Wireless Sensor Networks – Lab assignment 1

Preperations

1) Read through this material and try to understand the code for sensing temperature and humidity, and sending and receiving packets in TinyOS. The code is also available in the course folder at either bilda.kth.se or KTH social.

2) Read through the tutorial about radio communication in TinyOS
http://docs.tinyos.net/index.php/Mote-mote_radio_communication

Sensor node
(Temp and humidity)

Base station

Objective

The goal with this lab assignment is to understand how to send the temperature and humidity samples from a sensor board to a base station and display the samples on a computer screen. A packet acknowledgement function needs to be implemented and the sample values needs to be presented in SI units (International System of Units). The radio on the sensor node should be turned on only when sending the samples, to minimize the power consuming of the battery. Example code for sensing and, sending and receiving packets has been provided.

Requirements

The requirements are the followings:

- Send the temperature and humidity samples from the sensor node to the base station every 5 seconds. The base station should display the both recent samples on the computer screen at same time.
- Turn on the radio only when the sensor node is sending. Let one of the LEDs present when the radio is on and a second LED when it's sends a packet.
- Implement your own acknowledgement function for the packets that are sent from the sensor board. If a packet is lost, send the same sample again until the receiver receives and acknowledges it. The acknowledgement packet should have a maximum payload of 16-bits.
- Convert the raw temperature and humidity samples to SI units (maximum 2 decimals) and display it on the computer screen.
- If the temperature or humidity (in SI units) increases/decreases more than 15% than the normal room temperature/humidity, send the samples every 500 milliseconds to the base station. If it's less than 15%, send it again every 5 seconds. The fourth requirement must be solved before solving this requirement.

Converting sample values

The sample values from the temperature and humidity can be converted to SI units as follows:

For Temperature, Oscilloscope returns a 14-bit value that can be converted to degrees Celsius:

$$temperature = -39.60 + 0.01*t$$

where t is the raw output of the temperature sensor.

Humidity is a 12-bit value that is not temperature compensated.

$$humidity = -4 + 0.0405*h + (-2.8 * 10^{-6})*(h^2)$$

where h is the raw output of the relative humidity sensor.

Hints

- Use code for reading humidity and temperature found in the provided source code.
- It may be easier to create one code for the sensor node and another code for the base station.
- Use the Java code, *monitor_output*, to display the sample values from the node. Convert the raw sample values to SI units in the Java code. Important: NesC doesn't support float variables.
- The LEDs can be useful for debugging. An example is turning on a LED when a function executes.
- Using strings in java: http://docs.oracle.com/javase/7/docs/api/java/lang/String.html
- Converting string to double, in Java :
  `new temp =  new Double("123").doubleValue();`

Getting started with the example code

1) Start the VMware application and choose Advantic-TinyOS.

2) Download the framework(labb1.zip) of this lab assignment from course folder and save it in the folder

   `/opt/tinyos-main-read-only/KTH/`

   The framework is measuring the temperature sensor and shows the raw sample value from the sensor on the screen. Go to the folder

   `/opt/tinyos-main-read-only/KTH/`

   and type in

   `unzip labb1.zip`

   to unpack.

3) Plug in the Tmote Sky platform to the USB interface of the computer and type in
   `motelist`
   to see if the system has recognized and installed the node. If not, call the instructor.

To start sensing temperature and humidity.

4) Go to the folder labb1/tempandhumid and type in

   `make tmote install` if you have the blue sensor nodes

   or

   `make xm1000 install` if you have the red sensor nodes

   to install the application. The application measures the temperature and displays on the screen. Wait for that the application is installed successfully on the node and go on with next step.

5) Start the java application which prints everything it receives from the node,
   `java Monitor_output –comm serial@/dev/ttyUSB0:telos`

6) Now should see the raw value from the temperature on the screen, updated every second.

To start sending data over the air

7) Go to the folder labb1/radio and change the AM_type_id in the file RadioAndAck.h to a random number (1-255). Use this number on all of your nodes.

8) To install the application, type in

```
make tmote install,1
```
or
```
make xm1000 install,1
```

This will install and set the node identity to 1. Unplug the first node and connect the second node to the USB and type in

```
make tmote install,2
```
or
```
make xm1000 install,2
```

to install the application on the second node and set the node identity to 2 .

9) Start the java application which prints everything it receives from the node,
```
java Monitor_output –comm serial@/dev/ttyUSB0:telos
```

10) Each node will display on the LEDs, the least significant bit is has received from the other node. The node connected to the computer will also display on the computer screen the node id and the counter received from the sensor node.

Task

Change the code and fulfill the requirements which you find on the first page.

```
/*
##############################################################################
The application sends the raw samples from the temperature sensor every
second on the computer. The java application on the computer will receive
the sample and display it on the computer screen.
java Monitor_output –comm serial@/dev/ttyUSB0:telos
##############################################################################
*/
//TempandhumidAppC.nc

#define NEW_PRINTF_SEMANTICS
#include <Timer.h>
#include "printf.h"

configuration TempandhumidAppC {
}
implementation
{
  components MainC,
    LedsC,
    PrintfC,
    TempandhumidC,
                      SerialStartC,
    new TimerMilliC() as Timer0,
    new TimerMilliC() as Timer1,
    new SensirionSht11C() as Temp, //Component used for measuring temp
    new SensirionSht11C() as Humidity;//Component used for measuring
humidity

  TempandhumidC -> MainC.Boot;

  //On-board leds
  TempandhumidC.Leds -> LedsC;

  //Timers
  TempandhumidC.Timer0 -> Timer0;
  TempandhumidC.Timer1 -> Timer1;

  //PrintfFlush
  //TempandhumidC.PrintfControl -> PrintfC;
  //TempandhumidC.PrintfFlush -> PrintfC;

  //Temperature and Humidity
  TempandhumidC.ReadTemp -> Temp.Temperature;
  TempandhumidC.ReadHumidity -> Humidity.Humidity;
}


##############################################################################

//TempandhumidC.nc

#include <Timer.h>
#include "printf.h"

module TempandhumidC{
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Read<uint16_t> as ReadTemp;
  uses interface Read<uint16_t> as ReadHumidity;
}
```

```
implementation{
  void requestSensorValue();

  event void Boot.booted()  {
     requestSensorValue();
  }

  //Requests the temperature sample value
  void requestSensorValue()  {
      call ReadTemp.read();
  }

   //Displays the temperature on the screen and the temp status on the leds
  void displayTemp(uint16_t val)  {
    printf("Temp: %u ",val);

    //Prints all it has in its print buffer
    printfflush();

    //Starts the timer and sets it to 1 second. Timer0.fired event will
    //be called after 1 second.
    call Timer0.startOneShot(1000);
  }

  //This function is executed when the temperature is measured from the
sensor
  event void ReadTemp.readDone(error_t error, uint16_t value)  {
    displayTemp(value);
  }

  //This function is executed when the humidity is measured from the sensor
  event void ReadHumidity.readDone(error_t error, uint16_t value)  {
  }

  event void Timer0.fired(){
    call Leds.led0Toggle(); //Turns the led on and off
    requestSensorValue(); //Requests sensor value
  }

  event void Timer1.fired(){
  }
 }
```

```
//File: RadioAndAck.h
################################################################################
#Sends a packet containing the NODE_ID and a counter every 500ms. The counter  #
#increment for every packet it sends.                                          #
#It also receives a packet containing a counter value and displays the least   #
#significant bits of the counter on the LEDs.                                  #
################################################################################

#ifndef RADIOANDACK_H
#define RADIOANDACK_H

enum {
  AM_RADIOANDACKMSG = 8,
};

//The message sent over the air.
//The default maximum payload is 28byte
typedef nx_struct RadioAndAckMsg {
  nx_uint16_t nodeid;  //NODE ID of the node, 2byte
  nx_uint16_t counter; //A counter of number of sent packet, 2byte
} RadioAndAckMsg;

#endif
```

```
//File: RadioAndAckApp.nc
#include <Timer.h>
#include "RadioAndAck.h"
#include "printf.h"

configuration RadioAndAckAppC {
}
implementation {
  components MainC,
    LedsC,
    RadioAndAckC,
    ActiveMessageC,
    new TimerMilliC() as Timer0,
    new AMSenderC(AM_RADIOANDACKMSG),
    new AMReceiverC(AM_RADIOANDACKMSG);


  RadioAndAckC.Boot -> MainC;
  RadioAndAckC.Leds -> LedsC;
  RadioAndAckC.Timer0 -> Timer0;

  RadioAndAckC.Packet -> AMSenderC;
  RadioAndAckC.AMPacket -> AMSenderC;
  RadioAndAckC.AMSend -> AMSenderC;
  RadioAndAckC.AMControl -> ActiveMessageC;
  RadioAndAckC.Receive -> AMReceiverC;
}
```

```
/*
FILE: RadioAndAckC.nc
##############################################################################
#Sends a packet containg the NODE_ID and a counter every 500ms. The         #
#counter increment for every packet it sends.                               #
#It also receives a packet containg a counter value and displays the        #
#least signifigant bits of the counter on the leds.                         #
##############################################################################
*/
#include <Timer.h>
#include "RadioAndAck.h"
#include "printf.h"

module RadioAndAckC {
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Packet;
  uses interface AMPacket;
  uses interface AMSend;
  uses interface SplitControl as AMControl;
  uses interface Receive;
}

implementation {
  uint16_t counter = 0;  //A counter used for counting the sent packets.
  bool busy = FALSE; //TRUE if the radio is used
  message_t pkt;

  event void Boot.booted() {
    call AMControl.start(); //Starts the radio
    call Timer0.startOneShot(500); //Start the timer which will activate
                                   //the app.
  }

  //Event used for sending a packet when Timer0 times out
  event void Timer0.fired() {
    counter++;
    if (!busy) {
      RadioAndAckMsg* btrpkt = (RadioAndAckMsg *)(call
Packet.getPayload(&pkt, sizeof(RadioAndAckMsg)));
      btrpkt->nodeid = TOS_NODE_ID; //Node ID
      btrpkt->counter = counter; //Counter for count of sent packet

      if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(RadioAndAckMsg))
      == SUCCESS) {
           busy = TRUE; //TRUE if the radio has resourse to send a packet
      }
    }
    call Timer0.startOneShot(250); //Times out every 250ms and the timer
                                   //sends a packet
  }

  //When a packet is received, this event will be called
  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t
len) {
    if (len == sizeof(RadioAndAckMsg)) {
      RadioAndAckMsg* btrpkt = (RadioAndAckMsg*)payload;
      call Leds.set(btrpkt->counter);
      printf("NODE ID: %u Counter: %u\n",btrpkt->nodeid ,btrpkt->counter);
      printfflush();
    }
    return msg;
  }
```

```
  //When the packet has been sent successfully
  event void AMSend.sendDone(message_t* msg, error_t error) {
    if (&pkt == msg) {
      busy = FALSE; //Sets that the radio is not busy.
    }
  }


  //When the radio signals that it may have been started
  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) { //The radio has been started successfully
    }
    else {
      call AMControl.start(); //The radio has not been started. It tries
                              //again to start.
    }
  }


  //When the radio has been turned off successfully
  event void AMControl.stopDone(error_t err) {
  }

}
```