UYGULAMA NOTU

**UMD-B12 – RF MODEM BOARD**

Bu uygulama notunda, UTR-C12U data transceiver kullanılarak iki bilgisayar arasında kablosuz iletişim uygulaması anlatılmıştır. Bu uygulama notu ile beraber ürün kılavuzunun okunması tavsiye edilir.

PC

PC

Seri port

Seri port

Mikrodenetleyici

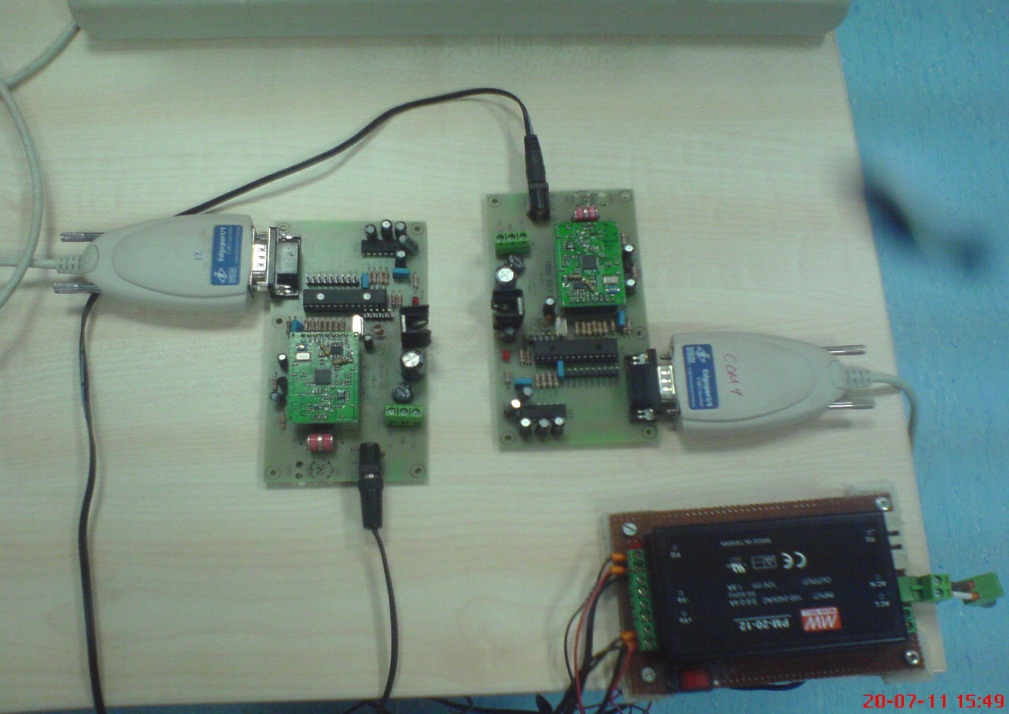
Mikrodenetleyici

UTR-C12

UTR-C12

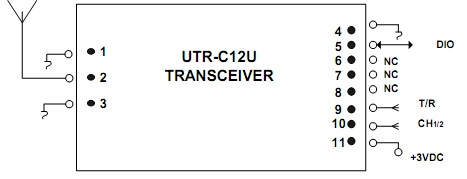
Şekil 1- Basitçe iki modem arasındaki işlem yukardaki gibidir.

1. **Giriş**



Şekil - 2

Öncelikle kullanacağımız elemanları tanıyalım. Şekil 2’de görülen iki büyük yeşil kart UMD-B12 RF modem kartlarıdır. Şekildeki siyah kutu, kartlara gerekli elektrik enerjisini sağlayan adaptördür. Modem kartları üzerindeki daha yeşil olan küçük dörtgenler ise UTR-C12U data transceiverdır. Bu modüllerin görevi, 1 ve 0’lardan oluşan elektriksel bilgiyi elektromanyetik dalgalara çevirip havaya vermek ve tam tersini yapmaktır. Yani, hem alıcı hem de verici olarak çalışabiliyor. Modem kartına bağlanmış büyük gri aygıt, kart ile bilgisayar arasındaki bağlantıyı sağlayan seri porttur. Modem kartı üzerinde bulunan siyah küçük dikdörtgen cisim ise bizim programlayacağımız mikrodenetleyicidir. Sonuç olarak, UMD-B12 RF modem kartı, UTR-C12U RF modülün enerjisi ve çevreyle bağlantısı için bir taban oluşturuyor.



Şekil 3

Şimdi kullanacağımız UTR-C12U RF modülü inceleyelim. Modül havadan aldığı elektromanyetiksel bilgiyi elektriksel (dijital) bilgiye dönüştürür ve bunu 5. pininden verir. Ve yahut, 5. pinden aldığı elektriksel bilgiyi elektromanyetiksel bilgiye çevirir. Hangi işlemi yapacağı 9. pine verilecek 1 ve 0 ile belirlenir. 1 verilirse modül verici; 0 verilirse modül alıcı konumuna geçer. 10. pin modülün kullanacağı kanalı seçer.

UTR-C12U data tranceiver ile bilgisayar arasındaki bağlantıyı mikrodenetleyici yönetecek. Mikrodenetleyici olarak PIC16F876A kullanılacak. PIC bacakları ile UTR-C12U pinleri arasındaki bağlantı şekil 4’te gösterilmiştir. Modülün 5. pini (DIO) , PIC’in RA0 bacağına bağlanmıştır. Şekil 4’te bağlantı yapılmış modülün 6. pini bu projede kullanılmayacaktır.

1. **Data kodlaması**

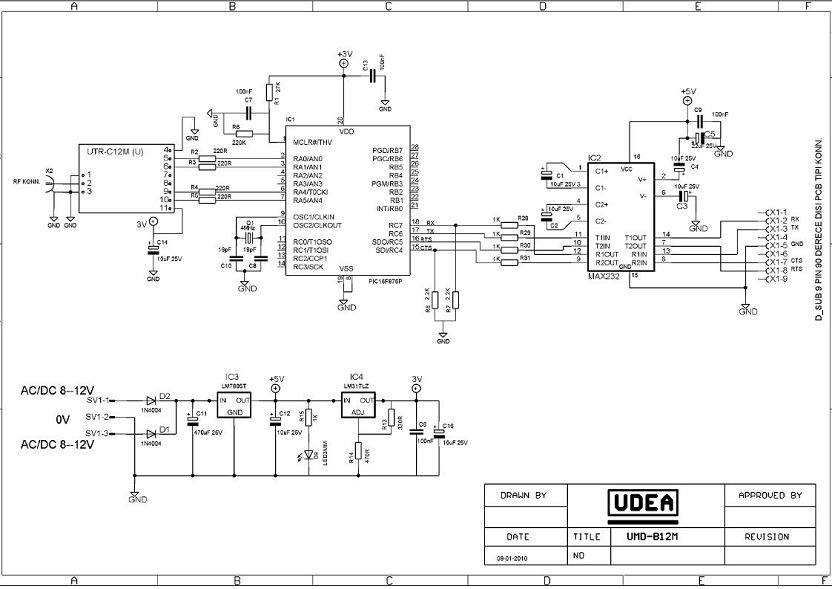
UTR-C12U’un data hızı saniyede 2400 bittir. Bu demektir ki; bir modül tarafından gönderilen bir bitin diğer modül tarafından algılanabilmesi için, bit uzunluğu en az 1/2400 saniye (= 417 us) olmalıdır. Yani gönderilecek bit 1’e eşitse 417 boyunca 1 gönderilecek. Fakat biz 1 ve 0’lardan oluşan dijital bilgimizi kodlayarak göndereceğiz.

Kodlama işlemi şöyle anlatılabilir: Diyelim ki bir bit göndereceğiz, bu bit de 1’e eşit. 1 ms 1, 450 us 0 göndereceğiz. Bit 0’a eşitse, 450 us 1, 1 ms 0 göndereceğiz. Yani,

Bit = “ 1 “ --- Bit = “ 0 “ ---

1 ms 450 us 450 us 1 ms

Neden kodlama yaptığımıza gelince, bu, modülün elektriksel bilgiyi elektromanyetiksel bilgiye dönüştürken yaptığı FSK modülasyon ile ilgili. Sürekli 1’lerden veya 0’lardan oluşan bir bilgiye FSK modülasyon yapılamaz. Bu yüzden, biz de hep 1 bilgisi gönderilmesi durumunda sağlıklı modülasyon yapılabilmesi için gönderilecek datayı kodladık. Fakat sağlıklı modülasyon için data hızımızdan feragat ettik, çünkü gönderdiğimiz bir bit 1450 us uzunluğunda.

Şekil 4

1. **Data protokolü**

Diyelim ki 5 byte’lık ‘tugba’ bilgisini göndermek istiyoruz. Bunun için önce preamble adını verdiğimiz, 5 byte’lık ( 0xAA ) sinyalini gönderiyoruz. Bu sinyal ardışık 40 bit 1 ve 0’dan oluşuyor ve donanımın senkronizasyonunu sağlıyor. Daha sonra senkron adını verdiğimiz sinyali yolluyoruz. Bu sinyal 24 ms 1, ve 24 ms 0’dan oluşuyor. Havada antenin alabileceği bir gürültü mutlaka vardır, ama senkron gibi bir gürültü olamaz. Bu yüzden alıcının göndereceğimiz bilgi ile gürültüyü ayırabilmesi için senkron gönderiyoruz. Alıcı, sürekli senkron arar, bulduğu anda bilgiyi de almaya başlar. Senkron yazılımın senkronizasyonunu sağlar. Senkronu gönderdikten hemen sonra bilgi gönderilir.

Data = Preamble + Senkron + bytesayısı + ‘<’ + byte1 + byte2 + .... + ‘>’ + CheckSum

Bytesayısı ve CheckSum bilginin doğru gidip gitmediğini kontrol edebilmek için yollanır. CheckSum, ( < + byte1 + byte2 + .... + > ) karakterlerinin hexadesimal değerlerinin toplamıdır. Alıcı tarafında karakterler toplanır, alınan CheckSum değeri ile karşılaştırılır, eğer aynıysa bilgi doğru alınmış hükmü verilir.

1. **Mikrodenetleyicinin programlanması**

(PIC C programlama kodunun tamamını raporun sonunda bulabilirsiniz.)

PIC’ in ilk ayarları yapılır.

TRISA=0x01; // başlangıçta modül alıcı konumunda olduğu için, RA0 input durumundadır. PORTA=0x00; // RA4 bacağı ‘0’ yapılır. Böylece modül alıcı konumuna alınmıştır.

ADCON1=0x07; // A portu analogdur, bu komutla dijital yapılır.

TRISC6=1; // Bu komutlar UART ayarları için.

TRISC7=1;

BRGH=1;

SPBRG=25;

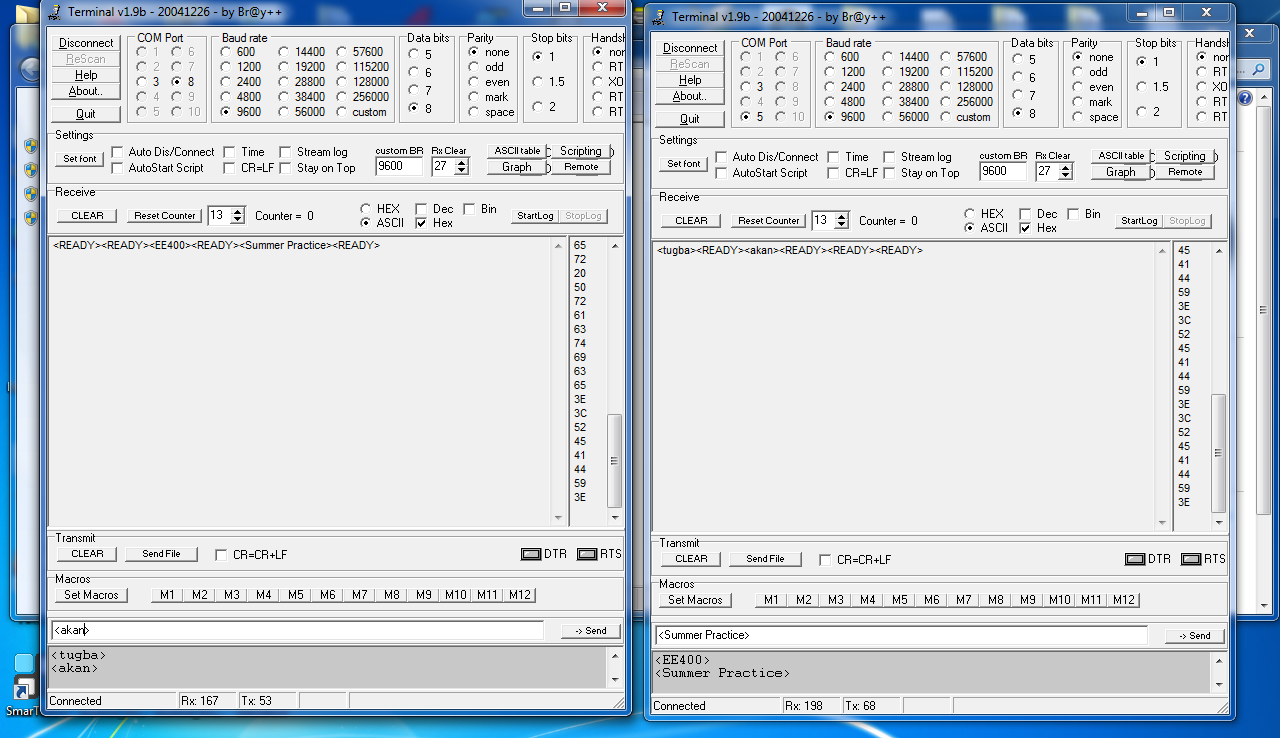
SYNC=0;

TXEN=1;

CREN=1;

SPEN=1; // Sonuç olarak UART ‘Baud Rate’ i 9600 yapıldı.

Sistemin bilgisayar ile iletişimi için Terminal programı açılır. Programın baud rate ayarı 9600 yapılır. ( Burada RF modül ile seri portun baud rate’leri karıştırılmamalıdır, ikisi farklı ve bağımsız şeylerdir. Modülün baud rate’ i tasarımı esnasında belirlenir(2400). Seriporttan gönderilecek datanın baud rate’ i ise PIC’ i programlarken yapılan UART ayar komutları ile kullanıcı tarafından belirlenir(9600). ) Seri port kablosunun hem bilgisayara hem de modem kartına takılı olduğundan emin olunuz. Daha sonra ‘connect’e tıklanır. Bu program sayesinde PIC’e istediğimiz verileri gönderebileceğiz, ve PIC’ten gelen verileri de görebileceğiz. PIC’i programladıktan sonra iki terminal programı açıp, bunları farklı modem kartlarına bağlayacağız. Aralarında sadece kablosuz iletişim kuran bu kartlar, birbirlerinin gönderdiği dataları alabilecekler. Daha anlamlı olması için Şekil 5‘e bakabilirsiniz. Şekil 5’te iki terminal programı aynı bilgisayarda açıldı, ama farklı bilgisayarlarda da açılabilir.



Şekil 5

Şimdi sistemin nasıl çalıştığına bakalım. Sistem başlangıçta alıcı konumunda olduğu için sürekli senkron arar. Senkron başarıyla bulunduğunda data okumaya başlanır. Önce byte sayısı alınır. Sonra gönderilen bilgiyi okumaya başlanır. Bilgi mutlaka “<” ile başlayıp “>” ile bitmelidir. Bilgi “<” ile başlamazsa senkron alınmışsa bile gelen bilgiyi almaz, Terminal’e ‘error’ yazdırır ve tekrar senkron aramaya devam eder. Bilgi “>” ile bitmez ise gelen serinin bitmediğini düşünür, ve <TIME OUT> hatası verir. Bu hata ayıklaması için, Timer1 sayıcısı ayarlanmıştır ve interrupt kullanılır. En son CheckSum alınır. Byte sayısı ve alınan karakterlerin toplamı kontrol edilir. Eğer karşılaştırma sonucu doğru çıktıysa bilginin doğru alındığı hükmü verilir ve bilgi seriporttan bilgisayara gönderilir. Terminal programı sayesinde ekrana basılır. Sistem sonra tekrar alıcı konumuna geçer.

Sistem alıcı konumundayken senkron aramasının yanı sıra, seriporttan gelecek bilgiyi de bekler. Yani, terminal programına bir yazı yazdığımızda ve gönderdiğimizde sistem bunu alır. ‘<’ ve ‘>’ karakteri arasında yolladığımız bilgi, ‘TransmitDataReady’ değişkenini 1 yapar. Bu değişkenin 1 olması, sistemin senkron aramasını durdurur ve sistemi ‘verici’ konumuna alır. Yani, PIC’in RA4 bacağı (modülün 9. pini) 1 yapılır. Seriporttan alınan bilgi PIC tarafından kodlanır ve modüle verilir. Modül de bu bilgiyi elektromanyetik dalgalara gönderip havaya verir, karşı taraftan alınması beklenir. Tabi bilgi data protokolüne uygun şekilde gönderilir (preamble ve senkron gönderimi vs.).

1. **Data okunması**

RF’ten gelen datayı okuyan fonksiyon aşağıdaki gibidir. Datanın her biti, 1 ile başlar, sonra 0 gelir. Bu noktada ‘datanın kodlanması’ kısmını tekrar okumanız gerekebilir. Eğer gelen bit 1 ise, uzun 1, sonra kısa 0 gelir. Eğer gelen bit 0 ise, kısa 1, sonra uzun 0 gelir.

char DataRead()

{

int k=0;

char RF0=0;

unsigned int timereg2=0; // sıfır ve birlerin uzunluğunun tayini için arttırılacak değişken

while(k!=8) // 1 byte = 8 bit

{

RF0=RF0<<1;

do{ timereg2++;} while(RA0==1); // Bitin ilk kısmındaki 1 in uzunluğuna bakılır.

if( timereg2 >=130) // Eğer uzun ise, 1 bilgisi gelmiştir.

RF0=RF0 | 0x01;

timereg2=0;

do{ timereg2++;} while(RA0==0); // 1'den sonra 0 gelir. Bunun uzunluğuna bakılır.

if( timereg2 >=120) // Eğer uzun ise, 0 bilgisi gelmiştir.

RF0=RF0 & 0xFE;

k++;

}

CLRWDT();

return RF0; // Fonksiyon bir byte bilgi döndürür.

}

1. **PIC C kodu**

#include <pic.h>

#include "delay.c"

void InitBoardReciever();

void InitBoardTransmitter();

void InitBoard();

void SendData(char);

void SendSencron();

void EepromWrite(char);

char EepromRead();

char DataRead();

int Sencron();

void excess();

void SendReady();

void Error();

void TimeOut();

void Overflow();

void Reset();

/\*Hata mesajları\*/

char Ready[]= "<READY>";

char Ovf[]= "<OVERFLOW>";

char TimeO[]= "<TIME OUT>";

char Err[]= "<ERROR>";

char RecievedChar[50]; //buffer

int i=0, TransmitDataReady=0, k;

char CheckSum; // RF olarak gönderilen son byte CheckSum'dır, gönderilen karakterlerin ascii // kodlarının toplamıdır. Datanın doğru alınıp alınmadığını kontrol etmek için gönderilir.

char ByteCount; //Alınan toplam byte sayısı

int StartRecieve=0;

int main()

{ CLRWDT();

int j;

char temp;

InitBoard();

SendReady();

while(1)

{

while(!Sencron()) //Senkron bakar

{

CLRWDT();

if(TransmitDataReady) //Eğer seriporttan gelen data yollanmaya hazırsa senkron bakmayı keser,senkron bakması receive için gereklidir

break;

}

if(TransmitDataReady) //Data hazır mı diye bakar, hazırsa yollar.

{

CLRWDT();

InitBoardTransmitter(); //kart sürekli reciever modda olduğu için yollamadan önce transmittera alır.

CheckSum=0;

TransmitDataReady=0;

SendSencron(); //Preamble+Senkron yollar

SendData(i); //Toplam Byte sayısını yollar

for(j=0;j<i;j++) //Sonra Datayı yollamaya başlar

{

if(RecievedChar[j]!='>') //Sona gelmediyse normal yollama işlemini yapar, her yolladığı karakteri Checksum ile toplar

{

SendData(RecievedChar[j]);

CheckSum=CheckSum+RecievedChar[j];

}

else if(RecievedChar[j]=='>') //Sona geldiyse sonuncu karakteri yollar, sonrada Checksum modüle yollar, seri portada ready yollar

{

CheckSum=CheckSum+RecievedChar[j];

SendData(RecievedChar[j]);

SendData(CheckSum);

excess();

SendReady();

}

}

CLRWDT();

i=0;

InitBoardReciever(); //Sadece Transmit Data hazırsa tx moduna girer, normalde hep rx modunda bekler.

}

else //Sencron alınmıştır. Data okumaya başlanır.

{

RCIE=0;

CheckSum=0;

CLRWDT();

do

{

temp=DataRead();

if(i==0)

ByteCount=temp; //ilk aldığı karakter toplam byte sayısı olduğu için checksum a dahil etmez.

else

{

CheckSum=CheckSum+temp; //Son karakteri alana kadar data almaya devam eder.

RecievedChar[i]=temp;

}

i++;

}while(temp!='>' || i==1); //">" işaretinin ascii kodu 62 olduğundan, byte sayısı olarak 62 aldığında durmaması için i==1 de loopu kesmez.

RecievedChar[i]=DataRead(); //Son karakteri de aldıktan sonra Checksum almak için bir karakter daha alır.

CLRWDT();

if(CheckSum==RecievedChar[i]) //Alınan checksum ile hesaplanan Checksum eşit mi diye bakar.

{

if(ByteCount==(i-1)) //Karşı taraftan yollanan toplam byte sayısı ile kendi hesapladığı byte sayısını karşılaştırır.

{

//eğer checksum ve byte sayıları tutarsa,seri porta transmit işlemine başlar.

for(j=1;j<i;j++)

{

while(!TXIF && !TRMT); //Transmit işlemi bitene kadar, yani tx ready flagi 1 olana kadar bekler.

TXREG=RecievedChar[j]; //seri porta tektek aldığı karakterleri basar.

}

SendReady(); //gönderme işlemi bittikten sonra seri porta ready basar.

ByteCount=0; //bir sonraki işlem için byte sayısını 0 yapar.

}

}

i=0;

RCIE=1; //seri porta transmit işlemi bittiği için recieve interruptu aktif eder

}

}

}

void InitBoard()

{

CLRWDT();

OPTION=0x8E;

TRISA=0x01; // başlangıçta receiver konumunda olduğu için, RA0 input durumundadır.

PORTA=0x00;

RA5=EepromRead(); //Kanal Seçmek için eeprom u okur.

ADCON1=0x07;

TRISC6=1;

TRISC7=1;

GIE=1;

PEIE=1;

BRGH=1;

RCIE=1;

TXIE=0;

SPBRG=25; // Baud Rate = 9600

SYNC=0;

TXEN=1;

CREN=1;

SPEN=1;

T1CON = 0b00110000;

TMR1IE=0;

}

void InitBoardTransmitter()

{

CLRWDT();

TRISA=0x00;

RA4=1; //4. bacak modülün tx/rx mod seçim bacağı olduğundan, modülü transmitter moda alır.

RCIE=0;

}

void InitBoardReciever()

{

CLRWDT();

TRISA=0x01; //4. bacağı 0 yaparak modülü reciever moduna alır.

PORTA=0x00;

RCIE=1;

}

void SendData(char RF1) // Clock yok. Bu yüzden gönderilecek data kodlanarak gönderilecektir. Göndereceğimiz 1 bit,

{ // bir kısa bir uzun olmak üzere iki parçadan oluşacak.

// Gönderilebilecek en kısa anlamlı parça uzunluğu 1/( RF modülün baudrate'i) dir. Bu durumda, 1/2400, 417 us'dir.

// Sonuç olarak, kısa parça 450 us, uzun parça 1 ms olarak gönderilecektir.

int Databit=8;

char TestBit;

CLRWDT();

while (Databit!=0) //8 bit(1 byte) yollar

{

TestBit= RF1 & 0x80;

RF1=RF1<<1;

if(TestBit == 0x80)

{

RA0=1; // Eğer gönderilecek bit 1'e eşitse, --- 1 ms '1', 450 us '0' --- gönderir.

DelayMs(1);

RA0=0;

DelayUs(225);

DelayUs(225);

}

else

{

RA0=1; // Eğer gönderilecek bit 0'a eşitse, --- 450 us '1', 1 ms '0' --- gönderir.

DelayUs(225);

DelayUs(225);

RA0=0;

DelayMs(1);

}

Databit--;

}

}

void excess() // Gönderilen datanın son biti yanlış okunuyor. Bu yüzden data sonuna fazladan, okunmayacak

{ // bir bit ( 0 veya 1, ne olduğu faketmez) ekliyoruz. Böylece datamız doğru okunuyor.

RA0=1;

DelayMs(1);

RA0=0;

DelayUs(225);

DelayUs(225);

}

void SendSencron() //Preamble + Senkron yollar.

{

CLRWDT();

SendData(0xAA); // Preamble = 5 byte AA

SendData(0xAA);

SendData(0xAA);

SendData(0xAA);

SendData(0xAA);

RA0=1; // Sencron 24 ms 1, 24 ms 0'dan oluşuyor.

DelayMs(24);

RA0=0;

DelayMs(24);

CLRWDT();

}

void EepromWrite(char data) //eeproma bilgi yazar.

{

CLRWDT();

while(WR);

EEADR=0xFF;

EEDATA = data;

EEPGD = 0;

WREN = 1;

GIE = 0;

EECON2 = 0x55;

EECON2 = 0xaa;

WR = 1;

WREN = 0;

GIE = 1;

EEIF = 0;

}

char EepromRead() //eeprom u okur

{

CLRWDT();

EEADR = 0xFF;

EEPGD = 0;

RD = 1;

return EEDATA;

}

char DataRead() // Bu fonksiyon, RFten gelen datayı okur. Gelen bir bit data, 1 ile başlar (bknz. SendData). Eğer gelen bit 1 ise,

{ // 1 uzun, 0 ise kısadır. Eğer gelen bit 0 ise, 1 kısa, 0 ise uzundur.

int k=0;

char RF0=0;

unsigned int timereg2=0;

while(k!=8) // Bir byte = 8 bit

{

RF0=RF0<<1;

do{ timereg2++;} while(RA0==1); // Datanın başındaki 1 in uzunluğuna bakılır.

if( timereg2 >=130) // Eğer uzun ise, 1 bilgisi gelmiştir.

RF0=RF0 | 0x01;

timereg2=0;

do{ timereg2++;} while(RA0==0); // 1'den sonra 0 gelir. Bunun uzunluğuna bakılır.

if( timereg2 >=120) // Eğer uzun ise, 0 bilgisi gelmiştir.

RF0=RF0 & 0xFE;

k++;

}

CLRWDT();

return RF0; // Fonksiyon bir byte bilgi döndürür.

}

int Sencron () // Bu fonksiyon, Sencron arar.

{

unsigned int timereg1=0;

do { timereg1++; } while(RA0==1); // Sencronun ilk kısmındaki 1'e bakar.

if(timereg1>2000 )

{

timereg1=0;

do { timereg1++; } while (RA0==0); // Sencronun ikinci kısmındaki 0'a bakar.

if(timereg1>2000 )

{

return 1; // Sencron bulunduysa, fonksiyon geriye 1 döndürür.

}

}

CLRWDT();

return 0; // Sencron bulunmadığı sürece 0 döndürür.

}

void Reset() //watchdog timer'ı clear etmeyerek işlemciyi resetler.

{

PSA=0;

PS2=0;

PS1=0;

while(1);

}

void SendReady() //RS232'ye <READY> mesajı yollar.

{

int x;

x=RCIE;

RCIE=0;

for(k=0;k<7;k++)

{

CLRWDT();

while(!TXIF && !TRMT);

TXREG=Ready[k];

}

RCIE=x;

}

void Error() //Seri porta Genel Hata mesajı(<ERROR>) yollar.

{

for(k=0;k<7;k++)

{

CLRWDT();

while(!TXIF && !TRMT);

TXREG=Err[k];

}

TMR1IF=0;

Reset();

}

void Overflow() //40 den fazla byte girildiğinde seri porta hata(<OVERFLOW>) yollar{

{

for(k=0;k<10;k++)

{

CLRWDT();

while(!TXIF && !TRMT);

TXREG=Ovf[k];

}

TMR1IF=0;

Reset();

}

void TimeOut() //iki byte arasında 500ms den fazla süre geçerse seri porta hata(<TIME OUT>) basar.

{

for(k=0;k<10;k++)

{

CLRWDT();

while(!TXIF && !TRMT);

TXREG=TimeO[k];

}

TMR1IF=0;

Reset();

}

void interrupt recieve() // Seriporttan her data gelişinde interrupta girer, datayı RecievedChar[] dizisine kaydeder.

{

char temp;

if(i>40)

Overflow();

CLRWDT();

if(RCIE && RCIF) //Yeni data için mi interrupta girildiğini kontrol eder.

{

RCIE=0;

temp=RCREG;

TMR1IE=1; //seri porttan datayı aldığı için timer interruptları aktif eder.

if(temp=='<' && StartRecieve==0) //Data "<" ile başlıyor mu diye kontrol eder.

StartRecieve=1;

if(!StartRecieve) //Başlamıyorsa hata verir.

Error();

else if(StartRecieve) //Başlıyorsa datayı almaya başlar.

{

CLRWDT();

TMR1H=0; //Data almaya başladığı için timer'ı sıfırlar ve zaman tutmaya başlar.

TMR1L=0;

TMR1ON=1;

if(RecievedChar[i-1]=='C') //gelen datanın kanal değiştirme komutu olup olmadığına bakar.

{

TMR1ON=0;

if(temp != 0x30 && temp != 0x31) //kanal değiştirme komutundan bir sonraki komuta bakar, alınan değer 0 veya 1 değilse,

Error(); //istenilen kanal numarası mevcut olmadığı için hata verir.

else //Hata yoksa kanalı değiştirir, yeni kanal numarasını eeproma kaydeder.

{

if(temp==0x30) //eğer gelen sayı 0 ise, eeproma 0 yazar,

{

TXREG='0';

EepromWrite(0x00);

}

else

{

TXREG='1'; //eğer gelen sayı 1 ise, eeproma 1 yazar.

EepromWrite(0xFF);

}

RA5=EepromRead(); //5. bacak kanal seçme bacağı olduğundan eepromdaki değeri 5. bacağa atar.

CheckSum=CheckSum-RecievedChar[i-1]; //Checksum a kanal değiştirme komutunu dahil etmemk için kanal değiş komutunu checksumdan çıkartır.

i=i-1; //kanal değiştirme komutunu toplam byte sayısına dahil etmemek için 1 çıkartır.

}

}

else //kanal değiştirme komutu değilse, normal aldığı datayı buffera yazar.

{ //aynı zamanda iki data alışı arasındaki süreyi ölçmek için zaman tutar.

TMR1H=0;

TMR1L=0;

RecievedChar[i]=temp;;

if(temp == '>') //eğer son karakter ise data gönderilmeye hazır olduğu için TransmitDataReady'i 1 yapar.

{

TMR1IE=0; //Son karakteri aldığı için timer interruptları kapatır.

TMR1ON=0;

TransmitDataReady=1; //Datanın transmite hazır olduğunu belirtir.

StartRecieve=0; //Bir sonraki data almaya başladığında tekrardan ilk karakterin < olmasını düzgün kontrol edebilmek için sıfırlar.

}

i++;

}

}

CLRWDT();

RCIE=1; //recieve interruptu tekrar aktif eder.

RCIF=0; //interrupt flagini siler.

}

else if(TMR1IF) //eğer interrupta zaman aşımından dolayı girdiyse, time out hatası verir.

{

TMR1ON = 0;

TimeOut();

}

}