Instituto Superior Técnico

Applications and Computation for the Internet of Things

2nd Lab work: Network of Sensor and Actuators

Group:		
Student 1:		
Student 2:		
Student 3:		

Goal:

The goal of this work is to sense physical quantities and to control actuators according to the measured environment. Sensors and actuators will be driven by controllers communicating over a wireless link based on the Enhanced ShockBurst (ESB) protocol.

Description:

Build an embedded system to simultaneously control 3 LEDs depending on the state of 3 different sensors – temperature, rotation angle (potentiometer) and light intensity (phototransistor).

- The RED LED associated with the temperature sensor must be turned ON when the temperature reading is greater than 26 ºC (to be eventually redefined at the laboratory).
- The GREEN LED controlled by the potentiometer must continuously blink at a rate between 0.2 and 2 seconds, depending on the rotation applied to the sensor.
- The BLUE LED associated with the phototransistor must continuously and linearly change its own light intensity based on the light intensity sensed in the surrounding environment from darkness, when the LED should be fully ON, to brightness, where the LED should be OFF. Darkness stands for when the sensor is fully covered and brightness for when the sensor is under a ceiling

light (not under a smartphone flashlight) – see reference 5 about Pulse Width Modulation.

In the first part of the work the sensors will be connected to an Arduino controller and the actuators will be connected to another Arduino controller. A diagram of the circuit to be assembled is provided in the figure below.

References:

1. https://www.arduino.cc/en/Reference/digitalWrite
2. https://www.arduino.cc/en/Reference/AnalogRead
3. https://www.arduino.cc/en/Reference/Serial
4. https://www.arduino.cc/en/Tutorial/Calibration
5. https://www.arduino.cc/en/Tutorial/PWM
6. https://www.arduino.cc/en/Reference/Delay (or https://www.arduino.cc/reference/en/language/functions/time/millis/)

Recommendations:

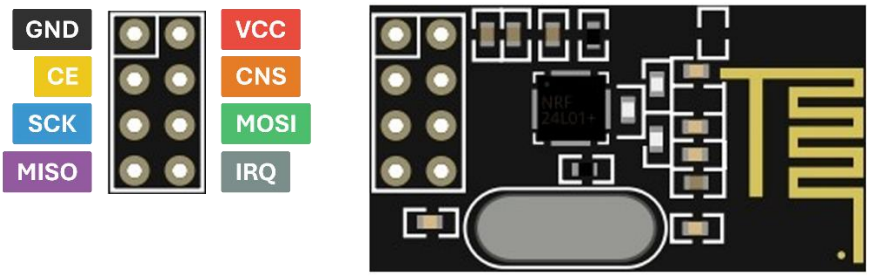
For a safe working environment and to prevent any damage to the hardware, please adhere to the following recommendations. As you progress with your work, check off each box to ensure all safety measures are followed.

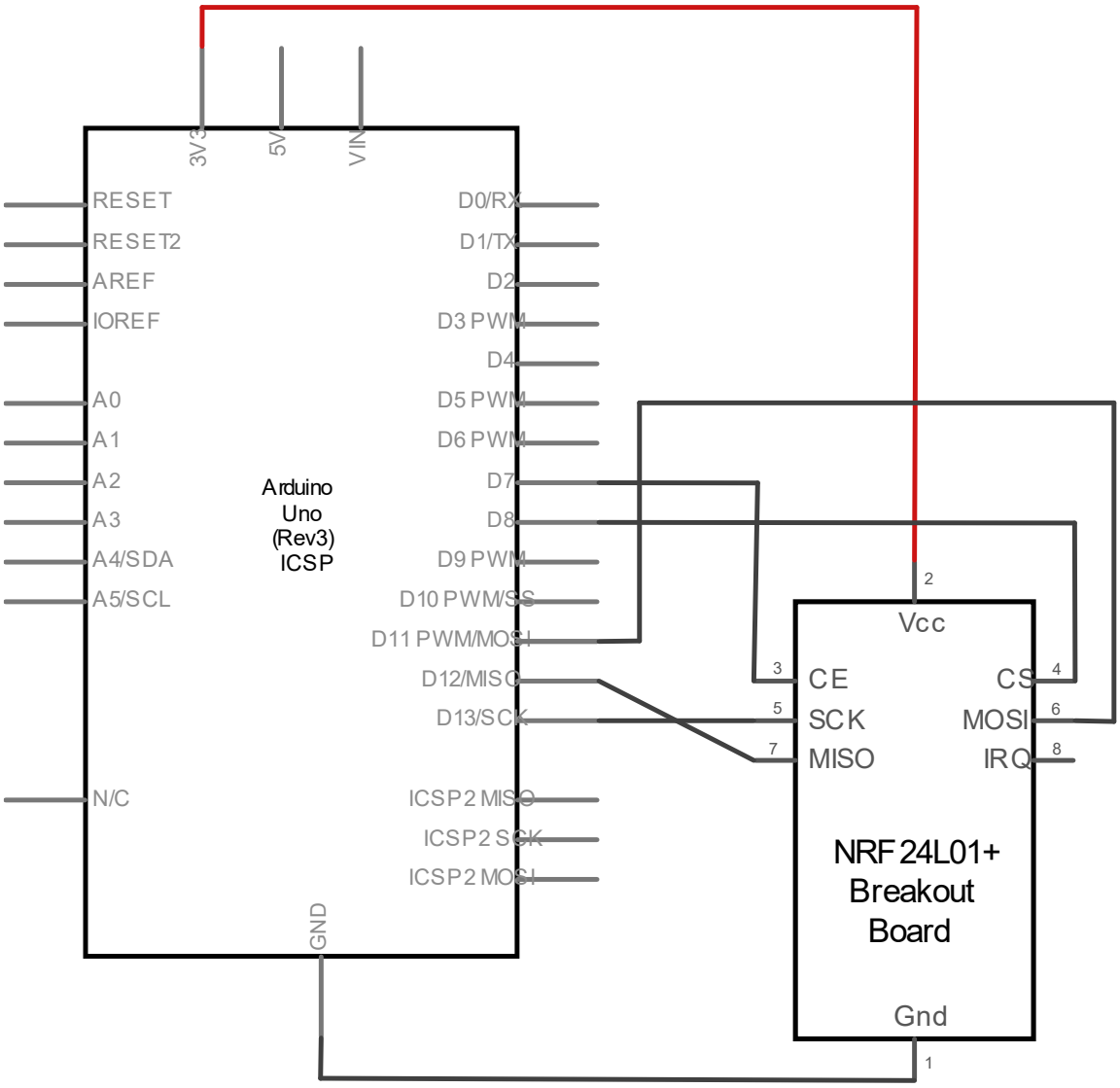
Always ensure that the circuit is disconnected from the power source (either the power supply or the PC) when you are working on it.	
Before connecting the circuit to the power source and turning it on, please consult with the teacher or the person in charge of the laboratory.	
Verify that all components of the circuit (such as resistors, capacitors) are properly connected to prevent short circuits or hardware damage.	

Message Transmission:

To put the system to work it is necessary to implement the communication link connecting two controllers. For this we will use a custom library built on top of the radio frequency modules given on the Arduino Starter Kit (nRF24L01). These modules will allow communication between every Arduino on your classroom or at home.

The nRF24L01 modules need to be connected to your Arduino UNO boards in the following way:





On campus your two controllers will need to be connected to the local network, more precisely, need to be connected to the controller pre-installed in your laboratory classroom. The code below is a template of all the required libraries and functions that need to be initialised to allow a correct behaviour of your devices:

defaultAtCampus.ino

```ino
#include <SPI.h>
#include <RF24.h>
#include <RF24Network.h>
#include "CampusStudentNode.h"

uint16_t sensorNode = 01; // change to your classroom sensorNode
char this_node_name[NAME_LENGTH] = "G01I01"; // change to your group num
char other_node_name[NAME_LENGTH] = "G01I02"; // change to your group num
int channel = 90;

CampusStudentNode studentNode(sensorNode, name, channel);

void setup()
{
  Serial.begin(115200);
  while (!Serial)
  {
    // some boards need this because of native USB capability
  }
  studentNode.init();
  // put your setup code here, to run once
}

void loop()
{
  studentNode.performEssentialOperations();
  // put your main code here, to run repeatedly
}
```

At home you can continue your work but in a simplified manner. Since you are outside the campus you can not reach the local network; you need to create a network of you own. In this case you should use the template below for your work.

Keep in mind that the library you will be using is now *HomeStudentNode*, instead of *CampusStudentNode*. Be aware that the initialisation of the HomeStudentNode do not receive the *SensorNode* (installed on campus) but your node ID (*this_node*).

defaultAtHome.ino

```
#include <SPI.h>
#include <RF24.h>
#include <RF24Network.h>
#include "HomeStudentNode.h"

uint16_t this_node = 00;
uint16_t other_node = 01;
char name[NAME_LENGTH] = "G01I01";
int channel = 90;

HomeStudentNode studentNode(this_node, name, channel);

void setup()
{
  Serial.begin(115200);
  while (!Serial)
  {
    // some boards need this because of native USB capability
  }
  studentNode.init();
  // put your setup code here, to run once
}

void loop()
{
  // put your main code here, to run repeatedly
}
```

To send messages to other controllers you can run two different functions, sendMessage() or sendPayload(). These two functions accept a name/ID, a type of message and a value of any data type.

```
studentNode.sendMessage(other_node_name, 'T', temperature);

studentNode.sendPayload(02, 'T', temperature);
```

The sendPayload() function only works if you know the node ID you want to send the message to. This function will only work at home environment since you specify the node IDs of the controllers. At campus is better to use the sendMessage() function

since you typically only know the names of the nodes. At home is impossible to use the sendMessage() function.

To receive messages from other controllers you can run the receiveSimpleMessage() function. It accepts any kind of data types and returns a Message with a specific format.

```
struct Message { char type = '\0'; T content; };
```

```
Message<int> msg = studentNode.receiveSimpleMessage<int>();
```

To call the Message arguments just type `msg.type` and `msg.content`.

Mapping analog measurements:

Usually, the digital readings retrieved from the sensors do not correspond directly to the value of the physical quantity, but rather to values between 0 and a maximum binary value (such as, for a 10 bits word, 1023 = $2^{10} - 1$). Therefore, some mapping may be required.

When the value is relative just to an offset a simple mapping is adequate, but in general a more complex scale conversion will be needed. The `map` function, available in the Arduino IDE, simplifies these types of conversions. For example, when rotating a servo motor with input from a potentiometer we know that when the value of the potentiometer is 0 then the servo angle must be 0º as well. Logically, if the value of the potentiometer is 1023 (its maximum reading) then the servo angle must be 180º (its maximum position):

```
map (value, 0, 1023, 0, 180)
```

Besides this, some sensors require a linearisation of its transfer function (physical quantity → electrical quantity, such as voltage). For example, to convert the reading from a circuit with a temperature sensor (in our case a temperature dependent resistor) to the real temperature, several transformations may be required:

- to linearise the transfer function of the sensing device $R_T = f(T)$, and
- to linearise the transfer function of the circuit in which the sensor is included.

For the device and configuration to be used (based on a TMP35 component) check the function:

```
T = ((sensor value / 1024.0) * 5.0) – 0.5) * 100;
```

Programming with analog sensors:

To access an external analog sensor, you must attach it to an Arduino analog pin. The software allocation of a sensor to an analog pin is done using the following code:

```
int const tempSensor = A0;
```

where `A0` is the physical pin where the sensor is attached to.

To read the value assigned to a specific pin use the Arduino function `analogRead(PIN)`, as follows:

```
int temperatureValue = analogRead(tempSensor);
```

Debug:

In order to control some variables and *debug* your program it is possible to print them to the Serial Monitor available in the Arduino IDE. This kind of tool is useful, for example, to keep track of the temperature variation or to help the calibration of the system. To use this feature, first start a serial communication with the PC:

```
void setup() {…; Serial.begin(9600); …;}
```

Then, just `Serial.print` your variables and/or strings:

```
Serial.println(temperatureValue);
```

Program the application:

Add your program listing (adequately structured and commented). Organise your program in separate code blocks (basic blocks), one for each function to be implemented (e.g.: read temperature sensor, control temperature alarm), even if you feel such a program overstructured. In the second part of the work, you will have to quickly redistribute several functions across different Arduino controllers.

Avoid, or use very carefully, the delay function provided in the Arduino IDE since it just stops the execution flow during the specified time interval.

Results:

1. Describe the structure of the programs developed for both controllers.

    a. Describe the design pattern (such as tasks/modules/basic blocks in the round-robin loop).

b. Describe the implementation issues (such as implementation of tasks) you consider relevant for the overall app.

2. The code of the 1st controller (master).

3. The code of the 2nd controller (slave).

4. For each of the three pair sensor-actuator describe:

a. the mapping process implemented.

b. the calibration setup.

c. the process, or technique, used to modulate the behaviour of the actuator.

d. the setup prepared to demonstrate the functionality of the system.

5. What are the timing constraints of the system.

6. Evaluate the performance of the ESB in your system (data rate, latency).

7. Mount the light sensor and the corresponding LED face to face so that the sensor detects the light emitted by its LED. To reduce the influence of other light sources you can place the sensor and the LED on both extremes of a paper tube, face to face. Do you see any changes in the behaviour of the LED? If so, explain them and if not explain why.

8. To demonstrate the modularity of your programs setup and run the following configuration (to be specified by the teacher at the demo): the original master is now the slave, i.e., it now controls the LEDs; the original slave is now the master, i.e., it now controls the sensors.