v roce 2008 v rámci sdružení CESNET, kde byla vytvořena jako náhrada za starší hardwarovou platformu COMBO. Karty rodiny COMBOv2 jsou osazeny FPGA čipem Virtex-5, jehož výrobcem je společnost Xilinx. FPGA čip zajišťují vysokou flexibilitu karet COMBOv2.

Pro připojení karty k hostitelskému počítači slouží sběrnice PCI Express x8. Do rodiny COMBOv2 karet patří několik typů karet lišících se maximálními možnými přenosovými rychlostmi. Jde například o karty v konfiguracích 2x10 GB/s nebo 4x1 GB/s. Zajímavostí celé rodiny karet COMBOv2 je karta COMBOL-GPS poskytující přesný hodinového a PPS (*pulse per second*) signál. [10].

Platforma NetCOPE se využívá na řadě interface karet odvozených právě z COMBOv2 karet, které jsou na trh dodávány společností INVEA-TECH. Tyto FPGA karty jsou určeny pro vývoj akcelerovaných, časově kritických aplikací.

Karty jsou uzpůsobeny pro osazení optickými či metalickými síťovými transceivery. Jelikož jsou vybaveny výkonnými čipy FPGA, umožňují zpracování informací z vysokorychlostních počítačových sítí. Podporovány jsou rychlosti v řádech jednotek až desítek gigabitů. V současné době se připravuje karta podporující rychlost 100 Gb/s. Tato karta bude fyzicky osazena čtyřmi 25 G transceivery [6]. Rychlá komunikace mezi kartou a klientským

počítačem je zajištěna použitím sběrnice PCI-Express.

Jako příklad na trhu dostupných karet využívajících vývojový framework NetCOPE, uvádím tyto [8]:

- COMBO-4G: *4x1Gb transceiver*,
- COMBO-20G: *2x10Gb transceiver*,
- COMBO-80G: *2x40Gb transceiver*,
- COMBO-100G: *4x25Gb transceiver*[1].

Dle požadované propustnosti a rychlosti zpracování jsou FPGA karty osazeny adekvátními FPGA od společnosti Xilinx – čipem FPGA Virtex-5 (u COMBO-4G) až čipem FPGA Virtex-7 u plánované COMBO-100G.

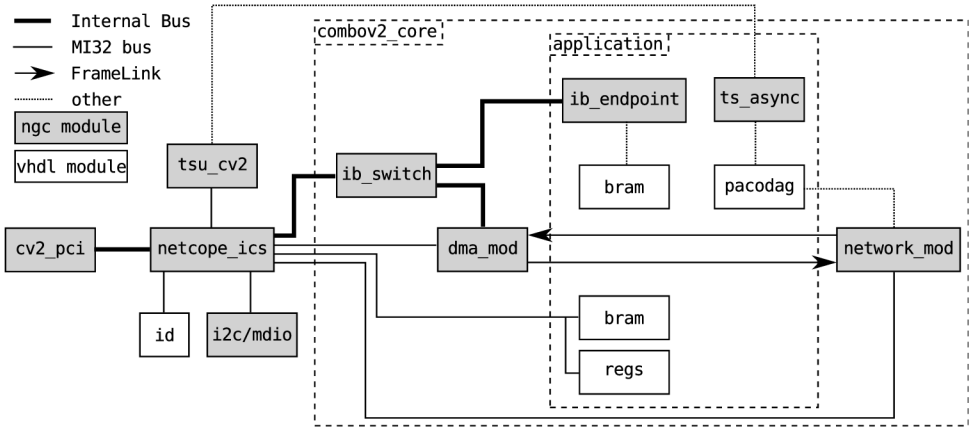
V rámci této bakalářské práce je provedena anotace hardwaru síťové karty postavené na platformě NetCOPE – NIC NetCOPE pro použití v systému EDK. Tato síťová karta je postavena nad COMBOv2 kartou se dvěma 10G transceivery (nese označení `COMBOv2_10G2`).

4.4 Moduly NIC NetCOPE

Síťová karta NIC NetCOPE je vnitřně složena z několika modulů [22]. Informace o těchto modulech a jejich funkci obsahuje následující část.

4.4.1 COMBOv2 Core

Modul COMBOv2 Core obsahuje typickou strukturu síťové aplikace, což je možné vidět na obrázku 4.5.



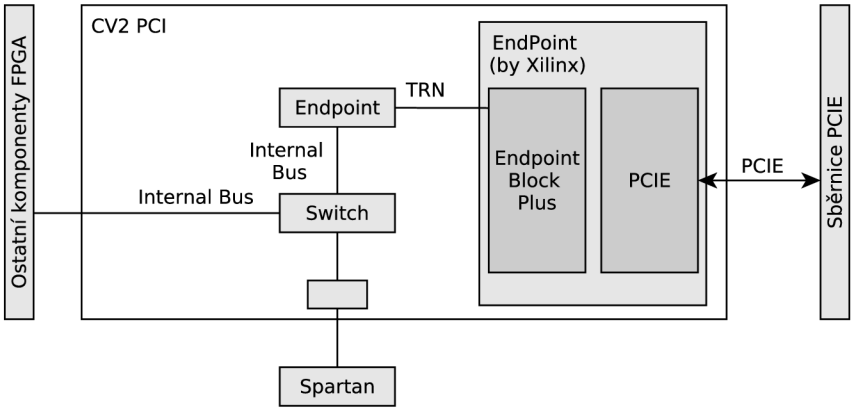
Obrázek 4.5: Ukázková vnitřní struktura síťové aplikace NIC NetCOPE [22]

Část, označená na obrázku 4.5 jako `application`, je určena pro uživatelskou aplikaci. Z této části je vyveden port sběrnice MI32. Ten umožňuje připojení různých uživatelských modulů, a tak je zajištěna rozšiřitelnost platformy. [22] Vlastní moduly mohou být rovněž připojeny na sběrnici FrameLink mezi `DMA module` a `network module` [22].

[1] Je možné též použít konfiguraci 10x10Gb

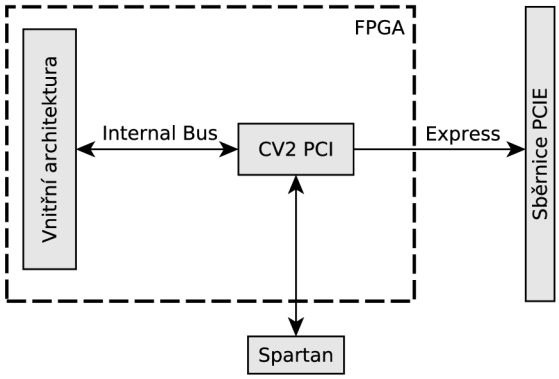
4.4.2 Modul cv2_pci

Modul cv2_pci slouží pro připojení uživatelské aplikace vytvořené pomocí platformy NetCOPE ke sběrnici PCI Express. Vnitřní struktura modulu cv2_pci je znázorněna na obrázku 4.6.



Obrázek 4.6: Vnitřní struktura modulu CV2 PCI

Druhou funkcí modulu cv2_pci je nahrávání aplikace do čipu FPGA [22]. K tomuto účelu využívá modul cv2_pci samostatného FPGA typu Spartan, jehož jedinou úlohou je řízení této činnosti. Schéma zapojení FPGA čipu typu Spartan je viditelné na obrázcích 4.6 a 4.7.

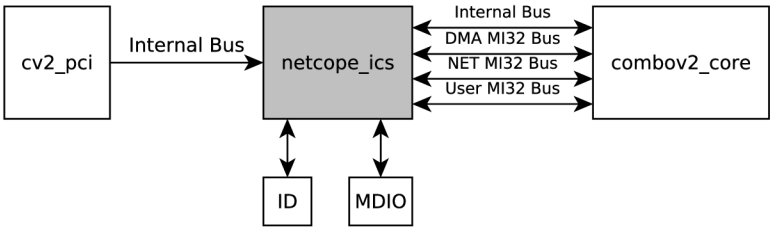


Obrázek 4.7: Umístění komponenty CV2 PCI v hierarchii FPGA s NetCOPE

4.4.3 Modul netcope_ics

Netcope_ics slouží k propojení jednotlivých částí platformy NetCOPE. Je vytvořen jako základní sběrnicový systém obsahující navíc několik servisních výstupů [22]. Tyto servisní výstupy směřují do modulů ID (a MDIO), jak ukazuje obrázek 4.8. MDIO slouží pro řízení fyzické vrstvy ethernetu a zjišťování jejího stavu [15].

Běžné (neservisní) výstupy z komponenty *netcope_ics* jsou realizovány sběrnicemi *Internal Bus* a *Local Bus*.



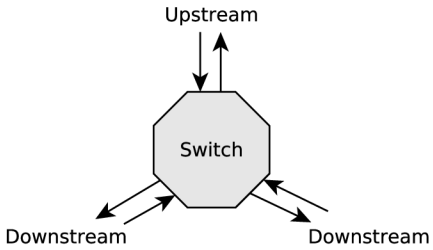
Obrázek 4.8: Napojení komponenty *netcope_ics* na ostatní komponenty

Komponenta netcope_ics je navržena tak, aby byla napojena přímo na modul *cv2_pci* na jedné straně a na modul *combov2_core* obsahující uživatelskou aplikaci na straně druhé. S modulem *cv2_pci* komunikuje přes sběrnicí InternalBus, komunikace s uživatelskou částí aplikace je realizována sběrnicemi InternalBus a třemi sběrnicemi MI32 [22]. Pro rychlé přenosy dat mezi NetCOPE aplikací a DMA je určena sběrnice InternalBus.

4.4.4 Modul gics_ib_switch_slave_synth

Přepínací (*switch*) komponenta slouží k přeposílání příchozí komunikace po sběrnci InternalBus do výstupních portů. Podle specifikace [23] existují dva typy InternalBus switchů, a to *master* a *slave*.

InternalBus Switch typu master dokáže přeposílat příchozí komunikaci přímo danému cílovému portu, jelikož dokáže zjistit cílovou adresu. Pro použití v hardwaru ale není nejvhodnější, jelikož spotřebovává více zdrojů. Schéma InternalBus Switche je znázorněno na obrázku 4.9.

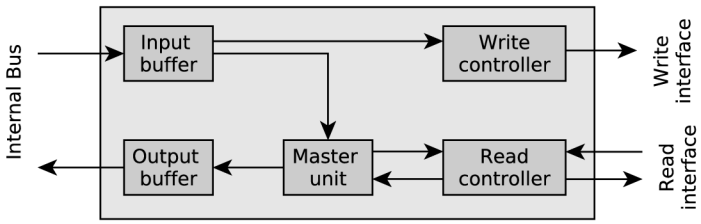


Obrázek 4.9: InternalBus Switch

Naopak InternalBus Switch Slave nedokáže z paketu zjistit cílovou adresu. Princip jeho činnosti je takový, že vše ze směru `upstream` broadcastuje do směru `downstream` a naopak. O konečném zpracování paketů rozhodne koncový dekodér. Takto zjednodušený InternalBus Switch spotřebovává méně zdrojů, proto je použit v modulu *gics_ib_switch_slave_synth*.

4.4.5 Modul gics_ib_endpoint_synth

Modul InternalBus Endpoint převádí požadavky přicházející po sběrnici InternalBus do rozhraní obecného adresového dekodéru. Operace na sběrnici mohou být provedeny jednak mezi FPGA komponentou a pamětí v uživatelském PC, ale také mezi dvěma FPGA komponentami. Vnitřní struktura modulu InternalBus Endpoint je vidět na obrázku 4.10.

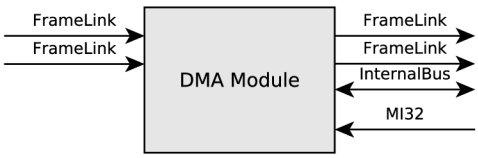


Obrázek 4.10: InternalBus Endpoint [22]

4.4.6 Varianty DMA modulů v NIC NetCOPE

DMA module v NetCOPE obstarává DMA přenosy mezi vstupně/výstupními buffery na kartě a pamětí RAM. Ke komunikaci se využívá linek typu FrameLink (ty umožňují rychlou komunikaci s vysokou propustností). Karta NIC NetCOPE může obsahovat několik variant DMA modulů, dle požadovaných vlastností:

- Modul dma_mod_2x64b_rxtx_gen – úkolem modulu je vysokorychlostní komunikace s pamětí RAM v počítači. Vstupním rozhraním modulu jsou dva 64 bitové FrameLinky určené právě pro vysokorychlostní komunikaci. Struktura modulu je viditelná na obrázku 4.11 [22].



Obrázek 4.11: 64 bitový DMA modul obsažený v platformě NetCOPE

Paralelně se vstupním rozhraním jsou z výstupního rozhraní vyvedeny dvě vysokorychlostní sběrnice typu FrameLink, obě 64 bitové. Navíc je na rozhraní modulu přivedena sběrnice InternalBus sloužící k obousměrnému přenosu dat mezi hostem a DMA modulem v platformě NetCOPE. Poslední sběrnicí přivedenou na rozhraní modulu je sběrnice MI32.

Pro zaslání dat do uživatelské paměti PC potřebuje uživatel aplikace v platformě NetCOPE pouze zaslat data na RX FrameLink rozhraní DMA modulu. DMA modul již sám automaticky provede zápis těchto dat do RAM paměti [22].

Pro vyčtení dat z RAM paměti a jejich zápis na COMBOv2 kartu s platformou NetCOPE musí uživatel softwarově připravit požadovaná data a iniciovat práci DMA modulu. DMA modul potom tato data opět automaticky vyčte z paměti RAM a přes TX FrameLink rozhraní je pošle do karty. Na každém komunikačním rozhraní je implementován buffer o velikosti 4 kB [22].

- Modul dma_mod_2x64b_rxtx_16kbyte_gen – jediným rozdílem od jednotky `dma_mod_2x64b_rxtx_gen` je přítomnost 16 kB bufferů na každém komunikačním kanálu. Velikost bufferů potom ovlivňuje maximální velikost přenášeného rámce, proto je modul `dma_mod_2x64b_rxtx_16kbyte_gen` vhodný v případech, kdy se pracuje s Ethernetovými Jumbo rámci [22].

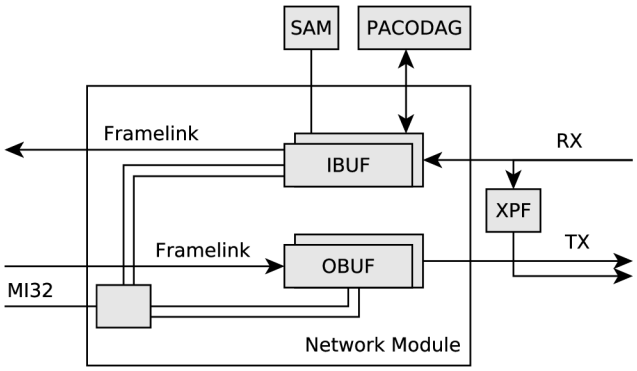
- Modul dma_mod_64b_16rx2tx_gen – tento modul je použit v aplikacích, kde se data z FPGA posílají do 16 oddělených DMA bufferů vyhrazených v paměti RAM. Také najde využití v případě osazení více jádrových procesorů.

  Ve srovnání s modulem `dma_mod_2x64b_rxtx_16kbyte_gen` obsahuje pouze jedno RX FrameLink rozhraní. Rozhraní TX je nezměněno. Na každém RX a TX kanálu jsou implementovány jen 4 kB buffery [22].

4.4.7 Varianty Network modulů v NIC NetCOPE

Obecný popis network modulu byl uveden v části 4.1.2. V síťové kartě existuje několik variant síťových modulů (network module):

- Network_mod_10g2_64 – Vnitřně je složen ze vstupních bufferů (IBUFs), jejichž úkolem je příjem rámců ze síťového rozhraní, převod do rámců sběrnice FrameLink a jejich zaslání uživatelské aplikaci NetCOPE.



Obrázek 4.12: Vnitřní struktura síťového modulu platformy NetCOPE

Obrázek 4.12 ukazuje, že síťový modul obsahuje také výstupní buffery (OBUFs). Ty jsou vyhrazeny pro případ, kdy uživatelská aplikace odesílá data do sítě. Při této činnosti jsou rámce sběrnice FrameLink převedeny na rámce síťového rozhraní a jsou pomocí tohoto rozhraní odeslány do sítě.

Na obrázku 4.12 je také možné vidět samplovací jednotku (*SAM*) a jednotku PACODAG. Účelem samplovací jednotky je řízené zahazování příchozích rámců. Jednotka

PACODAG (*PAcket COntrol DAta Generator*) potom přidává kontrolní data ke každému přenášenému paketu [25]. Poslední jednotkou, která dosud nebyla popsána je jednotka XPF. Jejím úkolem je přeposílání paketů ze vstupu přímo na výstup, ale v reálném aplikaci se většinou nepoužívá.

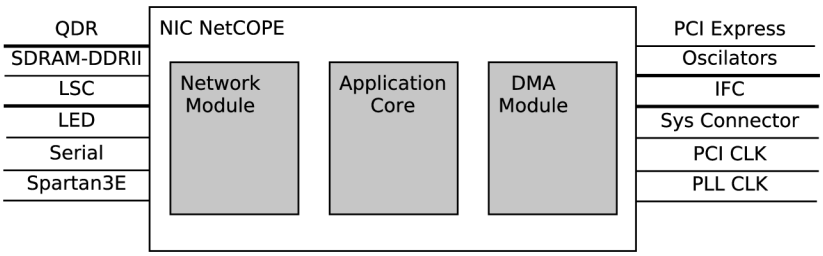
- Network_mod_10g2_64_16kbyte – tento modul má na každém komunikačním kanále buffer o velikosti 16 kB (na rozdíl od 4 kB bufferů v případě `network_mod_10g2_64`). Velikost těchto bufferů ovlivňuje maximální možnou délku rámce (jako u DMA modulů). Opět je v kombinaci s tímto síťovým modulem možné použít Ethernetové Jumbo rámce [22].

4.4.8 Modul tsu_cv2

Modul `tsu_cv2` [22] (*TimeStampUnit* on COMBOv2) obstarává generování přesných časových značek. Pokud je v systému dostupná karta ComboL-GPS, je tato jednotka schopná generovat své časové značky pomocí GPS signálu. Pokud karta ComboL-GPS není dostupná, modul `tsu_cv2` generuje vlastní, méně přesné, časové značky.

4.5 Popis NetCOPE z pohledu IP-core

IP-core platformy NetCOPE je vnitřně složen k částí popsaných v části 4.4. Propojení těchto modulů je řešeno opět na vnitřní úrovni IP-core, tedy uživatele integrujícího IP-core NetCOPE nezajímá. Rozhraní IP-core NetCOPE je zobrazeno na obrázku 4.13.



Obrázek 4.13: Rozhraní IP-core platformy NetCOPE

Tabulka 4.2 popisuje funkci jednotlivých signálů rozhraní.

Signál	**Popis funkce**	**Signál**	**Popis funkce**
QDR	pro připojení k paměti QDR	PCI Express	pro připojení rozhraní PCI Express
SDRAM-DDRII	pro připojení ke konektoru paměti DDRII	Oscilators	pomocné hodiny
LSC (Low Speed Connector)	pro připojení pomalých periferií např. COMBOL-GPS	IFC *(Interface Connector)*	pro připojení druhé karty (např. karty se síťovým rozhraním); součástí je XAUI
LED	pro ovládání LED	Sys Connector	pomocné signály
Serial	konektor sériového rozhraní (RS-232)	PCI CLK	hodinový signál pro ovládání sběrnice PCI
Sparta3E	pro připojení FPGA Spartan (určeno ke konfiguraci a bootování hlavního FPGA Virtex-5)	PLL CLK	konfigurovatelné hodiny

Tabulka 4.2: Rozhraní NetCOPE IP-core

Kapitola 5

Návrh a implementace

Následující kapitola přináší informace o návrhu a řešení převodu platformy NetCOPE do EDK. Nastiňuje kroky převodu, od převodu NetCOPE jako celku po převod jednotlivých částí NetCOPE a napojení mikroprocesoru MicroBlaze.

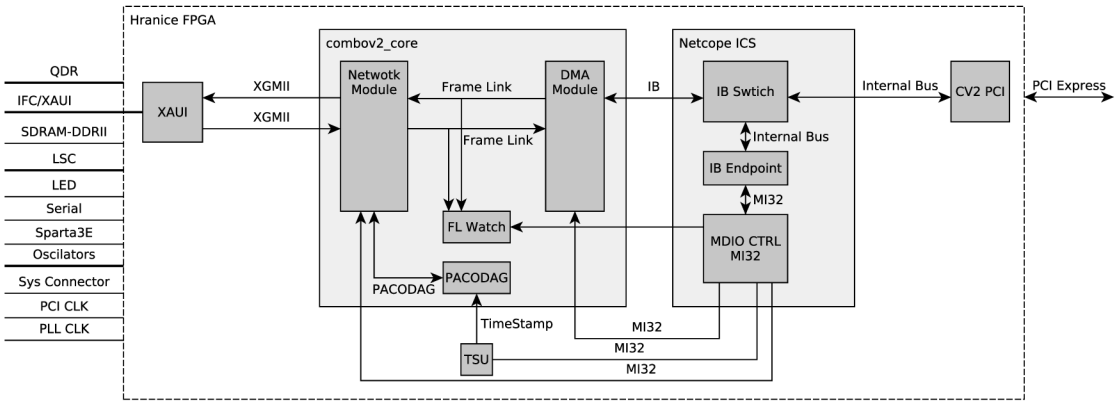
5.1 Návrh

Jako výchozí projekt, na němž je proveden převod NetCOPE do EDK byl zvolen projekt síťové karty NIC, která vznikla v rámci skupiny TMC sdružení CESNET z. s. p. o. Celý převod by měl být zakončen napojením mikroprocesoru MicroBlaze (viz 3.3.2). Pomocí mikroprocesoru MicroBlaze je možné konfigurovat součásti NIC NetCOPE, jako je DMA modul, síťový modul, atd. Jednotlivé kroky převodu jsou následující:

- *Převod NIC NetCOPE IP-core jako celku* – prvním krokem pro převod platformy NetCOPE do EDK je anotace NIC NetCOPE jako celku. Rozhraní NIC NetCOPE je znázorněno na obrázku 4.13. Během této fáze převodu dojde k nastudování PSF a používání EDK.

- *Anotace jednotlivých částí NIC NetCOPE* – druhým krokem v převodu platformy NetCOPE do EDK je anotace jednotlivých součástí síťové karty. Tyto součásti jsou znázorněny na obrázku 5.1. Cílem této fáze převodu, která již překračuje rámec zadání této bakalářské práce, je plně propojený design síťové karty NIC NetCOPE vytvořený v prostředí EDK. Takto vytvořený design umožňuje uživatelskou změnu jednotlivých komponent v EDK. Komponenty anotované v tomto kroku jsou na obrázku 5.1 znázorněny odstíny šedé. Na obrázku 5.1 je též znázorněno rozhraní NIC NetCOPE (podobně jako je popsáno v části 4.5). Jelikož ale většína signálů rozhraní není v reálném designu NIC NetCOPE zapojena, jsou na obrázku 5.1 přivedeny jen na rozhraní FPGA.

- *Integrace platformy NetCOPE s procesorem MicroBlaze* – Cílem této fáze převodu NetCOPE do EDK je zapojení mikroprocesoru MicroBlaze (viz 3.3.2) do designu platformy NetCOPE. Z předchozího kroku jsou dostupné anotace jednotlivých částí platformy NetCOPE, ke kterým bude mikroprocesor připojen. Orientační schéma připojení mikroprocesoru MicroBlaze do platformy NetCOPE je znázorněno na obrázku 5.2. Bloky označené v obrázku 5.2 odstíny šedé jsou do designu přidávány v tomto kroku.

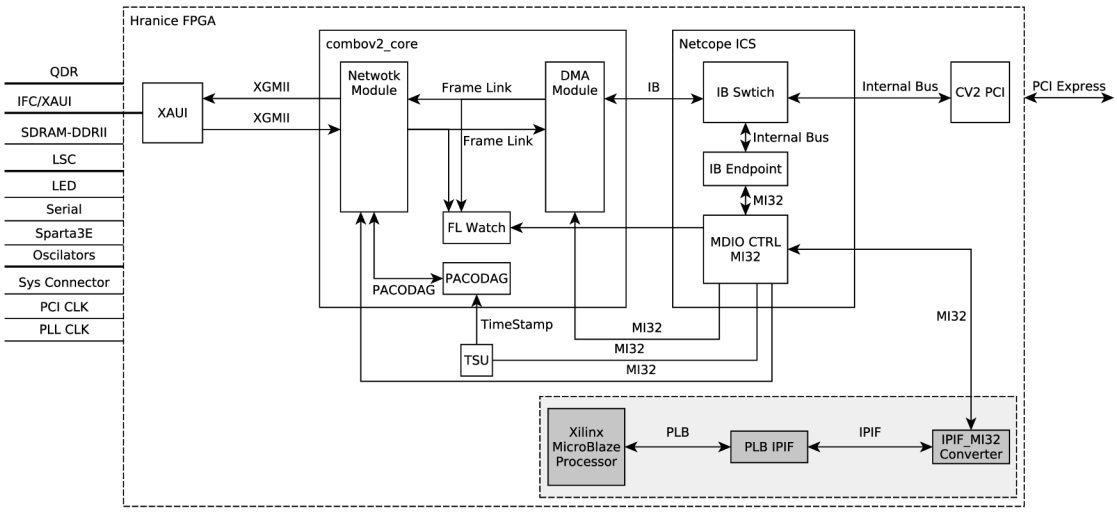


Obrázek 5.1: Vnitřní uspořádání NIC NetCOPE

Pro připojení mikroprocesoru ke komponentám platformy NetCOPE bude nutné transformovat sběrnici PLB na sběrnici IPIF. Tento převod zajišťuje komponenta PLB IPIF. Pro převod IPIF na MI32 slouží komponenta IPIF_MI32 Converter.

Dále bude třeba vyřešit, jaká úroveň mikroprocesoru bude integrována. Zda bude integrován pomalý mikroprocesor s malou spotřebou zdrojů FPGA nebo bude implementován rychlejší mikroprocesor, jehož spotřeba zdrojů je ale také větší.

Dalším úkolem, který bude nutné vyřešit je optimalizace software pro řízení mikroprocesoru, aby bylo možné tento software nahrát do vyhrazené paměti uvnitř FPGA, tato paměť je velikostě omezená.



Obrázek 5.2: Vnitřní uspořádání NIC NetCOPE s mikroprocesorem MicroBlaze

5.2 Implementace

V rámci této bakalářské práce byl implementován první krok převodu platformy NetCOPE do EDK. Jako základní projekt pro převod platformy NetCOPE do EDK byl zvolen projekt síťové karty postavené na platformě NetCOPE (*NIC NetCOPE*). Volba na NIC NetCOPE padla zejména proto, že se jedná o konkrétní funkční použití NetCOPE, ne jen o čistý NetCOPE. Dalším aspektem, který rozhodl právě o tomto projektu je potenciální integrace s mikroprocesorem MicroBlaze a množství operací, které by mohl MicroBlaze nastavovat.

Již v první fázi, kdy se specifikovaly jednotlivé kroky překladu platformy NetCOPE do EDK, bylo rozhodnuto o vnitřní reprezentaci zdrojových kódů platformy. Jsou na výběr dvě možnosti, a to reprezentace formou zdrojových kódů VHDL nebo reprezentace na úrovni souborů NGC. Soubory NGC vnikají překladem HDL zdrojových kódů v XST syntéze (viz 3.1.1), neobsahují zdrojový kód. To je vhodnější zejména v případech, kdy má být výsledné řešení (IP-core anotované pomocí PSF) poskytováno třetím stranám, kterým má být skryto vnitřní řešení.

5.2.1 Postup práce

Postup práce při převodu NIC NetCOPE pro použití v EDK je následující:

- Anotace rozhraní NetCOPE pomocí souboru MPD – po získání zdrojových souborů celku platformy NetCOPE ve formátu NGC je nutné anotovat hardware pomocí PSF (viz 3.3.1).

  Nejprve je nutné anotovat rozhraní síťové karty, které je popsáno v části 4.5 na obrázku 4.13 a podle informací uvedených v souboru `cards/combov2/chips/fpga_u0.vhd` v repozitáři NIC projektu Liberouter. Rozhraní anotovaného IP-core se zapisuje do souboru MPD, který má strukturu popsanou v části 3.4.3. Soubor MPD také anotuje, zda se mají při syntéze na vstupně-výstupních portech vytvořit třístavové buffery. Tuto informaci určuje parametr `THREE_STATE`. V případě NIC NetCOPE jsou již třístavové buffery implementovány ve zdrojovém kódů. V souboru MPD tak bylo nutné nastavit u vstupně výstupních signálů parametr `THREE_STATE` na hodnotu FALSE.

- Anotace použitých součástí IP-core v souboru BBD – dále je nutné anotovat, které součásti platformy NetCOPE budou do převedeného IP-core zahrnuty. Tedy, které součásti ve formátu NGC jsou pro překlad NIC NetCOPE využívány (seznam modulů viz část 4.4). Tato informace je obsažena v souboru BBD, jehož obsah byl upřesněn v části 3.4.3.

5.2.2 Problémy při převodu platformy NetCOPE do EDK

Během převodu platformy NetCOPE do EDK se objevila řada problémů spojených jednak s nedostatečnou dokumentací prostředí EDK a formátu PSF, ale také s nekompatibilitou překladového systému CESNET, z. s. p. o. a překladového systému, který se využívá při překladu designu v prostředí EDK.

- Prvním problémem spojeným s převodem NetCOPE do EDK je nedostatečná dokumentace použití formátu PSF a práce s prostředím EDK. Tento problém se objevil například v době prvních pokusů o přeložení platformy NetCOPE z EDK. Překladový systém EDK nerozpoznal `top-level` entitu designu a proto nevybral vhodnou komponentu ze seznamu v souboru BBD a nepřeložil tak design. Řešením tohoto problému

je vytvoření VHDL obálky (viz 3.4.3) a její propojení s periferií pomocí souboru PAO (viz 3.4.3). Tato informace však není uvedena ani v referenčním manuálu (viz [27, str. 10]).

- Druhým problémem převodu NetCOPE do EDK je nekompatibilita překladových systémů používaných sdružením CESNET a nástrojem EDK. Tato nekompatibilita vyústila v problém s nástrojem MAP, kdy nebyly korektně zpracovány některé informace ze souboru omezení (UCF) a nebyly správně napojeny vstupně-výstupní piny. EDK interně využívá pojmenování pinů ve formátu `peripheral_PORTNAME`, naopak překladový systém sdružení CESNET vynechává část `peripheral_`. V EDK lze do formátu, kterým pojmenovává CESNET přejmenovat pouze vstupní a výstupní piny. Vstupně-výstupní piny takto pojmenovat nelze.

  Řešením druhého problému bylo ponechání pojmenování vstupně výstupních pinů v EDK ve formátu `peripheral_PORTNAME`. Touto úpravou ale došlo k nekonzistenci mezi souborem MPD a souborem UCF, proto bylo nutné zasáhnout i do souboru UCF, kde musely být vstupně-výstupní porty přejmenovány stejným způsobem. Toto řešení bylo zvoleno z důvodu jeho nenáročnosti – vstupně-výstupních pinů je v souboru UCF mnohem méně, než pinů ostatních.

5.2.3 Testovací design

Testovací design vzniklý překladem pomocí EDK byl porovnán z hlediska spotřeby zdrojů s designem získaným z překladového systému sdružení CESNET. Tabulka 5.1 přináší srovnání těchto designů.

Zdroj	**Překlad systémem EDK**	**Překlad systémem CESNET**
Slice registry	24 005 z 97 280 (24%)	24 009 z 97 280 (24%)
Slice LUT	25 858 z 97 280 (26%)	25 880 z 97 280 (26%)
LUT Flip Flop páry	34 422	34 401
BlockRAM/FIFO	56 z 212 (26%)	57 z 212 (26%)
BUFDS	3 z 8 (37%)	3 z 8 (37%)
CRC64	4 z 16 (25%)	4 z 16 (25%)
PCIe	1 z 1 (100%)	1 z 1 (100%)
SYSMON	1 z 1 (100%)	1 z 1 (100%)

Tabulka 5.1: Srovnání spotřeby zdrojů při překladu NIC NetCOPE překladem v EDK a pomocí překladového systému CESNET

Srovnáme-li oba designy, zjistíme, že design vzniklý překladem pomocí překladového systému sdružení CESNET spotřebovává přibližně stejné množství zdrojů. Naopak s ohledem na frekvenci je na tom tento design mnohem lépe, dosáhl maximální frekvence 307,962 MHz. Design vzniklý překladem v EDK dosáhl maximální frekvence 222,519 MHz.

Kapitola 6

Závěr

Cílem této bakalářské práce byl převod platformy NetCOPE do EDK (*Xilinx Embedded Development Kit*). V úvodní části byla představena technologie čipů FPGA, společně s vývojovými technologiemi pro čipy FPGA. Dále byla představena stavební jednotka hardware v prostředí EDK – IP-core. Společně s popisem IP-core byl představen anotační formát PSF (*Platform Specific Format*) sloužící k anotaci hardware pro použití v prostředí EDK.

Dále byl v této práci představen framework NetCOPE. Platforma byla v úvodu popsána jednak obecně, byl však kladen důraz také na popis typických oblastí jejího použití včetně specifikace komunikačních sběrnic použitých v NetCOPE. Nakonec byly popsány jednotlivé fyzické vnitřní moduly této platformy.

Platforma NetCOPE využívá ke svému běhu mj. i COMBOv2 karty. Těm byla věnována část kapitoly 4. COMBOv2 karta zde byla obecně popsána, ale byl zde uveden i popis jednotlivých logických částí platformy NetCOPE (DMA modulu, síťového modulu a propojovacího systému).

Stěžejní částí této bakalářské práce je kapitola 5, v níž byly shrnuty informace o implementaci součástí PSF pro použití platformy NetCOPE v systému EDK. V této kapitole byl rovněž popsán postup použití již vytvořené anotace hardware v prostředí EDK.

Podařilo se nastudovat systém anotace hardware pomocí formátu PSF a jeho použití v prostředí EDK. Je vytvořena sada skriptů anotující platformu NetCOPE pro použití v tomto prostředí. Správnost anotačních skriptů se podařilo ověřit přidáním NIC NetCOPE do EDK. Vzniklý design odpovídá množstvím spotřebovaných zdrojů designu vytvořenému pomocí překladového systému sdružení CESNET, z. s. p. o. Frekvence designu vzniklého překladem v EDK je naopak nižší – 222,519 MHz oproti 307,962 MHz. Důvodem této nižší frekvence mohou být různá nastavení překladových systémů.

Pro další budoucí vývoj aplikací s NetCOPE v EDK je nutné provést převod jednotlivých modulů NetCOPE do EDK, ten již probíhá a je zčásti hotov. Potom bude možné vložit do takto převedené platformy mikroprocesor MicroBlaze. Takto upravenou platformu bude možné řídit pomocí tohoto mikroprocesoru, nebude tak vyžadovat aktivní řízení přes systémovou sběrnici PCI Express.

Literatura

[1] XILINX. *AXI IIC Bus Interface v1.02a* [online]. říjen 2012 [cit. 2014-05-06]. Dostupné z: http://www.xilinx.com/support/documentation/ip_documentation/axi_iic/v1_02_a/pg090-axi-iic.pdf.

[2] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL*. 1. vyd. Praha: BEN - technická literatura, 2006, 349 s. ISBN 80-730-0198-5.

[3] XILINX. *Development System Reference Guide v10.1* [online]. 2008 [cit. 2014-02-07]. Dostupné z: http://www.xilinx.com/itp/xilinx10/books/docs/dev/dev.pdf.

[4] XILINX. *EDK Concepts, Tools, and Techniques. A Hands-On Guide to Effective Embedded System Design* [online]. duben 2011 [cit. 2014-02-12]. Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/edk_ctt.pdf.

[5] VIKTORIN, Jan. *EDK, MicroBlaze, Linux*. 2011.

[6] PUŽMANOVÁ, Rita. *Ethernet s kapacitou 40 a 100 Gbit/s* [online]. září 2010 [cit. 2014-03-10]. Dostupné z: http://archiv.cesnet.cz/sdruzeni/napsali-o-nas/2010/09/201009_ETM.html.

[7] KOŠEK, Martin a Jan KOŘENEK. FlowContext: Flexible Platform for Multigigabit Stateful Packet Processing. In: BERTELS, Koen. *Proceedings: 2007 International Conference on Field-Programmable Logic and Applications (FPL) : Amsterdam, The Netherlands, August 27-29, 2007*. Los Alamitos, US: IEEE Computer Society, 2007, s. 804-807. ISBN 1424410592.

[8] INVEA-TECH. *FPGA karty* [online]. 2013 [cit. 2014-03-04]. Dostupné z: https://www.invea.com/cs/produkty-sluzby/fpga-karty.

[9] ŠŤASTNÝ, Jakub. *FPGA prakticky*. Plzeň: BEN – technická literatura, 2011. ISBN 978-80-7300-261-9.

[10] MATOUŠEK, Jiří. FPGA-Based Packet Generator. In: *STUDENT EEICT 2011: Proceedings of the 17th conference*. vyd. 1. Brno: Brno University of Technology, 2011, s. 312-314. ISBN 978-80-214-4272-6. Dostupné z: http://www.feec.vutbr.cz/EEICT/2011/sbornik/02-Magisterske%20projekty/10-Pocitacove%20systemy/07-xmatou06.pdf.

[11] LIBEROUTER. *FrameLink* [online z neveřejného repozitáře]. 2011 [cit. 2014-03-27].

[12] KORČEK, Pavol, Vlastimil KOŠÁŘ, Martin ŽÁDNÍK, Karel KORANDA a Petr KAŠTOVSKÝ. Hacking NetCOPE to run on NetFPGA-10G. In: *Proceedings of the*

2011 ACMIEEE Seventh Symposium on Architectures for Networking and Communications Systems. Washington, DC: IEEE Computer Society, 2011, s. 1-2. ISBN 9780769545219. Dostupné z: http://www.fit.vutbr.cz/research/view_pub.php.cz?id=9686.

[13] CELOXICA. *Handel-C Language Reference Manual* [online]. 2005 [cit. 2014-02-04]. Dostupné z: http://babbage.cs.qc.edu/courses/cs345/Manuals/HandelC.pdf.

[14] *IEEE 1666 Standard SystemC Language Reference Manual* [online]. 2005 [cit. 2014-02-06]. Dostupné z: http://standards.ieee.org/getieee/1666/.

[15] TURNER, Ed a Law DAVID. *IEEE P802.3ae MDC/MDIO* [online]. září 2001 [cit. 2014-04-26]. Dostupné z: http://www.ieee802.org/3/efm/public/sep01/turner_1_0901.pdf.

[16] ROUSE, Margaret. IP core (intellectual property core). *WhatIs.com* [online]. 2011 [cit. 2014-03-18]. Dostupné z: http://whatis.techtarget.com/definition/IP-core-intellectual-property-core.

[17] XILINX. *LocalLink Interface Specification* [online]. červenec 2005 [cit. 2014-04-25]. Dostupné z:http://www.xilinx.com/aurora/aurora_member/sp006.pdf.

[18] XILINX. *LogiCORE IP Endpoint Block Plus v1.14 for PCI Express* [online]. duben 2010 [cit. 2014-04-04]. Dostupné z: http://www.xilinx.com/support/documentation/ip_documentation/pcie_blk_plus_ds551.pdf.

[19] XILINX. *LogiCORE IP MicroBlaze Micro Controller System (v1.1)* [online]. duben 2012 [cit. 214-05-02]. Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ds865_microblaze_mcs.pdf.

[20] XILINX. *LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a)* [online]. září 2010 [cit. 2014-05-06]. Dostupné z: http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf.

[21] MicroBlaze Soft Processor Core. XILINX. *Xilinx.com* [online]. 2013 [cit. 2014-05-02]. Dostupné z: http://www.xilinx.com/tools/microblaze.htm.

[22] LIBEROUTER. *NetCOPE Firmware Guide 10G2* [online z neveřejného repozitáře]. 2011 [cit. 2014-03-12].

[23] LIBEROUTER. *NetCOPE ICS* [online z neveřejného repozitáře]. 2011 [cit. 2014-03-26].

[24] MARTÍNEK, Tomáš. *NetCOPE Platform* [online]. 2009 [cit. 2014-03-04]. Dostupné z: http://merlin.fit.vutbr.cz/ant/technology/netcope_platform.html.

[25] LIBEROUTER. *PACODAG* [online z neveřejného repozitáře]. 2011 [cit. 2014-03-27].

[26] XILINX. *Physical Design Rule Check* [online]. 2005 [cit. 2014-02-24]. Dostupné z: http://www.cs.indiana.edu/hmg/le/project-home/xilinx/ise_8.1/doc/usenglish/de/dev/phys_drc.pdf.

[27] XILINX. *Platform Specification Format Reference Manual Embedded Development Kit (EDK) 14.1* [online]. duben 2012 [cit. 2014-02-12]. Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_2/psf_rm.pdf.

[28] XILINX. *PROMGen* [online]. 2009 [cit. 2014-04-25]. Dostupné z: http://www.cs.indiana.edu/hmg/le/project-home/xilinx/ise_8.1/doc/usenglish/de/dev/promgen.pdf.

[29] KRÁL, Jiří. *Řešené příklady ve VHDL: hradlová pole FPGA pro začátečníky*. 1. vyd. Praha: BEN - technická literatura, 2010, 127 s. ISBN 978-80-7300-257-2.

[30] *SystemC* [online]. 2014 [cit. 2014-02-12]. Dostupné z: http://en.wikipedia.org/wiki/SystemC.

[31] OPEN SYSTEMC INITIATIVE. *SystemC 2.0.1 Language Reference Manual, Revision 1.0* [online]. 2003 [cit. 2014-02-07]. Dostupné z: http://www.es.ele.tue.nl/~mininoc/doc/SystemC_2_0_LRM_v1_0.pdf.

[32] LYSECKY, Susan. *Verilog By Example* [online]. 2006 [cit. 2014-02-05].

[33] *Verilog-XL Reference Manual 1.6c* [online]. Červen 1993 [cit. 2014-02-06]. Dostupné z: http://www.eecis.udel.edu/~elias/verilog/verilog_manuals/.

[34] SYNARIO DESIGN AUTOMATION. *VHDL Reference Manual* [online]. Březen 1997 [cit. 2014-02-07]. Dostupné z: http://www.ics.uci.edu/~jmoorkan/vhdlref/Synario%20VHDL%20Manual.pdf.

[35] XILINX. *Virtex-6 FPGA Configuration* [online]. listopad 2013 [cit. 2014-04-25]. Dostupné z: http://www.xilinx.com/support/documentation/user_guides/ug360.pdf.

[36] XILINX. *XST Synthesis Overview* [online]. 2009 [cit. 2014-01-04]. Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_using_xst_for_synthesis.htm.

[37] XILINX. *XST User Guide v10.1* [online]. 2008 [cit. 2014-02-07]. Dostupné z: http://www.xilinx.com/itp/xilinx10/books/docs/xst/xst.pdf.

[38] CESNET. *Záměr 2007: Programovatelný hardware* [online]. 2006 [cit. 2014-03-04]. Dostupné z: http://archiv.cesnet.cz/doc/2007/zprava/proghw.html#nc_layers.

Příloha A

Obsah DVD a instrukce pro použití projektu v Xilinx Platform Studio

Obsah složek na DVD je následující:

- `BP_pdf` – Složka obsahuje text práce ve formátu PDF
- `BP_tex` – Složka obsahuje text práce ve formátu TEX
- `EDK_mhs` – Složka obsahuje soubor MHS pro rychlé vložení designu do projektu
- `EDK_pcore` – Složka obsahuje NetCOPE pro použití v EDK
- `EDK_projekt` – Složka obsahuje projekt z programu Xilinx Platform Studio s anotovaným NetCOPE
- `EDK_ucf` – Složka obsahuje soubor omezení designu pro použití v Xilinx Platform Studio

Postup pro použití projektu v Xilinx Platform Studio

1. Spuštění programu Xilinx Platform Studio příkazem `xps` z terminálu
2. Volba možnosti *Open Project* v hlavním okně programu
3. Otevření souboru `system.xmp` ve složce `EDK_projekt` na DVD