N°16 | JANVIER - FÉVRIER 2017

HACKABLE MAGAZINE

DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

France MÉTRO. : 7,90 € – CH : 13 CHF – BEL/LUX/PORT.CONT : 8,90 € – DOM/TOM : 8,50 € – CAN : 14 $ CAD

COURANT / BOIS

Gravez les étonnantes figures de Lichtenberg sur du bois avec de l'électricité...

p. 58

CARTES / NFC

Faites peur à vos amis en lisant leur carte bancaire sans contact depuis votre Raspberry Pi

p. 50

TERMINAL / PI

Ne vous déconnectez plus jamais de la ligne de commandes de votre Raspberry Pi avec Screen

p. 70

WINDOWS / PI

Utilisez les applications graphiques de votre Raspberry Pi depuis Linux, MacOS et Windows

p. 78

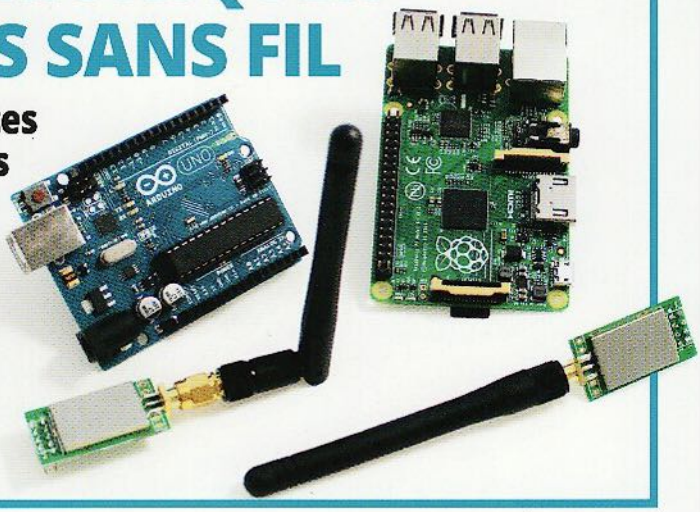
Arduino, Raspberry Pi, modules nRF24L01...

FAITES COMMUNIQUER VOS PROJETS SANS FIL

et sur de longues distances pour une poignée d'euros

p. 18

- Choisissez et découvrez les modules nRF24L01
- Établissez une communication entre deux Arduino
- Faites communiquer votre Pi et vos montages



RADIO / SDR

Explorez les ondes et la radio logicielle, en réception et en émission grâce au HackRF One

p. 36

BLUETOOTH / PI

Dialoguez en Bluetooth entre votre Raspberry Pi et vos montages Arduino grâce à Python

p. 88

L 19338 - 16 - F: 7,90 € - RD

ÉDITO

La Raspberry Pi est-elle l'avenir de Linux ?

Voilà une question intéressante. Si elle était posée correctement ce serait encore mieux. Déjà, on ne dit pas « Linux », mais « GNU/Linux », un système reposant sur les outils GNU et utilisant un noyau Linux (na !). Ensuite, il faut définir ce que veut dire un « avenir » pour un tel système. Le Wiktionnaire nous dit que c'est le « Futur, ce qui va arriver ». Là, la question initiale ne veut presque plus rien dire...

Voyons cela autrement. Le système GNU/Linux est depuis un certain temps maintenant massivement utilisé sur Internet. Quand vous visitez un site web par exemple, il est presque certain que vous avez affaire à une machine GNU/Linux d'une façon ou d'une autre. Concernant les PC et postes de travail, les choses n'en sont pas à ce stade et, il faut se rendre à l'évidence, ne le seront sans doute jamais...

Mais la Raspberry Pi a radicalement changé la donne. GNU/Linux est LE système pour utiliser une Pi et parmi vous qui lisez ceci, beaucoup n'ont sans doute jamais essayé une distribution GNU/Linux sur PC ou n'ont peut-être tout simplement pas été convaincus par leur première expérience. Vous êtes donc nombreux à faire connaissance avec ce système grâce à cette adorable framboise...

Donc non, la Raspberry Pi n'est pas l'avenir de Linux. La Raspberry Pi **EST** GNU/Linux, pour nombre d'utilisateurs.

Comme vous le savez peut-être, *Hackable* n'est pas le seul magazine que nous éditons et nous avons dans notre besace *Linux Pratique* qui depuis 1999 aide ses lecteurs à découvrir et apprivoiser GNU/Linux. Or, faire ses premiers pas dans ce domaine aujourd'hui est aussi très souvent faire ses premiers pas avec sa Pi. C'est pourquoi la publication sœur de *Hackable* s'adapte pour son numéro 99 et opte pour une nouvelle formule intégrant un cahier Raspberry Pi.

L'idée n'est bien entendu pas de couvrir les mêmes sujets ici et dans *Linux Pratique*, ni même d'avoir une approche similaire, mais tout simplement de faire bénéficier l'ensemble des utilisateurs GNU/Linux, quelle que soit leur plateforme, de l'expérience des auteurs contribuant au magazine depuis des années. GNU/Linux est bien plus « grand » que Raspberry Pi et bon nombre de connaissances générales, d'astuces, de pratiques, de logiciels, d'outils... sont tout simplement universels à tous les GNU/Linux.

Hackable ne peut, et ne doit d'ailleurs pas, couvrir ces généralités, mais se concentrer sur la bidouille et l'exploration électronique/matérielle. Cette nouvelle formule de *Linux Pratique*, quant à elle, pourra cependant vous intéresser et, je l'espère, vous plaire, si comme bien d'autres vous venez du monde Windows ou macOS et éprouvez tantôt quelques difficultés dans le dressage de vos manchots. Mais je préfère encore vous laisser juge, jetez un œil non loin de l'endroit où vous avez attrapé ce 16ème numéro. Nul doute que vous y trouverez aussi un *Linux Pratique* à feuilleter.

Notez au passage que cet édito n'est absolument pas sponsorisé par *Linux Pratique*. Je n'ai d'ailleurs rien reçu pour m'inciter à l'écrire, rien... même pas un muffin, du pain d'épice, un cookie ou un croissant lors d'une réunion de rédactions... malgré le nombre de conseils avisés donnés à la rédactrice en chef... Mon incorruptibilité me perdra, contrairement à ma gourmandise :)

Denis Bodor

Hackable Magazine
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIÈRE GRISE

10, Place de la Cathédrale - 68000 Colmar
Tél. : 03 67 10 00 20 – Fax : 03 67 10 00 21
E-mail : lecteurs@hackable.fr
Service commercial : cial@ed-diamond.com
Sites : http://www.hackable.fr/
http://www.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Kathrin Scali
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20

Impression : pva, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
Service des ventes : Abomarque : 09 53 15 21 77
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : 2427-4631
Commission paritaire : K92470
Périodicité : bimestriel
Prix de vente : 7,90 €

FSC www.fsc.org MIXTE Papier issu de sources responsables FSC® C015136

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans Hackable Magazine est interdite sans accord écrit de la société Les Éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à Hackable Magazine, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire. Toutes les marques citées dans ce numéro sont déposées par leur propriétaire respectif. Tous les logos représentés dans le magazine sont la propriété de leur ayant droit respectif.

Suivez-nous sur Twitter
@hackablemag

À PROPOS DE HACKABLE...

HACKS, HACKERS & HACKABLE

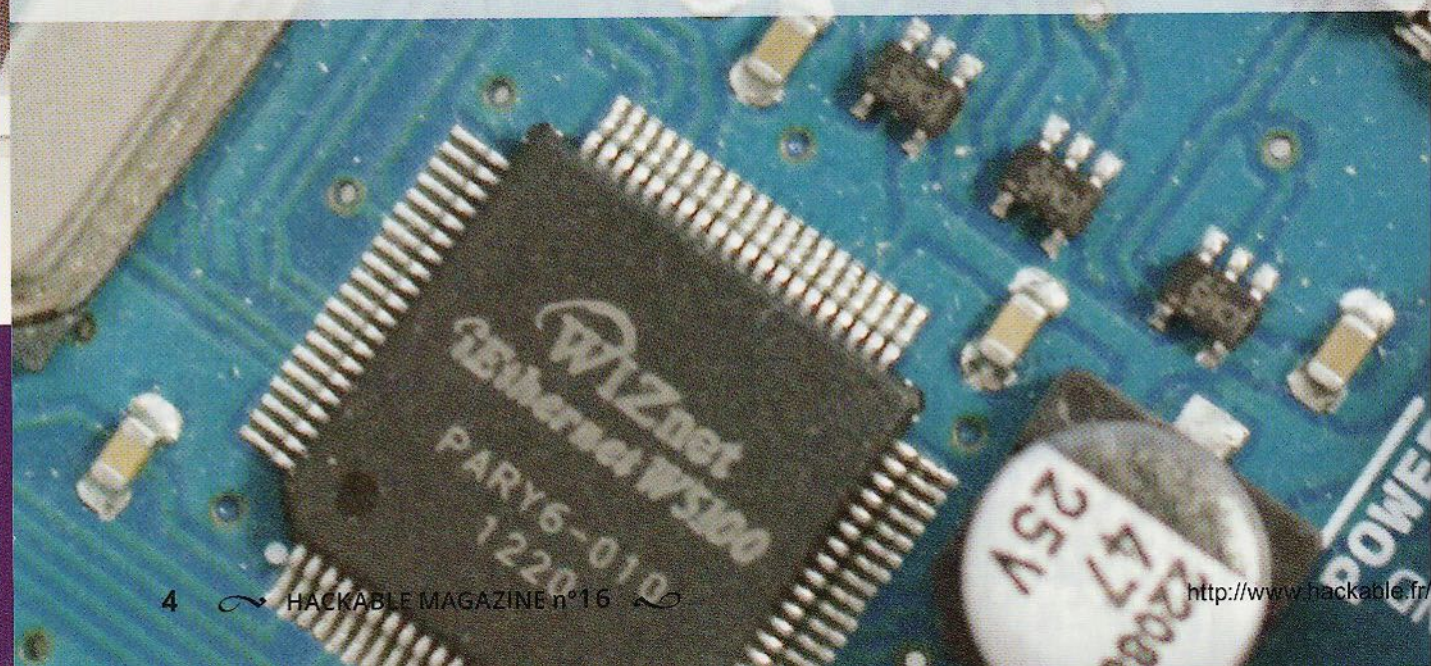
Ce magazine ne traite pas de piratage. Un **hack** est une solution rapide et bricolée pour régler un problème, tantôt élégante, tantôt brouillonne, mais systématiquement créative. Les personnes utilisant ce type de techniques sont appelées **hackers**, quel que soit le domaine technologique. C'est un abus de langage médiatisé que de confondre « pirate informatique » et « hacker ». Le nom de ce magazine a été choisi pour refléter cette notion de **bidouillage créatif** sur la base d'un terme utilisé dans sa définition légitime, véritable et historique.

SOMMAIRE

UTILISEZ UNE CARTE SD AVEC VOTRE ARDUINO

Denis Bodor

Le microcontrôleur d'une carte Arduino dispose de trois types de mémoire, la SRAM, la flash et l'EEPROM. Chacune a son utilité spécifique, mais les trois sont en quantité réduite. L'utilisation d'une mémoire externe (EEPROM) permet de facilement disposer d'un espace de stockage pour des données, mais là encore, on parle de quelques dizaines de kilooctets. Que faire si on a besoin de bien plus d'espace ? C'est simple, il suffit de faire exactement la même chose que pour tous les produits électroniques du marché : utiliser une carte SD ou microSD !



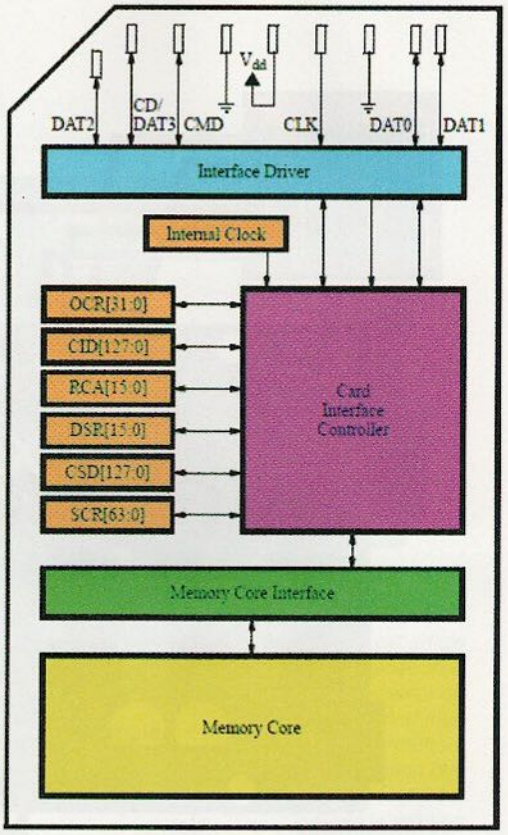
Se retrouver à l'étroit lorsqu'un projet évolue est monnaie courante, c'est là tout le principe de l'écriture de croquis Arduino et de la programmation de microcontrôleurs en général. Il faut penser, dès le départ, que nous ne disposons pas d'une masse énorme de mémoire comme c'est le cas, par exemple avec un PC ou une Raspberry Pi. Une technique généralement utilisée consiste donc à judicieusement choisir ce qui est stocké où, ou en d'autres termes, faire usage de façon efficace des différents types de mémoire, en fonction des types d'informations qu'on utilise.

Dans ce sens, le plus important est le code qui ne peut être exécuté qu'en flash. Les données, quant à elles, peuvent être migrées à un autre endroit, récupérées en cas de besoin et stockées en mémoire vive (SRAM) pour être manipulées avant d'être à nouveau proprement rangées. Nous avons vu, par exemple, dans le précédent numéro, qu'il était possible d'attacher une mémoire EEPROM à une carte Arduino et d'y stocker ainsi des mesures de capteurs. C'est une solution robuste et fiable, mais l'espace disponible reste limité.

Pour passer à une tout autre catégorie en termes de volume, la solution consiste à utiliser une carte SD ou microSD. Là, il n'est plus question que de quelques kilos ou mégaoctets, mais de gigaoctets ! Avec ce genre de volume, l'organisation de la mémoire prend également une tout autre tournure et on utilise alors des notions et des structures qui sont celles propres aux disques durs : partitions, système de fichiers, répertoires, fichiers, etc.

1. LES CARTES SD ET MMC

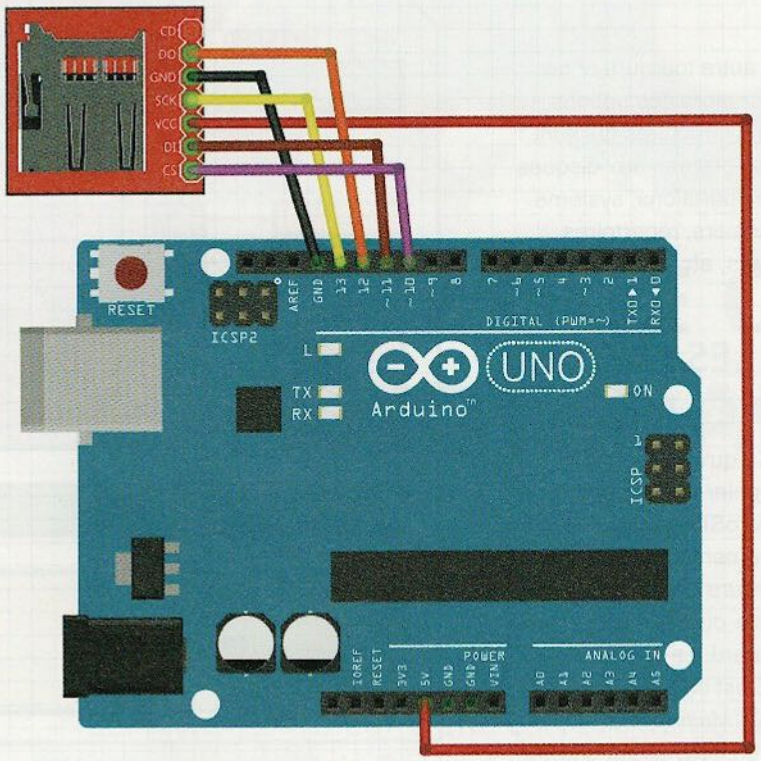
Ce qu'il est courant d'appeler « carte SD » ou « microSD » répond techniquement à la désignation « *Secure Digital Memory Card* » ou SDC, qui est un standard. La technologie SDC est une évolution de MMC (*Multi Media Card*) tout en conservant une compatibilité. Normalement, un équipement prévu pour lire et écrire sur des SDC fera exactement de même avec les anciennes MMC.



Voici l'anatomie interne d'une carte SD avec en jaune la mémoire flash. Contrairement à ce qu'on pourrait penser, c'est loin qu'être un simple support de stockage. Techniquement parlant, c'est même un système complet avec processeur, périphériques, interface et un logiciel pour faire fonctionner le tout.

Une telle carte se compose d'une mémoire flash et d'un contrôleur intégré. Vous n'avez donc pas directement accès physiquement à la mémoire qui est gérée pour vous. Cette gestion, en plus d'offrir un moyen d'accès simplifié, se charge également de tous les problèmes liés à l'électronique de la carte et à la nature même de la mémoire utilisée. Une carte SD possède, par exemple, généralement plus d'espace de stockage qu'il n'y paraît pour compenser les défauts qui peuvent être présents sur le support.

Vous ne le voyez pas, mais la mémoire des cartes SD et d'autres périphériques comme les clés USB de stockage, n'est pas parfaite. Elle possède des défauts et s'use, mais les constructeurs ont mis en place une vaste gamme de techniques permettant de garantir la fiabilité, de protéger les données et de pallier à ce type de problèmes. Ceci se fait via un *Flash Translation Layer* ou FTL, une couche se trouvant entre



L'ajout d'un lecteur de carte SD ou microSD n'est pas quelque chose de très complexe. Ceci se résume à choisir le bon module, de qualité, et à une poignée de câbles. Tout le reste est une question de bibliothèque et de croquis.

la mémoire à proprement parler et l'interface de connexion (Sata, USB, SPI, etc.). Une simple carte SD intègre donc un microcontrôleur et tout un système, dont le logiciel peut parfois être mis à jour (firmware). Il existe même des firwmares alternatifs pour des usages... on va dire... « spécifiques ». Le choix de la mémoire flash, sa quantité par rapport à l'espace de stockage annoncé et les performances du logiciel, impactent directement la qualité du produit et son prix. Autrement dit : toutes les cartes SD ne se valent pas.

Du point de vue qui nous intéresse ici cependant, la technologie embarquée ne nous concerne pas vraiment puisque nous nous contentons d'utiliser ces cartes et de communiquer avec elles. Cette communication peut être faite selon deux modes différents : le mode SD et le mode SPI. Par défaut, c'est le mode SD qui est utilisé, mais il est possible de demander à une carte de basculer sur le mode SPI en spécifiant l'état de certaines broches et en lui envoyant un message spécifique. La carte cesse alors d'utiliser le mode SD et bascule en SPI.

Si le terme SPI vous dit quelque chose, c'est tout simplement, car nous l'avons déjà largement utilisé dans les différents articles du magazine. La communication SPI est courante et permet aux cartes Arduino, via une connexion simple à 4 fils, de dialoguer avec bon nombre de composants et de modules. On utilise pour cela les connexions suivantes (sur une Arduino UNO) :

- 13 : SCLK ou CLK pour *CLocK* (horloge) est le signal qui rythme la communication ;
- 11 : MOSI pour *Master Out Slave In* véhicule les données du maître (Arduino) vers l'esclave (ici la carte SD) ;
- 12 : MISO pour *Master In Slave Out* transporte les informations dans le sens opposé ;
- 10 : SS ou CS pour *Slave Select* ou *Chip Select* permet à celui qui contrôle la communication (le maître, l'Arduino) de demander à l'esclave d'être attentif à la communication. Le standard SPI permet de connecter les lignes SCLK, MISO et MISO à plusieurs esclaves et CS est utilisé pour les désigner, les adresser. On utilise généralement la broche 10 (car à côté des autres), mais ceci n'est en rien imposé. C'est vous qui choisissez la broche utilisée pour CS sur la carte Arduino.
- alimentation et masse.

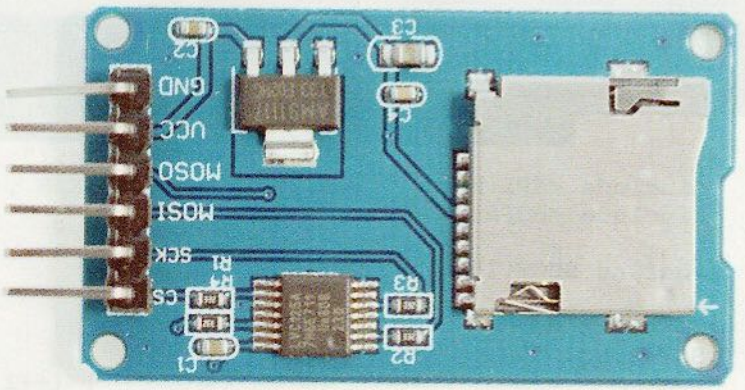
Ce dernier point nous amène à une caractéristique importante des cartes SD et microSD : la tension. En effet, la tension spécifiée dans les documentations

de l'ensemble des constructeurs précise une alimentation entre 2,7V et 3,6V. Cette même tension est également utilisée pour déterminer les niveaux logiques (ce qui est un 1 ou un 0).

La grande majorité des cartes Arduino travaille en 0/5V et non en 0/3,3V, et il est important de ne pas connecter directement les broches d'une carte SD à un Arduino. Vous trouverez en ligne une vaste collection de modules (*breakout board*) à des prix divers et variés. Certains sont de simples supports sans le moindre composant électronique, d'autres ajoutent un régulateur permettant d'alimenter la carte SD en 3,3V à partir de 5V et d'autres enfin embarquent une adaptation de tension complète assurant un maximum de protection. C'est bien entendu ce type de modules qu'il faudra choisir.

J'ai acquis les miens sur eBay, auprès d'un vendeur appelé *bobowaytoway* sous la désignation « *Micro SD Storage Board Reader TF Card Memory Shield Module SPI Pour Arduino* », pour moins de trois euros pièce (livraison incluse depuis l'Angleterre). Ce modèle, également en vente auprès d'autres vendeurs, intègre un support microSD (à « clic »), un régulateur et un adaptateur de tension facilement reconnaissable par son boîtier de 14 broches soudé sur le circuit imprimé.

La connexion du module lui-même à la carte Arduino suivra tout simplement celle décrite précédemment. Notez toutefois que certains modules disposent d'une broche supplémentaire notée CD pour *Card Detect*. Il s'agit d'un simple contact, par défaut à la masse ou en l'air, qui change d'état lorsqu'une carte est présente dans l'emplacement. Ceci permet, par exemple, de faire réagir votre croquis Arduino en cas de changement de carte.



Le module utilisé dans cet article dispose d'un régulateur de tension et d'un circuit permettant l'adaptation des niveaux logiques. C'est la solution à privilégier afin de s'assurer d'un fonctionnement optimal avec une vaste gamme de cartes (Arduino ou non). Notez l'erreur sur les libellés des broches : MOSO en lieu et place de MISO.

2. UTILISER LA BONNE BIBLIOTHÈQUE ET PREMIER ESSAI

L'environnement de programmation Arduino est livré avec la bibliothèque **SD**. Celle-ci repose sur un code écrit par William Greiman, mais n'offre qu'un nombre limité de fonctionnalités par rapport à la version à jour de William : **SdFat**. **SD** est initialement installable pour permettre l'utilisation de microSD avec le shield Ethernet.

En faisant un petit tour dans le gestionnaire de bibliothèques, vous découvrirez que **SdFat** n'est pas là. Il faudra vous rendre sur **https://github.com/greiman/SdFat** afin de télécharger le répertoire **SdFat** et son contenu pour le copier dans le répertoire **libraries** de votre carnet de croquis. Dès le prochain démarrage de l'environnement Arduino, vous pourrez l'utiliser.

Pour nous assurer que tout est bien connecté, commençons pas collecter quelques informations sur la carte présente dans le module :

Fichier Édition Croquis Outils Aide

```
#include <SPI.h>
#include <SdFat.h>

#define SDCS 10

// objet sd
SdFat sd;

void setup() {
  uint32_t taille;
  cid_t cid;

  Serial.begin(115200);

  // indispensable initialisation
  Serial.println("Initialisation SD...");
  if (!sd.cardBegin(SDCS,SPI_HALF_SPEED)) {
    Serial.println("Erreur initialisation");
    return;
  }

  // Type de carte
  Serial.print("\nType de carte: ");
  switch (sd.card()->type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Inconnu");
  }

  // obtention de la taille de la carte
  taille = sd.card()->cardSize();
  if(!taille) {
    Serial.println("Erreur taille");
    return;
  }
  Serial.print("Taille physique: ");
  Serial.print((taille*512)/1024/1024);
  Serial.println(" Mo");
```

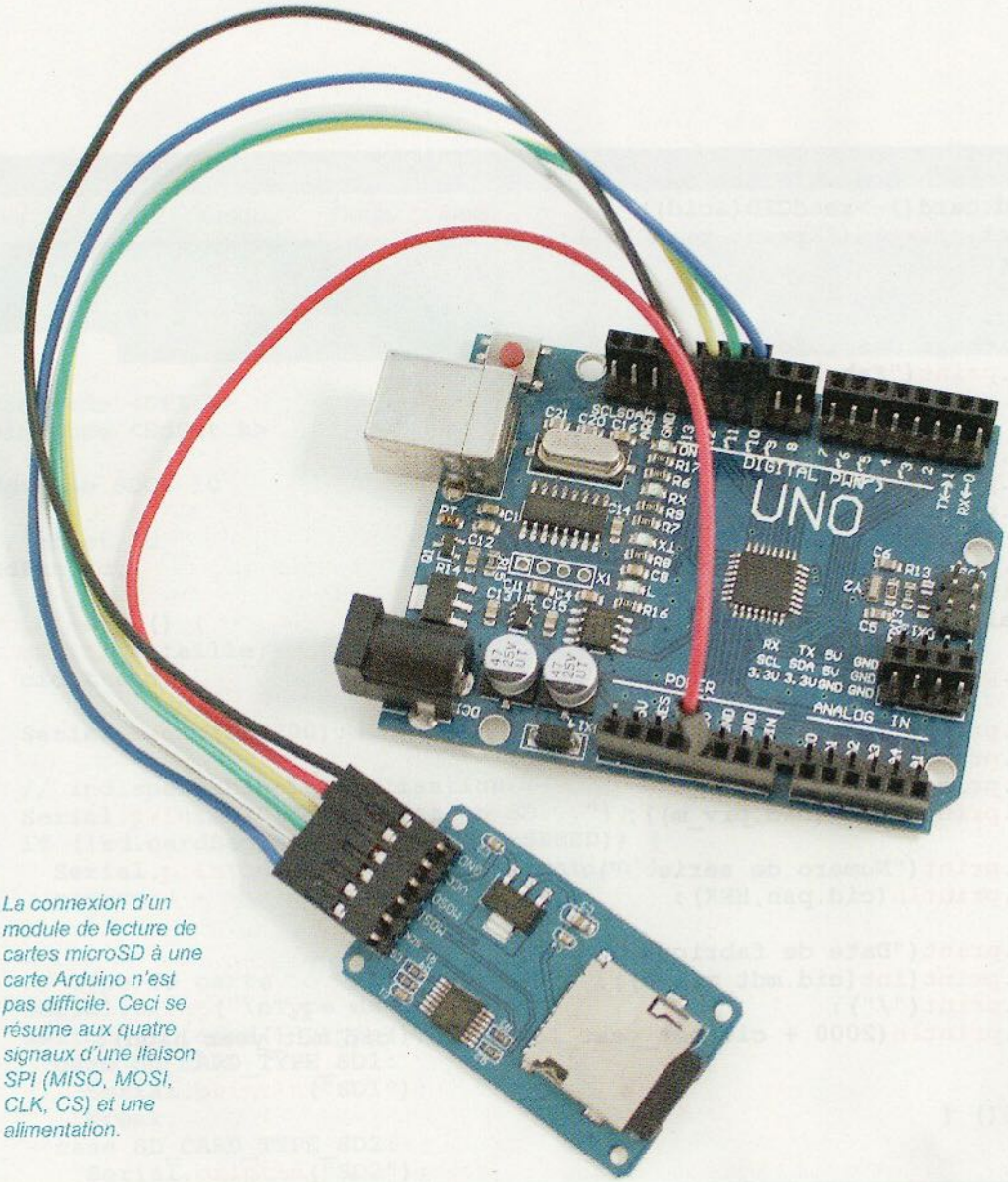
```
  // obtention des informations
  if (!sd.card()->readCID(&cid)) {
    Serial.println("Erreur readCID");
    return;
  }

  // affichage des informations
  Serial.print("Fabricant ID: ");
  Serial.println(int(cid.mid), HEX);

  Serial.print("OEM ID: ");
  Serial.print(cid.oid[0]);
  Serial.println(cid.oid[1]);

  Serial.print("Produit: ");
  for (uint8_t i = 0; i < 5; i++) {
    Serial.print(cid.pnm[i]);
  }
  Serial.println("");

  Serial.print("Version: ");
  Serial.print(int(cid.prv_n));
  Serial.print(".");
  Serial.println(int(cid.prv_m));

  Serial.print("Numero de serie: ");
  Serial.println(cid.psn,HEX);

  Serial.print("Date de fabrication: ");
  Serial.print(int(cid.mdt_month));
  Serial.print("/");
  Serial.println(2000 + cid.mdt_year_low + 10 * cid.mdt_year_high);
}

void loop() {
}
```

Arduino

Nous avons là plusieurs étapes dans le croquis.

La première consiste à initialiser la communication et obtenir un objet **sd** avec lequel travailler. Cette action, déclenchée avec la méthode **cardBegin** utilisée avec le numéro de broche pour CS et la vitesse de communication souhaitée, permet de débuter le dialogue. Le passage du mode SD au mode SPI est opéré pour nous et nous n'avons pas besoin de nous soucier des détails du protocole.

Nous nous informons ensuite sur le type de carte utilisée. Trois réponses sont possibles : SD1, SD2 et SDHC, correspondant à différentes déclinaisons des spécifications officielles. Notez que le SDXC (*Secure Digital eXtended Capacity*) n'est qu'un SDHC (*Secure Digital High Capacity*) pouvant aller jusqu'à 2 To de stockage.



La connexion d'un module de lecture de cartes microSD à une carte Arduino n'est pas difficile. Ceci se résume aux quatre signaux d'une liaison SPI (MISO, MOSI, CLK, CS) et une alimentation.

Nous récupérons ensuite, avec `sd.card()->cardSize()` la taille de la carte. Attention, ceci n'est ni la taille physique réelle et cachée de la mémoire flash dans le périphérique, ni la taille utilisable. C'est la taille matérielle apparente, en nombre de blocs disponibles, servant de base au support. Exactement comme la taille physique d'un disque dur. Ici, nous utilisons ce nombre de blocs, multiplié par la taille d'un bloc (512 octets) et divisé pour obtenir des mégaoctets (ou plutôt des mébioctets selon la désignation officielle puisque quelqu'un a eu la bonne idée de passer à la trappe des termes utilisés depuis l'aube de l'informatique et décrivant pourtant sa nature profonde : des 1 et des 0).

Enfin, on passe à quelque chose d'un peu plus intéressant en récupérant le contenu d'un registre très spécial de la carte. Celle-ci ne provient pas de la mémoire de stockage, mais d'un registre interne du circuit qui sert d'interface : la valeur de CID pour *Card IDentification*. Il s'agit de 16 octets nous fournissant des informations très spécifiques sur le produit. Le plus clair est encore de tout simplement voir ce qu'affiche ce croquis dans le moniteur série :

```
Initialisation SD...

Type de carte: SD2
Taille physique: 942 Mo
Fabricant ID: 2
OEM ID: TM
Produit: SA01G
Version: 0.3
Numero de serie: F00109C
Date de fabrication: 6/2010
```

Comme vous le voyez, nous récupérons tout un tas d'information intéressantes comme le fabricant (ici « TM » semble être *Toshiba Memory*) ou encore la date de fabrication. Si vous voulez savoir si votre carte est ancienne, voilà la réponse, celle-ci a plus de 6 ans. Ces informations, également récupérables par d'autres biais, sur une Pi par exemple (essayez `cat /sys/block/mmcblk0/device/date` par exemple), permettent également de détecter les produits douteux ou falsifiés qui contiennent souvent des informations incomplètes ou incohérentes.

Bien... C'est très joli tout ça, mais il est temps de passer aux choses sérieuses...

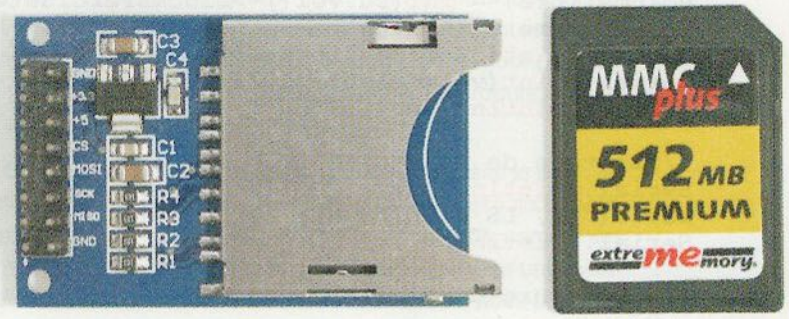
3. LIRE LE CONTENU D'UN FICHIER

Il existe une différence fondamentale entre accéder à une carte SD et lire un fichier. Contrairement à une mémoire EEPROM comme celle intégrée dans le microcontrôleur d'une carte Arduino, ou celle qu'il est possible de connecter en SPI ou en i2c, une carte SD contient généralement un **système de fichiers**. Comprenez bien que ceci est purement « organisationnel » et n'a rien à voir avec le fonctionnement même de la mémoire. Nous l'avons dit précédemment, la flash d'une carte SD est organisée en blocs. Ceux-ci sont structurés d'une certaine manière qui est invariable, car physiquement implémentée.

Pour simplifier les choses et permettre à l'utilisateur de ne pas avoir à courir après ces morceaux de données, on décide d'organiser tout cela en fournissant une vision abstraite sous forme de fichiers et de répertoires (qui n'est qu'une métaphore). Cette organisation qui fait le lien entre ce que vous voyez et l'emplacement effectif des données sur un support est un système de fichiers.

Windows ainsi que la quasi-totalité des appareils électroniques (smartphones, appareils photo, lecteurs MP3, etc.) utilisent le système de fichiers FAT (FAT16 ou FAT32 plus précisément). Il en existe d'autres : GNU/Linux utilise généralement ext4 ou ext3, Mac OS utilise HFS+ ou encore BSD utilise FFS ou UFS. Les versions actuelles de Windows utilisent généralement NTFS en lieu et place de l'historique FAT qui reste bien entendu utilisable.

Certains systèmes de fichiers sont plus complexes que d'autres et demandent souvent davantage de mémoire (vive). Si FAT, qui a vu le jour dans sa première version en 1977, est toujours là, ce n'est pas un hasard. Il est certes communément supporté, mais surtout il ne demande pas beaucoup de ressources pour être utilisé. Il n'y a donc rien d'étonnant au fait qu'il puisse être utilisé avec une carte Arduino.



Certains modules, qui n'en sont pas pour autant toujours moins chers, utilisent des résistances en diviseurs de tensions afin de permettre une connexion de la carte SD à un système utilisant 0/+5V. Notez sur cette photo une carte SD telle qu'il doit en rester des milliers au fond des tiroirs : 512 malheureux mégaoctets, parfaitement inutile pour un smartphone ou un APN, mais largement suffisants pour un projet Arduino.

Avant de poursuivre, voici le croquis que nous allons utiliser :

Fichier Édition Croquis Outils Aide

```
#include <SPI.h>
#include <SdFat.h>

SdFat sd;

#define SDCS 10
#define BUFFER_SIZE 100
uint8_t buf[BUFFER_SIZE];

void setup() {
  SdFile fichier;
  Serial.begin(115200);

  Serial.println("Initialisation SD...");
  if (!sd.begin(SDCS,SPI_HALF_SPEED)) {
    Serial.println("Erreur initialisation");
    return;
  }

  // Volume total
  // taille d'un cluster en nombre de blocs
  uint32_t volume = int(sd.vol()->blocksPerCluster());
  // ...fois le nombre de clusters
  volume *= sd.vol()->clusterCount();
  // ...fois la taille d'un bloc (toujours 512 octets)
  volume *= 512;
  Serial.print("Taille du volume: ");
  // en mégaoctets
  Serial.print(volume/1024/1024);
  Serial.println(" Mo");

  // Espace libre
  uint32_t volumelibre = sd.vol()->freeClusterCount();
  volumelibre *= int(sd.vol()->blocksPerCluster());
  volumelibre *= 512;
  Serial.print("Espace libre: ");
  Serial.print(volumelibre/1024/1024);
  Serial.println(" Mo");

  // Listage de la racine
  Serial.println(" ");
  sd.ls("/", LS_SIZE|LS_R);
  Serial.println(" ");

  // Ouverture du fichier
```

```
  if (!fichier.open(&sd, "toto.txt", O_READ)) {
    Serial.println("Erreur ouverture fichier");
    return;
  }

  // Lecture du fichier
  int n = fichier.read(buf, sizeof(buf));
  Serial.print(n);
  Serial.println(" octet(s) lu(s)");

  String myString = String((char *)buf);
  myString.trim();

  Serial.print("\"");
  Serial.print(myString);
  Serial.println("\"");

  /// Fermeture du fichier
  fichier.close();
}

void loop() {}
```

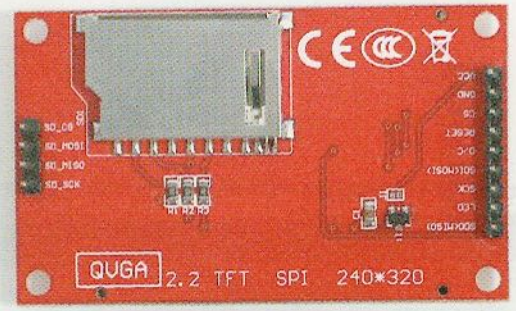
Arduino

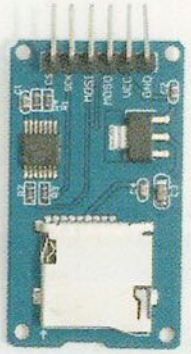
Nous avons précédemment mis en œuvre `sd.cardBegin()` pour initialiser la communication avec la carte SD, cependant, cette méthode, comme le précise la documentation dans le fichier `SdFat.h` n'existe qu'à des fins de diagnostics. Lorsqu'on utilise « réellement » la bibliothèque, il convient de choisir `sd.begin()` en lieu et place, avec les mêmes arguments.

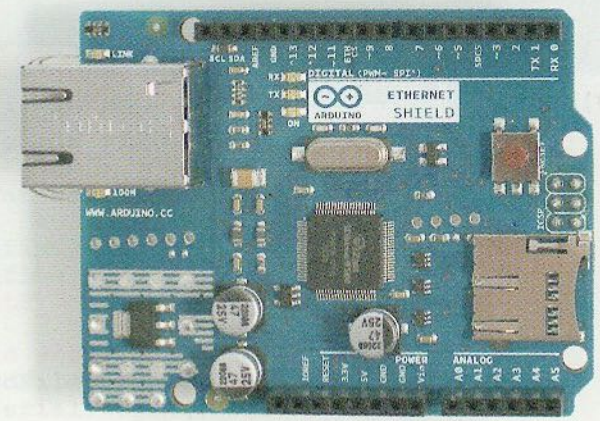
Ici, avant de nous attaquer à la lecture d'un fichier à proprement parler, nous commençons par obtenir quelques informations concernant le système de fichiers : sa taille totale qui n'est pas la taille physique de la carte (il peut y avoir plusieurs partitions avec plusieurs systèmes de fichiers, comme les microSD des Raspberry Pi), et l'espace encore disponible. Cette information nous est remontée sous la forme d'une quantité de *clusters* (ce qui doit vous rappeler quelque chose si vous avez connu `CHKDSK.EXE` et `SCANDISK.EXE`). Le cluster est l'unité de stockage d'un système FAT. Chaque cluster est constitué de plusieurs blocs d'une certaine taille en octets. Pour obtenir l'espace total ainsi que celui disponible, il faut donc prendre ce nombre de clusters, le multiplier par le nombre de blocs par cluster et par la taille d'un bloc. Ceci nous donne une quantité en octets qu'on fait ensuite s'afficher en Mo (ou Mio si vous préférez).

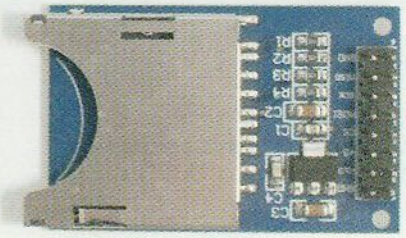
Le fait d'utiliser `sd.begin()` et non `sd.cardBegin()` nous donne directement accès au système de fichiers FAT s'il est présent. De ce fait, certaines méthodes très pratiques sont utilisables, comme `ls()` permettant de rapidement lister les fichiers. Ceci cependant est davantage un outil pour prendre en main la bibliothèque ou pour peaufiner un croquis, qu'autre chose.

Ce qui nous amène à la partie qui nous intéresse le plus : la lecture d'un fichier. Notre cible est ici un fichier `toto.txt` contenant une simple ligne de texte. Nous commençons, comme sur n'importe quel système, par ouvrir le fichier et vérifier une éventuelle erreur à cette étape. Si ce n'est pas le cas, nous pouvons enchaîner sur la lecture d'une masse de données.









Voici une petite collection de « lecteurs » de différentes origines et qualités. En rouge, un écran TFT équipé d'un emplacement SD (3,3V uniquement), à sa droite notre module préféré, en bas à gauche le shield Ethernet Arduino et son emplacement microSD et enfin, un module SD d'entrée de gamme.

La méthode `read()` sur l'objet qui représente le fichier (ici `fichier`) prend en argument une zone de mémoire (un tableau) et une taille indiquant la quantité d'octets que nous souhaitons lire. La méthode va alors déclencher la lecture du fichier et copier une certaine quantité de données dans notre tableau, et nous retourne le nombre d'octets effectivement lus. Si une valeur inférieure à la taille demandée est retournée, c'est que la lecture est arrivée en fin de fichier, si la valeur est négative (typiquement -1) c'est qu'une erreur est survenue.

Ici, `buf` est un tableau de 100 `uint8_t` (valeur sur 8 bits non signée, ou en d'autres termes des octets). Notre fichier contient bien moins de données que cela et l'appel nous retourne 14. Ce n'est pas grave, en dehors d'un beau gâchis de mémoire, ce qui est important est le fait de disposer d'un tableau suffisamment grand pour y mettre la quantité d'octets **souhaitée**. Dans le cas contraire, nous aurions un débordement de tampon (*buffer overflow*), synonyme de gros problèmes par la suite...

Nous disposons donc à ce stade d'un tableau, contenant notre texte et beaucoup, beaucoup d'octets `0`. Ça tombe bien, un `0` est ce qui marque la fin d'une chaîne en C/C++. Pour obtenir quelque chose de plus facile à utiliser, nous faisons de ce texte une chaîne (`String`), utilisons la méthode `trim()` pour supprimer les *blancs* avant et après la chaîne, puis l'envoyons sur le moniteur série. Enfin, nous n'oublions pas de fermer le fichier.

L'exécution de ce croquis nous donne quelque chose comme :

```
Initialisation SD...
Taille du volume: 941 Mo
Espace libre: 876 Mo

        14 toto.txt
         0 mon_rep/
  40996563 doomsday_1.10.3.dmg
   2217925 lm_hs27_apn.pdf
  22737285 hexen_1_1.zip

14 octet(s) lu(s)
"Coucou Monde!"
```

Il existe d'autres méthodes pour lire des données d'un fichier. Celle-ci est celle qu'on pourrait qualifier de plus « brute ». Une autre solution, parfois plus pratique, consiste à utiliser **fgets()** permettant de récupérer des lignes de texte (terminées par **\n** ou **\r\n**) à chaque appel :

```
int n = fichier.fgets(buf, sizeof(buf));
Serial.print(n);
Serial.println(" octet(s) lu(s)");

String myString = String((char *)buf);
myString.trim();

Serial.print("\"");
Serial.print(myString);
Serial.println("\"");
```

Ici, avec un fichier composé de plusieurs lignes, seule la première sera récupérée et affichée. La position de la prochaine lecture dans le fichier sera automatiquement celle correspondant à la fin de la précédente, et ainsi de suite jusqu'à arriver à la fin du fichier. Pour revenir au début du fichier, sans le fermer et le rouvrir, il faudra utiliser la méthode **rewind()** sur **fichier** pour le « rembobiner ». Il est amusant de constater que la terminologie issue de vieilles technologies est toujours présente (ici la lecture de bandes magnétiques).

4. CRÉER UN FICHIER ET Y ÉCRIRE

Lire les données d'une carte ou d'un fichier est intéressant mais, en pratique, l'écriture sera sans doute plus utile. Je pense naturellement à d'éventuels sondes et capteurs comme le montage que j'ai détaillé dans le précédent numéro, et utilisant une EEPROM externe.

Étonnamment, l'écriture d'un croquis pour ce genre d'usage est bien plus simple que la lecture. La bibliothèque met en effet à disposition les méthodes **print()** et **println()** telles qu'elles existent, par exemple, pour communiquer avec le moniteur série. Il suffit alors d'ouvrir le fichier en écriture et d'utiliser ces méthodes :

```
#include <SPI.h>
#include <SdFat.h>

SdFat sd;

#define SDCS 10
#define BUFFER_SIZE 100
uint8_t buf[BUFFER_SIZE];

void setup() {
  SdFile fichier;
  Serial.begin(115200);

  Serial.println("Initialisation SD...");
  if (!sd.begin(SDCS,SPI_HALF_SPEED)) {
    Serial.println("Erreur initialisation");
    return;
  }
```

```
  // Listage de la racine
  sd.ls("/", LS_SIZE|LS_R);

  // Ouverture du fichier
  if (!fichier.open(&sd, "essai.txt", O_RDWR | O_CREAT | O_AT_END )) {
    Serial.println("Erreur ouverture fichier");
    return;
  }

  fichier.println("Coucou ! ceci est un test.");
  fichier.sync();

  /// Fermeture du fichier
  fichier.close();

  // Listage de la racine
  Serial.println(" ");
  sd.ls("/", LS_SIZE|LS_R);
}

void loop() {}
```

Inutile ici d'utiliser un tableau pour quelque chose d'aussi simple. Enregistrer ainsi des données de type « journal d'activité » comme des mesures de capteurs est un jeu d'enfant. Il est, bien entendu, possible de composer les données à inscrire via un tableau et des fonctions comme **snprintf()** tel que nous l'avions fait avec l'afficheur température/humidité dans le numéro 14, mais la solution présentée ici est bien plus accessible pour ce type d'usage. On gardera les *jeux* avec des chaînes, des tableaux et des pointeurs pour des usages plus spécifiques.

La différence concernant l'accès aux fichiers en lecture ou en écriture réside dans le troisième argument de la fonction **open()**, le drapeau ou *flag* en anglais. Nous avons précédemment utilisé **O_READ** pour la lecture seule et ici une combinaison utilisant plusieurs drapeaux réunis via un *OU* logique : **O_RDWR**, **O_CREAT** et **O_AT_END** (pouvant être replacé ici par une macro, **FILE_WRITE** (définie dans **ArduinoFiles.h**)).

Les drapeaux utilisables, définis dans **FatApiConstants.h** sont :

- **O_READ** ou **O_RDONLY** : ouverture en lecture ;
- **O_WRITE** ou **O_WRONLY** : ouverture en écriture ;
- **O_RDWR** : ouverture en lecture et écriture (il s'agit en réalité d'un **O_READ | O_WRITE**) ;
- **O_AT_END** : se positionne automatiquement à la fin du fichier après l'ouverture, parfait pour un ajout ;
- **O_CREAT** : créé automatiquement le fichier si celui-ci n'existe pas encore. Si le fichier existe, ce drapeau n'a aucun effet, sauf si on indique également **O_EXCL**, auquel cas une erreur sera provoquée. Ceci permet de créer un fichier s'il n'existe pas, mais de ne pas toucher aux données existantes dans le cas contraire ;
- **O_SYNC** : appelle automatiquement la fonction **sync()** sur le fichier après chaque écriture. Ceci permet de s'assurer que les données soient effectivement écrites immédiatement dans le fichier. En effet, afin de garantir de bonnes performances, l'écriture effective peut être différée jusqu'à la fermeture du fichier. Ceci permet de s'assurer du contraire, au détriment

des performances. Il n'est pas recommandé d'utiliser ce drapeau si votre croquis écrit dans un fichier octet par octet ;

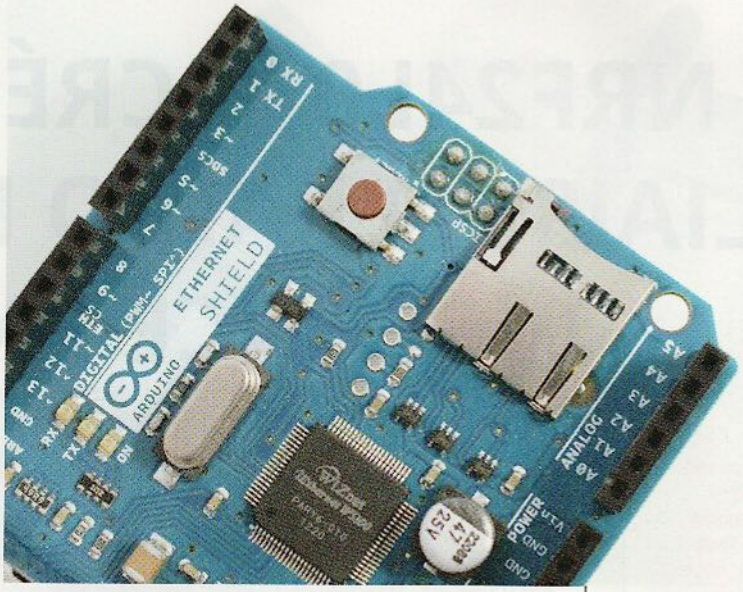
- `O_TRUNC` : si le fichier existe et est ouvert sans erreur en écriture sa taille est passée à zéro. Ceci permet d'écraser le contenu d'un fichier avec de nouvelles données, par opposition au fait d'en ajouter.

Notez un point important clairement précisé dans les fichiers de la bibliothèque : en aucun cas vous ne pouvez ouvrir un fichier de multiples fois. Ceci risque de conduire à une corruption de vos données.

5. QUOI D'AUTRE ?

Beaucoup de choses. La bibliothèque `SdFat` fournit un nombre de fonctionnalités impressionnant. Avec elle vous pourrez non seulement, comme ici, lire et écrire des fichiers, mais également : créer, supprimer et naviguer dans les répertoires, formater une carte, gérer plusieurs accès à des cartes SD, mesurer les performances d'un support en lecture et en écriture, manipuler les clusters directement, gérer les métadonnées des fichiers (dates), effacer de façon sécurisée des données (par écrasement), obtenir des informations sur les partitions, etc.

La bibliothèque est livrée avec un ensemble d'exemples très complets couvrant toutes les fonctionnalités proposées et mettant en scène différents scénarios d'utilisation. Notez cependant que la syntaxe utilisée par le programmeur pourra vous paraître un peu déroutante puisqu'il s'agit principalement de C++, alors que la majorité des croquis Arduino utilisent généralement une syntaxe plus proche du C (bien que le compilateur intégré soit C++).



Le shield Ethernet Arduino dispose d'un emplacement microSD fonctionnant exactement comme le module dont nous parlons ici. Seules différences notables, le prix et la connexion « en dur » du signal CS à la broche 4 (la 10 étant utilisée pour le CS du circuit intégré Ethernet Wiznet W5500).

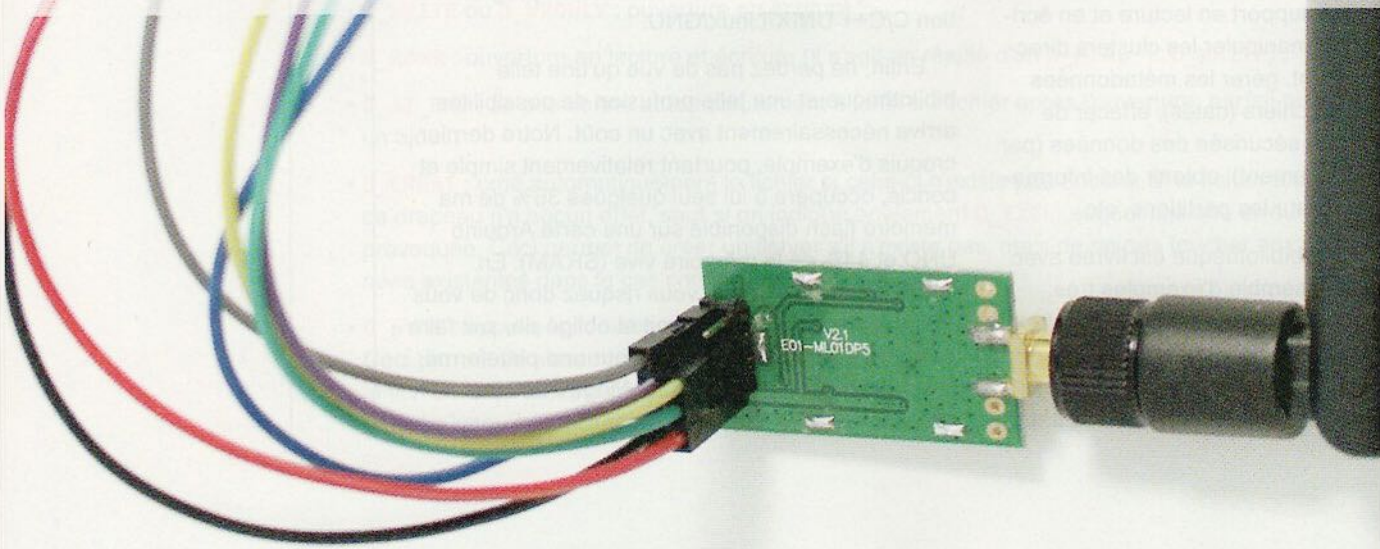
À l'instar de la technologie des supports SD et microSD, la bibliothèque est relativement complexe, mais vous permettra de rapidement arriver à un résultat. Nous avons ici à faire à quelque chose qui se rapproche beaucoup de la programmation C/C++ sur un système informatique tant au niveau de la logique utilisée que de la syntaxe des fonctions et méthodes proposées. L'auteur s'est largement inspiré des fonctions utilisées en programmation C/C++ UNIX/Linux/GNU.

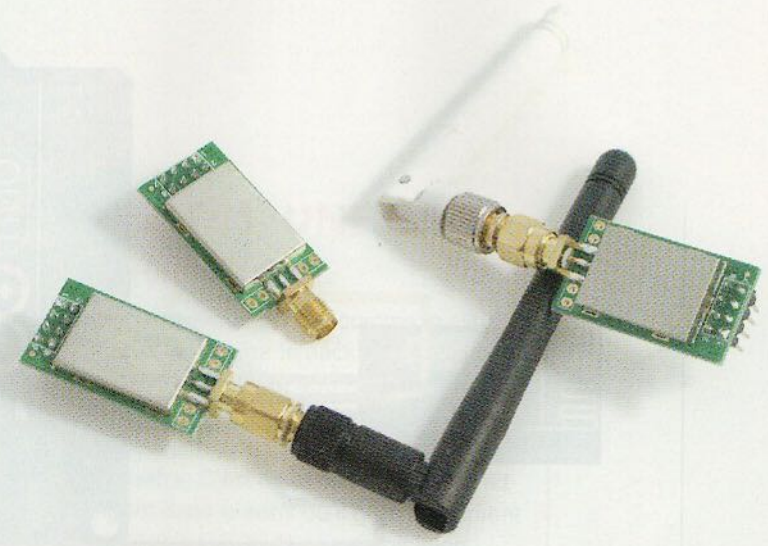
Enfin, ne perdez pas de vue qu'une telle bibliothèque et une telle profusion de possibilités arrive nécessairement avec un coût. Notre dernier croquis d'exemple, pourtant relativement simple et concis, occupera à lui seul quelques 36% de ma mémoire flash disponible sur une carte Arduino UNO et 44% de la mémoire vive (SRAM). En fonction de votre projet, vous risquez donc de vous retrouver rapidement à l'étroit et obligé de, soit faire des concessions, soit d'opter pour une plateforme plus « musclée » (comme une Mega 2560 par exemple). DB

NRF24L01 : CRÉEZ UNE LIAISON RADIO LONGUE DISTANCE ENTRE CARTES ARDUINO

Denis Bodor

Dès lors qu'on parle de communication sans fil, les options sont légion. Le Wifi et le Bluetooth viennent immédiatement à l'esprit mais, comme on peut s'en douter, la portée de ce type de solutions est assez réduite, car initialement non prévues pour ce type d'usage longue distance. Il faut alors se tourner soit vers la bidouille (type antenne Ricoré ou autre), des équipements coûteux (antenne à haut gain) ou tout simplement opter pour une autre technologie. L'une d'entre elles est très économique et très populaire : l'utilisation de modules nRF24L01.





Les modules utilisés pour les essais entourant cet article sont équipés d'une puce nRF24L01+ de Nordic et intègrent un amplificateur pour l'émission (PA) ainsi qu'un amplificateur faible bruit (LNA) pour la réception. Ceux-ci, associés à une antenne et une alimentation adaptées, permettent d'obtenir une distance de communication pouvant aller jusqu'à 800 mètres.

Tous les modules nRF24L01 reposent sur un composant unique, une puce de *Nordic Semiconductor* destinée à une utilisation polyvalente allant, bien entendu, des modules de communication aux claviers et souris sans fil, en passant par les carillons de porte, les capteurs, les sondes, les transmetteurs audios, les manettes de jeu, les télécommandes, etc. On retrouve littéralement ce circuit intégré, ou un autre membre de sa famille, partout.

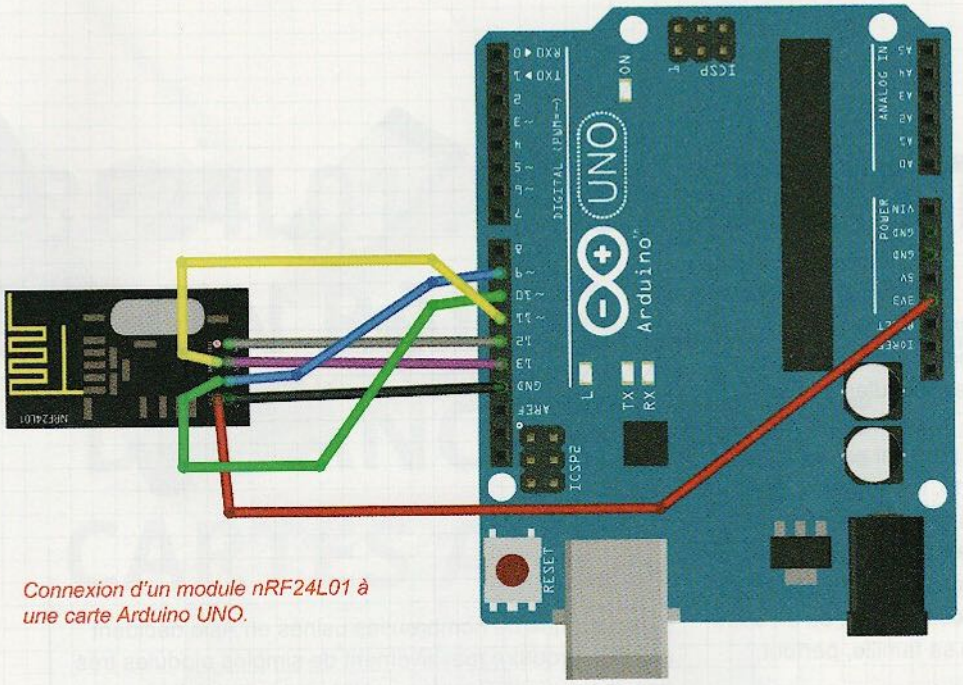
Le nRF24L01 est une puce de contrôle pour émetteur-récepteur (*transceiver* en anglais) à faible consommation utilisant la fréquence de 2,4 Ghz (bande ISM donc). Pouvant être alimenté entre 1,9 et 3,6 volts, celui-ci affiche une consommation extrêmement basse en fonction de son mode de fonctionnement et le rend parfaitement adapté à une utilisation autonome sur pile bouton CR2032 ou AAA. L'intégration est également l'une des raisons de la popularité de cette puce, car peu de composants supplémentaires sont nécessaires : un oscillateur à quartz, un circuit et une antenne.

1. À PROPOS DES MODULES NRF24L01

Bien évidemment, la profusion de nRF24L01 dans différents produits a rapidement conduit à une production massive et à une baisse proportionnelle des coûts de fabrication. Il n'en fallait pas plus pour que de nombreuses usines en Asie décident alors de produire massivement de simples modules très peu chers, de différentes qualités pour satisfaire les besoins des bidouilleurs du monde entier. Et lorsque je parle prix réduit, il faut bien comprendre qu'il n'est pas rare de trouver de tels modules pour 1€ pièce, port offert !

Bien entendu, en fonction du coût, la qualité, tant de fabrication qu'en termes de fonctionnalités varie grandement, ainsi que la distance maximum de communication. Dans les grandes lignes, on peut distinguer deux types de modules :

- ceux très peu chers ne disposant que du strict minimum : la puce, un quartz, quelques composants passifs (résistances et condensateurs) et une antenne directement dessinée sous la forme d'une piste sur le circuit imprimé. Différents essais de la part de nombreux utilisateurs ayant partagé leurs expérimentations en ligne montrent des distances de communication pouvant aller jusqu'à 80 ou 100 mètres en terrain découvert.
- ceux plus chers, entre 3 et 5€, disposant de deux circuits supplémentaires et, souvent, d'un connecteur d'antenne. Ces ajouts se composent d'un amplificateur de puissance abrégé PA pour *Power Amplifier* destiné à augmenter la puissance du signal en émission, et d'un amplificateur à faible bruit ou LNA pour *Low Noise Amplifier* améliorant la réception du signal tout en tentant de limiter les interférences. Ces deux circuits se placent entre le nRF24L01 et l'antenne. Le connecteur complémentaire, pouvant être en SMA ou en RP-SMA (comme



Connexion d'un module nRF24L01 à une carte Arduino UNO.

les périphériques Wifi), permet l'utilisation d'une antenne de meilleure qualité qu'une simple piste sur un circuit imprimé. Une bonne antenne est synonyme de gain passif plus important et donc d'une meilleure émission/réception. La portée de ce type de modules peut atteindre quelque 800 mètres ou plus, en l'absence d'obstacle, le tout pour quelques euros.

Le second modèle est, par définition, plus intéressant pour bon nombre d'applications et en particulier si vous cherchez une solution pour faire communiquer deux montages, soit sur une bonne distance, soit à l'intérieur d'un bâtiment plein de murs (comme la plupart des bâtiments). Les deux amplificateurs intégrés sont directement pilotés par le nRF24L01 qui, lui-même, dialogue avec votre montage Arduino en SPI. Il faut cependant prendre en considération un point important : l'alimentation en 3,3V nécessaire pour le module, généralement fournie par la carte Arduino, peut demander un courant allant jusqu'à 120 mA (en crête) en émission lorsque l'amplificateur est configuré au maximum de sa puissance. Or, une carte Arduino classique comme la UNO n'est en mesure de fournir, via sa broche « 3.3V », qu'un courant de 50 mA. Si le module tente de consommer davantage, soit il ne fonctionnera plus correctement, soit la tension va s'effondrer et le module va se réinitialiser.

Il faudra donc prévoir, dans certains cas, une alimentation 3,3V supplémentaire. Notez qu'il existe des circuits déjà tout prêts pour cet usage (merci la popularité des modules nRF24L01). Vendus entre 1 et 2€ pièce, ceux-ci sont constitués d'un régulateur de tension (LDO, souvent un AMS1117-3.3) permettant une alimentation 5V par la carte Arduino (capable de fournir bien plus de courant), et d'un emplacement où vient s'enficher directement le module nRF24L01. Cherchez simplement « *nRF24L01 Socket* » sur eBay et vous trouverez votre bonheur.

À propos de tension, on remarquera que le nRF24L01, bien que pouvant être alimenté de 1,9 à 3,6V, supporte sans problème l'utilisation de niveaux logiques 0/+5V comme c'est le cas avec la plupart des cartes Arduino. Il n'y a donc aucun problème dans le fait de connecter directement le module à la carte, aucune adaptation de niveau n'est nécessaire. La connexion en elle-même se fera avec :

- 11 : MOSI (*Master Out Slave In*) pour les données de l'Arduino vers le module, broche 6 du module ;
- 12 : MISO (*Master In Slave Out*) pour les données dans le sens inverse, broche 7 du module ;
- 13 : SCK (*CLocK*) pour le signal d'horloge qui cadence la communication, broche 5 du module ;

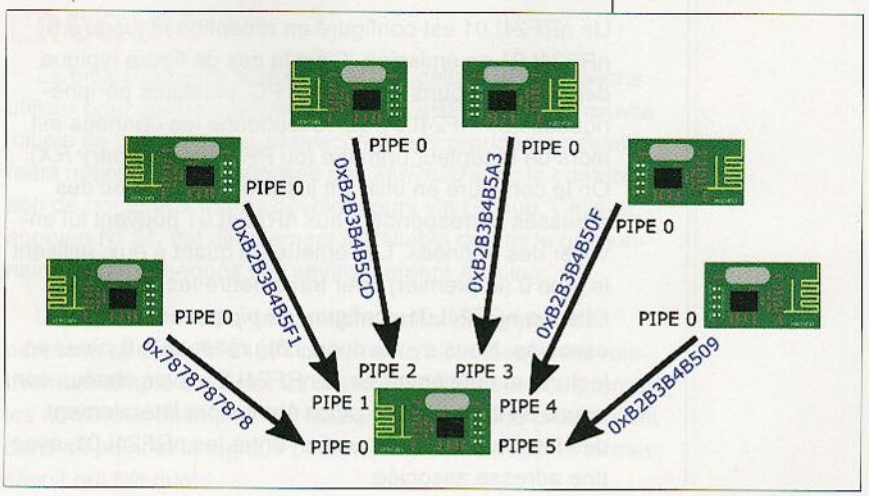
- 10 : CSN (*Slave Select Negative* ou *Chip Select Negative*) permet à celui qui contrôle la communication (Arduino) de demander au module de prendre en compte les instructions en SPI. Cette broche, la 4 sur le module, peut être reliée à n'importe quelle broche de l'Arduino. Ceci est configuré dans le croquis ;
- 9 : CE (*Chip Enable*) possède souvent la même signification que CS ou CSN, mais ici permet de contrôler l'émission radio du module (en plus de commandes envoyées en SPI). Là encore cette broche (3) peut être reliée à n'importe quelle broche de l'Arduino, tout cela étant contrôlé via le croquis ;
- alimentation (VCC) 3,3V connectée à la broche 2 du module ;
- la masse (GND) connectée à la broche 1 du module ;
- la broche 8 du module reste souvent non connectée. Il s'agit de la ligne d'interruption (IRQ) permettant au module de signaler que quelque chose doit être fait ou doit être pris en compte (le module « interrompt » le fonctionnement normal du croquis pour qu'on « s'occupe de lui »).

Ce brochage fait référence à une carte Arduino UNO, il faudra bien entendu l'adapter si vous utilisez une autre carte comme l'Arduino Mega 2560 disposant des connexions SPI sur d'autres broches (50, 51, 52, 53).

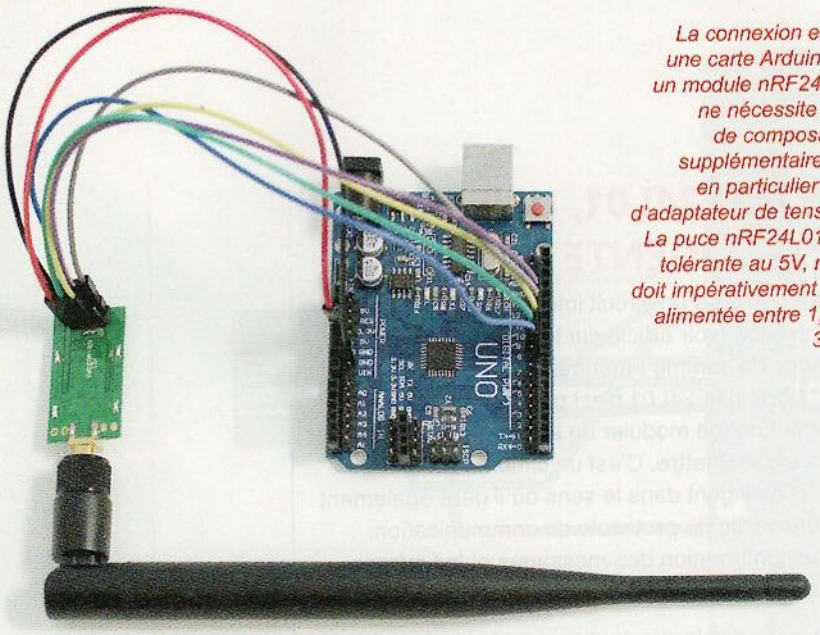
2. LE NRF24L01, UNE PUCE « INTELLIGENTE »

Contrairement à un circuit intégré comme le si4021 de Silicon Labs (voir article sur le hack d'une vieille télécommande comme émetteur 433 Mhz dans *Hackable n°12*), le nRF24L01 n'est pas une simple interface permettant de moduler un signal en fonction des données à transmettre. C'est un circuit qu'on pourrait qualifier d'intelligent dans le sens où il gère également une bonne partie du protocole de communication, jusqu'à la confirmation des messages et les retransmissions.

Cette puce de Nordic, tout comme les autres de la même famille, intègre un « accélérateur matériel de protocole » (dixit le site de Nordic) appelé « technologie *Enhanced ShockBurst* ». Celle-ci permet de décharger presque totalement le processeur hôte (le microcontrôleur de la carte Arduino dans notre cas) de toutes les tâches liées à la gestion de la communication. Ceci inclut énormément de choses, du format des messages à la gestion des destinataires en passant par les délais d'attente, les retransmissions de messages, la confirmation de réception, l'intégrité des données... Ceci est idéal, par exemple, pour les périphériques comme les souris sans fil ou les claviers, n'embarquant pas nécessairement beaucoup de ressources en termes de puissance de calcul. Ceci nous arrange également ici, puisque cela fait bien moins de travail pour notre croquis.



Nous avons ici un module nRF24L01 en réception, c'est le PRX et 6 modules en émission. L'ensemble des 6 pipes disponibles est utilisé. C'est le schéma classique utilisé pour certains ensembles clavier/souris sans fil.



La connexion entre une carte Arduino et un module nRF24L01 ne nécessite pas de composants supplémentaires et en particulier pas d'adaptateur de tension. La puce nRF24L01 est tolérante au 5V, mais doit impérativement être alimentée entre 1,9 et 3,6V.

Du fait de cette « intelligence » embarquée, un certain nombre de concepts importants doivent être assimilés pour pouvoir utiliser des modules nRF24L01. L'un des points les plus importants est la notion de *pipe* (« paille-pe » ou « tuyau/canaux » en anglais, mais nous conserverons le terme original ici), associée à celle d'adresses.

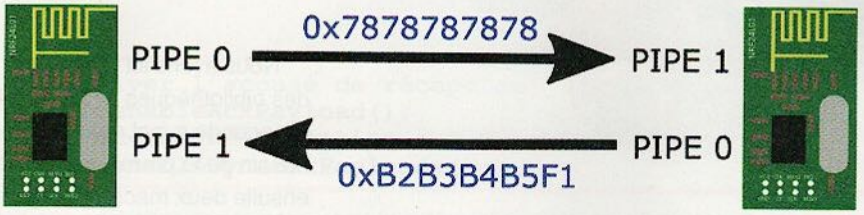
Un nRF24L01 peut établir des connexions à un niveau logique qui n'est pas lié aux fréquences utilisées (les canaux, les vrais), mais à la façon de désigner les acteurs de la communication. Chaque nRF24L01 peut utiliser des *pipes* pour établir des connexions avec un ou plusieurs autres nRF24L01. Six *pipes* sont disponibles pour votre utilisation, numérotés de 0 à 5, sur chaque nRF24L01.

Deux cas de figure peuvent alors se présenter :

- Un nRF24L01 est configuré en réception et jusqu'à 6 nRF24L01 en émission. C'est le cas de figure typique des claviers/souris sans fil (un PC, plusieurs périphériques). Le nRF24L01 qui réceptionne les données est alors un récepteur primaire (ou PRX pour *primary RX*). On le configure en utilisant jusqu'à 6 *pipes* avec des adresses correspondant aux nRF24L01 pouvant lui envoyer des données. Les émetteurs quant à eux, utilisent le *pipe* 0 (le premier) pour transmettre les messages. Chaque nRF24L01 configure ce *pipe* avec l'adresse associée. Nous avons donc, côté récepteur, 6 *pipes* en lecture, et côté émetteur, 6 nRF24L01, avec chacun son *pipe* 0 en écriture. Les *pipes* étant alors littéralement des tuyaux établissant un lien entre les nRF24L01, avec une adresse associée.
- Deux nRF24L01 dialoguent échangeant les rôles d'émetteur et de récepteur en fonction du sens des messages. Là, chaque nRF24L01 utilise alors deux *pipes*. Le premier est configuré avec le *pipe* 0 en écriture en spécifiant une adresse, et un second *pipe* (disons le 1) en lecture en spécifiant une autre adresse. Le second nRF24L01 configure son *pipe* 0 avec la même adresse que le *pipe* 1 du premier nRF24L01, et son *pipe* 1, pour la lecture, avec l'adresse du *pipe* 0 du premier nRF24L01. Imaginez cela comme une connexion série entre deux Arduino. Le RX de l'un sur le TX de l'autre et inversement. Sauf qu'ici, les câbles portent des noms : leurs adresses.

Voir les *pipes* ou les adresses comme les désignations des nRF24L01 est trompeur. Éliminez cette idée de votre esprit le plus rapidement possible, elle ne fait que compliquer les choses. Mon conseil est de bien imager dans votre esprit que les adresses utilisées ne sont pas des dénominations des modules nRF24L01, mais bel et bien des *pipes* eux-mêmes. Ce sont les noms des liaisons, des tuyaux, des connexions virtuelles... Exactement comme pour une liaison série, si ce n'est qu'il n'y a pas de RX/TX (sauf en se les imaginant comme les numéros de *pipe*), mais que les câbles portent des noms, des adresses choisies pour les désigner.

Vous l'avez compris, le *pipe* 0 est spécial. C'est le seul qui peut être utilisé pour l'envoi de données,



Dans le cas d'une communication bidirectionnelle entre deux nRF24L01, deux pipes sont utilisés, véhiculant chacun des messages dans un sens. C'est le même principe que dans une communication série, à l'exception du fait qu'ici les « câbles » portent des « noms ».

mais aussi pour la réception. Les *pipes* 1 à 5 ne peuvent être configurés que pour la réception. Généralement, dans la mesure du possible, on n'utilisera pas le *pipe* 0 en lecture, sauf si effectivement nous avons 6 nRF24L01 émetteurs et un récepteur.

Cette différenciation entre le *pipe* 0 et les autres s'étend aux adresses utilisables. Le *pipe* 0 peut être configuré avec une adresse de 40 bits unique, mais les *pipes* 1 à 5 ne peuvent être configurés qu'avec une adresse composée de 32 bits communs et de 8 bits de poids faible (l'octet complètement à droite dans un croquis Arduino pour un `uint64_t`) au choix. Exemple pour une communication entre un nRF24L01 récepteur et 6 nRF24L01 émetteurs :

- *pipe* 0 : `0x7878787878`,
- *pipe* 1 : `0xB2B3B4B5F1` (bits de poids faible à `F1`),
- *pipe* 2 : `0xB2B3B4B5CD` (bits de poids faible à `CD`),
- *pipe* 3 : `0xB2B3B4B5A3` (bits de poids faible à `A3`),
- *pipe* 4 : `0xB2B3B4B50F` (bits de poids faible à `0F`),
- *pipe* 5 : `0xB2B3B4B509` (bits de poids faible à `09`).

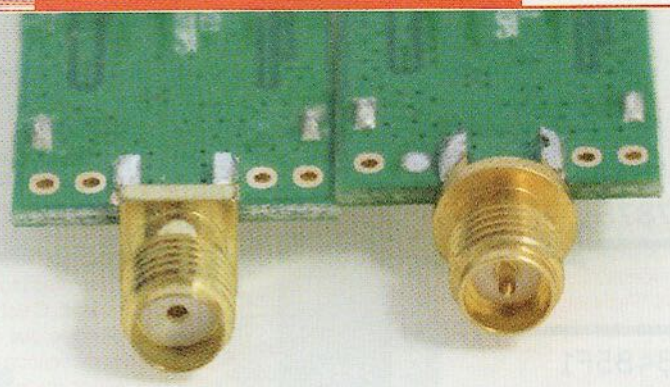
Au niveau de la transmission elle-même, ces valeurs ou adresses sont présentes, modulées, dans le signal (en GFSK, *Gaussian frequency-shift keying*, qui est un type de modulation de fréquences) sous forme de bits. Il est théoriquement possible d'utiliser n'importe quelle adresse de son choix, mais en ayant connaissance de certaines répercussions quant à la forme du signal lui-même.

En cas de bruit parasite ou de distorsion du signal, certaines adresses peuvent être plus facilement mal interprétées que d'autres par un nRF24L01. L'octet `0x01` par exemple nous donne en binaire `0b00000001` qui peut alors aisément être pris pour un « bruit ». De la même manière, l'octet `0xAA` nous donne `0b10101010` qui, avec sa séquence de `1` et de `0` peut être vu comme un signal de synchronisation (préambule). Ceci ne signifie pas pour autant qu'il est interdit d'utiliser ces valeurs, mais on évitera diverses combinaisons comme `0xFFFFFFFFFF`, `0x0000000000`, `0xAA01AA01AA`, `0xAAAAAAAAAA` ou encore `0x0000010001`. De façon générale, on tentera de ne pas utiliser de simples mélanges de `0xAA`, `0x55`, `0x2A`, `0x01`, `0x0a`, `0x15`, `0x05` et de `0x02` formant des répétitions de `0b01` entrecoupées de séries de `0b00`.

3. ARDUINO, BIBLIOTHÈQUE ET CROQUIS

La bibliothèque la plus populaire et celle que nous allons utiliser pour commander nos modules nRF24L01 est l'œuvre initiale de J. Coliz (alias *maniacbug*) puis reprise (et actuellement maintenue) et optimisée par *TMRh20* avec la contribution de nombreux autres développeurs via GitHub. Celle-ci, appelée RF24, est directement installable depuis le gestionnaire de bibliothèques de l'environnement Arduino.

Une fois le principe de fonctionnement des *pipes*, des adresses et des nRF24L01 compris, la bibliothèque est relativement simple à utiliser. La seule difficulté pour comprendre les nombreux exemples qui l'accompagne réside uniquement dans ce point et la logique particulière, ou en d'autres termes, savoir qui fait quoi.



Les modules vendus sur le Web peuvent être équipés d'un connecteur d'antenne. Si le module est vendu sans l'antenne, il faudra être très attentif à ce connecteur : à gauche du SMA et à droite du RP-SMA, très commun avec les adaptateurs et routeurs Wifi.

Contrairement à l'exemple fourni par la bibliothèque pour débuter, nous allons ici écrire deux croquis afin de bien séparer un couple Arduino Uno + nRF24L01 prenant un rôle et un autre couple Arduino Nano + nRF24L01 en prenant un autre. Notre objectif ici sera de faire envoyer un message par un duo, de le recevoir avec un autre qui retournera un message de confirmation.

Notez que ce mécanisme n'est pas réellement nécessaire avec les nRF24L01, car le circuit intégré prend en charge de lui-même la notion d'accusé de réception. Vos croquis ne sont donc, en principe, pas obligés de s'assurer de cette manière de la bonne transmission d'un message. Ceci nous permettra cependant d'asseoir à la fois les notions de rôle d'émetteur et de récepteur et celles de *pipes* et d'adresse.

Nous utiliserons donc deux cartes Arduino et les rôles de chacune dans notre scénario n'ont pas réellement d'importance (du moment qu'on ne confond pas cartes et ports au moment de charger le croquis en mémoire).

3.1 Premier croquis

Nous allons commencer par le croquis équipant la carte Arduino chargée de répondre aux demandes, et nous allons le découper en petits morceaux pour bien comprendre chaque étape. Nous commençons donc par inclure les bibliothèques nécessaires et déclarer quelques macros, variables et objet :

```
#include <SPI.h>
#include <RF24.h>

#define PIN_CS  10
#define PIN_CE  9

RF24 radio(PIN_CE,PIN_CS);

const uint64_t pipeA =
0xF0F0F0F0E1LL;
const uint64_t pipeB =
0xF0F0F0F0D2LL;
```

Nous avons ici l'inclusion des bibliothèques **SPI** et **RF24**, la seconde ayant naturellement besoin de la première. Viennent ensuite deux macros désignant les broches utilisées pour les signaux CSN et CE du module qui, comme explicité précédemment, permettent d'adresser le module et de changer son mode de fonctionnement. Ces macros sont immédiatement utilisées pour déclarer l'objet qui désormais représentera le module nRF24L01 connecté dans le reste du croquis et que nous utiliserons pour le contrôler : **radio**.

Enfin, nous déclarons deux variables de type **uint64_t** (valeurs entières non signées sur 64 bits) afin d'y stocker les adresses de 40 bits des *pipes* que nous allons utiliser. Ces quelques lignes sont strictement identiques pour les deux croquis et c'est généralement une bonne idée de tout simplement les copier/coller pour éviter toutes fautes de frappe.

Nous pouvons ensuite passer à la composition de la fonction **setup()** chargée de configurer les modules et l'ensemble des éléments du croquis. Nous commençons par initialiser la communication avec le moniteur série en 115200 bps ainsi que le module radio :

```
void setup() {
  Serial.begin(115200);
  Serial.println(F("Go go go!"));

  radio.begin();
```

Nous activons ensuite la fonctionnalité *ShockBurst* permettant l'accusé de réception automatique, géré en coulisse par le nRF24L01 pour nous :

```
// Activer l'accusé de réception
radio.enableAckPayload();
// L'accusé de réception est dynamique
radio.enableDynamicPayloads();
```

Notez que ceci n'est pas forcément nécessaire ici puisque tout l'objet de l'essai est précisément d'avoir une réponse en retour en plus de la fonctionnalité offerte de base par les nRF24L01.

Vient ensuite la configuration des amplificateurs (PA et LNA), dans le cas où ils sont présents :

```
// réglage de l'amplificateur
radio.setPALevel(RF24_PA_LOW);
//radio.setPALevel(RF24_PA_HIGH);
```

Par défaut, c'est **RF24_PA_HIGH** qui est utilisé, mais cette configuration provoque une consommation excessive de courant (jusqu'à 120 mA) alors que la broche 3,3V d'une carte Arduino ne peut pas en fournir autant (mes alimentations sont encore en chemin alors que j'écris ces lignes). Comme nous sommes en phase d'essai, les deux ensembles sont relativement proches et ne nécessitent donc pas une pleine utilisation des amplificateurs. Notez au passage que **RF24_PA_LOW** permet tout de même des communications sur des distances plus importantes que le Wifi en intérieur. Le fonctionnement de ces montages pour test n'a posé aucun problème avec une telle configuration d'un bout à l'autre de nos locaux et ce avec quelques 5 murs en pierre de taille entre l'un et l'autre nRF24L01.

Arrive alors le moment de configurer les *pipes* :

```
// réglage des canaux/pipes
radio.openWritingPipe(pipeA);
radio.openReadingPipe(1,pipeB);
```

La méthode **openWritingPipe()** nous permet de connecter un *pipe* en écriture et plus exactement le *pipe* 0, le seul pouvant être utilisé pour cela. Inversement, **openReadingPipe()** est utilisé pour la connexion d'un *pipe* en lecture. Ici deux paramètres sont nécessaires, le numéro du *pipe* entre 0 et 5 et l'adresse à utiliser. Notez que vous ne pouvez pas impunément configurer à la fois le *pipe* 0 en écriture et en lecture, car en utilisant ensuite la méthode **startListening()**, la configuration en écriture sera perdue. Il vous faudra à nouveau utiliser **openWritingPipe()** avant de tenter d'écrire sur ce *pipe* avec **write()**. De façon générale, et dans la mesure du possible, le plus simple est de tout simplement utiliser le *pipe* 0 que pour l'écriture.

Enfin, nous touchons à la fin de la fonction **setup()** et pouvons nous mettre en attente de messages en réception avec :

```
  // En écoute
  radio.startListening();
}
```

Nous pouvons maintenant nous pencher sur la boucle principale **loop()**. Notre objectif est ici d'être en écoute d'un éventuel message et si l'un d'entre eux arrive, utiliser son contenu et tout simplement le retourner à l'expéditeur. En d'autres termes, on lit sur un *pipe* et si quelque chose se présente, on l'écrit sur l'autre. Le message qui sera ici utilisé est une simple valeur entière non signée de type **unsigned long** correspondant, pour l'expéditeur à la valeur de **micros()** (le **millis()** pour les microsecondes).

En disposant de modules avec un connecteur RP-SMA et étant donné que les fréquences utilisées par les nRF24L01 sont similaires à celles du Wifi, il sera possible d'utiliser une antenne Wifi provenant d'un routeur, une netcam ou un adaptateur USB sans le moindre problème.

Nous débutons donc notre fonction **loop()** en vérifiant si des données sont disponibles en lecture :

```
void loop() {
  unsigned long temps;

  // Des données sont-elles disponibles ?
  if(radio.available()){
```

Si cela vous rappelle quelque chose, c'est normal, c'est exactement le même principe que pour une lecture sur le port série (**Serial**) de l'Arduino. **available()** est VRAI si des données peuvent être prises en charge. Si tel est le cas, nous récupérons toutes les données jusqu'à la dernière :

```
    // oui, lecture
    while (radio.available()) {
      radio.read(&temps,
        sizeof(unsigned long));
    }
```

Et nous les stockons dans **temps**. Notez que si plusieurs messages arrivent trop rapidement, **temps** ne contiendra que la dernière valeur. Nous pouvons alors cesser d'écouter et passer en émission :

```
    // arrête émission
    radio.stopListening();
    // envoi de la réponse
    radio.write(&temps,
      sizeof(unsigned long));
```

stopListening(), comme son nom l'indique, demande au nRF24L01 de stopper l'écoute afin que nous puissions, avec **write()** écrire, en d'autres termes émettre, les données contenues dans **temps**. Notez que la méthode **write()**, tout comme **read()**, prend en argument un **pointeur** et non la variable elle-même (notez le **&** devant le nom), ainsi que la taille des données à lire (et donc ici celle d'un **unsigned long**, le type de la variable **temps**).

Une fois cette opération terminée, il ne reste plus qu'à se remettre à l'écoute d'éventuels messages et accessoirement notifier de ce qui vient de se passer sur le moniteur série :

```
    // reprise de la réception
    radio.startListening();

    // Affichage
    Serial.print(F("pong !  "));
    Serial.println(temps);
  }
}
```

3.2 Second croquis

Notre second croquis, fonctionnant donc sur la seconde carte Arduino, repose exactement sur les mêmes principes et est en grande partie similaire au précédent. Je vous le livre donc en entier en une fois :

```
#include <SPI.h>
#include <RF24.h>

#define PIN_CS  10
#define PIN_CE  9

RF24 radio(PIN_CE,PIN_CS);

const uint64_t pipeA = 0xF0F0F0F0E1LL;
const uint64_t pipeB = 0xF0F0F0F0D2LL;

void setup() {
  Serial.begin(115200);
  Serial.println(F("Go go go!"));

  radio.begin();

  // Activer l'accusé de réception
  radio.enableAckPayload();
  // L'accusé de réception est dynamique
  radio.enableDynamicPayloads();

  // réglage de l'amplificateur
  radio.setPALevel(RF24_PA_LOW);
  //radio.setPALevel(RF24_PA_HIGH);

  // réglage des canaux (pipe)
  radio.openWritingPipe(pipeB);
  radio.openReadingPipe(1,pipeA);

  // En écoute
  radio.startListening();
}

void loop() {
  // Arrêt écoute
  radio.stopListening();
  Serial.println(F("Envoi"));

  // Utilisation de micro comme message
  unsigned long message = micros();
  // envoi
  if (!radio.write(&message, sizeof(unsigned long) )){
    Serial.println(F("erreur"));
  }

  // Mise en écoute
  radio.startListening();

  // Enregistrement du début du délais
```

```
    unsigned long debut = micros();
    boolean timeout = false;

    // Tant que rien à lire
    while (!radio.available()){
      // 600ms de passé, on abandonne
      if (micros()-debut > 600000 ){
        timeout = true;
        break;
      }
    }

    // abandon pour cause de délais écoulé ?
    if(timeout){
      Serial.println(F("Erreur : timeout."));
    } else {
      // variable pour le message reçu
      unsigned long reception;
      // lecture des données reçues
      radio.read(&reception, sizeof(unsigned long));
      // enregistrement du moment de la réception
      unsigned long fin = micros();

      // Affichage
      Serial.print(F("Message: "));
      Serial.print(message);
      Serial.print(F(", reponse: "));
      Serial.print(reception);
      Serial.print(F(", temps complet: "));
      Serial.print(fin-message);
      Serial.println(F(" microsecondes"));
    }

    delay(1000);

}
```

Les différences résident dans le corps de la boucle principale, mais utilisent la même logique d'attente et d'envoi de messages. Notez cependant, dans **setup()**, que l'utilisation des variables contenant les adresses des *pipes* sont inversées. **pipeB** est utilisé en écriture (donc le *pipe* 0) et **pipeA** en lecture sur le *pipe* 1.

La gestion des délais repose entièrement sur les valeurs retournées par **micros()** qui forment également le contenu du message envoyé. En fin de boucle, nous affichons les données de chaque envoi avec le contenu du message envoyé (**message**), la valeur du message reçu en réponse (**reception**) et le nombre de microsecondes séparant le moment de l'envoi (**message** également) de celui juste après la fin de la réception (**fin**).

Dans la trousse à outils de l'utilisateur de modules nRF24L01, comme dans celle de l'amateur de radio logicielle, les adaptateurs SMA/RP-SMA trouvent bonne place. Ces simples « changeurs de genre » pourront, à l'occasion, vous sortir de belles épines du pied...

3.3 Résultat

Une fois chaque croquis chargé dans les cartes Arduino, en commençant par celui de réception/réponse, on constate en observant le moniteur série sur le second Arduino, le déroulement des opérations :

```
Go go go!
Envoi
Message: 11916, reponse: 11916, temps complet: 1192 microsecondes
Envoi
Message: 1014640, reponse: 1014640, temps complet: 1200 microsecondes
Envoi
Message: 2017568, reponse: 2017568, temps complet: 1196 microsecondes
Envoi
Message: 3020488, reponse: 3020488, temps complet: 1160 microsecondes
Envoi
Message: 4023392, reponse: 4023392, temps complet: 1192 microsecondes
Envoi
Message: 5026312, reponse: 5026312, temps complet: 1160 microsecondes
Envoi
Message: 6029200, reponse: 6029200, temps complet: 1200 microsecondes
```

Un peu plus d'une milliseconde pour ce type d'application n'est pas quelque chose de bien important. Le nRF24L01 par défaut communique à une vitesse de 1Mbps, c'est plus que ne peuvent en gérer ici nos croquis dans ce contexte. Il est possible de changer cette vitesse avec la méthode `setDataRate()`, prenant en argument `RF24_2MBPS`, `RF24_1MBPS` ou `RF24_250KBPS` (uniquement disponible dans la version nRF24L01+ du composant). Cette méthode retourne un booléen à VRAI si l'opération est un succès et à FAUX dans le cas contraire. Notez que l'utilisation de cette méthode avec l'argument `RF24_250KBPS` vous permettra également de déterminer si votre module est un nRF24L01 ou un nRF24L01+ (en cas de doute).

POUR FINIR

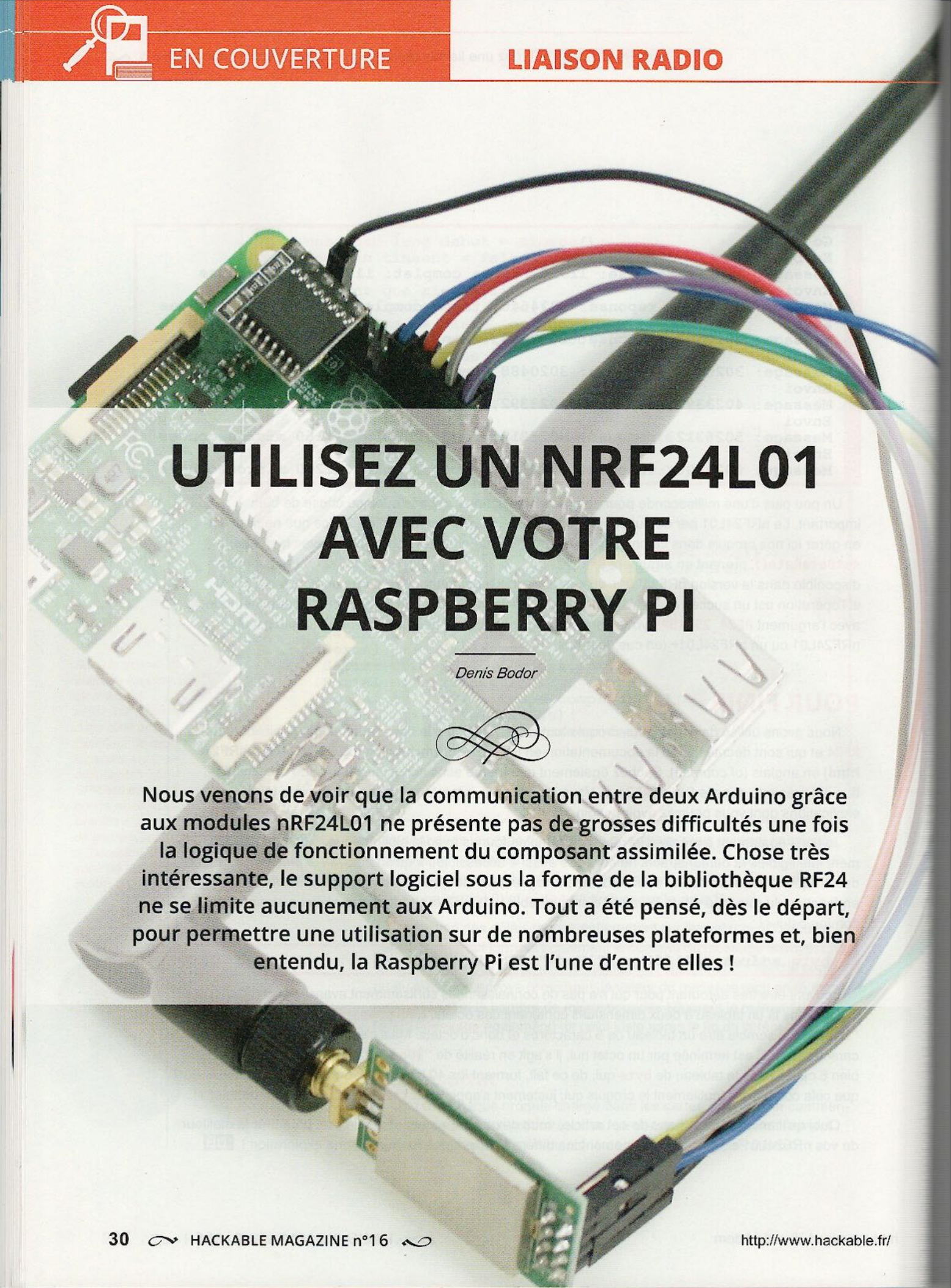
Nous avons utilisé dans ces deux croquis toutes les fonctions de base fournies par la bibliothèque `RF24` et qui sont décrites dans la documentation en ligne (**http://tmrh20.github.io/RF24/classRF24.html**) en anglais (*of course* !). Sachez également que même sans accès au Web, cette documentation est présente dans le fichier `RF24.h`. Vous trouverez là, sous forme de commentaires toutes les explications dont vous pourrez avoir besoin.

Je tiens également à préciser ici que les exemples fournis avec la bibliothèque, bien que parfaitement intelligibles et démonstratifs ne sont pas aussi « accessibles » qu'on pourrait le penser. La plupart des croquis mettent en scène deux Arduino partageant un même code, changeant de comportement en fonction du rôle joué (et de la valeur d'une variable `role`). Ceci en plus de certains « raccourcis » comme celui-ci :

```
byte addresses[][6] = {"1Node","2Node"};
```

peuvent être très déroutant pour qui n'a pas de connaissances suffisamment avancées en C/C++. Nous avons là un tableau à deux dimensions contenant des octets. `addresses[0]` par exemple contient `"1Node"` qui semble être un tableau de 5 caractères et donc d'octets. Mais c'est oublier qu'une chaîne de caractères en C est terminée par un octet nul, il s'agit en réalité de `"1Node\0"`, nous avons donc bel et bien 6 cases dans le tableau de `byte` qui, de ce fait, forment les 40 bits d'une adresse de *pipe*. Je trouve que cela complexifie inutilement le croquis qui, justement s'appelle, `GettingStarted` (« pour débuter »)...

Quoi qu'il en soit, sur la base de cet article, vous devriez être maintenant fin prêt pour tirer le meilleur de vos nRF24L01 et analyser sereinement les différents exemples fournis. Bonne exploration ! DB



UTILISEZ UN NRF24L01 AVEC VOTRE RASPBERRY PI

Denis Bodor

Nous venons de voir que la communication entre deux Arduino grâce aux modules nRF24L01 ne présente pas de grosses difficultés une fois la logique de fonctionnement du composant assimilée. Chose très intéressante, le support logiciel sous la forme de la bibliothèque RF24 ne se limite aucunement aux Arduino. Tout a été pensé, dès le départ, pour permettre une utilisation sur de nombreuses plateformes et, bien entendu, la Raspberry Pi est l'une d'entre elles !

Faire communiquer deux cartes à microcontrôleurs comme des Arduino est un bon point de départ, mais parfois, pour certains projets les ressources disponibles ou tout simplement l'objectif même du projet imposent l'utilisation de quelque chose d'un peu plus musclé. C'est généralement là qu'une carte comme la Raspberry Pi entre en jeu en changeant totalement la donne.

Le cas de figure le plus courant alors est celui où un ou plusieurs montages à base d'Arduino sont utilisés comme éléments distants (capteur, afficheur, contrôleur mécanique, etc.) et la Raspberry Pi comme système central orchestrant l'ensemble ou regroupant les informations collectées.

Nous allons donc brièvement ici ajouter un pendant Raspberry Pi à la démonstration précédente, 100% Arduino, en remplaçant tout simplement l'une des deux cartes par une Pi, et plus précisément celle qui envoie les messages et attend le retour pour ensuite afficher les détails des opérations.

1. PRÉPARATION ET INSTALLATION

Pour pouvoir utiliser un module nRF24L01 avec votre Pi, vous devez avant toutes choses activer le support SPI de la carte. Il y a plusieurs façons de faire, mais je passe généralement par l'édition du fichier **/boot/config.txt** afin d'ajouter ou décommenter (retirer le **#**) la ligne :

```
dtparam=spi=on
```

Ce réglage peut également être fait via un **sudo raspi-config**, en passant par ***Advanced Options*** et ***SPI*** pour activer le support. Dans les deux cas, un redémarrage de la Raspberry Pi est nécessaire, ce après quoi, le bus SPI sera activé et accessible, entre autres, via :

```
$ ls -la /dev/spi*
crw-rw---- 1 root spi 153, 0 déc.  12 16:17 /dev/spidev0.0
crw-rw---- 1 root spi 153, 1 déc.  12 16:17 /dev/spidev0.1
```

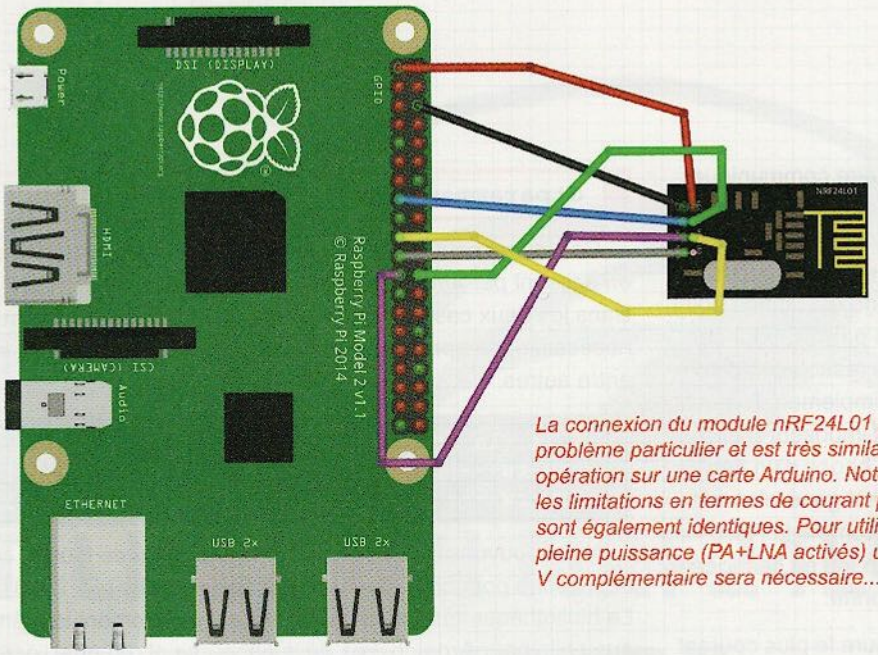
Nous pouvons ensuite passer à l'installation de tout le nécessaire pour contrôler et utiliser les modules nRF24L01. La bibliothèque que nous utiliserons est exactement la même que celle pour Arduino. Les développeurs ont, en effet, écrit un unique code, qui fonctionne tout aussi bien sur toutes les plateformes supportées. Cette installation ne sera pas aussi simple que celle dans l'environnement Arduino et, bien que cela ne me plaise pas énormément, devrait être faite à la main, indépendamment du système de gestion de paquets. Nous devrons récupérer les sources de la bibliothèque, la compiler et installer les éléments afin de pouvoir écrire du code en faisant usage.

La première étape consistera à installer les paquets nécessaires pour la suite des opérations, avec **sudo apt-get update** puis **sudo apt-get install build-essential make git**. Il est probable qu'une partie de ces éléments soient déjà présents sur votre Pi, ce n'est pas un problème, le cas échéant ils seront mis à jour.

La récupération des sources se fera avec la commande **git** directement depuis le dépôt officiel sur GitHub (cf. article dans le numéro précédent sur Git et GitHub) :

```
$ git clone https://github.com/TMRh20/RF24.git
Clonage dans 'RF24'...
remote: Counting objects: 3017, done.
remote: Total 3017 (delta 0), reused 0 (delta 0), pack-
reused 3017
Réception d'objets: 100% (3017/3017), 1.29 MiB | 1.20 MiB/s,
fait.
Résolution des deltas: 100% (1780/1780), fait.
Vérification de la connectivité... fait.
```

Nous obtenons ainsi un répertoire **RF24** dans le répertoire courant. Nous avons ici l'ensemble du code pour toutes les plateformes. Pour compiler tout cela pour la Raspberry Pi, nous commençons par nous placer dans le répertoire et configurer les sources :



La connexion du module nRF24L01 ne pose pas de problème particulier et est très similaire à la même opération sur une carte Arduino. Notez au passage que les limitations en termes de courant pouvant être fourni sont également identiques. Pour utiliser le module à pleine puissance (PA+LNA activés) une alimentation 3,3 V complémentaire sera nécessaire...

```
$ cd RF24
$ ./configure --driver=SPIDEV
```

Ici, nous choisissons d'utiliser le support SPIDEV. Je vous laisse le soin de consulter le site officiel pour d'autres options (dont MRAA). Nous passons ensuite à la compilation :

```
$ make
[...]
[Linking]
arm-linux-gnueabihf-gcc -pthread -shared -Wl,-soname,librf24.so.1
-march=armv7-a -mtune=cortex-a7 -mfpu=neon-vfpv4 -mfloat-abi=hard
-Ofast -Wall -pthread  -o librf24.so.1.2.0 RF24.o spi.o
gpio.o compatibility.o interrupt.o
```

Puis directement à l'installation :

```
$ sudo make install -B
[...]
[Installing Libs to /usr/local/lib]
[Installing Headers to /usr/local/include/RF24]
```

À ce stade, la bibliothèque est installée dans votre système (**/usr/local/lib/librf24.so***) ainsi que les différents fichiers permettant d'écrire des programmes C et C++ (dans **/usr/local/include/RF24**). Mais ne nous arrêtons pas en si bon chemin et poursuivons en installant également le support Python.

Pour cela, commencez par installer les paquets **python-dev**, **libboost-python-dev** et **python-setup-tools** avec **sudo apt-get install**. Rendez-vous ensuite dans le sous-répertoire **pyRF24** et utilisez :

```
$ ./setup.py build
running build
running build_ext
building 'RF24' extension
creating build
[...]
```

Notez que cette opération prend beaucoup de temps et consomme énormément de mémoire vive. En cas de problème, quittez les applications inutiles et éventuellement, le mode graphique.

```
$ sudo ./setup.py install
[...]
Processing dependencies for
RF24==1.2.0
Finished processing
dependencies for RF24==1.2.0
```

À ce stade, tout est installé dans le système de la Pi et prêt à être utilisé. Il ne nous reste plus qu'à connecter le module à la carte, exactement comme pour les Arduino, sans autre élément nécessaire qu'une poignée de câbles :

- 1 GND (masse) sur la broche 25, 9, 14, 30, 34 ou 39 de la Pi,
- 2 VCC (alimentation 3,3V) sur 1 ou 17,
- 3 CE sur 15 (gpio22),
- 4 CSN sur 24 (gpio8),
- 5 SCK sur 23,
- 6 MOSI sur 19,
- 7 MISO sur 21.

2. PORTAGE DU CROQUIS ARDUINO

Il est temps de passer au code et une petite surprise nous attend. En effet, comme la bibliothèque Arduino, et donc le code de base du support, est en « langage Arduino » qui n'est autre que du C/C++, la façon naturelle de l'utiliser avec une Raspberry Pi est aussi du C++. Ne vous inquiétez pas, si vous n'avez jamais fait de C ou de C++ sur une Pi, mais que vous écrivez des croquis Arduino, vous ne serez pas vraiment dépaysé.

Un croquis Arduino **EST** du C++. Quelques principes mêmes de l'écriture de programmes dans ce langage vous sont simplement dissimulés lorsque vous utilisez l'environnement Arduino. Ceci concerne, par exemple, les « bibliothèques » incluses en début de programme. Il ne s'agit pas réellement de bibliothèques, mais de fichiers d'entête permettant au programme d'avoir un « point d'accès » aux bibliothèques qui sont liées durant le processus de compilation (édition de liens).

De la même façon, il n'y a pas de **setup()** et de **loop()** dans un code C/C++ classique, mais une fonction qui est appelée automatiquement à l'exécution du programme : **main()**, la fonction principale. En réalité, ce que fait un croquis Arduino est :

```
int main(void) {
  init();
  initVariant();

  setup();

  for (;;) {
    loop();
    if (serialEventRun)
serialEventRun();
  }

  return 0;
}
```

Nous avons une boucle **for()** répétant sans cesse un appel à **loop()**, juste après l'appel à **setup()**. Avec un programme écrit en C ou C++, il vous suffit de faire cela vous-même. Voici donc l'adaptation de notre croquis Arduino, porté vers la Pi :

```
#include <cstdlib>
#include <iostream>
#include <sstream>
#include <string>
#include <unistd.h>
#include <RF24/RF24.h>

// CSN est géré par la Pi et CE est sur
// la GPIO 22
RF24 radio(22,0);

const uint64_t pipeA = 0xF0F0F0F0E1LL;
const uint64_t pipeB = 0xF0F0F0F0D2LL;

int main(int argc, char** argv){
  unsigned long message;
  unsigned long debut;
  bool timeout;
  unsigned long reception;
  unsigned long fin;

  printf("Go go go!\n");
```

```
radio.begin();

// Activer l'accusé de réception
radio.enableAckPayload();
// L'accusé de réception est dynamique
radio.enableDynamicPayloads();
// réglage de l'amplificateur
radio.setPALevel(RF24_PA_LOW);

// réglage des canaux (pipe)
radio.openWritingPipe(pipeB);
radio.openReadingPipe(1,pipeA);

// affichage d'un résumé de la configuration
radio.printDetails();

// En écoute
radio.startListening();

// boucle principale (comme le loop() Arduino)
while(1) {
   // Arrêt écoute
   radio.stopListening();

   printf("Envoi\n");
   message = millis();

   // Envoi avec vérification d'erreur
   if(!radio.write(&message, sizeof(unsigned long))) {
      printf("Erreur : write.\n");
   }

   // Mise en écoute
   radio.startListening();

   // Enregistrement du début du délais
   debut = millis();
   timeout = false;

   // Tant que rien à lire
   while (!radio.available()) {
      if (millis()-debut > 600) {
         timeout = true;
         break;
      }
   }

  // abandon pour cause de délais écoulé ?
   if(timeout){
      printf("Erreur : timeout.\n");
   } else {
      // lecture des données reçues
      radio.read(&reception, sizeof(unsigned long));
      // enregistrement du moment de la réception
      fin = millis();

      // Affichage
      printf("Message: %lu, reponse: %lu, temps complet: %lu\n",
                        message, reception, fin-message);
   }

   // dodo 1 seconde
   sleep(1);
 }
 return 0;
}
```

On retrouve littéralement toutes les méthodes utilisées et toute la logique du programme avec simplement quelques petits changements de-ci de-là. Point de `Serial.println()` ici, par exemple, mais `printf()` affichant les messages directement sur la console. En grande majorité, les lignes de codes sont tout à fait identiques.

Pour compiler notre code et obtenir rapidement un programme exécutable, le plus simple est de tout bonnement reposer sur ce qui est mis à notre disposition, et donc les exemples livrés avec RF24. Nous nous plaçons donc dans le sous-répertoire `examples_linux`, y ajoutons notre fichier que nous appelons `rfsend.cpp` et modifions le fichier `Makefile` en changeant la ligne :

```
# define all programs
PROGRAMS = gettingstarted gettingstarted_call_
response transfer pingpair_dyn
```

en

```
# define all programs
PROGRAMS = rfsend gettingstarted gettingstarted_
call_response transfer pingpair_dyn
```

Ainsi, en plus des autres exemples, notre `rfsend.cpp` sera compilé en une commande `rfsend`. Il ne nous reste plus qu'à nous plier d'un `make` pour construire tout cela, puis, lancer la commande en question :

```
$ sudo ./rfsend
Go go go!
================ SPI Configuration ================
CSN Pin          = CE0 (PI Hardware Driven)
CE Pin           = Custom GPIO22
Clock Speed      = 8 Mhz
================ NRF Configuration ================
STATUS           = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0
RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1     = 0xf0f0f0f0d2 0xf0f0f0f0e1
RX_ADDR_P2-5     = 0xc3 0xc4 0xc5 0xc6
TX_ADDR          = 0xf0f0f0f0d2
RX_PW_P0-6       = 0x20 0x20 0x00 0x00 0x00 0x00
EN_AA            = 0x3f
EN_RXADDR        = 0x03
RF_CH            = 0x4c
RF_SETUP         = 0x03
CONFIG           = 0x0e
DYNPD/FEATURE    = 0x3f 0x06
Data Rate        = 1MBPS
Model            = nRF24L01+
CRC Length       = 16 bits
PA Power         = PA_LOW
Envoi
Message: 893376, reponse: 893376, temps complet: 1
Envoi
Message: 894377, reponse: 894377, temps complet: 2
```

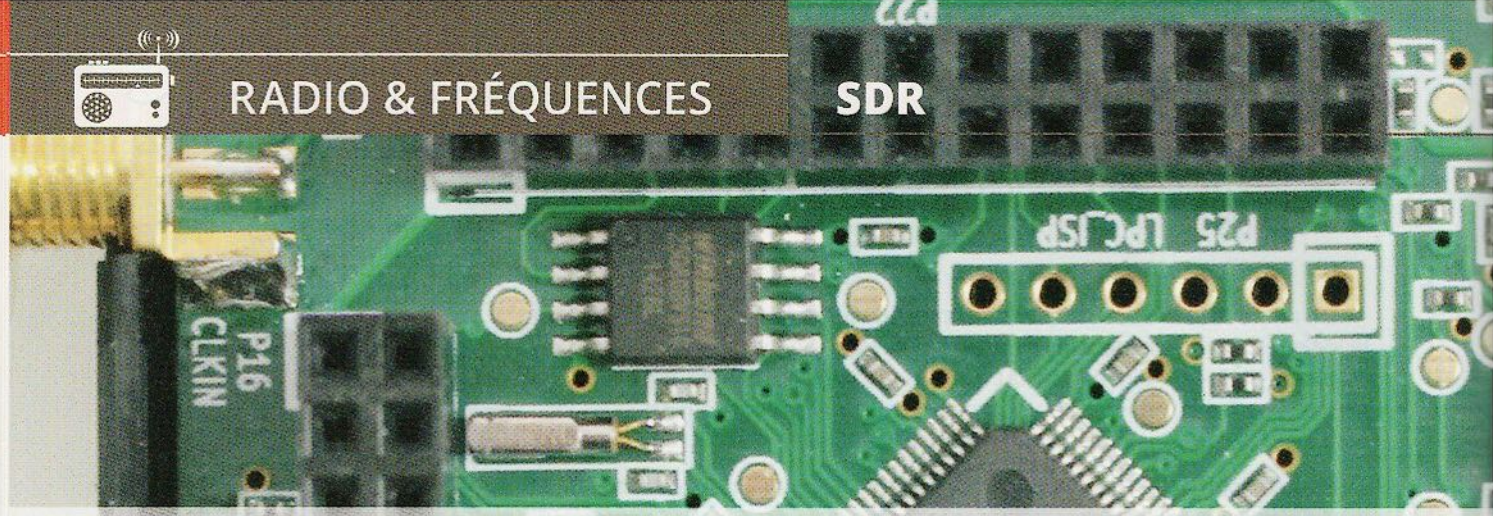
Le comportement est parfaitement similaire à ce que nous avons pu constater entre les deux cartes Arduino, et pour cause, c'est la même bibliothèque et le même code de démonstration !

3. LE RESTE C'EST POUR VOUS !

Cet article succinct ne présente qu'une façon d'approcher le support des nRF24L01, mais il en existe d'autres. L'une d'entre elles consiste à utiliser le module Python dont l'utilisation est très similaire aux notions vues en C++. Notez cependant que c'est là avant tout une question de préférences, car contrairement à ce qu'on pourrait penser, le C/C++ n'est pas forcément le langage le plus compliqué à utiliser. Essayez par exemple de convertir un entier long comme une durée en millisecondes en séquences d'octets (`bytearray`) en Python et vous vous rendrez rapidement compte que le C/C++ fait cela d'une façon bien plus naturelle.

En termes de perspectives, le présent article en compagnie du précédent, vous a normalement donné les clés pour mettre en œuvre tous types de réalisations. On peut aisément imaginer ainsi un ensemble de capteurs sur base Arduino équipés de nRF24L01, régulièrement interrogés pour collecter des mesures ou encore une simple télécommande multicanaux permettant de contrôler les luminaires ou des actionneurs mécaniques. Les principaux avantages des nRF24L01 sont leur faible coût et la portée de la communication. Là encore, il y a de quoi imaginer bien des applications...

Enfin, il serait également dommage de ne pas saisir l'occasion de s'essayer au C/C++ sur Raspberry Pi et ainsi découvrir les joies de la programmation « native » afin de réaliser ses propres outils en ligne de commandes. Vous pourrez constater que les choses ne sont pas très différentes de ce que vous connaissez déjà sur Arduino et que la frontière est davantage imaginaire que pratique... DB



POUR ALLER PLUS LOIN EN RADIO LOGICIELLE : HACKRF ONE

Denis Bodor

Nous avons traité précédemment dans ce magazine du fantastique domaine de la radio logicielle (SDR) et de l'utilisation de récepteurs DVB-T pour recevoir toutes sortes de signaux radio. Le matériel disponible s'est grandement diversifié depuis lors avec de plus en plus de récepteurs construits sur la même base RTL-SDR, mais conçus dans l'optique d'une utilisation radio. Une autre plateforme, moins économique, permettant d'aller plus loin encore est également disponible : le HackRF One de Great Scott Gadgets.

Dans le second numéro de *Hackable*, il y a maintenant plus de deux ans, nous avions exploré le monde de la radio logicielle via l'utilisation de petites clés USB initialement destinées à la réception de la TV numérique TNT (DVB-T). Depuis, ces périphériques n'ont pas vraiment changé, mais se sont sensiblement améliorés.

En effet, bon nombre d'amateurs et d'électroniciens ont décidé, plutôt que d'utiliser du matériel originellement prévu pour une tout autre utilisation, de reprendre le même concept et la même architecture matérielle tout en apportant des petites modifications destinées à en améliorer le fonctionnement. On retrouve à présent ces récepteurs RTL-SDR, vendus sous différentes marques, un rien plus chers que les modèles génériques de réception TNT, garantissant davantage de qualité, de stabilité et parfois de fonctionnalités.

Tous ces périphériques se distinguent les uns des autres par leurs gammes de fréquences utilisables (de quelques mégahertz à gigahertz), la fréquence d'échantillonnage impactant la largeur de bande (gamme de fréquences qu'il est possible de « voir »), la stabilité de l'horloge vis-à-vis de facteurs externes comme la température, ou encore la qualité de fabrication et la gestion de la dissipation thermique (si le périphérique chauffe trop, il change de comportement et la réception est perturbée).

Le HackRF One se présente sous la forme d'un boîtier plastique noir disposant d'un connecteur micro-USB, quelques leds et boutons et de connecteurs SMA : un pour l'antenne et deux pour des signaux d'horloge optionnels.

Mais l'objet du présent article ne concerne pas ces produits, mais une catégorie de périphériques d'un niveau supérieur. En effet, qu'il s'agisse d'un récepteur USB TNT à 10€, d'une version améliorée comme le NESDR SMArt, le NESDR Mini ou le NESDR Nano, à quelques 30€, ou encore des équipements plus performants comme le SDRPlay, le AirSpy ou le FunCube entre 150 et 200€, tous ont en commun une caractéristique : ce sont des récepteurs.

Le HackRF One de Great Scott Gadgets, comme le BladeRF de Nuand ou encore les USRP d'Ettus Research ne se limitent pas à la réception de signaux, mais peuvent également émettre. C'est une tout autre catégorie de matériels.

Je me dois de préciser ici que, dès lors qu'il s'agit d'émission de signaux radio, nous entrons dans un domaine strictement encadré. Quels que soient le matériel utilisé, les fréquences utilisables, la puissance et d'autres caractéristiques, il s'agit d'éléments régis au niveau légal. En l'absence d'une licence radioamateur, votre champ d'action est et **DOIT** être **fortement** limité. Techniquement, vous n'avez de base le droit d'utiliser que les

bandes ISM (Industriel, Scientifique et Médical), celles-là même utilisées par le Wifi (2,4 à 2,5 GHz et 5,725 à 5,875 GHz), les appareils sans fil comme les carillons et capteurs (433,05 à 434,79 MHz) ou encore les bonnes vieilles télécommandes de radio-modélisme et autres téléphones sans fil (26,957 à 27,283 MHz). Que les choses soient claires : **vous DEVEZ connaître et respecter la législation en vigueur dans votre pays**, avec ou sans licence radio-amateur ! Ceci ne souffre d'aucune exception, pas pour 5 minutes, pas même pour une demi-seconde ou « juste pour voir » !

1. PETIT RAPPEL SUR LA RADIO LOGICIELLE

Historiquement, la réception et l'émission de signaux radio ont été rendues possibles par la création de montages électroniques, d'abord à base de lampes puis de transistors. La technique consiste à émettre des ondes électromagnétiques générées par un circuit matériel et à les réceptionner de la même façon avec un autre circuit.

Pour véhiculer des informations via ces ondes, qu'il s'agisse de sons, de signaux intermittents (type morse) ou de données, on les mélange avec le signal de base selon différents procédés appelés modulations. De l'autre côté de la liaison, la tâche de celui qui réceptionne le signal consiste à extraire (démoduler) cette information du signal afin de la restituer le plus fidèlement possible. L'exemple typique de ce type de technique est le cas de la diffusion de programmes radiophoniques, généralement désignés par les termes de « radio FM ». D'un côté, nous avons un émetteur modulant un signal audio sur une fréquence donnée (entre 87,5 et 108 MHz) et de l'autre des récepteurs, démodulant le signal et restituant le son (postes radio, autoradios, etc.). Tout ceci est entièrement matériel.

La radio logicielle, ou SDR en anglais pour *Software Defined Radio*, délègue une partie du traitement du signal à des programmes. La réception se fait, bien entendu, avec un matériel dédié, mais les données brutes desquelles doivent être extraites les informations sont numérisées par un convertisseur analogique-numérique et envoyées à un système informatique. C'est là que l'aspect logiciel prend le relais, pour toutes les tâches de traitement comme le filtrage, la décimation (sous-échantillonnage pour ne garder qu'une partie des données brutes), la démodulation, le décodage et la restitution de l'information transportée.

Cette technique n'est pas récente, mais jusqu'à il y a environ 5 ans les équipements permettant de recevoir des signaux radio, les numériser et les envoyer à un PC, par

L'emplacement pour l'antenne utilise un connecteur SMA comme la plupart des équipements SDR, et non RP-SMA, plus courant sur les équipements Wifi par exemple. Différentes leds permettent de visualiser l'état et le fonctionnement du périphérique.

exemple, étaient relativement rares, coûteux et implicitement réservés à un public averti. Les choses ont changé il y a quelques années grâce à quelques développeurs Linux qui se sont aperçus que les périphériques USB de réception TNT (DVB-T) n'étaient constitués que de deux éléments matériels : un tuner chargé de recevoir les signaux radio (puce Elonics E4000), et un convertisseur analogique-numérique (Realtek RTL2832U) envoyant le signal numérisé via une connexion USB. L'ensemble du traitement du signal jusqu'à l'affichage de l'image était donc fait de façon entièrement logicielle et il s'est avéré possible de recevoir bien plus que la TNT avec ces périphériques : le projet RTL-SDR venait de naître.

La disponibilité et le faible coût de ce qui est à présent désigné sous le terme de « récepteur RTL-SDR » ont permis à toute une légion d'amateurs, de programmeurs et d'ingénieurs d'expérimentés dans le domaine de la radio logicielle, d'explorer librement le domaine. En quelques années, une masse importante de projets, de documentations, de logiciels et de supports pédagogiques ont vu le jour.

Un certain nombre d'utilisateurs ont également entrepris de pousser le matériel à ses limites en y apportant des modifications permettant d'en augmenter la qualité ou d'ajouter des fonctionnalités. Ces hacks et modifications ont conduit à une nouvelle génération de récepteurs USB, intégrant ces modifications et permettant, pour un coût un peu supérieur, d'avoir un matériel encore plus adapté à la réception radio logicielle, mais se basant sur les mêmes composants et en particulier le même convertisseur analogique-numérique Realtek RTL2832U.

Pour utiliser un tel matériel, qu'il s'agisse d'un récepteur TNT/DVB-T ou d'un produit revu et corrigé, il suffit de le connecter à son PC, son Mac, sa tablette Android ou sa Raspberry Pi en USB, puis d'utiliser l'un des nombreux logiciels gratuitement disponibles (certains sont des logiciels libres) : SDR#, Gqrx, HDSDR, Linrad, SDR Touch, etc.

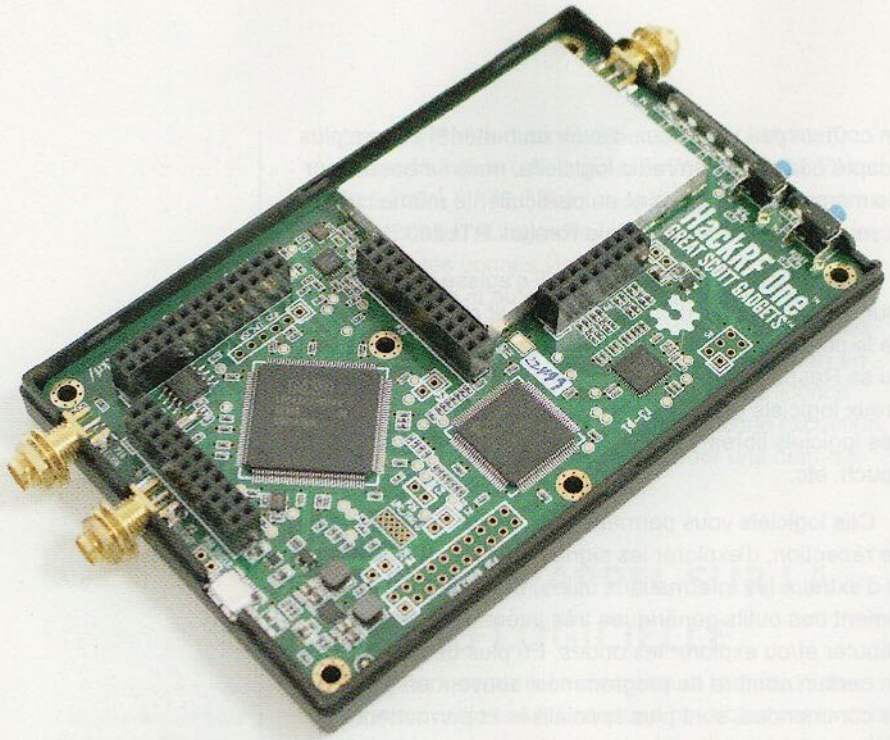
Ces logiciels vous permettent de régler la fréquence de réception, d'explorer les signaux, de les démoduler et d'extraire les informations utiles. Ce sont généralement des outils génériques très intéressants pour débuter et/ou explorer les ondes. En plus de ces outils, un certain nombre de programmes, souvent en ligne de commandes, sont plus spécialisés et permettent de collecter des données, décoder des signaux spécifiques ou encore de calibrer votre récepteur.

Enfin, nous avons l'incontournable GNU Radio, une boîte à outils extrêmement complète doublée d'une interface graphique (*GNU Radio Companion*) permettant de construire de toutes pièces toute la partie logicielle d'un ensemble de réception SDR, en fonction de ses besoins.

Le gain de popularité du domaine, l'investissement de départ minime, la disponibilité du matériel et la masse d'information en ligne disponible (en anglais) rendent aujourd'hui l'initiation à la radio logicielle et à l'exploration des ondes plus que jamais accessible par tout un chacun. Mais entre recevoir des signaux et en émettre, il y a presque tout un monde...

2. LE HACKRF ONE

Début 2012, Michael Ossmann, initialement ingénieur en sécurité informatique, décide de créer un périphérique SDR à partir de zéro, plus performant que les récepteurs RTL-SDR et surtout capable d'émettre en plus de recevoir. Le périphérique, nommé HackRF One, voit ainsi le jour via l'entreprise que Michael a créée, Great Scott Gadgets, pour fabriquer une autre de ses créations, l'Ubertooth, une plateforme d'expérimentation autour des technologies Bluetooth.



À l'intérieur du HackRF One, on trouve les connecteurs d'extension permettant d'étendre les fonctionnalités ou de connecter, par exemple, un oscillateur plus stable et plus précis pour les applications de type GPS et GSM.

Le HackRF One, comme l'Ubertooth, est un matériel libre (ou *open source hardware* en anglais). Bien que la définition soit relativement floue (ou interprétable à sa convenance), un matériel libre ou open source, comme un logiciel libre ou open source, offre une ouverture et des libertés quant à son utilisation, sa reproduction et sa modification. Dans le cas du HackRF One, Michael fournit l'ensemble des éléments qui composent son périphérique (schémas, outils, documentation, circuits, logiciels internes, etc.) tout en permettant leur reproduction et leur modification. Techniquement et légalement, n'importe qui disposant des connaissances et des moyens adéquats peut construire, produire et vendre des HackRF One (ce qui est d'ailleurs effectivement arrivé puisque Gareth Hayes a lancé, il y a 2 ans, une campagne de financement collaboratif Indiegogo pour produire le HackRF Blue, un clone du HackRF One à coût réduit).

Le HackRF One est disponible à la vente en France sur Amazon par Great Scott Gadgets directement ou par NooElec, ainsi que chez Passion Radio Shop. L'objet vous coûtera entre 270€ et 340€ selon la source et, vous l'avez compris, ce n'est pas tout à fait la même tranche budgétaire qu'un récepteur RTL-SDR à moins de 30€. Il est vain de comparer les deux types de produits pour la simple raison que le HackRF permet d'émettre et ne repose pas du tout sur la même architecture. Le fait est que ce périphérique est le moins cher de sa catégorie, face au BladeRF x40 ou x115 de Nuand à quelques 500€ ou 1000€, ou encore aux USRP d'Ettus Research entre 700€ et plus de 5000€ !

En plus de la capacité d'émettre des signaux, le HackRF possède des caractéristiques très intéressantes, même en réception :

- Une plage de fréquences très importante, entre 1 MHz et 6 GHz. Ceci, comparé aux récepteurs RTL-SDR avec une gamme généralement située entre 24 Mhz et 1,7 GHz donne accès, entre autres, aux bandes utilisées par le Wifi, le Bluetooth ou encore les modules à base de nRF24L01.
- Une fréquence d'échantillonnage jusqu'à 20 millions d'échantillons par seconde (20 MS/s). Cette caractéristique, qui n'est que de 2,4 MS/s est la quantité de mesures par seconde qui peuvent être faites en numérisant le signal. Ceci impacte, entre autres choses, la largeur de bande qu'il est possible d'observer en une fois. Avec un HackRF One et en calant le récepteur, par exemple sur 100 Mhz, il sera donc possible

d'observer les signaux entre 90Mhz et 110Mhz, contre 97,6 Mhz et 102,6 Mhz avec un récepteur RTL-SDR.

- Tout comme avec les matériels RTL-SDR, les échantillons sont encodés sur 8 bits. Ceci représente la résolution maximum de chaque échantillon ou tranche numérisée. Comparé à un BladeRF et ses échantillons 12 bits, le HackRF One se place donc en retrait sur ce point, mais pour la moitié du prix.
- Des amplificateurs contrôlables à plusieurs étapes du cheminement du signal. Ceci permet de régler le gain (l'amplification) avec une plus grande précision et donc d'arriver à capter des signaux faibles, normalement noyés dans le bruit de fond, tout en limitant la distorsion pour les signaux plus forts.
- Plusieurs connecteurs internes permettant d'ajouter des cartes d'extension (existantes ou faites maison). Un excellent exemple est le PortaPack H1 permettant d'utiliser le HackRF One de manière autonome en fournissant un écran, une batterie, une interface utilisateur et différentes applications.
- Une possibilité logicielle pour alimenter une antenne active ou un amplificateur externe directement par le connecteur d'antenne. Le HackRF One est en mesure d'alimenter ainsi un montage en 3,3V jusqu'à 50mA en courant continu. Un exemple est le LNA4ALL, un petit amplificateur faible bruit prenant place entre le connecteur d'antenne du HackRF et l'antenne elle-même. Celui-ci peut être alimenté de façon externe ou, dans sa configuration Bias-T (ou *bias tee*), directement par le HackRF One.

Tout ce charabia technique n'est sans doute pas clair si vous débutez en radio logicielle ou n'avez, pour l'heure, fait que quelques essais avec un récepteur RTL-SDR et une application comme SDR# ou Gqrx. Dans les grandes lignes, le HackRF One, du point de vue de la réception, peut opérer sur bien plus de fréquences, en vous permettant de voir plus de signaux à la fois, tout en offrant davantage de fonctionnalités. Ceci, en plus de la capacité d'émettre des signaux, justifie à mon sens parfaitement l'investissement de quelques 300€... à condition que vous en fassiez quelque chose, bien entendu.

3. UTILISATION DU HACKRF ONE

Le HackRF One, comme le BladeRF ou les récepteurs RTL-SDR, bénéficie d'un support dans la plupart des logiciels que ce soit sous Windows, Mac OS ou GNU/Linux. Attention cependant, car si tous les récepteurs RTL-SDR, quel que soit leur coût, origine et qualité, partagent le même support (pilote, prise en charge, outils spécifiques, etc.) et sont compatibles entre eux, ce n'est pas le cas du HackRF One. En d'autres termes, il faut que l'application ou l'outil en ligne de commandes que vous allez utiliser, supporte directement le HackRF One.

Les applications génériques, de SDR# à GNU Radio en passant par Gqrx et RF Analyzer (Android) prennent généralement en charge tous les périphériques les plus populaires, et donc le HackRF One. Certains outils cependant, souvent utilisables en ligne de commandes, sont spécifiques aux périphériques RTL-SDR. C'est le cas, par exemple de `rtl_power` dont nous avons parlé dans *Hackable n°9*, de Dump1090 pour la réception des signaux ADSB des aéronefs ou encore de `rtl_433` permettant de décoder et d'afficher les messages de très nombreux produits utilisant les fréquences autour de 433 Mhz (télécommandes, sondes météo, carillon, etc.). Il existe cependant parfois des déclinaisons « HackRF » de ces outils, souvent parfaitement fonctionnelles, en attendant

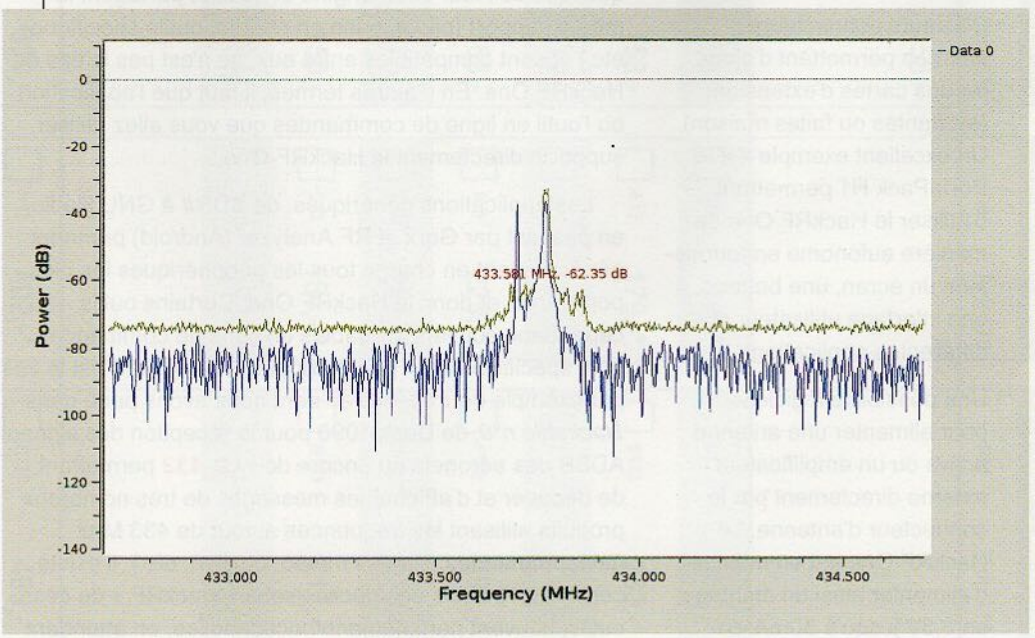
que l'ensemble des outils et applications ne passent sur SoapySDR, un support unique pour plus d'une douzaine de périphériques différents.

Il n'est pas possible, dans cet article, de couvrir tout ce qu'il est envisageable de faire avec un HackRF One, car l'étendue des possibilités offertes par un simple périphérique RTL-SDR est déjà énorme, et si l'on ajoute la capacité d'émettre, cela démultiplie d'autant les possibilités. Nous allons donc nous concentrer sur un exemple pratique simple et classique consistant à repérer un signal, l'enregistrer et le réémettre pour prendre la place de l'émetteur initial.

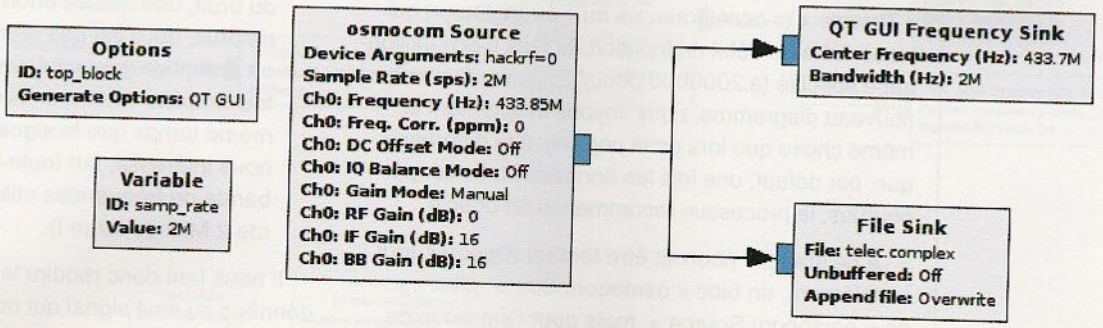
Notre victime sera, une fois n'est pas coutume, une télécommande de prise électrique. De marque AquaFORTE, celle-ci utilise une fréquence de 433,92 Mhz et une modulation similaire à celle que nous avons étudiée dans le numéro 12 avec un autre modèle. Le message envoyé par la télécommande à la prise et la façon dont il est encodé n'est pas important ici. Nous allons, en effet, nous en tenir à l'émission radio elle-même et profiter des avantages offerts par le HackRF One pour enregistrer le signal et le rejouer.

Pour cela, nous allons utiliser GNU Radio Companion, un HackRF One et, idéalement, un récepteur RTL-SDR pour vérifier notre transmission. Je pars ici du principe que si vous vous intéressez au HackRF One, vous avez sans doute déjà investi, comme moi, dans un ou plusieurs récepteurs économiques de ce type (si ce n'est pas le cas, ce n'est sans doute pas une bonne idée de commencer avec un HackRF One). Ce complément au HackRF One est relativement important pour vous assurer que vous transmettez bel et bien le bon signal sur la bonne fréquence.

La première étape de notre expérience consistera à, tout d'abord, repérer le signal qui nous intéresse. Pour cela, dans GNU Radio Companion nous utiliserons un bloc « osmocom Source » en spécifiant une fréquence de



Le graphique présente deux pics lorsque la télécommande est utilisée. Celui de gauche n'a rien à voir avec le signal, il s'agit d'un artefact du convertisseur analogique/numérique appelé « DC spike ». Le pic de droite en revanche provient de la télécommande et est pile sur 433,92 Mhz.



Notre diagramme GNU Radio Companion nous permet d'utiliser le HackRF One et d'enregistrer les signaux reçus sous forme de données brutes dans un fichier, tout en les visualisant en temps réel.

réception proche de celle de la télécommande, sans pour autant être précisément la même. Ici, la mention « 433,92 Mhz » est présente sur la télécommande elle-même et nous réglons donc la fréquence sur 433,85 Mhz (`433.85e6`). La fréquence d'échantillonnage ou *sample rate* peut être réglée jusqu'à 20000000 avec le HackRF One, mais nous nous contenterons de 2000000.

Pour observer le signal, nous ajoutons un bloc « QT GUI Frequency Sink », puis connectons les deux blocs en cliquant sur l'étiquette bleue du premier puis celle du second. Il ne reste plus qu'à exécuter le diagramme et observer ce qui se passe en pressant le bouton de la télécommande.

Bien entendu, nous voyons apparaître le signal de la télécommande, précisément sur la fréquence attendue. Pourquoi ne pas avoir réglé la réception sur cette fréquence ? Tout simplement en raison de l'autre pic qui apparaît de façon continue. Celui-ci provient du convertisseur ADC présent dans le HackRF One, c'est le *DC spike*, systématiquement présent en réception SDR. Certains supports dans les logiciels SDR permettent de masquer ce signal, mais techniquement parlant il est toujours présent. Or, ici, nous voulons enregistrer ce que nous recevons et le rejouer, si le *DC spike* est mélangé au signal convoité, il sera plus difficile de le filtrer et de réutiliser notre enregistrement.

Pour stocker notre réception dans un fichier, inutile de démoduler quoi que ce soit. Il nous suffit d'enregistrer les données brutes (*raw*) fournies par le matériel. Celles-ci prennent la forme d'une série de deux valeurs en virgule flottante, formant de concert un nombre complexe, désigné sous le nom de données I/Q ou de signal numérique en quadrature. Je ne rentrerai pas dans le détail ici, car ce n'est pas pertinent pour l'exercice, mais j'ai expliqué ce principe dans le second numéro de la revue.

Pour enregistrer ces données, nous ajoutons à notre diagramme un bloc « File Sink » en précisant un nom de fichier dans ses propriétés et le connectons à la sortie du bloc « osmocom Source ». À présent, ce que nous allons voir en lançant l'exécution sera enregistré immédiatement dans le fichier.

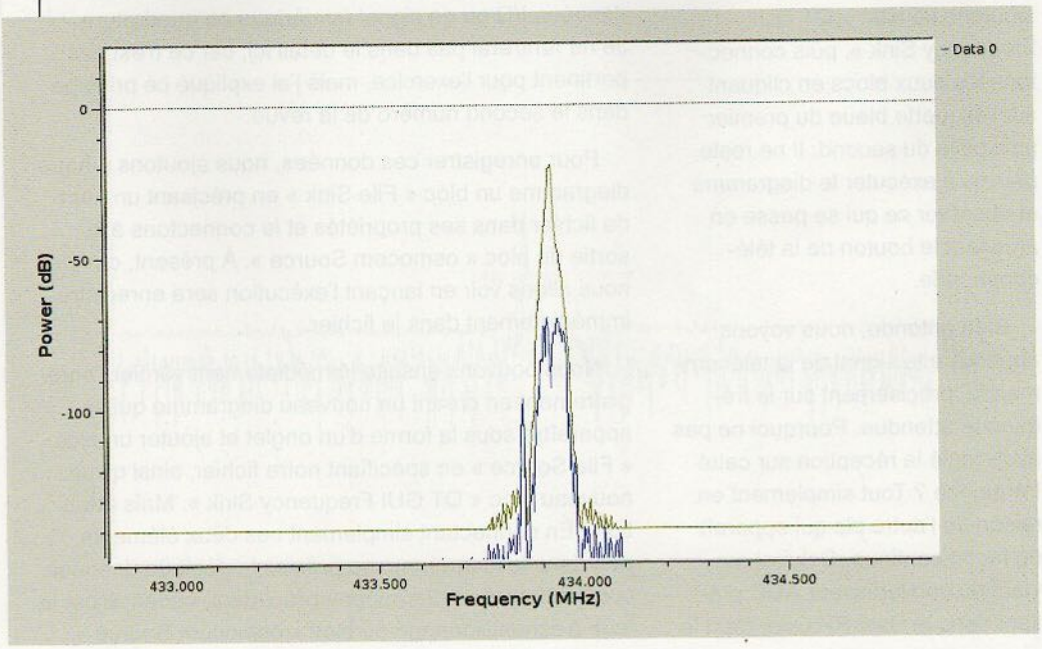
Nous pouvons ensuite immédiatement vérifier l'enregistrement en créant un nouveau diagramme qui va apparaître sous la forme d'un onglet et ajouter un bloc « File Source » en spécifiant notre fichier, ainsi qu'un nouveau bloc « QT GUI Frequency Sink ». Mais attention ! En connectant simplement ces deux éléments, rien dans le diagramme ne précise le débit de données, contrairement au diagramme précédent, cadencé par le taux d'échantillonnage du bloc « osmocom Source ».

Il nous faut donc ajouter un nouveau bloc, « Throttle » (« accélérateur » ou « étrangleur ») qui va contrôler le débit en fonction du taux d'échantillonnage spécifié (à 2000000 donc). En exécutant notre nouveau diagramme, nous voyons exactement la même chose que lors de la capture, à la différence que, par défaut, une fois les données du fichier parcourues, le processus recommence en boucle.

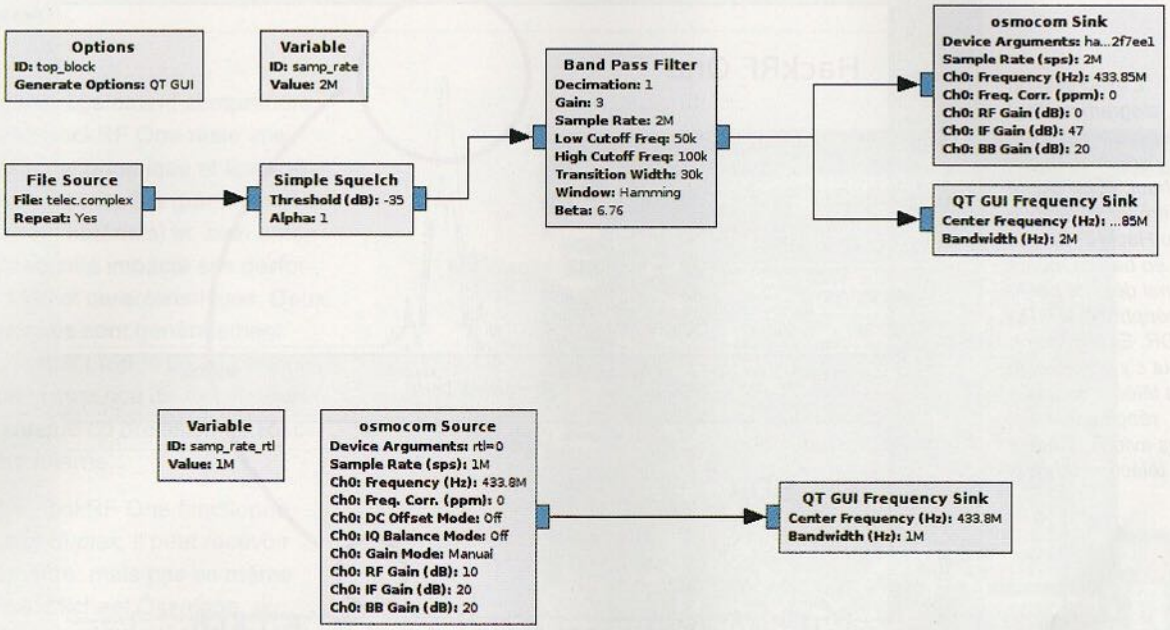
À ce stade, il pourrait être tentant d'ajouter, tout simplement, un bloc « osmocom Sink », alter ego de « omsocom Source », mais pour l'émission de données/signaux, et tout bonnement le connecter à la sortie du bloc « File Source » (en supprimant « Throttle »). Mais nous avons plusieurs problèmes :

- L'enregistrement ne contient pas uniquement le signal qui nous intéresse, mais également le *DC spike* dont nous avons parlé précédemment (et peut-être d'autres signaux). En envoyant les données au HackRF One, nous allons donc aussi émettre un signal sur la fréquence correspondant à ce pic.
- Nous devons amplifier le signal reçu avant de l'émettre, car il ne peut pas être transmis aussi faiblement qu'il a été enregistré.
- Notre enregistrement contient du bruit, une masse énorme de bruit, qui n'est pas gênant en réception, mais qui, une fois amplifié va être émis en même temps que le signal qui nous intéresse, sur toute la bande de fréquences utilisée (de 2 Mhz de large !).

Il nous faut donc réduire les données au seul signal qui nous intéresse et pour ce faire, nous allons commencer par ajouter un bloc « Simple Squelch ». Celui-ci permet de rendre muet des signaux s'ils sont en dessous d'un certain niveau de puissance. En rejouant notre enregistrement, nous pouvons facilement visualiser la puissance de notre signal exprimée en décibels (dB). Nous pouvons alors utiliser cette valeur dans les propriétés du bloc en guise de seuil (*threshold*), ici à -35 dB.



En utilisant notre fichier et en filtrant les signaux qu'il contient, nous arrivons à isoler uniquement ce qui nous intéresse. On voit ici clairement que seule une bande de fréquences données est conservée. Le bruit en bleu est limité à quelques -80 dB alors que notre signal, en brun/vert monte à -7 dB. Notez la présence du DC spike, mais très fortement atténué.



Ce diagramme final permet d'utiliser notre enregistrement, filtrer son contenu en ne gardant que ce qui nous intéresse pour ensuite l'envoyer au HackRF One. Notez les deux blocs inférieurs formant un ensemble totalement séparé nous permettant, avec un récepteur RTL-SDR en supplément, de visualiser le signal émis par le HackRF One.

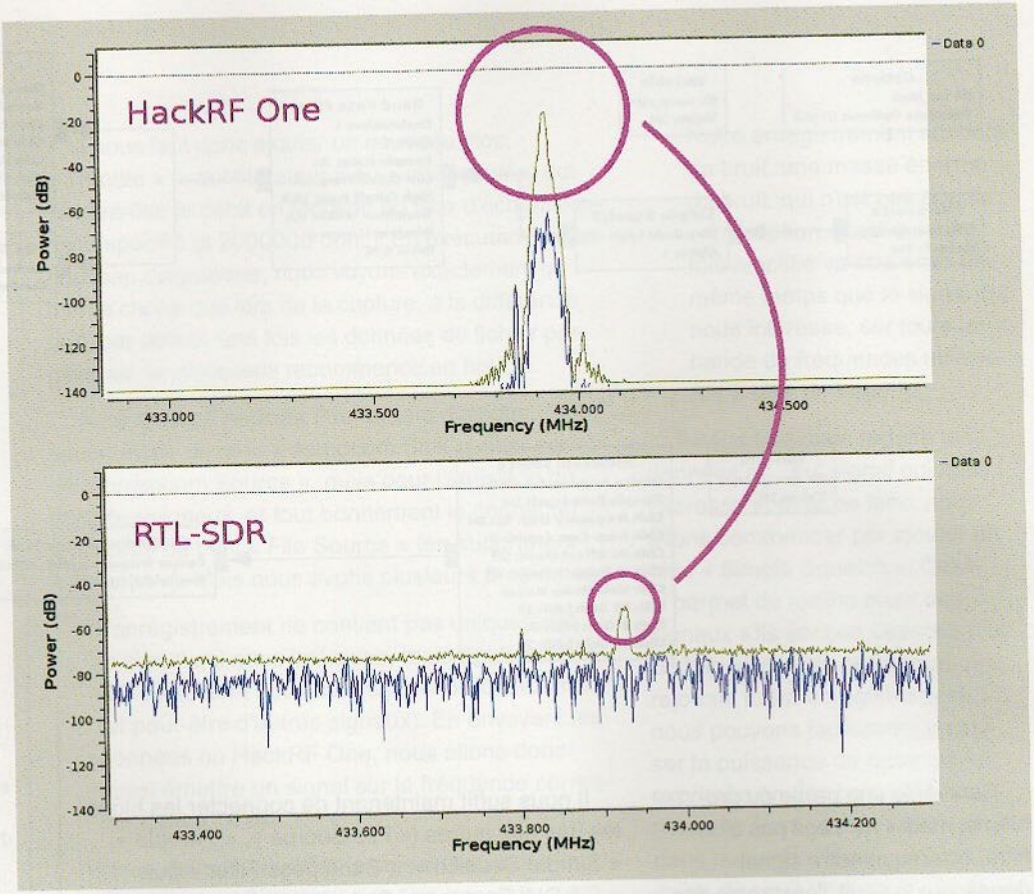
Ceci règle une partie du problème, mais il ne s'agit pas d'un filtre, et lorsque notre signal dépassera le seuil, l'ensemble des données sera utilisé, y compris le bruit et le *DC spike*. De plus, notre signal est extrêmement faible, nous devons l'amplifier. Un bloc appelé « Band pass Filter » nous permet de régler ces deux points en une fois puisqu'il permet de ne laisser passer qu'une « bande » donnée de fréquence et de spécifier un gain pour augmenter la puissance du signal qui nous intéresse.

En ajoutant ce bloc, nous pouvons spécifier dans ses propriétés une fréquence de coupure haute et basse ainsi qu'une largeur de transition (pour avoir une atténuation graduelle en limite de la bande spécifiée). Ici, j'utiliserai respectivement les valeurs de 50 KHz, 100 Khz et 30 Khz, limitant effectivement le filtrage à une bande de 50 Khz de large, isolant notre signal.

Il nous suffit maintenant de connecter les blocs les uns aux autres (« File Source », « Throttle », « Simple Squelch », « Band pass Filter » puis « QT GUI Frequency Sink ») et d'exécuter le diagramme pour visualiser le résultat. Bien entendu, les propriétés définies vont totalement dépendre du signal reçu et de l'enregistrement effectué. À ce stade, il s'agit de jouer avec ces réglages jusqu'à obtenir un signal le plus « propre » possible.

Une fois satisfait du résultat, il est temps de passer à l'émission en ajoutant le bloc « osmocom Sink », en parallèle à « QT GUI Frequency Sink » et en supprimant « Throttle » devenu alors inutile. Dans les propriétés, on prendra soin de régler la fréquence d'émission exactement sur celle que nous avons utilisée pour la capture et l'enregistrement. N'oubliez pas, notre signal n'est pas pile sur cette fréquence. On règlera également l'amplification (IF gain) entre 1 et 47, en commençant par une valeur réduite et en augmentant jusqu'à obtenir une réaction de la prise télécommandée.

À ce stade, si vous possédez un récepteur RTL-SDR, il est intéressant de l'utiliser en ajoutant un nouveau bloc « omsocom Source »



L'exécution du diagramme final nous présente deux graphiques. En haut, notre signal filtré tel qu'envoyé au HackRF One et en bas ce même signal détecté par le périphérique RTL-SDR. Et comme on peut s'y attendre, la prise télécommandée réagit comme si nous avions utilisé la télécommande...

et en le connectant à un nouveau bloc « QT GUI Frequency Sink ». Ceci nous permettra de voir le signal effectivement émis par le HackRF One. Pour que Gnu Radio puisse faire la différence entre les deux périphériques utilisés, nous devons changer les propriétés des deux blocs osmocom. On règlera le « Device Argument » de l'un à `hackrf=` suivi de la fin du numéro de série du HackRF One et l'autre à `rtl=0` pour utiliser le premier récepteur RTL-SDR présent.

En exécutant le diagramme, la prise télécommandée réagit et nous visualisons aussi bien le signal utilisé par le HackRF One que celui détecté par le récepteur RTL-SDR. On pourra ensuite ajuster le gain dans les propriétés de « osmocom Sink » pour augmenter la puissance du signal si besoin.

Cette démonstration est transposable à bien des produits utilisant une communication radio. Sonnette sans fil, télécommande de porte de garage, sondes diverses, et même parfois certaines télécommandes d'ouverture centralisée de véhicule. Inutile cependant de vous le rappeler, dans tous les cas, n'utilisez et ne testez que vos propres produits et installations, et restez **toujours** dans les bandes de fréquences ISM !

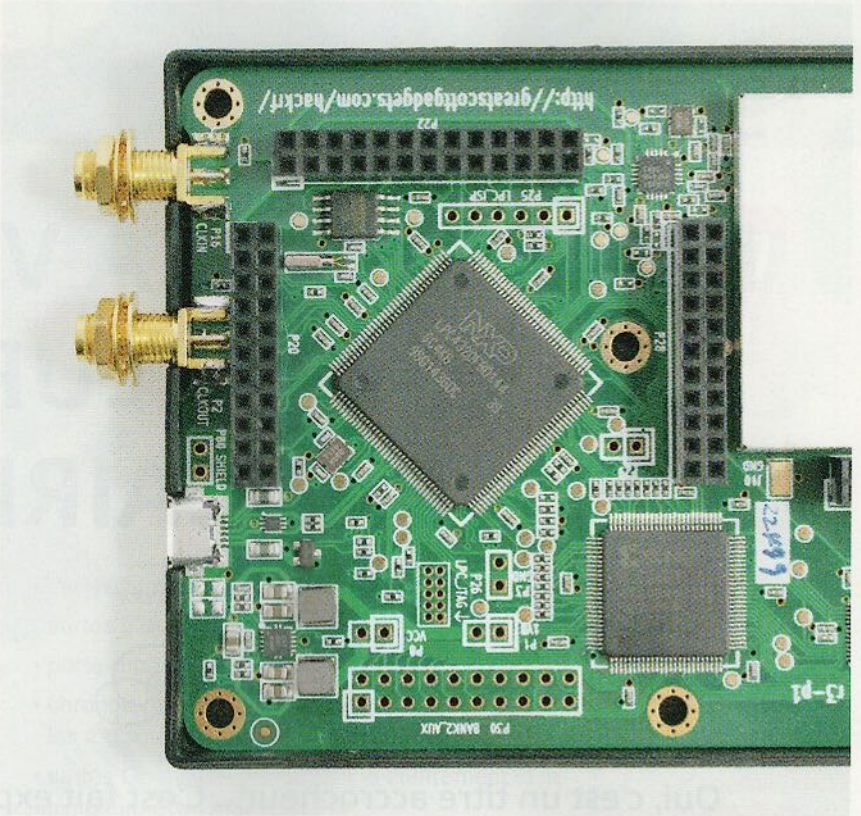
CONCLUSION

Comme vous pouvez le constater, le HackRF One bien que pouvant être utilisé comme un récepteur RTL-SDR avec les applications courantes, ne délivre son réel potentiel qu'à condition de mettre en œuvre des outils et des techniques plus avancés, nécessitant généralement davantage d'investissement personnel. Il ne s'agit pas d'un jouet et même si de nos jours certains ont tendance à dépenser facilement quelques 600€ pour un smartphone dernier cri pour remplacer un précédent modèle encore parfaitement fonctionnel, ces quelques 300€ représentent une somme à ne pas investir à la légère.

Il faut également comprendre que le HackRF One reste une solution économique et financièrement accessible (par rapport à d'autres matériels) et, bien entendu, que cela impacte ses performances et caractéristiques. Deux reproches sont généralement évoqués à propos de ce périphérique : l'absence de *full-duplex* et le manque de précision de l'oscillateur interne.

Le HackRF One fonctionne en *half-duplex*, il peut recevoir et émettre, mais pas en même temps. Michael Ossmann, au moment où j'écris ceci, finalise une carte d'extension pour le HackRF One, appelée *Neapolitan Board*, permettant d'ajouter cette fonctionnalité au HackRF One. Pour le coût d'un simple ajout, il sera donc bientôt possible de corriger ce problème sans avoir à investir, par exemple, dans un Nuand BladeRF.

Concernant la stabilité de l'oscillateur, dans l'état actuel des choses, la plupart des utilisations envisageables avec un HackRF One sont parfaitement réalisables. Si en revanche, vous souhaitez étudier des signaux et standards nécessitant une grande précision (GPS, GSM, etc.), vous allez rencontrer des difficultés. Là encore, une modification simple du HackRF One est possible, en lui fournissant un signal d'horloge de 10 Mhz plus stable (via l'entrée CLKIN ou le port d'extension). Ceci prend généralement la forme d'un oscillateur TCXO (0,5 ppm), monté sur un circuit et enfiché dans le HackRF One. Je n'en ai pas encore fait l'essai, ma commande est toujours quelque part entre Shenzhen et l'Alsace (j'espère).



L'architecture interne du HackRF One n'a pas grand-chose à voir avec celle d'un récepteur RTL-SDL si ce n'est dans les grandes lignes. On voit ici clairement deux des principaux éléments du périphérique, un microcontrôleur NXP et un composant programmable appelé FPGA de chez Xilinx. Ces deux éléments traitent les données issues du convertisseur analogique/numérique (et numérique/analogique) avant transmission en USB. Ces deux éléments peuvent être reprogrammés pour améliorer le fonctionnement du HackRF One (mise à jour) ou accueillir des programmes destinés à un usage spécifique.

Là encore, tout ceci est relativement technique et demande des connaissances plus avancées que la simple utilisation d'applications plus « ludiques ». En poussant davantage dans ce sens, c'est selon moi, à ce moment que les choses deviennent réellement amusantes. Ceci demande du temps, beaucoup d'investissement et de recherches, mais c'est en arrivant finalement à programmer un outil pour le HackRF One qu'on se rend réellement et pleinement compte de son utilité et son intérêt.

Acquérir un HackRF One ou tout autre matériel du même type, n'est pas la simple dépense d'une somme conséquente, mais plutôt un engagement à vous torturer l'esprit et surtout à aimer ça... DB

FAITES PEUR À VOS AMIS EN LISANT LEUR CARTE BANCAIRE !

Denis Bodor

Oui, c'est un titre accrocheur... C'est fait exprès, je n'ai pas pu me retenir. Dans les faits ce qui va suivre n'a absolument rien d'illégal rassurez-vous. Le but est avant tout de faire prendre conscience au plus grand nombre la présence de technologies que vous ne soupçonnez pas nécessairement et qui sont partout autour de vous, y compris dans votre poche, bien cachées dans votre porte-feuille...

Nous transportons partout avec nous une quantité non négligeable de cartes de toutes sortes : carte bancaire, titre de transport, carte de fidélité, carte d'assuré social, etc. Avec nous où que nous allions et pourtant, nous ne savons pas ce qu'elles contiennent réellement. Jetons donc un œil...

Même si j'évoque les termes de « carte bancaire » dans le titre de l'article, le sujet ou plutôt l'outil que nous allons découvrir ici ne se limite pas à cela. Cardpeek, de son petit nom, permet de lire le contenu de *smartcards* respectant le standard ISO/IEC 7816. Ce standard se divise en 4 parties définissant le format physique de la carte jusqu'à la syntaxe des données qui lui permettent de communiquer. Il est amusant d'ailleurs de remarquer que ISO 7816-1 est ce qu'on appelle généralement, dans le langage familier, un « format carte de crédit », 85 x 54 x 0,76 mm (peut-être utile afin d'impressionner un interlocuteur dans une discussion).

Cardpeek est l'œuvre d'Alain Pannetrat et est une application disponible pour GNU/Linux, Windows et macOS. Nous allons ici en faire usage, bien entendu, avec une Raspberry Pi puisque l'outil existe dans Raspbian sous forme de paquet installable d'un simple petit coup de `sudo apt-get install cardpeek` (qui installe également `cardpeek-data`). Cardpeek est capable de lire et interpréter les informations de :

- cartes EMV : typiquement les cartes bancaires, VISA, MasterCard, etc. ;
- passeports biométriques sans contact ;
- cartes d'identité belges eID ;
- titres de transport de type Navigo/Calypso et compatibles (MOBIB, VIVA, RavKav, etc.) ;
- cartes SIM des téléphones mobiles ;
- cartes d'assuré social Vitale 2 ;
- porte-monnaie électroniques Moneo ;
- chronotachygraphes numériques (qui remplacent les « disques » papiers des camionneurs) ;
- cartes OpenPGP permettant le chiffrement et la signature avec PGP/GnuPG ;
- cartes Mifare Classic de NXP (cf. *Hackable n°10*).

1. LE MATÉRIEL NÉCESSAIRE

Cardpeek est en mesure de lire le contenu de cartes à puce et de cartes sans contact, mais ceci demande, bien entendu l'utilisation de périphériques spécifiques. Cardpeek n'accède cependant pas directement aux périphériques ni même aux cartes, mais passe par un intermédiaire, un service, prenant la forme d'un démon PC/SC (pour *Personal Computers with Smart Cards*). Il vous faudra donc installer ce service en plus de Cardpeek sur votre Raspberry Pi, via les paquets `pcscd` et `pcsc-tools`.

Vient alors la problématique du lecteur ou plutôt des lecteurs. Si vous comptez explorer le domaine, il vous faudra deux types de lecteurs : un lecteur de smartcard (carte à puce) et un lecteur sans contact (RFID/NFC). Ce second type a déjà été évoqué dans les pages du numéro 10 et en particulier dans

Voici le matériel utilisé pour cet article, en compagnie d'une Raspberry Pi. Tous ces lecteurs fonctionnent sans le moindre problème avec PC/SC et Cardpeek et vous permettront d'en apprendre beaucoup sur ce que vous avez réellement dans vos poches...

Ce minuscule lecteur de smartcards pliable de SCM est sans le moindre doute mon préféré. Tellement plus pratique qu'un lecteur standard et son câble, c'est un accessoire parfait pour le plus curieux et le plus impulsif des bidouilleurs...

l'article sur l'utilisation d'un lecteur NFC avec une Pi. Le modèle le plus couramment utilisé est l'ACR122U d'ACS que vous trouverez pour moins d'une trentaine d'euros sur eBay par exemple. Ce n'est pas le modèle de meilleure qualité, en particulier pour jouer avec les tags NFC, mais il est directement pris en charge par le service PC/SC contrairement à un modèle plus intéressant, le SCL3711 de SCM (très intéressant pour une utilisation avec la libNFC, mais pas compatible CCID et donc presque inutilisable avec PC/SC et Cardpeek).

En ce qui concerne les cartes à puce, la grande majorité des lecteurs sont compatibles CCID et sont donc reconnus directement. Une simple recherche de « smartcard reader CCID » vous remontera une bonne quantité d'offres de vendeurs eBay, mais tantôt également des sites comme Banggood ou DealeXtreme. Pour les essais faits durant la rédaction de cet article, trois lecteurs ont été utilisés :

- un lecteur SCM SCR335 relativement ancien ;
- un lecteur SCR 355 provenant d'une offre réservée aux abonnés TER afin de recharger leur abonnement en ligne ;
- et un adorable lecteur SCM SCR3500 pliable que je qualifierai très scientifiquement de « parfaitement trop choupy mignon » :)

Un point très important doit être pris en considération concernant le lecteur RFID/NFC ACS ACR122U : sa prise en charge par le noyau Linux sous la forme de deux modules chargés automatiquement (**pn533** et **nfc**). Le support du noyau est un héritage direct du lien de parenté qui existe entre GNU/Linux et Android. Le noyau Linux est en effet au cœur d'Android, un système fonctionnant majoritairement sur smartphones et tablettes, du matériel généralement équipé de supports et de contrôleurs NFC. Ici cependant, cette prise en charge directe interfère avec le fonctionnement de PC/SC.

Ainsi à la connexion du lecteur ACR122U sur votre Pi, le noyau déclenche la prise en charge du matériel (sortie de la commande **dmesg**) :

```
usb 1-1.5: New USB device found, idVendor=072f, idProduct=2200
usb 1-1.5: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1.5: Product: ACR122U PICC Interface
usb 1-1.5: Manufacturer: ACS
nfc: nfc_init: NFC Core ver 0.1
NET: Registered protocol family 39
pn533 1-1.5:1.0: NFC: NXP PN532 firmware ver 1.6 now attached
usbcore: registered new interface driver pn533
```

De de fait, lors du test du lecteur avec la commande **pcsc_scan**, le service PC/SC attend qu'un lecteur soit disponible indéfiniment :

```
$ pcsc_scan
PC/SC device scanner
V 1.4.23 (c) 2001-2011, Ludovic Rousseau <ludovic.rousseau@free.fr>
Compiled with PC/SC lite version: 1.8.11
Using reader plug'n play mechanism
Scanning present readers...
Waiting for the first reader...
```

Le problème est, bien entendu, que cette disponibilité n'arrive jamais puisque le noyau « accapare » le périphérique qui n'est plus utilisable par ailleurs. Il faut donc explicitement dire au noyau d'ignorer ce matériel. Ceci se fait en ajoutant un fichier **/etc/modprobe.d/blacklist-libnfc.conf** dans le système, contenant les noms des modules qui ne doivent jamais être chargés :

```
blacklist pn533
blacklist nfc
```

Après un petit redémarrage de la Pi (ou un déchargement manuel des modules avec **sudo rmmod**), le branchement provoque, au niveau noyau, une simple notification de connexion d'un périphérique USB :

```
usb 1-1.4: new full-speed USB device number 4 using dwc_otg
usb 1-1.4: New USB device found, idVendor=072f, idProduct=2200
usb 1-1.4: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1.4: Product: ACR122U PICC Interface
usb 1-1.4: Manufacturer: ACS
```

Côté test avec **pcsc_scan**, on peut alors voir que le lecteur est maintenant bien pris en charge et en attente d'une carte ou d'un tag à lire :

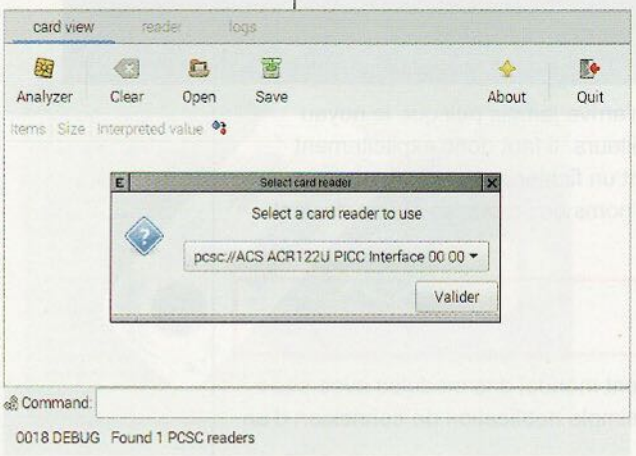
```
$ pcsc_scan
PC/SC device scanner
V 1.4.23 (c) 2001-2011, Ludovic Rousseau <ludovic.rousseau@free.fr>
Compiled with PC/SC lite version: 1.8.11
Using reader plug'n play mechanism
Scanning present readers...
0: ACS ACR122U PICC Interface 00 00

Fri Dec  2 11:03:04 2016
Reader 0: ACS ACR122U PICC Interface 00 00
  Card state: Card removed,
```

2. UTILISONS CARDPEEK

Cardpeek est une application graphique, elle devra donc être lancée depuis l'environnement graphique PIXEL (qui est en réalité le gestionnaire de fenêtres OpenBox et l'environnement LXDE avec un thème Raspberry Pi, il faut rendre à César ce qui est à César) ou en déportant l'affichage graphique sur une autre machine (ça tombe bien, il y a un article sur le sujet dans ce numéro).

Le fonctionnement de Cardpeek est très intéressant et très bien pensé. Il est en réalité, en lui-même, une interface graphique chargée de dialoguer avec PC/SC pour obtenir des informations des cartes ou tags, mais le travail d'analyse et de traitement de ces informations est fait par un ensemble de scripts écrits en langage Lua. Un jeu de scripts est livré de base avec Cardpeek, mais cette architecture modulaire vous permet, avec un peu d'efforts et de connaissances de base, d'ajouter aisément vos propres scripts ou ceux d'autres utilisateurs, pour d'autres cartes et tags.

Dès le lancement, Cardpeek vous demande de sélectionner le lecteur à utiliser. Ceux-ci sont mis à disposition par le service PC/SC, si votre lecteur n'apparaît pas dans la liste, assurez-vous que celui-ci n'a pas été pris en charge directement par le noyau Linux, le rendant alors inaccessible par PC/SC.



Au démarrage de l'application, vous serez invité à choisir le lecteur à utiliser. Dans bien des cas, il n'y en aura qu'un, directement mis à disposition par le service PC/SC (sauf si vous avez connecté le lecteur ACR122U et un lecteur de cartes à puce en même temps).

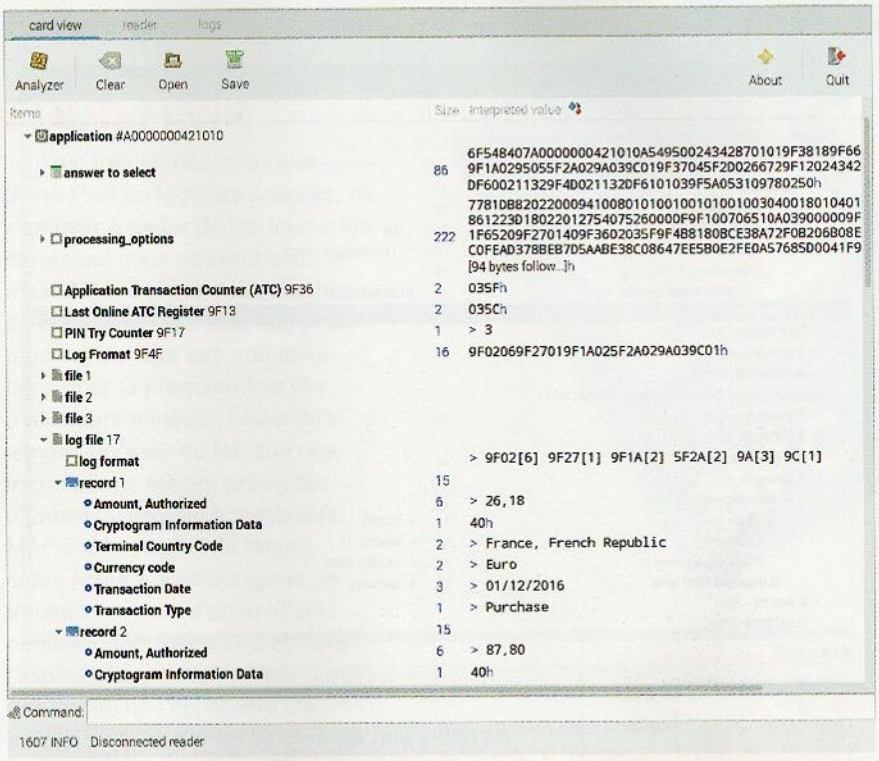
Le bouton ***Analyzer*** en haut à gauche affichera un menu des diverses options d'analyse de cartes disponibles (fournis par les scripts Lua par défaut). « atr » est un script très basique permettant simplement d'afficher l'ATR (*Answer To Reset*) d'une carte ou d'un tag. C'est le message automatiquement envoyé

lorsqu'il ou elle est réinitialisé(e) et qui fournit des informations sur le type de carte ou de tag. Cardpeek peut, avec ces informations, tenter de déduire le modèle, l'identité et l'utilité d'une carte.

Mais le plus intéressant est sans le moindre doute les autres scripts disponibles et en particulier l'entrée ***EMV*** du menu. Celle-ci vous permettra de collecter et d'afficher des données relatives à une carte bancaire, que ce soit avec un lecteur de carte à puce ou l'ACR122U et une carte disposant de la fonctionnalité de paiement sans contact.

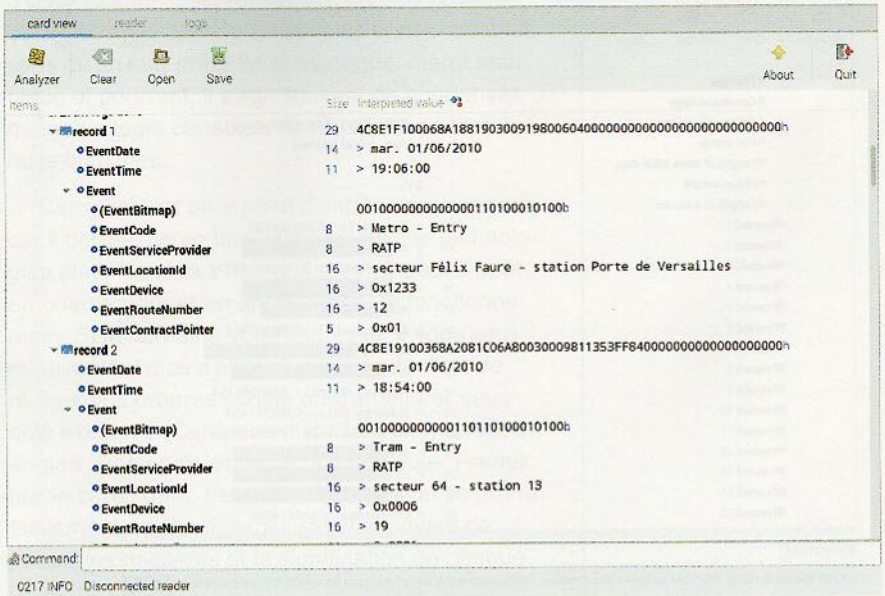
Les informations affichées dépendent totalement du type de carte, de son modèle et des fonctionnalités intégrées. Il est intéressant de noter que d'anciennes cartes (peut-être maintenant arrivées à expiration) étaient vraiment très « bavardes » en fournissant, par exemple le nom du porteur. Une véritable « fuite » d'informations personnelles, en particulier dans le cas d'une lecture sans contact. Mais actuellement, ces informations sont beaucoup plus limitées, mais toujours assez impressionnantes.

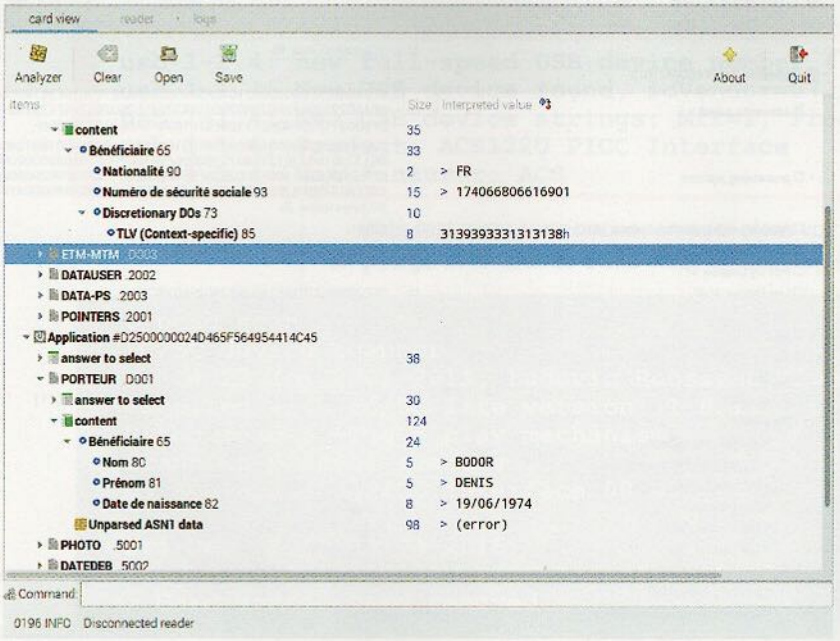
Ma carte Visa Gold, par exemple, fournit sans authentification particulière, les dernières transactions effectuées avec le montant et la date sur une plage d'environ deux mois. D'autres modèles comme ma Visa « standard » (bleue) d'une autre banque ne fournit pas ces informations, pas plus que la MasterCard d'une de mes collègues.



Le lecteur des informations d'une carte bancaire supportant le paiement sans contact conduit parfois à des résultats très surprenants. Ici, ma carte Visa Gold me liste mes dernières transactions avec, comme ici, un paiement/achat de 26,18€ le 1er décembre...

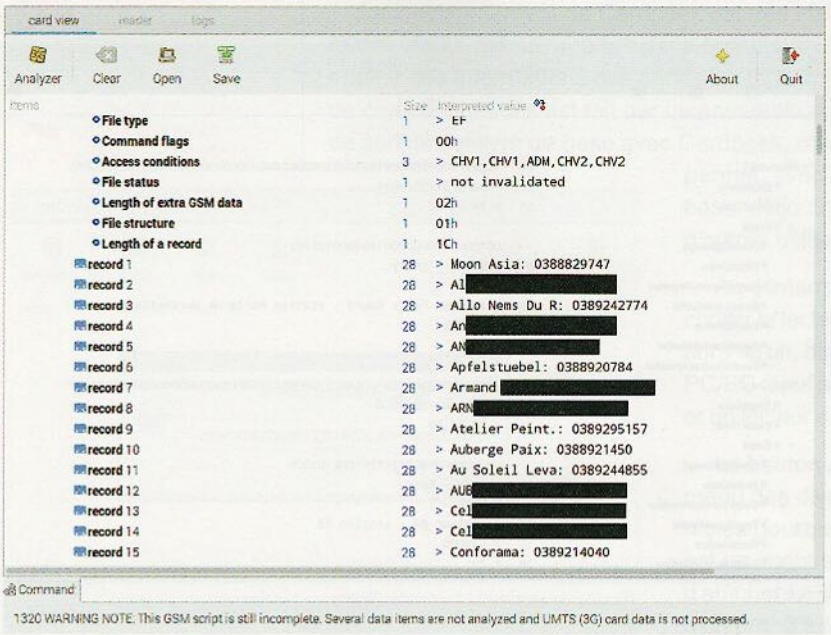
La carte Navigo utilisée pour les tests contient en plus des informations personnelles sur le porteur, un journal complet d'événements associés avec date, heure et lieux d'utilisation. Effrayant !





Une carte Vitale 2 contient bien plus d'informations que le script associé ne permet de traduire. On voit ici simplement le nom du porteur et son numéro d'assuré. Le reste est relativement inintelligible pour le commun des mortels.

Cardpeek est également capable de lire le contenu de cartes SIM de téléphones mobiles. Ceci, à condition d'entrer le code PIN correct, vous permettra entre autres choses de récupérer le répertoire de contacts qui s'y trouve (et pouvoir réserver une table dans votre restaurant préféré, ouf, sauvé !).



Notez que l'accès à ce type de données ne présente aucun risque pour la carte elle-même et en particulier pour le compteur de code PIN. En effet, aucune authentification n'est nécessaire pour les obtenir, en d'autres termes le code PIN n'est pas demandé. CardPeek et le script EMV se limitent à la consultation d'informations non sécurisées, non protégées (mais qui peut-être devraient l'être).

D'autres scripts et d'autres cartes peuvent également être utilisés, via une lecture avec contact. C'est le cas, par exemple, de la carte d'assuré social Vitale 2 présentant différentes données, parmi lesquelles nom, prénom et date de naissance de l'assuré ainsi que son NIRPP ou NIR (le bon vieux « numéro de sécu »). D'autres informations sont présentes dans la carte, mais le script associé ne les décode pas et se contente d'afficher les données brutes.

Il vous est également possible de lire des titres de transport Calypso et compatibles comme les cartes Navigo. N'ayant pas à disposition une carte récente de ce genre, je ne saurai être totalement affirmatif quant aux données contenues et dispensées actuellement. La carte usager en ma possession a plus de 6 ans et ne dispose pas de technologie sans contact, mais elle fournit joyeusement un lot d'informations personnelles ainsi qu'un journal complet d'événements avec dates, heures et lieux d'utilisation dans le métro parisien. J'espère sincèrement que ce genre d'informations n'est pas accessible en mode sans contact

avec les nouvelles cartes, voire ne sont plus accessibles du tout sans authentification...

La technologie Calypso n'est pas utilisée que par la RATP, mais également par la SNCF ainsi que par des infrastructures de transport en commun de plusieurs grandes villes. Ici en Alsace par exemple, les abonnements de travail TER SNCF (carte Alséo) utilisent cette technologie, mais le script dédié ne semble pas en mesure d'interpréter les données correctement, car elles sont organisées de façon différente d'un passe Navigo. Quoi qu'il en soit, il est intéressant de noter que les abonnés TER peuvent « recharger » leur abonnement mensuel directement en ligne, via un site dédié (service Dom'TER) et un périphérique USB, vendu à l'abonné pour 5€. Il s'avère en réalité que le périphérique en question est un lecteur SCM Microsystems SCR 355, parfaitement utilisable avec PC/SC et Cardpeek...

Enfin, un autre script intéressant intégré à Cardpeek est celui permettant de lire les cartes SIM de mobiles « GSM ». Attention cependant, le code PIN de la carte est demandé, il y a donc un risque de blocage en cas de nombre de tentatives infructueuses trop important. L'accès à une carte SIM vous permettra, par exemple, de lister les contacts qu'elle contient. Chose qui peut être relativement pratique en cas de problème, si votre smartphone est endommagé et que vous ne disposez pas d'autres solutions pour récupérer votre répertoire.

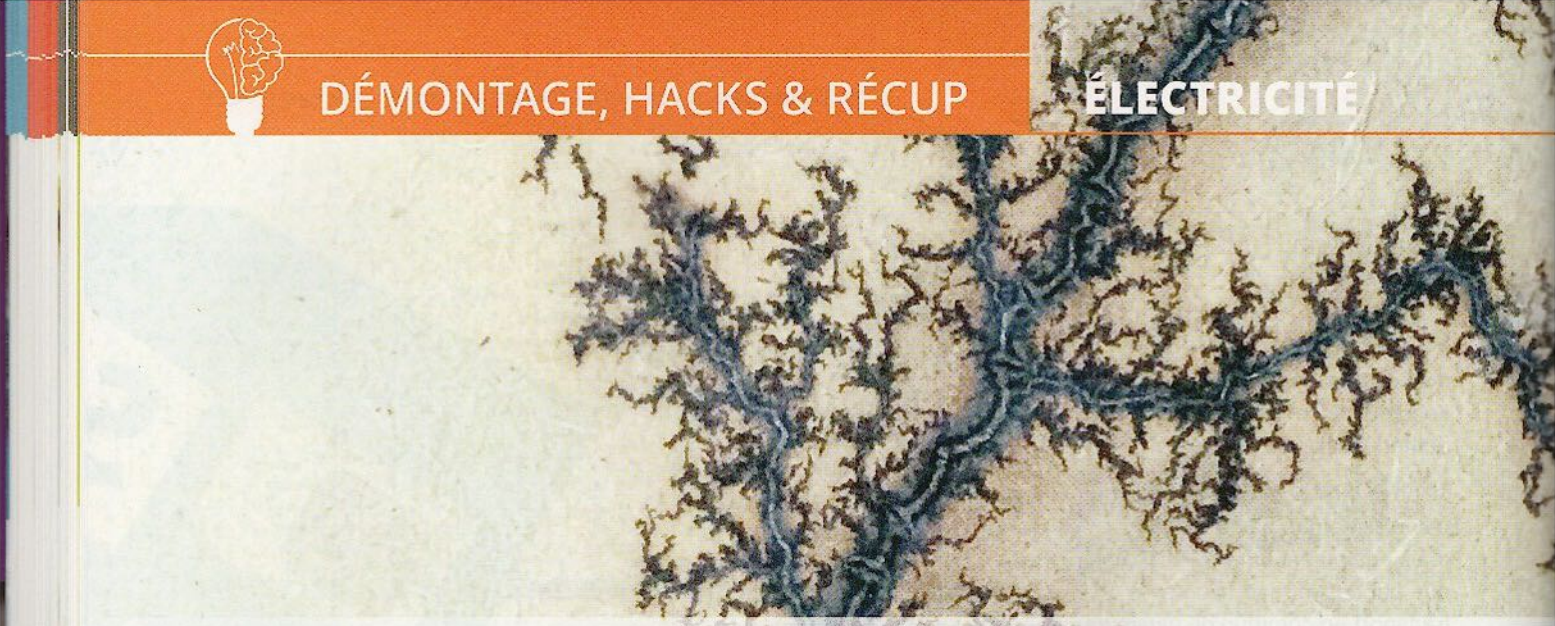
POUR FINIR

Il est très amusant, à l'aide d'une Pi et de lecteurs adaptés, de s'amuser à tenter de lire toutes les cartes qui nous passent sous la main. Mais il y a là quelque chose de bien plus important. Jusqu'au moment de faire ces manipulations pour la première fois (il y a plusieurs années), j'étais totalement ignorant du fait que ces informations étaient présentes et aussi facilement accessibles. Même si la plupart du temps, ces « fuites » sont bénignes, je trouve important d'en avoir pleinement conscience et d'en faire l'expérience par soi-même. C'est une chose de l'apprendre via un article, un billet sur un blog ou via une vidéo en ligne, c'en est une autre de le voir, le constater de son propre chef.

Le monde des smartcards est captivant et ceci s'applique au sens large du terme. J'inclurai ici, maladroitement et de façon erronée, dans cette désignation tout autant les cartes à puces, les tags RFID/NFC, mais également les cartes et supports à piste magnétique. Toutes ces choses passent dans notre vie comme de simples consommables sans qu'on leur prête un quelconque intérêt technique et pourtant, il s'agit souvent de condensés de technologie capables de vous faire passer des nuits blanches.

Cardpeek est un « point d'entrée » intéressant, car il permet de se familiariser avec ces technologies (sauf les pistes magnétiques) et petit à petit en comprendre et en appréhender le fonctionnement. Si le domaine vous intéresse, il vous sera relativement aisé d'aller plus loin en tentant de rédiger vos propres scripts ou d'améliorer ceux déjà existants (Cardpeek dispose d'un manuel, en anglais, très complet de ce point de vue). Prenez garde cependant, c'est un jeu auquel on se prend facilement, fouillant, explorant et essayant de percer les mystères et la signification de chaque donnée... DB

Ne jetez pas le support sur lequel est livrée votre carte SIM. Non seulement vous y trouvez des informations qui peuvent être très importantes pour débloquer la carte en cas de problème, mais en plus cela fera office de support pour lire une carte SIM avec un lecteur standard : un petit morceau d'adhésif et le tour est joué !



GRAVEZ LE BOIS AVEC DE L'ÉLECTRICITÉ !

Denis Bodor

L'électricité est quelque chose de fantastique ! Elle éclaire et chauffe nos lieux de vie et de travail, nous permet de communiquer, remplace avantageusement d'autres formes d'énergies, prolonge et sauve des vies, anime nos bien aimés ordinateurs et smartphones...
La fée électricité est partout, louée de tous pour son utilité, mais elle est également artiste à ses heures en dessinant pour le plaisir de nos yeux de magnifiques et intrigantes figures...

ATTENTION !

Les explications qui vont suivre impliquent l'utilisation de tensions très importantes et leur mise en œuvre présente un risque d'électrocution ou d'électrisation. Cet article détaille la mise en œuvre de techniques et de pratiques qui sont présentées dans un but purement pédagogique. Ni l'éditeur, ni le magazine, ni moi-même ne saurions être tenus responsables en cas d'accident si vous tentez de reproduire ces manipulations.

Disons-le clairement : ces manipulations sont très dangereuses et comportent des risques. C'est à vous d'agir en personne responsable, de respecter les principes de sécurité élémentaires et de prendre toutes les mesures de protection nécessaires pour minimiser les risques d'accident. Utiliser des hautes tensions ne doit pas être fait à la légère, vous DEVEZ être conscient des risques et agir en conséquence.

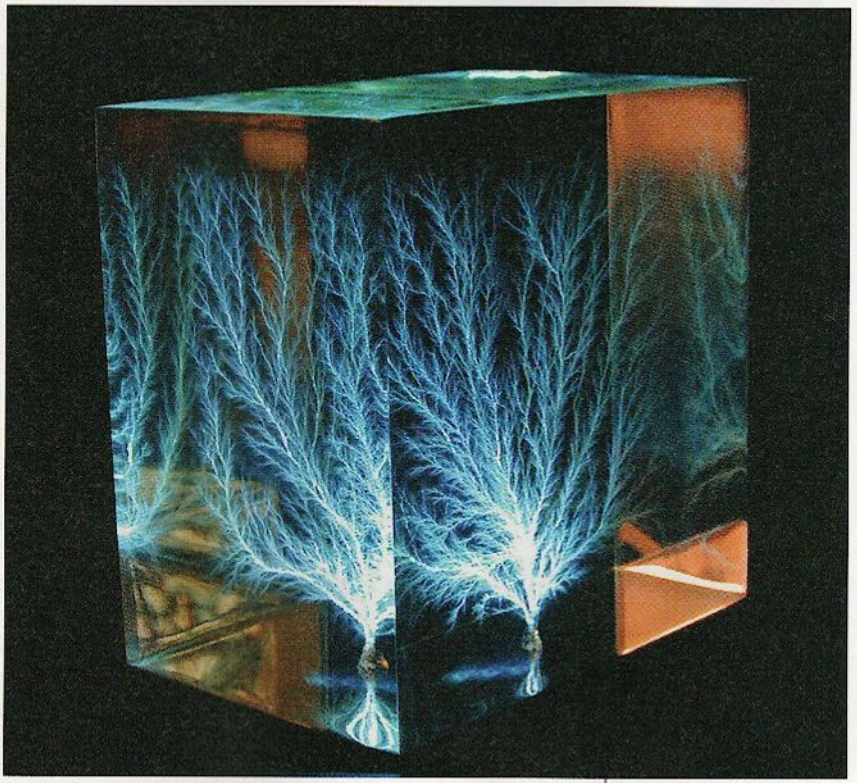
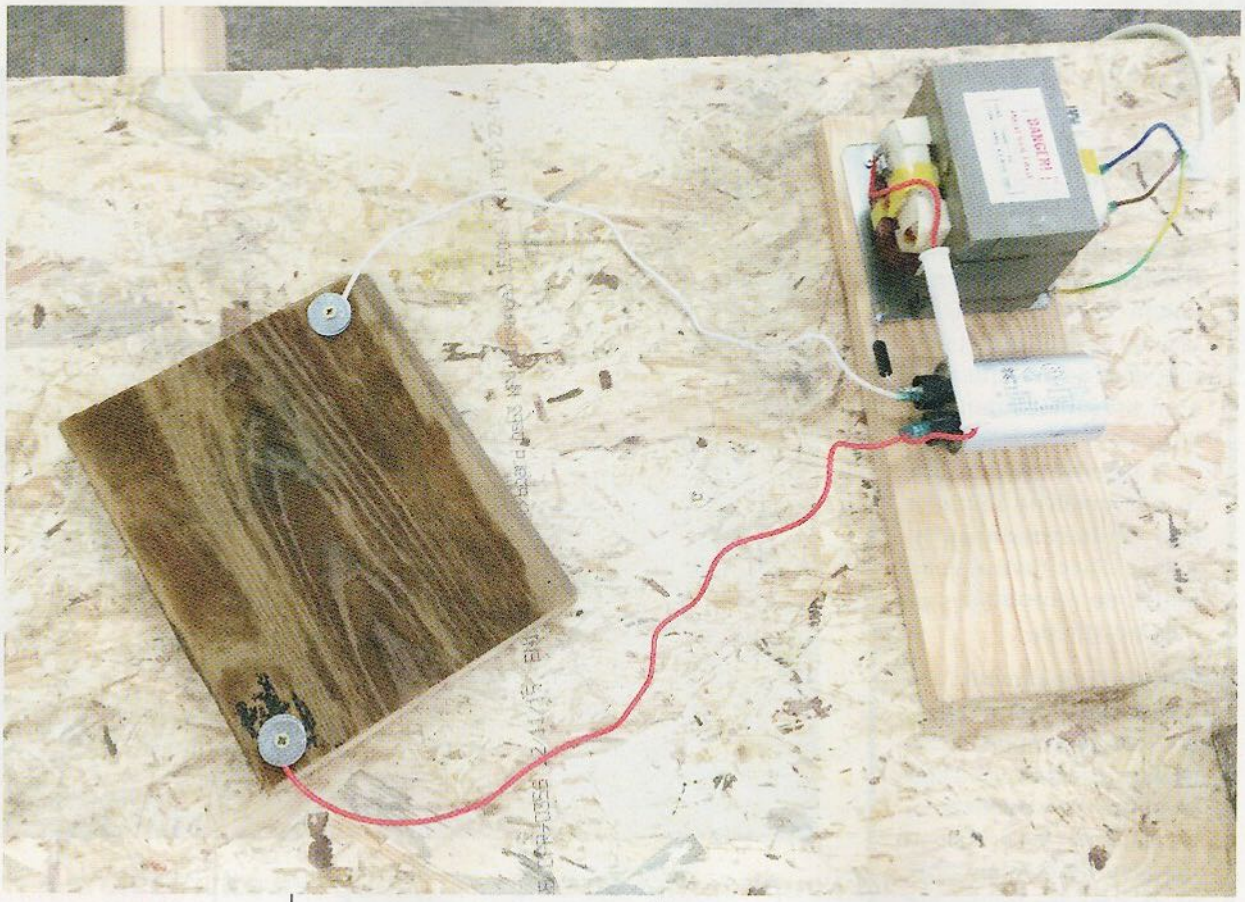


Figure de Lichtenberg capturée dans un bloc d'acrylique de 8 cm de côté, ici illuminé par une led bleue (Source Wikipédia, photographie par Bert Hickman, www.teslamania.com).

L'un des plus impressionnants phénomènes électriques existant dans la nature est sans le moindre doute la foudre. Magnifique, impressionnante, terrifiante parfois pour certains, capricieuse, indomptable...

Indomptable ? Pas si sûr ! Certes, il n'est pas question de jouer avec des millions de volts, bien que cela reste possible à simuler (mais pas par nous), mais de simplement tenter de retrouver, de capturer et de conserver l'esprit et l'aspect de ce phénomène, au point d'en faire une forme d'art.

Le physicien allemand Georg Christoph Lichtenberg a eu la même idée en 1777 et a donné son nom à ces « gravures » qu'on appelle à présent des figures de Lichtenberg. Son idée consistait à utiliser un générateur capacitif afin de construire et accumuler une charge électrostatique importante et la décharger rapidement à la surface d'un matériau isolant afin de conserver, tracés dans de la poussière, le chemin et le dessin de cette décharge. Inventant un principe qui est encore utilisé



Voici le dispositif expérimental utilisé : un transformateur de four micro-onde, équipé en série sur le secondaire d'un fusible et d'un condensateur, avec une diode en parallèle. L'ensemble de ces éléments provient du four.

aujourd'hui dans les imprimantes laser (électrophotographie), Lichtenberg laissa ainsi et grâce à ses recherches (et bien d'autres choses), son nom dans l'histoire.

Aujourd'hui encore, cette même technique est utilisée pour produire ces figures dans des blocs d'acrylique (polyméthacrylate de méthyle ou PMMA). Ceux-ci sont chargés par exposition à un faisceau d'électrons qui, en raison de leur énergie et leur vitesse, pénètrent dans l'isolant, décélèrent et y restent « coincés ». La charge accumulée atteint une telle densité et une telle tension dans le support qu'on flirte alors avec le claquage (tension de claquage). On frappe alors le support avec une pointe, pénétrant légèrement le support et le claquage se produit, déchargeant en une fraction de seconde le matériau. Cette décharge crée des chemins de conduction dans le support, formant des arcs électriques à haute température qui fondent l'acrylique et le fracture. La forme de la décharge, similaire à celle de la foudre (qui est également une décharge électrostatique disruptive), reste gravée dans la masse, créant de captivantes et uniques sculptures.

À ce stade, vous vous dites « *Je n'ai pas d'accélérateur linéaire sous la main, mais Denis va m'expliquer comment en faire un avec un trombone, du papier de chewing-gum et une vieille lampe de poche* ». Désolé, mais non, je ne suis pas MacGyver (incompatibilité capillaire), mais ce que je peux vous proposer en revanche est quelque chose d'un peu plus accessible, pour obtenir les mêmes figures, plus calmement, gravées sur du bois.

1. À LA RECHERCHE DE VOLTS... DE PLEIN PLEIN DE VOLTS !

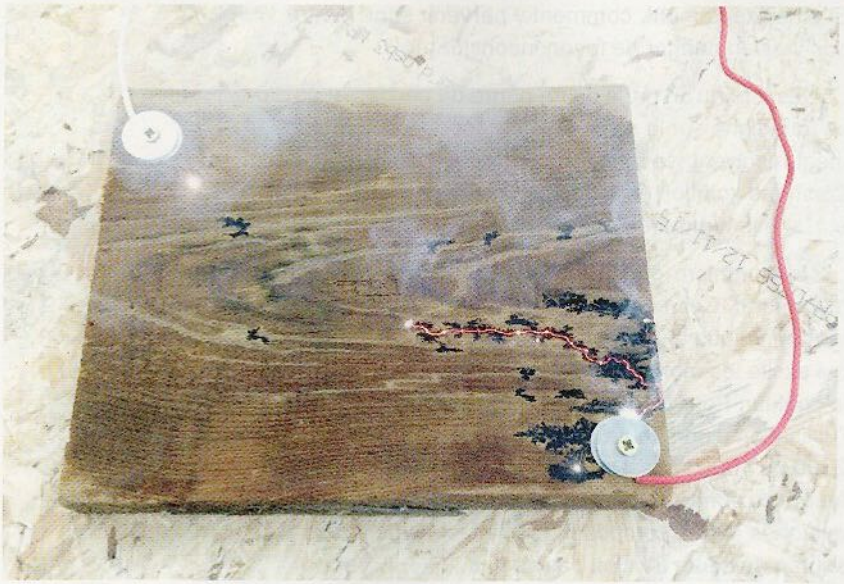
Vous l'avez compris, pour obtenir un quelconque résultat, nous avons besoin d'une tension suffisante. Certes, le fait d'utiliser un support comme le bois et de ne pas utiliser de décharges électrostatiques nous dispense d'avoir recours à des centaines de milliers de volts, mais le principe que nous allons mettre en œuvre nécessite tout de même plusieurs milliers de volts.

La question est donc : comment obtenir une telle tension ? Et plus exactement, comment y parvenir sans mettre nos vies en danger de façon inconsidérée ?

Dès lors qu'on parle de ce genre de choses et qu'on se renseigne sur le sujet, un élément apparaît rapidement récurrent : le transformateur de four micro-onde. C'est une solution économique, facilement accessible et... souvent une très mauvaise idée.

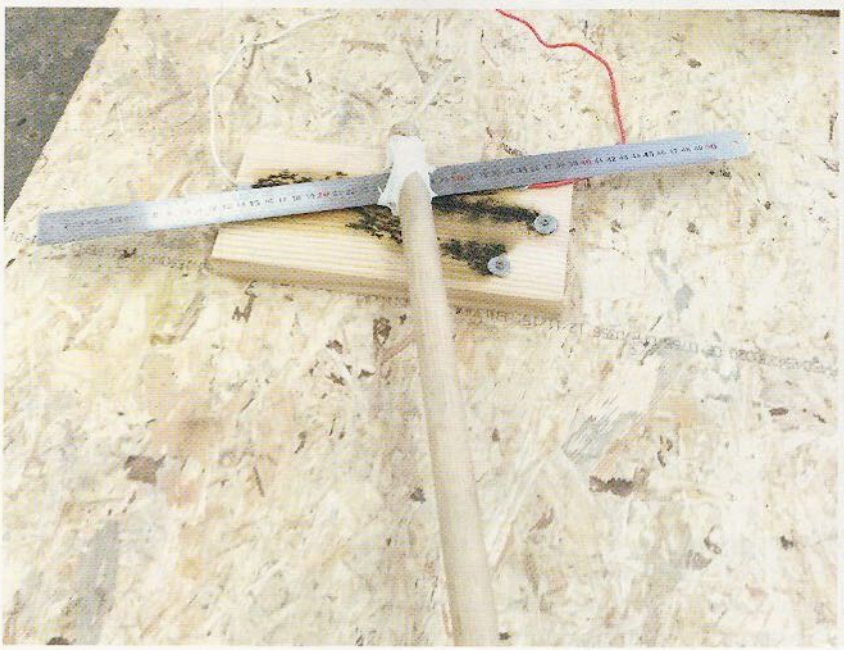
Un four micro-onde repose sur un élément appelé magnétron. Celui-ci permet de générer une onde électromagnétique à une fréquence plus ou moins précise qui excite les molécules d'eau, ce qui génère alors de la chaleur. Pour fonctionner, un magnétron nécessite l'utilisation d'une haute tension de l'ordre de quelques 2000 ou 4000 volts. De ce fait, en démontant un vieux four micro-onde et en récupérant le transformateur et quelques autres composants, on dispose alors d'une source de haute tension parfaitement utilisable. Mais... et ce « mais » est capital, un four micro-onde nécessite également beaucoup de courant qui n'est limité que par les caractéristiques du transformateur lui-même.

Le bois n'est pas suffisamment conducteur pour permettre le passage du courant. Il est donc humidifié avec une solution de carbonate de soude ou de sel, dans l'eau.



Voici la toute première tentative de réalisation d'une figure de Lichtenberg. On voit clairement le tracé se former petit à petit en plusieurs endroits et le parcours du courant chauffant le tracé déjà formé.

Lors de la mise hors tension, et après déconnexion, il est plus que probable que le condensateur soit encore pleinement chargé. Il convient alors de relier les deux électrodes pour le décharger. Prudence étant mère de sûreté, une règle métallique au bout d'un long manche à balai en bois est la solution la plus prudente que j'ai imaginé... eh oui, avec les gants en prime !



Ces transformateurs utilisent le courant domestique (230V) et sont composés de deux bobines : une primaire reliée à l'alimentation domestique et une secondaire fournissant quelques 2000 volts avec un courant de l'ordre de 0,5 ampères en fonctionnement standard. Cette seconde bobine est d'une part reliée au châssis du transformateur et d'autre part au magnétron, ainsi qu'à un condensateur et une diode de puissance.

Le problème ici n'est pas tant la tension, mais le courant. Si vous utilisez ce type de solution pour obtenir une haute tension, en cas de contact avec la sortie du transformateur, c'est un demi-ampère qui traversera massivement votre corps. Là, nous ne parlons pas d'une « châtaigne » vous laissant le bras engourdi quelques minutes, mais d'un risque réel et important de décès par électrocution !

Pire encore, comme le transformateur forme une isolation galvanique entre le courant domestique et votre montage, les systèmes de protection qu'il s'agisse d'un fusible, d'un disjoncteur ou d'un interrupteur différentiel, **ne fonctionneront pas**. En effet, du point de vue de l'installation électrique, la bobine primaire du transformateur ne présentera pas de fuite à la terre et ne dépassera probablement pas la valeur maximum de courant admise, le courant continuera donc de circuler dans votre corps sans interruption. Sachant qu'avec ce genre de tensions, les contractions musculaires peuvent parfaitement vous « coller » littéralement au

La formation de la figure s'accompagne généralement d'un dégagement important de fumée. On voit ici sur la gauche un arc électrique établissant un pont entre l'électrode et une zone davantage conductrice. Ceci provoque une brûlure et parfois l'apparition d'une flamme. Le risque d'incendie est un autre point important à prendre en compte dans ce genre d'expérience.

conducteur, on parle ici de mort quasi certaine si les conditions jouent en votre défaveur.

La tension en œuvre, l'absence de protection, et les risques liés à la technique que je vais décrire rendent l'utilisation d'un transformateur de micro-onde infiniment plus dangereux qu'un choc électrique lié au « courant secteur ». Vous ne démontez pas une prise de courant sous tension, car cela présente un risque, alors ne jouez pas inconsciemment avec quelque chose qui peut, et va, vous tuer au moindre faux pas.

Une autre option que j'ai également vue mise en œuvre pour ce genre de chose est le transformateur *flyback* généralement présent dans les écrans à tube cathodique. Là, le problème est sensiblement différent, car le courant en sortie, de l'ordre de 20mA est bien moindre, mais la tension beaucoup plus importante, pouvant aller jusqu'à 35000 V (crête à crête). Avec ce genre de tension très importante, la formation d'un arc électrique devient un énorme problème. La rigidité diélectrique ou tension maximum qu'un milieu peut supporter avant la formation d'un arc électrique, de l'air est de l'ordre de 3000V par millimètre. Je vous laisse faire le calcul, mais vous l'avez compris, ici il n'est pas même nécessaire d'entrer en contact avec un conducteur pour se mettre en danger.

Ce qui nous amène donc à la solution que j'avais initialement choisi d'utiliser : le transformateur pour enseignes néon (ou NST, de l'anglais *neon-sign transformer*). Celui-ci est un équipement destiné à alimenter des tubes originellement remplis avec du gaz néon pour des enseignes publicitaires. Notez que ce qu'on appelle souvent par erreur des « tubes néon » devrait être normalement désigné par « tube fluorescent ». Nous ne parlons pas ici de ballast pour ce type de tubes, mais d'un tout autre type de luminaires.

Ces transformateurs présentent à mon sens plusieurs avantages. Ils fournissent une tension importante, entre 2000 et 15000 volts, mais avec un courant limité à quelques 15 à 30 mA. Le danger est toujours là, bien sûr, mais dans une tout autre échelle qu'avec un composant démonté d'un four micro-onde. Ils sont également prévus et fabriqués pour être utilisés comme des blocs d'alimentation. De ce fait, ils sont protégés, encastrés dans un boîtier, isolés, souvent scellés et résistants aux manipulations, exactement comme l'alimentation de votre ordinateur portable. Enfin, il s'agit d'objets conçus et manufacturés en fonction des caractéristiques qu'ils possèdent et donc

Voici ma première figure que je jugerai comme acceptable sur un morceau de contreplaqué. On peut voir clairement la forme caractéristique rappelant celle de la foudre ou des décharges d'une bobine Tesla.

équipés de connecteurs et de câbles adaptés aux tensions fournies (le plus souvent très souples et gainés de silicone).

Ce type de transformateurs se trouve généralement sur les sites d'enchères en ligne ou dans les boutiques web spécialisées dans les enseignes lumineuses et vous coûtera quelques 50€ pièce (en cherchant bien). J'ai acquis deux exemplaires de ce type de transformateur sur eBay auprès d'un vendeur anglais (*tinytimjukeboxes*). Il s'agit de modèles SP226V fabriqués par Evertron, fournissant 3200V/24mA en 30KHz. Compacts, ils sont activables via un interrupteur connecté par une ficelle (isolant) et disposent d'un potentiomètre permettant d'ajuster la luminosité du tube où ils sont normalement connectés. Mais... il ne fonctionnent pas pour ce type d'application ! En effet, il s'agit de « transformateurs » à découpage s'ajustant automatiquement à la puissance nécessaire à l'alimentation d'un tube et cette fonctionnalité interdit leur utilisation ici (le transformateur ne détecte tout simplement pas de tube, et pour cause).

Je fais mention ici de cette mésaventure pour que vous ne fassiez pas la même erreur. Ces alimentations pour néon sont de plus en plus courantes, car plus polyvalentes que le transformateur ferromagnétique classique et adaptables à plusieurs modèles d'enseignes. Pour produire des figures de Lichtenberg, il vous faut un transformateur classique et non à découpage (ou électronique), plus difficile à trouver et surtout plus lourd (frais de port en conséquence).

J'ai donc été obligé de me rabattre sur la solution la plus dangereuse, littéralement terrifié à l'idée de faire une erreur. Chose non seulement légitime, mais également salvatrice. Le principe est exactement le même que pour

d'autres appareillages comme les outils électroportatifs ou les machines d'atelier : c'est lorsque vous n'en avez plus peur que vous allez faire une bêtise et qu'un accident va arriver.

2. DES FIGURES DE LICHTENBERG SUR DU BOIS

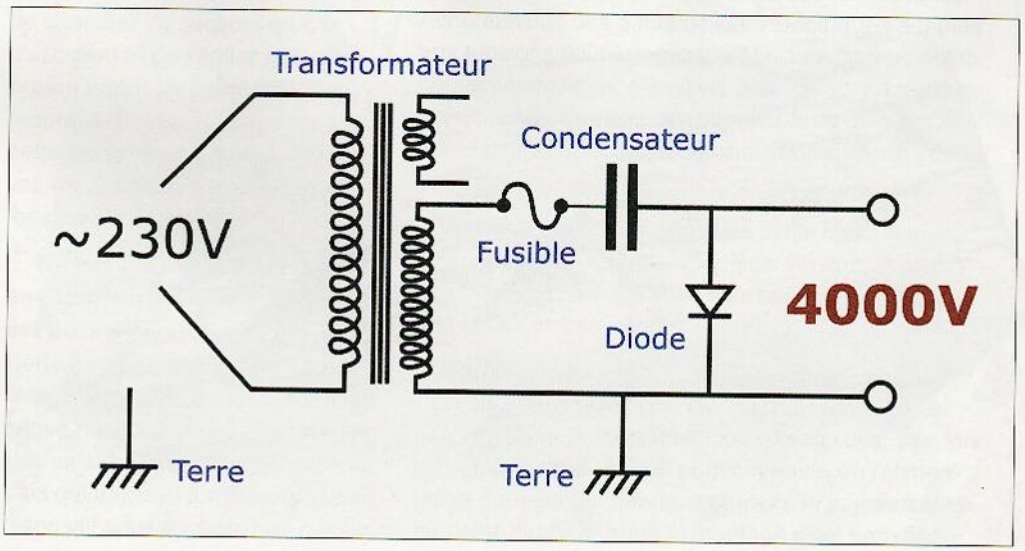
Le principe en œuvre ici est fort simple, même s'il n'est pas aussi impressionnant qu'une décharge électrostatique faisant apparaître subitement la figure souhaitée. Il consiste à utiliser un morceau de bois, qui n'est pas suffisamment conducteur par lui-même et de l'humidifier avec une solution de carbonate de soude. On dispose ensuite les deux électrodes à chaque bout du support et on met en marche le transformateur.

Le courant qui circule alors fait chauffer les électrodes qui brûlent le bois. Cette carbonisation facilite le passage du courant, le carbone étant plus conducteur que la solution, il offre moins de résistance au passage du courant, et la brûlure se propage lentement en direction de l'autre électrode. Les petites imperfections de la fibre du bois, la non-uniformité de l'application de la solution, impactent le parcours du courant et guident de manière chaotique le tracé du dessin.

Il en résulte des figures fractales où la moindre brûlure ouvre un chemin pour le courant et propage l'effet. Le tout guidé, dans les grandes lignes, par la position des électrodes.

Pour initier le passage du courant, il faut un élément partiellement conducteur. L'eau pure n'est pas conductrice, c'est pourquoi on utilise généralement un additif. Le sel est généralement la solution la plus simple et économique, mais donne des résultats moyens tout en ayant un impact sur le bois et sa couleur. On peut alors préférer le bicarbonate de soude, tout aussi facile à trouver en supermarché, ou mieux encore, le carbonate de soude.

Une solution de carbonate de soude est plus conductrice que celle au bicarbonate. On obtiendra très facilement l'un à partir de l'autre, en chauffant le bicarbonate à une température de 100°C. Ceci peut être fait dans un four classique (pas micro-onde). Il vous suffira de placer le bicarbonate de soude dans un plat, enfourné pour une bonne heure et celui-ci se transformera en carbonate. Notez cependant que le carbonate de soude, plus précisément, le carbonate de sodium est irritant sur la peau,



Le circuit utilisé est presque identique à celui du four micro-onde. Seule une bobine secondaire n'est pas utilisée ici, celle fournissant une basse tension avec un courant important, reliée au filament du magnétron. Notez que le secondaire utilisé voit l'une de ses connexions reliées au châssis du transformateur et à la terre.

Un vieux morceau de bois, poncé avec plus ou moins de rigueur peut trouver une seconde jeunesse une fois marqué par ce procédé. Mes différents essais avec le présent sujet m'ont permis d'apprendre que le ponçage jusqu'à obtenir une surface bien lisse est généralement indispensable pour un résultat acceptable.

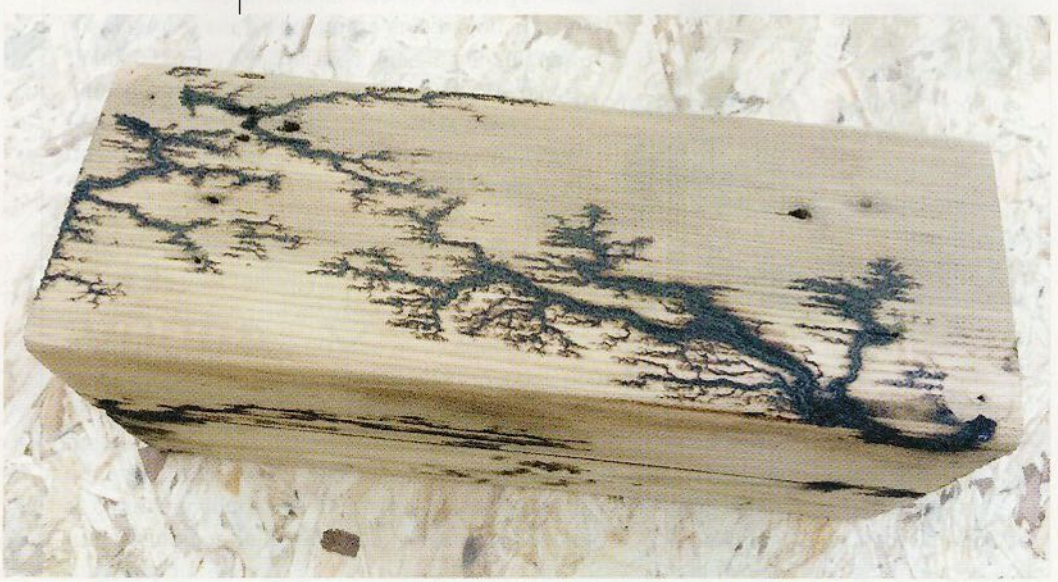
contrairement au bicarbonate de sodium. Mieux vaut donc utiliser un récipient réservé pour cet usage et éventuellement mettre des gants. Je rappelle, à toutes fins utiles, que le carbonate de soude ou le bicarbonate de soude n'est pas de la soude caustique (hydroxyde de sodium). Ne jouez pas avec la soude pour ce genre d'application !

Une autre recommandation importante : brûler du bois dégage de la fumée, beaucoup de fumée. Il est donc de bon ton de faire ce genre de choses à l'extérieur ou dans un local bien ventilé, et d'éventuellement prévoir un système d'extraction, même improvisé (un aspirateur d'atelier fait merveille pour ce genre de choses).

La procédure en elle-même est donc relativement simple : humidifier le bois avec la solution conductrice, positionner les électrodes et activer l'alimentation, pour voir la figure se dessiner plus ou moins lentement.

3. CONSEILS DE SÉCURITÉ

Vous l'avez compris, techniquement, réaliser ces figures n'est pas compliqué. Tout cela consiste à le faire de la manière la plus sûre possible. Dans un premier temps,



Voici la face la plus réussie de cette pièce, les autres ne méritant malheureusement pas d'être prises en photo. La technique doit encore être améliorée jusqu'à arriver à une bonne probabilité de réussite. On voit clairement ici l'impact des fibres du bois sur le tracé de la figure, qui semble découpée en tranches horizontales... Ça m'apprendra à faire preuve de laxisme dans mon ponçage !

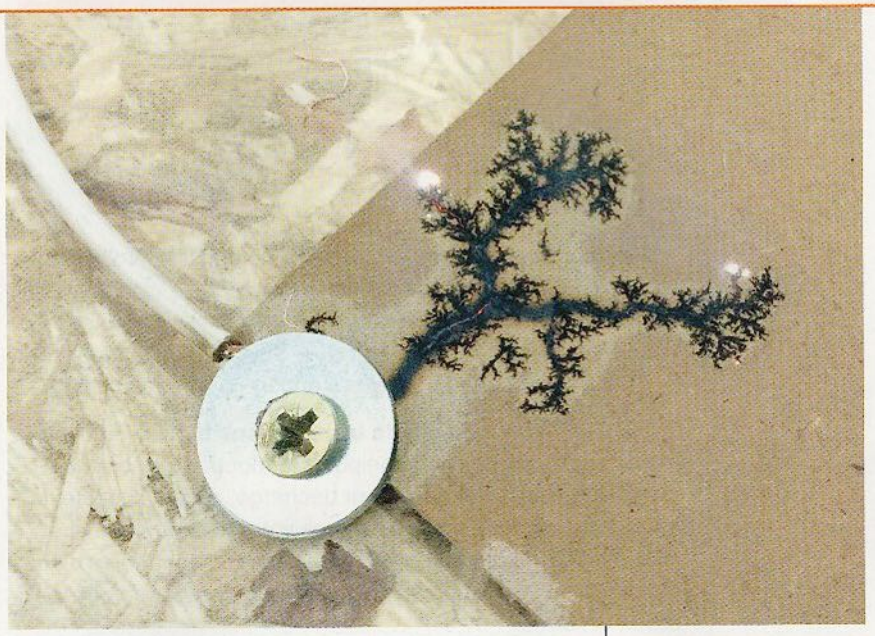
ôtez-vous cette idée même de l'esprit. Il n'y a pas de façon sûre de faire circuler un demi-ampère sous 4000V dans un morceau de bois couvert d'un liquide conducteur ! C'est une opération dangereuse et cela le restera quoi que vous fassiez. La seule manière d'être 100% sûr qu'il ne vous arrivera rien est de tout simplement vous abstenir de le faire. Et c'est précisément ce que je vous conseille.

Pour ceux d'entre vous qui n'ont pas l'intention de suivre ces sages paroles, que je n'ai moi-même pas suivi, voici quelques conseils élémentaires de prudence (que j'ai suivis à la lettre). Il n'y a pas d'ordre à cette liste, tout est important.

Ne faites pas confiance aux interrupteurs. Débrancher le dispositif est le seul moyen d'avoir la garantie qu'il n'est pas sous tension. À chaque fois que j'ai manipulé l'installation, celle-ci était physiquement déconnectée de toute alimentation.

Ne travaillez pas seul. Ayez quelqu'un à proximité en mesure de chercher du secours en cas d'accident et de vérifier que votre propre protocole de sécurité est respecté. Expliquez clairement à cette personne en quoi consistent vos manipulations et quels sont les risques.

Ne laissez rien traîner. Un environnement propre et rangé est un environnement plus sûr. Cette remarque est valable pour bien d'autres choses. La dernière chose que vous voulez c'est trébucher sur un câble ou un bout de bois qui traîne et tomber sur votre dispositif sous tension.



Gros plan sur l'une des électrodes en cours de tracé. Ici, celle-ci est placée en bordure du support afin d'éviter de laisser une « zone vide » et on constate clairement que le médium ou MDF, composé de particules de bois compressés, permet un rendu bien plus fin que le bois standard naturel, car parfaitement plat et lisse.

Ne portez pas de bijoux ou d'objets pouvant entrer en contact avec le montage. Pas de bagues, de clés, de pendentif, de montre, de smartphone...

Travaillez avec un éclairage suffisant. Vous devez être en mesure de voir ce que vous faites et d'envisager au mieux la situation.

Séparez clairement la zone de préparation et la zone d'expérimentation. Tout ce qui peut être fait loin de la source de courant doit l'être. Ces expériences impliquent de l'eau, qui ne fait **jamais** bon ménage avec de l'électricité. Éloignez la solution, la visserie, les outils... de la zone d'expérimentation. Quelques pas de plus ne sont pas chers payés face au risque de renverser le carbonate de soude en solution sur l'alimentation.

Portez des gants et des chaussures isolants. Achetez-vous des gants isolants électriques adaptés aux tensions utilisées (classe 1) et isolez-vous du contact avec le sol. N'oubliez pas que vous manipulez également un liquide susceptible de laisser passer le courant et qu'il peut donc compromettre votre isolation.

Vérifiez tout, plusieurs fois avant chaque manipulation ou mise en route. Vérifiez, vérifiez, et revérifiez encore. Lorsque vous êtes absolument sûr que tout est en ordre... vérifiez encore une fois ! N'assumez **jamais** de quoi que ce soit, assurez-vous-en.

Méfiez-vous du condensateur. Déchargez le condensateur en reliant les deux électrodes, une fois le montage hors tension et déconnecté de l'alimentation domestique. Mettez **toujours** le condensateur en court-circuit lorsque le montage n'est pas utilisé.

Pensez à ce que vous faites et à rien d'autre. Votre attention doit être entièrement dirigée vers vos manipulations. Si vous avez autre chose en tête, faites une pause et remettez-vous à l'ouvrage l'esprit vide et clair. Si vous êtes fatigué, remettez au lendemain, tout sera encore là, et vous aussi.

Enfin, **utilisez la règle de la main dans le dos** (ou de la main dans la poche). Ne manipulez le montage, même hors tension, même le condensateur déchargé, qu'avec une seule main, la seconde étant placée dans votre dos (ou dans une poche). Ceci ne garantit pas votre survie à un choc électrique, mais minimise le risque qu'une décharge passe par votre cœur, et l'arrête.

4. TECHNIQUES ET ASTUCES

Les images qui illustrent cet article proviennent d'une journée entière d'expérimentation. Durant ce temps, j'ai découvert un certain nombre de points intéressants.

Le support est important pour de bons résultats. Il semblerait que plus le bois est plat et lisse, plus les figures sont fines et visibles. Un morceau de bois à peine dégrossis, plein de veines et de relief semble favoriser le passage du courant dans les endroits où le liquide stagne. Les craquelures et fissures forment soit des barrières impossibles à franchir, soit des canaux où circule le courant en ligne droite. Le contreplaqué est plus réceptif à la gravure, mais ne donne pas d'aussi bons résultats qu'une plaque en aggloméré comme celles généralement utilisées pour les fonds de placards ou de tiroirs. Un ponçage est nécessaire pour permettre l'humidification et la pénétration de la solution.

La solution à base de sel ou de carbonate de soude s'évapore rapidement. Les électrodes ont tendance à chauffer, tout comme les canaux carbonisés qui se forment et où circule le courant. Aux abords de la figure, le liquide disparaît jusqu'à ce que le courant ne passe plus. Doser la quantité de solution à appliquer est dépendante du support et de son absorption. Ce n'est pas une science exacte, il est nécessaire de tâtonner à chaque

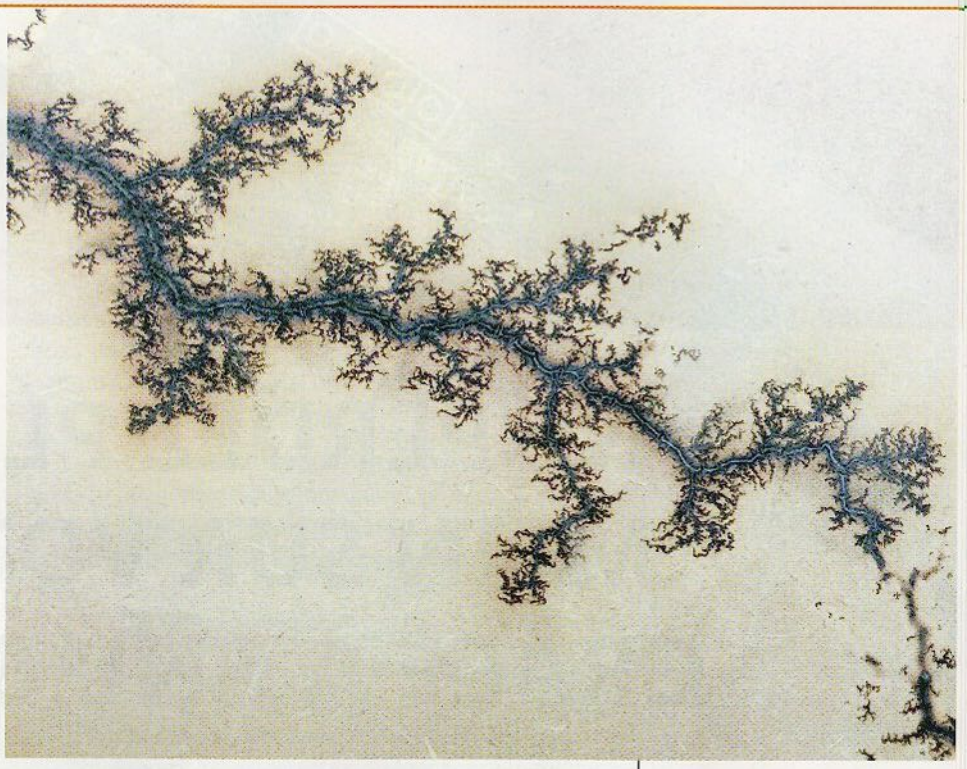
Après maints essais, les résultats se font de plus en plus appréciables. Voici celui obtenu en fin de journée en deux étapes. D'abord coin supérieur droit et inférieur gauche. Arrêt. Déconnexion. Application d'une nouvelle couche de solution et reconnexion, via le coin supérieur gauche et inférieur droit. Arrêt juste avant le contact des deux tracés.

fois. Il faut également faire très attention aux bordures du support, ayant tendance à accumuler davantage de solution et donc à fournir un passage préférentiel au courant, sans créer de tracés très artistiques. Deux ou trois pages web préconisent d'utiliser un vaporisateur pour humidifier le support en cours de gravure. Ceci me paraît être bien trop dangereux pour être ne serait-ce qu'envisagé, en ce qui me concerne.

L'utilisation d'un transformateur de four micro-onde présente un problème, car le courant fourni est très important. La brûlure du support semble être bien plus profonde que des figures réalisées avec un transformateur pour tubes néon. Ceci est particulièrement visible lorsque les deux tracés qui se dessinent finissent par se rejoindre. Un arc électrique unique se forme, et un courant important circule, gâchant souvent une figure prometteuse. Mieux vaut interrompre l'expérience juste avant ce contact.

Il peut être intéressant de stopper le tracé, débrancher l'alimentation, décharger le condensateur, déconnecter les électrodes pour réappliquer la solution sur le support et éventuellement déplacer les électrodes pour forcer le dessin d'une nouvelle figure qui finira par rejoindre la précédente.

De façon générale, c'est un processus de répétition tentatives/échecs, pour arriver à un bon résultat. Il y a toujours une part aléatoire et chaque figure est unique. Il ne faut pas partir dans l'idée que le premier tracé sera forcément une réussite.



Gros plan d'une figure sur panneau de particules. On distingue bien chaque petit chemin emprunté par le courant ainsi que le canal général en direction de l'électrode opposée.

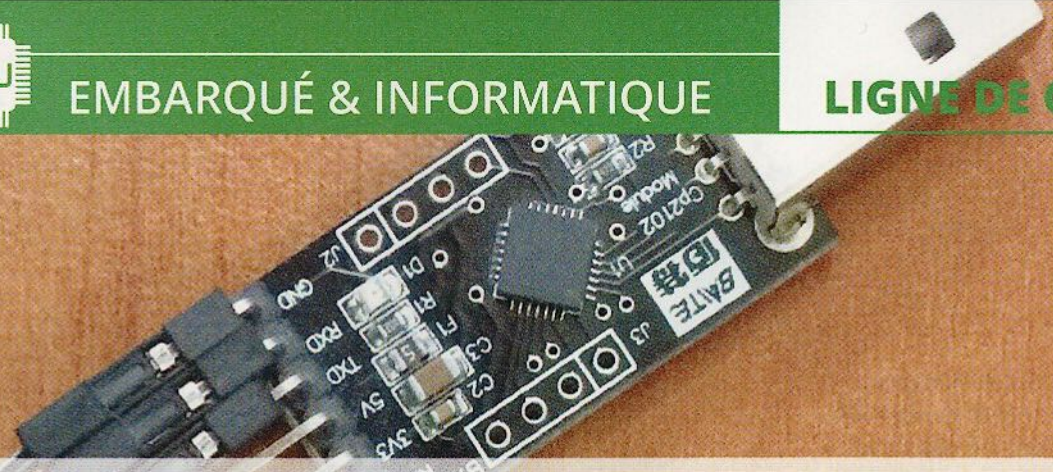
CONCLUSION ET PERSPECTIVES

En débutant mes expérimentations, j'avais bon espoir d'obtenir rapidement des résultats magnifiques. Ceci n'a, bien entendu, pas été le cas. C'est en forgeant qu'on devient forgeron, paraît-il. Je ne sais pas si cela est effectivement applicable pour le travail du métal, mais pour la réalisation de figures de Lichtenberg sur bois, c'est clairement le cas ! En fin de journée, mes « œuvres » étaient nettement de meilleure qualité que dans la matinée...

Contrôler avec précision la tension et le courant semble être capital pour obtenir des tracés moins aléatoires. Ceci demandera cependant davantage de recherche pour trouver des solutions adaptées, éventuellement pilotées numériquement avec un pupitre de contrôle et une liaison isolée par fibre optique (comme c'est le cas pour le kit musical One Tesla par exemple).

Il pourrait également être intéressant de voir comment guider le tracé, autrement qu'en fonction de l'application de la solution de carbonate de soude. Je pense, par exemple, à l'utilisation de graphite (crayon) permettant, peut-être de tracer une forme générale que la figure pourrait suivre.

Enfin, l'option des électrodes mobiles est également une piste à suivre, en permettant l'utilisation sous tension et l'application en direct en des endroits précis du support. **DB**



NE PERDEZ PLUS LA MAIN SUR VOTRE PI GRÂCE À GNU SCREEN

Denis Bodor

Le problème n'est ni nouveau ni rare, vous travaillez en ligne de commandes sur votre Raspberry Pi et êtes interrompu. Vous aimeriez bien retrouver votre session exactement comme vous l'avez laissé, mais si vous fermez le terminal ou vous vous déconnectez, vous vous retrouverez ensuite avec un shell tout neuf. Des solutions, il en existe plusieurs, l'une d'entre elles consiste à utiliser un multiplexeur de terminal comme GNU Screen.

GNU Screen fonctionne un peu comme une fenêtre de terminal dans une session graphique, il offre un cadre pour l'exécution d'un programme (typiquement un shell). Mais à la différence d'un LXTerminal qui ne fonctionne qu'en mode graphique, GNU Screen fera de même dans une console ou à l'intérieur même de LXTerminal. Il dispose de sa propre gestion d'onglets, permettant d'avoir plusieurs lignes de commandes indépendantes, mais peut aussi et surtout être mis en attente (détaché) pour être « repris en main » ensuite, depuis une autre session ou même à distance.

1. INSTALLATION ET PREMIERS PAS

Installer GNU Screen sur votre Pi est un jeu d'enfant puisqu'un simple `sudo apt-get install screen` fera l'affaire et vous disposerez alors de la seule commande fournie par le paquet : `screen`.

En lançant cette commande, vous serez accueilli par un message précisant la version utilisée et la licence d'utilisation, ainsi qu'une invitation à utiliser la barre d'espace pour continuer. Et là... rien du tout. Vous vous retrouvez sur la ligne de commandes telle qu'elle se présentait précédemment. Mais ne vous y trompez pas, vous êtes à présent effectivement dans GNU Screen, en train de faire fonctionner un shell.

Vous verrons plus loin dans l'article comment modifier la configuration pour rendre cet état de fait plus visible, mais en attentant, il est temps de découvrir quelques fonctionnalités. Vous êtes sur une ligne de commandes (dans un shell Bash) gérée par Screen dans une fenêtre. Cette notion de fenêtre peut être vue comme un équivalent aux onglets d'un émulateur de terminal comme LXTerminal.

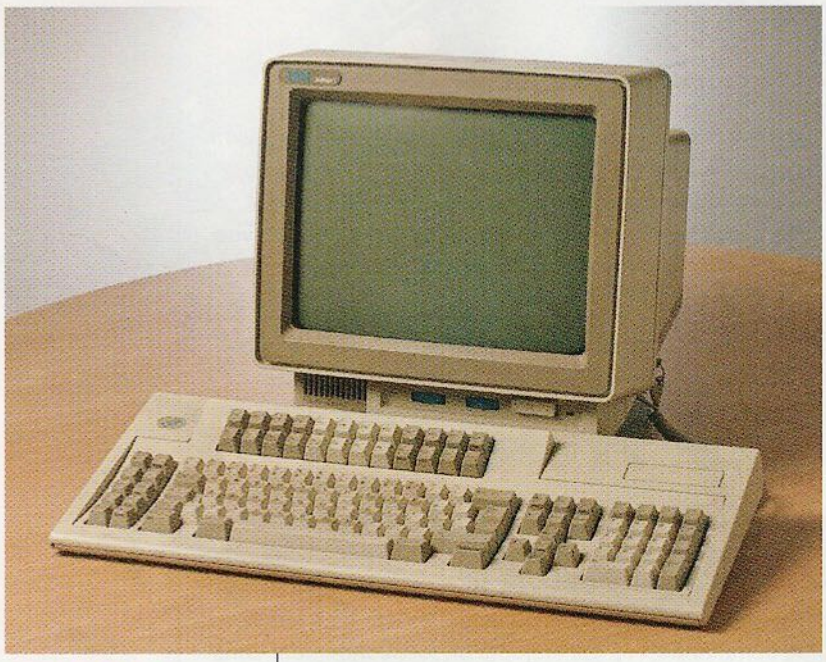
Lancez la commande `top` permettant d'afficher les processus en cours d'exécution sur votre Pi. Cette commande s'exécute de façon interactive et la liste est rafraîchie tant que vous n'utilisez pas la touche *q* pour quitter.

Utilisez à présent la séquence de touches *Ctrl+a* (souvent notée *^A*) puis *c*. Ceci aura pour effet de créer une nouvelle fenêtre Screen et vous verrez alors apparaître une nouvelle ligne de commandes. À ce stade, la commande `top` est toujours en fonctionnement dans la première fenêtre, mais vous disposez d'une nouvelle ligne de commandes pour travailler.

Vous pouvez à n'importe quel moment basculer de l'une à l'autre fenêtre en utilisant *Ctrl+a Ctrl+a*. Vous pouvez également utiliser *Ctrl+a* puis la barre d'espace qui a pour effet de passer à la fenêtre suivante (en bouclant sur la première une fois arrivé à la dernière). Vous pouvez accéder directement à la fenêtre de votre choix, pour les dix premières fenêtres, en utilisant *Ctrl+a* puis le numéro de la fenêtre (en commençant par 0). Ici, *Ctrl+a 0* vous montre la fenêtre avec `top` et *Ctrl+a 1* la nouvelle fenêtre avec la ligne de commandes.

Une dernière solution pour basculer d'une fenêtre à l'autre consiste à utiliser *Ctrl+a "* (guillemet de la touche "3"). Ceci provoquera l'affichage de la liste des fenêtres existantes, vous permettant une sélection avec les flèches haut/bas et un basculement en utilisant la touche *Entrée*. Vous remarquez, qu'en plus du numéro de fenêtre, un nom apparaît. Il s'agit par défaut de la commande avec laquelle a été ouverte la fenêtre en question. Ici `bash` est le shell obtenu à la création de chaque nouvelle fenêtre.

Vous pouvez utiliser *Ctrl+a c* autant de fois qu'il vous plaît pour créer autant de fenêtres/onglets qu'il vous faut. Mais, il va vite devenir difficile de vous y retrouver puisque, par défaut, Screen ne présente pas d'informations sur le numéro de la fenêtre courante. Une solution pour remédier à ce problème consiste à changer le nom de vos fenêtres.



Avant l'arrivée en masse de l'informatique personnelle, les ordinateurs étaient des machines massives auxquelles on se connectait à l'aide de terminaux. Bien que ce type d'équipement, principalement composé d'un écran et d'un clavier, ai presque totalement disparu aujourd'hui, le terme « terminal » est resté (Source Wikipedia & ClickRick / IBM 3486 terminal / CC BY-SA 3.0).

Revenez par exemple sur la fenêtre 0 (*Ctrl+a 0*), puis utilisez *Ctrl+a A* (majuscule). Screen vous affichera, au bas de l'écran, un message : « Set window's title to: bash ». Il vous suffira alors d'effacer « bash » et de saisir le nom de votre choix avant de valider. Dès lors, le nom affiché suite au *Ctrl+a "* correspondra à celui que vous avez spécifié.

2. GNU SCREEN ET CONNEXIONS DISTANTES

Le fait de disposer de fenêtres/onglets permettant d'utiliser la ligne de commandes directement depuis la console sans avoir à lancer toute l'interface graphique est une chose très pratique. Mais toute la puissance de Screen se fait jour dès lors qu'il s'agit d'une utilisation intermittente. Il peut s'agir, par exemple d'une utilisation dans un terminal graphique puis d'une reprise en mode texte ou encore via la console série, mais aussi et surtout de connexion réseaux.

Se connecter à distance, en ligne de commandes, sur sa Raspberry Pi se fait très simplement avec SSH. Il suffit de préciser l'utilisateur et l'adresse IP de destination sous la forme `ssh pi@192.168.10.10` par exemple. Et, bien entendu, cela fonctionne tout aussi bien entre un PC sous GNU/Linux et la Pi, que depuis une machine Windows (avec Putty par exemple), un Mac (SSH est installé par défaut) ou même une autre Pi.

Le fait de lancer Screen et d'utiliser le shell de cette façon vous permet alors de détacher Screen du terminal en cours d'utilisation (série, console, graphique), de le laisser suivre son cours avec toutes les commandes qui y sont lancées, pour enfin le réattacher à un nouveau terminal.

Pour détacher Screen du terminal en cours c'est tout simple, il vous suffit d'un *Ctrl+a d*, et vous vous retrouverez sur votre ligne de commandes habituelle, précédée d'un message comme « [detached from 973.pts-0.raspberrypi] ». Vous pouvez alors quitter votre terminal ou même fermer votre session en console, Screen sera toujours là à vous attendre. Ceci peut être très intéressant, par exemple si vous travaillez sur la Pi directement, puis devez vous éloigner pour ensuite établir une connexion SSH. Il vous suffira alors de réattacher Screen pour retrouver vos affaires là où vous les avez laissées.

Pour retrouver une session Screen détachée, s'il n'y en a qu'une seule, il vous suffira d'utiliser `screen -r`. Si la session n'a pas été détachée, parce qu'elle est toujours à l'écran et que vous avez dû partir précipitamment, pas de problème, vous pouvez la détacher à distance avec `screen -D` ou mieux encore, la détacher et la réattacher en une fois au terminal en cours avec `screen -rD`.

Par défaut, Screen utilisera la seule session qu'il trouvera. Rien ne vous empêche cependant de lancer plusieurs fois **screen** avec, pour chaque session, ses propres fenêtres/onglets. Si plus d'une session est en cours sur la machine, il vous faudra utiliser l'option **-ls** pour les lister :

```
$ screen -ls
There are screens on:
        2327.pts-0.raspberrypi   (03/11/2016 15:35:32)    (Detached)
        973.pts-0.raspberrypi    (03/11/2016 11:41:43)    (Attached)
2 Sockets in /var/run/screen/S-pi.
```

Il vous suffira ensuite d'utiliser **screen -r** (ou **-rD**) suivi du numéro figurant en début de ligne comme, par exemple ici, **screen -r 2327**. Ce chiffre correspond au numéro de processus ou PID de la commande **screen** que vous aviez lancée, suivi de l'identifiant de terminal (**pts-0**) et du nom d'hôte de la machine (**raspberrypi**). Dans la sortie de **screen -ls** sont également précisés la date et l'heure du lancement de la session Screen, ainsi que l'état de cette dernière (attachée ou non).

Pour vous faciliter les choses, et si le paquet **bash-completion** est installé, il ne vous sera même pas nécessaire de tout saisir. Ici, par exemple, un simple **screen -r 2** suivi d'une pression sur la touche de tabulation, complètera automatiquement la ligne.

Notez que si vous prenez l'habitude de lancer plusieurs sessions de Screen sur un même système, il pourra être judicieux d'utiliser des noms particuliers afin de repérer rapidement vos sessions. Les options **-t** et **-S** sont là pour ça. En lançant par exemple **screen -t configuration -S maintenance**, vous démarrerez une session appelée « maintenance » avec une première fenêtre appelée « configuration ». La sortie de **screen -ls** ressemblera alors à ceci :

```
$ screen -ls
There is a screen on:
        2482.maintenance         (03/11/2016 15:48:42)    (Detached)
1 Socket in /var/run/screen/S-pi.
```

Le numéro de processus est toujours là, mais le nom spécifié vous aidera à identifier la bonne session plus rapidement.

Une autre option qui peut s'avérer très intéressante, en particulier si vous aidez quelqu'un à distance, est **-x**. Celle-ci a pour effet de vous attacher à une session non détachée. De ce fait, une même session sera utilisable dans deux terminaux, ou en d'autres termes, l'écran sera alors partagé. En dehors de l'assistance, ceci pourra être utile dans le cadre d'une formation afin que l'enseignant puisse observer l'activité d'un utilisateur et même intervenir pour corriger un problème. Il lui suffira de se connecter en SSH à la machine et d'utiliser **screen -x**.

Toutes ces options, **-r**, **-rD** ou **-x** peuvent s'accompagner de **-p** en précisant le nom donné à une fenêtre pour directement y accéder et s'éviter une manipulation supplémentaire après avoir réattaché la session.

Enfin, pour quitter une session de Screen ou fermer une fenêtre, rien de plus simple : il vous suffit de quitter le shell qui y est lancé avec **exit** ou *Ctrl+d*. Il vous est également possible d'utiliser *Ctrl+a k* pour « tuer » la fenêtre et le programme qui y est en cours d'exécution. Lorsque la dernière fenêtre est fermée, Screen quitte automatiquement la session et se termine.

3. PERSONNALISATION DE GNU SCREEN

L'une des principales limitations qu'on rencontre en commençant à utiliser fréquemment GNU Screen concerne le fait que l'outil soit peu loquace. En effet, l'objectif initial du projet était d'ajouter des fonctionnalités aux terminaux tout en minimisant l'aspect intrusif. La première version de GNU Screen est apparue il y a presque 30 ans, un moment où la plupart des affichages en mode console se limitaient à quelques 80 colonnes (132 avec de la chance) et 25 lignes. Occuper ne serait-ce qu'une ligne pour afficher des informations revenait à perdre un précieux espace à l'écran.

Aujourd'hui la masse d'informations qu'il est possible d'afficher dans une console ou dans un émulateur de terminal nous permet de consacrer un peu de cet espace pour des données moins importantes que le contenu primaire.

GNU Screen utilise un fichier de configuration qu'il vous est possible de personnaliser. Il se trouve dans `/etc`, mais il sera préférable de ne pas toucher à un tel fichier système et de simplement composer un `.screenrc` placé dans votre répertoire personnel. Celui-ci sera utilisé en priorité et vous permettra d'ajuster la configuration et le comportement de GNU Screen pour l'utilisateur concerné uniquement.

Vous pouvez créer ce fichier avec votre éditeur préféré (Vim, Nano, Emacs, etc.), il s'agit d'un simple fichier texte contenant une suite de directives. Je vais vous donner ici ma configuration personnelle qui me satisfait depuis des années.

Nous commençons donc par ajouter une ligne :

```
startup_message off
```

Ceci aura pour effet de faire disparaître le message d'accueil présent au lancement de Screen et vous permettra d'accéder directement au contenu de la première fenêtre. Ce faisant, il devient difficile de savoir que Screen vient d'être lancé, mais nous allons corriger cela un peu plus loin.

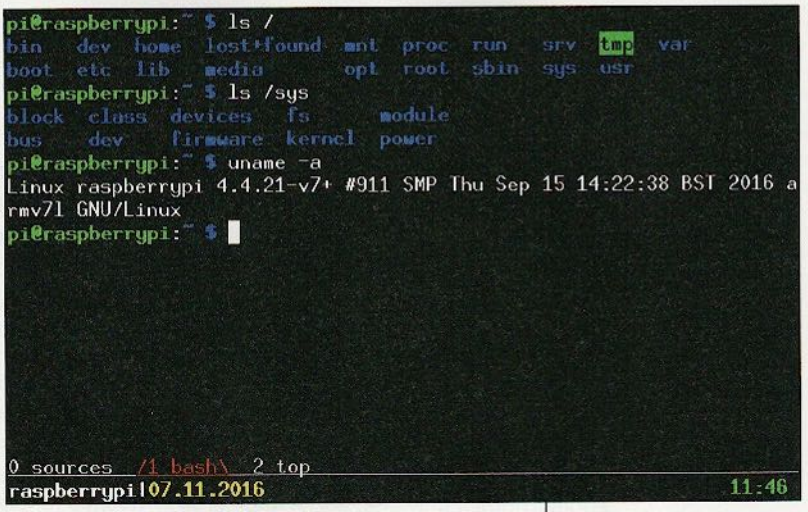
Une seconde ligne sera :

```
vbell off
```

Par défaut, Screen utilise ce qu'on appelle un signalement visuel. En utilisant un terminal ou une console sur PC, une erreur de manipulation est généralement signalée par un signal sonore. L'allergie des collègues aux bips intempestifs ou l'absence de sortie audio conduisent généralement à l'utilisation d'une équivalence visuelle via une inversion rapide des couleurs à l'écran. C'est le signalement visuel ou *visual bell* (« cloche visuelle ») en anglais. Par défaut, Screen utilise ce système, qui est pratique pour une Pi, mais utilisant généralement une connexion SSH vers mes Raspberry Pi, je préfère revenir au bip classique. C'est là l'objet de cette ligne.

Passons maintenant aux choses sérieuses en ajoutant quelques informations de base concernant la session en cours :

```
hardstatus on
hardstatus alwayslastline "%{+b kw}%H%{kg}|%c|%{ky}%d.%m.%Y"
```



Notre configuration finale nous permet d'avoir une session de GNU Screen présentant non seulement des informations complémentaires sur le système en bas d'écran, mais aussi, et surtout une gestion d'onglets bien visible et sympathique à l'œil.

hardstatus pour *hardware status line* ou « ligne d'état matérielle » en bon français, correspond à une fonctionnalité présente sur d'anciens terminaux physiques où une ligne supplémentaire de l'écran était **matériellement** dédiée à l'affichage d'informations. Le fait de spécifier à **on** cette option active l'utilisation et l'émulation de cette ligne.

Nous pouvons ensuite, après **hardstatus alwayslastline** plaçant la ligne au bas de l'écran, spécifier son contenu. Le format d'usage utilise des séquences d'échappement qui, au premier regard, peuvent sembler absconses. Chaque séquence débute par **%** et possède une signification particulière. Si nous simplifions celle présentée ici, en retirant la gestion des couleurs, nous avons :

```
hardstatus alwayslastline "%H|%c|%d.%m.%Y"
```

Avec respectivement pour :

- **%H** : le nom d'hôte de la machine,
- **%c** : l'heure courante au format « HH:MM » sur 24h,
- **%d** : le numéro du jour dans le mois,
- **%m** : le numéro du mois dans l'année,
- **%Y** : l'année complète (par opposition à **%y** qui n'affiche que les deux derniers chiffres).

Les caractères **|** et **.** sont simplement ajoutés pour embellir le tout et ajouter un peu de mise en forme. Ceci nous affichera donc quelque chose comme **raspberrypi|11:33|04.11.2016** au bas de l'écran.

Il est possible d'ajouter des couleurs et du formatage de texte en utilisant des séquences comme :

- **%{+b kw}** : passage en gras et fond noir (**k**) avec tracé en blanc (**w** pour *white*),
- **%{kg}** : vert (**g** comme *green*) sur noir,
- **%{ky}** : jaune (**y** comme *yellow*) sur noir.

Nous aurons alors au final : le nom d'hôte en blanc, l'heure courante en vert et la date en jaune, le tout en gras. Ceci nous permettra déjà d'avoir des informations de base et l'indication que nous utilisons bien Screen et non le shell seul.

Enfin, nous ajoutons, dans le même esprit, une nouvelle ligne d'état avec l'option **caption**. Celle-ci se distingue de **hardstatus** dans le sens où elle peut ne pas être toujours affichée. **hardstatus** est une vraie ligne d'état alors que **caption** n'est souvent affiché que s'il y a plus d'une fenêtre dans la session. Nous pouvons cependant forcer l'affichage et spécifier un contenu ainsi :

```
caption always "%{+u wk}%?%-w%?%{rk}/%n %t\%{wk}%?%+w%?"
```

L'utilisation des séquences d'échappement et des couleurs est exactement la même que pour **hardstatus**, mais nous avons ici quelque chose en plus : une condition. L'idée est de lister tous les noms de fenêtres existantes et de mettre en avant celle qui est active. En éliminant la gestion de la couleur, nous avons :

```
caption always "%?%-w%?/%n %t\%?%+w%?"
```

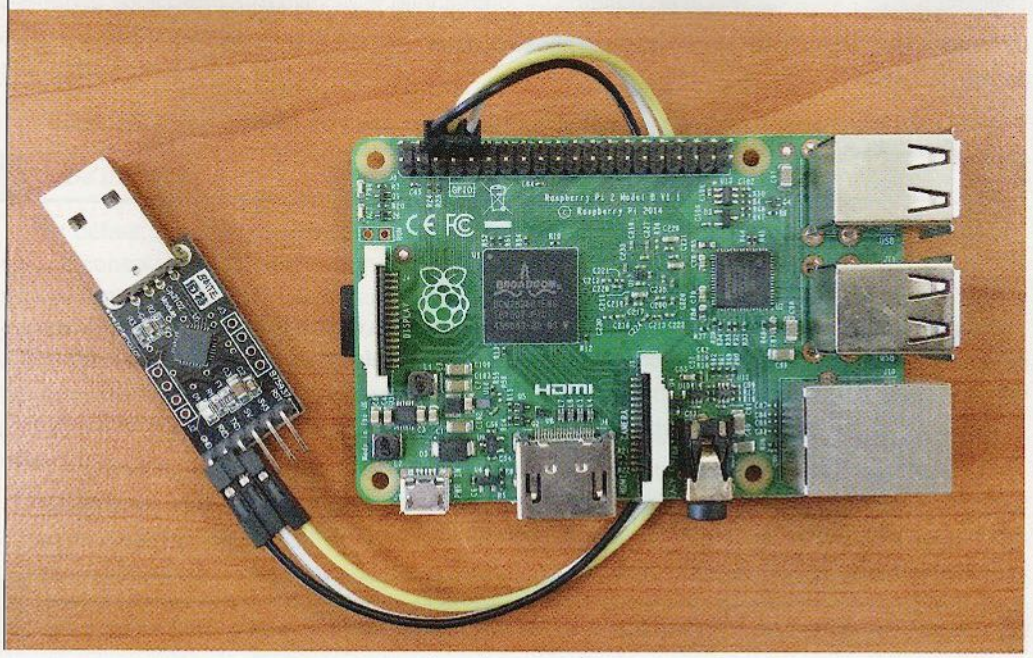
Pour décrypter tout cela, commençons par le plus simple :

- `%t` : le nom de la fenêtre courante,
- `%n` : le numéro de la fenêtre courante,
- `%-w` : le numéro et le nom de toutes les fenêtres avant celle en cours d'utilisation,
- `%+w` : le numéro et le nom de toutes les fenêtres après celle en cours.

Nous pouvons donc, de base, déjà afficher ce qui nous intéresse avec `"%-w/%n %t\%+w"` : numéros/noms de toutes les fenêtres avant la nôtre, de la nôtre, puis de toutes les autres. S'il n'y a pas de fenêtre avant la nôtre, nous affichons une chaîne vide, idem pour les fenêtres après la nôtre. Ceci n'est pas très propre et si vous souhaitez personnaliser davantage, mieux vaut prendre cette éventualité en compte.

C'est précisément là qu'intervient `%?`, signifiant : « afficher ce qu'il y a jusqu'au prochain `%?` à condition que cela ne se résume pas à une chaîne vide ». Ceci nous permet de conditionner l'affichage des fenêtres avant et après la nôtre. `%?%-w%?%` signifie ici « afficher `%-w` s'il y a effectivement des fenêtres avant celle en cours d'utilisation ».

Nous ne l'utilisons pas pleinement ici, mais cela pourra vous être utile pour créer votre propre configuration, il est possible d'avoir une condition « sinon » avec `%:`. Ainsi, avec `%?%-w%:*w%?`, nous décrivons : « si la liste des fenêtres précédentes ne donne pas une chaîne, afficher cette



GNU Screen sera également utile pour se connecter à la console série de votre Raspberry Pi. Il peut, en effet, être utilisé sur un système comme GNU/Linux ou Mac OS X pour communiquer avec la Pi via un adaptateur USB série comme celui présenté ici. Il suffit de le lancer en spécifiant le port série à utiliser (/dev/ttyUSB0 par exemple) et la vitesse (115200).

liste, sinon afficher le caractère * ». Ceci étant valable pour `caption` comme pour `hardstatus`, et pour n'importe quelle séquence d'échappement, nous obtenons une grande souplesse de personnalisation (couleurs, information, caractères supplémentaires, etc.).

En résumé, ce fichier de configuration vous permettra de supprimer le message d'accueil, d'ajouter des informations colorées concernant le système en bas d'écran ainsi qu'une visualisation rapide des fenêtres/onglets disponibles.

POUR FINIR

J'utilise GNU Screen depuis des années et ne pourrait plus m'en passer. C'est tout simplement l'un des quelques paquets que j'installe systématiquement (en compagnie de Vim et Midnight Commander) dès que je démarre pour la première fois une nouvelle Raspberry Pi sur Raspbian.

Il y aurait encore beaucoup de choses à dire sur l'utilisation et la personnalisation de votre configuration. Il est possible, par exemple, d'utiliser Screen comme émulateur de terminal pour se connecter à la console série d'une Pi, de diviser l'écran en plusieurs zones correspondant aux fenêtres et donc de garder un œil sur l'ensemble d'une session, de personnaliser les touches de raccourcis, ou encore de faire apparaître dans la ligne d'état différentes informations provenant de scripts.

Au-delà de la simple possibilité de détacher et d'attacher des sessions, GNU Screen peut rapidement devenir indispensable, pour peu qu'on prenne le temps de connaître l'outil et de l'adapter à ses préférences. Et c'est précisément ce que j'espère vous avoir donné envie de faire avec cet article... DB

rofessionnels, Collectivités, R & D...

HACKABLE MAGAZINE

M'abonner ?

Me réabonner ?

Choisir le papier, le PDF, la base documentaire, ou les trois ?

Permettre à mes équipes de lire les magazines en PDF, consulter la base documentaire ?

C'est possible ! Rendez-vous sur :

http://proboutique.ed-diamond.com

pour consulter les offres !

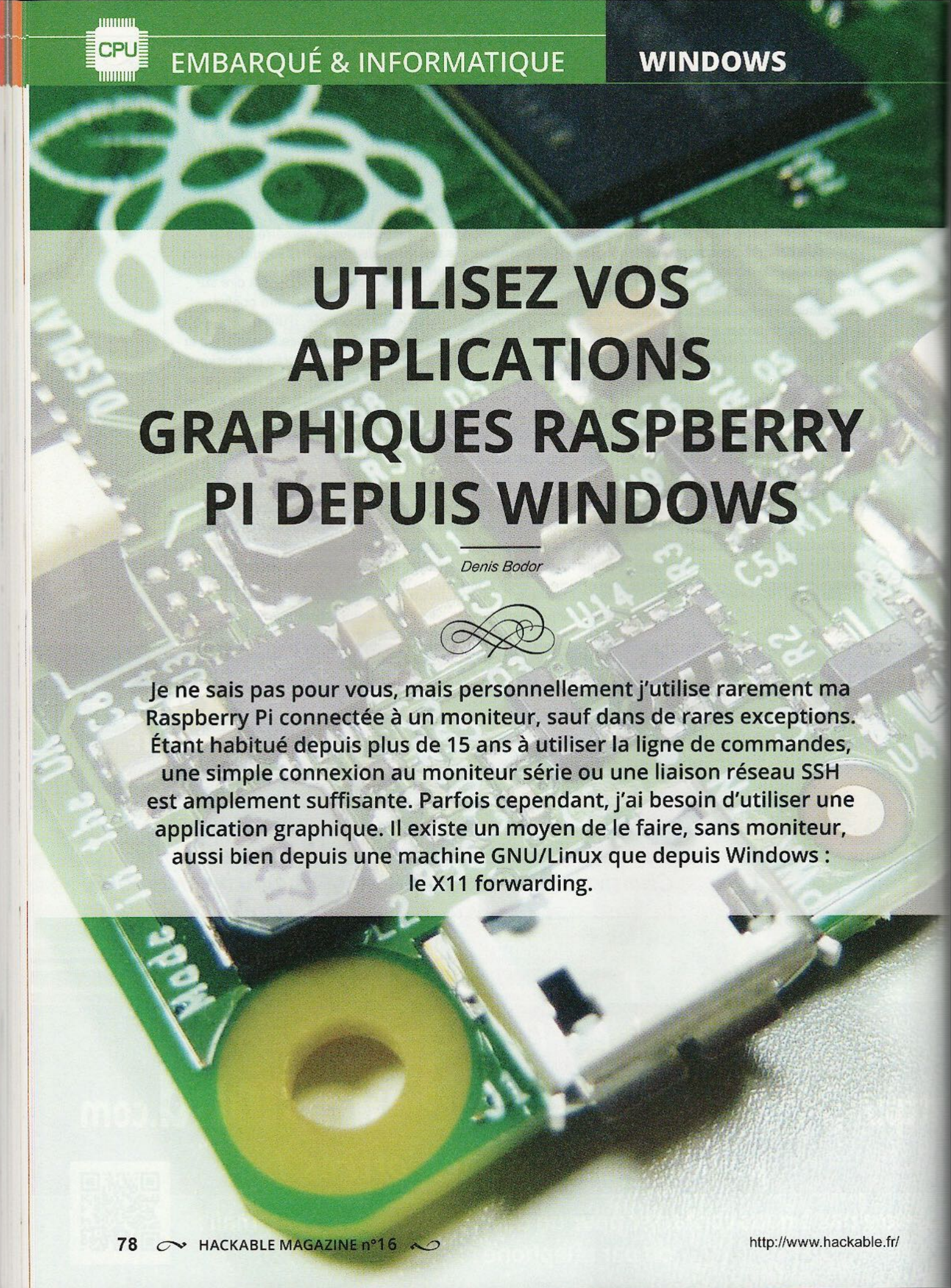
'hésitez pas à nous contacter pour un devis personnalisé par e-mail :

opro@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

UTILISEZ VOS APPLICATIONS GRAPHIQUES RASPBERRY PI DEPUIS WINDOWS

Denis Bodor

Je ne sais pas pour vous, mais personnellement j'utilise rarement ma Raspberry Pi connectée à un moniteur, sauf dans de rares exceptions. Étant habitué depuis plus de 15 ans à utiliser la ligne de commandes, une simple connexion au moniteur série ou une liaison réseau SSH est amplement suffisante. Parfois cependant, j'ai besoin d'utiliser une application graphique. Il existe un moyen de le faire, sans moniteur, aussi bien depuis une machine GNU/Linux que depuis Windows : le X11 forwarding.



La solution généralement préconisée pour obtenir un affichage graphique déporté consiste à utiliser un outil appelé VNC (*Virtual Network Computing*). Ceci est d'ailleurs décrit en détail dans la documentation officielle de la Fondation et consiste à lancer un serveur VNC (RealVNC) sur la Pi pour ensuite utiliser un client VNC sur Windows, Mac ou GNU/Linux et obtenir un accès au bureau. Cela est tellement bien intégré qu'il est possible d'activer cette option directement dans les préférences de l'interface graphique. Mais voilà, je suis vieux jeu, et je n'aime pas utiliser une solution alors qu'une autre existe depuis des lustres pour faire exactement la même chose... (comment ça « rétrograde » ? Pas du tout, « attaché à la culture UNIX », nuance).

VNC c'est très bien et très pratique, mais c'est oublier que, par défaut, le système graphique prend ce type de fonctionnalités en charge d'une autre façon depuis des temps immémoriaux. Le système Raspbian d'une Raspberry Pi est basé sur Debian, une distribution GNU/Linux connue et utilisée de longue date. Comme la quasi-totalité des distributions GNU/Linux, celle-ci utilise un environnement graphique reposant sur *X Window System* (alias X11, X Window ou tout simplement X). Le principe de fonctionnement est fort simple, X se charge de gérer le système d'affichage graphique ainsi que la prise en charge des périphériques d'entrée (souris, clavier, etc.) et permet à des clients d'afficher des « choses » (pixels, lignes, surfaces, arc de cercle, etc.) en passant par son intermédiaire.

Ceci est un terminal X de NCD datant des années 90, une simple interface moniteur/clavier disposant d'une connexion Ethernet et permettant d'utiliser des applications graphiques exécutées sur un ordinateur distant. Ce qui est détaillé dans cet article revient, dans les grandes lignes, à émuler ce type de matériel (Source Wikipédia / Richard J. Kettlewell).

Sans rentrer outre mesure dans les détails techniques, X fournit une solution d'affichage unifiée et détachée du matériel. Quel que soit le type d'adaptateur graphique dont vous disposez, que ce soit une Pi ou un PC avec une carte graphique, les « directives » utilisées par les programmes sont les mêmes. Le programmeur n'a pas à se soucier du matériel, il demande simplement que des pixels s'activent à l'écran.

Pour simplifier encore avantage les choses, des bibliothèques sont utilisées pour éviter de devoir gérer soi-même des choses comme le dessin de boutons, de fenêtres, de boites de dialogue, de menus... Ces bibliothèques fournissent ce qu'on appelle un toolkit graphique et les plus connues sont GTK+, Qt, Motif, Lesstif, ou encore Elementary. Le programmeur n'a ainsi qu'à demander « *je veux une fenêtre, avec une barre de menu, trois boutons à cet endroit...* » et tout ceci s'affichera parfaitement, quels que soient la plateforme et le matériel utilisé, du moment qu'un serveur X est en marche.

X fonctionne en mode client/serveur. Nous avons la partie serveur qui fait le travail et la partie cliente qui dicte ce qui doit être fait. Et c'est là que les créateurs de X, dès les premières heures de sa naissance, ont eu une excellente idée : faire fonctionner cette architecture client/serveur aussi bien localement qu'à travers le réseau. En fait, c'était même le concept de départ puisqu'à cette époque (milieu des années 80) le principe même qui était alors en place se résumait à un gros ordinateur et plein de terminaux pour pouvoir l'utiliser.

Il est encore aujourd'hui possible de reposer sur ce mécanisme, après des dizaines d'années d'évolution de X. Un client X peut, sur votre Raspberry Pi, afficher son interface sur un serveur X se trouvant sur une autre Pi, un PC, un Mac, etc. C'est ce qu'on appelle du *X11 forwarding*.

VNC en revanche utilise une approche différente qui revient à « attraper » les pixels s'affichant dans l'interface graphique de la Pi, par exemple, et les envoyer au travers du réseau pour les afficher dans la fenêtre de l'application VNC. Ce fonctionnement est plus proche de la notion de « bureau à distance » telle qu'on la retrouve dans Windows par exemple. C'est un déport de l'affichage existant. En termes d'utilisation à proprement parler, VNC est généralement utilisé pour exporter tout l'affichage alors que le X11 forwarding permet à une application graphique seule de s'afficher à distance. On se retrouve donc d'un côté avec tout un bureau et de l'autre avec uniquement la fenêtre de l'application dans son environnement graphique habituel, local.

1. PRINCIPE ET VÉRIFICATION

L'affichage déporté permis par le X11 forwarding est intégré de base dans X et il fonctionne dans les deux sens : que vous ayez un serveur X en marche sur votre Pi et demandiez à des applications locales de s'y afficher ou, au contraire que vous ayez un serveur X local et souhaitez lancer des applications de la Pi s'y affichant. Mais ceci est par défaut bloqué pour des raisons de sécurité. En effet, qu'il s'agisse de l'affichage des pixels ou des informations des périphériques d'entrée, tout ceci circulerait « en clair » sur le réseau et pourrait, en principe, être analysé et lu.

On n'utilise donc presque plus ce type de communication et c'est pourquoi elle est désactivée à la base (les serveurs X « n'écoutent » plus le réseau par défaut). Ce qu'il est possible de faire en revanche, c'est de laisser transiter ces données par un canal de communication qu'on sait sécurisé, car chiffré et authentifié, en d'autres termes, via une connexion SSH. SSH permet d'établir une connexion en ligne de commandes, parfaitement sécurisée, en spécifiant simplement un nom d'utilisateur et l'adresse de la Pi, puis en utilisant le mot de passe associé.

Depuis une machine GNU/Linux, un Mac ou une autre Pi, cela se résume à une commande du type `ssh pi@adresse` et vous voici avec un shell sur votre Pi cible. C'est également faisable depuis une machine Windows en utilisant un client SSH comme PuTTY.

Côté Pi, il faut simplement s'assurer que le serveur SSH, qui accepte ces connexions, autorise effectivement le transfert de données X11. C'est là quelque chose d'activé par défaut, en principe, mais il est de bon ton (et rassurant en cas de problème) de jeter un œil dans le fichier `/etc/ssh/sshd_config` et de s'assurer qu'une ligne `X11Forwarding yes` est bien présente et non commentée (non précédée d'un `#`). Si elle s'y trouve, votre configuration est déjà prête pour la suite, vous n'avez rien de plus à faire.

2. ENTRE UN GNU/LINUX ET UNE PI

Toutes les distributions GNU/Linux s'accompagnent généralement d'un serveur X, d'une interface graphique et d'un client SSH. Tout ce que vous avez à faire se résume à ouvrir une fenêtre de terminal et d'entrer la commande **ssh -X pi@adresse**, où **adresse** est l'adresse IP de votre Raspberry Pi. S'il s'agit de votre première connexion en SSH, un message vous demandera de confirmer l'identité de la machine à laquelle vous tentez de vous connecter :

```
The authenticity of host '192.168.10.234 (192.168.10.234)' can't be established.
ECDSA key fingerprint is 80:41:e0:af:74:08:96:d0:6d:6f:a1:0b:14:f8:5d:55.
Are you sure you want to continue connecting (yes/no)?
```

La connexion SSH permet deux choses : une communication chiffrée et une authentification. Ce second point fonctionne dans les deux sens. Le client prouve qui il est en vous demandant de saisir votre mot de passe, mais inversement, le serveur prouve son identité en vous donnant son empreinte (*fingerprint*). Il s'agit d'un processus cryptographique et cette empreinte qui s'affiche ne peut provenir que du serveur disposant d'un certificat qui lui est propre.

À ce stade en principe, vous êtes censé vérifier que l'empreinte affichée correspond bien à la machine (la Pi) à laquelle vous tenter de vous connecter, pour vous assurer qu'on ne détourne pas votre communication. Cette empreinte unique à un système peut être obtenue sur la Pi avec la commande :

```
$ ssh-keygen -lf /etc/ssh/ssh_host_ecdsa_key
256 80:41:e0:af:74:08:96:d0:6d:6f:a1:0b:14:f8:5d:55
   /etc/ssh/ssh_host_ecdsa_key.pub (ECDSA)
```

Si la série de chiffres et de lettres est identique, vous savez que vous vous connectez à la bonne cible, et vous pouvez alors taper **yes** pour continuer. Votre mot de passe utilisateur vous sera ensuite demandé pour que, à votre tour, vous prouviez qui vous êtes. Une fois celui-ci validé, vous obtiendrez une ligne de commandes.

Notez qu'il vous est possible de « simplifier » la vérification de l'empreinte en utilisant l'option **-v** côté serveur pour obtenir :

```
$ ssh-keygen -lvf /etc/ssh/ssh_host_ecdsa_key
256 80:41:e0:af:74:08:96:d0:6d:6f:a1:0b:14:f8:5d:55 /etc/ssh/ssh_host_
ecdsa_key.pub (ECDSA)
+---[ECDSA 256]---+
|.oo=o   ...E     |
|o.+ ooo          |
|.=.o.+..         |
|..ooo o.         |
|  o.oo  S        |
| . o.            |
|  .              |
|                 |
|                 |
+-----------------+
```

Et l'option **-o VisualHostKey=yes** (ou ajouter **VisualHostKey=yes** dans le fichier **~/.ssh/config**) côté client :

```
$ ssh -o VisualHostKey=yes pi@192.168.10.234
The authenticity of host '192.168.10.234 (192.168.10.234)' can't be established.
ECDSA key fingerprint is 80:41:e0:af:74:08:96:d0:6d:6f:a1:0b:14:f8:5d:55.
+---[ECDSA 256]---+
|.oo=o  ...E      |
|o.+ ooo          |
|.=.o.+..         |
|..ooo o.         |
|  o.oo  S        |
| . o.            |
|  .              |
|                 |
|                 |
+-----------------+
Are you sure you want to continue connecting (yes/no)?
```

Avec ces options, une représentation semi-graphique de l'empreinte accompagne la version textuelle. Elle n'est pas forcément intéressante pour une première connexion, où une vérification méticuleuse s'impose, mais plutôt lors des connexions suivantes. Si le dessin change, quelque chose de pas très net est en train de se passer...

Mais revenons à nos moutons. C'est ici l'option **-X** qui génère toute la magie du X11 forwarding. Avec la ligne de commandes obtenue arrive à présent automatiquement le transfert des données X. Ceci est rapidement vérifiable en utilisant :

```
$ echo $DISPLAY
localhost:10.0
```

$DISPLAY est une variable d'environnement. Celle-ci, lorsqu'elle existe, contient la désignation de l'affichage que doivent utiliser les applications graphiques. Si vous utilisez cette commande dans un terminal dans l'interface graphique de la Pi, vous obtiendrez quelque chose comme **:0**, à savoir l'affichage numéro 0. Ici, nous avons quelque chose d'autre : un nom de machine (d'hôte) et un numéro. **localhost** est un nom générique désignant la machine elle-même. L'affichage désigné est donc le numéro 10, premier écran, de la machine elle-même. Étrange, non ?

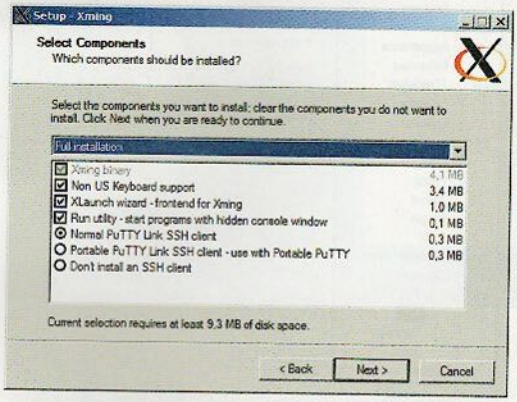
En réalité, SSH travaille en coulisse et se charge comme un grand de prendre tout ce qui arrive à cet endroit pour le faire transiter par ses propres canaux de communication (principe du « tunnel »). Ce qui ressort à l'autre bout de la connexion est alors envoyé directement au serveur X qui suppose qu'il s'agit d'informations locales (puisque fournies par le client SSH). Du point de vue de X, le réseau n'est donc pas physiquement utilisé (du moins en dehors du système).

De ce fait, si vous lancez une commande graphique, comme **xclock** par exemple (paquet **x11-apps**), la fenêtre va apparaître non pas sur la Pi, mais sur l'écran de la machine où vous avez lancé la commande **ssh**.

Notez cependant que celle-ci s'exécute bel et bien sur la Pi et non sur votre PC. Ceci signifie aussi que les fichiers que vous ouvrirez, par exemple, avec une application ainsi lancée, seront également ceux de la Pi. Il n'y a que la fenêtre qui s'affiche de votre côté, tout le reste se trouve du côté Raspberry Pi.

3. ENTRE UN PC WINDOWS ET UNE PI

Windows, contrairement aux distributions GNU/Linux n'est livré en standard, ni avec un client SSH, ni avec un serveur X. Il vous faudra donc installer ces deux éléments pour pouvoir utiliser ces fonctionnalités. Il existe plusieurs projets et plusieurs façons de faire et nous avons déjà évoqué dans les pages du magazine l'une des options : Cygwin (*Hackable n°3*).



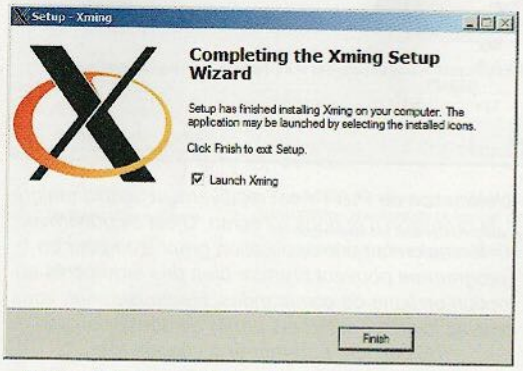
Xming permet d'installer en plus du serveur X différents éléments, dont un client SSH qui n'est autre que le classique PuTTY. Ceci vous évite donc de l'installer en parallèle et donc de faire d'une pierre deux coups : graphique et mode texte.

Ce projet consiste à permettre l'utilisation de presque la totalité de ce qui existe sous GNU/Linux dans Windows en se basant sur une solution simple : une bibliothèque permettant d'utiliser des versions Windows des programmes GNU/Linux avec un minimum de changement pour les développeurs. Ceci nécessite cependant soit d'installer les versions Cygwin de tous les programmes et applications (dont un serveur X) soit de recompiler, pour Cygwin, ces éléments en utilisant les sources originales.

Installer un serveur X sous Windows se fait comme n'importe quelle application, une suite de clics frénétiques sur « Next » sans rien lire et le tour est joué...

Une autre solution consiste à se tourner vers un autre projet appelé Xming (**http://www.straightrunning.com/XmingNotes/**) regroupant un serveur X et un client SSH (PuTTY). Je n'aime pas particulièrement la politique mise en œuvre pour ce projet consistant à débloquer l'accès à la dernière version dite « Website Build », contre un don de 10 livres sterling ou une contribution au projet. Cependant, dans une démarche d'essai nous pouvons nous contenter de la version publique disponible sur **https://sourceforge.net/projects/xming/files/Xming**. Celle-ci est bien plus ancienne (version 6.9 contre 7.7), mais fonctionne très bien.

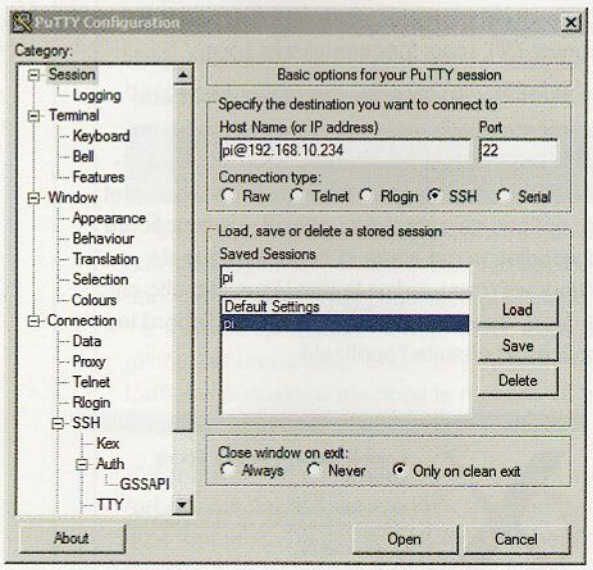
Vous devrez télécharger et installer deux éléments : `Xming-6-9-0-31-setup.exe` qui est l'applicatif lui-même et `Xming-fonts-7-7-0-10-setup.exe` qui est un ensemble complet de polices de caractères pour X. La procédure d'installation est aisée et assez typique de Windows (c'est-à-dire le classique clic-clic-clic, ***Suivant***, ***Suivant***, ***Terminer***). Installez d'abord les polices et ensuite l'applicatif.



Au terme de la procédure d'installation, il vous sera proposé de lancer automatiquement le serveur X. C'est une opération qu'il ne vous faudra pas oublier lors des démarrages suivants, et ce avant la connexion SSH à votre Pi.

En fin d'installation, une fenêtre avec une case précochée vous propose de lancer directement le serveur X. Vous cliquez et... rien ne se passe. Du moins, rien de visible, si ne n'est l'apparition d'une petite icône dans la barre des tâches (le *systray*) vous indiquant que le serveur fonctionne. En effet, un serveur X avec aucune application ne souhaitant l'utiliser n'a finalement rien à afficher. Il n'en reste pas moins qu'il est en fonctionnement et prêt à faire son travail.

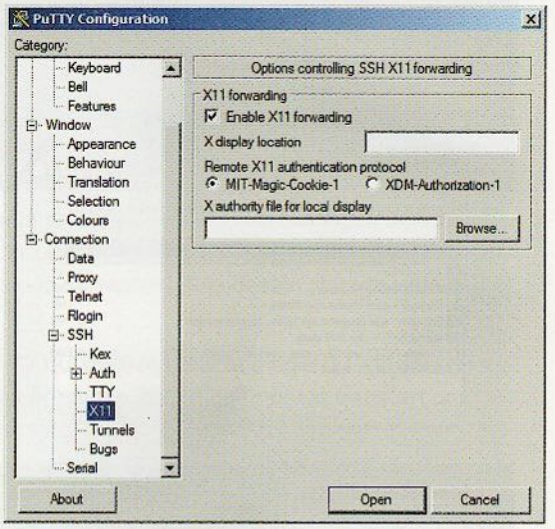
Les options d'installation par défaut permettent la mise en place de PuTTY, sans doute le client SSH le plus utilisé sous Windows. Vous trouverez celui-ci directement dans le menu « Démarrer » en cherchant son nom. Dans la fenêtre qui apparaît lors de son lancement, vous pouvez préciser le nom d'utilisateur et l'adresse IP de votre Raspberry Pi sous la forme `pi@adresse`.



L'utilisation de PuTTY est relativement simple malgré la profusion d'options à l'écran. C'est ce qui arrive lorsqu'on fait une application graphique pour un programme pouvant prendre bien des arguments en option en ligne de commandes. Heureusement, vous pouvez ici gérer différents profils de connexion afin de ne pas tout paramétrer à chaque fois...

Pour utiliser le X11 forwarding, naviguez dans les options sur la gauche, section ***Connexion***, ***SSH*** et ***X11***, puis cochez la case ***Enable X11 forwarding***.

Enfin, cliquez sur ***Open*** pour déclencher la connexion. Lors de votre première connexion, le même phénomène que sous GNU/Linux va se produire avec une demande de confirmation de l'empreinte du serveur, puis la demande de mot de passe. Le résultat sera également le même une fois ce mot de passe validé : vous obtenez une ligne de commandes sur votre Pi.

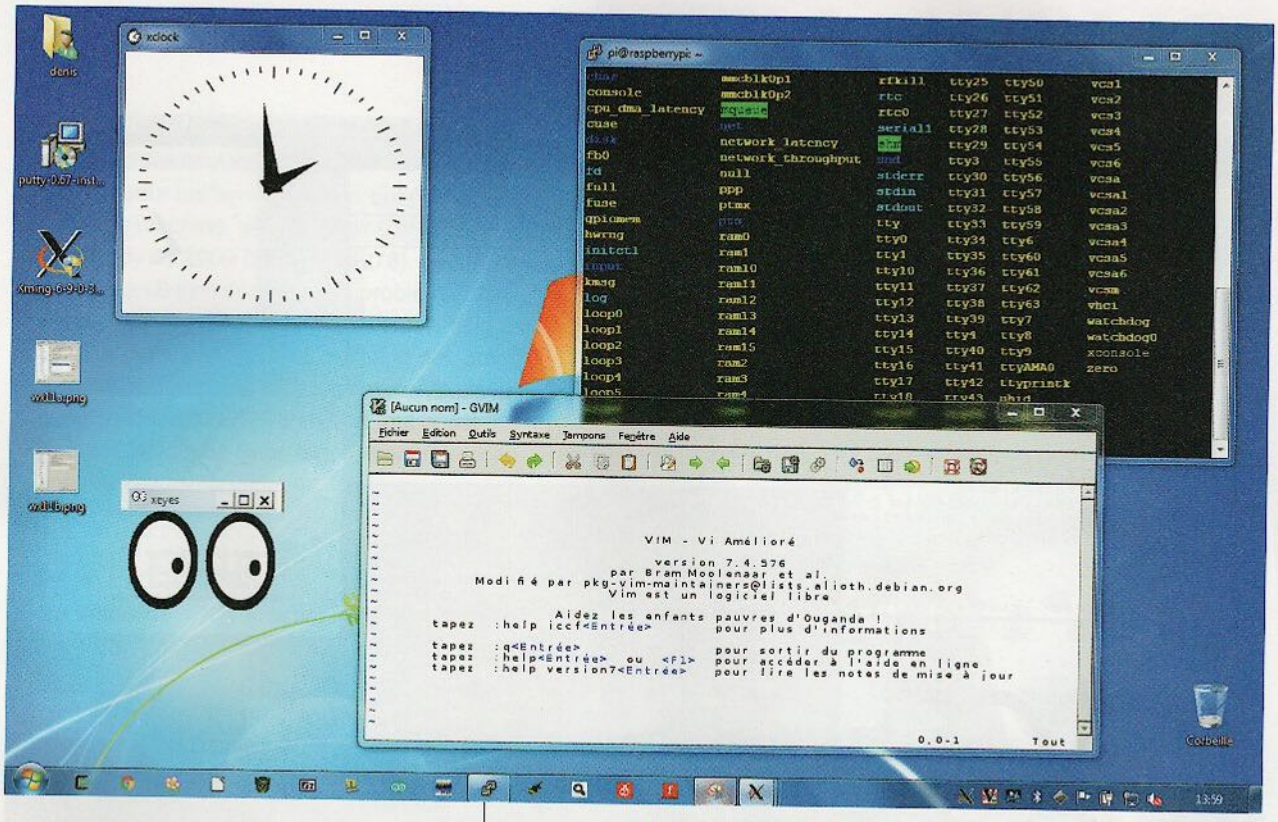


L'option importante ici concerne le X11 forwarding. Une fois celle-ci activée, vous pourrez lancer tout aussi bien des commandes en « mode texte » sur votre Raspberry Pi que des applications graphiques, une fois la connexion établie.

Là, vous pouvez immédiatement vous amuser à lancer n'importe quelle application graphique et son affichage apparaîtra immédiatement (ou presque) sous la forme d'une fenêtre Windows. Mais ne vous y trompez pas, ces applications s'exécuteront effectivement sur la Pi, avec les bibliothèques présentes sur la Pi et tous les éléments graphiques installés sur la Pi (Figure en haut, ci-contre).

4. ENTRE UN MAC ET UNE PI

Mac OS X (ou OS X, ou encore dernièrement, macOS) est à la base un système UNIX, comme GNU/Linux. Mais celui-ci n'utilise pas un système



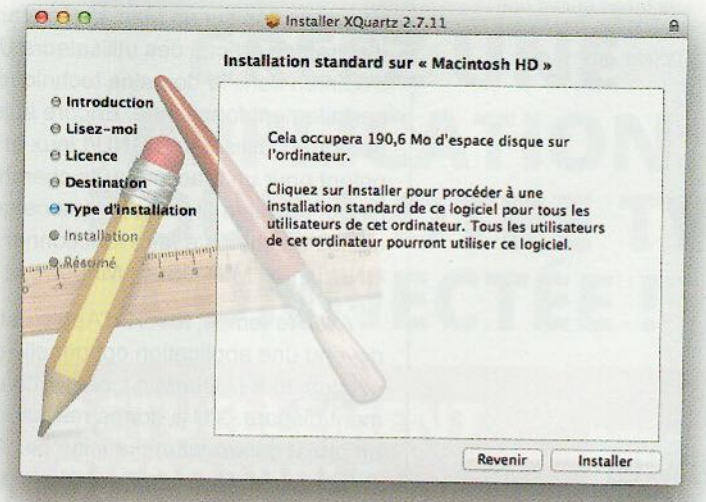
Les autres captures Windows de cet article ont bien été faites sous Windows 7 malgré leur apparence très XP. Ici, je suis délibérément (et temporairement) repassé en thème Aero juste pour le plaisir de vos yeux et vous faire profiter du mélange X11 et Windows 7 (si ça c'est pas une preuve d'amour pour mes adorables lecteurs... bon maintenant on remet tout comme c'était avant et on redémarre vite sous Debian !)

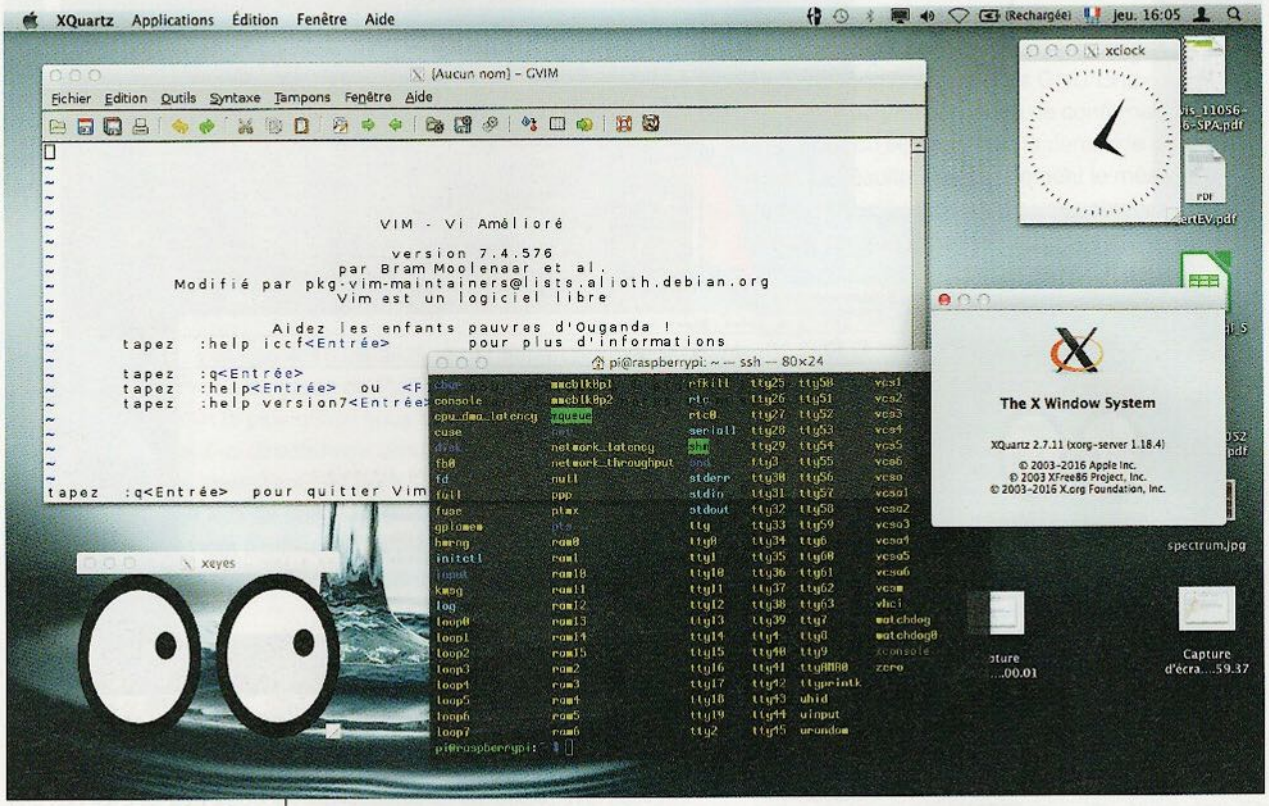
d'affichage X11, mais une solution 100% Apple appelée Quartz. SSH en revanche, tout comme un émulateur de terminal, est disponible par défaut aussi bien pour la partie serveur que cliente, vous n'avez donc rien à installer à ce niveau.

Concernant le serveur X en revanche, il faudra vous rendre sur **https://www.xquartz.org** afin de télécharger un fichier DMG, compatible Mac OS X 10.6.3 ou supérieur, l'ouvrir et double-cliquer sur le *XQuartz.pkg* qu'il contient pour procéder à l'installation. Enfin, un message vous précisera que vous devez vous déconnecter et vous reconnecter (fermeture/ouverture de session) pour que les changements soient pris en compte.

Il est amusant de remarquer qu'Apple, lors de l'arrivée des Mac et de leur « révolutionnaire » système Mac OS X, a rapidement intégré une application appelée simplement X11 permettant de faire précisément ce que nous décrivons

XQuartz remplace l'application X11 depuis Mac OS X 10.8. L'installation se fera très simplement après téléchargement d'un fichier DMG directement sur le site du projet qui est maintenant un effort communautaire auquel Apple prête assistance.





L'intégration du serveur X dans Mac OS X est bien plus élégante que dans Windows et ne comporte aucun problème particulier. Le serveur se lance même de lui-même dès lors qu'on établit une connexion SSH avec l'option -X.

ici. Tenant à afficher l'héritage UNIX du système, provenant de NeXTSTEP (cf. *Hackable n°4* et son article sur la résurrection d'une station NeXTstation Turbo), l'intégration de X11 permettait d'exécuter facilement n'importe quelle application X, recompilée sur un Mac. L'idée était, je pense, d'attirer les grâces des utilisateurs UNIX, leaders d'opinion dans le domaine technique, et cela a parfaitement fonctionné. Encore aujourd'hui, de nombreux utilisateurs GNU/Linux et surtout BSD, optent pour un Mac lorsqu'ils cherchent un ordinateur portable (il faut dire que ce que sont les seules machines à faire fonctionner facilement GNU/Linux, Mac OS X, Windows et BSD).

Avec le temps, le X11 d'Apple est ensuite devenu une application optionnelle puis, avec la version 10.8 (*Mountain Lion*), cette option a totalement disparu. X11 a donné naissance à XQuartz, un projet communautaire initié par Apple, mais indépendant. Une page web du support Apple précise cependant que « *Apple est toujours impliqué dans le projet XQuartz. Nous nous assurons notamment que le logiciel X11 fonctionne correctement sous macOS et avec les dernières versions de XQuartz* ».

Quoi qu'il en soit, passé cette installation, vous disposerez d'un serveur X11 parfaitement fonctionnel. Mieux encore, et il faut bien reconnaître ça à Apple (malgré plein d'autres choses moins plaisantes), cette intégration est de très bonne facture. Le simple fait d'ouvrir un terminal pour procéder à la connexion SSH avec l'option **-X** provoque le lancement du serveur X. Il n'est donc même pas utile de le lancer soi-même.

De plus, l'affichage est très rapide, contrairement à la solution Windows. Les tests que j'ai effectués pour les trois systèmes ont été faits sur une même machine, un MacBook Core 2 Duo 2.0 Unibody de 2009 (modèle A1278). GNU/Linux et sa Debian 8 s'en sortent sans problème bien entendu, Windows 7 Pro 32 bits et son Xming est à la traîne avec une faible réactivité et des problèmes pour reconnaître l'organisation du clavier Mac, et Mac OS X 10.7.5 s'en sort tout aussi bien que GNU/Linux sinon légèrement mieux en termes de réactivité.

POUR FINIR

La solution du X11 forwarding n'entre pas directement en concurrence avec celle reposant sur VNC. L'objectif n'est en effet pas tout à fait le même. Il ne s'agit pas ici d'un accès au bureau à distance, mais de tout autre chose. La Raspberry Pi que j'ai utilisée pour les tests, par exemple, a été initialement installée sans interface graphique (Raspbian Lite) et je n'ai, bien entendu, pas eu besoin de l'installer pour utiliser ainsi des applications graphiques.

Un autre point important concerne l'architecture même de l'ensemble. Le X11 forwarding impose l'existence d'un serveur X et même si les trois systèmes courants que sont GNU/Linux, Windows et macOS possèdent des solutions adaptées, il n'en va pas de même sous Android, par exemple (il existe des serveurs X comme XSDL, mais on ne peut pas vraiment dire que ce soit utilisable).

VNC repose sur l'utilisation d'une application cliente et non d'un serveur d'affichage. Ceci implique que l'application doit exister pour la plateforme que vous souhaitez utiliser, certes, mais il ne s'agit que d'une application. Un autre exemple est celui de l'utilisation de VNC dans un navigateur web. Il existe des extensions pour Chrome et Firefox faisant office de clients VNC alors qu'il n'existe pas d'équivalent X.

En résumé, il s'agit de deux solutions très différentes ayant chacune leurs avantages et leurs inconvénients. Il vous revient d'opter pour l'une ou l'autre en fonction de vos besoins ponctuels. DB

NE MANQUEZ PAS LA NOUVELLE FORMULE !

GNU/LINUX MAGAZINE n°200

CRÉEZ UNE APPLICATION POUR VOTRE TV CONNECTÉE !

ACTUELLEMENT DISPONIBLE

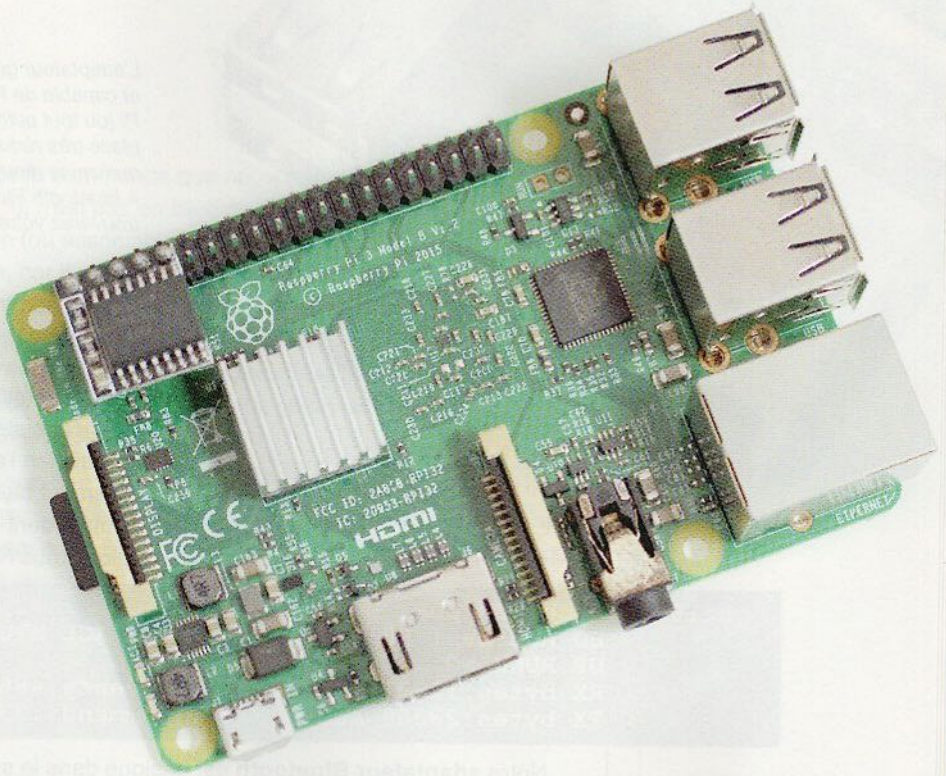
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

CONTRÔLEZ VOS MONTAGES BLUETOOTH DEPUIS VOTRE PI

Denis Bodor

Dans le précédent numéro, nous avons décrit comment ajouter une connectivité Bluetooth à un projet Arduino et ainsi pouvoir lui communiquer des ordres depuis un smartphone Android. Mais il peut également être très intéressant de remplacer le smartphone par une carte Raspberry Pi et ainsi gagner en souplesse et surtout ne plus être dépendant d'une application figée, peu configurable, difficile, voire impossible à modifier.



La carte Raspberry Pi 3 intègre de base un support pour le Wifi et le Bluetooth (Classic et Low Energy). Aucun accessoire complémentaire ne sera alors nécessaire, si ce n'est l'indispensable radiateur permettant d'éviter des plantages en cas de très forte charge du système. Le module enfiché à gauche du logo Pi est une horloge permettant de conserver l'heure entre deux redémarrages (module RTC).

Comme je l'ai dit dans le précédent numéro, s'attacher à la création d'une application pour smartphone, qu'il s'agisse d'Android ou iOS, n'est pas une tâche aisée. Il y a beaucoup à apprendre et cela demande un certain temps avant d'acquérir une confiance suffisante passé la première étape de l'application « Bonjour monde ».

Si l'on souhaite contrôler ou communiquer avec un montage à base d'Arduino et d'un module Bluetooth, il est bien plus simple de le faire avec un système plus facile à prendre en main : typiquement, une Raspberry Pi et quelques lignes de Python. Mais pour cela, encore faut-il connaître et savoir utiliser les bonnes commandes permettant de contrôler le Bluetooth sur une Pi.

La Raspberry Pi 3 intègre le support du Wifi et du Bluetooth, *Classic* comme *Low Energy* (BLE), mais si vous ne disposez que d'un précédent modèle, l'ajout d'une simple clé USB règlera vite le problème et vous apportera les fonctionnalités souhaitées. Un modèle minuscule souvent recommandé est fabriqué par Plugable Technologies sous le nom USB-BT4LE. Celui-ci vous coûtera quelques 15€, se trouve facilement sur Amazon par exemple et vous fournira les mêmes fonctionnalités que celles disponibles sur une Pi 3 (à l'exception du Wifi bien sûr).

Les explications qui vont suivre sont relativement génériques et s'adapteront à n'importe quel montage. Le croquis Arduino correspondant au montage que nous cherchons à contrôler est disponible dans le dépôt GitHub du précédent numéro à l'adresse **https://github.com/Hackable-magazine/Hackable15**.

1. PRENDRE EN MAIN LE BLUETOOTH ET VÉRIFIER QUE TOUT MARCHE

Il est possible de configurer et gérer le Bluetooth sur la Raspberry Pi en utilisant l'environnement graphique, mais ceci reste très limité et très orienté vers l'utilisation de périphériques comme un clavier ou une souris Bluetooth. Ce qui nous occupe ici est sensiblement différent puisqu'il s'agit non pas de faire en sorte que le système utilise le périphérique, mais de contrôler directement un montage « maison » communiquant en Bluetooth.

De plus, en cas de problème, il est important de connaître la mécanique qui se trouve sous le capot. Notre voie sera donc la ligne de commandes, offrant un accès complet à la configuration.

L'adaptateur généralement recommandé et capable de fournir du Bluetooth à une Pi (ou tout autre système) occupera une place très réduite et ne vous coûtera qu'une quinzaine d'euros. Une simple recherche de « Bluetooth Plugable » sur Amazon et vous trouverez votre bonheur...

La première manipulation que nous allons effectuer consistera à nous assurer que le Bluetooth, aussi bien l'aspect matériel que logiciel, est effectivement bien pris en charge. Pour ce faire, utilisez simplement la commande **hciconfig**, qui devrait alors vous retourner quelque chose comme :

```
hci0:   Type: BR/EDR  Bus: UART
        BD Address: B8:27:EB:10:1F:A8  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING
        RX bytes:108313 acl:0 sco:0 events:4865 errors:0
        TX bytes:28350 acl:0 sco:0 commands:3343 errors:0
```

Notre **adaptateur Bluetooth** est désigné dans le système par **hci0**. Il possède comme **adresse matérielle** « **B8:27:EB:10:1F:A8** », est actif et en marche (**UP RUNNING**). Cette commande provient du paquet **bluez**, mais il en existe d'autres, plus intéressantes fournies par le paquet **bluez-tools**. BlueZ est le nom du projet ayant en charge de fournir les fonctionnalités Bluetooth à GNU/Linux, que ce soit sous la forme de pilotes, de services ou d'outils, et ce aussi bien pour la Raspberry Pi que pour les distributions GNU/Linux sur PC.

bluez-tools nous fournit par exemple la commande **bt-adapter** nous permettant d'obtenir davantage d'informations :

```
$ bt-adapter -i
[hci0]
  Name: raspberrypi
  Address: B8:27:EB:10:1F:A8
  Alias: raspberrypi [rw]
  Class: 0x0
  Discoverable: 0 [rw]
  DiscoverableTimeout: 180 [rw]
  Discovering: 0
  Pairable: 1 [rw]
  PairableTimeout: 0 [rw]
  Powered: 1 [rw]
  UUIDs: [PnPInformation,
00001800-0000-1000-8000-00805f9b34fb,
    00001801-0000-1000-8000-00805f9b34fb, AVRemoteControl,
    AVRemoteControlTarget]
```

On retrouve ici non seulement l'adresse matérielle, mais également le nom affiché du périphérique. Les produits disposant d'une connectivité Bluetooth sont identifiés par leur adresse qui est unique et par leur nom, qui peut être tantôt changé et qui, dans GNU/Linux, correspond par défaut au nom du système (nom d'hôte). Ici, ce nom pourra être modifié par l'édition du fichier **/etc/bluetooth/main.conf** et un redémarrage du service avec **sudo service bluetooth restart**.

On voit également dans le résultat de la commande que notre adaptateur n'est pas visible de l'extérieur (**Discoverable: 0**), n'est pas en train de chercher des périphériques (**Discovering: 0**), mais est appairable (ou associable). La notion d'appairage (*pairing* en anglais) est fondamentale en Bluetooth, car il s'agit d'une sécurité permettant de limiter les accès aux périphériques tout en n'ayant pas besoin, à chaque fois, de montrer patte blanche. Une fois deux périphériques appairés, généralement après vérification d'un code ou d'un mot de passe, ils peuvent communiquer librement. Les périphériques Bluetooth Smart ou Bluetooth Low Energy fonctionnent de façon un peu différente à ce niveau selon leur type, mais nous resterons concentrés ici sur le Bluetooth Classic.

Il est possible de manipuler notre adaptateur Bluetooth avec la commande **bt-adapter** et en utilisant l'option **--set** pour changer sa configuration. De la même manière, le paquet **bluez-tools** offre également la commande **bt-device** permettant de gérer les périphériques (détection, association, collecte d'informations, etc.). Je trouve cependant plus facile d'utiliser un autre outil, fourni par le paquet **bluez** : **bluetoothctl**.

2. JOUER AVEC LES PÉRIPHÉRIQUES ET LES APPAIRER

bluetoothctl est un programme interactif. Une fois lancé, vous obtenez une nouvelle ligne de commandes vous permettant de saisir des instructions propres à la gestion du Bluetooth. Vous pourrez quitter cette interface en utilisant simplement la commande **exit** ou avec le raccourci Ctrl+d. Nous lancerons la commande ainsi :

```
$ bluetoothctl -a
[NEW] Controller B8:27:EB:10:1F:A8 raspberrypi [default]
Agent registered
[bluetooth]#
```

Notez que celle-ci n'a pas besoin d'être lancée avec **sudo**, l'utilisateur par défaut ayant déjà les droits adéquats et la commande ne contrôlant pas directement le matériel, mais plutôt le service Bluetooth associé. L'option **-a** nous permet de faire en sorte de déclarer automatiquement un agent. En effet, le fait que le Bluetooth s'utilise également en mode graphique, directement sur le bureau, nécessite un mécanisme particulier.

Lors d'une association par exemple, il est nécessaire que l'utilisateur saisisse un code PIN ou un mot de passe. Or ceci ne peut pas être proposé par un service travaillant en tâche de fond dans le système. Il est alors fait appel à un agent, présentant une fenêtre et attendant la saisie dont le contenu sera ensuite communiqué au service qui procèdera à l'association Bluetooth avec le périphérique.

Ici, via l'option **-a**, nous spécifions que nous n'avons pas besoin d'un agent externe, mais que la commande **bluetoothctl** elle-même servira d'interface et d'agent en même temps. Ainsi, lorsque nous demanderons une association nous pourrons saisir le code ou le mot de passe correspondant dans la foulée.

Une fois la commande **bluetoothctl** lancée, nous pouvons afficher des informations sur notre adaptateur avec **list** et **show**. Ceci nous présente peu ou prou la même chose que les commandes précédentes. Si plus d'un adaptateur est présent sur le système, il est également

Il existe des adaptateurs Bluetooth USB de toutes sortes. Cette petite collection présente en bas à droite un modèle récent intégrant Bluetooth Classic et Bluetoth Low Energy (BLE), à gauche un vieux modèle d'entrée de gamme et en rouge, un périphérique industriel de très bonne facture, mais ne supportant malheureusement pas le BLE.

possible de choisir celui à utiliser avec un **list** puis une commande **select** suivie de l'adresse matérielle de l'adaptateur.

La commande **devices** nous permet de lister les périphériques disponibles, mais s'il s'agit de votre première utilisation du Bluetooth avec votre Raspberry Pi, celle-ci ne retournera probablement rien du tout. Il nous faut en effet tout d'abord provoquer une recherche des périphériques Bluetooth qui nous entourent et donc demander à l'adaptateur de passer en mode découverte (*discovering* à *yes*) avec la commande **scan on**.

Si cette commande vous retourne un message « *Failed to start discovery: org.bluez.Error.NotReady* », assurez-vous via un **show** que l'adaptateur est bien en marche (« **Powered: yes** »). Si ce n'est pas le cas, pliez-vous simplement d'un **power on** :

```
[bluetooth]# power on
Changing power on succeeded
[CHG] Controller B8:27:EB:10:1F:A8 Powered: yes
```

avant d'utiliser **scan on** :

```
[bluetooth]# scan on
Discovery started
[CHG] Controller B8:27:EB:10:1F:A8 Discovering: yes
```

Au bout de quelques instants, le ou les périphériques qui peuvent être recherchés (*discove rable*) dans votre voisinage vont commencer à être listés :

```
[NEW] Device 00:19:5D:EE:A4:24 Hackable15
[NEW] Device 78:CA:39:BF:E4:DA iMac de yann
[NEW] Device 00:19:5D:EE:A4:1D HackableBT2
[NEW] Device 49:83:39:88:BE:21 49-83-39-88-BE-21
[CHG] Device 49:83:39:88:BE:21 RSSI: -91
[NEW] Device 78:E9:94:D7:DC:41 78-E9-94-D7-DC-41
[CHG] Device 78:E9:94:D7:DC:41 RSSI: -84
[CHG] Device 78:E9:94:D7:DC:41 RSSI: -95
[CHG] Device 78:E9:94:D7:DC:41 RSSI: -86
[CHG] Device 78:CA:39:BF:E4:DA RSSI: -82
[NEW] Device 43:07:CA:F8:FB:A8 43-07-CA-F8-FB-A8
[CHG] Device 43:07:CA:F8:FB:A8 RSSI: -99
[CHG] Device 43:07:CA:F8:FB:A8 RSSI: -87
[CHG] Device 43:07:CA:F8:FB:A8 RSSI: -97
[NEW] Device 79:58:75:F5:00:07 79-58-75-F5-00-07
```

Notez que la commande **scan** change l'état de l'adaptateur qui se met à la recherche de périphériques, mais que celle-ci n'est pas bloquante. Les messages apparaissent au fur et à mesure alors que vous gardez la main sur l'outil. Les messages concernent aussi bien les découvertes de périphériques que les changements les concernant, comme la variation de la puissance du signal (RSSI) ou encore le fait d'obtenir le nom associé à

l'adresse matérielle. Gardez simplement à l'esprit que ces messages ne sont pas le résultat de votre commande, mais juste une information sur ce qui se passe.

Après quelque temps, vous pourrez utiliser la commande **devices** pour lister vos découvertes alentour :

```
[bluetooth]# devices
Device 00:19:5D:EE:A4:24 Hackable15
Device 78:CA:39:BF:E4:DA iMac de yann
Device 00:19:5D:EE:A4:1D HackableBT2
Device 49:83:39:88:BE:21 49-83-39-88-BE-21
Device 78:E9:94:D7:DC:41 78-E9-94-D7-DC-41
```

Nous voyons ici que plusieurs périphériques ont été trouvés. Certains donnent leur nom d'autres pas. Certains sont à moi et d'autres à quelqu'un dans le voisinage. Celui qui nous intéresse ici est « Hackable15 », c'est le module Bluetooth utilisé dans le précédent numéro en compagnie d'une carte Arduino permettant de contrôler la couleur d'une led depuis un smartphone Android.

Nous pouvons obtenir des informations sur ce périphérique :

```
[bluetooth]# info 00:19:5D:EE:A4:24
Device 00:19:5D:EE:A4:24
        Name: Hackable15
        Alias: Hackable15
        Class: 0x0c0800
        Paired: no
        Trusted: no
        Blocked: no
        Connected: no
        LegacyPairing: yes
```

Nous voyons ici son nom et adresse, mais également le fait qu'il ne soit pas associé, ni de confiance, ni bloqué, ni connecté. Nous apprenons également que ce périphérique supporte le *legacy pairing*, correspondant à une méthode d'association propre au Bluetooth 2.0 (normal puisqu'il ne s'agit pas d'un périphérique Bluetooth Low Energy). Nous pouvons éventuellement tenter une connexion, qui est vouée à l'échec par définition sans association, afin d'obtenir tantôt plus d'informations et en particulier le ou les UUID permettant à l'outil de connaître les services disponibles et donc quelques caractéristiques supplémentaires. Exemple :

```
[bluetooth]# connect 00:19:5D:EE:A4:1D
Attempting to connect to 00:19:5D:EE:A4:1D
[CHG] Device 00:19:5D:EE:A4:1D Connected: yes
[CHG] Device 00:19:5D:EE:A4:1D UUIDs:
        00001101-0000-1000-8000-00805f9b34fb
Failed to connect: org.bluez.Error.NotAvailable
[CHG] Device 00:19:5D:EE:A4:1D Connected: no
[bluetooth]# info 00:19:5D:EE:A4:1D
Device 00:19:5D:EE:A4:1D
        Name: Barre72
        Alias: Barre72
        Class: 0x0c0800
```

```
        Paired: no
        Trusted: no
        Blocked: no
        Connected: no
        LegacyPairing: yes
        UUID: Serial Port
(00001101-0000-1000-8000-00805f9b34fb)
```

Cette dernière ligne nous indique que l'UUID rapporté par le périphérique décrit un port série (*Serial Port*). Parfois, en particulier avec des PC ou des Mac ayant la connectivité Bluetooth activée, il n'est pas même nécessaire de tenter une connexion et nous obtenons une liste bien plus complète de services :

```
UUID: Audio Source              (0000110a-0000-1000-8000-00805f9b34fb)
UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
UUID: Handsfree Audio Gateway   (0000111f-0000-1000-8000-00805f9b34fb)
UUID: Serial Port               (00001101-0000-1000-8000-00805f9b34fb)
UUID: Service Discovery Serve.. (00001000-0000-1000-8000-00805f9b34fb)
UUID: Headset AG                (00001112-0000-1000-8000-00805f9b34fb)
UUID: GN                        (00001117-0000-1000-8000-00805f9b34fb)
```

Mais revenons à notre périphérique « Hackable15 »... La phase suivante consiste à associer ou appairer le périphérique en utilisant la commande **pair** suivie de l'adresse matérielle correspondante :

```
[bluetooth]# pair 00:19:5D:EE:A4:24
Attempting to pair with 00:19:5D:EE:A4:24
[CHG] Device 00:19:5D:EE:A4:24 Connected: yes
Request PIN code
[agent] Enter PIN code: 424242
[CHG] Device 00:19:5D:EE:A4:24 UUIDs:
        00001101-0000-1000-8000-00805f9b34fb
[CHG] Device 00:19:5D:EE:A4:24 Paired: yes
Pairing successful
[CHG] Device 00:19:5D:EE:A4:24 Connected: no
```

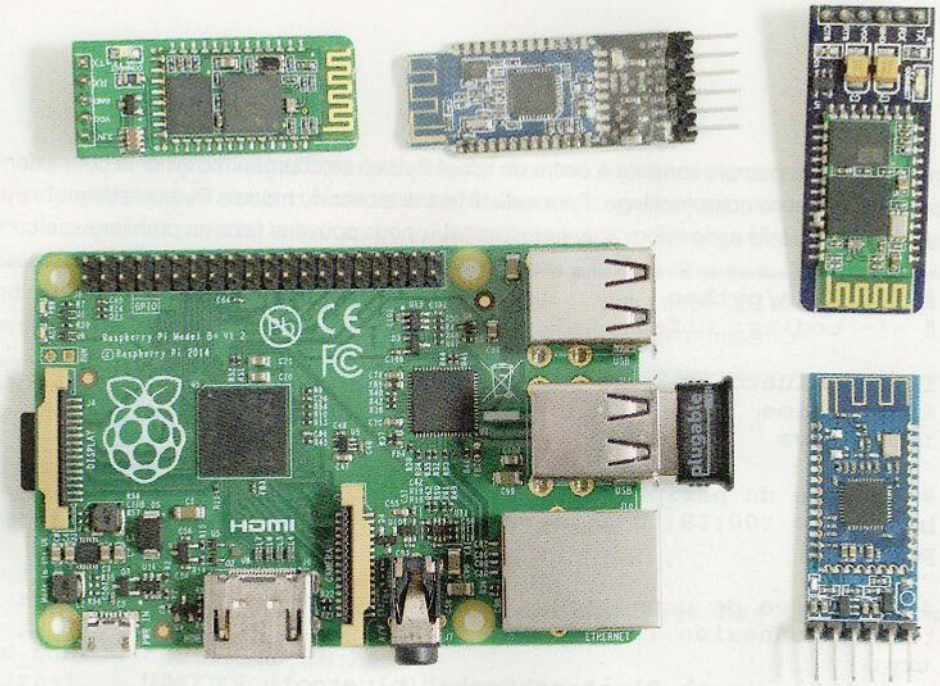
Ce mélange de messages et de résultats doit être vu d'une part en :

```
[CHG] Device 00:19:5D:EE:A4:24 Connected: yes
[CHG] Device 00:19:5D:EE:A4:24 UUIDs:
        00001101-0000-1000-8000-00805f9b34fb
[CHG] Device 00:19:5D:EE:A4:24 Paired: yes
[CHG] Device 00:19:5D:EE:A4:24 Connected: no
```

et d'autre part en :

```
[bluetooth]# pair 00:19:5D:EE:A4:24
Attempting to pair with 00:19:5D:EE:A4:24

Request PIN code
[agent] Enter PIN code: 424242

Pairing successful
```

Suite à la commande, l'outil a besoin du code permettant l'association. Celui-ci nous est alors demandé, puisque **bluetoothctl** lancé avec l'option **-a** embarque un agent. Nous entrons



La Raspberry Pi Model B+ équipée d'un adaptateur USB disposera d'une connectivité Bluetooth identique à sa grande sœur la Pi 3. Disposés autour de la Pi, quelques modules Bluetooth de différentes origines utilisables très facilement et pour offrir une fonctionnalité Bluetooth/série à n'importe quel projet.

424242 correspondant au code que nous avions configuré au moment de paramétrer le module dans le numéro précédent. Suite à cela, la tentative d'association a lieu et réussit avec le message « Pairing successful ».

Notre périphérique est maintenant associé, ce qui signifie qu'un lien de confiance existe entre notre système et lui, et qu'il n'est plus nécessaire de procéder à d'autres authentifications via un code. Notre périphérique apparaît maintenant aussi bien dans le résultat de la commande **devices** que dans celui de la commande permettant de lister les périphériques associés :

```
[bluetooth]# paired-devices
Device 00:19:5D:EE:A4:24 Hackable15
```

Bien entendu l'association, une fois en place, perdure après avoir quitté **bluetoothctl** et même entre deux redémarrages. Attention cependant, celle-ci est liée à l'adaptateur et sa propre adresse matérielle. En d'autres termes, si vous utilisez un adaptateur USB et le replacez par un autre (d'un modèle strictement similaire ou non), les associations ne seront plus en place et il faudra recommencer.

Pour « désassocier » un périphérique, le cas échéant, il vous suffira d'utiliser la commande **remove** suivie de l'adresse du périphérique. Notez à ce propos que pour réassocier ce même périphérique, il faudra au préalable qu'il soit à nouveau découvert (**scan on**) et présent dans la liste des périphériques (**devices**) dont les entrées expirent quelque temps après la découverte si la recherche est désactivée (**scan off**).

3. COMMUNIQUER EN BLUETOOTH AVEC PYTHON

À présent que notre montage Arduino est associé, nous pouvons communiquer avec lui. Il existe plusieurs techniques comme par exemple le fait d'utiliser la commande **rfcomm** du paquet **bluez** pour initier la connexion et ajouter un pseudo port série la représentant. On doit alors ensuite utiliser un outil de communication série ou un programme « maison » pour utiliser ce port et échanger des données avec le périphérique Bluetooth distant.

Une solution plus simple consiste à écrire un script Python se chargeant d'initier la connexion puis de communiquer avec notre montage. Pour cela, il faut disposer du module Python adéquat : **python-bluez**. Une fois installé avec **sudo apt-get install**, nous pouvons faire un premier essai :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import bluetooth
import time
import sys

# adresse du périphérique Bluetooth
bd_addr = "00:19:5D:EE:A4:24"
port=1

# tentative de connexion
print "Connexion..."
try:
  sock=bluetooth.BluetoothSocket(bluetooth.RFCOMM)
  sock.connect((bd_addr, port))
except:
  print "Erreur lors de la connexion !"
  sys.exit(1)

# Ca marche, sock représente la connexion désormais
print "Connecté."
time.sleep(2)

# Tentative de déconnexion
print "Déconnexion..."try:
  sock.close()
except:
  print "Erreur lors de la déconnexion !"

print "Déconnecté."
```

Ce script ne fait pas grand-chose, mais nous permet de vérifier le fonctionnement de la connexion. On utilise le module Python **bluetooth** en compagnie de l'adresse matérielle du périphérique auquel se connecter (et qui est désormais appairé). On crée alors un objet **sock** retourné par **BluetoothSocket()**. La syntaxe **try/except** nous permet de détecter un problème et d'y réagir facilement.

La liaison effectuée est une liaison série reposant sur le protocole Bluetooth RFCOMM, typique des modules qu'on peut utiliser avec des montages Arduino. Une fois la communication établie, notre script marque une pause de deux secondes puis invoque la méthode **close** pour mettre un terme à la liaison. Là encore, le **try/except** est bien pratique.

Le script une fois lancé nous affiche normalement simplement :

```
Connexion...
Connecté.
Déconnexion...
Déconnecté.
```

Ce script forme un squelette qui peut nous servir de base pour tous nos projets. Dans le cas qui nous intéresse ici, nous voulons envoyer un message au montage permettant de choisir la couleur que doit prendre la led qui s'y trouve connectée. Ce message prend la forme de quelques octets : 0, 0, la valeur de rouge, de vert, de bleu et le caractère « ; ». Pour simplifier l'utilisation, notre outil va prendre en argument une valeur de teinte entre 0 et 359, puis transformer celle-ci en rouge, vert et bleu, avec une saturation et une valeur maximum.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import bluetooth
import sys
import colorsys

def hsv2rgb(h,s,v):
  return tuple(int(i * 255) for i in colorsys.hsv_to_rgb(h,s,v))

bd_addr = "00:19:5D:EE:A4:24"
port=1
msg = bytearray("\x00\x00\x00\x00\x00;")

if len(sys.argv) != 2:
  print "Donnez-moi une teinte (0-359) !"
  sys.exit(1)

couleur = int(sys.argv[1])

if not (0 <= couleur <= 359):
  print "Mauvaise couleur/teinte !"
  print "Entre 0 et 359 svp."
  sys.exit(1)

print "Envoi..."

try:
  sock=bluetooth.BluetoothSocket(bluetooth.RFCOMM)
  sock.connect((bd_addr, port))
except:
  print "Erreur lors de la connexion !"
  sys.exit(1)

msg[2],msg[3],msg[4] = hsv2rgb(couleur/360.0,1,1)
sock.send(str(msg))

try:
  sock.close()
except:
  print "Erreur lors de la déconnexion !"
```

Pour la conversion de couleur, nous utilisons le module **colorsys** qui est livré en standard avec Python et ajoutons une fonction simple, **hsv2rgb**, qui nous retourne les valeurs au bon format (entre 0 et 255 et non entre 0,0 et 1,0).

Nous préparons notre message sur la forme d'un **bytearray**, un tableau d'octets puis, après avoir vérifié que la valeur passée en argument correspond bien à nos attentes, nous le

modifions avec les valeurs de couleur calculées. Le tout est enfin envoyé avec la méthode `send()` de notre objet `sock` sous la forme d'une chaîne de caractères à destination du montage Arduino.

Pour utiliser ce script et envoyer une couleur au montage, il nous suffit maintenant de l'appeler en lui passant un nombre entre 0 et 359. Immédiatement, la led connectée prend la couleur demandée. Mission accomplie !

4. POUR FINIR... OU PRESQUE

Étant d'un naturel à préférer la ligne de commandes aux interfaces graphiques, j'ai une tendance à tout naturellement écrire des scripts et des programmes pour ce type d'interface. Python permet cependant de développer rapidement des outils graphiques (Qt, Tk, Gtk+, etc.) qui peuvent offrir plus de souplesse avec, par exemple ici, un sélecteur de couleur similaire à l'application Android que nous avons vu dans le précédent numéro.

Mais hors du cadre de la liaison avec notre montage, une telle communication Bluetooth entre Arduino et Raspberry Pi pourra satisfaire bien des besoins. On imagine ainsi aisément quelques exemples :

- sonde de température ou autre ;
- afficheur LCD distant ;
- imprimante thermique déportée ;
- télécommande Bluetooth de moteurs ou servo ;
- contrôle de prises de courant ou de luminaires pour une solution domotique ;
- etc.

Il est également possible de voir en la Raspberry Pi non pas un maître, mais un esclave, attendant des connexions de périphériques. Ceci demande un peu plus de configuration au niveau système et nous reviendrons peut-être sur le sujet dans un prochain numéro. Dans ce genre de scénario, il devient alors possible d'obtenir une ligne de commandes de la part de la Pi depuis un smartphone via une application faisant office d'émulateur de terminal.

Le Bluetooth est un terrain de jeu très intéressant qui offre une vaste compatibilité. Le Wifi aussi, me direz-vous, mais à la différence du Wifi, le Bluetooth fonctionne avec des équipements plus anciens ou peu puissants, ne pouvant supporter le Wifi. De plus, je trouve que le Bluetooth, du fait de prendre ici la forme d'une liaison série, est bien plus simple à mettre en œuvre. Mais à chaque projet son cahier des charges et ses besoins. La clé du succès c'est avant tout d'utiliser les bons outils pour les bonnes tâches... DB

Les adaptateurs USB Bluetooth industriels se déclinent tantôt en des modèles assez originaux. Celui-ci, le Parani SD1000U de SENA, n'est pas à proprement parler un périphérique Bluetooth, mais apparaît comme un port série fournissant une liaison transparente sans avoir à utiliser de pilote Bluetooth spécifique.