Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

David Lapajne

Cevovodni procesor HIP v vezju FPGA z okoljem za razhroščevanje

diplomsko delo na univerzitetnem študiju

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Implementirajte cevovodno različico procesorja HIP v vezju FPGA. Procesorju dodajte še enoto za seštevanje in odštevanje v plavajoči vejici ter dva direktna predpomnilnika za ukaze in operande. Poleg procesorja implementirajte še razhroščevalno enoto s pomočjo katere bo mogoče slediti izvajanju ukazov v cevovodu, spremljati stanje v pomnilniku ter predpomnilnikih in komunicirati z razhroščevalnim okoljem na osebnem računalniku. Razhroščevalno okolje implementirajte v jeziku Java.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani David Lapajne sem avtor diplomskega dela z naslovom:

Cevovodni procesor HIP v vezju FPGA z okoljem za razhroščevanje

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 28. avgusta 2016 Podpis avtorja:

Zahvaljujem se rejnikoma Renati in Lojzetu za vso pomoč pri izobraževanju. Zahvala gre tudi Tajdi za podporo pri izdelavi te diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško
ANTLR	ANother Tool for Language Recognition
CPU	Central Processing Unit
FP	Floating Point
FPGA	Field-Programmable Gate Array
HIP	HIpotetični procesor
PROM	Programmable Read Only Memory
RAM	Random Access Memory
ROM	Read Only Memory
RISC	Reduced Instruction Set Computer
UART	Universal Asynchronous Receiver Transmitter
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit

kratica	**slovensko**
CPE	Centralna Procesna Enota
PV	Plavajoča vejica
FPGA	vrsta programabilnega vezja
HIP	izmišljena centralna procesna enota
PROM	Programabilni bralni pomnilnik
RAM	Pomnilnik z naključnim dostopom
ROM	Bralni pomnilnik
RISC	procesor z manjšim številom ukazov
UART	Univerzali asihronski sprejemnik in oddajnik
VHDL	vrsta jezika HDL
VHSIC	visoko hitrostno integrirano vezje

Povzetek

Diplomska naloga obsega implementacijo cevovodne centralne procesne enote imenovane hipotetični procesor (HIP), narejen po principih opisanih v knjigi [1]. Vsebuje logiko za premoščanje, seštevalnik števil zapisanih v plavajoči vejici. Dodan ima ukazni in podatkovni predpomnilnik. Preko razhroščevalne enote je mogoče brati vse splošnonamenske in vmesne registre v cevovodu HIP ter tako spremljati izvajanje poteka prevedenega programa. HIP se izvaja na FPGA čipu na razvojni ploščici Spartan 3E, kjer se nahaja podporno vezje za nadzor in komunikacijo. Zunanji program, napisan v Javi, je možno pognati na različnih operacijskih sistemih. V razhroščevalniku je možno pisati program v zbirnem jeziku. Vsebuje prevajalnik iz zbirnega v strojni jezik. Strojno kodo poganja tako, da v razhroščevalno enoto v HIP pošilja ukaze in podatke, ki so potrebni za izvajanje. Pri izvajanju se vsako urino periodo prebere vsebina registrov v CPE. Vidna je tudi vsebina glavnega pomnilnika in predpomnilnika.

Ključne besede: HIP, ANTLR, VHDL, zbirni jezik, prevajalnik, razčlenjevalnik, razhroščevalnik, FPGA.

Abstract

This thesis describes the implementation of a central processing unit with pipeline called Hypothetical processor (HIP), which is described in book [1]. It contains logic for data forwarding, an adder for floating point numbers and it has an instruction and data cache. Through the debug unit it is possible to read from and write to all general and to other registers in the HIP pipeline and therefore monitor the flow of the compiled program. HIP runs in the FPGA chip on the Spartan 3E development board where supporting logic for monitoring is present. The external program written in Java runs on different operating systems. The monitoring program contains a text editor where it is possible to write in the assembler language. It also contains a compiler which translates an assembler code to HIP machine code. Operations and data are sent to the debug unit to HIP. Each clock cycle, the monitoring program reads the content of every register in the CPU. The content of the main memory and cache is seen too.

Keywords: HIP, ANTLR, VHDL, assembler, compiler, parser, debugger, FPGA.

Poglavje 1

Uvod

Hipotetični računalnik HIP je model centralne procesne enote, ustvarjen za namene študija. Na modelu lahko preučujemo probleme, ki se pojavijo pri načrtovanju centralnih procesnih enot in rešitve, s katerimi rešujemo prepreke do hitrejšega in učinkovitejšega delovanja centralne procesne enote. Model HIP in vso dodatno vezje je mogoče napisati v programskem jeziku VHDL. S pomočjo razvojnih ploščic s FPGA čipom in ostalimi vhodno/izhodnimi enotami je s programskimi orodji mogoče prevesti celoten model procesorja HIP, predpomnilnik in ostalo podporno logiko v obliko, ki ga je mogoče simulirati na tem FPGA čipu. Med delovanjem je v sam FPGA čip nemogoče imeti neposreden vpogled v signale in posledično v registre. Zato ima HIP model dodano še nadzorno enoto in UART vmesnik. Nadzorna enota je razhroščevalna enota, ki nadzoruje delovanje procesorja HIP, pošilja podatke in sprejema ukaze zunanjega programa preko UART vmesnika. Zunanji program je razhroščevalnik z grafičnim vmesnikom, ki ga je mogoče poganjati na različnih operacijskih sistemih. Omogoča lažjo uporabo in nudi možnost branja in pisanja v registre, ki so prisotni v procesorju HIP in v predpomnilnik. Program vsebuje prevajalnik, ki prevede ukaze iz zbirnega jezika v strojni jezik procesorja HIP. Z razhroščevalnikom pa jih nato izvaja preko nadzorne enote. Po vsaki urini periodi je mogoče videti vsebino registrov in tako preverjati delovanje

procesorja HIP in prevedenega programa.

Poglavje 2

Strojni nivo

2.1 Programski jezik in orodja uporabljena na strojnem nivoju

V diplomski nalogi je uporabljen programski jezik VHDL (VHSIC HDL - Very High Speed Integrated Circuit Hardware Description Language). Jezik omogoča z enostavnimi programskimi izrazi opisovati posamezne logične strukture. Od kombinatoričnih struktur, kot so kodirniki, dekodirniki, izbiralniki (multiplekserji) in drugih logičnih funkcij, do sekvenčnih struktur, kot so D pomnilne celice, ROM, RAM in avtomatov.
Namesto resničnostne tabele 2.1, skrajšane tabele 2.2 ali Boolove enačbe (2.1), lahko za opis funkcije multiplekserja uporabimo izraz 2.2.

$$\begin{aligned} f(a,b,s) &= \overline{a}bs + a\overline{b}\overline{s} + ab\overline{s} + abs \\ &= \overline{s}a + sb \end{aligned} \tag{2.1}$$

```
izhod <= a when s = '0' else b;
```
(2.2)

Jezik VHDL omogoča hitrejšo izdelavo modelov, saj ima opis ločen od izvajanja. Odpravljanje napak in testiranje pravilnosti opisa je hitrejše in

a	b	s	f(a,b,s)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Tabela 2.1: Resničnostna tabela multiplekserja z dvema vhodoma a, b in izbiralnim vhodom s.

s	f(a,b,s)
0	a
1	b

Tabela 2.2: Tabela izhodnih vrednosti multiplekserja z dvema vhodoma a, b in izbirnim vhodom s.

lažje. Ko modeliramo vezje z mnogimi vhodi in izhodi, je izbira jezika VHDL primerna.

2.1.1 Razvojna orodja Xilinx Design Tools

Orodje ISE Project Navigator vsebuje grafični vmesnik, ki omogoča lažje delo pri modeliranju logičnih vezij. Omogoča lažji dostop do podpornih orodij, kot so ISim HDL Simulator, izvaja zakasnitvene analize na različnih nivojih, tvori poročila, sporoča napake in opozorila. Opis orodij, funkcij in njihova uporaba je opisana v navodilih [2]. Program ISE Project Navigator posamezne logične enote, zaradi lažjega pregleda nad modeliranim vezjem, prikazuje hierarhično. Orodje lahko naredi sintezo posameznih logičnih enot in preveri pravilnost zapisa datotek zapisanih v jeziku VHDL. Tvori programabilne datoteke za posamezne čipe, predvsem FPGA (field-programmable gate array), na katerih v realnosti preverimo delovanje opisanega modela. FPGA je integrirano vezje, ki ga razvijalec lahko programira po lastnih zahtevah in željah. Preverjanje pravilnosti delovanja opisanega vezja, je s čipom FPGA bistveno hitrejše, kot traja izdelava opisanega integriranega vezja na siliciju, na katerem se preveri pravilnost delovanja.

Podjetje Xilinx ponuja različne razvojne ploščice na katerih je nameščeno integrirano vezje FPGA. Na razvojnih ploščicah se poleg čipa FPGA nahajajo dodatna integrirana vezja, ki so s čipom FPGA povezana preko nogic. Za razvojno ploščico Spartan3E je podroben opis zunanjih naprav in povezav opisan v uporabniškem priročniku [3]. V priročniku je v poglavju 7. opisan primer kako povezati pine na DB9 DCE serijskem konektorju s signali na FPGA čipu. DCE serijski konektor na razvojni ploščici je povezan z UART modulom na FPGA čipu. Tako lahko razhroščevalna enota na FPGA čipu komunicira z napravami, ki se nahajajo izven razvojne ploščice.

2.2 Hipotetični računalnik HIP

Cevovodni procesor HIP služi kot model za študij centralne procesne enote. Centralna procesna enota je logično vezje, ki izvaja strojne ukaze ustvarjene zanj. Je registrsko-registrski računalnik vrste RISC (Reduced Instruction Set Computer) in ima relativno malo ukazov, kar pomeni enostavnejši zapis njegovega modela v jeziku VHDL. Značilnosti procesorja HIP so:

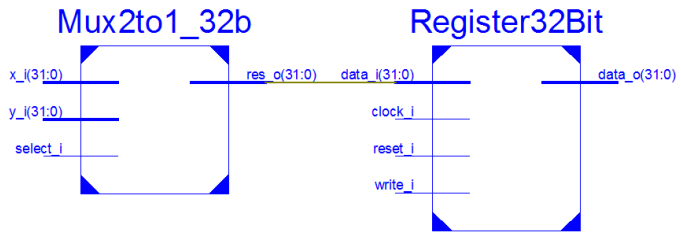
- 3-operandni registrsko-registrski (load/store) procesor,
- dolžina pomnilniške besede je 8 bitov,
- dolžina pomnilniškega naslova je 32 bitov,
- dolžina ukazov je 32 bitov,
- vsebuje 32 splošnonamenskih registrov od R0-R31, katerih dolžina je 32 bitov,
- vse ALE operacije se izvedejo v eni urini periodi,
- poravnanost sestavljenih pomnilniških operandov je obvezna,
- uporablja pravilo debelega konca.

Cevovod je sestavljen iz petih stopenj. Vsaka stopnja ima pripadajoče ime; IF, ID, EX, MEM, WB. Podatkovna enota petstopenjskega cevovoda je opisana v knjigi [1] v poglavju 7.2. Cevovod pohitri delovanje procesorja, saj se nekatere stvari lahko izračunavajo sočasno. Torej procesor v eni urini periodi stori več kot ne-cevovodni HIP. V ne-cevovodnem procesorju HIP se lahko v isti periodi vedno izvaja samo en ukaz, kar pa pri cevovodnem procesorju ne drži. V vsaki stopnji se lahko izvaja del enega ukaza. Ker se v stopnjah cevovoda nahaja več ukazov naenkrat, obstaja možnost, da pride do podatkovnih nevarnosti. Nevarnost nastane takrat, ko ukaz zahteva branje vsebine iz splošnonamenskega registra, v katerega bo ukaz, ki je prej prišel v cevovod, šele kasneje zapisal vsebino v ta register. Tudi pri skočnih

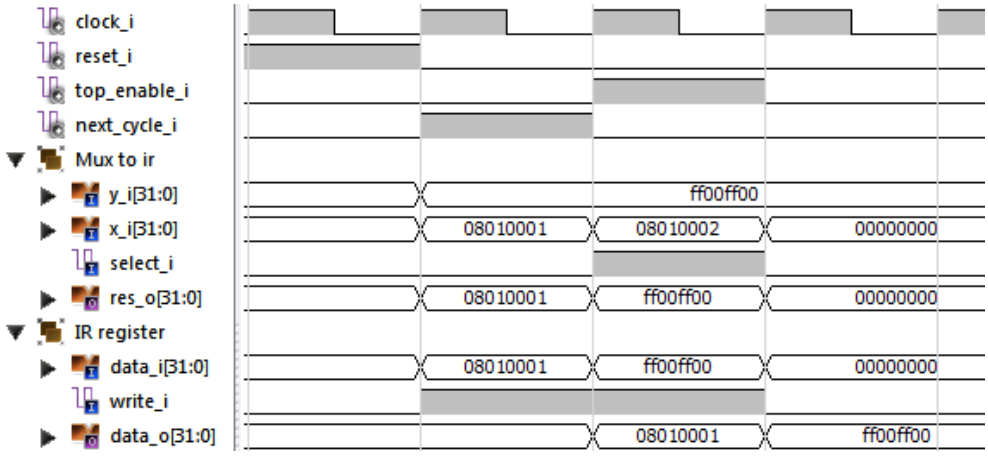
ukazih je potrebno reševati kontrolne nevarnosti. Skočna nevarnost se zgodi, ko je v stopnji EX ukaz, ki spremeni vsebino registra PC. Ukazom, ki se nahajajo za skočnim ukazom, se ne smejo izvesti, ko se spremeni register PC. HIP mora vsebovati logiko za ugotavljanje podatkovnih in kontrolnih nevarnosti ter jih mora tudi znati reševati. Z vgrajeno logiko za premoščanje je delovanje procesorja mogoče še dodatno pohitriti. Logika za premoščanje omogoča procesorju pridobiti podatek iz naslednjih stopenj cevovoda, če je že na voljo, še preden bo zapisan v splošnonamenski register.

2.2.1 Branje in pisanje v registre

Razhroščevalnik ima možnost pisati in brati v vmesne registre v cevovodu. Pred vsakim registrom se nahaja izbiralnik, ki izbira kateri podatek se bo zapisal v register. V vhod v multiplekser sta povezana dva signala, od katerih je eden namenjen za normalno delovanje procesorja HIP, drugi pa omogoča pisanje v register preko razhroščevalne enote. Razhroščevalna enota kontrolira vse izbiralnike pred registri, v katere lahko piše. Povezava med izbiralnikom in registrom je prikazana na sliki 2.1. Na primer, med izvajanjem naslednje urine periode v procesorju HIP, razhroščevalna enota vedno omogoča, da so preko izbiralnika izbrani vedno tisti podatki na vhodu registra, kot so opisani v knjigi [1]. Na sliki 2.2 je razviden primer pisanja podatka v register IR. S signalom “select_i”, ki ga tvori razhroščevalna enota, se izbira ali se bo podatek pisal iz prvega ali iz drugega izvora. V drugi urini periodi se izvaja naslednji cikel v procesorju HIP, kakor je razvidno iz signala “next_cycle_i”. Zato se prebran podatek iz ukaznega predpomnilnika piše v register IR. Ko je omogočen signal “top_enable_i”, pa se v register IR piše podatek iz razhroščevalne enote, kot je vidno v 3. urini periodi.



Slika 2.1: Izbiralnik in register za namene razhroščevanja.



Slika 2.2: Primer pisanja v register preko izbiralnika.

2.3 Predpomnilnik

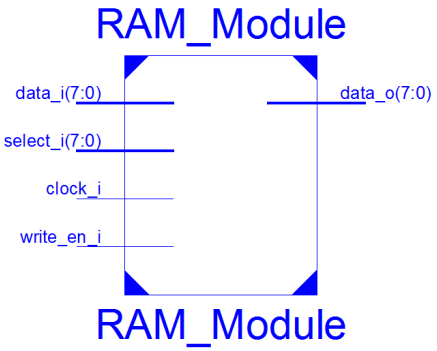
Predpomnilnik je vezje, v katerem so začasno shranjeni operandi in podatki. Predpomnilniki so namenjeni predvsem zmanjšanju prepada med počasnostjo glavnega pomnilnika in hitrostjo centralne procesne enote. Brez predpomnilnika bi namreč procesor večino časa čakal na podatke ali ukaze iz relativno počasnega glavnega pomnilnika.
Predpomnilnik v HIP je zgrajen po Harvardski arhitekturi. Vsebuje dva ločena predpomnilnika. Prvi je namenjen za ukaze, drugi pa za podatke. Do ukaznega predpomnilnika se dostopa v prvi (IF) stopnji cevovoda. HIP lahko iz ukaznega predpomnilnika izvršuje samo bralne operacije. Problema poravnanosti naslovov iz ukaznega predpomnilnika ni, saj so vsi ukazi dolgi 32 bitov in za poravnanost poskrbi prevajalnik. Do drugega, podatkovnega predpomnilnika, HIP dostopa v četrti (MEM) stopnji. Tu se lahko izvršijo bralni ali pisalni ukazi. Potrebno je reševati poravnanost naslovov, saj dostopi do podatkov niso vedno 32-bitni.

2.3.1 Sestava predpomnilnika

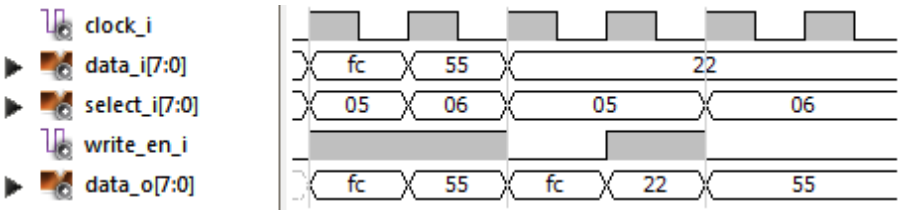
Predpomnilnik je sestavljen iz podatkovnega in kontrolnega dela. Podatkovni del vsebuje podatke. Kontrolni del pa vsebuje dodatne kontrolne podatke, ki povedo, iz katerega naslova glavnega pomnilnika je bil preslikan blok v predpomnilnik. Vsebuje tudi ali je blok veljaven in umazan. Blok postane umazan takrat, ko se v blok piše s pisalnim ukazom.

Podatkovni del predpomnilnika je sestavljen iz štirih modulov, kot je prikazano na sliki 2.3. Na sliki 2.3 je primer RAM, ki ima naslovljivih 2^8 besed. Vsaka beseda hrani osem bitov, torej en bajt. Vhodna in izhodna podatkovna širina vodila znaša osem bitov.

Ko se pri naslavljanju RAM pojavijo na vhodu signali “select_i”, se iz modula RAM na izhod postavijo naslovljeni podatkovni biti “data_o”. V primeru pisanja v RAM s podatkovnimi signali “data_i”, se poleg naslovnih signalov postavi na logično 1 še signal “write_en_i”. Vezje ob prehodu urine



Slika 2.3: Shema modula RAM.



Slika 2.4: Nivoji signalov pri branju in pisanju v RAM.

periode iz nizkega v visoko stanje zapiše podatek v RAM. Slika 2.4 prikazuje primer naslavljanja modula RAM in pisanje v njega. V prvi urini periodi se v RAM zapiše podatek FC_{16} na naslov 5_{16}. V drugi urini periodi pa se zapiše podatek 55_{16} na naslov 6_{16}. V tretji urini periodi se iz modula RAM prebere podatek iz naslova 5_{16}. Na izhodu "data_o" je viden prej shranjen podatek.

HIP ima 32-bitni dostop do predpomnilnika. V eno RAM besedo se lahko shrani osem bitov ali en bajt naenkrat. Da je mogoče shraniti štiri bajte podatkov, so potrebni štirje RAM moduli. To velja za operandni in za podatkovni predpomnilnik.

Kontrolni del predpomnilnika mora iz naslova in ukaza ugotoviti v kateri RAM modul se bodo shranili ali brali podatki. Pri shranjevalnih ukazih, ki shranijo en bajt, se ta podatek shrani samo v en RAM modul. Pri ukazih,

kjer se shranita dva bajta, pa se podatek shrani v dva RAM modula. Zaradi poravnanosti pri shranjevanju se 16-biten podatek shrani v prvi in drugi RAM modul ali v tretji in četrti RAM modul. Podatek se pri shranjevanju štirih bajtov zapiše v vse RAM module.

Na sliki 2.5 je prikazano vezje "Cache_Data_Write_Align". To vezje poskrbi, da se pri shranjevanju biti pravilno poravnajo po pripadajočem naslovu in tipu shranjevalnega ukaza. Kontrolni del predpomnilnika poskrbi, da se podatek zapiše v prave RAM module. Vezje "Block_Align" omogoča bralnim ukazom pravilno postavitev bitov na izhod predpomnilnika. Če se bere en bajt iz tretjega RAM modula, mora vezje "Block_Align" bite 23..16 premakniti na pozicijo 7..0. Podobno je tudi pri branju iz drugih naslovov in pri branju 16-bitnih podatkov. Ostali biti, ki zaradi tipa bralnega ukaza niso relevantni pri branju iz blokov RAM, vezje postavi na logično 0. Pri 8-bitnem branju, se biti od 31..8 postavijo na nič. Ne glede ali branje zahteva ukaz za nepredznačeno branje ali ne. Tabela 2.3 prikazuje, kateri RAM moduli so aktivni pri pisanju v predpomnilnik. Na podlagi spodnjih dveh bitov naslova, s katerim se dostopa do predpomnilnika in tipa ukaza za shranjevanje, se določi v katere RAM module se bo podatek zapisal. Signal "type_i" je direktno preslikan iz operandnih bitov 27 in 26. Biti v signalu "write_o"pa so izračunani po formuli. V tabeli so napisane samo veljavne možnosti. Predpomnilniško vezje nima detekcije pravilne poravnanosti naslovov, ne glede ali gre za pisalni ali bralni ukaz. To je naloga kontrolnega dela HIP.

V kontrolnem delu predpomnilnika se nahaja pomnilnik za hranjenje metapodatkov in kombinatorično vezje, ki na podlagi ukazov in metapodatkov upravlja s predpomnilnikom. Predpomnilnik je razdeljen na več enako velikih blokov. Vsak blok ima naslovljivih določeno število sosednjih besed. Vsak blok ima v kontrolnem delu svoj kontrolni naslov, ki pove kateri del podatkov se je preslikal iz glavnega pomnilnika ter dva kontrolna bita, ki povesta ali je blok veljaven (bit V) in umazan (bit U). Pri dostopu do predpomnilnika se dostopa tudi do kontrolnih metapodatkov.

address_i[1..0]	type_i[1..0]	write_o[3..0]
00	00	0001
01	00	0010
10	00	0100
11	00	1000
00	01	0011
10	01	1100
00	10	1111

Tabela 2.3: Resničnostna tabela pri pisanju v predpomnilnik. "Type_i" bita sta vzeta iz 27. in 26. operandnega bita. Signal "write_o" je izračunan iz tipa ukaza in iz spodnjih dveh bitov naslova.

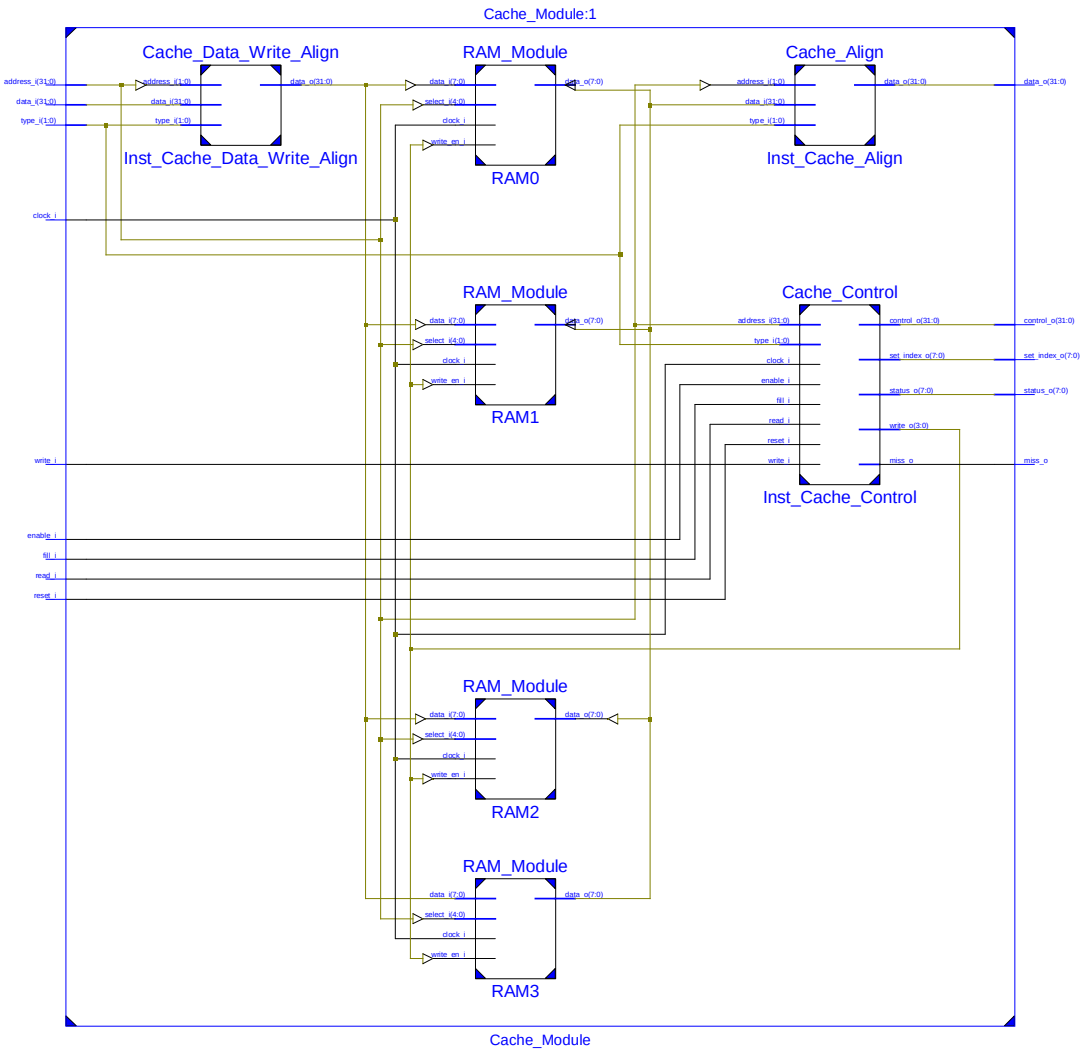
Kombinatorično vezje na podlagi kontrolnih bitov ve ali je podatek prisoten v enem izmed blokov. Pri pisanju se nastavlja umazani bit in se ga postavi na "1", če se naslov in del naslova v kontrolnem delu ujemata. Neujemanje naslovov pomeni zgrešitev v predpomnilniku.

V diplomski nalogi je implementiran direktni predpomnilnik. Oba predpomnilnika imata velikost izračunano po formuli 2.3.

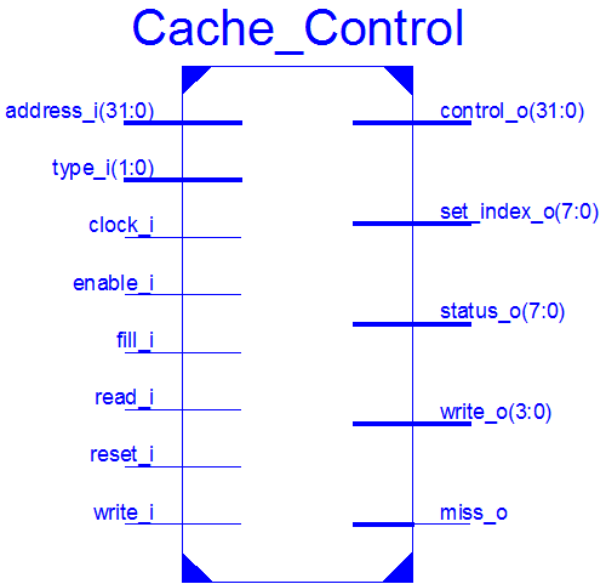
$$\begin{aligned} velikost &= št.\ pom.\ besed\ v\ bloku \times št.\ blokov \times št.\ modulov \\ &= 8 \times 4 \times 4 \end{aligned} \tag{2.3}$$

Torej vsak predpomnilnik ima po 4 bloke in 4 RAM module. Vsak blok lahko naslovi 8 pomnilniških besed, kar skupaj znaša 128 besed.

V kontrolnem delu se nahaja sekvenčno vezje, ki je odgovorno za shranjevanje kontrolnih podatkov. Vsak blok v predpomnilniku ima svoj veljavni bit, umazani bit in zgornje bite naslova, od kjer je bil blok iz glavnega pomnilnika preslikan. Direkten tip predpomnilnika ima indeks, iz katerega bloka se bere ali v kateri blok se piše, vsebovan že v naslovu. Vezje v predpomnilniku mora biti pri zgrešitvah sposobno prepisati stare podatke z novimi. Vezje mora bloke z umazanim bitom zapisati nazaj v glavni pomnilnik. To vezje je implementirano v "Top_Controller" modulu.



Slika 2.5: Shema RAM.



Slika 2.6: Shema RAM kontrolerja.

Zgrešitev se zgodi takrat, ko blok pri bralnem ali pisalnem dostopu v predpomnilniku ni prisoten, torej vrednost veljavnega bita naslovljenega bloka je nič. Prav tako se zgrešitev zgodi, ko se kontrolni biti naslova bloka in naslovljenega bloka razlikujejo med seboj, čeprav je veljaven bit postavljen na ena. Pri zgrešitvah se vse stopnje cevovoda ustavijo in čakajo določeno število urinih period. V predpomnilnik se v deseti urini periodi zapišejo novi podatki. Veljavni bit se po koncu pisanja novih podatkov postavi na ena, umazani bit na nič in v kontrolni del se zapišejo naslovni biti bloka. Naenkrat se lahko v predpomnilnik preslika samo en blok novih podatkov.

S signalom "fill_i", prikazanim na sliki 2.6, se umazani bit ne postavi v visoko stanje. Razhroščevalna enota uporablja signal"fill_i" pri polnjenju naslovljenega bloka z novimi podatki. Na izhodu kontrolnega dela so tudi signali "write_o", ki poskrbijo za pisanje in branje v RAM module. Signali predpomnilniškega kontrolnega modula so opisani v tabeli 2.4.

Na sliki 2.7 so prikazani signali pri polnjenju, branju in pisanju podatkov

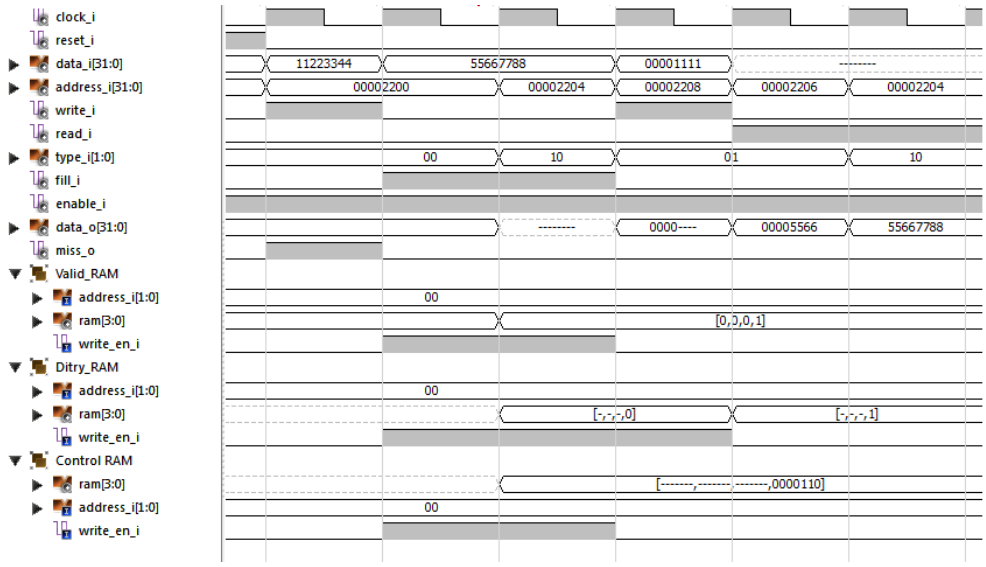
vhodi/izhodi	št. bitov	opis
address_i	32	naslovni signali
type_i	2	način pisanja (8, 16, 32-bitni)
fill_i	1	polnjenje bloka
read_i	1	branje iz predpomnilnika
write_i	1	pisanje v predpomnilnik
control_o	32	kontrolni biti naslova naslovljenega bloka
set_index_o	8	del naslova naslovljenega bloka
status_o	8	kontrolni biti naslovljenega bloka
write_o	4	kontrolni biti za naslavljanje RAM blokov
miss_o	1	kontrolni bit za zgrešitev pri pisanje ali branju

Tabela 2.4: Pomen signalov kontrolnega modula predpomnilnika.

v predpomnilnik. V prvi urini periodi je prikazano pisanje v predpomnilnik. Naslovljen blok ni veljaven, torej predpomnilnik javi zgrešitev. V drugi urini periodi se preko signala "fill_i" zapiše 8-bitni podatek v naslovljeni predpomnilniški blok. V kontrolnem delu predpomnilnika se umazani bit naslovljenega bloka postavi na 0 in veljavni bit na 1. V naslovni del kontrolnega dela se zapiše zgornji del bitov naslova novega bloka, iz katerega je bil blok preslikan. V tretji urini periodi se v blok zapiše 32-bitni podatek.

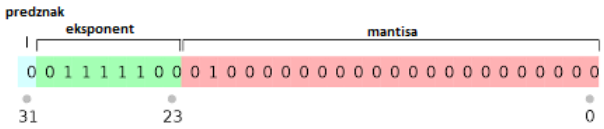
2.4 Enota za računanje v plavajoči vejici

V računalništvu je poleg celih števil potrebno operirati tudi s števili zapisanimi v posebnem zapisu, ki mu pravimo zapis v plavajoči vejici (PV). Števila, ki jih je mogoče zapisati, imajo zelo širok razpon. Število je mogoče zapisati kot $m \times r^e$, kjer je m mantisa - koeficient, r baza ali radiks in e je eksponent. Eksponent pove na katerem mestu je decimalna vejica, ki se z razliko z zapisom s fiksno vejico lahko premika. Od tod tudi njeno ime. Baza r je konstanta in ima vrednost 2. Eksponent in mantisa pa sta lahko

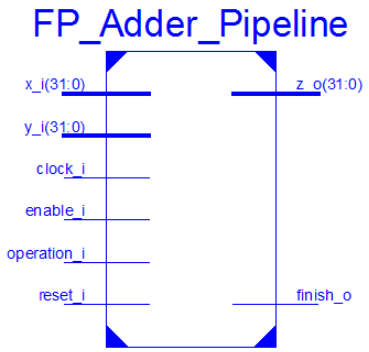


Slika 2.7: Shema RAM kontrolerja.

različno dolgi. Standardizirani so trije zapisi števil v plavajoči vejici. Eden izmed teh zapisov je zapis v enojni natančnosti. Zapis je dolg 32 bitov. Skrajno levi bit predstavlja predznak. Naslednjih osem bitov predstavljajo eksponent, ki je zapisan z odmikom. Ostalih 23 bitov predstavlja mantiso. Pri zapisu mantise je prvi bit mantise, ki se nahaja pred vejico, implicitno podan. Če je vrednost eksponenta večja od nič, potem je implicitno podan bit enak ena in ta števila so normalizirana. V drugih primerih pa je prvi bit enak nič. Temu številu pravimo denormalizirano število. Obstajata še dve posebni števili, ki predstavljata neskončnost in “ni število - NaN (Not a Number)”. Neskončnost je predstavljena, če so v eksponentu vsi biti enaki ena in v mantisi vsi biti enaki nič. Če je v mantisi kateri izmed bitov enak ena, potem gre za zapis NaN. Logika za izvajanje različnih operacij morajo biti sposobna obdelovati vsa predstavljena števila, do katerih lahko pride. V naslednjih implementiranih računskih enotah se obdelujejo števila, ki so zapisana v enojni natančnosti. Slika 2.8 prikazuje zgradbo zapisa v plavajoči vejici v enojni natančnosti.



Slika 2.8: Zapis števila v PV v enojni natančnosti.



Slika 2.9: Shema enote za seštevanje in odštevanje v PV.

2.4.1 Seštevanje in odštevanje

HIP ima implementirano logiko za seštevanje in odštevanje v plavajoči vejici (PV). Nahaja se v EX stopnji cevovoda. Potrebna sta bila dva dodatna ukaza. Eden za seštevanje in eden za odštevanje. Ukaz je zakodiran v formatu 2. Operacijska koda ukaza za operacije v plavajoči vejici je 110111_2. Spodnjih enajst bitov ukaza za seštevanje je 00000000000_2, za odštevanje pa 00000000001_2. V zbirnem jeziku za HIP je operacija za seštevanje označena s "FADD", za odštevanje pa s "FSUB". Kontrolni del HIP kontrolira enoto za seštevanje in odštevanje v PV na podlagi ukaza, ki se nahaja v registru IR1. Računanje v PV je zahtevnejše in je izvedeno v več korakih. Med izvajanjem so ostale stopnje cevovoda v čakajočem stanju. Rezultat se v sedmi urini periodi, preko izbiralnika, shrani v register C. Shema enote za seštevanje in odštevanje v PV je prikazana na sliki 2.9.

V prvem koraku se oba operanda shranita v začasna registra. V drugem

Št. A	Št. B	Rezultat
0	B	B
A	0	A
NaN	B	NaN
A	NaN	NaN
$+/-\infty$	$+/-\infty$	$+/-\infty$
$+/-\infty$	$-/+\infty$	NaN
normalizirano	normalizirano	normalizirano
denormalizirano	denormalizirano	denormalizirano ali normalizirano

Tabela 2.5: Vrste števil pri seštevanju.

koraku se preveri ali je kateri od operandov posebno število. V drugem koraku se tudi razširi mantiso na 28 bitov. Najpomembnejši bit mantise se postavi na 1, če je število normalizirano. V primeru denormaliziranega števila pa se postavi na 0. Poleg tega se izračuna razliko med eksponentoma in razliko se shrani v začasni register "d". V primeru, da je eno število posebno, je tudi rezultat posebno število in operacija se zaključi. Seštevanje števil, ki imajo poseben pomen, ni vedno možno. V tabeli 2.5 so prikazani možni vhodi dveh števil v plavajoči vejici in kakšen je rezultat pri seštevanju. Normalizirana in denormalizirana števila se pošljejo v naslednji korak, kjer se ju uredi po velikosti. Mantiso manjšega števila se pomakne za d mest v desno. Na izpraznjena mesta se vstavijo ničle. V četrtem koraku se izvede seštevanje ali odštevanje mantis. Tip operacije je odvisna od vhodne operacije in predznakov števil. Pri seštevanju je potrebno upoštevati tudi preliv, če do njega pride. Pri prelivu je potreben premik mantise v desno za en bit. Večji eksponent je potrebno povečati za ena in ga začasno shraniti v register. Pri odštevanju se lahko zgodi, da so na prvih mestih ničle. Rezultat je potrebno pomakniti v levo za ustrezno število mest. Štetje ničel je narejeno s posebnim vezjem. Vezje in algoritem za štetje začetnih ničel je opisan v članku [5]. V primeru, da se pomika zaradi

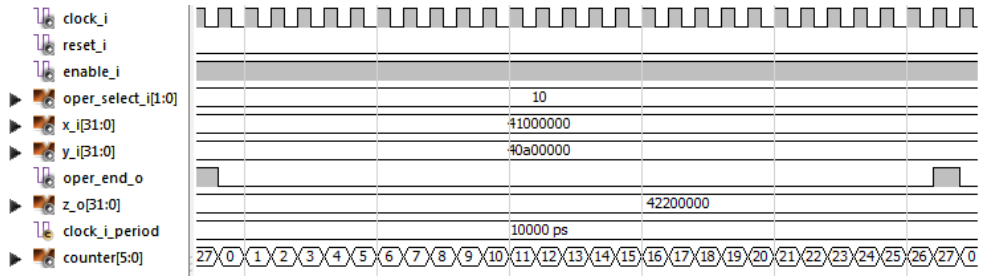
premajhnega eksponenta ne more v celoti izvesti, je rezultat denormalizirano število. Eksponent je v tem primeru enak nič. Če je število ničel večje od eksponenta, se izvede pomik v levo za vrednost eksponenta. Če sta enaka, je nova vrednost eksponenta enaka ena in prav tako se izvede pomik v levo. V ostalih primerih se od eksponenta odšteje število začetnih ničel in mantiso se premakne v levo. Najpomembnejši bit v mantisi je enak ena. Z šestem koraku je potrebno mantiso zaokrožiti na 23 bitov. Zaokroženje se izvede tako, da se mantiso zaokroži k najbližjemu številu. Pri zaokroževanju lahko pride do preliva mantise. Pri prelivu je potrebno mantiso pomakniti v desno za en bit in eksponent je potrebno povečati za ena. Končni rezultat se shrani v končni register in v zadnjem koraku se kontrolno enoto, preko signala "finish_o", obvesti, da je računanje končano. Kontrolna enota HIP preko izbiralnika pred registrom C izbere, da se rezultat iz enote za računanje v plavajoči vejici zapiše v register C. Enota za seštevanje in odštevanje v plavajoči vejici vsebuje kompleksnejša vezja, zato je tudi prostorsko in časovno potratna. V primeru razvojne ploščice Spartan 3E, enota vzame okoli 10 procentov celotne razpoložljivosti vezja. To je okoli 520 rezin (slices) v čipu FPGA.

2.4.2 Množenje

Enota za množenje je v resnici enostavnejša kakor enota za seštevanje in odštevanje. Enota za množenje v prvem koraku shrani oba operanda v začasna registra. V drugem koraku se mantiso razširi in izvede seštevanje eksponentov, kot je prikazano s formulo (2.4). Od rezultata se odšteje 127, saj sta eksponenta zapisana z odmikom.

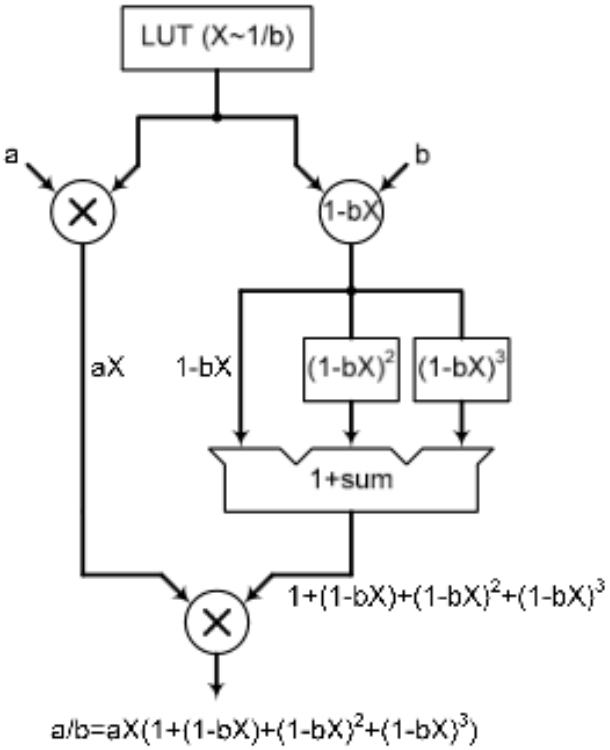
$$\begin{aligned} 2^{eksponent-127} &= 2^{a-127} \times 2^{b-127} \\ eksponent - 127 &= a + b - 254 \\ eksponent &= a + b - 127 \end{aligned} \tag{2.4}$$

V naslednjih korakih se izvaja množenje tako, da se v vsakem koraku



Slika 2.10: Primer množenja dveh števil v PV.

preveri najvišji bit mantise drugega operanda. Če je bit enak ena, se začasnemu rezultatu prišteje vrednost mantise prvega operanda. Za naslednji korak se mantiso prvega operanda pomakne v desno za en bit in shrani v začasni register. Mantiso drugega operanda se pomakne v levo za en bit in shrani v začasni register. Ta korak se izvaja, dokler se ne preverijo vsi biti drugega operanda. Ker je mantisa drugega operanda dolga 24 bitov, ta korak traja 24 urinih period. Če pride do preliva pri parcialnem seštevanju, je na koncu potrebno povečati eksponent rezultata za ena in mantiso pomakniti za en bit v desno. Rezultat se na koncu zaokroži na 23 bitov. Če pride pri zaokroževanju do preliva, je ponovno potrebno povečati eksponent za ena. Zaokrožen rezultat se shrani v končni register in v naslednji urini periodi se iz enote za množenje pošlje rezultat in signal "finish_o" se postavi v visoko stanje, kar nakazuje, da je operacija končana. Enota za množenje je bolj enostavna kot enota za seštevanje in odštevanje, kar se kaže tudi v manjši zasedenosti čipa FPGA. Enota zasede okoli 260 ali 5 procentov rezin v čipu FPGA. Za množenje je bilo potrebno dodati nov ukaz "FMUL". Operand za množenje v plavajoči vejici je 110111_2. Spodnjih enajst bitov ukaza so enaki zapisu 00000000010_2. Potek množenja dveh števil in rezultat, po 28. ciklu, je prikazan na sliki 2.10.



Slika 2.11: Potek deljenja v PV.

2.4.3 Deljenje

Deljenje v plavajoči vejici je redka operacija, ampak prinaša velike prihranke, če je deljenje strojno realizirano. Za deljenje je uporabljena formula (2.5). Princip delovanja delilnika je vzet iz članka [6]. Potek računanja je prikazan na sliki 2.11.

$$\frac{a}{b} \approx aX_0(1 + (1 - bX_0) + (1 - bX_0)^2 + (1 - bX_0)^3) \tag{2.5}$$

$$X_0 \approx \frac{1}{b} \tag{2.6}$$

Formula izračuna količnik dveh števil. Operand X_0 je približna recipročna vrednost delitelja b (2.6). V prvem koraku se vrednost X_0

Korak	Operacija	Št. A	Št. B	Rezultat
1.	dostop		1/B	X_0
2.	×	X_0	b	BX0
3.	-	1	b	1BX0
4.	+	1	1BX0	SUM
5.	×	1BX0	1BX0	2BX0
6.	+	2BX0	SUM	SUM
7.	×	2BX0	1BX0	3BX0
8.	+	3BX0	SUM	SUM
9.	×	X_0	SUM	SUM
10.	×	a	SUM	SUM

Tabela 2.6: Koraki pri deljenju.

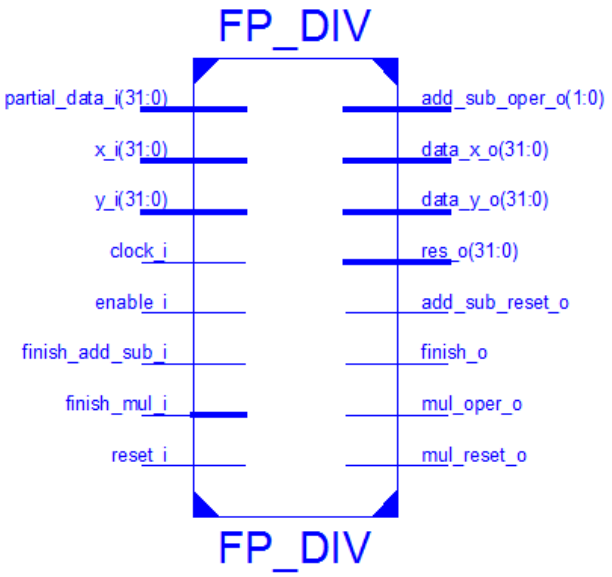
pridobi iz vnaprej definirane tabele recipročnih vrednosti. Tabela recipročnih vrednosti vsebuje 16 6-bitnih zapisov. Vrednosti v tabeli se naslavlja z najpomembnejšimi biti mantise. Dobljeno delno recipročno mantiso se razširi na 24 bitov. Eksponent se izračuna istem koraku. Število se začasno shrani v register X_0. V drugem koraku se izračuna zmnožek med številoma b in X_0. Enota za deljenje nima svojega množilnika. Oba faktorja pošlje v skupen množilnik, kjer se števili zmnožita in zmnožek se shrani v register "BX0". V tretjem koraku se od konstante 1, zapisane v plavajoči vejici, odšteje v prejšnjem koraku izračunan rezultat. Tudi za seštevanje in odštevanje se uporabi isto enoto, kot jo uporabljata ukaza "FADD" in "FSUB". Začasni rezultat se shrani v začasni register "1BX0". V četrtem koraku se seštejeta vrednosti ena in vrednost iz registra "1BX0". Vsota se shrani v začasni register "SUM". Vsi nadaljnji koraki so opisani v tabeli 2.6. Ob koncu desetega koraka se kontrolno enoto HIP obvesti s signalom "finish_o', da je operacija zaključena. Na izhod "res_o" se postavi vrednost, ki je prišla v zadnjem koraku iz množilnika. Trenutno implementiran delilnik nima vezja za računanje z denormaliziranimi števili. Prav tako ne

zazna ali sta operanda posebni števili. Delilnik pri posameznih korakih uporablja ostali dve enoti za seštevanje, odštevanje in enoto za množenje. Shranjuje si samo vmesne rezultate. Zaradi enostavnosti se množenje in seštevanje ne izvaja sočasno. Po formuli (2.5), bi vsekakor bilo mogoče nekatere korake izvajati sočasno in tako pohitriti računanje. Operacija deljenja traja 169 urinih period, kot je prikazano v formuli (2.7).

$$\begin{aligned} trajanje = &\ 1 + (4 \times seš./odš.) + (5 \times množ.) \\ = &\ 1 + 4 \times 7 + 5 \times 28 \\ = &\ 169 \end{aligned} \tag{2.7}$$

Na sliki 2.12 so prikazani vhodni in izhodni signali delilnika. V enoto so povezani signali, po katerih prideta števili, ki ju je potrebno deliti. Signala za konec seštevanja in množenja sta potrebna, da delilnik ve, kdaj lahko začne izvajati naslednji korak. Delni rezultat prihaja preko signala "partial_data_i". Na izhodu enote so potrebni signali, ki vsebujejo operanda za seštevanje ali množenje. Poleg tega so potrebni signali, ki povedo, katera enota mora izvajati naslednji korak. Delilnik sam po sebi ni zapletena enota za realizacijo, kar se kaže tudi pri zasedenosti čipa. Implementirani delilnik zasede 190 rezin na čipu FPGA. Delilnik je mogoče pohitriti s pohitritvijo množilnika in seštevalnika. Množilnik je mogoče narediti tako, da izračuna rezultat v desetih urinih periodah. Podobno velja za seštevalnik, ki ga je mogoče narediti tako, da vsoto izračuna v treh urinih periodah. Tako bi tudi delilnik potreboval 63 urinih period. To je skoraj 2,6-kratna pohitritev. Ker delilnik le upravlja s seštevalnikom in množilnikom, je potrebna še dodatna enota, ki določa ali delilnik lahko upravlja s seštevalnikom in množilnikom ter od kod se bodo operandi za računanje jemali.

Seštevalnik, množilnik in delilnik skupaj zasedejo okoli 1000 rezin oziroma 21 procentov rezin na čipu FPGA. HIP skupaj s predpomnilnikoma zasede nekaj čez 2000 rezin. Ob tem dejstvu je jasno razvidno, da logika za računanje v plavajoči vejici lahko zasede okoli polovico rezin, kot jih zasede logika za HIP, ki zasede 41 procentov rezin.



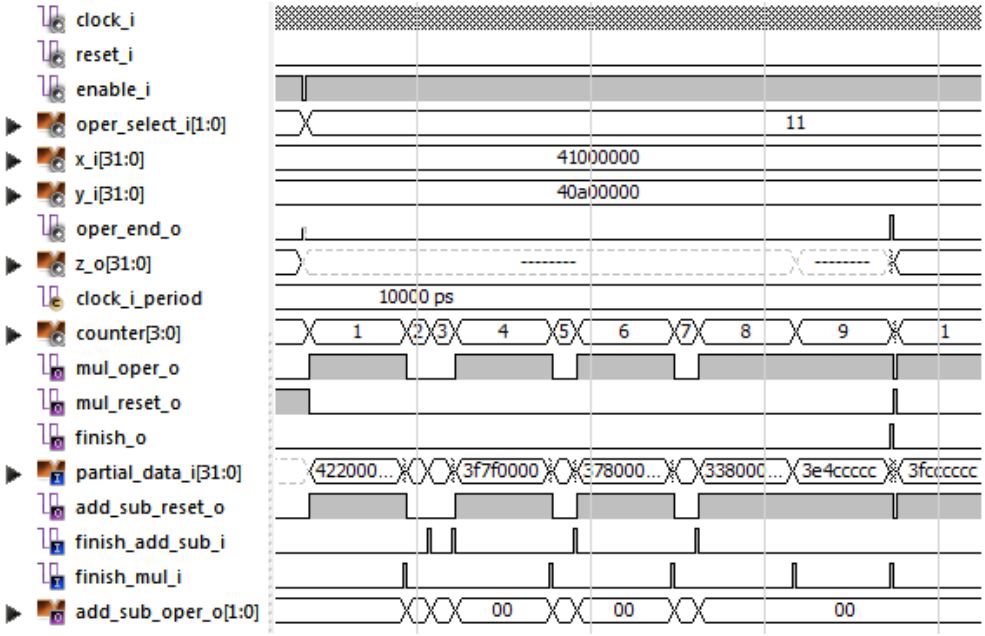
Slika 2.12: Shema enote za deljenje v PV.

Skupaj HIP in enota za računanje v plavajoči vejici zasedeta okoli 2900 ali 62 procentov rezin čipa FPGA, ki se nahaja na ploščici Spartan 3E. Ukaz za deljenje je podoben ostalim operacijam za računanje v plavajoči vejici. Operand je enak 110111_2, funkcijski del ukaza pa je enak 00000000011_2. Zapis ukaza v zbirnem jeziku je "FDIV". Na sliki 2.13 je prikazan primer deljenja dveh števil.

2.5 TopController - Razhroščevalna enota

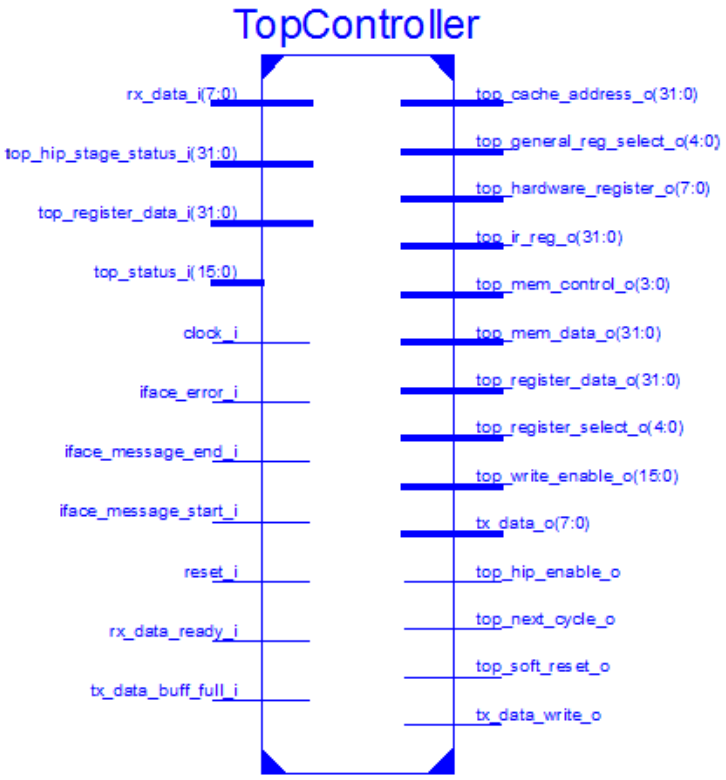
Kontrolna enota z imenom "TopController" je vmesnik med procesorjem HIP in UART modulom. Kontrolna enota na podlagi prejetih ukazov iz zunanje naprave nadzoruje procesor HIP. Prebere lahko vsebino iz splošnonamenskih registrov, ostalih registrov v cevovodu ter podatke in metapodatke iz obeh predpomnilnikov.

Shema kontrolne enote je prikazana na sliki 2.14. Preko signalov



Slika 2.13: Primer deljenja števil v PV.

"rx_data_i" in "rx_data_ready_i", kontrolna enota sprejema podatke iz UART modula. Pri sprejemanju ukaza se modul vedno zanese na to, da bo ob visokem signalu "rx_data_ready_i" prejel pravilen podatek. Podobno je pri pošiljanju odgovora v UART modul. Na izhodu so prisotni signali "tx_data_o", "tx_data_write_o" in "tx_data_buff_full_i", ki so potrebni pri pošiljanju odgovora. Modul na izhod "tx_data_o" nastavi 8-bitni podatek. Z visokim signalom "tx_data_write_o" sporoči UART modulu, naj si začasno shrani podatek ter naj ga pošlje zunanji napravi. Vhodni signal "tx_data_buff_full_i" sporoči kontrolerju, naj ne pošilja podatkov v UART modul, saj bo v nasprotnem primeru podatek izgubljen. Vhod "top_register_data_i" je namenjen povezavi z registri, ki jih lahko kontroler prebere. S signalom "top_register_select_o" izbira med registri. Podobno je s signalom "top_general_reg_select_o", ki pove kateri izmed splošnonamenskih registrov bo izbran. Kontroler uporabi izhodni signal "top_register_data_o" za pisanje podatka v izbrani register. Vsi vmesni registri, predpomnilnik in



Slika 2.14: Shmema modula “TopController”.

bit	register	bit	register
0	splošnonamenski register	8	C
1	PC	9	MAR
2	PC1	10	SDR
3	IR	11	IR2
4	PC2	12	C1
5	A	13	IR3
6	B	14	EPC
7	IR1	15	enota za PV
17	podatki iz predpomnilnika		
18	kontrolni naslov naslovljenega bloka v pred.		

Tabela 2.7: Indeksi registrov pri branju in pisanju.

bit 3..2	bit 1	bit 0
tip branja/pisanja	0 - ukazni/1 - podatkovni	0 - branje/1 - pisanje

Tabela 2.8: Pomen bitov pri upravljanju predpomnilnika.

vhod v splošnonamenske registre si ta signal delijo, zato je potreben še dodaten signal, "top_write_en_o", ki omogoči pisanje samo v enega. Tabela 2.7 opisuje kateri indeksi morajo biti omogočeni v primeru pisanja v registre. Pisanje ali branje v predpomnilnik je mogoče s signalom "top_cache_address_o", ki pove v kateri blok se bo podatek zapisal. S signalom "top_mem_control_o" se izbere v kateri predpomnilnik se bo pisalo. V tabeli 2.8 so opisani biti za upravljanje s predpomnilnikoma. Bita 3 in 2 skupaj povesta za kakšen tip branja ali pisanja gre. "00_2" pomeni 8-bitno, "01_2" 16-bitno, "10_2" pa 32-bitno pisanje ali branje. Signala "top_ir_reg_o" in "top_mem_data_o" prevzemata vlogo registra, če je predpomnilnik onemogočen. Iz prvega se prebere naslednji ukaz, iz drugega pa podatek.
S signalom "top_hardware_register_o" se omogoča določene module in njihov način delovanja v HIP. Tabela 2.9 prikazuje posamezne bite z opisom

biti	opis
0	
1	premoščanje omogočeno
2	predpomnilnik omogočen
3	
4	
5	
6	onemogočanje detekcije kontrolnih nevarnosti
7	

Tabela 2.9: Opis bitov registra, ki omogoča posamezne module.

funkcije.

2.5.1 Ukazi za kontrolno enoto

Ukazi so sestavljeni iz množice bajtov. Vsak ukaz se začne z znakom "A", desetiško 65 in konča z znakom "B", desetiško 66. Takoj po začetnem znaku se nahaja znak, ki označuje tip ukaza. Ker je število ukazov manj kot 255, za določitev ukaza zadostuje en bajt. Morebitni dodatni parametri, potrebni pri ukazu, se nahajajo za znakom za ukaz. V tabeli 2.10 so opisani vsi ukazi, ki jih prepozna razhroščevalna enota. V tabeli so primeri ukazov, kjer so posamezni znaki zapisani v šestnajstiški obliki. Znak "n" je oznaka za en bajt. Po znaku "n" pa je število, ki pove pozicijo bajta. V primeru, da je bajtov veliko, je razpon opisan z dvema "n", za vsakim pa sta poziciji začetnega in končnega bajta. Razhroščevalna enota za vsakim ukazom pošlje odgovor. Odgovori na ukaze so prikazani v tabeli 2.11. Dolžina odgovorov v bajtih je v naprej določena. Izjema je dolžina ogovora na ukaz "NextCycle", kjer sta možni dve fiksni dolžini. Dolžina je odvisna od tega ali je predpomnilnik omogočen ali ne. Tabela 2.12 na grobo prikazuje zgradbo odgovora na ukaz "NextCycle". Tabela 2.13 prikazuje zgradbo odgovora na ukaz "NextCycle" bolj natančno. Pri omogočenem

Ukazi	znak za ukaz	primer
GetGeneralRegister	C	41 43 n0 42
GetGeneralRegisters	D	41 44 42
SetGeneralRegister	E	41 45 n0 n1 n2 n3 n4 42
SetGeneralRegisters	F	41 46 n0..n127 42
NextCycle	G	41 47 42
SoftReset	H	41 48 42
SetStageRegister	I	41 49 n0 n1 n2 n3 n4 42
GetStageRegister	K	41 4B n0 42
GetStageRegisters	K	41 4C 42
SetHardware	N	41 4E n0 42
GetHardware	N	41 4F 42
SetInstrutionRegister	W	41 57 n0 n1 n2 n3 42
SetMemoryDataRegister	X	41 58 n0 n1 n2 n3 42
GetCacheData	P	41 50 n0 n1 n2 n3 n4 42
SetCacheData	Q	41 51 n0 n1 n2 n3 n4 n5..n132 42

Tabela 2.10: Ukazi za razhroščevalno enoto in njihova dolžina.

Ukazi	znak za ukaz	primer odgovora
GetGeneralRegister	C	41 43 n0 n1 n2 n3 42
GetGeneralRegisters	D	41 44 n0..n127 42
SetGeneralRegister	E	41 45 42
SetGeneralRegisters	F	41 46 42
NextCycle	G	41 47 n0 42
NextCycle (predpomnilnik)	G	41 47 n0 n1 n2 42
SoftReset	H	41 48 42
SetStageRegister	I	41 49 42
SetStageRegisters	J	41 4A 42
GetStageRegister	K	41 4B n0 n1 n2 n3 42
GetStageRegisters	K	41 4C n0..n59 42
SetHardware	N	41 4E n0 42
GetHardware	N	41 4F n0 42
SetInstrutionRegister	W	41 57 42
SetMemoryDataRegister	X	41 58 42
GetCacheData	P	41 50 n0..n45 42
SetCacheData	Q	41 51 42

Tabela 2.11: Zgradba odgovorov na posamezne ukaze.

Predpomnilnik	Bajt 0	Bajt 1	Bajt 2
onemogočen	status cevovoda	$\emptyset$	$\emptyset$
omogočen	status cevovoda	status ukaz. pred.	status podat. pred.

Tabela 2.12: Zgradba možnih odgovorov za ukaz “NextCycle”.

bajt\biti	7	6	5	4	3	2	1	0	Pomen
0	0	0	0	0	x	x	x	x	status cevovoda
1	P	0	0	C	W	D	V	M	ukazni pred.
2	P	0	0	C	W	D	V	M	podatkovni pred.

Tabela 2.13: Zgradba statusa cevovoda in predpomnilnika.

Biti	Status	Pomen
7	P	Konec zgrešitvene kazni
6	0	
5	0	
4	C	Zamenjan blok
3	W	Pisanje v blok
5	D	Umazan bit
1	V	Veljaven bit
0	M	Zgrešitev

Tabela 2.14: Pomen statusnih bitov predpomnilnika.

predpomnilniku, se dodatno pošljeta še dva bajta, ki vsebujeta status obeh predpomnilnikov. Natančnejšo obrazložitev pomenov posameznih bitov v statusu predpomnilnika prikazuje tabela 2.14. Tabela 2.15 prikazuje mogoča stanja cevovoda v HIP. Procesor HIP ima pri izvajanju programa različne statuse, ki so odvisni od ukazov v izvajanju. Pri normalnem delovanju je status "OOOOO". V primeru, da je potrebno razrešiti podatkovno nevarnost je status cevovoda "OMOOO". Če je pri skokih potrebno razrešiti kontrolno nevarnost, je status "MMOOO". Cevovod je med izvajanjem operacije v plavajoči vejici v statusu "OOFOO". Statusi "TTTOO", "TTTTO" in "TTTTT" nastopijo, ko se v cevovodu izvaja ukaz TRAP. V primeru omogočenega predpomnilnika je mogoč še status "MMMMM", če pride do zgrešitve v predpomnilniku. Statuse tvori kontrolna enota HIP. Tvori jih na podlagi stanja v cevovodu in ukazov v

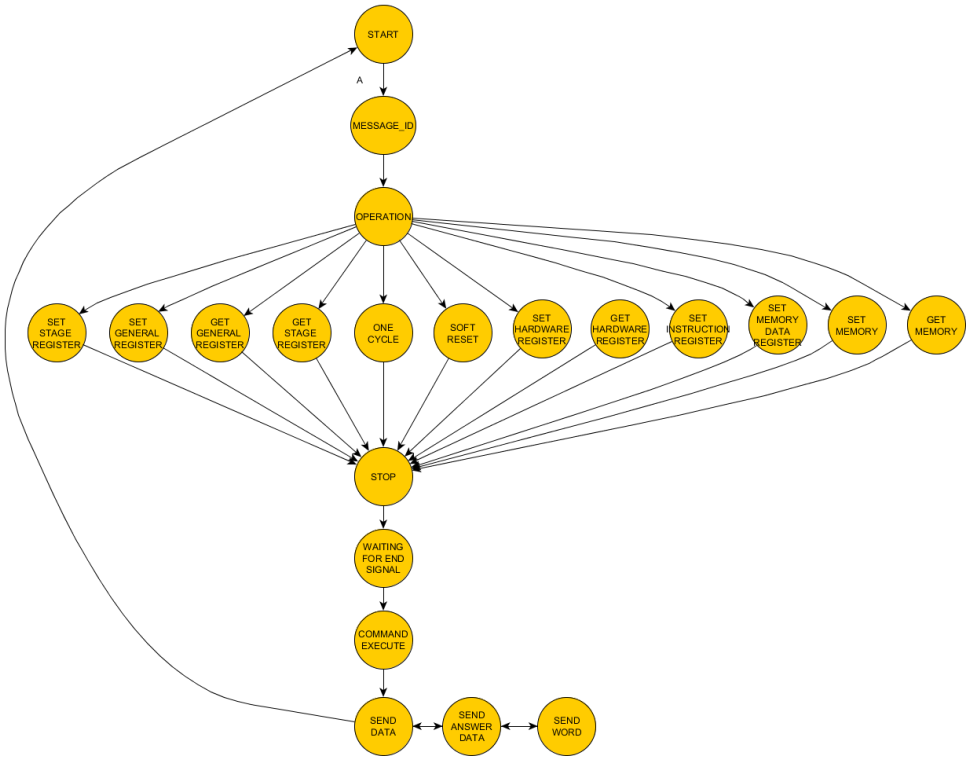
Biti[3..0]	Status
0000	
0001	OOOOO
0010	MMOOO
0011	OMOOO
0100	MMMMM
0101	TTTOO
0110	TTTTO
0111	TTTTT
1000	OOFOO

Tabela 2.15: Možni statusi cevovoda.

izvajanju.

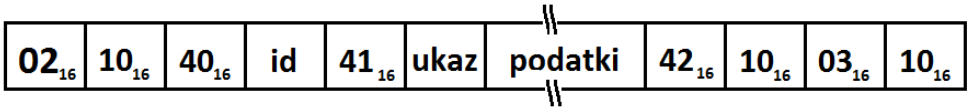
2.5.2 Diagram stanj kontrolne enote

Kontrolna enota je sposobna obdelati veliko različnih ukazov, zato je tudi diagram prehajanja stanj kompleksen. Groba slika diagrama je prikazana na sliki 2.15. V stanju "START" kontroler čaka na začetni bajt "A", nato se prestavi v stanje "MESSAGE_ID". V tem stanju čaka na poljuben bajt. V stanju "OPERATION" sprejme bajt, ki določa operacijo, ki jo bo kontrolna enota izvedla. Bajti, ki so zapisani v tabeli 2.10 določajo naslednje stanje v diagramu. Po sprejetem bajtu se morajo sprejeti in začasno shraniti vsi naslednji bajti, ki so v naprej določeni po dolžini. Zadnji prejeti bajt mora biti znak "B", ki ga kontroler pričakuje v stanju "STOP". Po stanju "STOP" kontroler preide v stanje "WAITING_FOR_END_SIGNAL". V tem stanju kontroler pričakuje visok signal "iface_message_end_i", ki ga kontrolerju pošlje zunanji modul, ko je bilo sporočilo dokončno prejeto. Po sprejetem signalu se v stanju "COMMAND_EXECUTE" začne izvrševati ukaz, ki je bil sprejet v stanju "OPERATION". Po koncu izvrševanja ukaza, se v stanju "SEND_ANSWER"prične pošiljati odgovor na ukaz. V

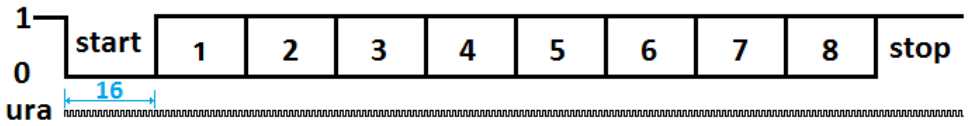


Slika 2.15: Diagram prehajanj stanj v "Top_Controller" modulu.

stanju "SEND_ANSWER" se odgovor ovije z dodatnimi bajti. Ovojnica je prikazana na sliki 2.16. Diagram je pri pošiljanju podatkov v stanju "SEND_DATA". V tem stanju se, odvisno od prejetega ukaza, pošlje ustrezen odgovor. Veliko podatkov je 32-bitne velikosti; te se pošilja v stanju "SEND_WORD". Kontroler se po koncu pošiljanja vrne v prejšnje stanje. Po končanem pošiljanju podatkov, se diagram ponovno prestavi v stanje "SEND_DATA". Po koncu pošiljanja odgovora se diagram prestavi na stanje "START", kjer kontroler čaka na nov ukaz.



Slika 2.16: Zgradba odgovora na ukaz.



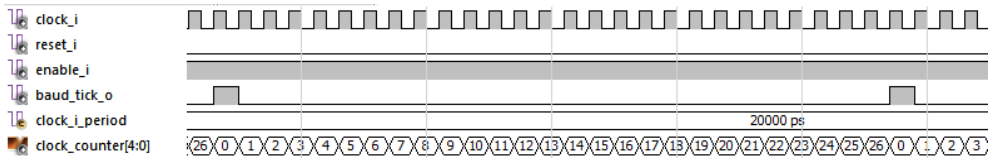
Slika 2.17: Poslan/spejet podatek preko UART vmesnika

2.6 UART

UART je univerzalni asinhronski sprejemnik/oddajnik (Universal asynchronous receiver/transmitter). Preko UART modula je možno komunicirati z zunanjimi napravami, ki morajo v naprej vedeti s kolikšno hitrostjo v baudih bodo pošiljale in prejemale podatke. V večini primerov se pošilja po 8 bitov naenkrat, možne pa so tudi drugačne izvedbe (5-9 bitov). Vsak poslan podatek ima še začetni "start" in končni "stop" bit. Končni stop bit lahko traja več kot traja pošiljanje enega bauda. Pošiljanje je podrobneje zapisano v poglavju 2.6.3. Primer poslanih podatkov je prikazan na sliki 2.17. UART je narejen po principu opisanem v knjigi [4] v 7. poglavju.

2.6.1 Baud generator modul

Modul "Baud generator" tvori periodičen signal za sprejemni RX in oddajni TX modul. RX modul potrebuje signal za določanje trenutka branja signala na vhodu sprejemnika RX. Oddajni modul TX pa potrebuje signal za pošiljanje signala ven iz TX modula. Modul vsebuje števec, ki se povečuje vsako urino periodo. Modul ima nastavljen generični parameter za urino frekvenco in število baudov na sekundo. Pri sintezi modula, se



Slika 2.18: Primer ene periode notranje ure za potrebe RX in TX modula.

izračuna do katerega števila se števec povečuje, preden začne šteti od začetka. Pred ponastavitvijo števca se v naslednji urini periodi na izhodu generatorja postavi signal "baud_tick_o" na 1. Med štetjem pa je signal "baud_tick_o" postavljen na 0. Število urnih period za postavitev signala "baud_tick_o" je izračunan po formuli 2.8.

$$st.u.p = \frac{frek.}{baud/s \times 16} \tag{2.8}$$

Spartan 3E razvojna ploščica uporablja zunanjo uro s frekvenco 50 MHz, ki jo nato deli z dve. Modul ima nastavljeno pošiljanje in prejemanje s 57600 baudov/s. Da se lahko izvaja dovolj natančno merjenje nivojev prejetih bitov mora na vsak baud preteči 16 visokih signalov "baud_tick_o". Torej za Spartan 3E je števec urinih period nastavljen na 27. Na sliki 2.18 je prikazan primer tvorjenja signala "baud_tick_o", ki predstavlja notranjo uro za modul UART.

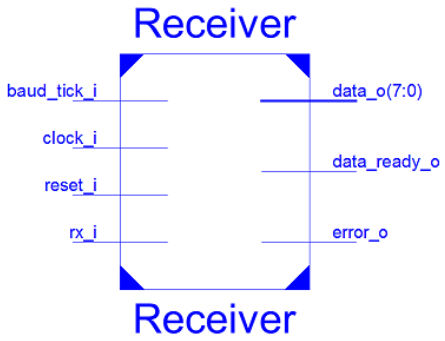
Za vsak 8 bitni podatek je potrebnih 10 baudov. Torej je mogoče izračunati, kolikšen je največji možen prenos uporabnih podatkov:

$$prenos = baud/s \times \frac{8}{10} = 46080\ bit/s \tag{2.9}$$

Hitrost prenosa podatkov zadošča za cilje diplomske naloge.

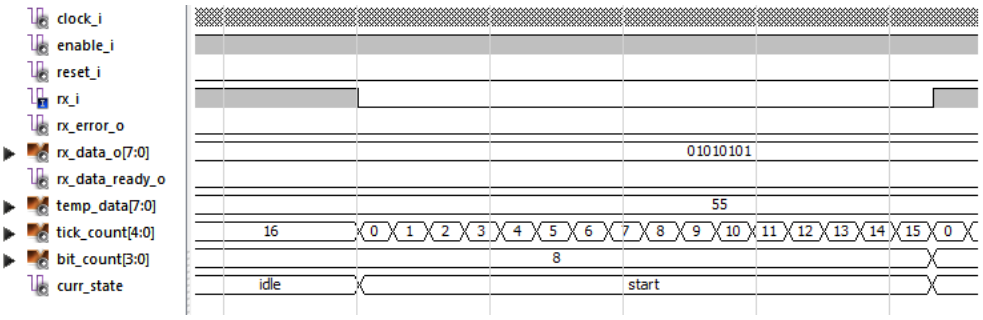
2.6.2 Sprejemni modul "RX"

Slika 2.19 prikazuje "Sprejemni" modul. "Sprejemni" modul sprejema vhodni signal in ga obdela v 8-bitne podatke. Glavna vhodna signala sta "rx" in "baud_tick". Na vhod "rx_i" je pripeljan signal "R7" iz "RXD"

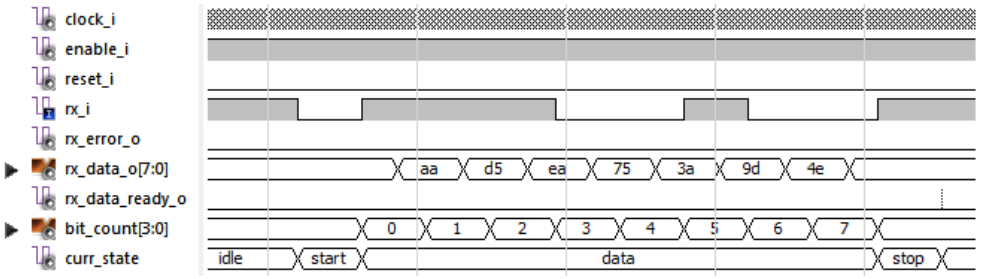


Slika 2.19: Sprejemni "Receiver - RX" modul.

vhoda na razvojni ploščici. Modul preko signala "baud_tick" preverja vhodni signal "rx" ter ga pretvori v 8-bitni izhodni podatek "data_o". Vhod "rx_i" je v mirovanju vedno v visokem stanju. Prehod signala iz visokega stanja v nizkega, nakazuje na začetek pošiljanja podatka. Prvi bit je vedno '0', in ga imenujemo bit "start". Nato se pošlje 8 podatkovnih bitov. Zadnji "stop" bit nakazuje na konec pošiljanja trenutnega podatka. Zadnji bit je vedno je v visokem stanju. Zadnji bit lahko traja 1, 1,5 ali 2 dolžini bita. Vrednost podatkovnih bitov se odčita na sredini vsakega bita. Notranja ura ima periodo 16-krat hitrejšo, kot traja en baud. Podatkovni bit se odčita v 8. periodi. Za natančnejšo določitev nivoja bita, se trikrat odčita nivo signala. Pri tem morajo biti imeti vsi odčitki enako vrednost, sicer se privzame, da je pri branju prišlo do napake. Sprejemni modul "Receiver" ima tudi "error_o" izhod, ki se postavi na "1", če pride do napake pri branju. Pri vsakem odčitku podatkovnega bita, se ta shrani na najvišje mesto v začasni 8-bitni register. Pri pisanju v začasni register se izvede pomik vseh bitov v desno, proti manj pomembni poziciji. Register je torej realiziran kot pomikalnik. Pri branju zadnjega "stop" bita se v 15. urini periodi na izhodu modula postavi signal "data_ready_o" na 1. S tem RX modul nakaže uspešno branje 8-bitnega podatka. Signal "data_o" na izhod preda 8-bitni prebran podatek.

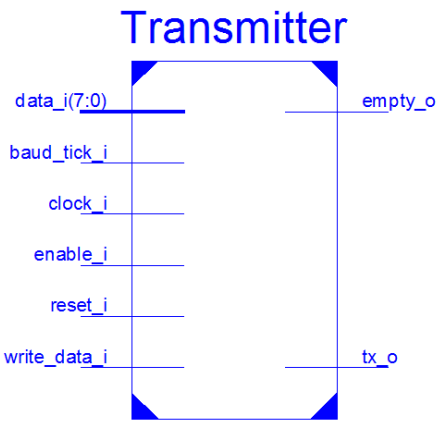


Slika 2.20: Primer branja enega bita v RX modulu.



Slika 2.21: Primer ene periode notranje ure za potrebe RX in TX modula.

Slika 2.21 prikazuje števec period in vhodni signal "rx_i". Nivoji bitov morajo trajati 16 urinih period. RX modul ima implementiran avtomat, ki krmili branje signala "rx_i" in dostavlja podatke na izhod. Na sliki 2.21 se vidi, da ima avtomat štiri možna stanja. Prvo stanje je stanje mirovanja. Vhod "rx_i" je v visokem stanju. Ob spremembi signala "rx_i" iz visokega stanja v nizkega, avtomat preide v naslednje stanje. V drugem stanju se sprejema bit "start", v tretjem pa sprejema podatkovne bite. V zadnjem stanju avtomata poteka branje bita "stop". V 16 urini periodi se na izhod dostavijo podatki. Nato se RX modul vrne v stanje mirovanja, kjer avtomat čaka na nov podatek.



Slika 2.22: Oddajni "Transmitter" modul

2.6.3 Oddajni modul "TX"

Oddajni modul "TX" je namenjen pošiljanju podatkov v zunanjo napravo. Za pošiljanje uporablja notranjo uro, ki jo tvori modul "baud_generator".

Na sliki 2.22 so prikazani vhodni in izhodni signali. 8-bitni podatek se zapiše v vmesni register ob visokem "write_data_i" signalu. V mirovanju je izhodni signal "tx_o" vedno postavljen na 1. Pri pisanju v notranji začasni register, se začne pošiljati začetni "start" bit in izhodni signal "empty_o" se postavi na "0", kar pomeni, da je podatek v pošiljanju. Po prvem bitu se začnejo pošiljati podatkovni biti. Na vsakih 16 period se naredi pomik v registru in nato se začne pošiljati naslednji podatkovni bit. Na koncu se pošlje še "stop" bit, ki ima vedno vrednost "1". Zadnji bit lahko traja več dolžin. Uporabljena je dolžina dveh bitov. Ob koncu pošiljanja podatka in poslanega "stop" bita se na izhod signala "empty_o" postavi vrednost "1", kar nakazuje, da je notranji register prazen. Takrat je možno poslati nov podatek.

```
NET "leds_o[7]" LOC = F9;
NET "leds_o[6]" LOC = E9;
NET "leds_o[5]" LOC = D11;
NET "leds_o[3]" LOC = F11;
NET "leds_o[2]" LOC = E11;
NET "leds_o[1]" LOC = E12;
NET "leds_o[0]" LOC = F12;
NET "rx_i" LOC = R7;
NET "tx_o" LOC = M14;
NET "clock_i" LOC = C9;
NET "leds_o[4]" LOC = C11;
NET "reset_i" LOC = K17;


NET "clock_i" IOSTANDARD = LVCMOS25;
NET "leds_o[0]" IOSTANDARD = LVCMOS25;
NET "reset_i" IOSTANDARD = LVCMOS25;
NET "rx_i" IOSTANDARD = LVTTL;
NET "tx_o" IOSTANDARD = LVTTL;
#Created by Constraints Editor (xc3s500e-fg320-4) - 2015/09/28
NET "clock_i" TNM_NET = clock_i;
TIMESPEC TS_clock_i = PERIOD "clock_i" 40 ns HIGH 50%;
```

Slika 2.23: Datoteka UCF.

2.6.4 Vhodi in izhodi

Vse izhode in vhode v model, ko ga implementiramo na čipu FPGA, je potrebno povezati na realne nogice. V posebni datoteki so definirani vsi vhodi v modul in izhodi iz modula, ki so povezani na pripadajoča imena nogic. Vsebina datoteke je prikazana na sliki 2.23.

Poglavje 3

Programski nivo

Cilj pri izbiri programskih orodij je bila prenosljivost razhroščevalnega programa na različne operacijske sisteme. Programski jezik Java je bila ena izmed izbir, ki je na koncu prevladovala. Uporabljen je grafični vmesnik Swing, ki je dobro poznan pri Java razvijalcih programske opreme. Pri uporabi jezika Java je pomanjkanje podpore za dostop do serijskega vmesnika šibki člen. Potreben prevajalnik je bil implementiran s pomočjo prosto dostopne knjižnice ANTLR (ANother Tool for Language Recognition). Na podlagi slovničnih pravil je s to knjižnico mogoče ustvariti razčljenjevalnik, ki je uporabljen pri prevajanju zbirne kode v strojno kodo.

3.1 Programski jeziki in orodja uporabljena na programskem nivoju

3.1.1 JAVA

Programski jezik Java je splošnonamenski, objektno oziroma razredno orientiran jezik. Skuša biti čim bolj neodvisen od implementacije procesorjev in operacijskih sistemov. Program napisan v programskem jeziku Java, je mogoče izvajati program v različnih operacijskih sistemih. Program se prevede v bajtno kodo. To je množica operacijskih kod, ki jih

pozna Java virtualni stroj. Dovolj je, da je za vsak operacijski sistem narejen Java virtualni stroj, ki zna izvajati program, preveden v bajtno kodo. Tako je mogoče poganjati isti program na različnih operacijskih sistemih.

3.1.2 Eclipse

Eclipse je razvojno orodje namenjen razvoju programske opreme. Namenjen je tudi za razvoj programov v programskem jeziku Java.

3.1.3 ANTLR

ANTLR (ANother Tool for Language Recognition) knjižnica je namenjena tvorjenju razčlenjevalnika (ang. parser). Razčlenjevalnik je program, ki bere, procesira, izvaja in transformira strukturirane ali binarne datoteke. Knjižnica je napisana v Javi in je prosto dostopna na spletu. Knjižnico je možno uporabiti v različnih operacijskih sistemih. Poleg knjižnice ANTLR obstajajo tudi drugi programi, ki tvorijo razčlenjevalnike. Med najbolj znanimi je program YACC (Yet Another Compiler-Compiler). Pri delu je prav prišla tudi knjiga [8], kjer so opisana osnovna pravila za opis slovnice jezika in navodila za tvorjenje razčljenjevalnika.

3.1.4 Knjižnica RXTX

Knjižnica RXTX je najuporabnejša knjižnica za Javo, kjer je potrebno dostopati do serijskega vmesnika. Ima skupen vmesnik za vse operacijske sisteme. Potrebna pa je dodatna knjižnica, napisana za specifični operacijski sistem, ki omogoča fizičen dostop do serijskega vmesnika. Knjižnica RXTX in navodila za namestitev sta dostopni na spletu [7].

3.2 Zbirni jezik in pravila

Zbirni jezik ima množico vnaprej znanih pravil, ki je zelo omejena in manjša, kot množica pravil v visokonivojskih jezikih. Zbirni jezik omogoča večji pregled nad izvajanjem ukazov na procesorju, kar pomeni optimalnejše in hitrejše izvajanje programa. Program se mora, za uspešno prevedbo v strojni jezik, skladati s pravili zbirnega jezika. Program, napisan v zbirnem jeziku, je sestavljen iz množice enostavnih stavkov, ti pa iz posameznih izrazov. Izrazi so sestavljeni iz simbolov. Izraz v zbirnem jeziku prikazuje primer 3.1.

Koda 3.1: Primer ukaza ADDI.

```
1       ADDI R1, R0, #1
```

Izraz je sestavljen iz sedmih simbolov. "ADDI" je simbol za operacijo seštevanja, ki sešteje vrednost drugega podanega registra in takojšnji operand. Rezultat pa shrani v prvi podan register. Simbola "R1" in "R0" ponazarjata dva splošnonamenska registra. Šesti simbol "#" ponazarja, da se za simbolom nahaja število v desetiškem zapisu. Sedmi simbol je število. Tretji in peti simbol "," sta ločili za ločevanje imen registrov in takojšnih operandov.

3.2.1 Zbirni jezik za HIP

Zbirni jezik za HIP ima manj pravil kot visokonivojski programski jeziki. Ima okrajšave za ukaz. Imena splošnonameskih registrov so vnaprej znana. Pri nekaterih izrazih je potrebno pred desetiškim številom dodati znak "#". Desetiška števila je mogoče predznačiti. Števila lahko zapišemo v šestnajstiški, osmiški in binarni obliki. Vsak izmed zadnjih treh zapisov ima pripadajočo predpono za zapis. Za šestnajstiška števila uporabimo predpono "0x", za osmiška "0", za binarni zapis pa "0b". Začetek podatkovnega dela se označi z ".data", programski del pa s ".code" ali s ".text". V podatkovnem delu so lahko napovedani različni tipi podatkov.

Podatek ali podatke se označi z imeni, na katere se lahko v programskem delu sklicujemo. Vsi tipi podatkov imajo svoje označbe, kot na primer oznaka ".byte", ".word", ".ascii", ".space" in ".align". Podatke med seboj ločujemo z vejico.

3.3 Prevajalnik

Prevajalnik je program ali množica programov, ki pretvorijo izvorni program napisan v določenem jeziku v drug jezik. V primeru HIP mora prevajalnik program, napisan v zbirnem jeziku (ang. assembler), prevesti v strojni jezik (ang. machine code). Strojni jezik je sestavljen iz množice ukazov, katere razume in izvaja procesor. Zbirni jezik je nizkonivojski jezik, precej podoben strojnemu jeziku, torej ukazom, ki jih pozna procesor HIP. Zbirni jezik uporablja okrajšave (ang. mnemonic), ki so v večini primerov neposredno povezane z ukazi v procesorju. Sestavni del zbirnega jezika so tudi direktive. To so operacije, ki jih prevajalnik uporabi pri prevajanju. Procesorju so med izvajanjem nevidni, kajti uporabijo se samo med prevajanjem. Primer direktive je ".org". Z direktivo se določi naslov, pri katerem se začne podatkovni ali ukazni sklop programa. Obstajata pa še dve pomembni direktivi ".code" ali ".text", ki označujeta začetek programskega sklopa in direktiva ".data", ki označuje začetek podatkovnega sklopa.

3.3.1 Leksikalna in sintaksna analiza

Analiza programa poteka po podobnem principu, kot človeško branje. Razdelimo jo na dva dela. Najprej je potrebno prebrati vsak znak posebej in jih nato smiselno združevati v množico znakov. Vsaka množica znakov ima lahko svoj pomen. V primeru, da so v množici samo števke, zapisani znaki predstavljajo število. Pri analizi pomena besed je potrebno definirati tako posamezne znake, kot skupine, ki se lahko pojavijo v vhodnem nizu znakov. Če se pojavi zaporedje znakov ".org", potem je to oznaka, ki

označuje napoved naslova za podatkovni ali programski del. Zaporedje znakov ".org" pa se lahko pojavi tudi pri deklaraciji niza tipa ".ascii". Pri analizi je potrebna pazljivost, da ne prihaja do napačnih interpretacij zapisanih znakov. Pravila definiranja besed v slovarju so odvisna od generatorja razčljenjevalnika. Podobni so regularnim izrazom. Ti opisujejo zaporedje znakov, ki ustrezajo iskanim vzorcem. Definicija vzorcev, ki se pojavljajo v zbirnem jeziku so definirani v posebnem delu slovnice (grammar) imenovane "lexer".

Koda 3.2: Primer definicije slovarja.

```
1    ORG : '.org';
2    WORD : '.word';
3    INT : [0] | [1-9] | [1-9][0-9]+;
4    HEX : '0x' [0-9a-fA-F]+;
5    REGISTER : ('R'|'r')([0-9]|([0-2][0-9])|([3][0-1]));
6    IDENTIFIER : [_a-zA-z][_0-9a-zA-Z]*;
```

V primeru 3.2 so definirani vzorci označeni z imeni. Prevajalnik, pri prevajanju kode iz zbirnega jezika v strojni jezik, sprva naredi leksikalno analizo. Pri tem zbira prebrane znake in ugotavlja ali se zaporedje znakov ujema z definiranimi vzorci. Pri ujemanju, se zbranim znakom doda ime pravila, torej znake se označi z značko ter se jih pošlje nivo višje v razčljenjevalnik. V primeru, da se pri prevajanju na vhodu pojavi zaporedje znakov ".org" se v razčlenjevalnik pošlje nov element z značko "ORG". Če se na vhodu pojavi zaporedje števk "{0, 1, 2..9, 10, 11..99, 100..999, 1000...}" se v razčljenjevalnik pošlje zbrane znake z značko "INT". Velika množica različnih zaporedij znakov narekuje, da zaporedja zapišemo s pravili. Razčljenjevalnik je podoben "lekserju", le da deluje na nivoju višje. Razčljenjevalnik se primarno ukvarja z značkami in s pravilnostjo zaporedja značk, ki prihajajo iz nižjega nivoja. Razčljenjevalnik je prav tako opisan s pravili. Opis pravil razčlenjevalnika je podoben, kot je opis leksikalnih pravil. 3.3 prikazuje kratek primer definicije slovnične strukture.

Koda 3.3: Primer definicije slovnice.

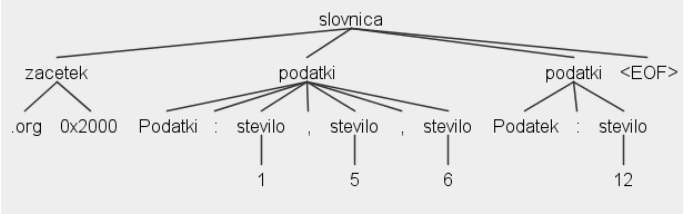
```
1    slovnica : zacetek podatki* EOF;
2    zacetek : ORG (INT | HEX);
3    podatki : IDENTIFIER ':' stevilo (',' stevilo)*;
4    stevilo : INT | HEX;
```

Za pravilnost zapisa, mora razčlenjevalnik prepoznati vse možnosti, ki se lahko pojavijo na vhodu. Značka "slovnica" je oznaka, pod katero spada pravilo "zacetek podatki* EOF", ki pomeni, da se mora na vhodu pojaviti značka "zacetek", nato nič ali več zapisov z oznako "podatki" in poseben znak, ki označuje konec na vhodu. Oznaka "zacetek" je sestavljena iz značk "ORG" in ene izmed množnosti "INT" ali "HEX". Pri znački "podatki", ki ni obvezen, lahko pa se pojavi večkrat, so potrebne značke "IDENTIFIER", ":" in "stevilo". Pri opisu pravil z oklepaji in znakom "*" ponazorimo, da se lahko za značko "stevilo" pojavi še znak "," in nova značka "stevilo". Znak "*" označuje relacijo nič proti mnogo. To pomeni, da se celoten zapis v oklepajih lahko ne pojavi ali pa se pojavi enkrat ali večkrat.

S knjižnico ANTLR je zapis leksikalnih pravil in pravil razčljenjevalnika možno navesti v skupni datoteki. Celotna slovnica iz prejšnjega primera je prikazana v 3.4.

Koda 3.4: Primer definicije jezika.

```
1    grammar HipAsmTest;
2    @header {
3    package si.fri.hip.antlr.test;
4    }
5    // parser rules
6    slovnica : zacetek podatki* EOF;
7    zacetek : ORG (INT | HEX);
8    podatki : IDENTIFIER ':' stevilo (',' stevilo)*;
9    stevilo : INT | HEX;
10   // lexer rules
11   ORG : '.org';
12   WORD : '.word';
```



Slika 3.1: Drevesna struktura vhoda.

```
13  INT : [0] | [1-9] | [1-9][0-9]+;
14  HEX : '0x' [0-9a-fA-F]+;
15  REGISTER : ('R'|'r')([0-9]|([0-2][0-9])|([3][0-1]));
16  IDENTIFIER : [_a-zA-z][_0-9a-zA-Z]*;
17
18  // one line comments
19  COMMENTS: ';'~[\r\n]* -> skip;
20  // skip spaces, tabs, newlines, \r (Windows)
21  WS      : [ \t\r\n]+ -> skip;
```

Primer kode 3.5 prikazuje pravilen zapis in razčlenjevalnik ga prepozna kot pravilnega.

Koda 3.5: Primer vhodnih podatkov.

```
1  .org 0x2000
2  Podatki: 1, 5, 6
3  Podatek: 12
```

Drevesna struktura zapisanega programa je prikazana na sliki 3.1, kjer je vidna drevesna struktura vhoda, ki jo zgradi razčlenjevalnik.

3.4 Razčlenjevalnik

Definicija slovnice za zbirnik HIP je obsežnejša, zato so tudi drevesne strukture na podlagi vhoda kompleksnejše. Še vedno pa nima tako obsežne slovnice kot visokonivojski jeziki. Program v zbirnem jeziku je sestavljen iz

enega ali več segmentov. Znak "+" za značko pomeni, da se oznaka lahko pojavi enkrat ali večkrat.

Koda 3.6: Pravila za zbirnik

```
1    program : segment+ EOF
2            ;
```

Zbirnik ima tri ločena pravila za segmente. Podatkovni segmenti, ki vsebujejo podatke, ukazni segmenti vsebujejo operacije in segment za programske pasti, kjer se nahajajo operacije za obdelavo pasti.

Koda 3.7: Pravila za zbirnik.

```
1    segment : dataSegment
2            | codeSegment
3            | trapSegment
4            ;
```

Vsi tipi segmentov se začnejo z deklaracijo tipa segmenta. V primeru segmenta za pasti, se za značko "P_TRAP" nahaja še številka pasti. Za deklaracijo tipa, se nahaja značka "P_ORG", za značko, pa celo število, ki označuje na katerem naslovu se začne segment. Pri podatkovnih segmentih se za naslovom nahajajo ukazi. Znak "*" za oznako "codeLines" pomeni, da je lahko ukazov več. Enako je pri podatkovnih segmentih.

Koda 3.8: Pravila za zbirnik

```
1    dataSegment : P_DATA P_ORG integer
2                  dataLines*
3                ;
4    codeSegment : (P_CODE | P_TEXT) P_ORG integer
5                  codeLines*
6                ;
7    trapSegment : P_TRAP integer P_ORG integer
8                  codeLines*
9                ;
```

Deklaracijo podatkov je možno podati na tri načine. Prvi način je podajanje z oznako. “LABEL” je oznaka na katero se lahko sklicujemo v programu. Po oznaki “LABEL” se nahaja znak “:“, nato pa “dataType”, ki napove tip podatkov, ki so v nadaljevanju navedeni. Drugi način deklaracije je možen pri tipu podatka “.space”. Oznaki “LABEL” in “:“, pri tem tipu podatka, nista obvezni. Potrebno je deklarirati količino rezerviranega prostora v bajtih. Tretji način deklaracije je namenjen poravnavanju podatkov. Pod oznako “integer” navedemo pozitivno celo število, prevajalnik pa mora poravnati naslednji podatek v pomnilniku. Poravnava podatka poteka tako, da se naslednji deklarirani podatek poravna na naslov, ki je izračunan po formuli (3.1), kjer je “y” izračunan naslov, “x” je trenutni naslov in “i” je celoštevilska vednost.

$$y = x - (x \ mod \ i) + i \tag{3.1}$$

Koda 3.9: Pravila za zbirnik.

```
1    dataLines    :
2            LABEL ':' dataType          # labelDataLine
3        |   (LABEL ':')? dataTypeSpace
4            integer                     # spaceDataLine
5        |   dataTypeAlign integer       # alignDataLine
6        ;
```

V zbirniku je mogoče napovedati več tipov podatkov, kot je prikazano v kodi 3.10. Največkrat uporabljen je celoštevilski tip. Celoštevilskih tipov “dataTypeInt” poznamo več vrst. “dataTypeDouble” označuje tip podatkov zapisan v plavajoči vejici. Z oznako “dataTypeString” se označuje nize znakov. “STRING” je značka za niz. Zbirnik omogoča napoved referenc. Referenca je tip podatka, ki vsebuje naslov drugega podatka. Z oznako “P_WORD” se napove 4-bajtni tip podatka.

Koda 3.10: Pravila za zbirnik.

```
1    dataType     : dataTypeInt dataInt
```

```
2                   | dataTypeDouble dataDouble
3                   | dataTypeString STRING
4                   | P_WORD dataAddress;
```

Celoštevilčni tipi so označeni z oznakami, prikazanimi v 3.11. Podatke je potrebno označiti s tipom, da prevajalnik rezervira pravilno količino prostora v pomnilniku. Za tip "P_WORD" prevajalnik rezervira 4 bajte, za "P_WORD16" pa rezervira 2 bajta.

Koda 3.11: Celoštevilčni tipi.

```
1    dataTypeInt : P_WORD
2                | P_BYTE
3                | P_WORD64
4                | P_WORD16
5                ;
```

Podatkovna pravila so zapisana v primeru 3.13. Mogoče je navesti en podatek ali več podatkov. Če je podatkov več, se jih ločuje z vejico. Oznaka "integer" označuje celoštevilčen podatek. "(',' integer)*" označuje možnost navajanja več naslednjih podatkov ločenih z vejico.

Koda 3.12: Pravila za zbirnik.

```
1    dataInt     : integer (',' integer)*;
2    dataDouble  : doubleNum (',' doubleNum)*;
3    dataAddress: LABEL (',' LABEL)*;
```

Celoštevilski tip je mogoče zapisati v štirih številskih sistemih. V desetiškem sistemu, je mogoče število zapisati s predznakom "+" ali "-". Ostali sistemi so šestnajstiški, osmiški in binarni. Zapisi imajo svojo predpono. Šestnajstiški zapis ima predpono "0X" ali "0x", osmiški zapis ima "0". Predpona za dvojiški zapis je "0b" ali "0B".

Koda 3.13: Pravila za zapis celih števil.

```
1    integer : '+'? decimal
2            | '-' decimal
3            | atom ;
```

```
4     atom     : hex
5              | octal
6              | binary ;
```

Segmenti za kodo so sestavljeni iz ukazov v posamezni vrstici. Pred vsakim ukazom je mogoče zapisati oznako "(LABEL ':')?". Uporabimo jo takrat, ko želimo uporabiti sklicevanje na oznako v programu. V večini primerov se oznake uporablja pri sklicevanju pri skočnih ukazih.

Koda 3.14: Pravila za zapis ukazov.

```
1     codeLines    : (LABEL ':')? code ;
```

HIP ima več različnih tipov ukazov, ki so navedeni v primeru 3.15.

Koda 3.15: Pravila za zapis ukazov.

```
1     code         : store
2                  | load
3                  | ale
4                  | control
5                  | system
6                  | nop
7                  | halt ;
```

Vsak ukaz za shranjevanje se začne z operacijsko kodo označeno z oznako "STORE_OPCODE". Oznaka "expression" označuje izraz. Ta je lahko enostavno seštevanje dveh konstant. Prva oznaka "register" označuje ime registra, ki predstavlja bazo naslova, druga oznaka pa označuje ime registra, katerega vrednost se bo shranila v predpomnilnik.

Koda 3.16: Pravila za zapis ukaza za shranjevanje v predpomnilnik.

```
1     store: STORE_OPCODE expression
2              '(' register ')' ',' register
3          ;
```

Ukaz za branje iz pomnilnika je podoben ukazu za shranjevanje. Razlika je le, da prva oznaka "register" označuje ime registra, v katerega se bo zapisal

podatek, prebran iz predpomnilnika. Druga oznaka "register" označuje ime registra, v katerem je shranjen bazni naslov.

Koda 3.17: Pravila za zapis ukaza za branje iz pomnilnika.

```
1    load : LOAD_OPCODE register ','
2             expression '(' register ')'
3         ;
```

Aritmetične logične operacije delimo na na tri tipe. Prvi tip je zapis operacije za nalaganje dveh bajtov v zgornji del ponornega registra. Zgornji biti so od 31 do 16. Prvi parameter je okrajšava za operacijo, drugi parameter je ime registra, v katerega se bosta v zgornji del shranila dva bajta. Za vejico je mogoče zapisati celo število ali ime oznake s katerim se označi podatek v podatkovnem segmentu. Drugi način je zapis operacij, ki se jih zapisuje v formatu 1. Te operacije uporabljajo takojšni operand. Operacije v formatu 1 uporabljajo dva registra. Prvi register je ime registra v katerega se shrani rezultat operacije, drugi register je operand, ki se ga uporabi pri operaciji. Drugi operand je takojšni operand, ki je podan s celim številom ali z referenco na podatek. Tretji način zapisa je zapis v formatu 2. Prvi register je ime registra, v katerega se shrani rezultat. Druga dva registra sta vhodna registra, ki se ju uporabi pri operaciji.

Koda 3.18: Pravila za zapis ukaza za branje iz pomnilnika.

```
1    ale : ALE_OPCODE_LHI register ','
2                  '#'? (integer | LABEL)
3        | ALE_OPCODE_IMMEDIATE register ',' register ','
4                  '#'? (integer | LABEL)
5        | ALE_OPCODE register ','
6                  register ',' register
7        ;
```

Poznamo več vrst kontrolnih ukazov. Najpogostejši so pogojni skoki z oznako "BRANCH_OPCODE". Sestavljeni so iz imena operacije, nato je zapisana oznaka registra, katerega vrednost se primerja ali je pogoj izpolnjen

ter ime oznake, na katero se sklicujemo v programu v primeru izpolnjenega pogoja. Ime oznake, se pri prevajanju pretvori v naslov, nato pa se izračuna relativni odmik med naslovom, ki sledi oznaki in naslovom trenutnega ukaza. Z "JUMP_OPCODE" se označuje brezpogojne skoke. Za imenom operacije se nahaja ime oznake ali celo število. V oklepajih je ime registra, ki predstavlja bazni naslov. "CALL_OPCODE" je tip ukaza, podoben prejšnjemu. Na prvem mestu, po imenu operacije, je ime registra, v katerega se shrani naslov naslednjega ukaza. "TRAP_OPCODE" je operacija za programsko sprožanje pasti. Za imenom operacije se nahaja znak "#" in celo število.

Koda 3.19: Pravila za zapis kontrolnih ukazov.

```
1    control :
2        BRANCH_OPCODE register ',' LABEL
3        | JUMP_OPCODE (LABEL | integer) '(' register ')'
4        | CALL_OPCODE register ',' (LABEL | integer)
5            '(' register ')'
6        | TRAP_OPCODE '#' integer
7        | RFE_OPCODE
8        ;
```

"INTERUPT_OPCODE" je namenjen omogočanju in onemogočanju prekinitev. Ukaz je sestavljen samo iz imena ukaza. "MOV_OPCODE" oznaka predstavlja ukaz za prenos vsebine podanega registra v register EPC ali obratno.

Koda 3.20: Pravila za zapis sistemskih ukazov.

```
1    system   : INTERUPT_OPCODE
2             | MOV_OPCODE register
3             ;
```

Pravilo za zapis celih števil je podan v 3.21. Celo število je možno zapisati z neobveznim predznakom "+" ali "-".

Koda 3.21: Pravila za zapis celih števil.

```
1    integer : '+'? decimal
```

```
2               | '-' decimal
3               | atom
4               ;
```

Pod značko “atom” spadajo pravila za zapis celih števil v šestnajstiškem, osmiškem in binarnem številskem sistemu.

Koda 3.22: Pravila za zapis celih števil.

```
1     atom      : hex
2               | octal
3               | binary
4               ;
```

Poglavje 4

Grafični uporabniški vmesnik

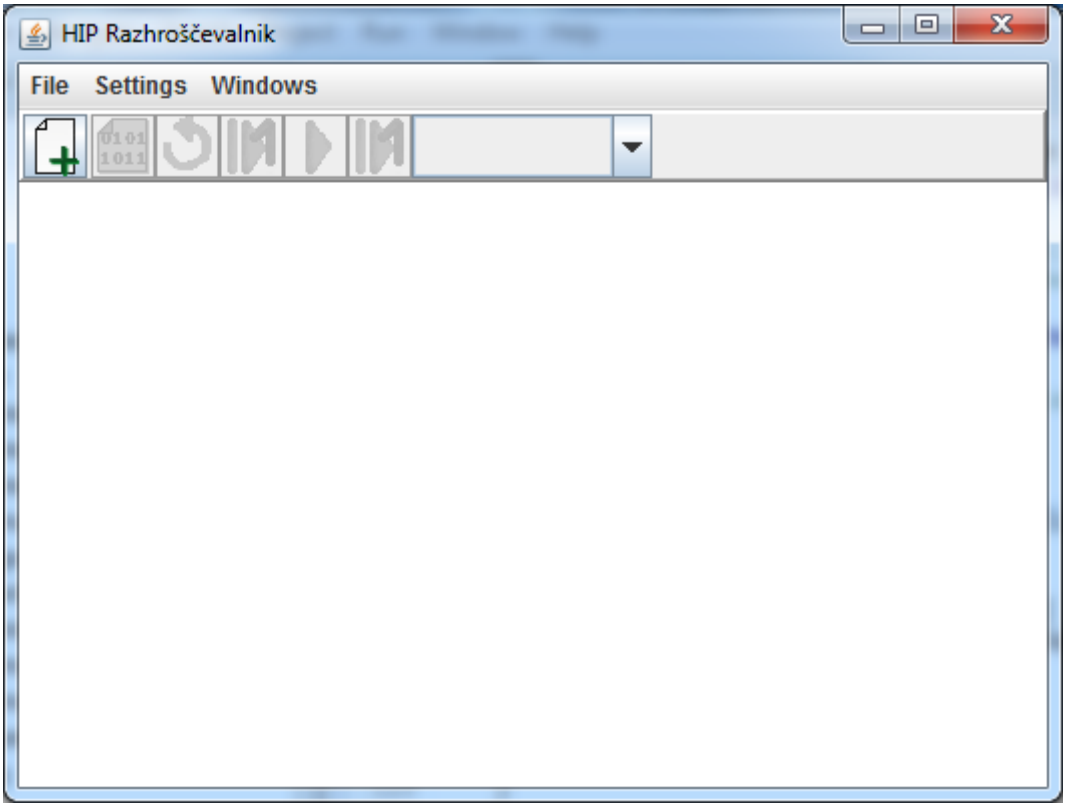
Grafični uporabniški vmesnik povezuje uporabnika in razhroščevalnik. Napisan je v programskem jeziku Java. Uporabniku omogoča lažje upravljanje z razhroščevalnikom. Program napisan v zbirniku za HIP prevede. Nato pa preko serijske povezave komunicira z razhroščevalno enoto v FPGA čipu. Program nudi enostaven urejevalnik teksta za pisanje programa v zbirnem jeziku. Ima osnovne nastavitve za povezavo preko serijskega vmesnika.

Pri zagonu programa se prikaže osnovno okno, kot je prikazano na sliki 4.1.

V meniju "File" z izbiro "Open File", odpremo datoteko, ki vsebuje program napisan v zbirnem jeziku za HIP. Program je možno popravljati v odprtem oknu.

4.1 Nastavitve

Razhroščevalnik za HIP nudi osnovne nastavitve za serijski vmesnik, predpomnilnik in način izvajanja kode v HIP. Nastavitve za serijski vmesnik pridejo v poštev pri povezovanju razhroščevalnika z razvojno ploščico. Nastavitve za serijski vmesnik so nastavljene tako, da je potrebno le izbrati ime ustreznega serijskega vmesnika iz padajočega seznama.



Slika 4.1: Glavno okno HIP Razhroščevalnika.

Nastavitve predpomnilnika omogočajo, da se omogoči predpomnilnik v HIP na razvojni ploščici. Na razvojni ploščici sta implementirana direktna predpomnilnika s štirimi bloki po 32 besed.

Pri izvajanju ukazov v HIP je privzeto ugotavljanje podatkovnih nevarnosti. Ugotavljanje podatkovnih nevarnosti je mogoče onemogočiti tako, da se v "Settings" → "HIP Settings" omogoči nastavitev "Brez detekcije pod. nev.". Možno je tudi omogočiti skočne reže pri zakasnjenih skokih.

Enoto za omogočanje premoščanja je mogoče vklopiti z nastavitvijo "Settings" → "HIP Settings" → "Omogoči premoščanje". Skočne reže pri HIP je prav tako mogoče omogočiti s "Settings" → "HIP Settings" → "Zakasnjeni skoki".

4.2 Okna

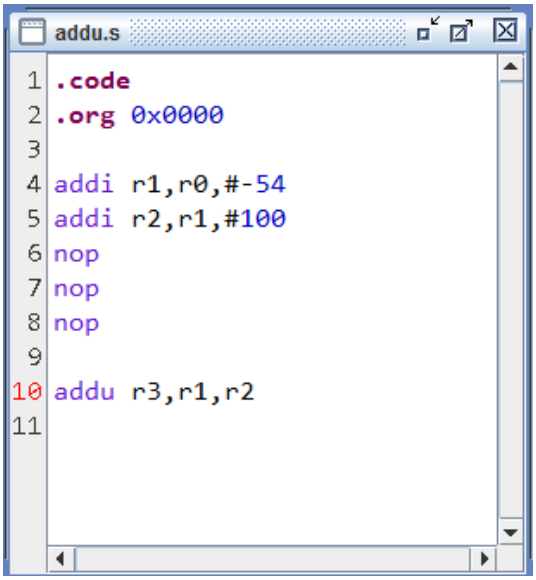
Po prevajanju programa se odprejo dodatna okna, ki uporabniku prikažejo vsebino posameznih registrov, podatke v glavnem pomnilniku, predpomnilnikih in statistiko.

4.2.1 Urejevalnik kode

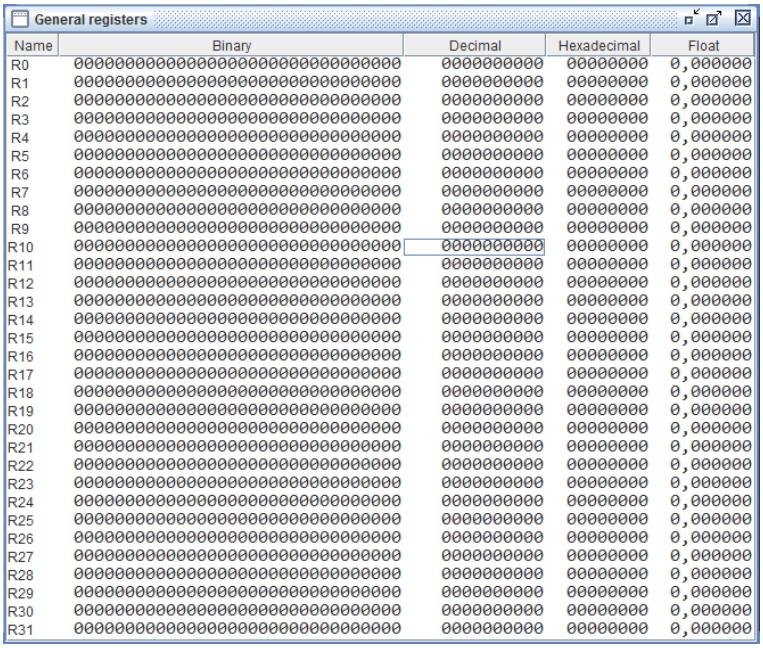
Urejevalnik kode je namenjen pisanju in popravljanju kode. Okno se odpre z gumbom "New file"  . Odpreti je možno tudi datoteko s programom. Rezervirane besede, kot so ukazi in registri se obarvajo z drugo barvo. Napisan program se prevede s klikom na gumb "Compile"  . Primer kode v urejevalniku je prikazan na sliki 4.2.

4.2.2 Splošnonamenski registri

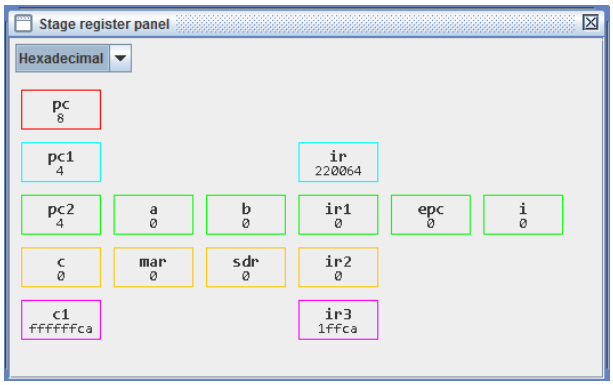
Okno prikazuje vsebino splošnonamenskih registrov med izvajanjem programa. Vsebina registrov je prikazana v dvojiškem, desetiškem in šestnajstiškem zapisu. Dodan je zapis v plavajoči vejici, kot je prikazano na sliki 4.3.



Slika 4.2: Urejevalnik kode.



Slika 4.3: Okno splošnonamenskih registrov.



Slika 4.4: Vmesni registri v cevovodu.

4.2.3 Vmesni registri v cevovodu

Okno prikazuje vsebino vmesnih registrov v HIP med izvajanjem programa. Okno vmesnih registrov je prikazano na sliki 4.4. Vsebina registrov se osveži vsako izvedeno urino periodo.

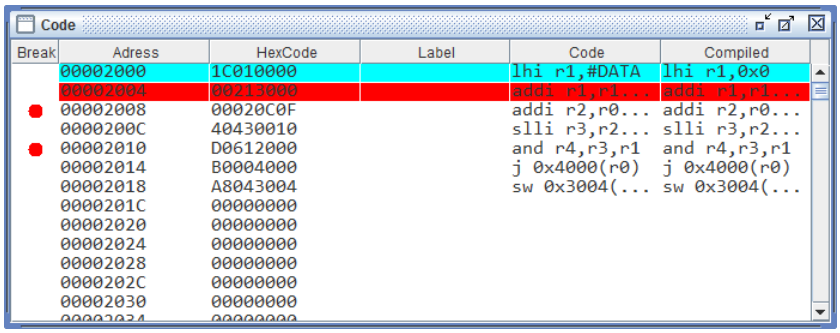
4.2.4 Koda z naslovi

Prevedena koda je vidna v oknu, kot je prikazano na sliki 4.5. V prvem stolpcu je mogoče nastaviti ustavitvene točke. Pri ustavitvenih točkah se program ustavi. Drugi stolpec vsebuje naslove. V tretjem stolpcu so operacije zapisane v šestnajstiški obliki. Četrti stolpec vsebuje imena skočnih naslovov, če ti obstajajo. Peti stolpec vsebuje napisano kodo, šesti pa prevedeno kodo, kjer so številke formatirane v šestnajstiški obliki.

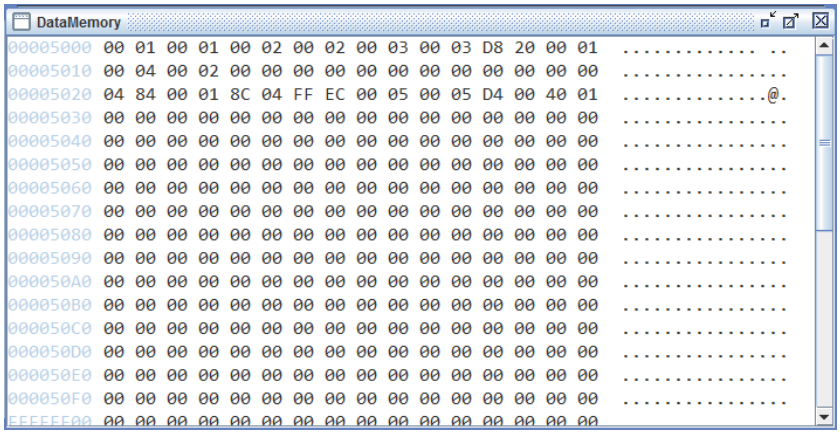
Med izvajanjem kode, se vrstice obarvajo z barvami. Barve so odvisne od tega, v kateri stopnji cevovoda se posamezni ukaz nahaja.

4.2.5 Glavni pomnilnik

Po prevajanju se strojna koda in podatki nahajajo v glavnem pomnilniku v razhroščevalniku. Primer podatkov v glavnem pomnilniku je prikazan na sliki 4.6. Na levi strani okna so naslovi. Na sredini so prikazani podatki



Slika 4.5: Okno z naslovi, prevedenimi in napisanimi ukazi.

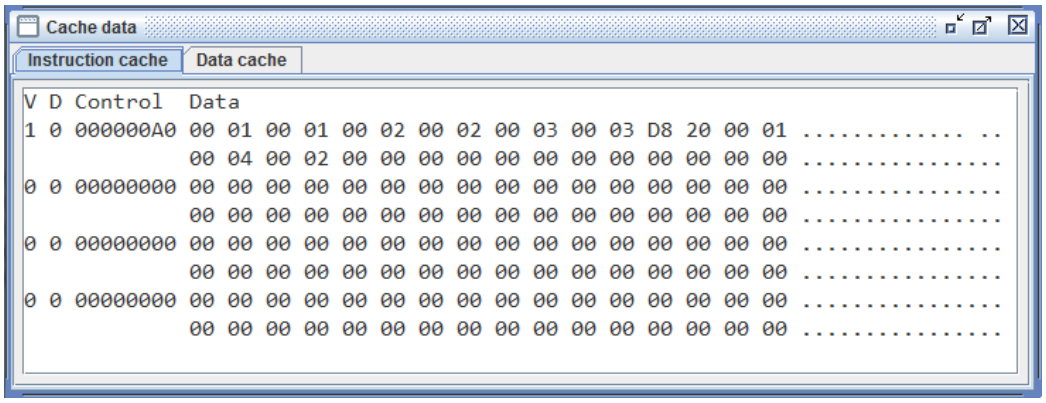


Slika 4.6: Okno s podatki v glavnem pomnilniku-

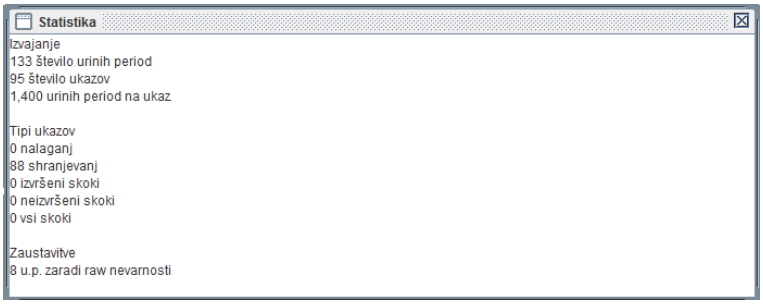
in operacije v šestnajstiški obliki v naraščajočem vrstnem redu po naslovih. Na desni strani so prikazani podatki v znakovni obliki. Podatki in ukazi so zapisani v blokih po 256 bajtov. Na spodnjem delu seznama se nahaja tudi 64 32-bitnih naslovov. Ti naslovi so rezervirani za skočne naslove namenjene za pasti.

4.2.6 Ukazni in podatkovni predpomnilnik

Vsebina podatkovnega in ukaznega predpomnilnika je prikazana v oknu, ki je prikazano na sliki 4.7. Vsak predpomnilnik ima svoj zavihek. Prikazani



Slika 4.7: Okno s podatki ali ukazi v blokih predpomnilnika.

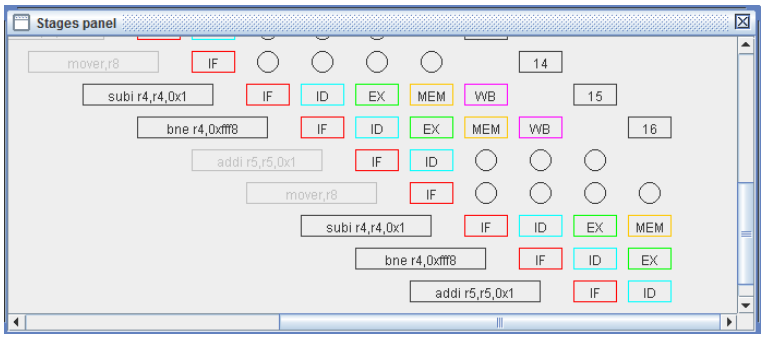


Slika 4.8: Okno za statistiko.

so podatki v posameznem bloku. Vsakemu bloku pripada tudi umazan in veljavni bit ter trenutni naslov bloka. Okno se prikaže po prevajanju v primeru omogočenega predpomnilnika. Vsebina posameznega bloka se spremeni pri pisanju v predpomnilniški blok.

4.2.7 Statistika

Pri izvajanju programa se beleži statistika. Izvaja se štetje urinih period, število ukazov, število branj in pisanj. Šteje se vrste skokov in povprečno število urinih period, potrebnih za en ukaz. Primer statistike je prikazan na sliki 4.8.



Slika 4.9: Okno za prikaz izvajanja programa po urinih periodah.

4.2.8 Izvajanje ukazov po stopnjah v cevovodu

Okno med izvajanjem prikazuje ukaze, ki so bili in so v izvajanju po posameznih stopnjah v dani urini periodi. Slika 4.9 prikazuje primer izvajanja programa. Na sliki so vidni ukazi, ki so bili razveljavljeni zaradi skokov ter mehurčki, ki so bili vstavljeni zaradi reševanja cevovodnih nevarnosti. V primeru omogočenega predpomnilnika in predpomnilniških zgrešitev so tudi prikazane čakalne urine periode.

4.3 Izvajanje programa

Program je potrebno odpreti in nato prevesti. Program mora biti sintaktično pravilen, drugače razhroščevalnik obvesti uporabnika na napako. Pravilno preveden program lahko poganjamo s tipko  . Program napreduje za eno urino periodo. Bližnjica za poganjanje programa za eno urino periodo je tipka “F5”. Program je možno s tipko “reset”  izvajati od začetka. Z gumbom  se program izvaja do prekinitve. Prekinitev izvajanja se sproži s pritiskom na gumb  . V primeru, ko pride register PC do prekinitvene točke v oknu 4.2.4, program ustavi izvajanje. V primeru več programskih segmentov je na voljo izbira naslova, pri katerem se začne izvajati program. Začetni naslov je potrebno izbrati pred začetkom izvajanja programa. V nasprotnem primeru je potrebno program pognati znova.

Poglavje 5

Sklepne ugotovitve

Cilj diplomske naloge je implementirati HIP, predpomnilnik, enoto za seštevanje in odštevanje v plavajoči vejici in razhroščevalno enoto z UART modulom za komuniciranje z zunanjo napravo.

Zunanja naprava je osebni računalnik, ki ima lahko različne operacijske sisteme. Prevajalnik in razhroščevalnik sledita zahtevi po prenosljivosti. Pri implementaciji HIP sem uporabil znana orodja, kot so Xilinx Design Tools, ki olajša programiranje z VHDL jezikom. Napisan model HIP s predpomnilnikom in podpornim vezjem za razhroščevanje je bilo potrebno prevesti v obliko, ki se naloži na FPGA čip. Vezje obdeluje ukaze preko UART modula in pošilja odgovore nazaj v zunanjo napravo. V zunanji napravi z izbranim operacijskim sistemom je bilo potrebno implementirati prevajalnik za zbirni jezik. Prevajalnik iz zbirnega jezika v strojni jezik, je bil napisan v programskem jeziku Java z uporabo ANTLR knjižnice, ki je tvorila ogrodje razčljenjevalnika. Preveden program se pošlje v izvajanje v HIP na FPGA čip. Programiranje je olajšano z grafičnim vmesnikom, med izvajanjem programa na HIP pa se vidi vsebina splošnonamenskih, vmesnih in ostalih registrov. Grafični vmesnik prikazuje prevedeno kodo v šestnajstiškem zapisu, kjer je možno spremljati spremembe, ki se dogajajo s shranjevalnimi ukazi.

Poseben izziv je bilo implementirati branje in pisanje v predpomnilnik

preko razhroščevalne enote. Pri omogočenemu predpomnilniku, grafični vmesnik rahroščevalnika prikaže spremembe v blokih, ki jih naredi program v HIP. Prikazuje tudi vse podatke, ki so pomembni za izvajanje prevedenega programa. Pri implementaciji enot za operacije v plavajoči vejici sem ugotovil, da zavzamejo veliko prostora na čipu FPGA. Vse operacije v plavajoči vejici je bilo potrebno implementirati v več korakih, čeprav se v današnjih namiznih procesorjih seštevanje in odštevanje izvede v enem ciklu. Podobno je z množenjem in deljenjem. Na račun zmanjšane hitrosti je možno izvajati tudi množenje in deljenje. Ker je zasedenost čipa več kot 80 odstotna, se lahko zgodi, da enota za deljenje ne deluje pravilno. Čeprav je bila pravilnost delovanja preizkušena v ISim simulatorju. Potrebno je bilo zmanjšati frekvenco ure, s katero deluje procesor. Zunanjo uro se deli z dve, kar pomeni, da je maksimalna frekvenca delovanja procesorja 25 MHz.

Procesorji se vedno bolj izpopolnjujejo. Tudi HIP se lahko dodatno izpopolni. Enostavna možnost je povečanje predpomnilnika. Druga dopolnitev bi bila implementacija skočnega predpomnilnika. Tretja dopolnitev bi bila implementacija strojnih prekinitev. Zanimiva bi bila tudi implementacija krmilnika za VGA monitor, kjer bi lahko s programom pisali v del pomnilnika, ki bi bil namenjen prikazovanju slike. Pri HIP bi bilo dobro narediti še celoštevilsko množenje in deljenje. Možnosti za izboljšave je veliko, te pa bo, verjamem, uresničil še kdo drug.

Literatura

[1] D. Kodek, *Arhitektura in organizacija računalniških sistemov*, Založba Bi-Tim, d.o.o., 2008

[2] *ISE In-Depth Tutorial* [Online]. Dosegljivo: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ise_tutorial_ug695.pdf [Dostopano 15. 8. 2016]

[3] *Spartan-3E FPGA Starter Kit Board User Guide* [Online]. Dosegljivo: http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf [Dostopano 15. 8. 2016]

[4] Chu, Pong P., *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*, John Wiley Sons, Inc., Hoboken, New Jersey, 2008, ISBN-13: 9780470185315

[5] Članek: Vojin G. Oklobdzija *An Algorithmic and Novel Design of a Leadding Zero Detector Circuit : Comparison with Logic Synthesis*

[6] T.-J. Kwon, J. Sondeen, J. Draper *Floating-Point Division and Square Root using a Taylor-Series Expansion Algorithm* [Online]. Dosegljivo: http://www.isi.edu/~draper/papers/mwscas07_kwon.pdf [Dostopano 15. 8. 2016]

[7] RXTX Wiki [Online]. Dosegljivo: http://rxtx.qbang.org/wiki/index.php [Dostopano 29. 7. 2016].

[8] T. Parr, *The definitive ANTLR 4 Reference*, The Pragmatic Programmers, LLC., 2013, ISBN-13: 978-1-93435-699-9