SVEUČILIŠTE U SPLITU
Odjel za stručne studije

Julije Ožegović

DIGITALNA I MIKROPROCESORSKA TEHNIKA

UPUTE ZA LABORATORIJSKE VJEŽBE

IV izdanje

Split, prosinac 2003.

SADRŽAJ

PREDGOVOR

Laboratorij Digitalne i mikroprocesorske tehnike organiziran je sa ciljem da studentima omogući stjecanje praktičnih iskustava na području sinteze digitalnih sklopova i razvoja programske podrške za mikrokontrolere. U okviru laboratorijskih vježbi predviđeno je upoznavanje s elementima za realizaciju digitalnih sklopova, te primjena metoda sinteze obrađenih na predavanjima i auditornim vježbama.

Laboratorij se u ovoj fazi sastoji od 12 vježbi koje se obavljaju na modelu DELAB1 i razvojnom sustavu STK200.

U uvodu je dat pregled osnovnih svojstava algebre logike i Booleovih funkcija.

Vježba 1. omogućava upoznavanje s osnovnim svojstvima različitih tehnologija izrade digitalnih integriranih krugova.

Vježba 2. bavi se minimizacijom Booleovih funkcija i sintezom digitalnih sklopova primjenom NI i NILI logičkih vrata.

Vježba 3. bavi se sumom po modulu, te omogućava uvid u sumatore, komparatore i generatore pariteta. Zadatak na vježbi je izraditi koder, dekoder i korektor za Hammingov kod.

Vježba 4. uvodi kombinacione sklopove srednje skale integracije: multipleksere, demultipleksere/dekodere i enkodere. Zadatak je realizirati Booleovu funkciju primjenom multipleksera i demultipleksera.

Vježba 5. bavi se sintezom logičkih sklopova primjenom programabilnih struktura kao što su EPROM i GAL.

Vježba 6. uvodi studenta u osnovne sekvencijalne sklopove, bistabile. Uz mjerenja na standardnim bistabilima, zadatak na vježbi je realizirati opći bistabil.

Vježbe 7., 8. i 9. bave se konačnim diskretnim automatima. Sinteza se obavlja primjenom logičkih vrata i programabilnih struktura.

Vježba 10. ilustrira primjenu algoritama kroz programski model Turingovog stroja.

Vježbe 11. i 12 odnose se na mikrokontroler AVR AT89S8515, za kojeg studenti trebaju pripremiti i izvesti jednostavan program, te se tako upoznati s osnovnim svojstvima arhitekture mikroprocesora i s razvojnim ciklusom programske podrške.

Razvoj laboratorija kretat će se ka sve većoj primjeni računala u sintezi i simulaciji digitalnih automata.

Split, prosinac 2000. dr.sc. Julije Ožegović

UVOD

Kod **analognih** sustava, informaciju prenosimo jednim električnim signalom, kod kojeg razlikujemo R razina. Kod **digitalnih** sustava, informaciju prenosimo s n električnih signala, kod kojih razlikujemo samo po dvije razine. Izabrali smo binarni brojevni sustav, a električni signali mogu zauzeti samo vrijednosti 0 i 1. Vrijedi uvjet kodiranja:

$$2^n \geq R \quad \text{ili} \quad n \geq \mathrm{ld}(R)$$

odnosno iz dimenzije dinamike informacijskog volumena prešli smo u dimenziju prostora (n signala, n vodiča):

$$2B \cdot \mathrm{ld}(R) \cdot T \cdot 1 = 2B \cdot 1 \cdot T \cdot n$$

Informaciju dakle prikazujemo binarnim brojem, odnosno kompleksijom (kodnom riječju) od n binarnih znamenki.

ALGEBRA LOGIKE:

Algebra logike je definirana kao struktura:

$$< x, \&, V, -, =, S >$$

To su Booleova varijabla x koja uzima vrijednosti 0 i 1 iz S, elementarni operatori konjunkcije (&), disjunkcije ($\vee$) i negacije (-), operator jednakosti, te skup Booleovih konstanti S=<0 i 1>. Operatori su definirani kao:

x_1 x_2	$x_1 \& x_2$	$x_1 \vee x_2$	$\overline{x}_1$
0 0	0	0	1
0 1	0	1	1
1 0	0	1	0
1 1	1	1	0

Vrijede postulati:

Zatvorenost: $x_1 \vee x_2 \in S \;;\; x_1 \& x_2 \in S$

Neutralni element: $x_1 \vee 0 = x_1 \;;\; x_1 \& 1 = x_1$

Komutativnost: $x_1 \vee x_2 = x_2 \vee x_1 \;;\; x_1 \& x_2 = x_2 \& x_1$

Distributivnost: $x_1 \vee (x_2 \& x_3) = (x_1 \vee x_2) \& (x_1 \vee x_3)$

$x_1 \& (x_2 \vee x_3) = (x_1 \& x_2) \vee (x_1 \& x_3)$

Komplementiranje: $x_1 \vee \overline{x}_1 = 1 \;;\; x_1 \& \overline{x}_1 = 0$

Asocijativnost: $x_1 \vee (x_2 \vee x_3) = (x_1 \vee x_2) \vee x_3$

$x_1 \& (x_2 \& x_3) = (x_1 \& x_2) \& x_3$

Osnovni teoremi su:

Apsorpcija: $x_1 \vee 1 = 1 \; ; \; x_1 \& 0 = 0$

Idepotentnost: $x_1 \vee x_1 = x_1 \; ; \; x_1 \& x_1 = x_1$

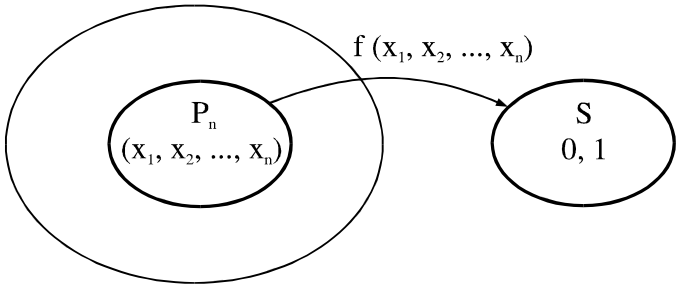
Dvostruka negacija: $\overline{\overline{x}}_1 = x_1$

De Morganovi teoremi: $\overline{x_1 \vee x_2} = \overline{x}_1 \& \overline{x}_2 \; ; \; \overline{x_1 \& x_2} = \overline{x}_1 \vee \overline{x}_2$

BOOLEOVA FUNKCIJA:

Korištenjem elemenata algebre logike pišemo algebarske izraze uvodeći operatorske veze među varijablama. Možemo kazati da je ukupna vrijednost izraza, kao zavisna varijabla, ovisna o uvrštenim vrijednostima ulaznih, nezavisnih varijabli. Dobili smo Booleovu funkciju, čija je karakteristika da su ulazne i izlazna varijabla Booleove varijable, tj. mogu poprimiti samo vrijednost 0 ili 1.

Booleova funkcija $f(x_1, x_2, x_3, ..., x_n)$ je preslikavanje skupa $P_n(x_1, x_2, ..., x_n)$, $n>0$, u skup konstanti $S=<0,1>$, gdje je $P_n(x_1, x_2, ..., x_n)$ skup svih kompleksija (kodnih riječi) varijabli x_1 .... x_n, još ga zovemo i nadskup skupa svih varijabli $X=<x_1, x_2, ..., x_n>$, slika 1.



Slika 1. - Booleova funkcija kao preslikavanje

Preslikavanje se definira za sve ili samo za neke kompleksije ulaznih varijabli. Nepotpuno specificirana funkcija ima smisla ako pretpostavimo da se određena kompleksija ulaznih varijabli u praksi neće nikada pojaviti na ulazu u funkciju. To su one kompleksije koje u postupku kodiranja nisu iskorištene. Za njih funkciju

nije potrebno definirati, odnosno možemo je definirati proizvoljno. Takve kompleksije nazivamo redundantnima.

Kako je $P_n(X)$ konačan, osnovni način zapisivanja Booleove funkcije je **tablicom istine**, u koju s lijeve strane upisujemo sve kompleksije u prirodnom binarnom nizu, a s desne strane upisujemo vrijednosti jedne ili više funkcija (koje su funkcije istih varijabli). Svakoj funkciji pripada jedan stupac desne strane tablice. Vrijednosti funkcije mogu biti iz skupa <0,1>, ili nedefinirane (R). Svaki redak označavamo rednim brojem i od 0 do 2^n-1, koji odgovara vrijednosti pripadne kompleksije varijabli promatrane kao prirodni binarni broj, slika 2.

i	x_1	x_2	y_1	y_2	y_3
0	0	0	0	1	1
1	0	1	1	0	1
2	1	0	1	1	0
3	1	1	0	0	1

Slika 2. - Tablica istine

Kod zapisivanja funkcije **algebarskim izrazom** koristimo potpuni disjunktivni ili potpuni konjunktivni normalni oblik. Njihovi osnovni dijelovi su minterm $m_i(x_1, x_2,\ldots x_n)$ i maksterm $M_i(x_1, x_2,\ldots x_n)$.

Minterm i-tog retka od n varijabli je konjunkcija svih n varijabli, gdje varijable koje u pripadnoj kompleksiji imaju vrijednost 1 u konjunkciju ulaze nenegirane, a one ostale ulaze negirane. m_i je jednak jedinici kada uvrstimo pripadnu kompleksiju, a jednak nuli za sve ostale.

Maksterm i-tog retka od n varijabli je disjunkcija svih n varijabli, gdje varijable koje u pripadnoj kompleksiji imaju vrijednost 0 u disjunkciju ulaze nenegirane, a one ostale ulaze negirane. M_i je jednak nuli kada uvrstimo pripadnu kompleksiju, a jednak jedinici za sve ostale.

Primijetimo mogućnost jednostavnog pisanja minterma i-tog retka tako da ispišemo redom sve varijable konjunktivno vezane, te nakon toga negiramo one koje u pripadnoj kompleksiji imaju vrijednost 0. Maksterm i-tog retka pišemo tako da ispišemo redom sve varijable disjunktivno vezane, te nakon toga negiramo one koje u pripadnoj kompleksiji imaju vrijednost 1. Za slučaj n=2 imamo:

m_0: $\overline{x}_1 \& \overline{x}_2$, m_1: $\overline{x}_1 \& x_2$, m_2: $x_1 \& \overline{x}_2$, m_3: $x_1 \& x_2$

M_0: $x_1 \vee x_2$, M_1: $x_1 \vee \overline{x}_2$, M_2: $\overline{x}_1 \vee x_2$, M_3: $\overline{x}_1 \vee \overline{x}_2$

Prednost je potpunih normalnih oblika što ih neposredno možemo ispisati iz tablice istine.

Potpuni disjunktivni normalni oblik (PDNO) je disjunktivna veza onih minterma za koje je vrijednost funkcije $T_i = 1$ (zbog $x\&0 = 0$ i $x \vee 0 = x$).

a) **PDNO**: $$f(x_1, x_2, \ldots, x_n) = \bigvee_{i=0}^{2^n-1} T_i \& m_i$$

Potpuni konjunktivni normalni oblika (PKNO) je konjunktivna veza onih maksterma za koje je vrijednost funkcije $T_i = 0$ (zbog $x \vee 1 = 1$ i $x\&1 = x$).

b) **PKNO**: $$f(x_1, x_2, \ldots, x_n) = \&_{i=0}^{2^n-1} (T_i \vee M_i)$$

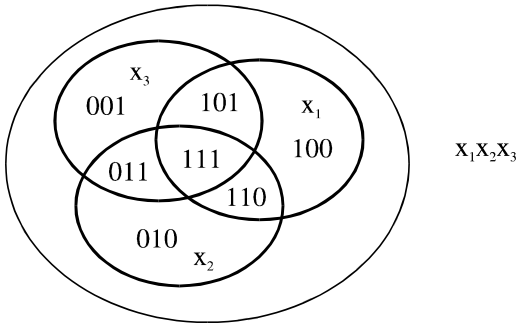
Funkcija koja sadrži sve minterme jednaka je jedinici, a funkcija koja sadrži sve maksterme jednaka je nuli.

$$\bigvee_{i=0}^{2^n-1} m_i = 1 \qquad\qquad \&_{i=0}^{2^n-1} M_i = 0$$

Primjenom De Morganovih teorema slijedi da su minterm i maksterm usko povezani. Veza je dana izrazima:

$$m_i = \overline{M}_i \qquad\qquad M_i = \overline{m}_i$$

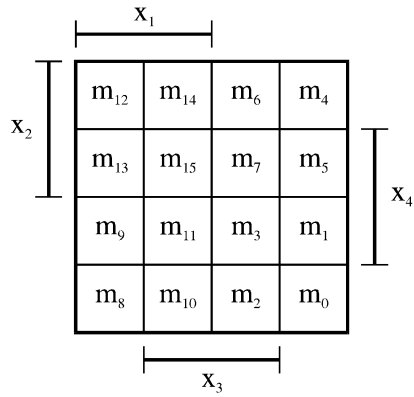
Za **grafički prikaz** koristimo Vennove dijagrame, slika 3, kod kojih unutar univerzalnog skupa definiramo područja nad kojima se pojedina varijabla definira kao logička jedinica.



Slika 3. - Grafički prikaz Vennovim dijagramima

Presjecišta pojedinih područja su mjesta nad kojima je eventualno više varijabli definirano kao logička jedinica. Svako presjecište definirano je, dakle, jednom kombinacijom vrijednosti varijabli koja ima značenje kompleksije varijabli, pa nad njim možemo definirati vrijednost funkcije, odnosno preslikavanje u skup S=<0,1>.

Kod problema s većim brojem varijabli nije prikladan geometrijski oblik kruga pa primjenjujemo **Veitchev dijagram** gdje su područja formirana u obliku kvadrata, slika 4. U polja koja odgovaraju mintermima m_0 - m_{15} upisujemo vrijednost funkcije (0,1).



Slika 4. - Veitchev dijagram za n=4

Elementarne funkcije algebre logike su sve funkcije sa jednom ili dvije ulazne varijable. Funkciji od n varijabli pripada tablica istine od $N=2^n$ redaka. Desnu stranu tablice možemo popuniti na $K=2^N$ načina, pa različitih funkcija od n ima:

$$K = 2^N = 2^{2^n}$$

Za dvije varijable x_1 i x_2 ima $2^{2^2} = 16$ različitih funkcija koje su prikazane tablicom:

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_A	f_B	f_C	f_D	f_E	f_F
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Da bismo izrazili proizvoljnu Booleovu funkciju, potreban nam je takav skup elementarnih funkcija pomoću kojih možemo izraziti bilo koju funkciju. Takav skup zovemo **potpunim skupom funkcija** algebre logike. Nastojimo da on sadrži što manji broj operatora, kako bi kod realizacije trebali što manji broj različitih vrsta sklopova.

Kako je algebra logike definirana nad operacijama &, $\vee$ i -, a to su elementarne funkcije disjunkcije, konjunkcije i negacije, odnosno f_8, f_E, i f_3, to je skup funkcija <&, $\vee$ i -> po definiciji potpuni skup. To je potvrđeno primjenom potpunih normalnih oblika. Dakle, potpuni skup elementarnih funkcija je onaj pomoću kojega možemo izraziti funkcije &, $\vee$ i -, jer to znači da tada možemo izraziti sve funkcije.

U praksi su nam interesantni **Pierce** (**NILI**, negacija disjunkcije, engl. **NOR)** i **Shaeffer** (**NI**, negacija konjunkcije, engl. **NAND)** operatori (f_1 i f_7), koji su svaki za sebe potpuni skup funkcija, pa pomoću njih sve logičke funkcije možemo izraziti jednom vrstom logičkih elemenata, odnosno logičkih sklopova.

Pierceov operator (NILI) Shaefferov operator (NI)

$$x_1 \downarrow x_2 = \left(\overline{x_1 \vee x_2}\right) \qquad x_1 | x_2 = \left(\overline{x_1 \& x_2}\right)$$

KOMBINACIONE LOGIČKE STRUKTURE

Zamislimo proizvoljni digitalni sklop. Prema ranijem dogovoru o korištenju binarnog brojevnog sustava, signali koji ulaze i izlaze iz sklopa mogu imati samo vrijednost 0 ili 1, dakle odgovaraju Booeleovim varijablama. Ako naš digitalni sklop generira izlaze na osnovu trenutne vrijednosti ulaznih signala, možemo ga opisati Booleovom funkcijom. Odatle postupak sinteze kod kojeg najprije zadajemo Booleovu funkciju, minimiziramo je, te crtamo shemu digitalnog sklopa.

Preslikavanje koje obavlja Booleova funkcija je trenutno, dok stvarni digitalni sklop unosi neko kašnjenje. To kašnjenje često možemo zanemariti. Za digitalni sklop koji opisujemo Booleovom funkcijom i koji izlaz generira samo na osnovu trenutne vrijednosti ulaznih signala kažemo da nema memorije, i nazivamo ga **kombinacionom logičkom strukturom** (skraćeno KLS).

SEKVENCIJALNI DIGITALNI SKLOPOVI

Za razliku od kombinacionih, **sekvencijalni digitalni sklopovi** izlaz generiraju na osnovu trenutnih i prošlih vrijednosti ulaza. Takvi sklopovi moraju raspolagati s memorijom u kojoj pamte podatke o proteklim događajima, kako bi u sadašnjosti mogli donijeti odluku o izlaznim vrijednostima.

Sekvencijalni sklopovi mogu raditi **asinkrono**, tako da reagiraju na promjene ulaza u trenutku njihovog nastanka. Asinkroni sklopovi su osjetljivi na kašnjenja koja nastaju u samim sklopovima, pa nisu pogodni za masovnu proizvodnju. Stoga se prvenstveno koriste **sinkroni** sklopovi, čiji je rad istovremen (sinkron) s posebnim taktnim signalom. Sinkroni sklopovi su mnogo pouzdaniji. Oni rade u diskretnom vremenu, određenom taktnim signalom.

Najjednostavniji sekvencijalni sklopovi pamte samo jedan bit informacije, 0 ili 1, pa ih nazivamo **bistabilima**. To su elementarni sekvencijalni sklopovi. Složeni sekvencijalni sklopovi raspolažu memorijom sastavljenom od više bistabila. Njih nazivamo **automatima**.

LABORATORIJSKI MODEL DELAB1

Laboratorijske vježbe odvijaju se na modelu DELAB1, koji je ostvaren sa sljedećim svojstvima:

- podržava sve predviđene vježbe
- raspolaže vlastitim izvorima napajanja i signala, te indikatorima
- omogućava neposredan rad s digitalnim integriranim krugovima
- sadrži sve najvažnije integrirane krugove malog i srednjeg stupnja integracije
- omogućava rad s programabilnim strukturama srednjeg i visokog stupnja integracije.

Izgled modela prikazan je na slici 5. Funkcionalni blokovi su:

izvori	generator
indikator	tehnologija
ni (2 kom)	nili
suma po modulu	sumator/komparator
bistabil	LSI
multiplekser	demultiplekser

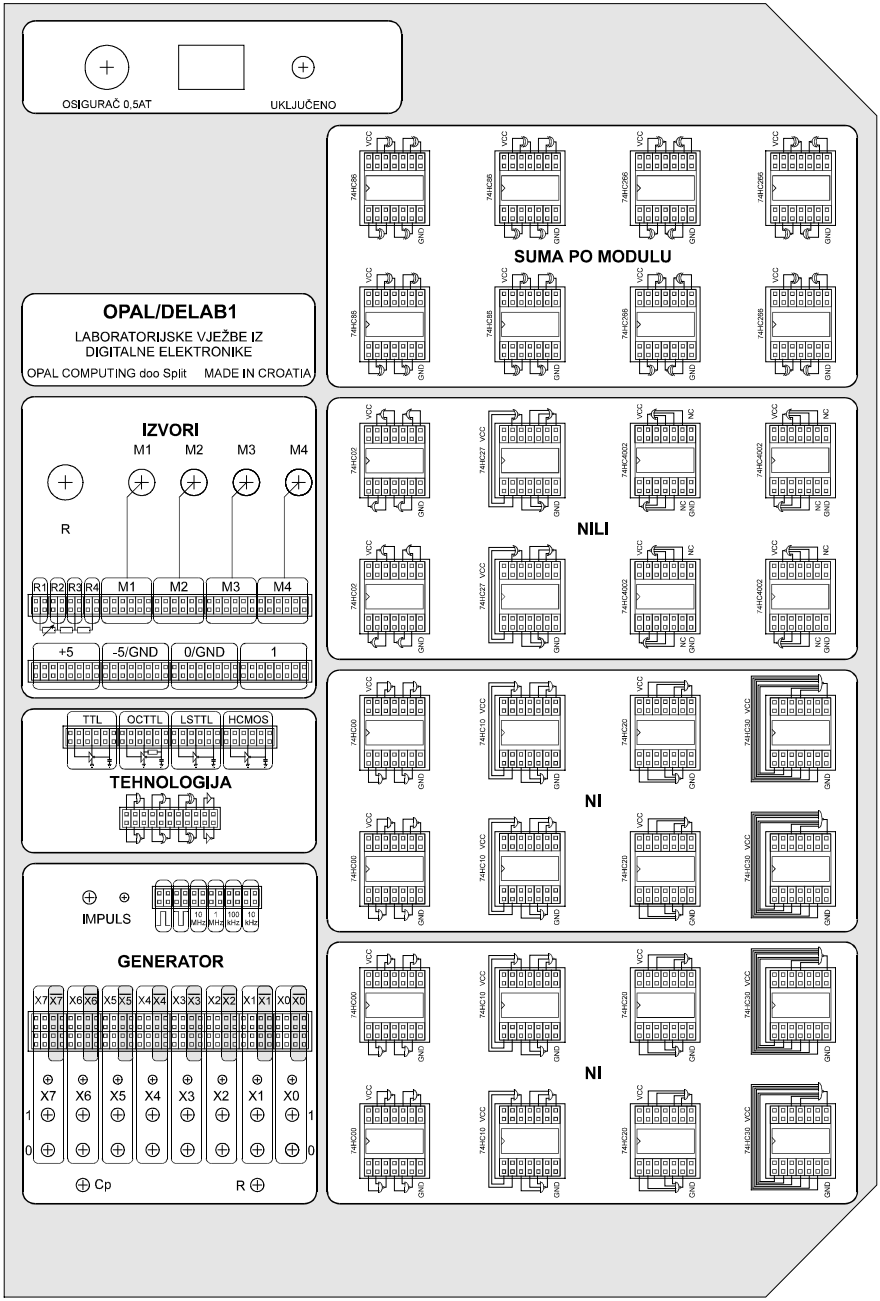
Blok **IZVORI** sadrži priključke izvora napajanja 5V=, priključke izvora 0 (masa) i 1 (+5V preko otpornika 1kohm), priključke za mjerne instrumente M1 - M4, te promjenljivi otpornik R za opterećivanje izlaza logičkih vrata (Vježba 1.), slika 6.

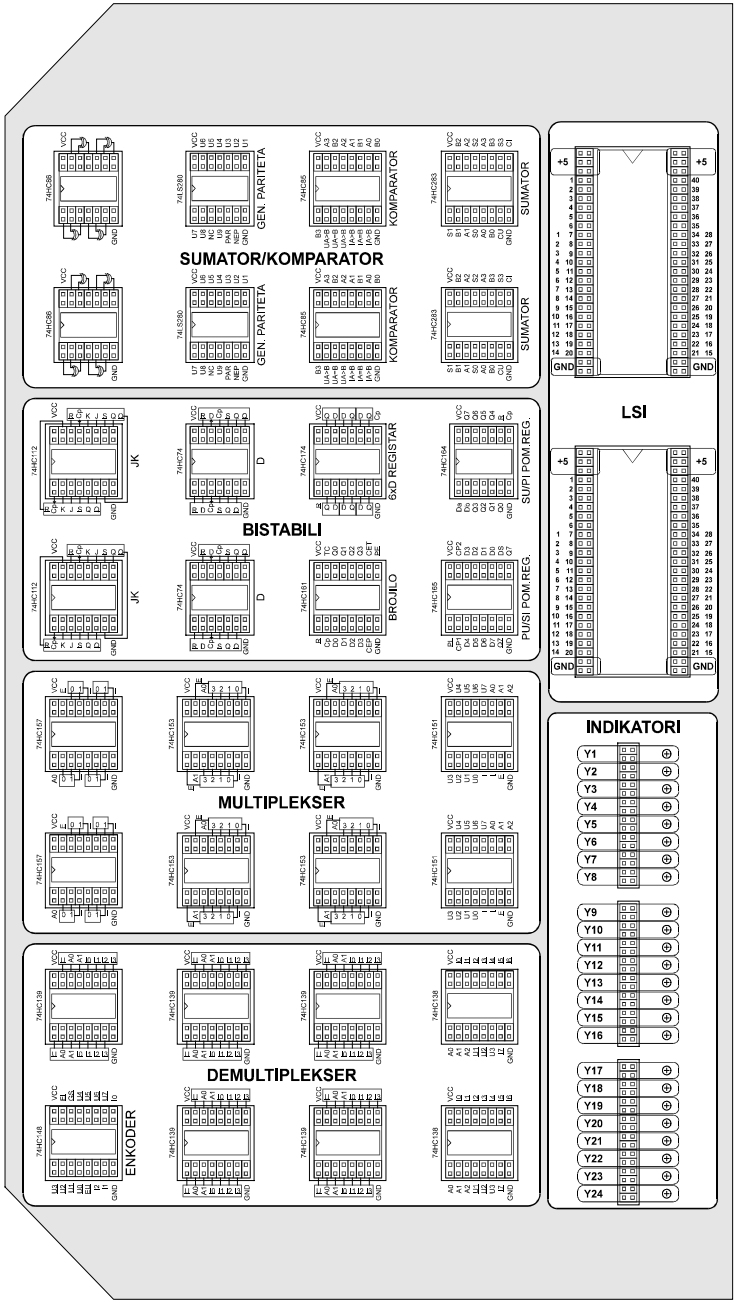
Blok **GENERATOR** sadrži generator kodnih riječi (x_7-x_0) koji omogućava postavljanje proizvoljne kodne riječi i sekvencijalno generiranje prirodnog binarnog niza, generator pojedinačnog impulsa, te generator digitalnog signala (četvrtki) frekvencije 10 MHz, 1 MHz, 100 kHz i 10 kHz, slika 7.

Blok **INDIKATOR** sastoji se od 24 svjetlosna (LED) indikatora digitalnog signala.

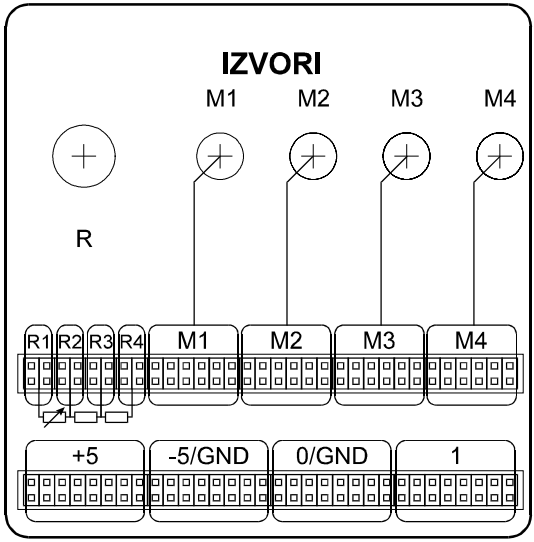
Blok **TEHNOLOGIJA** sadrži komponente potrebne za mjerenje tehnoloških i funkcionalnih karakteristika logičkih vrata, slika 8.

Blok **LSI** sadrži 2 40-iglična podnožja za umetanje LSI komponenti (EPROM, GAL, mikroprocesori itd.).

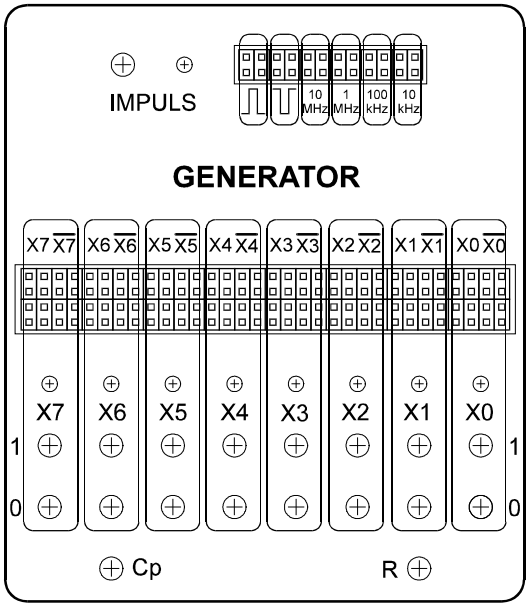




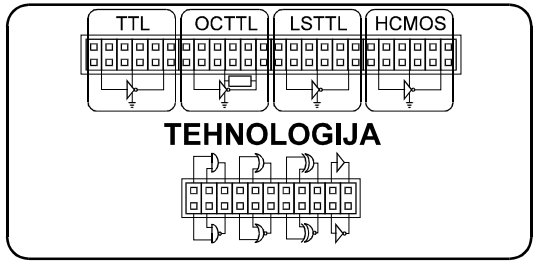
Slika 5



Slika 6. - Blok IZVORI

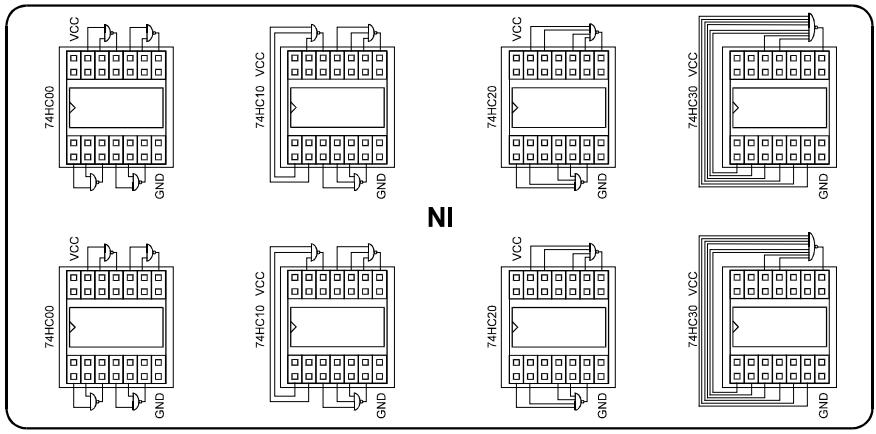


Slika 7. - Blok GENERATOR



Slika 8. - Blok TEHNOLOGIJA

Blok **NI** (dva bloka) sadrži integrirane krugove s NI (NAND) logičkim vratima, slika 9.



Slika 9. - Blok NI

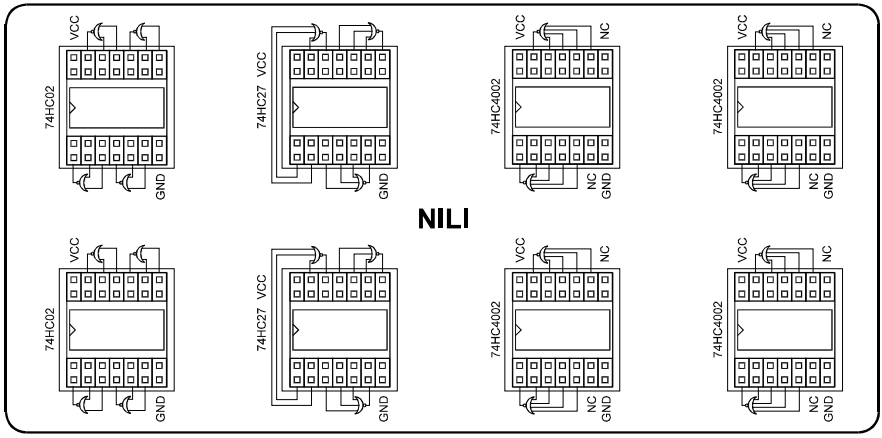
Korišteni su integrirani krugovi:

74xx00	četiri dvoulazni NI vrata s bipolarnim (TP) izlazom
74xx10	tri troulazna NI vrata s bipolarnim (TP) izlazom
74xx20	dva četveroulazna NI vrata s bipolarnim (TP) izlazom
74xx30	jedna osmeroulazna NI vrata s bipolarnim (TP) izlazom.

Blok **NILI** sadrži integrirane krugove s NILI (NOR) logičkim vratima, slika 10.

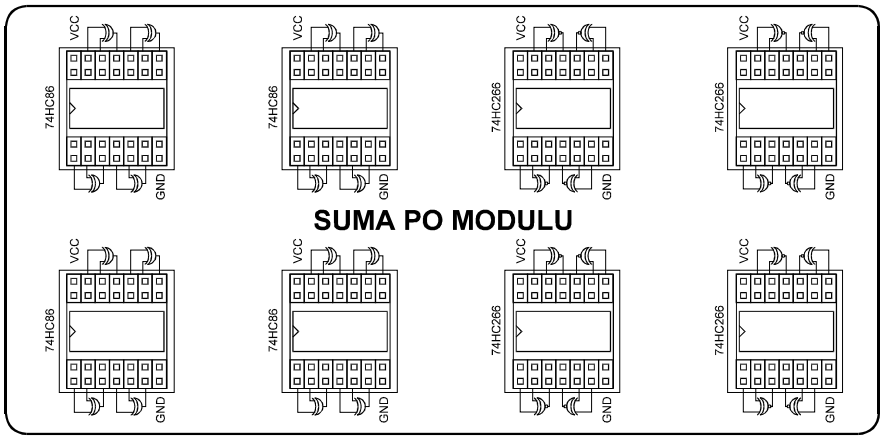
Korišteni su integrirani krugovi:

74xx02	četiri dvoulazni NILI vrata s bipolarnim (TP) izlazom
74xx27	tri troulazna NILI vrata s bipolarnim (TP) izlazom
74xx4002	dva četveroulazna NILI vrata s bipolarnim (TP) izlazom, kompatibilni po izvodima s CMOS krugom 4002.



Slika 10. - Blok NILI

Blok **SUMA PO MODULU** sadrži integrirane krugove s EX-ILI (EX-OR) i EX-NILI (EX-NOR) logičkim vratima, slika 11.



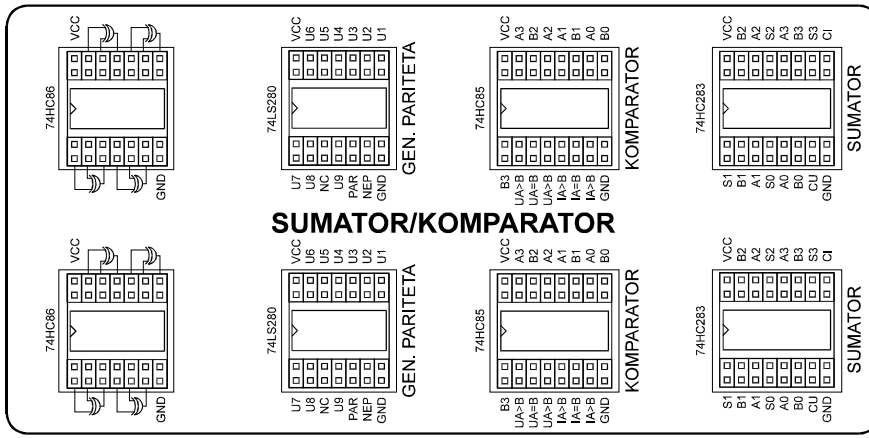
Slika 11. - Blok SUMA PO MODULU

Korišteni su integrirani krugovi:

74xx86	četiri dvoulazni EX-ILI vrata s bipolarnim (TP) izlazom
74xx266	četiri dvoulazni EX-NILI vrata s otvorenim kolektorom.

EX-ILI vrata koriste se za izgradnju kodera ili dekodera, a EX-NILI vrata za izgradnju generatora pogrješke i korektora Hammingovog koda.

Blok **SUMATOR/KOMPARATOR** sadrži integrirane krugove s EX-ILI (EX-OR), generator pariteta, sumator i komparator, slika 12.



Slika 12. - Blok SUMATOR/KOMPARATOR

Korišteni su integrirani krugovi:

74xx86 četiri dvoulazni NILI vrata s bipolarnim (TP) izlazom (dodatak bloku SUMA PO MODULU)

74xx280 generator pariteta, izračunava parni i neparni paritet za 9 ulaza, koristi se za izgradnju kodera ili dekodera Hammingovog koda

74xx85 4-bitni komparator, uspoređuje dva četverobitna pozitivna broja i pokazuje da li je A veći, jednak ili manji od B

74xx283 4-bitni sumator, zbraja dva četverobitna broja, preuzima pretek sa manje značajnog i generira pretek ka značajnijem bloku znamenki.

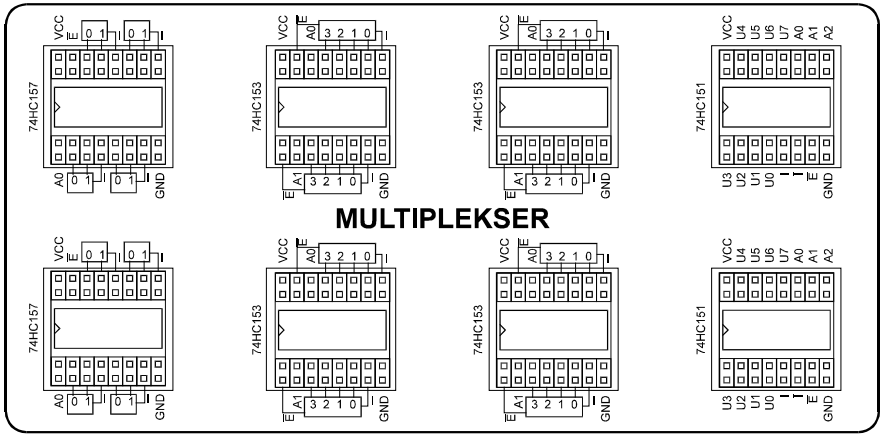
Blok **MULTIPLEKSER** sadrži integrirane krugove s multiplekserima različite veličine, slika 13.

Korišteni su integrirani krugovi:

74xx157 četiri multipleksera m=1 sa zajedničkim adresnim (A0) i kontrolnim (E) ulazima

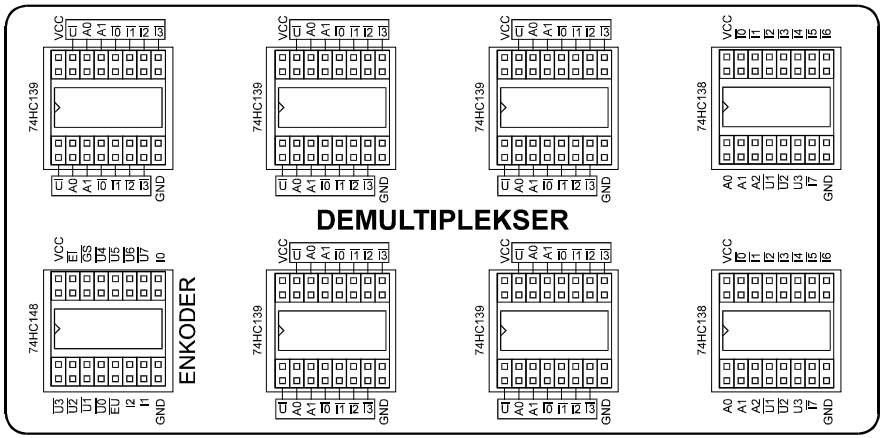
74xx153 dva multipleksera m=2 sa zajedničkim adresnim (A0, A1) i odvojenim kontrolnim (E) ulazima.

74xx151 jedan multiplekser m=3, raspolaže komplementarnim izlazom i jednim kontrolnim ulazom.



Slika 13. - Blok MULTIPLEKSER

Blok **DEMULTIPLEKSER** sadrži integrirane krugove s demultiplekserima različite veličine, slika 14.



Slika 14. - Blok DEMULTIPLEKSER

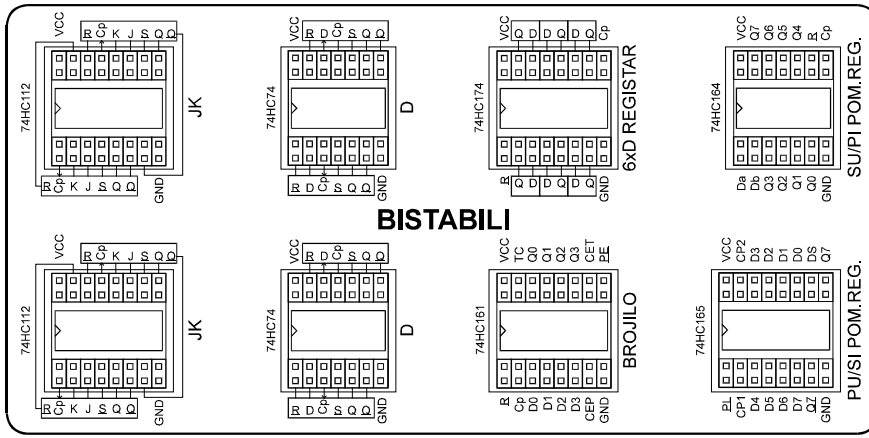
Korišteni su integrirani krugovi:

74xx139 dva demultipleksera m=2 sa odvojenim adresnim (A0, A1) i kontrolnim (E) ulazima

74xx138 jedan demultiplekser m=3, s invertiranim izlazima, raspolaže s tri kontrolna ulaza (U1, U2 i U3)

74xx148 jedan enkoder prioriteta m=3, sklop koji generira u prirodnom binarnom obliku redni broj aktivnog ulaza. Raspolaže s ulazom i izlazom za povezivanje u seriju s drugim enkoderima kada treba više od 8 ulaza.

Blok **BISTABILI** sadrži integrirane krugove s bistabilima, registrima, brojilima i pomačnim registrima različite vrste, slika 15.



Slika 15. - Blok BISTABILI

Korišteni su integrirani krugovi:

74xx112 dva JK bistabila s asinkronim RS ulazima, s odvojenim taktnim ulazima osjetljivim na silazni brid taktnog signala

74xx74 dva D bistabila s asinkronim RS ulazima, s odvojenim taktnim ulazima osjetljivim na uzlazni brid taktnog signala

74xx174 6-bitni D registar s asinkronim R (reset) ulazom, služi za pamćenje 6-bitne kodne riječi

74xx161 sinkrono 4-bitno binarno brojilo s paralelnim ulazima, asinkronim R ulazom, te signalima za povezivanje u niz

74xx164 8-bitni pomačni registar sa serijskim ulazom i paralelnim izlazima, te asinkronim R ulazom

74xx166 8-bitni pomačni registar sa paralelnim ulazima i serijskim izlazom, te asinkronim R ulazom

Model DELAB1 sadrži vlastito napajanje iz mreže 220V/50Hz.

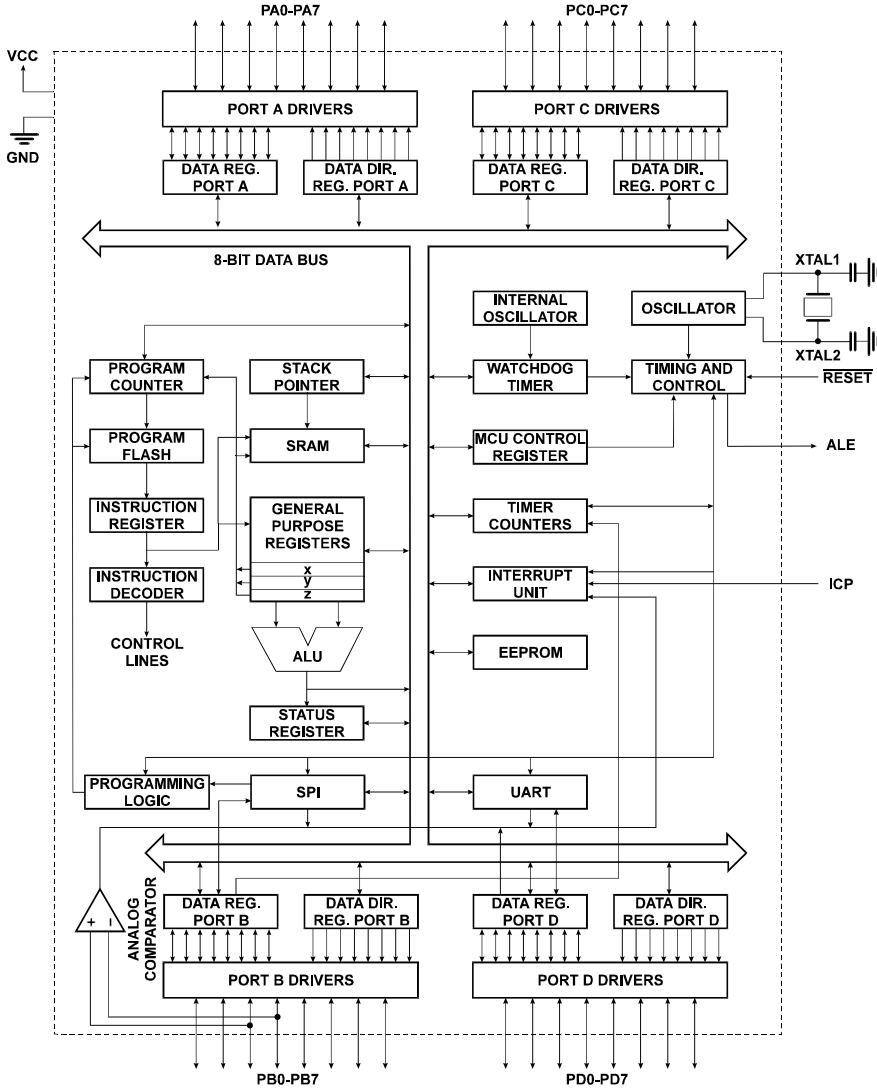
OPIS MIKROKONTROLERA AT90S8515

U laboratorijskim vježbama iz područja arhitekture mikroračunala koristit ćemo mikrokontroler AT90S8515 firme Atmel, USA. Njegove mogućnosti i svojstva su:

- AVR RISC (Reduced Instruction Set Computer) arhitektura
  - 118 naredbi, većina se izvrši u jednom taktnom periodu
  - 32 osam-bitna radna registra opće namjene
  - do 8 MIPs (Mega Instruction per Second) na 8 MHz
- Memorija
  - 8 KB flash EPROM memorije za program, ISP (In System Programming) mogućnost programiranja u sustavu
  - 512 bajta SRAM-a
  - 512 bajta EEPROM-a, ISP
- U/I sklopovi
  - jedan 8-bitni vremenski sklop
  - jedan 16-bitni vremenski sklop Načini rada: Usporedi, Zapamti stanje, dvostruki 8-, 9- ili 10-bitni PWM
  - analogni komparator
  - programabilni Watch-Dog sa svojim oscilatorom
  - programabilni asinkroni serijski vezni sklop (UART)
  - programabilni sinkroni serijski vezni sklop (SPI)
- Posebne mogućnosti mikrokontrolera
  - načini rada s uštedom energije
  - vanjski i unutrašnji izvori prekida
  - CMOS tehnologija visoke brzina, male potrošnje energije
  - potpuno statički rad
- Ulazi / Izlazi, kućište
  - 32 programabilna U/I priključka
  - 40-pin PDIP, 44-pin PLCC i TQFP
- Radni napon i brzina rada
  - 2.7 – 6.0 V, 0 – 4 MHz (AT90S8515-4)
  - 4.0 – 6.0 V, 0 – 8 MHz (AT90S8515-8)

AVR procesorska jezgra kombinira bogati skup naredbi s 32 radna registra opće namjene. Zadnjih šest od 32 registra mogu se koristiti kao tri 16-bitna indeksna registra. Sva 32 registra su izravno spojena na aritmetičko-logičku jedinicu ALU (Arithmetic Logic Unit), omogućavajući da se dvama neovisnim registrima može pristupiti u jednom ciklusu sabirnice, koji je ujedno i taktni ciklus. Ostvarena arhitektura raspolaže s posebnom sabirnicom za programsku, a s posebnom za

podatkovnu memoriju pa je vrlo efikasna kod izvršenja programa. Postiže se do 10 puta veća brzina rada za istu frekvenciju takta od uobičajenih CISC (Complete Instruction Set Computer) mikrokontrolera. Blok shema mikrokontrolera prikazana je na slici 16.



Slika 16. - Blok shema mikrokontrolera AT90S8515

AT90S8515 ima **tri memorijska adresna** područja:

1. ISP Flash EPROM programska memorija – u ovu memoriju se upisuje program koji mikrokontroler izvršava. AT90S8515 ima 8 KB memorije, a kako su sve naredbe 32 ili 64 bitne, ova memorija je organizirana kao 4 K × 16 bita. Izdržljivost ove memorije je najmanje 1000 ciklusa pisanja i brisanja.
2. SRAM podatkovna memorija – unutarnji RAM ima 512 bajta, a izvana možemo spojiti ukupno 64 Kb RAM-a. Ova memorija gubi sadržaj pri nestanku napona napajanja, osim ako se podrži vanjskom baterijom.
3. EEPROM podatkovna memorija – 512 bajta. Ova memorija zadržava sadržaj pri nestanku napona napajanja. Služi za zapisivanje parametara koje povremeno mijenjamo, a moramo zapamtiti i kad nema napona napajanja.

AT90S8515 raspolaže s 13 **prekida**, od kojih su za vježbe značajna 2:

Adresa	Ime	Funkcija
`$000`	`RESET`	Nakon *reseta* mikrokontrolera program se izvodi od adrese nula.
`$007`	`TIM0_OVF`	Vremenski sklop 0 prekid za pretek brojača.

AT90S8515 raspolaže s više **ulazno/izlaznih registara**, od kojih su za vježbe značajni slijedeći:

U/I Adresa	Ime	Funkcija	U/I Adresa	Ime	Funkcija
`$3F`	`SREG`	Statusni Registar	`$18`	`PORTB`	Izlaz porta B
`$3E`	`SPH`	SP viši bajt	`$17`	`DDRB`	Smjer porta B
`$3D`	`SPL`	SP niži bajt	`$16`	`PINB`	Ulaz porta B
`$39`	`TIMSK`	VS 0 maske prekida	`$15`	`PORTC`	Izlaz porta C
`$33`	`TCCR0`	VS 0 upravljački r.	`$14`	`DDRC`	Smjer porta C
`$32`	`TCNT0`	VS 0 vrijednost	`$13`	`PINC`	Ulaz porta C
`$1B`	`PORTA`	Izlaz porta A	`$12`	`PORTD`	Izlaz porta D
`$1A`	`DDRA`	Smjer porta A	`$11`	`DDRD`	Smjer porta D
`$19`	`PINA`	Ulaz porta A	`$10`	`PIND`	Ulaz porta D

Statusni registar – SREG Sadrži globalni bit za dozvolu prekida (bit **I**) koji se postavlja naredbom “sei”, a briše naredbom “cli”. Tu su i bitovi **Z** (Zero – označava da je rezultat 0), **C** (Carry – označava da je nastao pretek).

Stack pointer – SP 16-bitni indikator sloga.

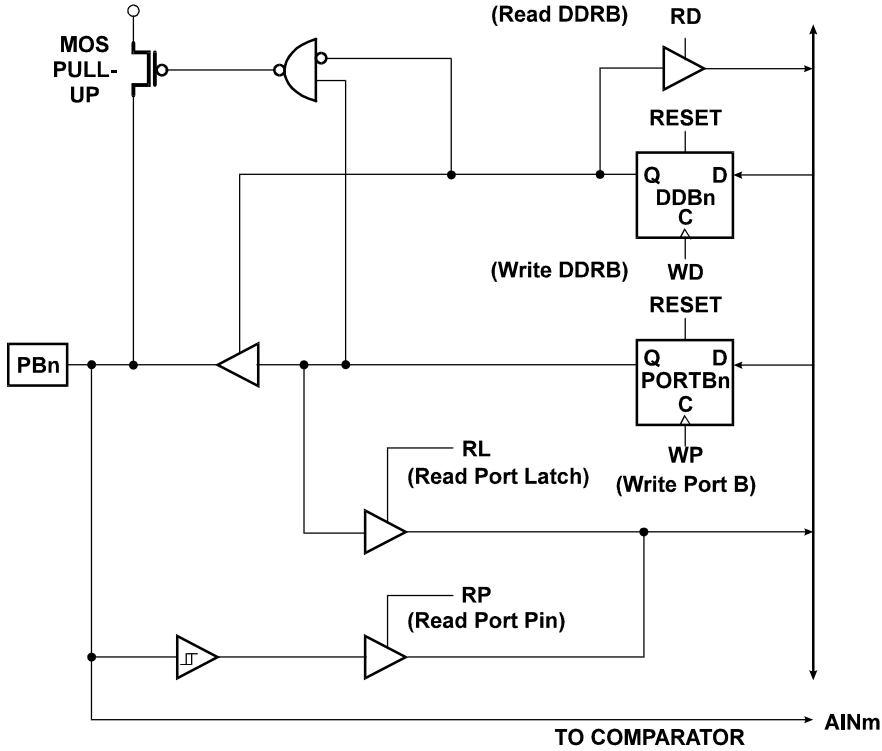
PORTx Izlazni registri U/I sklopova. Ako je priključak definiran kao izlaz, ovdje upisujemo vrijednost koja će se pojaviti na izlazima mikrokontrolera.

DDRx Registri smjera U/I sklopova. Logičko '1' za pojedini priključak označava izlaz, a logičko '0' označava ulaz.

PINx Služi za čitanje stanja vanjskih nožica mikrokontrolera, bilo da je pin definiran kao ulaz ili izlaz.

TIMSK, TCCR0, TCCR0 Registri za upravljanje vremenskim sklopom 0. O ovim registrima više u opisu vremenskog sklopa 0.

AT90S8515 ima **32 U/I priključka** organizirana u 4 U/I sklopa. Svaki od 32 U/I priključka možemo definirati kao ulaz i/ili izlaz, upisom u registre za smjer DDRx. Na slici 17. prikazana je izvedba jednog U/I priključka.



Slika 17. - Blok shema U/I priključka

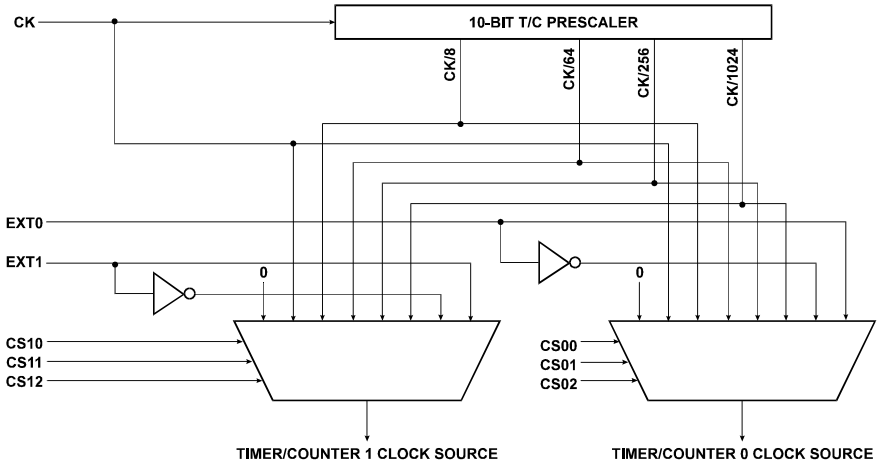
U/I sklop A (PA7..PA0) U/I sklop A je 8-bitni dvosmjerni paralelni vezni sklop. Za svaki priključak sklopa može se aktivirati unutarnji otpornik za pritezanje na napon napajanja. Izlazna pojačala sklopa A mogu primiti (*sink*) 20 mA i direktno pogoniti LED indikatore. Posebna funkcija: U/I sklop A se koristi za pristup vanjskom RAM-u ako smo ga spojili.

U/I sklop B (PB7..PB0) Vrijedi isto što i za U/I sklop A. Posebna funkcija: neki priključci sklopa B se koriste za programiranje flash i EEPROM memorije, te za komunikaciju preko SPI sklopa.

U/I sklop C (PC7..PC0) Vrijedi isto što i za U/I sklop A. Posebna funkcija: Sklop C se koristi za pristup vanjskom RAM-u ako smo ga spojili.

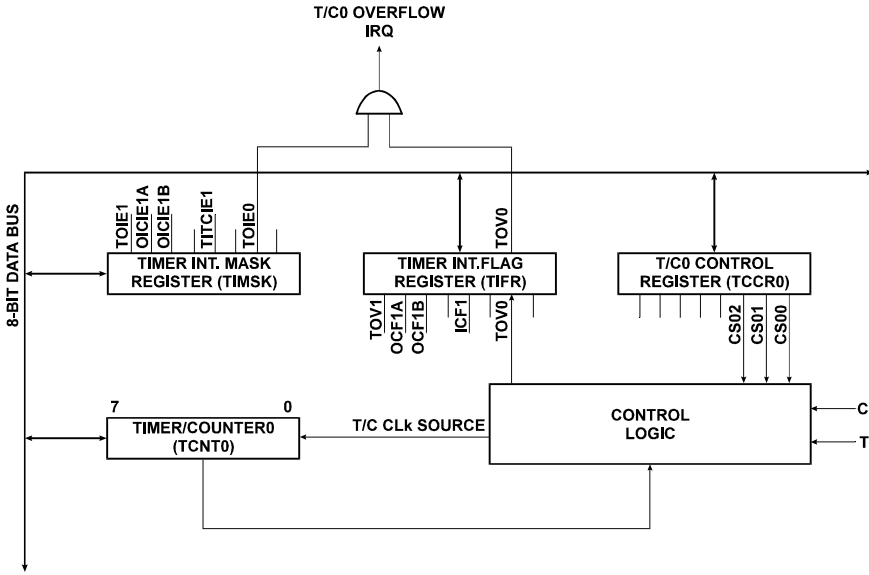
U/I sklop D (PD7..PD0) Vrijedi isto što i za U/I sklop A. Posebna funkcija: neke priključke U/I sklopa D koristimo i za UART, tu su i upravljački signali za čitanje i pisanje vanjskog RAM-a, te ulazi za vanjske prekide i brojila vremenskog sklopa.

AT90S8515 raspolaže s dva **vremenska sklopa (Timer/Counter, TC)**, od kojih ovdje opisujemo T/C0. Na slici 18. prikazano je generiranje takta za vremenski sklop 0. Ugrađeno predbrojilo služi za dijeljenje frekvencije taktnog signala. Multiplekserima se bira željeni unutrašnji ili vanjski taktni signal.



Slika 18. - Generator takta za vremenske sklopove

Na slici 19. prikazana je blok shema vremenskog sklopa 0 za AVR mikrokontrolere.



Slika 19. - Vremenski sklop 0 (8-bitni)

Raspored priključaka mikrokontrolera AT90S8515 prikazan je na slici 20. Osim ranije opisanih, interesantni su signali:

Vcc Napon napajanja.

GND Masa.

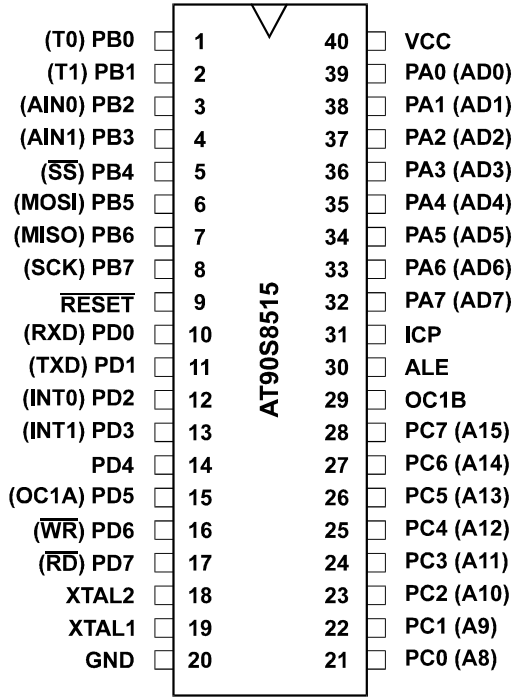
RESET ulaz. Niski nivo na ovom priključku duži od 50 ns generira reset (iniciranje mikrokontrolera, kao kad ga tek uključimo).

XTAL1 i XTAL2 Na ove priključke spajamo kvarcni kristal kojim kontroliramo frekvenciju taktnog signala mikrokontrolera. Ako koristimo vanjski izvor takta, spajamo ga na XTAL1 (ulaz), a frekvenciju možemo provjeriti na XTAL2 (izlaz).

ICP Ulaz za vremenski sklop 1 za funkciju ulaznog prihvata (Input Capture).

OC1B Izlaz za vremenski sklop 1 za funkciju izlaza komparatora (Output Compare).

ALE koristi se pri upotrebi vanjske RAM memorije (Address Latch Enable).



Slika 20. - Raspored priključaka za AT90S8515

OPIS RAZVOJNOG SUSTAVA STK200

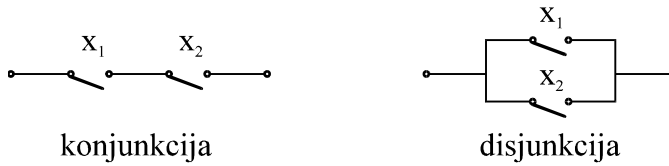
Razvojni sustav STK200 sastoji se od pločice s mikrokontrolerom, asemblera, simulatora i ISP programa. Pločica STK200 se priključi posebnim kabelom na paralelni vezni sklop računala (umjesto pisača), te se ISP programom može čitati ili programirati sadržaj memorije mikrokontrolera.

Na samoj pločici ugrađena su tipkala priključena na U/I sklop D, a LED diode na U/I sklop B. Pločica se napaja s vanjskog izvora napona do 12V.

Razvojni ciklus korisničkog programa provodi se u koracima unosa teksta izvornog programa, prevođenja (asembliranja), provjere na simulatoru i provjere na stvarnom mikrokontroleru (STK200). Pogrješke se ispravljaju istim postupkom.

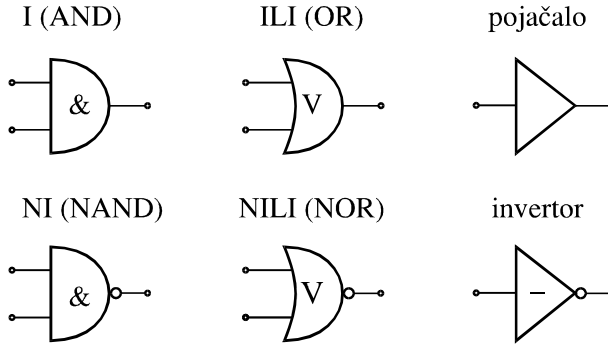
VJEŽBA 1: TEHNOLOŠKA REALIZACIJA LOGIČKIH VRATA

Primjena Booleovih funkcija za opisivanje rada digitalnih sklopova interesantna je zbog mogućnosti direktnog crtanja sheme. Svaki od operatora zamjenjujemo kod realizacije odgovarajućim sklopom. Npr. u relejnoj tehnici konjunkciju realiziramo serijskom, a disjunkciju paralelnom vezom kontakata, slika.1.1.



Slika 1.1. - Relejna realizacija konjunkcije i disjunkcije

Kod elektronički realiziranih funkcija, svaka elementarna operacija zahtijeva primjenu odgovarajućeg sklopa. Te sklopove nazivamo logičkim sklopovima ili **logičkim vratima**. Oni se danas isključivo proizvode i koriste kao integrirani krugovi. Simboli sklopova koji realiziraju najvažnije elementarne funkcije dani su na slici 1.2. Dijelimo ih po tehnologiji izrade, gustoći (nivou) integracije i karakteristikama ulaza i izlaza.

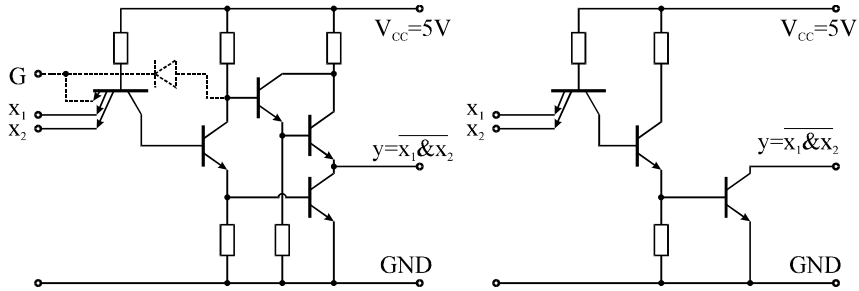


Slika 1.2. - Simboli elementarnih logičkih sklopova

Osnovna podjela vrši se s obzirom na tehnologiju izrade. Najčešće se koriste TTL, NMOS i CMOS tehnologije.

TTL tehnologija pogodna je za SSI i MSI integrirane krugove. Karakterizira je upotreba bipolarnih tranzistora, te višeemiterskih tranzistora na ulazu. Kašnjenje signala na jednim vratima je reda veličine 4-10 ns. Brzinu rada ograničavaju RC konstante prisutne zbog parazitnih kapaciteta PN spoja i otpornika za polarizaciju tranzistora. Smanjenje otpora otpornika je moguće,

ali uz povećanje potrošnje, pa se proizvodi nekoliko familija TTL integriranih krugova (74xx = normalni, 74Lxx = niska potrošnja i brzina, 74Hxx = velika brzina i potrošnja, 74Sxx = normalni shottky, 74LSxx = shottky sa malom potrošnjom, 74ALSxx novija familija sa manjom dimenzijom tranzistora). Sheme TTL logičkih sklopova prikazane su na slici 1.4.

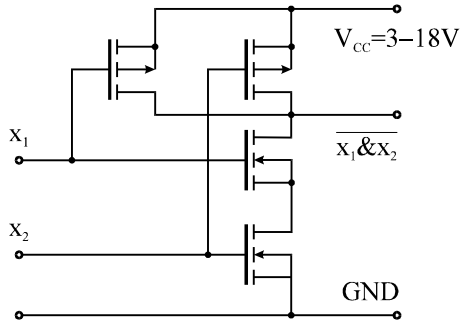


Slika 1.4. - TTL NI logički sklop s bipolarnim (crtkano TS) i OC izlazom

S i LS familije koriste shottkyjev efekt na spoju metal - poluvodič, kojim se smanjuje napon zasićenja PN spoja. Time se efektivno smanjuju naboji na parazitnim kapacitetima, pa se uz istu potrošnju povećava brzina rada. Po brzini su 74xx i 74LSxx ekvivalentne, a ova posljednja troši oko četiri puta manje energije. Standardni napon napajanja TTL integriranih krugova je 5V i najčešće se koristi i kod drugih tehnologija. Primjeri TTL integriranih krugova su 7400=4x2NI, 7402=4x2NILI itd.

NMOS tehnologija koristi se za proizvodnju LSI integriranih krugova. Sklopovi se sastoje od NMOS FET-ova. Zbog malih dimenzija moguća je velika gustoća pakiranja. Kašnjenje signala na jednim vratima je reda veličine 10-100ns. Potrošnja je nešto manja od TTL tehnologije, ali ipak značajna zbog velikog broja elemenata na jednoj pločici silicija. Napon napajanja je 5V, a kod nekih starijih krugova još i +12V i -5V. Primjer NMOS integriranog kruga je mikroprocesor Z80.

CMOS tehnologija koristi se za proizvodnju SSI i MSI, te od nedavno i LSI krugova. Karakterizira je upotreba komplementarnih (P i N kanalnih) MOS tranzistora koji rade u protu fazi. Time se potrošnja svodi na struje punjenja i pražnjenja parazitnih kapaciteta, te je kod malih brzina rada izrazito niska. Povećanjem brzine rada, potrošnja dostiže onu kod NMOS tehnologije. Kašnjenje signala na vratima je reda veličine 10-100ns. Napon napajanja je 5V (3V do 18V za SSI serije 4000). Primjer CMOS integriranog kruga je 6116 2Kx8 statički RAM. Shema CMOS logičkog sklopa prikazana je na slici 1.5.



Slika 1.5. - CMOS NI logički sklop s bipolarnim izlazom

Po gustoći integracije vršimo podjelu na SSI, MSI, LSI i VLSI integrirane krugove. **SSI** (niski stupanj integracije) realizira se integracijom do 12 ekvivalentnih logičkih vrata (oko 100 elemenata) po pločici silicija, **MSI** (srednji stupanj integracije) podrazumijeva 100 vrata (1000 elemenata), **LSI** (visoki stupanj integracije) podrazumijeva 1000 vrata (10000 elemenata), a **VLSI** (vrlo visoki stupanj integracije) više od 10000 vrata po pločici (100000 elemenata). Fizička veličina pločice silicija ograničena je površinskom gustoćom pogrješaka u proizvodnji. Stoga se teži smanjenju dimenzija tranzistora, čime se postižu manji parazitni kapaciteti i veća brzina rada.

Izlazi integriranih krugova mogu biti realizirani kao bipolarni, unipolarni i TRI-STATE.

Bipolarni izlazi (TP = totem-pole) generiraju vrijednosti 0 i 1. U logičkoj 1 služe kao izvorišta, a u logičkoj 0 kao potrošači struje.

Unipolarni izlazi (OC = open-collector) generiraju samo logičku 0, i mogu biti samo potrošači struje. Kod logičke 1 tranzistor se isključuje, a potrebni naponski nivo postižemo spajanjem otpornika prema naponu napajanja. Unipolarni izlazi se koriste kada je potrebno spojiti više izlaza na zajedničku točku (sabirnicu). Mana im je produženo vrijeme porasta signala iz 0 u 1 u odnosu na bipolarne izlaze.

Tri-state izlazi (TS, bipolarni sa trećim stanjem visoke impedancije) imaju karakteristike bipolarnih, a sposobnost isključenja omogućava spajanje na sabirnicu.

Važna karakteristika izlaza je **faktor izlaznog grananja** (FAN-OUT), koji nam kaže koliko standardnih ulaza možemo spojiti na jedan izlaz, a da pri tom naponi signala ostanu u propisanim granicama. Za TTL i NMOS tehnologiju te su granice:

0,8V max za logičku 0 (V_{ol}) 2,4V min za logičku 1 (V_{oh})

Pri tome izlaz u logičkoj 1 može dati maksimalnu struju I_{ohM}, a u logičkoj 0 uzeti maksimalnu struju I_{olM}.

Ulazi imaju karakteristike ulaznih elektroda tranzistora. Kod CMOS i NMOS tehnologije to su vrata MOS tranzistora, pa je ulazni otpor vrlo velik. Ulazni kapacitet je reda veličine 10pF. Kod TTL tehnologije to su emiteri višeemiterskog tranzistora. U logičkoj 1 teče vrlo mala inverzna struja PN spoja ka emiteru (red veličine 40mA). U logičkoj 0 tranzistor vodi i tada teče struja emitera (red veličine 1mA) u smjeru iz emitera.

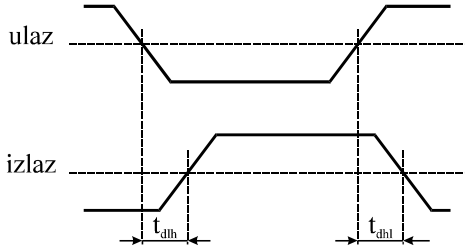
Faktor ulaznog grananja kaže nam koliko stvarni ulaz opterećuje izlaz na koji je spojen u odnosu na standardni ulaz za promatranu tehnologiju i varijantu izrade. Ovaj faktor je najčešće 1. Ulaz u logičkoj 1 troši struju I_{ih}, a u logičkoj 0 daje struju I_{il}. Stoga je faktor grananja za određeni integrirani krug dat funkcijom minimuma:

$$FG = \min (I_{ohM} / I_{ih} ; I_{olM} / I_{il})$$

Mjera kvalitete pojedine tehnologije je produkt potrošnje i vremena kašnjenja koji ima dimenziju energije (J). **Vrijeme kašnjenja** definirano je za prijelaze na izlazu:

t_{dhl} = prijelaz iz 1 u 0 t_{dlh} = prijelaz iz 0 u 1

Oba vremena mjerimo kao kašnjenje između sredine brida ulaznog i sredine brida izlaznog signala logičkih vrata, slika 1.6.



Slika 1.6. - Definicija vremena kašnjenja

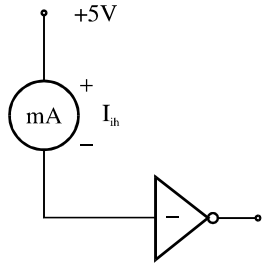
ZADATAK

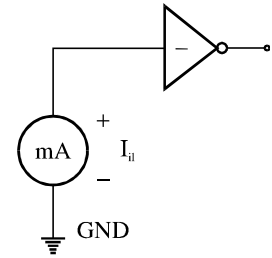
1. Za TTL, LSTTL, OCTTL i CMOS invertore na modelu izmjeriti struju potrošnje, vremena kašnjenja, ulazne struje i maksimalne izlazne struje. Izračunati faktore grananja i produkte kašnjenja i potrošnje. Pod strujom potrošnje podrazumijevamo struju koju sklop uzima iz izvora za napajanje (5V).

Struju potrošnje mjeriti tako da se očitani rezultat podijeli sa 4 za TTL, LSTTL i CMOS, te sa 6 za OCTTL invertore. Toliko je naime logičkih vrata integrirano u

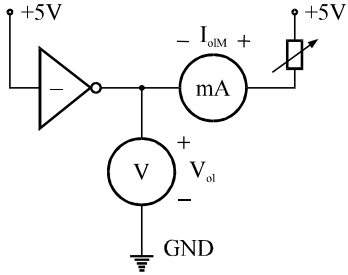
korištenim integriranim krugovima, a nožica za napajanje im je zajednička. Ulaze kod mjerenja pritegnuti u logičko 1. Za CMOS tehnologiju potrošnju izmjeriti i za ulazne signale u obliku niza impulsa (četvrtki) frekvencije 10kHz i 1MHz.

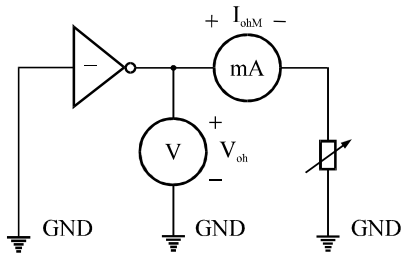
Struje I_{ih} i I_{il} mjeriti prema sljedećim mjernim shemama:





Struje I_{olM} i I_{ohM} mjeriti prema sljedećim shemama. Promjenljivim otpornikom povećavati izlaznu struju do postizanja graničnih uvjeta za izlazni napon:





Rezultate unijeti u tablicu:

	I_{cc} mA	I_{il} mA	I_{ih} mA	I_{olM} mA	I_{ohM} mA	FG_l	FG_h	FG	P_d mW	t_{dhl} ns	t_{dlh} ns	K nJ
TTL												
TTL OC												
TTL LS												
CMOS												
" 10kHz		—	—	—	—	—	—	—		—	—	
" 1MHz		—	—	—	—	—	—	—		—	—	

2. Za osnovne logičke sklopove na modelu snimiti tablice istine.

VJEŽBA 2: MINIMIZACIJA BOOLEOVIH FUNKCIJA

Pojednostavljenje ili minimizacija funkcija algebre logike važna je za realizaciju kombinacionih digitalnih sklopova, jer primjena neminimizirane funkcije implicira uporabu složenijih i time skupljih sklopova. Osnovni kriteriji minimizacije su minimalnost (broja tranzistora, logičkih vrata, integriranih krugova, potrošnje, površine), kašnjenje (minimalnost i jednolikost), postojanje metoda minimizacije i pogodnost algebarskog oblika za prijelaz na NI ili NILI vrata.

Minimizacija logičkih funkcija sastoji se u redukciji minterma (ili maksterma) i broja varijabli tako da u minimalnom obliku nalazimo nužne konjunkcije varijabli tipa minterma (ili disjunkcije varijabli tipa maksterma) koje nazivamo **elementarnim članovima**. Do elementarnih članova se dolazi iz PDNO ili PKNO pomoću susjednosti.

Dvije konjunkcije (disjunkcije) su **susjedne**, ako se sastoje od istih varijabli, a razlikuju se samo u stanju jedne. Primjenom postulata:

$$\overline{A} \vee A = 1 \quad \left(\overline{A} \& A = 0\right) \quad \text{i} \quad A \& 1 = A \quad \left(A \vee 0 = A\right)$$

reduciramo jedan član i jednu varijablu iz svakog para susjednih minterma (maksterma) koji pripadaju PDNO (PKNO). Na primjer:

$$\left(x_1 \& x_2 \& x_3\right) \vee \left(\overline{x}_1 \& x_2 \& x_3\right) = x_2 \& x_3 \& \left(\overline{x}_1 \vee x_1\right) =$$
$$= x_2 \& x_3 \& 1 = x_2 \& x_3$$

Neki od članova može biti susjedan sa dva druga člana, i ta susjednost može biti potrebna za postizanje minimalnog oblika. Zbog teorema $x \vee x = x$ ($x \& x = x$), izraz imamo pravo **proširiti** dopisivanjem postojećeg člana potreban broj puta, te na taj način iskoristiti sve susjednosti za potrebe minimizacije.

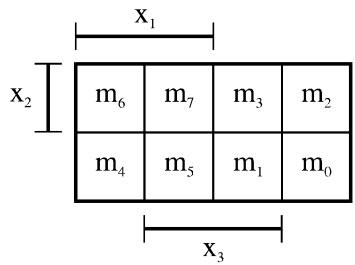
Ukoliko funkcija nije potpuno specificirana, tj. za neke kompleksije nije definirano preslikavanje, u postupku reduciranja funkciji **pridajemo** vrijednost koja nam odgovara za minimizaciju (0 ili 1). Efekt ovog postupka je ekvivalentan proširenju izraza postojećim članom, tj. postiže se skraćenje člana za jednu varijablu.

Metode minimizacije svode se načelno na ispitivanje susjednosti pojedinih minterma (maksterma). Jedna od metoda je metoda minimizacije pomoću **Veitch**eva **dijagrama**, kod kojega su područja susjednih kompleksija fizički susjedna. Pravila za minimizaciju pomoću Veitcheva dijagrama su:

1. Upisati funkciju u Veitchev dijagram i to samo 1 za PDNO, odnosno samo 0 za PKNO. U oba slučaja upisujemo R za redundantne kompleksije.

2. Svakom mintermu (makstermu) za koji je funkcija jednaka jedinici (nuli) tražimo susjeda. Ukoliko promatrani minterm (maksterm) nema susjednog člana, proglašavamo ga elementarnim članom.
3. Površine susjednih minterma (maksterma) udružujemo u veće pravokutnike koji predstavljaju izraze dužine n-1, gdje je n broj varijabli. Eliminirana je ona varijabla po kojoj se dva udružena područja razlikuju. Članovima dužine n-1 ponovo tražimo susjede koji također moraju imati dužinu n-1. Ukoliko ih pronađemo, eliminiramo još jednu varijablu i dobijemo član dužine n-2. Članovi koji nemaju susjede, smatraju se elementarnima. Višestruko zaokruživanje neke površine ekvivalentno je dopisivanju postojećeg člana normalnog oblika. Zaokruživanje redundantnog člana R ekvivalentno je njegovom dopisivanju u algebarski oblik.
4. Postupak ponavljamo sve dok ne ostanu samo elementarni članovi. Cilj je sve jedinice (nule) obuhvatiti što manjim brojem što većih površina.
5. Ispisujemo minimalni algebarski oblik funkcije (MDNO ili MKNO), tako da za svaku površinu ispišemo pripadni elementarni član. Uzimamo samo nužne elementarne članove.

Popis susjednih minterma za n = 3 dan je na slici 2.4. Pri tome je oznakom 0/1 označena susjednost članova m_0 i m_1, a oznakom 0-1-2-3 označen je član koji je dobiven udruživanjem minterma m_0, m_1, m_2 i m_3.



Slika 2.4. - Veitchev dijagram za n=3

Susjednost članova dužine 3:

po x_1: 0/4, 1/5, 2/6, 3/7

po x_2: 0/2, 1/3, 4/6, 5/7

po x_3: 0/1, 2/3, 4/5, 6/7

Susjednost članova dužine 2:

po x_1: 0-2/4-6, 1-3/5-7, 0-1/4-5, 2-3/6-7

po x_2: 1-5/3-7, 0-4/2-6, 0-1/2-3, 4-5/6-7

po x_3: 0-4/1-5, 2-6/3-7, 0-2/1-3, 4-6/5-7

Susjednost članova dužine 1:

po x_1: 4-5-6-7/0-1-2-3

po x_2: 0-1-4-5/2-3-6-7

po x_3: 0-2-4-6/1-3-5-7

Direktna realizacija minimalnog disjunktivnog normalnog oblika (MDNO) zahtijeva 3 vrste sklopova: I, ILI i NE. Stoga prelazimo u oblik pogodan za realizaciju sa NI operatorom (Shaeffer) primjenom **dvostruke negacije**:

$$f = \overline{\overline{f}}$$

Djelujemo li na MDNO donjom negacijom, možemo primijeniti De Morganov teorem:

$$\overline{(x_1 \vee x_2)} = \overline{x}_1 \& \overline{x}_2$$

pa će disjunkcije među elementarnim članovima preći u konjunkcije, a svi članovi koji su u stvari konjunkcije varijabli, bit će negirani. Novonastale konjunkcije zajedno s gornjom negacijom i negirani članovi predstavljaju NI operatore.

Za realizaciju NILI vratima, umjesto dualnog postupka preko PKNO koristimo MDNO inverzne (negirane) funkcije.

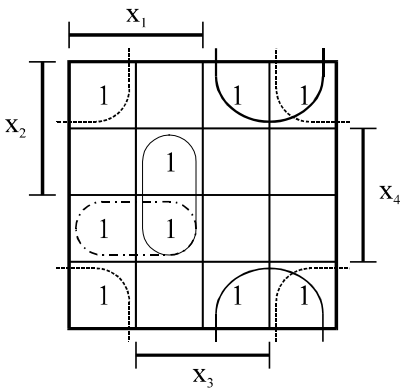
PRIMJER

Booleova funkcija zadana je u PDNO:

$f(x_1, x_2, x_3, x_4) = V\,(0,2,4,6,8,9,11,12,15)$

a) Minimizirati zadanu funkciju i realizirati pomoću NI operatora.

Vrijednosti funkcije upisujemo u Veitchev dijagram i minimiziramo:



Elementarni članovi su:

0-4-8-12

0-2-4-6

(minterme 0 i 4 koristimo 2 puta)

11-15

9-11 ili 8-9

(proizvoljno, samo jedan je nužan)

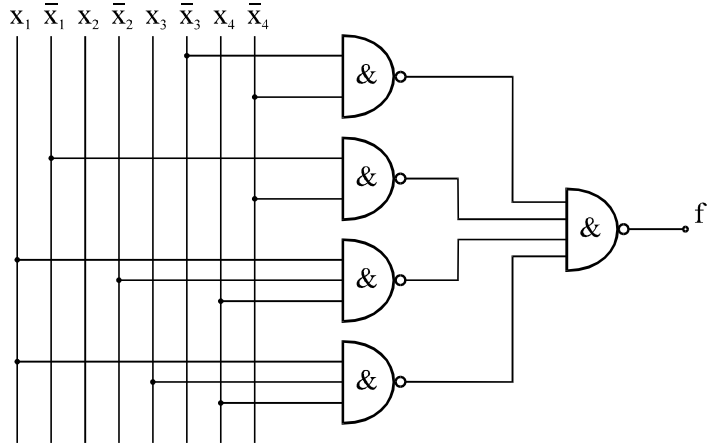
Slijedi da je:

$$f = (\overline{x}_3 \& \overline{x}_4) \vee (\overline{x}_1 \& \overline{x}_4) \vee (x_1 \& \overline{x}_2 \& x_4) \vee (x_1 \& x_3 \& x_4)$$

Primjenom dvostruke negacije i De Morganovih teorema dobijemo:

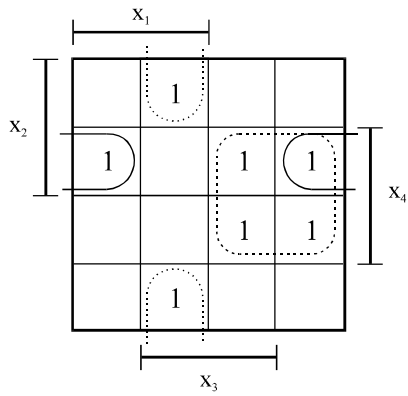
$$f = \overline{\overline{f}} = \overline{\overline{x}_3 \& \overline{x}_4} \,\&\, \overline{\overline{x}_1 \& \overline{x}_4} \,\&\, \overline{x_1 \& \overline{x}_2 \& x_4} \,\&\, \overline{x_1 \& x_3 \& x_4}$$

Nacrtamo shemu:



b) Minimizirati zadanu funkciju i realizirati pomoću NILI operatora.

Vrijednosti inverzne funkcije upisujemo u Veitchev dijagram i minimiziramo:



Elementarni članovi su:

1-3-5-7

5-13

(minterm 5 koristimo 2 puta)

10-14

Slijedi da je:

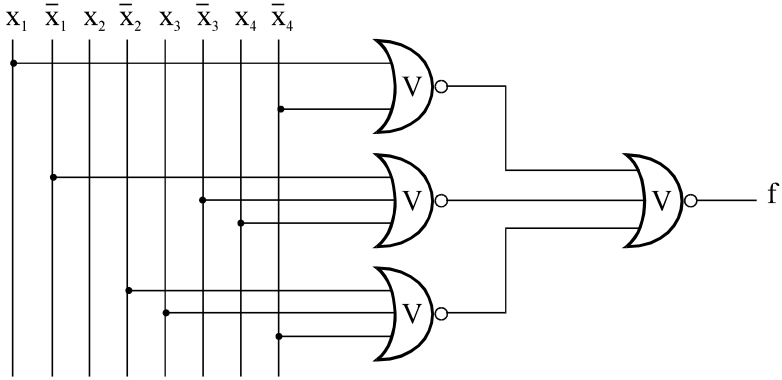
$$\bar{f} = (\bar{x}_1 \,\&\, x_4) \vee (x_1 \,\&\, x_3 \,\&\, \bar{x}_4) \vee (x_2 \,\&\, \bar{x}_3 \,\&\, x_4)$$

Negacijom obiju strana i primjenom dvostruke negacije i De Morganovih teorema nad pojedinačnim članovima dobijemo:

$$\bar{\bar{f}} = f = \overline{\overline{(\bar{x}_1 \,\&\, x_4)} \vee \overline{(x_1 \,\&\, x_3 \,\&\, \bar{x}_4)} \vee \overline{(x_2 \,\&\, \bar{x}_3 \,\&\, x_4)}} =$$

$$= \overline{x_1 \vee \bar{x}_4} \;\vee\; \overline{\bar{x}_1 \vee \bar{x}_3 \vee x_4} \;\vee\; \overline{\bar{x}_2 \vee x_3 \vee \bar{x}_4}$$

Nacrtamo shemu:



ZADATAK

1. Zadanu funkciju minimizirati i realizirati pomoću NI vrata na laboratorijskom modelu.
2. Zadanu funkciju minimizirati i realizirati pomoću NILI vrata na laboratorijskom modelu.
3. Zadanu funkciju minimizirati i simulirati na računalu.

VJEŽBA 3: HAMMINGOV KODER, DEKODER I KOREKTOR

Pojmove kao elemente konačnog skupa pojmova u digitalnoj tehnici prikazujemo jednoznačno simbolima u obliku kompleksija (kodnih riječi) binarnih znamenki. Postupak dodjeljivanja kompleksije nekom pojmu zovemo **kodiranje**. Ukoliko imamo N pojmova, moramo izabrati kompleksije s takvim brojem bita m, da za svaki pojam imamo jedinstvenu kombinaciju bita u kompleksiji. Vrijedi uvjet:

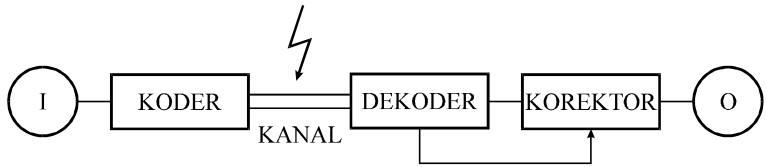
$$2^m \geq N$$

Ukoliko je broj pojmova jednak cjelobrojnoj potenciji broja 2, tada će sve kompleksije biti iskorištene i govorimo o **koncentriranom kodu**. U suprotnom, imat ćemo slobodnih kompleksija, pa informacijski kapacitet koda nije u potpunosti iskorišten. Govorimo o **redundantnom kodu**.

Informacije prenosimo komunikacijskim kanalima u obliku poruka. Na putu kroz kanal one su izložene djelovanju smetnji, pa očekujemo da ćemo pojedine bitove primiti neispravno (u praksi 1 na 10^6). Neispravno primljen bit u okviru binarnog sustava znači prijem 0 umjesto 1 ili obrnuto. Problem je kako prepoznati neispravne kompleksije (detekcija) i ispraviti ih (korekcija pogrješki).

Nastanak pogrješke u okviru jedne kompleksije prevodi tu kompleksiju u neku drugu. Ukoliko ova posljednja ima neko značenje u kodu, pogrješku nije moguće detektirati. Moramo imati neke kompleksije koje nemaju značenja kako bismo ih mogli prepoznati, a postupak kodiranja mora osigurati da će kod nastanka specificiranog maksimalnog broja pogrješki originalna kompleksija biti prevedena u neku bez značenja. Svjesno uvođenje kompleksija bez značenja znači povećavanje redundancije koda.

Kod korištenja redundantnih kodova komunikacijski se sustav sastoji od izvorišta informacije koje generira koncentrirani kod, kodera koji koncentrirani kod prevodi na redundantni, komunikacijskog kanala, dekodera koji detektira pogrješku, korektora koji je korigira (ako to kod omogućuje) i odredišta informacije. Dekoder i korektor prevode redundantni kod natrag u koncentrirani, slika 3.1.



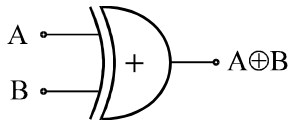
Slika 3.1. - Model komunikacijskog sustava

Broj bitova u kojima se razlikuju dvije kompleksije nazivamo **kodnom udaljenošću** ili **distancom**. Dvije kompleksije sa distancom 1 nazivamo **susjednima**.

Ako očekujemo djelovanje najviše L pogrješki u okviru jedne kompleksije (ovisno o statističkim karakteristikama komunikacijskog kanala), neispravne kompleksije će se od originalne razlikovati u najviše L bitova. Dakle, uvjet detekcije pogrješki je da minimalna distanca između bilo koje dvije ispravne kompleksije bude L+1, odnosno uvjet korekcije pogrješki je minimalna distanca od 2L+1.

Problem je kako konstruirati kod da bude zadovoljen uvjet distance, a da proces kodiranja i dekodiranja bude što jednostavniji. U praksi su nam interesantni kodovi s paritetnim ispitivanjima, među kojima su i Hammingovi kodovi.

Kodovi sa **paritetnim ispitivanjima** su takvi kodovi kod kojih se bitovima originalne kompleksije koncentriranog koda dodaju paritetni kontrolni bitovi, tako da za definirano paritetno ispitivanje broj jedinica bude paran ili neparan. Paritetna ispitivanja sklopovski realiziramo sumatorima po modulu 2 (ekskluzivno ILI), slika 3.2.



A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Slika 3.2. - Sumator po modulu 2 (ekskluzivno ILI)

Hammingovi kodovi koriste takva paritetna ispitivanja koja omogućavaju detekciju i korekciju jedne pogrješke, a neposredni rezultat ispitivanja pokazuje koji je bit kompleksije primljen pogrješno.

Svakoj kompleksiji od m informacijskih elemenata dodajemo k kontrolnih elemenata. Od k kontrolnih elemenata možemo formirati 2^k binarnih kompleksija, od kojih svaka predstavlja tzv. položajni broj čiji dekadski ekvivalent daje položaj pogrješke. Taj broj mora sadržavati dovoljno elemenata da se položaj opiše jednoznačno. Potrebna je i jedna kompleksija kojom konstatiramo stanje "nema pogrješke" (uzima se ona koja sadrži same nule). Stoga treba biti zadovoljena relacija:

$$2^k \geq m+k+1$$

Hammingov kod generiramo pomoću matrice H(n,k) gdje je n=m+k u koju upisujemo n stupaca od po k elemenata. Sadržaj svakog stupca je prirodno binarno kodiran redni broj tog stupca sa početkom od broja 00..01.

$$H(n,k) = \begin{vmatrix} 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \cdots & 1 & 1 \\ 1 & 0 & \cdots & 0 & 1 \end{vmatrix}$$

Množenjem Hammingove matrice sa vektorom čiji su elementi bitovi kompleksije Hammingova koda (dužine n elemenata) dobivamo sustav od k jednadžbi koje nam predstavljaju paritetna ispitivanja u dekoderu.

Da bismo dobili vrijednost kontrolnih bitova ovaj sustav moramo riješiti po izabranim nepoznanicama. Očito je da se neki bitovi kodne kompleksije pojavljuju u samo jednoj jednadžbi. To su oni u čijem se stupcu Hammingove matrice pojavljuje samo jedna jedinica. Izborom ovih bitova za kontrolne, odnosno za nepoznanice u sustavu jednadžbi, dobivamo k linearno nezavisnih jednadžbi sa po jednom nepoznanicom, čije je rješenje trivijalno.

Uzmimo na primjer kod sa n=7, m=4 i k=3. Tada imamo:

$$H(7,3) \cdot |A| = \begin{vmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{vmatrix} = 0$$

što daje sljedeći sustav jednadžbi dekodera:

$$a_4 \oplus a_5 \oplus a_6 \oplus a_7 = 0$$

$$a_2 \oplus a_3 \oplus a_6 \oplus a_7 = 0$$

$$a_1 \oplus a_3 \oplus a_5 \oplus a_7 = 0$$

izaberemo li a_1, a_2 i a_4 za kontrolne bitove, te korištenjem svojstva $a \oplus a = 0$ dobijemo jednadžbe kodera:

$$a_4 = a_5 \oplus a_6 \oplus a_7$$

$$a_2 = a_3 \oplus a_6 \oplus a_7$$

$$a_1 = a_3 \oplus a_5 \oplus a_7$$

Pojedinačna pogrješka može nastati na bilo kojem od m+k elemenata. Rezultat pojedinog ispitivanja će biti 1 ako je pogrješno primljeni bit sadržan u pripadnoj

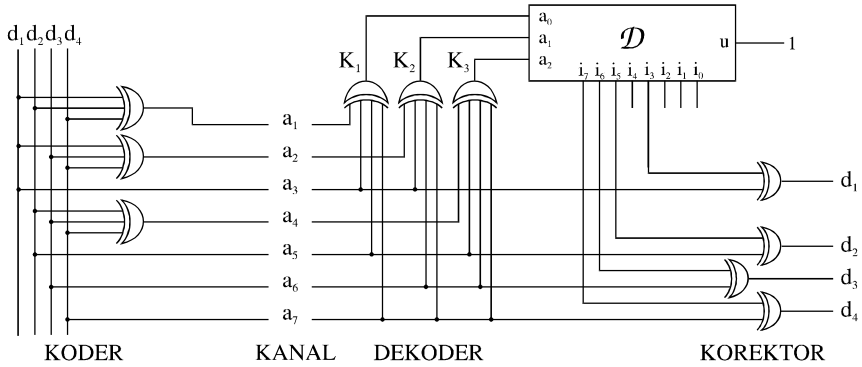
jednadžbi. Rezultat ostalih ispitivanja će biti 0. Nakon izvršenih k paritetnih ispitivanja, od dobivenih rezultata formiramo kompleksiju bita. S obzirom na formu Hammingove matrice, kompleksija će u prirodnom binarnom kodu prikazati redni broj pogrješnog bita, odnosno lokaciju pogrješke. Ukoliko nema pogrješke, dobit ćemo kompleksiju 0.

Npr. uz m=4 mora biti k=3, a to je 2^3=8 pozicionih brojeva u svrhu lokacije pogrješke na nekoj od m+k=7 pozicija:

poz. broj	000	001	010	011	100	101	110	111
pozicija	ispr.	1	2	3	4	5	6	7

Ukoliko je pogrješno primljen informacijski bit, potrebno ga je korigirati. Zbog karaktera binarnog sustava pogrješka se pojavljuje kao promjena iz 0 u 1 i obratno, pa se i korekcija svodi na komplementiranje, odnosno promjenu iz 0 u 1 i obratno.

Komplementiranje možemo vršiti sumatorom po modulu 2 ako jedan od ulaza koristimo kao kontrolni. Za pogrješni bit kontrolni ulaz sumatora mora biti 1, a za ostale (ispravne) moraju biti 0. Dakle, binarnu informaciju o poziciji pogrješnoga bita moramo prevesti u kontrolni signal za taj bit. To postižemo primjenom sklopa koji aktivira onaj svoj izlaz čiji redni broj odgovara binarnom broju dovedenom na njegove adresne ulaze, a to je demultiplekser/dekoder. Ostaje nam realizirati potrebne sklopove.



ZADATAK

1. Konstruirati minimalni Hammingov koder i dekoder. Zadan je broj informacijskih bitova. Realizirati korištenjem EX-ILI, EX-NILI i generatora pariteta na modelu, te provjeriti rad sustava za jednostruke pogrješke.
2. Ispitati funkcionalnost generatora pariteta, sumatora i komparatora.

VJEŽBA 4: REALIZACIJA LOGIČKIH FUNKCIJA POMOĆU MULTIPLEKSERA I DEMULTIPLEKSERA

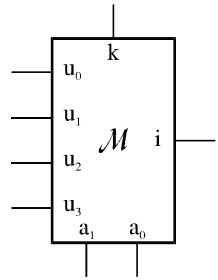
Metode minimizacije Booleovih funkcija vode ka realizaciji sklopova korištenjem logičkih vrata. Ova vrata realiziraju se integriranim krugovima niskog stupnja integracije (SSI).

Kod srednjeg stupnja integracije (MSI) imamo do 100 logičkih vrata (ili oko 1000 komponenti) na jednoj pločici silicija. Međutim, primjena poznatih metoda realizacije Booleovih funkcija zahtijeva pristup svim ulazima i izlazima logičkih vrata. To onemogućava primjenu MSI integriranih krugova jer smo ograničeni brojem praktično i ekonomično izvedenih nožica njihovih kućišta. MSI tehnologija interesantna je za realizaciju posebnih funkcionalnih struktura. Uz ostale, u tu grupu sklopova spadaju digitalni multiplekseri i demultiplekseri. Ovdje ćemo spomenuti i enkodere prioriteta.

Multiplekser je logički sklop koji na informacijski izlaz **i** propušta vrijednost onog od $n=2^m$ informacijskih ulaza $u_0...u_{n-1}$, čiji je redni broj prisutan u prirodnom binarnom obliku na m adresnih ulaza $a_0...a_{m-1}$. Na osnovu ove definicije, moguće je napisati algebarski izraz multipleksera:

$$i = \bigvee_{j=0}^{2^m-1} m_j \& u_j$$

gdje je m_j minterm adresnih varijabli. Stvarni multiplekser raspolaže i s kontrolnim ulazom k, čiji aktivni nivo uključuje izlazno pojačalo. Simbol multipleksera za m=2 prikazan je na slici 4.1.



Slika 4.1. - Multiplekser m=2

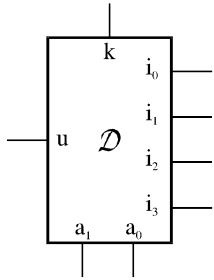
Multiplekser možemo koristiti kao selektor kada su vrijednosti adresnih varijabli stacionarne u odnosu na ulazne varijable. Ukoliko se adresne varijable mijenjaju ciklički uz stacionarne ulazne varijable, govorimo o paralelno-serijskoj konverziji. Multiplekseri se standardno izvode sa po 1, 2, 3 i 4 adresna ulaza, odnosno 2, 4, 8 i 16 informacijskih ulaza. U okviru jednog integriranog kruga,

obično se realizira više multipleksera sa zajedničkim adresnim ulazima (zbog spomenutog ograničenja broja nožica na kućištu).

Demultiplekser je logički sklop koji vrijednost informacijskog ulaza **u** propušta na onaj od $n=2^m$ informacijskih izlaza $i_0...i_{n-1}$, čiji je redni broj prisutan u prirodnom binarnom obliku na m adresnih ulaza $a_0...a_{m-1}$. Na osnovu ove definicije, moguće je napisati algebarski izraz multipleksera:

$$i_j = m_j \& u$$

gdje je m_j minterm adresnih varijabli. Stvarni demultiplekser raspolaže i s kontrolnim ulazom k, čiji aktivni nivo uključuje izlazna pojačala. Simbol demultipleksera za m=2 prikazan je na slici 4.2.



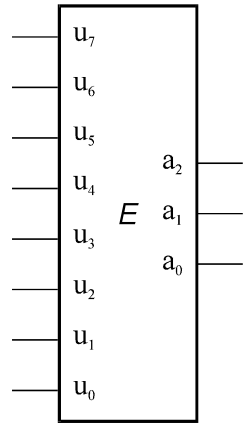
Slika 4.2. - Demultiplekser m=2

Demultiplekser možemo koristiti kao dekoder, kada je potrebno aktivirati jedan od n izlaza, te za serijsko-paralelnu konverziju.

Enkoder prioriteta je logički sklop koji na m adresnih izlaza generira u prirodnom binarnom obliku redni broj najznačajnijeg aktivnog ulaza od $n=2^m$ informacijskih ulaza. Definiramo ga funkcionalnom tablicom:

u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	a_2	a_1	a_0
1	X	X	X	X	X	X	X	1	1	1
0	1	X	X	X	X	X	X	1	1	0
0	0	1	X	X	X	X	X	1	0	1
0	0	0	1	X	X	X	X	1	0	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	0	0	1	X	1	0	1
0	0	0	0	0	0	0	1	0	0	0

U gornjem primjeru aktivna razina je 1, a u_7 ima najviši prioritet. Simbol enkodera prioriteta za m=3 prikazan je na slici 4.3.



Slika 4.3. - Enkoder prioriteta m=3

Enkoder prioriteta ne koristimo za realizaciju Booleovih funkcija.

REALIZACIJA BOOLEOVIH FUNKCIJA

Primjena multipleksera i demultipleksera u realizaciji Booleovih funkcija zahtijeva razradu novih metoda minimizacije. U okviru ove vježbe razmatrat ćemo realizaciju Booleovih funkcija pomoću multipleksera i demultipleksera.

Multiplekser ima jedan izlaz, na kojem se treba pojaviti očekivana vrijednost funkcije. Stoga vrijedi i = f(x), odnosno:

$$\bigvee_{j=0}^{2^m-1} m_j(a) \& u_j = \bigvee_{j=0}^{2^m-1} m_j(x) \& T_j$$

Primjećujemo strukturnu sličnost lijeve i desne strane gornjeg izraza. Za n=m imamo trivijalno rješenje:

$$a_k = x_{m-k},\ k=0...m-1 \qquad \text{i} \qquad u_j = T_j,\ j=0...2^m-1$$

Funkciju realiziramo tako da na adresne ulaze multipleksera dovedemo varijable funkcije, a na informacijske ulaze multipleksera vrijednost funkcije (konstante 0 i 1).

Za slučaj n>m, trivijalno rješenje nije moguće. Izraz za PDNO funkcije, gdje nalazimo članove tipa minterma dužine n varijabli, treba transformirati na način da se ponovo uspostavi strukturna sličnost.

Uočimo da su mintermi m(x) kompleksije svih n varijabli, i da neki od njih imaju prvih m varijabli iste vrijednosti. Takvih grupa minterma sa po m istih prvih varijabli ima 2^m, a svaka takva grupa broji 2^{n-m} članova. Na osnovu svojstva asocijativnosti pojedine minterme možemo pisati razbijene u dva dijela:

$$m(x_1....x_n) = m(x_1...x_m)\&m(x_{m+1}...x_n)$$

Minterme s istim prvim dijelom grupiramo u PDNO i na osnovu svojstva distributivnosti izlučimo zajednički faktor:

$$f(x) = \bigvee_{j=0}^{2^m-1} m(x_1...x_m)\&\left(\bigvee_{k=0}^{2^{n-m}-1} m(x_{m+1}...x_n)\& T_{j\cdot 2^{n-m}+k}\right)$$

Izraz u zagradi je PDNO neke funkcije od $x_{m+1}...x_n$, koju nazivamo **preostalom funkcijom**, zavisnom od **j**. Varijable $x_{m+1}...x_n$ nazivamo **preostalim varijablama.**

$$f_j(x_{m+1}...x_n) = \bigvee_{k=0}^{2^{n-m}-1} m(x_{m+1}...x_n)\& T_{j\cdot 2^{n-m}+k}$$

Funkciju možemo pisati kao:

$$f(x) = \bigvee_{j=0}^{2^m-1} m_j(x_1...x_m)\& f_j$$

Postignuta je strukturna sličnost sa izrazom za izlaz multipleksera, pa opet imamo trivijalno rješenje:

$$a_k = x_{m-k},\ k=0...m-1 \qquad i \qquad u_j = f_j,\ j=0...2^m-1$$

Funkciju realiziramo tako, da na adresne ulaze multipleksera dovodimo izabranih (u gornjem izvodu prvih) m varijabli x, a na informacijske ulaze multipleksera 2^m preostalih funkcija f_j.

Preostale funkcije realiziramo logičkim vratima ili multiplekserima. U posljednjem slučaju imamo strukturu koja se zove **multipleksersko stablo**.

Veličina sklopova preostalih funkcija ovisi o izboru adresnih varijabli za multiplekser. Za slučaj realizacije logičkim vratima, interesantan nam je što manji broj logičkih vrata, dakle što bolja mogućnost minimizacije preostalih funkcija. Kod korištenja multiplekserskog stabla, želimo eliminirati pojedine grane stabla, pa nam je interesantno da što veći broj preostalih funkcija bude funkcija jedne varijable (koju realiziramo samo jednim invertorom). Stoga je poseban slučaj n=m+1, jer su sve preostale funkcije zapravo funkcije jedne varijable.

Demultiplekser kod realizacije Booleovih funkcija koristimo kao dekoder. Ako informacijski ulaz spojimo u jedinicu, svaki od 2^m izlaza demultipleksera realizira jedan minterm:

$$i_j = m_j\&u, \quad u=1 \quad \Rightarrow i_j = m_j$$

Za n=m imamo trivijalno rješenje:

$$f(x) = \bigvee_{j=0}^{2^m-1} m_j(x)\& T_j = \bigvee_{j=0}^{2^m-1} m_j(a)\& T_j = \bigvee_{j=0}^{2^m-1} i_j \& T_j$$

Funkciju realiziramo tako da na informacijski ulaz demultipleksera dovedemo jedinicu, na adresne ulaze demultipleksera dovedemo varijable funkcije:

$$a_k = x_{m-k},\ k=0...m-1$$

a na vanjska ILI vrata dovedemo one izlaze demultipleksera i_j, za koje je vrijednost funkcije T_j jednaka jedinici. Stvarni demultiplekseri često imaju invertirane (negirane) izlaze, pa pišemo:

$$f(x) = \bigvee_{j=0}^{2^m-1} i_j \,\&\, T_j = \overline{\overline{\bigvee_{j=0}^{2^m-1} i_j \,\&\, T_j}} = \overline{\overline{\bigwedge_{j=0}^{2^m-1} i_j \,\&\, T_j}}$$

tj. na vanjska NI vrata spajamo one negirane izlaze, za koje je vrijednost funkcije jednaka jedinici.

Za slučaj n>m, trivijalno rješenje nije moguće. Izraz za PDNO funkcije, gdje nalazimo članove tipa minterma dužine n varijabli, treba transformirati na način da se ponovo pojave članovi tipa minterma duljine m. Istim postupkom kao kod multipleksera funkciju razbijemo na parcijalne (preostale) funkcije:

$$f(x) = \bigvee_{j=0}^{2^m-1} m_j(x_1 \ldots x_m) \& f_j \quad ; \quad f_j(x_{m+1} \ldots x_n) = \bigvee_{k=0}^{2^{n-m}-1} m(x_{m+1} \ldots x_n) \& T_{j\cdot 2^{n-m}+k}$$

Postignuta je ista veličina članova kao kod izraza za izlaz demultipleksera, pa opet imamo trivijalno rješenje:

$$f(x) = \bigvee_{j=0}^{2^m-1} m_j(x) \& f_j = \bigvee_{j=0}^{2^m-1} m_j(a) \& f_j = \bigvee_{j=0}^{2^m-1} i_j \,\&\, f_j$$

$$a_k = x_{m-k},\ k=0...m-1$$

Funkciju realiziramo tako, da na adresne ulaze demultipleksera dovodimo izabranih (u gornjem izvodu prvih) m varijabli x, a informacijskim izlazima demultipleksera aktiviramo 2^m preostalih funkcija f_j. Problem je konjunkcija $i_j f_j$.

Ako preostale funkcije realiziramo demultiplekserom, dobijemo **demultipleksersko stablo**. Pri tome koristimo ulaze demultipleksera preostalih funkcija koji su s njima konjunktivno vezani.

Veličina sklopova preostalih funkcija ovisi o izboru adresnih varijabli za demultiplekser. Kod demultiplekserskog stabla, želimo eliminirati pojedine grane stabla, pa nam je interesantno da što veći broj preostalih funkcija bude konstanta 0 ili 1.

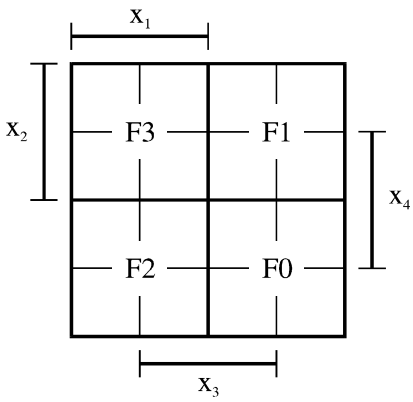
I ovdje je poseban slučaj n=m+1, jer možemo pretpostaviti da koristimo demultipleksersko stablo s osnovnim demultiplekserom m=1. Taj realizira samo minterme jedne varijable (x i $\overline{x}$), a to su funkcije jedne varijable, pa nam takav osnovni demultiplekser nije potreban. Neposredno koristimo dva demultipleksera druge razine m=n-1.

Preostale funkcije izračunavamo korištenjem Veitchevih dijagrama. Izborom jedne varijable, PDNO funkcije i Veitchev dijagram se raspadaju na dva dijela. Svaki dio predstavlja po jednu preostalu funkciju. Izborom sljedećih adresnih varijabli, PDNO i Veitchev dijagram se raspadaju na 4, 8, itd. dijelova. Nakon upisivanja funkcije u Veitchev dijagram, postupak izbora adresnih varijabli obavljamo u sljedećim koracima:

1. Sagledamo mogućnost minimizacije i za adresnu varijablu izaberemo onu po kojoj je najteže vršiti minimizaciju.
2. Podijelimo Veitchev dijagram na dva dijela s obzirom na izabranu adresnu varijablu, te za cijeli dijagram ponavljamo postupak pod 1, tako da je sljedeća izabrana varijabla zajednička za sva područja.
3. U okviru dobivenih područja vršimo minimizaciju preostalih funkcija koje realiziramo logičkim vratima i dovodimo na odgovarajuće ulaze multipleksera.

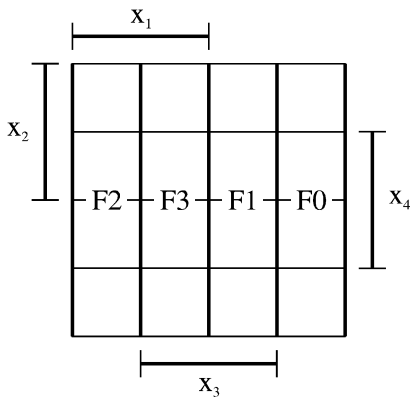
Za funkciju sa četiri varijable moguće je kod realizacije sa multiplekserom sa dva adresna ulaza (m=2) ostvariti šest različitih kombinacija adresnih ulaza, za koje podjela Veitchevog dijagrama na područja izgleda kao na slikama 4.4. - 4.9.

podjela po x_1 i x_2



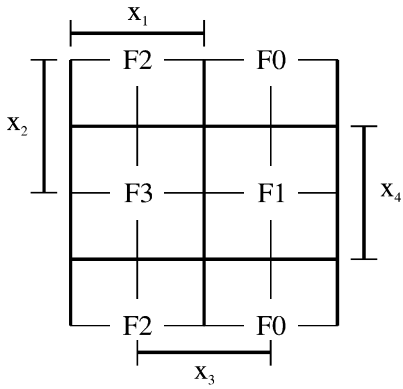
Slika 4.4.

podjela po x_1 i x_3



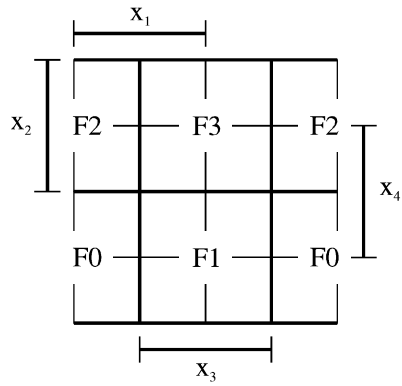
Slika 4.5.

podjela po x_1 i x_4



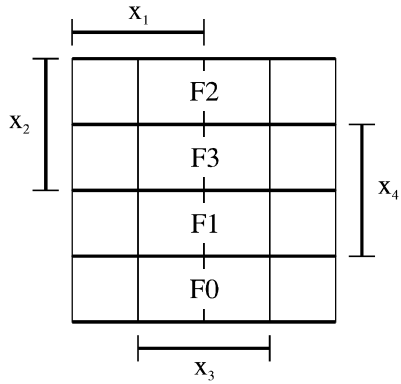
Slika 4.6.

podjela po x_2 i x_3



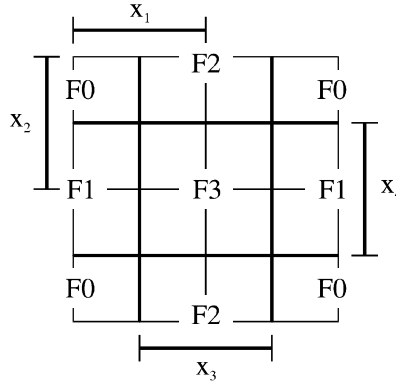
Slika 4.7.

podjela po x_2 i x_4



slika 4.8.

podjela po x_3 i x_4



slika 4.9.

PRIMJER

1) Realiziraj funkciju četiri varijable (n=4) korištenjem multipleksera m=3. Funkcija je zadana u PDNO:

$$f(x_1, x_2, x_3, x_4) = \vee (0, 3, 4, 5, 9, 10, 12, 13)$$

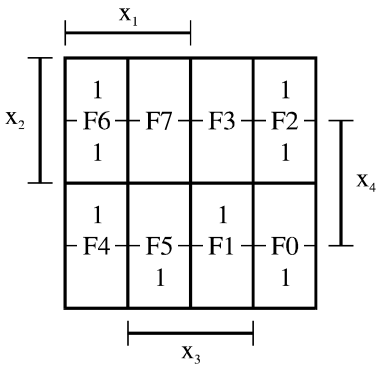
Algebarski bi postupak izgledao kako slijedi:

$$f = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 \vee \overline{x}_1\overline{x}_2x_3x_4 \vee \overline{x}_1x_2\overline{x}_3\overline{x}_4 \vee \overline{x}_1x_2\overline{x}_3x_4 \vee$$
$$\vee x_1\overline{x}_2\overline{x}_3x_4 \vee x_1\overline{x}_2x_3\overline{x}_4 \vee x_1x_2\overline{x}_3\overline{x}_4 \vee x_1x_2\overline{x}_3x_4$$

Prvo odredimo tri adresne varijable (x_1, x_2, x_3) i odredimo adresne ostatke koji će predstavljati ulaze multipleksera.

$$f = \overline{x}_1\overline{x}_2\overline{x}_3\left(\overline{x}_4\right) \vee \overline{x}_1\overline{x}_2x_3\left(x_4\right) \vee \overline{x}_1x_2\overline{x}_3\left(x_4 \vee \overline{x}_4\right) \vee x_1\overline{x}_2\overline{x}_3\left(x_4\right) \vee$$
$$\vee\, x_1\overline{x}_2x_3\left(\overline{x}_4\right) \vee x_1x_2\overline{x}_3\left(x_4 \vee \overline{x}_4\right)$$

Očito je da se adresni ostaci daleko jednostavnije određuju pomoću Veitchevog dijagrama:



Slijedi:

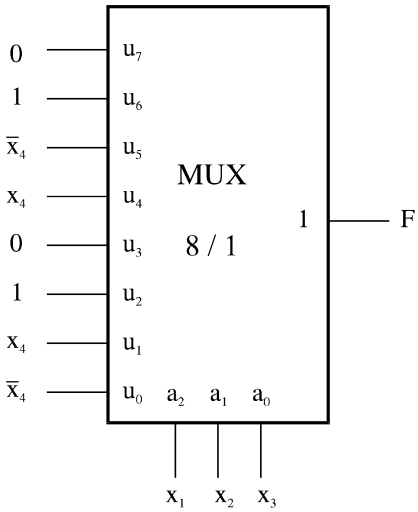
$F0 = \overline{x}_4$ $\quad F1 = x_4$

$F2 = 1$ $\quad F3 = 0$

$F4 = x_4$ $\quad F5 = \overline{x}_4$

$F6 = 1$ $\quad F7 = 0$

Nacrtamo shemu:

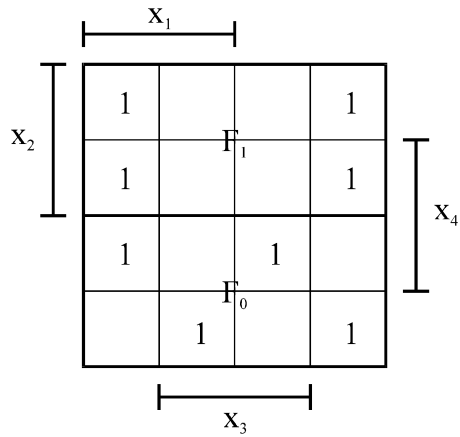


2) Realiziraj istu funkciju demultipleksera m=3.

Funkcija je zadana u PDNO:

$$f(x_1, x_2, x_3, x_4) = \vee (0, 3, 4, 5, 9, 10, 12, 13)$$

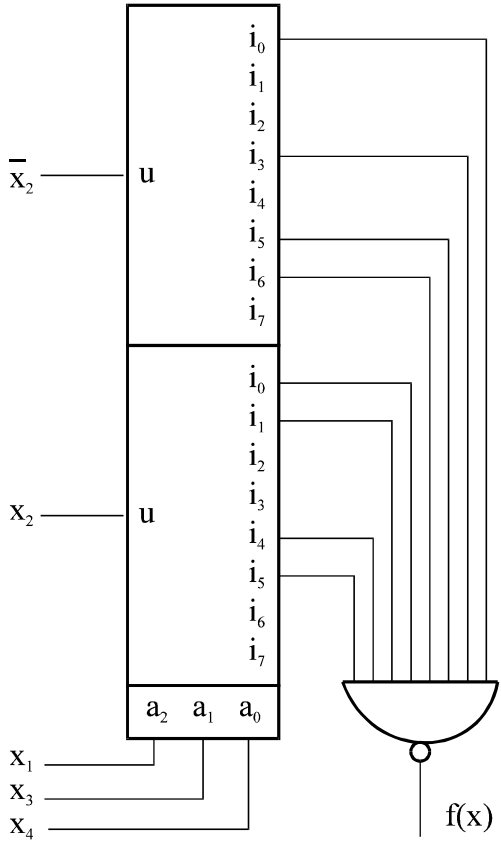
Funkciju odmah upisujemo u Veitchev dijagram. Izaberimo varijablu x_2 za adresnu varijablu (nepostojećeg) osnovnog demultipleksera m=1.



Preostale funkcije su funkcije tri varijable, pa imamo trivijalni slučaj.

x_1	x_3	x_4	F_0	F_1
0	0	0	1	1
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	1	0
1	1	1	0	0

Nacrtamo shemu:

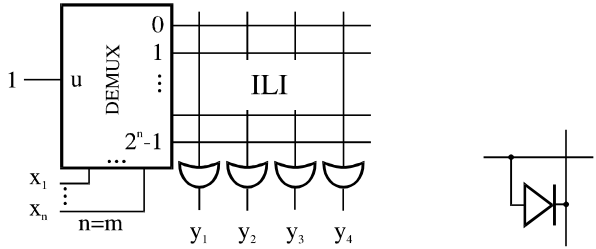


ZADATAK

1. Zadanu Booleovu funkciju minimizirati i realizirati pomoću multipleksera. Verificirati funkcionalnost sklopa na laboratorijskom modelu.
2. Zadanu Booleovu funkciju minimizirati i realizirati pomoću demultipleksera. Verificirati funkcionalnost sklopa na laboratorijskom modelu.
3. Ispitati funkcionalnost multipleksera, demultipleksera i enkodera prioriteta.

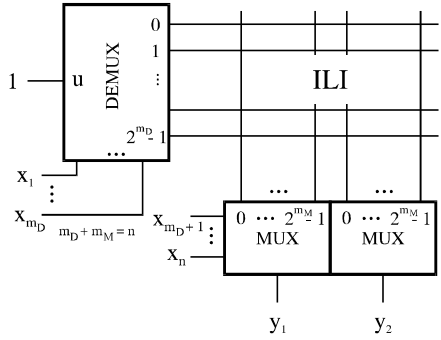
VJEŽBA 5: PROGRAMABILNE LOGIČKE STRUKTURE

Realizacija Booleovih funkcija pomoću demultipleksera m=n zasniva se na činjenici da demultiplekser zapravo realizira sve minterme PDNO funkcije. Potrebne minterme dovoljno je povezati na ILI vrata, realizirana npr. u diskretnoj diodnoj tehnici. Ovakva struktura posebno je pogodna za realizaciju više funkcija istih varijabli, pa govorimo o **demultiplekseru i diodnoj ILI matrici**, slika 5.1.



Slika 5.1. - Struktura sa programabilnom ILI matricom

Problem ove realizacije je dimenzija diodne matrice, koja bi trebala biti kvadratnog oblika. To postižemo **multipleksersko-demultiplekserskom strukturom**, slika 5.2.



Slika 5.2. - MD struktura

Trebaju biti zadovoljene jednadžbe:

$$m_D + m_M = n$$

$$2^{m_D} \approx M \cdot 2^{m_M}$$

gdje su:

m_D - broj adresnih varijabli demultipleksera

m_M - broj adresnih varijabli multipleksera

M - broj multipleksera = broj funkcija

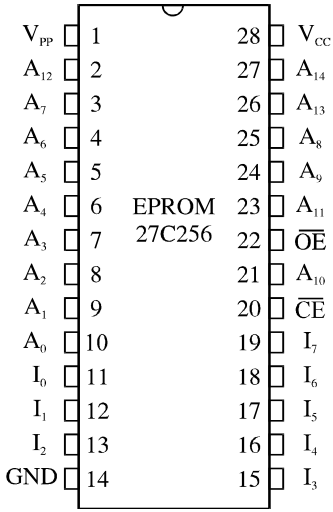
To je osnovica za realizaciju svih memorijskih struktura:

- ROM (Read Only Memory), sadržaj upisan kod izrade,
- PROM(Programmable ROM), sadržaj se mijenja pregaranjem osigurača,
- RAM (Random Access Memory), koji može biti statički (SRAM), realiziran pomoću bistabila i dinamički (DRAM), koji koristi kapacitet između vrata i kanala MOSFETa

- EPROM (Erasable PROM), koristi tehnologiju lebdećih vrata, briše se UV svjetlom,
- EEPROM (Electrically EPROM), lebdeća vrata, brisanje električko bajt po bajt
- FEPROM (Flash EPROM), lebdeća vrata, brisanje električko blok po blok

Memorije koristimo za pohranjivanje podataka i programa.

Za realizaciju Booleovih funkcija možemo koristiti **ROM**, a posebno **EPROM** zbog niske cijene i mogućnosti reprogramiranja (ako zadovoljava brzina rada). Jedna od mnogih EPROM komponenti je i 27C256, EPROM s matricom kapaciteta 256Kbita. Matrica je organizirana kao 32Kx8, odnosno 32Kbyte, slika 5.3. Ovom komponentom možemo realizirati 8 proizvoljnih funkcija 15 ulaznih varijabli.

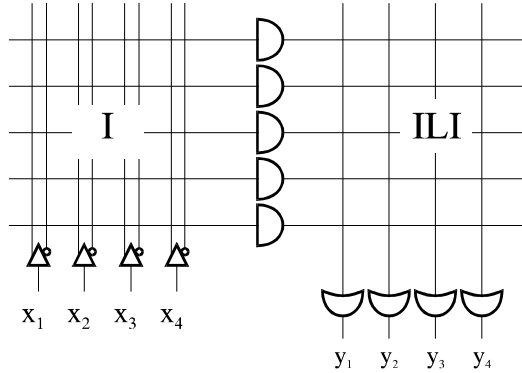


A_0 - A_{14}	adrese
CE	Chip Enable
OE	Output Enable
I_0 - I_7	izlazi

Slika 5.3. - EPROM 27C256

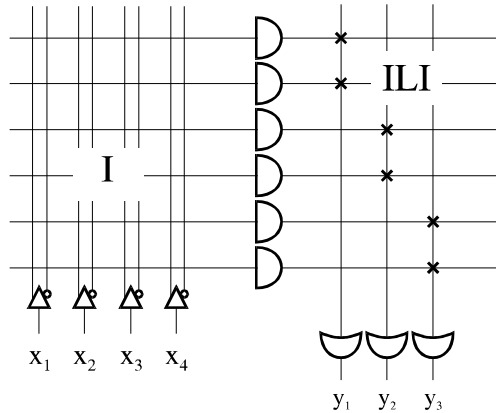
Uočimo da se demultiplekser sastoji od niza I vrata, koja realiziraju minterme. Sklop sa slike 5.1. shematski možemo prikazati strukturom sa I i ILI matricom, slika 5.4., kod koje je I matrica nepromjenljiva (fiksna), a ILI matrica promjenljiva (programabilna).

Realizacija s fiksnom I matricom ne omogućava minimizaciju funkcija, te se prelazi na **PLD** (Programmable Logic Device) strukture, s programabilnom I i programabilnom ILI matricom, izvedene u LSI tehnologiji.



Slika 5.4. - Struktura sa I i ILI matricom

U praksi nam programabilna ILI matrica najčešće nije potrebna, jer će se rijetko različite funkcije sastojati od istih elementarnih članova. Uvodimo strukturu s fiksnom ILI matricom, pod nazivom **PAL** (Programmable Array Logic) , slika 5.5.

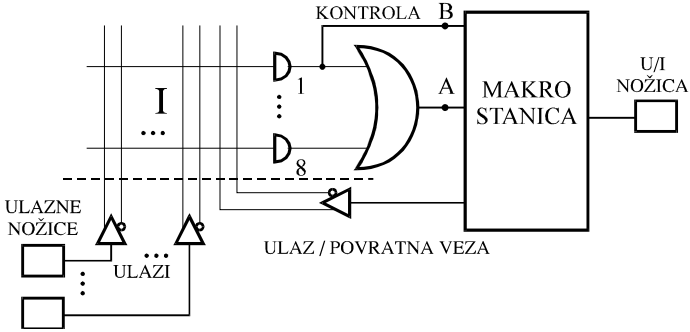


Slika 5.5. - Struktura s fiksnom ILI matricom

I matricom realiziramo elementarne članove, koje dovodimo na unaprijed definirana mjesta u ILI matrici. Tako se ILI matrica može prikazati samo ILI vratima.

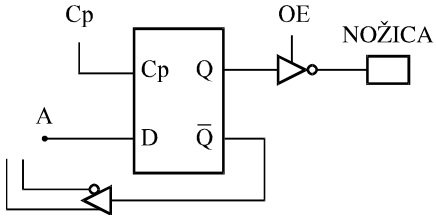
PAL komponente realiziraju se u bipolarnoj tehnologiji, s osiguračima kao programabilnim elementima. Primjenom CMOS tehnologije i EEPROM upravljačkih bitova, ostvarene su **GAL** (Generic Array Logic) komponente, kod

kojih je moguće brisanje i ponovno upisivanje I matrice. Često korištena komponenta je GAL 16V8, čiji je jedan od 8 identičnih dijelova prikazan na slici 5.6. Izlaz ILI vrata povezan je sa U/I nožicom preko programabilne "makro stanice".

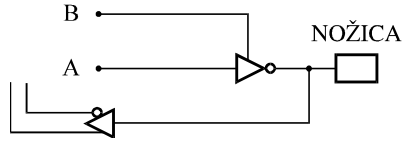


Slika 5.6. - 1/8 GAL komponente

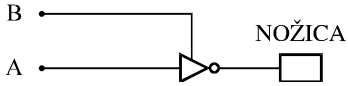
Svaka makro stanica može biti konfigurirana na četiri načina kao: sekvencijalni izlaz - preko D bistabila, slika 5.7a; kombinacioni ulaz/izlaz, slika 5.7b; kombinacioni izlaz slika 5.7c; ili kombinacioni ulaz, slika 5.7d.



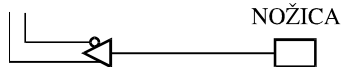
Slika 5.7a



Slika 5.7b



Slika 5.7c

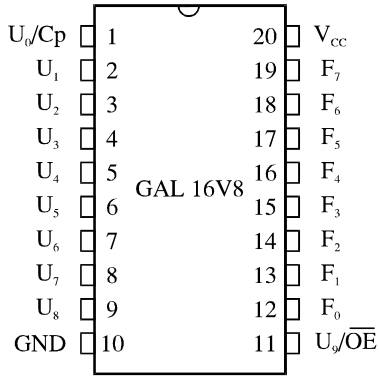


Slika 5.7d

GAL 16V8 ima 20 nožica, slika 5.8.: U_1-U_8 su ulazi, U_0/Cp koristi se kao ulaz ili taktni ulaz za bistabile, $U_9/\overline{OE}$ koristi se kao ulaz ili kontrolni ulaz za izlazna pojačala (Output Enable), a F_1-F_8 su izlazi makro stanica. Napajanje je +5V (V_{CC}), a zajednička je točka GND (zemlja).

Za određivanje sadržaja upravljačkih bita GALa koristit ćemo program CUPL (Universal Compiler for Programmable Logic). Program zahtijeva određeni oblik ulazne datoteke tipa PLD:

- zaglavlje s informacijama
- deklaracija varijabli ulaza i izlaza
- deklaracija unutrašnjih varijabli (nije obvezna)
- logičke jednadžbe varijabli izlaza



Slika 5.8. - GAL 16V8

U zaglavlju se nalaze podaci o nazivu i datumu izrade projekta, broju revizije, imenu izvođača, te korištenoj PAL ili GAL komponenti. Pri deklariranju ulaza i izlaza pridjeljujemo im imena varijabli i njihov polaritet. U dijelu deklaracije unutrašnjih varijabli navode se algebarski izrazi potrebni za preglednije pisanje konačnih funkcija. Funkcije koje realiziramo možemo zadati jednadžbama ili tablicama istine. Pri zadavanju nije potrebno paziti na minimalnost, jer CUPL sam vrši minimizaciju prema zadanom nivou.

Program CUPL na osnovu ulazne datoteke vrši minimizaciju zadanih funkcija i izračunava raspored upravljačkih bita za izabranu komponentu. On kreira, po izboru korisnika, nekoliko izlaznih datoteka. LST datoteka sadrži pregled svih pogrješki u originalnom kodu programa, DOC datoteka sadrži minimizirane jednadžbe funkcija, simboličku tablicu varijabli i kartu rasporeda upravljačkih bita, te skiciranu shemu korištenih nožica u GALu, ABS datoteka sadrži podatke koje koristi dio CUPL programa za simulaciju, a JED datoteka sadrži tablicu upravljačkih bita u tzv. JEDEC (Joint Electron Device Engineering Council) formatu. JED datoteku šaljemo na programator, koji vrši konkretno programiranje GAL komponente.

PRIMJER

Zadane su tri Booleove funkcije u PDNO.

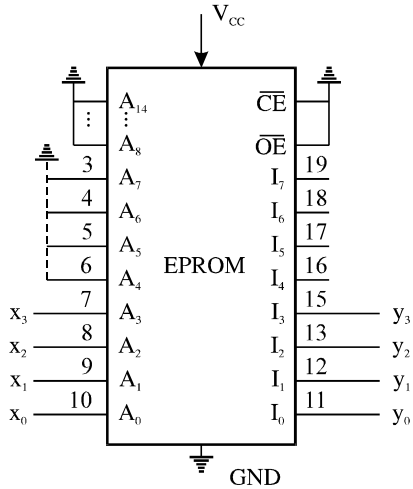
$$f_1 (x_1, x_2, x_3, x_4) = \vee (0, 3, 6, 9, 14, 15)$$

$$f_2 (x_1, x_2, x_3, x_4) = \vee (0, 2, 4, 12, 14)$$

$$f_3 (x_1, x_2, x_3, x_4) = \vee (0, 1, 2, 6, 11)$$

1. Zadane Booleove funkcije realizirati korištenjem EPROMa 27C256.

Na raspolaganju nam je model prema slici 5.9. Varijable funkcije x_3-x_0 dovodimo na adresne ulaze A_3-A_0. Nepotrebni adresni ulazi spojeni su na logičko "0" (zemlja). Izlazi EPROM-a I_3-I_0 su izlazi funkcija y_3-y_0. Kontrolni ulazi CS i OE su spojeni na "0", tako da je EPROM trajno selektiran.



Slika 5.9. - Laboratorijski model sa EPROMom

Funkcije realiziramo tako da retke tablice istine upisujemo u registre EPROMa. Pišemo tablicu istine funkcija i tablicu EPROMa u heksadecimalnom obliku.

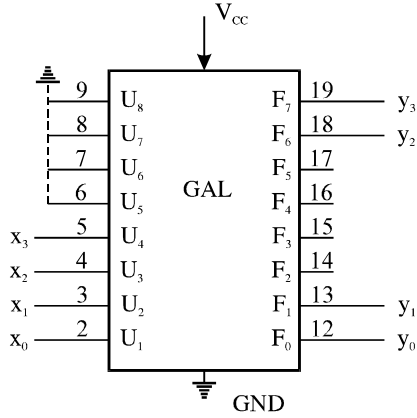
x_3	x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	1	1	1
0	0	0	1	1	0	0
0	0	1	0	1	1	0
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	0	0	0
0	1	1	0	1	0	1
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	0	0
1	0	1	1	1	0	0
1	1	0	0	0	1	0
1	1	0	1	0	0	0
1	1	1	0	0	1	1
1	1	1	1	0	0	1

A_3 - A_0	I_2 - I_0
00	07
01	04
02	06
03	01
04	02
05	00
06	05
07	00
08	00
09	01
0A	00
0B	04
0C	02
0D	00
0E	03
0F	01

Podatke iz desnog dijela tablice unesemo u memoriju programatora, te programiramo EPROM.

2. Zadane Boole-ove funkcije realizirati korištenjem GALa 16V8

Na raspolaganju nam je model prema slici 5.10. Varijable funkcije x_3-x_0 dovodimo na ulaze U_4-U_1. Nepotrebni ulazi spojeni su na logičko "0" (zemlja). Izlazi F_7, F_6, F_1 i F_0 GALa su izlazi funkcija y_3-y_0.



Slika 5.10. - Laboratorijski model sa GAL-om

a) Programiranje u programu CUPL korištenjem algebarskih izraza

Najprije kreiramo PLD datoteku. U zaglavlju navedemo tražene podatke. Prva riječ retka zaglavlja je ključna i ne može se mijenjati. Komentare, dio programa koji CUPL ignorira, započinjemo s /*, a završavamo s */. Pri deklaraciji ulaza i izlaza navodimo samo one koje stvarno i koristimo. Pri definiciji logičkih jednadžbi izlaza upisali smo PDNO, dakle sve potrebne minterme bez minimizacije. Svaki izraz u PLD datoteci (osim komentara) završava s ";". Disjunkcija se označava znakom #, a konjunkcija znakom &.

```
Name      primjer1;
Partno    xxxx;
Date      05/10/95;
Revision  01;
Designer  student XY;
Company   FESB;
Assembly  None;
Location  None;
Device    g16v8;

/*****************************************/
/* PRIMJER 1.                            */
/* REALIZACIJA BOOLE-OVIH FUNKCIJA       */
/*      PREKO PDNO                       */
/*****************************************/

/**  Ulazi  **/
pin [2..5]        = [x0..3];

/**  Izlazi  **/
pin [12..13]      = [y0..1];
pin 18            = y2;

/** Definicija izlaznih varijabli **/

y0 = (!x3 & !x2 & !x1 & !x0) #
     (!x3 & !x2 &  x1 &  x0) #
     (!x3 &  x2 &  x1 & !x0) #
     ( x3 & !x2 & !x1 &  x0) #
     ( x3 &  x2 &  x1 & !x0) #
     ( x3 &  x2 &  x1 &  x0);

y1 = (!x3 & !x2 & !x1 & !x0) #
     (!x3 & !x2 &  x1 & !x0) #
     (!x3 &  x2 & !x1 & !x0) #
     ( x3 &  x2 & !x1 & !x0) #
     ( x3 &  x2 &  x1 & !x0);

y2 = (!x3 & !x2 & !x1 & !x0) #
     (!x3 & !x2 & !x1 &  x0) #
     (!x3 & !x2 &  x1 & !x0) #
     (!x3 &  x2 &  x1 & !x0) #
     ( x3 & !x2 &  x1 &  x0);
```

b) Programiranje u CUPL-u pomoću tablice istine

Zaglavlje i deklaracija ulaza i izlaza su ostali nepromijenjeni. Pri realizaciji pomoću tablice istine, formiramo ulazno i izlazno polje podataka ključnom riječi FIELD. Ključnom riječi TABLE definira se struktura formata "ulazno polje" => "izlazno polje". Znak => je operator pridruživanja (određenoj vrijednosti ulaznog polja pridružuje se tražena vrijednost izlaznog polja). Vrijednosti kojim se zadaju ulazi i izlazi su u heksadecimalnom obliku, osim ako nije drugačije naznačeno.

```
Name      primjer2;
Partno    xxxx;
Date      05/10/95;
Revision  02;
Designer  student XY;
Company   FESB;
Assembly  None;
Location  None;
Device    g16v8;

/*****************************************/
/*    PRIMJER 2.
/* REALIZACIJA BOOLE-OVIH FUNKCIJA       */
/* PREKO TABLICA ISTINE                  */
/*****************************************/

/**  Ulazi  **/
pin [2..5]      = [x0..3];
/**  Izlazi  **/
pin [12..13]    = [y0..1];
pin 18          = y2;
/** Definicija izlaznih varijabli **/
FIELD ulaz  = [x3..0];
FIELD izlaz = [y2..0];
TABLE ulaz => izlaz {
 0 => 7; 1 => 4; 2 => 6; 3 => 1;
 4 => 2; 5 => 0; 6 => 5; 7 => 0;
 8 => 0; 9 => 1; A => 0; B => 4;
 C => 2; D => 0; E => 3; F => 1;}
```

Nakon što je CUPL kreirao sve potrebne datoteke, datoteka JED se može unijeti u programator koji na osnovu nje programira GAL.

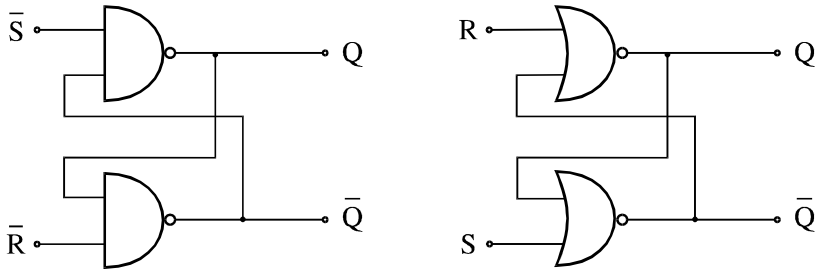
ZADATAK

1. Korištenjem programa CUPL, programatora i programabilnih komponenti EPROM 27C256 i GAL 16V8, realizirati na laboratorijskom modelu zadane Booleove funkcije.

VJEŽBA 6: MEMORIJSKI ELEMENTI

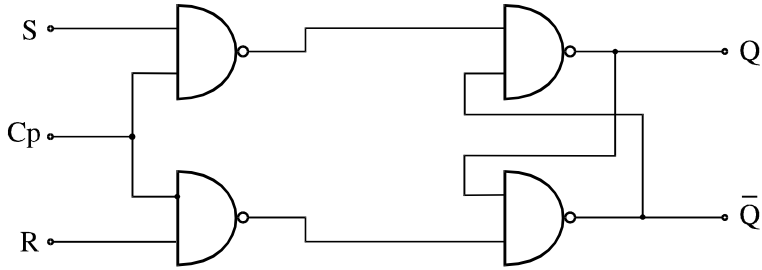
Osnovni memorijski element za pamćenje vrijednosti Booleove varijable ima dva stabilna unutrašnja stanja (0 ili 1) i nazivamo ga **bistabilom**. Promjena stanja memorijskog elementa ovisi o trenutnoj vrijednosti na njegovim ulazima q_1, q_2,..., q_n, kao i o njegovom unutrašnjem stanju. Unutrašnje stanje očitavamo na izlazima Q i $\overline{Q}$.

Bistabil je simetričan sklop i kod realizacije TTL tehnologijom sastoji se od dva tranzistora, koji sačinjavaju dvostruko pojačalo s jakom pozitivnom povratnom vezom. Veliko pojačanje u petlji povratne veze ne dozvoljava da sklop oscilira, nego dovodi do toga da jedan tranzistor vodi (u zasićenju) dok drugi ne vodi. Osnovna konstrukcija bistabila pomoću dvaju logičkih vrata (za NI i NILI) prikazana je na slici 6.1:



Slika 6.1. - RS bistabil realiziran logičkim vratima

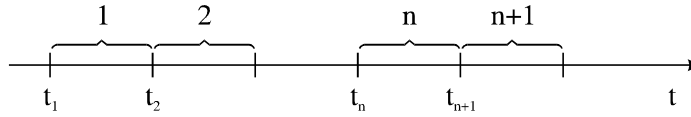
Bistabil ostaje u istom stanju sve dok se na ulazu ne pojavi aktivni impuls ili nivo koji ga prebacuje u drugo. Kako se ovi prijelazi ne bi dešavali u nedovoljno definiranim vremenskim trenucima, potrebno je odvojiti vrijeme promjene ulaznih nivoa od trenutka u kojem se vrši prebacivanje. To znači da ćemo ulazne signale propuštati do bistabila u skladu sa posebnim taktnim signalom Cp, kojim su vremenski trenuci precizno određeni. Shema takvog RS bistabila prikazana je na slici 6.2.



Slika 6.2. - RS bistabil sa taktnim ulazom

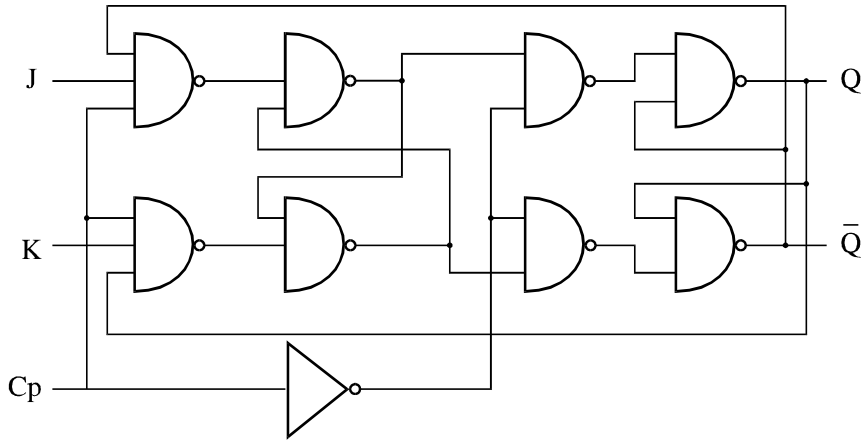
Ulazni signali djeluju samo dok je ulaz Cp u logičkoj jedinici. Da bismo što točnije odredili trenutak prijelaza, impuls koji dovodimo na Cp mora biti kratak. Raznim metodama može se postići da bistabil bude osjetljiv samo na brid impulsa (uzlazni ili silazni). Bistabili sa sinkronim ulazima najčešće raspolažu i sa asinkronim R i S ulazima koji služe za dovođenje bistabila u poznato početno stanje.

Taktni signal definira nam **diskretno vrijeme**, pa govorimo o vremenskom trenutku t_n u kojem dolazi do promjene, te vremenskom odsječku **n** koji započinje u trenutku t_n i traje koliko i period taktnog signala, slika 6.3.. Unutar odsječka ne nastupaju nikakve promjene. Kod ove definicije zanemarena su kašnjenja na samom bistabilu.



Slika 6.3. - Diskretna vremenska os

Jedna od struktura osjetljivih na brid je **master-slave** bistabil. On se sastoji od dva serijski vezana bistabila, od kojih je prvi osjetljiv na logičko 1 na taktnom ulazu, a drugi na logičko 0. Dok je Cp = 1, prvi bistabil (master) prihvaća promjene signala na ulaznim linijama i slijedi ih odgovarajućom promjenom stanja. Drugi bistabil (slave) ne reagira na ove promjene, već pamti prethodni sadržaj. Kod Cp = 0, situacija je obrnuta. Dakle, kod prijelaza Cp sa 1 u 0 drugi bistabil će prijeći u novo stanje ovisno o trenutnom stanju prvoga. Shema master-slave bistabila prikazana je na slici 6.4.



Slika 6.4. - Shema Mater-Slave JK bistabila

Bistabile zadajemo **tablicama prijelaza** i **funkcijama prijelaza**. Tablice prijelaza su slične tablicama istine, ali u sebi sadrže i vremenski odnos. Sa lijeve strane navodimo vrijednosti ulaznih varijabli i stanje bistabila u n-tom (sadašnjem) trenutku, a sa desne stanje bistabila nakon nastupa taktnog signala, tj. u n+1 trenutku. Postoji potpuni i skraćeni oblik tablice prijelaza, slika 6.5:

potpuni oblik

$(q_1,$	$q_2,$	...,	$q_n,$	$Q)^n$	Q^{n+1}
0	0	...	0	0	1
0	0	...	0	1	0

skraćeni oblik

$(q_1,$	$q_2,$	...,	$q_n)^n$	Q^{n+1}
0	0	...	0	$\overline{Q}^n$
0	0	...	1	Q^n

Slika 6.5. - Oblici tablice prijelaza bistabila

Funkcije prijelaza u sebi također sadrže vremenski odnos. One određuju stanje bistabila u n+1 odsječku, koje će bistabil zauzeti na osnovu vrijednosti ulaza i stanja u n-tom odsječku. Jednadžbe dobivamo iz tablice prijelaza i pišemo ih u dva oblika:

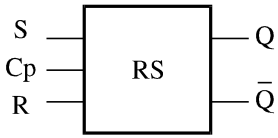
$$Q^{n+1} = f\left(Q, q_1, \ldots, q_n\right)^n \qquad \text{ili} \qquad Q^{n+1} = \left(G_1 Q \vee G_2 \overline{Q}\right)^n$$

Desni ili kanonski oblik jednadžbe prijelaza, gdje su G_1 i G_2 Booleove funkcije ulaznih varijabli, razdvaja funkciju prijelaza prema prethodnom stanju bistabila.

STANDARDNI BISTABILI

Neki od mogućih bistabila se standardno koriste i proizvode kao elementarni bistabili. To su RS, JK, D i T bistabili. Njihove funkcije prijelaza zovu se karakteristične jednadžbe. Realiziraju se kao sinkroni (imaju taktni ulaz), a često im se izvode i asinkroni R i S ulazi radi dovođenja u poznato početno stanje. Definicije standardnih bistabila su:

RS bistabil, X = izlaz neodređen, uvjet ispravnog rada je RS=0



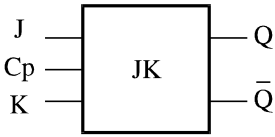
R	S	Q^{n+1}
0	0	Q^n
0	1	1
1	0	0
1	1	X

$$Q^{n+1} = \left(S \vee \overline{R} Q\right)^n$$

ili

$$Q^{n+1} = \left(S\overline{Q} \vee \overline{R} Q\right)^n$$

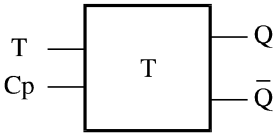
JK bistabil



J	K	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	$\overline{Q}^n$

$$Q^{n+1} = \left(\overline{K}Q \vee J\overline{Q}\right)^n$$

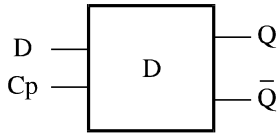
T bistabil



T	Q^{n+1}
0	Q^n
1	$\overline{Q}^n$

$$Q^{n+1} = \left(\overline{T}Q \vee T\overline{Q}\right)^n$$

D bistabil

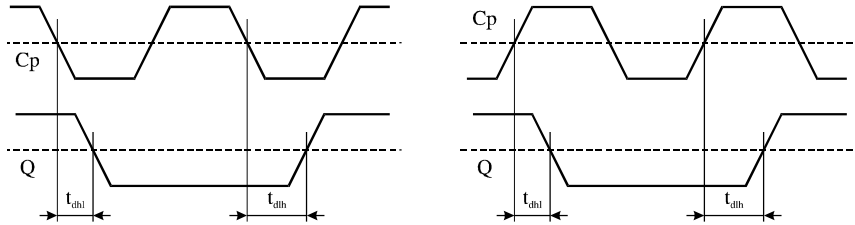


D	Q^{n+1}
0	0
1	1

$$Q^{n+1} = D^n$$

RS bistabil je elementarni bistabil, ali mu je prijelaz neodređen kada su oba ulaza (R i S) u 1. Uvjet ispravne primjene je RS=0. **JK** bistabil je univerzalni bistabil, koji realizira sve vrste preslikavanja. **T** bistabil mijenja stanje nastupom taktnog signala (uz T=1), pa ga koristimo za dijeljenje frekvencije sa 2 i realizaciju brojila. **D** bistabil se ponaša kao sklop za kašnjenje tj. vrijednost ulaza D se prenese na izlaz nakon nastupa taktnog signala. Kašnjenje je jednako periodu taktnog signala.

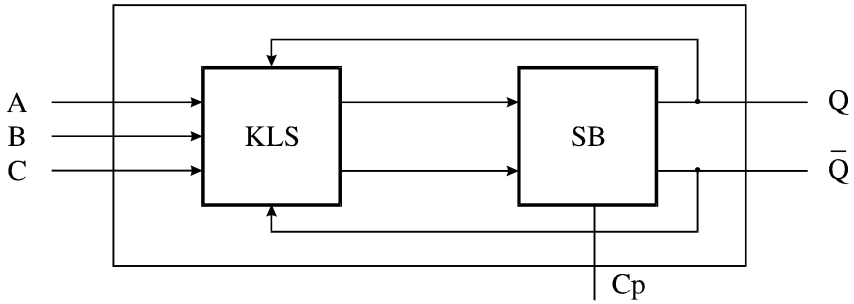
Brzina rada realnog bistabila ovisna je o kašnjenju između aktivnog dijela taktnog signala i promjene na izlazu bistabila, slika 6.6. Ovo kašnjenje ne treba miješati sa funkcijom bistabila kao memorijskog elementa, odnosno elementa za kašnjenje, koje se odvijaju pod kontrolom taktnog signala.



Slika 6.6. - Definicija kašnjenja bistabila

OPĆI BISTABILI

Opći bistabil (memorijski element) opisujemo aplikacijskom jednadžbom ili pripadnom tablicom prijelaza, a realiziramo ga pomoću nekog od standardnih bistabila. Koristimo model prema slici 6.7.



Slika 6.7. - Model općeg bistabila

Pošto su izlazi općeg zapravo izlazi standardnog bistabila (SB), moramo tako modificirati signale A, B, C da bi standardni bistabil vršio upravo neophodne prijelaze. To postižemo kombinacionom logičkom strukturom (KLS). Konstruiranje KLS provodimo primjenom jedne od tri raspoložive metode.

Metoda rekonstrukcije se zasniva na činjenici da su poznati prijelazi standardnog bistabila, te da je moguće rekonstruirati potrebne vrijednosti njegovih ulaza u potpunoj tablici prijelaza. Rekonstruirane vrijednosti dopisujemo sa desne strane tablice prijelaza. Tako u tablici sa lijeve strane imamo ulazne varijable i stanja općeg bistabila, a sa desne ulaze standardnog bistabila u trenutku n. Ove dijelove tablice prijelaza koristimo kao tablicu istine, na osnovu koje realiziramo KLS (vidi Vježbu 2). Metoda rekonstrukcije je pogodna za sve standardne bistabile, a naročito za RS i T bistabil. Kod RS bistabila omogućava neposrednu kontrolu uvjeta RS = 0.

Metoda izjednačavanja koristi mogućnost izjednačavanja aplikativne i karakteristične jednadžbe. Izjednačavanjem članova uz Q i onih uz $\overline{Q}$ dobivaju se ulazne jednadžbe koje definiraju KLS. Ova metoda pogodna je za JK bistabile. Za K minimiziramo $\overline{G}_1$.

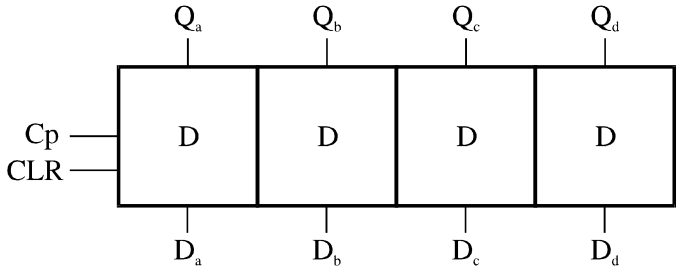
$$\left(\overline{K}Q \vee J\overline{Q}\right)^n = \left(G_1 Q \vee G_2 \overline{Q}\right)^n \Rightarrow K = \overline{G}_1 \text{ i } J = G_2$$

Treća je **metoda za D bistabile** pošto je $Q^{n+1} = D^n$. Tablica prijelaza postaje tablica istine za D, koju minimiziramo i realiziramo KLS.

Korištenjem bistabila i tehnologije srednjeg stupnja integracije realiziramo složene sekvencijalne strukture. To su registri, pomačni registri i brojila.

REGISTAR

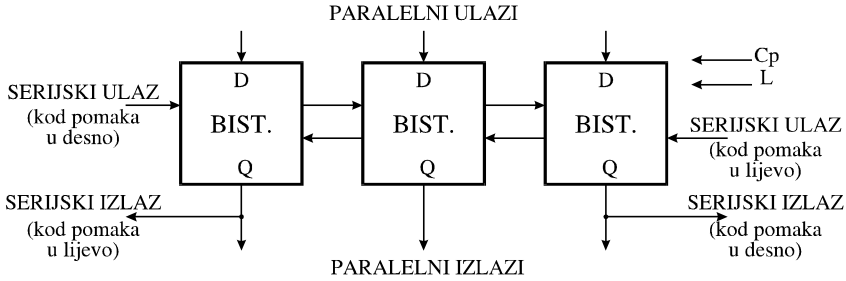
Registar je sklop koji nam služi za pamćenje kodnih riječi dužine m bitova. Za pamćenje svakog pojedinog bita koristimo vlastiti D bistabil, čiji je taktni ulaz spojen na zajednički taktni signal Cp. Ulazi u bistabile D su ulazi registra, a izlazi bistabila Q su izlazi registra, slika 6.8. Kako se pamćenje podatka vrši u istom trenutku, sinkrono sa taktnim signalom, govorimo o paralelnim ulazima i izlazima. Registri se izvode sa 4 i 8 bistabila i najčešće raspolažu sa asinkronim CLR (R) ulazom, kojim se dovode u početno stanje 0.



Slika 6.8. - Blok shema registra

POMAČNI REGISTAR

Pomačni registri (shift register) su posebno izvedeni registri kod kojih se podaci pomiču u lijevo ili u desno sinkrono sa dovedenim taktnim impulsima. Imamo više izvedbi ovih sklopova s obzirom na vrstu ulaza i izlaza. Svaki pomačni registar ima serijski ulaz i izlaz, a neki imaju i paralelne ulaze, paralelne izlaze ili oboje. Blok shema pomačnog registra prikazana je na slici 6.9.



Slika 6.9. - Blok shema pomačnog registra

Podatak koji dolazi na ulaz prvog bistabila upiše se u taj bistabil, a prethodni sadržaj prvoga prebacuje se u drugi, iz drugog u treći itd. Na izlazu posljednjeg bistabila pojavljuje se serija od n bitova prethodno pohranjenih u bistabilima registra. Dakle, sadržaj registra prolazi kroz sve memorijske elemente pomičući se

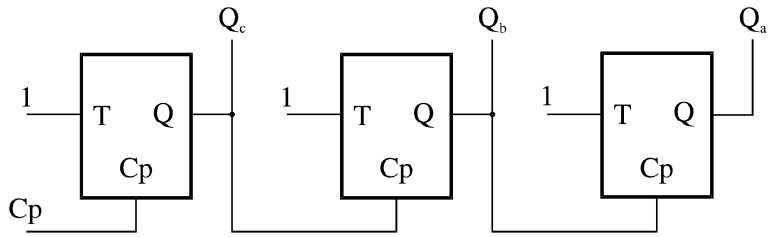
u lijevo ili u desno ovisno o konstrukciji registra. Osim po načinu izvedbe ulaza i izlaza, pomačne registre razlikujemo po smjeru pomaka (dvosmjerni ili jednosmjerni), te po broju bistabila (najčešće 4 i 8).

Kod pomačnog registra sa paralelnim ulazima podaci se upisuju u bistabile na aktivnom dijelu posebnog taktnog signala za paralelni upis L, dok je kod pomačnog registra sa paralelnim izlazima sadržaj bistabila stalno dostupan za očitavanje.

BROJILO

Brojilo je sklop tako povezanih m bistabila, da njihovo sljedeće stanje ovisi samo o prethodnom. Prijelaz nastaje u trenutku nastupa taktnog signala. Brojilo je jedna vrsta generatora sekvence, dakle automata bez vanjskih ulaza, kod kojega su stanja ujedno i izlazi. Zbog toga nije moguće ponavljanje simbola u izlaznoj sekvenci. Prolaskom kroz sva stanja brojilo na izlazima generira konačnu sekvencu kodnih kompleksija. Ukoliko je to prirodni binarni niz od 2^m kodnih kompleksija, govorimo o binarnom brojilu, a ako su to binarno kodirane dekadske znamenke (BCD kod), govorimo o dekadskom brojilu.

Bez obzira da li je realizirano kao sinkrono ili asinkrono, binarno brojilo funkcionalno možemo prikazati shemom na slici 6.10. Prijelaz iz stanja u stanje ovog sklopa je takav da na izlazima bistabila imamo upravo sekvencu binarnih brojeva.



Slika 6.10. - Blok shema binarnog brojila

PRIJENOS PODATAKA

Brojila, registri i pomačni registri, kao i multiplekseri i demultiplekseri nalaze primjenu u prijenosu podataka. Obzirom na prostorno-vremenske odnose, razlikujemo paralelni i serijski prijenos.

Kod paralelnog prijenosa kodne riječi dužine n bitova, koja predstavlja neko slovo, za svaki bit postoji zaseban signal i svi bitovi istog slova prenose se istovremeno. Dakle, za paralelni prijenos od n bitova potrebno je n paralelnih vodova. Prijenos više slova odvija se serijski u vremenskom nizu. Za sinkronizaciju prijenosa pojedinih slova koristi se obično poseban vod (n+1).

Za serijski prijenos dovoljan je jedan vod, a bitovi se nižu na vremenskoj osi jedan iza drugoga. Na predajnoj strani potrebno je informaciju koja postoji u paralelnom obliku prevesti u serijski, dakle sa prostorne prevesti u vremensku dimenziju. Na prijemnoj strani moramo provesti prostorno razdvajanje dobivenih signala. Da bi to bilo moguće, neophodan je proces sinkronizacije i to:

- sinkronizacija po bitu - radi ispravnog uzorkovanja primljenog signala
- sinkronizacija kodne riječi - da bi prijemnik mogao točno odrediti kojem kodnom mjestu pripada pojedini primljeni bit

Serijski prijenos koristimo za komunikaciju na većim udaljenostima, jer zahtijeva samo jedan vod, a time i manju cijenu kanala. Paralelni prijenos iziskuje veći broj vodova, ali i manje komplicirane i jeftinije sklopove. Primjenjujemo ga na manjim udaljenostima, npr. između računala i pisača.

Sinkronizacija po bitu ostvaruje se na dva načina, sinkronim i asinkronim prijenosom. Sinkroni signal uvijek je praćen taktnim impulsima koji se prenose posebnim kanalom ili se superponiraju osnovnoj informaciji (linijski kod). Asinkroni signal se prenosi bez taktnih impulsa. Ispred kodne riječi šalje se STARTNI, a nakon nje STOPNI bit. Tako se na osnovu poznate brzine prijenosa postiže sinkronizacija bita i kodne riječi.

Prevođenje iz prostorne u vremensku dimenziju nazivamo paralelno-serijskom konverzijom. Potrebne sklopove realiziramo korištenjem pomačnih registara ili multipleksera i demultipleksera s adresnim brojilima.

Pomačni registri su pogodni jer imaju mogućnost pamćenja podataka, a ujedno vrše i paralelno-serijsku konverziju i obrnuto.

Pri realizaciji konverzije pomoću multipleksera i demultipleksera moramo koristiti brojilo koje nam generira 2^m = n adresnih kompleksija. Mijenjajući adresne ulaze u prirodnom binarnom nizu, upravljamo izborom bita sa informacijskih ulaza multipleksera na izlaz i tako realiziramo paralelno-serijsku konverziju. Serijsko-paralelnu konverziju izvodimo pomoću demultipleksera i brojila. Ovdje se javlja problem pamćenja podataka, pa je potrebno koristiti registre.

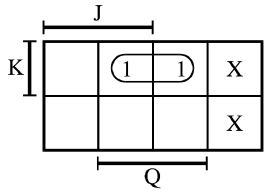
PRIMJER

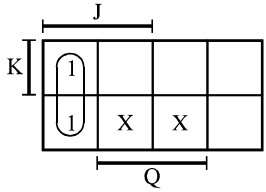
Korištenjem RS bistabila i NI vrata realizirati JK bistabil.

Koristimo metodu rekonstrukcije. Potpunoj tablici prijelaza JK bistabila dopisujemo rekonstruirane vrijednosti za R i S, te minimiziramo:

$(J$	K	$Q)^n$	Q^{n+1}	$(R$	$S)^n$
0	0	0	0	X	0
0	0	1	1	0	X
0	1	0	0	X	0
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	1	0	X
1	1	0	1	0	1
1	1	1	0	1	0

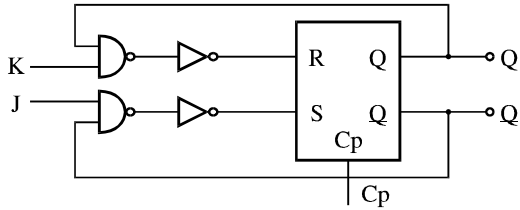
R^n: S^n:





$$R^n = K \cdot Q = \overline{\overline{K \cdot Q}} \qquad S^n = J \cdot \overline{Q} = \overline{\overline{J \cdot \overline{Q}}}$$

Nacrtamo shemu:



ZADATAK

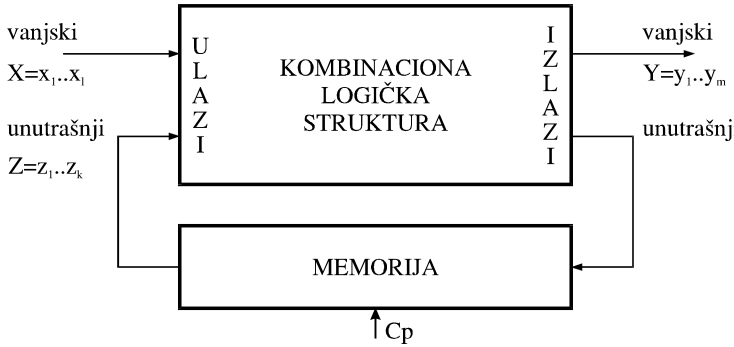
1. Na laboratorijskom modelu snimiti tablice prijelaza standardnih bistabila.
2. Za D i T bistabile izmjeriti vrijeme kašnjenja i najvišu taktnu frekvenciju.
3. Zadani opći bistabil realizirati pomoću zadanog standardnog bistabila. Provjeriti rad sklopa na modelu.
4. Snimiti funkciju rada registra, pomačnog registra i brojila.
5. Snimiti funkciju paralelno-serijske konverzije pomoću pomačnog registra i pomoću multipleksera i demultipleksera.

VJEŽBA 7: GENERATOR SEKVENCE

Generator sekvence je automat bez ulaza, koji na svom izlazu daje konačnu sekvencu izlaznih slova. S obzirom na način rada, razlikujemo generatore koji konačnu sekvencu ponavljaju beskonačan broj puta, te one koji se nakon prolaza kroz sva stanja zaustavljaju. Smatramo da je zadavanjem sekvence ujedno zadan i sam automat koji je generira, te odmah možemo nacrtati graf automata (Mooreov model jer nema ulaza) i minimalnu tablicu prijelaza. Ukoliko se slova sekvence ne ponavljaju, možemo stanja automata i izlazna slova kodirati identično. Takav generator sekvence nazivamo **brojilom**.

Strukturna sinteza automata vezuje se na rezultate apstraktne sinteze. Rezultat strukturne sinteze je sklop koji stvarno vrši zadano preslikavanje. U okviru ove vježbe obradit ćemo strukturnu sintezu za automat sa ulazima, a zatim ćemo prodiskutirati primjenu za generatore sekvenci.

Strukturnu sintezu automata vršimo korištenjem modela sa slike 7.1, koji se sastoji od memorije i kombinacione logičke strukture. Memoriju automata realiziramo pomoću k memorijskih elemenata, čijim su kompleksijama kodirana stanja automata. KLS raspolaže s vanjskim ulazima x, čijim su kompleksijama kodirani simboli ulaznog alfabeta U, te unutrašnjim ulazima z na kojima se pojavljuju stanja bistabila i čijim su kompleksijama kodirana stanja automata iz skupa S.



Slika 7.1. - Model realizacije automata

Kombinaciona logička struktura na osnovu vanjskih i unutrašnjih ulaza generira unutrašnje izlaze, a to su kontrolni signali kojima upravlja prijelazima pojedinih bistabila memorije. Na taj način realizira funkciju prijelaza automata δ. KLS na isti način realizira i funkciju izlaza automata λ (Vježba 8.).

U prvom koraku strukturne sinteze vršimo kodiranje automata. To znači da određujemo broj bistabila memorije i broj ulaznih i izlaznih varijabli, te vršimo

dodjeljivanje kodnih kompleksija stanjima i simbolima. Pri tome trebaju biti zadovoljeni uvjeti kodiranja:

skup U sa p članova u kodiramo kompleksijama od l varijabli x: $2^l \geq p$

skup I sa q članova i kodiramo kompleksijama od m varijabli y: $2^m \geq q$

skup S sa n članova s kodiramo kompleksijama od k bistabila z: $2^k \geq n$

Način kodiranja bitno utječe na veličinu automata. Kod kodiranja ulaza i izlaza moramo voditi računa o kompatibilnosti sa izvorištem i odredištem informacije i najčešće nemamo potpunu slobodu rada. Školski primjeri nisu povezani u sustav, što nam daje slobodu da ulaze i izlaze kodiramo proizvoljno.

Kodiranje stanja realiziramo dodjeljivanjem kompleksija stanja bistabila stanju automata. Ne postoje egzaktne metode kodiranja koje garantiraju minimalni automat, ali kao uspješnu strategiju možemo koristiti pravilo da stanjima među kojima imamo veće mogućnosti prijelaza dodjeljujemo susjedne kompleksije ili barem one sa što manjom distancom. U postupku kodiranja koristimo Veitchev dijagram, jer su susjedne kompleksije na njemu i fizički susjedne. Početno stanje u pravilu kodiramo kompleksijom 00...0.

Drugi korak strukturne sinteze je uvrštavanjem kodnih kompleksija u tablicu prijelaza i izlaza automata. Time dobijemo novu tablicu, čije dijelove možemo prepoznati kao tablice prijelaza za opće bistabile i tablice istine za izlazne varijable. Na osnovu tih tablica vršimo minimizaciju i realizaciju Booleovih funkcija (Vježba 2.) i općih bistabila (Vježba 6.)

U slučaju kad se radi o generatoru sekvence koji nema ulaze, postupak je identičan postupku za automat, osim što su tablice jednostavnije jer izostavljamo stupce za ulaze.

PRIMJER

Realiziraj pomoću JK bistabila i NI vrata generator sekvenci koji na svom izlazu daje slijed brojeva 0,1,3,5,7, ... Nema ponavljanja simbola, pa se radi o brojilu.

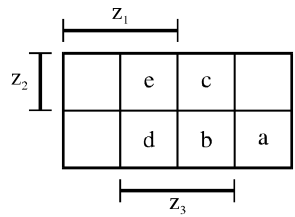
Ispisujemo tablicu prijelaza i izlaza generatora sekvence kao apstraktnog automata:

s^n	s^{n+1}	i^n
a	b	0
b	c	1
c	d	3
d	e	5
e	a	7

Broj bistabila određujemo relacijom:

$$2^k \geq 5 \Rightarrow k=3$$

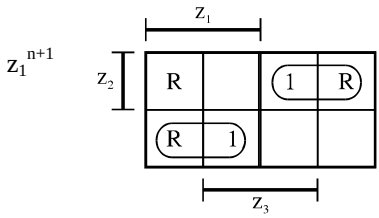
Stanja i izlaze kodiramo identično pomoću Veitchevog dijagrama:



Uvrstimo kodove stanja u tablicu prijelaza i izlaza:

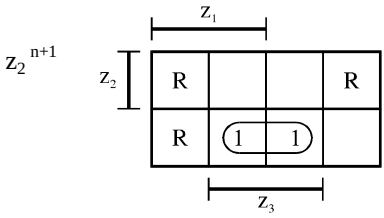
s^n	$(z_1$	z_2	$z_3)^n$	$(z_1$	z_2	$z_3)^{n+1}$	$(y_1$	y_2	$y_3)^n$
a	0	0	0	0	0	1	0	0	0
b	0	0	1	0	1	1	0	0	1
-	0	1	0	R	R	R	R	R	R
c	0	1	1	1	0	1	0	1	1
-	1	0	0	R	R	R	R	R	R
d	1	0	1	1	1	1	1	0	1
-	1	1	0	R	R	R	R	R	R
e	1	1	1	0	0	0	1	1	1

Minimizacija:



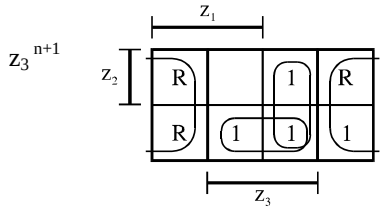
$$z_1^{n+1} = \left(\bar{z}_2 z_1 \vee z_2 \bar{z}_1\right)^n = \left(\overline{K}Q \vee J\overline{Q}\right)^n$$

$$K_1 = z_2 \qquad J_1 = z_2$$



$$z_2^{n+1} = \left(0 z_2 \vee z_3 \bar{z}_2\right)^n = \left(\overline{K}Q \vee J\overline{Q}\right)^n$$

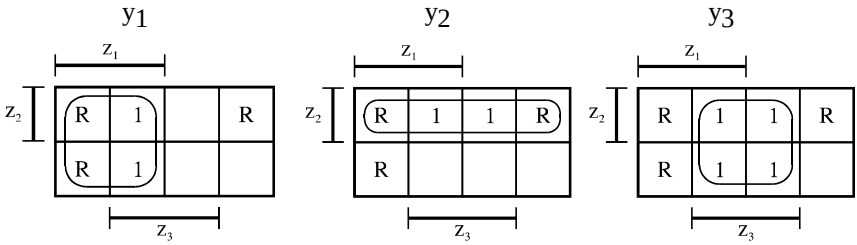
$$K_2 = 1 \qquad J_2 = z_3$$



$$z_3^{n+1} = \left(\left(\overline{z}_2 \vee \overline{z}_1\right)z_3 \vee 1\overline{z}_3\right)^n =$$

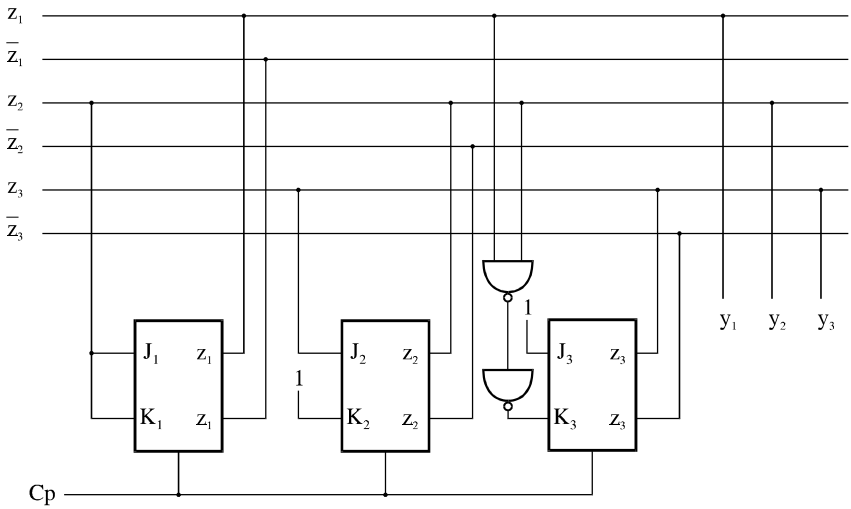
$$= \left(\left(\overline{z_1 z_2}\right)z_3 \vee 1\overline{z}_3\right)^n = \left(\overline{K}Q \vee J\overline{Q}\right)^n$$

$$K_3 = z_1 z_2 = \overline{\overline{z_1 z_2}} \qquad J_3 = 1$$



$y_1 = z_1$ $y_2 = z_2$ $y_3 = z_3$

Nacrtamo shemu:



ZADATAK

Izvršiti kodiranje stanja i realizirati zadani generator sekvence minimalnim brojem sklopova.

VJEŽBA 8: KONAČNI DIGITALNI DISKRETNI AUTOMATI

Automat je logička struktura čije ponašanje ovisi o trenutnoj vrijednosti ulaza (ako postoje) i o događajima iz prethodnih diskretnih vremenskih trenutaka. To znači da automat ima neku vrstu memorije, kojom pamti prethodne događaje. Stanje memorije određuje sekvencu proteklih događaja i zajedno sa trenutnim ulazima definira prijelaz u sljedeće stanje. Govorimo o automatima koji su:

- **digitalni**, ulazi i izlazi su binarne numeričke vrijednosti
- **diskretni**, rade u diskretnom vremenu pod kontrolom taktnog signala
- **konačni**, broj stanja memorije je konačan i time je automat ostvariv
- **determinirani**, funkcije prijelaza i izlaza su jednoznačne
- **potpuno ili nepotpuno specificirani**, sekvenca ulaznih događaja nema, odnosno ima unutrašnja ograničenja
- **sinkroni**, rad bistabila je sinkroniziran taktnim signalom.

Sinteze automata provodimo kroz dvije faze, apstraktnu i strukturnu sintezu. Predmet apstraktne sinteze je zadavanje, zapisivanje i minimizacija apstraktnog automata kao algebarskog modela. Strukturnom sintezom realiziramo konkretni automat kroz kodiranje stanja, ulaza i izlaza, te realizaciju strukture sklopova na osnovu modela realizacije.

FORMALNI OPIS AUTOMATA

Diskretni konačni digitalni automat zadan petorkom:

$$A = \langle U, S, I, \delta, \lambda \rangle$$

gdje je:

$U = (u_1, u_2, ..., u_p)$ kodiran s $X = (x_1, x_2, ..., x_l)$, $2^l \geq p$

$I = (i_1, i_2, ..., i_q)$ kodiran s $Y = (y_1, y_2, ..., y_m)$, $2^m \geq q$

$S = (s_0, s_1, ..., s_n)$ kodiran s $Z = (z_1, z_2, ..., z_k)$, $2^k \geq n$

Skup U nazivamo **ulaznim alfabetom** automata, skup I predstavlja **izlazni alfabet**, a skup S je **skup unutrašnjih stanja** automata; ukoliko je ovaj skup konačan govorimo o konačnom automatu. Skup X je skup ulaznih, a skup Y izlaznih varijabli. Skup Z je skup bistabila, odnosno njihovih izlaznih varijabli. Funkcije δ i λ opisuju prijelaz i izlaz automata.

Funkcija prijelaza δ je definirana kao:

$$s(t+1) = \delta(s(t), u(t)); \qquad s(t+1) = \delta(SxU, t); \qquad SxU \Rightarrow S$$

Ova relacija označava da je stanje u koje će prijeći automat određeno sadašnjim stanjem s(t) i slovom ulaznog alfabeta u(t) trenutno prisutnim na ulazu automata.

Kako su skupovi S i U konačni, funkcija δ je definirana nad kartezijevim produktom SxU, dakle obavlja preslikavanje iz skupa SxU u skup S. Pri tome je kartezijev produkt dvaju skupova skup koji sadrži sve uređene parove članova tih skupova. Ukoliko je funkcija prijelaza definirana jednoznačno govorimo o determiniranim automatima, te ako je definirana za sve kombinacije ulaza i izlaza (za sve članove skupa SxU) govorimo o potpuno specificiranim automatima.

Funkciju izlaza λ definiramo na dva načina, pa razlikujemo Mooreov i Mealyev model automata:

Mealy: $i(t) = \lambda(s(t), u(t));$ $i(t) = \lambda(SxU, t)$ $SXU \Rightarrow I$

Moore: $i(t) = \lambda(s(t));$ $i(t) = \lambda(S, t)$ $S \Rightarrow I$

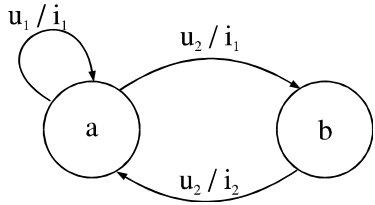
Kod Mealyeva modela izlazni simbol u trenutku n ovisi o trenutno prisutnom ulaznom simbolu, te trenutnom stanju. Kod Mooreova modela izlazni simbol je isključivo funkcija trenutnog stanja. Stoga Mealyev model reagira jedan period taktnog signala prije Mooreova, ali je nužno pojavu ulaznih simbola sinkronizirati sa taktnim signalom. Kod Mooreova modela ulaz djeluje posredno preko sljedećeg stanja, pa ovaj automat zapravo uzorkuje ulaznu sekvencu. Funkcija izlaza Mealyeva modela vrši preslikavanje iz SxU u I, a Mooreova iz S u I.

ZAPISIVANJE AUTOMATA

Automat zapisujemo tako da definiramo skupove U, I i S, te zapišemo funkcije δ i λ. U praksi je dovoljno zapisati funkcije, jer tako implicitno zadajemo i skupove. Formalne jezike kojima zapisujemo funkcije nazivamo standardnim jezicima. Automate zapisujemo grafički - pomoću orijentiranih grafova i tabelarno - pomoću tablica prijelaza i izlaza.

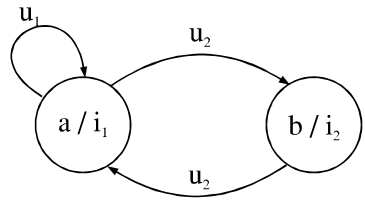
a) zapisivanje automata pomoću **orijentiranog grafa**

Grafove formiramo od krugova (čvorova) koji predstavljaju stanja automata povezanih usmjerenim dužinama koje označavaju prijelaze iz stanja u stanje. Kod Mealyeva modela krugovima su označena samo unutrašnja stanja, a dužine predstavljaju prijelaze i njima dodijeljene izlazne simbole, slika 8.1.



Slika 8.1. - Graf Mealyeva automata

Kod Mooreova modela krugovima su osim stanja označeni i izlazi kao funkcije stanja, a usmjerene dužine predstavljaju samo prijelaze, slika 8.2.



Slika 8.2. - Graf Mooreova automata

b) zapisivanje automata pomoću **tablice prijelaza i izlaza**

Automat zadajemo tablicom tako da funkciju prijelaza upisujemo u tablicu prijelaza, a funkciju izlaza u tablicu izlaza. Retke tablice dodijelimo stanjima s(t), a stupce ulaznim slovima u(t). Presjecište stupca i retka odgovara članu kartezijevog produkta SxU, te na to mjesto upisujemo s(t+1) kod tablice prijelaza, odnosno i(t) kod tablice izlaza. Uobičajeno je tablice prijelaza i izlaza pisati zajedno. Tablice prijelaza i izlaza za Mealyev model prikazane su na slici 8.3.

	$u_1(t)$	. . .	$u_l(t)$	$u_1(t)$	. . .	$u_l(t)$
$s_1(t)$	$s_i(t+1)$	. . .	$s_i(t+1)$	$i_i(t)$	. . .	$i_i(t)$
.	.		.	.		.
.	.		.	.		.
.	.		.	.		.
$s_k(t)$	$s_i(t+1)$	. . .	$s_i(t+1)$	$i_i(t)$	. . .	$i_i(t)$

Slika 8.3. - Tablica prijelaza i izlaza za Mealy-ev model

Za Mooreov model tablica izlaza ima samo jedan stupac jer izlazi ovise isključivo o s(t), slika 8.4.

	$u_1(t)$	. . .	$u_l(t)$	izlaz
$s_1(t)$	$s_i(t+1)$	. . .	$s_i(t+1)$	$i_i(t)$
.	.		.	.
.	.		.	.
.	.		.	.
$s_k(t)$	$s_i(t+1)$	. . .	$s_i(t+1)$	$i_i(t)$

Slika 8.4. - Tablica prijelaza i izlaza za Mooreov model

ZADAVANJE AUTOMATA

Kod početnog zadavanja automata, ideju o funkciji automata moramo zabilježiti nekim od standardnih jezika. Sekvencu slova ulaznog alfabeta koja se u vremenskom slijedu pojavljuju na ulazima automata, nazivamo ulazna riječ. Analogno, sekvenca izlaznih slova čini izlaznu riječ. Automat na osnovu ulazne riječi generira izlaznu.

Automat možemo promatrati kao **transformator sekvence**, i tada ga zadajemo pravilima o preslikavanju sekvenci. Češće promatramo preslikavanje ulazne sekvence na simbol, dakle automat izlaznim simbolom signalizira pojavu određene sekvence na ulazu. Nazivamo ga **akceptorom sekvence**. Konačno, možemo promatrati rad automata korak po korak, tj. simbol po simbol. Tablicu početno zadanog automata nazivamo primitivnom **tablicom**.

Kod promatranja automata kao akceptora sekvenci, koristimo jezik regularnih izraza. Regularnim izrazima opisujemo ulazne sekvence za koje će automat dati neki izlazni simbol.

Kod promatranja rada automata korak po korak, grafom opisujemo ponašanje automata na osnovu pojedinog ulaznog simbola. Time neposredno zadajemo funkciju prijelaza i izlaza. Najčešće koristimo graf u obliku **potpunog stabla**.

MINIMIZACIJA AUTOMATA

Postupak početnog zadavanja automata nije jednoznačan, jer prednost dajemo točnosti funkcije automata pred minimalnošću samog zapisa. Moguće je da ćemo isti automat zadati na različite načine, a da oni vrše istu funkciju. Takve različite automate, koji vrše istu funkciju, nazivamo **ekvivalentnima**.

Dva apstraktna automata sa istim ulaznim i izlaznim alfabetom su ekvivalentna ako za proizvoljnu sekvencu na ulazu daju istu sekvencu na izlazu. U skupu ekvivalentnih automata minimalan je onaj koji ima minimalan broj stanja. Ostali automati iz skupa, iako imaju veći broj stanja nemaju veće mogućnosti od minimalnog.

Višak stanja ne unosi poboljšanje kvaliteta i može se dokazati da je njegova pojava rezultat dupliciranja pojedinih stanja u postupku početnog zadavanja. Takva stanja zovemo ekvivalentnima. Dva stanja istog automata s_i i s_j su ekvivalentna, ako za proizvoljnu ulaznu sekvencu dobijemo istu izlaznu sekvencu, bez obzira da li je početno stanje automata bilo s_i ili s_j.

Iz definicije ekvivalentnosti dvaju stanja slijedi **nužan i dovoljan uvjet ekvivalencije**. Nužan uvjet kaže da su dva stanja potencijalno ekvivalentna ako su im reci u tablici izlaza identični. To znači da će i prvi simbol izlaznih sekvenci biti isti za oba stanja. Dovoljan uvjet kaže da su dva stanja ekvivalentna ako je zadovoljen nužan uvjet i ako su im reci u tablici prijelaza identični. To znači da će nakon prvog simbola ulazne sekvence automat prijeći u isto stanje, što garantira

identičnost nastavka izlazne sekvence. Ovaj uvjet je prestrog, jer ne vodi računa o eventualnoj ekvivalentnosti sljedećih stanja.

Minimizacija stanja automata se svodi na traženje ekvivalentnih stanja i zamjenu svakog skupa jednim stanjem. Za određivanje klasa skupova ekvivalentnih stanja koristimo minimizaciju primitivne tablice, Huffman-Mealyev algoritam (HM algoritam) i Paul-Ungerov algoritam.

a) minimizacija **primitivne tablice**

Kod ove metode polazimo od pretpostavke da su sva stanja međusobno neekvivalentna. Koristimo se činjenicom da su stanja sa zadovoljenim nužnim i dovoljnim uvjetom ekvivalencije, dakle sa istim recima u tablici prijelaza i izlaza, apriori ekvivalentna. Metoda ne garantira dobivanje minimalnog automata. Koraci minimizacije su:

1. U primitivnoj tablici uspoređujemo stanja redak po redak. Sva stanja koja imaju uz iste ulaze ista sljedeća stanja i izlaze, proglašavamo ekvivalentnima.
2. U svim skupovima međusobno ekvivalentnih stanja reduciramo sva suvišna stanja i zamjenjujemo ih jednim proizvoljno uzetim stanjem iz skupa.
3. Nakon ispuštanja suvišnih stanja, ponavljamo proceduru i time otklanjamo strogost dovoljnog uvjeta. Dobiveni automat može, ali ne mora biti minimalan.

b) minimizacija **Huffman-Mealyevim algoritmom**

Kod Huffman-Mealyeva algoritma polazimo od pretpostavke da su sva stanja sa zadovoljenim nužnim uvjetom ekvivalentna. Koristeći kriterij istih izlaza formiramo primarne klase (skupove) ekvivalentnosti za koje ispitujemo zatvorenost. Kao rezultat dobijemo skup zatvorenih klasa stanja, koje zamjenjujemo s po jednim stanjem. Dobiveni automat je minimalan. Koraci minimizacije su:

1. Stanja razvrstamo u primarne klase ekvivalentnosti prema nužnom uvjetu.
2. Ispitujemo zatvorenost klasa ekvivalentnosti. Klasa je zatvorena ako sva stanja klase imaju iste prijelaze u klase, ili ako klasa sadrži samo jedno stanje. U suprotnom, klasa je otvorena.
3. Otvorene klase razgrađujemo na nove klase s obzirom na razlike u prijelazima, te ponovo kontroliramo zatvorenost prema točki 2. Postupak ponavljamo sve dok ne dobijemo sve zatvorene klase.
4. Svaku zatvorenu klasu zamijenimo s po jednim stanjem minimalnog automata, te crtamo minimalni graf i tablicu.

c) minimizacija **Paul-Ungerovim algoritmom**

Paul-Ungerov (PU) algoritam polazi od definicije implikacije: Skup stanja Sp (podskup skupa S) impliciran je skupom Sr_i ako se u skupu Sr_i nalaze sva stanja u koja automat prelazi iz stanja iz skupa Sp uz ulazni simbol u_i. Podskupova Sr_i ima onoliko, koliko ima ulaznih simbola u.

Skup stanja Sp je skup ekvivalentnih stanja ako su, uz zadovoljen nužan uvjet ekvivalencije, svi skupovi Sr_i skupovi ekvivalentnih stanja.

To znači da ćemo formirati podskupove Sp od stanja sa zadovoljenim nužnim uvjetom, te ispitivati ekvivalentnost pripadnih skupova Sr_i. Ukoliko su oni ekvivalentni, nakon zamjene oznaka stanja jednim stanjem izjednačit će se reci tablice prijelaza stanja iz S_p, te će time biti zadovoljen dovoljan uvjet.

U praksi, koristimo metodu **tablice implikanata**, kod koje skupove Sp formiramo sistematski od po dva stanja iz skupa S, dakle ispitujemo sve parove stanja. Koraci minimizacije su:

1. Crtamo trokutastu matricu sa n-1 redaka i stupaca (n = broj stanja). Retke indeksiramo od 2 do n, a stupce od 1 do n-1. Na taj način svako mjesto u matrici odgovara jednom paru stanja bez ponavljanja (bez glavne dijagonale), odnosno jednom od mogućih skupova Sp.
2. U polja matrice upisujemo implikante. Znakom "X" označavamo polja za stanja koja su apriori neekvivalentna zbog različitih izlaza. Znakom "V" označavamo polja za stanja koja su evidentno ekvivalentna (isti reci u tablici prijelaza i izlaza). Za stanja čija ekvivalentnost ovisi o skupovima Sri, u polja matrice upisujemo te parove stanja (to su implikanti).
3. Vršimo inspekciju i tražimo kontradikcije. Ako je neki od upisanih implikanata u kontradikciji sa ekvivalentnošću tog para stanja ("X" u odgovarajućem polju), upisujemo "X" i u promatrano polje. Inspekciju ponavljamo s lijeva na desno sve dok otkrivamo nove kontradikcije.
4. Ispisujemo minimalni automat koji se sastoji od onih stanja koja nemaju ekvivalenta, te od po jednog iz svakog skupa ekvivalentnih.

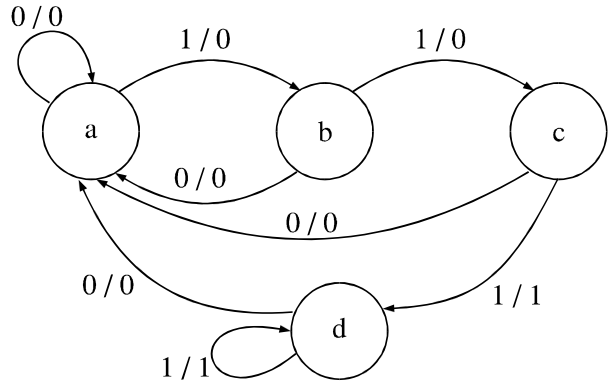
REALIZACIJA AUTOMATA

Na osnovu dobivene minimalne tablice, možemo provesti postupak strukturne sinteze automata. Kodiramo stanja, ulaze i izlaze te kodne kompleksije uvrštavamo u tablicu apstraktnog automata. Dobivena tablica je hibrid tablice istine za izlazne varijable, odnosno tablice prijelaza općih bistabila za memoriju. Detalji strukturne sinteze opisani su u Vježbi 7, a postupci minimizacije Booleovih funkcija u Vježbi 2. i postupci realizacije općih bistabila u Vježbi 6.

PRIMJER

Realizirati minimalni automat koji nakon tri uzastopne jedinice na ulazu daje jedinicu na izlazu. Koristiti Mealyev model. Na raspolaganju su JK bistabili i NI vrata.

Iz opisa funkcije automata pokušavamo nacrtati graf automata:



Automat se nalazi u stanju A uvijek nakon prijema 0, koja nije dio tražene sekvence. U stanju B smo nakon prijema prve, u stanju C nakon prijema druge i u stanju D nakon prijema treće jedinice. Iz grafa ispisujemo primitivnu tablicu prijelaza i izlaza automata.

s^n	s^{n+1}		i^n	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	a	c	0	0
c	a	d	0	1
d	a	d	0	1

Primjenom metode minimizacije primitivne tablice vidimo da su stanja c i d ekvivalentna jer imaju iste izlaze i prijelaze. Posljednji redak tablice brišemo, a za stanje c upisujemo umjesto prijelaza u d prijelaz u c (c je reprezentant ekvivalentnih stanja c i d). Iako je u ovom jednostavnom primjeru gotovo evidentno da je dobiveni automat minimalan, pokušajmo provesti minimizaciju HM metodom da bi se u to uvjerili.

Primjenom 1. koraka minimizacije formiramo dvije klase stanja s obzirom na izlaz. Jedna je karakterizirana izlazima (0,0), a druga izlazima (0,1) uz ulaze 0 i 1 respektivno. To su klase a_1(a,b) i b_1(c). Ispitujemo zatvorenost klasa:

a_1(0,0)	a	b
0	a_1	a_1
1	a_1	b_1

b_1(0,1)	c
0	a_1
1	b_1

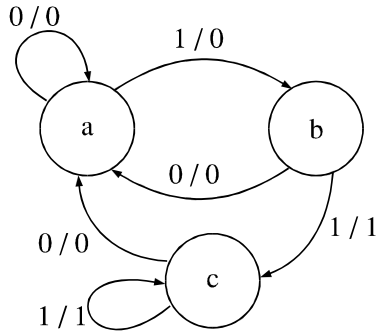
Klasa b_1 je zatvorena jer sadrži samo jedan član. Klasa a_1 nije zatvorena jer njeni članovi nemaju iste prijelaze u klase. Dok automat uz ulaz 0 prelazi u stanje a, koje je član klase a_1, za ulaz 1 iz stanja a prelazi u stanje b, koje je član klase a_1, a iz stanja b prelazi u stanje c koje je član klase b_1. Stoga klasu a_1 moramo razbiti na dvije klase pa dobijemo:

$a_2(0,0)$	a
0	a_2
1	b_2

$b_2(0,0)$	b
0	a_2
1	c_2

$c_2(0,1)$	c
0	a_2
1	c_2

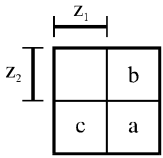
Klase a_2, b_2 i c_2 su zatvorene jer sadrže samo po jedan član, što znači da je automat već bio minimalan. Možemo nacrtati minimalni graf automata:



odnosno minimalnu tablicu prijelaza i izlaza:

s^n	s^{n+1}		i^n	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	a	c	0	0
c	a	c	0	1

Pristupamo kodiranju stanja automata pomoću Veitchevog dijagrama:



a = 00
b = 01
c = 10

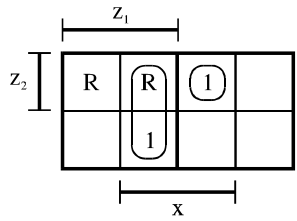
dok su ulazi i izlazi kodirani u okviru teksta zadatka.

Tablica prijelaza i izlaza nakon uvrštenja kodova izgleda:

$(z_1$	z_2	$x)^n$	$(z_1$	$z_2)^{n+1}$	y^n
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	R	R	R
1	1	1	R	R	R

Pristupamo realizaciji pojedinih bistabila i izlazne funkcije:

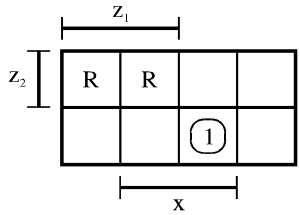
z_1^{n+1}



$$z_1^{n+1} = \left(xz_1 \vee z_2 x\overline{z}_1\right)^n = \left(\overline{K}Q \vee J\overline{Q}\right)^n$$

$$K_1 = \overline{x} \quad J_1 = z_2 x = \overline{\overline{z_2 x}}$$

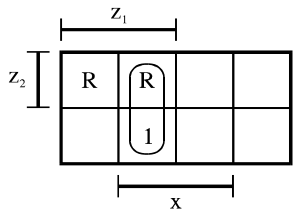
z_2^{n+1}



$$z_2^{n+1} = \left(0z_2 \vee \overline{z}_1 x\overline{z}_2\right)^n = \left(\overline{K}Q \vee J\overline{Q}\right)^n$$

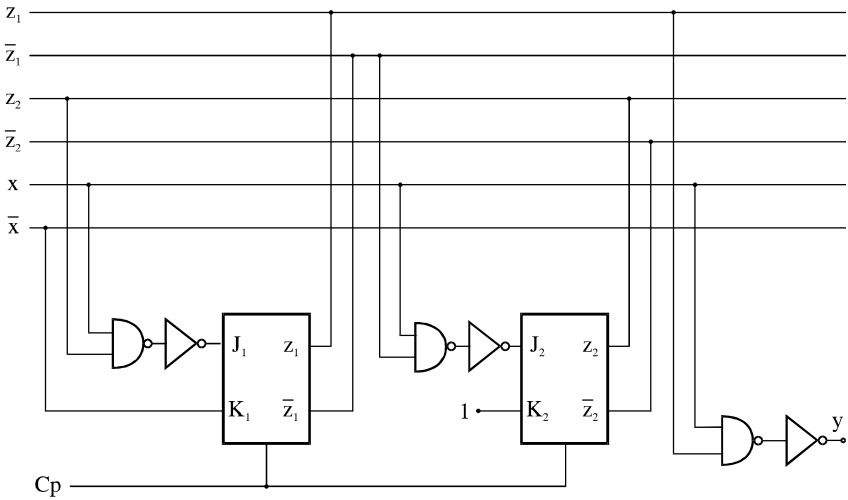
$$K_2 = 1 \quad J_2 = \overline{z}_1 x = \overline{\overline{z}_1 x}$$

y^n



$$y^n = z_1 x = \overline{\overline{z_1 x}}$$

Nacrtamo shemu:



ZADATAK

Minimizirati zadani automat i realizirati ga minimalnom logičkom strukturom. Provjeriti funkcionalnost sklopa na laboratorijskom modelu.

VJEŽBA 9: REALIZACIJA AUTOMATA POMOĆU PROGRAMABILNIH STRUKTURA

Specificiranjem minimalnog automata završava se apstraktna sinteza konačnog diskretnog digitalnog automata (Vježba 8). Strukturna sinteza provodi se u svrhu realizacije konkretnog automata logičkim sklopovima, a prema modelu realizacije. Korištenjem postupaka za minimizaciju Booleovih funkcija (Vježba 2) i općih bistabila (Vježba 6) vršimo sintezu sklopovlja automata integriranim krugovima malog stupnja integracije (Vježba 7).

Interesantno je pogledati mogućnost realizacije automata komponentama srednjeg i visokog stupnja integracije. Realizacija Booleovih funkcija primjenom multipleksera i demultipleksera, te programabilnih logičkih struktura, obrađena je u Vježbama 4 i 5.

Nakon provedenog kodiranja automata, uvrštavamo kodove u tablicu prijelaza i izlaza automata i dobijemo novu tablicu, čije smo dijelove prepoznali kao tablice prijelaza bistabila memorije, te tablice istine izlaznih varijabli. Ovdje nam je interesantno svojstvo svih tih pojedinačnih funkcija, da im je zajednička lijeva strana tablice. To znači da su one sve funkcije istih varijabli, odnosno stanja bistabila i vrijednosti ulaznih varijabli u sadašnjem trenutku.

Više funkcija istih varijabli efikasno realiziramo demultiplekserima s diodnom matricom, odnosno na toj osnovi izvedenim programabilnim strukturama (Vježba 5). Uočimo da je ukupni broj funkcija koje realizira KLS automata jednak broju izlaznih varijabli (vanjski izlazi) i broju kontrolnih ulaza u bistabile memorije (unutrašnji izlazi). Ukoliko koristimo bistabile sa jednim ulazom, konkretno D bistabile, funkcija ukupno ima M:

$$M = k + m$$

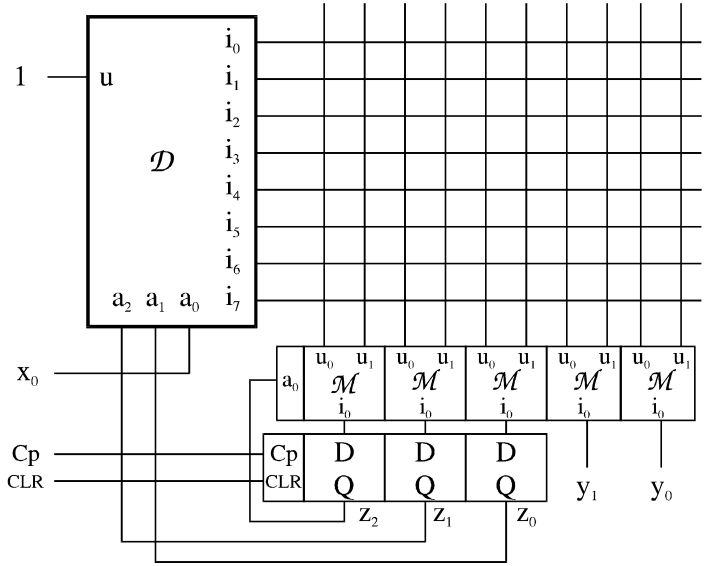
Memoriju automata realiziramo registrom D bistabila. Programabilna struktura sa samim demultiplekserom i ILI (diodnom) matricom nije u praksi pogodna zbog izduženog oblika matrice. Rješenje je u korištenju multipleksersko - demultiplekserske strukture sa M multipleksera (za svaku funkciju po jedan). Korištenje D bistabila je opravdano zbog minimalnog broja funkcija, odnosno multipleksera. Da bi matrica bila kvadratična, biramo veličinu demultipleksera i multipleksera tako da broj stupaca bude jednak broju redaka matrice. Pri tome broj ulaza u strukturu treba biti jednak zbroju ulaznih varijabli x i izlaza iz bistabila z. Vrijedi:

$$m_D + m_M = k + l \qquad 2^{m_D} \approx M \cdot 2^{m_M} \quad \Rightarrow$$

$$m_D \approx \frac{k + l + ld(M)}{2} \; ; \; m_M \approx \frac{k + l - ld(M)}{2}$$

Biramo odgovarajuće cjelobrojne vrijednosti. Varijable lijeve strane tablice prijelaza i izlaza automata (z pa x) dovodimo redom na ulaze multipleksera pa demultipleksera. Time osiguravamo da stupci matrice odgovaraju cjelovitim dijelovima stupaca tablice.

Multiplekserske - demultiplekserska struktura s D bistabilima kao struktura realizacije automata (MDD) prikazana je na slici 9.1.

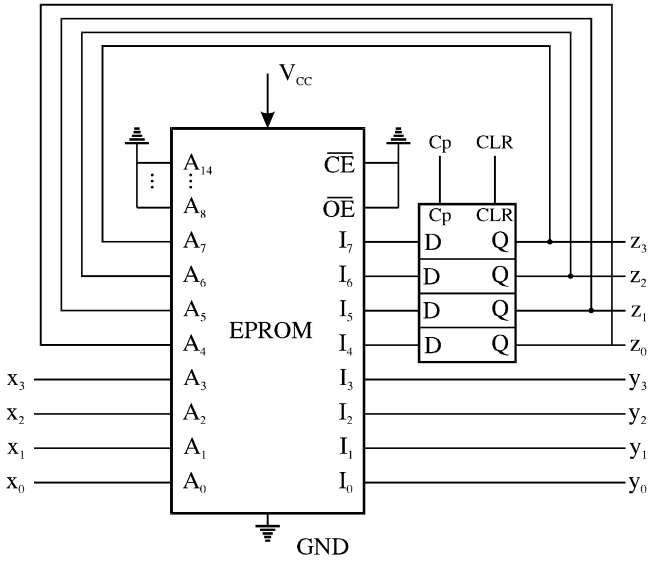


Slika 9.1. - MDD realizacija automata

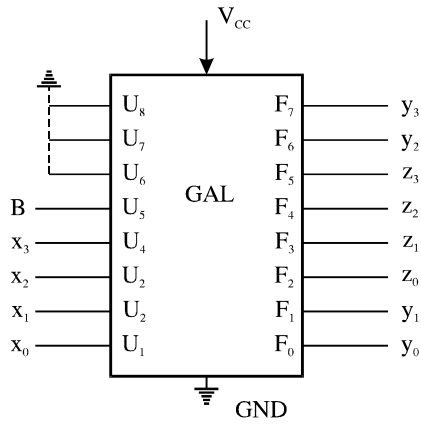
Umjesto multiplekserske - demultiplekserske strukture koristimo ROM komponente, u praksi EPROM zbog mogućnosti reprogramiranja (ako zadovoljava brzina rada). Uz EPROM nam je nužan i registar sa dovoljnim brojem D bistabila. Tablicu prijelaza i izlaza automata prevodimo u tablicu sadržaja EPROM-a heksadecimalnim zapisivanjem kodnih kompleksija. Na raspolaganju nam je laboratorijski model realizacije automata sa EPROM-om i D registrom, slika 9.2.

Od raspoloživih 15 adresnih ulaza EPROM-a, koristimo svega osam, i to 4 manje značajna za vanjske ulaze (varijable x), a 4 značajnija za unutrašnje ulaze (izlazi bistabila z). Takvu 8 bitnu adresu možemo predstaviti jednim dvoznamenkastim heksadecimalnim brojem, čija manje značajna znamenka odgovara kompleksiji varijabli, a značajnija kompleksiji stanja. Model koristi svih osam izlaza EPROM-a, tako da 4 manje značajna služe kao vanjski izlazi (varijable y), a 4 najznačajnija kao upravljački signali za D ulaze bistabila memorije. Izlaznu

kompleksiju također možemo izraziti dvoznamenkastim heksadecimalnim brojem, kod kojega niža znamenka definira izlaz, a viša prijelaz automata.



Slika 9.2. - Realizacija automata EPROM-om i registrom



Slika 9.3. - Realizacija automata GAL-om

Analiza optimalnih programabilnih struktura za realizaciju Booleovih funkcija dovela je do komponenti sa programabilnom I i fiksnom ILI matricom, PAL i GAL. Upoznali smo strukturu GALa koja sadrži programabilne makro stanice na izlazima, Vježba 5. Te se stanice mogu konfigurirati tako da promatrani izlaz bude sekvencijalan (povezan na izlaz D bistabila), ili kombinatoran (povezan na

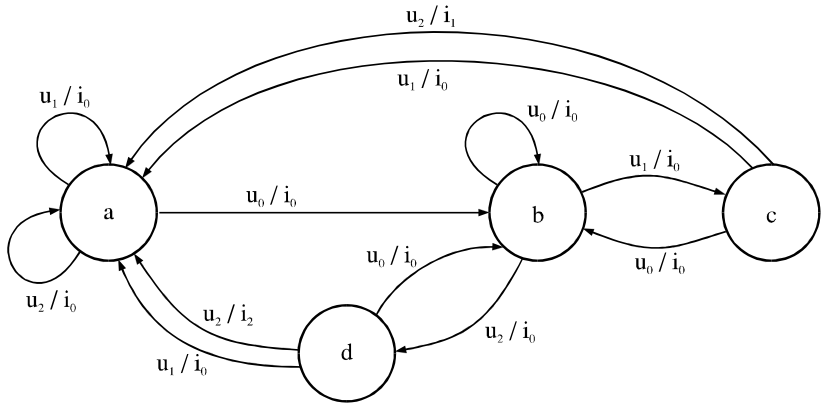
izlaz ILI vrata). Ovakvom komponentom neposredno realiziramo automat, koristeći interne povratne veze makro stanica. Na raspolaganju nam je laboratorijski model prema slici 9.3. D bistabili GALa ne raspolažu asinkronim CLR ulazom, pa je on nadomješten ulazom B (brisanje), čijim se aktiviranjem i nastupom taktnog signala stanje bistabila prevodi u stanje 00..0.

U okviru ove vježbe realizirat ćemo zadani automat korištenjem multipleksersko - demultiplekserske strukture s D bistabilima (MDD), EPROM-a s D bistabilima, te GALa.

PRIMJER

Realizirati Mealyev automat koji na izlazu daje slovo i_1, ako je na ulazu nastupila sekvenca $u_0u_1u_2$, a i_2 ako je na ulazu $u_0u_2u_2$. U svim ostalim slučajevima na izlazu je i_0. Članovi skupa ulaznih slova su u_0, u_1 i u_2.

Automat zadajemo grafom:



Iz grafa formiramo tablicu prijelaza i izlaza

s^n	s^{n+1}			i^n		
	u_0	u_1	u_2	u_0	u_1	u_2
a	b	a	a	i_0	i_0	i_0
b	b	c	d	i_0	i_0	i_0
c	b	a	a	i_0	i_0	i_1
d	b	a	a	i_0	i_0	i_2

Koristimo HM algoritam za minimizaciju

$a_1(i_0, i_0, i_0)$	a	b	$b_1(i_0, i_0, i_1)$	c	$c_1(i_0, i_0, i_2)$	d
u_0	a_1	a_1	u_0	a_1	u_0	a_1
u_1	a_1	b_1	u_1	a_1	u_1	a_1
u_2	a_1	c_1	u_2	a_1	u_2	a_1

Klasa a_1 je otvorena, pa dalje imamo:

$a_2(i_0, i_0, i_0)$	a
u_0	b_2
u_1	a_2
u_2	a_2

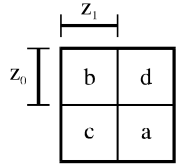
$b_2(i_0, i_0, i_0)$	b
u_0	b_2
u_1	c_2
u_2	d_2

$c_2(i_0,i_0, i_1)$	c
u_0	b_2
u_1	a_2
u_2	a_2

$d_2(i_0,i_0, i_2)$	d
u_0	b_2
u_1	a_2
u_2	a_2

Zadani automat je minimalan. Kodiramo stanja, ulaze i izlaze:

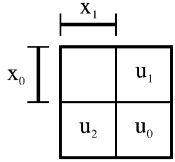
stanja:



a = 00
b = 11
c = 10
d = 01

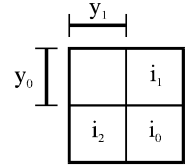
$2^k \geq 4 \quad \Rightarrow \quad k = 2$

ulazi:



$u_0 = 00$
$u_1 = 01$
$u_2 = 10$

izlazi:



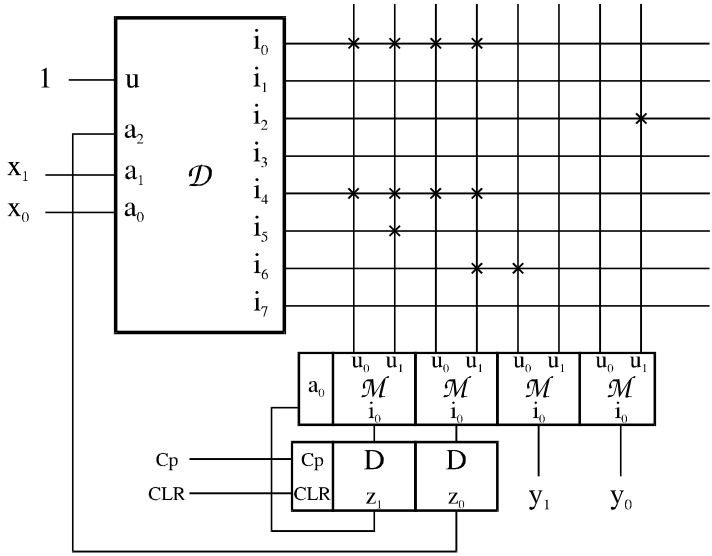
$i_0 = 00$
$i_1 = 01$
$i_2 = 10$

Upisujemo kodove u tablicu prijelaza i izlaza:

s^n	$(z_1$	z_0	x_1	$x_0)^n$	$(z_1$	$z_0)^{n+1}$	$(y_1$	$y_0)^n$
a	0	0	0	0	1	1	0	0
	0	0	0	1	0	0	0	0
	0	0	1	0	0	0	0	0
	0	0	1	1	R	R	R	R
d	0	1	0	0	1	1	0	0
	0	1	0	1	0	0	0	0
	0	1	1	0	0	0	1	0
	0	1	1	1	R	R	R	R
c	1	0	0	0	1	1	0	0
	1	0	0	1	0	0	0	0
	1	0	1	0	0	0	0	1
	1	0	1	1	R	R	R	R
b	1	1	0	0	1	1	0	0
	1	1	0	1	1	0	0	0
	1	1	1	0	0	1	0	0
	1	1	1	1	R	R	R	R

a) Automat realiziramo multipleksersko-demultiplekserskom strukturom i D registrom:

Realiziramo funkcije $D_1^n=Q_1^{n+1}$, $D_2^n=Q_2^{n+1}$, y_1 i y_0, ukupno 4 funkcije od 4 varijable. Optimalno je koristiti demultiplekser sa $m_D=3$ (8 redaka matrice) i $m_M=1$ (4*2=8 stupaca matrice), pa je matrica kvadratična. Ako za adresnu varijablu multipleksera izaberemo z_1, preostale varijable z_0, x_1 i x_0 dovodimo redom na ulaze demultipleksera. Preostale funkcije definirane su u stupcima dvaju dijelova tablice, odnosno stupci matrice odgovaraju cjelovitim dijelovima stupaca tablice.



b) Automat realiziramo EPROM-om i D registrom:

Iz tablice prijelaza i izlaza automata ispisujemo na osnovu modela po slici 9.2. tablicu sadržaja EPROM-a u heksadecimalnom obliku:

$A_7..A_0$	$I_7..I_0$
00	30
01	00
02	00
03	00
10	30
11	00
12	02
13	00

$A_7..A_0$	$I_7..I_0$
20	30
21	00
22	01
23	00
30	30
31	20
32	10
33	00

Podatke iz tablice unesemo u programator i programiramo EPROM 27C256.

c) Automat realiziramo GAL-om. Pišemo program u jeziku CUPL:

```
Name      autn;
Device    g16v8;
/*************************************/
/* REALIZACIJA AUTOMATA GAL-OM       */
/*************************************/

/**  Ulazi  **/
pin 1      = clk;       /* taktni signal     */
pin [2..3] = [x0..1];   /* ulazi             */
pin 6      = clr;       /* brisanje          */

/**  Izlazi  **/
pin [14..15] = [z0..1]; /* izlazi bistabila  */
pin [12..13] = [y0..1]; /* izlazi            */

/** Deklaracije **/
field stanje = [z1..0];
                  /* definicija polja stanje */
$define A 'b'00
$define B 'b'11
$define C 'b'10
$define D 'b'01

sa    = stanje:A; sb    = stanje:B;
sc    = stanje:C; sd    = stanje:D;

field ulaz = [clr,x1..0];
u0    = ulaz:0; u1    = ulaz:1; u2    = ulaz:2;
clear = ulaz:[4,5,6,7];

sequence stanje {
present A       if u0           next B;
                if u1           next A;
                if u2           next A;
                if clear        next A;
```

```
present B       if u0           next B;
                if u1           next C;
                if u2           next D;
                if clear        next A;
present C       if u0           next B;
                if u1           next A;
                if u2           next A;
                if clear        next A;
present D       if u0           next B;
                if u1           next A;
                if u2           next A;
                if clear        next A;}

y0 = sc & u2;
y1 = sd & u2;
```

JED datoteku prenesemo na programator i programiramo GAL 16V8.

ZADATAK

Zadani automat minimizirati i realizirati korištenjem multipleksersko-demultiplekserske strukture, EPROM/a s registrom i GALa. Funkcionalnost dizajniranih sklopova provjeriti na laboratorijskim modelima.

VJEŽBA 10: TURINGOV STROJ

Algoritmom postavljenog zadatka iz pripadne klase zadataka nazivamo u potpunosti definiran program akcija i način na koji treba te akcije primijeniti na trenutne vrijednosti ulaznih podataka, kako bismo u konačnom broju koraka došli do rješenja. Determiniranost procedure, jednoznačnost rezultata, različitost ulaza iz skupa dozvoljenih ulaznih vrijednosti i razloživost na elemente osnovne su karakteristike algoritma.

Algoritamskim jezikom nazivamo formalni jezik zapisa algoritma. Svaki algoritamski jezik karakteriziran je operatorima i logičkim uvjetima. Usprkos svoje univerzalnosti algoritamski jezici nisu uvijek optimalni za opis konkretnih klasa zadataka, pa su na njihovoj osnovi razvijeni jezici programiranja.

U praksi rješavamo dva osnovna problema: način sinteze automata i mogućnost njegove primjene. Zadati neki problem, tako da nas vodi na automatiziranu sintezu odnosno analizu, znači odrediti njegov algoritam. Složenost algoritamskih postupaka možemo ocijeniti na tri načina:

- na osnovu potrebne veličine određenih parametara automata.
- na osnovu složenosti njemu pripadnog programa.
- kroz definiciju složenosti klase algoritama, kojoj pripada promatrani algoritam.

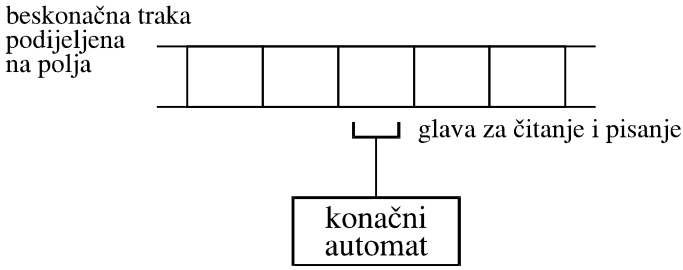
Na osnovu svojstava algoritma određujemo modele za njihovo izvođenje. Modeli su apstraktne strukture, "strojevi", koje raspolažu modulima za izvršenje operatora algoritma.

TURINGOV STROJ

Turingov stroj je algoritam, odnosno model koji raspolaže beskonačnom memorijom (traka), glavom za čitanje/upisivanje (R/W) i automatom koji izvršava program zapisan u algoritamskom jeziku. Turingov stroj je važan zbog toga što se vjeruje da skup proračuna koji on može izvesti uključuje sve proračune koje bilo koji automat može izvesti. Ovo se ne može dokazati jer izraz "bilo koji automat" uključuje i one koji još nisu uvedeni. Može se pokazati da Turingov stroj može izračunati sve parcijalno - rekurzivne funkcije, a matematičari smatraju da ova klasa uključuje sve funkcije koje se mogu izračunati. Ovaj model je koristan u teoriji automata kao polazište za dobivanje modela manjih mogućnosti uvođenjem određenih ograničenja.

MODEL

Model Turingovog stroja je konačni automat spojen sa glavom za čitanje i pisanje (R/W) koja se kreće po beskonačnoj traci, slika10.1.



Slika 10.1. - Model Turingovog stroja

Traka je podijeljena na polja koja sadrže po jedan simbol iz skupa T = (T_1, ..., T_m), gdje je T_1 = b prazan simbol.

R/W glava u jednom vremenskom trenutku može ispitati sadržaj jednog polja trake, te automat na osnovu očitanog simbola i unutarnjeg stanja izvršava zadanu akciju. Konačni skup stanja automata dan je sa S = (S_1, ..., S_n). Akcijom se može promijeniti stanje automata, simbol R/W glave i položaj glave prema traci.

Promatramo model koji ima jednu traku, beskonačnu na obje strane, s jednom R/W glavom koja ispituje jedno polje trake u jednom koraku akcije. Ispitivanjem simbola ispod glave i sadašnjeg stanja automat određuje:

1. Novi simbol koji će biti upisan u polje ispod glave, u isto ono polje iz kojeg je ulazni simbol očitan.
2. Kretanje glave u odnosu na traku, ili za jedno polje u lijevo (L), ili za jedno polje u desno (R).
3. Sljedeće stanje konačnog automata.
4. Treba li se zaustaviti ili ne.

Kako Turingov stroj ima konačan broj stanja i konačan alfabet trake, može se opisati tablicom ili grafom prijelaza. Svaki prijelaz mora specificirati novi simbol glave, smjer kretanja i novo stanje. Uvodimo dogovor da se automat zaustavlja kada otkrije nespecificiranu kombinaciju, ili komandom stop (S). Turingov automat možemo opisati petorkama:

$$\langle S_i, T_i, T_j, \begin{vmatrix} S \\ L \\ R \end{vmatrix}, S_j \rangle$$

Prva dva elementa daju sadašnje stanje i simbol glave. Ostali elementi opisuju akciju koja će se izvršiti. T_j je novi simbol, S_j je sljedeće stanje automata, dok L i R specificiraju lijevo odnosno desno kretanje, a H zaustavljanje. Nizom ovakvih

petorki opisujemo rad Turingovog automata. Uočimo da jedna petorka odgovara retku tablice prijelaza i izlaza automata kojem su ulazni i izlazni alfabet identični.

Turingov automat započinje s radom izvodeći transformaciju početnih podataka zapisanih na traku. Ako se automat zaustavi, rezultat transformacije je sadržaj trake u trenutku zaustavljanja. Ako se automat nikada ne zaustavi, rezultat nije definiran. Bilo koji automat koji je u skladu sa originalnom Turingovom formulacijom naziva se Q automatom.

OGRANIČENJA NAD PETORKAMA

Postavljanjem ograničenja na kombinacije akcija koje se mogu izvoditi u jednom pomaku, možemo kreirati druge modele automata. Može se pokazati da je, ukoliko se primjeni jedno od sljedećih ograničenja, dobivena klasa automata ekvivalentna klasi Q automata:

1. Dozvoljavamo automatu da ne izvede pomak R/W glave. Ovo označimo znakom N na mjestu u petorci na koje dolazi simbol za pomak. Ove automate nazivamo N automatima.
2. Automat mora promijeniti stanje i može izvršiti ili upis ili pomak, ali ne oboje. Automati ove klase nazivaju se F automatima, a opisuju se četvorkama.
3. Automat mora izvesti pomak i može ili promijeniti stanje ili izvršiti upisivanje, ali ne oboje.
4. Automat mora izvršiti upisivanje i može ili izvesti pomak ili promijeniti stanje, ali ne oboje.
5. Automat može izvesti samo jednu akciju: pomak, upis ili promjenu stanja.

OGRANIČENJE TRAKE

Može se pokazati da se modeli ekvivalentni Turingovom mogu definirati sa trakom koja je beskonačna samo u jednom smjeru ili sa proizvoljnim konačnim brojem beskonačnih traka.

Određeni problemi mogu se efikasnije rješavati korištenjem mreže više elementarnih Turingovih strojeva. Takva rješenja opisujemo dijagramima tijeka.

PRIMJER

Realiziraj Turingov stroj koji proizvoljan binarni broj množi sa 2. Pretpostavljamo da je binarni broj upisan na traku i da je sa lijeve i sa desne strane upisan prazan simbol, te da se u početnom trenutku glava za čitanje nalazi nad poljem sa najmanje značajnim bitom:

b01000101b

Turingov stroj očitava simbol iz trenutnog polja trake, množi ga sa 2 i upisuje rezultat u isto polje. Na osnovu izračunatog preteka, automat prelazi u stanje koje nosi informaciju o njemu. Pošto je pretek samo 0 ili 1 ($2 \cdot 0 = 0,\ 1 \cdot 2 = 10$), Turingov stroj će imati samo dva stanja. Stanje za koje je pretek jednak 0 je ujedno početno stanje. Rad takvog stroja definiran je petorkama:

$< S_{00}, 0, 0, L, S_{00} >$	$< S_{01}, 0, 1, L, S_{00} >$
$< S_{00}, 1, 0, L, S_{01} >$	$< S_{01}, 1, 1, L, S_{01} >$
$< S_{00}, b, b, S, S_{00} >$	$< S_{01}, b, 1, S, S_{00} >$

Turingov stroj simuliran je programom na laboratorijskom računalu. Petorke automata se upisuju u programsku datoteku nekim od editora. Stanja automata zadaju se dvoznamenkastim decimalnim brojevima (00 do 99), simboli trake su iz skupa svih alfanumeričkih znakova, a instrukcije o pomaku glave su: L=lijevo, R=desno i S=stop. Početni sadržaj trake zadaje se nakon aktiviranja simulacijskog programa. Moguć je upis svih alfanumeričkih znakova. Simulacijski program omogućava izvršenje algoritma korak po korak ili odjednom.

ZADATAK

Definirati Turingov stroj koji rješava postavljeni zadatak, te provjeriti njegovu funkcionalnost korištenjem programa za simulaciju.

VJEŽBA 11: UVOD U AVR MIKROKONTROLERE

Vježba 11 omogućava uvid u osnove programiranja AVR mikrokontrolera. Program koristi razvojni sustav STK200. Funkcija programa je da vrijednost sa ulaza (tipke) prenosi na izlaz (LED diode). Zadatak na vježbi je izmijeniti program tako da se promjene aktivna ulazna tipkala i LED diode. Slijedi ispis programa digital1.asm s komentarima:

```
;***********************************************************************
;*
;* Naslov / Title:                    DIGITAL1.ASM
;* Inačica / Version    :             1.00
;* Zadnja izmjena / Last updated:     06.01.2000.
;* Ciljni mikrokontroler / Target:    Svi AVR-ovi.
;*
;* Autor / Author:                    Tomislav Čanić, 091-502-92-71
;*                                    tomislav.canic@si.tel.hr
;*
;* OPIS / DESCRIPTION:
;* Vježba AVR1 za predmet "Digitalna i mikroprocesorska tehnika".
;*
;* Jednostavni program koji studentima predstavlja osnove programiranja
;* mikrokontrolera AVR familije, firme ATMEL (www.atmel.com).
;*
;* Programski zadatak:
;* -------------------
;* Stanje U/I priključaka PD7 i PD5 prenijeti na U/I priključke sklopa B
;* ne-invertirano (pritiskom tike uključi se LED, otpustom tipke isključi
;* se LED), a stanje U/I priključka PD2 invertirano (pritiskom tipke isključi
```

```
;* se LED, otpustom tipke uključi se LED). Negativna logika je ovdje skrivena.
;* Potrebno je obratiti pažnju na sljedeće:
;*  a) Interni pull-up otpornik - vezano uz tipke.
;*  b) Source current i Sink current - vezano uz spoj LED dioda.
;*
;*
;* Zadatak na vježbi:
;* -------------------
;* Izmijeniti program tako da izlazi budu PD6, PD5 i PD4,
;* s tim da invertirani budu PD6 i PD5, a ne-invertiran PD4.
;*
;*
;* Sadržaj programa:
;* ---------------
;* 1. Inicijalizacija indikatora sloga
;* 2. Inicijalizacija I/O sklopova mikrokontrolera
;* 3. Početno postavljanje registara
;* 4. Primjer poziva podprograma
;* 5. Očitavanje vrijednosti sa ulaza mikrokontrolera (tipke)
;* 6. Jednostavna logička operacija (EX-ILI)
;* 7. Postavljanje izlaza mikrokontrolera (uključi/isključi LED diode).
;*
;*****************************************************************************

;Komentar u asembleru počinje sa ";".

.NOLIST                     ;Isključi listanje sadržaja "include" datoteke.
.include "8515def.inc"      ;Definicije nožica i registara mikrokontrolera 8515.
.LIST                       ;Uključi listanje ostatka programa.
```

```
.def POMOCNI1 = r20          ;Kako ne možemo direktno pisati u neke registre
.def POMOCNI2 = r21          ;mikrokontrolera, neke od općih registara ćemo
                             ;upotrijebiti kao pomoćne.

     .ORG $0000              ;Početak programa na adresi nula.
                             ;$ označava heksadecimalni broj.

pocetak:                     ;Iniciranje indikatora sloga

     ldi  POMOCNI1, Low(RAMEND)   ;RAMEND je adresa zadnje RAM lokacije.
     out  SPL, POMOCNI1           ;Adrese se na slog spremaju prema
     ldi  POMOCNI1, High(RAMEND)  ;početku memorije, zato indikator sloga
     out  SPH, POMOCNI1           ;postavimo na kraj memorije.


;Postavljanje ulaza/izlaza. '1' znači izlaz, '0' znači ulaz.
;0b ispred broja označava binarni broj.
     ldi  POMOCNI1, 0b10100100    ;PB7, PB5 i PB2 su izlazi, ostali ulazi.
     out  DDRB, POMOCNI1          ;Na pločici su to LED diode.

     ldi  POMOCNI1, 0b00000000    ;Svi PD su ulazi.
     out  DDRD, POMOCNI1          ;Na pločici su to tipke.

     out  DDRA, POMOCNI1          ;Sklopove A i C koje ne koristimo
     out  DDRC, POMOCNI1          ;postavimo kao ulaze.

;Ovaj jednostavan program će samo proslijeđivati stanje na ulaznom
;sklopu D (tipke) na sklop B (LED diode), s tim da će LED na PB2
```

```
;svijetliti ako tipka na PD2 NIJE pritisnuta. Uočite da će svijetliti
;samo one LED diode koje su spojene na nožice definirane kao izlaz
;(a to su PB7, PB5 i PB2).


glavna_petlja:

     rcall    tipke              ;Primjer poziva podprograma.

     out  PORTB, POMOCNI1        ;Na izlaz upišemo rezultat

     rjmp glavna_petlja          ;Beskonačna petlja.


;---------------------------------------------------------------------------
;podprogram za očitavanje stanja tipki

tipke:
     in   POMOCNI1, PIND         ;I/O registar sadrži trenutno stanje tipki

     ldi  POMOCNI2, 0b00000100   ;Konstanta nam služi za negiranje PB2.
     eor  POMOCNI1, POMOCNI2     ;Negiranje pojedinog vršimo pomoću
                                 ;EKSKLUZIVNO-ILI operacije.
                                 ;AVR nema posebni registar za akumulator.
                                 ;Svi opći registri mogu poslužiti za to.

     ret                         ;povratak u glavni program.

;***************************************************************************
```

VJEŽBA 12: UPOTREBA VREMENSKOG SKLOPA

Vježba 12 omogućava uvid u rad vremenskog sklopa i organizaciju rada programa kao automata pogonjenog prekidnim zahtjevom vremenskog sklopa. Program detektira pritisak na tipku, filtrira istitravanje kontakata, te mijenja vrijednost pripadnog izlaza.

```
;*************************************************************************
;*
;* Naslov / Title:                    DIGITAL2.ASM
;* Inačica / Version    :             3.00
;* Zadnja izmjena / Last updated:     12.01.2000.
;* Ciljni mikrokontroler / Target:    AT90S8515
;* Autor / Author:                    Tomislav Čanić, 091-502-92-71
;*                                    tomislav.canic@si.tel.hr
;* OPIS / DESCRIPTION:
;* Vježba 2 za predmet "Digitalna i mikroprocesorska tehnika".
;*
;*   2. UPOTREBA VREMENSKOG SKLOPA
;*   ============================
;*
;* Ovaj program studente uvodi u način upotrebe ugrađenog vremenskog
;* sklopa i korištenja prekida.
;*
;* Programski zadatak:
;* -------------------
;* Tipke spojene na U/I sklop D čine tastaturu sa značenjem brojeva '0'-'7'.
;* Vremenski sklop 0 mikrokontrolera programirati tako da se svakih 10 ms
;* izvrši prekidni podprogram koji očitava stanje tipki. Ukoliko se unaprijed
```

```
;* zadani broj puta očita da je pritisnuta ista tipka, kod 0-7 se
;* upiše u registar TIPKA. Ako je više tipki pritisnuto, bit će
;* detektirana tipka s većim brojem. Ako nije pritisnuta nijedna
;* tipka, u registar TIPKA bit će upisano $FF.
;*
;* Uloga glavnog programa jest čekanje na pritisak tipke, te promjena
;* stanja pripadne LED diode, nakon čega se sve ponavlja.
;*
;* Inicijalizacijski dio programa mora uraditi sljedeće:
;* -postaviti SP na kraj RAM-a
;* -postaviti U/I sklop D kao ulaz, a U/I sklop B kao izlaz
;* -postaviti početno stanje U/I sklopa B (LED)
;* -nekorištene U/I sklopove A i C postaviti kao ulaze
;* -lokaciju TIPKA postaviti početno na $FF
;* -pripremiti vremenski sklop 0 tako da se prekid desi svakih 10ms
;*
;* Zadatak na vježbi:
;* ------------------
;* Promijeniti glavni program tako da se inicijalno upale gornje 4 LED
;* umjesto sadašnjih donjih 4 LED diode.
;*
;* Promijeniti inicijalizacijski dio programa tako da se prekidni podprogram
;* izvrši svakih 15 ms umjesto svakih 10 ms. Pritisak na tipku detektirati
;* nakon najmanje 1000 ms (1 sekundu).
;*
;* Sadržaj vježbe:
;* ---------------
;* 1. Inicijalizacija indikatora sloga
;* 2. Inicijalizacija I/O U/I sklopova mikrokontrolera
```

```
;* 3. Početno postavljanje registara i LED dioda
;* 4. Inicijalizacija dodatne periferije - 8 bitni vremenski sklop 0
;* 5. Očitavanje vrijednosti ulaza mikrokontrolera iz prekidnog podprograma
;* 6. Postavljanje izlaza mikrokontrolera koristeći negativnu logiku
;*    (radi LED dioda spojenih prema Vdd).
;*
;*************************************************************************

.NOLIST                     ;Isključi listanje sadržaja inc datoteke.
.include "8515def.inc"      ;Definicije nožica i registara mikrokontrolera.
.LIST                       ;Uključi listanje ostatka programa.

.def POMOCNI = R16          ;Kako ne možemo direktno pisati u neke registre
.def RFF = R17              ;mikrokontrolera, neke od općih registara ćemo
                            ;upotrijebiti kao pomoćne. RFF će imati
                            ;vrijednost $FF koju ćemo često koristiti.

.def Brojac_Prekida =R18    ;R18 će služiti za brojanje prekida. Ako je neka
                            ;tipka pritisnuta, uvećavamo R18. Ako R18 dosegne
                            ;unaprijed zadanu vrijednost, detektiramo tipku.

.def TIPKA = R19            ;Registar R19 će sadržavati kod tipke.

.def Brojac_tipke = R20     ;Za petlju za određivanje koja je od osam
                            ;tipki pritisnuta.
.def PreCount = R21         ;Obnova Brojača 0 za vrijeme 10 ms.

.def STANJE = R22           ;Ovo koristi podprogram za provjeru tipaka
                            ;za trenutno stanje ulaza.
```

```
.def Zadnja = R23           ;Privremeno pamti zadnju tipku.

.def SaveSREG = R24         ;Spremište za SREG za vrijeme prekida.

.def MASKA =R25             ;Za toggle funkciju.

.equ Zadano = 5             ;Broj prekida da bi tipka bila detektirana.

.equ ReloadValue = 100      ;Vrijednost za TCNT0 registar. Objašnjeno
                            ;kod opisa Brojača 0 dalje u programu.
                            ;PreCount = 256 - (vrijeme u mikro-sek. / 64).

;***********************************************************************

     .ORG $0000                 ;Početak programa na adresi nula.
     rjmp pocetak

     .ORG OVF0addr              ;AT90S8515 data-sheet, str. 15: na adresi
     rjmp TIM0_OVF              ;7 mora se nalaziti Timer0 prekidni podprogram.
                                ;Prije izvršenja prekidnog podprograma, procesor
                                ;zabrani daljnje prekide.
                                ;Naredba "reti" na kraju ih opet omogući.

;***********************************************************************

     .ORG $0010

TIM0_OVF:
```

```
     in   SaveSREG,SREG          ;Sačuvaj Statusni Registar.
     out  TCNT0, PreCount        ;Obnova Brojača 0 (reload).
     in   STANJE, PIND           ;Stanje tipaka u registar STANJE.
     cp   STANJE, RFF            ;$FF ako nije pritisnuta nijedna tipka.
                                 ;Tipke su naime povezane prema masi, pa
                                 ;pritisnuta tipka daje logičko '0'.
                                 ;To je negativna logika na ulazu.
     breq nije_pritisnuto        ;Branch If Equal - skoči ako je jednako.
                                 ;Usporedili smo ulaz sa kodnom riječi koja
                                 ;znači da nijedna tipka nije pritisnuta.
                                 ;Tada se jednostavno vratimo u glavnu petlju.

     cp   Brojac_Prekida,RFF     ;Ako je Brojac_Prekida = $FF, to je
                                 ;znak da je tipka već bila detektirana
     breq izlaz                  ;i treba čekati otpust.

     ldi  Brojac_tipke,7;Počinjemo od tipke '7'.
pritisnuto:
     rol  STANJE                 ;Rotiramo STANJE ulijevo kroz bit prijenosa C.
     brcc eto_je                 ;Branch if C Cleared - skoči ako je C = '0',
                                 ;ako bit broj Brojac_tipke = '0', upravo ta
                                 ;tipka je pritisnuta i idemo dalje,
     dec  Brojac_tipke           ;inače prelazimo na tipku nižeg rednog broja
     rjmp pritisnuto             ;i tako do tipke '0'.

eto_je:
     cpi  Brojac_Prekida, Zadano ;Je li tipka dovoljno dugo pritisnuta?
     breq dovoljno_dugo
     cp   Brojac_tipke, Zadnja   ;Ako je ista tipka, uvečaj Broj_Prekida
```

```
     breq ista_tipka
     clr  Brojac_Prekida          ;inače ga postavi na nulu
     mov  Zadnja, Brojac_tipke    ;i postavi registar Zadnja.
     rjmp  izlaz

dovoljno_dugo:
     mov  TIPKA, Brojac_tipke     ;Svi uvjeti su ispunjeni.
     ser  Brojac_Prekida          ;Znak prekidnoj rutini da mora čekati
                                  ;otpust tipke.
     rjmp izlaz

ista_tipka:
     inc  Brojac_Prekida          ;Ista tipka, uvećavamo do Zadano.
     rjmp izlaz

nije_pritisnuto:
     ser  TIPKA                   ;Nije ništa pritisnuto.
     clr  Brojac_Prekida          ;Detekcija tipke ispočetka.
izlaz:
     out  SREG,SaveSREG           ;Obnovi Statusni Registar.
     reti                         ;Iz servisne rutine izlazi se ovom naredbom,
                                  ;koja ponovo omogućuje prekide.

;*********************************************************************

pocetak:

;Postavljanje indikatora sloga
```

```
     ldi  POMOCNI, Low(RAMEND)    ;RAMEND je adresa zadnje RAM lokacije.
     out  SPL, POMOCNI            ;Adrese se na slog spremaju prema
     ldi  POMOCNI, High(RAMEND)   ;početku memorije, zato indikator sloga
     out  SPH, POMOCNI            ;postavimo na kraj memorije.


;Postavljanje ulaza/izlaza. '1' znači izlaz, '0' znači ulaz.

     ser  POMOCNI                 ;Set Register, svi bitovi na '1'.
     out  DDRB, POMOCNI           ;U/I sklop B sve izlazi (LED diode).

     ldi  POMOCNI, 0b11110000     ;Početno stanje LED dioda, upaljene
     out  PORTB, POMOCNI          ;su diode na PB3, PB2, PB1 i PB0.

     clr  POMOCNI                 ;CLear Register, svi bitovi na '0'.
     out  DDRD, POMOCNI           ;U/I sklop D sve ulazi (tipke).
     out  DDRA, POMOCNI           ; U/I sklopove A i C koje ne koristimo
     out  DDRC, POMOCNI           ;postavimo kao ulaze.

     ser  RFF                     ;RFF = $FF
     ser  TIPKA                   ;TIPKA = $FF
     clr  Brojac_Prekida          ;Tipka još nije pritisnuta.
     clr  MASKA                   ;Početmo stanje maske za toggle.

;AVR ima ugrađen 8-bitni vremenski sklop. Ovdje ćemo ga koristiti kao Counter,
;tj. Brojač 0. Kako je 8-bitni, možemo brojati od 0 do 255 ($FF).
;Postavljanjem odgovarajućih upravljačkih registara AVR mikrokontrolera
;možemo zadati da Brojač 0 broji frekvenciju kristala na koji je spojen
;mikrokontroler (za STK-200 to iznosi 4 MHz), ili frekvenciju kristala
```

```
;podijeljenu s 8, 64, 256 ili 1024, ili impulse s odgovarajuće nožice
;mikrokontrolera. Za naše potrebe, frekvenciju kristala podijelit ćemo s 256
;i dovesti je na ulaz Brojača 0.
;Dijeljenje obavlja predbrojač koji predstavlja pred-brojilo koje može brojati
;do već spomenutih 8, 64, 256 ili 1024. Da bi predbrojač dijelio s 256, u
;registar TCCR0 moramo upisati vrijednost 4 (AT90S8515 data-sheet, str. 24):

     ldi  POMOCNI, $04
     out  TCCR0, POMOCNI          ; predbrojač dijeli s 256.

;To znači da će na ulazu Brojača 0 biti frekvencija 4 MHz / 256,
;tj. Brojač 0 će se uvećati svako 256 / 4 MHZ = 64 mikro-sekunde.
;Kad se Brojač 0 uveća do 255 ($FF), iduće uvećanje će ga postaviti
;ponovo na nulu (slično kao kod automobilskog brojača koji se okrene
;na nulu nakon 99999 kilometara).
;Servisna rutina se pozove za svaku tu promjenu brojača sa 255 na 0.
;Ako u Brojač 0 (TCNT0 registar) početno upišemo 255, do preteka na
;nulu će doći nakon 64 mikro-sekunde od dozvole rada Brojača 0.
;Ako upišemo 254, vrijeme će iznositi 2 x 64 = 128 mikro-sekundi.
;Dakle, vrijeme iznosi (256 - Početna Vrijednost) x 64 mikro-sekunde.
;Rješavanjem 10 ms = 10 000 mikro-sek. = (256 - Poč. Vr.) x 64 mikro-sek.
;dobivamo da je Početna Vrijednost = 256 - (10 000 / 64) = 99.75, što
;zaokružujemo na najbližu vrijednost, a to je 100.
;Stvarno vrijeme iznosi (256 - 100) x 64 = 9.984 mikro-sekundi.
;Greška iznosi 16 mikro-sekundi, ali moramo oduzeti nekoliko mikro-sekundi
;dok mikrokontroler stvarno započne izvođenje servisne rutine unutar
;koje moramo ponovo napuniti registar TCNT0 vrijednošću 100.
;Stvarna greška je tako oko 14-15 mikro-sekundi.
;Kad bismo morali realizirati sat realnog vremena, to bi bilo neprihvatljivo.
```

```
;Za realizaciju tastature, greška je nebitna i zanemarujemo je.

     ldi  PreCount, ReloadValue       ;U registru PreCount ćemo držati
                                      ;reload vrijednost.
     out  TCNT0, PreCount             ;Prvi upis u Brojač 0.

     ldi  POMOCNI, 1 << TOIE0         ;Dozvola prekida za Brojač 0.
     out  TIMSK, POMOCNI              ;"<<" označava pomak u lijevo.

     sei                              ;Opća dozvola prekida.

;Tek nakon opće dozvole za servisne rutine prekidi će stvarno i biti omogućeni.

glavna_petlja:

     cp   TIPKA,RFF                   ;Ako nijedna tipka nije pritisnuta,
     breq glavna_petlja               ;ponovo u petlju.

     rcall    toggle                  ;Promijenimo stanje pripadnog bita.

     ser  TIPKA                       ;Poništimo stanje da možemo očitati
                                      ;sljedeću tipku.
     rjmp glavna_petlja               ;Beskonačna petlja.

;----------------------------------------------------------------------------
;
toggle:

     ldi  MASKA, 1                    ;Priprema za pomicanje.
```

```
     cpi  TIPKA, 0                  ;Ako je redni broj tipke 0, ne treba pomak,
     breq nula_je                   ;i idemo odmah na promjenu najnižeg bita.
posmak:
     lsl  MASKA                     ;Pomak ulijevo onoliko puta
     dec  TIPKA                     ;koliki je redni broj tipke.
     brne posmak

nula_je:
     in   POMOCNI, PORTB            ;Uzmemo stanje LED dioda.
     eor  POMOCNI, MASKA            ;Promijenimo stanje bita TIPKA (toggle).
     out  PORTB, POMOCNI            ;Novo stanje LED dioda.

     ret                            ;povratak u glavnu petlju.

;***************************************************************************
```

DODATAK: PREGLED INSTRUKCIJA AVR MIKROKONTROLERA

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3

RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None 1 / 2 /	3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None 1 / 2 /	3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None 1 / 2 /	3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None 1 / 2 /	3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None 1 / 2 /	3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PX¬PC+k + 1	None 1 /	2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PX¬PC+k + 1	None 1 /	2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None 1 /	2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None 1 /	2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None 1 /	2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None 1 /	2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None 1 /	2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None 1 /	2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None 1 /	2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None 1 /	2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None 1 /	2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None 1 /	2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None 1 /	2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None 1 /	2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None 1 /	2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None 1 /	2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None 1 /	2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None 1 /	2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None 1 /	2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None 1 /	2
Mnemonics	**Operands**	**Description**	**Operation**	**Flags**	**#Clocks**
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1

LDI	Rd, K	Load Immediate	Rd ← K	N one	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to	SRAM (k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2

CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)¬C,Rd(n+1)← Rd(n),X¬Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)¬C,Rd(n)← Rd(n+1),X¬Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles Rd(3..0)¬	Rd(7..4),Rd(7..4)¬Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	N one	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	3
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1