VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2017 Bc. David Oczka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Dávkové zpracování biomedicínských výpočtů na superpočítači Rivyera

Batch Processing of Biomedical Calculations on a Rivyera Supercomputer

2017 Bc. David Oczka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zadání diplomové práce

Student: **Bc. David Oczka**

Studijní program: N2649 Elektrotechnika

Studijní obor: 3901T009 Biomedicínské inženýrství

Téma: Dávkové zpracování biomedicínských výpočtů na superpočítači Rivyera
Batch Processing of Biomedical Calculations on a Rivyera Supercomputer

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Vypracování přehledu používaných technik pro masivně paralelní výpočty s obvody FPGA.
2. Seznámení se s architekturou superpočítače Rivyera.
3. Návrh a realizace řešení vzdálené obsluhy a dávkového zpracování biomedicínských dat na superpočítači Rivyera.
4. Experimentální ověření funkce a provedení výkonových testů.
5. Zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

[1] WIENBRANDT, Lars. The FPGA-based high-performance computer RIVYERA for applications in bioinformatics. In: *Language, Life, Limits: Proceedings 10th Conference on Computability in Europe, CiE 2014.* New York, NY, USA: Springer, 2014, pp. 383–392. ISBN 978-3-319-08018-5.
[2] EIJKHOUT, Victor. *Introduction to High Performance Scientific Computing.* Lulu. com, 2014. ISBN 978-1257-99254-6.
[3] PARNELL, Karen and Nick MEHTA. *Programmable Logic Design Quick Start Handbook.* 4th ed. [s.l.]: Xilinx Inc., 2003. 225 s. Dostupné z: http://www.xilinx.com/univ/beginnersbookjune2003ver2.pdf.
[4] ASHENDEN, Peter J. *The designer's guide to VHDL.* 2nd ed. San Francisco: Morgan Kaufmann, 2002. ISBN 1-55860-674-2.
[5] KAŠÍK, Vladimír. *Programování hradlových polí.* Učební text a návody do cvičení. Ostrava: VŠB-TU Ostrava, 2012.

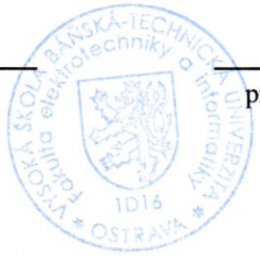
Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Vladimír Kašík, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Ing. Jiří Koziorek, Ph.D.	prof. RNDr. Václav Snášel, CSc.
vedoucí katedry	*děkan fakulty*



Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

David Oczka

Datum odevzdání diplomové práce: 28.4.2017

Poděkování

Rád bych poděkoval panu Ing. Vladimíru Kašíkovi, Ph.D. za cenné rady, konzultace a odborný dohled při vedení mé diplomové práce.

Abstrakt

Práce pojednává o problematice dávkového zpracování dat na hardwarové architektuře hradlových polí zastoupené superpočítačem Rivyera, jehož výpočetní výkon je zprostředkován několika desítkami programovatelných hradlových polí. Cílem práce je navrhnout a realizovat řešení vzdálené obsluhy a dávkové zpracování dat na superpočítači Rivyera. A dále také poskytnout postup pro vývoj a řešení výpočtů s použitím paralelního přístupu na superpočítači Rivyera. Výpočetní postupy jsou řešeny kombinací výpočetního jádra na programovatelných hradlových polích, popsaných v jazyce VHDL, a obslužné hostitelské aplikace v programovacím jazyce Java. Ukázkou výpočetních postupů s použitím superpočítače Rivyera je paralelní zpracování CT obrazů do podoby digitálně rekonstruovaného rentgenového snímku z určité směrové projekce.

Klíčová slova

FPGA, Rivyera, VHDL, Java, Zpracování obrazu, CT, DRR

Abstract

This thesis describes batch processing of data at hardware architecture of gate arrays represented by Rivyera supercomputer, its computational power is mediated by several dozens of field programmable gate arrays. The aim of thesis is to design and implement solution of remote control and batch processing of data at Rivyera supercomputer and also to provide a method to development and resolution of calculation by using parallel computing of Rivyera supercomputer. Computational methods are solved by combining of computational core of field programmable gate arrays, described using VHDL language, and handler host application, written in Java programming language. An example of computational methods using Rivyera supercomputer is parallel processing of CT images into digitally reconstructed radiographs from a certain directional projection.

Key words

FPGA, Rivyera, VHDL, Java, Image processing, CT, DRR

Obsah

Seznam použitých zkratek

AAR	Automatic Artifact Removal / Automatické odstranění artefaktů
API	Application Programming Interface / Aplikační programové rozhraní
CMT	Compare Match Timer
CSS	Cascading Style Sheets / Kaskádové styly
CT	Computed Tomography / Počítačová tomografie
DICOM	Digital Imaging and Communications in Medicine
DRAM	Dynamic Random Access Memory
DRR	Digitally Reconstructed Radiograph / Digitálně rekonstruovaný snímek
EEG	Elektroencefalografie
EEPROM	Electrically Erasable Programmable Read-Only Memory
EKG	Elektrokardiografie
FPGA	Field Programmable Gate Array / Programovatelná hradlová pole
FPLA	Field Programmable Logic Array / Programovatelná logická pole
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment / Vývojové prostředí
JAR	Java Archive
JVM	Java Virtual Machine
LUT	Look-Up Table / Programovatelná logická (vyhledávací) tabulka
MRI	Magnetic Resonance Imaging / Magnetická rezonance
MVC	Model-view-controller
PHP	PHP: Hypertext Preprocessor
PLD	Programmable Logic Device / Programovatelné logické zařízení
PNG	Portable Network Graphics
PROM	Programmable Read Only Memory / Programovatelná permanentní paměť
RAM	Random Access Memory
RGB	Red Green Blue
RTG	Rentgen
SO	Shared Object
URL	Uniform Resource Locator / Jednotná adresa zdroje
USG	Ultrasonografie
VHDL	VHSIC Hardware Description Language / Jazyk pro popis VHSIC hardwaru
VHSIC	Very High Speed Integrated Circuit / Velmi rychlý integrovaný obvod

Seznam ilustrací

Seznam tabulek

Seznam úseků zdrojového kódu

Úvod

Superpočítač Rivyera je komerčně dostupnou, velmi výkonnou kombinací programovatelných hradlových polí a jejich obsluhy pomocí softwarové aplikace napsané v programovacích jazycích C nebo Java. Architektura superpočítače se skládá z periférie se sloty pro FPGA karty, které umožňují zapojit až 16 karet. Každá z karet obsahuje 8 FPGA, celkově tedy až 128 FPGA, které uživatel může použít pro paralelní zpracování. Vývoj i komunikace se superpočítačem Rivyera probíhá na několika vrstvách, z nichž každá vyžaduje jiný přístup. Pro vývoj i spouštění vytvořených řešení je využíváno aplikačních programovacích rozhraní (API), která jsou dodána výrobcem.

Cílem práce je vytvořit postupy pro vývoj a řešení biomedicínských výpočtů s použitím paralelního přístupu na superpočítači Rivyera. Jako ukázková úloha řešení biomedicínských výpočtů prostřednictvím použití paralelního přístupu je vypracována aplikace pro paralelní zpracování obrazu, která ze vstupních CT snímků vytváří digitálně rekonstruovaný radiogram (DRR).

1 Dávkové výpočty v biomedicíně

V biomedicíně, a obecně biologii, je mnoho problémů, k jejichž řešení je nutno použít matematické operace, které mohou zahrnovat velice složité výpočty. Většinou se jedná o výpočty numerické, jejichž operandy mají pohyblivou řadovou čárku. Takovéto operace jsou často prováděny na superpočítačích, protože běžná výpočetní technika nemá pro jejich výpočet dostatečný výkon. Dalším úskalím u těchto výpočetních problémů jsou kromě vysoké složitosti výpočtů také obrovské objemy dat, které skrze rovnice a matematické modely protékají. [1][2]

Pro představu o objemech dat, které se v biomedicíně běžně vyskytují, je možné uvést například EEG, u kterého je snímán signál až z 256 elektrod po dobu několika minut až hodin. Dalším příkladem může být rekonstrukce obrazů z rentgenu či magnetické rezonance, při takových rekonstrukcích je potřeba vzít v úvahu celou sérii obrazů, která obsahuje i několik stovek snímků.

Dalším aspektem těchto výpočtů jsou jejich časové nároky na dobu zpracování výsledků, které často bývají vyžadovány takřka v reálném čase. Při uvážení jak velké jsou objemy dat a jak náročné operace jsou s nimi vyžadovány, je zapotřebí opravdu vysokého výkonu, který není možné dodat běžnými výpočetními zařízeními.

1.1 Paralelní přístup ke zpracování výpočtů

Značné zrychlení výpočtů přináší paralelní přístup ke zpracování, což znamená, že výpočetní problém je rozdělen do několika menších úloh, ať už stejného či různého výpočetního postupu a tyto menší úlohy jsou pak zpracovávány na hardwaru, který má k dispozici více procesorů či jiných výpočetních jednotek, jako například grafické procesory. Nejčastěji se paralelizmus využívá u výpočetních superpočítačů, které mají k dispozici desítky až stovky procesorů a fungují na principu počítačového clusteru, což je volné spojení několika počítačů, které se díky síťovému propojení tváří jako jeden celek. Tento přístup tedy nabízí úsporu času zajištěnou tím, že si jeden úkol rozdělí několik výpočetních jednotek. Ovšem tento postup nemusí být vždy výhodným řešením. Příprava problému do tvaru, který je možný paralelizovat může mnohdy zabrat delší dobu, než čas, za který je možné problém vyřešit s jedinou výpočetní jednotkou. [1][2]

Výpočetní jednotky však nemusí být pouze počítačové procesory, které obsluhuje operační systém, ale může se také jednat o různá zařízení bez operačního systému (bare metal) nebo různé periferní digitální obvody, které mají svůj specifický přístup k výpočtům.

1.2 Přehled výpočetně náročných biomedicínských výpočtů

Vzhledem k množství možných náročných biomedicínských či biologických výpočtů, bude tato kapitola zpracována jako rešeršní práce na téma biomedicínských výpočtů s využitím programovatelných hradlových polí. Toto téma je zvoleno, protože přístup k řešení problémů s využitím superpočítače Rivyera je taktéž postaven na principu využití programovatelných hradlových polí. Rešerše se bude zaměřovat na některé z výpočtů, které je možné na poli biomedicíny řešit na programovatelných hradlových polích.

1.2.1 Rešeršní práce

Téma práce:	Biomedicínské výpočty s využitím FPGA
Topic:	Biomedical computation using FPGA
Příbuzné termíny a synonyma:	Medicínské výpočty, Biomedicínské kalkulace, Výpočty s použitím FPGA
Related terms or synonyms:	Medical computations, Biomedical calculations, Computations using FPGA
Časové hledisko:	od roku 2010 do 2016
Jazykové vymezení:	Angličtina
Typ dokumentu:	Časopisecké články, příspěvky ve sborníku
Nadřazené obecnější termíny:	Biomedicína, Medicína, FPGA
Broader terms:	Biomedicine, Medicine, PFGA
Podřazené podrobnější termíny:	Elektrokardiografie, Elektroencefalografie, Ultrasonografie, Počítačová tomografie, Magnetická rezonance, Programovatelná hradlová pole
Narrower terms:	Electrocardiography, Electroencephalography, Ultrasonography, Computed tomography, Magnetic Resonance Imaging, Field Programmable Gate Arrays
Zdroje vyhledávání:	Databáze IEEE Xplore Digital Library (8)
Sources of searching:	Database IEEE Xplore Digital Library (8)
Počet článků:	8
Number of articles:	8
Klíčové slova:	Biomedicínské výpočty, EKG, EEG, USG, CT, FPGA
Keywords:	Biomedical computation, ECG, EEG, USG, CT, FPGA

1.2.1.1 Cíl rešerše

Tato rešeršní práce si klade za cíl nastínit přehled výpočetně náročných biomedicínských výpočtů, které jsou řešeny s využitím programovatelných hradlových polí. Zpracované články se zabývají převážně zpracováním dat z oblasti ultrasonografie, elektrokardiografie, elektroencefalografie, magnetické rezonance a počítačové tomografie.

1.2.1.2 Články

1.2.1.2.1 FPGA Implementation of EEG System-on-Chip with Automatic Artifacts Removal based on BSS-CCA Method

Bibliografický záznam

CHOU, Chia-Ching, Tsan-Yu CHEN a Wai-Chi FANG. FPGA implementation of EEG system-on-chip with automatic artifacts removal based on BSS-CCA method. In: *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)* [online]. IEEE, 2016, s. 224-227 [cit. 2017-03-29]. DOI: 10.1109/BioCAS.2016.7833772. ISBN 978-1-5090-2959-4. Dostupné z: http://ieeexplore.ieee.org/document/7833772/

Úvod

Tento článek popisuje možnosti automatického odstranění artefaktů z EEG signálu s použitím metody BSS-CCA implementovanou na technologii programovatelných hradlových polí.

Obsah

Článek se zabývá popisem systému pro automatické odstranění svalových artefaktů, způsobených elektrickou aktivitou svalstva, z vícekanálového elektroencefalografického signálu s využitím metody separace neurčených zdrojů, která je založena na kánonické korelační analýze (BSS-CCA). EEG signál může být velmi snadno narušen jinými elektrickými vlivy, protože je velice slabý. Tento signál může být použit pro aplikace tvořící rozhraní mozek-stroj, avšak pro tento záměr musí být signál zbaven všech artefaktů. Dalším předpokladem pro tato rozhraní je filtrace (a obecně předzpracování) signálu v reálném čase. Nejvýznamnějším artefaktem u EEG signálu je artefakt vznikající díky mrkání očí, který musí být odstraněn ať už v zájmu správné funkcionality rozhraní mozek-stroj, ale obecně pro správnou interpretaci EEG signálů. Dalším významným svalovým artefaktem je kousání či zatínání čelisti. Oba zmíněné artefakty je systém, popsaný v tomto článku, schopen odstranit.

K získání surového elektroencefalografických signálů bylo použito komerčního systému NeuroScan^{TM}. Všechna data byla naměřena povrchovými (skalpovými) elektrodami s rozložením v mezinárodním systému 10-20 a s vzorkovací frekvencí 128 Hz. Použitý hardware používá početní operace s pevnou čárkou jako náhradu za operace s plovoucí čárkou, což vylepšuje jeho výkon a také snižuje cenu. Práce popisuje algoritmus implementující metodu BSS-CCA a pro ověření výsledků z této metody byl použit systém MATLAB a jeho toolbox pro automatické odstranění artefaktu (AAR).

Tabulka I porovnává výsledné korelační koeficienty s pevnou čárkou pro metodu BSS-CCA implementovanou na programovatelných hradlových polích a korelační koeficienty s plovoucí čárkou z toolboxu systému MATLAB pro automatické odstranění artefaktů.

Tabulka I – Korelační koeficienty pro hardwarový návrh metody BSS-CCA (označeno jako pevná čárka) a AAR toolbox systému MATLAB

Artefakt	Metoda	Subjekt 1	Subjekt 2	Subjekt 3	Subjekt 4
Mrkání očí	Pevná čárka	0,8927	0,8690	0,8476	0,9364
	AAR	0,8765	0,8946	0,8437	0,8718
Kousání	Pevná čárka	0,9365	0,9510	0,9431	0,9412
	AAR	0,8990	0,9351	0,9245	0,9260

Závěr článku ukazuje, že výsledky hardwarového návrhu s pevnou čárkou, týkající se artefaktů mrkání očí a kousání, jsou velmi blízké a dokonce i lepší než průměr výsledků z toolboxu systému MATLAB pro automatické odstranění artefaktů. Výsledky ukazují, že tento systém má dostatečné kvality, aby mohl být použit pro aplikace rozhraní mozek-stroj, které vyžadují zpracování v reálném čase.

1.2.1.2.2 FPGA-based Digital Filter Design for Biomedical Signal

Bibliografický záznam

OZPOLAT, Erman, Baris KARAKAYA, Turgay KAYA a Arif GULTEN. FPGA-based digital Filter Design for Biomedical Signal. In: *2016 XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH)* [online]. IEEE, 2016, s. 70-73 [cit. 2017-03-29]. DOI: 10.1109/MEMSTECH.2016.7507523. ISBN 978-617-607-913-2. Dostupné z: http://ieeexplore.ieee.org/document/7507523/

Úvod

Článek se zabývá implementací filtru s konečnou impulsní odezvou pro filtraci elektroencefalografického signálu s využitím technologie programovatelných hradlových polí.

Obsah

Práce popisuje implementaci filtru s konečnou impulsní odezvou na hardwarové technologii programovatelných hradlových polí, kde je pro výpočet koeficientů filtru použito komerčního výpočetního systému MATLAB.

Pro simulace filtrace bylo použito EEG signálu, který byl zaznamenán Mezinárodní federací klinické neurofyziologie. Pro hardwarový návrh na FPGA bylo použito jazyku Verilog. V systému MATLAB byly spočítány koeficienty pro filtr 4. řádu s typem dolní propust, které byly následně použity v hardwarovém návrhu při filtraci EEG signálu. Hardwarový návrh pracuje se znaménkovými čísly s pevnou čárkou o bitové délce 16 bitů.

Implementovaný hardwarový filtr získává data po částech z prostředí systému MATLAB. Každá část má 14 vstupů. Data jsou vyfiltrována skrze filtr s danými koeficienty a navrácena do systému MATLAB, skrze který jsou vizualizována.

Závěr práce zmiňuje, že použití programovatelných hradlových polí umožňuje zpracovat EEG signál skrze filtr rychleji. Filtr zpracovává signál v reálném čase a může být v reálném čase také upravován. Dále je také uvedeno, že systém provádí všechny operace s pevnou čárkou. Aritmetika čísel s pevnou čárkou je trochu přesnější než aritmetika čísel s plovoucí čárkou, ovšem sčítání a násobení čísel s pevnou čárkou zanáší do výsledků tohoto systému nechtěné numerické chyby. Tento filtr může být v budoucnu použit pro jakýkoliv biomedicínský signál, avšak EEG signál má pro tento filtr výhodnou frekvenční charakteristiku.

1.2.1.2.3 FPGA-Based Detection of QRS Complexes in ECG Signal

Bibliografický záznam

HASSEN, Amina El, Aymeric HISTACE, Mehdi TEROSIET a Olivier ROMAIN. FPGA-based detection of QRS complexes in ECG signal. In: *2015 Conference on Design and Architectures for Signal and Image Processing (DASIP)* [online]. IEEE, 2015, s. 1-7 [cit. 2017-03-29]. DOI: 10.1109/DASIP.2015.7367244. ISBN 978-1-4673-7738-6. Dostupné z: http://ieeexplore.ieee.org/document/7367244/

Úvod

Tento článek se zabývá detekcí QRS komplexů v elektrokardiografickém signálu s použitím algoritmu implementovaného na programovatelných hradlových polích.

Obsah

Článek popisuje algoritmus pro automatickou detekci QRS komplexů v EKG signálu implementovaný na technologii programovatelných hradlových polí. Algoritmus je rozdělen do tří hlavních částí: filtrace, zvýraznění kontrastu a detekce. Detekce je založena na časových a amplitudových mezích zvýrazněných dat.

Úspěšnost algoritmu byla otestována na 48 záznamech 30minutových EKG signálů z databáze MIT/BIH Arrhythmia, kdy algoritmus implementovaný na programovatelných hradlových polích úspěšně detekoval 95,35 % QRS komplexů s falešnou detekcí pouhých 8 %. Práce dále porovnává úspěšnost algoritmu s úspěšností jiných algoritmů a popisuje jejich principy.

V dalším popisu detekčního algoritmu jsou blíže popsány jednotlivé části. Filtrační část je zastoupena filtrem typu pásmová propust, která má za cíl odstranit v nízkých frekvencích vlnu P a T a ve vysokých frekvencích elektromyografická rušení. Částí pro zvýraznění kontrastu se rozumí zesilovač QRS komplexu, který má zajistit potlačení falešných detekcí. Princip tohoto specifického zesilovače je derivace signálu, umocnění každého bodu na druhou a následně integrace, přičemž se ze tří vrcholů stane pouze jeden. Tento signál je pak v detekční části podroben hledání maxima na základě amplitudových mezí a v případě, že QRS komplex není nalezen v 166% periodě RR* (průměr posledních pěti period RR) je upotřebena časová mez, která snižuje amplitudovou mez 10 krát.

Hardwarový návrh byl implementován s použitím systému MATLAB a otestován s použitím systému ModelSim. Veškeré algoritmy byly otestovány s použitím technologie programovatelných hradlových polí. Výsledná nejlepší úspěšnost při ignorování tří záznamů, které měly úspěšnost značně horší, je 98,82 %, což je nejlepší výsledek s použitím FPGA vůbec (při srovnání s uvedenými úspěšnostmi jiných algoritmů).

1.2.1.2.4 A simple FPGA system for ECG R-R interval detection

Bibliografický záznam

ABDULLAH, Hadeel N. a Bassam H. ABD. A simple FPGA system for ECG R-R interval detection. In: *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)* [online]. IEEE, 2016, s. 1379-1382 [cit. 2017-03-30]. DOI: 10.1109/ICIEA.2016.7603800. ISBN 978-1-4673-8644-9. Dostupné z: http://ieeexplore.ieee.org/document/7603800/

Úvod

Tento článek popisuje systém pro detekci R-R intervalu v elektrokardiografickém signálu implementovaném na programovatelných hradlových polích.

Obsah

Tento systém implementovaný na programovatelných hradlových polích identifikuje časový okamžik, ve kterém se vyskytuje právě vrchol vlny R. Systém si klade za cíl také detekci srdečních arytmií. Hardwarový návrh byl implementován s použitím jazyku VHDL a byl otestován s použitím systému MATLAB. K testování byly použity elektrokardiografické záznamy z databáze MIT/BIH Arrhythmia.

Tabulka I – Výsledky detekce

Data set	Počet úderů	Průměr nedetekovaných úderů %	Úspěšnost %
100	370	0	100
103	355	0	100
113	298	0,34	99,66
201	437	0,46	99,54
212	456	1,09	98,91
205	445	1,34	98,66
221	404	0,5	99,5
231	288	0,69	99,31
234	453	0,44	99,56

Obecná počítačová analýza EKG signálu se sestává ze dvou částí. První část zajišťuje měření specifických vlastností EKG křivky a druhá část zajišťuje detekci arytmií. Tento FPGA systém se

skládá ze tří částí. Předzpracování, extrakce vlastností a klasifikace. Vlna R je detekována poměrně jednoduchým způsobem. Jsou analyzovány vždy tři vzorky. Pokud je druhý vzorek větší než první a zároveň větší než třetí a zároveň je druhý vzorek také větší než 50 % maxima signálu, jedná se o vlnu R. Při každém detekování vlny R je vypočtena frekvence srdečního rytmu a také proměnlivost srdečního rytmu.

Závěrem autor uznává, že existují metody detekce QRS komplexu, které se přibližují 100% úspěšnosti, nicméně dodává, že tato práce dokazuje, že takového výsledku je možné dosáhnout i s jednoduchým, spolehlivým a levným návrhem. Tento systém nedetekuje pouze QRS komplex, ale také počítá interval R-R, srdeční frekvenci a změnu srdečního rytmu. Konečné výsledky jsou uvedeny v tabulce I.

1.2.1.2.5 Architecture Design for Median Filter

Bibliografický záznam

CHATTERJEE, Subarna, Ajoy Kumar RAY, Rezaul KARIM a Arindam BISWAS. Architecture Design for Median Filter. In: *2011 Third National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics* [online]. IEEE, 2011, s. 247-250 [cit. 2017-03-30]. DOI: 10.1109/NCVPRIPG.2011.59. ISBN 978-1-4577-2102-1. Dostupné z: http://ieeexplore.ieee.org/document/6133047/

Úvod

Článek se zabývá implementací mediánového filtru na technologii programovatelných hradlových polí pro zpracování ultrasonografického obrazu.

Obsah

Práce se zabývá implementací mediánového filtru za použití technologie programovatelných hradlových polí k odstranění specifického šumu z ultrasonografického obrazu. Jedná se o šum typu speckle (flíčky), který často postihuje ultrazvukový obraz a degraduje jeho kvalitu a tím i diagnostickou rozlišitelnost. Proto je potřeba tento šum kvalitně odfiltrovat. Cílem této práce bylo vytvořit účinnou techniku za použití mediánového filtru, která tento problém v ultrasonografických obrazech vyřeší.

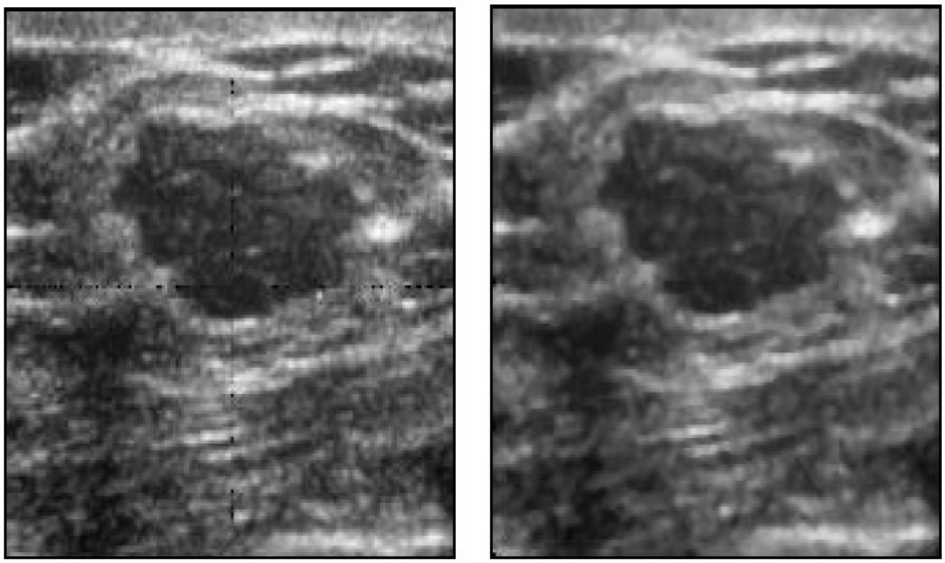
V rámci práce byly vytvořeny záznamy reálných ultrazvukových obrazů, na které byl aplikován algoritmus pro odstranění šumu za použití hardwarové architektury, kterou zastupují programovatelná hradlová pole. Pro vývoj algoritmu bylo použito výpočetního systému MATLAB a pro výslednou implementaci hardwarového návrhu zase jazyka VHDL.

Je zde popsán rychlý třídící algoritmus, který každou novou hodnotu porovnává paralelně se všemi hodnota v seznamu a tím určuje její výslednou pozici v setříděném seznamu, který je základem pro mediánový filtr. Popsaný mediánový filtr má oblast zájmu o rozloze 3x3 pixely.

Výsledná implementace filtru na programovatelných hradlových polích rozděluje vstupní obraz na bloky 16x16 pixelů z důvodů kompatibility s obrazy ve standardu PAL o rozlišení 640x480 pixelů. Pro implementaci bylo použito zařízení Virtex-II Pro VP70 od firmy Xilinx.

Výstupy algoritmu a mediánového filtru jako celku je možné vidět na obrázcích 1 a 2, kdy obrázek 1 zobrazuje testovací fotografii v originále a v zašuměné a vyfiltrované podobě. Obrázek 2 zobrazuje výsledek filtrace na reálném ultrasonografickém obraze.

Obrázek 1 – Výsledek filtrace mediánovým filtrem
(a) Originální obraz, (b) Obraz s šumem typu Salt & Pepper, (c) Vyfiltrovaný obraz



Obrázek 2 – Výsledek filtrace mediánovým filtrem
(a) Originální obraz s šumem, (b) Vyfiltrovaný obraz

1.2.1.2.6 Hardware architecture for advanced image processing

Bibliografický záznam

GRABOWSKI, K. a A. NAPIERALSKI. Hardware architecture for advanced image processing. In: *IEEE Nuclear Science Symposuim & Medical Imaging Conference* [online]. IEEE, 2010, s. 3626-3633 [cit. 2017-03-30]. DOI: 10.1109/NSSMIC.2010.5874488. ISBN 978-1-4244-9106-3. Dostupné z: http://ieeexplore.ieee.org/document/5874488/

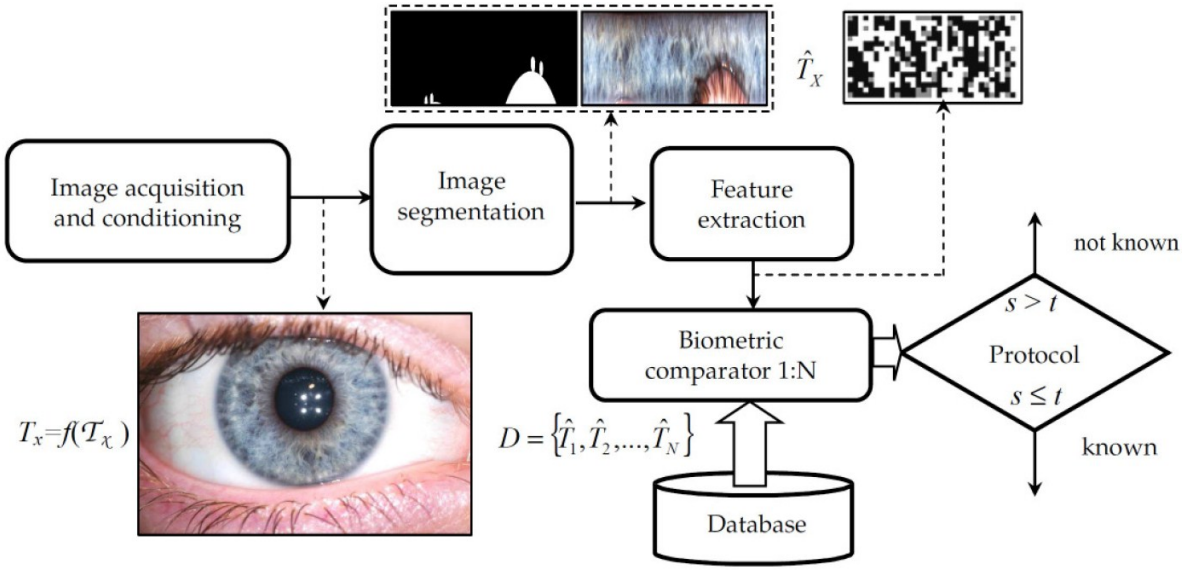
Úvod

Článek popisuje nový přístup k pokročilému zpracování a analýze obrazu na hardwarové úrovni, čímž jsou myšlena programovatelná hradlová pole.

Obsah

Tento článek popisuje hardwarový systém založený na programovatelných hradlových polích pro pokročilou analýzu biometrických dat jako součást systému s celkovou biometrickou identifikací. Většina práce se zabývá popisem systému pro biometrickou identifikaci duhovky, který byl vybrán jako ukázkový příklad aplikace daného algoritmu.

Algoritmus identifikace lidské duhovky je prováděn v několika krocích. Po pořízení snímku je provedena segmentace duhovky, která označí oblast zájmu, tedy oblast, kde se nachází pouze oční duhovka. V tomto procesu jsou označeny hranice vnějšího a vnitřního obvodu duhovky. Dalším krokem je normalizace kruhového obrazu duhovky. Normalizace převádí kruhový obraz duhovky na čtvercový obraz v polárním souřadnicovém systému.

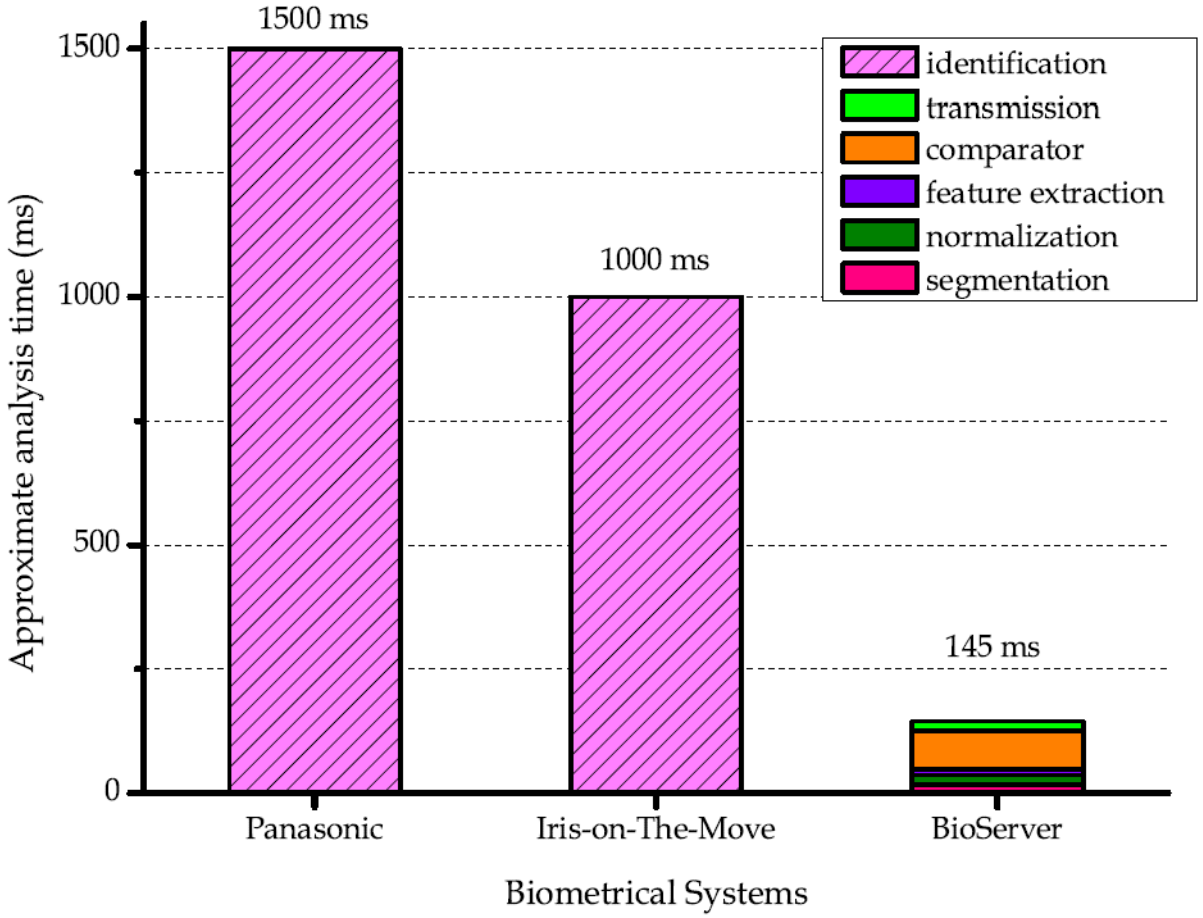


Obrázek 1 – Blokové schéma systému pro identifikaci duhovky

Následujícím krokem je rozlišení jednotlivých vlastností/znaků duhovky. Tyto jednotlivé znaky rohovky jsou rozeznávány skrze filtraci pomocí textur. Výstupem předešlého kroku je binární

sekvence nesoucí informace o vlastnostech/znacích zpracované rohovky. Zmíněná binární sekvence zaujímá délku o 2048 bitech. Posledním částí algoritmu je biometrická komparace, která porovnává binární sekvence a určuje hodnotu (skóre) podobnosti. Podle daného skóre je pak vyhodnoceno, zdali je testovaná duhovka rozpoznána či nikoliv.

Práce mimo jiné porovnává výsledky svého systému s dalšími, však komerčními, systémy. Toto porovnání je možné vidět níže na obrázku 2.



Obrázek 2 – Porovnání systému pro identifikaci rohovky (BioServer) s dalšími systémy

1.2.1.2.7 FPGA Implementation of EM Algorithm for 3D CT Reconstruction

Bibliografický záznam

CHOI, Young Kyu, Jason CONG a Di WU. FPGA Implementation of EM Algorithm for 3D CT Reconstruction. In: *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines* [online]. IEEE, 2014, s. 157-160 [cit. 2017-03-30]. DOI: 10.1109/FCCM.2014.48. ISBN 978-1-4799-5111-6. Dostupné z: http://ieeexplore.ieee.org/document/6861610/

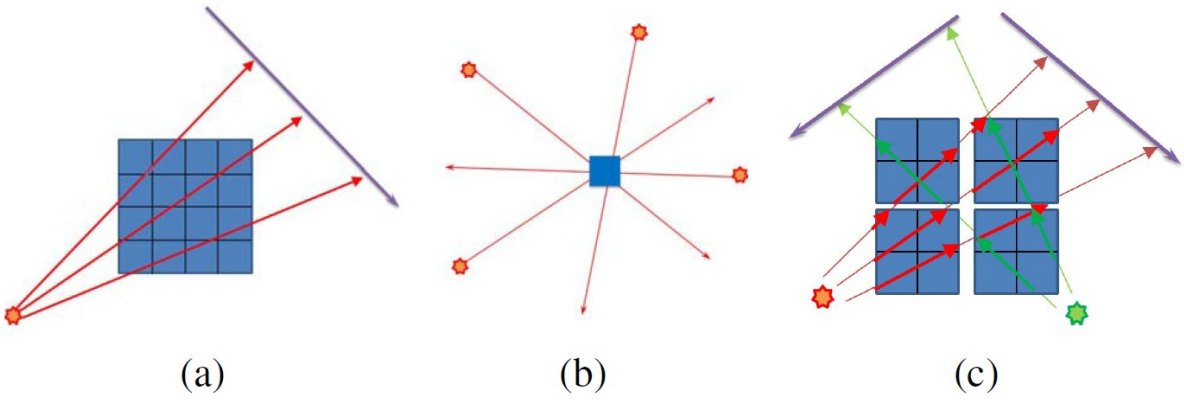
Úvod

Práce popisuje implementaci algoritmu očekávané maximalizace (EM) pro trojrozměrnou rekonstrukci počítačové tomografie na programovatelných hradlových polích.

Obsah

Článek se zabývá řešením rychlosti EM algoritmu při použití během rekonstrukce trojrozměrné počítačové tomografie, která nabízí nižší radiační zátěž pacienta, ovšem pomalá rychlost algoritmu překáží praktickému využití. Novým přístupem je použití programovatelných hradlových polí, která umožňují s použitím sinogramu a voxelové intenzity zrychlit algoritmus až 85 krát.

Během testování byl použit procesor Intel Xeon 5138 a CT projekce o velikosti 672x72x15022 pixelů. Takto velká projekce zabrala tomuto procesoru 54 hodin. Problémem algoritmu je, že data nejsou nijak recyklována a jsou pokaždé načtena z externí paměti znovu, z toho plyne přístup do externí paměti pro voxelovou intenzitu 1004 krát za iteraci a pro sinogram 831 krát. Takový postup si žádá recyklaci (použití již načtených) dat. Hlavním přínosem této práce tedy je nalezení možných metod pro recyklaci dat a tím urychlení algoritmu očekávaných maximalizací.



Obrázek 1 – Metody pro snížení přístupů do paměti
(a) Paprsková, (b) Voxelová, (c) Hybridní (Paprsková + Voxelová)

Práce dále popisuje tři metody pro recyklaci dat a tím snížení přístupů do externí paměti. První metodou je paprsková metoda, která využívá toho, že když každý paprsek prochází více voxely, jak je vidět na obrázku 1 (a), tak sinogram z přední projekce nebo sinogram ze zadní projekce může být uložen jako lokální proměnná a může být použit bez potřeby dalšího přístupu do externí paměti. Tento přístup snižuje počet přístupů do paměti v každé iteraci 1004 krát.

Druhou metodou je voxelová metoda, zobrazená na obrázku 1 (b), jejímž principem je ukládání seznamu paprsků, které procházejí daným voxelem. V této metodě je do lokální proměnné uložena voxelová intenzita. Tento postup by algoritmus značně urychlil, ale problém je v tom, že momentálně neexistuje účinný postup, který by dokázal vytvářet seznam paprsků procházejících daným voxelem za chodu. Ovšem seznamy by pak stejně byly velice objemné, což by bylo taktéž kontraproduktivní.

Třetí metodou je hybridní metoda, zobrazená na obrázku 1 (c), která využívá výhody z obou předchozích metod. Do lokální proměnné je uložena voxelová intenzita a je vytvořen seznam paprsků procházejících skrze daný voxel, ovšem s tím rozdílem, že jeden voxel je nyní rozšířen na blok o velikosti 64x64x1 voxel. Tyto bloky jsou pak uloženy do lokální paměti. Tento přístup zredukoval obsazení paměti seznamy z 302 GB na pouhých 58 MB.

Všechny testy algoritmu byly provedeny a algoritmus sám byl implementován na zařízení Convey HC-1ex, které se sestává ze čtyř programovatelných hradlových polí Virtex 6 LX760 od firmy Xilinx. Výsledná implementace algoritmu je tedy oproti jednojádrovém procesoru 85 krát rychlejší.

1.2.1.2.8 Detection of a brain tumor using segmentation and morphological operatorsfrom MRI scan with FPGA

Bibliografický záznam

THOMAS, H M William a S C Prasanna KUMAR. Detection of a brain tumor using segmentation and morphological operatorsfrom MRI scan with FPGA. In: *2015 International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)* [online]. IEEE, 2015, s. 728-731 [cit. 2017-03-30]. DOI: 10.1109/ICATCCT.2015.7456979. ISBN 978-1-4673-9223-5. Dostupné z: http://ieeexplore.ieee.org/document/7456979/

Úvod

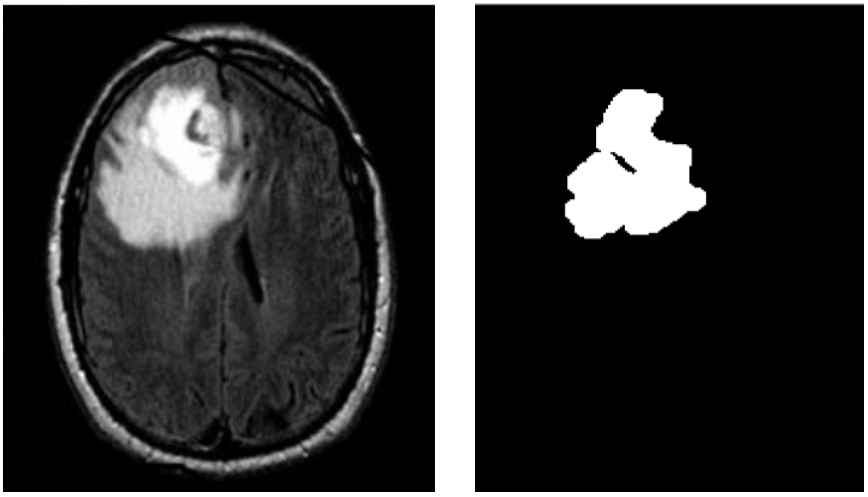
Článek se zabývá detekcí mozkových nádorů na snímcích z magnetické rezonance s použitím metod segmentace obrazu, které jsou implementovány na programovatelných hradlových polích.

Obsah

Tento článek popisuje metodu detekce mozkových nádorů pomocí segmentace obrazu s použitím programovacích hradlových polí, kdy je implementace metody zajištěna v systému MATLAB. Systém MALTAB nabízí grafické prostředí k vývoji hardwarových návrhů pro programovatelná hradlová pole, ve kterém byl algoritmus této práce navržen. Systém MATLAB je dále využit pro předzpracování vstupních dat a následné zpracování výsledků.

Princip metody je rozdělen do tří částí. Předzpracování dat, které načítá obrazová data, v případě potřeby je transformuje do odstínů šedi, upravuje rozměry, odstraňuje také případný šum a převádí na textový soubor, který pokračuje do další části. Druhou částí je samotné zpracování obrazu s použitím technologie programovatelných hradlových polí, v této implementaci zastoupené zařízením Spartan 3 od firmy Xilinx. Třetí částí je zpracování výsledných dat, která navrátí programovatelné hradlové pole. Toto zpracování je opět implementováno v systému MATLAB a zajišťuje převod výsledků z textového souboru do obrazové podoby.

Provedený experiment potvrdil, že použití programovatelných hradlových polí pro segmentaci obrazu a morfologické operace je velmi rychlé, přesné a účinné. Výsledky této metody identifikace mozkových nádorů je možné vidět níže na obrázku 1.



Obrázek 1 – Vstupní obraz (vlevo) a výstupní obraz (vpravo)

2 Programovatelná hradlová pole

Programovatelná hradlová pole, anglickou zkratkou FPGA, jsou speciální digitální integrované obvody, které obsahují různě složité programovatelné bloky pospojované nastavitelnou maticí spojů. Jak již název napovídá, jde o programovatelná logická zapojení, která mohou být popsána některým z programovacích jazyků sloužících pro popis hardwaru. Nejpoužívanějšími programovacími jazyky pro popis zapojení na hradlových polích jsou jazyky VHDL a Verilog. [11]

Na dnešních programovatelných hradlových polích je možné zapojit, jak jednoduché logické obvody, tak celý komplikovaný procesor včetně všech jeho periferií. Mezi nejznámější výrobce programovatelných hradlových polí patří firmy Xilinx a Altera. [12]

2.1 Historie a vývoj

Mezi první digitální hradlová pole se řadí programovatelná logická pole (FPLA) firmy Signetics Corporation, která využívala programovatelná pole s logickým součinem, následované logickým součtem a pro uložení logického návrhu paměť PROM, tedy bylo možné je naprogramovat pouze jednou. S postupným vývojem přibývaly bloky logických prvků, jako například klopné obvody, které nabízely vývojářům možnost vytvářet i sekvenční logické obvody. Tyto obvody se nazývaly programovatelná logická zařízení (PLD). Rozvoj opětovně použitelných pamětí přinesl možnost nových logických prvků, které se dají nazvat konfigurovatelnými logickými tabulkami (LUT), které jsou schopny uchovat libovolnou logickou funkci, nezávisle na její složitosti, protože uchovávají pouze výslednou pravdivostní tabulku. Tyto logické prvky jsou využívány i dnes v současných programovatelných hradlových polích (FPGA). [11][12][13]

2.2 Výhody a nevýhody

Hlavní výhodou programovatelných hradlových polí je variabilnost logických obvodů, které je možné na těchto deskách navrhnout. Tento přístup je velmi výhodný během návrhu prototypů, protože je možné návrh velmi rychle upravovat a v podstatě okamžitě jej i testovat.

Za nevýhodu můžeme považovat omezení návrhu dostupností logických prvků na dané desce nebo také cenu těchto desek, která je poměrně vysoká. [11][12][13]

2.3 Architektura

Vzhledem k tomu, že superpočítač Rivyera využívá programovatelná hradlová pole od firmy Xilinx, bude se tato kapitola věnovat výhradně architektuře hradlových polí této firmy. Všechna programovatelná hradlová pole firmy Xilinx jsou konfigurovány pomocí statické paměti RAM, což v praxi znamená, že daná konfigurace logických bloků se musí při každém připojení napájení znovu načíst, protože paměť RAM je energeticky závislá, tedy po odpojení napájení ztrácí veškeré informace v sobě uložené. Výsledkem tohoto přístupu je možnost opakované programovatelnosti, tedy změny dané konfigurace. Bylo by možné použít i paměti EEPROM, ale ty jsou oproti pamětím RAM pomalejší. [11][12][13]

2.4 Programovatelná hradlová pole řady Spartan-6

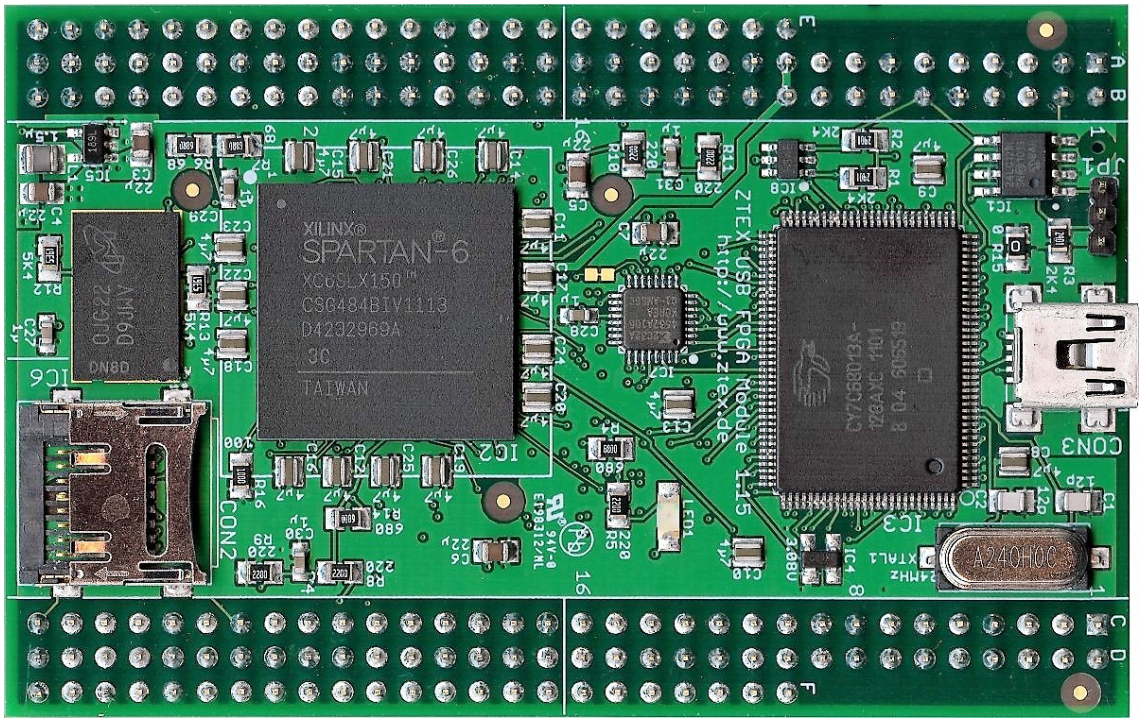
Programovatelná hradlová pole Spartan-6 vyrábí firma Xilinx, která je největším výrobcem na trhu s FPGA technologiemi.

2.4.1 Parametry

Řada Spartan-6 nabízí podle dané verze od 3840 do 147443 logických buněk, které jsou oproti starší řadě rychlejší a mají poloviční spotřebu energie. Nově jsou k dispozici LUT s šesti vstupy. Dále jsou k dispozici také blokové paměti typu RAM o velikosti 18kb, DSP48A1 řezy obsahující násobičku 18x18, sčítačku a 48 bitový akumulátor (střádač), anebo také paměťové řadiče. [15]

2.4.2 Spartan-6 XC6SLX150

XC6SLX150 je nejlépe vybavenou verzí z řady Spartan-6 od firmy Xilinx. Tato verze obsahuje 147 tisíc logických prvků, 180 násobiček a sčítaček, a 184 tisíc klopných obvodů. Superpočítač Rivyera používá právě tuto verzi FPGA, a proto je v tabulkách 2.1 a 2.2 porovnání vlastností a parametrů s ostatními verzemi z řady Spartan-6. [15]



Obrázek 2.1 – Prototypová deska s FPGA Spartan-6 XC6SLX150

Tabulka 2.1 – Tabulka srovnání jednotlivých variant v rodině Spartan-6 (1. část) [15]

Zařízení	Logické prvky (1)	Konfigurovatelné logické bloky			DSP48A1 řezy (3)
		Řezy (2)	Klopné obvody	Distribuovaná paměť RAM (kb)	
XC6SLX4	3840	600	4800	75	8
XC6SLX9	9152	1430	11440	90	16
XC6SLX16	14579	2278	18224	136	32
XC6SLX25	24051	3758	30064	229	38
XC6SLX45	43661	6822	54576	401	58
XC6SLX75	74637	11662	93296	692	132
XC6SLX100	101261	15822	126576	976	180
XC6SLX150	147443	23038	184304	1355	180

Tabulka 2.2 – Tabulka srovnání jednotlivých variant v rodině Spartan-6 (2. část) [15]

Zařízení	Bloková paměť RAM		CMT (5)	Paměťové řadiče	Vstupně-výstupní skupiny pinů	Uživatelské vstupně-výstupní piny
	18 kb (4)	Max (kb)				
XC6SLX4	12	216	2	0	4	132
XC6SLX9	32	576	2	2	4	200
XC6SLX16	32	576	2	2	4	232
XC6SLX25	52	936	2	2	4	266
XC6SLX45	116	2088	4	2	4	358
XC6SLX75	172	3096	6	4	6	408
XC6SLX100	268	4824	6	4	6	480
XC6SLX150	268	4824	6	4	6	576

Poznámky k tabulce 2.1 a 2.2:

1. Počet logických prvků je u řady Spartan-6 vyšší díky novým LUT s šesti vstupy.
2. Každý řez obsahuje čtyři LUT a osm klopných obvodů.
3. Každý DSP48A1 řez obsahuje násobičku 18x18, sčítačku a akumulátor (střádač).
4. Bloková paměť RAM má v základu 18 kb, ale je možné blok použít jako dva nezávislé bloky s velikostí 9 kb.
5. Každý CMT (generátor hodinového signálu) obsahuje dva digitální generátory a jeden fázový závěs.

3 Superpočítač Rivyera

Superpočítač Rivyera je kombinací výpočetního jádra na úrovni programovatelných hradlových polí a předzpracování a obsluhy dat na úrovni programovacích jazyků C nebo Java. Výrobcem a dodavatelem je německá firma SciEngines, která jej dodává jako serverové řešení. Superpočítač se dodává v několika konfiguracích, které se od sebe různí celkovým počtem FPGA karet v FPGA periferii, která nabízí v maximálním osazení prostor až pro 256 jednotlivých FPGA. [16]

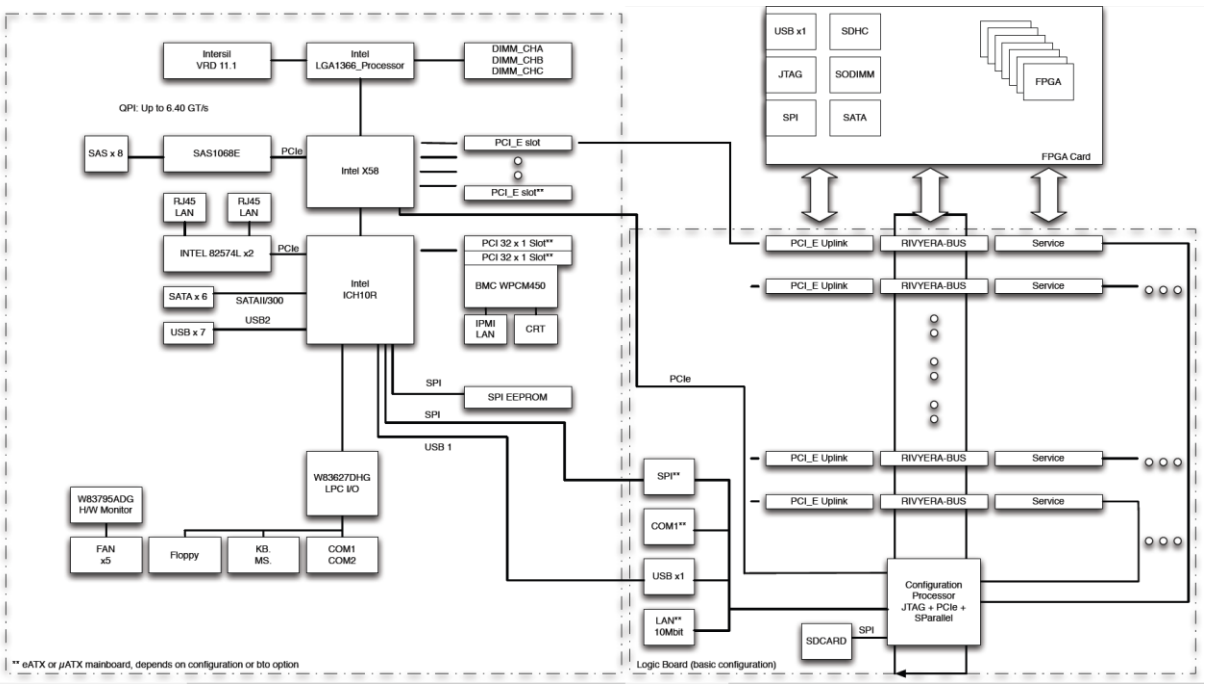
Obrázek 3.1 – Celkový pohled na superpočítač Rivyera [16]

3.1 Architektura

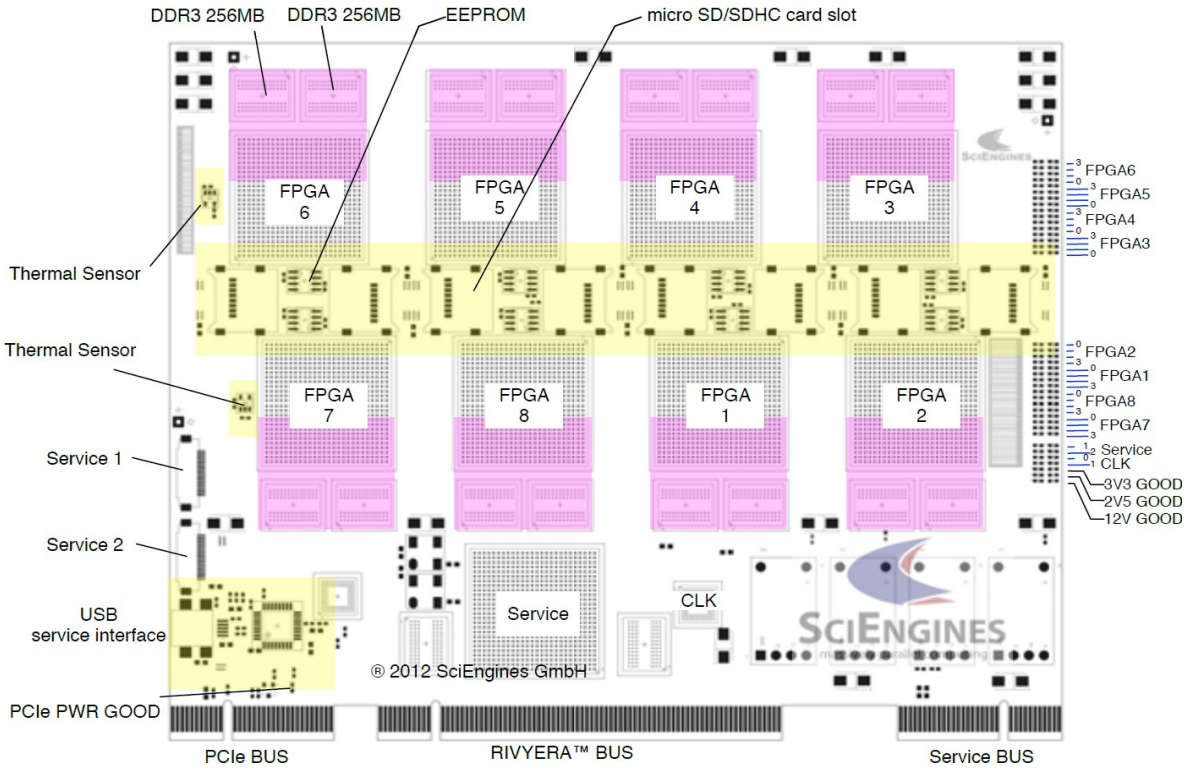
Z jednoduchého pohledu se superpočítač Rivyera skládá ze dvou základních částí. První částí je periferie se sloty pro FPGA karty a druhou je počítač serveru.

3.1.1 FPGA periferie

FPGA periferie má 16 slotů pro FPGA karty z nichž každá obsahuje osm FPGA. Karty i jednotlivá FPGA jsou připojeny do speciální sběrnice Rivyera bus a jsou uspořádány do tzv. systolického řetězu, kdy je každé člen přímo spojen se svým sousedem a vytváří tak řetězovité uspořádání. Toto zapojení umožňuje rychlou komunikaci mezi dvěma sousedními body, ale může být problematické v delších řetězcích. Spolehlivou komunikaci mezi oběma částmi zajišťuje standartní systémová sběrnice PCI-Express, přičemž každá FPGA karta s osmi FPGA má vlastní PCI-Express spojení s počítačem serveru. Ve výsledném schématu zapojení se periferie s FPGA kartami chová jako host a aplikace, v jazyce C nebo Java, běžící na počítači serveru, jako hostitel. [22][25]



Obrázek 3.2 – Zapojení superpočítače Rivyera (Počítač serveru vlevo, FPGA periferie vpravo) [22]



Obrázek 3.3 – Schéma zapojení FPGA karty [22]

3.1.1.1 Paměť

Po stránce paměťové nabízí každé FPGA 512 MB dynamické paměti DRAM. Navíc každé FPGA nabízí možnost rozšíření paměti o 2 GB, tedy celková paměť jednoho FPGA může činit až 2,5 GB, což pro Rivyeru plně obsazenou 16 FPGA kartami znamená celkovou paměť až 320 GB. V případě, že by ani tento paměťový prostor nesplňoval nároky aplikace, je možné při zachování stejného počtu karet připojit další FPGA až do celkového počtu 256 FPGA. Další možností je zvýšení paměti pomocí SD karty až do velikosti 32 GB na jedno FPGA. [16]

3.1.2 Počítač serveru

Počítač serveru je taktéž výkonným strojem, který se však může lišit kus od kusu, protože se jedná o počítač serverové kvality z aktuální komerční nabídky. V případě našeho počítače se jedná o stroj s procesorem Intel(R) Xeon(R) CPU E3-1230 V2 s taktovací frekvencí 3.30 GHz a operační pamětí o velikosti 15 GB. Počítač má pro pevný disk sběrnici SATA-II a nabízí uložiště o velikosti až 4 TB, naše verze má k dispozici 2 TB.

3.2 Aplikační programovací rozhraní

Aplikační programovací rozhraní neboli API je pro superpočítač Rivyera dodáno výrobcem, firmou SciEngines. API obsahuje definice proměnných a funkcí pro ovládání a komunikaci mezi hostitelskou aplikací a hosty v podobě FPGA.

3.2.1 Inicializace superpočítače Rivyera

Než vůbec může být spuštěna hostitelská aplikace, musí být skrze příkazový terminál (využitím pomocné aplikace firmy SciEngines) zavolán příkaz ***se_machine start***, který uvede do chodu celou periferii (cluster) s FPGA. [22]

3.2.2 Hostitel

Na straně hostitele je třeba vykonávat obsluhu hosta v podobě FPGA periférie, skrze kterou hostitel komunikuje s jednotlivými FPGA. Úloha hostitele je zajistit alokaci FPGA periférie, naprogramovat jednotlivá FPGA daným logickým návrhem a předzpracovat a obsluhovat data, která budou odesílána a přijímána z jednotlivých FPGA. Na konci výpočetního procesu by měl hostitel zase všechna FPGA odprogramovat a uvolnit FPGA periférii.

Pro výše zmíněné postupy nabízí firma SciEngines API pro programovací jazyky C a Java, které k těmto účelům nabízí všechny potřebné datové typy, funkce a struktury. V případě, že dojde k nějaké chybě či neošetřené výjimce, aplikační rozhraní implementuje i odpovídající typy výjimek, které umožňují ošetření těchto výjimečných či chybových stavů.

V podkapitolách níže jsou popsány nejčastěji užívané třídy, užitečné konstanty a metody aplikačního programovacího rozhraní implementovaného v jazyce Java včetně možných přetížení a souvisejících výjimek. Všechny uvedené metody musí mít ošetřené případné výjimky. [18]

3.2.2.1 Třídy

SeAddress

Třída ***SeAddress*** je v API používána pro adresaci jednotlivých FPGA. Adresa je definována kombinací vstupních parametrů čísla kontroléru, čísla slotu FPGA karty, čísla FPGA a čísla registru. Jednotlivá identifikační čísla mohou být nahrazena konstantami, které jsou popsány níže. [18]

SeOptions

Třídu SeOptions je nutné použít pro nastavení funkcionality FPGA periferie během její alokace, která je prováděna metodou ***se_allocMachine***. Konstruktor této třídy má jako vstupní parametry výčtové typy ***SeWriteBehavior*** a ***SeRoutingMethod***.

Výčtový typ ***SeWriteBehavior*** nese hodnoty ***se_write_async*** a ***se_write_sync***, které zastupují metodu synchronního a asynchronního zápisu dat.

Výčtový typ ***SeRoutingMethod*** nese hodnoty ***se_routing_normal*** a ***se_routing_systolic***, které zastupují metodu klasického spojení a spojení pomocí metody systolického řetězení. [18]

3.2.2.2 Konstanty

SE_ADDR_FPGA_ALL

Konstanta, která sdružuje všechna identifikační čísla FPGA do jednoho a umožňuje vícenásobnou adresaci FPGA. Konstantu je možné kombinovat také s ***SE_ADDR_SLOT_ALL*** a ***SE_ADDR_CONTR_ALL***. [18]

SE_ADDR_SLOT_ALL

Konstanta, která sdružuje všechna identifikační čísla slotů FPGA karet do jednoho a umožňuje vícenásobnou adresaci slotů FPGA karet. Konstantu je možné kombinovat také s ***SE_ADDR_FPGA_ALL*** a ***SE_ADDR_CONTR_ALL***. [18]

SE_ADDR_CONTR_ALL

Konstanta, která sdružuje všechna identifikační čísla kontrolérů do jednoho a umožňuje jejich vícenásobnou adresaci. Konstantu je možné kombinovat také s ***SE_ADDR_FPGA_ALL*** a ***SE_ADDR_SLOT_ALL***. [18]

SE_READ_ACTIVE

Konstanta k nastavení aktivního čtecího módu pro metodu ***se_read***. [18]

SE_READ_PASSIVE

Konstanta k nastavení pasivního čtecího módu pro metodu ***se_read***. [18]

SE_TIMEOUT_INFINITE

Konstanta nesoucí hodnotu pro nastavení nekonečného časového limitu dané operace. Konstantu je možné použít například pro časový limit naprogramování FPGA, čtení nebo zápisu dat. [18]

3.2.2.3 Alokace periferie (Java)

Pro alokaci periferie je v aplikačním programovacím rozhraní statická metoda ***se_allocMachine***, která má jako vstupní parametr identifikační číslo zařízení (Číslo zařízení je pro případ, že by bylo k dispozici více periferií nebo by šlo o alokaci testovací periferie, která je pouze virtuální). Tato statická metoda nabízí také další přetížení, která umožňují alokaci s různými dodatkovými nastaveními, která jsou zastoupena instancí třídy ***SeOptions***. [18]

Možná přetížení metody *se_allocMachine*:

- ***static void se_allocMachine(int machine)*** vyhazuje ***SeApiException***
- ***static void se_allocMachine(int machine, SeOptions options)*** vyhazuje ***SeApiException***

3.2.2.4 Uvolnění periférie (Java)

Uvolnění periferie je v aplikačním programovacím rozhraní zajištěno statickou metodou ***se_freeMachine***, která má jako vstupní parametr identifikační číslo zařízení. [18]

Možná přetížení metody *se_freeMachine*:

- ***static void se_freeMachine(int machine)*** vyhazuje ***SeApiException***

3.2.2.5 Naprogramování jednotlivých FPGA (Java)

Programování jednotlivých FPGA probíhá pomocí statické metody ***se_program***, která má jako vstupní parametry identifikační číslo zařízení, adresu cílového FPGA a cestu k souboru typu bit, který obsahuje sestavený logický návrh našeho řešení. Dalším parametrem je časový limit pro naprogramování. Metoda zajistí, že adresované FPGA bude naprogramováno vybraným logických návrhem.

Adresace je zprostředkována pomocí instance třídy ***SeAddress***. V případě, že bychom chtěli naprogramovat daným logickým návrhem všechna dostupná FPGA, je možné využít při vytváření instance třídy ***SeAddress*** předdefinované konstanty ***SE_ADDR_FPGA_ALL***. Pro neomezený časový limit je dále možné použít předdefinované konstanty ***SE_TIMEOUT_INFINITE***. [18]

Možná přetížení metody *se_program*:

- ***static void se_program (int machine, SeAddress addr, String filename, long timeout)*** vyhazuje ***SeApiException***

3.2.2.6 Odprogramování jednotlivých FPGA (Java)

Odprogramování jednotlivých FPGA je zajištěno pomocí statické metody ***se_deprogram***, která má jako vstupní parametry identifikační číslo zařízení a adresu cílového FPGA. [18]

Možná přetížení metody *se_deprogram*:

- ***static void se_deprogram (int machine, SeAddress addr)*** vyhazuje ***SeApiException***

3.2.2.7 Zápis dat (Java)

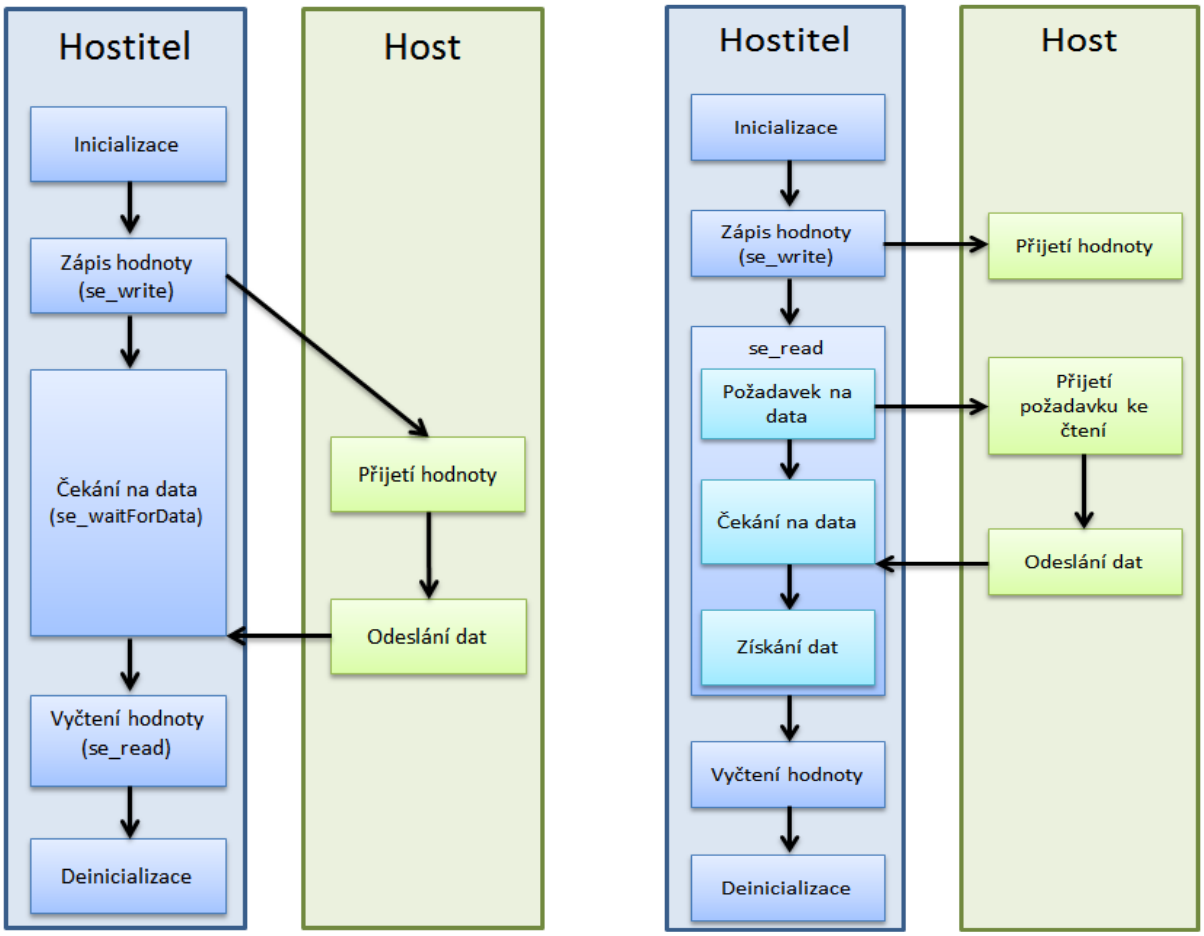
Zápis dat do FPGA probíhá pomocí metody ***se_write***, která pomocí vstupních parametrů adresuje cílové FPGA, datový paket, jejich počet a časový limit. Metoda navrací počet datových paketů, které byly úspěšně zapsány. [18]

Možná přetížení metody *se_write*:

- ***static long se_write (int machine, SeAddress addr, ByteBuffer payload, long size, long timeout)*** vyhazuje ***SeApiException***

3.2.2.8 Čtení dat (Java)

Čtení dat z FPGA je zajištěno statickou metodou se_read, která ve vstupních parametrech adresuje cílové FPGA, cílový datový paket, jejich počet, mód čtení a časový limit. Čtení může probíhat ve dvou módech, které jsou popsány na obrázcích níže. V pasivním módu čtení hostitel vyčkává, dokud host data neodešle. V aktivním módu čtení host vyčkává na požadavek hostitele. Metoda navrací počet datových paketů, které byly vyčteny. [18]



Obrázek 3.4 – Blokové schéma principu pasivního (vlevo) a aktivního čtení (vpravo)

Možná přetížení metody *se_read*:

- ***static long se_read (int machine, SeAddress addr, ByteBuffer payload, long size, int mode, long timeout)*** vyhazuje ***SeApiException***

3.2.2.9 Čekání na data (Java)

Metoda pro zajištění čekání na data je určena pasivnímu módu čtení dat. Jedná se o metodu se_waitForData, která má jako vstupní parametry identifikační číslo zařízení, číslo kontroléru a časového limitu. Metoda navrací adresu FPGA, které má připravená data. [18]

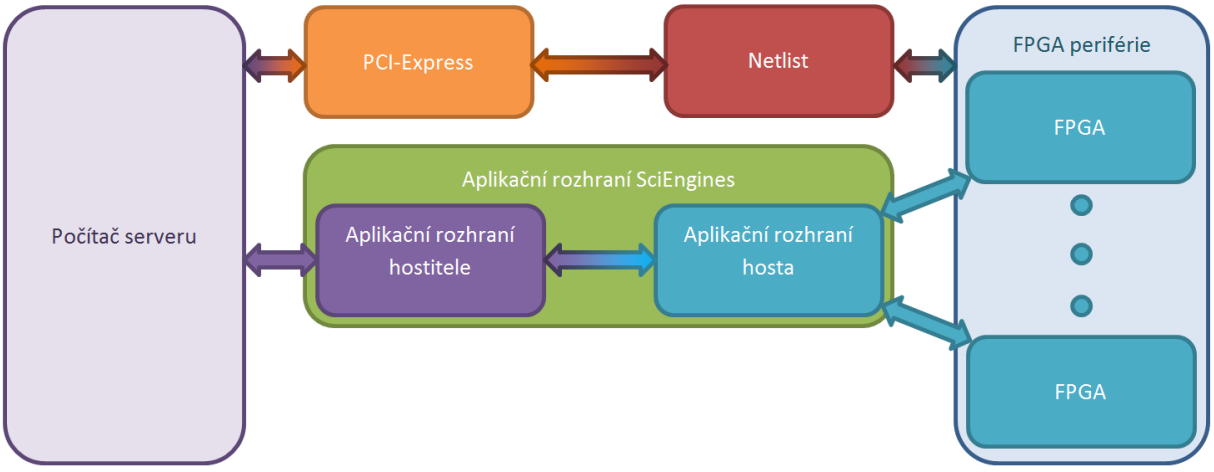
Možná přetížení metody *se_waitForData*:

- ***static SeAddress se_waitForData (int machine, int controller, long timeout)*** vyhazuje ***SeApiException***

3.2.3 Host

Na straně hosta je zapotřebí obsloužit vstupní registr, zpracovat přijatá data odpovídajícím způsobem a navrátit je do výstupního registru. Připravenost vstupních i výstupních registrů k zápisu či čtení je ošetřeno pomocí bitových příznaků.

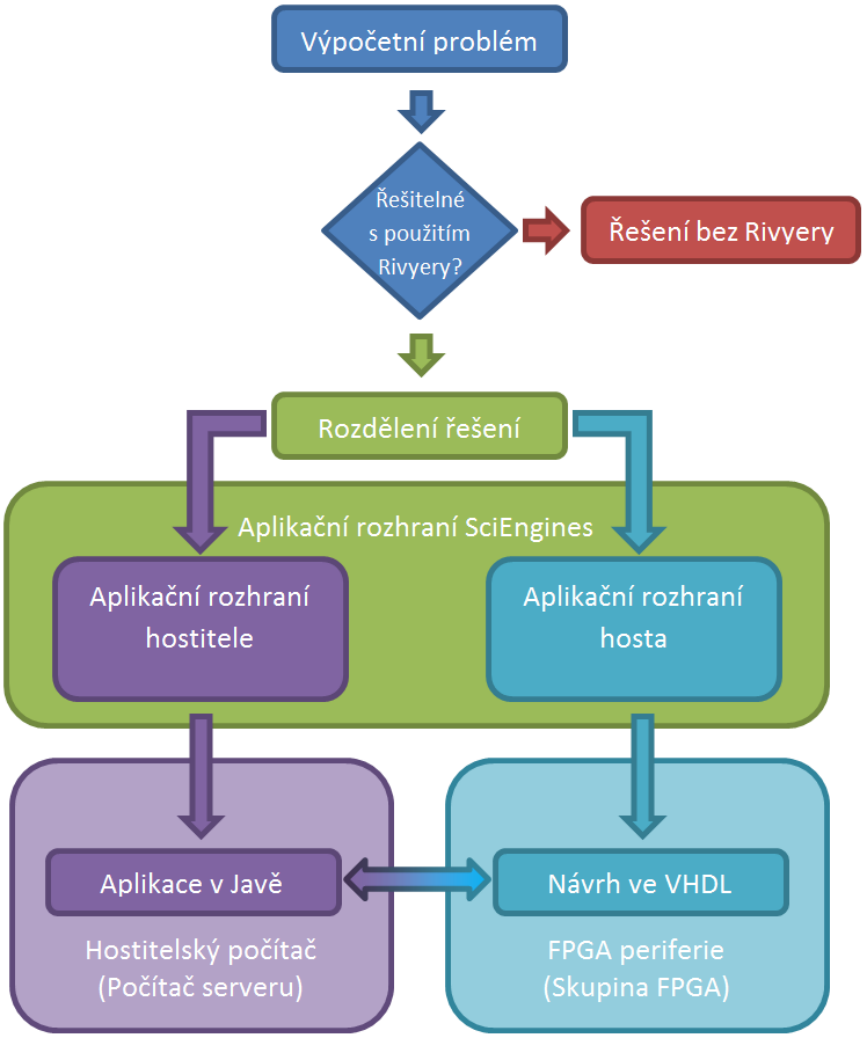
Firma SciEngines nabízí pro logické návrhy hosta API pro programovací jazyk VHDL a také netlist, který danému logickému návrhu na programovatelných hradlových polích udává, kde se nachází daný vstup či výstup rozhraní superpočítače Rivyera. [19]



Obrázek 3.5 – Blokové schéma principu koncové komunikace mezi hostitelem a hostem

4 Superpočítač Rivyera a jeho využití

Řešení výpočetně složitých problémů použitím superpočítač Rivyera je složitý proces, který je ovšem pro uživatele (vývojáře řešícího problém) zjednodušený tak, jak jen je to možné. Základní myšlenkou je rozdělení problému na dvě části. První částí je hostitelská aplikace, napsaná v jazyce Java nebo C, která obsluhuje druhou část, hostovací kombinačně-sekvenční návrh logického obvodu nahraný na deskách FPGA, který je popsán v jazyce VHDL. Je možné tedy danou problematiku řešit tak, že hostitelská aplikace se stará o komunikaci se softwarovou vrstvou, předzpracování dat a hostovací logický návrh se stará o zpracování dat na té nejsurovější, hardwarové, úrovni. Toto spojení nabízí vysoký výkon, protože můžeme poměrně snadno data v jakémkoliv formátu skrze hostitelskou aplikaci rozložit na surová data a tak pak zpracovat logickým obvodem implementovaným na velmi rychlých deskách FPGA. Zpracovaná data jsou pak samozřejmě opět v surové formě navrácena hostitelské aplikaci, která je musí složit do odpovídajícího formátu.



Obrázek 4.1 – Blokové schéma postupu implementace řešení s použitím superpočítače Rivyera

4.1 Postupná implementace řešení problému

Tato kapitola popisuje postup, kterým je možné vytvořit řešení pro daný problém použitím superpočítače Rivyera. Tento postup předpokládá, že vývojář má již připravena veškerá nastavení a požadavky na vývoj pro superpočítač Rivyera (potřebný software je nainstalován a správně nastaven).

4.1.1 Řešitelnost problému na surové binární úrovni

Než vůbec vývojář vztáhne metaforickou ruku po superpočítači Rivyera, je třeba se zamyslet nad tím, zda je vůbec problém vhodný pro zpracování na hradlových polích, tedy vhodný pro zpracování na binární úrovni a to způsobem, který nabízí právě hradlová pole a jazyk VHDL. Ne každý problém je vhodné řešit pomocí hradlových polí, i přestože je výkon ohromující, při použití špatným způsobem se může obrátit proti vývojáři (delší doba implementace, vyšší cena, ukončení vývoje z důvodu omezení dané technologie, se kterým se nepočítalo).

4.1.2 Rozdělení problematiky

Dalším velkým úkolem je rozdělení řešení na dvě části, část hostitele a část hosta. Hostitelská část, implementovaná v Javě nebo C, bude zajišťovat předzpracování dat a obsluhu jednotlivých hradlových polí. Část hosta se bude starat pouze o zpracování dat, která obdrží od hostitele. Obě části musí být vytvořeny tak, aby na sebe dokonale navazovaly.

4.1.3 Implementace hostitele (Java, C)

Hostitelská aplikace běžící na počítači serveru bude zodpovídat za předzpracování dat a obsluhu jednotlivých programovatelných hradlových polí v periferní části superpočítače Rivyera. Komunikace mezi hostitelem a hostem probíhá skrze datové pakety o velikosti 64 bitů. Data pro zpracování je proto třeba rozdělit na jednotlivé pakety, které budou odeslány do a přijímány z hosta. Data přijatá od hosta je pak třeba opět složit do výstupního formátu.

Aplikace samozřejmě musí zajistit také alokaci periferie hradlových polí a nahrání daného logického návrhu v podobě bitového souboru do jednotlivých hradlových polí.

4.1.4 Implementace hosta (VHDL)

Logický návrh hosta implementuje výpočetní postup již předzpracovaných dat, která přijme skrze vstupní 64 bitový registr, zpracuje je v rámci možností daného hradlového pole a skrze výstupní registr data zase navrátí hostiteli.

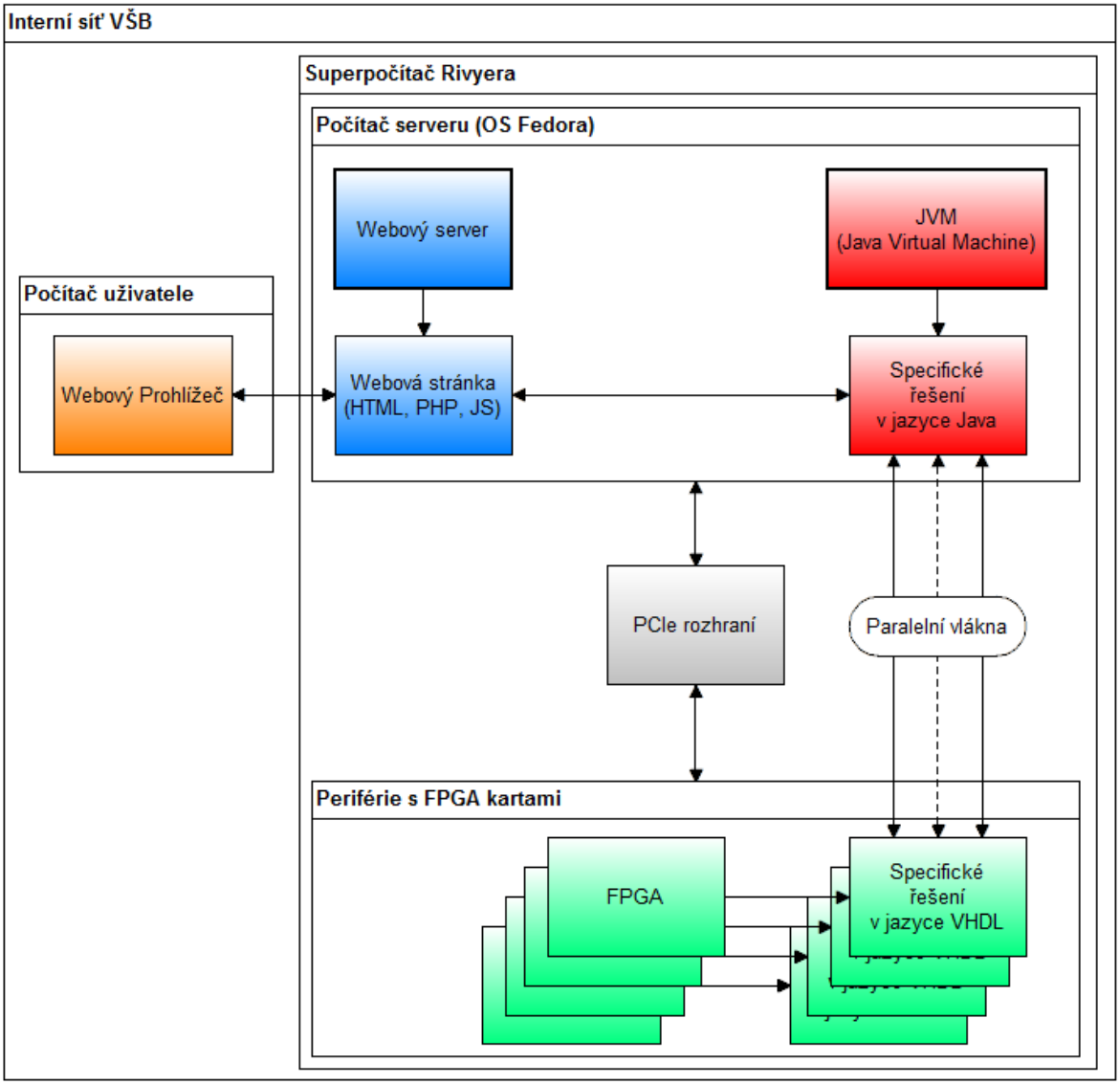
4.1.5 Spuštění výsledné aplikace

Výslednou aplikaci je možné spustit za předpokladu přidružení všech potřebných knihoven, které jsou k dispozici s aplikačním rozhraním od firmy SciEngines. Důležitým bodem pro zajištění funkce superpočítače Rivyera je zapnutí periférie skrze příkazový terminál operačního systému. Bez uvedení periferie s hradlovými poli do chodu, skrze příkazovou řádku, se bude aplikace hostitele chovat nepředvídatelně a může vyhazovat nesouvisející výjimky.

5 Návrh a postupný vývoj řešení vzdálené obsluhy a dávkového zpracování

5.1 Představa řešení

Ideou řešení bylo vytvoření vzdáleného přístupu a ovládání superpočítače Rivyera. Uživatel v interní síti by se skrze prohlížeč připojil na webovou stránku superpočítače, vybral by požadovanou aplikaci zpracování, nahrál data, počkal na zpracování a pak si stáhl výsledky. Pokročilejší uživatel s dostatečnými oprávněními by dále mohl nahrát vlastní řešení nějakého problému, které by se poté přidalo do výběru možných aplikací.



Obrázek 5.1 – Schéma řešení vzdáleného ovládání superpočítače Rivyera

5.2 Koncept úloh k řešení

Výsledek této práce popisuje postup pro vývoj a řešení výpočetních úloh s použitím superpočítače Rivyera a paralelního zpracování dat. Na superpočítači je možné řešit rozličné problémy, jejichž řešení nejsou omezena výpočetním výkonem, případně úložným prostorem. Vzhledem ke kombinaci vyšších jazyků, tedy C nebo Java, je možné jakýkoliv problém rozložit na operace na surové binární úrovni a je tedy možné zpracovávat data z různých datových formátů na hardwarové úrovni hradlových polí bez toho, aby bloky pro případné předzpracování zabíraly na hradlovém poli místo.

5.3 Vývojové a testovací prostředky

Vzhledem k tomu, že architektura superpočítače Rivyera je postavena na principu hostitele a hosta, využité programovací jazyky se pro každou tuto úroveň liší. Hostitelskou aplikace, běžící v linuxovém operačním systému Fedora, je možné implementovat v jazycích C a Java, pro které poskytuje výrobce superpočítače Rivyera aplikační programovací rozhraní. V našem případě je implementace uskutečněna v jazyce Java, protože jde o poměrně moderní programovací jazyk s objektovým přístupem. Aplikace hosta je pak řešena na úrovni hradlových polí, jejichž strukturální a funkční zapojení je popsáno jazykem VHDL.

Superpočítač Rivyera je dodán jako serverové řešení, a tedy počítač, pod kterým hardware superpočítače běží, nemá žádný monitor, klávesnici a myš, ale je připojen do internetové sítě, odkud je možné jej vzdáleně ovládat. Vývoj hostitelské a hostující aplikace probíhá mimo počítač a operační systém superpočítače Rivyera a je tedy nezávislý na místě a operačním systému, ovšem výstupy vývoje samozřejmě musí operační systém superpočítače podporovat. Vývoj hostitelské i hostující aplikace byl uskutečněn pod operačním systémem Windows a testování probíhalo skrze vzdálený přístup na server superpočítače Rivyera.

5.3.1 IntelliJ IDEA 2016.3.1

Pro návrh hostitelské aplikace v jazyce Java bylo použito vývojové prostředí IntelliJ IDEA od firmy JetBrains s.r.o., které podporuje několik programovacích jazyků a spoustu dalších funkcí, jako například automatické doplňování a nápovědu, podporu verzovacích softwarů (Git, SVN), zvýrazňování syntaktických i gramatických chyb a mnoho dalšího. Pro naše potřeby byla využita zdarma dostupná verze Community, která je určena pro vývoj na platformách JVM (Java Virtual Machine) a Android. Aplikace byla použita také pro vývoj webových stránek.

5.3.2 ISE Design Suite 14.1

Návrh zapojení na hradlových polích je realizován ve vývojovém prostředí ISE Design Suite od firmy Xilinx, která je zároveň také výrobcem desek hradlových polí použitých v superpočítači Rivyera. Superpočítač Rivyera využívá ve své době nejvyšší řadu desek hradlových polí označenou jako Spartan6 XC6SLX150, které pro naprogramování (nahrání navrženého logického obvodu) vyžadují placenou verzi vývojového prostředí nazvanou System Edition.

5.3.3 WinSCP

Pro přenos a vzdálenou manipulaci se soubory na serveru superpočítače Rivyera byl použit program WinSCP. Jedná se o klienta pro přenos souborů na vzdálené servery skrze protokoly STFP, FTP a SCP. Připojení k serveru je zabezpečeno pomocí SSH protokolů.

5.3.4 PuTTY

Pro vzdálené konzolové ovládání byl použit program PuTTY, který umožňuje zadávat příkazy a číst výstup konzole vzdáleného serveru skrze protokoly SSH a telnet.

5.3.5 Webový Nette Framework

Pro snadnější vývoj webových stránek v jazyce PHP byl použit Framework Nette, který má spoustu výhod a zjednodušení, oproti vývoji webových stránek od úplného začátku. Nabízí například šablonovací systém, spolehlivé zabezpečení, podporu moderních webových jazyků a objektový přístup.

5.3.6 Počítač použitý k testům

Veškeré testování a případné časové výsledky byly naměřeny na serverovém počítači Rivyery, jehož parametry je možné nalézt v kapitole 3.1.2.

5.4 Vývoj aplikace pro superpočítač Rivyera

Kapitola popisuje postupy pro vývoj aplikace pro superpočítač Rivyera, čímž se rozumí aplikace hostitele v jazyce Java a logický návrh hosta v jazyce VHDL. V podkapitolách jsou uvedeny prerekvizity a důležité úseky zdrojových kódů nezbytné k vytvoření aplikace.

5.4.1 Aplikace hostitele (Java)

Hostitelská aplikace má několik důležitých úkolů. V chronologickém pořadí jsou tyto úkoly následovné. Aplikace musí zajistit alokaci FPGA periferie, naprogramovaní cílových FPGA logickým návrhem, musí obsluhovat veškeré souborové vstupy a výstupy, zajišťuje jejich předzpracování formou rozkladu a převodu na data v surové binární formě, kterou akceptuje vstupně-výstupní rozhraní FPGA periférie, které je zastoupeno aplikačním programovým rozhraním firmy SciEngines. Dále je třeba obsluhovat vstupy a výstupy všech cílových FPGA, po splnění požadavků odprogramovat jednotlivá FPGA, uvolnit FPGA periferii a zajistit dopracování dat a případné souborové výstupy.

5.4.1.1 Prerekvizity

Obecně prvním předpokladem pro vývoj aplikací v jazyce Java je instalace vývojové sady pro platformu Java (Java Development Kit), která obsahuje základní nástroje pro vývoj. Minimální verze, kterou musí vývojová sada podporovat, aby byla kompatibilní s aplikačním programovacím rozhraním superpočítače Rivyera, je 1.7 (označena také jako Java 7).

Dalším krokem při vývoji aplikace je založení projektu v některém z vývojových prostředí akceptujících jazyk Java. Samozřejmě je možné vyvíjet aplikaci i bez vývojového prostředí, ovšem tento přístup je zbytečně komplikovaný a v dnešní době překonaný. Jako vývojové prostředí je možné použít například programy Eclipse, IntelliJ Idea nebo NetBeans IDE.

Vytvořenému projektu je třeba nastavit verzi virtuálního prostředí platformy Java (Java Runtime Environment), pro kterou je aplikace určena. V našem případě se musí výhradně jednat o verzi 1.7 (případně verzi 7).

Pro využívání možností a funkcí aplikačního programovacího rozhraní firmy SciEngines je zapotřebí přiřadit k projektu také knihovnu (soubor typu JAR neboli Java Archive, který je založený na kompresním principu ZIP), která obsahuje odpovídající balíčky jmenných prostor a umožní je využívat ve zdrojovém kódu (soubor SciEngines_API_java.jar).

Jako výstupní soubor je pak vhodné, v případě konzolové aplikace, nastavit spustitelný soubor typu JAR, který je možné spustit z příkazového terminálu.

```
java -jar cesta_k_souboru/soubor.jar
```

Úsek kódu 5.1 – Spuštění spustitelného souboru typu JAR

Výstupní aplikace dále potřebuje pro využívání funkcí aplikačního programovacího rozhraní firmy SciEngines tzv. sdílenou knihovnu, která je reprezentována souborem typu SO (soubor libSciEngines_API_java_linux64.so). Cestu k této knihovně je nutné přidat do systémové proměnné, která obsahuje souborové cesty, kam spuštěné aplikace přistupují při hledání knihoven. V případě, že knihovna není nalezena, výsledná aplikace se ukončí při prvním zavolání kterékoliv funkce z aplikačního programovacího rozhraní firmy SciEngines. Cestu do složky s knihovnou je možné přidat pomocí příkazu níže.

```
export LD_LIBRARY_PATH = $LD_LIBRARY_PATH:/cesta/složka_obsahující_knihovnu
```

Úsek kódu 5.2 – Přidání složky s knihovnou do systémové proměnné

V případě, že se aplikace i po nalezení knihovny chová nepředvídatelně, je dobré zkontrolovat, zdali je FPGA periferie vůbec zapnutá. Kontrolu je možné provést skrze terminál zavoláním příkazu ***se_machine status***, který vypíše momentální stav periferie. V případě, že je periferie vypnutá, je nutné ji zapnout pomocí příkazu ***se_machine start***. V případě jiného stavu je možné periferii také restartovat pomocí příkazu ***se_machine restart***. Pokud nastaly jiné obtíže související s FPGA periferií, je možné použít také příkaz ***se_machine test***, který vyvolá hardwarový test, jehož výsledek napoví, zda je chyba na straně periferie.

5.4.1.2 Návrh aplikace

V připraveném projektu je třeba vytvořit třídu, která implementuje metodu ***main***, která je hlavní metodou aplikace a je zavolána ihned po jejím spuštění. V této metodě bude implementován celý náš program.

První krokem hostitelské aplikace je alokace FPGA periférie. Pro tento účel si vytvoříme instanci třídy ***SeOptions***, ve které nastavíme chování periferie. Instanci třídy ***SeOptions*** pak předáme jako vstupní parametr metodě ***se_allocMachine***, které navíc předáme také index periférie. Vzhledem k tomu, že metoda ***se_allocMachine*** může vyhodit výjimku, třeba v případě, že index periférie nebude validní, musí být metoda uzavřena v bloku ***try-catch***. Protože všechny následující metody jsou ošetřeny stejnou výjimkou (***SeApiException***), můžeme do stejného bloku ***try-catch*** uzavřít i zbytek programu, který využívá takto ošetřené metody.

```
SeOptions options = new SeOptions(
                    SeOptions.SeWriteBehavior.se_write_async,
                    SeOptions.SeRoutingMethod.se_routing_normal);

se_allocMachine(machineIndex, options);
```

Úsek kódu 5.3 – Alokace FPGA periférie

Pro čistotu kódu je dobré provést před alokací periférie kontrolu, zda je index periférie validní. Kontrolu provedeme pomocí metody ***se_getMachineCount***, která navrací počet dostupných periférií. V případě, že je index menší než počet dostupných periférií, index je validní. V opačném případě by se měl program ukončit nebo zakročit jiným způsobem.

```
int machineCount = se_getMachineCount();

if (machineIndex >= machineCount)
{
    System.out.println("Machine does not exist!");
    return;
}
```

Úsek kódu 5.4 – Kontrola validity indexu periférie

Dalším krokem je naprogramování jednotlivých FPGA pomocí logického návrhu, který byl vytvořen v jazyce VHDL s použitím aplikačního programovacího rozhraní firmy SciEngines a byl převeden na soubor typu BIT.

Než začneme s programováním jednotlivých FPGA, je třeba znát jejich počet, abychom je mohli jednotlivě adresovat. Počet dostupných slotů získáme pomocí metody ***se_getSlotCount***, která má jako vstupní parametr index periférie. Počet jednotlivých FPGA na daném slotu pak můžeme zjistit pomocí metody ***se_getFPGACount***, která má jako vstupní parametry index periférie a index slotu. Nicméně, vzhledem k tomu, že víme, že na každém slotu je karta s 8 FPGA, stačí nám počet dostupných slotů znásobit osmi. V případě, že chceme naprogramovat všechna FPGA stejným logickým návrhem, je možné je adresovat pomocí konstant ***SE_ADDR_SLOT_ALL*** a ***SE_ADDR_FPGA_ALL***.

I když použijeme k naprogramování všech FPGA konstanty, celkové počty slotů a FPGA se budou hodit v dalších částech programu, kdy je budeme využívat k adresaci.

```
int slotCount = se_getSlotCount(machineIndex);
int maxFPGACount = slotCount * 8;
```
Úsek kódu 5.5 – Získání počtu dostupných slotů a celkového počtu dostupných FPGA

Samotné naprogramování pak probíhá pomocí metody ***se_program***, která má jako vstupní parametry index periférie, instanci třídy ***SeAddress*** reprezentující adresu FPGA, textový řetězec obsahující cestu k souboru typu BIT a časový limit v milisekundách. Při vytváření instance třídy ***SeAddress*** použijeme jako vstupní parametry index kontroléru, index slotu, index FPGA a index registru. Index kontroléru a index registru je podle dokumentace nulový.

```
int controllerIndex = 0;
int registerIndex = 0;

SeAddress address = new SeAddress(controllerIndex, SE_ADDR_SLOT_ALL,
                    SE_ADDR_FPGA_ALL, registerIndex);

se_program(machineIndex, address, "logicCircuit.bit", 1000);
```
Úsek kódu 5.6 – Naprogramování všech FPGA logickým návrhem

Následujícím krokem je již samotný cíl vyvíjené aplikace, tedy zpracování dat na cílových FPGA pomocí zápisu a čtení. Princip implementace této části závisí hlavně na tom, jaký problém aplikace řeší a jak velkou část řeší hostitelská část.

Vzhledem k cíli této práce zde bude popsán algoritmus pro paralelní zpracování vstupních dat a rozdělování úloh mezi jednotlivá FPGA. Pro vývoj a práci se superpočítačem Rivyera není nutné využívat právě tento algoritmus. Záleží pouze na vývojáři, jak danou implementaci vyřeší.

Prvním krokem algoritmu je zjištění počtu vstupních úloh ke zpracování. Algoritmus pak připraví pole vláken o velikosti počtu vstupních úloh nebo, v případě velkého počtu úloh, o velikosti celkového počtu dostupných FPGA. Vedle toho je také připraveno pole pravdivostních hodnot o stejné velikosti, které bude určovat obsazenost daného vlákna.

```
taskCount = 80;

if (taskCount > maxFPGACount) {
    threadArray = new Thread[maxFPGACount];
    assignedThreadArray = new boolean[maxFPGACount];
}
else {
    threadArray = new Thread[taskCount];
    assignedThreadArray = new boolean[taskCount];
}
```
Úsek kódu 5.7 – Algoritmus pro přiřazování úloh vláknům část 1

Aplikace během zpracovávání poběží na tolika vláknech, kolik jich bude v připraveném poli, a navíc bude udržovat své hlavní vlákno, které bude očekávat dokončení všech úloh a nakonec se postará o dokončení programu. Udržování hlavního vlákna je zajištěno pomocí cyklu ***while*** a kontroly stavu zpracování úloh.

Proces přiřazování úloh jednotlivým vláknům a kontrola stavu jsou řízeny pomocí několika statických proměnných, které mají zajištěno, že k nim v jednu chvíli přistupuje pouze jedno vlákno. Hlavní vlákno, v cyklu ***while***, prochází pomocí cyklu ***for*** pole pravdivostních hodnot, které označuje, zdali je dané vlákno, a tedy i FPGA, obsazeno. Pokud vlákno obsazeno není, je mu přiřazena nová úloha a cyklus pokračuje dále. Zároveň je kontrolováno, kolik úloh již bylo přiřazeno a kolik zpracováno a podle toho je cyklus ***while*** na hlavním vlákně ukončen.

```
while (processedTaskCount < taskCount) {
    for (int fpga = 0; fpga < threadArray.length; fpga++) {
        if(currentTaskIndex < taskCount) {
            if (!assignedThreadArray[fpga]) {
                assignedThreadArray[fpga] = true;
                final int workTaskIndex = currentTaskIndex++;
                final int currentFPGAIndex = fpga;

                threadArray[fpga] = new Thread(
                        new Runnable()
                        {
                            @Override
                            public void run() {
                                // Preprocess data here!
                                // Write and read data from FPGA here!

                                // Unlock FPGA Thread
                                threadLock.lock();
                                try {
                                    processedTaskCount++;
                                    assignedThreadArray[currentFPGAIndex] = false;
                                }
                                finally {
                                    threadLock.unlock();
                                }

                                // Kill thread
                                Thread.currentThread().interrupt();
                            }
                        });
```

```
                // Start thread
                 threadArray[fpga].start();
            }
        }
        else {
            break;
        }
    }

    // Sleep for keep thread alive
    try {
        Thread.sleep(1);
    }
    catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Úsek kódu 5.8 – Algoritmus pro přiřazování úloh vláknům část 2

Kroky předzpracování dat se zaobírají rozdělením vstupních dat na datové pakety, které budou odeslány na cílová FPGA. Datové pakety jsou uloženy prostřednictvím třídy ***ByteBuffer***, které je nastavena velikost jednoho pole představujícího paket na 64 bitů. Celá instance této třídy, naplněná daty, je pak předána metodě ***se_write***, která data zapíše na cílové FPGA, které je určeno parametrem adresy prostřednictvím instance třídy ***SeAddress***.

Pokud pracujeme s 32bitovými daty, je výhodné uložit do jednoho paketu dvě hodnoty, jak je to ukázáno v úseku kódu níže. Ovšem ten počítá pouze se sudým počtem vstupních hodnot.

```
int[] data = new int[1000];

int dataLength = data.length;
int payloadByteLength = dataLength * (Integer.SIZE / Byte.SIZE);
int payloadLength = payloadByteLength / (Long.SIZE / Integer.SIZE);

ByteBuffer payload =
ByteBuffer.allocateDirect(payloadByteLength).order(ByteOrder.nativeOrder());

int index = 0;
for (int i = 0; i < dataLength; i += 2)
    payload.asLongBuffer().put(index++,
    (((long)data[i]) << 32) | (data[i + 1] & 0xffffffffL));
```

Úsek kódu 5.9 – Inicializace a naplnění třídy ByteBuffer dvěma hodnotami na paket

Předzpracovaná data v podobě instance třídy ***ByteBuffer*** je již možné zapsat do cílového FPGA a počkat na odpověď.

```
address = new SeAddress(controllerIndex, 0, 0, registerIndex);
se_write(machineIndex, address, payload, payloadLength, 30000);
```

Úsek kódu 5.10 – Zápis dat na FPGA na prvním slotu a jeho první pozici

Vyčtení zpracovaných hodnot z FPGA je pak zajištěno pomocí metody ***se_read***, která má dva čtecí módy. Data jsou navrácena opět v instanci třídy ***ByteBuffer***, kterou do metody vložíme jako vstupní parametr.

```
address = new SeAddress(controllerIndex, 0, 0, registerIndex);
se_read(machineIndex, address, payload, payloadLength, SE_READ_ACTIVE, 30000);
```

Úsek kódu 5.11 – Vyčtení dat z FPGA na prvním slotu a jeho první pozici

V posledních krocích je třeba odprogramovat všechna FPGA a uvolnit periférii. Pro odprogramování je připravena funkce ***se_deprogram***, která má jako vstupní parametry index periférie a adresu v podobě instance třídy ***SeAddress***.

```
SeAddress address = new SeAddress(controllerIndex, SE_ADDR_SLOT_ALL,
                    SE_ADDR_FPGA_ALL, registerIndex);

se_deprogram(machineIndex, addr);
```

Úsek kódu 5.12 – Odprogramování všech FPGA

Uvolnění periférie je pak možné provést pomocí metody ***se_freeMachine***, která má jako vstupní parametr index periférie. V případě, že jsme celý program uzavřeli do bloku ***try-catch***, za touto metodou jej můžeme ukončit.

```
se_freeMachine(machineIndex);
```

Úsek kódu 5.13 – Uvolnění FPGA periférie

5.4.1.3 Kompatibilita s webovým rozhraním

Vzhledem k tomu, že práce se zabývá vzdáleným ovládáním superpočítače Rivyera skrze webové rozhraní, znamená to, že právě hostitelská aplikace musí být ovladatelná prostřednictvím webového rozhraní. Webovým rozhraním v této práci se rozumí webový server pracující na protokolu HTTP, který je bezestavový. K tomuto účelu však musí být hostitelská aplikace uzpůsobena.

Aplikace bude na serveru spouštěna prostřednictvím jazyka PHP, který je schopen ji spustit skrze příkazový terminál a navrátit její textový výstup. Ovšem není schopen zasahovat za běhu programu nebo odchytávat její ukončovací kódy.

Jako vstupy do aplikace je možné použít vstupní argumenty při spouštění skrze terminál. Jako výstup však můžeme považovat pouze textový výstup aplikace. Všechny výstupy, které tedy chceme získat, se budou nacházet v navráceném textu nebo budou pevně stanoveny. Pro tyto účely je v aplikaci naimplementována vlastní třídy Loggeru, který do textu vkládá s běžným textem nezaměnitelné značky, které obklopují výstupní hodnoty. Můžeme si tak například navrátit ukončovací kód aplikace, název výstupního souboru či složky obsahující výstupní data.

Příklady značek: „***#RETURN_VALUE#***“, „***#OUTPUT_FILENAME#***“, „***#EOL#***“.

5.4.2 Logický návrh hosta (VHDL)

Logický návrh hosta má za úkol pouze obsloužit vstupní registr, zpracovat přijatá data a předat je na výstupní registr. Vlastní návrh řešení je zjednodušen, jak jen to bylo možné. Vývojář se stará pouze o implementace modulu, který je pak instanciován na hlavním logickém návrhu, který byl vytvořen firmou SciEngines.

5.4.2.1 Prerekvizity

Než začneme s vývojem logického návrhu, je třeba mít připraveno vývojové prostředí ISE od firmy Xilinx, které podporuje programovatelné hradlové pole Spartan-6 XC6SLX150. Dále je třeba mít licenci k plné verzi produktu, která povoluje vytváření souborů BIT právě pro výše zmíněné programovatelné hradlové pole.

Pokud máme vývojové prostředí připraveno, je zapotřebí vytvořit projekt. Máme několik možností. Můžeme projekt vytvořit ručně, ale tento přístup je poměrně složitý z důvodů toho, co všechno je v něm třeba nastavit, je zde vysoké riziko chyby. Dále je možné použít grafickou aplikaci dodanou firmou SciEngines, která se jmenuje ProjectCompass a požadovaný projekt vygeneruje. Asi nejjednodušším a nejvariabilnějším způsobem je použití autorem vytvořené aplikace RivyeraISEProjectGenerator (soubor RivyeraISEProjectGenerator.jar), která pomocí připravené šablony, vytvoří projekt s požadovaným názvem. Vytvořený projekt je pak okamžitě připraven k použití.

```
java –jar RivyeraISEProjectGenerator.jar nazev_projektu
```

Úsek kódu 5.14 – Příkaz pro vytvoření ISE projektu s názvem nazev_projektu

5.4.2.2 Logický návrh

Samotný vývoj logického návrhu je relativně jednoduchý. V projektu, vygenerovaném aplikací RivyeraISEProjectGenerator, je soubor v jazyce VHDL s názvem „nazev_projektu_main.vhd“, který obsahuje uživatelský modul, který má předpřipravené sekce, do kterých vývojář může vkládat své vlastní kódy, a který bude po sestavení projektu instanciován v logickém návrhu firmy SciEngines.

Logický návrh uživatelského modulu má již sám o sobě v podstatě pevnou strukturu, která v případě, že se vývojář bude držet v předpřipravených sekcích, zajistí správnou funkci návrhu. Sekcí se rozumí ohraničený úsek ve zdrojovém kódu, který je vyznačen pomocí komentářů.

Uživatelský modul má nadefinovány signály, se kterými vývojář bude pracovat a bude skrze ně obsluhovat vstupní a výstupní registr daného FPGA. Vše, co musí vývojář v logickém návrhu udělat, je obsloužit vstupní a výstupní registry podle toho, jaký příkaz byl skrze aplikační programovací rozhraní právě zrovna zadán. Příkazy jsou dva a to příkaz zápisu, který byl vyvolaný hostitelskou aplikací při použití metody ***se_write***, a příkaz čtení, který byl vyvolán hostitelskou aplikací při použití metody ***se_read***. V logickém návrhu je momentální příkaz reprezentován signálem ***api_i_tgt_cmd_in***, který je možné porovnat s konstantami z aplikačního programovacího rozhraní ***CMD_WR*** a ***CMD_RD***, které reprezentují dva výše zmíněné příkazy.

Příchozí data jsou uložena v signálu ***api_i_data_in*** a po jejich vyčtení by měl vývojář potvrdit jejich přečtení pomocí bitového příznaku v signálu ***api_i_rd_en***, kterému nastaví logickou hodnotu 1. Výstupní data ukládá vývojář do signálu ***api_o_data_out*** a po jejich uložení musí potvrdit zapsání dat pomocí bitového příznaku v signálu ***api_o_wr_en***, kterému nastaví logickou hodnotu 1.

Ve které části návrhu a kdy bude vývojář zpracovávat přijatá data, záleží na něm a na právě řešeném problému. Přístup k implementaci řešení bude rozhodně rozdílný v situaci, kdy se budou zpracovávat vzájemně nezávislé jednotlivé hodnoty a kdy se budou zpracovávat vzájemně závislá obrazová data jako celek.

5.5 Testovací aplikace

Pro testovací účely a pochopení principů funkce a návrhu pro superpočítač Rivyera byla vytvořena jednoduchá aplikace, která převádí obrazová data z barevného modelu RGB do odstínů šedé. V aplikaci, čímž se rozumí aplikace hostitele v jazyce Java a logický návrh hosta v jazyce VHDL, se hostitelská část stará o načtení obrázku, rozložení na jednotlivé pixely a odeslání do části hosta, která provádí převod barev na odstíny šedé.

Vzhledem k jednoduchosti celé aplikace probíhá zpracování obrazových dat pouze na jednom FPGA a obrazová data jsou do něj odesílána najednou, vždy po dvou pixelech na paket.

```
r <= to_integer(unsigned(api_i_data_in(23 downto 16)));
g <= to_integer(unsigned(api_i_data_in(15 downto 8)));
b <= to_integer(unsigned(api_i_data_in(7 downto 0)));

r_f <= (r * redFactor) / percents;
g_f <= (g * greenFactor) / percents;
b_f <= (b * blueFactor) / percents;
gray <= std_logic_vector(to_unsigned(r_f + g_f + b_f, 8));

pixel <= api_i_data_in(31 downto 24) & gray & gray & gray;
```

Úsek kódu 5.15 – Logický návrh v jazyce VHDL pro převod jednoho pixelu z RGB na odstín šedé

5.6 Ukázková aplikace

Pro ukázkovou aplikaci byl vybrán proces vytváření digitálně rekonstruovaných radiologických obrazů, kdy je ze série CT snímků vytvořena nová projekce z určitého úhlu. Samotný princip vytváření DRR není tak složitý, ovšem vstupní data, která jsou zastoupena velkým počtem snímků CT, mohou problematiku zkomplikovat.

Vzhledem k cílům práce nejsou jako vstupní data použity soubory typu DICOM, ale pouze obrazová data ve formátu PNG. Použití pouze obrazových dat je užito hlavně z důvodů anonymizace a zjednodušení předzpracování vstupních dat. Takto upravená data mají také menší velikost, která je výhodná při vzdáleném zpracování skrze web.

Princip aplikace spočívá v načtení obrazových dat hostitelskou aplikací, odeslaní celého snímku do cílového FPGA, které provede rotaci o požadovaný úhel a simulaci průchodu rentgenového záření a navrátí hostitelské aplikaci výstupní data.

Vstupní série CT obrazů bývá poměrně obsáhlá a většinou čítá několik stovek snímků. Aby byla operace vytváření DRR obrazů uspíšena je ukázková aplikace paralelizována a ke zpracování jsou využita všechna dostupná FPGA. Přidělování jednotlivých snímků volným FPGA je zajištěno algoritmem pro rozdělování úloh, který je popsán výše.

Vzhledem k tomu, že se jedná o obrazy v odstínech šedi, je v jednom datovém paketu odesíláno několik pixelů najednou. V případě ukázkové aplikace se jedná o osm pixelů najednou. Tento aspekt přenosu je nutné přizpůsobit odpovídajícímu bitovému rozsahu vstupních dat, která jsou však v našem případě u PNG obrazů pouze 8bitová. Nicméně reálná data z CT snímků bývají často alespoň 12bitová.

Pro rotaci obrazu je použit algoritmus bilineární interpolace, který je navržen v jazyce VHDL. Algoritmus klasicky pracuje s reálnými čísly, která v případě, že nejsou konstantní, není možné na FPGA syntetizovat. Tento problém je však možné obejít tak, že všechna reálná čísla posuneme o několik řádů (podle zvolené přesnosti výpočtů), budeme s nimi pracovat jako s čísly celými a výsledky pak posuneme o odpovídající počet řádů níže. Musíme však dávat pozor na zvýšení či snížení řádů při operacích krát a děleno a tyto výkyvy kompenzovat.

```
if (X = WIDTH_HEIGHT) then
   Y <= Y + 1;
   X <= 0;
end if;

rotatedX <= ((X - centerXY) * cosine)
            - ((Y - centerXY) * sine)
            + centerXY * precision;

rotatedY <= ((X - centerXY) * sine)
            + ((Y - centerXY) * cosine)
            + centerXY * precision;
```

```
rotX <= rotatedX / precision;
rotY <= rotatedY / precision;

if (rotX >= 0 and rotY < WIDTH_HEIGHT - 1
    and rotY >= 0 and rotY < WIDTH_HEIGHT - 1) then

   p1 <= inputImage((rotY * WIDTH_HEIGHT) + rotX);
   p2 <= inputImage(((rotY + 1) * WIDTH_HEIGHT) + rotX);
   p3 <= inputImage((rotY * WIDTH_HEIGHT) + (rotX + 1));
   p4 <= inputImage(((rotY + 1) * WIDTH_HEIGHT) + (rotX + 1));

   f1 <= (p1 + ((p3 - p1) * ((rotatedX - (rotX * precision)) / precision))
          / precision);
   f2 <= (p2 + ((p4 - p2) * ((rotatedX - (rotX * precision)) / precision))
          / precision);

   outputImage((Y * WIDTH_HEIGHT) + X) <=
          (f1 + ((f2 - f1) * (rotatedY - (rotY * precision)) / precision)
           / precision);
end if;

X <= X + 1;
```

Úsek kódu 5.16 – Logický návrh v jazyce VHDL pro rotaci obrazu skrze bilineární interpolaci

Simulace průchodu rentgenové záření je zajištěna jednoduchou operací získání maxima ve všech řádcích obrazové matice. Pro vyšší rychlost manipulace s obrazovými daty je obrazová matice již v hostitelské aplikaci tzv. zploštěna (z angl. slova flattening) z dvourozměrné matice na jednorozměrný vektor.

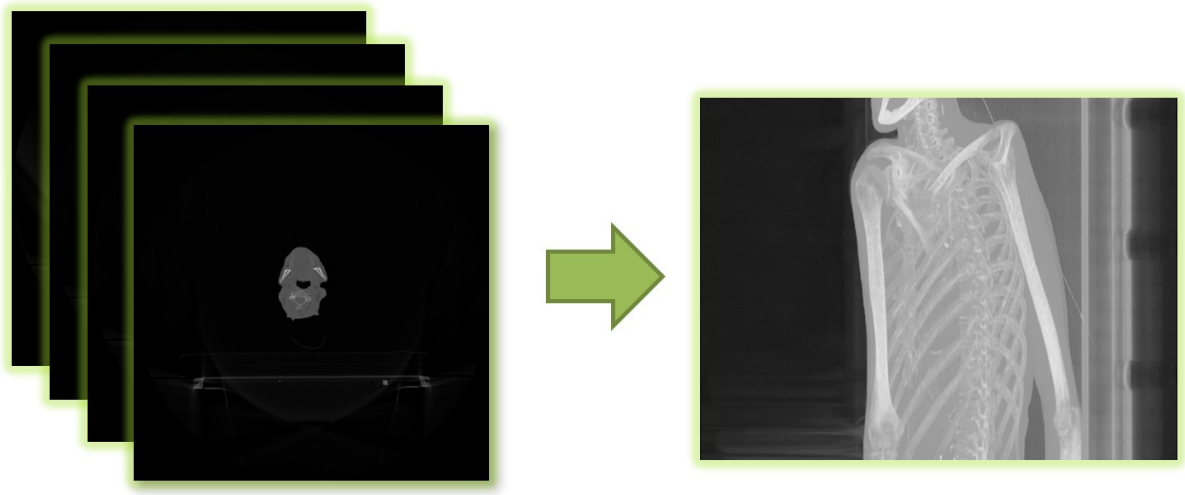
```
if (X = WIDTH_HEIGHT) then
   Y <= Y + 1;
   X <= 0;
   lineMax <= 0;
end if;

if (outputImage((Y * WIDTH_HEIGHT) + X) > lineMax) then
      lineMaxs(Y) <= outputImage((Y * WIDTH_HEIGHT) + X);
end if;

X <= X + 1;
```

Úsek kódu 5.17 – Logický návrh v jazyce VHDL pro simulaci průchodu RTG záření tkání

Finálním výstupem každého FPGA je vektor hodnot o velikosti vstupní šířky snímků, který představuje jeden řádek výsledného DRR obrazu. Po dokončení zpracování všech snímků jsou jednotlivé řádky spojeny a převedeny na výsledný obraz ve formátu PNG. Výstupní snímek má pak původní šířku, ale jeho výška je dána počtem vstupních CT snímků.



Obrázek 5.2 – Příklad vstupních a výstupních dat

5.7 Webové rozhraní

Pro webové rozhraní je použit webový server Apache HTTP Server verze 2.4.10. Webové stránky jsou dynamicky generovány pomocí jazyka PHP prostřednictvím webového frameworku Nette, který umožňuje snadný vývoj a ladění. Framework zajišťuje základní zabezpečení a umožňuje přidávat rozšíření v podobě přídavných balíčků.

Framework Nette využívá architekturu MVC (Model-View-Controller), která odděluje výstupní vrstvu od vrstvy logické. V principu se uživatel webu připojí na webový server prostřednictvím URL adresy, podle které se tzv. kontrolér dotáže logického modelu na data, která jsou mu vzápětí navrácena. Kontrolér data upraví do příslušné podoby a uloží je do šablony, která je následně vykreslena uživateli.

Framework využívá pro popis modelu a kontroléru jazyk PHP a pro popis šablon je používán jazyk Latte v kombinaci s jazyky HTML, CSS a JavaScript.

5.7.1 Vzhled a struktura webu

Veškerý obsah webového rozhraní je pro úplnou jednoduchost reprezentován jedinou webovou stránkou, která dynamicky mění obsah zobrazovaných částí. Protože je použita MVC architektura, jsou jednotlivé zobrazované části podmíněně generovány kontrolérem a není tedy možné je nalézt ve zdrojovém kódu webové stránky.

Webový protokol HTTP je bezstavovým systémem, a proto jsou všechna data uživatele ztraceny s každým obnovením stránky, což v MVC architektuře znamená při každém odeslání formuláře. Data

jednotlivých akcí uživatele jsou proto ukládána na serveru prostřednictvím relací, které jsou schopny data udržet po stanovenou dobu nebo do uzavření prohlížeče, kdy uživatel automaticky ukončí danou relaci.

Výsledný vzhled stránky je cíleně jednoduchý a prvky se přizpůsobují změnám velikosti prohlížeče, což umožňuje použití stránek také na mobilních zařízeních. Grafické provedení v zelené barvě může mít příjemný uklidňující účinek.

Obrázek 5.3 – Stránka pro nahrávání souborů zobrazovaná po přihlášení

5.7.2 Zabezpečení

Webové rozhraní je přístupné z interní sítě univerzity, a proto je momentálně zabezpečeno pomocí hesla. Veškerý webový obsah je logickou vrstvou webu poskytnut až po zadání správného hesla. Přístup více uživatelů je řešen pomocí relací, které ukládají jednotlivé stavy přihlášených uživatelů na straně serveru. V případě neaktivity uživatelů delší jak 15 minut relace vyprší.

V případě, že by se dva uživatelé najednou pokusili spustit aplikaci na superpočítači Rivyera, podaří se to prvnímu uživateli v pořadí a druhému aplikace skončí chybou. Přístup více procesů je totiž ošetřen již v aplikačním programovacím rozhraní firmy SciEngines. V budoucnu bude tato problematika ošetřena databázovou vrstvou a uživatel nebude schopen aplikaci spustit.

Obrázek 5.4 – Přihlašovací stránka webového rozhraní

5.7.3 Plány do budoucna

Do budoucna je v plánu přidat databázovou vrstvu, která umožní pokročilejší možnosti uživatelské interakce. Mezi dostupnými funkcemi bude například pokročilý uživatelský systém s přístupovými právy, ošetření vícenásobného přístupu, různé statistiky nebo systém pro přidávání vlastních aplikací.

5.7.4 Vzdálené ovládání aplikací superpočítače Rivyera

Výsledné vzdálené ovládání superpočítače Rivyera je zprostředkováno pomocí webového rozhraní, které umožňuje na počítači serveru spustit aplikaci, která přímo pracuje s FPGA periférií. Proces spuštění aplikace na superpočítači Rivyera probíhá v několika níže popsaných krocích.

5.7.4.1 Nahrání dat na server

Prvním krokem po přihlášení do webového rozhraní je nahrání vstupních dat, která chce uživatel zpracovat. Data jsou prostřednictvím webového prohlížeče vybrána a odeslána na server, kde jsou uložena do připravené složky momentálně přihlášeného uživatele. Složky pro nahrané soubory jsou odlišeny názvem, který je stejný jako jedinečné identifikační číslo relace. Uvnitř této složky aktuální relace jsou další složky označené pouze čísly, které určují pořadí nahraného souboru či souborů. Po ukončení relace je celá složka nahraných souborů smazána.

Výše popsaný adresářový systém je takto strukturován z důvodů jednoduššího předávání souborové cesty adresáře cílové aplikaci. Stránka a formulář pro nahrávání souborů je zobrazen na obrázku 5.3. Po odeslání formuláře pro nahrávání souborů jsou úspěšně nahrané soubory uživateli zobrazeny jako horizontální pás ikon s názvy souborů.

Obrázek 5.5 – Stránka výběru aplikace se seznamem nahraných souborů

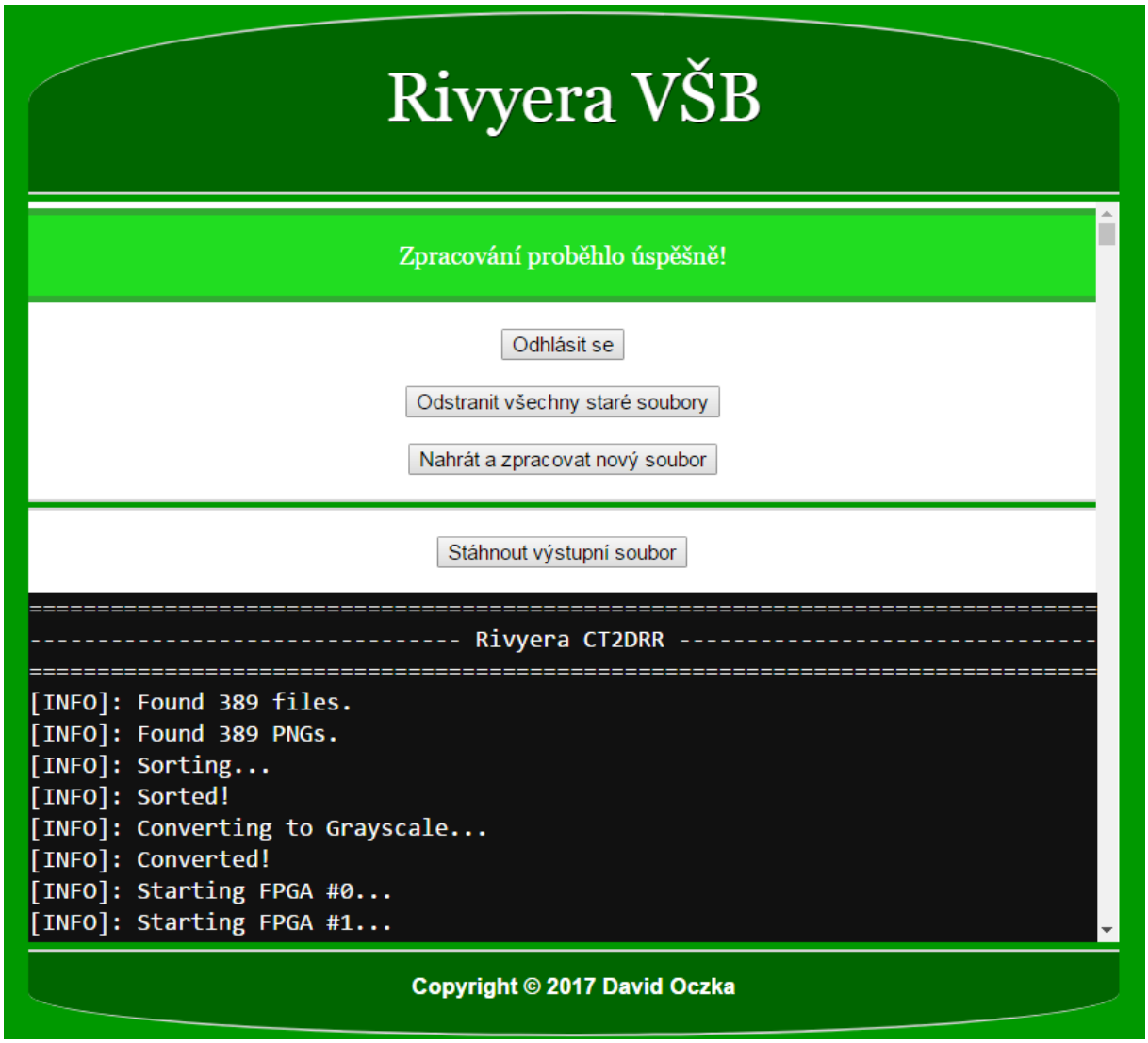
5.7.4.2 Výběr aplikace

Dalším krokem je jednoduchý výběr aplikace, která má vstupní data zpracovat. Momentálně jsou k dispozici dvě aplikace: převod barevného obrázku na černobílý a vytvoření DRR. Po odeslání formuláře webový server spustí prostřednictvím terminálu vybranou aplikaci, které jako vstupní argumenty uvedl cestu k adresáři obsahující vstupní data.

To, jaké argumenty budou vybrané aplikaci zadány, je momentálně pevně dáno ve zdrojovém kódu. V budoucnu bude tento přístup nahrazen dynamickým nastavením komponent formuláře pro danou aplikaci. Bude možné vybrat komponenty jako číselný posuvník, textové pole, datum a čas, zaškrtávací políčka a další.

5.7.4.3 Výpis a stažení výsledku

Po odeslání formuláře s vybranou aplikací bude prohlížeč uživatele čekat na odpověď, která přijde ve chvíli, kdy spuštěná aplikace navrátí výstup. Výpis z aplikace je pak skrze textové značky analyzován a výsledek je vypsán uživateli. V této fázi zpracování se uživateli také zobrazí tlačítko pro stažení výsledku. K dispozici je také tlačítko pro navrácení na stránku pro nahrávání souborů, ze které může uživatel celý proces opakovat.



Obrázek 5.6 – Stránka s výpisem aplikace a stažením souboru

6 Závěr

Hlavním cílem této práce bylo navrhnout a popsat řešení vzdálené obsluhy superpočítače Rivyera, které by umožnovalo přístup ze školní sítě. Vzdálená obsluha je řešena pomocí webového serveru, na kterém běží webového rozhraní. Je tak možné vzdáleně spouštět a interagovat s hostitelskými aplikacemi superpočítače Rivyera. Implementován je také obousměrný datový tok mezi uživatelským počítačem a počítačem serveru, který je zprostředkován skrze nahrávací a stahovací webové formuláře.

V úvodní části práce je zpracována rešeršní práce na téma „Biomedicínské výpočty s použitím FPGA“, která uvádí do problematiky složitých biomedicínských výpočtů a jejich řešení s použitím programovatelných hradlových polí. Rešeršní práce se zabývá několika články z problematiky elektrokardiografie, elektroencefalografie a z oblasti zpracování obrazu.

Další část práce uvádí do tématu programovatelných hradlových polí a zabývá se popisem programovatelného hradlového pole typu Spartan-6 od firmy Xilinx, které je využito jako výpočetní jednotka v superpočítači Rivyera.

Mimo návrh a popis řešení vzdálené obsluhy se práce také zaměřuje na popis parametrů a principu funkce superpočítače Rivyera. Jsou zde popsány úlohy hostitelské aplikace a také úlohy logického návrhu hosta. K dispozici je popis aplikačního programovacího rozhraní firmy SciEngines, které je používáno pro vývoj hostitelských aplikací superpočítače Rivyera.

V další kapitole práce je pak obecný popis postupu pro návrh řešení určité problematiky s použitím superpočítače Rivyera, kdy je zapotřebí daný výpočetní problém vhodně rozdělit mezi hostitelskou část běžící na počítači serveru a část hosta běžící na FPGA periférii.

Jako příklad a případná pomoc při vývoji vlastních aplikací byla vytvořena ukázkové aplikace, která se skládá ze dvou částí, čímž se rozumí hostitelská aplikace v jazyce Java a logický návrh hosta v jazyce VHDL. Obě dvě části jsou v práci podrobně popsány a jsou zde také přiloženy důležité úseky zdrojového kódu. Aplikace jako celek je přizpůsobena spuštění skrze webové rozhraní a je z něj přístupná autorizovanému uživateli webu, který se prokázal heslem.

Ukázková aplikace se zabývá zpracováním CT snímků a vytváří z nich digitálně rekonstruované projekce z určitého úhlu. Je tím tedy simulováno RTG vyšetření, jehož výsledek je rekonstruován prostřednictvím vstupních CT snímků.

V poslední části je popsáno webové rozhraní, které bylo implementováno prostřednictvím jazyků PHP, HTML, CSS, Latte a JavaScript ve webovém frameworku Nette, které běží na webovém serveru Apache. Web je přístupný po zadání hesla, načež má uživatel možnost nahrát soubory a vybrat aplikaci, kterou budou data na serveru zpracována. Vzhled webového rozhraní je koncipován jednoduše a přehledně, s ohledem na různou velikost webového prohlížeče, což umožňuje zobrazení také na mobilních zařízeních.

Do budoucna je v plánu vytvořit mnohem komplexnější rozhraní s databázovou vrstvou, které bude umožňovat větší uživatelskou kontrolu, díky které budou uživatelé s odpovídajícími oprávněními moci nahrávat vlastní aplikace a dynamicky tak měnit možnosti vzdálené obsluhy superpočítače Rivyera.

Literatura

[1] HŘEBÍČEK, Jiří. *Vědecké výpočty v matematické biologii*. Brno: Akademické nakladatelství CERM, 2012. ISBN 978-80-7204-781-9.

[2] HŘEBÍČEK, Jiří a Jan ŽIŽKA. *Vědecké výpočty v biologii a biomedicíně*. Brno: Masarykova universita, 2007. Dostupné také z: http://portal.med.muni.cz/download.php?fid=502.

[3] CHOU, Chia-Ching, Tsan-Yu CHEN a Wai-Chi FANG. FPGA implementation of EEG system-on-chip with automatic artifacts removal based on BSS-CCA method. In: 2016 IEEE Biomedical Circuits and Systems Conference (BioCAS) [online]. IEEE, 2016, s. 224-227 [cit. 2017-03-29]. DOI: 10.1109/BioCAS.2016.7833772. ISBN 978-1-5090-2959-4. Dostupné z: http://ieeexplore.ieee.org/document/7833772/

[4] OZPOLAT, Erman, Baris KARAKAYA, Turgay KAYA a Arif GULTEN. FPGA-based digital Filter Design for Biomedical Signal. In: *2016 XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH)* [online]. IEEE, 2016, s. 70-73 [cit. 2017-03-29]. DOI: 10.1109/MEMSTECH.2016.7507523. ISBN 978-617-607-913-2. Dostupné z: http://ieeexplore.ieee.org/document/7507523/

[5] HASSEN, Amina El, Aymeric HISTACE, Mehdi TEROSIET a Olivier ROMAIN. FPGA-based detection of QRS complexes in ECG signal. In: *2015 Conference on Design and Architectures for Signal and Image Processing (DASIP)* [online]. IEEE, 2015, s. 1-7 [cit. 2017-03-29]. DOI: 10.1109/DASIP.2015.7367244. ISBN 978-1-4673-7738-6. Dostupné z: http://ieeexplore.ieee.org/document/7367244/

[6] ABDULLAH, Hadeel N. a Bassam H. ABD. A simple FPGA system for ECG R-R interval detection. In: *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)* [online]. IEEE, 2016, s. 1379-1382 [cit. 2017-03-30]. DOI: 10.1109/ICIEA.2016.7603800. ISBN 978-1-4673-8644-9. Dostupné z: http://ieeexplore.ieee.org/document/7603800/

[7] CHATTERJEE, Subarna, Ajoy Kumar RAY, Rezaul KARIM a Arindam BISWAS. Architecture Design for Median Filter. In: *2011 Third National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics* [online]. IEEE, 2011, s. 247-250 [cit. 2017-03-30]. DOI: 10.1109/NCVPRIPG.2011.59. ISBN 978-1-4577-2102-1. Dostupné z: http://ieeexplore.ieee.org/document/6133047/

[8] GRABOWSKI, K. a A. NAPIERALSKI. Hardware architecture for advanced image processing. In: *IEEE Nuclear Science Symposuim & Medical Imaging Conference* [online]. IEEE, 2010, s. 3626-3633 [cit. 2017-03-30]. DOI: 10.1109/NSSMIC.2010.5874488. ISBN 978-1-4244-9106-3. Dostupné z: http://ieeexplore.ieee.org/document/5874488/

[9] CHOI, Young Kyu, Jason CONG a Di WU. FPGA Implementation of EM Algorithm for 3D CT Reconstruction. In: *2014 IEEE 22nd Annual International Symposium on Field-Programmable*

Custom Computing Machines [online]. IEEE, 2014, s. 157-160 [cit. 2017-03-30]. DOI: 10.1109/FCCM.2014.48. ISBN 978-1-4799-5111-6. Dostupné z: http://ieeexplore.ieee.org/document/6861610/

[10] THOMAS, H M William a S C Prasanna KUMAR. Detection of a brain tumor using segmentation and morphological operatorsfrom MRI scan with FPGA. In: *2015 International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)* [online]. IEEE, 2015, s. 728-731 [cit. 2017-03-30]. DOI: 10.1109/ICATCCT.2015.7456979. ISBN 978-1-4673-9223-5. Dostupné z: http://ieeexplore.ieee.org/document/7456979/

[11] ŠŤASTNÝ, Jakub. *FPGA prakticky: realizace číslicových systémů pro programovatelná hradlová pole*. Praha: BEN - technická literatura, 2010. ISBN 9788073002619.

[12] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL*. Praha: BEN - technická literatura, 2006. ISBN 8073001985.

[13] KAŠÍK, Vladimír. Programování hradlových polí. Učební text a návody do cvičení. Ostrava: VŠB-TU Ostrava, 2012.

[14] PARNELL, Karen and Nick MEHTA. *Programmable Logic Design Quick Start Handbook.* 4th ed. [s.l.]: Xilinx Inc., 2003. 225 s. Dostupné z: http://www.xilinx.com/univ/beginnersbookjune2003ver2.pdf.

[15] XILINX. *Spartan-6 Family Overview*. 2011. Dostupné také z: https://www.xilinx.com/support/documentation/data_sheets/ds160.pdf.

[16] SCIENGINES. *RIVYERA S6-LX150 Datasheet: Interní firemní dokumentace*. 2013.

[17] SCIENGINES. *RIVYERA API Host-API (C/C++): Interní firemní dokumentace*. 2013.

[18] SCIENGINES. *RIVYERA API Host-API (Java): Interní firemní dokumentace*. 2013.

[19] SCIENGINES. *RIVYERA API Machine-API (VHDL): Interní firemní dokumentace*. 2013.

[20] SCIENGINES. *RIVYERA ActiveRead: Interní firemní dokumentace*. 2013.

[21] SCIENGINES. *RIVYERA Ping-Pong: Interní firemní dokumentace*. 2013.

[22] SCIENGINES. *RIVYERA Getting Started Guide: Interní firemní dokumentace*. 2012.

[23] SCIENGINES. *RIVYERA se_mon User Documentation: Interní firemní dokumentace*. 2012.

[24] SCIENGINES. *RIVYERA API Simulation Guide: Interní firemní dokumentace*. 2013.

[25] WIENBRANDT, Lars. *The FPGA-Based High-Performance Computer RIVYERA for Applications in Bioinformatics* [online]. s. 383 [cit. 2017-04-18]. DOI: 10.1007/978-3-319-08019-2_40. Dostupné z: http://link.springer.com/10.1007/978-3-319-08019-2_40.

[26] ASHENDEN, Peter J. *The designer's guide to VHDL*. 2nd ed. San Francisco, CA: Morgan Kaufmann, c2002. ISBN 1-55860-674-2.