Hőmérő alkalmazás készítése

Feladatunk egy olyan alkalmazás készítése, amely a TelosB mote-okon található Sensirion SHT11 hőmérséklet és páratartalom érzékelő szenzor segítségével másodpercenként méri a környezet hőmérsékletét. A hőmérséklet mérését a nullás led ki és bekapcsolásával kell jelezni, míg azt, hogy a hőmérséklet elérte a 30 C°-ot a TelosB mote-on található egyes led-el kell jelezni, és azt hogy a hőmérséklet 30 C° alatt van a második led-del kell jelezni. Az alkalmazás továbbfejlesztéseként küldjük el a mért értékeket a PC felé a soros port-on keresztül, és egy java kód segítségével írjuk ki az értékeket a konzolra.

A feladat megoldásához először is hozzunk létre egy teljesen üres mappát, és nevezzük el *Homero*-nek. Ebbe a mappába hozzunk létre egy *HomeroC.nc* file-t. Ez a file fogja tartalmazni az alkalmazásunk konfigurációját. A *HomeroC.nc* file-ba írjuk be az alábbi kódot:

```
configuration HomeroC
{}
implementation
{
     components MainC;
     components LedsC;
     components new TimerMilliC();
     components new SensirionSht11C();
     components HomeroP;

     HomeroP.Boot-> MainC;
     HomeroP.Leds -> LedsC;
     HomeroP.Timer -> TimerMilliC;
     HomeroP.Read -> SensirionSht11C.Temperature;
}
```

A *MainC* komponens biztosítja a *Boot* interface-t, a *LedsC* komponens biztosítja a *Leds* interface-t (*(TOSROOT)/tos/interfaces*), a *HomeroP* komponens, pedig az általunk készített alkalmazást tartalmazza. Emellett szükségünk van még a *TimerMilliC()* komponensre az időzítés megvalósításához, valamint a *SensirionSht11C()* komponensre a TelosB mote-on található hőmérséklet és páratartalom szenzor kiolvasásához. Ez a két komponens úgynevezett generikus komponens, ezért szükséges a komponens megadásánál a *new* parancs. Egy generikus komponensből több is lehet egy programban, és mindegyik egy új különálló komponenst fog jelölni. A *TimerMilliC()* esetében például ez azt fogja jelenteni, hogy több ilyen komponenst is megadhatunk és mindegyik egy különálló időzítőként fog működni. A *TimerMilliC()* komponens biztosítja a *Timer* interface-et ami az időzítők kezeléséhez tartalmaz parancsokat (*(TOSROOT)/tos/lib/timer*). A *SensirionSht11()* segítségével tudjuk kiolvasni a SensirionSht11 szenzorról az aktuális hőmérséklet és páratartalom értéket. Ez a komponens a *(TOSROOT)/tos/platforms/telosa/chips/sht11* mappában található. Ha megnyitjuk a *SensirionSht11C.nc* file-t, akkor láthatjuk, hogy két *Read* interface-t (*(TOSROOT)/tos/interfaces*) biztosít, melyeket *Temperature*-nek és *Humidity*-nek neveztek el, hogy meg lehessen különböztetni ezeket az interface-eket. A *HomeroC* konfigurációban ezért a korábbiakkal ellentétben meg kell adni azt, hogy a komponens melyik interface-ét szeretnénk az alkalmazásunkhoz, azaz a *HomeroP* modulhoz kötni. Ezt a *komponensnév.interfacenév* formában adhatjuk meg.

A következő lépésben adjuk meg a *HomeroP* modult is:

```
module HomeroP
{
     uses interface Boot;
     uses interface Leds;
     uses interface Timer<TMilli>;
     uses interface Read<uint16_t>;
}
implementation
{
     event void Boot.booted()
     {
          call Timer.startPeriodic(1000);
     }

     event void Timer.fired(){
          call Leds.led0Toggle();
          call Read.read();
     }

     event void Read.readDone(error_t result, uint16_t val ){
          int32_t Temp;
          if(result==SUCCESS){
               Temp=-3960+(int32_t)val;
               if(Temp>=3000){
                    call Leds.led1On();
                    call Leds.led2Off();
               }else{
                    call Leds.led1Off();
                    call Leds.led2On();
               }
          }
     }
}
```

Ha a *Boot.booted()* event generálódik, akkor elindítjuk az időzítőt a *Timer.startPeriodic(uint32_t dt)* utasítással, úgy hogy másodpercenként generálódjon a *Timer.fired()* event. A *Read* interface-en keresztül lehet kiolvasni a SensirionSht11 szenzorról a mért hőmérséklet és páratartalom értékeket, jelen esetben a hőmérséklet értéket, hisz a konfigurációban ezt kapcsoltuk a *Read* interface-hez. Ha megnyitjuk a *Read.nc* file-t akkor láthatjuk, hogy tartalmaz egy *read()* parancsot, illetve egy *readDone(error_t result, uint16_t val)* event-et. A *read()* parancs segítségével tudjunk elindítani az adatok kiolvasását a szenzorról. Az adatok kiolvasásának végét a *readDone()* event jelzi. Ebben az event-ben a *result* a kiolvasás eredményét, míg *val* a kiolvasott értéket tartalmazza. Itt megvizsgáljuk, hogy helyesek-e a kiolvasott adatok, majd a szenzor specifikációjában megadott konverziós képlet alapján átszámoljuk a kapott értékeket Celsius fokra, és ennek megfelelően kapcsolgatjuk a led-eket. A konverziós képlet a következő:

$$T = d_1 + d_2 \cdot SO_T$$

VDD	d_1 (°C)	d_1 (°F)
5V	-40.1	-40.2
4V	-39.8	-39.6
3.5V	-39.7	-39.5
3V	-39.6	-39.3
2.5V	-39.4	-38.9

SO_T	d_2 (°C)	d_2 (°F)
14bit	0.01	0.018
12bit	0.04	0.072

Hogy ne kelljen, tört számokkal dolgozni az értékeket 100-al szorozzuk meg. Harmadik lépésként pedig hozzuk létre a *Makefile* file-t, mely ez előbbiekben elkészített alkalmazás fordításához szükséges. A *Makefile*-nak a következő két sort kell tartalmazza:

```
COMPONENT=HomeroC
include $(MAKERULES)
```

Az első sor megadja konfigurációs file nevét, amely az alkalmazás komponenst tartalmazza. A második sor, pedig meghívja a TinyOS build rendszerét, mely segítésével lefordítható az alkalmazásunk. Végül fordítsuk le az alkalmazásunkat, majd töltsük fel a TelosB mote-ra. Az alkalmazás fordítása és felprogramozása az alábbi módon történik:

```
make telosb install
```

Fejlesszük tovább az alkalmazásunkat úgy, hogy a mért értékeket a soros port-on keresztül, küldje el. Ehhez először is hozunk létre egy a rádiós adatküldésnél megismert módon és okból egy *Homero.h* header filet:

```
#ifndef HOMERO_H
#define HOMERO_H

enum{
     AM_HOMEROMSG = 6,
};

typedef nx_struct HomeroMsg{
     nx_uint16_t val;
}HomeroMsg;

#endif
```

Ezután alakítsuk át a *HomeroC* konfigurációt, kiegészítve két új komponenssel, amelyek a soros port bekapcsolásáért, és az azon keresztüli adat továbbításért felelősek.

```
....
components SerialActiveMessageC;
components new SerialAMSenderC(AM_HOMEROMSG);
....
HomeroP.SplitControl-> SerialActiveMessageC;
HomeroP.AMSend->SerialAMSenderC;
....
```

Ezek a komponensek a *SerialActiveMessageC* és az *SerialAMSenderC()*. Az előbbi a soros port bekapcsolásához használatos *SplitControl* interfac-et biztosítja, míg az utóbbi a soros port-on az üzenet elküldéséhez használható *AMSend* interface-et biztosítja.
A *SplitControl* interface a *(TOSROOT)/tos/interfaces* mappában található. Ha megnyitjuk a *SplitControl.nc* file-t akkor találunk két parancsot a *start()* és a *stop()* parancsot, illetve két event-et, a *startDone(error_t error)* és a *stopDone(error_t error)* event-et. A parancsok segítségével lehet elindítani, illetve leállítani a soros port-ot, míg az event-ek jelzik a soros port elindulását, illetve leállását.
Egészítsük ki az új HomeroP modult is:

```
....
uses interface SplitControl;
uses interface AMSend;
....
     event void Boot.booted(){
          call SplitControl.start();
     }

     event void SplitControl.startDone(error_t err){
          if(err==SUCCESS){
               call Timer.startPeriodic(1000);
          }else{
               call SplitControl.start();
          }
     }

     event void SplitControl.stopDone(error_t err){}

     event void Timer.fired(){
          call Leds.led0Toggle();
          call Read.read();
     }

     bool busy = FALSE;
     message_t pkt;

     event void Read.readDone(error_t result, uint16_t val ){
          int32_t Temp;
          if(result==SUCCESS){
               Temp=-3960+(int32_t)val;
               if(Temp>=3000){
                    call Leds.led1On();
                    call Leds.led2Off();
```

```
                }else{
                     call Leds.led1Off();
                     call Leds.led2On();
                }
                if(!busy){
                     HomeroMsg*   btrpkt   =   (HomeroMsg*)(call
AMSend.getPayload(&pkt, sizeof(HomeroMsg)));
                     btrpkt->val = val;
                     if(call      AMSend.send(AM_BROADCAST_ADDR,
&pkt, sizeof(HomeroMsg))==SUCCESS){
                          busy = TRUE;
                     }
                }
          }
     }

     event void AMSend.sendDone(message_t* msg, error_t error)
{
          busy = FALSE;
     }
}
```

Itt lényégében kiegészítettük az alkalmazásunkat a soros port bekapcsolásával, és az üzenet elküldésével. A folyamat teljes mértékben megegyezik a korábbiakban a rádiós adatküldésnél tanultakkal. Készítsük el a számítógépen a soros port-ról beérkező adatok fogadását és az értékek átkonvertálását, majd azokat a konzolon megjelenítő java kódot.

```
import static java.lang.System.out;
import net.tinyos.packet.*;
import net.tinyos.message.*;
import net.tinyos.util.PrintStreamMessenger;
import java.io.*;

class Homero implements MessageListener{

     private MoteIF moteIF;

     public Homero(MoteIF moteIF){
          this.moteIF=moteIF;
          this.moteIF.registerListener(new HomeroMsg(),this);
     }

     public void messageReceived(int dest_addr,Message msg){
          if (msg instanceof HomeroMsg) {
               HomeroMsg homeroData = (HomeroMsg)msg;
               out.println("A    homersekelt:    "    +    (-
39.4+(0.01*(double)homeroData.get_val())));
          }
     }

     public static void main(String[] args) throws Exception
```

```
    {
        PhoenixSource phoenix = null;
        MoteIF mif = null;

        if( args.length == 0 ){
            phoenix                                          =
BuildSource.makePhoenix(PrintStreamMessenger.err);
        } else if( args.length == 2 && args[0].equals("-
comm") ) {
            phoenix    =    BuildSource.makePhoenix(args[1],
PrintStreamMessenger.err);
        } else {
            System.err.println("usage:  java  Homero  [-comm
<source>]");
            System.exit(1);
        }
        mif = new MoteIF(phoenix);
        Homero app= new Homero(mif);
    }
}
```

A net.tinyos.message tartalmaz egy *MoteIF* osztályt. Ez teszi lehetővé, hogy könnyen tudjunk fogadni és küldeni *mig* (Message Interface Generátor) által generált csomagokat. Ezt a következőekben fogjuk megnézni. Mielőtt használnák ezt a *MoteIF*-et szükség van arra, hogy a *net.tinyos.packet.BuildSource.makePhoenix* parancs segítségével megnyissunk egy csomagforrást. A csomagforrás megadásával adhatjuk meg azt, hogy a java kód hogyan éri el a mote-ot. Például *–comm serial@/dev/ttyUSB0:telosb*.
A fenti kódban megvizsgáljuk a parancssori paramétereket, és megnyitunk a megadott, vagy ennek hiányában az alapértelmezett (sf@localhost:9001) csomagforrást.
A csomagok fogadásához pedig a konstruktorban szükségünk van egy *registerListener()* függvényre, mely regisztrálja az osztályunkat, a *HomeroMsg* fogadására, majd amikor egy ilyen csomag érkezik, akkor a *messsageReceived()* függvény mindig meghívódik. Itt van lehetőségünk kiolvasni a csomag tartalmát, átkonvertálni az és kiíratni a konzolra. Utolsó lépésben készítsük el a Makefile-t is:

```
COMPONENT=HomeroC
BUILD_EXTRA_DEPS = HomeroMsg.java Homero.class
CLEAN_EXTRA = *.class HomeroMsg.java

HomeroMsg.java: Homero.h
    mig java -target=null -java-classname=HomeroMsg Homero.h
HomeroMsg -o $@

HomeroMsg.class: HomeroMsg.java
    javac HomeroMsg.java

Homero.class: Homero.java
    javac Homero.java

include $(MAKERULES)
```

A *BUILD_EXTRA_DEPS* sor azt mondja meg, hogy az alkalmazásunknak vannak külső függőségei, ez pedig a *HomeroMsg.java* és a *Homero.class*. A *CLEAN_EXTRA* sor azt mondja meg, hogy milyen plusz file-okat kell törölni a *make clean* utasítás meghívása után. A következő sor pedig lényegében meghívja a *mig*-et, ami létrehozza a *Homero.h* header file alapján azt a *HomeroMsg.java*-t ami tartalmazza azt az osztályt amit a *MoteIF* osztály vár. Ezután ezt és a *Homero.java* alkalmazást is lefordítjuk. A *mig* paraméterezése a következőképpen értelmezhető:

mig	meghívja a míg-et
java	java osztályt fog készíteni
-target=null	a null platform számára
-java-classname=HomeroMsg	elnevezi az osztályt *HomeroMsg*-nek
Homero.h	a *Homero.h* felhasználásával
HomeroMsg	a *HomeroMsg* struktúra alapján
-o $@	írja bele ezeket a *HomeroMsg.java*-ba

Mivel a *mig* tool segítségével hozzuk létre a szükséges java file-t a header file-ban lévő struktúra alapján, ez megköveteli, hogy a struktúra neve és az üzenet am típusának neve megegyezzen. Ezért kell az am típust mindenképpen *AM_HOMEROMSG*-nek elnevezni.