Internet of Things – A.Y. 2020/2021

Home Challenge #3 – Diego Savoia (Personal Code: 10535515)

To implement the code requested for this challenge, I used the example seen during the laboratory session (RadioCountToLeds) as a reference. In the header file (named foo.h) I modified the existing message structure so to include both the counter and the mote id.

The fooAppC.nc file is basically the same as the one in the example. Apart from the standard MainC and fooC components, we have: the AMSenderC, the AMReceiverC and the ActiveMessageC components to manage the communication; a TimerMilliC component to handle the timer of the mote; a LedsC component to handle the LEDs of the mote.

The main modifications are done in the fooC.nc file. The Boolean variable "locked" and the mote counter are initialized, as well as the packet variable. When the mote is booted and the AMControl.startDone is executed, the timer of the mote is started with a period that depends on the id of the mote: in particular, a simple "if" case on the variable TOS_NODE_ID allows to start the timer of 1 Hz if mote 1 is executing the code, 3 Hz if mote 2, 5 Hz if mote 3.

Every time the timer is fired, the mote enters the function MilliTimer.fired. Here, a new message is created (with the structure defined in foo.h): the field counter is filled with the value of the counter variable of the mote, while the field id is filled with the value of the TOS_NODE_ID variable of the mote. If the packet is sent correctly (of course in broadcast), the locked variable is set to TRUE and then released after the sending routine is done.

Upon receiving a message of this kind, the mote increments its own counter variable (which is of course potentially different from the one contained in the received message). Then, a simple "if" case handles all the requested cases:

- If the counter contained in the received message modulo 10 is equal to zero, all the three LEDs of the mote are turned off.
- Else, if the id field of the received message is equal to 1, then led0 of the mote is toggled.
- Else, if the id field of the received message is equal to 2, then led1 of the mote is toggled.
- Else, if the id field of the received message is equal to 3, then led2 of the mote is toggled.

Notice that, since the sending mote does not receive its own messages, this means that mote 1 will never toggle its led0, as well as mote 2 will never toggle its led1 and mote 3 will never toggle its led2.

To better debug the code, I also included the printf function, adding the TestPrinfC.nc and TestPrintfAppC.nc files in the challenge folder, including "printf.h" in my files, and adding the PrintfC and SerialStartC components in the fooAppC.nc file, as well as modifying accordingly the makefile to include this functionality. I added a printf in each one of the four if cases listed above, to understand when the LEDs were set to 000, or which LED was being toggled.

Finally, I created a new simulation in Cooja, adding three Sky motes, each of them executing the main.exe file obtained by the compiling process. I could see all the printf outputs in the "Mote output" section. Then, using as filter "ID:2" I isolated all the printfs (and so the "LEDs toggling") of mote 2. Writing down the LEDs status according to the printed modifications, I could retrieve the LEDs values in the form "led2led1led0". As noticed before, led1 is never toggled by mote 2.