



INTERNET DES OBJETS

PROJET - DEVELOPPEMENT D'UNE STATION METEO CONNECTEE

Samuel Mertenat
Internet et Communication - Classe T2f
Kit ARDUINO #134

TABLE DES MATIERES

1 Historique du document.....	3
2 Introduction.....	4
3 Buts et Objectifs du Projet	4
4 Analyse	4
4.1 Arduino UNO	4
4.2 Arduino IDE.....	5
4.3 Fonctionnement du breadboard	6
4.4 Langage « Arduino »	6
4.5 Courant électrique.....	10
4.6 Capteurs	12
4.7 La mémoire.....	14
4.8 Application Controller Interface (ACI).....	15
5 Conception.....	20
5.1 Blink.....	20
5.2 Logging.....	20
5.3 Mémoire libre	21
5.4 Schéma de connexion des modules	21
5.5 Utilisation des capteurs	22
5.6 Watchdog Timer	23
5.7 Bluetooth LE – nRF8001	23
6 Réalisation	24
6.1 Blink.....	24
6.2 Logging.....	24
6.3 Mémoire libre	26
6.4 Utilisation des capteurs	26
6.5 Bluetooth LE – nRF8001	27
7 Tests et validation	29
7.1 TP01.....	29
7.2 TP02.....	32
7.3 TP03	35
8 Problèmes rencontrés & Solutions	37
9 Acquis	37
10 Perspectives	38
11 Conclusion	38
12 Annexe	38
13 Références	39

INTERNET DES OBJETS

PROJET - DEVELOPPEMENT D'UNE STATION METEO CONNECTEE

1 HISTORIQUE DU DOCUMENT

13/03/2015 : Rendu du TP01

- Installation de l'IDE et création d'un premier projet : « IoT »
- Développement d'un programme permettant de faire clignoter une LED et faisant appel au « Serial » afin d'afficher un compteur sur le terminal
- Développement d'un module de « tracing », permettant d'afficher des informations relatives à des erreurs, à du débogage ou simplement des informations
- Analyse du fonctionnement de la mémoire et création d'une méthode permettant d'afficher la quantité de mémoire Ram disponible

25/03/2015 : Rendu du TP02

- Prise en compte des corrections à réaliser suite à la correction du TP01 (4.5.4, 4.5 et 0)
- Analyse du fonctionnement d'un breadboard (4.3), d'un diviseur de tension (4.5.6), d'une résistance « pullup » (4.5.5), d'un capteur DHT11 (4.6.1) et d'une photorésistance (4.6.2)
- Réalisation du schéma complet des connexions des différents modules à l'Arduino Uno (logiciel « Fritzing » ; 5.4)
- Conceptualisation de l'utilisation des capteurs (5.5) et du Watchdog Timer (5.6)
- Développement de la partie relative aux capteurs (6.4) et validation du fonctionnement du programme (7.2)

17/04/2015 : Rendu du TP03

- Prise en compte des corrections à effectuer (4.6.2)
- Analyse du fonctionnement de l'ACI (4.8)
- Conception de l'utilisation de la puce nRF8001 (5.7)
- Réalisation de la partie relative à l'utilisation du nRF8001 (6.5)
- Tests et validation (7.3)

2 INTRODUCTION

Dans le cadre de ce projet, nous allons essayer de mettre en place une station météo connectée, de laquelle nous pourrons consulter les données depuis un Smartphone, via le Bluetooth Low Energy. La station météo sera basée sur un Arduino Uno, auquel nous associerons par la suite, capteurs et actuateurs, tels qu'un capteur de température et d'humidité ou une photorésistance. Le projet se déroulera sur plusieurs sessions de travaux pratiques durant lesquelles nous implémenterons, au fur et à mesure, les différentes fonctionnalités et concepts vus en cours.

3 BUTS ET OBJECTIFS DU PROJET

Ce projet a pour but de mettre en pratique la matière étudiée durant le cours « Internet des objets » en développant, petit à petit, une station météo.

Plus précisément :

- Mise en œuvre et utilisation d'une plateforme de prototypage Arduino
- Développement d'une station météo connectée
- Assimilation d'éléments particuliers et ciblés
 - Programmation en C/C++
 - Capteurs et actuateurs simples
 - Communication Bluetooth Low Energy

4 ANALYSE

4.1 ARDUINO UNO

L'Arduino UNO est une plateforme de développement très bon marché pour débuter à bricoler avec de l'électronique ou de l'automatisation. Son IDE est multiplateformes et l'Arduino ne nécessite, au minimum, que d'être branché à un port USB pour fonctionner.

Caractéristiques :

• Micro contrôleur :	ATmega328
• Tension d'alimentation interne :	5V
• Tension d'alimentation (recommandée) :	7 à 12V, limites : 6 à 20 V
• Entrées / sorties numériques :	14 dont 6 sorties PWM
• Entrées analogiques :	6
• Courant max par broches E / S :	40 mA
• Courant max sur sortie 3,3V :	50mA
• Mémoire Flash :	32 KB (bootloader 0.5 KB)
• Mémoire SRAM :	2 KB
• Mémoire EEPROM :	1 KB
• Fréquence d'horloge :	16 MHz
• Dimensions :	68.6mm x 53.3mm
• IDE de développement :	Arduino

L'arduino Uno est donc un bon compris pour réaliser notre station météo. En effet, la plateforme est très répandue et de nombreux capteurs et actuateurs sont disponibles sur le marché. De plus, il ne consomme que peu d'énergie, ce qui en fait un bon candidat pour un système énergétiquement autonome. Cependant, la mémoire embarquée étant assez faible, il faudra être attentif lors de la partie conception et réalisation.

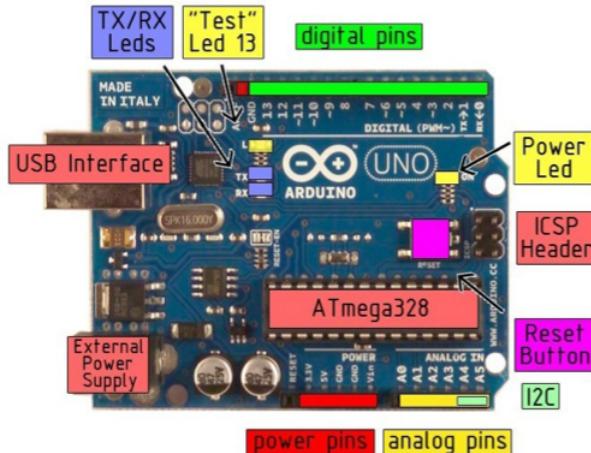


Figure 1: Carte Arduino Uno

4.2 ARDUINO IDE

Le logiciel de développement pour l'Arduino est disponible à l'adresse « <http://arduino.cc/en/Main/Software> » (version 1.6, Mac OS X édition).

L'interface est assez sommaire et permet de prendre rapidement le programme en mains.

Pour créer un nouveau projet, il suffit de cliquer sur « File > New » ; pour y ajouter un fichier : « Flèche vers le bas » (en haut à gauche) puis « New Tab ».

La compilation du code et le téléversement de celui-ci sur l'Arduino se font avec les boutons indiqués sur la capture ci-dessus.

Des librairies tierces peuvent être utilisées et doivent être installées dans le dossier « Arduino/librairies » (par simple copier / coller).

Librairie(s) installée(s) :

- DHT-sensor_library (Adafruit) : <https://github.com/adafruit/DHT-sensor-library>

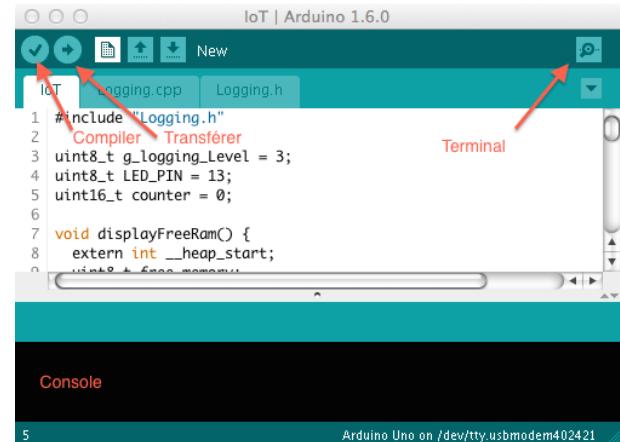


Figure 2: Interface de l'Arduino IDE

4.3 FONCTIONNEMENT DU BREADBOARD

Solderless Breadboards

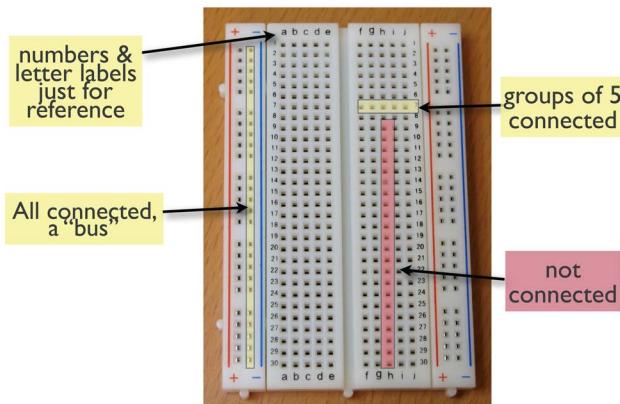


Figure 3: Planche à pain ou breadboard
(<http://yourduino.com/Photos/BreadBoard-1.jpg>)

une séparation au centre.

L'image ci-dessous permet de bien résumer ce principe de fonctionnement. Pour disposer d'une idée plus précise d'un circuit électronique, nous pouvons observer le schéma réalisé au point 5.4 de ce présent rapport, où l'on peut observer un Arduino Uno connecté à une photorésistance, à un capteur de température / humidité et à un module Bluetooth Low Energy.

Une planche à pain ou breadboard, en anglais, est utilisé à des fins de prototypage et permet, sans soudure, de réaliser des circuits électroniques. L'avantage de ce système est d'être totalement réutilisable.

Son fonctionnement est assez simple, les trous des deux premières et les deux dernières colonnes sont reliés en bus, quant aux colonnes du milieu, les trois sont reliés en bus par ligne, avec

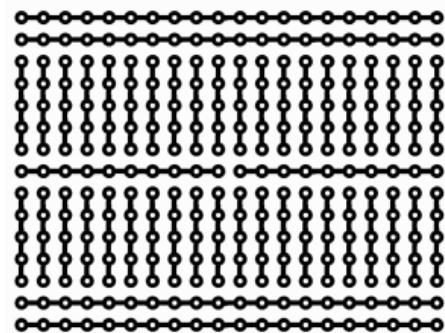


Figure 4: Fonctionnement des liaisons
(<http://upload.wikimedia.org/wikipedia/commons/7/75/Breadboard-144dpi.gif>)

4.4 LANGAGE « ARDUINO »

4.4.1 LE CODE MINIMAL

```
void setup()      //fonction d'initialisation de la carte
{
    //contenu de l'initialisation
}

void loop()       //fonction principale, elle se répète (s'exécute) à l'infini
{
    //contenu de votre programme
}
```

4.4.2 LES VARIABLES

Voilà les types de variables les plus répandus :

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
int	entier	-32 768 à +32 767	16 bits	2 octets
long	entier	-2 147 483 648 à +2 147 483 647	32 bits	4 octets
char	entier	-128 à +127	8 bits	1 octet
float	décimale	-3.4×10^{-38} à $+3.4 \times 10^{-38}$	32 bits	4 octets
double	décimale	-3.4×10^{-38} à $+3.4 \times 10^{-38}$	32 bits	4 octets

Voici le tableau des types non signés, on repère ces types par le mot unsigned (de l'anglais : non-signé) qui les précède :

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
unsigned char	entier non négatif	0 à 255	8 bits	1 octets
unsigned int	entier non négatif	0 à 65 535	16 bits	2 octets
unsigned long	entier non négatif	0 à 4 294 967 295	32 bits	4 octets

Une des particularités du langage Arduino est qu'il accepte un nombre plus important de types de variables.

Type	Quel nombre il stocke ?	Valeurs maximales du nombre stocké	Nombre sur X bits	Nombre d'octets
byte	entier non négatif	0 à 255	8 bits	1 octets
word	entier non négatif	0 à 65535	16 bits	2 octets
boolean	entier non négatif	0 à 1	1 bits	1 octets

Exemples :

```
boolean variable = FALSE; // variable est fausse car elle vaut FALSE, du terme anglais "faux"
boolean variable = TRUE; // variable est vraie car elle vaut TRUE, du terme anglais "vrai"

int variable = 0; // variable est fausse car elle vaut 0
int variable = 1; // variable est vraie car elle vaut 1
int variable = 42; // variable est vraie car sa valeur est différente de 0

int variable = LOW; // variable est à l'état logique bas (= traduction de "low"), donc 0
int variable = HIGH; // variable est à l'état logique haut (= traduction de "high"), donc 1
```

4.4.3 LES TABLEAUX

```
float notes[20]; //on créer un tableau dont le contenu est vide, on sait simplement qu'il contiendra 20 nombres
float note[] = {0,0,0,0 /*, etc.*/ };

float note[] = {};

void setup()
{
    note[0] = 0;
    note[1] = 0;
    note[2] = 0;
    note[3] = 0;
    //...
}
```

4.4.4 LES OPERATIONS LOGIQUES

4.4.4.1 IF ... ELSE IF

```
int prix_voiture = 5500;
if(prix_voiture < 5000)
{
    //la condition est vraie, donc j'achète la voiture
}
else if(prix_voiture == 5500)
{
    //la condition est vraie, donc j'achète la voiture
}
else
{
    //la condition est fausse, donc je n'achète pas la voiture
}
```

4.4.4.2 SWITCH

```
int options_voiture = 0;
switch (options_voiture)
{
    case 0:
        //il n'y a pas d'options dans la voiture
        break;
    default:
        //retente ta chance ;)
        break;
}
```

4.4.4.3 CONDITION TERNAIRE

```
int prix_voiture = 5000;
int achat_voiture = FALSE;
achat_voiture= (prix_voiture == 5000) ? TRUE : FALSE;
```

4.4.4.4 WHILE

```
while(/* condition à tester */)
{
    //les instructions entre ces accolades sont répétées tant que la condition est vraie
}
```

4.4.4.5 DO WHILE

```
do
{
    //les instructions entre ces accolades sont répétées tant que la condition est vrai
}while(/* condition à tester */);
```

4.4.4.6 FOR

```
for(int compteur = 0; compteur < 5; compteur++)
{
    //code à exécuter
}
```

4.4.5 LES FONCTIONS

4.4.5.1 SANS PARAMETRE

```
void fonction()
{
    int var = 24;
    return var; //ne fonctionnera pas car la fonction est de type void
}
```

4.4.5.2 AVEC PARAMETRE(S)

```
int x = 64;
int y = 192;

void loop()
{
    maFonction(x, y);
}

int maFonction(int param1, int param2)
{
    int somme = 0;
    somme = param1 + param2;
    //somme = 64 + 192 = 255

    return somme;
}
```

4.4.6 L'UTILISATION DU « SERIAL »

Le « Serial » permet d'écrire et de recevoir des données via le terminal. Pour ce faire, il faut initialiser la « ligne » avec un « `Serial.begin(9600)` », 9600 étant la vitesse en bits par seconde.

Les données peuvent être ensuite lues et écrites avec les méthodes suivantes (cf. doc) : `print()`, `write()`, `read()`, etc.

```
uint8_t LED_PIN = 13;
uint16_t counter = 0;

// the setup function runs once when you press reset or power the board
void setup() {
    Serial.begin(9600);           // initiates the serial communication
    pinMode(LED_PIN, OUTPUT);     // sets the pin as output
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED_PIN, HIGH); // switches on the led
    Serial.print("The led is switched on; Number of times: ");
    Serial.println(++counter, DEC);
    delay(1000);                // waits for 1000ms -> 1s
    digitalWrite(LED_PIN, LOW);   // switches off the led
    delay(1000);
}
```

Source : <http://arduino.cc/en/reference/serial>

4.4.7 DIVERS

- Récupérer le temps : `millis()`
- Réajuster une valeur selon un intervalle : `map(value, fromLow, fromHigh, toLow, toHigh)`

4.5 COURANT ELECTRIQUE

4.5.1 SENS DU COURANT

Le courant électrique se déplace selon un sens de circulation. Un générateur électrique, par exemple une pile, produit un courant. Et bien ce courant va circuler du pôle positif vers le pôle négatif de la pile, si et seulement si ces deux pôles sont reliés entre eux par un fil métallique ou un autre conducteur. Ceci, c'est le sens conventionnel du courant.

4.5.2 INTENSITE DU COURANT

On mesure la vitesse du courant, appelée intensité, en Ampères avec un Ampèremètre. En général, en électronique de faible puissance, on utilise principalement le milliampère (mA) et le micro-Ampère (μ A), mais jamais bien au-delà.

4.5.3 TENSION

Autant le courant se déplace, ou du moins est un déplacement de charges électriques, autant la tension est quelque chose de statique. Pour bien définir ce qu'est la tension, sachez qu'on la compare à la pression d'un fluide.

4.5.4 LOI D'OHM

$$U = R * I$$

Dans le cas d'une LED, on considère, en général, que l'intensité la traversant doit-être de 20 mA. Si on veut être rigoureux, il faut aller chercher cette valeur dans le datasheet.

On a donc $I = 20 \text{ mA}$.

Ensuite, on prendra pour l'exemple une tension d'alimentation de 5V (en sortie de l'Arduino, par exemple) et une tension aux bornes de la LED de 1,2V en fonctionnement normal. On peut donc calculer la tension qui sera aux bornes de la résistance :

$$U_r = 5 - 1,2 = 3,8 \text{ V}$$

Enfin, on peut calculer la valeur de la résistance à utiliser :

$$\text{Soit : } R = U / I$$

$$R = 3,8 / 0,02$$

$$R = 190 \text{ Ohms}$$

(Corrections TP01 ; ajout d'un schéma¹)

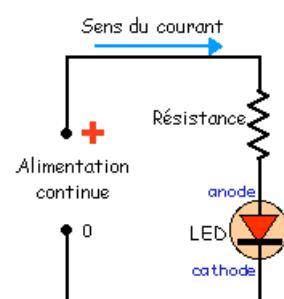


Figure 5: Schéma de la connexion

¹ <http://www.ptitrain.com/electronique/tekno/thumbnails/04b.gif>

4.5.5 RESISTANCE « PULLUP »

Les résistances « pullup » sont fréquemment utilisées lors de l'utilisation de microcontrôleurs ; elles permettent de « corriger » un signal à un niveau logique High ou Low. Par exemple, si une broche est configurée en tant qu'entrée, que rien n'y est connecté et que le programme lit son état, il est difficile de prédire si le signal sera à un niveau High ou Low. Pour palier à ce problème, nous pouvons utiliser des résistances « pullup » ou « pulldown », qui permettent aussi d'utiliser moins de courant. Les résistances « pullup » sont les plus rependues et sont généralement utilisées avec des boutons ou des switches.

Calcul d'une résistance « pullup » :

En définissant arbitrairement la limite du courant à 1 mA lorsque le bouton est pressé, avec une tension V de 5 V, quelle doit-être la valeur de la résistance ?

La loi d'Ohm :

$$U = R * I$$

En appliquant au schéma, on obtient :

$$V_{cc} = (I \text{ à travers } R1) * R1$$

En isolant la résistance, on obtient :

$$R1 = \frac{V_{cc}}{I \text{ à travers } R1} = \frac{5V}{0.001A} = 5k \text{ Ohms}$$

Remarque : il ne faut pas utiliser une résistance trop élevée, ce qui diminuerait la tension, mais ralentirait le temps de réponse de la broche d'entrée pour détecter la variation².

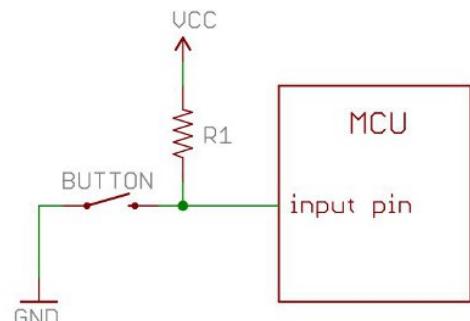


Figure 6: Schéma électrique d'un bouton-poussoir

4.5.6 DIVISEUR DE TENSION

Un diviseur de tension, comme son nom l'indique, permet de réduire une tension en une plus faible. Ce mécanisme est possible en disposant deux résistances en série ; la tension de sortie sera donc une fraction de la tension d'entrée. Les applications sont variées : potentiomètre, photorésistance, etc.

Le circuit se compose donc de deux résistances et d'une tension d'entrée, en voici quelques exemples, sur la figure ci-dessous.

Afin de calculer la tension de sortie, la tension d'entrée et la valeur des

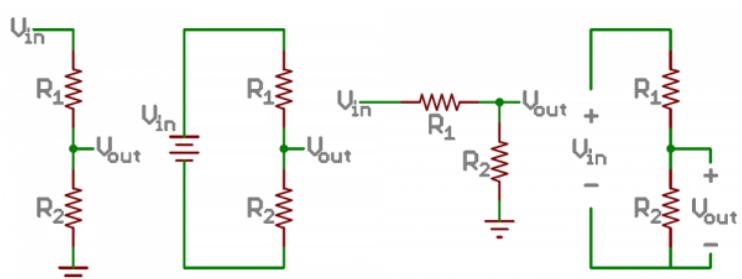


Figure 7 : Exemples de diviseur de tension

² <https://learn.sparkfun.com/tutorials/pull-up-resistors>

deux résistances sont nécessaires. La résistance R1 se trouve toujours à au plus près de la tension d'entrée ; la résistance R2 au plus près de la terre. La fraction de la tension d'entrée peut donc être mesurée entre ces deux résistances.

L'équation se présente ainsi :

$$V_{out} = V_{in} * \frac{R2}{R1 + R2}$$

Cette équation peut être simplifiée dans certaines situations. Si la valeur de la résistance R1 est à égal à la résistance R2, la tension de sortie est égale à la moitié de la tension d'entrée.

$$V_{out} = V_{in} * \frac{R2}{R1 + R2} = V_{in} * \frac{R}{2R} = \frac{V_{in}}{2}$$

Si la valeur de la résistance R2 est beaucoup plus élevée que la résistance R1, la tension de sortie sera proche de la tension d'entrée.

$$V_{out} = V_{in} * \frac{R2}{R1 + R2} \approx V_{in} * \frac{R2}{R2} = V_{in}$$

A l'inverse, si la valeur de la résistance R2 est beaucoup plus faible que la résistance R1, la tension passera majoritairement par la résistance R1.

$$V_{out} = V_{in} * \frac{R2}{R1 + R2} \approx V_{in} * \frac{0}{R1} = 0$$

4.6 CAPTEURS

4.6.1 DHT11

Le capteur DHT11 permet de mesurer la température et l'humidité et est calibré d'usine. Son principal avantage est d'être très bon marché et relativement précis, comme nous pouvons le voir sur la figure ci-dessous.

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 °C	±5%RH	±2°C	1	4 Pin Single Row

Le capteur utilise une connexion « single-wire serial interface » pour communiquer et se synchroniser avec le microprocesseur ; une communication dure en principe, environ 4 ms. Les données sont codées sur 40 bits. Pour plus d'informations quant à l'implémentation du protocole de communication, nous pouvons nous référer à la datasheet du capteur, disponible à l'adresse : <http://www.micropik.com/PDF/dht11.pdf> (pages 5-9).

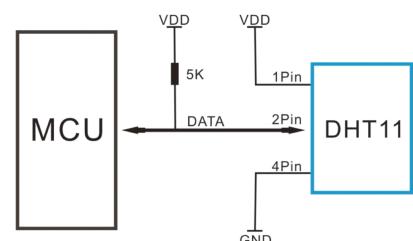


Figure 8: Branchement d'un capteur DHT11

Son branchement est ais  , il ne n  cessite que de trois connexions :

- Pin 1 : Vcc 5V
- Pin 2 : Data → résistance « pullup » d'environ 5k Ohms + pin digitale sur l'Arduino
- Pin 4 : GND

La valeur de la résistance « pullup » peut être facilement calculée à l'aide de la formule présentée au point 4.5.5, en utilisant un courant de 1mA, correspondant à la consommation moyenne du capteur à ~5 V.

$$R1 = \frac{Vcc}{I \text{ à travers } R1} = \frac{5V}{0.001A} = 5k \text{ Ohms}$$

4.6.2 PHOTORESISTANCE

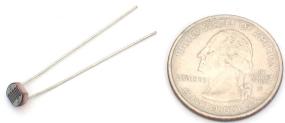


Figure 9: Une photorésistance

Une photorésistance est un composant électronique dont la résistivité varie en fonction de la lumière ambiante, bien que qu'elle ne soit sensible de la même manière à toutes les longueurs d'onde (plus sensible à des couleurs comme le jaune / rouge qu'à des couleurs froides).

Sur le graphe ci-dessous, trouvé sur internet, nous pouvons constater l'évolution de l'illumination en lux, en fonction de la résistance.

Afin d'avoir une représentation un peu plus claire de ce que représente des « lux », voici quelques exemples tirés de notre quotidien :

- Nuit de plein lune : 0.5 lux
- Rue de nuit bien éclairée : 20-70 lux
- Appartement bien éclairé : 200-400 lux
- Extérieur par ciel couvert : 500-2500 lux
- Extérieur en plein soleil : 50000-100000 lux

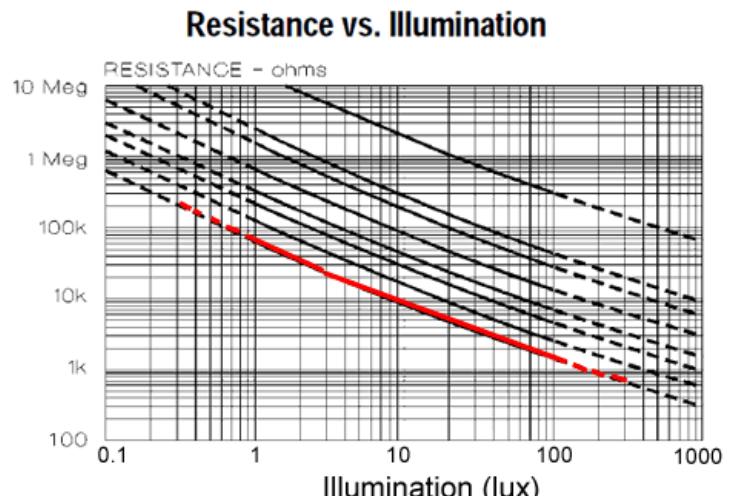
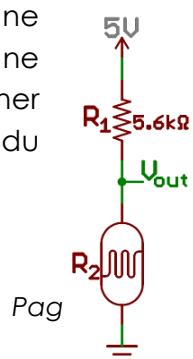


Figure 10: Graphique illustrant l'illumination en fonction de la résistance
(<http://cdn.instructables.com/FNE/0LR9/FVS7L1OF/FNE0LR9FVS7L1OF.LARGE.gif>)

Pour mesurer la résistivité d'un capteur de ce type, il suffit de connecter une broche au 5 V et de relier la seconde à une résistance « pulldown », à une entrée analogique de l'Arduino et à la terre. Nous pouvons ensuite déterminer la résistance de la photorésistance à l'aide de l'Arduino, en fonction du courant mesuré.



$$V_o = V_{cc} \times \frac{R}{R + \text{Photorésistance}}$$

Le courant est donc inversement proportionnel à la résistance de la photorésistance et donc, aussi inversement proportionnel à la luminosité mesurée.

Les deux tableaux ci-dessous illustrent le courant mesuré, basé sur le capteur de photorésistance à 5 V et avec des résistances « pulldown » de 1k et de 10k Ohms.

Ambient light like...	Ambient light (lux)	Photocell resistance (Ω)	LDR + R (Ω)	Current thru LDR + R	Voltage across R
Dim hallway	0.1 lux	600K Ω	610 K Ω	0.008 mA	0.1 V
Moonlit night	1 lux	70 K Ω	80 K Ω	0.07 mA	0.6 V
Dark room	10 lux	10 K Ω	20 K Ω	0.25 mA	2.5 V
Dark overcast day / Bright room	100 lux	1.5 K Ω	11.5 K Ω	0.43 mA	4.3 V
Overcast day	1000 lux	300 Ω	10.03 K Ω	0.5 mA	5V

Figure 11: Lux mesurés avec 5 V et une résistance de 10k Ohms

Ambient light like...	Ambient light (lux)	Photocell resistance (Ω)	LDR + R (Ω)	Current thru LDR+R	Voltage across R
Moonlit night	1 lux	70 K Ω	71 K Ω	0.07 mA	0.1 V
Dark room	10 lux	10 K Ω	11 K Ω	0.45 mA	0.5 V
Dark overcast day / Bright room	100 lux	1.5 K Ω	2.5 K Ω	2 mA	2.0 V
Overcast day	1000 lux	300 Ω	1.3 K Ω	3.8 mA	3.8 V
Full daylight	10,000 lux	100 Ω	1.1 K Ω	4.5 mA	4.5 V

Figure 12: Lux mesurés avec 5 V ett une résistance de 1k Ohms

(Corrections TP03 ; ajout d'un schéma³)

4.7 LA MEMOIRE

L'Arduino ne disposant que de très peu de mémoire, il est important de l'utiliser avec parcimonie. Comme nous pouvons l'observer sur la figure ci-dessous, la mémoire RAM est divisée en 4 parties distinctes: « .data variables », « .bss variables », « heap » et « stack ». La mémoire à notre disposition, pour instancier des variables, débute à la partie « heap » (« __heap_start ») et se termine à la fin de la « stack » (« __brkval »).

³ <http://www.ptittrain.com/electronique/tekno/thumbnails/04b.gif>

(Corrections TP01) Le « sketch » ou programme réalisé avec l'IDE Arduino sera stocké dans la mémoire FLASH de l'Arduino, qui en comporte 32kb, lors du « téléversement ». Quant à la mémoire SRAM, de 2kb, elle permettra de stocker les variables utilisées tout au long du programme ; c'est cette quantité de mémoire qui peut être réduite en faisant varier le niveau de logging.

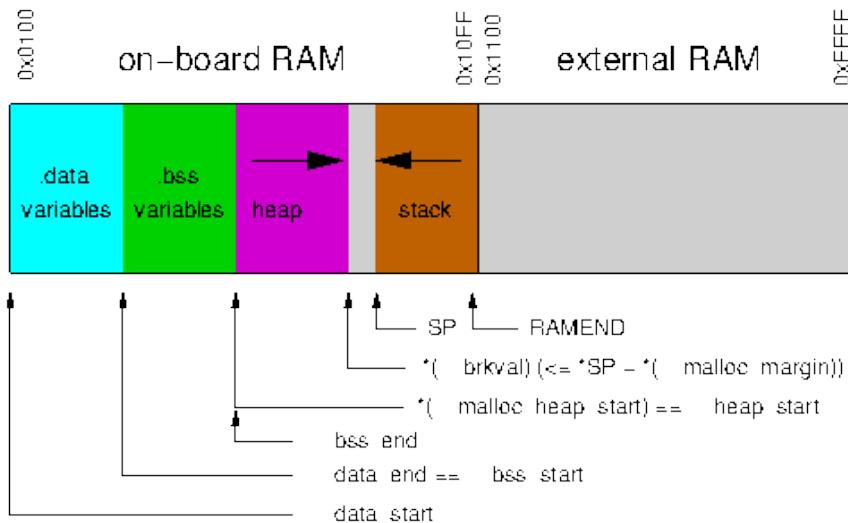


Figure 13: La mémoire

4.8 APPLICATION CONTROLLER INTERFACE (ACI)

Le ACI est une interface série, fonctionnant de manière bidirectionnelle, qui permet de gérer la puce nRF8001. L'ACI se place entre la couche application et les couches inférieures de la pile Bluetooth.

4.8.1 FONCTIONNEMENT

Le trafic d'information généré par l'ACI est bidirectionnel et le contrôle est exercé par le contrôleur de l'application (« application controller »). Ce dernier peut envoyer des commandes ACI au nRF8001 ; le nRF8001, peut quant à lui, envoyer des informations au contrôleur de l'application, de manière indépendante.

Les commandes envoyées par le contrôleur au nRF8001 peuvent être classifiées par :

- Commandes système (« System commands ») : commandes utilisées pour la configuration et la gestion du mode opérationnel du nRF8001
- Commandes de données (« Data commands ») : commandes utilisées en état connecté, de façon « point-to-point » afin de transférer ou recevoir des données

Les informations transmises du nRF8001 à l'ACI sont appelées des événements (« events ») ; ils peuvent être liés au système ou aux données.

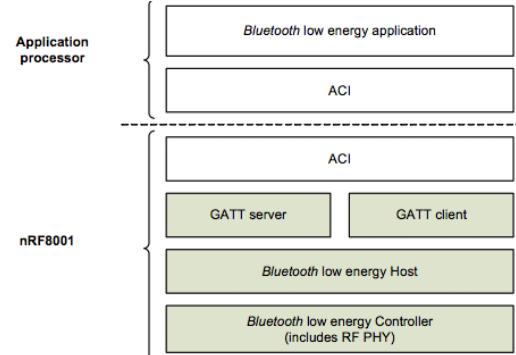


Figure 14: Pile du Bluetooth BLE

Sur la figure ci-dessous, nous pouvons observer 4 scénarios liés à l'utilisation de l'ACI :

1. System command – System event

Le contrôleur envoie une commande système (« System command ») au nRF8001, qui lui répond avec acquittement, sous la forme d'un événement (« System event »).

2. System event

Le nRF8001 envoie un événement (« event ») au contrôleur de l'application, déclenché par une condition prédéfinie.

3. Data command – Data event

Le contrôleur envoie une commande de données (« Data command ») afin de notifier un envoi ou une réception de données au nRF8001. Si la transaction réussit, les données sont envoyées sous la forme d'un événement (« event »).

4. Data event

Le nRF8001 envoie un événement (« event ») au contrôleur, déclenché par un transfert de données ou une condition prédéfinie relative au transfert de données

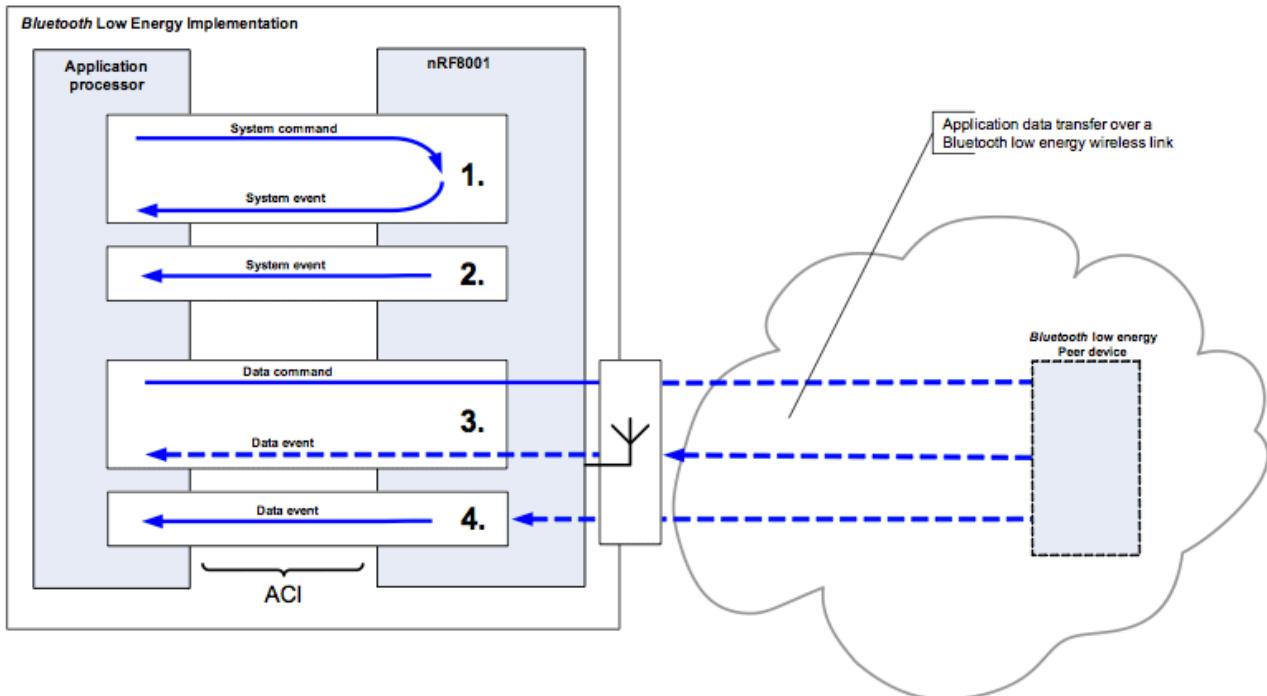


Figure 15: Principe de fonctionnement de l'ACI

4.8.1.1 STRUCTURE DES PAQUETS

Les différentes commandes et événements transitent par le biais de paquets. Chaque paquet est composé d'un entête de 2 octets (« Packet header ») et d'une charge utile (« Payload »), pouvant aller de 0 à 30 octets. L'ordre des données suit le format « Little Endian », soit que l'octet le moins significatif est transféré en premier.

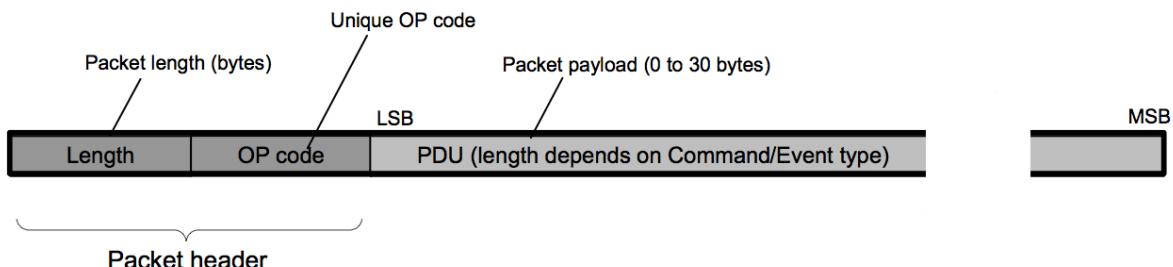


Figure 16: Structure d'un paquet ACI

Structure du paquet :

- Length : contient la taille du paquet, excluant la taille du champ « Length »
- OP code : contient l'identifiant unique de la commande / de l'événement
- PDU : contient la charge utile, dépend du type de paquet ACI à transmettre

L'ACI propose trois types de paquets, que sont les « System commands », les « Data commands » et les « Events ».

4.8.2 SYSTEM COMMANDS

Les « System commands » sont des commandes envoyées par l'ACI au nRF8001 ; elles permettent de contrôler la configuration, le mode d'utilisation et le comportement du nRF8001.

Quelques commandes :

- Test (0x01) : permet d'activer ou désactiver le mode « test » sur le nRF8001
- Echo (0x02) : permet de tester le bon fonctionnement de la couche de transport de l'ACI
- DtmCommand (0x03) : permet d'envoyer une commande « Direct Test » au nRF8001, en passant par l'interface de l'ACI
- Sleep (0x04) : permet d'activer le mode veille sur la puce radio (en veille jusqu'à une commande « Wakeup »)
- Wakeup (0x05) : permet de « réveiller » le nRF8001 après une période en veille

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Length	1	1	Packet length
Command	1	0x05	Wakeup

Figure 17: Exemple du paquet d'une commande ACI « Wakeup »

D'autres commandes existent ; pour plus d'informations, se référer au document « nRF8001_PS_v1.3.pdf⁴ » aux pages 96-132.

4.8.3 DATA COMMANDS

Les « Data commands » sont des commandes envoyées par l'ACI au nRF8001 ; elles sont utilisées lors de transferts de données entre un périphérique et la puce radio. Ces commandes permettent d'initialiser le transfert de données entre la puce radio le périphérique couplé :

- Quand le nRF8001 agit en tant que serveur GATT, il peut :
 - Initialiser le transfert de données, stockées localement, vers le périphérique couplé
 - Recevoir des données du périphérique
- Quant le nRF8001 agit en tant que client GATT, il peut :
 - Envoyer des données vers le périphérique couplé
 - Demander la transmission de données de la part du périphérique
 - Demander la transmission de données de la part du périphérique sous la forme d'une indication (« Handle value indication ») ou d'une notification (« Handle value notification »).

Quelques commandes :

- SetLocalData (0x0D) : permet de transmettre ou de recevoir des données lorsque le nRF8001 est en mode connecté, avec un périphérique
- SendData (0x15) : permet d'envoyer des données à périphérique couplé, par le biais d'un « service transmit pipe »
- SendDataAck (0x16) : permet de confirmer la réception de données d'un périphérique
- RequestData (0x17) : permet de demander des données à un périphérique, par le biais d'un « service receive pipe »
- SendDataNack (0x18) : permet d'envoyer un non-acquittement de réception de données de la part d'un périphérique

Message field/ parameter	Value size (bytes)	Data value	Description
Header			
Length	1	3	
Command	1	0x18	SendDataNack
Content			
PipeNumber	1		On which pipe the data is negatively acknowledged.
ErrorCode	1		Attribute protocol error code to be sent to the peer device.

Figure 18: Exemple du paquet d'une commande ACI « SendDataNack»

Pour plus d'informations, se référer au document « nRF8001_PS_v1.3.pdf » aux pages 133-138.

⁴ https://www.nordicsemi.com/eng/nordic/download_resource/17534/16/61913825

4.8.4 EVENTS

Les « events » sont des messages envoyés par le nRF8001 à destination de l'ACI ; ils peuvent être de simples réponses ou des événements asynchrones, déclenchés par un facteur lié à un certain événement :

- Response events : permet l'acquittement d'une commande
- Asynchronous events : permet l'indication à l'ACI qu'une condition a été remplie.
Par exemple, un « DisconectedEvent » est généré lorsque la connexion radio a été perdue. Ces événements n'ont pas une relation temporelle régulière ou prévisible avec l'ACI.

Quelques évènements :

- DeviceStartedEvent (0x81) : permet d'indiquer un « reset recovery » ou un changement d'état
- EchoEvent (0x82) : permet de retourner une copie de l'« Echo ACI message »
- HardwareErrorEvent (0x83) : permet de retourner des informations liées au débogage d'éventuelles erreurs hardware
- CommandResponseEvent (0x84) : permet de confirmer la réception ou l'exécution d'une commande ACI
- ConnectedEvent (0x85) : permet d'indiquer qu'une connexion a été établie avec un périphérique

D'autres évènements existent ; pour plus d'informations, se référer au document « nRF8001_PS_v1.3.pdf⁵ » aux pages 139-157.

4.8.5 SERVICE PIPES

Le concept de « service pipes » est propre à l'ACI du nRF8001, et non pas au Bluetooth. Il permet de simplifier l'accès à un service caractéristique dans un client / serveur. Un « service pipe » peut être considéré comme un tuyau, où circulent des données de / à un client / serveur.

Les « service pipes » permettent de pointer vers des déclarations spécifiques dans un service ; par exemple, la déclaration caractéristique de la température dans un service d'un thermomètre. La programmation de la configuration des « service pipes » se fait à l'intérieur du nRF8001 et est fixe pour la durée de l'application. Le type et le nombre de tuyaux sont à définir selon l'application et les données y seront transmises lors de l'exécution du programme.

L'installation des « service pipes » définit :

- La direction des transferts de données : transmission / réception
- La localisation du serveur : données stockées sur le nRF8001 ou sur le périphérique
- La nécessité des acquittements
- L'automatisation des acquittements

⁵ https://www.nordicsemi.com/eng/nordic/download_resource/17534/16/61913825

- L'authentification de la connexion
- Broadcast : données peuvent être envoyées via « connectable advertisement packets » ou « un-connectable advertisement packets »

Chaque « service pipe » dispose d'un identifiant unique ; lorsque des données sont envoyées / reçues à / de un serveur, l'identifiant permet à la partie logiciel de faire la correspondance entre le GATT et les fonctionnalités fournies par le « service pipe ».

Sur la figure ci-dessous, nous pouvons observer comment différents « service pipes » peuvent être assignés à deux déclaration de caractéristique d'un service.

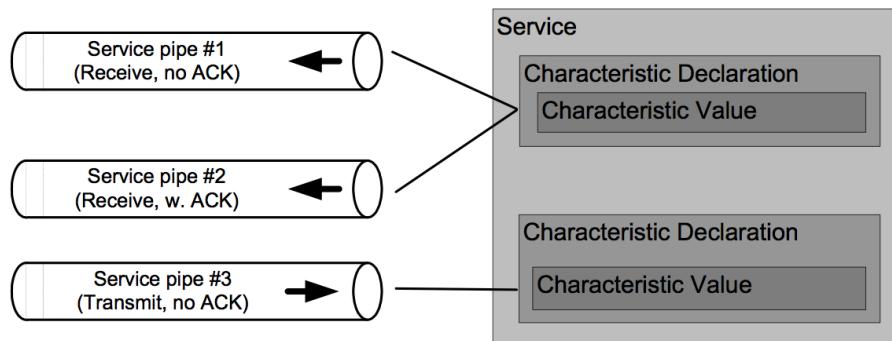


Figure 19: « Service pipes » assignés à un service

Pour plus d'informations, se référer au document « nRF8001_PS_v1.3.pdf » aux pages 59-64.

5 CONCEPTION

5.1 BLINK

A l'étape 5 du premier TP, nous devions nous inspirer du code de l'exemple « Blink » (Files > Examples > 01.Basics > Blink), faisant clignoter une la led 13 de l'Arduino toutes les deux secondes afin d'afficher à chaque allumage, un message sur la console, ainsi que le nombre de récurrence de cet événement. Pour réaliser ceci, il suffit de déclarer une nouvelle variable qui nous servira de compteur (« counter »), que l'on incrémentera à chaque tour de la « loop ». L'affichage se fait ensuite en utilisant le « Serial », en débutant par l'initialiser dans la fonction « setup() » (« Serial.begin(9600) ») et en affichant ensuite un message avec l'instruction « Serial.println() ».

5.2 LOGGING

A l'étape 6 du premier TP, nous devions réaliser un système nous permettant de « logger » notre code, car la plateforme Arduino n'en propose pas par défaut. Le principe consiste à afficher les logs sur la console, selon leur type (ERROR, INFO, DEBUG) et après quel temps d'exécution ils ont eu lieu. D'autre part, le niveau de « logging », qui peut prendre une valeur allant de 1 à 3, permet de spécifier la quantité de logs, et donc d'économiser de la mémoire (1 < 3).

Deux fichiers sont nécessaires à l'implémentation d'un tel système :

- Logging.cpp : Module contenant les méthodes « PrintHeader() » permettant d'afficher le moment où le message de log est affiché (hh-mm-ss), ainsi que le type de log (ERROR, INFO, DEBUG) et la méthode « PrintSerial », permettant d'utiliser une macro à deux paramètres et donc, de formater sur une seule ligne, toutes ces infos.
- Logging.h : Entête du fichier Logging.cpp contenant des instructions pour le préprocesseur afin de définir des macros pour les 3 types de logs. Le niveau de log définit ce que sera ou non compilé ; si le niveau de log est inférieur au niveau de log requis par les macros, celles-ci ne seront donc pas compilées.

5.3 MEMOIRE LIBRE

A l'étape 7 du premier TP, nous devions écrire une fonction permettant de calculer la quantité de mémoire libre. Bien que cette donnée soit affichée dans la console, il est intéressant de comprendre comment celle-ci est calculée.

Pour ce faire, il suffit d'instancier une variable (partie de la mémoire « stack »), d'en récupérer l'adresse et d'en soustraire l'adresse de la variable externe « `__heap_start` ».

5.4 SCHEMA DE ONNEXION DES MODULES

Sur le schéma ci-dessous, réalisé avec le logiciel « Fritzing », nous pouvons observer la manière dont les différents modules sont reliés à l'Arduino UNO.

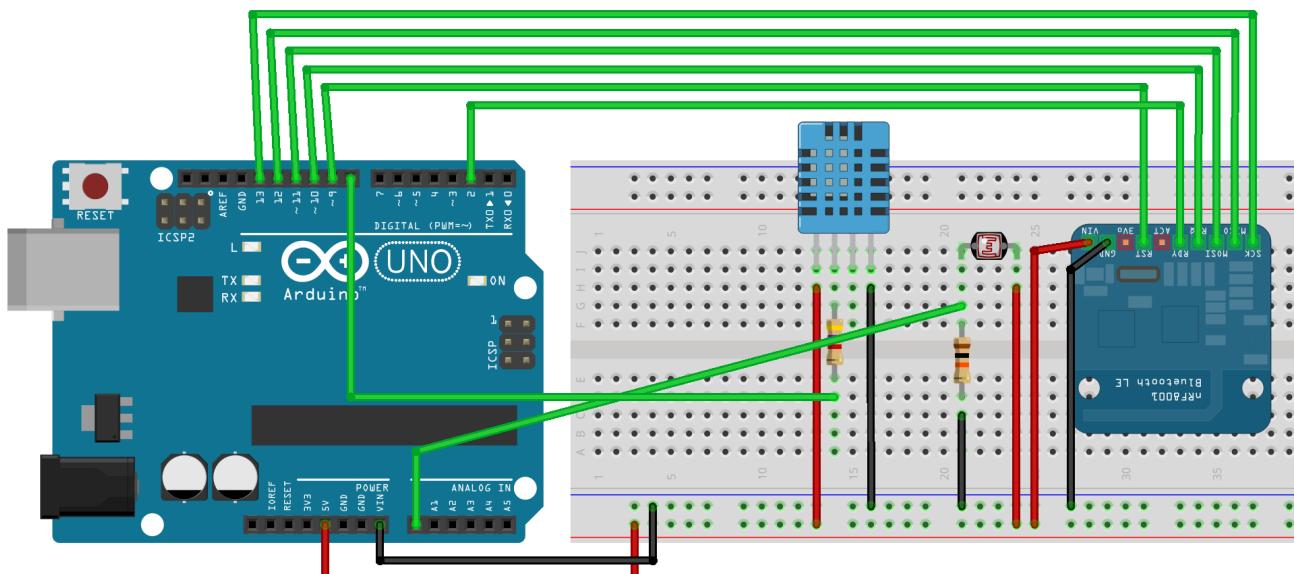


Figure 20: Schéma de connexion des différents modules

DHT11 :

- Pin 1 : 5 V
- Pin 2 : pin 8 (digitale) + résistance de 4.7 Ohms
- Pin 3 : terre

Photorésistance :

- Pin 1 : pin A0 (analogique) + résistance de 10kOhms + terre
- Pin 2 : 5 V

nRF8001⁶ :

- Vin : 5 V
- GND : terre
- SCK : SPI clock, pin 13
- MISO : SPI MISO, pin 12
- MOSI : SPI MOSI, pin 11
- REQ : SPI Chip Select, pin 10
- RST : pin 9 (pour initialiser la carte)
- RDY : pin 2 (pour les interruptions)
- ACT : -

5.5 UTILISATION DES CAPTEURS

5.5.1 DHT11

Avant de pouvoir utiliser le capteur de température / humidité, il est nécessaire de copier / coller la librairie d'Adafruit dans l'IDE et d'inclure dans notre projet le fichier d'entêtes « DHT.h ». L'utilisation de cette librairie nous facilite grandement la tâche ; toutes les fonctions nécessaires à l'utilisation et à la prise de mesures sont, en effet, déjà implémentées.

Déroulement de l'utilisation du DHT11 :

- Importation de la librairie et référence vers le fichier « DHT.h »
- Définition du pin utilisé et déclaration du type de capteur utilisé (DHT11)
- Initialisation du capteur (dht.begin())
- Lecture du capteur (dht.readTemperature(), dht.readHumidity())
- (optionnel) Affichage de la mesure

5.5.2 PHOTORESISTANCE

L'utilisation de la photorésistance est très simple, il suffit de lire la valeur renvoyée par celle-ci, à l'aide de l'instruction « analogRead(A0) » (A0 étant l'entrée analogique où la photorésistance étant connectée).

Afin d'améliorer l'exactitude de la mesure de la luminosité, nous pouvons imaginer une phase de calibration au démarrage du programme, afin de calquer les mesures suivantes sur une valeur référence. D'autre part, il peut être intéressant de réduire l'intervalle des valeurs, en les disposant sur une échelle plus réduite. Nous utiliserons, dans notre cas, un intervalle de 0 à 100, à l'aide de la méthode « map() ».

⁶<https://learn.adafruit.com/getting-started-with-the-nrf8001-bluefruit-le-breakout/hooking-everything-up>

5.6 WATCHDOG TIMER

Afin de palier au problème de la consommation énergétique de la station météo, il peut paraître nécessaire de réfléchir à comment économiser du courant et donc, si l'Arduino est connecté à une batterie, d'en prolonger la durée de vie.

Watchdog est un mécanisme incorporé au processeur AVR, qui permet de réinitialiser le processeur si une erreur survient. Il peut être aussi utilisé en tant que « timer », afin de réveiller le processeur d'un état de veille (« power-down sleep »). C'est donc pour cette application que nous allons l'utiliser, afin de mettre en veille et de réveiller l'Arduino entre les périodes de mesures. Plus de détails sur ce mécanisme peuvent être trouvés aux chapitres 9 et 10.8 du datasheet sur l'ATmega328P⁷.

Cette partie étant optionnel, la conception est à retrouver sur le site d'Adafruit : <https://learn.adafruit.com/low-power-wifi-datalogging/power-down-sleep> et le code utilisé, adapté pour notre station météo, provient du même site⁸.

5.7 BLUETOOTH LE – NRF8001

Afin de pouvoir utiliser la puce radio nRF8001, diverses acquisitions seront à effectuer :

- Télécharger et importer la librairie BLE de Nordic Semiconductor⁹
- Télécharger et importer les fichiers « nRF8001Device.cpp / .h »¹⁰
- Télécharger et importer les fichiers « BLE_broadcast_TxPower » et « BLE_connect_TxPower.h »

D'autre part, diverses modifications / ajouts seront à réaliser à l'intérieur du sketch :

- Inclure les fichiers d'entêtes relatifs à l'ACI et au bus SPI
- Définir des macros permettant de spécifier des arguments pour le constructeur de la classe « nRF8001Device »
- Définir des macros permettant d'utiliser ou non le code réalisé dans le TP précédent afin de réaliser des mesures de température, d'humidité et de luminosité
- Instancier un objet de la classe « nRF8001Device » et utiliser les différentes méthodes de la classe afin de l'initialiser et de l'utiliser

⁷ <http://www.atmel.com/Images/doc8161.pdf>

⁸ https://github.com/tdicola/Low_Power_Wifi_Datalogger/archive/master.zip

⁹ <https://github.com/NordicSemiconductor/ble-sdk-arduino>

¹⁰ <http://cyberlearn.hes-so.ch/mod/folder/view.php?id=439257>

6 REALISATION

6.1 BLINK

Afin de faire clignoter une led, nous devons tout d'abord définir le numéro du pin sur lequel elle est connectée.

```
uint8_t LED_PIN = 13;
```

Pour nous définissons le pin en tant que sortie, il nous permet ainsi d'y faire « sortir » du courant. Dans le cas d'un bouton, par exemple, on l'aurait défini en tant qu'entrée (INPUT).

```
pinMode(LED_PIN, OUTPUT); // sets the pin as output
```

Nous pouvons dès à présent utiliser la led. Pour cela, nous pouvons utiliser la méthode « digitalWrite() », qui permet d'allumer une led (niveau logique haut) ou d'éteindre une led (niveau logique bas). Dans le cas d'un « HIGH », la tension de sortie du pin sera donc de 5 V, ce qui permet d'alimenter leds, capteurs, etc.

```
digitalWrite(LED_PIN, HIGH); // switches on the led
delay(1000); // waits for 1000ms -> 1s
digitalWrite(LED_PIN, LOW); // switches off the led
delay(1000);
```

Quant à la fonction « delay() », elle permet de mettre en pause le programme pendant un temps t [ms] donné.

6.2 LOGGING

6.2.1 LOGGING.CPP

Les méthodes « PrintHeader() » et PrintSerial » sont appelées par les différentes macros de logging et permettent d'afficher convenablement les messages et arguments passés à celles-ci.

La méthode « PrintHeader() » permet d'afficher le temps d'exécution jusqu'à l'appel de l'une des macros de logging. Nous utilisons la méthode « millis() » pour récupérer le temps d'exécution [ms], duquel nous pouvons en tirer les heures / minutes / secondes à l'aide de divisions et de modulus.

```
void PrintHeader(const char* szHeaderType) {
    // returns the number of ms since the Arduino began running
    unsigned long time = millis();
    Serial.print(szHeaderType);
    Serial.print("\tat time : ");
    Serial.print(time / 3600000); // hours
    time = time % 3600000;
    Serial.print("h ");
    Serial.print(time / 60000); // minutes
    time = time % 60000;
    Serial.print("m ");
    Serial.print((time / 1000) % 60); // seconds
    Serial.print("s : \t");
}
```

La méthode « PrintSerial() » permet simplement d'afficher deux paramètres sur la même ligne, ce qui n'est pas possible avec un seul « Serial.print() ».

```
void PrintSerial(const char* format, int arg1) {
    Serial.print(format);
    Serial.println(arg1);
}
```

6.2.2 LOGGING.H

Le fichier « Logging.h » contient les différentes macros de logging. Nous débutons par déclarer la constante de préprocesseur « LOGGING_H », importer les librairies nécessaires au bon fonctionnement du programme, à « récupérer » la variable « g_logging_level », instanciée dans le fichier « Logging.cpp » et à déclarer une constante « TR_LOGLEVEL » à destination du préprocesseur, afin de définir le niveau de logging et donc, le niveau de compilation des différentes macros ci-dessous. Par exemple, en assignant la valeur 0 à « TR_LOGLEVEL », nous obtiendrons le programme le plus léger ; cependant, nous disposerons d'aucune information de logging dans la console.

```
#pragma once
#ifndef LOGGING_H
#define LOGGING_H

#include <stdio.h>
#include "Arduino.h"

extern uint8_t g_logging_Level;

//logging macros
#ifndef TR_LOGLEVEL
#define TR_LOGLEVEL 3
#endif
```

On « importe » ensuite les deux méthodes écrites dans le fichier « Logging.cpp », afin de pouvoir les appeler depuis les macros.

```
void PrintHeader(const char* szHeaderType);
void PrintSerial(const char* format, int arg1);
```

Ci-dessous, les deux macros permettant d'afficher des logs à titre informatif, qui nécessitent un (« TraceInfo() ») ou deux arguments (« TraceInfoFormat »). Le « résultat » de ces deux macros sera visible dans la console uniquement si la variable destinée au préprocesseur « TR_LOGLEVEL » est plus grande ou égale à 2 et que le programmeur ait décidé d'afficher ces logs (« g_logging_level » ≥ 2).

```
#if(TR_LOGLEVEL >= 2)
#define TraceInfo(format) if (g_logging_Level >= 2) {PrintHeader("INFO"); Serial.println(format);}
#define TraceInfoFormat(format, arg1) if (g_logging_Level >= 2) { PrintHeader("INFO"); PrintSerial(format, arg1);}
#else
#define TraceInfo(format)
#define TraceInfoFormat(format, arg1)
#endif
```

Le « # » permet de définir des instructions pour le préprocesseur.

6.3 MEMOIRE LIBRE

Comme précédemment expliqué dans la partie conception, la mémoire libre peut être déterminée en soustrayant l'adresse d'une variable fraîchement instanciée par la première adresse disponible (« `_heap_start` »).

```
int displayFreeRam() {
    extern int _heap_start;
    uint8_t free_memory;
    return ((int)&free_memory) - ((int)&_heap_start);
}
```

(Corrections TP01) En utilisant le « `heap_start` », nous ne pouvons que traiter le cas où de la mémoire dynamique est allouée. Afin de palier à ce manqueument, nous pouvons utiliser la « `_brkval` ». Exemple¹¹ :

```
int freeMemory() {
    int free_memory;
    if ((int)_brkval == 0) {
        free_memory = ((int)&free_memory) - ((int)&_heap_start);
    } else {
        free_memory = ((int)&free_memory) - ((int)_brkval);
        free_memory += freeListSize();
    }
    return free_memory;
}
```

6.4 UTILISATION DES CAPTEURS

6.4.1 DHT11

Afin d'utiliser le capteur, il nous faut tout d'abord importer la librairie, réalisée par Adafruit , définir la broche utilisée par le capteur et créer une instance d'un capteur DHT11 :

```
#include "DHT.h"
#define DHTPIN 8
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
```

(Compléments) constructeur de la classe DHT :

```
DHT::DHT(uint8_t pin, uint8_t type, uint8_t count) {
    _pin = pin;
    _type = type;
    _count = count;
    firstreading = true;
}
```

On peut ensuite passer à l'initialisation du capteur, dans la boucle « `setup()` », à l'aide de la fonction « `begin()` » :

```
dht.begin();
```

(Compléments) méthode « `begin()` » :

```
void DHT::begin(void) {
    // set up the pins!
    pinMode(_pin, INPUT);
    digitalWrite(_pin, HIGH);
    _lastreadtime = 0;
}
```

¹¹ <http://playground.arduino.cc/code/AvailableMemory>

Nous pouvons dès lors passer à la prise de mesures, à l'aide des méthodes « dht.readTemperature() » et « dht.readHumidity() » :

```
h = dht.readHumidity();
t = dht.readTemperature();
```

(Compléments) le code des méthodes « readTemperature() » et « readHumidity() » est disponible dans le fichier « DHT.cpp », livré avec la librairie.

6.4.2 PHOTORESISTANCE

La photorésistance étant un capteur analogique, il très aisément de recueillir la mesure effectuée par celle-ci à l'aide la méthode « analogRead(A0) » :

```
l = analogRead(A0);
```

Afin de « calibrer » les valeurs lues par l'Arduino, il peut être intéressant de mesurer des valeurs référence. Cette phase se déroulera sur les 5 premières secondes d'exécution du programme, où le capteur sera continuellement lu et permettra, si la valeur est inférieure ou supérieure aux bornes en place (Low level : 0 / High level : 1023), de les modifier.

```
// for calibration ...
int lowLight = 0;
int highLight = 1023;

void setup() {
    while (millis() < 5000) {
        l = analogRead(A0);
        if (l > highLight)
            highLight = l;
        if (l < lowLight)
            lowLight = l;
    }
}
```

Nous pouvons ensuite étalonner les valeurs sur une échelle plus réduite, en prenant en considération les deux variables précédemment calibrées, à l'aide de la fonction « map() ».

```
l = map(l, lowLight, highLight, 0, 100);
```

6.5 BLUETOOTH LE – NRF8001

Afin d'utiliser la puce nRF8001, nous débutons par importer les différentes librairies nécessaires au bon fonctionnement de la puce radio et nous incluons les fichiers d'entêtes correspondants. Nous incluons également le fichier « BLEconnect_TxPower.h » ou le fichier « BLEboradcast_TxPower.h », en fonction du mode de fonctionnement désiré (mode connecté / broadcast).

```
#include <SPI.h>
#include <aci_cmds.h>
#include <aci_evts.h>
#include <hal_aci_t1.h>
#include <lib_aci.h>
#include <EEPROM.h>

#include "NRF8001Device.h"
#include "_BLE_connect_TxPower.h"
// #include "_BLE_broadcast_TxPower.h"
```

Nous définissons ensuite des macros, sous forme de paramètres, pour l'instanciation d'un objet « nRF8001Device » et les broches où est connectée la puce radio.

```
static const hal_aci_data_t setup_msgs[NB_SETUP_MESSAGES] PROGMEM = SETUP_MESSAGES_CONTENT;
// for storing the service pipe data created in nRFgo Studio
#ifndef SERVICES_PIPE_TYPE_MAPPING_CONTENT
    static services_pipe_type_mapping_t services_pipe_type_mapping[NUMBER_OF_PIPES] =
SERVICES_PIPE_TYPE_MAPPING_CONTENT;
#else
    #define NUMBER_OF_PIPES 0
    static services_pipe_type_mapping_t* services_pipe_type_mapping = NULL;
#endif

#define REQ 10
#define RDY 2
#define RST 9
```

Nous instancions ensuite un objet de la classe « nRF8001Device », qui nous permettra d'utiliser la puce Bluetooth.

```
// creates a instance of nRF8001Device
nRF8001Device nrf(
    setup_msgs,
    NB_SETUP_MESSAGES,
    services_pipe_type_mapping,
    NUMBER_OF_PIPES,
    nRF8001Device::CONNECT,
    REQ,
    RDY,
    RST);

// ACI event status on the nRF8001
aci_evt_opcode_t lastStatus = ACI_EVT_DISCONNECTED;
```

Nous initialisons ensuite l'objet créé précédemment dans la fonction « setup() », à l'aide des méthodes « setDeviceName() » et « begin() ».

```
// sets the device name
nrf.setDeviceName("IoT_Samuel");
// starts the nRF8001
bool nrfIsStarted = nrf.begin(1, 0x0100, 0);
if (nrfIsStarted == false)
    TraceError(F("nRF8001 device can't be started"));
```

Nous pouvons ensuite utiliser les différentes méthodes relatives à la puce radio dans la boucle principale (« loop() ») du sketch. Nous récupérons ensuite l'état dans lequel se trouve l'ACI et nous l'affichons sur la console.

```
// gets all aci events at regular intervals
nrf.pollACI();

// gets the device state
aci_evt_opcode_t status = nrf.getState();

if (status != lastStatus) {
    if (status == ACI_EVT_DEVICE_STARTED) {
        TraceInfo(F("Advertising started"));
    }
    if (status == ACI_EVT_CONNECTED) {
        TraceInfo(F("Connected!"));
    }
    if (status == ACI_EVT_DISCONNECTED) {
        TraceInfo(F("Disconnected or advertising timed out"));
    }
    // updates the last status to this one
    lastStatus = status;
}
```

Si une connexion est active, nous affichons sur la console la mesure actuelle (température, humidité et luminosité). Le code nécessaire à la prise de mesures est défini dans des macros, ce qui permet d'activer ou non les mesures et ainsi, d'économiser de la mémoire.

```

if (status == ACI_EVT_CONNECTED) {
    #if (MEASURING)
        // reads the sensors
        h = dht.readHumidity();
        t = dht.readTemperature();
        l = analogRead(A0);

        // checks if any reads failed and exit early (to try again).
        if (isnan(h) || isnan(t) || isnan(l)) {
            TraceInfo(F("Failed to read from DHT sensor or from the photocell!"));
            return;
        }
        // displays the measures
        Serial.print(F("Humidity: "));
        Serial.print(h);
        Serial.print(F(" % Temperature: "));
        Serial.print(t);
        Serial.print(F(" *C Light level: "));
        Serial.print(l);
        Serial.println(F(" [0 - 1023]"));
        delay(5000);
    #endif
}

```

Les définitions nécessaires à la prise de mesures, déclarées en amont des méthodes « `setup()` » et « `loop()` ».

```

// 1/0 : On/Off
#ifndef MEASURING
#define MEASURING 1
#endif

#if (MEASURING)
#include "DHT.h"
#define DHTPIN 8
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
// variables for the measures...
float h = 0; // humidity
float t = 0; // temperature
float l = 0; // light
#endif

```

Et l'initialisation du capteur DHT11 se fait à l'intérieur de la boucle « `setup()` ».

```

#if (MEASURING)
    dht.begin(); // initialization the DHT11
#endif

```

7 TESTS ET VALIDATION

7.1 TP01

Afin d'attester le bon fonctionnement des macros de logging, nous pouvons incorporer des appels vers celles-ci à l'intérieur du code réalisé pour les étapes de ce présent TP. Nous utilisons donc les méthodes « `TraceErrorFormat()` », « `TraceInfoFormat()` » et `TraceDebugFormat()` » et les constantes « `TR_LOGLEVEL` » et « `g_logging_level` » afin de faire varier la taille du programme compilé et l'affichage en sortie sur la console.

En définissant un « `TR_LOGLEVEL` » à 0, nous obtenons un code compilé d'une taille de 184 bytes, ce qu'illustre le fait qu'aucune des macros n'a été compilée et donc, nous obtenons aucun message sur la console. Mais en définissant des niveaux de logging plus élevés, nous pouvons constaté, comme le montrent très bien les figures ci-dessous, que la taille des programmes augmente, ainsi que la quantité d'information affichée sur la console.

7.1.1 LOGGING DE NIVEAU 1

Niveau de logging à 1 : affichage sur la console que des messages d'erreur ; taille des données de 242 bytes.

```
Sketch uses 2,940 bytes (9%) of program storage space. Maximum is
32,256 bytes.
Global variables use 242 bytes (11%) of dynamic memory, leaving
1,806 bytes for local variables. Maximum is 2,048 bytes.
```

Figure 21: Compilation du programme avec un niveau de logging de 1

```
ERROR at time : 0h 0m 4s : Print an error message ...
ERROR at time : 0h 0m 1s : Print an error message ...
ERROR at time : 0h 0m 4s : Print an error message ...
ERROR at time : 0h 0m 6s : Print an error message ...
ERROR at time : 0h 0m 8s : Print an error message ...
ERROR at time : 0h 0m 10s : Print an error message ...
```

Figure 22: Sortie sur la console

7.1.2 LOGGING DE NIVEAU 2

Niveau de logging à 2 : affichage sur la console des messages d'erreur et d'information; taille des données de 290 bytes.

```
Sketch uses 3,310 bytes (10%) of program storage space. Maximum is
32,256 bytes.
Global variables use 290 bytes (14%) of dynamic memory, leaving
1,758 bytes for local variables. Maximum is 2,048 bytes.
```

Figure 23: Compilation du programme avec un niveau de logging de 2

```

INF 0s :      Free memory : 1751
INFO at time : 0h 0m 0s : The led is switched on; #1
INFO at time : 0h 0m 0s : Free memory : 1751
INFO at time : 0h 0m 0s : The led is switched on; #1
ERROR at time : 0h 0m 2s : Print an error message ...
INFO at time : 0h 0m 2s : The led is switched on; #2
ERROR at time : 0h 0m 4s : Print an error message ...

```

Figure 24: Sortie sur la console

7.1.3 LOGGING DE NIVEAU 3

Niveau de logging à 3 : affichage sur la console de tous les types de logging (ERROR, INFO & DEBUG) ; taille des données de 242 bytes.

Avec cette figure, nous pouvons aussi attester du bon fonctionnement de la méthode de calcul de la mémoire libre, qui retourne une valeur de 1719. Ce n'est, certes, pas identique à la valeur affichée dans la console, mais celle-ci en est très proche (+ 7 bytes).

```

Done compiling.

Sketch uses 3,370 bytes (10%) of program storage space. Maximum is
32,256 bytes.
Global variables use 322 bytes (15%) of dynamic memory, leaving
1,726 bytes for local variables. Maximum is 2,048 bytes.

3
Arduino Uno on /dev/tty.usbmodem1411

```

Figure 25: Compilation du programme avec un niveau de logging à 3

```

INFO at time : 0h 0m 0s : Free memory : 1719
INFO at time : 0h 0m 0s : The led is switched on; #1
DEBUG at time : 0h 0m 2s : Print a debug message ...
ERROR at time : 0h 0m 2s : Print an error message ...
INFO at time : 0h 0m 2s : The led is switched on; #2
DEBUG at time : 0h 0m 4s : Print a debug message ...
ERROR at time : 0h 0m 4s : Print an error message ...

```

Figure 26: Sortie sur la console

Synthèse des tests réalisés :

- ✓ Affichage de la valeur du compteur à chaque itération
- ✓ Affichage des logs de niveau 1
- ✓ Affichage des logs de niveaux 1-2
- ✓ Affichage des logs de niveaux 1-3
- ✓ Variation de la taille du programme compilé en fonction du niveau de logging
- ✓ Affichage de la quantité de mémoire libre au démarrage du programme

7.2 TP02

Afin de vérifier le bon fonctionnement du programme, consistant à prendre des mesures de température, d'humidité et de luminosité à des intervalles régulières, nous pouvons utiliser la console, dans laquelle on écrit les résultats issus du capteur DHT11 et de la photorésistance.

```
// displays the values
Serial.print(F("Humidity: "));
Serial.print(h);
Serial.print(F(" % Temperature: "));
Serial.print(t);
Serial.print(F(" *C Light level: "));
Serial.print(l);
Serial.println(F(" [0 - 100]"));
```

Ce code nous permet d'obtenir les messages suivants dans la console :

```
Humidity: 52.50 % Temperature: 20.80 *C Light level: 61 [0 - 100]
Humidity: 52.50 % Temperature: 20.80 *C Light level: 34 [0 - 100]
Humidity: 52.50 % Temperature: 20.80 *C Light level: 65 [0 - 100]
Humidity: 52.50 % Temperature: 20.80 *C Light level: 28 [0 - 100]
Humidity: 52.50 % Temperature: 20.80 *C Light level: 30 [0 - 100]
```

Figure 27: Sortie sur la console

L'étape suivante consiste à réaliser des mesures sur une plus longue période afin de démontrer des variations de température, d'humidité ou luminosité.

Conditions de la mesure :

- Durée : 60 minutes ; une mesure par minute
- Capteurs : DHT22 (température + humidité) et photorésistance
- Conditions : Fenêtre ouverte / fermée ; Store ouvert / fermé ; Lampe éteinte / allumée

#	Time	T	H	L
0	72	19.4	45.2	76
1	137	19.4	45.2	76
2	203	19.4	45.2	78
3	268	19.4	45.2	79
4	333	19.4	45.2	76
5	399	19.4	45.2	75
6	464	19.4	45.2	76
7	530	19.4	45.2	76
8	595	19.4	45.2	79
9	660	19.4	45.2	80
10	726	19.4	45.2	80
11	791	19.4	45.2	81
12	856	19.4	45.2	81
13	922	19.4	45.2	81
14	987	19.4	45.2	1
15	1052	19.4	45.2	1
16	1118	19.4	45.2	0
17	1183	19.4	45.2	1
18	1249	17.1	38.2	1

#	Time	T	H	L
19	1314	17.1	38.2	1
20	1379	17.1	38.2	1
21	1445	17.1	38.2	1
22	1510	17.1	38.2	1
23	1576	17.1	38.2	1
24	1641	17.1	38.2	1
25	1706	17.1	38.2	1
26	1772	17.1	38.2	9
27	1837	17.1	38.2	96
28	1903	17.1	38.2	95
29	1968	17.1	38.2	95
30	2034	17.1	38.2	95
31	2099	17.1	38.2	96
32	2165	17.1	38.2	93
33	2230	17.1	38.2	90
34	2296	17.1	38.2	94
35	2361	17.1	38.2	86
36	2427	14.6	45.7	86
37	2492	14.6	45.7	88

#	Time	T	H	L
38	2558	14.6	45.7	89
39	2623	14.6	45.7	86
40	2689	14.6	45.7	85
41	2773	18.6	81.2	94
42	2838	18.6	81.2	96
43	2904	18.6	81.2	96
44	2969	18.6	81.2	79
45	3069	19.9	44	69
46	3134	19.9	44	80
47	3200	19.9	44	81
48	3265	19.9	44	79
49	3331	19.9	44	78
50	3396	19.9	44	77
51	3462	19.9	44	76
52	3527	19.9	44	75
53	3593	19.9	44	73
54	3658	19.9	44	73
		19.3	44.8	57.8

Figure 28: Mesures effectuées sur une période de 60 minutes

Les mesures ont été récoltées à l'aide d'un petit script réalisé en Java, permettant de lire les données provenant du câble USB, d'en extirper les valeurs et de les enregistrer dans un fichier .csv.

L'utilisation du script, disponible dans le dossier « tp02/ ¹² », s'effectue à l'aide d'un terminal :

```
java -jar ArduinoDataCollector.jar /dev/tty.usbmodem1411 3600
```

- /dev/tty.usbmodem1411: port où l'Arduino est connectée
- 3600: durée, en seconds, de la mesure

Nous avons ensuite tout loisir de créer un graphique à partir de ces données.

¹² <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/blob/master/tp02/ArduinoDataCollector.jar>

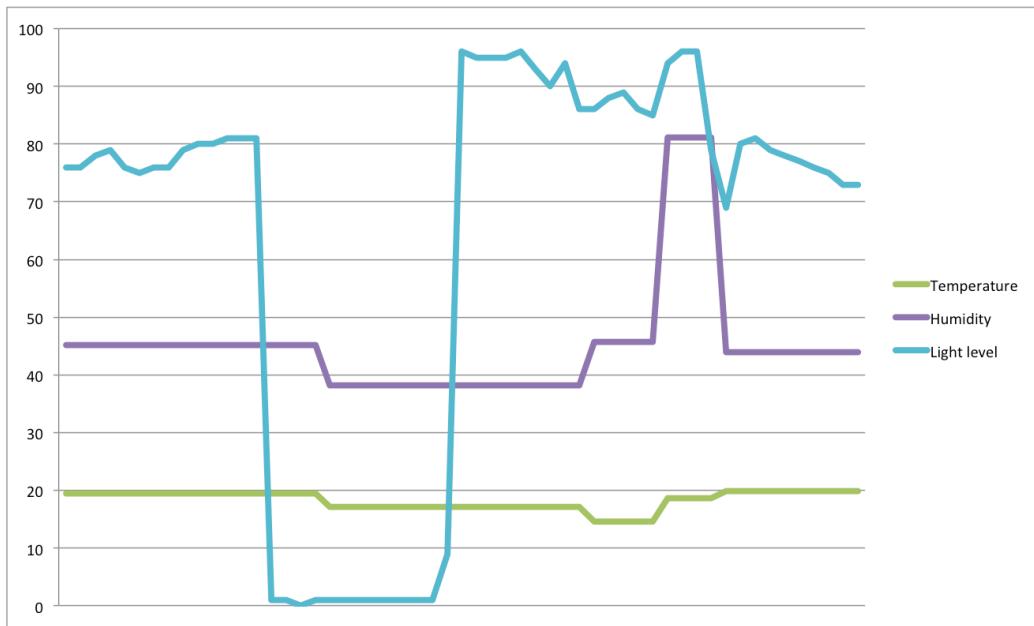


Figure 29: Graphique illustrant les variations des mesures effectuées précédemment

Remarque : Suite à l'implémentation de Watchdog, dans le but de limiter la consommation entre les mesures, le capteur DHT22 me semble moins réactif aux variations environnantes que précédemment. Cependant, la lecture de la photorésistance ne pose aucun problème.

Le programme est fonctionnel avec l'utilisation de Watchdog, cependant, n'ayant pas le matériel nécessaire, il m'a été impossible de mesurer les éventuelles et supposées, économies de courant.

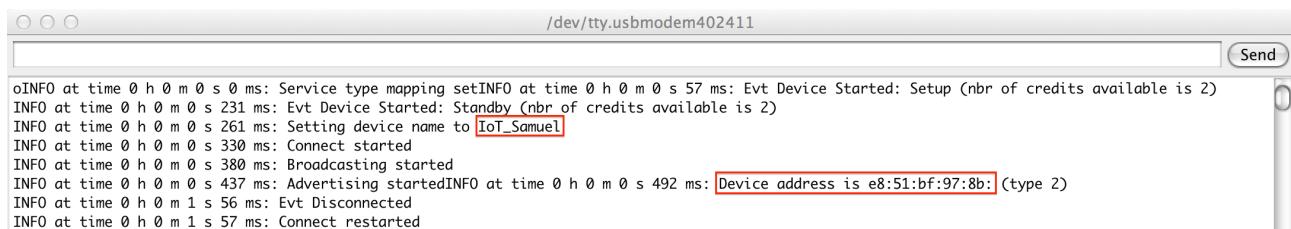
Synthèse des tests réalisés :

- ✓ Affichage de la température, de l'humidité et de la luminosité sur la console
- ✓ Prise de mesures sur une période d'une heure, à l'aide d'un script réalisé en Java et traitement des données avec Excel
- ✓ Implémentation de Watchdog (optionnel)

7.3 TP03

Dans le but de vérifier le fonctionnement de la partie Bluetooth, nous débutons par utiliser l'entête « BLE_connect_TxPower.h » afin de rendre possible la connexion d'un périphérique à la puce radio.

En démarrant la console, nous pouvons constater que la puce est bien initialisée avec l'identifiant désiré (« IoT_Samuel ») et que celle-ci dispose d'une adresse physique.



```

oINFO at time 0 h 0 m 0 s 0 ms: Service type mapping setINFO at time 0 h 0 m 0 s 57 ms: Evt Device Started: Setup (nbr of credits available is 2)
INFO at time 0 h 0 m 0 s 231 ms: Evt Device Started: Standby (nbr of credits available is 2)
INFO at time 0 h 0 m 0 s 261 ms: Setting device name to IoT_Samuel
INFO at time 0 h 0 m 0 s 330 ms: Connect started
INFO at time 0 h 0 m 0 s 380 ms: Broadcasting started
INFO at time 0 h 0 m 0 s 437 ms: Advertising startedINFO at time 0 h 0 m 0 s 492 ms: Device address is e8:51:bf:97:8b:37 (type 2)
INFO at time 0 h 0 m 1 s 56 ms: Evt Disconnected
INFO at time 0 h 0 m 1 s 57 ms: Connect restarted

```

Figure 30: Initialisation de la puce nRF8001

Ensuite, à l'aide de l'application « nRF Master Control Panel » pour Android, nous pouvons lister et nous connecter à la station météo désirée, via le bouton « CONNECT».

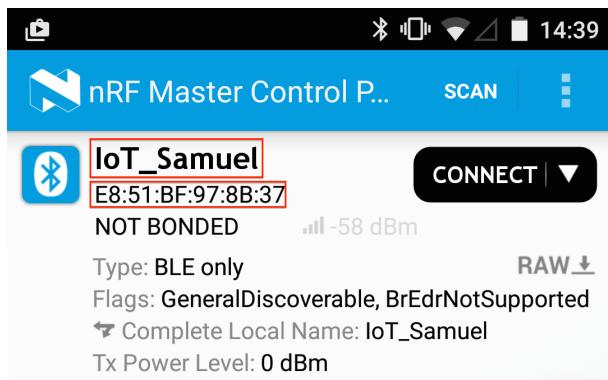


Figure 31: Liste des périphériques BLE en mode « connecté »

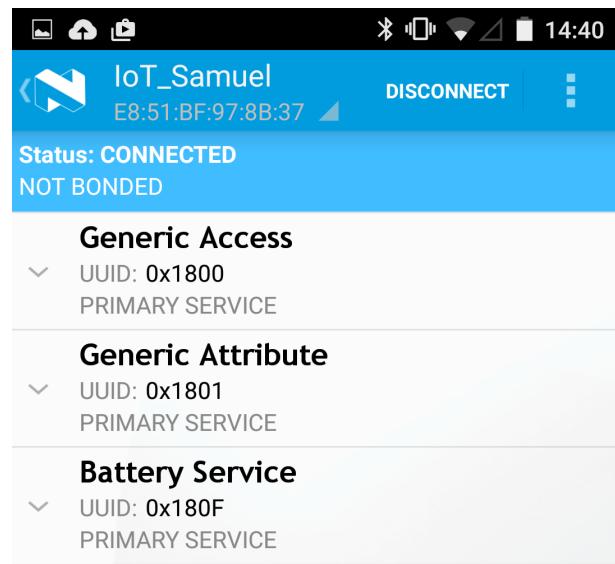


Figure 32: Caractéristiques fournies par la station météo

En s'y connectant, nous pouvons observer, dans l'application mobile, les différentes caractéristiques définies pour ce service, ainsi que le nom et l'adresse physique de la puce, identiques aux informations visibles sur la console. De plus, la connexion d'un périphérique conduit à la prise de mesures et à l'affichage de celles-ci sur la console.

```

INFO at time 0 h 0 m 15 s 50 ms: Evt Disconnected
INFO at time 0 h 0 m 15 s 52 ms: Connect restarted
INFO at time 0 h 0 m 15 s 89 ms: Broadcasting restarted
INFO at time 0 h 0 m 16 s 51 ms: Evt Disconnected
INFO at time 0 h 0 m 16 s 52 ms: Connect restarted
INFO at time 0 h 0 m 16 s 89 ms: Broadcasting restarted
INFO at time 0 h 0 m 16 s 201 ms: Evt Connected
INFO at time 0 h 0 m 16 s 202 ms: Connected!Humidity: 42.00 % Temperature: 20.00 *C Light level: 921.00 [0 - 1023]
INFO at time 0 h 0 m 21 s 521 ms: Evt PipeStatus: pipes_open_bitmap[0] is 0x00000003
Humidity: 41.00 % Temperature: 18.00 *C Light level: 921.00 [0 - 1023]
Humidity: 44.00 % Temperature: 23.00 *C Light level: 923.00 [0 - 1023]
Humidity: 42.00 % Temperature: 20.00 *C Light level: 916.00 [0 - 1023]
Humidity: 41.00 % Temperature: 18.00 *C Light level: 923.00 [0 - 1023]

```

Autoscroll No line ending 9600 baud

Figure 33: Affichage des mesures lorsque qu'un périphérique est connecté au nRF8001

En incluant maintenant le fichier d'entêtes « BLE_broadcast_TxPower.h », nous pouvons constater la disparition du bouton « CONNECT ». C'est tout à fait normal, car nous nous trouvons maintenant en mode « broadcast ». Ce mode permet uniquement d'émettre des paquets et n'accepte pas de connexions.

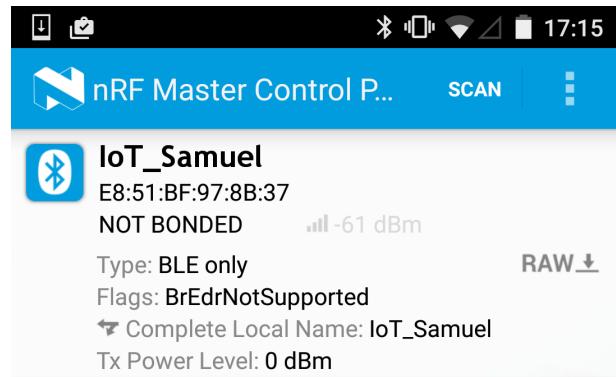


Figure 34: Liste des périphériques BLE en mode « broadcast »

Finalement, suite à la compilation du programme, nous pouvons attester du bon fonctionnement des macros, servant à « isoler » le code dédié aux mesures », observant la taille du sketch et la mémoire dédiée aux variables, en fonction de l'activation ou non des mesures (« MEASURING » à 1 ou à 0).

- « MEASURING » à 1 : taille du sketch : 17kb (52 %), taille des variables : 850 bytes (41%)
- « MEASURING » à 0 : taille du sketch : 14kb (43%), taille des variables : 810 bytes (39%)

The screenshot shows the Arduino IDE interface with the title bar "IoT | Arduino 1.6.0". The tabs at the top are "IoT", "Logging.cpp", "Logging.h", "NRF8001Device.cpp", and "NRF 001". The code editor displays the following C++ code:

```
#ifndef MEASURING
#define MEASURING 1
#endif // !MEASURING
```

The status bar at the bottom shows the message "Done compiling." and the memory usage information: "Sketch uses 17,094 bytes (52%) of program storage space. Maximum is 32,256 bytes. Global variables use 850 bytes (41%) of dynamic memory, leaving 1,198 bytes for local variables. Maximum is 2,048 bytes."

Figure 35: Taille du sketch et de la mémoire avec les mesures

The screenshot shows the Arduino IDE interface. The title bar reads "IoT | Arduino 1.6.0". The top menu bar includes "File", "Edit", "Tools", "Sketch", "Help", and "Serial Monitor". Below the menu is a toolbar with icons for file operations. The central workspace has tabs for "IoT", "Logging.cpp", "Logging.h", "NRF8001Device.cpp", and "NRF8001Device.h". The "Logging.cpp" tab is active, displaying the following C++ code:

```
15  
16 #ifndef MEASURING  
17 #define MEASURING 0  
18 #endif
```

A progress bar at the bottom of the code editor indicates the compilation process. The status bar at the bottom of the IDE displays the message "Done compiling.".

Sketch uses 14,008 bytes (43%) of program storage space.
Maximum is 32,256 bytes.
Global variables use 810 bytes (39%) of dynamic memory,
leaving 1,238 bytes for local variables. Maximum is 2,048
bytes.

Figure 36: Taille du sketch et de la mémoire sans les mesures

Synthèse des tests réalisés :

- ✓ Importation des différents fichiers / librairie nécessaires au nRF8001
 - ✓ Instanciation, initialisation et utilisation du nRF8001 en mode « Connect » et « Broadcast »
 - ✓ Transformation du code précédent (prise de mesures) sous la forme de macros et lecture des capteurs lorsqu'un périphérique est connecté au nRF8001

8 PROBLEMES RENCONTRES & SOLUTIONS

13/03/2015 : Rendu du TP01 :

- J'ai rencontré quelques difficultés à comprendre le fonctionnement des macros de logging, mais après plusieurs explications, tout est rentré dans l'ordre.

25/03/2015 : Rendu du TP02 :

- Je n'ai pas rencontré de soucis à implémenter les tâches relatives au TP02.

17/04/2015 : Rendu du TP03 :

- Je n'ai pas rencontré de soucis à implémenter les tâches relatives au TP03.

9 ACQUIS

TP01 :

- Ce premier travail pratique a été l'occasion d'apprivoiser la plateforme Arduino et de prendre en mains son IDE. D'autre part, il nous a permis de comprendre le fonctionnement de macros de logging et d'appréhender le fonctionnement de la mémoire sur l'Arduino.

TP02 :

- Ce second travail pratique a été l'occasion d'installer une librairie tierce et de l'appréhender, afin de pouvoir utiliser le capteur DHT11. D'autre part, ce travail m'a permis d'approfondir mes connaissances en électricité, en étudiant le fonctionnement des diviseurs de tension et des résistances « pullup ». Par curiosité, je me suis intéressé au Timer Watchdog, permettant de réduire la consommation de la carte Arduino.

TP03 :

- Ce troisième travail pratique nous a permis d'appréhender le fonctionnement de l'ACI et de ses différents événements et commandes permettant la communication avec la puce nRF8001. D'autre part, nous avons été amenés à s'imprégnés d'une classe C++ tierce (« nRF8001Device »), d'en comprendre le fonctionnement et d'en utiliser les principales méthodes.

10 PERSPECTIVES

Cette première approche avec l'Arduino laisse entrevoir de belles opportunités, en y branchant, par exemple, des capteurs afin de démarrer notre projet de station météo connectée.

Cette seconde approche, plus concrète, nous a permis de récolter des données réelles de température, d'humidité et de luminosité. Ces données pourront être traitées et consultées dans le futur, suite aux prochains travaux pratiques.

Cette troisième approche, très théorique et orientée principalement sur la communication entre l'ACI et la puce nRF8001, nous a permis d'appréhender la technologie Bluetooth. Tout nous laisse à penser, que dans des travaux futurs, nous serons amenés à réaliser les différentes fonctions nécessaires aux transferts de données entre la puce et les différents périphériques.

11 CONCLUSION

Ce second travail fût intéressant, car il nous a permis de lier des concepts théoriques (diviseur de tension, résistance « pullup », etc) à des actes plus pratiques, comme la lecture des différents capteurs. Cela nous laisse donc de belles perspectives devant nous, à commencer par l'implémentation d'une communication sans-fil (Bluetooth) et à appréhender l'aspect de la consommation énergétique.

12 ANNEXE

Le code source du projet se trouve sur Git, à l'adresse : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project>.

- TP01 : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp01>
- TP02 : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp02>
- TP03 : <https://forge.tic.eia-fr.ch/git/samuel.mertenat/iot-project/tree/master/tp03>

13 REFERENCES

- <http://www.louisreynier.com/fichiers/KesacoArduino.pdf>
- <http://openclassrooms.com> (cours sur l'Arduino, plus disponible)
- <http://arduino.cc/en/Reference/Serial>
- <http://playground.arduino.cc/Code/AvailableMemory>
- <http://www.nongnu.org/avr-libc/user-manual/malloc.html>
- <http://arduino.cc/en/reference/map>
- Photorésistance : <http://www.instructables.com/id/Photocell-tutorial/?ALLSTEPS>
- Résistance « pullup » : <https://learn.sparkfun.com/tutorials/pull-up-resistors>
- Diviseur de tension : <https://learn.sparkfun.com/tutorials/voltage-dividers>
- DHT11 : <http://www.micropik.com/PDF/dht11.pdf>
- nRF8001 :
https://www.nordicsemi.com/eng/nordic/download_resource/17534/16/24946618