FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO
UNIVERZA V LJUBLJANI

Marjana Erdelji

Primerjava implementacij množilnikov v logaritemskem številskem sistemu

MAGISTRSKO DELO

Mentor: doc. dr. Patricio Bulić

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.:132 -MAG-RI/2010
Datum: 17. 6. 2010

Marjana ERDELJI, univ. dipl. inž. rač. in inf.

Ljubljana

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Primerjava implementacij množilnikov v logaritemskem številskem sistemu**

Comparison of multiplier implementations in a logarithmic number system

Tematika naloge:
Glede na kriterije porabe prostora integriranega vezja, hitrosti izvajanja in porabe moči primerjajte tri algoritme za neeksaktno množenje: Mitchellov algoritem (MA), operandno dekompozicijski Mitchellov algoritem (OD-MA) in enostavni iterativni množilnik (SIM). Za primerjavo naj služi tudi aritmetično natančen matrični množilnik (ang. array multiplier). Podajte analizo napake MA in OD-MA, ki množita operande z logaritemsko aproksimacijo ter analizo napake pri SIM. SIM je podobno osnovan kot MA, a ne uporablja logaritemske aproksimacije. Pokažite, kako lahko iterativno poljubno zmanjšate njegovo napako. Podajte tudi analizo porabe moči za vse navedene izvedbe množilnikov. Primerjave opravite s strojno implementacijo na FPGA vezjih. Pričakovati je, da pokažete, da neeksaktno množenje prinaša enostavnejšo implementacijo, in s tem manj logičnih vrat, to je manjšo porabo prostora, večjo hitrost in manjšo porabo moči. Ugotovite uporabnost neeksaktnega množenja v aplikacijah digitalnega procesiranja signalov.

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Mentor:
doc. dr. Patricio Bulić

Dekan:
prof. dr. Franc Solina

[ORIGINAL IZDANE TEME]

[IZJAVA O AVTORSTVU]

Zahvala

Zahvala gre v prvi vrsti mentorju doc. dr. Patriciu Buliću za vso pomoč in nasvete ter za pregled in komentarje vsebine naloge.

Za poganjanje matlabovega programa na zmogljivejšem računalniku se zahvaljujem Mitji Placerju. Za kopiranje, tiskanje in vezavo dela, spodbudo in koristne nasvete pri reševanju kaotičnih tragedij se zahvaljujem dr. Tadeju Podgorniku in dr. Gregorju Geršaku.

Zahvaljujem se podjetju Harpha Sea d.o.o., ki mi je omogočila študijski dopust za dokončanje tega dela. Prav tako se iz srca zahvaljujem vsem sodelavcem za podporo in dobro voljo.

Zahvaljujem se Gregorju Kodru za prijaznost, potrpežjivost in ogromno izkazane ljubezni. Prav tako se zahvaljujem staršema za vso pomoč pri vsakdanjem življenju.

In nenazadnje se zahvaljujem vsem prijateljem, znancem in drugim, ki jih v zahvali nisem izrecno omenila, npr. prijateljem motoristom, še posebej Božidar Fabjanu, Branku Merharju, Marku Sorčiču in Borutu Mavcu, kolegu po »fohu« Antonu Galunu, prijateljema Lei Jakopin in Urošu Steletu, itd., čeprav so tudi oni posredno ali neposredno pripomogli k nastanku tega dela.

“... I was sitting in the rooms of the Analytical Society, at Cambridge, my head leaning forward on the table in a kind of dreamy mood, with a table of logarithms lying open before me. Another member, coming into the room, and seeing me half asleep, called out, "Well, Babbage, what are you dreaming about?" to which I replied "I am thinking that all these tables" (pointing to the logarithms) "might be calculated by machinery."

Charles Babbage, 1812/1813

Kazalo

Slike

Grafi

Tabele

Kratice

Kratica	Pomen	Razlaga
BMP	ang. Bitmap	Format slike v katerem je shranjena slika
DSP	ang. Digital Signal Processing	Digitalno procesiranje signalov (DPS), digitalna obdelava signalov
FPGA	ang. Field Programmable Gate Array	Programirljivo integrirano vezje tehnologije FPGA
JPEG	ang. Joint Photographic Experts Group	Metoda za kompresijo slik ali format slike, zapisan s to metodo (JPG)
LNS	ang.Logarithmic Number Systems	Logaritemski številčni sistemi
LUT	ang. Look-Up Table	Vpogledna tabela
MA	ang. Mitchell Algorithm	Mitchellov algoritem za množenje
OD	ang. Operand Decomposition	Dekompozicija operandov
OD-MA	ang. Operand Decomposition - Mitchell Algorithm	Mitchellov algoritem z dekompozicijo operandov
SIM	ang. Simple Iterative Multiplier	Enostavni iterativni množilnik
VLSI	ang. Very Large Scale Integration	Integrirano vezje z visoko stopnjo integracije

Povzetek

Primerjava implementacij množilnikov v logaritemskem številskem sistemu

Magistrsko delo se osredotoča na množilnike v logaritemskem številskem sistemu. Natančneje so opisani množilniki, primerni za aplikacije digitalnega procesiranja signalov. Algoritmi digitalnega procesiranja signalov vsebujejo veliko število množenj, kar je potratno v smislu časovnega izvajanja, porabe površine vezja in porabe moči. Obstajajo metode za poenostavitev množenja, npr. odrezani in logaritemski množilniki, ki zaradi manjše kompleksnosti porabijo manj površine, se izvajajo hitreje in porabijo manj moči, vendar pa prinašajo napako rezultata. Kljub temu se te metode uporabljajo v primerih, kjer je hitrejše izvajanje bolj pomembno od točnosti. To delo se omejuje predvsem na množilnike v logaritemskem številskem sistemu. Glede na kriterije porabe površine integriranega vezja, hitrosti izvajanja in porabe moči je bila narejena primerjava naslednjih aritmetično neeksaktnih množilnikov: množilnik z Mitchellovim algoritmom (MA), množilnik z Mitchellovim algoritmom z dekompozicijo operandov (OD-MA) in enostavni iterativni množilnik (ang. Simple Iterative Multiplier, SIM). Podana je analiza napake algoritmov MA in OD-MA, ki množita operande z logaritemsko aproksimacijo ter analiza napake pri SIM. SIM je podobno zasnovan kot MA, vendar poenostavi aproksimacijo rezultata, s čemer poenostavi naknadno implementacijo korekcijskih vezij, njegovo napako pa lahko iterativno poljubno zmanjšamo. Za primerjavo je bila potrebna strojna implementacija teh množilnikov na Spartan3 in Virtex6 Low Power FPGA vezjih. Strojne implementacije vsebujejo seštevalnike in pomikalnike, zato niso potratni v porabi logičnih vrat in porabi moči. Primerjavo teh neeksaktnih množilnikov smo v vseh kriterijih opravili tudi z aritmetično eksaktnim matričnim množilnikom (ang. array multiplier). Opravili smo tudi strojno implementacijo cevovodnih različic teh množilnikov. Rezultati prikazujejo, da poenostavitev implementacije prinaša manjšo porabo površine vezja, s tem pohitri izvajanje in zmanjša porabo moči, ampak na račun povečanja napake. S primerom glajenja slik, ki za konvolucijo slike s konvolucijsko masko uporablja neeksaktne množilne algoritme je bilo ugotovljeno, da je njihova napaka po vrednosti primerljiva z napako, ki jo ustvari konverzija slike iz formata bmp v jpg. Neeksaktni množilniki so torej uporabni v aplikacijah, ki dopuščajo določen odstotek tolerance na napako.

Ključne besede: računalniška aritmetika, binarno množenje, digitalno procesiranje signalov, logaritemski številski sistem, FPGA

Abstract

Comparison of multiplier implementations in a logarithmic number system

The master's thesis discusses binary multipliers. Reported in detail are multipliers suitable for digital signal processing applications. Digital signal processing algorithms often rely heavily on a large number of multiplications, which is both time and power consuming. There are many practical solutions to simplify multiplication, like truncated and logarithmic multipliers, which consume less time and power, but introduce errors. Nevertheless, they can be used in situations where a shorter time delay is more important than accuracy. The thesis presents multipliers in a logarithmic number system. Following arithmetically inexact multipliers were compared against speed, resources required for implementation, power consumption and error rate: Mitchell's algorithm based multiplier (MA), operand decomposition based multiplier (OD-MA) and a simple iterative multiplier (SIM). Error analysis of these multipliers was made. Through a SIM iterative procedure it is possible to achieve an arbitrary accuracy. For the hardware implementation assessment, the multipliers were implemented on the Spartan3 and Virtex6 Low Power FPGA chip. The hardware solution involves adders and shifters, so it is not gate and power consuming. These arithmetically inexact multipliers were also compared with arithmetically exact array multiplier based on all before mentioned criteria. Pipelined hardware implementation of all these multipliers was made. The results show how simplifying the implementation of multipliers lowers required resurces, speeds up the implementation and reduces power consumption, but at the expense of increased error rate. Smoothing images application, based on convolution of image and convolution mask, was used to determine how comparable is error generated by arithmetically unexact multipliers with error generated by the image conversion from bmp to jpg format. Unexact multipliers are therefore useful in applications that allow a ceratin percentage of error tolerance.

Key words: computer arithmetic, binary multiplication, digital signal processing, logarithmic number system, FPGA

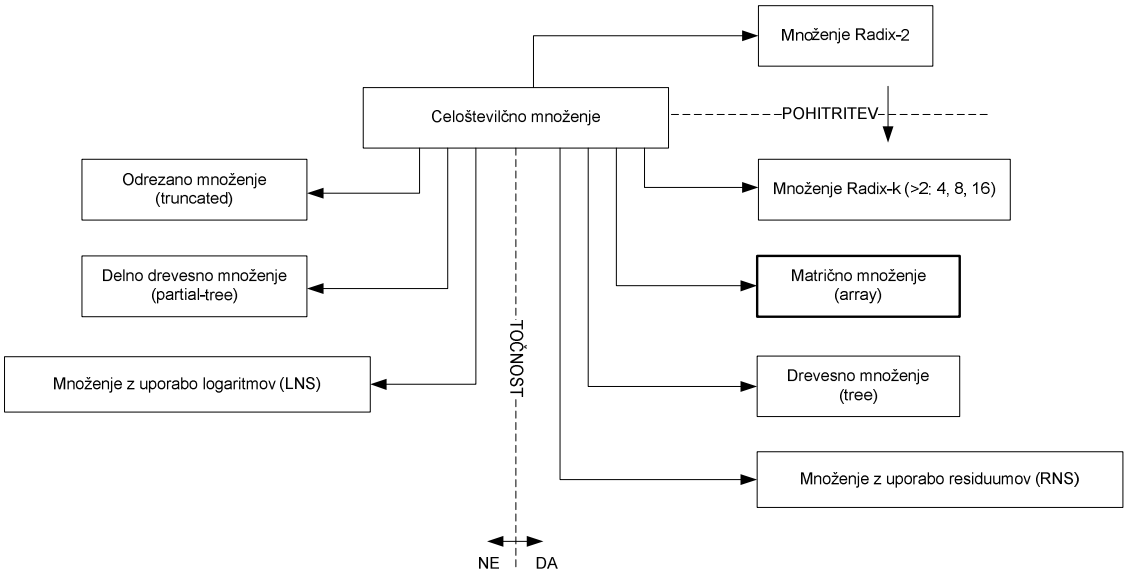
Poglavje 1

Uvod

Množenje je bila vedno časovno, površinsko in energijsko potratna operacija, še posebej za operande večjih velikosti. To ozko grlo je najbolj prisotno pri aplikacijah digitalnega procesiranja signalov oziroma digitalne obdelave signalov (DPS). Najbolj znani primeri aplikacij so filtriranje s končnim enotinim odzivom (FIR), hitra Fourierjeva transformacija (FFT) in diskretna kosinusna transformacija (DCT), ki množično uporabljajo aritmetični operaciji seštevanje in množenje [17].

V mnogih aplikacijah DPS absolutno točen rezultat aritmetičnih operacij ni potreben, zato je sprejemljiv tudi približen rezultat množenja. Poleg tega imajo mnogi DPS sistemi opravka s signali, ki vsebujejo šum. S tem ko odstopamo od zahteve po absolutni točnosti rezultata, lahko zmanjšamo kompleksnost vezja in posledično tudi velikost vezja, zakasnitev potovanja signala čez vezje ter število preklapljanj logičnih gradnikov, torej zmanjšamo porabo moči vezja.

Obstaja mnogo praktičnih rešitev za poenostavitev množenja [4, 11, 17, 18, 19, 21, 23, 24]. Na grobo jih lahko klasificiramo v dve veji, ena je odrezano množenje [24] (ang. truncated multiplication) in druga logaritemsko množenje [19]. Spodnja slika prikazuje njuno umestitev v splošen diagram klasifikacije celoštevilčnega množenja.

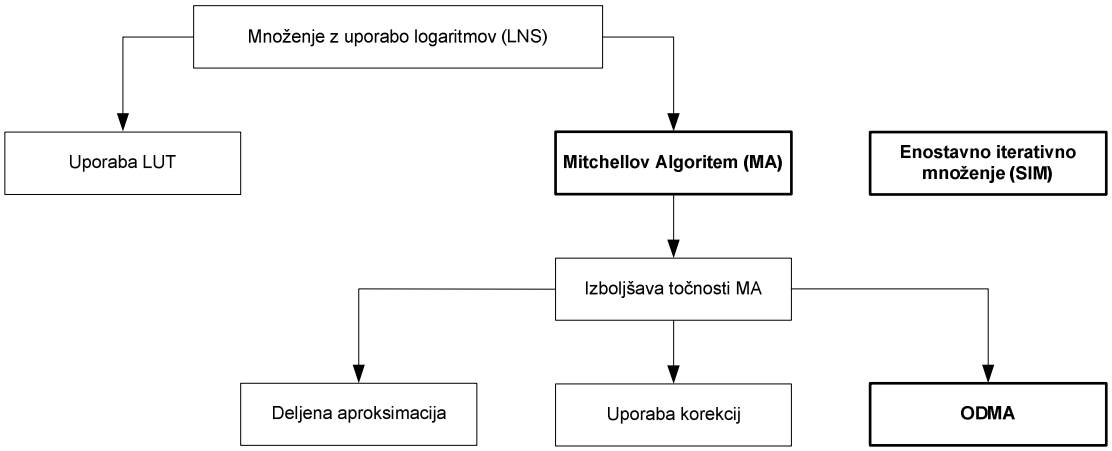


Slika 1: Klasifikacija metod celoštevilčnega množenja

To delo se omejuje na logaritemske številske sisteme (ang. Logarithmic Number Systems, LNS). LNS predstavljajo primerno metodo za implementacijo množenja v aplikacijah DPS. Aritmetika LNS je uporabna predvsem zaradi enostavnosti. Operacijo množenja v LNS

namreč nadomešča seštevanje logaritemskih vrednosti operandov. Ta enostavnost prinaša precejšno slabost: za pretvorbo v LNS in obratno moramo izračunati logaritemske in antilogaritemske vrednosti. Ker teh vrednosti ne moremo izračunati točno, uporabimo aproksimacijo funkcij logaritma in antilogaritma. Računanje v LNS tako prinaša izgubo točnosti na račun višje hitrosti izvajanja operacij, manjše površine vezja in porabe moči.

Logaritmi se pogosto računajo s pomočjo Taylorjeve vrste ali s pomočjo tabele shranjenih vrednosti logaritmov vseh možnih operandov. Ti metodi sta neučinkoviti v smislu hitrosti izvajanja, porabe površine vezja in porabe moči. V strokovni literaturi je bilo predstavljenih veliko metod za izboljšavo logaritemske aritmetike [1, 2, 5, 8, 14, 17, 18, 19, 21]. Na grobo pa jih lahko razvrstimo v dve skupini: interpolacija z uporabo vpoglednih tabel (ang. Look-Up Table, LUT) in logaritemsko računanje z Mitchellovim algoritmom (MA). Spodnja slika prikazuje klasifikacijo množenja z LNS metodo. Začetne študije so pokazale, da je implementacija logaritmov z interpolacijskimi metodami precej natančna, ampak zahteva veliko računanja in dodatnega pomnilnika [17]. Po drugi strani pa MA ne zahteva nobenega pomnilnika, je enostaven in učinkovit algoritem, na račun izgube nekaj točnosti. Kljub temu obstaja veliko interesa za uporabo MA v aplikacijah DPS.



Slika 2: Klasifikacija množenja z metodo LNS

MA za množenje dveh števil je enostaven. Logaritemske vrednosti vhodnih števil se sešteje in z antilogaritmom vsote dobi rezultat množenja. Na točnost rezultata vpliva metoda, s katero dobimo vrednost logaritma in antilogaritma. Mitchell je za izračun vrednosti uporabil odsekovno premočrtno aproksimacijo logaritemskih in antilogaritemskih krivulj [19].

Aproksimacija prinaša nekaj izgube točnosti. Predlaganih je bilo precej metod za izboljšavo. V tem delu sta predstavljeni dve novejši metodi za izboljšavo MA: Mitchellov algoritem z dekompozicijo operandov (ang. Operand Decomposition MA, OD-MA) in enostavni iterativni množilnik (ang. Simple Iterative Multiplier, SIM). Metoda OD-MA prestavlja predprocesni korak k MA, med tem ko je SIM zasnovan na isti ideji predstavitve števil kot MA, a se izogne kompleksnemu popravljanju rezultata in računanju ostankov, kar posledično omogoči enostavnejšo izvedbo korekcijskih vezij, s tem pa višjo hitrost delovanja ter manjšo porabo površine vezja in moči.

V poglavju 2 je narejen uvod v aritmetično operacijo množenje, v katerem je predstavljeno množenje predznačenih in nepredznačenih celih števil. Omenjeni so načini za pohitritev celoštevilčnega množenja ter hitrost, točnost in aplikacije DPS.

V naslednjih poglavjih 3, 4, 5 in 6 so predstavljeni algoritmi za množenje, ki so bili detajlno preučeni in v poglavju 7 je bila opravljena njihova primerjava glede na kriterije: napaka, hitrost izvajanja, izkoriščenost površine naprave in poraba moči.

V poglavju 3 je opisan matrični množilnik, ki je aritmetično eksakten in se izkaže za primernega za implementacijo na FPGA, še posebej njegova cevovodna različica. Za primerjavo z aritmetično neeksaktnimi množilniki je podana njegova strojna implementacija.

V poglavju 4 je opisan MA, ki predstavlja aritmetično neeksaktno množenje. Podana je analiza napake MA. Algoritem je zanimiv v smislu izboljšave na račun aritmetične neeksaktnosti (glede na kriterije primerjave). Podana je strojna implementacija množilnika zasnovanega na MA.

V poglavju 5 je opisan Mitchellov algoritem z dekompozicijo operandov (OD-MA). Gre za predprocesiranje Mitchellovega algoritma. Na vhodu Mitchellovega algoritma imamo dekomponirane operande, s čimer se zmanjša napaka Mitchellovega algoritma. Podana je strojna implementacija množilnika zasnovanega na Mitchellovem algoritmu z dekompozicijo operandov.

V poglavju 6 sledi enostavni iterativni množilnik (SIM), ki je zasnovan podobno kot Mitchellov algoritem. Dodano je korekcijsko vezje, ki iterativno izboljšuje točnost rezultata s poljubno toleranco. Podana je strojna implementacija enostavnega iterativnega množilnika.

V zadnjem poglavju sledi primerjava implementacij 16, 32 in 64-bitnih množilnikov na Spartan-3 in Virtex-6 FPGA integriranem vezju: matrični množilnik, množilnik zasnovan na MA, množilnik zasnovan na OD-MA ter množilnik SIM. Pri SIM množilniku je poleg osnovne strukture implementirano tudi vezje za korekcijo napake. Pri vseh so bile za oceno možne pohitritve implementirane tudi cevovodne različice. Kriteriji za primerjavo so: napaka, hitrost izvajanja, poraba površine vezja in poraba moči. Sledi primerjava množilnikov z eksperimentalnimi rezultati, kjer se s konkretno aplikacijo glajenja slik ugotavlja kako in v kakšni meri neeksaktno množenje vpliva na rezultat.

Poglavje 2

Uvod v množenje

Do poznih 70-ih letih prejšnjega stoletja večina računalnikov ni imela ukaza za množenje, zato so programerji uporabljali podprogram za množenje, ki je v zanki delal pomike operandov in v akumulator shranjeval delne rezultate produkta. Motorola 6809 je bil eden od prvih mikroprocesorjev, ki je imel namenski strojni ukaz za množenje. Šlo je za podobno rutino, ki je bila implementirana v mikrokodi ukaza MUL. Z naraščanjem števila tranzistorjev na integriranem vezju (Moore-ov zakon), je z implementacijo več seštevalnikov na enem integriranem vezju postalo možno seštevati delne produkte naenkrat, in ne ponovno uporabiti en sam seštevalnik. Zaradi tega, ker so algoritmi digitalnega procesiranja signalov večino časa izvajali množenje, so načrtovalci porabili veliko procesorske površine, da bi naredili množenje kar se da hitro – enota za MAC (ang. Multiply and ACcumulate), ki je izvrševala operacije v eni urini periodi in je v zgodnjih DSP-jih zavzemala večino površine na integriranem vezju.

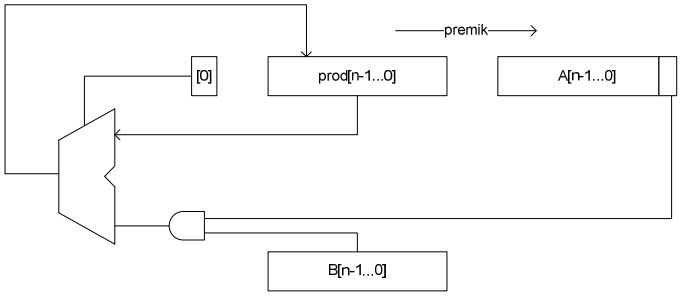
Zmogljivost splošno namenskih računalnikov je opazno padla pri večini zahtevnejših računskih aplikacijah. Ena od alternativ je (bila) uporaba strojne opreme po meri (aplikacijsko specifična vezja, ASIC). Možna je (bila) tudi realizacija procesorjev s fiksno funkcijo (npr. FFT integrirano vezje). V tem poglavju so za lažje razumevanje kasnejših poglavij opisane osnove aritmetične operacije množenje.

Množenje je aritmetična operacija, ki jo izvajamo nad operandi, ki so lahko predstavljeni na več načinov. Imamo predznačena in nepredznačena števila. Nepredznačena števila se uporabljajo predvsem za računanje z naslovi. Pri predznačenih lahko števila predstavimo kot: predznak in velikost (ang. sign magnitude), dvojiški komplement, eniški komplement in predstavitev z odmikom (ang. biased) z enojno (32-bit) ali dvojno natančnostjo (64-bit). Najbolj uveljavljena oblika zapisa je v dvojiškem komplementu. Glede na to, ali je število celo ali realno, ga lahko zapišemo v fiksni ali plavajoči vejici. Aritmetika v plavajoči vejici se je pri splošno namenskih računalnikih vedno obravnavala ločeno od tiste v fiksni vejici. Razlog za to je v relativni redkosti operacij s števili v plavajoči vejici in v večji kompleksnosti njihove realizacije. Od prvih računalnikov naprej je bila ta aritmetika zato največkrat realizirana programsko. Danes so zaradi nižje cene logike te enote na večini računalnikov standardne (enota za operacije v plavajoči vejici, ang. Floating Point Unit, FPU). Aritmetika s števili v plavajoči vejici je podobna tisti s števili v fiksni vejici, z dvema razlikama: Pri operacijah je treba poleg mantise uporabiti tudi eksponent, osnova za izvedbo operacij pa je aritmetika s fiksno vejico. Druga razlika je v zaokroževanju rezultata operacije. Pri aritmetiki v fiksni vejici računamo s celimi števili in zaokroževanja ni, zato je aritmetika celih števil običajno točna. Števila v plavajoči vejici imajo zaradi zaokroževanja določeno količino matematične točnosti. Obseg števil, ki jih lahko predstavimo s števili v fiksni vejici, je za veliko problemov premajhen. To velja še posebej za probleme iz znanosti in tehnike, kjer pogosto nastopajo zelo velika ali zelo majhna števila.

Implementacija množilnika je odvisna od tega, kako so predstavljeni operandi.

2.1. Množenje predznačenih in nepredznačenih celih števil

Najbolj enostaven množilnik izračuna produkt dveh n-bitnih nepredznačenih celih števil bit za bitom (ang. Radix-2 Multiplication). Množilnik je sestavljen iz treh n-bitnih registrov **A**, **B** in **prod**, pomikalnika, primerjalnika z '0', multiplekserja in »in« vrat, kot je prikazano na sliki spodaj. Števili, ki jih množimo sta $a_{n-1}a_{n-2} \ldots a_0$ in $b_{n-1}b_{n-2} \ldots b_0$ v registrih **A** in **B**.



Slika 3: Arhitektura množenja Radix-2

En korak množenja je sestavljen iz dveh delov:

Tabela 1: Korak množenja enostavnega množilnika celih števil Radix-2

(i)	Če ima najmanj pomemben bit v **A** vrednost '1', potem se register **B** sešteje z vrednostjo v **prod**, v nasprotnem primeru pa se vrednosti **prod** doda vrednost '0'.
(ii)	Registra **prod** in **A** se pomakneta za eno mesto v desno.

Po n-korakih se vrednost produkta nahaja v registrih **prod** in **A**. Ta algoritem je binarna oblika ročnega računanja, ki smo ga vajeni iz osnovne šole. Pravimo mu tudi dolgo množenje (ang. long multiplication). Čas računanja je daljši kot pri drugih metodah, vendar je njegova realizacija na račun enostavnosti cenejša in površinsko manjša.

Drugače je pri računanju produkta dveh predznačenih celih števil. Lahko bi števili pretvorili v pozitivno obliko, ju nepredznačeno množili, rezultat pa negirali, če sta bili števili različno predznačeni. Pretvorbam se lahko izognemo z Boothovim algoritmom [12, 22] (ang. Booth recoding), ki omogoča množenje brez pretvarjanja ne glede na predznaka obeh števil. Veliko računalnikov ima vgrajeno dodatno logiko, ki pospeši izvajanje korakov Boothovega algoritma. Zanimivo pa je, da se na računalnikih, ki te logike nimajo, običajno uporablja pretvarjanje v obliko predznak in velikost. Programska realizacija Boothovega algoritma (isto velja za mikroprogramsko) je namreč zaradi posebnih operacij, ki jih zahteva Boothov algoritem, pogosto počasnejša od tiste s pretvarjanjem [12].

2.2. Pohitritev celoštevilčnega množenja

Način za pohitritev je uporaba Radix-k množenja, kjer v vsakem ciklu namesto enega bita (Radix-2) množimo več bitov naenkrat (npr. 2 bita pri Radix-4, 3 bite pri Radix-8, ...). Najpogosteje se uporablja množenje Radix-4. Več to tem je zapisano v [10, 22].

Še en način za pohitritev množenja celih števil je uporaba večih seštevalnikov. Tak pristop zahteva več površine na integriranem vezju. Dobro poznana implementacija takega

množilnika je matrični množilnik (ang. array multiplier). Njegova struktura je na videz podobna matriki. Za njegovo implementacijo je potrebnih $(n-1)^2$ 1-bitnih polnih seštevalnikov in n^2 »in« logičnih vrat za konjunkcije $x_i y_i$ [10, 22]. Zakasnitev matričnega množilnika narašča linearno z n. Ker je vsaka od n vrstic seštevalnikov v matriki zasedena samo $\frac{1}{n}$-ti del časa, je mogoče pričeti z naslednjim množenjem že veliko pred zaključkom prejšnjega. S cevovodnim izvajanjem matričnega množilnika lahko po začetni latenci dosežemo maksimalno n-kratno pohitritev.

Poleg matričnega množilnika obstajajo tudi drevesni množilniki. Seštevalniki so povezani v strukturo drevesa, ki ima minimalno zakasnitev čez vezje. Najbolj znana so drevesa Wallace [22]. Število seštevalnikov je enako kot pri matričnem množilniku. Razlika pa je v tem, da so zaradi strukture drevesa delni produkti bližje korenu drevesa, s čemer se zmanjša zakasnitev. Njihova slabost je neregularna povezanost seštevalnikov. V [22] so omenjeni tudi drugi pohitritveni množilniki, ki v tem delu niso tako zanimivi.

S cevovodnim izvajanjem na splošno dosežemo večjo prepustnost, s čimer po začetni latenci dosežemo višjo hitrost izvajanja, kot če bi povečevali kompleksnost necevovodne različice.

Pohitritev dosežemo tudi s poenostavljenimi metodami, ki ne zagotavljajo točnosti. Naj omenim dve: odrezan množilnik in logaritmično množenje.

Odrezan množilnik [23, 24] (ang. truncated multiplier) se obsežno uporablja pri digitalnem procesiranju signalov ali digitalni obdelavi signalov (DPS). Odrezani množilniki množijo dve n-bitni števili v n-bitni produkt. Osnovna ideja te metode je v zavrnitvi n manj pomembnih bitov produkta. Napaka, ki pri tem nastane, pa se kompenzira z dodanim korekcijskim vezjem za zmanjšanje aproksimacijske napake.

Največja prednost logaritmičnega množenja je substitucija množenja s seštevanjem. Pred tem je potrebno pretvoriti operande iz celoštevilčnega številskega sistema v logaritemski številski sistem. Logaritmično množenje dveh operandov N_1 in N_2 se torej izvaja v treh fazah: 1. izračun logaritmov operandov, 2. seštevanje logaritmov operandov, 3. izračun antilogaritma vsote. Problem te enostavne ideje je, kako dobimo logaritemske in antilogaritemske vrednosti operandov. Ker teh vrednosti ne moremo izračunati točno, uporabimo aproksimacijo funkcij logaritma in antilogaritma.

Neeksaktna aritmetika predstavljaja kompromis med natančnostjo in časom izvajanja. Poraba moči pa je pri poenostavljenih metodah običajno manjša.

2.3. Hitrost, točnost in aplikacije DPS

V mnogih aplikacijah DPS, ki se izvajajo v realnem času, je hitrost najbolj pomemben parameter pri načrtovanju. Doseganje višjih hitrosti je možno tudi na račun določene tolerance do napake aritmetičnih operacij. Pri procesiranju signalov imamo opravka s signali, ki so popačeni s šumom, ki ga ustvarjajo: neidealni senzorji, postopki kvantizacije, ojačevalniki, in podobno, prav tako pa napako ustvarjajo algoritmi, ki v osnovi temeljijo na določenih predpostavkah. Netočni rezultati so torej neizogibni tudi s strani same aplikacije.

En primer je frekvenčno puščanje (ang. frequency leakage), ki ima za posledico napačne magnitude frekvenčnih vzorcev pri spektralni analizi.

Tehnike za kompresijo signalov po kosinusni ali valovni transformaciji vsebujejo postopek kvantizacije. Pri kvantizaciji koeficientov transformacije je manj smiselno računati koeficiente visoke natančnosti in jih nato odrezati na določeno število mest, kot pa enostavno prihraniti dragocene resurse in pred postopkom kvantizacije izračunavati manj točne rezultate.

V mnogih algoritmih za procesiranje signalov, ki vključujejo računanje korelacije med signali, pa na primer točen rezultat sploh ni pomemben, le maksimalna vrednost korelacije.

Dodatne majhne napake, ki jih povzročajo neeksaktni množilniki, ne vplivajo bistveno na rezultat in so v praksi sprejemljive.

V aplikacijah, kjer je bolj pomembna hitrost računanja kot točnost rezultata, se za najbolj primerni metodi izkažeta odrezano množenje in množenje v logaritemskem številskem sistemu [13, 24].

Poglavje 3

Matrični množilnik

Matrični množilnik (ang. array multiplier) je aritmetično eksakten. Sestavljen je iz polnih seštevalnikov v matriki podobni strukturi, kot je prikazano na sliki 4. Njegova struktura je zelo regularna, s kratkimi povezavami, ki povezujejo horizontalne, vertikalne in diagonalne polne seštevalnike med seboj. Sestavljen je iz $(n-2)\cdot(n-1)$ 1-bitnih seštevalnikov s shranjevanjem prenosa (ang. Carry Save Adders, CSA), zadnja vrstica pa je sestavljena iz $n-1$ 1-bitnih seštevalnikov s plazovitim prenosom (ang. ripple-carry adders). Ravno zaradi regularnosti ima enostavno in učinkovito postavitev na VLSI (ang. Very Large Scale Integration).

Algoritem za matrično množenje je enostaven, potrebuje le operaciji pomik in seštevanje. Pri množenju potrebujemo za to veliko število seštevalnikov. Množenje dveh n-bitnih števil N_1 in N_2 namreč lahko vedno opišemo kot n seštevanj ustrezno pomaknjenih števil. Naj bo i-ti bit z desne števil N_1 in N_2 označen z n_{1i} in n_{2i}. Potem je produkt matričnega množilnika P_{MM}:

$$P_{MM} = N_1 N_2 = \left(\sum_{i=0}^{n-1} n_{1i}\, 2^i\right)\left(\sum_{j=0}^{n-1} n_{2j} 2^j\right) = \left(\sum_{i=0}^{n-1} 2^i\right)\left(\sum_{j=0}^{n-1} n_{1i} n_{2j} 2^j\right) \qquad (2)$$

Vsota, ki jo sestavljajo členi $n_{1i}n_{2j}2^j, j = 0,1, \ldots n-1$, je n-bitno dvojiško število, v katerem so posamezni biti določeni s konjunkcijo $n_{1i}n_{2i}$. Enačba pravi, da moramo sešteti n takih števil: zaradi faktorja 2^i je vsako naslednje število treba pomakniti za en bit v levo [12].

Na podlagi strojne implementacije je podan algoritem za matrično množenje.

Tabela 2: Algoritem za matrično množenje

Vhod:	N_1[0..n-1], N_2[0..n-1]
Korak 1:	$P_{MM}(0) = N_1(0)\cdot N_2(0)$
	$s_{0,1..n-2} = N_1(1..n-2)\cdot N_2(0)$
	$c_{0,1..n-1} = 0$
Korak 2:	za i=1..n-1 ponavljaj
	za j=1..n-2 ponavljaj
	$s_{i,j} = (s_{i-1,j} + c_{i-1,j} + (N_1(j-1)\cdot N_2(i))) \bmod 2$
	$c_{i,j} = (s_{i-1,j} + c_{i-1,j} + (N_1(j-1)\cdot N_2(i))) \text{ rem } 2$
	$s_{i,n-1} = (c_{i-1,n-1} + (N_1(n-2)\cdot N_2(i)) + (N_1(n-1)\cdot N_2(i-1))) \bmod 2$
	$c_{i,n-1} = (c_{i-1,n-1} + (N_1(n-2)\cdot N_2(i)) + (N_1(n-1)\cdot N_2(i-1))) \text{ rem } 2$
	$P_{MM}(i) = s_{i,1}$
Korak 3:	$P_{MM}(n) = s_{n,1} = (s_{n-1,1} + c_{n-1,1}) \bmod 2$
	$c_{n,1} = (s_{n-1,1} + c_{n-1,1}) \text{ rem } 2$
	za j=2..n-2 ponavljaj
	$P_{MM}(n+j-1) = s_{n,j} = (s_{n-1,j} + c_{n-1,j} + c_{n,j-1}) \bmod 2$

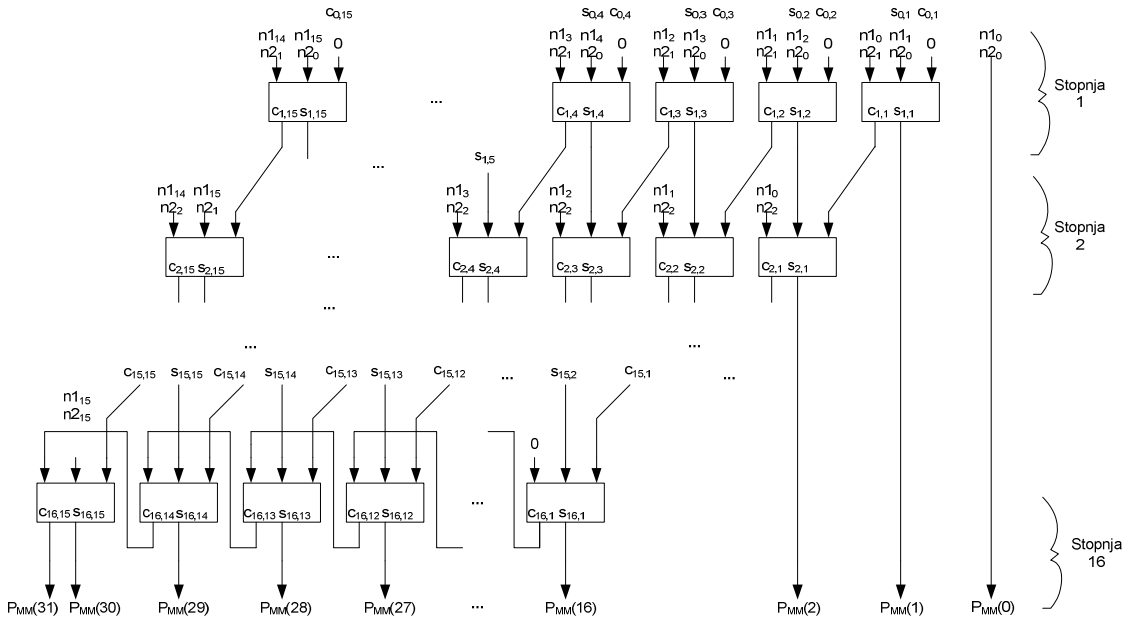
	$c_{n,j} = (s_{n-1,j} + c_{n-1,j} + c_{n,j-1}) \text{ rem } 2$
	$P_{MM}(2n-2) = s_{n,n-1} =$ $= (c_{n-1,n-1} + c_{n,n-2} + (N_1(n-1) \cdot N_2(n-1)) + (N_1(n-1) \cdot N_2(n-1))) \bmod 2$
	$P_{MM}(2n-1) = c_{n,n-1} =$ $= (c_{n-1,n-1} + c_{n,n-2} + (N_1(n-1) \cdot N_2(n-1)) + (N_1(n-1) \cdot N_2(n-1))) \text{ rem } 2$

Delovanje algoritma za matrično množenje je prikazano v spodnjem primeru.

Tabela 3: Primer množenja z algoritmom za matrično množenje

Vhod:	$N_1 = 18 = (00010010)_2$, $N_2 = 58 = (00111010)_2$
	$P_{MM}(0) = 0$
	$P_{MM}(1) = s_{11}(0,0,0) = 0$
	$P_{MM}(2) = s_{21}(0,1,0) = 1$
	$P_{MM}(3) = s_{31}(0,0,0) = 0$
	$P_{MM}(4) = s_{41}(0,1,0) = 1$
	$P_{MM}(5) = s_{51}(0,0,0) = 0$
	$P_{MM}(6) = s_{61}(0,0,0) = 0$
	$P_{MM}(7) = s_{71}(0,1,1) = 0$
	$P_{MM}(8) = s_{81}(1,1,0) = 0$
	$P_{MM}(9) = s_{91}(1,1,0) = 0$
	$P_{MM}(10) = s_{A1}(1,0,0) = 1$
	$P_{MM} = (0000010000010100)_2 = 1.044$

3.1. Strojna implementacija algoritma



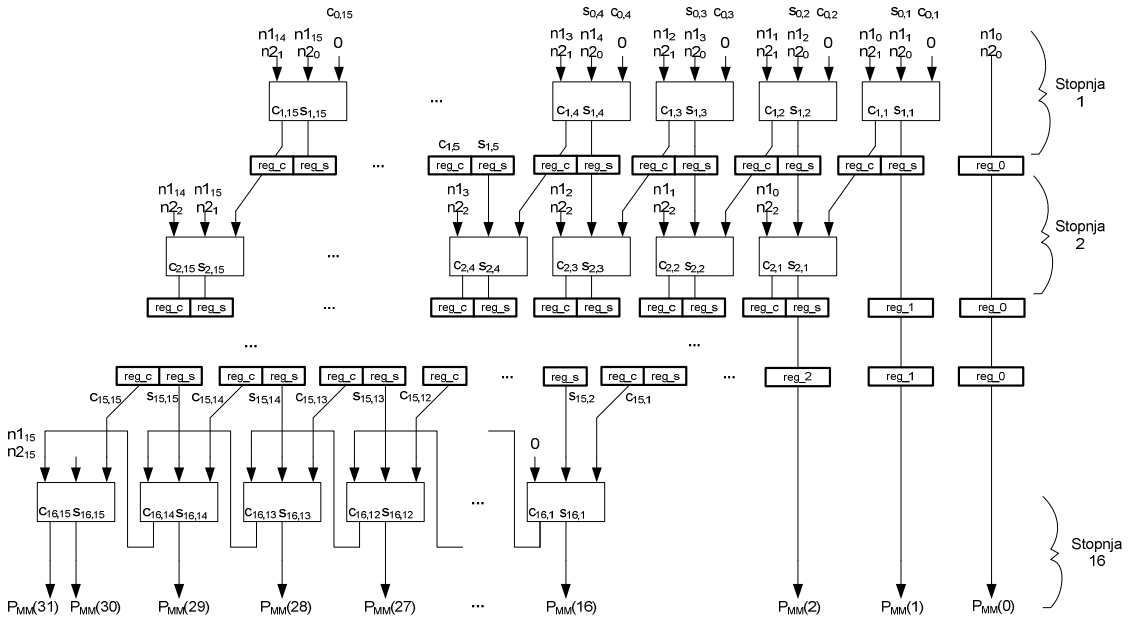
Slika 4: Matrični množilnik za množenje 16-bitnih nepredznačenih števil

Vsak od pravokotnikov je polni 1-bitni seštevalnik.

Za realizacijo potrebujemo $(n-1)^2$ 1-bitnih polnih seštevalnikov ter n^2 »in« logičnih vrat za konjunkcije $n_{1i}n_{2j}$, kjer je n dolžina števil N_1 in N_2.

Implementacija 16-bitnega matričnega množilnika vsebuje 1x 17-bitni seštevalnik in 448x 2-bitnih seštevalnikov.

3.2. Implementacija matričnega množilnika s cevovodom



Slika 5: 16-stopenjski cevovodni matrični množilnik za množenje 16-bitnih nepredznačenih števil

Vsi vhodi in prenosi iz posameznih stopenj cevovoda so registrirani, in vhodi tudi primerno zakasnjeni, kar zaradi preglednosti na sliki ni prikazano.

Implementacija 16-bitnega matričnega množilnika s cevovodom vsebuje 1x 17-bitni seštevalnik, 448x 2-bitnih seštevalnikov, 61x 16-bitnih registrov, 1x 1-bitni register, 1x 2-bitni register, 1x 3-bitni register, 1x 4-bitni register, 1x 5-bitni register, 1x 6-bitni register, 1x 7-bitni register, 1x 8-bitni register, 1x 9-bitni register, 1x 10-bitni register, 1x 11-bitni register, 1x 12-bitni register, 1x 13-bitni register, 1x 14-bitni register, 1x 15-bitni register in 1x 32-bitni register.

3.3. Izkoriščenost naprave

Tabela 4: Izkoriščenost naprave Xilinx Spartan-3 za matrični množilnik

Množilnik [n=16]	Št. 4-vhodnih LUT tabel	Št. rezin	Št. flip-flop rezin	Št. V/I blokov
Matrični množilnik	816	424	67	66
Cevovodni matrični množilnik*	746	589	1.039	66

*16 stopenjski cevovod

Tabela 5: Maksimalna frekvenca matričnega množilnika na napravi Xilinx Spartan-3

Množilnik [n=16]	Maksimalna frekvenca [MHz]
Matrični množilnik	41,771
Cevovodni matrični množilnik*	235,026

*16 stopenjski cevovod

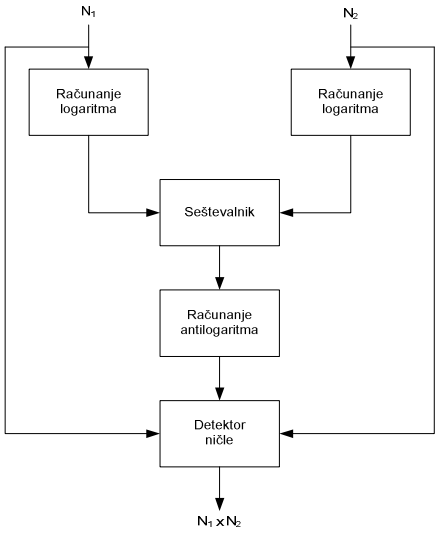
Matrični množilniki zahtevajo veliko število logičnih komponent in so pri današnji stopnji razvoja integriranih vezij še vedno razmeroma dragi [12]. Srečamo jih predvsem na zelo hitrih računalnikih in na nekaterih računalnikih za posebne namene (npr. pri digitalnih signalnih procesorjih).

Zakasnitev matričnega množilnika narašča linearno z n – ima kompleksnost $O(n)$, kjer je n dolžina vhodnih števil. Ker je vsaka od n vrstic seštevalnikov v matriki zasedena samo $\frac{1}{n}$ –ti del časa, je mogoče pričeti z naslednjim množenjem že veliko pred zaključkom prejšnjega (možna je nadgradnja s cevovodnim procesiranjem) [12]. Rezultati simulacije iz zgornje tabele prikazujejo 5,63-kratno pohitritev z uporabo cevovoda.

Poglavje 4

Mitchellov algoritem za množenje

Mitchellov algoritem (MA) za množenje je zasnovan na binarnem logaritmiranju, ki množenje zamenja s seštevanjem. Postopek je enostaven: logaritma vhodnih števil seštejemo in z antilogaritmom vsote dobimo produkt. Logaritma in antilogaritma ne moremo izračunati točno, zato potrebujemo njuno aproksimacijo. Mitchell je logaritem aproksimiral z odsekovno linijsko funkcijo[19]. Aproksimacijske vrednosti logaritma in antilogaritma dobimo kar iz števil samih s pomikanjem in seštevanjem. V primeru, da je eden od vhodnih števil enak nič, se za zagotovitev ničle na izhodu uporabi detektor ničle.



Slika 6: Logaritemsko množenje dveh števil

MA za množenje dveh vhodnih n-bitnih števil N_1 in N_2 je podan v spodnji tabeli. Logaritem operanda je zapisan v obliki karakterističnega dela in mantise:

Tabela 6: MA za množenje

Vhod:	N_1, N_2: n-bitna vhodna operanda
Korak 1:	Izračunaj $k_1 \leftarrow$ pozicija vodilnega bita '1' števila N_1
Korak 2:	Izračunaj $k_2 \leftarrow$ pozicija vodilnega bita '1' števila N_2
	% k_1 in k_2 tvorita karakteristični del logaritmov N_1 in N_2
Korak 3:	Izračunaj $x_1 \leftarrow N_1$ pomakni v levo za n-k_1 bitov
Korak 4:	Izračunaj $x_2 \leftarrow N_2$ pomakni v levo za n-k_2 bitov
	% x_1 in x_2 tvorita mantisi logaritmov N_1 in N_2
Korak 5:	Izračunaj k_{12}=$k_1 + k_2$
Korak 6:	Izračunaj x_{12}=$x_1 + x_2$
Korak 7:	Če je $x_{12} \geq 2^n$ (torej $x_1 + x_2 \geq 1$): (i) $k_{12} = k_{12} + 1$

	(ii) Dekodiraj k_{12} in vstavi x_{12} v P_{MA} na to pozicijo Če pogoj ne velja: (i) Dekodiraj k_{12} in vstavi '1' v P_{MA} na to pozicijo (ii) V P_{MA} vstavi x_{12} takoj za to enico
Korak 8:	Aproksimiraj $N_1 N_2 = P_{MA}$

Pozicija vodilnih enic števil N_1 in N_2 je določena s pomikom bitov v levo, dokler na prvem mestu besede ne dobimo vrednosti '1'. Ob vsakem pomiku se dekrementira števec, ki je imel ob inicializaciji vrednost velikosti besede. Končna vrednost števcev k_1 in k_2 tvorita celi del logaritmov števil N_1 in N_2. Preostala sekvenca bitov tvori ulomkovni del, to je mantisi x_1 in x_2. Obe vrednosti seštejemo v k_{12} in x_{12}. Glede na pogoj $x_1 + x_2 \geq 1$ dekodiramo vrednost karakterističnega dela vsote k_{12} (celi del). Tako dobimo pozicijo končnega produkta, kamor se vstavi vodilna enica. Mantisa vsote x_{12} (decimalni del) se vstavi takoj na desno stran enice. Podan je primer izračuna množenja števil 234 in 198 z MA.

Tabela 7: Primer množenja z MA

Vhod:	N_1 = 234 = $(11101010)_2$, N_2 = 198 = $(11000110)_2$
Korak 1:	k_1 = $(0111)_2$
Korak 2:	k_2 = $(0111)_2$
Korak 3:	x_1 = $(11010100)_2$
Korak 4:	x_2 = $(10001100)_2$
Korak 5:	$k_{12} = k_1 + k_2$= $(1110)_2$
Korak 6:	$x_{12} = x_1 + x_2$= $(101100000)_2 \geq 2^8$
Korak 7:	$x_{12} > 2^8$, $k_{12} = k_{12} + 1$ = $(1111)_2$
Korak 8:	P_{MA} = $(1011000000000000)_2$ = 45.056 (Relativna napaka je 2,754%)
Točen rezultat:	$P_{true} = 46.332$

Hitro opazimo, da je MA enostaven za izračun, in zahteva samo operacije pomika in seštevanja. V podanem primeru je relativna napaka 2,754%.

4.1. Analiza napake Mitchellovega algoritma

MA je zasnovan na aproksimaciji logaritma in antilogaritma, ki prinaša precejšnjo napako. Mitchell v [19] prikazuje, da pride do napake množenja zaradi ulomkovnega dela logaritma, in narašča s številom vrednosti '1' v mantisi števila. Prikazani so pomembnejši deli Mitchellove analize napake.

Naj bo N binarno število na intervalu $2^{k+1} > N \geq 2^j$ in $k \geq j$. Število N lahko zapišemo kot:

$$N = \sum_{i=j}^{k} 2^i Z_i \tag{3}$$

Ker je Z_k najbolj pomemben bit, lahko predpostavimo $Z_k = 1$ za vsak $k \geq j$. S faktorizacijo vrednosti 2^k iz N dobimo:

$$N = 2^k \left(1 + \sum_{i=j}^{k-1} 2^{1-k} Z_i \right) \tag{4}$$

Uvedemo novo spremenljivko za drugi argument v oklepajih:

$$\sum_{i=j}^{k-1} 2^{i-k} Z_i = x \tag{5}$$

Ker je $k \geq j$, bo vrednost x na intervalu $0 \leq x < 1$ in se imenuje ulomkovni člen ali mantisa binarnega števila N. Število N je sedaj predstavljeno kot:

$$N = 2^k (1 + x) \tag{6}$$

kjer je k karakteristični del števila ali položaj vodilne enice. Pravi logaritem binarnega števila in Mitchellova aproksimacija logaritma sta v tem zaporedju podana v enačbah (7) in (8). Z lg je označen logaritem z osnovo 2.

$$lg(N)_{true} = k + lg(1 + x) \tag{7}$$

$$lg(N)_{MA} = k + x \tag{8}$$

Iz zgornjih enačb vidimo aproksimacijo Mitchellove metode: lg $(1 + x)$ je aproksimiran z x. Če ne bi uporabili aproksimacije, bi morali uporabiti že izračunane vrednosti logaritmov iz vpogledne tabele (ang. Look-Up Table, LUT), kar je potratno v smislu porabe procesorske površine.

Naslednji dve enačbi predstavljata točno in aproksimacijsko vrednost logaritma produkta dveh števil:

$$lg(N_1 * N_2)_{true} = k_1 + \lg(1 + x_1) + k_2 + \lg(1 + x_2) \tag{9}$$

$$lg(N_1 * N_2)_{MA} = k_1 + x_1 + k_2 + x_2 \tag{10}$$

Z antilogaritmom enačbe (9) dobimo točen produkt P_{true}:

$$P_{true} = 2^{k_1 + k_2} (1 + x_1)(1 + x_2) \tag{11}$$

Mitchell je v svojem delu predlagal naslednjo aproksimacijo produkta P_{MA}:

$$P_{MA} = (\mathrm{N}_1 \cdot \mathrm{N}_2)_{\mathrm{MA}} = \begin{cases} 2^{k_1 + k_2}(1 + x_1 + x_2), & x_1 + x_2 < 1 \\ 2^{k_1 + k_2 + 1}(x_1 + x_2), & x_1 + x_2 \geq 1 \end{cases} \tag{12}$$

kjer je P_{MA} odvisen od prenosa vsote mantis $(x_1 + x_2)$.

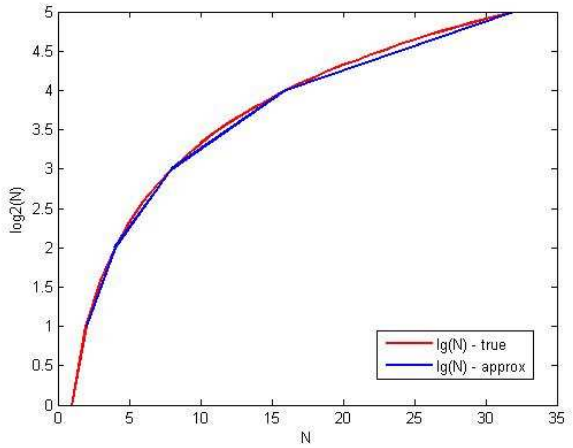
Relativna napaka množenja z MA E_r je tako definirana kot:

$$E_r = \frac{P_{true} - P_{MA}}{P_{true}} \tag{13}$$

kjer je P_{true} točna vrednost produkta z uporabo standardnega binarnega množenja. Če v enačbi (13) nadomestimo člena P_{true} in P_{MA} z (11) in (12), dobimo:

$$E_r = \begin{cases} \dfrac{1 + \mathrm{x}_1 + \mathrm{x}_2}{(1 + \mathrm{x}_1)(1 + \mathrm{x}_2)} - 1, & x_1 + x_2 < 1 \\ \dfrac{2(\mathrm{x}_1 + \mathrm{x}_2)}{(1 + \mathrm{x}_1)(1 + \mathrm{x}_2)} - 1, & x_1 + x_2 \geq 1 \end{cases} \tag{14}$$

Enačba (14) predstavlja napako množenja z MA, ki je odvisna od prenosa vsote mantis $(x_1 + x_2)$.



Slika 7: Krivulji točne vrednosti binarnega logaritma in aproksimirane vrednosti MA logaritma

Zgornja slika prikazuje potek točne vrednosti binarnega logaritma in aproksimirane vrednosti Mitchellovega logaritma. V točkah, kjer je mantisa (ulomkovni člen) enaka '0', imata krivulji isto vrednost. V tabeli spodaj so prikazane vrednosti binarnega logaritma (stolpec Točna vrednost $\lg N$) in aproksimirane vrednosti Mitchellovega logaritma (stolpec Aproksimirana vrednost MA $\lg N$) za vrednosti N od 1 do 16.

Tabela 8: Vrednosti binarnega logaritma in aproksimacije MA logaritma

N	Točna vrednost $\lg N$	Aproksimirana vrednost MA $\lg N$
1	0,00000	0,000
2	1,00000	1,000
3	1,58496	1,500
4	2,00000	2,000
5	2,32193	2,250
6	2,58496	2,500
7	2,80735	2,750
8	3,00000	3,000
9	3,16992	3,125

10	3,32193	3,250
11	3,45942	3,375
12	3,58496	3,500
13	3,70043	3,625
14	3,80735	3,750
15	3,90689	3,875
16	4,00000	4,000

Absolutna vrednost napake Mitchellove aproksimacije logaritma se nahaja na intervalu $0 \leq napaka \leq 0{,}08639$ in doseže maksimalno vrednost, ko ima ulomkovni člen aproksimacije algoritma vrednost $0{,}44$ [19].

MA ima precejšnjo relativno napako. Relativna napaka se povečuje s številom bitov mantise, ki imajo vrednost '1'. Napaka se nahaja na intervalu $0 \leq napaka \leq -11{,}1\%$ in doseže maksimalno vrednost pri $x_1 = x_2 = \frac{1}{2}$. Povprečna relativna napaka pa znaša 3,8% [17, 19]. Napako lahko zmanjšamo z več zaporednimi množenji.

Mitchell je predlagal naslednji analitični izraz za korekcijo napake:

$$P_{MAC} = (\mathrm{N_1} \cdot \mathrm{N_2})_{\mathrm{MAC}} = \begin{cases} (\mathrm{N_1} \cdot \mathrm{N_2})_{\mathrm{MA}} + 2^{\mathrm{k_1}+\mathrm{k_2}}(\mathrm{x_1} \cdot \mathrm{x_2}),\ \mathrm{x_1} + \mathrm{x_2} < 1 \\ (\mathrm{N_1} \cdot \mathrm{N_2})_{\mathrm{MA}} + 2^{\mathrm{k_1}+\mathrm{k_2}}(1 - \mathrm{x_1})(1 - \mathrm{x_2}),\ \mathrm{x_1} + \mathrm{x_2} \geq 1 \end{cases} \quad (15)$$

kjer sta $2^{\mathrm{k_1}+\mathrm{k_2}}(\mathrm{x_1} \cdot \mathrm{x_2})$ in $2^{\mathrm{k_1}+\mathrm{k_2}}(1 - \mathrm{x_1})(1 - \mathrm{x_2})$ korekcijska člena Mitchellove korekcije.

Za izračun korekcijskih členov moramo izračunati produkt $(\mathrm{x_1} \cdot \mathrm{x_2})$ ali $(1 - \mathrm{x_1})(1 - \mathrm{x_2})$, odvisno od vrednosti $\mathrm{x_1} + \mathrm{x_2}$ po enačbi (12), ga pomakniti za faktor $2^{\mathrm{k_1}+\mathrm{k_2}}$ in prišteti z vrednostjo $(\mathrm{N_1} \cdot \mathrm{N_2})_{\mathrm{MA}}$ v produkt. Korekcijo napake lahko izvedemo iterativno, vendar šele po tem, ko izračunamo vrednost člena $\mathrm{x_1} + \mathrm{x_2}$. Napako lahko zmanjšamo do poljubne vrednosti. MA z enim korekcijskim členom ima maksimalno napako 2,8% [19].

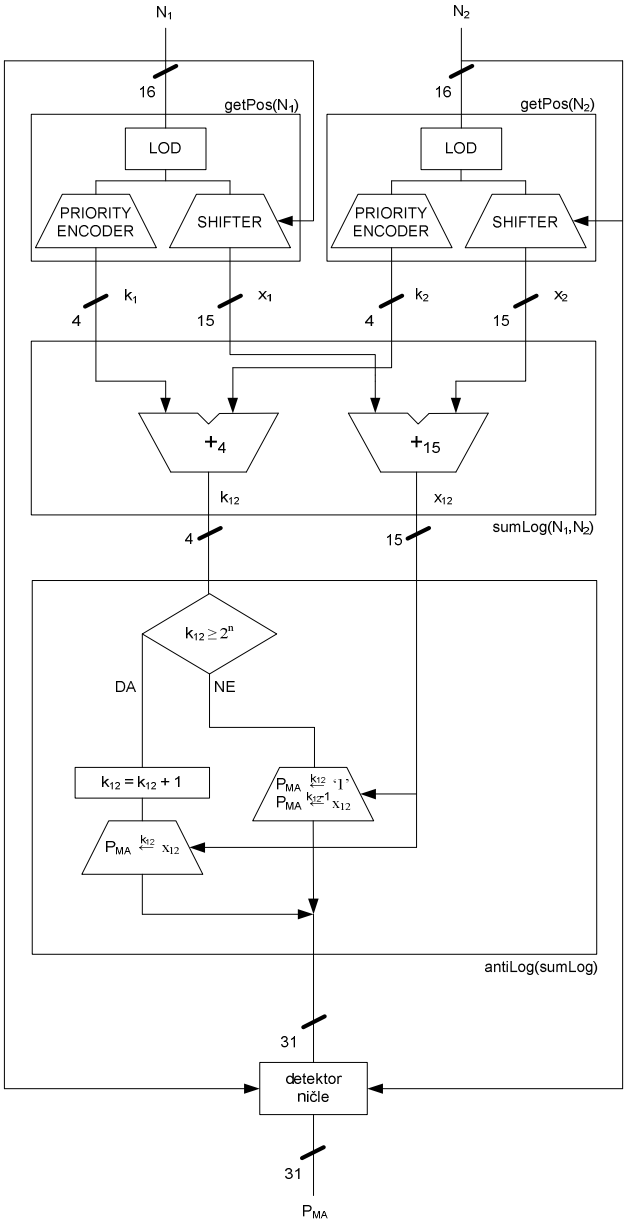
Številna prizadevanja za izboljšanje točnosti MA le povečujejo kompleksnost sistema. Hall [8] je izpeljal različne enačbe za korekcijo napak logaritemske in antilogaritemske aproksimacije v štirih ločenih območjih, odvisno od vrednosti mantise, s katerimi je zmanjšal povprečno napako na 2%. Abed in Siferd [1, 2], sta izpeljala korekcijske enačbe s koeficienti v obliki eksponentov števila 2, ki zmanjšujejo napako pri isti enostavnosti rešitve. Med drugimi metodami, ki za korekcijo napake MA uporabljajo vpogledne tabele, je McLarenova metoda [18]. McLarenova metoda uporablja vpogledno tabelo s 64 korekcijskimi koeficienti, ki se izračunajo v odvisnosti od vrednosti mantis, in ima sprejemljivo točnost ter kompleksnost.

Novejši pristop h korekciji napake MA, ki zmanjšuje število bitov mantise z vrednostjo '1', se imenuje Mitchellov algoritem z dekompozicijo operandov (OD-MA) in je opisana v petem poglavju.

Korekcijo napake lahko pri MA izvedemo iterativno, vendar šele po tem, ko izračunamo vrednost člena $\mathrm{x_1} + \mathrm{x_2}$. Enostavni iterativni množilnik (SIM), ki je opisan v šestem poglavju, s poenostavitvijo aproksimacije logaritma iz enačbe (12), računa korekcijske člene skoraj takoj, ko se prične izračun aproksimacije produkta. Tako vrsto paralelizma se lahko

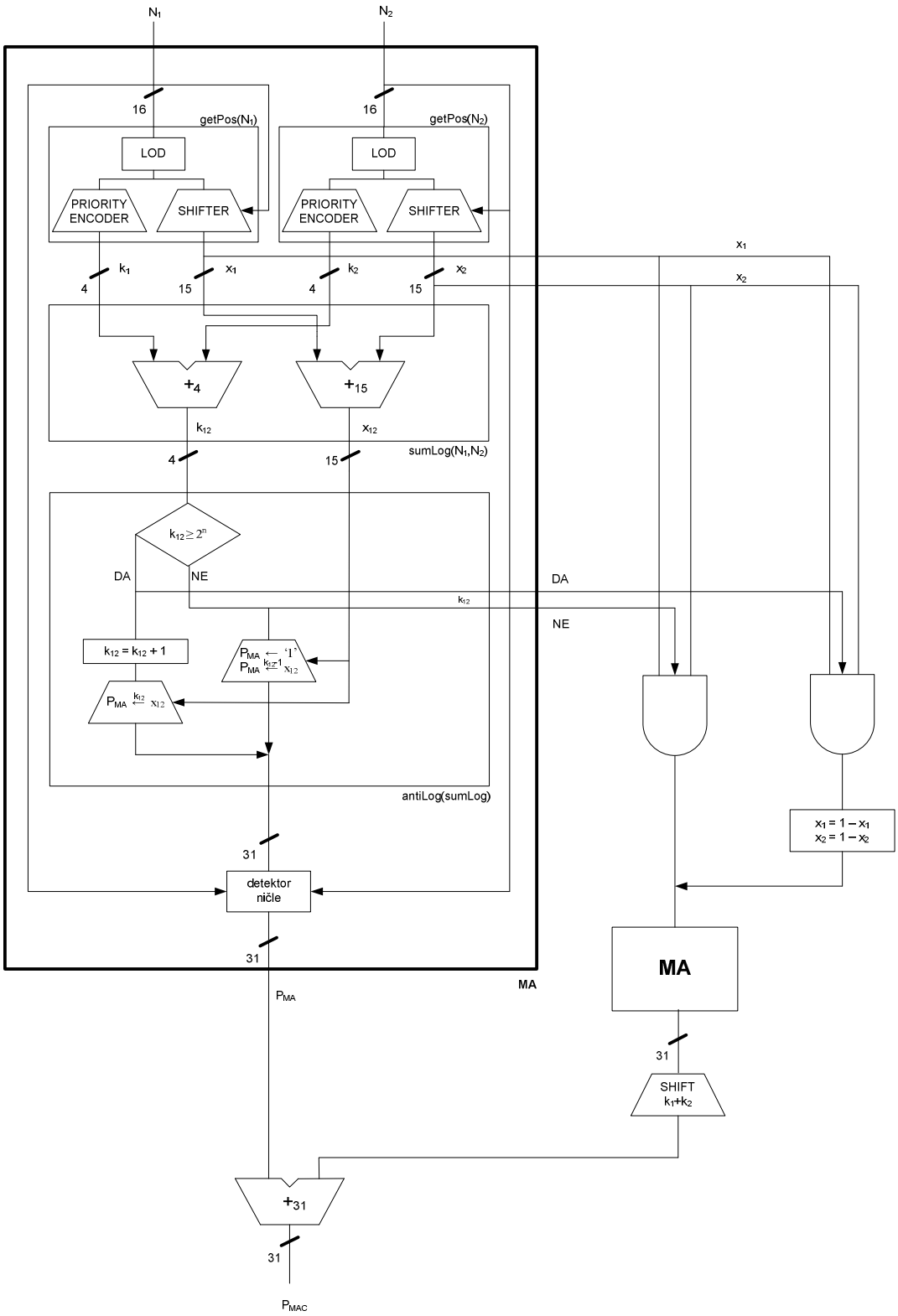
učinkovito implementira s cevovodom in s tem zmanjša kompleksnost potrebne logike in poviša hitrost izvajanja.

4.2. Strojna implementacija algoritma



Slika 8: Arhitektura 16-bitnega množilnika MA

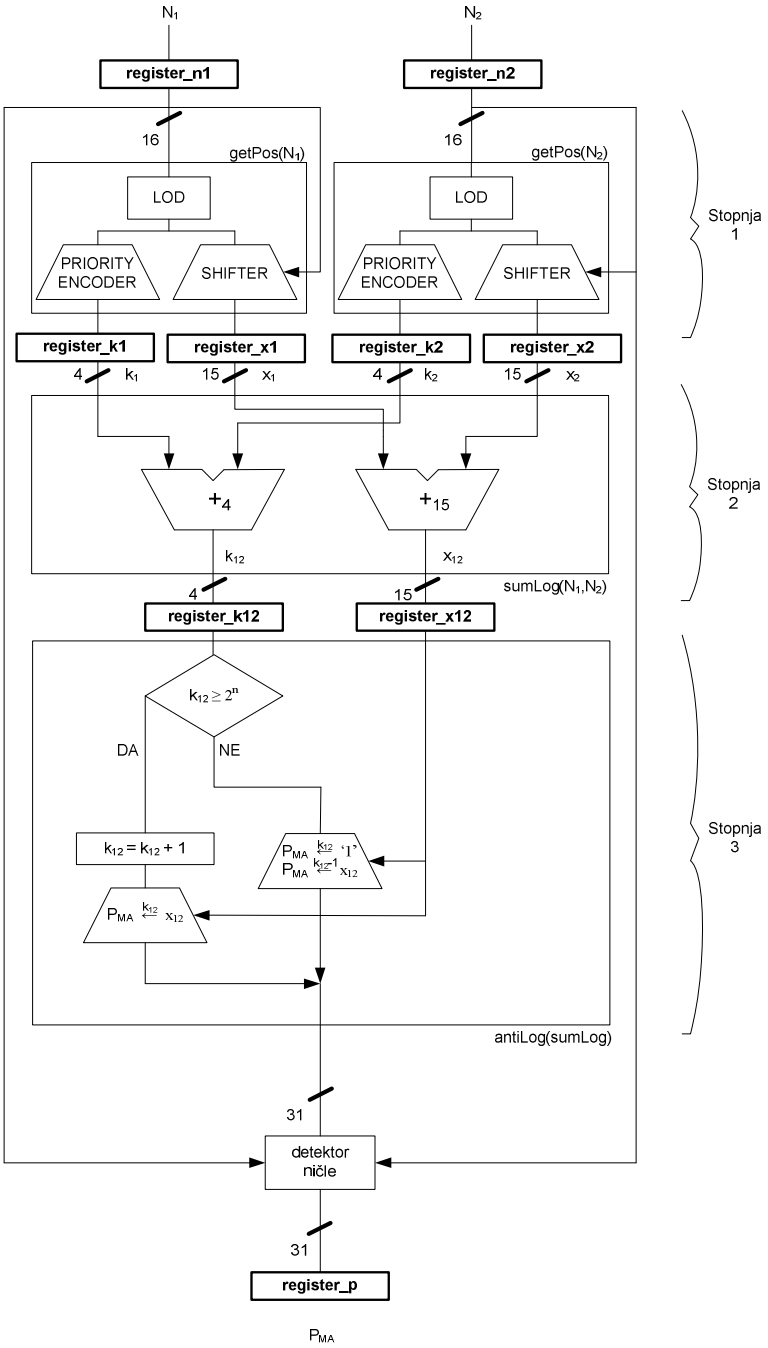
Implementacija 16-bitnega množilnika MA vsebuje tri seštevalnike, 5-bitnega za izračun karakterističnega dela, 16-bitnega za izračun mantise in 32-bitnega za seštevanje delnih vsot produktov (v enoti antiLog – ni razvidno iz slike), 32-bitni logični pomikalnik (ang. shifter) v desno in 32-bitni 4-v-1 multiplekser za izračun operacij logaritma (detekcija vodilne enice) in antilogaritma (dekodiranje vsote mantise s karakterističnim delom).



Slika 9: Arhitektura 16-bitnega množilnika MA z enonivojsko korekcijo

Implementacija 16-bitnega množilnika MA z enonivojsko korekcijo vsebuje dva MA množilnika, 32-bitni logični pomikalnik (ang. shifter) in dve »in« vrati.

4.3. Implementacija Mitchellovega algoritma s cevovodom



Slika 10: Arhitektura cevovodnega 16-bitnega množilnika MA

Implementacija 16-bitnega množilnika MA s cevovodom vsebuje tri seštevalnike: 5-bitnega za izračun karakterističnega dela, 16-bitnega za izračun mantise in 32-bitnega za seštevanje delnih vsot produktov (v enoti antiLog – ni razvidno iz slike), 32-bitni logični pomikalnik (ang. shifter) v desno, 32-bitni 4-v-1 multiplekser za izračun operacij logaritma (detekcija vodilne enice) in antilogaritma (dekodiranje vsote mantise s karakterističnim delom), 1x 32-bitni register, 2x16-bitni register, 3x 15-bitni register in 3x 4-bitni register.

4.4. Izkoriščenost naprave

Tabela 9: Izkoriščenost naprave Xilinx Spartan-3 za MA

Množilnik [n=16]	Št. 4-vhodnih LUT tabel	Št. rezin	Št. flip-flop rezin	Št. V/I blokov
MA	598	308	80	66
Cevovodni MA*	554	290	168	66

*3 stopenjski neuravnotežen cevovod

Tabela 10: Maksimalna frekvenca MA na napravi Xilinx Spartan-3

Množilnik [n=16]	Maksimalna frekvenca [MHz]
MA	51,807
Cevovodni MA*	86,790

*3 stopenjski neuravnotežen cevovod

Izkoriščenost naprave je v primerjavi z matičnim množilnikom boljše. Na MA zasnovan množilnik porabi za 36% manjše število 4-vhodnih LUT tabel in 38% manjše število rezin.

Frekvenca necevovodnega matričnega množilnika je za 10 MHz manjša od MA. MA s cevovodom prinaša 1,67-kratno pohitritev.

Poglavje 5

Mitchellov algoritem z dekompozicijo operandov

Mitchellov algoritem z dekompozicijo operandov (OD-MA) je metoda za izboljšanje točnosti množenja MA, in je bila prvič predstavljena v [11]. Dekompozicija operandov (OD) se izvaja nad vhodnimi operandi in tako predstavlja predprocesni korak MA. Z OD se zmanjša število bitov v operandu, ki imajo vrednost '1'. Posledično se zmanjša tudi število prenosov v koraku seštevanja mantis logaritmov, in izboljša točnost MA, kar je matematično dokazano v [17].

Predspostavimo dva n-bitna operanda N_1 in N_2:

$$N_1 = [x_{n-1}x_{n-2} \dots x_2x_1x_0]$$
$$N_2 = [y_{n-1}y_{n-2} \dots y_2y_1y_0] \tag{16}$$

Operanda N_1 in N_2 sta dekomponirana v naslednje štiri operande A, B, C in D:

$$A = [\mathrm{a}_{n-1}\mathrm{a}_{n-2} \dots \mathrm{a}_2\mathrm{a}_1\mathrm{a}_0]$$
$$B = [b_{n-1}b_{n-2} \dots b_2b_1b_0]$$
$$C = [c_{n-1}c_{n-2} \dots c_2c_1c_0] \tag{17}$$
$$D = [d_{n-1}d_{n-2} \dots d_2d_1d_0]$$

kjer se posamezni biti dekomponiranih operandov izračunajo z uporabo naslednjih enačb:

$$a_i = x_i \vee y_i$$
$$b_i = x_i \wedge y_i$$
$$c_i = \sim x_i \wedge y_i \tag{18}$$
$$d_i = x_i \wedge \sim y_i$$

Produkt se nato iz dekomponiranih operandov izračuna na naslednji način:

$$N_1 * \mathrm{N}_2 = (A * B) + (C * D) \tag{19}$$

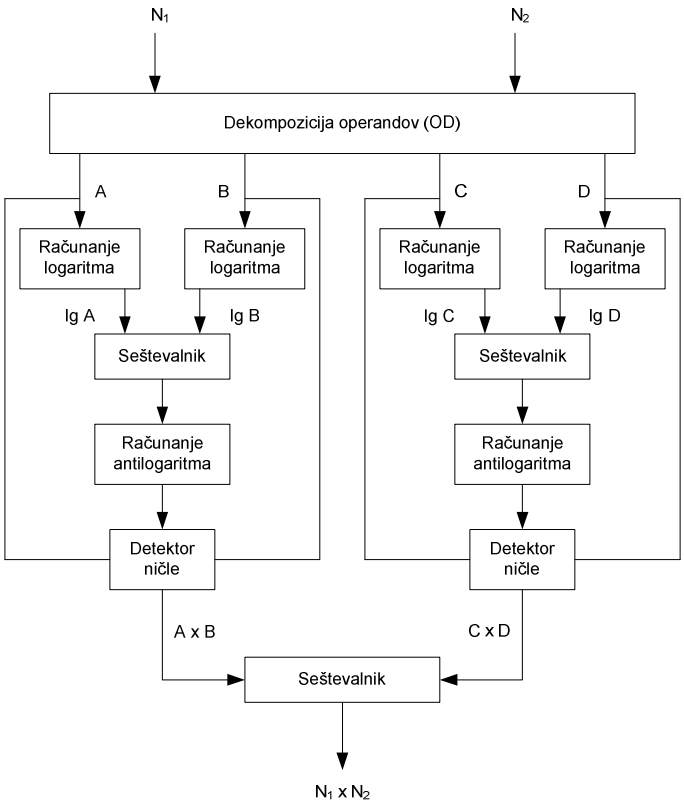
Zmanjševanje števila bitov, ki imajo vrednost '1', zmanjša porabo moči zaradi manjšega števila preklapljanj logičnih vrat pri izvajanju množenja. V [11] je analitično pokazano, da OD zmanjšuje verjetnost pojavitve bitov z vrednostjo '1' v operandih iz 0,5 na 0.25%. Spodnji primer prikazuje dekompozicijo števil 234 in 198.

Tabela 11: Primer OD

Vhod:	$N_1 = 234 = (11101010)_2$ $N_2 = 198 = (11000110)_2$
Dekompozicija operandov:	$A = (11101110)_2 = 238$ $B = (11000010)_2 = 194$ $C = (00000100)_2 = 4$ $D = (00101000)_2 = 40$

5.1. OD in MA

Točnost MA se izboljša z uporabo OD, ki nastopi kot predprocesni korak MA. Blok diagram algoritma OD-MA je prikazan na spodnji sliki.



Slika 11: Blokovna shema algoritma OD-MA

Na zgornji sliki vidimo, da je arhitektura OD-MA sestavljena iz OD, dveh paralelnih MA enot in seštevalnika. OD je enostavna in zahteva le 6 n-bitnih dvovhodnih »in«, »ali« in »negacija« vrat, kjer je n dolžina operanda.

Algoritem OD-MA je opisan v spodnji tabeli:

Tabela 12: Algoritem za množenje OD-MA

Vhod:	N_1, N_2: n-bitna vhodna operanda
Korak 1:	Dekompozicija števil N_1 in N_2 v števila A, B, C in D
Korak 2:	Izračunaj $k_A \leftarrow$ pozicija vodilnega bita '1' števila A
Korak 3:	Izračunaj $k_B \leftarrow$ pozicija vodilnega bita '1' števila B
	% $k_A$in $k_B$tvorita karakteristični del logaritmov A in B
Korak 4:	Izračunaj $x_A \leftarrow$ A pomakni v levo za n-k_A bitov
Korak 5:	Izračunaj $x_B \leftarrow$ B pomakni v levo za n-k_B bitov
	% x_A in x_B tvorita mantisi logaritmov A in B
Korak 6:	Izračunaj $k_{AB} = k_A + k_B$
Korak 7:	Izračunaj $x_{AB} = x_A + x_B$
Korak 8:	Če je $x_{AB} \geq 2^n$ (torej $x_A + x_B \geq 1$): (i) $k_{AB} = k_{AB} + 1$ (ii) Dekodiraj k_{AB} in vstavi x_{AB} v P_{AB} na to pozicijo Če pogoj ne velja: (i) Dekodiraj k_{AB} in vstavi '1' v P_{AB} na to pozicijo (ii) V P_{AB} vstavi x_{AB} takoj za to enico
Korak 9:	Aproksimiraj $AB = P_{AB}$
Korak 10:	Izračunaj $k_C \leftarrow$ pozicija vodilnega bita '1' števila C
Korak 11:	Izračunaj $k_D \leftarrow$ pozicija vodilnega bita '1' števila D
	% k_C in k_D tvorita karakteristični del logaritmov C in D
Korak 12:	Izračunaj $x_C \leftarrow$ C pomakni v levo za n-k_C bitov
Korak 13:	Izračunaj $x_D \leftarrow$ D pomakni v levo za n-k_D bitov
	% x_C in x_D tvorita mantisi logaritmov C in D
Korak 14:	Izračunaj $k_{CD} = k_C + k_D$
Korak 15:	Izračunaj $x_{CD} = x_C + x_D$
Korak 16:	Če je $x_{CD} \geq 2^n$ (torej $x_C + x_D \geq 1$): (i) $k_{CD} = k_{CD} + 1$ (ii) Dekodiraj k_{CD} in vstavi x_{CD} v P_{CD} na to pozicijo Če pogoj ne velja: (i) Dekodiraj k_{CD} in vstavi '1' v P_{CD} na to pozicijo (ii) V P_{CD} vstavi x_{CD} takoj za to enico
Korak 17:	Aproksimiraj $CD = P_{CD}$
Korak 18:	Aproksimiraj $N_1N_2 = P_{OD-MA} = P_A + P_B$

Vhodni števili N_1 in N_2 sta dekomponirani v števila A, B, C in D po algoritmu iz zgornje tabele. Oba para dekomponiranih operandov A in B ter C in D se nato ločeno zmnoži z MA v P_{AB} ter P_{CD} in sešteje v končni produkt P_{OD-MA}.

Primer s spodnje tabele prikazuje delovanje algoritma OD-MA. Števili N_1=234 in N_2=198 sta dekomponirani v štiri števila A=238, B=194, C=4 in D=40. Rezultata dekomponiranih produktov: P_{AB} =45.056 in P_{CD} =168, se nato seštejeta v produkt P_{OD-MA} =45.224.

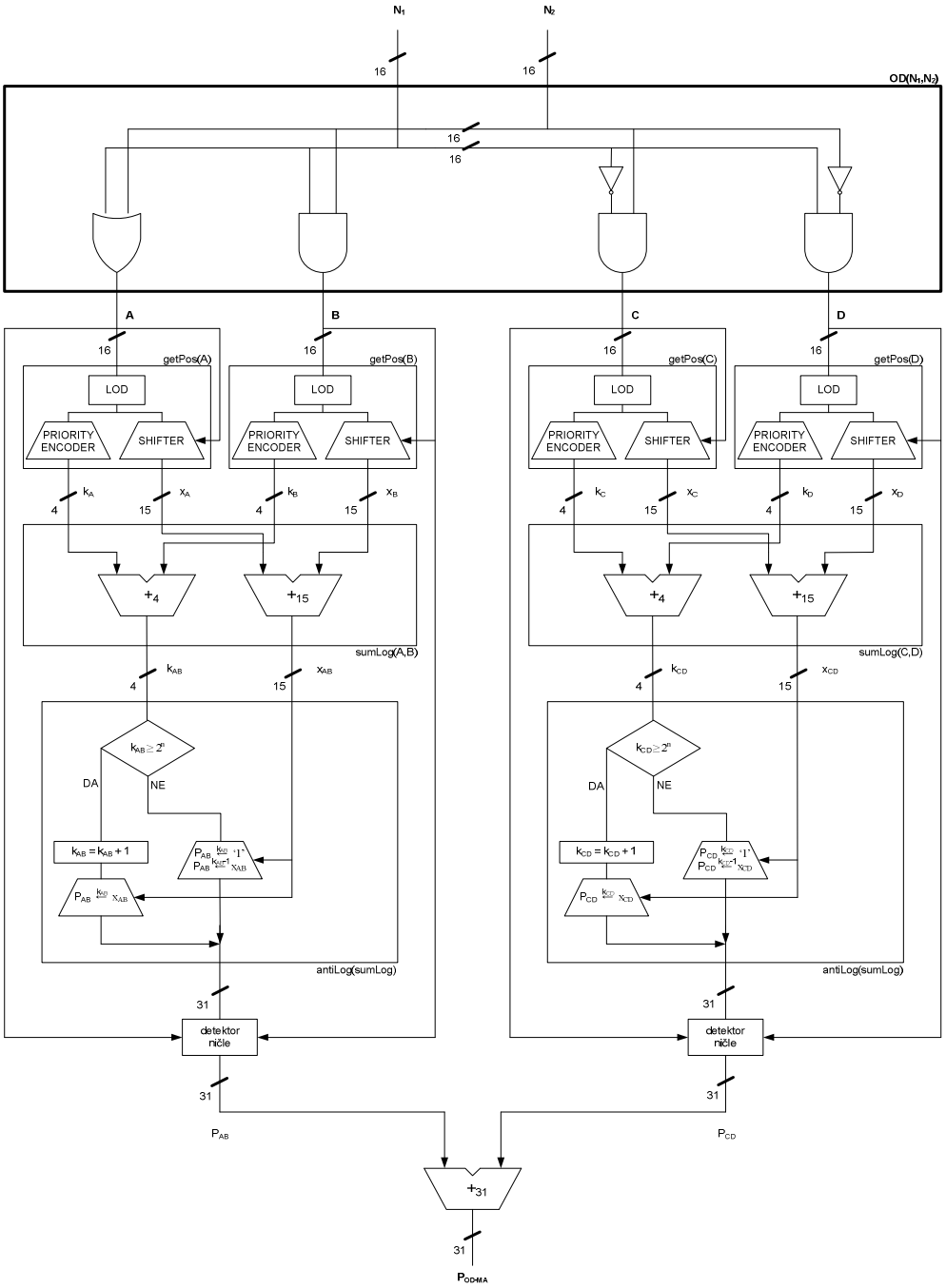
Tabela 13: Primer množenja z algoritmom OD-MA

Vhod:	N_1 = 234 = $(11101010)_2$, N_2 = 198 = $(11000110)_2$
Korak 1:	A = 238 = $(11101110)_2$ B = 194 = $(11000010)_2$ C = 4 = $(00000100)_2$ D = 40 = $(00101000)_2$
Korak 2:	k_A = $(0111)_2$
Korak 3:	k_B = $(0111)_2$
Korak 4:	x_A = $(11011100)_2$
Korak 5:	x_B = $(10000100)_2$
Korak 6:	$x_{AB} = x_A + x_B = (101100000)_2$
Korak 7:	$k_{AB} = k_A + k_B = (1110)_2$
Korak 8:	$x_{AB} > 2^8, k_{AB} = k_{AB} + 1 = (1111)_2$
Korak 9:	P_{AB} = $(1011000000000000)_2$ = 45.056
Korak 10:	k_C = $(0010)_2$
Korak 11:	k_D = $(0101)_2$
Korak 12:	x_C = $(00000000)_2$
Korak 13:	x_D = $(01010000)_2$
Korak 14:	$x_{CD} = x_C + x_D$ = $(01010000)_2$
Korak 15:	$k_{CD} = k_C + k_D$ = $(0111)_2$
Korak 16:	$x_{CD} < 2^8$
Korak 17:	P_{CD} = $(0000000010101000)_2$ = 168
Korak 18:	$N_1N_2 = P_{OD-MA} = P_{AB} + P_{CD}$ = = $(1011000010101000)_2$ = 45.224 (Relativna napaka je 2,391%)
Točen rezultat:	$P_{true} = 46.332$

Rezultat množenja z OD-MA z zgornjega primera lahko primerjamo z rezultatom množenja istih števil z MA, ki je podan v tabeli 7. Relativna napaka pri MA znaša 2,754%, pri OD-MA pa 2,391%. Pri množenju števil 234 in 198 doseže OD-MA za 13,18% boljši rezultat od MA.

Obstajajo tudi druge metode, ki se uporabljajo skupaj z OD: deljena aproksimacija in uporaba korekcij [17].

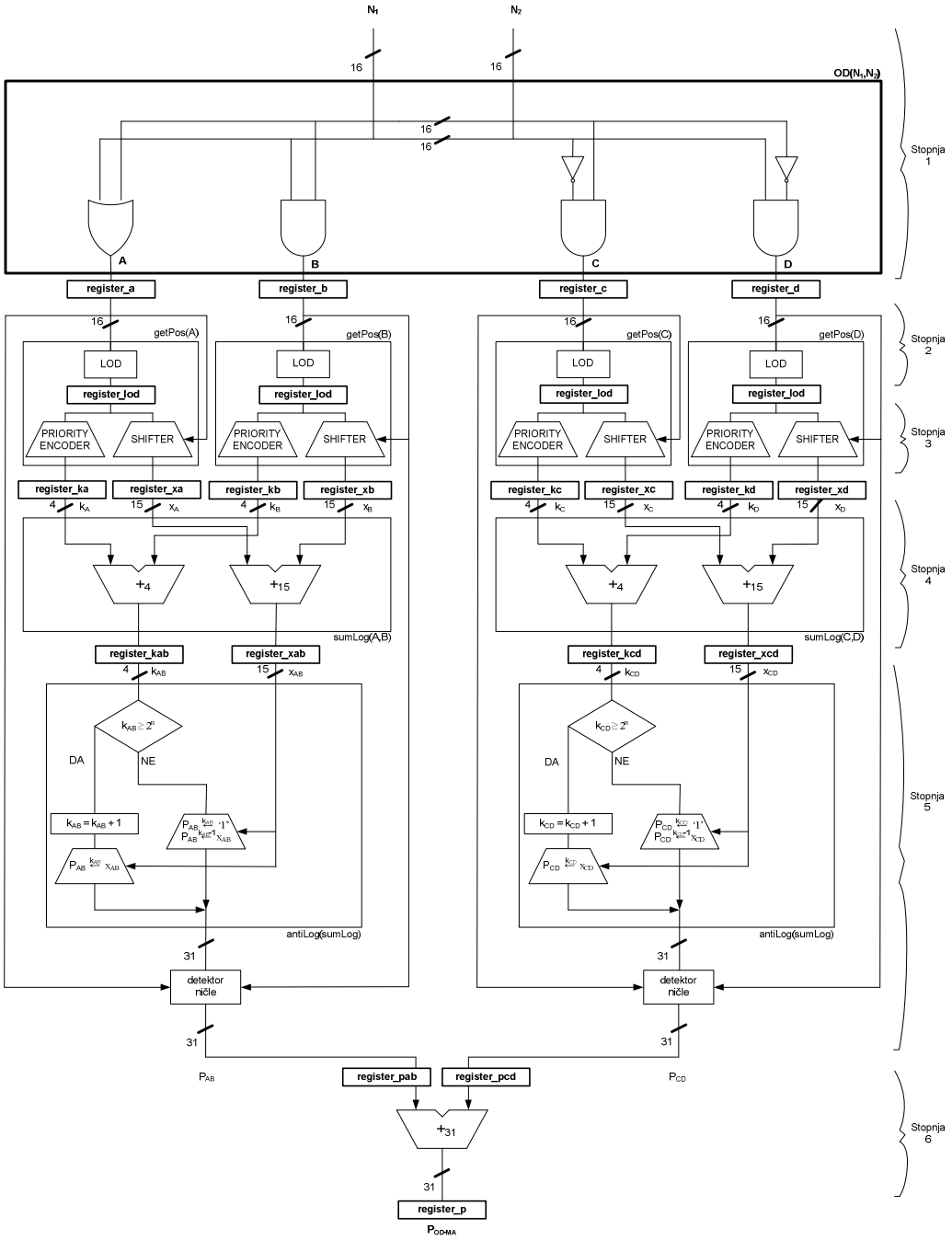
5.2. Strojna implementacija algoritma



Slika 12: Arhitektura 16-bitnega OD-MA množilnika

Implementacija 16-bitnega množilnika OD-MA vsebuje 2x 5-bitni seštevalnik za izračun karakterističnega dela, 2x 16-bitni seštevalnik za izračun mantise, 1x 32-bitni seštevalnik za vsoto delnih produktov, 2x 32-bitni logični pomikalnik (ang. shifter) v desno in 2 x32-bitna 4-v-1 multiplekserja za izračun operacij logaritma (detekcija vodilne enice, LOD) in antilogaritma (dekodiranje vsote mantise s karakterističnim delom).

5.3. Implementacija OD-MA množilnika s cevovodom



Slika 13: Arhitektura cevovodnega 16-bitnega OD-MA množilnika

Implementacija 16-bitnega množilnika OD-MA vsebuje 2x 5-bitni seštevalnik za izračun karakterističnega dela, 2x 16-bitni seštevalnik za izračun mantise, 1x 32-bitni seštevalnik za vsoto delnih produktov, 2x 32-bitni logični pomikalnik (ang. shifter) v desno, 2 x32-bitna 4-v-1 multiplekserja za izračun operacij logaritma (detekcija vodilne enice, LOD) in antilogaritma (dekodiranje vsote mantise s karakterističnim delom), 3x 32-bitni register, 12x 16-bitni register, 4x 15-bitni register, 4x 4-bitni register in 2x 1-bitni register.

5.4. Izkoriščenost naprave

Tabela 14: Izkoriščenost naprave Xilinx Spartan-3 za OD-MA

Množilnik [n=16]	Št. 4-vhodnih LUT tabel	Št. rezin	Št. flip-flop rezin	Št. V/I blokov
OD-MA	1.337	688	91	66
Cevovodni OD-MA*	1.112	593	362	66

*6 stopenjski cevovod

Tabela 15: Maksimalna frekvenca OD-MA na napravi Xilinx Spartan-3

Množilnik [n=16]	Maksimalna frekvenca [MHz]
OD-MA	39,682
Cevovodni OD-MA*	86,981

*6 stopenjski cevovod

V primerjavi z izkoriščenostjo naprave pri necevovodni implementaciji MA iz tabele 9 opazimo, da je število 4-vhodnih LUT tabel naraslo za faktor 2,24, število rezin pa za faktor 2,23.

Maksimalna frekvenca je v primerjavi z matričnim množilnikom (tabela 5) in MA (tabela 10) najmanjša, in sicer manjša za 2,089MHz in 12,125MHz v tem vrstnem redu. S cevovodnim izvajanjem dobimo 2,19-kratno pohitritev.

Poglavje 6

Enostavni iterativni množilnik

Enostavni iterativni množilnik (ang. Simple Iterative Multiplier, SIM) je novejši množilnik, zasnovan na istem principu predstavitve števil kot MA, a uporablja drugačno korekcijo za zmanjševanje napake. Z uporabo paralelnih korekcijskih vezij zmanjšujemo napako, ki je zelo majhna že pri uporabi dveh paralelnih korekcijskih vezij [4]. Korekcijske člene (KČ) lahko izračunamo skoraj takoj po izračunu aproksimacije produkta. Tako pa lahko z implementacijo cevovoda dosežemo visoko stopnjo paralelizma množilnika in povišamo hitrost izvajanja.

Če pogledamo binarno predstavitev števil v (4-6), lahko izpeljemo točno vrednost množenja:

$$\begin{aligned} P_{true} &= N_1 N_2 = 2^{k_1}(1 + x_1)2^{k_2}(1 + x_2) \\ &= 2^{k_1+k_2}(1 + x_1 + x_2) + 2^{k_1+k_2}(x_1 x_2) \end{aligned} \tag{20}$$

Da se izognemo aproksimacijski napaki, moramo upoštevati naslednjo relacijo:

$$x2^k = N - 2^k \tag{21}$$

Kombinacija enačb (20) in (21) nam da:

$$\begin{aligned} P_{true} = (N_1 N_2) &= 2^{(k_1+k_2)} + \\ +(N_1 - 2^{k_1})2^{k_2} + (N_2 - 2^{k_2})2^{k_1} &+ \\ +(N_1 - 2^{k_1})(N_2 - 2^{k_2}) & \end{aligned} \tag{22}$$

Naj bo:

$$P_{SIM}^{(0)} = 2^{(k_1+k_2)} + (N_1 - 2^{k_1})2^{k_2} + (N_2 - 2^{k_2})2^{k_1} \tag{23}$$

prva aproksimacija produkta. Lahko napišemo točno vrednost:

$$P_{true} = P_{SIM}^{(0)} + (N_1 - 2^{k_1})(N_2 - 2^{k_2}) \tag{24}$$

Metoda SIM je zelo podobna MA. Zadnji člen $(N_1 - 2^{k_1}) \cdot (N_2 - 2^{k_2})$ enačbe (22) zahteva množenje, zato ga enostavno zanemarimo. Tako dobimo aproksimacijo množenja $P_{SIM}^{(0)}$, ki zahteva samo nekaj pomikalnikov in seštevalnikov.

Enačba množilnika MA (12) zahteva primerjavo vsote $x_1 + x_2$ z '1'. Namesto, da bi ta pogoj zanemarili ali aproksimirali produkt na podoben način kot v enačbi (12), lahko produkt $(N_1 - 2^{k_1}) \cdot (N_2 - 2^{k_2})$ enostavno izračunamo z isto enačbo kot za izračun prve aproksimacije $P_{SIM}^{(0)}$.

Postopek lahko iterativno ponavljamo, dokler ne dobimo točnega rezultata. Ker pri SIM primerjava vsote $x_1 + x_2$ z '1' ni potrebna, lahko pričnemo s korekcijo takoj ko odstranimo vodilne enice vhodnih operandov N_1 in N_2. To nam tudi omogoča večjo stopnjo paralelizma pri cevovodnem izvajanju.

Absolutna napaka po prvi aproksimaciji je:

$$E^{(0)} = P_{true} - P_{SIM}^{(0)} = \\ = (N_1 - 2^{k_1})(N_2 - 2^{k_2}) \tag{25}$$

kjer je $E^{(0)} \geq 0$. Faktorja v (25) sta binarni števili, ki jih dobimo tako, da odstranimo vodilno enico števil N_1 in N_2. Sedaj lahko ponovimo korak množenja z novimi faktorji:

$$E^{(0)} = C^{(1)} + E^{(1)} \tag{26}$$

kjer je $C^{(1)}$ aproksimacija vrednosti $E^{(0)}$ in $E^{(1)}$ je absolutna napaka aproksimacije $E^{(0)}$.

Kombinacija enačb (24) in (26) je:

$$P_{true} = P_{SIM}^{(0)} + C^{(1)} + E^{(1)} \tag{27}$$

Zdaj lahko dodamo vrednost aproksimacije $E^{(0)}$ k aproksimaciji produkta $P_{SIM}^{(0)}$ kot KČ, s katerim zmanjšamo napako aproksimacije:

$$P_{SIM}^{(1)} = P_{SIM}^{(0)} + C^{(1)} \tag{28}$$

Če ponovimo postopek množenja z i KČ, lahko aproksimiramo produkt na naslednji način:

$$P_{SIM}^{(i)} = P_{SIM}^{(0)} + C^{(1)} + C^{(2)} + \ ... + C^{(i)} = \\ = P_{SIM}^{(0)} + \sum_{j=1}^{i} C^{(j)} \tag{29}$$

Postopek lahko ponavljamo, dokler ne dosežemo željene tolerance napake ali dokler zadnja mantisa ne doseže vrednosti '0'. Takrat je končni rezultat točen: $P_{SIM} = P_{true}$. Število iteracij, ki so potrebne za točen rezultat, je enako številu bitov z vrednostjo '1' v tistem od obeh operandov, ki ima manjše število bitov z vrednostjo '1'. Algoritem SIM je podan spodaj.

Tabela 16: SIM z *i* KČ

Vhod:	N_1, N_2: n-bitna vhodna operanda $P_{SIM}^{(0)} = 0$: 2n-bitna vrednost prve aproksimacije $C^{(i)} = 0$: 2n-bitni i-ti člen korekcije $P_{SIM}^{(i)} = 0$: 2n-bitni produkt
Korak 1:	Izračunaj $k_1 \leftarrow$ položaj vodilne enice števila N_1
Korak 2:	Izračunaj $k_2 \leftarrow$ položaj vodilne enice števila N_2
Korak 3:	Izračunaj $(N_1 - 2^{k_1})2^{k_2} \leftarrow$ pomakni $(N_1 - 2^{k_1})$ za k_2 bitov v levo
Korak 4:	Izračunaj $(N_2 - 2^{k_2})2^{k_1} \leftarrow$ pomakni $(N_2 - 2^{k_2})$ za k_1 bitov v levo
Korak 5:	Izračunaj $k_{12} = k_1 + k_2$
Korak 6:	Izračunaj $2^{(k_1+k_2)}$: dekodiraj k_{12}
Korak 7:	Izračunaj $P_{SIM}^{(0)} = 2^{(k_1+k_2)} + (N_1 - 2^{k_1})2^{k_2} + (N_2 - 2^{k_2})2^{k_1}$
Korak 8:	Ponovi i-krat (ali dokler ni N_1 ali N_2 enak '0'): (a) Priredi vrednosti: $N_1 = N_1 - 2^{k_1}$, $N_2 = N_2 - 2^{k_2}$ (b) Izračunaj $k_1 \leftarrow$ položaj vodilne enice N_1 (c) Izračunaj $k_2 \leftarrow$ položaj vodilne enice N_2 (d) Izračunaj $(N_1 - 2^{k_1})2^{k_2} \leftarrow$ pomakni $(N_1 - 2^{k_1})$ za k_2 bitov v levo (e) Izračunaj $(N_2 - 2^{k_2})2^{k_1} \leftarrow$ pomakni $(N_2 - 2^{k_2})$ za k_1 bitov v levo (f) Izračunaj $k_{12} = k_1 + k_2$ (g) Izračunaj $2^{(k_1+k_2)}$: dekodiraj k_{12} (h) Izračunaj $C^{(i)} = 2^{(k_1+k_2)} + (N_1 - 2^{k_1})2^{k_2} + (N_2 - 2^{k_2})2^{k_1}$
Korak 9:	Aproksimiraj $N_1N_2 = P_{SIM}^{(i)} = P_{SIM}^{(0)} + \sum_i C^{(i)}$

Spodnji primer prikazuje množenje dveh celih števil 234 in 198 s 3KČ.

Tabela 17: Primer množenja z SIM s 3KČ

Vhod$^{(0)}$:	$N_1 = 234 = (11101010)_2$, $N_2 = 198 = (11000110)_2$
Izračun $P_{SIM}^{(0)}$:	$k_1 = (0111)_2,\ N_1 - 2^{k_1} = (01101010)_2$ $k_2 = (0111)_2,\ N_2 - 2^{k_2} = (01000110)_2$ $k_1 + k_2 = (1110)_2$ $(N_1 - 2^{k_1})2^{k_2} = (011010100000000)_2$ $(N_2 - 2^{k_2})2^{k_1} = (010001100000000)_2$ $2^{(k_1+k_2)} = (100000000000000)_2$ $P_{SIM}^{(0)} = (1001100000000000)_2 = 38.912$ (Relativna napaka je 16,014%)
Vhod$^{(1)}$:	$N_1^{(1)} = (01101010)_2$, $N_2^{(1)} = (01000110)_2$
Izračun $C^{(1)}$ in $P_{SIM}^{(0)}$:	$k_1^{(1)} = (0110)_2,\ N_1^{(1)} - 2^{k_1^{(1)}} = (00101010)_2$ $k_2^{(1)} = (0110)_2,\ N_2^{(1)} - 2^{k_2^{(1)}} = (00000110)_2$ $k_1^{(1)} + k_2^{(1)} = (1100)_2$ $\left(N_1^{(1)} - 2^{k_1^{(1)}}\right)2^{k_2^{(1)}} = (101010000000)_2$ $\left(N_2^{(1)} - 2^{k_2^{(1)}}\right)2^{k_1^{(1)}} = (000110000000)_2$ $2^{(k_1^{(1)}+k_2^{(1)})} = (1000000000000)_2$ $C^{(1)} = (1110000000000)_2 = 7.168$ $P_{SIM}^{(1)} = 46.080$ (Relativna napaka je 0,543%)

Vhod$^{(2)}$:	$N_1^{(2)} = (00101010)_2$, $N_2^{(2)} = (00000110)_2$
Izračun $C^{(2)}$ in $P_{SIM}^{(2)}$:	$k_1^{(2)} = (0101)_2,\ N_1^{(2)} - 2^{k_1^{(2)}} = (00001010)_2$ $k_2^{(2)} = (0010)_2,\ N_2^{(2)} - 2^{k_2^{(2)}} = (00000010)_2$ $k_1^{(2)} + k_2^{(2)} = (0111)_2$ $\left(N_1^{(2)} - 2^{k_1^{(2)}}\right) 2^{k_2^{(2)}} = (0101000)_2$ $\left(N_2^{(2)} - 2^{k_2^{(2)}}\right) 2^{k_1^{(2)}} = (1000000)_2$ $2^{(k_1^{(2)} + k_2^{(2)})} = (10000000)_2$ $C^{(2)} = (11101000)_2 = 232$ $P_{SIM}^{(2)} = 46.312$ (Relativna napaka je 0,043%)
	$N_1^{(3)} = (00001010)_2$, $N_2^{(3)} = (00000010)_2$
Izračun $C^{(3)}$ in $P_{SIM}^{(3)}$:	$k_1^{(3)} = (0011)_2,\ N_1^{(3)} - 2^{k_1^{(3)}} = (00000010)_2$ $k_2^{(3)} = (0001)_2,\ N_2^{(3)} - 2^{k_2^{(3)}} = (00000000)_2$ $k_1^{(3)} + k_2^{(3)} = (0111)_2$ $\left(N_1^{(3)} - 2^{k_1^{(3)}}\right) 2^{k_2^{(3)}} = (100)_2$ $\left(N_2^{(3)} - 2^{k_2^{(3)}}\right) 2^{k_1^{(3)}} = (000)_2$ $2^{(k_1^{(3)} + k_2^{(3)})} = (10000)_2$ $C^{(3)} = (10100)_2 = 20$ $P_{SIM}^{(3)} = 46.332$ (Relativna napaka je 0,0%)
Točna vrednost:	$P_{true} = P_{SIM} = P_{SIM}^{(3)} = 46.332$

6.1. Analiza napake SIM

Relativna napaka po prvi aproksimaciji je:

$$E_r^{(0)} = \frac{P_{true} - P_{SIM}^{(0)}}{P_{true}} = \frac{E^{(0)}}{P_{true}}$$

$$= \frac{(N_1 - 2^{k_1})(N_2 - 2^{k_2})}{P_{true}} = \frac{2^{(k_1+k_2)} x_1 x_2}{2^{(k_1+k_2)}(1 + x_1 + x_2 + x_1 x_2)} \tag{30}$$

$$= \frac{x_1 x_2}{(1 + x_1 + x_2 + x_1 x_2)}$$

kjer je $0 \leq x_1, x_2 < 1$.

Relativna napaka po drugi aproksimaciji, to je po aproksimaciji s prvim KČ, je:

$$E_r^{(1)} = \frac{P_{true} - P_{SIM}^{(1)}}{P_{true}} = \frac{P_{true} - (P_{SIM}^{(0)} + C^{(1)})}{P_{true}} = \frac{E^{(1)}}{P_{true}} \tag{31}$$

Relativna napaka po i-ti aproksimaciji je:

$$E_r^{(i)} = \frac{E^{(i)}}{P_{true}} = \frac{\left(N_1^{(i)} - 2^{k_1^{(i)}}\right)\left(N_2^{(i)} - 2^{k_2^{(i)}}\right)}{P_{true}} \tag{32}$$

Za zmanjšanje relativne napake mora biti trenutna $i + 1$ iteracija nižja od absolutne napake prejšnje i-te aproksimacije:

$$E_r^{(i+1)} < E_r^{(i)} \tag{33}$$

Sledi dokaz, da to velja. Ker v vsaki iteraciji operandom odstranimo vodilno enico, velja:

$$k_1^{(i)} \leq k_1 - i,\ k_2^{(i)} \leq k_2 - i \tag{34}$$

Zapišemo lahko:

$$N_1^{(i)} = N_1 - 2^{k_1} - \sum_{j=1}^{i-1} 2^{k_1^{(j)}} \tag{35}$$

Isto velja za $N_2^{(i)}$:

$$N_2^{(i)} = N_2 - 2^{k_2} - \sum_{j=1}^{i-1} 2^{k_2^{(j)}} \tag{36}$$

Iz (34), (35) in (36) izpeljemo:

$$N_1^{(i+1)} - 2^{k_1^{i+1}} < N_1^{(i)} - 2^{k_1^i} \tag{37}$$

in:

$$N_2^{(i+1)} - 2^{k_2^{i+1}} < N_2^{(i)} - 2^{k_2^i} \tag{38}$$

Torej velja:

$$E_r^{(i+1)} < E_r^{(i)} \tag{39}$$

Če predpostavimo najslabši možni primer, ko so vsi biti v operandu '1', potem se število enic karakterističnega števila z vsako iteracijo zmanjša za 1. Velja naslednje:

$$k_1^i = k_1 - i, k_2^i = k_2 - i \tag{40}$$

Maksimalna relativna napaka i-te iteracije je:

$$\begin{aligned} E_{r\,max}^{(i)} &= \frac{2^{(k_1 - i + k_2 - i)} x_1^{(i)} x_2^{(i)}}{2^{(k_1+k_2)}(1 + x_1 + x_2 + x_1 x_2)} \\ &= 2^{-2i} \frac{x_1^{(i)} x_2^{(i)}}{(1 + x_1 + x_2 + x_1 x_2)} \end{aligned} \tag{41}$$

Očitno je, da se maksimalna relativna napaka zmanjšuje eksponentno s stopnjo najmanj 2^{-2i}, in doseže vrednost '0', ko je eden od operandov 0 ali pa ne vsebuje enic. Spodnja tabela prikazuje maksimalno relativno napako za različna števila KČ.

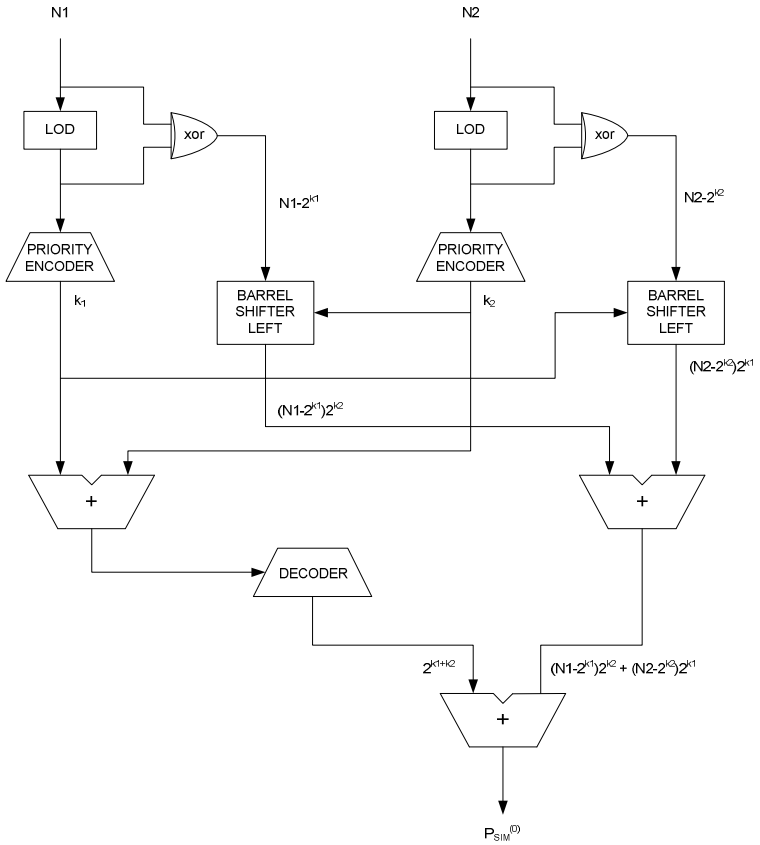
Tabela 18: Maksimalna relativna napaka za različna števila KČ

Št. KČ	Maksimalna relativna napaka* [%]
0	25,000
1	6,250
2	1,560
3	0,390
4	0,097
5	0,024

*dobljeno iz [4].

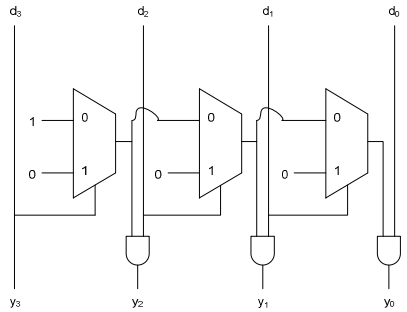
Opazimo lahko, da bodo vse relativne napake s samo 3KČ pod 0,5% in s 4KČ pod 0,1%.

6.2. Strojna implementacija algoritma



Slika 14: Arhitektura SIM množilnika

Zgornja slika predstavlja arhitekturo SIM množilnika brez korekcije (v nadaljevanju SIM). SIM izračuna eno aproksimacijo produkta glede na enačbo (23). Sestavljen je iz dveh detektorjev vodilnih enic (ang. Leading One Detector, LOD), dveh »xor« vrat, dveh kodirnikov, dveh 32-bitnih pomikalnikov, dveh 32-bitnih seštevalnikov, enega dekoderja in enega 64-bitnega seštevalnika. Implementacija LOD je prikazana na spodnji sliki.

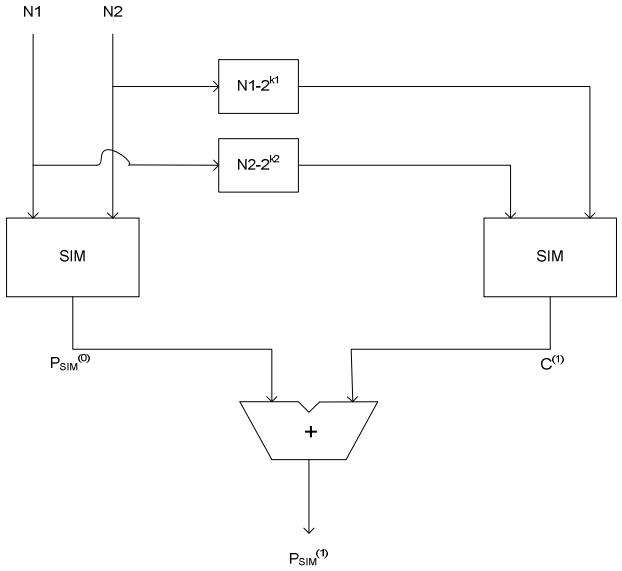


Slika 15: Arhitektura detektorja vodilne enice (LOD)

Enota LOD detektira vodilno enico operanda in jo pošlje naprej v pomikalnik. Poleg tega detektira tudi ničelne operande.

Pomikalnik se uporablja za pomik ostankov glede na enačbo (23). Dekodirna enota dekodira $k_1 + k_2$, to je postavi vodilno enico produkta.Vodilna enica in dva pomaknjena ostanka se seštejeta v aproksimiran produkt $P_{SIM}^{(0)}$. SIM je del implementacije korekcijskega vezja za izračun $P_{SIM}^{(i)}$ in $C^{(i)}$.

Spodnja slika prikazuje primer implementacije SIM s korekcijskim vezjem, ki ga uporabimo za izboljšanje točnosti množilnika. S korekcijskim vezjem izračunamo člen $C^{(1)}$ iz enačbe (28), ki je aproksimacija člena $(N_1 - 2^{k_1})(N_2 - 2^{k_2})$ iz enačbe (22). Spodnja slika prikazuje vezje z enim nivojem korekcije. Množilnik je sestavljen iz dveh SIM, kot je razvidno iz spodnje slike. Levi se uporablja za izračun aproksimacije produkta $P_{SIM}^{(0)}$ in desni za izračun KČ $C^{(1)}$.



Slika 16: Vezje za enonivojsko korekcijo SIM

Vsako korekcijsko vezje je implementirano kot SIM in se uporablja za izračun KČ po enačbi (29).

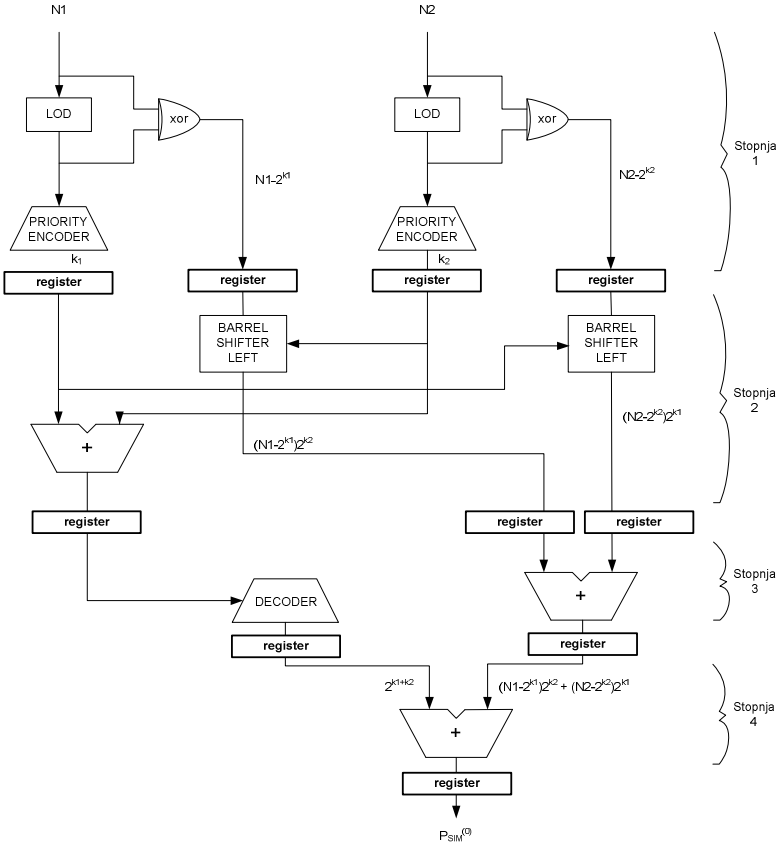
Implementacija 16-bitnega množilnika SIM vsebuje 2x 32-bitni seštevalnik, 3x 32-bitni logični pomikalnik in 2x 16-bitna dvo-vhodna »xor« vrata.

SIM+1KČ vsebuje 5x 32-bitni seštevalnik, 6x 32-bitni logični pomikalnik in 4x 16-bitna dvo-vhodna »xor« vrata.

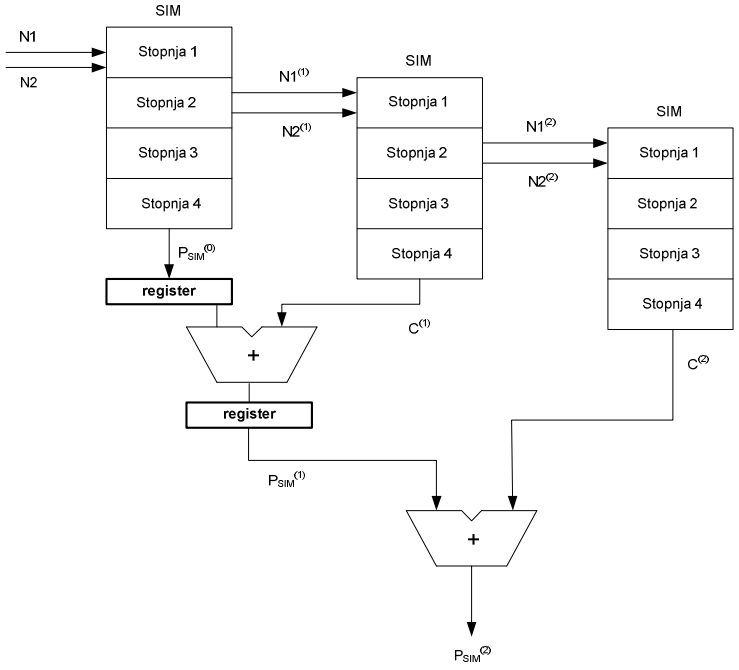
SIM+2KČ vsebuje 8x 32-bitni seštevalnik, 9x 32-bitni logični pomikalnik in 6x 16-bitna dvo-vhodna »xor« vrata.

SIM+3KČ vsebuje 11x 32-bitni seštevalnik, 12x 32-bitni logični pomikalnik in 8x 16-bitna dvo-vhodna »xor« vrata.

6.3. Implementacija SIM množilnika s cevovodom



Slika 17: Arhitektura cevovodnega SIM



Slika 18: Arhitektura cevovodnega SIM z dvema paralelnima KČ

Implementacija 16-bitnega množilnika SIM s cevovodom vsebuje 2x 32-bitni seštevalnik, 1x 5-bitni seštevalnik, 2x 32-bitni logični pomikalnik, 1x 1/32 dekoder, 2x 16-bitna dvo-vhodna »xor« vrata, 6x 32-bitni register, 4x 16-bitni register, 1x 5-bitni register in 2x 4-bitni register.

SIM+1KČ vsebuje 5x 32-bitni seštevalnik, 2x 5-bitni seštevalnik, 4x 32-bitni logični pomikalnik, 2x 1/32 dekoder, 4x 16-bitna dvo-vhodna »xor« vrata, 12x 32-bitni register, 6x 16-bitni register, 2x 5-bitni register in 4x 4-bitni register.

SIM+2KČ vsebuje 8x 32-bitni seštevalnik, 3x 5-bitni seštevalnik, 6x 32-bitni logični pomikalnik, 3x 1/32 dekoder, 6x 16-bitna dvo-vhodna »xor« vrata, 18x 32-bitni register, 8x 16-bitni register, 3x 5-bitni register in 6x 4-bitni register.

SIM+3KČ vsebuje 11x 32-bitni seštevalnik, 4x 5-bitni seštevalnik, 8x 32-bitni logični pomikalnik, 4x 1/32 dekoder, 8x 16-bitna dvo-vhodna »xor« vrata, 24x 32-bitni register, 10x 16-bitni register, 4x 5-bitni register in 8x 4-bitni register.

6.4. Izkoriščenost naprave

Tabela 19: Izkoriščenost naprave Xilinx Spartan-3 za SIM

Množilnik	Št. 4-vhodnih LUT tabel	Število rezin	Št. flip-flop rezin	Število V/I blokov
SIM	544	282	64	99
SIM+1KČ	1.099	564	74	99
SIM+2KČ	1.596	814	817	99
SIM+3KČ	1.937	993	1.108	99
Cevovodni SIM*	404	216	262	99
Cevovodni SIM+1KČ*	803	427	497	99
Cevovodni SIM+2KČ*	1.189	635	707	99
Cevovodni SIM+3KČ*	1.546	824	922	99

*4-stopenjski cevovod

V primerjavi z matričnim, MA in OD-MA množilnikom porabi SIM brez korekcije najmanj resursov (4-vhodnih LUT tabel in števila rezin). Vendar število resursov z vsakim dodanim korekcijskim členom narašča za pribiližno faktor 2. Kljub temu pa je SIM z 1 KČ manjši od OD-MA.

Tabela 20: Maksimalna frekvenca SIM na napravi Xilinx Spartan-3

Množilnik [n=16]	Maksimalna frekvenca [MHz]
SIM	58,435
SIM+1KČ	46,609
SIM+2KČ	39,953
SIM+3KČ	36,072
Cevovodni SIM*	126,933
Cevovodni SIM+1KČ*	126,933
Cevovodni SIM+2KČ*	126,933
Cevovodni SIM+3KČ*	126,933

*4 stopenjski cevovod

Maksimalna frekvenca množilnika SIM je v primerjavi z matričnim množilnikom, MA in OD-MA najvišja, in sicer za 16,664 MHz, 6,628 MHz in 18,753 MHz v tem vrstnem redu. S cevovodnim izvajanjem dobimo 3,52-kratno pohitritev v primerjavi z najvišjo frekvenco SIM+3KČ.

Poglavje 7

Rezultati meritev in simulacij

Za evalvacijo izkoriščenosti naprave in zmogljivosti v delu predstavljenih množilnikov sta bili uporabljeni napravi Xilinx Spartan-3 xc3s1500fg676-5 in Xilinx Virtex-6 Low Power xc6vlx75tl-1Lff784 FPGA [27]. Implementirani so bili naslednji 16, 32 in 64-bitni množilniki: matrični množilnik, množilnik zasnovan na Mitchellovem algoritmu (MA), množilnik zasnovan na Mitchellovem algoritmu z operandno dekompozicijo (OD-MA) ter enostavni iterativni množilnik (SIM). Pri SIM množilniku je bilo poleg osnovne strukture implementirano tudi vezje za korekcijo napake. Pri vseh so bile implementirane tudi cevovodne različice.

Načrtovanje množilnikov je potekalo v strojnem programskem jeziku VHDL z uporabo razvojne aplikacije Xilinx ISE 11.4 - WebPack [29]. Upoštevana so bila naslednja navodila načrtovanja za čim boljšo izkoriščenost arhitekture FPGA in čim višjo hitrost delovanja vezja:

- Čisto sinhronsko vezje
- Podvajanje logike na kritičnih poteh
- Cevovodno delovanje
- Uporaba dekodiranih avtomatov stanj
- Uporaba osnovnih gradnikov (seštevalnik, pomikalnik) za manjšo porabo moči
- Ročno popravljanje kritičnih vezij

Sinteza implementacije množilnikov na FPGA je bila opravljena s programskim orodjem Xilinx Xst Release 11.3 [31].

V naslednjih podpoglavjih sledi:

- Analiza napake
- Analiza hitrosti
- Analiza porabe vezja
- Analiza porabe moči

ki so bili glavni kriteriji primerjave množilnikov. Analiza napake je bila opravljena v programskem paketu Matlab [26]. Analiza hitrosti in porabe površine integriranega vezja je bila opravljena na podlagi sintez vezij z razvojno aplikacijo Xilinx ISE WebPack. Za analizo porabe moči je bil uporabljen programski paket Xilinx XPower Analyzer [28].

7.1. Analiza napake

Za evalvacijo povprečne in maksimalne relativne napake, so bili v Matlabu [26] pognani algoritmi množilnikov z vhodnimi podatki, ki so obsegali vse kombinacije n-bitnih pozitivnih števil. Povprečna relativna napaka je bila izračunana po enačbi:

$$E_R = \frac{1}{N}\sum_{i=1}^{N} E_r \tag{42}$$

kjer je N število množenj. Na primer, za 16-bitna števila je bil pognan algoritem nad vsemi kombinacijami števil od 1 do 65.535 in izračunana je bila povprečna relativna napaka. Za 12-bitna števila je bil pognan algoritem nad vsemi kombinacijami števil od 1 do 4.095 in izračunana je bila povprečna relativna napaka [4].

Naslednje tabele prikazujejo rezultate analize napake za algoritme aritmetično neeksaktnih množilnikov: MA, OD-MA, SIM brez korekcije in SIM z 1-3 korekcijskimi členi (KČ). Matrični množilnik je aritmetično eksakten, zato ni vključen v analizo napake.

Tabela 21: Povprečna relativna napaka E_R

Množilnik [n=16/12*]	**Povprečna napaka E_R ↑** [%]
OD-MA	3,528
MA	3,823
SIM*	9,412

*dobljeno iz [4].

Tabela 22: Povprečna relativna napaka E_R množilnika SIM s korekcijo [4]

Št. korekcijskih členov [n=16]	**Povprečna relativna napaka E_R ↓** [%]
0	9,412
1	0,987
2	0,107
3	0,012

Tabela 23: Maksimalna relativna napaka E_R ter porazdelitev relativne napake

Množilnik [n=16/12*]	**Maks. napaka E_R ↑ [%]**	$E_R <$ 0,1% [%]	$E_R <$ 0,5% [%]	$E_R <$ 1% [%]	$E_R <$ 5% [%]	$E_R <$ 10% [%]	$E_R <$ 20% [%]
SIM+3KČ*	0,390	98,000	**100,000**	**100,000**	n.d.	n.d.	n.d.
SIM+2KČ*	1,560	70,600	95,500	99,400	n.d.	n.d.	n.d.
SIM+1KČ*	6,250	19,300	47,400	65,200	n.d.	n.d.	n.d.
MA	11,113	99,091	99,091	99,232	99,687	99,955	**100,000**
OD-MA	11,113	**99,170**	99,170	99,484	**99,856**	**99,980**	**100,000**
SIM*	25,000	n.d.	n.d.	n.d.	n.d.	n.d.	n.d.

*dobljeno iz [4].

Rezultati analize napake prikazujejo, da ima OD-MA za približno 0,3% manjšo povprečno relativno napako E_R v primerjavi z MA. Največjo povprečno relativno napako ima SIM brez korekcije (tabela 21). S samo enim KČ k SIM relativno napako v povprečju opazno zmanjšamo pod 1%. To pomeni, da se SIM+1KČ v primerjavi z OD-MA, ki vsebuje 2 paralelna MA, v povprečju odreže bolje (tabela 22).

Maksimalna relativna napaka E_R je največja pri SIM brez korekcije. Maksimalna relativna napaka pri MA in OD-MA znaša 11,113%, glede na porazdelitev napake pa opazimo, da je napaka manjša pri OD-MA, vendar ne opazno veliko. SIM s KČ doseže že pri 1KČ manjšo maksimalno napako kot pri MA in OD-MA, vendar je vrednost porazdelitve napake pri 1KČ za $E_R < 0{,}1\%$ precej nižja od ostalih. SIM s 3KČ pa doseže točno vrednost že pri $E_R < 0{,}5\%$, kar je 4-krat bolje kot pri MA in OD-MA (tabela 23).

7.2. Analiza hitrosti

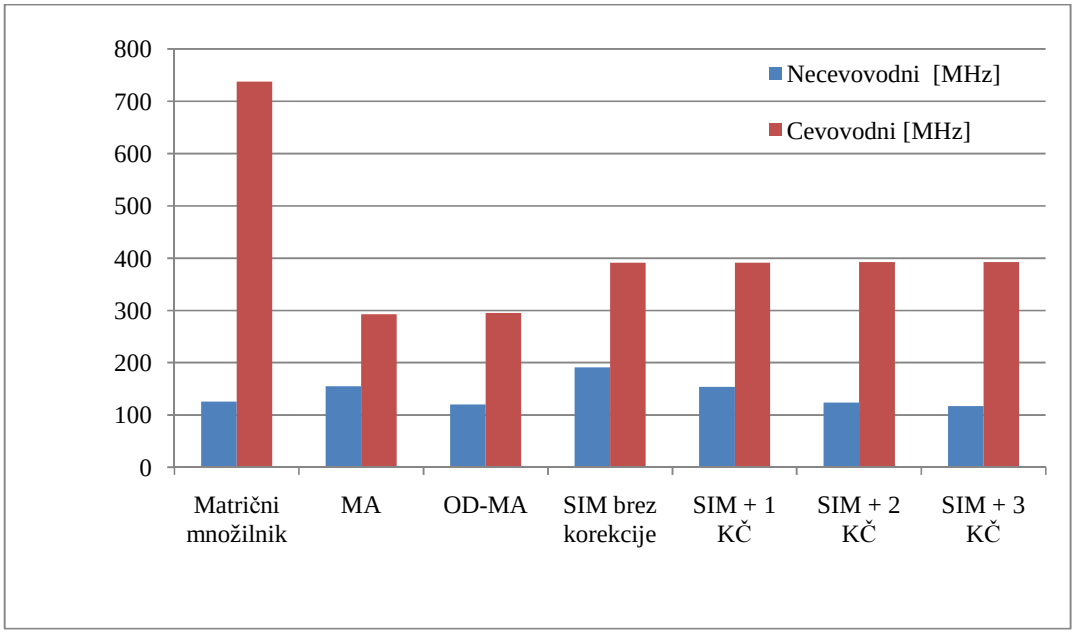
Za analizo hitrosti množilnikov se je uporabila maksimalna frekvenca, ki jo implementirani množilnik lahko doseže na FPGA integriranem vezju. Množilniki različnih dolžin n operandov s cevovodnim in necevovodnim izvajanjem, so bili implementirani na dveh FPGA integriranih vezjih, Xilinx Spartan-3 in Xilinx Virtex-6. Primerjava analize hitrosti med obema FPGA-jema zato ni enakovredna. Implementacija na dveh FPGA-jih služi le za potrditev koncepta.

Tabela 24: Maksimalna frekvenca in pohitritev za necev. in cev. množilnike na napravi Xilinx Spartan-3

Množilnik [n=16]	Necevovodni [MHz]	Cevovodni [MHz]	**Pohitritev ↓ [/]**
Matrični množilnik	41,771	**235,026**	5,62653516
SIM+3KČ	36,072	126,933	3,51887891
SIM+2KČ	39,953	126,933	3,17705804
SIM+1KČ	46,609	126,933	2,72335815
OD-MA	39,682	86,981	2,19195101
SIM	**58,435**	126,933	2,17220844
MA	51,807	86,79	1,67525624

Tabela 25: Maksimalna frekvenca in pohitritev za necev. in cev. množilnike na napravi Xilinx Virtex-6

Množilnik [n=16]	Necevovodni [MHz]	Cevovodni [MHz]	**Pohitritev ↓** [/]
Matrični množilnik	125,766	**737,463**	5,863
SIM+3KČ	117,495	392,619	3,342
SIM+2KČ	123,765	392,619	3,172
SIM+1KČ	153,856	391,389	2,544
OD-MA	120,151	295,072	2,456
SIM	**191,083**	391,389	2,048
MA	154,991	292,838	1,889

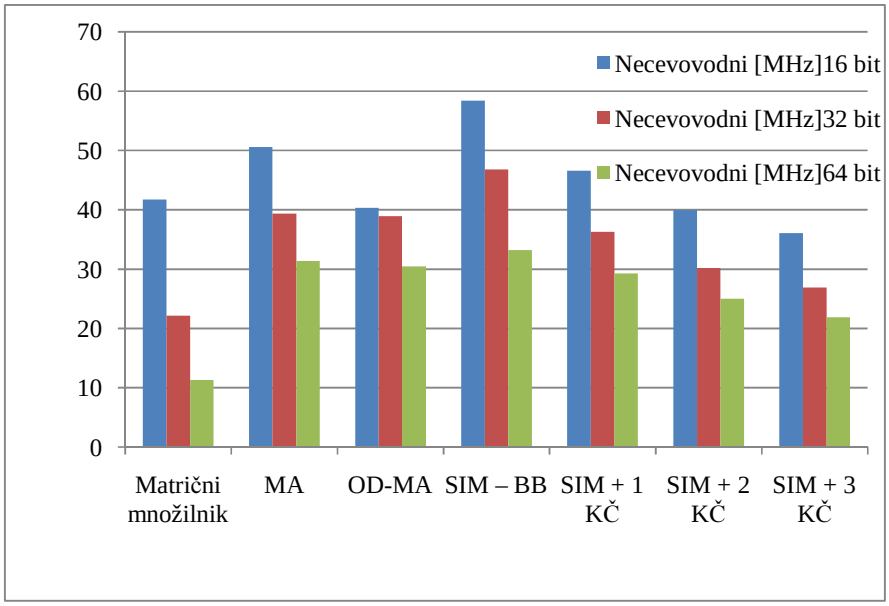


Graf 1: Primerjava maksimalne frekvence cevovod nih in necevovodnih množilnikov na Xilinx Spartan-3

Najvišjo maksimalno frekvenco necevovodnih množilnikov ima SIM brez korekcije, kar lahko pomeni tudi, da ima najmanj kompleksno strukturo. Z vsakim dodanim KČ se maksimalna frekvenca zniža, to pa zato, ker KČ ne morejo delati popolnoma paralelno, najprej je treba izračunati ostanke, s tem pa se povečuje pot skozi vezje. Kljub temu ima SIM z 1 KČ višjo frekvenco kot OD-MA. To velja tudi za SIM z 2KČ. Prav tako ima tudi MA višjo frekvenco kot OD-MA (tabela 24).

Z implementacijo cevovodnega izvajanja pri vseh množilnikih dosežemo pohitritev. Najbolj pa je pohitritev opazna pri matričnem množilniku. Matrični množilnik s cevovodom ima največjo maksimalno frekvenco (tabela 24, graf 1). Cevovodni matrični množilnik upravičuje porabo površine.

Če pogledamo pohitritve na Xilinx Virtex-6 FPGA-ju, so približno v sorazmerju s pohitritvijo na Xilinx Spartan-3. Takoj pa se vidi, kako doseže isti množilnik skoraj 3-kratno frekvenco samo zaradi implementacije na novejši tehnologiji FPGA-jev (tabela 25).



Graf 2: Primerjava maksimalne frekvence necevovodnih množilnikov glede na različno dolžino vhodnih operandov na Xil inx Spartan-3

Primerjava glede na različne dolžine operandov, tj. 16, 32 in 64-bitov nam pove, da z naraščanjem dolžine operanda maksimalna frekvenca pada (graf 2, podatki so zaradi preglednosti v prilogi).

7.3. Analiza porabe vezja

Za analizo porabe površine množilnikov se je uporabila izkoriščenost naprave. Izkoriščenost naprave pomeni porabo resursov na FPGA integriranem vezju. Množilniki različnih dolžin operandov n, s cevovodnim in necevovodnim izvajanjem, so bili implementirani na dveh FPGA integriranih vezjih Xilinx Spartan-3 in Xilinx Virtex-6. Pri Xilinx Spartan-3 opazujemo porabo naslednjih resursov: število 4-vhodnih LUT tabel, število rezin, število flip-flopov in število V/I blokov. Pri Xilinx Virtex-6 pa zaradi drugačne strukture integriranega vezja FPGA opazujemo porabo naslednjih resursov: število LUT rezin, število registrskih rezin, število LUT in flip-flop parov ter število V/I blokov. Primerjava porabe resursov med obema FPGA-jema ni enakovredna. Implementacija na dveh FPGA-jih služi le za potrditev koncepta.

Primerljivi množilniki so:

- matrični množilnik, MA, (OD-MA) in SIM brez korekcije (osnovna metoda)
- (OD-MA) in SIM z 1KČ (osnovna metoda s korekcijo enega nivoja)
- SIM brez korekcije, SIM + 1KČ, SIM + 2 KČ, SIM + 3 KČ (naraščanje korekcije)

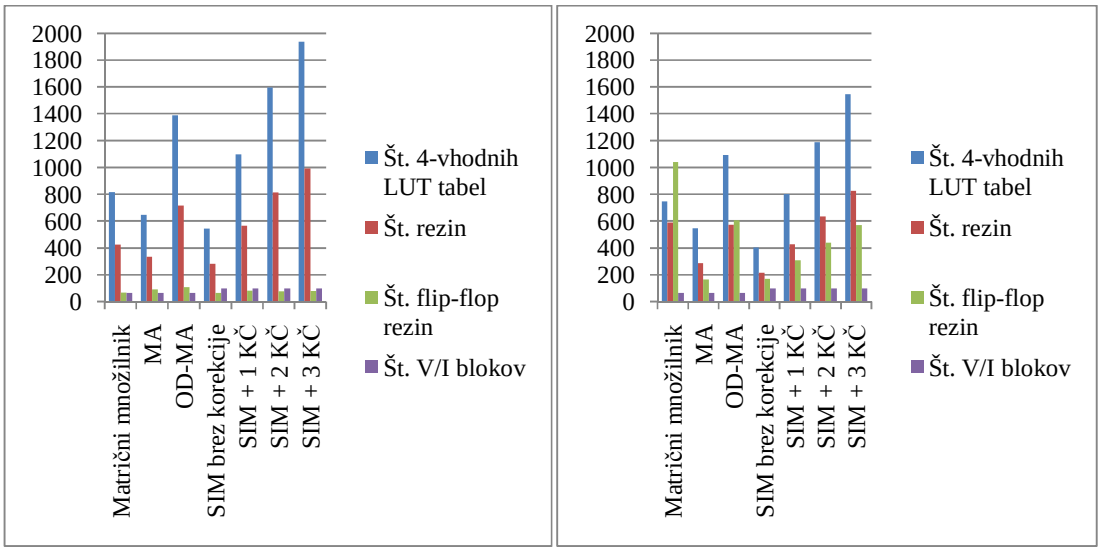
Opazimo lahko, da zahteva aritmetično eksakten matrični množilnik več resursov (upoštevamo predvsem število 4-vhodnih LUT tabel) kot MA in SIM brez korekcije. Lahko rečemo, da je poraba resursov eksaktnega matričnega množilnika primerljiva neeksaktnima množilnikoma OD-MA in SIM+1KČ (tabela 26).

Tabela 26: Izkoriščenost naprave Xilinx Spartan-3 za necevovodne množilnike

Množilnik [n=16]	**Št. 4-vhodnih LUT tabel ↑**	Št. rezin	Št. flip-flop rezin	Št. V/I blokov
SIM	544	**282**	**64**	99
MA	598	308	80	**66**
Matrični množilnik	816	424	67	**66**
SIM+1KČ	1.099	564	74	99
OD-MA	1.337	688	91	**66**
SIM+2KČ	1.596	814	817	99
SIM+3KČ	1.937	993	1.108	99

Tabela 27: Izkoriščenost naprave Xilinx Spartan-3 za cevovodne množilnike

Cevovodni množilnik [n=16]	**Št. 4-vhodnih LUT tabel ↑**	Št. rezin	Št. flip-flop rezin	Št. V/I blokov
SIM	404	**216**	262	99
MA	554	290	**168**	**66**
Matrični množilnik	746	589	1.039	**66**
SIM+1KČ	803	427	497	99
OD-MA	1.112	593	362	**66**
SIM+2KČ	1.189	635	707	99
SIM+3KČ	1.546	824	992	99



Graf 3: Poraba resursov necevevodnih množilnikov (levo) in poraba resursov cevovodnih množilnikov (desno) na Xilinx Spartan-3

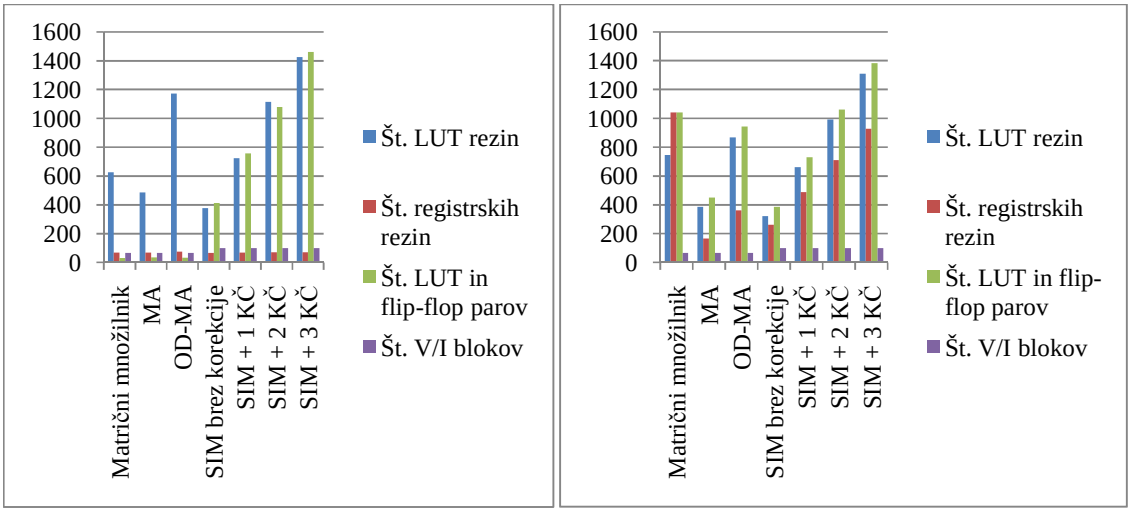
Pri primerjavi porabe resursov necevovodnih in cevovodnih množilnikov je največja razlika opazna pri povečanem številu flip-flop-ov na račun dodanih registrov za cevovodno izvajanje. Sprememba je največja pri matričnemu množilniku, ki ima od ostalih množilnikov najbolj globok cevovod (tabela 27, graf 3).

Tabela 28: Izkoriščenost naprave Xilinx Virtex-6 za necevovodne množilnike

Množilnik [n=16]	**Št. LUT rezin ↑**	Št. registrskih rezin	Št. LUT in flip-flop parov	Št. V/I blokov
SIM	377	**66**	411	99
MA	484	69	36	**66**
Matrični množilnik	627	68	**31**	**66**
SIM+1KČ	723	67	756	99
SIM+2KČ	1.115	72	1.080	99
OD-MA	1.171	76	32	**66**
SIM+3KČ	1.427	72	1.462	99

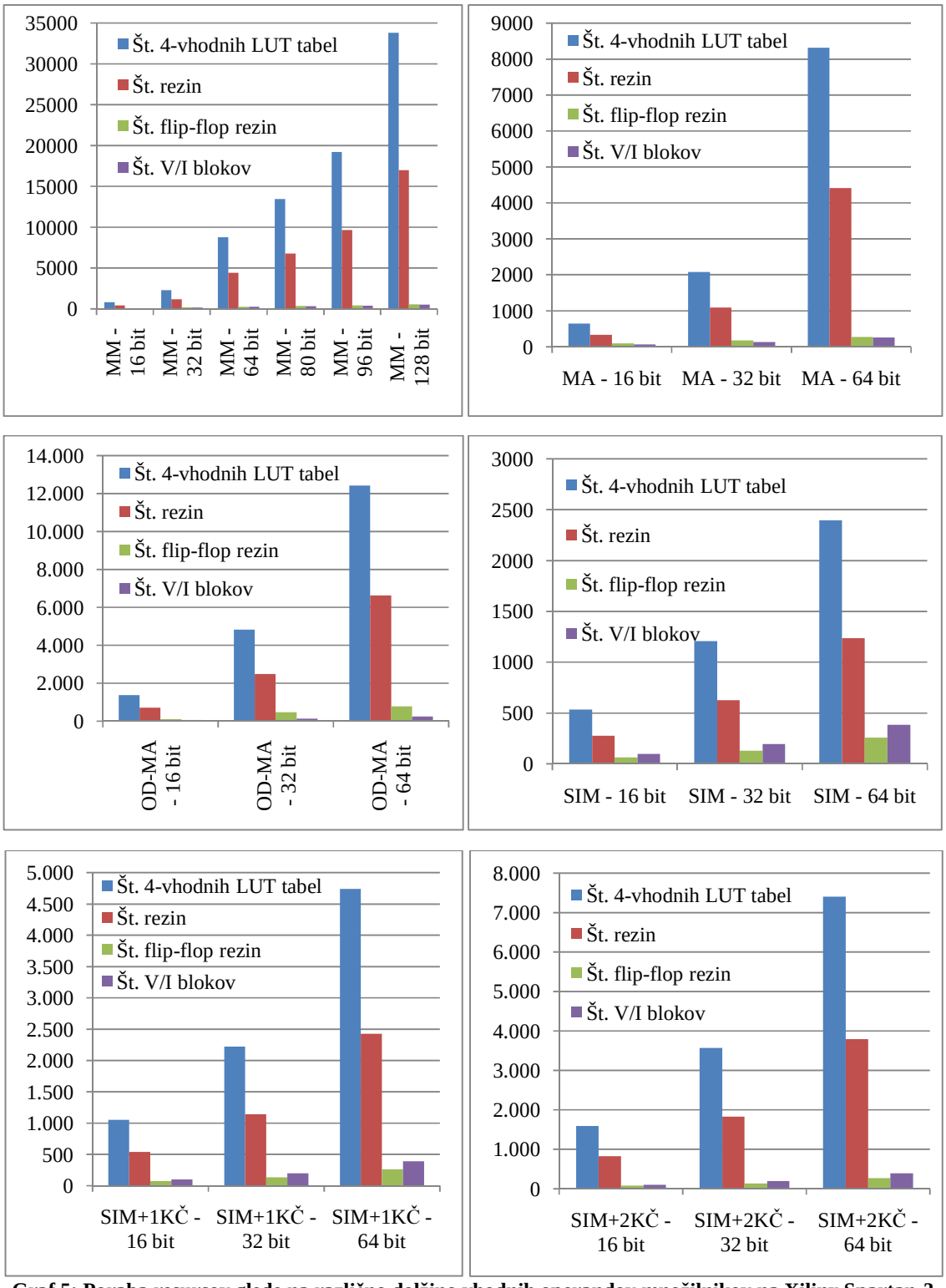
Tabela 29: Izkoriščenost naprave Xilinx Virtex-6 za cevovodne množilnike

Cevovodni množilnik [n=16]	**Št. LUT rezin ↑**	Št. registrskih rezin	Št. LUT in flip-flop parov	Št. V/I blokov
SIM	321	262	**386**	99
MA	387	**165**	451	**66**
SIM+1KČ	662	489	730	99
Matrični množilnik	746	1.039	1.039	**66**
OD-MA	868	362	943	**66**
SIM+2 KČ	991	710	1.061	99
SIM+3 KČ	1.309	927	1.382	99



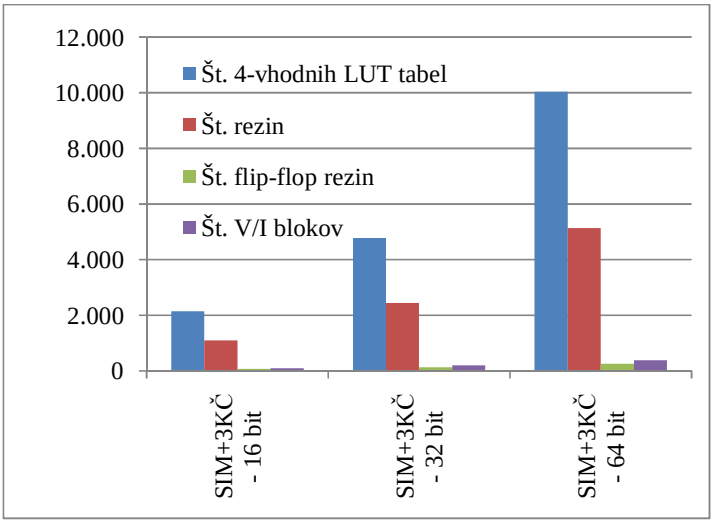
Graf 4: Poraba resursov necevevodnih množilnikov (levo) in poraba resursov cevovodnih množilnikov (desno) na Xilinx Virtex-6

Prikazani so rezultati porabe resursov tudi za Xilinx Virtex-6 FPGA (tabeli 28 in 29). Razmerja porabe resursov množilnikov so podobna kot pri Xilinx Spartan-3 FPGA.



Graf 5: Poraba resursov glede na različno dolžino vhodnih operandov množilnikov na Xilinx Spartan-3

(se nadaljuje)



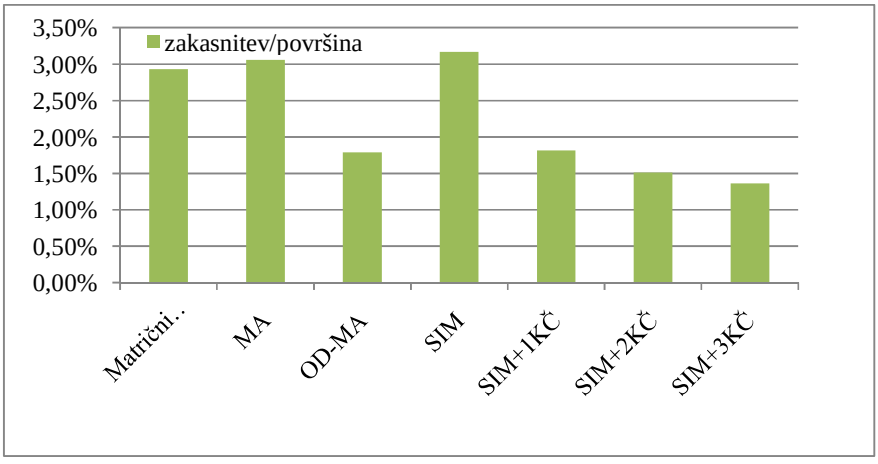
Graf 5: Poraba resursov glede na različno dolžino vhodnih operandov množilnikov na Xilinx Spartan-3

(nadaljevanje s prejšnje strani)

Zanimiva je primerjava porabe resursov glede na velikost operandov, to je 16, 32 in 64-bitov. Pri matričnemu množilniku poraba resursov z dolžino operandov precej narašča (nelinearno strmo). Implementacija 64-bitnega matričnega množilnika porabi 32% razpoložljivih LUT rezin, 64-bitni MA 31%, 64-bitni OD-MA 46%, 64-bitni SIM 8%, 64-bitni SIM+1KČ 17%, 64-bitni SIM+2KČ 27% in 64-bitni SIM+3KČ 37% razpoložljivih LUT rezin.

Razmerje velikosti operandov in porabe resursov pri MA in OD-MA je približno faktor 2. Prav tako je videti linearno razmerje porabe resursov z vsakim dodanim KČ glede na SIM brez korekcije (graf 5, podatki so zaradi preglednosti navedeni v prilogi).

Razmerje zakasnitev (najdaljša kombinacijska pot vezja) in površina (število 4-vhodnih LUT tabel) je zanimivo. Pomeni kako kompleksen je implementiran množilnik. Najmanjšo zakasnitev in hkrati najmanjšo porabo površine vezja ima SIM brez korekcije, sledi mu MA (graf 6, podatki so zaradi preglednosti v prilogi).



Graf 6: Primerjava razmerja zakasnitev/površina necevovodnih množilnikov na Xilinx Spartan-3

Najmanjšo porabo resursov ima SIM brez korekcije.

7.4. Analiza porabe moči

Več moči kot integrirano vezje porabi, na višji temperaturi deluje in se izvaja počasneje. Zanesljivost naprave se manjša z delovanjem na visokih temperaturah. Ker je število vrat ključna determinanta porabe moči obstajajo omejitve porabe moči, ki vplivajo na količino logike, ki jo lahko integriramo v napravo. Zmanjševanje porabe moči tudi zmanjšuje vpliv elektromagnetnega polja (ang. ElectroMagnetic Interference, EMI) in šum.

Po določitvi maksimalne dovoljene ambitentne temperature, se z naslednjo enačbo [30] izračuna maksimalno dovoljeno porabo moči naprave P_{max}:

$$P_{max} = \frac{T_J T_A}{\Theta_{\mathrm{JA}}} \tag{43}$$

kjer je T_J maksimalna dovoljena temperatura stika silicija izražena v enoti °C, T_A dovoljena ambientna temperatura izražena v enoti °C in Θ_{JA} termalna upornost stik-ambient naprave izražena v enoti °C/W.

Statična moč (ang. static ali quiescent power) predstavlja moč, ki jo naprava porabi, ko je prižgana in sprogramirana ter ne obstaja nobenega preklapljanja. To vključuje puščanje tranzistorjev (ang transistor leakage), interno porabo moči in porabo moči zunanje bremenitve.

Statična poraba moči se glasi:

$$P_s = VI = \frac{V^2}{R} \tag{44}$$

Dinamična moč (ang. dynamic power) se porablja, ko je uporabniška logika aktivna. Dinamično porabo moči lahko zapišemo z enačbo [30]:

$$P_D = VI = V\left(C\frac{dV}{dt}\right) = CV^2 f \tag{45}$$

Zadnji dve enačbi veljata za interno porabo moči in naraščajočo moč, ki jo povzročajo I/O vmesniki (ang. buffer) in povezave med napravami. V obeh primerih se moč spreminja s kvadratom delovne napetosti. Sprememba iz 5V na 3,3V teoretično zmanjša moč za skoraj 60%. Prav tako je pomembna frekvenca preklapljanja.

FPGA tehnologija je zasnovana na CMOS SRAM, zato v statičnem stanju generira malo toka. Dinamično porabo moči primarno določa frekvenca ure, na kateri delujejo sinhronska vezja. Na primer, 20MHz ura povzroča 40 milijonov prehodov na sekundo. Na lastnosti pomembno vpliva tudi parazitna kapacitivnost vezij.

Kapacitivnost je odvisna od karakteristik povezav (širina, dolžina in impendanca) in števila porazdeljenih bremen (flip-flopi, logična vrata in povezave med njimi). Dolge povezave imajo praviloma višjo kapacitivnost kot kratke. Pri zahtevah po daljših povezavah je potrebno povezave zaključiti z pasovnimi tranzistorji ali varovalnimi povezavami.

7.4.1. Rezultati simulacije porabe moči

Zmožnost načrtovanja in ocenjevanja porabe moči sistema ima velik vpliv na načrtovanje logičnih vezij, izbiro ustreznih elektronskih komponent in tiskanih vezij in zanesljivost sistema.

Xilinxovo orodje Xilinx XPower Analayzer [28], ki je bilo v tem delu uporabljeno za ocenjevanje in primerjavo porabe moči množilnikov, analizira logična vezja množilnikov in z upoštevanjem podatkovnega modela FPGA naprave oceni porazdelitev moči logičnega vezja množilnikov. Pri ocenjevanju so bili uporabljeni naslednji parametri:

Tabela 30: Paramteri za ocenjevanje porabe moči množilnikov na napravi Xilinx Spartan-3

Parameter	Vrednost
V_{CCINT}	1,2 V
V_{CCAUX}	2,5 V
V_{CCO}	2,5 V
T_A	25,0°C
Hitrost preklapljanja (ang. toggle rate)	12,5
	Ostali nastavljivi parametri imajo privzeto vrednost.

Opaziti je, da se večino moči porabi zaradi statične moči na V/I (velja tako za Xilinx Spartan-3 kot Xilinx Virtex-6). Ta se porablja ob mirovnem delovanju FPGA-ja. Moč, ki jo porabijo logika in signali (dinamična moč) predstavlja porabo moči zaradi delovanja implementiranega množilnika, in je primerna za realno primerjavo porabe moči (tabele 31-34).

Najmanjšo porabo dinamične moči ima SIM brez korekcije. Z dodanimi korekcijskimi členi se skupna moč ne povečuje. Moč, ki jo porabijo logika in signali se skoraj podvoji z vsakim dodanim korekcijskim členom, a še vedno to predstavlja majhen odstotek glede na skupno moč. Podobno se poraba dinamične moči OD-MA približno podvoji glede na MA in ima največjo vrednost od vseh množilnikov (tabele 31-34).

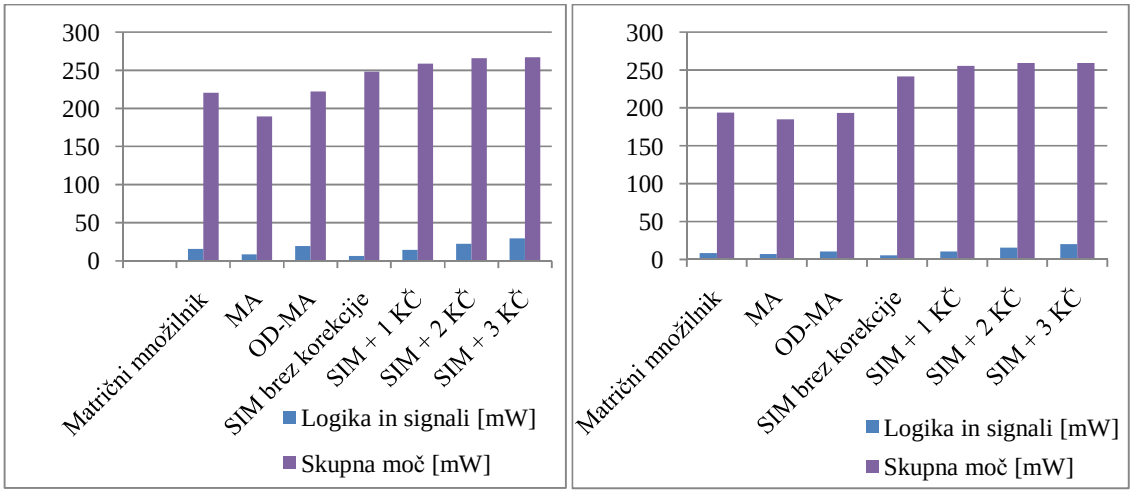
Ker nas pri porabi moči zanima predvsem dinamična moč, sta v grafih 8 in 9 izpuščena parametra V/I bloki ter statična moč in ura.

Tabela 31: Ocena porabe moči pri frekvenci 40MHz za necev. množilnike na napravi Xilinx Spartan-3

Množilnik [n=16]	**Logika in signali** ↑ [mW]	V/I bloki [mW]	Statična moč in ura [mW]	Skupna moč [mW]
SIM	6,76	88,77	152,39	247,93
MA	8,48	**29,27**	**151,78**	**189,52**
SIM+1KČ	14,35	92,07	152,38	258,81
Matrični množilnik	15,46	52,49	152,30	220,25
OD-MA	19,57	50,95	152,02	222,54
SIM+2KČ	22,25	91,02	152,58	265,85
SIM+3KČ	29,36	84,97	152,59	266,92

Tabela 32: Ocena porabe moči pri frekvenci 40MHz za cev. množilnike na napravi Xilinx Spartan-3

Cevovodni množilnik [n=16]	**Logika in signali** ↑ [mW]	V/I bloki [mW]	Statična moč in ura [mW]	Skupna moč [mW]
SIM	5,46	83,47	152,99	241,93
MA	7,15	**25,77**	**151,98**	**184,89**
Matrični množilnik	8,36	31,22	154,18	193,76
SIM+1KČ	10,77	91,47	153,65	255,88
OD-MA	10,95	29,19	153,26	193,41
SIM+2KČ	15,69	89,08	154,49	259,25
SIM+3KČ	20,26	84,26	154,92	259,44



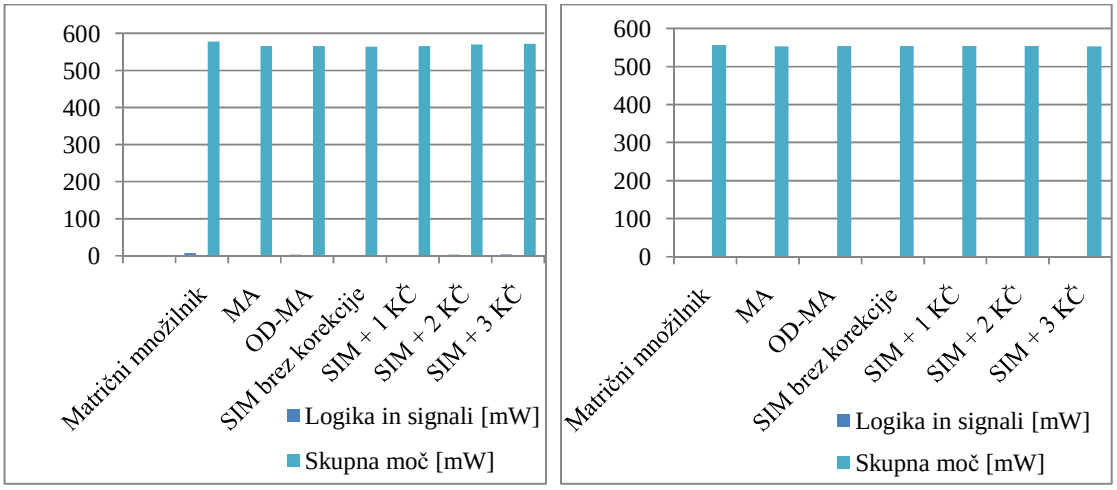
Graf 7: Ocena porabe moči pri frekvenci 40MHz za necevovodne (levo) in cevovodne (desno) množilnike na napravi Xilinx Spartan-3

Tabela 33: Ocena porabe moči pri frekvenci 40MHz za necev. množilnike na napravi Xilinx Virtex-6

Množilnik [n=16]	**Logika in signali** ↑ [mW]	V/I bloki [mW]	Statična moč in ura [mW]	Skupna moč [mW]
SIM	0,73	13,37	**549,87**	**563,98**
MA	1,61	13,68	550,56	565,85
SIM+1KČ	1,68	14,89	549,93	566,5
SIM+2KČ	2,94	16,75	550,23	569,92
OD-MA	3,37	**12,46**	550,53	566,37
SIM+3KČ	4,26	17,27	550,25	571,78
Matrični množilnik	7,41	19,82	550,26	577,49

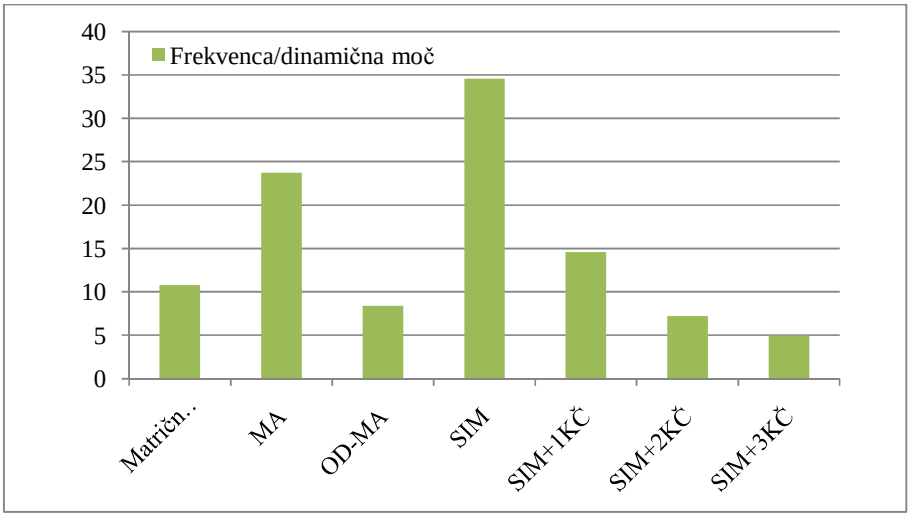
Tabela 34: Ocena porabe moči pri frekvenci 40MHz za cev. množilnike na napravi Xilinx Virtex-6

Cevovodni množilnik [n=16]	**Logika in signali** ↑ [mW]	V/I bloki [mW]	Statična moč in ura [mW]	Skupna moč [mW]
MA	0,27	2,18	550,46	**552,91**
SIM	0,28	3,98	**550,40**	554,65
SIM+1KČ	0,39	2,36	551,14	553,89
SIM+2KČ	0,48	1,51	551,98	553,96
OD-MA	0,51	2,37	551,60	554,48
SIM+3KČ	0,52	**1,12**	552,15	553,8
Matrični množilnik	0,58	2,49	553,17	556,25



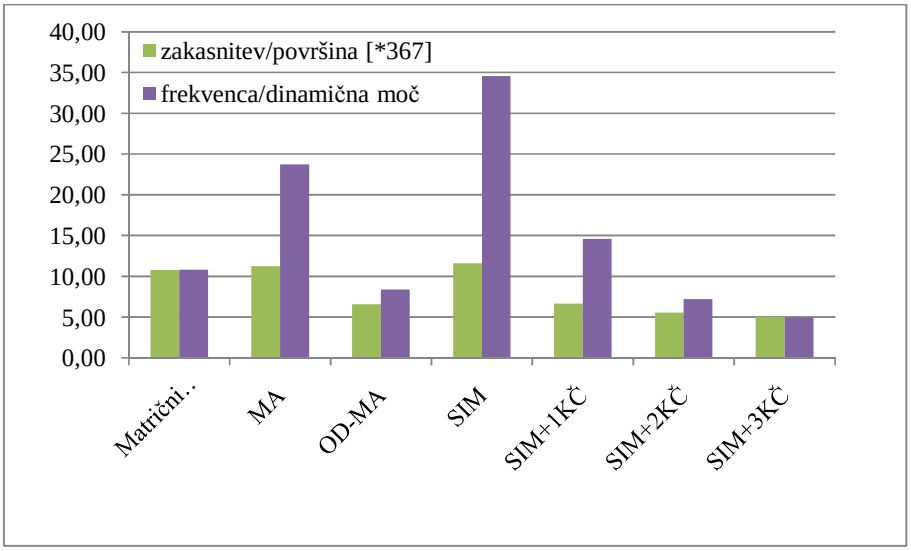
Graf 8: Ocena porabe moči pri frekvenci 40MHz za necev. (levo) in cev. (desno) množilnike na napravi Xilinx Virtex-6

Poraba moči množilnikov implementiranih na Xilinx Virtex-6 je manjša v primerjavi z implementacijo na Xilinx Spartan-3, tudi tukaj na račun boljše tehnologije FPGA. Glede na razmerja porabe moči množilnikov na Xilinx Spartan-3 opazimo, da matrični množilnik na Xilinx Virtex-6 nima isto srednje ampak najslabšo oceno.



Graf 9: Razmerje maksimalne frekvence in porabljene dinamične moči necev. množilnikov na napravi Xilinx Spartan-3

Najboljše razmerje maksimalna frekvenca/dinamična moč ima SIM brez korekcije in sicer ima največjo maksimalno frekvenco in najmanjšo porabo dinamične moči. Sledi mu MA (graf 10).



Graf 10: Razmerji zakasnitev/prostor ter frekvenca/dinamična moč necevovodnih množilnikov na napravi Xilinx Spartan-3

Razmerji zakasnitev/površina (normalizirano) ter frekvenca/dinamična moč sta še enkrat prikazani skupaj v grafu 11. Najmanjšo zakasnitev in hkrati maksimalno frekvenco, najmanjšo porabo površine vezja in najmanjšo porabo dinamične moči ima SIM brez korekcije.

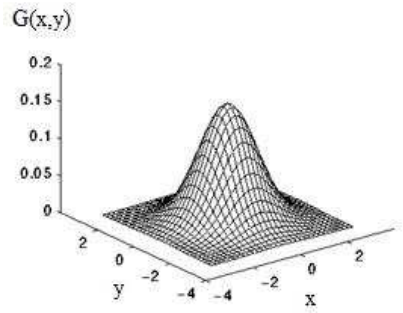
7.5. Primerjava množilnikov z eksperimentalnimi rezultati

Zanima nas kako in v kakšni meri napaka množilnikov vpliva na rezultat konkretne aplikacije. Z algoritmi množilnikov izvajamo enostavno aplikacijo konvolucije glajenje slike (odstranjevanje šuma). Vhod aplikacije je slika, ki jo preberemo in z množilnim algoritmom zmnožimo vsako točko slike (ang. pixel) s konvolucijsko matriko (ali masko) za glajenje.

Glajenje izvedemo z Gaussovo funkcijo (ali porazdelitev):

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{46}$$

ki je krožno simetrična funkcija (slika 19).



Slika 19: Gaussova porazdelitev s srednjo vrednostjo v točki (0,0) in standardno deviacijo $\sigma = 1$

Ideja glajenja je vpliv ene točke razširiti na sosednje. To dosežemo s konvolucijo slike z Gaussovo funkcijo. Slika je množica diskretnih točk, zato rabimo za konvolucijo diskretno aproksimacijo Gaussove porazdelitve (glej Konvolucijska matrika 1 in 2). Rezultat množenja z Gaussovo aproksimacijsko matriko je uteženo povprečje soseščine točke, kjer ima vrednost centralne točke največjo vtež. Slika postane zamegljena (odstranjevanje šuma) in večja kot je standardna deviacija σ Gaussove aproksimacije, večji je efekt.

Tabela 35: Primer konvolucijskega algoritma

```
rgb = imread('lena', 'bmp');
for i=1:1:size(rgb,1)
    for j=1:1:size(rgb,2)
        for a=-2:1:2
            for b=-2:1:2
                q1=rgb(i+a,j+b);
                q2=mask(a+3,b+3);
                sum=sum+KONV(q1,q2);
            end
        end
        rgc(i,j)=sum/N;
    end
end
```

Zgornja tabela prikazuje primer konvolucijskega algoritma. Sliko preberemo z ukazom *imread* [26]. Premikamo se skozi vse točke slike (črnobela slika je dvodimenzionalna) in v vsaki točki računamo produkt slike s konvolucijsko matriko. Funkcija **KONV(q1,q2)**

predstavlja množenje števil q1 in q2 z aritmetično neeksaktnimi množilnimi algoritmi: MA, OD-MA, SIM, SIM+1KČ, SIM+2KČ, SIM+3KČ. Za oceno napake jih primerjamo z rezultatom kovolucijskega algoritma, ki uporablja navadno množenje (točna vrednost) v Matlabu [26].

7.5.1. Filtriranje šuma (glajenje) slike z Gaussovo masko

Uporabljeni sta bili dve konvolucijski matriki: Gaussova aproksimacija z $\sigma = 1$ in $\sigma = 1{,}4$. Originalna slika je črnobela, formata bmp in rezultat konvolucije z različnimi množilnimi algoritmi je prikazan na slikah 20 in 21. Napaka konvolucije je podana v tabelah 36 in 37.

$$\frac{1}{273}\begin{vmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{vmatrix}$$

Konvolucijska matrika 1: Diskretna Gaussova aproksimacija z $\sigma = 1$

V spodnji tabeli je prikazana napaka konvolucijskega algoritma za glajenje slike, ki uporablja neeksaktne množilne algoritme in Gaussovo aproksimacijo z $\sigma = 1$ (Konvolucijska matrika 1). Maksimalna napaka predstavlja največji odmik od točne vrednosti točk in povprečna napaka je povprečje napak v vseh točkah.

Tabela 36: Maksimalna in povprečna napaka množilnih algoritmov ($\sigma = 1$)

Množilni algoritem $[\sigma_G = 1]$	**Maksimalna napaka ↓** [%]	Povprečna napaka [%]
SIM	9,02	2,77
OD-MA	4,31	0,98
MA	3,92	1,57
SIM+1KČ	0,78	0,17
SIM+2KČ	0	0
SIM+3KČ	0	0

Največjo maksimalno napako ima množilni algoritem SIM, sledi mu OD-MA, ki pa ima v primerjavi z MA v povprečju manjšo napako. Že SIM z 1 KČ doseže napako manjšo od 1%, brez napake sta pa SIM+2KČ in SIM+3KČ.

$$\frac{1}{115}\begin{vmatrix} 2 & 5 & 4 & 5 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 5 & 4 & 5 & 2 \end{vmatrix}$$

Konvolucijska matrika 2: Diskretna Gaussova aproksimacija z $\sigma = 1{,}4$

V spodnji tabeli je prikazan rezultat konvolucijskega algoritma za glajenje slike, ki uporablja neeksaktne množilne algoritme in Gaussovo aproksimacijo z $\sigma = 1{,}4$ (Konvolucijska matrika 2). Maksimalna napaka predstavlja največji odmik od točne vrednosti točk in povprečna napaka je povprečje napak v vseh točkah.

Tabela 37: Maksimalna in povprečna napaka množilnih algoritmov ($\sigma = 1,4$)

Množilni algoritem [$\sigma_G = 1,4$]	**Maksimalna napaka ↓** [%]	Povprečna napaka [%]
SIM	9,02	2,83
OD-MA	5,49	1,04
MA	5,49	1,81
SIM+1KČ	0,39	0,07
SIM+2KČ	0,39	0,07
SIM+3KČ	0	0

Največjo maksimalno napako ima množilni algoritem SIM, sledita mu OD-MA in MA, vendar ima OD-MA v primerjavi z MA v povprečju manjšo napako. Že SIM z 1 KČ in 2 KČ dosežeta napako manjšo od 1%, brez napake pa je SIM+3KČ.

Spodnji sliki 20 in 21 prikazujeta rezultat konvolucije originalne slike v formatu bmp s konvolucijsko matriko 1 (slika 20) in konvolucijsko matriko 2 (slika 21). Slike si sledijo v naslednjem zaporedju: originalna slika, rezultat konvolucije z navadnim množenjem v Matlabu, rezultat konvolucije z množilnimi algoritmi MA, OD-MA, SIM, SIM+1KČ, SIM+2KČ in SIM+3KČ. Z rdečim robom je označena slika, ki vsebuje tudi vidno največjo napako množilnega algoritma. V obeh primerih gre za SIM, zdi se, da je neprimeren za aplikacijo glajenja slike.

Slika 20: Rezultat konvolucije originalne slike s konvolucijsko matriko 1 (Original, Matlab, MA, OD-MA, SIM, SIM+1KČ, SIM+2KČ, SIM+3KČ)

Slika 21: Rezultat konvolucije originalne slike s konvolucijsko matriko 2 (Original, Matlab, MA, OD-MA, SIM, SIM+1KČ, SIM+2KČ, SIM+3KČ)

7.5.2. Primerjava napake množilnih algoritmov z napako jpg konverzije

Konverzija slike iz slikovnega formata bmp (ang. bitmap) v slikovni format jpeg (ang. Joint Photographic Experts Group, jpg) je kompresijska metoda, ki odstranjuje detajle slike, ki jih človeško oko običajno ne zazna. Torej ima slika formata jpg napako v primerjavi s sliko formata bmp. Zanimalo nas je kako velika je napaka konverzije slike (d-0). Zanimalo nas je tudi, kakšna je razlika napake (d-1), če glajenje slike izvedemo pred konverzijo (konvolucijski algoritem 1) ali pa po konverziji slike v jpg (konvolucijski algoritem 2). Podana je tudi napaka glajenja slike v formatu jpg zaradi uporabe neeksaktnih množilnih algoritmov (e-1). Za oceno napake uporabimo konvolucijo z navadnim množenjem v Matlabu (točna vrednost m_true). Cilj je bil ugotoviti, ali je velikost napake konverzije primerljiva z napako zaradi neeksaktnega množenja.

Tabela 38: Primer izračuna razlik in napake glajenja slike glede na konverzijo iz bmp v jpg format (v Matlabu)

Konvolucijski algoritem 1:	i1 = imread('lena.bmp', 'bmp');
	i2 = KONV(i1,mask);
	i3 = imwrite(i2, 'lena.jpg', 'jpg');

Konvolucijski algoritem 2:	j1 = imread('lena.bmp', 'bmp');
	j2 = imwrite(j1, 'lena.jpg', 'jpg');
	j3 = KONV(j2,mask);

d-0:	max(abs(j2-i1)) *% Maksimalna napaka konverzije.*
d-1:	max(abs(j3-i3)) *% Maksimalna razlika napak glajenja slik z množilnimi algoritmi pred ali po konverziji v format jpg.*
e-1:	max(abs(m_true-j3)) *% Maksimalna razlika napake množilnih algoritmov.*

V zgornji tabeli je podan primer izračuna razlik in napake glajenja slike glede na konverzijo slike v jpg format, kjer je **KONV(i,mask)** množenje števil i in mask z aritmetično neeksaktnimi množilnimi algoritmi: MA, OD-MA, SIM, SIM+1KČ, SIM+2KČ, SIM+3KČ. Rezultati so podani v tabeli 39 (za $\sigma_G = 1$) in v tabeli 40 ($\sigma_G = 1{,}4$).

Maksimalna razlika konverzije ni odvisna od množilnega algoritma in ima vrednost:

$$d\_0 = \max\big(\text{abs}(\text{j2} - \text{i1})\big) = \mathbf{6,67}\%, \tag{47}$$

ki je večja od maksimalne napake neeksaktnih množilnikov (če zanemarimo SIM):

Tabela 39: Razlika in napaka množilnih algoritmov glede na konverzijo iz bmp v jpg format ($\sigma = 1$)

Množilni algoritem $[\sigma_G = 1]$	d-1 max(abs(j3-i3)) [%]	**e-1↓** max(abs(m_true-j3)) [%]
SIM	5,88	9,02
OD-MA	6,27	**4,31**
MA	6,27	**3,92**
SIM+1KČ	7,06	**0,78**
SIM+2KČ	6,67	**0,00**
SIM+3KČ	6,67	**0,00**

Razlika napake, če glajenje slike izvedemo pred konverzijo ali po konverziji slike v jpg, je pri SIM+2KČ in SIM+3KČ enaka maksimalni razliki konverzije d-0. To pa zato, ker je njuna napaka zaradi uporabe neeksaktnih množilnih algoritmov enaka '0'. Ostali imajo približno isti odmik od vrednosti d-0. Napaka zaradi uporabe neeksaktnih množilnih algoritmov je največja pri SIM, sledi mu OD-MA.

Tudi za vrednost $\sigma = 1{,}4$ je maksimalna razlika konverzije večja od maksimalne napake neeksaktnih množilnikov (če zanemarimo SIM):

Tabela 40: Razlika in napaka množilnih algoritmov glede na konverzijo iz bmp v jpg format ($\sigma = \mathbf{1,4}$)

Množilni algoritem $[\sigma_G = 1{,}4]$	d-1 max(abs(j3-i3)) [%]	**e-1↓** max(abs(m_true-j3)) [%]
SIM	9,80	9,41
MA	9,80	**5,49**
OD-MA	10,20	**5,10**
SIM+1KČ	10,20	**0,78**
SIM+2KČ	7,84	**0,39**
SIM+3KČ	6,67	**0,00**

Napaka množilnih algoritmov je po vrednosti primerljiva z napako kompresije dane slike v format jpg. Kar pomeni, da je toleranca kompresije slik v format jpg pri uporabi aplikacije konvolucije glajenje slik po vrednosti primerljiva z napako neeksaktnih množilnih algoritmov (če zanemarimo SIM).

Zaključek

Analizirane so bile znane in novejše metode za učinkovito množenje. Za vsako prestavljeno metodo je bila podana strojna implementacija na Xilinx Spartan-3 in Virtex-6 Low Power FPGA. Kriteriji primerjave so bili: napaka, hitrost, poraba površine integriranega vezja in poraba moči.

Ugotovljeno je bilo, da množilnik SIM generira največjo relativno napako, vendar pa doseže na račun manjše kompleksnosti vezja največjo maksimalno frekvenco in ima najmanjšo porabo dinamične moči. Zato tudi doseže najboljši rezultat razmerij maksimalna frekvenca/dinamična moč in zakasnitev/površina. Z vsakim dodanim KČ, ki zmanjšuje njegovo napako, se linearno povečuje površina vezja, prav tako se linearno povečuje poraba dinamične moči in zmanjšuje maksimalna frekvenca, ker ne morejo delati popolnoma paralelno.

OD-MA predstavlja izboljšavo točnosti MA, a ne bistveno, njegova poraba resursov pa je približno dvakratna. SIM+1KČ je v tem primeru boljša izbira, saj ima manjšo napako in porabo resursov.

Z naraščanjem dolžine operanda, implementirane so bile 16, 31 in 64-bitne različice, maksimalna frekvenca pada. Narašča pa poraba površine vezja, najbolj strmo pri matričnem množilniku (približno kvadratično), pri ostalih približno linearno.

Aritmetično eksakten matrični množilnik porabi več resursov kot MA in SIM. Njegova poraba je po velikosti primerljiva z množilnikoma OD-MA in SIM+1KČ.

V primeru aplikacije, ki tolerira napako do 10% je SIM najboljša izbira. Vsi parametri primerjave razen napake namreč dosežejo boljše vrednosti od ostalih množilnikov. V primeru aplikacije, ki tolerira napako do 1% se izkaže cevovodni SIM že z 1 ali 2 KČ boljša izbira kot MA in OD-MA.

V primeru aplikacije glajenje slike je bilo ugotovljeno, da ima SIM največjo napako, ki je v primerjavi z ostalimi tudi vizuelno opazna in se zato zdi neprimerna za to aplikacijo. V primeru kompresije slike iz bmp formata v format jpg je bilo ugotovljeno, da je napaka množilnih algoritmov po vrednosti primerljiva z napako kompresije (če zanemarimo SIM), kar pomeni, da je kompresija slik tolerantna na napako v tem delu omenjenih neeksaktnih množilnih algoritmov.

Priloge

Dodatek A: Rezultati simulacij

Tabela 41: Izkoriščenost naprave Xilinx Virtex-6, vlx75tl-1Lff784 za necevovodne različice različnih dolžin operandov

Množilnik	Št. 4-vhodnih LUT tabel	Št. rezin	Št. flip-flop rezin	Št. V/I blokov
Matrični množilnik - 16 bit	816	424	67	66
Matrični množilnik - 32 bit	2.270	1.159	163	130
Matrični množilnik - 64 bit	8.754	4.424	281	258
Matrični množilnik - 80 bit	13.421	6.762	350	322
Matrični množilnik - 96 bit	19.182	9.663	422	386
MA - 16 bit	647	335	92	66
MA - 32 bit	2.083	1.087	180	130
MA - 64 bit	8.317	4.414	268	258
OD-MA - 16 bit	1.387	714	108	66
OD-MA - 32 bit	4.821	2.491	475	130
OD-MA - 64 bit	12.427	6.619	786	258
SIM - 16 bit	533	276	64	99
SIM - 32 bit	1.209	625	130	195
SIM - 64 bit	2.394	1.236	257	387
SIM+1KČ - 16 bit	1.049	541	74	99
SIM+1KČ - 32 bit	2.221	1.141	130	195
SIM+1KČ - 64 bit	4.742	2.429	261	387
SIM+2KČ - 16 bit	1.590	817	73	99
SIM+2KČ - 32 bit	3.564	1.825	132	195
SIM+2KČ - 64 bit	7.399	3.785	262	387
SIM+3KČ - 16 bit	2.146	1.108	76	99
SIM+3KČ - 32 bit	4.778	2.444	130	195
SIM+3KČ - 64 bit	10.035	5.135	262	387

Tabela 42: Maksimalna frekvenca 16-bitnih necevovodnih množilnikov na Xilinx Spartan-3

Množilnik [n=16]	Maksimalna frekvenca [MHz]
Matrični množilnik	41,771
MA	50,554
OD-MA	40,335
SIM	58,435
SIM+1KČ	46,609
SIM+2KČ	39,953
SIM+3KČ	36,072

Tabela 43: Maksimalna frekvenca 32-bitnih necevovodnih množilnikov na Xilinx Spartan-3

Množilnik [n=32]	Maksimalna frekvenca [MHz]
Matrični množilnik	22,183
MA	39,367
OD-MA	38,955
SIM	46,815
SIM+1KČ	36,271
SIM+2KČ	30,177
SIM+3KČ	26,909

Tabela 44: Maksimalna frekvenca 64-bitnih necevovodnih množilnikov na Xilinx Spartan-3

Množilnik [n=64]	Maksimalna frekvenca [MHz]
Matrični množilnik	11,336
MA	31,424
OD-MA	30,423
SIM	33,243
SIM+1KČ	29,256
SIM+2KČ	25,025
SIM+3KČ	21,876

Tabela 45: Razmerje zakasnitev/površina necovovodnih množilnikov na Xilinx Spartan-3

Množilnik	Zakasnitev [ns]	Površina [št. vh. 4-LUT tabel]	Zakasnitev/površina
Matrični množilnik	23,940	816	2,93%
MA	19,780	647	3,06%
OD-MA	24,795	1.387	1,79%
SIM	17,219	544	3,17%
SIM+1KČ	19,928	1.099	1,81%
SIM+2KČ	24,137	1.596	1,51%
SIM+3KČ	26,436	1.937	1,36%

Tabela 46: Razmerje frekvenca/dinamična moč necevovodnih množilnikov na Xilinx Spartan-3

Množilnik [n=16]	Maksimalna frekvenca [MHz]	Logika in signali (dinamična moč) [mW]	Frekvenca/dinamična moč [MHz/mW]
Matrični množilnik	41,771	3,87	10,79
MA	50,554	2,13	23,73
OD-MA	40,330	4,81	8,38
SIM	58,435	1,69	34,58
SIM + 1 KČ	46,609	3,20	14,56
SIM + 2 KČ	39,953	5,56	7,19
SIM + 3 KČ	36,072	7,29	4,95

Literatura

[1] K. H. Abed, R. E. Sifred, »VLSI Implementation of a Low-Power Antilogarithmic Converter«, *IEEE Transactions on Computers*, št. 52, zv. 9, str. 1221-1228, sept. 2003.

[2] K. H. Abed, R. E. Sifred, »CMOS VLSI Implementation of a Low-Power Logarithmic Converter«, *IEEE Transactions on Computers*, št. 52, zv. 11, str. 1421-1433, nov. 2003.

[3] P. J. Ashenden, »The designer's guide to Vhdl 2nd edition«, *Elsevier Inc*, 2008.

[4] Z. Babić, A. Avramović, P. Bulić, »An Iterative Logarithmic Multiplier«, *Microprocessors and Microsystems*, doi: 10.1016/j.micpro.2010.07.001, 2010.

[5] T. A. Brubaker, J. C. Becker, »Multiplication Using Logarithms Implemented with Read-Only Memory«, *IEEE Transactions on Computers*, št. c-24, zv. 8, avg 1975.

[6] G. M. Chaudhary, F. Kharbash, »High Performance Fast Multiplier«, *Region 5 Conference, 2008 IEEE*, str. 1-4, 2008.

[7] T. Fryza, »Introduction to fixed-point multiplication and signal processing application«, *Radioelektronika'09, 19th International Conference*, str. 283-286, 2009.

[8] E. L. Hall, D. D. Lynch, S. J. Dwyer III, »Generation of Products and Quotients Using Approximate Binary Logarithms for Digital Filtering Applications«, *IEEE Transactions on Computers*, št. C-19, zv. 2, str. 97-105, feb. 1970.

[9] F. Haohuan, O. Mencer, W. Luk, »Optimizing residue arithmetic on FPGAs«, *International Conference on ICECE Technology*, str. 41-48, 2008.

[10] J. L. Hennessey, D. A. Patterson, »Computer Architecture: A Quantitative Approach, 4th ed.«, *Morgan Kauffman Pub.*, pogl. I.2-I.4, I.9, 2007.

[11] M. Ito, D. Chinnery, K. Keutzer, »Low Power Multiplication Algorithm for Switching Activity Reduction through Operand Decomposition«, *Proc. IEEE Int'l Conf. Computer Design (ICCD)*, 2003.

[12] D. Kodek, »Arhitektura in organizacija računalniških sistemov«, *Bi-Tim*, 2008.

[13] M. Y. Kong, J. M. P. Langlois, D. Al-Khalili, »Efficient FPGA implementation of complex multipliers using the logarithmic number system«, *IEEE International Symposium on Circuits and System*, str. 3154-3157, jun. 2008.

[14] D. K. Kostopoulus, »An Algorithm for the Computation of Binary Logarithms«, *IEEE Transactions on Computers*, št. 40, zv. 11, nov. 1991.

[15] A. De, P. P. Kurur, C. Saha, »Fast Integer Multiplication Using Modular Arithmeric«, *Dept. of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur, UP, India*, 2008.

[16] R. G. Lyons, »Understanding digital signal processing, 2nd edition«, *Prentice Hall*, 2004.

[17] V. Mahalingam, N. Rangantathan, »Improving Accuracy in Mitchells Logaritmic Multiplication Using Operand Decomposition«, *IEEE Transactions on Computers*, št. 55, zv. 2, str. 1523-1535, dec. 2006.

[18] D. J. McLaren, »Improved Mitchell-Based Logarithmic Multiplier for Low-power DSP Applications«, *Proceedings of IEEE International SOC Conference 2003*, str.53-56 in 17-20, sept. 2003.

[19] J. N. Mitchell, »Computer Multiplication and Division Using Binary Logarithms«, *IRE Transactions on Electronic Computers*, št. EC-11, str. 512-517, avg. 1962.

[20] N. Nedjah, »A Review of Modular Multiplication Methods and Respective Hardware Implementations«, *Department of Electronics Engineering and Telecommunications, Engineering Faculty, State University od Rio de Janeiro, Rio de Janeiro, Brazil, Informatica*, 2006.

[21] M. E. Paramasivam, R. S. Sabeenian, »An efficient bit reduction binary multiplication algorithm using vedic methods«, *Advance Computing Conference (IACC)*, str. 25-28, 2010.

[22] B. Parhami, »Computer Arithmetic: Algorithms and Hardware Designs«, *Oxford University Press*, 2001.

[23] N. Petra, D. De Caro, V. Garofalo, E. Napoli, A. G. M. Strollo, »Truncated Binary Multipliers With Variable Correction and Minimum Mean Square Error«, *IEEE Transactions on Circuits and systems*, 2009.

[24] M. H. Rais, »Efficient Hardware Realization of Truncated Multipliers using FPGA«, *International Journal of Applied Science*, zv. 5, št. 2, str. 124-128, 2009.

[25] (2007) Multiplication in FPGAs. Dostopno na: http://www.andraka.com/multipli.htm.

[26] (2010) The MathWorks Matlab documentation. Dostopno na: http://www.mathworks.com/access/helpdesk/help/techdoc/.

[27] (2010) Xilinx Inc. Spartan-3 FPGA Data Sheets. Dostopno na: http://www.xilinx.com/support/documentation/spartan-3_data_sheets.htm.

[28] (2010) Xilinx Inc. Xilinx XPower. Dostopno na: http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm.

[29] (2010) Xilinx ISE WebPACK Design Software. Dostopno na: http://www.xilinx.com/tools/webpack.htm.

[30] (2010) Xilinx XPower Estimator User Guide. Dostopno na: http://www.xilinx.com/support/documentation/user_guides/ug440.pdf

[31] (2010) Xilinx XST User Guide. Dostopno na http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf.

Izjava o samostojnosti dela

Izjavljam, da sem magistrsko delo izdelala samostojno pod vodstvom mentorja doc. dr. Patricia Bulića. Izkazano pomoč drugih sodelavcev sem v celoti navedla v zahvali.

Marjana Erdelji