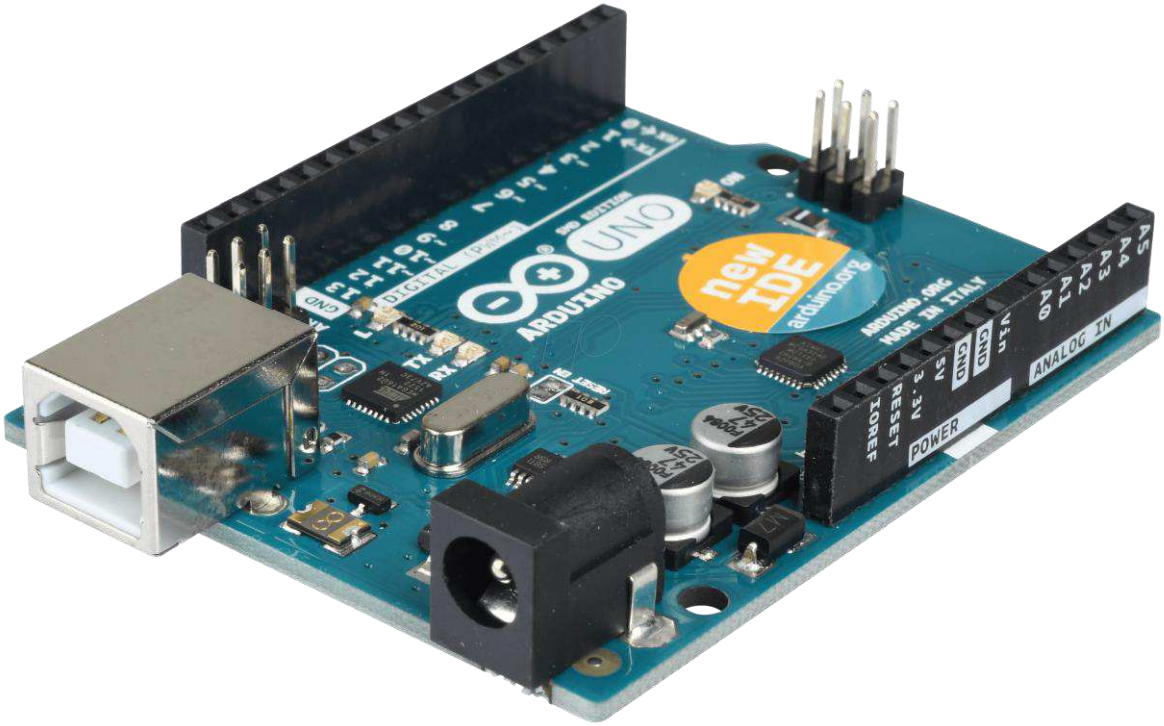
Team 1 :
BENKHOUDA Kamel – BERGEN Frédéric – GOUET Nathan

ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

Embedded Systems : Architecture and Programming

Lab Session 1 & 2 – Arduino and Network: Reminder



3 Global Questions [6pts]

3.1 Arduino IDE [1.5pts]

1) The Arduino language is merely a set of C/C++ functions that can be called from your code. All standard C and C++ constructs supported by avr-g++ should work in Arduino.

2) Your sketch undergoes minor changes (e.g. automatic generation of function prototypes) and then is passed directly to a C/C++ compiler, avr-gcc or avr-g++, which turns the human readable code into machine readable instructions (or object files).

3) The source code license is the " GNU GENERAL PUBLIC LICENSE Version 2, June 1991". This license provides the user to share and change free software to make sure the software is free for all its user. That is to say, the user can distribute copies of free software, receive source code or can get it if he want it, change the software and do new free programs, knowing that he has the right to do so. So, the license provides the user rights and to be effective he is subjected to some restrictions about the distributions of copies of the software, or modifications of it.

3.2 The Arduino Platform [2pts]

1) A digital pin on the Arduino can be configured as either inputs or outputs. Arduino pins default to INPUTS, so they don't need to be explicitly declared as inputs with pinMode() when you're using them as inputs.

Pins configured as OUTPUT with pinMode() are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits.

The analog pins can be used identically to the digital pins, using the aliases A0 (for analog input 0), A1, etc. The analog pins also have pullup resistors, which work identically to pullup resistors on the digital pins. The main function of the analog pins for most Arduino users is to read analog sensors, the analog pins also have all the functionality of general purpose input/output (GPIO) pins.

2) The Arduino Yún is a microcontroller board based on the ATmega32u4 and the Atheros AR9331. The clock speed is 16 MHz, which means that the Arduino can do a single command in 1/16000000 of a second. The clock speed is important because it gives us the execution time of a command.

3) Our Arduino board has characteristics has such :

Microcontroller	ATmega32u4
Operating Voltage	5V
Input Voltage	5V
Digital I/O Pins	20
PWM Channels	7
Analog Input Channels	12
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (of which 4 KB used by bootloader)
SRAM	2.5 KB

4) The Arduino Yún is an Arduino board unlike any other. While programming it is very similar to the Arduino Leonardo and uses the same processor, the Atmel ATmega32U4, it also has an additional processor, an Atheros AR9331, running Linux and the OpenWrt wireless stack.

The Yún has a number of different physical characteristics and connectors than the Leonardo. There is an on-board SD slot, Ethernet jack, and a USB-A Host connector. There is no power barrel connector on the Yún; it can be powered from the micro-USB connector.

So, I will prefer the Yun over the Leonardo if I have the need for an embedded Wifi and Linux system and use it as an ethernet interface.

3.3 Serial Communication [1.5pts]

1) The baud rate represents the number of signal or symbol changes that occur per second. It is a measure of transmission speed. Different baud rates between two machines communicating can lead to bits being “skipped” by the slowest one when receiving from the other.

2) If I transfer data at a speed of 115200 bits per second and I send a message of 48 bytes, i twill take 0.42 ms.

3) If I send a message from the Arduino to the computer using the serial connection, I can't use digital inputs/output because the serial reads and writes on USB and digital pins 0 (RX) and 1 (TX).

3.4 ADC – Analog-to-Digital Converter [1pts]

1) The purpose of an ADC, Analog to Digital Converter is to convert an analog signal into a digital signal. The conversion works in 4 steps : anti-aliasing, sampling, quantizing, digital coding.

2) The characteristics of Arduino's ADC are that the ADC has a 10 bit resolution. So it can convert to 1024 different values, the analogRead() function returns an integer from 0 to 1023. Moreover, its maximum reading rate is about 10 kHz. We can find these information on the Arduino reference and on the official forum.

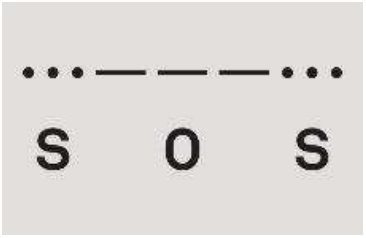
4 Practical Questions [14pts]

4.1 Digital Output [1.5pts]

1) We have to create a code that : Blink a LED as 1s ON and 0,5s OFF

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT); //configures the pin "13" to be an output.
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(500);                        // wait for 0,5 second
}
```

2) Now, we must create a code that blink a led to display the "SOS" message using the Morse code. By searching it on the internet, we found that the Morse code for a "SOS" message is :



So, we must create a code that will blink a led as ON and OFF successively 3 times and during a short period of time between each change of state, after that the code must blink the led as ON and OFF again successively 3 times but during a longer period of time for each ON, and finally again 3 times during a short period of time between each change of state.

That can be illustrated as a code like the one below :

```
// the loop function runs over and over again forever
void loop() {
  int i;
  for(i=0;i<3;i++)
  {
    digitalWrite(13, HIGH);
    delay(500);
    digitalWrite(13, LOW);
    delay(500);
  }
  for(i=0;i<3;i++)
  {
    digitalWrite(13, HIGH);
    delay(2000);
    digitalWrite(13, LOW);
    delay(500);
  }
  for(i=0;i<3;i++)
  {
    digitalWrite(13, HIGH);
    delay(500);
    digitalWrite(13, LOW);
    delay(500);
  }
  delay(2000);
}
```

//during 0.5 seconds the LED is ON then during 0.5 it's OFF, successively 3 times. We have 3 "points".

//during 2 seconds the LED is ON corresponding to a "line" then it's OFF during 0.5 creating a "point", successively 3 times.

//during 0.5 seconds the LED is ON then during 0.5 it's OFF, successively 3 times. We have 3 "points".

//delay of 2 sec between each SOS

4.2 Serial communication [0.5pt]

1. We must create a program which will display "Hello Arduino World !" every 2 seconds.

```
// the setup function runs once when you press reset or power the board
void setup() {
  //Serial communication to be establish at 9600 bail raud
  Serial.begin(9600);

  }

// the loop function runs over and over again forever
void loop() {
  Serial.println("Hello Arduino world!");
  delay(2000);
}
```

//Set speed of exchange

//Write on the serial communication

//In the loop, it will write every 2 sec

To open the window where we can see the message, we have to go to tools then in the Serial Monitor.

4.3 Digital Input [2pts]

1. Now, we must create a code that :

Without interruptions : Count the number of times the button is pressed and released (if you press then release the button, increment a counter). You will display this number on your computer using the serial connection each time it changes. (And only when it changes!)

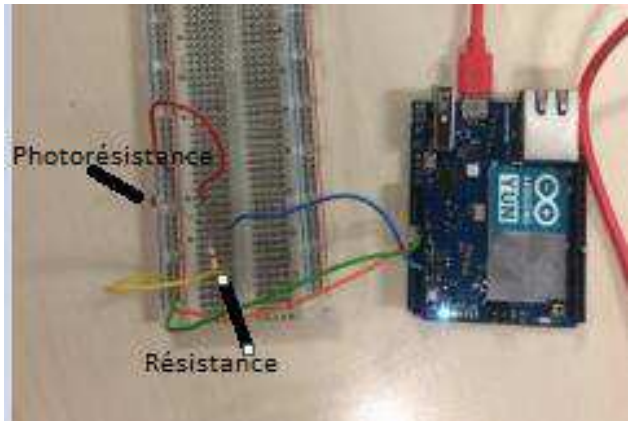
```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status
int lastbuttonState = 0;
int compteur = 0;

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);
  //bool high = false;
  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if(buttonState != lastbuttonState && buttonState == LOW)
  {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
    compteur++;
    Serial.println(compteur);
    }
  else
  {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
  lastbuttonState = buttonState ;
  }
```

2. Now, we have to do it again using the feature of "interruption". An interruption is the fact of asking our program to handle an event when it occurs while it is not waiting for it (our program "interrupt" itself, leaves its current line of code, will handle the event, then returns to its previous line of code).

```
const int buttonPin = 2;     // the number of the pushbutton pin
int buttonPressed = 0;


volatile int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
  // Attach an interrupt to the ISR vector
  attachInterrupt(digitalPinToInterrupt(buttonPin), compteur, FALLING);
  Serial.begin(9600);
}

void loop() {

}

void compteur() {
  buttonState = digitalRead(buttonPin);

  if(buttonState == LOW){
  buttonPressed++;
  Serial.print("The button was pressed and released: ");
  Serial.println(counter,DEC);
  Serial.print(" times");
  }
}
```

4.4 ADC [4pts]

1. This time, we must convert the value obtained using a LDR and display it every 1 second on our computer using the serial connection. After, we see the result in the Serial Monitoring tool.



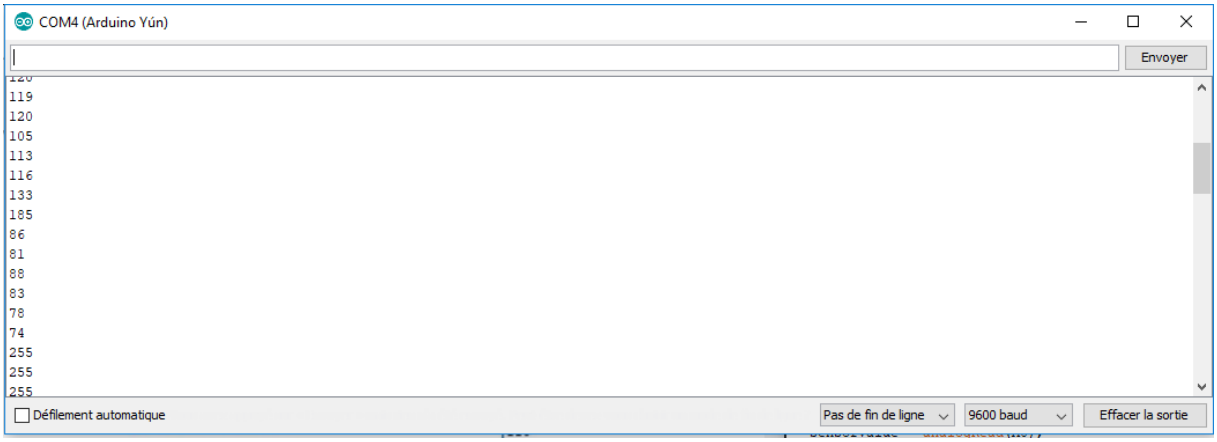
```
int sensorPin = A0;
int sensorValue = 0;

// the setup function runs once when you press reset or power the board
void setup() {
  //Serial communication to be establish at 9600 bail raud
  Serial.begin(9600);

  }

// the loop function runs over and over again forever
void loop() {
  sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(1000);
}
```

//analog input of our Arduino
//initialization of our sensorValue

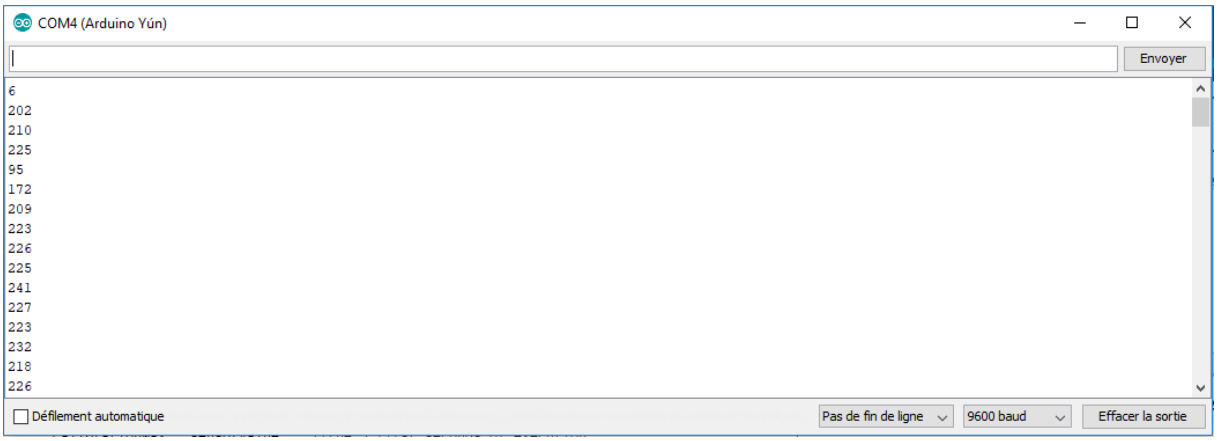
We can see that when we hide the light from the LDR, the sensorValue printed in the window is lower than when the LDR can catch all of it, which is totally normal and proves us that the code is working.

2)

```
int sensorPin = A0;
int sensorValue = 0;

int CalibrationMax = 0;
int CalibrationMin = 0;

// the setup function runs once when you press reset or power the board
void setup() {
  //Serial communication to be establish at 9600 bail raud
  Serial.begin(9600);
  while(millis() < 5000)
  {
    sensorValue = analogRead(A0);
    if(sensorValue > CalibrationMax)  //Start of the calibration step
    {                                 //Takes the lowest and highest values measured during
      CalibrationMax = sensorValue;   //the 5 first seconds of execution
    }
    if(sensorValue < CalibrationMin)
    {
      CalibrationMin = sensorValue;
    }
  }
}
// the loop function runs over and over again forever
void loop() {
  //sensorValue = analogRead(A0);
  Serial.println(map(analogRead(A0), CalibrationMin, CalibrationMax, 0, 255)); //Map the calibration between 0 and 255
  delay(1000);
}
```



3) We use the same code as above, we must only change the LDR by a potentiometer. By using, the calibration code, we have results as such :



4) We are using a 5V Arduino, and connecting the sensor directly into an Analog pin, so we can use these formulas to turn the 10-bit analog reading into a temperature:

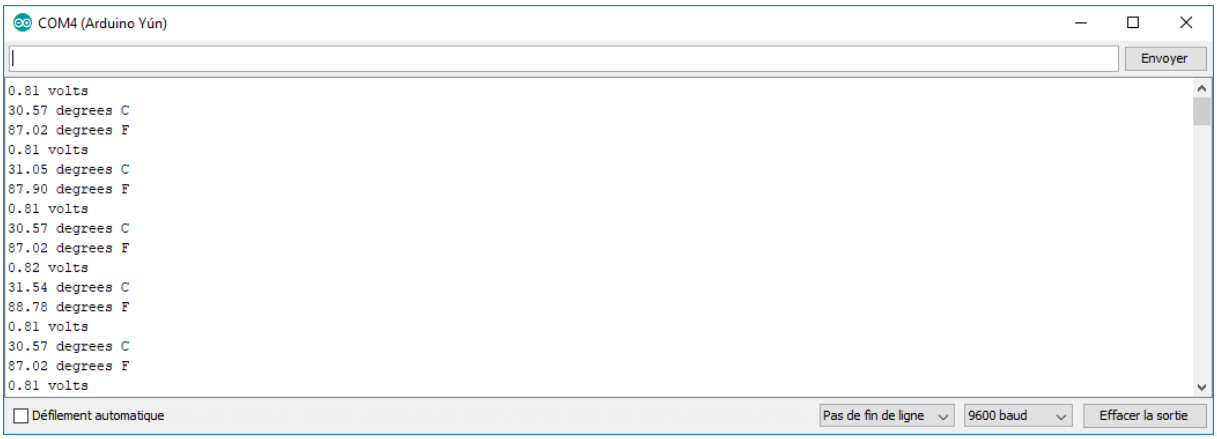
Voltage at pin in milliVolts = (*reading from ADC*) * (5000/1024)
This formula converts the number 0-1023 from the ADC into 0-5000mV (= 5V)

Then, to convert millivolts into temperature, use this formula :

Centigrade temperature = [(analog voltage in mV) - 500] / 10
= [[(reading from ADC) * (5000/1024)] – 500] / 10

```
1. //TMP36 Pin Variables
2. int sensorPin = 0; //the analog pin the TMP36's Vout (sense) pin is
   connected to
3.                        //the resolution is 10 mV / degree centigrade
   with a
4.                        //500 mV offset to allow for negative
   temperatures
5.
6. /*
7.  * setup() - this function runs once when you turn your Arduino on
8.  * We initialize the serial connection with the computer
9.  */
10.void setup()
11.{
```

```
12.   Serial.begin(9600);  //Start the serial connection with the computer
13.                        //to view the result open the serial monitor
14.}
15.
16.void loop()                     // run over and over again
17.{
18. //getting the voltage reading from the temperature sensor
19. int reading = analogRead(sensorPin);
20.
21. // converting that reading to voltage, for 3.3v arduino use 3.3
22. float voltage = reading * 5.0;
23. voltage /= 1024.0;
24.
25. // print out the voltage
26. Serial.print(voltage); Serial.println(" volts");
27.
28. // now print out the temperature
29. float temperatureC = (voltage - 0.5) * 100 ;  //converting from 10 mv
    per degree wit 500 mV offset
30.                                              //to degrees ((voltage -
    500mV) times 100)
31. Serial.print(temperatureC); Serial.println(" degrees C");
32.
33. // now convert to Fahrenheit
34. float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;
35. Serial.print(temperatureF); Serial.println(" degrees F");
36.
37. delay(1000);                                      //waiting a second
38.}
```



4.5 I²C network [3pts]

1) We had the Master Role, so our code was :

Team 1 :
BENKHOUDA Kamel – BERGEN Frédéric – GOUET Nathan

Master Code :

```
#include <Wire.h>

#define SLAVE_ADDRESS 0x60

void setup() {
  Wire.begin(SLAVE_ADDRESS);
  randomSeed(analogRead(3));
  Serial.begin(9600);
}

byte x = 0;

void loop() {

  Serial.print("Sent: ");
  Serial.print("What is the current LDR value and the number of times the button has been pressed and released?");
  Serial.print("\n");

  Wire.beginTransmission(0x60);
  Wire.write("What is the current LDR value and the number of times the button has been pressed and released?");
  Wire.endTransmission();

  delay(7000); // wait 7 seconds

  Serial.println("Requesting Data");
  Wire.requestFrom(SLAVE_ADDRESS, 1);
```

```
  int bytes = Wire.available();
  Serial.print("Slave sent ");
  Serial.print(bytes);
  Serial.print(" of information\n");

  for(int i = 0; i< bytes; i++)
  {
    char c = Wire.read();
    Serial.print("Slave Sent: ");
    Serial.print(c);
    Serial.print("\n");
  }
  delay(500);
}
```

Slaver Code :

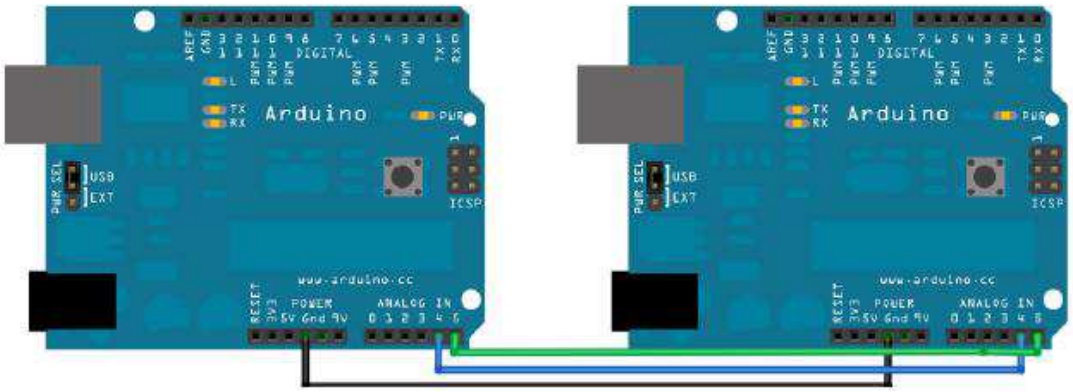
```
#include <Wire.h>

#define SLAVE_ADDRESS 0x60

/* --------- PARAMETERS BUTTON COUNTER ---------- */

// this constant won't change:
const int  buttonPin = 7;    // the pin that the pushbutton is attached to

// Variables will change:
int buttonPushCounter = 0;   // counter for the number of button presses
int buttonState = 0;         // current state of the button
int lastButtonState = 0;     // previous state of the button

/* --------- PARAMETERS TEMPERATURE ---------- */

// These constants won't change:
const int sensorPin = A0;    // pin that the sensor is attached to

// variables:
int sensorValue = 0;         // the sensor value
int sensorMin = 1023;        // minimum sensor value
int sensorMax = 0;           // maximum sensor value


void setup() {
  Wire.begin(SLAVE_ADDRESS);
  randomSeed(analogRead(3));
```

```
  Serial.begin(9600);

  // initialize the button pin as a input:
  pinMode(buttonPin, INPUT);

/* --------- CALIBRATION --------- */

  // calibrate during the first five seconds
  while (millis() < 5000) {
    sensorValue = analogRead(sensorPin);

    // record the maximum sensor value
    if (sensorValue > sensorMax) {
      sensorMax = sensorValue;
    }

    // record the minimum sensor value
    if (sensorValue < sensorMin) {
      sensorMin = sensorValue;
    }
  }
}

byte x = 0;

void loop() {
```

```
/* --------- START BUTTON COUNTER ---------- */

  // read the pushbutton input pin:
  buttonState = digitalRead(buttonPin);

  // compare the buttonState to its previous state
  if (buttonState != lastButtonState) { // it means that the button was pressed
    // if the state has changed, increment the counter
    if (buttonState == LOW) {
      // if the current state is HIGH then the button went from off to on:
      buttonPushCounter++;
     // Serial.print("Number of button pushes and releases: ");
     // Serial.println(buttonPushCounter);
    }
    // Delay a little bit to avoid bouncing
    delay(50);
  }
  // save the current state as the last state, for next time through the loop
  lastButtonState = buttonState;

/* --------- START TEMPERATURE SENSOR ---------- */

// read the sensor:
  sensorValue = analogRead(sensorPin);


  // apply the calibration to the sensor reading
  sensorValue = map(sensorValue, sensorMin, sensorMax, 0, 255);

  // in case the sensor value is outside the range seen during calibration
  sensorValue = constrain(sensorValue, 0, 255);

/* ------- CALL FUNCTION TO RECEIVE ------- */
  // call function to receive potential requests
  Wire.onReceive(receiveEvent);

  Serial.println(buttonPushCounter,DEC);
  Serial.println(sensorValue,DEC);
  delay(1000);

}

void receiveEvent(int bytes)
{
  if(Wire.available() != 0)
  {
    for(int i = 0; i< bytes; i++)
    {
      x = Wire.read();
      Serial.print("Received: ");
      Serial.print(x, HEX);
      Serial.print("\n");
```

```
    }
    // call function to write information
    Wire.onRequest(sendInformation);
  }
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void sendInformation()
{
  Serial.print("Sending: ");
  Serial.print(buttonPushCounter, DEC);
  Wire.write(buttonPushCounter);
  Serial.print(" & ");
  Serial.print(sensorValue, DEC);
  Wire.write(sensorValue);

  Serial.print("\n");

  //Wire.write(buttonPushCounter);
  //Wire.write(sensorValue);
}
```

2) We have connected our two grounds to see the results according this picture :



4.6 Wifi network [3pts]

1)

```
/*
 Yún HTTP Client

 This example for the YunShield/Yún shows how create a basic
 HTTP client that connects to the internet and downloads
 content. In this case, you'll connect to the Arduino
 website and download a version of the logo as ASCII text.
 created by Tom igoe
 May 2013
This example code is in the public domain.
http://www.arduino.cc/en/Tutorial/HttpClient

 */
```

```
#include <Bridge.h>
#include <HttpClient.h>

void setup() {
 // Bridge takes about two seconds to start up
 // it can be helpful to use the on-board LED
 // as an indicator for when it has initialized
 pinMode(13, OUTPUT);
 digitalWrite(13, LOW);
 Bridge.begin();
 digitalWrite(13, HIGH);

 SerialUSB.begin(9600);

 while (!SerialUSB); // wait for a serial connection
}

void loop() {
 // Initialize the client library
 HttpClient client;

 // Make a HTTP request:
 client.get("http://www.arduino.cc/asciilogo.txt");

 // if there are incoming bytes available
 // from the server, read them and print them:
 while (client.available()) {
   char c = client.read();
   SerialUSB.print(c);
 }
 SerialUSB.flush();

 delay(5000);
}
```

2)