Projet PERI :
Dispositif météorologique

Rapport de projet

Paul MABILLOT & Adrien FERREIRA & Pierre MAHÉ

25 avril 2015

UPMC
PARIS UNIVERSITAS

Table des matières

1 Description du projet **3**

2 Montage **3**
2.1 Arduino 3
2.2 Raspberry Pi 4

3 Codage **5**
3.1 Structure de données 5
3.2 Arduino 6
3.2.1 Initialisation 6
3.2.2 Lecture 7
3.2.3 Écriture 7
3.3 Raspberry Pi 7
3.3.1 Initialisation 7
3.3.2 Lecture, écriture 7

4 Protocole de niveau applicatif **8**

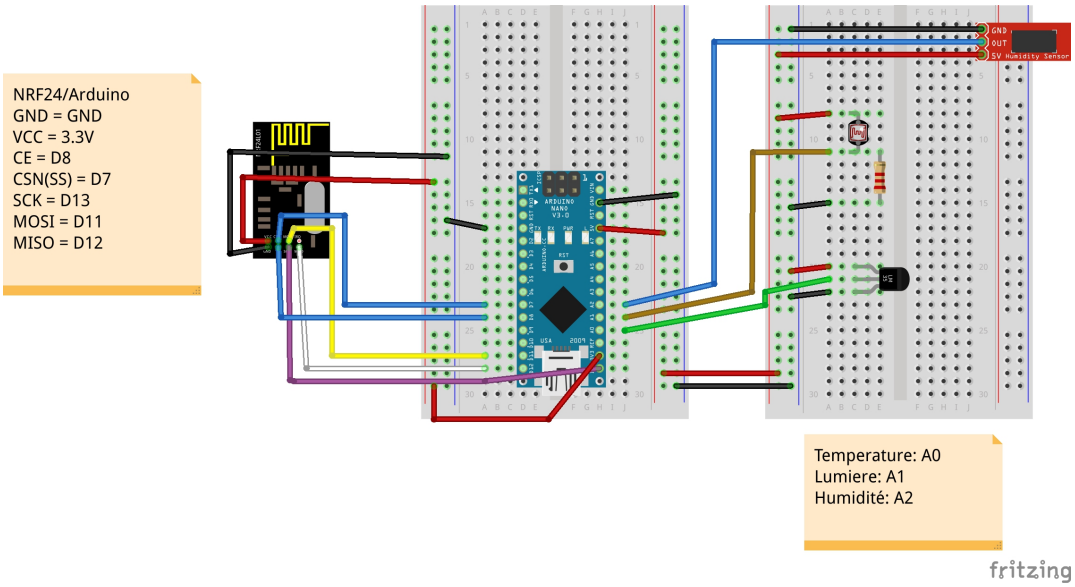
5 Serveur **8**
5.1 Installation 8
5.2 Mise en place 9
5.2.1 Création de l'index.php 9
5.2.2 Récupération de la valeur 10
5.2.3 Rafraîchissement automatique 10
5.2.4 La partie Graphique 11
5.2.5 Bonus 11

1 Description du projet

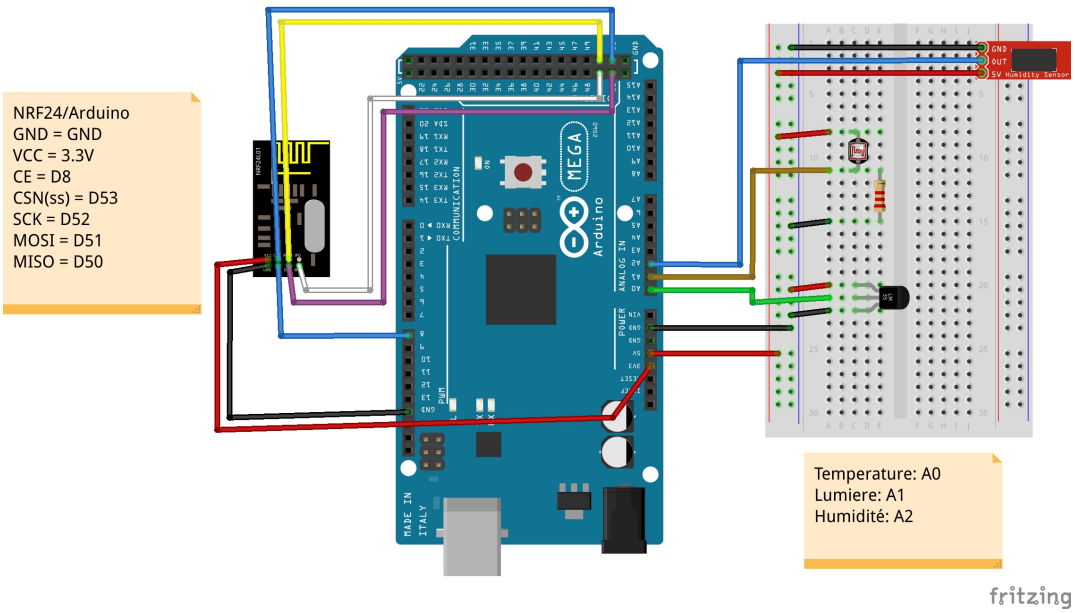
2 Montage

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.

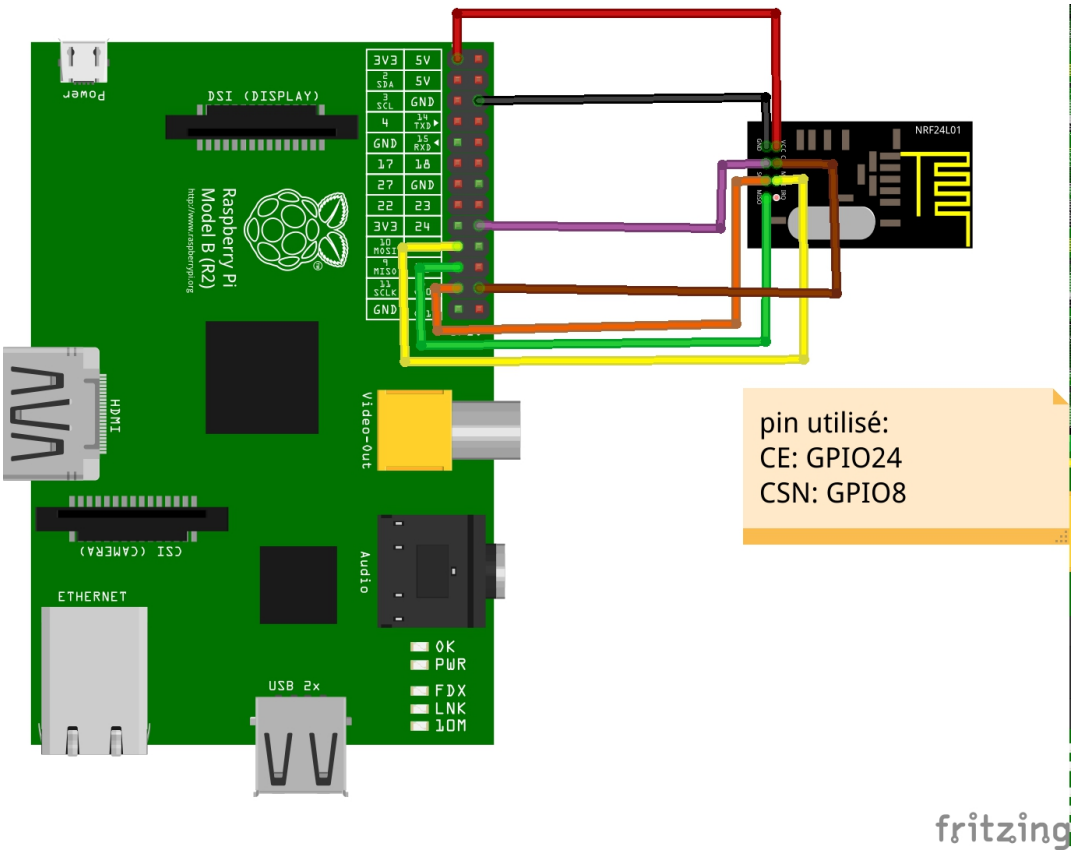


Pour l'arduino Mega le montage est un peu différent. Branchez le pin CSN(SS) au pin D53, le SCK au pin D52, le MOSI au pin D51 et le MISO au pin D50.



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.

5 Serveur

5.1 Installation

Dans cette partie, nous allons nous pencher sur code. Par un soucis de lisibilité nous n'allons pas commenter toute les fonctions, nous indiquerons autant que faire se peu les noms des fichiers et des fonctions, à vous de regarder dans les fichiers du projet le code exacte.

-Installation Apache2 et PHP

```
sudo apt-get install apache2 php5 libapache2-mod-php5
```

-Installation Mysql (inutile dans notre cas)

```
sudo apt-get install mysql-server php5-mysql
```

-Installation Mysql (inutile dans notre cas)

```
sudo apt-get install mysql-server php5-mysql
```

-Répertoire du serveur

```
/var/www/
```

Commandes Serveur :

```
$service apache2 stop : arreter serveur
$service apache2 start : demarrer serveur
$service apache2 restart : redemarrage serveur
```

5.2 Mise en place

5.2.1 Création de l'index.php

Dans un premier temps nous allons créer la page d'accueil de notre station météo. On souhaite y afficher les informations en temps réel ainsi qu'un graphique regroupant les informations des derniers jours.

Dans la section *"station"* nous afficherons les données
Dans la section *"recap"* nous afficherons le graphique

```
<!DOCTYPE html>
    <html>
      <head>
      <meta name="description" content="Bienvenue sur le site de
    meteorologique SAR" />
      <meta charset="utf-8" />
      <link rel="stylesheet" href="css/style.css" />
      <title>Site meteorologique</title>
      </head>

      <body>
      <section id="meteo">
        <h1>Station Meteo</h1>

        <section id="station">
        </section>

        <section id="recap">
        </section>
      </section>
      </body>
    </html>
```

Votre raspbery ne sera pas forcement connecté à internet, il est donc important d'importer les librairies pour assurer le bon fonctionnement de votre station météo même hors ligne.
C'est pour cela que nous avons choisi la bibliothèque jquery et jqPlot.
Le code suivant est a ajouter dans le *<head>*.

La librairie jquery :

```
<script src="js/jquery-1.11.2.min.js"></script>
```

La libraire de gestion de graphique jqPlot :

```
  <script type="text/javascript" src="js/jqPlot/jquery.jqplot.min.js"
    ></script>
```

```
    <script type="text/javascript" src="js/jqPlot/plugins/jqplot.canvasTextRenderer.min.js"></script>
<script type="text/javascript" src="js/jqPlot/plugins/jqplot.canvasAxisLabelRenderer.min.js"></script>
```

5.2.2 Récupération de la valeur

Les données envoyés par votre arduino sur votre raspberry (voir plus haut) sont sauvegardés dans le fichier xml/fichier.xml de votre serveur apache. Nous allons écrire une fonction PHP permettant de récupérer la dernière valeur ajoutée dans ce fichier.

```
function getLastValue($path){
    $fp = fopen($path, "r");

    if (flock($fp, LOCK_SH)) {
      $xml = simplexml_load_file($path);
      flock($fp, LOCK_UN);
      $taille = sizeof($xml);
      $last = $xml->timestamp[$taille-1];
    }
    return $last['valeur'].' '.$last->temperature.' '.$last->humidite.' '.$last->luminosite;
}
```

La fonction getLastValue, qui prend en paramètre le chemin vers votre fichier XML, s'occupe d'ouvrir ce fichier en lecture seul.
On verrou est alors placé pour éviter une écriture durant la lecture.
La fonction `simplexml_load_file` s'occupe de parser un fichier XML et retourne un objet facilement navigable.
Une fois l'objet obtenu on relâche le verrou.
On s'occupe de récupérer le nombre total d'élément dans le fichier XML afin de récupérer le dernier timestamp que l'on stock.
La fonction retourne alors le timestamp, la température, l'humidité et la luminosité du dernier envoi.

5.2.3 Rafraîchissement automatique

Afin de rendre notre page dynamique nous souhaitons que les derniers valeurs envoyés par nos sondes soient directement affichés. Pour cela nous allons utilisé notre fonction PHP écrite précédemment.
Malheureusement comme le PHP est géré coté serveur il est impossible de réaliser cela sans l'aide de javascript.

Pour cela, il faut utiliser le fichier `getLast.php` qui va lui même appeler le fichier `loadData.js`, Qui va appeler toute les 1000ms une fonction qui va retourner les nouvelles valeurs du timestamp, de la temperatuyre, de l'humidité et de la luminosité.
Il ne bous reste plus qu'à traiter ces données.
Pour la luminosité on avons decider de faire l'approximation suivant : si il y a beaucoup de lumière alors on peut considérer qu'il fait beau si au contraire il y a peu de lumière alors nous pouvons dire que le temps est couvert. De plus nous avons ajouter un balance jour/nuit (cf. fonction `traitementLumino`). Le traitement permet de charger des différentes images dans le html, selon le temps qu'il fait.

5.2.4 La partie Graphique

Nous afficherons les données des 3 derniers jours sur notre graphique lors du chargement de la page. Pour cela nous savons que chaque information est reçu toute les 1s, il faut donc traiter 86400 événements par jour, soit dans notre cas 259200 événements (60*60*24*3) (cf. `loadGraphe.php`).
Nous ne souhaitons pas réafficher le graphique après chaque réception de donnée, cela serai trop lourd pour notre serveur.
Contrairement aux données, nous recalculons le graphique qu'au moment du rafraîchissement de la page par le navigateur web.
Nous avons utiliser un fonction javascript pour afficher le graphique.

```
function getAllValues()
```

Comme précédemment la fonction ouvre le fichier XML, positionne le verrou, mais ici elle va sauvegarder uniquement les données des 259200 événements disponibles et retourner un tableau.
Afin de fournir ces données à la bibliothèque *jqPLot*, voir le fichier `loadGraph.php`.

```
<?php
  include("gestionFile.php");
  /* Recuperer les timestamp dans le fichier XML */
  $res = getAllValues("xml/fichier.xml");
  $tabjs = php2js($res);
?>
```

Le code PHP récupères les données et les mets en forme pour le javascript à l'aide de la fonction *php2js*.

5.2.5 Bonus

Il est possible de faire de la mise ne page en ajoutant des feuilles de style.