Politenico di Milano

Dipartimento Elettronica, Informazione e Bioingegneria

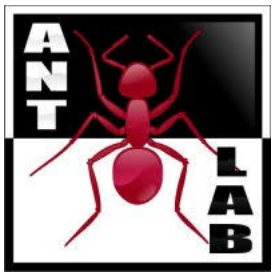
Homework IoT Project

RadioSenderToLeds

Author:
Francesco Monti
Matr: 919755

Supervisor:
Dr. Edoardo Longo
Dr. Matteo Cesana

March 19, 2020



Abstract

This document contains the documentation for the first activity for the course "Internet of Things", Academic Year 2019/2020.
We firstly list the requirements, then we present our implementation.
All the code can be found in the following GitHub repository: https://github.com/Framonti/IoT_Projects

0.1 Requirements

The goal of the activity is to develop a small application for Telosb, a small component that supports TinyOS. This application should be deployed on three different motes that communicate with each other through the radio component; they will react to the messages, toggling their LEDs according to some rules.
The motes send their messages in broadcast, with different frequencies (1 Hz for mote1, 3 Hz for mote2, 5 Hz for mote3).

0.2 Implementation

The implementation follows the guidelines presented during the lectures; therefore, we have three different files:*RadioSenderToLeds.h*, *RadioSenderToLedsAppC.nc*, and *RadioSenderToLedC.nc*.
In this document, we discuss only the application logic part. The other two files are necessary, but only state what kind of components we're using and therefore they don't offer discussion points. We just want to point out some implementation decisions and not offer a detailed commentary of the code.

```
event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      if(TOS_NODE_ID == 1){
          call MilliTimer.startPeriodic(TIMERMILLISNODE1);
       }
       else if (TOS_NODE_ID == 2){
         call MilliTimer.startPeriodic(TIMERMILLISNODE2);
       }
       else {
         call MilliTimer.startPeriodic(TIMERMILLISNODE3);
[...]
```

The first design decision we made is related to how we can manage the different frequencies for each node. In this exercise, as shown in the code, we decided to check TOS_NODE_ID in order to instantiate the correct timer for each mote. A more correct approach would have been to create a different logic file for each node, in order to reduce code length and logic complexity. However, we preferred to keep the number of delivered item as low as possible.

```
1  event message_t* Receive.receive(message_t* bufPtr, void*
    payload, uint8_t len) {
2    if (len == sizeof(radio_sender_msg_t)) {
3      radio_sender_msg_t* rsm = (radio_sender_msg_t*)payload;
4      counter++;
5      if((rsm-> counter % 10) == 0){
6        call Leds.led0Off();
7        call Leds.led1Off();
8        call Leds.led2Off();
9        return bufPtr;
10     }
11     else {
12       if (rsm-> sender_id == 1) {
13     call Leds.led0Toggle();
14   }
15       else if (rsm -> sender_id == 2){
16     call Leds.led1Toggle();
17       }
18       else {
19     call Leds.led2Toggle();
20 [...]
```

The previous code simply implement the logic required when receiving a message and should be self-explanatory. To note, when a node receives a message with counter exactly divisible by 10, we don't update the LEDs according to the sender_id. This is not explicitly written in the requirements (we could have set all the LEDs off, but then turn on the one corresponding to the sender_id), but we felt this behaviour to be more compliant.

0.3 Simulation

We simulate our implementation with Cooja, and we obtained the required behaviour. As expected, each mote has only two LEDs blinking, with different frequencies.