# Projet PERI :
# Dispositif météorologique

Rapport de projet

Paul MABILLOT & Adrien FERREIRA & Pierre MAHÉ

24 avril 2015

UPMC
PARIS UNIVERSITAS

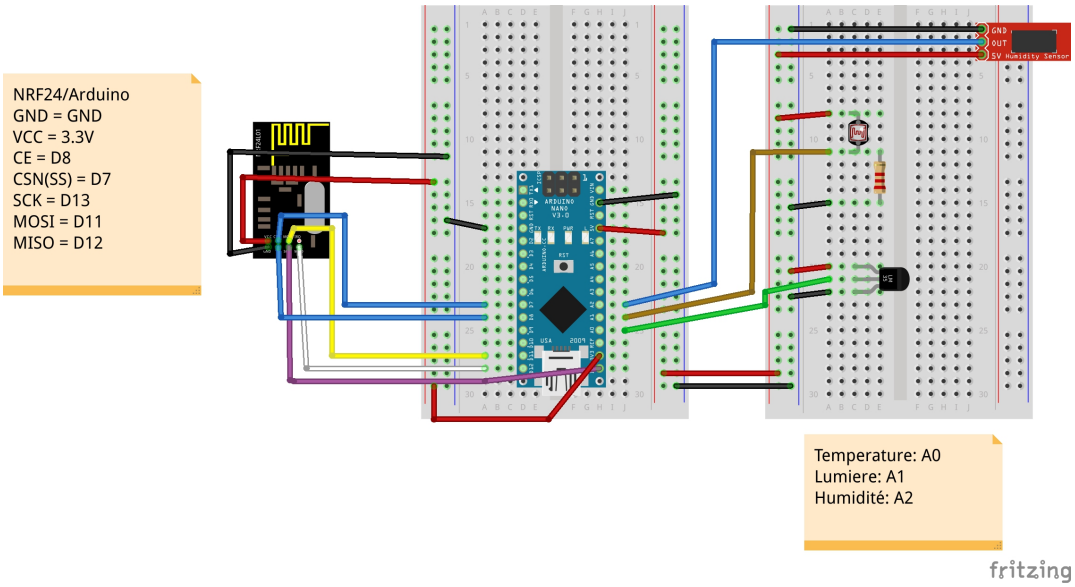
# Table des matières

# 1 Description du projet

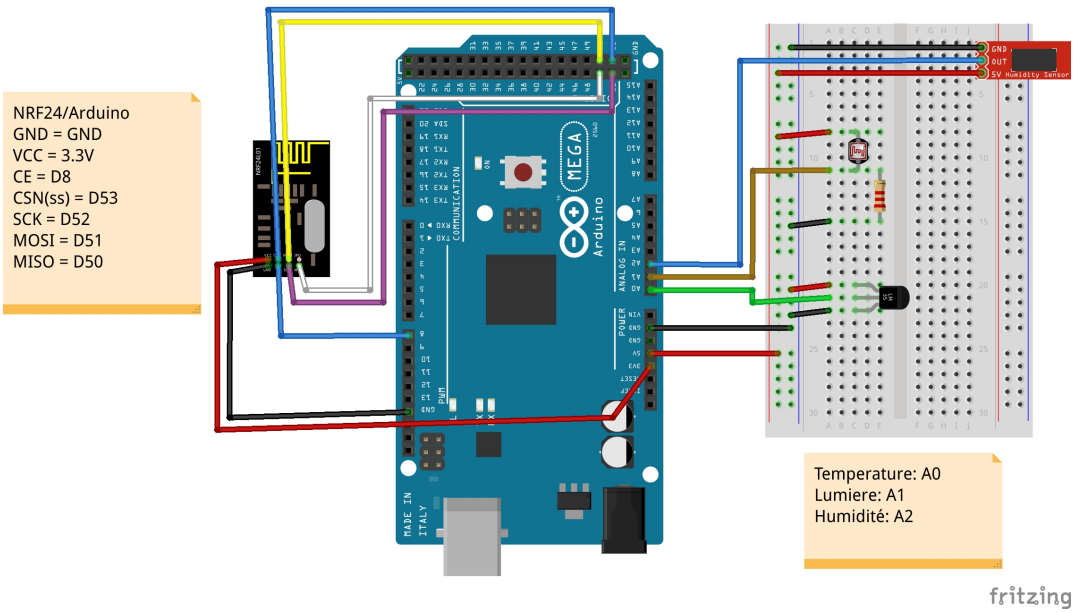
# 2 Montage

## 2.1 Arduino

Faire le montage suivant votre carte Arduino.
Pour les cartes Nano et One les branchements sont similaire. À noter que nous n'utilisons pas de led externe mais celle intégrée au circuit.

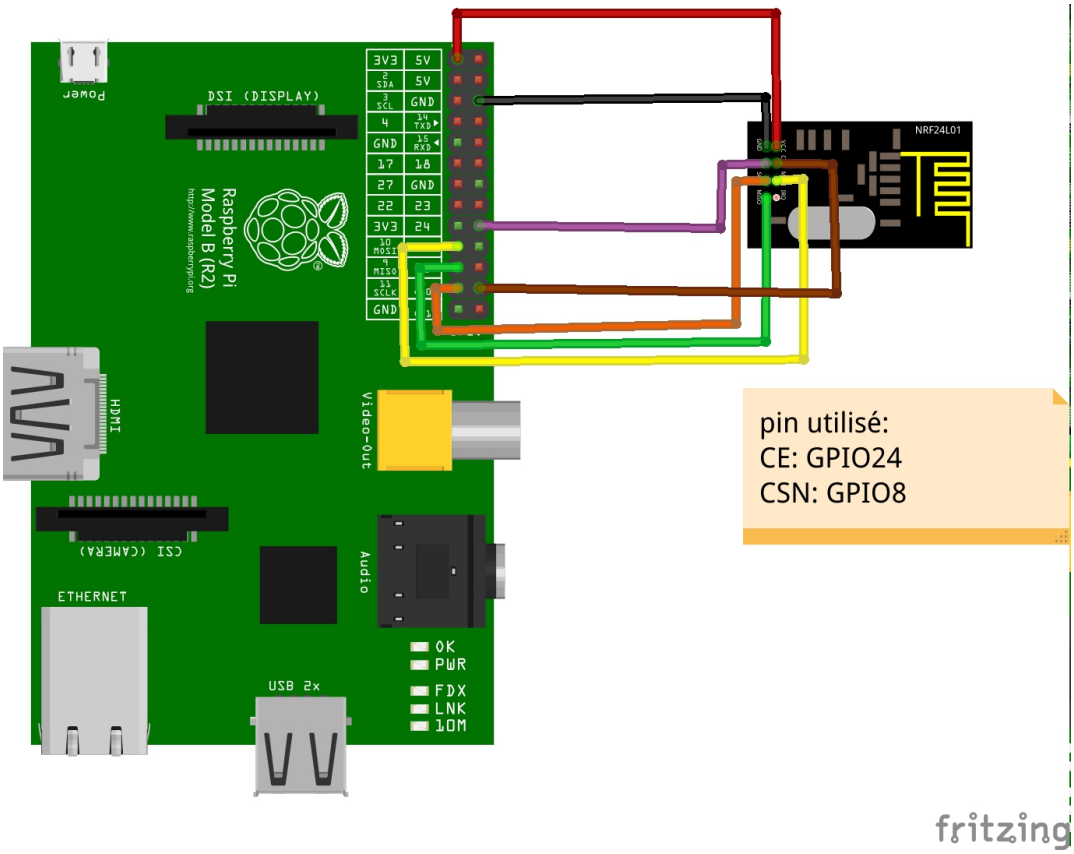



Pour l'arduino Mega le montage est un peu différent. Branchez le pin CSN(SS) au pin D53, le SCK au pin D52, le MOSI au pin D51 et le MISO au pin D50.




## 2.2 Raspberry Pi

Faire le montage suivant sur votre carte Raspberry Pi.
Dans cet exemple, nous avons utilisé une Raspberry Pi modèle B mais pour les modèles B+ et 2 la retro-compatibilité est assurée, il suffit de procéder au même branchement que le schéma.




**Attention : Le module nRF24 est fragile, il n'accepte qu'une alimentation 3.3V, ne surtout pas le brancher sur le pin 5V au risque de l'endommager.**

# 3 Codage

## 3.1 Structure de données

Cette structure est commune aux deux cartes, elle permet de structurer les données envoyées.

```
typedef struct t_payload{
  uint16_t cmd;
  uint16_t timeStamp;
  uint16_t valLight;
  uint16_t valTemperature;
  uint16_t valHumidity;
} payload;
```

Pour garantir l'interopérabilité, il est impératif de définir explicitement la taille des types manipulés. **Attention : certains compilateurs rajoutent des octets de bourrage pour aligner la taille de la structure en mémoire. Veillez donc à accorder les options des deux compilateurs (Arduino et Raspberry).** L'option `packed` peut être utilisée comme directive pour le compilateur.

L'endianess de l'Arduino et la Raspberry étant identique il n'a pas été nécessaire d'opérer à des conversions, attention cependant au portage sur d'autres systèmes. `cmd` utile pour définir le type de traitement à réaliser (cf : Protocole). Les autres champs sont présent pour stocker les données à proprement parler.

## 3.2 Arduino

Pour envoyer les données a intervalle reguliere, vous devez installer la bibliothèque *SimpleTimer* (téléchargeable sur le site officiel Arduino).
Il faut tout d'abord armer un timer avec un temps (en milliseconde) et un pointeur de fonction (sans paramètre) dans le `setup`. Dans le `loop`, appelez la fonction `run`.

```
SimpleTimer timer;
timer.setInterval(DELAY, fctnCallback);
timer.run()
```

Pour communiquer avec la module nRF24L01+, vous devez installer une seconde bibliothèque. *lien du dépôt*. Seul les fichiers `RF24.cpp` et `RF24.h` sont à copier dans le dossier `libraries` de l'IDE sketch.

### 3.2.1 Initialisation

La sequence d'instruction à utiliser pour initialiser le module nRF24 est :

```
radio.begin();
radio.setPALevel(RF24_PA_MAX);  // la force d'emission du signal
radio.setDataRate(RF24_1MBPS);  // le debit de donnees

// identifiant arbitraire du canal de emission
radio.openWritingPipe(0x0000000001LL);
// identifiant du canal de reception pour le pipe 1
radio.openReadingPipe(1, 0x0000000002LL);
```

**Attention : Il n'est possible d'écouter que sur 6 canaux simultanément.** Attention à correctement synchroniser les canaux utilisés par les nœuds. Si deux modules écrivent sur le même canal, des collisions vont apparaître.

### 3.2.2 Lecture

Avant de pouvoir envoyer des données grâce au module nRF24, il faut l'activer en lecture grâce à l'instruction : `startListening()`. Pour savoir si une donnée est en attente dans le buffer de réception, il faut utiliser l'instruction `avaiable()` non-bloquante. Pour récupérer la valeur, il faut utiliser `read`.

### 3.2.3 Écriture

Il n'est pas possible d'écouter et d'émettre simultanément. De ce fait, avant une écriture, il faut prendre soin d'appeler `stopListening()` pour arrêter l'écoute. Ensuite, utilisez `write()` pour envoyer les données. Ne pas oublier de réactiver l'écoute ensuite.

## 3.3 Raspberry Pi

Pour communiquer avec la module nRF24L01+ depuis la Raspberry, utilisez les fonctions présentes dans les fichiers `RF24.cpp`, `RF24.h`, `bcm2835.c` et `bcm2835.h`.

### 3.3.1 Initialisation

La phase d'initialisation est similaire à celle de l'Arduino.

```
/* les deux premiers arguments sont les pins GPIO a utiliser pour CE
   et CS (definie BCM2835.h) la cadence du pin SPI.*/
RF24 radio(BCM2835_SPI_CS_GPIO24, 800000, BCM2835_SPI_SPEED_8MHZ);
radio.begin();
radio.setPALevel(RF24_PA_MAX);     // la force d'emission du signal
radio.setDataRate(RF24_1MBPS);     // le debit de donnees

// identifiant du canal de reception pour le pipe 1
radio.openReadingPipe(1,0x0000000001LL);
// identifiant arbitraire du canal de emission
radio.openWritePipe(0x0000000002LL);

radio.startListening();
```

### 3.3.2 Lecture, écriture

La manière de procéder et les fonctions à appeler sont les mêmes que pour l'Arduino.

# 4 Protocole de niveau applicatif

Pour pouvoir synchroniser les horloges des deux cartes, nous avons créé un protocole simple. Au démarrage l'Arduino envoi un message avec `cmd = INIT` et se met en attente de la réponse de la Raspberry. Quand elle reçoit ce paquet la Rasperry répond avec son estampille temporelle dans `timestamp` et `cmd = INIT`. L'Arduino sauvegarde cette valeur et l'utilisera pour dater ses envois (en l'incrémentant). De ce fait, si la carte Arduino venait à s'éteindre, elle cherchera à se re-synchroniser au démarrage. Suite à cela, l'Arduino pourra envoyer ses valeurs de capteur avec `cmd = DATA` et les autres champs remplis en conséquence.