

```

/*****
|                                     |
|                               MAIN |
|                                     |
|*****|
void main(void){
    unsigned char i;
    Initialise();
    #asm("sei")

    LogFile[LogLinePointerRead][0]=0;
    Kapcsolobemenetek = ~( ((PINC&0xFC)>>1) | ((PINA.2)<<7) | PIND.7 );
    for (i=0;i<5;i++) ADC_Tomb[i]=0;
    for (i=0;i<IDCount;i++) FogadTomb[i]=0;
    FogadTomb[IDDugattyuPOZ]=0xFF;

    BuzzerBE();
    PORTB = 0xFF;
    delay_ms(100);
    BuzzerKI();
    PORTB = 0x00;

    while (1){
        Kapcsolobemenetek = ~( ((PINC&0xFC)>>1) | ((PINA.2)<<7) | PIND.7 ); // Kapcsolobemenetek beolvasása
        PORTB = FogadTomb[IDVezzerlesiAllapot]; // LED-ek beállítása a változónak megfelelően
        ADCCalculations(); // ADC értékek feldolgozása, átváltása
        if (!DelayMS){ // Alacsony frekvenciás késleltetés delay nélkül
            //...bármilyen...
            DelayMS=1000; // Késleltetés időtartama (Timer0 inkrementálja)
        }
    }
}

/*****
|                                     |
|                               ADC és feldolgozó |
|                                     |
|*****|

#define ADC VREF TYPE 0x40
interrupt [ADC_INT] void ADC_Interruption(void){
    unsigned char i;
    if(ADC_Scrap++){
        ADC_Tomb[ADC_Loop] += ADCW;
        if (++ADC_Loop >= 5){ ADC_Loop=0; ADC_AtlagDB++; }
        ADMUX = ADC_VREF_TYPE + ADC_Loop + 3;
        ADC_Scrap = 0;

        if(ADC_AtlagDB >= 40){
            for (i=0; i<5; i++){
                ADC_OsszegTomb[i] = ADC_Tomb[i];
                ADC_Tomb[i]=0;
            }
            ADC_Szamold = 1;
            ADC_AtlagDB = 0;
        }
    }
}

}

```

```

unsigned char Feny (unsigned int Feny40value){
    Feny40value /= 40;
    Feny40value = 1023-Feny40value;
    Feny40value *= 49;
    Feny40value /= 110;
    Feny40value >>=1;
    return Feny40value;
}

// 40 összeadott érték átlaga
// Nem a diódát, hanem a pozitív oldali ellenállást mérjük
// Ellenállás feszültsége 0,1mV-ban
// I=U/R | R=100K -> [0,1uA] | R=1K -> [10uA]
// [uA] -> [lux]
// [lux] Fényerősség lux-ban

unsigned int Homero (unsigned int ADC40value){
    float Arany;
    Arany=(float)ADC40value/(40*1024-ADC40value);
    Arany*=1.363636;
    Arany=log(Arany);
    Arany*=2.744032e-04;
    Arany+=3.354016e-03;
    Arany=100/Arany;
    return Arany;
}

// Arany=R/Rin (40 összeadott érték átlaga)
// 30k az R1, 22k az NTC 25°C-os ellenállása (1.363636 = 30/22)
// Adatlapból vett érték. 100k-s NTC esetén: 2.460382e-04;
// Adatlapból vett érték. 100k-s esetén is ennyi
// Reciprok és szorzás 100-al
// [0,01K] Érték visszaadása század kelvinben

unsigned char Gazkar (unsigned int Gazkar40value){
    Gazkar40value /= 40;
    if (Gazkar40value>840) Gazkar40value = 840-174;
    else if (Gazkar40value>174) Gazkar40value -= 174;
    else Gazkar40value=0;
    Gazkar40value *= 51;
    Gazkar40value /= 133;
    return Gazkar40value;
}

// 40 összeadott érték átlaga
// Gázkar itt éri el a maximumát (4,25V)
// Gázkar minimuma (0,85V)
// Alatta 0-nak vesszük
// Átkonvertálás 0-255-ös skálára:
// *255/(840-174) = *51/133
// Gázkar pozíció 0-255

void ADCCalculations(void){
    if (ADC_Szamold){
        HomersekletBE = Homero(ADC_OsszegTomb[ADHomOsszegBE]);
        HomersekletKI = Homero(ADC_OsszegTomb[ADHomOsszegKI]);
        HomersekletDoboz = Homero(ADC_OsszegTomb[ADHomOsszegDoboz]);
        FenyeroDoboz = Feny(ADC_OsszegTomb[ADFenyOsszeg]);
        GazkarPozicio = Gazkar(ADC_OsszegTomb[ADGazkarPozOsszeg]);
        ADC_Szamold = 0;
    }
}

// ADC értékek feldolgozása, átváltása
// Van 40 új mérés, számolni kell

/*****
|                               |
|          Buzzer              |
|******/

const unsigned char UtemHossz=40;
unsigned char HangLoop;
unsigned char HangUtem;
unsigned int HangSav;

// Egy ütem hossza [ms]-ben

interrupt [TIM0_COMP] void Timer0_1kHz(void){
    #asm("sei")
    #asm("wdr")
    BuzzerKimenet();
    if (DelayMS) DelayMS--;
}

```

```

void HangUzenet(unsigned int HangSav1){
    HangUtem=0;
    HangLoop=UtemHossz-1;
    HangSav=HangSav1;
}

void BuzzerKimenet(){
    if (HangSav){
        if ((++HangLoop)==UtemHossz){ HangUtem++; HangLoop=0; }
        if (HangUtem<=16){
            if ((HangUtem)&&(HangSav&(1<<(16-HangUtem)))) BuzzerBE();
            else BuzzerKI();
        }else{
            BuzzerKI();
            HangSav=0;
        }
    }
}

/*****
|                               |
|           TWI (I2C)         |
|                               |
*****/

unsigned char TWIProgress;
unsigned int  TWIReturnValue;
unsigned char TWIUtasitas;

void TWIError(){
    ERROR(6);
    HibaKodok &= 0x00FF;
    HibaKodok |= (unsigned int)TWSR<<8;
    TWIProgress=0;
}

unsigned char LogByteToSend;
unsigned char IsLogToSend(){ // Megvizsgálja van-e logfájl bejegyzés a pufferben, és bekészíti a küldendő bájtot LogByteToSend változóba
    // [0] Épp nincs küldendő bejegyzés [1] Van küldendő, de ez a sor utolsó eleme [2] Van küldendő, nem utolsó [3] Van küldendő, új sor kezdete
    unsigned char LogLinePointerReadNext;
    if (LogCharPointerRead<LogFile[LogLinePointerRead][0]){
        LogByteToSend=LogFile[LogLinePointerRead][LogCharPointerRead++];
        if (LogCharPointerRead==LogFile[LogLinePointerRead][0]) return 1;
        else return 2;
    }else{
        LogLinePointerReadNext=LogLinePointerRead+1;
        if (LogLinePointerReadNext>=LogLineCount) LogLinePointerReadNext=0;
        if (LogLinePointerReadNext==LogLinePointerWrite){
            return 0;
        }else{
            LogLinePointerRead=LogLinePointerReadNext;
            LogByteToSend=LogFile[LogLinePointerRead][0];
            LogCharPointerRead=1;
            return 3;
        }
    }
}
}

```

// Hangüzenet lejátszásának indítása
// Ütem nullázása
// Így a köv timer órajelnél indul is a lejátszás
// Hangsáv átadása

// Ha van lejátszandó hangsáv
// Ütem időtartama letelt, léptetés a következő ütemre
// Ha a lejátszandó hangsávnak még nincs vége
// Bitnet megfelelően be/kikapcsolása a buzzernek

// Utolsó ütem vége
// Hangsáv törlése

// TWI hiba
// TWI előző hibakódjának törlése
// Új hibakód másolása a HibaKodok felső bájtjába
// Aktuális TWI utasítás félbeszakítása, ugrás a ciklus elejére

// Ha az adott sorból még van mit elküldeni
// Bájt bekészítése
// [return 1] A sor utolsó elemét küldjük
// [return 2] Lesz még több eleme is a sornak
// Nincs küldés alatt lévő sor
// Következő sor
// Tömb végén ugrás a 0. elemhez
// Következő sorban még nincs új adat (éppen oda ment/fog menteni)
// [return 0] Nincs küldendő logfájl
// Van sor amit el kell küldeni
// Következő sor pointerének aktiválása
// Sor első bájtjának bekészítése küldésre
// Pointer resetelése és léptetése 1-re
// [return 3] Új sor első eleme

```

void TWIReturn(){
    if (TWIUtasitas<IDCount) TWIReturnValue=FogadTomb[TWIUtasitas]; // Változó kiolvasás ID alapján
    else switch (TWIUtasitas-IDCount){ // Vezérlőtől (RS485) származó változók lekérdezése
        // Saját adatok lekérdezése
        case IDTempomatSebesseg: TWIReturnValue=TempomatSebesseg; break;
        case IDGazkarPozicio: TWIReturnValue=GazkarPozicio; break;
        case IDKapcsolobemenetek: TWIReturnValue=Kapcsolobemenetek; break;
        case IDEnkoderTemp: TWIReturnValue=EnkoderTemp; break;
        case IDEnkoderTemp: EnkoderTemp=0; break;
        case IDEnkoderButton: TWIReturnValue=EnkoderButton; break;
        case IDHomersekletBE: TWIReturnValue=HomersekletBE; break;
        case IDHomersekletKI: TWIReturnValue=HomersekletKI; break;
        case IDHomersekletDoboz: TWIReturnValue=HomersekletDoboz; break;
        case IDFenyeroDoboz: TWIReturnValue=FenyeroDoboz; break;
        case IDHibaKodok2: TWIReturnValue=HibaKodok; break;
        default: TWIReturnValue=0x0000; break;
    }
}

void TWIAdatErkezett(){ // Kapott integer eltárolása ID alapján
    switch ((TWIUtasitas&0x3F)-IDCount){
        case IDTempomatSebesseg: TempomatSebesseg=TWIReturnValue; break;
        case IDBuzzer: HangUzenet(TWIReturnValue); break;
    }
}

interrupt [TWI] void TWI_Interrupt(void){
    if ((TWSR&0xF8)==0x60) TWIProgress=10; // Own SLA+W has been received, ACK has been returned
    else{
        switch(TWIProgress){
            // Utasítás értelmezése
            case 11: if ((TWSR&0xF8)!=0x80) TWIError(); // Previously addressed with own SLA+W, data received, ACK returned
                     else{
                         TWIUtasitas=TWDR; // Utasítás kiolvasása TWI regiszterből
                         if (TWIUtasitas&0x80) TWIProgress=20; // Ha írni akar, ugrás a fogadás részhez
                         else if (TWIUtasitas&0x40) TWIProgress=30; // Ha adatot vár (logfile-t), ugrás ahhoz a részhez
                         else TWIReturn(); // Ha adatot vár (változót), előkészítjük az értékét küldésre
                     }
                     break;

            // Változó küldése
            case 12: if ((TWSR&0xF8)!=0xA0) TWIError(); // A STOP or repeated START condition received while still addressed as slave
                     break;
            case 13: if ((TWSR&0xF8)!=0xA8) TWIError(); // Own SLA+R has been received, ACK has been returned
                     else TWDR=(TWIReturnValue>>8); // Integer első felének küldése (felső bájt)
                     break;
            case 14: if ((TWSR&0xF8)!=0xB8) TWIError(); // Data byte in TWDR has been transmitted, ACK has been received
                     else TWDR=TWIReturnValue; // Integer második felének küldése (alsó bájt)
                     break;
            case 15: if ((TWSR&0xF8)!=0xC0) TWIError(); // Data byte in TWDR has been transmitted, NOT ACK has been received
                     break;

            // Változó fogadása
            case 21: if ((TWSR&0xF8)!=0x80) TWIError(); // Previously addressed with own SLA+W, data received, ACK returned
                     else TWIReturnValue=(unsigned int)TWDR<<8; // Adat első felét megkaptuk, tárolás
                     break;
            case 22: if ((TWSR&0xF8)!=0x80) TWIError(); // Previously addressed with own SLA+W, data received, ACK returned
                     else TWIReturnValue+=TWDR; // Adat második felét is megkaptuk, egyesítés és tárolás
                     break;
        }
    }
}

```

```

    case 23: if ((TWSR&0xF8)!=0xA0) TWIError(); // A STOP or repeated START condition received while still addressed as slave
            else TWIAdatErkezett(); // Fogadott értékének másolása az ID-nak megfelelő változóba
            break;

    // Logfájl küldése
    case 31: if ((TWSR&0xF8)!=0xA0) TWIError(); // A STOP or repeated START condition received while still addressed as slave
            break;
    case 32: if ((TWSR&0xF8)!=0xA8) TWIError(); // Own SLA+R has been received, ACK has been returned
            else{
                if (IsLogToSend()==3) TWDR=LogByteToSend; // Van küldendő logbejegyzés, küldés (értéke a sor hosszát adja ezzel együtt)
                else { TWDR=0; TWIProgress++; } // Éppen nincs több log bejegyzés, 0-s méret küldése, ugrás a lezárásra
            }break;
    case 33: if ((TWSR&0xF8)!=0xB8) TWIError(); // Data byte in TWDR has been transmitted, ACK has been received
            else{
                if (IsLogToSend()>1) TWIProgress--; // Még nem az utolsó byte, ne léptessük tovább a TWIProgress-t
                else TWIProgress++; // Utolsó bájt esetén NACK-ot várunk
                TWDR=LogByteToSend; // Bájt küldése
            }break;
    case 34: if ((TWSR&0xF8)!=0xB8) TWIError(); // Data byte in TWDR has been transmitted, ACK has been received
            else TWDR=0xFF; // Bármilyen küldése, hogy TWI master NACK-olhasson valamire
            break; // TWI masternek fogadás előtt el kell döntenie, hogy ACK/NACK, azért kell
    case 35: if ((TWSR&0xF8)!=0xC0) TWIError(); // Data byte in TWDR has been transmitted, NOT ACK has been received
            break;

    // Egyéb esetben hiba
    default: TWIError();
            break;
    }
}

TWCR=0xC5; // Interrupt feloldása
TWIProgress++; // TWI loop következő fázisba léptetése
}

/*****
|                               RS 485                               |
*****/

void Transmit(void){
    delay_us(50); // Megvárjuk míg a vezérlő átadja a csatornát. 10 us-el már megy, de a késések miatt szkópon az csak csak 1-2 us (necces)
                  // Azért kell, mert különben az UART várja a hosszú szünet "nullájának" stopbitjét (ami akár milyen hosszú is lehet)
    if (UCSRA & (1<<UDRE)){ // Szabad az UART
        PORTD.2 = 1; // RS485 Adó mód
        switch(ReceiveID){ // Kiválasztott adat küldése
            case IDGazkarPozicio : UDR = GazkarPozicio; break;
            case IDKapcsolobemenetek : UDR = Kapcsolobemenetek; break;
            case IDTempomatSebesseg : UDR = TempomatSebesseg; break;
            default : UDR = 0x00; break;
        }
    }else ERROR(7);
}

interrupt [USART_TXC] void usart_tx_isr(void){
    PORTD.2 = 0; // RS485 Vevő mód
    Process=0; // Utasítás bájtot várunk
}

```

```

interrupt [USART_RXC] void usart_rx_isr(void){
    unsigned char Status;
    if ((Status = UCSRA) & RX_COMPLETE){
        if ((Status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0){ // Érkezett adat
            ReceivedByte = UDR; // és nem hibás
            if(ReceivedByte==0xFF) Szinkron++; // Szinronizáló karakterek számolása
            else Szinkron=0;
            if(Szinkron==6){ // 6. 0xFF érkezett
                if (!SzinkronOK) ERROR(1); // Nem számítottunk szinkronizálásra, azaz csúszás volt
                Szinkron=SzinkronOK=0;
                Process=0; // Utasítás bájt következik
            }else if (!SzinkronOK){ // Ha szinkronizálás nincs folyamatban
                if(!LogFogadas){ // Nincs logfájl fogadás folyamatban
                    switch(Process){
                        case 0: if (ReceivedByte & 0x80){ // Változó fogadás/küldés
                            if (ReceivedByte & 0x40){ // Küldeni kell
                                if (ReceivedByte==0xFF) SzinkronOK=1; // Szinronizálás történik
                                else{
                                    ReceiveID=ReceivedByte&0x3F; // Változó ID tárolása
                                    Transmit(); // Küldés
                                }
                            }else ReceiveID=ReceivedByte&0x3F; // Fogadandó változó azonosítójának tárolása
                            Process++;
                        }else{ // Logfile fogadása
                            LogFogadas=1;
                            LogCharPointerWrite=0; // Sor elejére ugrás
                            ReceivedTemp=ReceivedByte&0x3F; // Logfájl csomag méretének meghatározása
                            if (LogLinePointerWrite==LogLinePointerRead){ // Még nem továbbította, de már íránk felül
                                LogWriteDisabled=1; // Letiltjuk, hogy felülírhasssa
                                ERROR(2);
                            }else LogWriteDisabled=0;
                        }break;
                    }
                    // Csak változó fogadás esetén:
                    case 1: ReceivedTemp=ReceivedByte; // Felső adatbájt mentése temp-be
                            Process++;
                            break;
                    case 2: if (ReceiveID<IDCount) // Érvényes ID esetén
                                FogadTomb[ReceiveID]=(unsigned int) (ReceivedTemp<<8)+ReceivedByte; // Változó mentése
                                else ERROR(5);
                                Process=0; // Ciklus nullázása, új utasítás következik
                                break;
                }
            }
        }
        if (LogFogadas){ // Log file fogadás
            if (!LogWriteDisabled){ // Nincs letiltva a mentés
                if (LogCharPointerWrite>=LogCharSize) ERROR(3); // Hosszabb a kapott logfile mint a puffer
                else LogFile[LogLinePointerWrite][LogCharPointerWrite]=ReceivedByte; // Mentés a pufferbe
            }
            if (++LogCharPointerWrite==ReceivedTemp){ // Temp: méret => Utolsó adatot is megkaptuk
                if (!LogWriteDisabled) // Ha nem volt letiltva a mentés
                    if (++LogLinePointerWrite>=LogLineCount) LogLinePointerWrite=0; // Pointer léptetése
                LogFogadas=0; // Bejegyzés vége, ciklus nullázása, új utasítás következik
            }
        }
    }
}
}
}

```