ACTIVITY1 - IoT

By Stefano Agresti, Gianluca Guarro

PERSONAL INFO

***Name**: Stefano Agresti*

***Person code**: 10685203*

***Matricola**: 913079*

-

***Name**: Gianluca Guarro*

***Person code**: 10658177*

***Matricola**: 918696*

ASSIGNMENT

Create a Cooja simulation with three TinyOS (sky motes), called 1,2 and 3. The three motes communicate over the radio. The message is composed by a counter and the sender id. All the messages are sent in BROADCAST. Messages are sent at: 1Hz for mote 1, 3Hz for mote 2, 5Hz for mote 3. Messages sent by mote 1 toggle led0, messages sent by mote 2 toggle led1, messages sent by mote 3 toggle led2, messages with 'counter mod 10' == 0 turn off all the LEDs. The counter is incremented every time a message is received.

EXECUTION

1. MESSAGE CREATION

First step is to build the structure of the message inside the header file called "*Activity1.h*". This file is very simple, it just specifies how a message is structured and defines a couple of variables used later. We know from the assignment that the message needs to contain two values, one for the counter and one for the sender id, so we inserted two unsigned ints of 16 bits to maintain these two information. The four constants defined here are used for the different frequencies to be used in the different motes, plus the *AM_RADIO_COUNT_MSG* constant, used to initialize the *AMSenderC* and *AMReceiverC* components.

2. CONFIGURATION

Second step is to link the components used in the source code to the interfaces that control the interaction with the various modules implemented inside the mote. This task is performed inside the "*Activity1AppC.nc*" file, which, again, is rather simple. First we specify which components we need, then we wire them to those we use in the next file ("*Activity1C.nc*"). Among the components we need, there are *AMSenderC* and *AMReceiverC* to communicate over the radio, a Timer used to send messages periodically and *LedsC* to toggle the motes' leds on/off.

3. MODULES AND INTERNAL LOGIC

Next, we need to actually implement the logic inside the mote. The idea is simple: once the mote is correctly started, a periodic timer will start (with different frequencies according to the mote id); each time the timer is fired, the mote sends a message containing its counter, plus its own id; when a message is received, the internal counter is increased and the various leds are turned on/off according to the assignment rules. Let's see in details how these steps are accomplished:

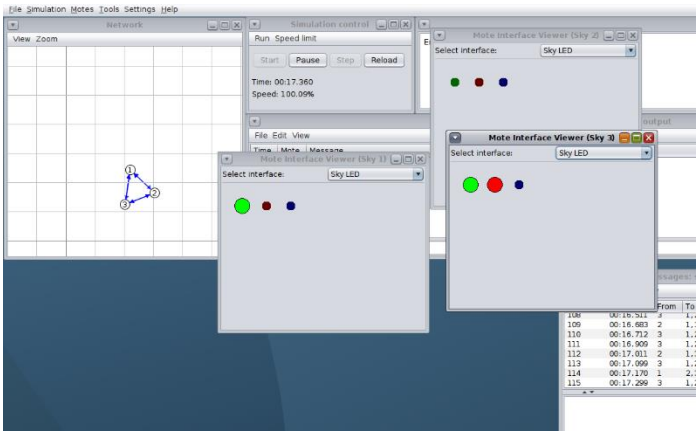
- START: This operation is handled by the *AMControl*, called inside the *Boot* event. Once the mote is started, the event *AMControl.startDone* is generated. After checking its success, we initialize the node id with the id assigned by Cooja in the *TOS_NODE_ID* variable, before starting a timer.
- TIMER FIRED: This generates the event *MilliTimer.fired*. Inside the event function, we first check if the mote is busy sending another message; if it's not, we prepare and send a message through the *AMSend* component.
- MESSAGE RECEIVED: This generates the event *Reciever.receive*. After checking the correct size of the message, we unpack it, increase the internal counter and perform a different action depending on the message content (if sender id is 1, led0 is turned on/off,… and if counter is multiple of 10, all leds are turned off).

4. MAKEFILE

Last step is to prepare the *Makefile*. Again, nothing complicated, just two lines of code.

5. SIMULATION

Once the code is ready, we can build it using the "*make telosb*" command and run a simulation on *Cooja*. To do that, it's sufficient to open a new simulation, add three sky motes selecting the *main.exe* file created in the *Build* folder and press start. Showing the Leds on the three motes, it will be possible to observe how they're turned on and off at different frequencies. Looking at the radio messages, we can also confirm that this is happening with the right frequencies.



Simulation on Cooja