POKAZNA VEŽBA 7

Strukture za računanje

Potrebno predznanje

- Urađena pokazna vežba 6
- Poznavanje aritmetičkih digitalnih sistema i aritmetičko-logičkih jedinica
- Osnovno znanje upravljačkih jedinica digitalnih sistema

Šta će biti naučeno tokom izrade vežbe?

Nakon urađene vežbe bićete u mogućnosti da:

- Rukovodite različitim nivoima apstrakcije u vašem digitalnom sistemu – nakon implementiranja jednostavnije komponente, istu ćete koristiti kao crnu kutiju unutar složenijeg sistema
- Primenite osnovne principe projektovanja složenih sistema – modularnost, apstrakciju i skrivanje informacija
- Opišete složeni digitalni sistem u VHDL jeziku koristeći instanciranje modula
- Projektujete strukture za računanje kao digitalne sisteme
- Projektujete upravljačke jedinice za strukture za računanje
- Razumete različite tipove upravljačkih jedinica i izaberete odgovarajući tip za datu primenu
- Projektujete upravljačke jedinice koje izvršavaju određeni program na datoj strukturi za računanje.

Apstrakt i motivacija

Digitalni sistemi su pokazali svoju pravu moć i pregršt mogućnosti primene onog trenutka kada su počeli da se primenjuju za računanje. Mašina za računanje složenih matematičkih operacija ubrzo je evoluirala u mašinu koja je postala svačiji lični asistent – obaveštava nas, omogućuje nam da pričamo sa udaljenim osobama, priča sa nama, planira nam dan, zabavlja nas, upravlja našim kućnim budžetom i asistira nas na još veliki broj načina. Naravno, još uvek može da rešava parcijalne diferencijalne jednačine, ukoliko vam ikad to zatreba. Ova mašina je, svakako, vaš omiljeni lični računar, a mozak te mašine je procesor – univerzalna struktura za računanje. U ovoj vežbi ćemo se upoznati sa strukturama za računanje, naučićemo kako se one projektuju i kako se njima upravlja. Kroz jednostavan primer, implementiraćete vašu prvu strukturu za računanje. Nakon ove vežbe bićete pripremljeni da rešite konačni zadatak našeg predmeta – projektovanje univerzalne strukture za računanje, procesora.

TEORIJSKE OSNOVE

1. Opis složenih digitalnih sistema u VHDL jeziku

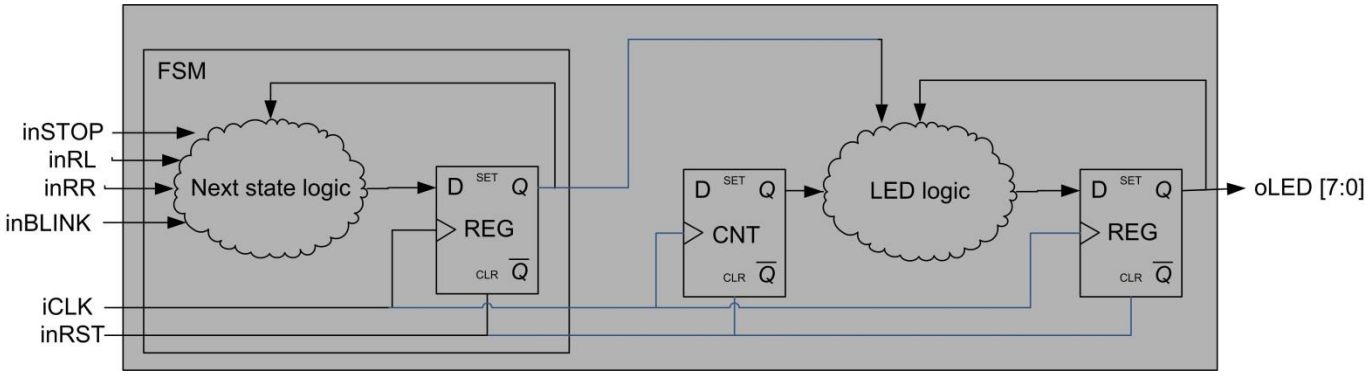
Osnovni principi prilikom projektovanja složenih sistema, ne samo u digitalnom svetu, su principi apstrakcije, modularnosti i skrivanja informacija. Principi modularnosti i apstrakcije znače da se neki deo sistema može „zatvoriti“ u crnu kutiju i koristiti samo posmatrajući njegove ulaze i izlaze (apstrakcija se odnosi na izdvajanje bitnih osobina datog dela sistema, odn. njegovog ponašanja). Kada koristimo deo sistema kao crnu kutiju, kažemo da se nalazimo na višem nivou apstrakcije, u odnosu na nivo u kome je projektovan sadržaj te crne kutije. Složeni sistemi se projektuju kombinovanjem ovih crnih kutija povezivanjem njihovih prolaza. Koristeći ovaj pristup, sistem se može posmatrati kroz nivoe apstrakcije – od najnižih (digitalna logička kola, flip-flopovi) do najviših (vrh hijerarhije sistema).

Princip skrivanja informacija znači da, nakon što deo sistema „zatvorimo“ u crnu kutiju, korišćenje tog dela sistema ne sme da zavisi od njegovog načina realizacije, odn. od onoga što se nalazi u crnoj kutiji. Razumevanje ovog principa zahteva veću pažnju – ovaj princip *ne zabranjuje* korisniku modula da poznaje arhitekturu sistema koji koristi. Naprotiv, princip olakšava rad korisniku, jer korisnik *ne treba da bude primoran* da poznaje arhitekturu sistema ako želi da ga iskoristi kao komponentu. Jedan od najčešćih razloga kašnjenja u projektima je potreba da se ulazi u tuđ sistem, razume njegov rad, modifikuje po potrebi i iskoristi u svom sistemu. **Ovo nikada ne bi trebao biti slučaj**! Nakon što je neki modul završen, niko ko ga koristi ne bi trebao biti primoran da razume njegovu unutrašnjost da bi ga koristio, a sve potrebne promene unutar njega treba da vrši autor modula. Ovaj princip obavezuje autora nekog modula da ga napravi potpunog, iskoristljivog isključivo putem svojih prolaza i specifikacije funkcionalnosti.

Pričajući u VHDL jeziku, kada završite projektovanje vašeg *entiteta (entity)*, on mora biti iskoristljiv kroz svoje *prolaze (ports)* i specifikaciju ponašanja za date ulaze. Korisnik modula, odn. projektant sistema na višem nivou apstrakcije, nikada ne treba biti primorana da razume *arhitekturu (architecture)* modula da bi ga koristila.

Ovi principi su potreban uslov da bi se projektovali složeni sistemi, pošto na visokim nivoima složenosti postaje nemoguće projektovati sistem vodeći računa o svakoj komponenti na najnižem nivou apstrakcije. Zamislite samo da se najnoviji procesori u celosti projektuju na nivou logičkih kola, koliko bi takvo projektovanje trajalo i da li bi uopšte bilo moguće? Mi smo već koristili ove principe u ranijim primerima, kada smo kombinacionu mrežu prikazali pomoću oblačića. Oblačić je u stvari crna kutija koja predstavlja skup logičkih kola koji realizuju funkciju koju mi želimo, no mi koristimo oblačić kao crnu kutiju ne vodeći računa o njegovom sadržaju, već samo posmatrajući njegove ulaze i izlaze.

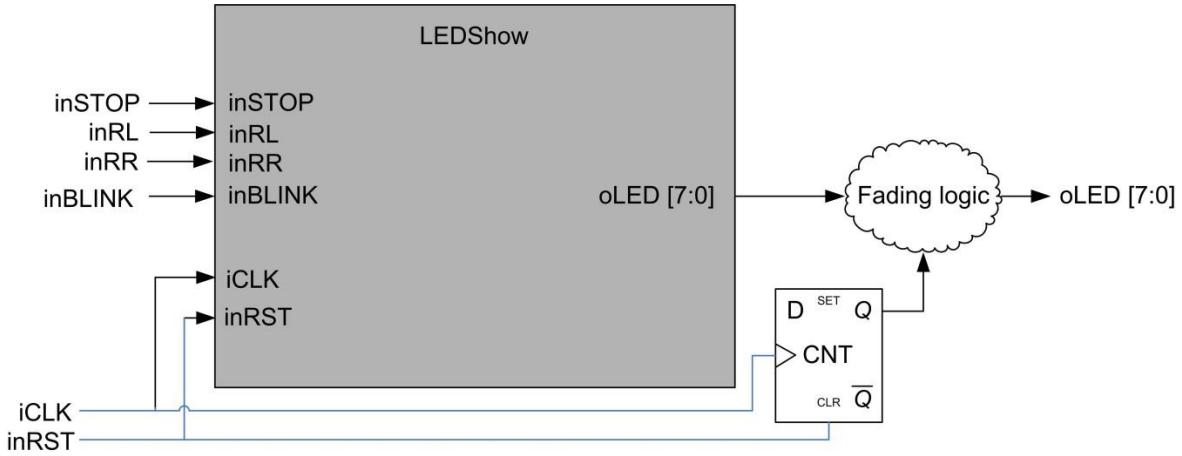
Slično možemo da uradimo i sa ranije realizovanim LED show sistemom. Kao što je prikazano na Slici 1-1, mi taj sistem možemo „zatvoriti“ u crnu kutiju i koristiti ga u složenijim sistemima samo preko njegovih ulaza/izlaza.



Slika 1-1. LED show sistem zatvoren u crnu kutiju

Nakon što pređemo na viši nivo apstrakcije, LED show sistem koristimo kao komponentu komunicirajući sa njom putem njenih ulaza/izlaza (odn. prolaza).

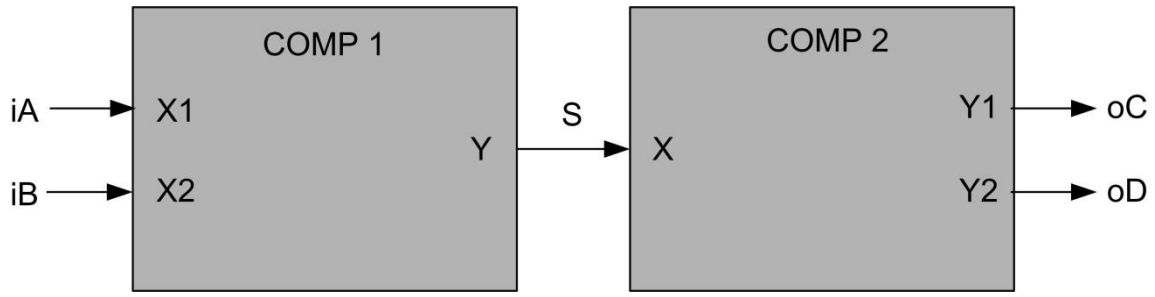
VHDL jezik i alat Xilinx ISE podržavaju ranije navedene principe prilikom projektovanja složenih digitalnih sistema omogućavajući definisanje više entiteta unutar jednog projekta i instanciranje jednog entiteta u drugom. Na primer, pretpostavimo da želimo da iskoristimo naš LED show sistem u složenijem sistemu u kome, sem LED show entiteta, želimo da postavimo još jedan brojač između izlaza LED show sistema i izlaza ka LED diodama. Na višem nivou apstrakcije, možemo da **instanciramo** komponentu LED show sistema i povežemo je na ostatak sistema, kompletirajući sistem kao na Slici 1-2. U terminologiji digitalnih sistema, opis sistema na najvišem nivou hijerarhije se naziva **top level**.



Slika 1-2. Top level složenog digitalnog sistema

Instanciranje komponenti se u VHDL jeziku vrši unutar opisa arhitekture sistema. Kako bi pokazali kako se vrši instanciranje komponenti, posmatrajmo za početak top level sistema na Slici 1-3 koji se sastoji od dve komponente COMP1 i COMP2. Neka komponenta COMP1 ima dva ulaza (X1, X2) i jedan izlaz (Y). Neka komponenta COMP2 ima jedan ulaz (X) i dva izlaza (Y1, Y2). Komponente treba povezati kao na Slici 3-3. Komponente COMP1 i COMP2 su definisane kao posebni entiteti, a njihovi opisi se nalaze u posebnim VHDL datotekama.

Listing 3-1 prikazuje arhitekturu najvišeg nivoa apstrakcije sistema gde se vidi način instanciranja komponenti u VHDL jeziku. Sistem ima dva ulaza (iA, iB) i dva izlaza (oC, oD) koji treba da se povežu da ulaze komponente COMP1 i izlaze komponente COMP2, respektivno. Izlaz komponente COMP1 povezati sa ulazom komponente COMP2. Komponente se međusobno vezuju žicama, odn. signalima u VHDL-u. U ovom primeru, neka to bude signal S.



Slika 1-3. Primer sistema sa dve komponente

Listing 1-1. Arhitektura sistema sa Slike 3-3

```
architecture Behavioral of MyTopLevel is

    component COMP1 is
        port ( X1 : in std_logic;
               X2 : in std_logic;
               Y : out std_logic );
    end component;

    component COMP2 is
        port ( X : in std_logic;
               Y1 : out std_logic;
               Y2 : out std_logic );
    end component;

    signal S : std_logic;

begin

    iCOMP1 : COMP1 port map (
        X1 => iA,
        X2 => iB,
        Y => S
    );

    iCOMP2 : COMP2 port map (
        X => S,
        Y1 => oC,
        Y2 => oD
    );

end Behavioral;
```

Instanciranje komponente zahteva sledeće tri akcije:

- treba dodati VHDL datoteku u kojoj je opisana komponenta u projekat (izborom **Project --> Add Source...**)
- unutar arhitekture opisa vrha hijerarhije, deklaracija komponente treba biti napisana bre ključne reči begin, ona je identična opisu entiteta u svemu sem u ključnoj reči na početku i kraju opisa
- Nakon ključne reči begin, komponenta se treba instancirati koristeći sintaksu iz Listinga 1-1. Prilikom navođenja veza, ime pre operatora obrnute dodele (**=>**) je naziv prolaza komponente, a ime nakon njega je naziv signala u vrhu hijerarhije koji se povezuje na odgovarajući prolaz.

Sve žice unutar vrha hijerarhije koje služe da bi se povezale dve komponente treba deklarisati kao interne signale.

2. Aritmetički elementi struktura za računanje

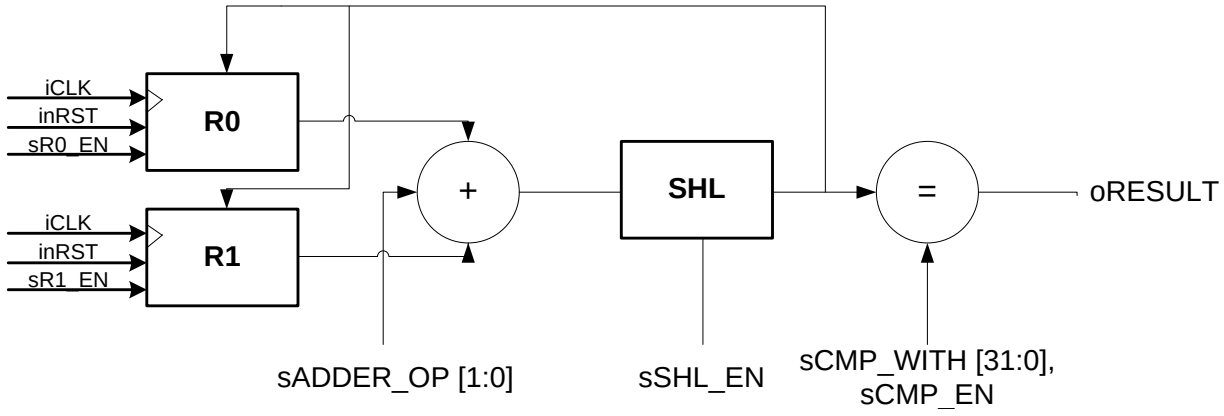
Strukture za računanje su digitalni sistemi projektovani s ciljem izvršavanja matematičkih aritmetičkih i logičkih operacija. Najkorišćenije komponente unutar struktura za računanje su standardne kombinacione i sekvencijalne mreže sa kojima ste već upoznati:

- Sabirači i oduzimači
- Množači i delioci
- Pomerači
- Komparatori
- Aritmetičko-logičke jedinice
- Multiplekseri
- Registri

Svaka struktura za računanje ima bar tri dela:

1. Registri opšte namene – za čuvanje izračunatih vrednosti
2. Aritmetičko-logički deo – za izvršavanje računanja
3. Upravljački deo – za upravljanje registrima i aritmetičko-logičkim delom strukture.

Primer dela strukture za računanje je dat na slici 2-1.



Slika 2-1. Primer aritmetičkog dela strukture za računanje

Struktura za računanje sa slike 2-1 se može analizirati kroz ranije definisane delove:

1. Registri opšte namene – R0 i R1 su dva 32-bitna registra, koriste se za čuvanje izračunatih vrednosti
2. Aritmetičko-logički deo se sastoji iz:
    a. 32-bitnog sabirača, koji prima dva 32-bitna operanda na ulazu i računa 32-bitni zbir prikazujući ga na izlazu (prenos se zanemaruje)
    b. 32-bitnog pomerača za jedno mesto u levo
    c. 32-bitnog komparatora, koji poredi dve vrednosti sa ulaza i na izlazu postavlja vrednost 1 ukoliko su ulazu jednaki, a 0 u suprotnom.
3. Upravljačka jedinica (nije prikazana na slici) kontroliše rad ostatka sistema:
    a. Registri se kontrolišu pomoću dva signala dozvole upisa (jedan po registru); ukoliko je signal dozvole aktivan (na vrednosti 1), u registar se upisuje vrednost sa ulaza, u suprotnom registar pamti staru vrednost
    b. Sabirač je kontrolisan 2-bitnim signalom sADDER_OP[1:0]
        i. Ukoliko je taj signal na vrednosti “00”, prvi operand sabirača je vrednost registra R0, a drugi operand je vrednost registra R1
        ii. Ukoliko je taj signal na vrednosti “01”, prvi operand sabirača je konstanta 1, a drugi operand je vrednost registra R1
        iii. Ukoliko je taj signal na vrednosti “10”, prvi operand sabirača je vrednost registra R0, a drugi operand je konstanta 0
        iv. Ukoliko je taj signal na vrednosti “11”, prvi operand sabirača je konstanta 0, a drugi operand je vrednost registra R1
    c. Pomerač je kontrolisan pomoću signala dozvole pomeranja sSHL_EN; ukoliko je signal dozvole aktivan (na vrednosti 1), pomerač pomera ulaznu vrednost za jedno mesto ulevo, a u suprotnom ne pomera ulaznu vrednost
    d. Komparator se kontroliše 32-bitnim operandom sCMP_WITH[31:0], u pitanju je vrednost sa kojom se poredi izlaz iz pomerača, kao i signalom dozvole poređenja sCMP_EN; ukoliko je signal dozvole neaktivan, komparator će na izlazu uvek davati vrednost 0 i neće vršiti poređenje.

Struktura za računanje neće vršiti nikakve operacije dok ne dobije aktivne kontrolne signale. Upravo pomoću ovih kontrolnih signala upravljačka jedinica može da kontroliše rad strukture za računanje – jednostavnim definisanjem vrednosti kontrolnih signala u svakom trenutku vremena. Okrećemo se sada projektovanju upravljačkih jedinica.

3. Upravljačke jedinice struktura za računanje

Upravljačka jedinica je centralna upravljačka stanica strukture za računanje. Njen zadatak je da definiše vrednosti kontrolnih signala čitavoj strukturi za računanje, za svaku komponentu. Upravljačke jedinice se mogu implementirati na dva načina:

- **Opšte upravljačke jedinice** se projektuju tako da mogu da izvršavaju proizvoljan skup operacija, tzv. program. Ove upravljačke jedinice primaju na ulazu *instrukcije*, tj. šifre koje govore upravljačkoj jedinici koju operaciju treba da izvrši u datom koraku. Instrukcije su često upamćene u memoriji iz koje se preuzimaju jedna za drugom i izvršavaju.

- **Specifične upravljačke jedinice** se projektuju tako da izvršavaju jedan predodređeni niz operacija. Ove upravljačke jedinice ne primaju instrukcije, nego prilikom svakog pokretanja generišu isti niz kontrolnih signala i time izvršavaju konstantan program. One su jednostavnije za projektovanje, ali ograničavaju mogućnosti sistema kojim upravljaju.

Prema tipu digitalnog sistema kojim je implementirana upravljačka jedinica, možemo razlikovati:

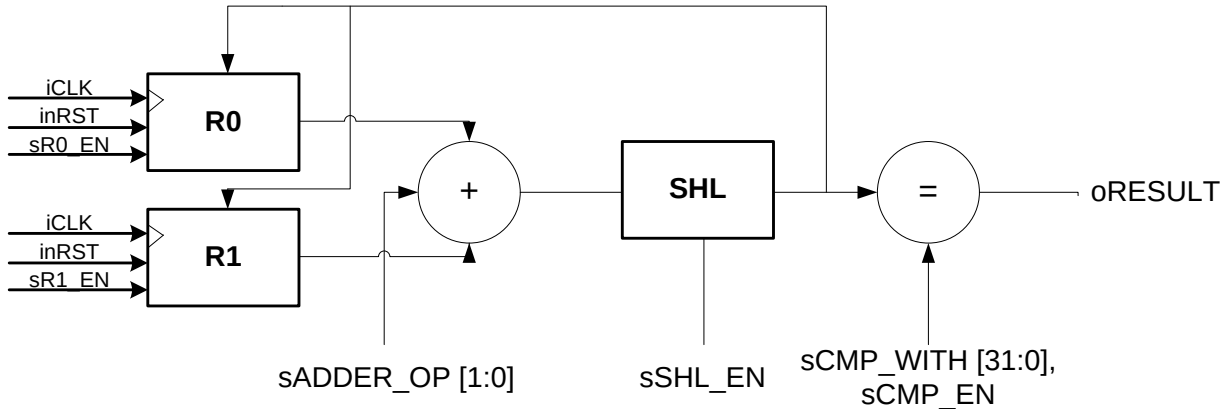
- **Kombinacione upravljačke jedinice** koje generišu kontrolne signale kombinacionom logikom, u zavisnosti od instrukcije koja se trenutno nalazi na njihovom ulazu. Ovim načinom se mogu implementirati jedino opšte upravljačke jedinice i one tada instrukcije izvršavaju uvek u jednom ciklusu takta sistema. *Specifične upravljačke jedinice se ne mogu implementirati na ovaj način, osim ukoliko želite da vaš program ima samo jednu instrukciju!*
- **Upravljačke jedinice zasnovane na automatu sa konačnim brojem stanja** prolaze kroz niz stanja i time izvršavaju niz instrukcija. Trenutno stanje automata i, ukoliko postoji, trenutna instrukcija, određuju izlaz automata, tj. kontrolne signale ka ostatku sistema.
    - Opšte upravljačke jedinice se implementiraju na ovaj način ukoliko instrukciju treba izvršiti u više faza, gde svaka faza odgovara stanju automata.
    - Specifične upravljačke jedinice se na ovaj način implementiraju veoma često, a tada jedno stanje automata odgovara jednoj instrukciji programa.
- **Mikroprogramske upravljačke jedinice** izvršavaju mikroprogram definisan u mikroprogramskoj memoriji unutar upravljačke jedinice. Mikroprogram definiše niz kontrolnih signala potrebnih da bi se izvršila data instrukcija i/ili program.
    - Opšte upravljačke jedinice se implementiraju na ovaj način ukoliko instrukciju treba izvršiti u više faza, a svaka faza odgovara jednoj mikroinstrukciji mikroprogama unutar upravljačke jedinice.
    - Specifične upravljačke jedinice se implementiraju na ovaj način ukoliko je željeni program zapamćen unutar upravljačke jedinice kao mikroprogram.

Na ovom predmetu nećemo implementirati mikroprogramske upravljačke jedinice, zbog njihove složenosti. U ovoj vežbi ćemo implementirati specifičnu upravljačku jedinicu i primer opšte kombinacione upravljačke jedinice.

ZADACI

4. Aritmetički deo strukture za računanje

Kao prvi korak, implementiraćemo strukturu za računanje sa slike 4-1.



Slika 4-1. Primer aritmetičkog dela strukture za računanje

Opišite ovu upravljačku jedinicu u jednoj VHDL datoteci. Jedini ulazi sistema su takt i reset, dok je jedini izlaz oRESULT.

Svi kontrolni signali trebaju biti definisani kao *interni signali*. Oni će biti definisani upravljačkom jedinicom koju ćemo naknadno dodati.

Referencirajte se na teorijski deo vežbe za objašnjenje funkcionisanja svake od komponenata.

5. Specifična upravljačka jedinica

Sada ćemo implementirati jednostavnu specifičnu upravljačku jedinicu koja izvršava sledeći kratki program:

```
R0 <- (R0 + R1) << 1
R1 <- R1 + 1
R1 == 4 ?
```

Odlučite šta treba da budu vrednosti upravljačkih signala da bi se implementirala svaka od instrukcija ovog programa. Implementirajte upravljačku jedinicu kao automat sa 3 stanja, jedan po instrukciji. Neka se automat beskonačno vrti između ova tri stanja.

Početne vrednosti registara (u resetu) postavite na R0 = 0 i R1 = 0.

Ovaj program računa vrednost sledeće sume:

$$\sum_{i=0}^{N-1} i * 2^{N-i}$$

Kada oRESULT dobije vrednost 1, tj. kada je R1 = 4, vrednost registra R0 je vrednost ove sume za N = 4. Promenom vrednosti sa kojom se poredi vrednost registra R1 moguće je lako ovaj program modifikovati tako da računa vrednost sume za proizvoljno N (naravno, ograničeno širinom registra).

Opciono pokušajte modifikovati upravljačku jedinicu tako da kada vrednost oRESULT dostigne vrednost 1, automat prelazi u četvrto, konačno stanje, u kome sistem ne radi ništa (NOP) i automat ostaje stalno u tom stanju.

6. Opšta upravljačka jedinica implementirana kao kombinaciona mreža

Sada ćemo modifikovati upravljačku jedinicu tako da ona prima instrukcije. Ovom modifikacijom upravljačka jedinica postaje opšta i može da realizuje proizvoljan program u okviru skupa instrukcija koje podržava.

Dodaćemo još jedan ulaz sistema, iINSTR[4:0], koji će predstavljati 5-bitnu instrukciju. Instrukcija treba da bude poslata ka upravljačkoj jedinici.

Podržaćemo skup od šest instrukcija, prikazanih u tabeli 6-1.

Tabela 6-1. Instrukcije podržane strukturom za računanje

Mnemonik	Kod	Instrukcija
ADD	000	dest <= R0 + R1
INC	001	R1 <= R1 + 1
MOV	010	dest <= src
CMP4	011	R1 == 4 ?
SHL	100	dest <= src << 1
ASH	101	dest <= (R0+R1) << 1

Instrukcija ima sledeći format:

- 3 bita – kod instrukcije
- 1 bit – odredišni registar (*dest*)
- 1 bit – izvorni registar (*src*)

Na primer, instrukcija 01001 je MOV instrukcija iz registra R1 u registar R0.

Implementirajte ovakvu upravljačku jedinicu podržavajući ovih šest instrukcija kroz kombinacionu mrežu – direktno definišući kontrolne signale u zavisnosti od trenutne instrukcije.

Simulirajte strukturu za računanje koristeći ove tri instrukcije iz prethodnog problema kao ulaze (definišite ih u test benchu).

ZAKLJUČAK

Čestitamo na implementaciji vaše prve strukture za računanje! Struktura za računanje koju ste upravo implementirali je veoma jednostavan primer procesora – strukture koja može da računa i odgovara na instrukcije. Ovaj procesor je veoma ograničen u svojim mogućnostima, ali sada ste spremni da projektujete konačni zadatak ovog predmeta – univerzalnu strukturu za računanje sa opštom upravljačkom jedinicom – RISC procesor.