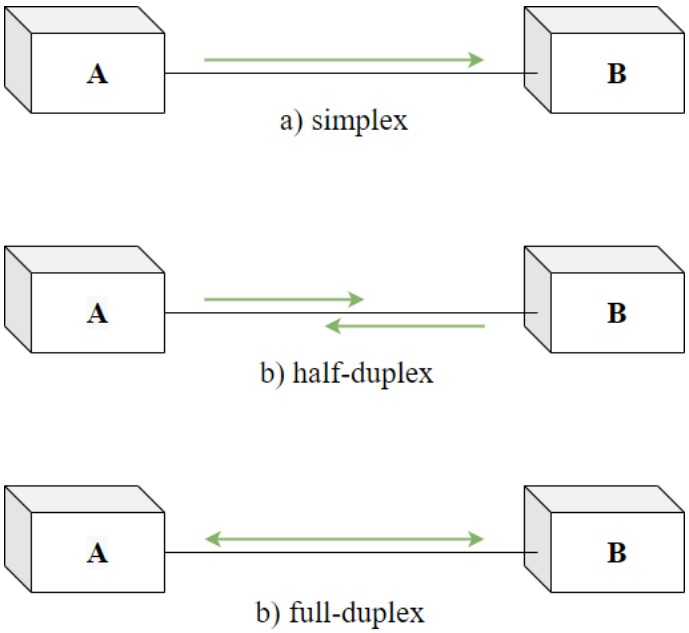
UART (*Universal Asynchronous Receiver-Transmitter*) komunikacija

1 Serijska komunikacija

Serijska komunikacija veoma je važna sa stanovišta projektovanja mikrokontrolerskih sistema i njena uloga je razmena podataka u vidu bitova između dva ili više mikrokontrolera, kao i mikrokontrolera i periferija koje vrše akviziciju i obradu podataka. Serijsku komunikaciju karakteriše razmena podataka u serijskom obliku, odnosno prenos podataka bit po bit. Za razliku od serijske komunikacije, paralelna komunikacija podrazumeva simultani prenos većeg broja bitova. Serijska komunikacija, zbog načina prenosa podataka koji je karakteriše, zahteva upotrebu manjeg broja pinova, što jeste jedna od prednosti serijske komunikacije u odnosu na paralelnu komunikaciju. U zavisnosti od načina prenosa podataka između uređaja razlikuju se tri metode: simplex, half-duplex i full-duplex metoda. Simplex metoda (slika 1 a)) podrazumeva unidirekcionu razmenu podataka, gde samo jedan od uređaja šalje podatke. Half-duplex metoda (slika 1 b)) označava prenos podataka u oba smera, međutim, ne istovremeno. U datom trenutku samo jedan uređaj može da šalje podatke, dok drugi uređaj prima date podatke. Ukoliko se razmena podataka vrši u oba smera, i uređaji mogu istovremeno da šalju podatke jedni drugima, radi se o full-duplex metodi (slika 1 c)).



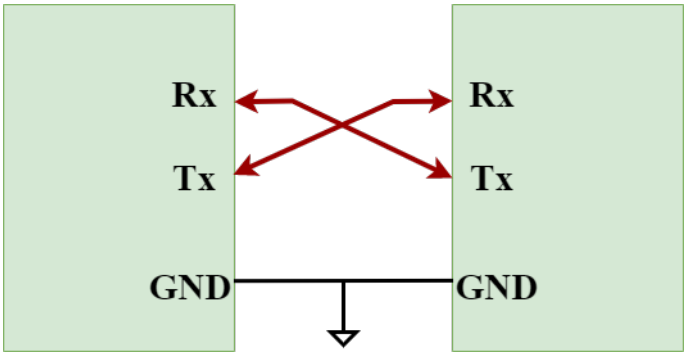
Slika 1: Ilustracija a) simplex b) half-duplex i c) full-duplex metode prenosa podataka

U odnosu na postojanje signala takta, razlikuju se sinhrona i asinhrona serijska komunikacija. Sinhrona serijska komunikacija podrazumeva prenos podataka bit po bit sinhronizovano sa definisanim taktom i ovakav prenos podataka zahteva dodatnu liniju za signal takta. Kod asinhrone serijske komunikacije podaci se prenose bit po

bit bez upotrebe signala takta, što zahteva upotrebu nekoliko parametara kojima se obezbeđuje pravilan prenos i prijem podataka. Kako bi se ostvarila komunikacija između uređaja baziranih na mikrokontrolerima, potrebno je da se razmena podataka vrši po predefinisanim komunikacionim protokolima, koji su veoma značajni sa stanovišta dizajna firmvera, kao i hardvera. Serijski komunikacioni protokoli su široko rasprostranjeni pri razmeni podataka u embedded sistemima i veliki broj periferija, kao što su analogno-digitalni konvertori, LCD (*Liquid Crystal Display*) i različite senzorske periferije, podržava neki od serijskih komunikacionih protokola. Dodatno, ovi komunikacioni protokoli su značajni jer omogućavaju i razmenu podataka sa personalnim računarima, koji u najvećem broju slučajeva sadrže neki od interfejsa serijske komunikacije. Međutim, jedan od najvećih problema serijske komunikacije jeste detektovanje i otklanjanje grešaka koje nastaju u procesu razmene podataka. Moguća mesta nastanka grešaka jesu u logici rada predajnika, u električnom kolu predajnika, u kodovanju serijske komunikacije, na magistrali, u procesu dekodovanja podataka, u električnom kolu prijemnika, ili u logici rada prijemnika. Neki od komunikacionih protokola sinhrone serijske komunikacije jesu SPI (*Serial Peripheral Interface*), I^2C (*Inter-Integrated Circuit*), CAN (*Controller Area Network*) i LIN (*Local Interconnect Network*) protokol. U slučaju asinhrone serijske komunikacije neki od komunikacionih protokola su RS-232, RS-422 i RS-485 protokol.

2 UART

UART modul predstavlja električno kolo koje može biti samostalno integrisano kolo ili se nalaziti u sastavu mikrokontrolera. Uloga UART modula jeste asinhroni prenos i prijem podataka u serijskom obliku. UART je point-to-point komunikacioni interfejs, što podrazumeva da se razmenu podataka vrši između dva UART uređaja koji komuniciraju direktno jedan sa drugim. Asinhrona komunikacija, kao što je prethodno navedeno, podrazumeva da ne postoji takt signal koji sinhronizuje izlazne bitove predajnika sa odabiranjem bitova na ulazu prijemnika. Iz tog razloga uređaji koji sadrže UART modul mogu da komuniciraju upotrebom svega dve linije za razmenu podataka i zajedničke mase, koja nije komunikaciona linija, ali je važna u pogledu referenciranja naponskih nivoa u odnosu na istu tačku. Svi mikrokontroleri koji komuniciraju unutar jedne mreže moraju biti vezani na istu referentnu tačku kako bi mogli na ispravan načina da prepoznaju podatke koji pristižu sa drugih mikrokontrolera. Na slici 2 prikazan je način povezivanja dva uređaja koji sadrže UART modul. Rx i Tx linije označavaju komunikacione linije, gde je Rx prijemna (*receive*) linija, a Tx predajna (*transmit*) linija. Dakle, Tx linija jednog uređaja vezuje se na Rx liniju drugog, i obrnuto, a ovakav način povezivanja uređaja naziva se ukršteno povezivanje. Takođe, na slici prikazana je i zajednička masa (GND) ova dva uređaja.

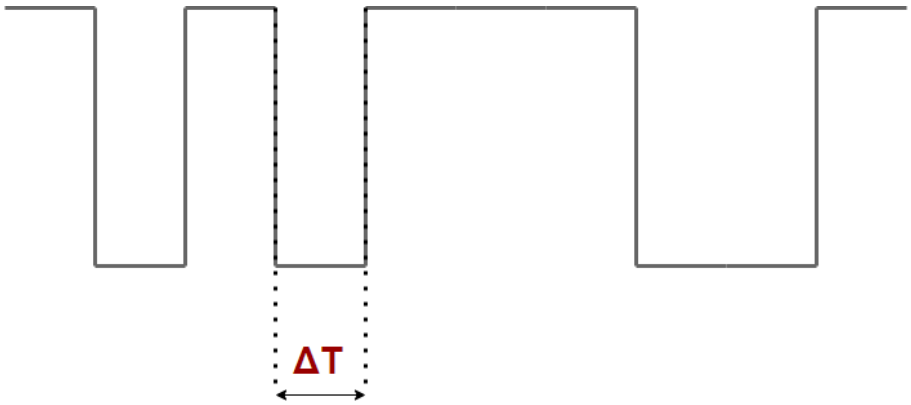


Slika 2: Povezivanje UART modula

Kako bi podaci bili ispravno poslati i dekodovani potrebno je da na magistrali bude definisano stabilno stanje kada nema razmene podataka, odnosno kada linije nisu aktivne (*idle mode*). Kod komunikacionog protokola UART komunikacije ovo stanje definisano je kao visok naponski nivo, odnosno, logička jedinica. Prvi uređaj putem Tx linije zadaje stanje logičke jedinice kada nema razmene podataka, a dato stanje se propagira na Rx ulaz drugog uređaja. Takođe, drugi uređaj na Tx liniji održava visoki naponski nivo za vreme neaktivnog stanja, a on se prenosi na Rx liniju prvog uređaja.

2.1 Prenos podataka kod UART komunikacije

Kako je u pitanju asinhrona serijska komunikacija i signali nisu sinhronizovani jedni sa drugima pomoću zajedničkog signala takta, neophodno je uvođenje određenih parametara, kao što su bitovi koji označavaju početak i kraj poruke koja se prenosi – start i stop bit, kao i brzina komunikacije. **Start bit** ima vrednost logičke nule, odnosno slanjem start bita predajnik vrši promenu stanja Tx linije sa visokog na niski naponski nivo, a ova opadajuća ivica signala naznačava prijemniku da je započeta razmena podataka. **Stop bit** označava kraj prenosa određene sekvence podataka i ima vrednost logičke jedinice. Predajnik uvek šalje jedan start bit kako bi započeo komunikaciju, a u zavisnosti od konfiguracije broj stop bitova može biti jedan ili više. Između start i stop bita vrši se prenos bitova podataka. Prenos podataka podrazumeva promenu stanja na Tx liniji između logičke 1 i 0 u odgovarajućim vremenskim trenucima. Broj bitova podataka može biti od pet do devet, a najčešće je osam. Prvi bit podataka koji predajnik šalje najčešće jeste LSB (*Least Significant Bit*). Za ispravnu razmenu podataka putem UART modula neophodno je da uređaji koji komuniciraju imaju podešene iste brzine prenosa i prijema bitova kako bi svi bitovi imali jednako trajanje i kako bi odabiranje signala bilo vršeno u ekvidistantnim vremenskim trenucima. **Baud rate** određuje brzinu UART komunikacije i označava broj bitova prenesenih u jednoj sekundi. Ukoliko je baud rate dva uređaja podešen na iste vrednosti smatra se da su UART uređaji sinhronizovani, trajanje svakog bita ΔT je tačno određeno kao recipročna vrednost baud rate-a (slika 3), a odabiranje bita vrši se u trenutku $\Delta T/2$. Neke od predefinisanih vrednosti koje se koriste za baud rate su 9 600, 19 200, 38 400, 57 600 i 115 200 itd. U odnosu na definisani baud rate, kod asinhrone serijske komunikacije moguće je odrediti koliko vremena je potrebno za prenos određene količine podataka.



Slika 3: Dužina trajanja jednog bita

Kako na UART komunikaciju utiču neusklađenost brzine komunikacije dva uređaja, smetnje usled elektromagnetnog zračenja i udaljenost uređaja, potrebno je definisati odgovarajući način kontrole pojavljivanja grešaka tokom komunikacije. Osim toga, greške u komunikaciji mogu se javiti i ukoliko se koriste uređaji sa oscilatorima male klase tačnosti, gde dolazi do pogrešnog odabiranja bitova i određeni podaci mogu biti izgubljeni. Kontrola pojave grešaka u UART komunikaciji se vrši upotrebom **bita parnosti**. Bit parnosti je opcioni bit koji se može slati nakon bitova podataka i on govori o ukupnom broju logičkih 1 u signalu koji se prenosi, čime naznačava da li je u toku komunikacije došlo do promena u podacima. Ukoliko se bit parnosti koristi za proveru nastanka greške u komunikaciji, potrebno je da korisnik definiše da li se radi o parnoj ili neparnoj parnosti. Parna parnost označava da je ukupan broj logičkih 1 u signalu koji se prenosi paran. Dakle, ukoliko se radi o parnoj parnosti i ukoliko je broj logičkih 1 u bitovima podataka paran, bit parnosti postavlja se na logičku 0, čime se obezbeđuje da je ukupan broj logičkih 1 u signalu paran. U slučaju da se u bitovima podataka nalazi neparan broj logičkih 1, bit parnosti se postavlja na logičku 1. Za neparnu parnost važi da ukupan broj logičkih jedinica u signalu treba da bude neparan, što znači da je vrednost bita parnosti u slučaju neparnog broja logičkih 1 u podacima 0. Predajnik i prijemnik moraju imati podešene iste parnosti. Prijemni uređaj proverava bit parnosti i broj logičkih jedinica u signalu i na taj način može da detektuje da je došlo do promene vrednosti jednog od bitova, i tada dati podaci moraju biti poslati ponovo. Iako je ovaj način provere greške jednostavan, ne daje informaciju o mestu u signalu na kome je došlo do greške. Takođe, u slučaju pojave grešaka na više bitova, tako da je ukupan broj logičkih jedinica ostao paran, odnosno, neparan, nije moguće detektovati greške pomoću bita parnosti.

Za realizaciju ovakvog načina provere postojanja greške predajnik treba da sadrži logiku za generisanje bita parnosti, a prijemnik logiku za proveru bita parnosti. Pri implementaciji logike za generisanje i proveru bita parnosti koristi se eks-ili (xor) logička funkcija. Realizacija ovakvih logičkih kola biće objašnjena kroz primer parne parnosti i tri bita podataka:

- **Generisanje bita parnosti**

  Ulazi kola za generisanje bita parnosti su bitovi podataka koji se prenose. U slučaju da je broj logičkih jedinica u tri bita podataka neparan, potrebno je da

se bit parnosti postavi na vrednost 1, u suprotnom na vrednost 0. Pri minimizaciji date funkcije na disjunktivnu formu (zbir proizvoda ulaznih promenljivih i njihovih invertovanih vrednosti) koriste se Karnoove tabele. U Karnoovu tabelu unose se vrednosti iz kombinacione tabele, zatim se sve jedinice prekrivaju četvorougaonim maskama veličine 1 i svakoj od maski se pridružuje jedan proizvod ulaznih promenljivih. Odgovarajuća tablica istinitosti i Karnoova tabela date su u tabelama 1 i 2, redom, dok je rezultujuće logičko kolo prikazano na slici 4. Promenljive koje u svim poljima maske imaju vrednost 0 se invertuju, dok promenljive koje imaju vrednost 1 ostaju neinvertovane. Na ovaj način dolazi se do izraza:

$$P = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot C$$

Dalje, upotrebom osnovnih postulata Bulove algebre dolazi se do sledećeg:

$$P = \overline{A} \cdot (\overline{B} \cdot C + \overline{B} \cdot BC) + A(\overline{B} \cdot \overline{C} + B \cdot C)$$

Ukoliko se eks-ili funkcija predstavi preko i i ili funkcije kao $A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$ prethodni izraz postaje:

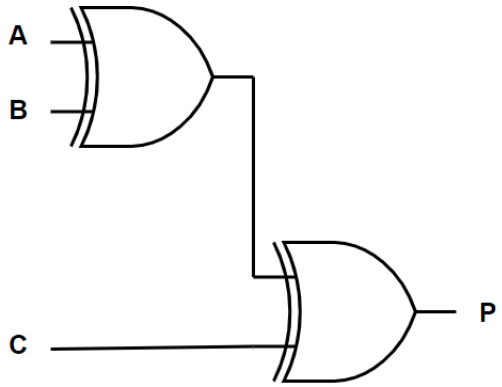
$$P = \overline{A} \cdot (B \oplus C) + A \cdot (\overline{B \oplus C}) = A \oplus B \oplus C$$

A	**B**	**C**	**P**
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**1**

Tabela 1: Tablica istinitosti - generisanje bita parnosti

A/BC	**00**	**01**	**11**	**10**
0	0	1	0	1
1	1	0	1	0

Tabela 2: Karnoova tabela - generisanje bita parnosti



Slika 4: Logičko kolo za generisanje bita parne parnosti

- **Provera bita parnosti**

  Zadatak kola za proveru bita parnosti je da utvrdi da li je pri prenosu podataka došlo do promene vrednosti nekog od bitova. Ulazi ovog kola su bitovi podataka koji pristižu na prijemni uređaj putem Rx linije, kao i bit parnosti. Vrednost izlaza kola za proveru bita parnosti je 1, ukoliko je došlo do greške (broj logičkih jedinica je neparan), ili 0, ukoliko greška nije detektovana. Na ovaj način formira se tablica istinitosti (tabela 3), a zatim na prethodno opisan način i Karnoova tabela (tabela 4), koja u ovom slučaju ima četiri vrste i četiri kolone. Logičko kolo koje se dobija kao rezultat ilustrovano je slikom 5.

  Minimizaciom funkcije u disjunktivnoj formi upotrebom Karnoove tabele dobija se:

$$E = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot P+\overline{A}\cdot B\cdot\overline{C}\cdot\overline{P}+\overline{A}\cdot B\cdot C\cdot P+A\cdot B\cdot\overline{C}\cdot P+A\cdot B\cdot C\cdot\overline{P}+A\cdot\overline{B}\cdot\overline{C}\cdot\overline{P}+A\cdot\overline{B}\cdot C\cdot P$$

$$E = \overline{A}\cdot\overline{B}\cdot(\overline{C}\cdot P+C\cdot\overline{P})+\overline{A}\cdot B\cdot(\overline{C}\cdot\overline{P}+C\cdot P)+A\cdot B\cdot(\overline{C}\cdot P+C\cdot\overline{P})+A\cdot\overline{B}\cdot(\overline{C}\cdot\overline{P}+C\cdot P)$$

$$E = (\overline{A}\cdot B + A\cdot\overline{B})\cdot(\overline{C}\cdot\overline{P} + C\cdot P) + (\overline{A}\cdot\overline{B} + A\cdot B)\cdot(\overline{C}\cdot P + C\cdot\overline{P})$$

  Kako važi da je $A \oplus B = A\overline{B} + \overline{A}B$, sledi:

$$E = (A \oplus B)\cdot(\overline{C \oplus P}) + (\overline{(A \oplus B)}\cdot(C \oplus P)$$

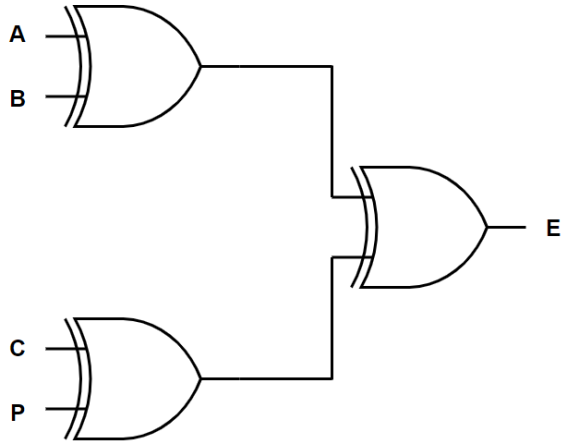
$$E = (A \oplus B) \oplus (C \oplus P)$$

A	**B**	**C**	**P**	**E**
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**1**
0	0	1	1	**0**
0	1	0	0	**1**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**1**
1	0	0	1	**0**
1	0	1	0	**0**
1	0	1	1	**1**
1	1	0	0	**0**
1	1	0	1	**1**
1	1	1	0	**1**
1	1	1	1	**0**

Tabela 3: Tablica istinitosti - proveravanje bita parnosti

AB/CP	**00**	**01**	**11**	**10**
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

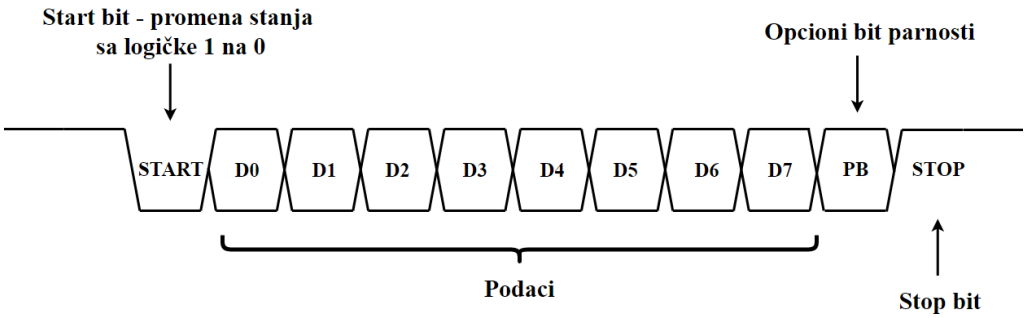
Tabela 4: Karnoova tabela - proveravanje bita parnosti



Slika 5: Logičko kolo za proveru parne parnosti

Dakle, podaci koji se prenose UART modulom su organizovani u pakete, a svaki paket se sastoji od jednog start bita, pet do devet bitova podataka, opciono jednog bita parnosti i jednog ili više stop bitova (slika 1). U uređaju koji šalje podatke, paralelnim prenosom, putem magistrale podataka bitovi se šalju UART modulu iz CPU (*Central Processing Unit*), memorije i sl. Predajni UART modul dodaje start bit, stop bit i bit parnosti (opciono) i na taj način formira paket. Paket se šalje na prijemnik bit po bit, preko Tx pina predajnika. Prijemnik, zatim, očitava paket bit po bit sa Rx pina. Sa prijemne strane uklanja se start bit, stop bit i bit parnosti i dalje se paralelnim prenosom bitovi podataka šalju na magistralu podataka sa prijemne strane. Pre same razmene podataka na predajnom i prijemnom uređaju moraju biti definisane iste vrednosti sledećih parametara UART komunikacionog protokola: baud rate, broj stop bitova, broj bitova podataka i parnost. Pri komunikaciji dva uređaja putem UART modula tok razmene podataka podrazumeva sledeće korake:

- Neaktivno stanje na Tx liniji je visok naponski nivo koji označava da se ne vrši razmena podataka.
- Predajnik započinje komunikaciju sa prijemnikom tako što šalje start bit u trajanju od ΔT, čija je vrednost nizak naponski nivo.
- Prijemnik nakon detekcije opadajuće ivice aktivira tajmer koji meri vreme ΔT.
- Nakon vremena ΔT, odnosno, po isteku start bita, predajnik započinje serijsko slanje bitova podataka, koji takođe imaju tačno određeno vremensko trajanje ΔT.
- Prijemnik vrši odabiranje signala na magistrali frekvencijom koja je definisana baud rate-om.
- Opciono, nakon bitova podataka, predajnik šalje jedan bit parnosti.
- Jedan ili više stop bitova označavaju kraj paketa i magistrala se postavlja na visok naponski nivo (povratak u neaktivno stanje).
- Tek nakon isteka trajanja stop bitova komunikacija između uređaja može da se nastavi.



Slika 6: Paket podataka koji se prenosi putem UART modula

3 UART modul PIC18F87K22 mikrokontrolera

Mikrokontroler PIC18F87K22 sadrži dva EUSART (*Enhanced Universal Synchronous Asynchronous Receiver Transmitter*) modula namenjena za serijsku komunikaciju – EUSART1 i EUSART2. EUSART modul mikrokontrolera može biti konfigurisan kao full-duplex asinhroni sistem čija je uloga razmena podataka sa periferijama kao što je PC. EUSART modul takođe može biti konfigurisan i kao half-duplex sinhroni sistem za komunikaciju sa periferijama poput A/D i D/A konvertora. U slučaju upotrebe EUSART modula za sinhronu half-duplex komunikaciju, modul može da radi kao master ili slave uređaj. Pinovi PIC18F87K22 mikrokontrolera koje koriste EUSART moduli su pinovi porta C i G – pinovi RC6 i RC7 su konfigurisani kao Tx i Rx pinovi EUSART1, dok su pinovi RG1 i RG2 konfigurisani kao Tx i Rx pinovi EUSART2.
EUSART moduli se podešavaju preko tri registra:

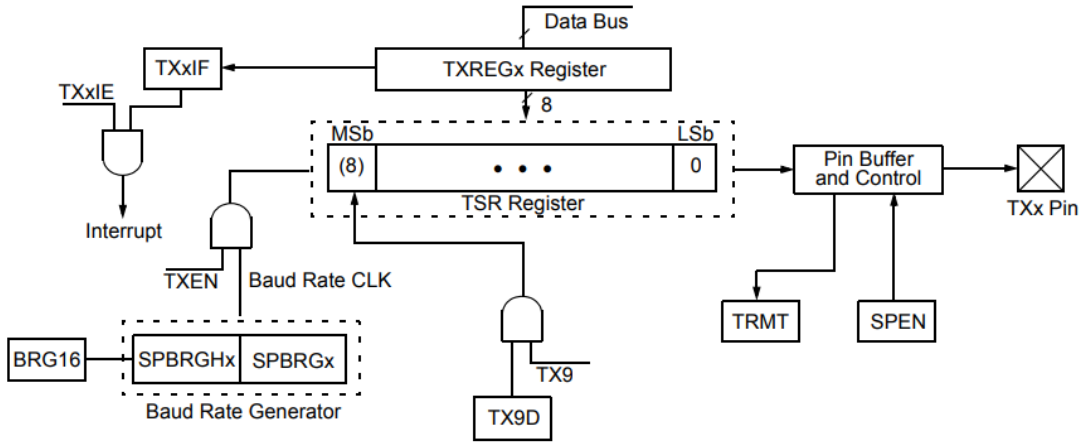
1. TXSTAx (*Transmit Status and Control*),
2. RCSTAx (*Receive Status and Control*),
3. BAUDCONx (*Baud Rate Control*)

Postavljanjem bita B4 registra TXSTAx na 1 modul se konfiguriše za sinhronu komunikaciju, dok vrednost nula označava asinhroni mod rada modula.

Baud rate komunikacije podešava se kroz baud rate generator postavljanjem vrednosti para registara SPBRGHx (EUSARTx Baud Rate Generator Register High Byte) i SPBRGx (EUSART Baud Rate Generator Register). Na osnovu željenog baud rate-a i frekvencije oscilatora mikrokontrolera moguće je doći do vrednosti koju je potrebno upisati u navedene registre upotrebom formula navedenih u datasheet-u mikrokontrolera. U asinhronom modu rada modula baud rate se dodatno podešava i kroz registre TXSTAx i BAUDCONx. Takođe, proizvođač je u datasheet-u mikrokontrolera naveo vrednosti stvarnog baud rate-a, greške, kao i vrednosti koje se upisuju u SPBRGx registre za vrednosti baud rate-a koje se najčešće koriste i različite vrednosi frekvencija oscilatora mikrokontrolera.

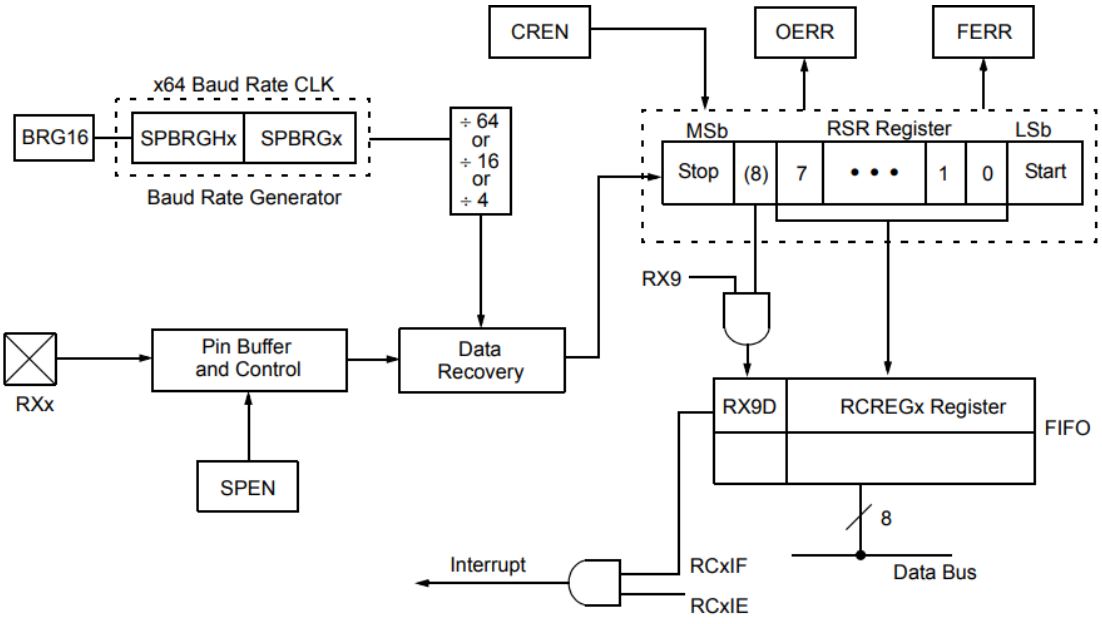
Kako bi se EUSARTx modul konfigurisao za asinhronu serijsku komunikaciju (UART), kao što je prethodno navedeno, potrebno je postaviti vrednost SYNC bita TXSTAx (B4) registra na logičku nulu. Podrazumevani mehanizam kodovanja podataka u ovom slučaju je NRZ (*Non-Return-to-Zero*), koji podrazumeva predstavu logičke jedinice kao pozitivnog (visokog) napona, i nule kao nultog ili negativnog napona, bez postojanja međustanja. Elementi paketa bitova koji se na ovaj način prenosi su jedan start bit, osam ili devet bitova podataka i jedan stop bit. Bit parnosti nije hardverski podržan, ali ga je moguće implementirati softverski i njegovu vrednost upisivati na mesto devetog bita podataka. Pri prenosu podataka prvi bit koji se šalje, odnosno prima, je LSB.

Na slici 7 prikazan je blok dijagram predajnog dela EUSART modula u asinhronom modu rada. Putem magistrale podataka podaci pristižu u TXREGx (*Read/Write Transmit Buffer*) registar. Kada je poslat stop bit prethodnog paketa, podaci se iz TXREGx prenose u TSR (*Transmit Shift Register*) registar i bit TXxIF se postavlja na vrednost logičke 1, što označava da je registar TXREGx ispražnjen. Odatle se podaci, bit po bit, prenose na izlazni TXx pin.



Slika 7: Blok dijagram EUSART asinhronog predajnika PIC18F87K22 mikrokontrolera

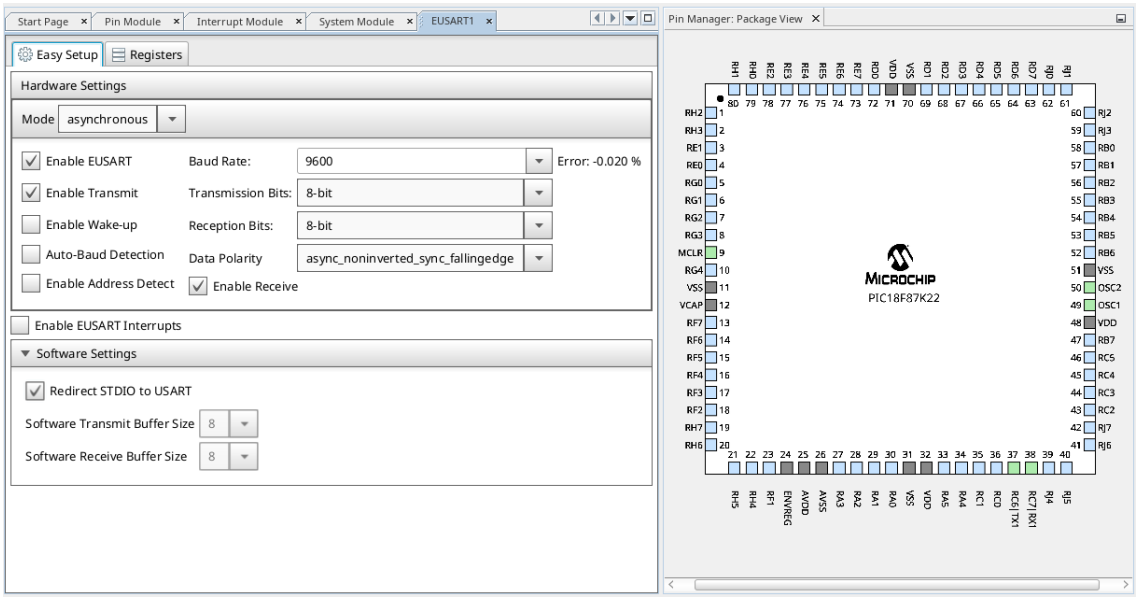
Prijem podataka vrši se preko pina RXx. Bit po bit pristiže u RSR (*Receive Shift Register*) registar, odakle se bitovi podataka šalju u RCREGx FIFO (*First In First Out*) prijemni bafer, kao i dodatni adresni, odnosno, bit parnosti, ukoliko postoji, a bit RCxIF se u tom trenutku setuje. Dalje se iz RCREGx registra podaci paralelno prenose putem magistrale podataka (slika 8).



Slika 8: Blok dijagram EUSART asinhronog prijemnika PIC18F87K22 mikrokontrolera

4 UART biblioteka MPLAB

Kako bi se koristio UART modul mikrokontrolera u *MPLAB Code Configurator (MCC)* alatu neophodno je dodati iz *Device Resources* padajućeg menija pod poljem *Peripherals→EUSART - EUSART1* modul. Tom prilikom se u *MCC* dodaje nova kartica pod nazivom EUSART1 prikazana na slici 9.



Slika 9: EUASRT1 modul nakon dodavanja u konfigurator

Odabran mod rada EUSART modula je asinhroni (UART), uključen je modul (*Enable EUSART*) kao i prijem *Enable Receive* i slanje *Enable Transmit* podataka. Polje *Enable Wake-up* omogućuje mogućnost automatskog buđenja mikrokontrolera iz stanja spavanja kada se detektuje silazna ivica na RX pinu. *Auto Baud Detection* omogućuje da mikrokontroler automatski detektuje brzinu prenosa podataka, a da bi to bilo moguće neophodno je da računar ili sistem sa kojim se komunicira prosledi 0x55 (0b01010101, odnosno pravougaoni signal) što omogućuje merenje trajanja jednog bita na osnovu čega se određuje baud rate. Odabiranjem *Enable Address Detect* funkcionalnosti omogućuje se automatska detekcija adrese korišćenjem dodatnog, devetog bita (rezervisanog kao bit pariteta). U padajućoj listi *Baud Rate* moguće je specificirati brzinu prenosa podataka, gde se za podešenu brzinu oscilatora određuje i greška od željene vrednosti (u ovom slučaju $-0,02$ %). Zatim je podešeno u padajućim listama *Transmission Bits* i *Reception Bits* da će u razmeni podataka biti poslato odnosno primljeno osam bita podataka. Moguće je odabrati i devet bita, ali se najčešće deveti bit koristi kao bit pariteta, mada to nije obavezno. *Data Polarity* lista omogućuje izbor logičkog stanja u stanju pripravnosti Tx linije gde je odabrano neinvertovano stanje tj. u stanju pripravnosti linija za slanje se nalazi na visokom stanju (stanje logičke jedinice). Polje *Enable EUSART Interrupts* omogućuje automatsku detekciju prijema podataka i/ili mogućnost slanja podataka (prazan izlazni registar), metodom prekida o kojoj će biti više reči u narednim poglavljima. Opcija

Redirect STDIO to USART omogućuje korišćenje C funkcija za ispis i očitavanje karaktera iz konzole (*putch*, *printf* za ispis i *getch* za očitavanje).

Nakon generisanja koda u folderu *Source Files* → *MCC Generated Files* nalazi se fajl *eusart1.c*, koji sadrži funkcije za rukovanje UART modulom mikrokontrolera.

Funkcija *EUSART1_Initialize* ima zadatak da inicijalizuje UART modul mikrokontrolera. Za slučaj sa slike 9 podešeni su registri:

- BAUDCON1 se postavlja na vrednost 0x48 čime se isključuje automatsko određivanje brzine prenosa podataka, bira se neinvertovano stanje pripravnosti, isključuje mogućnost buđenja po prijemu podataka.
- RCSTA1 se postavlja na vrednost 0x90 čime se bira osmobitni režim prijema podataka, omogućuje se prijem podataka, isključuje se korišćenje 9 bita kao adresnog bita.
- TXSTA1 se postavlja na vrednost 0x24 što obezbeđuje slanje osmobitnih podataka po poruci, uključuje mogućnost slanja podataka, obezbeđuje asinhroni mod rada i omogućuje korišćenje dva registra (SPBRGH1 i SPBRG1) za podešavanje brzine prenosa podataka.
- SPBRGH1 i SPBRG1 registara na vrednosti 0x06 i 0x82, čijim se spajanjem (konkatenacijom) dobija vrednost 0x0682 odnosno dekadno 1666.

Brzina prenosa podataka je definisana izrazom (1).

$$Baud\_rate = \frac{F_{osc}}{4 \cdot (SPBRGH1 : SPBRG1 + 1)}, \tag{1}$$

gde F_{osc} predstavlja frekvenciju rada mikroprocesora. Pošto je brzina oscilatora mikrokontrolera 16 MHz i unutar mikrokontrolera se množi 4x, takt procesora je frekevencije 64 MHz. Pošto je spajanjem SPBRGH1 i SPBRG1 dobijena vrednost 1666, dobija se da je brzina prenosa 9598.08 bps. Ovom prilikom se javlja greška od -1.92 bps, što odgovara -0,02 % prikazanih na slici 9.

Funkcija *EUSART1_is_tx_ready* vraća bool vrednost u zavisnosti od toga da li je izlazni bafer (TXREG1) za slanje podataka prazan (bit PIR1bits.TX1IF), pod uslovom da je uključena mogućnost slanja podataka (bit TXSTA1bits.TXEN).

Funkcija *EUSART1_is_rx_ready* vraća bool vrednost u zavisnosti od toga da li je ulazni bafer za prijem podataka pun tj. da li je podatak primljen posredstvom UART-a (bit PIR1bits.RC1IF).

Funkcija *EUSART1_is_tx_done* vraća bool vrednost u zavisnosti od toga da je slanje podataka završeno (bit TXSTA1bits.TRMT) tj. da je izlazni šift registar prazan.

Funkcija *EUSART1_get_last_status* vraća vrednost greške koja je registrovana tj. Framing ili Overrun greška. Prilikom korišćenja ove funkcije neophodno je uključiti EUSART1 Interrupt u podešavanjima MCC, pomoću koje se vrši automatska detekcija greške. Pošto ova funkcionalnost nije implementirana, u ovom slučaju, ne postoji mogućnost automatske detekcije greške. Zbog toga i pripadajuće funkcije za rukovanje greškama neće biti pozvane (*EUSART1_DefaultFramingErrorHandler*, *EUSART1_DefaultOverrunErrorHandler* i *EUSART1_DefaultErrorHandler*).

Funkcija *EUSART1_Read* prvo proverava da li postoje pristigli podaci, a ako ne postoje ova funkcija će čekati u beskonačnoj petlji do prijema podataka (bit PIR1bits.RC1IF). Nakon prijema podataka proverava se da li je došlo do Overrun greške (bit RCSTA1bits.OERR) i ako je došlo do greške briše statuse grešaka (bit RCSTA1bits.CREN). Na kraju se vraća vrednost primljenog bajta preko UART-a (registar RCREG1).

Funkcija *EUSART1_Write* čeka da se oslobodi mesto u izlaznom registru proverom bita PIR1bits.TX1IF. Kada se prostor oslobodi prosleđena osmobitna promenljiva se smešta u izlazni registar (registar TXREG1), nakon čega interni UART modul automatski šalje podatke preko Tx pina.

Funkcije *getch* i *putch* predstavlja funkcije za redirekciju ispisa na konzolu u ispis na UART.

5 Primeri

Primer 1: *Slanje podataka preko UART-a*

Listing 1: Slanje podataka preko UART-a

```
1 #include "mcc_generated_files/mcc.h"
2 #include <stdio.h>
3 #include <math.h>
4
5 void main(void)
6 {
7     SYSTEM_Initialize();
8
9     printf("Ispis na UART terminal\n");
10    EUSART1_Write('a');
11    while (1)
12    {
13        printf("Zdravo!\n");
14        __delay_ms(500);
15    }
16 }
```

U listingu 1 dat je kod koji nakon inicijalizacije UART modula šalje podatke korišćenjem printf funkcije. Funkcija printf predstavlja standardnu C funkciju za ispis podataka u konzolu koja se u ovom slučaju redirektuju na UART. Na kraju stringa koji se šalje pridodat je karakter $\backslash n$ koji označava novi red (*new line*). Zatim se pozivom funkcije *EUSART1_Write* šalje karakter *a*. Važno je istaći da se u C programskom jeziku karakter nalazi između apostrofa (‘’) dok se string nalazi između navodnika (“”). Na kraju se ulazi u beskonačnu petlju gde sa sa pauzom od 500 ms šalje poruka *“Zdravo!”* uz pridodati karakter za novi red.

Primer 2. *Slanje informacija o pritisku tastera putem UART-a*

U okviru listinga 2 prikazan je kod čija je namena očitavanje pritiska tastera i slanje odgovarajućih informacija o pritisku tastera putem UART-a.

Upotrebom *typedef enum* definišemo tip *stanje*, koje može imati jednu od četiri predefinisane vrednosti: *cekanje*, *taster0*, *taster1* ili *taster2*.

Funkcija *debounce* vrši proveru pritiska tastera uz otkalnjanje efekta treperenja kontakta, upotrebom vremenske zadrške. Za prosleđen sadržaj PORT registra odgovarajućeg porta, određeni pin, vreme trajanje vremenske zadrške, kao i stanje na pinu koje je potrebno detektovati, funkcija *debounce* vraća vrednost *true* ukoliko je detektovana adekvatna promena stanja na pinu, i ukoliko se to stanje na pinu ustalilo u datom vremenu.

Funkcijom *provera_pritiska* vrši se provera stanja na pinovima RB0, RB1 i RB2, odnosno, vrši se detekcija pritiska tastera koji odgovaraju ovim pinovima. U okviru funkcije definiše se promenljiva *tasteri* tipa *stanje*, čija se inicijalna vrednost postavlja na *cekanje*. U odnosu na detekciju pritiska tastera upotrebom funkcije *debounce*, vrednost promenljive *tasteri* smenjuje se između stanja *taster0*, *taster1* i *taster2*, i prosleđuje kao izlaz funkcije.

U okviru *main* funkcije vrši se inicijalizacija sistema, zatim se definiše promenljiva *tasteri*, tipa *stanje*. U beskonačnoj petlji koristi se switch-case struktura za proveru vrednosti promenljive *tasteri*. Switch-case struktura podrazumeva navođenje izraza čije se vrednosti proveravaju (switch(izraz)), u našem slučaju to je promenljiva *tasteri*, i različitih slučajeva, odnosno, vrednosti koje ta promenljiva može imati (case slučaj1 ...). Ukoliko se vrednost promenljive poklapa sa nekim od navedenih slučajeva, izvršava se kod u okviru tog case-a. U slučaju da je vrednost promenljive *tasteri cekanje*, poziva se funkcija *provera_pritiska* i očitavaju se stanja na pinovima RB0, RB1 i RB2. Ukoliko je neki od odgovarajućih tastera pritisnut, promenljiva *tasteri* uzima jedno od stanja *taster0*, *taster1* ili *taster2* i izvršava se kod u okviru odgovarajućeg case-a, koji podrazumeva ispis poruke *"Pritisnut je taster RBx"* i vraćanje stanja promenljive *tasteri* na *cekanje*. Dodatno, nakon izvršavanja koda u okviru svakog case-a dodaje se naredba *break*, kako bi se dalje izvršavanje koda u okviru switch-case strukture prekinulo u datoj tački. U okviru *default* slučaja vrednost *tasteri* postavlja se na *cekanje*, što podrazumeva ponovnu proveru pritiska tastera.

Listing 2: Slanje informacija o pritisku tastera putem UART-a

```
#include "mcc_generated_files/mcc.h"
#include <stdio.h>

typedef enum {cekanje, taster0, taster1, taster2} stanje;

bool debounce(uint8_t *port, uint8_t pin,uint16_t vreme, bool stanje)
{
    static bool flag = false;
    if((*port&(1<<pin)) == stanje<<pin)
    {
        flag = true;
        while(vreme--)
            __delay_ms(1);
        if((*port&(1<<pin)) == stanje<<pin && flag == true)
            return true;
    }
    return false;
}

stanje provera_pritiska()
{
    stanje tasteri = cekanje;
    static bool flag2 = false;
    static bool flag3 = false;
    static bool flag4 = false;
```

```
26      if(debounce(&PORTB, 0, 5,true) && flag2)
27          {
28              tasteri = taster0;
29              flag2 = false;
30          }
31      if(debounce(&PORTB, 0, 5, false))
32          {
33              flag2 = true;
34          }
35
36      if(debounce(&PORTB, 1, 5,true) && flag3)
37          {
38              tasteri = taster1;
39              flag3 = false;
40          }
41      if(debounce(&PORTB, 1, 5, false))
42          {
43              flag3 = true;
44          }
45      if(debounce(&PORTB, 2, 5,true) && flag4)
46          {
47              tasteri = taster2;
48              flag4 = false;
49          }
50      if(debounce(&PORTB, 2, 5, false))
51          {
52              flag4 = true;
53          }
54      return tasteri;
55  }
56
57  void main(void)
58  {
59      SYSTEM_Initialize();
60      stanje tasteri = cekanje;
61      while (1)
62      {
63          switch(tasteri)
64          {
65              case cekanje :
66                  tasteri = provera_pritiska();
67                  break;
68              case taster0 :
69                  printf("Pritisnut taster RB0\n");
70                  tasteri = cekanje;
71                  break;
72              case taster1 :
73                  printf("Pritisnut taster RB1\n");
74                  tasteri = cekanje;
75                  break;
76              case taster2 :
77                  printf("Pritisnut taster RB2\n");
78                  tasteri = cekanje;
79                  break;
80              default:
81                  tasteri = cekanje;
82          }
83
84      }
85  }
```

Primer 3: *Prijem podataka preko UART-a*

U listingu 3 dat je kod koji za cilj ima promenu stanja na portu D, na osnovu podataka primljenih putem UART-a. Podaci koji pristižu putem prijemne linije

UART modula smeštaju se u promenljivu *a*, tipa *char*. Pre samog očitavanja primljenih podataka vrši se provera prijemne linije, odnosno, proverava se da li na prijemnoj liniji postoje podaci spremni za očitavanje. U ovu svrhu koristi se funkcija *EUSART1_is_rx_ready()*, koja proverava da li je RC1IF bit setovan, odnosno, da li su podaci učitani u RCREG1 registar. Ukoliko su podaci učitani u registar, funkcija *EUSART1_is_rx_ready()* vraća vrednost logičke 1 tj. vrednost RC1IF bita. Upotrebom funkcije *EUSART1_Read* vrši se očitavanje podataka iz RCREG1 registra, a ta vrednost upisuje se u promenljivu a. Ukoliko je primljen karakter '+', vrednost LATD registra se inkrementira i ispisuje se poruka *"Uvecaj"*, praćena karakterom koji naznačava novi red (\n), dok se u slučaju očitavanja '-', vrednost u LATD registru dekrementira i ispisuje se poruka *"Smanji"*, praćena \n karakterom.

Listing 3: Prijem podataka preko UART-a

```
#include "mcc_generated_files/mcc.h"
#include <stdint.h>

void main(void)
{
    SYSTEM_Initialize();

    char a;

    while (1)
    {
        if(EUSART1_is_rx_ready())
        {
            a = EUSART1_Read();

            if(a == '+')
            {
                printf("Uvecaj\n");
                LATD++;

            }
            if(a == '-')
            {
                printf("Smanji\n");
                LATD--;
            }

        }


    }
}
```

Primer 4: *Iscrtavanje podataka u Data Visualizer-u*

U listingu 4 dat je kod za generisanje signala rampe i sinusoide.

Listing 4: Prikaz podataka preko UART-a

```
#include "mcc_generated_files/mcc.h"
#include <math.h>
#include <string.h>

void main(void)
{
    SYSTEM_Initialize();

```

```
 9      uint8_t rampa=0;
10      float sinus;
11      uint8_t dt;
12      uint8_t temp[4];
13      while (1)
14      {
15          rampa++;
16
17          dt++;
18          sinus = 255*sin(2*M_PI/255.0*dt);
19
20          memcpy(temp,&sinus,4);
21
22          EUSART1_Write(0x03);
23          EUSART1_Write(rampa);
24          for(uint8_t i=0; i<4; i++)
25             EUSART1_Write(temp[i]);
26          EUSART1_Write(0xFC);
27      }
28  }
```

Signal rampe realizovan je jednostavnom inkrementacijom osmobitne promenljive rampa. Važno je istaći da kada promenljiva rampa dostigne vrednost 255, sledeći inkrement vrednost resetuje na 0 (zbog prekoračenja *uint8_t* tipa). Kako bi se generisala sinusoida u 255 tačaka po periodi neophodno je napraviti horizontalnu inkrementaciju od po $\frac{360^\circ}{255} \approx 1.41^\circ$ odnosno u radijanima $\frac{2\pi}{255}$. Zbog toga se u izrazu za određivanje sinusoidalnog signala funkciji sin prosleđuje $2\frac{2\pi}{255}dt$ gde se promenljiva *dt* povećava sa svakom iteracijom petlje. Ovako je obezbeđeno da kada dođe do prekoračenja promenljive *dt* bude i generisana jedna perioda signala. Važno je istaći da bi se koristila funkcija sin neophodno je uključiti biblioteku *math.h*. Da bi vrednost amplitude sinusoidalnog signala bila samerljiva sa aplitudom rampe, amplituda je pomnožena sa 255.

Kako bi se signali vizualizovali neophodno je ispoštovati standard za slanje podatka po protokolu prikazanom na slici 10.

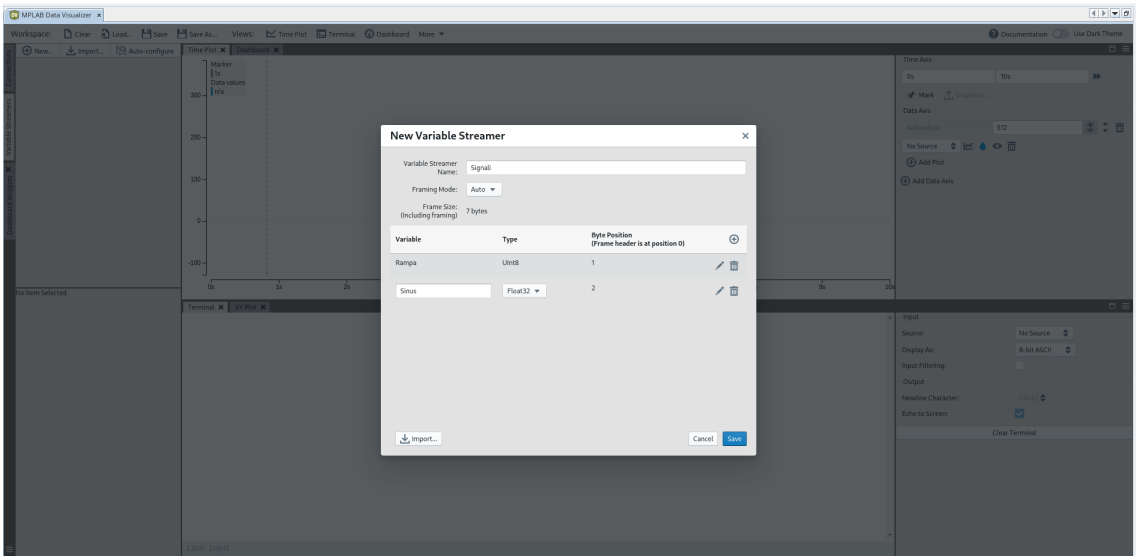
0x03	rampa	sinus 1. bajt	sinus 2. bajt	sinus 3. bajt	sinus 4. bajt	0xFC

Slika 10: Struktura paketa Data Visualizer

Kako bi se adekvatno formirao paket neophodno je poslati na početku paketa start bajt (*0x03*) i na kraju paketa stop bajt (*0xFC*), dok se između prosleđuju podaci. U ovom primeru prvo će se poslati osmobitna promenljiva rampa, a nakon nje decimalna (*float*) vrednost sinusoide. Važno je istaći da tip *float* zauzima četiri bajta. Ako bi se za slanje podataka koristila funkcija printf, ona bi ovu decimalnu vrednost pretvorila u string, a zatim poslala. Npr. ako je vrednost 123.2 onda bi se poslalo pet bajta (za svaki karakter po jedan bajt), a u slučaju da je 1.2 poslala bi se tri bajta. Pošto je protokolom specificirano da se moraju poslati četiri bajta za *float* vrednost neophodno je preneti svaki od bajta koji sačinjavaju decimalni tip ponaosob. Takođe, dodatni problem je što *float* tip nema mogućnost šiftovanja bita jer se podaci skladište spram standarda IEEE 754 i ne čuvaju se na isti način na koji se čuva celobrojna vrednost. Zbog toga se pravi niz *temp* koji je veličine četiri

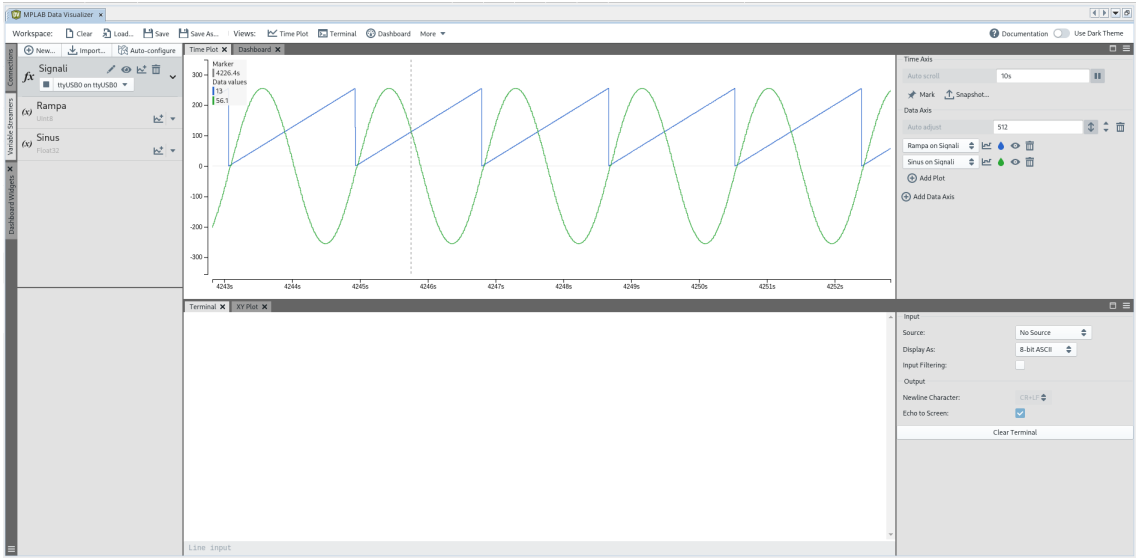
bajta u koji će se smestiti float promenljiva. Kako bi se to izvršilo iskorišćenja je funkcija *memcpy* iz biblioteke *string.h* za kopiranje bloka memorije u drugi. Funkcija *memcpy* očekuje kao prvi parametar pokazivač na prvi element bloka memorije u koji se žele kopirati vrednosti, kao drugi parametar pokazivač na prvi element bloka memorije iz koga se žele kopirati podaci i na kraju se specificira koliko bajta se želi kopirati. Sada je moguće pristupiti svakom od četiri bajta koji sačinjavaju *float* tip. Tako formatirani podaci se šalju korišćenjem funkcije *EUSART1_Write* gde se prosleđuje prvo podaci o rampi, a zatim se u for petlji šalju četiri bajta.

Nakon slanja podataka neophodno je u alatu *Data Visualizer (DV)* podesiti odgovarajući *Variable streamer*. Nakon pravljenja novog *Variable streamer-a* dobija se prozor prikazan na slici 11.



Slika 11: Podešavanje Variable Streamer-a u DV

Ime *streamer-a* je Signali, a imena promenljivih prate vrednosti realizovane u firmveru (prvi bajt predstavlja promenljivu Rampa i veličine je osam bita i narednа četiri bajta predstavljaju promenljivu Sinus koja je tipa *float32* odnosno *float* veličine 32 bita). Nakon toga neophodno je sačuvati *streamer* i pokrenuti komunikaciju sa pločom. Kao Source odabrati odgovarajući port i klikom na + pored Sinusa i Rampe dodati ih na polje predviđeno za prikaz signala. Nakon pokretanja prikaza signala dobijaju se signali prikazani na slici 12.



Slika 12: Prikaz signala u DV