UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
KATEDRA ZA ELEKTRONIKU

Predmet: MIKROPROCESORSKI SISTEMI

PRENOS PODATAKA I KORIGOVANJE GREŠKE PRIMENOM HAMINGOVOG KODA

Student:
Dušan Živković 9622

Niš, oktobar 2007.

Sadržaj:

1. UVOD

Svaki put kada se informacija prenosi kroz neki kanal, može doći do greške u prenosu usled šuma. Ustvari, čak i kada je informacija smeštena u nekoj komponenti za skladištenje podataka (memoriji), može se oštetiti, jer nijedan hardver nije apsolutno siguran. Ovo se naravno ne odnosi samo na kompjuterske informacije. Glas kada se prenosi kroz vazduh može se iskvariti usled vetra, šuma, visoke temperature...Zato vrlo je značajna mogućnost otkrivanja i korekcije grešaka unutar jednog sistema.

U opštem slučaju postoje dva pristupa:

1. *Kontrola greške unapred*, u kome svaki karakter ili blok podataka koji se šalje sadrži dodatne (redundatne) informacije. Na osnovu ovih informacija primalac može ne samo da otkrije postojanje greške, već i da odredi koji su bitovi u primljenom podatku pogrešni i da grešku ukloni negacijom pogrešnih bitova.

2. *Kontrola greške unazad*, odnosno kontrola sa povratnom spregom. U ovom slučaju svaki karakter ili blok podataka koji se šalje sadrži samo neophodne dodatne informacije koje omogucuju primaocu da otkrije prisustvo greške, ali ne i mesto na kome se ona nalazi. U tom slucaju jedini nacin za dobijanje korektnih podataka je ponavljanje prenosa svih karaktera ili blokova podataka koji imaju grešku.

U praksi, broj dodatnih bitova brzo raste sa porastom dužine podataka koji se prenose. Posledica toga je:
1. Metod sa kontrolom greške unazad je najceše koristi u telekomunikaconom prenosu podataka.
2. Metod sa kontrolom greške unapred se češće koristi pri čuvanju podataka na memorijskim medijumima u računarskom sistemu.

Što se tiče metoda za otkrivanje grešaka, faktori od kojih zavisi izbor metode:
1. Broja bitova sa greškom (eng. bit error rate, BER) na komunikacionoj liniji. BER predstavlja verovatnoću u definisanom vremenskom intervalu da jedan bit ima grešku. Tako BER=10-4 znači da, u proseku, 1 od 10000 bita ima grešku u definisanom vremenskom intervalu. Za najveći broj mreža BER je oko 10-4 , dok je unutar računarskog sistema obično vrednost za BER 10-4 ili manje.
2. Tipa greške, tj. da li se greška javila na pojedinačnom bitu ili na grupi uzastopnih bitova. U poslednjem slučaju reč je o proširenoj (eksplozivnoj) grešci (eng. burst error).

2. Hamingovo kodiranje

U telekomumikacijama hamingov kod je linearni kod za korekciju gresaka, nazvan po njegovom izumitelju Ricardu Hamingu. Hamingov kod moze detektovati i korigovati jednobitnu gresku i moze detektovati ali ne i korigovati dvobitnu gresku, odnosno Hamingova distanca izmedju poslatog i primljenog kodnog zapisa mora biti nula ili jedan za pouzdanu komunikaciju.

2.1.ISTORIJA

Haming je četrdesetih godina proslog veka radio u Bell laboratoriji na Bell model V racunaru, masini zasnovanoj na elektromehanickom releju. Unos je vrsen sa punch karticama, koje su nedvosmisleno nosile greske. Tokom radne nedelje specijalni kod bi nalazio greske i lampica bi zasvetlela, tako da su operateri mogli da otklone problem. Posle radnog vremena ili vikendom, kada nije bilo operatera, masina bi jednostavno nastavila sa sledecim poslom.
Hamig je radio vikendom i bio ogorcen sto je morao da restartuje programe zbog nepouzdanosti citaca kartica. Nekoliko narednih godina radio je na problemu otklanjanja gresaka, razvijajuci mocan niz algoritama. 1950 je obelodanio ono sto je danas poznato kao Hamingov kod.
Odredjeni broj jednostavnih kodova za detekciju gresaka je bio u upotrebi i pre Hamingovog, ali nijedan nije bi tako efikasan.

Pre Hamingovog koda u upotrebi su bili sledeći kodovi:

Parity bit:

Parity bit je bit koji pokazuje da li je broj bitova vrednosti jedan paran ili neparan. Ako je neki bit promenjen tokom prenosa, promenice se parity bit i greska može biti detektovana u toj tacki (parity bit može biti i sam promenjen). Obično vrednost parity bita 1 pokazuje da je neparan broj jedinica u zapisu, a vrednost 0 da je u piatanju paran broj jedinica u zapisu. Drugim rečima zapis i parity zajedno mogu sadržati paran broj jedinica.
Parity ne prijavljuje na kom bitu je greska iako moze da otkrije.

Dva iz pet kod:

Četrdesetih godina prošlog veka Bell je koristio prefinjeniji *m od n* kod poznat kao *dva iz pet* kod. Kod garantuje da svaki niz od 5 bitova ima tačno dve jedinice. Racunar prijavljuje grešku ako blok od pet bitova nema tačno dve jedinice, ali ako neki bit u nizu od pet se promeni u jedinicu a drugi u nulu, kod ne prijavljuje grešku.

Ponavljanje:

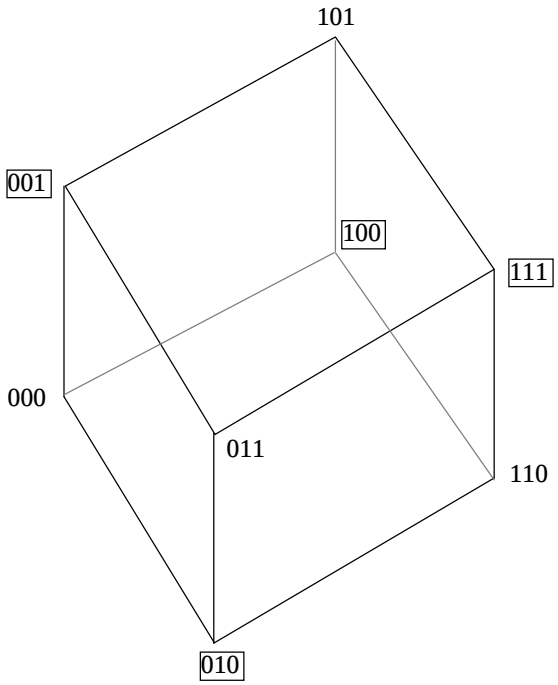
Ponavlja svaki bit više puta da bi osigurao da je prošao. Ako je bit koji se šalje 1, *n=3 ponavljanje kod* šalje"111", Ako tri primljena bita nisu identična, javlja se greška. Ovaj kod takodje nije efikasan.

2.2. Pojam Hamingove distance

▪ Hamingova distanca (HD)
- broj bitova u kojima se dve digitalne reči razlikuju
  HD(101, 100) = 1
  HD(111, 000) = 3

▪ Kôdna distanca (D_{min})
- minimalna Hamingova distanca između dve bilo koje reči

Ilustracija kôdne distance:



Detekcija greške

▪ Kôd može detektovati do d bitova greške ako je minimalna distanca kôda

$$D_{min} \geq d + 1$$

Korekcija greške

▪ Kôd može korektovati do c bitova greške ako je minimalna distanca kôda

$$D_{min} \geq 2c + 1$$

Kodovi parnosti

▪ parna parnost

ukupan broj jedinica u kodnoj reči je paran:
- 0110110 1
- 0001011 0

▪ neparna parnost

ukupan broj jedinica u kodnoj reči je neparan:

- 0110110 0
- 0001011 1

2.3. Pojam sindrom reč

Sindrom reč se dobija poređenjem dve K-bitne vrednosti bit po bit uzimanjem ekskluzivne disjunkcije. Sindorm reč je neoznačen ceo broj dužine K koji ima vrednost K izmedju 0 i 2^K-1.
Vrednost 0 označava da nema grešaka u zapisu, dok ostalih 2^K-1 vrednosti određuju mesto greške, u slučaju da greška postoji. Kako greška može da se javi na bilo kom od M bitova podatka i K bitova koji se koriste u proveri, mora da važi 2^K-1 ≥ M+K. Na osnovu ove nejednakosti možemo da odredimo potreban broj bitova kao funkciju dužine reči za koju se vrši provera.

Želja je da generisana sindrom rec ima sledece karakteristike:

• Ako su svi bitovi u njoj 0 tada nije otkrivena nikakva greška.
• Ako u reči postoji tačno jedna 1, tada greška postoji u jednom od 4 bita za proveru, dok je zapis podatka u redu i ne zahteva nikavu korekciju.
• Ako sindrom sadrži više od jednog bita koji ima vrednost 1, tada vrednost sindrom reči posmatrane kao ceo neoznačen broj određuje poziciju na kojoj se nalazi greška. Korekcija se sastoji u komplementiranju vrednosti odgovarajućeg bita podatka.

Da bi ove karakteristike i ostvarili uredimo bitove podatka i bitove za proveru u reč dužine 12 bita na način prikazan u tabeli:

Pozicija bita	Broj pozicije	Bit za proveru	Bit podatka
12	1 1 0 0		M8
11	1 0 1 1		M7
10	1 0 1 0		M6
9	1 0 0 1		M5
8	1 0 0 0	C4	
7	0 1 1 1		M4
6	0 1 1 0		M3
5	0 1 0 1		M2
4	0 1 0 0	C3	
3	0 0 1 1		M1
2	0 0 1 0	C2	
1	0 0 0 1	C1	

Pozicije bitova podataka i bitova za proveru

$C_1 = M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7$
$C_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7$
$C_3 = M_2 \oplus M_3 \oplus M_4 \oplus M_8$
$C_4 = M_5 \oplus M_6 \oplus M_7 \oplus M_8$

gde $\oplus$ označava operaciju ekskluzivne disjunkcije.

2.4. Hamingov kôd

Ako može više error correcting bita da bude ukljnučeno u poruku, i ako oni mogu da budu uredjeni tako da različiti pogrešni bitovi prouzrokuju različite greške, onda pogrešni bitovi mogu biti identifikovani. U 7-bitnom zapisu, moguće je sedam pojedinačnih grešaka, pa tri bita za kontrolu grešaka mogu javiti ne samo da postoji greška, već i na kom bitu se ona javila.
opšti algoritam:
1. sve pozicije bitova koje su stepen broja dva su parity bitovi (pozicije 1,2,4,8,16,32...)
2. Sve ostale pozicije bitova su za šifrovanje podataka
3. svaki parity bit računa parnost za neke od bitova u zapisu. Pozicija parity bita odredjuje sekvencu bitova koji se proveravaju i preskaču

- pozicija 1 (n=1): preskače 0 bit (0=n-1), proverava 1 bit (n), preskače 1 bit (n), proverava 1 bit (n), preskače 1 bit (n) itd
- pozicija 2 (n=2): preskače 1 bit (1=n-1), proverava 2 bita (n), preskače 2 bita (n), proverava 2 bita (n), preskače 2 bita (n) itd
- pozicija 4 (n=4): preskače 3 bita (3=n-1), proverava 4 bita (n), preskače 4 bita (n), proverava 4 bita (n), preskače 4 bita (n) itd
- pozicija 8 (n=8): preskače 7 bitova (7=n-1), proverava 8 bitova (n), preskače 8 bitova (n), proverava 8 bitova (n), preskače 8 bitova (n) itd
- pozicija 16 (n=16): preskače 15 bitova (15=n-1), proverava 16 bitova (n), preskače 16 bitova (n), proverava 16 bitova (n), preskače 16 bitova (n) itd
- pozicija 32 (n=32): preskače 31 bit (31=n-1), proverava 32 bita (n), preskače 32 bita (n), proverava 32 bita (n), preskače 32 bita (n) itd

Opšte pravilo za poziciju n: preskače n-1 bitova, proverava n bitova, preskače n bitova, proverava n bitova...
Drugim rečima parity bit na poziciji 2^k proverava bitove na pozicijama koje imaju bit k u svom binarnom obliku

Kodovi za korekciju grešaka koriste višestruke *parity* bitove koji su smešteni sa bitovima podataka. Svaki bit provere je *parity* bit grupe bitova u poruci. Prilikom čitanja primnjene poruke proveravaju se svi *parity* bitovi i ako su svi uredu, znači da nije došlo do greške u prenetoj poruci. Međutim ako je jedan ili više *parity* bitova neispravno, može se utvrditi koji je od bitova pogrešan. Jednostruka greška nastaje kada je vrednost jednog bita u poruci promenjena sa 1 na 0 ili sa 0 na 1. Ako je utvrđen pogrešan bit, može biti lako korigovan komplementiranjem njegove vrednosti. Jedan od najčešće korišćenih kodova sa korekcijom greške u RAM memorijama je Hamingov kod. Glavne odlike Hamingovog koda su:

- Detekcija greške je jednostavna (dodavanjem *parity* bitova)
- Korekcija greške je kompleksna i zahteva Hamingov kôd sa više *parity* bitova

U Hamingovom kodu, p - *parity* bitova je dodato na m - bitova u poruci, tako da imamo novonastalu kodnu reč on m+p - bitova. Bitske pozicije su numerisane u sekvenci od 1 do m+p. Sve pozicije u reči, čiji je broj stepena 2 (pozicije 1,2,4,8,...) koriste se za *parity* bitove. Ostale pozicije su namenjene za bitove originalne poruke. Izgled kodovane poruke, za originalnu poruku od 8-bitova:

d_7	d_6	d_5	d_4	p_3	d_3	d_2	d_1	p_2	d_0	p_1	p_0

pozicije: 12 1

Vidimo da je originalnoj 8-bitnoj poruci dodato 4 *parity* bita tako da sad sekvenca ima 12 bitova.

Prilikom izbora broja *parity* bitova mora biti zadovoljena sledeća relacija:

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

na primer: za 8-bitnu poruku poruka potrebna su nam 4 *parity* bita, tako da će podatak koji prenosimo sadržati ukupno 12 bitova.

- *parity* bitovi su 1,2,4,8, ... (P1,P2,P3,P4)

- 3 = 1+2
- 5 = 1+4
- 6 = 2+4 P1 proverava bitove 3, 5, 7, 9, 11
- 7 = 1+2+4 P2 proverava bitove 3,6,7,10,11
- 9 = 1+8 P4 proverava bitove 5,6,7,12
- 10 =2+8 P8 proverava bitove 9,10,11,12
- 11 = 1+2+8
- 12 = 4+8

Tako da vrednosti P1,P2,P4 i P8 računamo kao:

P1 = XOR nad bitovima (3,5,7,9,11)
P2 = XOR nad bitovima (3,6,7,10,11)
P4 = XOR nad bitovima (5,6,7,12)
P8 = XOR nad bitovima (9,10,11,12)

Korišćena ekskluzivna OR operacija ustvari vrši neparnu funkciju. Ona je jednaka 1 kada imamo neparan broj jedinica između posmatranih promenjivih, a jednaka 0 kada imamo paran broj jedinica. To nam obezbeđuje da je broj jedinica na proveravanim pozicijama uključujući i bitove parnosti, uvek paran.

Na mestu prijema kompozitna reč od 12 bitova se ponovo proverava na grešku. Parnost reči se proverava nad istim grupama bitova u reči uključujući i bitove parnosti. Tako da će mo sad imati četiri bita provere:

C1 = XOR nad bitovima (1,3,5,7,9,11)
C2 = XOR nad bitovima (2,3,6,7,10,11)
C4 = XOR nad bitovima (4,5,6,7,12)
C8 = XOR nad bitovima (8,9,10,11,12)

Dobijena vrednost 0 za proverene bitove ukazuje na parnu parnost, a dobijena vrednost 1 na neparnu parnost. Pošto su bitovi zapisani sa parnom parnosti, rezultat C=C8C4C2C1=0000, ukazuje da nije došlo do greške pri prenosu. Uopšteno, za $C \neq 0$, 4-bitni binarni broj ukazuje na poziciju bita gde je nastala greška, ako se radi o jednostrukoj grešci. Tako recimo ako dobijemo da je C=0101, znači da je došlo do greške u petom bitu i taj bit treba komplementirati.

Hamingov kod se može koristiti za kodnu reč proizvoljne dužine. Generalno za p - *parity* bitova i originalnu reč od m - bitova imamo ukupno p+m bitova koji mogu biti kodovani sa 2^k-1 bitova. Grupe bitova za generisanje *parity* bitova i bitova provere na prijemu određuju se iz liste binarnih brojeva iz liste od 1 do 2^k-1. Svaka grupa bitova počinje sa brojem koji je stepena 2, na primer 1,2,4,8,16 itd. Ovi brojevi su takođe i pozicije *parity* bitova.

Osnovni Haming kod može detektovati i korigovati greške u samo jednom bitu. Upotrebom dodatnih *parity* bitova mogu se korigovati jednostruke greške i detektovati dvostruke. ako posmatramo kodiranu 12-bitnu reč i dodamo još jedan *parity* bit. Ako recimo imamo reč 001110010100P_{13}, gde je P_{13} rezultat ekskluzivne OR operacije nad ostalih 12 bitova. Ovako nastaje 13-bitna kodna reč 0011100101001 (parna parnost). Na prijemu se nalaze bitovi provere i *parity* bit P za svih 13 bitova kodne reči. Ako je P=0 znači da je parnost korektna (parna parnost), ali ako je P=1, parnost nad svih 13 bitova je nekorektna (neparna parnost). Iz ovoga proizilaze sledeće četiri situacije:

- ako je $C = 0$ i $P = 0$ nije prisutna greška

- ako je $C \neq 0$ i $P = 1$ detektovana je jednostruka greška koja može biti korigovana

- ako je $C \neq 0$ i $P = 0$ detektovana je dvostruka greška koja ne može biti korigovana

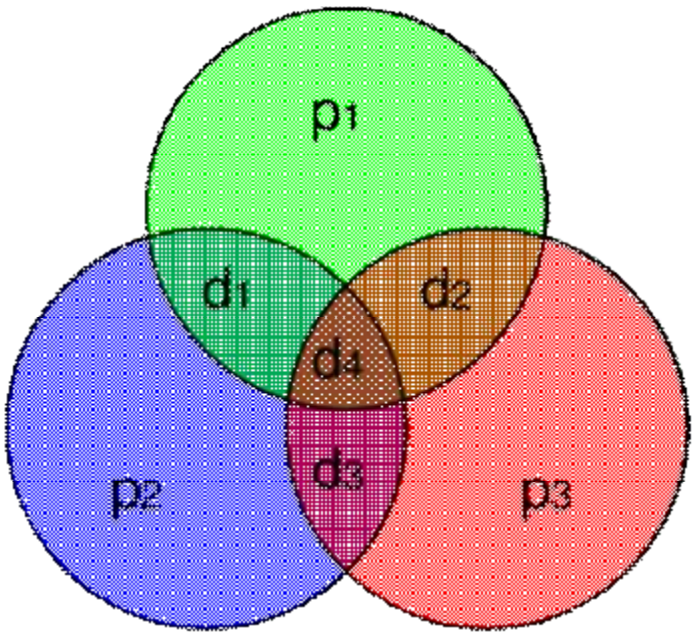
- ako je $C = 0$ i $P=1$ greška je nastala u P_{13} bitu

2.4.1.Hamingov kod sa dodatnim parity bitom

Hamingov kod ima minimalnu Hamingovu distasncu 3, što znači da kod može detektovati i otkloniti pojedinačnu grešku. Uključivanjem dodatnog parity bita moguće je povećati distancu na 4. Ovo omogućava kodu da može da detektuje i otkloni pojedinačnu grešku, i istovremeno da detektuje ali ne i da otkloni dvostruku grešku. Takođe se može koristiti i za detektovanje 3 greške ali ne može da koriguje nijednu.
Znači, ako je minimalna distanca bilo kog koda za korekciju grešaka data sa W, onda W*(W/2) grešaka je moguće detektovati, od kojih je ((W-1)/2) moguće korigovati.

2.4.2.Haming (7,4) kod

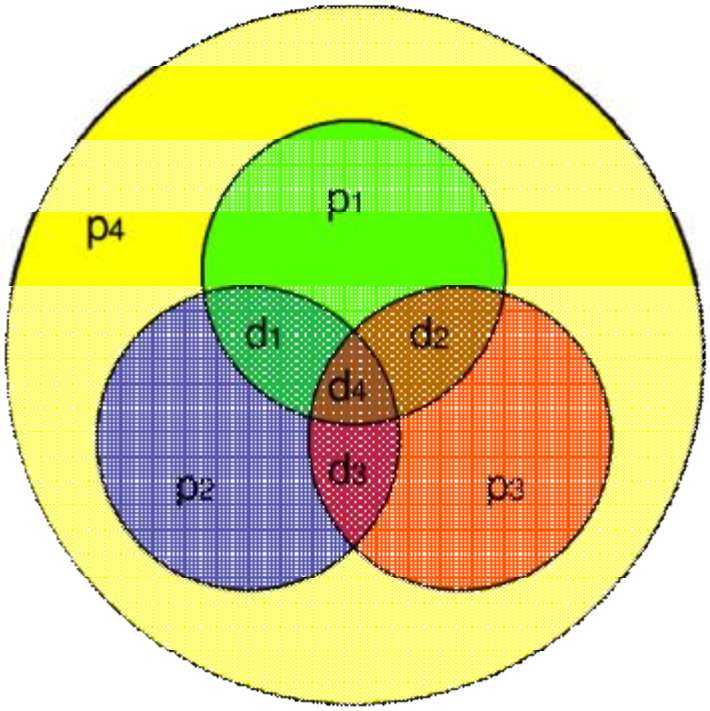
1950 Haming je predstavio (7,4) kod. On šifruje 4 bita zapisa u 7 bitova dodavanjem 3 parity bita. On može da detektuje i otkloni pojedinačnu grešku, i da detektuje dvostruku grešku.
Na primer 1011 se šifruje u 0110011 gde su crvene cifre parity bitovi.



grafički prikaz 4 bita informacije i 3 parity bita i koji parity bit dodeljuje kom bitu podatka

2.4.3. Haming (8,4) kod

Hamingov (7,4) s elako može proširiti u (8,4) dodavanjem extra parity bita na vrh (7,4) šifrovane reči.
Na primer 1011 se šifruje u 01100110 gde su crvene cifre parity iz koda (7,4), a zelena cifra je Parity dodat Hamingovim (8,4) kodom. Zelena cifra uzrokuje da (7,4) kod bude paran.



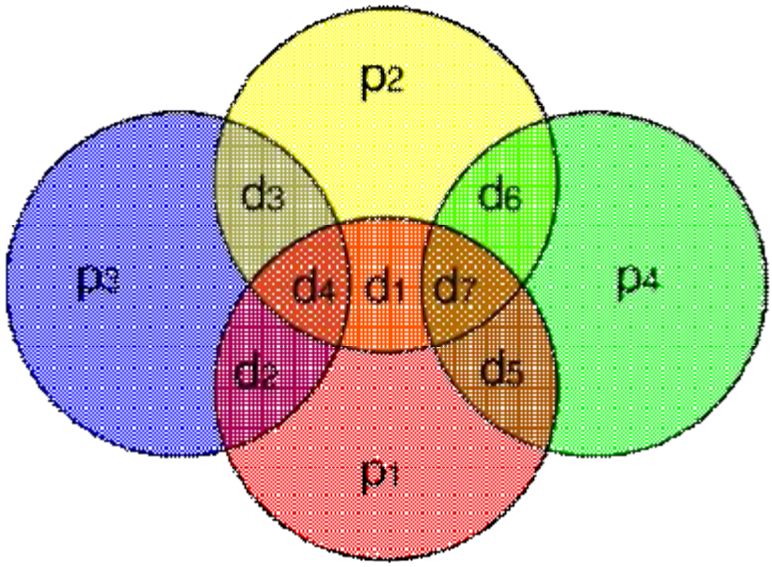
primer sa prethodne slike sa dodatnim parity bitom

2.4.4. Haming (11,7) kod

Posmatramo reč od 7 bitova 0110101. Da bi pokazali kako se Hamingov kod koristi da bi detektovao grešku, pogledaćemo tabele.

- d označava podatak i p parity bit

Prvo se podaci ubacuju na odgovarajuća mesta, a parity bitovi se računaju za svaki pojedinačno koristeći parnost (da budu deljivi sa dva)
Slika pokazuje koji od parity bitova pokriva koji bit informacije



prikaz 7 bitova podataka i 4 parity bita i koji parity bit se dodeljuje kom bitu iz zapisa

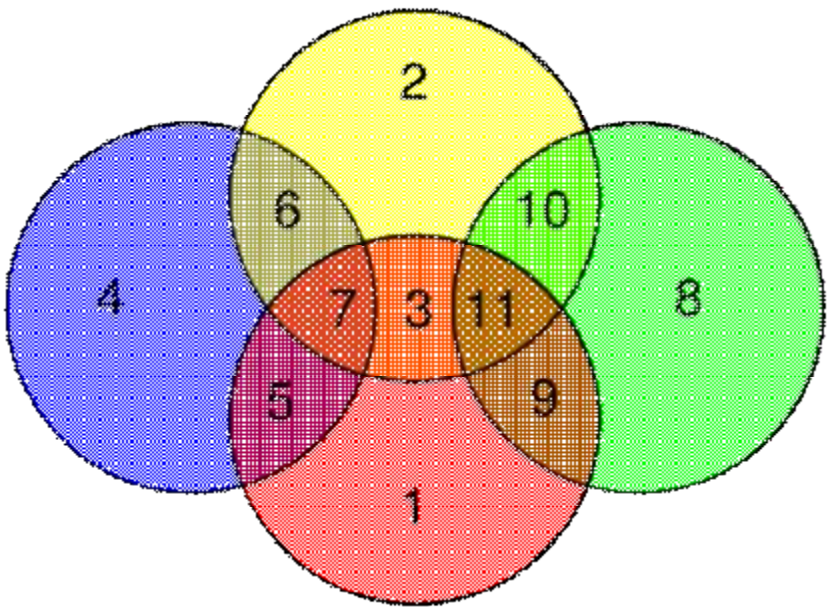
	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7
Data word (bez parity bitova):			0		1	1	0		1	0	1
p_1	1		0		1		0		1		1
p_2		0	0			1	0			0	1
p_3				0	1	1	0				
p_4								0	1	0	1
Data word (sa parity bitovima):	1	0	0	0	1	1	0	0	1	0	

računanje parity bitova Hamingovog koda

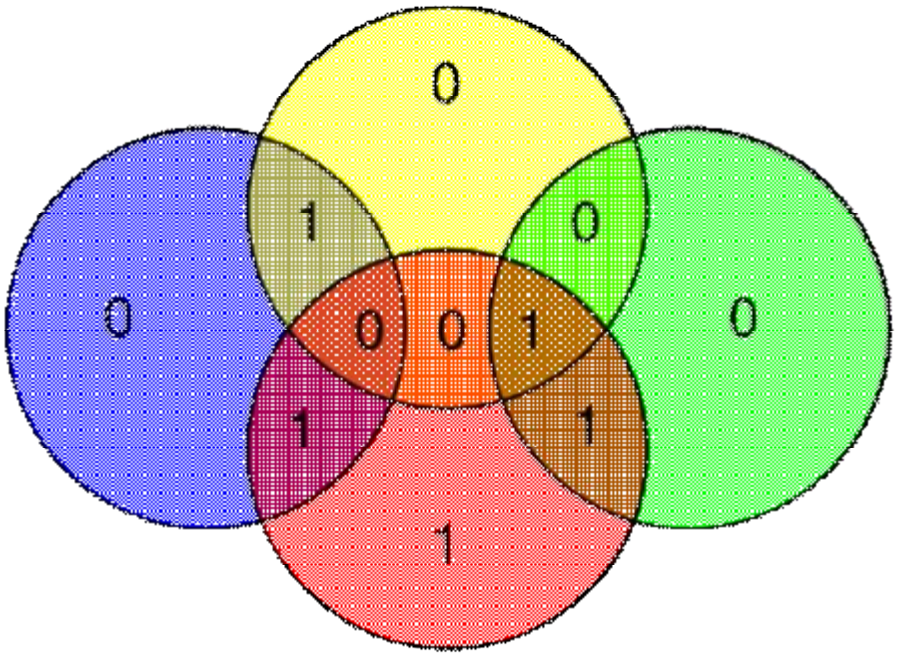
Nova reč, sa parity bitovima, je sada 10001100101. Sada uzimamo da je zadnji bit "pokvaren" i promenjen iz 1 u 0. Nova reč je 10001100100 i sada kada anliziramo kako je Hamingov kod napravljen obojićemo svaki parity bit koji je 1 (kada je provera parnosti negativna).

	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	Parity provera	Parity bit
Primlojena data word:	1	0	0	0	1	1	0	0	1	0	0		
p_1	1		0		1		0		1		0	Fail	1
p_2		0	0			1	0			0	0	Fail	1
p_3				0	1	1	0					Pass	0
p_4								0	1	0	0	Fail	1

provera parity bitova



promenjena pozicija zapisa i parity bitova

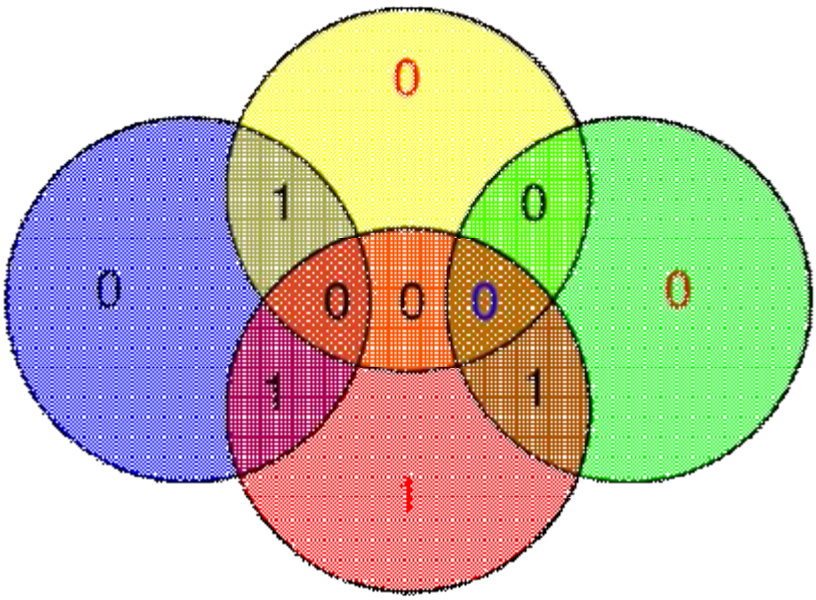


vrednost zapisa iz primera. Vrednost u crvenom, žutom, zelenom i plavom krugu je parna

Zadnji korak je odrediti vrednost parity bitova (imati u vidu da je bit sa najmanjim indexom određuje vrednost jedinica i ide skroz desno). Celobrojna vrednost parity bita je 11, što znači da jedanaesti bit u reči (uključujući i parity bitove) je pogrešan i mora biti zamenjen.

	p_4	p_3	p_2	p_1	
Binary	1	0	1	1	
Decimal	8		2	1	$\Sigma = 11$

Zamenom jedanaestog bita promenjena reč 10001100100 postaje opet 10001100101. Uklanjanjem Hamingovog koda dobija se originalna reč 0110101.



Greška na jedanaestom bitu uzorkuje pogrešnu parnost u crvenom, žutom i zelenom krugu

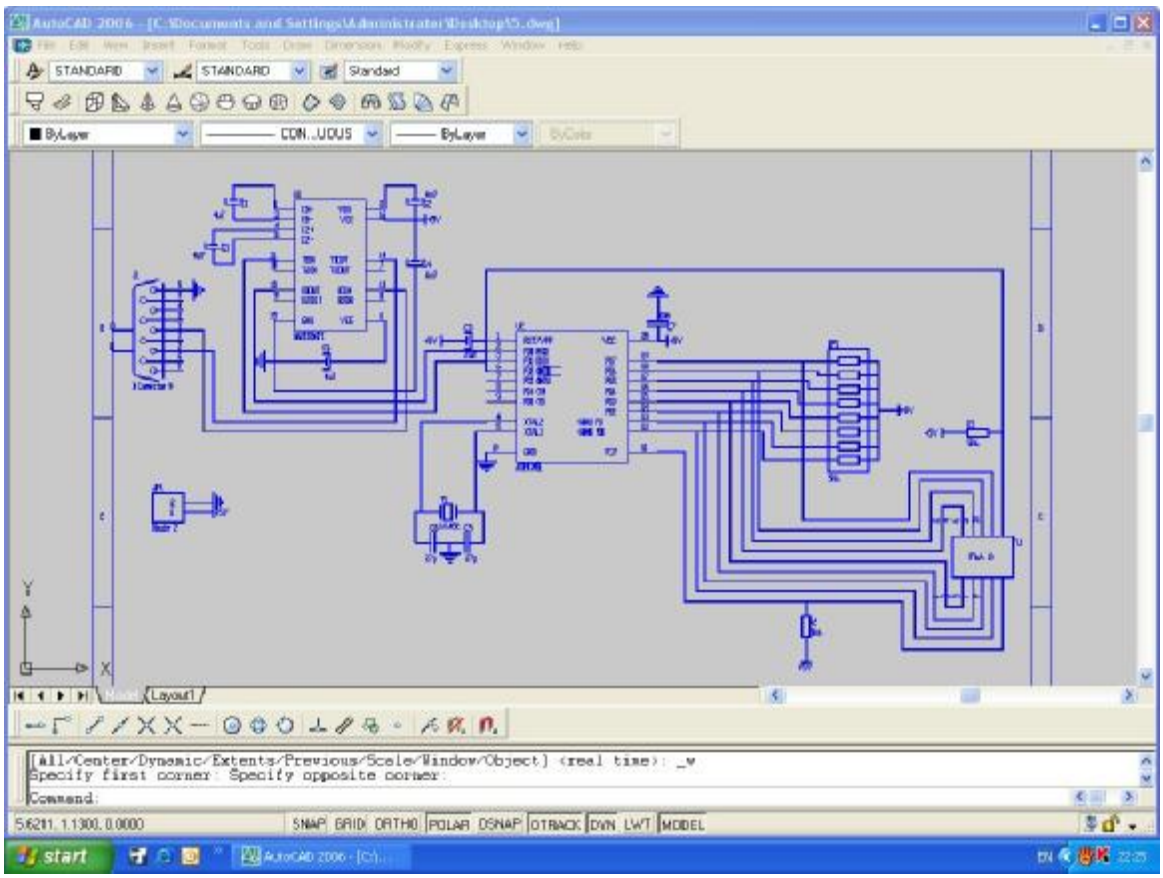
Važno je znati da kako parity bitovi ne proveravaju jedni druge, ako provera nekog parity bita ne uspe, a ostali uspeju, onda je greška u parity bitu a ne u nekom bitu iz zapisa.

3. Zadatak

Mikrokontroler dobija podatke iz PC-a i prenosi ih do drugog mikrokontrolera koristeći Hamingov kod

Koriste se procesori AT89C2051

Šema



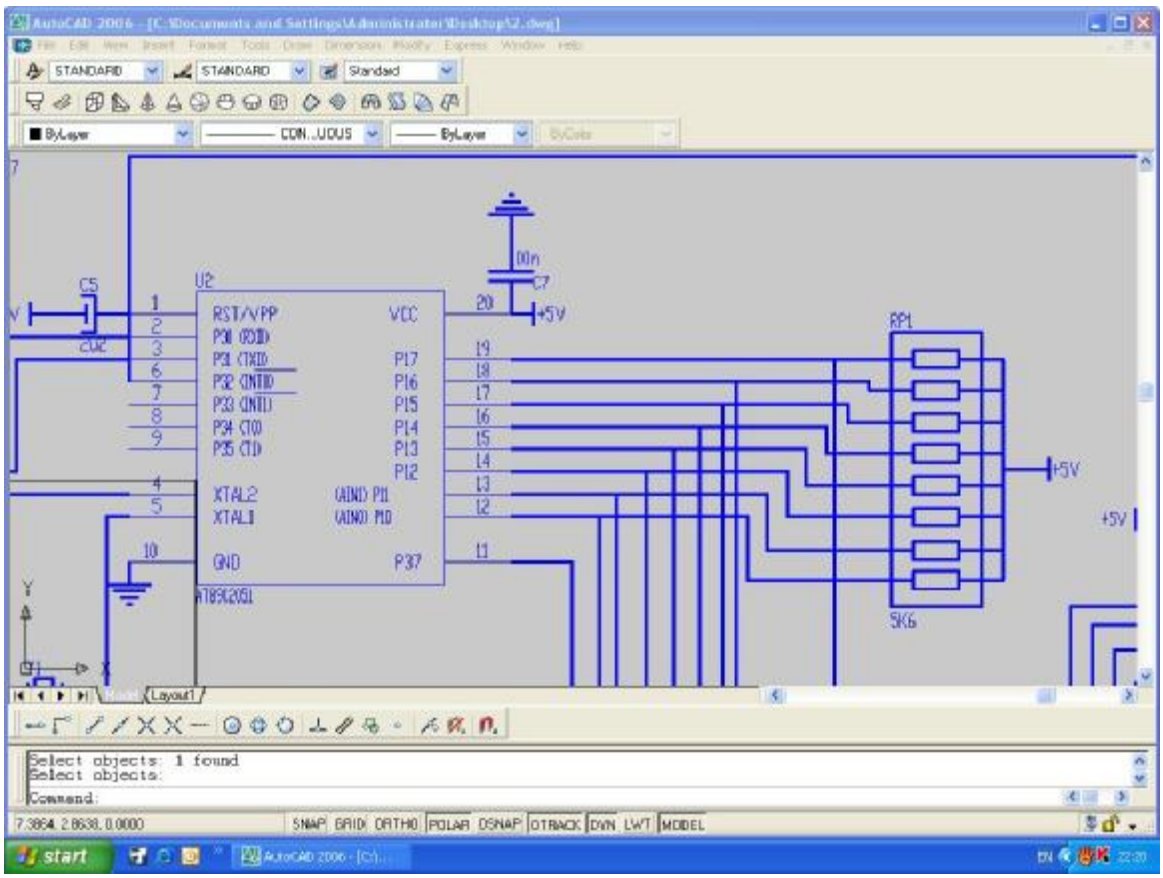
1. pin - za reset, vezan kondenzator ako blokira da se resetuje
2. pin - RX služi za komunikaciju sa RS232 portom
3. pin - TX služi za komunikaciju sa RS232 portom
4. pin - za njega i za 5. pin je vezan kristal koji daje takt procesoru
10. pin - za masu (logička 0)
20. pin - (zadnji), za napajanje - 5V(na njega ide logička jedinica = 5V) i na njega se vezuje kondenzator koji otklanja smetnje u jednosmernom režimu
Od 12. do 19. pina je port 1 (kada se prime podaci iz PC-a postave ih na port i pošalju drugom procesoru)
6. pin - javi drugom procesoru da su spremni podaci za slanje
11. pin - služi da drugi procesor javi prvom da je preuzeo podatke i da može da šalje dalje

3.1. Handshake procedura

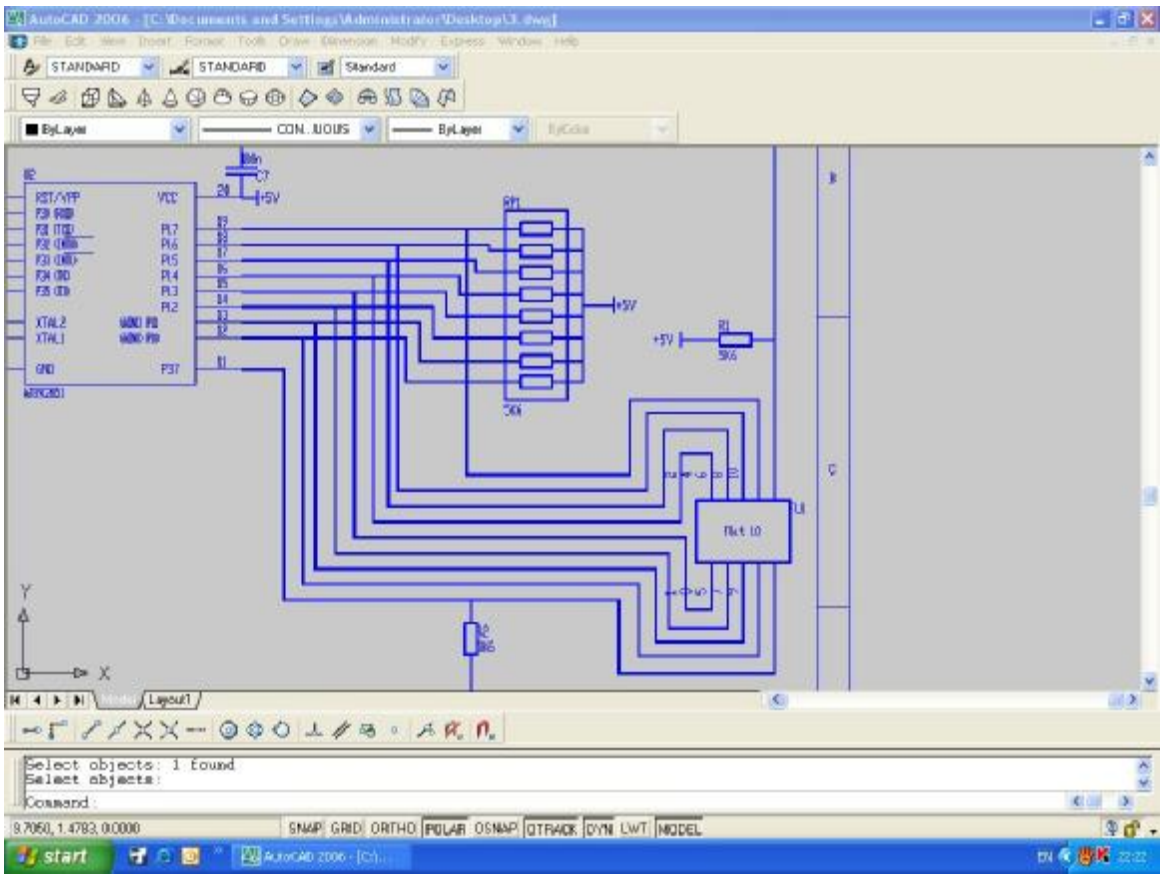
Radnje koje obavljaju šesti i jedanaesti pin su poznate kao *handshakeing tehnika* prenosa. To je tehnka pomoću koje se ostvaruje asinhrono prenos informacije između dve komponente A i B.
Redosled događaja je sledeći:

1. komponenta A postavlja važeće podatke na linije za podatke i aktivira liniju Spreman (ready) čije stanje ukazuje da su podaci na linijama za podatke važeći
2. prijeman komponenta B testira stanje na liniji Spreman. Kada ustanovi da su podaci važeći prihvata podatke i nakon toga aktivira signal Potvrda pomoću koga obaveštava predajnik da će prihvatiti podatke
3. prijeminik kada prihvati podatke deaktivira signal Potvrda, tj. Obaveštava predajnik da je primio podatke
4. predajnik testira stanje na liniji Potvrda i kada ono postane neaktivno, što znači da su podaci od strane prijemnika prihvaćeni, deaktvira liniju Spreman i ukida podatke na linijama za podatke. Nakon ovog trenutka predajnik može da aktivira novi ciklus prenosa podataka

Linije za podatke i upravljanje, pomoću kojih se povezuju dve komponente, kao i signalizaciona sekvenca, koja je potrebna da bi se uspešno obavilo komuniciranje između dva uređaja, čine interfejs uređaja. Što su interfejsi srodniji komunikacija između dve komponente se lakše ostvaruje.



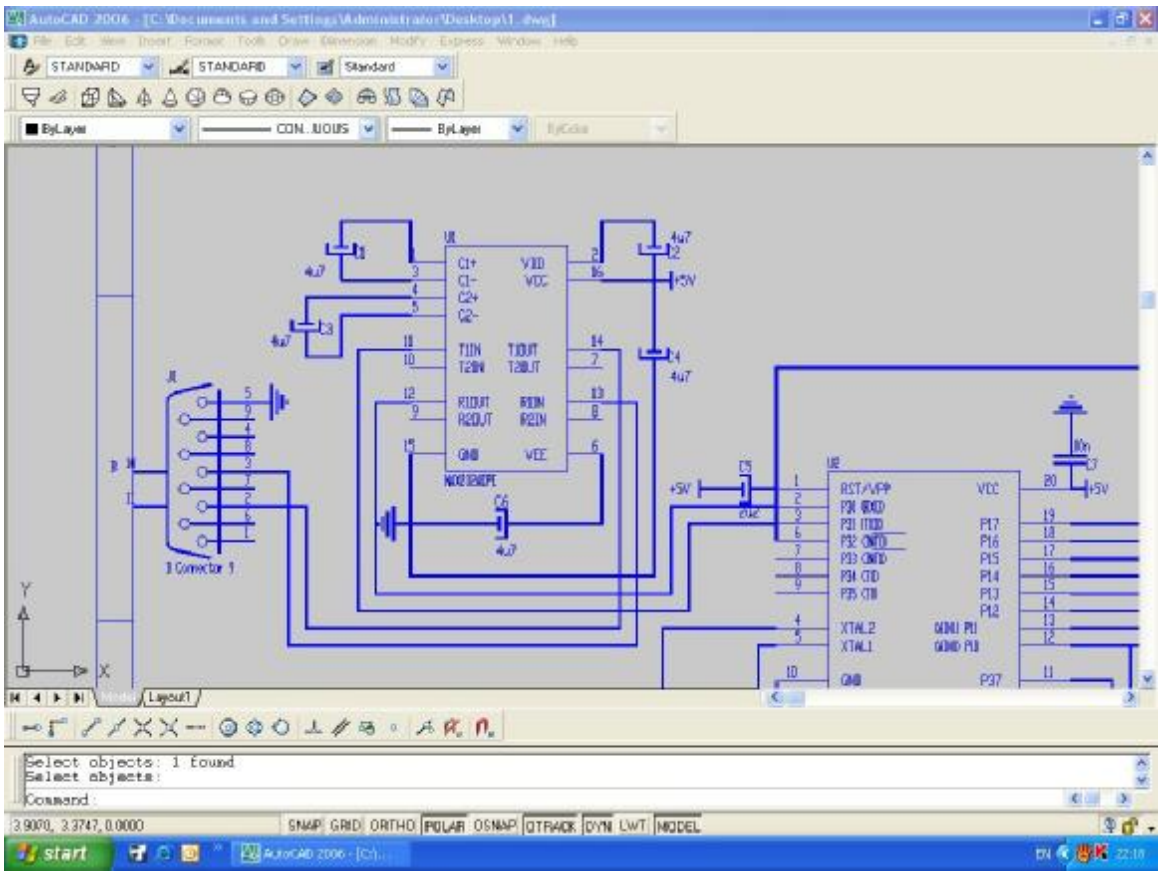
Između dva procesora se stavljaju pull up otpornici prema 5V, na otpornoj letvici zato što ih procesori nemaju integrisane u sebi a služe da neutrališu smetnje i važnije da definišu logičku jedinicu (uvek je drže na 5V)

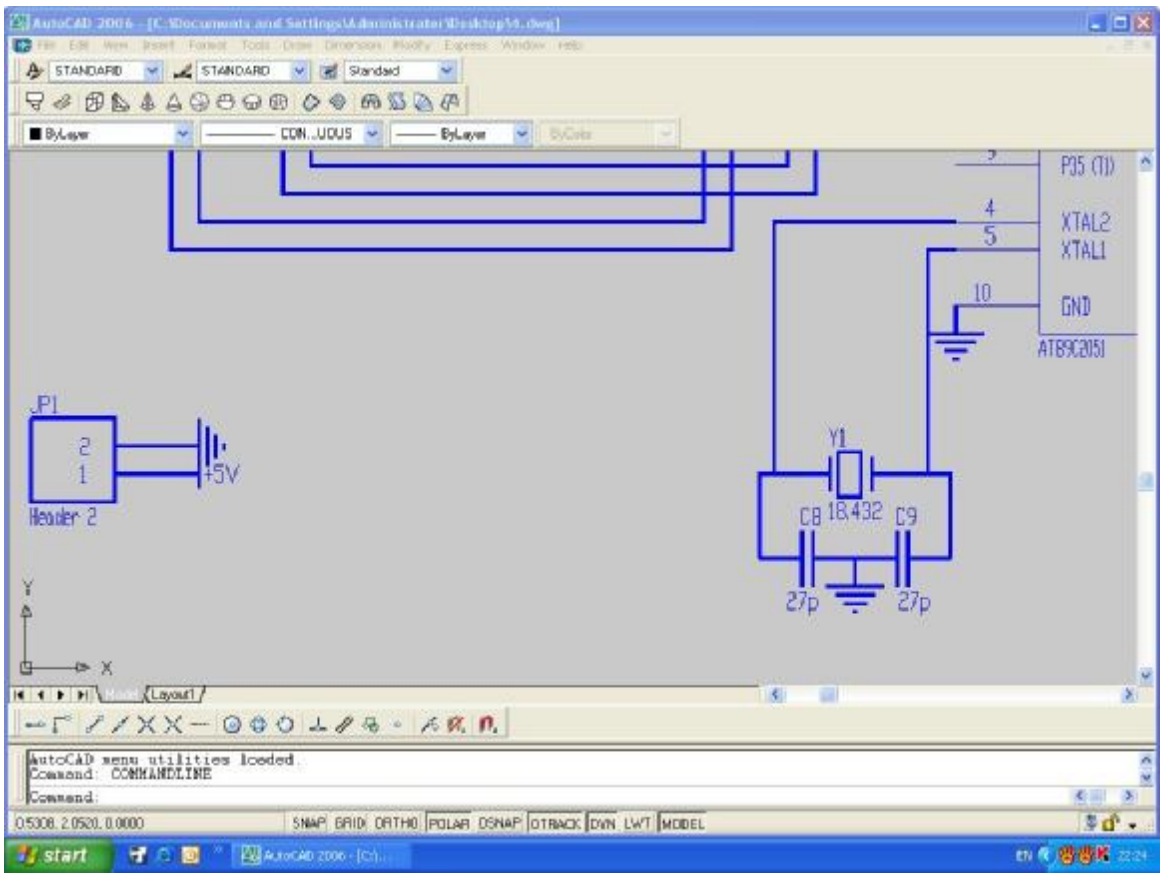


dve pločice su povezane 10-pinskim flat konektorom (na svakoj pločici je po jedan procesor)

Podaci iz PC-a preko 9-pinskog (subD-9) konektora dolaze na prvu pločicu
između PC-a i procesora ide MAX232 koji služi za konverziju logičke 0 u 1 (kod procesora 0 je 0V a 1 je 5V, a kod PC-a 0 je 12V a 1 je 12V ili 0V)
MAX232 se uvek vezuje sa 5 kondenzatora za komunikaciju

na konektoru se koriste pinovi:
2 - RX (ide na TX procesora)
3 - TX
5 - 0V


STANDARD
STANDARD
Standard
ByLayer
CONTINUOUS
ByLayer
U1
MAX232DIPE
C1+
C1-
C2+
C2-
VDD
VCC
T1IN
T2IN
T1OUT
T2OUT
R1OUT
R2OUT
R1IN
R2IN
GND
VCC
4u7
+5V
J1
DB9 Connector F
U2
RST/VPP
VCC
XTAL2
XTAL1
GND
P37
+5V
Select objects: 1 found
Select objects:
Command:
SNAP GRID ORTHO POLAR OSNAP OTRACK DYN LWT MODEL
start


STANDARD
STANDARD
Standard
ByLayer
CONTINUOUS
ByLayer
P35 (T1)
XTAL2
XTAL1
GND
AT89C2051
JP1
Header 2
+5V
Y1
C8
18.432
C9
27p
27p
Layout1
AutoCAD menu utilities loaded.
Command: COMMANDLINE
Command:
SNAP GRID ORTHO POLAR OSNAP OTRACK DYN LWT MODEL
start

3.2. Opis procesa

Prvi mikrokontroler (predajnik) dobija određenu poruku iz PC-a dužine 4 bajta. Ukoliko dobije dužu poruku, podeliće je na više poruka dužine 4 bajta. Ako ostane neki bajt viška, ili sama uneta poruka sadrži manje od 4 bajta, čeka se dalji unos.
Prva dva bajta unete poruke su podaci, treći određuje koji od njih ima simuliranu grešku (vrednosti su od 0 do 3), dok četvrti bajt označava bit na kome će se simulirati greška (vrednosti od 0 do 7).
Predajnik od prva dva bajta pravi četiri i onda simulira grešku (promeni bit u bajtu kako mu je zadato). Tako formiranu poruku prenosi do drugog mikrokontrolera (prijemnik). Kada prijemnik dobije poruku, proverava sva četiri primnljena bajta i ako postoji greška koriguje je. Nakon toga, četiri bajta vrati na dva bajta podataka. Tako obrađenu poruku šalje PC-u. Pored toga šalje i poruku u obliku u kome je primio (sa greškom).

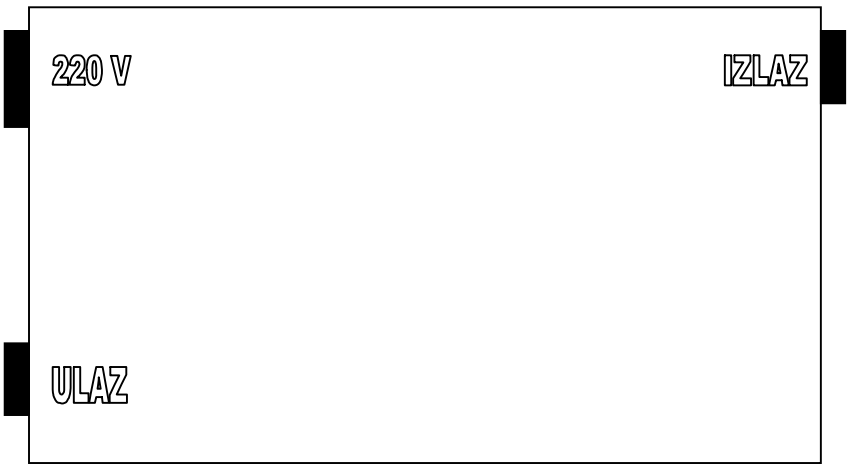
4. Laboratorijska vežba za studente

1. Cilj ove vežbe je upoznavanje studenata sa mogućnošću detektovanja i otkalnjanja greške prilikom prenosa podataka primenom Hamingovog koda. Radi pripreme za ovu vežbu neophodno je upoznati se sa prethodnom dokumentacijom.

2. Za realizaciju ove vežbe potrebno je sledeće:
- PC koji ima dva serijska porta, najmanje 256 MB RAM memorije i minimum operativni sistem Windows 98;
- uređaj koji je namenjen za ovu vežbu (u kome se nalaze dva mikrokontrolera koji poruke izmedju sebe prenose uz pomoć kodiranja Hamingovim kodom);
- dva kabla za konekciju sa serijskim portom
- kabl za napajanje.

Na PC-u je neophodno da bude instaliran program za komunikaciju preko serijskog (RS232) porta – *Terminal v1*.

3. Uređaj se poveže sa računarom pomoću kablova za RS232 komunikaciju. Treba imati u vidu da se ulaz, odnosno mikrokontroler koji prima podatke iz PC-a (predajnik), nalazi pored napajanja; a izlaz, odnosno mikrokontroler koji predaje korigovane podatke računaru (prijemnik), nalazi naspram napajanja na uređaju što je prikazano ispod na slici 1.

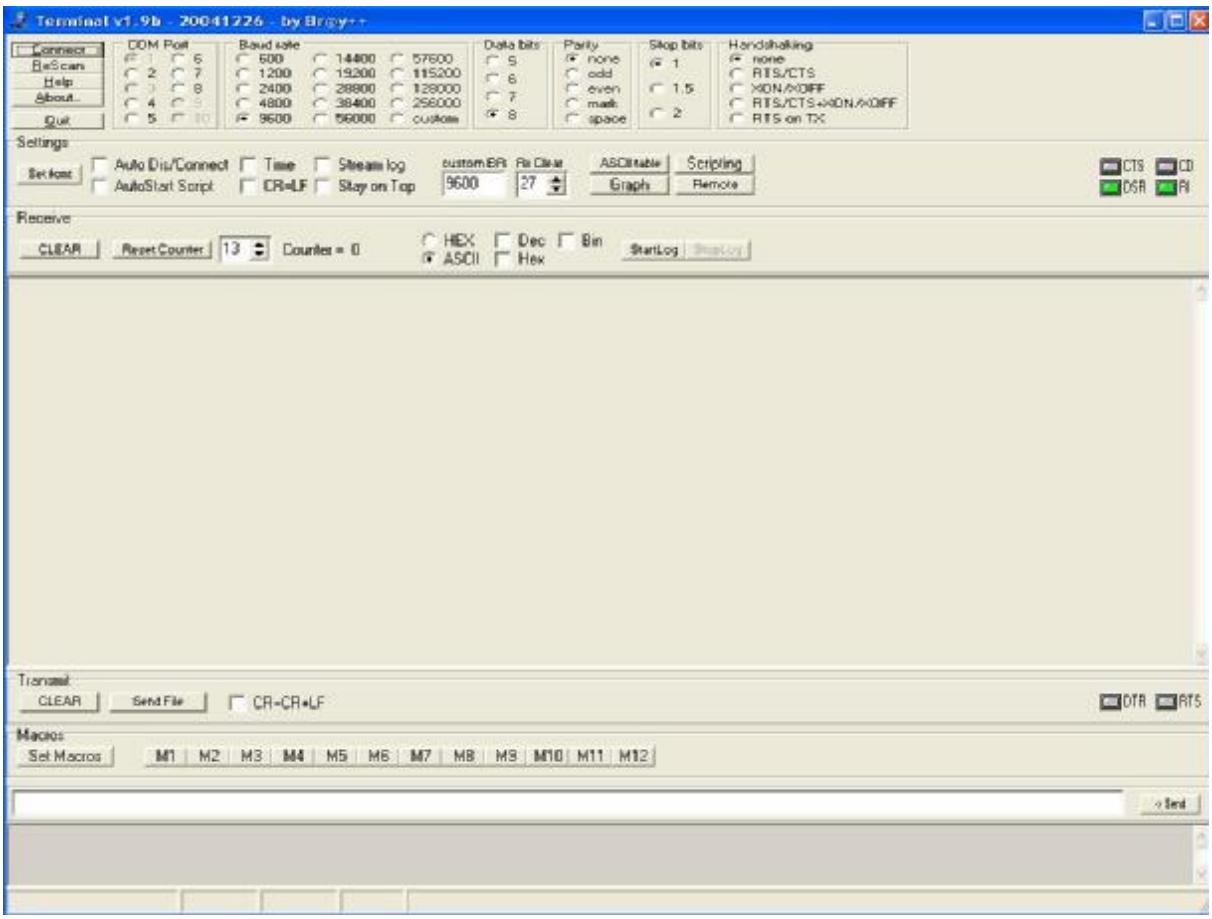


Slika 1 – šema uređaja

Nakon toga se uređaj priključi na napajanje.

4. Aplikacija se startuje klikom na ikonicu programa *Terminal v1*.

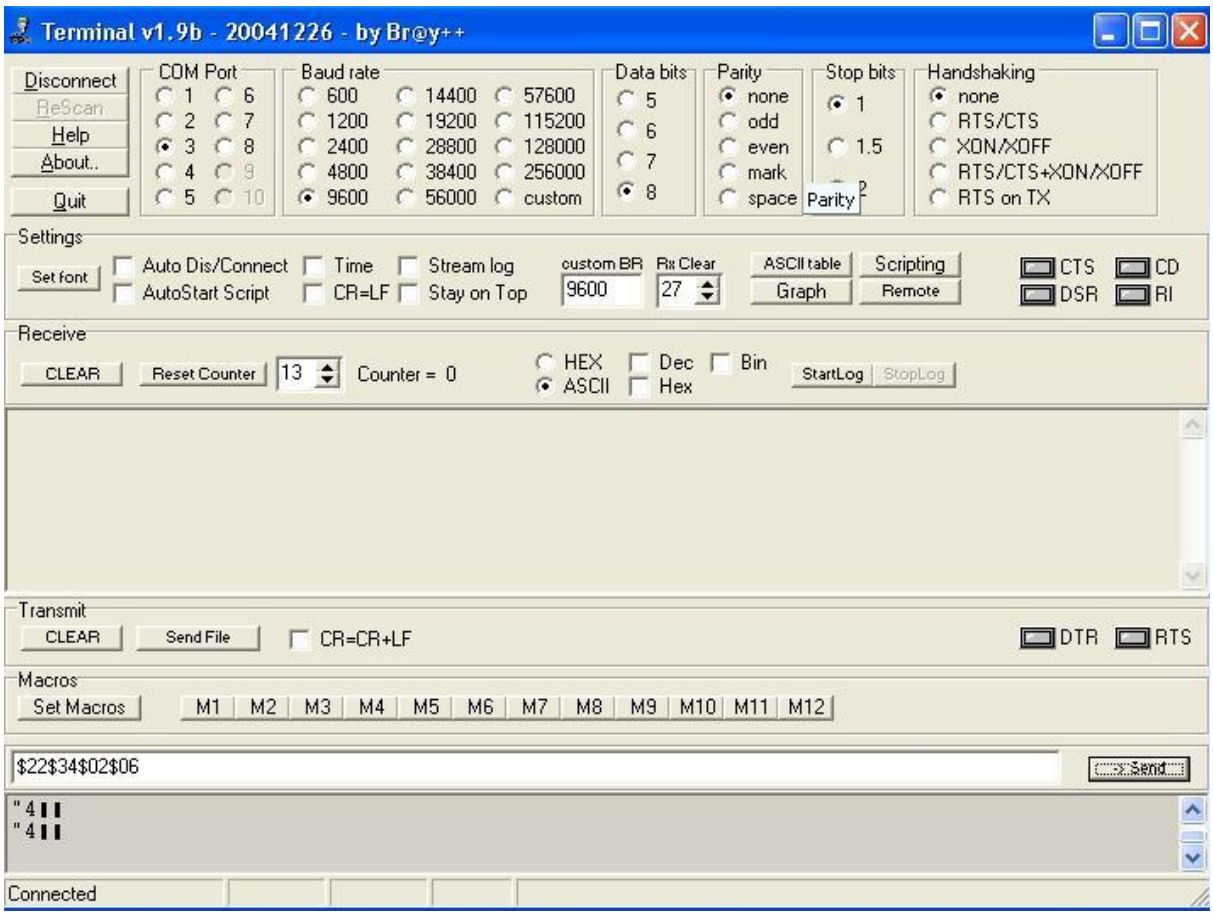
Ovo je potrebno uraditi dva puta, da bi na ekranu imali dva prozora, jedan preko koga predajemo podatke prvom mikrokontroleru i drugi koji nam prikazuje rezultate koje računaru šalje drugi mikrokontroler. Izgled otvorenog prozora prikazan je na slici 2.



Slika 2 – izgled otvorenih prozora

5. U programu *Terminal* je potrebno podesiti parametre za komunikaciju na sledeći način. Za prijem podataka se prvo odabere COM port u zavisnosti od toga koji se port koristi na računaru. U drugom prozoru za *Receive* podesi se i prijem na HEX (da bi primljeni zapis bio heksadecimalni) i nakon toga se klikne na komandu *Connect*. Za predaju podataka se odabere COM port i klikne se takođe na *Connect*. Nakon ovoga ostvarena je konekcija i ka prijemnom i ka predajnom mikrokontroleru.

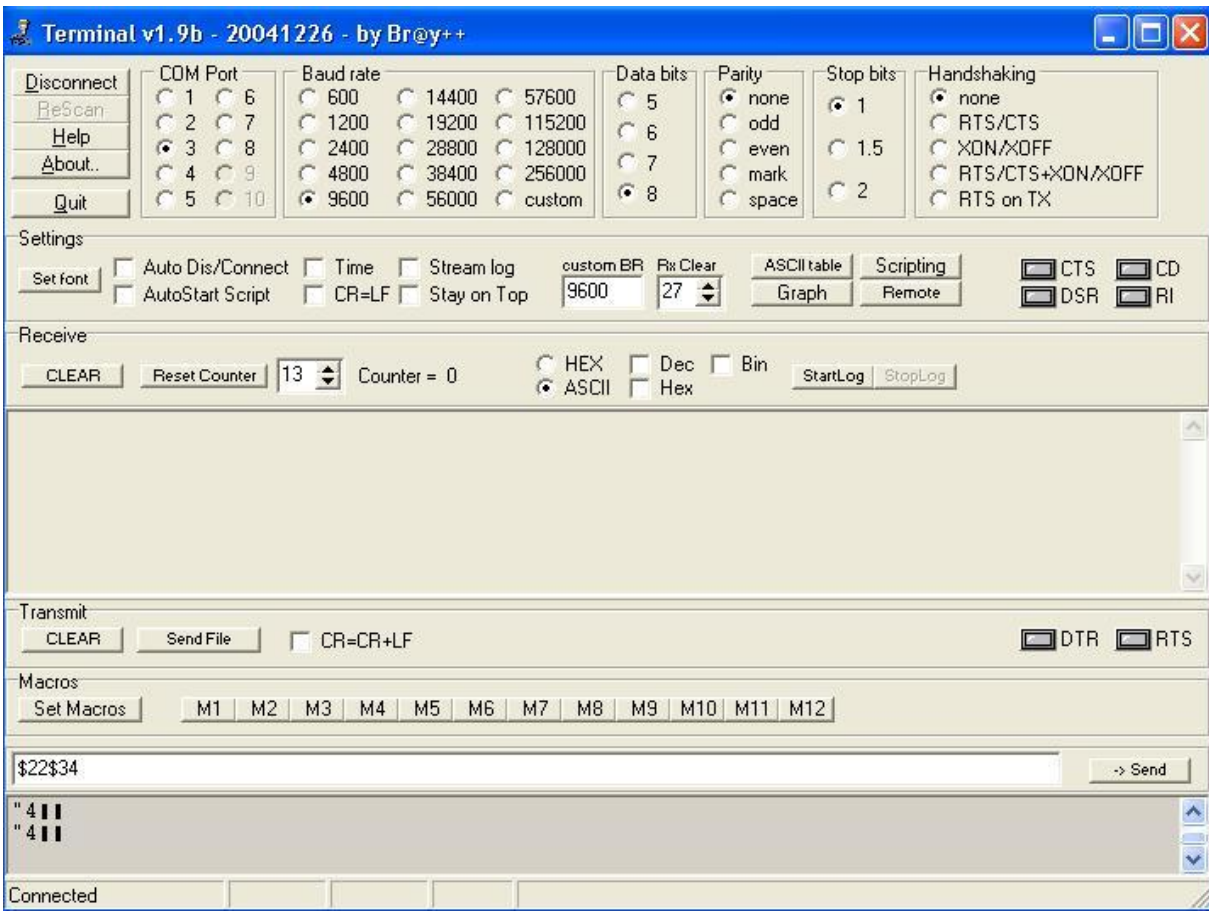
6. Nakon ovoga potrebno je predati poruku iz računara prvom mikrokontroleru (predajniku). To se realizuje na sledeći način: u otvorenom prozoru u prazno polje levo od komande *Send* se unosi poruka dužine četiri bajta u heksadecimalnom zapisu, npr $22$34$02$06 što se može videti na slici 3. Ovde znak $ označava da je u pitanju heksadecimalna vrednost bajta. Ukoliko se unese duža poruka, ona će biti podeljena na više njih dužine četiri bajta koje će biti pojedinačno obrađene. Ako ostane neki bajt viška, ili sama uneta poruka sadrži manje od četiri bajta, uređaj čeka dalji unos.



Slika 3 - izgled prozor sa unetom porukom

7. Simulaciju greške pravi sam student prilikom unosa poruke na računaru. Greška se simulira pomoću trećeg i četvrtog bajta unete poruke. Treći bajt određuje koji bajt ima simuliranu grešku (moguće vrednosti su od $00 do $03), a četvrti bajt je oznaka bita na kome se simulira greška (moguće vrednosti su od $00 do$07).

Znači da u unetoj poruci iz primera grešku određuju $02 i $06, a podaci su $22 i $34 kao što se vidi na slici 4 ispod.

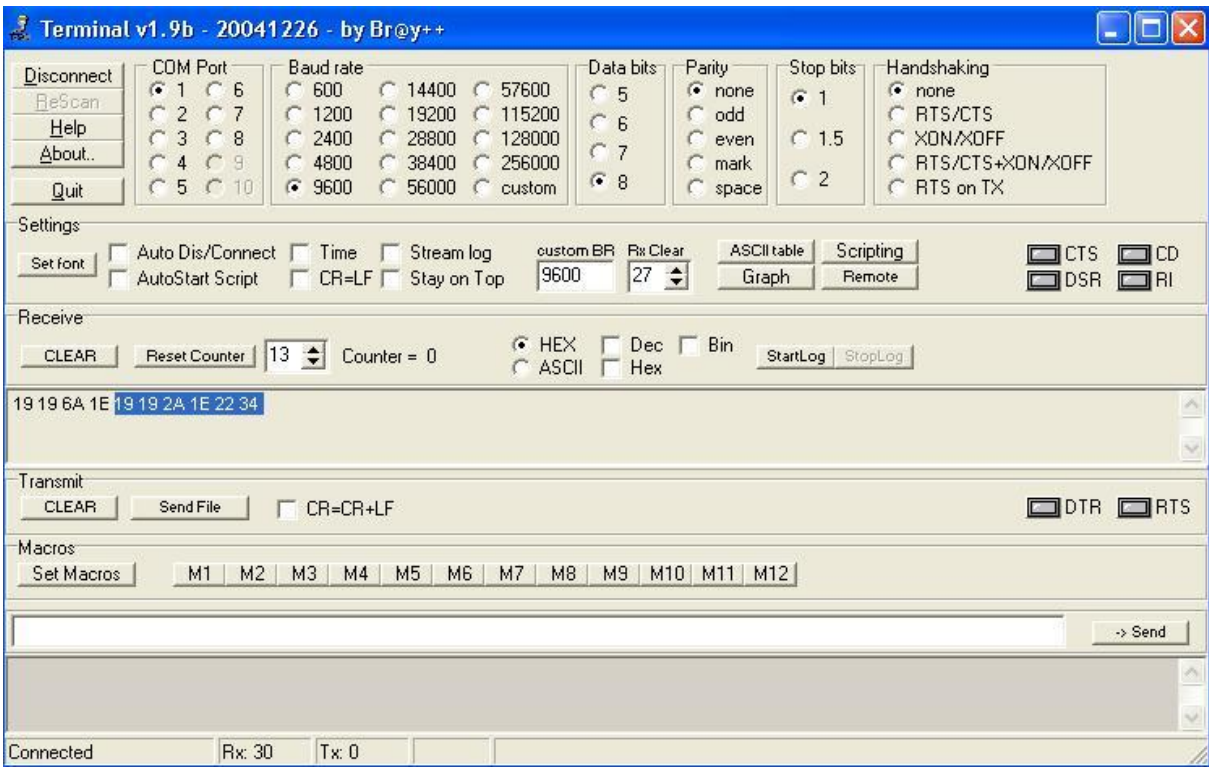


Slika 4 - unet zapis bez bajtova koji određuju simulaciju greške

8. Nakon unosa poruke ona se šalje prvom mikrokontroleru pomoću komande *Send*

Prvi mikorkontroler od prva dva bajta poruke napravi četiri dodavanjem dodatnih bitova Hamingovog koda.

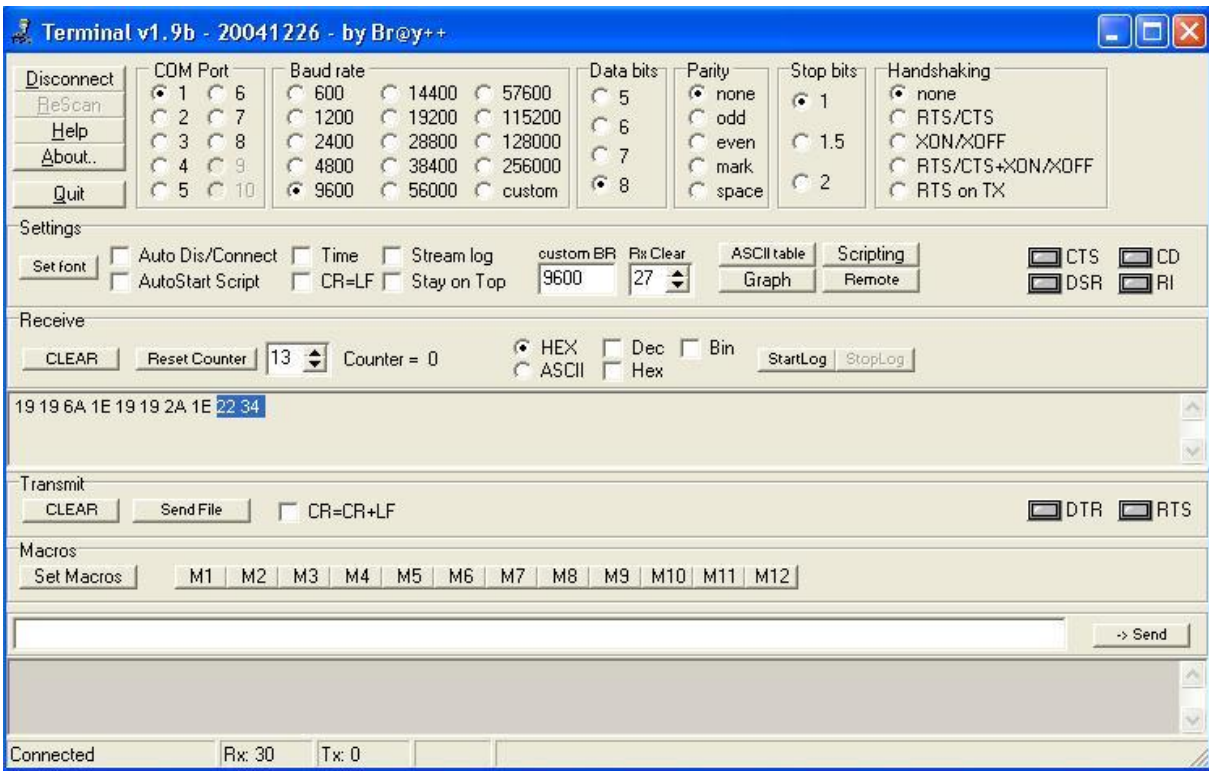
Na njima simulira grešku pomoću trećeg i četvrtog unetog bita. Za dati primer od bajtova $22 i $34 formirana su četiri bajta 19 19 6A 1E. U ovoj reči greška je kako je i zadato u bajtu 6A. Ovo se vidi na drugom *Receive* prozoru I prikazano je na slici 5.



Slika 5 - prikaz simulirane greške na receive prozoru

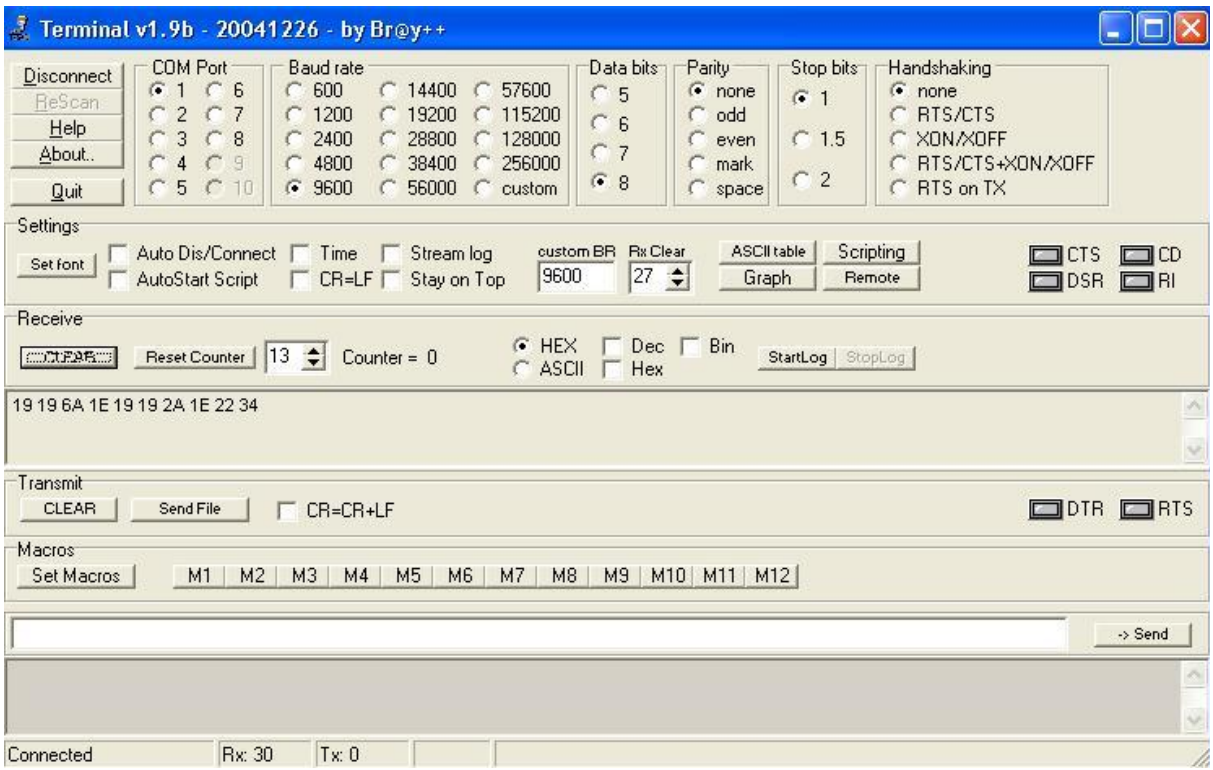
Iz priloženog se vidi da se detektovana i uneta greška poklapaju.

9. Tako formiranu poruku automatski šalje drugom mikrokontroleru koji je prihvata i proverava da li u četiri primljena bajta postoji greška. Detektovanu grešku ispravlja i šalje računaru, tako da sada na drugom prozoru pored zapisa sa greškom 19 19 6A 1E, vidimo i poruku u kojoj je ta greška već korigovana 19 19 2A 1E kao što je prikazano na slici 6.



Slika 6 - izled receive prozora sa ispravljenom greškom

10. Nakon toga zapis sa korigovanom greškom 19 19 2A 1E se vraća u dva bajta oduzimanjem dodatnih bitova Hamingovog koda, i takvu konačnu poruku takođe šalje računaru. Na drugom prozoru se pored zapisa sa greškom 19 19 6A 1E i korigovanog zapisa 19 19 2A 1E, vidi i ovaj zapis $22 $34 kao što je prikazano na slici 7.



Slika 7 - konačan izlged receive ekrana posle završenog procesa

Na osnovu unete poruke na prozoru za slanje i primljnog zapisa na drugom prozoru, utvrđuje se da je greška simulirana kako je i zadato, detektovana od strane drugog mikrokontrolera i korigovana takođe. Poruka na prvom prozoru (prva dva bajta) je identična kao konačna poruka na drugom prozoru.

5. Prilog 1

Predajnik:

```
#include <int.c>
#include <intrins.h>



unsigned char bdata RTemp;
sbit RTemp_0 = RTemp^0;
sbit RTemp_1 = RTemp^1;
sbit RTemp_2 = RTemp^2;
sbit RTemp_3 = RTemp^3;
sbit RTemp_4 = RTemp^4;
sbit RTemp_5 = RTemp^5;
sbit RTemp_6 = RTemp^6;
sbit RTemp_7 = RTemp^7;


unsigned char bdata LS;
sbit LS_0=LS^0;
sbit LS_1=LS^1;
sbit LS_2=LS^2;
sbit LS_3=LS^3;
sbit LS_4=LS^4;
sbit LS_5=LS^5;
sbit LS_6=LS^6;
sbit LS_7=LS^7;


void com_baudrate ()
{
EA = 0;

t_disabled = 1;

TR1 = 0;
ET1 = 0;


PCON |= 0x80;

TMOD = 0x21;

TH1 = 246;//(unsigned char) (256 - (XTAL / (16L * 12L * baudrate)));
```

```
TR1 = 1;
EA = 1;
}



void com_initialize (void)
{

  t_end = 0;
  com_baudrate ();

  EA = 0;




  SM0 = 0; SM1 = 1;
  SM2 = 0;
  REN = 1;


  ES = 1;
//  PS = 0;   ??????????????????

  EA = 1;
}

void init(){


  com_initialize();

  ilen=0;
}


void Hamming(unsigned char *indata, unsigned char *outdata)
{         // u indata baferu su dva bajta koja treba da se prepakuju u 4 bajta i upisu u
outdata

          RTemp = indata[0] & 0xF;     // Niza nibla
          LS_0 = (RTemp_0 ^ RTemp_1 ^ RTemp_3);
          LS_1 = (RTemp_0 ^ RTemp_2 ^ RTemp_3);
          LS_2 = RTemp_0;
          LS_3 = (RTemp_1 ^ RTemp_2 ^ RTemp_3);
          LS_4 = RTemp_1;
          LS_5 = RTemp_2;
          LS_6 = RTemp_3;
```

```
	LS_7 = 0;
	outdata[0] = LS;

	RTemp = (indata[0] >> 4) & 0xF;	// visa nibla
	LS_0 = (RTemp_0 ^ RTemp_1 ^ RTemp_3);
	LS_1 = (RTemp_0 ^ RTemp_2 ^ RTemp_3);
	LS_2 = RTemp_0;
	LS_3 = (RTemp_1 ^ RTemp_2 ^ RTemp_3);
	LS_4 = RTemp_1;
	LS_5 = RTemp_2;
	LS_6 = RTemp_3;
	LS_7 = 0;
	outdata[1] = LS;

	RTemp = indata[1] & 0xF;	// Niza nibla
	LS_0 = (RTemp_0 ^ RTemp_1 ^ RTemp_3);
	LS_1 = (RTemp_0 ^ RTemp_2 ^ RTemp_3);
	LS_2 = RTemp_0;
	LS_3 = (RTemp_1 ^ RTemp_2 ^ RTemp_3);
	LS_4 = RTemp_1;
	LS_5 = RTemp_2;
	LS_6 = RTemp_3;
	LS_7 = 0;
	outdata[2] = LS;

	RTemp = (indata[1] >> 4) & 0xF;	// visa nibla
	LS_0 = (RTemp_0 ^ RTemp_1 ^ RTemp_3);
	LS_1 = (RTemp_0 ^ RTemp_2 ^ RTemp_3);
	LS_2 = RTemp_0;
	LS_3 = (RTemp_1 ^ RTemp_2 ^ RTemp_3);
	LS_4 = RTemp_1;
	LS_5 = RTemp_2;
	LS_6 = RTemp_3;
	LS_7 = 0;
	outdata[3] = LS;
}

void PutError(unsigned char *message, unsigned char byteno, unsigned char bitno)
{
	unsigned char p;

	LS = message[(byteno % 4)];
	RTemp = 1 << (bitno % 8);

	p = LS & RTemp;
	if (p == 0)
	{
		LS = LS | RTemp;
	}
	else
```

```
        {
                RTemp = RTemp ^ 0xFF;
           LS = LS & RTemp;
        }

   message[(byteno % 8)] = LS;
}



void main(void){
        unsigned char k;
        unsigned char Hammed[4];

        init();

        P3_7=0;

        while (1)
        {
           if (ilen >= 4)
                {
                        Hamming(inbuf, Hammed);
                        PutError(Hammed, inbuf[2], inbuf[3]);

                        ilen = 0;

                for (k = 0; k < 4; k++)
                        {
                                P1 = Hammed[k];

                                P3_7=1;
                           while (!P3_2);
                                P3_7=0;
                                while (P3_2);
                        }
                }
        }
}
```

Int:

```
//#include "reg_c51.H"

//#include <REG2051.H>

#include <AT892051.H>



#define  OLEN  5                    /* size of serial transmission buffer   OVO JE BUFFER
ZA 20 TASTERA A IMA I REZERVA*/
#define  ILEN  4

unsigned char t_end;
unsigned char tbuf[OLEN];
unsigned char t_disabled;
unsigned char inbuf[ILEN];
unsigned char ilen;



/************************************************************************
*******/
/*       serial:  serial receiver / transmitter interrupt                */
/************************************************************************
*******/

/************************************************************************
*******/
/*       serial:  serial receiver / transmitter interrupt                */
/************************************************************************
*******/
serial () interrupt 4 using 2    {     /* use registerbank 2 for interrupt      */
  unsigned char c;
  unsigned char i;

         if (RI)
         {                    /* if receiver interrupt               */
                  c = SBUF;/* read character                       */
                  RI = 0;

                  if (ilen < 4)
                  {
                           inbuf[ilen] = c;
                  ilen++;
                  }

         }
```

```
        if (TI)
        {
                TI = 0;
                if (t_end>0)
                {
                        SBUF = tbuf [0];
                        for (i=0;i<t_end-1;i++)
                                tbuf[i]=tbuf[i+1];
                        t_end--;
                }
                else
                        t_disabled = 1;
        }
}
```

Prijemnik:

```
#include <int.c>
#include <intrins.h>



unsigned char bdata RTemp;
sbit RTemp_0 = RTemp^0;
sbit RTemp_1 = RTemp^1;
sbit RTemp_2 = RTemp^2;
sbit RTemp_3 = RTemp^3;
sbit RTemp_4 = RTemp^4;
sbit RTemp_5 = RTemp^5;
sbit RTemp_6 = RTemp^6;
sbit RTemp_7 = RTemp^7;


unsigned char bdata LS;
sbit LS_0=LS^0;
sbit LS_1=LS^1;
sbit LS_2=LS^2;
sbit LS_3=LS^3;
sbit LS_4=LS^4;
sbit LS_5=LS^5;
sbit LS_6=LS^6;
sbit LS_7=LS^7;



void com_baudrate ()
{
EA = 0;

t_disabled = 1;

TR1 = 0;
ET1 = 0;


PCON |= 0x80;

TMOD = 0x21;

TH1 = 246;//(unsigned char) (256 - (XTAL / (16L * 12L * baudrate)));


TR1 = 1;
```

```
EA = 1;
}



void com_initialize (void)
{

  t_end = 0;
  com_baudrate ();

  EA = 0;




  SM0 = 0; SM1 = 1;
  SM2 = 0;
  REN = 1;


  ES = 1;
//  PS = 0;   ??????????????????

  EA = 1;
}

void init(){


  com_initialize();

  ilen=0;
}

char putchar (char c)
{
while (com_putchar (c) != 0);
return (c);
}


void Ispravi(unsigned char *indata)
{
        unsigned char i, bec;

        for (i = 0; i < 4; i++)
        {
                RTemp = indata[i];
                LS_0 = (RTemp_2 ^ RTemp_4 ^ RTemp_6);
```

```
		LS_1 = (RTemp_2 ^ RTemp_5 ^ RTemp_6);
		LS_3 = (RTemp_4 ^ RTemp_5 ^ RTemp_6);
		bec = 0;
		if (LS_0 != RTemp_0)
			bec += 1;
		if (LS_1 != RTemp_1)
			bec += 2;
		if (LS_3 != RTemp_3)
			bec += 4;

		switch (bec) {
			case 1 :RTemp_0 ^= 1;
					break;
			case 2 :RTemp_1 ^= 1;
					break;
			case 3 :RTemp_2 ^= 1;
					break;
			case 4 :RTemp_3 ^= 1;
					break;
			case 5:	RTemp_4 ^= 1;
					break;
			case 6 :RTemp_5 ^= 1;
					break;
			case 7 :RTemp_6 ^= 1;
					break;
		//	default:
		}

		indata[i] = RTemp;
	}
}


void DeHamming(unsigned char *indata, unsigned char *outdata)
{
	RTemp = indata[0];
	LS_0 = RTemp_2;
	LS_1 = RTemp_4;
	LS_2 = RTemp_5;
	LS_3 = RTemp_6;
	RTemp = indata[1];
	LS_4 = RTemp_2;
	LS_5 = RTemp_4;
	LS_6 = RTemp_5;
	LS_7 = RTemp_6;
	outdata[0] = LS;

	RTemp = indata[2];
	LS_0 = RTemp_2;
	LS_1 = RTemp_4;
```

```
        LS_2 = RTemp_5;
        LS_3 = RTemp_6;
        RTemp = indata[3];
        LS_4 = RTemp_2;
        LS_5 = RTemp_4;
        LS_6 = RTemp_5;
        LS_7 = RTemp_6;
        outdata[1] = LS;
}


void main(void){
  bit BioNula;
  unsigned char i, j;
  unsigned char Buf[4];
  unsigned char DeHammed[2];

  init();

  BioNula = 0;
  P3_2 = 0;
  i = 0;

        while (1)
        {
                if (P3_7 == 0)
                {
                BioNula = 1;
                        P3_2 = 0;
                }
           if ((P3_7 == 1) && (BioNula))
                {
                        BioNula = 0;
                        if (i < 4)
                        {
                                Buf[i] = P1;
                                i++;
                        }
                        P3_2 = 1;
                }



                if (i == 4)
                {

                        for (j = 0; j < 4; j++)
                        {
                                putchar(Buf[j]);
                        }
```

```
                    Ispravi(Buf);
                    DeHamming(Buf, DeHammed);

                    for (j = 0; j < 4; j++)
                    {
                        putchar(Buf[j]);
                    }

                    for (j = 0; j < 2; j++)
                    {
                        putchar(DeHammed[j]);
                    }

                    i = 0;
                }

        }       // while(1)

}
```

6. Prilog 2

predajnik:

```
BL51 BANKED LINKER/LOCATER V6.00

BL51 BANKED LINKER/LOCATER V6.00, INVOKED BY:
C:\KEILC51\C51\BIN\BL51.EXE tast.obj TO Send


MEMORY MODEL: SMALL


INPUT MODULES INCLUDED:
  tast.obj (TAST)
  C:\KEILC51\C51\LIB\C51S.LIB (?C_STARTUP)
  C:\KEILC51\C51\LIB\C51S.LIB (?C?CLDPTR)
  C:\KEILC51\C51\LIB\C51S.LIB (?C?CLDOPTR)
  C:\KEILC51\C51\LIB\C51S.LIB (?C?CSTPTR)
  C:\KEILC51\C51\LIB\C51S.LIB (?C?CSTOPTR)


LINK MAP OF MODULE:  Send (TAST)


        TYPE    BASE      LENGTH    RELOCATION   SEGMENT NAME
        -----------------------------------------------------

        * * * * * * *  D A T A   M E M O R Y  * * * * * * *
        REG     0000H     0008H     ABSOLUTE     "REG BANK 0"
              0008H     0008H               *** GAP ***
        REG     0010H     0008H     ABSOLUTE     "REG BANK 2"
              0018H     0008H               *** GAP ***
        DATA    0020H     0002H     BIT_ADDR     ?BA?TAST
        DATA    0022H     000CH     UNIT         ?DT?TAST
        DATA    002EH     000AH     UNIT         _DATA_GROUP_
        IDATA   0038H     0001H     UNIT         ?STACK

        * * * * * * *  C O D E   M E M O R Y  * * * * * * *
        CODE    0000H     0003H     ABSOLUTE
        CODE    0003H     0017H     INBLOCK      ?PR?COM_BAUDRATE?TAST
        CODE    001AH     0006H     INBLOCK      ?PR?INIT?TAST
              0020H     0003H               *** GAP ***
        CODE    0023H     0002H     ABSOLUTE
        CODE    0025H     0141H     INBLOCK      ?PR?_HAMMING?TAST
        CODE    0166H     007AH     UNIT         ?C?LIB_CODE
        CODE    01E0H     0053H     INBLOCK      ?PR?SERIAL?TAST
        CODE    0233H     0047H     INBLOCK      ?PR?_PUTERROR?TAST
```

```
        CODE    027AH     0047H     INBLOCK      ?PR?MAIN?TAST
        CODE    02C1H     0014H     INBLOCK      ?PR?COM_INITIALIZE?TAST
        CODE    02D5H     000CH     UNIT         ?C_C51STARTUP


OVERLAY MAP OF MODULE:   Send (TAST)


SEGMENT                          DATA_GROUP
  +--> CALLED SEGMENT          START    LENGTH
----------------------------------------------
?C_C51STARTUP                  -----    -----
  +--> ?PR?MAIN?TAST

?PR?MAIN?TAST                  002EH    0004H

BL51 BANKED LINKER/LOCATER V6.00


  +--> ?PR?INIT?TAST
  +--> ?PR?_HAMMING?TAST
  +--> ?PR?_PUTERROR?TAST

?PR?INIT?TAST                  -----    -----
  +--> ?PR?COM_INITIALIZE?TAST

?PR?COM_INITIALIZE?TAST        -----    -----
  +--> ?PR?COM_BAUDRATE?TAST

?PR?_HAMMING?TAST              0032H    0006H

?PR?_PUTERROR?TAST             0032H    0005H


SYMBOL TABLE OF MODULE:  Send (TAST)

  VALUE           TYPE          NAME
  ----------------------------------

  -------         MODULE        TAST
  C:0000H         SYMBOL        _ICE_DUMMY_
  B:0020H.4       PUBLIC        RTemp_4
  B:0020H.5       PUBLIC        RTemp_5
  B:0020H.6       PUBLIC        RTemp_6
  D:0090H         PUBLIC        P1
  B:0020H.7       PUBLIC        RTemp_7
  C:01E0H         PUBLIC        serial
  B:00A8H.7       PUBLIC        EA
```

```
D:0022H         PUBLIC        t_end
D:0023H         PUBLIC        t_disabled
B:00B0H.2       PUBLIC        P3_2
D:0024H         PUBLIC        inbuf
B:00A8H.4       PUBLIC        ES
B:00B0H.7       PUBLIC        P3_7
B:0098H.0       PUBLIC        RI
B:0098H.1       PUBLIC        TI
D:0021H         PUBLIC        LS
C:0025H         PUBLIC        _Hamming
C:027AH         PUBLIC        main
C:0233H         PUBLIC        _PutError
D:0028H         PUBLIC        ilen
B:0021H.0       PUBLIC        LS_0
B:0021H.1       PUBLIC        LS_1
B:0021H.2       PUBLIC        LS_2
D:0099H         PUBLIC        SBUF
D:0087H         PUBLIC        PCON
B:0021H.3       PUBLIC        LS_3
B:0021H.4       PUBLIC        LS_4
D:0029H         PUBLIC        tbuf
B:0021H.5       PUBLIC        LS_5
B:0021H.6       PUBLIC        LS_6
D:0089H         PUBLIC        TMOD
C:001AH         PUBLIC        init
B:0021H.7       PUBLIC        LS_7
B:00A8H.3       PUBLIC        ET1
```

BL51 BANKED LINKER/LOCATER V6.00

```
D:008DH         PUBLIC        TH1
B:0098H.7       PUBLIC        SM0
B:0098H.6       PUBLIC        SM1
B:0098H.5       PUBLIC        SM2
B:0088H.6       PUBLIC        TR1
C:02C1H         PUBLIC        com_initialize
B:0098H.4       PUBLIC        REN
D:0020H         PUBLIC        RTemp
C:0003H         PUBLIC        com_baudrate
B:0020H.0       PUBLIC        RTemp_0
B:0020H.1       PUBLIC        RTemp_1
B:0020H.2       PUBLIC        RTemp_2
B:0020H.3       PUBLIC        RTemp_3
-------         PROC          SERIAL
-------         DO
D:0017H         SYMBOL        c
D:0017H         SYMBOL        i
-------         ENDDO
C:01E0H         LINE#         27
```

```
C:01E7H         LINE#         31
C:01EAH         LINE#         32
C:01EAH         LINE#         33
C:01ECH         LINE#         34
C:01EEH         LINE#         36
C:01F5H         LINE#         37
C:01F5H         LINE#         38
C:01FCH         LINE#         39
C:01FEH         LINE#         40
C:01FEH         LINE#         42
C:01FEH         LINE#         44
C:0201H         LINE#         45
C:0201H         LINE#         46
C:0203H         LINE#         47
C:020AH         LINE#         48
C:020AH         LINE#         49
C:020DH         LINE#         50
C:0218H         LINE#         51
C:0227H         LINE#         52
C:0229H         LINE#         53
C:022BH         LINE#         55
C:022EH         LINE#         56
C:022EH         LINE#         57
-------         ENDPROC       SERIAL
-------         PROC          COM_BAUDRATE
C:0003H         LINE#         29
C:0003H         LINE#         30
C:0003H         LINE#         31
C:0005H         LINE#         33
C:0008H         LINE#         35
C:000AH         LINE#         36
C:000CH         LINE#         39
C:000FH         LINE#         41
C:0012H         LINE#         43
C:0015H         LINE#         46
C:0017H         LINE#         47
C:0019H         LINE#         48
-------         ENDPROC       COM_BAUDRATE
```



```
-------         PROC          COM_INITIALIZE
C:02C1H         LINE#         52
C:02C1H         LINE#         53
C:02C1H         LINE#         55
C:02C4H         LINE#         56
C:02C6H         LINE#         58
C:02C8H         LINE#         63
C:02CCH         LINE#         64
```

```
C:02CEH         LINE#         65
C:02D0H         LINE#         68
C:02D2H         LINE#         71
C:02D4H         LINE#         72
-------         ENDPROC       COM_INITIALIZE
-------         PROC          INIT
C:001AH         LINE#         74
C:001AH         LINE#         77
C:001CH         LINE#         79
C:001FH         LINE#         80
-------         ENDPROC       INIT
-------         PROC          _HAMMING
D:0032H         SYMBOL        indata
D:0035H         SYMBOL        outdata
C:0025H         LINE#         83
C:002BH         LINE#         84
C:002BH         LINE#         86
C:0031H         LINE#         87
C:003DH         LINE#         88
C:0049H         LINE#         89
C:004DH         LINE#         90
C:0059H         LINE#         91
C:005DH         LINE#         92
C:0061H         LINE#         93
C:0065H         LINE#         94
C:0067H         LINE#         95
C:0071H         LINE#         97
C:007EH         LINE#         98
C:008AH         LINE#         99
C:0096H         LINE#         100
C:009AH         LINE#         101
C:00A6H         LINE#         102
C:00AAH         LINE#         103
C:00AEH         LINE#         104
C:00B2H         LINE#         105
C:00B4H         LINE#         106
C:00C1H         LINE#         108
C:00D0H         LINE#         109
C:00DCH         LINE#         110
C:00E8H         LINE#         111
C:00ECH         LINE#         112
C:00F8H         LINE#         113
C:00FCH         LINE#         114
C:0100H         LINE#         115
C:0104H         LINE#         116
C:0106H         LINE#         117
C:0113H         LINE#         119
C:0123H         LINE#         120
C:012FH         LINE#         121
```

```
C:013BH         LINE#         122
C:013FH         LINE#         123
C:014BH         LINE#         124
C:014FH         LINE#         125
C:0153H         LINE#         126
C:0157H         LINE#         127
C:0159H         LINE#         128
-------         ENDPROC       _HAMMING
-------         PROC          _PUTERROR
D:0032H         SYMBOL        message
D:0005H         SYMBOL        byteno
D:0036H         SYMBOL        bitno
-------         DO
D:0007H         SYMBOL        p
-------         ENDDO
C:0233H         LINE#         131
C:0239H         LINE#         132
C:0239H         LINE#         135
C:0245H         LINE#         136
C:0257H         LINE#         138
C:0259H         LINE#         139
C:025BH         LINE#         140
C:025BH         LINE#         141
C:025FH         LINE#         142
C:0261H         LINE#         144
C:0261H         LINE#         145
C:0264H         LINE#         146
C:0268H         LINE#         147
C:0268H         LINE#         149
-------         ENDPROC       _PUTERROR
-------         PROC          MAIN
-------         DO
D:0006H         SYMBOL        k
D:002EH         SYMBOL        Hammed
-------         ENDDO
C:027AH         LINE#         154
C:027AH         LINE#         158
C:027CH         LINE#         160
C:027EH         LINE#         162
C:027EH         LINE#         163
C:027EH         LINE#         164
C:0285H         LINE#         165
C:0285H         LINE#         166
C:0296H         LINE#         167
C:02A3H         LINE#         169
C:02A6H         LINE#         171
C:02ADH         LINE#         172
```

```
  C:02ADH         LINE#         173
  C:02B4H         LINE#         175
  C:02B6H         LINE#         176
  C:02B9H         LINE#         177
  C:02BBH         LINE#         178
  C:02BEH         LINE#         179
  -------         ENDPROC       MAIN
  -------         ENDMOD        TAST

  -------         MODULE        ?C?CLDPTR

BL51 BANKED LINKER/LOCATER V6.00


  C:0166H         PUBLIC        ?C?CLDPTR
  -------         ENDMOD        ?C?CLDPTR

  -------         MODULE        ?C?CLDOPTR
  C:017FH         PUBLIC        ?C?CLDOPTR
  -------         ENDMOD        ?C?CLDOPTR

  -------         MODULE        ?C?CSTPTR
  C:01ACH         PUBLIC        ?C?CSTPTR
  -------         ENDMOD        ?C?CSTPTR

  -------         MODULE        ?C?CSTOPTR
  C:01BEH         PUBLIC        ?C?CSTOPTR
  -------         ENDMOD        ?C?CSTOPTR

Program Size: data=41.0 xdata=0 code=734
LINK/LOCATE RUN COMPLETE.  0 WARNING(S),  0 ERROR(S)
```

Prijemnik:

```
BL51 BANKED LINKER/LOCATER V6.00


BL51 BANKED LINKER/LOCATER V6.00, INVOKED BY:
C:\KEILC51\C51\BIN\BL51.EXE tast.obj TO Recv


MEMORY MODEL: SMALL


INPUT MODULES INCLUDED:
  tast.obj (TAST)
  C:\KEILC51\C51\LIB\C51S.LIB (?C_STARTUP)
  C:\KEILC51\C51\LIB\C51S.LIB (?C?CLDPTR)
  C:\KEILC51\C51\LIB\C51S.LIB (?C?CLDOPTR)
  C:\KEILC51\C51\LIB\C51S.LIB (?C?CSTPTR)
  C:\KEILC51\C51\LIB\C51S.LIB (?C?CSTOPTR)


LINK MAP OF MODULE:  Recv (TAST)


        TYPE    BASE     LENGTH    RELOCATION   SEGMENT NAME
        -----------------------------------------------------

        * * * * * * *  D A T A   M E M O R Y  * * * * * * *
        REG     0000H    0008H    ABSOLUTE     "REG BANK 0"
              0008H    0008H             *** GAP ***
        REG     0010H    0008H    ABSOLUTE     "REG BANK 2"
              0018H    0008H             *** GAP ***
        DATA    0020H    0002H     BIT_ADDR     ?BA?TAST
        BIT     0022H.0  0000H.1   UNIT         _BIT_GROUP_
              0022H.1  0000H.7           *** GAP ***
        DATA    0023H    000DH     UNIT         _DATA_GROUP_
        DATA    0030H    000CH     UNIT         ?DT?TAST
        IDATA   003CH    0001H     UNIT         ?STACK

        * * * * * * *  C O D E   M E M O R Y  * * * * * * *
        CODE    0000H    0003H     ABSOLUTE
        CODE    0003H    0017H     INBLOCK      ?PR?COM_BAUDRATE?TAST
        CODE    001AH    0006H     INBLOCK      ?PR?INIT?TAST
              0020H    0003H             *** GAP ***
        CODE    0023H    0002H     ABSOLUTE
        CODE    0025H    00C2H     INBLOCK      ?PR?_ISPRAVI?TAST
        CODE    00E7H    007CH     INBLOCK      ?PR?_DEHAMMING?TAST
        CODE    0163H    007AH     UNIT         ?C?LIB_CODE
        CODE    01DDH    0069H     INBLOCK      ?PR?MAIN?TAST
        CODE    0246H    0048H     INBLOCK      ?PR?SERIAL?TAST
```

```
        CODE    028EH     0027H     INBLOCK      ?PR?_COM_PUTCHAR?TAST
        CODE    02B5H     0014H     INBLOCK      ?PR?COM_INITIALIZE?TAST
        CODE    02C9H     0012H     INBLOCK      ?PR?_PUTCHAR?TAST
        CODE    02DBH     000CH     UNIT         ?C_C51STARTUP


OVERLAY MAP OF MODULE:   Recv (TAST)


SEGMENT                          BIT_GROUP          DATA_GROUP
  +--> CALLED SEGMENT          START    LENGTH     START    LENGTH
-----------------------------------------------------------------

BL51 BANKED LINKER/LOCATER V6.00


?C_C51STARTUP                -----    -----      -----    -----
  +--> ?PR?MAIN?TAST

?PR?MAIN?TAST                0022H.0  0000H.1    0023H    0007H
  +--> ?PR?INIT?TAST
  +--> ?PR?_PUTCHAR?TAST
  +--> ?PR?_ISPRAVI?TAST
  +--> ?PR?_DEHAMMING?TAST

?PR?INIT?TAST                -----    -----      -----    -----
  +--> ?PR?COM_INITIALIZE?TAST

?PR?COM_INITIALIZE?TAST      -----    -----      -----    -----
  +--> ?PR?COM_BAUDRATE?TAST

?PR?_PUTCHAR?TAST            -----    -----      -----    -----
  +--> ?PR?_COM_PUTCHAR?TAST

?PR?_ISPRAVI?TAST            -----    -----      002AH    0003H

?PR?_DEHAMMING?TAST          -----    -----      002AH    0006H


SYMBOL TABLE OF MODULE:  Recv (TAST)

  VALUE          TYPE          NAME
  ----------------------------------

  -------        MODULE        TAST
  C:0000H        SYMBOL        _ICE_DUMMY_
  B:0020H.4      PUBLIC        RTemp_4
  B:0020H.5      PUBLIC        RTemp_5
```

B:0020H.6 PUBLIC RTemp_6
D:0090H PUBLIC P1
B:0020H.7 PUBLIC RTemp_7
C:0246H PUBLIC serial
B:00A8H.7 PUBLIC EA
D:0030H PUBLIC t_end
D:0031H PUBLIC t_disabled
B:00B0H.2 PUBLIC P3_2
D:0032H PUBLIC inbuf
B:00A8H.4 PUBLIC ES
B:00B0H.7 PUBLIC P3_7
B:0098H.0 PUBLIC RI
B:0098H.1 PUBLIC TI
D:0021H PUBLIC LS
C:01DDH PUBLIC main
D:0036H PUBLIC ilen
B:0021H.0 PUBLIC LS_0
B:0021H.1 PUBLIC LS_1
B:0021H.2 PUBLIC LS_2
D:0099H PUBLIC SBUF
D:0087H PUBLIC PCON
B:0021H.3 PUBLIC LS_3
B:0021H.4 PUBLIC LS_4
D:0037H PUBLIC tbuf

B:0021H.5 PUBLIC LS_5
B:0021H.6 PUBLIC LS_6
D:0089H PUBLIC TMOD
C:001AH PUBLIC init
B:0021H.7 PUBLIC LS_7
C:0025H PUBLIC _Ispravi
B:00A8H.3 PUBLIC ET1
D:008DH PUBLIC TH1
C:00E7H PUBLIC _DeHamming
B:0098H.7 PUBLIC SM0
B:0098H.6 PUBLIC SM1
B:0098H.5 PUBLIC SM2
B:0088H.6 PUBLIC TR1
C:02B5H PUBLIC com_initialize
C:02CFH PUBLIC _putchar
B:0098H.4 PUBLIC REN
D:0020H PUBLIC RTemp
C:0003H PUBLIC com_baudrate
B:0020H.0 PUBLIC RTemp_0
B:0020H.1 PUBLIC RTemp_1
B:0020H.2 PUBLIC RTemp_2
C:028EH PUBLIC _com_putchar

```
B:0020H.3       PUBLIC        RTemp_3
-------         PROC          _COM_PUTCHAR
D:0007H         SYMBOL        c
C:028EH         LINE#         19
C:028EH         LINE#         20
C:028EH         LINE#         24
C:0298H         LINE#         25
C:029BH         LINE#         31
C:029DH         LINE#         33
C:02A7H         LINE#         35
C:02ABH         LINE#         36
C:02ABH         LINE#         37
C:02AEH         LINE#         38
C:02B0H         LINE#         39
C:02B0H         LINE#         41
C:02B2H         LINE#         43
C:02B4H         LINE#         44
-------         ENDPROC       _COM_PUTCHAR
-------         PROC          SERIAL
-------         DO
D:0017H         SYMBOL        c
D:0017H         SYMBOL        i
-------         ENDDO
C:0246H         LINE#         56
C:024DH         LINE#         60
C:0250H         LINE#         61
C:0252H         LINE#         62
C:0254H         LINE#         63
C:0256H         LINE#         64
C:0259H         LINE#         66
C:0259H         LINE#         68
C:025CH         LINE#         69
C:025CH         LINE#         70
C:025EH         LINE#         71
C:0265H         LINE#         72
```



```
C:0268H         LINE#         73
C:0282H         LINE#         74
C:0284H         LINE#         75
C:0286H         LINE#         77
C:0289H         LINE#         78
C:0289H         LINE#         79
-------         ENDPROC       SERIAL
-------         PROC          COM_BAUDRATE
C:0003H         LINE#         30
C:0003H         LINE#         31
C:0003H         LINE#         32
```

```
C:0005H         LINE#         34
C:0008H         LINE#         36
C:000AH         LINE#         37
C:000CH         LINE#         40
C:000FH         LINE#         42
C:0012H         LINE#         44
C:0015H         LINE#         47
C:0017H         LINE#         48
C:0019H         LINE#         49
-------         ENDPROC       COM_BAUDRATE
-------         PROC          COM_INITIALIZE
C:02B5H         LINE#         53
C:02B5H         LINE#         54
C:02B5H         LINE#         56
C:02B8H         LINE#         57
C:02BAH         LINE#         59
C:02BCH         LINE#         64
C:02C0H         LINE#         65
C:02C2H         LINE#         66
C:02C4H         LINE#         69
C:02C6H         LINE#         72
C:02C8H         LINE#         73
-------         ENDPROC       COM_INITIALIZE
-------         PROC          INIT
C:001AH         LINE#         75
C:001AH         LINE#         78
C:001CH         LINE#         80
C:001FH         LINE#         81
-------         ENDPROC       INIT
C:02CBH         SYMBOL        L?0063
-------         PROC          L?0062
-------         ENDPROC       L?0062
C:02CBH         SYMBOL        L?0063
-------         PROC          _PUTCHAR
D:0005H         SYMBOL        c
C:02CFH         LINE#         83
C:02D1H         LINE#         84
C:02D1H         LINE#         85
C:02D8H         LINE#         86
C:02DAH         LINE#         87
-------         ENDPROC       _PUTCHAR
-------         PROC          _ISPRAVI
D:002AH         SYMBOL        indata
-------         DO
D:0007H         SYMBOL        i
D:0006H         SYMBOL        bec
```

```
-------         ENDDO
C:0025H         LINE#         90
C:002BH         LINE#         91
C:002BH         LINE#         94
C:002DH         LINE#         95
C:002DH         LINE#         96
C:003CH         LINE#         97
C:0048H         LINE#         98
C:0054H         LINE#         99
C:0060H         LINE#         100
C:0062H         LINE#         101
C:006AH         LINE#         102
C:006BH         LINE#         103
C:0073H         LINE#         104
C:0075H         LINE#         105
C:007DH         LINE#         106
C:0081H         LINE#         108
C:00A0H         LINE#         109
C:00A5H         LINE#         110
C:00A7H         LINE#         111
C:00ACH         LINE#         112
C:00AEH         LINE#         113
C:00B3H         LINE#         114
C:00B5H         LINE#         115
C:00BAH         LINE#         116
C:00BCH         LINE#         117
C:00C1H         LINE#         118
C:00C3H         LINE#         119
C:00C8H         LINE#         120
C:00CAH         LINE#         121
C:00CFH         LINE#         122
C:00CFH         LINE#         124
C:00CFH         LINE#         126
C:00DEH         LINE#         127
C:00E6H         LINE#         128
-------         ENDPROC       _ISPRAVI
-------         PROC          _DEHAMMING
D:002AH         SYMBOL        indata
D:002DH         SYMBOL        outdata
C:00E7H         LINE#         131
C:00EDH         LINE#         132
C:00EDH         LINE#         133
C:00F1H         LINE#         134
C:00F5H         LINE#         135
C:00F9H         LINE#         136
C:00FDH         LINE#         137
C:0101H         LINE#         138
C:0108H         LINE#         139
C:010CH         LINE#         140
C:0110H         LINE#         141
```

```
  C:0114H         LINE#         142
  C:0118H         LINE#         143
  C:0122H         LINE#         145
  C:012FH         LINE#         146
  C:0133H         LINE#         147
  C:0137H         LINE#         148
  C:013BH         LINE#         149

BL51 BANKED LINKER/LOCATER V6.00


C:013FH         LINE#         150
  C:0146H         LINE#         151
  C:014AH         LINE#         152
  C:014EH         LINE#         153
  C:0152H         LINE#         154
  C:0156H         LINE#         155
  -------         ENDPROC       _DEHAMMING
  -------         PROC          MAIN
  -------         DO
  B:0022H.0       SYMBOL        BioNula
  D:0023H         SYMBOL        i
  D:0004H         SYMBOL        j
  D:0024H         SYMBOL        Buf
  D:0028H         SYMBOL        DeHammed
  -------         ENDDO
  C:01DDH         LINE#         159
  C:01DDH         LINE#         165
  C:01DFH         LINE#         167
  C:01E1H         LINE#         168
  C:01E3H         LINE#         169
  C:01E6H         LINE#         171
  C:01E6H         LINE#         172
  C:01E6H         LINE#         173
  C:01E9H         LINE#         174
  C:01E9H         LINE#         175
  C:01EBH         LINE#         176
  C:01EDH         LINE#         177
  C:01EDH         LINE#         178
  C:01F3H         LINE#         179
  C:01F3H         LINE#         180
  C:01F5H         LINE#         181
  C:01FCH         LINE#         182
  C:01FCH         LINE#         183
  C:0203H         LINE#         184
  C:0205H         LINE#         185
  C:0205H         LINE#         186
  C:0207H         LINE#         187
  C:0207H         LINE#         191
```

```
C:020CH         LINE#         192
C:020CH         LINE#         194
C:020EH         LINE#         195
C:020EH         LINE#         196
C:0211H         LINE#         197
C:0215H         LINE#         199
C:021DH         LINE#         200
C:022EH         LINE#         202
C:0230H         LINE#         203
C:0230H         LINE#         204
C:0233H         LINE#         205
C:0237H         LINE#         207
C:0239H         LINE#         208
C:0239H         LINE#         209
C:023DH         LINE#         210
C:0241H         LINE#         212
C:0244H         LINE#         213
C:0244H         LINE#         215
-------         ENDPROC       MAIN
```

BL51 BANKED LINKER/LOCATER V6.00

```
-------         ENDMOD        TAST

-------         MODULE        ?C?CLDPTR
C:0163H           PUBLIC        ?C?CLDPTR
-------         ENDMOD        ?C?CLDPTR

-------         MODULE        ?C?CLDOPTR
C:017CH           PUBLIC        ?C?CLDOPTR
-------         ENDMOD        ?C?CLDOPTR

-------         MODULE        ?C?CSTPTR
C:01A9H           PUBLIC        ?C?CSTPTR
-------         ENDMOD        ?C?CSTPTR

-------         MODULE        ?C?CSTOPTR
C:01BBH           PUBLIC        ?C?CSTOPTR
-------         ENDMOD        ?C?CSTOPTR
```

Program Size: data=44.1 xdata=0 code=740
LINK/LOCATE RUN COMPLETE. 0 WARNING(S), 0 ERROR(S)

Dodatak

U dodatku je dokument koji sadrži opis procesora koji je korišćen sa karakteristikama, dat od strane proizvođača.