Projekat iz predmeta: Projektovanje ugrađenih računarskih sistema
Prof. Dr. Mile Stojčev
Elektronski fakultet u Nišu

RS 232 terminal

Studenti : Miroslav Božić
Aleksandar Atanasovski

septembar 2008.

SADRŽAJ:

1. Specifikacija zahteva

Potrebno je realizovati računarsku aplikaciju i odgovarajući hardver koji se sastoji od mikrokontrolera koji omogućavaju razmenu kratkih tekstualnih poruka između dva računara (terminala) koristeći asinhronu serijsku komunikaciju. Projekat je predviđen za demonstraciju serijske komunikacije korišćenjem rs-232 protokola i najnovijeg usb 2.0 standarda. U okviru softvera se nalazi i interfejs koji se lako koristi i preko koga je moguće podešavati razne parametre prenosa. Softver se realizuje kao Windows aplikacija iz koje se vrši slanje i prijem podataka preko jednostavnog hardvera. Projekat implementira i metode za detekciju i otklanjanje grešaka u prenosu kao što su bit parnosti i vertikalna parnost, takođe omogućava izbor različitih brzina asinhronog prenosa.

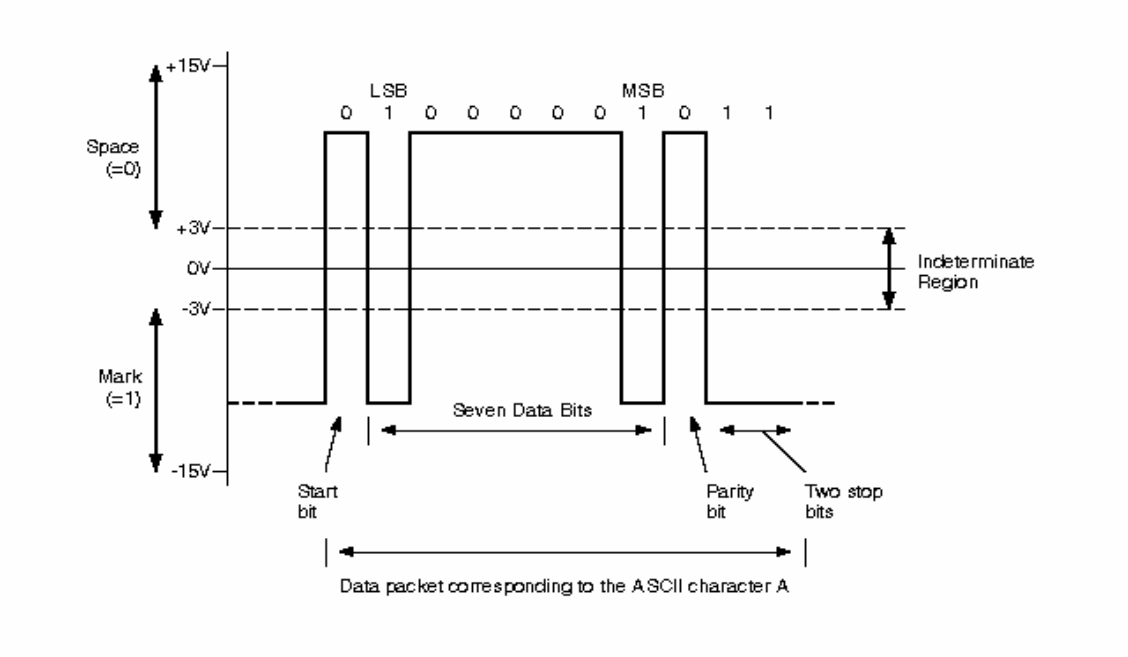
2. Asinhrona serijska komunikacija

Serijski način prenosa podataka nastao je u cilju smanjenja broja komunikacionih linija izmedju dva terminala. Podaci se kroz komunikacionu liniju šalju sekvencialno, bit po bit za razliku od paralelnog prenosa podataka. Zbog toga je kod serijske komunikacije broj komuniikacionih linija (signala) sveden na minimum kao kod asinhronog prenosa koji nije sinhronizovan taktom. Serijski prenos se dakle deli na sinhroni i asinhroni u zavisnosti da li se koriste dodatne linije za sinhronizaciju. Takođe se mogu koristiti i neki drugi kontrolni siignali i linije u zavisnosti od protokola.
Asinhorona serijska komunikacija se odlikuje malom brzinom prenosa ali i odustvom dodatne linije za sinhronzacioni takt. Zbog toga je asinhroni prenos manje pouzdan što ograničava njegovu brzinu. Međutim uz napredne tehnike otklanjanja i detekcije gešaka i sa razvojem tehnologije (dizajn kablova i izbor naponskih nivoa) brzina prenosa je stalno u porastu (USB 2.0). Najčešće korišćeni standard za ovaj način komunikacije su rs-232 i USB.

2.1 Rs-232 standard

Standard za računarsku komunikaciju rs-232 spada u grupu takozvanih *"single ended"* komunikacija (podaci se preose preko jedne žice). Ovaj standard je predstavljen 1969. godine i uz nekoliko revizija je ostao u upotrebi do danas ali sve manje u savremenim računarima. Namera projektanata ovog standarda je bila da se omogući povezivanje računara sa modemima tj, DTE (*data terminal equipement*) terminala sa DCE (*data comunication equipement*) komunikacionom opremom. Uprkos svojoj prvobitnoj nameni rs-232 se koristi za prenos podataka izmedju različitih uređaja pa se zato ponekad naziva i "najnestandardniji standard". Trenutno aktuelna verzija ovog standarda je rs-232 E iz 1991. godine. Komunikacija ovim standardom je veoma jednostavna jer se njime definiše samo komunikacija na fizičkom nivou i nivou podataka. Na fizičkom nivou se definišu dva stanja na liniji : *MARK* stanju odgovara naponski nivo u opsegu od -12V do -3V , ovaj opseg napona definiše i logicku jedinicu. Stanje *SPACE* je u opsegu napona od +3V do +12V i predstavlja logičku nulu pri prenosu podataka. Stanja *MARK* i *SPACE* definisu neaktivnu (idle) i aktivnu liniju respekktivno. Opseg napona izmedju -3V i +3V se naziva mrtvom zonom i služi da bi se smanjio uticaj smetnji. Napon na linijama ne sme biti veći od 25V u odnosu na masu Električne karakteristike tj. naponski nivoi se veoma razlikuju u zavisnosti od aplikacije i uređaja kao i revizije standarda što takođe opravdava naziv "nestandardni standard" koji rs-232 ima. Zbog toga su razvijena razna kola za prilagođavanje ovih naponskih nivoa (na pr. Max 232).

Podaci se šalju u paketima od po 8 ili 5 bita. Pre slanja podataka linija je u stanju idle sve dok se ne pojavi start bit, nakon toga sledi povorka od 8 bitova podataka i opciono bit parnoti. Okvir se završava sa 1 1,5 ili 2 stop bita. Da bi se komunikacija obavljala uspešno neophodno je da se izvrši sinhronizacija prijemnika i predajnika kako bi brzine uzorkovanja na oba terminala bile jednake. Ovo podrazumeva da je i veličina polja za podatke jednaka (8 ili 5 bita). Na ovaj način je svaki podatak uokviren start i stop bitovima kojima se odvaja od sledećeg i prethodnog paketa podataka ali se obezbeđuje i minimalno jedan prelaz iz stanja *MARK* U stanje *SPACE*.



Kod asinhrone komunikacije se definiše tzv “bodova brzina” koja predstavlja koliko se bitova podataka može preneti u jedinici vremena. Bitska brzina prijemnika i predajnika ne smeju se razlikovati više od 3 do 5 procenata u zavisnosti od brzine jer može doći do pogrešne intepretacije podataka. Bitska brzina se krece u rasponu od 2400 do 115000 bodova i na njnu najviše utiču fizička ogrničenja u komunikaciji. Ova brzina zavisi od toga kolikom brzinom prijemnik može da uzorkuje stanje na liniji. Obično je ta brzina zbog pouzdanosti bude nekoliko puta (najčešće 16) veća od brzine prenosa.

2.2 Detekcija grešaka

Zbog nedostatka sinhro signala rs-232 je podložan greškama u prenosu pa je njihova detekcija i otklanjanje veoma bita zbog postizanja i održavanja željene brzine.

Detekcija grešaka kod ovog načina konunikacije je obezbeđena jednim redundantnim bitom koji se dodaje poruci koji se naziva “bit parnosti”. Ovaj bit ima dvojako značenje. Kod parne parnosti ovaj bit se setuje kao 0 ili 1 da bi se podatak dopunio kako bi imao paran broj jedinica. Kod neparne parnosti ovaj bit ima obrnutu ulogu tj. dodaje se podatku da bi ukupan broj jedinica bio neparan.

	paran broj jedinica	neparan broj jedinica
parna parnost	**0**	**1**
neparna parnost	**1**	**0**

Ovakav način detekcije grešaka je veoma neefikasan jer ne postoji mogućnost otklanjanja navalnih grešaka. Takođe nemoguće je detektovati grešku u slučaju da je broj oštećenih bitova paran. Zato se ovaj mehanizam često koristi uz takozvanu vertikalnu parnost koja se sastoji u sledecem: Za paket podataka od N bajtova formira se dodatni bajt vertikalne parnosti koji za svaki n-ti bit (n=0-7) u svim bajtovima formira bit parnosti.

0	**0**	**1**	**0**	**1**	**1**	**0**	**0**
1	**1**	**1**	**0**	**1**	**1**	**1**	**1**
0	**1**	**0**	**0**	**0**	**1**	**0**	**0**
1	**0**	**0**	**0**	**0**	**1**	**1**	**1**

← vertikalna parnost

Korišćenje vertikalne parnosti u kombinaciji sa bitom parnosti nudi mogućnost ne samo otkrivanja greške već i njeno ispravljanje ali samo u slučaju da se radi o neparnom broju grešaka kod neparnog broja bajtova. Ispravljanje se jednostavno vrši invertovanjem n-tog bita u N-tom bajtu ukoliko je pogrešan bit parnosti za N-ti bajt na n-toj poziciji koja se određuje iz vertikalne parnosti. Naravno potrebno je da se i bitovi vertikalne i horizontalne parnosti prenesu bez greške što nije zagarantovano jer se oni prenose preko istog komunikacionog kanala kao i korisni podaci.

Ni vertikalna ni horizontalna parnost kada se koriste zajedno nisu u potpunosti efikasne u otkrivanju grešaka. Na primer ako se na istoj bitskoj poziciji u nekoliko bajtova pojavi greška ona će biti otkrivena samo ukoliko u tim bajtovima ne bude paran broj grešaka. Isti slučaj je i kada je vertikalna parnost validna a horizontalna ne. Dakle da bi se detektovala greška potrebno je da jedna od parnosti bude

validna (ukoliko se desio neparan broj grešaka). Za ispravljanje grešaka je potrebno da i jedna i druga parnost pravilno ukazuju na pozicuju greške.
Postoji još jedan problem koji se odnosi na detekciju lažnih grešaka ukoliko se vertikalna i horizontalna parnost koriste zajedno. To se dešava kada su narušeni bitovi koji se nalaze u različitim vrstama i kolonama kao na slici:

1	1	0	0	0
0	**1**	0	**0**	1
0	**1**	0	**1**	1
1	0	1	1	0

Podebljanim slovima su predstavljeni pogrešno preneti bitovi. Na prijemnoj strani će se uz pomoć parnosti otkriti greška u drugom i trećem redu kao i u drugoj i četvrtoj koloni. Zbog toga će biti pogrešno detektovane greške na pozicijama u drugom redu četvrtoj koloni i u trećem redu drugoj koloni. Ovo je posledica toga što se greška trazi u preseku pozicija vertikalne i horizontalne parnosti.

Postoje i mnogo bolje ali i složenije tehnike za prevenciju grešaka kao sto su čeksuma i CRC (kod USB-a).
Na bazi rs-232 razvijeni su i drugi standardi koji se razlikuju samo na fizičkom nivou komunikacije kao na primer rs-485, rs-422 i drugi.

2.3 Usb serijski interfejs

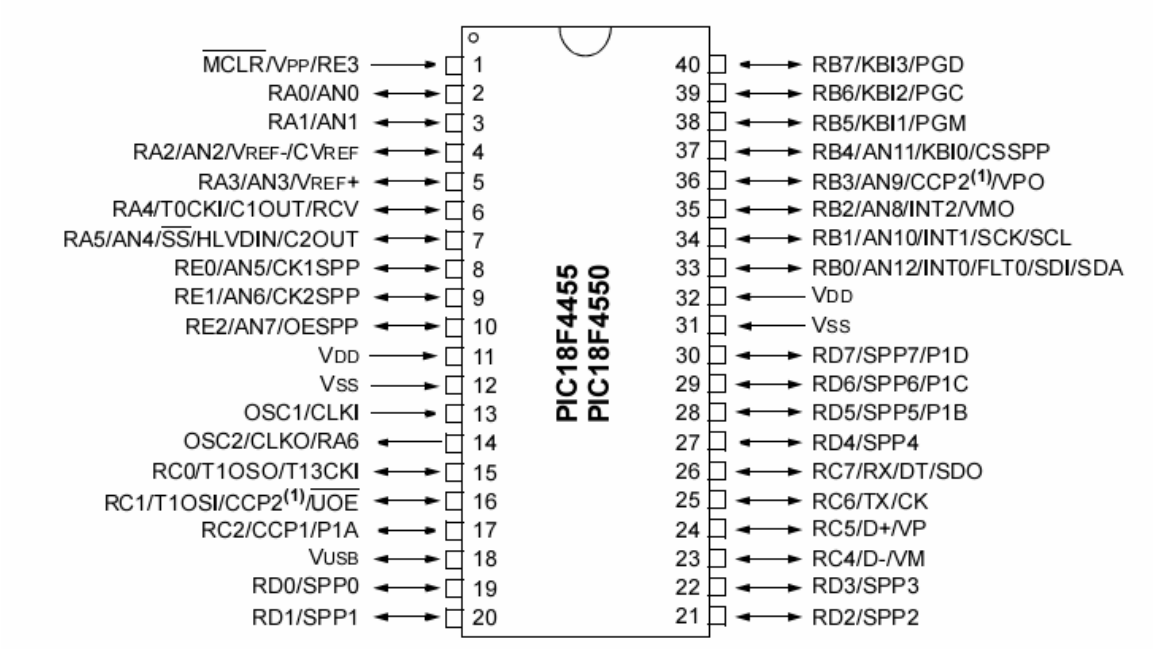
Danas najkorišćeniji vid serijske komunikacije je usb. Odlikuje se veoma velikim brzinama prenosa koje se kreću od 1.5 Mbit/s kod sporog i 12MB/s kod brzog prenosa pa sve do neverovatnih 480 Mbit/s kod USB 2.0 standarda. Usb je razvijen za komunikaciju raznih ulzno izlaznih uređaja sa računarom po principu *plug&play* (ubaci i koristi). Danas se usb podrška može naći u gotovo svim novijim mikrokontrolerima kao što je slučaj i kod PIC 18F4550.
Usb prenosi podatke preko D- i D+ linija i zato spada u grupu diferencijalnih načina prenosa. Veći napon na D+ u odnosu na liniju D- definiše logičku nulu i obrnuto. Zbog toga je smanjen i uticaj šuma jer se on poništava zbog diferencijalnog načina prenosa. Na ovaj način je fizički ostvarljivo prenošenje podataka velikim brzinama.
Usb protokol je za razliku od rs-232 mnogo apstraktniji i organizovan je u više nivoa i zato je veoma komplikovan za implementaciju.

Komunikacija se odvija po principu master slave. Master je host tj. računar koji može komunicirati sa do 127 usb uređaja i obezbediti napajanje od 500 mA za njihov rad. Zato se svaki tok podataka kod usb-a definiše u odnosu na host.
Najbitniji deo komunikacije je enumeracija koja se dešava oko 100 ms posle priključenja uređaja na host. Tada računar prikuplja podatke o uređaju preko posebnih struktura podataka koje se zovu deskriptori. Oni sadrže podatke vezane za ime i proizvođača uređaja, njegovu potrošnju, format podataka koji uređaj koristi prilikom prenosa. U procesu enumeracije host na osnovu primljenih podataka (deskriptora) od strane uređaja određuje drajver uređaja koji će se koristiti i uređaju zatim dodeljuje redni broj od 1 do 127. Komunikacija se odvija preko posebnih bafera za komunikaciju koji se nazivaju endpointi. Njihov broj varira u zavisnosti od uređaja. Najbitniji je nulti endpoint jer se njime prenose kontrolne informacije i vrši enumeracija.
Prilikom enumeracije host prima deskriptore od uređaja koji se šalju kontrolnim transferom. Najpre se šalje *device* deskriptor posle čega host resetuje uređaj. Host potom zahteva slanje konfuguracionog deskriptora. Tu se specificira da je uređaj HID tipa kako bi host pronašao odgovarajući upravljački program. Na kraju slede report deskriptori koji opisuju šablon po kome se šalju podaci.
U našem projektu smo koristili Windows-ov ugrađeni usb drajver HID koji nudi jednostavnost u implementaciji ali ima ogranicenja u pogledu brzine i veličine bloka podataka za prenos. HID drajver se koristi za komunikaciju sa standardnim U/I uređajima kao što su miš , tastatura , skener i drugi. Njegova ograničavajuća brzina od 64 kb/s je sasvim zadovoljavajuća za našu aplikaciju. Da bi host tretirao priključeni uređaj kao Hid kompatibilan potrebno je da se u postupu enumeracije pošalje odgovarajući skup deskriptora. Podaci se šalju u paketima. Svaki paket sadrži identifikator PID koji određuje koji tip paketa se prenosi (token,data,handshake). Nakon toga slede podaci (opciono).
Za detekciju grešaka koristi se CRC postupak koji je za razliku od bita parnosti gotovo stoprocentno efikasan. Zahvaljujući tome kao i diferencijalnom načinu prenosa postižu se velike brzine prenosa. Takođe se koristi i *handshaking* kako bi se host obavestio da li je paket uspešno prenet (Ack i Nak) ili da li je došlo do zagušenja (Stall). Postoje četiri načina prenosa u zavisnosti od brzine i obima prenosa. Hid interfejs podržava samo *control* i *interrupt* transfer čime se limitira njegova brzina prenosa.

3. Mikrokontroler PIC 18F4550

3.1 Karakteristike

Izbor odgovarajućeg mikrokontrolera za ovaj projekat je bio uslovljen pre svega na podršku za usb komunikaciju dok je asinhroni serijski modul implementiran u gotovo svim novijim mikrokontrolerima. Zbog toga je upotrebljen PIC 18F4550 koji po karakteristikama premašuje poptrebe projekta ali svojom niskom cenom i dostupnošću na tržistu nije imao alternativu.
Pic 18F4550 pripada 18F seriji mikrokontrolera kompanije Microchip. Mikrokontroleri ovog proizvođača se odlikuju malom cenom i što je najvažnije najboljom besplatnom tehničkom podrškom (kompajleri, razvojni sistemi programatori). Ovi mikroprocesori imaju *harvard* strukturu pa je memorijska mapa podeljena na programsku i memoriju za podatke kao i EEPROM. CPU koristi tehniku preklapanja kako bi se sve instrukcije (osim grananja) izvršavale jedan ciklus. Zbog toga se osnovni takt deli sa 4 jer se faze izvršenja naredbi preklapaju. Sve naredbe su fiksne dužine od 2 bajta tako da je adresiranje memorije ograničeno. Zbog toga se memorija deli na 16 stranica a izbor stranice se vrši u odgovarajućim kontrolnim registrima. Ova osobina značajno usporava rad mikrokontrolera međutim napredniji kompajleri vrše pametno planiranje raspodele memorije kako bi se varijable koje se zajedno koriste nalazile u istoj memorijskoj banci. Programska memorija je 32KB dok je RAM veličine 2 KB. Takođe postoji i 256B EEPROM-a. Procesor poseduje prošireni skup instukcija u odnosu na ranije serije (16 i 17) kao i nove načine adresiranja. Tako su dodate naredbe za hardversko množenje i deljenje, inkrementiranje i dekrementriranje sa uslovnim skokom, naredbe za čitanje tabela i druge. Programski brojač je širine 21 bit i njemu se može pristupati samo indirektno preko određenih registara. Procesor poseduje i stek ali se on nažaost može koristiti samo indirektno tako što se u poseban registar upisuje željeni sadržaj i potom posebnom instrukcijom sadržaj tog registra stavlja na stek. Kod čitanja sa steka vrednost se takođe nalazi u tom registru. Oscilator pruža brojne mogućnosti prilikom izbora radnog takta koji je ujedno i takt za periferije. Maksimalni eksterni takt je 48MHZ što daje cpu takt od 12MHZ. Za to se koristi PLL kolo i delitelji frekvencije. Najbitnije je da se za rad usb modula mora obezbediti takt od 24MHZ , a PIC ima prednost u odnosu na konkurenciju što taj takt može biti nezavistan od takta za CPU i druge jedinice. Mehanizam interapta je organizovan kao jedan interapt vektor koji sadrži adresu prekidne rutine u kojoj se treba ispitati izvor prekida i preduzeti željena akcija, dakle nema interapt vektora za svaki ili grupu izvora prekida što je jedan od nedostataka ovog mikrokontrolera jer se time gubi na brzini i preglednosti koda.
Ovaj mikrokontroler ima bogat skup hardverskih periferija koje mu omogućavaju primenu u gotovo svim aplikacijama.
Mikrokontroler poseduje 35 izlazno ulaznih linija koje se multipleksiraju sa *data* registrima portova i signalima drugih modula kao što je prikazano na slici:



Od perifernih modula Pic18f4550 ima: veoma napredan *USB 2.0* kontroler velike brzine sa čak 1kB memorije za podatke, 10 bitni 13-to kanalni AD konvertor, 4 tajmerske jedinice (dve imaju mogucnost PWM-a), SPI , I2C i USART modul kao i analogni komparator. Radna frekvencija je do 48MHz dok je efektivna 4 puta manja zbog *pipelining*-a. Moduli koji se koriste u ovom projektu su USART modul i USB pa ćemo ukratko opisati njihove karakteristike.

3.2 USART modul

Univerzalni sinhroni/asinhroni serijski prmopredajnik ima mogućnosti *half* i *full duplex* prenosa, automatske detekcije i kalibracije bodove brzine. Za komunikaciju se koriste IO pinovi RC6 kao TX i RC7 kao RX za asinhroni prenos. Rad modula se kontroliše pomoću 3 registra. To su:

TXSTA : Transmit status and control registar sa sledecim bitovima:

CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D

CSRC - nebitan kod asinhronog prenosa
TX9 – “1” uključuje prenos devetog bita
TXEN – predaja dozvoljena sa setovanjem ovog bita
SYNC – ”0” je sinhroni a “1” asinhroni mod rada
BRGH– setuje veliku brzinu prenosa

TRMT – status predajnog pomeračkog registra, “1” registar je prazan
TX9D– deveti bit za prenos (najcešće parnost)

TCSTA : Receive status and control registar sa sledecim bitovima:

SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	c

SPEN – ukljuvuje/iskljucuje serijski port
RX9 – “1” uključuje prenos devetog bita
SREN – nebitan kod asinhronog prenosa
CREN – prijem podataka uključen setovanjem ovog bita
ADDEN – adress detect enable bit
FERR – framing eror bit za detekciju frame grešaka
OERR – overrun eror bit za detekciju overrun grešaka
RX9D– deveti bit za prenos (najcešće parnost)

BAUDCON:Baud rate control register

ABDOF	RCIDL	RXDTP	TXCKP	BRG16	–	WUE	ABDEN

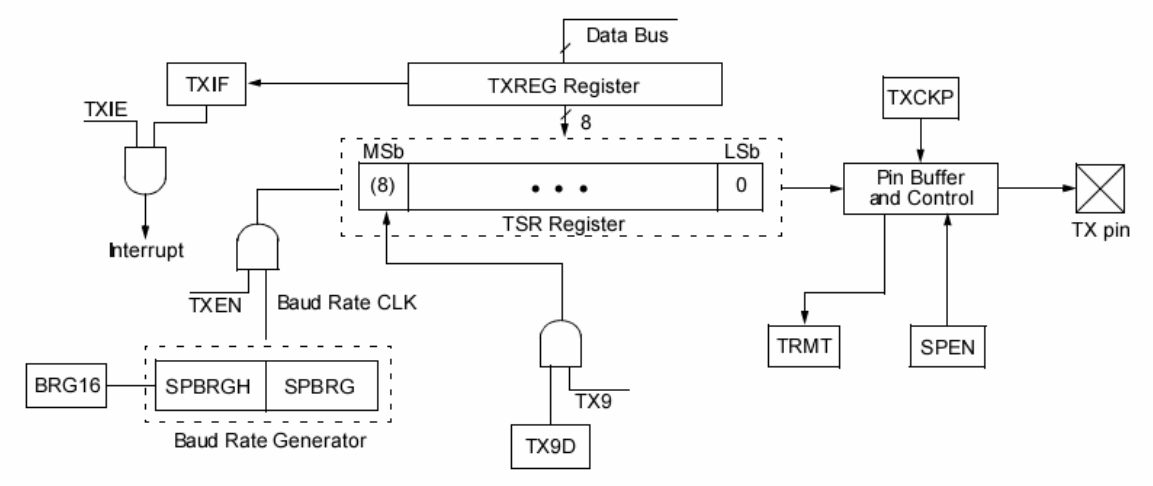
ABDOF – auto baud acquisition rollover bit
RCIDL – receive operstion idle status bit
RXDTP – ako je “1” rx podaci su invertovani
TXCKP – ako je “1” tx podaci su invertovani
BRG16– uključuje 16-bitni generator bodove brzine
WUE – wakeup enable bit
ABDEN – uključuje merenje bodove brzine sledećeg karaktera (55h)

Izbor takta se vrši setovanjem odgovarajućih bitova i upisom vrednosti u registarski par SPBRGH:SPBRG. U zavisnosti od kombinacije koristi se sledeća tabela za računanje željene brzine asinhrone komunikacije:

Konfiguracioni bitovi			**BRG/EUSART mod**	**Formula za računanje brzine**
SYNC	BRG16	BRGH		
0	0	0	8 bitni	Fosc/[64(n+1)]
0	0	1	8 bitni	Fosc/[16(n+1)]
0	1	0	16 bitni	Fosc/[16(n+1)]
0	1	1	16 bitni	Fosc/[4(n+1)]

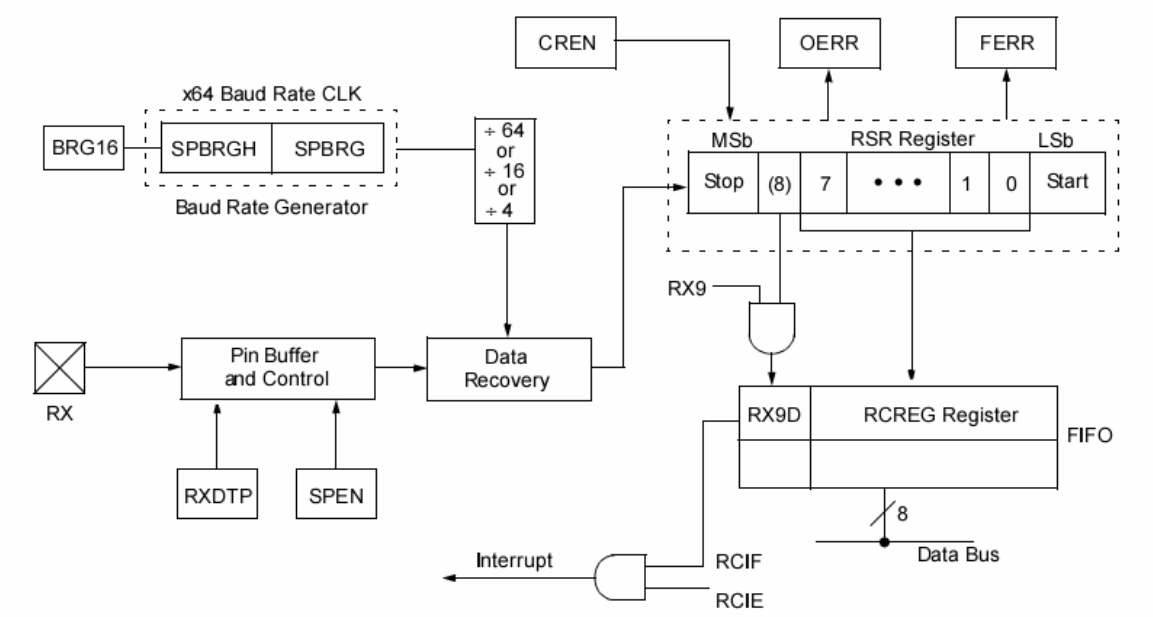
U dokumentaciji se mgu naći tablice sa standardnim bodovim brzinama i procentualnim odstupanjima za različite konfiguracije takta procesora.

Prenos podaka se vrši preko predajnog i prijemnog kola asinhronog primopredajnika.
Predajno kolo je prikazano na sledećoj slici:



TXREG sadrzi 8-bitni podatak koji se treba poslati. Kada se u ovaj registar upiše neka vrednost njegov sadržaj se prebacuje u TSR registar za šiftovanje. Tada je TXREG prazan što može izazvati interapt. TSR registar se ne puni podatkom iz TXREG sve dok se ne prenese poslednji stop bit. TSR je duzine 9 bita. Prvih 8 je vrednost koja se uzima iz TXREG dok je 9-ti bit bit parnosti (opciono). Kod prenosa svih 9 bitova prvo je potrebno upisati 9 bit pa onda vrednost u TXREG kako bi se izbeglo da se 9-ti bit upiše posle napuštanja 8-bitne vrednosti iz TSR registra. TSR se taktuje signalom iz baud rate generatora koji je ustvari klok za šiftovanje. Ovaj registar je vezan za odgovarajuci pin preko koga se prenose serijski podaci.

Prijemno kolo je slične konstrukcije i radi po sličnom principu.

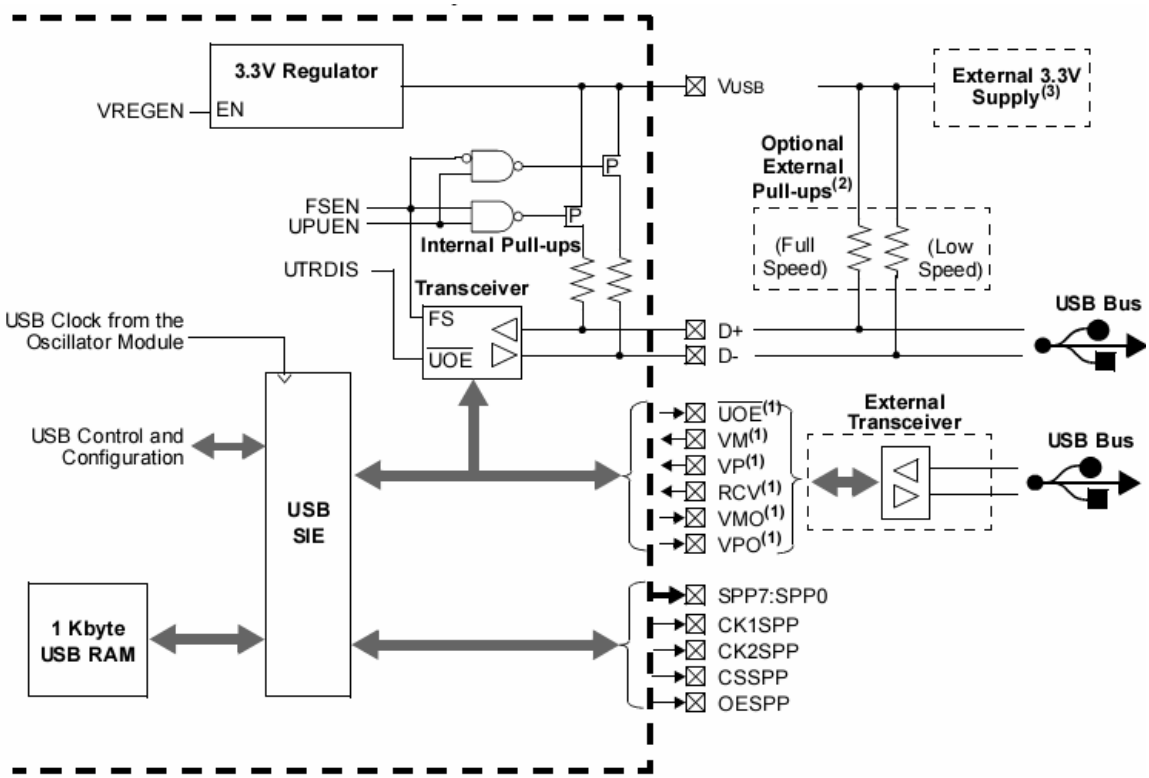


Podaci sa ulaza se vode u *datarecovery* kolo koje radi na 16 puta većem taktu od bodove brzine kako bi se izbegle greške jer je brzina uzorkovanja veća od bodove brzine. Primljeni podaci se pomeraju kroz prijemni šift registar RSR dužine 9 bita. Posle toga se ovim podacima može pristupiti preko RCREG registra dok se kao i kod predaje 9-tom bitu pristupa preko kontrolnog registra. Kada se RCREG napuni vrednošću iz RSR može se generisati interapt kako bi se obradili preuzeti podaci.

Iz prethodnog se može zakljuciti da se za serijsku komunikaciju mogu koristiti dva načina rada. Jedan je zasnovan na prekidima koji se generišu pri završetku slanja ili prijema svakog bajta tako da se u interapt potprogramu može prihvatiti i obraditi primljeni podatak ili pripremiti slanje sledećeg bajta. Drugi način je sličan samo se umesto interapta koristi prozivka (engl *pooling*) statusnih bitova koji ukazuju na završetak transfera jednog bajta.

3.3 USB modul

Najbitnija karakteristika PIC18F4550 koja ga izdvaja od ostalih iz familije je usb komunikacioni modul i interfejs. Usb komunikacija je veoma složena i nije jedinstvena zbog čega ćemo samo ukratko opisati osobine usb modula korišćenog mikrokontrolera. Blok šema modula prikazana je na slici:



Fizička veza ostvaruje se preko dva pina D+ i D- dok se ostali koriste za povezivanje eksternog transivera. Podaci se primaju i prenose preko posebnog bafera kapaciteta 1KB koji je podeljen na 16 tzv. endpointa i njihove deskriptore koji sadrze podatke o karakteru endpointa (smeru toka podataka), veličini endpointa i njegovoj početnoj adresi. Pristup podacima koji se primaju ili šalju se obavlja preko ovih endpointa. USB serijski interfejs vrši primopredaju podataka i on mora biti taktovan frekvencijom od 24MHz. Modul ima ugradjen 3.3V regulator koji služi za napajanje *pull up* otpornika na D+ liniji što je bitno kod početne faze prepoznavanja od strane hosta. Njime se određuje i brzina prenosa (spor ili brz transfer).

Komunikacija se zasniva na interaptima kojih ima nekoliko. Za komunikaciju je najvažniji onaj koji se generiše kada se primi neki token (podatak). Tada se u odgovarajućim registrima može pročitati u kom endpointu se podatak nalazi kako bi mu se pristupilo. Postoje i drugi prekidi koji se okidaju u slučaju greške , reseta, zagušenja ili aktivnosti na liniji. Svi oni se mogu iskoristiti za kontrolu komunikacije.

Zbog svoje kompleksnosti koja uglavnom višestruko premašuje složenost većine aplikacija rad sa usb portom i mikrokontrolerom se najčešće izvodi pomoću gotovih modula. Pogotovo treba imati u vidu da je za oživljavanje usb komunikacije potrebno pisanje posebnog drajvera uređaja na strani hosta kao i odgovarajućeg softvera u mikrokontroleru. Zahvaljujući pomenutom HID standardu i gotovim programskim modulima ovaj veliki problem se veoma lako

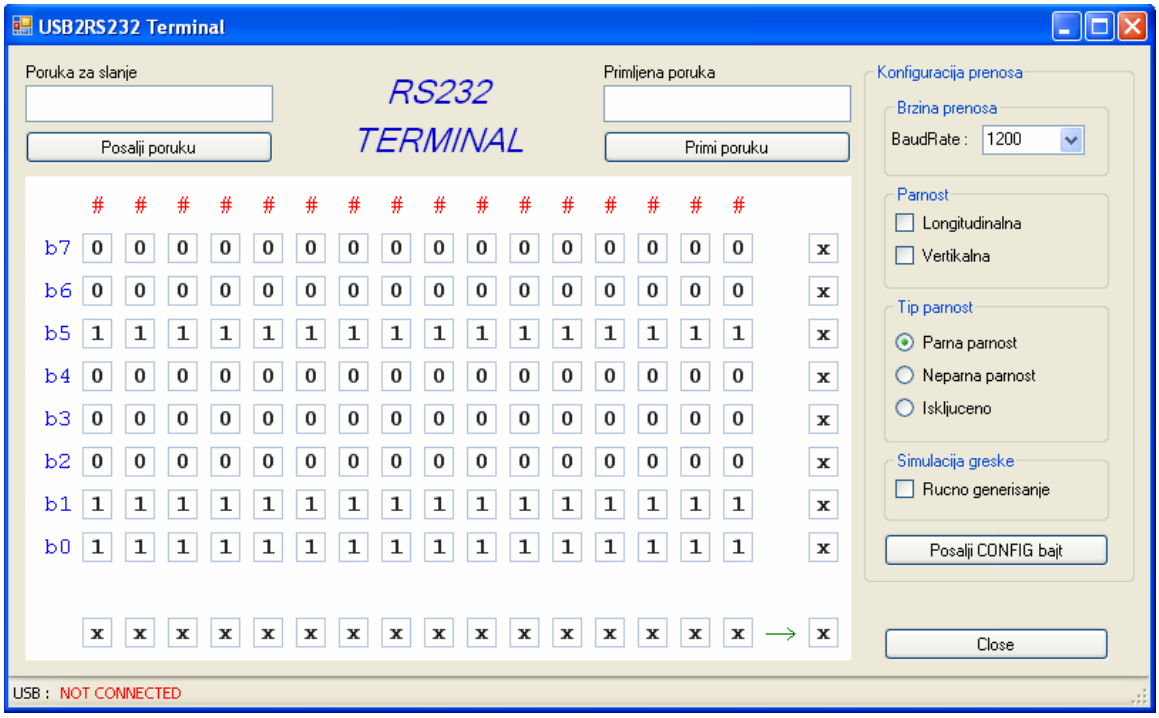
rešava pogotovo ako se radi sa PIC mikrokontrolerima. Sve detaljnije informacije vezane za usb komunikacioni standard se mogu naći na zvaničnom sjtu standarda *www.usb.org*.

4. Aplikacioni program računara

4.1 Razvojno okruženje

Računarska aplikacija nudi interfejs kojim se može vršiti kontrola parametara komunikacije i unositi poruka za slanje ili čitati poruka koja je primljena. Softver je razvijen u programskom okruzenju .NET u programskom jeziku C#. Korisnički interfejs je veoma jednostavan i lak za korišćenje. Za rad programa je potrebno da bude instalirana 2.0 verzija .NET *framework*-a pod operativnim sistemom WINXP SP2 kako bi program mogao da se kompajlira i izvršava.

Izgled prozora aplikacije:



U softver je ugrađena bibilioteka za rad sa usb portom koja radi po HID standardu čime je olakšana komunikacija sa mikrokontrolerom. Nedostatak ove biblioteke je što ograničava veličinu paketa na 8 bajtova što za nas nije veliki problem jer brzina komunikacije nije od presudnog značaja. Razvoj ovakvih komunikacionih funkcija za usb je veoma napredan i zahteva obilje tehničke dokumentacije i programiranja. Od biblioteka funkcija se koristi i GDI za iscrtavanje polja bitova u radnoj površini prozora.

4.2 Uputstvo za rad sa programom

Rad sa terminalom je jednostavan. U gornjem levom uglu se nalazi polje za unos gde se može uneti tekstualna poruka koja se želi poslati i koja će se tretirati kao niz ascii vrednosti. Neposredno posle unosa svakog slova menja se njegova bitska reprezentacija u sredini ekrana. Korisnik može uneti maksimalno 16 znakova. Neuneti znakovi se tretiraju kao karakter #.
Prilikom prijema poruke sadržaj poruke se može videti u tekstualnom formatu u drugom polju za unos. Binarna reperezentacija primljene poruke se prikazuje takođe grafički gde se mogu videti i vertikalna i horizontalna parnost. Na taj način se može utvrditi da li je paket primljen bez greske i eventualno postojeca greška otkloniti. Ukoliko je otkriven pogrešno preneti bit pomoću ovih tehnika detekcije, on se ispisuje crvenom bojom kako bi bio laše uočljiv.
Sa desne strene prozora nalazi se deo u kome se podešavaju parametri komunikacije. Može se izabrati jedna od četiri ponuđene bitske brzine iz padajućeg menija (1200,2400,9600 i 19200 kbps). Zatim se može izabrati način detekcije grešaka. Mogu se izabrati longitudinalna (horizontalna) ili vertikalna parnost ili obe. Ukoliko nije izabrana ni jedna parnost grafički prikaz bitova za ove parnosti će biti “x”. Ispod ovih setovanja se može odrediti karakter izabranih parnosti odnosno da li su izabrane parnosti parne ili neparne. Ukoliko se štiklira stavka “isključeno” neće se koristiti parnost, a polja za izbor tipa parnosti će biti zasenčena.
Štikliranjem stavke “ručno generisanje” korisniku se omogućava da direktno manipuliše bitovima kreirane poruke uključujući i bitove u vertikalnij i horizontalnoj parnosti. Tako se nudi mogućnost simulacije grešaka i posmatranje efekata koji se pri tome postižu posmatranjem primljenih podataka i bitova parnosti. Ova manipulacija se vrši klikom miša na određeni bit tj. njegov grafički prikaz kada se njegova vrednost invertuje.
Pre slanja treba kliknuti na dugme “pošalji konfiguracioni bajt” kojim se šalju kontrolni podaci mikrokontroleru koji opisuju karakter komunikacije koji opisuju izabrani parametri. Ispod ovog dugmeta se nalazi dugme kojim se izlazi iz aplikacije.
U statusnoj liniji na dnu prozora nalazi se informacija o statusu veze usb porta i mikrokontrolera čime se može detektovati da li je softver prepoznao komunikacioni hardver na usb portu.

4.3 Implementacija programa

Najkomplekniji deo programa je iscrtavanje polja bitva unete poruke i interakcija sa korisnikom na pritisak tasterom miša. Ova funkcija se realizuje tako što se nakon unosa ili prijema podataka koji su u ascii formatu odredi njihova binarna reprezentacija dužine 8 bita. Ovo se izvodi tako što se osmobitna vrednost svakog karaktera “and”-uje sa vrednošću $80 i ispituje se da li je različita od

nule. Da bi se ispitao svaki bit ascii vrenost se pomera za jedno mesto levo u svakoj iteraciji.

```
for (int i = 0; i < 16; i++)
        {
          for (int j = 0; j < 8; j++)
          {
            if (((slovaZaSlanje[i] << j) & 128) != 0)
              asciiSlova[j, i] = 1;
            else
              asciiSlova[j, i] = 0;
          }
        }
```

Za svaki tekst koji se želi poslati izračunava se parnost. Vertikalna i horizontalna parnost se određuju na strani računara i kao posebni bajtovi se prenose u mikrokontroler. Razlog njihovog generisanja na strani racunara a ne u mikrokontroleru je to što se u programu može vršiti direktno postavljanje vrednosti određenih bitova radi simulacije grešaka pa bi na primer računanje horizontalne parnosti u mikroprocesoru bilo bez unete namerne greške. Parnost se realizuje jednostavnim brojanjem bitova u svakom karakteru kao i na svakoj poziciji za te karaktere.

Za horizontalnu parnost se koristi sledeći skup operacija:

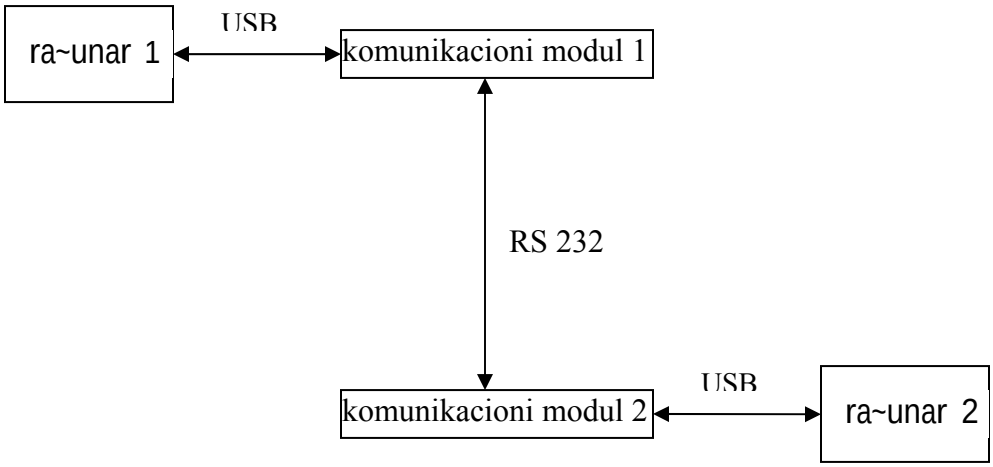
```
for (int i = 0; i < 16; i++)
          {
            brojJedinica = 0;
            for (int j = 0; j < 8; j++)
            {
              if (asciiSlova[j, i] == 1)
                brojJedinica++;
            }
            if ((brojJedinica % 2) == 0)
              lParnost[i] = 0;
            else
              lParnost[i] = 1;
          }
```

Isti princip se koristi i za vertikalnu parnost mada je moguće jednostavnije rešenje korisšćenjem logičke operacije *exor* međutim taj način nije pogodan zbog potrebe prikaza na ekranu.
Slanje podataka na usb port se obavlja upotrebom programske komponente za usb koja omogućava slanje u paketima od po 8 bajtova. Poziv funkcije za slanje je u obliku : this.usb.SpecifiedDevice.SendData(data); gde je data niz bajtova za slanje. Prijem se obavlja tako što se generiše događaj u kome se isčitava prijemni bafer. Pre rada sa usb uređajem potrebno je proveriti da li takav postoji za šta se koristi funkcija usb_OnSpecifiedDeviceArrived koja generiše događaj kada se u usb priključi uređaj koji ima zadato ime.

5. Komunikacioni model sistema

5.1 Blok šema

Asinhrona serijska komunikacija između dva računara se obavlja ne direktno već preko interfejsa odnosno mikrokontrolera koji se koristi za prijem podataka sa računara i njihovo slanje drugom mikrokontroleru i obrnuto.
Blok šema sistema koji smo realizovali data je na slici:

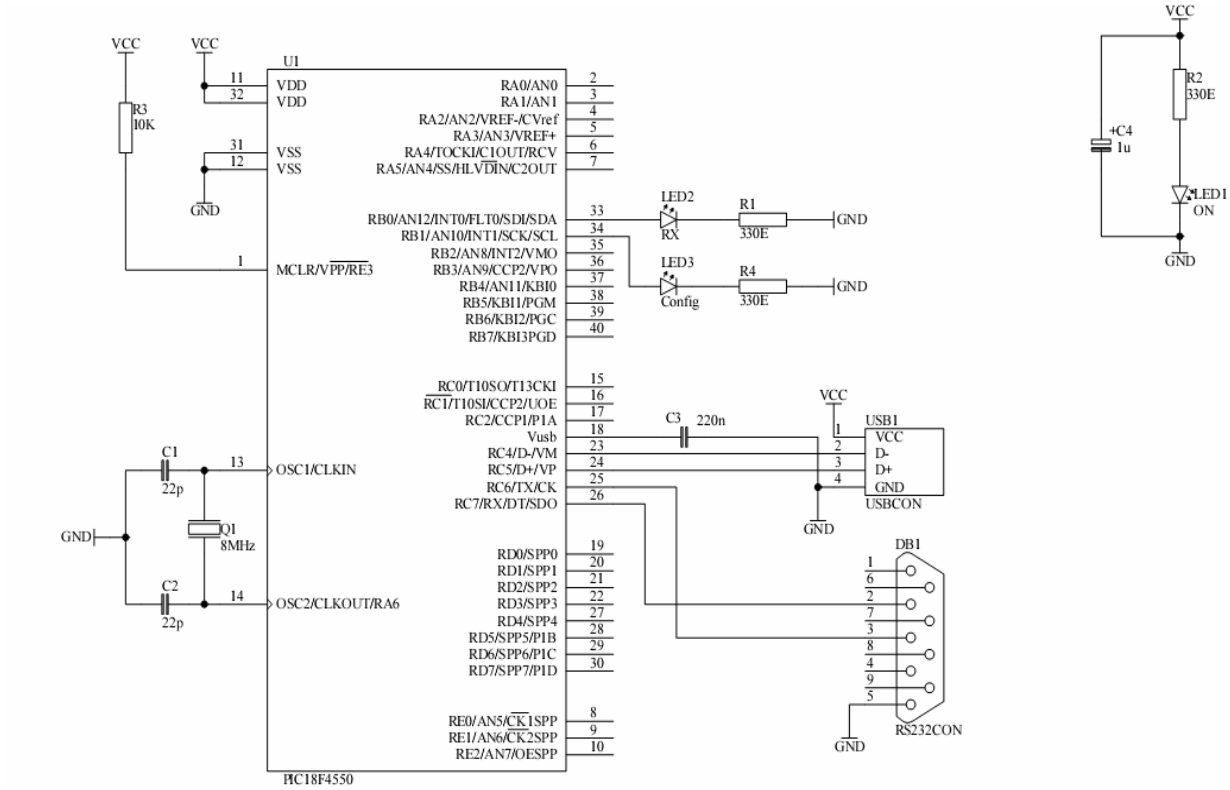


Računari 1 i 2 su povezani indirektno preko komunikacionih modula (hardvera) . Računari se koriste kao terminali na kojima se vrši unos slanje i prijem podataka i na taj način prati uspešnost komunikacije. Za vezu računara sa mikrokontrolerom koji ustvari predstavlja komunikacioni modul koristi se usb interfejs jer je on najpogodniji za implementaciju i nudi napajanje za komunikacioni hardver tj. mikrokontroler. Svi podaci koji se šalju najpre se preko usb veze proslede mikrokontroleru. Veoma je bitano slanje konfiguracione reči koja služi da se njome setuje brzina komunikacije. U zavisnosti od izabrane brzine softver u mikrokontroleru podešava svoje kontrolne registre da bi se ostvarila komunikacija sa željenim karakteristikama. Kada mikrokontroler obavi podešavanja i konfigurisanje za prenos u tekućoj sesiji na hardverskom modulu se upali ya kratko narandžasta led dioda kojom se korisnik informiše da moze poslati poruku. Primljene podatke mikrokontroler šalje drugom mikrokontroleru koristeći asinhroni prenos preko svog USART modula. Kada se prime podaci oni se preko usb veze predaju drugom računaru tako da se njihov sadržaj može videti na ekranu i na taj način utvrditi da li je bilo grešaka u prenosu. Kada modul primi neki paket podataka on obaveštava korisika da je primio paket paljenjem crvene led diode.

Tada korisnik moze pročitati sadržaj paketa iz komunikacionog softvera klikom na dugme za prijem kada će podaci biti prebačeni u računar putem usb komunikacije.

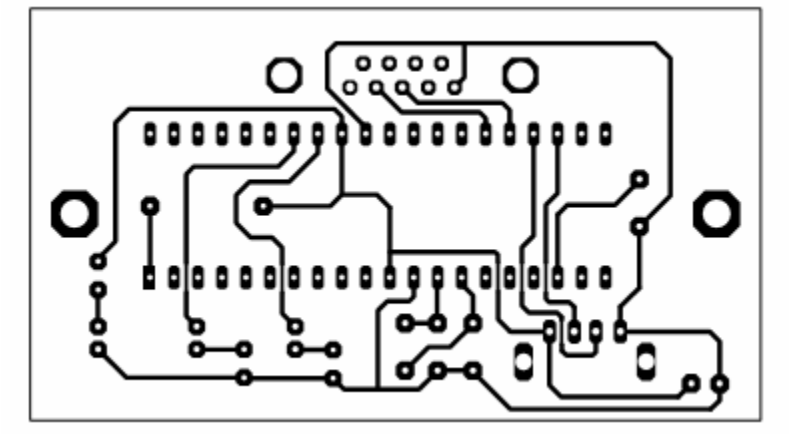
5.2 Hardverski modul

Hardver za oba komunikaciona terminala je isti. Šema veze je jednostavna i data je na slici :

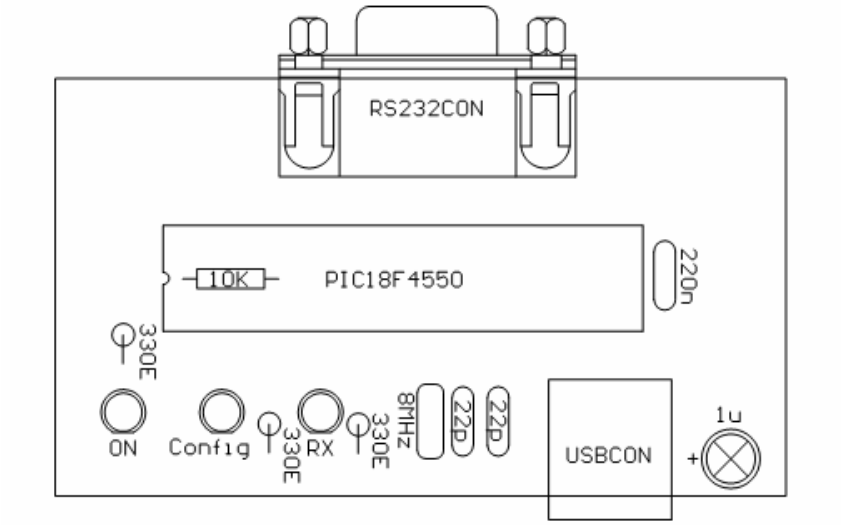


Na pinove OSC1 i OSC2 su priključeni kvarni oscilator od 8MHZ i kondenzatori od po 22pF koji zajedno sa internim invertorom procesora čine Pierce-ov oscilator koji taktuje procesor i interne module mikrokontrolera. Kondenzator C3 je vezan u konfiguraciju za usb uređaj napajan sa magistrale a njegovu vrednost propisuje proizvođač kao i vrednosti za C1 i C2. MCLR pin je preko *pull- up* otpornika vezan za Vcc jer je to pin za reset sa aktivnom logičkom nulom. PortB se koristi za upravljanje led diiodama koje daju statusne informacije o prijemu konfiguracione reči. Napon napajanja dolazi sa usb konektora odnosno računara i iznosi 5V pa se zato za led diode koriste otpornici od 330 oma. Yelena led dioda služi kao indikator prisutnosti ovog napajanja.

Stampana ploča je malih dimenzija, kao i električna šema izrađena je u softverskom alatu Protel.



Montažna šema je data na sledećoj slici:



Na pločici se nalaze konektor za usb tipa A i DSub-9 konektor za rs-232 vezu. Za povezivanje sa računarom i drugim modulom koriste se standardni komercijalni kablovi.

Na sledećim slikama je izgled modula u kutiji:

6. Softver mikrokontrolera

Najčešći programski jezik za programiranje mikrokontrolera u današnje vreme je C zbog toga smo za naš projekat koristili besplatni softverski razvojni alat CSC. Kao hardverski alat upotrebljen razvojni sistem easy Pic5 domaćeg proizvođača mikroelektronika. Pogodnost ovog razvojnog alata je to što nudi gotove komponente za komunikaciju, posebno za usb. Serijski prenos je zbog standardizacije sa C jezikom realizovan pomoću naredbi *getc*i *putc.*

Softver za mikrokontroler je sadržan u 3 fajla:

1) Fajl usb_desc_hid 8-byte koji sadrži deskriptore za usb:

```
////////////////////////////////////////////////////////////////////////
///                     usb_desc_hid.h                              ////
////                                                                ////
//// An example set of device / configuration descriptors for use with ////
//// CCS's HID Demo example (see ex_usb_hid.c)                       ////
////                                                                ////
////////////////////////////////////////////////////////////////////////
////                                                                ////
//// Version History:                                               ////
////                                                                ////
//// June 20th, 2005:                                               ////
////   PIC18Fxx5x initial release                                       ////
////                                                                ////
//// March 21st, 2005:                                                ////
////   EP 0x01 and EP 0x81 now use USB_EP1_TX_SIZE and USB_EP1_RX_SIZE ////
////     to define max packet size, to make it easier for dynamically ////
////     changed code.                                              ////
////   EP 0x01 and EP 0x81 will now use 1ms polling interval if using  ////
////     a full speed device.                                       ////
////                                                                ////
//// May 4th, 2004: Optimization and cleanup.                         ////
////               Some definitions may have changed.               ////
////                                                                ////
//// May 6th, 2003: Fixed non-HID descriptors pointing to faulty       ////
////               strings                                          ////
////                                                                ////
//// August 2nd, 2002: Initial Public Release                         ////
////                                                                ////
////                                                                ////
////////////////////////////////////////////////////////////////////////
////       (C) Copyright 1996,2005 Custom Computer Services          ////
//// This source code may only be used by licensed users of the CCS    ////
//// C compiler.  This source code may only be distributed to other    ////
//// licensed users of the CCS C compiler.  No other use,              ////
//// reproduction or distribution is permitted without written         ////
//// permission.  Derivative programs created using this software      ////
//// in object code form are not restricted in any way.                ////
////////////////////////////////////////////////////////////////////////

#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include <usb.h>

  ////////////////////////////////////////////////////////////////
  ///
  ///  HID Report.  Tells HID driver how to handle and deal with
  ///  received data.  HID Reports can be extremely complex,
```

```
///  see HID specifcation for help on writing your own.
///
///  CCS example uses a vendor specified usage, that sends and
///  receives 2 absolute bytes ranging from 0 to 0xFF.
///
//////////////////////////////////////////////////////////////////////

const char USB_CLASS_SPECIFIC_DESC[] = {
  6, 0, 255,    // Usage Page = Vendor Defined
  9, 1,         // Usage = IO device
  0xa1, 1,      // Collection = Application
  0x19, 1,      // Usage minimum
  0x29, 8,      // Usage maximum

  0x15, 0x80,       // Logical minimum (-128)
  0x25, 0x7F,       // Logical maximum (127)

  0x75, 8,      // Report size = 8 (bits)
  0x95, 8,      // Report count = 8 bytes
  0x81, 2,      // Input (Data, Var, Abs)
  0x19, 1,      // Usage minimum
  0x29, 8,      // Usage maximum
  0x75, 8,      // Report size = 8 (bits)
  0x95, 8,      // Report count = 8 bytes
  0x91, 2,      // Output (Data, Var, Abs)
  0xc0          // End Collection
};

//if a class has an extra descriptor not part of the config descriptor,
// this lookup table defines where to look for it in the const
// USB_CLASS_SPECIFIC_DESC[] array.
//first element is the config number (if your device has more than one config)
//second element is which interface number
//set element to 0xFFFF if this config/interface combo doesn't exist
const int16 USB_CLASS_SPECIFIC_DESC_LOOKUP[USB_NUM_CONFIGURATIONS][1] =
{
//config 1
  //interface 0
    0
};

//if a class has an extra descriptor not part of the config descriptor,
// this lookup table defines the size of that descriptor.
//first element is the config number (if your device has more than one config)
//second element is which interface number
//set element to 0xFFFF if this config/interface combo doesn't exist
const int16 USB_CLASS_SPECIFIC_DESC_LOOKUP_SIZE[USB_NUM_CONFIGURATIONS][1] =
{
//config 1
  //interface 0
    32
};


//////////////////////////////////////////////////////////////////////
///
///  start config descriptor
///  right now we only support one configuration descriptor.
///  the config, interface, class, and endpoint goes into this array.
///
//////////////////////////////////////////////////////////////////////

#DEFINE USB_TOTAL_CONFIG_LEN    41  //config+interface+class+endpoint+endpoint (2 endpoints)

const char USB_CONFIG_DESC[] = {
//IN ORDER TO COMPLY WITH WINDOWS HOSTS, THE ORDER OF THIS ARRAY MUST BE:
  //  config(s)
  //  interface(s)
  //  class(es)
  //  endpoint(s)
```

```
//config_descriptor for config index 1
    USB_DESC_CONFIG_LEN, //length of descriptor size         ==1
    USB_DESC_CONFIG_TYPE, //constant CONFIGURATION (CONFIGURATION 0x02)    ==2
    USB_TOTAL_CONFIG_LEN,0, //size of all data returned for this config     ==3,4
    1, //number of interfaces this device supports      ==5
    0x01, //identifier for this configuration.  (IF we had more than one configurations)     ==6
    0x00, //index of string descriptor for this configuration     ==7
    0xC0, //bit 6=1 if self powered, bit 5=1 if supports remote wakeup (we don't), bits 0-4 unused and bit7=1        ==8
    0x32, //maximum bus power required (maximum milliamperes/2)  (0x32 = 100mA)

//interface descriptor 1
    USB_DESC_INTERFACE_LEN, //length of descriptor     =10
    USB_DESC_INTERFACE_TYPE, //constant INTERFACE (INTERFACE 0x04)      =11
    0x00, //number defining this interface (IF we had more than one interface)   ==12
    0x00, //alternate setting    ==13
    2, //number of endpoins, except 0 (pic167xx has 3, but we dont have to use all).      ==14
    0x03, //class code, 03 = HID    ==15
    0x00, //subclass code //boot    ==16
    0x00, //protocol code     ==17
    0x00, //index of string descriptor for interface     ==18

//class descriptor 1  (HID)
    USB_DESC_CLASS_LEN, //length of descriptor   ==19
    USB_DESC_CLASS_TYPE, //dscriptor type (0x21 == HID)     ==20
    0x00,0x01, //hid class release number (1.0) (try 1.10)     ==21,22
    0x00, //localized country code (0 = none)      ==23
    0x01, //number of hid class descrptors that follow (1)     ==24
    0x22, //report descriptor type (0x22 == HID)              ==25
    USB_CLASS_SPECIFIC_DESC_LOOKUP_SIZE[0][0], 0x00, //length of report descriptor          ==26,27

//endpoint descriptor
    USB_DESC_ENDPOINT_LEN, //length of descriptor                ==28
    USB_DESC_ENDPOINT_TYPE, //constant ENDPOINT (ENDPOINT 0x05)        ==29
    0x81, //endpoint number and direction (0x81 = EP1 IN)      ==30
    0x03, //transfer type supported (0x03 is interrupt)        ==31
    USB_EP1_TX_SIZE,0x00, //maximum packet size supported               ==32,33
  #if USB_USE_FULL_SPEED
    1,  //polling interval, in ms.  (cant be smaller than 10)     ==34
  #else
    10,  //polling interval, in ms.  (cant be smaller than 10)     ==34
  #endif

//endpoint descriptor
    USB_DESC_ENDPOINT_LEN, //length of descriptor                ==35
    USB_DESC_ENDPOINT_TYPE, //constant ENDPOINT (ENDPOINT 0x05)        ==36
    0x01, //endpoint number and direction (0x01 = EP1 OUT)     ==37
    0x03, //transfer type supported (0x03 is interrupt)        ==38
    USB_EP1_RX_SIZE,0x00, //maximum packet size supported               ==39,40
  #if USB_USE_FULL_SPEED
    1
  #else
    10 //polling interval, in ms.  (cant be smaller than 10)   ==41
  #endif
};

//****** BEGIN CONFIG DESCRIPTOR LOOKUP TABLES ********
//since we can't make pointers to constants in certain pic16s, this is an offset table to find
//  a specific descriptor in the above table.

//NOTE: DO TO A LIMITATION OF THE CCS CODE, ALL HID INTERFACES MUST START AT 0 AND BE SEQUENTIAL
//      FOR EXAMPLE, IF YOU HAVE 2 HID INTERFACES THEY MUST BE INTERFACE 0 AND INTERFACE 1
#define USB_NUM_HID_INTERFACES   1

//the maximum number of interfaces seen on any config
//for example, if config 1 has 1 interface and config 2 has 2 interfaces you must define this as 2
#define USB_MAX_NUM_INTERFACES   1

//define how many interfaces there are per config.  [0] is the first config, etc.
const char USB_NUM_INTERFACES[USB_NUM_CONFIGURATIONS]={1};
```

```
  //define where to find class descriptors
  //first dimension is the config number
  //second dimension specifies which interface
  //last dimension specifies which class in this interface to get, but most will only have 1 class per interface
  //if a class descriptor is not valid, set the value to 0xFFFF
  const int16 USB_CLASS_DESCRIPTORS[USB_NUM_CONFIGURATIONS][1][1]=
  {
  //config 1
    //interface 0
      //class 1
      18
  };

  #if (sizeof(USB_CONFIG_DESC) != USB_TOTAL_CONFIG_LEN)
    #error USB_TOTAL_CONFIG_LEN not defined correctly
  #endif


////////////////////////////////////////////////////////////////////
///
///   start device descriptors
///
////////////////////////////////////////////////////////////////////

  const char USB_DEVICE_DESC[USB_DESC_DEVICE_LEN] ={
    //starts of with device configuration. only one possible
      USB_DESC_DEVICE_LEN, //the length of this report   ==1
      0x01, //the constant DEVICE (DEVICE 0x01)  ==2
      0x10,0x01, //usb version in bcd (pic167xx is 1.1) ==3,4
      0x00, //class code ==5
      0x00, //subclass code ==6
      0x00, //protocol code ==7
      USB_MAX_EP0_PACKET_LENGTH, //max packet size for endpoint 0. (SLOW SPEED SPECIFIES 8) ==8
      0x61,0x04, //vendor id (0x04D8 is Microchip, or is it 0x0461 ??)
      0x20,0x00, //product id   ==11,12  //don't use ffff says usb-by-example guy.  oops
      0x00,0x01, //device release number  ==13,14
      0x01, //index of string description of manufacturer. therefore we point to string_1 array (see below)  ==15
      0x02, //index of string descriptor of the product  ==16
      0x00, //index of string descriptor of serial number  ==17
      USB_NUM_CONFIGURATIONS  //number of possible configurations  ==18
  };


////////////////////////////////////////////////////////////////////
///
///   start string descriptors
///   String 0 is a special language string, and must be defined.  People in U.S.A. can leave this alone.
///
///   You must define the length else get_next_string_character() will not see the string
///   Current code only supports 10 strings (0 thru 9)
///
////////////////////////////////////////////////////////////////////

//the offset of the starting location of each string.  offset[0] is the start of string 0, offset[1] is the start of string 1, etc.
char USB_STRING_DESC_OFFSET[]={0,4,12};

char const USB_STRING_DESC[]={
  //string 0
      4, //length of string index
      USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
      0x09,0x04,   //Microsoft Defined for US-English
  //string 1
      8, //length of string index
      USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
      'R',0,
      'S',0,
      '2',0,
      '3',0,
      '2',0,
```

```
  //string 2
      28, //length of string index
      USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
      'T',0,
      'E',0,
      'R',0,
      'M',0,
      'I',0,
      'N',0,
      'A',0,
      'L',0,
};

#ENDIF
```

U prethodnoj konfiguraciji smo izabrali transfer 8-mo bitnih podataka u paketima dužine 8, verziju usb 1.1 standarda, ime uređaja i interval prozivke od 1ms. Ostali podaci su standardni, a detaljniji opis se može naći u dokumentaciji standarda. Zajedničko za sve deskriptore je da počinju podatkom o svojoj dužini i kodom tipa deskriptora.

2) UsbToRS232Term je fajl koji sadrži aplikacioni kod u programskom jeziku C.

```
#define __USB_PIC_PERIF__ 1

#if defined(__PCH__)
      #include <18F4550.h>
      #fuses
HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL2,CPUDIV1,VREGEN
      #use delay(clock=48000000)
#endif

#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

// CCS Library dynamic defines
#DEFINE USB_HID_DEVICE  TRUE //Tells the CCS PIC USB firmware to include HID handling code.
#define USB_EP1_TX_ENABLE  USB_ENABLE_INTERRUPT  //turn on EP1 for IN bulk/interrupt transfers
#define USB_EP1_TX_SIZE  64  //allocate 64 bytes in the hardware for transmission
#define USB_EP1_RX_ENABLE  USB_ENABLE_INTERRUPT  //turn on EP1 for OUT bulk/interrupt transfers
#define USB_EP1_RX_SIZE  64  //allocate 64 bytes in the hardware for reception

// CCS USB Libraries
#include <pic18_usb.h>  //Microchip 18Fxx5x hardware layer for usb.c
#include <usb_desc_hid 8-byte.h>  //USB Configuration and Device descriptors for this UBS device
#include <usb.c>  //handles usb setup tokens and get descriptor reports

int j = 0;
int8 prijemniPaket[20];
```

```
void usb_debug_task(void) {
  static int8 last_connected;
  static int8 last_enumerated;
  int8 new_connected;
  int8 new_enumerated;

  new_connected=usb_attached();
  new_enumerated=usb_enumerated();

  last_connected=new_connected;
  last_enumerated=new_enumerated;
}

#INT_RDA
void serial_isr()                    // Serial Interrupt
{
        disable_interrupts(GLOBAL);          // Disable Global Interrupts

        if (j<19)
        {
                prijemniPaket[j] = getc();
                j++;
        }
        else
        {
                prijemniPaket[j] = getc();
                j = 0;
                output_high(PIN_B0);
        }

        enable_interrupts(GLOBAL);              // Enable Global Interrupts
}


void main()
{
        int8 in_data[8];
        int8 out_data[8];
        int8 paketZaSlanje[20];


        int i;

        set_tris_b(0x00);
        output_b(0);
        enable_interrupts(INT_RDA);
        enable_interrupts(GLOBAL);

        usb_init();
        delay_ms(1);
```

```
while (TRUE)
{
usb_task();
usb_debug_task();
if (usb_enumerated())
        {
if (usb_kbhit(1))
                {
usb_get_packet(1, in_data, 8);
                        switch (in_data[0])
                        {
                                case 14:
                                        //output_b(in_data[1]);
                                        for (i=1;i<8;i++)
                                        {
                                                paketZaSlanje[i-1] = in_data[i];
                                        }
                                        out_data[0] = 15;
                                        break;
                                case 16:
                                        for (i=1;i<8;i++)
                                        {
                                                paketZaSlanje[i+6] = in_data[i];
                                        }
                                        out_data[0] = 17;
                                        break;
                                case 18:
                                        for (i=1;i<7;i++)
                                        {
                                                paketZaSlanje[i+13] = in_data[i];
                                        }
                                        out_data[0] = 0;


                                        for (i=0;i<20;i++)
                                                putc(paketZaSlanje[i]);

                                        break;
                                case 1:
                                        switch (in_data[1])
                                        {
                                                case 1:
                                                        setup_uart(1200);
                                                        output_high(PIN_B1);
                                                        delay_ms(700);
                                                        output_low(PIN_B1);
                                                        break;
                                                case 2:
                                                        setup_uart(2400);
                                                        output_high(PIN_B1);
                                                        delay_ms(700);
```

```
                                        output_low(PIN_B1);
                                        break;
                                case 3:
                                        setup_uart(9600);
                                        output_high(PIN_B1);
                                        delay_ms(700);
                                        output_low(PIN_B1);
                                        break;
                                case 4:
                                        setup_uart(19200);
                                        output_high(PIN_B1);
                                        delay_ms(700);
                                        output_low(PIN_B1);
                                        break;
                                }

                                out_data[0] = 0;
                                break;
                        case 22:
                                output_low(PIN_B0);
                                out_data[0] = 23;
                                for (i=1;i<8;i++)
                                {
                                        out_data[i] = prijemniPaket[i-1];
                                }
                                break;
                        case 24:
                                out_data[0] = 25;
                                for (i=1;i<8;i++)
                                {
                                        out_data[i] = prijemniPaket[i+6];
                                }
                                break;
                        case 26:
                                out_data[0] = 27;
                                for (i=1;i<7;i++)
                                {
                                        out_data[i] = prijemniPaket[i+13];
                                }
                                break;

                        }
                        usb_put_packet(1, out_data, 8, USB_DTS_TOGGLE);

        delay_ms(1);
        }
        }
        }
}
```

Glavni deo programa vrši inicijalizaciju registara mikrokontolera usb komunikacionog modula i prijemnih i predajnih bfera. Dozvoljavaju se prekidi koje generiše serijski modul kao i globalni prekid. Procedura usb_init obavlja pripremu i slanje deskriptora hostu kako bi mu ovaj dodelio usb adresu, ona takođe dozvoljava usb prekide jer se kompletna usb komunikacija kod HID standarda obavlja u prekidnim rutinama. Nakon ovih inicijalizacija sledi bekonačna petlja u kojoj se najpre opslužuje usb komunikacija kako bi podaci bili primljeni u bafer ili poslati. Takđe se proverava i da li je uređaj i dalje priključen. Ako je uređaj enumerisan vrši se obrada primljenih informacija od računara. Proverava se konfiguraciona reč i podešava brzina. Kada se bafer za slanje napuni podaci se šalju pomocu funkcije *putc*. Na kraju petlje je pauza od 1ms zbog *pooling* intervala usb porta. Prijem se obavlja u interaptu koji se generiše svaki put kada se prijemni bafer USART modula napuni. U ovoj rutini treba zabraniti prekide kako ona nebi bila prekinuta od strane usb prekida i time došlo do gubljenja dela informacija. U prekidu se koristi brojač podataka kako bi znali koliko bajtova je primljeno. U programu se takođe vrši paljenje i gašenje led dioda na portu B kada se podaci prime i kada se podese parametri komunikacije.

3) Fajl disasemblerskog listinga UsbToRS232Term.lst. Ovaj fajl sadrži asemblerski kod celog programa ali i ugrađenih komunikacionih funkcija za usb i rs-232. Negov sadržaj je dosta veliki pa ga nećemo prikazati već će biti priložen uz izvorni kod.

7.Primeri laboratorijskih vežbi

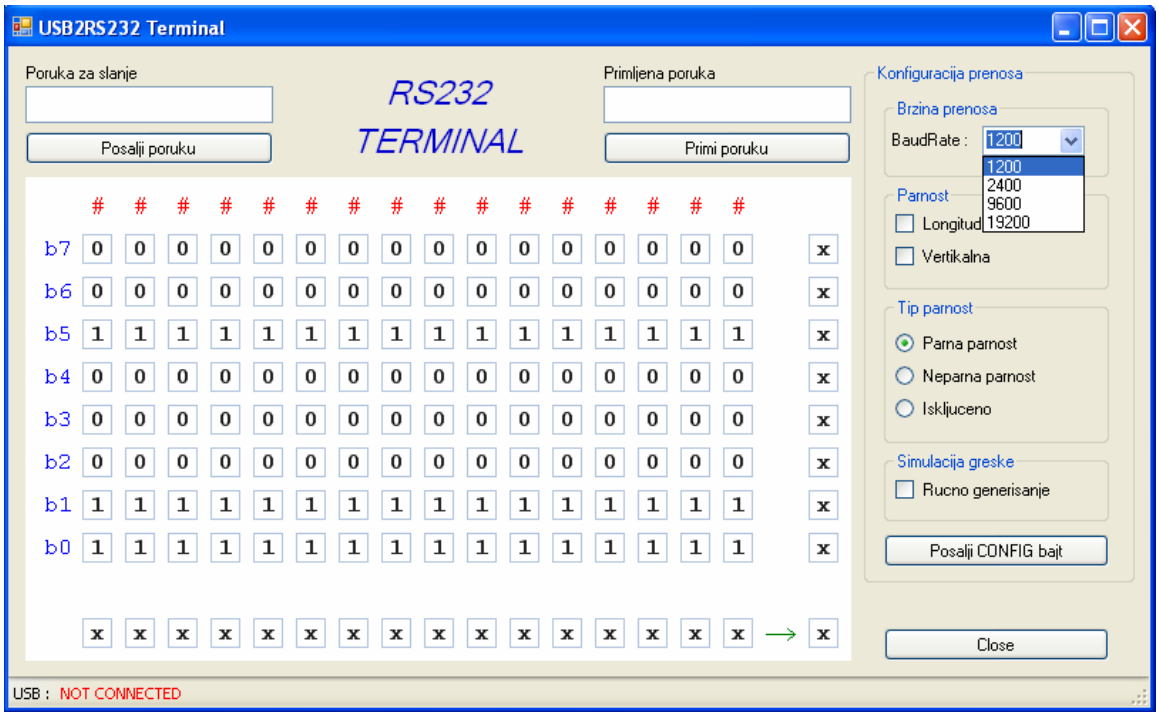
Prvenstvena namena ovog projekta je edukacija i simulacija asinhronog serijskog prenosa kao i tehnika parnosti za detekciju grešaka. Zbog toga smo pripremili četiri laboratorijskih vezbi koje obuhvataju sve što je ovim projektom realizovano i sve što se može simulirati.

7.1 Instalacija i povezivanje

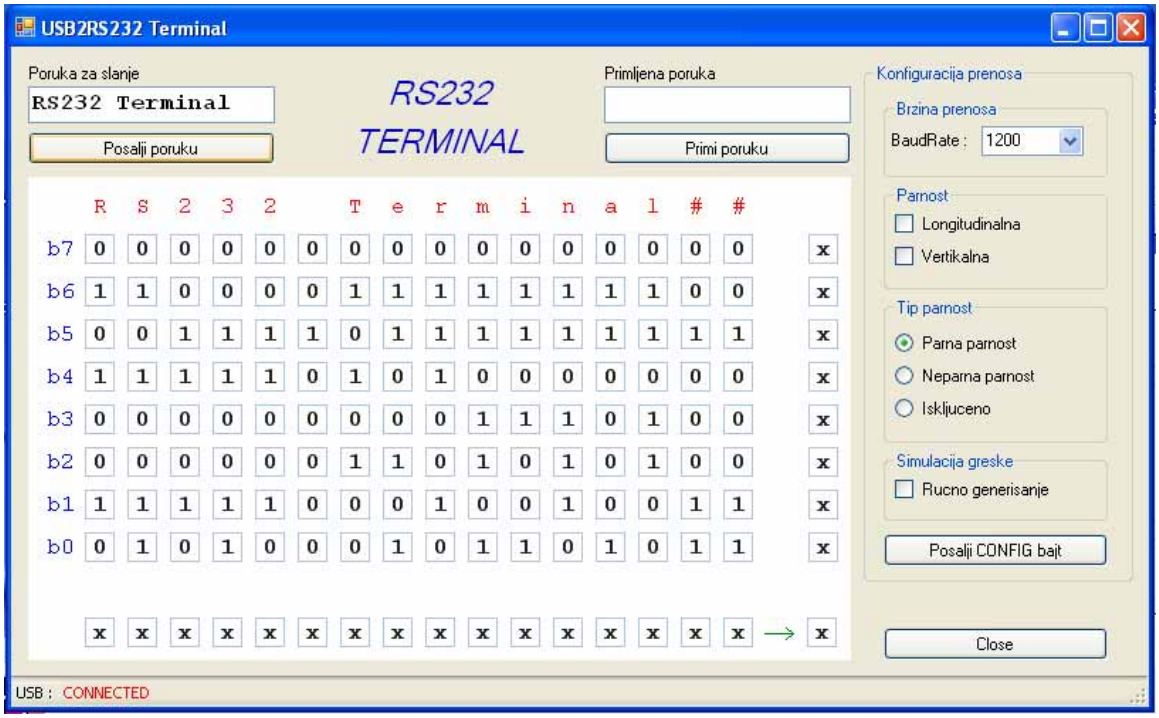
Za rad sa rs-232 terminalom potrebna su dva računara, dva komunikaciona modula, jedan rs-232 kabl i dva usb kabla. Najpre treba na oba računara instalirati aplikaciju koja na računar instalira i .NET paket ukoliko ga nema. Potom treba kablom za usb povezati po jedan računar i modul, a sa rs-232 kablom povezati oba komunikaciona modula. Ukoliko je sve uspešno povezano zelene led diode na oba modula treba da budu upaljene, a u statusnoj liniji prozora teba da piše “connected”. Tada je sve spremno za rad.

7.2 Vežba1: Podešavanje komunikacije i slanje podataka bez detekcije greškaka

Nakon povezivanja i startovanja aplikacije treba podesiti parametre. I prijemna i predajna strana trebaju podesiti istu bodovu brzinu i nakon toga poslati konfiguracioni bajt kako bi mikrokontroler bio konfigurisan na zeljenu bodovu brzinu.



Potom predajna strana upisuje poruku u polje za unos koja ne sme biti veća od 16 karaktera. Nakon toga podatke treba poslati.



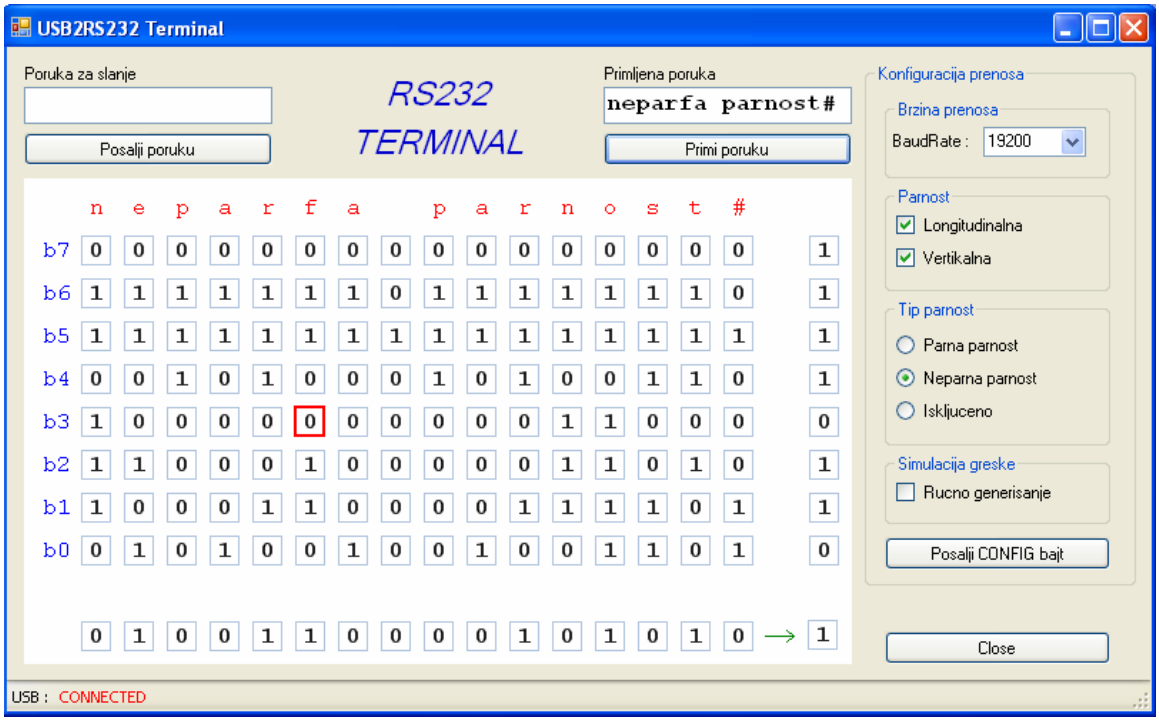
Kada prijemna strana primi podatke, na modulu se pali crvena led dioda koja signalizira da podaci mogu biti pročitani.
Isti ovaj postupak može se ponavljati više puta za druge bodove brzine. Svaki put kada se menja brzina potrebno je poslati konfiguracioni bajt.

7.3 Vežba2: Prenos podataka sa informacjama o parnosti

Ova vežba je slična kao prethodna samo treba uključiti neku od parnosti. Moguće su 6 kombinacije za parnu, neparnu, vertikalnu i horizontalnu parnost. Ukoliko nije došlo do grešaka u prenosu bitovi parnosti na prijemnoj strani će biti isti kao i na predajnoj. Ove opcije takođe treba proveriti na svim brzinama.

7.4 Vežba3: Prenos sa simulacijom jednostruke greške

Ova vežba pokazuje kako se vertikalna i horizontalna parnost mogu uspešno koristiti za detekciju i ipravljanje grešaka kod asinhrone komunikacije.
Da bi se simulirale greške potrebno je posle unosa proizvoljne poruke za slanje stiklirati stavku “ručno generisanje”. Nakon toga klikom miša treba promeniti odnosno invertovati neki bit i poslati poruku sa ovako ubačenom greškom.
Ako se na primer pošalje poruka “neparna parnost” na prijemu će se dobiti poruka “neparfa parnost”. Bit koji je pogrešno prenet je uokviren i može se ispraviti tako što se jednostavno invertuje jer se radi o jednostrukoj grešci.

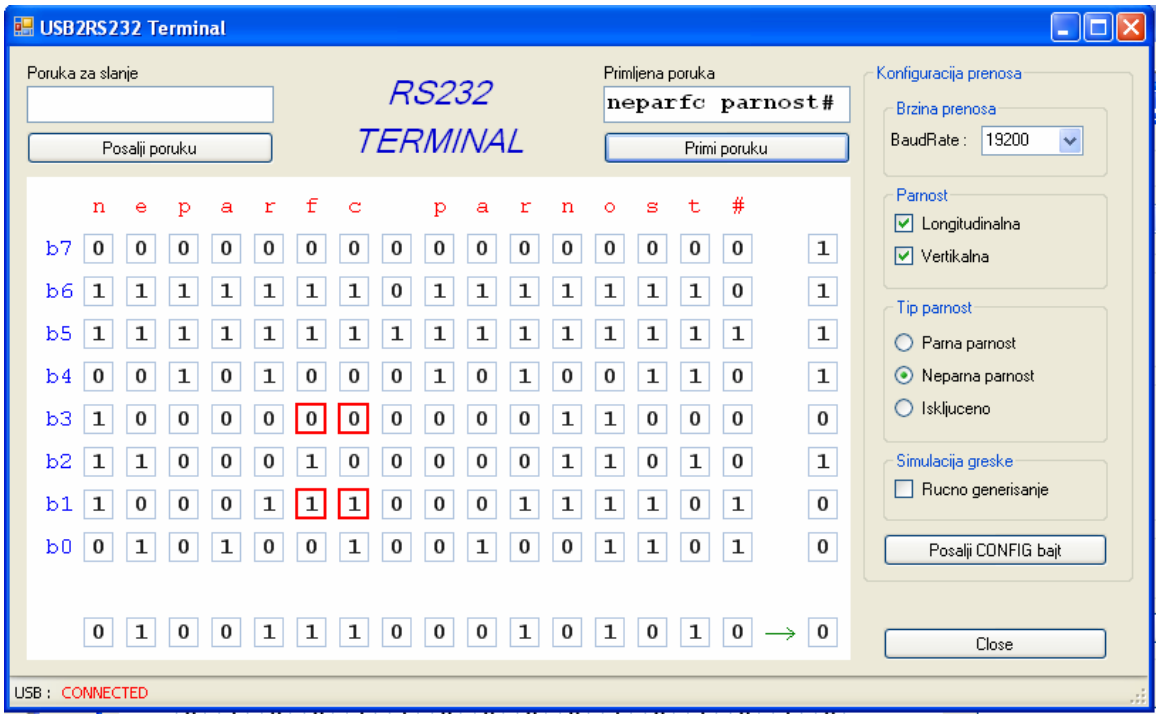


Vežba se moze ponoviti za različite dužine poruke i uz promenu tipa parnosti.
U svim slučajevima greška će biti detektovana i moguće je ispraviti.

7.5 Vežba4: Prenos sa simulacijom visestruke greške

U ovoj vežbi se jasno vidi nedostatak parnosti kod otkrivanja grešaka. U nekim slucčajevima greška neće biti otkrivena dok će u nekim biti otkrivene i lažne greške.

Isto kao i u prethodnoj vežbi upiše se proizvoljna poruka za slanje i izaberu vertikalna i horizontalna parnost bilo parna ili neparna. Sada treba promeniti dva bita u istoj koloni ili vrsti (na b3 n i a ili na n b3 i b1) i poslati poruku. Poruka će biti primljena pogrešno ali je zbog parnog broja grešaka nemoguće locirati grešku. U tom slučaju će zbog korišćenja vertikalne i horizontalne parnosti istovremeno greška biti otkrivena ali ne i locirana.
Ponoviti postupak ali sa nesto drugačijom simulacijom greške. Sada treba promeniti bitove u različitim vrstama i kolonama na primer b3 na n i b1 na a. Nakon toga poslati poruku. Na prijemnoj strani će se detektovati lažne greške na pozicijama ukrštanja vertikalne i horizontalne parnosti.



Postupak treba ponoviti za razne kombinacije pogrešno prenetih bitova i utvrditi efikasnost metode za detekciju grešaka.

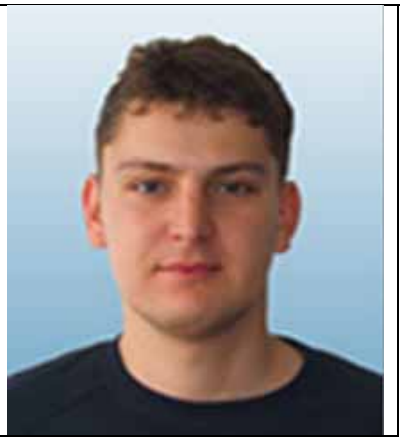
8. Zakljucak

Koristeći realizovan hardver i softver uvideli smo mnoge nedostatke u detekciji grešaka metodom provere bita parnosti i neke od njih detaljno opisali. Studenti koji će izvoditi laboratorijske vežbe pomoću ovog projekta će moći to i praktično proveriti. Kroz projekat smo se detaljnije upoznali sa asinhronom serijskom komunikacijom i usb standardom koji je u ovom projektu korišćen samo u onoj meri u kojoj je bio potreban. Nadamo se da će ovaj vid komunikacije u potpunosti zameniti rs-232 komunikaciju bar kada je u pitanju veza ovakvih mikroračunarskih sistema sa PC računarima.

9. Literatura

PIC mikrokontroleri-Nebojša Matić
Programiranje mikrokontrolera PicBASIC-om-Vojo Milanović
www.microchip.com
www.usb.org
www.mikroelektronika.co.yu

10. Autori projekta

Miroslav Božić	
Broj indeksa:11006 SSS:Elektrotehničar automatike, Tehnička škola Zaječar Datum rođenja : 19.01.1984. Adresa : Svetozara Markovića 135, Zaječar Godina upisa na fakultet : 2003. Smer : Računari Mail : bozic_miroslav @yahoo.com	

Atanasovski Aleksandar	
Broj indeksa:11099 SSS:Elektrotehničar elektronike, ETŠ ‘’Nikola Tesla’’ , Niš Datum rođenja: 17.10.1984. Adresa : Cara Dušana 97 1/3, Niš Godina upisa na fakultet : 2003. Smer : Računari Mail : alekata1 @yahoo.com	