A4. Auditorna vežba 4: Automati

Monday, 7 December 2020 12:22

Automat sa konačnim brojem stanja

★ **Definicija**. Automat sa konačnim brojem stanja je sekvencijalni sistem koji definišemo pomoću sledećih šest komponenti:

- **Skup stanja (S)**. Skup svih vrednosti koje može da ima dati sekvencijalni sistem; vrednost predstavlja stanje automata.
- **Skup ulaza (U)**. Skup svih vrednosti koje mogu da se pojave na ulazu datog sekvencijalnog sistema.
- **Skup izlaza (I)**. Skup svih vrednosti koje mogu da se pojave na izlazu datog sekvencijalnog sistema.
- **Početno stanje (S_0).** Vrednost koju dati sekvencijalni sistem ima na početku rada sistema; uvek važi da je $S_0 \in S$.
- **Funkcija prelaza (f_P).** Funkcija koja definiše naredno stanje (odn. vrednost) kao funkciju trenutnog stanja (vrednosti) i vrednosti na ulazu sistema.
- **Funkcija izlaza (f_I)**. Funkcija koja definiše vrednost izlaza sistema kao funkciju trenutnog stanja (vrednosti) i/ili vrednosti na ulazu sistema.

Funkcija prelaza se matematički definiše kao funkcija koja, za svaku kombinaciju trenutnog stanja i mogućih vrednosti na ulazu, definiše naredno stanje automata, odn. narednu vrednost koja će biti upisana u ovaj sekvencijalni sistem.

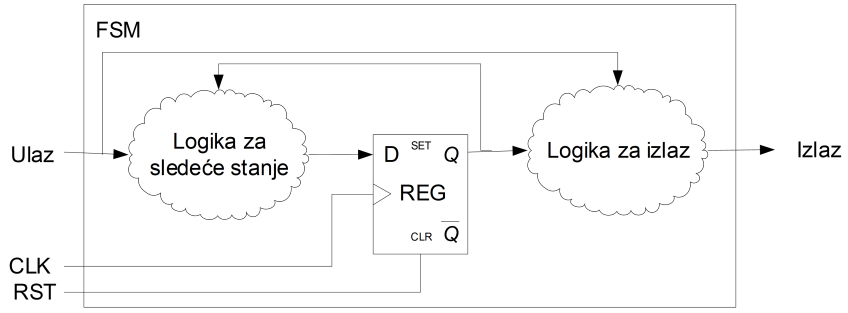
$$f_P\colon\ (S, U) \to S$$

Funkcija izlaza se matematički definiše kao funkcija koja, za svaku kombinaciju trenutnog stanja i mogućih vrednosti na ulazu, definiše vrednost izlaza automata. **Izlaz uvek zavisi od stanja automata**, a može ili ne mora da zavisi od ulaza.

★ U zavisnosti od toga da li izlaz automata zavisi od ulaza ili ne, razlikujemo dva tipa automata:

- **Mealy**-ev automat, kod kojeg izlaz **zavisi** od ulaza, dakle: $f_I\colon\ (S, U) \to I$
- **Moore**-ov automat, kod kojeg izlaz **ne zavisi** od ulaza, dakle: $f_I\colon\ S \to I$.

★ **Implementacija** automata u digitalnim sistemima je data sledećom šemom.



Sistem se sastoji iz sledećih delova:

- **Registar**, čija vrednost predstavlja trenutno stanje automata (S); na početku rada sistema (posle resetovanja), početna vrednost registra je početno stanje automata (S_0);
- **Kombinaciona mreža na ulazu registra**, koja računa funkciju prelaza (f_P) automata;
- **Kombinaciona mreža na izlazu registra**, koja računa funkciju izlaza (f_I) automata.

★ **Opis automata u VHDL-u** oslikava tri komponente koje se nalaze u fizičkoj implementaciji automata:

- **Sekvencijalni proces** za opis registra;
- **Kombinacioni proces** za opis funkcije prelaza (kod automata, standard je da se funkcija promene vrednosti opisuje u posebnom kombinacionom procesu);
- **Kombinacioni proces (ili uslovna dodela)** za opis funkcije izlaza.

Za definisanje stanja automata koristi se **korisnički definisan tip signala** (enumeracija) u kojoj kao vrednosti definišemo imena stanja automata. Tipično se definišu dva signala tog tipa, jedan predstavlja registar koji sadrži trenutno stanje automata (sSTATE), a drugi predstavlja signal između funkcije prelaza i registra, koji predstavlja naredno stanje automata (odn. rezultat računanja funkcije prelaza).

```
-- definisanje tipa signala, a zatim i signala koji predstavlja registar u automatu
-- ovaj deo opisa se piše u arhitekturi, pre ključne reči "begin" (na istom mestu gde i signali)
type tSTATE is (STATE_0, STATE_1, STATE_2);      -- primer enumeracije sa ukupno 3 naziva za stanja
signal sSTATE : tSTATE;
signal sNEXT_STATE : tSTATE;
```

Sledi šablon za VHDL opis automata.

```
-- registar
process (iCLK, iRST) begin
    if (iRST = '1') then
        sSTATE <= STATE_0;                          -- pretpostavimo da je STATE_0 početno stanje
    elsif (iCLK'event and iCLK = '1') then
        sSTATE <= sNEXT_STATE;                      -- registar samo preuzima rezultat iz funkcije prelaza
    end if;
end process;
```

```
-- funkcija prelaza
process (sSTATE, iX) begin
    -- opis funkcije prelaza, najčešće pomoću CASE strukture; sledi primer funkcije prelaza
    case (sSTATE) is
        when STATE_0 => sNEXT_STATE <= STATE_1;     -- primer bezuslovnog prelaza
        when STATE_1 =>
            if (iX = '1') then                      -- primer uslovnog prelaza
                sNEXT_STATE <= STATE_2;
            else
                sNEXT_STATE <= STATE_1;
            end if;
        when others => sNEXT_STATE <= STATE_0;      -- poslednji slučaj je when_others zbog potpune definisanosti
    end case;
end process;

-- funkcija izlaza
oY <= '1' when sSTATE = STATE_2 else '0';           -- primer funkcije izlaza
```

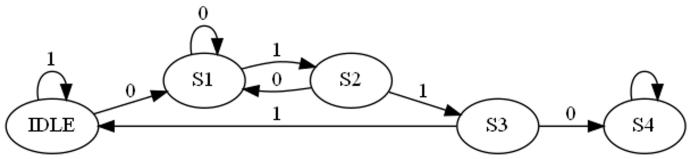
Primer automata

Kao primer automata posmatrajmo **detektor sekvence 0110**. Ovaj sistem, zasnovan na automatu, treba da na izlazu postavi vrednost 1 kada se na jednobitnom ulazu pojavi tražena sekvenca bita (0110). U suprotnom, na izlazu treba biti vrednost 0.

Definisaćemo svih šest komponenti automata.

Skup stanja: $\mathrm{S} = \{IDLE,\ S1,\ S2,\ S3,\ S4\}$. Stanje S_n znači da je do tada primeljno prvih *n* bita iz sekvence. Stanje ćemo predstaviti signalom *sSTATE*.
Skup ulaza: $\mathrm{U} = \{0,\ 1\}$. Ulaz je jednobitan, pa imamo ukupno dve moguće vrednosti. Ulaz ćemo predstaviti signalom *iX*.
Skup izlaza: $\mathrm{I} = \{0,\ 1\}$**.** Izlaz je takođe jednobitan, pa imamo ukupno dve moguće vrednosti. Izlaz ćemo predstaviti signalom *oY*.
Početno stanje: $S_0 = \mathrm{IDLE}$.

Funkcija prelaza se, na primer, može definisati pomoću grafa prelaza stanja.



Funkcija izlaza: definisana je tekstom zadatka. $\mathrm{oY} = \begin{cases} 1, & sSTATE = S4 \\ 0, & u\ suprotnom \end{cases}$. Funkcija izlaza se može definisati i na grafu prelaza stanja, definišući vrednost izlaza pored svakog izlaza. Primećujemo da je ovo automat Moore tipa, pošto izlaz zavisi samo od trenutnog stanja, a ne i od ulaza.

Funkcija prelaza i izlaza se može definisati i istinitosnom tablicom, pod uslovom da se dogovorimo o vrednostima koje će predstavljati svako stanje iz skupa stanja. Postoji više načina na koji možemo kodovati stanja, a jedan od najjednostavnijih je obično binarno kodovanje: da stanja numerišemo redom od 0 do N-1, gde je N ukupan broj stanja.

Primer mogućeg kodovanja stanja:

STANJE	KOD
IDLE	000
S1	001
S2	010
S3	011
S4	100

Funkciju prelaza i izlaza tada možemo predstaviti i sledećom istinitosnom tablicom:

STANJE	ULAZ	f_P	f_I
000	0	001	0
000	1	000	0
001	0	001	0
001	1	010	0
010	0	001	0
010	1	011	0
011	0	100	0
011	1	000	0
100	0	100	1
100	1	100	1

Funkcija izlaza ne zavisi od ulaza - to se jasno vidi iz tabele; prilikom realizacije funkcije izlaza, ne moramo ulaz uzimati u obzir.

Iz navedene tablice istinitosti možemo na klasičan način završiti projektovanje automata:

- definišemo 3 funkcije za svaki bit (odn. svaku kolonu u tabeli) rezultata funkcije prelaza;
- definišemo 1 funkciju za izlaz automata.

Funkcije otkrivamo postupkom minimizacije, nakon čega sintetišemo kombinacione mreže za funkciju prelaza i funkciju izlaza. Između njih stavljamo 3-bitni registar za čuvanje trenutnog stanja.

101	0	---	0
101	1	---	0
110	0	---	0
110	1	---	0
111	0	---	0
111	1	---	0

Međutim, kada automat opisujemo u VHDL-u ne moramo voditi računa o načinu kodovanja stanja, kao ni o jednačinama funkcije prelaza i izlaza. Potrebno je samo da definišemo enumeraciju za stanja i opišemo funkcije koristeći standardne VHDL konstrukcije za opis kombinacionih mreža.

VHDL opis primera automata.

```
-- registar
process (iCLK, iRST) begin
    if (iRST = '1') then
        sSTATE <= IDLE;                             -- početno stanje
    elsif (iCLK'event and iCLK = '1') then
        sSTATE <= sNEXT_STATE;
    end if;
end process;

-- funkcija prelaza
process (sSTATE, iX) begin
    case (sSTATE) is
        when IDLE => if (iX = '0') then             -- u svakom stanju, ako stigne odgovarajući bit, prelazimo u
                         sNEXT_STATE <= S1;         -- sledeće stanje, u suprotnom se vraćamo u stanje prema grafu
                     else
                         sNEXT_STATE <= IDLE;
                     end if;
        when S1 => if (iX = '1') then
                       sNEXT_STATE <= S2;
                   else
                       sNEXT_STATE <= S1;
                   end if;
        when S2 => if (iX = '1') then
                       sNEXT_STATE <= S3;
                   else
                       sNEXT_STATE <= S1;
                   end if;
        when S3 => if (iX = '0') then
                       sNEXT_STATE <= S4;
                   else
                       sNEXT_STATE <= IDLE;
                   end if;
        when S4 => sNEXT_STATE <= S4;               -- konačno stanje automata (bezuslovni prelaz)
        when others => sNEXT_STATE <= IDLE;         -- potrebno zbog potpunog opisa kombinacione mreže
    end case;
end process;

-- funkcija izlaza
oY <= '1' when sSTATE = S4 else '0';                -- izlaz je 1 samo u konačnom stanju automata
```

Minimizacija automata

< Primer zadatka sa opisom postupka rešavanja je dat u posebnoj datoteci. >