Izvedba sustava sa FRISC-V procesorom

Kelković, Petra

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1258

IZVEDBA SUSTAVA SA FRISC-V PROCESOROM

Petra Kelković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1258

IZVEDBA SUSTAVA SA FRISC-V PROCESOROM

Petra Kelković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1258

Pristupnica:	**Petra Kelković (0036539562)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Mario Kovač

Zadatak: **Izvedba sustava sa FRISC-V procesorom**

Opis zadatka:

Proučiti RV32I skup aritmetičko logičkih naredaba za RISC-V arhitekturu procesora. Projektirati i provjeriti izvedbu FRISC-V procesora s naglaskom na protočnu arhitekturu. Izvesti i ispitati sustav sa FRISC-V u XILINX FPGA sklopovima te povezati sa pločicom sa ulazno izlaznim sklopovima te izvesti primjere upravljanja ulazno izlaznim sklopovima od strane FRISC-V procesora.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

Uvod

U okviru predmeta Arhitektura računala 1 na Fakultetu elektrotehnike i računarstva razvija se RISC-V procesor, nazvan FRISC-V [1]. Procesor ima 32-bitnu jezgru i podržava izvođenje naredbi za obradu cjelobrojnih podataka (preciznije, podržava podskup naredbi RV32I [2] skupa naredaba RISC-V ISA). Implementiran je kao procesor s mekom jezgrom, korištenjem jezika za opis digitalnog sklopovlja SystemVerilog i Xilinx programskih alata.

Cilj ovog rada bio je unaprijediti spomenuti procesor kako bi se mogao sintetizirati za rad na razvojnoj FPGA pločici PYNQ-Z2 [3]. Daljnji cilj bio je proširiti dizajn procesora kako bi se omogućilo upravljanje ulazno-izlaznim pinovima na samoj pločici na koju je smješten procesor te na pločici periferija LPCXpresso Base Bord [4].

Za postizanje navedenog cilja bilo je potrebno proučiti RV321 skup aritmetičko-logičkih naredaba te način rada protočne strukture sa pet razina, a zatim se detaljno upoznati sa dotad izvedenom inačicom procesora. Stoga u prvo poglavlje stavljam opis skupa RV321, a u drugo opis rada protočne strukture i hazarda koji pritom mogu nastati. U treće poglavlje stavljam opis funkcionalnog dijela inicijalno izvedenog sustava te zahtjeve na proširenje i unaprjeđenje sustava.

U četvrto poglavlje rada stavljam izvedbu proširenja sustava. U njemu se opisuje implementacija novog upravljačkog modula za FRISC-V procesor, optimizacija memorijskog modula, razvoj modula za signalizaciju kraja programa, kao i razvoj modula 8-bitnog izlaznog registra te njihova integracija s postojećim sustavom.

Peto poglavlje stavlja fokus na aplikacije koje su razvijene u svrhu testiranja i verifikacije sustava. Uz to, navedeni su rezultati verifikacije.

1. Skup naredbi RV32I

RISC-V arhitektura seta naredaba (ISA – eng. Instruction Set Architecture) jest javno dostupna (eng. open source) arhitektura temeljena na načelima RISC arhitekture. Skraćenica RISC dolazi od Reduced Instruction Set Computer i odnosi se na vrstu arhitekture s obzirom na skup naredaba. RISC arhitektura sadrži manji set jednostavnih naredbi koje se mogu izvoditi vrlo brzo – za neke jednostavnije naredbe dovoljan je 1 ciklus da se izvedu. Razvoj RISC arhitekture započeo je 80-ih godina, dok je prije toga na tržištu vladala dominacija Complex Instruction Set Computer (CISC) arhitekture. RISC arhitektura brzo je stekla popularnost zahvaljujući boljim performansama i jeftinijoj izvedbi u odnosu na CISC arhitekturu. Na današnjem tržištu vlada dominacija RISC arhitekture i većina modernih procesora ima upravo tu arhitekturu.

Osnovna RISC-V ISA sadrži naredbe za obradu cjelobrojnih podataka. Postoji nekoliko verzija RISC-V ISA, a ovise o širini registara procesora te širini korisničkog adresnog prostora. Dvije osnovne verzije su RV32I i RV64I, koje rade sa širinama od 32, odnosno 64 bita. Slovo I na kraju svake kratice označava rad sa cjelobrojnim podatcima, a podrazumijeva operacije čitanja i pohrane cjelobrojnih vrijednosti, osnovne aritmetičko-logičke operacije te operacije za kontrolu toka programa.

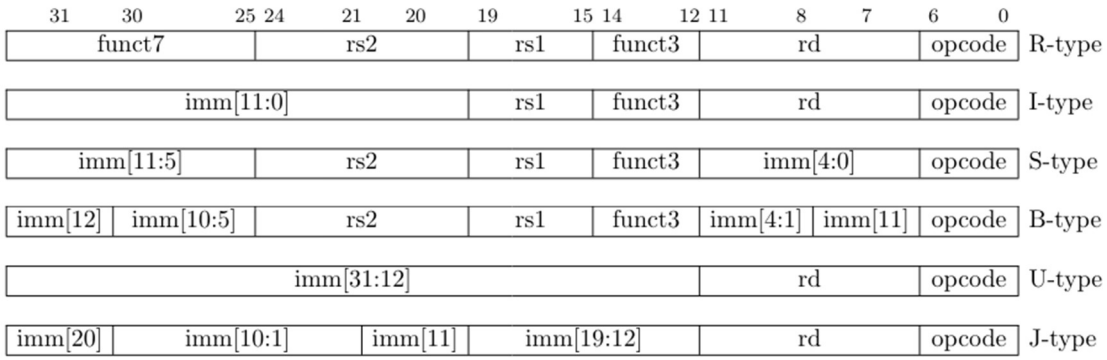
Uz osnovne verzije, RISC-V ISA može se nadograditi brojnim standardnim i ne standardnim proširenjima. Primjeri standardnih proširenja su proširenje za množenje i dijeljenje (M), proširenje za atomarne operacije (A), proširenje za rad s pomičnim zarezom jednostruke (F) i dvostruke (D) preciznosti i mnogi drugi. Dakle, izbor ove arhitekture naredaba daje mnoge mogućnosti za jednostavnu i efikasnu nadogradnju procesora u budućnosti.

RISC-V ISA postavlja sljedeće zahtjeve na arhitekturu procesora:

- Procesor treba sadržavati 31 registar opće namjene
- Procesor treba sadržavati registar x0 koji je ožičen tako da je njegova vrijednost uvijek u logičkoj 0
- Procesor treba sadržavati poseban registar namijenjen čuvanju adrese naredbe koja je u izvođenju (eng. program counter, PC)

RV32I inačica sastoji se od 47 osnovnih naredaba, a svaka od njih pripada jednom od četiri osnovna tipa: R-tipu, I-tipu, S-tipu ili U-tipu. FRISC-V procesor koristi prošireni način

kodiranja naredbi u kojem se uz osnovne tipove javljaju još i J-tip i B-tip. B-tip razlikuje se od S-tipa samo u raspodijeli bitova neposredne vrijednosti, isto kao i J-tip od U-tipa.



Slika 1.1. Prikaz strukturi naredaba ovisno o tipu kojem pripada

Na slici 1.1 prikazane su strukture svih 6 tipova naredaba koje koristi FRISC-V. Strukturu R-tipa imaju aritmetičko-logičke naredbe čija su oba operanda registri (rs1 i rs2). I-tipom su kodirane aritmetičko-logičke naredbe čiji su operandi registar i neposredna vrijednost, a ovim tipom su dodatno kodirane i naredba za učitavanje vrijednosti iz memorije LOAD te naredba za skok koja povratnu adresu pohranjuje u registar povratne adrese (LR, Link Register) JALR. Strukturu S-tipa ima naredba za pohranu vrijednosti registra u memoriju, STORE. Naredbe za uvjetne skokove imaju strukturu B-tipa. U-tip kodira naredbe LUI i AUIPC, koje se koriste za učitavanje viših 20 bita u zadani registar. J-tip kodira naredbu bezuvjetnog skoka JAL.

Od ukupno 47 osnovnih naredaba skupa RV32I, procesorska jezgra podržava 37 naredaba za obavljanje aritmetičko-logičkih operacija, upravljanje tokom programa te spremanje i čitanje cjelobrojnih podataka iz memorije. Procesorska jezgra nema implementirane sljedeće naredbe: naredbe za upravljanjem pristupu memoriji *FENCE* i *FENCE.I*, naredbe za upravljanje statusnim registrom (eng. Control Status Register, CSR) *CSRRW, CSRRS*, *CSRRC, CSRRWI, CSRRSI i CSRRCI,* naredbu za ispitivanje *EBREAK* te naredbu za slanje zahtjeva operacijskom sustavu ECALL.

2. Način rada protočne strukture

Protočna struktura ili cjevovod (eng. pipeline) jest naziv za organizaciju izvođenja naredbi u kojoj se proces izvođenja naredbe rastavlja na nekoliko jednostavnijih koraka, pogodnih paralelizmu. Ti koraci se nazivaju faze, a svaku fazu izvodi jedna od razina protočne strukture. Protočna struktura počela se koristiti pojavom procesora RISC arhitekture, koja je tražila načine kako ubrzati rad procesora koristeći sredstva s kojima već raspolaže. Prije pojave protočne strukture, koristila se slijedna organizacija izvođenja naredbi, koja izvede naredbu u potpunosti, a zatim kreće na novu. Glavni cilj protočne strukture bio je ubrzati izvođenje pojedine naredbe, točnije smanjiti broja ciklusa potrebnih za izvođenje jedne naredbe.

2.1. Opis protočne strukture

Za efikasan rad protočne strukture, potrebno ju je pažljivo podijeliti na razine koje mogu izvoditi paralelne poslove. S obzirom na to da svaka naredba mora proći kroz svaku razinu protočne strukture, možemo zaključiti da brzina rada protočne strukture ovisi o brzini rada njezine najsporije faze. Stoga bi za efikasan rad trebalo podijeliti protočnu strukturu na razine otprilike jednakog trajanja, što je i napravljeno kod FRISC-V protočne arhitekture.

FRISC-V protočna arhitektura podijeljena je na 5 razina:

- Razina dohvata naredbe (eng. Instruction Fetch, IF)
- Razina dekodiranja naredbe (eng. Instruction Decode, ID)
- Razina izvođenja naredbe (eng. Execution, EXE)
- Razina za pristup memoriji (eng. Memory, MEM)
- Razina upisa u registre (eng. Writeback, WB)

Razine se izvode redoslijedom kojim su navedene.

Razina dohvata naredbe upravlja programskim brojilom, registrom koji prati na kojoj je adresi naredba koja iduća ide u izvođenje. U ovoj fazi također dolazi do pristupa instrukcijskom dijelu memorije te se iz njega dohvaća naredba čija je adresa u programskom brojilu. S obzirom na to da su naredbe široke 32 bita, što znači da su spremljene na adresama u memoriji koje su djeljive s 4, programsko brojilo se u svakom ciklusu poveća za 4. Osim

te vrijednosti, programsko brojilo može poprimiti i vrijednost adrese skoka koju mu prosljeđuje razina izvođenja naredbe. Do toga dolazi ukoliko prilikom izvođenja naredbe skoka bude zadovoljen uvjet skoka. Ova razina prosljeđuje vrijednost programskog brojila i dohvaćenu naredbu sljedećoj razini, razini dekodiranja.

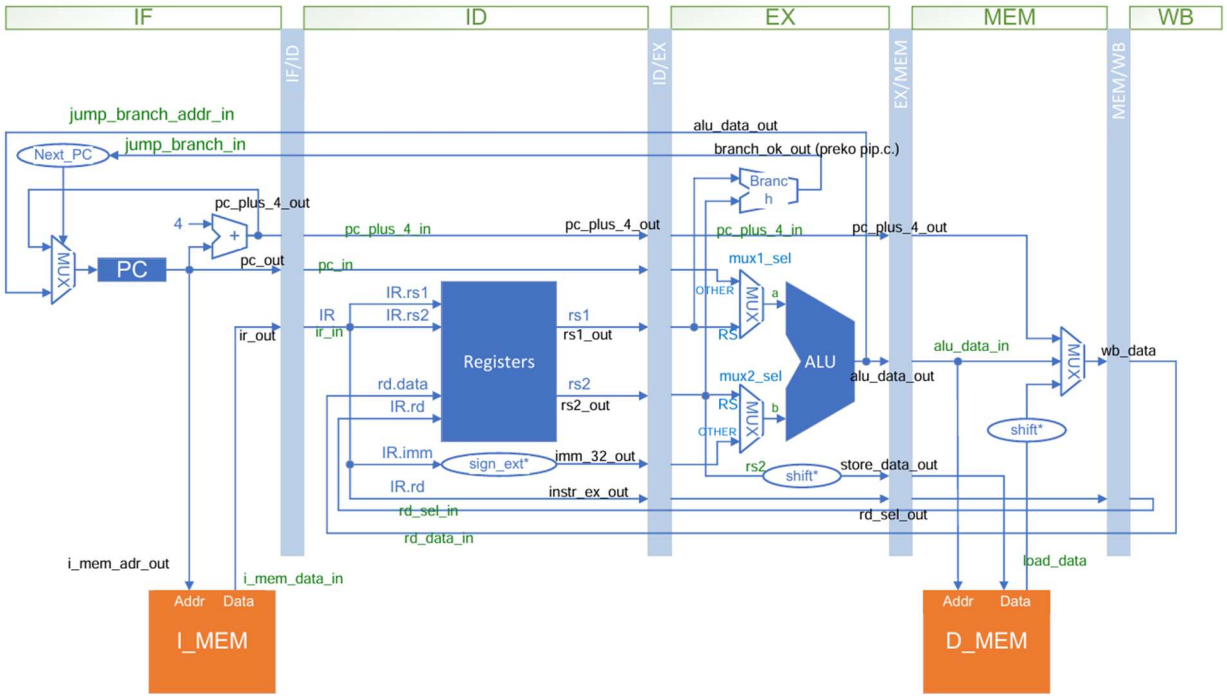
Razina dekodiranja naredbe, kao što joj i sam naziv kaže, upravlja dekodiranjem naredbe. Iz strojnog koda naredbe ona izvlači sve podatke potrebne za njeno izvođenje – o kojoj naredbi je riječ, što su operandi, koristi li se neposredna vrijednost i koja je njezina vrijednost te na koju adresu ili u koji registar se treba spremiti rezultat naredbe. Dekodiranje se temelji na prepoznavanju tipa naredbe (opisani u 1. poglavlju) i rastavljanju strojnog koda na dijelove od kojih je sastavljen. Osim dekodiranja, ova razina je također zadužena za čuvanje registara opće namjene i rukovanje s njima. Na temelju strojnog koda, ona saznaje koji registri su operandi i čita njihovu vrijednost, koja se prosljeđuje razini za izvođenje. Spremanje podataka u registre obavlja se na temelju signala koji ovoj razini pristižu iz zadnje razine, razine za upis u registre. Dakle, registri su spremljeni u ovoj razini te je ona jedina koja ima mogućnost čitanja i pisanja u njih.

Razina izvođenja naredbe prima vrijednosti operanda i vrijednost programskog brojila iz prošle razine, uz informaciju o operaciji koja se treba izvesti. U ovoj razini izvode se aritmetičko-logičke operacije, stoga ona sadrži aritmetičko-logičku jedinicu (eng. Arithmetic Logic Unit, ALU). Aritmetičko-logička jedinica prima dva ulazna operanda, a vraća rezultat zadane operacije. Operandi koji ulaze u ALU odabiru se korištenjem dva multipleksora. Prvi multipleksor bira hoće li prvi operand biti prvi registar ili programsko brojilo, a drugi multipleksor bira hoće li drugi operand biti drugi registar ili neposredna vrijednost. Rezultat operacije prosljeđuje se sljedećoj razini ili razini dohvata naredbe u slučaju naredbe skoka. Osim aritmetičko-logičkih operacija, ova razina se bavi i poravnavanjem podataka koje je potrebno pohraniti u memoriju u idućoj razini. Poravnavanje podataka je nužno jer procesorska jezgra podržava operacije nad riječima (32b), poluriječima (16b) i bajtovima (8b). Uz to, ova razina je zadužena i za provjeru uvjeta naredbi skoka. Ukoliko je uvjet skoka zadovoljen, ona izravno šalje adresu iduće naredbe razini za dohvat naredbe, koja tu vrijednost smješta u programsko brojilo.

Razina za pristup memoriji prima vrijednost rezultata aritmetičko-logičke jedinice, vrijednost programskog brojila te podatak za pohranu od prethodne razine. U ovoj razini dolazi do čitanja podataka iz podatkovnog dijela memorije, odnosno pohrane podataka, što ju čini jedinom razinom koja ima pristup podatkovnom dijelu memorije. Ukoliko se radi o

naredbama koje čitaju/pohranjuju podatak u memoriju, adresa podatka spremljena je u rezultatu iz aritmetičko-logičke jedinice koji je pristigao iz prošle razine. Prilikom čitanja podatka iz memorije, ova razina obavlja poravnavanje podatka na odgovarajuće mjesto u 32-bitnom zapisu. Ova razina je također zadužena za odabir podatka kojeg će proslijediti sljedećoj razini, a koji će se u konačnici spremiti u zadani registar. Za taj odabir koristi multipleksor koji ima tri ulaza: programsko brojilo, rezultat aritmetičko-logičke naredbe i podatak pročitan iz podatkovne memorije, te se jedan od njih prosljeđuje razini upisa u registre.

Razina upisa u registre jest najjednostavnija razina od dosad navedenih. Ona prima podatak i broj registra u koji se taj podatak treba pohraniti i te informacije prosljeđuje razini dekodiranja naredbe.



Sl. 1.1 Grafički prikaz protočne strukture

Slika 1.1 grafički prikazuje uloge svake pojedine faze i signale pomoću kojih komuniciraju. Imena ulaznih signala obojena su u zelenu boju, a imena izlaznih signala u crnu. Na slici su prikazani samo najvažniji signali koji omogućuju rad procesora.

2.2. Hazardi

Hazardi su događaji koji uzrokuju odstupanja od normalnog tijeka izvođenja naredaba u protočnoj strukturi. U tim situacijama protočna struktura ne može obraditi sljedeću naredbu odmah u sljedećem periodu, nego mora poduzeti određene akcije, poput zaustavljanja protoka ili ubacivanja mjehurića. Postoje tri različite vrste hazarda, a to su strukturni, podatkovni i upravljački hazardi.

Strukturni hazard je naziv za odstupanja uzrokovana strukturom procesora, najčešće „uskim grlom“ memorije zbog kojeg se dohvat naredbe iz memorije i naredba čitanja / spremanja podatka u memoriju ne mogu odraditi u jednom koraku. S obzirom na to da FRISC-V procesor ima Harvard arhitekturu sa posebnim instrukcijskim i podatkovnim sučeljem prema memoriji, do ovog hazarda ne može doći te ga neću detaljnije opisivati.

Podatkovni hazard uzrokovan je činjenicom da se razina upisa u registre nalazi na samom kraju protočne strukture. Takva arhitektura stvara probleme ukoliko se nađu dvije naredbe takve da druga naredba za rad treba rezultate prve naredbe, no prva naredba još nije došla do WB razine i njezini rezultati nisu još pohranjeni. Kada bi u tom trenutku protočna struktura normalno nastavila s radom, druga naredba bi koristila pogrešne, neizmijenjene vrijednosti operanda i zato dobila pogrešan rezultat. Protočna struktura ovaj hazard rješava ubacivanjem mjehurića na prikladno mjesto. **Mjehurić** je naziv za stanje razine u kojem ona ne obavlja nikakav posao te u kojem dolazi do brisanja podataka koji se u tom trenutku u njoj nalaze. Stanje mjehurića se prenosi protočnom strukturom, poput naredaba, uzrokujući odgodu izvođenja sljedećih naredaba. Podatkovni hazard može se pojaviti između razina ID i MEM te između razina ID i EXE.

Podatkovni hazard između razina ID i MEM može se prepoznati tako što je registar u koji se treba spremiti rezultat naredbe u MEM razini jednak jednom od registara koji se koriste kao operandi naredbe u ID razini. U tom slučaju protočna struktura privremeno zaustavlja (eng. stall) rad razina IF i ID, a u razinu EXE ubacuje mjehurić. Nakon ubacivanja mjehurića, protočna struktura normalno nastavlja s radom te mjehurić nastavlja putovati protočnom strukturom.

Podatkovni hazard također se može pojaviti između naredaba koje se nalaze u razinama ID i EXE. On se prepoznaje na jednak način kao i prethodni. Protočna struktura tada privremeno zaustavlja razine IF i ID, a u razinu EXE ubacuje mjehurić. No, ponovnim pokretanjem rada dolazi do novog hazarda – par naredaba koji je uzrokovao hazard u razinama ID i EXE sada

se nalazi u ID i MEM. Zato protočna struktura ubacuje novi mjehurić u EXE razinu, dok razine IF i ID i dalje stoje zaustavljene. Napokon, ubacivanjem drugog mjehurića problem je riješen i protočna struktura nastavlja s normalnim radom.

Upravljački hazard nastaje prilikom ispunjenja uvjeta naredbe grananja te naredbi skokova. Do njega dolazi zbog činjenice da se protočna struktura oslanja na slijedno izvođenje naredbi – dakle, naredbe u protočnu strukturu ulaze točno onim redoslijedom kojim su zapisane u programu, bez obzira na to jesu li one točno te naredbe koje se trebaju sljedeće izvesti ili ne. Naredbe skoka ispituju je li uvjet skoka ispunjen ili nije u razini EXE. U tom trenutku se u razinama IF i ID nalaze naredbe koje bi se trebale izvesti slijedno nakon naredbe skoka, no zbog akcije skoka do njihovog izvođenja ne smije doći. Upravo zato, kada protočna struktura prepozna zadovoljenje uvjeta skoka u razini EXE, u razine IF i ID ubacuju se mjehurići. U idućem ciklusu adresa skoka se nalazi u programskom brojilu, a prethodno ubačena dva mjehurića nastavljaju putovati protočnom strukturom, smještajući se u razine ID i EXE.

3. FRISC-V procesor

Naziv FRISC-V dolazi od FER RISC-V i označava verziju RISC-V procesora koja se razvija i predaje na FER-u. FRISC-V ima arhitekturu registar-registar (eng. load-store architecture) koja omogućuje brzo izvođenje naredaba, jednostavan format naredaba te protočnu strukturu na kojoj je jezgra procesora temeljena. Sam procesor je izveden kao procesor s mekom jezgrom [5], korištenjem jezika SystemVerilog za opis sklopovlja. S obzirom na organizaciju pristupa memoriji, FRISC-V ima Harvard arhitekturu koja ima razvijena 2 neovisna pristupa memoriji, podatkovni i instrukcijski pristup.

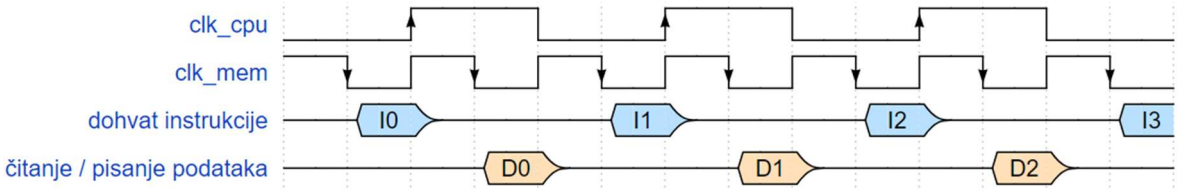
3.1. Opis modula FRISC-V procesora

Memorija ima mogućnost proizvoljnog konfiguriranja veličine. U ovom radu se koristila veličina od 1024 lokacije širine 32 bita, dakle na raspolaganju stoje adrese od 0x000 do 0xFFC. Početna adresa za spremanje naredaba također se može konfigurirati te je u ovom radu 0x000. Memorija je izvedena kao memorija s nasumičnim pristupom u koju se može i čitati i pisati (eng. Random Access Memory, RAM), bez dijela koji bi bio namijenjen isključivo za čitanje (eng. ROM). Memorijom se upravlja pomoću 32-bitnih podatkovnih i adresnih sabirnica, signala za omogućavanje rada (eng. enable signals), signala koji govore je li u tijeku čitanje ili pisanje u memoriju te dodatnih pomoćnih signala. Uz instrukcijski i podatkovni pristup, memorija ima izveden i dodatan pristup za verifikaciju ispravnog rada sustava. Rad memorije sinkronizira se korištenjem signala takta za ispitivanje ukoliko je procesor u načinu rada za ispitivanje, odnosno signalom takta memorije ukoliko je procesor u normalnom načinu rada.

Kako bi procesi čitanja i pisanja u pojedine dijelove memorije bili što brži te kako bi se ciklus takta iskoristio na što bolji način, razvijen je poseban modul za raspodjelu takta, odgovoran za sinkronizaciju rada pojedinih dijelova procesora.

Modul za raspodjelu takta (eng. clock divider) je modul koji na ulazu prima signal takta iz vanjskog izvora, a na izlaz postavlja signale takta clk_cpu i clk_mem. Signali takta su izvedeni tako da je frekvencija takta clk_mem dva puta veća od frekvencije takta clk_cpu. Signal clk_cpu je glavni signal takta koji sinkronizira rad procesora, memorije i samog modula za raspodjelu takta. U prvoj polovici ciklusa takta clk_cpu dolazi do dohvaćanja naredbe (eng. fetch), a u drugoj polovici ciklusa se po potrebi upisuju/čitaju podatci iz

podatkovnog dijela memorije. Okidač za obje ove akcije jest padajući brid (eng. negedge) signala clk_mem. Opisani vremenski odnosi mogu se vidjeti na slici 3.1.



Sl. 3.1 Vremenski dijagram odnosa signala takta [6]

Takva vremenska raspodjela uzima u obzir da je podacima koji se postave na sabirnicu potrebno neko vrijeme da se stabiliziraju prije nego što poprime svoju konačnu vrijednost.

Modul procesorske jezgre friscv_cpu temelji se na protočnoj strukturi s pet razina te je svaka razina izvedena kao zasebni modul. Uz tih pet modula dodan je i šesti, upravljački modul friscv_pipeline koji prvenstveno upravlja protočnom strukturom te resetiranjem i hazardima. Moduli su razvijeni tako da što vjernije slijede podjelu zadataka po razinama opisanu u poglavlju 2.1, stoga ću u opisima modula koji slijede usmjeriti pažnju način na koji su funkcionalnosti izvedene i signale koji omogućuju ovakav način rada.

IF modul upravlja programskim brojilom i dohvaćanjem naredbe iz instrukcijskog dijela memorije, s kojim komunicira putem instrukcijskog sučelja. Programsko brojilo (PC) može poprimiti jednu od sljedeće tri vrijednosti: 0 ukoliko je aktivan reset, PC + 4 koja se postavlja u pc_plus_4_out signal ili jump_branch_addr_in. Jump_branch_addr_in ulazi u IF modul iz EXE modula te se sprema u PC kada dođe do izvođenja naredbe skoka.

ID modul prima strojni kod naredbe iz IF razine te na temelju tipa naredbe (navedeni u 1. poglavlju) rastavlja njen strojni kod. Iz strojnog koda naredbe saznaje koji su brojevi registra operanda i registra rezultata, kolika je neposredna vrijednost i koji je operacijski kod naredbe koja se treba izvršiti. Na temelju tih podataka na svoj izlaz postavlja signale koji ulaze u EXE razinu i koji joj govore što se treba izvesti. Ovaj modul također ima ulogu upravljanja registrima opće namjene, koji su izvedeni kao polje 32-bitnih podataka veličine 32 bita. Prilikom prvog pokretanja procesora te nakon resetiranja, svi su registri postavljeni u nulu.

EXE modul je zadužen za izvođenje aritmetičko-logičkih operacija te stoga ima izvedenu aritmetičko-logičku jedinicu. Podržane operacije su zbrajanje, oduzimanje, logički I, logički ILI, logički XOR, logički posmaci ulijevo i udesno, aritmetički posmak udesno i usporedbe

s predznakom i bez predznaka. EXE razina je također zadužena za provjeru istinitosti uvjeta naredaba skoka, za što koristi poseban modul za određivanje istinitosti grananja, tzv. branch unit.

Modul za određivanje istinitosti uvjeta grananja (eng branch unit) jest modul koji prima operande i pripadni uvjet, a vraća signal branch_ok_out koji iznosi 1 ako je uvjet zadovoljen, a 0 ako nije. Za određivanje istinitosti uvjeta, modul za određivanje istinitosti uvjeta grananja koristi signale:

- C (eng. carry flag) –signal za prijenos
- Z (eng. zero flag) – signal za nulu
- N (eng. negative flag) – signal za negativnu vrijednost
- V (eng. overflow flag) – signal za preljev

Na temelju ulaznih signala automatski se postavljaju pripadni pomoćni signali, koje zatim određuju istinitost uvjeta skoka.

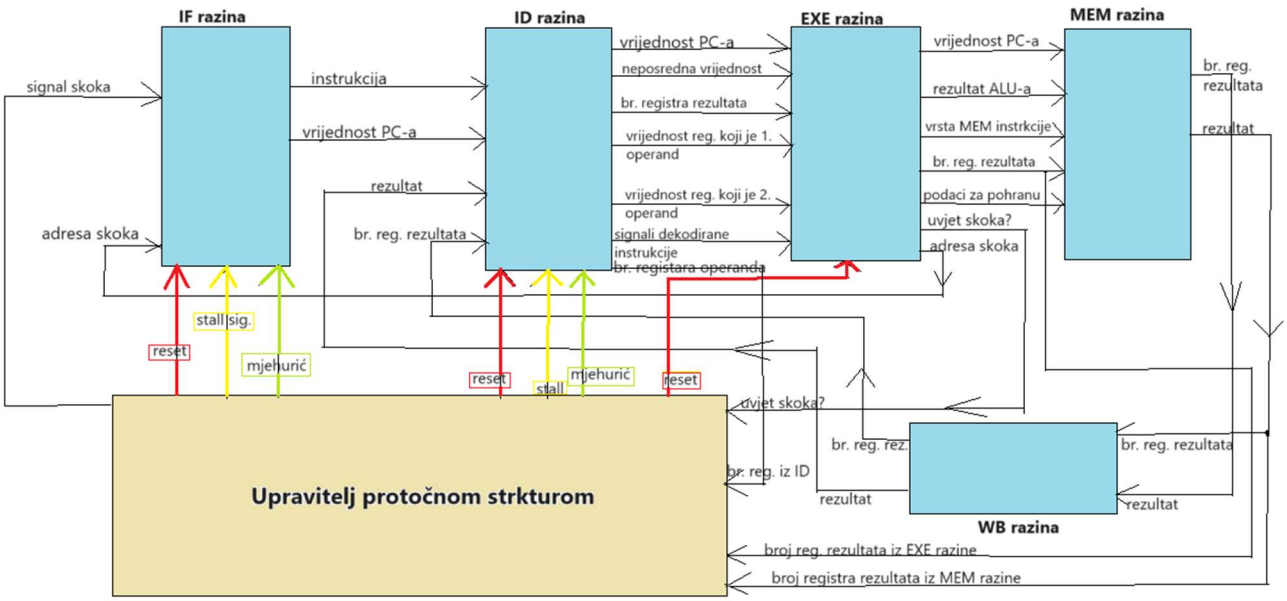
MEM modul je dio jezgre koji ima podatkovno sučelje prema memoriji te obavlja operacije pisanja i čitanja u memoriju. Prilikom postavljanja adrese s koje se treba pročitati podatak iz memorije, najprije se adresa „poravna“ – ukoliko se dohvaća podatak veličine poluriječi adresa mora biti djeljiva s 2 (zadnji bit adrese je 0), a ako se dohvaća podatak veličine riječi adresa mora biti djeljiva s 4 (zadnja dva bita adrese su 0). Prilikom čitanja podataka iz memorije, 32-bitna sabirnica uvijek vraća podatak veličine riječi pa modul uzima samo one bajtove koji su mu potrebni (npr. prvi bajt za naredbu load byte), a ostatak ispuni bitom predznaka ili nulama, ovisno radi li operacija s podacima bez predznaka ili s predznakom.

WB modul direktno prosljeđuje signale koje je primio od MEM modula ID modulu. Ovaj modul je jedini od navedenih koji ne sprema niti jedan signal.

Svaki modul, osim WB-a, ima posebne interne registre, tzv. buffere, u koje sprema vrijednost ulaznih i izlaznih signala na pozitivni brid signala takta cpu_clk. Bufferi osiguravaju stabilnost i konzistentnost podataka tijekom cijelog ciklusa.

Modul za upravljanje protočnom strukturom (eng. pipeline control) upravlja svim razinama protočne strukture, resetiranjem i zaustavljanjem razina, omogućuje naredbe skoka i kontrolira protok mjehurića koji nastaju prilikom hazarda. S obzirom na to da taj modul nije radio ispravno u početnoj verziji sustava, ovdje ga neću opisivati. Detaljan opis nove izvedbe ovog modula stavljam u poglavlje 4.1.

Na slici 3.2 opisan način povezivanja razina prikazan je grafički. Na slici su prikazani samo najvažniji signali koji omogućavaju rad i komunikaciju između razina.



Sl. 3.2 Grafički prikaz modula razina i najvažnijih signala

3.2. Zahtjevi na proširenje sustava

Originalna verzija sustava nije bila u potpunosti funkcionalna. Naime, modul za upravljanje protočnom strukturom je pogrešno postavljao signale za zaustavljanje i signale mjehurića pa su naredbe skoka blokirale rad cijele jezgre, slično kao i podatkovni hazardi. Stoga je prvi zahtjev bio popraviti upravljački dio jezgre.

Daljnji zahtjev bio je implementirati sustav na pločicu PYNQ-Z2 s ARM Cortex-A9 procesorskom jezgrom i uspostaviti komunikaciju između osobnog računala i FRISC-V procesora. Kako bi se sustav mogao implementirati na pločicu, bilo je potrebno izmijeniti dizajn memorije objedinjenjem signala takta koji su stvarali poteškoće pri sintezi. Također, za potrebe komunikacije sustava s vanjskim svijetom bilo je potrebno razviti modul vanjskog izlaznog registra pomoću kojeg će se moći upravljati periferijama pločice.

S obzirom na to da jednom pokrenuti procesor nikada ne bi trebao stati s radom, bilo je potrebno napraviti modul koji će signalizirati kraj izvođenja programa. Taj modul bio bi od

velike koristi pri testiranju rada procesora, a kasnije bi pružao korisnu informaciju korisniku kada može započeti s čitanjem sadržaja memorije.

Za potrebe komunikacije bilo je potrebno razviti aplikacije na strani osobnog računala i na strani pločice koje će dati korisniku jednostavno sučelje za upravljanje FRISC-V procesorom. Sučelje treba sadržavati opcije resetiranja procesora, punjenja memorije, čitanja sadržaja memorije, brisanja memorije, upisivanja programa te pokretanja i zaustavljanja rada procesora.

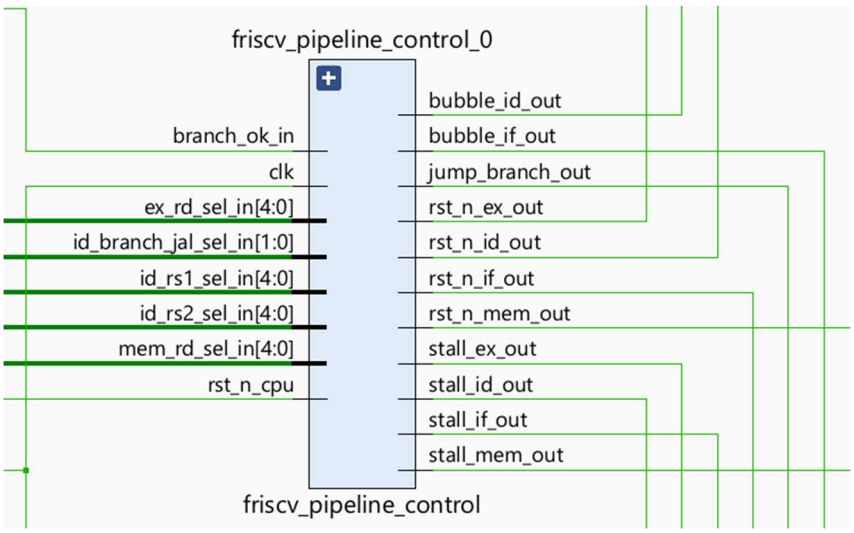
4. Izvedba proširenja sustava

Proširenje sustava može se podijeliti u dva dijela: unaprjeđenje već postojećih dijelova i dodavanje potpuno novih dijelova. Dijelovi sustava koji su unaprijeđeni su modul za upravljanje protočnom strukturom i memorija, a dijelovi koji su dodani su modul za signaliziranje kraja programa i modul izlazne jedinice opće namjene.

Za izvedbu proširenja sustava korišten je programski alat Xilinx Vivado, verzija 2022.2, a za testiranje rada izvedbe korišten je programski alat Xilinx Vitis, verzija 2022.2. Dodani moduli pisani su u jeziku za modeliranje digitalnog sklopovlja SystemVerilogu, jednako kao i ostatak sustava.

4.1. Implementacija FRISC-V upravljačkog modula

Glavne namjene modula za upravljanje protočnom strukturom (eng. pipeline control) su resetiranje razina, upravljanje zaustavljanjem i protokom mjehurića kroz razine prilikom hazarda te upravljanje naredbama skoka. Upravitelj protočnom strukturom ima posebne signale za resetiranje, zaustavljanje i ubacivanje mjehurića za svaku razinu.



Sl. 4.1. Blok dizajn modula za upravljanje protočnom strukturom s ulaznim i izlaznim signalima

Na slici 4.1. prikazan je blok dizajn modula sa svim njegovim ulaznim signalima (lijevo na slici) i izlaznim signalima (desno na slici).

Ulazni signali u ovaj modul su signal takta, signal globalnog reseta te signali koji služe za određivanje je li došlo do hazarda u izvođenju naredbi. Kako bi modul mogao odrediti je li u izvođenju došlo do pojave podatkovnog hazarda, prima ulazne signale koji predstavljaju brojeve registra koji sudjeluju u izvođenju naredbi iz ID, EXE i MEM razine. Za određivanje upravljačkog hazarda, modul prima signal iz ID razine koji daje informaciju o kojoj vrsti naredbe skoka se radi (branch / jal) te signal iz EXE razine koji javlja je li uvjet skoka zadovoljen.

Izlazni signali iz ovog modula su signali reseta i signali zaustavljanja za svaku razinu osim razine WB, signali mjehurića za razine IF i ID te signal za upravljanje skokovima. Razina WB nema izvedeno spremanje signala niti funkcionalnost koja se aktivira taktom te za nju nije imalo smisla stvarati signale za resetiranje i zaustavljanje, pošto se tako i tako nema što za resetirati / zaustaviti.

Resetiranje sustava izvedeno je u negativnoj logici, a može se aktivirati postavljanjem globalnog signala za resetiranje na logičku nulu ili pritiskom na gumb za resetiranje na pločici. Bilo koji od ta dva događaja postavit će signal reseta procesora na logičku nulu, a taj signal ulazi u procesorsku jezgru upravo u modul za upravljanje protočnom strukturom. Signal reseta u modulu je izveden kao sinkroni signal, koji aktivira signale za resetiranje svake pojedine razine, u trenutku kada uđe u aktivno stanje.

Signali za zaustavljanje (eng. stall signal) govore razini da privremeno stane s radom. Oni se aktiviraju ukoliko dođe do podatkovnog hazarda između razina ID i EXE ili između razina ID i MEM. Podatkovni hazardi se jednostavno prepoznaju – ukoliko su brojevi registara koji izlaze iz ID razine i broj registra koji izlazi iz EXE ili MEM razine isti, to znači da je došlo do podatkovnog hazarda. Tada se signali za zaustavljanje IF i ID razine postavljaju se u logičku jedinicu, no samo ukoliko u tom trenutku u EXE razini nije naredba skoka. Ukoliko je u EXE razini u tom trenutku naredba skoka, upravljački hazard ima veći prioritet od podatkovnog te do izvođenja naredbe koja je u tom trenutku u ID razini nikada neće doći.

S obzirom na ulogu signala za zaustavljanje u sustavu, jasno je da signali za zaustavljanje EXE i MEM razine nikada ne bi trebali biti u aktivnom stanju, stoga su trajno postavljeni u logičku nulu.

Mjehurići (eng. bubble) su signali koji daju znak razini da se naredba koju ona trenutno obrađuje nikada neće izvesti i da ne mora obavljati svoje uobičajene funkcionalnosti. Postavljanjem signala mjehurića u logičku jedinicu, podatci koje razina u tom trenutku

obrađuje gube svaki smisao te bi bilo poželjno da se postave na inicijalne vrijednosti. Također, ukoliko signale postavimo na inicijalne vrijednosti, problem kruženja mjehurića kroz razine rješava se sam od sebe – protočnom strukturom kružit će inicijalne vrijednosti signala, koje podrazumijevaju NOP naredbu („prazna“ naredba koja ne obavlja nikakav posao – istovjetna naredbi ADDI x0, x0, 0). Dakle, takva izvedba omogućava da se mjehurić u protočnu strukturu ubaci na vrlo jednostavan način, a on dalje nastavlja samostalno kružiti protočnom strukturom, bez potrebe za aktiviranjem novih signala mjehurića. Upravo zbog toga je odlučeno da će ulogu signala mjehurića igrati signali za resetiranje, a signali mjehurića će se aktivirati tamo gdje je tijekom mjehurića potrebno sačuvati dio funkcionalnosti. Potreba za očuvanjem dijela funkcionalnosti javila se samo u razinama IF i ID i stoga jedino te dvije razine imaju svoje signale mjehurića.

IF razina je odgovorna za upravljanje programskim brojilom te je jako bitno da se pojavom mjehurića vrijednost programskog brojila ne resetira na početnu. Upravo zato, kada bi trebalo doći do pojave mjehurića u IF fazi, istovremeno dolazi do aktivacije oba signala – signala mjehurića i signala za resetiranje. Prilikom resetiranja IF faze, provjerava se je li aktivan signal mjehurića. Ukoliko jest, vrijednost programskog brojila se ne mijenja, a u suprotnom se postavlja u inicijalnu vrijednost. Signal mjehurića za ovu razinu se aktivira ako se u ID razini nalazi naredba skoka JAL.

ID razina je odgovorna za spremanje izračunatih vrijednosti u registre te je također vrlo bitno da pojava mjehurića ne spriječi tu funkcionalnost. Zato se u ID razini prilikom resetiranja provjerava je li signal mjehurića aktivan – ukoliko jest, vrijednosti registara se ne resetiraju i dolazi do upisa nove vrijednosti registra primljene iz WB razine. Signal mjehurića za ovu razinu aktivira se ako je u EXE razini zadovoljen uvjet grananja ili ako se u ID razini nalazi naredba skoka JAL koja će se izvesti (ako je istovremeno u EXE razini zadovoljen uvjet grananja ili ako je prošla naredba također bila naredba skoka JAL, do izvođenja ove JAL naredbe nikad neće doći).

S obzirom na to da u razinama EXE i MEM ne postoji funkcionalnost koju bi trebali obavljati tijekom mjehurića, te razine nemaju vlastite signale mjehurića, već signali za resetiranje u potpunosti obavljaju tu ulogu.

„Signal mjehurića“ za EXE razinu aktivira se pojavom podatkovnih hazarda, ispunjenjem uvjeta skoka i pojavom naredbe skoka JAL (koji će se sigurno izvesti) u ID razini.

„Signal mjehurića“ za razinu MEM nikada ne ulazi u aktivno stanje, zato što se je nemoguće da signal mjehurića treba započeti u razini MEM. Mjehurić u razinu MEM može doći u obliku resetiranih vrijednosti, no nikada neće u njoj inicijalno nastati.

Signal za upravljanje naredbom skoka jump_branch_out ulazi direktno u IF razinu i upravlja postavljanjem nove adrese u programsko brojilo. Taj signal se nalazi u logičkoj jedinici ukoliko je u EXE razini zadovoljen uvjet grananja ili ako je u prošlom ciklusu bila naredba skoka JAL u razini ID te je u ovom ciklusu ta naredba došla na red za izvođenje.

4.2. Optimizacija memorijskog modula

Memorijski modul na svom ulazu prima tri različita signala takta – signal takta procesora, signal takta memorije i signal takta za ispitivanje. Rad prva dva signala takta opisan je u poglavlju 3.1. Signal takta za ispitivanje aktivan je u ispitnom načinu rada te se njegove vrijednosti ručno postavljaju na ulaz procesora, kako bi se lakše mogao pratiti rad memorije.

Problem nastaje u činjenici da memorija treba imati iste proceduralne blokove u normalnom i u ispitnom načinu rada, no procedura koja se aktivira na signal takta ne može imati dva različita izvora takta.

Uočeno je da memorijski proceduralni blokovi u jednom trenutku trebaju samo jedan izvor takta na koji će se aktivirati – ukoliko je procesor u ispitnom načinu rada, to je signal takta za ispitivanje, a ukoliko smo u normalnom načinu rada, to je signal takta za memoriju. Stoga je odlučeno u dizajn memorije dodati multipleksor koji će kontrolirati na koji signal takta se memorijski blokovi aktiviraju ovisno o načinu rada procesora. Multipleksor je izveden pomoću jednostavne kombinacijske logike:

```
always_comb begin
    if (debug_mode)
    begin
        clk <= clk_debug;
    end
    else begin
        clk <= ~clk_mem;
    end
end
```

Kôd 4.1 – Kombinacijski proces multipleksora takta

Signal takta clk poprima vrijednost signala takta clk_debug ako je procesor u načinu rada za ispitivanje, inače poprima vrijednost negiranog signala takta clk_mem. Razlog negacije signala takta clk_mem jest taj da se memorijski proceduralni blok treba aktivirati na rastući brid signala takta za ispitivanje, a na padajući brid signala takta za memoriju te je ovako postignuto da se cijeli proceduralni blok aktivira na rastući brid signala clk.

Ova promjena u dizajnu memorije omogućila je uspješnu sintezu cijelog dizajna procesora.

4.3. Modul za signaliziranje kraja programa

Procesorska jezgra je zamišljena tako da sama od sebe nikada ne stane s radom, već izvodi naredbe zapisane u memoriji u beskonačnost. Takva izvedba smislena je za procesorsku jedinicu, no otežava testiranje rada procesora jer je teško znati u kojem trenutku je procesor završio s izvođenjem programa. Iz tog razloga dizajnu se dodaje modul za signaliziranje kraja programa kojeg će određeni niz naredaba aktivirati i koji će programeru dati do znanja da je program završio s izvođenjem.

Odlučeno je da će upis podatka 0x17 na zadnju adresu memorije 0xFFC biti akcija koja će označavati kraj programa. Ta akcija može se jednostavno implementirati sljedećim nizom asemblerskih naredaba:

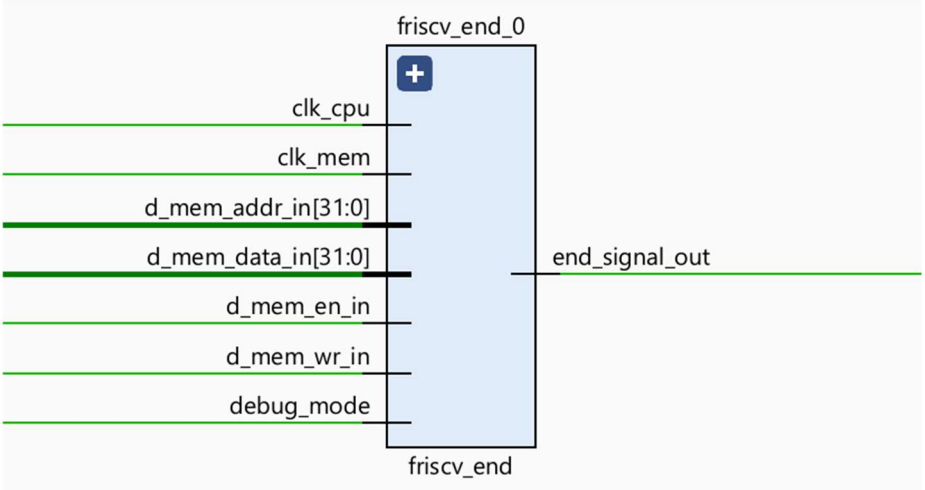
```
LUI s10, %hi(0xFFC)       ;učitaj viših 20 bitova adrese u reg s10
ADDI s10, s10, %lo(0xFFC) ;učitaj nižih 12 bitova adrese u s10
ADDI s11, x0, 0x17        ;spremi podatak za kraj programa u s11
SW s11, 0(s10)            ;pohrani sadržaj reg. s11 na adresu u s10
```

Kôd 4.2 – Niz asemblerskih naredaba za signaliziranje kraja programa

Kako bi modul mogao prepoznati tu akciju te postaviti izlazni signal za kraj programa, na ulazu prima sljedeće signale:

- Signale takta procesora (clk_cpu) i memorije (clk_mem)
- Signal ispitivanja procesora (debug_mode)
- Adresne (d_mem_addr_in) i podatkovne 32-bitne signale (d_mem_data_in)
- Signal za omogućavanje rada podatkovne memorije (d_mem_en_in)
- Signal pisanja u podatkovnu memoriju (d_mem_wr_in)

Modul prima signale takta kako bi radio sinkronizirano s ostalim dijelovima procesora. Za detekciju akcije koja označava kraj programa, modul mora primati adresu i podatak iz naredbe i uspoređivati ih s zadanim vrijednostima (0xFFC, 0x17). Modul prima ostala tri signala kako bi bio siguran da je u tom trenutku uistinu akcija pisanja u podatkovnu memoriju u tijeku te da se to događa od strane procesorske jezgre, a ne od strane sučelja za ispitivanje (dakle, debug_mode mora biti postavljen u 0).
Modul ima jedan izlazni signal, end_signal_out, koji se postavlja u logičku jedinicu kada se detektira akcija koja označava kraj programa. Modul sa svim ulaznim i izlaznim signalima prikazan je na slici 4.2.



Sl. 4.2. Blok dizajn modula za signaliziranje kraja programa

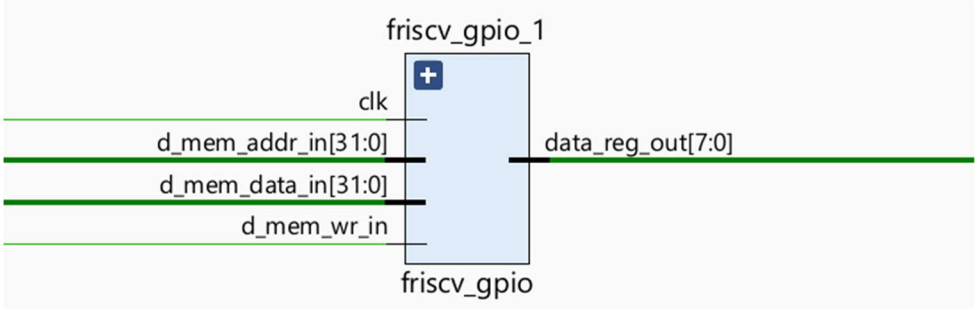
Radi lakšeg testiranja, signal end_signal_out spojen je na lampicu LED5 na pločici PYNQ Z2 koja svijetli zeleno kada program završi s izvođenjem.

S obzirom na to da je ova akcija jedini način za saznati kada je jezgra gotova s izvođenjem programa, dužnost je programera koji piše program za izvođenje na pločici s FRISC-V procesorskom jezgrom da ručno doda naredbe za signaliziranje kraja programa na kraj svog programa. U protivnom, kraj programa neće biti detektiran i lampica RGB LED5 nikad neće zasvijetliti.

4.4. Modul 8-bitnog izlaznog registra

Za potrebe komunikacije procesorske jezgre s periferijom pločice, pojavila se ideja izlaznog registra kojim bi procesorska jezgra upravljala, a koji bi bio izravno spojen na periferije pločice. Ta ideja ostvarena je modulom izlaznog registra.

Modul izlaznog registra služi kao poveznica između procesorske jezgre i periferije pločice. Procesorska jezgra može korištenjem naredbe za pohranu na jednostavan način programirati izlazni registar, koji programirane vrijednosti signala prosljeđuje periferiji na koju je fiksno ožičen. Blok dizajn modula prikazan je na slici 4.3. Ulazi u modul nalaze se na lijevoj strani bloka, a izlaz se nalazi na lijevoj.



Sl. 4.3 – Blok dizajn modula izlaznog registra

Ulazni signali modula su sljedeći:

- Signal takta clk, na koji je spojen signal takta memorije
- Adresne (d_mem_addr_in) i podatkovne (d_mem_data_in) 32-bitne signale podatkovne memorije
- Signal d_mem_wr_in koji govori je li u tijeku pisanje u podatkovnu memoriju

Modul prima adresne signale koje konstantno uspoređuje sa svojom baznom adresom te na taj način zna kada je naredba namijenjena njemu. U trenutku kada je adresa na adresnoj sabirnici jednaka baznoj adresi izlaznog registra, ukoliko je u tijeku akcija pisanja, modul sprema podatke sa podatkovne sabirnice u interni registar podataka.

Vrijednost internog registra trajno je postavljena na izlazni 8-bitni izlaz, data_reg_out.

Bazna adresa registra jest adresa koju modul prima kao parametar u trenutku instanciranja. Ta adresa mora se nalaziti izvan memorijskog raspona, jer će u suprotnom pokretati neželjenu akciju pisanja podataka (namijenjenih izlaznom registru) u memoriju.

U dizajn procesora dodane su dvije instance modula izlaznog registra. Prva instanca, gpio1, može se adresirati adresom **0x10000**, a na nju su spojene sljedeće periferije:

- [0], [1], [2] i [3] – LED lampice 0, 1, 2 i 3 na pločici PYNQ Z2
- [4] – plavo svjetlo LED lampice 4 na pločici PYNQ Z2
- [5] – zeleno svjetlo LED lampice 4 na pločici PYNQ Z2
- [6] – crveno svjetlo LED lampice 4 na pločici PYNQ Z2

Najviši bit izlaznog registra 1 nije ni s čime povezan i njegovo programiranje neće imati nikakav utjecaj.

Druga instanca, nazvana gpio2, može se adresirati adresom **0x20000** te su na nju spojene sljedeće periferije:

- [0], [1], [2], [3], [4], [5], [6], [7] - Arduino GPIO portovi od 0 do 7, slijedno

Instanca gpio1 namijenjena je programiranju pločice PYNQ Z2 na koju je postavljen i sam FRISC-V procesor, a instanca gpio2 je namijenjena programiranju pločice s periferijama, LPCXpresso Base Board, koristeći Arduino pinove.

Sljedeći kod ilustrira korištenje izlaznog registra gpio1 za paljenje lampica na pločici:

```
lui x20, %hi(0x10000)      ;učitaj baznu adresu GPIO1 u reg x20
addi x20, x20, %lo(0x10000)
addi x19, x0, 0b00000001   ;paljenje lampice LED0
sw x19, 0(x20)
addi x19, x0, 0b00000010   ;paljenje lampice LED1
sw x19, 0(x20)
addi x19, x0, 0b00000111   ;paljenje LED0, LED1 i LED2
sw x19, 0(x2)
addi x19, x0, 0b00001000   ;paljenje LED3
sw x1, 0(x20)
addi x19, x0, 0b00000000   ;ugasi sve lampice
sw x1, 0(x20)
```

Kôd 4.3 – Niz asemblerskih naredaba za upravljanje lampicama korištenjem izlaznog registra 1

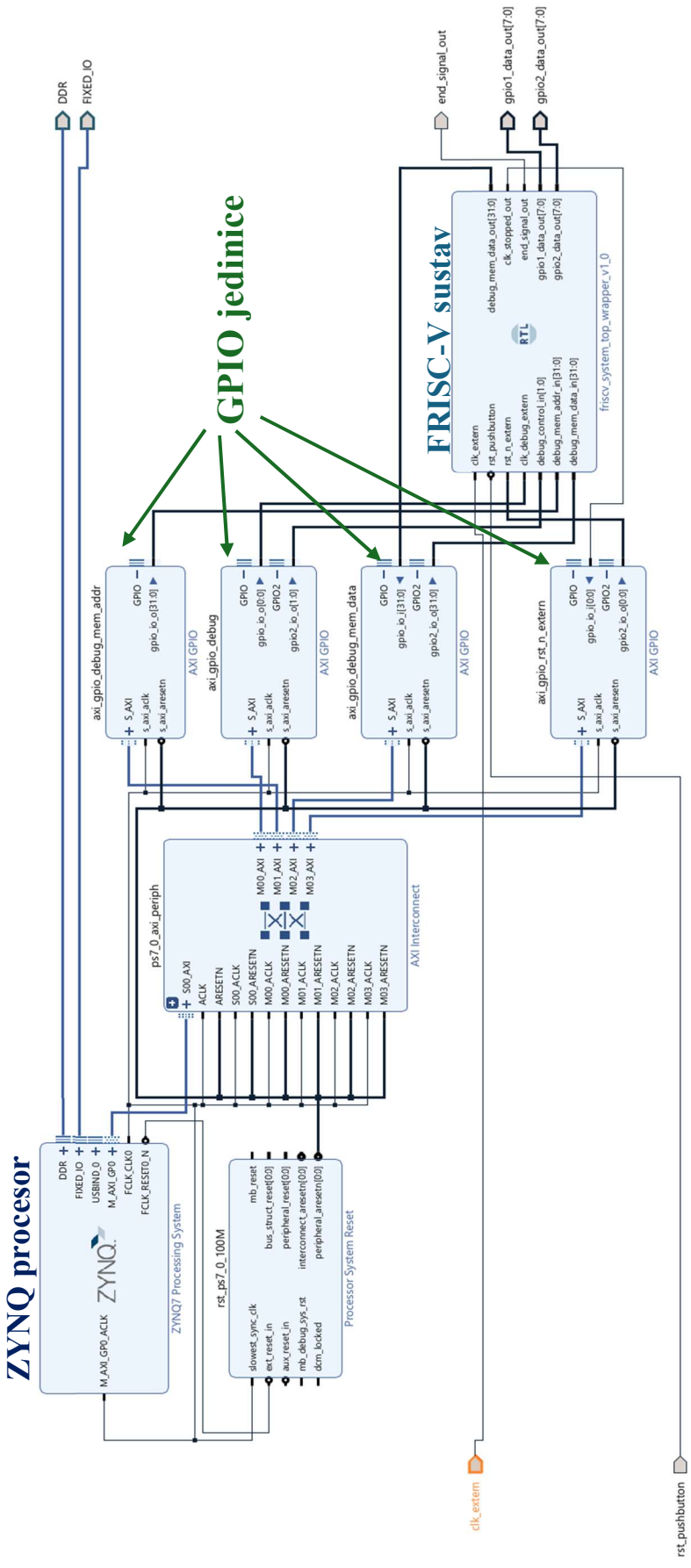
U kôdu 4.3 žuto su označene naredbe koje obavljaju operaciju programiranja izlaznog registra gpio1. Kao što je vidljivo na primjeru, programiranje izlaznog registra obavlja se vrlo jednostavno, korištenjem naredbe za pohranu podataka SW. Na isti način može se programirati i izlazni registar gpio2 – jedina razlika jest ta što bi se u registar x20 početno učitala bazna adresa izlaznog registra gpio2.

5. Testiranje i verifikacija sustava

Za potrebe testiranja i komunikacije s procesorom, FRISC-V stavljen je u blok dizajn zajedno s ZYNQ procesorom s dvije ARM Cortex A9 jezgre. Komunikacija između ova dva procesora ostvarena je korištenjem četiri ulazno-izlaznih jedinica opće namjene:

- GPIO za postavljanje adrese na memorijsko sučelje za ispitivanje FRISC sustava
  - Ima 32-bitni izlazni signal, spojen na debug_mem_addr_in signal FRISC-a
- GPIO sa dva kanala za postavljanje signala za ispitivanje
  - Prvi kanal ima 1-bitni izlazni signal za postavljanje signala takta clk_debug
  - Drugi kanal ima 2-bitni izlazni signal – bit [1] je spojen na način rada memorije (pisanje/čitanje), a bit [0] na signal debug_mode
- GPIO sa dva kanala za pisanje i čitanje podataka sa memorijskog sučelja za ispitivanje FRISC-a
  - Pisanje: 32-bitni izlazni signal, spojen na debug_mem_data_in signal
  - Čitanje: 32-bitni ulazni signal, spojen na debug_mem_data_out signal
- GPIO sa dva kanala za postavljanje reseta i čitanje upravljačkih signala FRISC-a
  - Prvi kanal ima 1-bitni izlazni signal spojen na rst_n_extern
  - Drugi kanal ima 1-bitni ulazni signal na koji je spojen signal clk_stopped

Na slici 5.1 može se vidjeti blok dizajn cijelog sustava na čipu i načini na koji su dijelovi sustava međusobno povezani. Ovaj dizajn je izdvojen iz aplikacije Vivado u obliku .xsa datoteke, te je putem Vitis okruženja programiran na PYNQ-Z2 pločicu.



Sl. 5.1 – Blok dizajn sustava na čipu sa označenim dijelovima

Kako bi ARM procesor mogao komunicirati s FRISC-V sustavom, najprije je razvijen aplikacijski sustav koji se pokreće na ARM procesoru. Aplikacijski sustav ARM-a upravlja FRISC-V procesorom tako da programira GPIO-ove koji ih povezuju. Razvijene su funkcije za resetiranje FRISC-a, postavljanje debug načina rada, pokretanje i zaustavljanje rada, brisanje memorije, upis programa, upis podataka, čitanje niza memorijskih adresa i čitanje jedne memorijske adrese.

S druge strane, ARM-ov aplikacijski sustav komunicira s korisnikom preko uspostavljene serijske veze. Svaka od funkcija koja se može izvesti kodirana je brojem od 1 do 9. ARM neprekidno čeka dolazak broja preko serijske veze od strane korisnika, a zatim sukladno primljenom broju pokreće zadanu funkciju.

Kako bi korisnik mogao slati ARM-u broj željene funkcije putem serijske veze, razvijena je aplikacija pisana u Pythonu. Aplikacija uspostavlja serijsku vezu s UART sučeljem na pločici, a zatim čeka u terminalu upis broja. Kada se u terminal upiše broj, ona ga prosljeđuje ARM-ovom aplikacijskom sustavu, koji zatim ispunjava traženi zahtjev.

Korištenjem ovih aplikacijskih sustava provjeren je rad FRISC-V procesora. Provjere ispravnog rada memorijskog modula nisu eksplicitno provedene, s obzirom na to da je taj dio ključan za rad sustava (pogotovo instrukcijska memorija) i njegov neispravan rad uzrokovao bi neispravan rad svih testiranih programa. Dakle, ispravnim izvršavanjem bilo kojeg programa možemo zaključiti da memorija radi ispravno. Modul za upravljanje protočnom strukturom testiran je pokretanjem programa koji sadrži niz instrukcija s raznim kombinacijama podatkovnih i upravljačkih hazarda te je dobiveno očekivano stanje memorije, čime je zaključeno da modul ispravno radi. Moduli za signaliziranje kraja programa i izlazni registri testirani su pokretanjem jednostavnih programa na FRISC-V sustavu te se njihov rad lako mogao provjeriti promatranjem svijetljenja lampica. Na FRISC-V procesoru zatim su pokrenuti kompliciraniji programi koji testiraju cjelokupnu funkcionalnost procesora te su sva izvođenja dala točne rezultate.

Zaključak

U proteklih nekoliko godina, studenti su pod vodstvom prof. Maria Kovača započeli s razvojem FER-ove inačice 32-bitnog RISC-V procesora s mekom jezgrom, zvanog FRISC-V. U sklopu ovog završnog rada, taj procesor je unaprijeđen i proširen, u svrhu implementacije procesora na Xilinx FPGA pločicu čijim će periferijama procesor moći upravljati.

U upravljačkom dijelu protočne strukture ispravljene su greške. Modul za upravljanje protočnom strukturom kontrolira rad razina korištenjem signala za resetiranje, signala za zaustavljanje i signala za mjehuriće. Modul je napravljen s ciljem da uspješno detektira podatkovne i upravljačke hazarde te sukladno time podiže i spušta upravljačke signale svake pojedine razine.

Promijenjen je dizajn memorijskog modula. Dodan je dio koji kontrolira koji signal takta je aktivan, kako bi dizajn memorije postao pogodan sintezi sustava.

Za potrebe testiranja i pokretanja programa na procesoru, dodan je modul za signaliziranje kraja programa. Akcija „kraja programa“ jest upis podatka 0x17 na zadnju memorijsku adresu, 0xFFC te je stoga pri pisanju programa potrebno imati na umu da se na kraj mora dodati niz naredaba koje izvode navedenu akciju. Signal za kraj programa spojen je na LED lampicu na pločici kako bi bilo vizualno vidljivo kada je procesor završio s korisnim radom.

Kako bi procesor mogao komunicirati s periferijama, razvijen je modul 8-bitnog izlaznog registra. U dizajn sustava dodane su dvije instance modula izlaznog registra –prva instanca, čija je bazna adresa 0x10000, namijenjena je upravljanju periferijama na pločici na kojoj se nalazi i sam FRISC-V, dok je druga instanca, s baznom adresom 0x20000, namijenjena upravljanju periferijama vanjske pločice preko Arduino GPIO pinova.

FRISC-V procesor spojen je zajedno sa dvojezgrenim ARM Cortex-A9 procesorom i ulazno-izlaznim jedinicama opće namjene u cjelokupni sustav na čipu. Za potrebe komunikacije korisnika s FRISC-V procesorom, razvijena je aplikacijska podrška koja se izvodi na ARM procesoru te aplikacija koja se izvodi na osobnom računalu korisnika. Aplikacijska podrška za ARM procesor jest dio koji rukuje komunikacijom između korisnika sa strane računala i FRISC-V procesorom.

Implementirani sustav pogodan je za korištenje u edukacijske svrhe. Korištenjem razvijenih aplikacija, korisnik može na jednostavan način programirati FRISC-V procesor da izvodi program sastavljen od skupa naredaba iz RV32I. Razvijeni sustav također je pogodan za proširenja i budući razvoj, s obzirom na to da se RISC-V ISA može nadograditi brojnim standardnim i nestandardnim proširenjima.

Literatura

[1] Kovač M., *Arhitektura računala 1R*, predavanja 2023.

[2] Andrew Waterman i Krste Asanović, *The RISC-V Instruction Set Manual, Volume I: User-Level ISA*, Document Version 2.2, svibanj 2017. Poveznica: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf, pristupljeno 20. svibnja 2024.

[3] AMD, pločica AUP PYNQ-Z2, Poveznica: https://www.amd.com/en/corporate/university-program/aup-boards/pynq-z2.html, pristupljeno 13. lipnja 2024.

[4] Embedded Artists, LPCXpresso Base Board, Poveznica: https://www.embeddedartists.com/products/lpcxpresso-baseboard/, pristupljeno 13. lipnja 2024

[5] Vučić M., Ugradbeni računalni sustavi: Upotreba 32-bitnih mikrokontrolera. Materijali za predavanja. Zagreb, 2021.

[6] *WaveDrom Editor*, online alat za izradu vremenskih dijagrama. Poveznica: https://wavedrom.com/editor.html, pristupljeno 13. lipnja 2024.

Sažetak

Izvedba sustava sa FRISC-V procesorom

Pod vodstvom prof. Maria Kovača, studenti su razvili FER-ovu inačicu 32-bitnog FRISC-V procesora. Ovim radom unaprijeđen je FRISC-V procesor, omogućujući njegovu implementaciju na Xilinx FPGA pločicu te samostalno upravljanje periferijama. Upravljački modul protočne strukture je unaprijeđen te može uspješno detektirati i rješavati podatkovne i upravljačke hazarde u protočnoj strukturi. Optimiziran je memorijski modul i dodan sustav za signalizaciju kraja programa. Razvijen je i modul 8-bitnog izlaznog registra za upravljanje periferijama. Sustav je integriran s ARM Cortex-A9 dvojezgrenim procesorom, te je jednostavno programiranje i korištenje omogućeno putem razvijenih aplikacija.

Ključne riječi: RISC-V arhitektura, RV32I ISA, FRISC-V procesor, protočna struktura, izlazni registar, signal za kraj programa

Summary

Implementation of FRISC-V processor system

Under the guidance of prof. Mario Kovač, students developed FER's version of a 32-bit FRISC-V processor. This thesis enhanced the FRISC-V processor, enabling its implementation on a Xilinx FPGA board and independent control of peripherals. The control module of the instruction pipeline was improved and is now capable of successfully detecting and resolving data and control hazards within the pipeline. The memory module was optimized, and a system for end-of-program signaling was added. An 8-bit output register module was also developed for peripheral management. The system was integrated with a dual-core ARM Cortex-A9 processor, and simple programming and usage were facilitated through the developed applications.

Keywords: RISC-V architecture, RV32I ISA, FRISC-V processor, instruction pipeline, output register, end-of-program signal

Skraćenice

ISA	*Instruction Set Architecture*	arhitektura seta naredaba
FPGA	*Field Programmable Gate Arrays*	programirljivo polje logičkih sklopova
PC	*Program Counter*	programsko brojilo
LR	*Link Register*	registar povratne adrese
CSR	*Control Status Register*	statusni registar
IF	*Instruction Fetch Stage*	razina dohvata naredbe
ID	*Instruction Decode Stage*	razina dekodiranja naredbe
EXE	*Execution Stage*	razina izvođenja naredbe
MEM	*Memory Stage*	razina za pristup memoriji
WB	*Writeback Stage*	razina za upis u registre
ALU	*Arithmetic Logic Unit*	aritmetičko-logička jedinica
RAM	*Random-Access Memory*	memorija s nasumičnim pristupom
ROM	*Read-Only Memory*	memorija isključivo za čitanje
NOP	*No Operation*	naredba bez funkcionalnosti
GPIO	*General Purpose Input/Output*	ulazno-izlazna jedinica opće namjene