FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
Zavod za automatiku i procesno računarstvo

UPRAVLJANJE PROCESOM KORIŠTENJEM USLUGE KRATKIH PORUKA

Diplomski rad broj 1231

Goran Jurković

Zagreb, 2001.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
POVJERENSTVO ZA DIPLOMSKI ISPIT

Zagreb, 06. studenog 2000.

Zavod: **Zavod za automatiku i procesno računarstvo**

Predmet: Računala I

DIPLOMSKI ZADATAK br. 1231

Pristupnik: **Goran Jurković**

Studij: Elektrotehnike
Smjer: Automatika

Zadatak: **UPRAVLJANJE PROCESOM KORIŠTENJEM USLUGE KRATKIH PORUKA**

Opis zadatka:

Proučiti način funkcioniranja GSM mreže i usluge kratkih poruka (SMS). Projektirati mikroračunalo koje će primati i slati kratke poruke nadzornom računalu. Ostvariti program za upravljanje procesom korištenjem nadzornog računala i usluge kratkih poruka.

Zadatak uručen pristupniku: 07. studenog 2000. u 12:00 sati

Rok za predaju rada: najkasnije do kraja akademske godine u kojoj je zadan diplomski zadatak

Mentor:

Doc.dr.sc. Mario Kovač

Djelovođa:

Doc.dr.sc. Ivan Petrović

Predsjednik povjerenstva za diplomski ispit:

Prof.dr.sc. Nedjeljko Perić

Zahvaljujem se mentoru prof. dr. Mariu Kovač,
i prof. dr. Mariu Žagar na uzoru,
zahvaljujem se roditeljima na
financijskoj i moralnoj podršci,
i posebno se zahvaljujem Mirjani Senić na poticaju i
podršci svih ovih godina tijekom moga studija,
koja nažalost više nije sa mnom…

SADRŽAJ

1. UVOD

Komunikacija među ljudima postala je vrlo popularna već nakon pojave prvog telefona. Od tada pa do današnjeg dana, telefonija je brzo napredovala, pa je tako došla i do prvih mobilnih analognih telefona. Još veću popularnost telefonija je dobila uvođenjem digitalne GSM mreže. Svojom pojavom GSM (Groupe Spéciale Mobile) telefonija, donijela je još puno dodatnih usluga, među kojima su i podatkovna komunikacija, WAP (Wireless Application Protocol), te vrlo popularna usluga kratkih poruka. U biti nije sigurno dali je usluga kratkih poruka pojačala popularnost GSM telefonije, ili je obrnuto. U svakom slučaju, gotovo da nema čovjeka u «razvijenom» dijelu svijeta koji nije barem jednom poslao kratku poruku (eng. SMS, Short Message Service)…

Porastom popularnosti usluge kratkih poruka, počele su se javljati nove ideje. No većina ih se svodi na automatizaciju slanja kratkih poruka, odnosno, njihovo primanje, tumačenje, te izvršavanje određene akcije. Tako se putem kratkih poruka mogu dobivati razne obavijesti od vremenske prognoze, kino predstava, dnevnih horoskopa, informacija iz zemlje i svijeta, a sve su potpuno automatizirane i ovise o davatelju takvih usluga. No nije se niti na tome stalo. Išlo se i dalje do na primjer, rezervacije autobusnih karata, ili kupovina cvijeća preko usluge kratkih poruka ili WAP-a. Također, i većina banaka je tu vidjela svoju budućnost, tako da se putem usluga kratkih poruka mogu dobiti informacije o stanju tekućeg računa, itd.… Dakle, mogućnosti su nebrojene, i neograničene. Jedino ih sputava mašta programera koji rade poslužitelje za navedene usluge.

Tako, u jednom trenutku pojavila se ideja kontroliranja stvarnih fizičkih procesa putem usluge kratkih poruka ili WAP-a. Takvih projekata ima dosta u svijetu, a napravljen je jedan i na «Fakultetu elektrotehnike i računarstva» u okviru predmeta «Odabrana poglavlja iz programskog inženjerstva», pod nazivom «SMS i WAP oslonjene usluge». Ovaj projekt se bazira na mreži osobnih računala, od kojih je jedno spojeno na GSM mobilni aparat, i jedno na PLC (Programable logic controller). PLC se brine za spoj između računala i stvarnog procesa kojim se upravlja. Kada kažem proces, mislim na bilo koji fizički aparat, ili stroj, kojim se može upravljati, paliti ili gasiti ili raditi neku drugu akciju. To može biti bilo što. Od aparata za kuhanje kave, aparata za zalijevanje travnjaka, sustava za grijanje kućanstva, klima uređaja, do sustava za gašenje požara u nekom laboratoriju, ili alarmnog sustava. Dakle bilo koji stroj može biti objekt upravljanja. Naravno poanta je upravljati sa što više strojeva ili procesa u nekom postrojenju ili kućanstvu, kako bi ovaj način upravljanja zaista imao smisla…

E sada, kako se upravlja? Kada se dogodi neka akcija, npr.: kvar nekog stroja u laboratoriju, nadzorno računalo pošalje kratke poruke osobama koje su zadužene za takve vrste akcija. Tada ta osoba, iako može biti na drugom kraju svijeta, može svojom kratkom porukom koju šalje nadzornom računalu narediti da se izvrši neka akcija, npr.: gašenje tog dijela laboratorija kako bi se spriječila potencijalna katastrofa…

Ovaj navedeni projekt je interesantan, i ima smisla kod vrlo velikih postrojenja ili objekata. No nema baš previše smisla staviti jedno ili više osobnih računala u manje kućanstvo ili u neko vozilo. Zato je interesantan upravo ovaj diplomski rad. Naime, puno je ekonomičnije nadzorno računalo, i PLC zajedno «ugraditi» u jedan mikro-kontrolor sa popratnim elementima. Prvenstveno radi se o daleko manjim dimenzijama, nego u slučaju osobnog računala, isto tako radi se o daleko manjoj cijeni, i potrošnji energije. Kako je nadzorno računalo na ovaj način smanjeno na veličinu jedne knjige, moguće ga je ugraditi i u osobni automobil. Mogućnosti su neograničene. Definitivno je najinteresantnije koristiti ovakav način kontroliranja procesa u protuprovalne svrhe ili protiv krađe manjih objekata, automobila. Uz određena proširenja korištenja usluga koje daje GSM mreža, moguće je koristiti ovaj projekt i za autorizaciju prilikom ulaska u objekt. Putem određenih

sabirnica, moguće je nadzorno računalo spojiti sa drugim kontrolorima u mrežu i povećati mu mogućnosti. No, najinteresantnija stvar kod ovog projekta je upravo to, da je kontrolno računalo sa dodanim mu GSM aparatom, potpuno autonomno u radu. Osobno računalo mu se može pridružiti poput klijenta kako bi korisnik mogao urediti bazu podataka, pridružiti određene dozvole za određene akcije korisnicima, te pogledati zapisnik akcija koje su izvedene…

No niti to nije sve. Putem posebne sabirnice, mogućnosti nadzornog računala moguće je proširiti sa dodatnim mikro-kontrolorima, i tako ga prilagoditi bilo kojoj prilici ili postrojenju. Mogućnosti su neograničene, i samo ovise o mašti programera i financijskim mogućnostima s kojima raspolažemo.

2. GSM MREŽA I NAČIN RAZMJENE KRATKIH PORUKA

GSM (Groupe Spéciale Mobile) mreža je bežična mreža za mobilnu komunikaciju bazirana na digitalnom prenošenju podataka sa mnogim dodatnim uslugama među kojima je i usluga kratkih poruka. Kao što je već navedeno u prvom poglavlju, GSM mreža kao i usluga kratkih poruka je dobila vrlo veliku popularnost, tako da će u narednim potpoglavljima biti više rečeno o samom razvoju mreže i načinu razmjene kratkih poruka putem GSM uređaja i nadzornog računala.

2.1. Kako je počeo razvoj GSM mreže

Razvoj GSM mreže je započeo ranih 80-ih godina. Već na samom početku vidjela se važnost takve mreže stoga su i planovi razvoja europske mobilne komunikacijske infrastrukture tekli upravo u tom smjeru. 90-ih godina se GSM naglo razvijao, tako da su danas njegove inačice DCS1800 i PCS1900 raširene po cijelom svijetu.

Sve je započelo 1982. godine kada je Europska komisija za pošte i telekomunikacije CEPT (European Conference of Posts and Telecommunications) donijela dvije važne odluke. Prva odluka je bila da osnuje grupu čiji je naziv bio «Groupe Spéciale Mobile», koja će razviti i odrediti standarde za buduću europsku mrežu mobilne telefonije, dok je druga preporuka bila da se za rad sustava uzmu dva frekvencijska opsega u području od 900 MHz. GSM kratica dolazi od «Groupe Spéciale Mobile», čije je značenje kasnije preimenovano u «Global System for Mobile Communications», što više nije GSM kratica, ali je ona zadržana.

Razlog zalaganja CEPT-a za ove odluke je pokušaj rješavanja problema koji je postojao zbog nekoordiniranog razvoja mobilne komunikacije, jer su nacionalne mreže individualno stvarale svoje standarde. Najveći problem ovakvog razvoja mobilne komunikacije je bio nemogućnost korištenja jednog mobilnog uređaja dok se putuje po Europi i isto tako problem nemogućnosti razvoja jake europske industrije mobilnih komunikacija koja bi mogla konkurirati na svjetskom tržištu.

Već 1986. godine razvoj je krenuo prema digitalnim mrežama jer se već tada smatralo da je analogna mobilna mreža nedovoljna za pokrivanje kapaciteta i zahtjeva 90-ih godina. Na osnovu analiza za GSM mrežu su rezervirana 2 frekvencija područja od 900MHz za Europsko područje. Isto tako postignut je veliki napredak u razvoju dogovornih standarda. Koristit će se digitalni, a ne analogni sustav.

Digitalni sustav nudi poboljšanu efikasnost spektra, bolju kvalitetu prijenosa podataka i nove usluge uključujući poboljšane sigurnosne mogućnosti. Digitalni sustav omogućuje korištenje VLSI tehnologije (Very Large Scale Integration) koja vodi ka korištenje manjih i jeftinijih mobilnih telefona. I konačno, digitalna tehnologija približava završetku razvoj ISDN-a (Integrated Services Digital Network) koju će GSM imati kao sučelje.

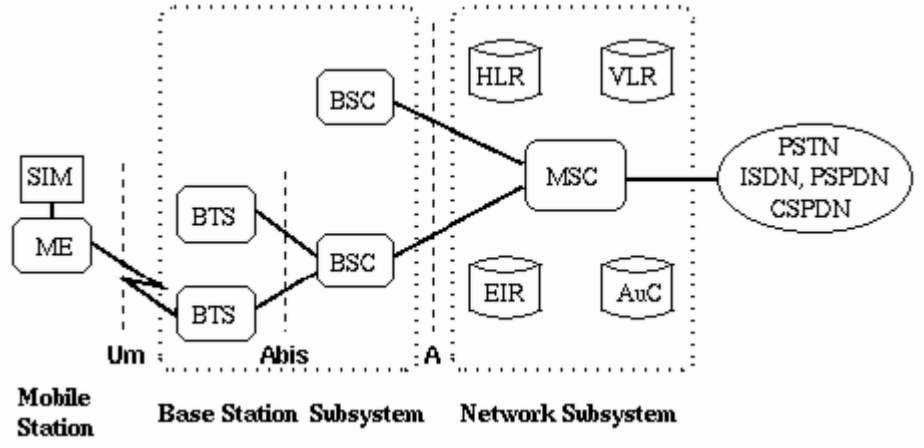
Isto tako velika odlika GSM sustava je mogućnost međunarodnog roaminga. Roaming je kretanje iz države u državu, a da isti mobilni telefon uvijek radi, naravno uz određenu naplatu ove usluge. Isto tako GSM nudi dobru kvaliteta zvuka, malen i jeftin uređaj, te sposobnost da ga koristi velik broj korisnika. GSM je uzet krajem 1989 godine od ETSI (European Telecommunications Standards Institute), a oni su finalizirali GSM standard tijekom 1990. GSM usluga je počela s radom 1991. godine.

2.2. GSM zahtjevi

GSM mora imati daleko kvalitetniji zvuk nego što je to imala analogna mreža na 900MHz. Ovo mora biti ispunjeno u svim operativnim uvjetima. GSM mora imati mogućnost enkripcije korisnikovih podataka. Sustav mora raditi u cijelom frekvencijskom području 890-915MHz i 935-960MHz. GSM mora koristiti međunarodni standardizirani signalni sustav kako da bi se dopustilo povezivanje mobilnih "prekidačkih" centara. Također, uvođenje GSM mreže mora biti moguće uz minimalne promjene na postojećoj fiksnoj mreži. GSM mreža mora imati što jeftiniji korisnikov uređaj. Mobilni uređaj mora biti u mogućnosti raditi u svim državama koje koriste GSM mrežu. Maksimalna fleksibilnost s ostalim servisima kao što je ISDN.

2.3. Princip rada GSM-a

GSM mreža se sastoji od tri dijela: **mobilne stanice** (MS - *Mobile Station*) koja je slična bežičnom telefonu s dodatnim mogućnostima, **primopredajne bazne stanice** (BTS - *Base Transceiver Station*) koja upravlja vezom s mobilnom stanicom, te na kraju od **upravljača baznim stanicama** (BSC - *Base Station Controller*) koji upravlja većim brojem primopredajnih baznih stanica. Na slici 2.1. nalazi se shema uređaja koji su potrebni za ostvarivanje komunikacije unutar GSM mreže.



Slika 2.1. Princip rada GSM mreže.

Mobilna stanica (MS): Digitalni mobilni telefon i **SIM** (*Subscriber Identity Module*) kartica tvore mobilnu stanicu. SIM kartica sadrži sve korisnikove identifikacijske pojedinosti kao što su **IMSI** (*International Mobile Subscriber Identity*), memoriju za čuvanje telefonskih brojeva, informacije o troškovima, kratke poruke (SMS), brojeve pinova i informacije o međunarodnom roamingu. IMSI broj je potreban kako bi se korisnik mogao predstaviti mreži uz ostale korisnikove i sigurnosne informacije.

IMEI (*International Mobile Equipment Identity*) je serijski broj GSM aparata, te predstavlja ekvivalent **ESN** broju kod analognih telefona.

Primopredajna bazna stanica (BTS): Sastoji se od radio primopredajnika s antenom koja pokriva jednu zonu, a komunicira sa MS putem radio sučelja. BTS-ovi su međusobno povezani kako bi omogućili korisniku prelaženje iz jedne u drugu zonu. Antena može imati različite oblike.

Upravljač baznim stanicama (BSC): Ovaj kontrolor je neophodan za zajednički rad BTS-ova. Upravlja dodjelom i otpuštanjem radio kanala, te vodi računa o prelasku iz jedne u drugu zonu.

Cijeli niz BTS-ova je spojen na jedan upravljač baznim stanicama koji nadgleda svaki poziv i odlučuje kada je potrebno proslijediti poziv drugom BTS-u, kao i kojem BTS-u.

Ostatak mreže: Određeni broj BSC-ova je pod kontrolom preklopničkog centra (**MSC** - *Mobile service Switching Centre*). MSC radi u suradnji s četiri baze podataka - **HLR**, **VLR**, **EIR** i **AuC**, te zajedno tvore komunikaciju između korisnika mobilne stanice i ostalih mrežnih uređaja.

Preklopnički centar (MSC). To je sučelje koje povezuje sustav baznih stanica i preklopnički podsustav mreže mobilnih telefona. MSC je također i sučelje između GSM mreže i **PSTN**-a. MSC generira sve troškovnike i brine se o tome da se svaka upotreba usmjerava prema pravom korisničkom računu. MSC ima vrlo kompliciran posao jer, za razliku od standardne telefonske mreže GSM pretplatnik može vršiti pozive sa bilo kojeg mjesta unutar mreže. Mora se pobrinuti da se pozivi usmjere prema pretplatniku bez obzira gdje se on nalazi i bez obzira na to da li miruje ili prelazi iz jednu u drugu zonu za vrijeme poziva. Situacija postaje još kompleksnija ako dva pretplatnika iste GSM mreže uspostave komunikaciju s međusobno udaljenih lokacija. Da bi se pojednostavnilo upravljanje pretplatnikovim pozivima, svakom MSC-u je dodijeljeno njegovo vlastito područje rada. MSC sada upravlja pretplatnikovim pozivima samo u svom vlastitom području.

HLR (*Home Location Register*) je centralna baza podataka svih pretplatnika koja sadrži detalje o njihovom identitetu, uslugama kojima oni imaju pristup i lokaciji na kojoj su se nalazili prilikom posljednje provjere. Dvije ključne reference koje se koriste prilikom usmjeravanja poziva su IMSI i **MSISDN** (*Mobile Subscriber Integrated Services Digital Network*) brojevi. IMSI je jedinstveni broj pridružen pretplatniku, zapisan u SIM kartici, a mreža ga koristi za komunikaciju s pretplatnikom.

VLR (*Visitor's Location Register*) je baza podataka povezana s MSC-om, koja privremeno čuva informacije o svakoj mobilnoj stanici unutar područja koje opslužuje pripadajući MSC. Informacije u VLR-u uključuju identitet mobilne stanice, područje u kojem je zadnji put zabilježena, pretplatnikovi podaci i dostupne servise dostupne pretplatniku. MSC koristi VLR prilikom svakog pokušaja mobilne stanice da ostvari poziv, kako bi provjerio da li se zahtjev može ispuniti.

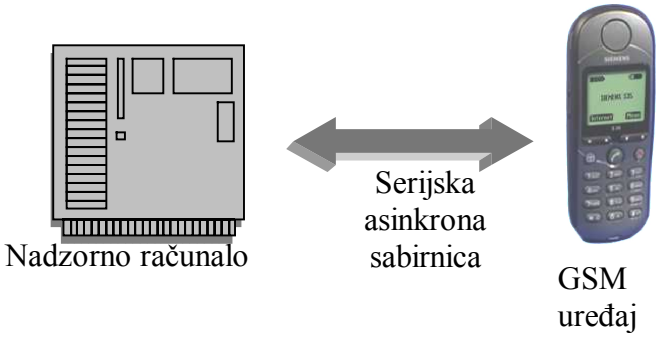
EIR (*Equipment Identity Register*) osigurava da je mobilna oprema važeća i da joj je dozvoljen rad unutar PLMN-a. EIR ima tri kategorije: bijelu, sivu i crnu listu. Bijela lista sadrži popis IMEI brojeva svih mobilnih uređaja odobrenih od strane bilo kojeg GSM centra za odobravanje (u Europi ih je tri). Mobilnoj opremi koja se nalazi na sivoj listi je dozvoljen rad ali će njihova upotreba uključiti "alarm" mrežnom operatoru. Ova pogodnost omogućuje mrežnom operatoru da identificira pretplatnika koji koristi izgubljeni ili ukradeni mobilni uređaj. Izgubljeni ili ukradeni mobiteli također se mogu upisati na crnu listu što onemogućava njihov rad unutar matične PLMN ili unutar drugih PLMN-a u svijetu.

CEIR (*Central Equipment Identity Register*). Centralnim EIR-om upravlja **MoU Permanent Secretariat** (Dublin, Irska). Svaki član MoU-a (*Memorandum of Understanding*) mora povezati EIR svoje mreže s CEIR-om. Prednost CEIR koncepta je što omogućuje svakom mrežnom operatoru da ograniči ili zabrani rad bilo koje mobilne stanice u svim PLMN-ima povezanim u CEIR.

AUC (*Authentication Centre*) provjerava ispravnost SIM kartica u svakoj pojedinoj mobilnoj stanici. Korištenjem tajnih informacija koje AUC posjeduje, a koje su također sadržane u SIM kartici, vrše se kompleksni matematički proračuni. Vjerodostojnost se potvrđuje ako se rezultati dvaju proračuna podudaraju.

2.4. Usluga kratkih poruka i način komunikacije između GSM-a i nadzornog računala

U samom projektu potrebno je naći način na koji se kratka poruka može poslati u GSM svijet i primiti iz njega. Postoje specijalni moduli, dijelovi GSM aparata koji služe samo za kratke poruke, ali iz ekonomskih razloga i težine dobavljivosti oni nisu korišteni. Odlučio sam se za jednu daleko jednostavniju i ekonomičniju opciju. Koristio sam GSM telefon, nižeg cjenovnog razreda, ali sa jednom posebnom opcijom: ugrađenim standardnim modemom. Na slici 2.2. prikazana je takva konfiguracija, i način na koji se poruka može poslati i primiti.



Slika 2.2. Način slanja i primanja kratkih poruka

Postoji dosta GSM uređaja koji imaju ugrađen modem, no oni su obično u višem cjenovnom razredu. Dakle, neki od popularnijih modela koji imaju modem su: Nokia 8210, Nokia 8850, Nokia 7110, Nokia 6210, Sagem 850, Sagem 950, Siemens S10, Siemens S25, Siemens C35i, Siemens M35i, Siemens S35i, Siemens SL45. Izbor je pao na Siemens C35i koji je ujedno mogućnostima vrlo napredan i dosta je jeftin. Znači, teoretski moguće je spojiti bilo koji od gore navedenih telefona na nadzorno računalo, ali puna funkcionalnost će se ostvariti samo spajanjem ova 3 modela: Siemens C35i, M35i i S35i. Kod ostalih neke funkcije nisu implementirane, kao stvarno vrijeme (RTC – Real Time Clock), itd.…
Nadalje, povezivanje GSM uređaja je ostvareno putem asinkrone serijske veze na nadzorno računalo, pod nazivom RS232. Sam GSM uređaj koristi pozitivnu TTL asinkronu serijsku vezu, koja se konvertira na RS232 nivoe u samom spojnom kabelu. Iz tog razloga na nadzornom računalu postoji i priključak sa TTL nivoima, čime je izbjegnuta dvostruka konverzija (i nadzorno računalo radi sa TTL nivoima koji se konvertiraju u RS232). Dakle, to je način razmjene podataka između GSM uređaja i kontrolora. Komunikacija se odvija na brzini od 19200bps, unutar 8 bita, bez bita pariteta, i sa jednim stop bitom. Sam način slanja i primanja kratkih poruka ostvaruje se «AT» naredbama…

Što su to «AT» naredbe? Kao što sam na početku naveo, telefon mora imati standardni modem. U biti riječ modem dolazi od riječi modulator i demodulator što se vežu za prijenos podataka putem analognih linija. No kako je GSM baziran na digitalnoj komunikaciji, ovdje nije potreban modem. Ono što modem predstavlja je procesor, koji zna tumačiti AT naredbe od strane nadzornog računala i iste izvršavati. Tako, postoji standardni set AT naredbi, koje su proširene GSM AT setom, inačica 07.05. i 07.07. U tom proširenom setu, nalaze se i naredbe vezane za uslugu kratkih poruka,

odnosno njihovo slanje i primanje. Također, Siemens C35i ima i neke dodatne naredbe za poziv trenutnog vremena, itd.…
Neke od korištenih AT naredbi su prikazane u tablici 2.1., dok se puni popis naredbi, te njihov detaljan opis nalazi u literaturi [5] i u prilogu na CD-u. Isti je moguće naći na Internetu na Siemensovim stranicama: www.siemens.com.

AT+CGMI	Prikaz proizvođača mobilnog telefona
AT+CGMM	Prikaz modela telefona
AT+CGMR	Prikaz inačice telefona
AT+CGSN	Prikaz serijskog broja telefona (IMEI)
AT+CCLK	Prikaz stvarnog vremena (RTC)
AT+CSMS	Odabir usluge poruka
AT+CMGF	Odabir formata kratkih poruka
AT+CNMI	Prikaz novih pristiglih poruka
AT+CMGL	Lista svih poruka
AT+CMGR	Čitanje kratke poruke
AT+CMGS	Slanje kratke poruke
AT+CMGC	Slanje SMS naredbe

Tablica 2.1. Prikaz nekoliko AT naredbi.

Određenim nizom AT naredbi moguće je iščitati novu poruku, dekodirati ju na standardni ASCII oblik, te ju protumačiti. Poruku je potrebno dekodirati zato što se poruka prima i šalje u PDU standardnom formatu za poruke. PDU nosi puno informacija, među kojima su i broj centra za poruke, broj pošiljatelja, valjanost poruke, format poruke i mnoge druge informacije.

Protumačena poruka nosi neku informaciju, ili naredbu za stanje određenog procesa, kojim nadzorno računalo upravlja. Isto tako, određenim nizom AT naredbi moguće je poslati poruku, tj. obavijest korisniku da se promijenilo neko stanje u procesu koje se kontrolira. Slanje poruke je dosta kompliciranо i zahtjeva pretvorbu ASCII 7 bitne tekstualne poruke u 8 bitnu zapisanu u hexadecimalnom zapisu i poslanu kao tekstualni niz. To se naziva PDU kodiranje. Isto tako PDU sadrži druge informacije. Detaljnije se može naći u literaturi [18], dok se u tablici 2.2 nalazi osnovni opis PDU formata dovoljan za potrebe nadzornog računala.

Okteti	Opis
00	Dužina broja centra za usluge kratkih poruka. Ako je 0 znači da se koriste postavke već postavljene unutar GSM uređaja.
11	Prvi oktet poruke. Nosi postavke i način slanja poruke. Detalje o njemu mogu se naći u literaturi [18].
00	Referenca poruke. «00» znači da ju telefon sam postavlja.
0B	Dužina adrese. Dužina telefonskog broja (11).
91	Tip adrese. «91» znači da se radi o standardnom internacionalnom formatu telefonskog broja.

6407281553F8	Telefonski broj je zapisan u polu-oktetima (+46708251358). Dužina broja je neparna (11, bez znaka +), stoga je potrebno dodati završni znak «F». Dobiveni broj u polu-oktetima izgleda ovako: "46708251358F".
00	Identifikacija protokola.
00	Shema kodiranja podataka. Znači da se koristi standardni 7 bitni ASCII znakovi.
AA	Valjanost poruke. «AA» znači trajanje od 4 dana.
0A	Dužina korisnikove poruke. Predstavlja dužinu poruke u 7 bit ASCII znakovima (10).
E8329BFD4697D9EC37	Korisnička poruka. Ovi okteti predstavljaju poruku «hellohello».Kako napraviti pretvorbu iz 7 bit ASCII znakova u oktete prikazano je u tablici 2.3.

Tablica 2.2. Način kodiranja poruke u PDU blok.

Sada kada bi htjeli poslati tu istu poruku koju smo kodirali prema tablici 2.2. to bi izgledalo ovako:

```
AT+CMGF=0     // Postavi PDU mod
AT+CSMS=0     // Provjeri dali modem podržava SMS naredbe
AT+CMGS=23  // Pošalji poruku, 23 okteta (bez dvije početne nule)
>0011000B916407281553F80000AA0AE8329BFD4697D9EC37<CTRL+Z>
```

U tablici 2.3. prikazano je na koji način se tekstualna ASCII 7 bitna poruka kodira u oktete. Tada se ti okteti predstavljaju hexadecimalnim brojevima u ASCII obliku. Poruka u hexadecimalnim brojevima prikazana je u tablici 2.4.

h	**e**	**l**	**L**	**o**	**h**	**e**	**l**	**l**	**o**
104	**101**	**108**	**108**	**111**	**104**	**101**	**108**	**108**	**111**
1101000	1100101	1101100	1101100	1101111	1101000	1100101	1101100	1101100	1101111
1101000	110010 1	11011 00	1101 100	110 1111	11 01000	1 100101	1101100	1101100	110111 1

Tablica 2.3. Način kodiranja 7 bit ASCII u 8 bit.

1 1101000	00 110010	100 11011	1111 1101	01000 110	100101 11	1101100 1		1 1101100	110111
E8	**32**	**9B**	**FD**	**46**	**97**	**D9**		**EC**	**37**

Tablica 2.4. Poruka prikazana u hexadecimalnom obliku.

Ovo je bio način slanja poruka i kodiranje poruka kakav je korišten u programskoj podršci za nadzorno računalo. Sada će biti prikazan način čitanja poruka i njihovo dekodiranje u ASCII format. O svemu detaljnije može se naći u literaturi [18].

```
AT+CMGL=0         // Izlistati će sve novo pristigle poruke
+CMGL: 3,1,,22    // Broj 3 označava poruku, a 22 označava broj okteta bez centra
069183950805F1240B918395584475F40000108013418392800 3C1F51B
```

Recimo da je primljena poruka prikazana u tablici 2.5. tada bi njen sadržaj bio «hellohello» a detalji oko kodiranja su prikazani u tablici 2.6.

07 917283010010F5 040BC87238880900F10000993092516195800 3C16010

Tablica 2.5. Primljena poruka u PDU formatu.

Okteti	Opis
07	Dužina broja centra za uslugu kratkih poruka. (u ovom slučaju iznosi 7 okteta)
91	Tip adrese odnosno broja. U ovom slučaju internacionalni.
72 83 01 00 10 F5	Broj centra pošiljatelja za uslugu kratkih poruka (u decimalnim polu-oktetima). Dužina broja je neparna (11), pa mu je dodan «F» kako bi broj bio pravilno kodiran u oktetima. Broj centra je "+27381000015".
04	Prvi oktet primljene poruke. Za detalje pogledati literaturu [18].
0B	Dužina telefonskog broja pošiljatelja ($0B_{16} = 11_{10}$)
C8	Tip adrese odnosno broja. Pogledati literaturu [18] za detalje.
72 38 88 09 00 F1	Broj pošiljatelja (zapisan u decimalne polu-oktete), sa završnim znakom «F». Broj je «27838890001»
00	Identifikacija protokola. Pogledati literaturu [18] za detalje.
00	Shema kodiranja podataka. Pogledati literaturu [18] za detalje.
99 30 92 51 61 95 80	Vrijeme slanja poruke (polu-okteti). Pogledati literaturu [18] za detalje.
0A	Dužina teksta dekodirane poruke u 7 bit ASCII znakovima (10).
E8329BFD4697D9EC37	Tekst poruke «hellohello» kodiran 8 bitno hexadecimalno. Način kodiranja prikazan je u tablici 2.3. i 2.4.

Tablica 2.6. Detalji oko kodiranja primljene poruke u PDU formatu.

Kao što se vidi iz prethodne tablice, puno informacija je zapisano u PDU bloku koji se dobije kao odgovor na naredbu za čitanje poruke. U tablici 2.6. puno informacija je i zanemareno jer nisu bitni za nadzorno računalo stoga ih nisam niti objašnjavao. No, cjelokupna dokumentacija može se naći na www.etsi.org i u literaturi [18].

3. NADZORNO RAČUNALO - POSLUŽITELJ

Kao što sam već u uvodu naveo osnovne prednosti ovog projekta i ovakvog načina kontroliranja procesa putem usluge kratkih poruka, no navesti ću ih ponovno:

- male dimenzije nadzornog računala – poslužitelja
- mala potrošnja energije, mogućnost rada na baterije
- potpuna autonomnost u radu
- zahtijeva samo GSM uređaj za normalan rad
- jednostavno spajanje na proces već postojećim digitalnim ulazima izlazima
- osobno računalo nije neophodno za rad i koristi se kao klijent
- moguće proširenje mogućnosti dodavanjem dodatnih mikro-kontrolora na CAN sabirnicu

Nadzorno računalo – poslužitelj, se bazira na mikroprocesoru DS80C390 proizvođača DALLAS Semiconductors, koji predstavlja sam vrh u svojoj klasi. Korišteni mikroprocesor, ili popularno zvan mikrokontroler, spada u porodicu Intel 80C51 mikro-kontrolora.

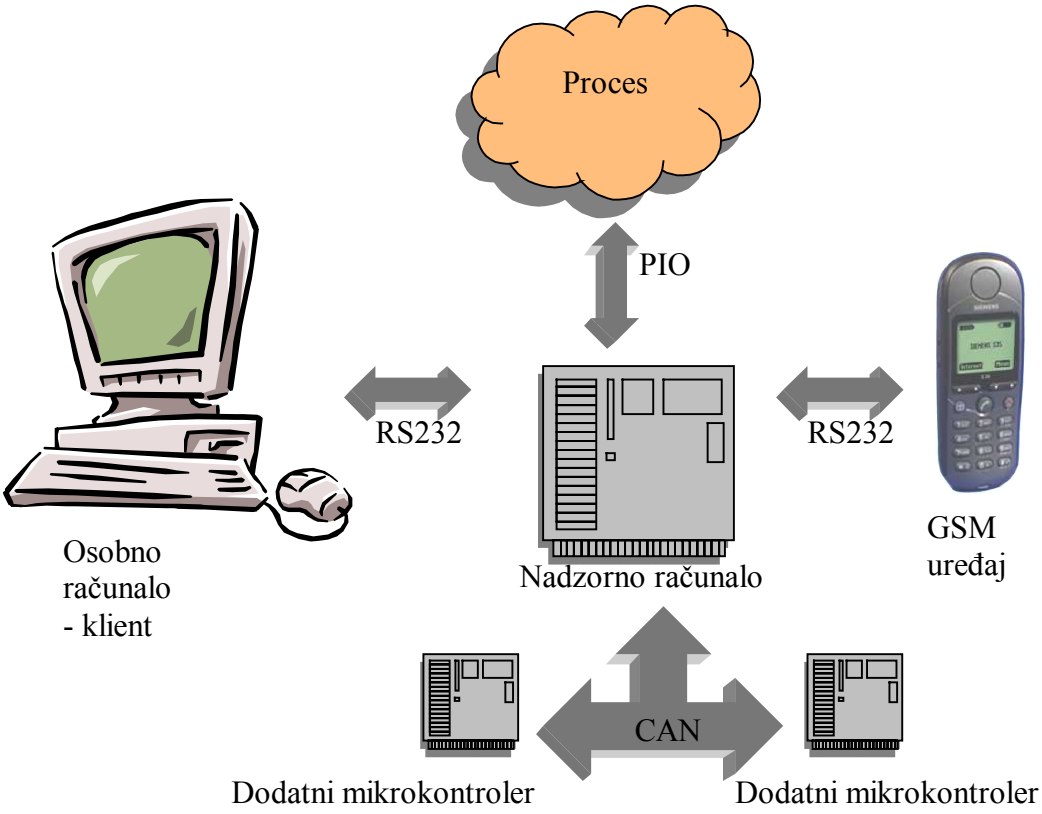
Nadalje nadzorno računalo je opremljeno sa 64KB programske i 64KB podatkovne memorije čiji se podatci čuvaju i nakon nestanka napajanja sa pomoćnom baterijom i popratnom elektronikom. Isto tako mikrokontroler ima internih 4KB memorije, i 32KB stalne memorije za startni program. Tu se radi o takozvanoj ROM emulaciji o kojoj će više biti govora u potpoglavlju 3.2. ROM emulacija je veoma interesantna kod razvojnih aplikacija upravo iz razloga što se programska memorija veoma jednostavno puni iz osobnog računala preko serijske komunikacije.

Također, nadzorno računalo ima dvije serijske asinkrone sabirnice popularno zvane RS232 za komunikacija sa osobnim računalom i GSM uređajem. Obje su veoma važne za rad i razvoj programske podrške, te za normalan rad nadzornog računala. Kao što sam već spomenuo, da bi nadzorno računalo normalno funkcioniralo neophodno je da je GSM uređaj spojen preko serijske sabirnice na nadzorno računalo. Sve ostalo nije neophodno. No poželjno je spojiti osobno računalo prilikom izrade programske podrške, uređivanja baze podataka, prelistavanja zapisnika o poduzetim akcijama, i slično.

Za komunikaciju sa ostalim mikro-kontrolorima u svrhu proširenja mogućnosti, nadzorno računalo posjeduje CAN sabirnicu. To je dvosmjerna serijska diferencijalna sabirnica na koju je moguće spojiti vrlo velik broj uređaja. O ovoj sabirnici će biti više govora u potpoglavlju 3.5.
Nadzorno računalo posjeduje i dvije paralelne digitalne ulazno izlazne sabirnice za spajanje na vanjski svijet odnosno procese. Jedna sabirnica je 8 bitna, dok je druga 4 bitna. Obje mogu biti ulazne, izlazne ili dvosmjerne, a potpuno su upravljive iz programske podrške. Ovo je ujedno i najjednostavniji način da se nadzorno računalo spoji na vanjski svijet, na razne aparate, senzore i slično. O ovim sabirnicama će biti više govora u potpoglavlju 3.6.

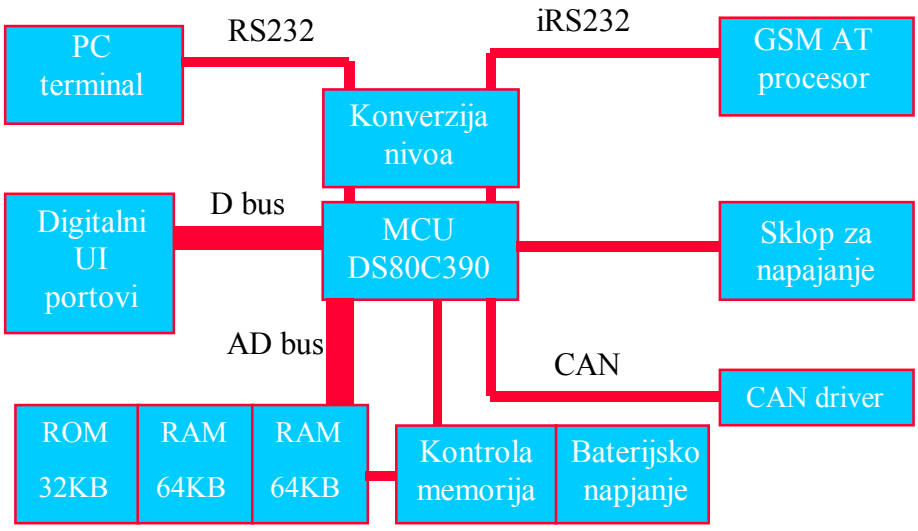
Mikroprocesor radi na 11,0592MHz što je ujedno odabrana frekvencija zbog asinkrone serijske komunikacije koja se odvija na 19200bps. Iako ovo nije najveća frekvencija na kojoj procesor može raditi, dovoljno je velika da pokrije zahtjeve projekta, a ujedno je i potrošnja energije manja ako je frekvencija niža.

Na slici 3.1. nalazi se uobičajena konfiguracija nadzornog računala spojenog na proces, i dva dodatna mikro-kontrolora spojena na nadzorno računalo, koji nisu neophodni, ali ih je moguće spojiti. Na slici 3.2. nalazi se blok shema nadzornog računala.



Slika 3.1. Uobičajena konfiguracija nadzornog računala i procesa

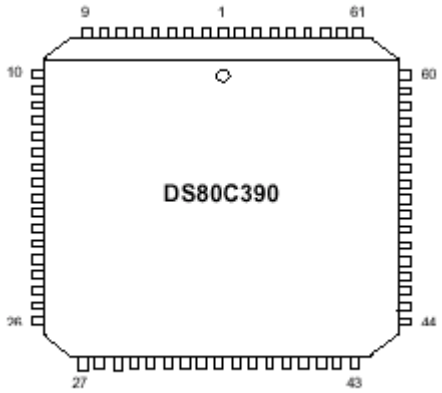
U narednim potpoglavljima poglavlja 3, biti će detaljno objašnjen svaki dio nadzornog računala, kao i električna shema svakog dijela. Kompletna shema cjelokupnog nadzornog računala, nalazi se u prilogu. Jedino sklop za reset i oscilator neće biti zasebno objašnjeni jer su previše jednostavni. Kako DS80C390 posjeduje unutar same jezgre cjelokupni sklop za reset sa raznim zaštitama, u projektu je dodan samo kratkospojnik (JP6) kojim se procesor dovodi u stanje reseta. Oscilator je izveden na najjednostavniji mogući način kakvog preporuča sam proizvođač. Sastoji se od kristalnog rezonatora (Y1) frekvencije 11,0592MHz i dva kondenzatora (C8,C9) kapaciteta 47pF.



Slika 3.2. Blok shema nadzornog računala.

3.1. Nadzorno računalo i korišteni procesor (mikrokontroler)

U projektu je korišten DALLAS Semiconductors DS80C390. To je 8-bitni mikroprocesor visoke radne brzine, i mnogobrojnih dodatnih mogućnosti. Spada u porodicu Intel 80C51 mikroprocesora, s tim da je veoma unaprijeđen. Pored daleko većeg broja priključaka (digitalnih ulazno/izlaznih vodova), ujedno posjeduje i dvije asinkrone serijske sabirnice, dvije CAN2.0B sabirnice, dodatne vanjske prekide, razne zaštite od pada napona napajanja, nadzorni sklop (eng. Watchdog), unaprijeđene načine rada sa svrhom manjeg utroška energije, može adresirati do 4MB memorije. Na slici 3.3. prikazan je izgled procesora u 68 pinskom PLCC kućištu.



Slika 3.3. mikroprocesor 80C390

Ovaj mikro-kontrolor (mikroprocesor) je najnoviji proizvod DALLAS-a u svojoj klasi, dakle porodici Intel 80C51. Vrlo je interesantan upravo zbog dvije serijske asinkrone sabirnice i CAN sabirnice. Dvije asinkrone serijske sabirnice su neophodne za ostvarivanje ovoga projekta, dok je postojanje CAN sabirnice poželjno i veoma interesantno za umrežavanje više mikro-kontrolora.

Dakle, DS80C390 je brzi 8051 uskladiv mikroprocesor. Redizajnirana procesorska jezgra izvršava instrukcije procesora 8051 i do 3 puta brže nego originalna jezgra, na istoj procesorskoj brzini. DS80C390 podržava najveću brzinu jezgre od 40MHz, rezultirajući brzinom od oko 100MHz ekvivalentnu 8051 jezgri. Opcionalni interni umnoživač frekvencije, omogućava procesoru da izvana radi na manjoj frekvenciji, a unutar same jezgre na maksimalnoj mogućoj, što smanjuje elektromagnetske smetnje kod ostalih komponenti. Nadalje, matematički ubrzivač dodatno ubrzava izvršavanje matematičkih operacija nad 32 i 16-bitnim brojevima.

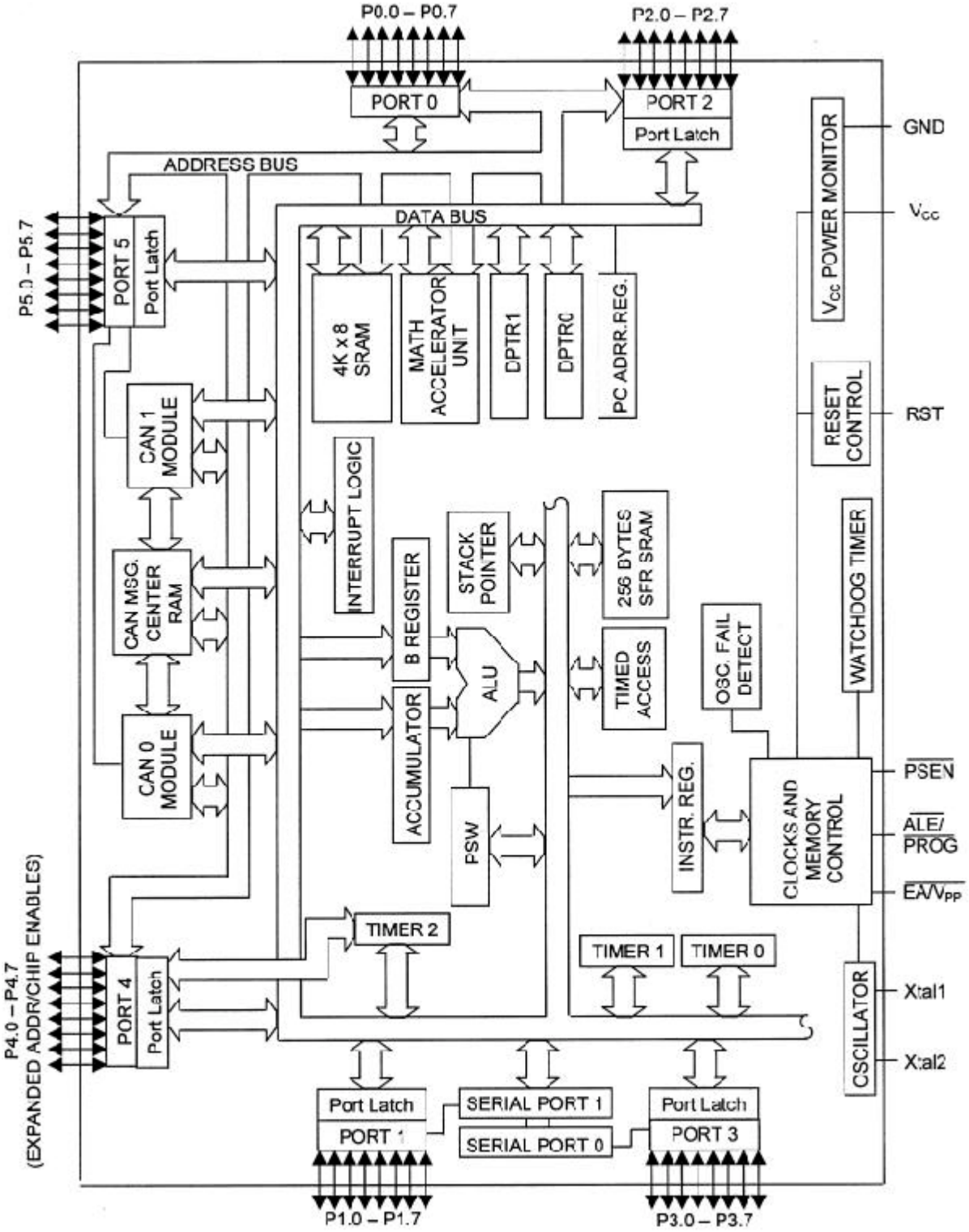
Sve standardne opcije 8051 su sadržane unutar DS80C390, kao tri brojača, serijski port, i četiri 8-bitna ulazno/izlazna porta, uz dodana dva 8-bitna porta dodijeljena radu sa memorijom. U dodatku, DS80C390 ima još jedan serijski port, sedam dodatnih prekidnih linija, programibilni nadzorni sklop (eng. Watchdog), zaštitu od neispravnog napajanja, i propada napona napajanja. Procesor pruža dvostruki podatkovni pokazivač sa inkrementalnim i dekrementalnim mogućnostima, za ubrzanje prebacivanja bloka podataka u memoriji. Također može podesiti brzinu MOVX naredbe prilikom pristupa podatkovnoj memoriji od 2 strojna ciklusa do 12 strojnih ciklusa zbog fleksibilnosti u adresiranju vanjskih memorija i ostale periferije.

Procesor sadrži 4KB interne statičke memorije, koja može služiti u više svrha, u kombinaciji sa MOVX naredbom, kao programska memorija, ili kao memorija stoga.

Novi PMM (Power Management Mode) je veoma koristan kod prenosivih dizajna, aplikacija koje štede energiju. Ovo omogućava programskoj podršci da se sama prebaci sa standardnog strojnog

ciklusa od 4 perioda signala takta po ciklusu na 1024 perioda signala takta po ciklusu. Na ovaj način program će se izvršavati daleko sporije, ali će potrošnja biti malena kao kada je procesor u «standby» modu.

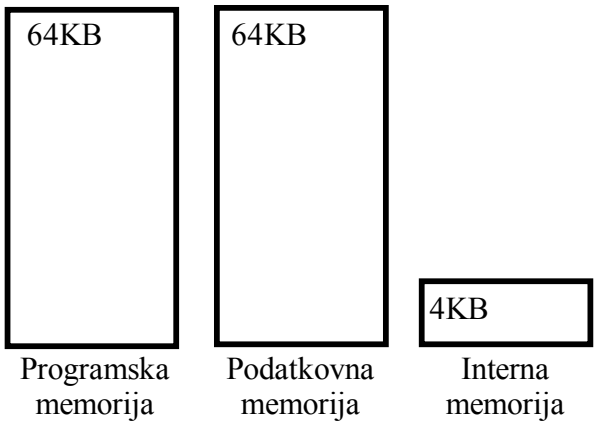
Na slici 3.4. prikazana je blok shema unutrašnjosti mikroprocesora DS80C390 sa svim funkcionalnim blokovima.



Slika 3.4. Blok shema jezgre procesora DS80C390

3.2. Konfiguracija memorije na nadzornom računalu i ROM emulacija

Kako je ovaj diplomski rad više razvojni projekt, zamišljen je na taj način da bude dovoljno snažan i programskom i podatkovnom memorijom. Tako nadzorno računalo posjeduje 64KB programske i 64KB vanjske podatkovne i 4KB unutarnje podatkovne memorije. Konfiguracija memorije je prikazana na slici 3.5.



Slika 3.5. Konfiguracija memorije

Uobičajeno je da dizajn sa mikro-kontrolorom za programsku memoriju ima EPROM, koji se napuni jednom finalnom inačicom programa i to dosta komplicirano sa posebnim strojevima. No, kako se ovdje radi o razvojnom projektu, poželjno je da se programska memorija može brzo i jednostavno puniti sa osobnog računala, preko serijske asinkrone sabirnice, koja se inače koristi za razmjenu podataka između osobnog računala i nadzornog računala odnosno mikro-kontrolora. Tako se nova inačica programa za mikro-kontrolor može u svakom trenutku prebaciti na nadzorno računalo i početi ga izvršavati. Ali, Intel 8051 porodica mikro-procesora, kao i DS80C390 ne podržavaju takav način rada. Stoga sam u dizajnu samog uređaja morao ugraditi posebne mehanizme, odnosno dodatno sklopovlje koje će omogućiti emulaciju programske memorije u podatkovnoj memoriji. Ovo sklopovlje omogućuje 2 načina rada memorije koja je spojena na DS80C390 mikroprocesor. To su:

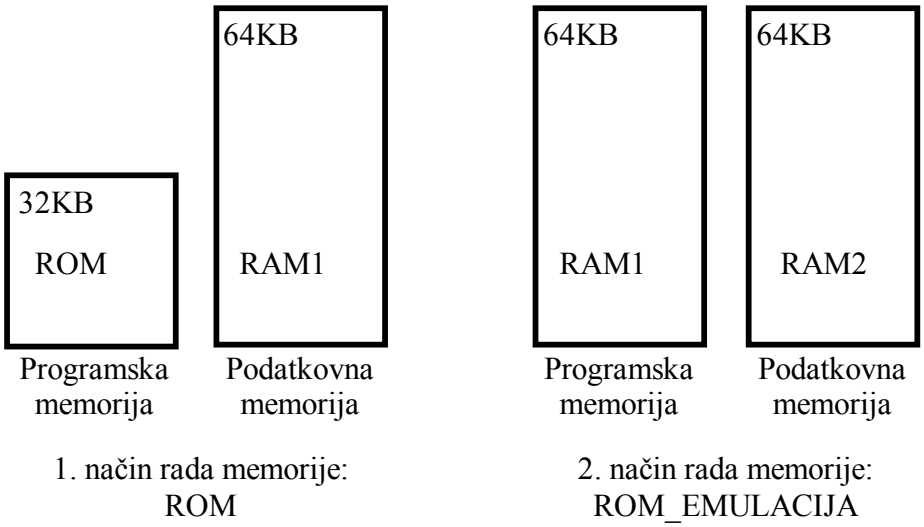
1. ROM: Punjenje memorije programom
2. ROMEMU: Izvršavanje programa

U 1. načinu rada program koji se izvršava je stalni, nepromjenjivi pokretački program koji se nalazi u EPROM (Electrically Programable Read Only Memory) memoriji, i popularno se naziva «bootstrap loader». Memorijski prostor u RAM (Random access memory) koji služi za programsku memoriju se sada ponaša kao podatkovna memorija. Program koji se izvršava iz EPROM-a, uzeti će podatke sa serijske asinkrone sabirnice iz osobnog računala, po potrebi ih prevesti na odgovarajući oblik i spremiti u podatkovnu memoriju. Kada je proces punjenja memorije završen, pozove se sklopovlje za prebacivanje načina rada i to preko linije «/ROMEMU_SWP» (pogledati shemu u prilogu), i nakon automatskog reseta koji će se dogoditi sam zbog isteka vremena nadzornog sklopa, procesor će preći u 2. način rada.

U 2. načinu rada EPROM je u stanju visoke impedancije, i ponaša se kao da ne postoji. U ovom načinu rada dio memorije koji je napunjen programom u prethodnom koraku, sada se ponaša kao

ROM (Read Only Memory) memorija, i iz nje se izvršava program. Drugi dio RAM memorije se ponaša kao programska memorija i sa njom se radi unutar programa. U nju se upisuju baze podataka, logovi akcija, i svi ostali podatci. Podatci i program unutar RAM memorije se čuvaju i nakon nestanka napona napajanja sa baterijom.

Ako bi programer htio u nekom trenutku napuniti memoriju novim programom, on bi nadzornom računalu poslao zahtjev za prelazak u 1. način rada. Prelazak se odvija kao i iz 1. u 2. Jednostavno preko linije «/ROMEMU_SWP» i nakon reseta. Ako upalimo nadzorno računalo ono će odmah ići u 1. način rada, gdje će Bootstrap program provjeriti dali se u memoriji zaista nalazi program i tada će sam preći u 2. način rada. Na slici 3.6. prikazano je memorijsko područje u oba načina rada.



Slika 3.6. Memorijska područja

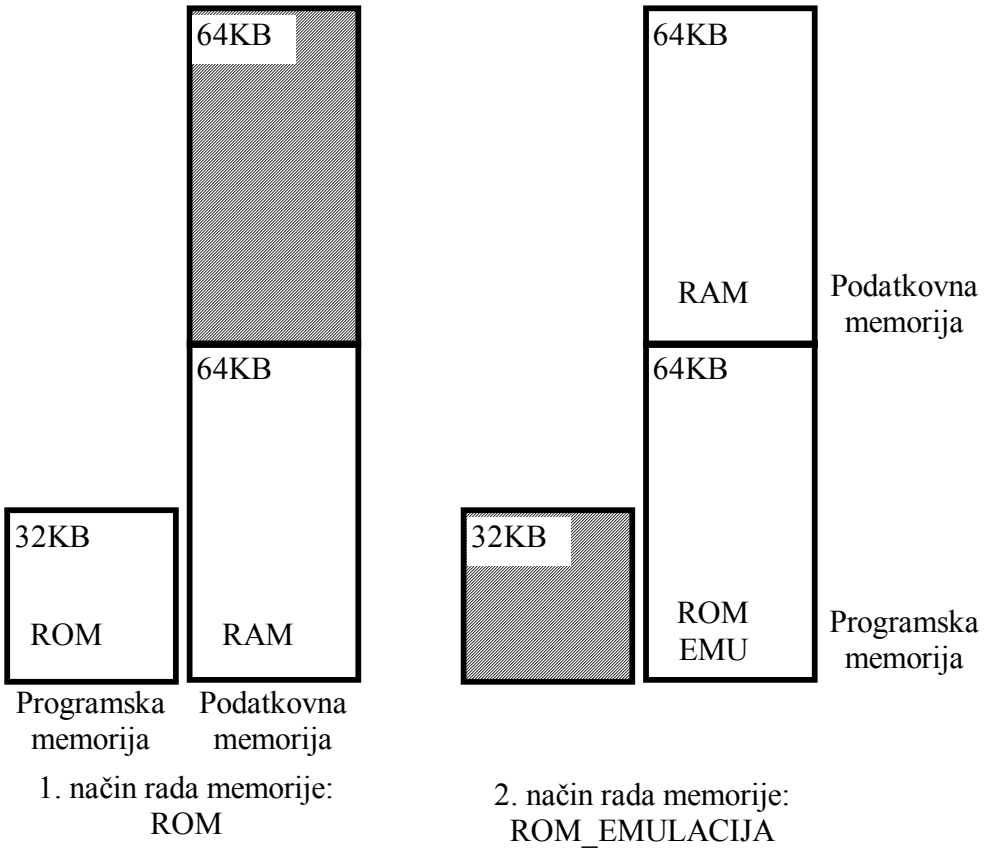
Kako se ovdje ustvari radi o 3 fizičke memorije:

- Stvarna programska memorija (EPROM), veličine 32KB
- Emulirana programska memorija (RAM), veličine 64KB
- Stvarna podatkovna memorija (RAM), veličine 64KB

Radi jednostavnosti korištene su fizički 2 memorije:

- ROM memorija (EPROM), veličine 32KB
- emulirana programska i podatkovna memorija (RAM), veličine 128KB

Na ovaj način je postignuta ušteda prostora, energije i jednostavnosti dizajna. RAM memorija od 128KB se podijeli na 2 dijela od kojih je jedan prostor za emulaciju programske memorije, a druga polovina je prostor za podatkovnu memoriju. Izgled i način korištenja memorija je prikazan na slici 3.7.



Slika 3.7. Izgled i konfiguracija memorije.

Prilikom dizajna popratnog sklopovlja za emulaciju ROM memorije, morao sam voditi računa o nekoliko kriterija. Prvi i osnovni je brzina rada popratnog sklopovlja. Dakle, vremensko kašnjenje signala kroz sklopovlje za preklapanje memorija mora biti što je moguće kraće, i približno jednakog kašnjenja na svim izlazima, kako ne bi došlo do lažnih impulsa i do pogrešnog rada. Nadalje, morao sam voditi računa o tome, da se preklapanje ne smije izvesti u bilo kojem trenutku. Tko zna što bi se dogodilo ako bi preklapanje bilo u nekom strojnom ciklusu, ili pak bilo gdje unutar strojnog ciklusa. Zato sam išao na vrlo pouzdan način, a to je preklapanje memorija isključivo za vrijeme reset signala (/RSTOL). Za to vrijeme procesor ne poziva memoriju, i izvršavanje programa će se nastaviti od samog početka. Dakle, u tom slučaju sve je pouzdano i zna se što će se dogoditi. Nadalje, trebalo je osigurani preklapanje memorija samo onda kada to korisnik, odnosno program zatraži. Ovdje glavnu ulogu ima signal «/ROMEMU_SWP».

Korisnik, odnosno program, kada želi preći u drugi način rada memorija, jednostavno aktivira impulsom ovaj signal, i pusti nadzorni sklop (eng. Watchdog) da izvrši reset nakon isteka vremena. Nakon reseta prelazi se u drugi način rada.

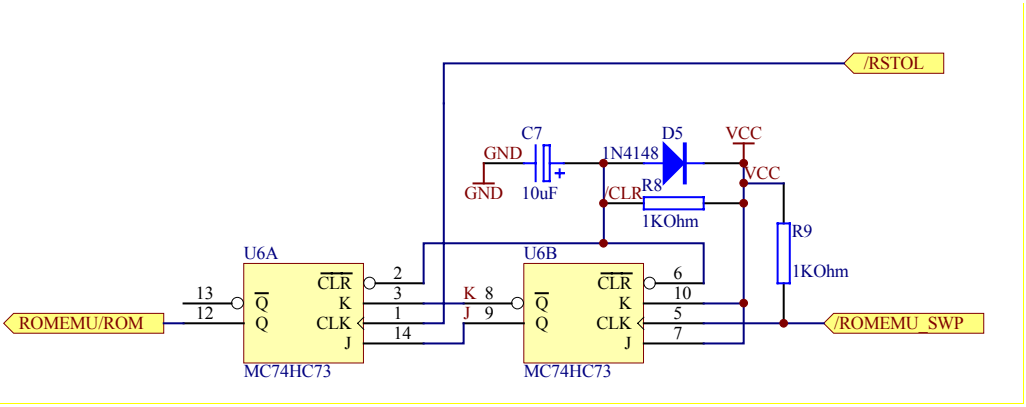
E sada, kako bi se dobio signal «ROMEMU/ROM» iz ova dva prethodno navedena signala, i uz prethodno navedene uvjete, koristio sam 2 JK bistabila u nizu. Kod ovog signala nije presudna brzina, jer reset procesora traje izvjesno dugo vrijeme. Isto tako, prilikom paljenja nadzornog računala, potrebno je osigurati da se nadzorno računalo nalazi u 1. načinu rada, odnosno da je signal «ROMEMU/ROM» u logičkoj 0. To se postiže RC mrežom na signalu «/CLR» na oba JK bistabila, čime se forsira brisanje bistabila prilikom dolaska napona napajanja.
Kada programer odnosno program želi preći iz jednog načina rada u drugi, tada pošalje impuls procesorom na signal «/ROMEMU_SWP». Znači, program postavi logičku 0 na izlaz procesora,

port5 bit 6, zadrži ju kratko vrijeme, i onda vrati natrag logičku 1. Time je aktiviran prvi u nizu JK bistabil. Kako su JK ulazi prvog bistabila stalno u logičkoj jedinici, na svaki okidni impuls signala «/ROMEMU_SWP» prvi bistabil će promijeniti stanje izlaza. Drugi JK bistabil će proslijediti to stanje onda kada se na njegovom ulazu pojavi impuls. A impuls će se pojaviti kada procesor ode u stanje reseta, odnosno kada signal «/RSTOL» ode u logičku 0.

Znači, program kada zatraži promjenu načina rada memorija impulsom na liniji «/ROMEMU_SWP», također mora izazvati i reset procesora. Reset se postiže jednostavno tako da se dozvoli istek vremena brojila nadzornog sklopa.

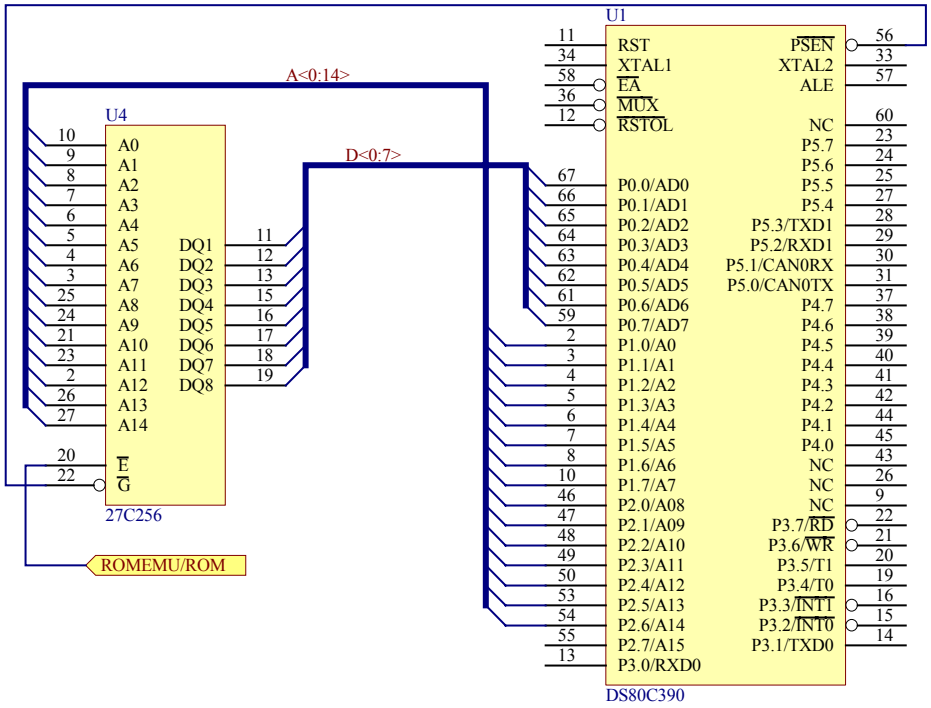
Nadzorni sklop (eng. Watchdog) je napredna funkcija mikroprocesora, koja ima ulogu kontrole ispravnosti rada programa. Program, ako ispravno radi, mora osigurati resetiranje brojila nadzornog sklopa, kako se ne bi izveo reset procesora. Ako iz bilo kojeg razloga program ne bi ispravno radio, ili ako bi se program našao u mrtvoj petlji zbog neke pogreške, resetiranje brojila nadzornog sklopa ne bi bilo osigurano, i nakon nekog vremena, nadzorni sklop bi napravio reset procesora. Ovim načinom se osigurava da procesor uvijek radi ispravno.

Kako bih dobio preklapanje memorija točno u određenom trenutku i to točno prilikom reseta procesora, iskoristio sam svojstvo nadzornog sklopa. Isječak sheme sa bistabilima i pripadnim signalima, prikazan je na slici 3.8.



Slika 3.8. Spoj JK bistabila i dobivanje signala «ROMEMU/ROM».

E sada, signal «ROMEMU/ROM» diktira način rada. Za preklapanje memorija se brinu 2 multipleksora 4 na 1. Multipleksori su interesantniji od diskretnih logičkih funkcija upravo zbog približnih vremena kašnjenja signala sa ulaza na izlaz. Isto tako, stavljanjem više logičkih NI vrata u niz, odziv će biti sporiji nego kod multipleksora. Kako su redovito EPROM memorije sporije od SRAM memorija, EPROM je spojen na signal «/PSEN» i «ROMEMU/ROM» direktno bez popratne logike. Čak štoviše, ovo je bilo vrlo lako zadovoljiti upravo iz jednostavnog kriterija da se EPROM koristi samo kao programska memorija (signal /PSEN), i samo onda kada je prvi način rada. Na slici 3.9. prikazan je spoj EPROM memorije na potrebne signale.



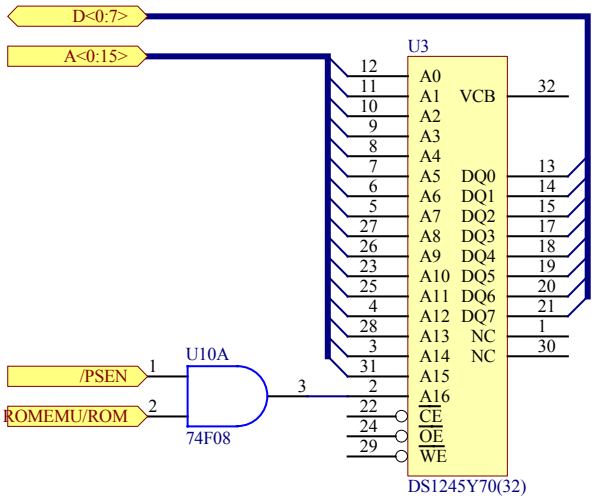
Slika 3.9. Spoj EPROM memorije i potrebni signali.

Problem je predstavljalo preklapanje statičke memorije (SRAM), koja je radila u nekoliko režima rada, i sa različitim područjima rada (prvih 64KB ili drugih 64KB).
Nižih 64KB se koristi za programsku memoriju, što je već opisano u prethodnom tekstu. Viših 64KB se koristi za podatkovnu memoriju. Koja polovina memorije će biti adresirana određuje signal «A16».

Dakle, ako se radi o 1. načinu rada, tada se SRAM memorija ponaša kao podatkovna memorija i u nju se upisuju program. Naravno tada se radi o nižih 64KB, što znači da signal «A16» mora biti u 0. Isto tako, na memoriju se prosljeđuju samo signali «/RD» i «/WR» jer memorija radi samo kao podatkovna. Slično je sa dobivanjem signala «A16» i u drugom načinu rada. No nešto je kompliciranije, jer se signal «A16» mijenja u ovisnosti o signalu «/PSEN». Dakle, kada procesor želi pristupiti programskoj memoriji (signal «/PSEN» aktivan u logičkoj 0) tada signal «A16» mora biti u 0, kako bi se adresirala donja polovina memorije, dakle prvih 64KB. Kada je «/PSEN» signal neaktivan, onda se sigurno radi o čitanju podatkovne memorije, što znači da se treba adresirati gornja polovina SRAM memorije, dakle signal «A16» mora biti u 1. Na slici 3.10. je prikazan način dobivanja signala «A16» sa diskretnom logikom, dok se u tablici 3.1. vide stanja signala, način rada memorije i akcija koja se pri tome poduzima.

Način rada memorije	1 ROM	1 ROM	2 ROMEMU	2 ROMEMU
ROMEMU/ROM	0	0	1	1
/PSEN	0	1	0	1
A16	0	0	0	1
Akcija	Čitanje bootstrap programa iz ROM-a	Čitanje ili pisanje korisničkog programa u donju polovinu RAM-a	Čitanje korisničkog programa iz donje polovine RAM-a	Čitanje ili pisanje podataka u gornju polovinu RAM-a

Tablica 3.1. Dobivanje signala A16 i te pripadne akcije nad memorijom.



Slika 3.10. Način dobivanja signala «A16» diskretnim logičkim krugovima.

No nešto je kompliciranije sa signalima «/RE» i «/WE», kako ne bi bilo zabune na shemi su označeni kao signali «/OE1» i «/WE». Kako dobiti iste? Jednostavno.
Kada se radi o 1. načinu rada, onda se na SRAM prosljeđuju samo signali za čitanje i pisanje podatkovne memorije, dakle «/RD» i «/WR». Znači, signal «ROMEMU/ROM» je u logičkoj 0, čime je izlaz iz drugog NI logičkog sklopa (U11B) uvijek u logičkoj 1, te se signal «/RD» slobodno prosljeđuje kroz logički sklop I (U10B) na «/OE1», dok je prolaz signala «/PSEN» na «/OE1» blokiran.

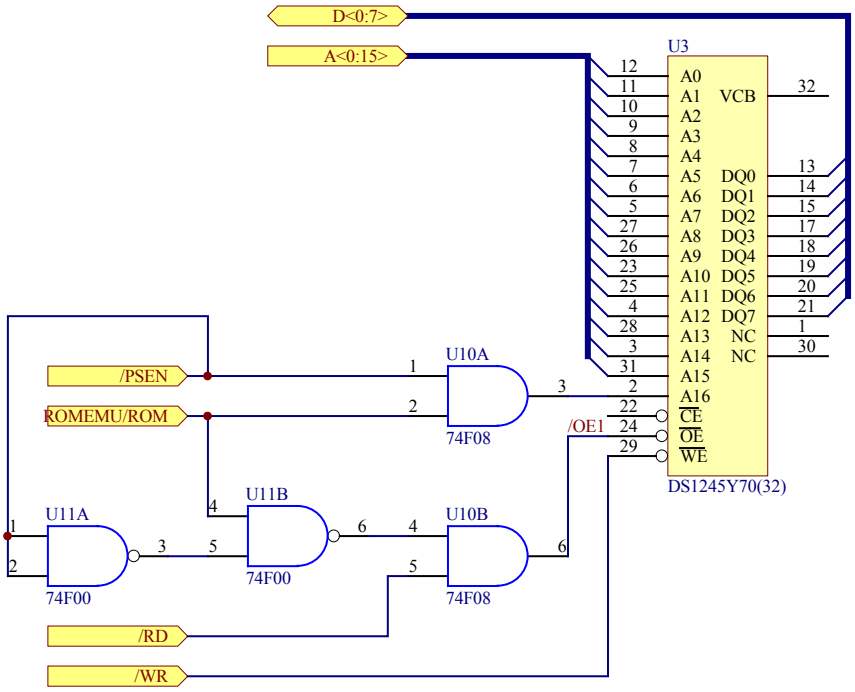
No u 2. načinu rada, kada je signal «ROMEMU/ROM» u logičkoj 1, na «/OE1» prolaze oba signala, i «/PSEN» i «/RD». Koja će akcija biti izvedena ovisi o «/PSEN» signalu, dakle, dali se radi o čitanju programske memorije, na signal «/OE1» se prosljeđuje signal «/PSEN», odnosno, ako je «/PSEN» neaktivan (logička 1), tada se radi o čitanju ili pisanju u podatkovnu memoriju. Znači u tom slučaju se na signale «/OE1» i «/WE» prosljeđuju signali «/RD» i «/WR», pri čemu je signal «A16» u logičkoj 1. Dalje, studijom signala iz procesora, zaključio sam kako se signal «/WR» pojavljuje samo u jednom trenutku i samo kada se piše u podatkovnu memoriju, što znači da se signal «/WR» može trajno spojiti na «/WE». Shema u diskretnoj logici prikazana je na slici 3.11., dok su stanja signala, te pripadne akcije opisane u tablici 3.2. i 3.3.

Ulazi	/PSEN	0	1	1	1
	/RD	1	0	1	1
	/WR	1	1	0	1
Izlazi	A16	0	0	0	0
	/OE1	1	0	1	1
	/WE	1	1	0	1
Akcija		Čitanje bootstrap programa iz ROM-a	Čitanje korisničkog programa iz donje polovine RAM-a	Upis korisničkog programa u donju polovinu RAM-a	Nema akcije

Tablica 3.2. Stanja signala i pripadne akcije kada je memorija u 1. načinu rada

Ulazi	/PSEN	0	1	1	1
	/RD	1	0	1	1
	/WR	1	1	0	1
Izlazi	A16	0	1	1	1
	/OE1	0	0	1	1
	/WE	1	1	0	1
Akcija		Čitanje korisničkog programa iz donje polovine RAM-a	Čitanje podataka iz gornje polovine RAM-a	Upis podataka u gornju polovinu RAM-a	Nema akcije

Tablica 3.3. Stanja signala i pripadne akcije kada je memorija u 2. načinu rada

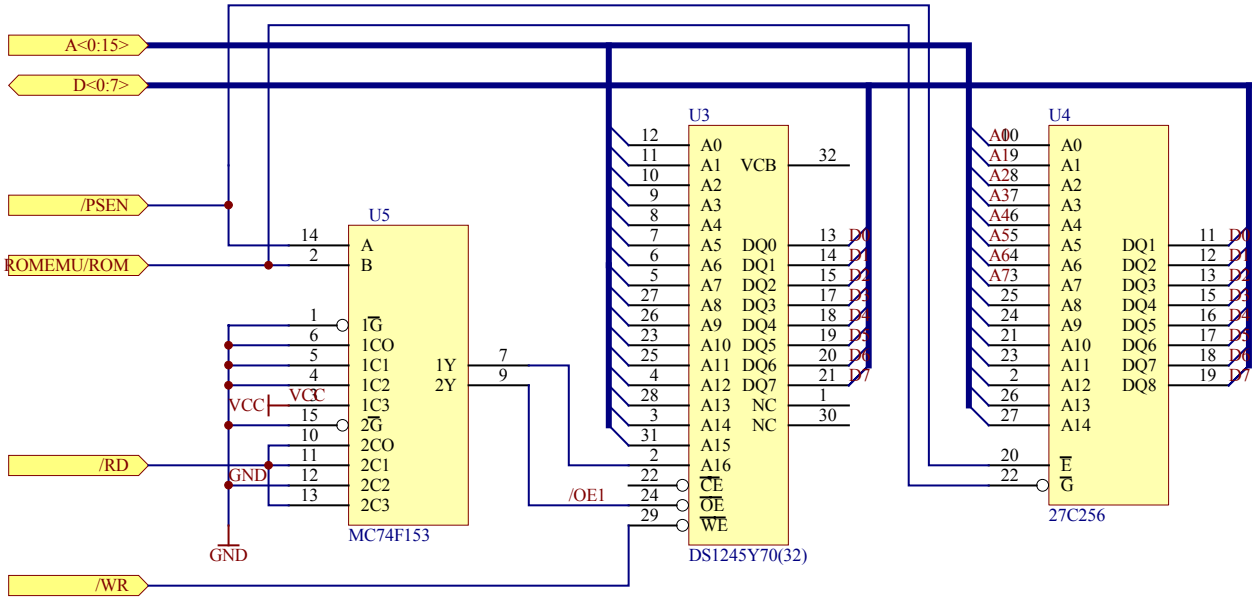


Slika 3.11. Dobivanje signala «/OE1» i «/WE» na SRAM

Gornje sheme na slikama 3.9. do 3.11. prikazane su u diskretnoj logici zbog lakšeg shvaćanja principa rada. Isto su tako prikazane i tablice 3.1. do 3.3. No, kako sam već naveo zbog različitih vremena kašnjenja signala kroz niz logičkih sklopova, bio sam prisiljen iste izvesti sa multipleksorom. Navedena shema svih signala prikazana je na slici 3.12. Na slici se vrlo lijepo vidi kako multipleksor prosljeđuje podatak sa jednog od 4 ulaza na izlaz, a koji će biti proslijeđen, ovisi o signalima «/PSEN» i «ROMEMU/ROM». Pripadni signali za multipleksor su prikazani u tablici 3.4.

ROMRMU/ROM	0	0	1	1	Odabir ulaza
/PSEN	0	1	0	1	
A16	0	0	0	1	Proslijedi na izlaz
/OE1	/RD	/RD	0	/RD	

Tablica 3.4. Pripadni signali na multipleksorima.



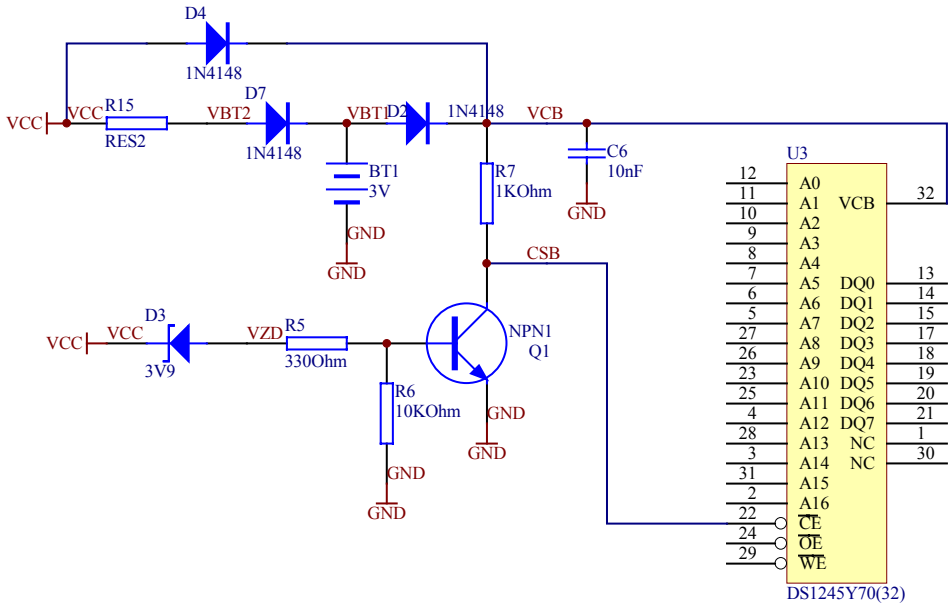
Slika 3.12. Shema logike za dobivanje potrebnih signala na memorijama.

3.3. Čuvanje podataka u statičkoj memoriji prilikom nestanka napona napajanja

Najjednostavniji način čuvanja podataka u statičkim RAM memorijama nakon nestanka napajanja je korištenje takozvane «Nonvolatile» SRAM memorije. To su SRAM memorije koje u svom kućištu sadrže rezervne izvore napajanja (baterije) i potrebnu elektroniku koja gasi memoriju i stavlja ju u «duboki san» kako bi se potrošnja energije svela na najmanju moguću razinu. Jedna od takvih memorija je i Dallas DS1245. No problem ih je nabaviti, a i imaju dosta visoku cijenu.

Stoga sam išao na dizajn sa običnom SRAM memorijom i dodavanjem potrebne elektronike koja će memoriju dovesti u «duboki san» i napajati ju iz pričuvne baterije čim nestane glavno napajanje. Potrebna dodatna logika je prilično jednostavna i zasniva se na jednom bipolarnom tranzistoru NPN tipa u spoju zajedničkog emitera, tj. u spoju prekidača. Prilikom dizajna potrebne logike najviše je pridano pažnje dodatnoj potrošnji koju unosi ta dodatna logika, i potrošnja je svedena na minimum.

Također koristi se spoznaja, kako SRAM memorije u «dubokom snu» čuvaju podatke i prilikom napona daleko manjih od napona napajanja u normalnom radu (+5V). Taj napon može pasti i do 2V. Ovakve podatke daju proizvođači memorija. U dizajnu koristim bateriju dovoljno velikog napona, koja bi trebala zadovoljiti naponske potrebe memorija većine proizvođača. Shema dodatne elektronike za čuvanje podataka prikazana je na slici 3.13.



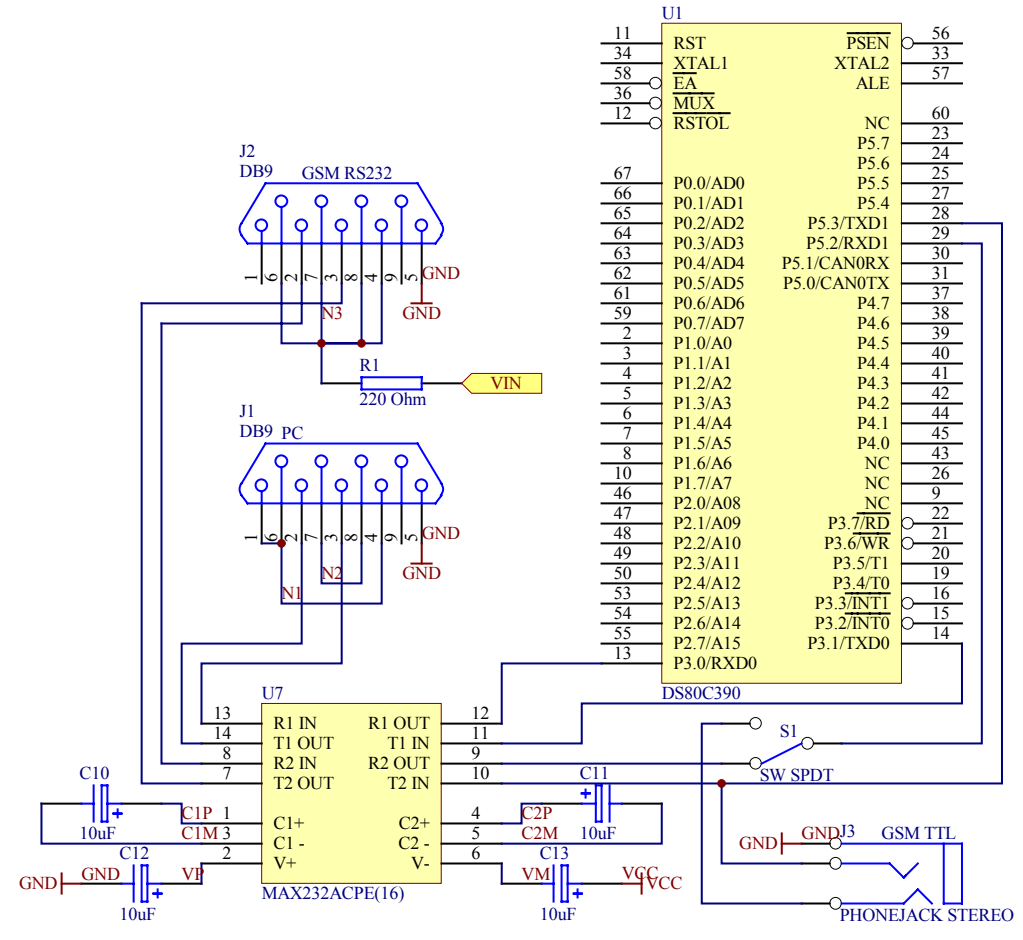
Slika 3.13. Popratna logika za čuvanje podataka u memoriji nakon nestanka napona napajanja.

Inače memorija koju sam koristio je NEC uPD431000AZ, čiji se detaljni podatci mogu naći u prilogu na CD-u u file-u pod imenom «uPD431000A.pdf». To je statička RAM memorija od 128Kbyte-a, organizirana u riječi od 8 bita, rađena u CMOS tehnologiji.

3.4. Asinkrona serijska komunikacija i popratno sklopovlje

Kao što sam već u potpoglavlju 3.1. naveo kako DS80C390 ima dvije asinkrone serijske sabirnice, ali kod njih se radi o pozitivnim TTL logičkim naponskim nivoima, gdje je logička 0 od 0V do 0.7V, a logička 1 od 2.7V do 5V. Kako je RS232 protokol po standardu izveden sa negativnim naponskim nivoima, gdje je logička 1 od –3V do –15V, a logička 0 od +3V do +15V, potrebno je napraviti konverziju signala u oba smjera. Za to se brine popularni integrirani krug MAX232 proizvođača MAXIM. Shema spajanja istog je prikazana na slici 3.14. i to je standardni spoj istog kakav preporuča sam proizvođač. Više detalja moguće je naći u prilogu na CD-u u file-u «1798 (MAX232).pdf».

Otpornik R1 je dodan zbog standardnog GSM kabela koji radi konverziju RS232 nivoa na TTL nivoe kakve koristi GSM uređaj. Konverziju radi upravo MAX232 koji se inače napaja iz RS232 konektora. Tako da upravo taj otpornik daje potreban napon za konverziju.



Slika 3.14. Shema spajanja MAX232 na mikroprocesor DS80C390.

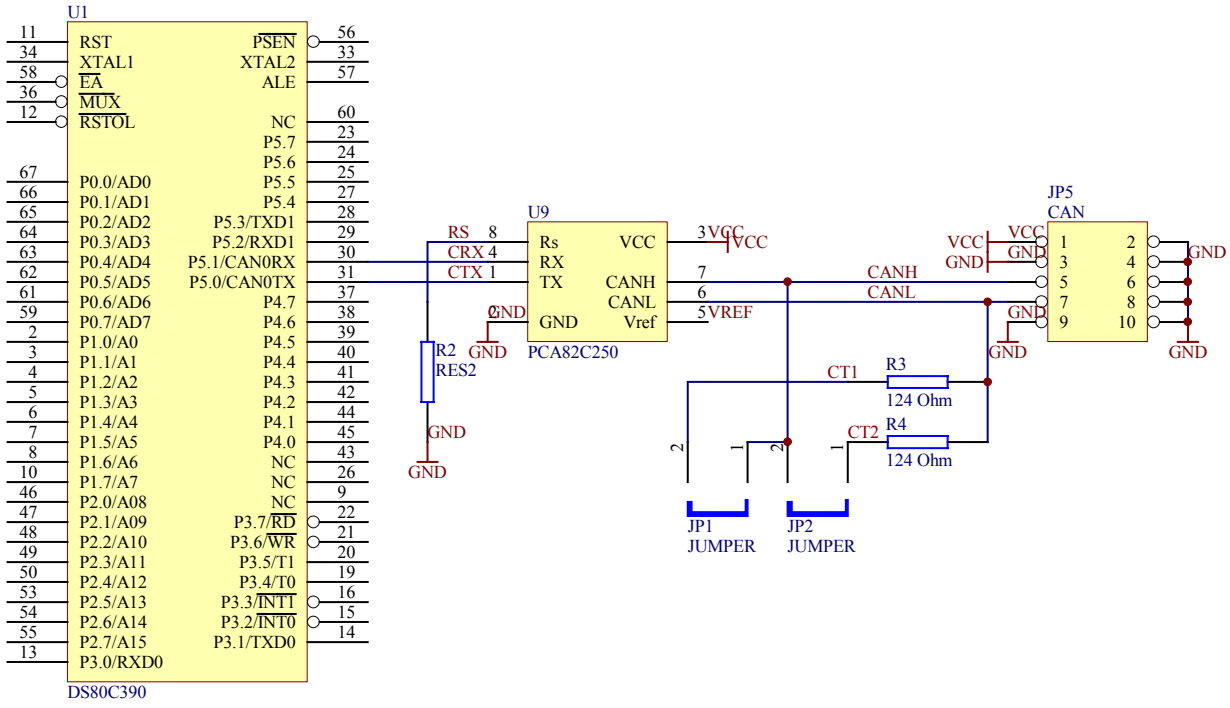
Kako sam već spomenuo da GSM uređaj koristi TTL naponske razine za komunikaciju, nije potrebno raditi dvostruku konverziju: TTL na RS232 pa onda RS232 na TTL. Zato sam u dizajnu uveo kratkospojnik (S1), kojim se odabire dali je GSM uređaj spojen na klasični RS232 konektor ili na TTL konektor.

3.5. CAN sabirnica i popratno sklopovlje

Kao što sam već u potpoglavlju 3.1. naveo kako procesor ima ugrađenu podršku za CAN sabirnicu, ona je ukomponirana u dizajn sa potrebnim dodatnim sklopovljem. Kako je CAN2.0B sabirnica serijska dvosmjerna diferencijalna sabirnica potrebno je izvesti prilagodbu nivoa signala iz procesora na CAN sabirnicu. Za to se brine integrirani krug PCA82C250 proizvođača Philips. Sve o ovom integriranom krugu može se naći u prilogu na CD-u u file-u pod nazivom «PCA82C250.pdf».

U dizajnu, koristio sam najjednostavniji spoj ovog integriranog kruga bez galvanskog odvajanja. Kako se ovdje radi o razvojnoj aplikaciji, smatrao sam kako će nadzorno računalo biti spojeno na galvanski odvojeni izvor napajanja, te stoga da nije potrebno galvanski odvajati i CAN sabirnicu. Inače, CAN sabirnica, kao što sam već naveo, je serijska diferencijalna sabirnica koja je na krajevima zaključena sa otporima od 124Ω, a svi uređaji se spajaju paralelno na sabirnicu unutar tih zaključenja. Kako se ovdje radi o razvojnoj aplikaciji, u dizajnu je omogućeno da se kratkospojnicima (JP1 i JP2) mogu uključiti jedan ili oba zaključna otpornika. Ovo je interesantno jer se prilikom razvoja obično ne koriste pune mogućnosti sabirnice nego se spoje samo 2 uređaja kako bi se provjerila ispravnost komunikacije. Isto tako radi se o vrlo malim udaljenostima između ta dva uređaja tako da nije potrebno strogo poštivati pravila o zaključenjima na sabirnici i naravno ta sabirnica fizički zaista ne postoji pa se moraju provizorno stavljati zaključni otpori koje ponekad nemamo pri ruci, itd. Stoga je jednostavno odabirom kratkospojnika dobiti iste…

Konektor (JP3) je priključni konektor na CAN sabirnicu. Na slici 3.15. je prikaza shema spajanja integriranog kruga PCA82C250 na procesor.

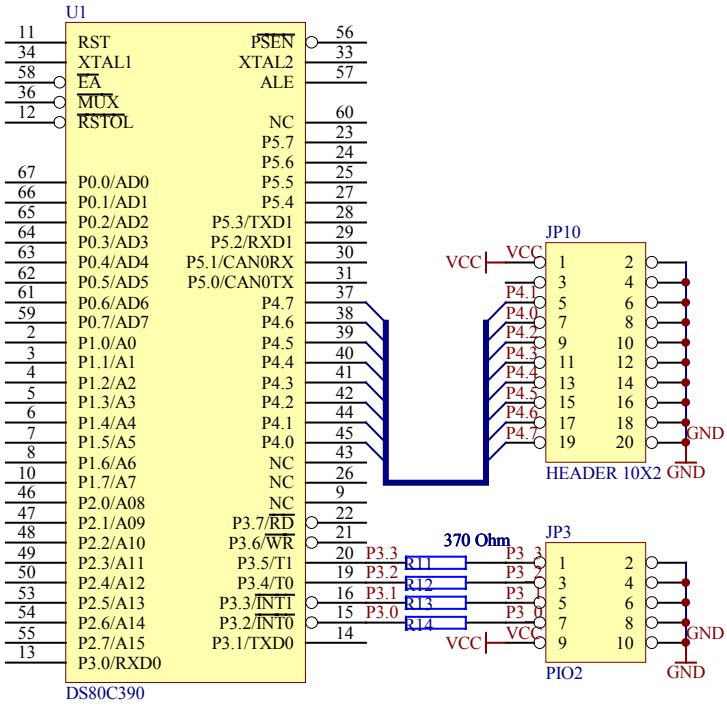


Slika 3.15. Shema popratne logike zadužene za CAN sabirnicu.

3.6. Paralelne digitalne sabirnice za spajanje na vanjski svijet

Iako procesor ima dosta veliki broj ulazno izlaznih digitalnih linija, konkretno njih 40, većina ih je potrošena na komunikaciju sa memorijom i ostalim perifernim sklopovima, popularno zvanim «glue logic». Ipak ostalo je 15 slobodnih ulazno izlaznih linija. 3 sam ostavio neiskorištene za moguće buduće preinake dizajna nadzornog računala, dok ih je 12 izvedeno za spajanje na proces putem 2 konektora JP10 i JP3. Konektor JP10 ima 8 ulazno izlaznih linija, te liniju za napajanje koja može biti potrebna mogućim senzorima i slično. Drugi konektor JP3, ima 4 ulazno izlazne linije, koje su spojene preko zaštitnih otpornika na procesor, te liniju za napajanje. Zaštitni otpornici su ovdje dosta pogodni jer se na ovaj konektor mogu spojiti direktno svjetleće diode ili slična trošila. Na ovaj način je procesor zaštićen od prevelike struje mogućeg potrošača.

Na slici 3.16. prikazana je shema spoja procesora i konektora za spajanje na proces.



Slika 3.16. Shema spoja procesora na digitalne paralelne sabirnice

Svaki bit ulazno izlaznog paralelnog porta je u biti dvosmjerni. Kako su svi bitovi bez snažnog zatezanja prema Vcc odnosno +5V, imaju takozvani «week pullup». Stoga svaki bit koji se programski postavi u logičku 1, postaje i ulazni, jer ga sklop priključen na njega može povući u logičku 0. Tako da izlazni sklop postaje ulazni, odnosno dvosmjerni. Dalje, moja preporuka je da se ukupna struja na svim bitovima ograniči na 10-tak mA, zbog prevelike discipacije snage u kristalu koja dovodi do povišene temperature mikro procesora, i mogućeg uništenja istog.

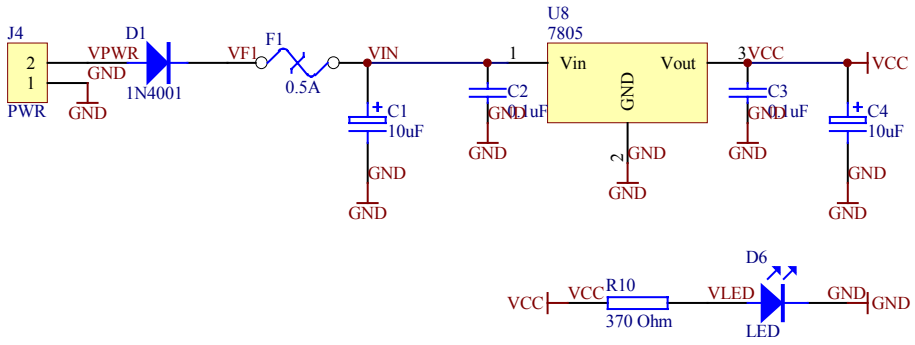
3.7. Sklop za napajanje i zaštitu

Sklop za napajanje ima nekoliko zaštita nadzornog računala, i to isključivo zbog ljudskog faktora pogreške. Tako sklop posjeduje zaštitnu diodu D1 od spajanja vanjskog napajanja pogrešnog polariteta, te osigurač F1 isto tako zbog mogućeg kratkog spoja koji može izazvati korisnik.

Ujedno sklop za stabilizaciju napona je dobro poznati 7805, koji se brine za održavanje napona procesora i ostalih elemenata na 5V. Ulazni napon se može kretati u granicama od 9V do 24V, ali ga je poželjno držati do najviše 15V zbog povećane discipacije snage i temperature stabilizatora ako je i ulazni napon povećan.

Tu su još kondenzatori C1, C2, C3, C4 zbog dodatne zaštite od smetnji i oscilacija, te dodatnog filtriranja napona napajanja. Otpornik R10 i svjetleća dioda D6 služe isključivo u signalizacijske svrhe, kako bi korisnik znao kada nadzorno računalo radi.

Shema sklopa za napajanja, stabilizaciju, i zaštitu prikazan je na slici 3.17.



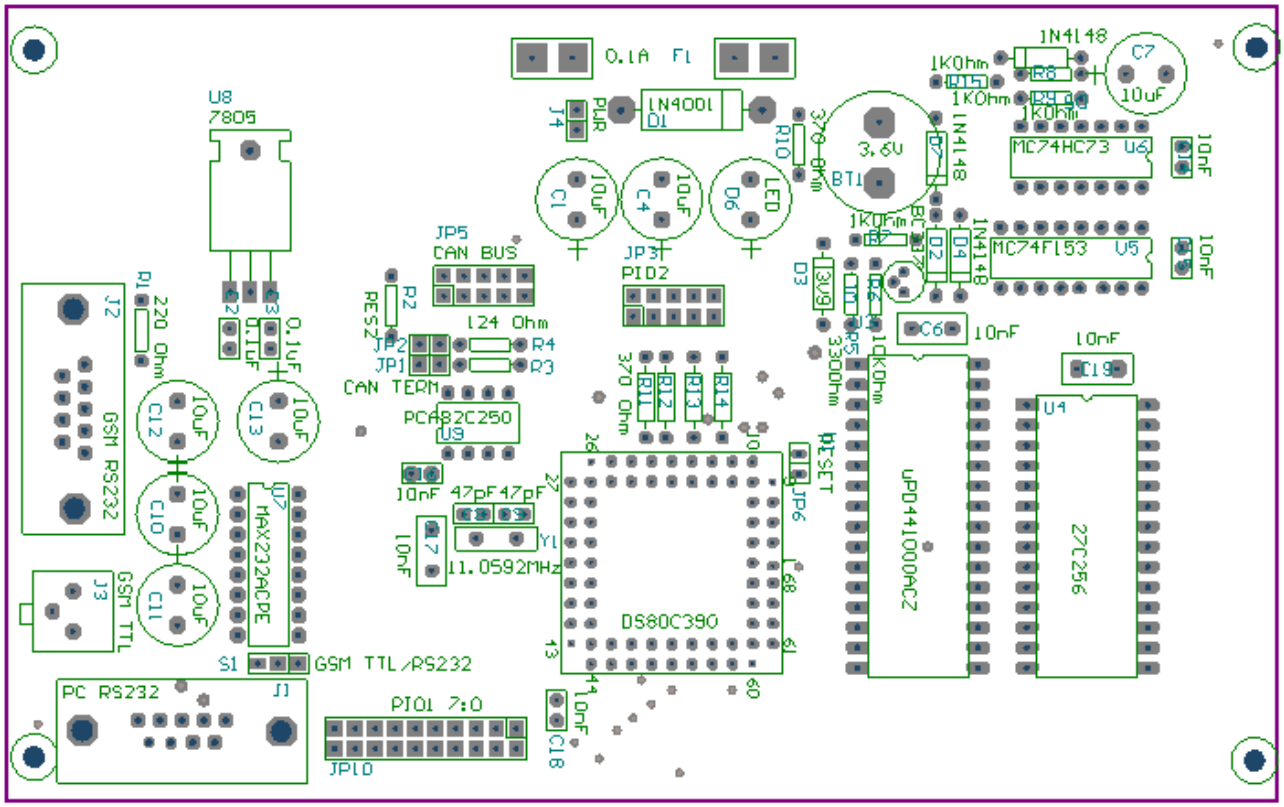
Slika 3.17. Shema sklopa za napajanje i zaštitu nadzornog računala.

Praktičnom provjerom sklopa i sheme za napajanje, moja je preporuka da se koristi slabiji osigurač F1. Preporučujem korištenje osigurača od 0,1A. Kako čitav uređaj u normalnom radu troši 80mA, a u modu štednje energije oko 40mA, stoga će osigurač od 100mA spriječiti moguće uništenje procesora prilikom korisnikovog nestručnog rada sa uređajem.

3.8. Izvedba nadzornog računala

Čitav uređaj je izveden na dvoslojnoj tiskanoj pločici dimenzija 160x100 mm, koje su ujedno standardne «Euro card» dimenzije. Sam uređaj se može izvesti nešto manjih dimenzija, ali kako je tiskana pločica izvedena amaterski, bilo je poželjno napraviti što deblje vodove, kako bi bile veće šanse ispravne izvedbe tiskane pločice, i ispravnog rada samog uređaja.

Kako je ovo prva inačica tiskane pločice, tako na istoj ima nekih fizikalno prostornih problema koji su vrlo jednostavno riješeni. No sam dizajn sheme uređaja je ispravan, tako da neću ulaziti u detaljnije rasprave oko kozmetičkih problema tiskane pločice. Izgled, tj. izvedba nadzornog računala prikazana je na slici 3.18.

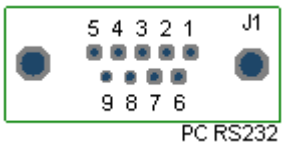


3.18. Izvedba nadzornog računala

Na nadzornom računalu nalazi se nekoliko konektora za spajanje računala na vanjski svijet. To su konektor za spajanje na PC (J1), odnosno terminal, zatim konektor za spajanje na GSM uređaj putem RS232 nivoa (J2), konektor za spajanje GSM uređaja putem TTL nivoa (J3), kratkospojnik za odabir načina spajanja GSM uređaja (S1), konektor za spajanje na izvor napona napajanja (J4) koje može biti od 8V do 24V, konektor za spajanje na CAN sabirnicu (JP5), zatim kratkospojnici (JP1, JP2) za terminaciju CAN sabirnice, konektor za spajanje paralelnih digitalnih portova 1 i 2 (JP10 i JP3), i na kraju kratkospojnik za spajanje reset tipkala (JP6). U daljnjem tekstu biti će obrađen svaki konektor zasebno.

a) Izvedba konektora za spajanje na PC odnosno terminal (J1) prikazana je na slici 3.19. Kako se vidi sa slike, za spajanje su bitna samo 3 signala: slanje podataka, primanje podataka i uzemljenje. Konektor je izveden kao ženski DB9 konektor koji se spaja direktno na osobno računalo (PC) ili pomoću produžnog kabela. Ostali signali su samo kontrolni i potrebni su za uspostavu komunikacije takozvani «handshake» između terminala i nadzornog računala.

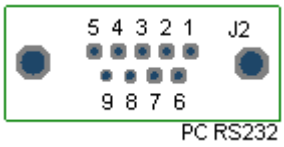
Slijedeći izvodi su spojeni zajedno: 1,6,4, te 7,8. Postavke terminala su 19200bps bez bitova pariteta i jedan stop bit. Smjer je relativan u odnosu na nadzorno računalo.



Broj izvoda	Smjer signala	Opis
1		Detekcija nositelja podataka
2	→	Slanje podataka
3	←	Primanje podataka
4		Terminal spreman
5	—	Uzemljenje
6		Podatak spreman
7		Zahtjev za slanje
8		Komanda za slanje
9		Indikacija zvona

Slika 3.19. Izvedba konektora za spajanje na PC odnosno terminal.

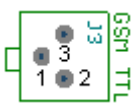
b) Izvedba konektora za spajanje na GSM uređaj putem serijskog asinkronog protokola RS232 naponskih nivoa (J2) prikazana je na slici 3.20. Konektor je izveden kao muški DB9. Komunikacija sa GSM uređajem je istih parametara kao i sa osobnim računalom. Jedino se ovdje ne koriste nikakve linije za uspostavu komunikacije. Linije 6,7,8 i 4 su jednostavno izvor napajanja za GSM spojni kabel koji u sebi obično ima uređaj za konverziju naponskih nivoa na TTL nivoe s kakvima obično rade svi GSM uređaji. Konverzija je obično bazirana na MAX232 integriranom krugu. Kao što se može primijetiti, ovdje se radi dvostruka konverzija, prvo sa TTL nivoa procesora na RS232, i onda u kabelu sa RS232 na TTL nivoe s kakvim radi GSM uređaj. Kako bi se izbjegla dvostruka konverzija na nadzornom računalu postoji slijedeći konektor.



Broj izvoda	Smjer signala	Opis
1		
2	←	Primanje podataka
3	→	Slanje podataka
4		+V
5	—	Uzemljenje
6		+V
7		+V
8		+V
9		

Slika 3.20. Izvedba konektora za spajanje na GSM uređaj putem RS232 naponskih nivoa.

c) Izvedba konektora za spajanje na GSM uređaj putem serijskog asinkronog protokola TTL naponskih nivoa (J3) prikazana je na slici 3.21. Sve je potpuno jednako kao i kod prethodnog konektora, jedina je razlika u tome što su naponski nivoi TTL razine, tako da spojni kabel ne treba imati nikakve dodatne sklopove za konverziju.



Broj izvoda	Smjer signala	Opis
1	—	Uzemljenje
2	←	Primanje podataka
3	→	Slanje podataka

Slika 3.21. Izvedba konektora za spajanje na GSM uređaj putem TTL naponskih nivoa.

d) Izvedba kratkospojnika (S1) kojim se odabire način komunikacije prema GSM uređaju je prikazana na slici 3.22.

Izvedba konektora	Spoj izvoda	Odabir komunikacije
	1-2	TTL
	2-3	RS232

Slika 3.22. Izvedba kratkospojnika za odabir spajanja GSM uređaja.

e) Izvedba konektora za spajanje napona napajanja (J4) prikazana je na slici 3.23.

Izvedba konektora	Spoj izvoda	Opis
	1	+8V do +24V
	2	Uzemljenje

Slika 3.23. Izvedba konektora za spajanje napona napajanja.

f) Izvedba konektora za spajanje na CAN sabirnicu (JP5), prikazan je na slici 3.24.

Izvedba konektora	Broj izvoda	Opis
	1	+5V izvor
	2,3,4,6,8,9,0	Uzemljenje
	5	CAN H
	7	CAN L

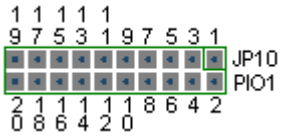
Slika 3.24. Izvedba konektora za spajanje na CAN sabirnicu.

g) Izvedba kratkospojnika za uključivanje zaključnih otpora na CAN sabirnici (JP1,JP2) prikazani su na slici 3.25. Svaki kratkospojnik spaja po jedan otpor od 124Ω.

Izvedba konektora	Spoj izvoda	Odabir komunikacije
	1-2	Zaključni otpor 1
	3-4	Zaključni otpor 2

Slika 3.25. Izvedba kratkospojnika za terminaciju CAN sabirnice.

h) Izvedba konektora paralelnih digitalnih ulaza izlaza, PIO1 (JP10) i PIO2 (JP3), prikazani su na slici 3.26. Oba porta su dvosmjerni. Kada program postavi logičku 1 na neki izvod, tada ta linija postaje ulazna i izvana se može povući na logičku 0. Dakle, kako ne bi došlo do oštećenja procesora, sklop koji se spaja na ulazni izvod nadzornog računala mora biti izveden po tehnologiji otvorenog kolektora. Isto tako potrošači koji se spajaju na paralelne izvode ne bi trebali svi zajedno preći ukupnu struju od 20mA, kako ne bi došlo do pregrijavanja kristala procesora i njegovog uništenja. PIO2 ima zaštitne serijske otpore zbog ograničenja struje, no svejedno treba obratiti pozornost na maksimalnu struju potrošača.

Izvedba konektora	Broj izvoda	Funkcija
	2,4,6,8,0	Uzemljenje
	9	Izvor napona +5V
	1	PIO2_3
	3	PIO2_2
	5	PIO2_1
	7	PIO2_0
	2,4,6,8,10,12,14,16,18,20	Uzemljenje
	1,3	Izvor napona +5V
	19	PIO1_7
	17	PIO1_6
	15	PIO1_5
	13	PIO1_4
	11	PIO1_3
	9	PIO1_2
	7	PIO1_1
	5	PIO1_0

Slika 3.26. Izvedba konektora paralelnih digitalnih ulaza izlaza PIO1 i PIO2.

Ovime završavam poglavlje čvrstog dijela nadzornog računala, te prelazim na programsku podršku za nadzorno računalo.

4. PROGRAMSKA PODRŠKA UNUTAR PROJEKTA – POKRETAČKI PROGRAM

Veoma bitna funkcija nadzornog računala je upravo ROM emulacija. Ovakav način rada omogućuje vrlo jednostavno punjenje memorije najnovijom inačicom izvršnog programa prilikom razvoja programske podrške. Punjenje programa se odvija preko asinkrone serijske komunikacije kojom je inače nadzorno računalo spojeno na osobno računalo. Ipak, kako bi se mogla memorija napuniti najnovijim izvršnim programom preko serijske sabirnice, mikro procesor je potrebno pokrenuti nekim programom, koji je zadužen za punjenje memorije i izvršenje programa koji se nalazi u toj memoriji. To je pokretački program koji se popularno naziva «Bootstrap loader». Pokretački program se jednom pohrani u ROM odnosno EPROM memoriju i potpuno je univerzalan, dakle ima jedinstvenu funkciju samog punjenja i potpuno je svejedno za koju primjenu se koristi samo nadzorno računalo. Ovaj program mora biti što je moguće jednostavniji kako bi bio pouzdaniji u radu, bez mogućih slučajnih pogrešaka.

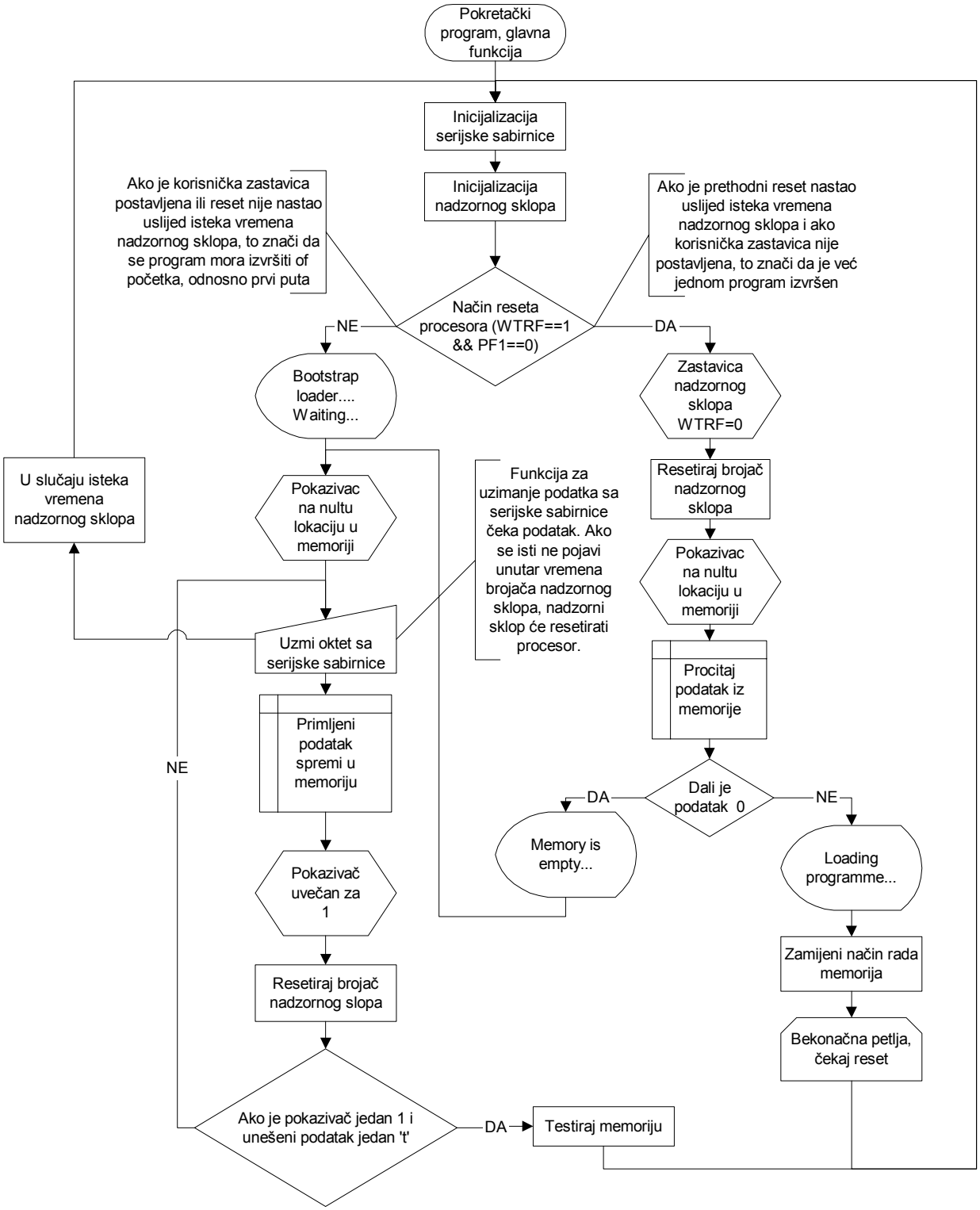
Inačica pokretačkog programa koja se koristi je 1.1. koja potpuno zadovoljava zadane kriterije. Također, istom sam dodao funkciju provjere dijela memorije u koju se pohranjuje izvršni program. Ipak, postoji jedno ograničenje. Kako bi izvršni program potpuno pouzdano radio, ne smije biti veći od 60KB. Iako je veličina memorije rezervirane za izvršni program nešto veća i iznosi 64KB, zadnjih 4KB se ne smije koristiti, pretpostavljam iz razloga što mikro procesor 80C390 posjeduje 4KB interne memorije koja se mapira na zadnja 4KB unutar 64KB vanjske memorije.

Također pretpostavljam da bi se ovaj nedostatak mogao riješiti daljnjim razvijanjem pokretačkog programa i traženjem posebnih postavki za internu memoriju koje specificira proizvođač procesora, možda je moguće istu memoriju isključiti kada procesor izvršava pokretački program. Na taj način bi se moglo koristiti sva 64KB za izvršni program. No kako je izvršni program razvijen za svrhe nadzornog računala manji od 16KB, ovo ograničenje ne predstavlja nikakav problem, te bi daljnji razvoj pokretačkog programa bi bespotreban gubitak vremena.

Pokretački program, inicijalizira asinkroni serijski port, memoriju, nadzorni sklop, i pošalje obavijest korisniku o spremnosti prijema izvršnog programa. Pokretački program zatim pričeka nekoliko sekundi da mu korisnik pošalje program, ako ga korisnik ne pošalje, tada pokretački program provjeri dali u memoriji već postoji program (koji je ostao od prethodnog punjenja, jer se podatci u memoriji čuvaju i nakon nestanka napona napajanja) te ga izvrši. Ako isti ne postoji, onda pokretački program čeka beskonačno dugo da mu korisnik pošalje izvršni program. Dijagram toka programa iz kojega se ovo vidi je prikazan na slici 4.1.

Za vrijeme slanja programa preko serijskog porta, pokretački program uzima podatke i zapisuje ih slijedno u memoriju počevši od nulte lokacije pa sve dok se program šalje. Kada završi prijenos, pokretački program izvršava izvršni program.

U narednim potpoglavljima biti će opisane gotovo sve funkcije pokretačkog programa, dok se ispis cijelog pokretačkog programa nalazi u prilogu 3. Program je pisan u ANSI C jeziku, a koristi se Keil MicroVision2 programski paket sa C prevodiocem. Demo inačica programskog paketa nalazi se na CD-u u prilogu.



Slika 4.1. Dijagram toka glavne funkcije pokretačkog programa.

4.1. Glavna funkcija pokretačkog programa

Ova funkcija se pokreće prilikom izvršavanja pokretačkog programa. Funkcija ima nekoliko osnovnih dijelova koji će biti manje-više opisani kako rade. Stoga, prvi dio funkcije inicijalizira varijable *pokazivač i varijabla koje su bitne za upis primljenog programa preko serijskog porta na određenu lokaciju u memoriji.

Nakon toga se pozove funkcija za inicijalizaciju serijske sabirnice i nadzornog sklopa (eng. Watchdog timer).

```
main () {

  volatile unsigned char xdata *pokazivac;
  unsigned char data varijabla;

  ser_init();                // Inicijalizacija serije
  wd_i();                    // Uključivanje Watch Dog-a
```

Sada se provjerava zastavica nadzornog sklopa i korisnička zastavica kako bi se ustanovio razlog prethodnog reseta procesora. Stoga ako je prethodni reset uslijedio nakon isteka vremena nadzornog sklopa izvršiti će se naredni kod, koji će prvo resetirati nadzorni sklop kako se ne bi dogodio neželjeni reset procesora. Zatim će uzeti podatak sa nulte lokacije iz memorije u koju se sprema izvršni program i spremiti ga u varijablu.

```
  if (WTRF==1 && PF1==0) {   // Izvršava se nakon WDT reseta procesora
     WTRF = 0;
     wd_r();                 // Resetiraj WDT
     pokazivac=0;            // Postavi pokazivac na početnu lokaciju RAM-a
     varijabla=*pokazivac;   // Uzmi podatak iz RAM-a sa te lokacije
```

Ako je varijabla jednaka 0, to znači da je memorija prazna, što znači da ne postoji izvršni program u memoriji. Ako je tako, obavijestiti ćemo korisnika o tome i izaći iz svih «if» petlji te nastaviti glavnu funkciju

```
     if (varijabla == 0) {   // Ako je ta lokacija prazna, Program ne
                             // postoji i ponovno ćemo ući u petlju za
                             // prijenos programa preko serije
           printf("\n Memory is empty... \n");
           printf(" Please, flash memory...\n");
```

Ako nešto ipak ima u memoriji, tada obavijesti o tome korisnika, zatim pozove funkciju za prebacivanje načina rada memorija. Kako smo trenutno u 1. načinu rada, nakon reseta procesora preći ćemo u 2. način rada memorija. I tada se u beskonačnoj petlji zaustavi izvršavanje programa, kako bi se pričekao istek vremena nadzornog sklopa koji će tada izvršiti reset procesora.

```
     } else {                  // Ako nešto ima u memoriji
           printf("\n Loading programme...\n");
           romemu_swap();      // Naredbu za prebacivanje memorija nakon
                               // WDT reseta procesora
           while (1);          // Čekaj da WDT resetira procesor
     }
```

Ako se pokretački program izvršava nakon uključivanja procesora, odnosno zadnji reset nije prouzročen istekom vremena nadzornog sklopa, tada se pokretački program u biti izvršava od

početka. Prvo obavijestimo korisnika o programu koji se trenutno izvršava, i damo mu osnovno instrukcije što treba napraviti.

```
  } else {                          // Izvršava se nakon pravog reseta procesora
     PF1=0;          // Dojava programu da se ne izvrši od početka
     printf ("\n\n***********************************");
     printf("***********************************\n");
     printf("********** Bootstrap loader **** v1.1");
     printf(" **** by Goran Jurkovich **********\n");
     printf("***********************************");
     printf("***********************************\n");

     printf("\n If you want to flash memory, send programme");
     printf(" within next few seconds...\n");
     printf(" If you want to test and erase memory,");
     printf(" press 't' now...\n\n Waiting...\n");
  }
```

Korisnik sada može poslati novu inačicu izvršnog programa putem serijske sabirnice, ili pritiskom na slovo 't' pokrenuti test i brisanje memorije u koju se smješta izvršni program.

Nastavak glavne funkcije.
Ovdje se izvršava beskonačna kružna petlja. No prvo je potrebno postaviti pokazivač na nultu lokaciju u memoriji.

```
  pokazivac=0;                      // Postavi početnu lokaciju u RAM-u
  while (1) {                       // Beskonačna petlja
```

Ovdje pozivamo funkciju _getkey, koja vraća podatak koji je primljen sa serijske sabirnice. No poanta je u tome da ona čeka dok se isti ne pojavi. Tako da će program stajati na ovoj lokaciji sve dok korisnik ne pošalje neku informaciju nadzornom računalu. Ako korisnik ništa ne pošalje unutar nekoliko sekundi, nadzorni sklop će resetirati procesor.

```
     varijabla=_getkey();           // Uzmi karakter sa serije0
```

Ukoliko korisnik ipak nešto pošalje, taj će se podatak spremiti na trenutnu lokaciju u memoriji na koju pokazuje pokazivač, i tada će se uvećati pokazivač, te pokazivati na slijedeću memorijsku lokaciju. Nakon toga resetiramo brojač nadzornog sklopa, kako se ne bi dogodio neželjeni reset procesora.

```
     *pokazivac=varijabla;          // Spremi ga na trenutnu lokaciju u RAM
     pokazivac++;                   // Postavi pokazivac na slijedeću lokaciju
     wd_r();                        // Resetiraj Watch Dog Timer
```

Ako se radi o prvom poslanom podatku, to znači da taj podatak može biti ili prvi oktet izvršnog programa ili naredba korisnika za test memorije. Tada provjeri dali je varijabla slovo t, ako je, pozove funkciju za test memorije.

```
     if (pokazivac==1)              // Ako je to prvi poslani karakter
           if (varijabla =='t')     // I ako je on 't'
                 mem_test();        // Tada napravi test memorije
  }
}
```

Petlja se izvršava ponovno, uzimajući slijedeći podatak za memoriju i tako sve dok se šalje izvršni program.

4.2. Funkcija za inicijalizaciju serijske sabirnice

Na mikro procesoru DS80C390 postoje dvije serijske asinkrone sabirnice kao što je već navedeno u 3. poglavlju. Te sabirnice je potrebno pokrenuti i postaviti postavke komunikacije. Obje sabirnice su postavljene jednako na isti vremenski brojač «Timer1» i komunikacija se odvija na 19200bps. Podatci se šalju u paketima od 8 bita, bez bita pariteta sa jednim stop bitom.

Način inicijalizacije je gotovo standardan i radi se prema «kuharici», a svi parametri i detalji su dani u literaturi [16], koju daje proizvođač mikro procesora.

```
void ser_init (void) {
  EA=0;                       // Onemogući prekide
  EPF1=0;                     // Onemogući Power fail prekid

  // Komunikacija preko RS232 postavljena na 19200bps 8-N-1
  SCON0 = 0x52;         // Postavljanje moda prvog serijskog porta
  SCON1 = 0x52;         // Postavljanje moda drugog serijskog porta
  PCON |= 0x80;         // Udvostručavanje brzine serijskog porta0
  SMOD_1 = 1;           // Udvostručavanje brzine serijskog porta1

  TMOD = 0x20;          // 2 za Timer1 (RS232)
  TH1 = 0xFD;           // Postavljanje reload vrijednosti za 9600
  TL1 = 0xFD;
  IP  = 0;              // prioritet
  TR1 = 1;              // Startaj Timer1

  ET1 = 0;              // Onemogući prekidnu rutinu za Timer1
  EA  = 1;              // uključi globalne interapte
  P5CNT |=0x20;         // Postavljanje drugog serijskog porta
                        // na port P5.2 i P5.3
}
```

4.3. Funkcija za promjenu načina rada memorija

Ova funkcija stvara negativan impuls na signalu «/ROMEMU_SWP» potreban da se prebaci stanje prvog JK bistabila. Način rada je opisan u 3. poglavlju.

Svi ulazno-izlazni signali na portovima mikro procesora se postavljaju u logičku 1 prilikom reseta. Kada se pozove funkcija za promjenu načina rada memorija, tada se postavi signal «/ROMEMU_SWP» u logičku 0. Petlja služi za stvaranje vremenskog kašnjenja, i tada se ponovno signal postavi u logičku 1. Na ovaj način je dobiven negativan impuls na signalnoj liniji «/ROMEMU_SWP», potreban za promjenu načina rada memorija…

```
void romemu_swap(void) {
  unsigned int data i;

  P5 &= 0xBF;           // Postavljanje /ROMEMU_SWP linije u 0
  for(i=0;i<100;i++) _nop_();
  _nop_();        // Pričekaj neko vrijeme
  P5 |= 0x40;           // Postavljanje /ROMEMU_SWP linije u 1
```

Postavljanje korisničke zastavice u 1. Ovu zastavicu program koristi kao dojavu pokretačkom ili izvršnom programu da se izvede od početka, iako je reset prouzročio nadzorni sklop. Razlog zbog kojeg se koristi korisnička zastavica je taj, da se zastavica ne briše prilikom reseta procesora. Ovo je jedini način kako programu reći da se izvrši od početka, jer ne postoje drugi mehanizmi kojima bi program detektirao razlog prethodnog reseta prouzročenog nadzornim sklopom.

```
  PF1=1;                // Dojava programu da se izvrši od početka
}
```

4.4. Funkcija za inicijalizaciju i reset nadzornog sklopa

Funkcija za inicijalizaciju nadzornog sklopa je definirana u literaturi [16] koju daje proizvođač mikro procesora. Prvo se postavi specijalni registar CKCON prema definiciji, te se njime odabire vrijeme trajanja brojača nadzornog sklopa prije reseta. Vrijeme je postavljeno na oko 5 sekundi.

Nadalje, TA registar se postavi u 10101010_2 i odmah zatim u 01010101_2 te se resetira brojač nadzornog računala. Ova promjena bitova TA registra je potrebna kod izmjene nekih specijalnih registara procesora koji imaju veću važnost i zaštitu od slučajnog mijenjanja prilikom pogrešnog rada programa. Znači specijalni registri koji ulaze u važnije, promjena je moguća jedino na ovaj način.

```
void wd_i(void) {
   CKCON |= 0xC0;        // Postavljanje dijeljenja WD-a na 2^26

   TA = 0xAA;            // Toglle TA bits. Ovo zahtijevaju neki registri
   TA = 0x55;            // kao zaštitu od slučajnog mijenjanja
```

Odmah nakon izmjene bitova TA registara moguće je promijeniti sadržaj određenog specijalnog registra koji ima ovu zaštitu. U ovom slučaju radi se o zastavici kojom se brojač nadzornog sklopa postavlja u 0. Ovime se program štiti od slučajnog reseta odmah poslije uključenja nadzornog sklopa.

```
   RWT = 1;              // Reset WD timer-a

   TA = 0xAA;
   TA = 0x55;
```

Setiranjem ove zastavice, brojač nadzornog sklopa počinje sa odbrojavanjem.

```
   EWT = 1;              // Omogućavanje Watch Dog-a
}
```

Funkcija za resetiranje brojača nadzornog sklopa je veoma jednostavna. Radi se o promjeni bitova TA registra i odmah zatim setiranjem zastavice koja je zadužena za reset brojača.

```
void wd_r(void) {
   TA = 0xAA;
   TA = 0x55;
   RWT = 1;              // Reset WD timer-a
}
```

4.5. Funkcija za provjeru ispravnosti memorije

Funkcija za provjeru ispravnosti memorije nema neku veliku važnost, tako da ju neću posebno opisivati. Ovdje se radi o upisivanju različitih podataka u cijelu memoriju i onda čitanje te memorije. Ako upisani i pročitani podatak nisu jednaki, ta se lokaciju ispisuje na zaslon korisnika i prijavljuje se greška u memoriji. Ako je sve u redu, ispisuje se obavijest korisniku.

Memorija se puni sa raznim podatcima i u oba smjera. Prvo se upisuje 10101010_2, pa zatim 01010101_2, onda brojevi od 0 do 255 i tako stalno do kraja memorije, te se na kraju u memoriju upisuju 00. Na ovaj način se memorija očisti od svih podataka.

Kombinacije bitova koje se upisuju su uobičajene, jer te kombinacije najlakše otkrivaju uobičajene pogreške koje mogu nastati prilikom izrade samog sklopa odnosno nadzornog računala.

U funkciju je na početku još dodan kod koji isključuje nadzorni sklop, kako ne bi došlo do reseta procesora prilikom testa memorije. Isto tako, nadzorni sklop se na kraju ponovno uključuje.

```
void mem_test(void) {
  volatile unsigned char xdata *pokazivac;
  unsigned char data varijabla;
  unsigned int data i;
  bit uredu;

  TA = 0xAA;
  TA = 0x55;
  EWT = 0;                // Onemogućavanje Watch Dog-a


  printf ("\nTesting memory...\n");
  uredu=1;
  printf ("\nFilling memory with AAh up... ");
  for (i=0;65535>i;i++) {
     pokazivac=i;
     *pokazivac=0xAA;
     varijabla=*pokazivac;
           if (varijabla != 0xAA) printf (" %u",i);
           if (varijabla != 0xAA) uredu=0;
  }
  printf ("\nFilling memory with AAh down... ");
  for (i=65535;i>0;i--) {
     pokazivac=i;
     *pokazivac=0xAA;
     varijabla=*pokazivac;
     if (varijabla != 0xAA) printf (" %u",i);
     if (varijabla != 0xAA) uredu=0;
  }
  printf ("\nFilling memory with 55h up... ");
  for (i=0;65535>i;i++) {
     pokazivac=i;
     *pokazivac=0x55;
     varijabla=*pokazivac;
     if (varijabla != 0x55) printf (" %u",i);
     if (varijabla != 0x55) uredu=0;
  }
  printf ("\nFilling memory with 55h down... ");
  for (i=65535;i>0;i--) {
     pokazivac=i;
```

```
      *pokazivac=0x55;
      varijabla=*pokazivac;
      if (varijabla != 0x55) printf (" %u",i);
      if (varijabla != 0x55) uredu=0;
   }
   printf ("\nFilling memory with pattern... ");
   for (i=0;65535>i;i++) {
      pokazivac=i;
      *pokazivac=i%255;
   }
   for (i=0;65535>i;i++) {
      pokazivac=i;
      varijabla=*pokazivac;
      if (varijabla != i%255) printf (" %u",i);
      if (varijabla != i%255) uredu=0;
   }
   printf ("\nFilling memory with 00h... ");
   for (i=0;65535>i;i++) {
      pokazivac=i;
      *pokazivac=0x00;
   }
   for (i=0;65535>i;i++) {
      pokazivac=i;
      varijabla=*pokazivac;
      if (varijabla != 0x00) printf (" %u",i);
      if (varijabla != 0x00) uredu=0;
   }
   if (uredu == 1) printf("\n\nMemory is OK!\n");
   else printf ("\n\nERROR in memory!\n");

   TA = 0xAA;
   TA = 0x55;
   EWT = 1;                // Omogućavanje Watch Dog-a
}
```

Ovime je završen opis programske podrške pokretačkog programa.

5. PROGRAMSKA PODRŠKA UNUTAR PROJEKTA – IZVRŠNI PROGRAM

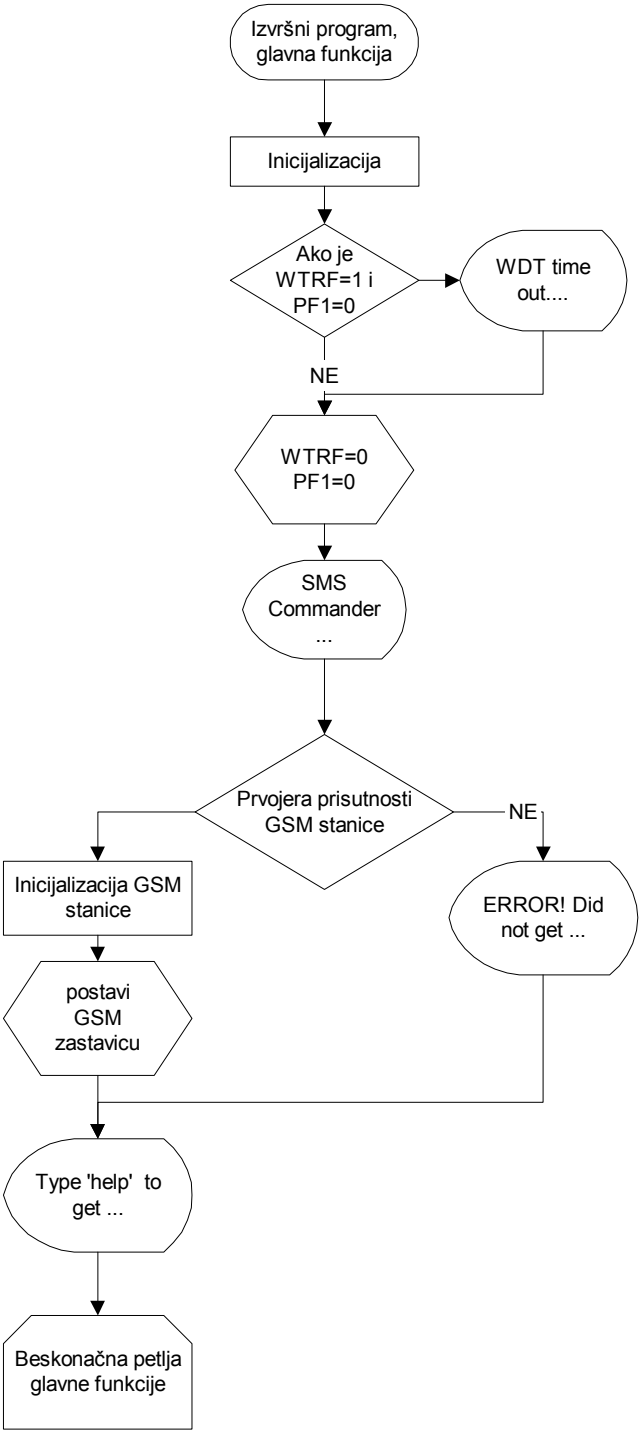
Upravljanje procesom korištenjem usluge kratkih poruka pored nadzornog računala, zahtijeva i izvršni program. Izvršni program je zadužen za interakciju sa korisnikom tj. sa osobnim računalom, zatim za komunikaciju za mobilnim GSM uređajem u svrhu slanja i primanja poruka, te primanje i uspostavljanje poziva prema operateru. Izvršni program također provjerava stanje paralelnim digitalnih ulaza te potpuno automatizirano šalje obavijest operateru putem usluge kratkih poruka. Isto tako provjerava primljene kratke poruke te dekodira poruku i izvršava komandu operatera potpuno automatizirano. Ovo su osnovne akcije koje podržava nadzorno računalo odnosno izvršni program.

Izvršni program koristi vrlo mali dio resursa nadzornog računala i služi samo za demonstraciju onoga što se zaista može napraviti. Isto tako svrha projekta nije razvoj programa u ekonomske svrhe, nego znanstveno istraživačke. Tako da stvarne mogućnosti nadzornog računala definira naručitelj projekta, odnosno mašta i znanje programera.

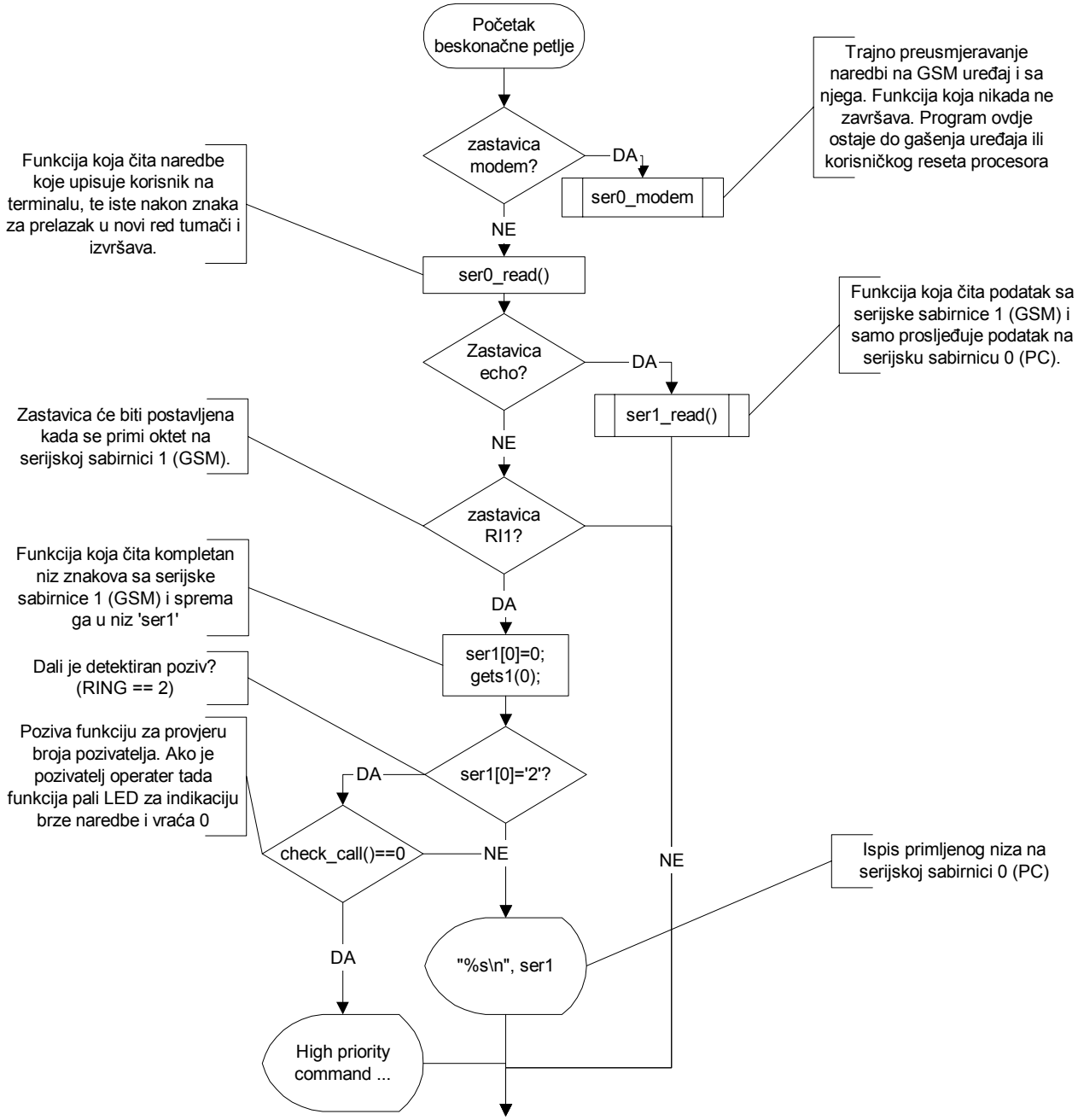
Interakcija prema korisniku na osobno računalu je veoma jednostavna i veoma lako i brzi se svladavaju sve komande i mogućnosti programa. Isto tako postoji i naredba za pomoć, no detaljno će sve biti opisane u poglavlju «Rukovanje uređajem».

Na slikama 5.1. do 5.4. su prikazani dijagrami tokova glavne funkcije izvršnog programa.

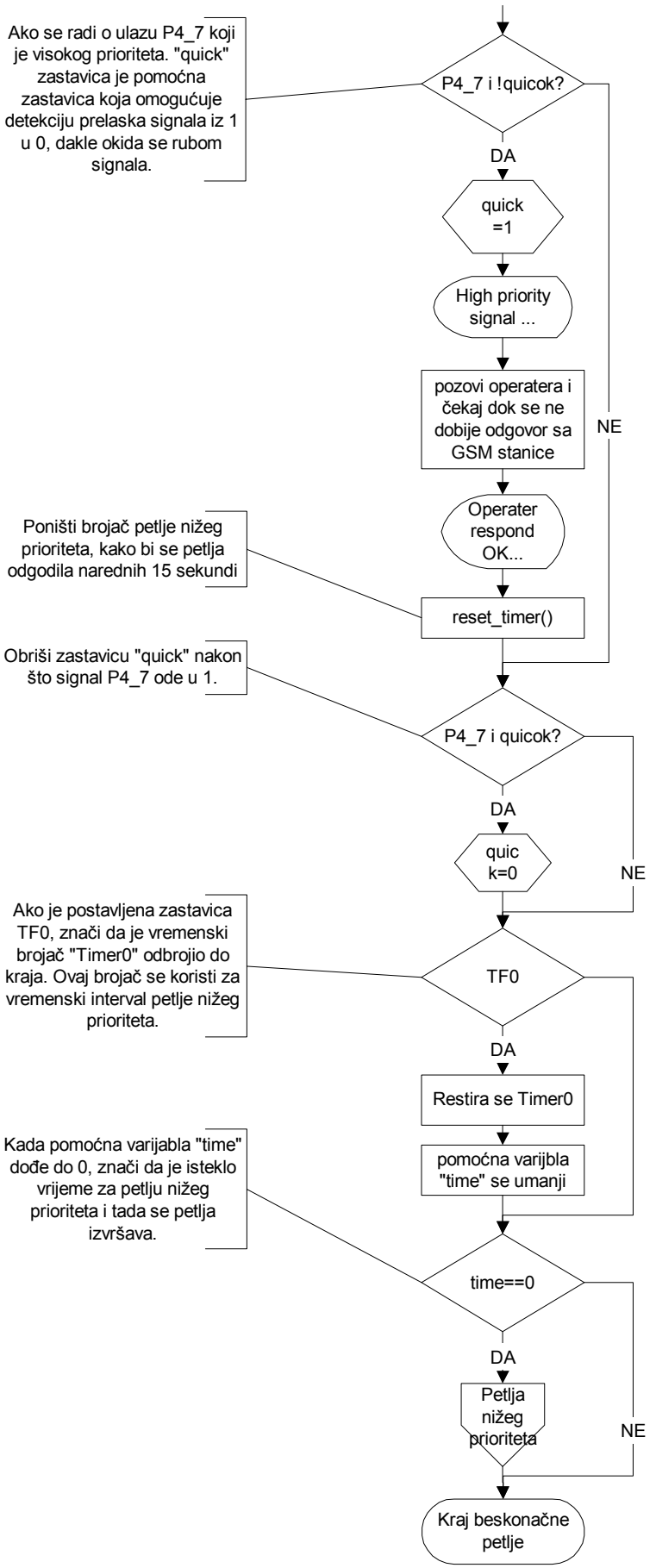
U narednim potpoglavljima biti će opisane samo važnije funkcije koje se koriste unutar izvršnog programa, dok se ispis cijelog programa nalazi u prilogu 4. Program je pisan u ANSI C jeziku, a koristi se Keil MicroVision2 programski paket sa C prevodiocem. Demo inačica programskog paketa nalazi se na CD-u u prilogu.



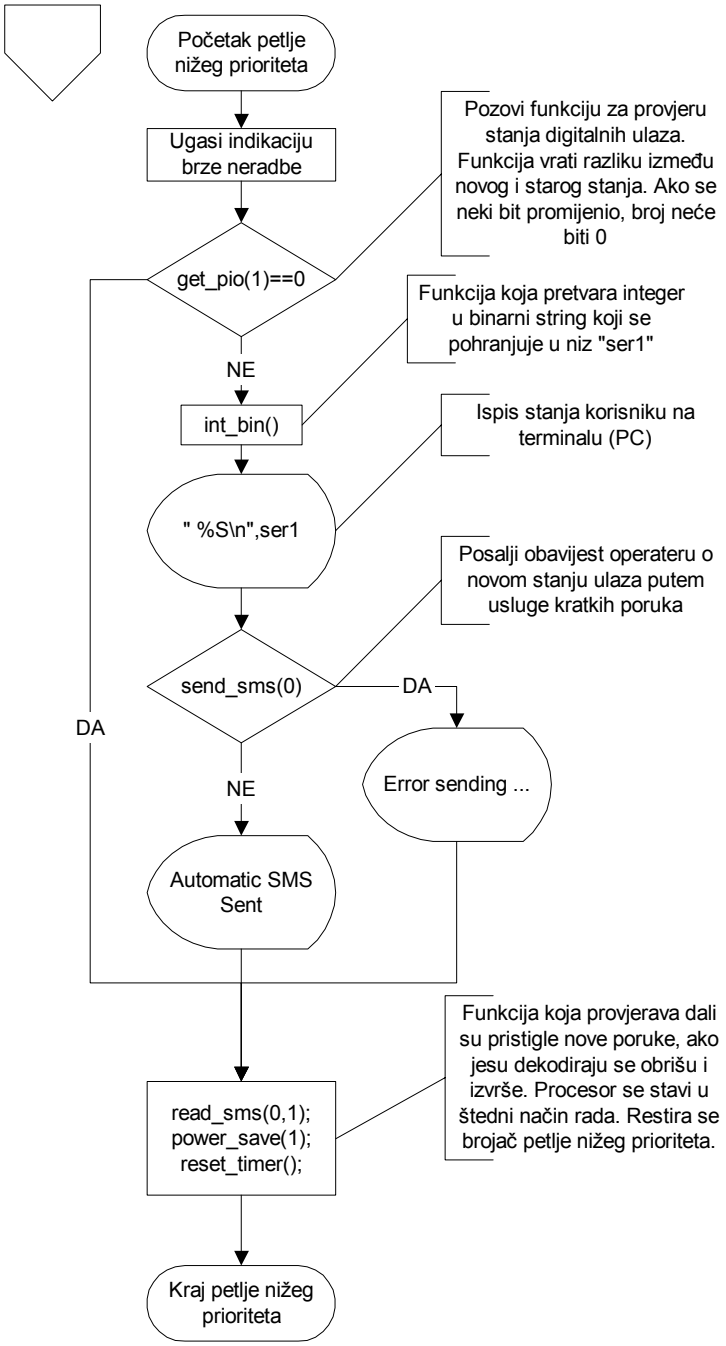
Slika 5.1. Dijagram toka glavne funkcije izvršnog programa.



Slika 5.2. Dijagram toka beskonačne petlje unutar glavne funkcije, 1. dio.



Slika 5.3. Dijagram toka beskonačne petlje unutar glavne funkcije, 2. dio.



Slika 5.4. Dijagram toka petlje nižeg prioriteta.

5.1. Glavna funkcija izvršnog programa

Glavna funkcija se pokreće odmah prilikom izvršavanja programa. Ona poziva razne funkcije za inicijalizaciju serijskih sabirnica, memorije, nadzornog sklopa i ostalih dijelova mikro procesora. Također u njoj se definira i početni telefonski broj operatera koji se sprema u niz MSNO. Također se uzmu trenutna stanja paralelnih digitalnih ulaza, te se postavi vremenski brojač «Timer0» za izvršavanje cikličke petlje nižeg prioriteta. Između ostalog postave se početne vrijednosti nekih globalnih varijabli koje se kasnije koriste u programu.

```
void main (void) {
  int xdata i;
  bit quick=0;

  strcpy(MSNO,"+38598754369");     // Broj operatera

  ser0_broj=0;                // Kažemo da nema znakova na ulazu serije0
  ser1_broj=0;                // Kažemo da nema znakova na ulazu serije1
  com_broj=0;                 // Kažemo da broj znakova komande nula
  com0_set=0;                 // Kažemo da komanda nije postavljena
  echo=0;                     // Direktno preusmjeravanje sa serije1 na seriju0
                              // bez dekodiranja odgovora
  modem=0;                    // Isključeno stalno preusmjeravanje

  ser_init();                 // Inicijalizacija serije
  wd_i();                     // Uključivanje Watch Dog-a
  get_pio(1);                 // Uzimanje trenutnih PIO vrijednosti

  CKCON &= 0xF7;              // Postavljanje dijeljenja clocka sa 12 za timer0
  TL0=0x01;                   // Postavljanje reload vrijednosti
  TH0=0x01;
  TMOD &= 0xF0;               // Timer 0 u MOD0
  TMOD |= 0x01;               // Timer 0 u MOD1: 16 bit
  TF0=0;                      // Skini zastavicu overfloat
```

Sada program provjeri dali je prošli reset nastao uslijed neočekivanog reseta procesora prouzročenog istekom vremena nadzornog sklopa. Kako je većina reseta prouzročena nadzornim sklopom, program mora raspoznavati neočekivani od namjernog reseta. Za to se brine korisnička zastavica PF1, koju program postavi u 1, ako želi dojaviti programu da se radi o namjernom resetu prouzročenom istekom vremena nadzornog sklopa. Takav namjerni reset se događa prilikom preklapanja memorija, tj. prelaska iz pokretačkog programa u izvršni, ili obrnuto. Ova provjera nema nikakvu važnost za samo izvršavanje programa, jer ako bi se i dogodio neočekivani reset iz nekog neobjašnjenog razloga, program će se izvesti od početka kao da se ništa nije dogodilo. Ova naredba je više dojava korisniku da zna da je došlo do neočekivanog reseta.

```
  if (WTRF==1 && PF1==0) printf("\n WDT time out...\n");
```

Ovdje se skidaju zastavice nadzornog sklopa i korisnička zastavica o kojoj je malo prije bilo govora, te se resetira brojač nadzornog sklopa. Poslije toga se obavijesti korisnik o trenutnom programu koji se izvršava, pokuša se prozvati GSM uređaj spojen na nadzorno računalo. Na ovaj način se provjeri dali je isti spojen pravilno. Nadzorno računalo za ispravan rad zahtijeva samo stalni spoj sa GSM uređajem. Svi drugi spojevi nisu neophodni.

```
  WTRF = 0;              // Resetiraj bit WatchDog Timera
  wd_r();                // Resetiraj WatchDog Timer
  PF1=0;                 // Dojava programu da se ne izvrši od početka
```

```
printf ("\n\n*******************************");
printf("*********************************\n");
printf("******* SMS COMMANDER **** v%s",version);
printf(" **** by Goran Jurkovich **********\n");
printf("*********************************");
printf("*********************************\n\n");

gsm=0;                              // GSM uređaj inicijalno ne postoji
sprintf(ser1,"at\n");               // Prozivanje GSM uređaja
gets1(0);                           // Slanje komande na seriju1
sprintf(ser1,"atz\n");              // Resetiranje GSM uređaja
gets1(0);                           // Slanje komande na seriju1
sprintf(ser1,"ate0v0\n");           // Isključivanje 'echo' f-je
gets1(0);                           // Slanje komande na seriju1
sprintf(ser1,"at\n");
gets1(0);
if (ser1[0]=='0') gsm=1;
    else printf(" ERROR! Did not get respond from GSM device!\n");
```

Znači program je poslao GSM uređaju komandu «AT» i on odgovara sa «OK» odnosno «0» ako je u modu brojevnih odgovora. Program provjeri odgovor i ako je odgovor «0» postavi zastavicu «gsm». Slijedeća linija provjeri dali je zastavica «gsm» postavljena i prema njima pali LED na PIO2_0 u slučaju pogreške, odnosno PIO2_3 ako je GSM uređaj uredno spojen, te ju gasi nakon 15 sekundi. Ovo paljenje LED indikacije je čisto u razvojne svrhe tako da korisnik zna dali je sve u redu nakon staranja programa ako nadzorno računalo nije spojeno na terminal odnosno osobno računalo.

```
if (gsm) PIO2_3=0; else PIO2_0=0;// Indikacija korisniku
                                 // ako terminal nije spojen
```

Ako je GSM uređaj spojen, tada se uključi postavka za prikaz broja koji zove, te se isključe alarmne i vibracijske dojave poziva zbog tišeg rada GSM uređaja pošto sve akcije izvršava nadzorno računalo. Poslije toga korisniku ispisuje neke osnovne informacije o spojenom GSM-u, kao proizvođač uređaja, model, programska podrška, itd.… Sve detaljno o naredbama koje se šalju GSM uređaju može se naći u literaturi [5] te u prilogu na CD-u i na web stranicama www.siemens.com.

```
if (gsm) {
    sprintf(ser1,"at+clip=1\n");        // Uključivanje CLIP f-je
    gets1(0);                           // Slanje komande na seriju1
    sprintf(ser1,"at+calm=2\n");        // Isključivanje Zvučnih alarma
    gets1(0);                           // Slanje komande na seriju1
    sprintf(ser1,"at+cvib=0\n");        // Isključivanje vibracijskog alarma
    gets1(0);                           // Slanje komande na seriju1

    sprintf(ser1,"at+cgmi\n");gets1(1); // Naredba GSM stanici
    printf(" GSM Station Vendor is %s\n",ser1);

    sprintf(ser1,"at+cgmm\n");gets1(1); // Naredba za ispis modela
    printf(" GSM Station Model is %s\n",ser1);

    sprintf(ser1,"ati9\n");gets1(1);    // Naredba za ispis procesora
    printf(" GSM Station Software is %s\n",ser1);
    ser1[0]=0;gets1(0);                 // Ako ima još koja dolazna poruka
    }
printf("\n Type 'help' to get some assistance...\n\n");
```

Sada se resetira brojač petlje nižeg prioriteta i isti se starta.

```
reset_timer0();             // Resetiraj timer0
TR0=1;                      // Startaj Timer0
```

Ovdje se izvršava beskonačna petlja koja poziva funkcije za automatizaciju slanja i primanja poruka, zatim interakciju sa korisnikom i GSM uređajem, stanja paralelnih digitalnih ulaza, itd. Postoje dvije petlje. Odnosno petlja višeg prioriteta i unutar nje, petlja nižeg prioriteta koja se izvršava svakih 15 sekundi ali nakon neaktivnosti korisnika.

```
while (1) {
```

Ovdje program provjeri dali je postavljena zastavica «modem» odnosno dali je korisnik zatražio stalno preusmjeravanje naredbi na GSM uređaj, odnosno odgovora natrag na terminal. Ova funkcija je korisna prilikom razvoja programske podrške, kako bi korisnik mogao ispitati naredbe za GSM uređaj. Ako trajno preusmjeravanje nije uključeno, tada se poziva funkcija za uzimanje komande od terminala, tj. za interakciju sa korisnikom.

```
if (modem)                  // Ako je selektiran modem
      ser0_modem();         // Napravi trajnu redirekciju
else
      ser0_read();          // Pročitaj podatak sa serije0
```

Ovdje program provjeri dali je uključena zastavica «echo», odnosno dali se radi direktno preusmjeravanje odgovora sa GSM uređaja na terminal ili se radi sa dekodiranjem odgovora. Znači ako je direktno, tada se poziva funkcija «ser1_read» koja samo preusmjeri karakter sa serije 1 na seriju 0, a ako se dekodira komanda, tada se provjerava dali postoji podatak na serijskoj sabirnici 1 provjerom zastavice «RI1».

```
if (echo) {                 // Ako je direktna redirekcija
      ser1_read();          // Pročitaj podatak sa serije1
} else if (RI1) {           // Ako nije 'ECHO' dekodiraj odgovor
```

Poziva se funkcija «gets1» koja vraća odgovor sa GSM uređaja u nizu «ser1». Onda se provjeri dali je prvi znak u odgovoru «2» što znači poziv detektiran, i ako je, tada se poziva funkcija «check_call» koja provjerava tko zove nadzorno računalo, odnosno, dali ga zove operater. Tada se obavijesti o tome korisnik i uradi određena akcija. Ako odgovor nije «2», tada se korisniku ispisuje odgovor na terminal. Ovo je dio kojim se uključuje proces visokog prioriteta koji zahtjeva brzi odziv na komandu udaljenog operatera. Više o ovome je opisano u poglavlju «Rukovanje uređajem».

```
      ser1[0]=0;
      gets1(0);
      if (ser1[0]=='2')       // Ako se detektira poziv
      if (check_call()==0)    // Provjeri tko zove
      printf(" High priority command detected!\n");
      else printf("%s\n",ser1);     // Ispisi odgovor na terminalu
}
```

Digitalni paralelni ulaz PIO1_7 odabran je kao ulazni signal visokog prioriteta. Stoga ova if petlja provjerava dali je taj ulaz u 0, i da je zastavica «quick» u 0. Ova zastavica se koristi iz jednostavnog razloga, kako bi se detektirala promjena ulaza iz 1 u 0, te kako se ne bi ulazilo u ovu petlju svaki

puta, nego samo prvi puta. Znači, ako su zadovoljeni uvjeti, tada se zastavica «quick» postavi u 1, obavijesti se korisnika o detekciji signala visokog prioriteta, i tada se šalje naredba GSM uređaje da uspostavi poziv prema operateru.

```
if (!P4_7 && !quick) {        // Ako se radi o informaciji velikog
      quick=1;                // prioriteta
      printf("\n High priority signal detected!\n");
      sprintf(ser1,"\natdt%s;\n",MSNO);
      printf1();              // Posalji komandu GSM-u
```

Ovdje se čeka odgovor GSM uređaja, odnosno da udaljeni operater prekine poziv, i o tome se obavijesti korisnika na terminalu.

```
      while (!RI1) wd_r();    // Dok nema odgovora čekaj
      printf(" Operator respond OK!\n");
```

Slijedeći niz komandi se šalje na GSM uređaj, kako bi se ponovno dobila kontrola nad njim. Funkcija «gets1» uzima tekst iz niza «ser1» i pošalje ga GSM uređaju kao komandu. Zatim čeka odgovor i spremi ga natrag u niz «ser1».

```
      sprintf(ser1,"at\n");
      gets1(0);
      sprintf(ser1,"ate0v0\n");
      gets1(0);
```

Resetira se brojač petlje nižeg prioriteta, kako se narednih 15 sekundi ne bi pozivao GSM uređaj.

```
      reset_timer0();
}
```

Slijedećim kodom se detektira prelazak signala visokog prioriteta iz logičke 0 u 1, čime se obriše zastavica «quick» i pripremi program za slijedeću detekciju prelaska signala iz 1 u 0.

```
if (P4_7 && quick) {
      quick=0;
}
```

Slijedeća petlja detektira istek vremena brojača petlje nižeg prioriteta. Kako jedan ciklus brojača traje oko 1/14 sekunde pri normalnom načinu rada, a potrebno je osigurati da se petlja nižeg prioriteta izvrši svakih 15 sekundi. To se postiže ovom petljom koja osigura 15*14 isteka brojača i tek tada se poziva petlja nižeg prioriteta. Slijedećih nekoliko linija resetira brojač, i umanji varijablu «time» za jedan koja ustvari predstavlja «pravi» brojač petlje nižeg prioriteta.

```
if (TF0) {              // Timer za petlju nižeg prioriteta
      TR0=0;            // Zaustavi Timer0
      TL0=0x01;         // Reload vrijednost za Timer 0
      TH0=0x01;
      TF0=0;            // Skini zastavicu overfloat
      TR0=1;            // Startaj Timer0
      time--;
```

No problem nastaje kada se procesor stavi u štedni način rada kada se strojni ciklus uspori oko 256 puta. Tada jedan ciklus brojača traje oko 15 sekundi. Slijedeća linija detektira kada je procesor u

štednom načinu rada, i tada zaista jedan ciklus brojača «timer0» traje koliko i jedan ciklus brojača petlje nižeg prioriteta. Ovom linijom je ujedno i završena petlja brojača nižeg prioriteta.

```
            if (PMR&0x40) time=0;   // Ako je u power save načinu rada
      }
```

Ulazak u petlju nižeg prioriteta se događa samo kada je varijabla «time» jednaka 0, odnosno kada je brojač odbrojio 15 sekundi. Isključi se indikacija brzog procesa 15 sekundi nakon njegove detekcije.

```
      if (!time) {                  // Petlja nižeg prioriteta
            PIO2_3=1;               // Isključi brzi proces
```

Slijedeća if petlja provjerava trenutno stanje signala digitalnih paralelnih ulaza pozivom funkcije «get_pio» koja vraća promjenu stanja. Stoga ako promjena postoji, funkcijom «int_bin» stanje ulaza prebaci se u tekstualni niz i spremi u niz «ser1». U niz «MSN» se spremi broj operatera i pozove funkcija za slanje kratke poruke «send_sms».

```
            if(get_pio(1)!=0) {                 // Ako se dogodi promjena ulaza
                  int_bin();                    // Prebaci PIO u string
                  printf(" %s\n",ser1);         // Ispisi trenutno stanje
                  sprintf(ser1,"%s\n",ser1);    // Ispisi ga u SMS buffer
                  strcpy(MSN,MSNO);             // Definiraj broj kojemu se šalje
                  i=send_sms(0);                // Posalji obavijest korisniku
                  if (i!=0) printf(" Error sending automatic SMS!\n");
                  else printf(" Automatic SMS Sent!\n");
            }
```

Slijedeća funkcija «read_sms» provjeri stanje kratkih poruka u GSM uređaju, tj. provjeri dali je pristigla nova poruka. Ako postoji ista se pročita, dekodira i izvrši naredba operatera.

```
            read_sms(0,1);
```

Kada nema aktivnosti korisnika 15 sekundi, nije loše staviti procesor u štedni način rada. Procesor i dalje radi, ali 256 puta sporije, i sam se vraća u normalan način rada kada se pojavi podatak na jednoj ili drugoj serijskoj sabirnici. I slijedeća linija poziva funkciju «reset_timer0» koja resetira brojač petlje nižeg prioriteta. Ovom linijom ujedno i završava petlja nižeg prioriteta.

```
            power_save(1);              // Stavi procesor u spori način rada
            reset_timer0();             // Resetiraj timer0
      }
```

Svaki puta se poziva funkcija «wd_r» koja resetira brojač nadzornog sklopa, čime se program štiti od neočekivanog reseta procesora.

```
      wd_r();                           // Resetiraj WatchDog Timer
   }
}
```

5.2. Funkcija za inicijalizaciju serijskih sabirnica i memorijskih područja procesora

Na mikro procesoru DS80C390 postoje dvije serijske asinkrone sabirnice kao što je već navedeno u 3. poglavlju. Te sabirnice je potrebno pokrenuti i postaviti postavke komunikacije. Obje sabirnice su postavljene jednako na isti vremenski brojač «Timer1» i komunikacija se odvija na 19200bps. Podatci se šalju u paketima od 8 bita, bez bita pariteta sa jednim stop bitom. Ujedno se postave postavke memorije i oslobađa ulazno-izlazni port P4 na procesoru za potrebe nadzornog računala.

Način inicijalizacije je gotovo standardan i radi se prema «kuharici», a svi parametri i detalji su dani u literaturi [16], koju daje proizvođač mikro procesora.

```
void ser_init (void) {
  EA=0;                    // Onemogući prekide
  EPF1=0;                  // Onemogući Power fail prekid

  // Komunikacija preko RS232 postavljena na 19200bps 8-N-1
  SCON0 = 0x52;            // Postavljanje moda prvog serijskog porta
  SCON1 = 0x52;            // Postavljanje moda drugog serijskog porta

  PCON |= 0x80;          // Udvostručavanje brzine prvog serijskog porta
  SMOD_1 = 1;            // Udvostručavanje brzine drugog serijskog porta

  TMOD = 0x20;           // 2 za Timer1 (RS232)
  TH1 = 0xFD;            // Postavljanje reload vrijednosti za 9600
  TL1 = 0xFD;

  IP  = 0;               // prioritet
  PS1 = 1;               // Visoki prioritet serije1
  TR1 = 1;               // Startaj Timer1

  ET1 = 0;               // Onemogući prekidnu rutinu za Timer1
  EA  = 1;               // uključi globalne interapte

        TA = 0xAA;       // Promjena TA bitova za Timed Access
        TA = 0x55;       // Specijalne registre
  P5CNT |=0x20;          // Postavljanje drugog serijskog porta
                         // na port P5.2 i P5.3
        TA = 0xAA;       // Promjena TA bitova za Timed Access
        TA = 0x55;       // Specijalne registre
  P5CNT &=0xF8;          // Konfiguracija od 64KB

        TA = 0xAA;       // Promjena TA bitova za Timed Access
        TA = 0x55;       // Specijalne registre
  P4CNT &=0x80;          // Konfiguracija od 64KB

  PMR = 0x27;            // Isključivanje ALE signala
        TA = 0xAA;       // Promjena TA bitova za Timed Access
        TA = 0x55;       // Specijalne registre
  MCON = 0x10;           // Postavke memorije

}
```

5.3. Funkcija za inicijalizaciju i reset nadzornog sklopa

Funkcija za inicijalizaciju nadzornog sklopa je definirana u literaturi [16] koju daje proizvođač mikro procesora. Prvo se postavi specijalni registar CKCON prema definiciji, te se njime odabire vrijeme trajanja brojača nadzornog sklopa prije reseta. Vrijeme je postavljeno na oko 5 sekundi.

Nadalje, TA registar se postavi u 10101010_2 i odmah zatim u 01010101_2 te se resetira brojač nadzornog računala. Ova promjena bitova TA registra je potrebna kod izmjene nekih specijalnih registara procesora koji imaju veću važnost i zaštitu od slučajnog mijenjanja prilikom pogrešnog rada programa. Znači specijalni registri koji ulaze u važnije, promjena je moguća jedino na ovaj način.

```
void wd_i(void) {
   CKCON |= 0xC0;        // Postavljanje dijeljenja WD-a na 2^26

   TA = 0xAA;            // Toglle TA bits. Ovo zahtijevaju neki registri
   TA = 0x55;            // kao zaštitu od slučajnog mijenjanja
```

Odmah nakon izmjene bitova TA registara moguće je promijeniti sadržaj određenog specijalnog registra koji ima ovu zaštitu. U ovom slučaju radi se o zastavici kojom se brojač nadzornog sklopa postavlja u 0. Ovime se program štiti od slučajnog reseta odmah poslije uključenja nadzornog sklopa.

```
   RWT = 1;              // Reset WD timer-a

   TA = 0xAA;
   TA = 0x55;
```

Setiranjem ove zastavice, brojač nadzornog sklopa počinje sa odbrojavanjem.

```
   EWT = 1;              // Omogućavanje Watch Dog-a
}
```

Funkcija za resetiranje brojača nadzornog sklopa je veoma jednostavna. Radi se o promjeni bitova TA registra i odmah zatim setiranjem zastavice koja je zadužena za reset brojača.

```
void wd_r(void) {
   TA = 0xAA;
   TA = 0x55;
   RWT = 1;              // Reset WD timer-a
}
```

5.4. Funkcija za promjenu načina rada memorija

Ova funkcija stvara negativan impuls na signalu «/ROMEMU_SWP» potreban da se prebaci stanje prvog JK bistabila. Način rada je opisan u 3. poglavlju.

Svi ulazno-izlazni signali na portovima mikro procesora se postavljaju u logičku 1 prilikom reseta. Kada se pozove funkcija za promjenu načina rada memorija, tada se postavi signal «/ROMEMU_SWP» u logičku 0. Petlja služi za stvaranje vremenskog kašnjenja, i tada se ponovno signal postavi u logičku 1. Na ovaj način je dobiven negativan impuls na signalnoj liniji «/ROMEMU_SWP», potreban za promjenu načina rada memorija…

```
void romemu_swap(void) {
  unsigned int data i;

  P5 &= 0xBF;           // Postavljanje /ROMEMU_SWP linije u 0
  for(i=0;i<100;i++) _nop_();
  _nop_();        // Pričekaj neko vrijeme
  P5 |= 0x40;           // Postavljanje /ROMEMU_SWP linije u 1
```

Postavljanje korisničke zastavice u 1. Ovu zastavicu program koristi kao dojavu pokretačkom ili izvršnom programu da se izvede od početka, iako je reset prouzročio nadzorni sklop. Razlog zbog kojeg se koristi korisnička zastavica je taj, da se zastavica ne briše prilikom reseta procesora. Ovo je jedini način kako programu reći da se izvrši od početka, jer ne postoje drugi mehanizmi kojima bi program detektirao razlog prethodnog reseta prouzročenog nadzornim sklopom.

```
  PF1=1;                // Dojava programu da se izvrši od početka
}
```

5.5. Funkcija za promjenu načina rada procesora i prelazak u štedni način

Slijedeća funkcija služi za prelazak u štedni način rada mikro procesora. Odabir moda rada se izvršava preko postavki specijalnih registara koji su definirani u literaturi [16] koju izdaje proizvođač.

U štednom načinu rada procesor radi oko 256 puta sporije, ali i dalje izvršava program. Procesor u tom modu rada troši isto kao kada je potpuno zaustavljen. Procesor će se potpuno automatizirano vratiti u normalni način rada čim se pojavi podatak na bilo kojoj serijskoj sabirnici.

```
void power_save(bit mod_rada) {

        if (mod_rada == 1) {
                PMR |= 0xA0;    // Prelazak u clock / 4
                PMR |= 0xE0;    // Prelazak u clock / 1024
        } else {
                PMR |= 0xA0;    // Prelazak u clock / 4
        }
}
```

5.6. Funkcija za ispis naredbe na GSM uređaj

Funkcija ispisuje tekstualni niz «ser1» na serijsku sabirnicu 1, na kojoj je spojen GSM uređaj. Funkcija je standardna funkcija «putchar» koja je prepravljena da radi sa serijskom sabirnicom 1 umjesto standardno sa serijskom sabirnicom 0. Isto tako funkciji je dodana while petlja koja uzima jedan po jedan karakter iz niza «ser1» i ispisuje ga.

```
void printf1 (void)  {
  unsigned char data i,c;
  i=0;
  while (ser1[i] != '\0') {        // Petlja koja se izvršava do '\0'
      c=ser1[i];
```

Sve naredne linije koda predstavljaju standardnu funkciju «putchar» koju nije potrebno posebno objašnjavati.

```
      if (c == '\n')  {
            if (RI1)  {
                  if (SBUF1 == XOFF)  {
                        do  {
                              RI1 = 0;
                              while (!RI1);
                        }
                        while (SBUF1 != XON);
                        RI1 = 0;
                  }
            }
            while (!TI1);
            TI1 = 0;
            SBUF1 = 0x0d;                        /* output CR  */
      }
      if (RI1)  {
            if (SBUF1 == XOFF)  {
                  do  {
                        RI1 = 0;
                        while (!RI1);
                  }
                  while (SBUF1 != XON);
                  RI1 = 0;
            }
      }
      while (!TI1);
      TI1 = 0;
      SBUF1 = c;
      i++;
  }
}
```

5.7. Funkcija za čitanje stanja paralelnih digitalnih ulaza

Slijedeća funkcija uzima bit po bit stanje sa paralelnih digitalnih ulaza i stavlja ih u 16 bitni broj, tj. varijablu «pio». Na ovaj način je lakše manipulirati stanjima i svi se ulazi nalaze na jednom mjestu. Tako se isto može čuvati trenutno stanje ulaza u varijabli «pio», prethodno stanje ulaza u varijabli «pio_old». Ako se napravi ekskluzivna ili (eng. XOR) funkcija nad njima dobiti će se koji su bitovi promijenjeni. Funkcija će vratiti varijablu u kojoj je zapisana promjena starog na novo stanje bitova.

Funkcija u slijedećih desetak linija uzima podatke sa ulaza i bit po bit ih slaže u varijablu.

```
unsigned int get_pio (bit pio_tog) {
  unsigned int pio_t,j,k,l;
  unsigned char i;
  if (pio_tog) pio_old=pio;
  pio=0;
  pio_t=0;
  if (P4_7) pio |=0x0080;
  if (P4_6) pio |=0x0040;
  if (P4_5) pio |=0x0020;
  if (P4_4) pio |=0x0010;
  if (P4_3) pio |=0x0008;
  if (P4_2) pio |=0x0004;
  if (P4_0) pio |=0x0002;
  if (P4_1) pio |=0x0001;
  if (PIO2_3) pio |=0x0800;
  if (PIO2_2) pio |=0x0400;
  if (PIO2_1) pio |=0x0200;
  if (PIO2_0) pio |=0x0100;
```

Slijedećom for petljom se provjerava bit po bit i uspoređuje staro stanje sa novim. Ova petlja je ustvari programska implementacija «ekskluzivne ili» funkcije (eng. XOR), koja nije implementirana u ANSI C programskom jeziku.

```
  for (i=0;i<12;i++) { //Petlja za dobivanje: pio_t=pio XOR pio_old;
      j=pio;
      k=pio_old;
      j=_iror_(j,i);
      k=_iror_(k,i);
      j&=0x01;
      k&=0x01;
      l=j+k;
      l&=0x01;
      l=_irol_(l,i);
      pio_t|=l;
  }
```

Funkcija vraća promjenu stanja.

```
  return (pio_t);
}
```

5.8. Funkcija za komunikaciju sa GSM uređajem

Funkcija «gets1» je zamišljena kao standardna funkcija «gets» koja uzima tekst sa serijske sabirnice 0 sve dok ne dođe znak za prelazak u novi red. Funkcija «gets1» ima istu ulogu s tim da uzima podatke sa serijske sabirnice 1 i proširena je time da se može uzeti i više redova teksta i da se prije uzimanja odgovora prvo pošalje naredba. Dakle, ova je funkcija prilagođena potrebama komunikacije sa GSM uređajem. U tekstualni niz «ser1» se postavi naredba koja se želi poslati GSM uređaju. Funkcija «gets1» pošalje tu naredbu i čeka odgovor od GSM uređaja, te ga vraća natrag u nizu «ser1». Funkcija će vratiti odgovor do prvog prelaska u novi red ili čitav tekst što se zadaje kao parametar funkcije «OE» (eng. One Enter). Ako je 1 tada će uzeti tekst do prvog prelaska u novi red, inače uzima cijeli tekst.
Prvi dio funkcije poziva funkciju «printf1» koja šalje naredbu GSM uređaju. Ostale linije su inicijalizacija brojača karaktera, petlje čekanja i ostalih stvari.

```
void gets1 (bit OE) {
  char c;
  int data i;
  if (strlen(ser1)>1)  printf1();        // Posalji naredbu GSM stanici
  ser1_broj=0;               // Postavi broj upisanih znakova na 0
  ser1[0]=0;                 // Završi niz sa '\0'
  i=-2;
```

Sada slijedi petlja koja umanjuje varijablu «i» sve dok ona ne postane 0, ili dok se ne pojavi znak na ulazu serijske sabirnice 1 (zastavica RI1). Kada se pojavi karakter, on se sprema u varijablu «c» i sprema se na «ser1_broj» mjesto u nizu koje predstavlja trenutno mjesto odnosno znak teksta. Ujedno se brojač čekanja (varijabla «i») postavlja na najveću vrijednost.

```
  do {
     if (RI1){               // Ako se pojavi karakter na ulazu serije1
           c=SBUF1;                              // Uzmi karakter sa serije1
           ser1[ser1_broj++]=c;                  // spremi ga u buffer
           i=-2;                                 // Postavi odbrojavanje
```

Ovdje program provjerava da nije slučajno primljeni tekst premašio veličinu spremnika niza. Slijedeća naredba provjerava dali je primljeni karakter, znak za prelazak u novi red, ako jeste, onda se provjerava dali je to prvi karakter u tekstu koji se preskače. Ako nije, tada se provjerava dali se želi uzeti cijeli tekst. Ako se ne želi uzeti cijeli tekst onda se brojač «i» postavi u 0 i izlazi iz petlje. Naravno potrebno je obrisati zastavicu «RI1» kako bi se mogao primiti slijedeći podatak.

```
           if (ser1_broj>160) ser1_broj--;     // Ako smo prekoračili buffer
           if (c==0x0a || c==0x0d) {           // Ako se radi o 'Enter'
                 if (ser1_broj<=1) ser1_broj=0; // Zanemari linije bez teksta
                 else if (OE) i=0;             // Ako je setiran OE tada se
           }                                   // odmah izlazi van
           RI1=0;                              // Uzeo znak sa serije1
     }
  } while (i--);
```

Niz teksta se završava sa null karakterom. Poziva se funkcija za resetiranje brojača petlje nižeg prioriteta. Na ovaj način se osigurava da se ne pristupa GSM uređaju slijedećih 15 sekundi unutar petlje nižeg prioriteta. I na kraju se pozove funkcija za reset brojača nadzornog sklopa.

```
  ser1[ser1_broj]='\0';                        // Završavam string sa '\0'
  reset_timer0();                              // Resetiraj timer0
  wd_r();                                      // Resetiraj WDT
}
```

5.9. Funkcija za unos teksta koji se šalje kao kratka poruka

Kako poruka može biti velika i do 160 znakova, te u sebi može sadržavati i slova, brojeve, znakove i znak za prelazak u novi red, morao sam napraviti posebnu funkciju koja će omogućiti korisniku da napiše tekst te unos završi sa tipkom «ESC». Tako je nastala i ova funkcija. Funkcija će omogućiti korisniku da unese tekst, te će ga spremiti u niz «ser1» i pozvati funkciju za slanje kratkih poruka.

Na početku se provjeri dali je korisnik dobro zadao naredbu, ako nije izvijesti se korisnika o načinu korištenja naredbe.

```
int send_sms_input (void) {
  unsigned char i,c;
  if (strlen(ser0)<10 && ser0[8]!='+') { // Pogreška pri unosu naredbe
     printf("\nERROR! SENDSMS command. Usage:\n");
     printf(" SENDSMS +PHONE_NUMBER\n");
     printf("\n For example: SENDSMS +12345678901\n");
     return(-1);
  }
```

Sada se iz ulaznog niza «ser0» izdvaja samo broj i sprema u niz «MSN». Poslije toga nekoliko obavijesti korisniku o načinu upisa teksta. Tada se postavi pokazivač na početak niza «ser1» i isključi brojač nadzornog sklopa kako ne bi došlo do neželjenog reseta procesora prilikom unosa teksta.

```
  i=8;
  while ((c=ser0[i++]) !=0 && c!=0x0F && c!=0x0D && c!=0x0A) MSN[i-9]=c;
                       // Kopiram broj iz 'ser0'
  MSN[i-9]=0;          // Dodajem ga u MSN i Završavam string sa '\0'

  printf("OK!\n\n Sending SMS to MSN: %s",MSN);
  printf("\n Now, enter your text for SMS,");
  printf(" and finish with 'CTRL+Z' or 'ESC'\n\n");

  ser1[0]=0;
  ser1_broj=0;
       TA = 0xAA;
       TA = 0x55;
       EWT = 0;                // Onemogućavanje Watch Dog-a
```

Slijedeće petlja uzima karakter sa serijske sabirnice 0 standardnom funkcijom «getchar» i sprema ga u varijablu «c» i u niz «ser1» na trenutno mjesto «ser1_broj». Ako se pak radi o znaku za brisanje, tada se trenutna pozicija unutar niza «ser1» smanjuje za 2 broja, kako bi se obrisao znak za brisanje i slovo koje se želi obrisati. Petlja se radi ako korisnik upiše «CTRL+Z» ili «ESC»

```
  do {      // Petlja za čitanje texta sa terminala
     c=getchar();            // Uzimam podatak sa serije0
     ser1[ser1_broj++]=c;    // Upisi trenutni znak u string
     if (c==0x08) {          // Ako se radi o 'delete'
           ser1_broj-=2;     // Tada pokazivac stavi na zadnji točan znak
     }
  } while (c!=0x1a && c!=0x1b && ser1_broj<130);
  // Ako se radi o 'CTRL+Z' ili 'ESC' izađi van
```

Sada se ponovno uključuje brojač nadzornog sklopa za kontrolu ispravnosti rada procesora i odmah se pozove funkcija za resetiranje brojača nadzornog sklopa.

```
  TA = 0xAA;
  TA = 0x55;
  EWT = 1;                    // Omogućavanje Watch Dog-a
  wd_r();                     // Reset Watch Dog Timer-a
```

Niz «ser1» se završava sa null karakterom, izvijesti korisnika o slanju poruke i pošalje poruka pozivom funkcije za slanje kratkih poruka.

```
  ser1[--ser1_broj]=0;        // Završavam string sa '\0'
  printf("\n\n Sending SMS...\n");
  wait(1000);
  i=send_sms(0);
```

Funkcija «send_sms» će vratiti 0 ako je slanje uspjelo, odnosno –1 ako nije. Stoga program provjeri što je funkcija vratila i o tome izvijesti korisnika, te isto vrati nazad programu koji je pozvao ovu funkciju.

```
  if (i==0) {
     printf(" SMS sent!\n");
     return(0);
  } else {
     printf(" SMS sending failed!\n");
     return(-1);
  }
}
```

5.10. Funkcija za čitanje naredbi i interakciju prema korisniku

Unos naredbi koje zadaje korisnik negdje se moraju zapisati kako bi se naredbe kasnije mogle interpretirati. Upravo u tu svrhu služi ova naredba. Ona se poziva ciklički u glavnoj petlji glavne funkcije i ona provjeri dali je novi karakter došao na ulazu serijske sabirnice 0. Karakter se sprema u varijablu «c». Poslije toga se resetira brojač petlje nižeg prioriteta, kako bi se ista odgodila narednih 15 sekundi neaktivnosti korisnika.

```
void ser0_read (void) {
  char data c;

  if (RI) {            // Ako se pojavi karakter na ulazu serije0

     c=getchar();      // Uzimam podatak sa serije0
     reset_timer0();
```

Sada se provjeri koji je dali je upisani znak «*», te ako jeste, onda se izvršava prethodno izvršena naredba pozivom funkcije za dekodiranje komande.

```
     if (c=='*' && ser0_broj==0) {       // Ako želimo ponoviti staru
           printf("\r%s",ser0);          // već izvršenu komandu
           decode_command();
     } else {                            // Inače normalna procedura
```

Ako pak nije prvi znak «*» tada ide normalna procedura upisa novog karaktera u niz «com0» i niz «ser0». Niz «com0» sadrži samo naredbu (naredba je od ostalih parametara odvojena razmakom) koja se pretvara u mala slova. Ovo je bitno za kasnije dekodiranje naredbe. U slijedeću petlju se ulazi samo ako već nije upisana čitava naredba. To se vidi po zastavici «com0_set» koja se postavi onda kada korisnik upiše znak za razmak, ili znak za prelazak u novi red ili ako se radi o početnom nizu «AT». Tada se ujedno postavi zastavica «com0_set».

```
     if (com0_set==0) {            // Ako komanda već nije upisana do kraja
           com_broj=ser0_broj;     // Izjednači trenutni znak komande
                                   // upisanim stringom

           com0[com_broj]=tolower(c);    // Pretvori velika u mala slova

           if (c==0x20 || c==0x0a || com_broj>8) {  // Ako se radi o
                 // razmaku, ili enteru, ili premašenom spremniku za
                 // komande, tada Završi komandu
                 com0_set=1;
                 com0[com_broj]=0; // Na kraj stavi '\0'
           }

           // Ako se radi o 'at' komandi onda je ona već završena
           if (com_broj==1 && com0[0]=='a' && com0[1]=='t'){
                 com0[++com_broj]=0;     // Završavam naredbu sa '\0'
                 com0_set=1;             // Postavljam kraj komande
           }
     }
```

U nizu «ser0» nalazi se i komanda i parametri iza komande. Sada se trenutni karakter upisuje u niz «ser0». Provjeri se da unijeti tekst ne prelazi veličinu niza.

```
     ser0[ser0_broj]=c;                  // Upisi trenutni znak u string
     if (ser0_broj<60) ser0_broj++;      // Ako nismo na kraju
```

```
      // buffera za string onda pomakni pokazivac na slijedeći
```

Ako se radi o znaku za brisanje, tada se trenutna pozicija unutar niza «ser0» smanjuje za 2, tj. za znak za brisanje i znak koji zaista želimo obrisati. Isto tako provjerimo da možda korisnik nije pritisnuo višak znakova za brisanje, tada postavljamo trenutnu poziciju u nizu «ser0» na nultu. Isto tako, ako je nakon brisanja korisnik počeo brisati i naredbu, program treba postaviti zastavicu «com0_set» kako bi program znao da naredba nije završena.

```
      if (c==0x08) {          // Ako se radi o 'delete'
            ser0_broj-=2;     // Tada pokazivac stavi na zadnji točan znak
            if (ser0_broj>250) ser0_broj=0; // Ako je broj manji od 0
            if (com_broj>ser0_broj) com0_set=0; // Isto ako to zadire u
                              // komandu, tada makni zastavicu da je završena
      }
```

Ako se pak radi o znaku za prelazak u novi red, onda se niz «ser0» mora završiti sa null karakterom, i tada se poziva funkcija za tumačenje upisane naredbe «decode_command». Poslije toga postavi trenutnu poziciju unutar niza «ser0» na nultu i obriše zastavicu «com0_set».

```
      if (c==0x0a) {          // Ako se radi o 'Enter' tada izvrši
                              // naredbe koja je upisana
            ser0[ser0_broj]=0;// Završavam string sa '\0'

            decode_command(); // Protumači upisanu naredbu

            ser0_broj=0;      // Postavi broj upisanih znakova na 0
            com0_set=0;
      }
      }
   }
}
```

5.11. Funkcija za provjeru i identifikaciju broja pozivatelja

Kada netko pozove GSM uređaj on će ispisati obavijest o pozivu i broj pozivatelja ako isti postoji, tj. nije skriven. Ova funkcija će iz niza «ser1» izdvojiti broj pozivatelja. Točna specifikacija o tome što GSM uređaj šalje može se naći u literaturi [5]. GSM će poslati nešto kao:

2
+CLIP: +XXXXXXXXX,NNN

gdje X predstavlja broj pozivatelja, a N tip broja. Funkcija prvo provjeri dali je niz «ser1» u kojem se nalazi odgovor veći od 8 znakova, kako bi se uvjerio da zaista postoji broj.

```
int check_call (void) {
  unsigned char i,j;
  unsigned char broj[16];
  if (strlen(ser1)<8) return(-1);         // Ako nema +CLIP:
```

Nakon toga traži se znak «:» iza teksta «+CLIP». Sada se traži znak «+» koji se nalazi na početku broja pozivatelja. Sada varijabla «i» sadrži početnu lokaciju broja unutar niza «ser1».

```
  i=0;j=0;
  while (ser1[i]!=':') i++;
  while (ser1[i]!='+') i++;
```

Sada se unutar petlje broj pozivatelja prepisuje u niz «broj». Petlja će završiti kada se dođe do znaka «,». Ako se prekorači spremnik od 15 znakova, znači nešto nije u redu i funkcija vraća –1.

```
  while (ser1[i]!=',') {
      broj[j]=ser1[i];
      i++;j++;
      if (j>15) return (-1);              // Ako se prekorači spremnik
  }
```

Niz koji sadrži broj završava se null karakterom. Nakon toga se niz «broj» uspoređuje sa nizom «MSNO» koji sadrži broj operatera, i ako su nizovi jednaki upali se LED indikacija na izlazu PIO2_3. Indikacija će biti uključena narednih 15 sekundi do izvršenja petlje nižeg prioriteta. Ako nije broj pozivatelja jedan broju operatera, tada funkcija vraća –1. Inače vraća 0.

```
  broj[j]=0;
  if(strcmp(MSNO,broj)==0) PIO2_3=0;      // Ako je broj operatera upali LED
  else return (-1);
  return (0);
}
```

Razlog zašto se pali indikacija prilikom poziva operatera biti će detaljnije objašnjena unutar poglavlja «Rukovanje uređajem».

5.12. Funkcija za kodiranje i slanje kratkih poruka

Kao što je već opisano u poglavlju 2 način slanja i kodiranja kratkih poruka na GSM uređaj, tako je potrebno napraviti funkciju u izvršnom programu koja će tekst poruke kodirati i istu poslati prema pravilima. Funkcija «send_sms» radi upravo to: uzima broj primatelja iz niza «MSN» te tekst poruke iz niza «ser1» kojemu se dodaje informacija o sustavi koji poruku šalje, i takva se onda poruka kodira. Funkcija vraća –1 ako slanje nije uspjelo, odnosno 0 pri uspješnom slanju.

Prije svega potrebno je definirati korištene varijable i provjeriti dali je prisutan GSM uređaj. Tada se tekst poruke sprema u niz «buffer» kojemu se dodaje informacija o sustavu.

```
int send_sms (bit rc) {
  unsigned char i,j,j2,b,b2,c,c2,len_m,len_n;
  unsigned char xdata buffer[340],buffer2[340],buffer3[4];
  long wait_time=0xFFFFF;

  if (!gsm) return (-1);           // Ako GSM uređaj nije pronađen
  sprintf(buffer,"%s\n\nSent by SMS COMMANDER v%s",ser1,version);
```

Sada se tekst u nizu «buffer» pretvara u 7-bitni ASCII kod. Varijabla «len_m» sadrži dužinu poruke. Varijabla «len_n» sadrži dužinu broja primatelja.

```
  for (i=0;i<=strlen(buffer);i++) buffer[i]=toascii(buffer[i]);
  len_m=strlen(buffer);      // Duzina poruke
  len_n=strlen(MSN)-1;       // Duzina MSN broja
```

Slijedeća petlja radi rotaciju normalnog telefonskog broja u internacionalnom obliku u polu oktet kakav zahtijeva PDU format. Petlja se završava kada se dođe do zadnjeg broja. I niz «MSN» se završava null znakom.

```
  // Broj je potrebno staviti u semi octet!
  // Normalan broj     +12345678901      +123456789012
  // Semi oktet        2143658709F1      214365870921
  i=0;
  do {
     if (MSN[i+2]==0)              // Ako dođemo do kraja stringa
     MSN[i]='F';                   // Tada ubacujemo 'F'
     else
     MSN[i]=MSN[i+2];              // Ili normalno rotiramo 2 člana iza
     i+=2;
  } while (i<len_n);
  MSN[i]=0;
```

Sada se 7 bitni ASCII niz u kojega je smještena poruka jedan po jedan bit prebaci u niz «buffer2» prema PDU standardu.

```
                  // Konverzija 7 bit niza u 8 bit niz te u ASCII HEX
  j=0;j2=0;       // Polazim od početnih karaktera oba niza
  b=0x01;         // Prvi karakter ulaznog niza, uzima se 0 bit
                  // b = 0000 0001 (2)
  b2=0x01;        // Prvi karakter izlaznog bita stavlja se na 0 bit
                  // b2= 0000 0001 (2)
  i=2;            // Niti jedan niz nije došao do kraja
  c2=0;           // Prvi karakter ulaznog niza je '\0'
                  // Ovo je bitno zbog OR naredbe koja se kasnije radi
  do {
```

```
    c=buffer[j];            // Stavim karakter ulaznog niza u 'c'
    if (c==0) i=1;          // Ako smo došli do kraja ulaznog niza > i=1
    c&=b;                   // Izvrim C = (C AND b) maskiram samo 1 bit
    b=_crol_(b,1);          // Rotiram b u lijevo kako bi se slijedeći puta
                            // uzeo slijedeći bit ulaznog niza
    if (c != 0) c2|=b2;     // Ako je 'c' različit od 0, znaci maskirani
                            // bit je 1 => C2 = (C2 OR b2): Digni u 1
                            // taj bit koji se zapisuje na izlazu
    b2=_crol_(b2,1);        // Rotiraj b2 u lijevo kako bi se slijedeći
                            // puta zapisivalo u slijedeći bit

    if (b==0x80) {          // Ako je 'b' na početku (7-bit), znaci ulazni
                            // karakter je završen. Idemo na slijedeći
          b=0x01;           // Opet postavljamo masku na 0 bit
          if (i==2) j++;    // Ako nije kraj ulaznog niza postavi
                            // pokazivac na slijedeći karakter
    }
    if (b2==0x01){          // Ako je 'b' na početku, znaci izlazni
                            // oktet je završen.
          buffer2[j2]=c2;   // Zapisi 'c2' karakter u izlazni niz
                            // Na trenutno mjesto
          c2=0;             // Postavi prazan karakter radi kasnije
          b2=0x01;          // OR naredbe
          j2++;             // Postavi pokazivac na slijedeći karakter
          if (i==1) i=0;    // Ako je ulazni niz završen (i=1), tada
    }                       // Znaci da je kraj konverzije (i=0)
} while (i);
buffer2[j2]=0;
```

Sada je poruka konvertirana u 8 bitni niz prema PDU standardu te se nalazi u nizu «buffer2». No u tom obliku je neupotrebljiv. Kako bi se mogao poslati, potrebno je svaki polu oktet prebaciti u ASCII prezentaciju hexadecimalnog broja. To se radi u vrlo jednostavnoj petlji sa dvije varijable «i» i «j». Gotovi niz poruke nalazi se u nizu «buffer» koji je završen null znakom.

```
j=0;
for (i=0;i<strlen(buffer2);i++) {      // Uzimam po jedan karakter
    c=buffer2[i];                      // ulaznog niza
    sprintf(buffer3,"%bX",c);          // konvertiram ih u ASCII-HEX
                                       // koji je ujedno sada velik
                                       // 2 karaktera
          // Sada se svaki od njih zapisuje u izlazni niz
    if (buffer3[1]==0){    // Ako je broj manji od 0x10, dakle
                           // jednoznamenkasti, onda mu treba dodati
                           // 0 ispred
          buffer[j++]='0';buffer[j++]=buffer3[0];
    }else{                 // Inače je dvoznamenkasti broj veći od 0x10
          buffer[j++]=buffer3[0];buffer[j++]=buffer3[1];
    }
}
buffer[j]=0;          // U 'buffer' se nalazi konvertiran niz
```

Sada se dobiveni niz ukomponira u ostatak PDU bloka sa već definiranim postavkama i sprema u «buffer2». Jedino što korisnik može odabrati je dali želi informaciju o primljenosti poruke na strani primaoca ili ne. Isto tako, ako je broj primatelja poruke manji od 10, tada se konverzijom gubi vodeća 0, što je riješeno jednom if naredbom.

```
// Sada je taj niz potrebno ukomponirati u PDU standard
// Ako ne treba delivery report, tada promijeni '31' u '11'
if (rc==1) i='3'; else i='1';
```

```
  if (len_m<0x10) // Ako je broj manji, opet isti problem, manjka '0'
  sprintf(buffer2,"00%c1000%bX91%s0000AA0%bX%s",i,len_n,MSN,len_m,buffer);
  else
  sprintf(buffer2,"00%c1000%bX91%s0000AA%bX%s",i,len_n,MSN,len_m,buffer);
```

U nizu «buffer2» nalazi se kompletan PDU blok sa svih postavkama koje zahtjeva standard. Sada se određenom AT naredbom naredi GSM uređaju da prijeđe u mod za slanje poruka, pričeka se neko vrijeme, i onda se pošalje PDU blok. Na kraju naredbe se kaže GSM uređaju koliko okteta je dugačak PDU blok. PDU blok se završava «CTRL+Z».

```
  sprintf(ser1,"\nat+cmgs=%bd\n\r",(_cror_(strlen(buffer2),1))-1);
  gets1(0);       // Posalji naredbu za prelazak u PDU mod za slanje
  wait (10000);  // Pričekaj malo
  sprintf(ser1,"%s%c",buffer2,0x1a);
  printf1();                        // Posalji PDU blok
  ser1[0]=0;                        // Posalji prazan string
```

Nakon toga se čeka odgovor GSM uređaja, ali ograničeno vrijeme.

```
  while (!RI1 && wait_time) {       // Čekaj dok GSM vrati odgovor
     wd_r();                        // Resetiraj WDT
     wait_time--;
  }
```

Kada se pojavi odgovor, tada se provjeri dali je GSM uređaj vratio pogrešku ili da je slanje prošlo uredu. Ako je poruka poslana, program vraća 0, odnosno u slučaju greške vraća –1.

```
  gets1(1);                                     // Uzmi odgovor
  if (strncmp(ser1,"+CMGS:",6)==0) return (0); // Ako je slanje uredu
  return (-1);                 // Ako je došlo do greške u slanju SMS-a
}
```

5.13. Funkcija za čitanje i dekodiranje kratke poruke

Kao što je već opisano u poglavlju 2 način primanja i dekodiranja kratkih poruka sa GSM uređaja, stoga je potrebno napraviti funkciju u izvršnom programu koja će primiti poruku u PDU formatu i istu pretvoriti u tekst poruke prema pravilima PDU formata. Funkcija «read_sms» radi upravo to: uzima broj pošiljatelja iz PDU niza, te tekst poruke koja se dekodira natrag u ASCII 7 bitni zapis. Funkcija radi veoma slično kao «send_sms» samo što je konverzija u suprotnom smjeru. No svejedno veći dio koda je veoma sličan. Funkciji se predaju 2 parametra, prvi je «TYPE», a drugi «DEL». Prvi parametar je broj s kojim se kaže koju vrstu poruka želimo pročitati (0 – nove nepročitane poruke, 1 – stare primljene poruke, 4 – sve poruke), dok drugi parametar kaže dali te poruke želimo poslije čitanja obrisati ili ih ostaviti.

Na početku, funkcija inicijalizira korištene varijable i nizove. Poslije toga program resetira brojač petlje nižeg prioriteta kako bi se petlja odgodila sljedećih 15 sekundi. Nakon toga funkcija provjeri dali je prisutan GSM uređaj, te ako nije vraća –1. Nakon toga pošalje GSM uređaju naredbu za listanje kratkih poruka: «AT+CMGL=TYPE».

```
int read_sms (char TYPE, char DEL) {
  int br=-1;
  unsigned char c,c2;
  int i;
  unsigned int nbr,pos,pos2,pos3,j,j2;
  unsigned char len,tmp,b,b2,mes;
  unsigned char xdata buf[20*162];

  reset_timer0();                          // Resetiraj Timer0
  if(!gsm) return(-1);                     // Ako GSM uređaj nije pronađen
  sprintf(ser1,"at+cmgl=%bd\n\r",TYPE);    // Naredba za listanje poruka
  printf1();                               // Slanje naredbe na GSM
```

Sada čeka na odgovor GSM uređaja i to ograničeno vrijeme. Svaki puta petlja resetira brojač nadzornog sklopa kako ne bi došlo do neželjenog reseta procesora dok se čeka na odgovor GSM uređaja. Nakon toga se postave broj upisanih znakova «nbr» na 0 i niz «buf» se zaključi null znakom.

```
  while((br--) && (!RI1)) wd_r();          // Čekaj neko vrijeme na odgovor

  nbr=0;                                   // Postavi broj upisanih znakova na 0
  buf[0]=0;                                // Završi niz sa '\0'
  i=-1;
```

Slijedeća petlja uzima znak po znak odgovora od GSM uređaja i sprema ga u niz «buf». Ako se prekorači veličina niza, tada se izlazi iz petlje. Isto tako prvi znak za prelazak u novi red se zanemaruje. Svaki puta se resetira brojač nadzornog sklopa kako bi se izbjegao neočekivani reset procesora.

```
  do {
      if (RI1){                // Ako se pojavi karakter na ulazu serije1
            c=SBUF1;                       // Uzmi karakter sa serije1
            buf[nbr++]=c;                  // spremi ga u buffer
            i=-1;                          // Postavi odbrojavanje
            if (nbr>3200) i=0;             // Ako smo prekoračili buffer
            if (c==0x0a || c==0x0d) {      // Ako se radi o 'Enter'
                  if (nbr<=1) nbr=0;
            }
```

```
            RI1=0;                          // Uzeo znak sa serije1
        }
        wd_r();
    } while (i--);
```

Niz «buf» se završava se null znakom, i pričeka se neko vrijeme. Tada program provjeri dali uopće postoji i jedna poruka ili ih nema. Ako nema poruka, funkcija vraća –1.

```
buf[nbr]='\0';                          // Završavam string sa '\0'
buf[nbr+1]=0;
wait(50);                               // Malo Pričekaj i resetiraj WDT
if(nbr<8) return (-1);                  // Nema SMS poruka
```

Sada se čitav odgovor dekodira prema potpoglavlju 2.4. Unutar ovih nekoliko petlji se traži broj kratke poruke unutar spremnika na SIM kartici na GSM uređaju. Naravno broj je zapisan u ASCII kodu kojega je potrebno prikladno pretvoriti u klasičan 8 bitni broj. Nakon toga se obavijesti korisnika o tome. Ovaj broj je ujedno bitan kako bi se ista poruka mogla obrisati iz spremnika na GSM uređaju.

```
pos=0;                                  // Trenutni pokazivac u stringu
do {
    if (buf[pos]!='+') return(-1);      // Provjerim dali počinje sa '+'
    while(buf[++pos]!=',');             // Tražim prvi ','
    mes=0;                              // Broj poruke=0
    pos2=pos-1;                         // Sada odbrojavam u lijevo
    while((c=buf[--pos])!=' ') {        // Do razmaka i pročitam broj
          if ((pos2-pos)==0) mes+=(c-48);           // poruke (3)
          if ((pos2-pos)==1) mes+=((c-48)*10);
    }
    printf("\n SMS Number: %bu\n",mes);
```

Sada se iz PDU bloka izdvaja broj pošiljatelja poruke i dužina korisniče poruke. Broj se sprema u MSN, nakon čega se iz polu okteta vraća u klasičan ASCII kod.

```
    len=0;                              // Duzina čitavog PDU bloka bez SMSC
    while(buf[++pos]!='\n');
    pos--;
    pos2=pos-1;
    while((c=buf[--pos])!=',') {
          if ((pos2-pos)==0) len+=(c-48);
          if ((pos2-pos)==1) len+=((c-48)*10);
          if ((pos2-pos)==2) len+=((c-48)*100);
    }                                   // Izdvajam taj broj (22)
    while(buf[pos++]!='\n');
    pos2=pos;                           // Početak PDU bloka
    pos+=(hex_char (buf[pos],buf[pos+1])+1)*2;
    pos+=2;
    pos2=pos;                           // Početak bloka sa brojem pošiljatelja
    tmp=(hex_char(buf[pos],buf[pos+1]));      // Duzina broja
    pos3=pos+tmp+4;                           // Na kraj broja
    pos+=4;                                   // Na početak broja
    if (tmp&0x01) pos3++;                     // Ako je neparan broj
    c=0;
    do MSN[c++]=buf[pos++]; while (pos<pos3);
    MSN[c++]=0;
    MSN[c--]=0;
    do {
          MSN[c]=MSN[c-2];
```

```
        c-=2;
    } while (c>0);
    MSN[0]='+';
    MSN[tmp+1]=0;
    printf(" MSN Number: %s\n",MSN);

    pos=pos3;                                   // Iza broja PDU
    pos+=18;                                    // Iza timestamp
    pos3=pos;
    tmp=hex_char(buf[pos],buf[pos+1]);          // Duzina poruke
    pos+=2;
```

Sada se program trenutno nalazi na početku korisničke poruke unutar PDU bloka. Prema pravilima kodiranja PDU bloka, program dekodira 8 bitni hexadecimalni ASCII zapis u klasični 7 bitni ASCII tekstualni zapis. Dekodirana poruka se sprema u niz «ser1».

```
    j=pos;j2=0;         // Polazim od početnih karaktera oba niza
    b=0x01;             // Prvi karakter ulaznog niza, uzima se 0 bit
                        // b = 0000 0001 (2)
    b2=0x01;            // Prvi karakter izlaznog bita stavlja se na 0 bit
                        // b2= 0000 0001 (2)
    i=2;                // Niti jedan niz nije došao do kraja
    c=0;
    c2=0;               // Prvi karakter ulaznog niza je '\0'
                        // Ovo je bitno zbog OR naredbe koja se kasnije radi
do {
    c=hex_char(buf[j],buf[j+1]);
                        // Stavim karakter ulaznog niza u 'c'
    c&=b;               // Izvrim C = (C AND b) maskiram samo 1 bit
    b=_crol_(b,1);      // Rotiram b u lijevo kako bi se slijedeći puta
                        // uzeo slijedeći bit ulaznog niza
    if (c != 0) c2|=b2;     // Ako je 'c' različit od 0, znaci maskirani
                            // bit je 1 => C2 = (C2 OR b2): Digni u 1
                            // taj bit koji se zapisuje na izlazu
    b2=_crol_(b2,1);        // Rotiraj b2 u lijevo kako bi se slijedeći
                            // puta zapisivalo u slijedeći bit

    if (b==0x01) {          // Ako je 'b' na početku (8-bit), znaci ulazni
                            // karakter je završen. Idemo na slijedeći
          b=0x01;           // Opet postavljamo masku na 0 bit
          if (i==2) j+=2;   // Ako nije kraj ulaznog niza postavi
                            // pokazivac na slijedeći karakter
    }
    if (b2==0x80){          // Ako je 'b2' na početku, znaci izlazni
                            // oktet je završen.
          ser1[j2]=c2;      // Zapisi 'c2' karakter u izlazni niz
                            // Na trenutno mjesto
          c2=0;             // Postavi prazan karakter radi kasnije
          b2=0x01;          // OR naredbe
          j2++;             // Postavi pokazivac na slijedeći karakter
          if (j2>=tmp) i=0;
    }                       // Znaci da je kraj konverzije (i=0)
    wd_r();                 // Reset WDT-a
} while (i);
```

Poruka se završava null karakterom, i ispisuje na terminal.

```
    ser1[j2]=0;
    ser1[j2+1]=0;
    printf("\n%s\n",ser1);
```

Ako je broj pošiljatelja poruke jednak broju operatera, tada program poziva funkciju za dekodiranje korisničke naredbe unutar kratke poruke. Na ovaj način je omogućeno operateru da mijenja stanja paralelnih digitalnih izlaza. Isto tako ako je parametar funkcije «DEL» postavljen, program će obrisati poruku iz spremnika na GSM uređaju.

```
      // Ako je broj pošiljatelja jednak operateru, izvrši SMS
      if (strcmp(MSN,MSNO)==0) {
            decode_sms();
            sprintf(ser1,"\nat+cmgd=%bu\n",mes); gets1(0);
      }
      if (DEL) {  // Ako je uključeno brisanje poruka, tada iste obrisi
            sprintf(ser1,"\nat+cmgd=%bu\n",mes); gets1(0);
      }
```

Sada se petlja nastavlja i provjerava dali postoji još jedna poruka unutar odgovora kojeg je program spremio u niz «buf». Ako više ne postoji poruka, tada o tome izvijesti korisnika.

```
      pos=j;
      while(buf[++pos]!='\n' && buf[pos-1]!=0 && pos<nbr);
      pos++;
  } while (buf[pos]=='+');
  printf("\n End of messages.\n");
  return(0);
}
```

6. NAČIN KORIŠTENJA UREĐAJA I PROGRAMSKE PODRŠKE

Nadzorno računalo je prvenstveno zamišljeno kao razvojno okruženje za programsku podršku. Upravo ugrađena ROM emulacija mu daje tu posebnost. Pored toga posjeduje izuzetno snažan mikro procesor u svojoj porodici Intel 80C31 kompatibilnih mikro procesora. Veliki broj paralelnih digitalnih ulaza-izlaza, 2 serijske sabirnice i CAN sabirnica mu daju tu posebnost razvojnog okruženja. No kako je ovaj projekt usmjeren u pravcu upravljanja procesima putem usluge kratkih poruka, u tom smjeru će ići i ovo poglavlje, te ću opisati način rada sa programskom podrškom razvijenom za projekt, ali ću posvetiti jedno potpoglavlje za razvoj programske podrške.

Način rada sa razvijenom programskom podrškom za nadzorno računalo je više nego jednostavno i lako se uči. Pored toga postoje naredbe za dobivanje pomoći, i ako korisnik upiše neku naredbu krivo, programska podrška će ga uputiti kako se ispravno koristi neka naredba.

Dakle u narednim potpoglavljima će biti opisana većina naredbi koju podržava izvršni program nadzornog računala trenutne inačice 0.50.

Kao što sam već naveo, prvo potpoglavlje biti će uvod za korisnika kako koristiti razvojno okruženje za mikro kontrolore 80C31, naravno usmjereno radu na nadzornom računalu.

Slijedeće potpoglavlje je posvećeno načinu komunikacije osobnog računala sa nadzornim računalom i način korištenja terminal programa, te način punjenja programske memorije novo inačicom izvršnog programa.

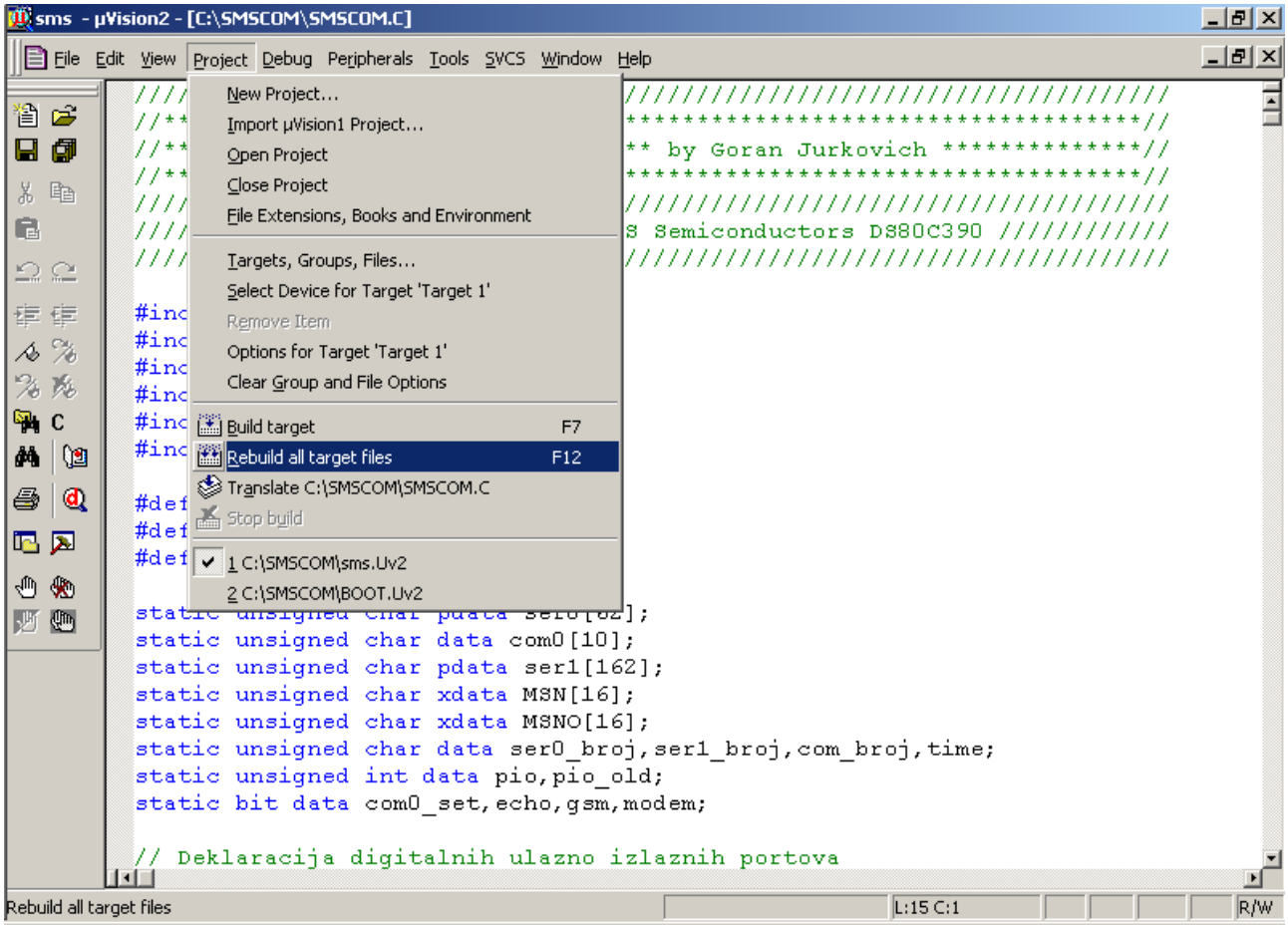
Naredna poglavlja opisuju rad sa nadzornim računalom i naredbe za interakciju korisnika i programa.

6.1. Način korištenja razvojnog okruženja Keil za razvoj programske podrške

Najpopularnije razvojno okruženje za mikro-kontrolore iz porodice Intel 80C31 je Keil programski paket microVision inačica 2. Instalacija ovog paketa nalazi se na CD-u u prilogu. Kako bi sve sigurno radilo ispravno, potrebno je microVision instalirati u mapu «C:\WINPROGS\KEIL\». Isto tako potrebno je raspakirati zip datoteku u kojoj se nalazi programska podrška za nadzorno računalo u mapu «C:\SMSCOM\». Kako bi instalacija bila potpuna, potrebno je pokrenuti datoteku «C:\SMSCOM\BackupSC.BAT».

Sada je u microVision programu potrebno otvoriti projekt nadzornog računala izvršni program odabirom u izborniku «Project» i onda «Open Project» i onda se odabere datoteka «C:\SMSCOM\SMS.UV2».

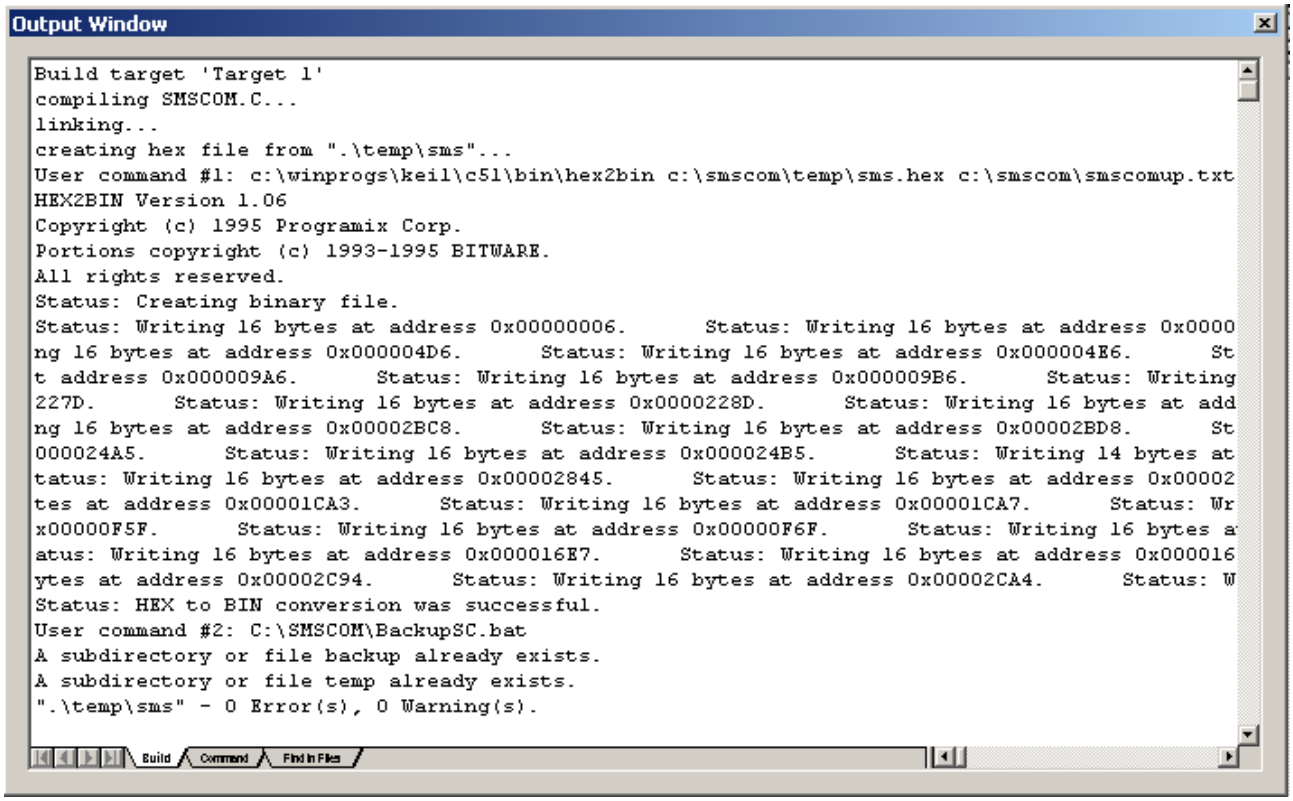
Kada se otvori taj projekt, tada se dobije ispis programa kao na slici 6.1. Unutar datoteka sms.uv2 i sms.opt nalaze se postavke projekta. Ako je sve instalirano kao što je navedeno, tada će se jednostavnim odabirom u izborniku «Project» i zatim «Rebuild all target files» kao što je prikazano na slici 6.1. program prevesti, i napraviti sve potrebne datoteke, kao i binarna datoteka potrebna za prijenos u nadzorno računalo putem serijske asinkrone sabirnice.



Slika 6.1. Izgled programskog paketa Keil microVision2 sa otvorenim projektom.

Kada se pokrene gore navedena akcija, tada ćemo dobiti na ekranu prozor prikazan na slici 6.2. Ovdje se vidi dali je prevođenje C jezika prošlo uredu, te dali su sve ostale konverzije prošle uredu.

Ako nema grešaka, kao što je to prikazano na slici 6.2. tada je datoteka za prenošenje na nadzorno računalo spremna.



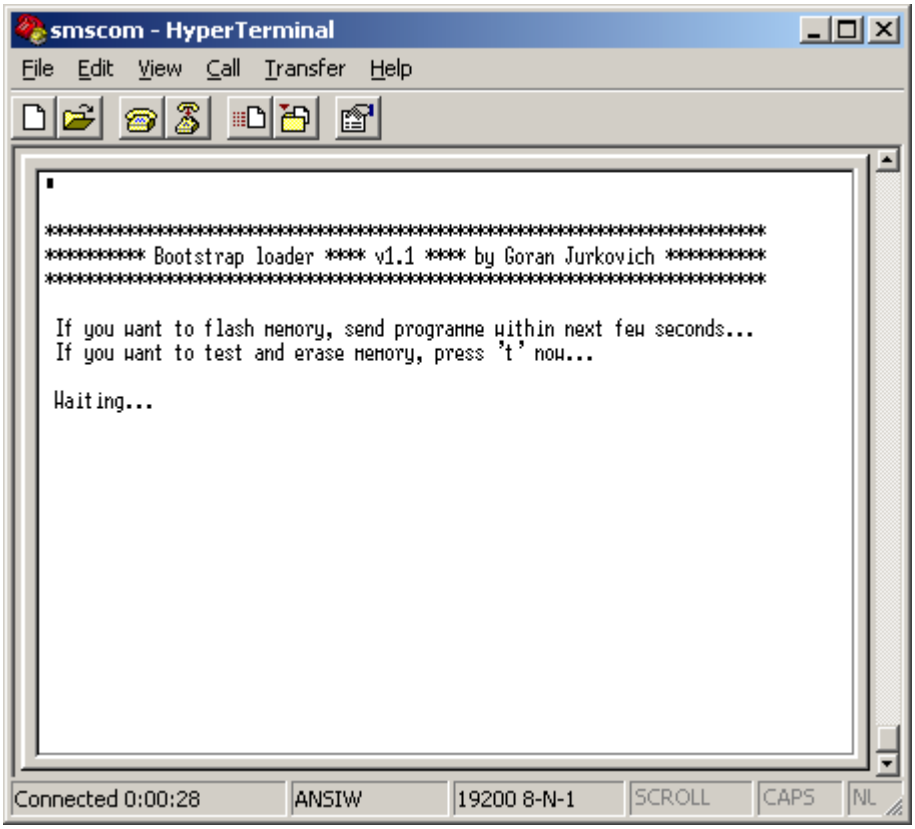
Slika 6.2. Izgled izlaznog prozora prilikom prevođenja programa.

Pored toga što prethodna naredba radi datoteku «C:\SMSCOM\SmsComUP.TXT» koja je potrebna za prijenos na nadzorno računalo, prethodna naredba će izvesti i spremanje pomoćne kopije trenutnog C programa u mapi «C:\SMSCOM\BACKUP\» sa trenutnim datumom i vremenom. Ova opcija je veoma zanimljiva i potrebna prilikom razvoja programske podrške, jer se ponekad dogodi da nova inačica programa ne radi, a ne sjećamo se što smo u program dodali. Ovako se na vrlo jednostavan način može doći do prethodnih inačica izvršnog programa. Za prijenos i komunikaciju sa nadzornim računalom najjednostavnije je koristiti Microsoft HyperTerminal koji dolazi kao dio operativnog sustava. Detalji oko korištenja istog nalaze se u slijedećem potpoglavlju.

6.2. Komunikacija nadzornog računala i osobnog računala, HyperTerminal

Kao što sam već naveo u prethodnom potpoglavlju najjednostavnije je koristiti Microsoft HyperTerminal za komunikaciju sa nadzornim računalom. HyperTerminal je dio operativnog sustava Microsoft Windows 95 ili noviji. Ako isti nije instaliran potrebno ga je instalirati ulaskom u «Control Panel» zatim odabirom «Add/Remove Programs» i zatim jahačem «Microsoft Windows» pronaći pod «Communications» «HyperTerminal» i selektirati kvačicu. No pretpostavljam da operativni sustav u većini slučajeva ima instaliran HyperTerminal.

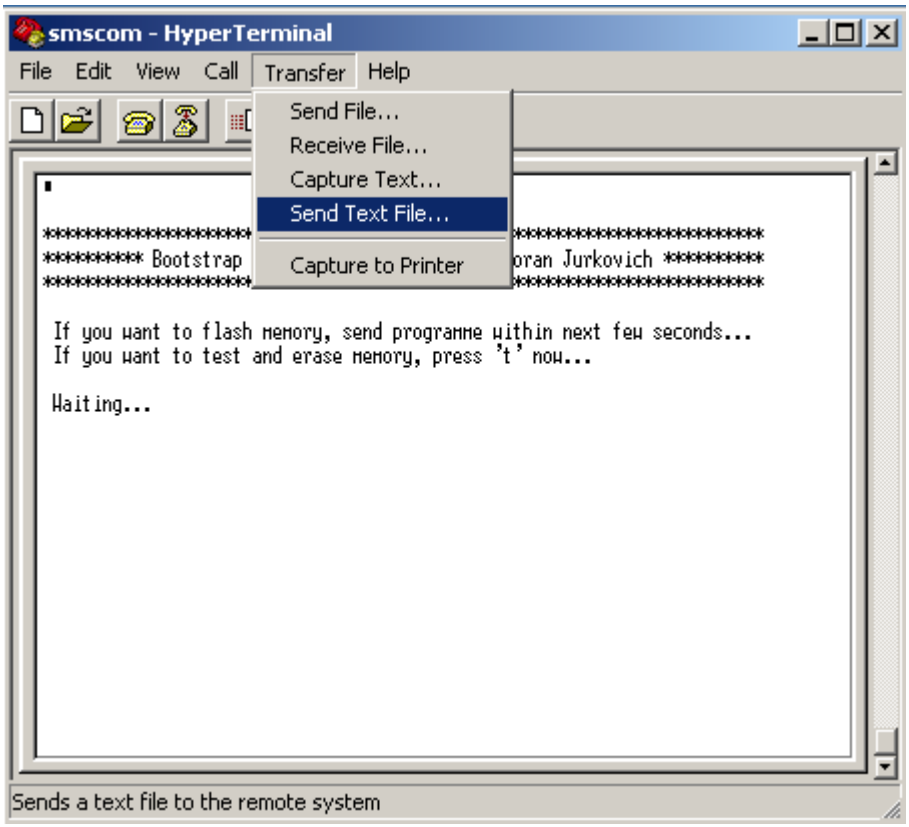
Ovdje je korišten HyperTerminal iz Microsoft Windows 2000, i izgled istog prikazan je na slici 6.3. Postavke komunikacije mogu se pozvati otvaranjem datoteke «C:\SMSCOM\SMSCOM.HT». No moguće da ta datoteka neće raditi sa svim inačicama HyperTerminal programa. Ako se dogodi da datoteka ne radi, onda je potrebno ručno namjestiti postavke komunikacije. Potrebno je odabrati ispravan serijski priključak na osobnom računalu na koji je spojeno nadzorno računalo (npr. COM1). Zatim potrebno je postaviti brzinu komunikacije na 19200 bps, bez bita pariteta i sa jednim stop bitom. «Handshake» može biti «hardware».



Slika 6.3. Izgled HyperTerminala sa pokrenutim nadzornim računalom.

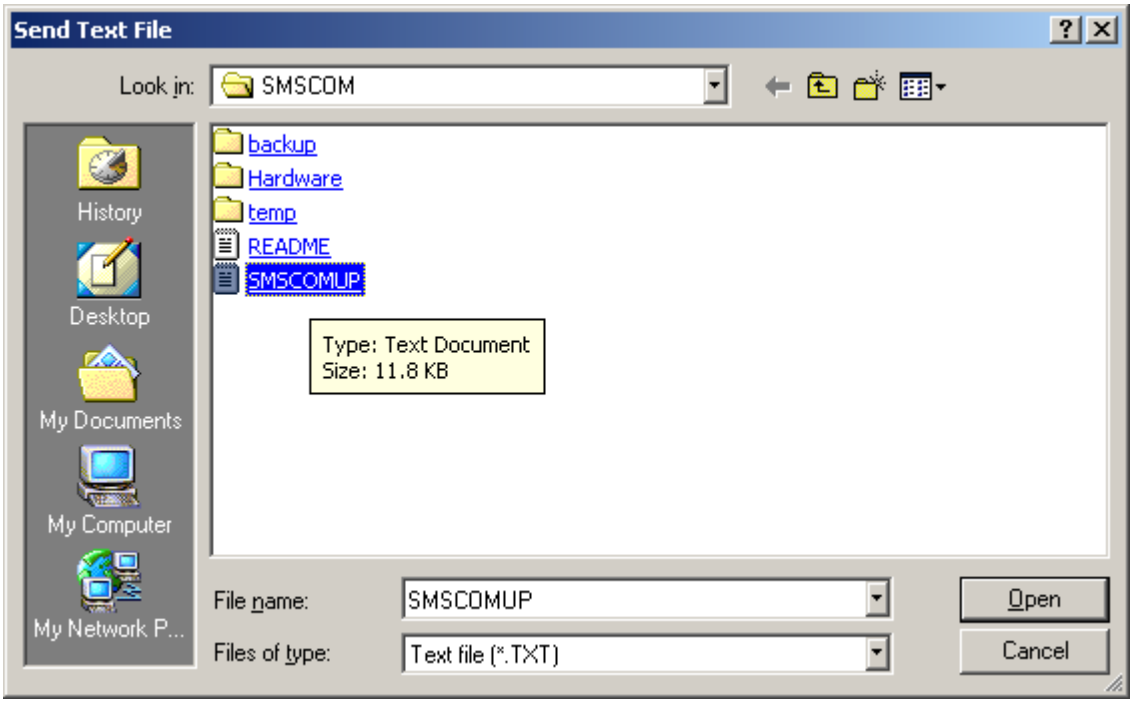
Ako su komunikacija i postavke uredu, nakon paljenja nadzornog računala dobiti ćemo tekst kao što je to prikazano na slici 6.3. Sada korisnik ima 5 sekundi za slanje nove inačice izvršnog programa ili može pritisnuti tipku «t» za test programske memorije.

Ako korisnik želi poslati nadzornom računalu novu inačicu onda to može učiniti odabirom naredbe «Transfer» u izborniku, zatim «Send Text File», kao što je to prikazano na slici 6.4.



Slika 6.4. Način slanja nove inačice nadzornom računalu.

Nakon toga otvara se dijalog sa upitom koja se datoteka želi poslati. Izaberemo «C:\SMSCOM\SmsComUP.TXT» i pritisnemo gumb «Open». Izgled dijaloga za odabir datoteke prikazan je na slici 6.5.



Slika 6.5. Izgled dijaloga za slanje programa.

I nakon toga pričekamo do 3 minute dok se program pošalje. Koliko će trajati prijenos programa ovisi o njegovoj veličini. Veličina trenutne inačice izvršnog programa je oko 15KB čiji prijenos traje manje od 1 minute.

Ovaj odabir datoteke za slanje kao što je prikazano u prethodnim koracima, potrebno je napraviti unutar 5 sekundi. Nakon toga ako korisnik ne pošalje nadzornom računalu novu inačicu programa, pokretački program izvršava trenutnu inačicu izvršnog programa koja već postoji u programskoj memoriji.

Ako prvi puta šaljemo program, onda je najjednostavnije na samom startu pritisnuti tipku «t» čime će se pokrenuti testiranje programske memorije i nakon toga brisanje iste. Tada će pokretački program ustanoviti kako je memorija prazna i čekati da mu korisnik pošalje izvršni program. Na ovaj način korisnik može polako odabrati koji program želi poslati.

Tekst koji će nadzorno računalo ispisati nakon pritiska slova «t» izgleda ovako:

```
**********************************************************************
********** Bootstrap loader **** v1.1 **** by Goran Jurkovich **********
**********************************************************************

 If you want to flash memory, send programme within next few seconds...
 If you want to test and erase memory, press 't' now...

 Waiting...

Testing memory...

Filling memory with AAh up...  60935 60939 60943 61191 61195 61199
Filling memory with AAh down...  61199 61195 61191 60943 60939 60935
Filling memory with 55h up...  60935 60939 60943 61191 61195 61199
Filling memory with 55h down...  61199 61195 61191 60943 60939 60935
Filling memory with pattern...  60935 60939 60943 61191 61195 61199
Filling memory with 00h...

ERROR in memory!
```

Brojevi koje je ispisao su lokacije u memoriji gdje je pronađena pogreška. To su lokacije iznad 60KB. Razlog zašto su se tu pojavile pogreške već je objašnjen u poglavlju 4. ali ukratko ću ponoviti. Radi se o mapiranju internih 4KB memorije na posljednja 4KB vanjske programske memorije. Unutar tih 4KB interne memorije nalaze se i specijalni registri za CAN sabirnicu i neke druge stvari i tu dolazi do pogreške.

Nakon što program prijavi grešku procesor se resetira i pokuša izvršiti program u programskoj memoriji. No kako je prošla akcija pored testa i obrisala memoriju, pokretački program će ispisati slijedeću poruku i čekati da mu korisnik pošalje program:

```
 Memory is empty...
 Please, flash memory...
```

Kada mu korisnik pošalje program, ili ako već postoji, pokretački program će ispisati slijedeću poruku, prebaciti način rada memorija i nakon reseta izvršiti poslani program u memoriji:

```
Loading programme...
```

Na ekranu ćemo dobiti informaciju o izvršnom programu, zatim informaciju o spojenom GSM uređaju ako je sve uredu.

```
*****************************************************************
******* SMS COMMANDER **** v0.50 **** by Goran Jurkovich **********
*****************************************************************

 GSM Station Vendor is SIEMENS
 GSM Station Model is C35i
 GSM Station Software is SIEMENS Gipsy Soft Protocolstack V2.540

 Type 'help' to get some assistance...
```

Za normalan rad nadzorno računalo, odnosno izvršni program zahtjeva samo spoj sa GSM uređajem. Ako isti nije detektiran, program će ispisati poruku pogreške… No ako je sve u redu, rad možemo započeti unosom naredbe «HELP».

```
help
OK!

SMS COMMANDER version 0.50 list of commands:

HELP          This list of commands
*             Repeat last command
INFO          Information about hardware and software
TIME          Information about date and time
TEST          Data memory test and erase
RESET         Restarting programme
OFF           Shuts down the processor
FLASH         Program memory upload over serial connection
ECHO b        Direct redirection of answers from the GSM to the terminal
              without decoding, b: 0 - disable, 1 - enable, or nothing
AT            Redirected command to the GSM station
MODEM         Permanent redirection to the GSM station
OUT1 bbbbbbb Setting PIO port 1 bits 7 to 0, b:<0,1>
OUT2 bbbb     Setting PIO port 2 bits 3 to 0, b:<0,1>
INPUT         Reading PIO ports 1 and 2
SENDSMS N     Sends SMS to number 'N', 'N': in MSN format: +385XXXXXXXXX
READSMS X Y   Reads SMS X: 0 - REC UNREAD, 1 - REC READ, 4 - ALL
                        Y: 1 - delete SMS after read
OPMSN         Operater's phone number in MSN format: +385XXXXXXXXX
```

Nadzorno računalo će ispisati sve podržane komande kao što je to ovdje prikazano.

6.3. Naredba za dobivanje informacija o sustavu

Naredba je veoma jednostavna a daje neke osnovne informacije o sustavu. Jednostavno nadzornom računalu pošaljete naredbu «INFO». Nadzorno računalo će odgovoriti sljedeće:

```
info
OK!

SMS Commander information:
SMS Commander software version 0.50...
SMS Commander hardware version 1.0...

GSM station information:
GSM Station Vendor is SIEMENS
GSM Station Model is C35i
GSM Station Software is SIEMENS Gipsy Soft Protocolstack V2.540

PIO ports information:
PIO: 1111 11111111

Now is 01/09/10 20:00:43

Operater's phone number is +38598754369
```

Dakle, program ispiše inačicu programa, inačicu uređaja. Zatim ispiše proizvođača i model GSM uređaja, te ispiše programsku podršku odnosno komandni procesor GSM uređaja. Nakon toga ispiše trenutno stanje paralelnih digitalnih ulaza, i telefonski broj operatera.

6.4. Promjena telefonskog broja operatera

Operater je osoba kojoj se šalju putem kratkih poruka informacije o promjeni stanja digitalnih ulaza. Isto tako operater može kratkom porukom bilo gdje iz svijeta promijeniti stanje paralelnih digitalnih izlaza. Zamjena telefonskog broja operatera se izvodi naredbom «OPMSN», kao što je prikazano u primjeru:

```
opmsn +38598544574
OK!
```

Za provjeru i ispis trenutnog operatera upišite samo «OPMSN»:

```
opmsn
OK!

 Operater's phone number is +38598544574
```

Program će ispisati trenutni telefonski broj operatera.

NAPOMENA: Promjenjeni broj operatera će biti aktivan samo do reseta procesora ili gašenja nadzornog računala.

6.5. Naredba za izmjenu stanja paralelnih digitalnih izlaza.

Naredba «OUT1» odnosno «OUT2» se može napisati na komandnoj liniji u terminalu ili poslati putem kratke poruke. U oba slučaja program će promijeniti stanje izlaza.

Za demonstraciju rada odabrano je tako da se na PIO2 nalaze svjetleće diode koje indiciraju stanje logičke 0 na pojedinom bitu. No i stanje PIO1 izlaza-ulaza moguće je mijenjati ili programski stavljanjem logičke 0, ili stavljanjem kratkospojnika na nadzornom računalu. Dakle, svaki bit paralelnih digitalnih ulaza-izlaza je dvosmjerni.

PIO1 ima 8 bitova, dok PIO2 ima 4 bita. Naredba «INPUT» će ispisati trenutno stanje oba paralelna digitalna porta:

```
input
OK!

PIO: 1111 11111111
```

Prva 4 bita (u ovom slučaju 1111), odgovaraju PIO2 portu na kojega su priključene svjetleće diode. Bitovi predstavljaju redom PIO2_3 do PIO2_0. Drugih 8 bitova predstavljaju PIO1 port. Ovaj port je predviđen za stavljanje kratkospojnika kako bi se neki ulaz prebacio u logičku 0. Redom bitovi predstavljaju PIO1_7 do PIO1_0. Kada su ulazi-izlazi u stanju logičke 1, tada je svaki bit moguće povući u logičku 0 ili putem programske podrške ili na nadzornom računalu sa kratkospojnicima.

Sljedećom naredbom želimo upaliti drugu svjetleću diodu. To ćemo napraviti naredbom «OUT2»:

```
out2 1011
OK!
```

Svjetleća dioda se zaista upalila. Nakon 15 sekundi neaktivnosti korisnika, program će izvršiti petlju nižeg prioriteta koja provjerava stanja ulaza i ako ima promjene ispiše trenutno stanje i pošalje obavijest operateru putem usluge kratkih poruka. To izgleda ovako:

```
 PIO: 1011 11111111
 Automatic SMS Sent!
```

Operater će primiti ovakvu poruku. Uočite kako je isti tekst napisan i na zaslonu udaljenog GSM uređaja. Funkcija za slanje kratkih poruka, automatizirano dodaje i tekst «Sent by SMS COMMANDER v0.50» kako bi primatelj poruke ima u vidu da se ovdje radi o automatiziranom slanju poruke:

```
 SMS Number: 3
 MSN Number: +38598544574

PIO: 1011 11111111


Sent by SMS COMMANDER v0.50

 End of messages.
```

Isto tako udaljeni operater može poslati naredbu «OUT» putem usluge kratkih poruka. Nadzorno računalo kada istu dobije će izvršiti komandu. Dakle operater je poslao poruku u kojoj piše «OUT2

1111». Nadzorno računalo kada dobije tu poruku, i to nakon 15 sekundi neaktivnosti provjerava stanje tj. dali ima novih poruka, i pročita istu:

```
SMS Number: 3
 MSN Number: +38598544574

out2 1111
Ovo je proba slanja SMS poruke i naredbe.
Sa ENTER se moze preci u novi red.


Sent by SMS COMMANDER v0.50

 End of messages.
```

Program je ugasio prethodno upaljenu svjetleću diodu. Nakon 15 sekundi neaktivnosti korisnika, program će ponovno provjeriti stanje ulaza, i vidjeti da se stanje PIO2_2 promijenilo natrag u 1. O tome će izvijestiti korisnika i poslati obavijest operateru:

```
PIO: 1111 11111111
 Automatic SMS Sent!
```

Operater će dobiti sljedeću poruku:

```
 SMS Number: 3
 MSN Number: +38598544574

PIO: 1111 11111111


Sent by SMS COMMANDER v0.50

 End of messages.
```

Prethodni proces daljinskog upravljanja stanjima izlaza, ili primanje obavijesti o promjeni stanja putem kratkih poruka je dosta spor proces i traje od 15 sekundi na više, ovisno o kašnjenju slanja poruka kroz GSM mrežu. Dakle to ima smisla za sporije procese. No neki proces može zahtjevati hitnu obavijest operatera, kao npr.: «alarm, odnosno obavijest o nasilnom ulasku u objekt». Tada kašnjenje GSM mreže i nadzornog računala (15 sekundi) može biti previše sporo. PIO1_7 je rezerviran za brzu detekciju logičke 0. Kada vanjski proces promijeni stanje PIO1_7 u logičku 0, istog trenutka nadzorno računalo detektira promjenu i nazove operatera. Operater kada vidi da ga zove nadzorno računalo, prekida poziv i zna da se radi o brzoj signalizaciji. Nakon 15 sekundi dobiti će i obavijest o promjeni stanja ulaza, tako da brza signalizacija može biti korištena za više procesa. Dakle, nadzorno računalo detektira promjenu, i obavijesti korisnika o tome:

```
 High priority signal detected!
```

Nakon što operater prekine poziv, program ispiše:

```
 Operator respond OK!
```

15 sekundi nakon neaktivnosti korisnika, ako je taj ulaz još uvijek u promijenjenom stanju, program će poslati kratku poruku operateru.

Isto tako, slanje udaljenih naredbi nadzornom računalu putem usluge kratkih poruka može biti sporo za neke brže procese, kao npr.: «otvaranje vrata objekta». Jednostavnije je da operater nazove nadzorno računalo. Kada nadzorno računalo detektira poziv automatizirano mijenja stanje izlaza PIO2_3 u logičku 0, i tako ju drži 15 sekundi. Kašnjenje ovakvog tipa naredbe je manje od 5 sekundi koliko traje uspostava poziva unutar GSM mreže. Kada operater nazove nadzorno računalo, program će ispisati:

```
High priority command detected!
```

Odmah iza toga program pali svjetleću diodu koja se nalazi na PIO2_3, i gasi ju nakon 15 sekundi neaktivnosti korisnika za terminalom. Moram napomenuti kako se ovi pozivi ne plaćaju, jer nadzorno računalo koristi funkciju CLIP (identifikacija broja pozivatelja) i na osnovi broja ustanovi dali se radi o operateru.

6.6. Naredba za slanje i čitanje kratkih poruka

Poruku je moguće poslati iz terminala, a poruku šalje nadzorno računalo putem GSM uređaja. Naredba za slanje poruka je «SENDSMS» a primjer izgleda ovako:

```
sendsms +38598544574
OK!

 Sending SMS to MSN: +38598544574
 Now, enter your text for SMS, and finish with 'CTRL+Z' or 'ESC'
```

Sada upišemo tekst poruke i završimo ga sa «CTRL+Z» ili «ESC».

```
out2 1111
Ovo je proba slanja SMS poruke i naredbe.
Sa ENTER se moze preci u novi red.


 Sending SMS...
 SMS sent!
```

Poruka je uspješno poslana. Naredbom «READSMS» čitamo poruke iz spremnika na SIM kartici unutar GSM uređaja, ili novo pridošle poruke. Što ćemo pročitati se određuje brojem iza naredbe.

Prvi broj označava:
0 – Nove nepročitane poruke
1 – Primljene pročitane poruke
4 – Sve poruke

Drugi broj označava:
0 – Ništa ne radi
1 – Prikazane poruke briše iz spremnika na SIM kartici

Slijedeća naredba će izlistati nove nepročitane poruke i obrisati ih:

```
readsms 0 1
OK!

 SMS Number: 3
 MSN Number: +38598544574

out2 1111
Ovo je proba slanja SMS poruke i naredbe.
Sa ENTER se moze preci u novi red.


Sent by SMS COMMANDER v0.50

 End of messages.
```

Slijedeća naredba će ispisati sve primljene poruke, te obrisati ih:

```
readsms 1 1
OK!
```

```
 SMS Number: 2
 MSN Number: +38598872630

sta to nas veze jos veze?

 End of messages.
```

Ako sada pokušamo ponoviti istu naredbu dobiti ćemo sljedeće:

```
readsms 1 0
OK!

 ERROR reading messages, or there is no messages!
```

Program je pokušao pročitati poruke, no kako istih nema, prijavljuje grešku.

6.7. Naredba za reset i gašenje nadzornog računala

Naredba za reset nadzornog računala u biti nema neko veliko značenje jer postoji kratkospojnik za reset procesora. Ona bi dobila smisao ako se nadzorno računalo nalazi ugrađeno u instalaciju objekta, a terminal odnosno osnobno računalo spajamo udaljeno. Tada se može dogoditi da prilikom razvoja programa trebamo napraviti reset procesora. Naredba je jednostavna i samo se napiše «RESET», a program će odgovoriti:

```
reset
OK!
Restarting...


**********************************************************************
******* SMS COMMANDER **** v0.50 **** by Goran Jurkovich **********
**********************************************************************
```

Isto tako naredba za gašenje, ustvari stavlja procesor u takozvani SLEEP mod rada. Procesoru se zaustavlja signal takta i potrošnja je minimalna. Isto tako ova naredba nema previše smisla i ugrađena je isključivo u razvojne svrhe. Naredba se poziva sa «OFF»:

```
off
OK!
Press reset to power up processor...
Turning power down...
```

6.8. Naredba za prelazak u pokretački program te slanje nove inačice programa

Kada u radu i testiranju nadzornog računala razvijamo programsku podršku, te želimo novu inačicu izvršnog programa staviti na nadzorno računalo, to jednostavno pozovemo naredbom «FLASH». Program će preći na prvi način rada memorija, i napraviti reset procesora. Tada će se izvesti pokretački program što će omogućiti korisniku prijenos nove inačice programa. To izgleda ovako:

```
flash
OK!

 Going to load bootstrap again...
 Prepare to upload new programme...


**********************************************************************
********** Bootstrap loader **** v1.1 **** by Goran Jurkovich **********
**********************************************************************

 If you want to flash memory, send programme within next few seconds...
 If you want to test and erase memory, press 't' now...

 Waiting...
```

Što je više nego jednostavno. ROM emulacija nadzornog računala mu omogućuje vrlo naprednu funkciju kao što to imaju ugrađeno neki mikro procesori poput Siemens C167 koji su puno popularniji upravo zbog toga. No, Intel 80C31 porodica ima široku primjenu koju koči nedostatak ovakvog mehanizma. Stoga nadzorno računalo sa ovim mehanizmom ima veliku vrijednost…

6.9. Naredba za preusmjeravanje naredbe na GSM uređaj

Ponekada je potrebno poslati neku naredbu GSM uređaju. To je daleko češće prilikom razvoja programske podrške. Upravo iz tog razloga postoji naredba «AT» koju nadzorno računalo preusmjerava na GSM uređaj, i naravno natrag ispisuje odgovor. To je potrebno kada isprobavamo nove «AT» naredbe, ili prilikom ispravljanja pogreški unutar programske podrške. Primjer:

```
at+cgmi
+CGMI: SIEMENS
0
```

No na ovaj način je moguće preusmjeriti samo naredbe koje započinju sa «AT». Kako GSM uređaj ima i nekolicinu drugih naredbi, iz tog razloga sa ugradio naredbu «MODEM» koja nadzorno računalo pretvara u obični spoj osobnog računala i GSM uređaja. Ovaj spoj je trajan sve do gašenja nadzornog računala ili resetiranjem procesora nadzornog računala. To izgleda ovako:

```
modem
Permanent redirection to GSM device is enabled
You will have to press reset button to return to normal mode
```

Slijedeće naredbe se direktno preusmjeravaju na GSM uređaj.

```
OK
atv1
OK
at+cmgi
ERROR
```

7. ZAKLJUČAK

Ideja upravljanja procesima putem usluge kratkih poruka nije nova, i u svijetu se koristi već neko vrijeme, no uglavnom, ulogu nadzornog računala ima osobno računalo. To je brz način automatizacije, ali je nezgrapno, veliko i skupo ako se osobno računalo upotrebljava samo u svrhu automatizacije.

Puno ekonomičnije i jeftinije je koristiti nadzorno računalo sa mikro-kontrolorom. Mikro-kontrolor iz porodice Intel 80C31 je dosta dobar izbor zbog svoje jednostavnosti, cijene i popularnosti. Isto tako Dallas DS80C390 je daleko naprednija inačica i ima strašno velike mogućnosti. No veliki nedostatak ove porodice mikro-kontrolora je nemogućnost jednostavnog punjenja memorije novom inačicom programa. Ovo je veliki nedostatak ako se nadzorno računalo koristi kao i razvojno okruženje za programsku podršku. Upravo iz tog razloga sam nadzorno računalo nadogradio podrškom za punjenje memorije putem asinkrone serijske sabirnice i čuvanje memorije baterijom kada nema napajanja. Ove dvije funkcije veoma malo poskupljuju nadzorno računalo jer se radi o samo 2 dodana integrirana kruga iz porodice 74HCT.

Činjenica da se čuva podatkovna i programska memorija nakon nestanka napona napajanja daje još veće mogućnosti zbog čuvanja baze podataka korisnika i zapisnika svih izvršenih akcija. Dakle, nadzorno računalo je predviđeno i za ove mogućnosti što se jednostavno implementira u programskoj podršci, no nije predmet ovog znanstvenog projekta i više ulazi u komercijalne svrhe.

Isto tako nadzorno računalo je opremljeno CAN sabirnicom koja omogućuje veliko proširenje mogućnosti nadzornog računala dodavanjem novih mikro-kontrolora u mrežu. Iako je unutar uređaja implementirana podrška za CAN sabirnicu, istu je potrebno napraviti i u programskoj podršci.

Programska podrška je testirana na 2 modela GSM uređaja i to na Siemens S10A i Siemens C35i. Na oba GSM uređaja nadzorno računalo i programska podrška funkcioniraju potpuno ispravno. Pretpostavljam da bi nadzorno računalo ispravno radilo i sa Siemens S35i, Siemens M35i, te i na ostalim GSM uređajima koji su GSM 07.05. i 07.07. kompatibilni sa ugrađenim modem procesorom.

U programskoj podršci je ostvareno sve što je zadano kao zadatak ovoga projekta, dakle ostvareno je automatizirano slanje obavijesti korisniku putem usluge kratkih poruka kada se promijeni stanje digitalnih ulaza. Isto tako putem usluge kratkih poruka udaljeni korisnik-operater može mijenjati stanja paralelnih digitalnih izlaza. No, indikaciju stanja paralelnih ulaza, te promjenu stanja izlaza može raditi i korisnik izravno na osobnom računalu odnosno terminalu.

Programskoj podršci dodana je i automatizacija putem poziva za brze obavijesti i akcije. Ovo nije bio predmet projekta, ali mi se učinilo da otvara još jedan segment mogućnosti daljinskog upravljanja, tako da sam isti ugradio u program.

Nadalje, programska podrška je razvijena isključivo u smjeru jednostavnog korištenja i pristupačnosti prema korisniku, tako da se spojem osobnog računala i nadzornog računala, osobno računalo ponaša kao terminal sa interakcijom prema korisniku. Naredbe su jednostavne i njima je moguće gotovo potpuno iskoristiti mogućnosti programske podrške. Postoji naredba za pomoć, isto kao naredbe za jednostavno slanje i čitanje kratkih poruka na terminalu. Ovo je interesantno, jer svi znaju kako je komplicirano napisati poruku na maloj tipkovnici GSM uređaja. Na ovaj način

korisnik ima udobnost tipkovnice osobnog računala na kojoj piše poruku i pošalje ju. Isto tako korisnik na preglednom ekranu može ispisati primljene poruke na GSM uređaj.

Također, programska podrška ima dodatne funkcije koje se koriste za razvoj programske podrške, među kojima je i razvijen pokretački program, koji je neophodan za rad emulacije memorija i prijenos nove inačice programa putem serijske asinkrone sabirnice sa osobnog računala.

Dodavanjem razvojnog ANSI C okruženja proizvođača Keil, microVision 2, razvoj programske podrške postaje jednostavan i veoma brz. Svaka izmjena programa jednostavno se testira slanjem na nadzorno računalo i izvršavanjem. Na ovaj način nisu potrebni nikakvi simulatori rada nadzornog računala, jer se program testira na stvarnom mikro-kontroloru.

Korišteni program za terminal je ugrađen u sam operativni sustav, kao i program za prijenos nove inačice programa na nadzorno računalo obavlja isti program za terminal. Program se zove HyperTerminal i Microsoft ga daje unutar svog operativnog sustava Windows. Na ovaj način izbjegnuta je izrada terminal programa za osobno računalo koje bi bilo izrađeno isključivo za nadzorno računalo.

Sve su ovo prednosti nadzornog računala koje je moguće koristiti i za druge razvojne projekte a ne samo za upravljanje procesima putem usluge kratkih poruka. Postojanjem paralelnih digitalnih ulaza-izlaza, moguće je nadzorno računalo nadograditi dodatnim sklopovljem i koristiti ga u razne svrhe.

Mogućnosti programske podrške i mogućnosti nadzornog računala su ograničene samo maštom i znanjem programera, te zahtjevima naručitelja projekta. Razvijena programska podrška za nadzorno računalo iskorištava tek mali dio njegovih mogućnosti…

8. POPIS KORIŠTENE LITERATURE

[1] Davor Petrinović i Mladen Vučić:
Osnove projektiranja računalnih sustava,
Zavodska skripta, Zagreb, 2001.

[2] Mladen Vučić:
Upotreba mikrokontrolera u ugrađenim računalnim sustavima,
Zavodska skripta, Zagreb, 1999.

[3] Mladen Vučić i Davorka Petrinović:
Projektiranje ugrađenih računalnih sustava, laboratorijske vježbe,
Zavodska skripta, Zagreb, 1999.

[4] **GSM Technical Specification, GSM07.05,**
European Telecommunications Standards Institute, www.etsi.org , 1996

[5] **AT Command Set (GSM 07.07, GSM 07.05, Siemens Specific Commands)**,
Siemens, www.siemens.com , 2000.

[6] **DS80C390 Dual CAN High-Speed Microproccesor,**
DALLAS Semiconductor, www.dalsemi.com , 2001.

[7] **Controller Area Network (CAN) introduction for press**,
Siemens, 1998.

[8] **CAN Specification version 2.0,**
Robert Bosch GmbH, 1991.

[9] FM**27C256, (32K x 8) High Performance CMOS EPROM databook**,
Fairchild Semiconductors, 2001.

[10] **MOS INTEGRATED CIRCUIT µPD431000A, data sheet**,
NEC Corporation, 1995.

[11] **DS1245Y/AB 1024K Nonvolatile SRAM**,
DALLAS Semiconductor, 1998.

[12] **74HC/HCT73 Dual JK flip-flop with reset; negative-edge trigger, data sheet**,
Philips Semiconductors, 1990.

[13] **74F153 Dual 4-Input Multiplexer**,
Fairchild Semiconductor, 2000.

[14] **+5V Powered, Multichannel RS-232 Drivers/Recivers, MAX232 data sheet**,
MAXIM, 2000.

[15] **CAN controller interface PCA82C250, data sheet**,
Philips Semiconductors, 1994.

[16] **DS80C390, High-Speed Microcontroller User Guide Supplement,**
DALLAS Semiconductor, 2000.

[17] **80C51 family programmer's guide and instruction set**,
Philips Semiconductors, 1997.

[18] **GSM, SMS messages and PDU format**,
http://www.dreamfabric.com/sms/

[19] **GSM i princip rada GSM mreže,**
http://www.crogsm.com/

9. SAŽETAK

9.1. Hrvatski jezik

Nadzorno računalo je bazirano na Intel 80C31 kompatibilnom mikro-kontroloru, DALLAS DS80C390, koji je ojačana inačica istog. Nadzorno računalo je opremljeno sa 2 serijske asinkrone sabirnice, CAN sabirnicom, 64KB programske i 64KB podatkovne memorije, funkcijom prenošenja izvršnog programa putem serijske sabirnice za vrijeme rada, 2 paralelna digitalna dvosmjerna porta za proširenje, nadzornim sklopom (eng. Watchdog) za siguran rad programa, funkcijom za čuvanje podataka unutar memorije nakon nestanka napona napajanja.

Programska podrška se sastoji iz 2 dijela. Prvi dio je pokretački program koji je dopuna funkciji prenošenja izvršnog programa putem serijske sabirnice, takozvani «Bootstrap loader», dok je drugi dio sam izvršni program. Izvršni program ima implementiranu podršku za slanje i primanje kratkih poruka, automatizirano i iz terminala, također ima podršku odnosno algoritam za dojavu hitnih procesa, te izvršava hitne naredbe. Izvršni program ima ugrađen tekst procesor za interakciju sa korisnikom, na veoma jednostavan i lagan za učenje način. Program je građen na funkcijama tako da je moguća lagana dopuna programa i prilagođavanje potrebama korisnika.

9.2. English language

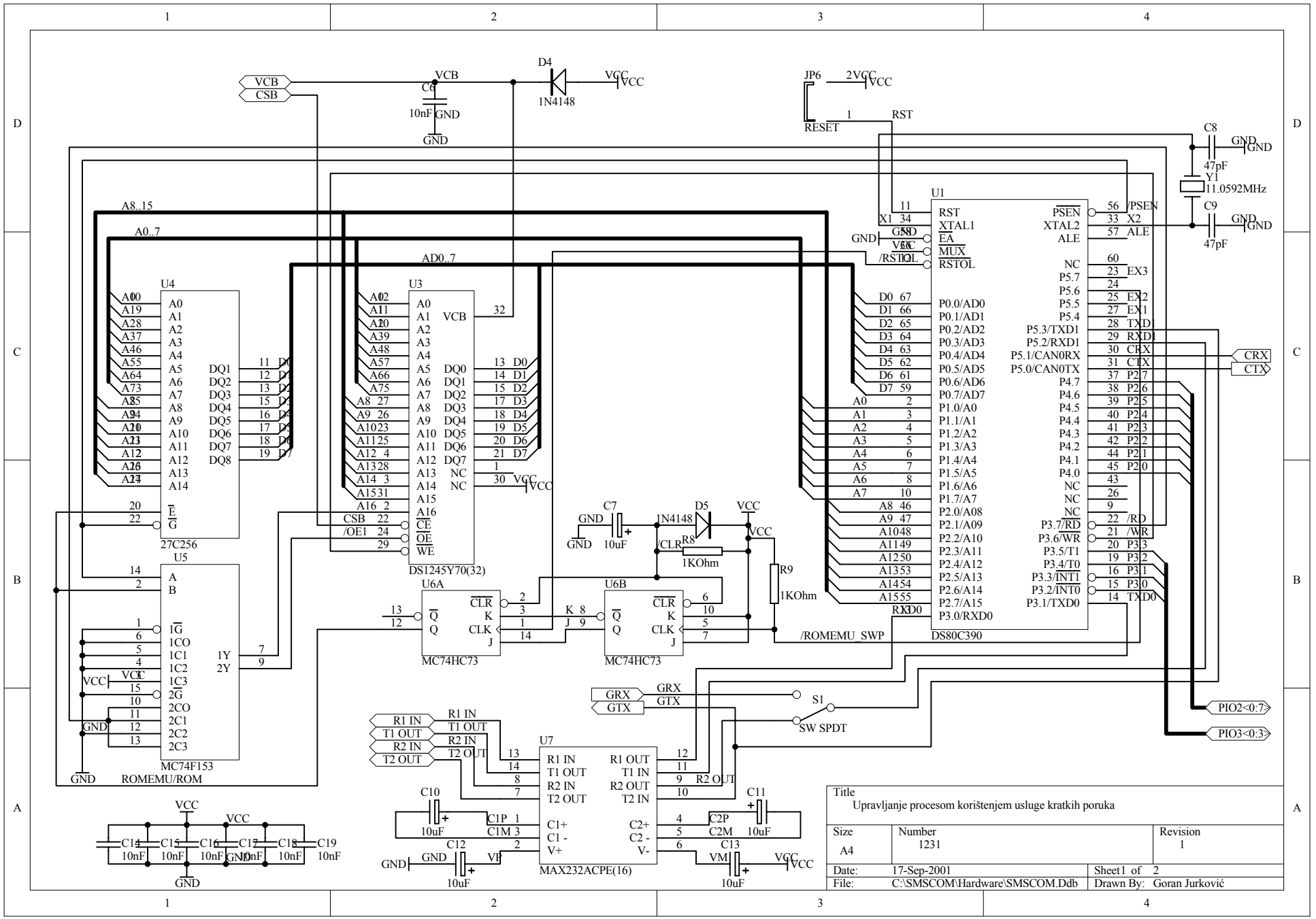
Server computer is based on Intel 80C31 compatible micro-controller, DALLAS DS80C390, witch is advanced version of the first one. Server computer is equipped with 2 serial bus, CAN bus, 64KB program and 64KB data memory, in circuit serial programming also called “Bootstrap loader”, 2 parallel digital bi-directional ports for expansion, watchdog timer, and saving data of both memory when there is no power supply.

Software support is based on 2 parts. First part is startup program, so called “Bootstrap loader” needed for in circuit serial programming, while the other part is main program. Software has implemented support for sending and receiving SMS, automatic and from terminal, also have support for urgent notification of changes in processes, very easy and user friendly command interpreter that is easy learnable. Main program is based on functions that can be easily upgraded to fulfill needs of user.

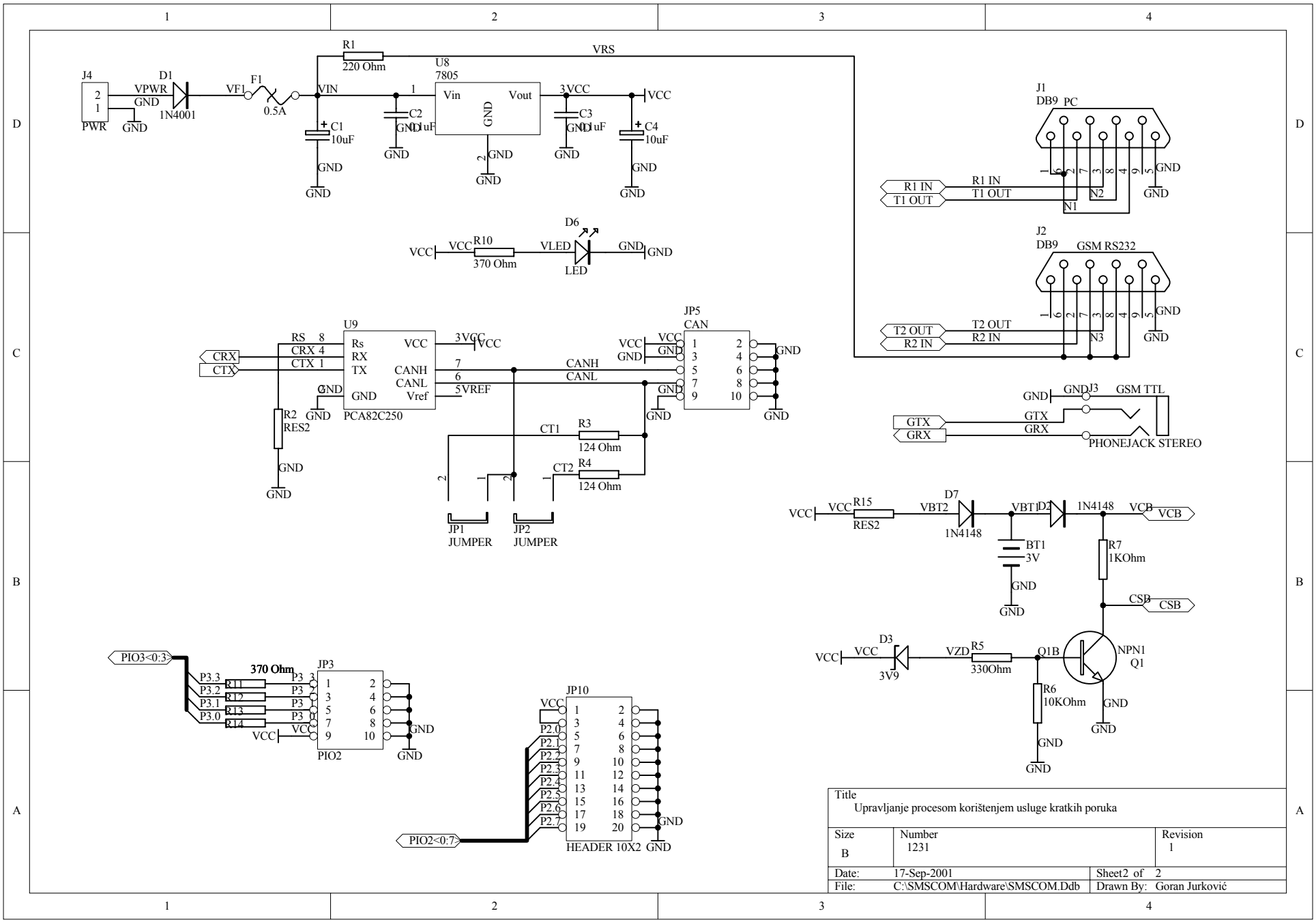
10. ŽIVOTOPIS

Rođen sam 27.09.1975. godine u lijepom malom gradu, Slavonskom Brodu, koji se nalazi na sjevernoj obali rijeke Save. U istom gradu sam odrastao, te sam se 1982. godine upisao u osnovnu školu «Ivan Goran Kovačić» u Slavonskom Brodu. Slijedeće godine, 1983., završavam prvi razred, sa pismenom pohvalom vijeća nastavnika za izuzetno zalaganje i postignut uspjeh u usvajanju nastavnog gradiva. 1984. godine završavam drugi razred također sa pismenom pohvalom vijeća nastavnika. 1985. godine završavam treći razred sa pismenom pohvalom vijeća nastavnika, te 1986. godine četvrti razred sa pismenom pohvalom vijeća nastavnika.
1988. godine nakon školskog i općinskog natjecanja, odlazim na regionalno natjecanje stvaralaštva mladih u Đakovu, gdje pobjeđujem i idem na republičko natjecanje. Dakle iste godine sam sudjelovao, te dobio diplomu za sudjelovanje na smotri tehničkog stvaralaštva mladih SR Hrvatske, koja se održala u Poreču, 19. do 22. svibnja 1988. godine. Disciplina u kojoj sam se natjecao, bila je elektronika. Za natjecanje, škola koju sam pohađao, dobila je računalo «ORAO» te monitor. Iste godine završavam šesti razred osnovne škole sa pismenom pohvalnicom za postignuti uspjeh i primjereno vladanje.
Slijedeće, 1989. godine, poslije školskog, općinskog i regionalnog natjecanja, dolazim ponovno na republičko natjecanje. Dakle, dobio sam diplomu za sudjelovanje na republičkoj smotri mladih tehničara, koja se održala u Sisku, 18. do 21. svibnja 1989. godine. Disciplina u kojoj sam se natjecao bila su računala, tj. programi za njih. Za moje natjecanje, škola koju sam pohađao, dobila je računalo «NOVA 64». Iste godine, završio sam tečaj za amaterskog radio operatera, te dobio ovlaštenje amaterskog radio operatora C i F razreda.
1990. godine, također prolazim školsko, općinsko, regionalno natjecanje te dolazim na republičko natjecanje. Na natjecanju dobivam diplomu za postignute rezultate u znanstveno-tehničkom stvaralaštvu mladih u elektronici. Natjecanje se održalo u Makarskoj. Zbog vrlo dobrog rezultata, idem na slijedeće natjecanje, tada posljednje, savezno natjecanje, te dobivam zlatnu medalju i diplomu za prikazane rezultate u radu s mladima na smotri znanstveno-tehničkog stvaralaštva mladih Jugoslavije. Natjecanje se također održalo u Makarskoj, u svibnju 1990. godine. Disciplina u kojoj sam se natjecao bila je elektronika. Iste godine, završavam osmi razred osnovne škole sa pismenom pohvalom za odličan uspjeh i primjereno vladanje.
Sve razrede osnovne škole završavam sa izvrsnim uspjehom.
1990. godine se upisujem u srednju tehničku školu u Slavonskom Brodu, smjer elektronika. Slijedeće godine, 1991. završavam prvi razred sa vrlo dobrim uspjehom.
1992. godine završavam drugi razred srednje tehničke škole sa vrlo dobrim uspjehom. Zbog domovinskog rata u Republici Hrvatskoj, bio sam prisiljen preći u Zagreb na daljnje školovanje. Tako da sam treći razred iste godine pohađao u srednjoj tehničkoj školi za elektroniku «Ruđer Bošković» u Zagrebu. Naredne godine, 1993., završio sam treći razred sa vrlo dobrim uspjehom.
I četvrti razred sam završio u istoj školi u Zagrebu. Dakle, 1994. godine, završio sam četvrti razred srednje tehničke škole sa vrlo dobrim uspjehom. Iste godine sam maturirao sa odličnim uspjehom te stekao stručnu spremu IV. Stupnja, struka elektrotehnika, obrazovni profil elektrotehničar, smjer elektronika.
Iste godine, u rujnu 1994., upisujem se na Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, smjer automatika. Fakultet završavam u rujnu 2001. godine.

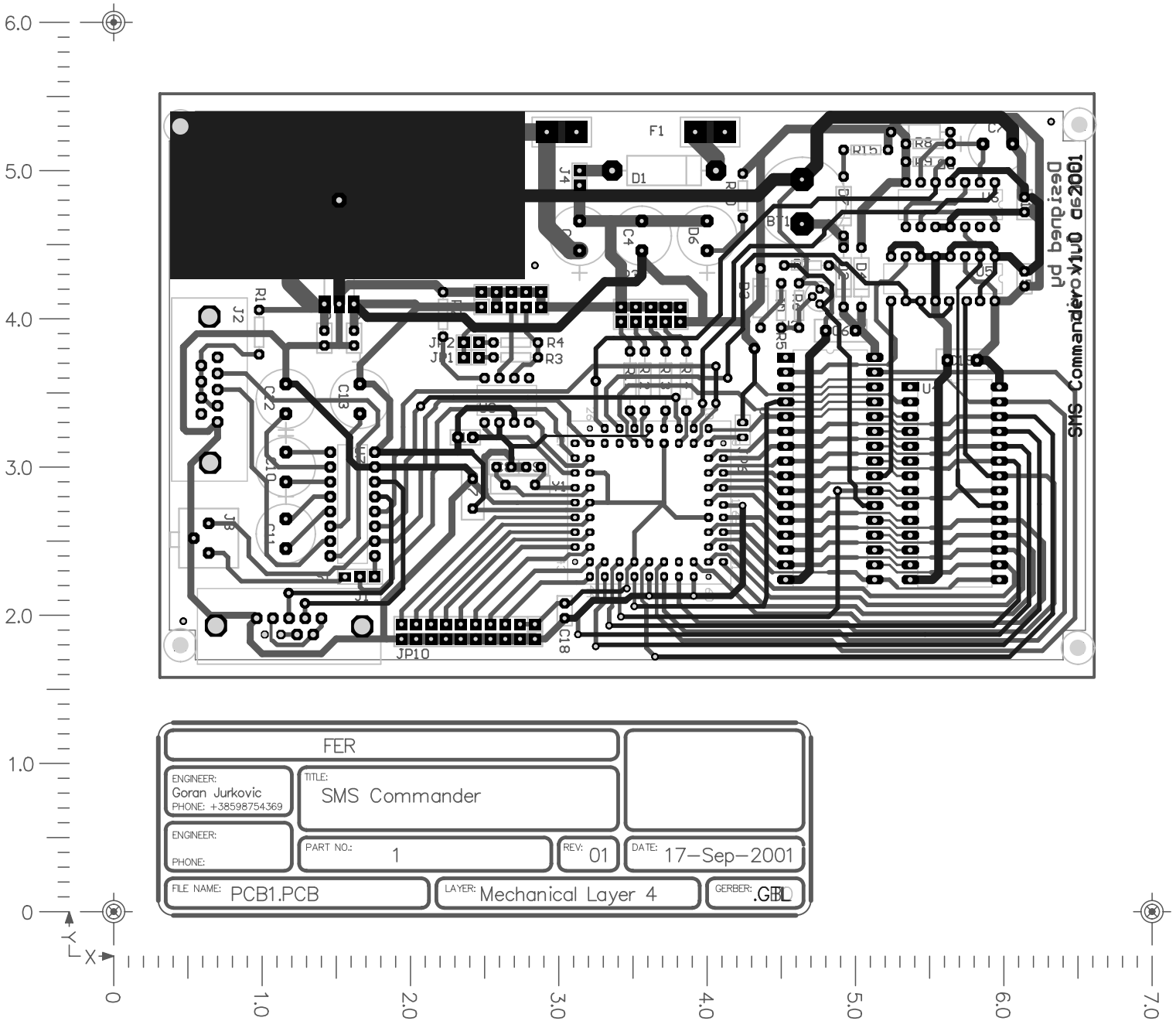
P.1. ELEKTRIČNE SHEME UREĐAJA

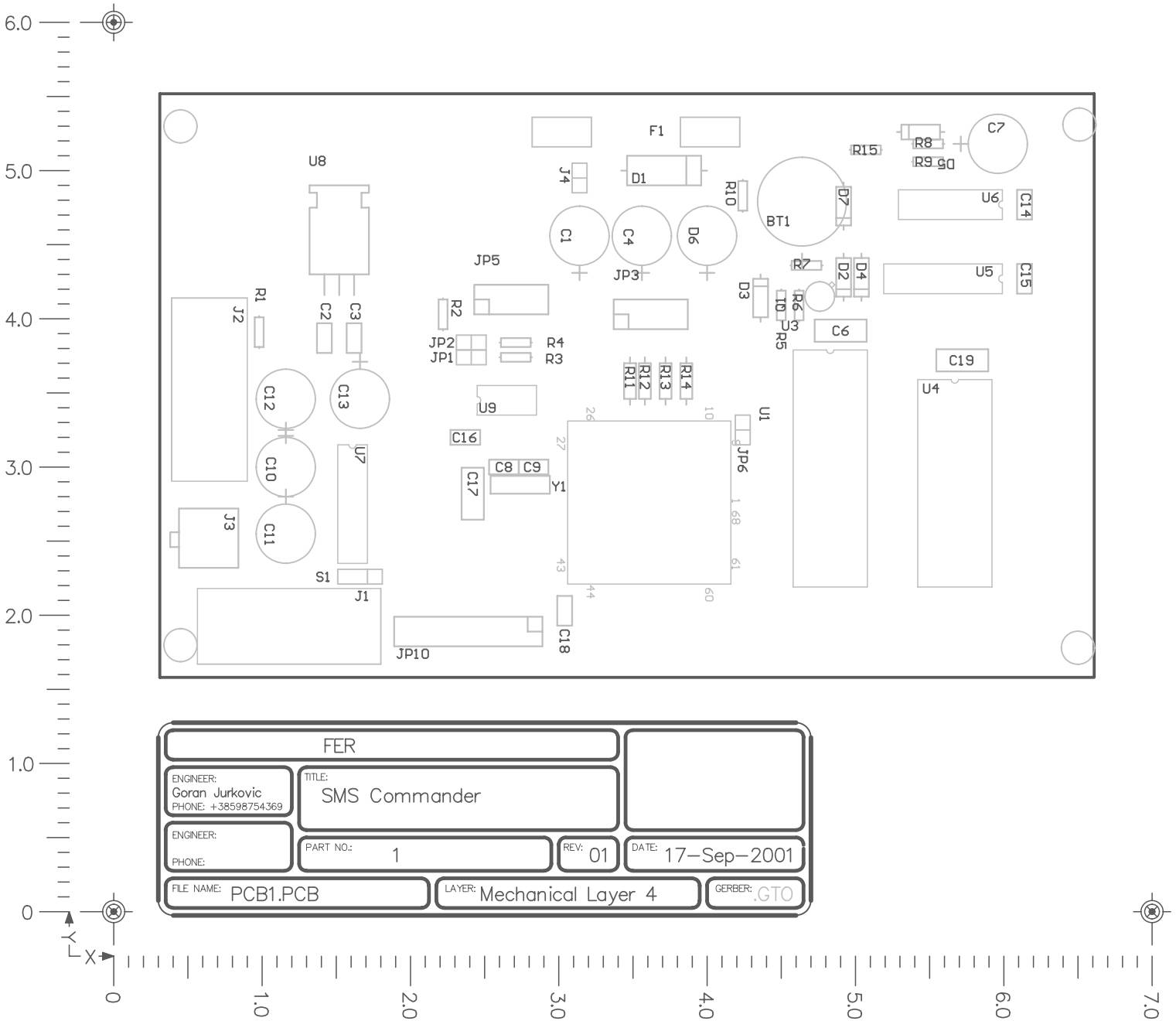


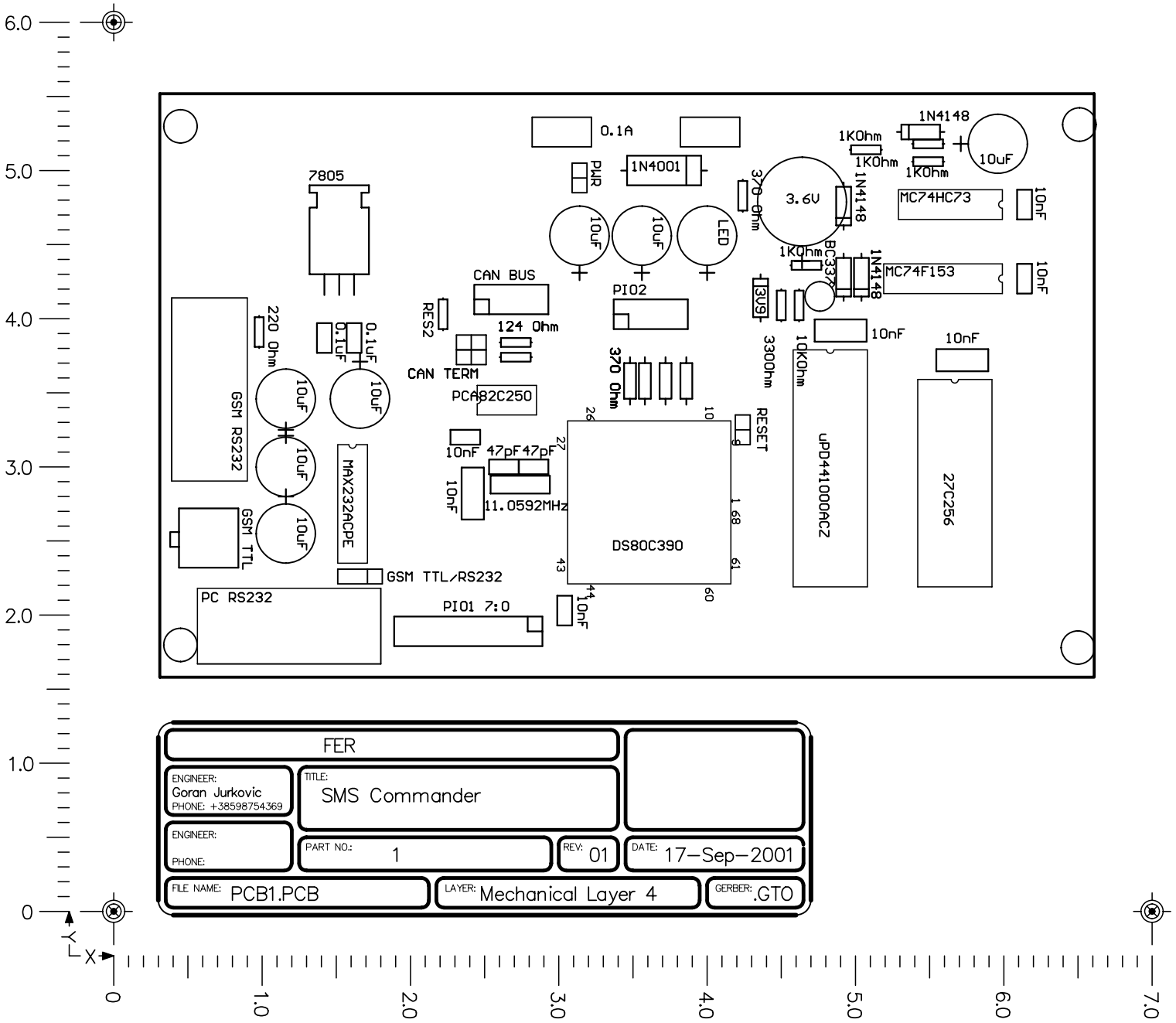
Title	Upravljanje procesom korištenjem usluge kratkih poruka		
Size A4	Number 1231		Revision 1
Date: 17-Sep-2001		Sheet 1 of 2	
File: C:\SMSCOM\Hardware\SMSCOM.Ddb		Drawn By: Goran Jurković	

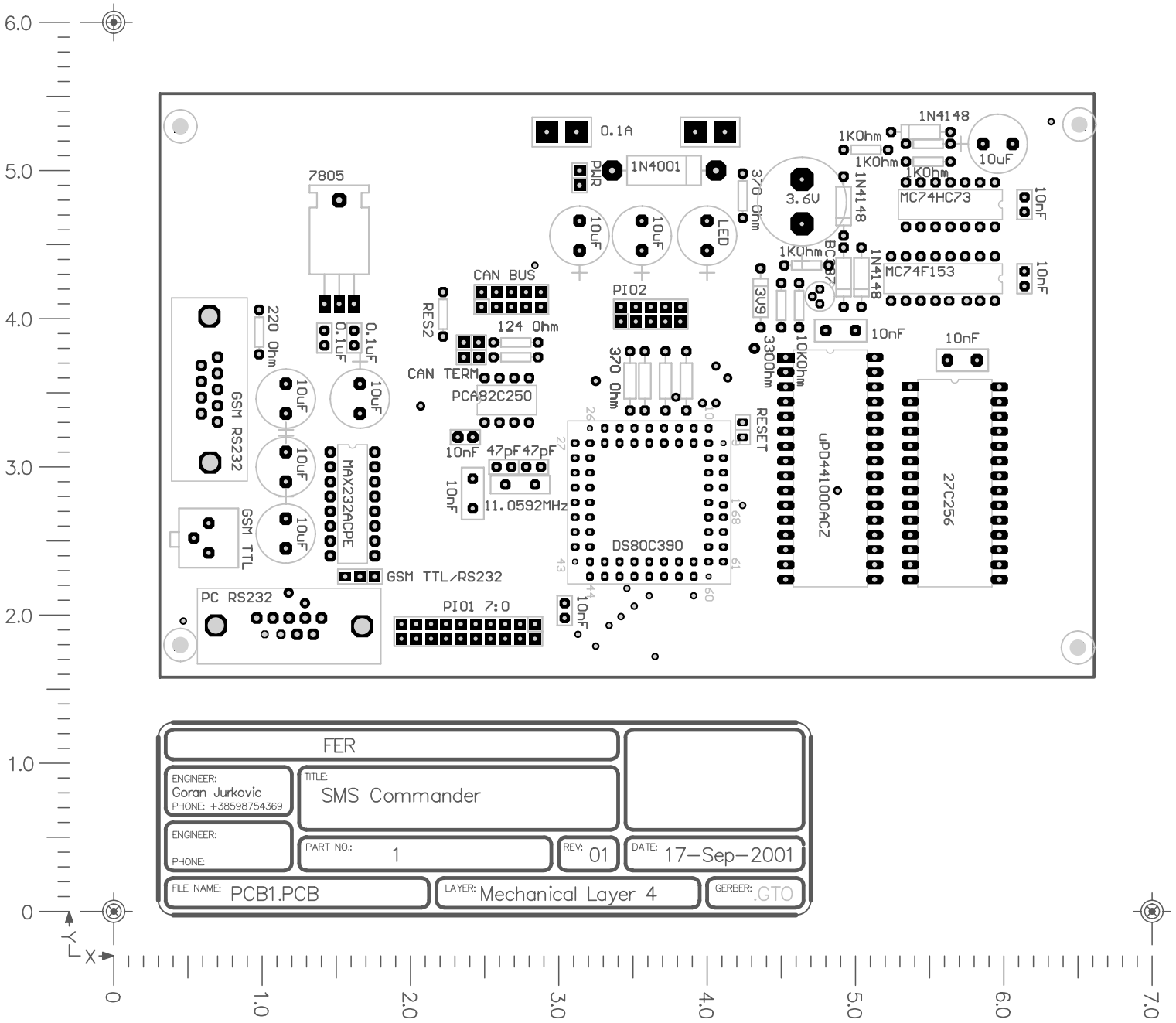

R1
220 Ohm
VRS
U8
7805
J4
D1
VPWR
GND
1N4001
PWR
F1
0.5A
VF1
VIN
Vin
Vout
3VCC
VCC
C1
10uF
C2
0.1uF
C3
0.1uF
C4
10uF
J1
DB9 PC
R1 IN
T1 OUT
J2
DB9
GSM RS232
T2 OUT
R2 IN
D6
R10
370 Ohm
VLED
LED
U9
PCA82C250
RS
CRX
CTX
RX
TX
CANH
CANL
VREF
JP5
CAN
R2
RES2
R3
124 Ohm
R4
124 Ohm
CT1
CT2
JP1
JUMPER
JP2
JUMPER
J3
GSM TTL
GTX
GRX
PHONEJACK STEREO
D7
1N4148
R15
RES2
VBT2
VBT1
BT1
3V
R7
1KOhm
CSB
D3
3V9
R5
330Ohm
Q1B
NPN1
Q1
R6
10KOhm
PIO3<0:3>
370 Ohm
JP3
PIO2
JP10
HEADER 10X2
PIO2<0:7>
Title
Upravljanje procesom korištenjem usluge kratkih poruka
Size
B
Number
1231
Revision
1
Date:
17-Sep-2001
Sheet2 of 2
File:
C:\SMSCOM\Hardware\SMSCOM.Ddb
Drawn By: Goran Jurković

P.2. TEHNIČKA DOKUMENTACIJA ZA IZRADU TISKANIH PLOČICA

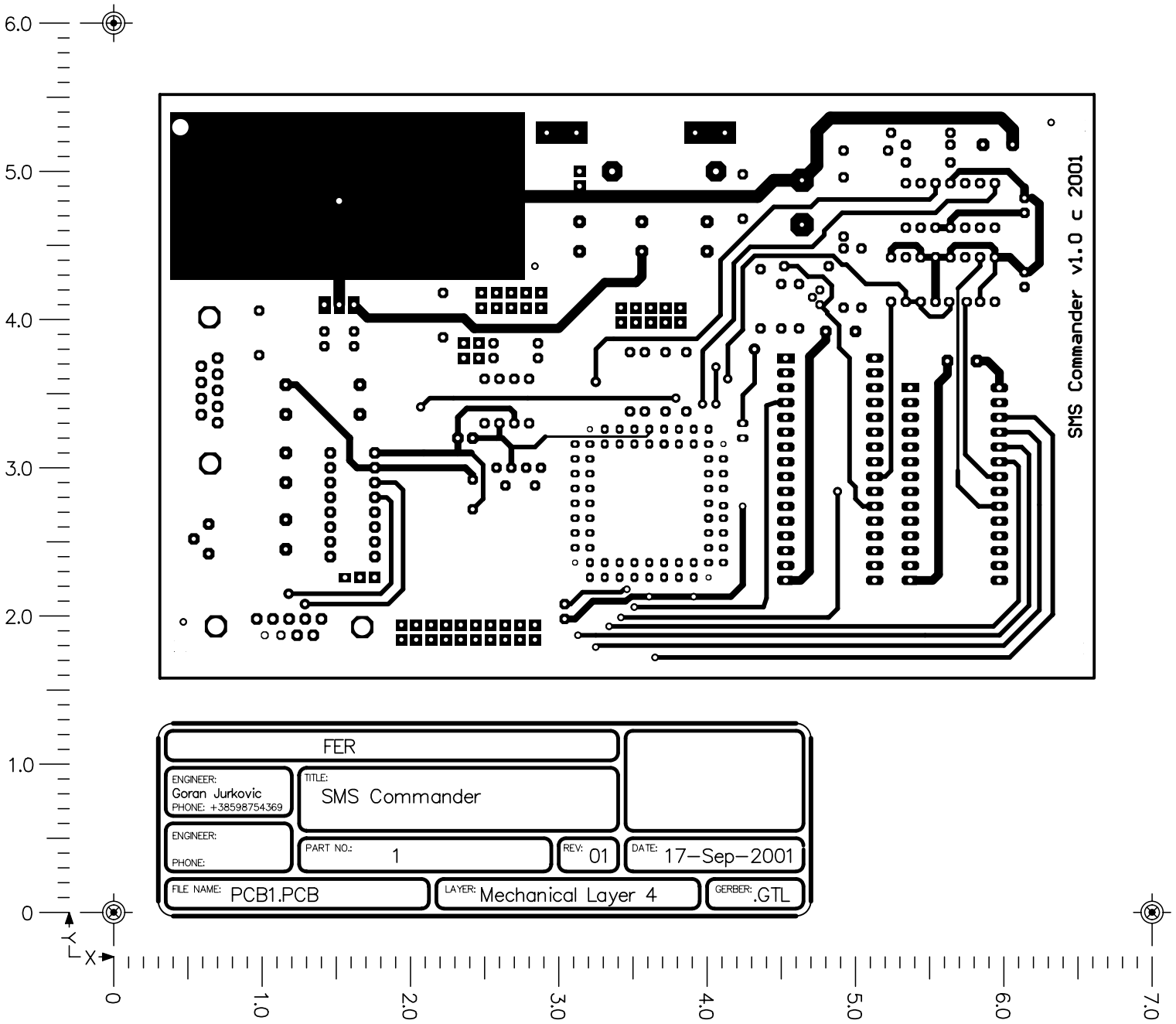

6.0
5.0
4.0
3.0
2.0
1.0
0
Y
X
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
SMS Commander v1.0 design 2001
Designed by
FER
ENGINEER:
Goran Jurkovic
PHONE: +38598754369
TITLE:
SMS Commander
ENGINEER:
PHONE:
PART NO.: 1
REV: 01
DATE: 17-Sep-2001
FILE NAME: PCB1.PCB
LAYER: Mechanical Layer 4
GERBER: .GBL

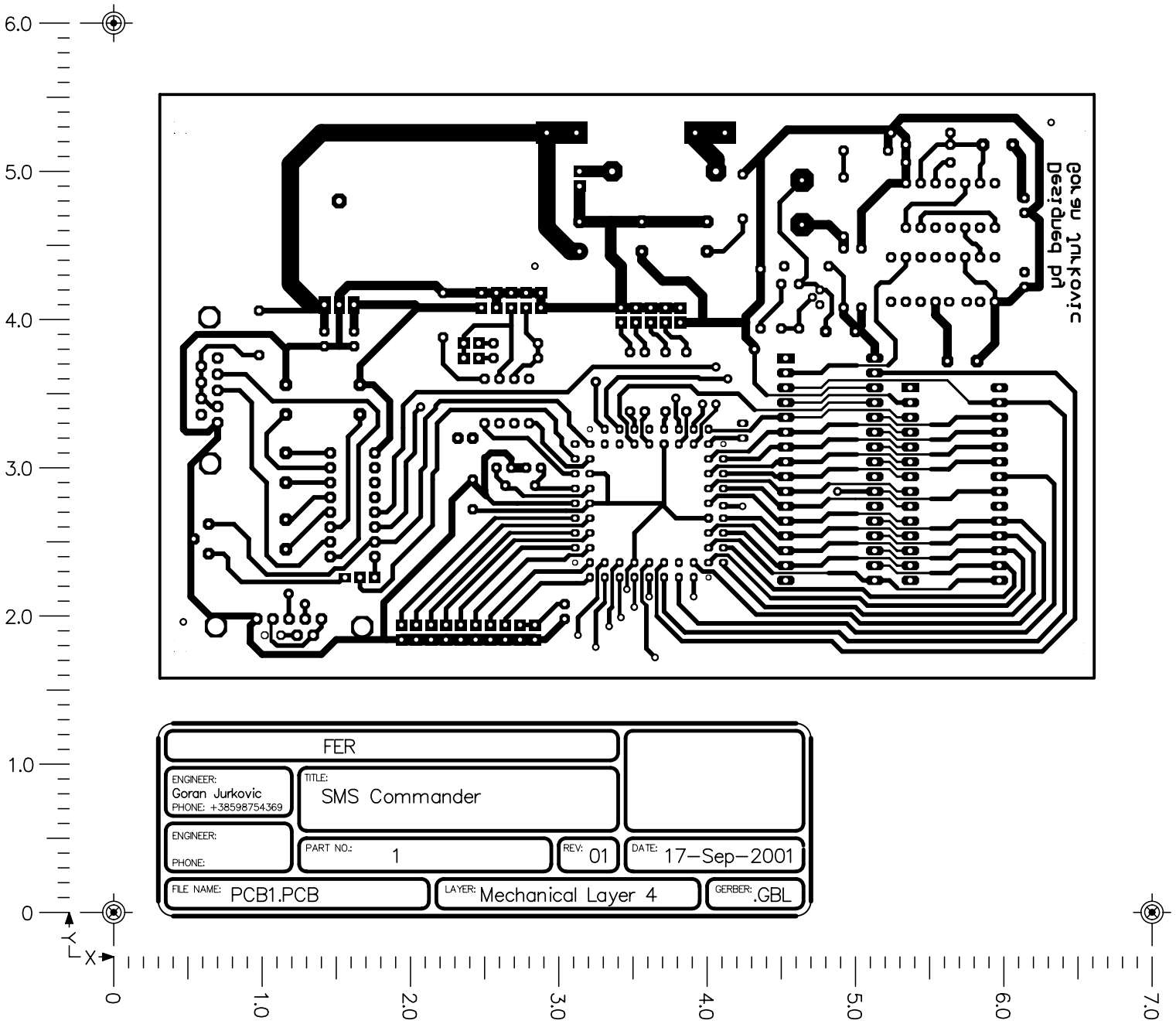

6.0
5.0
4.0
3.0
2.0
1.0
0
Y
X
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
U8
F1
J4
D1
R10
C7
R15
R8
R9
D5
U6
C14
BT1
D7
C1
C4
D6
JP5
JP3
R7
D2
D4
U5
C15
D3
R6
R5
U3
C6
R1
J2
C2
C3
R2
JP2
JP1
R4
R3
C19
U4
C12
C13
R11
R12
R13
R14
U9
U1
C16
JP6
C10
U7
C8
C9
Y1
C17
J3
C11
S1
J1
C18
JP10
FER
ENGINEER:
Goran Jurkovic
PHONE: +38598754369
TITLE:
SMS Commander
ENGINEER:
PHONE:
PART NO.: 1
REV: 01
DATE: 17-Sep-2001
FILE NAME: PCB1.PCB
LAYER: Mechanical Layer 4
GERBER: .GTO

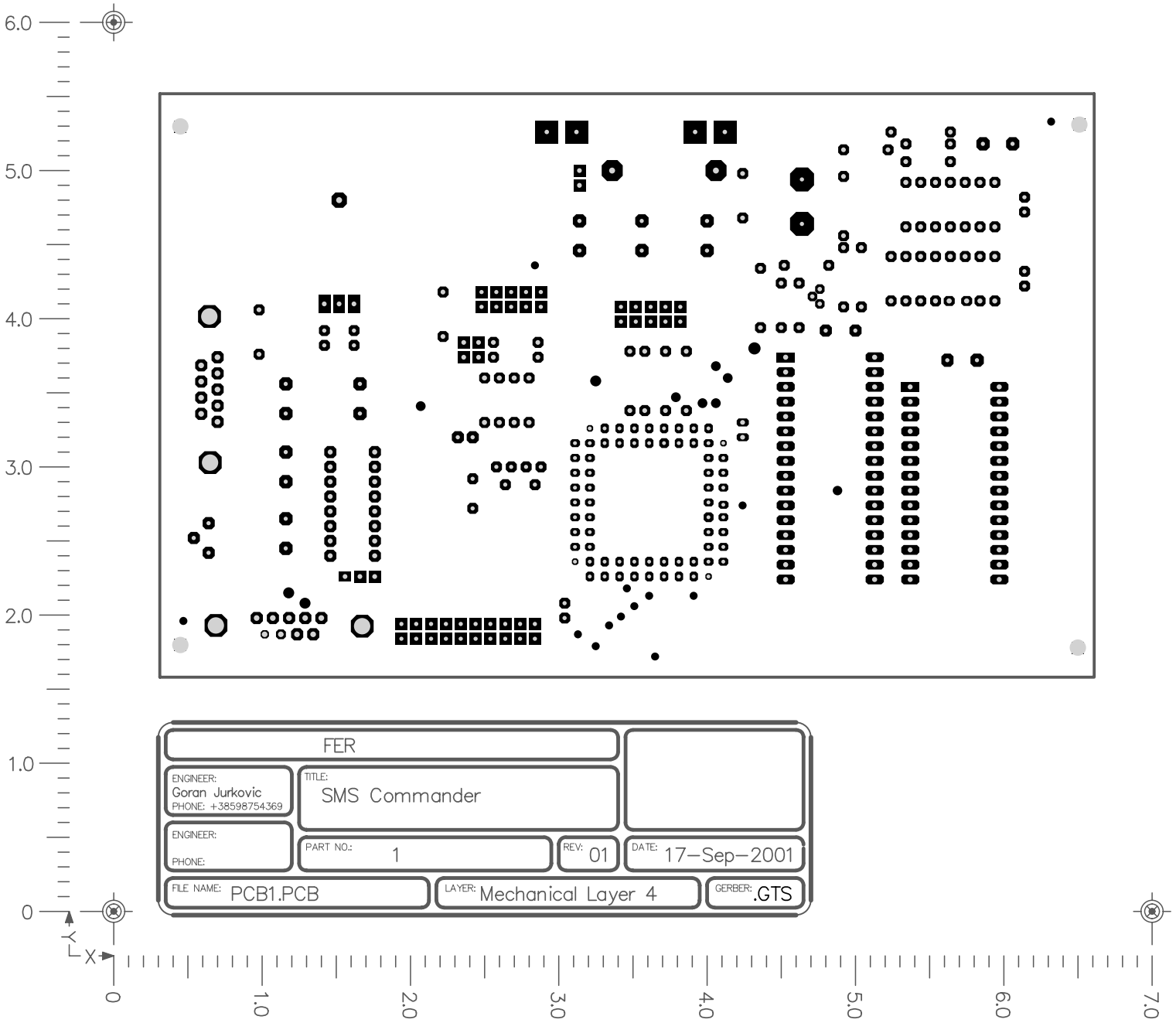

0.1A
1N4148
1KOhm
1KOhm
1KOhm
10uF
PWR
1N4001
7805
3.6V
370 Ohm
1N4148
MC74HC73
10nF
10uF
10uF
LED
1KOhm
BC337
1N4148
MC74F153
10nF
CAN BUS
PIO2
3V9
RES2
124 Ohm
10nF
10nF
220 Ohm
0.1uF
0.1uF
CAN TERM
370 Ohm
3300hm
10kOhm
GSM RS232
10uF
10uF
PCA82C250
10nF
47pF
47pF
RESET
10nF
11.0592MHz
MAX232ACPE
10uF
GSM TTL
DS80C390
uPD441000ACZ
27C256
GSM TTL/RS232
PC RS232
PIO1 7:0
10nF
FER
ENGINEER:
Goran Jurkovic
PHONE: +38598754369
TITLE:
SMS Commander
ENGINEER:
PHONE:
PART NO.: 1
REV: 01
DATE: 17-Sep-2001
FILE NAME: PCB1.PCB
LAYER: Mechanical Layer 4
GERBER: .GTO
6.0
5.0
4.0
3.0
2.0
1.0
0
Y
X
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

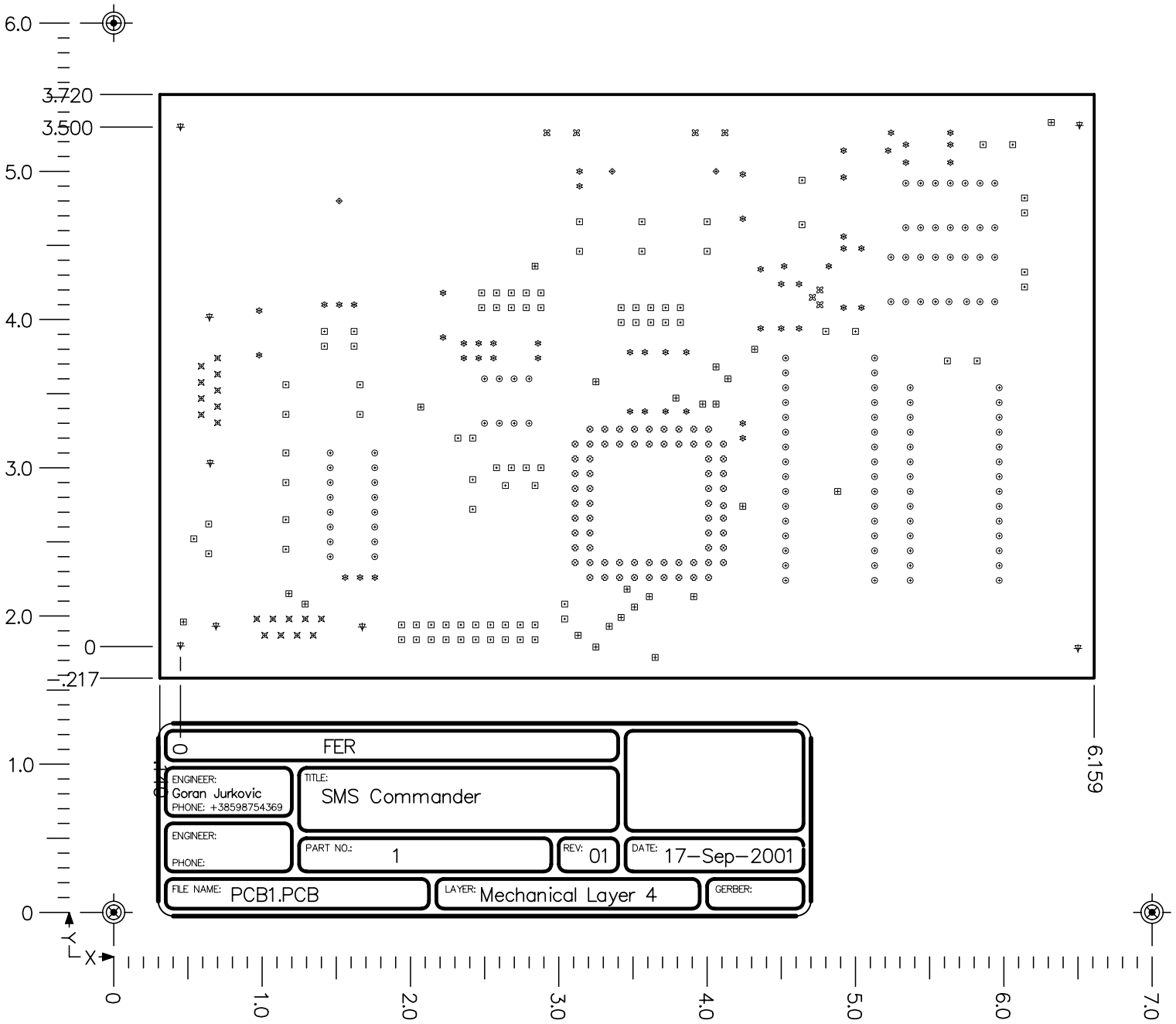


FER			
ENGINEER: Goran Jurkovic PHONE: +38598754369	TITLE: SMS Commander		
ENGINEER: PHONE:	PART NO.: 1	REV: 01	DATE: 17-Sep-2001
FILE NAME: PCB1.PCB	LAYER: Mechanical Layer 4	GERBER: .GTO	


6.0
5.0
4.0
3.0
2.0
1.0
0
SMS Commander v1.0 c 2001
FER
ENGINEER:
Goran Jurkovic
PHONE: +38598754369
TITLE:
SMS Commander
ENGINEER:
PHONE:
PART NO.: 1
REV: 01
DATE: 17-Sep-2001
FILE NAME: PCB1.PCB
LAYER: Mechanical Layer 4
GERBER: .GTL
Y
X
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0


6.0
5.0
4.0
3.0
2.0
1.0
0
Y
X
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
Designed by Goran Jurkovic
FER
ENGINEER:
Goran Jurkovic
PHONE: +38598754369
TITLE:
SMS Commander
ENGINEER:
PHONE:
PART NO.: 1
REV: 01
DATE: 17-Sep-2001
FILE NAME: PCB1.PCB
LAYER: Mechanical Layer 4
GERBER: .GBL


6.0
5.0
4.0
3.0
2.0
1.0
0
Y
X
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
FER
ENGINEER:
Goran Jurkovic
PHONE: +38598754369
TITLE:
SMS Commander
ENGINEER:
PHONE:
PART NO.: 1
REV: 01
DATE: 17–Sep–2001
FILE NAME: PCB1.PCB
LAYER: Mechanical Layer 4
GERBER: .GTS


6.0
3.720
3.500
5.0
4.0
3.0
2.0
0
-.217
1.0
0
6.159
0
FER
ENGINEER:
Goran Jurkovic
PHONE: +38598754369
TITLE:
SMS Commander
ENGINEER:
PHONE:
PART NO.: 1
REV: 01
DATE: 17-Sep-2001
FILE NAME: PCB1.PCB
LAYER: Mechanical Layer 4
GERBER:
Y
X
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

P.3. ISPIS PROGRAMSKE PODRŠKE, POKRETAČKI PROGRAM

```
////////////////////////////////////////////////////////////////////////
/*********************************************************************/
/**** Bootstrap loader **** v1.1 **** dizajniran za SMS Commander ******/
/*********************************************************************/
/******** For microcontroller DALLAS Semiconductors DS80C390 ***********/
/*********************************************************************/
////////////////////////////////////////////////////////////////////////

#include <reg390.h>
#include <stdio.h>
#include <intrins.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

char vrijeme;
char sekundi;

////////////////////////////////////////////////////////////////////////
// Deklaracija digitalnih ulazno izlaznih portova
////////////////////////////////////////////////////////////////////////
sbit PIO2_0=P3^2;
sbit PIO2_1=P3^3;
sbit PIO2_2=P3^4;
sbit PIO2_3=P3^5;
sbit PIO1_0=P4^1;
sbit PIO1_1=P4^0;
sbit PIO1_2=P4^2;
sbit PIO1_3=P4^3;
sbit PIO1_4=P4^4;
sbit PIO1_5=P4^5;
sbit PIO1_6=P4^6;
sbit PIO1_7=P4^7;


////////////////////////////////////////////////////////////////////////
// Funkcija za provjeru ispravnosti memorije i brisanje memorije
////////////////////////////////////////////////////////////////////////
void mem_test(void) {
  volatile unsigned char xdata *pokazivac;
  unsigned char data varijabla;
  unsigned int data i;
  bit uredu;

  TA = 0xAA;
  TA = 0x55;
  EWT = 0;                // Onemogućavanje Watch Dog-a


  printf ("\nTesting memory...\n");
  uredu=1;
  printf ("\nFilling memory with AAh up... ");
  for (i=0;65535>i;i++) {
      pokazivac=i;
      *pokazivac=0xAA;
      varijabla=*pokazivac;
            if (varijabla != 0xAA) printf (" %u",i);
            if (varijabla != 0xAA) uredu=0;
  }
  printf ("\nFilling memory with AAh down... ");
  for (i=65535;i>0;i--) {
```

```
      pokazivac=i;
      *pokazivac=0xAA;
      varijabla=*pokazivac;
      if (varijabla != 0xAA) printf (" %u",i);
      if (varijabla != 0xAA) uredu=0;
   }
   printf ("\nFilling memory with 55h up... ");
   for (i=0;65535>i;i++) {
      pokazivac=i;
      *pokazivac=0x55;
      varijabla=*pokazivac;
      if (varijabla != 0x55) printf (" %u",i);
      if (varijabla != 0x55) uredu=0;
   }
   printf ("\nFilling memory with 55h down... ");
   for (i=65535;i>0;i--) {
      pokazivac=i;
      *pokazivac=0x55;
      varijabla=*pokazivac;
      if (varijabla != 0x55) printf (" %u",i);
      if (varijabla != 0x55) uredu=0;
   }
   printf ("\nFilling memory with pattern... ");
   for (i=0;65535>i;i++) {
      pokazivac=i;
      *pokazivac=i%255;
   }
   for (i=0;65535>i;i++) {
      pokazivac=i;
      varijabla=*pokazivac;
      if (varijabla != i%255) printf (" %u",i);
      if (varijabla != i%255) uredu=0;
   }
   printf ("\nFilling memory with 00h... ");
   for (i=0;65535>i;i++) {
      pokazivac=i;
      *pokazivac=0x00;
   }
   for (i=0;65535>i;i++) {
      pokazivac=i;
      varijabla=*pokazivac;
      if (varijabla != 0x00) printf (" %u",i);
      if (varijabla != 0x00) uredu=0;
   }
   if (uredu == 1) printf("\n\nMemory is OK!\n");
   else printf ("\n\nERROR in memory!\n");

   TA = 0xAA;
   TA = 0x55;
   EWT = 1;                // Omogućavanje Watch Dog-a
}

//////////////////////////////////////////////////////////////////////
// Funkcija za prelazak u spori način rada zbog štednje energije
// 1 - spori način rada         0 - normalan način rada
// clock / 1024                 clock / 1
//////////////////////////////////////////////////////////////////////
void power_save(bit mod_rada) {

   if (mod_rada == 1) {
      PMR |= 0xA0;    // Prelazak u clock / 4
```

```
      PMR |= 0xE0;    // Prelazak u clock / 1024
   } else {
      PMR |= 0xA0;    // Prelazak u clock / 4
   }
}

//////////////////////////////////////////////////////////////////////
// Funkcija za inicijalizaciju Watch Dog Timera
//////////////////////////////////////////////////////////////////////
void wd_i(void) {
   CKCON |= 0xC0;       // Postavljanje dijeljenja WD-a na 2^26

   TA = 0xAA;           // Toglle TA bits. Ovo zahtijevaju neki registri
   TA = 0x55;           // kao zaštitu od slučajnog mijenjanja

   RWT = 1;             // Reset WD timer-a

   TA = 0xAA;
   TA = 0x55;

   EWT = 1;             // Omogućavanje Watch Dog-a
}

//////////////////////////////////////////////////////////////////////
// Funkcija za resetiranje Watch Dog Timera
//////////////////////////////////////////////////////////////////////
void wd_r(void) {
   TA = 0xAA;
   TA = 0x55;
   RWT = 1;             // Reset WD timer-a
}

//////////////////////////////////////////////////////////////////////
// Funkcija za inicijalizaciju serijskih portova
//////////////////////////////////////////////////////////////////////
void ser_init (void) {
   EA=0;                      // Onemogući prekide
   EPF1=0;                    // Onemogući Power fail prekid

   // Komunikacija preko RS232 postavljena na 19200bps 8-N-1
   SCON0 = 0x52;         // Postavljanje moda prvog serijskog porta
   SCON1 = 0x52;         // Postavljanje moda drugog serijskog porta
   PCON |= 0x80;
   SMOD_1 = 1;

   TMOD = 0x20;          // 2 za Timer1 (RS232)
   TH1 = 0xFD;           // Postavljanje reload vrijednosti za 9600
   TL1 = 0xFD;
   IP  = 0;              // prioritet
   TR1 = 1;              // Startaj Timer1

   ET1 = 0;              // Onemogući prekidnu rutinu za Timer1
   EA  = 1;              // uključi globalne interapte
   P5CNT |=0x20;         // Postavljanje drugog serijskog porta
                         // na port P5.2 i P5.3
   P5CNT &=0xF8;         // Konfiguracija od 64KB
   P4CNT &=0x80;         // Konfiguracija od 64KB
}
```

```
//////////////////////////////////////////////////////////////////////
// Funkcija za prelazak na drugi način rada memorije
//////////////////////////////////////////////////////////////////////
void romemu_swap(void) {
  unsigned int data i;

  P5 &= 0xBF;          // Postavljanje /ROMEMU_SWP linije u 0
  for(i=0;i<100;i++) _nop_();
  _nop_();       // Pričekaj neko vrijeme
  P5 |= 0x40;          // Postavljanje /ROMEMU_SWP linije u 1
  PF1=1;               // Dojava programu da se izvrši od početka
}


//////////////////////////////////////////////////////////////////////
void reset_funk (void) interrupt 0 {

}
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
// Glavni program
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
main () {

  volatile unsigned char xdata *pokazivac;
  unsigned char data varijabla;

  ser_init();                // Inicijalizacija serije
  wd_i();                    // Uključivanje Watch Dog-a

  if (WTRF==1 && PF1==0) {   // Izvršava se nakon WDT reseta procesora
     WTRF = 0;
     wd_r();                 // Resetiraj WDT
     pokazivac=0;            // Postavi pokazivac na početnu lokaciju RAM-a
     varijabla=*pokazivac;   // Uzmi podatak iz RAM-a sa te lokacije
     if (varijabla == 0) {   // Ako je ta lokacija prazna, Program ne
                             // postoji i ponovno ćemo ući u petlju za
                             // prijenos programa preko serije
           printf("\n Memory is empty... \n");
           printf(" Please, flash memory...\n");
     } else {                      // Ako nešto ima u memoriji
           printf("\n Loading programme...\n");
           romemu_swap();          // Naredbu za prebacivanje memorija nakon
                                   // WDT reseta procesora
           while (1);              // Čekaj da WDT resetira procesor
     }
  } else {                         // Izvršava se nakon pravog reseta procesora

     PF1=0;          // Dojava programu da se ne izvrši od početka
     printf ("\n\n***********************************");
     printf("***********************************\n");
     printf("********** Bootstrap loader **** v1.1");
     printf(" **** by Goran Jurkovich **********\n");
     printf("***********************************");
     printf("***********************************\n");

     printf("\n If you want to flash memory, send programme");
     printf(" within next few seconds...\n");
```

```
      printf(" If you want to test and erase memory,");
      printf(" press 't' now...\n\n Waiting...\n");
   }

   pokazivac=0;                        // Postavi početnu lokaciju u RAM-u
   while (1) {                         // Beskonačna petlja
      varijabla=_getkey();             // Uzmi karakter sa serije0
      *pokazivac=varijabla;            // Spremi ga na trenutnu lokaciju u RAM
      pokazivac++;                     // Postavi pokazivac na slijedeću lokaciju
      wd_r();                          // Resetiraj Watch Dog Timer
      if (pokazivac==1)                // Ako je to prvi poslani karakter
            if (varijabla =='t')       // I ako je on 't'
                  mem_test();          // Tada napravi test memorije
   }                                   // Petlja se nikada ne završava
                                       // Kada prestanu dolaziti podatci za upis
                                       // u memoriju, nakon isteka vremena WDT-a
                                       // Procesor će se resetirati
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```

P.4. ISPIS PROGRAMSKE PODRŠKE, IZVRŠNI PROGRAM

```
////////////////////////////////////////////////////////////////////////
//*********************************************************************//
//**** SMS Commander **** v0.5x **** by Goran Jurkovich **************//
//*********************************************************************//
////////////////////////////////////////////////////////////////////////
//////// For microcontroller DALLAS Semiconductors DS80C390 ////////////
////////////////////////////////////////////////////////////////////////

#include <reg390.h>
#include <stdio.h>
#include <intrins.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

#define XON  0x11
#define XOFF 0x13
#define version "0.50"

static unsigned char pdata ser0[62];
static unsigned char data com0[10];
static unsigned char pdata ser1[162];
static unsigned char xdata MSN[16];
static unsigned char xdata MSNO[16];
static unsigned char data ser0_broj,ser1_broj,com_broj,time;
static unsigned int data pio,pio_old;
static bit data com0_set,echo,gsm,modem;

// Deklaracija digitalnih ulazno izlaznih portova
sbit PIO2_0=P3^2;
sbit PIO2_1=P3^3;
sbit PIO2_2=P3^4;
sbit PIO2_3=P3^5;
sbit PIO1_0=P4^1;
sbit PIO1_1=P4^0;
sbit PIO1_2=P4^2;
sbit PIO1_3=P4^3;
sbit PIO1_4=P4^4;
sbit PIO1_5=P4^5;
sbit PIO1_6=P4^6;
sbit PIO1_7=P4^7;

////////////////////////////////////////////////////////////////////////
// Funkcija za slanje SMS poruke:  broj u formatu +38598xxxxxxx
//                                 poruka se prenosi u 'ser1'
////////////////////////////////////////////////////////////////////////
int send_sms (bit rc);

////////////////////////////////////////////////////////////////////////
// Funkcija za čitanje SMS poruke: broj u formatu +38598xxxxxxx
//                                       poruka se vraća u 'ser1'
//                                       broj se vraća u 'MSN'
//          read_sms (TYPE,DEL)          TYPE: 0 - Read new
//                                             1 - Read inbox
//                                             4 - Read all
//                                       DEL:  1 - Delete message
////////////////////////////////////////////////////////////////////////
int read_sms (char TYPE, char DEL);
```

```
//////////////////////////////////////////////////////////////////////
void reset_funk (void) interrupt 0 {

}

//////////////////////////////////////////////////////////////////////
// Funkcija za provjeru ispravnosti memorije i brisanje memorije
//////////////////////////////////////////////////////////////////////
void mem_test(void) {
  volatile unsigned char xdata *pokazivac;
  unsigned char data varijabla;
  unsigned int data i;
  bit uredu=1;

       TA = 0xAA;
       TA = 0x55;
       EWT = 0;                // Onemogućavanje Watch Dog-a

  printf ("\nTesting memory...\n");

  printf ("\nFilling memory with AAh... ");
  for (i=0;60000>i;i++) {
     pokazivac=i;
     *pokazivac=0xAA;
     varijabla=*pokazivac;
     if (varijabla != 0xAA) printf (" %u",i);
     if (varijabla != 0xAA) uredu=0;
  }
  printf ("\nFilling memory with pattern... ");
  for (i=0;60000>i;i++) {
     pokazivac=i;
               *pokazivac=i%255;
  }
  for (i=0;60000>i;i++) {
     pokazivac=i;
     varijabla=*pokazivac;
               if (varijabla != i%255) printf (" %u",i);
               if (varijabla != i%255) uredu=0;
  }
  printf ("\nFilling memory with 00h... ");
  for (i=0;60000>i;i++) {
     pokazivac=i;
               *pokazivac=0x00;
  }
  for (i=0;60000>i;i++) {
     pokazivac=i;
     varijabla=*pokazivac;
               if (varijabla != 0x00) printf (" %u",i);
               if (varijabla != 0x00) uredu=0;
  }
  if (uredu == 1) {
               printf("\n\nMemory is OK!\n");
  } else {
               printf ("\n\nERROR in memory!\n");
  }

       TA = 0xAA;
       TA = 0x55;
       EWT = 1;                // Omogućavanje Watch Dog-a

}
```

```
//////////////////////////////////////////////////////////////////////
// Funkcija za prelazak u spori način rada zbog štednje energije
// 1 - spori način rada         0 - normalan način rada
// clock / 1024                 clock / 4
//////////////////////////////////////////////////////////////////////
void power_save(bit mod_rada) {

        if (mod_rada == 1) {
                PMR |= 0xA0;    // Prelazak u clock / 4
                PMR |= 0xE0;    // Prelazak u clock / 1024
        } else {
                PMR |= 0xA0;    // Prelazak u clock / 4
        }
}

//////////////////////////////////////////////////////////////////////
// Funkcija za resetiranje Watch Dog Timera
//////////////////////////////////////////////////////////////////////
void wd_r(void) {
        TA = 0xAA;
        TA = 0x55;
        RWT = 1;                // Reset WD timer-a
}

//////////////////////////////////////////////////////////////////////
// Funkcija za inicijalizaciju Watch Dog Timera
//////////////////////////////////////////////////////////////////////
void wd_i(void) {
        CKCON |= 0xC0;          // Postavljanje dijeljenja WD-a na 2^26
        TA = 0xAA;
        TA = 0x55;
        RWT = 1;                // Reset WD timer-a
        TA = 0xAA;
        TA = 0x55;
        EWT = 1;                // Omogućavanje Watch Dog-a
}

//////////////////////////////////////////////////////////////////////
// Funkcija za resetiranje Timera petlje nižeg prioriteta
//////////////////////////////////////////////////////////////////////
void reset_timer0 (void) {
  time=15*14;                // Reload vrijednost za 15 sekundi
      // Ovo dalje nije potrebno u normalnom načinu rada, samo u
      // Power save, kada je čitav ciklus timera, sam timer0
      TR0=0;            // Zaustavi Timer0
      TL0=0x01;         // Reload vrijednost za Timer 0
      TH0=0x01;
      TF0=0;            // Skini zastavicu overfloat
      TR0=1;            // Startaj Timer0
}

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
// Funkcija za inicijalizaciju serijskih portova
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
void ser_init (void) {
  EA=0;                    // Onemogući prekide
  EPF1=0;                  // Onemogući Power fail prekid
```

```
  // Komunikacija preko RS232 postavljena na 19200bps 8-N-1
  SCON0 = 0x52;           // Postavljanje moda prvog serijskog porta
  SCON1 = 0x52;           // Postavljanje moda drugog serijskog porta

  PCON |= 0x80;         // Udvostručavanje brzine prvog serijskog porta
  SMOD_1 = 1;           // Udvostručavanje brzine drugog serijskog porta

  TMOD = 0x20;          // 2 za Timer1 (RS232)
  TH1 = 0xFD;           // Postavljanje reload vrijednosti za 9600
  TL1 = 0xFD;

  IP  = 0;              // prioritet
  PS1 = 1;              // Visoki prioritet serije1
  TR1 = 1;              // Startaj Timer1

  ET1 = 0;              // Onemogući prekidnu rutinu za Timer1
  EA  = 1;              // uključi globalne interapte

        TA = 0xAA;      // Promjena TA bitova za Timed Access
        TA = 0x55;      // Specijalne registre
  P5CNT |=0x20;         // Postavljanje drugog serijskog porta
                        // na port P5.2 i P5.3
        TA = 0xAA;      // Promjena TA bitova za Timed Access
        TA = 0x55;      // Specijalne registre
  P5CNT &=0xF8;         // Konfiguracija od 64KB

        TA = 0xAA;      // Promjena TA bitova za Timed Access
        TA = 0x55;      // Specijalne registre
  P4CNT &=0x80;         // Konfiguracija od 64KB

  PMR = 0x27;           // Isključivanje ALE signala
        TA = 0xAA;      // Promjena TA bitova za Timed Access
        TA = 0x55;      // Specijalne registre
  MCON = 0x10;          // Postavke memorije

}

////////////////////////////////////////////////////////////////////
// Funkcija za prelazak na drugi način rada memorije
////////////////////////////////////////////////////////////////////
void romemu_swap(void) {
        unsigned int data i;

  P5 &= 0xBF;           // Postavljanje /ROMEMU_SWP linije u 0
        for(i=0;i<100;i++) _nop_();
        _nop_();
  P5 |= 0x40;           // Postavljanje /ROMEMU_SWP linije u 1
        PF1=1;          // Dojava programu da se izvrši od početka
}
```

```
//////////////////////////////////////////////////////////////////////
// Funkcija za ispis texta u 'ser1' na izlazni uređaj, tj. na
// sekundarni serijski port
//////////////////////////////////////////////////////////////////////
void printf1 (void)  {
  unsigned char data i,c;
  i=0;
  while (ser1[i] != '\0') {  // Petlja koja se izvršava do '\0'
      c=ser1[i];             // Kod dalje je standardna putchar funkcija
      if (c == '\n')  {
            if (RI1)  {
                  if (SBUF1 == XOFF)  {
                        do  {
                              RI1 = 0;
                              while (!RI1);
                        }
                        while (SBUF1 != XON);
                        RI1 = 0;
                  }
            }
            while (!TI1);
            TI1 = 0;
            SBUF1 = 0x0d;                        /* output CR  */
      }
      if (RI1)  {
            if (SBUF1 == XOFF)  {
                  do  {
                        RI1 = 0;
                        while (!RI1);
                  }
                  while (SBUF1 != XON);
                  RI1 = 0;
            }
      }
      while (!TI1);
      TI1 = 0;
      SBUF1 = c;
      i++;
  }
}

//////////////////////////////////////////////////////////////////////
//Funkcija za ispis liste podržanih naredbi
//////////////////////////////////////////////////////////////////////
void help (void) {
  printf("\nSMS COMMANDER version %s list of commands:\n\n",version);
  printf("HELP         This list of commands\n");
  printf("*            Repeat last command\n");
  printf("INFO         Information about hardware and software\n");
  printf("TIME         Information about date and time\n");
  printf("TEST         Data memory test and erase\n");
  printf("RESET        Restarting programme\n");
  printf("OFF          Shuts down the processor\n");
  printf("FLASH        Program memory upload over serial connection\n");
  printf("ECHO b       Direct redirection of answers from the GSM to the
terminal\n");
  printf("             without decoding, b: 0 - disable, 1 - enable, or
nothing\n");
  printf("AT           Redirected command to the GSM station\n");
  printf("MODEM        Permanent redirection to the GSM station\n");
  printf("OUT1 bbbbbbb Setting PIO port 1 bits 7 to 0, b:<0,1>\n");
```

```
  printf("OUT2 bbbb    Setting PIO port 2 bits 3 to 0, b:<0,1>\n");
  printf("INPUT        Reading PIO ports 1 and 2\n");
  printf("SENDSMS N    Sends SMS to number 'N', 'N': in MSN format:
+385XXXXXXXXX\n");
  printf("READSMS X Y  Reads SMS X: 0 - REC UNREAD, 1 - REC READ, 4 - ALL\n");
  printf("                        Y: 1 - delete SMS after read\n");
  printf("OPMSN        Operater's phone number in MSN format:
+385XXXXXXXXX\n");
  printf("\n");

}

////////////////////////////////////////////////////////////////////////
// Funkcija za pisanje na PIO port 1
////////////////////////////////////////////////////////////////////////
void out1(void) {
  if (ser0[5]=='0') PIO1_7=0; else PIO1_7=1;
  if (ser0[6]=='0') PIO1_6=0; else PIO1_6=1;
  if (ser0[7]=='0') PIO1_5=0; else PIO1_5=1;
  if (ser0[8]=='0') PIO1_4=0; else PIO1_4=1;
  if (ser0[9]=='0') PIO1_3=0; else PIO1_3=1;
  if (ser0[10]=='0') PIO1_2=0; else PIO1_2=1;
  if (ser0[11]=='0') PIO1_1=0; else PIO1_1=1;
  if (ser0[12]=='0') PIO1_0=0; else PIO1_0=1;
}

////////////////////////////////////////////////////////////////////////
// Funkcija za pisanje na PIO port 2
////////////////////////////////////////////////////////////////////////
void out2(void) {
  if (ser0[5]=='0') PIO2_3=0; else PIO2_3=1;
  if (ser0[6]=='0') PIO2_2=0; else PIO2_2=1;
  if (ser0[7]=='0') PIO2_1=0; else PIO2_1=1;
  if (ser0[8]=='0') PIO2_0=0; else PIO2_0=1;
}

////////////////////////////////////////////////////////////////////////
// Funkcija za čitanje stanja portova u varijable
// vraća unsigned int, masku promijenjenih portova
// get_pio (pio_tog)   pio_tog: 0 - ne mijenja staru vrijednost portova
////////////////////////////////////////////////////////////////////////
unsigned int get_pio (bit pio_tog) {
  unsigned int pio_t,j,k,l;
  unsigned char i;
  if (pio_tog) pio_old=pio;
  pio=0;
  pio_t=0;
  if (P4_7) pio |=0x0080;
  if (P4_6) pio |=0x0040;
  if (P4_5) pio |=0x0020;
  if (P4_4) pio |=0x0010;
  if (P4_3) pio |=0x0008;
  if (P4_2) pio |=0x0004;
  if (P4_0) pio |=0x0002;
  if (P4_1) pio |=0x0001;
  if (PIO2_3) pio |=0x0800;
  if (PIO2_2) pio |=0x0400;
  if (PIO2_1) pio |=0x0200;
  if (PIO2_0) pio |=0x0100;
  for (i=0;i<12;i++) { //Petlja za dobivanje: pio_t=pio XOR pio_old;
      j=pio;
```

```
      k=pio_old;
      j=_iror_(j,i);
      k=_iror_(k,i);
      j&=0x01;
      k&=0x01;
      l=j+k;
      l&=0x01;
      l=_irol_(l,i);
      pio_t|=l;
   }
   return (pio_t);
}

////////////////////////////////////////////////////////////////////////
// Funkcija za konvertiranje PIO u string
////////////////////////////////////////////////////////////////////////
void int_bin(void) {
   unsigned char i,k;
   unsigned int j;

   k=0;
   sprintf(ser1,"PIO: ");
   for(i=0;i<12;i++) {
   j=_iror_(pio,11-i);
   j&=0x01;
   if (k==4) k++;
   if (j==0) ser1[k+5]='0'; else ser1[k+5]='1';
   k++;
   }
   ser1[18]=0;
   ser1[9]=' ';
}

////////////////////////////////////////////////////////////////////////
// Funkcija za čitanje stanja portova
////////////////////////////////////////////////////////////////////////
void input(void) {
   get_pio(0);
   int_bin();
   printf("\n%s\n",ser1);
}

////////////////////////////////////////////////////////////////////////
// Funkcija za čekanje
////////////////////////////////////////////////////////////////////////
void wait(unsigned int i) {
   unsigned int j,k;

   for(j=0;j<i;j++) {
      wd_r();
      for(k=0;k<100;k++) _nop_();
   }
}
```

```
////////////////////////////////////////////////////////////////////////
// Funkcija za konverziju HEX string u int
////////////////////////////////////////////////////////////////////////
unsigned char hex_char (char c1, char c2){
  unsigned char i;

  if (isdigit(c1)) c1-=48; else c1-=55;
  if (isdigit(c2)) c2-=48; else c2-=55;
  i=0;
  i |= c1;
  i=_irol_(i,4);
  i |= c2;
  return(i);
}

////////////////////////////////////////////////////////////////////////
// Funkcija za provjeru i promjenu preusmjeravanja sa GSM-a
////////////////////////////////////////////////////////////////////////
void getset_echo (void){
  if (strlen(ser0)>=6) {
      if (ser0[5]=='1') echo=1;
      if (ser0[5]=='0') echo=0;
  }
  printf("Direct redirection from GSM device without decoding is ");
  if (echo) printf("enabled\n"); else printf("disabled\n");
}

////////////////////////////////////////////////////////////////////////
// Funkcija za provjeru i promjenu stalnog preusmjeravanja na GSM
////////////////////////////////////////////////////////////////////////
void set_modem (void){
  modem=1;
  printf("Permanent redirection to GSM device is enabled\n");
  printf("You will have to press reset button to return to");
  printf(" normal mode\n");
}

////////////////////////////////////////////////////////////////////////
// Funkcija za čitanje podatka sa serije1: sprema ga u 'ser1' string
// gets1 (OE)     OE: 1 - Prekida se uzimanje podataka nakon '\n'
////////////////////////////////////////////////////////////////////////
void gets1 (bit OE) {
  char c;
  int data i;
  if (strlen(ser1)>1)  printf1();        // Posalji naredbu GSM stanici
  ser1_broj=0;                 // Postavi broj upisanih znakova na 0
  ser1[0]=0;                   // Završi niz sa '\0'
  i=-2;
  do {
      if (RI1){                // Ako se pojavi karakter na ulazu serije1

            c=SBUF1;                                // Uzmi karakter sa serije1
            ser1[ser1_broj++]=c;                    // spremi ga u buffer
            i=-2;                                   // Postavi odbrojavanje
            if (ser1_broj>160) ser1_broj--;         // Ako smo prekoračili buffer
            if (c==0x0a || c==0x0d) {               // Ako se radi o 'Enter'
                  if (ser1_broj<=1) ser1_broj=0; // Zanemari linije bez texta
                  else if (OE) i=0;                 // Ako je setiran OE tada se
            }                                       // odmah izlazi van
            RI1=0;                                  // Uzeo znak sa serije1
      }
```

```
   } while (i--);
   ser1[ser1_broj]='\0';                          // Završavam string sa '\0'
   reset_timer0();                                // Resetiraj timer0
   wd_r();                                        // Resetiraj WDT
}

//////////////////////////////////////////////////////////////////////////
// Funkcija za ispis trenutnog datuma i vremena: sprema se u 'ser1'
//////////////////////////////////////////////////////////////////////////
int get_time(void) {
   unsigned char i;

   sprintf(ser1,"at+cclk?\n");gets1(1);           // Naredba za ispis sata
   if (strncmp(ser1,"+CCLK:",6)!=0) return (-1);
   for (i=0;i<17;i++) ser1[i]=ser1[i+8];
   ser1[8]=' ';
   ser1[i]=0;
   return(0);
}

//////////////////////////////////////////////////////////////////////////
// Funkcija za unos broja operatera
// OPMSN +1234567890123
//////////////////////////////////////////////////////////////////////////
int msno_input (void) {
   unsigned char i;

   if(strlen(ser0)==6) {      // Ako nije upisan broj
       printf("OK!\n\n Operater's phone number is %s\n",MSNO);
       return (0);
   }
   if(strlen(ser0)>6) {       // Upisana je i varijabla
       if(ser0[6]!='+') {     // Krivo upisan podatak
             printf(" ERROR! OPMSN usage:\n\n OPMSN +1234567890123\n");
             return(-1);
       }
   }
   printf("OK!\n");
   i=0;
   do {
       MSNO[i]=ser0[(i++)+7];
   } while (ser0[i+6]!='\n' && ser0[i+6]!=0);
   MSNO[i]=0;
   return(0);
}

//////////////////////////////////////////////////////////////////////////
// Funkcija za unos texta za čitanje sms poruke
// readsms 1 0
//////////////////////////////////////////////////////////////////////////
int read_sms_input (void) {
   unsigned char i,j;
   if (strlen(ser0)!=12) { // Pogreška pri unosu naredbe
       printf("\nERROR! READSMS command. Usage:\n");
       printf(" READSMS X Y\n");
       printf(" Where X: 0 - Recived unread messages\n");
       printf("          1 - Recived read messages\n");
       printf("          4 - All messages\n");
       printf("       Y: 1 - Delete messages after read\n");
       printf("          0 - Don't delete messages\n");
       return(-1);
```

```
   }

   i=toint(ser0[8]);
   j=toint(ser0[10]);
   if(read_sms(i,j)!=0) {
       printf("\n ERROR reading messages, or there is no messages!\n");
       return (-1);
   }
   return(0);
}

////////////////////////////////////////////////////////////////////////
// Funkcija za unos texta za slanje sms poruke
////////////////////////////////////////////////////////////////////////
int send_sms_input (void) {
   unsigned char i,c;
   if (strlen(ser0)<10 && ser0[8]!='+') { // Pogreška pri unosu naredbe
       printf("\nERROR! SENDSMS command. Usage:\n");
       printf(" SENDSMS +PHONE_NUMBER\n");
       printf("\n For example: SENDSMS +12345678901\n");
       return(-1);
   }
   i=8;
   while ((c=ser0[i++]) !=0 && c!=0x0F && c!=0x0D && c!=0x0A) MSN[i-9]=c;
                        // Kopiram broj iz 'ser0'
   MSN[i-9]=0;          // Dodajem ga u MSN i Završavam string sa '\0'

   printf("OK!\n\n Sending SMS to MSN: %s",MSN);
   printf("\n Now, enter your text for SMS,");
   printf(" and finish with 'CTRL+Z' or 'ESC'\n\n");

   ser1[0]=0;
   ser1_broj=0;
         TA = 0xAA;
         TA = 0x55;
         EWT = 0;                // Onemogućavanje Watch Dog-a

   do {      // Petlja za čitanje texta sa terminala

//     while (!RI!) wd_r();
       c=getchar();            // Uzimam podatak sa serije0
       ser1[ser1_broj++]=c;    // Upisi trenutni znak u string

       if (c==0x08) {          // Ako se radi o 'delete'
             ser1_broj-=2;     // Tada pokazivac stavi na zadnji točan znak
       }

   } while (c!=0x1a && c!=0x1b && ser1_broj<130);
   // Ako se radi o 'CTRL+Z' ili 'ESC' izađi van
   TA = 0xAA;
   TA = 0x55;
   EWT = 1;                    // Omogućavanje Watch Dog-a
   wd_r();                     // Reset Watch Dog Timer-a

   ser1[--ser1_broj]=0;        // Završavam string sa '\0'
   printf("\n\n Sending SMS...\n");
   wait(1000);
   i=send_sms(0);
   if (i==0) {
       printf(" SMS sent!\n");
       return(0);
```

```
   } else {
      printf(" SMS sending failed!\n");
      return(-1);
   }
}

////////////////////////////////////////////////////////////////////////
// Funkcija za ispis informacija o hardware-u i software-u
////////////////////////////////////////////////////////////////////////
void info (void) {
   printf("\nSMS Commander information:\n");
   printf("SMS Commander software version %s...\n",version);
   printf("SMS Commander hardware version 1.0...\n\n");

   printf("GSM station information:\n");

   sprintf(ser1,"at+cgmi\n");gets1(1);    // Naredba GSM stanici
   printf("GSM Station Vendor is %s\n",ser1);

   sprintf(ser1,"at+cgmm\n");gets1(1);    // Naredba za ispis modela
   printf("GSM Station Model is %s\n",ser1);

   sprintf(ser1,"ati9\n");gets1(1); // Naredba za ispis procesora
   printf("GSM Station Software is %s\n",ser1);

   printf("\nPIO ports information:");
   input();                   // Čitanje trenutnog stanja PIO portova

   if (get_time()==0)         // Čitanje trenutnog vremena iz GSM-a
   printf("\nNow is %s\n",ser1);
   printf("\nOperater's phone number is %s\n",MSNO);
   printf("\n");
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
// Funkcija za tumačenje upisane naredbe
////////////////////////////////////////////////////////////////////////
void decode_command (void) {
   if (strcmp(com0,"help")==0) {
                                    // Ispis podržanih naredbi
            printf("OK!\n");
            help();

   } else if (strcmp(com0,"at")==0) {
                                    // Prosljeđivanje naredbe na GSM
            wait(100);
            strcpy(ser1,ser0);      // Ispis na seriju1 upisane naredbe
      if (echo==0 && gsm==1) {
            gets1(0);
            printf("%s\n",ser1);
      } else {
            printf1();
      }
   } else if (strcmp(com0,"echo")==0) {
            getset_echo ();         // Preusmjeravanje odgovora sa GSM-a

   } else if (strcmp(com0,"readsms")==0) {
            printf("OK!\n");
            read_sms_input();       // Čitanje SMS poruka
```

```
} else if (strcmp(com0,"modem")==0) {
          set_modem ();     // Trajno preusmjeravanje na GSM

} else if (strcmp(com0,"sendsms")==0) {      // Slanje SMS poruke
          send_sms_input();

} else if (strcmp(com0,"opmsn")==0) {        // Promjena operatera
          msno_input();

} else if (strcmp(com0,"info")==0) {
          printf("OK!\n");
          info();                       // Ispis informacije o sustavu

} else if (strcmp(com0,"time")==0) {
    if (get_time()==0)                  // Ispis trenutnog vremena
    printf("OK!\n\nNow is %s\n",ser1);
    else
    printf("ERROR! This GSM station doesn't support this function\n");

} else if (strcmp(com0,"out1")==0) {
          out1();                       // Postavljanje PIO1
          printf("OK!\n");

} else if (strcmp(com0,"out2")==0) {
          out2();                       // Postavljanje PIO2
          printf("OK!\n");

} else if (strcmp(com0,"input")==0) {
          printf("OK!\n");
          input();                      // Ispis trenutnog stanja portova

} else if (strcmp(com0,"test")==0) {
                                        // Test i brisanje memorije
          printf("OK!\n");
          mem_test();       // Provjera i brisanje podatkovne memorije
          PF1=1;            // Direkcija programu da se izvrši od početka
          while (1);        // Resetiranje programa

} else if (strcmp(com0,"off")==0) {
                            // Gašenje procesora
          printf("OK!\nPress reset to power up processor...\n");
          printf("Turning power down...\n\n");
          romemu_swap();    // Prelazak natrag u bootstrap loader
          _nop_();
          PCON |= 0x02;     // Prelazak u SLEEP način rada

} else if (strcmp(com0,"reset")==0) {
                            // Resetiranje programa
          printf("OK!\n");
          printf("Restarting...\n");
          PF1=1;            // Direkcija programu da se izvrši od početka
          while (1);        // Resetiranje programa

} else if (strcmp(com0,"flash")==0) {
                            // Upis programa u programsku memoriju
          romemu_swap();    // Prelazak natrag u bootstrap loader
          printf("OK!\n");
          printf("\n Going to load bootstrap again...\n");
          wait(2500);
          printf(" Prepare to upload new programme...\n");
          while (1);        // Resetiranje programa
```

```
   } else if (strlen(com0)>0) {
                                    // Ako nije ni jedno od prethodnog
                                    // -> greška u unosu
            printf("ERROR! Type 'help' for list of commands...\n");
   }
}

//////////////////////////////////////////////////////////////////////
// Funkcija za trajno preusmjeravanje naredbi na GSM uređaj
//////////////////////////////////////////////////////////////////////
void ser0_modem (void) {
   char c;
           TA = 0xAA;
           TA = 0x55;
             EWT = 0;                // Onemogućavanje Watch Dog-a
   while(1) {
       if (RI) {                    // Ako se pojavi karakter na ulazu serije0

             c=SBUF0;               // Pročitaj znak sa serije
             RI=0;                  // Skini zastavicu dolaznog znaka

             if (c == '\n')  { // Preusmjeri ga na modem
                   if (RI1)  {
                         if (SBUF1 == XOFF)  {
                               do  {
                                     RI1 = 0;
                                     while (!RI1);
                               }
                               while (SBUF1 != XON);
                               RI1 = 0;
                         }
                   }
                   while (!TI1);
                   TI1 = 0;
                   SBUF1 = 0x0d;                        /* output CR  */
             }
             if (RI1)  {
                   if (SBUF1 == XOFF)  {
                         do  {
                               RI1 = 0;
                               while (!RI1);
                         }
                         while (SBUF1 != XON);
                         RI1 = 0;
                   }
             }
             while (!TI1);
             TI1 = 0;
             SBUF1 = c;
       }
       while (RI1) {                // Ako se pojavi karakter na ulazu serije1

             c=SBUF1;               // Pročitaj znak sa serije
             RI1=0;                 // Skini zastavicu dolaznog znaka
             putchar(c);            // Posalji znak na seriju0
       }
   }
}
```

```
//////////////////////////////////////////////////////////////////////
// Funkcija za tumačenje naredbe iz SMS poruke
//////////////////////////////////////////////////////////////////////
void decode_sms(void) {
  unsigned char xdata command_sms[10],i;
  i=0;
  while (ser1[i]!=' ') command_sms[i++]=tolower(ser1[i]);
  command_sms[i]=0;
  if (strcmp(command_sms,"out1")==0) {
      if (ser1[5]=='0') PIO1_7=0; else PIO1_7=1;
      if (ser1[6]=='0') PIO1_6=0; else PIO1_6=1;
      if (ser1[7]=='0') PIO1_5=0; else PIO1_5=1;
      if (ser1[8]=='0') PIO1_4=0; else PIO1_4=1;
      if (ser1[9]=='0') PIO1_3=0; else PIO1_3=1;
      if (ser1[10]=='0') PIO1_2=0; else PIO1_2=1;
      if (ser1[11]=='0') PIO1_1=0; else PIO1_1=1;
      if (ser1[12]=='0') PIO1_0=0; else PIO1_0=1;
  }
  if (strcmp(command_sms,"out2")==0) {
      if (ser1[5]=='0') PIO2_3=0; else PIO2_3=1;
      if (ser1[6]=='0') PIO2_2=0; else PIO2_2=1;
      if (ser1[7]=='0') PIO2_1=0; else PIO2_1=1;
      if (ser1[8]=='0') PIO2_0=0; else PIO2_0=1;
  }

}

//////////////////////////////////////////////////////////////////////
// Funkcija za čitanje podatka i izdvajanje naredbe sa serije0
//////////////////////////////////////////////////////////////////////
void ser0_read (void) {
  char data c;

  if (RI) {            // Ako se pojavi karakter na ulazu serije0

      c=getchar();     // Uzimam podatak sa serije0
      reset_timer0();

      if (c=='*' && ser0_broj==0) {       // Ako želimo ponoviti staru
            printf("\r%s",ser0);          // već izvršenu komandu
            decode_command();
      } else {                            // Inače normalna procedura

      if (com0_set==0) {            // Ako komanda već nije upisana do kraja
            com_broj=ser0_broj;     // Izjednači trenutni znak komande
                                    // upisanim stringom

            com0[com_broj]=tolower(c);    // Pretvori velika u mala slova

            if (c==0x20 || c==0x0a || com_broj>8) {  // Ako se radi o
                  // razmaku, ili enteru, ili premašenom spremniku za
                  // komande, tada završi komandu
                  com0_set=1;
                  com0[com_broj]=0; // Na kraj stavi '\0'
            }

            // Ako se radi o 'at' komandi onda je ona već završena
            if (com_broj==1 && com0[0]=='a' && com0[1]=='t'){
                  com0[++com_broj]=0;     // Završavam commandu sa '\0'
                  com0_set=1;             // Postavljam kraj komande
            }
```

```
      }
      ser0[ser0_broj]=c;                  // Upisi trenutni znak u string
      if (ser0_broj<60) ser0_broj++;      // Ako nismo na kraju
      // buffera za string onda pomakni pokazivac na slijedeći

      if (c==0x08) {          // Ako se radi o 'delete'
            ser0_broj-=2;     // Tada pokazivac stavi na zadnji točan znak
            if (ser0_broj>250) ser0_broj=0; // Ako je broj manji od 0
            if (com_broj>ser0_broj) com0_set=0; // Isto ako to zadire u
                              // komandu, tada makni zastavicu da je završena
      }
      if (c==0x0a) {          // Ako se radi o 'Enter' tada izvrši
                              // naredbe koja je upisana
            ser0[ser0_broj]=0;// Završavam string sa '\0'

            decode_command(); // Protumači upisanu naredbu

            ser0_broj=0;      // Postavi broj upisanih znakova na 0
            com0_set=0;
      }
      }
   }
}

//////////////////////////////////////////////////////////////////////
// Funkcija za provjeru dolaznog poziva
// GSM vraća:
// RING
//
// +CLIP: +38598XXXXXX
//
//////////////////////////////////////////////////////////////////////
int check_call (void) {
   unsigned char i,j;
   unsigned char broj[16];
   if (strlen(ser1)<8) return(-1);        // Ako nema +CLIP:
   i=0;j=0;
   while (ser1[i]!=':') i++;
   while (ser1[i]!='+') i++;
   while (ser1[i]!=',') {
       broj[j]=ser1[i];
       i++;j++;
       if (j>15) return (-1);             // Ako se prekorači spremnik
   }
   broj[j]=0;
   if(strcmp(MSNO,broj)==0) PIO2_3=0;     // Ako je broj operatera upali LED
   else return (-1);
   return (0);
}
```

```
//////////////////////////////////////////////////////////////////////////
// Funkcija za čitanje podatka sa serije1
//////////////////////////////////////////////////////////////////////////
void ser1_read (void) {
  char c;

  while (RI1) {              // Ako se pojavi karakter na ulazu serije1
     c=SBUF1;                // Pročitaj znak sa serije
     RI1=0;                  // Skini zastavicu dolaznog znaka
     putchar(c);             // Posalji znak na seriju0
     reset_timer0 ();        // Postavi novu reload vrijednost u Timer
  }
}



//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
// Glavni program
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
void main (void) {
//volatile unsigned char xdata *pokazivac;
//unsigned char data varijabla;
//unsigned int temp;
//float temperatura;
  int xdata i;
  bit quick=0;

  strcpy(MSNO,"+38598754369");     // Broj operatera

  ser0_broj=0;               // Kažemo da nema znakova na ulazu serije0
  ser1_broj=0;               // Kažemo da nema znakova na ulazu serije1
  com_broj=0;                // Kažemo da broj znakova komande nula
  com0_set=0;                // Kažemo da komanda nije postavljena
  echo=0;                    // Direktno preusmjeravanje sa serije1 na seriju0
                             // bez dekodiranja odgovora
  modem=0;                   // Isključeno stalno preusmjeravanje

  ser_init();                // Inicijalizacija serije
  wd_i();                    // Uključivanje Watch Dog-a
  get_pio(1);                // Uzimanje trenutnih PIO vrijednosti

  CKCON &= 0xF7;             // Postavljanje dijeljenja clocka sa 12 za timer0
  TL0=0x01;                  // Postavljanje reload vrijednosti
  TH0=0x01;
  TMOD &= 0xF0;              // Timer 0 u MOD0
  TMOD |= 0x01;              // Timer 0 u MOD1: 16 bit
  TF0=0;                     // Skini zastavicu overfloat

  if (WTRF==1 && PF1==0) printf("\n WDT time out...\n");
  // Provjera dali je  nastao reset uslijed WD Timera

  WTRF = 0;             // Resetiraj bit WatchDog Timera
  wd_r();               // Resetiraj WatchDog Timer
  PF1=0;                // Dojava programu da se ne izvrši od početka


  printf ("\n\n*******************************");
  printf("*********************************\n");
  printf("******* SMS COMMANDER **** v%s",version);
```

```
printf(" **** by Goran Jurkovich **********\n");
printf("**********************************");
printf("**********************************\n\n");

gsm=0;                              // GSM uređaj inicijalno ne postoji
sprintf(ser1,"at\n");               // Prozivanje GSM uređaja
gets1(0);                           // Slanje komande na seriju1
sprintf(ser1,"atz\n");              // Resetiranje GSM uređaja
gets1(0);                           // Slanje komande na seriju1
sprintf(ser1,"ate0v0\n");           // Isključivanje 'echo' f-je
gets1(0);                           // Slanje komande na seriju1
sprintf(ser1,"at\n");
gets1(0);
if (ser1[0]=='0') gsm=1;
    else printf(" ERROR! Did not get respond from GSM device!\n");

if (gsm) PIO2_3=0; else PIO2_0=0;// Indikacija korisniku
                                 // ako terminal nije spojen

if (gsm) {
    sprintf(ser1,"at+clip=1\n");        // Uključivanje CLIP f-je
    gets1(0);                           // Slanje komande na seriju1
    sprintf(ser1,"at+calm=2\n");        // Isključivanje Zvučnih alarma
    gets1(0);                           // Slanje komande na seriju1
    sprintf(ser1,"at+cvib=0\n");        // Isključivanje vibracijskog alarma
    gets1(0);                           // Slanje komande na seriju1

    sprintf(ser1,"at+cgmi\n");gets1(1); // Naredba GSM stanici
    printf(" GSM Station Vendor is %s\n",ser1);

    sprintf(ser1,"at+cgmm\n");gets1(1); // Naredba za ispis modela
    printf(" GSM Station Model is %s\n",ser1);

    sprintf(ser1,"ati9\n");gets1(1);    // Naredba za ispis procesora
    printf(" GSM Station Software is %s\n",ser1);
    ser1[0]=0;gets1(0);                 // Ako ima još koja dolazna poruka
    }

printf("\n Type 'help' to get some assistance...\n\n");

power_save(1);              // Stavi procesor u spori način rada
reset_timer0();             // Resetiraj timer0
TR0=1;                      // Startaj Timer0

while (1) {           // Beskonačna petlja koja se stalno izvršava

    if (modem)                      // Ako je selektiran modem
          ser0_modem();             // Napravi trajnu redirekciju
    else
          ser0_read();              // Pročitaj podatak sa serije0

    if (echo) {                     // Ako je direktna redirekcija
          ser1_read();              // Pročitaj podatak sa serije1
    } else if (RI1) {               // Ako nije 'ECHO' dekodiraj odgovor
          ser1[0]=0;
          gets1(0);
          if (ser1[0]=='2')         // Ako se detektira poziv
          if (check_call()==0)      // Provjeri tko zove
          printf(" High priority command detected!\n");
          else printf("%s\n",ser1);     // Ispisi odgovor na terminalu
    }
```

```
      if (!P4_7 && !quick) {              // Ako se radi o informaciji velikog
            quick=1;                 // prioriteta
            printf("\n High priority signal detected!\n");
            sprintf(ser1,"\natdt%s;\n",MSNO);
            printf1();               // Posalji komandu GSM-u
            while (!RI1) wd_r();     // Dok nema odgovora čekaj
            printf(" Operator respond OK!\n");
            sprintf(ser1,"at\n");
            gets1(0);
            sprintf(ser1,"ate0v0\n");
            gets1(0);
            reset_timer0();
      }
      if (P4_7 && quick) {
            quick=0;
      }

      if (TF0) {               // Timer za petlju nižeg prioriteta
            TR0=0;             // Zaustavi Timer0
            TL0=0x01;          // Reload vrijednost za Timer 0
            TH0=0x01;
            TF0=0;             // Skini zastavicu overfloat
            TR0=1;             // Startaj Timer0
            time--;
            if (PMR&0x40) time=0;   // Ako je u power save načinu rada
      }                             // Tada se timer vrti 256 puta sporije
                                    // I nije potrebno odbrojavati brojač

      if (!time) {                  // Petlja nižeg prioriteta
                                    // Koja se izvršava 15 sekundi nakon
                                    // neaktivnosti korisnika, tj. nakon isteka
                                    // vremena timera 0

//          printf("\nLow priority loop!\n");   // DEBUG

            PIO2_3=1;               // Isključi brzi proces

            if(get_pio(1)!=0) {                 // Ako se dogodi promjena ulaza
                  int_bin();                    // Prebaci PIO u string
                  printf(" %s\n",ser1);         // Ispisi trenutno stanje
                  sprintf(ser1,"%s\n",ser1);    // Ispisi ga u SMS buffer
                  strcpy(MSN,MSNO);             // Definiraj broj kojemu se šalje
                  i=send_sms(0);                // Posalji obavijest korisniku
                  if (i!=0) printf(" Error sending automatic SMS!\n");
                  else printf(" Automatic SMS Sent!\n");
            }

            read_sms(0,1);
            // Provjeri nove poruke i izvrši ako korisnik ne koristi
            // komandni interpreter
            power_save(1);              // Stavi procesor u spori način rada
            reset_timer0();             // Resetiraj timer0
      }
      wd_r();                           // Resetiraj WatchDog Timer
   }
}
//////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////
// Funkcija za slanje SMS poruke:  broj u formatu +38598xxxxxxx
//                                 poruka se prenosi u 'ser1'
//                                 broj se prenosi u 'MSN'
//    send_sms (RC)                RC: 0,1 - request confirmation
////////////////////////////////////////////////////////////////////////
int send_sms (bit rc) {
  unsigned char i,j,j2,b,b2,c,c2,len_m,len_n;
  unsigned char xdata buffer[340],buffer2[340],buffer3[4];
  long wait_time=0xFFFFF;

  if (!gsm) return (-1);           // Ako GSM uređaj nije pronađen
  sprintf(buffer,"%s\n\nSent by SMS COMMANDER v%s",ser1,version);

  // U stringu 'buffer' se nalazi poruka
  // Prvo da text poruke prebacimo u 7-bit ASCII standard

  for (i=0;i<=strlen(buffer);i++) buffer[i]=toascii(buffer[i]);
  len_m=strlen(buffer);      // Duzina poruke
  len_n=strlen(MSN)-1;       // Duzina MSN broja

  // Broj je potrebno staviti u semi octet!
  // Normalan broj     +12345678901      +123456789012
  // Semi oktet        2143658709F1      214365870921
  i=0;
  do {
     if (MSN[i+2]==0)  // Ako dođemo do kraja stringa
     MSN[i]='F';             // Tada ubacujemo 'F'
     else
     MSN[i]=MSN[i+2];  // Ili normalno rotiramo 2 člana iza
     i+=2;
  } while (i<len_n);
  MSN[i]=0;

  ////////////////////////////////////////////////////////////////////////
  //  printf("\nBroj: %s\n",MSN);         // Zbog debugiranja
  //  printf("Poruka: %s\n",buffer);      // Zbog debugiranja
  ////////////////////////////////////////////////////////////////////////
  // Konverzija 7 bit niza u 8 bit niz te u ASCII HEX
  // Po definiciji PDU standarda

  j=0;j2=0;       // Polazim od početnih karaktera oba niza
  b=0x01;         // Prvi karakter ulaznog niza, uzima se 0 bit
                  // b = 0000 0001 (2)
  b2=0x01;        // Prvi karakter izlaznog bita stavlja se na 0 bit
                  // b2= 0000 0001 (2)
  i=2;            // Niti jedan niz nije došao do kraja
  c2=0;           // Prvi karakter ulaznog niza je '\0'
                  // Ovo je bitno zbog OR naredbe koja se kasnije radi
  do {
     c=buffer[j];      // Stavim karakter ulaznog niza u 'c'
     if (c==0) i=1;    // Ako smo došli do kraja ulaznog niza > i=1
     c&=b;             // Izvrim C = (C AND b) maskiram samo 1 bit
     b=_crol_(b,1);    // Rotiram b u lijevo kako bi se slijedeći puta
                       // uzeo slijedeći bit ulaznog niza
     if (c != 0) c2|=b2;     // Ako je 'c' različit od 0, znaci maskirani
                             // bit je 1 => C2 = (C2 OR b2): Digni u 1
                             // taj bit koji se zapisuje na izlazu
     b2=_crol_(b2,1);        // Rotiraj b2 u lijevo kako bi se slijedeći
                             // puta zapisivalo u slijedeći bit
```

```
    if (b==0x80) {          // Ako je 'b' na početku (7-bit), znaci ulazni
                            // karakter je završen. Idemo na slijedeći
          b=0x01;           // Opet postavljamo masku na 0 bit
          if (i==2) j++;    // Ako nije kraj ulaznog niza postavi
                            // pokazivac na slijedeći karakter
    }
    if (b2==0x01){          // Ako je 'b' na početku, znaci izlazni
                            // oktet je završen.
          buffer2[j2]=c2;   // Zapisi 'c2' karakter u izlazni niz
                            // Na trenutno mjesto
          c2=0;             // Postavi prazan karakter radi kasnije
          b2=0x01;          // OR naredbe
          j2++;             // Postavi pokazivac na slijedeći karakter
          if (i==1) i=0;    // Ako je ulazni niz završen (i=1), tada
    }                       // Znaci da je kraj konverzije (i=0)
} while (i);
buffer2[j2]=0;
// 7-bitni ulazni niz je konvertiran u 8-bitni niz
// Sada je isti potrebno prebaciti u HEX-ASCII
j=0;
for (i=0;i<strlen(buffer2);i++) {      // Uzimam po jedan karakter
    c=buffer2[i];                      // ulaznog niza
    sprintf(buffer3,"%bX",c);          // konvertiram ih u ASCII-HEX
                                       // koji je ujedno sada velik
                                       // 2 karaktera
          // Sada se svaki od njih zapisuje u izlazni niz
    if (buffer3[1]==0){    // Ako je broj manji od 0x10, dakle
                           // jednoznamenkasti, onda mu treba dodati
                           // 0 ispred
          buffer[j++]='0';buffer[j++]=buffer3[0];
    }else{                 // Inače je dvoznamenkasti broj veći od 0x10
          buffer[j++]=buffer3[0];buffer[j++]=buffer3[1];
    }
}
buffer[j]=0;          // U 'buffer' se nalazi konvertiran niz
buffer[++j]=0;        // zapisan u HEX, ASCII formatu

////////////////////////////////////////////////////////////////////////
// Sada je taj niz potrebno ukomponirati u PDU standard
// Ako ne treba delivery report, tada promijeni '31' u '11'
////////////////////////////////////////////////////////////////////////

if (rc==1) i='3'; else i='1';
if (len_m<0x10) // Ako je broj manji, opet isti problem, manjka '0'
sprintf(buffer2,"00%c1000%bX91%s0000AA0%bX%s",i,len_n,MSN,len_m,buffer);
else
sprintf(buffer2,"00%c1000%bX91%s0000AA%bX%s",i,len_n,MSN,len_m,buffer);

////////////////////////////////////////////////////////////////////////
// U 'buffer2' se nalazi gotov PDU
// Sada PDU, dodamo AT naredbu
////////////////////////////////////////////////////////////////////////

sprintf(ser1,"\nat+cmgs=%bd\n\r",(_cror_(strlen(buffer2),1))-1);
gets1(0);       // Posalji naredbu za prelazak u PDU mod za slanje
wait (10000);  // Pričekaj malo
sprintf(ser1,"%s%c",buffer2,0x1a);
printf1();                        // Posalji PDU blok
ser1[0]=0;                        // Posalji prazan string
while (!RI1 && wait_time) {       // Čekaj dok GSM vrati odgovor
    wd_r();                       // Resetiraj WDT
```

```
      wait_time--;
   }
   gets1(1);                                        // Uzmi odgovor
   if (strncmp(ser1,"+CMGS:",6)==0) return (0); // Ako je slanje uredu
   return (-1);                // Ako je došlo do greške u slanju SMS-a
}

////////////////////////////////////////////////////////////////////////
// Funkcija za čitanje SMS poruke: broj u formatu +38598xxxxxxx
//                                 poruka se vraća u 'ser1'
//                                 broj se vraća u 'MSN'
//    read_sms (TYPE,DEL)          TYPE: 0 - Read new
//                                       1 - Read inbox
//                                       4 - Read all
//                                 DEL:  1 - Delete message
////////////////////////////////////////////////////////////////////////
int read_sms (char TYPE, char DEL) {
   int br=-1;
   unsigned char c,c2;
   int i;
   unsigned int nbr,pos,pos2,pos3,j,j2;
   unsigned char len,tmp,b,b2,mes;
   unsigned char xdata buf[20*162];

   reset_timer0();                          // Resetiraj Timer0
   if(!gsm) return(-1);                     // Ako GSM uređaj nije pronađen
   sprintf(ser1,"at+cmgl=%bd\n\r",TYPE);  // Naredba za listanje poruka
   printf1();                               // Slanje naredbe na GSM

   while((br--) && (!RI1)) wd_r();        // Čekaj neko vrijeme na odgovor

   nbr=0;                                   // Postavi broj upisanih znakova na 0
   buf[0]=0;                                // Završi niz sa '\0'
   i=-1;
   do {
      if (RI1){               // Ako se pojavi karakter na ulazu serije1
            c=SBUF1;                        // Uzmi karakter sa serije1
            buf[nbr++]=c;                   // spremi ga u buffer
            i=-1;                           // Postavi odbrojavanje
            if (nbr>3200) i=0;              // Ako smo prekoračili buffer
            if (c==0x0a || c==0x0d) {     // Ako se radi o 'Enter'
                  if (nbr<=1) nbr=0;
            }
            RI1=0;                          // Uzeo znak sa serije1
      }
      wd_r();
   } while (i--);
   buf[nbr]='\0';                           // Završavam string sa '\0'
   buf[nbr+1]=0;
   wait(50);                                // Malo pričekaj i resetiraj WDT
   if(nbr<8) return (-1);                   // Nema SMS poruka

   ////////////////////////////////////////////////////////////////////
   // Dekodiranje odgovora
   // U 'buf' se nalazi čitav odgovor
   // Odgovor izgleda nekako ovako:
   // +CMGL: 3,1,,22
   // 069183950805F1240B918395584475F400001080134183928003C1F51B
   //
   // OK
   ////////////////////////////////////////////////////////////////////
```

```
pos=0;                                  // Trenutni pokazivac u stringu
do {
    if (buf[pos]!='+') return(-1);      // Provjerim dali počinje sa '+'
    while(buf[++pos]!=',');             // Tražim prvi ','
    mes=0;                              // Broj poruke=0
    pos2=pos-1;                         // Sada odbrojavam u lijevo
    while((c=buf[--pos])!=' ') {        // Do razmaka i pročitam broj
          if ((pos2-pos)==0) mes+=(c-48);          // poruke (3)
          if ((pos2-pos)==1) mes+=((c-48)*10);
    }
    printf("\n SMS Number: %bu\n",mes);
    len=0;                              // Duzina čitavog PDU bloka bez SMSC
    while(buf[++pos]!='\n');
    pos--;
    pos2=pos-1;
    while((c=buf[--pos])!=',') {
          if ((pos2-pos)==0) len+=(c-48);
          if ((pos2-pos)==1) len+=((c-48)*10);
          if ((pos2-pos)==2) len+=((c-48)*100);
    }                                   // Izdvajam taj broj (22)
    while(buf[pos++]!='\n');
    pos2=pos;                           // Početak PDU bloka
    pos+=(hex_char (buf[pos],buf[pos+1])+1)*2;
    pos+=2;
    pos2=pos;                           // Početak bloka sa brojem pošiljatelja
    tmp=(hex_char(buf[pos],buf[pos+1]));     // Duzina broja
    pos3=pos+tmp+4;                          // Na kraj broja
    pos+=4;                                  // Na početak broja
    if (tmp&0x01) pos3++;                    // Ako je neparan broj
    c=0;
    do MSN[c++]=buf[pos++]; while (pos<pos3);
    MSN[c++]=0;
    MSN[c--]=0;
    do {
          MSN[c]=MSN[c-2];
          c-=2;
    } while (c>0);
    MSN[0]='+';
    MSN[tmp+1]=0;
    printf(" MSN Number: %s\n",MSN);

    pos=pos3;                                // Iza broja PDU
    pos+=18;                                 // Iza timestamp
    pos3=pos;
    tmp=hex_char(buf[pos],buf[pos+1]);       // Duzina poruke
    pos+=2;

////////////////////////////////////////////////////////////////////////
// Konverzija HEX 8 bit niza u 7 bit ASCII
// Po definiciji PDU standarda
////////////////////////////////////////////////////////////////////////

    j=pos;j2=0;       // Polazim od početnih karaktera oba niza
    b=0x01;           // Prvi karakter ulaznog niza, uzima se 0 bit
                      // b = 0000 0001 (2)
    b2=0x01;          // Prvi karakter izlaznog bita stavlja se na 0 bit
                      // b2= 0000 0001 (2)
    i=2;              // Niti jedan niz nije došao do kraja
    c=0;
    c2=0;             // Prvi karakter ulaznog niza je '\0'
```

```
                        // Ovo je bitno zbog OR naredbe koja se kasnije radi
  do {
      c=hex_char(buf[j],buf[j+1]);
                        // Stavim karakter ulaznog niza u 'c'
      c&=b;             // Izvrim C = (C AND b) maskiram samo 1 bit
      b=_crol_(b,1);    // Rotiram b u lijevo kako bi se slijedeći puta
                        // uzeo slijedeći bit ulaznog niza
      if (c != 0) c2|=b2;     // Ako je 'c' različit od 0, znaci maskirani
                              // bit je 1 => C2 = (C2 OR b2): Digni u 1
                              // taj bit koji se zapisuje na izlazu
      b2=_crol_(b2,1);        // Rotiraj b2 u lijevo kako bi se slijedeći
                              // puta zapisivalo u slijedeći bit

      if (b==0x01) {          // Ako je 'b' na početku (8-bit), znaci ulazni
                              // karakter je završen. Idemo na slijedeći
            b=0x01;           // Opet postavljamo masku na 0 bit
            if (i==2) j+=2;   // Ako nije kraj ulaznog niza postavi
                              // pokazivac na slijedeći karakter
      }
      if (b2==0x80){          // Ako je 'b2' na početku, znaci izlazni
                              // oktet je završen.
            ser1[j2]=c2;      // Zapisi 'c2' karakter u izlazni niz
                              // Na trenutno mjesto
            c2=0;             // Postavi prazan karakter radi kasnije
            b2=0x01;          // OR naredbe
            j2++;             // Postavi pokazivac na slijedeći karakter
            if (j2>=tmp) i=0;
      }                       // Znaci da je kraj konverzije (i=0)

      wd_r();                 // Reset WDT-a
  } while (i);
      ser1[j2]=0;
      ser1[j2+1]=0;
      printf("\n%s\n",ser1);

      // Ako je broj pošiljatelja jednak operateru, izvrši SMS
      if (strcmp(MSN,MSNO)==0) {
            decode_sms();
            sprintf(ser1,"\nat+cmgd=%bu\n",mes); gets1(0);
      }
      if (DEL) {  // Ako je uključeno brisanje poruka, tada iste obrisi
            sprintf(ser1,"\nat+cmgd=%bu\n",mes); gets1(0);
      }
      pos=j;
      while(buf[++pos]!='\n' && buf[pos-1]!=0 && pos<nbr);
      pos++;
  } while (buf[pos]=='+');
  printf("\n End of messages.\n");
  ser1[0]=0;
  gets1(0);       // Pokupi odgovore ako još koji ima na GSM-u
  return(0);
}
//////////////////////////////////////////////////////////////////////////
```