A TinyOS operációs rendszer

[Beágyazott Információs Rendszerek Tervezése]

- Bevezető -

www.tinyos.net

A TinyOS

- vezeték nélküli szenzorhálózatokhoz kifejlesztett
- nyílt kódú (open source)
- ingyenes
- komponens alapú
- eseményvezérelt (event triggered)
- beágyazott operációs rendszer
- kapcsolódó programnyelv: NesC

A NesC programnyelv

- A C nyelv kiterjesztése
- A TinyOS koncepciójához és működéséhez igazított
- Szenzorhálózatokhoz jól illeszkedik
- Fordítás idejű konkurencia ellenőrzés

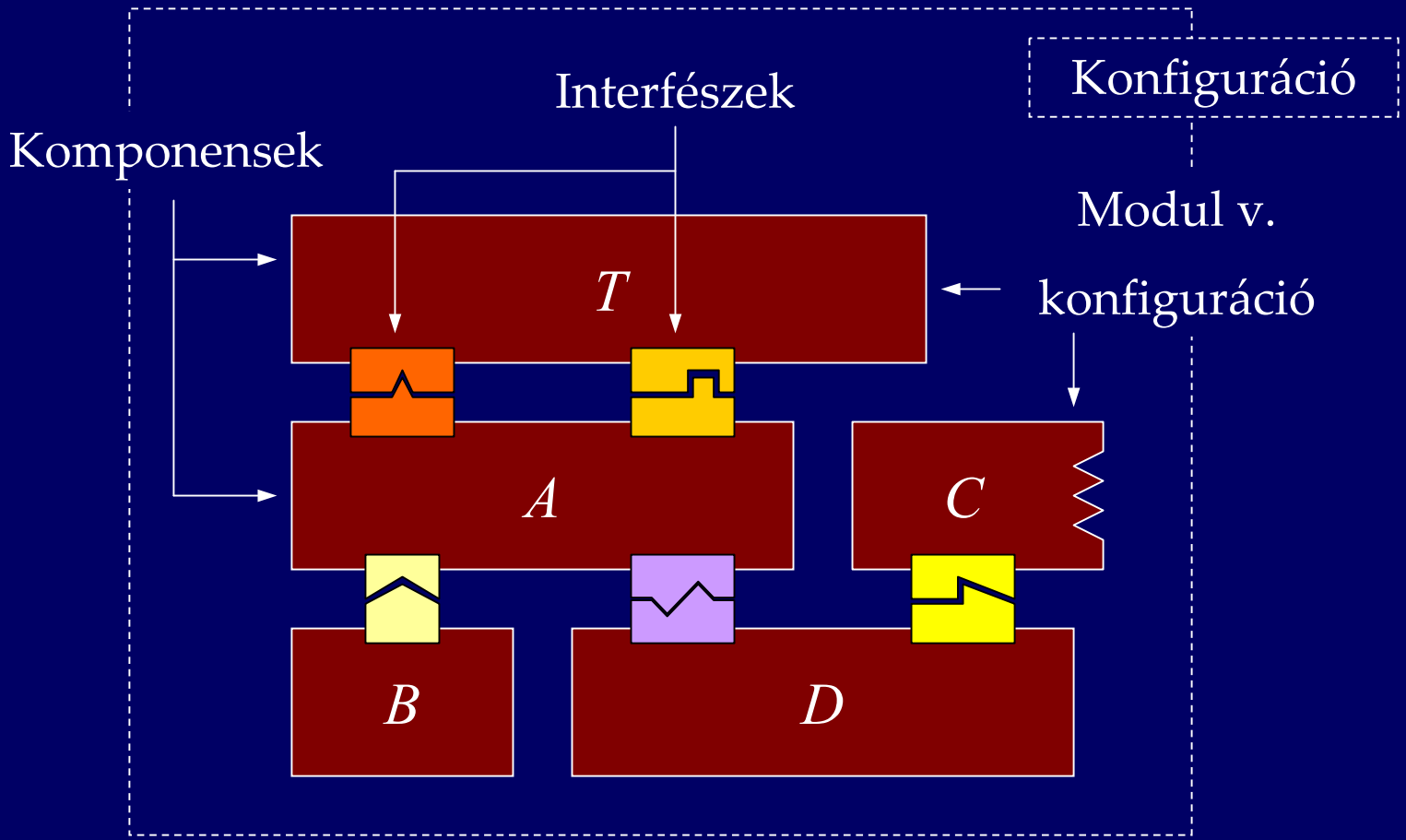
NesC alapelemek

- Komponensek (components): A NesC programok alap építőkövei
  - A komponensek interfészeken keresztül való összehuzalozásával áll össze a rendszer
  - Belső konkurencia: task
  - 2 fajta komponens
    - Modul
    - Konfiguráció
  - Statikus kapcsolat
    - Teljes program-ellenőrzés fordítási időben
    - Jobb kód generálás és analízis

NesC alapelemek

- A komponens szolgáltatást nyújt (*provides*) és használ (*uses*)
- Interfész
  - A modulok implementálják őket
  - A komponensek összehuzalozása az interfészeken keresztül történik
  - Két irányú, a *provider*t és a *user*t kapcsolja össze
  - A *provider* parancsokat (commands) specifikál (‘`provides`’)
  - A *user* eseményeket (events) specifikál (‘`uses`’)

NesC alapelemek



NesC interfész

```
interface <azonosító> { deklarációs-lista }
```

```
interface Timer
{
    command result_t start(char type, uint32_t interval);
    command result_t stop();
    event   result_t fired();
}
```

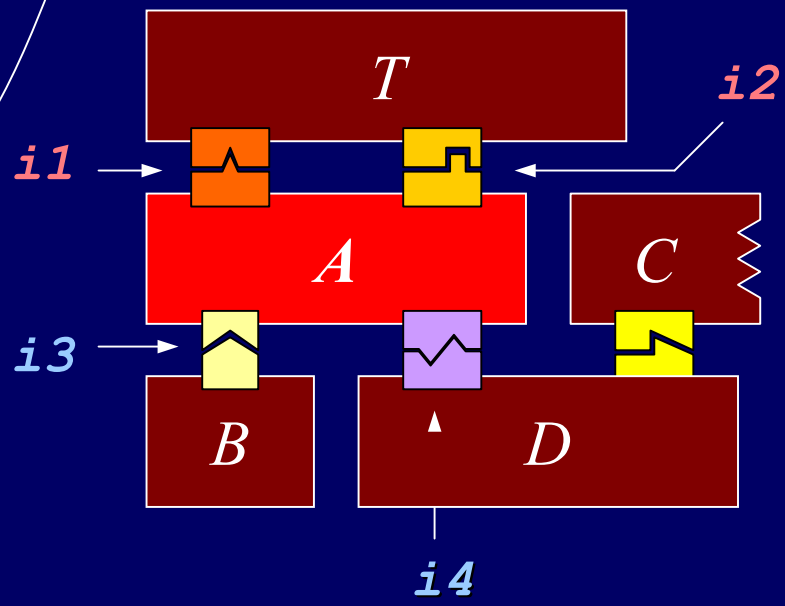
- Azonosító: különálló, globális névtérben, komponens és interfész típus
- deklarációs lista: függvény deklarációk (de nem definíciók), *command* és *event* tárolási osztály specifikációkkal.

NesC komponens definíció

```
module <azonosító> <specifikáció> <implementáció>
```

```
configuration <azonosító> <specifikáció> <implementáció>
```

```
module A
{
   provides
   {
      interface i1;
      interface i2;
   }
   uses
   {
      interface i3;
      interface i4;
   }
}
```



NesC interfész példányok

- Interfészek elnevezése, példányosítása:

```
interface i1;
```

Egy `i1` nevű `i1` típusú interfész

```
interface i1 as iX;
```

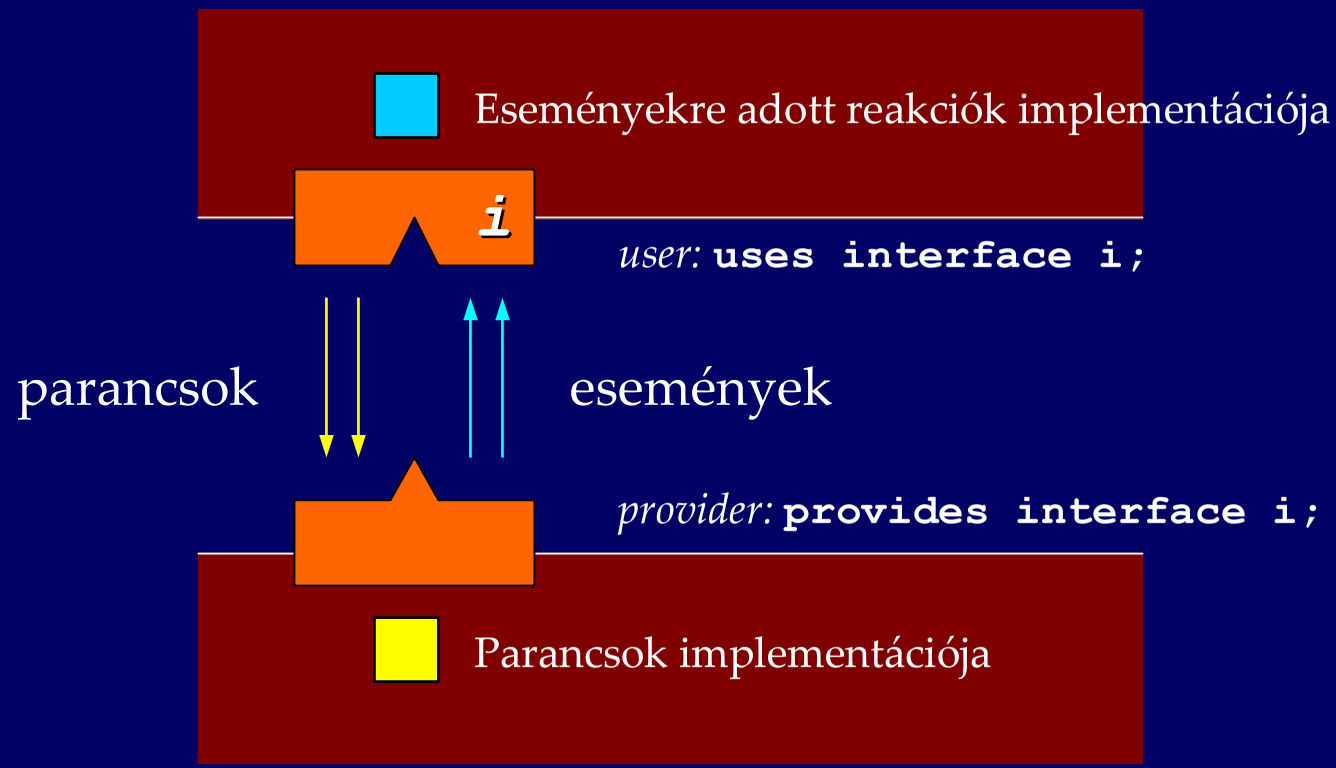
Egy `iX` nevű `i1` típusú interfész

Akkor muszáj, ha több azonos nevű interfész van!

- Modul implementáció
  - Az interfészeken szolgáltatott parancsok implementációja
  - Az interfészeken kapott eseményekre adott reakciók implementációja
  - Itt már függvények definíciói vannak

NesC parancsok és események

- A parancsok és az események „iránya”



NesC paraméterezett interfész

- Az interfészek paraméterezhetők: „tömbösíthetők”:

```
interface SendMsg S[uint8_t id];
```

= 256 darab `SendMsg` típusú `S` névvel jelölt interfész

- Parancs és esemény deklaráció is lehet a specifikációban, nem szükséges interfész definícióban lennie

NesC paraméterezett interfész

- Egy példa:

```
configuration GenericComm
{
    provides
    {
        interface StdControl as Control;
        interface SendVarLenPacket;
        interface SendMsg[uint8_t id];
        interface ReceiveMsg[uint8_t id];
    }
    uses
    {
        event result_t sendDone();
    }
}
implementation {...}
```

NesC interfészek

- Az StdControl interface

```
interface StdControl
{
    command result_t init();
    command result_t start();
    command result_t stop();
}
```

  - Felhasználás: **`init* (start|stop)*`**
  - Minden komponensnek meg kell hívnia az alkomponensei megfelelő függvényét a sajátjában!

NesC hivatkozás

- Hivatkozás parancsra/eseményre:

```
<interfész-azonosító>.<azonosító>
```

- ahol ‘**azonosító**’ valamely parancs v. esemény azonosítója
- Modul interfészre hasonlóan

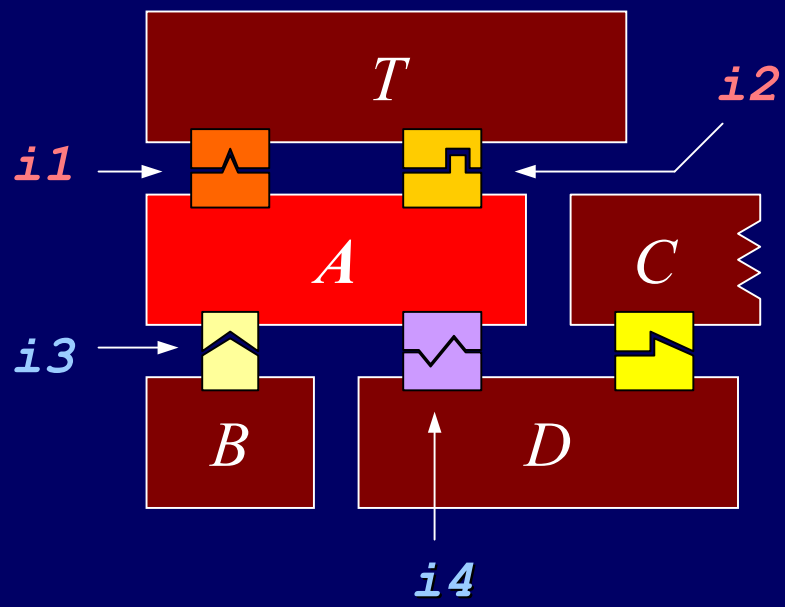
NesC implementáció

- Implementáció:
  - Modulban függvénydefiníciók (és egyéb C def.)
  - Konfigurációban összeköttetés-leírás

```
implementation { fordítási-egység }
```

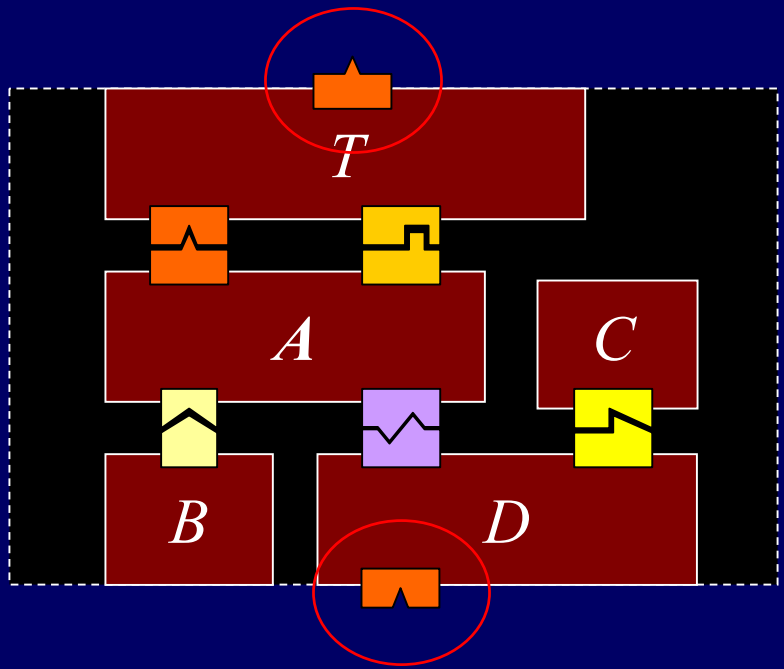
- Modul esetében:

```
implementation
{
    command result_t A.i1() {...}
    command result_t A.i2() {...}
    event result_t A.i3() {...}
    event result_t A.i4()
    {
        ... return SUCCESS;
    }
}
```



NesC konfigurációk

- A konfigurációk komponensek egy összehuzalozott rendszerének „helyettesítő”-i

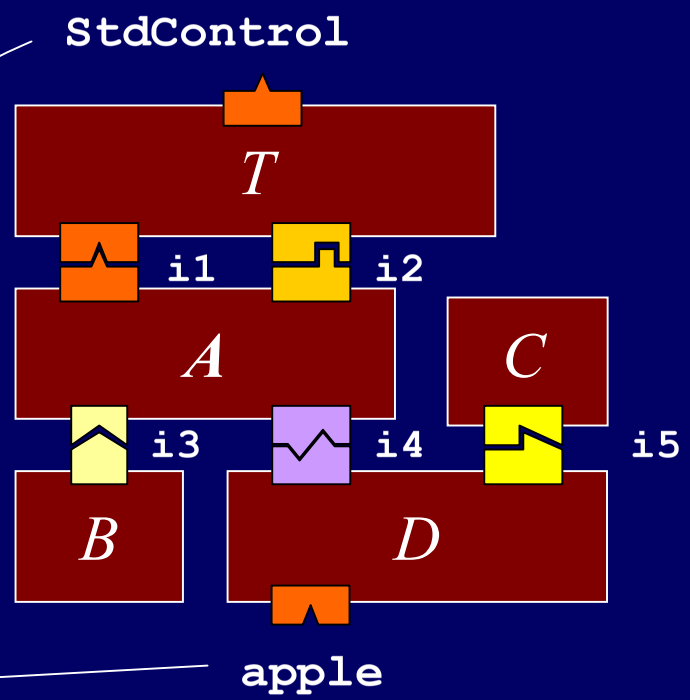


Újrafelhasználható komponens

NesC konfigurációk

```
configuration myConf
{
    provides interface StdControl;
    uses interface apple;
}

implementation
{
    components T, A, B, C, D;
    StdControl = T;
    T.i1 -> A.i1;
    T.i2 -> A.i2;
    A.i3 -> B.i3;
    A.i4 -> D.i4;
    D.i5 <- C.i5;
    apple = D;
    ...
}
```



NesC konfigurációk

- A komponens egyedek (is) átnevezhetők:

```
configuration myConf
{
    provides
        interface StdControl;
    uses interface apple;
}

implementation
{
    components T, A, B, C, D;
    StdControl = T;
    T.i1 -> A.i1;
    T.i2 -> A.i2;
    A.i3 -> B.i3;
    ...
}
```

```
configuration myConf
{
    provides
        interface StdControl;
    uses interface apple;
}

implementation
{
    components T as Q, A, B,
        C, D;
    StdControl = Q;
    Q.i1 -> A.i1;
    Q.i2 -> A.i2;
    A.i3 -> B.i3;
    ...
}
```

NesC taskok

- Taszk: függvényhívás késleltetett végrehajtással
- Taszk indítását kezdeményezik, majd az valamikor elindul és lefut
- Nincs végtelen ciklus és blokkoló utasítás
- Taszk indítását kezdeményezheti:
  - másik taszk
  - esemény (event)
  - utasítás (command)
- Taszk nem szakíthat meg taskot
- Taszkot megszakíthat HW eseményt kezelő függvény
- Hosszabb végrehajtási idejű is lehet

NesC HW IT

- HW eseményt kezelő függvény: HW IT-hez rendelve
- Bármikor futhat
- Más kódot (taszkot vagy másik HW kezelőt) megszakíthat
- Rövid legyen

NesC task kezelés

- Task definíció

```
task void taskname()
{
    ...
}
```

- Task indítás

```
...
post taskname();
...
```

- Taskok futása: FIFO

NesC konkurencia kezelés

1. Taszkok között nincs kiürítés.
2. HW eseményeket (IT) kezelő függvények viszont megszakítást okoznak!

Szinkron kód (SC):

A kódnak azon része (függvények, event-ek, command-ok, taszkok),
amely csak taszkokból érhető el

Aszinkron kód (AC):

A kódnak azon része, amely elérhető legalább egy IT kezelőből

Versenyhelyzet:

SC – SC ✓ **SC – AC** ! **AC – AC** !

Kölcsönös kizárás

- Közös erőforrások védelme:
  - Elérés csak szinkron kódból
  - Atomikus utasítások használata

Atomikus utasitások:

```
atomic
{ // védett szakasz
   ...
}
```

Késleltetik az interrupt végrehajtást:
• válaszidőt növelik, jittert okozhatnak
• command hivás és event küldés kerülendő (válaszidő a használt komponenstől függ)

Aszinkron kód jelzése:

```
async event ...

async command ...
```

A fordító figyelmeztet a potenciális versenyhelyzetre

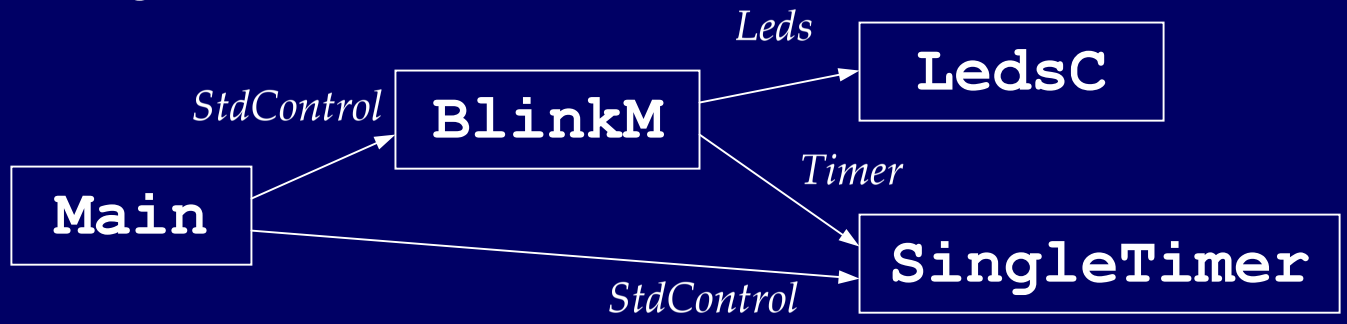
Példaprogram

Cél:

- LED-ek villogtatása
- Gyakoriságot óra vezérli

Felhasznált modulok:

- Main
- LED-vezérlő (LedsC)
- Óra (SingleTimer)
- Villogtató (BlinkM)

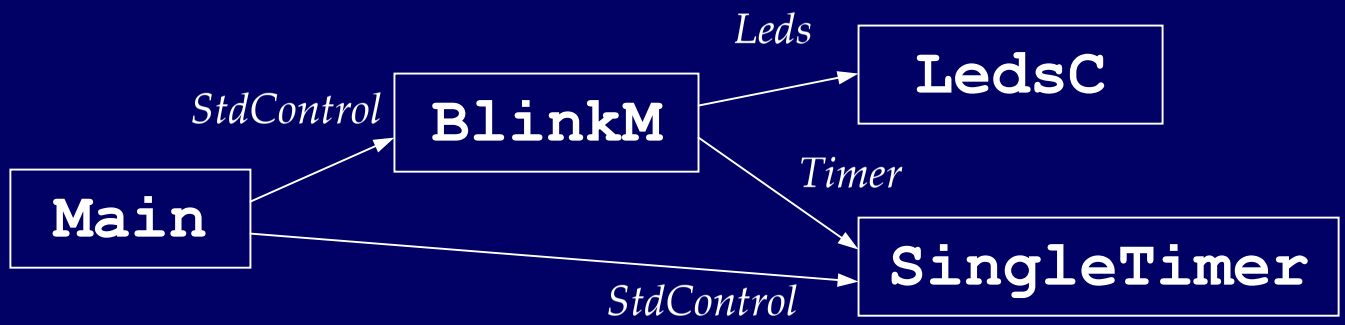


Példaprogram

Blink.nc – főprogram

```
configuration Blink {}

implementation
{
  components Main, BlinkM, SingleTimer, LedsC;
  Main.StdControl -> SingleTimer.StdControl;
  Main.StdControl -> BlinkM.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```



Példaprogram

BlinkM.nc

```
module BlinkM
{
 provides
 {
  interface StdControl;
 }
 uses
 {
  interface Timer;
  interface Leds;
 }
}
```

```
implementation
{
 command result_t StdControl.init()
  { call Leds.init(); return SUCCESS; }

 command result_t StdControl.start()
  { return call Timer.start(TIMER_REPEAT,
1000); }

 command result_t StdControl.stop()
  { return call Timer.stop(); }

 event result_t Timer.fired()
  { call Leds.redToggle(); return SUCCESS; }
}
```

Példaprogram: interfészek

StdControl

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}
```

Leds

```
interface Leds
{
   command result_t init();
   command result_t redOn();
   command result_t redOff();
   command result_t redToggle();
   ...
   command uint8_t  get();
   command result_t
         set(uint8_t value);
}
```

Timer

```
interface Timer
{
    command result_t start(char type, uint32_t interval);
    command result_t stop();
    event   result_t fired();
}
```

Példaprogram

SingleTimer.nc

```
configuration SingleTimer
{
  provides interface Timer;
  provides interface StdControl;
}

implementation
{
  components TimerC;
  Timer = TimerC.Timer[unique("Timer")];
  StdControl = TimerC;
}
```

Fordítás és futtatás

- Támogatott platformok:
  - avrmote
  - mica, mica2, mica2dot, mica128
  - pc

- Fejlesztés menete:

`make` *platform*: **`.ncc`** -> ncc compiler -> **`.exe`** -> avr-objcopy -> **`.srec`**

- **`make`** ***platform*** **`install:`**
  `.srec` -> platformfüggő letöltő

TOSSIM

- Kód változtatás nélkül futtatható PC-n
- Tetszőleges számú egyed (szimulált mote) létrehozható
- Minden mote ugyanazt a programot futtatja

- Fordítás: **`make pc`**
- Futtatás: **`build/pc/main.exe [options] 10`**

- Kimenet: szöveges debug-üzenetek
- Kijelzett események köre változtatható
- A kódba debug-üzenetek külön is beépíthetők
- Rádió kapcsolati mátrix beállítható

TOSSIM példa

- Kód változtatás nélkül futtatható PC-n
- Tetszőleges számú egyed (szimulált mote) létrehozható
- Minden mote ugyanazt a programot futtatja

- Fordítás: **`make pc`**
- Futtatás: **`build/pc/main.exe [options] 10`**

- Kimenet: szöveges debug-üzenetek
- Kijelzett események köre változtatható
- A kódba debug-üzenetek külön is beépíthetők
- Rádió kapcsolati mátrix beállítható

TOSSIM példa

```
$ make pc

$ export DBG=am,led
$ build/pc/main.exe -rf=my-lossy-
   matrix3.nss   3

$ export DBG=usr1
$ build/pc/main.exe -rf=my-lossy-
   matrix6.nss   6
```

Rádió üzenetek, LED-ek állapota

3 mote

6 mote

Veszteséges rádió modell