UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
PREDMET: MIKROPROCESORSKI SISTEMI

USART (UniversalSynchronous/Asynchronous Receiver/Transmitter) INTEL 8251

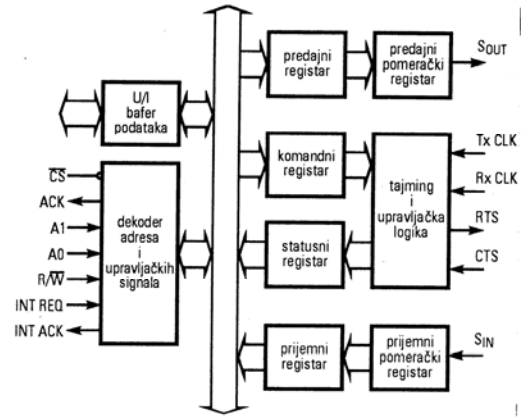
Studenti:
Dejan Micić 8845
Marija Dragićević 11072

S A D R Ž A J:

1. UVOD

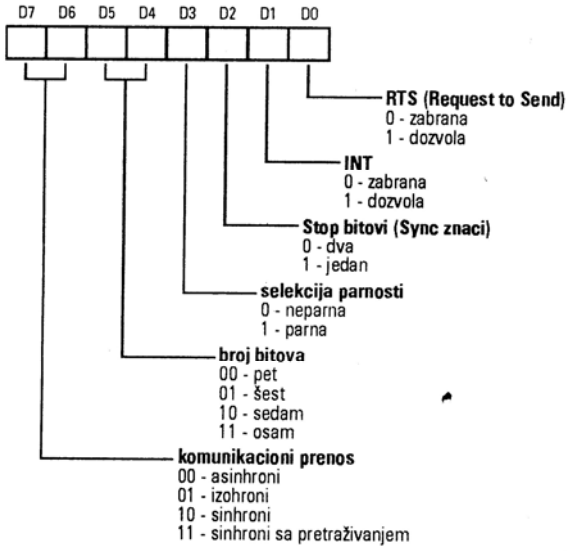
1.1. Serijski prenos

Serijska informacija se prenosi preko jedinstvene linije na principu bit - po - bit. Na ovaj način smanjuje se broj veza (žica) kojim se CPU povezuje sa U/I uređajem. Direktna posledica ovakvog rešenja je manja brzina prenosa informacija. Dodatno, problem se javlja zbog potreba za konverzijom formata podataka. Konverzija rezultira složenom tehničkom rešenju. Složenost se ogleda u ugrađivanju mogućnosti za sinhronizaciju I detekciju grešaka u prenosu. No nezavisno od nabrojanih nedostataka I problema veliki je broj U/I uređaja koji koriste serijski prenos. Tipični su tastature, miševi, štampaci i dr. kod kojih je, reklo bi se, brzina serijskog prenosa informacije adekvatna. Na slici 1.1 prikazano je kako se ovi U/I uređaji tipa USART za sinhrono/asihroni prenos povezuju sa CPU – om.



Slika 1.1.1 struktura hipotetičkog predajnika/prijemnika

Na ulazno/izlaznim krajevima postoje serijski interfejsi, tipa PDI. Između ostalog, jedna od glavnih funkcija serijskog interfejsa odnosi se na konverziju podataka iz paralelenog u serijski i obratno, iz serijskog u paralelni. Obično za konfiguracije sa slike 1.2 kaže se da je U/I uređaj povezan sa CPU – om preko serijskog U/I porta.

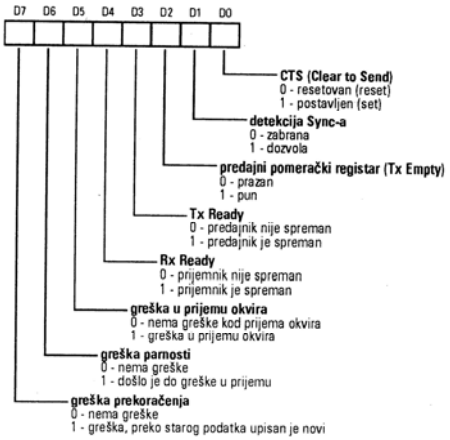


Slika 1.1.2. Sadržaj upravljačkog registra hipotetičkog USART-a

Serijski prenos predstavlja jedinstven izbor kada sistem mora da komunicira sa udaljenim terminalom ili sa drugim udaljenim sistemom. U ovom slučaju, komunikacija se ostvaruje preko mreže za prenos podataka koja može biti lokalna (do nekoliko kilometara u okviru fabrike) ili javna (koriste se standardne telefonske linije). Telefonska mreža je projektovana za prenos analognog signala govora pa se zbog toga serijski port, ciji je izlaz digitalni, povezuje na liniju preko modema (modem – **mod**ulator/**dem**odulator).

SERIJSKI U/I INTERFEJS TIPA PDI

Struktura jednog hipotetičkog serijskog U/I interfejsa tipa PDI prikazana je na slici 1.3.



Slika 1.1.3. Sadržaj statusnog registra hipotetičkog USART-a

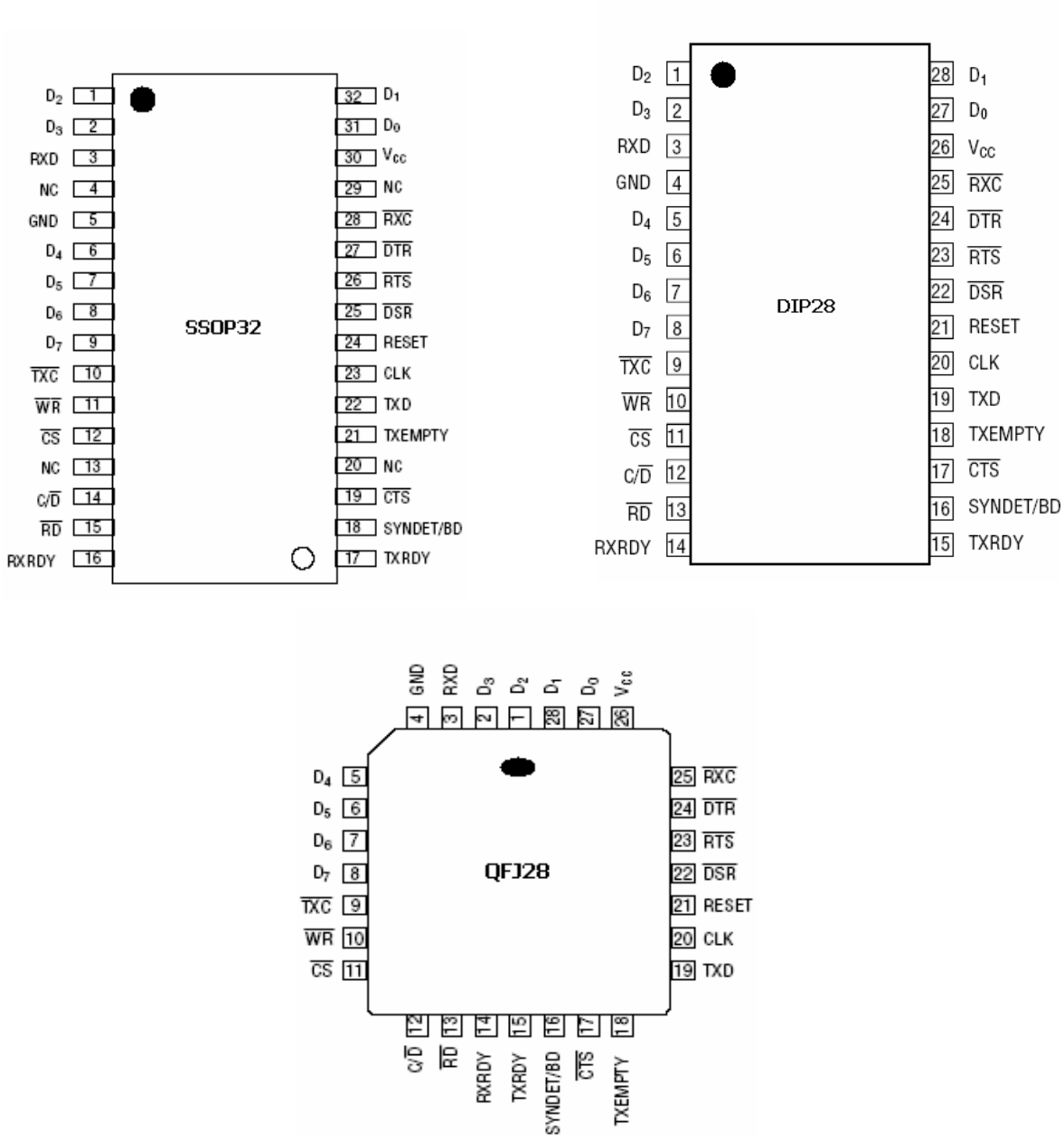
Među projektantima ovo kolo je poznato pod nazivom USART (Universal Asynchronous Receiver Transmitter). USART poseduje standardnu sekciju za spregu sa CPU – om koju čine blokovi U/I bafer podataka I dekoder adresa i upravljackih signala. Sekciju izlaznog serijskog puta podataka čine predajni bafer podataka i pomerački registar sa paralelnim punjenjem. Slično, ulazni serijski put čine pomerački registar i prijemni bafer podataka. Upravljački i statusni registar se koriste za definisanje i upravljanje konfiguracijom i monitoringom serijskog U/I interfejsa. Blok *tajming* i upravljačka logika koordiniše sekvencu U/I operacije kod interakcije sa spoljnim uređajem.

Detalji koji se odnose na definisanje nacina rada serijskog primo/predajnika dati su na slici 1.2, a detalji sadrzaja statusnog registra na slici 1.3. Uočimo da se *handshake* signalima RTS/CTS obezbeđuje mehanizam za korektan prenos podataka između USART – a i spoljnog uređaja. Nakon sto se znak iz predajnog registra prenese u predajni pomerački registar postavi se Tx *Ready* bit statusnog registra i aktivira signal INT REQ prema mikroprocesoru. Kada je i zadnji znak napustio predajni pomerački registar bit statusnog registra Tx Empty se postavi na jedan.

1.2. INTEL 8251

Intel-ov čip **8251** spada u **USART** (**Universal Synchronous/Asynchronous Receiver/Transmitter**) periferijske komponente. Ovaj čip može da (asinhrono i/ili sinhrono) prihvata i predaje podatke, i to istovremeno. Njegova namena je da preuzima podatke u paralelnom formatu od strane procesora (CPU) i prosledi ih serijski, posle konverzije. Isto tako, ovaj čip prihvata podatke serijski i prosleđuje ih ka procesoru, posle konverzije u paralelni format. Procesor u svakom trenutku može da pročita kompletan status čipa **8251**.

Napominjemo da je čip **8251 TTL** kompatibilan i proizvodi se u kućištu **DIP28**, dok je čip **82c51 CMOS** kompatibilan i izrađuje se u kućištima **DIP28**, **SSOP32** i **QFJ28**.



Sl 1.2.1. Raspored pinova **Intel**-ovog čipa **82(c)51**

U sledećoj tabeli prikazane su neke karakteristike čipova **8251** i **82c51**.

	8251	**82c51**
FAMILIJA	TTL	CMOS
ASINHRONA BRZINA	19,2 KBaud	64 KBaud
SINHRONA BRZINA	38,4 KBaud	64 KBaud
NAPAJANjE (jednostruko)	5V	3-6V
OPŠTE KARAKTERISTIKE	Potpuna kompatibilnost sa **INTEL**-ovim mikroprocesorom **8085**; kompatibilnost sa familijama **MCS-48**, **80**, **85** i **iAPX86**, **88**; jedan takt; potpuni dupleks i dvostruko baferovanje pri slanju/prijemu podataka; detekcija grešaka (na osnovu parnosti, prekoračenja i formata podataka); za *sinhroni prenos*: interna ili eksterna sinhronizacija; za *asinhroni prenos*: 1 ili 2 stop-bita, detekcija lažnog start-bita, automatski prekid detekcije i njena obrada; (eventualno) deljenje frekvencije takta (16 ili 64 puta)	

(**Napomena**: U daljem tekstu koristiće se isključivo naziv čipa **8251**.)

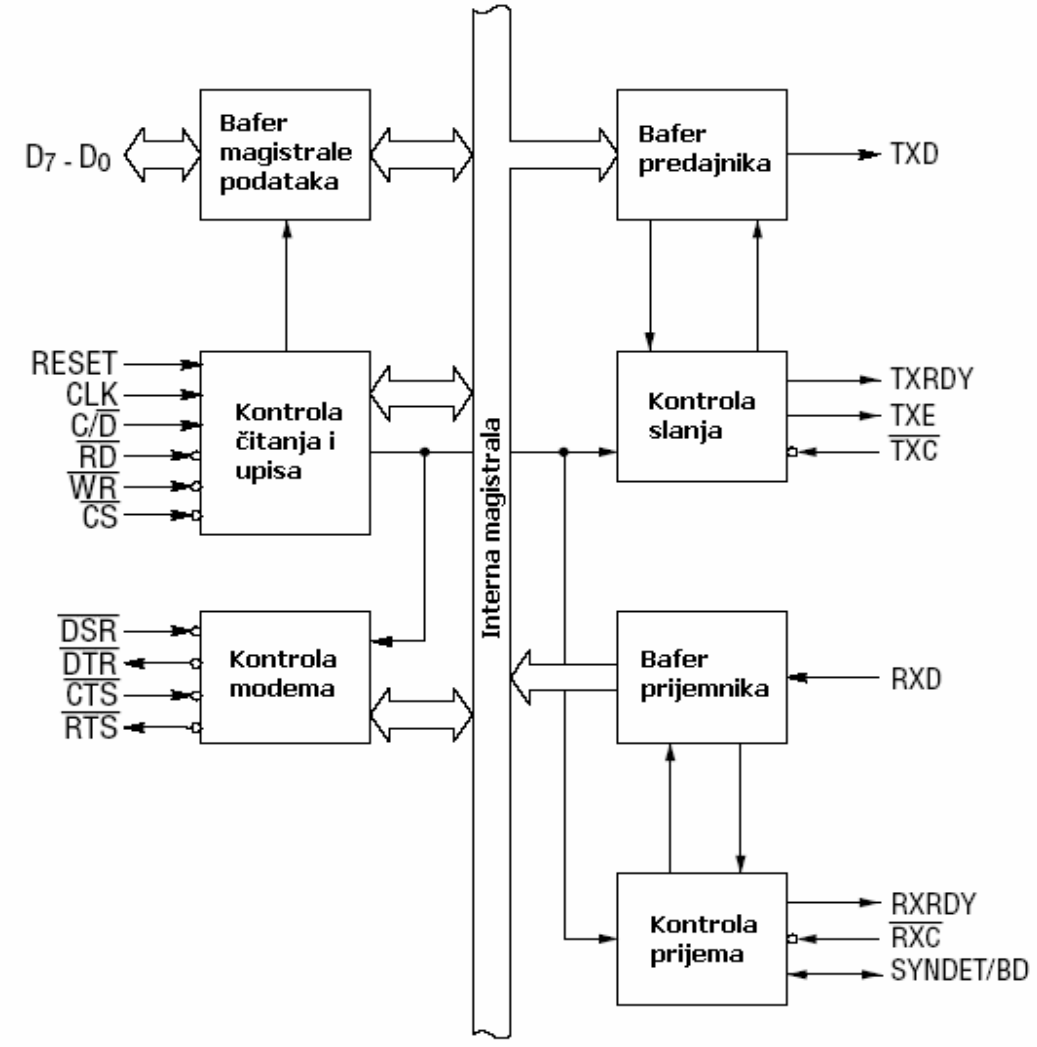
2. INTEL 8251

2.1.1.Opšti opis

Čip **8251** je u potpunosti prilagođen intelovom mikroprocesorskom sistemu 8085. Programiranje ovog čipa ostvaruje se sistemskim softverom, što važi i za druge I/O komponente pomenutog sistema. Čip **8251** može da podrži praktično bilo koji serijski način prijema/slanja podataka, uključujući i IBM-ovu *BI-SYNC* tehniku.

U komunikacionim sistemima, interfejs između procesora (CPU) i I/O periferijske komponente/uređaja treba da vrši konverziju formata podataka. Kada se podaci prenose od procesora ka periferiji, interfejs prihvata podatke od procesora u paralelnom formatu i konvertuje ih u serijski niz. Periferija prihvata podatke od pomenutog interfejsa serijski, tj. "bit po bit". U slučaju kada periferija šalje podatke ka procesoru, dešava se obrnut proces- interfejs prihvata podatke serijski, a šalje ih prema procesoru u paralelnom formatu.

Pored navedenog, pomenuti interfejs mora da obavi dodatne operacije - brisanje prethodno poslatog bita/podatka i pridruživanje dodatnih bitova karakteru koji se šalje. (Dodatni bitovi su: start-bit, stop-bitovi i bit parnosti.)



Sl.2.1. Funkcionalni blok-dijagram čipa **8251**

Bafer magistrale podataka

Ovaj 8-bitni i bidirekcioni bafer ima 3 stanja, i predstavlja vezu (interfejs) između čipa **8251** i magistrale podataka procesora **8085**. Podatak se šalje/prima preko bafera prilikom izvršenja ulazne/izlazne instrukcije od strane procesora. Preko *bafera magistrale podataka* prenose se i kontrolne i komandne reči, kao i informacija o statusu.

Kontrola čitanja/upisa

Kontrola čitanja/upisa ostvaruje se preko funkcionalnog bloka koji prihvata podatke sa *kontrolne magistrale* i na osnovu njih generiše kontrolne signale (za svaku operaciju). Blok sadrži **CWR** (***Control Word Register***) u koji se upisuju različiti kontrolni formati, zavisno od trenutne definicije funkcije čipa **8251**.

Reset

Dovođenjem "visokog nivoa" na pin Reset, čip **8251** prelazi u tzv. "***IDLE*" režim.** Čip ostaje u ovom režimu sve dok se ne upiše novi skup kontrolnih reči. Minimalno trajanje *Reset* signala treba da iznosi 6 taktnih impulsa.

Takt

Na pin CLK dovodi se signal sa generatora takta (vidi Sl. 2.2.) i ovaj signal predstavlja internu vremensku bazu u čipu **8251**. Frekvencija takta mora da bude bar 30 puta veća od frekvencije taktova RxC i TxC za sinhroni režim, odnosno 4,5 puta ukoliko je u pitanju asinhroni režim. (Napomena: $\overline{RxC}$ i $\overline{TxC}$ su taktovi prijemnika i predajnika čipa **8251**, respektivno).

Upis (Write)

Dovođenje "niskog nivoa" na pin $\overline{WR}$, podaci ili kontrolne reči prenose se sa procesora (CPU) ka **8251**.

Čitanje (Read)

Dovođenje "niskog nivoa" na pin $\overline{RD}$, podaci ili kontrolne reči prenose se sa **8251** ka procesoru (CPU).

Kontrola/podaci (C/D)

Zavisno od toga koji se signal dovodi na ovaj pin, kao i stanja na pinovima $\overline{WR}$ i $\overline{RD}$, određuje se da li se prenosi podatak, kontrolna reč ili informacija o statusu.

Selektovanje čipa (CS)

Dozvola rada čipa **8251** ostvaruje se dovođenjem "niskog nivoa" na pin $\overline{CS}$,. Sve dok to nije slučaj, suspendovani su i čitanje i upis.

U sledećoj tabeli prikazani su transferi podataka u zavisnosti od stanja na pinovima $C/\overline{D}$, $\overline{RD}$, $\overline{WR}$ i $\overline{CS}$.

$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	1	0	**8251** => **Data Bus**
0	1	0	0	**Data Bus** => **8251**
1	0	1	0	**Status** => **Data Bus**
1	1	0	0	**Data Bus** => **Control**
X	1	1	0	**Data Bus** => **3-State**
X	X	X	1	**Data Bus** => **3-State**

Kontrola modema (Modem Control)

Čip **8251** poseduje skup kontrolnih ulaza i izlaza (DSR, DTR, RTS i CTS), preko kojih može da se ostvari komunikacija sa gotovo svakim modemom. Pomenuti kontrolni signali mogu da se koriste i za druge namene (ukoliko je to neophodno).

Spremnost podataka (Data Status Ready)

Stanje ulaznog signala DSR (ovaj signal generiše modem) ispituje procesor (CPU) preko *Status Read* operacije. Ukoliko je stanje "nisko", to znači da su podaci spremni za slanje od strane modema.

Spremnost za prijem (Data Terminal Ready)

Stanje izlaznog signala DTR se postavlja "nisko" preko odgovarajućeg bita u *komandnoj instrukciji*. U tom slučaju, modemu se šalje informacija da je čip **8251** spreman da prihvati podatke.

Zahtev za slanje (Request to Send)

Stanje izlaznog signala RTS se postavlja preko odgovarajućeg bita u *komandnoj instrukciji*. Ukoliko je "nizak nivo", modemu se upućuje zahtev za slanje podataka ka čipu **8251**.

Brisanje za novo slanje (Clear to Send)

Kada je ulazni signal na "niskom nivou" i kada je bit TxEN postavljen na '1", čipu **8251** je dozvoljeno (serijsko) slanje podataka ka (modemu).

Bafer predajnika (Transmitter Buffer)

Namena ovog bafera je da prihvati podatke sa *bafera magistrale podatka* (u paralelnoj formi), zatim da ih konvertuje u serijski niz i prosledi na izlazni pin TxD.

Naravno, prilikom konverzije, transmisioni bafer dodaje serijskom nizu odgovarajuće bitove, zavisno od tehnike prenosa podataka.

Kontrola predajnika (Transmitter Control)

Zadatak ove logike (na koju dolaze i eksterni i interni signali) je da upravlja svim aktivnostima vezanima za serijsko slanje podataka.

Spremnost predajnika (Transmitter Ready)

Izlazni signal TxRDY upućuje se ka procesoru (CPU) u cilju indikacije da je čip **8251** spreman da prihvati karakter. TxRDY može da se koristi i kao prekid (inerrupt) u sistemu ili za operaciju "prozivanja". CPU proverava stanje TxRDY preko *Status Read* operacije. TxRDY se automatski resetuje kada se karakter (podatak) prihvati sa procesora.

Detekcija prestanka slanja podataka (Transmitter Empty)

Kada u čipu nisu upisani karakteri za slanje, stanje izlaznog pina TxE postavlja se na "visok nivo". Važno je napomenuti da je signal TxE nezavisan od bita TxEN smeštenog u okviru *komandne instrukcije*.

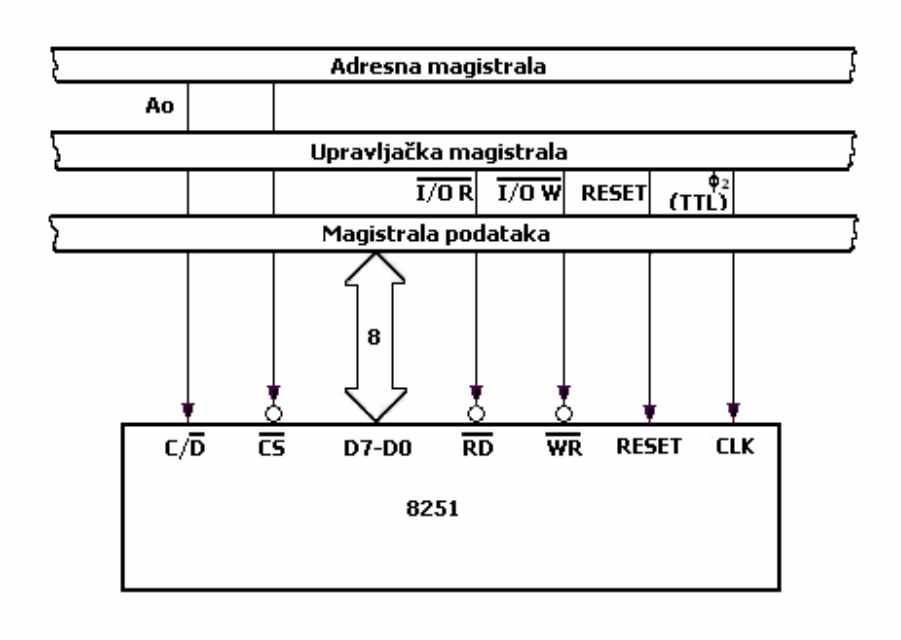
U polu-dupleks modu (*half duplexed mode*) signal TxE zapravo služi kao indikacija procesoru (CPU) da se završio *režim slanja* (*transmision mode*).

U "sinhronom" režimu rada "visok nivo" signala TxE prouzrokuje da karakter nije učitan. Tada se automatski šalju SYNC karakter(i), a kada se završi proces njihovog slanja, TxE se postavlja na "nizak nivo".

Takt predajnika (Transmitter Clock)

Takt predajnika (signal TxC) kontroliše brzinu kojom će podaci biti poslati. U sinhronom modu, brzina prenosa jednaka je aktuelnoj brzini u Baud-ima (*Baud Rate*). Tokom asinhronog moda, brzina prenosa jednaka je umnošku (1x, 16x ili 64x) aktuelne brzine. Na primer, ukoliko je **BaudRate** = 110, tada TxC uzima vrednosti: 110Hz (1x), 1,76KHz (16x) ili 7,06KHz (64x).

Čip **8251** serijski šalje podatke na svaku opadajuću ivicu signala TxC.



Sl.2.2. Povezivanje čipa **8251** na standardni sistem magistrala procesora **8085**

Bafer prijemnika (Receiver Buffer)

Bafer prijemnika preko pina TxD prihvata serijske podatke, konvertuje ih u paralelni format i prosleđuje ih ka procesoru (CPU). Tokom konverzije, bafer proverava podatke, u skladu sa izabranom tehnikom prenosa.

Kontrola prijemnika (Receiver Control)

Ova logika upravlja svim aktivnostima vezanim za prijem podataka.

Spremnost prijemnika (Receiver Ready)

Izlazni signal RxRDY upućuje se ka procesoru (CPU) u cilju indikacije da je čip **8251** spreman da prihvati karakter. RxRDY može da se koristi i kao prekid (*inerrupt*) u sistemu ili za operaciju "prozivanja". CPU proverava stanje RxRDY preko *Status Read* operacije. RxRDY se automatski resetuje kada procesor prihvati karakter (podatak).

Takt prijemnika (Receiver Clock)

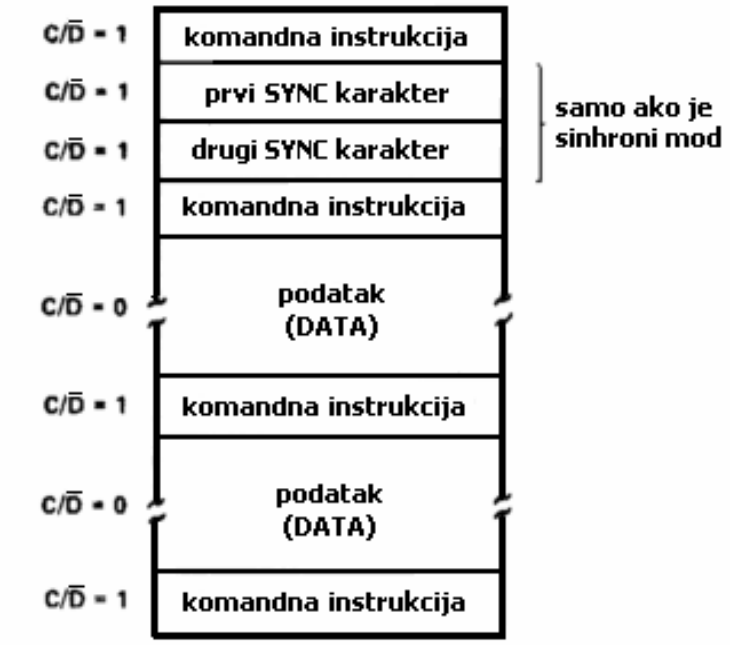
Takt predajnika (signal RxC) kontroliše brzinu kojom će podaci biti prihvaćeni. U sinhronom modu, brzina prijema jednaka je aktuelnoj brzini u Baud-ima (*Baud Rate*). Tokom asinhronog moda, brzina prijema je jednaka umnošku (1x, 16x ili 64x) aktuelne brzine. Na primer, ukoliko je **BaudRate** = 2400, tada RxC uzima vrednosti: 2,4KHz (1x), 38,4KHz (16x) ili 153,6KHz (64x).

Čip **8251** serijski šalje podatke na svaku rastuću ivicu signala RxC.

(**Napomena:** U većini komunikacionih sistema čip **8251** obavlja i slanje (predaju) i prijem podataka preko jedne linije veze. Posledica toga je da su brzine predaje i prijema jednake. Samim tim, signali TxC i RxC mogu da budu priljučeni na isti *generator takta*.)

2.1.2. Detaljan opis rada

Da bi čip **8251** podržao željenu komunikacionu tehniku, neophodno je da procesor (CPU) izvrši njegovu inicijalizaciju slanjem odgovarajućih *kontrolnih reči*.
Ovim kontrolnim rečima definišu se: brzina prenosa (Baud Rate), dužina karaktera, broj stop-bitova, sinhrone/asinhrone operacije, parna/neparna parnost itd. U sinhronom modu, omogućeno je da se izabere interna ili eksterna sinhronizacija.



Sl.2.3. Izgled bloka podatka

Kada se programira, čip **8251** je spreman da komunicira sa procesorom i periferijskim uređajem. Izlazni signal TxRDY se postavlja na "visok nivo" čime se signalizira procesoru (CPU) da je **8251** spreman da prihvati karaktere. Kad god CPU upiše karakter u **8251**, signal TxRDY se automatski resetuje. Sa druge strane, **8251** prihvata podatke od nekog I/O uređaja (npr. modema).

Nakon svake operacije čitanja od strane procesora (CPU), vrši se automatsko resetovanje signala RxRDY.

Naravno, čip **8251** ne može da započne slanje podataka sve dok ne prihvate signal CTS i bit TxEN (bit u okviru *komandne instrukcije*). Sve do momenta pojave signala *Reset,* izlazni pin TxD je "markiran", tj. nije moguće slanje podataka (preko ovog pina).

Programiranje čipa 8251

Pre svakog slanja/prijema, u čip **8251** moraju da se upišu *kontrolne reči* koje generiše CPU. Ovi kontrolni signali definišu kompletnu funkciju čipa **8251**, koji mora da prati interne/eksterne Reset operacije. *Kontrolne reči* dele se u dve grupe:

1. instrukcije režima rada ili instrukcije moda (*Mode instruction*),
2. komandne instrukcije (*Command instruction*).

Instrukcije režima rada

Ovim formatom definišu se generalne operacije čipa **8251**. On mora da "prati" (interne ili eksterne) Reset operacije. Tek kada u **8251** procesor upiše *instrukcije režima rada*, tzv. SYNC karakteri ili komandne instrukcije smeju biti ubačeni (u blok podatka).

Komandne instrukcije

Ovim formatom definiše se *statusna reč* koja se koristi za kontrolu trenutne operacije čipa **8251**.

Pre nego što se čip **8251** primenu u komunikacionom sistemu, u njega se upisuju instrukcije koje definišu režim rada (i to neposredno posle Reset operacije).

Definicija instrukcija režima rada (moda)

Čip **8251** može da se koristi kako u asinhronom, tako i u sinhronom režimu rada. O oba režima rada biće reči u ovom poglavlju.

Asinhroni režim rada - slanje podataka

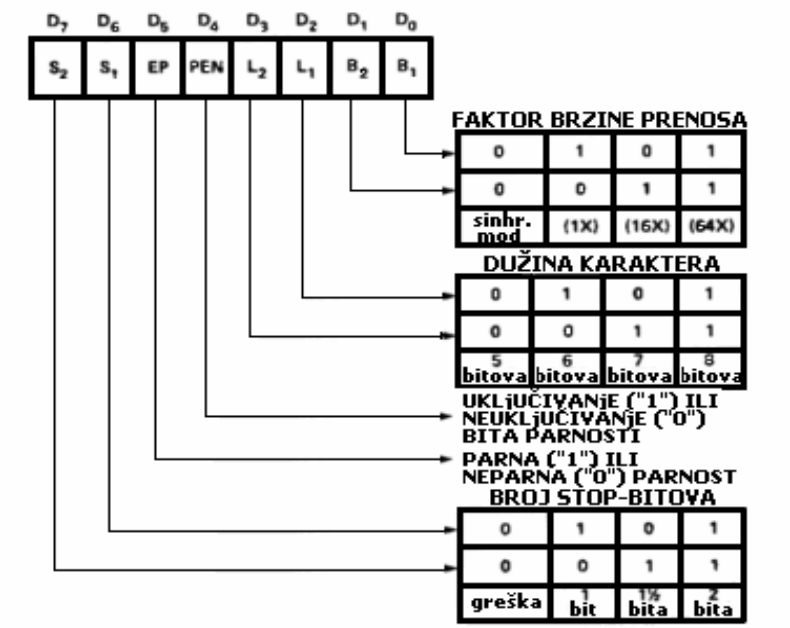
Prilikom svakog slanja karaktera od strane procesora ka čipu **8251**, ovaj formira *ram* podatka ili okvir (*frame*) koji sadrži start-bit, karakter koji se prenosi (poslao ga je CPU) , bit parnosti i unapred zadati broj stop-bitova. Zatim se ovako formirani podatak prenosi serijski preko izlaza TxD.

Važno je istaći da se podatak "šiftuje"(položaj preostalih bitova pomera se za jedno mesto, nakon što se jedan bit podatka pošalje) na opadajuću ivicu signala TxC.

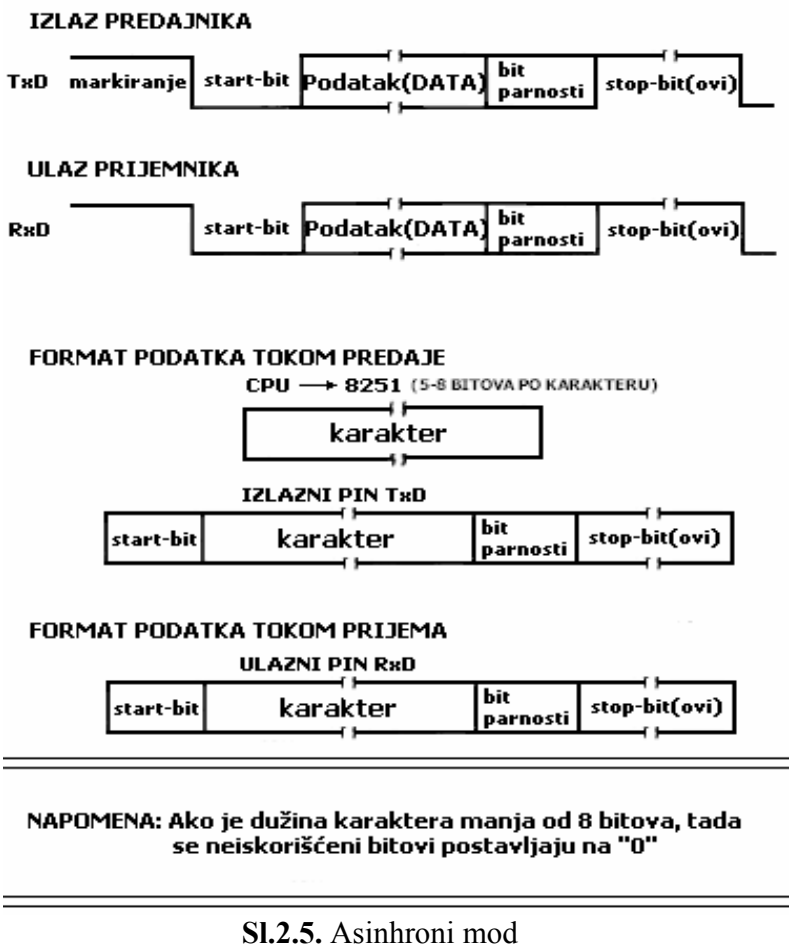
Ukoliko se to zahteva komandnom instrukcijom, na izlaz TxD mogu neprekidno da se šalju *Break* karakteri.

Kada se u čipu **8251** ne nalazi više karakteri (podaci) za slanje, izlaz TxD se "markira" (postavlja na "visoko stanje"), osim u slučaju kada je programiran prekid rada, odnosno **break** operacija.

(**Napomena:** *Instrukcija režima rada* definiše broj stop-bitova, kao i brzinu slanja podatka – pogledati **poglavlje 2.2!)**



Sl.2.4. Format instrukcije moda za asinhroni režim



Sl.2.5. Asinhroni mod

Asinhroni režim rada - prijem podataka

Validnost start bita se proverava na sredini bitskog intervala. Ukoliko se ponovo detektuje "nizak nivo", start-bit je validan i brojać bitova može da otpočne brojanje.

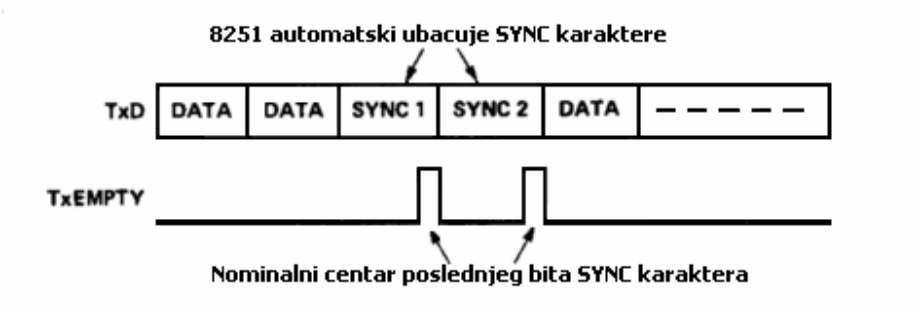
Ukoliko je prisutna greška parnosti, postavlja se *marker* (*fleg*) greške parnosti. Podatak i bitovi parnosti su semplovani na RxD pinu sa opadajućom ivicom RxC. Ako se "nizak nivo" detektuje kao stop-bit, postavlja se *marker* greške *rama* (frejma). Stop-bit signalizuje kraj prijema karaktera. Tada je ovaj karakter učitan u paralelni I/O bafer čipa **8251**. Preko pina RxRDY se signalizira procesoru da je karakter spreman za pribavljanje. Ukoliko CPU nije pribavio prethodni karakter iz paralelnog I/O bafera čipa **8251**, onda je taj karakter nepovratno izgubljen, jer je preko njega upisan novi karakter. U tom slučaju postavlja se marker prekoračenja. Svaki od pomenutih markera može da se resetuje od strane komandne instrukcije. Prisutnost pomenutih grešaka ne zaustavlja rad čipa **8251**.

Sinhroni režim rada (sinhroni mod) – slanje podataka

Izlaz TxD je neprekidno postavljen na "visok nivo" sve dok procesor (CPU) ne pošalje ka čipu **8251** prvi karakter, a to je obično karakter SYNC. Kada se signal CTS postavi na "nizak nivo", to znači da je prvi karakter poslat serijski. Svi karakteri se "šiftuju" na svaku rastuću ivicu signala TxC.

Kada otpočne slanje, podaci se srijski prosleđuju na izlazni pin TxD, i to brzinom jednakoj taktu TxC.

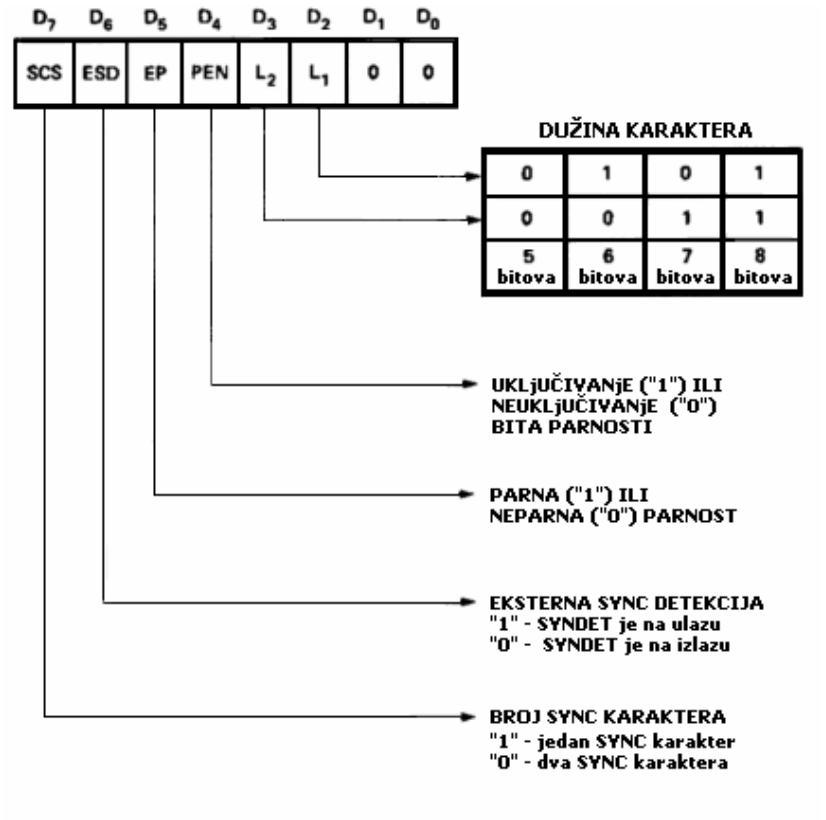
SYNC karakter(i) se automatski ubacuju u blok podatka koji se šalje preko izlaznog pina TxD. U ovom slučaju, pin TxEMPTY se postavlja na "visoko" u cilju signalizacije da je 8251 prazan i SYNC karakteri se šalju ka periferijskom uređaju. Kada se završilo "šiftovanje" SYNC karaktera (to znači da su poslati), pin TxEMPTY postavlja se na "nizak nivo". Dakle, pin TxEMPTY predstavlja interni reset i omogućava da se pošalje naredni karakter.



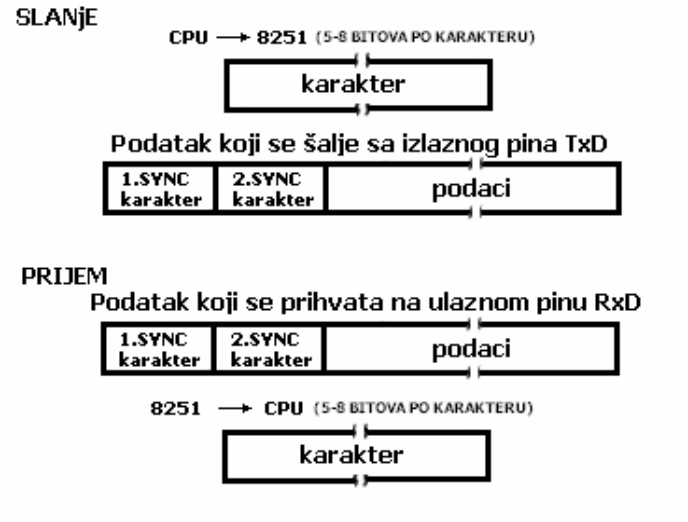
Sl.2.6. Ubacivanje SYNC karaktera

Sinhroni režim rada (sinhroni mod) – prijem podataka

U okviru ovog moda moguće su i interna i eksterna sinhronizacija. Ukoliko je prilikom programiranja izabrana interna sinhronizacija, prijemnik započinje rad u tzv. *HUNT* modu. Podaci na ulaznom pinu RxD se očitavaju na svaku opadajuću ivicu takta RxC. Sadržaj bafera Rx se neprekidno poredi sa prvim SYNC karakterom, sve do trenutka kada se pojavi odgovarajući karakter. Kada se detektuju oba SYNC karaktera, prijemnik završava HUNT mod i tada se nalazi u sinhronizaciji (sa karakterima koje prima iz procesora). Pin SYNDET postavlja se na "visoko stanje", **i resetuje automatski preko Status Read operacije.**



Sl.2.7. Format instrukcije moda za sinhroni režim



Sl.2.8. Sinhroni mod

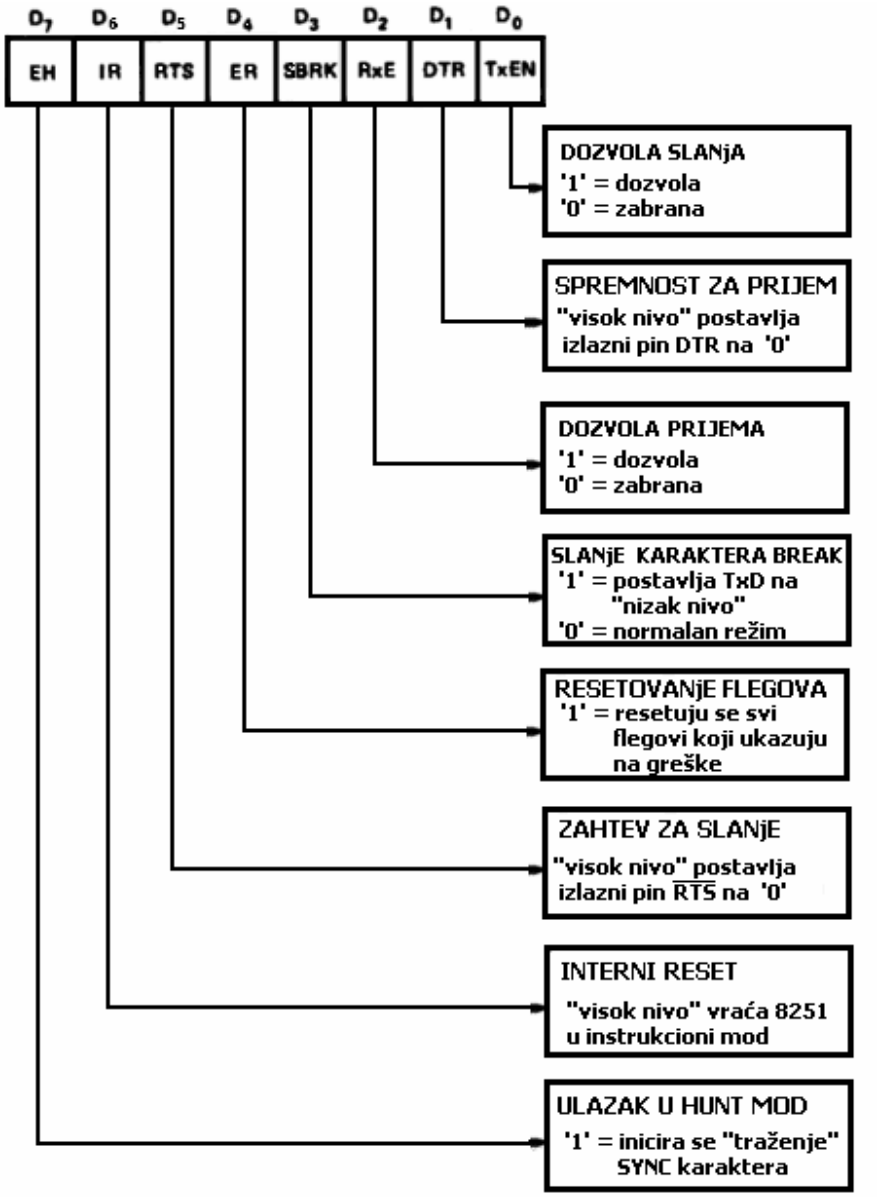
Ukoliko se želi eksterna sinhronizacija, dovodi se “visoki nivo” na pin SYNDET. Napominjemo da je dovoljno da impuls na pinu SYNDET traje jedan takt (RxC).

Ako se izgubi sinhronizacija, procesor (CPU) može da inicira ponovni ulazak prijemnika u *HUNT* mod.

Definicija komandne instrukcije

Čip **8251** je spreman za komunikaciju tek pošto se u njega upišu instrukcije režima rada i – ukoliko se radi o sinhronom modu – tzv. SYNC karakteri. Procesor najpre postavi kontrolni signal C/D na "1", a zatim učita komandnu instrukciju u **8251**.

Komandnim instrukcijama dozvoljava se slanje/prijem podataka, kontroliše rad modema, resetuje čip **8251** ukoliko je prisutna greška, kontrolišu se aktivnosti vezane za izabrani oblik komunikacije itd.

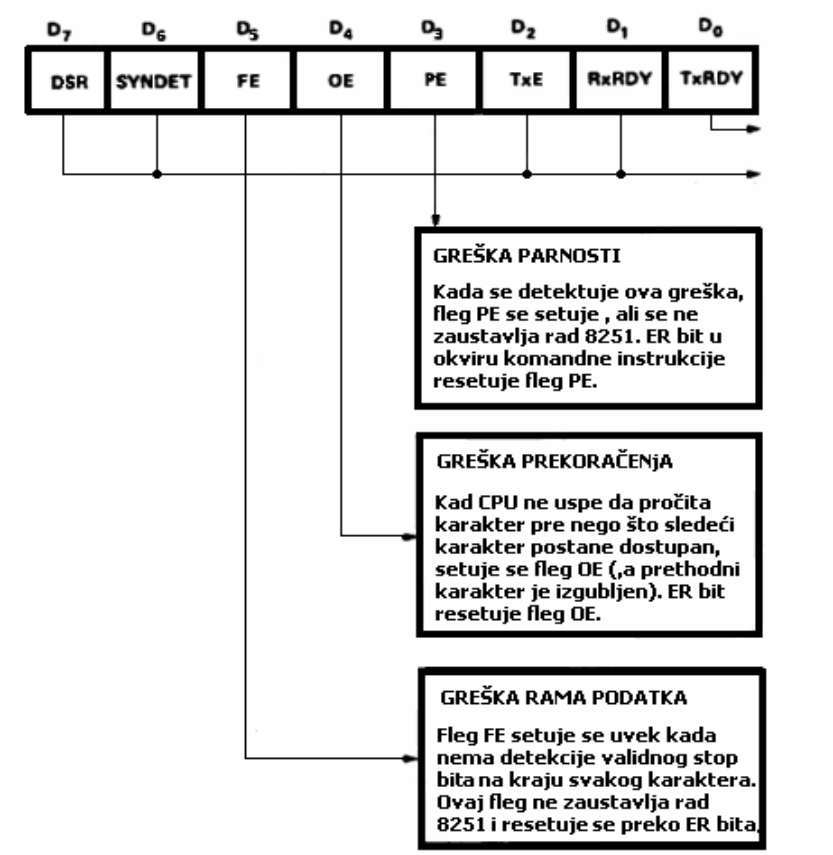


2.9. Format komandne instrukcije

Očitavanje statusa

Da bi komunikacioni sistemim pouzdano funkcionisao, vrlo često je neophodno da procesor (u svakom trenutku) ima informaciju o aktivnostima pojedinih delova (sistema). Na taj način, procesor (CPU) može da reaguje u slučaju kada su u sistemu prisutne greške. Naravno, informacija o statusu bitna je i za druge svrhe.

Procesor (CPU) izdaje komandu čitanja statusa preko ulaza C/D.

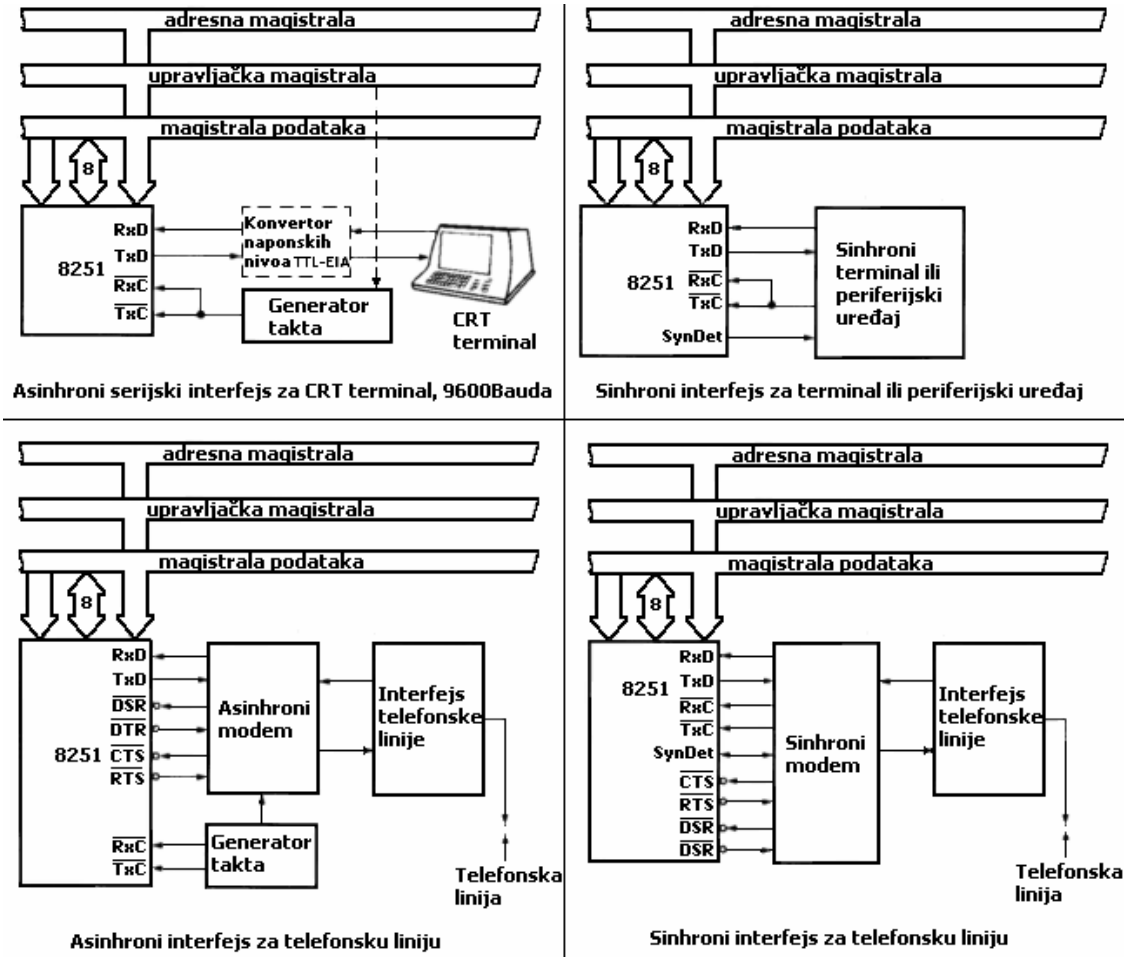


Sl.2.10. Format očitavanja statusa

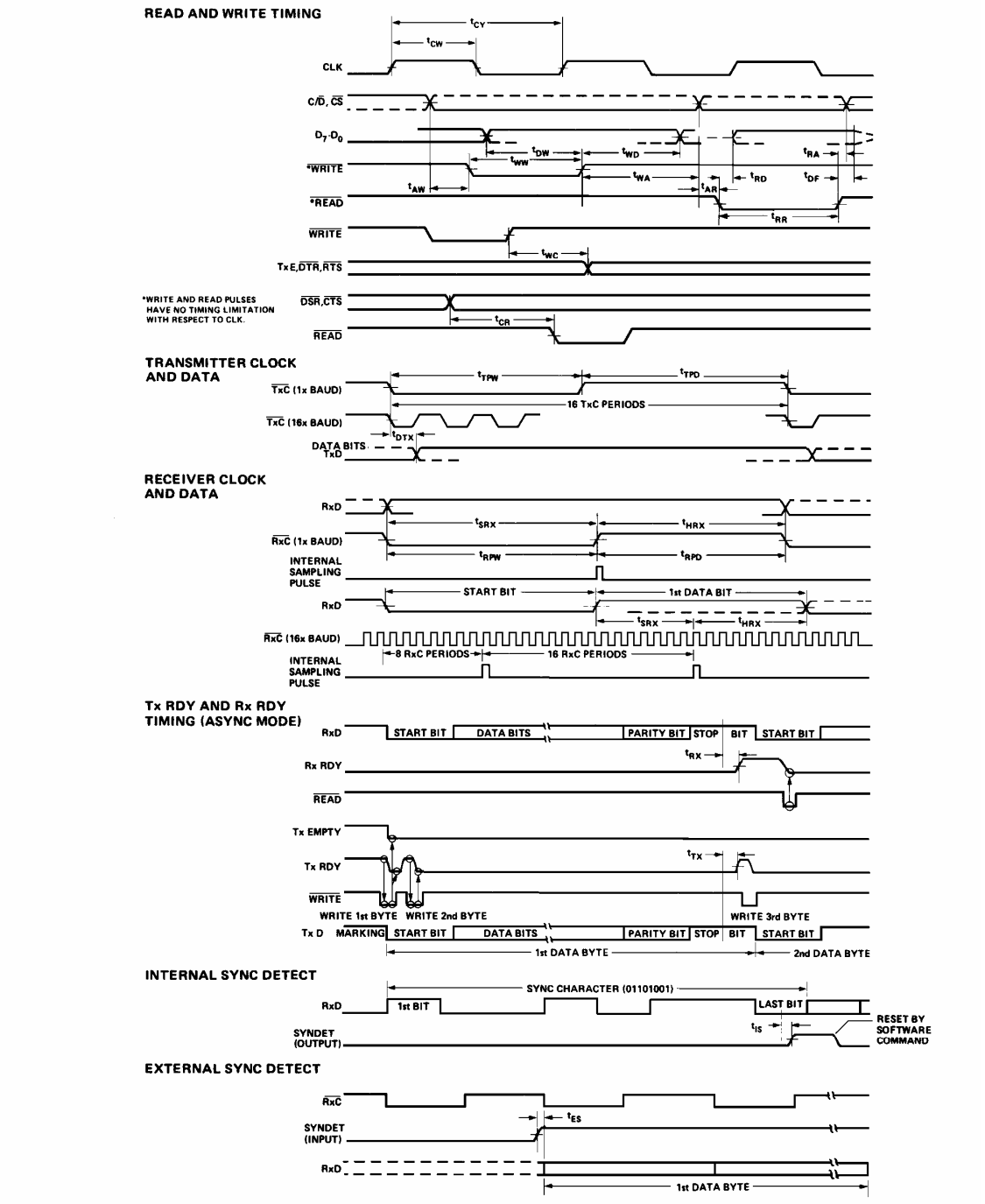
Čip **8251** može biti korišćen u aplikacijama gde se primenjuje tehnika “prozivanja” i u sistemima kojima se upravlja prekidima.

Kašnjenje ažuriranja statusa ne sme da bude duže od 16 taktova.

2.3.Primena

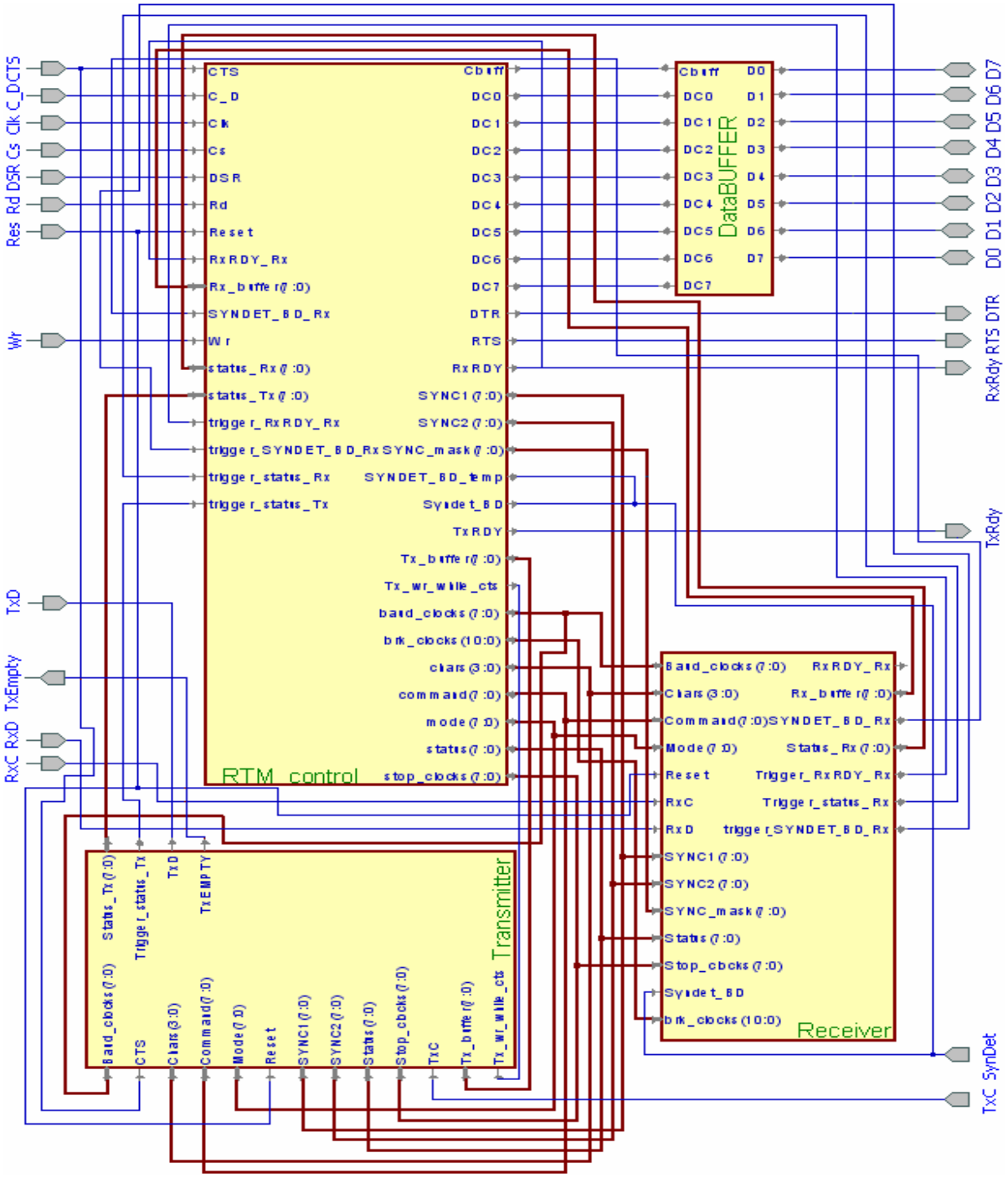

adresna magistrala
upravljačka magistrala
magistrala podataka
8
8251
RxD
TxD
RxC
TxC
Konvertor naponskih nivoa TTL-EIA
Generator takta
CRT terminal
Asinhroni serijski interfejs za CRT terminal, 9600Bauda
adresna magistrala
upravljačka magistrala
magistrala podataka
8
8251
RxD
TxD
RxC
TxC
SynDet
Sinhroni terminal ili periferijski uređaj
Sinhroni interfejs za terminal ili periferijski uređaj
adresna magistrala
upravljačka magistrala
magistrala podataka
8
8251
RxD
TxD
DSR
DTR
CTS
RTS
RxC
TxC
Asinhroni modem
Interfejs telefonske linije
Generator takta
Telefonska linija
Asinhroni interfejs za telefonsku liniju
adresna magistrala
upravljačka magistrala
magistrala podataka
8
8251
RxD
TxD
RxC
TxC
SynDet
CTS
RTS
DSR
DSR
Sinhroni modem
Interfejs telefonske linije
Telefonska linija
Sinhroni interfejs za telefonsku liniju

2.4.Signali


READ AND WRITE TIMING
CLK
C/D, CS
D7-D0
*WRITE
*READ
WRITE
TxE,DTR,RTS
DSR,CTS
READ
*WRITE AND READ PULSES HAVE NO TIMING LIMITATION WITH RESPECT TO CLK.
TRANSMITTER CLOCK AND DATA
TxC (1x BAUD)
TxC (16x BAUD)
DATA BITS TxD
16 TxC PERIODS
RECEIVER CLOCK AND DATA
RxD
RxC (1x BAUD)
INTERNAL SAMPLING PULSE
START BIT
1st DATA BIT
RxC (16x BAUD)
8 RxC PERIODS
16 RxC PERIODS
Tx RDY AND Rx RDY TIMING (ASYNC MODE)
START BIT
DATA BITS
PARITY BIT
STOP BIT
START BIT
Rx RDY
READ
Tx EMPTY
Tx RDY
WRITE
WRITE 1st BYTE
WRITE 2nd BYTE
WRITE 3rd BYTE
Tx D
MARKING
1st DATA BYTE
2nd DATA BYTE
INTERNAL SYNC DETECT
SYNC CHARACTER (01101001)
1st BIT
LAST BIT
SYNDET (OUTPUT)
RESET BY SOFTWARE COMMAND
EXTERNAL SYNC DETECT
SYNDET (INPUT)

3. PREDLOG REALIZACIJE ČIPA 8251 POMOĆU VHDL-a

*Na Slici 3.1. prikazan je predlog realizacije čipa **8251** pomoću softverskog alata **Active HDL** 5.1. Potom je izvršena njegova sinteza pomoću programskog paketa **ISE** kompanije **XILINX** .*



Sl.3.1. ***BDE*** dijagram čipa 8251

Realizacija čipa **8251** u VHDL-u je ostvarena u četiri celine (bloka):

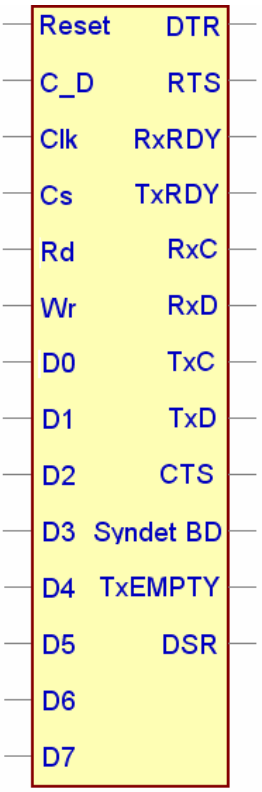
- Transmitter (Predajnik)
- Receiver (Prijemnik)
- RTM Control (Kontrolna upravljač jedinica čipa
- Data Buffer (Bafer magistrale podataka)

Naravno, bila je moguća realizacija i u šest celina. U tom slučaju blok-dijagram bi sadržao bafer prijemnika i bafer predajnika. Međutim, od takve realizacije se odustalo. Postojeća realizacija sa četiri celine (vidi Sliku 3.1) sadrži veliki broj eksternih i internih signala (44), a linije veze su «gusto» raspoređene. Sa dva dodatna bafera, broj signala bio bi uvećan na preko 60. Samim tim bi broj veza bio veliki i ne bi bilo moguće generisati blok-dijagram u okviru progama ***Active-HDL***.

Smatrali smo da je najbolje da pomenute bafere pridružimo celini (bloku) **RTM_control.** Na taj način su izdvojene četiri celine: veza sa CPU-om (data_buffer), upravljanje i kontrola čipa (RTM_control), predaja podataka (Transmitter) i prijem podataka (Receiver). Pomenute celine su sasvim dovoljne za shvatanje osnovnih funkcija čipa **8251**.

3.1. Čip 8251

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;



-- Čip 8251 ima 28 pinova, ali pinovi za napajanje (Vcc) i
-- i masu (Gnd) nisu od interesa za logičko projektovanje.



ENTITY A8251 IS
  PORT(

        D0       : inout std_logic;     -- Pinovi D0-D7 predstavljaju vezu čipa
        D1       : inout std_logic;     -- 8251 sa magistralom podataka
        D2       : inout std_logic;
        D3       : inout std_logic;
        D4       : inout std_logic;
        D5       : inout  std_logic;
        D6       : inout std_logic;
        D7       : inout std_logic;
        Reset : in std_logic;                    -- reset
        Cs       : in std_logic;                 -- selekcija cipa
        C_D   : in std_logic;                    -- komanda/podatak (zahtev procesora)
        Rd       : in std_logic;                 -- dozvola čitanja
        Wr       : in std_logic;                 -- dozvola upisa
        Clk   : in std_logic;                    -- takt
        RxC   : in std_logic;                    -- takt prijemnika
        TxC   : in std_logic;                    -- takt predajnika
        RxD   : in std_logic;                    -- ulazni pin za serijski prijem
        TxD   : out std_logic;                   -- izlazni pin za serijsko slanje
        TxEMPTY   : out std_logic;               -- indikacija da je predajnik spreman
                                                 -- za prijem novog podatka
        TxRDY     : out std_logic;               -- novi karakter za predaju
        RxRDY     : out std_logic;               -- novi karakter za prijem
        Syndet_BD: inout std_logic;              -- detekcija sinhronog karaktera
        DSR   : in std_logic;                    -- spremnost podatka
        CTS   : in std_logic;                    --"brisanje za novo slanje"
        DTR   : out std_logic;                   -- spremnost terminala
```



```
        RTS   : out std_logic                    -- zahtev za slanje
    );
END entity A8251;


ARCHITECTURE structure OF A8251 IS
  -- Deklarisanje internih signala

    Signal Cbuff :  std_logic;
    Signal  DC0:  std_logic;
    Signal  DC1:  std_logic;
    Signal  DC2:  std_logic;
    Signal  DC3:  std_logic;
    Signal  DC4:  std_logic;
    Signal  DC5:  std_logic;
    Signal  DC6:  std_logic;
    Signal  DC7:  std_logic;
    Signal  command : std_logic_vector(7 DOWNTO 0);
    Signal mode:  std_logic_VECTOR(7 downto 0);
    Signal Tx_buffer:  std_logic_VECTOR(7 downto 0);
    Signal Rx_buffer:  std_logic_VECTOR(7 downto 0);
    Signal SYNC1:  std_logic_VECTOR(7 downto 0);
    Signal SYNC2:  std_logic_VECTOR(7 downto 0);
    Signal SYNC_mask: std_logic_VECTOR(7 downto 0);
    Signal Tx_wr_while_cts:  std_logic;
    Signal baud_clocks:  std_logic_VECTOR(7 downto 0);
    Signal stop_clocks:  std_logic_VECTOR(7 downto 0);
    Signal brk_clocks:  std_logic_VECTOR(10 downto 0);
    Signal chars:  std_logic_VECTOR(3 downto 0);
    Signal status: std_logic_VECTOR(7 downto 0);
    Signal status_Rx:  std_logic_VECTOR(7 downto 0);
    Signal status_Tx:  std_logic_VECTOR(7 downto 0);
    Signal RxRDY_Rx:  std_logic;
    Signal SYNDET_BD_Rx:  std_logic;
    Signal SYNDET_BD_temp:  std_logic;
    Signal Trigger_status_Tx: std_logic;
    Signal Trigger_status_Rx: std_logic;
    Signal Trigger_RxRDY_Rx: std_logic;
    Signal trigger_SYNDET_BD_Rx: std_logic;


  -- Deklarisanje 4 komponente koje čine strukturu cipa 8251
```

```
Component DataBUFFER
        port (

        DC0      : inout std_logic;
        DC1      : inout std_logic;
        DC2      : inout std_logic;
        DC3      : inout std_logic;
        DC4      : inout std_logic;
        DC5      : inout  std_logic;
        DC6      : inout std_logic;
        DC7      : inout std_logic;
        D0       : inout std_logic;
        D1       : inout std_logic;
        D2       : inout std_logic;
        D3       : inout std_logic;
        D4       : inout std_logic;
        D5       : inout  std_logic;
        D6       : inout std_logic;
        D7       : inout std_logic;
        Cbuff : in  std_logic

                );

END Component;

Component RTM_control
        port (


        DC0      : inout std_logic;
        DC1      : inout std_logic;
        DC2      : inout std_logic;
        DC3      : inout std_logic;
        DC4      : inout std_logic;
        DC5      : inout  std_logic;
        DC6      : inout std_logic;
        DC7      : inout std_logic;
        Reset    : in std_logic;
        Cs       : in std_logic;
        C_D      : in std_logic;
        Rd       : in std_logic;
        Wr       : in std_logic;
        Clk      : in std_logic;
        Cbuff    : out std_logic;
        Command :inout std_logic_VECTOR(7 downto 0);
```

```
        mode        :out std_logic_VECTOR(7 downto 0);
        Tx_buffer  :out std_logic_VECTOR(7 downto 0);
        Rx_buffer  :in std_logic_VECTOR(7 downto 0);
        SYNC1      :out std_logic_VECTOR(7 downto 0);
        SYNC2      :out std_logic_VECTOR(7 downto 0);
        SYNC_mask:out std_logic_VECTOR(7 downto 0);
        status    :inout std_logic_VECTOR(7 downto 0);
        SYNDET_BD_temp:inout std_logic;
        Tx_wr_while_cts:out std_logic;
        baud_clocks:out std_logic_VECTOR(7 downto 0);
        stop_clocks:out std_logic_VECTOR(7 downto 0);
        brk_clocks :out std_logic_VECTOR(10 downto 0);
        chars      :out std_logic_VECTOR(3 downto 0);
        status_Rx  :in std_logic_VECTOR(7 downto 0);
        status_Tx  :in std_logic_VECTOR(7 downto 0);
        RxRDY_Rx    :in std_logic;
        SYNDET_BD_Rx:in std_logic;
        TxRDY            : out std_logic;
        RxRDY     : out std_logic;
        Syndet_BD: inout std_logic;
        trigger_status_Tx    :in std_logic;
        trigger_status_Rx    :in std_logic;
        trigger_SYNDET_BD_Rx :in std_logic;
        trigger_RxRDY_Rx     :in std_logic;
        DSR   : in std_logic;
        CTS   : in std_logic;
        DTR   : out std_logic;
        RTS   : out std_logic

                );

END Component;

Component Transmitter
        port (

                Reset:  in std_logic;
                Mode:   in std_logic_VECTOR(7 downto 0);
                Command:in std_logic_VECTOR(7 downto 0);
                Status: in std_logic_VECTOR(7 downto 0);
                Status_Tx  :out std_logic_VECTOR(7 downto 0);
                Tx_wr_while_cts: in std_logic;
                Trigger_status_Tx: inout std_logic;
                Tx_buffer: in std_logic_VECTOR(7 downto 0);
                Baud_clocks:in std_logic_VECTOR(7 downto 0);
                Stop_clocks:in std_logic_VECTOR(7 downto 0);
```

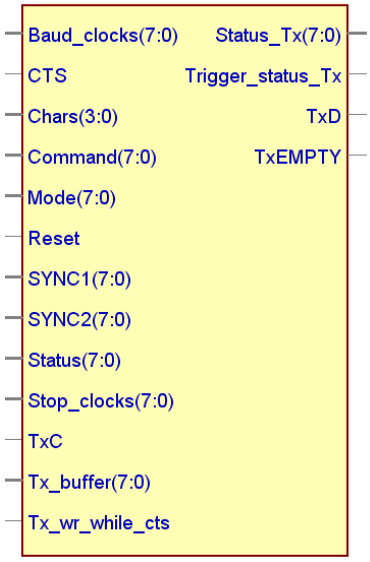
```
        Chars : in std_logic_VECTOR(3 downto 0);
        SYNC1 : in std_logic_VECTOR(7 downto 0);
        SYNC2 : in std_logic_VECTOR(7 downto 0);
        CTS   : in std_logic;
        TxC   : in std_logic;
        TxD   : out std_logic;
        TxEMPTY: out std_logic

        );

END Component;


 Component Receiver
        port (

        Reset : in std_logic;
        RxC   : in std_logic;
        RxD   : in std_logic;
        Mode:   in std_logic_VECTOR(7 downto 0);
        Command:in std_logic_VECTOR(7 downto 0);
        Status: in std_logic_VECTOR(7 downto 0);
        Status_Rx  :out std_logic_VECTOR(7 downto 0);
        Trigger_status_Rx: inout std_logic;
        Trigger_RxRDY_Rx: inout std_logic;
        trigger_SYNDET_BD_Rx: inout std_logic;
        Rx_buffer: out std_logic_VECTOR(7 downto 0);
        Baud_clocks:in std_logic_VECTOR(7 downto 0);
        Stop_clocks:in std_logic_VECTOR(7 downto 0);
        Chars : in std_logic_VECTOR(3 downto 0);
        RxRDY_Rx  : out std_logic;
        SYNDET_BD_Rx: inout std_logic;
        Syndet_BD : inout  std_logic;
        SYNC1 : in std_logic_VECTOR(7 downto 0);
        SYNC2 : in std_logic_VECTOR(7 downto 0);
        SYNC_mask: std_logic_VECTOR(7 downto 0);
        brk_clocks:in std_logic_VECTOR(10 downto 0)

        );

end Component;
```

```
BEGIN

    -- "Preslikavanje" portova komponenata

    I_DataBuffer : DataBuffer
          PORT MAP(
                              DC0 => DC0,
                              DC1 => DC1,
                              DC2 => DC2,
                              DC3 => DC3,
                              DC4 => DC4,
                              DC5 => DC5,
                              DC6 => DC6,
                              DC7 => DC7,
                              D0 => D0,
                              D1 => D1,
                              D2 => D2,
                              D3 => D3,
                              D4 => D4,
                              D5 => D5,
                              D6 => D6,
                              D7 => D7,
                              Cbuff => Cbuff
                                );

    I_RTM_control : RTM_control
          PORT MAP(

                              DC0 => DC0,
                              DC1 => DC1,
                              DC2 => DC2,
                              DC3 => DC3,
                              DC4 => DC4,
                              DC5 => DC5,
                              DC6 => DC6,
                              DC7 => DC7,
                              Reset => Reset,
                              Cs       => Cs,
                              C_D => C_D,
                              Rd =>  Rd,
                              Wr =>  Wr,
                              Clk => Clk,
                              Cbuff => Cbuff,
                              command => command,
                              mode => mode,
                              Tx_buffer => Tx_buffer,
```

```
                Rx_buffer => Rx_buffer,
                SYNC1 => SYNC1,
                SYNC2 => SYNC2,
                SYNC_mask =>  SYNC_mask,
                status => status,
                SYNDET_BD_temp => SYNDET_BD_temp,
                Tx_wr_while_cts => Tx_wr_while_cts,
                baud_clocks => baud_clocks,
                stop_clocks => stop_clocks,
                brk_clocks => brk_clocks,
                chars => chars,
                status_Rx => status_Rx,
                status_Tx => status_Tx,
                RxRDY_Rx => RxRDY_Rx,
                SYNDET_BD_Rx =>  SYNDET_BD_Rx,
                TxRDY => TxRDY,
                RxRDY => RxRDY,
                Syndet_BD => Syndet_BD,
                trigger_status_Tx => trigger_status_Tx,
                trigger_status_Rx => trigger_status_Rx,
                trigger_SYNDET_BD_Rx => trigger_SYNDET_BD_Rx,
                trigger_RxRDY_Rx =>  trigger_RxRDY_Rx,
                DSR => DSR,
                CTS => CTS,
                DTR => DTR,
                RTS => RTS

                );

  I_Transmitter : Transmitter
PORT MAP(
                Reset => Reset,
                mode => mode,
                command => command,
                status => status,
                Tx_wr_while_cts => Tx_wr_while_cts,
                Trigger_status_Tx => Trigger_status_Tx,
                Tx_buffer => Tx_buffer,
                Baud_clocks => Baud_clocks,
                Stop_clocks => Stop_clocks,
                Chars => Chars,
                SYNC1 => SYNC1,
                SYNC2 => SYNC2,
                CTS => CTS,
                TxC => TxC,
                TxD => TxD,
```

```
                TxEMPTY => TxEMPTY

                );

  I_Receiver : Receiver
PORT MAP(
                Reset => Reset,
                RxC => RxC,
                RxD => RxD,
                Mode => Mode,
                Command => Command,
                Status => Status,
                Status_Rx => Status_Rx,
                Trigger_status_Rx => Trigger_status_Rx,
                Trigger_RxRDY_Rx => Trigger_RxRDY_Rx,
                trigger_SYNDET_BD_Rx => trigger_SYNDET_BD_Rx,
                Rx_buffer => Rx_buffer,
                Baud_clocks => Baud_clocks,
                Stop_clocks => Stop_clocks,
                Chars => Chars,
                RxRDY_Rx => RxRDY_Rx,
                SYNDET_BD_Rx => SYNDET_BD_Rx,
                Syndet_BD => Syndet_BD,
                SYNC1 => SYNC1,
                SYNC2 => SYNC2,
                SYNC_mask => SYNC_mask,
                brk_clocks => brk_clocks

                );

  END structure;
```

3.2. Transmitter (Predajnik)



Blok **Transmitter** (predajnik) upravlja slanjem podataka bit-po-bit. Uređaj koji prima podatke šalje takt (TxC), kojim definiše brzinu slanja. Upravljački blok **RTM_control** šalje predajniku *komandu,* kao i informacije o *modu* (režimu rada) i *statusu.*

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;


entity Transmitter is
        port (


        Reset:  in std_logic;                                   -- reset
        Mode:   in std_logic_VECTOR(7 downto 0);                -- režim (mod)
        Command:in std_logic_VECTOR(7 downto 0);                -- komanda
        Status: in std_logic_VECTOR(7 downto 0);                -- status
        Status_Tx  :out std_logic_VECTOR(7 downto 0);           -- status predajnika
        Tx_wr_while_cts: in std_logic;                          -- upis
        Trigger_status_Tx: inout std_logic;                     -- promena statusa
        Tx_buffer: in std_logic_VECTOR(7 downto 0);             -- bafer predajnika
        Baud_clocks:in std_logic_VECTOR(7 downto 0);            -- brzina
        Stop_clocks:in std_logic_VECTOR(7 downto 0);            -- trajanje stop bitova
        Chars : in std_logic_VECTOR(3 downto 0);                -- broj karaktera
        SYNC1 : in std_logic_VECTOR(7 downto 0);                -- 1. sinhroni karakter
        SYNC2 : in std_logic_VECTOR(7 downto 0);                -- 2. sinhroni karakter
        CTS   : in std_logic;                                   -- "Brisanje za slanje"
        TxC   : in std_logic;                                   -- takt predajnika
        TxD   : out std_logic;                                  -- izlazni pin predajnika
        TxEMPTY: out std_logic                                  -- indikacija da je predajnik spreman
                                                                -- za prijem novog podatka
                );

                end entity Transmitter;

architecture Transmitter_1 of Transmitter is

begin

        Transmitter : process
```

```
variable parity             : std_logic;
variable serial_Tx_buffer   : std_logic_VECTOR(7 downto 0);
variable store_Tx_buffer    : std_logic_VECTOR(7 downto 0);
variable clk_count          : std_logic_VECTOR(7 downto 0);
variable char_bit_count     : std_logic_VECTOR(3 downto 0);


begin


  if ( RESET = '1') or ( command(6) = '1' ) then        -- resetovanje
              TxD <= '1';                               -- setovanje signala TxD i TxEmpty
              TxEMPTY <= '1';
              status_Tx <= status(7 downto 3) & '1' & status(1 downto 0);
              trigger_status_Tx <= not(trigger_status_Tx);

              wait until ( TxC = '0' ) and ( not TxC'stable );

  else

              -- Ukoliko je bafer predajnika pun...

       if (status(0) = '0') then

              --...ispituju se Tx, CTS i Tx_wr_while_cts
              if ( ( (CTS = '0') and (command(0) = '1') ) or ( Tx_wr_while_cts = '1' ) ) then

                  -- Ucitavanje podataka u serijski bafer

                  serial_Tx_buffer := Tx_buffer;
                  store_Tx_buffer := Tx_buffer;

                  -- Resetovanje TxEMPTY i setovanje TxRDY status bita

                      TxEMPTY <= '0';

                      if (command(2) = '1') then
                              status_Tx <= status(7 downto 3) & '0' & status(1) & '1';
                      else
                              status_Tx <= status(7 downto 3) & "001";
                      end if;

                      trigger_status_Tx <= not(trigger_status_Tx);
```

```
                if (mode(1 downto 0) /= "00") then

                -- Slanje Start bita

                        clk_count := baud_clocks;

        -- Određivanje broja taktnih ciklusa po bitu, (bitskog intervala),
        -- saglasno brzini prenosa

        while ( clk_count /= "00000000") loop
                TxD <= '0';
                wait until (TxC = '0') and (not TxC'stable);
                clk_count := clk_count - "00000001";
        end loop;

end if;

-- Slanje karaktera
char_bit_count := chars;

-- Određivanje broja bitova u karakteru

        while ( char_bit_count /= "0000") loop

                char_bit_count := char_bit_count - "0001";
                clk_count := baud_clocks;

-- Određivanje broja taktnih ciklusa po bitu, (bitskog intervala),
-- saglasno brzini prenosa

        while ( clk_count /= "00000000") loop
                TxD <= serial_Tx_buffer(0);
                wait until (TxC = '0') and (not TxC'stable);
                clk_count := clk_count - "00000001";
        end loop;

                serial_Tx_buffer := '0' & serial_Tx_buffer(7 downto 1);

end loop;

-- Slanje bita parnosti

if (mode(4) = '1') then                         -- dozvola parnosti

-- Određivanje bita parnosti (parna ili neparna parnost)
```

```
            parity :=   store_Tx_buffer(0) xor store_Tx_buffer(1) xor
            store_Tx_buffer(2) xor store_Tx_buffer(3) xor store_Tx_buffer(4) xor
            store_Tx_buffer(5) xor store_Tx_buffer(6) xor store_Tx_buffer(7) xor(not mode(5));
            clk_count := baud_clocks;                    -- slanje bita parnosi

            -- Određivanje broja taktnih ciklusa po bitu, (bitskog intervala),
            -- saglasno brzini prenosa

            while ( clk_count /= "00000000") loop
                  TxD <= parity;
                  wait until (TxC = '0') and (not TxC'stable);
                  clk_count := clk_count - "00000001";
            end loop;
      end if;

            -- Podatak je poslat. TxEmpty se postavlja na '1', što
            -- omogućava prihvatanje novog karaktera


      if ( not((((CTS = '0') and (command(0) = '1')) or (Tx_wr_while_cts = '1'))
            and (status(0) = '0'))) then
                  TxEMPTY <= '1';
                  status_Tx <= status(7 downto 3) & '1' & status(1 downto 0);
                  trigger_status_Tx <= not(trigger_status_Tx);
      end if;

      if (mode(1 downto 0) /= "00") then           -- asinhroni mod

      -- Slanje Stop bita

            clk_count := stop_clocks;

      -- Petljom se utvrđuje koliko taktnih ciklusa traje
      -- Stop bit

            while ( clk_count /= "00000000") loop
                  TxD <= '1';
                  wait until (TxC = '0') and (not TxC'stable);
                  clk_count := clk_count - "00000001";
            end loop;
      end if;

else

      -- Ako predajnik nije u funkciji ili ako su podaci
      -- "upisani" dok je CTS bio postavljen na "visoko" stanje
```

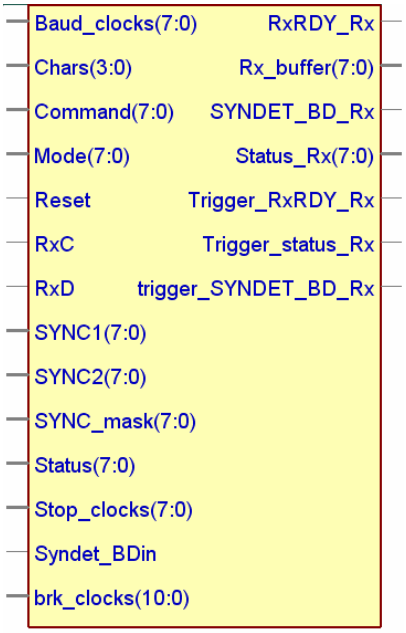
```
                TxD <= '1';                             -- markiranje
                TxEMPTY <= '1';
                wait until ( TxC = '0' ) and ( not TxC'stable );
    end if;

else                                                    -- ako je bafer predajnika prazan

                TxEMPTY <= '1';

        if (command(3) = '1') then                              -- slanje BREAK bita
                TxD <= '0';
                wait until ( TxC = '0' ) and ( not TxC'stable );
        else                                                    -- bez slanja BREAK bita

        if (mode(1 downto 0) = "00") then                       -- sinhroni mod

                if (CTS = '0') and (command(0) = '1') then      -- dozvola Tx

        -- Slanje prvog sinhronog karaktera   (SYNC1)

                serial_Tx_buffer := SYNC1;
                store_Tx_buffer := SYNC1;                       -- za parnost
                char_bit_count := chars;

        -- Slanje karaktera

        -- Petlja se izvršava onoliko puta koliko ima bitova u karakteru

                while ( char_bit_count /= "0000") loop

                char_bit_count := char_bit_count - "0001";
                clk_count := baud_clocks;

        -- Određivanje broja taktnih ciklusa po bitu, (bitskog intervala),
        -- saglasno brzini prenosa

                while ( clk_count /= "00000000") loop
                        TxD <= serial_Tx_buffer(0);
                        wait until (TxC = '0') and (not TxC'stable);
                        clk_count := clk_count - "00000001";
                end loop;
                        serial_Tx_buffer := '0' & serial_Tx_buffer(7 downto 1);
        end loop;

        if (mode(4) = '1') then                                 -- dozvola parnosti
```

```
-- Određivanje bita parnosti (parna ili neparna)

parity := store_Tx_buffer(0) xor store_Tx_buffer(1) xor
store_Tx_buffer(2) xor store_Tx_buffer(3) xor store_Tx_buffer(4) xor
store_Tx_buffer(5) xor store_Tx_buffer(6) xor store_Tx_buffer(7) xor (not mode(5));

clk_count := baud_clocks;

-- Slanje bita parnosti
-- Određivanje broja taktnih ciklusa po bitu,
-- saglasno brzini prenosa

        while ( clk_count /= "00000000") loop
                TxD <= parity;
                wait until (TxC = '0') and (not TxC'stable);
                clk_count := clk_count - "00000001";
        end loop;

end if;

if (mode(7) = '0') then                         -- u slucaju dva sinhrona karaktera

-- Slanje drugog sinhronog karaktera (SYNC2)

        serial_Tx_buffer := SYNC2;
        store_Tx_buffer := SYNC2;               -- za parnost
        char_bit_count := chars;

-- Slanje karaktera
-- Petlja se vrti onoliko puta koliko ima bitova u karakteru

        while ( char_bit_count /= "0000") loop
        char_bit_count := char_bit_count - "0001";
        clk_count := baud_clocks;

-- Određivanje broja taktnih ciklusa po bitu,
-- saglasno brzini prenosa

                while ( clk_count /= "00000000") loop
                        TxD <= serial_Tx_buffer(0);
                        wait until (TxC = '0') and (not TxC'stable);
                        clk_count := clk_count - "00000001";
                end loop;

                serial_Tx_buffer := '0' & serial_Tx_buffer(7 downto 1);
```

```
                end loop;

                if (mode(4) = '1') then          -- dozvola parnosti

        -- Određivanje bita parnosti (parna ili neparna)

                        parity := store_Tx_buffer(0) xor store_Tx_buffer(1) xor store_Tx_buffer(2) xor
                        store_Tx_buffer(3) xor store_Tx_buffer(4) xor store_Tx_buffer(5) xor
                        store_Tx_buffer(6) xor store_Tx_buffer(7) xor (not mode(5));

                        clk_count := baud_clocks;

                        -- Slanje bita parnosti

                        -- Odredjivanje broja taktnih ciklusa port bitu,
                        -- saglasno brzini prenosa

                        while ( clk_count /= "00000000") loop
                                TxD <= parity;
                                wait until (TxC = '0') and (not TxC'stable);
                                clk_count := clk_count - "00000001";
                        end loop;
                end if;
          end if;
     else                                                -- Tx nije aktivan

          TxD <= '1';                                    -- Setovanje signala TxD

          wait until ( TxC = '0' ) and ( not TxC'stable );

     end if;
  else                                                   -- asinhroni mod
          TxD <= '1';                                    -- Setovanje signala TxD

          wait until ( TxC = '0' ) and ( not TxC'stable );

                    end if;
               end if;
           end if;
    end if;

  end process transmitter;
end architecture Transmitter_1;
```

3.3. Receiver (Prijemnik)

Blok **Receiver** (prijemnik) upravlja prijemom podataka bit-po-bit. Važno je napomenuti da prijemnik dobija informacije o *modu* i *statusu*, kao i *komandu* od upravljačkog bloka **RTM_control,** a takt (RxC) od ulaznog uređaja koji šalje podatke.




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;


entity Receiver is
        port (


        Reset : in std_logic;                                   -- reset
        RxC   : in std_logic;                                   -- takt prijemnika
        RxD   : in std_logic;                                   -- ulazni pin
        Mode:   in std_logic_VECTOR(7 downto 0);                -- rezim (mod)
        Command:in std_logic_VECTOR(7 downto 0);                -- komanda
        Status: in std_logic_VECTOR(7 downto 0);                -- status
        Status_Rx  :out std_logic_VECTOR(7 downto 0);           -- status predajnika
        Trigger_status_Rx: inout std_logic;                     -- promena statusa
        Trigger_RxRDY_Rx: inout std_logic;                      -- promena RxRDy
        trigger_SYNDET_BD_Rx: inout std_logic;                  -- promena SYNDET_BD_Rx
        Rx_buffer: out std_logic_VECTOR(7 downto 0);            -- bafer predajnika
        Baud_clocks:in std_logic_VECTOR(7 downto 0);            -- brzina
        Stop_clocks:in std_logic_VECTOR(7 downto 0);            -- trajanje stop bitova
        Chars : in std_logic_VECTOR(3 downto 0);                -- broj karaktera
        RxRDY_Rx  : out std_logic;                               -- dolazak novog karaktera
        SYNDET_BD_Rx: inout std_logic;
        Syndet_BDin : in  std_logic;
        SYNC1 : in std_logic_VECTOR(7 downto 0);                -- 1. sinhroni karakter
        SYNC2 : in std_logic_VECTOR(7 downto 0);                -- 2. sinhroni karakter
        SYNC_mask: std_logic_VECTOR(7 downto 0);
        brk_clocks:in std_logic_VECTOR(10 downto 0)             -- trajanje BREAK signala

                );

end entity Receiver;
```

```
architecture Receiver_1 of Receiver is

begin

-- Proces unutar prijemnika
        receiver : process

-- Definisanje varijabli prijemnika
        variable  half_baud : std_logic_VECTOR (7 downto 0);
        variable  brk_count : std_logic_VECTOR(10 downto 0);
        variable status_var : std_logic_VECTOR(7 downto 0);
        variable sync_shift : std_logic_VECTOR(7 downto 0);
        variable parity : std_logic;
        variable got_sync : std_logic;
        variable  got_half_sync : std_logic;
        variable serial_Rx_buffer: std_logic_VECTOR(7 downto 0);
        variable store_Rx_buffer: std_logic_VECTOR(7 downto 0);
        variable clk_count  : std_logic_VECTOR(7 downto 0);
        variable char_bit_count: std_logic_VECTOR(3 downto 0);

                -- Varijabla next_cpu_control_word sadrzi sledecu upravljacku
                -- rec upcenu od strane procesora (CPU). Pomenuta varijabla
                -- je dvobitna i predstavlja:

                -- 00 = mod
                -- 01 = prvi sinhroni karakter
                -- 10 = drugi sinhroni karakter
                -- 11 = komanda

        begin

                if ( RESET = '1') or ( command(6) = '1' ) then       -- resetovanje

        -- Sa pojavom reseta, inicijalizuju se signali i varijable.

                        SYNDET_BD_Rx <= '0';
                        trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);
                        RxRDY_Rx <= '0';
                        trigger_RxRDY_Rx <= not(trigger_RxRDY_Rx);
                        got_half_sync := '0';

                        wait until (RxC = '1') and (not RxC'stable);

                else
```

```
-- Kada nema reset signala...

                    if (command(2) = '1') then                    -- Dozvola rada prijemnika

                       if (mode(1 downto 0) = "00") then          -- sinhroni mod

                          if (command(7) = '1') then              --  "HUNT" mod

                             if (mode(6) = '1') then              -- eksterni sinhroni mod

-- U slučaju eksterne sinhronizacije, pomoćni signal SYNDET_BD_Rx
-- uzima vrednost visoke impedanse

                                SYNDET_BD_Rx <= 'Z';
                                wait on SYNDET_BD_Rx;
-- Pojava spoljasnjeg SYNDET_BD_Rx, a samim tim i njegova promena
-- beleži se signalom trigger_SYNDET_BD_Rx

                                trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);

-- čip 8251 "čeka" pojavu usponske (rastuće) ivice (eksternog) signala SYNDET_BD.

                                wait until (SYNDET_BDin = '1') and (not SYNDET_BDin'stable);

                                SYNDET_BD_Rx <= '1';
                                trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);
                                status_Rx <= status(7) & '1' & status(5 downto 0);
                                trigger_status_Rx <= not(trigger_status_Rx);

-- Posle ostvarene sinhronizacije podaci se prikupljaju na
-- usponsku ivicu takta RxC

                                wait until (RxC = '1') and (not RxC'stable);

                    else                                          -- interni sinhroni mod

-- U slucaju interne sinhronizacije, resetuje se varijabla got_sync
-- i to pre početka LOOP petlje. Na ovaj način se indicira da nije
-- uspostavljena sinhronizacija

                                got_sync := '0';

-- Formiranje "hunt" petlje (HUNT LOOP) za uspostavljanje interne
-- sinhronizacije

                                while (got_sync = '0') loop
```

```
-- Bafer prijemnika se puni nulama, kako bi se izbegla lažna
-- detekcija sinhronog (SYNC) karaktera

                        serial_Rx_buffer := "00000000";
                        sync_shift := "00000000";

-- "Ubacuje" se petlja kojom se "šiftuju" (pomeraju za jedno mesto) bitovi signala RxD

                        while ( (SYNC_mask and sync_shift) /= SYNC1) loop

                        serial_Rx_buffer := RxD & serial_Rx_buffer(7 downto 1);

-- Formatiranje bitova u baferu prijemnika kako bi se olakšala
-- komparacija sa prvim sinhronim karakterom (SYNC1)


                        case (mode(3 downto 2)) is    -- duzina karaktera
                                when "00" =>
                                sync_shift := "000" & serial_Rx_buffer(7 downto 3);
                                when "01" =>
                                sync_shift := "00" & serial_Rx_buffer(7 downto 2);
                                when "10" =>
                                sync_shift := "0" & serial_Rx_buffer(7 downto 1);
                                when "11" =>
                                sync_shift := serial_Rx_buffer(7 downto 0);
                                when others =>
                        end case;

                        wait until (RxC = '1') and (not RxC'stable);

                end loop;

-- SYNC1 mora da se prihvati posle završetka petlje (loop)

-- U "hunt" modu se ne ispituje parnost sinhronih karaktera

                if (mode(4) = '1') then                 -- dozvola parnosti
                        wait until (RxC = '1') and (not RxC'stable);
                end if;

                if (mode(7) = '1') then                 -- jedan sinhroni karakter

-- U režimu sa jednim sinhronima karakterom, prihvatanjem
-- SYNC1 ostvaruje se sinhronizacija

                got_sync := '1';
```

```
                else                                    -- za dva sinhrona karaktera

-- U režimu sa dva sinhrona karaktera, "asemblira" se
-- sledeci karakter i poredi sa SYNC2, u cilju provere
-- sinhronizacije

                serial_Rx_buffer := "00000000";
                char_bit_count := chars;



-- Izračunavanje broja bitova po karakteru

                while (char_bit_count /= "0000") loop
                        serial_Rx_buffer := RxD & serial_Rx_buffer(7 downto 1);
                        char_bit_count := char_bit_count - "0001";
                        wait until (RxC = '1') and (not RxC'stable);
                end loop;



                case (mode(3 downto 2)) is    -- dužina karaktera
                        when "00" =>
                        serial_Rx_buffer := "000" & serial_Rx_buffer(7 downto 3);
                        when "01" =>
                        serial_Rx_buffer := "00" & serial_Rx_buffer(7 downto 2);
                        when "10" =>
                        serial_Rx_buffer := "0" & serial_Rx_buffer(7 downto 1);
                        when "11" =>
                        serial_Rx_buffer := serial_Rx_buffer(7 downto 0);
                        when others =>
                end case;

-- U "hunt" modu se ne ispituje parnost sinhronih karaktera

                if (mode(4) = '1') then                         -- dozvola parnosti
                        wait until (RxC = '1') and (not RxC'stable);
                end if;

-- Sinhronizacija ostvaruje van "hunt" petlje, ukoliko
-- je prihvaćen karakter SYNC2

                if (serial_Rx_buffer = SYNC2) then    -- if got sync
                        got_sync := '1';
                end if;
```

```
            end if;

        end loop;

-- Postavljanje (setovanje) signala SYNDET_BD u slučaju
-- ostvarene sinhronizacije

        SYNDET_BD_Rx <= '1';
        trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);

        if (command(0) = '1') then
                status_Rx <= status(7) & '1' & status(5 downto 0);
        else
                status_Rx <= status(7) & '1' & status(5 downto 3) & '1' & status(1 downto 0);
        end if;

        trigger_status_Rx <= not(trigger_status_Rx);

    end if;

end if;

-- "Asembliranje" bitova karaktera

        serial_Rx_buffer := "00000000";
        char_bit_count := chars;

-- Izračunavanje broja bitova po karakteru

        while (char_bit_count /= "0000") loop
                serial_Rx_buffer := RxD & serial_Rx_buffer(7 downto 1);
                char_bit_count := char_bit_count - "0001";
                wait until (RxC = '1') and (not RxC'stable);
        end loop;


        case (mode(3 downto 2)) is    -- dužina karaktera
                when "00" =>
                serial_Rx_buffer := "000" & serial_Rx_buffer(7 downto 3);
                when "01" =>
                serial_Rx_buffer := "00" & serial_Rx_buffer(7 downto 2);
                when "10" =>
                serial_Rx_buffer := "0" & serial_Rx_buffer(7 downto 1);
                when "11" =>
                serial_Rx_buffer := serial_Rx_buffer(7 downto 0);
```

```
                when others =>
        end case;

-- Provera parnosti

        if (mode(4) = '1') then                         -- dozvola parnosti

        parity := RxD;

        parity :=  serial_Rx_buffer(0) xor serial_Rx_buffer(1)
                xor serial_Rx_buffer(2) xor serial_Rx_buffer(3)
                xor serial_Rx_buffer(4) xor serial_Rx_buffer(5)
                xor serial_Rx_buffer(6) xor serial_Rx_buffer(7)
                xor (not mode(5)) xor parity;

-- Greška parnosti

-- Ukoliko se detektuje, postavlja se "marker" greške parnosti

        if (command(0) = '1') then
                status_Rx <= status(7 downto 4) & parity & status(2 downto 0);
        else
                status_Rx <= status(7 downto 4) & parity & '1' & status(1 downto 0);
        end if;

                trigger_status_Rx <= not(trigger_status_Rx);

                wait until (RxC = '1') and (not RxC'stable);

        end if;

                status_var := status;

-- Provera da li su sinhroni karakter(i) detektovani

-- Prvi sinhroni karakter

        if (got_half_sync = '1') then

-- Drugi sinhroni karakter

                if (serial_Rx_buffer = SYNC2) then

-- Postavljanje signala SYNDET_BD kako bi se označila
-- detekcija sinhronih karaktera
```

```
                    SYNDET_BD_Rx <= '1';
                    trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);

                    if (command(0) = '1') then
                        status_var := status_var(7) & '1' & status_var(5 downto 0);
                    else
                        status_var := status_var(7) & '1' & status_var(5 downto 3) &
                            '1' & status_var(1 downto 0);
                    end if;

                end if;

                got_half_sync := '0';

-- Ukoliko nije prihvacen SYNC1 ( u modu sa jednim sinhronim karakterom)

            else

-- Ispitivanje karaktera SYNC1

                if (serial_Rx_buffer = SYNC1) then

                    if (mode(7) = '0') then          --za dva sinhrona karaktera

-- U rezimu sa dva sinhrona karaktera, detekcija
-- prvog sinhronog karaktera (SYNC1) nije dovoljna da bi
-- se setovao SYNDET_BD. Potrebno je proveriti i karakter SYNC2.


                    got_half_sync := '1';

                    else                             --za jedan sinhroni karakter

-- U rezimu sa jednim sinhronim karakterom, detekcijom
-- prvog sinhronog karaktera (SYNC1) setuje se SYNDET_BD.

                    SYNDET_BD_Rx <= '1';
                    trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);

                        if (command(0) = '1') then
                        status_var := status_var(7) & '1' & status_var(5 downto 0);
                        else
                        status_var := status_var(7) & '1' & status_var(5 downto 3)
                            & '1' & status_var(1 downto 0);
                        end if;
```

```
                    end if;
                end if;

        end if;

-- Slanje prihvacenog karaktera u paralelni bafer

        Rx_buffer <= serial_Rx_buffer;

-- Ispitivanje statusa RxRDY

    if (status(1) = '1') then

-- Postavljanje "markera" prekoračenja, ukoliko prethodni
-- karakter nije prihvatio CPU

         if (command(0) = '1') then
                status_var := status_var(7 downto 5) & '1' & status_var(3 downto 0);
        else
                status_var := status_var(7 downto 5) & '1' & status_var(3) &
                        '1' & status_var(1 downto 0);
        end if;

    else

-- Postavljanje RxRDY u cilju obavestavanja procesora (CPU)
-- da prihvati novi karakter

                RxRDY_Rx <= '1';
                trigger_RxRDY_Rx <= not(trigger_RxRDY_Rx);

                if (command(0) = '1') then
                        status_var := status_var(7 downto 2) & '1' & status_var(0);
                else
                        status_var := status_var(7 downto 3) & "11" & status_var(0);
                end if;

        end if;

                status_Rx <= status_var;
                trigger_status_Rx <= not(trigger_status_Rx);

    else
```

```
-- Asinhroni režim rada
-- Dok se RxD ne postavi na "visoko" stanje, ne prihvata
-- Start bit novog karaktera

        if (RxD = '1') then

-- Postavljanje signala Break Detect (SYNDET_BD) na "nisko"
-- stanje kada RxD postane "visoko"

        brk_count := "00000000000";
        SYNDET_BD_Rx <= '0';
        trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);

                if (command(0) = '1') then
                        status_Rx <= status(7) & '0' & status(5 downto 0);
                else
                        status_Rx <= status(7) & '0' & status(5 downto 3) &
                                '1' & status(1 downto 0);
                end if;

                        trigger_status_Rx <= not(trigger_status_Rx);

-- Posle ostvarene sinhronizacije podaci se prikupljaju na
-- usponsku (rastuću) ivicu takta RxC


                wait until ((RxD = '0') and (not RxD'stable)) or (RESET = '1') or
                                (command(6) = '1');

-- Ako Reset nije aktivan

                if ((RESET = '0') and (command(6) = '0')) then

-- Uzorkovanje Start bita na sredini (za brzine 16X i 64X)
-- Varijabla "half_baud" je '0' za brzinu 1X

                half_baud := '0' & baud_clocks(7 downto 1);
                clk_count := half_baud;

--Potrebno je odrediti polovinu bitskog intervala
--kako bi se start bit uzorkovao na sredini

                while (clk_count /= "00000000") loop
                        wait until (RxC = '1') and (not RxC'stable);
                        clk_count := clk_count - "00000001";
                end loop;
```

```
-- Uzorkovanje Start bita na sredini bitskog intervala; na ovaj način se omogućava
-- detekcija lažnog Start bita

-- Ukoliko je detektovan pravi Start bit

                if (RxD = '0') then

-- Za brzinu 1X

                    if (mode(1 downto 0) = "01") then
                        wait until (RxC = '1') and (not RxC'stable);
                    end if;

--Petlja se izvršava sve dok se ne dostigne polovina bitskog intervala

                     clk_count := half_baud;                    -- half_baud je '0' za brzinu 1X

                while (clk_count /= "00000000") loop
                        wait until (RxC = '1') and (not RxC'stable);
                        clk_count := clk_count - "00000001";
                end loop;

                        brk_count := brk_count + ("000" & baud_clocks);

-- "Asembliranje" bitova karaktera

                        serial_Rx_buffer := "00000000";
                        char_bit_count := chars;

-- Izračunavanje broja karaktera po bitu

                        while (char_bit_count /= "0000") loop

-- Uzorkovanje Start bita na sredini bitskog intervala (za brzine 16X i 64X)
-- Varijabla "half_baud" je '0' za brzinu 1X

                        clk_count := half_baud;

-- Kod tela petlje se izvršava do sredine bitskog intervala

                        while (clk_count /= "00000000") loop
                                wait until (RxC = '1') and (not RxC'stable);
                                clk_count := clk_count - "00000001";
                        end loop;
```

```
-- Za brzinu 1X

                        if (mode(1 downto 0) = "01") then
                                wait until (RxC = '1') and (not RxC'stable);
                        end if;

-- Uzorkovanje na sredini bitskog intervala

                        serial_Rx_buffer := RxD & serial_Rx_buffer(7 downto 1);

                        if (RxD = '1') then

-- Postavljanje signala Break Detect (SYNDET_BD) na "nisko"
-- stanje kada RxD postane "visoko"

                        brk_count := "00000000000";
                        SYNDET_BD_Rx <= '0';
                        trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);

                                if (command(0) = '1') then
                                        status_var := status(7) & '0' & status(5 downto 0);
                                else
                                        status_var := status(7) & '0' & status(5 downto 3) & '1'
                                                &       status(1 downto 0);
                                end if;

                                        status_Rx <= status_var;
                                        trigger_status_Rx <= not(trigger_status_Rx);
                        else

-- "brk_count" se uvećava za vrednost taktnih
-- ciklusa po bitu


                                        brk_count := brk_count + ("000" & baud_clocks);
                        end if;

                                clk_count := half_baud;   -- half_baud = 0 za brzinu 1x

-- Kod tela petlje se izvršava do sredine bitskog intervala

                                while (clk_count /= "00000000") loop
                                        wait until (RxC = '1') and (not RxC'stable);
                                        clk_count := clk_count - "00000001";
                                end loop;
```

```
                    char_bit_count := char_bit_count - "0001";

                end loop;



                case (mode(3 downto 2)) is          -- dužina karaktera
                    when "00" =>
                    serial_Rx_buffer := "000" & serial_Rx_buffer(7 downto 3);
                    when "01" =>
                    serial_Rx_buffer := "00" & serial_Rx_buffer(7 downto 2);
                    when "10" =>
                    serial_Rx_buffer := "0" & serial_Rx_buffer(7 downto 1);
                    when "11" =>
                    serial_Rx_buffer := serial_Rx_buffer(7 downto 0);
                    when others =>
                    end case;

-- Bit parnosti

                    if (mode(4) = '1') then          -- dozvola parnosti

-- Uzorkovanje Start bita na sredini (za brzine 16X i 64X)
-- Varijabla "half_baud" je '0' za brzinu 1X

                    clk_count := half_baud;

-- Kod tela petlje se izvršava do sredine bitskog intervala

                    while (clk_count /= "00000000") loop
                        wait until (RxC = '1') and (not RxC'stable);
                        clk_count := clk_count - "00000001";
                    end loop;

-- Za brzinu 1X

                    if (mode(1 downto 0) = "01") then
                        wait until (RxC = '1') and (not RxC'stable);
                    end if;

-- Provera parnosti u centru bita parnosti

                    parity := RxD;

                    if (RxD = '1') then
```

```
-- Postavljanje Break Detect signala (SYNDET_BD) na
-- "nisko" stanje, ako je RxD na "visokom" stanju

                                brk_count := "00000000000";
                                SYNDET_BD_Rx <= '0';
                                trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);

                                        if (command(0) = '1') then
                                                status_var := status(7) & '0' & status(5 downto 0);
                                        else
                                                status_var := status(7) & '0' & status(5 downto 3)
                                                        & '1' &  status(1 downto 0);
                                        end if;

                                else

-- "brk_count" se uvećava za vrednost taktnih
-- ciklusa po bitu


                                        brk_count := brk_count + ("000" & baud_clocks);
                                end if;

-- Verifikacija parnosti

                                parity :=  serial_Rx_buffer(0) xor serial_Rx_buffer(1)
                                        xor serial_Rx_buffer(2) xor
                                serial_Rx_buffer(3) xor         serial_Rx_buffer(4)
                                        xor serial_Rx_buffer(5) xor
                                serial_Rx_buffer(6) xor serial_Rx_buffer(7)
                                        xor (not mode(5)) xor parity;

-- Greska parnosti

-- Postavljanje greske parnosti ukoliko je ona detektovana

                                if (command(0) = '1') then
                                        status_var := status_var(7 downto 4) & parity
                                                & status_var(2 downto 0);
                                else
                                        status_var := status_var(7 downto 4) & parity & '1'
                                                & status_var(1 downto 0);
                                end if;

                                if (mode(1) = '1') then         -- za brzine 16X ili 64X
                                        status_Rx <= status_var;
```

```
                    trigger_status_Rx <= not(trigger_status_Rx);
            end if;

            clk_count := half_baud;         -- half_baud = 0 za brzinu 1X

-- Kod tela petlje se izvršava do sredine bitskog intervala

                    while (clk_count /= "00000000") loop
                        wait until (RxC = '1') and (not RxC'stable);
                        clk_count := clk_count - "00000001";
                    end loop;

            end if;

-- Slanje primljenih podataka u paralelni bafer

            Rx_buffer <= serial_Rx_buffer;

-- Ispitivanje da li je bafer prijemnika pun

            if (status(1) = '1') then

-- Postavljanje (setovanje) "markera" (pokazivača), prekoračenja
-- ukoliko prethodni karakter nije pročitan

                    if (command(0) = '1') then
                        status_var := status_var(7 downto 5) & '1'
                                & status_var(3 downto 0);
                    else
                        status_var := status_var(7 downto 5) & '1'
                        & status_var(3)& '1' & status_var(1 downto 0);
                    end if;

            else

-- Setovanje (postavljanje)  RxRDY u cilju obaveštavanja
-- procesora (CPU) da prihvati novi karakter

                    RxRDY_Rx <= '1';
                    trigger_RxRDY_Rx <= not(trigger_RxRDY_Rx);

                    if (command(0) = '1') then
                        status_var := status_var(7 downto 2) & '1' &
                                    status_var(0);
                    else
                        status_var := status_var(7 downto 3) & "11"
```

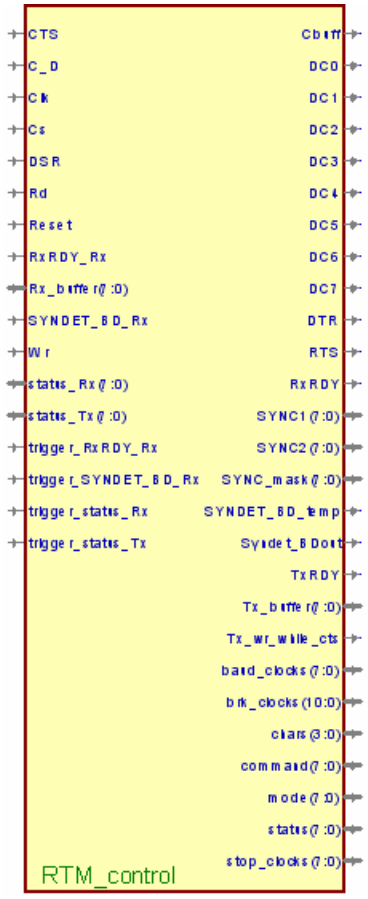
```
                                        status_var(0);
                    end if;

                end if;

                    status_Rx <= status_var;
                    trigger_status_Rx <= not(trigger_status_Rx);

-- Stop bit(ovi)

                    wait until (RxC = '1') and (not RxC'stable);

-- Provera prisustva "markera" (pokazivača) i Break-a

                if (RxD = '1') then

-- Setovanje signala Break Detect (SYNDET_BD)

                brk_count := "00000000000";
                SYNDET_BD_Rx <= '0';
                trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);

                    if (command(0) = '1') then
                        status_Rx <= status(7) & '0' & status(5 downto 0);
                    else
                        status_Rx <= status(7) & '0' & status(5 downto 3)
                            & '1' &status(1 downto 0);
                    end if;

                trigger_status_Rx <= not(trigger_status_Rx);

                else

-- Ako je RxD "nisko", postavlja se "markera" (pokazivača) greške u ramu podatka

                    if (command(0) = '1') then
                        status_Rx <= status(7 downto 6) & '1'
                            & status(4 downto 0);
                    else
                        status_Rx <= status(7 downto 6) & '1' &
                         status(4 downto 3) & '1' & status(1 downto 0);
                    end if;

                trigger_status_Rx <= not(trigger_status_Rx);
```

```
-- "brk_count" se uvećava za vrednost taktnih
-- ciklusa po bitu

                                brk_count := brk_count + ("000" & stop_clocks);

                                end if;

                        end if;

                end if;

            else
-- Ukoliko nije prihvaćen Start bit... čeka se njegova pojava

            wait until (RxC = '1') and (not RxC'stable);

            if (RxD = '0') then

-- Sve dok se ne prihvati Start bit, "brk_count" se uvećava


                brk_count := brk_count + "00000000001";


-- Detektovanje Break karaktera (SYNDET_BD)


                if (brk_count >= brk_clocks) then
                        SYNDET_BD_Rx <= '1';
                        trigger_SYNDET_BD_Rx <= not(trigger_SYNDET_BD_Rx);

                        if (command(0) = '1') then
                                status_Rx <= status(7) & '1' & status(5 downto 0);
                        else
                                status_Rx <= status(7) & '1' & status(5 downto 3) & '1'
                                        & status(1 downto 0);
                        end if;

                                trigger_status_Rx <= not(trigger_status_Rx);
                        end if;

                end if;

            end if;
```

```
            end if;

        else

         -- Ukoliko prijemnik nije u funkciji, resetuje se RxRDY_Rx

                                RxRDY_Rx <= '0';
                                trigger_RxRDY_Rx <= not(trigger_RxRDY_Rx);

                                wait until (RxC = '1') and (not RxC'stable);

                    end if;

               end if;

         end process receiver;

end architecture Receiver_1;
```

3.4. RTM_control (Kontrolna jedinica)

Blok **RTM_control** predstavlja upravljačku komponentu čipa 8251. Njegova uloga je da na osnovu komande koju dobija od strane procesora 8080, «režira» rada svih komponenata (blokova).




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
```

```
entity RTM_control is
        port (
```

```
DC0     : inout std_logic;                  -- Pinovi DC0-DC7 predstavljaju vezu
DC1     : inout std_logic;                  -- sa baferom magistrale podataka
DC2     : inout std_logic;
DC3     : inout std_logic;
DC4     : inout std_logic;
DC5     : inout  std_logic;
DC6     : inout std_logic;
DC7     : inout std_logic;
Reset   : in std_logic;                     -- reset
Cs      : in std_logic;                     -- selekcija cipa
C_D     : in std_logic;                     -- komanda/podatak (zahtev procesora)
Rd      : in std_logic;                     -- dozvola čitanja
Wr      : in std_logic;                     -- dozvola upisa
Clk     : in std_logic;                     -- takt
Cbuff   : out std_logic;                    -- dozvola upisa u bafer mag. podataka
command:inout std_logic_VECTOR(7 downto 0); -- komanda
mode   :out std_logic_VECTOR(7 downto 0);       -- režim (mod)
Tx_buffer:out std_logic_VECTOR(7 downto 0);    -- ka baferu predajnika
Rx_buffer:in std_logic_VECTOR(7 downto 0);     -- ka baferu predajnika
```

```
        SYNC1:out std_logic_VECTOR(7 downto 0);          -- 1.sinhroni karakter
        SYNC2:out std_logic_VECTOR(7 downto 0);          -- 2.sinhroni karakter
        SYNC_mask:out std_logic_VECTOR(7 downto 0);
        status :inout std_logic_VECTOR(7 downto 0);      -- status
        SYNDET_BD_temp    :inout std_logic;              -- detekcija sinhronog karaktera
        Tx_wr_while_cts   :out std_logic;
        baud_clocks:out std_logic_VECTOR(7 downto 0);   --brzina
        stop_clocks:out std_logic_VECTOR(7 downto 0);   -- trajanje bita Stop
        brk_clocks :out std_logic_VECTOR(10 downto 0); -- trajanje BREAK-a
        chars      :out std_logic_VECTOR(3 downto 0);   -- br. bitova u podatku
        status_Rx:in std_logic_VECTOR(7 downto 0);      -- informacija o statusu prijemnika
         status_Tx:in std_logic_VECTOR(7 downto 0);     -- informacija o statusu predajnika
        RxRDY_Rx :in std_logic;
        SYNDET_BD_Rx  :in std_logic;
        TxRDY: out std_logic;
        RxRDY: out std_logic;
        Syndet_BDout: out std_logic;
        trigger_status_Tx :in std_logic;                 -- promena statusa predajnika
        trigger_status_Rx  :in std_logic;                -- promena statusa prijemnika
        trigger_SYNDET_BD_Rx :in std_logic;
        trigger_RxRDY_Rx  :in std_logic;
        DSR   : in std_logic;                              -- spremnost podatka
        CTS   : in std_logic;                              --"brisanje za novo slanje"
        DTR   : out std_logic;                             -- spremnost terminala
        RTS   : out std_logic                              -- zahtev za slanje

        );

end entity RTM_control;

architecture RTM of RTM_control is

    -- Definisanje pomoćnih signala (subsignala).O ovim signalima biće reči kada
    -- budemo opisivali "okidni blok" (triggering block).

    signal status_main       : std_logic_VECTOR(7 downto 0);
    signal trigger_SYNDET_BD_main: std_logic := '0';
    signal trigger_status_main  : std_logic := '0';
    signal trigger_RxRDY_main   : std_logic := '0';
    signal SYNDET_BD_main        : std_logic;
    signal RxRDY_main           : std_logic;


    --**************************************************************

begin
```

```
-- Glavni proces
main : process

-- ****************************************************************

        -- Iz razloga sto određeni signali (mode, command, stop_clocks,
        -- baud_clocks, status, chars) ne preuzimaju nove vrednosti
        -- odmah, neophodno je definisati odgovarajuće varijable
        -- (mode_var, command_var, baud_clocks_var, stop_clocks_var,
        -- status_var, chars_var). Pomenute varijable biće korišćene
        -- i izvan glavnog procesa (main process).

        variable mode_var          : std_logic_VECTOR(7 downto 0);
        variable status_var        : std_logic_VECTOR(7 downto 0);
        variable command_var       : std_logic_VECTOR(7 downto 0);
        variable baud_clocks_var   : std_logic_VECTOR(7 downto 0);
        variable stop_clocks_var   : std_logic_VECTOR(7 downto 0);
        variable chars_var         : std_logic_VECTOR(3 downto 0);

        -- Varijabla next_cpu_control_word sadrži sledeću upravljačku
        -- reč upućenu od strane procesora (CPU). Pomenuta varijabla
        -- je dvobitna i predstavlja:

        -- 00 = mod
        -- 01 = prvi sinhroni karakter
        -- 10 = drugi sinhroni karakter
        -- 11 = komanda

        variable next_cpu_control_word : std_logic_VECTOR(1 downto 0);



        variable SYNC_var   : std_logic_VECTOR(7 downto 0);
        variable temp       : std_logic_VECTOR(10 downto 0);
        -- ****************************************************************


begin

        -- Inicijalizacija portova, signala i varijabli sa
        -- pojavom Reset signala. Inicijalizacija se obavlja
        -- na prednju ivicu takta (Clk) i kada je čip
        -- selektovan (Cs='0').

        wait until ( Clk = '1' ) and ( not Clk'stable );
```

```
if (Cs= '0') then

        if ( Reset = '1') or ( command_var(6) = '1' ) then

                DTR <= '1';
                RTS <= '1';

                command_var := "00000000";
                command <= command_var;

                status_var := "00000101";
                status_main <= status_var;
                trigger_status_main <= not(trigger_status_main); --Beleži promenu

                Tx_wr_while_cts <= '0';

                -- Posle reseta u Intel_8251 treba da se upiše
                -- informacija o rezimu rada (modu).

                next_cpu_control_word := "00";

        else
                -- Ukoliko nije aktivan signal Reset, ispituje se
                -- da li je aktivan signal čitanja (Rd)...

                if (Rd = '0') then

                        --... Zatim se ispituje da li je od strane procesora
                        -- zadato čitanje statusa (C_D). Ukoliko jeste...

                        if (C_D = '1') then

                                --...čita se vrednost sa DSR ulaza. Potom se vrednost
                                -- varijable status_var "puni" na sledeći način:

                                status_var := not(DSR) & status(6 downto 0);

                                -- Informacija o statusu se upućuje prema procesoru
                                -- preko BAFERA magistrale podataka.
                                Cbuff<= '1';
                                DC0 <= status_var(0);
                                DC1 <= status_var(1);
                                DC2 <= status_var(2);
                                DC3 <= status_var(3);
                                DC4 <= status_var(4);
```

```
                    DC5 <= status_var(5);
                    DC6 <= status_var(6);
                    DC7 <= status_var(7);

                if ( mode_var(1 downto 0) = "00") then          -- sinhroni mod


-- Pomoćni signal SYNDET_BD_main se resetuje. Nosilac informacije
-- o promeni ovog signala je trigger_SYNDET_BD_main.

                    SYNDET_BD_main <= '0';
                    trigger_SYNDET_BD_main <= not(trigger_SYNDET_BD_main);

-- Signal SYNDET_BD se resetuje. Nosilac informacije
-- o promeni ovog signala je trigger_SYNDET_BD_main.
-- Menja se status 8251 i to "belezi" trigger_status_main.

                    status_var := status(7) & '0' & status(5 downto 0);
                    status_main <= status_var;
                    trigger_status_main <= not(trigger_status_main);
                    end if;

                    else

-- Ukoliko se čita stanje ulaznog pina Rx, ispituje se da
-- li postoji dozvola čitanja (RxENABLE). Ako postoji...

                if (command_var(2) = '1') then    --Ukoliko postoji dozvola rada   prijemnika

-- ...Čip 8251 prosleđuje podatak koji je bio "čuvan"
-- u baferu prijemnika (Receive buffer).
                                        Cbuff<= '1';
                                        DC0 <= Rx_buffer(0);
                                        DC1 <= Rx_buffer(1);
                                        DC2 <= Rx_buffer(2);
                                        DC3 <= Rx_buffer(3);
                                        DC4 <= Rx_buffer(4);
                                        DC5 <= Rx_buffer(5);
                                        DC6 <= Rx_buffer(6);
                                        DC7 <= Rx_buffer(7);

-- Posle čitanja podatka, resetuje se ko-signal RxRDY_main.
-- RxRDY_main ćemo kasnije "preslikati" u RxRDY. Nosilac
-- informacije o promeni je signal trigger_RxRDY_main.

                                RxRDY_main <= '0';
```

```
                                trigger_RxRDY_main <= not(trigger_RxRDY_main);

-- Menja se status 8251 i to "belezi" trigger_status_main.
                                status_var := status(7 downto 2) & '0' & status(0);
                                status_main <= status_var;
                                trigger_status_main <= not(trigger_status_main);

                                end if;

                        end if;

-- Ako je od strane procesora zadata operacija upisa (Wr)...

                        elsif (Wr = '0') then

--...pinovi za podatke (Data) postavljaju se u stanje
-- visoke impedanse.
                                DC0 <= 'Z';
                                DC1 <= 'Z';
                                DC2 <= 'Z';
                                DC3 <= 'Z';
                                DC4 <= 'Z';
                                DC5 <= 'Z';
                                DC6 <= 'Z';
                                DC7 <= 'Z';
                                wait for 0 ns;
-- Ispituje se da li se upisuje komanda, mod ili sinhroni
-- karakter.
                                if (C_D = '1') then


-- Upis režima rada (moda)

                                        case (next_cpu_control_word) is

                                                when "00" =>

                                                        mode_var(0) := DC0;
                                                        mode_var(1) := DC1;
                                                        mode_var(2) := DC2;
                                                        mode_var(3) := DC3;
                                                        mode_var(4) := DC4;
                                                        mode_var(5) := DC5;
                                                        mode_var(6) := DC6;
                                                        mode_var(7) := DC7;
```

```
                    mode <= mode_var;

-- Određivanje broja bitova po karakteru

            chars_var := "0101" + ("00" & mode_var(3 downto 2) );
            chars <= chars_var;

                if ( mode_var(1 downto 0) = "00") then   -- sinhroni mod

-- Upisivanje sledeće kontrolne reci

                if ( mode_var(6) = '1') then        -- eksterni sinhroni mod
                    next_cpu_control_word := "11";-- komandna rec
                else                                 -- interni sinhroni mod
                    next_cpu_control_word := "01"-- prvi sinhroni
                    end if;                          -- karakter

-- U sinhronom režimu rada ,podatak (data) i bit parnosti upisuju
-- se u okviru jednog taktnog ciklusa. Nema stop bita.

                    stop_clocks <= "00000000";
                    stop_clocks_var := "00000000";
                    baud_clocks <= "00000001";
                    baud_clocks_var := "00000001";

                else                                -- asinhroni mod

-- Upisivanje sledece kontrolne reci

                next_cpu_control_word := "11"; -- komanda

-- Određivanje broja taktnih ciklusa za prenos podatka (Data)


            case ( mode_var(1 downto 0)) is-- postavljanje brzine prenosa


                    when "01" =>
                    baud_clocks_var := "00000001";
                    baud_clocks <= baud_clocks_var;
                    when "10" =>
                    baud_clocks_var := "00010000";
                    baud_clocks <= baud_clocks_var;
                    when "11" =>
                    baud_clocks_var := "01000000";
                    baud_clocks <= baud_clocks_var;
```

```
                                        when others =>

                                end case;


        -- Određivanje broja stop bitova po taktnom ciklusu

                                        case ( mode_var(7 downto 6)) is

                                                when "01" =>
                                                stop_clocks_var := baud_clocks_var;
                                                stop_clocks <= stop_clocks_var;
                                                when "10" =>
                                                stop_clocks_var := baud_clocks_var(7 downto 0)
                                                        +('0' & baud_clocks_var(7 downto 1) );
                                                stop_clocks <= stop_clocks_var;
                                                when "11" =>
                                                stop_clocks_var := baud_clocks_var(6 downto 0)
                                                        & '0';
                                                stop_clocks <= stop_clocks_var;
                                                when others =>

                                        end case;

        -- Izračunavanje broja taktova koji proteknu od detekcije
        -- Break-a do momenta kada RxD postane "nisko"

        -- Izračunavanje trajanja Start bita

                                        temp := "000" & baud_clocks_var;

        -- Izračunavanje trajanja karaktera

                                        while ( chars_var /= "0000") loop
                                                temp := temp + ( "000" & baud_clocks_var);
                                                chars_var := chars_var - "0001";
                                        end loop;

        -- Izračunavanje trajanja bita parnosti

                                        if (mode_var(4) = '1') then     -- ukoliko ima dozvole
parnosti
                                                temp := temp + ( "000" & baud_clocks_var);
                                        end if;

        -- Izračunavanje trajanja Stop bita
```

```
                                        temp := temp + ( "000" & stop_clocks_var);

        -- Varijabla temp se udvostručava, jer RxD treba da bude na
        -- "niskom" stanju tokom sekvence od dva karaktera

                                        brk_clocks <= temp(9 downto 0) & '0';

                                    end if;

-- Kada treba da primi prvi sinhroni karakter, next_cpu_control_word
-- uzima vrednost "01".
                                    when "01" =>

-- Prikazivanje prvog sinhronog karaktera sa magistrale podataka.

                                    SYNC_var(0) := DC0;
                                    SYNC_var(1) := DC1;
                                    SYNC_var(2) := DC2;
                                    SYNC_var(3) := DC3;
                                    SYNC_var(4) := DC4;
                                    SYNC_var(5) := DC5;
                                    SYNC_var(6) := DC6;
                                    SYNC_var(7) := DC7;

-- Upisivanje sledece kontrolne reci

                                    if (mode_var(7) = '0') then
                                    next_cpu_control_word := "10";    -- drugi sinhroni karakter
                                    else
                                    next_cpu_control_word := "11";    -- komanda
                                    end if;

-- Uključivanje prvog sinhronog karaktera u format podatka.

                                    case (mode_var(3 downto 2)) is        -- duzina karaktera
                                        when "00" =>
                                        SYNC1 <= "000" & SYNC_var(4 downto 0);
                                        SYNC_mask <= "00011111";
                                        when "01" =>
                                        SYNC1 <= "00" & SYNC_var(5 downto 0);
                                        SYNC_mask <= "00111111";
                                        when "10" =>
                                        SYNC1 <= "0" & SYNC_var(6 downto 0);
                                        SYNC_mask <= "01111111";
                                        when "11" =>
```

```
                    SYNC1 <= SYNC_var;
                    SYNC_mask <= "11111111";
                    when others =>

                end case;

                when "10" =>                    -- drugi sinhroni karakter
-- Prikazivanje sinhronog karaktera sa magistrale podataka.

                SYNC_var(0) := DC0;
                SYNC_var(1) := DC1;
                SYNC_var(2) := DC2;
                SYNC_var(3) := DC3;
                SYNC_var(4) := DC4;
                SYNC_var(5) := DC5;
                SYNC_var(6) := DC6;
                SYNC_var(7) := DC7;
-- Upisivanje sledece kontrolne reči. To je komanda.

                next_cpu_control_word := "11";
-- Pridruživanje drugog sinhronog karaktera u format podatka.

                case (mode_var(3 downto 2)) is      -- dužina karaktera
                    when "00" =>
                    SYNC2 <= "000" & SYNC_var(4 downto 0);
                    when "01" =>
                    SYNC2 <= "00" & SYNC_var(5 downto 0);
                    when "10" =>
                    SYNC2 <= "0" & SYNC_var(6 downto 0);
                    when "11" =>
                    SYNC2 <= SYNC_var;
                    when others =>

                end case;

                when "11" =>                        -- komanda
-- Čitanje komande sa magistrale podataka

                command_var(0) := DC0;
                command_var(1) := DC1;
                command_var(2) := DC2;
                command_var(3) := DC3;
```

```
                                        command_var(4) := DC4;
                                        command_var(5) := DC5;
                                        command_var(6) := DC6;
                                        command_var(7) := DC7;

                                        command <= command_var;

-- Upisivanje sledece kontrolne reči. To je sledeća komanda,
-- ukoliko nema signala Reset.

                                        next_cpu_control_word := "11";
                                        status_var := status;

-- Ukoliko prijemnik (Receiver) nije u funkciji,
-- resetuje se signal RxRDY.

                                        if (command_var(2) = '0') then          -- Dozvola rada prijemnika
                                                RxRDY_main <= '0';
                                                trigger_RxRDY_main <= not(trigger_RxRDY_main);
                                                status_var := status(7 downto 2) & '0' & status(0);
                                        end if;

-- Resetovanje markera greške, zavisno od komandne reci.

                                        if (command_var(4) = '1') then
                                                status_var := status_var(7 downto 6) &
                                                "000" & status_var(2 downto 0);
                                        end if;

-- Ažuriranje statusa

                                        status_main <= status_var;
                                        trigger_status_main <= not(trigger_status_main);

-- Postavljanje signala na izlaznim pinovima RTS i DTR,
-- zavisno od komandne reci

                                        RTS <= not(command_var(5));
                                        DTR <= not(command_var(1)) ;

                                        when others =>

                                        end case;

                                else                                    -- ako je upisan podatak za slanje
```

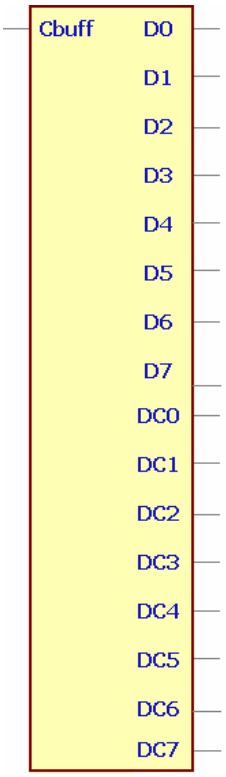
```
                        if (command_var(0) = '1') then          -- Dozvola rada predajnika
-- Podatak koji procesor šalje se upisuje u bafer magistrale
-- podataka...

                        case (mode_var(3 downto 2)) is          -- dužina karaktera
                                when "00" =>
                                Tx_buffer <= "000" & DC4 & DC3 & DC2 & DC1 &
                                                DC0;

                                when "01" =>
                                Tx_buffer <= "00" & DC5 & DC4 & DC3 & DC2 &
                                                DC1 & DC0;

                                when "10" =>
                                Tx_buffer <= "0" & DC6 & DC5 & DC4 & DC3 & DC2
                                                & DC1 & DC0;

                                when "11" =>
                                Tx_buffer <= DC7 & DC6 & DC5 & DC4 & DC3 &
                                                DC2 & DC1 & DC0;
                                when others =>

                                        null;

                        end case;

--... Kada je podatak učitan, resetuje se TxRDY.

                                status_var := status(7 downto 1) & '0';
                                status_main <= status_var;
                                trigger_status_main <= not(trigger_status_main);

-- Formiranje signala Tx_wr_while_cts na osnovu CTS.

                                if (CTS = '0') then
                                        Tx_wr_while_cts <= '1';
                                else
                                        Tx_wr_while_cts <= '0';
                                end if;

                        end if;

                end if;

                else
```

```
                        end if;
                    end if;
            end if;
end process main;

-- ***************************************************************************

-- ***************************************************************************
TxRDY_pin : block
-- ***************************************************************************
            begin
            -- Izlazni signal cipa TxRDY zavisi od signala CTS i TxENABLE. Kako
            -- signal CTS moze da se promeni u bilo kojem trenutku, definisali
            -- smo dodatni blok koji će svaku promenu CTS
            -- komande I statusa da “preslika” u TxRDy.
            TxRDY <= (not CTS) and command(0) and status(0);
end block TxRDY_pin;
-- ***************************************************************************
Triggering : block
-- ***************************************************************************
            begin
            -- Interni signali "status" i SYNDET_BD, kao i RxRDY (izlazni signal čipa
            -- 8251) "trpe" promenu u sva tri procesa. Zbog toga smo svaki od ova tri
```

```
            -- signala razdvojili u  pomoćne signale. Tako npr. postoje status_main,
            -- status_Rx i status_Tx itd. Zato, uvodimo "okidni blok" koji registruje
            -- svaku promenu "ko-signala". Dakle, "okidni blok" vrši nadgledanje procesa
            -- Ukoliko se "ko-signal" promenio, odgovarajući signal preuzima novu vrednost.


            status <= status_main when (not trigger_status_main'stable) else
            status_Rx when (not trigger_status_Rx'stable) else
            status_Tx when (not trigger_status_Tx'stable) else
            status;

            SYNDET_BD_temp <= SYNDET_BD_main when
                    (not trigger_SYNDET_BD_main'stable) else
            SYNDET_BD_Rx when (not trigger_SYNDET_BD_Rx'stable) else
                    SYNDET_BD_temp;
            Syndet_BD <= SYNDET_BD_temp;
            RxRDY <= RxRDY_main when (not trigger_RxRDY_main'stable) else
            RxRDY_Rx when (not trigger_RxRDY_Rx'stable) else
            status(1);
        end block triggering;

end architecture RTM;
```

3.5. DataBuffer (Bafer magistrale podataka)

Blok **DataBuffer** skladišti 8-bitni podatak, koji prihvata sa magistrale podataka procesora ili od strane bloka **RTM_control**. Radom bafera upravlja **RTM_control**, preko signala Cbuff.




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity DataBUFFER is
        port (

        DC0     : inout std_logic;      -- Pinovi DC0-DC7 predstavljaju vezu
        DC1     : inout std_logic;      -- sa unutrasnjim delom čipa 8251
        DC2     : inout std_logic;
        DC3     : inout std_logic;
        DC4     : inout std_logic;
        DC5     : inout std_logic;
        DC6     : inout std_logic;
        DC7     : inout std_logic;
        D0      : inout std_logic;      -- Pinovi DC0-DC7 predstavljaju vezu
        D1      : inout std_logic;      -- sa  magistralom podataka
        D2      : inout std_logic;
        D3      : inout std_logic;
        D4      : inout std_logic;
        D5      : inout std_logic;
        D6      : inout std_logic;
        D7      : inout std_logic;
        Cbuff : in    std_logic

            );
end entity DataBUFFER;

architecture DataBUFF of DataBUFFER is
begin

proc_buff: process (Cbuff)
begin

-- Signal Cbuff inicira kontrola (RTM_control)

        if Cbuff='1' then                       -- dozvola upisa na magistralu podataka
```

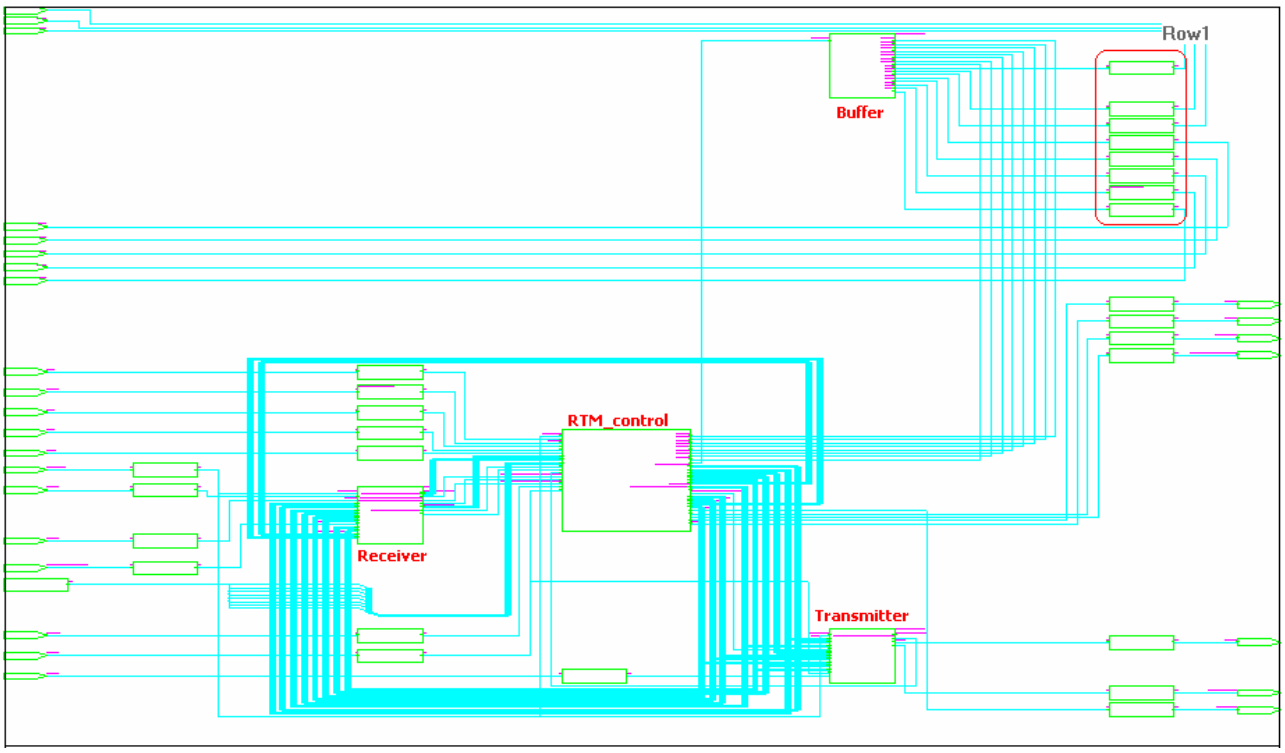
```
                D0 <= DC0;
                D1 <= DC1;
                D2 <= DC2;
                D3 <= DC3;
                D4 <= DC4;
                D5 <= DC5;
                D6 <= DC6;
                D7 <= DC7;

        end if;

        if Cbuff='0' then                       -- dozvola upisa sa magistrale

                if D0'event then
                        DC0<= D0;
                end if;
                if D1'event then
                        DC1<= D1;
                end if;
                if D2'event then
                        DC2<= D2;
                end if;
                if D3'event then
                        DC3<= D3;
                end if;
                if D4'event then
                        DC4<= D4;
                end if;
                if D5'event then
                        DC5<= D5;
                end if;
                if D6'event then
                        DC6<= D6;
                end if;
                if D7'event then
                        DC7<= D7;
                end if;

        end if;

end process;

end architecture DataBUFF;
```

4. SINTEZA I IMPLEMENTACIJA KOLA

Čip 8251 je sintetizovan i implementiran korišćenjem programskog paketa XILINX ISE6.3. Pomenuti programski paket služi za razvoj aplikacija baziranih na njihovim CPLD i FPGA kolima.

Za implementaciju UART čipa Intel 8251 iskorišćeno je FPGA kolo iz Xilinx-ove familije SPARTAN2 koje nosi oznaku XC4S30-VQ100. Radi se o «pakovanju» koje sadrži 30K gejtova, a napaja se sa 5V.

Na slici 4.1 prikazana je šema sinteze kola na najvišem nivou, koju je generisao program Xilinx FPGA Express. Na šemi su prikazani osnovni blokovi čipa, veze među njima, kao i baferi (ulazni i izlazni). Svakom bloku može da pristupi «dubinski», do nivoa samih gejtova.



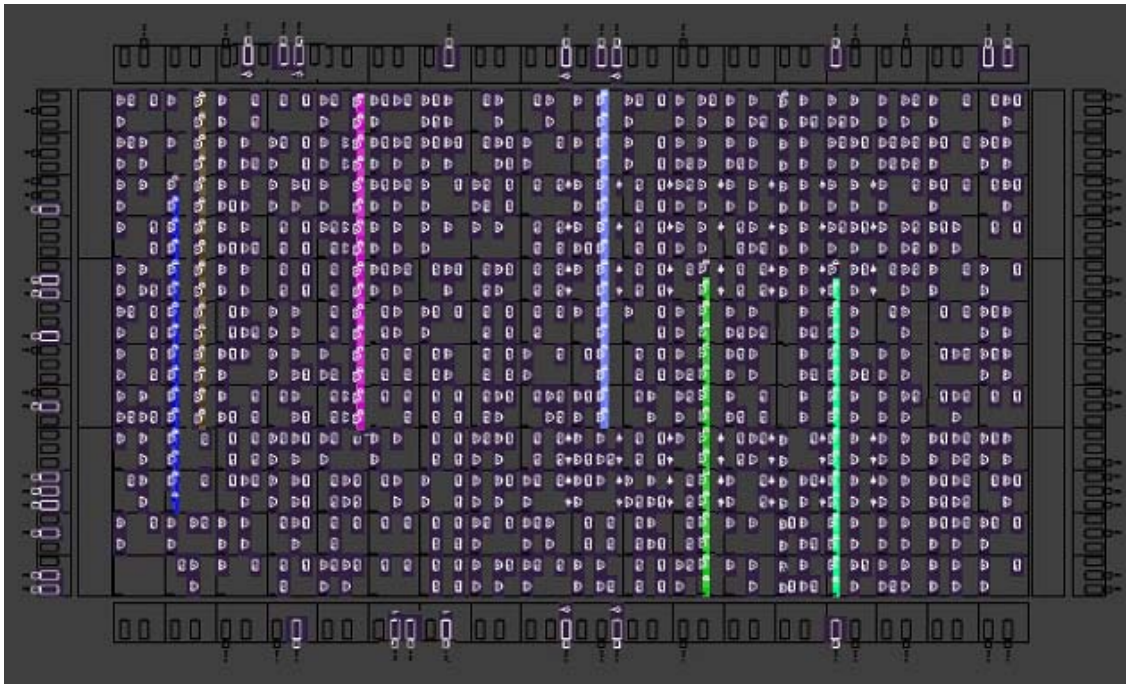
Slika 4.1: Šema sinteze

Naredni listing pokazuje rezultate implementacije:

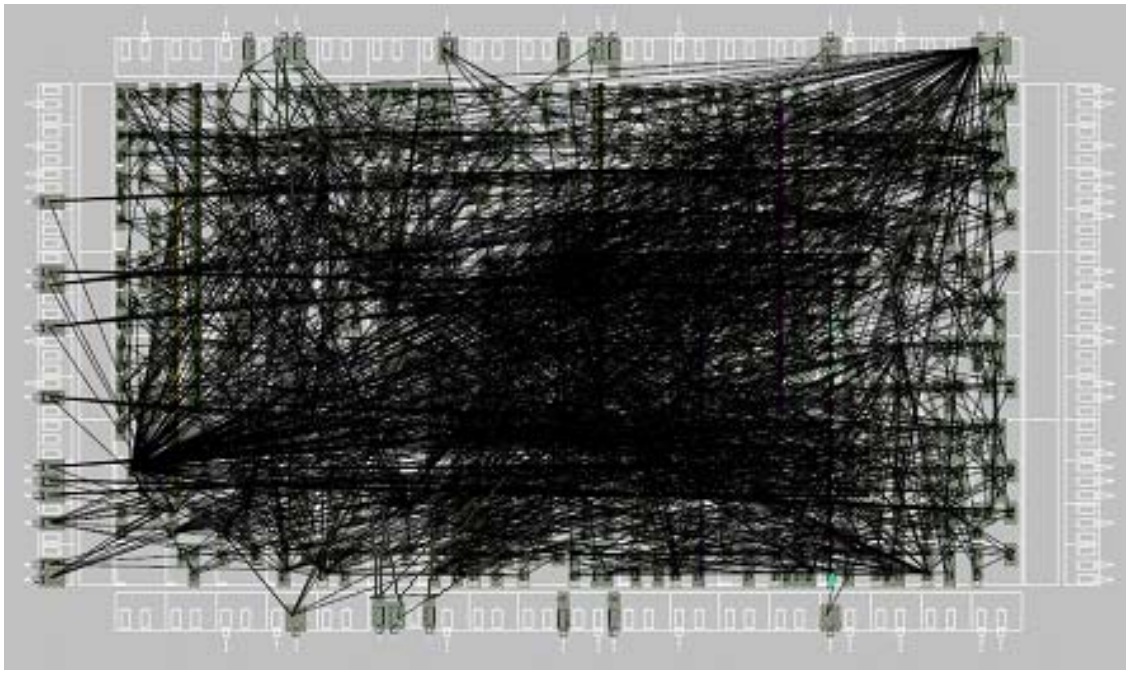
```
Design Summary
--------------
Number of errors:      0
Number of warnings:   37
Logic Utilization:
  Total Number Slice Registers:     409 out of    864   47%
    Number used as Flip Flops:              138
    Number used as Latches:                 215
  Number of 4 input LUTs:           554 out of    864   64%
Logic Distribution:
    Number of occupied Slices:                  418 out of    432   97%
    Number of Slices containing only related logic:    376 out of    430  87%
    Number of Slices containing unrelated logic:        42 out of    430   10%
       *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          516 out of    864   60%
      Number used as logic:                 523
      Number used as a route-thru:            4
   Number of bonded IOBs:           20 out of     60   33%
      IOB Latches:                           3
   Number of Tbufs:                56 out of    480   11%
   Number of GCLKs:                 4 out of      4  100%
   Number of GCLKIOBs:              4 out of      4  100%

Total equivalent gate count for design:  5,981
Additional JTAG gate count for IOBs:  994
```

Dakle, implemetnirano kolo zauzelo 5981 ekvivalentnih gejtova, kao i da je pridodato još 994 gejtova, neophodnih za realizaciju JTAG logike - preko koje se čip programira i testira. Maksimalna radna frekvencija čipa je 167.24 MHz.

Na slikama 4.2 i 4.3 vidimo unutrašnju strukturu dobijenu nakon implementacije kola gde se vide zauzeti blokovi unutar kola. Na slici 4.2 prikazan je razmeštaj ćelija gde možemo videti i koji pinovi kola su iskorišćeni. Iako se zapaža da izvodi kola nisu grupisani, to je da bi se postiglo približno isto kašnjenje veza do svih pinova. Izgled povezanosti samih veza prikazan je na slici 4.3



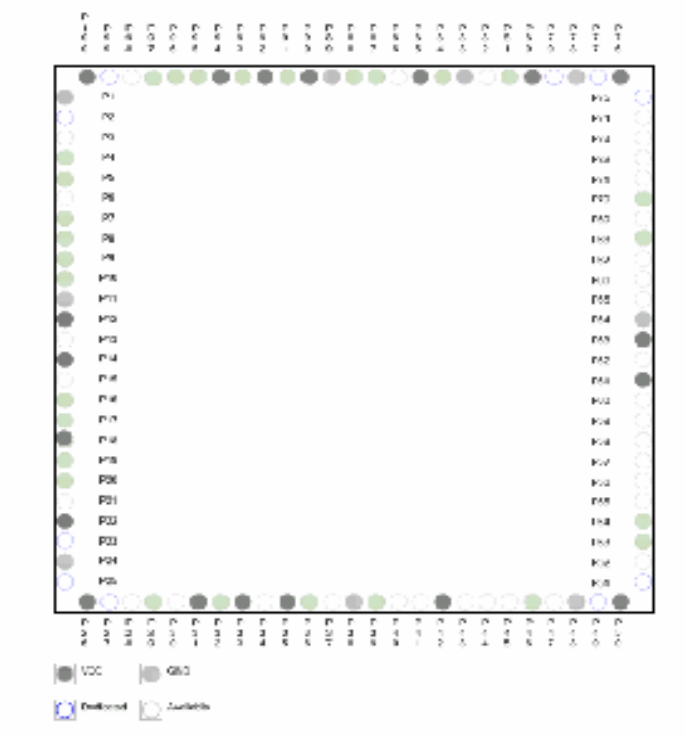
Slika 4.2: Šema zauzetoti ćelija FPGA kola



Slika 4.3: Šema veza ćelija FPGA kola

Analizirana je potrošnja čipa (primenom programa XPower) i ona iznosi 10.1mW. Kada je temperatura ambijenta 25 °C, temperatura kućišta iznosi 25,5 °C.

Na slici 4.4 prikazan je i sam fizički raspored pinova na FPGA kolu.



Slika 4.4: Raspored pinova na FPGA čipu

Ovim je završen ceo proces sinteze i implementacije kola. Program generiše i programski fajl kojim je potrebno isprogramirati FPGA kolo da bi se dobila željena funkcija.

5. TESTIRANJE RADA KOLA

PRIMER 1:

Programirati čip 8251 tako vrši prenos osmobitnog karaktera «00001111» (D7-D0) brzinom 1x, sa jednim start-bitom i bez bita parnosti.

Rešenje u VHDL-u

Sledeći Test_Bench procesira zadatak iz *Primera 1*

```
library ieee;
use ieee.std_logic_1164.all;

entity Test_1 is
end Test_1;

architecture TA of Test_1 is

        -- Deklarisanje cipa 8251 kao komponente
        component A8251
                PORT(

                        D0          : inout std_logic;
                        D1          : inout std_logic;
                        D2          : inout std_logic;
                        D3          : inout std_logic;
                        D4          : inout std_logic;
                        D5          : inout  std_logic;
                        D6          : inout std_logic;
                        D7          : inout std_logic;
                        Reset       : in std_logic;
                        Cs          : in std_logic;
                        C_D         : in std_logic;
                        Rd          : in std_logic;
                        Wr          : in std_logic;
                        Clk         : in std_logic;
                        RxC         : in std_logic;
                        TxC         : in std_logic;
                        RxD         : in std_logic;
                        TxD         : out std_logic;
                        TxEMPTY:out std_logic;
                        TxRDY    : out std_logic;
                        RxRDY      : out std_logic;
                        Syndet_BD: inout std_logic;
                        DSR    : in std_logic;
```

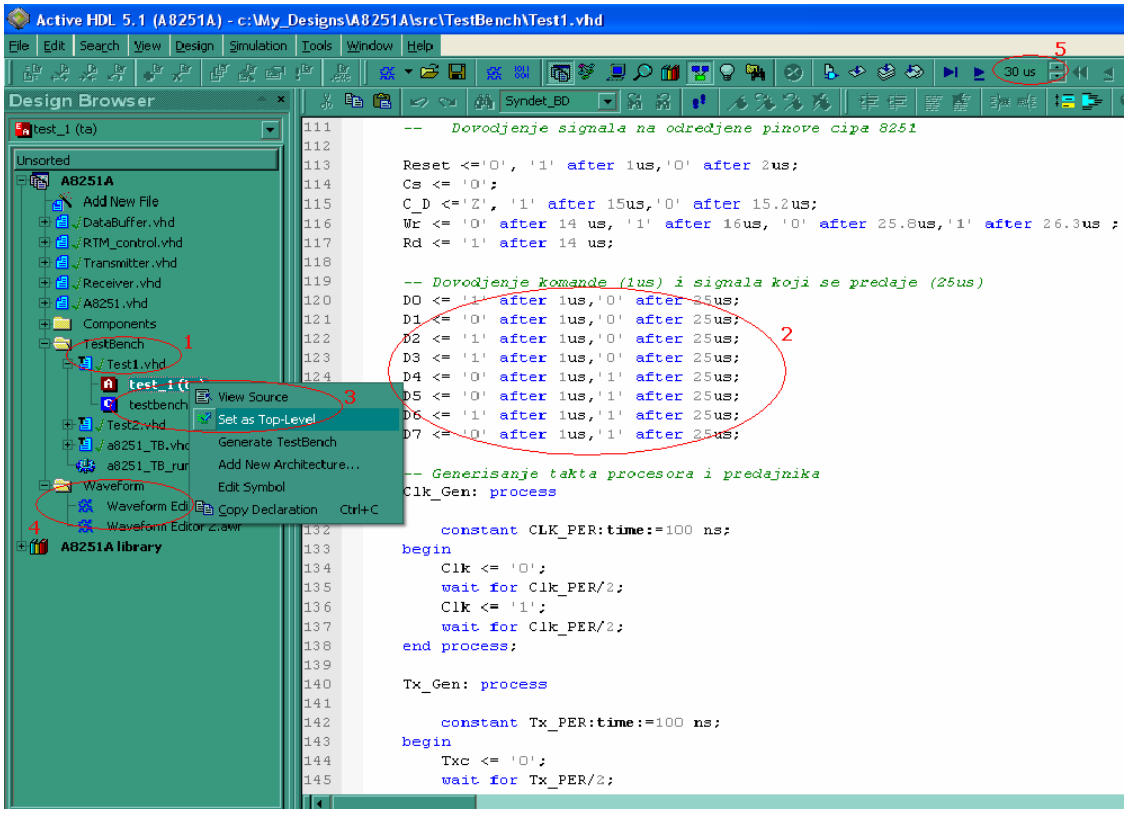
```
                CTS   : in std_logic;
                DTR   : out std_logic;
                RTS   : out std_logic
                              );

        END component;


        -- Deklaracija signala koji su ujedno ulazi i/ili izlazi čipa

        Signal D0          :  std_logic;
        Signal D1          :  std_logic;
        Signal D2          :  std_logic;
        Signal D3          :  std_logic;
        Signal D4          :  std_logic;
        Signal D5          :  std_logic;
        Signal D6          :  std_logic;
        Signal D7          :  std_logic;
        Signal Reset       :  std_logic;
        Signal Cs          :  std_logic;
        Signal C_D         :  std_logic;
        Signal Rd          :  std_logic;
        Signal Wr          :  std_logic;
        Signal Clk         :  std_logic;
        Signal RxC         :  std_logic;
        Signal TxC         :  std_logic;
        Signal RxD         :  std_logic;
        Signal TxD         :  std_logic;
        Signal TxEMPTY: std_logic;
        Signal TxRDY    : std_logic;
        Signal RxRDY    : std_logic;
        Signal Syndet_BD: std_logic;
        Signal DSR         :  std_logic;
        Signal CTS         :  std_logic;
        Signal DTR         :  std_logic;
        Signal RTS         :  std_logic;

begin

        -- Preslikavanje signala

        UUT : A8251
        port map (

                D0 => D0,
                D1 => D1,
                D2 => D2,
                D3 => D3,
                D4 => D4,
                D5 => D5,
```

```
            D6 => D6,
            D7 => D7,
            Reset => Reset,
            Cs => Cs,
            C_D => C_D,
            Rd  => Rd,
            Wr  => Wr,
            Clk => Clk,
            RxC => RxC,
            TxC => TxC,
            RxD => RxD,
            TxD => TxD,
            TxEMPTY => TxEMPTY,
            TxRDY => TxRDY,
            RxRDY => RxRDY,
            Syndet_BD => Syndet_BD,
            DSR => DSR,
            CTS => CTS,
            DTR => DTR,
            RTS => RTS

            );

-- Dovođenje signala na određene pinove čipa 8251

Reset <='0', '1' after 1us,'0' after 2us;
Cs <= '0';
C_D <='Z', '1' after 15us,'0' after 15.2us;
Wr <= '0' after 14 us, '1' after 16us, '0' after 25.8us,'1' after 26.3us ;
Rd <= '1' after 14 us;

-- Dovođenje komande (1us) i signala koji se predaje (25us)

D7 <= '1' after 1us,'0' after 25us;
D6 <= '0' after 1us,'0' after 25us;
D5 <= '1' after 1us,'0' after 25us;
D4 <= '1' after 1us,'0' after 25us;
D3 <= '0' after 1us,'1' after 25us;
D2 <= '0' after 1us,'1' after 25us;
D1 <= '1' after 1us,'1' after 25us;
D0 <= '0' after 1us,'1' after 25us;

-- Generisanje takta procesora i predajnika
Clk_Gen: process

            constant CLK_PER:time:=100 ns;
begin
            Clk <= '0';
            wait for Clk_PER/2;
            Clk <= '1';
            wait for Clk_PER/2;
```

```
	end process;

	Tx_Gen: process

		constant Tx_PER:time:=100 ns;
	begin
		Txc <= '0';
		wait for Tx_PER/2;
		Txc <= '1';
		wait for Tx_PER/2;
	end process;



end architecture TA;


configuration TESTBENCH_FOR_A8251 of Test_1 is
	for TA
		for UUT : A8251
			use entity work.A8251(A8251);
		end for;
	end for;
end TESTBENCH_FOR_A8251;
```

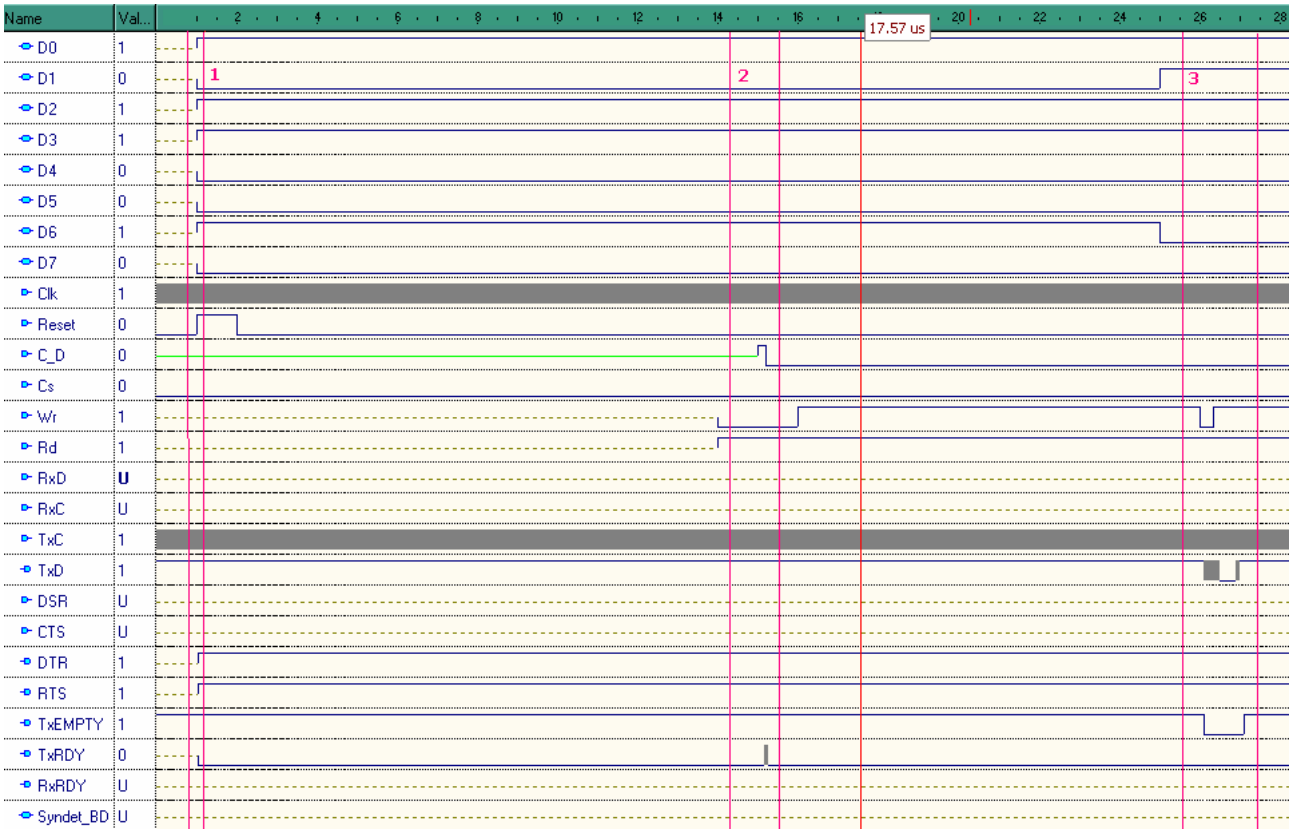
Testiranje rada čipa UART **8251** za **Primer 1** obavlja se na sledeći način:

1. Pokrenuti program ***Active-HDL*** čija se ikonica nalazi na *Desktop*-u
2. Otvoriti dizajn **UART 8251**
3. Otvoriti listing *TestBench*-a **Test_1**, koji se nalazi u *folder*-u *TestBench* **(1.** segment na slici 5.1)
4. U okviru tekstualnog dela *Test_Bench*-a upisati komandu i bitsku sekvencu koja je zadata vašoj grupi, umesto postojeće (2. segment na slici 5.1)
5. Izvršiti kompilaciju (kompajliranje) pritiskom na taster <F11>
6. Podesiti *Top level* entitet desnim klikom na ″+¨ ispred imena fajla **Test_1.vhd** i izborom opcije *Set as Top-Level* (3. segment na slici 5.1)



Slika 5.1: Testiranje projekta **UART 8251 – Primer 1**

7. Otvoriti odgovarajući *Waveform* fajl koji se nalazi u folderu *Waveforms* (4. segment na slici 5.1)
8. Podesiti trajanje simulacije na 30 μs (5. segment na slici 5.1)
9. Pokrenuti simulaciju pritiskom na taster <F5>
10. Ukoliko je potrebno, uvećati (taster <+>) ili umanjiti (taster <->) dijagram, radi bolje preglednosti
11. Odštampati dobijeni vremenski dijagram za **Primer 1**



Slika 5.2: Dijagram promene stanja signala čipa 8251 - **Primer 1**

Pre nego što počne bilo kakva operacija, CPU (PROCESOR) resetuje čip 8251 (**1**).

Sledeći korak je upisivanje komande (**2**), koju CPU (PROCESOR) šalje preko magistrale podataka.

U trećoj fazi CPU (PROCESOR) preko magistrale podataka šalje karakter, čip 8251 formira podatak (start-karakter-parnost-stop) koji se prosleđuje preko pina TxD i to bit-po-bit (**3**).

PRIMER 2:

Programirati čip 8251 tako vrši prijem osmobitnog karaktera «11110000» (D7-D0) brzinom 1x, sa jednim start-bitom i bez bita parnosti.

Rešenje u VHDL-u

Sledeći Test_Bench procesira zadatak iz *Primera 2*

```
library ieee;
use ieee.std_logic_1164.all;

entity Test_2 is
end Test_2;

architecture TA of Test_2 is

        -- Deklarisanje cipa 8251 kao komponente

        component A8251
                PORT(

                        D0          : inout std_logic;
                        D1          : inout std_logic;
                        D2          : inout std_logic;
                        D3          : inout std_logic;
                        D4          : inout std_logic;
                        D5          : inout  std_logic;
                        D6          : inout std_logic;
                        D7          : inout std_logic;
                        Reset       : in std_logic;
                        Cs          : in std_logic;
                        C_D         : in std_logic;
                        Rd          : in std_logic;
                        Wr          : in std_logic;
                        Clk         : in std_logic;
                        RxC         : in std_logic;
                        TxC         : in std_logic;
                        RxD         : in std_logic;
                        TxD         : out std_logic;
                        TxEMPTY:out std_logic;
                        TxRDY    : out std_logic;
                        RxRDY    : out std_logic;
                        Syndet_BD: inout std_logic;
                        DSR         : in std_logic;
                        CTS         : in std_logic;
```

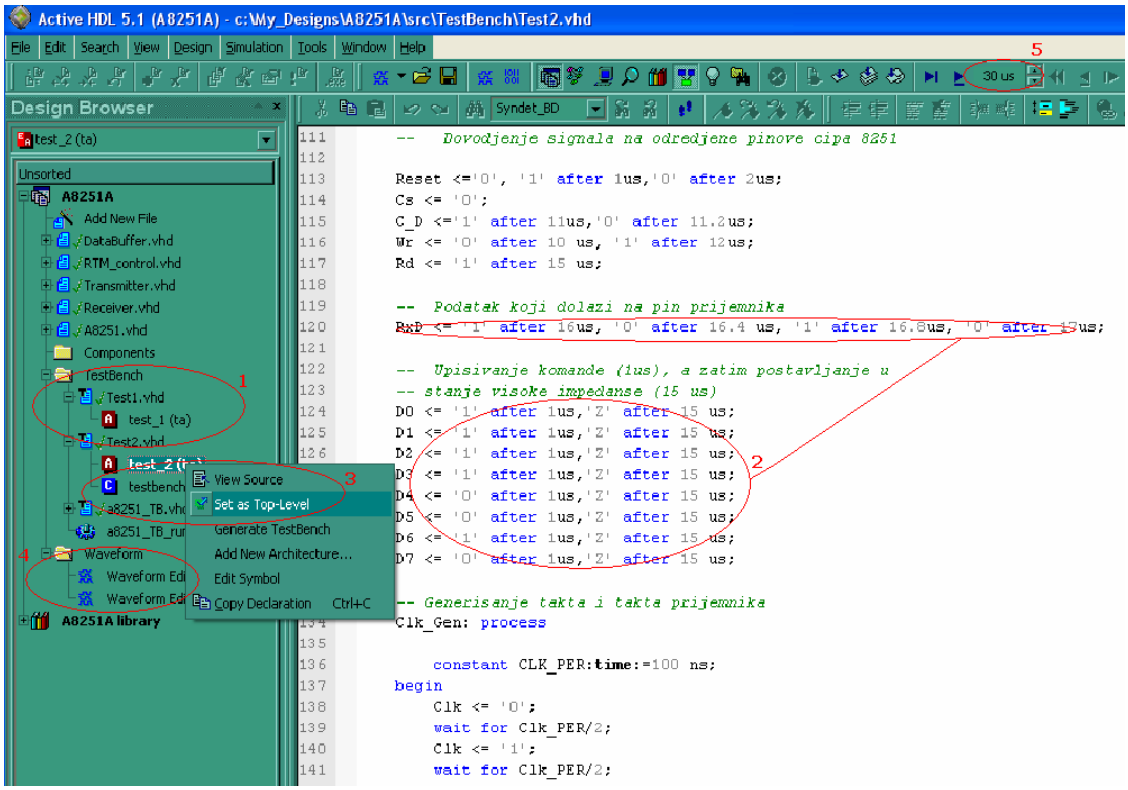
```
                DTR   : out std_logic;
                RTS   : out std_logic
                );

        END component;

        -- Deklaracija signala koji su ujedno ulazi i/ili izlazi čipa

        Signal D0          :  std_logic;
        Signal D1          :  std_logic;
        Signal D2          :  std_logic;
        Signal D3          :  std_logic;
        Signal D4          :  std_logic;
        Signal D5          :  std_logic;
        Signal D6          :  std_logic;
        Signal D7          :  std_logic;
        Signal Reset       :  std_logic;
        Signal Cs          :  std_logic;
        Signal C_D         :  std_logic;
        Signal Rd          :  std_logic;
        Signal Wr          :  std_logic;
        Signal Clk         :  std_logic;
        Signal RxC         :  std_logic;
        Signal TxC         :  std_logic;
        Signal RxD         :  std_logic;
        Signal TxD         :  std_logic;
        Signal TxEMPTY: std_logic;
        Signal TxRDY     : std_logic;
        Signal RxRDY     : std_logic;
        Signal Syndet_BD: std_logic;
        Signal DSR         :  std_logic;
        Signal CTS         :  std_logic;
        Signal DTR         :  std_logic;
        Signal RTS         :  std_logic;

begin

        -- Preslikavanje signala

        UUT : A8251
        port map (

                D0 => D0,
                D1 => D1,
                D2 => D2,
                D3 => D3,
                D4 => D4,
                D5 => D5,
                D6 => D6,
                D7 => D7,
                Reset => Reset,
```

```
        Cs => Cs,
        C_D => C_D,
        Rd  => Rd,
        Wr  => Wr,
        Clk => Clk,
        RxC => RxC,
        TxC => TxC,
        RxD => RxD,
        TxD => TxD,
        TxEMPTY => TxEMPTY,
        TxRDY => TxRDY,
        RxRDY => RxRDY,
        Syndet_BD => Syndet_BD,
        DSR => DSR,
        CTS => CTS,
        DTR => DTR,
        RTS => RTS

        );

-- Dovođenje signala na određene pinove cipa 8251

Reset <='0', '1' after 1us,'0' after 2us;
Cs <= '0';
C_D <='1' after 11us,'0' after 11.2us;
Wr <= '0' after 10 us, '1' after 12us;
Rd <= '1' after 15 us;

--  Podatak koji dolazi na pin predajnika
RxD <= '1' after 16us, '0' after 16.4 us, '1' after 16.8us, '0' after 17us;

--  Upisivanje komande (1us), a zatim postavljanje u
-- stanje visoke impedanse (15 us)

D0 <= '1' after 1us,'Z' after 15 us;
D1 <= '1' after 1us,'Z' after 15 us;
D2 <= '1' after 1us,'Z' after 15 us;
D3 <= '1' after 1us,'Z' after 15 us;
D4 <= '0' after 1us,'Z' after 15 us;
D5 <= '0' after 1us,'Z' after 15 us;
D6 <= '1' after 1us,'Z' after 15 us;
D7 <= '0' after 1us,'Z' after 15 us;

-- Generisanje takta procesora i takta prijemnika
Clk_Gen: process

        constant CLK_PER:time:=100 ns;
begin
        Clk <= '0';
        wait for Clk_PER/2;
        Clk <= '1';
```

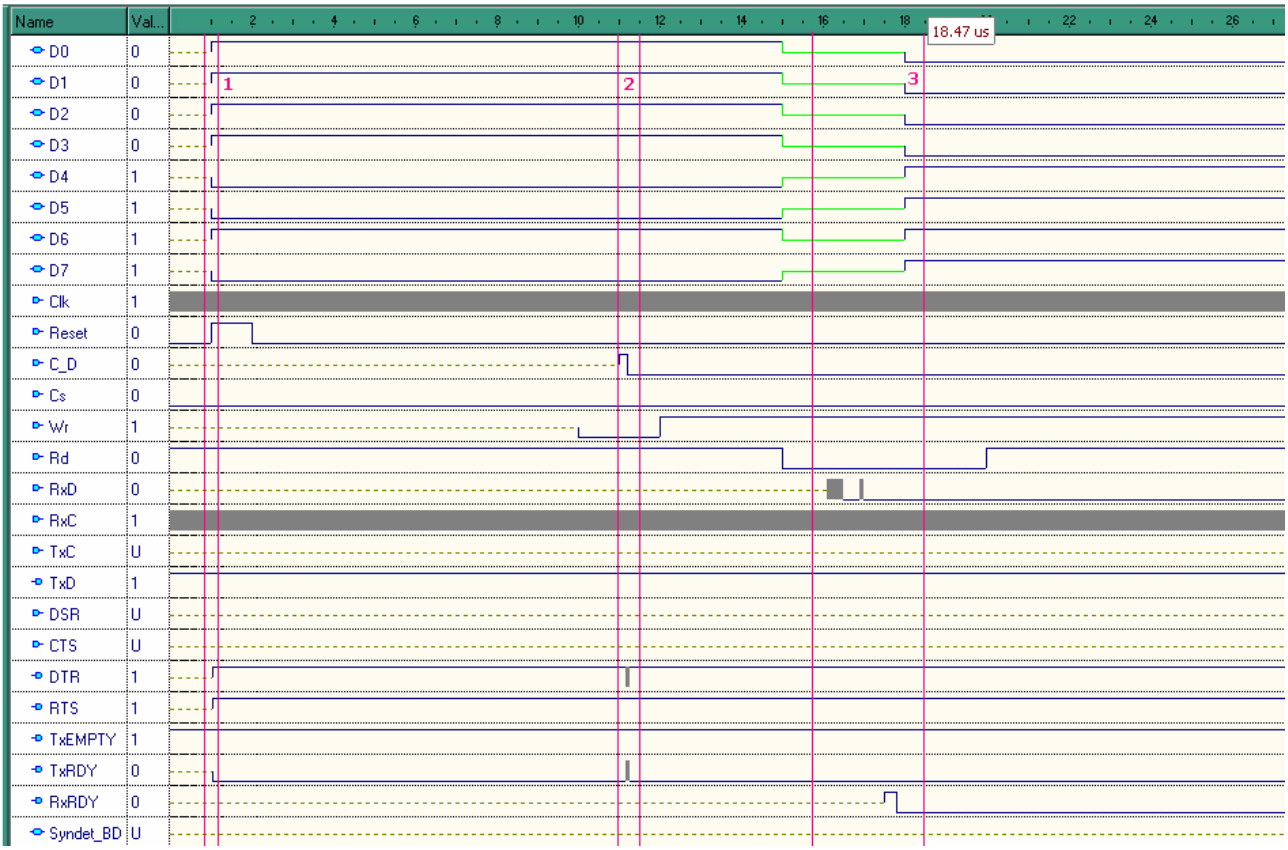
```
		wait for Clk_PER/2;
	end process;



	Rx_Gen: process

		constant Rx_PER:time:=100 ns;
	begin
		Rxc <= '0';
		wait for Rx_PER/2;
		Rxc <= '1';
		wait for Rx_PER/2;
	end process;


end architecture TA;


configuration TESTBENCH_FOR_A8251 of Test_2 is
	for TA
		for UUT : A8251
			use entity work.A8251(A8251);
		end for;
	end for;
end TESTBENCH_FOR_A8251;
```

Testiranje rada čipa UART **8251** za **Primer 2** obavlja se na sledeći način:

1. Pokrenuti program ***Active-HDL*** čija se ikonica nalazi na *Desktop*-u
2. Otvoriti dizajn **UART 8251**
3. Otvoriti listing *TestBench*-a **Test_2**, koji se nalazi u *folder*-u *TestBench* **(2.** segment na slici 5.3)
4. U okviru tekstualnog dela *Test_Bench*-a upisati komandu i bitsku sekvencu koja je zadata vašoj grupi, umesto postojeće (2. segment na slici 5.3)
5. Izvršiti kompilaciju (kompajliranje) pritiskom na taster <F11>
6. Podesiti *Top level* entitet desnim klikom na ″+¨ ispred imena fajla **Test_1.vhd** i izborom opcije *Set as Top-Level* (3. segment na slici 5.3)
7. Otvoriti odgovarajući *Waveform* fajl koji se nalazi u folderu *Waveforms* (4. segment na slici 5.3)
8. Podesiti trajanje simulacije na 30 μs (5. segment na slici 5.3)
9. Pokrenuti simulaciju pritiskom na taster <F5>
10. Ukoliko je potrebno, uvećati (taster <+>) ili umanjiti (taster <->) dijagram, radi bolje preglednosti
11. Odštampati dobijeni vremenski dijagram za **Primer 2**



Slika 5.3. Testiranje projekta **UART 8251 – Primer 2**



Slika 5.4: Dijagram promene stanja signala čipa 8251 - **Primer 2**

CPU (PROCESOR) najpre resetuje čip 8251 (**1**).

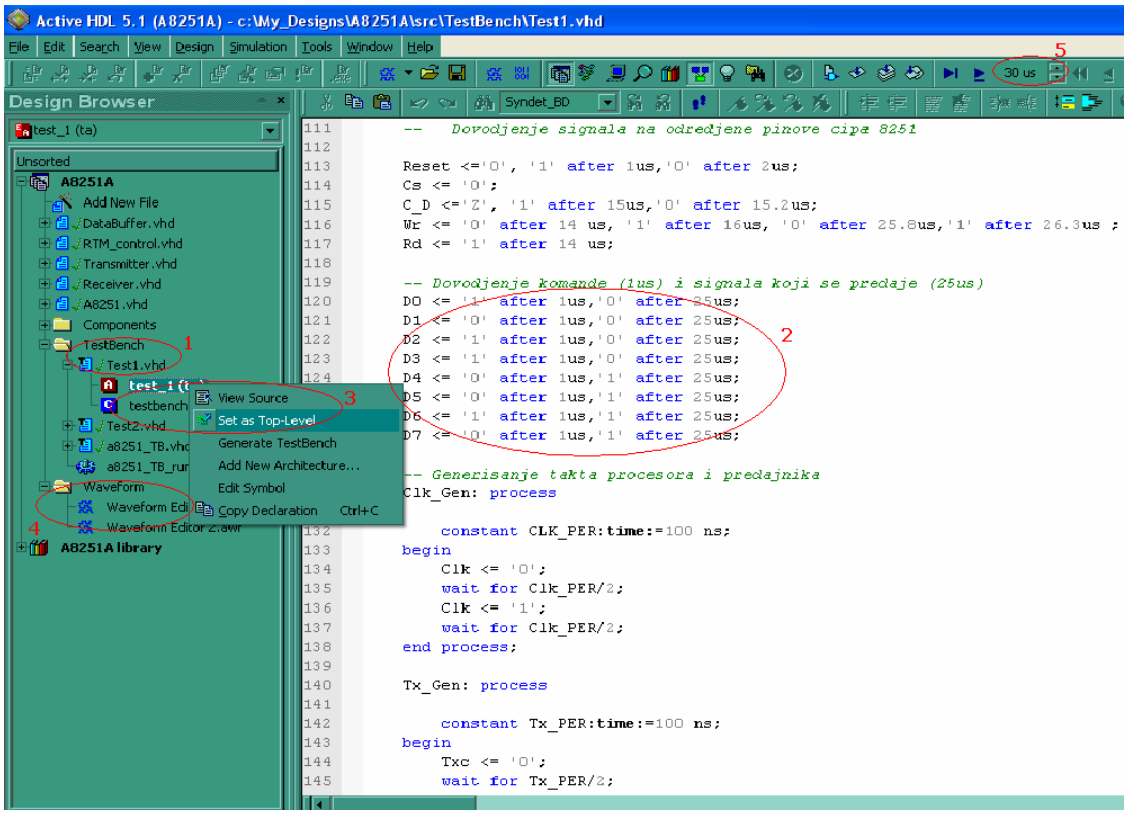
Sledeću korak je upisivanje komande (**2**), koju CPU (PROCESOR) šalje preko magistrale podataka.

Tokom treće faze, podatak se prihvata preko pina RxD, a zatim CPU (PROCESOR) preko magistrale podataka preuzima karakter (**3**).

6. ZADATAK

Proveriti ispravnost rada UART 8251 čipa na sledeći način:

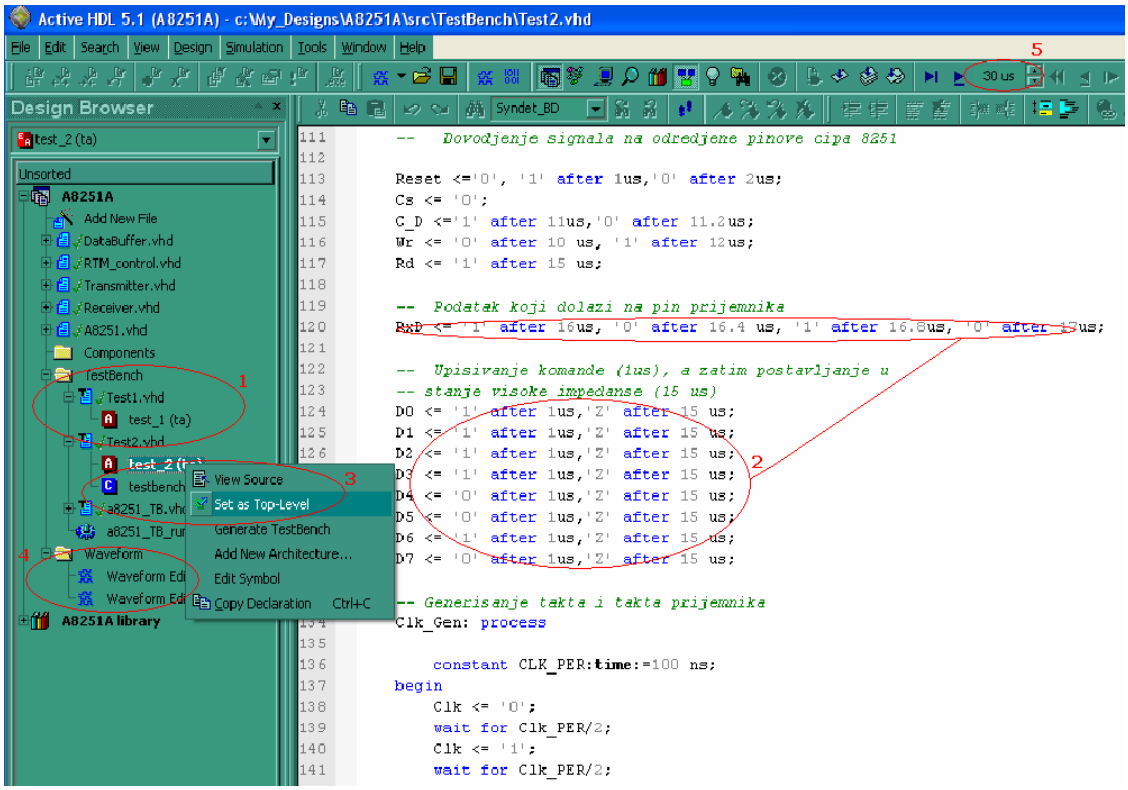
1. **Pokrenuti program *Active-HDL* čija se ikonica nalazi na *Desktop*-u**
2. **Otvoriti dizajn UART 8251**
3. **Otvoriti listing *TestBench*-a Test_1 (za Primer 1) ili Test_2 (za Primer 2), koji se nalazi u *folder*-u *TestBench* (1. segment na slikama 6.1 i 6.2)**
4. **U okviru tekstualnog dela *Test_Bench*-a upisati komandu i bitsku sekvencu koja je navedena u zadatku, umesto postojeće 2. segment na slikama 6.1 i 6.2)**
5. **Izvršiti kompilaciju (kompajliranje) pritiskom na taster <F11>**
6. **Podesiti *Top level* entitet desnim klikom na ″+″ ispred imena fajla Test_1.vhd (odnosno Test_2.vhd za drugi primer) i izborom opcije *Set as Top-Level* (3. segment na slikama 6.1 i 6.2)**



*Slika 6.1. Startovanje projekta UART 8251 – **Primer 1***

7. **Otvoriti odgovarajući *Waveform* fajl koji se nalazi u folderu *Waveforms* (4. segment na slikama 6.1 i 6.2)**

8. **Podesiti trajanje simulacije na 30 μs (5. segment na slikama 6.1 i 6.2)**
9. **Pokrenuti simulaciju pritiskom na taster <F5>**
10. **Ukoliko je potrebno, uvećati (taster <+>) ili umanjiti (taster <->) dijagram, radi bolje preglednosti**
11. **Odštampati dobijeni vremenski dijagram za primer x**



Slika 6.2. Startovanje projekta UART 8251 – Primer 2

Primer 1:

Grupa 1:

Programirati čip 8251 tako vrši slanje osmobitnog karaktera «10000001» (D7-D0) brzinom 16x, sa jednim start-bitom i bez bita parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 2:

Programirati čip 8251 tako vrši slanje osmobitnog karaktera «11000011» (D7-D0) brzinom 1x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 3:

Programirati čip 8251 tako vrši slanje osmobitnog karaktera «00110000» (D7-D0) brzinom 64x, sa jednim start-bitom i sa neparnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 4:

Programirati čip 8251 tako vrši slanje osmobitnog karaktera «10100001» (D7-D0) brzinom 16x, sa dva start-bita i bez bita parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 5:

Programirati čip 8251 tako vrši slanje osmobitnog karaktera «00000000» (D7-D0) brzinom 1x, sa jednim start-bitom i sa neparnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 6:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «11111111» (D7-D0) brzinom 64x, sa jednim i po start-bitom i bez bita parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 7:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «11000111» (D7-D0) brzinom 1x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 8:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «11000111» (D7-D0) brzinom 1x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 9:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «10001111» (D7-D0) brzinom 16x, sa jednim start-bitom i bez bita parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 10:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «11000111» (D7-D0) brzinom 1x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 11:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «00110110» (D7-D0) brzinom 64x, sa jednim start-bitom i sa neparnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 12:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «10101101» (D7-D0) brzinom 16x, sa dva start-bita i bez bita parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 13:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «00000010» (D7-D0) brzinom 1x, sa jednim start-bitom i sa neparnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 14:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «11111011» (D7-D0) brzinom 64x, sa jednim i po start-bitom i bez bita parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 15:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «11000101» (D7-D0) brzinom 1x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Grupa 16:
Programirati čip 8251 tako vrši slanje osmobitnog karaktera «11000110» (D7-D0) brzinom 1x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme slanja: _____

Primer 2:

Grupa 1:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «10111101» (D7-D0) brzinom 64x, sa jednim start-bitom i bez bita parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 2:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «11000010» (D7-D0) brzinom 16x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 3:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «00110000» (D7-D0) brzinom 1x, sa jednim start-bitom i i sa neparnim bitom parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 4:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «10101101» (D7-D0) brzinom 16x, sa dva start-bita i bez bita parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 5:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «00011000» (D7-D0) brzinom 64x, sa jednim start-bitom i sa neparnim bitom parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 6:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «11111111» (D7-D0) brzinom 1x, sa jednim i po start-bitom i bez bita parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 7:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «11000111» (D7-D0) brzinom 16x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 8:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «11110011» (D7-D0) brzinom 1x, sa jednim i po start-bitom i bez bita parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 9:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «10111110» (D7-D0) brzinom 64x, sa jednim start-bitom i bez bita parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 10:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «11000001» (D7-D0) brzinom 16x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 11:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «00110110» (D7-D0) brzinom 1x, sa jednim start-bitom i i sa neparnim bitom parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 12:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «10100001» (D7-D0) brzinom 16x, sa dva start-bita i bez bita parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 13:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «00010000» (D7-D0) brzinom 64x, sa jednim start-bitom i sa neparnim bitom parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 14:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «11111001» (D7-D0) brzinom 1x, sa jednim i po start-bitom i bez bita parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 15:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «11001111» (D7-D0) brzinom 16x, sa jednim start-bitom i sa parnim bitom parnosti.

Komanda: _____
Vreme prijema: _____

Grupa 16:
Programirati čip 8251 tako vrši prijem osmobitnog karaktera «11111011» (D7-D0) brzinom 1x, sa jednim i po start-bitom i bez bita parnosti.

Komanda: _____
Vreme prijema: _____

7. LITERATURA :

1. INTEL 8251 Programmable communication Interface, 1986.

2. Mile K. Stojčev, Branislav D. Petrović ARHITEKTURE I PROGRAMIRANJE μRAČUNARSKIH SISTEMA ZASNOVANIH NA FAMILIJI PROCESORA 80x86 I izdanje, Elektronski fakultet, Niš, 1999.