Mote-PC és PC-Mote soros kommunikáció

A feladat megoldásához lényében a *BlinkToRadio* alkalmazást kell átalakítani úgy, hogy ha a soros port-on keresztül, érkezik egy üzenet, akkor az abban az üzenetben lévő számláló értékének alsó három bit-jét jelenítse meg a led-eken, illetve ha rádión keresztül érkezik egy üzenet, akkor azt pedig továbbítsa a soros port-on keresztül a PC felé. A rádión keresztül érkező üzenetet a múlt órán elkészített eredeti *BlinkToRadio* alkalmazást futtató mote fogja generálni.
Az eredeti *BlinkToRadioC* konfigurációban a következő komponenseket használtuk:

```
...
components MainC;
components LedsC;
components BlinkToRadioP as App;
components new TimerMilliC() as Timer;
components ActiveMessageC;
components new AMSenderC(AM_BLINKTORADIO);
components new AMReceiverC(AM_BLINKTORADIO);
...
```

Itt tulajdonképpen az új alkalmazás létrehozásához, csak minimális változtatásokat kell, eszközölnünk.

Az első lépés, hogy mivel most nem csak a rádiót, hanem a soros port-ot is használni szeretnénk, így nemcsak a rádiót, hanem a soros port-ot is be kell kapcsoljuk. Ennek érdekében szükségünk van egy *SerialActiveMessageC* komponensre, mely segítségével ez megtehető. A második lépés, hogy mivel soros port-on szeretnénk adatot fogadni, ezért szükség van az ezt lehetővé tevő *SerialAMReceiverC* komponensre. Harmadik lépésként, mivel a rádión érkezett adatokat tovább szeretnénk küldeni a soros port-on keresztül a PC-nek, így szükségünk van az ezt biztosító *SerialAMSenderC* komponensre. Az *AMReceiverC* komponensre a rádiós adatfogadás miatt van szükségünk, illetve mivel nem küldünk a rádión keresztül üzentet, így az *AMSenderC* komponensre nincs szükségünk. A *TimerMilliC()* komponensre sincs szükségünk, hisz nem akarjuk periódikusan növelni a számláló értékét, és továbbküldeni. A módosított *BlinkToRadioC* konfiguráció így a következő képen néz ki:

```
...
components MainC;
components LedsC;
components BlinkToRadioP as App;
components ActiveMessageC;
components new AMReceiverC(AM_BLINKTORADIOMSG);
components SerialActiveMessageC;
components new SerialAMSenderC(AM_BLINKTORADIOMSG);
components new SerialAMReceiverC(AM_BLINKTORADIOMSG);
...
```

A módosított komponenseknek megfelelően az egyes komponensek össze huzalozását is módosítani kell az alábbiaknak megfelelően:

```
...
App.Boot -> MainC;
App.Leds -> LedsC;
App.AMControl -> ActiveMessageC;
App.Receive -> AMReceiverC;
App.SerialControl -> SerialActiveMessageC;
App.SerialAMSend -> SerialAMSenderC;
App.SerialReceive -> SerialAMReceiverC;
...
```

Mivel két *SplitControl* interface-ünk van, ezért az egyiket nevezzük *AMControl*-nak, a másikat pedig *SerialControl*-nak. Ezeket kössük be a *SerialActiveMessageC* komponensbe, illetve az *ActiveMessageC*-be. Emellett kelleni fog a soros port-hoz tartozó *AMSend* és

Receive interface. Ezeket nevezzük el *SerialAMSend*-nek és *SerialReceive*-nek. Az *AMSend* és a *Timer* interface-ek bekötésére nincs szükségünk. Módosítsuk még az *AM_BLINKTORADIO*-t *AM_BLINKTORADIOMSG*-re. Ennek jelentőségére a későbbiekben fogok kitérni. Figyeljük meg, hogy a rádiós és a soros port-i adatküldéshez nagyon hasonló komponenseket használunk, a különbség csak a *Serial* kulcsszó.

Második lépésként mivel módosítottuk az *AM_BLINKTORADIO*-t *AM_BLINKTORADIOMSG*-re, ezért hajtsuk végre ezt a módosítást a *BlinkToRadio.h* heade file-ban is.

```
...
enum{
	AM_BLINKTORADIOMSG = 6,
};
...
```

A harmadik lépés, hogy a *BlinkToRadioP*-t módosítsuk. Első lépésben a használt interface-eket módosítsuk, az alábbiaknak megfelelően:

```
	...
	uses interface Boot;
	uses interface Leds;
	uses interface SplitControl as AMControl;
	uses interface Receive;
	uses interface AMSend as SerialAMSend;
	uses interface Receive as SerialReceive;
	uses interface SplitControl as SerialControl;
	...
```

A *Timer* és az *AMSend* interface-re nincs szükségünk, viszont kell a két *SplitControl* interface az egyik neve *AMControl*, a másik neve *SerialControl* lesz. Kell a soros port-hoz tartozó *AMSend* és *Receive*, melyeket *SerialAMSend*-nek és *SerialReceive*-nek nevezünk. Végül a rádiós adatfogadáshoz pedig szükségünk van a *Receive* interface-re.

Negyedik lépésként módosítsuk az alkalmazásunkat. A definiált változók közül a *counter*-re és a *pkt*-re nincs szükségünk, a *busy* változót azonban tartsuk meg, szerepe hasonló lesz, mint a *BlinkToRadio* alkalmazásban. Kell azonban két új változó, melyek szerepét a későbbiekben meglátjuk:

```
	...
	message_t freeMsg;
	message_t *freeMsgPtr=&freeMsg;
	...
```

Az ötödik lépés, hogy mivel most mind a soros port-ra, mind a rádió-ra szükségünk van, ezért mindkettőt be kell kapcsoljuk. Mivel a készítendő alkalmazás olyan, hogy a rádión érkező üzenetet küldi el a soros port-on keresztül, ezért elsőként a soros port-ot kapcsoljuk be, és ha az bekapcsolt, csak utána kapcsoljuk be a rádiót, hogy ne fordulhasson elő olyan eset, hogy akkor jön a rádión üzenet, amikor a soros port még nincs bekapcsolva.
A kódrész a következő képen néz ki:

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

	event void SerialControl.startDone(error_t err) {
		if (err == SUCCESS) {
			call AMControl.start();
		}
		else {
			call SerialControl.start();
		}
	}
```

```
      event void AMControl.startDone(error_t err) {
            if (err != SUCCESS) {
                  call AMControl.start();
            }
      }
      event void SerialControl.stopDone(error_t err) {}
      event void AMControl.stopDone(error_t err) {}
...
```

Ha a mote elindult, azaz a *Boot.booted()* event generálódott, akkor bekapcsoljuk a soros port-ot. Ha ez sikeresen megtörtént, akkor pedig bekapcsoljuk a rádiót. Ha valamelyiket nem sikerül bekapcsolni, akkor megpróbáljuk újra bekapcsolni azt. A *BlinkToRadio* alkalmazáshoz hasonlóan deklarálni kell az *AMControl.stop()*-ot, és a *SerialControl.stop()*-ot is.

A hatodik lépésben a *Timer.fired()* event-et töröljük. Ennek a helyét fogja átvenni a *Receive.receive()* event, mely a rádión keresztüli adat fogadást valósítja meg. Az így beérkezett csomagot kell, továbbküldjük, a soros port-on keresztül. Az ehhez szükséges kódrészlet a következőképpen néz ki:

```
...
event message_t* Receive.receive(message_t* msg, void* payload, uint8_t
len) {
            if (busy)
                  return msg;
            if (call SerialAMSend.send(AM_BROADCAST_ADDR, msg,
sizeof(BlinkToRadioMsg))!= SUCCESS)
                  return msg;

            busy = TRUE;
            return freeMsgPtr;
      }
...
```

A *busy* flag azt fogja jelölni, hogy a soros port-on keresztül végrehajtandó üzent küldés befejeződött-e. Tehát először megvizsgáljuk, hogy a soros adatküldés befejeződött-e. Ha nem, akkor a rádión jött üzenetet visszaadjuk, hisz nem tudjuk elküldeni. Ha nem foglalt a soros port, akkor meghívjuk a *SerialAMSend.send()* parancsot, és a beérkezett üzenetet megpróbáljuk továbbítani a soros port-on keresztül. Ha ez nem sikerül, azaz a *SerialAMSend.send() FAIL*-el tér vissza, akkor is visszaadjuk az üzenetet. Végül, ha minden rendben, azaz elkezdődik a soros port-i adatküldés, akkor a *busy*-t *TRUE*-ra állítjuk, és visszatérünk a *freeMsgPtr*-el, mely egy olyan pointer, mely egy üres *message_t* struktúrára mutat. Erre azért van szükség, mert, ha hozzákezdtünk a soros port-i adatküldéshez, az adott *message_t* struktúrát használva, azt nem adhatjuk vissza a rádió-nak, hogy abba írhassa a következő üzenetet, amíg a soros port-i adatküldés be nem fejeződött. A soros port-i adatküldés befejezéséről pedig a *SerialAMSend.sendDone()* event-je segítségével kapunk értesítést, és itt a következőeket csináljuk:

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

Itt az előbb a küldés miatt fenntartott *message_t* struktúrát átadjuk a *freeMsg* pointernek, hogy majd a következő *Receive.receive()* event-ben ezt visszaadjuk a rádiós stack-nek, illetve a *busy* flag-et *FALSE*-ra állítjuk.

A hetedik lépéseként pedig megírjuk azt a részt, amikor a soros port-on érkező üzenetet megjelenítjük a led-eken. Ez pedig a következőképpen néz ki:

```
...
event message_t* SerialReceive.receive(message_t* msg, void* payload,
uint8_t len){
```

```
      BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
      call Leds.set(btrpkt -> counter);
      return msg;
}
...
```

Lényegében ez a *Receive.receive()* event-en csak annyi módosítást jelent, hogy átnevezzük *SerialReceive.receive()* event-nek, hogy a soros port-on érkező adatok kezelésére vonatkozzon.

A nyolcadik lépés a soros port-on az adatokat leküldő java alkalmazás a *SerialSend.java* és a soros port-on érkező adatokat a konzolra kiíró a *SerialReceiver.java* elkészítése. Először is a rádiós adatküldéshez hasonlóan ahol egy struktúrát definiáltuk, az adatok rádiós csomagbeli elhelyezkedésének megadására, a két java kód számára is szükség van egy java osztályra, ami alapján tudják kezelni és értelmezni a soros port-on érkező adatokat. Ez az osztály tartalmazza a soros port-on érkező csomagban lévő értékek kiolvasásához és a soros port-on elküldendő értékek csomagba beírásához szükséges setter és getter függvényeket. A *mig* a Message Interface Generator, az általunk definiált header (*BlinkToRadio.h*) alapján képes megcsinálni ezt az osztályt számunkra. Ennek a *mig* eszköznek a meghívása a *Makefile*-ban történik, a következő képen:

```
COMPONENT=BlinkToRadioC
BUILD_EXTRA_DEPS = SerialMsg.class
CLEAN_EXTRA = SerialMsg.class SerialMsg.java

SerialMsg.class: SerialMsg.java
      javac SerialMsg.java

SerialMsg.java: BlinkToRadio.h
      mig java -target=null -java-classname=SerialMsg BlinkToRadio.h
BlinkToRadioMsg -o $@
include $(MAKERULES)
```

A *BUILD_EXTRA_DEPS* sor azt mondja meg, hogy az alkalmazásunknak vannak külső függőségei, ez pedig a *SerialMsg.class*. A *CLEAN_EXTRA* sor azt mondja meg, hogy milyen plusz file-okat kell törölni a *make clean* utasítás meghívása után.
A következő sor lényegében megadja, hogy mitől függ a *SerialMsg.class*, és létrehozza azt a *SerialMsg.java* lefordításával.
A következő sor megadja, hogy hogyan kell létrehozni, a *SerialMsg.java*-t.

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

Mivel egy tool segítségével hozzuk létre a szükséges java file-t és osztályt, a *BlinkToRadio.h* header file-ban lévő struktúra alapján, ez megköveteli, hogy a struktúra neve és az üzenet AM típusának neve megegyezzen. Ezért kellett kicseréljük az *AM_BLINKTORADIO*-t *AM_BLINKTORADIOMSG*-re. A *net.tinyos.message* tartalmaz egy *MoteIF* osztályt. Ez teszi lehetővé, hogy könnyen tudjunk fogadni és küldeni *mig* által generált csomagokat. 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 kódunk hogyan éri el a

mote-ot. Például *–comm serial@/dev/ttyUSB0:telosb.* A SerialSend.java kódunknak a következőeket tartalmazza:

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

class SerialSend{

	private MoteIF moteIF;

	public SerialSend(MoteIF moteIF){
		this.moteIF=moteIF;
	}

	public void send(String counter)
	{
		SerialMsg msg=new SerialMsg();
		try{
			msg.set_counter(Integer.parseInt(counter));
			moteIF.send(MoteIF.TOS_BCAST_ADDR,msg);
			out.println("Message sent to the mote ");
		}catch(IOException e)
		{
			out.println("Cannot send message to mote ");
		}
	}

	public static void main(String[] args) throws Exception
	{
		PhoenixSource phoenix = null;
		MoteIF mif = null;
		String data=null;

		if( args.length == 1 ){
			phoenix =
			BuildSource.makePhoenix(PrintStreamMessenger.err);
			data=args[0];
		} else if ( args.length == 3 && args[0].equals("-comm") ) {
			phoenix = BuildSource.makePhoenix(args[1],
			PrintStreamMessenger.err);
			data=args[2];
		} else {
			System.err.println("usage: java TestSerial [-comm
			<source>] [data to send]");
			System.exit(1);
		}
		mif = new MoteIF(phoenix);
		SerialSend app= new SerialSend(mif);
		app.send(data);
		System.exit(0);
	}
}
```

A kódban megvizsgáljuk a paramétereket, és megnyitunk egy csomagforrást, valamint elküldjük a küldendő adatként megadott értéket a *send()* függvény segítségével. Ebben a függvényben létrehozunk egy *SerialMsg* típusú *msg*-t majd ebbe a *SerialMsg*-ben definiált

setter függvény segítségével bepakoljuk az elküldendő értékeket, majd elküldjük a soros port-on keresztül. A *SerialReceive.java* a következőeket kell, hogy tartalmazza:

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

class SerialReceive implements MessageListener{

	private MoteIF moteIF;

	public SerialReceive(MoteIF moteIF){
		this.moteIF=moteIF;
		this.moteIF.registerListener(new SerialMsg(),this);
	}

	public void messageReceived(int dest_addr,Message msg){
		if (msg instanceof SerialMsg) {
			SerialMsg serialData = (SerialMsg)msg;
			out.println("Data received: " +
			serialData.get_counter());

		}
	}

	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 TestSerial [-comm
			<source>]");
			System.exit(1);
		}
		mif = new MoteIF(phoenix);
		SerialReceive app= new SerialReceive(mif);
	}
}
```

A csomagok fogadásához először is a konstruktorban szükségünk van egy *registerListener()* függvényre, mely regisztrálja az osztályunkat, a *SerialMsg* fogadására, majd amikor egy ilyen csomag érkezik, akkor a *messsageReceived()* függvény mindig meghívódik. Itt van lehetőségünk kiolvasni a csomag tartalmát, és kiíratni azt a konzolra. Fordítsuk le a két java file-t is és teszteljük az alkalmazásunkat.