A Blink alkalmazás és átalakítása

A *Blink* alkalmazás */opt/tinyos/tinyos-2.1.0/apps* mappában található (ennek egy kicsit módosított változatát készítettük el órán, jelen jegyzetben arról lesz szó). Annak érdekében, hogy le tudjuk fordítani majd, fel tudjuk tölteni ezt az alkalmazást a TelosB mote-ra be kell állítanunk az ehhez szükséges környezeti változókat. Ezt a . */opt/tinyos-2.1.0/tinyos.sh* lefuttatásával érhetjük el. Az alkalmazás lefordítása a *make telosb* utasítás segítségével történik. Az alkalmazásunk felprogramozása a *make telosb reinstall* utasítással történik. Ha együtt szeretnénk elvégezni az alkalmazás lefordítását és a felprogramozást is, akkor a *make telosb install* utasítást használjuk. Láthatjuk, hogy a TelosB mote-on elkezdtek villogni a led-ek egy 3 bit-es bináris számlálónak megfelelően.

A *Blink* mint minden más *TinyOS* kód a *nesC* nyelven íródott. A *nesC* nyelv lényegében megegyezik, a C nyelvel, azonban ahhoz képest tartalmaz néhány kiegészítést is, a komponensek és a konkurencia kezelésére. Egy *nesC* alkalmazás egy vagy több komponensből összekötéséből épül fel. A komponensek egyrészt tartalmazzák az úgynevezett interface-eik neveit, illetve magát a komponens implementációját. Egy komponens használhat, illetve biztosíthat is interface-eket. Az interface-ek pedig lényegében parancsok és események gyűjteményei, amiket egy komponens használhat, vagy biztosíthat.

A *nesC* nyelvben két komponens típust különbözetünk meg van, a konfiguráció és a modul. A modul egy vagy több interface megvalósítását, vagy egy alkalmazás kódját tartalmazza. A konfiguráció, pedig az egyes modulok össze huzalozását tartalmazzák.

A *Blink* alkalmazás két komponensből épül fel. Egy modulból (*BlinkP.nc*) és egy konfigurációból (*BlinkC.nc*). Az egyes komponensek elnevezésének is megvan a maga konvenciója. A *Példa.nc* az interface-t jelöli, a *PéldaC.nc* a konfigurációt jelöli, míg a *PéldaP.nc* pedig a modult.

Ha megnyitjuk a *BlinkC.nc* file-t, akkor az alábbi kódot láthatjuk

```
configuration BlinkC {
}
implementation {
  components MainC, BlinkP, LedsC;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;
  components new TimerMilliC() as Timer3;

  BlinkP -> MainC.Boot;
  BlinkP.Timer1 -> Timer1;
  BlinkP.Timer2 -> Timer2;
  BlinkP.Timer3 -> Timer3;
  BlinkP.Leds -> LedsC;
}
```

Itt látható, hogy milyen komponensek kellenek a *Blink* alkalmazás létrehozásához. Kell először is egy MainC komponens, ami a *Boot* interface-et biztosítja. Ez az interface egyetlen eseményt az *event void booted()*-ot tartalmazza (*/tos/interfaces/*).

Ez az esemény a programunk belépő pontja, mert ez jelzi, hogy a mote elindult, és működőképes. Ezután láthatunk három *TimerMilliC()* komponenst, melyek lényegében a mikrokontrollerekben lévő időzítők kezeléséhez szükséges parancsokat és eseményeket tartalmazó interface-eket biztosítják (*Timer<precision_tag>*, */tos/lib/timer*). Az *as* kulcsszó csak annyit jelent, hogy az adott komponenst vagy akár interface-t is egy általunk

tetszőlegesen megválasztott névvel illetünk. A *TimerMilliC()* komponensek esetében azért van szükség az átnevezésre, hogy meg tudjuk különböztetni a három azonos komponenst. A *LedsC* pedig a led-ek kezeléséhez szükséges parancsokat tartalmazó interface-t biztosítja (*Leds*, */tos/interfaces*). A konfigurációs file legutolsó részében a komponensek össze huzalozása látható.
A másik file, amit meg kell vizsgáljunk az a *BlinkP.nc*. Ennek tartalma a következő:

```
module BlinkP
{
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Timer<TMilli> as Timer3;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
  event void Boot.booted()
  {
    call Timer1.startPeriodic( 250 );
    call Timer2.startPeriodic( 500 );
    call Timer3.startPeriodic( 1000 );
  }

  event void Timer1.fired()
  {
    call Leds.led0Toggle();
  }

  event void Timer2.fired()
  {
    call Leds.led1Toggle();
  }

  event void Timer3.fired()
  {
    call Leds.led2Toggle();
  }
}
```

Itt látható a *BlinkP* modul felépítése. Láthatóak azok az interface-ek amiket az előbb a *BlinkC*-ben bekötöttünk a megfelelő komponensekhez. Mint korábban említettem az *event void booted()* esemény, felelős a mote elindulásának jelzésért, így ha ez az esemény generálódik, akkor elindítunk három időzítőt, a *command void Timerx.startPeriodic(uint32_t time)* parancs segítségével. A *startPeriodic(uint32_t time)* parancs azt jelenti, hogy periodikusan szeretnénk működtetni az időzítőket. Ha az egyes időzítők időzítése lejárt, akkor egy *event void Timerx.fired()* esemény generálódik. Ezen esemény bekövetkezésekor meghívjuk a Led-ek villogtatásáért felelős *command void Leds.ledxToggle()* parancsot.

Alakítsuk át ezt a *Blink* alkalmazást úgy, hogy egy futófényt kapjunk. Ehhez elegendő lesz egy *Timer* komponens is. A másik kettőt vegyük ki a konfigurációs és az alkalmazás file-ból, és alakítsuk át a kódot az alábbiaknak megfelelően:

```
configuration BlinkC {
}
implementation {
  components MainC, BlinkP, LedsC;
  components new TimerMilliC() as Timer1;

  BlinkP -> MainC.Boot;
  BlinkP.Timer1 -> Timer1;
  BlinkP.Leds -> LedsC;
}

module BlinkP
{
  uses interface Timer<TMilli> as Timer1;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
  event void Boot.booted()
  {
    call Timer1.startPeriodic( 500 );
  }

  event void Timer1.fired()
  {
    if(call Leds.get()==0){
         call Leds.led0On();
         call Leds.led1Off();
         call Leds.led2Off();
    }else if(call Leds.get()==1){
         call Leds.led0Off();
         call Leds.led1On();
         call Leds.led2Off();
    }else if(call Leds.get()==2){
         call Leds.led0Off();
         call Leds.led1Off();
         call Leds.led2On();
    }else if(call Leds.get()==4){
         call Leds.led0On();
         call Leds.led1Off();
         call Leds.led2Off();
    }
  }
}
```

Itt lényegében a *command uint8_t Leds.get()* parancs segítségével kiolvassuk a led-ek aktuális állását, majd ennek megfelelően változtatjuk azt, hogy mely led-ek világítsanak. A get parancs, ahogy az látható is egy 8 bites értéket ad vissza, melynek alsó három bitje tartalmazza a led-eken aktuálisan megjelenített értéket.

Alakítsuk át úgy a kódot, hogy egy számlálót használjon a led-eken megjelenítendő bináris számláló megvalósításához:

```
configuration BlinkC {
}
implementation {
  components MainC, BlinkP, LedsC;
  components new TimerMilliC() as Timer1;

  BlinkP -> MainC.Boot;
  BlinkP.Timer1 -> Timer1;
  BlinkP.Leds -> LedsC;
}

module BlinkP
{
  uses interface Timer<TMilli> as Timer1;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
  uint8_t counter=0;
  event void Boot.booted()
  {
    call Timer1.startPeriodic( 500 );
  }

  event void Timer1.fired()
  {
    call Leds.set(counter++);
}
```

Ennél a feladatnál is elegendő egy időzítő. Amikor az időzítő lejár és generálódik az *event void Timer1.fired()* esemény, akkor megnöveljük a számláló értékét, és a *command void Leds.set(uint8_t val)* parancs segítségével megjelenítjük annak alsó három bitjét a led-eken.

A nesC nyelvben egy változó deklarálása sokkal pontosabb kell, hogy legyen, mint a standard C nyelvben. Ez azért szükséges, mert például egy *int* mást jelenthet egy TelosB mote-on vagy egy IntelMote2 mote-on. Az előbbi esetében egy 16 bit-es változót, míg az utóbbi esetben egy 32 bites változót kapunk. Ezért egy változót úgy definiálhatunk, hogy megadjuk a típusát, pl *int* vagy *uint*, ezután pedig a bitszélességét, tehát például 8,16,32. Nézzünk néhány példát: *int8_t*, *int16_t*, *int32_t*, *int64_t*, *uint8_t*, *uint16_t*, *uint32_t*, *uint64_t*. Ezen típusok mellett van még egy *bool* típus is a logikai műveletvégzésez. Vannak olyan platformok, amelyek támogatják a float vagy double típusokat, de az ezekkel való műveletvégzés szoftverben van megoldva, nem pedig hardverben (nincs hardveres lebegőpontos aritmetika).