VHDL implementacija UART-a sa LIFO memorijom

Andrea Ćirić 2016/202 OE

4. januar 2021

Mentori: *doc. dr Jelena Popović Božović, as. ms Vladimir Petrović*

Sadržaj

Apstrakt

Kod paralelne komunikacije cena, kao i kompleksnost sistema rastu usled istovremenog prenosa bitova preko većeg broja žica. Serijska komunikacija ublažava ovaj nedostatak i postaje efikasan kandidat u mnogim aplikacijama za daljinsku komunikaciju jer smanjuje izobličenje signala zbog svoje jednostavne strukture. Ovaj rad se fokusira na VHDL implementaciji UART-a sa LIFO memorijom koja obezbeđuje ispisivanje unetih karaktera u suprotnom smeru. Podaci se unose do trenutka kada je uneti karakter Enter tj. Carriage Return. UART LIFO u ovom radu podrazumeva da je veličina podatka koji se prenosi veličine ASCII koda karaktera odnosno 8 bitova i može raditi na osam različitih bitskih brzina.

1 Uvod

Universal Asynchronous Receiver Transmitter (UART) je tip serijskog protokola koji ima veliku primenu u prenosu podataka kratkog rastojanja, malih brzina i niske cene između računara i periferija. UART funkcioniše na taj način što pretvara paralelnu komunikaciju u serijsku i obratno.

UART implementiran VHDL jezikom može biti integrisan na FPGA u cilju postizanja kompaktnog, stabilnog i poverljivog prenosa podataka. U različitim sistemima postoje različite primene UART-a te i njegove različite karakteristike [3].

U ovom radu konstruisan je UART sa LIFO memorijom koji prenosi upisane karaktere do trenutka kada je upisan karakter Enter odnosno Carriage Return. Tada unete karaktere ispisuje u obrnutom redosledu.

Kod je pisan za spuštanje na Altera DE1-SoC Cyclone V FPGA sa ARM Cortex-A9 procesorom. Razvojno okruženje u kom je kucan VHDL kod je Intelov Quartus Prime Lite Edition, a simulacije su odrađene u Intelovom ModelSim-u.

U ostatku rada u sekciji 2 opisana je ukratko istorija VHDL jezika, u sekcijama 3-6 predstavljene su kreirane komponente i metod rada, dok je zaključak sa metodama poboljšanja napisan u sekciji 7.

2 VHDL

VHDL je primer jezika za opis hardvera (*Hardware Design Language - HDL*). Ovi jezici se obično zasnivaju na modelu diskretnih događaja tj. na simuliranju generisanih događaja i njihovoj obradi tokom vremena.

VHDL vodi poreklo iz 1980-ih. U to vreme većina sistema za dizajn kola koristila je grafičke jezike za opis hardvera. Zbog potrebe za pravljenjem kompleksnijih sistema nastali su tekstualni HDL-ovi pomoću kojih se lakše predstavljaju složeni proračuni zbog primene promenljivih, petlji, funkcija i rekurzije. Tekstualni HDL-ovi su u početku bili uglavnom samo tema istraživanja na univerzitetima, međutim predstavljanjem VHDL-a i njegovog konkurenta Verilog, njihova primena je značajno porasla [2].

VHDL je dizajniran u sklopu VHSIC programa Ministarstva odbrane u Sjedinjenim Američkim Državama. VHSIC predstavlja *very high-speed integrated circuits* tj. integrisana kola izuzetno velikih brzina. U početku su na dizajnu VHDL-a radile tri kompanije: *IBM, Intermetrics* i *Texas Instruments*.

VHDL koristi *procese* za modeliranje paralelnih događaja. Svaki proces modelira jednu komponentu potencijalno istovremenog hardvera. Za jednostavne hardverske komponente jedan događaj može biti dovoljan, dok je za složenije potreban veći broj procesa. Procesi komuniciraju putem signala, koji uglavnom odgovaraju fizičkim vezama (žicama).

3 Universal asynchronous receiver-transmitter - UART

UART predstavlja modul koji se koristi za asinhronu serijsku komunikaciju. Sastoji se iz dva bitna dela: prijemnika i predajnika. Predajnik (*transmitter*) prima osmobitni paralelni podatak i šalje ga na serijski izlaz, dok prijemnik prima podatak bit po bit sa serijskog ulaza i šalje ga na paralelni osmobitni izlaz. Ova dva procesa biće detaljnije objašnjena u nastavku teksta.

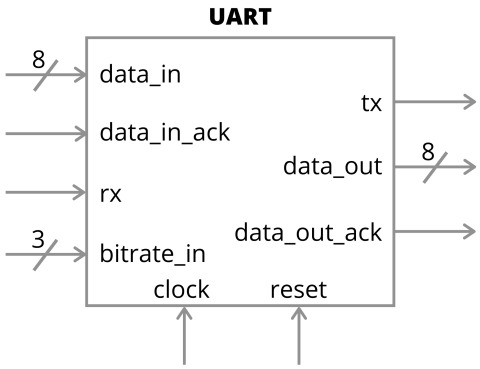
Od ulaznih signala koji se odnose na celu komponentu, UART prima taktni signal, signal reseta i vrednost *bitrate* predstavljenu 3-bitnim signalom *bitrate_in* (Slika 1.). Vrednosti bitrate generisane na osnovu signala *bitrate_in* prikazane su u tabeli 1.

Kada je u pitanju predajnik postoje dva ulazna signala, *data_in* koji predstavlja osmobitni ulazni podatak koji treba preneti na serijski izlaz i *data_in_ack* koji nosi informaciju o tome kada je signal *data_in* upisan i spreman za prenos. Izlazni signal *tx* predstavlja serijski interfejs na koji se šalju podaci sa ulaza.

Kada je u pitanju prijemnik, ulazni signal *rx* predstavlja serijski interfejs sa kojeg se očitavaju podaci i postavljaju na 8-bitni izlaz *data_out*. Osim tog postoji i signal *data_out_ack* koji nam govori kada je podatak postavljen na 8-bitni izlaz.

Tabela 1: Vrednosti bitrate.

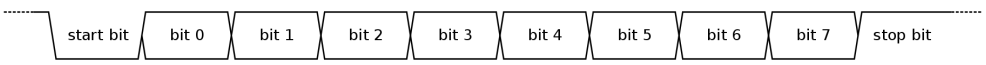
bitrate_in	bitrate value
"000"	1200 bits/s
"001"	2400 bits/s
"010"	4800 bits/s
"011"	9600 bits/s
"100"	19200 bits/s
"101"	38400 bits/s
"110"	57600 bits/s
"111"	115200 bits/s



Slika 1: Šema komponente UART

3.1 Oblik podataka

Podaci se prenose u NRZ (*Non Return to Zero*) kodu, odnosno, logičko 1 odgovara višem naponu i logičko 0 nižem naponu. Jedan podatak koji se šalje preko serijskog interfejsa sastoji se iz startnog bita vrednosti '0', trajanja jedan taktni period, koji označava početak prenosa bitova podataka čiji broj može varirati od 5 do 9. U ovom radu pretpostavljeno je da se prenose isključivo 8-bitni podaci. Nakon 8 bitova podatka prenosi se stop bit vrednosti '1' (Slika 2.). Samo u slučaju da je podatak definisan u navedenom obliku se smatra da je podatak ispravan.



Slika 2: Izgled jednog podatka na serijskom interfejsu

Trajanje jednog bita zavisi od učestanosti taktnog signala koja za *Cyclone V SoC FPGA* iznosi 50 MHz i vrednosti *bitrate* koja predstavlja broj bitova koje je moguće preneti u jednoj sekundi [1].

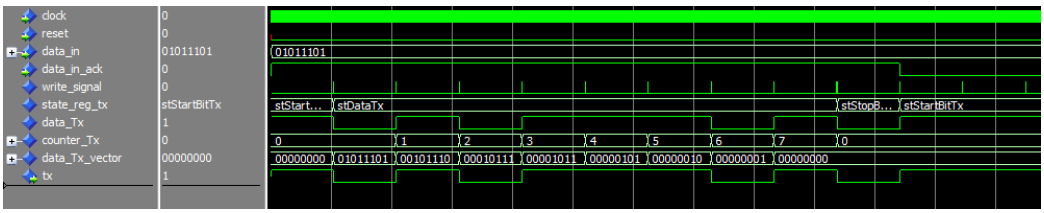
3.2 Slanje podataka

Proces slanja podataka podrazumeva slanje podataka koji se nalaze na 8-bitnom ulazu na serijski izlaz bit po bit redom od bita najmanje težine. Na osnovu frekvencije taktnog signala i *bitrate-a* generisan je signal *write* koji nosi informaciju u kojim trenucima će se ispisivati bit na serijski izlaz *tx*. Na slici 3 može se pratiti proces slanja podatka. Mašina stanja sastoji se iz tri različita stanja:

1. *stStartBitTx* - je stanje koje označava početak slanja podatka na izlaz. Kada se dobije znak da je unet podatak na 8-bitni paralelni ulaz (*data_in_ack* = '1') i signal *write* da informaciju da se ispiše signal na izlazu, tada se na izlaz ispisuje startni bit vrednosti '0', vrednosti sa paralelnog 8-bitnog ulaza se upisuju na privremeni registar, prelazi se u sledeće stanje i brojač *counter_Tx* koji se koristi u sledećem stanju se resetuje.

2. *stDataTx* - je stanje u kojem se ispisuju biti podataka jedan po jedan sve dok gore definisani brojač koji se u ovom stanju povećava nakon svakog upisa bita na izlaz ne izbroji da se na izlaz ispisalo svih 8 bitova. Svaki sledeći bit se šalje na osnovu periodičnog signala *write*. Kada su svi bitovi prosleđeni na izlaz prelazi se u sledeće stanje.

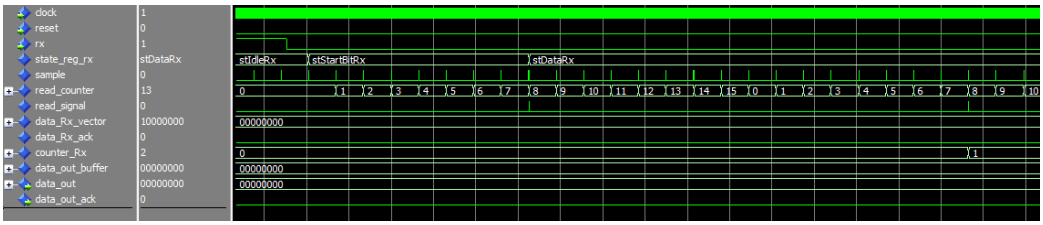
3. *stStopBitTx* - je stanje koje označava kraj slanja podatka. U toku tog stanja se na izlaz ispisuje stop bit vrednosti '1' pri signalu *write* i prelazi u prvo stanje.



Slika 3: Proces slanja podataka

3.3 Prijem podataka

Kod procesa prijema, suprotno u odnosu na proces slanja, podatak se očitava sa serijskog ulaza *rx* i postavlja na paralelni 8-bitni izlaz *data_out*. Obzirom da je komunikacija asinhrona, taktni signal na prijemnoj strani se generiše nezavisno od taktnog signala predajnika. Kako bi se obezbedila sinhronizacija na prijemnoj strani se vrši *oversampling*, odnosno odabira se 16 puta po bitu što predstavlja signal *sample* na slici 4. Kada je na prijemnoj strani detektovana opadajuća ivica, ona se smatra početkom start bita i počinje brojanje odbiraka (signal *read_counter*) [4].



Slika 4: Generisanje signala *read*

U idealnom slučaju svih 16 odbiraka start bita treba da imaju vrednost 0, ali kako bi se povećala otpornost na greške UART koristi samo neke od odbiraka za određivanje vrednosti bita. U ovom radu generisan je signal *read_signal* nakon svakog 8. odbirka (od 16 odbiraka). Mašina stanja čitanja podataka sastoji se od četiri stanja:

1. *stIdleRx* - je stanje u kom se čeka opadajuća ivica start bita i služi nam za sinhronizaciju. Pri visokoj vrednosti *sample* signala kada se detektuje pad na nulu na serijskom ulazu, prelazi se u sledeće stanje.

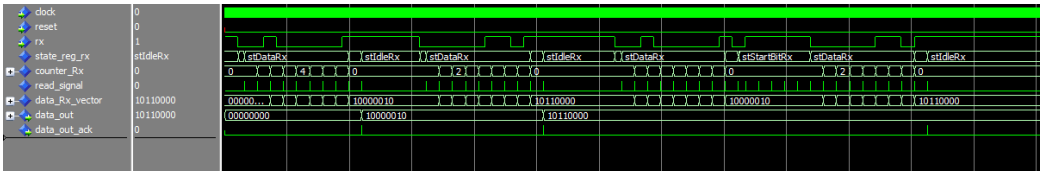
2. *stStartBitRx* - stanje proverava da li je stvarno u pitanju start bit, a ne *glitch* tako što pri *read* signalu očitava da li je i dalje nula na ulazu *rx* i ako jeste prelazi u sledeće stanje.

3. *stDataRx* - je stanje koje očitava vrednosti bitova podataka sa ulaza. Pri svakom *read* signalu očitava se vrednost na ulazu, povećava se brojač *counter_Rx* koji broji koliko bitova podataka je očitano i očitana vrednost se dodaje na niz koji čuva učitane vrednosti koje se prosleđuju na 8-bitni paralelni izlaz *data_Rx_vector*. Kada brojač kaže da smo očitali svih 8 bitova podataka prelazi se u sledeće stanje.

4. *stStopBitRx* - u ovom stanju se proverava da li je poslednji bit u paketu koji predstavlja jedan podatak ispravno uneti stop bit tj. vrednost '1'. Na *read* signal se očitava vrednost sa ulaza i ukoliko je '1'

data_Rx_vector se prosleđuje na 8-bitni paralelni izlaz *data_out* i generiše se signal *data_out_ack* koji nosi informaciju o tome kada je novi podatak postavljen na izlaz. Nakon očitavanja prelazi se u prvo stanje nezavisno od toga da li je prethodni uslov ispunjen.

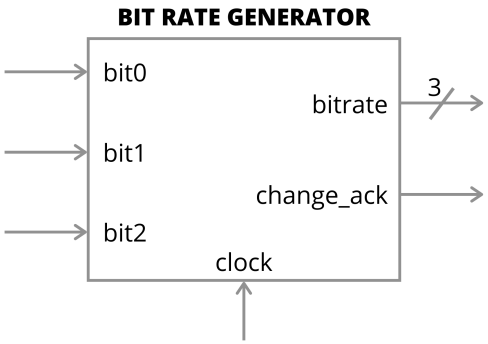
Na slici 5 moguće je pratiti gore opisani metod čitanja podataka.



Slika 5: Proces čitanja podataka

4 Bit Rate Generator

Vrednost *bitrate* koju je neophodno proslediti komponenti UART definiše se pomoću tri prekidača (Slika 6.) tako što se na osnovu vrednosti na tim prekidačima generiše 3-bitni vektor *bitrate* koji određuje vrednosti brzine prenosa kao u tebeli 1. Signal *change_ack* nosi informaciju o tome kada je došlo do promene *bitrate-a*.

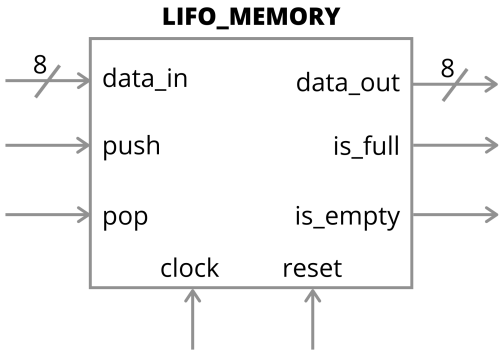


Slika 6: Šema komponente *Bit Rate Generator*

5 LIFO memorija

LIFO memorija radi po principu *Last In First Out* tj. kao stek. Pri visokoj vrednosti signala *push*, podatak sa ulaza *data_in* se upisuje u memoriju tako

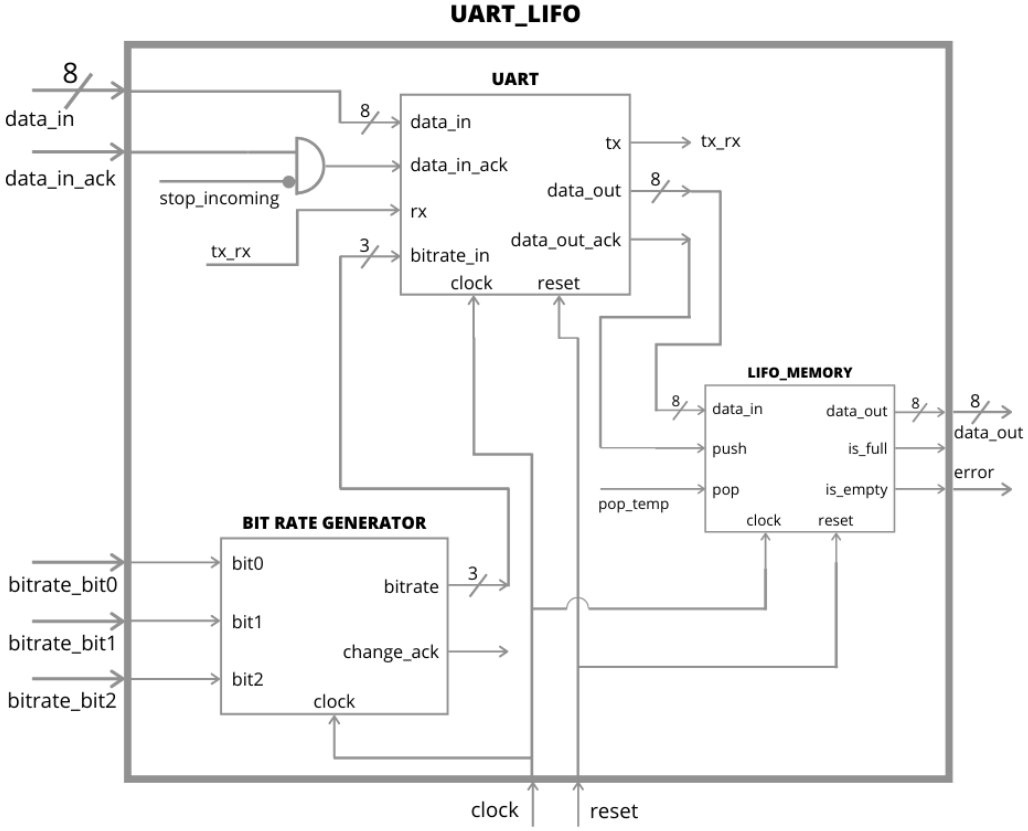
što se postavlja na vrh memorije. Pri visokoj vrednosti *pop* signala podatak sa vrha memorije se skida i prosleđuje na izlaz *data_out*. Na taj način se postiže ispis podataka u suprotnom redosledu u odnosu na onaj kojim su podaci uneti u memoriju. Signali *is_full* i *is_empty* nose informaciju o tome da li je popunjena cela izdvojena memorija i nije moguće dodavati još elemenata odnosno da li je memorija prazna i nije moguće više ispisivati vrednosti iz memorije na izlaz.



Slika 7: Šema komponente *LIFO memorija*

6 UART LIFO

Prethodne tri komponente (UART, BRG i LIFO memorija) spojene su kao na slici 8 i čine komponentu UART LIFO. Sa ulaza *data_in* očitavaju se podaci koji predstavljaju ASCII kodove karaktera sve dok uneta vrednost ne predstavlja *ENTER* tj. *CarriageReturn* = "00001101". Uneti podatak 8-bitnog paralelnog oblika se preko UART predajnika prenosi serijskom vezom do UART prijemnika. UART prijemnik pri generisanju 8-bitnog podatka generiše i signal *data_out_ack* kao informaciju da je podatak očitan koji ujedno predstavlja i znak da se podatak upiše na vrh memorije. Unutar komponente postoji brojač koji broji koliko podataka je upisano u memoriju i kada je detektovano da je unet *ENTER* onemogućava se dalji unos podataka *stop_incoming* signalom, generiše se *pop_temp* signal onoliko puta koliko karaktera treba ispisati. Period *pop_temp* signala zavisi od definisanog *bitrate*-a. Kada se završi ispis podataka iz memorije omogućava se ponovni unos podataka na taj način što se vrednost *pop_temp* vraća na '0'.

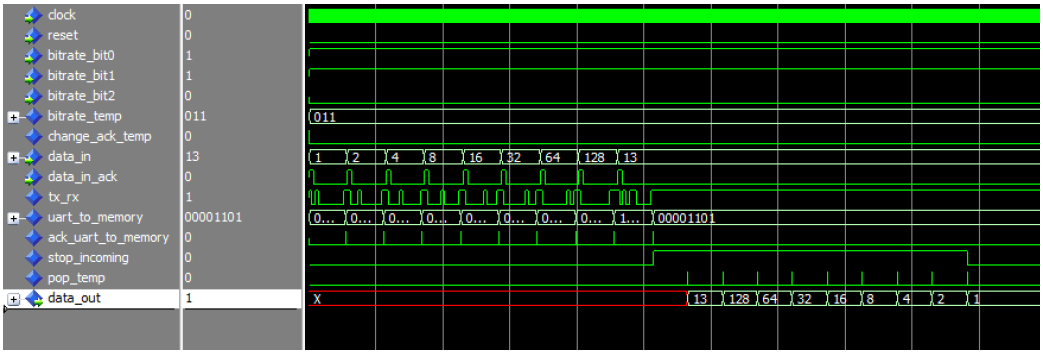


Slika 8: Šema komponente *UART LIFO*

7 Zaključak

Na slici 9 prikazan je rad UART LIFO komponente. Signal *data_in* predstavlja podatke koji se unose. Zbog lakšeg praćenja podaci su prikazani kao decimalni brojevi gde je $CarriageReturn =$ "00001101" zapravo broj 13. Na *data_in_ack* signal se prebacuje na serijski *tx_rx* sa kog se očitava i šalje kao 8-bitni *uart_to_memory* signal i na pikove *ack_uart_to_memory* upisuje u memoriju. Kada je unet *ENTER*, *stop_incoming* signal blokira dalji unos i na *pop_temp* ispisuje vrednosti u obrnutom redosledu u odnosu na unete vrednosti.

Pri unosu podataka neophodno je voditi računa u trajanju signala *data_in*. U slučaju da signal kraće traje može doći do toga da ne bude očitan i preskočen kao da se nije dogodio. Ovaj problem moguće je rešiti dodavanjem *FIFO - First in First Out* memorije [3]. Upis u memoriju ne zavisi od *bitrate*-a UART-a te se podaci mogu brže upisivati u memoriju pa iz memo-



Slika 9: Prikaz rada komponente UART LIFO

rije prosleđivati na UART. U tom slučaju je sistem nezavisan od *bitrate-a* i signali na ulazu i izlazu mogu kraće da traju.

Takođe, u slučaju da je potrebno izlazni signal dodatno obrađivati na računaru neophodno je poslati ga ponovo preko UART-a na računar.

Literatura

[1] *DE1-Soc User Manual*. **version** Version 1.2.3. Terasic Technologies Inc., Altera University Program. URL: `https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1004282204-de1-soc-user-manual.pdf`.

[2] Peter Marwedel. **in**: *Embedded System Design. Embedded Systems, Foundations of Cyber-Physical Systems, and the Internet of Things*. Third Edition. Springer International Publishing, 2018. **chapter** 2.7. ISBN: 978-3-319-56043-4.

[3] Naresh Patel, Vatsalkumar Patel **and** Vikaskumar Patel. "VHDL Implementation of UART with Status Register". **in**: *International Conference on Communication Systems and Network Technologies*. 2012 IEEE, **pages** 750–754. ISBN: 978-0-7695-4692-6/12.

[4] Lazar Saranovac **and** Ivan Popović. **in**: *Namenski računarski sistemi*. 2017. **chapter** 3.3. ISBN: 978-86-7466-703-3.