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, majd a TelosB mote-on található rádió segítségével továbbítja az adatokat a számítógéphez csatlakoztatott basestation mote felé. A basestation mote a beérkező adatok a soros port-on keresztül továbbítja a számítógépnek, ahol az adatokat a net.tinyos.tools.*Listen.java* alkalmazás segítségével jelenítjük meg.

A feladat elkészítésének lépései:

1. lépés: 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 konfiguráció tartalmazza azokat a komponenseket, illetve az összekapcsolásukat, amelyek az alkalmazásunkhoz szükségesek. A *HomeroC.nc* file-be írjuk be az alábbi kódot:

```
configuration HomeroC
{}
implementation
{
	components MainC;
	components LedsC;
	components HomeroP;

	HomeroP.Boot-> MainC;
	HomeroP.Leds -> LedsC;
}
```

A *MainC* komponens biztosítja a *Boot* interface-t, a *LedsC* komponens biztosítja a *Leds* interface-t, a *HomeroP* komponens, pedig az általunk készített alkalmazást tartalmazza. Ez a konfigurációnak nincsenek külső interface kapcsolatai, mivel önálló egység, ezért a felső blokk üres (nincs benne provides vagy uses interface).

2. lépés: A *HomeroP* komonenst már használtuk, és látjuk hogy hogyan kötöttük össze más komponensekkel, de még nem írtuk meg. Hozzuk létre a *HomeroP.nc* file-t, mely majd az alkalmazásunkat fogja tartalmazni. A file nevében található P jelzi, hogy ez a modul egy úgynevezett private modul, tehát ezt a modul-t ne használja más alkalmazás. A *HomeroP.nc* file-ba írjuk be az alábbi kódot:

```
module HomeroP
{
	uses interface Boot;
	uses interface Leds;
}
implementation
{
	event void Boot.booted()
	{
		call Leds.led0On();
	}
}
```

Látható, hogy a *HomeroP* modul használja, a *Boot*, illetve a *Leds* interface-eket, melyeket a *HomeroC.nc* file-ban már össze is kötöttünk az őket biztosító *MainC*, illetve *LedsC* komponensekkel. A felhasznált két interface a *(TOSROOT)/tos/interfaces* mappában található. Ha ebben a mappában a *Boot.nc* file-t megnyitjuk, akkor láthatjuk, hogy a *booted()* even-et tartalmazza, mely azt jelzi, hogy a rendszer megfelelően elindult. Ha ez bekövetkezett, akkor a *Leds* interface-ben található *led0On()* parancs segítségével bekapcsoljuk a TelsoB mote-on található 0-ás (piros) led-et.

3. lépés: Hozzunk létre egy *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.

4. lépés: Fordítsuk le az alkalmazásunkat, majd töltsük fel a TelosB mote-ra. Az alkalmazás fordítása és felprogramozása az alábbi módon történik:

```
make telosb install
```

A felprogramozás után a mote-on fel fog kapcsolódni a 0-ás, azaz a piros színű led.

5. lépés: Mivel a feladatunk az, hogy másodpercenként vegyünk mintát a mote-on található hőmérséklet és páratartalom szenzorról, ezért szükségünk van az alkalmazásunkban egy *TimerMilliC()* komponensre is. Egészítsük ki a *HomeroC* konfigurációnkat ezzel a komponenssel és kössük hozzá az alkalmazásunkat megvalósító *HomeroP* modulhoz.

```
components new TimerMilliC();

HomeroP.Timer -> TimerMilliC;
```

A *TimerMilliC* komponens egy generikus komponens, ezért szükséges a komponens megadásánál a *new* parancs. Ez azt jelenti, hogy a különböző helyeken megadott new *TimerMilliC* komponens különböző lesz, azaz külön lehet állítani benne a periódust. Ezzel szemben a nem generikus komponensekből, mint például a *LedsC* komponensből, mindig csak egy van, függetlenül attól, hogy hány másik komponensben szerepel. A *TimerMilliC* komponens biztosítja a *Timer* interface-t.

6. lépés: A *HomeroP* modulunkat is ki kell egészíteni az alábbiaknak megfelelően.

```
uses interface Timer<TMilli>;

event void Boot.booted()
{
      call Leds.led0On();
      call Timer.startPeriodic(1000);
}

event void Timer.fired()
{
      call Leds.led1Toggle();
}
```

Természetesen az első sort (*uses interface …*) a modul első blokkjába kell beleírni, ahol fel vannak sorolva a modul által használt interface-ek. A többi részt a modul implementációjába írjuk.

A *HomeroP* modul használja a *Timer* interface-t, mely az előbbi interface-ekkel ellentétben a *(TOSROOT)/tos/lib/timer* mappában található. Ha megnyitjuk a *Timer.nc* file-t akkor láthatjuk, hogy számos különböző parancsot tartalmaz a timer kezeléséhez, ezek közül nekünk most a *startPeriodic(uint32_t dt)* parancsra van szükségünk, mely segítségével egy periodikus timer-t tudunk elindítani. Mivel másodpercenként szeretnénk mérést végezni, ezért a milliszekundumban megadható periódusidőt 1000-re állítjuk. A *Timer.nc* file tartalmaz még egy *fired()* event-et, annak jelzésére, hogy a beállított időzítés lejárt. Ebben az event-ben a *Leds* interface által biztosított *led1Toggle()* parancs segítségével villogtatjuk az 1-es (zöld) led-et, hogy megfigyelhessük a timer működését.

7.lépés: A 4. lépésnek megfelelően programozzuk fel a TelosB mote-ot, és vizsgáljuk meg, hogy helyesen működik-e az alkalmazásunk. Azt kell tapasztalnunk, hogy a 0-as LED folyamatosan ég, míg az 1-es LED másodpercenként villog.

8. lépés: Egészítsük ki az alkalmazásunkat, azzal a komponenssel, amely segítségével ki tudjuk olvasni a hőmérséklet és páratartalom szenzorról a hőmérséklet értékeket. Ehhez a következő képen kell kiegészíteni *HomeroC* konfigurációnkat:

```
components new SensirionSht11C();

HomeroP.Read -> SensirionSht11C.Temperature;
```

A SensirionSht11 szenzorról a *SensirionSht11C* generikus komponens segítségével tudjuk kiolvasni a hőmérséklet értékeket. 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 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.

9. lépés: A *HomeroP* modult is ki kell egészíteni az alábbi formában:

```
uses interface Read<uint16_t>;
```

```
event void Timer.fired()
{
	call Leds.led1Toggle();
	call Read.read();
}

event void Read.readDone(error_t result, uint16_t val)
{
	call Leds.led2Toggle();
}
```

A *HomeroP* modul használja a *Read* interface-et, amely a *(TOSROOT)/tos/interfaces* mappában található. 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 tudunk elindítani az adatok kiolvasását a szenzorról. Mivel másodpercenként szeretnénk ezt megtenni, így ezt a parancsot a *fired()* event-ben kell meghívni. Az adatok kiolvasásának végét a *readDone()* event jelzi. Ebben az event-ben a *result* a kiolvasás eredményét, míg *val* a kiolvasott értéket tartalmazza. Annak érdekében, hogy láthassuk, az egyes kiolvasások megtörténtét, a *led2Toggle()* parancs segítségével minden kiolvasáskor villogtassuk a 2-es (kék) LED-et.

10. lépés: Programozzuk fel az alkalmazásunkat, és vizsgáljuk meg a működését. Azt kell tapasztalunk, hogy az 2-es LED egy kis késéssel villog az 1-es LED-hez képest, mert egy kis idő telik el a *read()* parancs és a *readDone()* event között.

11. lépés: A mért értékeket a rádió segítségével szeretnénk elküldeni a basestation mote felé. Ennek első lépéseként be kell kapcsolni a rádiót. A rádió bekapcsolásához az *ActiveMessageC* komponenssel kell kiegészíteni a *HomeroC* konfigurációnkat, mely a *SplitControl* interface-et biztosítja.

```
components ActiveMessageC;

HomeroP.SplitControl-> ActiveMessageC;
```

12. lépés: A *SplitControl* interface-et felhasználva egészítsük ki úgy a *HomeroP* modult, hogy a mote elindulása után kapcsolja be a rádiót:

```
uses interface SplitControl;

event void Boot.booted()
{
      call SplitControl.start();
}

event void SplitControl.startDone(error_t error)
{
      call Leds.led0On();
      call Timer.startPeriodic(1000);
}

event void SplitControl.stopDone(error_t error){}
}
```

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 rádiót, míg az event-ek jelzik a rádió elindulását, illetve leállását. Ebben az alkalmazásban csak elindítani szeretnénk a rádiót, megállítani nem, viszont mivel az adott interface összes even-tjét meg kell valósítani, ezért kell megadni a *HomeroP* modulban a *stopDone()* event-et is. A rádió elindítását a *booted()* event-ben végezzük, és ha elindult a rádió, azaz *startDone()* event generálódott, csak akkor indítjuk csak el a timer-t.

13. lépés: Programozzuk fel az alkalmazásunkat és nézzük meg a működését. Ugyan úgy kell működnie, mint előbb, csak most már a rádió is be van kapcsolva.

14. lépés: A rádió elindítása után még szükségünk van egy komponensre, amely segítségével lehetőség van a rádión keresztül történő adatküldésre. Ez a komponens az *AMSenderC* komponens. Egészítsük ki ezzel is a *HomeroC* konfigurációnkat.

```
components new AMSenderC(0x11);

HomeroP.AMSend->AMSenderC;
```

Az *AMSenderC* komponens is egy generikus komponens, paraméterként az elküldendő csomag *AM_TYPE* értékét várja. Ebben az alkalmazásban ez most *0x11*. Ez az *AMSenderC* komponens biztosítja az *AMSend* interface-t.

15. lépés: Az AMSend interface-t felhasználva egészítsük ki úgy a *HomeroP* modult, hogy a mért hőmérséklet értéket elküldje a rádión keresztül.

```
uses interface AMSend;
```

```
message_t mesResults;

event void Read.readDone(error_t result, uint16_t val)
{
      uint16_t *payload;
      payload=call AMSend.getPayload(&mesResults,2);
      *payload=val;
      call AMSend.send(AM_BROADCAST_ADDR,&mesResults,2);
}

event void AMSend.sendDone(message_t* msg, error_t error)
{
      call Leds.led2Toggle();
}
```

Az *AMSend* interface a *(TOSROOT)/tos/interfaces* mappában található. Ha megnyitjuk az *AMSend.nc* file-t, akkor megtaláljuk a *getPayload(message_t* msg, uint8_t len)* parancsot, amivel a kommunikációhoz használt *message_t* struktúra adat területét tudjuk elérni. Az üzenet elküldése a *send(am_addr_t addr, message_t* msg, uint8_t len)* parancs használható. Az üzenet elküldését, pedig a *sendDone(message_t* msg, error_t error)* even-t jelzi. Az alkalmazásunkban a másodpercenként mért hőmérséklet adatokat szeretnénk elküldeni, így első lépésben deklarálunk egy *mesResults* nevű *message_t* típusú változót, majd a *readDone* event-ben a getPayload parancs segítségével kapunk egy mutatót ennek a struktúrának az adatterületére (*payload*). Ennek a mutatónak a segítségével a *message_t* struktúra adatterületére betöltjük az aktuális hőmérséklet értéket. Az így feltöltött *message_t* struktúrát, a *send()* parancs segítségével küldjük el. Ebben a parancsban az *AM_BROADCAST_ADDR* változó azt jelenti, hogy broadcast-olva, azaz mindenkinek küldje el a csomagot. Végül, pedig a *sendDone()* event jelzi, az üzenet elküldését, és ezt a *led2Toggle()* parancs segítségével a 2-es led (kék) villogatásával jelezzük a TelosB mote-on.

16. lépés: Az elkészült alkalmazást programozzuk fel a TelosB mote-ra.

17.lépés: Egy másik TelosB mote-ra pedig programozzuk fel a *(TOSROOT)/apps /BaseStation* mappában található basestation alkalmazást.

18. lépés: A basestation mote-ot hagyjuk a számítógéphez csatlakoztatva, és gépeljük be a következő parancsot:

```
java net.tinyos.tools.Listen –comm serial@/dev/ttyUSB0:telosb
```

Ennek az alkalmazásnak a segítségével lehetőség van a soros port-on érkező csomagok megjelenítésére. Ha helyesen működik az elkészült alkalmazás, akkor a képernyőn az alábbihoz hasonló soroknak kell megjelenjen.

```
00 FF FF 00 01 02 00 11 A9 1A
00 FF FF 00 01 02 00 11 B0 1A
00 FF FF 00 01 02 00 11 B9 1A
00 FF FF 00 01 02 00 11 C1 1A
00 FF FF 00 01 02 00 11 C7 1A
00 FF FF 00 01 02 00 11 C9 1A
00 FF FF 00 01 02 00 11 C7 1A
00 FF FF 00 01 02 00 11 C2 1A
00 FF FF 00 01 02 00 11 C1 1A
00 FF FF 00 01 02 00 11 BE 1A
00 FF FF 00 01 02 00 11 BE 1A
```

Az fent látható sorok mindegyike, egy a rádión elküldött és a soros port-on beolvasott csomag tartalmát mutatják. A byte-ok jelentése a következő:

- Az első mező (00) jelzi, hogy a megérkezett csomag egy AM csomag.
- A második két mező (FF FF) a cél címét jelöli, amely most azért FF FF mert broadcast-olva küldtük el az üzentet.
- A harmadik két mező (00 01) a forrás címét jelöli.
- A negyedik mező (02) a csomagban található adatmező hossza.
- Az ötödik mező (00) a group-ot jelöli.
- A hatodik mező (06) az elküldött csomag AM_TYPE-ját jelöli.
- Az utolsó két mező pedig a mért hőmérséklet érték.