Rīgas Tehniska Universitāte

Datorvadības, automātikas un datortehnikas institūts

Datoru tīklu un sistēmas tehnoloģijas katedra

Mikroprocesoru tehnika
Laboratorijas darbs Nr. 6

Asist. R. Taranovs
Profesors V.Zagurskis
Students Vitālijs Hodiko
3.kurss 1.grupa

Uzdevums

1. Izmantojot Atmega128 aprakstu nokonfigurēt UART moduli datu sūtīšanai un saņemšanai (Tx un Rx). Datu freima formāts 8N1;
2. Pārsūtīt no CharonII uz datoru frāzi, kas pieprasa ievadīt skrejošās gaismas aizkavi;
3. Saņemt no datora kādu skaitli (vēlams arī divciparu skaitļus), kas veidotu skrejošās gaismas aizkavi.

Teorētiskais apraksts

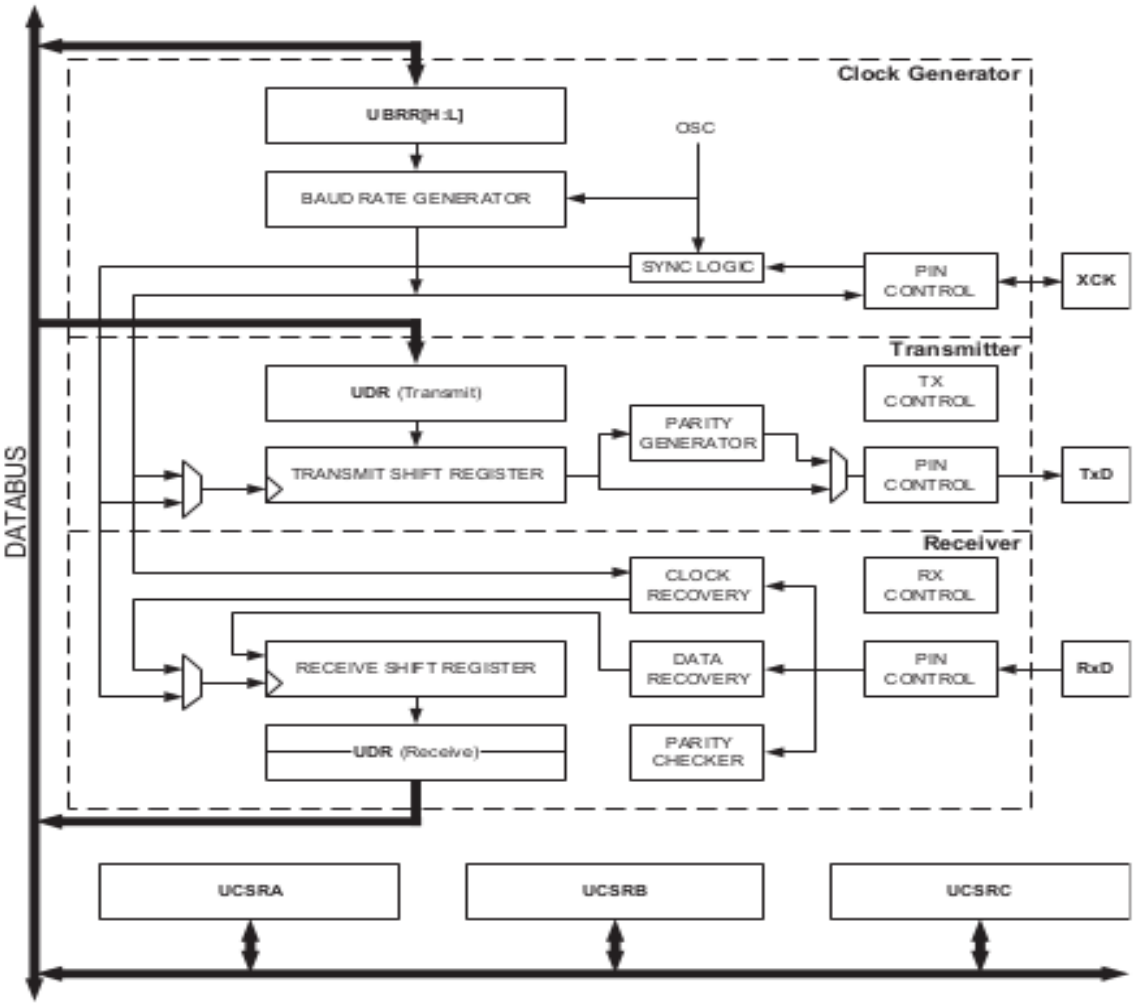
USART modulis

USART (Universal Synchronous asynchronous receiver/transmitter) modulis – ir skaitļošanas tehnikas daļa, kas pārveido datus no paralēlas formas seriālajā. USART parasti tiek izmantots kopā ar citiem komunikācijas standartiem, piemēram EIA RS- 232.

Parasti USART ir atsevišķs vai kā daļa no mikroshēmas, kas tiek izmantots seriāla pārraidē starp datoru un perifērijas ierīcēm caur seriālu portu (RS232). ATmega 128 mikrokontrollerī arī ir iebūvēts USART modulis, kurš aktivizē sazināšanas iespēju starp mikrokontrolleri un, piemēram, datoru. Patiesība to tur ir veseli divi USART1:0.

Tā galvenas īpašības ir:

1. Pilnduplekss
2. asinhronais/sinhronais režīms
3. Vedēja vai slave režīms
4. augstas izšķirtspējas pārraides ātruma ģenerators
5. pārraida vienlaicīgi līdz 9 bitiem un 1 vai 2 stop biti
6. paritātes ģenerēšana un pārbaude atbalstīta aparatūra
7. datu pārsniegšanas noteikšana
8. freima kļūdu noteikšana
9. trokšņa filtrēšana
10. daudz procesoru komunikācijas režīms
11. dubulta ātruma asinhronais režīms
12. trīs atsevišķi pārtraukumi



Attēls 1: USART modulis Atmega128 mikrokontrollerī

CPU pieejami I/O izejas pāradīti ar trekno līniju. Ar svītrotam līnijām tiek atdalītas trīs galvenas USART daļas: takts ģenerators, datu sūtītājs un datu saņēmējs. Vadības reģistri ir kopējie.

Takts ģenerācijas loģika sastāv no ārējas takts sinhronizācijas loģikas slave režīma un pārraides ātruma ģenerācijas. Pārraides pulksteņa(XCK) pins tiek izmantots tikai sinhrona pārraides režīma.

Raidītājs sastāv no viena rakstīšanas bufera, seriāla pārbīdes reģistra, paritātes ģeneratora un kontroles loģikas, lai apstrādātu dažādus freima formātus. Rakstīšanas buferis ļauj pārraidīt datus bez pārtraukuma starp freimiem.

Saņēmējs ir pati sarežģītāka USART moduļa daļa pulksteņa un datu atjaunināšanas bloku dēļ. Tie tiek izmantoti asinhronu datu saņemšanai. Vēl saņēmējs ietver sevi paritātes pārbaudi, kontroles loģiku, pārbīdes reģistru un divu līmeņu saņemšanas buferi(UDR). Uztvērējs atbalsta tādus pašus freima formātus ka raidītājs, ka arī spēj noteikt freima kļūdas, datu pārplūdi un paritātes kļūdas.

AVR USART savienojamībā ar AVR UART

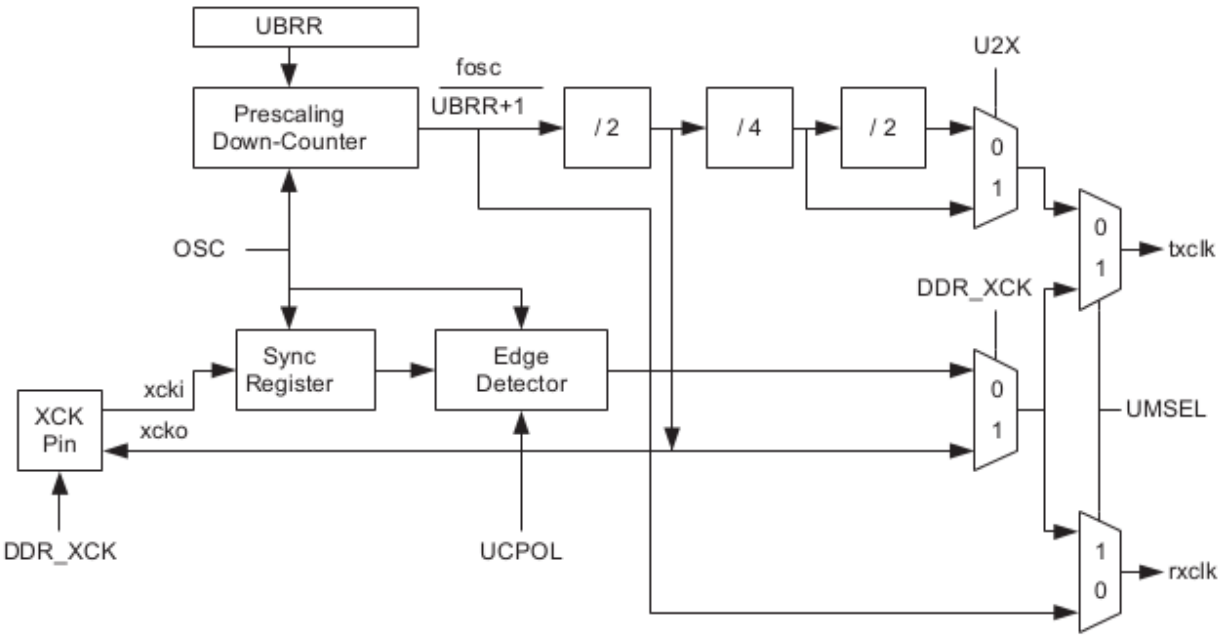
Tie ir pilnība savienojami, bet USART's ir pilnveidots:

Ir pievienots otrais bufera reģistrs, kas kopa darbojas ka riņķveida FIFO buferis. Tāpēc UDR jānolasa vienreiz par katru ienākošo freimu. Ne mazāk svarīgs ir tas fakts ka kļūdu karogi(FE-freima kļūda, DOR – datu pārpilde) un 9ais datu bits(RXB8) ir bufereti kopā ar datiem saņemšanas buferī. Tāpēc statusa bitu jānolasa pirms nolasa UDR reģistru. Citādi kļūdas status bus zaudēts, jo būs zaudēts bufera stāvoklis.

Saņēmēja pārbīdes reģistrs var darboties ka trešais bufera līmenis. Tas tika paveikts, ļaujot saņemtiem datiem palikt seriāla pārbīdes reģistrā, ja bufera reģistri ir pilni, kamēr nākošais starta bits būs saņemts. Tāpēc USART ir izturīgāks pret datu pārpildīšanos (DOR) stāvokļiem.

Takts ģenerācija

Takts ģenerācijas loģika ģenerē pamata pulkstenis raidītājam un saņēmējam. USART modulis atbalsta 4 pulksteņa režīmus: Parasto Asinhrono, 2x Asinhrono, Vedēja Sinhrono, Slave Sinhrono. UMSEL bits USART kontroles un statusa reģistrā C(UCSRC) nosaka asinhrono vai sinhrono režīmu. 2X asinhronais režīms ir kontrolēts ar U2X bitu UCSRA reģistrā. Sinhronajā režīmā XCK DDR reģistrs nosaka vai pulksteņa avots ir ārējais(slave) vai iekšējs(master).



Attēls 2: Takts ģenerācijas loģika

txclk Sūtītāja kristāls (Internal Signal)

rxclk Saņēmēja kristāls (Internal Signal)

xcki Ieeja no XCK pina(internal Signal, slave)

xcko Pulksteņa izeja uz XCK pina (Internal Signal, master)

fosc XTAL pina frekvence

Iekšēja Takts Ģenerēšana – Pārraides ātruma ģenerēšana

Izmanto sinhrona un asinhrona vedēja režīmos.

USART'a raidīšanas ātruma reģistrs(UBRR) un samazināšanas skaitītājs ir savienoti, lai kopa veidotu programmējamo priekšdalītāju vai pārraides ātruma ģeneratoru. Skaitītājs, kas darbojas uz sistēmas pulksteņa frekvences, ielāde UBRR vērtību katru reizi sasniedzot nulli vai, kad UBRR vērtība ir ierakstīta. Pulkstenis tiek ģenerētas katru reizi sasniedzot nulli un tas ir pārraides ātruma ģeneratora izeja (= fosc/(UBRR+1)). Raidītājs, atkarība no režīma, sadala šo vērtību ar 2, 8, 16. Bet saņēmēja pulkstenis un datu atgūšanas bloki izmanto šo vērtību pa tiešo. Taču atgūšanas bloka stāvokļa mehānisms izmanto 2, 8, 16 stāvokļus, kas tiek uzstādīti atkarīgi no režīma ar UMSEL, U2X un DDR_XCK bitiem.

Režīms	Pārraides ātrums	UBRR vērtība
Asinhronais parastais(U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asinhronais 2x (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Sinhronais vedēja	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Tabula 1: Vienādojumi, lai aprēķinātu UBRR reģistra uzstādījumus

BAUD – pārraides ātrums bits sekundē bps

UBRR – [0 - 4095]

Ārējas pulkstenis

Izmanto tikai slave režīmā.

Ārēja pulksteņa ieeja no XCK pina tiek salīdzināta ar sinhronizācijas buferi, lai minimizētu meta-stabilitātes varbūtību. Izeja no bufera iet caur frontes detektoru pirms tas var būt izmantots ar saņēmēju vai raidītāju. Šīs process aizņem 2 taktis un tāpēc maksimāla ārēja frekvence ir ierobežota:

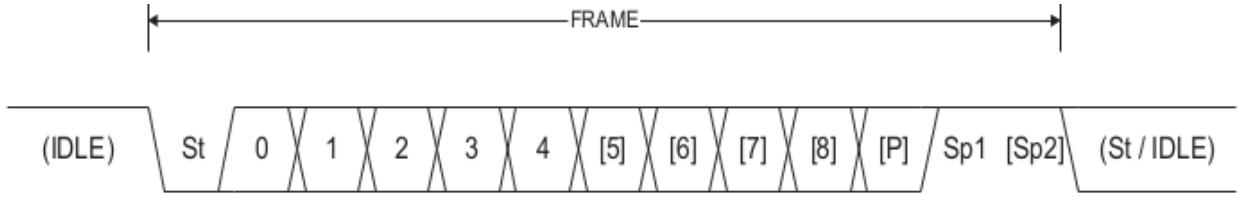
$$f_{XCK} < \frac{f_{OSC}}{4}$$

NB Ģenerātora frekvence ir atkarīga no sistēmas pulksteņa stabilitātes. Tāpēc ir ieteikts ievest robežas, lai izvairīties no datu zaudēšanas frekvenču dažādības dēļ.

Freima formāti

Viens kadrs ir noteikts kā viens datu bitu simbols ar sinhronizācijas bitiem(Starta/stop biti) un pēc izvēles ar paritātes bitiem kļūdu novēršanai. USART pieņem 30 kadru formātus.

- 1 starta bits
- 5, 6, 7, 8 vai 9 datu biti
- nav, pāra vai nepāra paritātes biti
- 1 vai 2 stop biti



Attēls 3: Iespējamie kadra formāti; iekavās biti pēc izvēles

St – starta bits, vienmēr LOW

n – 0..8 datu biti; min 5, max 9

P – paritātes biti

Sp – stop bits

IDLE – nav pārraides komunikācijas līnijā(Rx vai TX); jābūt HIGH

Kadra formāts tiek uzstādīts ar UCSZ2:0, UPM1:0 un USBS bitiem UCSRB un UCSRC reģistros. Rx un Tx izmanto vienādus uzstādījumus. Šo bitu izmaiņa bojas komunikāciju starp Tx un Rx. Rx ignorē otro stop bitu. Tāpēc kadra kļūda(FE) būs noteikta tikai ja pirmais stop bits būs nullē.

Paritātes bits ir noteikts izpildot visiem datu bitiem XOR operāciju:

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

$\mathbf{P_{even}}$ – pāra paritāte

$\mathbf{P_{odd}}$ – nepāra paritāte

$\mathbf{d_n}$ – n-tais datu bits

USART inicializācija

Inicializācijas process parasti sastāv no pārraides ātruma, kadra formāta uzstādīšanas un Rx vai Tx ieslēgšanas atkarīgi no vajadzības. Pārtraukumu vadītam USART'am, pirms inicializācijas jāatslēdz globālus pārtraukumus.

Datu sūtīšana(Tx)

Raidītājs tiek ieslēgts uzstādot TXEN(Transmit Enable) bitu UCSRB reģistrā. Sinhronajā režīma kristāls uz XCK pina būs izmantots ka pārraidīšanas pulkstenis(overriden).

Datu pārraidīšana tiek inicializēta ielādējot datus pārraidīšanas buferī(Ja UDRE karogs nav iestatīts, tad ierakstītie dati bus ignorēti). Tālāk dati būs pārsūtīti uz pārbīdes reģistru, ja tas atrodas tukšgaitas režīma vai uzreiz pēc iepriekšēja nosūtīta kadra pēdēja stop bita. Kad dati tika ielādēti, tas nosutis kadru ar nosacīto pārsūtīšanas ātrumu, U2X bitu vai XCK atkarība no režīma(Secīgi pa TxDn pinu).

Ja izmanto 9 bitu datus, tad 9 bitu jāieraksta TXB8 bitā UCSRB reģistrā pirms zemākais bits būs ierakstīts UDR reģistrā. To var izmantot, lai norādītu ka tiek raidīts adreses kadrs daudz procesoru komunikācijas režīma vai citiem nolūkiem, e.g. sinhronizācija.

Raidītāja karodziņi un pārtraukumi

Ir divi stāvokļa karogi:

- USART datu reģistrs tukšs(UDRE)
- Pārraide izpildīta(TXC)
    - Tas ir noderīgs pusdupleksa komunikācijas interfeisos(e.g. RS485), kur pārraidīšanas ierīcei vajag ieiet saņēmēja režīma un atbrīvot komunikācijas joslu uzreiz pēc nosūtīšanas beigām.

Atslēdzot Raidītāju(dzēšot TXEN bitu) tas tiks atslēgts tikai, kad visi tekoši un gaidāmie sūtījumi bus pabeigti, i.e., kad Raidīšanas Nobīdes Reģistrs un Buferis būs tukši. TxD pins būs brīvs no tā funkcijas.

USART datu saņēmējs

- Sastāv no divu līmeņu FIFO.
- Tiek ieslēgts uzstādot Ieslēgt Saņēmēju(RXEN). Pārraidīšanas ātrums, režīms un kadra formāts jābūt uzstādīts pirms var sākt saņemšanu. Sinhronajā režīma XCK pina pulkstenis bus izmantotas ka pārraidīšanas pulkstenis.
- Saņēmējs sak datu saņemšanu uzreiz kad saņem derīgu starta bitu. Katrs nākamais bits būs apstrādāts(sampled) ar noteikto pārraidīšanas ātrumu vai XCK kristālu un nobīdīts Saņēmēja Nobīdes Reģistrā kamēr pirmais stop bits nebūs saņemts. Otrais tiek ignorēts. Kad pirmais stop bits ir saņemts, i.e., nobīdes reģistra ir vesels kadrs, tas tiek pārvietots uz saņemšanas buferi, ko var nolasīt nolasot UDR I/O reģistru.
- Ja tiek izmantots 9ais bits, tad to jānolasa no RXB8 bita UCSRB reģistrā pirms nolasot UDR kadra zemākos bitus. Šīs nosacījums attiecas arī uz FE, DOR, UPE statusa karogiem. Nolasa statusu no UCSRA un pēc tam UDR. Nolasot UDR I/O reģistru mainīs saņēmēja FIFO bufera stāvokli un rezultāta arī TXB8, FE, DOR un UPE bitus, kas glabājas FIFO.

Ir tikai viens stāvokļa karogs:

- Saņemšana Pabeigta(RXC)

Ir trīs Saņemšanas Kļūdu karogi:

- Kadra Kļūda(FE),
- Datu Pārpildīšanas(Data OverRun), un
- Paritātes Kļūda(UPE).

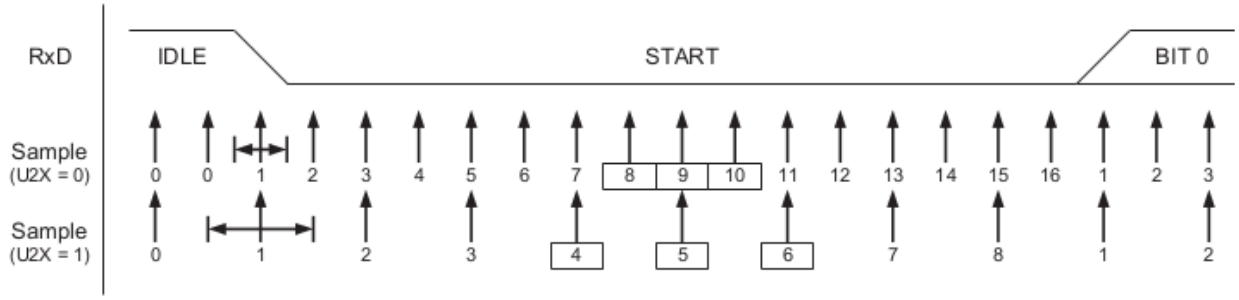
Tie visi glabājas UCSRA reģistrā. **NB** Nevar tikt mainīti ar programmatūru, taču tos jānotīra rakstot UCSRA reģistra, lai nodrošinātu savienojamību ar nākotnes USART implementācijam.

Pretstatā Raidītājam, tas tiek atslēgts uzreiz. Rx pins bus atbrīvots no tā funkcijām un saņemšanas buferis FIFO būs notīrīts.

NB FIFO mainīs savu stāvokļi kad vien saņēmēja buferis tiek izmantots, tapēc neizmanto read-modify-write instrukcijas(SBI un CBI) pret šo vietu. Esi uzmanīgs izmantojot bitu pārbaudes instrukcijas(SBIC un SBIS), tā kā tie arī mainīs FIFO stāvokli.

Asinhronais kristāla atgūšana(Asynchronous Clock Recovery)

Pulksteņa atgūšanas loģika sinhronizē iekšējo kristālu ar sērijveida ienākošiem kadriem. Apstrādes ātrums ir 16x pārraides ātruma parastajā režīma un 8x 2x ātruma režīma. Horizontāla bulta attēlo sinhronizācijas maiņu apstrādes procesa rezultāta. Ņemiet vēra lielu laika variāciju 2x ātruma režīma. Paraugi apzīmēti ar nullēm ir apstrādāti dīkstāves laikā(i.e. Nebija komunikācijas).



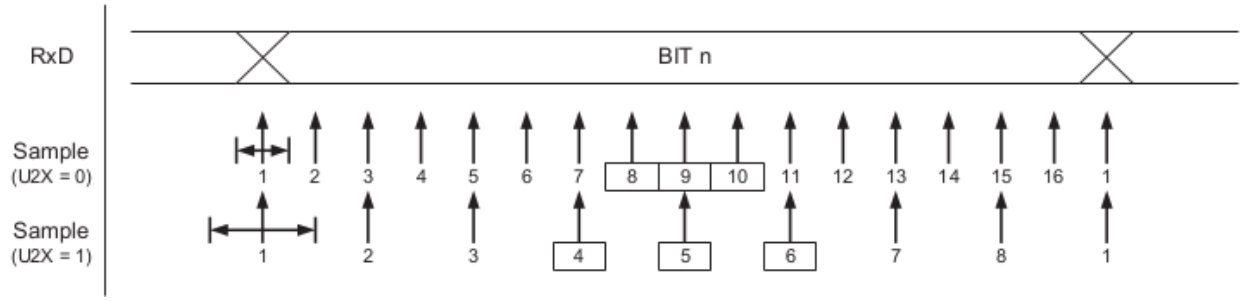
Attēls 4: Starta bita apstrādes procesa ilustrācija

Kad atgūšanas loģika noteic lejupslīdošo fronti uz Rx līnijas, starta bita noteikšana tiek uzsākta.

- Paraugs 1 – atskaites paraugs
- 8, 9, 10(parastajā režīma) ai 4, 5, 6(2x režīma) biti tiek izmantoti, lai noteiktu vai saņemts derīgs starta bits. Ja vismaz divi no tiem ir 1kā(vairākums uzvar), starta bits tiek uzskatīts par troksni un noraidīts. Un saņēmējs sak gaidīt nākamo lejupslīdošo fronti. Kad derīgs starta bits tiek saņemts kristāla atgūšanas loģika ir sinhronizēta un datu atgūšana var sākties.

Asinhrona datu atgūšana

Datu atgūšanas bloka izmanto stāvokļu automātu ar 16 stāvokļiem uz katru bitu parastajā režīma un 8 2x režīma.



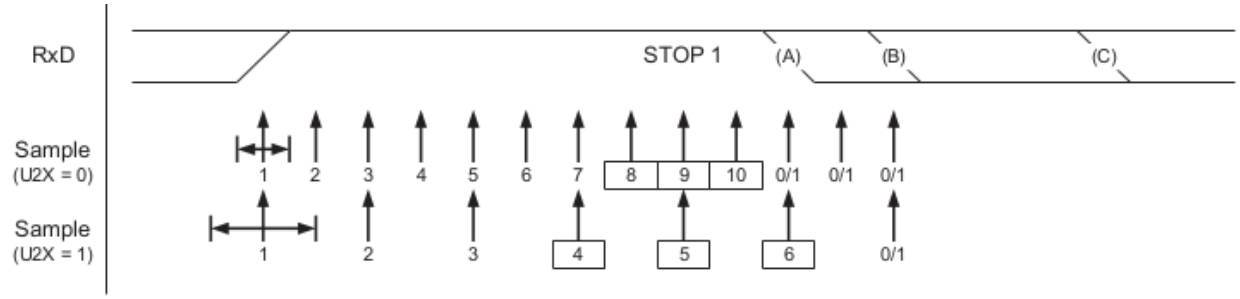
Attēls 5: Paritātes un Datu bita apstrāde

Katram paraugam tiek piešķirts skaitlis kas ir vienāds atgūta bloka stāvoklim.

Loģiskā līmeņi lēmums tiek pieņemts, pildot vairākuma balsojumu trim centrālam paraugu vērtībām vienam bitam(grafika tie ir attēloti ar parauga numuru iekš kvadrāta). Vairākuma balsojums notiek šādi:

- Ja vismaz divi no tiem ir HIGH, saņemtais bits tiek uzskatīts par 1 un otrādi.
- Tas darbojas ka ienākoša signāla uz Rx pina zemas caurlaides filtrs

  Atgūšanas process tiek turpināts kamēr viss kadrs bus saņemts(ieskaitot pirmo stop bitu).



Attēls 6: Stop un nākama Starta bita apstrāde

Tapāt tiek apstrādāts Stop bits. Ja tas tiek noteikts ka 0, tad FE būs set.

Jauna lejupslīdoša fronte, kas norāda uz nākama kadra starta bitu var būt uzreiz pēc pēdēja bita, kas tika izmantots aptaujā:

- (a) – var būt parastaja režīma
- (b) – tas tiek aizturēts līdz šejienei
- (c) – pilna garuma stop bits.

Agra starta bita noteikšana ietekmē saņēmēja uztvērēja darbības diapazonu.

Asinhronais darbības diapazons

Ir atkarīgs no kļūdas starp saņemto bitu ātrumu un iekšējo pārraides ātrumu. Ja raidītais sūta kadrus pie pārāk ātra/lēna bitu ātruma, vai iekšēji ģenerēta saņēmēja pārraidīšanas ātrumam nav līdzīgas pamata frekvences, saņēmējs nevarēs sinhronizēt kadrus ar starta bitu.

$$R_{slow}=\frac{(D+1)S}{S-1+D\cdot S+S_F};R_{fast}=\frac{(D+2)S}{(D+1)S+S_M}$$

Attēls 7: Formulas, lai aprēķināt atkarību

- D – datu izmērs un paritāte(D=5..10)
- S – apstrādes uz bitu(parasti S = 16; 2x režīma S = 8)
- S_F – pirmā parauga numurs, kas tika izmantots vairākuma aptauja(8 – parasti, 4 – 2x režīma)
- S_M - vidēja parauga numurs, kas tika izmantots vairākuma aptauja(9 – parasti, 5 – 2x režīma)
- R_{SLOW} – lēnākais ienākošo bitu ātrums ko var saņemt ar doto pārraides ātrumu
- R_{FAST} – ātrako ienākošo bitu ātrums ko var saņemt ar doto pārraides ātrumu

Ir divi iespējamie pārraides ātruma kļūdas avoti:

- XTAL kristālam vienmēr ir neliela nestabilitāte visa barošanas sprieguma un temperatūru diapazonā
- pārraides ātruma ģenerators ne vienmēr var precīzi sadalīt sistēmas frekvenci, tāpēc šāda gadījumā vajag izmantot UBRR vērtību kas dod pieļaujami zemu kļūdu.

Daudz procesoru komunikācijas režīms

Daudz Procesoru Komunikācijas Režīms(MPCM bits UCSRA reģistrā) ieslēdz saņemto kadru filtrēšanu. Kadri, kas nesatur adrešu informāciju būs ignorēti. Tas efektīvi samazina ieejas kadru skaitu kurus jāapstrādā CPU sistēma ar vairākiem MCU, kas komunicē izmantojot kopējo datu kopni. Raidītājs nav ietekmēts šajā režīma tieši, bet to vajag izmantot citādāk tāda sistēma.

Ja raidītais ir iestatīts uz 5..8 datu bitiem, tad pirmais stop bits nosaka vai kadra ir datu vai adrešu informācija. Ja raidītājs izmanto visus 9 datu bitus, tad ar devīto bita palīdzību identificē datu un adrešu kadrus. Kad kadra tipa bits(RXB8/STOP bits) ir 1 – adrešu kadrs un otrādi(Slave MCU būt iestatītam uz to pašu kadra formātu).

Šīs režīms nodrošina vairāku slave MCU datu saņemšanu no viena vedēja MCU. Tas tiek nodrošināts, pirmkārt, ar adreses dekodēšanu, kas nosaka kurš MCU tiek adresēts. Tas, kurš tika adresēts, saņems datus ka parasti, kamēr citi gaidīs nākamo adreses kadru.

Šādu procedūru jāizmanto daudz procesoru komunikācijas režīma:

- visiem MCU jābūt MPCM režīmā
- Vedējs nosūta adreses kadru un visi slave saņems un izlasīs to.
    - Slave MCU RXC karogs būs uzstādīts ka parasti
    - Katrs slave nolasīs UDR reģistru un noskaidros vai tika izvēlēts viņš. Ja ta it, viņš notīrīs MPCM bitu. Citādi gaidīs nākamo adreses kadru.
- Adresēts MCU saņems visus datu kadrus līdz jauns adreses kadrs būs saņemts.

- Kad pēdējais datu kadrs būs saņemts, adresēts MCU ieslēgs MPCM un gaidīs nākamo adreses kadru. Un tālāk viss atkārtojas no 2 soļa.

Kadra izmēru mazāku par 9 bitiem var izmantot, taču tas nav praktiski, jo saņēmējam jāpārslēdzas no n un n+1 kadra formātiem. Tas apgrūtina pilndupleksa komunikāciju, jo raidītājs un saņēmējs izmanto vienādus kadra formātus. Ja tiek izmantots 5..8 bitu formāts, raidītājam jāizmanto divi stop biti, jo pirmais norāda uz kadra tipu.

NB Neizmanto read-modify-write instrukcijas(SBI un CBI), lai uzstādīt/notīrīt MPCM bitu. MPCM atrodas tajā paša I/O vietā ka TXC karogs un to ar nejauši notīrīt.

Reģistri

UDRn	RXBn(Lasīt)
	TXBn(Rakstīt)
Def: 0x00	R/W

Tabula 2: USART I/O Datu Reģistrs

USART'a Raidītāja un Saņēmēja Datu Buferiem ir kopīga I/O adrese, kas norāda uz USART Datu Reģistru jeb UDR. TXB(var būt rakstīts tikai kad UDREn karogs UCSRA reģistra ir set, citādi dati tiek ignorēti) būs galamērķis datiem ierakstītiem UDR. Nolasot UDR būs atgriezts RXB saturs. Neizmantoti biti(e.g. 5 bitu datu formāta) bus ignorēti ar Raidītāju un dzēsti ar Saņēmēju.

UCSRnA	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
Def:0x20	R	R/W	R				R/W	R/W

Table 1: USART Statusa un Kontroles Reģistrs A

RXC(USART Receive Complete) – ir set kad satur nenolasītos datus. Var izmantot, lai ģenerētu Saņemšana Izpildīta pārtraukumu(RXCIE). Ja saņēmēju atslēdz saņemšanas buferis būs notīrīts un tādējādi RXC kļūs 0.

TXC(USART Transmit Complete) – ir set, kad kadrs no Pārraidīšanas Nobīdes reģistra ir pilnība nobīdīts un jaunu datu pārraidīšanas buferi nav. Tiek notīrīts, kad attiecīgs pārtraukums ir izsaukts, vai alternatīvi to var notīrīt ierakstot 1 tajā bita. Var izmantot Pārraidīšana Pabeigta pārtraukuma ģenerēšanai(TXCIE).

UDRE(USART Data Register Empty)- norāda vai raidītajā buferis ir gatavs saņemt jaunus datus. Tiek uzstādīts, kad buferis ir tukšs un notīrīts, kad ir dati ko pārraidīt un tie vēl nav pārvietoti uz pārbīdes reģistru. Var izmantot Datu Reģistrs ir Tukšs pārtraukuma ģenerēšanai(UDRIE). Tiek uzstādīts pēc atiestatīšanas(reset). **NB** Savietojamības pēc ar citam ierīcēm, vienmēr uzstādi tajā nulli, kad raksti UCSRA reģistrā.

FEn(Frame Error) - norāda uz pirmā stop bita stāvokļi nākamajā kadrā, kas glabājas saņemšanas buferī. I.e. Ir nulle kad nākama kadra pirmais stop bits ir viens. Var tikt izmantots, lai noteiktu ārpus sinhronizācijas apstākļus(out-of-sync conditions), pārtraukumu nosacījumus un protokolu vadību. Nav ietekmēts ar 2 stop bitu režīmu, tā kā saņēmējs ignorē visu izņemot pirmo stop bitu. **NB** Ir derīgs līdz nolasīs UDRn(RXBn). Vienmēr notīri to rakstot UCSRA reģistrā.

Var izmantot, lai noteiktu

- sinhronizācijas kļūdu

- protokola kļūdu
- sakaru zudumu

DORn(Data OverRun) - norāda uz datu zudumiem saņēmēja bufera aizpildīta stāvokļa esamības dēļ. Datu sadursme notiek, kad datu buferis ir pilns(2 kadri), tas ir jaunais kadrs, kurš gaidā saņēmēja nobīdes reģistra un jaunais starta bits ir noteikts. Ja tas ir uzstādīts, tas nozīme ka viens vai vairāki kadri tika nozaudēti starp pēdēja un nākama kadra nolasīšanas no UDR reģistra. **NB** Ir derīgs līdz nolasīs UDRn(RXBn). Vienmēr notīri to rakstot UCSRA reģistrā.

UPEn(Parity Error) - norāda ka nākamais kadrs saņemšanas buferī satur paritātes kļūdu. Ja pārbaude nav ieslēgta(UPM1:0) vienmēr tiek nolasīts ka nulle. **NB** Ir derīgs līdz nolasīs UDRn(RXBn). Vienmēr notīri to rakstot UCSRA reģistrā.

U2Xn(Double the USART Transmission Speed) - Ir spēka tikai asinhrona režīmā. Samazina pārraides ātruma dalītāju ar 2, kas efektīvi dubulto asinhronas komunikācijas ātrumu. **NB** Saņēmējs izmantos divreiz mazāk paraugu(8) datu salīdzināšanai un pulksteņa atjaunināšanai. Tāpēc šajā režīma jāuzstāda kārtīgāk pārraides ātrumu un sistēmas pulkstenis. Raidītāju tas neietekmē.

MPCMn(Multi-Processor Communication Mode) - Raidītājs nav ietekmēts ar šo bitu. Iestādot visi iesakošie kadri, kas nesatur adrešu informāciju tiks ignorēti.

UCSRnB	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
Def:0x00	R/W						R	R/W

Table 2: USART Statusa un Kontroles Reģistrs B

RXCIEn(RX Complete Interrupt Enable) - atļauj pārtraukumu uz RXC karoga(I-bit, RXC ir set). Pārtraukums būs izsaukts ja RXC ir 1ks. Lai notīrītu RXC bitu jānolasa datus no UDR reģistrā, citādi pārtraukums atkal izsauksies uzreiz pēc pabeigšanas.

TXCIE(TX Complete Interrupt Enable) - atļauj pārtraukumu uz TXC karoga(I-bit, TXC ir set). Pārtraukums tiks izsaukts ja TXC karogs būs 1ka. Pēc izsaukuma tas automātiski notīra TXC karogu.

UDRIEn(USART Data Register Empty Interrupt Enable) - atļauj pārtraukumu uz UDRE karoga(I-bit, UDRE ir set). Pārtraukums būs izsaukts kamēr UDRE ir set. Tas tiek notīrīts rakstot UDR. Tāpēc, ja ir izmantota uz pārtraukumiem bāzēta datu pārraidīšana, UDRE pārtraukumā jāieraksta dati UDR reģistra, lai notīrītu UDRE, i.e. Lai izietu no tekoša pārtraukuma, citādi tas uzreiz ieies tajā pēc pabeigšanas.

RXENn(Receiver Enable) - ieslēdz saņēmēju un pārņem RxD pina funkcijas. Atslēdzot FEn, DORn, UPEn biti paliek nederīgi.

TXENn(Transmitter Enable) - ieslēdz raidītāju un pārņem TxD pina funkcijas. Atslēdzot to tas atslēgsies tikai pēc tam, kad izejoši un gaidāmie dati būs nosūtīti(i.e. Raidītāja Pārbīdes Reģistrs un TXBn bus tukši).

UCSZn2(Character Size) - kombinācijā ar UCSZn1:0 bitiem uzstāda datu bitu izmēru kadrā

RXB8n(Receive Data Bit 8) - ir 9ais saņemtais bits 9 bitu datu formāta. Jālasa pirms nolasot UDRn.

TXB8n(Transmit Data Bit 8) - ir 9ais bits sūtīšanai 9 bitu datu formāta. Jāieraksta pirms rakstot UDRn.

UCSRnC	-	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn
Def:0x06	R/W							

Table 3: USART Statusa un Kontroles Reģistrs C

NB Šīs reģistrs nav pieejams Atmega103 savienojamības režīma

7 bits(Rezervēts) – Rezervēts lietošanai nākotnē. Savietojamībai ar nākotnes ierīcēm šis bits jābūt notīrīts.

UMSELn(USART Mode Select)

		XCK pins	
		Master	Slave
0	Asinhronais režīms		
1	Sinhronais režīms	Clock o/p	Clock i/p

Tabula 3: USART režīms

UPMn1:0(Parity Mode) - ieslēdz un nosaka paritātes ģenerēšanu un pārbaudi. Raidītājs automātiski ģenerēs un nosūtīs nosūtīto datu paritāti. Saņēmējs aprēķina kadra bitu paritāti un salīdzina ar saņemto paritātes bitu. Rezultāts tiek saglabās saņemšanas buferi kopā ar saņemtiem datiem un stop bitiem. Atšķirības gadījuma UPEn karogs būs set.

00	Atslēgts
01	Rezervēts
10	Pāra paritāte
11	Nepāra paritāte

Tabula 4: Paritātes režīmi

USBSn(Stop Bit Select)

0	1 stop bits
1	2 stop biti

Tabula 5: Stop bitu skaita uzstādīšana

UCSZn1:0(Character Size) - kombinācijā ar UCSZn2 bitu uzstāda datu bitu izmēru kadrā

000	5 -biti
001	6 -biti
010	7 -biti
011	8 -biti
100	Rezervēts
101	Rezervēts
110	Rezervēts
111	9 -biti

Tabula 6: Kadra datu bitu skaits

UCPOLn(Clock Polarity) - izmanto tikai sinhronajā režīma. Nosaka uz kura XCK frontes datus salīdzinās un kurā mainīs. e.g., kad UCPOL = 0 dati būs izmainīti augoša frontē un salīdzināti krītošajā un otrādi.

UCPOLn	Pārraidītie dati mainījās(TxDn pina izeja)	Saņemti dati apstrādāti(RxDn pina ieeja)
0	Augoša XCKn fronte	Krītoša XCKn fronte
1	Krītoša XCKn fronte	Augoša XCKn fronte

Tabula 7: UCPOL uzstādījumi

UBRRnH	-	-	-	-	UBRRn[11:8]
UBRRnL	UBRRn[7:0]				
Def: 0x00	R				
Def: 0x00	R/W				

Table 4: USART pārraides ātruma reģistrs H:L

NB UBRRnH nav pieejams ATmega103 savietojamības režīma

15:12 Biti(Rezervēts) – Rezervēts lietošanai nākotnē. Savietojamībai ar nākotnes ierīcēm šiem bitiem jābūt notīrītiem rakstot UBRRnH reģistrā.

UBRRn11:0: USARTn Baud Rate Register - 12 bitu reģistrs, kas glabā USART'a pārraides ātrumu. Notiekoša pārraide būs bojāta ja to mainīs. Rakstot UBRRnL izraisīs tūlītējo pārraides ātruma priekšdalītāja maiņu.

Programmas kods

```
/*****************************************************************/
#define FOSC 14745600UL      //Mikrokontrollera takts frekvences defin??ana
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
/*****************************************************************/
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
/*****************************************************************/
void USART_Init(unsigned int ubrr );
void USART_Transmit( unsigned char data );
unsigned char USART_Receive( void );
unsigned char length(unsigned char* msg);

void port_init(void);
void init_devices(void) ;
/*****************************************************************/
volatile unsigned long timer=0;
/*****************************************************************/
ISR(TIMER0_OVF_vect) {
        timer+=255*128;
}
/*****************************************************************/
int main( void ){
        init_devices();
        unsigned char i = 0, send = 1, *temp, msg_length,
                msg[] = "Ievadiet LED gaismas aizkaves vertiibu[apstiprinat ar ENTER] = ";

        temp = msg;
        msg_length = 64;

        USART_Init ( MYUBRR );

        /****LEDs**************/
        unsigned char port_data=0b00000001,x=8,stop=0;
        unsigned long *ptr_delay,time_delay[8]={0.5,1,0.5,1,0.5,1,0.5,1};
        unsigned long got_delay = 1;
        ptr_delay = time_delay;

        while(1){
                        if(send){
                                for(i = 0 ; i < msg_length; i++, temp++) {
                                        USART_Transmit ( *temp );
                                }
                                temp = msg;
                                send = 0;
                        }

                        unsigned char *begin, ii=0,iii;
```

```
                        if(!send & !stop){

                        //if ((*temp = USART_Receive()) == 'q') USART_Transmit (*temp);
                        //unsigned char *begin, ii=0, got_delay = 0;
                        begin = temp;

                                while( (*temp = USART_Receive()) != 'q'){
                                        temp++;
                                        ii++;
                                }

                        temp = begin;
                                got_delay=(long) atoi(temp);
                                stop = 1;
                        }
                        /******LEDs*****************************/
                        //if( timer>(FOSC)){
                        if( timer>(FOSC*(got_delay))){
                                ptr_delay++;

                                PORTD=~port_data;
                                port_data=port_data<<1;

                                if(x==0) {port_data=0b00000001; x = 8; ptr_delay = time_delay; stop = 0; send =
1;}
                                else    --x;
                                timer = 0;
                        }


        }
        return 0;
}
/*****************************************************************/
void port_init(void){
 DDRD  = 0xFF; //visas porta D l?nijas uz IZvadi
 PORTD  = 0x11; //porta D izejas l?niju l?me?i uz 0
}

void init_devices(void) {
sei();

//timer0 setup
TCCR0 = 0b00000101; //F_CPU/1024 111
TIMSK = 0b00000001; //overflow enable
TCNT0 = 0;                      //initialize timer

port_init();                    //Izsauc funkciju, kas inicializ? portus
}

/*global interrupt flag should be cleared (and interrupts globally disabled)
        when doing the initialization.*/
```

```
/*The TXC flag can be used tocheck that the Transmitter
        has completed all transfers, and the RXC flag can be used
        to check that there are no unread data in the receive buffer.
        Note that the TXC flag must be cleared before each transmission
        (before UDR is written) if it is used for this purpose.*/
void USART_Init(unsigned int ubrr ){
        cli();
        /* Set baud rate */
        UBRR0H = (unsigned char)(ubrr>>8);
        UBRR0L = (unsigned char)ubrr;

        /* Enable receiver and transmitter */
        UCSR0B = (1<<RXEN)|(1<<TXEN);

        /* Set frame format: 8data, 2stop bit */
        UCSR0C = (1<<USBS)|(3<<UCSZ0);
        sei();
}

/*************************************************************************/
/*A data transmission is initiated by loading the transmit buffer with the data to be transmitted*/
void USART_Transmit( unsigned char data ){
/* Wait for empty transmit buffer (UDRE is set)*/
while ( !( UCSR0A & (1<<UDRE)) );

/* Put data into buffer, sends the data */
UDR0 = data;
}

/*************************************************************************/
unsigned char USART_Receive( void ){
/* Wait for data to be received */
while ( !(UCSR0A & (1<<RXC0)) );

/* Get and return received data from buffer */
return UDR0;
}
/*************************************************************************
unsigned char length(unsigned char* msg){
        unsigned char size,*tmp;
        tmp = msg;
        while

        return size;
}*/
```

2. variants ar pārtraukumiem

```
/******* CONST ****************************************************/
#define FOSC 14745600UL //MCU System clock(XTAL pin frequency) cycles per second
#define BAUD 9600 // the transfer rate in bit per second (bps)
#define _UBRR FOSC/16/BAUD-1  //Contents of the UBRRH and UBRRL Registers, (0 -
4095)
```

```
#define SIZE 8
/****** HEADER FILES ************************************************/
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h> //atoi
/****** USER DEFINED TYPES ******************************************/
typedef struct ptr_object{
      int start;
      int end;
      unsigned char size;
      unsigned char *value;
} ptr_object;
/***** FUNCTION PROTOTYPES ******************************************/
void port_init(void);
void init_devices(void);

void USART_Init(unsigned int ubrr);
void USART_Transmit(unsigned char data);
unsigned char USART_Receive(void);

unsigned char _strlen(unsigned char *string); //string length
unsigned char _rotl(const unsigned char value, unsigned char shift); //circular
shift

void cbWrite(ptr_object *cb, unsigned char *_byte);
unsigned char cbRead(ptr_object *cb);
unsigned char cbIsFull(ptr_object *cb);
unsigned char cbIsEmpty(ptr_object *cb);
/****** GLOBAL VARIABLES ********************************************/
      /******  *****************/
      /****** TIMER0 *****************/
      volatile unsigned long timer = 0;

      /****** CBUFFER ****************/
      volatile ptr_object buffer;
      volatile unsigned char tx_complete = 1;
      volatile unsigned int rx_delay = 1;
/****** INTERRUPTS **************************************************/
ISR(TIMER0_OVF_vect) {
      timer+=255*128; }
//receive complete
ISR(USART0_RX_vect){
      //save data to buffer
      //*buffer.value = USART_Receive();
```

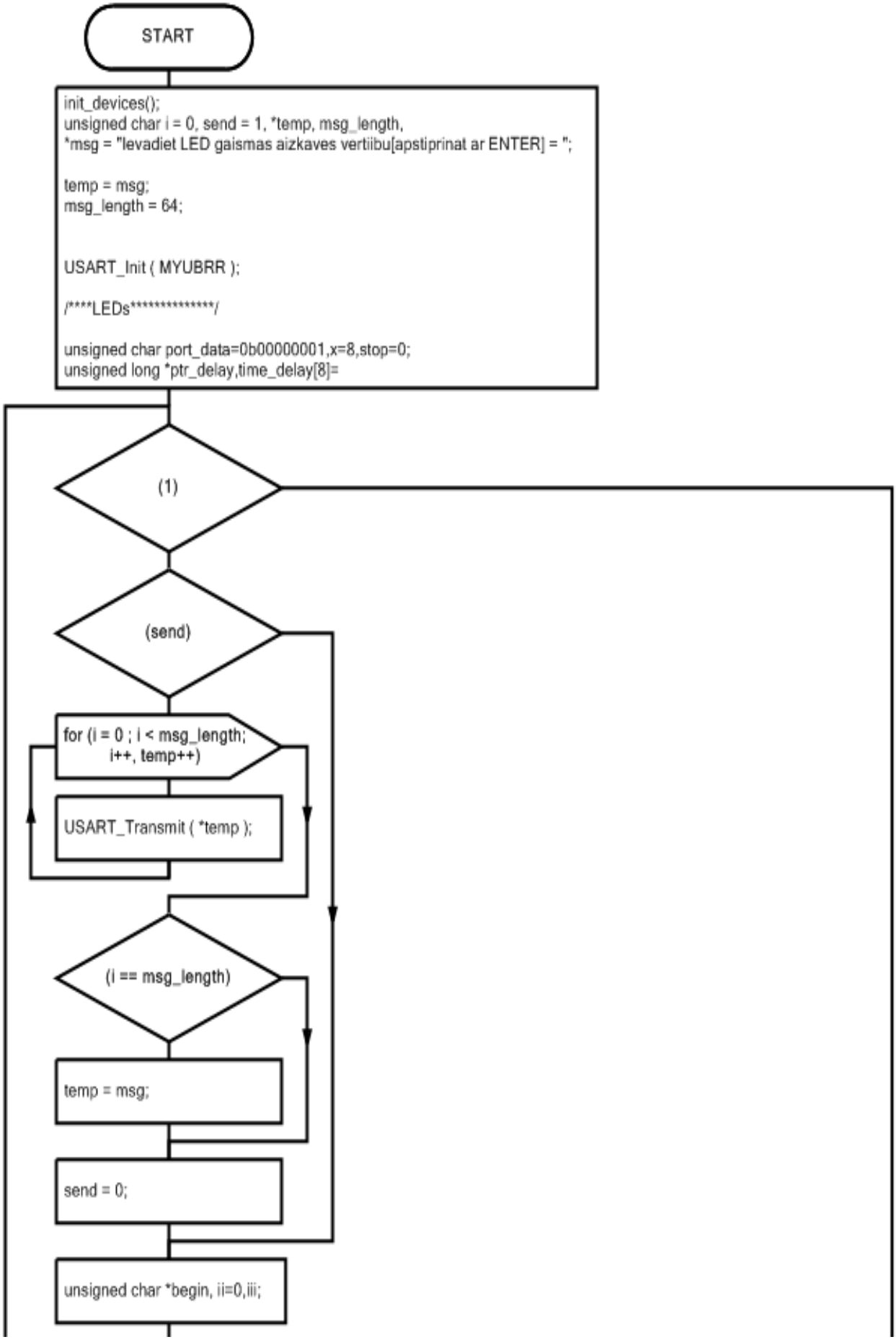
```
	unsigned char *tmp;
	*tmp = USART_Receive();
	cbWrite(&buffer, tmp);
	//set delay
	if (cbIsFull(&buffer) == 1){
		rx_delay = (unsigned int) atoi (buffer.value);
		do{
			USART_Transmit(*buffer.value);
			buffer.value++;
			buffer.start = (buffer.start + 1) % buffer.size;
		}while(!cbIsEmpty(&buffer));
	}
	//echo received data
	//USART_Transmit(*buffer.value);
	//allow communicate
	tx_complete = 1;
}
//UDR empty
ISR(USART0_UDRE_vect){
	if (tx_complete == 1){
		ptr_object elem;
		unsigned char tmp[] = "\n\rSYN:CHAARONII TO YOUR SEVICE\n\r";

		elem.value = tmp;
		elem.size = _strlen(elem.value);
		do{
			USART_Transmit(*elem.value);
			elem.size--;
			elem.value++;
		}while(elem.size);

		elem.value = "SYN";
		elem.size = _strlen(elem.value);
		tx_complete = 0;
	}
}
/****** PROGRAMM BODY ************************************************/
int main(void){
	init_devices();
	//LEDs
	unsigned char port_data = 0x80;

	while(1){
```

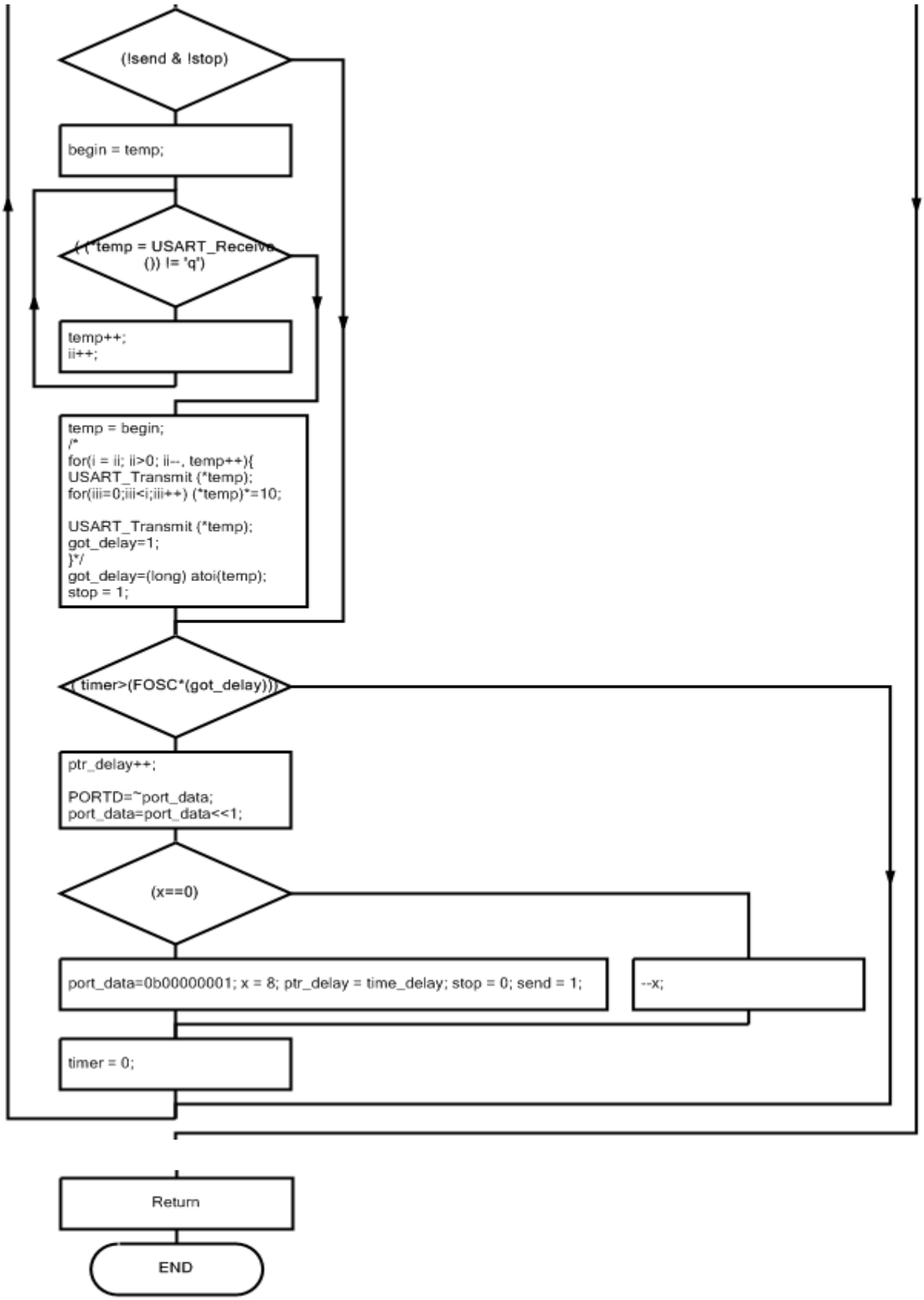
```
            /****** LEDs *****************/
            if(timer > (FOSC * (rx_delay)) ){
                  port_data = _rotl(port_data,1);
                  PORTD = ~port_data;
                  timer = 0;
            }
      }
      return 0;
}
/****** END PROGRAMM BODY ********************************************/
/****** SYSTEM CONFIGURATION *****************************************/
void port_init(void){
      DDRD  = 0xFF;     //port D o/p
      PORTD  = ~(0x00); //LEDs off

      sei();      //allow global interrupts
}
void init_devices(void){
      /**** TIMER0 SETUP **********/
      TCCR0 = 0x05;     //F_CPU/32 11[off(0)/1/8/32/64/128/256/1024]
      TIMSK = 0x01;     //overflow enable
      TCNT0 = 0;        //set start value
      /**** USART0 SETUP **********/
      USART_Init(_UBRR);
      /**** WATCHDOG SETUP ********/
      //COP_init();
      /**** PORT SETUP ************/
      port_init();
}
void USART_Init(unsigned int ubrr){
      /* Set baud rate */
      UBRR0H = (unsigned char)(ubrr >> 8);
      UBRR0L = (unsigned char)ubrr;
      /* Enable receiver and transmitter, complete interrupt enable */
      //UCSR0B = (1<<RXCIE0)|(1<<TXCIE0)|(1 << RXEN0)|(1 << TXEN0);
      UCSR0B = (1<<RXCIE0)|(1<<UDRIE0)|(1 << RXEN0)|(1 << TXEN0);
      /* Set frame format: 8data, 2stop bit */
      UCSR0C = (1 << USBS) |(3 << UCSZ0);
      //set by def |(3 << UCSZ0);   !?????
      //MAX buffer size + one empty to avoid start == end
      buffer.size = 2 + 1;
      buffer.start = 0;
      buffer.end = 0;
}
```

```
/****** IN PROGRAMM FUNCTIONS ***************************************/
void USART_Transmit( unsigned char data ){
      /* Wait for empty transmit buffer (UDRE is set)*/
      while ( !( UCSR0A & (1<<UDRE)) );

      /* Put data into buffer, sends the data */
      UDR0 = data;
}
unsigned char USART_Receive( void ){
      /* Wait for data to be received */
      while ( !(UCSR0A & (1<<RXC0)) );

      /* Get and return received data from buffer */
      return UDR0;
}
/******** BUFFER FUNC ****************************/
void cbWrite(ptr_object *cb, unsigned char *_byte) {
      //if(cbIsFull(cb)) return;
    cb->value[cb->end] = *_byte;
    cb->end = (cb->end + 1) % cb->size;
    if (cb->end == cb->start)
        cb->start = (cb->start + 1) % cb->size; /* full, overwrite */
}
unsigned char cbRead(ptr_object *cb){
      unsigned char *tmp;
      *tmp = cb->value[cb->start];
    cb->start = (cb->start + 1) % cb->size;
      return tmp;
}
unsigned char cbIsFull(ptr_object *cb) {
    return (cb->end + 1) % cb->size == cb->start; }
unsigned char cbIsEmpty(ptr_object *cb){
      return cb->end == cb->start;
}
/****** WIDE RANGE FUNCTIONS ***************************************/
/******** STRING LENGTH ****************************/
unsigned char _strlen(unsigned char *string){
      unsigned char length = 0;
      while(*(string+length)) length++;
      return length;
}

/******** CIRCULAR SHIFT ****************************/
unsigned char _rotl(const unsigned char value, unsigned char shift) {
```

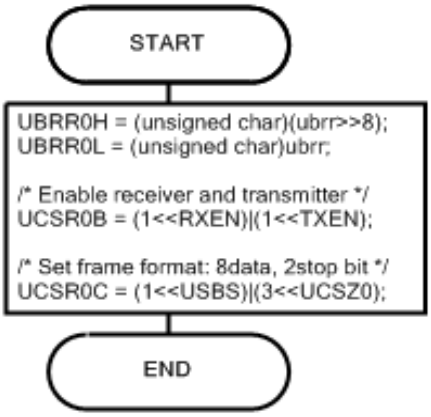
```
    if ((shift &= sizeof(value)*8 - 1) == 0) return value;
    return (value << shift) | (value >> (sizeof(value)*8 - shift));
    //return (value >> shift) | (value << (sizeof(value)*8 - shift)); //right
}
```

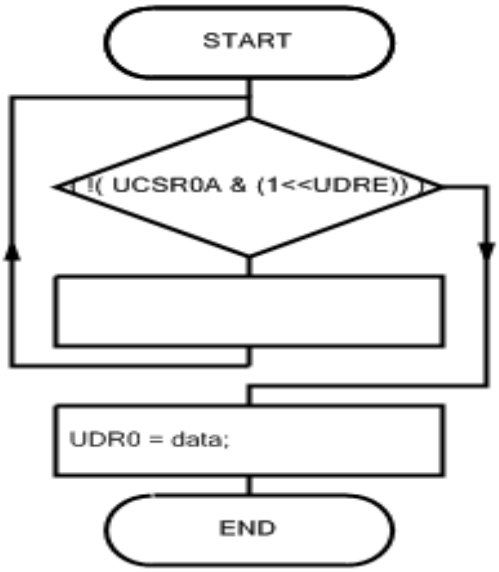
Blokshēma


START
init_devices();
unsigned char i = 0, send = 1, *temp, msg_length,
*msg = "Ievadiet LED gaismas aizkaves vertiibu[apstiprinat ar ENTER] = ";
temp = msg;
msg_length = 64;
USART_Init (MYUBRR);
/****LEDs**********/
unsigned char port_data=0b00000001,x=8,stop=0;
unsigned long *ptr_delay,time_delay[8]=
(1)
(send)
for (i = 0 ; i < msg_length;
i++, temp++)
USART_Transmit (*temp);
(i == msg_length)
temp = msg;
send = 0;
unsigned char *begin, ii=0,iii;

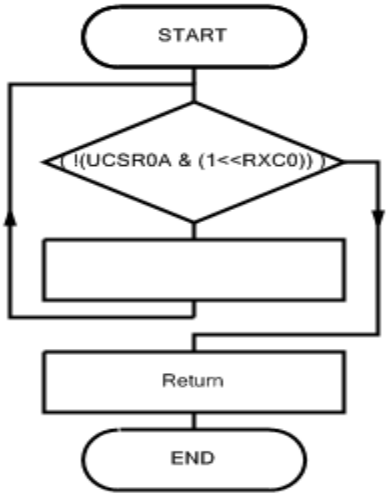


Attēls 8: Main funkcija(programmas ķermenis)





Attēls 9: USART_init() funkcija *Attēls 10: USART_Transmit() funkcija*



Attēls 11: USART_Receive() funkcija

Secinājumi

USART modulis ir diezgan sarežģīta mikrokontrollera daļa, kas atbild par tā komunikācijas spējam ar parējām iekārtam, e.g. Ar datoru. Lielas sistēmas ar vairākiem mikrokontrolleriem pievienotiem viena kopnē, mikrokontrolleris var savākt informāciju no vadāmiem mikrokontrolleriem, nosūtīt to datubāze un vadīt to mikrokontrollera darbības režīmu. Bez tā būtu apgrūtinoši komunicēt, neieskaitot citus komunikācijas veidus.

USART ir tas pats UART tikai ar vairākiem uzlabojumiem, kas palielina tā drošību un efektivitāti. Vārds 'asynhronious' nozīme ka UART atgūst kadra kristāla informāciju no datu plūsmas, izmantojot stop un start bitus, lai norādītu katra bitu piederību. Šajā režīma vajag tikai datus. Bet sinhrona datu pārraide, gan dati, gan frekvence. Taču frekvence tiek atjaunota atsevišķi no datu plūsmas un stop/start biti netiek izmantoti, tādējādi palielinot pārraides efektivitāti.

Pēc grūtības pakāpes dotais darbs bija vissarežģītākais, neskatoties uz to, ka īstenība lielākas bažas sagādāja pārtraukumu izmantošana. No citas puses rezultāts un pūles pieliktas tā sasniegšanai bija tā vērti. Darbs bija interesants un žēl ka izbrīvētais laiks šīm priekšmetam ir tik īss. Būtu aizraujoši apgūt vēl sarežģītākas lietas, saistītas ar mikrokontrolleriem, jo šie darbi bija tikai iepazīstinošie ar mikrokontrollera programmēšanu, to arhitektūru un uzbūvi.

Kopumā esmu apmierināts ar iegūtam zināšanām, bet vēlme uzzināt vairāk vēl nezūd...