Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: (Upisuje redakcija, ostaviti ovaj red)
DOI: (Upisuje redakcija, ostaviti ovaj red)

Implementacija vektorskog procesora baziranog na RISC-V setu instrukcija

Implementation of a vector processor based on a RISC-V instruction set

Nikola Kovačević, Vuk Vranjković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu prezentovan je 32-bitni vektorski procesor baziran na RISC-V setu instrukcija. Sistem je implementiran pomoću VHDL jezika za opis hardvera i namenjen je za soft-core primenu na FPGA platformama. Procesor je podeljen na dve celine, skalarno jezgro koje implementira RISC-V integer set instrukcija i vektorsko jezgro koje implementira RISC-V vektorski set instrukcija. Vektorsko jezgro je parametrizovano promenljivim brojem vektorskih linija, što omogućava korisniku da bira između performansi i ukupnog zauzeća resursa. Sistem je testiran na Zybo razvojnoj ploči, pri čemu je Vivado alat korišćen za njeno programiranje, analizu performansi i analizu utrošenih resursa.*

Ključne reči: *RISC-V, vektorski procesor, FPGA, Zybo.*

Abstract – *This paper presents a 32-bit vector processor based on the RISC-V instruction set. The system is implemented using VHDL hardware description language for soft-core applications on FPGA platforms. The processor is split into two parts, the scalar core, that implements the RISC-V integer instruction set, and the vector core that implements the RISC-V vector instruction set. The number of vector lanes inside the vector core is parametrized, so the user can make area-versus-performance trade-offs. The system was tested on a Zybo development board, using the Vivado tool to program it, and analyze resource utilization and performance.*

Keywords: *RISC - V, vector processor, FPGA, Zybo.*

1. UVOD

2004. godine Denardovo skaliranje (*eng. Dennard scaling*) [1] prestaje da važi i frekvencije rada procesora se od tada sve sporije povećavaju. Ta promena je mikroprocesorsku industriju primorala da pronalazi nova rešenja kako bi povećala performanse procesora i ono što se pokazalo kao prekretnica jeste stavljanje akcenta na paralelizam prilikom obrade podataka. Trenutno, najuspešnija i najzastupljenija arhitektura za paralelnu obradu podataka jeste GPU (*eng. Graphics processing unit*).

NAPOMENA:

Ovaj rad proistekao je iz master rada čiji mentor je bio dr Vuk Vranjković

No, Vektorski procesori, koji datiraju još od 60-tih godina prošlog veka, su jedna od arhitektura kod kojih je ova vrsta paralelizma takođe izražena, ali su do skora smatrani jako „skupim“. Jedan razlog je broj tranzistora, ali drugi, možda i bitniji, potreba za DRAM (*eng. Dynamic Random Accesss Memory*) memorijama koje mogu dovoljno brzo da „nahrane“ vektorski procesor podacima [1].

Napretkom tehnologije, potrebom za što većom paralelizacijom obrade podataka, sa što većom energetskom efikasnošću, pojavom RISC-V instrukcijskog seta, ovi procesori su ponovo skrenuli pažnju na sebe. Iz tih razloga ovaj rad se bavi analizom i implementacijom vektorskog procesora baziranog na RISC-V arhitekturi.

2. UVOD U VEKTORSKE PROCESORE

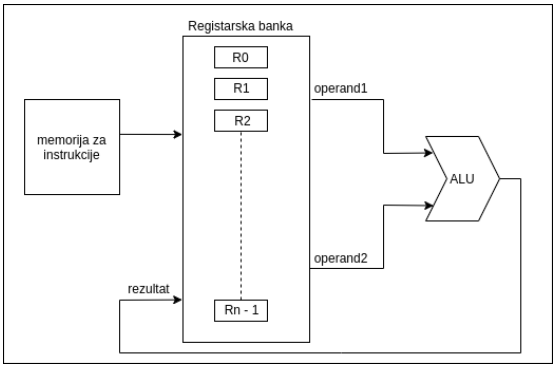
Kako bi povećali performanse, preko granica koje dopušta trenutna tehnologija izrade čipova, sistemske arhitekte pribegavaju različitim vrstama paralelizacije:

- Paralelizam na nivou instrukcija, skraćeno ILP (*eng. Instruction Level Parallelism*), omogućava istovremeno izvršavanje više instrukcija iz jednog sekvencijalnog skupa. Najjednostavniji primer su procesori sa protočnom obradnom [1], kod kojih pre nego što se završi egzekucija prethodne instrukcije, kreće se sa narednom.
- Paralelizam na nivou niti, skraćeno TLP (*eng. Thread Level Parallelism*), omogućava istovremeno izvršavanje instrukcija iz više odvojenih skupova. Najočigledniji primer su multiprocesorski sistemi kod kojih svaki procesor može da izvršava njemu dodeljen skup instrukcija.
- Paralelizam na nivou podataka, skraćeno DLP (eng. *Data Level Parallelism*), omogućava izvršavanje jedne iste operacije istovremeno nad nizovima elemenata. Primer su vektorski procesori koji jednu vektorsku instrukciju primenjuju na više podataka istovremeno.

Od prethodne tri vrste paralelizma, DLP se znatno bolje skalira, jer su podaci nad kojima vektorski procesor vrši određenu operaciju, na osnovu prihvaćene instrukcije, međusobno nezavnisni, dok kod arhitektura koje iskorišćavaju TLP i ILP, rešavanje zavisnosti između instrukcija zahteva dodatni hardver.

2.1. Vektorsko procesiranje

Princip rada skalarnih procesora predstavlja dobar uvod u vektorske procesore, te će stoga početak ove sekcije biti posvećen tome. Slika 1 ilustruje pojednostavljenu strukturu skalarnog procesora i na njoj su prikazane 3 ključne komponente: memorija za instrukcije, registarska banka i ALU (aritmetičko logička jedinica). Registarska banka unutar sebe skladišti određeni broj registara i na osnovu prihvaćene instrukcije jedan par tih registara biće pročitan i sproveden do aritmetičko logičke jedinice. Nad njima će se izvršiti određena operacija u zavisnosti od prihvaćenene instrukcije i rezultat će biti smešten u registarsku banku. Struktura vektorskih procesora je jako slična prethodno opisanoj, kao što se može videti na slici 2. Ključne komponente su iste, osim što je registarska banka zamenjena vektorskom registarskom bankom.

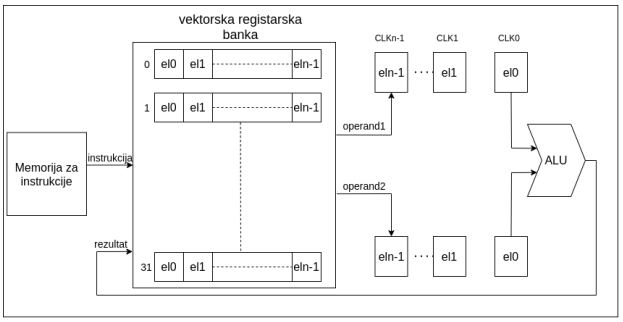


Slika 1: Pojednostavljena struktura skalarnog procesora

Razlika između te dve komponente jeste u tome što se registarska banka sastoji od registara određene širine (broj bita je određen arhitekturom procesora) dok se vektorska registarska banka sastoji od vektora, pri čemu svaki vektor unutar sebe sadrži određeni broj elemenata (širina pojedinačnog elementa je takođe određena arhitekturom). Prihvatom vektorske instrukcije, kao što se kod skalarnog procesora čitaju registri registarske banke i izvršava određena operacija nad njima, tako se kod vektorskog procesora, iz vektorske registarske banke, čitaju vektori i jedna ista operacija se izvršava nad elementima unutar njih. Unutar instrukcije se nalazi informacija kojim se vektorima pristupa i svakim taktom biće pročitan jedan par elemenata, počevši od elemenata na indeksu „*i*“. Nad njima će se izvršiti određena operacija i rezultat će biti smešten u određeni vektor unutar vektorske registarske banke.

2.2. Prednosti vektorskog seta instrukcija

Iz razloga što jedna kratka instrukcija može da opiše N operacija i da adresira 3N registarskih operanada, vektorski kod je kompaktan i neophodna propusna moć prilikom prihvata instrukcija iz memorije je mnogo manja. Vektorski set instrukcija umanjuje hardver neophodan prilikom dekodovanja vektorske instrukcije, jer se jedna vektorska instrukcija primenjuje N puta na *N* elemenata.Takođe, *N* operacija koje se izvode su međusobno nezavisne, te iz tog razloga nije neophodan hardver za detekciju zavisnosti.



Slika 2: Pojednostavljena struktura vektorskog procesora

Šablon koji vektorski procesor prati prilikom izvođenja operacija nad elementima vektorskog registra je regularan. To omogućava visok stepen paralelizma, jer bi vektorski procesor mogao da se implementira pomoću više paralelnih linija (*eng. Vector lanes*), pri čemu bi svaka linija vršila operacije nad jednim delom vektorskih elemenata. Vektorski set instrukcija može se dodati kao ekstenzija na već postojeći skalarni set.

2.3 RISC-V ISA(eng. Instruction Set Architecture)

Arhitektura skupa instrukcija (eng. ISA, Instruction Set Architecture) opisuje na koji način određeni procesor funkcioniše i koje su njegove mogućnosti. Ona opisuje registre koje će procesor imati kao i sve mašinske instrukcije koje će podržavati [2]. Iz tog razloga prilikom projektovanja procesora neophodno je odabrati određenu arhitekturu, čiji set ili podset instrukcija će biti podržan. U ovom radu odabrana je RISC-V ISA [3]. Ova arhitektura, koja je potekla sa Berkli (*eng. Berkeley*) univerziteta, je novi set instrukcija (ISA) koji je na početku bio zamišljen da podrži naučna istraživanja i edukaciju, ali za koji sada postoji nada da će postati arhitektura koja će biti besplatna i otvorena za sve industrijske implementacije.

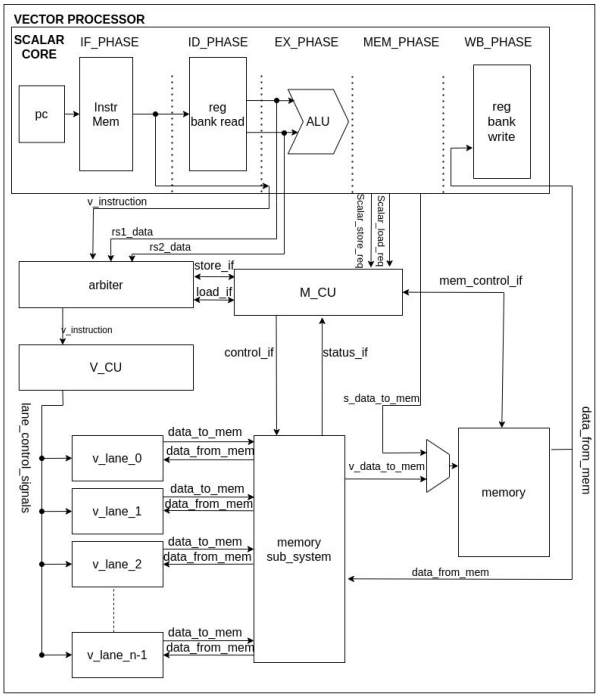
2.4 FPGA platforma i motivacija za njeno korišćenje

Vektorski procesor opisan u ovom radu je realizovan pomoću FPGA (*eng. Field Programmable Gate Arrays*) platforme. To su poluprovodnički uređaji zasnovani na matricama konfigurabilnih logičkih blokova povezanih pomoću programabilnih interkonekcija.

Sistemi koji se realizuju na FPGA platformama su najčešće akceleratori za aplikacije koje iskorišćavaju paralelizam između podataka. To su hardverski blokovi dizajnirani da obavljaju jednu vrstu zadatka sa ne toliko konfigurabilnih opcija. Oblasti u kojima je ovakav pristup zastupljen su: obrada slike i videa, mašinsko učenje (*eng. Machine learning*), emulacija hardvera, itd. Ovaj rad istražuje alternativnu mogućnost korišćenja FPGA platforme za kreiranje vektorskog procesora kao akceleratora opšte namene. Procesor bi posedovao standardan set instrukcija, tako da bi svako, bez iskustva sa dizajnom hardvera, mogao da ga programira. Takođe, zbog mogućnosti reprogramiranja dizajna na FPGA platformama, moćiće da se menjanju neke od karakteristika procesora kako bi se povećale performanse ili kako bi se optimizovala iskorišćenost resursa.

3. ARHITEKTURA VEKTORSKOG PROCESORA

U ovoj sekciji biće opisana RISC-V vektorska ekstenzija kao i mikroarhitektura procesora baziranog na njoj (slika 3). Osnovna ideja je da se pored skalarnog jezgra, koje podržava RISC-V *integer* set instrukcija, implementira i vektorsko jezgro koje bi se ponašalo kao ekstenzija na već postojeći set instrukcija.



Slika 3. Blok dijagram vektorskog procesora.

3.1 Opis skalarnog jezgra

Skalarno jezgro (*scalar core,* slika 3) je 32-bitni procesor bez dinamičkog izvršavanja instrukcija (*eng. In-order processor*), sa pet faza protočne obrade, koji implementira RISC-V *integer* set instrukcija. Osnovna uloga skalarnog jezgra jeste prihvat i prosleđivanje vektorskih instrukcija vektorskom jezgru, kao i izvršavanje ne vektorskog koda (skalarnih instrukcija).

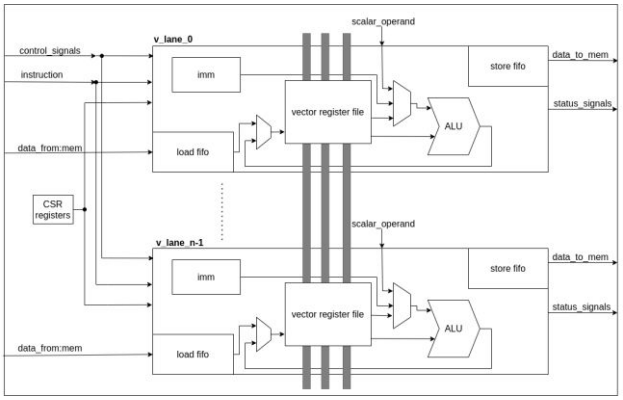
3.2 Programski model vektorske ekstenzije

Vektorska ekstenzija dodaje 32 vektorska registra (V0 – V31) na već postojeće registre skalarnog jezgra. Svaki vektorski registar biće predstavljen preko *VLEN* bita, pri čemu *VLEN* mora da bude stepen broja 2. To znači da ukoliko je potrebno da vektorski registar unutar sebe sadrži 32 elementa, pri čemu bi svaki bio 32-bitni, *VLEN* mora da bude 1024. Pored vektorskih registara, ekstenzija dodaje i 2 neprivilegovana CSR (*eng. Control Status registers*) registra (*vtype* i *vl*) koji programeru mogu da daju veću kontrolu. U 0.8 RISC-V „V“ nacrtu [4] predviđeno je da postoji 7 neprivilegovanih CSR registara, ali zbog skupa instrukcija koje će ovaj procesor da poseduje zaključeno je da su *vtype* i *vl* registri dovoljni.

3.3 Mikroarhitektura vektorskog jezgra

Vektorsko jezgro (slika 3) čine: A*rbiter*, V_CU, M_CU, vektorske linije i *memory_sub_system*

Vektorske linije (slika 3) su blokovi označeni sa *v_lane_0* do *v_lane_n*-1, što naznačava da je broj linija u vektorskom jezgru parametrizovan (u trenutnoj implementaciji broj linija mora bit stepen broja 2). Svaka vektorska linija sadrži kopiju funkcionalnih jedinica, deo vektorske registarske banke (pola ukoliko je broj linija 2, četvrtinu ukoliko je broj linija 4, itd), *load* i *store FIFO (eng. First In First Out)* memorije i mrežu za rutiranje. Takođe, svaka vektorska linija ima u potpunosti isti interfejs i kontrolisana je od strane istih kontrolnih signala. Na slici 4 je prikazan blok dijagram vektorskih linija.



Slika 4. Blok dijagram vektorskih linija

Vektorska registarska banka (*VRF - vector register file*) je distribuirana između vektorskih linija (označeno sivim linijama na slici 4) i na ovaj način izbegnuta je tradicionalna arhitektura jedne centralizovane banke [5] koja je zahtevala previše portova za upis i čitanje. Distribucija je realizovana tako da svaka banka unutar vektorskih linija ima 32 vektorska registra sa *VECTOR_LENGTH*/*NUM_OF_LANES* elemenata. *VECTOR_LENGTH* je parametar koji određuje koliko elemenata ima u jednom vektoru, dok je *NUM_OF_LANES* parametar koji određuje koliko ima vektorskih linija.

Load i Store FIFO blokovi (slika 4) su posrednici između memorije sa podacima (*memory, slika 3*) i VRF bloka. Prilikom izvršavanja instrukcije prihvata podataka podaci se prvo smeštaju u *load FIFO* memoriju i kada se svi prihvate, onda se, ukoliko vektorska linija nije zauzeta izvršavanjem druge instrukcije, smeštaju u *VRF*. Slično važi i za instrukcije upisa podataka u memoriju sa podacima, stim što se tada, podaci prvo smeštaju iz VRF modula u *store FIFO* memoriju i tek kada se svi smeste, onda se prosleđuju u memoriju sa podacima. Na ovaj način omogućeno je paralelno izvršavanje drugih instrukcija (ukoliko su nezavisne) dok vektorski procesor prenosi podatke iz memorije ili u memoriju.

Uloga Arbiter bloka je da prihvata instrukcije od skalarnog jezgra i da u zavisnosti od tipa prihvaćene instrukcije prosledi vektorsku instrukciju, vrednosti iz skalarnih registara i kontrolne signale *V_CU* modulu, M_CU modulu i vektorskim linijama. Arbiter je takođe zadužen za rešavanje zavisnosti prilikom paralelnog izvršavanja instrukcija prihvata/prenosa podataka i ostalih instrukcija.

Uloga *V_CU* (slika 3) bloka je da na osnovu vektorske instrukcije koju dobija od *Arbiter* komponente, generiše kontrolne signale koji upravljaju vektorskim linijama.

Memory_sub_system (slika 3) blok je zadužen da u cikličnom maniru prosleđuje podatke iz memorije sa podacima ka vektorskim linijama i obrnuto, od vektorskih linija ka memoriji sa podacima.

M_CU (slika 3) generiše kontrolne signale za memorijskim podsistemom i komunicira sa *Arbiter* komponentom i skalarnim jezgrom kako bi na osnovu informacija koje dobije od njih započeo transakcije prenosa podataka iz memorije ka procesoru, ili obrnuto, od procesora ka memoriji.

4. ANALIZA ISKORIŠĆENOSTI RESURSA I PERFORMANSI

Prethodno opisani sistem implementiran je na *Zybo* razvojnoj ploči koja pripada *Zynq-7000* familiji sistema na čipu (*eng. System on chip*) [6]. Za njeno programiranje korišćen je *Vivado* alat kompanije *Xilinx* [7], koji omogućava sintezu i implementaciju sistema opisanih u HDL (*eng. Hardware Description Language*) jezicima, kao i analizu utrošenih resursa i performansi. Na slikama 5 i 6 je prikazan izveštaj *Vivado* alata o utrošenosti resursa na *Zybo* razvojnoj ploči kada je broj vektorskih linija koje procesor poseduje 8 i 1, respektivno.

Resource	Utilization	Available	Utilization %
LUT	15156	17600	86.11
LUTRAM	883	6000	14.72
FF	7897	35200	22.43
BRAM	36.50	60	60.83
DSP	48	80	60.00
BUFG	1	32	3.13

Slika 5: Analiza iskorišćenosti resursa kada je broj vektorskih linija 8

Resource	Utilization	Available	Utilization %
LUT	8209	17600	46.64
LUTRAM	787	6000	13.12
FF	7422	35200	21.09
BRAM	24	60	40.00
DSP	6	80	7.50
BUFG	1	32	3.13

Slika 6: Analiza iskorišćenosti resursa kada je broj vektorskih linija 1

Ova dva slučaja su uzeta u obzir jer *Zybo* razvojna ploča nema dovoljno resursa da implementira procesor sa 16 vektorskih linija. Zauzetost BRAM ćelika je velika jer se one koriste za implementaciju VRF modula, *LOAD_FIFO* modula, *STORE_FIFO* modula, memorije sa podacima i memorije sa instrukcijama. Sa slike 5 i 6 se može videti drastična razlika u zauzetosti DSP ćelija, razlog za to je taj što svaki ALU modul unutar vektorskih linija koristi 6 DSP ćelija. Frekvencija rada procesora sa 8 vektorskih linija iznosi 95 Mhz, dok za procesor sa 1 vektorskom linijom ona iznosi 96 Mhz. Na maksimalnu frekvenciju utiče količina iskorišćenih resursa na FPGA platformi, jer što više resursa je neophodno, to je alatu teže da vrši povezivanje (*eng. Routing*).

5. ZAKLJUČAK

U ovom radu izvršena je vektorska ekstenzija 32-bitnog procesora, koji implementira RISC-V *integer* set instrukcija, prateći verziju 0.8 RISC-V - „V“ nacrta za vektorsku ekstenziju [4]. Za razliku od standardne implementacije procesora pomoću ASIC (*eng. Application Specific Integrated Circuit*) tehnologije, ovaj rad istražuje alternativnu mogućnost kreiranja vektorskog procesora koristeći FPGA platformu. Specifičnost ovih platformi je mogućnost reprogramiranja, čime bi mogle da se menjanju neke od karakteristika procesora kako bi se povećale performanse ili kako bi se optimizovala iskorišćenost resursa. Implementacija sistema izvršena je na *Zybo* razvojnoj ploči pomoću *Vivado* alata.

6. LITERATURA

[1] J. L. Hennessy and D. A. Patterson, „Computer Architecture - A Quantitative Approach, Sixth Edition“, Morgan Kaufmann, 2017

[2] Đ. Mišeljić i N. Kovačević, Napredni mikroprocesorski sistemi, Upoznavanje sa RISC-V procesorom, 2019.

[3] A. Waterman, K. Asanović, „The RISC-V Instruction Set Manual Volume I: User-Level ISA Document Version 2.2“, CS Division, EECS Department, University of California, Berkeley, 2017.

[4] „Working draft of the proposed RISC – V „V“ vector extension“, https://github.com/riscv/riscv-v-spec (pristupljeno u septembru 2020).

[5] C. Kozyrakis and David Patterson, „Overcoming the Limitations of Conventional Vector Processors“, Proceedings of the International Symposium on Computer Architecture, 2003.

[6] Zybo reference manual, https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual (pristupljeno u septembru 2020.)

[7] *https://www.xilinx.com/*, (pristupljeno u septembru 2020.)

7. KRATKA BIOGRAFIJA

Nikola Kovačević rođen je u Loznici 1995. god. Bečelor rad na Fakultetu Tehničkih Nauka, usmerenje Embeded sistemi i algoritmi, odbranio je 2018. godine, kada je izabran u zvanje saradnika u nastavi.
kontakt: nikolakovacevicftn@uns.ac.rs