Univerzitet u Nišu

Elektronski fakultet

Realizacija semafora na bazi mikrokontrolera AT89S8253

Projekat iz predmeta: **Embeded sistemi**
Predmetni nastavnik: Prof. Dr. Mile Stojčev

Semestar: **6.**

studijska grupa: **Elektronska kola i sistemi**

Studenti:

Miloš Petković, index 11591

Rade Rakić, index 11601

Datum prijema projekta: 18.06.2007.
Datum predaje projekta: 10.07.2007.

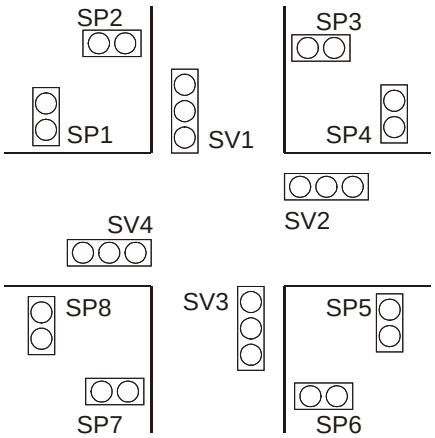
Sadržaj

Projektni zadatak

Realizovati mikroračunar, baziran na mikrokontroleru AT89S8253 shodno priloženoj šemi.

a) mikroračunar se koristi kao kontroler za upravljanje uličnim semaforom, kao što je prikazano na slici 1. Semafor čine 4 semaforske table sa po tri sijalice namenjene za regulaciju saobraćaja vozila (SV1, SV2, SV3, SV4) i 8 semaforskih tabli sa po dve sijalice namenjene za regulaciju prolaska pešaka (SP1, SP2,..., SP8).

b) Kreirati programsku sekvencu za regulaciju saobraćaja na raskrsnici gde se kao parametri unose vremena trajanja sekvenci crveno, žuto i zeleno.



Slika 1. Raskrsnica sa semaforima.

Razrada

Na slici 1. prikazana je jedna tipična raskrsnica. Može se zaključiti da određeni semafori rade u paraleli tj. istovremeno imaju ista stanja. Tako se mogu semafori po načinu rada svrstati u 4 grupe. Za regulaciju jednog pravca koriste se dve grupe i to jedna za pešake a druga za vozila.

	vozila	pešaci
pravac 1:	grupa1: SV1, SV3	grupa2: SP1, SP4, SP5, SP8
pravac 2:	grupa3: SV2, SV4	grupa4: SP2, SP3, SP6, SP7

Tabela 1. Semaforske grupe.

Iz tabele 1 može se videti kako su semafori grupisani. Svaka grupa deli iste kontrolne signale tako da su npr. za kontrolu semafora SP1, SP4, SP5 i SP8 potrebna samo dva kontrolna signala, jedan za crveno i drugi za zeleno svetlo. Na ovaj način je broj kontrolnih signala koji ide od mikrokontrolera do semafora smanjen na deset, što se može videti iz tabele 2 u kojoj je prikazan broj kontrolnih signala za svaku grupu.

grupa	grupa 1	grupa 2	grupa 3	grupa 4
broj signala	3	2	3	2

Tabela 2. Broj kontrolnih signala po semaforskoj grupi.

Ovde se može postaviti i pitanje zašto je za semafore za pešake potrebno koristiti posebno signal za zeleno, posebno za crveno kada su oni pri normalnom radu semafora komplementarni i jedan se može dobiti invertovanjem drugog. Ovo razmišljanje je u redu ali onda će uvek jedno svetlo na semaforu za pešake da gori. To nije poželjno za mod rada semafora kada samo treba da trepće žuto svetlo a sva ostala da budu isključena. Ovaj mod rada javlja se kada saobraćajac kontroliše saobraćaj.

Znači semafor ima dva moda rada: *normalni* i *trepćuće žuto*. Za svaki od ova dva moda potrbno je utvrditi broj stanja koji se javlja, kao i odgovarajuće vrednosti kontrolnih signala. Radi jednostavnosti pisanja, kontrolni signali su označeni sa S1 do S10 a njihova uloga u kontroli semafora prikazana je u tabeli 3.

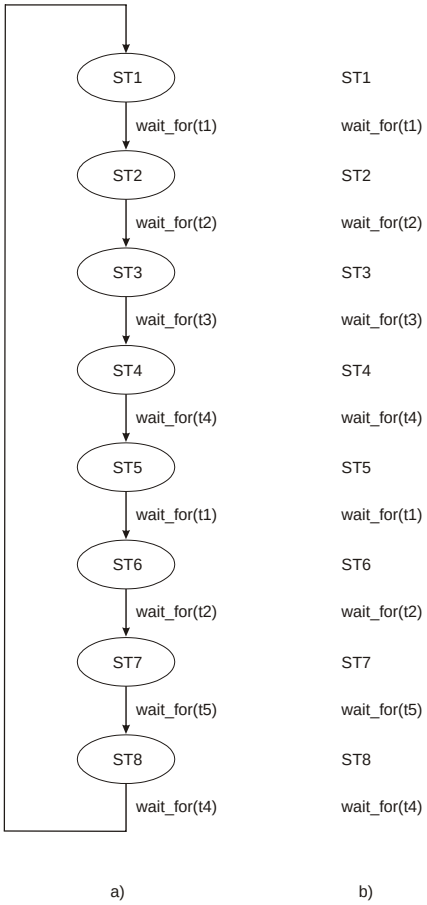
kontrolni signal	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
semaforski signal	grupa 2 zeleno	grupa 2 crveno	grupa 4 zeleno	grupa 4 crveno	grupa 1 zeleno	grupa 1 žuto	grupa 1 crveno	grupa 3 zeleno	grupa 3 žuto	grupa 3 crveno

Tabela 3. Uloga kontrolnih signala S1-S10 u kontroli semafora.

Pri *normalnom* radu semafora jedna tipična sekvenca stanja semafora za jedan ciklus je sledeća:

1. sve crveno,
2. priprema za zeleno za pravac 1 (pali se žuto dok još uvek svetli crveno na semaforu za vozila),
3. zeleno za pravac 1,
4. priprema za crveno za pravac 1 (gasi se zeleno a pali žuto na semaforu za vozila, dok se gasi zeleno i pali crveno na semaforu za pešake),
5. sve crveno,
6. priprema za zeleno za pravac 2,
7. zeleno za pravac 2,
8. priprema za crveno za pravac 2.

Radi lakšeg pisanja označićemo ova stanja sa ST1 do ST8 respektivno. Iz jednog stanja u drugo ulazi se nakon isteka vremena trajanja trenutnog stanja. Određena stanja imaju isto vreme trajanja zbog simetričnosti rada semafora tako npr. stanja ST1 i ST5 (sve crveno) imaju isto vreme trajanja *t1*. Takođe i stanja ST2 i ST6 imaju isto vreme trajanja *t2*, kao i stanja ST4 i ST8 koja imaju vremena trajanja *t4*. Stanja ST3 i ST7 (dozvola za pravc 1 i 2) mogu imati i različita vremena trajanja, *t3* i *t5* respektivno, kako bi se otklonili problemi oko saobraćaja pri nejednakom opterećenju pravaca. Na taj način, ako npr. pravacem 1 prolazi više vozila nego pravcem 2, onda se može podesiti da vreme kada je otvoren semafor u pravcu 1 traje duže nego na pravcu 2. Na slici 2.a prikazan je ASM dijagram za normalni rad semafora, dok je na slici 2.b prikazana odgovarajuća pseudo programska sekvenca.

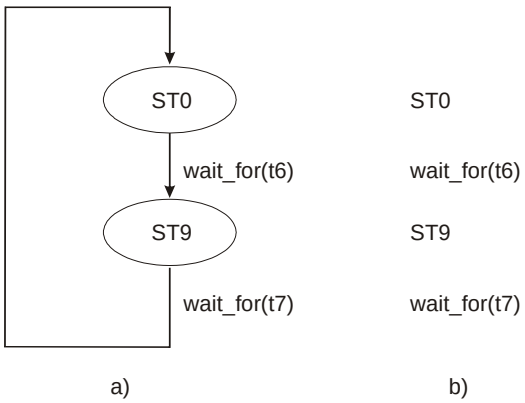


Slika 2. a) ASM dijagram za normalan rad b) pseudo kod

Pri modu rada semafora *trepćuće žuto* postoje samo dva stanja u sekvenci jednog ciklusa:

1. sve isključeno i
2. svetli žuto.

Ova stanja ćemo označiti sa ST0 i ST9, a vremena njihovih trajanja sa *t6* i *t7*. Na slici 3.a prikazan je ASM dijagram za rad semafora u modu *trepćuće žuto*, dok je na slici3.b prikazana odgovarajuća pseudo programska sekvenca.



Slika 3. a) ASM dijagram za rad trpćuće žuto b) pseudo kod

Tabela 4. sistematizuje prethodna razmatranja prikazom vrednosti kontrolnih signala za sva moguća stanja.

signal/ stanje	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
ST0	0	0	0	0	0	0	0	0	0	0
ST1	0	1	0	1	0	0	1	0	0	1
ST2	0	1	0	1	0	1	1	0	0	1
ST3	1	0	0	1	1	0	0	0	0	1
ST4	0	1	0	1	0	1	0	0	0	1
ST5	0	1	0	1	0	0	1	0	0	1
ST6	0	1	0	1	0	0	1	0	1	1
ST7	0	1	1	0	0	0	1	1	0	0
ST8	0	1	0	1	0	0	1	0	1	0
ST9	0	0	0	0	0	1	0	0	1	0

Tabela 4. Vrednosti kontrolnih signala u stanjima ST0-ST9.

Znači logički blok semaforskog sistema treba da ima 10 izlaza i jedan ulaz. Proizvoljno je odlučeno da se za izlaze koriste portovi 1 i 3 mikrokontrolera, i to svih osam pinova porta 3 i dva LSB pina porta 1, a za ulaz port dva, tačnije LSB pin porta dva. U tabeli 5. prikazana je veza između signala S1-S10 i pinova portova 1 i 3.

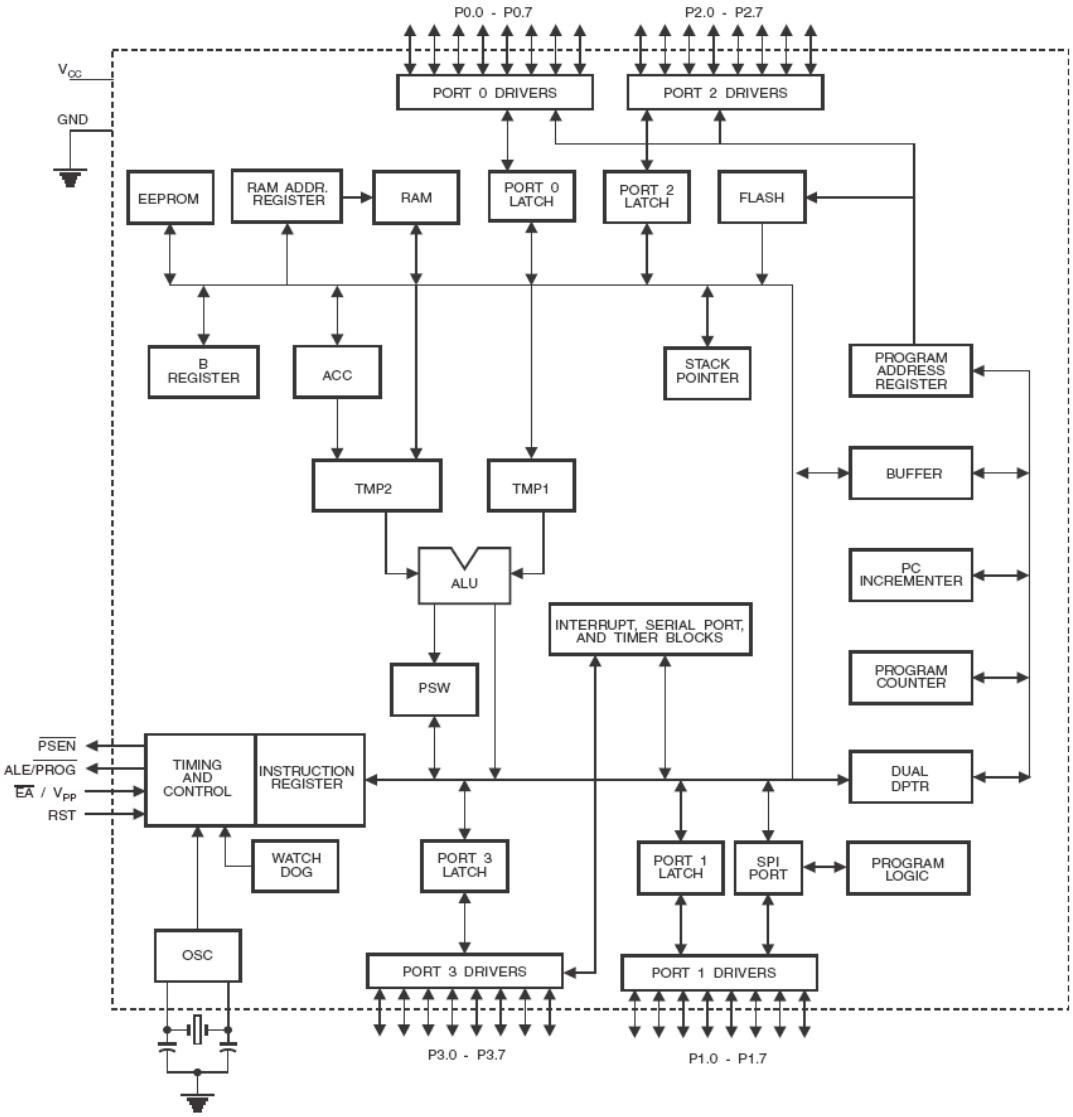
signal	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
pin porta	port 3 pin 0	port 3 pin 1	port 3 pin 2	port 3 pin 3	port 3 pin 4	port 3 pin 5	port 3 pin 6	port 3 pin 7	port 1 pin 0	port 1 pin 1

Tabela 5. Veza između kontrolnih signala i pinova portova.

Realizacija

Pošto je realizacija bazirana na upotrebi mikrokontrolera AT89S8253 onda će, najpre, biti opisana njegova struktura i osnovne osobine neophodne za realizaciju ovog projekta.

Ovaj mikrokontroler baziran je na strukturi 8251, a njegova blok blok šema prikazana je na slici 4.



Slika 4. Struktura mikrokontrolera AT89S8253.

AT89S8253 je osmobitini mikrokontroler visokih performansi izrađen u CMOS tehnologiji. Od memorijskih resursa sadrži:

- 12k fleš memoriju, koja podržava ISP (In-System Programmable) način programiranja, predviđenu za smeštanje programa,

- 2K EEPROM memorije za podatke, koja takođe podržava ISP, i
- 256 bajta RAM-a.

Ima 32 ulazno/izlazna pina, raspoređena u 4 porta sa po osam pinova. Poseduje *watchdog tajmer*, tri 16-bitna tajmera/brojača, devet izvora prekida i ugrađen oscilator (kvarcni oscilator se vezuje na pinove XTAL1 i XTAL2). Radi smanjenja potrošnje, pored normalnog moda rada, mikrokontroler može da radi u *Idle* modu ili *Power-down* modu.

Portovi od 0 do 3, kao i drugi resursi mikrokontrolera, mogu imati više funkcija. Ovde će biti pomenute samo one koje su bitne za ovu aplikaciju.

Kao što je već rečeno, za kontrolne signale će tokom rada biti korišćeni portovi 1, 2 i 3 sa već pomenutom ulogom. Važno je naglasiti da se port 1 koristi pri programiranju fleš memorije. Tako pinovi P1.5 do P1.7 imaju sledeću ulogu: P1.5 je ujedno serijski ulaz MOSI (*Master Output Slave Input*), P1.6 je MISO (*Master Input Slave Output*), i P1.7 je SCK (*Sampling Clock*).

Sam algoritam serijskog programiranja je sledeći:

1. priključi se napajanje i pin za reset RST se postavi na visoko, H.
2. uključi se serijski način programiranja slanjem instrukcije serijskog programiranja na pin MOSI/P1.5. Pri tome se na pin SCK/P1.7 dovodi takt frekvencije barem 16 puta manje od frekvencije rada mikrokontrolera.
3. programira se fleš slanjem odgovarajuće instrukcije za upis i podataka. Mogu se pri programiranju koristiti dva moda: bajtovski ili stranični.
4. upis na bilo koju lokaciju može se verifikovati slanjem naredbe za čitanje, pri čemu se sadržaj memorijske lokacije dobija na pinu MISO/P1.6.
5. RST se vraća na niski nivo i mikrokontroler može da počne sa izvršavanjem programa.

Za programiranje fleš memorije u ovom slučaju je korišćen program *ISP Programmer* (program napisao Adam Dybkowski).

Pri radu će se koristiti tajmer 0 za merenje vremena. Kontrola rada ovog tajmera definiše se preko registara TCON i TMOD sa adresama 88h i 89h respektivno. Njihova struktura prikazana je na slici 5.

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

a)

7	6	5	4	3	2	1	0
GATE1	C/T1#	M11	M01	GATE0	C/T0#	M10	M00

b)

Slika 5. Struktura registra a) TCON i b) TMOD.

Značenje kontrolnih bitova je sledeće:

- TR0 služi za dozvolu rada tajmera 0

- TF0 se postavlja na jedinicu kada se javi prekoračenje tajmera, mora se softverski obrisati

- IT0 se postavlja na nulu da bi se podesilo okidanje donjom ivicom (level trigger) interapt signala INT0# ili se postavlja na jedinicu da bi se podesilo okidanje opadajućom ivicom INT0# signala.

- IE0 se postavlja na nulu od strane hardvera kada je interapt opslužen (kada je ivično trigerovan) ili se postavlja hardverski kada se detektuje interapt na pinu INT0#

- GATE0 treba da se postavi na nulu da bi se dozvolio rad Tajmera 0 kada god je setovan TR0 ili treba da se postavi da bi se dozvolio rad tajmera/brojača 0 samo kada je INT0# postavljen na visoko i TR0 setovan

- C/T0 služi da se izabere mod rada kao brojač ili kao tajmer

- bitovima M10 i M00 bira se mod rada. Modovi rada su dati u tabeli 6.

M10	M00	mod
0	0	8-bitni tajmer/brojač sa 5-bitnim preskalerom
0	1	16-bitni tajmer/brojač
1	0	8- bitni tajmer/brojač sa automatskim reloadovanjem
1	1	8-bitni tajmer/brojač

Tabela 6. Modovi rada za tajmer 0.

Pored ova dva registra postoje i još dva registra TH0 i TL0 sa adresama 8Ch i 8Ah, respektivno, koji predstavljaju stanje brojača/tajmera.

Kada se koristi kao tajmer, on uvećava svoj sadržaj na svakih dvanaest taktnih intervala, što ujedno iznosi i vreme izvršenja jedne instrukcije. S obzirom da je to 16-bitni registar, to znači da maksimalni interval vremena koji može da se izmeri pri radnoj frekvenciji mikrokontrolera od 11.0592MHz iznosi približno 74ms.

Da bi mogao da se prihvati interapt od tajmera 0 ili nekog drugog izvora, mora da se setuje registar dozvole interapta, IE, sa adresom A8h. Struktura ovog registra data je u tabeli 7.

7	6	5	4	3	2	1	0
EA	-	ET2	ES	ET1	EX1	ET0	EX0

Tabela 7. Struktura registra IE.

Uloga ovih bitova je:

- EA bit ako nije postavljen onda se ignorišu svi interapti.

- ET2 bit dozvoljava prihvatanje interapta tajmera 2.

- ES je bit dozvole interapta za SPI i UART.

- ET1 bit dozvoljava prihvatanje interapta tajmera 1.

- EX1 bit dozvoljava prihvatanje spoljašnjeg interapta 1.

- ET0 bit dozvoljava prihvatanje interapta tajmera 0.

- EX0 bit dozvoljava prihvatanje spoljašnjeg interapta 0.

Watchdog tajmer broji instrukcione cikluse. Poseduje preskaler, tako da se maksimalni opseg može izabrati pomoću bitova PS2, PS1 i PS0. Maksimalni opseg u zavisnosti od stanja ovih bitova dat je u tabeli 8.

PS2	PS1	PS0	maksimalni opseg
0	0	0	16k
0	0	1	32k
0	1	0	64k
0	1	1	128k
1	0	0	256k
1	0	1	512k
1	1	0	1024k
1	1	1	2048k

Tabela 8. Maksimalni merni opseg watchdog tajmera.

Watchdog tajmer je kontrolisan preko kontrolnog registra WDTCON sa adresom A7h. Struktura registra prikazana je u tabeli 9.

7	6	5	4	3	2	1	0
PS2	PS1	PS0	WDIDLE	DISTRO	HWDT	WSWRST	WDTEN

Tabela 9. Struktura registra WDTCON.

Uloga bitova u kontroli je:

- PS2, PS1, PS0 su bitovi preskalera

- WDIDLE bit - ako nije postavljen, omogućava da watchdog tajmer nastavlja da broji i kada je mikrokontroler u Idle stanju; kada je on postavljen, watchdog tajmer ne broji u Idle stanju mikrokontrolera.

- DISTRO bit - kada nije postavljen, nakon što watchdog tajmer odbroji, onda se mikrokontroler resetuje a pin RST se postavi i može da resetuje čitav sistem oko mikrokontrolera; kada je ovaj bit postavljen, nakon isteka intervala watchdog tajmera, mikrokontroler se resetuje, a RST pin ostaje samo ulazni pin.

- HWDT bit - kada nije postavljen, watchdog može biti resetovan jednostavno postavljanjem bita WSWRST, međutim ukoliko je postavljen, onda se watchdog pokreće u rad upisom sekvence 1EH/E1H u WDTRST registar sa adresom A6h (posle ovakvog puštanja u rad ne može se resetovati watchdog bitom WSWRST već se mora upisati sekvenca 1EH/E1H u WDTRST SFR).

- WSWRST bit služi za resetovanje watchdog tajmera; kada se jednom ovaj bit postavi, automatski se resetuje u sledećem instrukcionom ciklusu.

- WDTEN bit je bit dozvole rada watchdog tajmera ukoliko je HWDT=0.

Da bi se smanjila potrošnja, mikrokontroler pored normalnog načina rada, poseduje još dva moda rada: Idle i Power-down mod. Stavljanje mikrokontrolera u Idle stanje znači da će on prestati sa izvršenjem instrukcija ali će sve periferije ostati aktivne. Ostaje u ovom stanju dok se ne resetuje ili prihvati interapt. Nakon obrade interapta nastavlja sa izvršenjem instrukcija od mesta u programu gde je ušao u Idle stanje.

Mikrokontroler ulazi u Idle stanje postavljanjem LSB bita PCON registra sa adresom 87h. Kada se mikrokontroler nađe u Power-down stanju onda se, ne samo prestaje sa izvršenjem instrukcija, već se i periferije zamrzavaju i oscilator prestaje sa radom. Iz ovog stanja se izlazi resetom ili primanjem spoljašnjeg interapta. U Power-down stanje se ulazi setovanjem četvrtog bita PCON registra.

Prethodno navedeni registri nalaze se u SFR (*Special Function Registers*) prostoru koji je grafički prikazan na slici 6.

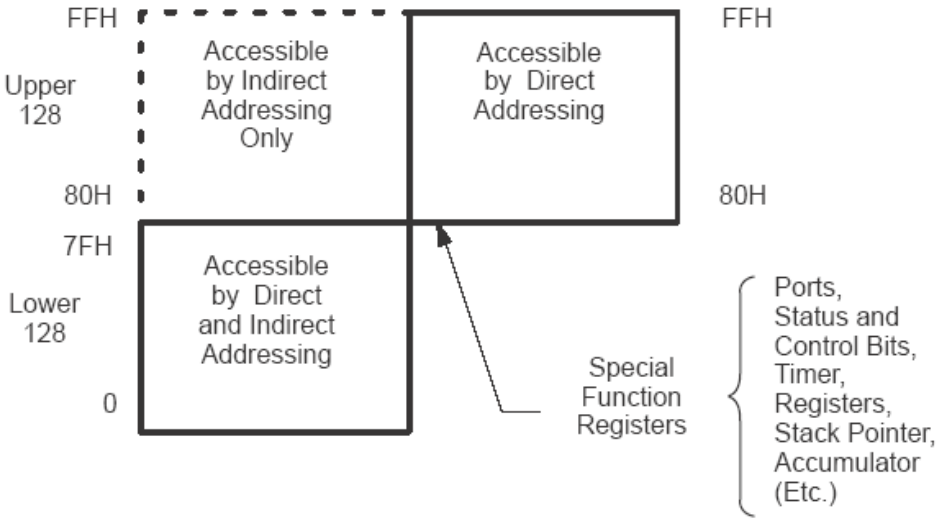
Table 1. AT89S8253 SFR Map and Reset Values

0F8H									0FFH
0F0H	B 00000000								0F7H
0E8H									0EFH
0E0H	ACC 00000000								0E7H
0D8H									0DFH
0D0H	PSW 00000000					SPCR 00000100			0D7H
0C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			0CFH
0C0H									0C7H
0B8H	IP XX000000	SADEN 00000000							0BFH
0B0H	P3 11111111							IPH XX000000	0B7H
0A8H	IE 0X000000	SADDR 00000000	SPSR 000XXX00						0AFH
0A0H	P2 11111111						WDTRST (Write Only)	WDTCON 0000 0000	0A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111						EECON XX000011		97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXXXXXX0	CLKREG XXXXXXX0	8FH
80H	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000	SPDR ########	PCON 0XXX0000	87H

Note: # means: 0 after cold reset and unchanged after warm reset.

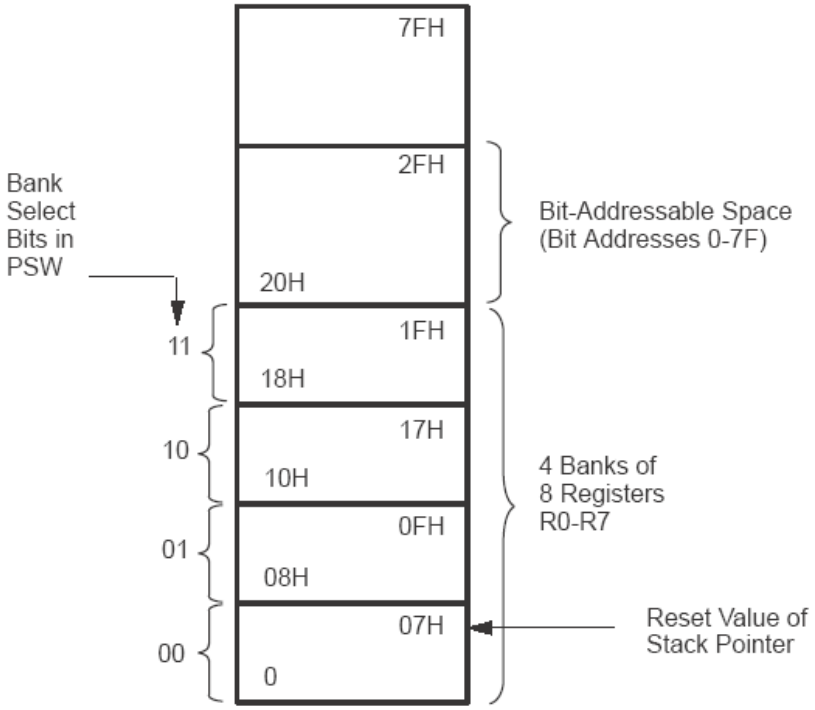
Slika 6. SFR prostor.

SFR deli adresni prostor sa RAM-om. To znači da se adrese za viših 128 bajtova RAM-a koriste za adresiranje SFR-a. U zavisnosti od načina adresiranja pristupa se ili SFR-u ili višem delu RAM-a. Tako se direktnim adresiranjem pristupa SFR-u, dok se samo indirektnim adresiranjem može pristupiti 128 višim bajtovima RAM-a. Ovo je grafički prikazano na slici 7.



Slika 7. Preklapanje SFR i RAM adresnog prostora.

Nižim 128 bajtovima RAM-a može da se pristupa i direktno i indirektno. Organizacija nižih 128 bajtova RAM-a prikazana je na slici 8.

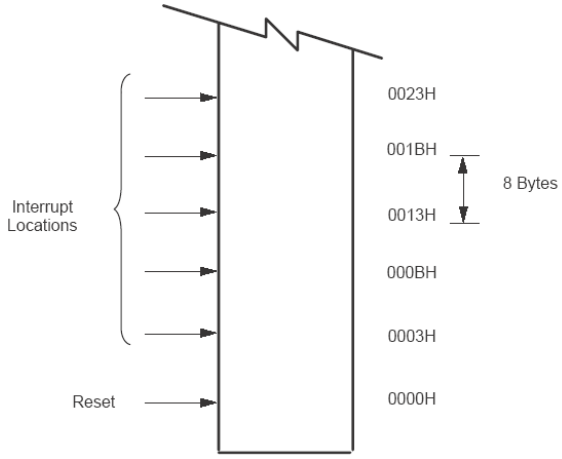


Slika 8. Organizacija nižih 128 bajtova RAM-a.

Kao što se sa slike 8. može videti, nižih 32 bajta je podeljeno u četiri banke registara. Svaka banka sadrži osam registra. Izbor banke obavlja se postavljanjem trećeg i četvrtog bita u PSW registru na odgovarajuću vrednost. Podrazumevana vrednost (*default*) selektuje prvu banku, a Stek Pointer (SP) je postavljen na adresu 07h. To znači da magacin zauzima prostor iznad prve banke. Međutim, ovo se može promeniti po potrebi tako da je položaj magacina proizvoljan kao i njegova veličina.

Memorijski prostor RAM-a u opsegu adresa od 20h do2Fh je bit adresibilan, ali to u ovoj aplikaciji nije od važnosti.

Važno je napomenuti da se podaci mogu čuvati i u EEPROM-u ili nekoj spoljašnjoj memoriji, mada se to ovde ne koristi. Fleš memorija isključivo služi za smeštanje programa i u nju se ne mogu upisivati podaci tokom rada mikrokontrolera. Grafički prikaz organizacije nižeg dela programske memorije prikazan je na slici 9.



Slika 9. Niži deo programske memorije.

Ovo je sve što je potrebno da se zna o kontroleru i njegovim funkcijama koje se koriste u ovoj aplikaciji. Dodatne informacije mogu se naći na atmelovom sajtu na adresi www.atmel.com.

Softver

Ono što se može zaključiti na osnovu ASM dijagrama rada semafora, o kojima je bilo reči u razradi, je da je veći deo posla mikrokontrolera da meri vremena trajanja stanja semafora. Merenje vremena može da se uradi na dva načina: softverski i hardverski. Softverski način podrazumeva da se napravi jedna petlja sa unapred definisanim brojem prolaska kroz pelju. Ukoliko je vreme izvršenja jednog ciklusa petlje poznato, onda se ukupno vreme dobija kao proizvod broja prolaska kroz petlju i vremena trajanja jednog ciklusa petlje. Najprostiji oblik ovakve petlje na C-u bi bio: for(i=0;i<n;i++). Ono što nije dobro kod ovakvog načina merenja vremena je što mikroprocesor sve svoje vreme troši na izvršavanje petlje i ne sme da prima interapte, jer će, u protivnom, izmereno vreme biti pogrešno. Upravo zbog toga je mnogo bolje koristiti tajmere mikrokontolera za merenje vremena, jer oni rade nezavisno od mikroprocesora. Ne samo da mikroprocesor za to vreme može da izvršava neki program, već može biti stavljen u Idle stanje radi smanjenja potrošnje energije. Ovo je vrlo povoljno sa stanovišta ove aplikacije jer procesor ne radi nikakva izračunavanja dok se semafoski sistem nalazi u jednom stanju.

Kao što je već napomenuto, tajmeri mikrokontolera su 16-bitni i maksimalni vremenski interval koji oni mogu da izmere pri radnoj frekvenciji kontrolera od 11.0592MHz je 74ms. Kako vremena menjanja svetala na semaforu iznose i do 1

minuta, to je očigledno nemoguće čisto hardverski, pomoću tajmera, meriti vreme. Iz tog razloga se pravi petlja u kojoj se koristi hardversko vreme merenja vremena. Neka je, za sada, *sleep_for()* funkcija koja setuje tajmer za merenje određenog vremenskog intervala i stavlja mikrokontroler u Idle stanje. Onda se na C-u može napisati sledeći kôd za nerenje dužih vremenskih intervala:

```
for(i=0;i<n;i++)
{
        sleep_for()
}
```

Približno vreme koje se meri ovim kôdom jednako je proizvodu broja prolazaka kroz petlju i vremena koje meri tajmer. Naravno, vreme koje se potroši za izvršavanje petlje i setovanje tajmera treba dodati ovom vremenu. Međutim, s obzirom da se ne čini značajna greška, ova vremena se zanemaruju. Poželjno je da tajmer meri što duži vremenski interval. Iako maksimalno vreme koje može da izmeri iznosi 74ms, radi lakšeg računanja, najbolje je izabrati za osnovni vremenski interval tajmera od 50ms. Setovanje Tajmera 0 da meri vreme od 50ms omogućeno je sledećim kôdom:

```
TMOD &= 0xF0; // setovanje moda rada Tajmera 0
TMOD |= 0x01; // setovanje moda rada Tajmera 0
TH0 = 0x3C; // upisivanje 8 MSB bita broja 15536 u viši registr Tajmera 0
TL0 = 0xB0; // upisivanje 8 LSB bita broja 15536 u niži registr Tajmera 0
```

Prva dva reda kôda postavljaju Tajmer 0 u mod rada: 16-bitni tajmer. Druga dva reda upisuju 15536 kao vrednost brojača tajmera. To znači da će tajmer da izbroji 65536-15536=50000 instrukcionih ciklusa. Pošto jedan instrukcioni ciklus ciklus traje 12 taktnih intervala i ako, radi lakšeg računanja, pretpostavimo da je takt 12MHz, onda jedan instrukcioni ciklus traje 1μs. Zaokruživanje taktne frekvencije na 12MHz ne unosi veliku grešku i ona iznosi 4.3ns pri merenju vremena od 50ms. Pored predhodna četiri reda kôda potrebno je dodati i kôd za resetovanje fleg bita tajmera i setovnje dozvole rada tajmera. Pošto će se mikrokontroler nalaziti u Idle stanju za vreme merenja 50ms, onda se mora pridodati i kôd za njegovo postavljanje u Idle stanje i za dozvolu interapta od strane Tajmera 0. Dozvola interapta se mora postaviti kako bi mogao mikrokontroler da se vrati u normalno stanje izvršenja nakon isteka željenog vremenskog intervala. Svi prethodno pomenuti delovi kôdova mogu biti objedinjeni u funkciji pod nazivom *sleep_for_50ms()*. Tačan C opis ove funkcije je:

```
void sleep_for_50ms(void)
{
        TMOD &= 0xF0; // setovanje moda rada Tajmera 0
        TMOD |= 0x01; // setovanje moda rada Tajmera 0
        TH0 = 0x3C; // upis 8 MSB bita broja 15536 u viši registr Tajmera 0
        TL0 = 0xB0; // upis 8 LSB bita broja 15536 u niži registr Tajmera 0

        ET0 = 1; //dozvola interapta Tajmera 0
        EA=1; //globalna dozvola interapta
```

```
        TF0 = 0; //brisanje overflow-a
        TR0 = 1; //pustanje u rad brojaca

        PCON |= 0x01; //stavljanje procesora u idle stanje radi ustede energije
    }
```

Može se uočiti da stavljanjem ove funkcije u petlju unosi oko dvadesetak instrukcionih ciklusa koji će da unesu grešku u ukupno merenje vremena. Ovo se može kompenzovati oduzmanjem broja ovih nepoželjnih ciklusa od broja ciklusa tajmera koje on treba da izmeri. S obzirom da u ovoj aplikaciji tačnost merenja vremena nije mnogo kritična, a greška koju unosi dvadesetk redova kôda iznosi 0.04%, ono se može zanemariti.

Mnogo bitnija je pouzdanost rada semafora. Iz tog razloga se, za slučaj da se iz nekog razloga mikrokontroler ne probudi iz Idle stanja, setuje watchdog tajmer, koji će da resetuje sistem ukoliko on sam nije resetovan na 128 instrukcionih ciklusa ili približno 128ms. Instrukcija za resetovanje tajmera je zato ubačena u funkciju *sleep_for_50ms()* iza instrukcije za stavljanje kontrolera u Idle stanje. Kompletna funkcija sleep_for_50ms() sada izgleda.

```
    void sleep_for_50ms(void)
    {
        TMOD &= 0xF0; // setovanje moda rada Tajmera 0
        TMOD |= 0x01; // setovanje moda rada Tajmera 0
        TH0 = 0x3C; // upis 8 MSB bita broja 15536 u viši registr Tajmera 0
        TL0 = 0xB0; // upis 8 LSB bita broja 15536 u niži registar Tajmera 0

        ET0 = 1; //dozvola interapta Tajmera 0
        EA=1; //globalna dozvola interapta

        TF0 = 0; //brisanje overflow-a
        TR0 = 1; //pustanje u rad brojaca

        PCON |= 0x01; //stavljanje procesora u idle stanje radi ustede energije

        WDTCON=0x6B; //resetovanje watch dog tajmera
    }
```

Inicijalizacija watchdog tajmera obavlja se na početku programa main().

Prevodjenje ASM dijagrama preko CASE naredbe u C kod ostvaruje se uvođenjem promenljive *stanje* koja uzima vrednost od 0 do 9. Promenljiva *stanje* se ispituje u naredbi CASE, a grane CASE strukture predstavljaju odgovarajuća stanja u kojima se

semafor nalazi. Odgovarajuće vrednosti kontrolnih signala postavljaju se na portovima kada program uđe u odgovarajuću granu.

Usvojeno je da se semafor nalazi u *normalnom* modu, modu 1, kada je nulti pin porta 2 na nuli. U suprotnom, nalazi se u modu *trepćuće žuto*, mod 2. Da bi se osiguralo da ne dođe do konfliktnih situacija pri prelazu iz jednog stanja u drugo, uvedena su neka ograničenja koja se tiču menjanja moda. Najpe, da bi se dozvolilo da se raskrsnica isprazni pre nego što se pređe iz jednog moda rada u drugi, onda se prelaz iz moda 1 u mod 2 izvršava tek nakon stanja ST1 ili ST5 (sve crveno), a obrnuto se prvo ulazi u stanje ST1. Radi jasnijeg stavljanja do znanja vozačima da je semafor isključen i da treba da se pripreme za poštovanje drugih znakova u saobraćaju (pravo prvenstva ili znakove koje daje saobraćajac) prvo se, pri ulasku u mod rada 2, ulazi u stanje ST0 (sve ugašeno). Jedino se iz ovog stanja može vratiti u mod rada 1.

Poštovanjem svega navedenog, napisan je sledeći *main()* kod:

```
void main()
{
        unsigned char stanje=1;
        P3=74;
        P1=2;
        P2|=1;
        while(1)
        {
                switch (stanje)
                {
                        case 0:
                        {
                                P3=0;
                                P1=0;
                                P2|=1;
                                if ((P2&0x01)==0) stanje=1;
                                else stanje=9;
                                wait_for(t6);
                                break;
                        }
                        case 1:
                        {
                                P3=74;
                                P1=2;
                                P2|=1;
```

```
            if ((P2&0x01)==1) stanje=0;
            else stanje=2;
            wait_for(t1);
            break;
        }
        case 2:
        {
            P3=106;
            P1=2;
            stanje=3;
            wait_for(t2);
            break;
        }
        case 3:
        {
            P3=25;
            P1=2;
            stanje=4;
            wait_for(t3);
            break;
        }
        case 4:
        {
            P3=42;
            P1=2;
            stanje=5;
            wait_for(t4);
            break;
        }
        case 5:
        {
            P3=74;
            P1=2;
            P2|=1;
            if ((P2&0x01)==1) stanje=0;
```

```
            else stanje=6;
            wait_for(t1);
            break;
        }
        case 6:
        {
            P3=74;
            P1=3;
            stanje=7;
            wait_for(t2);
            break;
        }
        case 7:
        {
            P3=198;
            P1=0;
            stanje=8;
            wait_for(t5);
            break;
        }
        case 8:
        {
            P3=74;
            P1=1;
            stanje=1;
            wait_for(t4);
            break;
        }
        case 9:
        {
            P3=32;
            P1=1;
            stanje=0;
            wait_for(t7);
            break;
```

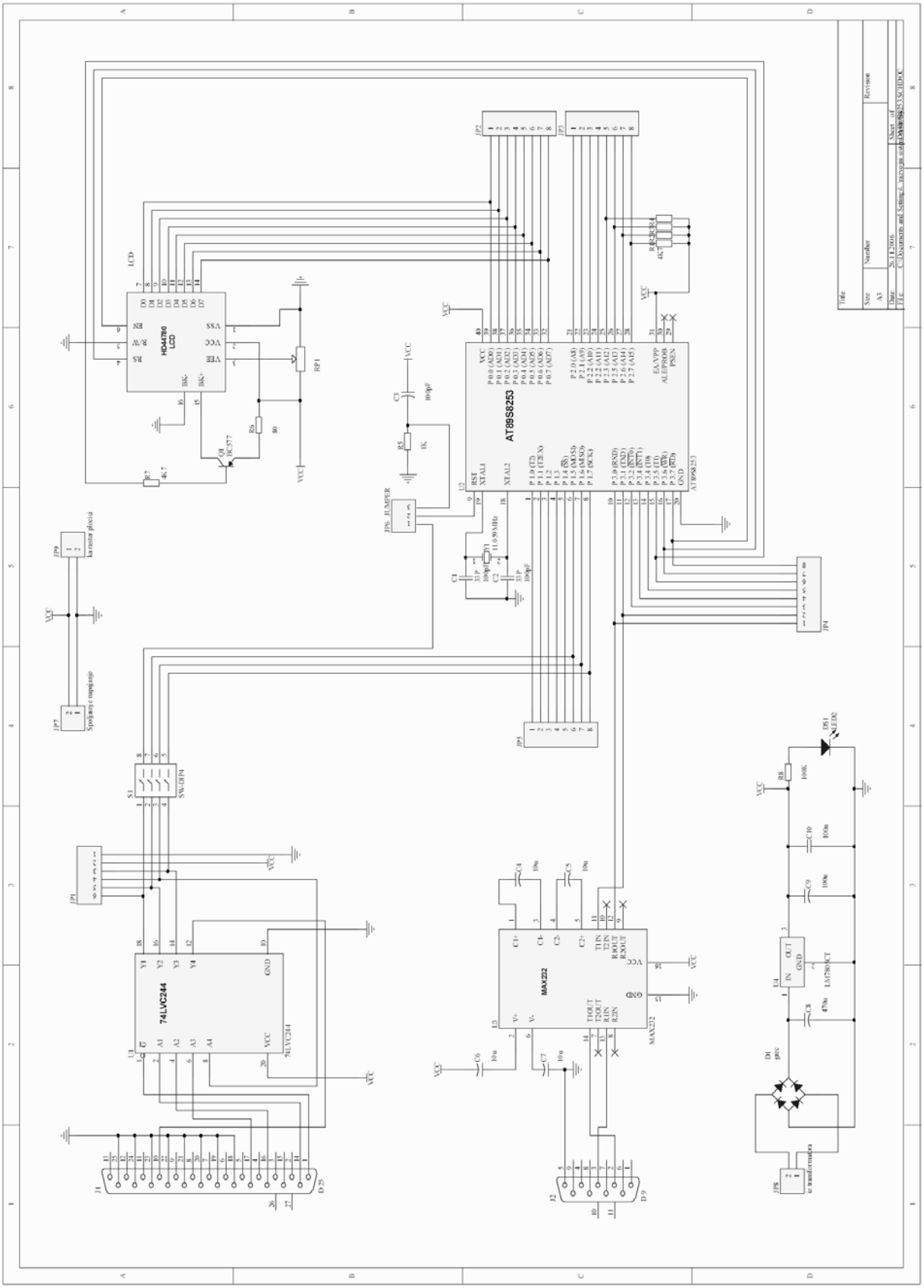
```
                    }
                    default:
                    {
                        P3=74;
                        P1=6;
                        P2|=1;
                        if ((P2&0x01)==1) stanje=0;
                        else stanje=2;
                        wait_for(t1);
                        break;
                    }
                }
            }
        }
```

Iz prethodnog kôda se može videti da su stanja i prelazi jasno definisani, što ga čini razumljivijim, ali ovako napisan kôd nije optimalan po brzini i veličini memorijskog prostora koji zauzima. Zbog toga je izvršena njegova optimizacija izbacivanjem CASE strukture i redova koji se ponavljaju. Kompletan program urađen preko CASE strukture priložen je u Prilogu A, dok je optimizovan program priložen u Prilogu B. Disasemblovan kôd neotimizovanog programa dat je u Prilogu D. On pokazuje lokaciju pojedinih naredbi u memorijskom prostoru.

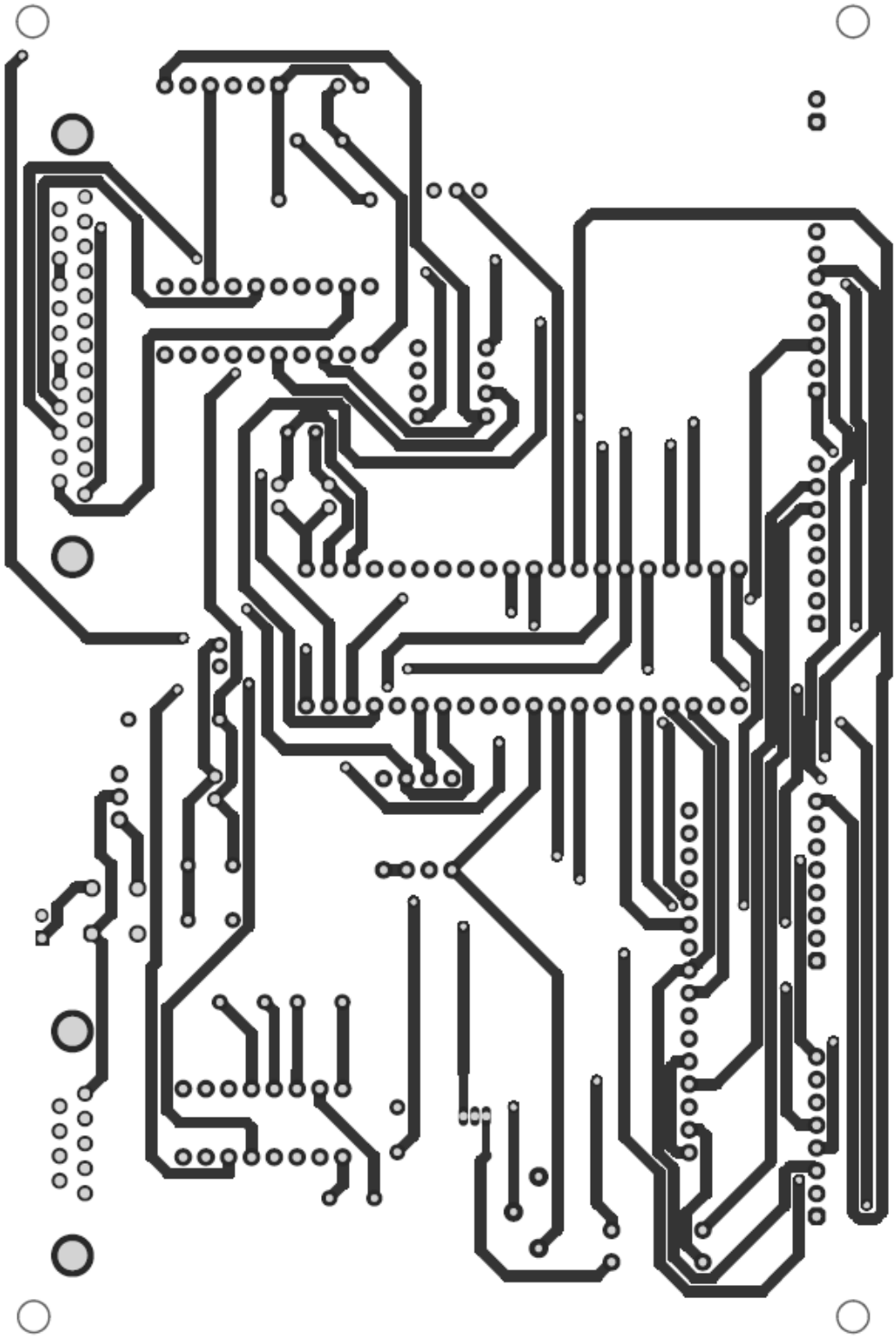
Hardver

Za hardversku realizaciju semaforskog sistema iskorišćena je PCB koju je projektovao student J. Jovanovića. Njena Blok šema data je na slici 10. Šema prikazuje kako su povezani mikrokontroler AT89S8253, ispravljač napona od 220V AC 50Hz na 5V DC (trafo je smesten van PCB ploce), bafersko kolo 74LS244 kao pomoćno kolo za spregu sa računarom preko paralelnog porta, kolo max232 za spregu sa računarom preko RS232 porta.

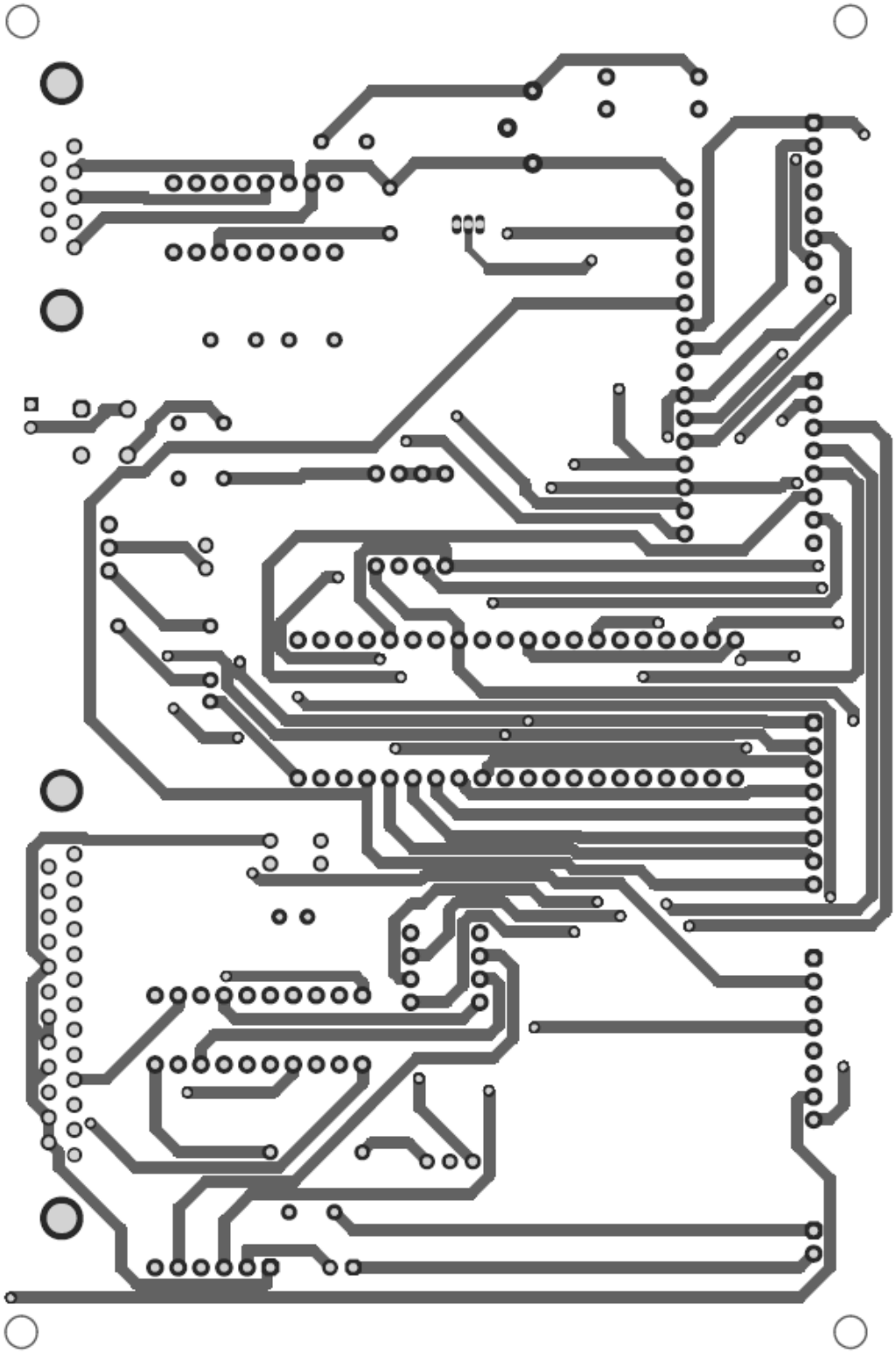
Gornja i donja strana štampane ploče (layout) prikazane su na slikama 11. i 12. respektivno, dok je šema rasporeda komponenata gornje strane ploče prikazana na slici 13.



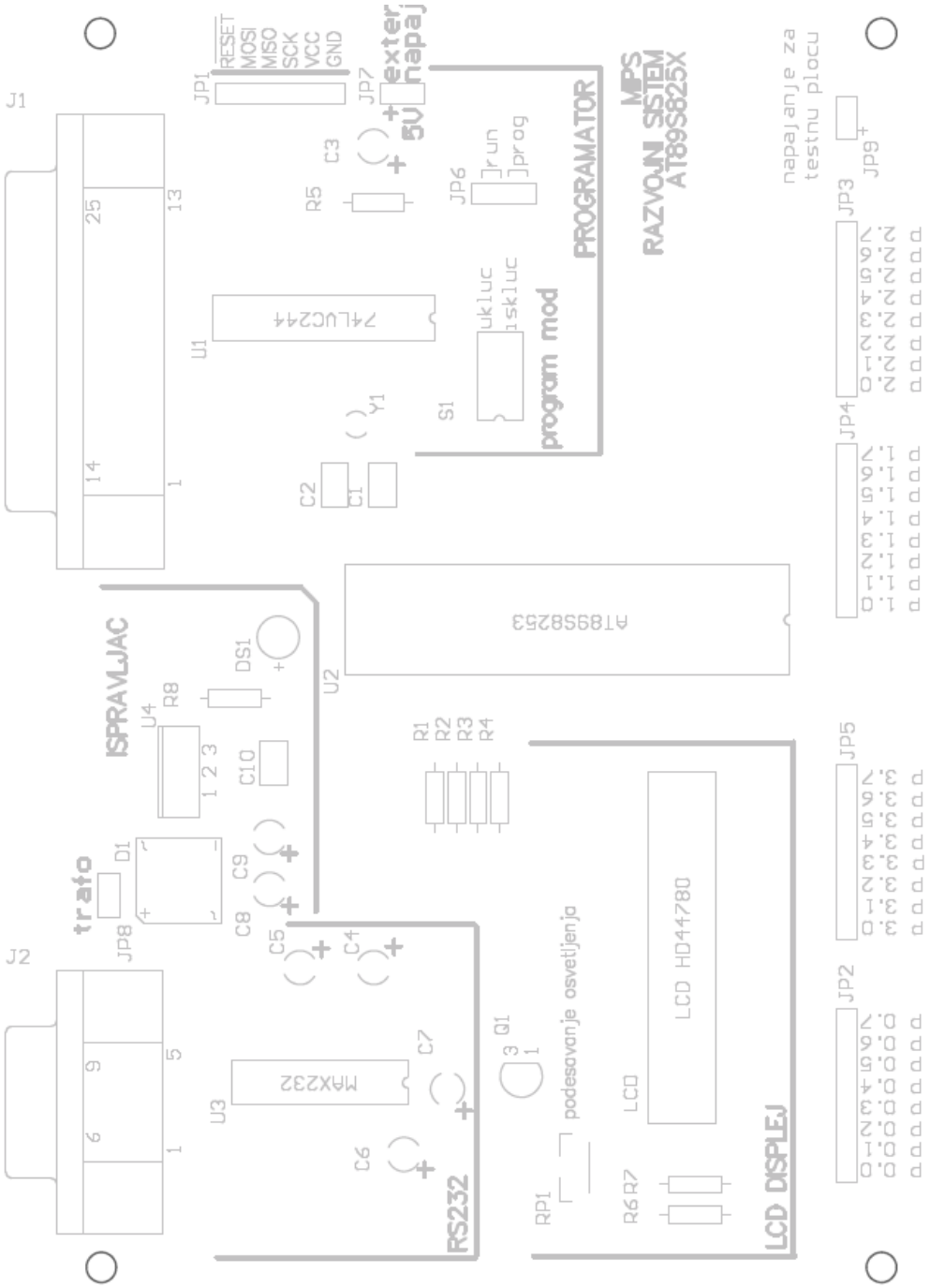
Slika 10. Blok šema elemenata smeštenih na PCB ploči.



Slika 11. Layout gornje strane PCB-a



Slika 12. Layout donje strane PCB-a



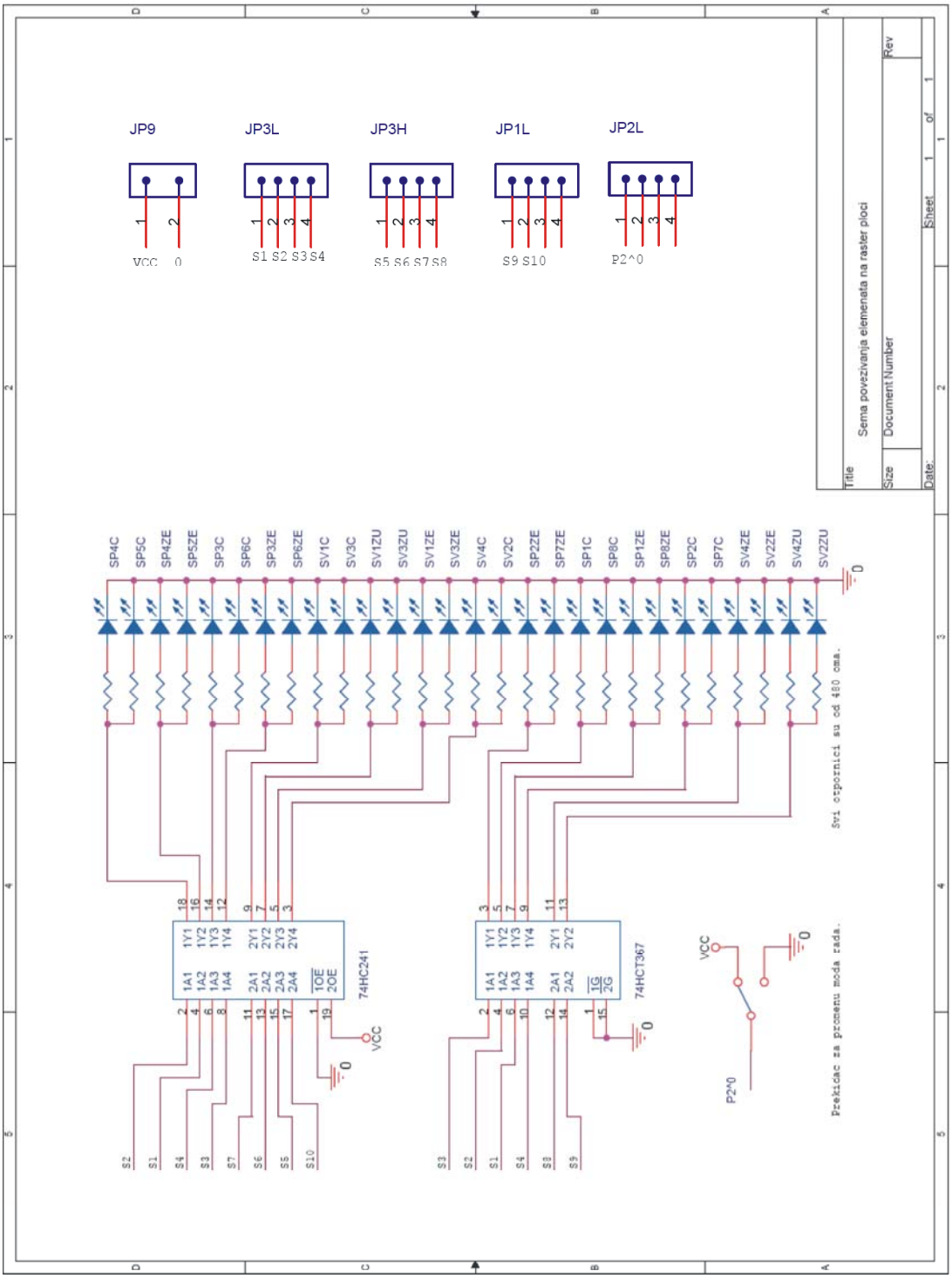
Slika 13. Prikaz rasporeda elemenata odštampan na gornjoj strani PCB-a

Rad semafora prikazuju LED koje su smestene odvojeno na raster ploči a napajaju se sa ploče na kojoj je smesten mikrokontroler. Na raster ploči nalaze se, takodje, i baferi preko kojih se diode sprežu sa mikrokontrolerom. Blok šema sistema na raster ploči prikazana je na slici 14.

Raster ploča se preko ženskih konektora spaja sa muškim konektorima na PCB-u. Šema povezivanja konektora data je u tabeli 10.

konektor sa raster ploče	JP9	JP3L	JP3H	JP1L	JP2L
konektor na PCB-u	JP9	JP4 nižih 4 pina	JP4 viših 4 pina	JP5 nižih 4 pina	JP3 nižih 4 pina

Tabela 10. Šema spajanja konektora raster ploče i PCB-a.



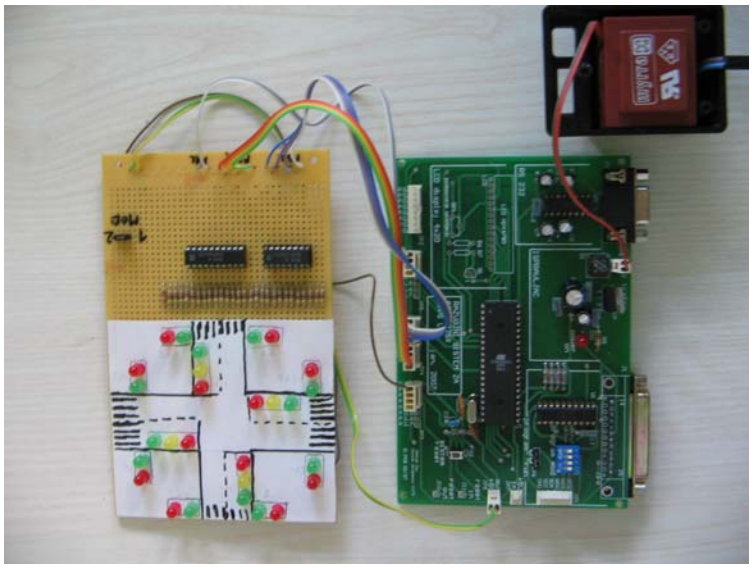
Slika 14. Blok šema sistema na raster ploči

Spisak upotrebljenih komponenata dat je u tabeli 11.

komponente	br. komada
AT89S8253	1
74LS244	1
MAX232	1
LM7805	1
QARTC 11.059MHz	1
GREC 1A-250V	1
DSUB9 konektor ženski	1
DSUB25 konektor muški	1
kabl muško-ženski 25 pinski paralelni	1
inline konektor 4 muško/žensko	4
inline konektor 2 muško/žensko	3
letvica sa trnovima	1
podnožja 40DIL	1
podnožja 20DIL	2
podnožja 16DIL	1
prekidač DIP4	1
Trafo 6V 500mA	1
LED žuta	4
LED crvena	13
LED zelena	12
elektrolitski kondenzator 470μF/25V	1
elektrolitski kondenzator 100μF/25V	1
elektrolitski kondenzator 10μF/50V	4
blok kondenzator 100nF/25V	5
blok kondenzator 33pF/25V	2
blok kondenzator 100pF/25V	1
otpornik 100k 1/4W	1
otpornik 4k7 1/4W	5
otpornik 1k 1/4W	1
otpornik 80 1/4W	1
raster ploča	1
74HCT367	1
74HC241	1
otpornik 480 1/4W	28
mikroprekidač 5mm	1
priključni kabl za 220V	1
kabl trakasti 9*25	1.5m
džamperi	2

Tabela 11. Spisak upotrebljenih komponenata.

Izgled realizovanog hardvera prikazan je na slici 15.



Slika 15. Izgled realizovanog hardvera

Testiranje

Testiranju sistema prethodilo je testiranje programa. Za tu svrhu korišćen je program *Keil uVision 3*. Nakon verifikacije ispravnosti napisanog programa iskorišćen je isti program *Keil uVision* za prevođenje C-koda u asemblerski, odnosno za kreiranje HEX fajla za programiranje kontrolera.

Potom je uz pomoć programa *ISP Programmer* generisani kôd prenešen u mikrokontroler. Rad sistema testiran je na realizovanom hardveru. Testirana su oba moda rada semafora, kao i rad nakon promene vremenskih parametara semafora. Testiranjem je utvrđeno da je projektni zadatak uspešno realizovan.

Zaključak

Semaforski sistem je analiziran i detaljno razrađen. Postavljeni su bili jasni softverski i hardverski ciljevi. Uz dalju analizu realizovani su hardver i softver. U toku rada uočene su manje rutinske greške koje su uspešno otklonjene. Nakon učitavanja programa u mikrokontroler izvršeno je testiranje celokupnog semaforskog sistema. Testiranje je prošlo uspešno tako da se može zaključiti da je uprkos manjim poteškoćama uspešno realizovan sistem.

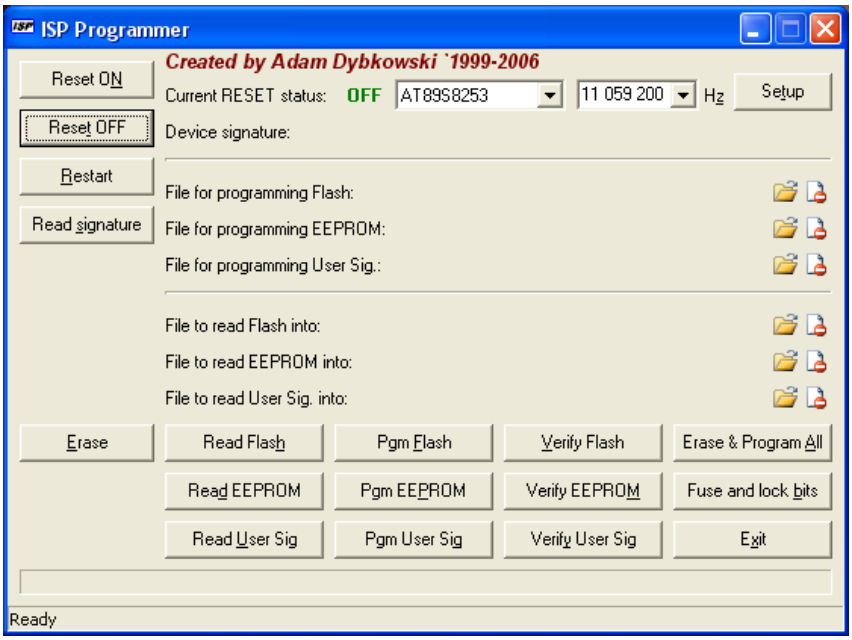
Moguća su dalja dorađivanja na ovu temu kao na primer kreiranje većeg semaforskog sistema od četiri, pet raskrsnica koje bi bile povezane. Ovaj veći semaforski sistem bi bio povezan tako što bi mikrokontroleri mogli da komuniciraju radi poboljšanja efikasnosti regulisanja saobraćaja. Tako bi npr. otkazivanjem rada jedog semaforskog sistema na raskrsnici mogao saobraćaj ka njoj da se redukuje.

Laboratorijska vežba za studente

Studenti bi trebalo da pročitaju prethodnu dokumentaciju radi pripreme za laboratorijsku vežbu. Njihov rad se svodi na testiranje semaforskog sistema i softversku modifikaciju rada semafora.

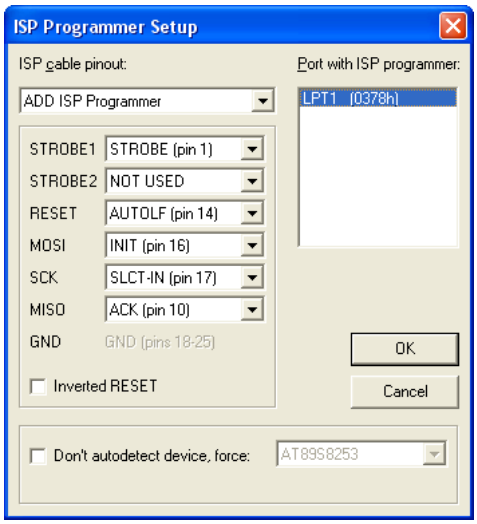
Prvo treba da provere rad semafora učitavanjem originalnog programskog kôda u mikrokontroler. To se radi u nekoliko koraka:

1. Prebaciti džamper (JP6) na PCB ploči u položaj za programiranje. Položaj je naznačen na pločici.
2. Prebaciti prekidače (SW-DIP4) koji su smešteni pored pomenutog džampera u stanje uključeno (on). Stanje on je napisano na prekidaču.
3. Povezati priloženim kablom PCB ploču sa paralelnim portom računara i priključiti napajanje.
4. Pokrenuti program *ISP Programmer*. Pojaviće grafički interfejs prikazan na Slici 16.



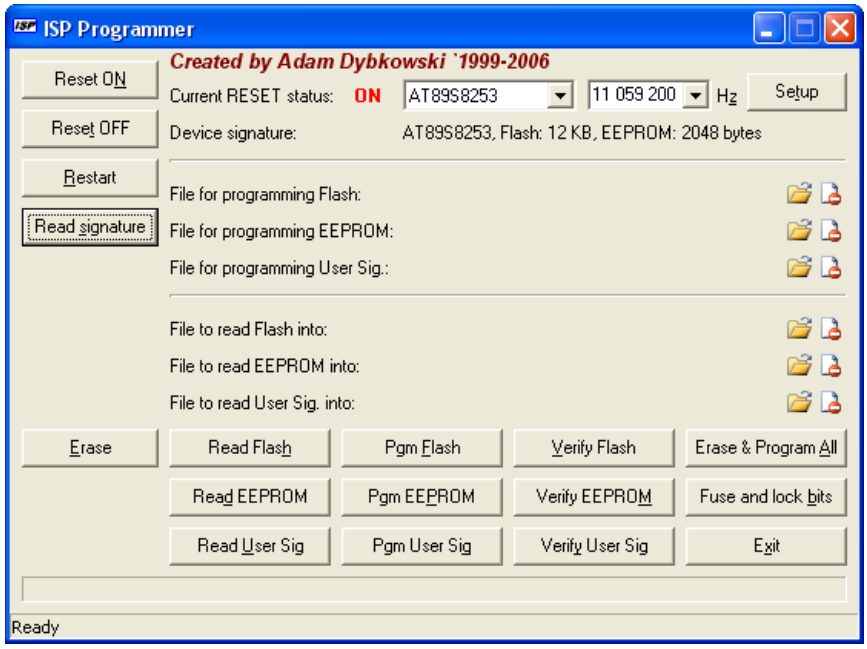
Slika 16. Grafički interfejs programa *ISP Programmer*.

5. Proveriti da li je dobro setovan program klikom na dugme Setup. Treba da se pojavi prozor sa slike 17. Ukoliko nije nešto dobro podešeno treba prepraviti. Zatvoriti prozor klikom na OK.



Slika 17. Prozor za setovanje pinova pri komunikaciji.

6. U glavnom prozoru u redu gde piše ”File for programming Flash:” kliknuti na ikonicu foldera. Pronaći odgovarajući HEX fajl po direktivi asistenta nadležnog za laboratorijsku vežbu. Otvoriti HEX fajl.
7. Kliknuti na „Read signature“. Ukoliko je sve dobro povezano trebao bi sada prozor da izgleda kao na slici 18. Ukoliko se javi UNKNOWN za “Device signature” treba proveriti da li je sve dobro povezano i pokušati ponovo. Moguće su poteškoće u komunikaciji ukoliko konektori kablova nisu dobro pričvršćeni.



Slika 18. Izgled glavnog prozora nakon prepoznavanja mikrokontrolera.

8. Kliknuti na dugme „Erase&ProgramAll“.
9. Nakon što je program uspešno upisan u flesh mikrokontrolera on se može isčitati specificiranjem ime fajla u koji će da se smešta ono što je isčitano i klikom na dugme „Read Flash“.

10. Isključiti napajanje PCB pločice i skloniti kabl za povezivanje sa računarom.
11. Prebaciti prekidače (SW-DIP4) u položaj *isključeno*.
12. Prebaciti džamper (JP6) u položaj *run*. Položaj naznačen na PCB ploči.
13. Povezati kablove od raster ploče sa PCB pločom prema oznakama. U slučaju da nema oznaka pitati asistenta.
14. Priključiti napajanje.
15. Semafor bi trebao da počne sa radom. Ukoliko je potrebno, pritisnuti taster za reset.
16. Promeniti položaj džampera na raster ploči radi provere rada drugog moda.

Nakon što se studenti uvere u ispravnost rada semaforskog sistema mogu, korišćenjem programa *Keil uVision 3* da otvore studentsku verziju originalnog programa semaforskog sistema po instrukcijama asistenta. Menjanjem vremena trajanja stanja semafora u zaglavlju .c fajla, sledećih par redova koda:

```
#define t1 2 // setovanje vremena u sekundama za sve crveno

#define t2 1 // setovanje vremena u sekundama za prelazni rezim zuto pre
zelenog

#define t3 8 // setovanje vremena u sekundama za zeleno pravac 1

#define t4 2 // setovanje vremena u sekundama za prelazni rezim zuto pre
crvenog

#define t5 8 // setovanje vremena u sekundama za zeleno pravac 2

#define t6 0.5 // setovanje vremena u sekundama      za trepcuce zuto off

#define t7 0.5 // setovanje vremena u sekundama za trepcuce zuto on
```

Studenti mogu podesiti prioritete smerova na raskrsnici tako da vreme trajanja zelenog svetla u jednom pravcu traje duže od drugog ili menjati frekvenciju treptanja žutog svetla.

Nakon odgovarajuće izmene programa treba kliknuti na dugme *build*. Proveriti da li je u opcijama "Options for target 1„ na podopciji „output" štiklirana opcija Create HEX file. Nakon toga ponoviti sekvencu učitavanja programa, tj. HEX fajla u flash memoriju mikrokontrolera i proveriti rad izmenjenog sistema. Po nalogu asistenta napisati odgovarajući kratak izveštaj za laboratorijsku vežbu.

Izmeniti program tako da pre isključenja semafora za neki pravac na semaforu za vozila prvo triput trepne zeleno svetlo pre nego što se upali žuto. Nacrtati odgovarajući ASM dijagram i priložiti ga uz izveštaj za laboratorijsku vežbu. Studenti uz dozvolu asistenta mogu pogledati apendix C u kome je priložen program koji ovo obavlja.

Prilog A

```
#include <reg52.h>

#define t1 2 // setovanje vremena u sekundama za sve crveno
#define t2 1 // setovanje vremena u sekundama za prelazni rezim zuto pre zelenog
#define t3 8 // setovanje vremena u sekundama za zeleno pravac 1
#define t4 2 // setovanje vremena u sekundama za prelazni rezim zuto pre crvenog
#define t5 8 // setovanje vremena u sekundama za zeleno pravac 2
#define t6 0.5 // setovanje vremena u sekundama      za trepcuce zuto off
#define t7 0.5 // setovanje vremena u sekundama za trepcuce zuto on
sfr WDTCON = 0xA7;

void sleep_for_50ms(void)
{
        TMOD &= 0xF0; // setovanje moda rada tajmera 0
        TMOD |= 0x01; // setovanje moda rada tajmera 0
        TH0 = 0x3C; // upis 8 MSB bita broja 15536 u viši registr tajmera 0
        TL0 = 0xB0; // upis 8 LSB bita broja 15536 u niži registr tajmera 0

        ET0 = 1; //dozvola interapta tajmera 0
        EA=1; //globalna dozvola interapta

        TF0 = 0; //brisanje overflow-a
        TR0 = 1; //pustanje u rad brojaca

        PCON |= 0x01; //stavljanje procesora u idle stanje radi ustede energije

        WDTCON=0x6B; //resetovanje watch dog tajmera
}
void sleep_for(unsigned char time)
{
        unsigned char i=0;
```

```
        for(;i<time;)
        {
                i++;
                sleep_for_50ms();
        }
}

void wait_for(float time)
{
        time *=20;
        sleep_for((unsigned char) time);
}

void intrpt() interrupt 1
{
}

void main()
{
        unsigned char stanje=1;
        P3=74;
        P1=2;
        P2|=1;
        while(1)
        {
                switch (stanje)
                {
                        case 0:
                        {
                                P3=0;
                                P1=0;
                                P2|=1;
                                if ((P2&0x01)==0) stanje=1;
                                else stanje=9;
                                wait_for(t6);
```

```
            break;
        }
        case 1:
        {
            P3=74;
            P1=2;
            P2|=1;
            if ((P2&0x01)==1) stanje=0;
            else stanje=2;
            wait_for(t1);
            break;
        }
        case 2:
        {
            P3=106;
            P1=2;
            stanje=3;
            wait_for(t2);
            break;
        }
        case 3:
        {
            P3=25;
            P1=2;
            stanje=4;
            wait_for(t3);
            break;
        }
        case 4:
        {
            P3=42;
            P1=2;
            stanje=5;
            wait_for(t4);
            break;
```

```
}
case 5:
{
        P3=74;
        P1=2;
        P2|=1;
        if ((P2&0x01)==1) stanje=0;
        else stanje=6;
        wait_for(t1);
        break;
}
case 6:
{
        P3=74;
        P1=3;
        stanje=7;
        wait_for(t2);
        break;
}
case 7:
{
        P3=198;
        P1=0;
        stanje=8;
        wait_for(t5);
        break;
}
case 8:
{
        P3=74;
        P1=1;
        stanje=1;
        wait_for(t4);
        break;
}
```

```
                case 9:
                {
                        P3=32;
                        P1=1;
                        stanje=0;
                        wait_for(t7);
                        break;
                }
                default:
                {
                        P3=74;
                        P1=6;
                        P2|=1;
                        if ((P2&0x01)==1) stanje=0;
                        else stanje=2;
                        wait_for(t1);
                        break;
                }
        }
    }
}
```

Prilog B

```
#include <reg52.h>

#define t1 2 // setovanje vremena u sekundama za sve crveno
#define t2 1 // setovanje vremena u sekundama za prelazni rezim zuto pre zelenog
#define t3 8 // setovanje vremena u sekundama za zeleno pravac 1
#define t4 2 // setovanje vremena u sekundama za prelazni rezim zuto pre crvenog
#define t5 8 // setovanje vremena u sekundama za zeleno pravac 2
#define t6 0.5 // setovanje vremena u sekundama      za trepcuce zuto off
#define t7 0.5 // setovanje vremena u sekundama za trepcuce zuto on
sfr WDTCON = 0xA7;

void sleep_for_50ms(void)
{
        //setovanje tajmera 0 za merenje 50ms

        //setovanje moda tajmera
        TMOD &= 0xF0;
        TMOD |= 0x01;

        ET0 = 1; //dozvola interapta tajmera 0
        EA=1; //globalna dozvola interapta

        //upisivanje pocetne vrednosti brojanja
        TH0 = 0x3C;
        TL0 = 0xB0;

        TF0 = 0; //brisanje overflow-a

        TR0 = 1; //pustanje u rad brojaca

        PCON |= 0x01; //stavljanje procesora u idle stanje radi ustede energije
```

```
        WDTCON=0x6B; //resetovanje watch dog tajmera
}

void sleep_for(unsigned char time)
{
        unsigned char i=0;
        for(;i<time;)
        {
                i++;
                sleep_for_50ms();
        }
}

void wait_for(float time)
{
        time *=20;
        sleep_for((unsigned char) time);
}

void intrpt() interrupt 1
{
}
void main()
{
        WDTCON=0x6B; //setovanje watch dog tajmera
        P3=74;
        P1=2;
        P2|=1;
        while(1)
        {
 mod1:          P3=74;
                P1=2;
                wait_for(t1);
                P2|=1;
                if ((P2&0x01)==1) goto mod2;
```

```
                P3=106;
                //P1=2;
                wait_for(t2);
                P3=25;
                //P1=2;
                wait_for(t3);
                P3=42;
                //P1=2;
                wait_for(t4);
                P3=74;
                //P1=2;
                wait_for(t1);
                P2|=1;
                if ((P2&0x01)==1) goto mod2;
                //P3=74;
                P1=3;
                wait_for(t2);
                P3=198;
                P1=0;
                wait_for(t5);
                P3=74;
                P1=1;
                wait_for(t4);
                goto mod1;
mod2:           P3=0;
                P1=0;
                wait_for(t6);
                P2|=1;
                if ((P2&0x01)==0) goto mod1;
                P3=32;
                P1=1;
                wait_for(t7);
                goto mod2;
        }
}
```

Prilog C

```
#include <reg52.h>

#define t1 2 // setovanje vremena u sekundama za sve crveno
#define t2 1 // setovanje vremena u sekundama za prelazni rezim zuto pre zelenog
#define t3 8 // setovanje vremena u sekundama za zeleno pravac 1
#define t4 2 // setovanje vremena u sekundama za prelazni rezim zuto pre crvenog
#define t5 8 // setovanje vremena u sekundama za zeleno pravac 2
#define t6 0.5 // setovanje vremena u sekundama      za trepcuce zuto off
#define t7 0.5 // setovanje vremena u sekundama za trepcuce zuto on
#define t8 0.25 // setovanje vremena u sekundama za trepcuce zeleno pre iskljucenja
sfr WDTCON = 0xA7;

void sleep_for_50ms(void)
{
        //setovanje tajmera 0 za merenje 50ms

        //setovanje moda tajmera
        TMOD &= 0xF0;
        TMOD |= 0x01;

        ET0 = 1; //dozvola interapta tajmera 0
        EA=1; //globalna dozvola interapta

        //upisivanje pocetne vrednosti brojanja
        TH0 = 0x3C;
        TL0 = 0xB0;

        TF0 = 0; //brisanje overflow-a

        TR0 = 1; //pustanje u rad brojaca
```

```
        PCON |= 0x01; //stavljanje procesora u idle stanje radi ustede energije
        WDTCON=0x6B; //resetovanje watch dog tajmera
}

void sleep_for(unsigned char time)
{
        unsigned char i=0;
        for(;i<time;)
        {
                i++;
                sleep_for_50ms();
        }
}

void wait_for(float time)
{
        time *=20;
        sleep_for((unsigned char) time);
}

void intrpt() interrupt 1
{
}
void main()
{
        WDTCON=0x6B; //setovanje watch dog tajmera
        P3=74;
        P1=2;
        P2|=1;
        while(1)
        {
 mod1:          P3=74;
                P1=2;
                wait_for(t1);
                P2|=1;
```

```
                if ((P2&0x01)==1) goto mod2;
                P3=106;
                //P1=2;
                wait_for(t2);
                P3=25;
                //P1=2;
                wait_for(t3);
// trepce zeleno
                P3=9;
                //P1=2;
                wait_for(t8);
                P3=25;
                //P1=2;
                wait_for(t8);
                P3=9;
                //P1=2;
                wait_for(t8);
                P3=25;
                //P1=2;
                wait_for(t8);
                P3=9;
                //P1=2;
                wait_for(t8);
                P3=25;
                //P1=2;
                wait_for(t8);

                P3=42;
                //P1=2;
                wait_for(t4);
                P3=74;
                //P1=2;
                wait_for(t1);
                P2|=1;
                if ((P2&0x01)==1) goto mod2;
```

```
                    //P3=74;
                    P1=3;
                    wait_for(t2);
                    P3=198;
                    P1=0;
                    wait_for(t5);
// trepce zeleno
                    P3=70;
                    //P1=0;
                    wait_for(t8);
                    P3=198;
                    //P1=0;
                    wait_for(t8);
                    P3=70;
                    //P1=0;
                    wait_for(t8);
                    P3=198;
                    //P1=0;
                    wait_for(t8);
                    P3=70;
                    //P1=0;
                    wait_for(t8);
                    P3=198;
                    //P1=0;
                    wait_for(t8);

                    P3=74;
                    P1=1;
                    wait_for(t4);
                    goto mod1;
 mod2:              P3=0;
                    P1=0;
                    wait_for(t6);
                    P2|=1;
                    if ((P2&0x01)==0) goto mod1;
```

```
                P3=32;
                P1=1;
                wait_for(t7);
                goto mod2;
        }
}
```

Prilog D

```
  126: ?C_STARTUP:    LJMP    STARTUP1
  127:
  128:                RSEG    ?C_C51STARTUP
  129:
  130: STARTUP1:
  131:
  132: IF IDATALEN <> 0
C:0x0000    020390   LJMP    STARTUP1(C:0390)
   45: void intrpt() interrupt 1
   46: {
   47: }
   48:
C:0x0003    32       RETI
C:0x0004    00       NOP
C:0x0005    00       NOP
C:0x0006    00       NOP
C:0x0007    00       NOP
C:0x0008    00       NOP
C:0x0009    00       NOP
C:0x000A    00       NOP
C:0x000B    020003   LJMP    intrpt(C:0003)
              C?FPMUL:
C:0x000E    EC       MOV     A,R4
C:0x000F    4D       ORL     A,R5
C:0x0010    6011     JZ      C:0023
C:0x0012    E8       MOV     A,R0
C:0x0013    49       ORL     A,R1
C:0x0014    7017     JNZ     C:002D
C:0x0016    ED       MOV     A,R5
C:0x0017    33       RLC     A
C:0x0018    EC       MOV     A,R4
C:0x0019    33       RLC     A
C:0x001A    04       INC     A
C:0x001B    600D     JZ      C:002A
C:0x001D    E4       CLR     A
C:0x001E    FC       MOV     R4,A
C:0x001F    FF       MOV     R7,A
```

```
C:0x0020    FE      MOV     R6,A
C:0x0021    FD      MOV     R5,A
C:0x0022    22      RET
C:0x0023    E9      MOV     A,R1
C:0x0024    33      RLC     A
C:0x0025    E8      MOV     A,R0
C:0x0026    33      RLC     A
C:0x0027    04      INC     A
C:0x0028    70F8    JNZ     C:0022
C:0x002A    020180  LJMP    C:0180
C:0x002D    12014B  LCALL   C:014B
C:0x0030    58      ANL     A,R0
C:0x0031    04      INC     A
C:0x0032    6009    JZ      C:003D
C:0x0034    E4      CLR     A
C:0x0035    CC      XCH     A,R4
C:0x0036    2481    ADD     A,#SP(0x81)
C:0x0038    5006    JNC     C:0040
C:0x003A    28      ADD     A,R0
C:0x003B    5009    JNC     C:0046
C:0x003D    02018A  LJMP    C:018A
C:0x0040    28      ADD     A,R0
C:0x0041    4003    JC      C:0046
C:0x0043    020187  LJMP    C:0187
C:0x0046    C0E0    PUSH    ACC(0xE0)
C:0x0048    EB      MOV     A,R3
C:0x0049    4A      ORL     A,R2
C:0x004A    7044    JNZ     C:0090
C:0x004C    B98006  CJNE    R1,#P0(0x80),C:0055
C:0x004F    D0E0    POP     ACC(0xE0)
C:0x0051    FB      MOV     R3,A
C:0x0052    020176  LJMP    C:0176
C:0x0055    EF      MOV     A,R7
C:0x0056    4E      ORL     A,R6
C:0x0057    701C    JNZ     C:0075
C:0x0059    BD8008  CJNE    R5,#P0(0x80),C:0064
C:0x005C    EB      MOV     A,R3
C:0x005D    FF      MOV     R7,A
C:0x005E    EA      MOV     A,R2
C:0x005F    FE      MOV     R6,A
```

```
C:0x0060    E9      MOV     A,R1
C:0x0061    FD      MOV     R5,A
C:0x0062    80EB    SJMP    C:004F
C:0x0064    E9      MOV     A,R1
C:0x0065    8DF0    MOV     B(0xF0),R5
C:0x0067    A4      MUL     AB
C:0x0068    FE      MOV     R6,A
C:0x0069    E5F0    MOV     A,B(0xF0)
C:0x006B    0200F7  LJMP    C:00F7
C:0x006E    E9      MOV     A,R1
C:0x006F    CD      XCH     A,R5
C:0x0070    F9      MOV     R1,A
C:0x0071    EA      MOV     A,R2
C:0x0072    FE      MOV     R6,A
C:0x0073    EB      MOV     A,R3
C:0x0074    FF      MOV     R7,A
C:0x0075    EF      MOV     A,R7
C:0x0076    89F0    MOV     B(0xF0),R1
C:0x0078    A4      MUL     AB
C:0x0079    FC      MOV     R4,A
C:0x007A    E5F0    MOV     A,B(0xF0)
C:0x007C    CE      XCH     A,R6
C:0x007D    89F0    MOV     B(0xF0),R1
C:0x007F    A4      MUL     AB
C:0x0080    2E      ADD     A,R6
C:0x0081    FF      MOV     R7,A
C:0x0082    E4      CLR     A
C:0x0083    35F0    ADDC    A,B(0xF0)
C:0x0085    CD      XCH     A,R5
C:0x0086    89F0    MOV     B(0xF0),R1
C:0x0088    A4      MUL     AB
C:0x0089    2D      ADD     A,R5
C:0x008A    FE      MOV     R6,A
C:0x008B    E4      CLR     A
C:0x008C    35F0    ADDC    A,B(0xF0)
C:0x008E    8067    SJMP    C:00F7
C:0x0090    EF      MOV     A,R7
C:0x0091    4E      ORL     A,R6
C:0x0092    7005    JNZ     C:0099
C:0x0094    BD80D7  CJNE    R5,#P0(0x80),C:006E
```

```
C:0x0097    80C3     SJMP     C:005C
C:0x0099    EF       MOV      A,R7
C:0x009A    8BF0     MOV      B(0xF0),R3
C:0x009C    A4       MUL      AB
C:0x009D    ACF0     MOV      R4,B(0xF0)
C:0x009F    EE       MOV      A,R6
C:0x00A0    8BF0     MOV      B(0xF0),R3
C:0x00A2    A4       MUL      AB
C:0x00A3    2C       ADD      A,R4
C:0x00A4    FC       MOV      R4,A
C:0x00A5    E4       CLR      A
C:0x00A6    35F0     ADDC     A,B(0xF0)
C:0x00A8    F8       MOV      R0,A
C:0x00A9    EF       MOV      A,R7
C:0x00AA    8AF0     MOV      B(0xF0),R2
C:0x00AC    A4       MUL      AB
C:0x00AD    2C       ADD      A,R4
C:0x00AE    E5F0     MOV      A,B(0xF0)
C:0x00B0    38       ADDC     A,R0
C:0x00B1    FC       MOV      R4,A
C:0x00B2    E4       CLR      A
C:0x00B3    33       RLC      A
C:0x00B4    CB       XCH      A,R3
C:0x00B5    8DF0     MOV      B(0xF0),R5
C:0x00B7    A4       MUL      AB
C:0x00B8    2C       ADD      A,R4
C:0x00B9    FC       MOV      R4,A
C:0x00BA    E5F0     MOV      A,B(0xF0)
C:0x00BC    3B       ADDC     A,R3
C:0x00BD    F8       MOV      R0,A
C:0x00BE    EE       MOV      A,R6
C:0x00BF    8AF0     MOV      B(0xF0),R2
C:0x00C1    A4       MUL      AB
C:0x00C2    2C       ADD      A,R4
C:0x00C3    FC       MOV      R4,A
C:0x00C4    E5F0     MOV      A,B(0xF0)
C:0x00C6    38       ADDC     A,R0
C:0x00C7    F8       MOV      R0,A
C:0x00C8    E4       CLR      A
C:0x00C9    33       RLC      A
```

```
C:0x00CA    CF      XCH     A,R7
C:0x00CB    89F0    MOV     B(0xF0),R1
C:0x00CD    A4      MUL     AB
C:0x00CE    2C      ADD     A,R4
C:0x00CF    FC      MOV     R4,A
C:0x00D0    E5F0    MOV     A,B(0xF0)
C:0x00D2    38      ADDC    A,R0
C:0x00D3    CF      XCH     A,R7
C:0x00D4    3400    ADDC    A,#0x00
C:0x00D6    CE      XCH     A,R6
C:0x00D7    89F0    MOV     B(0xF0),R1
C:0x00D9    A4      MUL     AB
C:0x00DA    2F      ADD     A,R7
C:0x00DB    FF      MOV     R7,A
C:0x00DC    E5F0    MOV     A,B(0xF0)
C:0x00DE    3E      ADDC    A,R6
C:0x00DF    FE      MOV     R6,A
C:0x00E0    E4      CLR     A
C:0x00E1    33      RLC     A
C:0x00E2    C9      XCH     A,R1
C:0x00E3    8DF0    MOV     B(0xF0),R5
C:0x00E5    A4      MUL     AB
C:0x00E6    2E      ADD     A,R6
C:0x00E7    FE      MOV     R6,A
C:0x00E8    E5F0    MOV     A,B(0xF0)
C:0x00EA    39      ADDC    A,R1
C:0x00EB    CD      XCH     A,R5
C:0x00EC    8AF0    MOV     B(0xF0),R2
C:0x00EE    A4      MUL     AB
C:0x00EF    2F      ADD     A,R7
C:0x00F0    FF      MOV     R7,A
C:0x00F1    E5F0    MOV     A,B(0xF0)
C:0x00F3    3E      ADDC    A,R6
C:0x00F4    FE      MOV     R6,A
C:0x00F5    E4      CLR     A
C:0x00F6    3D      ADDC    A,R5
C:0x00F7    FD      MOV     R5,A
C:0x00F8    33      RLC     A
C:0x00F9    D0E0    POP     ACC(0xE0)
C:0x00FB    FB      MOV     R3,A
```

```
C:0x00FC    5007    JNC     C:0105
C:0x00FE    0B      INC     R3
C:0x00FF    BB000F  CJNE    R3,#0x00,C:0111
C:0x0102    02018A  LJMP    C:018A
C:0x0105    EC      MOV     A,R4
C:0x0106    2C      ADD     A,R4
C:0x0107    FC      MOV     R4,A
C:0x0108    EF      MOV     A,R7
C:0x0109    33      RLC     A
C:0x010A    FF      MOV     R7,A
C:0x010B    EE      MOV     A,R6
C:0x010C    33      RLC     A
C:0x010D    FE      MOV     R6,A
C:0x010E    ED      MOV     A,R5
C:0x010F    33      RLC     A
C:0x0110    FD      MOV     R5,A
C:0x0111    020162  LJMP    C:0162
            C?CASTF:
C:0x0114    ED      MOV     A,R5
C:0x0115    D2E7    SETB    0xE0.7
C:0x0117    CD      XCH     A,R5
C:0x0118    33      RLC     A
C:0x0119    EC      MOV     A,R4
C:0x011A    33      RLC     A
C:0x011B    92D5    MOV     F0(0xD0.5),C
C:0x011D    2481    ADD     A,#SP(0x81)
C:0x011F    4006    JC      C:0127
C:0x0121    E4      CLR     A
C:0x0122    FF      MOV     R7,A
C:0x0123    FE      MOV     R6,A
C:0x0124    FD      MOV     R5,A
C:0x0125    FC      MOV     R4,A
C:0x0126    22      RET
C:0x0127    FC      MOV     R4,A
C:0x0128    E4      CLR     A
C:0x0129    CF      XCH     A,R7
C:0x012A    CE      XCH     A,R6
C:0x012B    CD      XCH     A,R5
C:0x012C    CC      XCH     A,R4
C:0x012D    24E0    ADD     A,#ACC(0xE0)
```

```
C:0x012F    5011    JNC     C:0142
C:0x0131    74FF    MOV     A,#0xFF
C:0x0133    80ED    SJMP    C:0122
C:0x0135    C3      CLR     C
C:0x0136    CC      XCH     A,R4
C:0x0137    13      RRC     A
C:0x0138    CC      XCH     A,R4
C:0x0139    CD      XCH     A,R5
C:0x013A    13      RRC     A
C:0x013B    CD      XCH     A,R5
C:0x013C    CE      XCH     A,R6
C:0x013D    13      RRC     A
C:0x013E    CE      XCH     A,R6
C:0x013F    CF      XCH     A,R7
C:0x0140    13      RRC     A
C:0x0141    CF      XCH     A,R7
C:0x0142    04      INC     A
C:0x0143    70F0    JNZ     C:0135
C:0x0145    30D5DE  JNB     F0(0xD0.5),C:0126
C:0x0148    020195  LJMP    C?LNEG(C:0195)
C:0x014B    E9      MOV     A,R1
C:0x014C    D2E7    SETB    0xE0.7
C:0x014E    C9      XCH     A,R1
C:0x014F    33      RLC     A
C:0x0150    E8      MOV     A,R0
C:0x0151    33      RLC     A
C:0x0152    F8      MOV     R0,A
C:0x0153    92D5    MOV     F0(0xD0.5),C
C:0x0155    ED      MOV     A,R5
C:0x0156    D2E7    SETB    0xE0.7
C:0x0158    CD      XCH     A,R5
C:0x0159    33      RLC     A
C:0x015A    EC      MOV     A,R4
C:0x015B    33      RLC     A
C:0x015C    FC      MOV     R4,A
C:0x015D    5002    JNC     C:0161
C:0x015F    B2D5    CPL     F0(0xD0.5)
C:0x0161    22      RET
C:0x0162    EC      MOV     A,R4
C:0x0163    30E710  JNB     0xE0.7,C:0176
```

```
C:0x0166    0F      INC     R7
C:0x0167    BF000C  CJNE    R7,#0x00,C:0176
C:0x016A    0E      INC     R6
C:0x016B    BE0008  CJNE    R6,#0x00,C:0176
C:0x016E    0D      INC     R5
C:0x016F    BD0004  CJNE    R5,#0x00,C:0176
C:0x0172    0B      INC     R3
C:0x0173    EB      MOV     A,R3
C:0x0174    6014    JZ      C:018A
C:0x0176    A2D5    MOV     C,F0(0xD0.5)
C:0x0178    EB      MOV     A,R3
C:0x0179    13      RRC     A
C:0x017A    FC      MOV     R4,A
C:0x017B    ED      MOV     A,R5
C:0x017C    92E7    MOV     0xE0.7,C
C:0x017E    FD      MOV     R5,A
C:0x017F    22      RET
C:0x0180    74FF    MOV     A,#0xFF
C:0x0182    FC      MOV     R4,A
C:0x0183    FD      MOV     R5,A
C:0x0184    FE      MOV     R6,A
C:0x0185    FF      MOV     R7,A
C:0x0186    22      RET
C:0x0187    E4      CLR     A
C:0x0188    80F8    SJMP    C:0182
C:0x018A    A2D5    MOV     C,F0(0xD0.5)
C:0x018C    74FF    MOV     A,#0xFF
C:0x018E    13      RRC     A
C:0x018F    FC      MOV     R4,A
C:0x0190    7D80    MOV     R5,#P0(0x80)
C:0x0192    E4      CLR     A
C:0x0193    80EF    SJMP    C:0184
          C?LNEG:
C:0x0195    C3      CLR     C
C:0x0196    E4      CLR     A
C:0x0197    9F      SUBB    A,R7
C:0x0198    FF      MOV     R7,A
C:0x0199    E4      CLR     A
C:0x019A    9E      SUBB    A,R6
C:0x019B    FE      MOV     R6,A
```

```
C:0x019C    E4      CLR     A
C:0x019D    9D      SUBB    A,R5
C:0x019E    FD      MOV     R5,A
C:0x019F    E4      CLR     A
C:0x01A0    9C      SUBB    A,R4
C:0x01A1    FC      MOV     R4,A
C:0x01A2    22      RET
            C?CCASE:
C:0x01A3    D083    POP     DPH(0x83)
C:0x01A5    D082    POP     DPL(0x82)
C:0x01A7    F8      MOV     R0,A
C:0x01A8    E4      CLR     A
C:0x01A9    93      MOVC    A,@A+DPTR
C:0x01AA    7012    JNZ     C:01BE
C:0x01AC    7401    MOV     A,#0x01
C:0x01AE    93      MOVC    A,@A+DPTR
C:0x01AF    700D    JNZ     C:01BE
C:0x01B1    A3      INC     DPTR
C:0x01B2    A3      INC     DPTR
C:0x01B3    93      MOVC    A,@A+DPTR
C:0x01B4    F8      MOV     R0,A
C:0x01B5    7401    MOV     A,#0x01
C:0x01B7    93      MOVC    A,@A+DPTR
C:0x01B8    F582    MOV     DPL(0x82),A
C:0x01BA    8883    MOV     DPH(0x83),R0
C:0x01BC    E4      CLR     A
C:0x01BD    73      JMP     @A+DPTR
C:0x01BE    7402    MOV     A,#0x02
C:0x01C0    93      MOVC    A,@A+DPTR
C:0x01C1    68      XRL     A,R0
C:0x01C2    60EF    JZ      C:01B3
C:0x01C4    A3      INC     DPTR
C:0x01C5    A3      INC     DPTR
C:0x01C6    A3      INC     DPTR
C:0x01C7    80DF    SJMP    C:01A8
  49: void main()
  50: {
  51:       unsigned char stanje=1;
C:0x01C9    750E01  MOV     0x0E,#0x01
  52:       P3=74;
```

```
C:0x01CC    75B04A   MOV      P3(0xB0),#0x4A
   53:        P1=2;
C:0x01CF    759002   MOV      P1(0x90),#0x02
   54:        P2|=1;
C:0x01D2    43A001   ORL      PPAGE_SFR(0xA0),#0x01
   55:        while(1)
   56:        {
   57:              switch (stanje)
C:0x01D5    E50E     MOV      A,0x0E
C:0x01D7    1201A3   LCALL    C?CCASE(C:01A3)
C:0x01DA    01FC     AJMP     C:00FC
C:0x01DC    00       NOP
C:0x01DD    021F01   LJMP     C:1F01
C:0x01E0    024202   LJMP     C:4202
C:0x01E3    025903   LJMP     C:5903
C:0x01E6    027004   LJMP     C:7004
C:0x01E9    028705   LJMP     C:8705
C:0x01EC    02AB06   LJMP     C:AB06
C:0x01EF    02C207   LJMP     C:C207
C:0x01F2    02D908   LJMP     C:D908
C:0x01F5    02F009   LJMP     C:F009
C:0x01F8    00       NOP
C:0x01F9    00       NOP
C:0x01FA    03       RR       A
C:0x01FB    07       INC      @R1
   58:              {
   59:                    case 0:
   60:                    {
   61:                          P3=0;
C:0x01FC    75B000   MOV      P3(0xB0),#0x00
   62:                          P1=0;
C:0x01FF    759000   MOV      P1(0x90),#0x00
   63:                          P2|=1;
C:0x0202    43A001   ORL      PPAGE_SFR(0xA0),#0x01
   64:                          if ((P2&0x01)==0) stanje=1;
C:0x0205    E5A0     MOV      A,PPAGE_SFR(0xA0)
C:0x0207    20E005   JB       0xE0.0,C:020F
C:0x020A    750E01   MOV      0x0E,#0x01
C:0x020D    8003     SJMP     C:0212
   65:                          else stanje=9;
```

```
C:0x020F    750E09   MOV      0x0E,#0x09
   66:                       wait_for(t6);
C:0x0212    7F00     MOV      R7,#0x00
C:0x0214    7E00     MOV      R6,#0x00
C:0x0216    7D00     MOV      R5,#0x00
C:0x0218    7C3F     MOV      R4,#0x3F
C:0x021A    12032F   LCALL    wait_for(C:032F)
   67:                       break;
C:0x021D    80B6     SJMP     C:01D5
   68:                    }
   69:                    case 1:
   70:                    {
   71:                       P3=74;
C:0x021F    75B04A   MOV      P3(0xB0),#0x4A
   72:                       P1=2;
C:0x0222    759002   MOV      P1(0x90),#0x02
   73:                       P2|=1;
C:0x0225    43A001   ORL      PPAGE_SFR(0xA0),#0x01
   74:                       if ((P2&0x01)==1) stanje=0;
C:0x0228    E5A0     MOV      A,PPAGE_SFR(0xA0)
C:0x022A    30E005   JNB      0xE0.0,C:0232
C:0x022D    750E00   MOV      0x0E,#0x00
C:0x0230    8003     SJMP     C:0235
   75:                       else stanje=2;
C:0x0232    750E02   MOV      0x0E,#0x02
   76:                       wait_for(t1);
C:0x0235    7F00     MOV      R7,#0x00
C:0x0237    7E00     MOV      R6,#0x00
C:0x0239    7D00     MOV      R5,#0x00
C:0x023B    7C40     MOV      R4,#0x40
C:0x023D    12032F   LCALL    wait_for(C:032F)
   77:                       break;
C:0x0240    8093     SJMP     C:01D5
   78:                    }
   79:                    case 2:
   80:                    {
   81:                       P3=106;
C:0x0242    75B06A   MOV      P3(0xB0),#0x6A
   82:                       P1=2;
C:0x0245    759002   MOV      P1(0x90),#0x02
```

```
  83:                       stanje=3;
C:0x0248  750E03  MOV     0x0E,#0x03
  84:                       wait_for(t2);
C:0x024B  7F00    MOV     R7,#0x00
C:0x024D  7E00    MOV     R6,#0x00
C:0x024F  7D80    MOV     R5,#P0(0x80)
C:0x0251  7C3F    MOV     R4,#0x3F
C:0x0253  12032F  LCALL   wait_for(C:032F)
  85:                       break;
C:0x0256  0201D5  LJMP    C:01D5
  86:                   }
  87:                   case 3:
  88:                   {
  89:                       P3=25;
C:0x0259  75B019  MOV     P3(0xB0),#0x19
  90:                       P1=2;
C:0x025C  759002  MOV     P1(0x90),#0x02
  91:                       stanje=4;
C:0x025F  750E04  MOV     0x0E,#0x04
  92:                       wait_for(t3);
C:0x0262  7F00    MOV     R7,#0x00
C:0x0264  7E00    MOV     R6,#0x00
C:0x0266  7D00    MOV     R5,#0x00
C:0x0268  7C41    MOV     R4,#0x41
C:0x026A  12032F  LCALL   wait_for(C:032F)
  93:                       break;
C:0x026D  0201D5  LJMP    C:01D5
  94:                   }
  95:                   case 4:
  96:                   {
  97:                       P3=42;
C:0x0270  75B02A  MOV     P3(0xB0),#0x2A
  98:                       P1=2;
C:0x0273  759002  MOV     P1(0x90),#0x02
  99:                       stanje=5;
C:0x0276  750E05  MOV     0x0E,#0x05
 100:                       wait_for(t4);
C:0x0279  7F00    MOV     R7,#0x00
C:0x027B  7E00    MOV     R6,#0x00
C:0x027D  7D00    MOV     R5,#0x00
```

```
C:0x027F    7C40     MOV      R4,#0x40
C:0x0281    12032F   LCALL    wait_for(C:032F)
  101:                       break;
C:0x0284    0201D5   LJMP     C:01D5
  102:                 }
  103:                 case 5:
  104:                 {
  105:                       P3=74;
C:0x0287    75B04A   MOV      P3(0xB0),#0x4A
  106:                       P1=2;
C:0x028A    759002   MOV      P1(0x90),#0x02
  107:                       P2|=1;
C:0x028D    43A001   ORL      PPAGE_SFR(0xA0),#0x01
  108:                       if ((P2&0x01)==1) stanje=0;
C:0x0290    E5A0     MOV      A,PPAGE_SFR(0xA0)
C:0x0292    30E005   JNB      0xE0.0,C:029A
C:0x0295    750E00   MOV      0x0E,#0x00
C:0x0298    8003     SJMP     C:029D
  109:                       else stanje=6;
C:0x029A    750E06   MOV      0x0E,#0x06
  110:                       wait_for(t1);
C:0x029D    7F00     MOV      R7,#0x00
C:0x029F    7E00     MOV      R6,#0x00
C:0x02A1    7D00     MOV      R5,#0x00
C:0x02A3    7C40     MOV      R4,#0x40
C:0x02A5    12032F   LCALL    wait_for(C:032F)
  111:                       break;
C:0x02A8    0201D5   LJMP     C:01D5
  112:                 }
  113:                 case 6:
  114:                 {
  115:                       P3=74;
C:0x02AB    75B04A   MOV      P3(0xB0),#0x4A
  116:                       P1=3;
C:0x02AE    759003   MOV      P1(0x90),#0x03
  117:                       stanje=7;
C:0x02B1    750E07   MOV      0x0E,#0x07
  118:                       wait_for(t2);
C:0x02B4    7F00     MOV      R7,#0x00
C:0x02B6    7E00     MOV      R6,#0x00
```

```
C:0x02B8    7D80     MOV      R5,#P0(0x80)
C:0x02BA    7C3F     MOV      R4,#0x3F
C:0x02BC    12032F   LCALL    wait_for(C:032F)
  119:                       break;
C:0x02BF    0201D5   LJMP     C:01D5
  120:                 }
  121:                 case 7:
  122:                 {
  123:                       P3=198;
C:0x02C2    75B0C6   MOV      P3(0xB0),#0xC6
  124:                       P1=0;
C:0x02C5    759000   MOV      P1(0x90),#0x00
  125:                       stanje=8;
C:0x02C8    750E08   MOV      0x0E,#0x08
  126:                       wait_for(t5);
C:0x02CB    7F00     MOV      R7,#0x00
C:0x02CD    7E00     MOV      R6,#0x00
C:0x02CF    7D00     MOV      R5,#0x00
C:0x02D1    7C41     MOV      R4,#0x41
C:0x02D3    12032F   LCALL    wait_for(C:032F)
  127:                       break;
C:0x02D6    0201D5   LJMP     C:01D5
  128:                 }
  129:                 case 8:
  130:                 {
  131:                       P3=74;
C:0x02D9    75B04A   MOV      P3(0xB0),#0x4A
  132:                       P1=1;
C:0x02DC    759001   MOV      P1(0x90),#0x01
  133:                       stanje=1;
C:0x02DF    750E01   MOV      0x0E,#0x01
  134:                       wait_for(t4);
C:0x02E2    7F00     MOV      R7,#0x00
C:0x02E4    7E00     MOV      R6,#0x00
C:0x02E6    7D00     MOV      R5,#0x00
C:0x02E8    7C40     MOV      R4,#0x40
C:0x02EA    12032F   LCALL    wait_for(C:032F)
  135:                       break;
C:0x02ED    0201D5   LJMP     C:01D5
  136:                 }
```

```
  137:                  case 9:
  138:                  {
  139:                        P3=32;
C:0x02F0    75B020   MOV     P3(0xB0),#0x20
  140:                        P1=1;
C:0x02F3    759001   MOV     P1(0x90),#0x01
  141:                        stanje=0;
C:0x02F6    750E00   MOV     0x0E,#0x00
  142:                        wait_for(t7);
C:0x02F9    7F00     MOV     R7,#0x00
C:0x02FB    7E00     MOV     R6,#0x00
C:0x02FD    7D00     MOV     R5,#0x00
C:0x02FF    7C3F     MOV     R4,#0x3F
C:0x0301    12032F   LCALL   wait_for(C:032F)
  143:                        break;
C:0x0304    0201D5   LJMP    C:01D5
  144:                  }
  145:                  default:
  146:                  {
  147:                        P3=74;
C:0x0307    75B04A   MOV     P3(0xB0),#0x4A
  148:                        P1=6;
C:0x030A    759006   MOV     P1(0x90),#0x06
  149:                        P2|=1;
C:0x030D    43A001   ORL     PPAGE_SFR(0xA0),#0x01
  150:                        if ((P2&0x01)==1) stanje=0;
C:0x0310    E5A0     MOV     A,PPAGE_SFR(0xA0)
C:0x0312    30E005   JNB     0xE0.0,C:031A
C:0x0315    750E00   MOV     0x0E,#0x00
C:0x0318    8003     SJMP    C:031D
  151:                        else stanje=2;
C:0x031A    750E02   MOV     0x0E,#0x02
  152:                        wait_for(t1);
C:0x031D    7F00     MOV     R7,#0x00
C:0x031F    7E00     MOV     R6,#0x00
C:0x0321    7D00     MOV     R5,#0x00
C:0x0323    7C40     MOV     R4,#0x40
C:0x0325    12032F   LCALL   wait_for(C:032F)
  153:                        break;
C:0x0328    0201D5   LJMP    C:01D5
```

```
  154:                    }
  155:                }
  156:        }
C:0x032B    0201D5   LJMP     C:01D5
  157: }
C:0x032E    22       RET
   39: void wait_for(float time)
C:0x032F    8F0B     MOV      0x0B,R7
C:0x0331    8E0A     MOV      0x0A,R6
C:0x0333    8D09     MOV      0x09,R5
C:0x0335    8C08     MOV      0x08,R4
   40: {
   41:         time *=20;
C:0x0337    AF0B     MOV      R7,0x0B
C:0x0339    AE0A     MOV      R6,0x0A
C:0x033B    AD09     MOV      R5,0x09
C:0x033D    AC08     MOV      R4,0x08
C:0x033F    7B00     MOV      R3,#0x00
C:0x0341    7A00     MOV      R2,#0x00
C:0x0343    79A0     MOV      R1,#PPAGE_SFR(0xA0)
C:0x0345    7841     MOV      R0,#0x41
C:0x0347    12000E   LCALL    C?FPMUL(C:000E)
C:0x034A    8F0B     MOV      0x0B,R7
C:0x034C    8E0A     MOV      0x0A,R6
C:0x034E    8D09     MOV      0x09,R5
C:0x0350    8C08     MOV      0x08,R4
   42:         sleep_for((unsigned char) time);
C:0x0352    AF0B     MOV      R7,0x0B
C:0x0354    AE0A     MOV      R6,0x0A
C:0x0356    AD09     MOV      R5,0x09
C:0x0358    AC08     MOV      R4,0x08
C:0x035A    120114   LCALL    C?CASTF(C:0114)
C:0x035D    12037C   LCALL    sleep_for(C:037C)
   43: }
C:0x0360    22       RET
   12: void sleep_for_50ms(void)
   13: {
   14: TMOD &= 0xF0; // setovanje moda rada tajmera 0
C:0x0361    5389F0   ANL      TMOD(0x89),#B(0xF0)
   15: TMOD |= 0x01; // setovanje moda rada tajmera 0
```

```
C:0x0364    438901   ORL      TMOD(0x89),#0x01
   16: TH0 = 0x3C; // upis 8 MSB bita broja 15536 u viši registr tajmera 0
C:0x0367    758C3C   MOV      TH0(0x8C),#0x3C
   17: TL0 = 0xB0; // upis 8 LSB bita broja 15536 u niži registr tajmera 0
   18:
C:0x036A    758AB0   MOV      TL0(0x8A),#P3(0xB0)
   19: ET0 = 1; //dozvola interapta tajmera 0
C:0x036D    D2A9     SETB     ET0(0xA8.1)
   20: EA=1; //globalna dozvola interapta
   21:
C:0x036F    D2AF     SETB     EA(0xA8.7)
   22: TF0 = 0; //brisanje overflow-a
C:0x0371    C28D     CLR      TF0(0x88.5)
   23: TR0 = 1; //pustanje u rad brojaca
   24:
C:0x0373    D28C     SETB     TR0(0x88.4)
   25: PCON |= 0x01; //stavljanje procesora u idle stanje radi ustede energije
   26:
C:0x0375    438701   ORL      PCON(0x87),#0x01
   27: WDTCON=0x6B; //resetovanje watch dog tajmera
C:0x0378    75A76B   MOV      WDTCON(0xA7),#0x6B
   28: }
C:0x037B    22       RET
   29: void sleep_for(unsigned char time)
C:0x037C    8F0C     MOV      0x0C,R7
   30: {
   31:         unsigned char i=0;
C:0x037E    750D00   MOV      0x0D,#0x00
   32:         for(;i<time;)
C:0x0381    E50D     MOV      A,0x0D
C:0x0383    C3       CLR      C
C:0x0384    950C     SUBB     A,0x0C
C:0x0386    5007     JNC      C:038F
   33:         {
   34:             i++;
C:0x0388    050D     INC      0x0D
   35:             sleep_for_50ms();
C:0x038A    120361   LCALL    sleep_for_50ms(C:0361)
   36:         }
C:0x038D    80F2     SJMP     C:0381
```

```
  37: }
C:0x038F    22      RET
  133:              MOV     R0,#IDATALEN - 1
C:0x0390    787F    MOV     R0,#0x7F
  134:              CLR     A
C:0x0392    E4      CLR     A
  135: IDATALOOP:   MOV     @R0,A
C:0x0393    F6      MOV     @R0,A
  136:              DJNZ    R0,IDATALOOP
C:0x0394    D8FD    DJNZ    R0,IDATALOOP(C:0393)
  185:              MOV     SP,#?STACK-1
  186:
  187: ; This code is required if you use L51_BANK.A51 with Banking Mode 4
  188: ;<h> Code Banking
  189: ; <q> Select Bank 0 for L51_BANK.A51 Mode 4
  190: #if 0
  191: ;     <i> Initialize bank mechanism to code bank 0 when using L51_BANK.A51 with Banking
Mode 4.
  192: EXTRN CODE (?B_SWITCH0)
  193:              CALL    ?B_SWITCH0      ; init bank mechanism to code bank 0
  194: #endif
  195: ;</h>
C:0x0396    75810E  MOV     SP(0x81),#0x0E
  196:              LJMP    ?C_START
C:0x0399    0201C9  LJMP    main(C:01C9)
C:0x039C    00      NOP
```