RUGÁSI KRISZTIÁN

PROJEKTFELADAT MECHATRONIKUSOKNAK
(BMEVIAUM039)
2020/21-1

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Rugási Krisztián

Ultrahangos távolságmérés mikrokontrollerrel

Konzulens:
Dr. Iváncsy Szabolcs

BUDAPEST, 2020

Tartalom

1. A FELADAT ISMERTETÉSE

A feladat egy mikrokontroller által vezérelt ultrahangos távolságmérő megtervezése és megvalósítása. A rendszer egy központi és egy kiegészítő egységből áll. A központi egység végzi a méréseket, kijelzi és eltárolja az adatokat, míg a kiegészítő egység a központi egység által mért távolság adatokat jeleníti meg, és bizonyos távolság alatt jelzést ad. A két egység rádión keresztül kommunikál egymással.

A távolságmérés pontosítása érdekében a hangsebesség hőmérsékletfüggését is figyelembe kell venni, így a hőmérsékletet is mérni kell.

További követelmények az egyes egységekkel szemben:

-*Központi egység*:

- A központi egységnek elemről vagy külső tápegységről is képesnek kell lennie működni.
- A mért adatok kijelzésére egy LCD kijelzővel kell rendelkeznie.
- A mért távolság adatokat egy EEPROM-ban kell eltárolnia, és azoknak USB-n keresztül kiolvashatóknak kell lenniük.
- Esetleges továbbfejlesztéshez valamilyen kivezetéssel kell rendelkeznie (pl.: SPI, I2C)
- Nyomógombok és LED-ek a felhasználóval való kommunikációhoz.

-*Távoli egység*:

- A távoli egységnek elemekről kell működnie.
- A távolságadatok kijelzésére nagyobb hétszegmenses kijelzővel vagy LED mátrix kijelzővel kell rendelkeznie.
- Nyomógombok és LED-ek a felhasználóval való kommunikációhoz.

2. A HARDVER LEÍRÁSA

2.1. A központi egység

2.1.1. TÁPELLÁTÁS

A központi egység elemekről vagy külső tápegységről is működtethető. Elemek esetén az eszköz 3 db sorosan kapcsolt 1,5 V-os elemről, vagy külső tápellátás esetén egy USB portról működik. Mivel az eszközön található IC-k 3 és 3,6 V közötti tápfeszültséggel működnek, az elemek feszültségét (vagy a külső tápfeszültséget) egy MIC6365-3.3YD5-TR típusú LDO lineáris feszültségstabilizátor alakítja 3,3 V-ra.

2.1.2. A FELHASZNÁLÓI FELÜLET

Az eszköz felhasználói felülete a következő elemekből áll:

- 2 db LED
- 2 db nyomógomb
- 1 db háromállású kapcsoló
- egy MCCOG21605B6W-FTPLWI típusú 2x16 karakteres LCD kijelző

Az egyik LED (PWR jelöléssel) a tápellátó áramkör részeként mindig világít ha az eszköz be van kapcsolva (megfelelő tápellátás mellett). A másik LED (MODE jelöléssel) a távoli egységgel való rádiókommunikáció során villog.

A két gomb egyike egy reset gomb, míg a másik gomb jelenleg nem szolgál semmire, esetleges továbbfejlesztéshez használható.

A háromállású kapcsoló az eszköz ki-be kapcsolására szolgál.

Az LCD kijelző jeleníti meg a mért távolságértékeket milliméterben és a mért hőmérsékletet is.

2.1.3. SZENZOROK

A hőmérséklet mérésére egy MCP9700T típusú termisztor szolgál, amely -40 és 125 °C között képes mérni a hőmérsékletet, kalibráció után tipikusan 1 °C pontossággal. A termisztornak egy analóg kimenete van, amely közvetlenül a mikrokontroller egy ADC bemenetére van kötve.

A távolság mérésére egy RCWL-1601 típusú ultrahangos távolságmérő szenzor szolgál. A szenzor egyszerűen használható, tartalmazza az adót, a vevőt, és a hozzájuk tartozó vezérlőáramkört is. A távolságméréshez csak egy rövid impulzust kell küldeni a Trigger bemenetre, ami után a visszaverődő jel hatására az Echo kimenet az eltelt idővel arányos

ideig lesz magas szinten. A hanghullámok kiküldése és visszaérkezése közötti idő kétféleképpen is mérhető: lehet mérni a Trigger jel és az Echo jel felfutó éle közti időt, vagy az Echo jel magas szintjének idejét is.

A szenzor körülbelül 10 és 5000 mm közötti távolságok mérésére képes, a pontossága pedig néhány milliméter.

2.1.4. Az adatok tárolása

A mért távolságadatok tárolására egy M24256 típusú, 256 kbit-es EEPROM szolgál. Az EEPROM I2C buszon keresztül kommunikál a mikrokontrollerrel.

2.1.5. A mikrokontroller

A választott mikrokontroller egy PIC24FJ64GB004 16 bites mikrokontroller. Ez rendelkezik beépített USB, I2C, SPI kommunikációs interfészekkel, amelyek megkönnyítik a fejlesztést a későbbiekben.

Ezek mellett a beépített 10 bites AD átalakítójával a hőmérsékletmérés felbontása ~0.32 ℃ lesz, amely ebben az alkalmazásban megfelelő.

2.1.6. Programozás és egyebek

A mikrokontroller programozó lábai ki vannak vezetve, így az egyszerűen programozható az áramkörben.
Emellett az I2C busz vezetékei is ki vannak vezetve, amelyek az eszköz későbbi továbbfejlesztéséhez nyújtanak segítséget.

2.2. A távoli egység

2.2.1. Tápellátás

A távoli egység a központi egységtől eltérően csak elemekről működtethető, külső tápegységről nem. Továbbá az eszközön található alkatrészek egy tágabb tápfeszültségtartományban is képesek működni (2.4-3.6 V) mint a központi egység esetében, így akár 2 db sorban kapcsolt 1.5 V-os elemmel is működtethető. Ugyanakkor 3 elemre lett tervezve az eszköz, az elemek feszültségét egy MIC6365-3.3YD5-TR típusú LDO lineáris feszültségstabilizátor alakítja 3,3 V-ra. (Ez az LDO található a központi egységben is.)

2.2.2. Felhasználói felület

A távoli egység felhasználói kezelőfelülete a következő elemekből áll:

- 4 db LED
- 2 db nyomógomb
- egy háromállású kapcsoló
- egy LT0565SRWK típusú 3 számjegyes hétszegmenses kijelző

Az egyik LED (PWR jelzéssel) a tápellátó áramkör részeként mindig világít, ha az eszköz be van kapcsolva (megfelelő tápellátás mellett). Egy másik LED (MODE jelzéssel) a távoli egységgel való rádiókommunikáció során villog.

A fennmaradó két LED (WARN1 és WARN2 jelzésekkel) a mért távolságok egy adott határérték alá esésekor világítanak figyelmeztetésképpen.

A két gomb egyike egy reset gomb, míg a másik gomb jelenleg szolgál semmire, esetleges továbbfejlesztéshez használható.

A háromállású kapcsoló az eszköz ki-be kapcsolására szolgál.

A hétszegmenses kijelző jeleníti meg a mért távolságokat centiméterben. Mivel az ultrahangos távolságmérő szenzor mérési határai 1-500 cm, a három számjegy elegendő a távolság egész centiméterben történő kijelzésére.

A távoli egység a központival szemben nem szolgáltat semmilyen információt a mért hőmérsékletről, csak a távolságokat jeleníti meg.

2.2.3. Szenzorok

Az eszköz csak egy MCP9700T típusú hőmérsékletérzékelő szenzort tartalmaz (ez a hőmérő szenzor található a központi egységben is). Ez arra szolgál, hogy a két hőmérsékletszenzor által mért értékek átlagolásával pontosabban mérhessük a hőmérsékletet.

2.2.4. A mikrokontroller

A távoli egység gyakorlatilag ugyanazt a mikrokontrollert tartalmazza, mint a központi egység. Ez a PIC24FJ64GA004 típusú 16 bites mikrokontroller. (Ezek a mikrokontrollerek két változatban kaphatóak, egy A és B változatban. A két verzió csak annyiban különbözik, hogy az A jelzésű nem rendelkezik beépített USB kommunikációs interfésszel, míg a B jelzésű igen. Mivel a távoli egységnél nincs USB kommunikációra szükség, az A verziót használja.)

2.2.5. Programozás és egyebek

A mikrokontroller programozó lábai ki vannak vezetve, így az egyszerűen programozható az áramkörben.

2.3. Az egységek közti kommunikáció

A rendszer két egysége rádión keresztül kommunikál egymással. A két egység közti távolság egy tipikus (tolatás közbeni távolságmérés) alkalmazásban legfeljebb néhány méter, ebben a távolságban kell megfelelően működnie a rádiókommunikációnak. A kommunikáció során a központi egység az általa mért hőmérsékletet és a távolságot küldi el a távoli egységnek, amely visszaküldi az általa mért hőmérsékletet.

Egy MRF89XAM8A típusú rádió adó-vevő modul van beépítve mindkét egységbe a rádiókommunikációhoz. A modul néhány fontos tulajdonsága:

- a rádiókommunikációhoz szükséges összes elemet tartalmazza (az antennát is beleértve)
- alacsony áramszükséglet (3 mA vevő üzemmódban és 25 mA adás közben), így jól használható elemes alkalmazásokhoz
- 863-870 MHz-es tartományban működik
- alacsony adatátviteli sebesség (~40 kbps), ez azonban ebben az alkalmazásban megfelelő
- a kommunikáció maximális távolsága beltéri alkalmazásokban 60-70 m, kültéri alkalmazásokban akár >700 m is lehet, ami ebben az alkalmazásban megfelelő
- adóként és vevőként is képes üzemelni, amire szükség lesz

3. A PROGRAM LEÍRÁSA

3.1. A működés leírása

A rendszer bekapcsolt állapotban ciklikusan működik (fél másodperc egy ciklus), aminek a lépései:

1. Távolság mérése az ultrahangos szenzorral, ha nincs visszatérő jel (feltehetően mert a mérési tartományban nincs semmilyen tárgy), akkor a ciklusnak itt vége van, fél másodperc múlva mérünk megint
2. Hőmérsékletmérés és a mérési eredmények kiküldése rádión a távoli egységnek
3. A távoli egység megkapja az adatokat, kijelzi őket, hőmérsékletet mér, és annak eredményét visszaküldi a központi egységnek
4. A központi egység megkapja az adatokat és ezek alapján kijelzi az mérési eredményeket és elmenti őket az EEPROM memóriába

3.2. A központi egység

3.2.1. INICIALIZÁLÁS

A bekapcsolást követően az egység inicializálása történik meg, amely során inicializáljuk a mikrokontrollert (oszcillátor, pin-ek, timer modulok, ADC, SPI, stb.), az I2C buszt (nem a mikrokontroller beépített I2C interfészét használjuk), a rádió adó-vevő modult, az LCD kijelzőt, és az EEPROM memóriát.

3.2.2. A TÁVOLSÁG MÉRÉSE

A távolság mérésére a szenzor ECHO lábának magas szinten töltött idejét mérjük, majd abból az alábbi képlet szerint kaphatjuk meg a távolságot:

$$d = \frac{v_s}{2} * t \tag{1}$$

, ahol v_s a hang sebessége a hőmérsékletet is figyelembe véve:

$$v_s = 331{,}3 * \sqrt{1 + \frac{T_C}{273{,}15}} \tag{2}$$

, ahol T_c a hőmérséklet Celsiusban

A távolság számításánál a kettővel való osztás azért szükséges, mert az oda-vissza út megtételéhez szükséges időt mértük.

3.3. A távoli egység

3.3.1. INICIALIZÁCIÓ

A bekapcsolást követően az egység inicializálása történik meg, amely során inicializáljuk a mikrokontrollert (oszcillátor, pin-ek, timer modulok, ADC, SPI, stb.), a rádió adó-vevő modult, és a 7 szegmenses kijelzőt.

3.3.2. A 7 SZEGMENSES KIJELZŐ MŰKÖDÉSE

A 7 szegmenses kijelző vezérlését egy timer végzi, amelynek a periódusideje úgy van beállítva, hogy a kijelző frissítése frekvenciája körülbelül 30 Hz legyen. A timer interrupt rutin lépteti a kijelzőt, és jeleníti meg a kiírandó érték következő számjegyét.

3.4. Az egységek közti kommunikáció

A rádió adó-vevő modult az alábbi beállításokkal használjuk, amiket a modul inicializációja során állítunk be (csak a fontosabb beállításokat felsorolva):

- 863 MHz
- frekvenciamoduláció
- $f_{dev} = 30\ kHz$
- 4 byte méretű csomagokat küldünk
- az adatátviteli sebesség 2 kbit/s
- maximális erősítés (0 dB)
- 4 byte-os cím (0x53, 0x59, 0x4E, 0x43) (mindkét egységre)
- CRC modul engedélyezve

A rádió adó-vevő modulokat az állapottól függően kapcsolgatni kell adó, illetve vevő üzemmódok között. Ahhoz, hogy ezek időzítései ne okozzanak problémákat a modulok állapotait a következőképpen állítjuk be:

- A központi modul STANDBY üzemmódban van alapból, majd egy sikeres távolságmérés után adó módba állítjuk az adatok kiküldéséhez.
- Ha befejeződött az adatok kiküldése, azt a rádió modul IRQ0 interruptja jelzi, aminek hatására vevő üzemmódba állítjuk a modult. Ebben a módban várjuk a távoli egység válaszát. Ha megkaptuk a választ akkor STANDBY üzzemmódba állítjuk a modult a következő mérésig. Ha nincs válasz, akkor vevő üzemmódban marad a következő mérésig.

- A távoli egység rádió modulja alapállapotban vevő üzemmódban van

- Ha kapott egy csomagot a központi egységtől, azt az IRQ0 interrupt jelzi, aminek hatására átáll adó módba és elküldi a választ.
- A válasz elküldésének végét a rádió modul IRQ0 interruptja jelzi, aminek hatására visszaáll a modul vevő üzemmódba.

4. AZ ESZKÖZ MŰKÖDÉSÉNEK TESZTELÉSE

4.1. A távolságmérés pontossága

A távolságmérés pontosságának alakulása néhány teszt eredményével:

Távolság [mm]	Mérés eredménye [mm]	Eltérés [mm]
50	44	-6
100	106	+6
200	205	+5
300	303	+3
400	399	-1
500	498	-2
600	594	-6
700	694	-6
800	793	-7
900	896	-4
1000	995	-5
1500	1501	+1
2000	2005	+5
3000	3004	+4

A szenzor a tesztek alapján 10-5000 mm tartományban képes mérni a távolságot.

4.2. A hőmérsékletszenzorok kalibrálása

A hőmérsékletmérő szenzorok 21 °C-on lettek kalibrálva, a hőmérsékletet az ezen a hőmérsékleten mért feszültségtől való eltérés alapján számoljuk feltéve, hogy 10 mV eltérés a hőmérsékletszenzor kimenetén 1 °C eltérésnek felel meg (adatlap alapján).

Ezekkel a beállításokkal a szenzor nagyjából 1 °C pontossággal méri a hőmérsékletet (mint ahogy azt az adatlap is írja).

4.3. A rádiókommunikáció távolsága

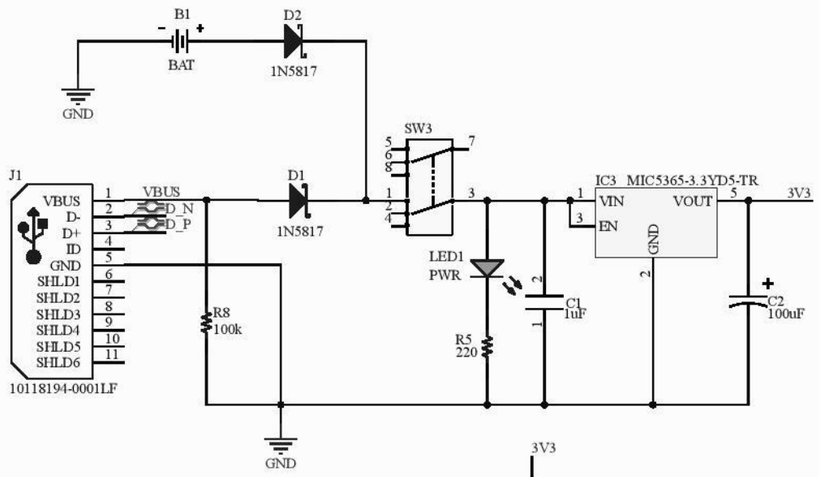
A rádiókommunikáció több távolságban és beltéri/kültéri körülmények között is tesztelve lett. A legnagyobb távolság amire a teszt el lett végezve: ~25 m úgy, hogy az adó és a vevő között több tárgy is volt (beleértve legalább 2 falat). Ezen a távolságon ~10 perc alatt 1 csomag/s adatátvitellel egy csomag sem veszett el.

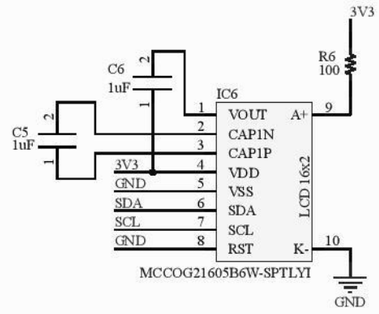
Mivel egy tipikus alkalmazásban a két egység maximum pár méterre lesz egymástól, ez megfelelőnek tekinthető. A tesztek alapján feltételezzük, hogy minden csomag megérkezik, hibakezelést elveszett csomagokra nem tartalmaz a program, ha esetleg mégis

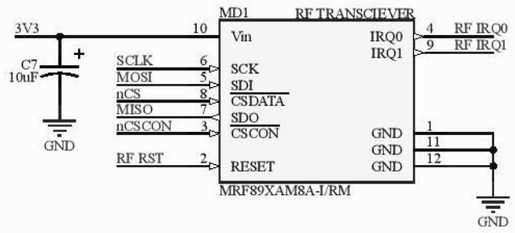
elveszne egy csomag, annak a hatása csak az lesz, hogy az adott fél másodperces mérési ciklus kimarad, nem kapunk új értékeket a kijelzőkön.

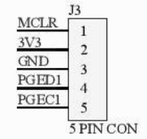
5. MELLÉKLETEK

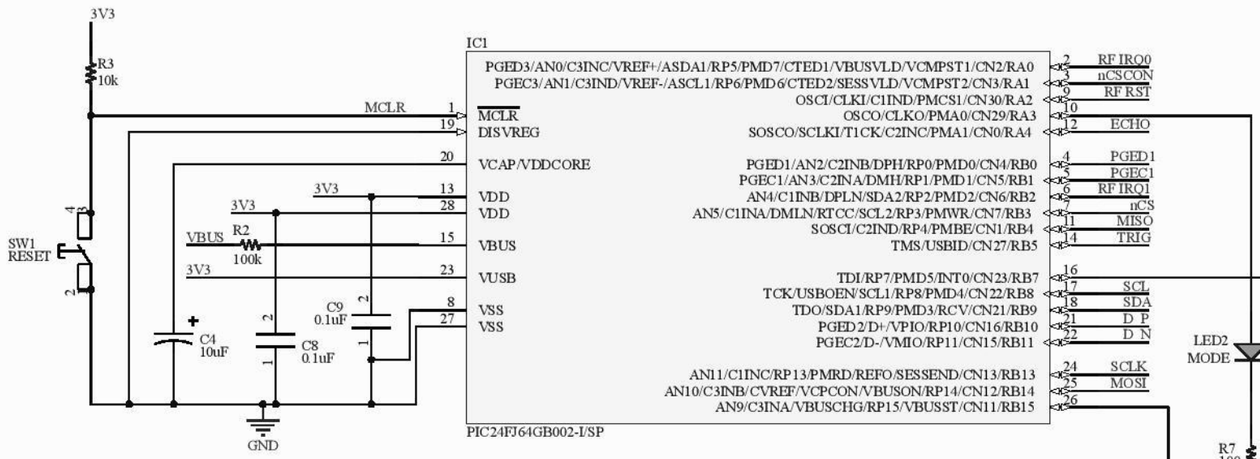
5.1. A kapcsolási rajzok

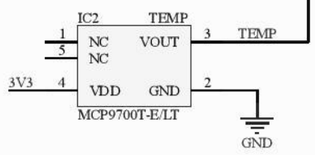


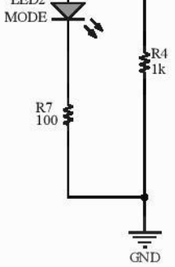


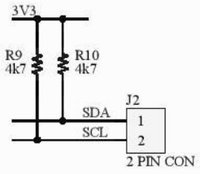


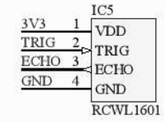


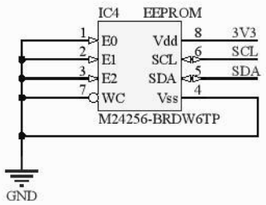




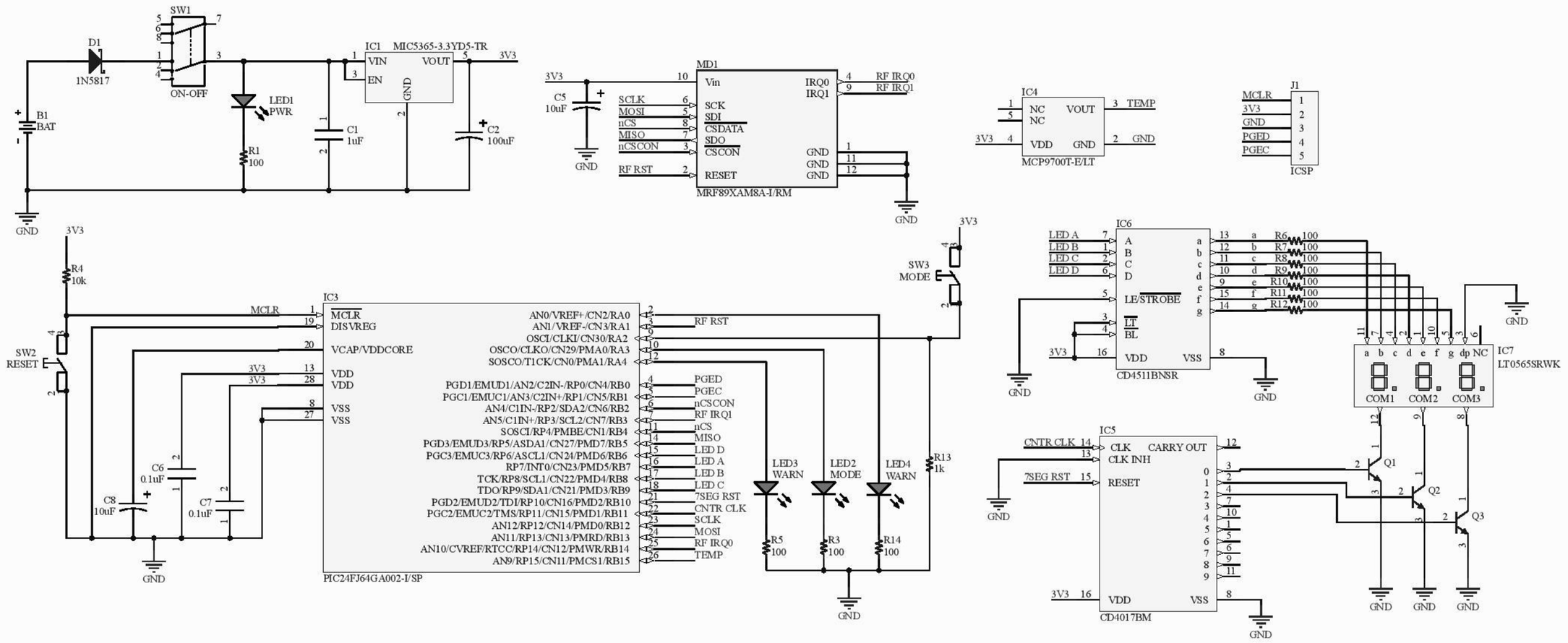






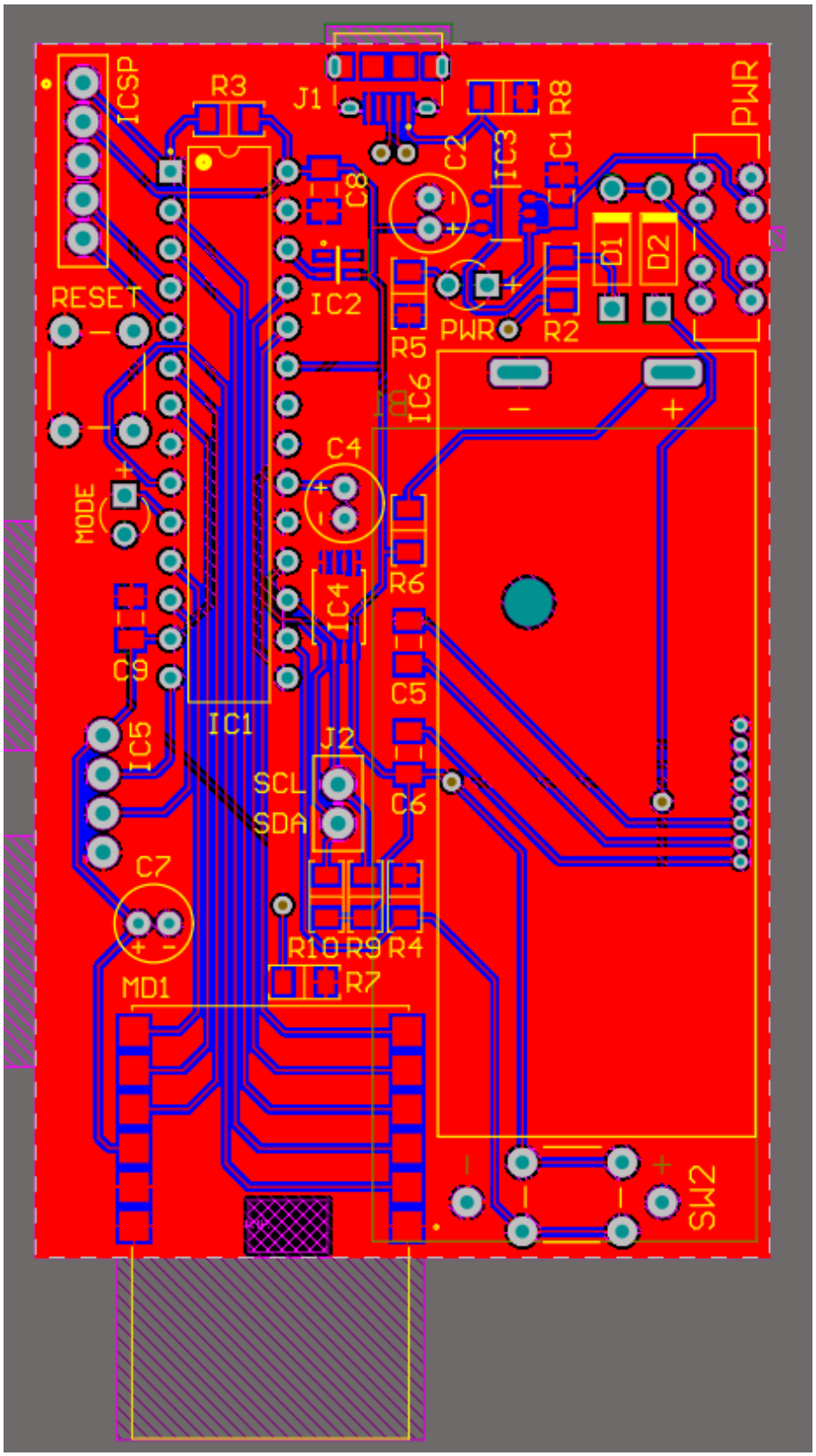


Title	*Ultrahangos távolságmérő központi egység*		
Size: A3	Number:1	Revision:1	
Date: 2020-12-13	Time: 21:01:57	Sheet 1 of 1	
File: C:\Users\Public\Documents\Altium\Projects\Ultrasonic distance sensor\Main.SchDoc			

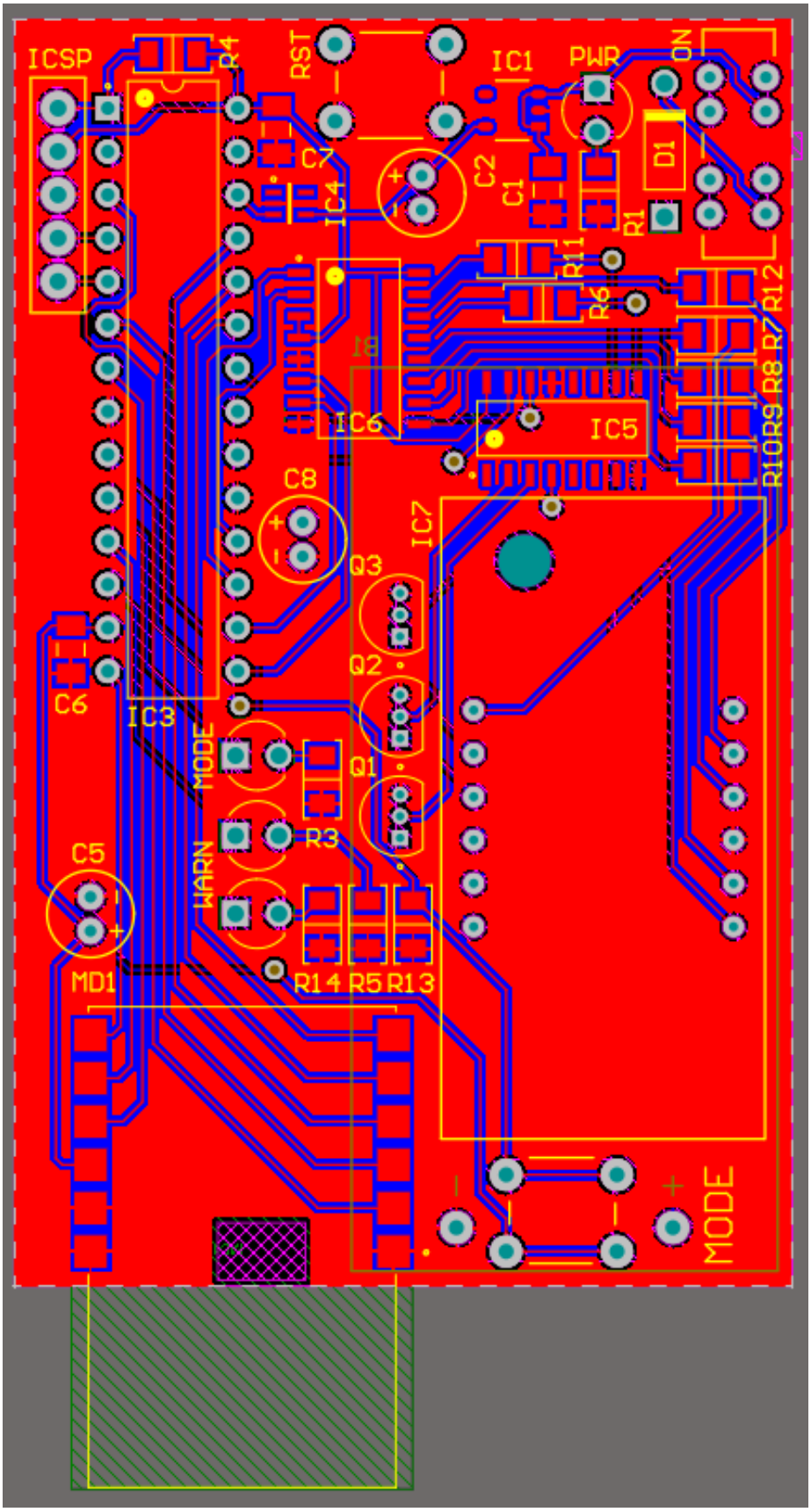


Title	Ultrahangos távolságmérő távoli egység	
Size: A3	Number	Revision
Date: 12-13-2020		Sheet of
File: C:\Users\..\TavoliEgyseg.SchDoc		Drawn By:

5.2. A PCB-k

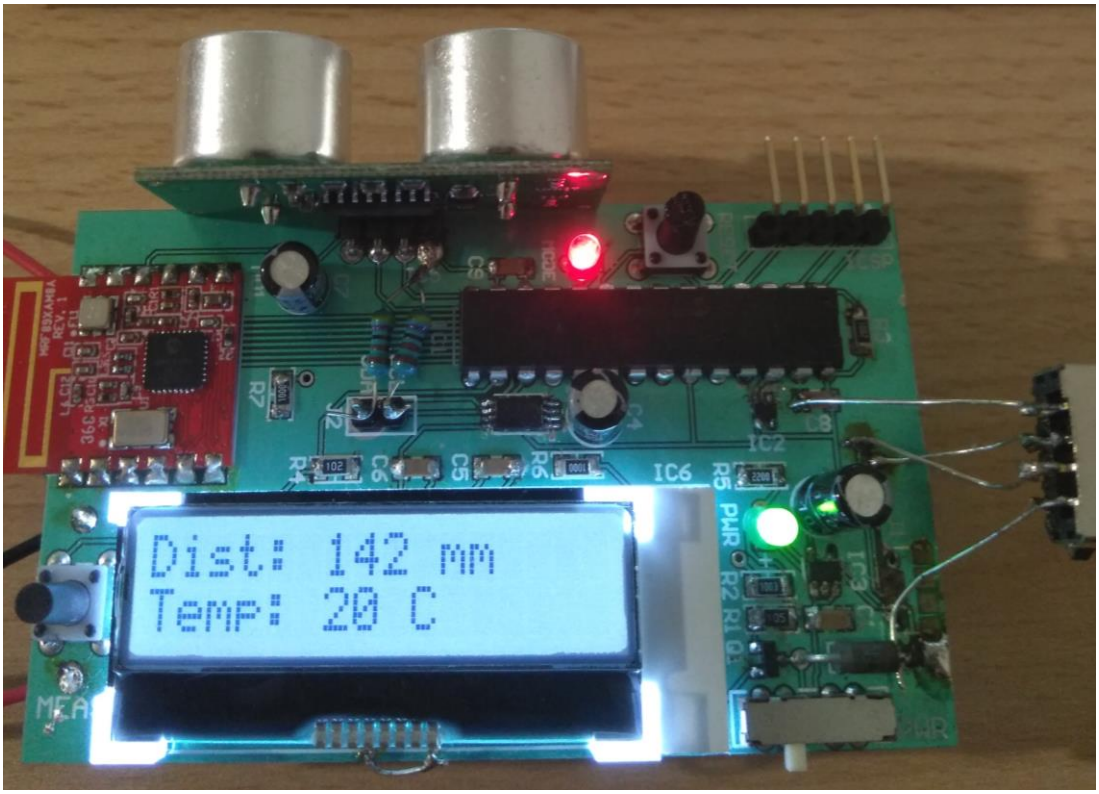


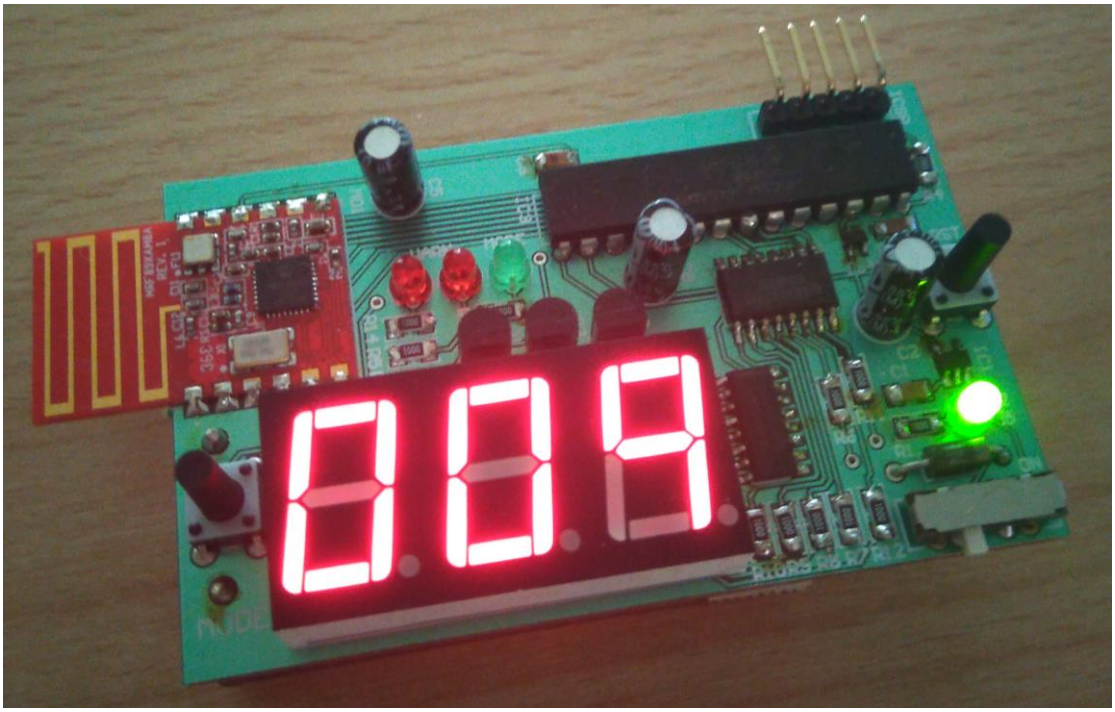
1. Központi egység



2. Távoli egység

5.3. Képek az eszközökről





5.4. A programok main.c fájljai

```
1 //Központi egység
  2
  3 #include "mcc_generated_files/mcc.h"
  4 #include "MRF89XAM8A.h"
  5 #include "LCD.h"
  6 #include "i2c.h"
  7 #include "eeprom.h"
  8 #include <math.h>
  9 #include <string.h>
 10
 11 #define FCY (_XTAL_FREQ / 2)
 12 #include "libpic30.h"
 13
 14 //device mode
 15 volatile bool MODE = 0;
 16 #define switchMode() (MODE ^=1)
 17
 18 //measurement data
 19 volatile bool SIGNAL;
 20 volatile float DIST = 0;
 21 volatile uint16_t TIME = 0;
 22 volatile float TEMP_MAIN = 0;
 23 volatile float TEMP_EXTERN = 0;
 24
 25 //radio mode
 26 volatile RF_MODE radio_mode = MODE_TX;
 27
 28 //EEPROM
 29 volatile uint16_t EEPROM_CURRENT_ADDR = 0;
 30
 31 //USB
 32 static uint8_t readBuffer[64];
 33 static uint8_t writeBuffer[64];
 34 #define USB_EEPROM_READ_REQUEST 0x7A
 35
 36 //functions
 37 void IO_InterruptHandler(void);
 38 uint16_t readADC(void);
 39 float readTemperature(void);
 40 float readTemperatureFiltered(uint8_t smp_num);
 41 void TriggerPulse(void);
 42 inline void incrementEEPROMAddress(void);
 43 char NumToChar(uint8_t num);
 44 void USBtransfer(void);
 45
 46 int main(void)
 47 {
 48     //initialize the device
 49     __delay_ms(10);
 50     SYSTEM_Initialize();
 51     I2C_Init();
 52     TMR2_Start();
 53
 54     //RF module init
 55     RF_Initialize();
 56     radio_mode = MODE_STANDBY;
 57     RF_SetMode(radio_mode);
 58     RF_SetFIFOThreshold(4);
 59     RF_SetPacketSize(4);
 60
 61     //LCD init
 62     LCD_Initialize();
 63     LCD_Home();
 64     LCD_PutChar('D');
 65     LCD_PutChar('i');
 66     LCD_PutChar('s');
```

```
 67      LCD_PutChar('t');
 68      LCD_PutChar(':');
 69      LCD_PutChar(' ');
 70      LCD_SetCursor(0x40);
 71      LCD_PutChar('T');
 72      LCD_PutChar('e');
 73      LCD_PutChar('m');
 74      LCD_PutChar('p');
 75      LCD_PutChar(':');
 76      LCD_PutChar(' ');
 77
 78      //init temp
 79      TEMP_MAIN = readTemperature();
 80      TEMP_EXTERN = TEMP_MAIN;
 81
 82      //get eeprom address
 83      EEPROM_CURRENT_ADDR = ((uint16_t)EEPROM_RandomRead(0x00, 0x00) << 8) |
(uint16_t)EEPROM_RandomRead(0x00, 0x01);
 84
 85      CN_SetInterruptHandler(IO_InterruptHandler);
 86
 87      while (1){
 88
 89          //send trigger
 90          SIGNAL = false;
 91          TriggerPulse();
 92          __delay_ms(60);
 93
 94          if(SIGNAL && TIME < 40000){
 95              //transmit mode
 96              radio_mode = MODE_TX;
 97              RF_SetMode(MODE_TX);
 98              RF_SetFIFOThreshold(2);
 99              //send time
100              WriteRFTransmitFIFO(TIME >> 8);
101              WriteRFTransmitFIFO(TIME);
102              //send temperature
103              int16_t temp = (int16_t)((TEMP_MAIN + 40) * 10); //(T+40)*10 in int
104              uint16_t tempu = (uint16_t)temp;
105              WriteRFTransmitFIFO(tempu >> 8);
106              WriteRFTransmitFIFO(tempu);
 107
108              USBtransfer();
109              __delay_ms(450);
110          }
111
112      }
113
114      return 1;
115  }
116
117  void IO_InterruptHandler(void){ //ultrasonic sensor, radio, button interrupts
are handled here
118
119      uint16_t porta = PORTA;
120      uint16_t portb = PORTB;
121
122      bool ECHO = porta & 16;
123      bool RF0 = porta & 1;
124      bool RF1 = portb & 4;
125      bool MEAS = portb & 128;
126
127      //echo signal interrupt
128      if(ECHO){
129          TMR2 = 58; //offset error correction
130          while(ECHO_GetValue()); //distance~ECHO_HIGH
131          TIME = TMR2; //time in us
132          TEMP_MAIN = readTemperature();
```

```
133         SIGNAL = true;
134         //LED_MODE_Toggle();
135     }
136 
137     //transmit mode radio interrupts
138     else if(radio_mode == MODE_TX){
139         if(RF0){ //FIFO >= FIFO_THRESHOLD interrupt (packet ready)
140             //LED_MODE_Toggle();
141         }
142         if(RF1){ //TX DONE interrupt
143             //switch to RX mode
144             //LED_MODE_Toggle();
145             radio_mode = MODE_RX;
146             RF_SetMode(MODE_RX);
147             RF_SetFIFOThreshold(3);
148         }
149     }
150     //receiver mode radio interrupts
151     else if(radio_mode == MODE_RX){
152         if(RF0){ //SYNC or ADDRESS match interrupt
153             //LED_MODE_Toggle();
154         }
155         if(RF1){ //FIFO > FIFO_THRESHOLD interrupt (packet received)
156             //get answer
157             LED_MODE_Toggle();
158             uint8_t CMD1 = ReadRFReceiveFIFO();
159             uint8_t CMD0 = ReadRFReceiveFIFO();
160             uint8_t TEMP1 = ReadRFReceiveFIFO();
161             uint8_t TEMP0 = ReadRFReceiveFIFO();
162 
163             //calculate temperature
164             uint16_t tempu = ((uint16_t)TEMP1 << 8) | (uint16_t)TEMP0;
////(T+40)*10 in uint
165             int16_t temp = (int16_t)tempu;
166             TEMP_EXTERN = (float)temp/10 - 40.0; //TEMP in C
167 
168             //calculate distance in mm
169             //float SPEED_OF_SOUND = 331.3*sqrt(1+(TEMP_MAIN+TEMP_EX-
TERN)/2/273.15);  //speed of sound at TEMP
170             float SPEED_OF_SOUND = 331.3*sqrt(1+(TEMP_MAIN)/273.15);  //speed
of sound at TEMP
171             DIST = (SPEED_OF_SOUND/2)*((float)TIME / 1000000)*1000;  //distance
in mm (min 2cm max 4m)
172 
173             //write result in eeprom (distance in mm in 16bit)(every 1 sec)
174             static bool b = 0;
175             if(b){
176                 EEPROM_Write(EEPROM_CURRENT_ADDR >> 8, EEPROM_CURRENT_ADDR,
((uint16_t)DIST) >> 8);         //first byte
177                 EEPROM_Write((EEPROM_CURRENT_ADDR + 1) >> 8, EEPROM_CUR-
RENT_ADDR + 1, (uint16_t)DIST);      //second byte
178                 incrementEEPROMAddress();
179                 EEPROM_Write(0x00, 0x00, EEPROM_CURRENT_ADDR >> 8);
180                 EEPROM_Write(0x00, 0x01, EEPROM_CURRENT_ADDR);
181             }
182             b++;
183 
184             //display results on LCD
185             //first line
186             LCD_SetCursor(0x06);
187             if((int)DIST >= 1000) LCD_PutChar(NumToChar(((int)DIST/1000) %
10));
188             if((int)DIST >= 100) LCD_PutChar(NumToChar(((int)DIST/100) % 10));
189             if((int)DIST >= 10) LCD_PutChar(NumToChar(((int)DIST/10) % 10));
190             LCD_PutChar(NumToChar((int)DIST % 10));
191             LCD_PutChar(' ');
192             LCD_PutChar('m');
193             LCD_PutChar('m');
```

```
194             LCD_PutChar(' ');
195             LCD_PutChar(' ');
196             LCD_PutChar(' ');
197             //second line
198             LCD_SetCursor(0x46);
199             int16_t T = (int16_t)(TEMP_MAIN +0.5);
200             if(T < 0){
201                 LCD_PutChar('-');
202                 T = -T;
203             }
204             if(T >= 100) LCD_PutChar(NumToChar((T/100) % 10));
205             if(T >= 10) LCD_PutChar(NumToChar((T/10) % 10));
206             LCD_PutChar(NumToChar(T % 10));
207             LCD_PutChar(' ');
208             LCD_PutChar('C');
209             LCD_PutChar(' ');
210             LCD_PutChar(' ');
211 
212             //standby mode until next measurement
213             radio_mode = MODE_STANDBY;
214             RF_SetMode(MODE_STANDBY);
215         }
216     }
217 
218 //    else if(MEAS && MODE){
219 //        __delay_ms(15);          //debounce delay
220 //        if(MEAS_GetValue()){     //MEAS button interrupt
221 //            //TriggerPulse();      //send trigger pulse
222 //            //TMR2 = 0;            //timer start
223 //        }
224 //    }
225 }
226 
227 uint16_t readADC(void){
228 
229     ADC1_ChannelSelect(TEMP);
230     ADC1_SoftwareTriggerEnable();
231     __delay_us(1);
232     ADC1_SoftwareTriggerDisable();
233     while(!ADC1_IsConversionComplete(TEMP));
234     uint16_t ADC_result = ADC1_ConversionResultGet(TEMP);
235 
236     return ADC_result;
237 }
238 
239 float readTemperature(void){
240     //reads temperature from thermistor IC
241     int ADC_diff = readADC() - 230; //ADC out is ~230 at ~21C, temperatures are
compared to these values
242     float Vdiff = 3.3*(float)ADC_diff / 1024;
243     float TEMP = 21 + 100*Vdiff; //10mV~1C
244 
245     return TEMP;
246 }
247 
248 float readTemperatureFiltered(uint8_t smp_num){
249     float result = 0;
250     int i;
251     for (i=0; i<smp_num; i++)
252         result+=readTemperature();
253 
254     return result/smp_num;
255 }
256 
257 void TriggerPulse(void){
258     TRIG_SetHigh();
259     __delay_us(10);
260     TRIG_SetLow();
```

```
261 }
262
263 inline void incrementEEPROMAddress(void){
264     if(EEPROM_CURRENT_ADDR == 0b0111111111111111 || EEPROM_CURRENT_ADDR ==
0b0111111111111110) EEPROM_CURRENT_ADDR = 0x0002; //rollover
265     else EEPROM_CURRENT_ADDR+=2;
266 }
267
268 char NumToChar(uint8_t num){
269     return num + '0';
270 }
271
272 void USBtransfer(void){
273
274     if(USBGetDeviceState() < CONFIGURED_STATE) return;
275     if(USBIsDeviceSuspended()== true) return;
276
277     if(USBUSARTIsTxTrfReady() == true){
278         uint8_t i;
279         uint8_t numBytesRead;
280         bool readRequest = false;
281
282         //usb read
283         numBytesRead = getsUSBUSART(readBuffer, sizeof(readBuffer));
284
285         for(i=0; i<numBytesRead; i++){
286             switch(readBuffer[i])
287             {
288                 case USB_EEPROM_READ_REQUEST:
289                     readRequest = true;
290                     break;
291                 default:
292                     break;
293             }
294         }
295
296         //usb write
297         if(numBytesRead > 0 && readRequest){
298             //read and send entire eeprom memory
299             uint16_t i, j;
300             uint8_t addr1;
301             uint8_t addr0;
302             uint8_t *p;
303
304             //send entire eeprom memory
305             for(i=0; i<512; i++){
306                 //read one page
307                 p = EEPROM_PageRead(i);
308
309                 //send page
310                 for(j=0; j<64; j++){
311                     writeBuffer[j] = *(p+j);
312                 }
313                 putUSBUSART(writeBuffer, sizeof(writeBuffer));
314                 CDCTxService();
315             }
316         }
317     }
318 }
319
320 /*END OF FILE*/
```

```c
//Távoli egység

#include "mcc_generated_files/mcc.h"
#include "MRF89XAM8A.h"
#include <math.h>

#define FCY (_XTAL_FREQ / 2)
#include "libpic30.h"

//mode
bool MODE = 0;
#define switchMode() (MODE ^=1)

//7 segment display
int current_display_pos = 2;
#define incrementDisplayPos() (current_display_pos = (current_display_pos + 1) % 3)
uint16_t DISPLAY_VALUE = 123;
#define DIGIT1 ((int)(DISPLAY_VALUE/100) % 10)
#define DIGIT2 ((int)(DISPLAY_VALUE/10) % 10)
#define DIGIT3 (DISPLAY_VALUE % 10)

//measurement data
volatile float DIST = 0;
volatile uint16_t TIME = 0;
volatile float TEMP_MAIN = 0;
volatile float TEMP_EXTERN = 0;

//radio mode
volatile RF_MODE radio_mode;


//functions
void IO_InterruptHandler(void);
void T1_InterruptHandler(void);
void displayClockPulse(void);
void displayDigit(int digit);
void displayNumber(int number); //for testing
uint16_t readADC(void);
float readTemperature(void);
float readTemperatureFiltered(uint8_t smp_num);

int main(void)
{
    // initialize the device
    __delay_ms(10);
    SYSTEM_Initialize();
    RF_Initialize();

    //init radio
    radio_mode = MODE_RX;
    RF_SetMode(radio_mode);
    RF_SetFIFOThreshold(3);
    RF_SetPacketSize(4);

    //init temp
    TEMP_MAIN = readTemperature();
    TEMP_EXTERN = TEMP_MAIN;

    CN_SetInterruptHandler(IO_InterruptHandler);
    TMR1_SetInterruptHandler(T1_InterruptHandler);

    while (1)
    {
        if(radio_mode == MODE_STANDBY){
            int16_t temp = (int16_t)((TEMP_EXTERN + 40) * 10); //(T+40)*10 in int
            uint16_t tempu = (uint16_t)temp;
```

```
67              radio_mode = MODE_TX;
68              RF_SetMode(radio_mode);
69              RF_SetFIFOThreshold(2);
70              WriteRFTransmitFIFO(0x00);
71              WriteRFTransmitFIFO(0x00);
72              WriteRFTransmitFIFO(tempu >> 8);
73              WriteRFTransmitFIFO(tempu);
74          }
75
76          if(DIST < 250) LED_WARN1_SetHigh();
77          else LED_WARN1_SetLow();
78          if(DIST < 500) LED_WARN2_SetHigh();
79          else LED_WARN2_SetLow();
80      }
81
82      return 1;
83  }
84
85
86  void IO_InterruptHandler(void){
87
88      uint16_t porta = PORTA;
89      uint16_t portb = PORTB;
90
91      bool RF0 = portb & 16384; //RB14
92      bool RF1 = portb & 8; //RB3
93      bool SW = porta & 4; //RA2
94
95      //transmit mode radio interrupts
96      if(radio_mode == MODE_TX){
97          if(RF0){ //FIFO >= FIFO_THRESHOLD interrupt (TX START)
98              //LED_MODE_Toggle();
99          }
100         if(RF1){ //TX DONE interrupt
101             //switch back to RX mode
102             radio_mode = MODE_RX;
103             RF_SetMode(MODE_RX);
104             RF_SetFIFOThreshold(3);
105             LED_MODE_Toggle();
106         }
107     }
108     //receiver mode radio interrupts
109     else if(radio_mode == MODE_RX){
110         if(RF0){ //SYNC or ADDRESS match interrupt
111         }
112         if(RF1){ //FIFO > FIFO_THRESHOLD interrupt (packet received)
113             uint8_t TIME1 = ReadRFReceiveFIFO();
114             uint8_t TIME0 = ReadRFReceiveFIFO();
115             uint8_t TEMP1 = ReadRFReceiveFIFO();
116             uint8_t TEMP0 = ReadRFReceiveFIFO();
117
118             //calculate temperature
119             uint16_t tempu = ((uint16_t)TEMP1 << 8) | (uint16_t)TEMP0;
////(T+40)*10 in uint
120             int16_t temp = (int16_t)tempu;
121             TEMP_MAIN = (float)temp/10 - 40.0; //TEMP in C
122             TEMP_EXTERN = readTemperature();
123
124             //calculate and display distance in cm
125             TIME = ((uint16_t)TIME1 << 8) + (uint16_t)TIME0;        //time in
us
126             float SPEED_OF_SOUND = 331.3*sqrt(1+(TEMP_MAIN+TEMP_EX-
TERN)/2/273.15);  //speed of sound at TEMP
127             DIST = (SPEED_OF_SOUND/2)*((float)TIME / 1000000)*100;  //distance
in cm (min 2cm max 4m)
128
129             //display
130             DISPLAY_VALUE = DIST;
```

```
131
132             radio_mode = MODE_STANDBY;
133         }
134     }
135
136 //    else if(SW){ //MODE button pressed
137 //         __delay_ms(15); //debounce delay
138 //         if(SW_MODE_GetValue()){
139 //         }
140 //    }
141 }
142
143 void T1_InterruptHandler(void){
144     //handles the 7 segment display
145     incrementDisplayPos(); //0-2 cycle
146     displayClockPulse();
147     switch (current_display_pos){
148         case 0:
149             displayDigit(DIGIT1);
150             break;
151         case 1:
152             displayDigit(DIGIT2);
153             break;
154         case 2:
155             displayDigit(DIGIT3);
156             break;
157         default:
158             break;
159     }
160 }
161
162 void displayClockPulse(void){
163     //increments decade counter by 1 //switches digits on 7 segment
164     CNTR_CLK_SetHigh();
165     __delay_us(2); //min 0.3 us
166     CNTR_CLK_SetLow();
167 }
168
169 void displayDigit(int digit){
170     //displays 1 digit on the 7 segment display
171     _LATB7 = (digit & ( 1 << 0 )) >> 0; //LED_A
172     _LATB8 = (digit & ( 1 << 1 )) >> 1; //LED_B
173     _LATB9 = (digit & ( 1 << 2 )) >> 2; //LED_C
174     _LATB6 = (digit & ( 1 << 3 )) >> 3; //LED_D
175 }
176
177 void displayNumber(int number){
178     //displays 3 digit number on the 7 segment display
179     //not needed in final program
180     int last_digit = number % 10;
181     int middle_digit = (int)(number/10) % 10;
182     int first_digit = (int)(number/100) % 10;
183
184     displayDigit(first_digit);
185     __delay_ms(6);
186     displayClockPulse();
187     incrementDisplayPos();
188
189     displayDigit(middle_digit);
190     __delay_ms(6);
191     displayClockPulse();
192     incrementDisplayPos();
193
194     displayDigit(last_digit);
195     __delay_ms(6);
196     displayClockPulse();
197     incrementDisplayPos();
198 }
```

```
199
200 uint16_t readADC(void){
201
202     ADC1_ChannelSelect(TEMP);
203     ADC1_SoftwareTriggerEnable();
204     __delay_us(1);
205     ADC1_SoftwareTriggerDisable();
206     while(!ADC1_IsConversionComplete(TEMP));
207     uint16_t ADC_result = ADC1_ConversionResultGet(TEMP);
208
209     return ADC_result;
210 }
211
212 float readTemperature(void){
213     //reads temperature from thermistor IC
214     float Vref_p = 2.6;
215     int ADC_diff = readADC() - 300; //ADC out is 300 at 21C
216     float Vdiff = Vref_p*(float)ADC_diff / 1024;
217     float TEMP = 21 + 100*Vdiff;
218
219     return TEMP;
220 }
221
222 float readTemperatureFiltered(uint8_t smp_num){
223     float result = 0;
224     int i;
225     for (i=0; i<smp_num; i++)
226         result+=readTemperature();
227
228     return result/smp_num;
229 }
```