



**ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE**  
**Fakulta elektrotechnická**  
**Katedra elektrických pohonů a trakce**

**Možnosti využití FPGA pro řízení pohonů**

**Usage of FPGA for controlling electric drives**

Diplomová práce

Studijní program: Elektrotechnika, Energetika a Management

Studijní obor: Elektrické pohony

Vedoucí práce: Ing. Jan Bauer, Ph.D.

**Petr Zakopal  
Praha 2023**



## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zakopal** Jméno: **Petr** Osobní číslo: **483802**  
Fakulta/ústav: **Fakulta elektrotechnická**  
Zadávající katedra/ústav: **Katedra elektrických pohonů a trakce**  
Studijní program: **Elektrotechnika, energetika a management**  
Specializace: **Aplikovaná elektrotechnika**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Oživení pracoviště s měničem DCM a PLC SIMATIC**

Název bakalářské práce anglicky:

**Workpalce with Rectifier DCM and PLC SIMATIC**

Pokyny pro vypracování:

- 1) Seznamte se s měničem řady DCM firmy SIEMENS
- 2) Oživte základní regulační smyčky měniče (otáčkovou, proudovou)
- 3) Prostudujte možnosti záznamu průběhů z měniče pomocí PLC nebo dotykového panelu
- 4) Pomocí PLC SIMATIC S1200 a dotykového panelu realizujte vzdálené ovládání a monitoring měniče
- 5) Na dotykovém panelu vytvořte obrazovku pro nastavování otáček nebo momentu motoru napájeného měničem

Seznam doporučené literatury:

- [1] Weidauer J., Messer R. Electrical Drives, Publics Erlangen, 2014
- [2] SCE Training Curriculum. Siemens AG, 2016
- [3] Durry B. The Control Techniques Drives and Controls Handbook 2nd ed., IET, 2009
- [4] Pavelka J., Kobrle P. Elektrické pohony a jejich řízení. 3. reprezované vydání. Praha: České vysoké učení technické v Praze, 2016. ISBN 978-80-01-06007-0.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Jan Bauer, Ph.D., katedra elektrických pohonů a trakce FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **24.01.2021**

Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Jan Bauer, Ph.D.  
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.  
podpis děkana(ky)

## III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta



## PROHLÁŠENÍ

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne \_\_\_\_\_

Petr Zákapal

## PODĚKOVÁNÍ

Nam quis commodo justo. Mauris diam metus, mattis sed rutrum in, volutpat sit amet sem. Nam bibendum commodo porttitor. Quisque eget lectus rutrum, molestie tortor id, iaculis nunc. Sed et maximus ipsum. Vivamus vel facilisis nisl. Curabitur eu nibh nec erat mollis finibus at in sapien. Mauris viverra sapien neque, nec lacinia odio laoreet eu. Quisque consectetur eros ac orci interdum scelerisque.

## **ABSTRAKT**

Nam quis commodo justo. Mauris diam metus, mattis sed rutrum in, volutpat sit amet sem. Nam bibendum commodo porttitor. Quisque eget lectus rutrum, molestie tortor id, iaculis nunc. Sed et maximus ipsum. Vivamus vel facilisis nisl. Curabitur eu nibh nec erat mollis finibus at in sapien. Mauris viverra sapien neque, nec lacinia odio laoreet eu. Quisque consectetur eros ac orci interdum scelerisque.

**Klíčová slova:** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis aliquam finibus sagittis. Nunc venenatis, augue quis luctus dictum, elit justo pharetra leo, nec viverra purus dui at quam.

## **ABSTRACT**

Nam quis commodo justo. Mauris diam metus, mattis sed rutrum in, volutpat sit amet sem. Nam bibendum commodo porttitor. Quisque eget lectus rutrum, molestie tortor id, iaculis nunc. Sed et maximus ipsum. Vivamus vel facilisis nisl. Curabitur eu nibh nec erat mollis finibus at in sapien. Mauris viverra sapien neque, nec lacinia odio laoreet eu. Quisque consectetur eros ac orci interdum scelerisque.

**Keywords:** Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis aliquam finibus sagittis. Nunc venenatis, augue quis luctus dictum, elit justo pharetra leo, nec viverra purus dui at quam.

## OBSAH

<b>1</b>	<b>Úvod.....</b>	<b>1</b>
<b>2</b>	<b>Embedded Systems .....</b>	<b>2</b>
2.1	Application Specific Integrated Circuit .....	3
2.2	Hardware Accelerated Applications .....	3
<b>3</b>	<b>Programovatelné hradlové pole – FPGA .....</b>	<b>5</b>
3.1	Vývoj FPGA z PLD .....	5
3.2	Aktuální složení FPGA .....	5
3.2.1	Generátory funkcí .....	6
3.2.2	Paměťové elementy .....	7
3.2.3	Logické buňky .....	7
3.2.4	Logické bloky .....	7
3.2.5	Propojení bloků.....	8
3.2.6	I/O bloky .....	8
3.2.7	Bloků speciálních funkcí .....	8
3.3	Programování .....	8
3.3.1	Forma tvorby algoritmu pro FPGA.....	8
3.3.2	Konverze HDL na konfigurační Bitstream.....	9
3.4	Spotřeba.....	10
3.5	Výpočetní výkon a propustnost.....	10
3.6	Využití .....	10
3.6.1	Aplikace v nepohonářských odvětví.....	10
3.6.2	Aplikace v elektrických pohonech.....	11
<b>4</b>	<b>Vývojová deska Digilent Zybo .....</b>	<b>12</b>
4.1	Základní přehled.....	12
4.1.1	CPU a FPGA čip .....	12
4.1.2	Uspořádání vývojové desky Zybo Zynq-7000 .....	15
4.2	Možné alternativy .....	18
<b>5</b>	<b>Model stroje .....</b>	<b>18</b>
5.1	Představení stroje.....	18
5.2	Matematický popis modelu stroje .....	18
5.3	Optimalizace modelu.....	19
<b>6</b>	<b>Aplikace pro FPGA a CPU .....</b>	<b>19</b>
6.1	Použité nástroje .....	19
6.1.1	Xilinx Vivado .....	19
6.1.2	Xilinx Vitis .....	19
6.1.3	PetaLinux .....	20
6.1.4	Programovací prostředí – operační systém Linux .....	21

6.2	Tvorba HW architektury Xilinx Vivado .....	21
6.2.1	Vivado Board Files .....	21
6.2.2	Tvorba HW architektury pro Digilent Zybo Zynq-7000 vývojovou desku .....	22
6.3	Tvorba PetaLinux .....	28
6.4	Tvorba SW pro CPU a FPGA .....	28
<b>7</b>	<b>Představení pracoviště .....</b>	<b>28</b>
<b>8</b>	<b>Dosažené výsledky .....</b>	<b>28</b>
	<b>Závěr .....</b>	<b>29</b>
	<b>Literatura .....</b>	<b>31</b>
<b>Příloha A</b>	<b>Seznam symbolů a zkratek .....</b>	<b>32</b>
A.1	Seznam symbolů .....	32
A.2	Seznam zkratek .....	32

## SEZNAM OBRÁZKŮ

2 - 1	Blokové schéma Embedded systému a řízeného fyzikálního systému. (převzato a upraveno z [2]) .....	3
3 - 1	Blokové schéma složení moderních FPGA.....	5
3 - 2	Základní koncept uspořádání FPGA. ....	6
3 - 3	Ukázka, jakým způsobem realizuje funkční generátor požadovanou funkci pomocí SRAM a MUX. ....	7
3 - 4	Blokové schéma převodu aplikace, naprogramované v procedurálním jazyce, na bitstream, který je vhodný pro konfiguraci FPGA. ....	9
4 - 1	Vývojová deska Digilent ZYBO Zynq-7000 ARM/FPGA SoC Trainer Board. ....	12
4 - 2	Detailní schéma čipu Zynq-7000, umístěného na vývojové desce <i>Digilent ZYBO Zynq-7000 ARM/FPGA SoC Trainer Board</i> . (převzato z [15]) .....	14
4 - 3	Vývojová deska Digilent ZYBO Zynq-7000 ARM/FPGA SoC Trainer Board vrchní pohled s vyznačením komponent. ....	15
4 - 4	Vývojová deska Digilent ZYBO Zynq-7000 ARM/FPGA SoC Trainer Board spodní pohled. ....	16
6 - 1	Blokový diagram tvorby spustitelné aplikace v prostředí Vitis. (převzato z [18], upraveno)	20
6 - 2	Xilinx Vivado – volba typu projektu. ....	22
6 - 3	Xilinx Vivado – výběr základního HW, pro který bude vytvářena architektura.....	23
6 - 4	Xilinx Vivado – vytváření Block Design. ....	23
6 - 5	Xilinx Vivado – vložení bloku ZYNQ7 Processing System.....	24
6 - 6	Xilinx Vivado – nastavení výstupních taktovacích signálů.....	25
6 - 7	Xilinx Vivado – propojení bloků taktování. ....	25
6 - 8	Xilinx Vivado – minimální funkční blokový design pro akcelerovanou aplikaci. ....	26
6 - 9	Xilinx Vivado – kritická upozornění vzniklá po validaci designu, která je možné ignorovat.	27
6 - 10	Xilinx Vivado – nastavení provádění úkonů syntézy, implementace a generování bitstreamu, volba použitych výpočetních jader a určení, kde se mají procesy vykonávat. ....	28

## **SEZNAM TABULEK**

3 - 1	Pravdivostní tabulka ukázkové funkce, realizované v generátoru funkcí, umístěném v logickém bloku FPGA.....	7
4 - 1	Popis označených komponent na vývojové desce Digilent Zybo Zynq-7000. (informace a značení z [16]) .....	17
5 - 1	Štítkové údaje stroje.....	18
5 - 2	Změřené parametry stroje.....	18
6 - 1	Ukázka nastavených AXI portů v Xilinx Vivado bloku <i>ZYNQ7 Processing System</i> .....	26





# 1 Úvod

V době, kdy byla od elektrických pohonů požadována spolehlivost, vysoká účinnost a nenáročné ovšem kvalitní řízení, byly k řízení využívány samotné digitální signálové procesory. Postupem času dochází ke zjištění, že výkon DSP není dostatečný a na některé aplikace, kde je vyžadováno provedení značného množství náročných výpočtů za co nejkratší čas, nejsou vhodné. Proto nastupuje éra logických programovatelných polí (FPGA), které jsou schopny tyto výpočty provést s velmi nízkými nároky na energii a za velmi krátký čas.

V mnoha odvětvích se již začíná využívat embedded systém s Application Specified Hardware, který je určen pouze na využití v předem dané aplikaci. Tento hardware slouží v dané aplikaci k jedinému účelu, který vykonává a na který je optimalizován. Tím se liší od procesoru, který vykonává mnoho instrukcí a využít ho pouze jako samostatnou výpočetní jednotku je z hlediska energetické i finanční náročnosti nevýhodné. Implementace hradlových polí přináší nejen v řízení elektrických pohonů zvýšení výpočetního výkonu, ale také snižování energetické náročnosti řízení.

Perspektiva logických programovatelných polí a hardwaerově urychlovaných aplikací je podpořena jejich využíváním i mimo obor elektrických pohonů a trakce. Z důvodu jejich veliké propustnosti, vysokých výpočetních výkonů a nízké energetické náročnosti jsou využívány v AI, machine learningu, zpracování obrazu, těžení kryptoměn a jiných nepohonářských aplikacích.

Nevýhodou problematiky FPGA je jejich složitější programovatelnost z hlediska tvoření aplikace. Aplikace je tvořena určitým postupem (workflow), který kladne vysoké nároky na vzdělání a zkušenosti vývojářů. Většina FPGA je programována pomocí jazyků Verilog či VHDL, které mohou pro softwarově orientované programátory představovat značnou překážku. Proto bylo vyvinuto tvoření aplikací pomocí vyšší úrovně syntézy (HLS), kdy je možné tvořit programy ve vyšších programovacích jazycích jako je například C, C++ či Python. HLS umožnilo rapidní rozšíření a využití Embedded FPGA Accelerated Applications v mnoha aplikacích a značně vylepšilo vývojářský požitek (developer experience, DX) při tvorbě aplikací.

Protože může být náročné vytvořit vlastní architekturu, složenou z CPU a spolupracujícího FPGA, je vhodné při prvotním vývoji aplikace využít dostupné vývojové desky obsahující již předpřipravené propojení jednotlivých komponent. Součástí těchto vývojových desek bývá také mnoho vstupů a výstupů (I/O) pro snadnější využití při lazení a tvoření aplikace. V této práci je využívána vývojová deska Zybo od firmy Digilent. Ovšem autor v textu představuje další možnosti, které mohou být pro konkrétní aplikace a využití vhodnější.

Tato práce se zajímá o aplikace a možné využití FPGA při řízení elektrických pohonů. Autor v ní představuje základní principy Hardware Accelerated Applications, z jakého důvodu je tento přístup perspektivní a proč je vhodné se orientovat tímto směrem.

## 2      **Embedded Systems**

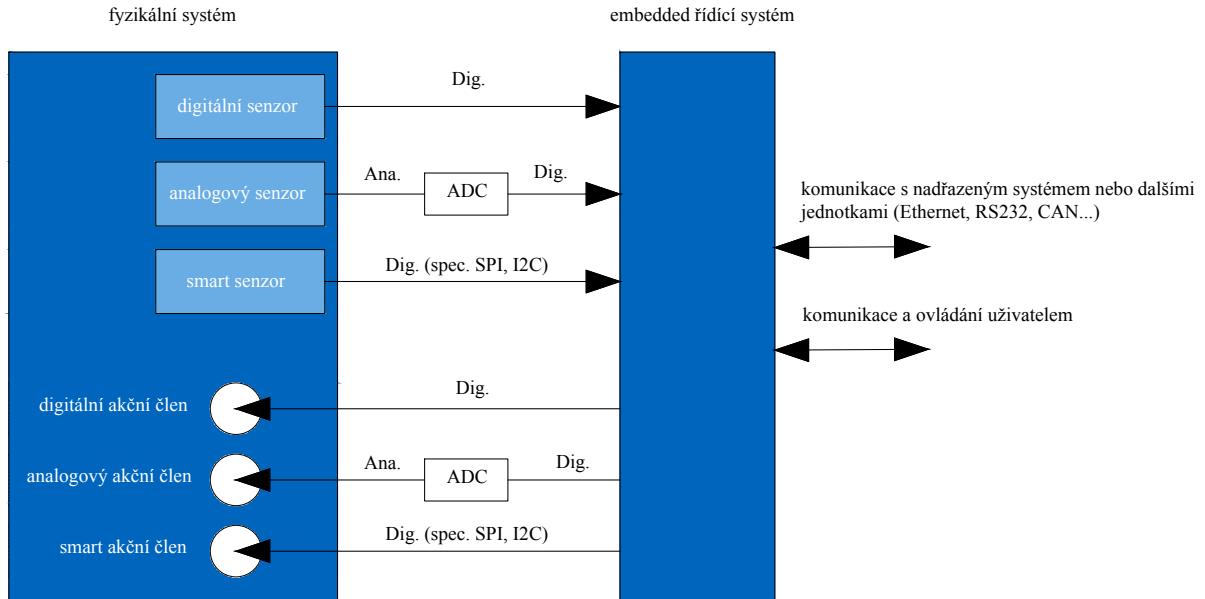
Embedded systémy je název pro skupinu zařízení, obecně systémů, které je možné charakterizovat jako specifické výpočetní zařízení, resp. počítače, které jsou určeny pro podporu funkce nebo řízení nějakého většího celku, produktu nebo fyzikálního systému. Oproti tomu osobní počítač je sice výpočetní zařízení, ale nelze mluvit o embedded systému, protože je určen pro mnoho univerzálních aplikací. [1]

Dalším důležitým rozdílem mezi *Embedded System* a obecným výpočetním zařízením je ten, že v případě embedded systému je interakce mezi systémem a uživatelem uměle omezena na základní ovládání či kontrolu funkce. Není předpokládáno, že by uživatel, jež aplikaci embedded systému využívá, výrazným způsobem zasahoval do jeho funkce. Naopak obecný výpočetní systém je uzpůsoben na podstatné zásahy uživatele. [1] [2]

Do embedded systému obecně vstupují vstupní signály, které jsou následně zpracovány a poté vybrané výsledky výpočtu jsou v podobě výstupní signálů výstupním produktem systému. Tyto produkty mohou pomocí akčních členů zasahovat do řízeného systému či aplikace. Vstupní signály většinou přicházejí ze speciálních snímačů, kompatibilních s embedded systémem (senzor teploty, senzor tlaku, senzor zrychlení, gyroskop apod.). Naopak jeho výstupem signály, vedoucí na konektory, na kterých se dle požadavků objevuje např. specifická hodnota napětí. Nebo mohou na výstupních pinech být připojené LED signalizace, komunikační sběrnice některých komunikačních systémů nebo výstupní LCD displaye. Způsob, kterým jsou kódovány vstupní a výstupní signály je většinou specificky určený aplikaci daného systému. [1]

K obecnému výpočetnímu systému je možné připojit vstupní periferie klasických osbních počítačů – myš, klávesnice, mikrofon. Jako hlavní výstupní periferie obecného systému je monitor. Jeho komunikace s periferiemi je většinou standardizována tak, aby bylo možné periferie libovolně zaměňovat bez změny funkčnosti. [1].

Na obrázku 2 - 1 je zobrazeno názorné blokové schéma řízení fyzikálního systému pomocí embedded systému. Tyto bloky mezi sebou komunikují pomocí digitálních signálů. Pokud tyto signály nejsou digitální, musí se před zpracováním v embedded systému zdiskretizovat.



Obr. 2 - 1 Blokové schéma Embedded systému a řízeného fyzikálního systému. (převzato a upraveno z [2])

## 2.1 Application Specific Integrated Circuit

S tématem embedded systémů se pojí pojem hardware, který je určen pro jedinou aplikaci. Tato skupina zařízení se nazývá *Application Specific Integrated Circuits*, popř. *Hardware* (ASICs, ASHW). V této oblasti je opět využíváno přesvědčení, že pokud je architektura HW přímo specializovaná na jednu aplikaci, je vysoká pravděpodobnost, že ji bude vykonávat bezchybně, kvalitně a rychle.

Tyto aplikace jsou využívány v širokém spektru oborů jako je např. zpracování zvuku, videa, výpočtu apod. Tyto ASIC mohou také vykonávat potřebné rychlé výpočty pro matematické modely elektrických strojů, které jsou využívány např. pro HIL.

Než je tento specifický obvod vytvořen, je nutné jej navrhnout, vyzkoušet a odladit. K tomu slouží logická programovatelná pole, ve kterých je možné požadovaný HW navrhnut a odladit před velko produkcí ASIC. Pokud velko produkce není z ekonomických důvodů možná, jsou FPGA využívány i v produkční oblasti, kde je HW struktura, která by byla přítomna na ASIC, vytvořena přímo na FPGA.

## 2.2 Hardware Accelerated Applications

V mnoha aplikacích, nejen při řízení elektrických pohonů, je vyžadováno, aby výpočty nebo zpracování dat probíhalo vysokou rychlostí. Tento problém nemůže být většinou vyřešen použitím běžného procesoru (CPU), který je optimalizován na provádění obecných komplexních funkcí řízení běhu programu, komunikace či přesunu dat. V moderním světě dochází k exponenciálnímu nárůstu množství dat, které je potřeba zpracovat. Aby tyto data bylo možné v požadovaném čase zpracovat, je třeba využít specifický HW, který bude schopen požadavky rychlosti a výkonu uspokojit. Tento proces se nazývá *Hardware Acceleration*. [3]

Princip hardwaerové akcelerace spočívá v přesunu výpočetně náročných aktivit na zvláštní oddělený hardware. Celkové řízení běhu aplikace a komunikace je ovšem stále přítomno na řídícím CPU. Oddělený hardware, na kterém dochází k akceleraci výpočtu je optimalizován na vykonávanou úlohu a jeho využití přináší zefektivnění běhu celkové aplikace. [3]

Struktura, ve které je využíváno více oddělených hardwarových procesorových a akceleračních jednotek, se často nazývá heterogenní. [3]

Hardwaerová akcelerace poskytuje rychlejší výpočty než CPU, protože využívá značné úrovně paralelismu výpočtů. Oproti tomu klasické CPU vykonává jednotlivé instrukce sériově. I v případě, že CPU má více jader a využívá více vláken, nemůže se úrovni paralelismu při dané energetické náročnosti HW vyrovnat.

Pro HW akceleraci je v mnoha oblastech využíváno několik druhů jednotek, které jsou optimální pro dané aplikace.

**Graphics Processing Units** (GPUs) jsou jednotky, které převážně slouží k akceleraci zpracování a renderování grafických úloh. V době rapidního rozvoje elektroniky a SW je možné využítí GPUs v mnoha odvětví umělé inteligence (AI) či kreativních odvětví. GPUs jsou využívány v aplikacích, kde není kladen veliký důraz na nízkou odezvu (latenci). [3]

**Tensor Processing Units** (TPUs) jsou jednotky, které slouží k provádění algoritmů strojového učení (machine-learning, ML). Jejich přímé datové propojení umožňuje velmi rychlý a přímý přenos dat. Díky přímému připojení nevyžadují využití pamětí, které by přenos dat zpomalovali. [3]

**Field Programmable Gate Arrays** (FPGAs) jsou jednotky, ve kterých není při výrobě pevně daná HW struktura. To umožňuje vytvoření, resp. naprogramování HW dle požadavků akcelerované aplikace. FPGAs jsou využívány i v on-line výpočtech matematických modelů elektrických strojů. Při realizaci této práce je pro akceleraci využíváno právě těchto programovatelných polí.

### 3 Programovatelné hradlové pole – FPGA

#### 3.1 Vývoj FPGA z PLD

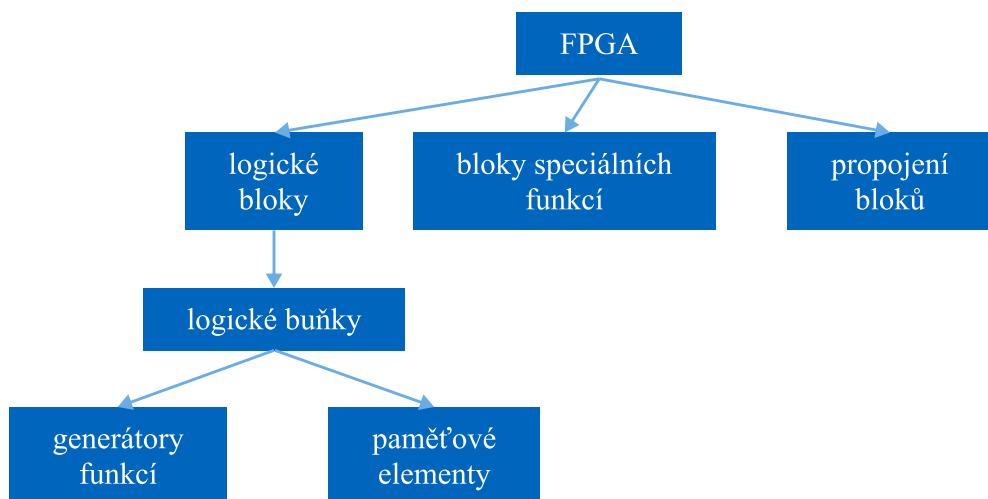
Programovatelné hradlové pole jsou zařízení, jejichž historický vývoj stojí na programovatelných logických zařízení (programmable logic devices, PLD). První PLD fungovala na principu Booleových funkcí součtu násobení (sum of products). Tyto zařízení obsahovala matici více vstupových bloků AND a OR. Programování požadované probíhalo pomocí přerušování vstupů do těchto logických bloků. Později byly do PLA přidány D klopné obvody s multiplexory. Díky těmto součástím bylo možné vytvářet logické kombinační a sekvenční obvody, resp. automaty. Posledním vylepšením PLA, které stalo před zrodem FPGA, spočívalo v umístění více PLA bloků (skládajících se z AND, OR, multiplexeru a D klopného obvodu) na jeden integrovaný čip. Programovatelné spojení různých PLA bloků a výstupů umožnilo vytvořit požadovanou funkci. [1]

#### 3.2 Aktuální složení FPGA

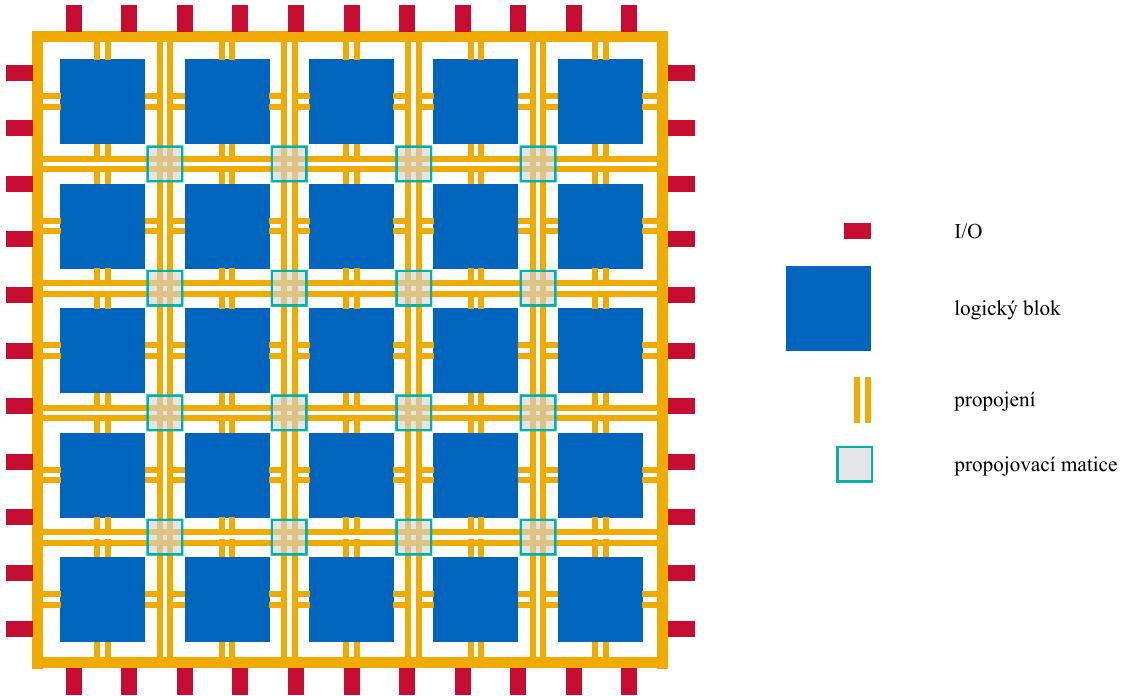
Moderní FPGA se skládají z 2D maticy propojených programovatelných logických bloků, bloků speciálních funkcí a propojů vytvořených pomocí CMOS technologie. Po obvodě FPGA jsou rozmístěny vstupní a výstupní piny (I/O), připojené na zvláštní logické bloky. Použité logické bloky se skládají z mnoha logických buněk, které se skládají z generátorů funkcí a paměťových elementů. [1]

Na obr. 3 - 2 je možné pozorovat názorné schéma základního konceptu uspořádání FPGA. Na schématu jsou vyznačeny logické bloky, jejich propojení, propojovací matice pro aktivování jednotlivých propojů a vstupů a výstupů (I/O) FPGA.

I přesto, že se tato práce věnuje využití FPGA pro řízení elektrických pohonů je vhodné představit základní části FPGA a nastinit jejich funkci.



Obr. 3 - 1 Blokové schéma složení moderních FPGA.



Obr. 3 - 2 Základní koncept uspořádání FPGA.

### 3.2.1 Generátory funkcí

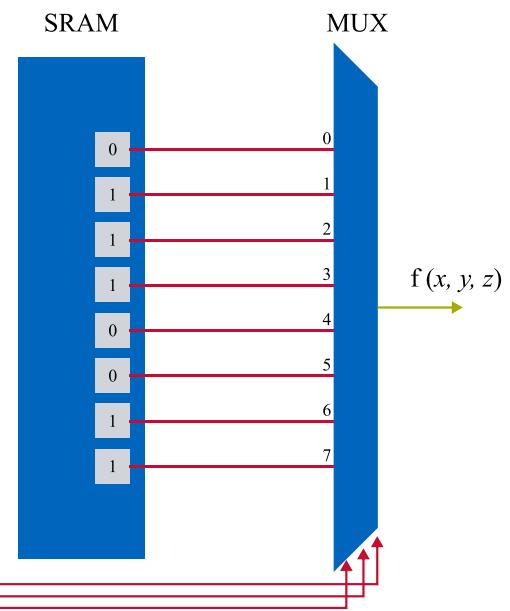
Oproti předchůdcům (PLD), které pro generování funkcí používaly logická hradla tvořená CMOS tranzistory, využívají FPGA generátory funkcí.

Logickou funkci je možné popsat pravdivostní tabulkou, která má určitý počet vstupů a odpovídající počet výstupů. Dle [1] je možné si představit, že se generátor dané funkce skládá ze samostatné statické paměti (SRAM), jejíž výstupy jsou přímo přivedeny na vstup multiplexeru (MUX). Signály výběru výstupů by odpovídaly vstupním proměnným a jednotlivé vstupy do MUX výstupům funkce.

Pro bližší pochopení funkce generátoru funkcí z předchozího odstavce je možné představit realizaci smyšlené logické funkce  $f(x, y, z) = \bar{x}z + y$ . Pravdivostní tabulka této smyšlené logické funkce je zobrazena v tab. 3 - 1. Odpovídající realizace pomocí MUX a SRAM je zobrazena na obr. 3 - 3. Tato reprezentace se nazývá look-up table (LUT). Grafické znázornění inspirováno [1].

Tab. 3 - 1 Pravdivostní tabulka ukázkové funkce, re-alizované v generátoru funkcí, umístěném v logickém bloku FPGA.

$i$	$x$	$y$	$z$	$f(x, y, z)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1



Obr. 3 - 3 Ukázka, jakým způsobem realizuje funkční generátor požadovanou funkci pomocí SRAM a MUX.

Výhoda této reprezentace funkcí oproti logickým hradlům je, že doba zpoždění signálu (propagation delay) pro funkci je konstantní. Respektivě je konstantní, pokud funkci je možné realizovat jednou LUT. Pro relizaci obecné funkce je zapotřebí multiplexer  $2^n \rightarrow 1$  a SRAM s počtem buněk  $2^n$ , kde  $n$  je počet vstupních proměnných dané funkce. [1]

### 3.2.2 Paměťové elementy

Paměťové elementy jsou v FPGA realizovány pomocí D-klopňových obvodů. Tyto obvody mohou při konfiguraci FPGA být nastaveny, že budou reagovat na nástupnou nebo sestupnou hranu časovacího signálu (clock, CLK) řídícího procesoru nebo na úroveň řídícího signálu (latch).[1]

Protože typ latch je citlivý na úroveň signálu, může být problematické dovést požadovaný signál na vstup klopňového obvodu v požadovaném čase. Velmi často jsou proto paměťové členy konfigurovány jako D-klopňové obvody reagující na hranu. Pokud je používán signál CLK vyšších frekvencí, je D-klopňový obvod reagující na hranu snadněji schopný reagovat v požadovaném čase. [1]

Často jsou na vstup paměťových elementů připojeny výstupy multiplexerů generátorů funkci. [1]

### 3.2.3 Logické buňky

Logické buňky jsou elementy, skládající se z generátorů funkci a generátorů funkci. Velmi často se počet logických buněk údává jako jeden ze základních parametrů FPGA, podle kterého je uživatel možný rozhodnout, zda je vhodný pro jeho aplikaci. Pomocí logické buňky nebo skupiny logických buněk je již možné vytvářet plnohodnotnou kombinacní a sekvenční logiku.[1]

### 3.2.4 Logické bloky

Logické bloky se skládají ze spojení několika logických buněk do jedné skupiny. Tato skupina buněk je obvykle na čipu geograficky blízko a proto dochází k minimalizaci zpoždění signálu. Velmi časté

je, že jednotlivé bloky mohou mít již předkonfigurovanou funkci, jako je např. sčítačka, dělička nebo násobička. [1]

### 3.2.5 Propojení bloků

Propojení bloků slouží ke spojení jednotlivých logických bloků a I/O. Pro spínání určených propojů jsou na čipu mezi jednotlivými propoji umístěny propojovací matice, resp. „přepínače“. Ty slouží ke spojení jinak oddělených propojů, logických bloků a I/O. [1] Na obr. 3 - 1 je prezentována 2D struktura pole. Ovšem pro zvětšení počtu LUTs, tudíž výpočetního výkonu, a snížení vzdáleností mezi logickými bloky je v moderních FPGA použita 3D struktura, kdy dochází k vrstvení jednotlivých logických bloků a jejich propojů do výšky. [4]

### 3.2.6 I/O bloky

I/O bloky jsou obvykle umístěny na okraji designu FPGA. Slouží k přivedení resp. vyvedení signálů FPGA na externí připojovací piny struktury. Tyto výstupní bloky mohou využívat různé standardy k přenosu informací typu *single-ended* (napětí vztaženo k referenční nule) (LVTTL, LVCMOS PCI, PCIe, SSTL) nebo typu *double data rate* (diferenciální signál, vztažený k výstupu jiného I/O bloku) (LVDS). I/O bloky jsou strategicky umístěny na okraj struktury, aby byla minimalizována vzdálenost mezi I/O blokem a hranicí FPGA, představující vnější okolí. [1] [4]

### 3.2.7 Bloky speciálních funkcí

Aby došlo např. ke snížení zpoždění signálu, který by bylo nutné vyvést z FPGA do externího CPU a naopak, jsou některé speciální funkce implementovány jako funkční bloky přímo do struktury FPGA. To umožňuje efektivní využití FPGA pro různorodé aplikace. [1]

**Block RAM (BRAM)** je blok, který slouží k uchování dat. Sice by bylo možné vytvořit paměťový blok z *Logických bloků*, ale docházelo by k omezení využití FPGA pro jeho původní aplikace a bylo by potřeba využít mnoho bloků. BRAM mají oddělený vstup a výstup, současně s odděleným CLK. Proto je možné do BRAM zároveň data zapisovat a zároveň z něj číst. [1]

**DSP**, resp. digital signal processing bloky slouží ke zpracování digitálního signálu. V těchto blocích jsou implementované funkce AND, OR, NAND, NOT, násobička sčítačky. Mají nízkou spotřebu. DSP bloky jsou často umístěny geograficky blízko BRAM, které slouží jako „mezipaměti“ (buffer). [1]

**Procesor** implementovaný do struktury FPGA snižuje časové zpoždění při obsluhování FPGA. [1]

**Digital Clock Manager** slouží k vytvoření jiného, resp. nižšího taktovacího signálu CLK, oproti původnímu zdrojovému CLK, pro různé bloky v FPGA. [1]

**Multi-Gigabit Transcievers** slouží k přenosu dat takovým způsobem, aby došlo k minimalizaci vlivu ručení na přenášená data. Obecně obstarávají optimální serializaci a paralelizaci dat. [1]

Struktura FPGA, která je obsahuje všechny zdroje a funkcionality, potřebné pro kompletní realizaci aplikace, se nazývá *platform FPGA*.

## 3.3 Programování

### 3.3.1 Forma tvorby algoritmu pro FPGA

K programování, resp. konfiguraci FPGA je možné přistupovat z několika úrovní. Jednou z využívaných metod popisu požadovaného HW na FPGA je popis struktury/toku signálu obvody (structural/data flow circuits). K tomuto popisu je využíváno jazyků HDL, VHDL a Verilog (Hardware Description Language, VSIC HDL). V těchto jazycích je využíváno logických členů AND, OR, NOT nebo bloků sčítaček

a násobiček. Forma popisu, jež naopak využívá vyššího programovacího jazyka než HDL je nazývána metoda popisu chování obvodů (behavioral circuits). Zatímco HDL slouží k popisu hardware s využitím nízké míry abstrakce, popis ve vyšších programovacích jazycích, které popis pomocí behavioral circuits umožňuje, je pro programátory značně příjemnější, protože využívá běžných procedurálních programovacích jazyků jako je C, C++ nebo Python. Tyto jazyky jsou následně přeloženy do HDL. Po překladu do HDL pomocí *high level synthesis* (HLS) jsou provedeny kroky *synthesis* (syntéza), *place-and-route* (umístění-a-pospojování) a *bitgen* (generace bitstreamu). [1]

Při použití HLS může vzniknout situace, že bude vytvořen algoritmus, který bude takovým způsobem komplexní, že ho nebude možné syntetizovat na FPGA. Oproti tomu při použití popisu pomocí structural/data flow circuits, je prakticky vždy algoritmus syntetizovatelný. [1]

V praxi je k tvorbě algoritmů často využíváno vyšších programovacích jazyků a HLS, protože je tento přístup pro značný počet vývojářů SW srozumitelnější. Dalším častým přístupem v praxi je použití specializovaných SW jako je MATLAB™ a Simulink, které při použití odpovídajících balíčků jsou schopny přeložit vytvořený algoritmus do HDL, který je poté možné dále zpracovat a použít pro konfiguraci FPGA.

### 3.3.2 Konverze HDL na konfigurační Bitstream

V části *Forma tvorby algoritmu pro FPGA* byly představeny dvě hlavní formy tvorby algoritmu pro FPGA. Tyto formy je však pro realizaci na FPGA nutné zpracovat.

Všechny vyšší úrovně reprezentace algoritmů jsou převedeny na HDL. Následným krokem je *syntéza* (synthesis), která slouží k převodu HDL na tzv. *netlist*. Při převodu je HDL převáděna na logické členy AND, OR apod. [1]

Po vytvoření netlistu je nutné rozhodnout, jakým způsobem je možné a výhodné realizovat jednotlivé bloky v logických buňkách a LUT. Konečné sloučení členů závisí na rozsahu vstupů realizovatelných LUT. Proces seskupování logických členů a určování funkce LUT se nazývá mapování (MAP). Výsledkem MAP je opět netlist. Tento netlist však reprezentuje FPGA členy (LUT, klopné obvody apod.). [1]

Po mapování následuje proces umisťování (placement), při kterém je rozhodováno, které logické bloky budou realizovat FPGA členy, získané v kroku MAP. [1]

Bloků, které jsou umístěny ve struktuře FPGA je nutné spojit pomocí dostupných propojů na FPGA. Proces spojování a optimalizace propojů takovým způsobem, aby bylo minimalizováno časové zpoždění signálu, se nazývá *routing*. Obvykle se proces slučuje s MAP do jedné fáze a nazývá se *place-and-route* (PAR). [1]

Posledním krokem je vytvoření binárního souboru, nazývaného *bitstream*, který je poté vkládán do FPGA. Tento proces převede netlist z kroku PAR na nastavení SRAM v jednotlivých logických buňkách FPGA tak, aby byl vytvořen požadovaný design v FPGA. Proces také převede konfiguraci propojů a propojovacích matic do SRAM, ovládající příslušné propoje a matice. [1]

Soubor *bitstream* je poté možný pomocí daného nástroje „nahrát nakonfigurovat jím FPGA“.



Obr. 3 - 4 Blokové schéma převodu aplikace, naprogramované v procedurálním jazyce, na bitstream, který je vhodný pro konfiguraci FPGA.

### **3.4 Spotřeba**

FPGA je využívano pro akceleraci aplikací také pro svou nízkou spotřebu energie. Oproti ASICs však FPGA má stále značnější spotřebu, proto je podnikán výzkum, který má za cíl jejich energetickou náročnost snížit ale zachovat jejich výkon a spolehlivost.

Nižší potřebný výkon pro realizaci nepohonářské aplikace podporuje výzkum a článek [5], ve kterém autoři představují svoji práci, v níž realizovali HW hru. Ve hře je hlavním úkolem aplikace výpočet stínů a odrazů materiálů. Způsob vykreslení, který je v aplikaci použit je nazýván *ray tracing*. Ray tracing je označován jako výpočetně náročný způsob, který není vhodný pro on-line aplikace ale pro vykreslování nepohyblivých obrazů, které není nutné zobrazovat v reálném čase. [6]

Autoři v textu popisují, že v případě využití FPGA pro výpočty v reálném čase byla jeho spotřeba 660 mW. Hru autoři vyzkoušeli spustit také na CPU platformě skládající se z Ryzen™ 4900H 8-core/16 threads 64-bit CPU @ up to 4,4 GHz clock. V případě testování na CPU byla indikována spotřeba 33 W. Tedy při použití FPGA spotřeba klesla přibližně 50x. [5].

I přes nízkou spotřebu energie v FPGA jsou prováděny výzkumy, jak minimalizovat disipaci elektrické energie v podobě tepla a přiblížit se tak energetické náročnosti ASICs.

Disipace energie v FPGA je rozdělena na statickou a dynamickou.

Statická disipace je způsobena zbytkovým proudem tranzistorů ve vypnutém stavu mezi drain a source elektrodou, mezi gate a drain a jevem nazvaným gate direct-tunneling. [7]

Dynamická disipace je způsobena spínacími a vypínacími ztráty použitych tranzistorů (obvykle CMOS) a je závislá na použitém napětí, frekvenci a kapacitě přechodů, kterou je třeba nabít a vybit při spínání a vypínání tranzistorů. [7]

## **3.5 Výpočetní výkon a propustnost**

### **3.6 Využití**

Programovatelná logická hradlová pole se pro svoji nízkou spotřebu, vysoký výpočetní výkon a klesající cenu elektroniky začínají využívat mnohem častěji v mnoha odvětví, ve kterých bylo doposavad využíváno CPU a GPU. Aplikace FPGA je možné v rámci této práce rozdělit na nepohonářské a pohonářské.

#### **3.6.1 Aplikace v nepohonářských odvětví**

Díky univerzalitě FPGAs je možné je využít v mnoha aplikacích různých odvětví. Stále se zvyšující požadavky na výpočetní výkon urychlují nasazování FPGAs do provozu, kde jsou v současné době instalovány CPU nebo GPU.

Poptávka po dostupnosti FPGA způsobila vznik Cloud služeb, které nabízí FPGA výkon on-demand. Jedním z velkých poskytovatelů je Amazon Web Services (AWS), který nabízí FPGA akceleraci v Cloudu. Tuto službu ocení především aplikace, které nejsou vázány na reálný hardware ale potřebují pouze dostupný výpočetní výkon, který mohou v průběhu tvorby, debugingu či realizace aplikace měnit bez nutnosti pořizování výkonných a někdy drahých FPGA desek. Více o *Amazon EC2 F1 Instances* služby virtuálních FPGA je dostupné na [8].

Existuje mnoho výpočetně náročných aplikací jako jsou např. výpočty finančních modelů pro ekonomiku, výpočty pro bioinformatiku, seismické modelování při hledání vzácných surovin apod. Více informací o těchto výpočetně náročných aplikacích je možné získat v [9].

Na akceleraci zpracování audiovizuálních děl je převážně určeno GPU. Ovšem pro aplikace, v nichž je vyžadováno zpracování obrazu v reálném čase s minimální spotřebou energie a nízkou hmotností apli-

kace, je často využíváno FPGA. Aplikace využití FPGA pro vozidla, která analyzují okolní prostor jsou popsány v [10]. Tyto aplikace nesou souhrnný název „inteligent spaces applications“. Obvykle je pro analýzu okolního prostoru využíváno více kamer, z nichž každá obsahuje vlastní výpočetní jádro (FPGA). Díky tomu výpočetně náročné aplikace, jako např. analýza hloubky obrazu pro rozpoznání objektů, probíhá v FPGA a ostatní nenáročné výpočty a řízení v SW. [10]

Protože momentálním trendem je snižování energetické náročnosti a zvyšování výpočetního výkonu dochází neustále k vývoji nových aplikací, které využívají FPGA pro akceleraci výpočetně náročných kroků, proto není možné všechny v tomto textu obsáhnout.

### 3.6.2 Aplikace v elektrických pohonech

V některých případech je elektrický pohon rozměrná a finančně náročná sestava, proto zkoumání určitých kritických stavů těchto soustav by mohlo být ekonomicky i technicky nevýhodné. V tomto případě je vhodné vytvořit přesný matematický model jednotlivých analyzovaných součástí a nezbytné náročné výpočty akcelerovat pomocí FPGA. Na základě odezvy modelu je poté možné analyzovat stavy, které by v případě analýzy na reálném stroji mohly způsobit jeho destrukci či částečnou ztrátu funkčnosti. Proto se v průmyslu využívá Hardware-in-the-loop simulation (HILS), kdy je vytvořen požadovaný matematický model, který poskytuje elektrické signály do testovaného systému a na základě jeho reakce je možné vyhodnotit díky matematickému modelu jakým způsobem by se choval reálný modelovaný systém. [10], [11]

Kromě HIL simulace je možné FPGA využít také pro řízení elektrických pohonů. Možnosti realizace řízení AC elektrických strojů pomocí FPGA a analogově digitálních převodníků (ADC) jsou prezentovány v [12]. V dokumentu jsou popisovány tři realizace řízení, resp. regulace pohonu. Nejprve byla regulace realizována pomocí hystérezních on-off regulátorů, následně byly použity PI regulátory. Pomocí nich byl pohon regulován na základně měření a změny vektoru statorového proudu, resp. jeho složek  $\alpha\beta$  po aplikování Clarkové transformace. Jako poslení prezentovaný způsob autoři realizovali model ovládání synchronního motoru na základě prediktivních regulátorů. [12]

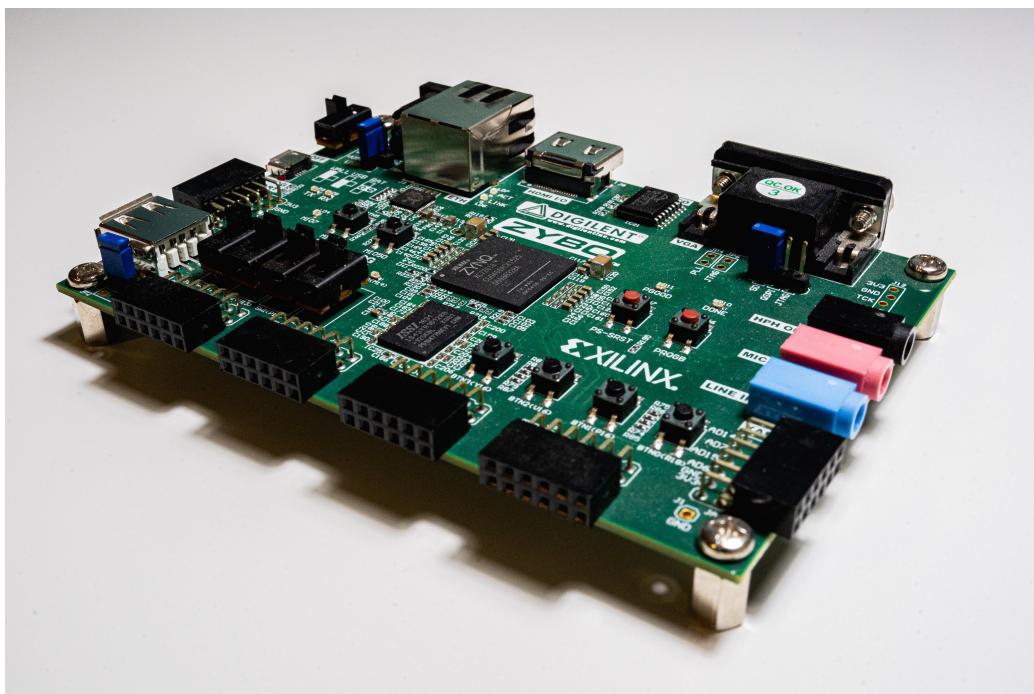
Všechny prezentované způsoby regulace v [12] byly před syntézou realizovány v prostředí MATLAB™ a Simulink. Tento způsob tvorby modelů a algoritmů je v praxi upřednostňován, protože umožnuje i expertům na řízení a regulaci pracovat na dané problematice bez znalostí mikroelektroniky, programování v HDL a způsobu fungování FPGA. Oproti tomu je třeba zvážit, jaké jsou požadavky na rychlosť, výkonnost a optimalizované řízení aplikace a zdali použití předpřipravených knihoven a zjednodušených nástrojů nebude mít příliš značný vliv na rychlosť výpočtu a tudíž zpracování dat a řízení v reálném čase. [12]

## 4 Vývojová deska Digilent Zynq

Vývoj akcelerovaných aplikací je možné realizovat na relativně velikém množství dostupného HW. V některých případech je design vývojových desek dokonce výrobcem uveřejňován a tudíž v případě dostatečných znalostí je dokonce možné si sestavit vlastní HW s dostupných komponent takovým způsobem, aby vyhovoval požadované embedded aplikaci. Výhodné ovšem je využít již připravená řešení vývojových desek, které zjednoduší prvotní tvorbu aplikace.

V této práci je realizován vývoj aplikace na vývojové desce *Digilent ZYBO Zynq-7000 ARM/FPGA SoC Trainer Board* od firmy Digilent. [13] Jedná se o model vývojové desky, který byl nahrazen novějšími variantami s označením *ZYBO Z7-10* a *ZYBO Z7-20*, které jsou stále v aktivním prodeji. Hlavním rozdílem představených desek je verze Zynq čipu, který v moderních deskách disponuje ARM procesorem s vyšší taktovací frekvencí a s modernějším FPGA s vyšším počtem LUT, klopných obvodů a s rozsáhlejší pamětí RAM apod. Bližší porovnání specifikací těchto desek je dostupné na [14].

V další části textu jsou představeny významné komponenty vývojové desky *Digilent ZYBO Zynq-7000 ARM/FPGA SoC Trainer Board*.



Z naznačené architektury je možné vyvodit, že se SoC skládá ze dvou hlavních částí, které je možné dále rozdělit na jednotlivé bloky:

- Processing System (PS),
  - Application processor unit (APU),
  - Memory interfaces,
  - I/O peripherals (IOP),
  - Interconnect,
- Programmable Logic (PL).

### **Blok PS**

Blok PS se skládá z dílčích bloků, které neslouží k akceleraci aplikací, ale k podpoře běhu hostitelského programu. Blok PS reprezentuje prakticky celou architekturu čipu vyjma části věnované PL.

### **Blok APU**

Blok APU obsahuje CPU Cortex-A9 a další podpůrné bloky jako např. přímý přístup do paměti (DMA controller), Genral interrupt controller (GIC) pro maskování a ovládání přerušení, watchdog a další podpůrné bloky.

### **Blok Memory interfaces**

Memory interfaces slouží k přístupu APU a PL k pamětím typu DDR3, DDR3L, DDR2 a LPDDR-2. Je možné také vybrat, zda šířka sběrnice bude 16 nebo 32 bitů. K dispozici jsou zakoponované kotroléry přenosu dat pro optimalizaci rychlosti, Static Memory Controller nebo Quad-SPI Controller.

### **Blok IOP**

IOP se skládá ze standardizovaných rozhraní vhodných pro průmyslovou komunikaci. Obsahuje nař. GPIO, Gigabit Ethernet, dva bloky USB Controller, dva bloky SD/SDIO Controller pro bootování SD karty, dva bloky SPI Controller, dva bloky CAN Controller, dva bloky UART Controller a dva bloky I2C Controller.

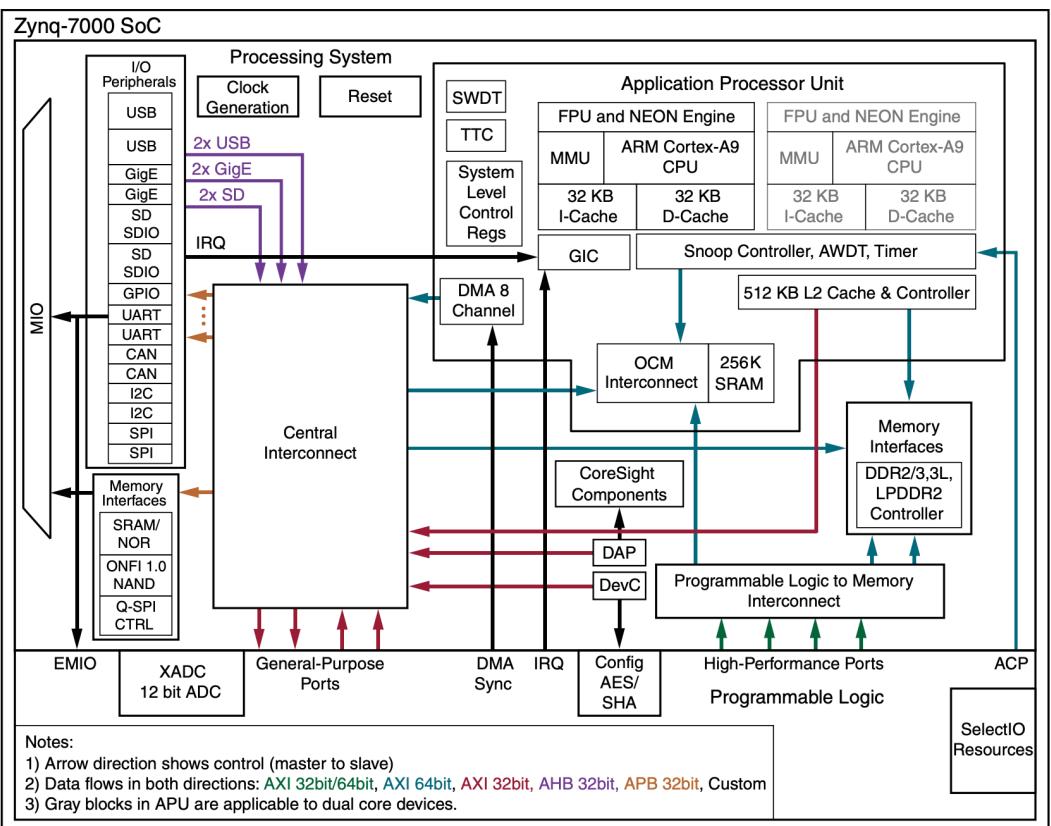
### **Blok Interconnect**

Blok Interconnect, resp. na obr. 4 - 2 označený Central Interconnect slouží k propojení jednotlivých bloků SoC dle požadované technologie a rychlosti.

### **Blok PL**

Blok PL reprezentuje logické programovatelné pole (FPGA), v němž jsou zakomponovány další podpůrné bloky jako např. blok zpracování digitálních signálů, řízení taktovacích hodin, analogově digitální převodník (ADC) apod.

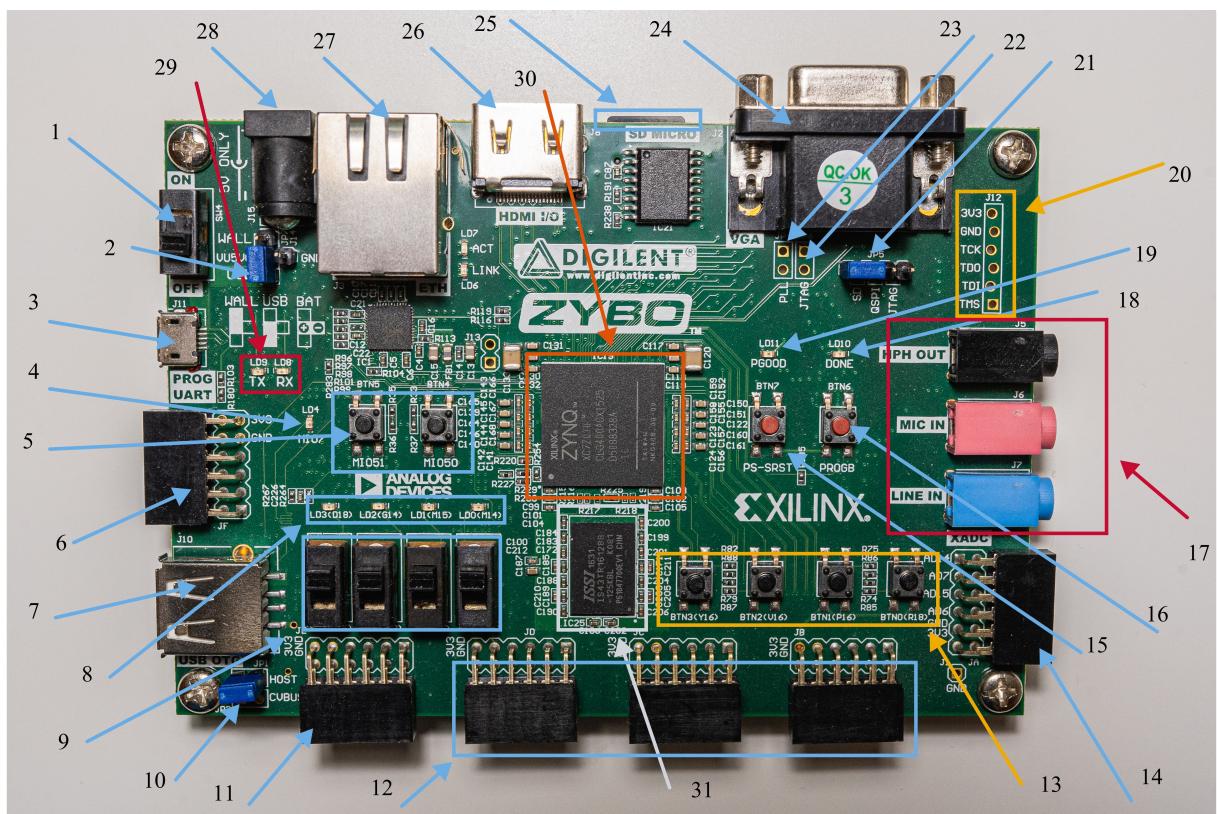
Veškeré technické specifikace, složení a parametry jmenovaných bloků a jsou uvedeny v [15].



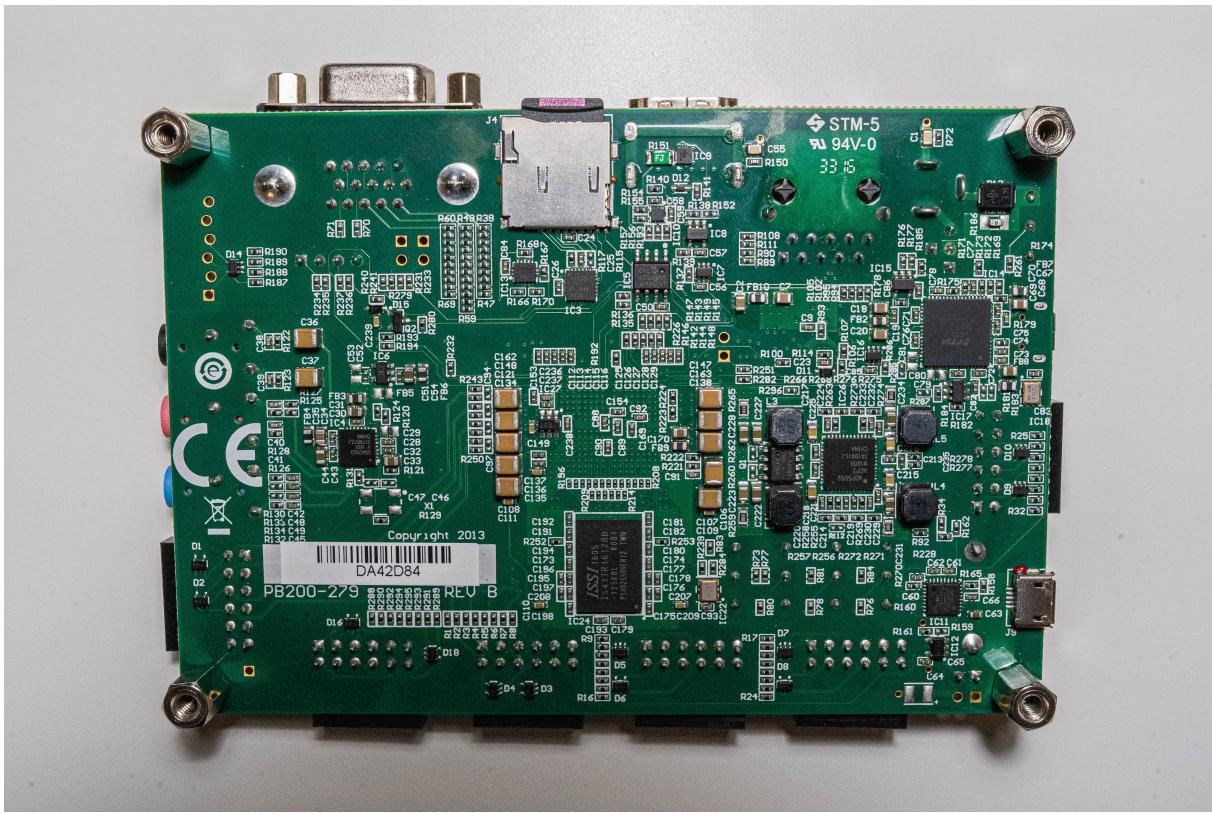
Obr. 4 - 2 Detailní schéma čipu Zynq-7000, umístěného na vývojové desce Digilent ZYBO Zynq-7000 ARM/FPGA SoC Trainer Board. (převzato z [15])

#### 4.1.2 Uspořádání vývojové desky Zybo Zynq-7000

Na obr. 4 - 3 je zobrazen horní pohled na vývojovou desku, na které jsou vyznačeny významné části, kterým je vhodné věnovat při práci s deskou pozornost. Číselné označení koresponduje s označením a vysvětlivkou v tabulce 4 - 1. Na spodní části desky se nachází menší počet součástek, které by byly významné pro ovládání a práci s deskou. Nicméně pro doplnění je spodní strana desky zobrazena na obr. 4 - 4.



Obr. 4 - 3 Vývojová deska Digilent ZYBO Zynq-7000 ARM/FPGA SoC Trainer Board vrchní pohled s vyznačením komponent.



Obr. 4 - 4 Vývojová deska Digilent ZYBO Zynq-7000 ARM/FPGA SoC Trainer Board spodní pohled.

Tab. 4 - I Popis označených komponent na vývojové desce Digilent Zybo Zynq-7000. (informace a značení z [16])

označení	popis	poznámka
1	Power Switch	galvanické sepnutí napájecího obvodu
2	Power Select Jumper and Battery Header	výběr napájecího vstupu konektor, USB, baterie
3	Shared UART/JTAG USB port	komunikace UART a JTAG debugging
4	MIO LED	multiplexed LED – možnost výběru signálu
5	MIO Pushbuttons (2)	multiplexed input
6	MIO Pmod	možnost připojení periférií
7	USB OTG Connectors	USB port typ A/micro USB (spodní část)
8	Logic LEDs (4)	zobrazování 1/0
9	Logic Slide Switches (4)	logický vstup 1/0
10	USB OTG Host/Device Select Jumpers	výběr módu zařízení
11	Standard Pmod	chráněné Pmod, limitace max. přenosu informace
12	High-speed Pmods (3)	jako standard ale bez ochrany, vyšší rychlosť
13	Logic Pushbuttons (4)	logický vstup 1/0
14	XADC Pmod	možnost analog/digi input/output, spojeno s ADC v Zynq
15	Processor Reset Pushbutton	reset PL, paměti v PS
16	Logic Configuration reset Pushbutton	reset PL, zrušení DONE informace
17	Audio Codec Connectors	stereo line in, mono mikrofon, stereo output
18	Logic Configuration Done LED	signál o úspěšném dokončení konfigurace PL
19	Board Power Good LED	1/0, 1 – nominální napětí na všech sběrnících
20	JTAG Port for optional external cable	externí JTAG
21	Programming Mode Jumper	výběr „programovacího vstupu“, SD karta, QSPI, JTAG
22	Independent JTAG Mode Enable Jumper	JTAG mimo PS, viditelné pouze PL
23	PLL Bypass Jumper	přemostění PLL (CLK), pro možnost konfigurace PLL
24	VGA connector	připojení displeje
25	microSD connector	na spodní straně
26	HDMI Sink/Source Connector	input/output, nutné implementovat encoding a decoding v logice
27	Ethernet RJ45 Connector	komunikace
28	Power Jack	napájení 5 V/2,5 A
29	TX/RX LED	indikace UART komunikace
30	Xilinx Zynq SoC	srdce desky
31	DDR2 Memory	RAM

## 4.2 Možné alternativy

### 5 Model stroje

Jak již bylo představeno v předchozích částečkách textu, akcelerované aplikace v FPGA je možné použít na různé účely. Součástí této práce je v PL realizovat akcelerovaný výpočet matematického modelu stroje. V této práci bude využito matematického modelu asynchronního motoru.

#### 5.1 Představení stroje

#### 5.2 Matematický popis modelu stroje

Tab. 5 - 2 Změřené parametry stroje.

Tab. 5 - 1 Štítkové údaje stroje.

$P_n$	12 kW
$U_n$	380 V
$I_n$	22 A
$n_n$	$1460 \text{ min}^{-1}$
$f_n$	50 Hz
$\cos(\varphi_n)$	0.8
$p_p$	2

$R_1$	370 mΩ
$R_2$	225 mΩ
$L_{1\sigma}$	2,27 mH
$L_{2\sigma}$	2,27 mH
$L_m$	82,5 mH
$L_1$	84,77 mH
$L_2$	84,77 mH
$J$	0,4 kg·m <sup>2</sup>

Kde  $P_n$  (W) je jmenovitý výkon stroje,  $I_n$  (A) je jmenovitý fázový proud stroje (efektivní hodnota),  $U_n$  (V) je jmenovité sdružené napájecí napětí stroje,  $f_n$  (Hz) je jmenovitá napájecí frekvence stroje,  $\cos(\varphi_n)$  (-) je jmenovitý účinník stroje,  $n_n$  ( $\text{min}^{-1}$ ) jsou jmenovité otáčky stroje,  $p_p$  (-) je počet polpárů stroje,  $R_1$  (Ω), resp.  $R_2$  (Ω) je statorový, resp. rotorový odpor,  $L_{1\sigma}$  (H), resp.  $L_{2\sigma}$  (H) je statorová, resp. rotorová rozptylová indukčnost stroje,  $L_m$  (H) je magnetizační indukčnost stroje,  $L_1$  (H), resp.  $L_2$  (H) je statorová, resp. rotorová indukčnost,  $J$  (kg·m<sup>2</sup>) je moment setrvačnosti hřídele.

V případě tvorby modelu, je využit model založený na výpočtu složek vektorů statorového proudu  $i_1$  a rotorového toku  $\psi_2$  v souřadnicovém systému  $\alpha\beta$  spojeném se statorem. Tedy při použití  $\omega_k = 0$ . Bude volena konstanta  $K = 2/3$ . Poté bude stavový popis systému vypadat následovně.

$$\frac{d}{dt} \begin{bmatrix} i_{1\alpha} \\ i_{1\beta} \\ \psi_{2\alpha} \\ \psi_{2\beta} \end{bmatrix} = \begin{bmatrix} -\frac{R_2 L_m^2 + L_2^2 R_1}{\sigma L_1 L_2^2} & 0 & \frac{L_m R_2}{\sigma L_1 L_2^2} & \frac{L_m}{\sigma L_1 L_2} \omega \\ 0 & -\frac{R_2 L_m^2 + L_2^2 R_1}{\sigma L_1 L_2^2} & -\frac{L_m}{\sigma L_1 L_2} \omega & \frac{L_m R_2}{\sigma L_1 L_2^2} \\ \frac{L_m R_2}{L_2} & 0 & -\frac{R_2}{L_2} & -\omega \\ 0 & \frac{L_m R_2}{L_2} & \omega - \frac{R_2}{L_2} & 0 \end{bmatrix} \begin{bmatrix} i_{1\alpha} \\ i_{1\beta} \\ \psi_{2\alpha} \\ \psi_{2\beta} \end{bmatrix} + \begin{bmatrix} \frac{1}{\sigma L_1} & 0 \\ 0 & \frac{1}{\sigma L_1} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_{1\alpha} \\ u_{1\beta} \\ u_{2\alpha} \\ u_{2\beta} \end{bmatrix}. \quad (5-1)$$

Stavový popis je vhodné doplnit o další rovnice, jež budou v simulaci využity.

$$M = \frac{3}{2} p_p \frac{L_m}{L_2} (\psi_{2\alpha} i_{1\beta} - \psi_{2\beta} i_{1\alpha}), \quad (5-2)$$

$$M - M_z = J \frac{d\Omega}{dt}, \quad (5-3)$$

$$\omega = p_p \Omega, \quad (5-4)$$

kde  $\sigma = 1 - L_m^2/(L_1 L_2)$  (-) je tzv. rozptyl,  $i_{1\alpha}$  (A) a  $i_{1\beta}$  (A) jsou složky vektoru statorového proudu  $i_1$  (A),  $\psi_{2\alpha}$  (Wb) a  $\psi_{2\beta}$  (Wb) jsou složky vektoru rotorového magnetického toku  $\psi_2$  (Wb),  $u_{1\alpha}$  (V) a  $u_{1\beta}$  (V) jsou složky statorového napětí  $u_1$  (V),  $p_p$  (-) je počet polpáru stroje,  $\omega$  ( $s^{-1}$ ) je elektrická úhlová rychlosť hřídele,  $\Omega$  ( $s^{-1}$ ) je mechanická úhlová rychlosť hřídele,  $M$  je vnitřní elektromechanický moment stroje a  $M_z$  (Nm) je moment zátěžný.

### 5.3 Optimalizace modelu

## 6 Aplikace pro FPGA a CPU

V této části jsou představeny jednotlivé nástroje, využívané při tvorbě programu pro CPU a akcelerované aplikace na FPGA. Je důležité zmínit, že na tzv. „host“ (ARM procesor) je skutečně možné spustit řídící program akcelerované aplikace. Na PL (FPGA) je ovšem vytvořen HW, který reprezentuje myšlené algoritmy aplikace. Proto není korektně správné mluvit o tom, že se vytváří program pro FPGA. Proto v této práci bude používáno označení pro vytváření HW na PL vytváření „kernel“.

Dále jsou v této části představeny postupy, které vedou k úspěšnému používání vývojových nástrojů. Následně je popsána tvorba samostatné akcelerované aplikace, která je hlavní náplní této práce.

### 6.1 Použité nástroje

Tvorba akcelerované aplikace může být obecně prováděna více způsoby. Tento způsob závisí na použitém vývojovém nástroji HW. V této práci je využíváno SoC od firmy Xilinx, proto je výhodné využívat již připravené nástroje, které umožní snazší vývoj SW, tvorbu HW a přípravu systému na SoC.

Veškerý používaný SW v této práci od firmy Xilinx je po registraci volně dostupný ke stažení na [17].

Tato část představuje jednotlivé použité nástroje firmy Xilinx a nastiňuje postupy, které je třeba do držet k úspěšnému zprovoznění a používání nástrojů.

#### 6.1.1 Xilinx Vivado

Xilinx Vivado je nástroj, používaný pro tvorbu HW architektury, resp. platformy, pro kterou bude následně možné vytvořit akcelerovanou aplikaci. Ve Vivado je možné tvořit HW návrh, převeditelný do HDL, který bude spustitelný v PL bez použití Vitis HLS. Kromě pouhé tvorby HW návrhu je ve Vivado dostupý simulátor a analýza vytvořeného návrhu.

Xilinx Vivado je součástí instalačního balíčku *Xilinx Unified Installer*, dostupného z [17]. Pro práci tvorbu HLS aplikace, kterou jsou schopni vytvořit i software vývojáři je vhodné stáhnout kompletní balíček s velikostí cca 89,4 GB (verze 2022.2 SFD), která obsahuje také nástroj Xilinx Vitis.

#### 6.1.2 Xilinx Vitis

Xilinx Vitis je nástroj, který slouží k vytváření akcelerovaných aplikací na zařízeních firmy Xilinx. Tento nástroj obsahuje základní vrstvu s názvem Xilinx Vitis HLS, která slouží jako jádro převodu vytvářených aplikací v C, C++, OpenCL do RTL. V programu Xilinx Vitis bude vytvářena největší část aplikace, proto je vhodné nastínit postup, jakým Vitis pracuje.

Nejprve je vytvořen tzv. „host program“, který je vyvíjen v C/C++, používající Xilinx Runtime (XRT) Application Programming Interface (API). Tento program je následně kompilován pomocí g++ kompilátoru, který vytvoří spustitelný soubor pro procesor. Tento host program komunikuje s akcelerovanou částí aplikace (kernel), umístěným v PL v FPGA. [18]

Poté Vitis HLS compiler přeloží C/C++ zdrojový kód pro kernel do register transfer levelu (RTL). Produkty této komplikace mají příponu „.xo“ (Xilinx Object) a mohou být spojovány do binárního souboru

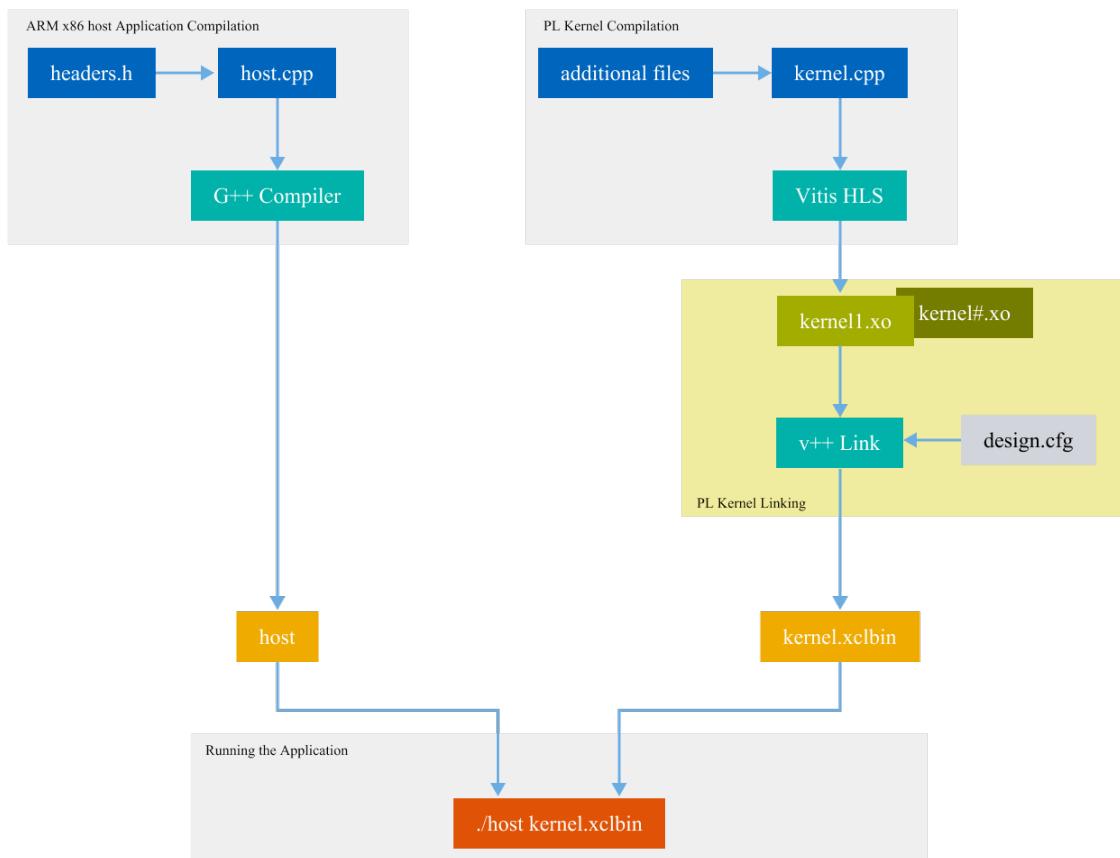
s příponou „.xclbin“ pomocí Vitis linkeru. [18]

Na obr. 6 - 1 je blokově znázorněn postup tvorby spustitelné aplikace v programu Vitis. Tento diagram předpokládá, že již byl vytvořen požadovaný HW ve Vivado a PetaLinux systém.

Blok označený *ARM x86 host Application Compilation* naznačuje, jakým způsobem je vytvářena aplikace pro host procesor. S pomocí hlavičkových a zdrojových souborů je pomocí G++ compileru vytvořen spustitelný soubor *host*, který je dále poté využit při spuštění akcelerované aplikace (kernel) na PL.

[18] Druhá věten naznačuje, jakým způsobem je vytvářena akcelerovaná aplikace, která bude realizována v PL. Pomocné a hlavičkové soubory napsané v C++ jsou pomocí Vitis HLS zpracovány dle postupu uvedeném v [18] (sekce Vitis Application Development Flow), výstupem z Vitis HLS jsou jendotlivé soubory Xilinx object (.xo), které jsou pomocí v++ Linkeru a volitelných dalších konfigurací spojeny-/spárovány do souboru *kernel.xclbin*, který je poté možné nakonfigurovat PL. [18]

Produkty větve programu pro CPU a konfigurace pro PL je po jejich dokončení možné použít na dané vývojové desce. Host program zařídí nakonfigurování PL pomocí souboru *kernel.xclbin* a následné zpracování výsledků. Blok s názvem *Running the Application* je kompletně vykonáván v prostředí PetaLinux v simulátoru (QEMU) nebo na fyzickém zařízení (vývojová deska).



Obr. 6 - 1 Blokový diagram tvorby spustitelné aplikace v prostředí Vitis. (převzato z [18], upraveno)

### 6.1.3 PetaLinux

I přes to, že se tento nástroj nazývá PetaLinux, nejedná se o samostatný systém. PetaLinux je nástroj, který slouží k vytvoření systému, který bude spuštěn na ARM procesoru na vývojové desce. Z tohoto systému je poté možné spuštět navazující programy pro host a kernel.

PetaLinux poskytuje distribuci systému Linux, který uživatel před tvorbou aplikace pomocí Xilinx Vitis využívá k tomu, aby vytvořil „operační systém“, na kterém bude schopen spuštět vytvářené aplikace. Tento systém je schopen si nakonfigurovat dle požadavků aplikace. Při tvoření tohoto systému je možné vybrat balíčky, které budou do systému nainstalovány, vytvořit uživatele systému nebo vybrat kde v paměti bude systém umístěn (RAM, SD karta apod.). [19]

PetaLinux Tools mohou sloužit také k debuggingu a virtualizaci systému pomocí QEMU. V případě tvorby systému, který je konfigurovaný uživatelem, je třeba postupovat obezřetně a z rozumem, protože se jedná o časově náročný postup.

#### 6.1.4 Programovací prostředí – operační systém Linux

Pro práci s představenými vývojovými prostředími jako je Xilinx Vivado, Xilinx Vitis a PetaLinux je nutné využívat podporovaných operačních systémů, které umožňují správnou funkci využitých nástrojů.

Jednotlivé požadavky na operační systémy je možné nalézt na stránkách dokumentace <https://docs.xilinx.com>. Pro nejnovější verze v době zpracování této práce jsou požadavky pro Xilinx Vivado dostupné v [20]. Požadavky na operační systém pro Xilinx Vitis v [18]. Pro využívání a tvorbu PetaLinux je třeba dodržet systémové požadavky uvedené v [21]. Pokud uživatel využívá starších verzí vývojových nástrojů, je doporučováno využít operační systém Linux. Pro tuto práci bylo využíván systém Ubuntu 18.04 LTS (Bionic Beaver), dostupný ke stažení na adrese <http://old-releases.ubuntu.com>. V průběhu práce došlo k aktualizování verzí SW, které byly z počátku roku 2022 a veškerá práce byla přenesena na novější verzi systému Ubuntu 20.04 LTS (Focal Fossa).

Je důležité poznamenat, že neplatí skutečnost, že když např. Vivado podporuje některou z novějších verzí Ubuntu, bude jí podporovat také PetaLinux. Vždy je doporučeno využívat starší verze a kontrolovat vzájemnou kompatibilitu, aby se předešlo zbytečné ztrátě času. Stahování SW, instalace a nastavování nástrojů je značně časově náročné.

V případě využívání představených nástrojů a systému Linux je třeba dbát na správné postupy instalaci a v případě problémů využívat dostupné dokumentace.

### 6.2 Tvorba HW architektury Xilinx Vivado

Aby bylo možné vytvořit akcelerovanou aplikaci ve Vitis pomocí HLS C++, je třeba připravit platformu, resp. hardware, pro který bude daná aplikace vyvíjena. K tvorbě platformy je využito SW Xilinx Vivado. V tomto programu je možné konfigurovat jednotlivé použité prvky jako je ZynQ jednotka, GPIO jednotky, DSP a další. Výsledkem tvorby platformy v této práci je vytvoření XSA souboru, který bude použit pro konfiguraci tvorby PetaLinux systému. Ve Vivado je možné také vytvářet aplikace přímo v VHDL, ale této funkcionality nebude v této práci využito.

V této sekci bude popsána tvorba platformy pro vývojovou desku Zybo Zynq-7000. Tvorba platformy pro jiné vývojové desky nebo zařízení je specifická pro daného výrobce a může se v nějakých případech lišit. Ovšem základní úkony jako je vložení potřebných bloků a celkový postup tvorby je zachován.

Postup tvorby platformy byl částečně převzat z [22].

#### 6.2.1 Vivado Board Files

Aby bylo možné snadněji vytvořit potřebnou HW architekturu, firmy většinou dodávají tzv. *Board Files*, jež jsou soubory, obsahující různá přednastavení, konfigurace, informace a podmínky připojení IP bloků k reálným součástím. [23]

Samozřejmě by bylo možné HW architekturu vytvořit i bez těchto předkonfigurovaných souborů,

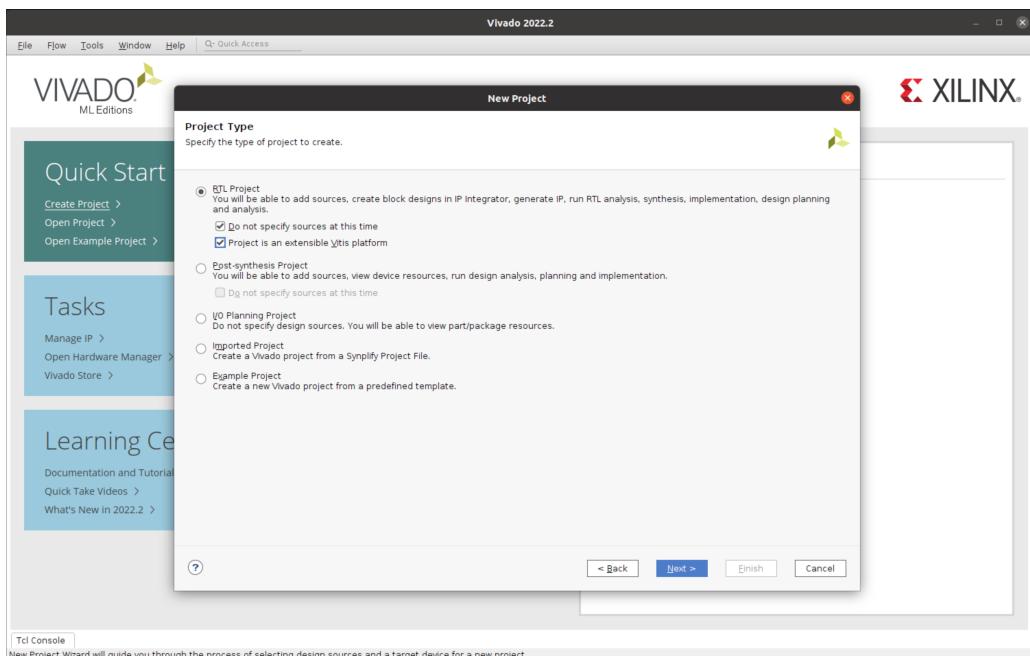
ovšem proces by byl značně náročnější. Pro používanou vývojovou desku Digilent Zybo Zynq-7000 je možné stáhnout tyto soubory z [23]. Způsob instalace board files je popsán v oficiální dokumentaci firmy Digilent, Inc. v [24].

Po úspěšné instalaci souborů je možné spustit Xilinx Vivado a vytvořit potřebnou HW architekturu pro akcelerovanou aplikaci.

## 6.2.2 Tvorba HW architektury pro Digilent Zybo Zynq-7000 vývojovou desku

V této části bude představen postup tvorby HW architektury pro vývojovou desku Digilent Zybo Zynq-7000. V případě, že je požadováno aktivování dalších speciálních bloků v čipu, např. pro specifickou akcelerovanou aplikaci, je třeba do blokového designu vložit odpovídající prvky, které zajistí potřebnou funkcionalitu.

Nejprve je nutné vytvořit nový Vivado projekt a pojmenovat ho dle požadavků. Při výběru typu projektu je nutné zvolit možnost *RTL Project* a aktivovat možnost *Project is an extensible Vitis platform*. Tento úkon je naznačen na obr. 6 - 2.

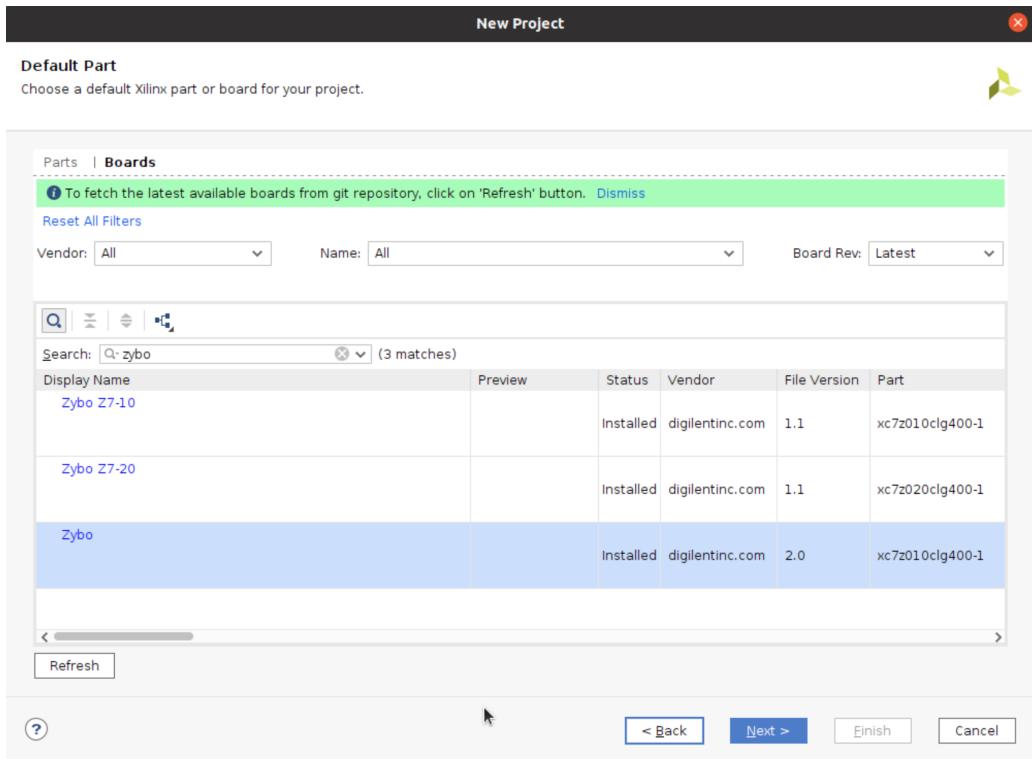


Obr. 6 - 2 Xilinx Vivado – volba typu projektu.

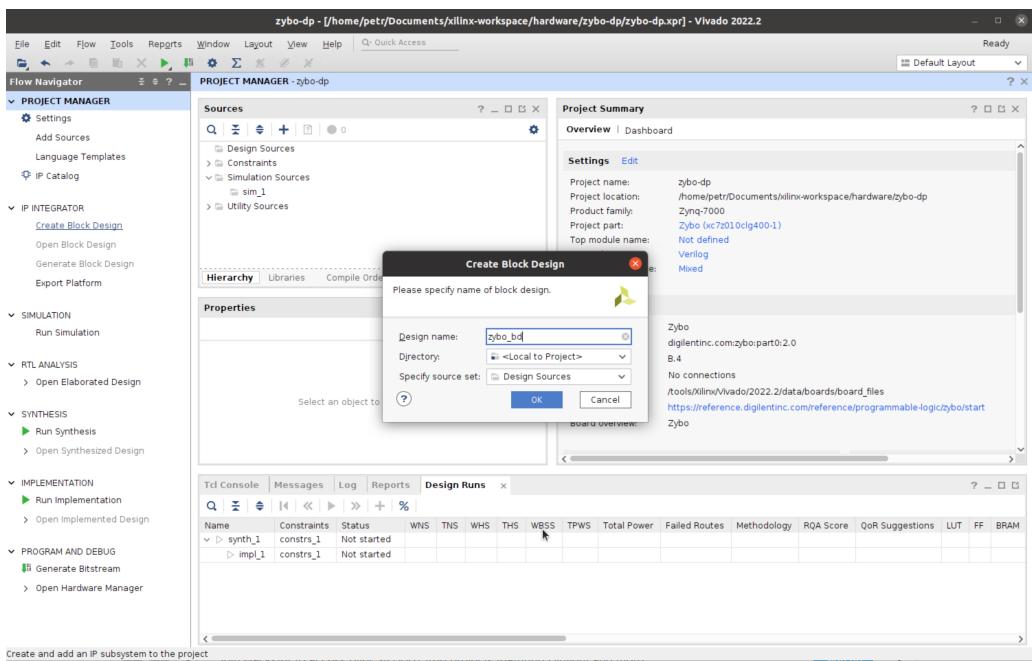
Následně v nabídce *Xilinx part* vybrat možnost *Board* a do vyhledávání zadat název využívané desky. V této práci bude využíváno desky *Zybo*. Díky instalovaným *board files*, představených v části *Vivado Board Files*, je možné nalézt požadovanou desku verze 2.0 a pokračovat v tvorbě HW. Výběr základního HW je zobrazen na obr. 6 - 3.

Po úspěšné inicializaci projektu je pro další pokračování nutné v menu *Flow Navigator/IP Integrator* zvolit možnost *Create Block Design* a vytvořit nový blokový design HW. Tvorba blokového designu je naznačena na obr. 6 - 4.

Nyní je možné již konečně přistoupit ke tvoření vlastní architektury. Jako první krok je nutné vložit blok *ZYNQ7 Processing System* a zvolit nově zobrazenou možnost *Run Block Automation*. V těchto pomocných automatizacích je většinou výhodné ponechávat nastavené výchozí hodnoty, které jsou pro většinu tvořený HW architektur a akcelerovaných aplikací dostačující. Menu s výběrem IP bloku pro

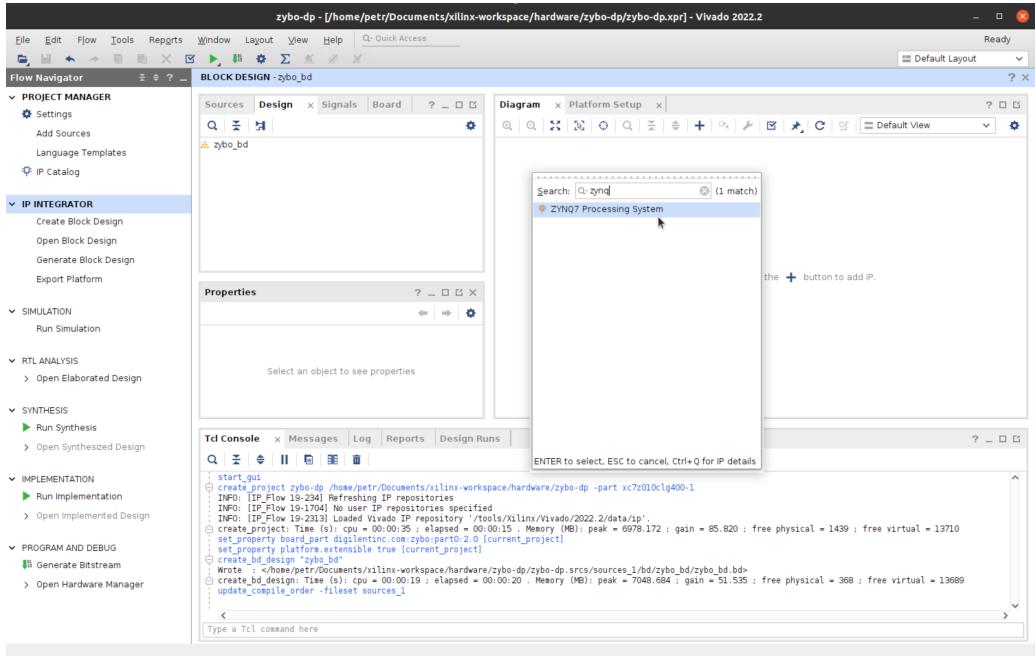


Obr. 6 - 3 Xilinx Vivado – výběr základního HW, pro který bude vytvářena architektura.



Obr. 6 - 4 Xilinx Vivado – vytváření Block Design.

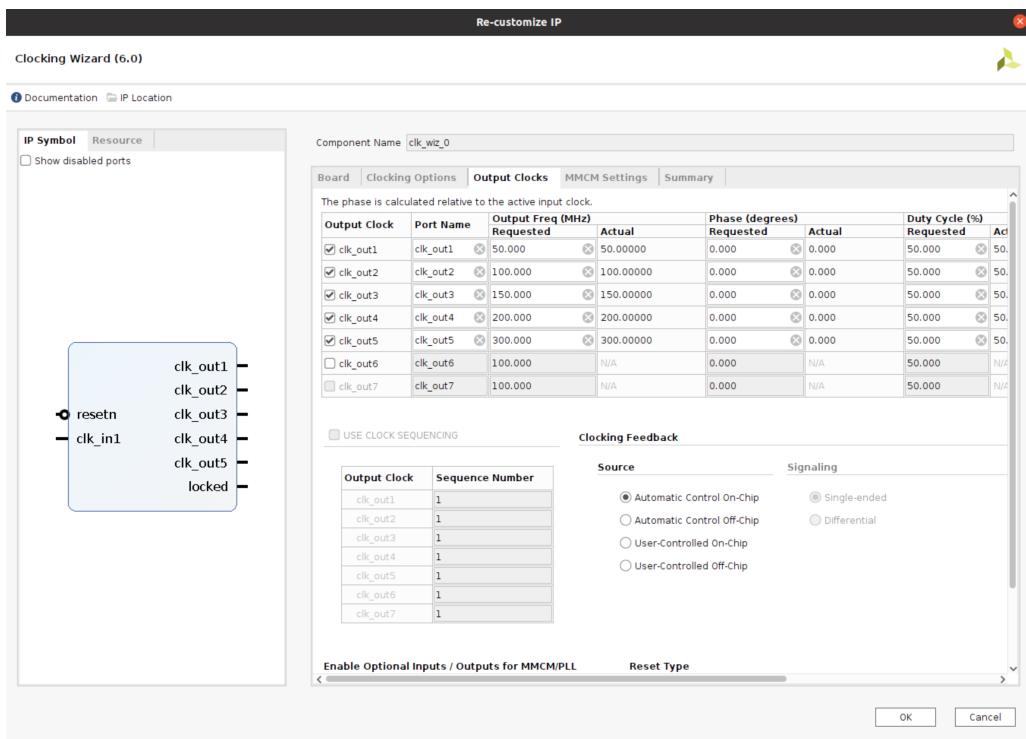
ZynQ je zobrazeno na obr. 6 - 5.



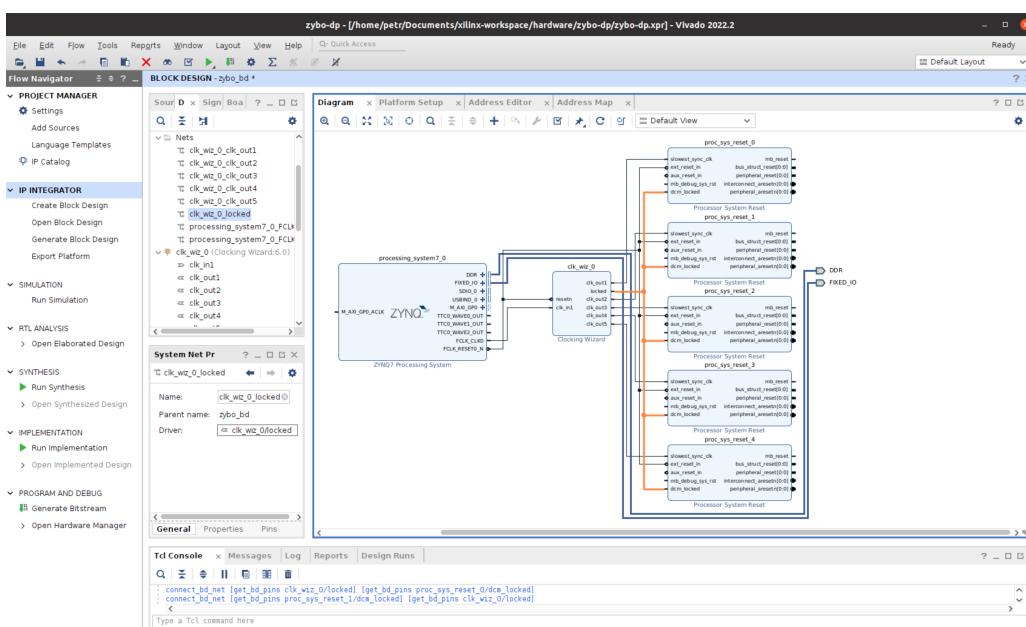
Obr. 6 - 5 Xilinx Vivado – vložení bloku ZYNQ7 Processing System.

Poté je pro funkční akcelerované aplikace nutné vložit do designu blok *Clocking Wizard*, ve kterém nastavit v záložce *Output Clocks*, aby byl signál aktivní v 0 a aktivovat pět výstupních signálů *Clock*, ve kterých nastavit požadovanou taktovací frekvenci na 50, 100, 150, 200 a 300 Hz. Poté je nutné na výstup bloku *ZYNQ7 Processing System* se jménem *FCLK\_CLK0* připojit vstup *clk\_in1* a k výstupu *FCLK\_RESET0\_N* vstup *resetn*.

Po nastavení bloku *Clocking Wizard* je zapotřebí do designu vložit pět bloků *Processor System Reset*. Následně propojit odpovídající výstupy bloku *Clocking Wizard* s názvem *clk\_outX*, kde *X* značí pořadí výstupního signálu, s odpovídajícími bloky *Processor System Reset* a jejich vstupy *slowest\_sync\_clk*. Ke všem vstupům *dcm\_locked* bloků *Processor System Reset* připojit výstup *locked Clocking Wizard*. A konečně ke všem vstupům *ext\_reset\_in* připojit výstup *FCLK\_RESET0\_N* ZynQ bloku. Představené propojení jednotlivých bloků je možné pozorovat na obr. 6 - 7.



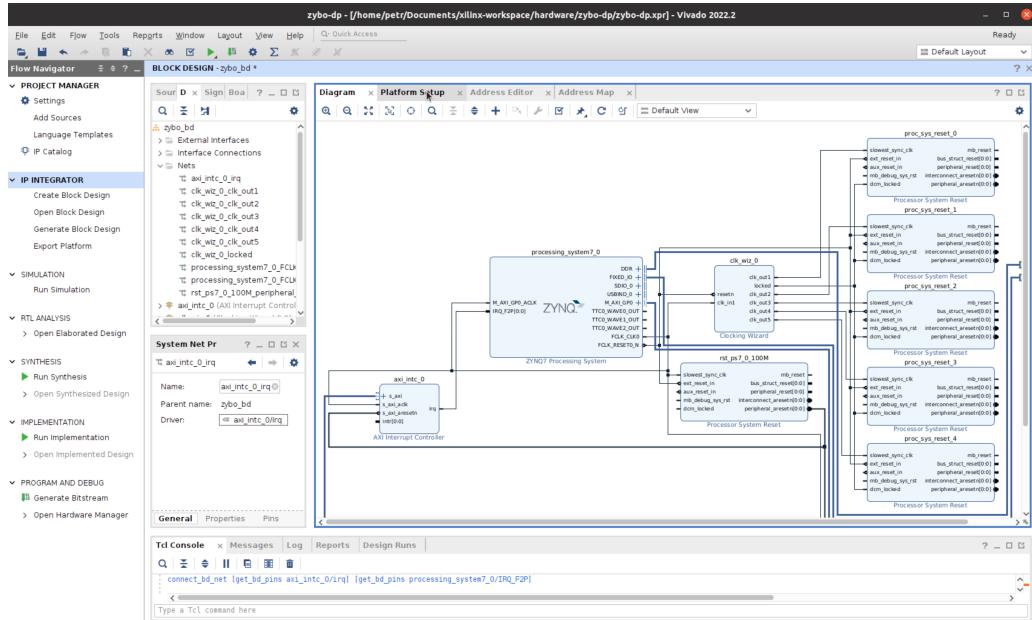
Obr. 6 - 6 Xilinx Vivado – nastavení výstupních taktovacích signálů.



Obr. 6 - 7 Xilinx Vivado – propojení bloků taktování.

Po úspěšném propojení bloků je vhodné otevřít nastavení *ZYNQ7 Processing System* a v záložce *Interrupts* povolit nastavení *Fabric Interrupts/PL-PS Interrupt Ports/IRQ\_F2P* přerušení. Následně do designu je nutné vložit další blok, řídící přerušení, se jménem *AXI Interrupt Controller*, otevřít jeho nastavení a v sekci *Processor Interrupt Type and Connection* změnit nastavení *Interrupt Output Connection* z *Bus* na *Single*. Následně je opět možné spustit automatické propojení jednotlivých bloků s výchozím nastavením.

Aby byly přerušení funkční, je třeba propojit výstup bloku *AXI Interrupt Controller* se jménem *irq* se vstupem bloku *ZYNQ7 Processing System IRQ\_F2P*. Minimální funkční blokový design je zobrazen na obr. 6 - 8.



Obr. 6 - 8 Xilinx Vivado – minimální funkční blokový design pro akcelerovanou aplikaci.

Nyní je možné přejít ze záložky *Diagram* do záložky *Platform Setup* kde je nutné nastavit určité potřebné konektory a výstupy. V záložce *AXI Port* je nutné pro blok *ZYNQ7 Processing System* nastavit parametry dle tabulky č. 6 - 1.

Tab. 6 - 1 Ukázka nastavených AXI portů v Xilinx Vivado bloku ZYNQ7 Processing System.

Name	Enabled	Memport	SP Tag
M_AXI_GP1	X	M_AXI_GP	-
S_AXI_ACP	O	-	-
S_AXI_HP0	X	S_AXI_HP	HP0
S_AXI_HP1	X	S_AXI_HP	HP1
S_AXI_HP2	X	S_AXI_HP	HP2
S_AXI_HP3	X	S_AXI_HP	HP3

Aby bylo možné zapisovat do globální paměti přes MAXI Adapter je nutné povolit funkci vybraných portů v bloku *AXI Interconnect*. V této práci byly povoleny porty *M01\_AXI* až *M32\_AXI*.

Následně v záložce *Clock* je nutné povolit *clk\_outx*, kde  $x \in \{1, 5\}$ , nastavit jejich odpovídající ID a jako výchozí použít taktovací signál 100 MHz.

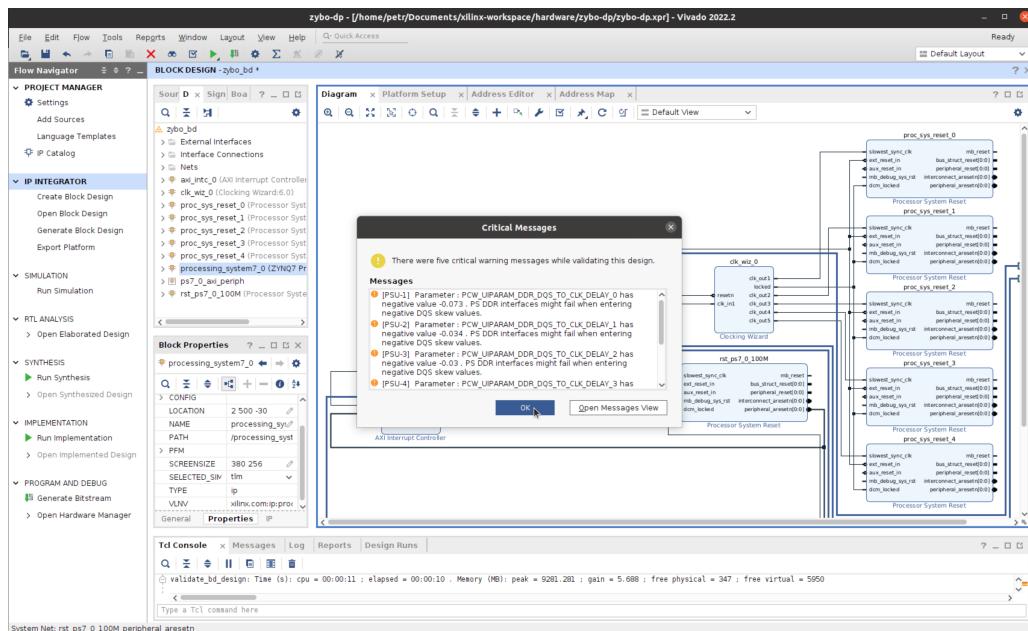
Dále je v záložce *Interrupt* nutné aktivovat výstup *intr* bloku *AXI Interrupt Controller*.

Aby bylo možné případně provádět HW-emulaci, je nutné v kartě *Diagram* zvolit blok *ZYNQ7 Processing System* a v záložce *Block Properties* v nabídce *SELECTED\_SIM\_MODEL* zvolit možnost tlm. [22]

Ovšem v této práci nebylo dosaženo funkční SW ani HW simulace z neznámých důvodů nemožnosti spuštění Emulátoru v prostředí Vitis. Pokud byl emulátor QEMU spuštěn zvlášť pomocí příkazové řádky a přes příkaz byly *scp* přesunuty potřebné soubory, jednalo se pouze o SW simulaci bez připojeného PL a tudíž nebylo možné algoritmy ověřit.

Před dalším pokračováním je možné design validovat ponocí příslušného tlačítka validace na horizontální ovládací liště. Pokud je již HW design vytvořen a nakonfigurován, je možné v kartě *Sources/Design Sources* vybrat vytvořený design a pomocí nabídky pravého tlačítka myši vybrat možnost *Create HDL Wrapper*. Zároveň dochází také k validaci designu. Pokud se vyskytují v designu různá kritická upozornění, která jsou zobrazena například na obr. 6 - 9 je stále možné pokračovat v tvorbě konečného produktu.

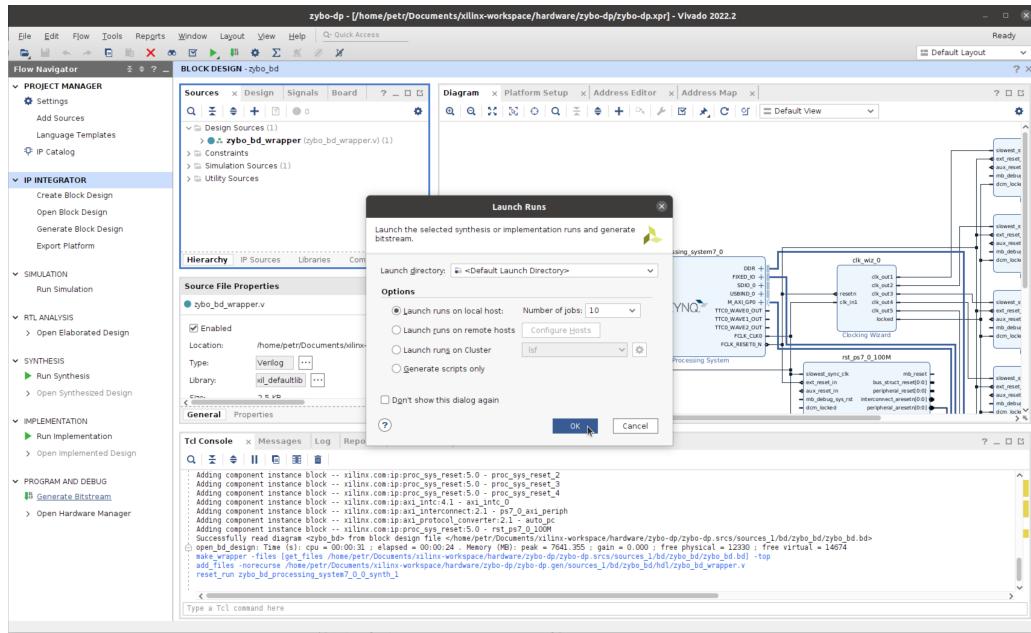
Po vytvoření *HDL Wrapper* je možné v menu *Flow Navigator/Program and Debug* zvolit krok *Generate Bitstream*. Pokud do tohoto kroku nebyla provedena syntéza ani implementace designu, objeví se hlášení, že je třeba tyto kroky provést a v případě pokračování v požadavku generování bitstreamu budou provedeny. Poté je v nabídce možné vybrat, zda procesy budou probíhat lokálně či na vzdáleném serveru nebo clusteru. Také je možné zvolit, kolik výpočetních jader procesoru se bude podílet na prováděných úkolech. V případě využívání osobního počítače autor práce doporučuje používat méně než polovinu dostupných jader. Tato volba vychází z experimentálního zjištění, že v případě využívání vyššího počtu jader může dojít k neočekávané chybě a proces provádění úkolů bude bez vypsání jakékoliv informace ukončen a proces syntézy, implementace a generace bitstreamu bude nutné spustit a provést znova. Ukázka nastavení procesu je zobrazena na obr. 6 - 10.



Obr. 6 - 9 Xilinx Vivado – kritická upozornění vzniklá po validaci designu, která je možné ignorovat.

Indikátor provádění jednotlivých procesů je umístěn v pravém horním rohu. Záznam prováděných procesů je poté umístěn v kartě *Log*.

Po úspěšném provedení jednotlivých kroků je zobrazena nabídka, která umožňuje nahlédnout na vy-



Obr. 6 - 10 Xilinx Vivado – nastavení provádění úkonů syntézy, implementace a generování bitstreamu, volba použitých výpočetních jader a určení, kde se mají procesy vykonávat.

tvořený design. Tuto nabídku je možné zavřít aniž by byla vykonávána jakákoli z nabízených možností.

Po ukončení procesů je pro možné použití vytvořeného designu pro tvorbu PetaLinux systému a aplikací v Xilinx Vitis nutné exportovat vytvořenou platformu. Export je proveden pomocí sekvence tlačítek *File/Export/Export platform*. Ve výběru platformy je výhodné zvolit možnost *Hardware and hardware emulation*, která umožňuje použít design pro skutečný HW i jeho emulaci. V nabídce *Platform State* je nutné vybrat možnost *Pre-Synthesis*, která umožňuje další zpracování aplikace v Xilinx Vitis pomocí V++. Důležitou volbou je zvolení možnost *Include bistream*, který byl produktem tvorby designu v této části.

Po nastavení dodatečných informací platformy je možné jej vyexportovat do požadované lokace, kde bude dále využívána.

### 6.3 Tvorba PetaLinux

### 6.4 Tvorba SW pro CPU a FPGA

## 7 Představení pracoviště

## 8 Dosažené výsledky

## Závěr

Aliquam dapibus leo velit, ultrices eleifend mi feugiat eget. Aliquam euismod facilisis turpis, nec lobortis libero aliquet sit amet. Aenean suscipit ante eget ipsum viverra hendrerit. Ut sed massa sed nisi tempus dapibus in eu enim. Nullam vitae odio laoreet, malesuada purus non, faucibus orci. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eget odio quis enim laoreet imperdiet nec eu nunc. Maecenas ut consequat purus. Duis faucibus risus nec metus cursus placerat. Phasellus sapien justo, laoreet in pulvinar ut, maximus nec velit.

## Literatura

- [1] SASS, Ronald; SCHMIDT, Andrew G. *Embedded systems design with platform FPGAs: principles and practices*. Boston: Morgan Kaufmann, 2010. ISBN 0123743338.
- [2] ANDINA, Juan J. R.; TORRE ARNANZ, Eduardo de la; VALDÉS PEÑA, María D. *FPGAs: Fundamentals, Advanced Features, and Applications in Industrial Electronics*. 1. vyd. Bosa Roca: CRC Press, 2017;2015; ISBN 9781439896990;1439896992;
- [3] What Is Accelerated Computing, and Why Is It Important? In: *Xilinx, Inc.* [Online]. [B.r.] [cit. 2022-11-03]. Dostupné z: <https://www.xilinx.com/applications/adaptive-computing/what-is-accelerated-computing-and-why-is-it-important.html>.
- [4] PANG, Aiken; MEMBREY, Peter; SLUŽBA), SpringerLink (online. *Beginning FPGA: Programming Metal: Your brain on hardware: Programming Metal*. Berkeley, CA: Apress, 2017;2016; č. Book, Whole. Dostupné také z: <https://go.exlibris.link/JzysQtpz>.
- [5] Sphery vs. shapes, the first raytraced game that is not software. In: *GitHub* [online]. 14. 07. 2022 [cit. 2022-11-06]. Dostupné z: <https://github.com/JulianKemmerer/PipelineC-Graphics/blob/main/doc/Sphery-vs-Shapes.pdf>.
- [6] Ray tracing (graphics). In: *Wikipedia* [online]. 06. 11. 2022 [cit. 2022-11-06]. Dostupné z: [https://en.wikipedia.org/wiki/Ray\\_tracing\\_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics)).
- [7] GROVER, Naresh; K.SONI, M. Reduction of Power Consumption in FPGAs - An Overview. *International journal of information engineering and electronic business*. 2012, roč. 4, č. 5, s. 50–69. ISBN 2074-9023.
- [8] Amazon EC2 F1 Instances. In: *Amazon AWS* [online]. [B.r.] [cit. 2022-11-06]. Dostupné z: <https://aws.amazon.com/ec2/instance-types/f1/>.
- [9] VANDERBAUWHEDE, Wim; BENKRID, Khaled. *High-Performance Computing Using FPGAs*. Springer Publishing Company, Incorporated, 2013. ISBN 1461417902.
- [10] RODRÍGUEZ-ANDINA, Juan J.; VALDÉS-PEÑA, María D.; MOURE, María J. Advanced Features and Industrial Applications of FPGAs—A Review. *IEEE Transactions on Industrial Informatics*. 2015, roč. 11, č. 4, s. 853–864. Dostupné z DOI: 10.1109/TII.2015.2431223.
- [11] Hardware-in-the-Loop (HIL) Simulation. In: *The MathWorks, Inc.* [Online]. [B.r.] [cit. 2022-11-06]. Dostupné z: <https://www.mathworks.com/discovery/hardware-in-the-loop-hil.html>.
- [12] NAOUAR, Mohamed-Wissem; MONMASSON, Eric; NAASSANI, Ahmad Ammar; SLAMA-BELKHODJA, Ilhem; PATIN, Nicolas. FPGA-Based Current Controllers for AC Machine Drives—A Review. *IEEE Transactions on Industrial Electronics*. 2007, roč. 54, č. 4, s. 1907–1925. Dostupné z DOI: 10.1109/TIE.2007.898302.
- [13] DIGILENT, Inc. Zybo. In: *Digilent Documentation* [online]. [B.r.] [cit. 2022-11-11]. Dostupné z: <https://digilent.com/reference/programmable-logic/zybo/start>.
- [14] DIGILENT, Inc. Zybo Z7 Migration Guide. In: *Digilent Documentation* [online]. [B.r.] [cit. 2022-11-11]. Dostupné z: <https://digilent.com/reference/programmable-logic/zybo-z7/migration-guide>.

- [15] XILINX, Inc. Zynq-7000 SoC Technical Reference Manual. In: *Xilinx Documentation* [online]. 02. 04. 2021 [cit. 2022-11-11]. Dostupné z: <https://docs.xilinx.com/v/u/en-US/ug585-Zynq-7000-TRM>.
- [16] DIGILENT, Inc. Zybo Reference Manual. In: *Digilent Documentation* [online]. [B.r.] [cit. 2022-11-11]. Dostupné z: <https://digilent.com/reference/programmable-logic/zybo/reference-manual>.
- [17] XILINX, Inc. Downloads. In: *AMD Xilinx Downloads* [online]. [B.r.] [cit. 2022-11-19]. Dostupné z: <https://www.xilinx.com/support/download.html>.
- [18] XILINX, Inc. Vitis Unified Software Platform Documentation: Application Acceleration Development (UG1393). In: *AMD Xilinx Documentation Portal* [online]. [B.r.] [cit. 2022-11-18]. Dostupné z: <https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration/>.
- [19] XILINX, Inc. Downloads. In: *AMD Xilinx PetaLinux Tools* [online]. [B.r.] [cit. 2022-11-19]. Dostupné z: <https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html>.
- [20] XILINX, Inc. Vivado Design Suite User Guide: Release Notes, Installation, and Licensing (UG973). In: *AMD Xilinx Documentation Portal* [online]. [B.r.] [cit. 2022-11-18]. Dostupné z: <https://docs.xilinx.com/r/en-US/ug973-vivado-release-notes-install-license/>.
- [21] XILINX, Inc. PetaLinux Tools Documentation: Reference Guide (UG1144). In: *AMD Xilinx Documentation Portal* [online]. [B.r.] [cit. 2022-11-18]. Dostupné z: <https://docs.xilinx.com/r/en-US/ug1144-petalinux-tools-reference-guide>.
- [22] HOSSEINABADY, Mohammad. Vitis 2021.1 Embedded Platform for Zybo-Z7-20, 2018. In: *Hackster.io, an Avnet Community* [online]. 16. 08. 2021 [cit. 2022-11-26]. Dostupné z: <https://www.hackster.io/mohammad-hosseinabady2/vitis-2021-1-embedded-platform-for-zybo-z7-20-d39e1a>.
- [23] DIGILENT, Inc. Vivado Board Files for Digilent FPGA Boards, 2022. In: *GitHub.io* [online]. 25. 03. 2022 [cit. 2022-11-26]. Dostupné z: <https://github.com/Digilent/vivado-boards>.
- [24] DIGILENT, Inc. Installing Vivado, Vitis, and Digilent Board Files. In: *Digilent Reference* [online]. [B.r.] [cit. 2022-11-26]. Dostupné z: <https://digilent.com/reference/programmable-logic/guides/installing-vivado-and-vitis>.
- [25] XILINX, Inc. SoCs with Hardware and Software Programmability. In: *Xilinx Website* [online]. [B.r.] [cit. 2022-11-11]. Dostupné z: <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>.
- [26] DIGILENT, Inc. Zybo Petalinux BSP Project, 2018. In: *GitHub.io* [online]. 29. 03. 2018 [cit. 2022-11-26]. Dostupné z: <https://github.com/Digilent/Petalinux-Zybo>.

## **Příloha A: Seznam symbolů a zkratek**

### **A.1 Seznam symbolů**

$\vec{F}$  (N) vektor síly

### **A.2 Seznam zkratek**

DCM DC Master