POKAZNA VEŽBA 6

Automati sa konačnim brojem stanja

Potrebno predznanje

- Urađena pokazna vežba 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
- 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.

Apstrakt i motivacija

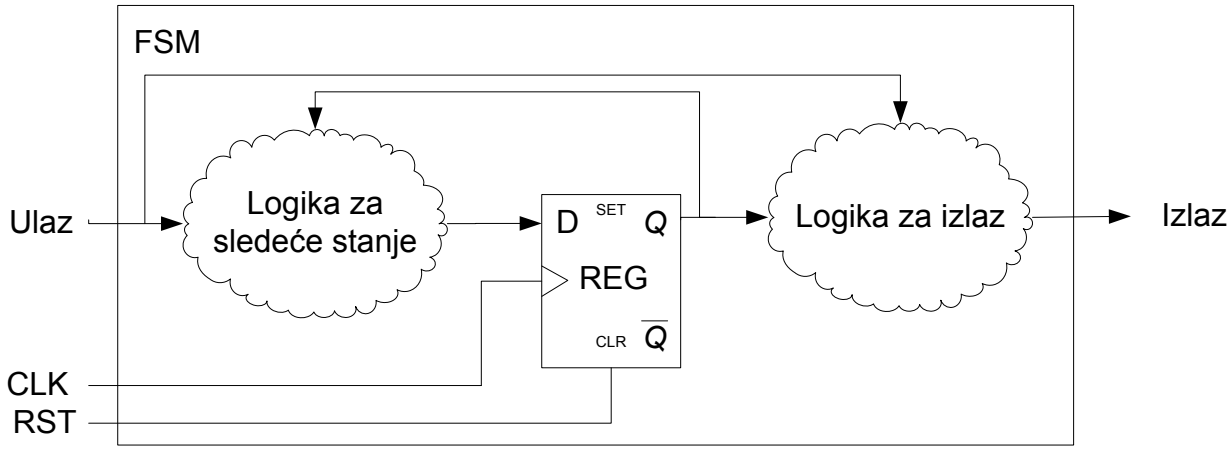
Veoma često digitalni sistem treba da izvršava sekvencu pre-definisanih zadataka. Da li je u pitanju žmigavac kod automobila, digitalni svetleći signali kao što je semafor ili sistem koji komunicira sa nekim drugim sistemom, sve ove operacije sadrže niz pre-definisanih koraka koji se trebaju izvršiti. 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 sistema 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 sistema, 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 računa **naredno stanje** automata na osnovu trenutnog stanja i ulaza,
- Kombinaciona mreža koja računa **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.

Funkcija prelaza automata se najčešće definiše pomoću **grafa prelaza stanja**. Primer će biti dat u narednom delu vežbe.

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

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

Listing 1-1 na narednoj strani daje opšti oblik opisa automata u VHDL jeziku. Za opis logike za računanje narednog stanja najpogodnije je koristiti CASE strukturu.

Listing 1-1. Automat u VHDL-u

```
process (<ulazi>, sSTATE) begin
    case (sSTATE) is
        <logika_za_racunanje_narednog_stanja, dodeljuje u sNEXT_STATE>
    end case;
end process;

process (iCLK, inRST) begin
    if (inRST = '0') then
        sSTATE <= <pocetno_stanje>;
    elsif (iCLK'event and iCLK = '1') then
        sSTATE <= sNEXT_STATE;
    end if;
end process;

<uslovna_dodela_ili_proces_za_racunanje_izlaza>
```

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 1-2 prikazuje kako se u VHDL-u opisuje tip enumeracije i definiše signal koji je tog tipa.

Listing 1-2. Deklaracija enumeracije u VHDL-u

```
architecture Behavioral of MySystem is

    type <typeName> is (<enumerationNames>);
    signal <signalName> : <typeName>;

begin
...
```

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.

Provera sistema sa automatima se može opisati VHDL testbenchom po sličnom principu kao i za opšte sekvencijalne mreže. U 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.

2. Opis složenih digitalnih sistema u VHDL jeziku

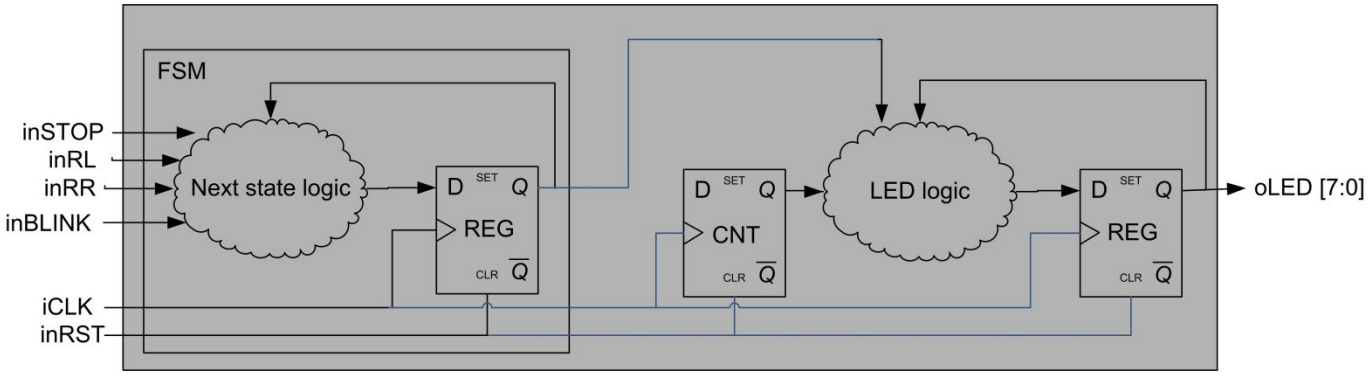
Osnovni principi prilikom projektovanja složenih sistema, ne samo u digitalnom svetu, su principi apstrakcije, modularnosti i skrivanja informacija. Principi modularnosti i apstrakcije znače da se neki deo sistema može „zatvoriti“ u crnu kutiju i koristiti samo posmatrajući njegove ulaze i izlaze (apstrakcija 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.

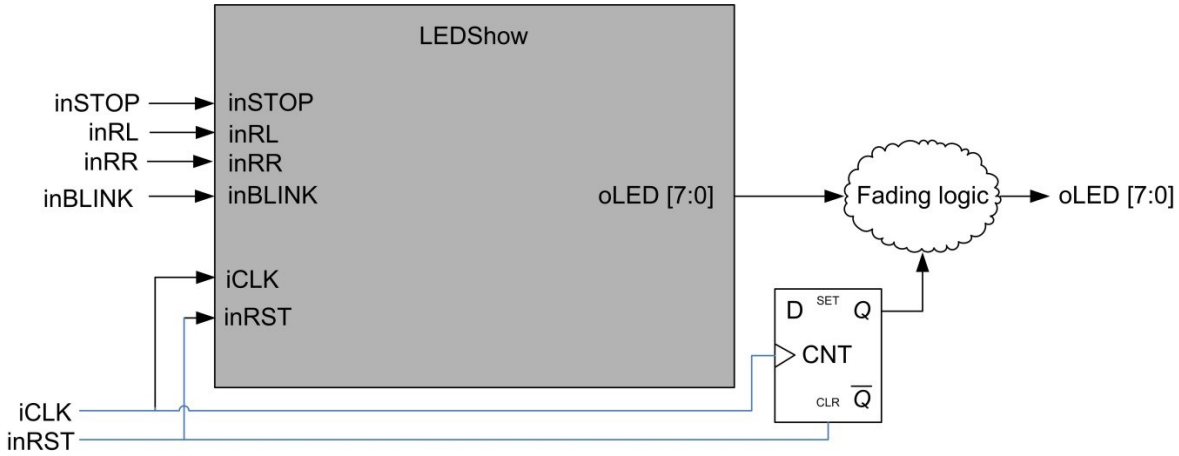
Slično možemo da uradimo i sa gotovim LED show sistemom koji ćemo implementirati kao prvi zadatak ove vežbe. Kao što je prikazano na Slici 2-1, mi taj sistem možemo „zatvoriti“ u crnu kutiju i koristiti ga u složenijim sistemima samo preko njegovih ulaza/izlaza.



Slika 2-1. 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).

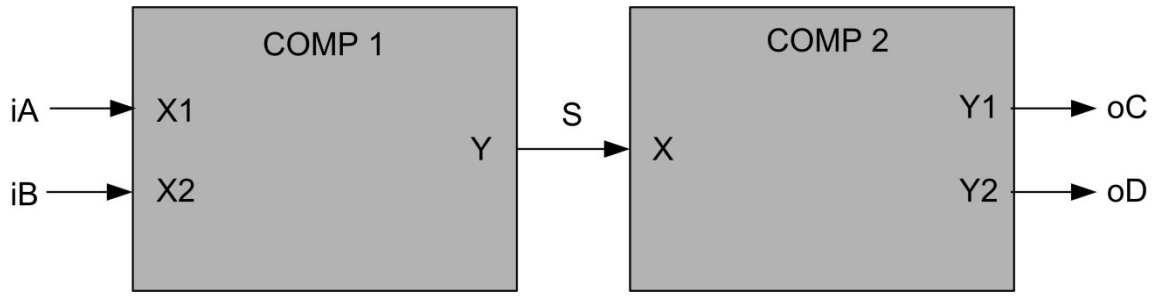
VHDL jezik i alat Xilinx ISE podržavaju ranije navedene principe prilikom projektovanja složenih digitalnih sistema omogućavajući definisanje više entiteta unutar jednog projekta i instanciranje jednog entiteta u drugom. Na primer, pretpostavimo da želimo da iskoristimo naš LED show sistem u složenijem 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 2-2. U terminologiji digitalnih sistema, opis sistema na najvišem nivou hijerarhije se naziva **top level**.



Slika 2-2. 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 2-3. Komponente COMP1 i COMP2 su definisane kao posebni entiteti, a njihovi opisi se nalaze u posebnim VHDL datotekama.

Listing 2-1 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 2-3. Primer sistema sa dve komponente

Listing 2-1. Arhitektura sistema sa Slike 2-3

```
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 2-1. 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.

ZADACI

3. LED Show

Vreme je da implementiramo naš automat! Projektovaćemo sistem koji kontroliše LED diode na E2LP platformi zasnovan na automatu sa konačnim brojem stanja.

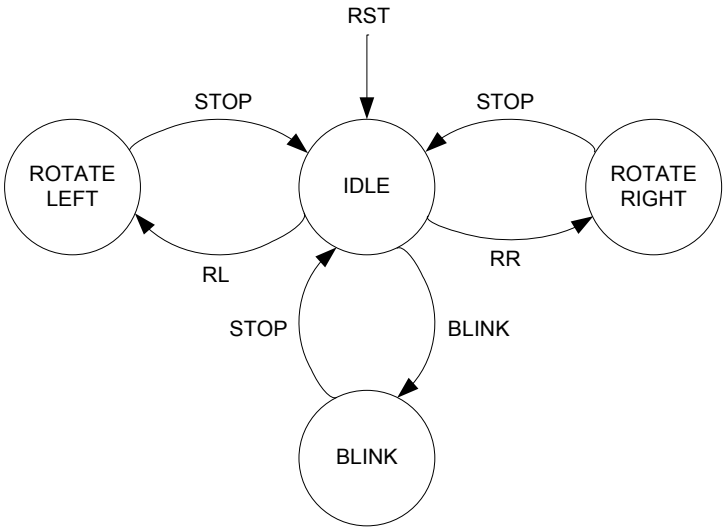
Stanja automata su: IDLE, ROTATE_LEFT, ROTATE_RIGHT, BLINK.

Početno stanje je IDLE.

Ulazi automata su: inSTOP, inRL, inRR, inBLINK.

Izlaz automata je: oLED.

Funkcija prelaza je data grafom na slici 2-1.

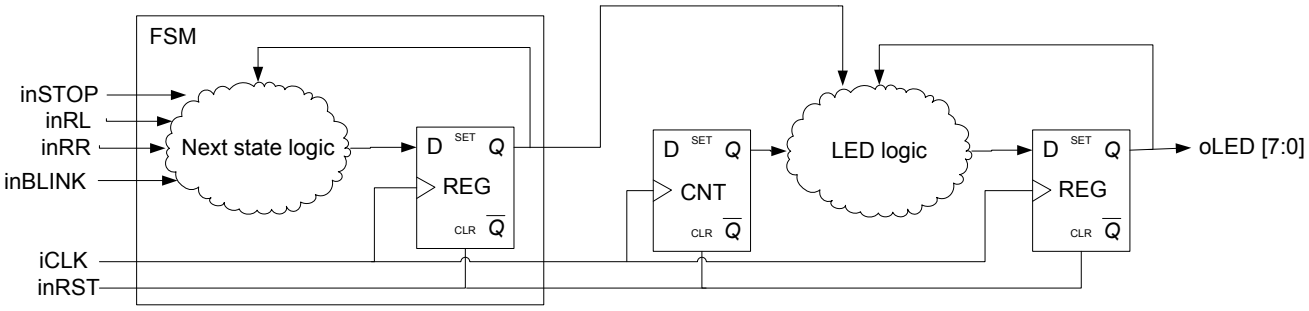


Slika 2-1. Graf prelaza stanja automata za LED show

Funkcija izlaza je definisana na sledeći način:

- U stanju IDLE, upaljena je samo dioda LED0.
- U stanju ROTATE_LEFT, jedna upaljena dioda rotira ulevo,
- U stanju ROTATE_RIGHT, jedna upaljena dioda rotira udesno,
- U stanju BLINK, diode se naizmenično pale/gase.

Blok dijagram sistema je dat na slici 2-2. Rotiranje i naizmenično paljenje i gašenje dioda treba da se dešavaju svaki sekund.



Slika 2-2. LED show sistem

4. Druga komponenta

Dopunićemo logiku na izlazu prema LED diodama gušenjem signala, tako da:

- Prvih 200 ms, diode koje treba da budu upaljene, to budu sa četvrtinom intenziteta,
- Od 200 ms do 400 ms, diode budu upaljene polovinom intenziteta,
- Od 400 ms do 600 ms, diode budu upaljene celim intenzitetom,
- Od 600 ms do 800 ms, diode budu upaljene polovinom intenziteta,
- Od 800 ms 1000 ms, diode budu upaljene četvrtinom intenziteta.

Intenzitet se definiše tako što se dioda periodično pali/gasi sa faktorom ispune jednakom potrebnom procentu intenziteta osvetljenja.

Implementirati dodatak sistema kao posebnu komponentu koja preuzima 8-bitnu vrednost na ulazu, guši je i formira 8-bitnu vrednost na izlazu. Spojiti dve komponente u vrhu hijerarhije sistema.

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.