N°11 | MARS - AVRIL 2016

# 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

ARDUINO

## Créez un afficheur numérique multifonction géant avec des bandes de leds

p. 18



NOUVEAUTÉ

## Faites fonctionner votre Raspberry Pi Zero comme un périphérique USB 2.0

p. 04



RASPBIAN

## Maîtrisez l'installation et la désinstallation d'applications sur Raspberry Pi

p. 70

IMPRESSION 3D

## Transformez votre Raspberry Pi en serveur d'impression avec OctoPrint

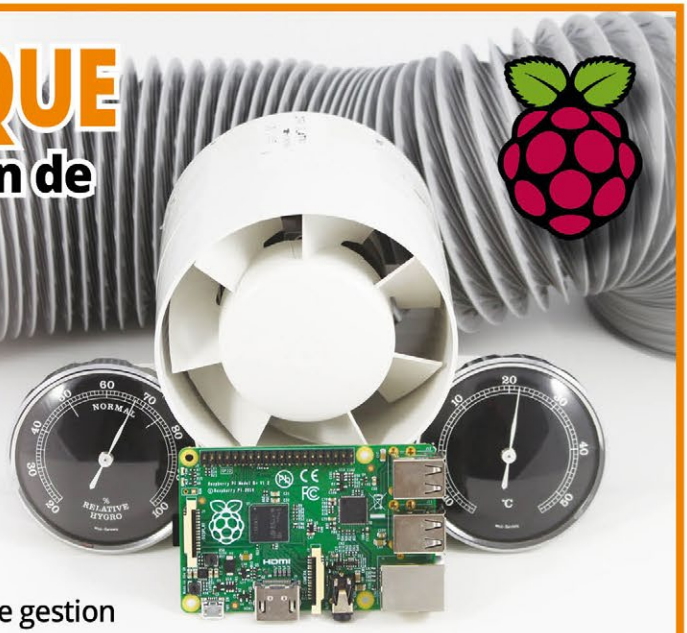
p. 82

## DOMOTIQUE

### Pilotez l'aération de votre habitat !

p. 50

- Température et humidité : pourquoi automatiser la ventilation ?
- Assemblez et configurez votre installation
- Programmez la gestion de la VMC
- Ajoutez une interface web de gestion



CAPTEURS

## Découvrez comment utiliser de simples leds comme détecteurs de lumière

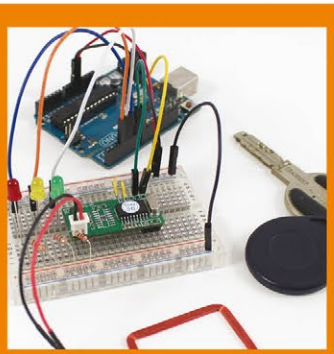
p. 32



ARDUINO

## Construisez votre serrure « sécurisée » à base de porte-clés RFID 125 Khz

p. 38



L 19338 - 11 - F: 7,90 € - RD

# INNOROBO PARIS FRANCE

## 24-26 MAI 2016

co-produit

INSIDE 3D PRINTING™

CONFERENCE & EXPO

SMART CITIES

MEDICAL & HEALTH

SMART HOMES

FACTORIES OF THE FUTURE

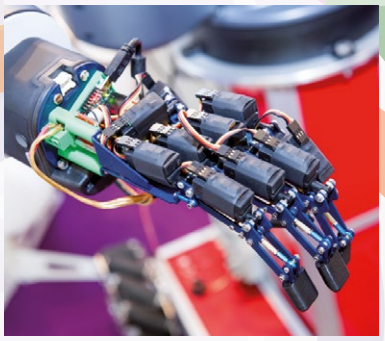
SERVICE TO INDUSTRIES

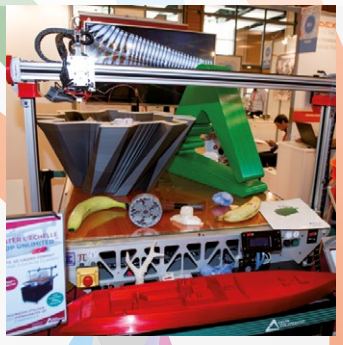
FIELD ROBOTICS

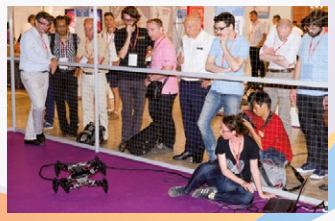
TECHNOLOGIES & FORESIGHT

Passionnés d'électronique, de mécanique et de nouvelles technologies ?

De l'impression 3D à l'intelligence artificielle, les technologies emblématiques font leur show, **ne les manquez pas !**











Ouverture aux professionels les mardi et jeudi

**Ouverture au grand public et aux étudiants le mercredi toute la journée**

YouTube theinnorobo

Pour la première fois INNOROBO est à PARIS. Toutes les info sur :

**www.innorobo.com**

risingmedia

PRODUIT PAR INNOECHO

# ÉDITO



Mars, enfin !

Non, pas la planète, celle-ci est malheureusement encore hors de portée de nos bidouillages et autres projets qui n'ont, pour l'instant, aucune prétention spatiale. Je parle naturellement du mois et du changement de saison qui l'accompagne : le printemps !

Qui dit changement de saison dit implicitement changement d'habitudes en termes de gestion de son habitat. En dehors du traditionnel nettoyage de printemps, consistant pour ma part à simplement déplacer les choses que je ne jette pas (on sait jamais, ça peut servir), c'est l'occasion de prévoir des travaux et des projets en prévision de l'été, période plus chaude, voire caniculaire.

Voilà précisément ce qu'ont fait Axelle et Ludovic Apvrille en se penchant sur la problématique de la gestion de l'aération de leur lieu de vie. L'objectif est simple : s'assurer le maintien d'un environnement sain, à la température et à l'hygrométrie contrôlées et, par la même occasion, réaliser des économies en profitant des conditions climatiques extérieures, fonctions des moments de la journée. Ils partagent donc ici avec nous leur installation basée sur le contrôle d'une VMC tout à fait standard par Raspberry Pi et un ensemble de configurations originales.

Bien entendu, comme à son habitude, le magazine ne se contentera pas de ce seul sujet et explorera, encore une fois, bien des domaines de bien des façons. Mais je vous laisse découvrir cela par vous-même sans plus attendre, retournant pour ma part à la construction des projets qui peupleront les prochains numéros (quand ils se décideront à fonctionner correctement).

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 : 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


## ACTUALIÉS

**04**
Raspberry Pi Zero : un port USB pas comme les autres

## ARDU'N'CO

**18**
Créez un (dé)compteur géant multi-usage et stressant

**32**
Utilisez des leds comme capteurs de lumière

**38**
Créez une serrure à carte/tag RFID

## EN COUVERTURE

**50**
Ventilation contrôlée par des framboises

## EMBARQUÉ & INFORMATIQUE

**70**
Gérez efficacement vos installations/désinstallations de logiciels dans Raspbian

**76**
Gérez vos installations/désinstallations de logiciels dans Raspbian : les commandes avancées

**82**
OctoPrint : transformez votre Raspberry Pi en serveur d'impression 3D

## ABONNEMENT

**17**
Offres spéciales professionnels

**59/60**
Abonnements tous supports

# RASPBERRY PI ZERO : UN PORT USB PAS COMME LES AUTRES

*Denis Bodor*

**L'arrivée récente d'un nouveau modèle de Raspberry Pi, la Zero, est une opportunité très intéressante en termes de coût et d'encombrement, mais cette minuscule carte possède un atout qui passe presque inaperçu si l'on ne suit pas activement les développements autour du noyau Linux : le port micro USB permettant la connexion de périphériques peut également servir dans le sens inverse ! En effet, avec un peu de ténacité, il est possible de faire passer la Raspberry Pi pour un périphérique USB.**



*Les câbles USB OTG se présentent généralement équipés d'un connecteur type A femelle d'un côté et d'un micro USB mâle de l'autre. Il faut cependant faire attention à ne pas confondre ce type de câbles avec de simples adaptateurs ou changeurs de genre. Un câble OTG possède en interne une connexion entre la broche 4 et la masse côté micro USB.*

Dans le monde de l'USB, on distingue généralement deux types d'acteurs : les hôtes et les périphériques. Un hôte USB est un ordinateur qui voit les connexions USB comme un bus sur lequel il trouve des périphériques (clavier, clé, disque dur, lecteur DVD, souris, etc.). Lors d'une connexion, l'hôte est prévenu de l'événement et se charge de l'analyse du périphérique, on dit qu'il les « énumère ». Les périphériques USB en revanche sont des esclaves sur le bus et n'agissent que lorsqu'ils en reçoivent l'ordre et qu'ils sont interrogés. Ils embarquent des informations décrivant leur identité, leurs fonctions et la façon de les utiliser, ce qui permet à l'hôte de choisir un pilote de périphérique en conséquence. Le plus souvent, aucun pilote spécifique n'est nécessaire, car le périphérique se conforme à un fonctionnement commun à une classe de périphérique (un clavier USB est un périphérique HID standard, quel que soit son modèle par exemple).

## 1. USB OTG ?

En plus de ces deux rôles s'ajoute l'USB OTG pour *On-The-Go*. Il s'agit d'une extension de la norme USB 2.0 qui permet aux périphériques de dialoguer entre eux sans avoir à passer par l'hôte. Par voie de conséquence, un périphérique USB OTG pourra également choisir d'être un hôte ou un périphérique. C'est un type de fonctionnement qui se généralise avec les tablettes Android par exemple. Ainsi lorsqu'une tablette est connectée à un PC, elle prendra le rôle de périphérique et apparaîtra comme un appareil compatible MTP (*Media Transfer Protocol*) permettant à l'utilisateur d'y placer ou d'y lire des images, photos, vidéos, sons, etc. Il est également possible de configurer certaines tablettes (ou smartphones) pour apparaître comme un simple disque USB ou encore comme un appareil Android pour le développement d'applications (ADB).

Inversement, si l'on branche sur une telle tablette une clé USB par exemple, celle-ci se comportera non plus comme un périphérique, mais comme un hôte et permettra à l'utilisateur d'accéder au contenu de la clé. En fonction des tablettes, il sera également possible de connecter d'autres périphériques comme un clavier, un récepteur RTL SDR et même une carte Arduino. La tablette peut donc changer de rôle en fonction des besoins en utilisant exactement le même port USB... ou presque.

Vous l'avez peut-être remarqué, mais un connecteur micro USB ne dispose pas de 4 broches, mais de 5. On retrouve ainsi la masse, l'alimentation (Vcc +5V) et les signaux D+ et D- comme sur un connecteur USB type A ou B, mais également une broche supplémentaire nommée « sense ». Celle-ci est utilisée pour que le système derrière le connecteur USB (la tablette par exemple) puisse savoir s'il doit fonctionner comme un hôte ou comme un périphérique. Si la broche « sense »

est à la masse, c'est qu'un adaptateur USB type A femelle est connecté et donc qu'un périphérique y sera branché, pas un hôte. Dans le cas contraire, il s'agit d'une connexion avec un hôte et là il faut agir comme un périphérique. Attention cependant, un adaptateur micro USB mâle vers USB Type A femelle n'est pas forcément un adaptateur OTG (avec la broche « sense » à la masse), et sans vérifier physiquement cette connexion, il n'est pas possible d'être sûr à 100%, il peut s'agir d'un simple adaptateur d'un type vers un autre. N'achetez pas d'adaptateurs sur la simple base qu'ils possèdent les bons connecteurs, pour être OTG, il FAUT que « sense » (broche 4) soit connecté à la masse (broche 5). Dans le cas contraire, la tablette ou le smartphone ne passera pas en mode USB hôte et vous n'accéderez pas au périphérique.

Pourquoi vous expliquer tout cela ? Simplement parce que la Raspberry Pi Zero, contrairement aux modèles B, B+, A, A+ et 2, dispose d'un connecteur micro USB femelle et non d'un type A femelle. Les modèles B, B+ et 2 intègrent un hub USB, ce qui en fait définitivement des hôtes. Les modèles A et A+ en revanche disposent d'un port USB directement connecté au SoC Broadcom (processeur), mais la fameuse broche « sense » est matériellement reliée à la masse directement sur le circuit imprimé de la carte. La Raspberry Pi Zero, en revanche, est différente : la broche « sense » du connecteur est reliée au SoC Broadcom qui peut donc savoir si son port USB doit fonctionner en tant qu'hôte ou que périphérique. Le port est donc un vrai port USB OTG, comme sur une tablette !

Oui, vous avez bien lu. Le circuit intégré au cœur des Raspberry Pi, de toutes les Raspberry Pi, dispose d'un contrôleur USB OTG. Il peut, en principe, fonctionner dans l'un ou l'autre rôle et donc faire office de périphérique USB (il se comporte alors comme un UDC pour *USB Device Controller*), comme bien d'autres nano-ordinateurs ou systèmes embarqués. Mais seul le modèle Zero met réellement en œuvre cette fonctionnalité plutôt que de considérer, de fait, que le connecteur USB ne s'utilise que pour y brancher des périphériques.

## 2. POURQUOI UTILISER UN NANO-ORDINATEUR COMME PÉRIPHÉRIQUE USB ?

« *À quoi bon utiliser ma Raspberry Pi ainsi ?* » me direz-vous. C'est vrai que cela peut paraître étrange à première vue, puisqu'une Pi est un ordinateur. Mais vous devez voir les choses sous un autre angle, plus... atypique. Un nano-ordinateur ou un système embarqué disposant de la fonctionnalité USB OTG peut littéralement devenir n'importe quel périphérique USB. En dehors de l'aspect ludique qui, avec plus ou moins de compétences en programmation, vous permet de faire passer votre framboise pour peu ou prou n'importe quoi, le plus simple est encore de vous donner quelques exemples :

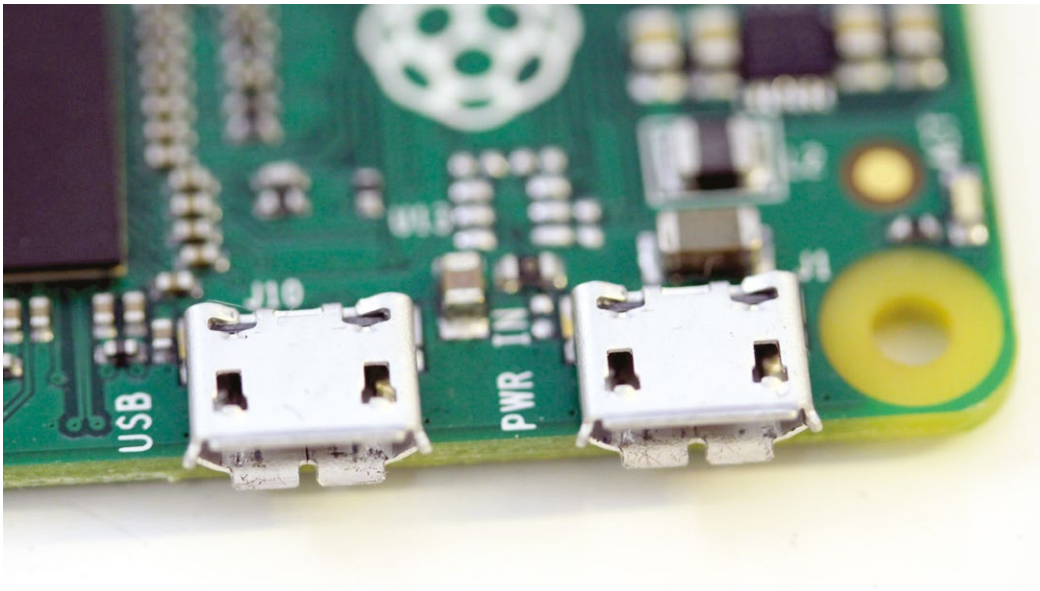
- En utilisant le port USB comme un périphérique série, il n'est plus nécessaire d'avoir recours à un adaptateur USB. Une connexion PC/Pi fera apparaître un nouveau port série côté PC rendant ainsi possible l'accès à la Raspberry Pi.
- En faisant passer la Pi pour un disque USB, il sera possible de transférer des données depuis un PC ou un Mac comme s'il s'agissait d'une clé USB de stockage.
- En configurant le port USB pour devenir un adaptateur réseau (Ethernet), il devient possible de créer un réseau entre le PC et la Pi et donc de procéder à des transferts de données, d'ouvrir des sessions distantes (SSH), etc. La Pi connectée ainsi au PC apparaîtra comme une nouvelle « carte réseau ».

- En faisant passer la Pi pour un périphérique MIDI sur USB, votre PC ou votre Mac verra une connexion vers un séquenceur MIDI et votre framboise deviendra un instrument de musique.
- Plus exotique, en faisant passer votre Raspberry Pi pour un périphérique USB vidéo, l'hôte ainsi connecté verra une source audio/vidéo et vous pourrez émuler un périphérique vidéo comme une webcam par exemple.

Mais le plus intéressant dans cette poignée d'exemples est, sans le moindre doute, le fait que tout ceci soit d'ores et déjà disponible dans le noyau Linux, et ce depuis longtemps (noyau version 2.4/2.6). Cela fait partie des fonctionnalités offertes dès lors que le pilote pour le contrôleur USB dispose de certaines caractéristiques et est en mesure de faire office d'UDC. Si ce dernier est en mesure de proposer une utilisation en tant que périphérique et non seulement d'hôte, toute une infrastructure est alors utilisable : c'est le support USB Gadget du noyau. Sur cette infrastructure et grâce à l'interface de programmation dédiée (API USB Gadget) repose une série de pilotes ou composants (les *Gadget Drivers*) fournissant, entre autres, les fonctionnalités énumérées précédemment.

Pour profiter de tout cela avec notre Raspberry Pi Zero, nous devrons donc :

- recompiler un noyau prenant en charge le bon pilote pour le support USB (DWC2) ;
- et activer le support USB Gadget ainsi que les pilotes associés sous forme de modules (leur compilation n'est pas déclenchée par défaut) ;
- générer un paquet pour ce nouveau noyau ;
- l'installer en configurant correctement le matériel (avec le *Device Tree* cf. *Hackable n°9*) ;
- faire quelques essais à la fois en mode hôte et en mode périphérique pour nous assurer que le port bascule de l'un à l'autre rôle correctement.




*Contrairement aux autres modèles de Raspberry Pi, le Zero propose un connecteur micro USB aussi bien pour l'alimentation que pour le connecteur de périphériques USB. Bien que la fonctionnalité USB OTG ne soit pas encore présente dans le noyau officiel, nul doute que cette éventualité a été planifiée lors de la conception de la carte.*

## 3. AVERTISSEMENT !

Je tiens à préciser ici que les explications qui vont suivre, bien que décrites le plus simplement et pédagogiquement possible, restent très techniques et nécessitent un minimum de connaissances de base à la fois sur le fonctionnement d'un système GNU/Linux et sur la façon de procéder à certaines manipulations. Il sera par exemple question de recompiler un noyau pour la Raspberry Pi, chose que j'ai exposée plus en détail (de façon préventive) dans le numéro 10. Tout ce qui va suivre ne risque en aucune manière d'endommager physiquement votre Pi Zero (sauf si vous branchez n'importe quoi n'importe comment). Le système qui se trouve sur la carte micro SD en revanche pourrait se retrouver inutilisable en cas de bêtise ou de bug, et une réinstallation plus ou moins complète sera alors nécessaire. Si vous souhaitez réduire les risques au minimum, utilisez une nouvelle carte micro SD avec un système Raspbian fraîchement installé et non le système que vous avez méticuleusement et patiemment personnalisé depuis des semaines ou des mois.

Autre point important, il est nécessaire de bien comprendre que les éléments utilisés et en particulier le noyau Linux en œuvre n'est pas considéré comme stable. À l'heure où cet article est rédigé, la version 4.4 (branche rpi-4.4.y) est celle de développement. Ainsi, non seulement cette version pourrait contenir des bugs encore non corrigés mais, en plus, une partie des explications pourrait ne pas fonctionner. La branche de développement doit être vue comme un bac à sable ou un brouillon d'une future version dans laquelle les développeurs ajoutent des fonctionnalités.

Nous nous aventurons ici sur un territoire changeant et instable afin de profiter des prémices de ce qui sera, sans doute, une fonctionnalité classique intégrée aux prochaines versions du système Raspbian pour la Raspberry Pi Zero. Tout comme le pilote pour écran LCD SPI (fbtft) a d'abord commencé sa carrière comme une modification « hasardeuse » de l'existant avant d'être intégré de façon stable, le support USB OTG en est actuellement aux premières étapes vers une prise en charge fiable. Les explications qui vont suivre découlent d'une discussion sur GitHub initiée le 3 décembre dernier et ayant conduit à une proposition d'intégration (*Pull Request*) le 26 du même mois par Noralf Trønnes (alias *Notro*. Oui, le même qui a développé le support pour les écrans LCD SPI). Enfin, pour enfoncer le clou et bien vous faire comprendre la situation, notez qu'un modérateur du forum officiel Raspberry Pi a été très explicite sur le sujet de l'USB OTG le 26/11 dernier : « *The USB driver hasn't been tested in gadget mode by us for about 3 years [...] Some hacking would be needed to make this work* » (« le pilote USB n'a pas été testé par nos soins en mode gadget depuis environ 3 ans [...] Un peu de hacking pourrait être nécessaire pour que ça fonctionne »).

## 4. PRÉPARATION ET COMPILATION

Pour profiter de cette fonctionnalité intéressante de la Raspberry Pi Zero, nous devons donc configurer et compiler un nouveau noyau intégrant les bons pilotes et les bons modules. L'utilisation « classique » des Pi étant d'être un nano-ordinateur, tout ce dont nous avons besoin n'est simplement pas activé par défaut.

Tenter de compiler un noyau directement sur une Pi n'est pas très intelligent, c'est une procédure gourmande en mémoire et en ressources processeur. On fait donc cela sur un PC sous Ubuntu ou Debian, mais en produisant un noyau à destination du processeur de la Raspberry Pi : c'est une compilation croisée. Pour ce faire, nous devons disposer d'une chaîne de compilation croisée, installée par exemple avec :

```
$ sudo dpkg --add-architecture armhf
$ sudo apt-get update
$ sudo apt-get install gcc-arm-linux-gnueabihf
```

Les sources du noyau adaptées pour les Raspberry Pi sont disponibles sur GitHub (**https://github.com/raspberrypi/linux**). C'est donc là que nous devrons les récupérer. Pour ce faire, le plus simple est d'utiliser la commande **git** du paquet du même nom :

```
$ mkdir endroit/
$ cd endroit/
$ git clone https://github.com/raspberrypi/linux.git
```

Les tests que nous avons faits ainsi que ceux des différents hackers ayant travaillé sur le sujet portent sur une version particulière du noyau contenue dans la branche « rpi-4.4.y ». La notion de branches permet de faciliter les développements sur un projet. Ceci permet d'explorer plusieurs évolutions possibles du code pour, à la fin, réunir (fusionner) les modifications et obtenir la prochaine version stable. La branche par défaut est actuellement celle nommée « rpi-4.1.y » qui n'est pas celle qui nous intéresse.

On commence donc par lister les branches disponibles à distance (*remotes*) :

```
% git branch -r
  origin/HEAD -> origin/rpi-4.1.y
  origin/linux_stable
  origin/master
  origin/rpi-3.10.y
[...]
  origin/rpi-4.4.y
  origin/rpi-4.4.y_irq
  origin/rpi-patches
```

Nous avons là notre **rpi-4.4.y**, qui est en avance sur le développement actuel (*HEAD*). Ce n'est pas grave, il nous suffit de créer une branche locale calquée sur la branche distante correspondante (on dit que la branche locale « suit » la branche distante) :

```
% git branch rpi-4.4.y origin/rpi-4.4.y
La branche rpi-4.4.y est paramétrée pour suivre
la branche distante rpi-4.4.y depuis origin.
```

On regarde ce qu'on a à présent localement :

```
% git branch
* rpi-4.1.y
  rpi-4.4.y
```

Super, elle est là ! Il nous suffit alors de basculer dessus :

```
% git checkout rpi-4.4.y
Checking out files: 100% (21817/21817), done.
Basculement sur la branche 'rpi-4.4.y'
Votre branche est à jour avec 'origin/rpi-4.4.y'.
```

On vérifie :

```
% git branch
  rpi-4.1.y
* rpi-4.4.y
```

Nous y sommes, les fichiers présents dans le répertoire correspondent maintenant à la version 4.4.y du noyau Linux pour Raspberry Pi. Cette utilisation de la Pi nécessite une configuration particulière afin de pouvoir profiter des fonctionnalités que nous « débloquons ». Il nous faut donc revoir la configuration du noyau. Plusieurs solutions sont envisageables, à commencer par récupérer le fichier de configuration depuis le dépôt GitHub du magazine. Mais on peut également préconfigurer les sources avec la configuration par défaut pour Raspberry Pi :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcmrpi_defconfig
```

puis la modifier avec :

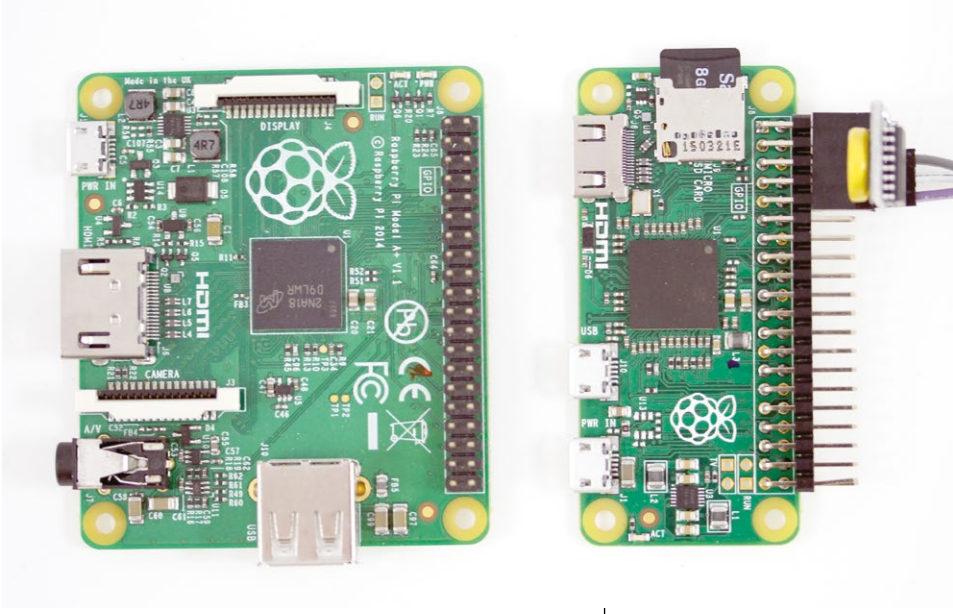
```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

ou

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- xconfig
```

Là, dans l'interface qui se présente vous devez apporter les modifications suivantes :

- **USB_DWC2** en module, menu ***Device Drivers > USB support > DesignWare USB2 DRD Core Support*** : c'est le pilote capable de prendre automatiquement en charge l'USB OTG sur le SoC Broadcom de la Pi (par défaut, c'est le pilote **dwc_otg** qui est utilisé et non **dwc2**) ;
- **IIO_BUFFER_CB** en module (et non « y ») : ***Device Drivers***, ***Industrial I/O support***, ***Enable buffer support within IIO***, ***IIO callback buffer used for push in-kernel interfaces*** : ceci n'est pas directement lié au support USB OTG, mais découle d'une autre dépendance ;
- **USB_GADGET** en module, menu ***Device Drivers***, ***USB support***, ***USB Gadget Support*** : ceci a pour effet d'activer le support USB Gadget et de faire apparaître un sous-menu supplémentaire, ***USB Gadget Drivers***, dans lequel on peut choisir les fonctionnalités à activer :
  - → **USB_ZERO** en module : ***Gadget Zero*** est une sorte de relais très pratique pour la mise au point de périphériques, de pilotes et d'outils USB ;
  - → **USB_AUDIO** en module : ***Audio Gadget*** permet de créer un périphérique de classe audio standard, comme une carte son USB ;
  - → **USB_ETH** en module : ***Ethernet Gadget (with CDC Ethernet support)*** active le support pour être une interface réseau filaire Ethernet CDC standard (pour ***Communication Device Class***) ;
  - → **USB_GADGETFS** en module : ***Gadget Filesystem*** est également utile pour les programmeurs en permettant d'écrire non pas un nouveau pilote, mais une application qui simule un périphérique USB ;
  - → **USB_MASS_STORAGE** en module : ***Mass Storage Gadget*** est, comme son nom l'indique, le pilote pour émuler un support de stockage comme une clé ou un disque dur USB ;



*La Raspberry Pi modèle A+ face à la nouvelle venue aussi mignonne que petite. Bien entendu, pour arriver à ce niveau d'intégration et de réduction de coûts, un certain nombre de fonctionnalités ont été sacrifiées. Mais l'essentiel est toujours là : le connecteur 40 broches, le SoC Broadcom et sa mémoire, le support pour la carte microSD et le connecteur HDMI (qui est devenu mini-HDMI au passage).*

→ **USB_G_SERIAL** en module : ***Serial Gadget (with CDC ACM and CDC OBEX support)*** permet de se faire passer pour un adaptateur USB/série ;

→ **USB_MIDI_GADGET** en module : ***MIDI Gadget*** permet de créer un périphérique audio MIDI avec une entrée et une sortie apparaissant dans le système comme un périphérique ALSA standard ;

→ **USB_G_PRINTER** en module : ***Printer Gadget*** sert à établir un lien entre l'imprimante USB émulée et un programme recevant les données ;

→ **USB_CDC_COMPOSITE** en module : ***CDC Composite Device (Ethernet and ACM)*** est un des pilotes réunissant deux périphériques avec ici une interface USB Ethernet pour le réseau, plus un convertisseur USB/série ;

→ **USB_G_ACM_MS** en module : ***CDC Composite Device (ACM and mass storage)***, idem mais avec un adaptateur USB/série plus un support de stockage ;

→ **USB_G_MULTI** en module : ***Multifunction Composite Gadget*** permet de composer un périphérique multi-usage avec au choix Ethernet, USB/série et support de stockage ;

→ **USB_G_HID** en module : ***HID Gadget*** permet de créer un périphérique USB HID pour *Human Interface Devices*, typiquement un périphérique de saisie comme un clavier ou une souris ;

→ **USB_G_WEBCAM** en module : ***USB Webcam Gadget*** permet de créer un périphérique composite faisant office de source vidéo (classe USB Vidéo) comme une webcam et audio (classe USB Audio) comme une carte son USB.

L'avantage de procéder ainsi à la configuration des sources du noyau est d'avoir l'opportunité de faire d'autres choix personnels et d'explorer les options disponibles. Réduire le nombre de fonctionnalités prenant la forme de modules, par exemple, peut énormément accélérer la compilation tout en économisant de l'espace de la carte SD de la Pi.

Si cependant vous préférez utiliser un fichier de configuration « tout fait » comme celui que nous proposons au téléchargement sur GitHub, il vous suffit de le récupérer et le placer à la racine des sources du noyau sous le nom **.config**, puis de vous plier d'un :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- oldconfig
```

Mais ce n'est pas tout. Activer les bonnes options dans le noyau n'est pas suffisant, encore faut-il que le noyau puisse utiliser ces fonctionnalités. Fut un temps cela se passait en chargeant plus ou moins manuellement les modules, mais aujourd'hui tout repose

sur le *Device Tree* (voir *Hackable n°9*). Pour rappel, il s'agit d'une description du matériel contenue dans un fichier structuré, qui est utilisé par le noyau pour prendre en charge des éléments matériels qui ne peuvent être détectés automatiquement. Vous pouvez voir cela comme une description statique du matériel.

Une carte Raspberry Pi Zero utilise par défaut le même fichier que le modèle B (le hub supplémentaire est détecté au démarrage), mais ceci n'est pas correct dans notre cas car, justement, nous voulons utiliser un pilote différent pour le support USB. Nous devons donc avoir un fichier *Device Tree* différent et spécifique au modèle Zero. Nous n'allons pas reproduire ce fichier ici (trop long), mais je vous invite, tout simplement à le récupérer sur le GitHub du magazine. Il vous faudra télécharger ce fichier **bcm2708-rpi-zero.dts** et le copier dans **arch/arm/boot/dts/** avant de modifier un autre fichier se trouvant dans le même répertoire, **Makefile** en ajoutant ceci :

```
dtb-$(CONFIG_ARCH_BCM2708) += bcm2708-rpi-zero.dtb
```

avant :

```
dtb-$(CONFIG_ARCH_BCM2708) += bcm2708-rpi-b.dtb
dtb-$(CONFIG_ARCH_BCM2708) += bcm2708-rpi-b-plus.dtb
[...]
```

Ceci aura pour effet de créer automatiquement un fichier **bcm2708-rpi-zero.dtb** à partir de **bcm2708-rpi-zero.dts** au moment de la construction de l'image du noyau et des modules. Mais là encore ce n'est pas suffisant, nous devons aussi prévoir des *overlays* afin de pouvoir choisir le pilote que l'on souhaite utiliser pour le contrôleur USB (soit **dwc-otg** pour un fonctionnement « normal », soit **dwc2** pour l'OTG). Les deux fichiers à récupérer sont **dwc-otg-overlay.dts** et **dwc2-overlay.dts**, et doivent être placés dans **arch/arm/boot/dts/overlays/**. Là encore, le fichier **Makefile** qui se trouve dans le répertoire doit être étoffé des lignes :

```
dtb-$(RPI_DT_OVERLAYS) += dwc2-overlay.dtb
dtb-$(RPI_DT_OVERLAYS) += dwc-otg-overlay.dtb
```

insérées aux alentours de :

```
[...]
dtb-$(RPI_DT_OVERLAYS) += dht11-overlay.dtb
dtb-$(RPI_DT_OVERLAYS) += enc28j60-overlay.dtb
dtb-$(RPI_DT_OVERLAYS) += gpio-ir-overlay.dtb
[...]
```

Comme avec le fichier précédent, ceci aura pour effet de créer les ***.dtb** automatiquement à la construction.

Nous sommes maintenant au bout de nos manipulations. La configuration du noyau est changée et les différentes briques du *Device Tree* sont ajoutées et intégrées aux sources. Il ne nous reste plus qu'à générer un paquet pour notre Pi Zero :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KBUILD_DEBARCH=armhf deb-pkg
```

Note importante : à l'heure où cet article est écrit, les manipulations doivent être faites à la main, mais il est probable, au vu des discussions en cours entre développeurs concernant cette modification, qu'elles ne soient plus nécessaires dans un futur plus ou moins proche. En effet,

Norto a proposé cela en tant que contribution officielle (*Pull Request* #1239). Si ceci est accepté, ces changements seront fusionnés avec les sources officielles de la branche « rpi-4.4.y » et vous récupérerez cela automatiquement. Après le **`git clone`** du début d'article, jetez un œil au contenu de **`arch/arm/boot/dts/`**, pour vérifier si, par hasard, tout ceci ne serait pas déjà présent...

Au terme de la construction, vous devez trouver, dans le répertoire parent, un groupe de 4 fichiers **`.deb`** qui devront être transférés sur la Raspberry Pi Zero. Étant donné la faible connectivité de cette dernière, j'opte généralement pour la méthode suivante : placer la micro SD où se trouve Raspbian d'ores et déjà configuré dans un lecteur et accéder à la première partition (FAT) depuis un PC, y créer un répertoire **`k`** et y copier les fichiers **`.deb`**, pour enfin remettre la carte micro SD dans la Pi Zero et démarrer le système. Les fichiers **`.deb`** seront alors, pour la Pi, dans **`/boot/k`** :

- **`linux-image-4.4.0-rc6+_4.4.0-rc6+-1_armhf.deb`** : l'image du noyau, les modules, les fichiers *Device Tree*, etc.,
- **`linux-headers-4.4.0-rc6+_4.4.0-rc6+-1_armhf.deb`** : les fichiers d'entête nécessaires à la compilation de certains programmes ;
- **`linux-firmware-image-4.4.0-rc6+_4.4.0-rc6+-1_armhf.deb`** : les *firmwares* en logiciel libre pour certains périphériques ;
- **`linux-libc-dev_4.4.0-rc6+-1_armhf.deb`** : les fichiers d'entête en rapport avec la bibliothèque C.

## 5. INSTALLATION ET CONFIGURATION

Nous voici à présent sur le système Raspbian de la Raspberry Pi Zero, fraîchement démarré. Nos paquets Raspbian pour notre noyau et les autres éléments liés se trouvent dans **`/boot/k`**. Tout ce que nous avons à faire dans un premier temps est donc d'installer ces paquets avec :

```
$ sudo dpkg -i /boot/k/*.deb
```

Ceci aura plusieurs conséquences en termes d'installation de fichiers dans tout le système, mais le plus notable est sans le moindre doute l'image du noyau lui-même, qui est installée dans **`/boot`** sous le nom **`vmlinuz-4.4.0-rc6+`**. Le nom de fichier est peut-être très différent du classique **`kernel.img`**, mais c'est bien le même type de fichier.

Plutôt que de renommer ce nouveau noyau, nous allons le laisser tel qu'il est et glisserons une ligne spéciale dans **`config.txt`** afin de l'utiliser. Ceci présente l'avantage qu'en cas de problème, il nous suffira d'accéder à la partition FAT de la micro SD avec un PC pour éditer **`config.txt`** et revenir à l'état précédent du système, sans avoir à jongler avec les noms de fichiers.

Mais disposer de l'image du noyau n'est pas suffisant. Dans **`/boot`** n'ont pas été copiés les fichiers *Device Tree* lors de l'installation des paquets. C'est normal, le script qui crée les paquets dans les sources du noyau n'est pas au courant de la structure propre à une installation Raspbian, mais repose sur une installation classique GNU/Linux. Les fichiers en question se trouvent donc naturellement dans **`/usr/lib/linux-image-4.4.0-rc6+`**. Il nous suffit de les copier nous-mêmes :

```
$ sudo cp /usr/lib/linux-image-4.4.0-rc6+/bcm2708-rpi-zero.dtb /boot
$ sudo cp /usr/lib/linux-image-4.4.0-rc6+/overlays/* /boot/overlays
```

Il ne nous reste plus alors qu'à éditer **/boot/config.txt** pour ajouter quelques lignes :

```
kernel=vmlinuz-4.4.0-rc6+
device_tree=bcm2708-rpi-zero.dtb
dtoverlay=dwc2
```

La première précise le noyau à démarrer, la seconde le *Device Tree* à utiliser et la dernière l'*overlay* concernant notre choix explicite du support USB. Ceci fait, enregistrez le fichier, croisez les doigts et... **sudo reboot**.

Si tout se passe bien et que le système redémarre pour arriver sur l'invite de connexion (sinon, essayez un **hdmi_safe=1** dans **config.txt**), il ne vous reste plus qu'à vous connecter et commencer par vous assurer de la version du noyau en cours :

```
$ uname -a
Linux raspberrypi 4.4.0-rc6+ #3 PREEMPT
Sat Jan 2 15:33:35 CET 2016 armv6l GNU/Linux
```

puis de regarder dans les messages système si vous y trouvez une mention du pilote que nous avons pris tant de temps à mettre en œuvre :

```
$ dmesg | grep dwc2
dwc2 20980000.usb: EPs: 8, dedicated fifos, 4080 entries in SPRAM
dwc2 20980000.usb: DWC OTG Controller
dwc2 20980000.usb: new USB bus registered, assigned bus number 1
dwc2 20980000.usb: irq 33, io mem 0x00000000
usb usb1: Manufacturer: Linux 4.4.0-rc6+ dwc2_hsotg
```

À ce stade, un premier test pourra consister à connecter un périphérique USB et voir si, avec un adaptateur USB OTG, il est correctement détecté. Si tel est le cas, il doit apparaître dans la sortie de la commande **lsusb**. Ceci correspond au fonctionnement « classique » de toutes les Raspberry Pi.

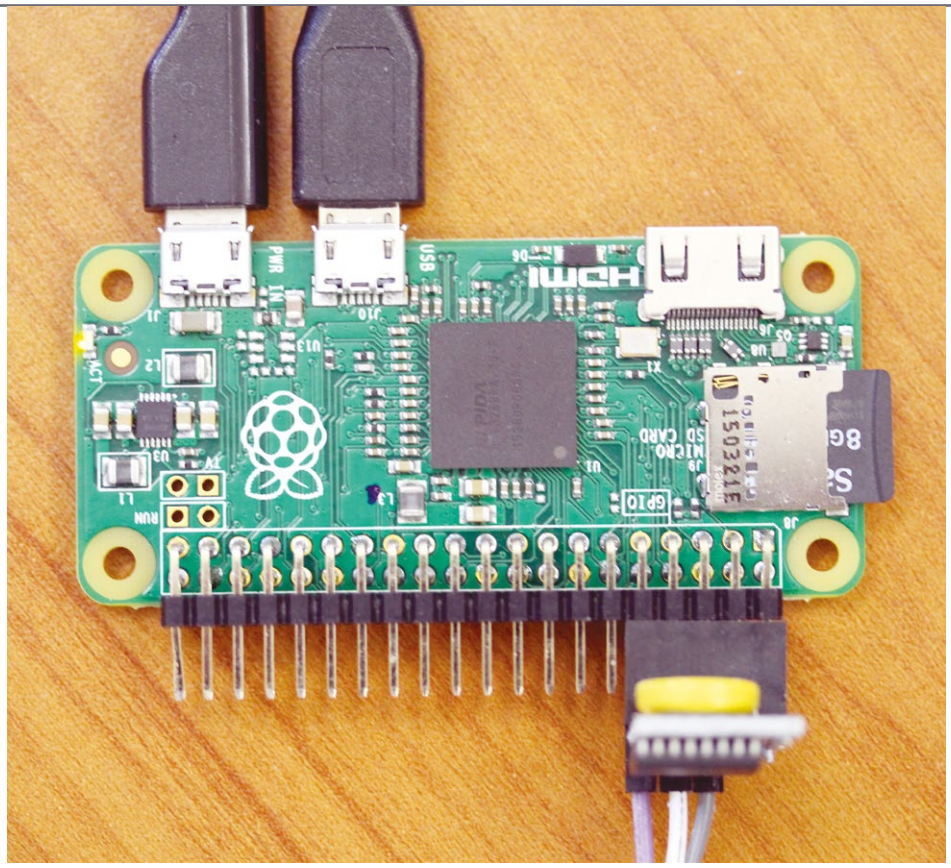
## 6. USB GADGET : EST-CE QUE ÇA MARCHE ?

Procédons tout d'abord à un essai simple grâce au module **g_serial**. L'objectif est de connecter le port USB de la Raspberry Pi Zero à un PC (ou une autre Pi) à l'aide d'un câble classique comme ceux utilisés pour les smartphones. Mais avant cela, côté Pi, nous chargeons le module et lançons le service permettant de fournir une invite de login :

```
$ sudo modprobe g_serial
$ sudo systemctl start getty@ttyGS0.service
```

Notez qu'ici nous utilisons Raspbian Jessie, la dernière version, qui remplace (bien malheureusement) le bon vieux système d'init System V (SysV init) par l'horreur qu'est Systemd (avis tout personnel, mais néanmoins objectif en ce qui me concerne). Cette dernière commande lance un service Getty sur **/dev/ttyGS0** correspondant à « notre côté » du « faux » port série.

Lorsque nous branchons le câble USB côté PC, un nouveau port apparaît et nous pouvons l'utiliser pour obtenir une session comme nous le ferions avec un adaptateur USB/série connecté aux broches 8 et 10 (GPIO14 et 15 correspondant à TXD/RXD).



*Compacité et faible coût sont les deux avantages les plus évidents de la Raspberry Pi Zero. Mais lorsqu'on y regarde de plus près et qu'on retrousse ses manches pour compiler un nouveau noyau, ces qualités s'en trouvent presque effacées tant l'utilisation comme périphérique USB peut ouvrir des perspectives intéressantes...*

C'est amusant et pratique, mais ce n'est pas très impressionnant. Tournons-nous alors vers quelque chose de plus exotique : **g_ether**, le module faisant passer notre Pi Zero pour une interface réseau Ethernet USB. Avant tout, arrêtez le service et déchargez le module :

```
$ sudo systemctl stop getty@ttyGS0.service
$ sudo rmmod g_serial
```

On peut ensuite charger le module **g_ether** et configurer le réseau côté Pi :

```
$ sudo modprobe g_ether
$ sudo ifconfig usb0 192.168.111.1 netmask 255.255.255.0
```

La commande **ifconfig** permet de rapidement et temporairement configurer une interface réseau. **usb0** correspond ici, encore une fois, au « côté Pi » de la connexion et nous donnons à cette interface l'adresse 192.168.111.1.

Après connexion côté PC GNU/Linux, nous avons également une nouvelle interface réseau qui apparaît (**usb0** également) et nous utilisons une commande **ifconfig** équivalente, mais avec une autre adresse bien sûr :

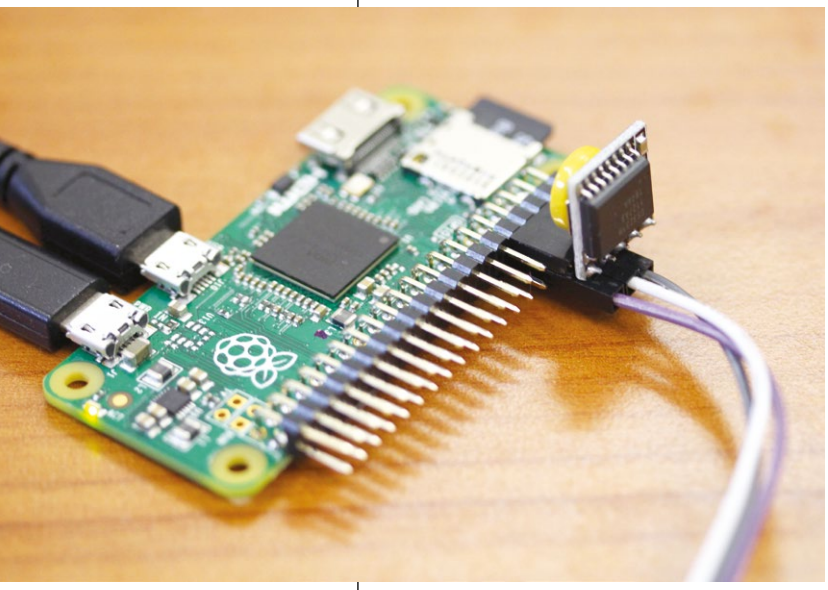
```
$ ifconfig usb0 192.168.111.2 netmask 255.255.255.0
```

Une liaison réseau est maintenant formée entre l'un (192.168.111.1) et l'autre (192.168.111.2) système et nous pouvons rapidement la tester avec **ping** et l'adresse distante :

```
% ping -c 3 192.168.111.1
PING 192.168.111.1 (192.168.111.1) 56(84) bytes of data.
64 bytes from 192.168.111.1: icmp_seq=1 ttl=64 time=0.351 ms
```

```
64 bytes from 192.168.111.1: icmp_seq=2 ttl=64 time=0.342 ms
64 bytes from 192.168.111.1: icmp_seq=3 ttl=64 time=0.346 ms

--- 192.168.111.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.342/0.346/0.351/0.015 ms
```



*Comme les modèles A et A+, la carte Raspberry Pi Zero ne dispose ni de hub USB, ni d'interface Ethernet. Ce modèle cependant va un cran plus loin en supprimant également le connecteur audio, ainsi que celui pour l'écran LCD et la caméra. Notez ici, à droite, l'ajout d'un module RTC DS3231 en i2c ainsi que la connexion pour la console série via un module série/USB 3,3V.*

Et, bien entendu, comme SSH est lancé par défaut avec Raspbian, nous pouvons tout aussi bien nous connecter à la Pi Zero depuis le PC avec un simple **`ssh pi@192.168.111.1`** (ou encore transférer des fichiers avec **`scp`**). Ceci, bien entendu, devrait également fonctionner entre la Pi Zero et une machine Windows ou Mac OS X (mais je n'ai pas testé).

Pour une configuration permanente, il faudra se battre avec **`systemd`**, le nouveau système d'initialisation utilisé depuis Raspbian Jessie. Et ce sans finir par faire une crise de nerfs. Précédemment, il aurait suffi d'ajouter **`g_ether`** dans le fichier **`/etc/modules`** ainsi que quelques lignes dans **`/etc/network/interface`**. Ce n'est plus le cas et un problème nous barre la route : **`systemd`** tente de charger les modules dans **`/etc/modules`** avant que le pilote du contrôleur USB ne soit chargé par le noyau. Il ne semble pas exister pour l'instant de solution simple qui ne relève pas du bricolage, du moins avec les modules **`g_*`** comme **`g_ether`** (la solution consiste à ne plus utiliser les modules en question, mais reposer directement sur le nouveau système de gestion USB Gadget, plus complexe et passant par une « composition » du périphérique dans **`/sys/kernel/config`**).

## CONCLUSION

Vous trouverez ou non un usage pour cette fonctionnalité, tout dépend de la nature de vos projets. Personnellement, je trouve que le simple fait de disposer d'une connexion réseau via USB de cette manière pour la Raspberry Pi Zero est quelque chose qui mériterait d'être mis bien plus en avant, car offrant des perspectives très intéressantes. Il est fort probable que les manipulations sur le noyau décrites ici deviennent un élément par défaut des futures versions de Raspbian, du moins pour ce modèle.

Enfin, même si cela est très technique, nous avons ici l'opportunité de jouer avec quelque chose qu'il est presque impossible d'obtenir sur PC et ainsi d'envisager l'émulation de bien des périphériques USB. Le niveau de programmation nécessaire est totalement hors du cadre du magazine, mais avec un peu de travail et de temps, une Pi Zéro peut être littéralement transformée en n'importe quel périphérique... **DB**

PROFESSIONNELS DES TICE, COLLECTIVITÉS, ÉCOLES D'INGÉNIEURS, UNIVERSITÉS, R & D, ENSEIGNANTS, ...

HACKABLE MAGAZINE



# PROFESSIONNELS !

## DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...

## ...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR :

# www.ed-diamond.com

## PDF COLLECTIFS PRO

Prix TTC en Euros / France Métropolitaine

| OFFRE | ABONNEMENT | 1 - 5 lecteurs Réf | 1 - 5 lecteurs Tarif TTC | 6 - 10 lecteurs Réf | 6 - 10 lecteurs Tarif TTC | 11 - 25 lecteurs Réf | 11 - 25 lecteurs Tarif TTC |
|---|---|---|---|---|---|---|---|
| PROHK2 | 6n° HK | ☐ PRO HK2/5 | 156,- | ☐ PRO HK2/10 | 312,- | ☐ PRO HK2/25 | 624,- |

PROFESSIONNELS : N'HÉSITEZ PAS À NOUS CONTACTER POUR UN DEVIS PERSONNALISÉ PAR E-MAIL : abopro@ed-diamond.com OU PAR TÉLÉPHONE : 03 67 10 00 20

## ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO OPEN SILICIUM

Prix TTC en Euros / France Métropolitaine

| OFFRE | ABONNEMENT | 1 - 5 connexion(s) Réf | 1 - 5 connexion(s) Tarif TTC | 6 - 10 connexions Réf | 6 - 10 connexions Tarif TTC | 11 - 25 connexions Réf | 11 - 25 connexions Tarif TTC |
|---|---|---|---|---|---|---|---|
| PROOS+3 | OS | ☐ PRO OS+3/5 | 90,- | ☐ PRO OS+3/10 | 180,- | ☐ PRO OS+3/25 | 360,- |
| PROH+3 | GLMF + HS + LP + HS + MISC + HS + OS | ☐ PRO H+3/5 | 447,- | ☐ PRO H+3/10 | 894,- | ☐ PRO H+3/25 | 1788,- |

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France HS = Hors-Série LP = Linux Pratique OS = Open Silicium

**SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE CI-DESSUS ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !**

| Voici mes coordonnées postales : | |
|---|---|
| Société : | |
| Nom : | |
| Prénom : | |
| Adresse : | |
| | |
| Code Postal : | |
| Ville : | |
| Pays : | |
| Téléphone : | |
| E-mail : | |

☐ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

☐ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

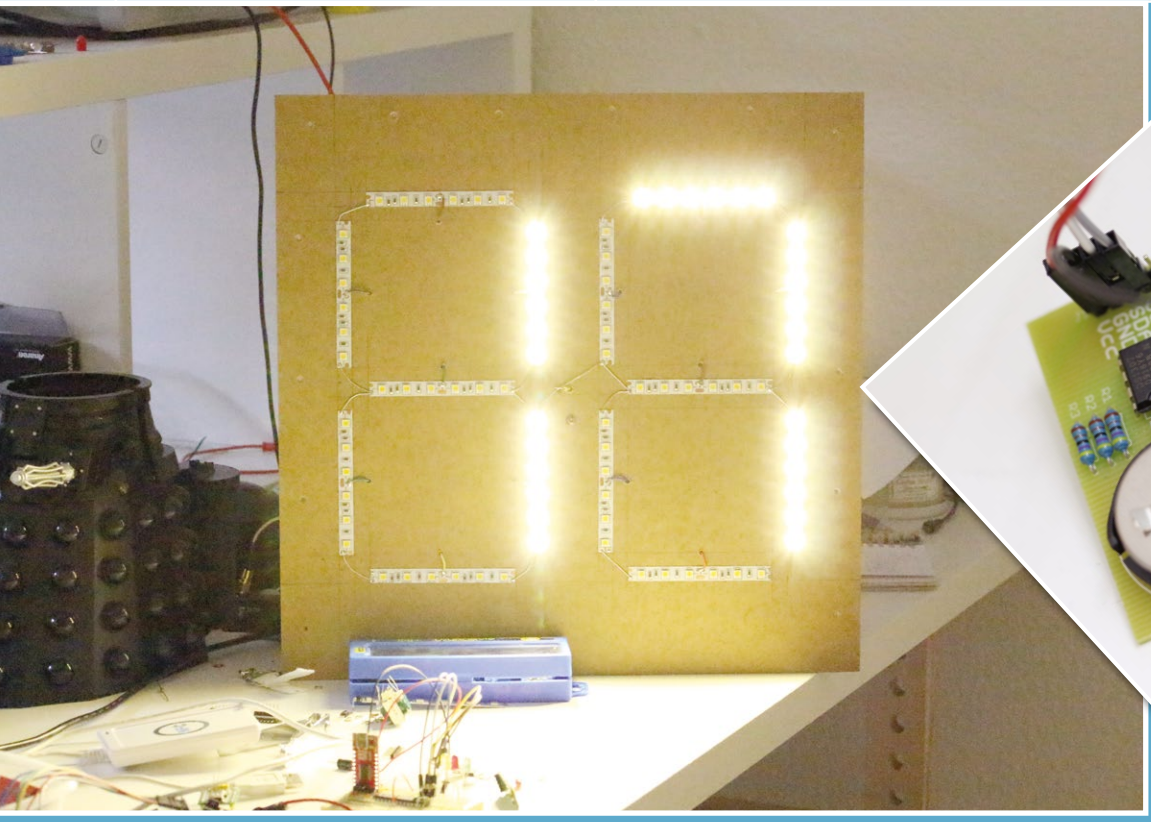
HACKABLE MAGAZINE

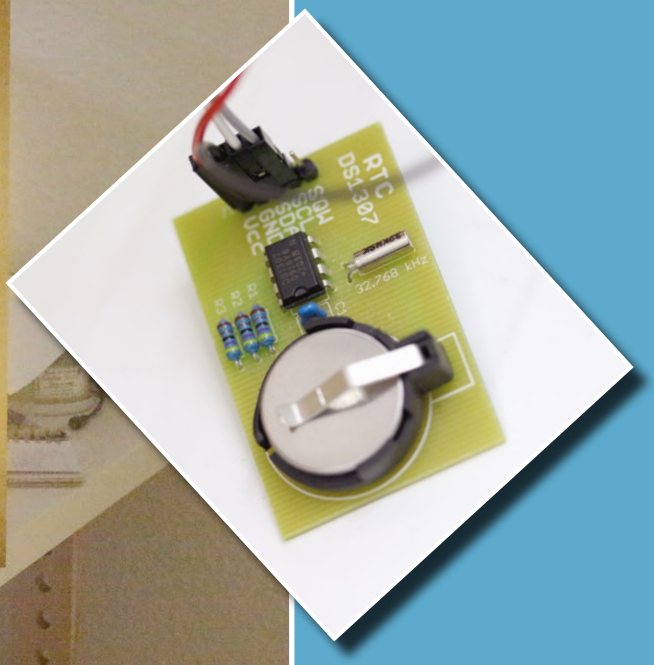
Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

**Vos remarques :**

.....

.....





# CRÉEZ UN (DÉ)COMPTEUR GÉANT MULTI-USAGE ET STRESSANT

*Denis Bodor*

Qui n'a pas besoin d'un afficheur géant à deux chiffres ou plus ? Dit comme ça, cela peut paraître un peu surprenant, mais voici l'idée : afficher une valeur numérique à deux gros chiffres avec des leds... plein de leds, et ce de façon visible de loin. Score, décompte, horloge, compteur, température... L'information à afficher est laissée à votre imagination, il n'y a que l'embarras du choix. Dans mon cas, l'objectif est simple : afficher le nombre de jours restants pour finir le prochain numéro du magazine !

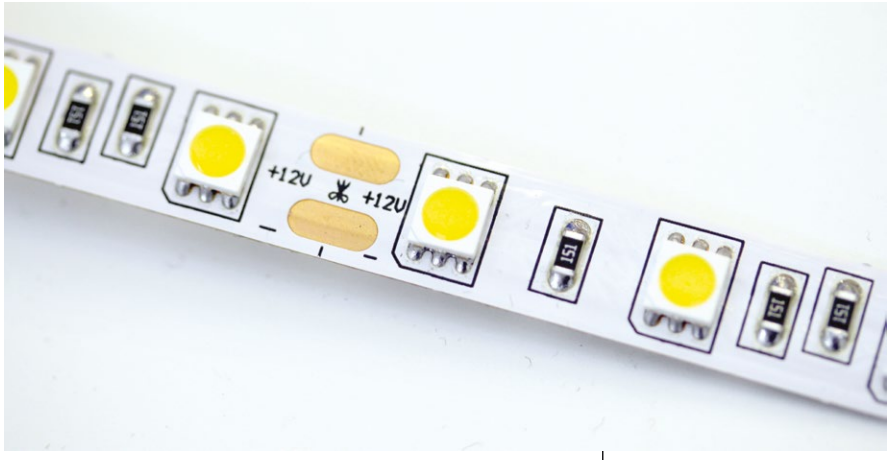
| NIVEAU | TEMPS | BUDGET |
|---|---|---|
| | 1 heure | 35 € (Arduino inclus) |

Les gros systèmes d'affichage, on en voit partout. Information sur votre vitesse au bord des routes, prix de carburant, horloges dans les lieux comme les gares, affichage du client suivant à la boucherie du coin, température actuelle sur une place publique... Mon objectif premier pour cet article est de créer un afficheur à deux chiffres d'environ trente centimètres de haut, couplé à une carte Arduino et un module horloge/RTC dans le but d'afficher un décompte du nombre de jours restants pour finaliser le prochain *Hackable* (oui, je travaille mieux en étant stressé).

Bien que je me sois personnellement arrêté sur un usage précis, il est important de remarquer que ce qui va suivre reste très générique, aussi bien quant à la façon d'obtenir cet afficheur, que concernant son utilisation finale. Le but est, en utilisant un nombre raisonnablement réduit de ports de l'Arduino, de faire ni plus ni moins que ce que l'on peut déjà obtenir avec des afficheurs 7 segments, mais en grand... voire en très grand si nécessaire.

## LE PRINCIPE

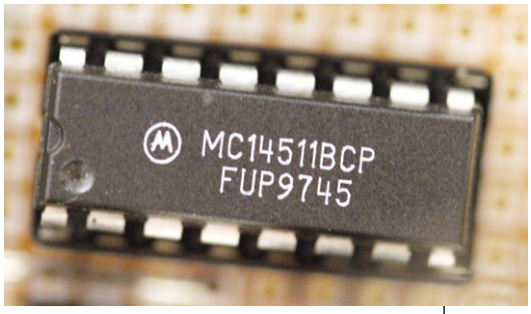
Le projet ici mélange plusieurs domaines : bricolage, électronique et programmation. Le bricolage concernera la création de deux afficheurs 7 segments. Ces composants, tels que vendus dans le commerce, se divisent en deux types : à anode commune et à cathode commune. Il ne s'agit que d'un ensemble de leds agencées de façon à former un chiffre. Comme les leds sont des composants polarisés il y a deux façons d'assembler tout ça : connecter toutes les broches négatives ensemble et alimenter chaque segment par les broches restantes (cathode commune) ou réunir toutes les broches positives et mettre à la masse les broches qui nous intéressent (anode commune).




*Les rubans de leds sécables se déclinent en bien des variations : couleur unique, leds RVB, leds intelligentes type WS2812b, avec ou sans résine de protection, etc. Le modèle utilisé ici est initialement destiné à l'éclairage intérieur, fonctionne en 12 volts et avec une densité de 60 leds blanches par mètre.*

Le bricolage consistera donc à créer de toutes pièces deux afficheurs 7 segments à partir de leds. Plusieurs solutions peuvent être envisagées, à commencer par l'assemblage de leds et de résistances sous forme de barrettes de plusieurs leds. Il nous en faudra 14 et ceci demande beaucoup de travail et de précision (un défaut d'alignement serait clairement visible). Heureusement pour nous, des leds ainsi assemblées sont directement disponibles à l'achat sous la forme de rubans sécables, généralement utilisés pour l'éclairage et la décoration intérieure. Il en existe de tous types et de toutes couleurs.

L'objectif du montage est, pour moi, de disposer d'un décompte du nombre de jours restants avant la mise en route de la mise en page d'un nouveau numéro du magazine. En d'autres termes, c'est le nombre de jours où je peux encore mettre quelques idées en pratique, expérimenter et rédiger pour un numéro donné. Le magazine étant bimestriel, je n'ai pas besoin de plus de deux chiffres, mais vous comprendrez rapidement que les explications qui vont suivre peuvent être adaptées pour n'importe quel nombre de chiffres. La taille de l'afficheur n'a pas



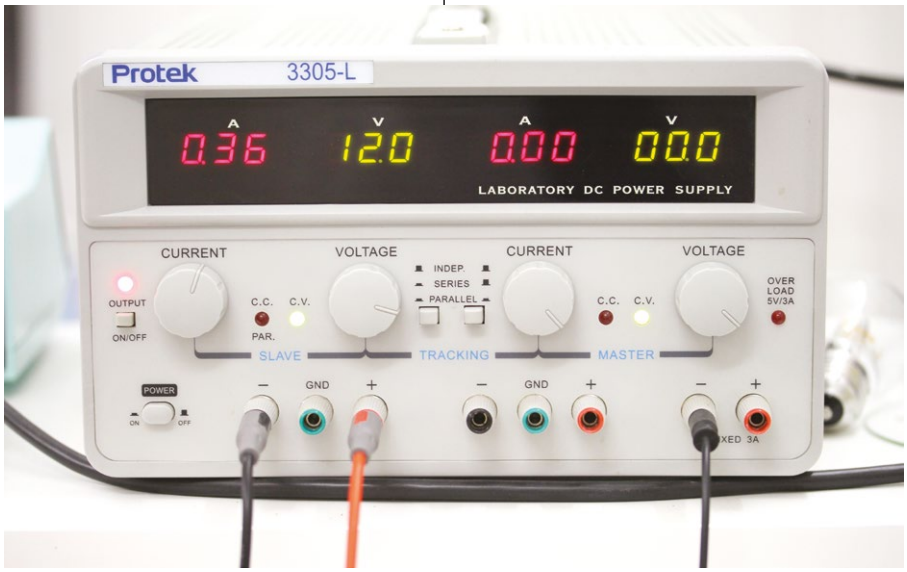

*Le 4511 est un circuit logique disponible chez de nombreux constructeurs. Ici un modèle un peu ancien (45ème semaine de 1997) de chez Motorola. D'autres composants fonctionnent de la même façon, ils sont tous décrits comme des « décodeurs BCD vers 7 segments ».*

besoin d'être titanesque, il faut simplement que je puisse lire le chiffre rapidement et à une distance de quelques mètres. Des chiffres d'environ 30 centimètres de haut suffiront pour me stresser...

Une fois les afficheurs 7 segments créés, il faudra pouvoir les piloter. Contrairement à des afficheurs classiques de moins de deux centimètres de hauteur, on parle ici de plusieurs leds par segment (6 pour être précis, sachant que chaque led du ruban utilisé intègre en réalité 3 *die* par led, ce qui donne en réalité 18 leds par segment). Il est absolument hors de question qu'une carte Arduino puisse piloter cela directement. De plus, 2 chiffres de 7 segments nous donnent 14 segments ce qui est un peu pénible, car nous risquons d'être rapidement limités si nous voulons étendre le projet (à plus de chiffres ou en intégrant d'autres fonctions). Heureusement pour nous, les afficheurs 7 segments existaient bien avant le « raz de marée » des microcontrôleurs et bien avant que quelqu'un ne parle de cartes Arduino. Pour piloter ces afficheurs ont donc été créés des circuits logiques intégrés spécialisés prenant en entrée une valeur binaire entre 0 et 9 et fournissant en sortie l'état des différents segments. On appelle ces composants des convertisseurs BCD vers 7 segments et le plus connu et le 4511.

Ce n'est pas tout, un circuit logique n'est pas en mesure de fournir le courant nécessaire aux leds et encore moins avec la tension attendue (12V). Nous devons donc utiliser les signaux en sortie, correspondants à l'état des segments, et les utiliser pour contrôler l'alimentation de nos segments à une tension bien supérieure aux 5 volts du reste du montage. Il existe bien des manières de procéder, mais nous allons utiliser ici un composant que nous n'avons pas encore abordé dans les pages du magazine : l'ULN2803A. Il ne s'agit pas d'un circuit logique, mais d'un ensemble de transistors réunis en un seul composant possédant 8 entrées et 8 sorties, prévu pour piloter des relais, des lampes, des séries de leds, des moteurs, etc.

*L'alimentation des morceaux de rubans de leds dépendra de leurs spécifications, mais aussi de la taille des segments et de l'affichage en cours. Ici, les chiffres « 17 » (5 segments de 6 leds) nécessitent quelques 360 mA, valeur qui dépassera l'ampère avec « 88 »...*




Et enfin, nous piloterons tout cela à partir d'une carte Arduino à laquelle sera également reliée une source de temps, une RTC à base de DS1307, nous fournissant une date et une heure précise.

# CE QU'IL VOUS FAUT

- Une carte Arduino quelconque ou une autre plateforme du même type. Il n'y a pas de restrictions particulières ici, mais tout dépendra de votre projet pour cet afficheur. Dans mon cas, il s'agit de décompter des jours et je reposerai sur un module RTC me fournissant l'information dont j'ai besoin. Il me faudra donc une bibliothèque pour ce module dans mon environnement afin de ne pas avoir à développer toute cette partie. Si votre projet ne repose pas sur une donnée temporelle précise, vous n'aurez pas besoin d'une telle chose et cette restriction sera levée.

- Environ 1,5m de ruban de leds blanches (*led strip*) : ici aussi, c'est un choix tout personnel, mais ce type de produit est généralement peu coûteux et très agréable d'utilisation. Il s'agit tout simplement de leds montées avec leurs résistances sur un ruban souple adhésif d'une longueur et avec une densité au choix. J'ai opté pour un rouleau de 5 mètres de leds blanches composé de 300 leds alimentées en 12 volts, vendu environ 9€ (port gratuit) sur eBay par un vendeur nommé *top-shopping-mall*. Le ruban est sécable toutes les 3 leds et pour ce projet chaque segment sera composé de 6 leds (~10 cm). Notez qu'il s'agit de simples leds blanches et non de leds « intelligentes » type WS2812b.

- Un support quelconque : les morceaux de ruban de leds sont équipés d'adhésifs double-face permettant de les coller sur le support de son choix. Il vous sera possible d'opter pour la matière avec laquelle vous êtes le plus à l'aise tout en prenant en compte certaines considérations : évitez les matériaux conducteurs, car même si l'adhésif est un isolant vous n'êtes pas à l'abri de mauvaises surprises ; choisissez quelque chose de rigide, mais facile à percer, l'objectif est de masquer au maximum les fils et donc de les placer à l'arrière du support ; gardez à l'esprit les éventuels problèmes liés à la température, en particulier avec les leds de forte luminosité.

- Deux circuits intégrés type 4511 : le 4511 est un classique, mais il n'est pas le seul pouvant fournir les fonctionnalités recherchées. C'est un convertisseur BCD et 7 segments. BCD signifie *Binary Coded Decimal* ou, en bon français, décimal codé binaire. Il s'agit d'une représentation d'une valeur numérique entre 0 et 9 par 4 bits : `0000=0`, `0001=1`, `0010=2`... `1001=9`. Ce type de circuit intégré prend donc en entrée 4 bits et fournit en sortie l'état des 7 segments (libellés « a » à « g ») permettant de dessiner un chiffre. Un cousin du 4511 est le 74LS47 (ou 7447) qui est parfaitement compatible.

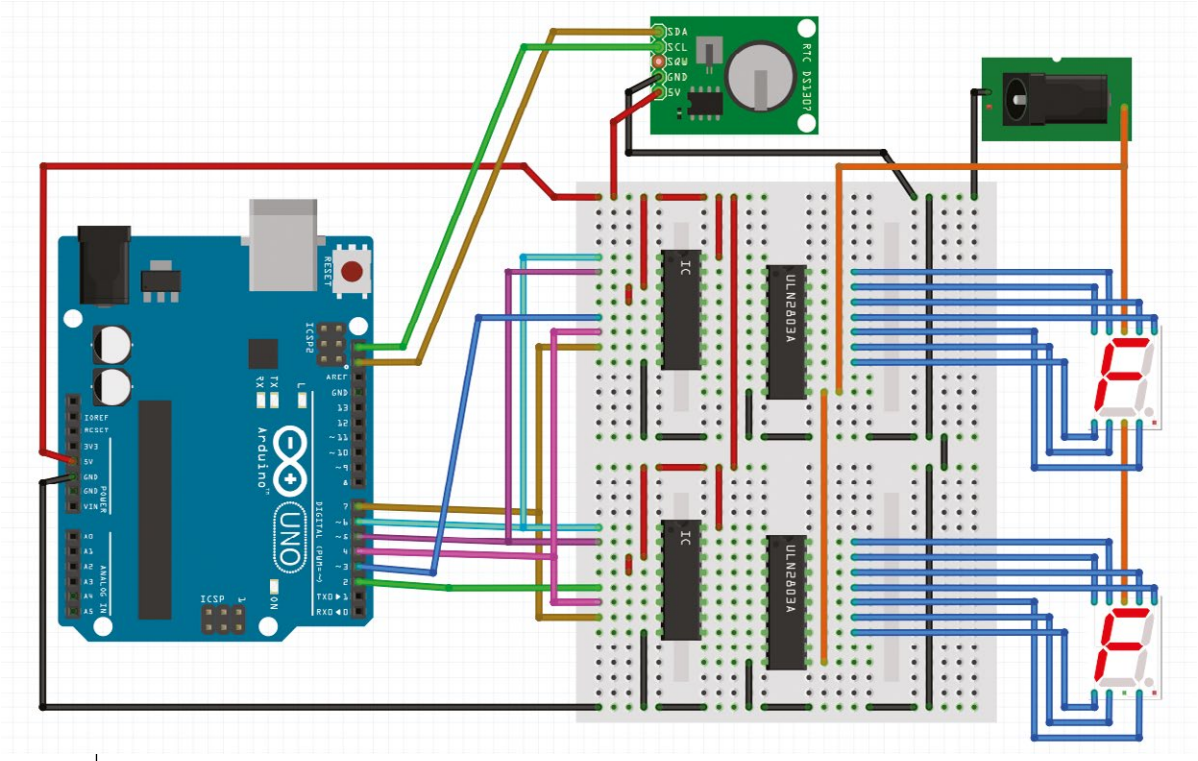
- Deux ULN2803A : un 4511 est parfaitement capable de prendre en charge un afficheur 7 segments standard. Chacune de ses sorties peut fournir un courant de 25 mA largement suffisant pour alimenter une voire deux leds composant un segment. Dans notre cas, avec des morceaux de ruban nécessitant quelque 60 mA, nous sommes hors limite. Nous devons donc, à partir des signaux fournis par le 4511, piloter nos morceaux de ruban avec le courant qu'ils demandent. Plusieurs solutions sont envisageables, transistors, MOSFET, relais... J'ai choisi ici d'opter pour le ULN2803A qui n'est pas un circuit logique, mais un ensemble de transistors en montage dit *Darlington* réunis dans un unique composant. Un ULN2803A pourra fournir jusqu'à 500 mA par sortie, et ce avec une tension pouvant aller jusqu'en 50 volts.

- Alimentation adaptée : le ruban de leds utilisé ici fonctionne en 12V et une carte Arduino n'est pas en mesure de fournir cette tension. Il nous faut donc une autre source d'alimentation. En faisant rapidement le calcul, chaque segment demande 60 mA et nous avons 7 segments fois deux afficheurs « maison », ce qui nous donne 60x7x2 = 840 mA en 12V, soit plus de 10W. Une alimentation 12V stable de 1 ampère devrait faire l'affaire lorsque nous affichons « 88 » (tous les segments allumés). Ce type d'adaptateur secteur est relativement courant, vous en trouvez pour une poignée d'euros aussi bien dans des boutiques en ligne, que chez Emmaüs (voire gratuitement si, comme moi, vous ne jetez jamais rien qui puisse être utile un jour).

- Un module RTC à base de DS1307 : c'est un complément classique dès lors qu'on souhaite travailler avec des dates et des heures. Ce type de module fournit une base temporelle relativement précise pour une paire d'euros (avec la pile et le port).

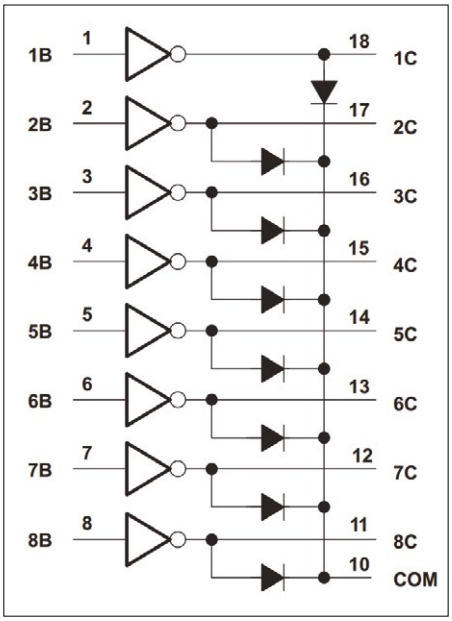
- Une platine à essais et câbles : ceci ne compte que pour l'expérimentation et sera ensuite écarté au bénéfice, par exemple, d'un assemblage sur une plaque pastillée moins chère, plus fiable et plus compacte pour la réalisation finale.

# LE MONTAGE




Certes, je reconnais que, vu comme ça, c'est un peu terrifiant. Je vous rassure, sur une platine à essais et avec la masse de connexions d'un côté comme de l'autre, c'est encore pire ! Cependant, tout ceci est largement moins complexe qu'il n'y paraît. Pour comprendre l'utilité et l'interconnexion des différents éléments, il suffit de voir cela tranche par tranche.

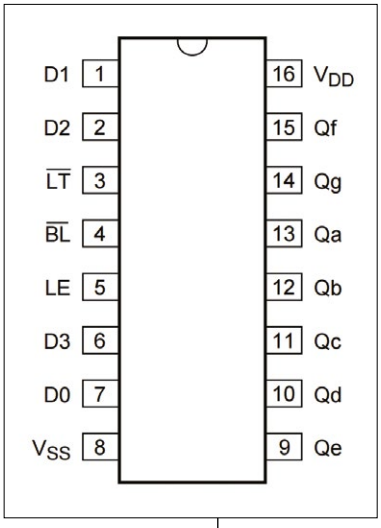
Nous commencerons par la fin (à droite) avec les afficheurs 7 segments « faits maison ». Tout comme avec ceux représentés sur le schéma, nous travaillons avec une anode commune. Tous les morceaux de rubans sont connectés à la source d'alimentation +12 volts. En connectant la borne négative à la masse, le courant circule et les leds s'allument. Nous allons donc contrôler cette mise à la masse à l'aide des ULN2803A qui, et c'est important de le comprendre, ne fournissent pas de courant, mais en acceptent sur leur sortie. Le schéma logique simplifié de l'ULN2803A est le suivant :



Sur la gauche, les broches 1 à 8 permettent de connecter un circuit logique ou un microcontrôleur, en 0/5 volts. Sur la droite et les broches 11 à 18, nous avons les « sorties » sur lesquelles sont connectées les bornes négatives de nos morceaux de ruban et donc avec une tension de 12 volts. La broche 10 est connectée à cette même tension. Ceci et grâce aux diodes présentes dans le composant, forme une protection permettant de connecter directement un moteur ou un relais à un ULN2803A sans avoir à ajouter une diode de roue libre (protection contre un retour de courant lors de l'effondrement du champ magnétique d'une bobine). Même si ce n'est pas notre cas ici, il faut **toujours** connecter sur la broche 10 la même source que celle alimentant les composants pilotés (+12v donc). Pour autant, il ne faut **jamais** voir dans cette broche une quelconque forme d'alimentation de l'ULN2803A : ce n'est pas un circuit logique, mais un ensemble de transistors Darlington (deux transistors montés en cascade). Enfin, chose qui n'apparaît pas dans le schéma logique, la broche 9 est la masse.

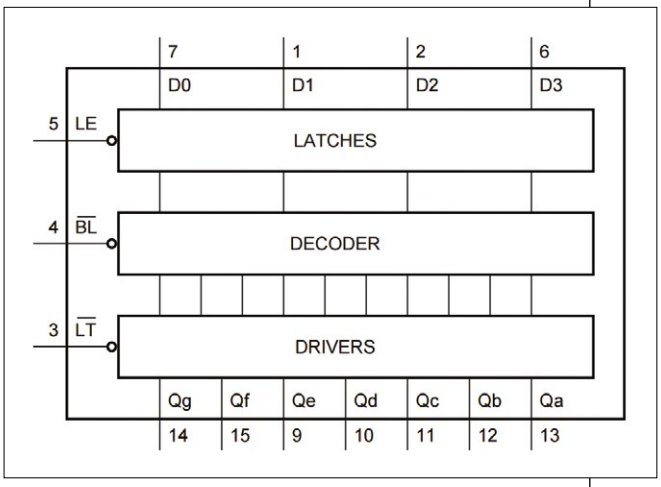
L'utilisation de l'UL2803A est donc relativement simple : à gauche le 5V, à droite le 12V. Dès qu'un niveau logique haut est appliqué sur une broche en entrée à gauche, la sortie correspondante laisse circuler le courant de la broche en question vers la masse.

Ce qui nous amène à la tranche suivante : le contrôle des entrées de l'ULN2803A. 7 d'entre elles sont directement reliées à la sortie du 4511, le fameux décodeur BCD vers 7 segments. La huitième n'est pas utilisée, tout comme la sortie correspondante de l'ULN2803A (mais rien n'empêche de faire évoluer le montage pour, par exemple, ajouter une led en guise de séparateur décimal en passant outre le 4511).

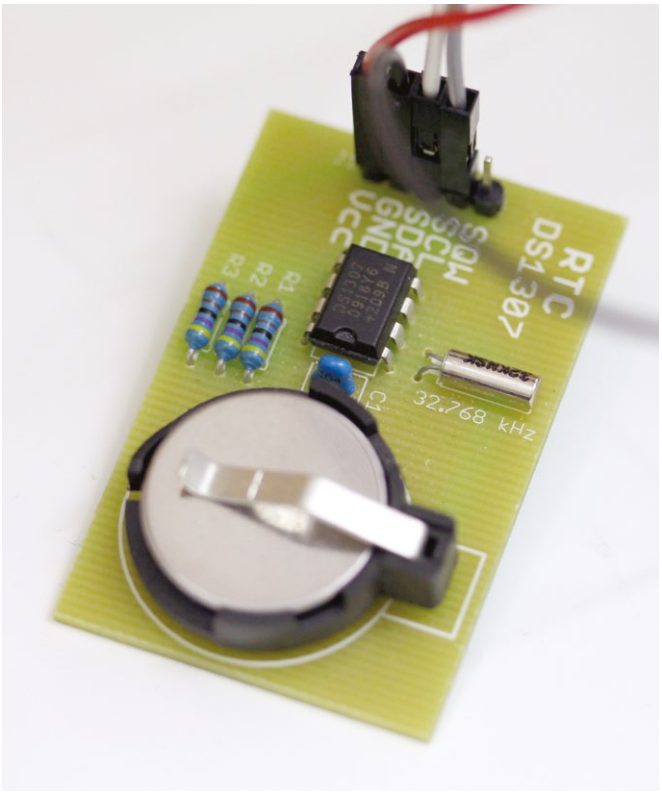



Le 4511 fait le plus gros du travail et ce, vous l'avez compris, entièrement du côté 5V du montage. Ce circuit dispose de 16 broches :

On retrouve l'alimentation (16) et la masse (8) ainsi que 7 sorties (9 à 15) correspondant aux 7 segments d'un afficheur. En entrée, nous avons 4 bits pouvant former une valeur entre 0 et 9 (les autres combinaisons de bits, correspondant aux valeurs de 10 à 15 ne provoquent rien, toutes les sorties sont à la masse). Il nous reste donc trois broches, /LT (3), /BL (4) et LE, possédant des fonctions particulières :




Nous avons ici, en haut nos 4 bits en entrée. LE est un signal permettant de demander au 4511 de prendre en compte les bits en entrée. La logique est donc la suivante : la carte Arduino définie l'état des 4 bits puis passe la broche LE de +5V à la




*L'indispensable module de tout projet nécessitant une base de temps fiable : le module RTC à base de DS1307 et sa pile bouton. Notez que celui-ci est relativement ancien et grossier. Les modèles plus récents sont plus compacts et généralement de meilleure facture.*

masse, ce qui permet de faire transiter ces bits sur le bloc suivant, contrôlé par /BL. Le « / » devant le nom dénote une utilisation en logique inversée (ou actif à l'état bas). « BL » signifie *blanking* et si cette broche est à la masse, l'ensemble des 7 sorties sera également à la masse. Ceci permet, par exemple, d'éteindre un chiffre non significatif de l'affichage, comme le « 0 » dans « 01 » (*ripple blanking* en anglais). Ici, nous n'avons pas besoin de ce signal, « 00 » fait partie du « jeu » (un double zéro est davantage stressant). Nous avons donc cette broche en permanence connectée au +5V.

/LT est également un signal actif à l'état bas. Lorsque cette broche est mise à la masse, l'ensemble des segments sera à l'état haut, permettant ainsi de s'assurer que les connexions et l'afficheur lui-même fonctionnent. « LT » signifie d'ailleurs *lamp test* (test des lumières). Là encore, nous n'avons pas besoin de contrôler cela depuis l'Arduino et relions donc cette broche au +5V.

La notion de *latches* contrôlées par la broche LE est très importante. On la retrouve dans un grand nombre de composants logiques aussi utiles que le 4511 (compteurs, registre à décalage, buffers, décodeurs, etc.). Ce signal demandant la prise en compte des valeurs sur les entrées nous permet d'économiser 4 sorties de la carte Arduino. En effet, il ne nous est pas nécessaire d'utiliser 4 sorties pour un 4511 et 4 autres pour un second. Tout ce que nous avons à faire est de présenter nos 4 bits aux deux 4511 et d'utiliser la broche LE du 4511 qui nous intéresse (ce qui nous donne un bus de 4 bits). Nous pouvons ainsi contrôler l'état de 14 segments avec seulement 6 broches de la carte Arduino. Mieux encore, chaque chiffre supplémentaire ne nous coûtera qu'une seule sortie en plus, celle devant être reliée au LE du 4511 supplémentaire.

Pour finir, ce circuit utilise un module RTC à base du très courant DS1307, déjà présenté dans ce magazine à plusieurs reprises. Il s'agit d'un montage composé d'un circuit intégré, un oscillateur à quartz, quelques composants passifs et d'un support pour pile bouton CR2032. C'est tout bonnement une horloge fournissant la date et l'heure lorsqu'on le lui demande. La présence de la pile permet de maintenir son fonctionnement même lorsque le reste du montage n'est pas alimenté. Ce type de module utilise un bus i2c et sa connexion se résumera à : l'alimentation +5V, la masse et les signaux SDA et SCL accessibles directement sur la carte Arduino via les broches dédiées (ou respectivement A4 et A5 sur Arduino Uno).

# LE CROQUIS

Fichier Édition Croquis Outils Aide

```
#include <Wire.h>
#include <Time.h>
#include <DS1307RTC.h>

// Macro pour s'y retrouver dans les broches
#define BIT0    7
#define BIT1    6
#define BIT2    5
#define BIT3    4
#define LATCH_G 3
#define LATCH_D 2

// Fonction envoyant un état bas durant 1ms
void latch(int digit) {
  digitalWrite(digit, LOW);
  delay(1);
  digitalWrite(digit, HIGH);
  delay(1);
}

// Fonction envoyant les deux valeurs à l'afficheur
void setdigits(int val) {
  if(val < 10) {
    // en dessous de 10, le chiffre de gauche est toujours 0
    digitalWrite(BIT0, LOW);
    digitalWrite(BIT1, LOW);
    digitalWrite(BIT2, LOW);
    digitalWrite(BIT3, LOW);
    latch(LATCH_G); // validation de la valeur
  } else {
    // sinon on traite les dizaines
    // On divise la valeur par 10
    // puis on teste chacun des 4 bits
    digitalWrite(BIT0, (val/10) & B00000001 ? HIGH : LOW);
    digitalWrite(BIT1, (val/10) & B00000010 ? HIGH : LOW);
    digitalWrite(BIT2, (val/10) & B00000100 ? HIGH : LOW);
    digitalWrite(BIT3, (val/10) & B00001000 ? HIGH : LOW);
    latch(LATCH_G); // validation de la valeur du chiffre de gauche
  }
  // exemple avec 54
  // 54 / 10 = 5
  // 5 * 10 = 50
  // 54 - 50 = 4
  // 54 - ((54/10) * 10) = 4
  digitalWrite(BIT0, (val - ((val/10) * 10)) & B00000001 ? HIGH : LOW);
  digitalWrite(BIT1, (val - ((val/10) * 10)) & B00000010 ? HIGH : LOW);
  digitalWrite(BIT2, (val - ((val/10) * 10)) & B00000100 ? HIGH : LOW);
  digitalWrite(BIT3, (val - ((val/10) * 10)) & B00001000 ? HIGH : LOW);
  latch(LATCH_D); // validation de la valeur du chiffre de droite
}
```

```
void setup() {
  // Tous les ports en sortie
  pinMode(BIT0, OUTPUT);
  pinMode(BIT1, OUTPUT);
  pinMode(BIT2, OUTPUT);
  pinMode(BIT3, OUTPUT);
  pinMode(LATCH_G, OUTPUT);
  pinMode(LATCH_D, OUTPUT);
  // Et on envoie "00" à l'afficheur
  setdigits(0);
}

// Structure contenant une date/heure complète
// Ici nous avons le 1er février 2016 à 00:00:00
tmElements_t hackable11 = {
  0,         // secondes
  0,         // minutes
  0,         // heures
  2,         // jour de la semaine (dimanche = 1)
  1+1,       // jour (le lendemain, la veille il reste 1 jour)
  2,         // mois
  2016-1970 // années depuis 1970
};

void loop() {
  // Structure pour contenir la date/heure depuis le DS1307
  tmElements_t tm;
  // Variable pour nos calculs, time_t est un unsigned long
  // défini dans Time.h
  time_t maintenant, prochain, diff;

  // Peut-on lire la date/heure ?
  if (RTC.read(tm)) {
    // oui !
    // On transforme les dates en nombre de secondes depuis
    // le 1er janvier 1970 (début de l'époque)
    maintenant = makeTime(tm);
    prochain = makeTime(hackable11);
    // On calcule la différence, c'est le nombre de secondes
    // entre maintenant et notre date/heure future
    diff = prochain - maintenant;
    // On divise le résultat par une macro qui est dans Time.h
    // et est 60*60*24 (60 secondes par minute, 60 minutes par heure,
    // et 24 heures par jour
    // Et on envoie cela à l'afficheur via notre fonction
    setdigits(diff / SECS_PER_DAY);
  } else {
    // la date n'a pas été lue
    if (RTC.chipPresent()) {
      // et le module n'est pas détecté
    } else {
      // le module est détecté, mais la date n'est pas valide
      // l'heure n'a sans doute pas été réglée
    }
  }
  // on boucle toutes les 5 secondes
  delay(5000);
}
```

Arduino

# À PROPOS DU CROQUIS

Le croquis est relativement simple et principalement composé d'opérations binaires et de calculs sur des dates/heures. L'ensemble pourrait être largement optimisé, mais le code perdrait alors en lisibilité, ce qui va tout simplement à l'encontre de l'objectif même du magazine. Nous gardons donc ici les choses délibérément et relativement simplistes.

Notre but est de changer l'état des ports 7 à 4, correspondant aux 4 bits attendus par le 4511, en fonction de la valeur d'une variable entière, elle-même pouvant naturellement s'exprimer en binaire. Ainsi, pour une valeur inférieure à 10, tout ce que nous avons à faire est de tester les bits qui nous intéressent. Afin de garder une certaine concision, nous utilisons l'opérateur ternaire directement dans les arguments du `digitalWrite()` selon la syntaxe : `condition ? si vrai : si faux`.

Pour tester le bit à la position zéro, le plus à gauche, nous utilisons un ET logique (`&`) : `valeur & masque = résultat`. 3, par exemple, s'écrit 11 en binaire. L'opération `3 & B00000001` retournera une valeur non nulle (donc VRAI) et la condition sera vérifiée. C'est également valable pour `1 & B00000001`, mais aussi `5 & B00000001` et `9 & B00000001`. Ce n'est pas un problème, bien au contraire, puisque dans les 4 cas, ce bit sera 1 sur la valeur que nous devons passer au 4511.

En combinant ce test avec l'opérateur ternaire, nous pouvons intégrer tout cela en une seule instruction, comme ici sur le premier bit du chiffre de gauche :

```
digitalWrite(BIT0, (val/10) & B00000001 ? HIGH : LOW);
```

C'est beaucoup plus succinct que :

```
if((val/10) & B00000001) {
  digitalWrite(BIT0, HIGH);
} else {
  digitalWrite(BIT0, LOW);
}
```

Nous n'avons plus qu'à distinguer le chiffre de gauche de celui de droite pour une valeur donnée. Pour celui de gauche, c'est simple : si la valeur est inférieure à 10, ce chiffre sera zéro, sinon nous divisons la valeur par 10 pour obtenir le nombre de dizaines et testons les bits du résultat. Pour le chiffre de droite, cela demande un peu plus de calcul, mais reste du niveau d'une classe de primaire. La seule chose à bien comprendre est le fait que nous travaillons ici avec des valeurs entières et que cela ne fonctionne qu'à cette condition. Exemple avec la valeur 54 :

- 54 / 10 = 5 ;
- 5 * 10 = 50 ;
- 54 - 50 = 4 ;
- donc 54 - ((54/10) * 10) = 4.

À aucun moment le croquis ne traite de valeur à virgule flottante, 54/10 donne 5, pas 5,4. En combinant cette opération avec le test binaire et l'opérateur ternaire, nous obtenons : `digitalWrite(BIT0, (val - ((val/10) * 10)) & B00000001 ? HIGH : LOW)`.

La seconde partie intéressante du croquis concerne le traitement de la date/heure. La valeur obtenue via la méthode `read()` permet de remplir une structure `tm` de type `tmElements_t` définie dans `Time.h` présent dans la bibliothèque du même nom (et installable tout comme *DS1307RTC* via le gestionnaire de bibliothèques) :

```
typedef struct  {
  uint8_t Second;
  uint8_t Minute;
  uint8_t Hour;
  uint8_t Wday;   // day of week, sunday is day 1
  uint8_t Day;
  uint8_t Month;
  uint8_t Year;   // offset from 1970;
}       tmElements_t, TimeElements, *tmElementsPtr_t;
```

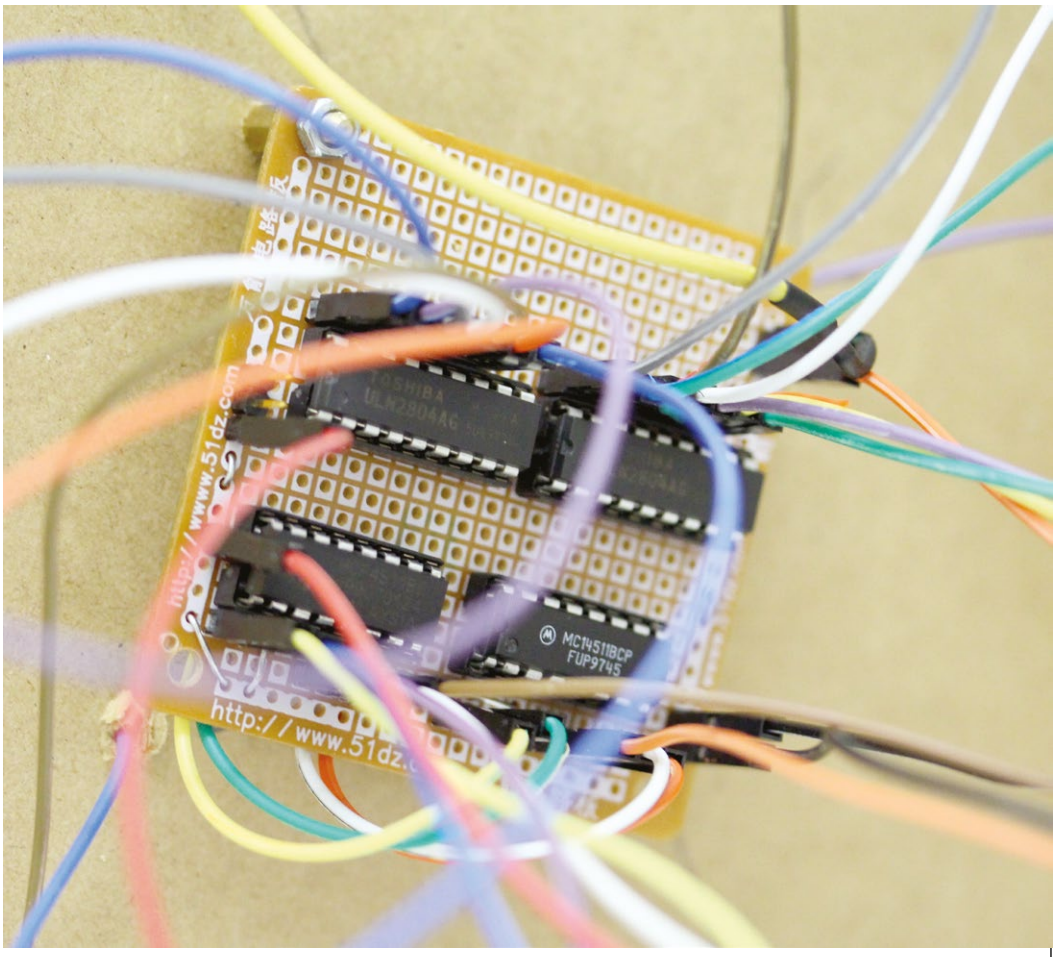
On trouve ici tous les éléments généralement utilisés sous forme de valeurs entières sur 8 bits. C'est très pratique pour afficher les données dans le moniteur série ou sur un écran LCD, mais dans notre cas c'est un peu pénible. La bibliothèque *Time* heureusement, met à notre disposition une fonction `makeTime()` prenant en argument une telle structure et retournant le nombre de secondes écoulées depuis de début de l'époque, le 1er janvier 1970 à minuit précise. Ce n'est pas une nouvelle forme de créationnisme, mais la représentation d'une date également appelée heure UNIX, heure POSIX ou *POSIX timestamp*. Son principal avantage est de pouvoir tenir dans une simple variable `unsigned long int` de 32 bits (oui, un problème se posera en 2107, mais, dans 91 ans j'en aurai 132 et aurai certainement d'autres soucis en tête, comme expliquer à mes arrières-petits-enfants ce qu'est un téléphone, une boussole, une carte ou un dictionnaire).

Pour calculer la différence en jours entre maintenant et une date dans le futur, tout ce que nous avons à faire est donc de convertir les deux informations en heure UNIX, faire une soustraction et diviser le tout par le nombre de secondes dans un jour, gracieusement fourni par la bibliothèque *Time* sous la forme d'une macro `SECS_PER_DAY`.

Notez la petite subtilité consistant à fixer une date future plus un jour. C'est une question de préférences, mais je pars du principe que la veille de la date butoir, même si la journée est entamée, il me reste du temps. Je préfère donc que l'afficheur m'indique « 00 » le jour même et non la veille.

# CONCLUSION

J'ai à présent mon générateur de stress personnel, mais celui-ci nécessite encore un peu de travail : remplacer la carte Arduino par un microcontrôleur Atmel AVR seul (moins cher), finaliser le support et le cadre, donner un coup de peinture, délimiter proprement les segments, ajouter une plaque de Plexiglass translucide de couleur (**plexiglas-shop.com** est absolument génial pour cela) et surtout l'accrocher bien en vue.



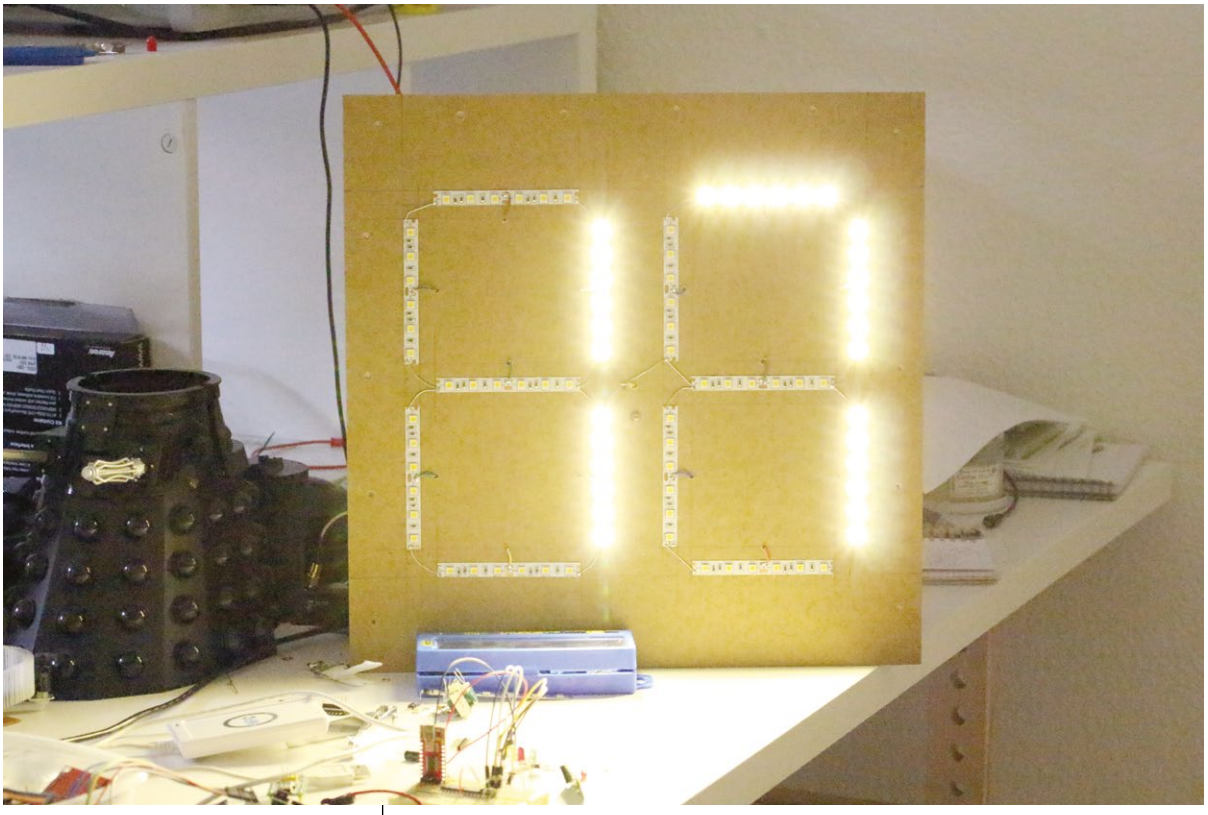

*L'assemblage du circuit sur platine à essais est très pratique pour valider le fonctionnement, mais en aucun cas quelque chose de viable dans le temps. La plaque pastillée ou la réalisation d'un circuit imprimé s'avère donc indispensable pour finaliser le projet.*

Mais je crois que le plus important n'est pas le projet lui-même, car je pense que peu de gens apprécient le fait de s'imposer ce genre de pression (Comment ça « masochiste » ? C'est un dispositif anti-procrastination !). Non, le plus intéressant est, je pense, le caractère ultra-polyvalent d'un tel système d'affichage. Jugez plutôt :

- supprimez le module RTC et connectez un capteur de température : vous avez un thermomètre géant ;
- laissez le module, révisez le code pour compter des secondes ou des minutes : vous obtenez un compte à rebours ;
- supprimez le module, ajoutez des boutons ou des capteurs : vous voici propriétaire d'un compteur (score, mouvement, numéro de client, etc.) ;
- supprimez le module, ajoutez un bouton, révisez le code : voilà un générateur de nombres aléatoires pour un loto ;
- etc.

Mais ce n'est pas tout, chaque sortie d'un ULN2803A peut supporter jusqu'à 500 mA et 50 V (attention tout de même, ce composant peut devenir très chaud dans certaines conditions et nécessitera un refroidissement adapté). Non seulement cela vous permet de contrôler une quantité assez impressionnante de leds, mais vous n'êtes absolument pas obligé de vous en tenir à cela... Les ULN2803A disposent de protections intégrées pour l'utilisation avec des relais. Remplacez les morceaux de rubans de leds par des relais pilotant... disons... des tubes fluorescents avec leur ballast électronique. Que diriez-vous de segments de 1,20 m ?

# (DÉ)COMPTEUR GÉANT



*Ceci est un résultat intermédiaire permettant de valider le concept et le montage avant de compléter la réalisation avec un cadre en bois à l'arrière pour dissimuler et protéger l'électronique, mais également à l'avant, pour pouvoir ajouter une plaque de Plexiglass translucide.*

Ce n'est pas assez original ? Pourquoi pas oublier totalement les lumières et contrôler des actionneurs ? Des segments constitués de cylindres de bois peints de chaque côté d'une couleur précise et des moteurs pour les faire tourner de 180°. Non ? Toujours trop « normal » ? Pourquoi pas, dans ce cas, des réservoirs cylindriques transparents qui se remplissent et se vident d'un liquide coloré, le tout grâce à des valves ou pompes pilotées par les sorties des ULN2803A ?

Vous voyez, il n'y a qu'à laisser faire son imagination pour voir fleurir les idées de projets sur la simple base de quelques composants, existant depuis très longtemps et toujours aussi utiles et amusants...

Une dernière remarque : ce dont nous venons de parler n'est ni nouveau, ni possible uniquement grâce aux cartes comme l'Arduino. De simples circuits logiques permettent de créer un tel compteur (NE555, 4040, 4060, etc.) et les amateurs d'électronique le font depuis des décennies, tout comme je l'ai fait il y a plus d'une quinzaine d'années. Le fait est cependant que les prix des microcontrôleurs comme les AVR d'Atmel ont totalement changé la donne dans ce domaine, tout en apportant plus de souplesse et de généricité. Il n'en reste pas moins que la construction d'un tel projet à base de circuits logiques reste un excellent exercice... **DB**

# DISPONIBLE DÈS LE 11 MARS

## LINUX PRATIQUE HORS-SÉRIE N°35 !




CRÉEZ UN BLOG WORDPRESS QUI VOUS RESSEMBLE !

Sous réserve de toutes modifications

## NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

**www.ed-diamond.com**



# UTILISEZ DES LEDS COMME CAPTEURS DE LUMIÈRE

*Denis Bodor*

**Dans le numéro précédent, Jonathan Piat nous exposait son système de communication par lumière visible utilisant des simples leds et deux cartes Arduino. Vous avez été nombreux à nous demander un peu plus d'information sur cette technique et en particulier sur l'utilisation de leds en guise de photodiodes ou, en d'autres termes, comment utiliser des leds comme des détecteurs de lumière...**

Les diodes électroluminescentes ou leds (DEL en bon français) sont, comme leur nom l'indique, des diodes émettant de la lumière lorsqu'un courant, fixé par le constructeur les traverse. Comme il s'agit de diodes, celles-ci sont polarisées et ne laissent passer le courant que dans un sens. En inversant la polarité, elles bloquent le passage du courant et ne s'allument pas. C'est exactement le fonctionnement normal du composant semi-conducteur qu'est une diode standard (la lumière en moins).

Mais ce n'est pas tout, utilisée dans ce sens, une diode affiche une capacité, comme un condensateur polarisé. En d'autres termes, une diode connectée à l'envers va accumuler un peu d'énergie et pouvoir la restituer par la suite. Ceci est tantôt spécifié dans les documentations techniques (*datasheet*) sous le nom *junction capacitance* et se mesure en quelques dizaines de picofarads. C'est peu, mais suffisant pour en obtenir quelque chose d'utilisable. D'autant plus que la vitesse à laquelle cette énergie va quitter le composant dépend de... son exposition à la lumière.

Il est donc possible, en jonglant un peu, d'utiliser une led comme détecteur de lumière. La technique est la suivante :

- appliquer une tension aux bornes du composant en reliant la tension positive à la cathode (-) de la led et la référence (la masse) à l'anode (+). La led est donc branchée à l'envers ;
- déconnecter la cathode et mesurer la variation de tension ;
- déduire de la vitesse à laquelle la tension chute l'exposition de la led à la lumière.

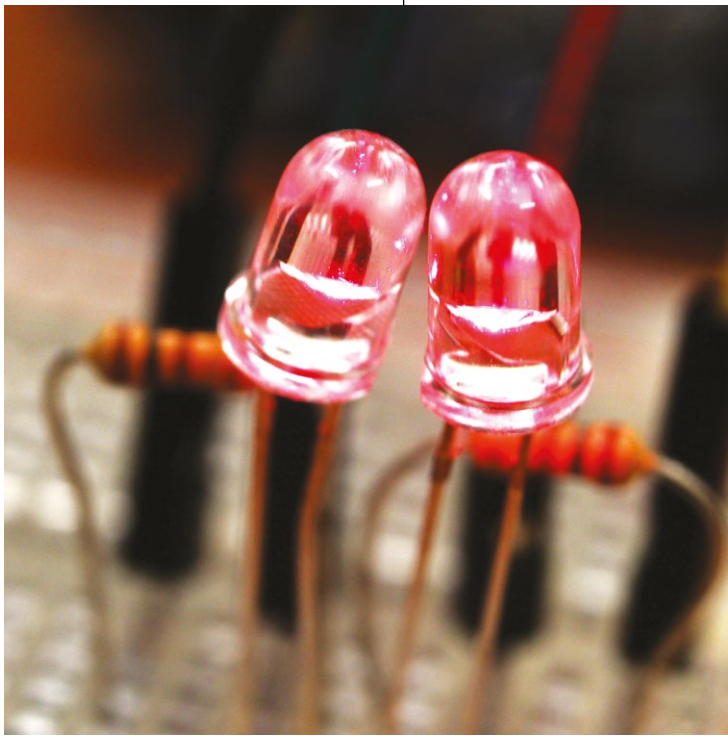
Avec une carte Arduino, faire cela est relativement facile puisque les différentes broches de la carte peuvent être configurées en sortie comme en entrée, et ce avec ou sans résistance de rappel. En entrée et sans résistance de rappel, une broche est dite « en haute impédance » et n'utilise donc que très très peu de courant pour changer d'état (vous pouvez imaginer l'équivalent d'une résistance de quelques 100 mégaohms en série sur une telle broche). Le courant s'échappant de la led va donc mettre un temps parfaitement mesurable pour circuler et la tension pour chuter.

Deux solutions s'offrent alors à vous :

- chronométrer le temps nécessaire pour que l'entrée soit lue comme étant à l'état bas (0 volts) ;
- ou mesurer la tension après une durée arbitrairement fixée.

Dans les deux cas, si la led est exposée à la lumière, soit le temps sera plus court, soit la tension lue sera plus basse à l'instant *t*.

Il faut cependant garder à l'esprit que nous utilisons ici une caractéristique propre aux leds, mais que le composant

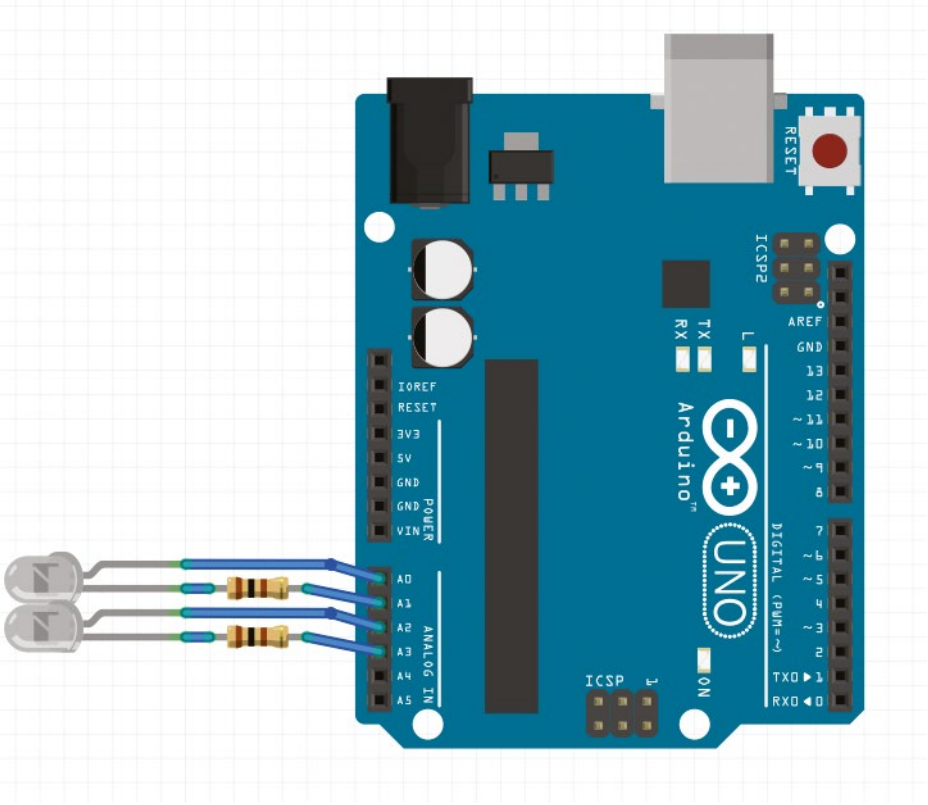


*À l'œil nu, les deux leds semblent utilisées dans un montage parfaitement standard, même s'il y a un léger scintillement en raison de l'alternance entre les rôles d'éclairage et de détection. Pourtant, l'une de ces leds est en train d'être utilisée pour mesurer la lumière ambiante.*

n'est, bien entendu, pas prévu pour une telle utilisation. Pour ça il existe des photodiodes, fonctionnant sur le même principe, mais spécialement conçues pour cet usage. Il faut également noter que certaines leds sont plus adaptées que d'autres pour ce type de projet. Il faudra privilégier les boîtiers transparents (cristal) au détriment des leds colorées émettant une lumière diffuse. Mes différentes expériences montrent que les leds rouges ou vertes fonctionnent particulièrement bien alors que les blanches sont peu réactives. Sans doute parce que le blanc n'est pas une couleur qu'une led peut émettre, mais qu'il s'agit en réalité de leds ultraviolettes intégrant un revêtement phosphoré qui, stimulé par un rayonnement UV, produit de la lumière blanche.

## 1. LE MONTAGE ET SON PRINCIPE

En suivant le principe décrit en début d'article, n'importe quelle led (transparente, rouge ou verte) sera en mesure de vous donner une indication de son exposition à la lumière. Nous allons cependant pousser un peu plus loin en utilisant deux leds de façon alternative : une led sert de capteur et l'autre émet de la lumière. Comme les leds sont généralement plus sensibles à la longueur d'onde de la lumière qu'elles émettent, ceci nous permet d'éclairer un « obstacle » qui, en fonction de la proximité, retournera la lumière « idéale ».




Alterner n'est pas nécessaire en soi, car avec une led qui éclaire et une led qui capte, cela fonctionnera parfaitement... Mais qui peut résister à l'envie de faire scintiller le tout pour ajouter une touche de confusion ?

Le montage est relativement simple puisqu'il est constitué de deux leds et leurs résistances, qu'on choisira d'une valeur satisfaisant deux critères : limiter le courant lorsqu'on allume une led et obtenir une valeur utilisable lorsqu'on lit la tension au moment de la détection. Ici, j'ai choisi des résistances de 100 ohms, laissant donc circuler environ 28 mA avec une tension de 2,2V aux bornes d'une led rouge de forte luminosité.

## 2. UTILISATION ET CROQUIS




```
#define LED1_P A0
#define LED1_N A1
#define LED2_P A2
#define LED2_N A3

#define PAUSE 20

void setup() {
  // Le convertisseur analogique/numérique
  // utilise les 1,1 volts comme référence
  analogReference(INTERNAL);

  // moniteur série
  Serial.begin(115200);

  // broches led 1 en sortie
  pinMode(LED1_N, OUTPUT);
  pinMode(LED1_P, OUTPUT);

  // broches led 2 en sortie
  pinMode(LED2_N, OUTPUT);
  pinMode(LED2_P, OUTPUT);
}

void loop() {
  int val=0;

  // allumage de la led 2
  pinMode(LED2_N, OUTPUT);
  digitalWrite(LED2_N, LOW);
  digitalWrite(LED2_P, HIGH);

  // On "charge" la led
  digitalWrite(LED1_N, HIGH);
  digitalWrite(LED1_P, LOW);
  // cathode de la led en entrée
  pinMode(LED1_N, INPUT);
  // désactivation de la résistance de rappel
  digitalWrite(LED1_N, LOW);
  // on attend un peu
```

```
  delay(PAUSE);
  // mesure de la tension sur la cathode
  val=analogRead(LED1_N);

  // On inverse les rôles des leds
  // et on fait de même
  pinMode(LED1_N, OUTPUT);
  digitalWrite(LED1_N, LOW);
  digitalWrite(LED1_P, HIGH);
  digitalWrite(LED2_N, HIGH);
  digitalWrite(LED2_P, LOW);
  pinMode(LED2_N, INPUT);
  digitalWrite(LED2_N, LOW);
  delay(PAUSE);
  val+=analogRead(LED2_N);

  // val contient la somme des valeurs lues
  // du convertisseur pour led 1 et led 2
  Serial.println(val);
}
```

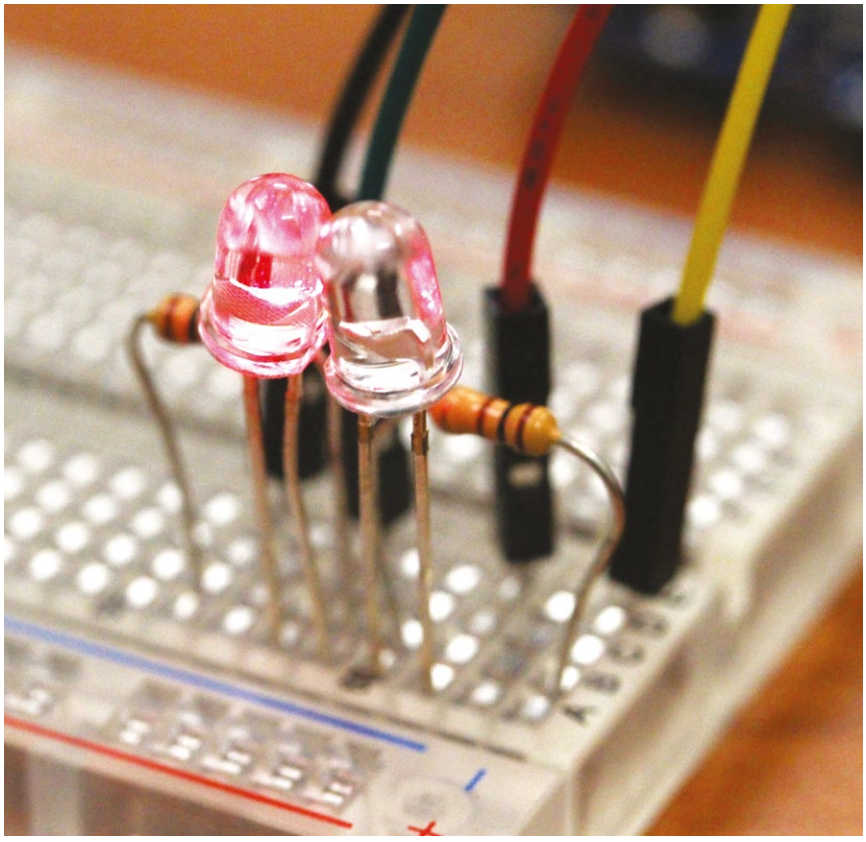
Arduino

Le principe de fonctionnement du code est celui décrit précédemment. À chaque tour de `loop()` on commence par allumer l'une des leds tout en alimentant la seconde en inversant la polarité. Ceci a pour effet de « charger » le composant.

Immédiatement après, on passe le port sur lequel est connectée la cathode de la led en entrée sans oublier de désactiver la résistance de rappel à la tension d'alimentation. À ce moment précis, l'entrée est en haute impédance et l'énergie stockée dans la led se met à circuler. En même temps, la tension chute. Après 20 ms, nous procédons à une lecture de la valeur. Puis nous réitérons toute l'opération en inversant les rôles des deux leds.

Au terme de l'exécution de la fonction, nous obtenons dans `val` la somme des deux valeurs ainsi mesurées et nous l'affichons sur le moniteur série. La valeur maximum lue sera donc 2048, en principe, dans l'obscurité totale et sans réflexion de la lumière. À l'approche d'un obstacle à quelques centimètres, la cathode de la led en réception verra sa tension chuter plus rapidement. Avec un délai de 20 ms, `val` peut descendre jusqu'à 0.

Les résultats sont très variables et fonctions du type de led, du délai utilisé, de la luminosité ambiante, mais également de la quantité de lumière retournée par l'objet placé devant les leds ainsi que de l'orientation des composants.



*Sur ce cliché avec un temps d'exposition plus court, on distingue clairement le moment précis où la led de gauche se charge de l'éclairage de l'environnement et celle de droite en phase de détection. Dans quelques 20 ms, les rôles seront inversés...*

## 3. CONCLUSION, LIMITATIONS ET PERSPECTIVES

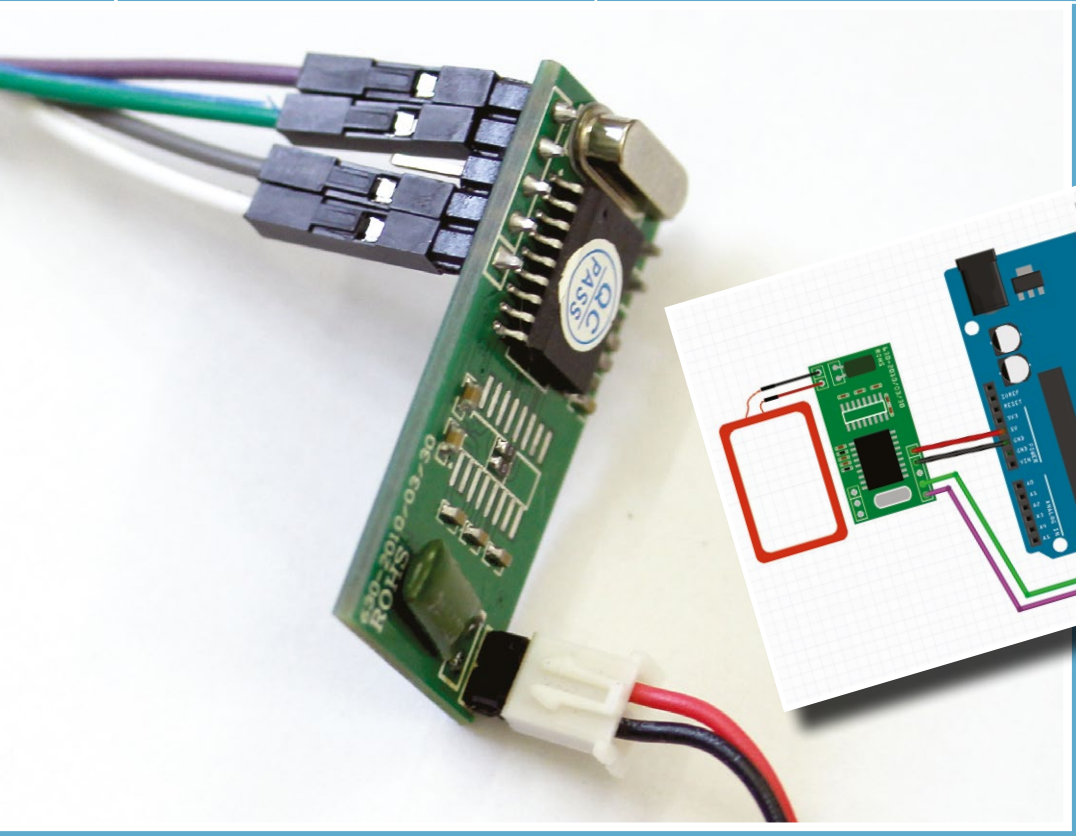
Les résultats sont là. En deux mots, ça fonctionne !

Il ne faut pas perdre de vue cependant que nous utilisons ici un composant pour autre chose que ce pour quoi il est prévu. Vous ne devez donc pas vous attendre à un comportement idéal et surtout pas à quelque chose d'équivalent à une photodiode ou même une simple photorésistance (LDR). La partie délicate de l'opération consiste à travailler « dans le flou ». Sans données constructeur, il n'est possible de calibrer le montage que par l'expérimentation.

Cela constitue toutefois une solution de fortune pour un dispositif ayant besoin d'obtenir l'information souhaitée, mais la véritable raison de parler de ce montage est tout autre : ici nous utilisons la mesure de la capacité de la led connectée à l'envers, c'est donc une détection capacitive. De ce fait, les explications et le croquis proposé pourront fonctionner avec n'importe quel composant capable de stocker une charge, comme... un humain par exemple.

Il est possible d'utiliser la même technique, ou presque, avec un simple connecteur métallique pour détecter un contact avec un doigt. Votre corps est en effet un gros condensateur et c'est de cette manière que les écrans tactiles capacitifs fonctionnent. Les problèmes de stabilité en revanche sont non négligeables puisque le simple fait d'utiliser les ports à proximité de celui mesuré suffit à perturber le montage et donner des résultats presque aléatoires.

Des circuits intégrés spécialisés existent pour ce genre de choses, prenant en compte l'ensemble des paramètres adéquats. C'est le cas par exemple de la technologie QTouch d'Atmel, fabricant du microcontrôleur équipant les cartes Arduino. Celle-ci est disponible sous forme de composants dédiés et est également intégrée dans certains microcontrôleurs. C'est également le cas pour les technologies équivalentes pour d'autres constructeurs. **DB**

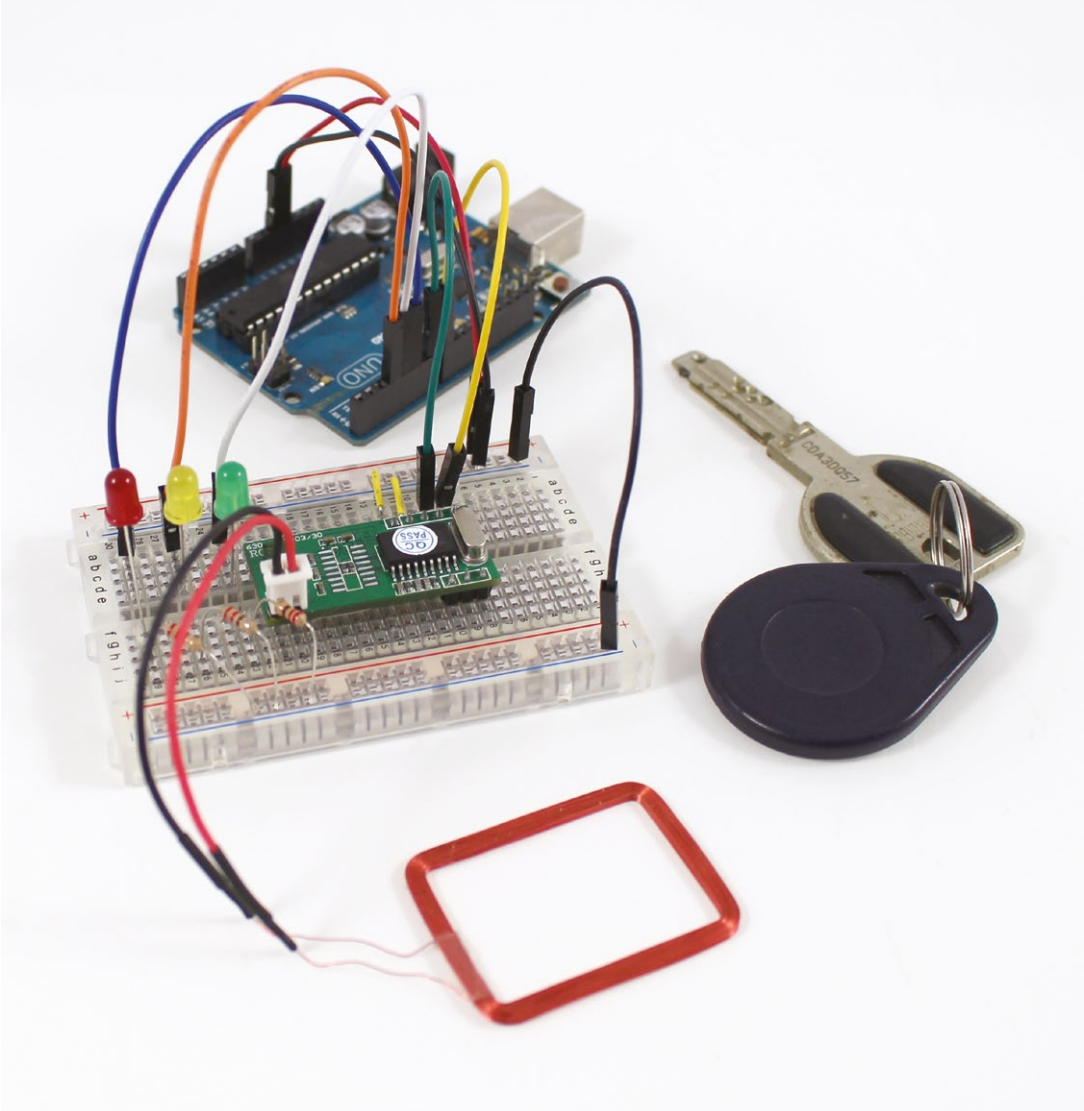


# CRÉEZ UNE SERRURE À CARTE/TAG RFID

*Denis Bodor*

Dans le précédent numéro, nous avons exploré les technologies NFC, relativement modernes. Mais l'utilisation du RFID est bien plus ancienne et toujours envisageable. Voyons ici ce qui peut être vu comme la solution la plus simple pour, par exemple « sécuriser » une entrée avec un lecteur de cartes/tags RFID économique. La technologie RFID utilisée ici repose sur les tags 125Khz, également appelés EM4100/EM4102, non inscriptibles, contenant simplement un numéro qu'il vous est possible de lire. Il ne s'agit pas de NFC, mais c'est une solution économique et simple, offrant le même niveau de sécurité qu'un digicode.

| NIVEAU | TEMPS | BUDGET |
|---|---|---|
| | 10 minutes | 40 € (Arduino inclus) |



Cette dernière affirmation est importante, car le caractère hollywoodien d'une porte verrouillée par un lecteur de carte pourrait laisser penser que la sécurité est renforcée. Ce n'est pas le cas ! La sécurité offerte est, tout au plus, équivalente à un code ou une clé standard, rien de mieux. Vous me direz dans les films et séries américaines il y a aussi des cartes magnétiques et des bombes avec des gros afficheurs rouges (du même genre que certains gamins de 14 ans récupèrent dans des réveils et placent dans des boites suspicieuses en clamant qu'ils ont « inventé » quelque chose).

Nous parlerons ici de cartes et de tags très courants qu'on peut trouver en quantité à des prix défiant toute concurrence. Une simple recherche sur eBay et vous trouverez, pour quelques 20€, des lots d'une centaine de cartes ou de porte-clés décrits comme « RFID tags 125Khz » ou « Badge de proximité 125Khz ». Ces tags, souvent désignés par le terme EM4100, contiennent une valeur numérique soi-disant unique, leur ID, qu'ils transmettent dès qu'un lecteur adapté le leur demande. Soi-disant, car la grande majorité des tags distribués aujourd'hui sont des clones du produit original d'EM Microelectronic qui a plus de 10 ans. Nous avons donc affaire à une sorte de prolifération incontrôlée où il est bien difficile de garantir l'unicité des identifiants.

Enfin, tout comme un digicode peut être lu et mémorisé, et une clé standard peut être copiée, ces tags sont

facilement falsifiables. Micah Elizabeth Scott en a fait la démonstration de façon fort élégante en 2008 en utilisant un simple microcontrôleur ATTiny85 et une bobine. Ces deux éléments combinés (rien de plus) et un peu de code assembleur permettaient de reproduire n'importe quel tag RFID de type EM4100 ou HID. Trammell Hudson a repris le concept en 2012 pour l'améliorer et le rendre plus générique.

# LE PRINCIPE

Les tags RFID EM4100 ou EM4102 (identiques en termes de fonctionnement) sont relativement simples. Placés dans le champ du lecteur, ceux-ci « hurlent » leur identifiant à tout va. Le lecteur récupère l'information et la transmet au système auquel il est connecté.

Cette simplicité qui va de paire avec une grande robustesse fait que les tags RFID 125 Khz EM4100 sont encore utilisés pour bien des usages : parking privé, entrées d'entreprise, système de pointage, machines à café, etc.

Notre idée est de faire de même, mais plutôt que d'acheter un ensemble clé en main permettant de contrôler la gâche électrique d'une porte ou d'un portail, nous allons construire un tel système sur la base d'une carte Arduino. Ceci nous permettra non seulement de contrôler la fermeture d'une porte mais, par conséquent, d'adapter le système à tout ce qui nous passe par la tête.

La philosophie du projet consiste à lire les tags présentés, obtenir leur identifiant (ID), valider quelques éléments techniques de base et vérifier leur présence dans une liste de tags autorisés. En fonction de cette vérification, on pourra alors faire réagir le montage positivement ou négativement. Pour les besoins de la démonstration, il s'agira d'allumer une led verte en cas de succès et rouge en cas d'échec, mais libre à vous de contrôler ce qui vous chante avec les sorties de l'Arduino. Ceci peut aller de la classique gâche électrique, au distributeur de nourriture pour un animal de compagnie en passant par les énigmes de géocaching (mettre un tag dans une cache et devoir le lire à un autre endroit où s'affichent les coordonnées de la prochaine cache par exemple).

# CE QU'IL VOUS FAUT

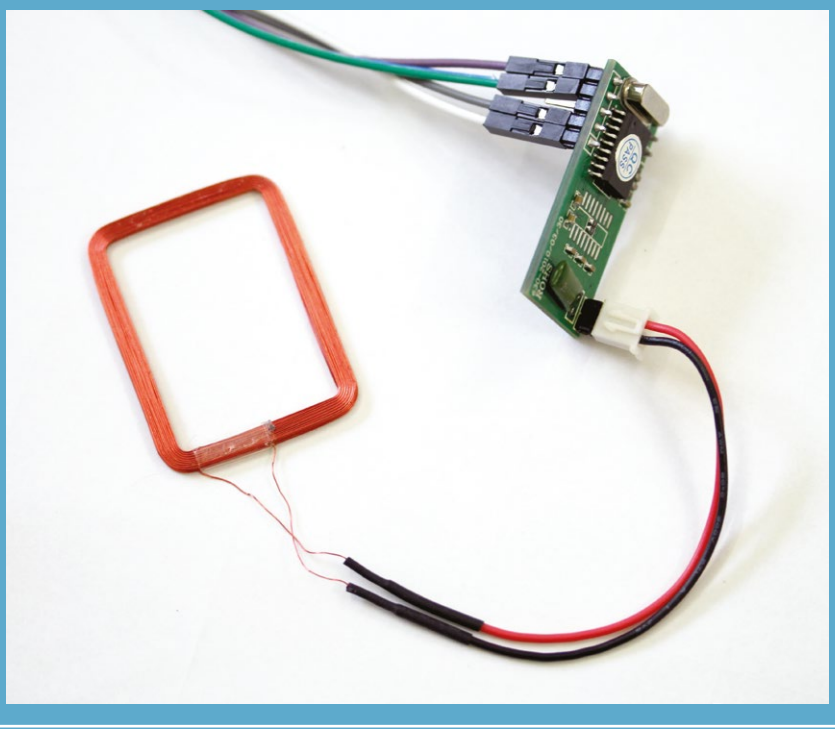
- **Une carte Arduino : n'importe quelle carte fera l'affaire en fonction du module lecteur utilisé. Les seules restrictions qui s'imposent à nous concernent le mode de connexion et les tensions en présence. La totalité des modules que j'ai testés fonctionne avec une liaison série et 90% d'entre eux en 5 volts. La communication se faisant la plupart du temps en 9600 bps (plus rarement en 19200), une liaison série logicielle (*SoftSerial*) pourra être utilisée et il ne sera donc pas nécessaire d'opter pour une carte Arduino avec plus d'un port de ce type. Une classique Uno, par exemple, se prêtera parfaitement à ce projet.**

- **Un module lecteur RFID EM4100 125 Khz : j'ai utilisé un vieux module de chez SeeedStudio (toujours en vente) pour cette réalisation, mais la plupart des composants existants feront parfaitement l'affaire. Pour une poignée d'euros, vous disposerez d'une bobine attachée à un petit circuit alimenté en 5 volts et proposant un ensemble de broches. La communication se fait en liaison série,**

**généralement à sens unique, du module vers le microcontrôleur (carte Arduino ici). Ces modules se composent généralement d'un circuit intégré chargé de la partie analogique (gestion des fréquences et modulation) et d'un microcontrôleur pour la mise en forme des données. Avant achat, assurez-vous que le module communique bien l'identifiant du tag en liaison série dite « TTL RS232 » (et non uniquement en Wiegand ou codage Manchester). Ce type de module est tantôt désigné sous le nom RDM630. Évitez absolument les lecteurs USB qui se présentent au PC comme un clavier USB, ceux-ci sont presque impossibles à utiliser avec une carte Arduino (sauf la Arduino Mega ADK proposant une fonctionnalité d'hôte USB).**

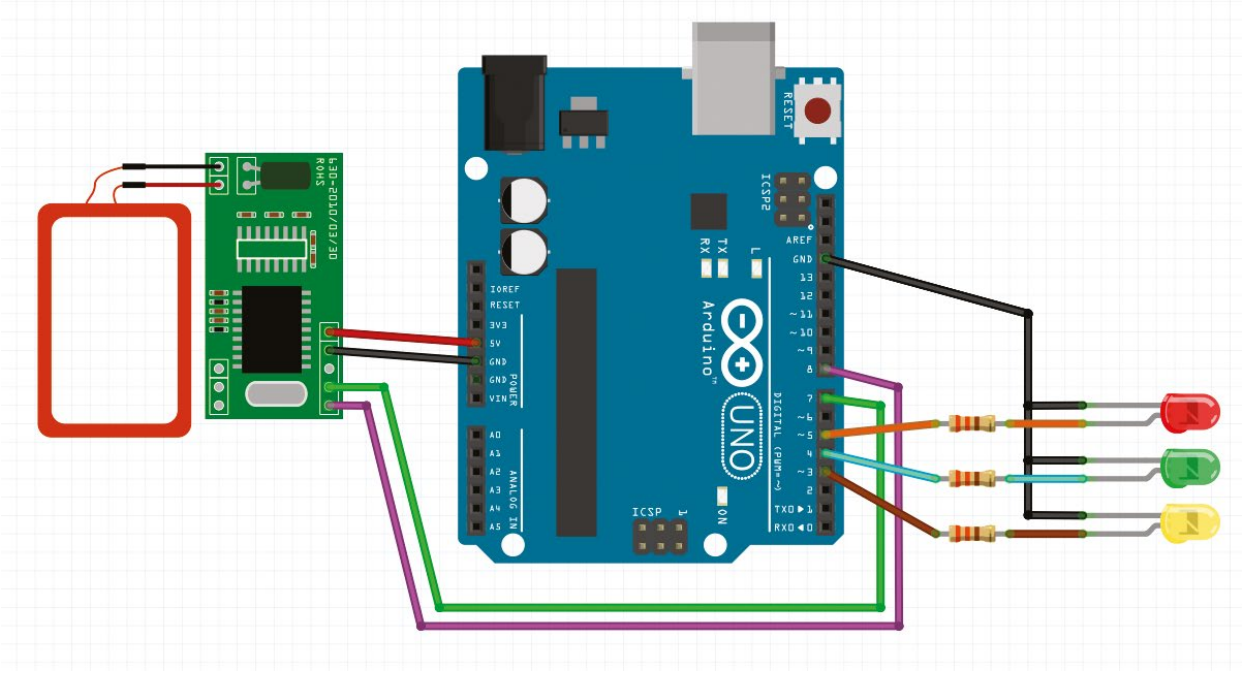
- **2 tags au minimum : tantôt les modules en vente sont accompagnés d'un tag, mais le plus souvent ils sont livrés seuls. Nous aurons besoin de deux tags, l'un valide l'autre non, au moins pour vérifier notre croquis. Prenez votre temps pour lister toutes les offres si vous passez par eBay, certains vendeurs proposent 5 tags à 25€ (!), d'autres 100 pour 16€+5€ de port, et d'autres encore 10 pour 3€ ou 100 pour 63€ (mais pour des tags transparents et ça c'est cool !). Il en existe de toutes les formes (porte-clés, carte, disque, etc.) et de toutes les couleurs. Faites le tour des offres, vérifiez les évaluations et prenez en compte les frais de port.**

- **Quelques câbles, résistances 330 Ohms et leds : rien de bien extraordinaire ici, nous devons simplement connecter le module lecteur à la carte Arduino (masse, alimentation, TX). Les résistances et les leds sont presque optionnelles, mais très agréables pour mettre au point votre code. Les autres composants annexes dépendront de votre objectif final : relais, MOSFET, écran LCD, Oled, buzzer...**



*Le module de lecture est un assemblage relativement simple : une bobine, un condensateur, un microcontrôleur et son quartz, et quelques composants passifs. Il n'en faut pas plus pour lire un tag RFID EM4100.*

# LE MONTAGE




# LE CROQUIS

Fichier Édition Croquis Outils Aide

```
#include <SoftwareSerial.h>

// Quelques leds
#define LROUGE 5
#define LVERTE 4
#define LJAUNE 3

// Liaison série logiciel sur broche 8 et 7 (RX, TX)
SoftwareSerial sSerial(8,7);

// Après une lecture, on ne traite plus
// les messages pendant PAUSE millisecondes
```

```
int pret = 1;
unsigned long previousMillis = 0;
// Temps de pause après lecture en 1/1000s
#define PAUSE 5000

// liste d'IDs acceptés
const unsigned long long auth[] = {
  0x0000002C00827F4A,
  0x0000030405060708,
  0x000000bbccddeeff,
  0x000000ff00042154,
  0x0000003344556677};

// fonction pour obtenir un octet depuis
// deux caractères hexa
byte hex2int(char haut, char bas) {
  byte ret=0;

  ret |= haut < 58 ? (haut-48)<<4 : (haut-55)<<4;
  ret |= bas < 58 ? bas-48 : bas-55;

  return ret;
}

// Fonction de validation de l'ID
// retourne "vrai" si ID dans la liste
bool valide(unsigned long long ID) {
  // on parcourt la liste (tableau)
  for(int i=0; i<(sizeof(auth)/sizeof(*auth)); i++) {
      // on retourne avec vrai si on trouve
      if(auth[i] == ID){ return true; }
    }
  // on a pas trouvé on retourne avec faux
  return false;
}

void setup() {
  // moniteur en 115200
  Serial.begin(115200);
  Serial.println("**** CONTROLE D'ACCES RFID ****");

  // softserial en 9600
  sSerial.begin(9600);

  // configuration leds
  pinMode(LROUGE, OUTPUT);
  pinMode(LJAUNE, OUTPUT);
  pinMode(LVERTE, OUTPUT);
}

void loop() {
  // nbr de caractères reçus du lecteur RFID
  int sret=0;
  // tableau pour le message du lecteur
  char cardmsg[14] = {0};
  // tableau pour l'ID du tag
  byte cardID[5] = {0};
  // valeur numérique sur 64 bits
  // (attention long long sur AVR == slow slow)
  unsigned long long numID = 0;

  // variable utilitaire pour les boucles
```

```
int i=0;

unsigned long currentMillis = millis();

// des trucs à lire ?
if(sSerial.available()) {
  // on récupère les caractères dans cardmsg
  // sret contient le nombre de caractères reçus
  sret = sSerial.readBytes(cardmsg, 14);
}

// Une lecture a déjà eu lieu
if(!pret) {
  // cela fait-il plus de PAUSE millisecondes ?
  if(currentMillis - previousMillis >= PAUSE) {
    // oui, on enregistre le moment
    previousMillis = currentMillis;
    // on éteint tout
    digitalWrite(LJAUNE, LOW);
    digitalWrite(LROUGE, LOW);
    digitalWrite(LVERTE, LOW);
    // et on est prêt à gérer les données lues
    pret=1;
  } else {
    return;
  }
}

// A-t-on reçu 14 caractères ?
if(sret == 14) {
  for(int i=0; i<5; i++) {
    // on compose un tableau de 5 valeurs 8 bits
    // à partir des caractères alpha-hexa
    cardID[i] = hex2int(cardmsg[i*2+1],cardmsg[i*2+2]);
    // ainsi qu'une variable numérique
    numID=(numID<<8)|cardID[i];
  }

  // on affiche le résultat
  for(int i=0; i<5; i++) {
    Serial.print(cardID[i],HEX);
    Serial.print(" ");
  }

  // ainsi que la partie du message du lecteur
  // avec l'ID de la carte lue pour comparaison
  Serial.print(": ");
  for(int i=1; i<11; i++) {
    Serial.print(cardmsg[i]);
  }

  // vérification de la somme de contrôle qui
  // accompagne l'ID
  Serial.print("  Checksum (0x");
  // Le checksum est également en alpha-hexa
  Serial.print(hex2int(cardmsg[11],cardmsg[12]),HEX);
  // Le checksum est un XOR des 5 valeurs de l'ID entre elles
  if(hex2int(cardmsg[11],cardmsg[12]) ==
        cardID[0]^cardID[1]^cardID[2]^cardID[3]^cardID[4]) {
    Serial.print(") OK");
  } else {
    Serial.print(") BAD !!");
    return;
  }
```

```
    // saut de ligne, hop !
    Serial.println("");

    // validation de l'accès
    // la valeur long long est-elle dans notre liste ?
    if(valide(numID)) {
      // ok, ID valide, on laisse passer, activons un relais, etc.
      Serial.println("ACCES AUTORISE");
      digitalWrite(LVERTE, HIGH);
    } else {
      // l'ID n'est pas dans la liste, on active les tourelles laser, etc.
      Serial.println("EXTERMINATE ! EXTERMINATE ! EXTERMINAAAAATE ! ");
      digitalWrite(LROUGE, HIGH);
    }

    // pour signaler la pause
    digitalWrite(LJAUNE, HIGH);
    pret=0;
  }
}
```

Arduino

# À PROPOS DU CROQUIS

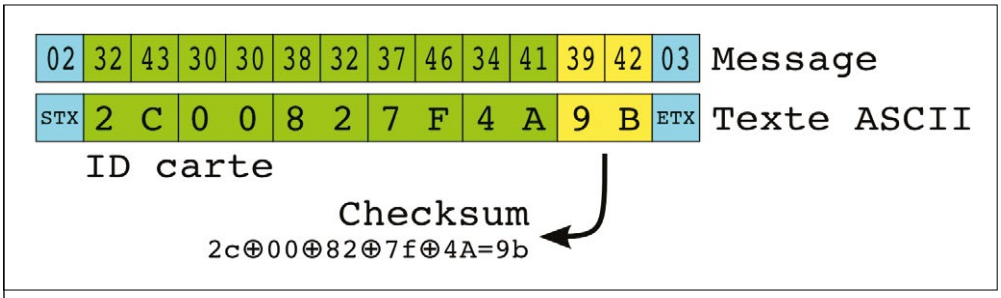
Voilà un gros croquis, ce qui est relativement contre-intuitif étant donné la simplicité du module de lecture. Le problème qui se pose est justement de gérer cette simplicité, car le module envoie des données textuelles et non des valeurs numériques.

## Software serial

Nous réservons le port série de l'Arduino pour le moniteur série de l'IDE. Sur une carte comme la UNO, nous n'avons pas d'autre port série, mais il existe une solution : faire usage d'une bibliothèque qui permet d'utiliser n'importe quelle broche comme un port série. Ceci n'est pas aussi efficace qu'une véritable interface (UART) et occupe de la mémoire, mais pour notre usage cela convient très bien. Pour l'utiliser, tout ce que nous avons à faire est d'ajouter la bibliothèque dans notre croquis et en déclarant `SoftwareSerial sSerial(8,7)`. Les arguments correspondent respectivement aux broches utilisées pour la réception et l'émission de données. Dès lors, `sSerial` pourra être manipulé comme `Serial`, à commencer par la configuration de la vitesse avec la méthode `begin()`.

## Du texte au numérique

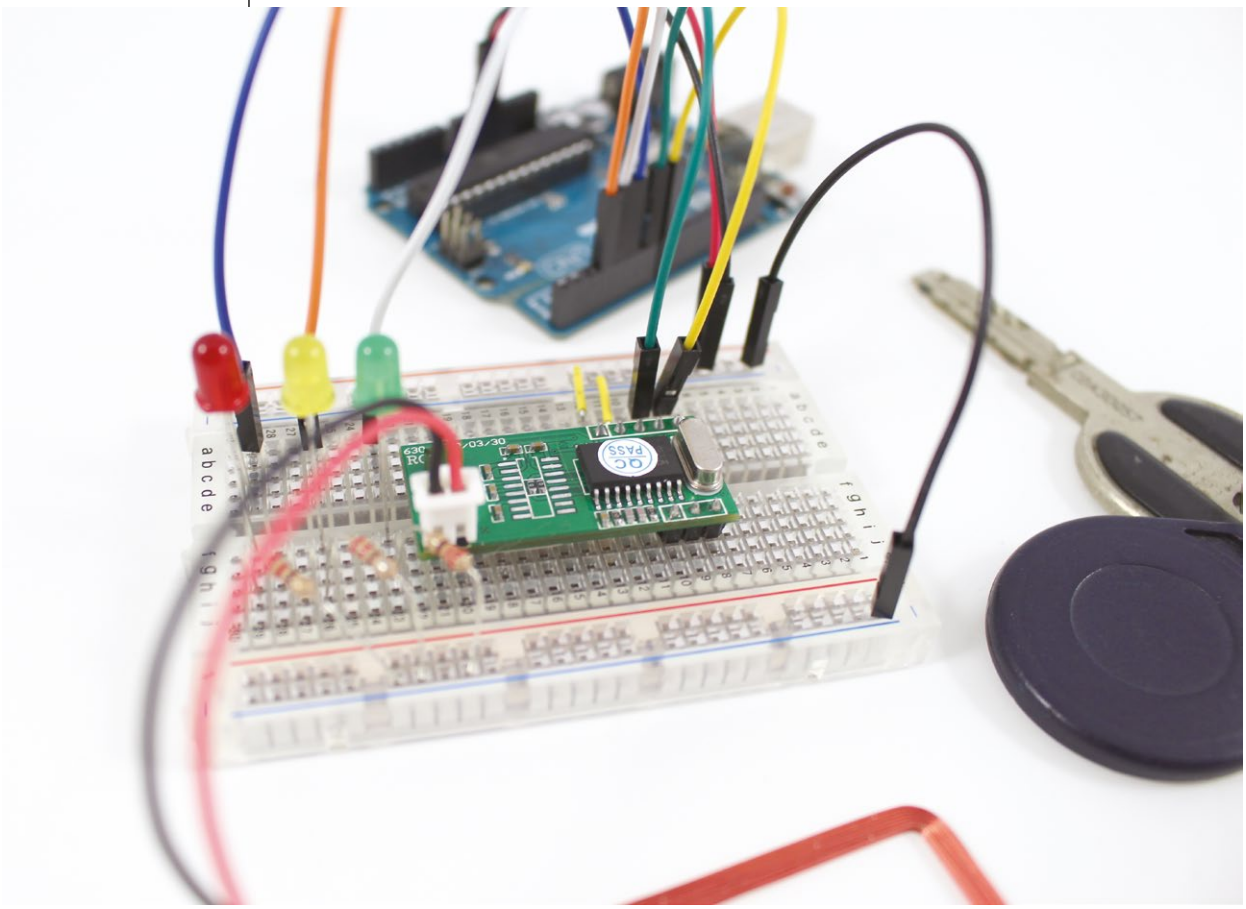
Le module, dès qu'un tag est lu, envoie un message composé de texte à l'Arduino. Il débute avec le caractère `0x02` en guise d'entête. Celui-ci correspond au caractère ASCII non imprimable STX pour *Start of TeXte*. S'en suivent 10 caractères qui ensemble représentent l'identifiant contenu dans le tag. Cet identifiant est en réalité composé d'un code fabricant de 8 bits et d'un numéro de 4 valeurs 8 bits. Ce que nous envoie donc le lecteur est la notation hexadécimale de ces 5 valeurs, sous la forme de 10 caractères. Voilà qui est très pratique pour afficher ces données sur un écran, mais pas du tout pour faire des calculs et des comparaisons.




*Exemple de message envoyé par le module lors de la lecture d'un tag RFID EM4100.*

À propos de calculs justement, deux autres caractères sont ensuite ajoutés. Là encore sous la forme d'une représentation hexadécimale d'une valeur correspondant à une somme de contrôle permettant de vérifier l'intégrité des 5 valeurs composants l'ID. Attention, j'ai bien dit « des valeurs » et non des caractères qui représentent cette valeur. Enfin, le message se termine avec le `0x03` qui, comme vous pouvez vous en douter est le caractère ASCII ETX pour *End of TeXte*.

*Le montage peut être fait simplement « en l'air » ou sur platine à essais puisque le module, souvent vendu déjà broché, utilise des dimensions parfaitement adaptées.*



Écarter le premier et dernier caractère est facile, mais nous avons un problème : le module envoie du texte et nous devons convertir celui-ci en valeurs pour pouvoir vérifier la somme de contrôle, qui elle aussi n'est qu'un texte.

Pour régler cela efficacement, nous créons une fonction `hex2int()` qui prend en argument les deux caractères qui forment une valeur 8 bits en hexadécimal et qui retourne la valeur sous la forme d'un octet (une vraie valeur numérique donc). Il suffit de jeter un œil sur une table ASCII pour nous rendre compte que le caractère « 0 » correspond à la valeur décimale `48`, le « 1 » à `49`, etc. Mais le « A » est le caractère `65`, le « B » est `66`... Il y a un trou entre « 9 » et « A », mais tout ce que nous avons à faire est de prendre la valeur du caractère et lui soustraire :

- 48 si c'est un caractère avant « : » (qui est juste après « 9 » dans la table ASCII) ;
- et 55 dans le cas contraire (parce que c'est forcément un « A », « B », « C », « D », « E » ou un « F »).

Ainsi, si le caractère est « 0 », sa valeur est 48. Si on lui retranche 48, cela nous donne la valeur 0 et ainsi, « 0 » = `0`. Pour une lettre, si c'est un « A » sa valeur est 65, en lui retranchant 55, nous arrivons à `10`, soit `0x0a` et donc « A » = `0x0a`.

S'il s'agit du premier caractère (celui à gauche), il nous suffit alors de prendre la valeur obtenue et de la décaler de 4 bits sur la gauche. Notre `0x0a` devient alors `0xa0`. Pour combiner les deux caractères en une seule valeur après ce décalage, un OU logique est suffisant : `0xa0` OU `0x03` donne `0xa3`, par exemple.

Notre code aurait pu s'écrire :

```
if(haut<58)
  ret |= (haut-48)<<4;
else
  ret |= (haut-55)<<4;

if(bas<58)
  ret |= (bas-48);
else
  ret |= (bas-55);
```

Mais l'utilisation de l'opérateur ternaire (condition `?` si vrai `:` sinon) permet un code bien plus compact. Tout ce que nous avons à faire pour obtenir une valeur avec notre fonction est alors d'utiliser `variable = hex2int(caract_gauche, caract_droit)`.

## La somme de contrôle

Grâce à notre `hex2int()`, il devient facile de manipuler l'ID du tag et la somme de contrôle. Nous transformons les cinq valeurs de l'ID que nous plaçons dans le tableau d'octet (`byte`) `cardID[]` puis nous calculons la somme de contrôle.

Celle-ci est un simple OU exclusif (XOR) entre toutes les valeurs de l'ID. Exemple avec l'ID `2C00827F4A`, nous calculons `0x2c` OU `0x00` OU `0x82` OU `0x7f` OU `0x4a` et obtenons 0x9b. Il nous suffit alors de vérifier l'égalité entre ce résultat et la valeur présente dans le message, en utilisant encore une fois `hex2int()`. Si les deux sont égales, alors l'intégrité de l'ID que nous avons reçu est vérifiée. Dans le cas contraire, on retourne (`return`) simplement en oubliant le message et en bouclant une nouvelle fois dans `loop()` pour une nouvelle lecture.

## Validation et long long

À ce stade, nous avons un ID vérifié pour le tag, nous pouvons donc sereinement nous pencher sur la vérification pour savoir s'il s'agit ou non d'un tag autorisé.

Il existe bien des manières de vérifier la présence d'une donnée dans une liste ou un tableau. Celle que j'ai choisie ici nous permettra de découvrir un type de données rarement utilisé sur Arduino (Atmel AVR 8 bits) : le `long long`. Les types les plus courants avec Arduino sont :

- `char` : 1 octet ;

- `byte` : 1 octet ;
- `int` : 2 octets ;
- `unsigned int` : 2 octets ;
- `long int` (ou `long`) : 4 octets ;
- `unsigned long int` (ou `unsigned long`) : 4 octets ;
- `float` : 4 octets ;
- `double` : 4 octets ;
- `bool` : 2 octets.

Dans notre cas, l'ID du tag fait 5 octets, soit 40 bits, un `unsigned long` n'est donc pas suffisant. Nous pouvons en revanche utiliser `unsigned long long` permettant de stocker des valeurs non signées sur 64 bits. Bien entendu, le microcontrôleur AVR des Arduino (sauf Zero et Due) repose sur un processeur 8 bits. Ceci signifie que pour manipuler des valeurs supérieures à cette taille, il doit le faire en plusieurs opérations et donc en plusieurs cycles d'horloge.

Avec un `int` ou même un `long` cela passe encore, mais dans le cas du `long long` ces manipulations peuvent s'avérer très lentes (à l'échelle des 16 Mhz de l'AVR) et problématiques dans certains cas. Ici ce n'est pas un souci mais, comme lu sur un forum Avrfreaks, gardez bien en tête que `long long` = « lent lent ».

Nous composons le contenu de notre variable `numID` de 64 bits dans la même boucle qui compose `cardID[]`. À chaque tour dans la boucle, nous décalons de 8 bits vers la gauche le contenu puis ajoutons, avec un OU logique, une nouvelle valeur 8 bits. Ce qui donne respectivement, par exemple :

- `0x0000000000000000` avant la boucle ;
- `0x000000000000002c` ;
- `0x0000000000002c00` ;
- `0x00000000002c0082` ;
- `0x000000002c00827f` ;
- `0x0000002c00827f4a` en sortie de la boucle.

Nous gaspillons 24 bits de SRAM pour `numID` et un peu de flash (grâce à `const`) pour stocker le tableau des ID acceptés.

Pour la validation en tant que telle, nous écrivons une fonction dédiée qui prend en argument un `unsigned long long` et le cherche dans le tableau `auth[]` avec une boucle très classique. Grâce à `sizeof()` retournant la taille d'une variable en octets nous obtenons la taille du tableau complet que nous divisons par celle d'un de ses éléments. Ceci nous permet de connaître le nombre d'éléments du tableau qu'il suffit alors de parcourir avec une boucle `for`. Si une correspondance est trouvée, notre fonction retourne VRAI immédiatement. Dans le cas contraire, la fonction se termine normalement en retournant FAUX. Pour valider un ID, il nous suffit donc de tester la valeur de retour de notre `valide()` et d'agir en conséquence.

## Le problème de la pause

Ces modules RFID sont très sympathiques, mais ont une fâcheuse tendance à être bien trop bavards. Un tag placé à proximité va provoquer non pas une lecture, mais toute une série. Or seul le premier message valide (de 14 caractères), accepté ou non, nous intéresse pour déclencher une action. Il nous faut donc un moyen de mettre en pause la lecture durant un certain temps (ici 5 secondes, tel que défini par la macro `PAUSE`).

Une simple utilisation de `delay()` n'est pas adaptée, car durant cette période, le module peut envoyer d'autres données qui vont s'accumuler dans le tampon de réception et donc forcément finir par être prises en compte au prochain tour de `loop()`. Nous devons donc bien lire les données, mais simplement ne rien en faire durant la période de pause.

La solution choisie consiste à utiliser une variable globale `pret`

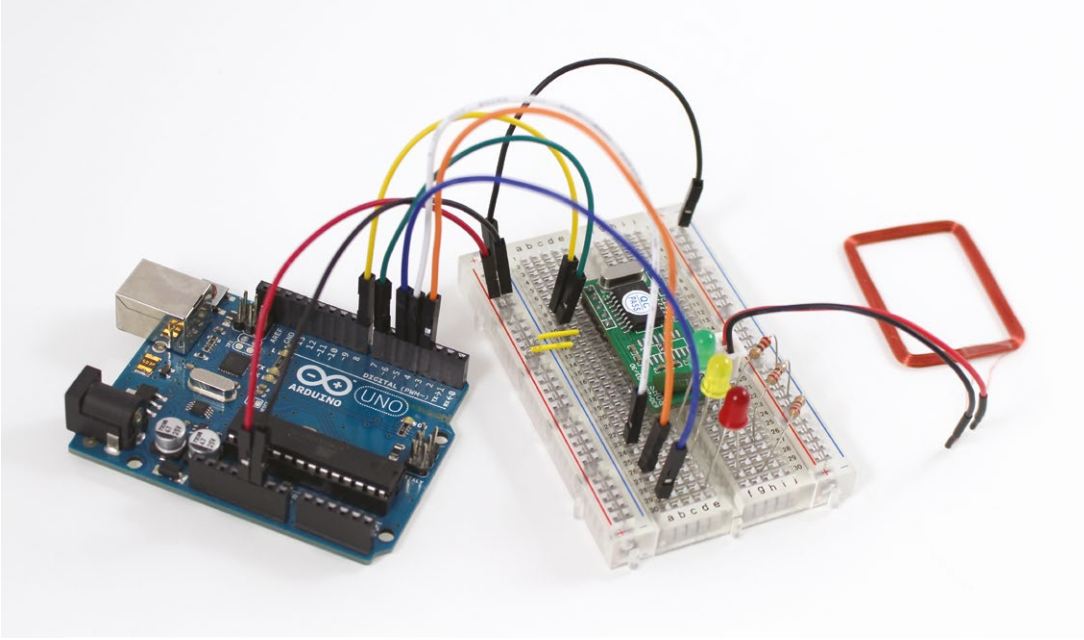
qui est à 1 si nous voulons prendre en compte des données et à 0 dans le cas contraire. Après le traitement de l'ID d'un tag, nous passons `pret` à 0. Cette valeur ne sera changée en 1 que si un délai de 5000 ms est écoulé depuis le dernier traitement. Pour mesurer ce délai, nous utilisons `millis()` permettant de connaître le nombre de millisecondes depuis le démarrage du croquis et ainsi ne bloquons pas l'exécution du code qui peut toujours récupérer les données reçues (et ne rien en faire). Si vous ne connaissez pas encore `millis()`, jetez un œil à l'exemple `BlinkWithoutDelay` livré avec l'environnement Arduino.

# CONCLUSION

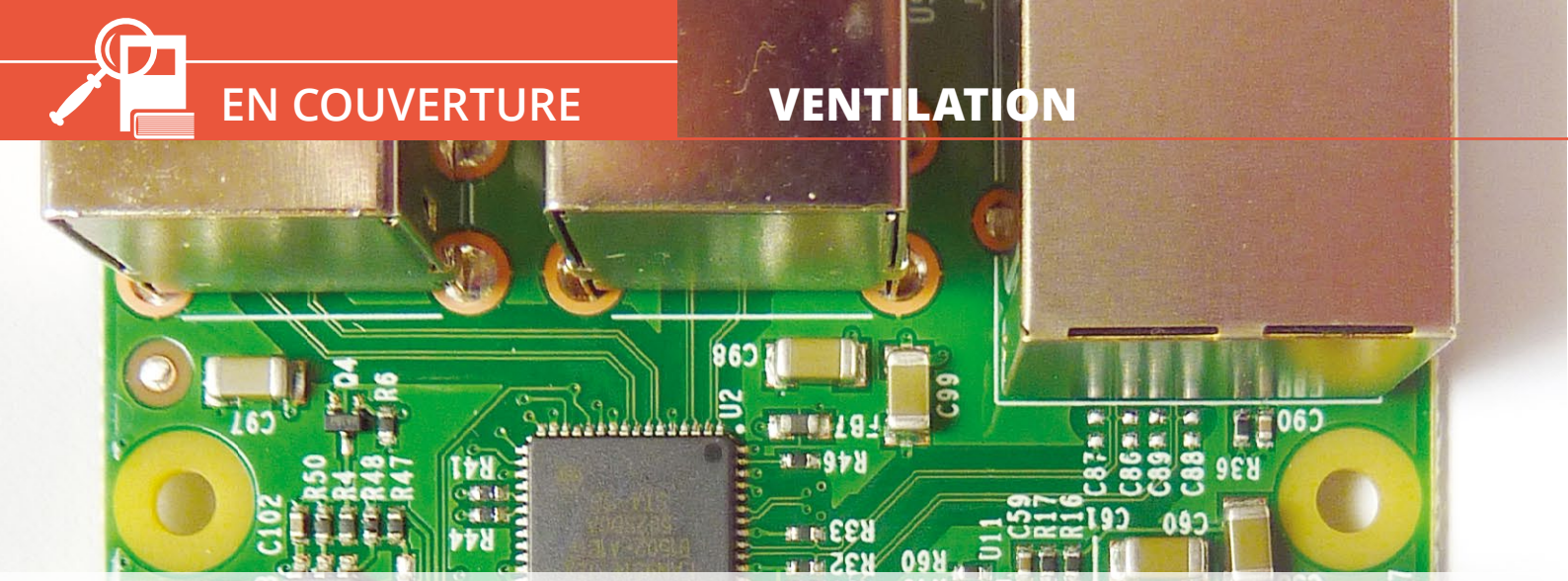
Cette petite réalisation nous a permis de faire connaissance avec une technologie qui, hors de son cadre d'origine peut d'avérer très amusante. En effet, il est important d'avoir une vision large des possibilités et de ne pas s'en tenir à une vision presque dogmatique imposée par un fabricant. Un porte-clés RFID EM4100 est relativement limité en termes de sécurité aujourd'hui, mais si l'on y voit un système permettant simplement de « relever » un identifiant sans contact on peut y voir :

- un moyen d'identifier une position pour un robot mobile ou un véhicule ;
- une méthode de saisie où chaque tag peut représenter une couleur, une forme, une lettre, un son ;
- une interface ergonomique pour un contrôle domotique où les tags seraient embarqués dans des effigies pour déclencher une « ambiance » particulière ;
- un système de base pour un jeu comme une roue de la Fortune avec chaque « case » correspondant à un tag ;
- etc.

Ne perdez simplement jamais de vue une chose importante : la sécurité est quasi nulle et à peine plus évoluée que celle d'un simple système de code-barres. **DB**



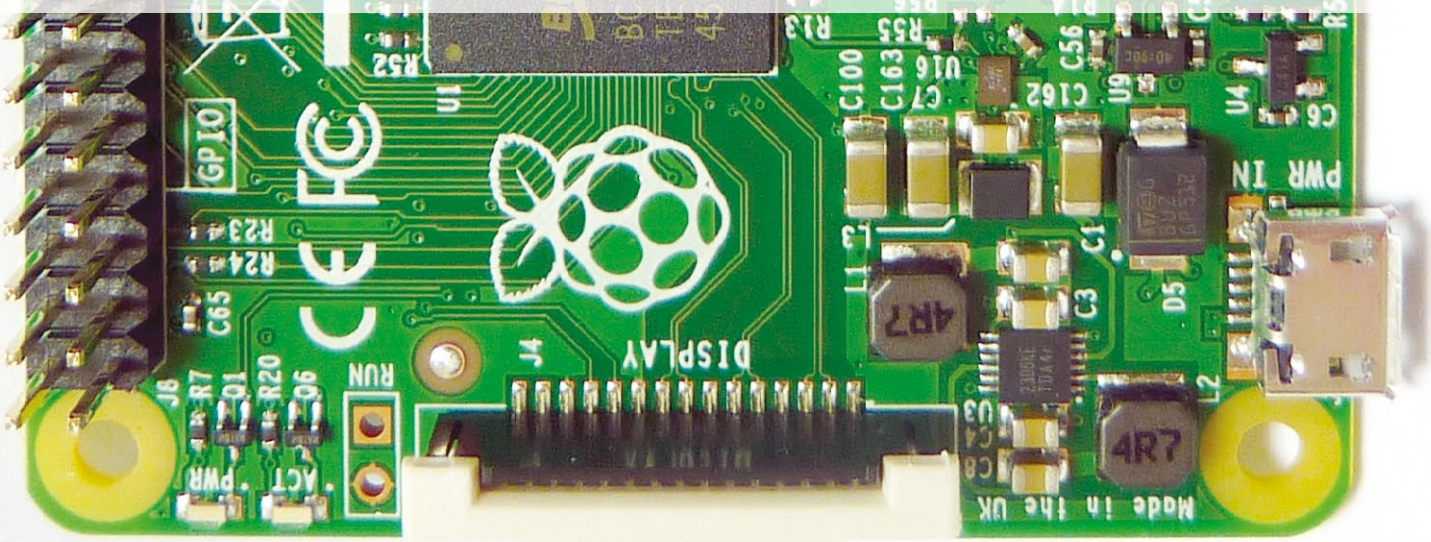

*Notre montage exemple utilise des leds, mais il est bien entendu possible de contrôler n'importe quoi en fonction de la lecture d'un tag RFID valide : relais, moteurs, lumière, buzzer, etc.*



# VENTILATION CONTRÔLÉE PAR DES FRAMBOISES

*Ludovic Apvrille & Axelle Apvrille*

**La ventilation d'une maison est indispensable pour son assainissement. L'article propose un moyen à bas coût pour ventiler automatiquement et efficacement une habitation, tout en laissant à l'utilisateur la possibilité de piloter manuellement sa ventilation depuis Internet.**



# 1. COMMENT VENTILER SON LOGEMENT SANS OUVRIR LA FENÊTRE

Ventiler son logement de façon permanente est une obligation légale en France métropolitaine **[1]**. Cette ventilation a pour objectif de renouveler l'air des logements, notamment afin d'évacuer la vapeur d'eau produite par la respiration ou par la cuisine : cela permet de limiter l'humidité des logements, et donc la prolifération de moisissures.

Pour ventiler un logement, outre le fait d'ouvrir la fenêtre ;-), il existe deux techniques. La première, dite « naturelle », consiste à installer des conduits de tirage, en général au niveau du plafond. L'air chaud, par convection, aura alors tendance à s'échapper du logement par les conduits, provoquant une aspiration d'air neuf dans le logement. La deuxième technique consiste à installer une Ventilation Mécanique Contrôlée – VMC – afin de forcer l'air vicié du logement à travers des conduits d'évacuation.

Les VMC sont de trois types (du plus ancien, au plus récent) :

1. Les ventilations multi-vitesses, dont la vitesse est réglable manuellement. Elles possèdent en général deux vitesses, et sont pilotées par un bouton à 3 positions (arrêt, vitesse 1, vitesse 2).
2. Les ventilations à détection de taux d'humidité (dites « hygroréglables »). Ces ventilations sont multi-vitesses, et adaptent leur vitesse en fonction du taux d'humidité de l'air extrait. C'est-à-dire qu'un taux faible d'humidité dans le logement fait commuter la ventilation sur la petite vitesse, alors qu'un taux élevé fait commuter la ventilation sur une vitesse plus élevée.
3. Les ventilations double-flux, qui consistent à réchauffer l'air aspiré dans le logement en le faisant passer à travers un échangeur thermique qui fonctionne avec l'air extrait du logement. Ces ventilations sont en général hygroréglables.

Les ventilations double-flux sont plus chères à l'achat, plus complexes à installer, mais elles permettent de diminuer la perte de chaleur due à la ventilation en période hivernale. Elles ont un autre défaut : en été, l'air chaud extrait du logement pendant la nuit réchauffe l'air entrant, diminuant ainsi la capacité du logement à se refroidir la nuit.

Les ventilations hygroréglables quant à elles ont l'inconvénient de réguler le flux uniquement selon l'humidité de l'air extrait. Pas possible par exemple d'augmenter le flux pour refroidir son logement en été durant la nuit. Autre cas qui fonctionne mal : si votre logement est un peu humide, mais qu'il pleut dehors, la ventilation va tout de même accélérer l'extraction, et donc faire rentrer plus d'humidité dans votre logement...

Les ventilations multi-vitesses ont comme défaut d'exiger de la part du l'utilisateur un contrôle manuel. Les deux autres types étant soit trop chers (double-flux), soit pas assez flexibles (hygroréglables), nous avons mis en place un moyen d'automatiser de façon « intelligente » la ventilation d'un logement par utilisation d'une ventilation possédant deux vitesses, tout en laissant l'utilisateur guider la ventilation dans certains cas (cuisine odorante, etc.)

# 2. OBJECTIFS DE L'AUTOMATISATION

Pourquoi automatiser une ventilation ? Principalement pour trois raisons :

1. Pour éviter de faire rentrer de l'humidité dans le logement, c'est-à-dire que si l'air qui entre risque d'augmenter le taux d'humidité du logement, alors, il vaut mieux ne pas ventiler.

2. Pour éviter de refroidir/réchauffer le logement à la mauvaise saison (consommation d'énergie supplémentaire), tout en conservant tout de même une ventilation minimale.
3. Pour profiter des possibilités de sur-ventilation. Cela permet de réchauffer la maison aux heures chaudes (automne et printemps surtout), d'éviter qu'elle ne se refroidisse trop en hiver (ventilation uniquement au-delà d'une certaine température, avec un nombre d'heures minimal par jour), et enfin d'éviter le réchauffement en été, en ventilant surtout pendant les heures fraîches. Par exemple, en été, un mode pourrait être d'être en ventilation vitesse 1 par défaut, et ventilation vitesse 2 dès que la température extérieure est inférieure à la température intérieure (sur-ventilation de refroidissement). D'ailleurs, outre le fait de refroidir la maison, cela donne une sensation de courant d'air, bénéfique pour résister à la chaleur.

Nous venons de voir que les paramètres de base pour automatiser de façon « intelligente » une VMC nécessitent le recours à des relevés à la fois de température et d'humidité. Nous verrons à la fin de l'article que d'autres capteurs peuvent aussi être utilisés pour améliorer cette « intelligence ».

# 3. ARCHITECTURE DE L'AUTOMATISATION

Bon, alors, vous voulez ventiler ? Voici le matériel pour contrôler votre ventilation !

## 3.1 Matériel minimal

- Une VMC bi-vitesse.
- Deux interrupteurs 250 Volts classiques. L'un des interrupteurs sert à basculer entre le mode piloté et l'ancien mode « manuel » (il est sage de garder ce mode en précaution !). L'autre permet de sélectionner la vitesse 1 ou 2.
- Deux relais capables de commander du 250V. Attention, nous recommandons fortement l'usage de relais opto-couplés, car lors de l'arrêt de la VMC, il peut y avoir un retour de charge important dû à la capacité inductive du moteur de la VMC. En effet, lors de l'arrêt de la VMC, le moteur va continuer à tourner, et donc jouer le rôle d'un alternateur qui va envoyer un courant vers les relais. Nous verrons plus tard comment le logiciel qui pilote les relais peut éviter en partie ce retour de charge (mais pas totalement...). Bref, nous avons grillé plusieurs relais non opto-couplés, donc... Certains relais se pilotent en 5V, d'autres en 3.3V, ou encore avec d'autres tensions. Le Raspberry Pi possède une alimentation externe en 5V, et une autre en 3.3V. Les GPIOs sont uniquement en 3.3V. Attention : référez-vous donc à la documentation ou au schéma de votre module à relais pour adapter la connexion entre vos relais et le Raspberry Pi.
- Un capteur d'humidité extérieure, un capteur de température extérieure, et un capteur de température intérieure.
- Un Raspberry Pi sur lequel sont connectés les capteurs (température, humidité) et chargé de commander les relais. Si vous désirez piloter la VMC depuis un site Internet/votre mobile, il faut aussi que votre carte ait un accès WIFI/Ethernet/Bluetooth. Vous pourriez également utiliser une carte de type Arduino, mais nous avons eu des problèmes de stabilité WIFI/Ethernet avec, c'est-à-dire

qu'il était difficile de maintenir le système opérationnel pendant des mois sans reboot.

## 3.2 Notre configuration matérielle

### 3.2.1 Architecture générale

Selon la configuration de votre logement, il peut être compliqué de relever la température, l'hygrométrie et de piloter la VMC sur une même carte. C'est notre cas : nous avions déjà un Raspberry Pi (appelé par la suite RPi1) qui gérait notre station météo, mais il était difficile de piloter la VMC depuis celui-ci (il aurait fallu tirer des fils – flemmingite aigüe). Nous avons donc placé un 2ème Raspberry Pi (appelé RPi2) proche de la VMC.

Ainsi, l'architecture générale de notre installation est représentée à la figure 1. Cette installation comporte :

- Des capteurs de température et d'humidité reliés par ondes radio à la base de notre station météo (une Oregon Scientific WMR200). Cette base comporte elle-même un capteur d'humidité et de température intérieure. Cette base est connectée en USB au RPi1.
- Un RPi1 en charge de l'« intelligence ». Il relève les données issues de la station météo, envoie des ordres à RPi2 pour commander la VMC, et gère le site web de pilotage de la VMC.

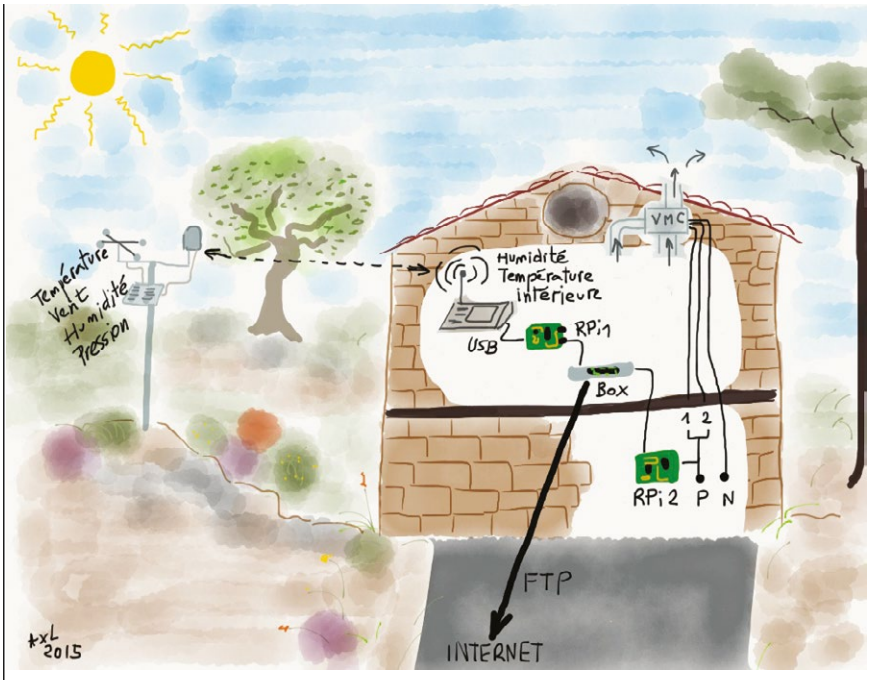



*Fig. 1 : Architecture générale de l'installation.*

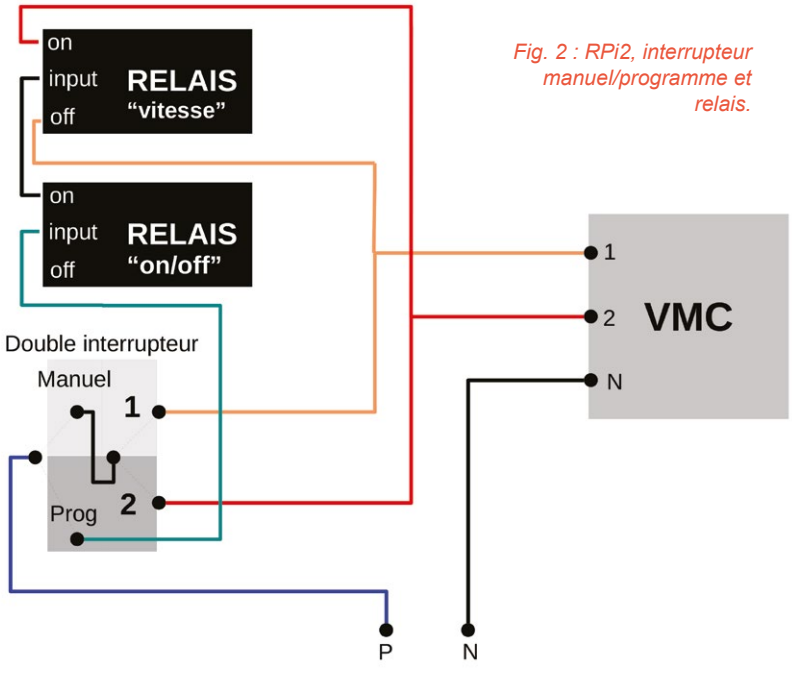



*Fig. 2 : RPi2, interrupteur manuel/programme et relais.*

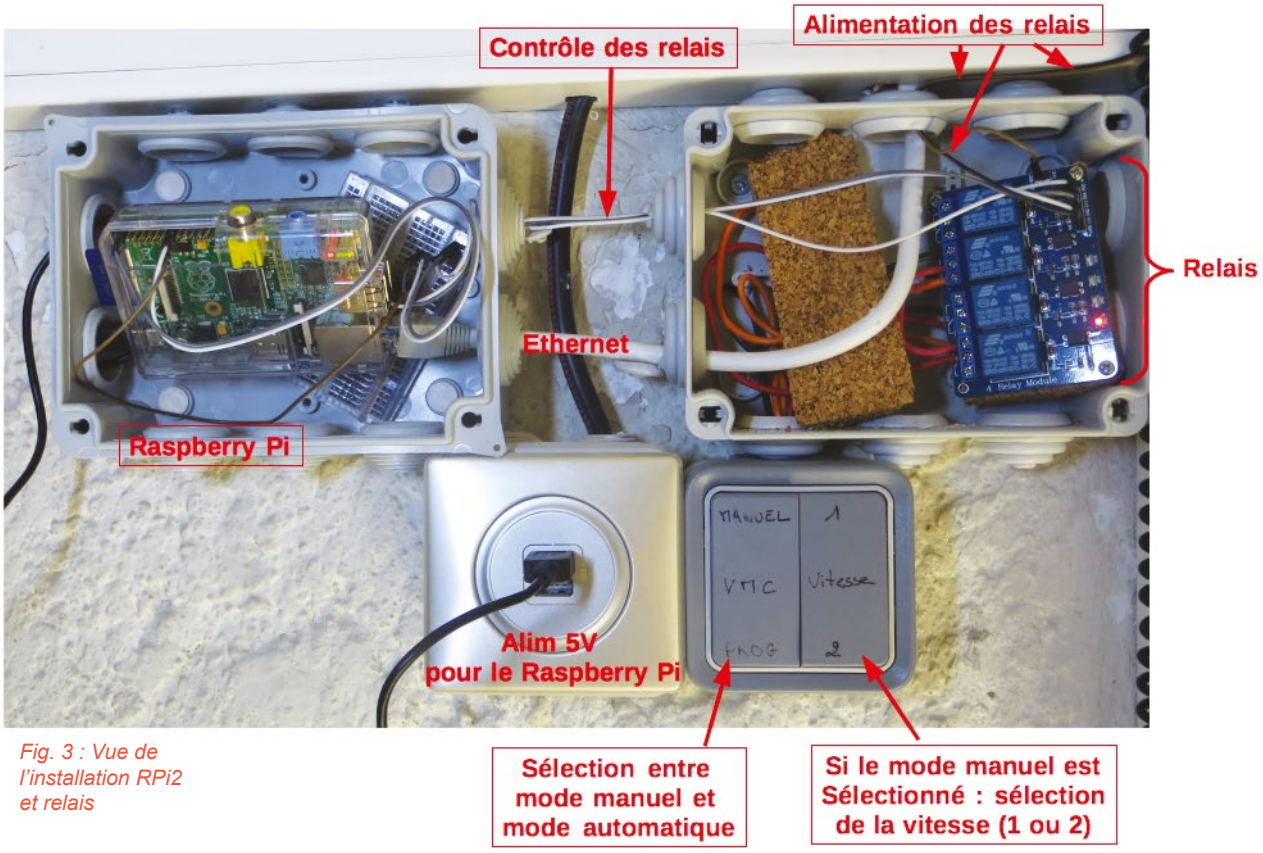



*Fig. 3 : Vue de l'installation RPi2 et relais*

- Un RPi2 en charge de la VMC. C'est lui qui, physiquement, coupe la VMC ou la met en vitesse 1 ou 2 via deux relais. Il reçoit ses ordres de RPi1 via HTTP. Un serveur web (très basique) tourne donc dessus. Le RPi2 est relié au RPi1 par Ethernet (mais l'on aurait bien sûr pu utiliser du WIFI, du Zigbee, etc.).
- Les deux relais de commande de la VMC. Le relais 1 sert à allumer la VMC, le relais 2 sert à commuter entre vitesse 1 et vitesse 2. Cette commande n'est active que si l'interrupteur de choix entre mode manuel et mode programmé est actif (voir la figure 2, page précédente).

L'ensemble RPi2 et relais est visible à la figure 3. La figure 4 présente le câblage entre RPi2 et les relais.

### 3.2.1 Configuration logicielle du RPi2 et serveur du RPi2

Le logiciel fonctionnant sur le RPi2 est assez simple : son rôle est principalement de lire les «GET HTTP » reçus de RPi1, d'exécuter l'ordre (activation/désactivation des relais) et de renvoyer un statut sous la forme d'une page HTML.

La première étape pour que le mini site Internet du RPI2 puisse commander les relais est d'activer les GPIOS du RPi2. Pour cela, en supposant que le RPi2 exécute une distribution Raspbian de Linux, il faut lancer un terminal sur le RPi2 puis configurer les GPIOs. Dans notre cas, nous utilisons les ports 22 et 23. Pour créer le port 22, il faut faire (en **sudo** ou root) :

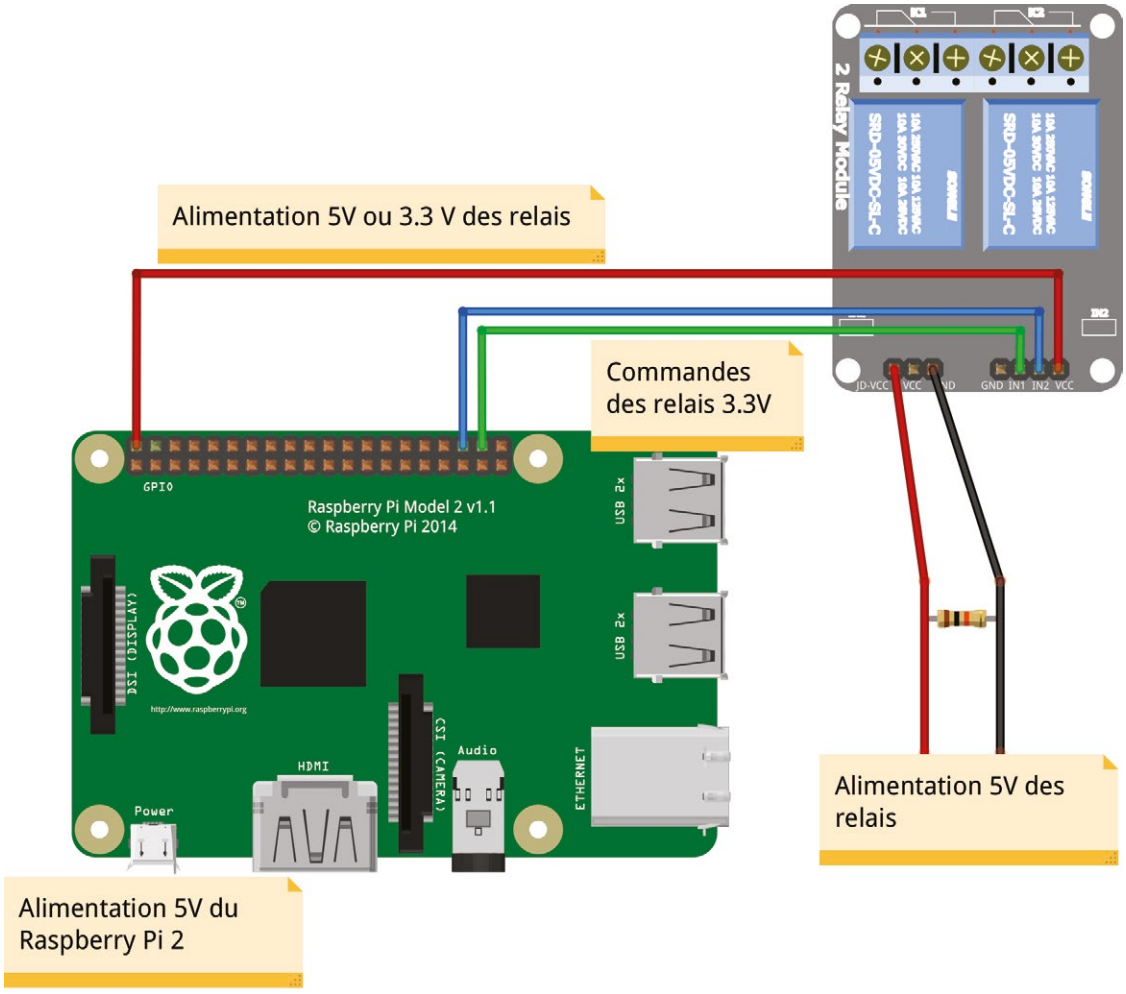



*Fig. 4 : Vue de l'installation RPi2 et relais.*

```
% echo 22 > /sys/class/gpio/export
```

Il faut ensuite configurer le fait que le port 22 soit un port de sortie :

```
% echo out > /sys/class/gpio/gpio22/direction
```

Il faut, selon le groupe dans lequel tournera le site internet du RPi2, donner le droit d'accès des relais à ce groupe :

```
SUBSYSTEM=="gpio*", PROGRAM="/bin/sh -c 'chown -R root:gpio /sys/class/gpio;
chmod -R 770 /sys/class/gpio; chown -R root:gpio /sys/devices/virtual/gpio;
chmod -R 770 /sys/devices/virtual/gpio'"
```

Le serveur web de contrôle des relais est écrit en quelques dizaines de lignes Python. Il n'offre aucune sécurité particulière : il doit donc se trouver derrière un pare-feu, et n'autoriser que certaines adresses sources (par exemple : le RPi1).

Le code Python comporte surtout 4 parties : le démarrage du serveur web, la gestion des requêtes HTTP, et l'activation des relais en fonction des requêtes, et les logs. L'ensemble du code est open source, et est disponible en **[2]**.

Commençons par le système de logs : les informations du serveur web sont écrites dans les logs du système comme suit :

```
def myPrint(s):
    s1 = "[RC] %s" %(s)
    syslog.syslog(s1)
```

Le démarrage du serveur web est assez simple :

```
 myPrint('Starting relay server')
       server = HTTPServer(('', PORT), MyHandler)
        myPrint('Started httpserver...')
        server.serve_forever()
   except KeyboardInterrupt:
        myPrint('^C received, shutting down server')
        server.socket.close()
```

La gestion des requêtes consiste à extraire si le GET comporte un mot clé tel que **onRelay1**, **ofRelay1**, **onRelay2**, **ofRelay2**. Dans le cas contraire, seul le statut est retourné. L'extrait suivant montre comment l'on peut gérer de telles requêtes :

```
class MyHandler(BaseHTTPRequestHandler):
    def do_GET(s):
         """Respond to a GET request."""
           myPrint("handling request")
  ...
           s.send_response(200)
           s.send_header("Content-type", "text/html")
           s.end_headers()
```

Les relais sont ensuite activés selon la chaîne reconnue dans la requête :

```
           #Testing s
           if (s.path[1:] == "onRelay1"):
               onRelay1()
           if (s.path[1:] == "ofRelay1"):
               offRelay1()
 ...
```

Enfin, la réponse est envoyée au client :

```
myPrint("Preparing answer")
s.wfile.write("<html><head><title>VMC SERVER</title></head>")
req = "<p>Nb Of requests:" + str(nbOfRequests) + "</p>"
s.wfile.write("<p>You accessed path: %s</p>" % s.path)
s.wfile.write("<p>Uptime: %s</p>" % uptime_string)
```

```
            s.wfile.write(req)
            myPrint("Preparing relay answer")
            s.wfile.write("<p>Relay #1: %s</p>" % relay1String)
            s.wfile.write("<p>Relay #2: %s</p>" % relay2String)
            nbOfRequests +=1
            myPrint("Request: All done")
```

Voici un retour typique. Notre serveur tournait (fièrement) depuis 47 jours.

```
You accessed path: /
Uptime: 4109338.25 sec.
Nb Of requests:7081
Relay #1: ON
Relay #2: OFF
```

Le lancement automatique du serveur sur le RPi2 se fait, par exemple, en ajoutant une ligne de commandes dans le script `/etc/rc.local` :

```
/usr/bin/python /usr/share/vmc/webserver4relays.py &
```

## 4. LOGICIEL DE COMMANDES

Pour les habitants de notre maison, il y a 6 modes différents d'utilisation de la VMC :

- Mode automatique ;
- OFF ;
- Vitesse 1 forcée ;
- Vitesse 2 forcée ;
- Pulse 1. La VMC est en vitesse 1 pendant 30 minutes, puis revient à son mode de fonctionnement précédent ;
- Pulse 2. La VMC est en vitesse 2 pendant 30 minutes, puis revient à son mode de fonctionnement précédent. C'est pratique à la cuisine notamment (mais pas uniquement ...) !

En mode automatique, c'est le RPi1 qui décide de l'état de la VMC. Il y a trois états possibles :

1. La VMC est arrêtée ;
2. La VMC est en vitesse 1 ;
3. La VMC est en vitesse 2.

Parallèlement à ces trois états possibles, nous avons défini deux politiques différentes, choisies par l'utilisateur : la politique qui consiste à conserver la chaleur, voire à réchauffer le logement – c'est le mode pour l'automne, l'hiver et le printemps –, et la politique qui consiste à rafraîchir le logement (l'été). Bon, n'allez pas croire que cela fait office de chauffage ou de climatisation, mais ça aide (presque gratuitement).

## 4.1 Conservation de la chaleur

Conserver la chaleur consiste à ventiler pendant les heures chaudes, et à moins ventiler pendant les heures froides. Lorsque la température extérieure est supérieure à la température intérieure, on choisit alors de forcer la ventilation en vitesse 2 pour réchauffer le logement. Reste qu'en hiver, les heures les plus chaudes correspondent à un air extérieur dont la température est toujours inférieure à la température intérieure (en France métropolitaine). A contrario, en automne et au printemps, il est possible que cela se produise : l'on a alors intérêt à mettre la ventilation en vitesse 2 pour réchauffer le logement. Le reste du temps, on définit simplement des plages horaires de fonctionnement de la VMC en vitesse 1. Enfin, il y a quelques limites : on arrête la VMC si la température extérieure est trop froide (pas forcément intéressant de ventiler par -10°C ! ), ou si l'humidité est trop élevée (pluie, brouillard...).

Voici notre fichier de configuration :

```
# Taux d'humidité au-dessus duquel la VMC est arrêtée
maxOutHumidity=94
#Température en dessous de laquelle la VMC est arrêtée
minOutTemp=0
```

## 4.2 Conservation de la fraîcheur

À l'inverse, la conservation de la fraîcheur consiste à ventiler le logement en vitesse 2 lorsque la température extérieure est inférieure à la température intérieure. C'est donc une surventilation de la maison pendant les nuits d'été. Il serait aussi bien entendu possible de définir une température au-delà de laquelle on ne ventile pas le logement, mais nous ne l'avons pas fait : ainsi, dans les plages où elle est en route, la VMC ventile par défaut en vitesse 1, et en vitesse 2 pour rafraîchir le logement.

## 4.3 Programme principal « autovmc »

Notre programme principal est écrit en Perl. Il est téléchargeable à cette adresse **[2]**. Nous détaillons par la suite les parties du code qui sont a priori les plus intéressantes. Nous présenterons au passage le système de logs qui permet notamment de propager des informations à l'utilisateur sur le site internet de contrôle et de configuration.

### 4.3.1 Vérification de la configuration

Cette section a pour objectif de vérifier que la configuration utilisateur a du sens. Par exemple, que la température minimale de ventilation n'est pas supérieure à 50 degrés.

```
sub check_config {
    my $minOutTemp = shift;
    my $maxOutHumidity = shift;

    if ($minOutTemp < -10 || $minOutTemp > 50) {
        syslog(LOG_ERR, "Incorrect value for minOutTemp=$minOutTemp");
        closelog();
        die "Incorrect value for minOutTemp";
    }
    ...
}
```

HACKABLE MAGAZINE

# DÉCOUVREZ NOS OFFRES D'ABONNEMENTS !

## PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

# www.ed-diamond.com



## LES COUPLAGES PAR SUPPORT :

VERSION
### PAPIER

Retrouvez votre magazine favori en papier dans votre boîte à lettres !




VERSION
### PDF

Envie de lire votre magazine sur votre tablette ou votre ordinateur ?




## SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

| Voici mes coordonnées postales : | |
|---|---|
| Société : | |
| Nom : | |
| Prénom : | |
| Adresse : | |
| | |
| Code Postal : | |
| Ville : | |
| Pays : | |
| Téléphone : | |
| E-mail : | |

❑ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

HACKABLE MAGAZINE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

**Vos remarques :**

.....

.....

# VOICI TOUTES LES OFFRES COUPLÉES AVEC HACKABLE !

**POUR LE PARTICULIER ET LE PROFESSIONNEL ...**

Prix TTC en Euros / France Métropolitaine

## CHOISISSEZ VOTRE OFFRE !

Prix en Euros / France Métropolitaine

| Offre | SUPPORT ABONNEMENT | PAPIER Réf | PAPIER Tarif TTC | PAPIER + PDF (PDF 1 lecteur) Réf | PAPIER + PDF (PDF 1 lecteur) Tarif TTC | PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf | PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Tarif TTC | PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf | PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Tarif TTC |
|---|---|---|---|---|---|---|---|---|---|
| HK | 6n° HK* | ☐ HK1 | 39,- | ☐ HK12 | 58,- | | | | |
| **LES COUPLAGES « EMBARQUÉ »** | | | | | | | | | |
| D | 6n° HK* + 4n° OS | ☐ D1 | 65,- | ☐ D12 | 98,- | ☐ D13 | 85,-* | ☐ D123 | 118,-* |
| E | 6n° HK* + 4n° OS + 6n° MISC | ☐ E1 | 105,- | ☐ E12 | 158,- | ☐ E13 | 179,-* | ☐ E123 | 232,-* |
| E+ | 6n° HK* + 4n° OS + 6n° MISC + 2n° HS | ☐ E+1 | 119,- | ☐ E+12 | 179,- | ☐ E+13 | 193,-* | ☐ E+123 | 253,-* |
| F | 6n° HK* + 4n° OS + 11n° GLMF | ☐ F1 | 125,- | ☐ F12 | 188,- | ☐ F13 | 229,-* | ☐ F123 | 292,-* |
| F+ | 6n° HK* + 4n° OS + 11n° GLMF + 6n° HS | ☐ F+1 | 183,- | ☐ F+12 | 275,- | ☐ F+13 | 287,-* | ☐ F+123 | 379,-* |
| G | 6n° HK* + 4n° OS + 6n° LP | ☐ G1 | 100,- | ☐ G12 | 150,- | ☐ G13 | 164,- * | ☐ G123 | 214,-* |
| G+ | 6n° HK* + 4n° OS + 6n° LP + 3n° HS | ☐ G+1 | 129,- | ☐ G+12 | 194,- | ☐ G+13 | 193,-* | ☐ G+123 | 258,-* |
| **LES COUPLAGES « GÉNÉRAUX »** | | | | | | | | | |
| H | 6n° HK* + 4n° OS + 6n° LP + 6n° MISC + 11n° GLMF | ☐ H1 | 200,- | ☐ H12 | 300,- | ☐ H13 | 402,-* | ☐ H123 | 499,-* |
| H+ | 6n° HK* + 4n° OS + 6n° LP + 3n° HS + 6n° MISC + 2n° HS + 11n° GLMF + 6n° HS | ☐ H+1 | 301,- | ☐ H+12 | 452,- | ☐ H+13 | 493,-* | ☐ H+123 | 639,-* |

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

**N'hésitez pas à consulter les détails des offres ci-dessus sur : www.ed-diamond.com !**

### 4.3.1 Relevé des valeurs météorologiques

Il s'agit là d'aller récupérer les valeurs extérieures de température et humidité, et la valeur intérieure de température. Cette récupération dépend bien entendu du type et de la configuration de votre station météo. Aussi, il faut vérifier que ces valeurs aient du sens – en cas de bug ... – et que ces valeurs soient récentes, par exemple en cas d'erreur dans le relevé ou le stockage des informations ... ou tout simplement si la pile d'un capteur devient trop faible. Imaginez que votre relevé de température date de 3 mois : la programmation de la VMC ne serait pas du tout adaptée !

Dans notre cas, le relevé des dernières valeurs se fait avec une requête dans la base de données SQL de stockage des valeurs météo. Ce code montre aussi le système de logs mis en place. D'une part, les appels à **debug_log** qui affichent les messages si le programme est démarré en mode « debug ». Et d'autre part, les appels à syslog qui ajoutent des messages au log du système (par exemple : **/var/log/messages**). Cela est particulièrement important dans la dernière partie du code pour expliquer l'absence d'actions sur la VMC lorsque les relevés sont incorrects.

```
sub read_weather {
    # This retrieves the most recent entry in the database
    my $dateTime='sqlite3 $weewx_dir/archive/weewx.sdb "select max(dateTime)
from archive;"';
    chomp($dateTime);

    # make sure dateTime is sound
    my $currentTime = time();
    print "weewxTime=$dateTime current=$currentTime\n";
    if ($currentTime - $dateTime > 10 * 60 * 60) {
        debug_log("Weewx time is obsolete");
        syslog(LOG_WARNING, "Last weewx measure too old: autovmc is quitting");
    }

    # This gets the temperature for that entry
    my $inTemp='sqlite3 /home/weewx/archive/weewx.sdb "select inTemp from
archive where dateTime=${dateTime};"';

    my $outTemp = 'sqlite3 /home/weewx/archive/weewx.sdb "select outTemp from
archive where dateTime=${dateTime};"';

    my $outHumidity = 'sqlite3 /home/weewx/archive/weewx.sdb "select
outHumidity from archive where dateTime=${dateTime};"';
    chomp($inTemp);
    chomp($outTemp);
    chomp($outHumidity);

    if ($inTemp < -20 || $inTemp > 50) {
        syslog(LOG_ERR, "Inner temperature is wrong: $inTemp"); closelog();
        die "Something wrong with inner temperature: $inTemp";
    }

    if ($outTemp < -20 || $outTemp > 50) {
```

```
        syslog(LOG_ERR, "Outer temperature is wrong: $outTemp"); closelog();
        die "Something wrong with outer temperature: $outTemp";
    }

    if ($outHumidity < 0 || $outHumidity > 100) {
        syslog(LOG_ERR, "Outer humidity is wrong: $outHumidity"); closelog();
        die "Something wrong with outer humidity: $outHumidity";
    }

    debug_log("Inner temperature: $inTemp\n");
    debug_log("Outer temperature: $outTemp\n");
    debug_log("Outer humidity:    $outHumidity\n");
    return $inTemp, $outTemp, $outHumidity;
}
```

### 4.3.1 Fonction « warm_house »

La fonction **warm_house**, dont l'objectif est décrit plus haut, s'implémente de la façon suivante (pour une meilleure lisibilité, nous avons ôté les lignes qui correspondent aux logs) :

```
 sub warm_house {
    my $inTemp = shift;
    my $outTemp = shift;
    my $outHumidity = shift;
    my $minOutTemp = shift;
    my $maxOutHumidity = shift;

    if (-e $pulse_indicator || (! -e $auto_indicator)) {
        syslog(LOG_NOTICE, "Manual mode or pulse detected - autovmc won't
do anything");
    } else {
        if ($outTemp > $minOutTemp && $outHumidity < $maxOutHumidity) {
            if ($outTemp > $inTemp) {
                system("$vmc_dir/vmctrl.pl --high");
            } else {
                system("$vmc_dir/vmctrl.pl --low");
            }
        } else {
            system("$vmc_dir/vmctrl.pl --stop");
        }
    }
    return $inTemp, $outTemp, $outHumidity;
}
```

### 4.3.1 Fonction « cool_house »

La fonction **cool_house** est plus simple. Son code Perl est le suivant :

```
sub cool_house {
    my $inTemp = shift;
    my $outTemp = shift;
```

```
    if (-e $pulse_indicator || (! -e $auto_indicator)) {
        syslog(LOG_NOTICE, "Manual mode or pulse detected - autovmc won't do
anything");
    } else {
        debug_log("\tAutomatic mode: let's do something smart (perhaps)\n");
        if ($outTemp < $inTemp) {
            system("$vmc_dir/vmctrl.pl --high");
        } else {
            system("$vmc_dir/vmctrl.pl --low");
        }
    }
}
```

## 4.4 Programmation à certaines heures

La programmation à certaines heures repose simplement sur le principe de mettre dans la crontab du RPi1 le lancement du script Python présenté précédemment. La crontab comprend en fait deux entrées :

- une pour lancer le script **autovmc** d'exécution du mode de fonctionnement de la VMC. Dans la configuration ci-dessous, il s'exécute toute les 10 minutes de 7h du matin à 21h50 ;
- une deuxième pour arrêter le VMC à 22h.

```
% crontab -l
*/10 7,8,9,10,11,12,13,14,15,16,17,18,19,20,21 * * * /home/axelle/scripts/
autovmc.pl &> /dev/null
0 22 * * * /usr/share/vmc/vmctrl.pl --stop &> /dev/null
```

# 5. SITE INTERNET POUR LE CONTRÔLE DU FONCTIONNEMENT ET LA CONFIGURATION DES MODES

Le site Internet de la VMC a deux objectifs. Tout d'abord, piloter la VMC manuellement dans le cas où la politique automatique ne convient pas. Ensuite, visualiser l'état courant de la VMC. Cela sert d'une part à contrôler que la politique actuelle ne comporte pas de bogues, et d'autre part à vérifier que le mode en cours correspond à ce que désire l'utilisateur.

## 5.1 Configuration des modes de fonctionnement

Le site Internet utilisateur de la VMC fonctionne sur le RPi1. Il permet de choisir entre le mode totalement manuel (off, vitesse 1, vitesse 2, pulse1, pulse2, cancel) et le mode automatique. Un extrait de ce site est visible à la figure 5, page suivante.

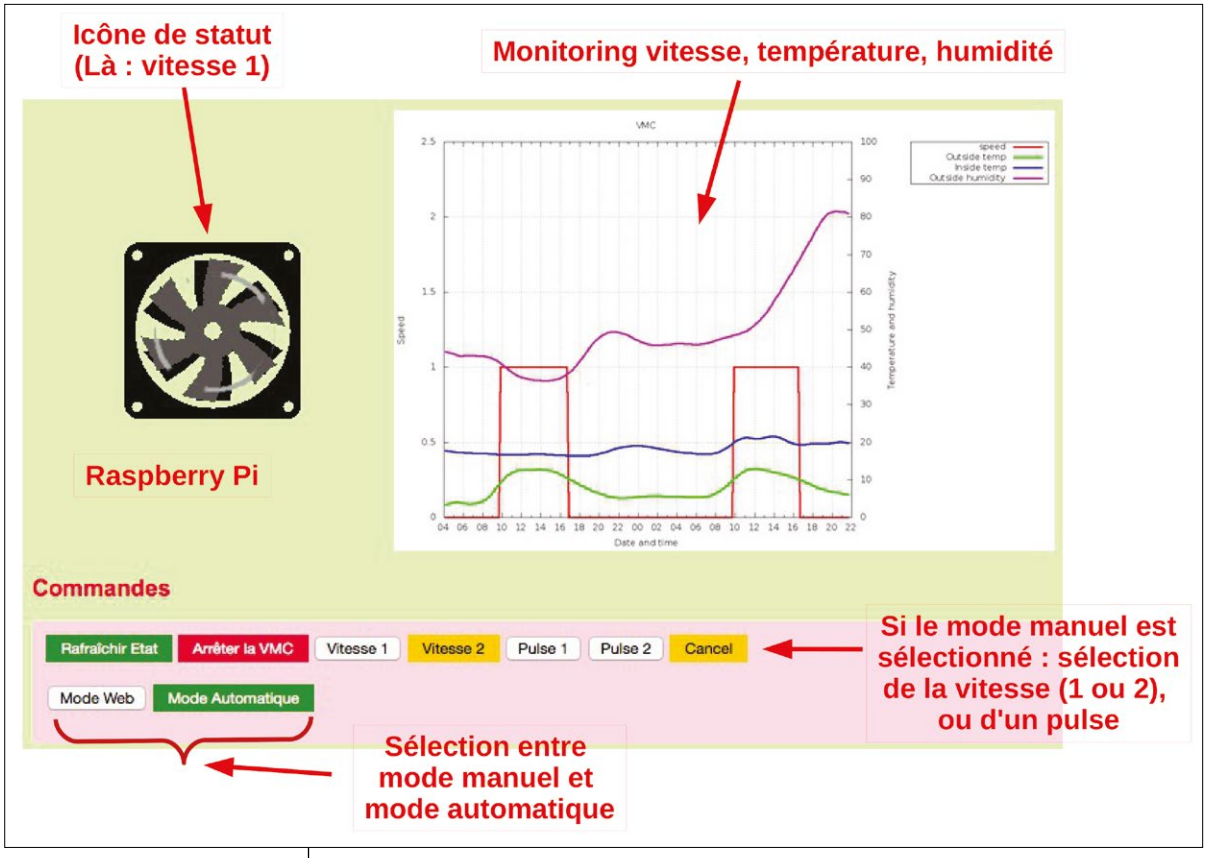



*Fig. 5 : Site internet de la VMC, RPI1.*

## 5.2 Vérification du fonctionnement de la VMC

Plusieurs informations sont disponibles pour vérifier le fonctionnement de la VMC (voir figure 6):

- un pictogramme visuel représentant un ventilateur, qui permet d'identifier rapidement le mode de fonctionnement (OFF, 1, 2) ;
- des informations de « debug » qui indiquent la valeur des relais, les températures extérieures et intérieures, et l'humidité extérieure ;
- une courbe qui présente l'évolution de la vitesse de la VMC, et des paramètres météorologiques. Cela permet de vérifier les plages de fonctionnement de la VMC. Cette courbe est présentée à la figure 6. Cette courbe est générée avec un script python et un script `gnuplot` que nous allons à présent expliquer.

Le script Perl suivant a pour objectif de récupérer les infos récentes de température, humidité et vitesse de la VMC afin de générer dans un fichier pris en entrée par le script `gnuplot`. Les principales fonctions du script Perl sont :

- récupérer l'heure/date courante afin de vérifier que les données sont récentes :

```
    my $dateTime='sqlite3 $weewx_dir/archive/weewx.sdb "select
max(dateTime) from archive;"';
    chomp($dateTime);

    # make sure dateTime is sound
    my $currentTime = time();
    if ($currentTime - $dateTime > 10 * 60 * 60) {
 debug_log("Weewx time is obsolete");
    }
```

• récupérer les données de température/humidité :

```
# This gets the temperature for that entry
    my $inTemp='sqlite3 /home/weewx/archive/weewx.sdb "select inTemp
from archive where dateTime=${dateTime};"';

    my $outTemp = 'sqlite3 /home/weewx/archive/weewx.sdb "select
outTemp from archive where dateTime=${dateTime};"
';

    my $outHumidity = 'sqlite3 /home/weewx/archive/weewx.sdb "select
outHumidity from archive where dateTime=${dat
eTime};"';
    chomp($inTemp);
    chomp($outTemp);
    chomp($outHumidity);
```

• vérifier que ces valeurs sont valides, e.g. :

```
if ($inTemp < -20 || $inTemp > 50) {
 debug_log("Something wrong with inner temperature: $inTemp");
 $inTemp = '?';
    }
```

• obtenir l'état de la VMC via une requête au site web de la VMC (RPi2) :

```
$res = Hijk::request({
     method       => "GET",
     host         => "$relayserver_url",
     port         => "$relayserver_port",
     path         => "$path"});
```

• extraire de la requête HTTP au site web du RPi2 les valeurs des relais, et donc la vitesse en cours de la VMC :

```
sub read_cmv_reply {
    my $page = shift;
    debug_log("--> read_cmv_reply()\n");
    my $speed;
```

```
    if ($page eq '?') {
 return '?';
    }

    if ($page =~ /Relay #1: ON/i) {
 if ($page =~ /Relay #2: ON/i) {
     $speed = 2;
 } else {
     $speed = 1;
 }
    } else {
 $speed = 0;
    }

    debug_log("<-- read_cmv_replay(): speed=$speed\n");
    return $speed;
}
```

- le **main** enchaîne toutes les fonctions précédentes et ajoute les valeurs à la fin d'un fichier de données :

```
# --------- MAIN ----------------------------------------
my $output_file;

GetOptions ("verbose" => \$debug
     )
    or usage();

# read current weather data
my ($intemp, $outtemp, $outhumidity) = read_weather();

# read VMC speed
my $speed = get_status();

# read time
my $mytime = get_time();

print "$mytime $speed $outtemp $intemp $outhumidity\n";
```

Ce fichier de données comprend typiquement les valeurs suivantes :

```
$ more times_vmc_generated.data
11:27:05:01:01 0 3.79411764705882 17.4 43.6470588235294
...
```

selon le format Mois:jour:heure:minutes:secondes température_extérieure température_intérieure humidité_extérieure.

Le script **gnuplot** utilise deux échelles différentes : axe y1 pour la vitesse de la VMC et l'axe y2 pour les valeurs de température et d'humidité. Ensuite, il trace les courbes en fonction des valeurs du fichier de données :

```
...
set yrange [0:2.5]
set y2range [0:100]
set y2tics 10
set grid
set xlabel "Date and time"
set ylabel "Speed"
set y2label "Temperature and humidity"
set title "VMC"
set key outside right box
plot "/usr/share/vmc/curve/times_vmc_generated.data" using 1:2 index 0
title "speed" with lines ls 2, '' using 1:3
 axes x1y2  smooth bezier with lines title "Outside temp" ls 3, '' using
1:4 axes x1y2  smooth bezier with lines t
itle "Inside temp" ls 4, '' using 1:5 axes x1y2 smooth bezier with lines
title "Outside humidity" ls 5
```

Enfin, un script Bash lie le tout : il lance le script Perl de génération des valeurs, puis génère un graphe à partir des 250 dernières valeurs générées, et enfin copie l'image générée par Gnuplot au niveau du site Internet utilisateur :

```
dir=/usr/share/vmc/curve
${dir}/curve_vmc.pl >> ${dir}/times_vmc_generated.data
tail -n 250 ${dir}/times_vmc_generated.data > ${dir}/.tmp-vmc
mv ${dir}/.tmp-vmc ${dir}/times_vmc_generated.data
gnuplot ${dir}/plot_vmc.gp
mv ${dir}/vmc.png /var/www/images
```

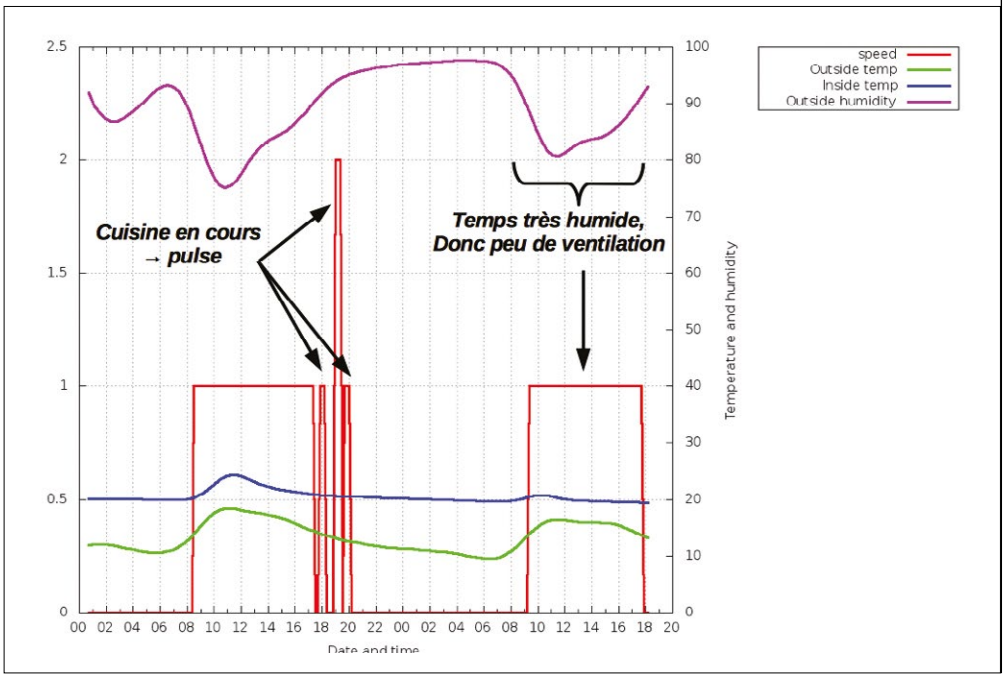



*Fig. 6 : Courbe de fonctionnement de la VMC.*

Cette tâche est effectuée toutes les 10 minutes, via la crontab.

```
$ crontab -e
*/10 * * * * /usr/share/vmc/curve/curve_script &> /dev/null
```

## 6. PISTES D'AMÉLIORATIONS ET CONCLUSION

L'article a montré comment éviter de faire rentrer de l'air « inadapté » (trop humide, froid, chaud) dans le logement. Néanmoins, il reste certains cas que l'automatisation ne résout pas, et qui nécessitent ainsi l'intervention de l'utilisateur, via le site Internet de la VMC, ou via les interrupteurs de commande manuelle. Voici quelques-uns de ces cas, et des pistes pour les résoudre.

- Évacuation des mauvaises odeurs (cuisine – lol la cuisinière est offusquée, etc.) : il faudrait bien entendu disposer de capteurs pour détecter ce problème. À défaut de disposer de tels capteurs, une idée serait sans aucun doute de disposer de capteurs de mouvement, par exemple, au niveau de la plaque de cuisson et du four, et dans les WC. La présence d'une personne déclencherait alors la VMC tant que la personne est présente, et pour une durée prédéterminée lorsque plus aucun mouvement n'est détecté.
- Présence d'humidité excessive après, par exemple, une douche, ou une cuisson « vapeur », ou encore après avoir passé la serpillère. La présence de nombreux invités est aussi une source d'humidité (évidemment, un pur geek n'invite personne, ne prend pas de douche et ne passe pas la serpillère ?). Là, un simple capteur d'humidité intérieur peut aider à détecter le problème – par exemple, par la montée rapide du taux d'humidité dans le logement – et à déclencher la VMC. La vitesse de la VMC peut être adaptée en fonction du taux de montée de ce taux d'humidité.
- Présence de pollution et/ou odeur à l'extérieur du logement. L'on peut citer par exemple le cas où un voisin fait brûler des végétaux à proximité de son logement. Dans ce cas, il faut disposer d'un capteur de pollution (taux de CO2 qui monte subitement, capteur de particules fines) qui entraîne l'arrêt de la VMC. Au passage, à l'heure de la COP21, on vous signale qu'il vaudrait mieux broyer ces déchets ;)
- N'hésitez pas à nous envoyer d'autres idées !

Reste qu'en l'état actuel, nous intervenons en fait très peu sur la VMC au quotidien. La facilité d'intervention pour forcer la VMC en dehors de son mode automatique fait que nous n'avons pas l'intention pour l'instant de nous équiper d'autres capteurs. La seule amélioration possible actuellement serait de disposer d'un panneau de contrôle avec des boutons physiques comprenant une diode – un peu comme dans les centrales nucléaires ou les sous-marins – qui permettrait de visualiser, sans son ordinateur ou son téléphone mobile, l'état de la VMC. Cela sera pour un futur numéro de *Hackable* ;-) **LAA**

## RÉFÉRENCES

**[1] http://www.developpement-durable.gouv.fr/Aeration-Ventilation,12909.html**

**[2] https://git.framasoft.org/axellec/hackable-vmc**

RENDEZ-VOUS SUR :

# www.ed-diamond.com




## ENVIE DE COMPLÉTER VOTRE COLLECTION ?

**Retrouvez tous les anciens numéros de Hackable !** (SELON LES DISPONIBILITÉS DU STOCK)

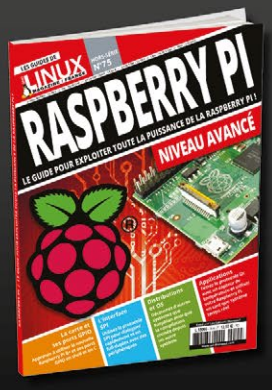



...

# EN PROMO ACTUELLEMENT !

## RASPBERRY PI

Exploitez toute sa puissance !




PRIX NORMAL

~~12,90 €~~

PRIX PROMO

**6,90€**

## RASPBERRY PI ET PYTHON

Exploitez toute la puissance de la Raspberry Pi et devenez un véritable expert du langage Python !




PRIX NORMAL

~~25,80 €~~

PRIX PROMO

**19€**

# GÉREZ EFFICACEMENT VOS INSTALLATIONS/ DÉSINSTALLATIONS DE LOGICIELS DANS RASPBIAN

*Denis Bodor*

**Faire connaissance avec la Raspberry Pi consiste souvent à la découverte d'un tout nouveau monde, en particulier lorsqu'on ne s'est jamais frotté à un système GNU/Linux comme Ubuntu ou Debian sur PC ou Mac. Les applications livrées avec Raspbian et installées par défaut facilitent grandement les premiers pas, mais arrive toujours le moment où il devient nécessaire de creuser davantage, et cela passe par l'installation/désinstallation de logiciels. Autrement dit, la gestion de paquets !**

La gestion de paquets est le cœur et la raison d'être d'une distribution GNU/Linux. Contrairement aux autres systèmes d'exploitation que sont Windows et Mac OS X, où respectivement on utilise un programme d'installation type `setup.exe` (oui, Windows a aussi ses fichiers MSI), où on copie simplement le programme dans le dossier « Applications », avec Raspbian les logiciels s'installent sous forme de paquets.

Ceux-ci sont construits à partir des sources créées par les développeurs à travers le monde. Il peut s'agir de petits projets autour d'un applicatif unique (comme le lecteur Xpdf), de projets plus importants comme le très populaire éditeur Vim ou d'organisations et fondations massives comme GNU, Mozilla, GNOME, Xorg, etc. Dans tous les cas, les responsables de paquets de chaque distribution utilisent ces sources pour créer des paquets correspondants, faciles à installer et gérer par les utilisateurs. Les paquets proviennent donc des personnes responsables de la distribution et non des projets dit « amont » créant les programmes (tantôt il peut s'agir des mêmes personnes, mais cela reste relativement rare).

## 1. PRINCIPES

Quelle que soit la distribution, un paquet est un fichier contenant un ou plusieurs programmes, des fichiers de données complémentaires, des exemples, des documentations ou des scripts. Mais ce n'est pas tout, il embarque également des informations importantes pour la distribution comme la plateforme concernée, la liste d'autres paquets nécessaires à son fonctionnement, un journal des changements faits, etc. Ce à quoi s'ajoutent également quelques scripts chargés de déclencher des actions, avant et après l'installation ou la désinstallation.

Les paquets sont mis à disposition sur Internet, organisés par types, par plateformes ou encore par versions de la distribution. Ils sont indexés de manière à permettre une recherche rapide. Dans le cas de Raspbian, les paquets se trouvent sur **raspbian.org** et plus précisément dans **http://mirrordirector.raspbian.org/raspbian/dists/** où sont stockés des dizaines de milliers de fichiers `.deb`, les fameux paquets.

Chaque installation de Raspbian (via une image SD ou NOOBS) intègre un certain nombre de logiciels et bibliothèques, mais ceux-ci sont également des paquets. Ils ont simplement été préinstallés dans votre système, peuvent être désinstallés, réinstallés ou mis à jour.

Si vous pointez votre navigateur sur le site en question, vous remarquerez que les dates accompagnant les répertoires et les fichiers sont excessivement récentes. En effet, les nouvelles versions des programmes intègrent le site et les paquets sont régulièrement reconstruits (en fonction de la politique de développement de la distribution).

Votre système sur Raspberry Pi dispose d'une copie de la liste des paquets, mais c'est à vous de la mettre à jour régulièrement pour être sûr de référencer les bonnes versions des paquets. La liste contient bien plus que des informations de base. On y trouve : le nom du paquet/logiciel, sa version, la plateforme sur laquelle il peut s'installer, le nom du créateur, diverses informations de classement et de catégorisation, des sommes de contrôle pour vérifier l'intégrité des données, une description ou encore, et c'est important, la liste des autres paquets qui sont nécessaires pour faire fonctionner le contenu de celui-ci : ses **dépendances**. Si l'on prend l'exemple du paquet `xpdf`, celui-ci a besoin de `lesstif2`, de `libpoppler19`, etc. Le tout avec des versions minimums. Mais ceci peut rapidement se compliquer, car le paquet `lesstif2`, lui, a besoin de `libfreetype6`, qui lui-même nécessite `zlib1g`, qui...

Rassurez-vous, vous n'êtes même pas censés voir le contenu de la liste. C'est pour cette raison qu'un ensemble d'outils spécifiques à chaque distribution permet de faciliter

la gestion des dépendances et d'automatiser les installations ainsi que la mise à jour. Dans le cas de Raspbian, qui est une adaptation de Debian à la Raspberry Pi, l'ensemble de l'infrastructure de gestion des paquets et les outils permettant les manipulations reposent sur APT (pour *Advanced Package Tool*). Ce sont ces outils et commandes que nous allons détailler à présent (ainsi que l'utilitaire sous-jacent).

## 2. COMMANDES DE BASE

Si vous avez suivi les tutoriels et autres explications que l'on trouve sur le Web à l'attention des utilisateurs de Raspberry Pi, vous devez au moins une fois avoir vu les commandes suivantes :

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

et je l'espère pour vous, les avez déjà utilisées. Mais que font-elles exactement ?

**apt-get** est l'outil principal de gestion des paquets (oui, il existe aussi **aptitude** qui est un outil de plus haut niveau). Il s'utilise en spécifiant une commande (**update** par exemple) et éventuellement des options. Nous utilisons ici deux commandes :

- **update** ordonne au système de mettre à jour la liste des paquets. Celle-ci est alors récupérée aux différents emplacements spécifiés dans la configuration (qui sont listés lorsqu'on valide la commande). Après cette opération, la liste sur la Raspberry Pi est synchronisée par rapport aux paquets disponibles, mais rien n'est installé ou désinstallé sur le système.
- **upgrade** va modifier votre installation. Sur la base de la liste fraîchement récupérée, **apt-get** va procéder à une comparaison des numéros de version entre les paquets actuellement installés et ceux disponibles dans la liste. Si une version plus récente pour un ou plusieurs paquets est disponible, le ou les paquets seront téléchargés et installés. C'est donc une opération de mise à jour.

Notez qu'il existe également la commande **dist-upgrade** qui se différencie de **upgrade** par le fait que, si nécessaire, le système peut retirer des paquets présents et en installer de nouveaux. Ceci arrive tantôt en cas de changement de dépendance ou de nom de paquet. Un **apt-get upgrade** ne supprimera **jamais** un paquet et n'en installera jamais un nouveau. Si la situation se présente pour un paquet, la version installée restera inchangée.

Pour maintenir très simplement son système à jour de la façon la plus équilibrée possible, on enchaîne généralement **update**, **upgrade** et **dist-upgrade**.

### 2.1 Installation et désinstallation

Installer un nouveau paquet se fait très simplement avec un **sudo apt-get install** suivi du nom du paquet à installer. Et là, vous vous demandez normalement : « mais je le sors d'où ce nom ? ». La plupart du temps, on installe un paquet pour disposer d'une nouvelle application ou d'un nouvel outil, parce qu'on a lu quelque part que c'est celui qu'il faut installer. Parfois, le nom du paquet est clairement spécifié et il suffit de le préciser.

D'autres fois en revanche on est un peu seul et on n'a trouvé aucune mention du nom du paquet. Mais rassurez-vous, car **apt-cache** est votre ami. Cet utilitaire vous permet de chercher et d'obtenir

des informations sur un paquet. Imaginez, par exemple, que vous soyez à la recherche d'un clone de ce bon vieux Norton Commander DOS. Une recherche sur le Web vous parle d'un *Midnight Commander* très populaire. Vous vous tournez alors vers **apt-cache** et :

```
$ apt-cache search midnight
avfs - virtual filesystem to access archives, disk images, remote locations
gkrellmitime - Internet time plugin for gkrellm
gnome-commander - nice and fast file manager for the GNOME desktop
gnome-commander-dbg - Debugging symbols for gnome-commander
krusader - twin-panel (commander-style) file manager
mc - Midnight Commander - a powerful file manager
mc-data - Midnight Commander - a powerful file manager -- data files
mc-dbg - Midnight Commander - a powerful file manager - debug package
moc - ncurses based console audio player
pilot - Simple file browser from Alpine, a text-based email client
tuxcmd - twin-panel (commander-style) file manager using GTK+ 2
```

Et voilà ! Non seulement vous avez trouvé Midnight Commander sous la forme du paquet **mc** (6ème ligne), mais en plus vous avez droit à quelques alternatives. **apt-cache** est un outil relativement rudimentaire et sa commande **search** procédera à la recherche sur les descriptifs courts (listés ici) et longs, mais cela s'arrête là. Il existe d'autres outils (**aptitude**, **axi-cache**, etc.) qui permettent des sélections plus poussées.

À propos de descriptifs justement, **apt-cache** vous permet également d'afficher les informations sur un paquet afin de vous assurer que c'est bien ce que vous voulez installer :

```
$ apt-cache show mc
Package: mc
Version: 3:4.8.3-10
Architecture: armhf
Maintainer: Debian MC Packaging
  Group <pkg-mc-devel@lists.alioth.debian.org>
Installed-Size: 1133
Depends: e2fslibs (>= 1.42.2), libc6 (>= 2.13-28),
  libcomerr2 (>= 1.01), libgcc1 (>= 1:4.4.0),
  libglib2.0-0 (>= 2.24.0), libgpm2 (>= 1.20.4),
  libslang2 (>= 2.2.4), mc-data (= 3:4.8.3-10)
Recommends: mime-support, unzip, perl
Suggests: zip, bzip2, links | w3m | lynx, arj, file,
  xpdf | pdf-viewer, dbview, odt2txt, gv, catdvi,
  djvulibre-bin, imagemagick, python, python-boto, python-tz
Homepage: http://www.midnight-commander.org
Priority: optional
Section: utils
Filename: pool/main/m/mc/mc_4.8.3-10_armhf.deb
Size: 404848
SHA256: 641eadfd8893c68c12740c50dfb578d27fe4faf33738442b7421f6a2a66b491f
SHA1: 2e3ef72ce1eb964ce647a0483e054d174f18838a
MD5sum: 0be1ca5dc5596d58c351122e37447af1
Description: Midnight Commander - a powerful file manager
GNU Midnight Commander is a text-mode full-screen file manager. It
uses a two panel interface and a subshell for command execution. It
```

```
includes an internal editor with syntax highlighting and an internal
viewer with support for binary files. Also included is Virtual
Filesystem (VFS), that allows files on remote systems (e.g. FTP, SSH
servers) and files inside archives to be manipulated like real files.
```

Il s'agit grossièrement de l'affichage des informations contenues dans la liste des paquets avec, comme information importante, le champ **Depends:** (« dépendances »). Les paquets listés ici seront automatiquement installés si vous ordonnez l'installation de **mc**. **Recommends:** est également intéressant puisqu'il liste des paquets qui ne seront pas installés automatiquement, mais sont fortement recommandés pour profiter au maximum des fonctionnalités liées au contenu du paquet. Et enfin, nous avons **Suggests:** qui comprend une liste de paquets qui généralement s'accordent bien avec celui que vous voulez installer.

L'installation à proprement parler se fera avec **sudo apt-get install** suivi du nom du paquet. Si des dépendances existent et donc que des paquets supplémentaires doivent être installés, **apt-get** vous demandera de confirmer l'opération. Dans le cas contraire, le paquet est simplement installé immédiatement.

Supprimer un paquet n'est pas plus compliqué puisqu'il suffit de changer **install** en **remove**. Là, en revanche la confirmation sera systématique puisque c'est une opération non sans conséquence.

Notez que les dépendances installées avec **mc** dans notre exemple, comme **mc-data** (principalement les fichiers de traduction du logiciel) ne sont pas automatiquement désinstallées. Mais **apt-get** vous le rappellera ensuite lors d'autres opérations. Exemple :

```
$ sudo apt-get install
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Le paquet suivant a été installé automatiquement
et n'est plus nécessaire :
  mc-data
Veuillez utiliser " apt-get autoremove " pour le supprimer.
0 mis à jour, 0 nouvellement installés, 0 à enlever
et 0 non mis à jour.
```

Vous pouvez utiliser **sudo apt-get remove mc-data**, mais il est plus simple de laisser faire le système avec **sudo apt-get autoremove**. **mc-data** n'était utile que pour **mc** et au fil des installations/désinstallations de paquets, ce type de dépendances inutiles a tendance à s'accumuler. **autoremove** permet de rapidement régler le problème et de faire un brin de ménage.

En parlant de ménage justement, il est probable que l'installation par défaut de Raspbian comprenne des paquets d'applications, d'outils et de bibliothèques dont vous ne vous servirez peut-être jamais. Vous pouvez lister les paquets actuellement installés avec la commande :

```
$ dpkg -l
Souhait=inconnU/Installé/suppRimé/Purgé/H=à garder
| État=Non/Installé/fichier-Config/dépaqUeté/échec-conFig/
  H=semi-installé/W=attend-traitement-déclenchements
|/ Err?=(aucune)/besoin Réinstallation (État,Err: majuscule=mauvais)
```

```
||/ Nom           Version          Architecture Description
+++-=============-================-============-=================================
ii  adduser       3.113+nmu3       all          add and remove users and groups
ii  alsa-base     1.0.25+3~deb7u1  all          ALSA driver configuration files
[...]
```

La liste est longue, plus de 1000 paquets dans une installation standard par défaut. Parmi les informations affichées, vous avez le nom du paquet bien sûr, mais aussi sa version et un court descriptif. La première colonne est importante. **ii** par exemple indique que le paquet est installé, mais vous pouvez également voir des paquets marqués **rc**. Exemple :

```
$ dpkg -l | grep Midnight
rc  mc    3:4.8.3-10  armhf  Midnight Commander - a powerful file manager
```

**grep** permet ici de limiter l'affichage aux lignes contenant le texte spécifié. Le résultat est surprenant, car nous venons tout juste de désinstaller **mc**, il ne devrait pas être dans la liste. **rc** en lieu et place de **ii** indique que le paquet n'est pas vraiment installé, mais que ses fichiers de configuration sont toujours présents. Et en effet :

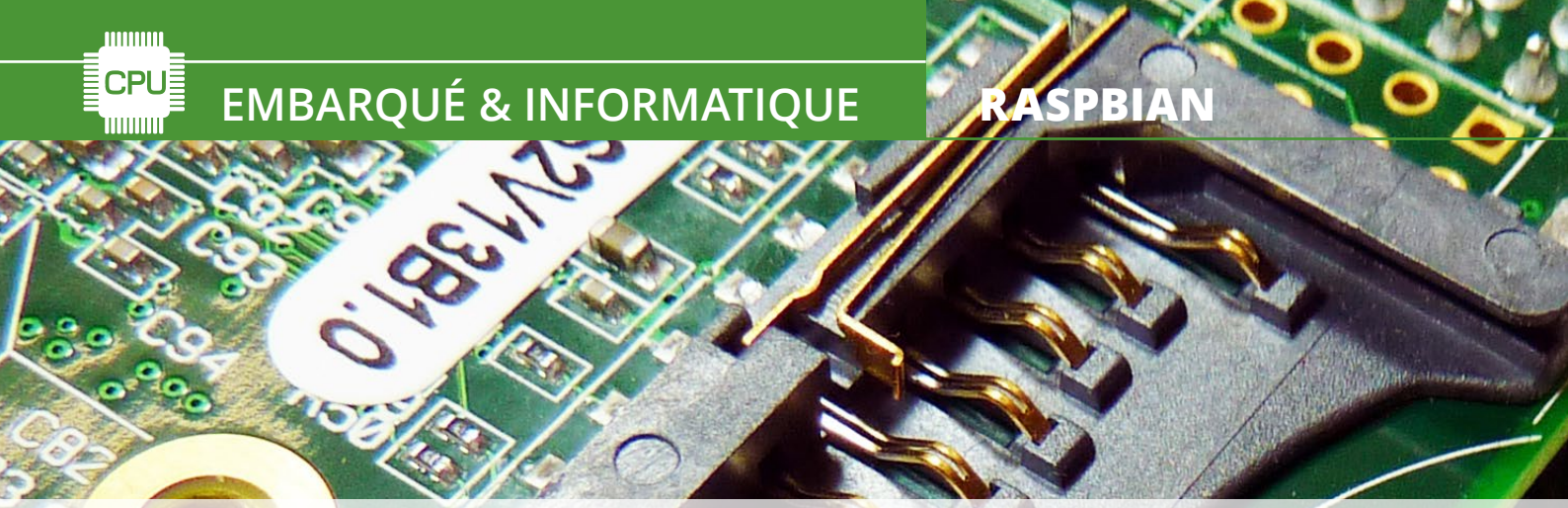
```
$ ls /etc./mc/
edit.indent.rc  edit.spell.rc  filehighlight.ini
mcedit.menu  mc.ext  mc.keymap  mc.keymap.default
mc.keymap.emacs  mc.menu  mc.menu.sr  sfs.ini
```

L'opération **remove** provoque la désinstallation du contenu du paquet, mais non de ses éventuels fichiers de configuration. Ceci permet de se réserver la possibilité d'une réinstallation, avec une configuration identique. En particulier si d'aventure vous avez modifié la configuration en question. Pour supprimer un paquet avec sa configuration, il faudra plutôt utiliser **sudo apt-get purge** et non simplement **remove**. Notez toutefois que les configurations n'occupent généralement pas une place importante et qu'il est souvent bon de se ménager une solution de rattrapage en cas d'erreur de manipulation. Sachez, de plus, que vous pouvez utiliser **purge** pour retirer toutes traces du paquet même après un **remove**. Lister tous les paquets dans cet état se fera avec un simple **dpkg -l | grep "^ii"**.

Enfin pour terminer avec le nettoyage et les commandes de base, parlons de **sudo apt-get clean**. Lorsque vous installez de nouveaux paquets, ceux-ci sont téléchargés et stockés temporairement dans le système. Il en va de même pour les mises à jour qui sont, tout bonnement, des installation de nouvelles versions. Au fil du temps, ces fichiers s'accumulent et finissent par occuper un espace conséquent. **clean** permet simplement de supprimer tous les paquets temporairement stockés afin de gagner de la place. Ce qui peut être bien utile, surtout avec un système sur carte SD...

## 3. C'EST TOUT POUR L'INSTANT

La gestion de paquets est un vaste domaine puisqu'il s'étend de l'installation/désinstallation jusqu'à la création de nouveaux paquets à partir de zéro et à sa soumission dans la distribution. Nous nous sommes ici délibérément restreints aux opérations de base vous permettant de sereinement gérer vos paquets. Nous verrons dans un autre article comment pousser plus avant les manipulations, gérer les dépendances, comprendre les relations entre fichiers et paquets, trouver rapidement le paquet souhaité, etc. **DB**



# GÉREZ VOS INSTALLATIONS/ DÉSINSTALLATIONS DE LOGICIELS DANS RASPBIAN : LES COMMANDES AVANCÉES

*Denis Bodor*

**L'ajout et la suppression de logiciels et de paquets avec le système le plus populaire sur la Raspberry Pi sont des activités de base sinon quotidiennes. Dans l'article précédent, nous n'avons en réalité qu'effleuré la surface des fonctionnalités qui s'offrent à vous dès lors que vous vous penchez sérieusement sur le sujet. Il est temps maintenant de regarder plus avant et de satisfaire des besoins plus spécifiques et ponctuels.**

La mécanique de fonctionnement des paquets, de ce qu'ils contiennent et des dépendances existantes entre eux est entièrement prise en charge par la distribution. La plupart du temps, vous n'avez pas vraiment besoin de chercher plus loin. Mais il arrive dans certains cas qu'on se pose des questions ou qu'on cherche des points précis. Pour répondre à ces besoins, ce sont les mêmes outils (**apt-get** et **dpkg**) qui sont utilisés, mais de façon un peu plus avancée, et tantôt des utilitaires plus spécialisés. C'est ce dont nous allons parler dans ce qui suit.

## 1. DES PAQUETS ET DES FICHIERS

Les paquets contiennent des fichiers, des répertoires et plein d'autres choses. Après avoir installé un paquet, il est tantôt utile de savoir ce qui a effectivement été changé dans le système. Pour ce faire, **dpkg** peut être utilisé avec l'option **-L** :

```
dpkg -L mc
[...]
/usr/share/doc/mc/copyright
/usr/share/doc/mc/NEWS.Debian.gz
/usr/share/doc/mc/README.Debian
/usr/bin
/usr/bin/mc
/usr/lib
/usr/lib/mc
/usr/lib/mc/mc-wrapper.sh
[...]
```

Bien entendu, on s'empressera de rediriger cette longue liste vers l'outil **grep** afin, par exemple de savoir quel programme (exécutable binaire) peut être lancé :

```
$ dpkg -L mc | grep "/usr/bin/"
/usr/bin/mc
/usr/bin/mcedit
/usr/bin/mcdiff
/usr/bin/mcview
```

Ceci est fort pratique, par exemple, lorsque le paquet ne se nomme pas du tout comme la ou les commandes qu'il installe. Inversement, on peut également avoir besoin de connaître le nom du paquet ayant installé une commande. Pour cela, la première chose à faire est de trouver le fichier exécutable correspondant. À cet effet, le système fournit gentiment l'outil **which** :

```
$ which mc
/usr/bin/mc
```

On combinera cela avec la commande **dpkg -S** pour obtenir :

```
$ dpkg -S 'which mc'
mc: /usr/bin/mc
```

Notez l'utilisation de l'apostrophe inverse (AltGr+è) permettant d'utiliser le résultat d'une commande en guise d'argument.

Bien entendu, **dpkg -S** ne fonctionne pas seulement avec les exécutables. Il est possible d'utiliser l'option **-S** pour chercher une simple chaîne dans la liste de tous les fichiers installés par tous les paquets et de retrouver les paquets correspondants. Exemple, si on me parle d'un fichier **mc.lib**, je ne sais pas où il se trouve et ni par quel paquet il est installé, je fais :

```
$ dpkg -S mc.lib
mc-data: /usr/share/mc/mc.lib
```

Il se trouve dans **/usr/share/mc** et appartient à **mc-data** (et non **mc**).

Ceci ne fonctionne qu'avec les paquets installés. Pour faire de même dans l'ensemble des paquets, y compris ceux que vous ne possédez pas, il faudra installer **apt-file** qui est un outil spécialisé pour cette tâche. **apt-file** n'utilise pas la liste de paquets standards (car elle ne contient pas l'information nécessaire). Immédiatement après l'installation, vous devrez vous plier d'un **apt-file update** pour mettre à jour la liste des fichiers. Notez que cette commande ne nécessite pas l'utilisation de **sudo**, vous pouvez travailler entièrement en tant qu'utilisateur standard.

Vous voulez connaître la liste des paquets qui installent quelque chose dans le répertoire **/usr/lib/apache** ? Il suffit de demander :

```
$ apt-file search -l /usr/lib/apache
apache2-dbg
apache2-suexec
apache2-suexec-custom
[...]
libapache2-webauth
libapache2-webkdc
php5-dbg
```

Quant à lister les fichiers d'un paquet avant de l'installer, rien de plus simple :

```
$ apt-file list -F gridsite
[...]
gridsite: /var/lib/gridsite/gridsitefoot.txt
gridsite: /var/lib/gridsite/gridsitehead.txt
```

À l'aide de ces quelques commandes, vous êtes maintenant en mesure de savoir précisément, à partir d'un simple nom de fichier, quels paquets installer ou désinstaller.

## 2. DÉPENDANCES À TOUT VA

Nous avons vu précédemment que la gestion des dépendances entre paquets est la clé de voûte du système de gestion. Tout ceci est géré pour vous et vous avez même l'opportunité de faire un brin de ménage de temps à autre. Mais les commandes vues pour l'instant ne permettent pas de répondre à deux importantes questions :

- quels paquets dépendent d'un paquet donné,
- quelle est la chaîne de dépendance complète pour un paquet (dépendances de dépendances, etc.).

Les outils de base permettent de répondre sommairement aux deux questions et en particulier **apt-cache**. Avec la commande **depend** nous affichons les dépendances :

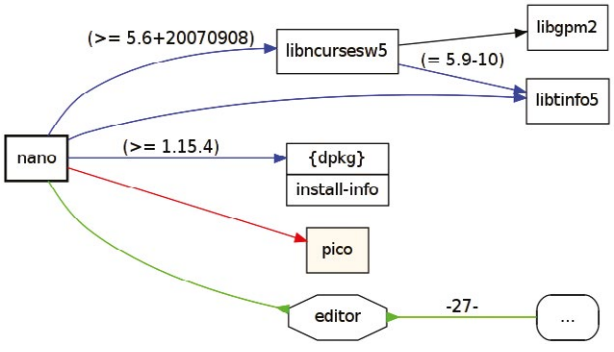
```
$ apt-cache depends mc
mc
  Dépend: e2fslibs
  Dépend: libc6
  Dépend: libcomerr2
  Dépend: libgcc1
  Dépend: libglib2.0-0
  Dépend: libgpm2
[...]
  Suggère: python-tz
  Recommande: mime-support
  Recommande: unzip
  Recommande: perl
```

Avec **rdepend**, nous procédons à l'opération inverse, ici la question est « qui a besoin de **libgpm2** » :

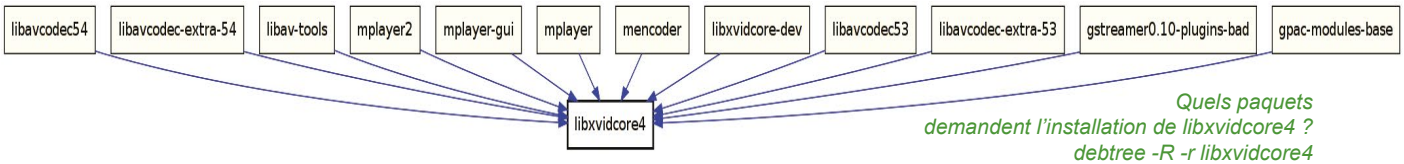
```
libgpm2
Reverse Depends:
  zhcon
  xwpe
  xemacs21-nomule
  xemacs21-mule-canna-wnn
[...]
  brltty
  aumix-gtk
  aumix
```

Ceci nous donne la liste complète indépendamment du fait qu'un paquet soit installé ou non. On peut cependant restreindre aux paquets installés :

```
$ apt-cache rdepends --installed libgpm2
libgpm2
Reverse Depends:
  vim
  mc
  libaa1
```




*L'éditeur nano demande l'installation de quelques autres paquets (bleu), entre en conflit avec pico (rouge) et fournit un paquet virtuel « editor » (vert). L'installation de nano provoquera la désinstallation de pico, et inversement. Un paquet ayant besoin de « editor » considérera la dépendance satisfaite si nano est installé.*




*Quels paquets demandent l'installation de libxvidcore4 ? debtree -R -r libxvidcore4*

Nous savons maintenant que si nous voulons nous débarrasser de **libgpm2**, il faudra également faire une croix sur **mc**, **vim** et un autre paquet (une bibliothèque).

Il est possible d'aller un peu plus loin qu'avec **apt-cache** en faisant usage de **apt-rdepend** qui nous apportera un peu plus d'informations :

```
$ apt-rdepends --reverse \
--state-follow=Installed --state-show=Installed libgpm2
Reading package lists... Done
Building dependency tree
Reading state information... Done
libgpm2
  Reverse Depends: libaa1 (>= 1.4p5-40)
  Reverse Depends: mc (>= 3:4.8.3-10)
  Reverse Depends: vim (>= 2:7.3.547-7)
libaa1
  Reverse Depends: gstreamer1.0-plugins-good (>= 1.2.0-1co6rpi3)
gstreamer1.0-plugins-good
mc
vim
```

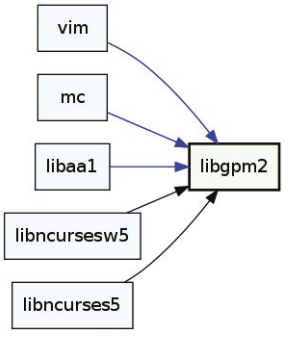
Là on constate qu'en supprimant **libgpm2**, nous allons provoquer la suppression des trois paquets précisés lors de la commande précédente, mais également que l'un de ces paquets, **libaa1**, est également nécessaire à un autre. **gstreamer1.0-plugins-good** va donc également être désinstallé, ce qui n'est pas forcément souhaitable.

En ce qui concerne le fait d'obtenir une représentation plus intelligible des dépendances, le plus simple est de faire appel à un autre outil : **debtree**. Comme son nom semble le suggérer, il permet de produire un arbre de dépendances et plus précisément un graphique. Il vous faudra installer **debtree** et **graphviz**, permettant de créer une image à partir d'une description du graphique.

**debtree** ne retourne pas d'information directement lisible par l'utilisateur, mais des données en langage *dot* utilisées par Graphviz. Je vous parlais précédemment de paquets fournissant des outils d'un nom différent : Graphviz ne propose pas de commande **graphviz** mais, entre autres, une s'appelant **dot** (pour les autres, je pense que vous savez comment faire maintenant). **dot** utilise des données en langage *dot* et en fait une image dans un format de votre choix (PNG pas exemple). Ainsi, si nous voulons une représentation des dépendances de l'éditeur **nano**, nous pouvons faire :

```
$ debtree nano | dot -T png > graph_nano.png
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
I: The following dependencies have been excluded
   from the graph (skipped):
I: libc6 multiarch-support
```

Nous obtenons une image parfaitement claire de la situation sous la forme du fichier **graph_nano.png**. Si vous travaillez en mode graphique sur votre Raspberry Pi, vous pouvez également rediriger la sortie sur la commande **display** : **debtree nano | dot -T png | display** (vous n'avez pas la commande **display** ? Peut-être pouvez-vous demander conseil à **apt-file**).




*Quels paquets actuellement sur mon système ont besoin de libgpm2 ? debtree -R -l libgpm2*

Soyons clairs, courir après les dépendances n'est pas une activité récurrente pour un utilisateur, mais c'est très agréable de comprendre et constater les interactions et la complexité d'un système comme Raspbian. L'une des motivations légitimes, en dehors de la simple curiosité (qui est parfaitement légitime) dans l'utilisation de ces outils est la quête de la réduction maximum de la complexité du système. Pas d'applications et de bibliothèques inutiles présentes, signifie moins de paquets et donc moins de paquets à mettre à jour, mais aussi plus d'espace libéré sur la carte SD.

Astuce bonus : vous voulez vraiment faire de la place dans votre système et voulez savoir quels sont les paquets qui occupent le plus de place ? Essayez **`dpkg-query -Wf '${Installed-Size}\t${Package}\n' | sort -n`**. Vous pouvez aussi utiliser **`dpigs -H`** du paquet **`debian-goodies`**.

## 3. RECHERCHES AVANCÉES

Trouver son bonheur dans plus de 36000 paquets disponibles n'est pas une tâche facile, en particulier lorsqu'on ne connaît pas précisément le nom de ce qu'on cherche. Bien sûr, vous avez la possibilité de rechercher sur le Web des choses comme « meilleur éditeur sous Linux », trouver quelques noms pour ensuite les rechercher dans la liste des paquets. Mais il existe une solution plus efficace.

La liste des paquets tenue à jour avec **`apt-get update`** intègre, pour la plupart d'entre eux, des tags définis selon un principe de classification à facette. Un paquet peut contenir par exemple un élément (programme, bibliothèque, données, etc.) définit par un rôle, un environnement dans lequel il fonctionne, ce avec quoi il a été créé, ce à quoi il sert, etc. Tout ceci est référencé dans les tags d'un paquet (pour la plupart d'entre eux).

Pour profiter au maximum de cette fonctionnalité, vous pouvez utiliser **`aptitude`** (installé par défaut) qui peut être vu comme une version plus « intelligente » de **`apt-get`** combinant diverses fonctionnalités, y compris celles fournies par **`apt-cache`**. Mieux vaut cependant se tourner vers l'outil spécialisé appelé **`debtags`** permettant à la fois de chercher les paquets, mais aussi les tags eux-mêmes.

Imaginons que vous soyez à la recherche d'un éditeur pour écrire vos programmes ou scripts. La première chose à faire sera de trouver le tag correspondant :

```
$ debtags tagsearch editor
devel::editor - Source Editor
field::electronics - Electronics
```

**`devel::editor`** semble être ici un choix évident. Nous pouvons alors l'utiliser pour procéder à une première recherche et répondre à la question « quels sont les éditeurs de code disponibles ? » :

```
$ debtags search "devel::editor"
cream - VIM macros that make the VIM easier to use for beginners
drpython - simple and customizable editor for the Python language
[...]
```

44 réponses, c'est beaucoup pour faire un choix. Je veux que mon éditeur s'utilise dans une interface graphique (c'est une recherche hypothétique, mon éditeur est et sera toujours Vi). Je dois donc trouver le tag correspondant : **`debtags tagsearch interface`**. Ceci me remonte plusieurs choix dont **`interface::x11 - X Window System`**. C'est parfait ! Je vais pouvoir préciser ma recherche :

```
$ debtags search "devel::editor && interface::x11"
eclipse-cdt - C/C++ Development Tools for Eclipse
eclipse-jdt - Eclipse Java Development Tools (JDT)
editra - simple multi-platform text editor
eric - full featured Python IDE
[...]
```

Nous voici arrivés à 17 résultats. C'est bien plus concis, mais je n'ai pas vraiment envie de tester toutes ces possibilités. J'aime bien l'interface de l'environnement de bureau LXDE installé par défaut, voyons voir quels sont ses tags :

```
$ debtags tag ls lxde
interface::x11
role::metapackage
scope::suite
suite::TODO
uitoolkit::gtk
```

**uitoolkit** est le « jeu de composants graphiques » (*toolkit*) utilisé pour l'interface utilisateur (UI comme *user interface*). C'est donc GTK qui est utilisé. Je peux maintenant restreindre davantage mon champ de recherche :

```
$ debtags search "devel::editor && interface::x11 && uitoolkit::gtk"
eclipse-cdt - C/C++ Development Tools for Eclipse
editra - simple multi-platform text editor
gecrit - simple, easy-to-use Python IDE
gwrite - simple GTK+ HTML5 rich text editor
xemacs21 - highly customizable text editor
```

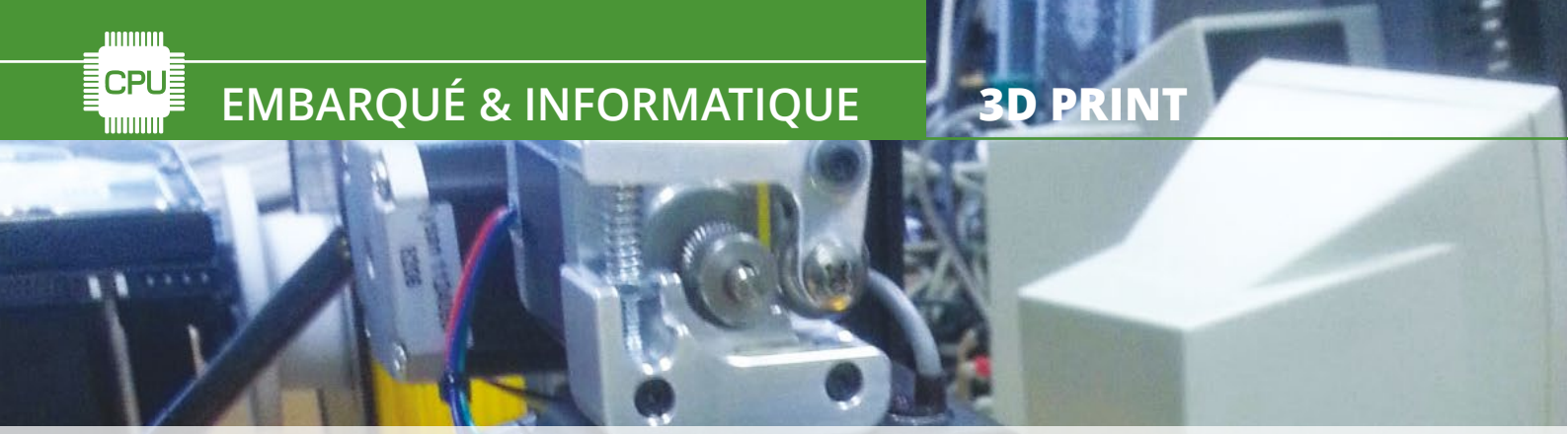
Et me voici arrivé à seulement 5 résultats, je n'ai plus qu'à installer et tester par moi-même. Merci **debtags** !

Bien sûr, cela fonctionnera avec bien d'autres types de recherches motivées par un besoin ou dans une optique de pure exploration. Avec quelques 32 facettes et plus de 600 tags en tout, vous avez de quoi largement satisfaire votre curiosité et faire connaissance avec tous les outils et applicatifs à votre disposition avec Raspbian.

## CONCLUSION

GNU/Linux est souvent perçu (et décrit) comme un système complexe, difficile à prendre en main, mais avec un peu de méthodologie et quelques explications basiques, il n'est finalement pas si difficile de partir à la découverte du système. Cette impression initiale est liée à la nature et l'histoire du système, son ouverture et sa richesse. Un GNU/Linux comme Raspbian, est un système UNIX ayant les mêmes qualités et la même réputation. Ceci me rappelle une citation dont l'origine s'est perdue avec le temps, mais est restée très populaire : « *Unix is user-friendly. It's just very selective about who its friends are* » (Unix est convivial. Il est simplement très sélectif concernant ses convives).

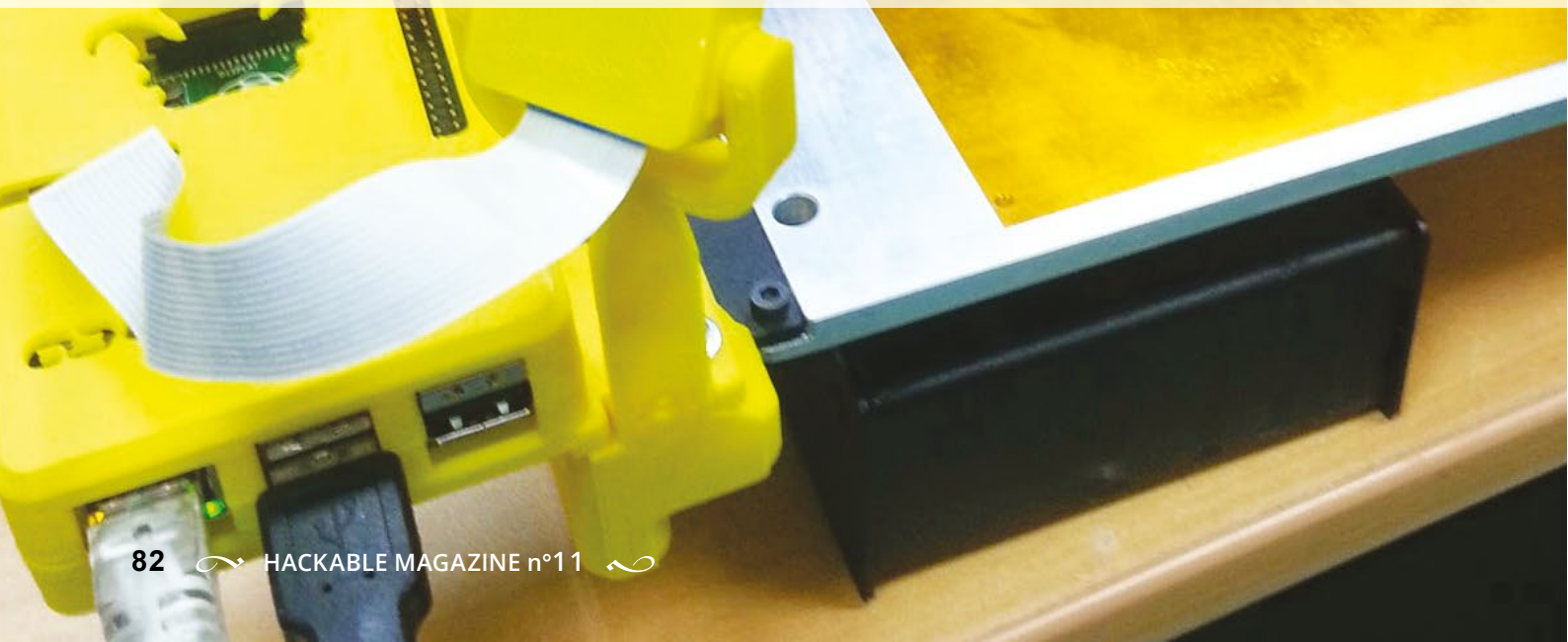
C'est simple, plus vous utiliserez le système, plus celui-ci s'ouvrira à vous et vous paraîtra convivial, ou en d'autres termes, plus vous deviendrez amis. **DB**



# OCTOPRINT : TRANSFORMEZ VOTRE RASPBERRY PI EN SERVEUR D'IMPRESSION 3D

*Yann Morère*

**Dans ces pages, nous allons convertir un Raspberry PI en serveur d'impression afin de permettre à un groupe de travail de partager, via le réseau, une imprimante 3D. Pour réaliser la supervision à distance d'une impression, on lui ajoutera un module caméra et on embarquera le tout sur l'imprimante pour réaliser de beaux « timeslapses ». Détaillons tout cela...**



# 1. INTRODUCTION

Je ne reviendrai pas sur la mise en œuvre de l'imprimante 3D utilisée dans ces pages, une PrintrBot Simple Metal, ni sur sa configuration. Initialement un ordinateur portable y était branché directement par souci de facilité/proximité de mise en œuvre, mais aussi pour le nomadisme des impressions. Cependant, dans le cadre d'une imprimante utilisée par un groupe de travail (laboratoire, fablab), il peut être alors intéressant de disposer d'un serveur d'impression directement accessible depuis le réseau local ou via Internet. Certains me rétorqueront que l'on peut très bien le faire à l'aide du PC portable, alors pourquoi utiliser un Raspberry Pi ? Tout d'abord, pour l'encombrement et la consommation électrique minimale qu'il requiert. Le Raspberry Pi est peu gourmand en ressources et pourra être installé directement sur le plateau de l'imprimante à l'aide de pièces imprimées. De plus, l'impression 3D ne demande pas beaucoup de puissance de calcul puisqu'il s'agit essentiellement de l'utilisation du port USB pour envoyer les commandes Gcode à l'imprimante.

On pourra ensuite ajouter à notre Raspberry Pi une webcam USB ou un module caméra spécifique afin de réaliser la supervision à distance de nos impressions. Là encore, on fixera la caméra à l'aide de pièces imprimées (figure 1).

Pour réaliser notre serveur d'impression, nous utiliserons le projet OctoPrint **[1]**, mais nous y reviendrons en détail dans la suite. Voyons maintenant le matériel nécessaire à la mise en œuvre de notre serveur.

# 2. MATÉRIELS ET SYSTÈME D'EXPLOITATION UTILISÉS

Nous aurons donc besoin d'un Raspberry Pi 512 revB ou d'un B+. L'achat d'un Pi 2 n'est pas vraiment nécessaire, car ces premières versions sont largement assez puissantes pour l'exécution d'OctoPrint. Cela vous permettra aussi d'économiser quelques euros. Par ailleurs, par souci de rapidité, nous installerons une version minimale de Raspbian, Minibian **[2]** qui n'embarque ni serveur graphique ni logiciels superflus. Cela permet de conserver toute la puissance de calcul pour notre application principale. Nous aurons aussi besoin d'une alimentation 2A, d'un câble réseau ou d'une clé WIFI pour l'accès réseau. En effet, mis à part le premier démarrage, l'installation et la configuration se feront exclusivement en mode console et à travers une connexion SSH.

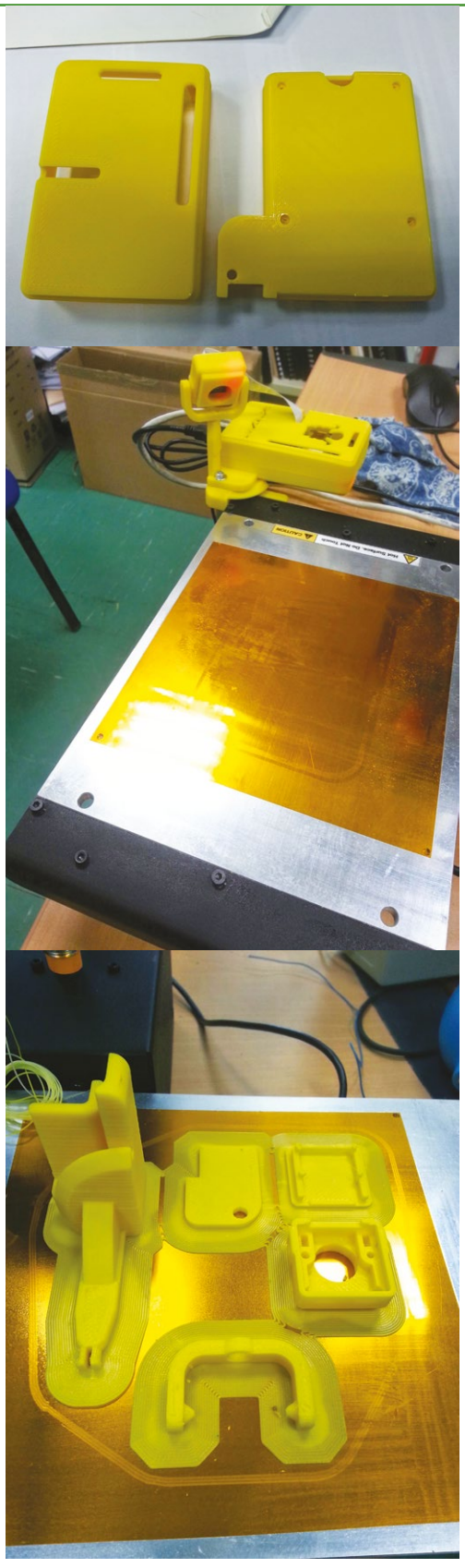


*Figure 1*



*Figure 2*

Nous aurons donc temporairement besoin d'un clavier USB et d'un écran : une TV en HDMI est suffisante, mais vous pouvez aussi brancher un écran VGA à l'aide d'un convertisseur HDMI-VGA.

Pour l'aspect supervision d'impression, j'ai utilisé le module camera spécifique au Raspberry Pi. Il est certes un peu cher, mais son intégration sur l'imprimante est très pratique par rapport à une webcam USB (figure 2).

En effet, le site web Thingiverse propose une série de pièces à imprimer pour l'intégration du Raspberry Pi et de son module caméra sur une imprimante Printrbot Simple Metal **[2]**. Ces pièces, après quelques modifications mineures (ajout du logo OctoPrint sur le boîtier, élargissement de la rainure pour la fixation au plateau de l'imprimante), ont été imprimées. Les différentes parties ont été assemblées et installées sur l'imprimante (figure 1).

**Le plateau se déplaçant lors de l'impression, il est impératif de regrouper les différents câbles (alimentation, USB et réseau) dans une gaine spiralée afin d'éviter tout arrachage ou pincement.**

Pour la partie logicielle, si vous désirez mettre en œuvre très rapidement OctoPrint, il vous est possible d'utiliser l'image OctoPi **[3]**, une image de carte SD basée sur Raspbian préconfigurée pour OctoPrint avec le support de la webcam/module caméra et SSL.

Cependant, si vous souhaitez utiliser les dernières versions de Raspbian, d'OctoPrint et comprendre comment fonctionne tout l'ensemble, l'installation manuelle est tout indiquée. C'est cette méthode que nous utiliserons dans ces pages.

Pour cela, nous choisissons d'utiliser Minibian **[4]** : une distribution minimale pour Raspberry Pi basée sur Raspbian. Son principal avantage est d'être très légère et complètement compatible avec Raspbian. Simplement elle n'embarque pas d'interface graphique ni d'outils superflus. Cette image est donc idéale pour les projets embarqués ou encore lorsque vous avez besoin de toutes les ressources du Pi pour des tâches spécifiques. Elle possède une faible empreinte mémoire et démarre très rapidement.

## 3. OCTOPRINT

Une fois le système d'exploitation installé et le réseau filaire configuré avec une adresse IP fixe, nous passons à l'installation d'OctoPrint.

### 3.1 Présentation

OctoPrint est un logiciel vous permettant de piloter votre imprimante 3D en lui envoyant des commandes Gcode. D'autres logiciels existent comme Repetier Host **[6]** ou encore Cura **[7]**. Mais OctoPrint est différent de ces autres solutions : il propose une interface web permettant de prendre le contrôle de l'imprimante à distance à travers le réseau local ou l'Internet tout en conservant une très bonne interactivité avec l'utilisateur. Pour cela, OctoPrint utilise massivement les frameworks et technologies actuelles comme AJAX et HTML5. Ce serveur a été développé pour être utilisable sur le Raspberry Pi. Ce dernier sera embarqué sur l'imprimante 3D et l'on pourra, par exemple, transformer son imprimante 3D en une imprimante WIFI par le simple ajout d'une clé WIFI et quelques configurations.

Les fonctionnalités d'OctoPrint sont les suivantes :

- Téléversement des fichiers Gcode sur le serveur pour l'impression. Il est aussi possible d'utiliser la carte SD de l'imprimante comme source de fichiers ;
- Sélection de fichiers pour l'impression. OctoPrint vous donnera aussi les statistiques d'utilisation de filament (longueur, volume), de temps d'impression ;
- Démarrage, arrêt et annulation d'une impression ;
- Lorsque l'on est connecté à l'imprimante, nous aurons accès à ses paramètres de fonctionnement (température extrudeur, température plateau), soit de manière textuelle ou graphique ;
- Accès au journal de communication entre le Raspberry Pi et l'imprimante pour une éventuelle vérification. Il est aussi possible d'envoyer directement des commandes gcode à l'imprimante.
- Prévisualisation graphique du fichier gcode : cela vous permet avant de lancer l'impression de vérifier les déplacements et les chemins d'impression par couche ;
- Supervision de l'imprimante par l'intermédiaire d'un streaming vidéo depuis une webcam ou le module caméra ;
- Création d'enregistrement Timelapse (à temps régulier ou par couche) permettant de réaliser une vidéo accélérée de votre impression. Ces vidéos sont stockées sur la carte SD du Raspberry Pi ;
- Contrôle d'accès à l'interface web par login et mot de passe. Les utilisateurs non autorisés ne pourront pas agir sur les fonctions de l'imprimante.
- Système de configuration via l'interface web, pour OctoPrint, l'imprimante et les comptes utilisateurs.

Il nous sera aussi possible d'ajouter des items dans les menus principaux d'OctoPrint pour l'arrêt du système ou encore l'activation du streaming video. Tout ceci sera détaillé dans la suite. Intéressons-nous maintenant à l'installation du serveur OctoPrint.

## 3.2 Installation

Pour cette partie, j'ai utilisé la page **[8]**, qui décrit précisément la marche à suivre. On commence par installer les paquets nécessaires à OctoPrint :

```
$ sudo apt-get install python-pip python-dev python-setuptools python-
virtualenv git libyaml-dev build-essential
```

Ensuite, on récupère les sources du projet à l'aide de git :

```
$ git clone https://github.com/foosel/OctoPrint.git
```

Puis on se déplace dans le répertoire des sources et l'on utilise les environnements virtuels python pour l'installation :

```
cd OctoPrint
virtualenv --system-site-packages venv
./venv/bin/python setup.py install
```

L'utilisation des environnements virtuels va nous permettre de réaliser des installations de Python isolées du système d'exploitation et séparées les unes des autres pour chaque projet. Cela permet de

pallier les problèmes de versions différentes de python pour des projets plus ou moins anciens, ou encore l'utilisation de bibliothèques incompatibles.

L'utilisation de l'option **--system-site-packages** impose que tout ce qui est installé sur le système d'exploitation soit disponible dans l'environnement virtuel **venv**.

La dernière ligne de commandes permet l'installation effective d'OctoPrint sur le système.

Cependant, afin de bénéficier des dernières nouveautés, nous allons utiliser la version de développement située dans la branche « devel » du projet. Pour l'instant, nous sommes sur la branche « master », la branche stable.

```
pi@printrbot:~/OctoPrint$ git status
# On branch master
nothing to commit (working directory clean)
```

On passe sur la branche « devel » à l'aide de la commande suivante :

```
pi@printrbot:~/OctoPrint$ git checkout devel
Switched to branch 'devel'
```

On recommence l'installation à l'aide des commandes :

```
$ virtualenv --system-site-packages venv
$ ./venv/bin/python setup.py install
```

On termine par la création du répertoire de configuration d'OctoPrint :

```
$ mkdir ~/.octoprint
```

Ce dernier recevra nos configurations spécifiques.

On remarquera que la version de développement ne prend en charge que les langues anglaise et allemande.

On peut maintenant tester le serveur OctoPrint à l'aide de la commande suivante :

```
pi@raspberrypi ~ $ ~/OctoPrint/venv/bin/octoprint
 * Running on http://0.0.0.0:5000/
```

Sur un Raspberry Pi de première génération, le premier démarrage d'OctoPrint peut prendre jusqu'à une minute avant d'être opérationnel (de nombreux fichiers sont copiés dans le répertoire **~/.octoprint**). Ensuite pour tester, on ouvre un navigateur et on saisi l'adresse IP du Raspberry Pi suivie de **:5000** pour indiquer le port d'écoute :

```
http://192.168.1.4:5000
```

Lors de ce premier lancement, l'application va remplir le répertoire `~/.octoprint` avec les fichiers de configuration par défaut. Nous les modifierons par la suite.

Lors de ce premier lancement, l'application vous demandera de configurer le contrôle d'accès à la gestion de l'imprimante. Pour cela, vous devrez ajouter des identifiants : utilisateur et mot de passe. Ce dernier sera l'administrateur de l'application. Vous pourrez par la suite ajouter d'autres utilisateurs depuis l'interface.

La version 1.3 d'OctoPrint est bien plus réactive que les précédentes. Si vous utilisez encore des versions antérieures, je vous conseille fortement de réaliser les mises à jour.

Pour une utilisation régulière, on préférera lancer automatiquement OctoPrint au démarrage du système. Pour cela, on copie les fichiers suivants :

```
$ sudo cp ~/OctoPrint/scripts/octoprint.init /etc/init.d/octoprint
$ sudo chmod +x /etc/init.d/octoprint
$ sudo cp ~/OctoPrint/scripts/octoprint.default /etc/default/octoprint
```

On peut ensuite éditer le fichier **/etc/default/octoprint** afin de vérifier que le chemin vers l'exécutable OctoPrint est correctement renseigné :

```
DAEMON=/home/pi/OctoPrint/venv/bin/octoprint
```

De la même manière, j'ai configuré les variables suivantes afin que le serveur utilise nos versions des fichiers de configuration :

```
# base directory to use
BASEDIR=/home/pi/.octoprint
# configuration file to use
CONFIGFILE=/home/pi/.octoprint/config.yaml
```

Finalement, nous ajoutons les scripts au démarrage du système à l'aide de la commande suivante :

```
$ sudo update-rc.d octoprint defaults
```

Ensuite, on peut gérer le démarrage et l'arrêt manuel du serveur à l'aide d'une commande du type :

```
$ sudo service octoprint {start|stop|restart}
```

Le service étant maintenant présent, nous allons gérer l'arrêt et le redémarrage du système depuis l'interface d'OctoPrint . Pour cela, nous aurons besoin d'écrire un fichier pour **sudo**, mais aussi de modifier le fichier de configuration **~/.octoprint/config.yaml**.

On commence par créer le fichier **/etc/sudoers.d/octoprint-shutdown** et on le renseigne avec la ligne suivante :

```
pi ALL=NOPASSWD: /sbin/shutdown
```

Cela nous permet de lancer la commande d'arrêt/redémarrage sans demande de mot de passe pour l'utilisateur **pi**.

Ensuite, on ajoute ce qui suit dans le fichier **~/.octoprint/config.yaml** :

```
system:
  actions:
  - name: Shutdown
    command: sudo shutdown -h now
    action: shutdown
    confirm: You are about to shutdown the system.
  - name: Reboot
    command: sudo shutdown -r now
    action: reboot
    confirm: You are about to reboot the system
```

Ceci va ajouter dans le menu système de l'interface web deux nouvelles entrées permettant d'arrêter et redémarrer notre serveur (figure 3).

Pour la prise en compte de ces nouveaux paramètres, il est nécessaire de redémarrer OctoPrint :

```
pi@raspberrypi:~/.octoprint$ sudo service octoprint restart
```

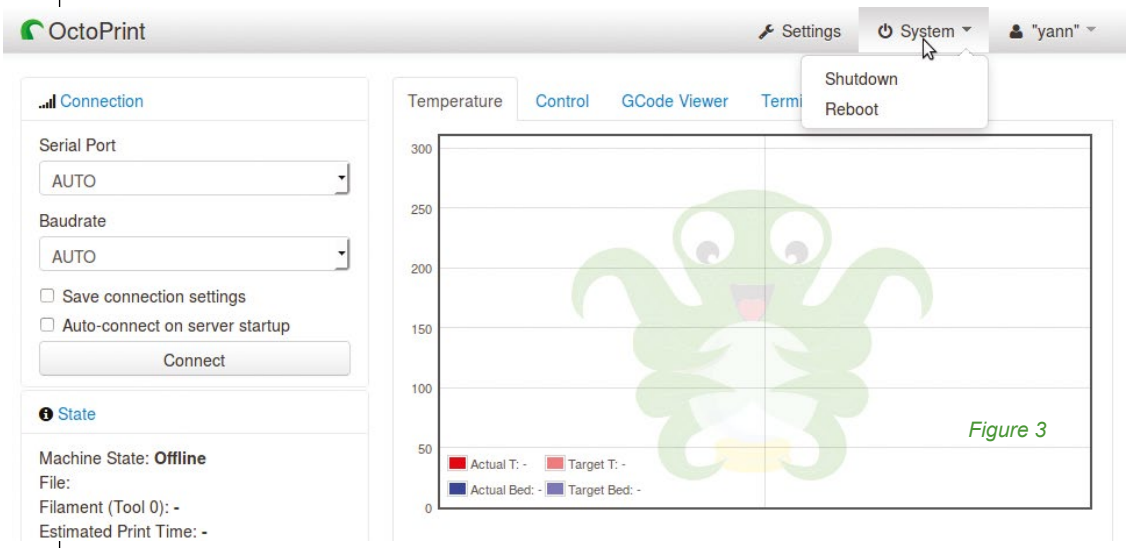



*Figure 3*

Dans le cas où vous auriez besoin d'utiliser OctoPrint via le port 80, soit à cause de restrictions au niveau de votre réseau, ou simplement par désir d'avoir des URL plus jolies (!!!). Il vous faudra utiliser HAProxy comme un « reverse proxy » au lieu de configurer OctoPrint pour fonctionner sur le port 80. Ceci procure plusieurs avantages :

- OctoPrint n'aura pas besoin de fonctionner avec les privilèges « root », ce qui serait imposé s'il fonctionnait directement sur le port 80 ;
- mjpg-streamer sera lui aussi accessible sur le port 80.

Pour plus d'informations à ce sujet, je vous renvoie à la lecture de la page **[8]**.

## 3.3 Installation du streaming vidéo

Afin de pouvoir réaliser la supervision de nos impressions à l'aide du module caméra/webcam, nous allons installer MJPG-Streamer **[9]**. Ce dernier récupère les images JPEG de la webcam et les envoie sous la forme d'un flux M-JPEG via le protocole HTTP. Il est alors possible de récupérer ce flux vidéo et de l'afficher dans un navigateur ou encore via VLC.

On commence par installer les outils nécessaires à la compilation :

```
$ cd ~
$ sudo apt-get install subversion libjpeg8-dev imagemagick
libav-tools cmake libraspberrypi-dev
```

**On notera que la bibliothèque `libraspberrypi-dev` est nécessaire, sinon la compilation échouera, car le compilateur ne trouvera pas le fichier `bcm-host.h`.**

Ensuite, on récupère les sources du projet, on le compile et on l'installe :

```
$ git clone https://github.com/jacksonliam/mjpg-streamer.git
$ cd mjpg-streamer/mjpg-streamer-experimental
$ make
$ sudo make install
```

On peut alors tester le bon fonctionnement de **mjpeg-streamer** avec le module camera à l'aide de la commande suivante :

```
$ ./mjpg_streamer -i "./input_raspicam.so -fps 5" -o "./output_http.so"
MJPG Streamer Version: svn rev: Répertoire non versionné
 i: fps.............: 5
 i: resolution........: 640 x 480
 i: camera parameters..............:

Sharpness 0, Contrast 0, Brightness 50
Saturation 0, ISO 400, Video Stabilisation No, Exposure compensation 0
Exposure Mode 'auto', AWB Mode 'auto', Image Effect 'none'
Metering Mode 'average', Colour Effect Enabled No with U = 128, V = 128
Rotation 0, hflip No, vflip No
 o: www-folder-path...: disabled
 o: HTTP TCP port.....: 8080
 o: username:password.: disabled
 o: commands..........: enabled
 i: Starting Camera
Encoder Buffer Size 81920
```

Dans le cas d'une webcam, la ligne de commandes pour le test est la suivante :

```
./mjpg_streamer -i "./input_uvc.so -y" -o "./output_http.so"
```

Ensuite, il suffit d'ouvrir un navigateur sur une autre machine du même réseau et de saisir l'adresse de votre Pi avec les options suivantes :

```
http://192.168.1.4:8080/?action=stream
```

Vous devriez voir apparaître l'image de votre module camera/webcam (figure 4).

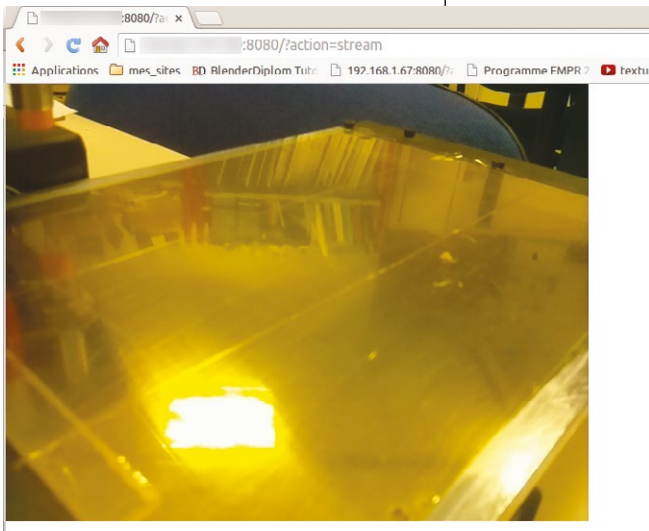



*Figure 4*

Votre supervision est opérationnelle, il faut maintenant intégrer son démarrage et son arrêt dans l'interface d'OctoPrint. Pour cela, nous aurons besoin d'écrire un petit script bash, mais aussi de modifier le fichier de configuration **~/.octoprint/config.yaml**.

On commence par mettre à jour la liste des bibliothèques dynamiques par **ldconfig** :

```
$ sudo /sbin/ldconfig -v
```

Cette opération n'est pas forcément nécessaire, mais cela évitera d'éventuelles déconvenues.

Ensuite on crée le script **/usr/local/bin/raspi_streamer** (mjpg_streamer est installé dans **/usr/local/bin** ) :

```
#!/bin/bash
# Start / stop streamer

Daemon=mjpg_streamer
DaemonBase=/usr/local
DaemonArgs="-i \"input_raspicam.so -fps 5\" -o \"output_http.so\""

case "$1" in
   start)
         eval LD_LIBRARY_PATH=${DaemonBase}/lib ${DaemonBase}/bin/${Daemon}
${DaemonArgs} >/dev/null 2>&1 &
         echo "$0: started"
     ;;
   stop)
         pkill -x ${Daemon}
         echo "$0: stopped"
     ;;
   *)
         echo "Usage: $0 {start|stop}" >&2
     ;;
esac
```

Dans le cas d'une webcam, vous remplacerez la ligne :

```
DaemonArgs="-i \"input_raspicam.so -fps 5\" -o \"output_http.so\""
```

par

```
DaemonArgs="-i \"input_uvc.so -y\" -o \"output_http.so\""
```

On ajoute ce qui suit dans le fichier **~/.octoprint/config.yaml** :

```
webcam:
  ffmpeg: /usr/bin/avconv
  snapshot: http://127.0.0.1:8080/?action=snapshot
  stream: http://192.168.1.4:8080/?action=stream
system:
  actions:
  - action: streamon
    command: /usr/local/bin/raspi_streamer start
    confirm: false
    name: Start video stream
  - action: streamoff
    command: /usr/local/bin/raspi_streamer stop
    confirm: false
    name: Stop video stream
  - name: Shutdown
    command: sudo shutdown -h now
```

```
    action: shutdown
    confirm: You are about to shutdown the system.
  - name: Reboot
    command: sudo shutdown -r now
    action: reboot
    confirm: You are about to reboot the system
```

Ceci permet d'ajouter deux nouveaux items dans le menu système, permettant la mise en route et l'arrêt du streaming vidéo (figure 5).

On n'oublie pas de rendre exécutable le nouveau script puis on redémarre OctoPrint pour la prise en compte de la nouvelle configuration :

```
pi@raspberrypi:~/.octoprint$ sudo chmod 755 /usr/local/bin/raspi_streamer
pi@raspberrypi:~/.octoprint$ sudo service octoprint restart
[ ok ] Restarting Octoprint Daemon: Octoprint.
```

Dans un premier temps, mes tests ont été réalisés à l'aide d'un Raspberry Pi 512 revB, puis j'ai utilisé un B+. Après ce changement, le streaming vidéo ne fonctionnait plus. Ainsi, lorsque j'essayais de réaliser une capture, j'obtenais le message suivant :

```
# raspistill -o /tmp/gate_now.jpg
mmal: mmal_vc_component_enable: failed to enable component: ENOSPC
mmal: camera component couldn't be enabled
mmal: main: Failed to create camera component
mmal: Failed to run camera app. Please check for firmware updates
```

Dans un premier temps, j'ai vérifié que la variable **gpu_mem** était bien renseignée dans le fichier **/boot/config.txt** :

```
gpu_mem=128
```

Mais cela n'a pas résolu le problème. Après quelques recherches, la page **[10]** donne une solution : cette erreur est due à un conflit avec le module 1-Wire (W1), dans le cas où les modules ne sont pas chargés dans le bon ordre au démarrage. Le problème peut être résolu en basculant le connecteur utilisé pour le bus 1-Wire sur le connecteur 18. Pour cela, on modifie le fichier **/boot/cmdline.txt** en ajoutant l'option suivante :

```
bcm2708.w1_gpio_pin=18
```

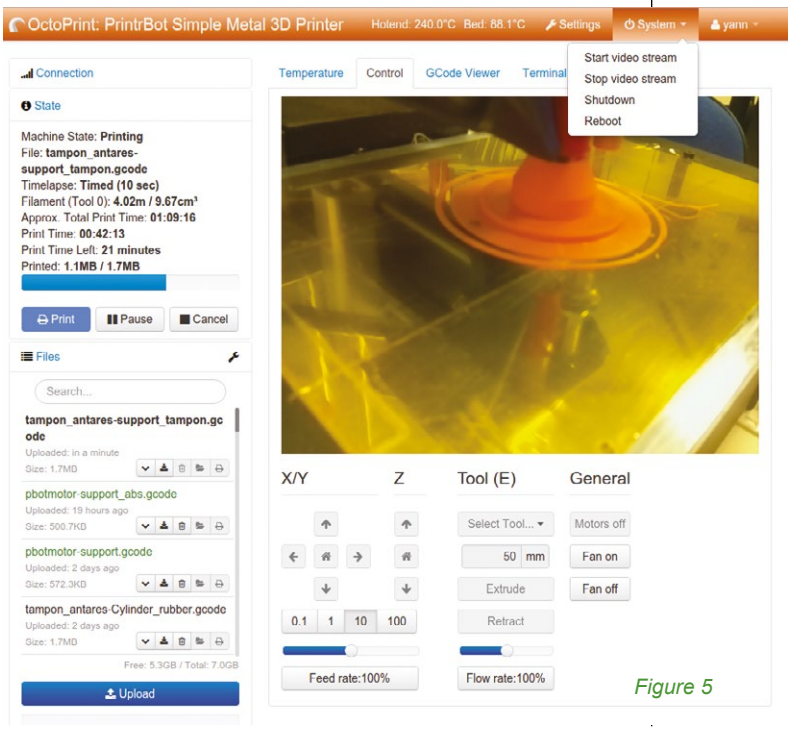


Figure 5

et l'on obtient :

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline
rootwait bcm2708.w1_gpio_pin=18
```

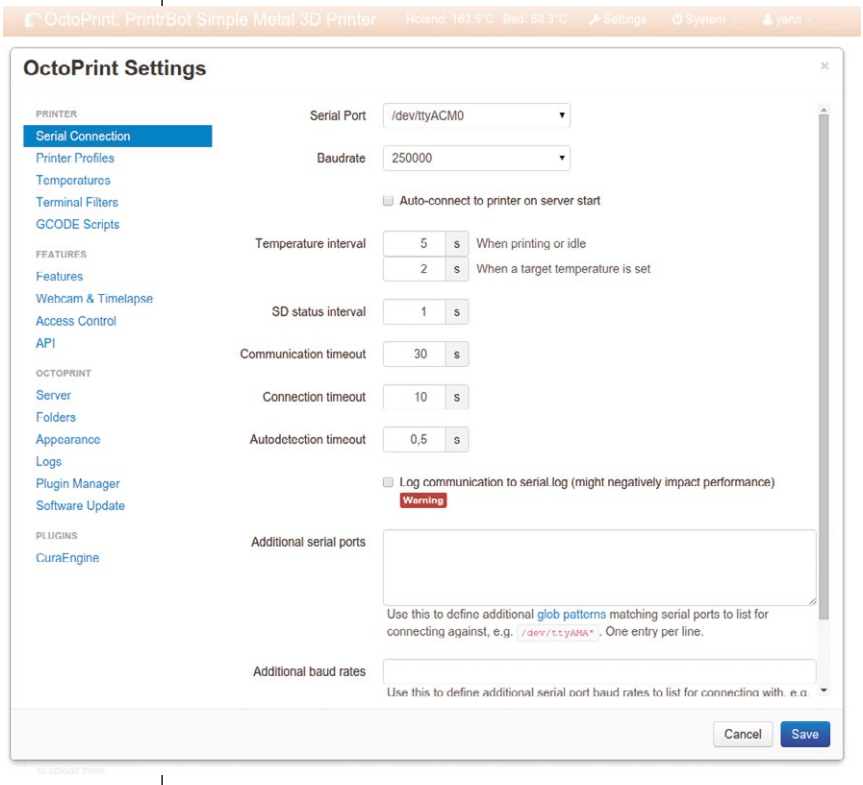



*Figure 6*

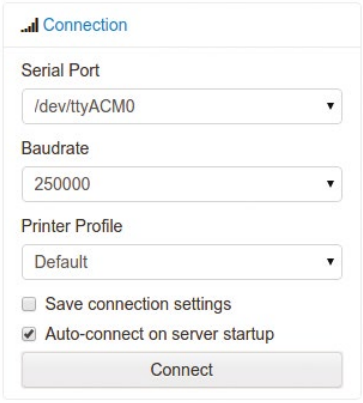



*Figure 7*

Après un redémarrage, la caméra fonctionne à nouveau. On peut tester la capture vidéo à l'aide de la commande suivante :

```
$ raspivid -o test.mp4
```

Voilà tout est maintenant fonctionnel, on peut passer à la configuration du serveur OctoPrint pour la prise en charge de notre imprimante.

## 3.4 Configuration

Dorénavant, toutes les configurations se feront à travers l'interface web d'OctoPrint. L'accès aux paramètres se fait grâce à l'onglet ***Settings*** qui ouvre une nouvelle fenêtre dans l'interface (figure 6). Je ne détaillerai pas tous les onglets, mais seulement les plus importants. Pour plus de détails, je vous renvoie vers la documentation d'OctoPrint **[11]**.

Nous allons commencer par la configuration la plus importante, la liaison série avec l'imprimante via le port USB. Bien qu'elle soit branchée en USB, notre imprimante est reconnue en tant `/dev/ttyACM0` et non `/dev/ttyUSB0`. Si vous vous demandez pourquoi, allez consulter l'article **[12]** pour plus d'explications :D. On peut alors fixer la vitesse de transfert au maximum : 250000 bauds. Ceci permet de piloter l'imprimante de manière plus réactive. Les versions antérieures ne permettaient pas d'utiliser cette vitesse, car elles n'intégraient pas `pyserial` (qui supporte cette vitesse). On était alors obligé d'utiliser une vitesse de 115200 bauds.

Le reste des paramètres est laissé par défaut. On sauvegarde et on teste la connexion à l'imprimante en cliquant sur le bouton ***Connect*** dans le menu ***Connection*** en haut à gauche de l'interface (figure 7).

Si tout se passe bien, le bouton ***Disconnect*** apparaît et la section ***State*** de l'interface est mise à jour indiquant que l'imprimante est opérationnelle. Le cas échéant, il faudra activer les « logs » de communication afin de trouver d'où vient le problème.

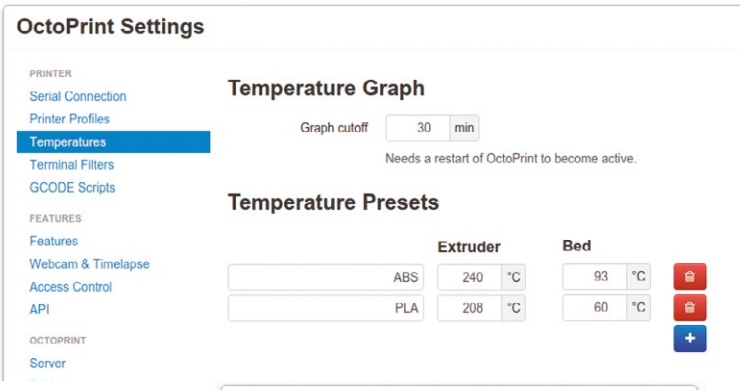
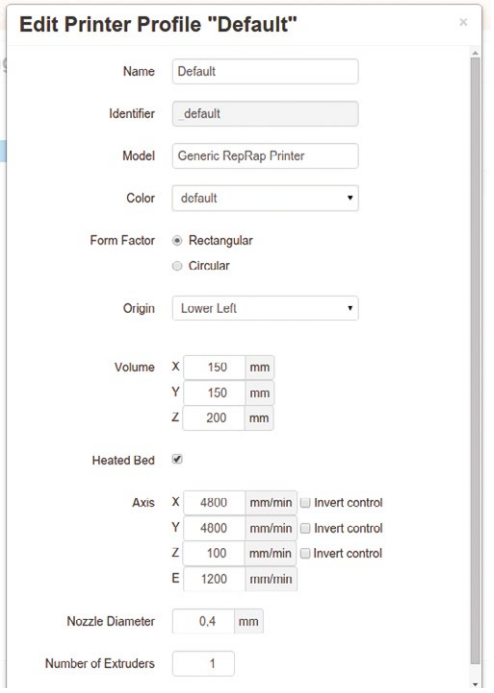



Figure 8

Nous allons maintenant définir les paramètres de notre imprimante dans les onglets ***Printer Profiles*** et ***Temperatures***. Le premier permet de définir les caractéristiques physiques de notre imprimante (dimensions du plateau, nombre et taille des extrudeurs, etc.). Le second va nous permettre de définir différentes températures (plateau et extrudeur) pour différents types de matériaux. Tout ceci est résumé en figure 8.




Je passe directement à l'onglet ***Features*** qui permet d'activer/désactiver les fonctionnalités relatives aux graphiques de température, à la visualisation du Gcode, au support de la carte SD et à d'autres options de la communication série. L'onglet ***Webcam & Timelapse*** vous permet de configurer l'accès au streaming vidéo du module caméra. Mais nous l'avons déjà renseigné dans le fichier `config.yaml`. Il est possible de tester l'accès au streaming vidéo, à la capture d'écran et vérifier la présence du programme `avconv` via les boutons idoines (figure 9).

L'onglet ***Access control*** permet de gérer les utilisateurs authentifiés de l'application. Il est possible de désactiver temporairement des comptes et aussi de rendre certains utilisateurs administrateurs du serveur d'impression.

L'onglet ***Appearance*** vous permet de modifier le titre et les couleurs de l'interface graphique. À l'avenir, il sera aussi possible de télécharger une archive pour la localisation de l'interface. Pour l'instant, seules les langues anglaise et allemande sont supportées.

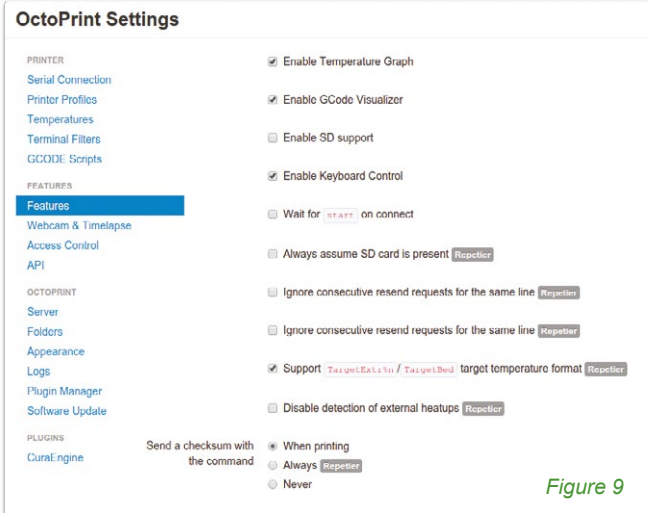
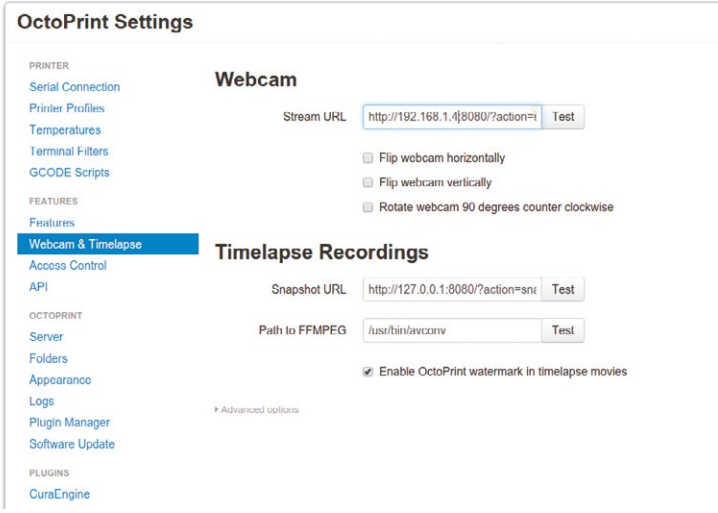



Figure 9



**L'installation du trancheur Cura sur le système d'exploitation vous permet ensuite d'imprimer directement à partir de fichiers STL. Après configuration du trancheur Cura, OctoPrint sera capable de générer directement le gcode à envoyer à l'imprimante .**

L'onglet ***Logs*** vous permet de télécharger et ensuite consulter le journal de l'application. Si vous avez activé la journalisation de la communication, le fichier **`serial.log`** sera aussi disponible. Un fichier journal sera également disponible si vous avez installé le trancheur Cura **[13]**.

OctoPrint possède aussi un gestionnaire de plugins accessible depuis l'onglet ***Plugins***. D'une manière très simple, il vous sera possible d'ajouter de nouvelles fonctionnalités à votre application.

Le dernier onglet ***Software Update*** vous permettra de mettre à jour très facilement OctoPrint.

Nous en avons terminé pour la partie configuration. Voyons maintenant comment installer CuraEngine afin de pouvoir trancher directement les fichiers STL à partir d'OctoPrint.

## 4. AJOUT DU TRANCHEUR CURAENGINE

Afin de pouvoir directement utiliser les fichiers issus de la modélisation 3D au format STL, nous allons installer CuraEngine **[14]**. OctoPrint possède un plugin pour l'intégration de ce dernier. Celui-ci permettra de proposer le tranchage vers le format gcode dès qu'un fichier STL sera téléversé sur le serveur.

Pour cela, il faut commencer par installer CuraEngine sur le système d'exploitation. L'installation d'un compilateur compatible C++11 (norme du langage C++ approuvée unanimement le 12 août 2011) est nécessaire :

```
$ sudo apt-get -y install gcc-4.7 g++-4.7
```

On poursuit en téléchargeant les sources du projet CuraEngine :

```
$ git clone -b legacy https://github.com/Ultimaker/
CuraEngine.git
```

Un patch est requis pour le bon fonctionnement du trancheur. Après l'avoir récupéré, on l'applique sur l'arborescence initiale des sources :

```
cd CuraEngine
wget http://bit.ly/curaengine_makefile_patch -O CuraEngine.
patch
patch < CuraEngine.patch
```

Ensuite on peut lancer la compilation :

```
make CXX=g++-4.7
```

Finalement, l'exécutable de CuraEngine est créé dans le répertoire **`build`** : **`~/CuraEngine/build/CuraEngine`**.

Il faut maintenant renseigner OctoPrint de la présence de ce nouveau programme. Pour cela, on se rend dans les ***Settings***→Onglet ***CuraEngine*** et on renseigne le chemin complet vers l'exécutable comme décrit en figure 10. On peut tester la validité de l'accès à l'exécutable par l'intermédiaire du bouton ***Test***.

Ensuite, pour pouvoir fonctionner, CuraEngine requiert un profil d'impression (spécificités de l'imprimante + configuration de l'impression : épaisseur de couche, vitesse, etc.) pour réaliser le tranchage. Par défaut, aucun profil n'est présent et il va falloir en importer un. Pour la Printrbot Simple Metal utilisée dans cet article, un profil de base peut être téléchargé à l'adresse **[15]**. L'importation est réalisée à l'aide du bouton ***Import Profile*** présent sur le même onglet (figure 10).

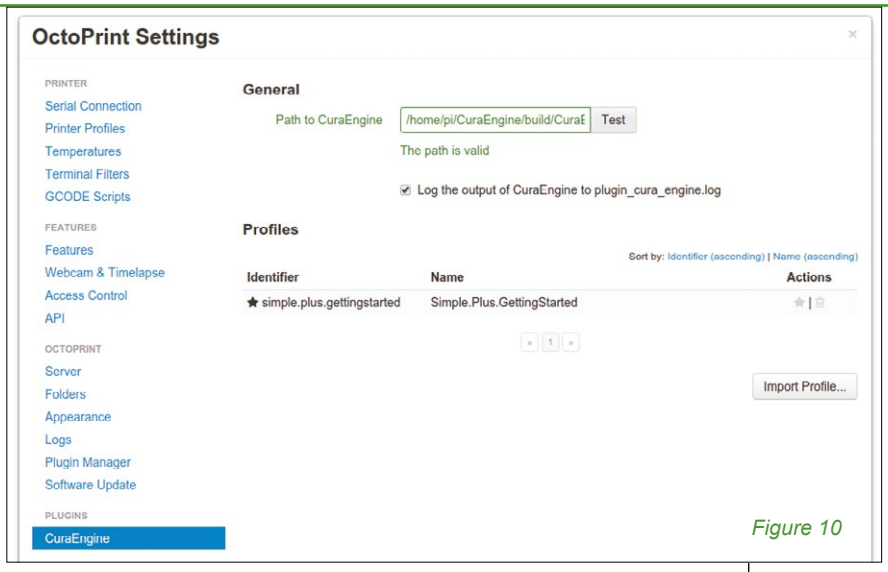



*Figure 10*

C'est terminé, nous verrons dans la section suivante comment utiliser le plugin pour réaliser le tranchage automatique d'un fichier STL téléversé sur le serveur OctoPrint.

## 5. MISE EN ŒUVRE POUR L'IMPRESSION

Dans un premier temps, nous allons décrire les 5 onglets présents dans l'interface. Le premier, nommé ***Temperature***, permet de contrôler manuellement les températures de l'extrudeur et du plateau. Il contient aussi le graphique de l'évolution de la température au cours du temps (figure 11).

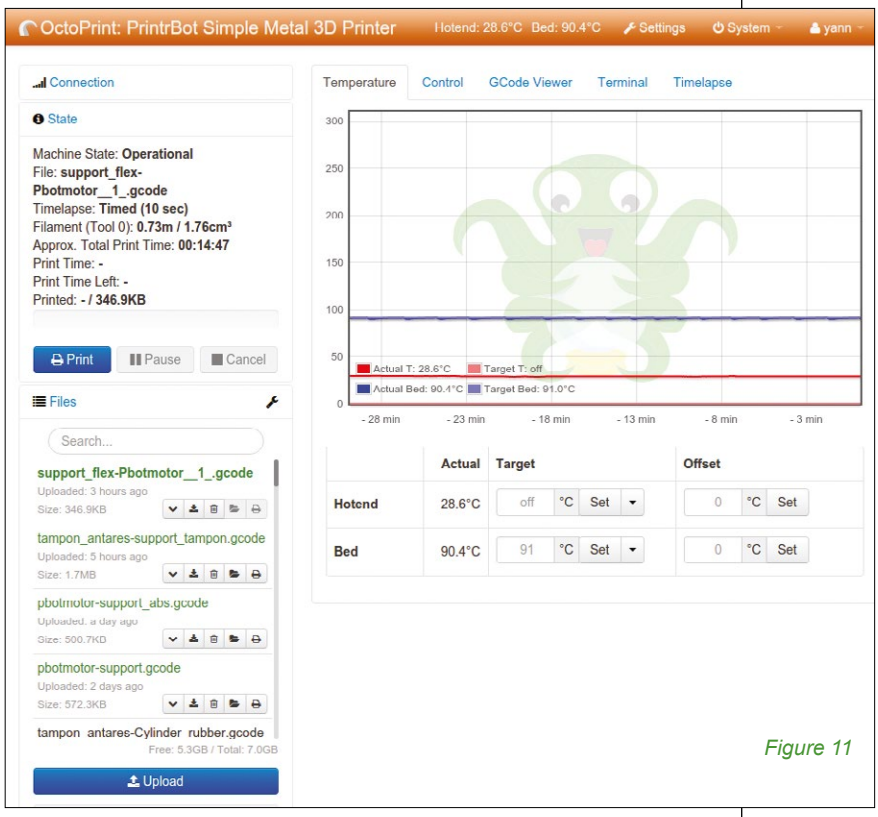



*Figure 11*

Le second nommé ***Control*** permet de piloter manuellement les organes de l'imprimante 3D : déplacement en X, Y et Z, activation de l'extrusion, marche/arrêt du ventilateur. S'il a été configuré et activé, le streaming vidéo est aussi affiché dans cet onglet (figure 12, page suivante).

Le troisième onglet ***Gcode viewer*** permet de visualiser les couches et les déplacements de l'extrudeur du fichier chargé pour la future impression. Cela permet éventuellement de détecter des erreurs. Il est possible de zoomer sur la zone graphique à l'aide de la molette de la souris. La bas de l'onglet renseigne les caractéristiques de la pièce à imprimer : taille et nombre de couches (figure 13, page suivante).

L'onglet ***Terminal*** vous permet de visualiser les commandes Gcode envoyées à l'imprimante. Il est possible d'envoyer manuellement ses

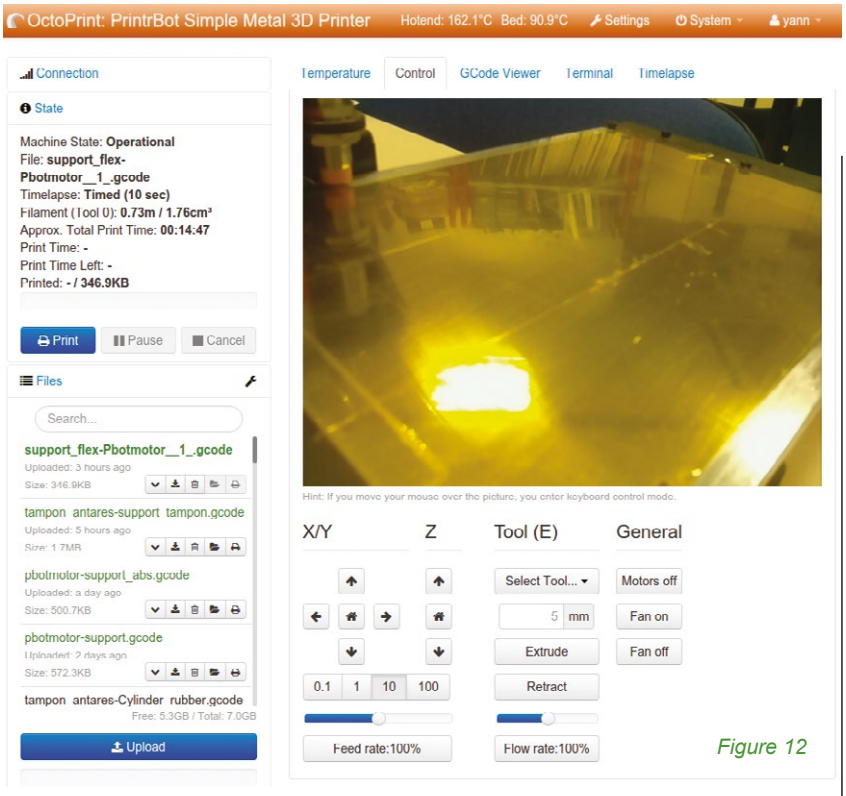



*Figure 12*

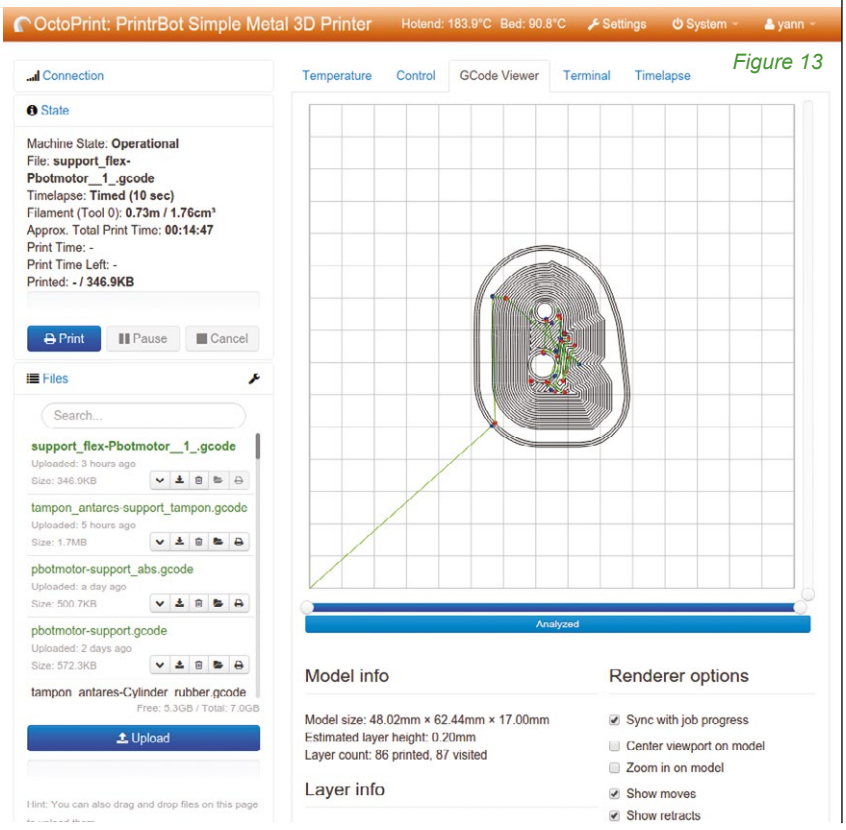



*Figure 13*

propres commandes à l'aide de la zone de texte prévue à cet effet (figure 14, ci-contre).

Le dernier onglet ***Timelapse*** vous permet de configurer l'effet d'ultra accéléré réalisé image par image sur la durée de votre impression. Vous obtiendrez alors une vidéo que vous pourrez télécharger à partir de ce même onglet. Il est possible de prendre une image par couches ou bien à intervalle de temps régulier (figure 15, ci-contre).

Voyons maintenant comment imprimer une pièce à l'aide de notre tout nouveau serveur d'impression 3D.

## 5.1 Cas d'un fichier Gcode

Je considère pour cette section que vous avez déjà modélisé et généré le fichier Gcode de votre pièce à imprimer à l'aide des outils libres Blender et Slic3r par exemple.

Afin de ne pas perdre de temps, on peut mettre en chauffe manuellement le plateau et l'extrudeur à la température associée avec le matériau. La montée en température de la buse est assez rapide, mais celle du plateau est bien plus longue, surtout dans le cas de l'ABS. Depuis l'onglet ***Control***, on ramène l'extrudeur à sa position initiale X, Y et Z à l'aide des boutons ***Home***.

Dans le cas de changement de matériau ou de couleur, il est préférable d'extruder quelques centimètres de filament après avoir déplacé la buse vers le haut pour permettre au plastique de s'échapper sans contraintes.

On commence par téléverser le fichier sur le serveur à l'aide du bouton ***Upload***. Une fois le fichier affiché dans la liste, on doit le charger dans le ***Gcode Viewer*** afin de pouvoir lancer l'impression. Cette action est réalisée en cliquant sur le petit bouton en forme de répertoire (figure 16). Cela vous permet aussi de réaliser une dernière vérification (couche par couche) et éventuellement éviter de gaspiller de la matière si le tranchage n'est pas conforme à celui attendu.




*Figure 16*

Tout est maintenant prêt, on peut lancer l'impression à l'aide du bouton ***Print***. L'imprimante fait ses tests, attend que les températures de travail soient atteintes puis démarre l'impression. Vous pouvez alors suivre le déroulement grâce au « streaming vidéo » présent dans l'onglet ***Control***. À tout moment, vous pouvez mettre l'impression en pause ou encore l'annuler grâce aux boutons prévus à cet effet (figure 6).

## 5.2 Cas d'un fichier STL

Comme dans le cas précédent, on téléverse le fichier STL sur le serveur OctoPrint à l'aide du bouton ***Upload***. Une fois listé, on remarque que l'on ne peut pas le charger dans le ***Gcode Viewer***, l'icône n'est pas présente. Mais une nouvelle icône est apparue : « Slice » (figure 17, page suivante). On va pouvoir réaliser le

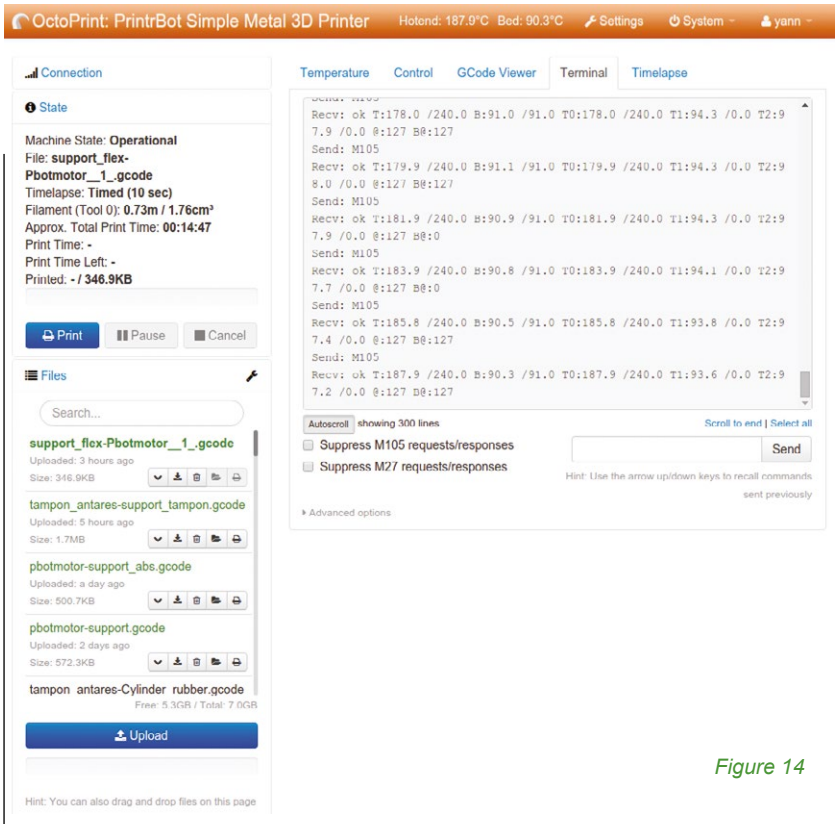



*Figure 14*

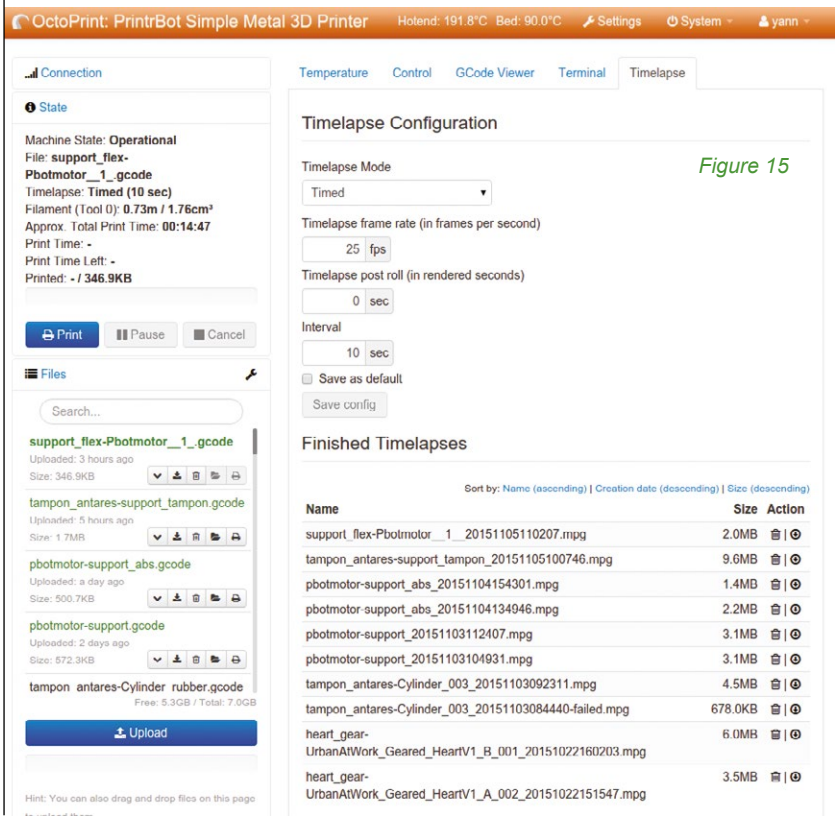



*Figure 15*

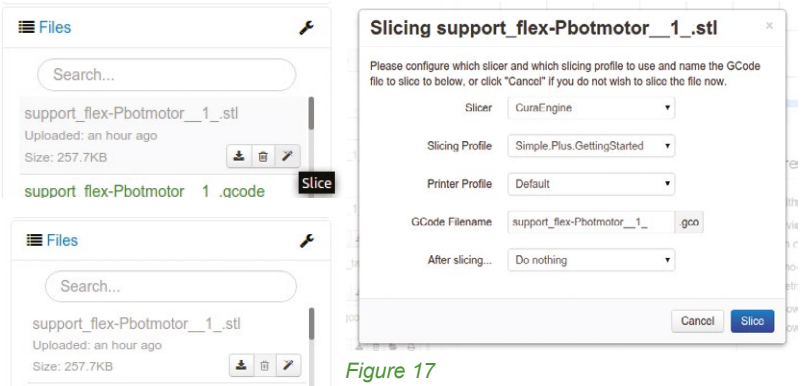



*Figure 17*

tranchage en cliquant sur cette dernière. Cette action ouvre la fenêtre de configuration de CuraEngine (figure 17). Nous conservons les paramètres par défaut et lançons le tranchage à l'aide du bouton ***Slice***.

La progression de l'action est affichée dans l'interface, puis, une fois terminée, le fichier Gcode généré apparaît (extension .gco). Il ne reste plus qu'à le charger dans le ***Gcode Viewer*** et lancer l'impression (figure 18).

## CONCLUSION

Nous avons terminé ce petit tour d'horizon du serveur d'impression 3D OctoPrint. Après quelques 8 mois d'utilisation, j'ai pu constater que le projet a beaucoup évolué avec l'ajout de fonctionnalités intéressantes et la disparition de nombreux bugs. La stabilité de l'ensemble est exemplaire même en usage intensif (plus de 8h par jour en continu). La précédente génération de Raspberry Pi est largement assez puissante pour ce type d'utilisation et sa petite taille permet de l'embarquer directement sur l'imprimante. Voilà, j'espère vous avoir donné envie d'essayer cette application qui permet de s'affranchir d'un PC plus ou moins dédié à l'imprimante. **YM**

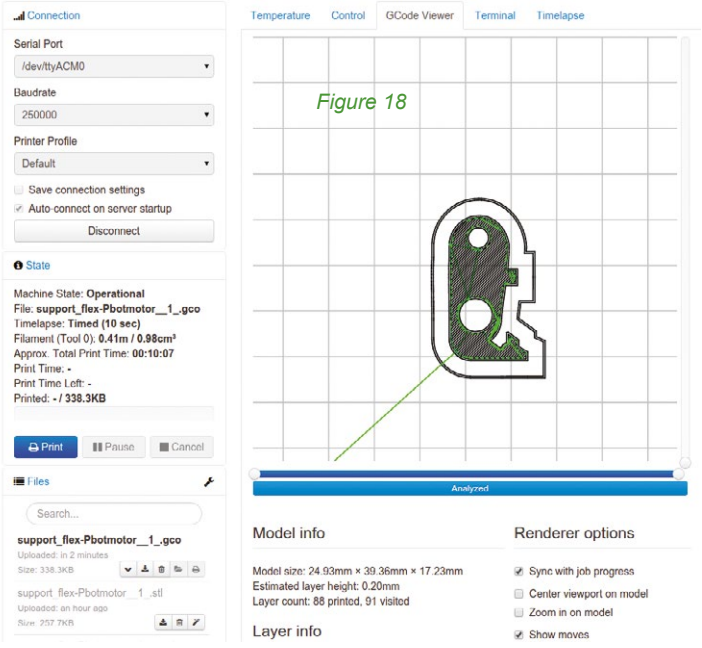



*Figure 18*

## RÉFÉRENCES

**[1] http://octoprint.org/**

**[2] http://www.thingiverse.com/thing:673133**

**[3] https://github.com/guysoft/OctoPi**

**[4] https://minibianpi.wordpress.com**

**[5] http://www.repetier.com/download-software/**

**[6] https://ultimaker.com/en/support/software/cura-1506**

**[7] https://github.com/foosel/OctoPrint/wiki/Setup-on-a-Raspberry-Pi-running-Raspbian**

**[8] https://github.com/jacksonliam/mjpg-streamer**

**[9] http://raspberrypi.stackexchange.com/questions/13764/what-causes-enospc-error-when-using-the-raspberry-pi-camera-module**

**[10] http://docs.octoprint.org/en/master/**

**[11] https://www.rfc1149.net/blog/2013/03/05/what-is-the-difference-between-devttyusbx-and-devttyacmx/**

**[12] https://github.com/foosel/OctoPrint/wiki/Plugin:-Cura**

**[13] https://github.com/Ultimaker/CuraEngine**

**[14] http://help.printrbot.com/Guide/2.+Getting+Started+with+Cura+on+Your+Printrbot+Simple/164**

# ACTUELLEMENT DISPONIBLE
## OPEN SILICIUM N°17 !




# LA CHASSE AUX BUGS NOYAU
## ...SUR RASPBERRY PI VIENT D'OUVRIR !

# NE LE MANQUEZ PAS
## CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
## www.ed-diamond.com

# Maker Faire® Paris

Présenté par Leroy Merlin

## 30 avril & 1er mai 2016

Paris expo porte de Versailles - Foire de Paris

# APPEL AUX MAKERS !

**Nous recherchons**

**Bricoleurs** ★ Ingénieurs

Amateurs ⚓ **Hackers** ⚓ Geeks

**Chercheurs** ★ Inventeurs

Entrepreneurs ◊ **Rêveurs**

Inscription sur

**makerfaireparis.com**