Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.382.322
DOI: https://doi.org/10.24867/26BE11Milic

HARDVERSKA IMPLEMENTACIJA RISC-V PROCESORA SA PROTOČNOM OBRADOM KOJA PODRŽAVA RV32IM SKUP INSTRUKCIJA

A HARDWARE IMPLEMENTATION OF A RISC-V PROCESSOR WITH PIPELINE PROCESSING THAT SUPPORTS THE RV32IM INSTRUCTION SET

Nemanja Milić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad ukratko opisuje implementaciju RV32IM skup instrukcija koji podržava pet faza protočne obrade uz detekciju hazarda. Korištene su uobičajene faze protočne obrade: IF, ID, EXE, MEM i WB. Polazna tačka rada je bila inicijalna verzija RV32I procesora. Nastavilo se sa implementiranjem kompletnog I skupa instrukcija. Nakon toga je dodat i M skup instrukcija. Za implementaciju M skupa instrukcija, uzeta su u obzir dva načina pristupa realizacije. Prvi način koristi Booth-4 algoritam, a drugi DSP blokove sa Zybo ploče za instrukcije množenja. Oba pristupa daju iste logičke rezultate, a razlika je u performansama. Rad se završava sa upoređivanjem perfomansi procesora RV32I sa RV32IM (Booth-4 algoritam i DSP).*

Ključne reči: *RV32I, RV32IM, Booth-4 algoritam, DSP, Zybo*

Abstract – *This paper briefly describes an implementation of the RV32IM instruction set that supports five stages of pipeline processing with hazard detection. The usual pipeline stages were used: IF, ID, EXE, MEM and WB. The starting point of the work was the initial version of the RV32I processor. The implementation of the complete I set of instructions followed. After that, the M instruction set was added. For implementation of the M instruction set, two ways of approaching the realization are taken into account. Both approaches produce the same logical results, the difference being in performance. The first method uses the Booth-4 algorithm, and the second uses DSPs from the Zybo board for multiplication instructions. The paper ends with a comparison of the performance of the RV32I processor with the RV32IM (Booth-4 algorithm and DSP).*

Keywords: : *RV32I, RV32IM, Booth-4 algorithm, DSP, Zybo*

1. UVOD

Arhitektura RISC-V procesora nastala je 2010. godine na Univerzitetu Berkley u cilju učenja i razvoja znanja iz oblasti digitalnih blokova.

Pre pomenute poboljšane verzije, 1990. godine nastala je verzija za edukativne svrhe u vidu RISC skup instrukcija DLX. Prvobitni cilj je bio da procesor podržava osnovne operacije, da se pomoću odgovarajućeg programskog jezika nađe najjednostavniji način za implementaciju istog, da cena takvog rešenja bude što niža, a da performanse budu što bolje [1].

Do danas, različite arhitekture su našle široku primenu i mogu se kategorisati u dve glavne grupe na sledeći način:

RISC – Ovaj pristup koristi brojne jednostavne instrukcije za izvršavanje programa, pri čemu se ove instrukcije izvršavaju unutar jednog mašinskog ciklusa. Proces dekodiranja instrukcija je brz (zahvaljujući fiksnoj širini kodiranja uputstva), a implementacija obrade toka je jednostavna. Dizajn procesora zahteva manje tranzistora. Međutim, rezultujući kod ima tendenciju da bude veći i zauzima znatnu količinu RAM-a [2].

CISC – U navedenoj paradigmi, manja količina složenih instrukcija se koristi za izvršavanje programa, obuhvatajući više ciklusa takta. Dekodiranje i implementacija ovih instrukcija je vrlo složena (najvećim delom zbog promenljive širine kodiranja instrukcija) i zahteva veći broj tranzistora. Kao rezultat prethodno navedenog, veličina koda je smanjena [2].

CISC procesori su izvršavali manje instrukcija po programu u odnosu na RISC, ali je RISC procesorima u proseku trebalo mnogo manje taktova za izvršavanje instrukcija (CPI – Clocks Per Instruction). Tokom dugogodišnjeg rada sa pomenutim procesorima otkriveno je da CISC procesori izvršavaju otprilike upola manje instrukcija od RISC procesora, ali je njihov CPI bio u proseku šest puta veći [3].

I skup instrukcija predstavlja osnovni skup instrukcija koji svaki procesor podržava. Iako je osnovni skup, podržava širok opseg instrukcija. Instrukcije koje spadaju pod ovim skupom jesu one koje ispunjavaju aritmetičke logičke operacije, upisuju rezultate u registre, rade sa skokovima grana i memorijama.

M skup instrukcija podržava operacije množenja i deljenja. U nastavku će biti prikazane dve verzije implementacije. Prva podrška Booth-4 algoritam za množenje, dok druga verzija izvodi množenje pomoću DSP blokova sa Zybo ploče.

2. REALIZACIJA

Sve korištene komponente implementirane su u VHDL jeziku korišćenjem Vivado Design Suite alata. Prvo će biti prikazana realizacija *I* skupa instrukcija, a potom i *M*

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Ivan Mezei, red. prof.

skupa instrukcija koji je bio realizovan na dva načina. Dizajn je podeljen na datapath i controlpath celine.

2.1 *I* skup instrukcija

Pod *I* skupom pripadaju instrukcije koje ispunjavaju promenu vrednosti registra unutar registarske banke na osnovu poređenja između dve vrednosti. Takođe dolaze u obzir i instrukcije za čije izvršenje je neophodno korišćenje ALU jedinice. Što se tiče instrukcija za poređenje, one obično upisuju vrednost 1 u registar ako je uslov ispunjen, odnosno 0 u slučaju da nije. Koriste se za implementaciju if-else instrukcija i nekih drugih uslovnih operacija. U slučaju da nije ispunjen uslov, može se postići efikasnije izvršavanje koda. U nastavku će biti prikazana implementacija unapređene verzije procesora koji podržava *I* skup instrukcija. U tabeli 1 se može videti detaljan prikaz formata instrukcija *I* skupa.

Tabela 1. I skup instrukcija

imm[31:12]				*rd*	"0110111"	***LUI***
imm[31:12]				*rd*	"0010111"	***AUIPC***
imm[20|10:1|11|19:12]				*rd*	"1101111"	***JAL***
imm[11:0]		*rs1*	"000"	*rd*	"1100111"	***JALR***
imm[12|10:5]	*rs2*	*rs1*	"000"	*imm[4:1|11]*	"1100011"	***BEQ***
imm[12|10:5]	*rs2*	*rs1*	"001"	*imm[4:1|11]*	"1100011"	***BNE***
imm[12|10:5]	*rs2*	*rs1*	"100"	*imm[4:1|11]*	"1100011"	***BLT***
imm[12|10:5]	*rs2*	*rs1*	"101"	*imm[4:1|11]*	"1100011"	***BGE***
imm[12|10:5]	*rs2*	*rs1*	"110"	*imm[4:1|11]*	"1100011"	***BLTU***
imm[12|10:5]	*rs2*	*rs1*	"111"	*imm[4:1|11]*	"1100011"	***BGEU***
imm[11:0]		*rs1*	"000"	*rd*	"0000011"	***LB***
imm[11:0]		*rs1*	"001"	*rd*	"0000011"	***LH***
imm[11:0]		*rs1*	"010"	*rd*	"0000011"	***LW***
imm[11:0]		*rs1*	"100"	*rd*	"0000011"	***LBU***
imm[11:0]		*rs1*	"101"	*rd*	"0000011"	***LHU***
imm[11:5]	*rs2*	*rs1*	"000"	*imm[4:0]*	"0100011"	***SB***
imm[11:5]	*rs2*	*rs1*	"001"	*imm[4:0]*	"0100011"	***SH***
imm[11:5]	*rs2*	*rs1*	"010"	*imm[4:0]*	"0100011"	***SW***
imm[11:0]		*rs1*	"000"	*rd*	"0010011"	***ADDI***
imm[11:0]		*rs1*	"010"	*rd*	"0010011"	***SLTI***
imm[11:0]		*rs1*	"011"	*rd*	"0010011"	***SLTIU***
imm[11:0]		*rs1*	"100"	*rd*	"0010011"	***XORI***
imm[11:0]		*rs1*	"110"	*rd*	"0010011"	***ORI***
imm[11:0]		*rs1*	"111"	*rd*	"0010011"	***ANDI***
"0000000"	*shamt*	*rs1*	"001"	*rd*	"0010011"	***SLLI***
"0000000"	*shamt*	*rs1*	"101"	*rd*	"0010011"	***SRLI***
"0100000"	*shamt*	*rs1*	"101"	*rd*	"0010011"	***SRAI***
"0000000"	*rs2*	*rs1*	"000"	*rd*	"0110011"	***ADD***
"0100000"	*rs2*	*rs1*	"000"	*rd*	"0110011"	***SUB***
"0000000"	*rs2*	*rs1*	"001"	*rd*	"0110011"	***SLL***
"0000000"	*rs2*	*rs1*	"010"	*rd*	"0110011"	***SLT***
"0000000"	*rs2*	*rs1*	"011"	*rd*	"0110011"	***SLTU***
"0000000"	*rs2*	*rs1*	"100"	*rd*	"0110011"	***XOR***
"0000000"	*rs2*	*rs1*	"101"	*rd*	"0110011"	***SRL***
"0100000"	*rs2*	*rs1*	"101"	*rd*	"0110011"	***SRA***
"0000000"	*rs2*	*rs1*	"110"	*rd*	"0110011"	***OR***
"0000000"	*rs2*	*rs1*	"111"	*rd*	"0110011"	***ANDI***

Unapređena aritmetičko logička jedinica sada podržava operacije ADD, SUB, SLT, SLTU, XOR, ILI, I, SLL, SRL i SRA. Immediate blok je unapređen kako bi podržavao sve nove instrukcije koje sadrži u sebi imm polje. Instrukcije koje sadrži imm polje su I, S, B, U i J tipa.

Branch control blok je implementiran u datapath delu, tačnije u ID fazi. Svrha ovog bloka jeste u cilju implementacije instrukcija B tipa. Instrukcije koje spadaju pod B tipom jesu BEQ (*Branch if equal*), BNE (*Branch if not equal*), BLT (*Branch if less than*), BGE (*Branch if greater or equal*), BLTU (*Branch if less than unsigned*) i BGEU (*Branch if greater or equal unsigned*).

Load Store blok je takođe implementiran u datapath celini. Postoji dve instance ovog bloka. Jedna instanca je u MEM dok se druga instanca nalazi u WB fazi. Cilj ovog bloka jeste potreba instrukcija za skladištenje podataka u memoriji, kao i čitanje iz iste. Instrukcije za skladištenje podataka su SB (*Store byte*), SH (*Store half word*) i SW (*Store word*). Instrukcije za čitanje podataka iz memorije su LB (*Load byte*), LH (*Load half word*), LW (*Load word*), LBU (*Load byte unsigned*) i LHU (*Load half word unsigned*).

U slučaju bajt instrukcija, uzima se najnižih 8 bita, dok će na prvih 24 bita biti podatak predstavljen isti kao što je sedmi bit, nakon toga sledi konkatenacija ta dva dela podatka. Tako se dobija 32-bitni izlazni podatak. U slučaju half word instrukcije, slična je priča samo što se sad uzima 16 nižih bita i spaja sa podatkom od 16 bita, koji će biti predstavljen kao sve nule ili jedinice u zavisnosti od 15. bita. U slučaju word instrukcije, koristi se ceo 32-bitni podatak koji je došao na ulaz bloka. Za neoznačene (unsigned) instrukcije konkatenacija se vrši uvek sa nulama. Ovde je potrebno napomenuti ulogu bloka memorija podataka. Unutrašnja struktura je podeljena na 8-bitnu ćeliju kako bi se omogućilo pisanje ili čitanje jednog bajta, pola reči (dva bajta) ili cele reči (četiri bajta).

2.2 *M* skup instrukcija

Na postojeću, unapređenu implementaciju RV32I procesora je dodat *M* skup instrukcija. Instrukcije koje se podrazumevaju pod ovim skupom jesu instrukcije množenja i deljenja. Najveća promena u ovom dodatku jeste implementacija množača i deljenika koji se nalaze u sklopu ALU jedinice. U nastavku će biti prikazane dve verzije implementacije. Prva podržava Booth-4 algoritam za množenje, dok druga verzija izvodi množenje pomoću DSP blokova. U tabeli 2 je moguće videti instrukcije koje spadaju pod *M* skupom.

Tabela 2. M skup instrukcija

"0000001"	*rs2*	*rs1*	"000"	*rd*	"0110011"	***MUL***
"0000001"	*rs2*	*rs1*	"001"	*rd*	"0110011"	***MULH***
"0000001"	*rs2*	*rs1*	"010"	*rd*	"0110011"	***MULHSU***
"0000001"	*rs2*	*rs1*	"011"	*rd*	"0110011"	***MULHU***
"0000001"	*rs2*	*rs1*	"100"	*rd*	"0110011"	***DIV***
"0000001"	*rs2*	*rs1*	"101"	*rd*	"0110011"	***DIVU***
"0000001"	*rs2*	*rs1*	"110"	*rd*	"0110011"	***REM***
"0000001"	*rs2*	*rs1*	"111"	*rd*	"0110011"	***REMU***

2.2.1 Booth-4 algoritam verzija

Booth-4 algoritam je algoritam množenja koji se koristi za efikasno izvođenje binarnog množenja celih brojeva sa ili bez predznaka. Smanjuje broj operacija sabiranja i oduzimanja potrebnih za množenje korišćenjem oblika predstavljanja sa predznakom.

Standardni Booth-4 algoritam je algoritam za binarne brojeve sa znakom, koji se često naziva i Butov algoritam množenja. Postoji i varijanta poznata kao Butov algoritam za neoznačene binarne brojeve. Obe verzije dele osnovnu ideju optimizacije množenja [4].

Množač je implementiran u sklopu ALU jedinice. Ovaj blok je neophodan zbog realizacije instrukcija MUL (*Multiply*), MULH (*Multiply High*), MULHSU (*Multiply High Signed Unsigned*) i MULHU (*Multiply High Unsigned*). Za samu implementaciju se koristi Booth-4 algoritam. Prednost je smanjenje kašnjenja u poređenju sa

normalnim procesom množenja. Za klasičan proces množenja kao što je već objašnjeno, potrebno je onoliko množenja koliko množilac ima bita.

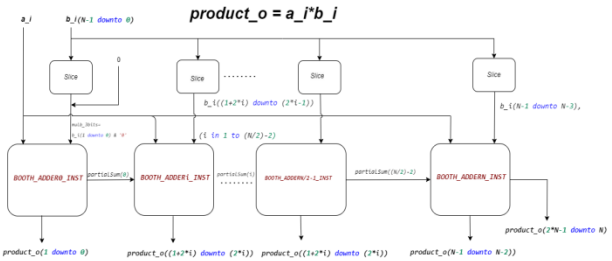
Za svaku dobijenu parcijalnu sumu potrebno je izvršiti sabiranje, što zahteva mnogo hardvera i može biti sporo.

U tabeli 3 prikazane su vrednosti Booth-4 algoritma prema kojima je implementiran poboljšani množač.

Tabela 3. Booth-4 algoritam encoding

Xi+1	Xi	Xi-1	*Radix-4 Booth encoding*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

Na slici 1 je prikazan blok dijagram množača koji koristi pomenuti algoritam.



Slika 1. Množač koji koristi Booth-4 algoritam

Množač se sastoji od 17 Booth sabirača. Treba obratiti pažnju na desnu stranu tabele Booth-4 algoritma. Brojevi 0, 1 i 2 se mogu predstaviti sa 2 bita (00, 01 i 10), zbog toga svaki Booth sabirač ima izlaz product_o od dva bita. Konkatenacijom svakog product_o signala od svih 17 sabirača dobija se podatak od donjih 34 bita, a parcijalna suma poslednjeg sabirača (product_o(2*N-1 downto 0)) predstavlja gornjih 34 bita proizvoda množača.

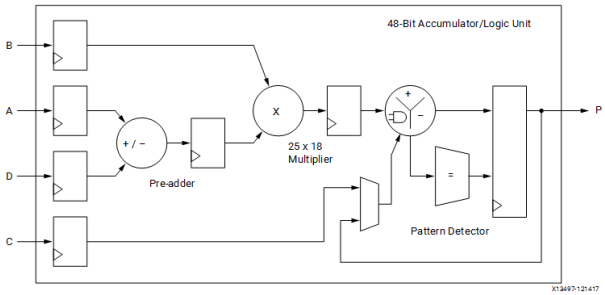
Svi pomenuti biti u celini predstavljaju izlaz množača od kog se uzimaju donji ili gornji biti u zavisnosti koja je instrukcija u pitanju. B operand odlučuje koji će se koeficijent za množenje biti izabran.

2.2.2 DSP verzija

Najsloženiji blok za izračunavanje na Xilinx FPGA ploči je DSP48 blok koji se može videti na slici 2. DSP48 blok je aritmetičko-logička jedinica (ALU) ugrađena u strukturu FPGA i sastoji se od lanca sačinjena od tri različita bloka.
Pomenuti lanac u DSP48 sadrži blok za sabiranje/oduzimanje povezan sa množačem koji je dalje povezan sa konačnim blokom za sabiranje/oduzimanje/akumulaciju. Pomenuti lanac omogućava jednoj jedinici DSP48 da implementira funkcije oblika [5]:

$$P = Bx(A+D)+C \quad (1)$$

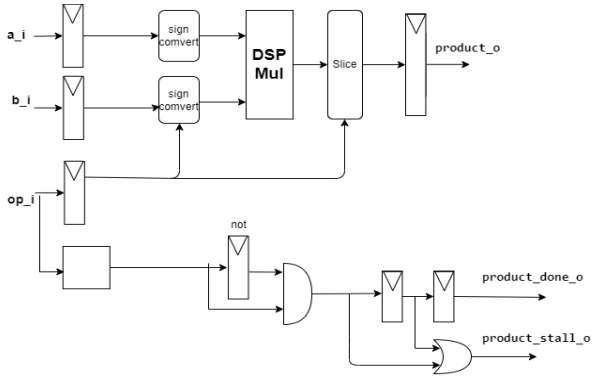
$$P \mathrel{+}= Bx(A+D) \quad (2)$$



Slika 2. Xilinx DSP48 [5]

Ideja ovog množača jeste što minimalnija upotreba hardverskih resursa u cilju dobijanja tačnih rezultata množenja. Sa korišćenjem DSP48 blokova na Zybo ploči nije potreban dodatni hardver, čime se direktno utiče na povećanje radne frekvencije procesora.

Ulazni operandi širine 32 bita se koriste za množenje. Rezultat je širine 64 bita. Potrebna su 2 DSP modula za svaku MUL instrukciju. Pošto će množač da podržava 4 MUL instrukcije, u rezultatima se može primetiti da se koristi 8 DSP modula. DSP48 podržava akumulaciju od 48 bita, dok je u ovom slučaju potrebna podrška za 64 bita. Iz tog razloga je potrebno 2 DSP modula po jednoj MUL instrukciji. Na slici 3 prikazana je blok šema pomenutog množača.



Slika 3. Blok šema množača koji koristi DSP

Ulazi u blok predstavljaju operandi a i b širine 32 bita, kao i op koji predstavlja jednu od četiri instrukcije. Izlazi množača jesu 32-bitni rezultat množenja kao i signali koji signaliziraju završetak operacije množenja, odnosno stall signal za množenje u slučaju da operacija nije završena.

Ovde se može primetiti činjenica da se operacija množenja ne izvršava samo u jednom taktu tokom EXE faze, nego u dva kloka. Naime, dodavanjem dva uzastopna registra utiče se na povećanje frekvencije. Propagaciono kašnjenje signala preko Bulove logike između dva uzastopna registracija u nizu ograničavanje brzine takta.

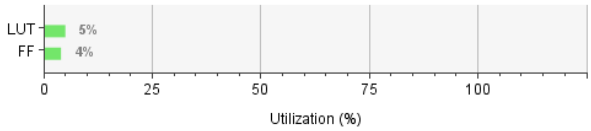
3. REZULTATI

U ovoj sekciji prikazani su rezultati tri verzije procesora:
- Inicijalna verzija procesora RV32I
- RV32IM koji koristi Booth-4 algoritam za množenje
- RV32IM koji koristi DSP sa Zybo ploče za množenje

3.1 Iskorišćenost resursa

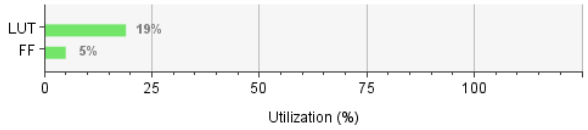
U ovoj sekciji će biti prikazan uvid koliko je bilo iskorišćeno resursa za svaku verziju procesora. Na slikama 4, 5 i 6 se redom može videti iskorišćenost resursa za procesore RV32I, RV32IM procesor koji koristi Booth-4 algoritam kao i za RV32IM procesor koji koristi DSP.

Resource	Utilization	Available	Utilization %
LUT	957	17600	5.44
FF	1348	35200	3.83



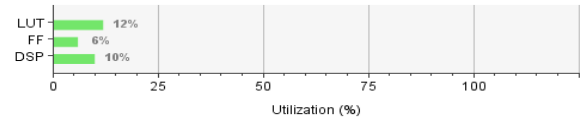
Slika 4. Iskorišćenost resursa RV32I

Resource	Utilization	Available	Utilization %
LUT	3378	17600	19.19
FF	1846	35200	5.24



Slika 5. Iskorišćenost resursa RV32IM koji koristi Booth-4 algoritam za množenje

Resource	Utilization	Available	Utilization %
LUT	2151	17600	12.22
FF	2016	35200	5.73
DSP	8	80	10.00



Slika 6. Iskorišćenost resursa RV32IM koji koristi DSP sa Zybo ploče za množenje

3.2 Maksimalna učestanost rada

Za sve tri verzije procesora je uzeta klok perioda od 10ns sa pretpostavkom da će raditi na 100MHz. Oduzimanje vremena Worst Negative Slack od pomenute klok periode dobija se konačna vrednost vremena koja se koristi za dobijanje maksimalne učestanosti.

$$f = 1/T \quad (3)$$

Inicijalna verzija procesora RV32I:
1/13.1 = 76.3MHz

RV32IM koji koristi Booth-4 algoritam za množenje:
1/47.6 = 21MHz

RV32IM koji koristi DSP sa Zybo ploče za množenje:
1/8.978 = 112.3MHz

Očigledno je da inicijalna verzija procesora, zahteva najmanje hardvera. Razlog je što podržava samo 4 instrukcije iz I skupa, pa samim tim i implementacija nije previše zahtevna. RV32IM procesor koji koristi Booth-4 algoritam, zahteva najviše hardvera od sve tri verzije. Razlog tome leži u implementiranju algoritma koji je suviše zahtevan. Poslednja verzija procesora koristi dodatni DSP hardver za množenje, ali i dalje ne zahteva toliki kapacitet kao druga verzija procesora.

Što se tiče radne učestalosti, inicijalna verzija iako nije previše zahtevana, nije dostigla postavljenih 100MHz. Najlošije rezultate je pokazala druga verzija sa Booth-4 algoritmom. Velika iskorišćenost resursa, kao i dugačke kritične putanje su dovele do loših rezultata.

Može se zaključiti da najbolje rezultate daje poslednja verzija procesora koja koristi DSP množenje. Razlog počiva na činjenicama da DSP resursi već postoje na korišćenoj Zybo ploči, stoga nema potrebe za dodatnim hardverom, što dovodi do velikih ušteda i povećanja radne učestalosti. Ova verzija procesora je ispunila očekivanje i početnu pretpostavku.

4. ZAKLJUČAK

Glavni zadatak rada koji je autor trebalo da ispuni, uspešno je odrađen. Implementirani su *I* i *M* skup instrukcija za RISC-V arhitekturu. Poslednja verzija dizajna u potpunosti podržava arhitekturu RV32IM.

Implementacija *M* skup instrukcija je izvedena na dva načina. Prva verzija podržava Booth-4 algoritam za operaciju množenja, dok druga verzija za operaciju množenja koristi DSP blokove koji se nalaze na razvojnoj Zybo ploči. Rezultat rada se ogleda u poređenju iskorišćenosti resursa kao i maksimalne radne učestanosti između pomenute dve implementirane verzije sa inicijalnom verzijom procesora. DSP verzija RV32IM procesora daje rezultate u najbržem mogućem roku. Nakon nje sledi RV32I verzija procesora, dok Booth-4 verzija RV32IM procesora omogućava rezultate u najsporijem roku u poređenju sa prethodne dve verzije. Što se tiče potrebnog hardvera za realizaciju procesora, po tom pitanju je najmanje zahtevna RV32I verzija, a prate je DSP i na kraju Booth-4 verzija RV32IM procesora.

5. LITERATURA

[1] https://en.wikipedia.org/wiki/RISC-V, pristupljeno septembar 2023.

[2]https://www.elektronika.ftn.uns.ac.rs/napredni-mikroprocesorski-sistemi/wp-content/uploads/sites/103/2018/03/Vezba1_Uvod_u_RISC-V_arhitekturu.pdf , pristupljeno septembar 2023.

[3] D. Patterson, "Reduced Instruction Set Computers Then and Now Computer, vol. 50, no. 12, pp. 10-12, 2017.

[4]https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm, pristupljeno avgust 2023.

[5]https://www.xilinx.com/htmldocs/xilinx2017_4/sdaccel_doc/uwa1504034294196.html, pristupljeno jul 2023.

Kratka biografija:

Nemanja Milić rođen je u Novom Sadu 1994. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Energetska elektronika i telekomunikacije odbranio je 2023.god.
Kontakt: nemanja.milic.ns@gmail.com