RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas tehnoloģijas fakultāte

Datorvadības, automātikas un datortehnikas institūts

Datoru tīklu un sistēmas tehnoloģijas katedra

**Mikroprocesoru tehnika**

1. **laboratorijas darbs**

|  |  |
| --- | --- |
| Izpildīja: | Eduards Šarņeckis |
| Grupa: | III RDB F02 |
| Apl. numurs: | 101RDB121 |

2012./2013. māc. gads

**Saturs**

[Uzdevums 3](#_Toc343344569)

[Programmas pirmteksts 4](#_Toc343344570)

[Programmas pirmteksta algoritma blokshēma 8](#_Toc343344571)

[ATmega128 USART modulis 10](#_Toc343344572)

[Secinājumi 15](#_Toc343344573)

# Uzdevums

1. Izmantojot Atmega128 aprakstu nokonfigurēt UART moduli datu sūtīšanai un saņemšanai (Tx un Rx).
2. Pārsūtīt no CharonII uz datoru kādu simbolu vai frāzi.
3. Saņemt no datora kādu simbolu vai frāzi.

# Programmas pirmteksts

#define F\_CPU 14745600UL //Mikrokontrollera taktsfrekvence

#define BAUD 9600 //Komunikacija

#define MYUBRR F\_CPU/16/BAUD-1 //UBRR konfiguracija

/\*\*\*\*\*\*\* Standarta C un specialo AVR biblioteeku ieklaushana\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <avr/io.h>

#include <avr/iom128.h>

#include <avr/interrupt.h>

#include <math.h>

#include <stdlib.h>

#include <stdint.h>

#include <stdio.h>

#include <avr/wdt.h>

#include <string.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define BIT0 0x01 //Nodefineta 0-taa bita maska

#define BIT1 0x02 //Nodefineta 1-taa bita maska

#define BIT2 0x04 //Nodefineta 2-taa bita maska

#define BIT3 0x08 //Nodefineta 3-taa bita maska

#define BIT4 0x10 //Nodefineta 4-taa bita maska

#define BIT5 0x20 //Nodefineta 5-taa bita maska

#define BIT6 0x40 //Nodefineta 6-taa bita maska

#define BIT7 0x80 //Nodefineta 7-taa bita maska

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Portu inicializacijas funkcija \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void port\_init()

{

DDRA = 0x00; //visas porta A linijas uz IEvadi

DDRB = 0x00; //visas porta B linijas uz IEvadi

DDRC = 0x00; //visas porta C linijas uz IEvadi

DDRD = 0xFF; //visas porta D linijas uz IZvadi

DDRE = 0x00; //visas porta E linijas uz IEvadi

DDRF = 0x00; //visas porta F linijas uz IEvadi

DDRG = 0x00; //visas porta G linijas uz IEvadi

PORTA = 0x00; //porta A atsienoshie rezistori pret +Vcc NEtiek izmantoti

PORTB = 0x00; //porta B atsienoshie rezistori pret +Vcc NEtiek izmantoti

PORTC = 0x00; //porta C atsienoshie rezistori pret +Vcc NEtiek izmantoti

PORTD = 0x00; //porta D izejas liniju limenji uz 0

PORTE = 0x00; //porta E atsienoshie rezistori pret +Vcc NEtiek izmantoti

PORTF = 0x00; //porta F atsienoshie rezistori pret +Vcc NEtiek izmantoti

PORTG = 0x00; //porta G atsienoshie rezistori pret +Vcc NEtiek izmantoti

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Kontrollera inicializacija\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init\_devices()

{

XDIV = 0x00; //takts impulsu dalitajs NEtiek izmantots

XMCRA = 0x00; //arejo atminju NEizmanto

MCUCR = 0x00; //NEtiek izmantoti nekadi energiju tauposhi stavokli

TCCR0=0b00000010;

TIMSK=0b00000001;

sei(); //atlauj globalos partraukumus

port\_init(); //inicialize portus

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Globalie mainigie\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

volatile unsigned long long taktis=0;

volatile int n = 0;

volatile int k = 1;

uint8\_t Temp;

char aizkave[5];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*Partraukuma no taimera/skaititaja0 parpildes funkcija\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ISR(TIMER0\_OVF\_vect)

{

taktis=taktis+256;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*USART parraides inicializacija\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void USART\_Transmit(unsigned char data)

{

/\* Wait for empty transmit buffer \*/

while ( !( UCSR0A & (1<<UDRE)) );

/\* Put data into buffer, sends the data \*/

UDR0 = data;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*USART modula partraukums\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ISR(USART0\_RX\_vect)

{

//define temp value for storing received byte

//Store data to temp

Temp=UDR0;

USART\_Transmit(Temp);

if (Temp==13){

k = atoi(aizkave);

int i=0;

for (i=0;i<5;i++){

aizkave[i]=0;}

n=0;

}

else

{

aizkave[n] = Temp;

n++;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*USARD modula inicializacija\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void USART\_Init(unsigned int ubrr)

{

/\* Set baud rate \*/

UBRR0H = (unsigned char)(ubrr>>8);

UBRR0L = (unsigned char)ubrr;

/\* Enable receiver and transmitter \*/

//UCSR0B = ((1<<RXEN)|(1<<TXEN))|(1<<RXCIE0);

UCSR0B = 0b10011000;

/\* Set frame format: 8data, 2stop bit \*/

UCSR0C = (1<<USBS)|(3<<UCSZ0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*USART simbolu rindas parraidisana\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void USART\_TransmitString(char data[])

{

int i = 0;

while (data[i]!='\0')

{

USART\_Transmit(data[i]);

i++;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*USART sanemsanas inicializacija\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char USART\_Receive()

{

/\* Wait for data to be received \*/

while ( !(UCSR0A & (1<<RXC)) );

/\* Get and return received data from buffer \*/

return UDR0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main funkcija (Ar sho sakas programmas izpilde)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

unsigned char i, port\_data; //Lokalie mainigie

init\_devices(); //Inicializejam kontrolleri

USART\_Init(MYUBRR); //Inicializejam USART moduli

port\_data=0b00000001;

i=0;

USART\_TransmitString("Hello, world!");

while(1) //Muzigais cikls, lai programma nekad nebeigtos

{

if (taktis>=(1843200\*k))

{

PORTD=~port\_data;

port\_data = port\_data<<1;

taktis = 0;

i++;

if (i==8)

{

port\_data = 0b00000001;

i=0;

}

}

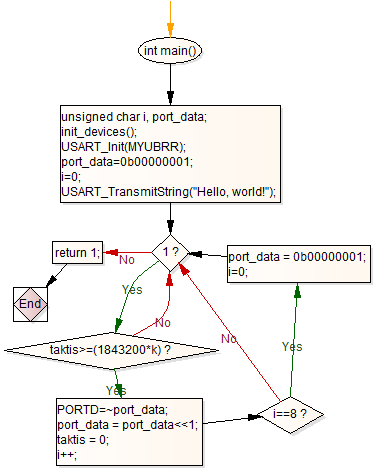
}

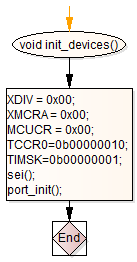
return 1;

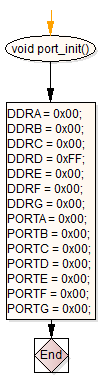
}

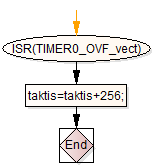
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

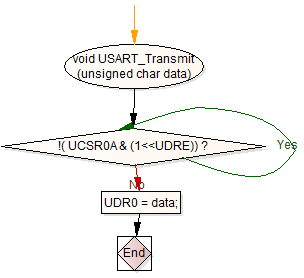
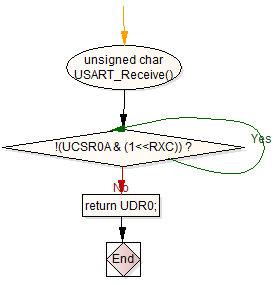
# Programmas pirmteksta algoritma blokshēma

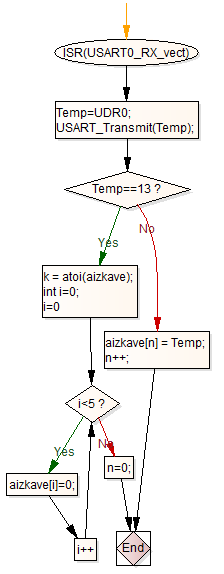
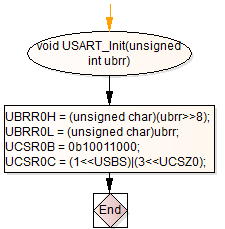
 Funkcija main() Funkcija init\_devices()

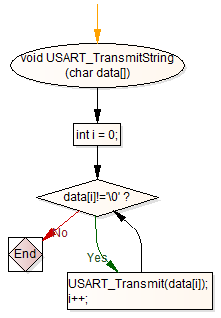


 Funkcija port\_init()

 ISR(TIMER0\_OVF\_vect)

 USART\_Transmit(unsigned char data) USART\_Receive()

 ISR(USART0\_RX\_vect) USART\_Init(unsigned int ubrr)

 USART\_TransmitString(char data[])

# ATmega128 USART modulis

USART (Universal Synchronous asynchronous receiver/transmitter) modulis – ir skaitļošanas tehnikas daļa, kas pārveido datus no paralēlas formas seriālajā. USART parasti tiek izmatots 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). ATmega128 mikrokontrollerī arī ir iebūvēts USART modulis (Att. 26), kurš aktivizē sazināšanas iespēju starp mikrokontrolleri un, piemēram, datoru.



1. attēls USART modulis Atmega128 mikrokontrollerī

USART modulis ATmega128 mikrokontrollerī ir sadalīts trīs blokos: taktsignāla ģeneratora, raidītāja un saņēmēja (angl. Clock Generator, Transmitter, Receiver).

USART moduli veido:

* UBRR[H:L] –reģistrs, kas uzdot ātrumu. Ierakstītais tur skaitlis noteic takts frekvences dalīšanas vērtība, vajadzīgais koeficients tiek izskaitļots vai nu pēc formulas, vai (standarta kvarcam ) pēc tabulas.



2. attēls Ātruma regulēšanas reģistrs UBRR

* Boda ātruma ģenerators (angl. BAUD RATE GENERATOR) – pirms dalītājs, kas ģenerē pārraidāmos impulsus ar noteiktu ātrumu, kuru var noteikt pēc formulas

, kur

**fosc**– mikrokontrollera takts frekvence, **UBRR** – UBRR[H:L] reģistra vērtība.

* Sinhronizēšanas loģika (angl. SYNC LOGIC ) – nepieciešama sinhronai pārraidei.
* Izvada kontrole (angl. PIN CONTROL) – nodrošina informācijas izvadi uz līniju.
* Taktēšanas impulsa pārnese (angl. XCK) – šā līnija tiek izmantota tikai sinhronā režīmā taktēšanas impulsu pārraidei/saņemšanai.
* UART datu reģistrs (angl. UDR) – reģistrs, kurš saglabā saņemtos vai sūtītos datus.
* Pārraides pārbīdes reģistrs (angl. TRANSMIT SHIFT REGISTER) – speciāls reģistrs, kurš ar katru padoto taktsimpulsu izvada vienu bitu.
* Paritātes ģenerators (angl. PARITY GENERATOR) – izskaitļo pārraidāmā freima datu paritātes bitu. Ja paritātes ģenerēšana ir atļauta, tad raidītāja vadības loģika ievietos freimā paritātes bitu starp pēdējo datu bitu un stop bitu.
* Pārraides vadības mezgls (angl. TX CONTROL) – nodrošina korektu freimu pārraidi.
* Pārraides līnija (angl. TxD) – līnija, pa kuru tiek pārraidīts freims.
* Saņēmēja pārbīdes reģistrs (angl. RECEIVE SHIFT REGISTER) – speciāls reģistrs, kurš ar katru padoto taktsimpulsu saglabā vienu bitu.
* Taktsimpulsu atgūšanas bloks (angl. CLOCK RECOVERY) – bloks tiek izmantots iekšējā taktsimpulsa ģeneratora un ienākošā freima sinhronizācijai.
* Datu atgūšanas bloks (angl. DATA RECOVERY) – iztver katru ienākošo bitu, kā arī satur zemo frekvenču filtru, kas nodrošina noturīgumu pret kļūdām.
* Paritātes pārbaude (angl. PARITY CHECKER) – speciāls bloks, kurš salīdzina saņemto paritātes bitu ar no freima datiem izskaitļotu.
* Statusa reģistrs UCSRA – speciāls statusa reģistrs:



3. attēls Statusa reģistrs UCSRA

* + RXCn – norāda, vai ir pabeigta datu saņemšana (0, kad datu buferis ir tukšs);
  + TXCn – norāda, kad datu pārraide ir pabeigta (t.i., pārbīdes reģistrs ir tukšs);
  + UDREn – karodziņš, kurš norāda, vai UDR reģistrs var saņemt datus;
  + FEn – norāda, ka ir notikusi freima kļūda;
  + DORn – datu pārpilde, notiek kad, saņemšanas buferis ir pilns, pārbīdes reģistrā ir jauns simbols un jauns starta bits tiek konstatēts;
  + UPEn – parāda vai ir notikusi paritātes kļūda;
  + U2Xn – divkāršots USART pārraides ātrumu asinhronajā režīmā;
  + MPCMn – ierakstot „1” iestāda vairākprocesoru komunikācijas režīmu.
* Statusa reģistrs UCSRB – speciāls statusa reģistrs:



4. attēls Statusa reģistrs UCSRB

* RXCIEn – ierakstot 1 aktivizē pārtraukumu uz datu saņemšanas pabeigšanu;
* TXCIEn – ierakstot 1 aktivizē pārtraukumus uz datu sūtīšana pabeigšanu;
* UDRIEn – ierakstot 1 aktivizē pārtraukumus uz datu reģistra iztukšošanu;
* RXENn – ierakstot 1 aktivizē USARTn uztvērēju;
* TXENn – ierakstot 1 aktivizē USARTn raidītāju;
* UCSZn2 – uzstāda datu bitu daudzumu freimā;
* RXB8n – šeit tiek saglabāts saņemtais 9. datu bits;
* TXB8n – šeit tiek saglabāts nosūtītais 9. datu bits;
* Statusa reģistrs UCSRC – speciāls statusa reģistrs:



5. attēls Statusa reģistrs UCSRC

* UMSELn – uzstāda sinhrono (1) vai asinhrono (0) USART moduļa režīmu;
* UPMn1:0 – uzstāda un aktivizē paritātes izskaitļošanu un pārbaudīšanu;
* USBSn – uzstāda stop bitu skaitu:



* UCSZN1:0 – uzstāda datu bitu daudzumu:



* UCPOLn – izmanto tikai sinhronajā režīmā, uzstāda attiecību starp datu izvades izmaiņām un datu ievades izmaiņām signālā un sinhronajās taktīs;

USART datu pārraides reģistrs (UBRR) un atpakaļ skaitītājs ir savienots tā ka tas funkcionē kā programmējams pirms dalītājs, jeb datu pārraides ātruma ģenerators. Atpakaļ-skaitītājs, kurš darbojas izmantojot sistēmas pulksteni (fosc), tiek piepildīts ar UBRR vērtību katru reizi kad skaitītājs saskaita atpakaļ līdz nullei. Šis pulkstenis ir datu pārraides ģeneratora pulksteņa izvads. Raidītājs dala datu pārraides pulksteņa izvadi ar 2, 8, vai 16 atkarībā no izvēlētā režīma.



BAUD – Datu pārraides ātrums (biti sekundē, bps)

fosc – sistēmas oscilatora takts frekvence.

UBRR – UBRRH un UBRRL saturs

**USART datu sūtītājs**

USART datu pārraidītājs tiek ieslēgts, iestatot pārraides atļaušanas (TXEN) bitu UCSRB reģistrā. Kad pārraidītājs ir ieslēgts, TxD pina porta darbība tiek pārvaldīta ar USART un tiek nodota funkcijai kā pārraidītāja seriālā izeja. Datu pārraides ātrums, darbošanās režīms kā arī freima formāts ir jāuzstāda pirms pārraides. Ja sinhronā pārraide netiek izmantota, takts uz XCK pina tiks pārtverta un tiks izmantots pārraides taktētājs.

Datu pārraide tiek uzsākta, ielādējot pārraides buferī pārraidei domātos datus. CPU var ielādēt datus pārraides buferī, ierakstot UDR reģistrā I/O vietu. Buferī ievietotie dati no pārraides bufera tiek pārvietoti uz pārbīdes reģistru, kad pārraides reģistrs ir gatavs jauna freima sūtīšanai. Pārbīdes reģistrā ielādē jaunus datus, kad tas ir bezdarbības režīmā (nav izejošu sūtījumu), vai tūlīt pēc pēdējā stop bita no iepriekšējā freima pārraides. Kad pārbīdes reģistrs tiek pielādēts ar jauniem datiem, tas pārraidīs vienu pilnu freimu reģistrā norādītajā pārraides ātrumā.

USART freima formāts tiek uzstādīts UCSRB un UCSRC izmantojot UCSZ2:0, UPM1:0 un USBS bitus. Saņēmējs un sūtītājs izmanto vienus un tos pašus uzstādījumus. Izmainot uzstādījumus jebkuram no šiem bitiem, tiks pārtraukta notiekošā komunikācija starp saņēmēju un sūtītāju.

USART rakstzīmes izmēra(UCSZ2:0) bitiem tiek izvēlēts datu bitu skaits freimā. USART paritātes režīma (UPM1:0) biti ieslēdz un iestāda paritātes bitu tipu. Izvēle starp to vai būs viens vai 2 stop biti tiek izdarīta ar USART stop bita izvēles (USBS) bitu. Saņēmējs ignorē otro stop bitu. Freima kļūda (Frame Error) tiks atklāta gadījumos, kad pirmais stop bits ir nulle.

**USART datu saņēmējs**

USART saņēmējs tiek ieslēgts ierakstot saņēmēja ieslēgšanas (RXEN) bitu UCSRB reģistrā “1”. Kad saņēmējs ir ieslēgts normālā RxD pina operācija tiek pārrakstīta USART un tiek iedota funkcija, kā saņēmēja seriālā ieeja. Datu pārraidīšanas ātrumu un freima formātu uzstāda pirms jebkura seriālā saņemšana ir pabeigta.

Saņēmējs sāk datu saņemšanu, kad tiek saņemts pirmais derīgais start bits. Katrs bits, kurš seko pēc start bita tiek samplēts pēc datu pārsūtīšanas ātruma un pārbīdīts uz saņēmēja pārbīdes reģistru, kamēr nav saņemts freima pirmais stop bits. Nākošais stop bits tiks ignorēts. Kad ir saņemts pirmais stop bits, tas nozīmē, kad viss freims ir ievietots pārbīdes reģistrā, pārbīdes reģistra saturs tiek pārvietots uz saņēmēja buferi. USART saņēmējam ir viens karogs, kas norāda saņēmēja stāvokli. Datu saņemšanas pabeigšanas (RXC) karogs norāda vai ir nenolasīti dati saņēmēja buferī. Šis karogs ir “1”, kad nenolasīti dati ir saņēmēja buferī, un nulle, kad saņēmēja buferis ir tukšs. Ja saņēmējs ir atslēgts (RXEN=0), saņēmēja buferis tiks nodzēsts un tātad RXC bits kļūs par “0”.

# Secinājumi

Sestā laboratorijas darba gaitā es iepazinos ar mikrokontrollera ATmega128 USART moduļa darbību.

Laboratorijas darba sākumā es nokonfigurēju programmu PuTTY komunikācijai ar ATmega128 caur COM portu un uzrakstīju programmas kodu, kas konfigurē USART moduli datu sūtīšanai un saņemšanai. Koda piemēri bija atrodami ATmega128 aprakstā. Tad es papildināju programmas kodu, lai no Charon II varētu pārsūtīt simbolu uz datoru. Izpildot pēdējo laboratorijas darba uzdevumu, es vēlreiz modificēju kodu, lai būtu iespējams saņemt no datora kādu simbolu vai frāzi un pārbaudīt to.

Šis laboratorijas darbs bija grūtāks nekā iepriekšējie, tomēr to bija interesanti izpildīt. Ar programmu PuTTY es jau saskaros pirmajā kursā - mācību priekšmetā „Datoru arhitektūra”. ATmega128 mikrokontrollera apraksts arī zināmi atviegloja darbu, jo tur varēja atrast koda piemērus darbam ar USART moduli.

Uzskatu, ka laboratorijas darbs ir veiksmīgi izpildīts, jo programmas kods strādā atbilstoši laboratorijas darba uzdevumam un tika iegūtas jaunas zināšanas par ATmega128 USART moduli.