POKAZNA VEŽBA 6

Automati sa konačnim brojem stanja

Potrebno predznanje

- Urađene pokazne vežbe 4 i 5
- Teorija automata sa konačnim brojem stanja

Šta će biti naučeno tokom izrade vežbe?

Nakon urađene vežbe, bićete u mogućnosti da:

- Projektujete digitalni sistem koji je zasnovan na automatu sa konačnim brojem stanja definisanim funkcijom prelaza i funkcijom izlaza
- Opišete digitalni sistem zasnovan na automatu sa konačnim brojem stanja u VHDL jeziku
- Opišete VHDL test bench za automat
- Kombinujete automate sa ostalim kombinacionim i sekvencijalnim komponentama u digitalnom sistemu

Apstrakt i motivacija

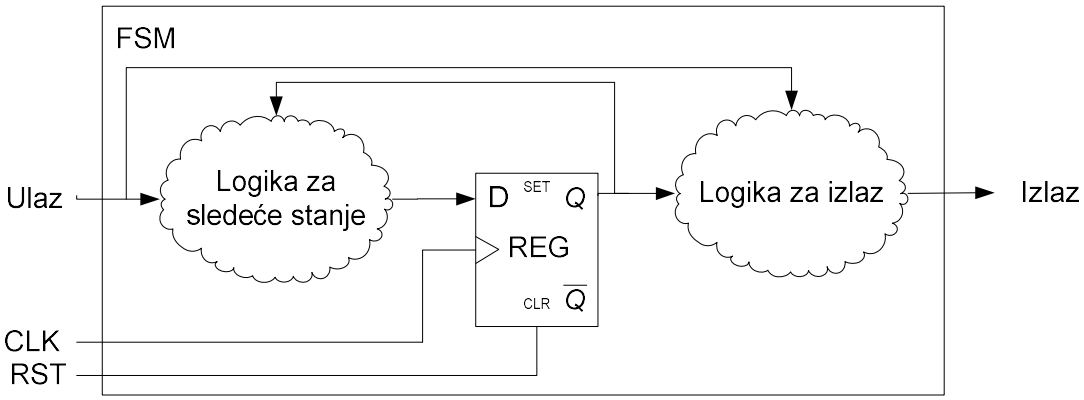
Veoma često digitalni sistem treba da izvršava sekvencu pre-definisanih zadataka. Da li je u pitanju žmigavac kod automobila, digitalni svetleći signali kao što je semafor ili sistem koji komunicira sa nekim drugim sistemom, sve ove operacije sadrže niz pre-definisanih koraka koji se trebaju izvršiti. Ovakvi sistemi imaju **konačan broj stanja** kroz koji treba da prolaze tokom svog životnog veka i u svakom stanju vrše neku operaciju. U teoriji digitalnih sistema, a i šire, ovim sistemima je dato ime: **automati sa konačnim brojem stanja** (eng. *finite state machine – FSM*). FSM se nalazi u svakom nešto složenijem digitalnim sistemu. U ovoj vežbi naučićete da projektujete i implementirate sistem zasnovan na FSM-u, kao i da ga opišete u VHDL jeziku i simulirate.

TEORIJSKE OSNOVE

1. Projektovanje automata sa konačnim brojem stanja

U digitalnim sistemima, sekvencijalne komponente su idealne za realizaciju automata sa konačnim brojem stanja. Vrednosti flip-flopova (registara) mogu predstavljati stanja automata, dok kombinaciona logika može da računa naredno stanje i izlaz automata. Ovo nas dovodi do jednog načina realizacije automata u digitalnom sistemu, koristeći sledeće komponente:

- **Registar** – memoriše **trenutno stanje** automata,
- **Kombinaciona mreža** koja određuje **naredno stanje** automata na osnovu trenutnog stanja i ulaza,
- **Kombinaciona mreža** koja određuje **izlaz** automata na osnovu trenutnog stanja i ulaza.



Slika 1-1. Automat

U zavisnosti od čega zavisi izlaz automata, razlikujemo:

- **Mealy**-eve automate, kod kojih izlaz zavisi od ulaza i trenutnog stanja automata,
- **Moore**-ove automate, kod kojih izlaz zavisi samo od trenutnog stanja automata.

U teoriji, automati se definišu pomoću sledećih šest veličina:

- **Skup vrednosti ulaza**, predstavljen svim vrednostima koje mogu imati ulazi automata,
- **Skup vrednosti izlaza**, predstavljen svim vrednostima koje mogu imati izlazi automata,
- **Skup vrednosti stanja**, predstavljen svim vrednostima koje može imati registar,
- **Početno stanje**, predstavljeno vrednošću registra u resetu,
- **Funkcija prelaza**, koja definiše kako se računa naredno stanje na osnovu trenutnog stanja i ulaza,
- **Funkcija izlaza**, koja definiše kako se računa izlaz na osnovu trenutnog stanja i, eventualno, ulaza.

U VHDL jeziku, automati se mogu opisati iz tri dela:

- Sekvencijalni proces koji opisuje registar,
- Kombinacioni proces koji računa naredno stanje,
- Kombinaciona dodela ili proces koji opisuje računanje izlaza automata.

Funkcija prelaza između stanja automata se najčešće definiše pomoću grafa prelaza stanja. Za opis logike za računanje narednog stanja najpogodnije je koristiti **CASE strukturu**.

Registar za memorisanje trenutnog stanja automata može da se realizuje kao interni signal tipa STD_LOGIC_VECTOR. No, tada stanja u kodu postaju nečitljiva, pošto imaju samo brojne vrednosti. Kao u programskim jezicima, VHDL za ovu svrhu omogućava da se definiše tip **enumeracije**. Vrednosti koje enumeracija dobije određuje alat za simulaciju/sintezu i nisu bitne za opis sistema. Listing 1-1 prikazuje kako se u VHDL-u opisuje tip enumeracije i definiše signal koji je tog tipa.

Listing 1-1. Deklaracija enumeracije u VHDL-u

```
architecture Behavioral of FSM is

    type <typeName> is (<IDLE_state>, <state1>, <state2>, ... , <stateN>);
    signal <signalName> : <typeName>;

begin
    -- telo arhitekture
end architecture;
```

Tip enumeracije **ne treba koristiti za definisanje ulaza i izlaza automata,** jer je enumeracija lokalno definisani tip, a ulazi i izlazi treba uvek da budu standardnog tipa kako bi sistemi koji komuniciraju sa našim sistemom preko tih ulaza i izlaza bili tipski kompatibilni.

NAPOMENA: Početno stanje sistema mora biti na prvom mestu u listi stanja.

Naredni Listing 1-2 prikazuje opšti oblik opisa automata u VHDL jeziku.

Listing 1-2. Automat u VHDL-u

```
architecture Behavioral of FSM is

    type tSTATE is ( <početno_stanje>, <state1>, <state2>, ... , <stateN>);
    signal sSTATE, sNEXT_STATE : tSTATE;

begin
--------- registar za pamćenje stanja ---------
process (iCLK, iRST) begin
    if (iRST = '1') then
        sSTATE <= <početno_stanje>;
    elsif (iCLK'event and iCLK = '1') then
        sSTATE <= sNEXT_STATE;
    end if;
end process;

--------- kombinacioni proces za definisanje sledećeg stanja -----
process (sSTATE, <ostali_ulazi>) begin
    case (sSTATE) is
        <logika_za_računanje_narednog_stanja tj. signala sNEXT_STATE>
    end case;
end process;

--------- uslovna dodela ili kombinacioni proces za definisanje izlaza
```

Provera ispravnosti rada sistema sa automatima se može opisati VHDL test bench-om, po sličnom principu kao i za opšte sekvencijalne mreže. Na početku treba resetovati sistem, a nakon toga menjati vrednosti ulaza prema željenom redosledu provere automata. Idealno, provera bi trebala da obuhvati sve moguće prelaze stanja automata.

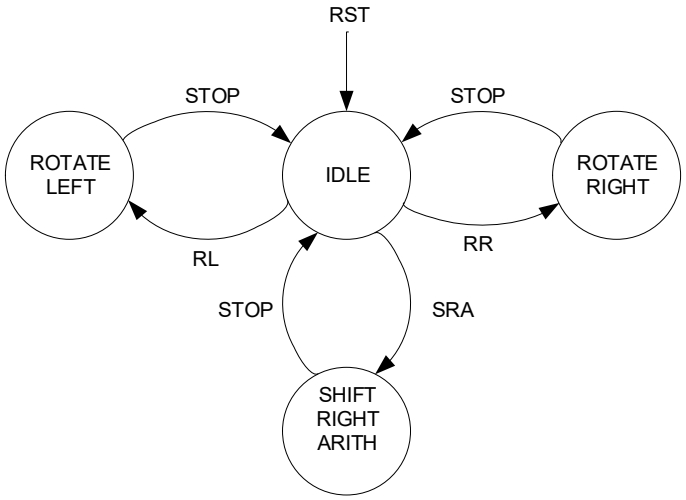
ZADACI

1. Automat pomerač

Vreme je da implementiramo naš prvi automat! Projektovaćemo sistem koji kontroliše 8-bitni oDATA izlaz, zasnovan na automatu sa konačnim brojem stanja:

- Stanja automata su: **IDLE**, **ROT_LEFT**, **ROT_RIGHT**, **SHIFT_RIGHT_ARITH**.
- Početno stanje je IDLE (**mora biti na prvom mestu u listi stanja**).
- Ulazi automata su: iRST, iCLK, iRL, iRR, iSRA, iSTOP.
- Izlaz automata je: oDATA[7:0].

Funkcija prelaza je data grafom na Slici 1-1.

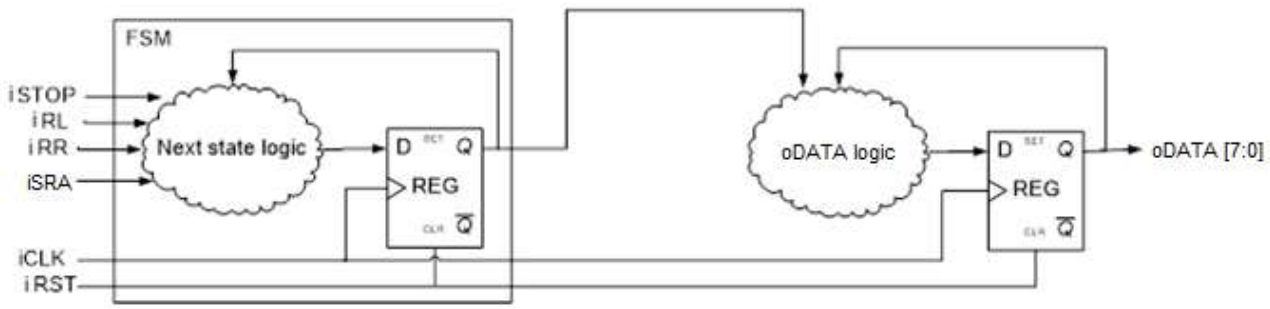


Slika 1-1. Graf prelaza stanja automata za primer 1

Funkcija izlaza je definisana na sledeći način:

- U stanju **IDLE**, vrednost izlaznog signala je ″**00110011**″.
- U stanju **ROT_LEFT**, trenutnu vrednost izlaznog signala rotiramo za jedan bit ulevo,
- U stanju **ROT_RIGHT**, trenutnu vrednost izlaznog signala rotiramo za jedan bit udesno,
- U stanju **SHIFT_RIGHT_ARITH**, trenutku vrednost izlaznog signala pomeramo za jedan bit udesno, aritmetički.

Blok dijagram sistema je dat na Slici 1-2.



Slika 1-2. Prikaz sistema

Primetite da u ovom sistemu postoje 2 registra i da se i na izlazu pamti vrednost u registru!

Sistem simulirati na sledeći način:

- Resetovati sistem 3.25 perioda takta
- Sistem zadržati u početnom stanju 10 perioda takta
- Aktivirati stanja (ROT_LEFT i ROT_RIGHT) po 5 perioda takta, a između stanja se vratiti na početno stanje IDLE po 10 perioda takta
- Zatim aktivirati stanje SHIFT_RIGHT_ARITH na 15 perioda takta i resetovati sistem 2 perioda takta

Nakon simulacije, potrebno je implementirati sistem na MAX10 platformi i povezati prolaze sistema na sledeće komponente i pinove, kao u Tabeli 1-1, a zatim proveriti ispravnost i ponašanje sistema.

Tabela 1-1. Pinovi za povezivanje automata-pomerača

Prolazi	Smer	Komponenta	Location	I/O standard
iCLK	input	i_CLK	PIN_H6	3.3-V LVTTL
iRST	input	i_RST	PIN_F1	3.3 V Schmitt Trigger
iRR	input	i_pb_right	PIN_E3	3.3 V Schmitt Trigger
iRL	input	i_pb_left	PIN_D1	3.3 V Schmitt Trigger
iSRA	input	i_sw[0]	PIN_M3	3.3 V Schmitt Trigger
iSTOP	input	i_pb_center	PIN_C2	3.3 V Schmitt Trigger
oDATA [7:0]	output	oLED[7:0]	D8, C10, C9, B10, A10, A11, A9, A8	3.3-V LVTTL

1.1. Dodatni zadatak – Brojač

Dopuniti logiku na izlazu dodavanjem brojača koji se uvećava svaki put kada je **vrednost na izlazu automata jednaka nuli**. Implementirati dodatak sistema kao posebnu komponentu koja preuzima 8-bitnu vrednost na ulazu, na osnovu koje kontroliše inkrementiranje brojača i formira 8-bitnu vrednost na izlazu. Proveriti ispravnost i ponašanje brojača u simulaciji.

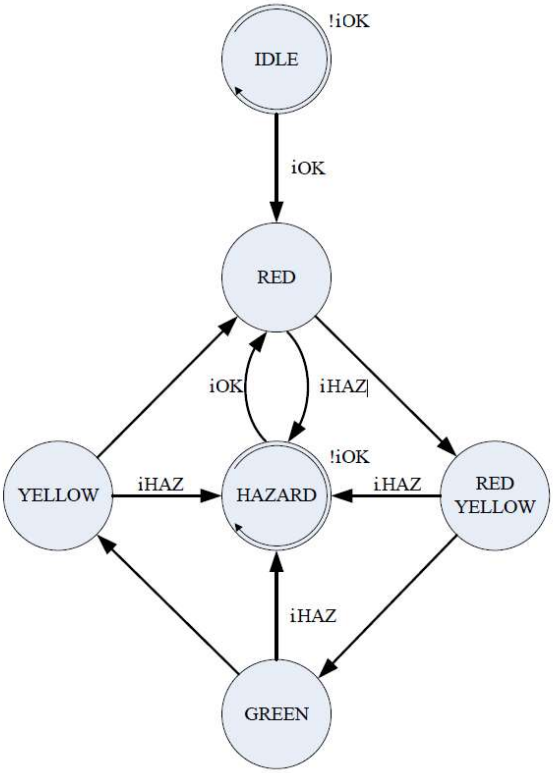
2. Semafor

Projektovati automat koji kontroliše rad jednostavnog semafora. Automat prolazi kroz 6 stanja: **IDLE**, **RED**, **RED_YELLOW**, **GREEN**, **YELLOW** i **HAZARD**. Početno stanje automata je **IDLE**.

Ulazi automata su:

- iOK – jednobitni ulaz pokreće sekvencu stanja ukoliko je semafor u stanju IDLE i nastavlja sa normalnim radom semafora ukoliko je bio u stanju HAZARD
- iHAZ – jednobitni ulaz koji iz bilo kog radnog stanja prebacuje semafor u stanje HAZARD.

Funkcija prelaza automata je data na Slici 2-1.



Slika 2-1. Graf prelaza stanja automata

U ovom delu zadatka, neka se prelazi između stanja automata dešavaju na svakoj rastućoj ivici takt signala. Funkcija prelaza je opisana grafom prelaza stanja:

- Iz početnog stanja **IDLE** automat prelazi u stanje **RED** kada se **iOK** postavi na vrednost '1'. Dok se ne aktivira iOK, automat ostaje u stanju **IDLE**.
- Kada pređe u stanje **RED**, bezuslovno prelazi u stanje **RED_YELLOW**, potom u stanje **GREEN**, zatim u stanje **YELLOW** i na kraju se vraća u stanje **RED** i tako nastavlja u krug.
- Iz bilo kog od ova pomenuta 4 stanja automat može preći u stanje **HAZARD** ukoliko je aktivan ulaz **iHAZ**. Kada automat pređe u stanje **HAZARD**, on ostaje u tom stanju sve dok **iOK** ne bude aktiviran, nakon čega prelazi u stanje **RED** i započinje novi ciklus.

Svi prelazi prikazani u grafu prelaza stanja sa strelicom bez pratećeg teksta su bezuslovni prelazi, odnosno automat iz trenutnog prelazi u sledeće stanje uvek, bez obzira na vrednost ulaza (sa izuzetkom, naravno **iRST** signala koji uvek automat prebacuje u početno stanje **IDLE**).

Tabela 2-1 prikazuje funkciju izlaza automata. Izlaze realizovati kao kombinacionu funkciju trenutnog stanja.

Tabela 2-1. Funkcija izlaza automata

State	R	Y	G
IDLE	0	0	0
RED	1	0	0
RED_YELLOW	1	1	0
GREEN	0	0	1
YELLOW	0	1	0
HAZARD	1	1	1

Sistem simulirati na sledeći način:

- Resetovati sistem 2.25 perioda takta
- Zadržati sistem u početnom stanju 2 perioda takta
- Pustiti semafor da radi tačno 5 punih ciklusa. Sledeći ciklus počinje kada semafor dođe ponovo u RED stanje.
- Pokvariti semafor tačno 25 perioda takta tj. zadržati se u stanju HAZARD.
- Nakon toga, ponovo pokrenuti semafor (bez aktiviranja reseta) i pustiti da radi 15 perioda takta.
- Resetovati sistem 2 perioda takta.

2.1. Implementacija semafora na MAX10 platformi

Nakon simulacije, potrebno je implementirati sistem na MAX10 platformi i povezati prolaze sistema na sledeće komponente i pinove, kao u Tabeli 2-2, a zatim proveriti ispravnost i ponašanje sistema.

Tabela 2-2. Pinovi za povezivanje semafora

Prolazi	Smer	Komponenta	Location	I/O standard
iCLK	input	i_CLK	PIN_H6	3.3-V LVTTL
iRST	input	i_RST	PIN_F1	3.3 V Schmitt Trigger
iOK	input	i_pb_down	PIN_C1	3.3 V Schmitt Trigger
iHAZ	input	i_pb_up	PIN_E1	3.3 V Schmitt Trigger
oRED	output	o_sem_red	PIN_H13	3.3-V LVTTL
oYELLOW	output	o_sem_yellow	PIN_E4	3.3-V LVTTL
oGREEN	output	o_sem_green	PIN_B1	3.3-V LVTTL

S obzirom da su promene između stanja realizovane na svaku rastuću ivicu takt signala, potrebno je produžiti vreme prelaska između stanja kako bi se one videle na platformi. Promenićemo sistem tako da prelazi između stanja traju **pola sekunde**. Kako bismo ovo ostvarili, dodaćemo štopericu u naš sistem koja će na svakih pola sekundi javljati automatu da može da promeni stanje pomoću **terminal count** signala. **Terminal count** signal štoperice treba da se iskoristi kao signal dozvole prelaza između stanja automata.

NAPOMENA: **Nemojte koristiti terminal count kao takt automata,** jer onda pravite više taktnih domena u sistemu što zahteva sinhronizacione korake sa kojima se još nismo upoznali. **Čitav sistem treba da radi na istom taktu, a terminal count treba iskoristiti kao signal dozvole u ostatku sistema.**

2.2. Naizmenično paljenje i gašenje “svetla” u stanju HAZARD

Kako bi stanje **HAZARD** bilo modelovano kao realni semafor, uvešćemo naizmenično paljenje i gašenje “svetla” u ovom stanju. Obično kada dođe do kvara semafora prikazano je samo treptajuće žuto svetlo (ili ne svetli ništa) umesto da bude upaljeno istovremeno i crveno i žuto i zeleno svetlo. Kada se automat nađe u ovom stanju, **pola sekunde** automat treba da se nađe u stanju gde je samo komponenta koja odgovara žutoj boji aktivna, a zatim **drugu polovinu sekunde** vrednost izlaznog signala treba da bude 0 (ne “svetli” ni jedna boja). Ovaj proces treba ponavljati sve dok **iOK** ulaz ne postane aktivan.

Postoji nekoliko načina da realizujete ovu funkcionalnost. Jedno rešenje je da ubacite novo pomoćno stanje i odgovarajući izlaz za njega. Primer drugog rešenja je da ubacite dodatnu kombinacionu logiku u funkciji izlaza za stanje **HAZARD**. Ukoliko Vam padne na pamet nešto treće, slobodno implementirajte na taj način.

ZAKLJUČAK

Upravo ste projektovali vaš prvi digitalni sistem zasnovan na automatu sa konačnim brojem stanja. Pored toga što ste naučili da u VHDL jeziku opišete i implementirate automat koji je unapred definisan, naučili ste kako da koristite automat kao komponentu unutar složenijeg digitalnog sistema. Automati su često korišteni u digitalnim sistemima zato što su korisni za izvršenje sekvence operacija u sistemu. U ostatku ovog predmeta, automate ćemo koristiti kao gradivni element procesora, u njegovom upravljačkom delu.