N°18 | MAI - JUIN 2017

HACKABLE MAGAZINE

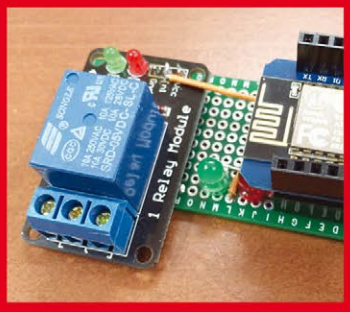
DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

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

ARDUINO / BOOT

Démarrez n'importe quel PC à distance avec une carte Arduino Ethernet ou un module wifi ESP8266

p. 14



PI / AUDIO

Configurez et utilisez un haut-parleur Bluetooth pour smartphone avec votre Raspberry Pi

p. 74

RÉTRO / FREEDOS

Oubliez les émulations poussives et ressuscitez un vieux PC grâce à FreeDOS et ses applications

p. 82

RASPBIAN / PAQUETS

Apprenez à gérer les paquets de différentes versions de Raspbian sur une seule Raspberry Pi

p. 94

Domotique / VoIP / Hack

Créez votre interphone connecté !

p. 42

- Connectez votre interphone/portier au net
- Renvoyez les appels sur votre smartphone
- Ouvrez à distance à vos visiteurs

ARDUINO / VGA

Affichez des images sur un écran VGA depuis votre Arduino UNO avec seulement 4 résistances !

p. 30

MATÉRIEL / DESSOUDER

Découvrez les différentes techniques, accessoires et équipements pour dessouder les composants

p. 04

L 19338 - 18 - F: 7,90 € - RD

ÉDITO

Cloud, cloud, cloud...

Je ne sais pas ce que vous en pensez, mais je trouve que ça devient pénible.

Qu'il existe une solution pour mettre ses informations personnelles « dans le nuage », autrement dit, sur un serveur web qui ne vous appartient pas avec une configuration logicielle que vous ne contrôlez pas, ne me dérange pas. Après tout, libre à chacun de décider si mettre ses relevés de températures, ses évènements de détection de mouvements ou encore la pression de ses radiateurs est quelque chose de raisonnable ou non.

Ce qui est plus dérangeant à mon sens c'est que cela devienne la solution par défaut. Non parce que c'est simplement bien plus facile à configurer de cette manière, mais parce que tout semble fait pour rendre l'installation « individuelle » vraiment fastidieuse et pénible. Parfois, cela en arrive même à prendre un petit côté « shareware » des années 90 : vous pouvez installer une version relativement ancienne très facilement sur votre Raspberry Pi, mais si vous souhaitez une version récente, soit vous utilisez le service en ligne proposé (cloud donc) soit vous devez être prêt à passer de longues nuits à apprendre, configurer, déjouer les pièges et vous énerver, car toute la chaîne de dépendance entre les éléments est à revoir manuellement.

Un excellent exemple concerne les solutions de *dashboard* permettant d'avoir une page web avec représentation graphique de vos données collectées par des capteurs. C'est presque systématique, dès que l'interface est travaillée, vous pouvez être certain que l'installation individuelle sera extrêmement éprouvante sinon cauchemardesque. On est alors tenté de céder à l'envie de tout simplement se créer un compte et utiliser le service en ligne. Cela ressemble aux jeux « *free to play* » où on se rend rapidement compte que ce n'est pas si « free » que ça...

Je comprends cependant la motivation des développeurs ou des sociétés derrière ces services et logiciels. Le développement coûte cher et il faut bien trouver un moyen de mettre du pain sur la table. Mais pour moi, moralement et philosophiquement, il y a des limites à ne pas dépasser. D'autant que dans d'autres domaines l'équilibre existe (comme WordPress ou Prestashop par exemple). Installer un blog ou une boutique en ligne qui ressemble à quelque chose, seul, sans être développeur/sysadmin, n'est pas insurmontable. Installer proprement un *dashboard* du même acabit est inaccessible à la quasi-totalité des hobbyistes. C'est triste. Énervant et triste...

Mais je suis d'un naturel buté et têtu. Donc non messieurs, je ne créerai pas de compte gratuit, mes relevés de températures me regardent moi et uniquement moi. Et lorsque je serai venu à bout du problème, bien sur terre et non dans les nuages, j'en ferai profiter le plus grand nombre (c'est vous ça, amis lecteurs). Na !

Denis Bodor

Hackable Magazine

est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIERE GRISE

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

Impression : pva, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12

Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
Service des ventes : Abomarque : 09 53 15 21 77
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : 2427-4631
Commission paritaire : K92470
Périodicité : bimestriel
Prix de vente : 7,90 €

MIXTE
Papier issu de sources responsables
FSC® C015136

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

DESSOUDEZ, CHANGEZ ET RÉCUPÉREZ DES COMPOSANTS

Denis Bodor

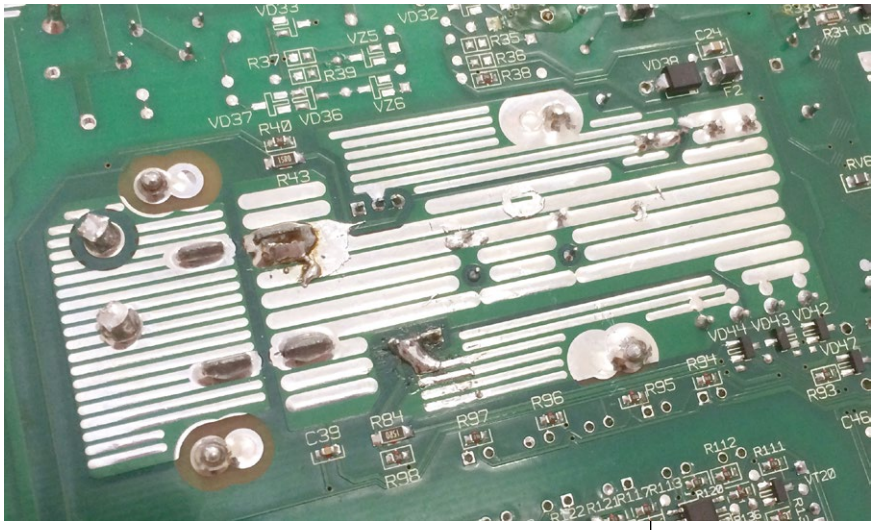
Souder des composants est une activité qu'on finit forcément par avoir dès lors qu'on s'intéresse peu ou prou à l'électronique. Les platines à essais sont, comme leur nom l'indique, principalement faites pour… les essais. Ensuite il faut passer à quelque chose de plus robuste et durable, et donc qui implique des soudures. L'opération inverse est assez différente puisque le but n'est pas de créer mais, soit de récupérer un élément pour un autre usage, soit de changer un élément défectueux. Voyons ensemble comment mener cela à bien avec différentes techniques utilisables…

Souder un composant, que ce soit en surface ou traversant n'est pas quelque chose de très difficile avec un peu d'entraînement et du matériel adapté. C'est une pratique relativement facile à acquérir puisqu'il s'agit, somme toute, de mettre le composant en place, appliquer de la chaleur avec la panne du fer à souder et ajouter de l'étain de soudure qui, une fois fondu, se glissera presque de lui-même là où il doit aller.

L'opération inverse n'est pas aussi aisée, du moins pas pour tous les composants. Deux cas de figure peuvent se présenter, soit le composant à dessouder possède deux ou trois pattes relativement proches et il est possible de les chauffer de concert, soit il dispose davantage de broches et dans ce cas, la solution consiste à retirer l'étain, point par point, afin de libérer le composant.

Une approche « entre deux » est possible dans certains cas puisqu'il est toujours possible d'ajouter de l'étain pour dessouder le tout en une fois, mais ceci ne fonctionnera pas avec, par exemple, un circuit intégré avec une douzaine de broches ou plus. Cette technique peut paraître contre-intuitive, puisqu'on ajoute de l'étain pour dessouder, pourtant elle fonctionne à merveille, mais... tout dépend la raison pour laquelle on dessoude.

En effet, quelle que soit la technique utilisée, il est nécessaire de prendre en compte un élément important : tous les composants n'aiment pas la chaleur, ou plutôt, tous ne la supportent pas de la même façon. Les composants sensibles, et donc souvent les plus coûteux, disposent même, dans leur documentation technique (*datasheet*) d'une température maximum de soudure ainsi qu'un délai durant lequel cette température peut être appliquée. D'autres éléments, comme les connecteurs par exemple, possèdent des supports en plastique qui peuvent fondre en dégageant généralement des fumées tout aussi désagréables que nocives.



Voici l'un des cauchemars du déssoudage de composants : les énormes morceaux de circuit généralement connectés à la masse et servant de radiateur. Mais il y a pire, les circuits multicouches avec ce genre de choses à l'intérieur du substrat...

La question qui se pose alors est : pourquoi voulez-vous dessouder un composant ou un élément ?

1. DESSOUDER POUR RÉPARER : MÉTHODE BARBARE OU PRESQUE

Comme dit précédemment, on peut vouloir dessouder un composant pour le remplacer, car il est défectueux ou supposé comme tel, ou alors, on veut le dessouder pour le récupérer et s'éviter un achat (ou simplement parce que le temps presse). Dans le premier cas, la survie du composant ou de l'élément à dessouder importe peu, mais votre attention doit se porter principalement sur la « survie » du circuit imprimé d'où est retiré le composant.

En effet, une température excessive ou une application prolongée peut avoir des conséquences sur les pistes, tout comme une contrainte mécanique trop importante. Le calme et la patience sont généralement les maîtres mots dans ce genre de situation. Je sais qu'il est très énervant de s'affairer à la tâche et de voir le composant refuser obstinément de bouger, mais la dernière

chose qui doit vous traverser l'esprit est de tenter de brutaliser le composant. Ceci aurait pour effet d'endommager le circuit, décoller une ou plusieurs pistes, voire de retirer un *via* (trou métallisé faisant généralement la liaison entre les diverses couches d'un circuit imprimé).

Dans ce dernier cas, au mieux c'est un circuit double face et vous devrez trouver un moyen de re-router le signal d'une autre manière et au pire, il s'agit d'un circuit multicouche et... vous voilà parti dans une chasse au trésor pour retrouver ce qui est connecté à quoi. Ce qui le plus souvent est une tâche futile, car vouée à l'échec.

Dans bien des cas, il est plus simple de détruire le composant devant être retiré. Pas pour le plaisir ou par cruauté, bien entendu, mais pour faciliter le dessoudage. Dans le cas d'un circuit intégré, par exemple, il est bien plus efficace de délicatement sectionner les pattes du composant pour ensuite les dessouder une à une. Il en va de même, parfois, pour les composants plus petits comme les leds, les résistances, les condensateurs ou encore les transistors. Ces derniers, tantôt vissés ou collés à des radiateurs également soudés au circuit, doivent être séparés mécaniquement avant déssoudage.

Si votre objectif n'est pas de récupérer le composant, votre meilleure amie sera sans le moindre doute une petite pince coupante de précision. Le genre de modèle miniature spécialement destiné aux travaux d'électronique, avec une face plane permettant une coupure nette au ras du circuit. En procédant ainsi, il ne vous restera plus ensuite qu'à retirer les pattes soudées une à une puis nettoyer les *via* avec une pompe à dessouder et/ou de la tresse à dessouder, pour ensuite placer un nouveau composant (cf. plus loin dans l'article).

Remarquez au passage que si votre objectif concerne le remplacement d'un circuit intégré, sur un ordinateur des années 80 par exemple, il sera préférable de souder un emplacement pour circuit intégré plutôt qu'un nouveau composant. Certaines de ces machines possèdent des composants relativement sensibles aux décharges électrostatiques, comme par exemple le CIA 6526 (*Complex Interface Adapter*) du Commodore 64 (oui, encore lui, c'est une obsession chez moi), pouvant être endommagé en touchant simplement les connecteurs joystick. Dans ce genre de situations, si vous prenez le temps de dessouder le composant pour le remplacer, autant préparer le terrain pour un éventuel prochain problème.

Parfois, une pince n'est même pas nécessaire, certains condensateurs par exemple peuvent être « arrachés » (par rotation), ne laissant que leurs pattes qui peuvent ensuite être dessoudées proprement, comme l'a montré la chaîne YouTube *Mr Carlson's Lab* (fantastiques vidéos si vous êtes anglophone et amateur d'électronique vintage).

De manière générale, dès lors qu'il s'agit de réparer un circuit en changeant les composants (le plus souvent les condensateurs puisque les problèmes viennent généralement de là dans 80% des cas, sinon plus) la seule chose à garder à l'esprit est de n'agir que dans le but de laisser le circuit intact. Peu importe l'état final du composant retiré, la seule chose qui compte c'est le circuit et ses pistes.

2. LES DIFFÉRENTES TECHNIQUES DE DÉSSOUDAGE

Si vous souhaitez dessouder des composants afin de les réutiliser, l'approche est totalement différente. Là, l'état final du circuit sur lequel ils se trouvent n'a aucune espèce d'importance et dans certains cas il est même envisageable d'attaquer le circuit à la disqueuse Dremel pour minimiser l'effort de soudure/déssoudage.

Le principal problème dans cette situation est de ne pas endommager le composant en le faisant monter trop haut en

température ou en le chauffant trop longtemps. Vos principaux ennemis ici seront les grandes surfaces de cuivre, souvent étamées, connectées à la masse (*ground plane*). Ces surfaces permettent une importante dissipation thermique qui, de fait, réduisent la concentration de chaleur à l'endroit où l'on souhaite qu'elle soit la plus importante et rapidement appliquée : les pattes des composants. Dans certains cas, comme avec les transistors et les MOSFET au format TO-220 par exemple, ces surfaces sont d'ailleurs justement parfois utilisées comme « radiateur ». Il peut être alors intéressant de s'attaquer directement au circuit pour réduire la taille de la surface et donc faciliter l'utilisation du fer.

Un autre cas de figure qui peut se présenter concerne les circuits multicouches comprenant tantôt un *ground plane* interne. Il est cependant plus rare de chercher à récupérer des composants sur ce type de circuit, comme les cartes mères de PC, mais la solution est exactement la même : réduire par tous les moyens possibles la taille de cette surface et donc, parfois, en venir à faire des petits morceaux avec le circuit.

2.1 Ajouter de l'étain

Comme évoqué précédemment, lorsque le composant à récupérer est de petite taille, que ses pattes sont peu nombreuses et proches les unes des autres (transistors en boîtier TO-92 ou leds, par exemple), ou qu'elles peuvent être dessoudées en plusieurs fois (résistances par exemple), la meilleure technique consiste sans doute à n'utiliser rien d'autre que la panne de son fer à souder et de l'étain.

L'opération se résume à ajouter le l'étain jusqu'à ce que l'ensemble, en fusion, touche toutes les pattes et, normalement, le composant tombe de lui-même. Il est possible de s'aider d'une pince ou de tout simplement faire jouer la gravité en lui donnant un petit coup de pouce via un mouvement sec du circuit. Vous pouvez également utiliser vos mains si vous avez une certaine tolérance à la douleur, même si après quelques « aïe » et « ouille » on finit souvent, miraculeusement par se souvenir où on a rangé sa pince préférée...

Mais le fait d'ajouter de l'étain de soudure n'est pas simplement une solution thermique pour déloger le composant. La plupart des étains de soudure (neufs) contiennent du flux, un composé chimique destiné à décaper et nettoyer le circuit et la patte du composant lors de la soudure. Ce flux est également utile pour passer la couche d'oxyde (ou de crasse) qui se forme souvent sur les vieilles soudures. Ceci permet aussi de diluer l'étain déjà en place avec le vôtre de façon à baisser sa température de fusion. En effet, selon le rapport étain/plomb, la température peut grandement varier (source : le guide « *Better Soldering* » de Weller) :

- 40/60 : 230°C ;
- 50/50 : 214°C ;
- 60/40 : 190°C ;
- 63/37 : 183°C ;
- 95/5 : 224°C.

Les étains sans plomb, maintenant obligatoires, ont généralement un point de fusion autour de 220°C.

Notez que, quels que soient l'étain et l'état du circuit, il est important d'assurer une ventilation adéquate lors de l'opération de déssoudage. Ceci est également valable de façon générale pour une soudure, mais dans le cas qui nous occupe ici, vous ne savez pas à quel produit chimique ou composé vous avez affaire. Une bonne solution pour éviter de respirer ces fumées, qu'on se doit d'estimer nocives par défaut, est d'utiliser un extracteur de fumées. Il en existe de toutes sortes vendus dans le commerce, mais une excellente solution économique consiste à recycler un ventilateur de PC et lui adjoindre un filtre à charbon actif. Le flux d'air aspiré du point de déssoudage capturera la majeure partie des éléments nocifs et évitera que vous vous intoxiquiez pour rien.

Le flux de soudure permet un nettoyage des pistes, des pattes des composants et de la tresse à dessouder. Ici une version en pâte, mais il en existe de nombreuses déclinaisons : liquide, en seringues, en feutres, etc. Comme vous pouvez en juger vu l'état, ce genre de choses se consomme lentement et votre petit durera donc longtemps...

La tresse à dessouder n'est rien d'autre que du fil de cuivre tressé. Elle est généralement vendue en petits rouleaux comme ceux-ci, par lot, pour quelques euros.

Toujours du point de vue de la sécurité, il est également recommandé de porter des gants (type latex, vinyle ou nitrile) et de vous laver les mains après toutes manipulations. Comme vous ne savez pas exactement de quoi est composé l'étain en place, mieux vaut vous montrer prudent. La directive RoHS visant à limiter l'utilisation de substances dangereuses comme le plomb et le cadmium, n'est effective que depuis 2006 et n'est pas applicable partout. Dans le doute, mieux vaut donc éviter tout risque de contact ou d'ingestion avec ces substances inconnues.

2.2 La tresse à dessouder

Lorsque le composant ou l'élément à dessouder/récupérer possède de nombreuses pattes (circuit intégré, réseau de résistances, connecteurs DB25, optocoupleurs, etc.) ou des pattes distantes les unes des autres sans possibilité de tout chauffer en une fois (relais, transformateurs, connecteurs d'alimentation, etc.), vous n'avez plus qu'une solution : retirer le maximum d'étain pour libérer chaque patte, et donc à terme, le composant tout entier.

Là, c'est un peu comme la légendaire question du vidage des sardines pour le barbecue, nous avons deux écoles et chacune d'elle a ses fervents et dévoués paroissiens : la pompe à dessouder (voir ci-après) et la tresse à dessouder.

Cette dernière se présente comme un simple ruban de cuivre (ou d'un autre métal plaqué de cuivre) tressé tantôt pré-imprégné de flux. Son principe de fonctionnement est simple, puisqu'il suffit de placer le bout de la tresse sur un point de soudure du composant et d'y poser la pane du fer à souder. La tresse chauffe, puis l'étain devient liquide et celui-ci est aspiré dans la tresse par capillarité. C'est exactement la même chose que lorsque vous épongez de l'eau avec de l'essuie-tout.

Une fois tout l'étain aspiré, il suffit de retirer le fer et la tresse. L'étain absorbé se solidifie en refroidissant et on coupe la partie imprégnée de la tresse avec une pince avant d'attaquer la patte suivante.

Si la tresse n'est pas déjà traitée dans ce sens, l'apport de flux de soudure facilite grandement les choses en permettant de supprimer l'inévitable couche d'oxyde se formant sur le cuivre. Le fer à souder a pour tâche ici de chauffer la tresse, l'étain, la patte du composant et la piste du circuit imprimé de façon à pouvoir aspirer le maximum, sinon l'intégralité, de l'étain en place. Bien entendu, jouer avec la panne du fer à souder de manière à décoller la patte

du composant du via où elle se trouve facilite le retrait final.

La forme de la coupure de la tresse joue également un rôle. Taillée en biais, par exemple, elle sera plus à même d'être glissée au plus proche du via pour nettoyer tout résidu d'étain. Il existe également plusieurs largeurs de tresse à dessouder, utilisables en fonction de la quantité de matière à aspirer. Plus la tresse est large, plus il faudra appliquer de chaleur pour lui faire faire son travail. Personnellement, je trouve qu'une tresse de 2 mm de large permet de dessouder efficacement tous les composants intéressants.

La tresse à dessouder se trouve dans n'importe quelle boutique en ligne pour quelques euros, mais gardez à l'esprit que cela reste un consommable. Il en existe de plusieurs qualités et marques, mais pour un usage courant l'achat auprès d'un vendeur de ShenZhen sur eBay est bien suffisant. On peut ainsi trouver des mini rouleaux de 1,5m par lot de 5 entre 2€ et 3€, port offert. Il suffit de chercher « *desoldering braid* » pour être inondé d'annonces en tout genre...

Le principe de fonctionnement de la tresse est simple : on fait chauffer la tresse, la patte du composant et l'étain et ce dernier, fondu, est aspiré par capillarité dans la tresse.

Le principal problème de la tresse à dessouder est le risque de surchauffe et de décollage des pistes (dans le cas d'une réparation). Ceci cependant est une affaire de technique et en particulier du fait de se retenir d'appliquer trop de contraintes mécaniques. La tresse ne fonctionnera pas mieux en appuyant de toutes vos forces ou en frottant la surface. Laissez faire son travail au fer et à la tresse et, éventuellement, ajoutez du flux pour faciliter l'aspiration. Au bout de quelque temps, vous développerez naturellement une certaine sensibilité... ou finirez par détester la tresse à dessouder.

2.3 La pompe à dessouder

La pompe à dessouder est restée longtemps mon accessoire préféré avant de me rendre compte finalement qu'il n'y a, en réalité, pas de technique exclusive pour le déssoudage de composants. On choisit simplement le bon outil en fonction de la tâche à accomplir et de l'humeur du moment.

La pompe à dessouder ne nécessite pas de consommable et agit par aspiration mécanique. Il s'agit d'un outil de la taille d'un gros feutre, composé d'un cylindre et d'un piston mû par

Une pompe à dessouder manuelle se résume à un piston armé en appuyant sur l'arrière de l'objet. On fait ensuite fondre l'étain et on déclenche la gâchette. Le ressort interne pousse le piston dans sa position de départ et l'étain est aspiré.

L'étain aspiré se retrouve dans le corps de la pompe qu'il faut vider régulièrement et tantôt déboucher. Un petit coup de lubrifiant après un nettoyage complet est généralement une bonne idée pour garantir un fonctionnement correct.

un ressort. Pour l'utiliser, on fait tout d'abord fondre l'étain à la base d'une des pattes du composant à dessouder puis, rapidement, on pose l'embout en téflon sur la patte et on déclenche le mécanisme. Le piston, poussé par le ressort interne, recule rapidement, créant une aspiration d'air qui entraîne l'étain fondu avec elle. Il est également possible d'utiliser la pompe et la panne du fer de concert, mais l'espace vide alors présent rend l'aspiration peu efficace.

En principe, cette aspiration, si l'embout est bien appliqué et l'étain encore bien liquide, permet de retirer toute la matière d'un coup, qui se retrouve alors dans le corps de la pompe. Après quelques opérations de ce type, l'étain finit invariablement par presque boucher l'embout, mais les concepteurs de l'outil ont pensé à tout. Il suffit, en effet, d'armer le piston puis de pousser encore davantage. Une tige interne sort alors par l'embout pour le libérer de l'étain excédentaire.

La pompe à dessouder fonctionne à merveille sur les petites pattes des composants et de préférence celles qui ne sont pas soudées à de grosses pistes ou de grandes surfaces de cuivre. Dès lors qu'on retire la panne du fer, la chaleur se dissipe et l'étain refroidit et il n'est pas rare que l'aspiration arrive bien trop tard. Il est alors parfois nécessaire de remettre de l'étain pour tenter de l'aspirer à nouveau en une fois, ou de tout simplement basculer sur la tresse à dessouder pour retirer ce qu'il reste.

Mais le point le plus pénible, à mon sens, dans l'utilisation de la pompe est le fait de devoir la nettoyer. Le corps dans lequel se trouve le piston finit par se remplir de projections d'étain, couvrant la surface intérieure, le ressort, la tige interne, etc. Souvent, le ressort ne fait plus son travail tant l'ensemble est encrassé et il faut donc régulièrement dévisser l'embout puis procéder à un nettoyage intégral pour revenir à un fonctionnement normal.

Une pompe à dessouder vous coûtera entre 3€ et 15€ (pour les modèles bling-bling de marque) et se trouve un peu partout. Il existe même des modèles intégrés à des fers à souder de 30W équipés de pannes creuses pour une dizaine d'euros. Je suis cependant assez réticent à cette combinaison n'offrant généralement aucune possibilité de réglage de température et semblant plus adaptée à des travaux d'électricité que d'électronique.

2.4 Il y a d'autres solutions

L'usage de la pompe ou de la tresse est sans le moindre doute la solution économique la plus courante. Mais en dehors de celle dont je vais parler dans un instant, il en existe d'autres. On peut par exemple citer l'utilisation d'une station de soudure à air chaud, sorte de croisement entre un sèche-cheveux miniature et la station de soudure/soudage. Utilisée généralement pour le soudage des composants de surface (SMD, CMS, etc.), celle-ci souffle de l'air chauffé à une température réglable entre 100°C et 500°C (selon le modèle) et permet donc de dessouder des composants tout aussi facilement.

Cette solution coûte cher et il est important de prendre garde, en fonction des composants, à ne pas surchauffer les composants, au risque d'endommager le précieux objet de votre convoitise.

Une autre solution alternative est parfois intéressante mais, là aussi coûteuse : l'utilisation d'un composé avec un point de fusion très bas. L'une des marques les plus

connues est *Chip Quik* qui se présente comme une pâte ou un fil ressemblant à s'y méprendre à de l'étain classique. L'idée est ici de faire fondre ce produit sur les broches du composant de manière à ce qu'il se mélange avec l'étain déjà en place. Son point de fusion étant bien plus bas que l'étain classique (environ 80°C) et comprenant des composés chimiques et des alliages ramollissant l'étain en place tout en refroidissant très lentement, la soudure ne tient tout simplement plus et le composant peut alors être retiré rapidement et sans effort.

Le produit en question coûtant quelques 15€ ou 20€ pour 70 cm de fil, il est évident que cette solution sera réservée aux composants ou aux situations où d'autres solutions ne sont pas envisageables ou trop risquées. Utiliser 5€ d'étain spécial pour récupérer un composant neuf à quelques euros n'a pas vraiment de sens.

3. UNE STATION DE DÉSSOUDAGE, ÇA CHANGE LA VIE !

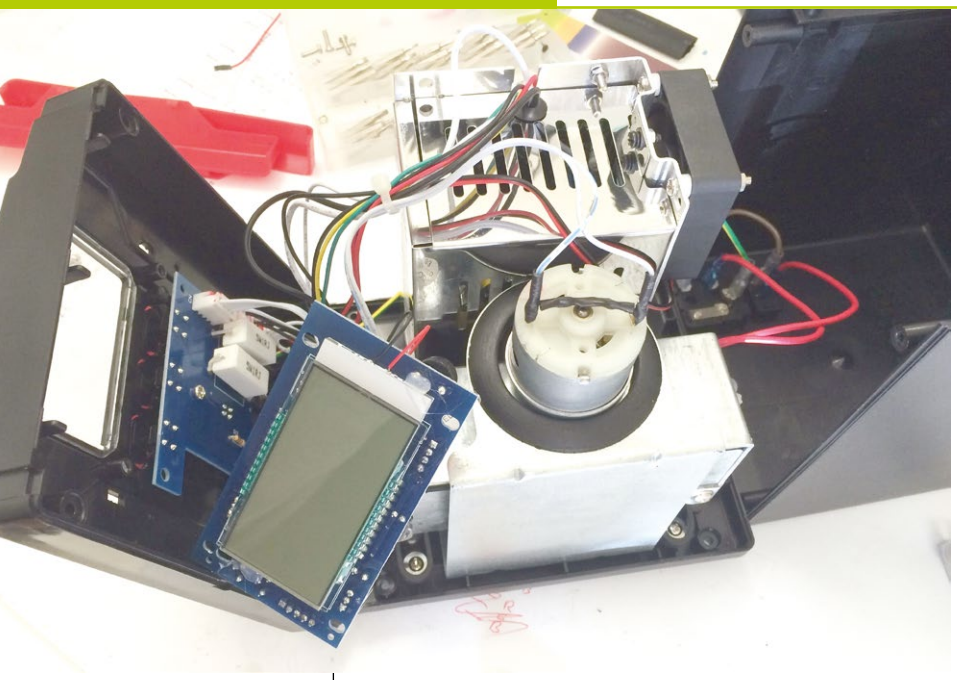
J'ai toujours jonglé entre les trois techniques de déssoudage déjà décrites ici, me satisfaisant des bienfaits et des lacunes de chacune sans pour autant vouloir investir dans un équipement plus conséquent et plus cher.

Et puis récemment j'ai vu une vidéo de la chaîne YouTube EEV Blog où le très australien *Dave Jones* se risquait à tester une station de déssoudage d'entrée de gamme acquise sur eBay. Connaissant le bonhomme et son avis sur les matériels de ce type, j'ai été surpris de constater que, finalement, le verdict n'était pas si catastrophique, voire plutôt bon. Le matériel testé était une station de déssoudage ZD985 de marque *Rhino Tools* mais, comme le précise Dave dans la vidéo, ce type de matériel est finalement générique et vendu sous bien des marques et désignations.

J'ai donc ainsi, en relative confiance, jeté mon dévolu sur un modèle assez similaire, voire presque totalement identique, vendu sous la désignation ZD-8915 sur eBay auprès d'un vendeur espagnol appelé « *japangameonline* » (mais d'où ces vendeurs pêchent leur nom ?). Le prix de l'objet, quelques 130€, plus 40€ de port (outch !) était identique aux nombreuses offres de ZD985 également présentes (les ZD985 et/ou ZD915 peuvent être trouvé tantôt aux alentours de 120€ port inclus). En dehors de l'aspect du matériel, celui-ci se différencie par le fait que le réceptacle d'étain était transparent et potentiellement en verre, alors que celui du modèle ZD985 est en plastique et opaque. Misant sur le fait que le reste du matériel serait de la même qualité que le modèle testé par Dave, et les entrailles identiques, j'ai donc pris ce petit risque qui s'est finalement avéré être gagnant.

Le principe de fonctionnement de la station de soudage est une simple fusion entre la station de soudage et la pompe à dessouder. La base de l'appareil intègre ainsi l'alimentation, l'électronique de contrôle de température et une pompe permettant l'aspiration de l'étain fondu.

La station de déssoudage est un croisement entre la station de soudage et la pompe. La panne au bout du pistolet est creuse et une pression sur la gâchette déclenche la mise en route de la pompe électrique placée dans la base de la station.

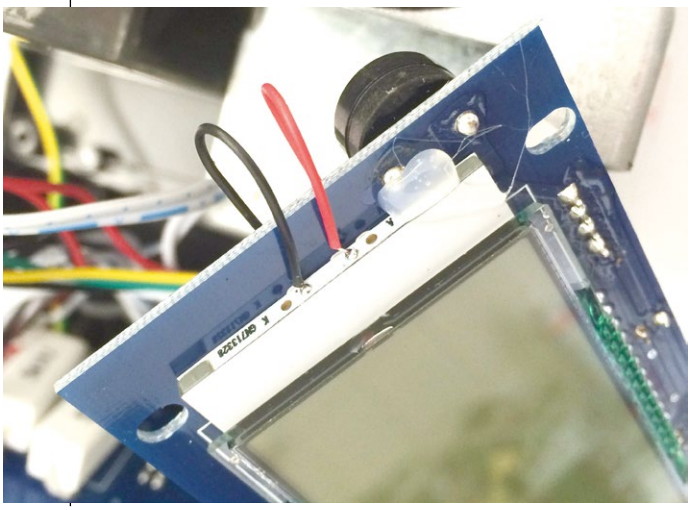


Ce modèle de station de déssoudage, d'entrée de gamme, est d'un niveau de qualité tout à fait respectable, contrairement à ce que la coque en plastique pourrait laisser penser.

Le pistolet qui est relié à cette base, à la fois avec un câble et un tuyau souple, se compose d'une panne creuse et d'un réceptacle permettant d'accueillir l'étain. L'activation du corps de chauffe et le choix de la température sont gérés par la base et le déclenchement de la pompe se fait via une gâchette sur le pistolet lui-même.

Ce modèle spécifique (je ne saurai l'affirmer pour les autres) est livré avec :

- un câble d'alimentation (un peu court) avec une prise EU,
- un support métallique pour poser le pistolet,
- une éponge pour nettoyer la panne,
- un jeu de trois pannes de tailles différentes,
- deux filtres prenant place dans le raccord côté base,
- quatre filtres se plaçant à l'arrière du réservoir sur le pistolet,
- trois tiges (type cure-pipe) permettant de nettoyer les pannes creuses.



Un petit coup d'œil à l'électronique de la bête montre que l'afficheur LCD est rétroéclairé à l'aide d'un module en plastique tenu en place avec de la colle thermique. Personnellement, je prends littéralement ceci comme un appel à modifier cette partie du matériel !

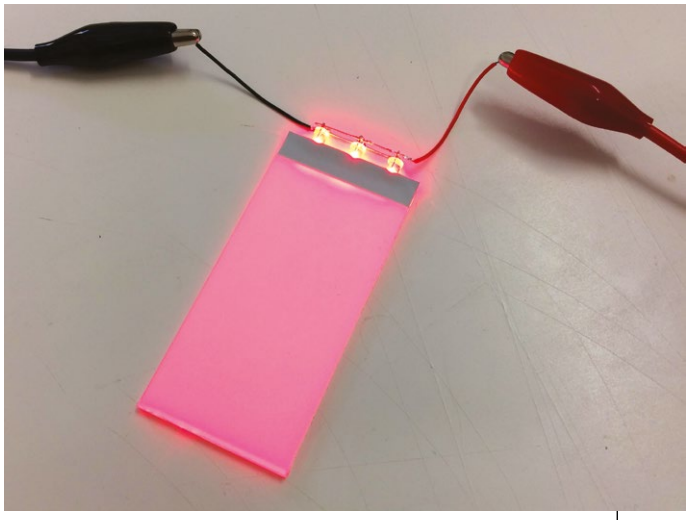
Les trois pannes sont livrées étamées et bouchées par de l'étain, la première opération consiste donc à raccorder l'ensemble, mettre la base en route, laisser chauffer la panne et jongler entre aspiration et usage d'une des tiges pour déboucher l'orifice. Ceci fait, on peut faire son premier essai en plaçant la panne autour d'une patte de composant, agitant légèrement l'ensemble pour libérer la patte à mesure que l'étain fond, puis en déclenchant l'aspiration. Et effectivement, tout comme le modèle ZD985, le déssoudage est rapide et sans bavure.

Il y a des moments dans la vie où on se dit qu'on aurait vraiment pu gagner un temps précieux en faisant preuve d'un peu d'ouverture et davantage de curiosité. Ce premier essai était l'un de ceux-là. Ces matériels semblent être disponibles depuis un certain temps et même si l'investissement reste non négligeable (quelques 170€) par rapport à la pompe ou la tresse à dessouder, le gain en souplesse, en vitesse et en facilité est énorme. Récupérer un connecteur DB9 ou DB15 se fait en un temps record, tout comme celui d'un petit transformateur, un circuit logique ou même un MOSFET de puissance généreusement soudé. La plupart du temps, les composants tombent tout simplement du circuit une fois l'étain aspiré sur la dernière broche.

Les composants récupérés sont parfaitement propres, tout comme le circuit. Le matériel est donc parfaitement adapté aussi bien pour la récupération sauvage que pour les réparations. Bien entendu, le budget à prévoir doit être en rapport avec l'usage du matériel. Pour un usage intensif, ce type de produit sera très certainement insuffisant, et pour une simple utilisation très occasionnelle la dépense ne sera pas justifiée. Sachant qu'une station de déssoudage de marque (Weller, Hakko, JBC, etc.) représente une coquette somme à quatre chiffres au minimum, ce matériel est, je pense, un juste équilibre pour un usage hobbyiste.

Du point de vue qualitatif, l'indispensable démontage de l'appareil révèle une construction relativement robuste et de qualité. Certes le plastique utilisé pour la coque est vraiment peu rassurant mais, à l'intérieur, tout est parfaitement isolé et assemblé proprement sans pour autant jouer dans le haut de gamme. La pompe est, par exemple, montée sur des supports souples pour limiter les vibrations, les raccords des tubes sont renforcés avec des colliers, la terre est sertie et fixée correctement avec une vis et non soudée, la plupart des vis sont sécurisées avec du vernis, etc.

Le seul détail qui me chipotait était d'ordre cosmétique et philosophique (cf. précédent édito). J'ai donc profité de l'ouverture de l'engin pour apporter ma touche personnelle en changeant la couleur du rétroéclairage de l'écran du blanc vers le rouge avec un résultat du plus bel effet, en accord avec le mélange noir/rouge du boîtier



Après quelques expérimentations hasardeuses, et le retrait des leds blanches, trois leds rouges 3 mm super-lumineuses (8000 mcd) sont finalement collées à la cyanoacrylate et soudées en parallèle. Le résultat est plutôt convaincant...

et son côté très « IBM » (j'ai remarqué ensuite que le voyant s'allumant lors du déclenchement de la pompe était bleu, ceci sera corrigé sous peu). Il est également fort probable que je déplace le bouton de mise sous tension de l'arrière à l'avant de la base, car l'emplacement initial n'est vraiment pas pratique (malgré la fonction de mise en veille automatique qui ne fait que baisser la température à 200°C, mais n'éteint pas l'appareil).

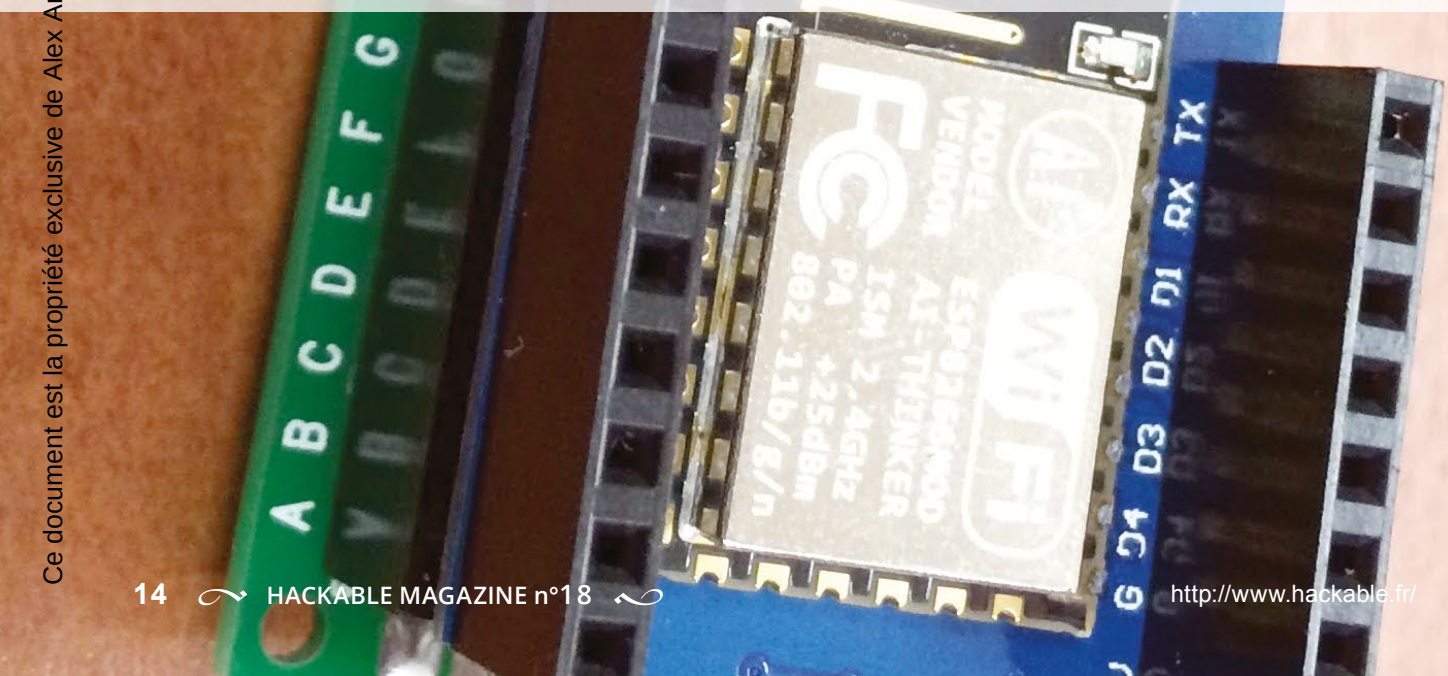
Le matériel étant maintenant réellement mien, il va rejoindre le reste de mon matériel en bonne et due place, à portée de main, prêt à servir à la moindre occasion ! **DB**

Et voilà, ma station de soudage est maintenant esthétiquement unique avec son écran rouge. Il ne reste plus qu'à déplacer le bouton de mise en route en façade et elle sera parfaite !

DÉMARREZ VOTRE PC À DISTANCE AVEC UNE CARTE ARDUINO OU ESP8266

Denis Bodor

Vous savez comment ça marche, à chaque fois que vous avez besoin d'un fichier il se trouve à l'autre bout du réseau sur une machine qui n'est pas en marche… Il existe des solutions intégrées permettant de démarrer un PC à distance, ceci s'appelle le Wake-on-LAN. Malheureusement, cette fonctionnalité n'est pas toujours présente et/ou utilisable. Pourquoi alors ne pas construire un montage permettant de presser le bouton à votre place, à distance ?



Le *Wake-on-LAN* ou WoL est une fonctionnalité standardisée dans les réseaux Ethernet permettant de démarrer une machine à distance. Celle-ci est souvent configurable dans le BIOS de la machine, mais ce n'est pas toujours le cas. Parfois l'option est totalement absente, et tantôt elle ne permet qu'une sortie de mise en veille. Dans mon cas, c'est cette seconde option qui est disponible, mais la machine en question étant équipée de quelques 16 Go de mémoire je n'ai pas pensé à prévoir une partition de *swap* dans mon système GNU/Linux lors de son installation. Or justement, en cas de mise en veille profonde, le contenu de la mémoire est copié dans cette partition avant extinction de l'alimentation. Pas de partition, pas de mise en veille profonde...

Mais la problématique va au-delà de ce petit désagrément. En effet, le *Wake-on-LAN* ne règle le problème de la mise en route à distance que pour une certaine catégorie de matériel compatible. Un vieux PC, une machine de collection (comme un NeXT ou un vieux Mac), un stockage réseau (NAS), une imprimante, ou n'importe quel équipement connecté en réseau filaire ou en Wifi, a toutes les chances de ne pas supporter une telle fonctionnalité. Enfin, le *Wake-on-LAN* ne fonctionne, bien entendu, qu'en connexion filaire et non en Wifi. Mettre un PC en route s'il ne dispose pas d'une connexion Ethernet demande un peu plus d'ingéniosité que de simplement utiliser des fonctions mises à disposition par le constructeur.

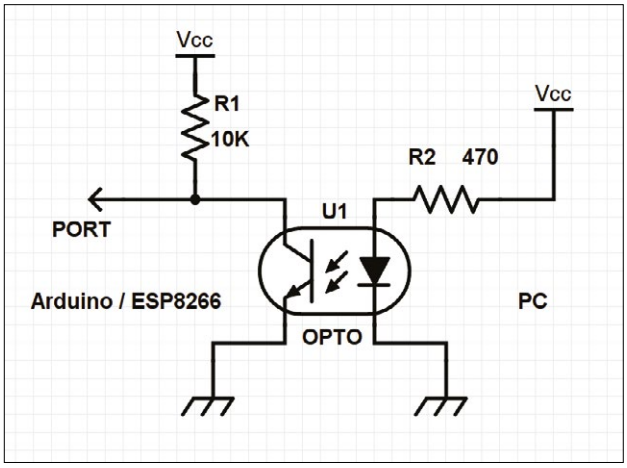
1. UN MONTAGE POUR CONTRÔLER LE DÉMARRAGE DU PC

Notre approche ici consistera à utiliser une carte Arduino équipée d'un *shield* Ethernet, mais nous envisagerons également une solution Wifi équivalente dans le même temps, reposant sur un module ESP8266 (type ESP-12 et/ou NodeMCU). Les bibliothèques `Ethernet` pour le shield et `ESP8266WiFi` pour l'ESP8266 sont suffisamment similaires pour que 90% du code reste identique, quelle que soit la solution choisie, filaire ou non.

Le principe de base reposera sur un serveur Telnet fonctionnant sur le microcontrôleur et permettant d'entrer des commandes et d'obtenir des réponses. Cette interface, accessible à partir de n'importe quelle machine du réseau (ou éventuellement au travers d'une box) fournira un moyen de contrôler un relais directement connecté en parallèle du bouton de mise sous tension. Nous utiliserons également une entrée de la carte, connectée à l'intérieur du PC à une source de +5V afin de vérifier que la machine est sous tension. Ceci, couplé avec le test d'un service réseau sur le PC, nous permettra de savoir si la machine est ou non en fonction et en ligne, afin de s'abstenir d'utiliser le bouton de façon abusive.

Plusieurs remarques sont nécessaires pour expliquer ces choix. Mon objectif ici est de mettre sous tension à distance une machine GNU/Linux pour, ensuite, m'y connecter via SSH. Il doit être possible d'obtenir un fonctionnement identique avec une machine Windows en utilisant un bureau à distance, VNC et/ou des outils comme ceux fournis par Cygwin.

Autre point important, mes machines sont généralement connectées entre elles par un VPN (réseau privé virtuel) les interconnectant dès leur mise en marche. Ceci permet de les rendre accessibles où qu'elles se trouvent (et où que je me trouve) à partir du moment où on est connecté au VPN. Généralement, une machine dans le lot est toujours allumée, car dédiée à la réception SDR (radio logicielle) et donc connectée au VPN. L'idée, en ce qui me concerne est donc de me connecter, où que je sois, à mon VPN, puis à cette machine (en SSH), puis à partir de là, me connecter en Telnet sur le montage décrit ici pour mettre en route le PC équipé du montage. Le VPN utilisé est OpenVPN, qui est utilisable aussi bien sous GNU/Linux que sous macOS ou Windows, mais d'autres solutions peuvent également être envisagées, y compris configurer



Mise en œuvre classique d'un optocoupleur avec ici à droite le côté PC (fil rouge d'un connecteur d'alimentation pour disque par exemple) et à gauche le côté Arduino/ESP8266. On voit clairement que les deux parties sont électriquement isolées l'une de l'autre.

sa box ou son routeur Internet pour rendre accessibles une ou plusieurs machines du réseau local depuis l'extérieur (*port forwarding* et/ou DMZ).

Vous l'avez compris, le montage n'est accessible ici que par le réseau local et ce choix est délibéré. Telnet, le protocole utilisé, consiste en un simple échange de caractères sans aucune protection ou chiffrement. La seule sécurité en place est la restriction de l'accès physique (Ethernet) ou le chiffrement du Wifi. Une communication Telnet (ou RSH) circulant sur le net peut être lue sans le moindre problème et c'est précisément la raison d'être de protocoles chiffrés et sécurisés comme SSH. Malheureusement, aussi bien pour Arduino que pour l'ESP8266, ceci est hors d'atteinte et nous devons nous contenter de communication « en clair » (l'ESP8266 offre des solutions comme `WiFiClientSecure` pour l'HTTPS, mais l'approche devrait être alors totalement différente). Je vous recommande donc fortement de ne pas rendre ce montage directement accessible depuis Internet.

En dehors de l'aspect purement « réseau », notre projet devra pouvoir contrôler le bouton de mise sous tension du PC et vérifier si la machine est en route ou non. Il n'est, bien entendu, pas question de connecter directement l'Arduino ou l'ESP8266 à la fois à une alimentation USB et électriquement au PC, nous utiliserons donc un relais pour contrôler la mise sous tension (avec des contacts en parallèle à ceux du bouton intégré au boîtier) et un optocoupleur pour détecter la présence d'alimentation dans le PC. De plus, deux leds (une verte et une rouge) nous permettront de rapidement notifier l'état de l'installation pendant le fonctionnement du serveur Telnet.

Un optocoupleur se résume à une led (généralement infrarouge) et un phototransistor réunis dans un petit boîtier. Ceux utilisés ici sont des Cosmo K1010 récupérés d'un circuit imprimé d'onduleur. Le fonctionnement de ce composant est relativement simple puisque la lumière de la led agit comme la base du transistor qui laisse alors passer le courant du collecteur vers l'émetteur. Il suffit de relier la led via une résistance à la partie qui contrôle et le collecteur du transistor, via une résistance de rappel à la tension d'alimentation, et à la partie contrôlée. L'émetteur du transistor et la cathode de la led sont tous deux reliés à la masse de **leur** circuit. Nous avons donc une isolation galvanique entre les deux parties. Du point de vue du PC, il ne s'agit que d'allumer une led via un des connecteurs d'alimentation interne (disque, ventilateur, led *power*, USB, etc.).

Les relais nécessitent généralement un peu plus de circuiterie, car il n'est pas pensable de contrôler directement la bobine du relais via une broche d'une carte Arduino, par exemple. On utilise donc en principe un transistor, une diode et une résistance afin de piloter la bobine via un courant circulant à la base du transistor. Il ne vous sera cependant pas nécessaire de construire ce circuit vous-même, car des modules intégrant tout cela (et tantôt également un optocoupleur) sont disponibles entre 1€ et 2,5€ sur eBay, Amazon, etc. Ces modules sont prévus pour être utilisés avec des cartes comme Arduino et requièrent simplement une masse, une alimentation et un signal de contrôle.

Notez que le plus souvent ces modules utilisent une logique inversée : le relais est actif lorsque

la broche de contrôle est à la masse. Ceci peut paraître surprenant, mais c'est quelque chose de fort pratique. En effet, par défaut lors de la mise sous tension, les broches de cartes comme Arduino sont généralement configurées en entrée et celles-ci ne sont donc pas connectées à la masse. De ce fait en configurant, dans le croquis, d'abord l'état de la broche avec `digitalWrite()` puis en la passant en sortie avec `pinMode()`, votre module relais ne sera jamais à la masse et ne s'activera donc pas lors d'un démarrage ou d'un reset de la carte Arduino.

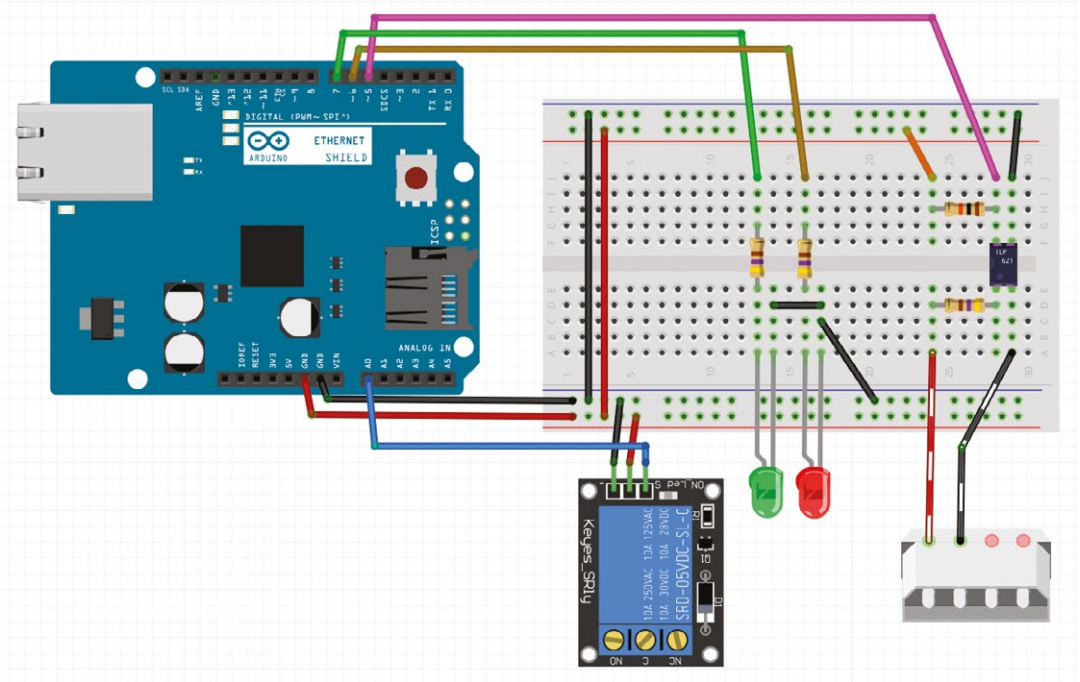
Remarques : que vous optiez pour la solution Arduino ou ESP8266, lisez les deux parties qui suivent. Les explications sont interchangeables et se complètent l'une l'autre, quelle que soit la plateforme.

1.1 Version Arduino + shield Ethernet

La version Arduino avec shield Ethernet est un peu plus simple que celle reposant sur l'ESP8266/NodeMCU, car utilisant 5V. Ceci nous permet de contrôler directement l'activité du module relais depuis une broche de la carte (A0 ici). Les leds de notification sont également directement connectées aux broches 7 (vert) et 6 (rouge).

La seule partie un peu plus complexe est celle concernant l'optocoupleur utilisant une résistance de rappel de 10 Kohm entre la tension d'alimentation et le collecteur du phototransistor ainsi qu'une connexion au port utilisé en lecture côté Arduino (5) et à la masse. La logique est ici inversée, lorsque la led de l'optocoupleur n'est pas allumée, l'Arduino va lire un niveau haut et lorsqu'elle l'est un niveau bas. En d'autres termes, si le PC est en route, l'Arduino lira `LOW` et non `HIGH`.

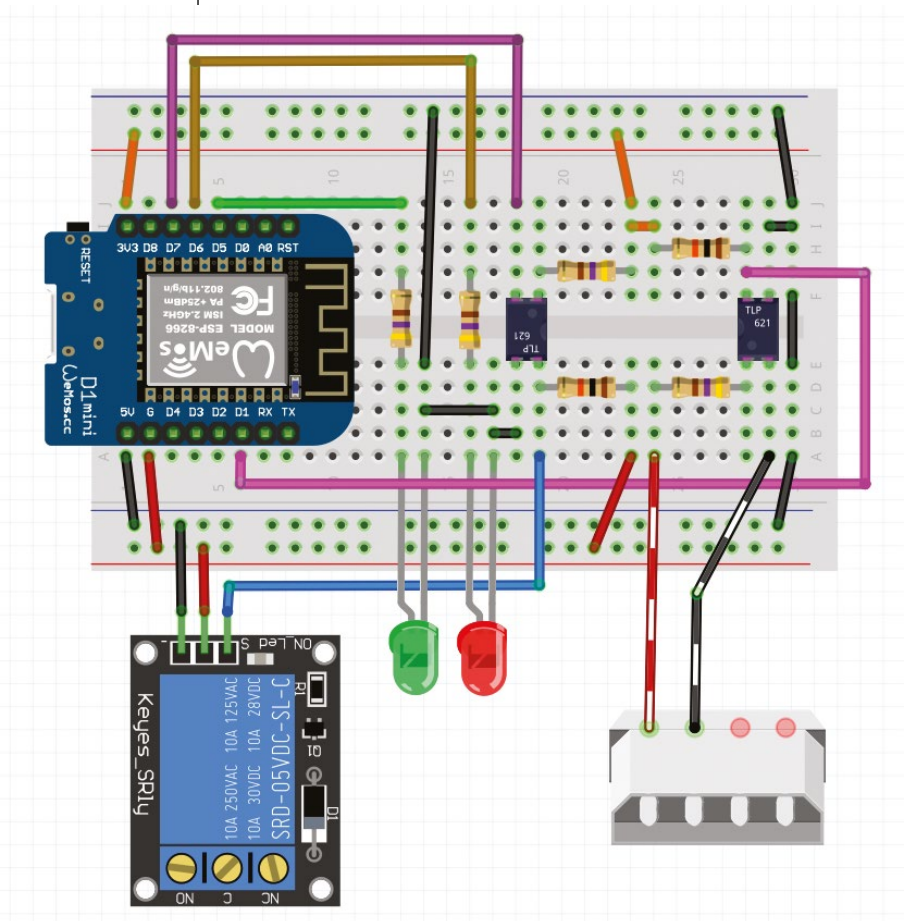
Notez que nous partons ici du principe que la led de l'optocoupleur est connectée à une alimentation +5V et que la résistance de 470 ohms est donc calculée dans ce sens. Si vous voulez tester une autre tension (+12V par exemple), il faudra adapter la valeur de la résistance.



1.2 Version ESP8266/NodeMCU/ WeMosD1

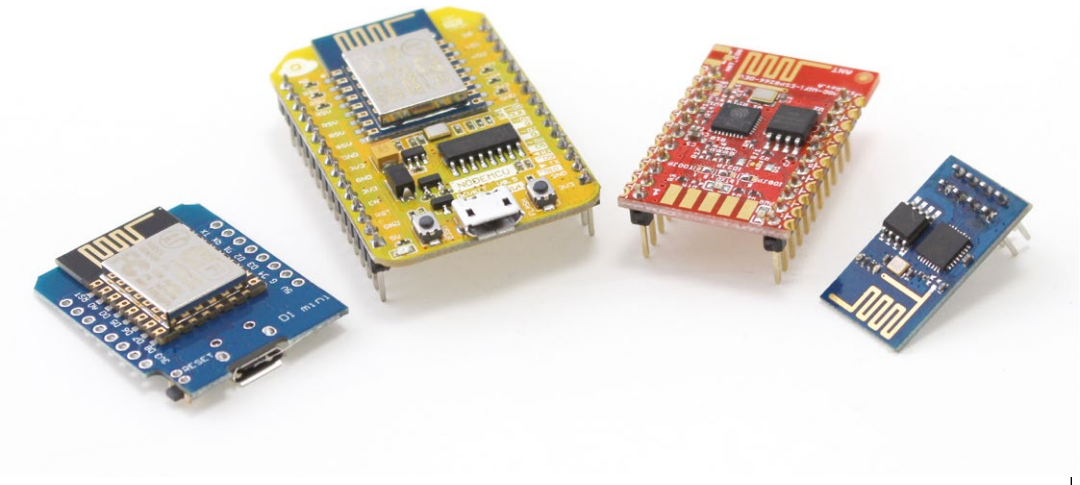
Dans les grandes lignes, le principe de fonctionnement est ici identique à celui d'une carte Arduino et son shield Ethernet, seuls les ports utilisés sont différents. Les leds et l'optocoupleur de détection de mise sous tension sont agencés de la même façon, mais nous devons contourner un petit problème : l'ESP8266 fonctionne en +3,3V, il est donc incapable de contrôler le module relais directement (ceci sera toutefois dépendant du module en question et de son transistor intégré).

Ici, le module utilisé ne possède pas d'optocoupleur qui règlerait le problème automatiquement et nous devons donc l'ajouter nous-mêmes. Ceci présente l'avantage d'offrir en principe une isolation supplémentaire, mais comme nous avons connecté l'alimentation du module relais à l'ESP8266, cet avantage disparaît (ce n'est pas bien grave avec un relais contrôlant un si faible courant avec une si petite tension). Cependant, l'optocoupleur nous permet de régler le problème de niveau de tension en séparant la partie 3,3V de celle en 5V.



Nous utilisons ici l'optocoupleur exactement à l'inverse de la partie détection de tension côté PC, mais avec une petite subtilité. En effet, nous ne contrôlons pas la led en appliquant une tension pour l'allumer, mais préférons inverser les niveaux logiques. Par défaut, la led est donc connectée à la tension d'alimentation via une résistance et sa cathode est reliée à l'ESP8266. De ce fait en passant la sortie à l'état haut, la led s'éteint, le collecteur du phototransistor va à la masse et le relais s'active. Ceci nous permet d'avoir un comportement adapté pour le démarrage du montage ou sa réinitialisation : le relais ne s'active pas tout seul.

Notez que ce circuit pourra également être utilisé avec le montage Arduino, mais cela ne présente que peu d'intérêt. Comme la masse et l'alimentation du module relais sont connectées à la carte, il n'y a pas d'isolation et le fait que l'Arduino ne pilote pas le module relais directement ne change rien à l'affaire. Dans l'absolu, une isolation parfaite pourrait être souhaitée, mais ceci implique l'utilisation de deux blocs d'alimentation séparés (un pour la carte et un pour le module relais).



Les modules ou cartes ESP8266 se déclinent en de nombreuses versions et sont tous utilisables avec l'environnement Arduino, sous réserve d'installer un support complémentaire via le gestionnaire de cartes.

2. LA LOGIQUE DU CROQUIS

Quelle que soit la plateforme choisie, le croquis fonctionnera de la même manière étant donné que la bibliothèque **Ethernet** pour Arduino et **ESP8266WiFi** reposent peu ou prou sur le même principe de fonctionnement et que seuls les classes et types changent. Les croquis complets pour ce projet sont relativement volumineux et ne seront donc pas reproduits dans leur intégralité ici. Vous pourrez néanmoins les télécharger sur le dépôt GitHub du magazine (**https://github.com/Hackable-magazine**). Nous nous contenterons ici de voir les éléments clés.

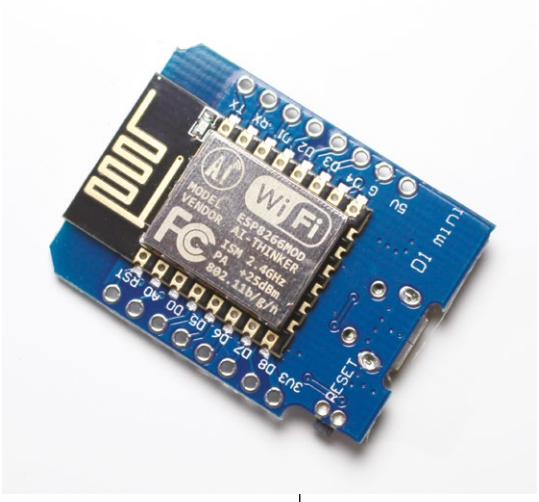
Notre croquis commence, comme toujours, par l'inclusion des bibliothèques utilisées et par la définition des macros facilitant la désignation des broches et valeurs utiles. Côté Arduino, nous avons donc :

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

// port IP à tester
#define TESTPORT 22
// broche de la led relais/erreur
#define RLEDPIN 7
// broche de la led online
#define OLEDPIN 6
// broche de test +5V
#define TESTPIN 5
// broche du relais
#define RELAIS A0
// temps de pression
#define RELDELAY 500
// taille max de commande
#define BUFSIZE 16
```

Et pour l'ESP8266 :

```
#include <ESP8266WiFi.h>

// port IP à tester
#define TESTPORT 22
// broche de la led relais/erreur
#define RLEDPIN D6
// broche de la led online
#define OLEDPIN D5
// broche de test +5V
#define TESTPIN D1
// broche du relais
#define RELAIS D7
// temps de pression
#define RELDELAY 500
// taille max de commande
#define BUFSIZE 16
```

TESTPORT est le port TCP/IP sur lequel le montage va tenter une connexion pour vérifier si la machine cible (le PC) est en fonction ou non. En effet, nous ne nous contenterons pas de simplement vérifier l'alimentation via l'optocoupleur, mais nous assurerons également que la machine répond. Ici le port 22 correspond au service SSH permettant l'accès distant en ligne de commandes. Sur une machine Windows, il faudra choisir le port correspondant au service auquel vous souhaitez accéder. Notez que le protocole utilisé n'a pas d'importance, tout



La version ESP8266 du montage a été construite ici avec ce module, un clone de WeMos D1 mini, lui-même compatible NodeMCU 1.0, acheté sur eBay pour moins de 4€ auprès d'un vendeur chinois appelé sale1213.

ce que nous voulons faire c'est obtenir une connexion et, si cela fonctionne, nous la terminerons aussitôt. Ceci est bien plus simple que de tenter un **ping** (ICMP), car ne nécessitant pas de bibliothèque supplémentaire.

RELDELAY correspond au délai de fermeture du contact du relais. 500 millisecondes devraient être largement suffisantes pour simuler une pression humaine sur le bouton. Et enfin, **BUFSIZE** nous permettra de fixer une limite à la taille des lignes reçues sur notre serveur. Le jeu de commandes envisageables est relativement réduit et il n'y a aucun intérêt à traiter des lignes complexes de 255 caractères tout en consommant énormément de mémoire. Tout ce qui dépassera 16 octets sera ignoré purement et simplement. Les autres macros déclarées parlent d'elles-mêmes et désignent les broches utilisées.

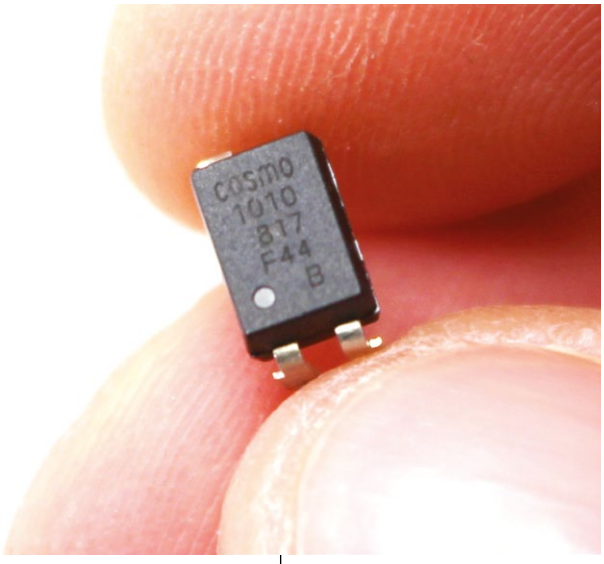
Viennent ensuite les déclarations des variables globales avec, pour l'Arduino :

```
// adresse MAC
byte mac[] = { 0x90, 0xA2, 0xDA, 0x0F, 0x0F, 0xA0};
byte testip[] = { 192, 168, 10, 166 };

// hôte distant (cible)
EthernetClient testclient;
// hôte distant (client telnet)
EthernetClient client;

// serveur telnet
EthernetServer server(23);
```

Et pour l'ESP8266 :

```
const char* ssid = "nomAPwifi";
const char* password = "mot2passe";
byte testip[] = { 192, 168, 10, 166 };

// hôte distant (cible)
WiFiClient testclient;
// hôte distant (client telnet)
WiFiClient client;

// serveur telnet
WiFiServer server(23);
```

Comme vous pouvez le voir, ces deux morceaux de croquis sont très similaires et ne changent que par le type d'objets déclarés. Dans le cas de l'interface Ethernet, nous devons en revanche préciser l'adresse matérielle, généralement marquée sur une étiquette sur le shield. Côté ESP8266, nous aurons besoin du nom du point d'accès wifi auquel se connecter et du mot de passe. Ces informations seront utilisées dans la fonction **setup()** pour établir la connexion au réseau.



Les optocoupleurs utilisés pour ce projet n'ont pas été achetés, mais dessoudés du circuit imprimé d'un onduleur hors d'usage (batterie usée). Il s'agit de composants Cosmo K1010, mais n'importe quel modèle courant fera l'affaire.

Côté objets déclarés, dans les deux cas nous avons **testclient** qui représentera la connexion à la machine cible pour tester son état. **client** sera le point de connexion d'un client à notre montage, en Telnet, et enfin, **server** représente notre montage lui-même. Différentes méthodes peuvent être utilisées avec ces objets pour gérer les connexions et lire/écrire des données.

À ce stade, le décor est planté et nous pouvons nous pencher sur des fonctions utiles que nous utiliserons dans la boucle principale **loop()** ou dans **setup()**. L'une d'entre elles est destinée à faciliter l'affichage de l'adresse que nous allons obtenir soit par connexion filaire, soit en Wifi. C'est l'adresse à laquelle notre montage sera joignable en Telnet :

```
// Affichage IP
void printIPAddress(){
  Serial.print("Mon adresse est ");
  for (byte i = 0; i < 4; i++) {
    Serial.print(Ethernet.localIP()[i], DEC);
    Serial.print(".");
  }
  Serial.println();
}
```

Là encore, il faut faire une distinction entre Arduino et ESP8266, mais ceci se résume à une simple fonction, **Ethernet.localIP()** dans un cas et **WiFi.localIP()** dans l'autre. La méthode **localIP()** retourne l'adresse en question et nous créons une boucle **for** pour parcourir et envoyer au moniteur série les quatre octets de l'adresse, joliment présentés.

Une autre fonction utile est celle permettant de tester la disponibilité de la machine cible. Celle-ci ne fait qu'utiliser des méthodes sur des objets existants et est donc commune aux deux plateformes :

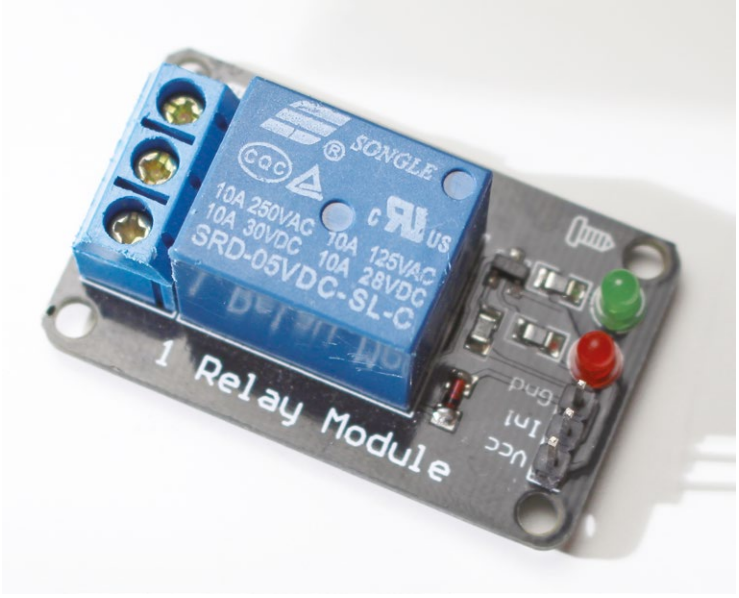
```
// Test de connexion
bool isOnline(int essais, int msec) {
  int i = 0;
  client.println("Test du +5V sur la machine");
  // si on ne détecte pas de 5V on ne va pas plus loin
  if(digitalRead(TESTPIN))
    return 0;
  client.print("Test de la machine sur le port ");
  client.print(TESTPORT);
  // on boucle autant d'essais demandés
  while(i < essais) {
    client.print(".");
    if(testclient.connect(testip, TESTPORT)==1) {
      // le serveur répond
      Serial.println("Connexion ok");
```

```
        testclient.stop();
        client.println();
        // on s'arrête là
        return 1;
      } else {
        // essai suivant après pause
        Serial.print("Erreur ");
        Serial.println(i);
        delay(msec);
        i++;
      }
    }
    client.println();
    // On arrive ici uniquement si le nombre
    // d'essais est dépassé
    return 0;
}
```

Notre fonction retournera `0` si la machine cible n'est pas allumée et accessible, et `1` dans le cas contraire. Pour faire ce test, nous utilisons deux paramètres passés en argument : le nombre d'essais à faire et le délai entre ces essais en millisecondes. Notre fonction est générique et sera utilisée pour tester la cible avant l'utilisation du relais/bouton, mais également après, pour éventuellement nous assurer que la machine ait bien démarré. En utilisant un nombre d'essais important, espacés d'une durée conséquente, nous pouvons donc laisser le temps au PC de démarrer et devenir accessible.

L'objectif ici est de faire au plus court. Inutile de tester la connexion à la machine si celle-ci n'a pas de courant. Nous commençons donc par vérifier l'état de la broche connectée en entrée à l'optocoupleur et retournons immédiatement `0` s'il n'y a pas d'alimentation (broche à l'état haut). Dans le cas contraire, nous pouvons tenter la connexion via une boucle décomptant le nombre d'essais demandés. Dès que nous avons une réponse, nous retournons `1`, la machine est disponible. Si le nombre d'essais est épuisé, par dépit, nous retournons `0`, la machine est bien en marche, mais la connexion ne fonctionne pas.

Il est étonnamment parfois moins cher d'acheter des modules relais déjà équipés de leds, transistor et diode sur les sites d'enchères en ligne que les composants à l'unité seuls auprès d'un détaillant en électronique. Et en plus, ça permet de gagner du temps...



La programmation c'est comme la cuisine, on prépare les ingrédients et les ustensiles, puis on peut se faire plaisir. Tout est maintenant prêt et nous pouvons entrer dans le vif du sujet en configurant notre montage et en initiant la connexion, via la fonction `setup()`. Dans le cas d'une carte Arduino avec le shield Ethernet, ceci est relativement rapide :

```
void setup() {
  digitalWrite(RELAIS, HIGH);
  pinMode(RELAIS, OUTPUT);

  pinMode(RLEDPIN, OUTPUT);
  pinMode(OLEDPIN, OUTPUT);
  pinMode(TESTPIN, INPUT);

  Serial.begin(115200);
  Serial.println("Demande DHCP...");
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Erreur de configuration via DHCP");
    while (1) {
      // erreur DHCP
      digitalWrite(RLEDPIN, HIGH);
      delay(250);
      digitalWrite(RLEDPIN, LOW);
      delay(250);
    }
  }
  printIPAddress();
  server.begin();
  digitalWrite(OLEDPIN, HIGH);
}
```

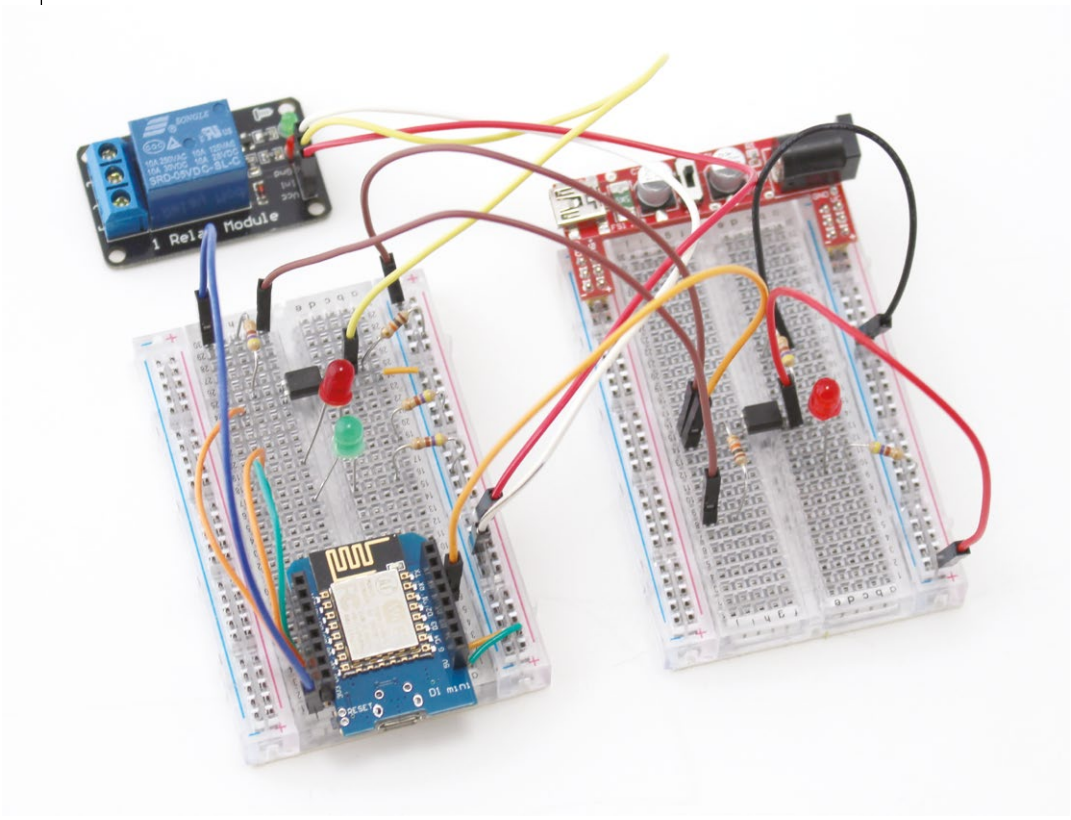
Tout tourne ici autour de **`Ethernet.begin()`** prenant en argument l'adresse MAC du shield. Si nous obtenons une adresse en retour et que la résolution DHCP a fonctionné, cette méthode retourne une valeur différente de **`0`** et nous affichons l'adresse obtenue avec **`printIPAddress()`** avant d'activer le serveur sur le port 23 (Telnet) et d'allumer la led verte (on est « en ligne »). Dans le cas contraire, un problème est survenu et nous n'allons pas plus loin, nous partons dans une boucle infinie faisant clignoter rapidement la led rouge.

Pour l'ESP8266, les choses sont un peu plus complexes, car la tentative de connexion n'est pas bloquante. Nous devons donc demander la connexion puis attendre en testant le résultat dans une boucle :

```
void setup() {
  uint8_t i = 0;
  pinMode(RELAIS, OUTPUT);
  digitalWrite(RELAIS, HIGH);

  pinMode(RLEDPIN, OUTPUT);
  pinMode(OLEDPIN, OUTPUT);
  pinMode(TESTPIN, INPUT);

  Serial.begin(115200);

  WiFi.begin(ssid, password);
  Serial.print("\nConnexion a ");
  Serial.println(ssid);
  while (WiFi.status() != WL_CONNECTED && i++ < 20)
    delay(500);
  if(i > 20){
```

```
    Serial.print("Erreur de connexion sur ");
    Serial.println(ssid);
    while (1) {
      // erreur DHCP
      digitalWrite(RLEDPIN, HIGH);
      delay(125);
      digitalWrite(RLEDPIN, LOW);
      delay(125);
    }
  }
  printIPAddress();
  server.begin();
  server.setNoDelay(true);
  digitalWrite(OLEDPIN, HIGH);
}
```

La connexion est établie avec **WiFi.begin()** prenant en argument le nom du point d'accès et le mot de passe associé. Nous devons ensuite vérifier le résultat de **WiFi.status()** jusqu'à obtenir **WL_CONNECTED** indiquant une connexion ou que le nombre d'essais soit dépassé (20 ici). Les itérations dans la boucle **while** terminées, la valeur de **i** nous renseigne sur l'état de l'opération et nous faisons clignoter la led rouge en cas de problème. Comme pour la carte Arduino, si tout se passe bien, le serveur est activé et la led verte de même.



La phase d'expérimentation se fait, comme toujours, sur platine à essais à grand recours de câbles. On voit ici à gauche le module de contrôle avec l'ESP8266, à droite la partie simulant le PC avec sa propre alimentation et à l'arrière le module relais.

3. LA BOUCLE PRINCIPALE

La boule principale, comme toutes ses consœurs, fera la majeure partie du travail. Elle consiste en une succession d'états conditionnels :

- on récupère la connexion d'un client à notre serveur Telnet dans l'objet **client**,
- on s'assure que nous avons des données à lire,
- on récupère ces données si elles ont une taille acceptable,
- on analyse le contenu et extrait une chaîne utilisable faisant office de commande,
- puis on réagit en fonction de la commande obtenue, si celle-ci est connue et valide.

Cette fonction **loop()** est identique pour la carte Arduino et pour l'ESP8266, en dehors de deux éléments : l'initialisation de la communication avec le client et le renouvellement DHCP. En effet, ce second point est géré directement en coulisse par ESP8266, mais doit être traité manuellement pour le shield Ethernet sur Arduino. En temps normal, un serveur (Telnet ou autre) dispose d'une adresse IP fixe, mais j'ai préféré ici garder une certaine souplesse et cela implique quelques lignes de code en plus.

En début de **loop()** sur Arduino, nous déclarons notre tampon destiné à recevoir les données et récupérons la connexion si un client est présent :

```
void loop() {
  // stockage des données reçues
  char clientline[BUFSIZE];

  // récupération de la connexion d'un client
  client = server.available();
```

Ceci est sensiblement différent avec l'ESP8266 :

```
  uint8_t clientline[BUFSIZE];

  if (server.hasClient()){
    if (!client || !client.connected()){
      if(client) client.stop();
      client = server.available();
      Serial.println("Connexion d'un client");
    }
  }
```

Tout d'abord, nous avons la déclaration du tampon, qui doit être un tableau de type **uint8_t**, et non de **char**, pour ne pas provoquer une erreur à la compilation. Ensuite, la gestion des connexions de l'ESP8266 ne permet pas de simplement utiliser **server.available()** pour obtenir un objet valide, mais nous devons tester la présence d'une connexion avec **hasClient()** sur l'objet **serveur** (nous-mêmes). Ceci nous permet de gérer le cas d'une connexion terminée, que nous pouvons alors « nettoyer » avec **client.stop()**.

Le reste de la fonction **loop()** consistera alors à nous assurer de la connexion, puis à obtenir les données sous la forme d'une chaîne :

```
// on a un client ?
if (client && client.connected()) {
  // oui. Combien d'octets à lire ?
  int toread=client.available();
  if(toread > 0) {
    // plus d'octets que la taille dispo ?
    if(toread>=BUFSIZE) {
      // trop long !
      while(client.available()) {
        // on lit sans rien garder
        client.read();
      }
    } else {
      // effacement du tampon
      memset(clientline, 0, sizeof(clientline));
      // lecture des données
      client.read(clientline, toread);
      // conversion en chaîne
      String commande = String(clientline);
      // Suppression des blancs et CRLF en fin
      commande.trim();
      // Affichage de l'ordre reçu
      Serial.print("Commande lue: [");
      Serial.print(commande);
      Serial.println("]");

      // ordre vide après suppression CRLS ou espace ?
      if(commande.length()==0) {
      } else [...]
```

`client.available()` nous retourne le nombre d'octets à lire, envoyés par le client. Nous utilisons cette valeur pour déterminer si nous pouvons ou non les gérer. Si la valeur dépasse la taille de la zone mémoire dédiée (notre tableau, en gardant une place pour le marqueur de fin de chaîne `\0`), la supposée commande est déjà inacceptable. Dans ce cas, nous « dépilons » les données avec `client.read()` sans rien en faire.

Dans le cas contraire, nous devons tout d'abord transférer les données dans le tampon `clientline` avec `client.read()` en lui passant en argument un pointeur vers où mettre ces données, et leur taille. Dès lors `clientline` contient une chaîne de caractères, mais non un objet `String` Arduino. Nous convertissons alors les informations avec `String(clientline)` pour obtenir `commande`, puis nous utilisons la méthode `trim()` pour nous débarrasser d'éventuels blancs et caractères non imprimables en fin de chaîne. Notez qu'il serait possible de traiter la chaîne sans faire usage du type `String`, mais ceci simplifie grandement les choses.

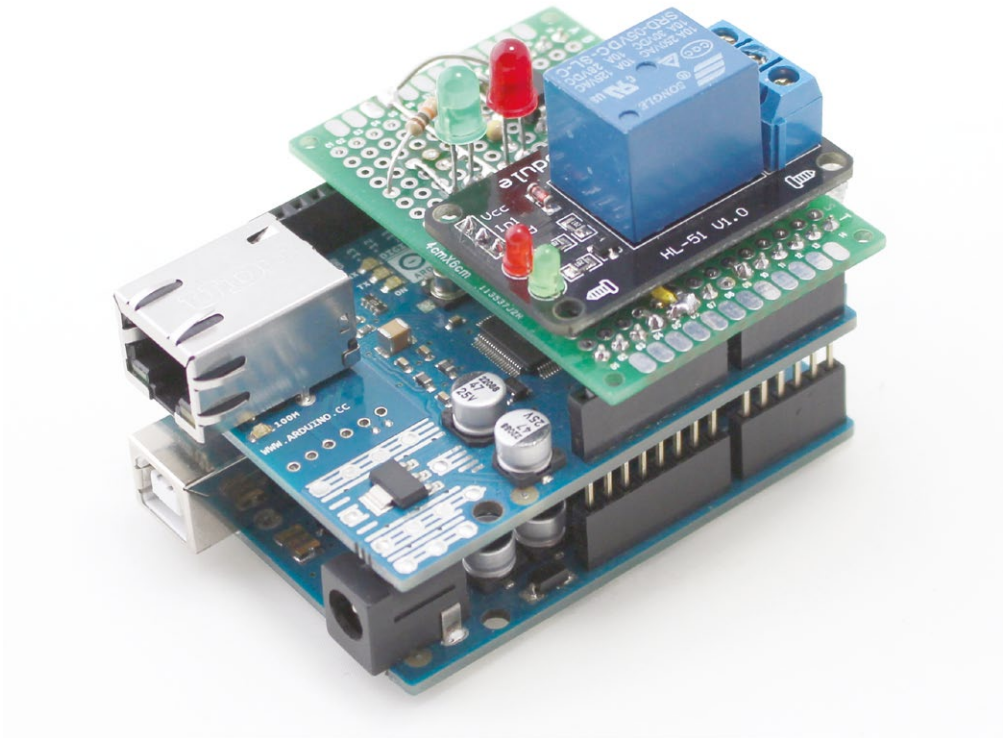
Il ne nous reste plus alors que la commande elle-même... s'il reste encore quelque chose. Point que nous testons avec la valeur retournée par la méthode `length()`. Si cette valeur est différente de `0` (chaîne vide), alors nous avons quelque chose à faire.

La suite est relativement simple puisqu'il s'agit d'enchaîner les `else if()` en utilisant la méthode `equals()` de l'objet `commande` pour tester des équivalences. C'est là que vous pouvez implémenter n'importe quel ordre devant être accepté. Exemple pour démarrer la machine :

```
} else if(commande.equals("boot")) {
  // test de connexion
  if(isOnline(1,1)) {
    client.println("\e[1;31mMachine deja en fonction!\e[0m");
  } else {
    // l'hôte ne répond pas, il doit être éteint
    client.println("\e[1;32mMise en route de la machine.\e[0m");
    // allumage
    digitalWrite(RELAIS, LOW);
    digitalWrite(RLEDPIN, HIGH);
    delay(RELDELAY);
    digitalWrite(RELAIS, HIGH);
    digitalWrite(RLEDPIN, LOW);
  }
} else [...]
```

Chaque condition **if** reposant sur **equals()** est la mise en oeuvre d'une commande donnée. Ici, si la commande est **"boot"**, nous commençons par vérifier si la machine n'est pas déjà en fonction, si tel n'est pas le cas, nous activons le relais ainsi que la led rouge durant un bref moment. Ceci aura pour effet de simuler la pression sur le bouton de démarrage du PC.

NOTE : Les caractères étranges comme **\e[1;31m** dans les messages sont des séquences d'échappement ANSI permettant d'ajouter un peu de couleur. **\e[1;31m** démarre l'utilisation du rouge en premier plan, **\e[1;32m** fait de même pour le vert, et **e[0m** revient à



Une fois le montage correctement testé, il ne reste plus qu'à passer à la réalisation sur plaque perforée qui prendra la forme d'un shield « maison » placé directement sur le shield Ethernet.

un affichage « normal ». Vous pourrez trouver facilement les correspondances couleur/code via une petite recherche sur le Web.

Une fois toutes les commandes implémentées, il reste à traiter le cas d'une commande inconnue et, enfin, finir par afficher une invite pour que le client connecté puisse taper la prochaine commande :

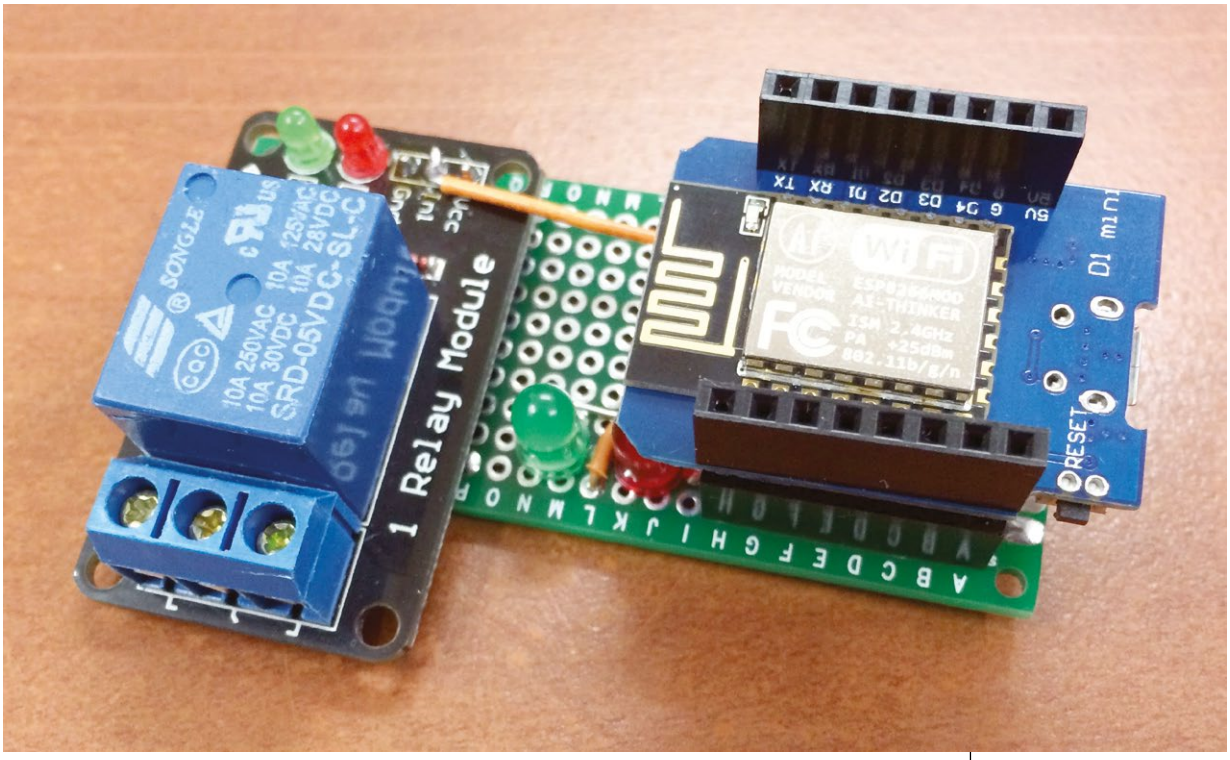
```
        } else {
          // commande inconnue
          client.println("Hein?");
        }
      }
      // invite de commandes
      client.print("mon> ");
```

Pour l'ESP8266, le croquis s'arrêtera là, mais pour l'Arduino avec son shield Ethernet il faudra encore gérer le renouvellement DHCP avec ce petit morceau de code (placé en dehors de toutes conditions précédemment définies, bien sûr) :

```
// gestion du bail DHCP
switch (Ethernet.maintain()) {
  case DHCP_CHECK_RENEW_FAIL:
    Serial.println("Error lors du renouvellement!");
    digitalWrite(OLEDPIN, LOW);
    break;
  case DHCP_CHECK_RENEW_OK:
    Serial.println("Renouvellement ok");
    printIPAddress();
    digitalWrite(OLEDPIN, HIGH);
    break;
  case DHCP_CHECK_REBIND_FAIL:
    Serial.println("Error lors de la demande");
    digitalWrite(OLEDPIN, LOW);
    break;
  case DHCP_CHECK_REBIND_OK:
    Serial.println("Demande ok");
    printIPAddress();
    digitalWrite(OLEDPIN, HIGH);
    break;
  default:
    // rien
    break;
}
```

Il faut reconnaître que nous avons là un croquis (et un article) d'une taille relativement conséquente et je n'ai pas listé toutes les commandes supportées. C'est là ce avec quoi il est commun de finir, dès lors qu'on travaille sur une connectivité réseau et un serveur s'utilisant de façon interactive.

Bien entendu, les croquis pour Arduino et pour ESP8266, disponibles au téléchargement sur GitHub, sont complets et pourront être directement utilisés/modifiés. Les commandes définies à cette date pour mes besoins personnels sont : **bye** ou **quit** pour terminer la connexion, **boot** pour utiliser le bouton, **bootw** pour faire de même, mais ensuite vérifier le démarrage



Voici la version ESP8266 du projet finalisé qui a pris place directement dans le boîtier PC. Le relais est alors relié au bouton de démarrage et la détection d'alimentation se fait via un connecteur placé sous le circuit. Reste ensuite à utiliser un bloc d'alimentation type « chargeur de smartphone » et à faire passer le câble à l'extérieur du PC pour le brancher sur l'onduleur à proximité...

avec `isOnline(5,1000)`, `sta` ou `status` pour afficher l'état de la machine et `lpress` pour une pression longue de 5 secondes sur le bouton en cas de plantage (celle-ci n'est pas dans les croquis téléchargeables, car trop dangereuse sans ajouter une confirmation).

4. VARIATIONS POSSIBLES

Sur la base de ce projet, il est possible d'extrapoler pour pouvoir contrôler bien d'autres choses qu'un PC. En effet, tout repose ici sur l'utilisation du réseau pour ne finalement que contrôler un relais. Celui-ci est connecté au bouton de mise sous tension, mais rien ne vous empêche de contrôler directement une prise d'alimentation du courant domestique. Bien entendu, dans ce cas il ne s'agira pas simplement d'une brève impulsion, mais d'un fonctionnement en va-et-vient permettant d'alimenter ou non un équipement (lampe, imprimante, disque externe, machine à café, etc.). Attention cependant, si vous souhaitez contrôler ainsi le courant domestique, une isolation galvanique dans les règles de l'art est indispensable et cela implique également une mise en boîtier du montage.

Ce projet est initialement destiné à une seule machine de type PC, mais il reste, aussi bien sur Arduino que sur ESP8266, bon nombre de broches à utiliser. Il est donc parfaitement envisageable d'étendre les fonctionnalités en contrôlant plusieurs PC ou périphériques en multipliant les optocoupleurs et les relais. Il pourrait également être intéressant d'ajouter d'autres modules comme une horloge RTC afin de garder une trace des activations dans un journal ou encore de connecter une sonde de température pour informer de l'état de santé de la machine.

Enfin, côté logiciel, on peut également envisager la mise en place d'une authentification par mot de passe ou un système de confirmation pour les ordres donnés. Les évolutions possibles ne manquent pas et, en ce qui me concerne, je pense que le projet ne s'arrêtera pas à cet article, et je l'espère, pour vous non plus... **DB**

RECYCLEZ UN VIEIL ÉCRAN VGA ET UTILISEZ-LE AVEC VOTRE ARDUINO

Denis Bodor

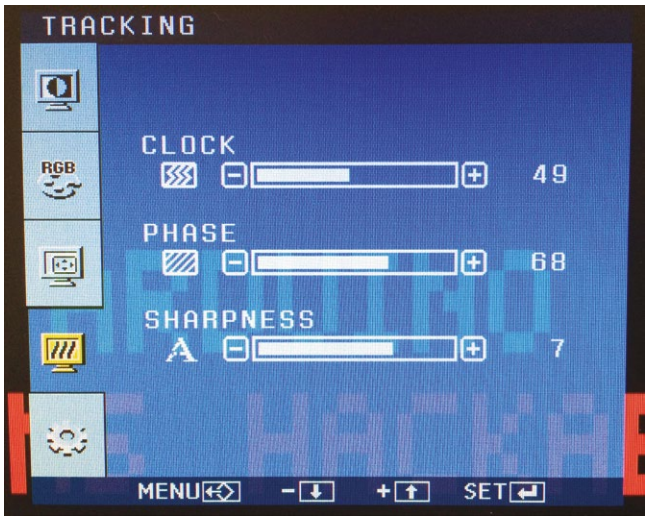
Aujourd'hui tout est numérique ou presque, et ceci est valable pour le dernier bastion analogique qui restait sur nos PC : l'affichage. Même si les connecteurs VGA sont encore tantôt disponibles sur les machines neuves, la connectique la plus utilisée est définitivement HDMI et DVI. Il y a donc naturellement, dans la nature, une ribambelle d'écrans VGA prenant la poussière dans leur coin. Ne pourrait-on pas les « sauver » d'une disparition certaine grâce à une carte Arduino ?

C'est ainsi, la technologie évolue et c'est une très bonne chose (sauf quand des zozos californiens décident qu'un connecteur audio est inutile juste pour vendre des accessoires). Vouloir utiliser un écran VGA sur un PC moderne aujourd'hui relève presque du masochisme. En comparaison de la qualité d'image obtenue avec une sortie numérique comme le HDMI, l'image d'un moniteur VGA semble... « sale », démodée, presque dérangeante.

Il reste cependant quelques usages pour un moniteur VGA :

- comme écran secondaire en complément d'un affichage principal en DVI ou HDMI ;
- avec un vieux PC ou portable par nostalgie ;
- via un convertisseur péritel/composite pour une ancienne console (l'image ne sera pas pire que l'originale) ;
- via un adaptateur économique HDMI/VGA avec une Raspberry Pi (on en trouve pour moins de 6€ sur eBay) ;
- mais aussi, via un montage simple, avec une carte Arduino.

Lorsque je dis « montage simple », il faut plutôt comprendre « ultra-simpliste » puisqu'en l'occurrence il ne s'agit que d'utiliser quatre résistances en plus de la connectique. Ceci est suffisant pour générer un signal susceptible d'être « compris » par un écran VGA, mais les possibilités offertes sont relativement faibles.



La plupart des moniteurs VGA LCD permettent un réglage du « pixel clock » et de la phase du signal car, contrairement à un écran à tube cathodique, les pixels affichés sont physiquement définis. En cas de mauvaise interprétation du signal, un effet de moirage et/ou des artefacts peuvent apparaître, en particulier lors de l'affichage de texte. La détection automatique de la fréquence et la phase du signal sont généralement provoquées par un bouton « auto » à la disposition de l'utilisateur, après qu'il ait affiché un damier de pixels blancs et noirs.

1. UN SIGNAL VGA ANALOGIQUE AVEC ARDUINO ?

La transmission d'informations vidéos vers un moniteur analogique repose sur la génération synchronisée de plusieurs signaux, 5 en totalité :

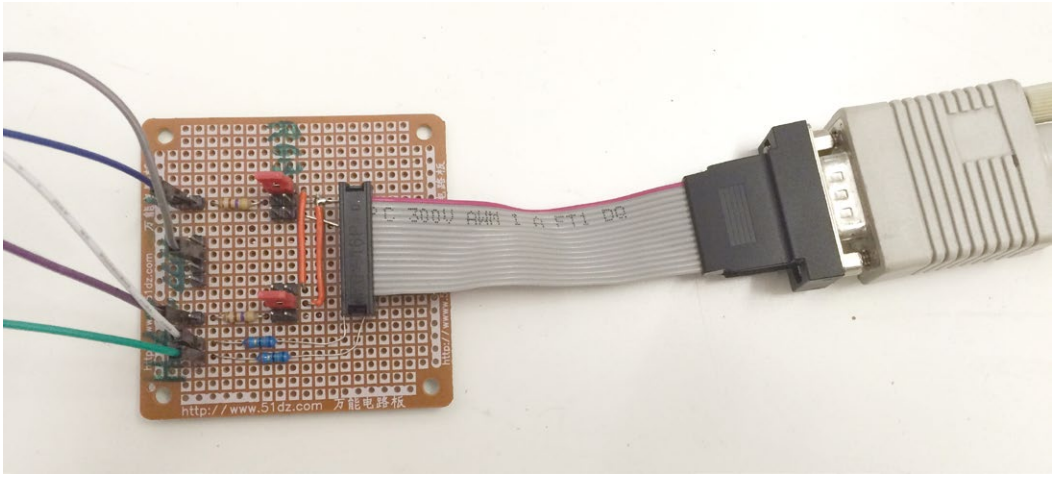
- *HSYNC*, le signal de synchronisation horizontale. Il faut imaginer le dessin sur le moniteur comme un rayon balayant l'écran (dans un tube cathodique). Celui-ci se déplace, de votre point de vue de gauche à droite et de haut en bas, ligne par ligne. Le signal HSYNC est une impulsion marquant le début et la fin d'une ligne ou, en d'autres termes, il borne l'affichage horizontal.
- *VSYNC* est un signal similaire, mais déterminant le début et la fin d'une image complète, d'un écran, d'un ensemble de lignes. HSYNC et VSYNC sont typiquement des signaux en 5V (parfois 3,3V).
- *R*, *G* et *B* donnent l'intensité des trois composantes de couleur pour un pixel : rouge, vert et bleu. Il s'agit d'une tension entre 0V et 0,7V définissant la quantité de chaque couleur et permettant ainsi d'obtenir n'importe quelle nuance. Ces tensions doivent être synchronisées avec les autres signaux de façon à déterminer avec précision la couleur de chaque point. Comme ces signaux ne sont pas faits de niveaux de tension prédéterminés (0 ou 0,7V), on parle de signaux analogiques et, par extension, de moniteurs analogiques, par opposition au DVI qui est un protocole purement numérique (ce sont des données binaires qui sont transmises à l'écran et donc avec des niveaux de tension discrets).

Et là, vous devez vous demander naturellement « mais qu'est-ce qui détermine le nombre de points par ligne ? » ou plutôt, comment est définie la résolution horizontale ? La réponse est simple, c'est le moniteur qui l'estime en fonction des modes qu'il connaît et de la fréquence de rafraîchissement vertical (VSYNC). La fréquence à laquelle les pixels horizontaux sont déterminés, ou plus exactement la largeur de ces pixels, repose sur la notion de *pixel clock* qui, sur certains moniteurs LCD, peut être ajustée manuellement dans les réglages de l'écran ou détecté automatiquement. Ceci permet d'éviter un effet de moirage en jouant sur la fréquence du *pixel clock* et la phase du signal.

Vous l'aurez compris, la création d'un signal vidéo analogique interprétable par un écran VGA repose donc entièrement sur une question de fréquences et de temporisations qui, si elles ne sont pas exactes, empêcheront le signal d'être interprété correctement. Mais de plus, tout doit être parfaitement synchronisé puisqu'il faut envoyer les valeurs des composantes de couleur au bon moment tout en gardant à l'esprit que c'est là quelque chose qui doit être fait constamment. Il ne s'agit pas d'envoyer des données d'affichage comme avec un écran LCD alphanumérique (HD44780) ou un module TFT couleur en SPI (voir *Hackable n°4*), il faut en permanence rafraîchir l'écran. Dès qu'on arrête, plus rien ne s'affiche.

Notez ici que je fais délibérément l'impasse sur un point important, mais qui n'entre pas en ligne de compte pour la réalisation du projet puisque nous n'allons pas écrire le code qui génère le signal, mais utiliser une bibliothèque qui le fera pour nous. Ce point se résume au fait que pour afficher une image de 640×480 pixels sur un écran, au regard des signaux et des impulsions, il faut envoyer une image sensiblement plus grande permettant de prendre en compte le balayage de l'écran par le faisceau d'électrons (dans un tube cathodique). Le temps que l'écran ramène le faisceau en haut à gauche de l'écran une fois arrivé en bas à droite, par exemple, il est nécessaire de continuer d'envoyer des informations, même si elles ne sont pas affichées. Il en va de même lorsque le faisceau revient à gauche après être arrivé en fin de ligne.

Cependant, même en dehors de ce point et en regardant simplement les définitions du standard VESA, on se rend rapidement compte qu'on risque de rencontrer un problème avec une carte Arduino. Le mode 640×480 par exemple, utilise une fréquence HSYNC de 37,5 Hkz et VSYNC de 75 Hz (il existe aussi un mode 85 Hz). Ceci peut sembler peu, mais on arrive à un *pixel clock* de 31,5 Mhz, ce qui signifie que la

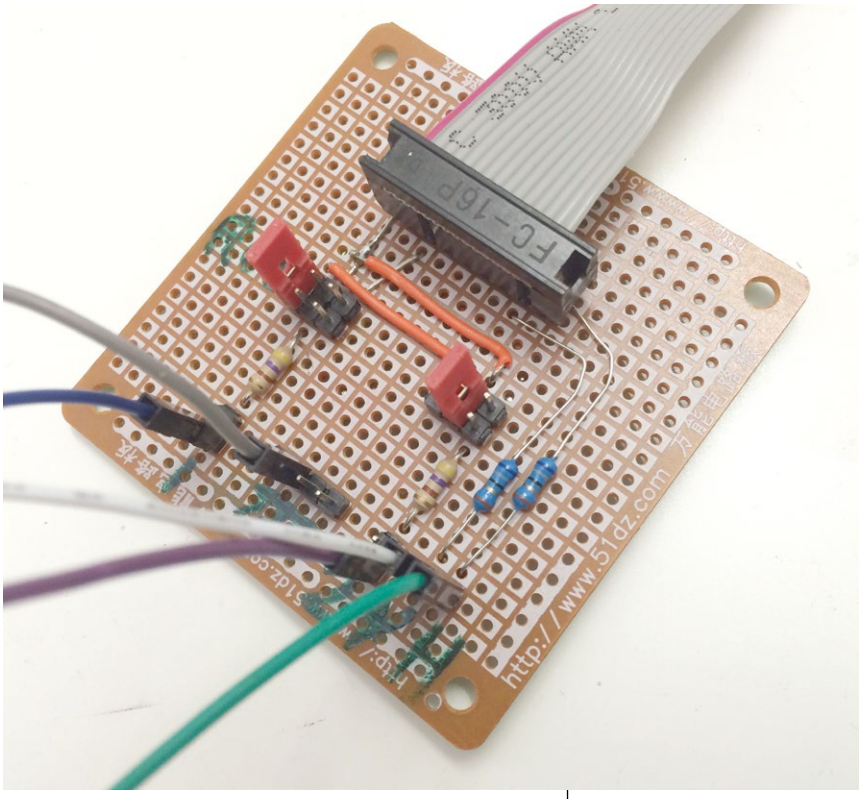


Le montage très simple à la base, une fois assemblé sur plaque pastillée, peut être un peu plus travaillé que le schéma de départ. Ici des cavaliers permettent de choisir très facilement deux des trois composantes de couleur, même si rouge/vert est le duo le plus intéressant.

carte Arduino devrait pouvoir définir les valeurs de rouge, vert et bleu à cette fréquence... alors que le microcontrôleur n'est cadencé qu'à 16 Mhz. Ceci sans compter le fait qu'une image de 640×480 pixels, même monochrome est constituée de 370200 pixels et donc de 38400 octets, ce qui est plus que les 32768 octets de flash disponibles et 18 fois plus que le volume de SRAM présente. Le mode 640×480 est donc clairement hors de portée... sauf si on décide de tricher !

Gérer les fréquences HSYNC et VSYNC est déjà difficile, mais tout à fait possible en utilisant les *timers* intégrés au microcontrôleur Atmel AVR d'une carte Arduino UNO par exemple. Mais il n'est pas possible, même en oubliant l'aspect analogique et en décidant arbitrairement d'utiliser les couleurs sans nuance, en tout ou rien, de jouer sur les broches de la carte à une fréquence d'environ deux fois celle sur processeur. Ce qu'on peut faire, en revanche, c'est décider de ne pas changer de couleurs à chaque pixel, mais à chaque groupe de pixels. Le moniteur pensera avoir affaire à une image de 640×480, mais nous ne considèrerons, d'un point de vue des données, qu'une résolution bien inférieure.

Cette astuce, et beaucoup de travail et d'expérimentations, ont permis à *Nick Gammon* d'obtenir un affichage VGA à partir d'une carte Arduino UNO et de proposer un code fonctionnel en 2012 sur son forum (**http://www.gammon.com.au/forum/?id=11608**). Ses résultats et son code ont ensuite été réutilisés et améliorés par *Sandro Maffiodo* pour créer une bibliothèque Arduino en 2015 (**https://github.com/smaffer/vgax**). C'est cette bibliothèque, VGAX, que nous allons utiliser ici.



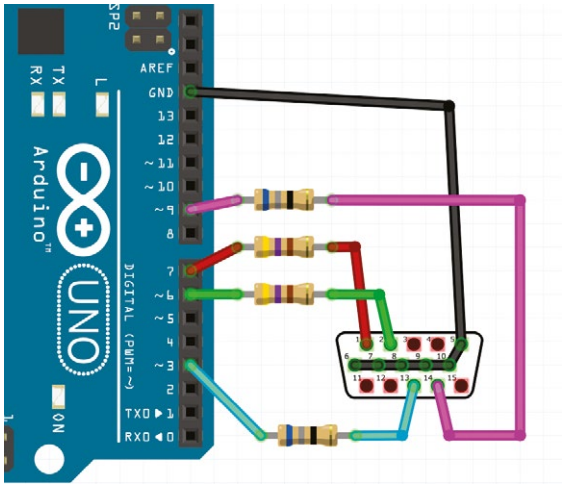
Deux résistances de 470 ohms et deux de 68 ohms, il n'en faut pas plus pour brancher une carte Arduino UNO à un écran VGA et obtenir une image. Seul problème cependant, le code de génération du signal vidéo accapare totalement le microcontrôleur et il est très difficile d'envisager une utilisation vraiment avancée avec le peu de mémoire qu'il reste.

Cette bibliothèque vous permettra, à l'aide de 4 résistances (deux de 68 ohms) et deux de 480 ohms, d'afficher des pixels, une image ou du texte avec une résolution de 120×60 pixels (mais la résolution pour le moniteur sera 640×480). Vous ne pourrez utiliser que 4 couleurs et ceci comprend le noir (pas de couleur), 2 couleurs au choix parmi rouge, vert et bleu, et une quatrième couleur, résultat de la combinaison des deux autres en synthèse additive :

- rouge + vert = jaune,
- rouge + bleu = magenta,
- bleu + vert = cyan.

Comme vous pouvez le constater, nous sommes loin d'un affichage digne d'un ordinateur familial des années 80. Un honorable et modeste Commodore 64 est en effet capable d'afficher des graphismes de 320×200 en 16 couleurs, mais il réalise cela via un circuit intégré dédié (le VIC-II) comme la totalité des machines de l'époque. Ceci n'est pas très différent de l'architecture actuelle des PC et Mac utilisant un processeur graphique

(GPU) intégré au processeur ou à la carte mère ou sous forme de carte additionnelle. Une solution similaire existe pour Arduino sous la forme d'un shield (Gameduino), mais ceci suppose un budget bien supérieur au coût de quelques résistances et connecteurs de récupération.



La partie la plus difficile dans ce montage consiste à trouver un connecteur VGA et à repérer les signaux. Bon nombre de moniteurs modernes sont toujours livrés avec des câbles VGA inutiles qu'il suffira de « torturer ». Notez qu'il est de bon ton de garder les connexions les plus courtes possible pour limiter les interférences et garantir une stabilité des signaux.

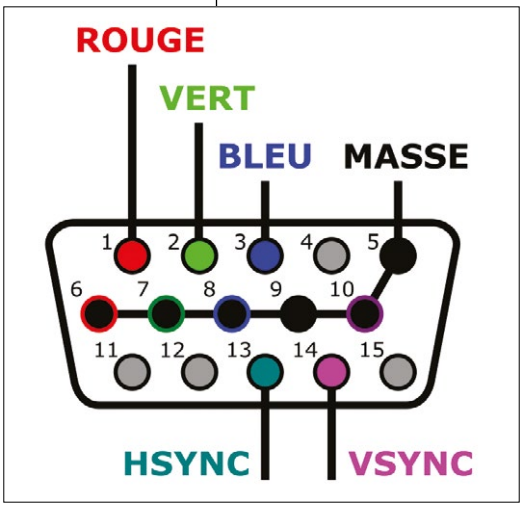
2. LE MONTAGE

Au niveau du câblage et de l'interface, le montage est simplissime puisque cela consiste tout bonnement à relier la carte Arduino UNO aux broches du connecteur VGA via des résistances. Les signaux HSYNC et VSYNC sont branchés respectivement aux broches 3 et 9 de la carte. En ce qui concerne les couleurs, ceci se fera via des résistances de 470 ohms.

Les moniteurs possèdent une résistance interne de 75 ohms entre les broches 1, 2 et 3 et la masse et le fait d'utiliser une valeur de 470 ohms pour les signaux de couleur prend donc la forme d'un montage en diviseur de tension. Lorsqu'une sortie de la carte Arduino est à l'état haut, les 5V sont donc divisés proportionnellement au ratio entre chacune des valeurs de résistance et la somme des deux : 75/(75+470)x5V = 0,69V, presque les 0,7V indiquant une intensité maximum.

Il n'est pas possible de faire varier cette tension comme le ferait une carte graphique de manière à jouer sur l'intensité des couleurs. Ceci ne vient pas du fait qu'une carte Arduino ne pourrait pas matériellement le faire avec, encore une fois, des diviseurs de tension, même si la carte ne dispose pas de vraies sorties analogiques (via un DAC), mais plutôt d'un problème d'architecture interne et de performances. *Sandro Maffiodo* utilise les broches 7 et 6 pour définir les couleurs utilisées. Celles-ci sont gérées par le microcontrôleur comme deux broches d'un ensemble de huit (le PORTD). C'est une astuce de programmation permettant de garantir, dans une certaine mesure, les délais précis d'activation de ces broches, mais cela vous interdit aussi d'utiliser n'importe quelle autre broche du PORTD (broches 0 à 7 d'une UNO) pour un autre usage.

Voici le brochage standard d'un connecteur VGA avec les signaux que nous allons utiliser. Les broches au centre du connecteur sont les masses que nous connectons ensemble et il ne reste que les trois couleurs et les signaux de synchronisation à gérer.



En termes de réalisations pratiques du circuit, on pourra bien sûr tout simplement vampiriser un câble VGA standard, retrouver les fils correspondant aux broches et souder directement les résistances. Il sera préférable cependant de prévoir quelque chose de plus robuste, en utilisant par exemple une plaque pastillée pour agencer les résistances et éventuellement prévoir une solution mécanique permettant la sélection des composantes de couleur avec des cavaliers. En ce qui me concerne, j'ai préféré garder

le câble VGA intact et utiliser un connecteur récupéré d'une vieille carte graphique.

Dans tous les cas, il est de bon ton de garder les câbles non blindés les plus courts possible afin de garantir une bonne intégrité des signaux et, par la même occasion, réduire les émissions parasites.

3. UTILISER LA BIBLIOTHÈQUE VGAX

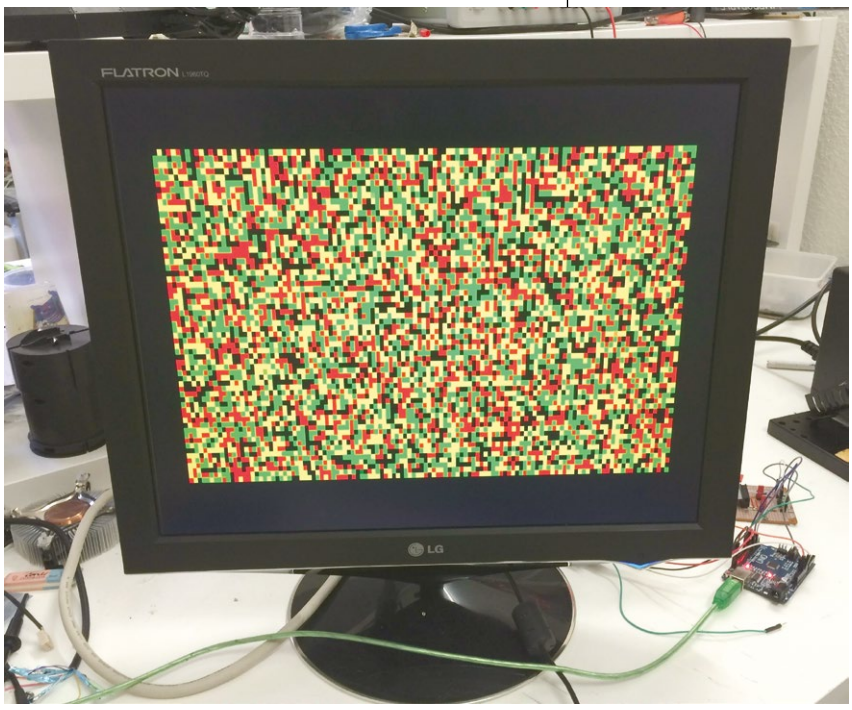
Avant toute chose, précisons ici qu'une carte comme une Arduino UNO n'est pas une plateforme réellement adaptée à ce type de choses et que tout ceci tient davantage du fait de relever un défi qu'autre chose. Ce qu'a réalisé *Sandro Maffiodo*, et *Nick Gammon* avant lui, est un *hack* et non quelque chose de réellement viable et utilisable pour un projet standard.

Certes, cela fonctionne et une image peut être affichée sur un moniteur VGA, mais les limitations sont très nombreuses. Nous avons déjà évoqué le fait de ne pouvoir utiliser que deux des trois composantes de couleur disponibles et la restriction quant à l'utilisation des broches du PORTD. Mais ce n'est pas tout, loin de là.

La mémoire d'affichage, ou *framebuffer*, occupe 1800 octets. Avec la mémoire utilisée pour le fonctionnement de la bibliothèque, cela ne laisse que quelques 200 octets de mémoire vive utilisables. Ajoutez simplement une bibliothèque et un peu de code pour supporter une horloge (RTC) et ce chiffre sera alors divisé par trois. On est donc très très rapidement à l'étroit.

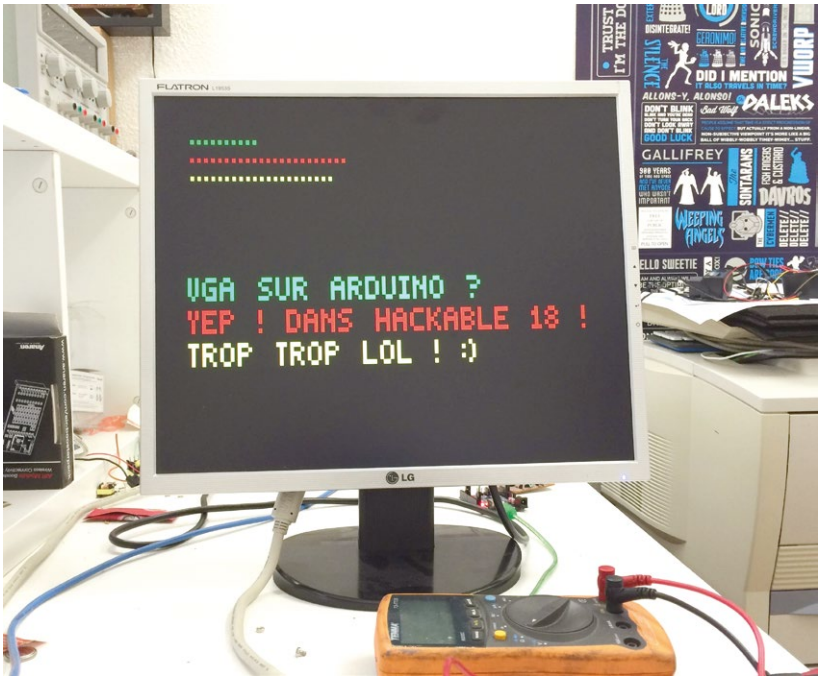
L'utilisation des broches de la carte est également très limitée de bien des façons. Le PORTD tel qu'il est utilisé ici est totalement indisponible, mais les fonctionnalités de communication série ou encore en SPI sont également passées sous silence. Les broches utilisées pour les signaux ne sont pas modifiables et sont intimement liées à l'utilisation des *timers*. Ceux-ci sont tous utilisés pour la génération du signal vidéo et n'importe quelle autre interruption impactera l'intégrité de l'affichage. De plus, étant donné l'utilisation alternative des *timers*, les fonctions comme `delay()` ou `millis()` ne sont plus accessibles et il faudra utiliser des équivalents mis à disposition par la bibliothèque.

De façon générale, la carte Arduino va littéralement passer le plus clair de son temps à produire le signal vidéo et n'importe quel traitement prolongé non interruptible perturbera l'affichage.



Le premier croquis permettant de tester le montage consiste à afficher des pixels de façon aléatoire. Ceci permet non seulement de se familiariser avec la bibliothèque, mais également de tester l'ensemble. L'image doit s'afficher de façon stable et sans parasite.

L'affichage de texte est l'une des utilisations réellement pratiques pour ce type de projet. Il faut cependant garder à l'esprit qu'il reste bien peu de ressources pour gérer les informations devant être affichées. Hors de question, par exemple, de présenter un texte provenant du moniteur série, celui-ci n'est tout simplement plus utilisable...



La bibliothèque VGAX de *Sandro Maffiodo* n'est pas installable directement depuis le gestionnaire de l'environnement Arduino. Vous devrez donc la récupérer sur **https://github.com/smaffer/vgax** et la copier dans un sous-répertoire **libraries** de votre répertoire de croquis. Ceci fait, au prochain lancement de l'environnement Arduino, celle-ci sera utilisable et les exemples viendront s'ajouter à ceux déjà présents dans le menu.

Pour commencer, et faire un premier test, le plus simple est encore d'utiliser l'exemple **RandomPixels**, que je reprends ici, commenté et légèrement modifié :

```
#include <VGAX.h>

// Objet représentant l'écran
VGAX vga;

void setup() {
  // initialisation
  vga.begin();
}

// Compteur
uint8_t cnt;

void loop() {
  // affichage d'un pixel aléatoire
  vga.putpixel(rand()%VGAX_WIDTH, rand()%VGAX_HEIGHT, cnt%4);
  cnt++;
}
```

Après inclusion de la bibliothèque et déclaration de l'objet **vga** nous permettant d'utiliser l'écran, celui-ci est initialisé avec la méthode **begin()** dans la fonction **setup()**. Il n'y a pas d'argument, ni pour la déclaration, ni pour l'initialisation, car rien ne peut être modifié. Le code de la bibliothèque est totalement optimisé pour l'utilisation des *timers* et broches associées et ne sert qu'une fonction et une seule.

Notre fonction **loop()** est simpliste. Après avoir déclaré une variable globale **cnt**, **loop()** ne fait que placer des pixels aléatoires à l'écran. La méthode **putpixel()** prend en argument une position horizontale, verticale et une couleur entre 0 et 3 correspondant respectivement au noir, à la première couleur,

la seconde et le mélange des deux. La syntaxe utilisée ici, avec `%`, est celle du modulo, le reste de la division euclidienne (la division « normale » qu'on a apprise à l'école). Les macros `VGAX_WIDTH` et `VGAX_HEIGHT` correspondent respectivement à la largeur de l'écran en pixels et sa hauteur. Là encore l'opération modulo est utilisée sur la valeur aléatoire retournée par `rand()`, nous obtenons donc toujours une valeur entre 0 et `VGAX_WIDTH-1`, ou `VGAX_HEIGHT-1`.

La méthode `putpixel()` est bien pratique si nous voulons dessiner à l'écran pixel par pixel, mais parmi les différentes fonctionnalités proposées, quelques unes sont bien plus pratiques.

Nous pouvons, par exemple, afficher du texte :

```
#include <VGAX.h>
// fichier de police
#include "font.h"

// objet écran
VGAX vga;

// notre chaîne de caractères
static const char str0[] PROGMEM="VGA AVEC ARDUINO";

void setup() {
  // initialisation
  vga.begin();
  // effacement écran
  vga.clear(0);

  // affichage du texte
  vga.printPROGMEM(
    (byte*)fnt_nanofont_data,              // police
    FNT_NANOFONT_SYMBOLS_COUNT,            // nombre de symboles
    FNT_NANOFONT_HEIGHT,                   // hauteur caractères
    3,                                     // séparation horizontale
    1,                                     // séparation verticale
    str0,                                  // chaîne à afficher
    0,                                     // emplacement X
    VGAX_HEIGHT/2-FNT_NANOFONT_HEIGHT/2,   // emplacement Y
    3);                                    // couleur
}

void loop() {
  vga.delay(100);
}
```

Un écran VGA, bien entendu, n'a aucune notion de « texte » ou de police de caractères, c'est donc à notre croquis de définir les symboles correspondant aux caractères utilisés. La bibliothèque met à disposition une page web et un bout de code JavaScript permettant, à partir d'une image construite avec un logiciel de dessin, de créer des lignes de code définissant la correspondance caractère/symbole. Pour utiliser cet outil, vous devrez ouvrir le fichier `2bitfont.html` avec votre navigateur web. Celui-ci se trouve dans un sous-répertoire `tools`, directement dans le répertoire de la bibliothèque, dans votre carnet de croquis.

Pour composer une nouvelle police de caractères, vous devez créer un fichier image où les symboles des caractères ASCII de 33 à 128 sont présentés sur une seule ligne. La hauteur de

l'image en pixels correspond à la hauteur des symboles et ceux-ci doivent être noirs sur blanc et séparés les uns des autres par une colonne de pixels blancs (si un symbole intègre une colonne blanche, placez-y un pixel rouge). La largeur maximum d'un symbole est de 8 pixels.

Ouvrez la page web avec votre navigateur, sélectionnez le fichier image à utiliser, spécifiez un nom de variable et cliquez sur « GENERATE ». Vous obtiendrez normalement des lignes de code à intégrer directement dans votre croquis ou, comme ici, dans un fichier **font.h** à inclure dans votre projet :

```
//font generated from BITFONZI - by Sandro Maffiodo
#define FNT_NANOFONT_HEIGHT 6
#define FNT_NANOFONT_SYMBOLS_COUNT 95
//data size=570 bytes
const unsigned char
fnt_nanofont_data[FNT_NANOFONT_SYMBOLS_COUNT][1+FNT_NANOFONT_HEIGHT]
PROGMEM={
{ 1, 128, 128, 128, 0, 128, 0, }, //glyph '!' code=0
{ 3, 160, 160, 0, 0, 0, 0, }, //glyph '"' code=1
[...]
{ 3, 96, 192, 0, 0, 0, 0, },  //glyph '~' code=93
{ 4, 48, 64, 224, 64, 240, 0, },  //glyph '£' code=94
};
```

En ce qui concerne l'utilisation de cette variable avec la méthode **printPROGMEM()**, vous remarquerez que les macros **FNT_NANOFONT_SYMBOLS_COUNT** et **FNT_NANOFONT_HEIGHT** sont définies pour vous. Tout ce qu'il vous reste à faire dans les arguments de la méthode est donc de spécifier l'emplacement du début du texte, l'espacement vertical (retour à la ligne avec **\n** dans la chaîne) et horizontal (espace), et la couleur du texte entre 0 et 3, comme pour **putpixel()**.

Si vous ne voulez pas vous amuser à générer votre propre police de caractères, vous pouvez tout simplement, réutiliser celle de l'exemple **BitFont** inclus avec la bibliothèque.

Une autre méthode très intéressante à utiliser est **copy()**. Celle-ci permet de copier les données d'une variable directement à l'écran. Exposé ainsi, ceci ne semple pas très amusant, mais c'est pourtant ce qui vous permet d'afficher des images très facilement. Ces images sont placées dans des variables exactement comme les symboles d'une police de caractères et un autre outil à votre disposition sous forme d'une page web vous permettra de les générer. Cette page **2bitimage.html** fonctionne comme la précédente, à la différence qu'ici c'est toute l'image qui est utilisée en une fois.

Composez ainsi une image d'une taille égale à 120×60 avec votre logiciel de retouche d'images préféré (The Gimp par exemple) en prenant soin de ne la créer qu'avec les quatre couleurs utilisables (rouge, vert, jaune et noir). Ouvrez ensuite la page web et sélectionnez le fichier à traiter. Nommez la variable comme il vous plaît puis cliquez sur « GENERATE ». Notez que la notion de nuance de couleurs n'existe pas ici. De ce fait, un pixel de l'image étant par exemple à 70% rouge deviendra tout simplement un pixel rouge. Il est donc recommandé de travailler votre image dans un premier temps en 24 bits puis de la passer ensuite en couleur indexée (4 couleurs) pour avoir un aperçu du résultat final.

La méthode **copy()** part du principe que les données font la taille de l'écran, il n'est donc pas question de travailler à partir d'une image d'une taille différente de 120×60. D'un autre

côté, le seul argument de la méthode est un pointeur sur la variable contenant les données : `vga.copy((byte*)img_varname_data)` (`varname` étant ici le nom utilisé dans la page web pour générer le code). Notez que le code généré déclare un tableau de `unsigned char`, un type classique en C/C++, mais que la méthode attend un point vers des `byte`. De ce fait on « force » ou *cast*, le type en le précisant entre parenthèses avant le nom de la variable (`(byte *)`, un pointeur sur un/des `byte`).

D'autres méthodes sont disponibles permettant, par exemple, de travailler avec des *sprites*, de petites images pouvant être « tamponnées » à l'écran, à partir de données issues du générateur d'image. Pour connaître l'ensemble des méthodes disponibles, il vous suffit de jeter un œil dans la bibliothèque elle-même et en particulier dans le fichier `VGAX.h`, où se trouve également la documentation en commentaire.

Vous pouvez également inclure `VGAXUtils.h` en lieu et place de `VGAX.h` afin de pouvoir utiliser quelques primitives graphiques basiques comme le tracé de lignes, rectangles, cercles, disques, etc.

Enfin, il est à noter que la bibliothèque VGAX permet également de générer du son en branchant un buzzer entre la broche A0 et la masse. Ceci est intégré de base dans le fonctionnement général de la bibliothèque et n'impacte donc pas la génération du signal vidéo. Ne vous attendez pas cependant à une grande qualité de ce point de vue, là encore, nous frôlons les limites absolues de ce que peut faire un ATMega328a d'une carte Arduino UNO.

Remplir la mémoire d'images composées spécifiquement et les afficher les unes à la suite des autres est une utilisation possible et accessible dans ce genre de situation. De quoi, par exemple, embellir une vitrine de magasin ou attirer le regard à peu de frais et bien plus facilement qu'avec des leds.

4. PROBLÈME ET SOLUTIONS

J'ai pu, comme d'autres utilisateurs de cette bibliothèque, constater que le comportement du croquis changeait en fonction de la version de l'environnement Arduino utilisée. En version 1.6.4, le code et les exemples de *Sandro Maffiodo* fonctionnent sans le moindre problème, mais il en va tout autrement avec la version 1.6.13 ou 1.8.1. Il semblerait en effet que l'optimisation du code réalisée lors de la compilation impacte directement les délais précisément ajustés dans la bibliothèque.

Les symptômes liés à ces changements sont un scintillement de l'image, presque permanent, qu'il y ait changement ou non dans les pixels affichés. L'effet est davantage présent lorsque l'image est rafraîchie rapidement. L'exemple `MarioSprites` par exemple, destiné à afficher de manière aléatoire une petite image donne un résultat presque entièrement composé de parasites et de « bavures » à l'écran.

Le problème provient très certainement d'une optimisation faite par le compilateur, mais déterminer son origine précise est un coût en temps qui ne vaut pas la chandelle. Le code de *Sandro Maffiodo* repose sur

le concept initial de *Nick Gammon,* mais intègre également des optimisations de *Charles Lohr* initialement développées pour son serveur Minecraft pour microcontrôleurs Atmel AVR. Lire le contenu de la bibliothèque est très instructif, mais appréhender son fonctionnement n'est pas une tâche aisée et rapide.

Il semblerait cependant que le problème de délais/temporisation réside dans la fonction **ISR(TIMER2_OVF_vect)** (qui est le vecteur d'interruption pour le débordement du *timer* 2). Ce code dans **VGAX.cpp** contient, entre autres, des lignes en assembleur, et en particulier des instructions **nop** (pas d'opération) destinées à « perdre » des cycles d'horloge pour obtenir la fréquence adéquate pour HSYNC :

```
[...]
    "no_audio:              \n\t"
    "       nop             \n\t" //c1
    "       nop             \n\t" //c1
    "       nop             \n\t" //c1
    //"     nop             \n\t" //c1
    "       nop             \n\t" //c1
    "       nop             \n\t" //c1
    "       nop             \n\t" //c1
    "       rjmp end        \n\t" //c2
[...]
```

Une ligne est déjà commentée par défaut et le fait de faire de même pour trois autres semble régler le problème, du moins avec l'environnement en version 1.8.1. Ceci est, bien entendu, un horrible bricolage à peine digne d'être qualifié de rustine, mais comme dit précédemment, l'ensemble des limitations déjà existantes ne justifie pas, selon moi, une investigation plus poussée.

Notez également que la question de la compatibilité du code de la bibliothèque avec une carte Arduino Mega 2560 a été soulevée dans les discussions sur GitHub concernant cette bibliothèque. À l'heure actuelle, aucun résultat fonctionnel n'est ressorti de ces discussions. Les quelques programmeurs qui s'y sont essayés ont fini par jeter l'éponge, l'effort ne justifiant pas le gain. Le fait de pouvoir utiliser la bibliothèque VGAX sur une carte Mega permettrait de disposer, en principe, de davantage de mémoire pour les croquis, mais les limitations en termes de performances et de résolution resteraient inchangées. Tout au plus ceci permettrait de revoir le fonctionnement de la bibliothèque de façon à utiliser un double tampon d'affichage. Ceci, en principe, devrait éliminer le scintillement de l'image lors de la création d'une animation (un tampon étant modifié alors que l'autre est affiché, puis on permute les tampons pour changer l'image. C'est la technique du *double buffering*).

CONCLUSION

Oui, il est possible de générer un signal vidéo analogique avec une carte Arduino, mais ceci accapare totalement les ressources de la carte et ne nous laisse que peu de marge de manœuvre pour faire quelque chose de véritablement utile. Quelques réalisations pratiques existent toutefois sur le Web, comme un jeu de Pong par exemple, mais cela reste davantage des *challenges* et démonstrations de compétences en programmation qu'autre chose.

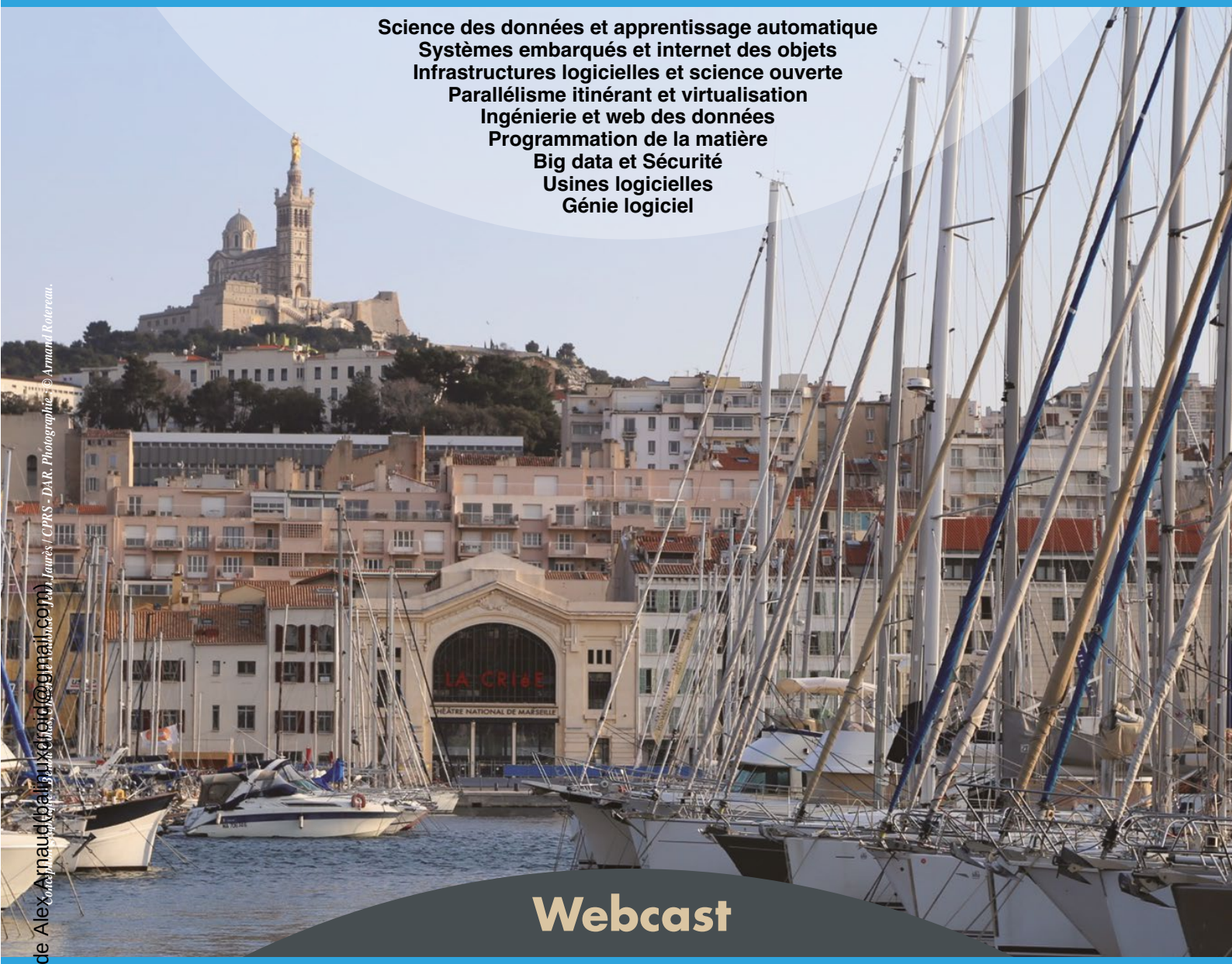
Il reste toutefois des choses relativement accessibles et utiles à faire avec ce type de montage comme l'affichage d'images ou de textes stockés en flash en fonction de l'état de broches restant utilisables (ou tout simplement d'une temporisation). La mise en œuvre d'une horloge sous forme de module RTC en i2c est également envisageable en tolérant le scintillement lors de la lecture des informations. Ceci peut être valable également pour un affichage de température et d'hygrométrie avec un module BME180 (voir *Hackable n°14*) qui sera alors bien plus visible qu'un afficheur LCD ou à leds.

Quoi qu'il en soit, si vous avez un vieux moniteur VGA qui traîne, quatre résistances n'est pas cher payé pour vous amuser à explorer le domaine et peut-être fouiller un peu dans les entrailles de la bibliothèque pour en tirer quelques leçons.... **DB**

JDEV 2017

Journées Développement Logiciel

Science des données et apprentissage automatique
Systèmes embarqués et internet des objets
Infrastructures logicielles et science ouverte
Parallélisme itinérant et virtualisation
Ingénierie et web des données
Programmation de la matière
Big data et Sécurité
Usines logicielles
Génie logiciel



Vieux-Port, Jaurès / CPRS - DAR. Photographie : © Arnaud Rauereau.

Webcast

4, 5, 6, 7 juillet 2017

Aix-Marseille Université, la Canebière

JDEV2017

Information, programme, réservation et inscription :

http://devlog.cnrs.fr/jdev2017

CRÉEZ VOTRE PORTIER AUDIO CONNECTÉ

Eric Le Bras

Les grandes surfaces de bricolage proposent une variété de modèles de portiers (ou « interphones ») audios ou vidéos. Filaire ou sans fil, analogique ou numérique, le point commun de ces appareils communicants est de reposer sur des conceptions propriétaires et fermées. Le plus souvent ces produits ne sont pas interopérables, et sont conçus pour ne fonctionner qu'avec le récepteur fourni. Pourtant, des logiciels libres basés sur des protocoles ouverts sont disponibles et permettraient d'adapter facilement l'appareil à l'usage que souhaite en faire son propriétaire. D'où l'envie de modifier un de ces appareils pour en faire un portier IP.

Dans un premier temps, j'avais jeté mon dévolu sur un modèle sans fil du commerce. Dès les premiers essais, je dus me rendre à l'évidence : les performances attendues n'étaient pas vraiment au rendez-vous. D'une part la médiocre qualité sonore et le niveau de bruit rendaient la conversation très difficile, voire impossible au-delà de quelques mètres de distance entre émetteur et récepteur ; d'autre part, l'autonomie de l'appareil, alimenté par 4 piles de type AA, était limitée à quelques semaines de fonctionnement. Ce dernier point aurait pu (il est vrai) être réglé en amenant l'énergie jusqu'au portail (un bloc d'alimentation était fourni), mais... si j'avais eu en projet d'amener des câbles depuis la maison jusqu'au portail, pourquoi aurais-je choisi ce coûteux modèle sans fil ?

La nécessité de tirer du câble s'imposa toutefois rapidement avec le projet de motoriser le portail. Ce chantier fût l'occasion d'amener le 230 volts depuis la maison, et, puisque j'en étais à installer des gaines et du câblage, j'en profitai bien évidemment pour tirer un câble réseau. Ainsi me disais-je, j'aurai tout loisir ensuite de mettre en œuvre une solution d'interphone réellement fonctionnelle.

La tentation était dès lors présente de récupérer le boîtier et les éléments périphériques de feu le portier sans fil, pour en faire un véritable interphone connecté ou portier IP. Une étude des solutions existantes m'amena à concevoir mon propre équipement, sur la base de protocoles, bibliothèques, langages et matériels ouverts et documentés. L'idée finale était de me servir de la platine de rue pour établir une liaison téléphonique avec un portable ou le téléphone fixe de la maison, en exploitant le réseau domestique et la connexion internet.

Budget final : une quinzaine d'euros (sans le boîtier, et sans le câble réseau).

Fig. 1 : La platine de rue. En apparence, un simple interphone du commerce sans fioriture... en apparence, seulement !

1. UN PEU DE CONCEPTION

La fonctionnalité attendue se présente de la manière suivante : lorsque le visiteur presse le bouton de l'interphone, une communication est établie avec un smartphone ou le téléphone fixe de la maison. Une fois la connexion établie, les deux personnes peuvent converser. Si l'occupant des lieux décide d'accorder son sésame au visiteur, il peut commander l'ouverture du portail ou du portillon en appuyant sur une touche du téléphone. On prévoit également une fonctionnalité de raccrochage automatique au bout d'un délai prédéfini, dans le cas où l'appel aboutit à la messagerie vocale.

L'interphone repose sur l'utilisation du protocole SIP, et de divers autres protocoles associés. Ces protocoles sont standardisés par l'IETF, et constituent aujourd'hui la norme pour réaliser des communications téléphoniques sur réseau de données (VoIP). L'interphone se comporte comme un téléphone IP (un *user agent* SIP), au détail près que le numéro appelé est paramétré à l'avance. Lorsque la connexion est sollicitée, il s'enregistre sur un serveur (un « trunk SIP ») puis établit une communication vers le destinataire défini dans les paramètres (un identifiant SIP, cet identifiant pouvant éventuellement correspondre à un numéro de téléphone, en l'occurrence le téléphone fixe de la maison).

Les fonctionnalités SIP sont fournies par la librairie libre **PJSIP** **[1]**.

1.1 À propos du trunk SIP

Le protocole SIP sert à initier une *session* entre deux (ou plus) *user agents* (téléphone IP, smartphone, softphone ou autre). Les *user agents* n'étant généralement pas joignables directement par leur adresse IP (adressage dynamique ou derrière un pare-feu), ils doivent s'être authentifiés auprès d'un *registrar* en utilisant leur propre identifiant SIP et mot de passe. Cet identifiant est semblable à une adresse e-mail, et se présente sous la forme **nom@registrar** (par exemple : clint.eastwood@sip.linphone.org).

Notons qu'il existe toutefois la possibilité de connecter entre eux deux *user agents* sans authentification préalable auprès d'un *registrar*. Ceci suppose que l'adresse IP de l'appelé soit connue et accessible. Dans ce cas, une *session* en mode *peer to peer* est initiée entre les deux *user agents*. Le client récepteur peut être un téléphone SIP du marché, voire un récepteur conçu sur le même modèle que l'interphone. Hormis ce cas particulier (non mis en œuvre dans le projet actuel, mais envisageable comme évolution), il faudra s'en remettre à la situation la plus habituelle, et donc posséder un compte auprès d'un *registrar* (un *trunk* SIP). L'offre est abondante, et généralement gratuite lorsque le service se limite à la mise en relation de *user agents* SIP connectés à Internet. Les services payants s'adressent plutôt aux entreprises, et permettent en plus les communications vers des numéros de téléphone.

À ce sujet, Free permet aux abonnés Freebox d'accéder à son propre *trunk* SIP, utilisé par la fonction téléphonie de la Freebox. Par rapport aux services gratuits, l'intérêt est de pouvoir appeler tous les numéros de téléphones fixes en France, y compris le propre numéro de l'abonné (en revanche les numéros de mobiles ne sont pas joignables - ceci n'est pas un véritable problème, car il reste possible de joindre un smartphone Android connecté au net comme nous le verrons plus loin). C'est à ma connaissance le seul moyen gratuit (pour un abonné Freebox) d'utiliser l'interphone avec le téléphone fixe du domicile. En bonus, vous disposerez d'une messagerie en cas d'absence. Une limitation toutefois : je ne suis pas parvenu à détecter les codes DTMF générés par l'appui sur les touches du téléphone pendant la communication ; cette limitation n'existe que lorsque le numéro de téléphone fixe est appelé. Si le correspondant est un *user agent* SIP connecté directement au net, même en passant par le *trunk* de Free, pas de problème.

Les informations pour activer le service SIP de Free se trouvent ici : **http://www.free.fr/assistance/89.html**.

Si vous n'êtes pas abonné Freebox, vous devrez créer un compte auprès d'un *trunk* gratuit. La procédure pour créer un compte sur **Linphone.org** est simple : rendez-vous sur **https://www.linphone.org/free-sip-service.html** puis renseignez les informations demandées. Le *username* sera le nom d'utilisateur qui vous identifiera auprès du *registrar*. Une fois votre compte créé, vos informations de connexion se présentent ainsi :

Fig. 2 : Mes paramètres SIP chez Linphone.org.

À noter que l'hébergeur français OVH propose comme alternative un service « VoIP Découverte » au tarif de 1,19 € par mois, accessible aux particuliers, permettant les appels vers les fixes en France ainsi que 40 destinations. Je n'ai pas testé cette solution.

2. CÔTÉ MATÉRIEL

Le choix de la plateforme matérielle s'est porté sur la carte Orange Pi Zero. Il s'agit d'une carte de développement bâtie autour du SoC Allwinner H2+. Les raisons qui ont présidé à ce choix sont les suivantes :

- présence d'un port réseau RJ45 100 Mbps ;
- Wifi intégré, avec une antenne laissant présager une certaine portée (à l'essai, la portée s'est révélée suffisante pour se connecter à une box située à environ 25 m avec traversée d'un mur porteur) ;
- alimentation PoE optionnelle (non testée) ;
- entrée micro et sortie ligne ;
- encombrement réduit (46 x 48 mm). Ce facteur était dicté par l'objectif initial d'embarquer la carte dans la platine de rue. Toutefois nous verrons dans un second temps que ce n'est pas cette conception qui fut finalement retenue. Reste que la présence des entrées/sorties audios (ainsi que 2 ports USB, une sortie vidéo et un récepteur infrarouge) sur un connecteur 13 broches facilite malgré tout grandement l'intégration dans une réalisation embarquée ;
- faible coût (6,70 € avec 256 Mo, 8,62 € en version 512 Mo). Avec moins de deux euros d'écart, le modèle 512 Mo est certainement le plus avantageux. Cependant, il est toujours intéressant de savoir que seulement 11 % de cette capacité s'avère nécessaire à l'usage.

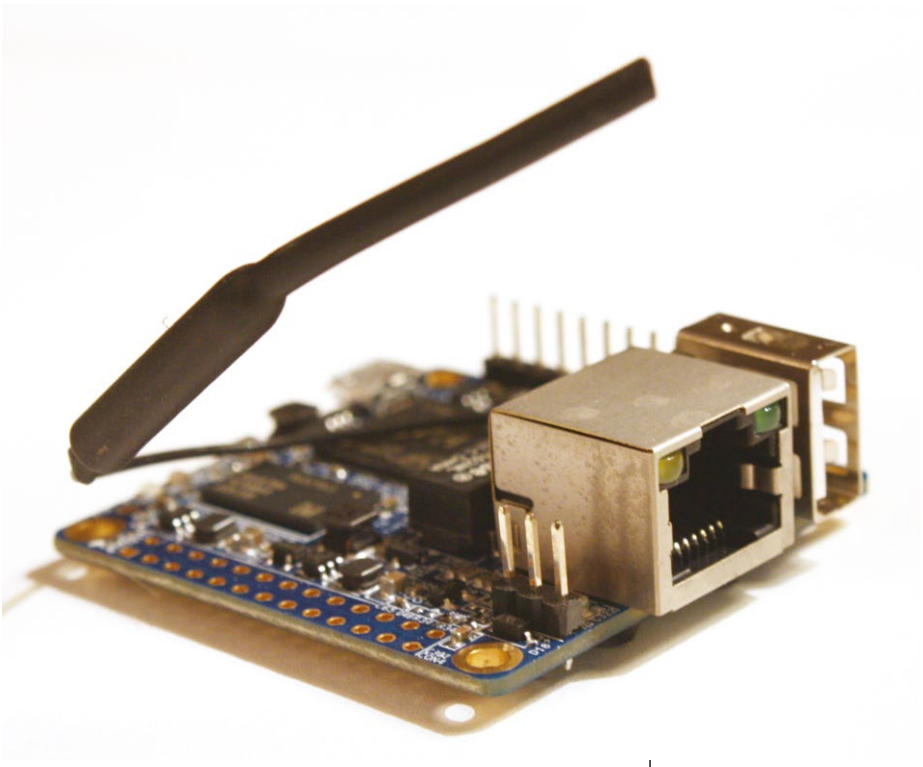


Fig. 3 : Contrairement à sa cousine Raspberry Pi Zero, Orange Pi Zero est dotée de deux interfaces réseau. Elle arbore fièrement sa petite antenne Wifi ainsi que son port RJ45 100 Mbit/s. Au premier plan, on remarque également les 26 trous du GPIO, où manquent encore les broches.

La Raspberry Pi Zero, un temps envisagée, présente un encombrement comparable, toutefois elle est fournie sans connectivité réseau (la Pi Zero W existe depuis peu, mais reste difficile à trouver en France), sans entrées/sorties audios, et avec une puissance notablement inférieure (mono cœur 1 GHz, là où l'Orange Pi Zero dispose d'un quadruple cœur 1,2 GHz). L'arrivée récente de cette dernière (novembre 2016) est donc apparue comme particulièrement opportune pour ce projet, et est parvenue à vaincre les réserves initiales concernant les limites du support (documentation et logiciels) fourni par le constructeur ainsi qu'une communauté limitée.

L'Orange Pi Zero présente également un GPIO de 26 broches reprenant la disposition du Raspberry Pi 1. Ce connecteur permettra l'interfaçage avec le bouton et les leds de la platine de rue, ainsi que les commandes d'ouverture du portail et/ou du portillon. À noter que ce GPIO n'est pas équipé d'origine. Vous devrez donc souder 2 *headers* de 13 broches au pas de 0,1 pouce (2,54 mm) dans les trous prévus avant de l'utiliser.

Notons au passage que la carte ne possède pas de sortie HDMI, ce qui la positionne d'emblée comme une plateforme tournée vers l'embarqué.

Le site communautaire **linux-sunxi.org** est un bon point de départ pour trouver des informations techniques sur les cartes basées sur la gamme de SoC AllWinner **[2]**.

En plus de la carte, vous aurez besoin de :

- Un module d'amplification. La sortie ligne n'offre pas suffisamment de puissance pour « driver » le haut-parleur de 0,5 W qui équipe la platine de rue. Suivant les conseils d'un ami chinois, nous avons choisi le PAM8403, un module d'amplification classe D de 3 W disponible au prix dérisoire de moins d'un euro chez les commerçants de l'empire du Milieu (à ce prix-là, prenez-en deux ou trois - si vous ne savez pas pourquoi maintenant, l'usage en viendra bien un jour !). Veillez à bien choisir un modèle équipé d'un potentiomètre de réglage du gain, indispensable pour ce montage. Le module contient deux canaux. La puissance d'un seul canal étant largement suffisante pour notre application, on utilisera indifféremment le gauche ou le droit.
- Un micro électret. Dans notre cas, nous avons réutilisé celui équipant la platine de rue, mais un modèle quelconque acheté ou récupéré sur un ancien appareil, périphérique PC ou jouet devrait faire l'affaire. Pour être utilisable avec l'Orange Pi Zero, le micro électret nécessite un circuit d'alimentation et de filtrage que nous pouvons réaliser facilement avec quelques résistances et condensateurs (deux résistances 0,25 W 6,8 kΩ et 4,7 kΩ, un condensateur électrolytique 47 µF et un condensateur céramique 1 µF).
- Un haut-parleur. Nous avons réutilisé le haut-parleur 0,5 W présent sur la platine de rue.
- Un bouton poussoir et une ou plusieurs leds. Dans le cadre de notre réalisation pratique, nous avons réutilisé les composants déjà disponibles sur l'interphone existant (voir plus loin).
- Une alimentation 5 V 2000 mA (voir paragraphe suivant).
- En option : un ou plusieurs relais pour commander l'ouverture du portail et/ou de la gâche. Le plus simple est peut-être de faire l'achat d'une « breakout board » équipée du nombre de relais désiré. Attention : les relais doivent fonctionner avec une logique de commande sous 3,3 volts. Très souvent ces cartes sont équipées d'un optocoupleur. Dans ce cas, piloter le relais revient à allumer ou éteindre une led, ce qui facilitera le raccordement avec le GPIO (voir l'article consacré aux relais dans *Hackable n° 7*). Toutefois, la sécurité apportée par l'optocoupleur ne sera effective qu'à condition que le relais ait sa propre alimentation. La tension commandée n'étant pas dangereuse, nous n'avons pas opté pour cette solution.

3. SCHÉMA DE PRINCIPE

Le schéma complet est présenté ci-dessous. Ce schéma comporte une partie numérique connectée au GPIO (à gauche) et une partie analogique reliée au connecteur 13 broches (à droite).

Une alimentation unique peut être utilisée pour l'Orange Pi Zero et le PAM8403 (ce dernier pouvant fonctionner avec une tension d'entrée comprise entre 2,5 volts et 5,5 volts), mais attention : si vous alimentez l'Orange Pi Zero via le port micro USB, vous ne pourrez pas alimenter le PAM8403 via les broches +5V et GND de l'OPZ. Le courant sera insuffisant et l'Orange Pi Zero ne démarrera pas. La solution consiste à alimenter l'Orange Pi Zero non par le port micro USB, mais directement via les broches +5V et

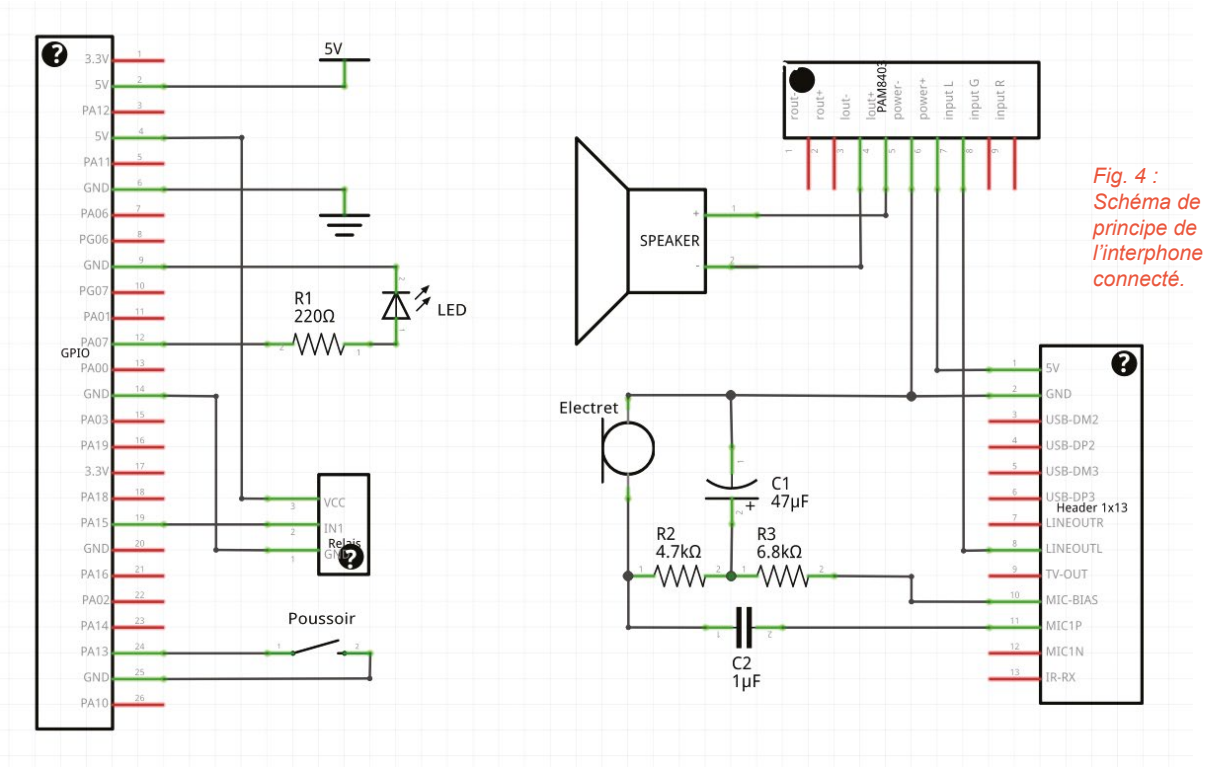


Fig. 4 : Schéma de principe de l'interphone connecté.

GND du GPIO. L'ampli PAM8403 prélèvera le courant nécessaire indifféremment sur les autres broches disponibles du GPIO ou du connecteur 13 broches, les broches +5V et GND étant toutes reliées électriquement entre elles. Ceci revient donc à raccorder les deux appareils en parallèle sur la même alimentation.

En résumé, l'Orange Pi Zero sera alimenté via les broches 4 (+5V) et 6 (GND) du GPIO, le PAM8403 est alimenté via les broches 1 (+5V) et 2 (GND) du connecteur 13 broches.

L'alimentation doit être suffisante pour faire fonctionner les deux appareils. Nos essais montrent que deux ampères suffisent. On pourra soit utiliser une alimentation USB, en coupant la prise micro USB pour récupérer les fils correspondant au +5V et à la masse, soit se procurer une alimentation 5 V continu universelle du type de celles proposées sur eBay en recherchant « 5V DC 2A universal switching power supply ».

Passons maintenant en revue le raccordement des différents éléments avec l'Orange Pi Zero.

3.1 Micro

Un électret est un microphone statique, qui ne génère pas de courant par lui-même (à la différence d'un microphone dynamique, dont la vibration de la membrane induit un courant dans une bobine se déplaçant dans le champ magnétique d'un aimant fixe). Un microphone statique doit être relié à une source de courant. L'onde sonore captée par la capsule va moduler ce courant.

Le schéma de montage testé et retenu dans le cadre de notre projet provient du site **www.sonelec-musique.com**, une mine d'informations concernant notamment l'utilisation des micros électrets, et dont nous vous recommandons la lecture **[3]**. Ce montage simple nécessite seulement deux résistances et deux condensateurs pour exploiter la capsule. Il a donné satisfaction dans le cadre du projet, avec des longueurs de câble raisonnables. Si toutefois ce montage ne fonctionne pas correctement pour vous, ou si le câble entre le montage et l'Orange Pi Zero doit être d'une

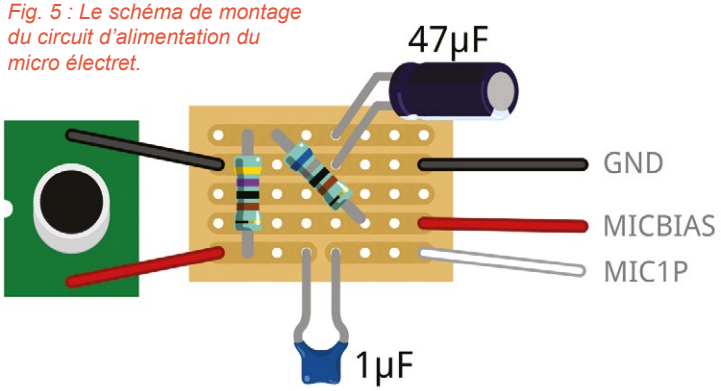


Fig. 5 : Le schéma de montage du circuit d'alimentation du micro électret.

certaine longueur, vous pourrez trouver un montage plus avancé, exploitant les entrées micro différentielles MIC1P et MIC1N, sur le diagramme schématique de l'Orange Pi Lite disponible sur le site de Xunlong **[4]**.

Notre micro électret possède deux pattes. L'une (reliée au boîtier de la capsule) doit être connectée à la masse. L'autre patte sert à la fois à alimenter le micro et à récupérer la modulation qui sera transmise à l'étage de pré-amplification. Elle est connectée d'une part à la source d'alimentation (broche MIC-BIAS de l'Orange Pi Zero), via deux résistances et un condensateur de filtrage. Le signal quant à lui est prélevé sur la même patte et envoyé sur la broche MIC1P. Comme l'alimentation et la sortie du micro se font sur la même patte, il est nécessaire de bloquer le passage du courant continu d'alimentation vers l'étage de pré-amplification, en utilisant à nouveau un condensateur.

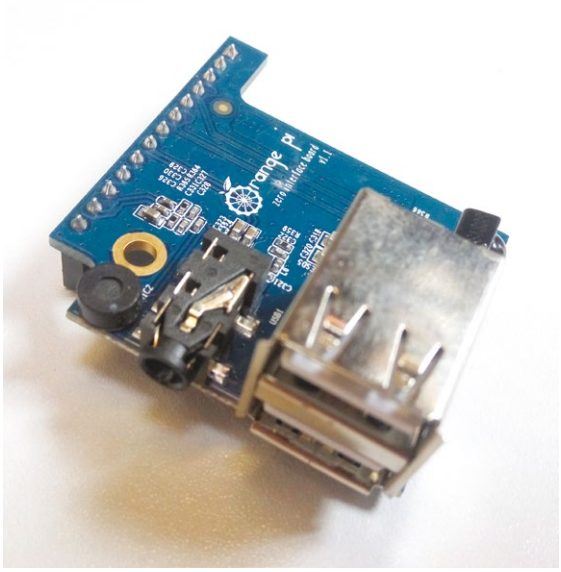


Fig. 6 : Cette carte d'extension se fiche sur le connecteur d'entrées/sorties 13 broches de l'Orange Pi Zero. Elle est dotée d'un micro électret, d'une sortie audio/vidéo jack, de deux ports USB et d'un capteur infrarouge. À moins de deux euros, c'est un moyen pratique et économique de commencer à tester les possibilités audios de l'Orange Pi Zero avant de s'attaquer à la réalisation des circuits nécessaires pour l'interphone.

Les résistances R1 et R2 permettent de limiter le courant d'alimentation. Selon les caractéristiques de la capsule, il peut s'avérer nécessaire d'adapter leurs valeurs. En l'absence d'informations précises, et si le niveau du signal s'avère insuffisant ou trop important, c'est l'expérimentation et le tâtonnement qui nous guideront.

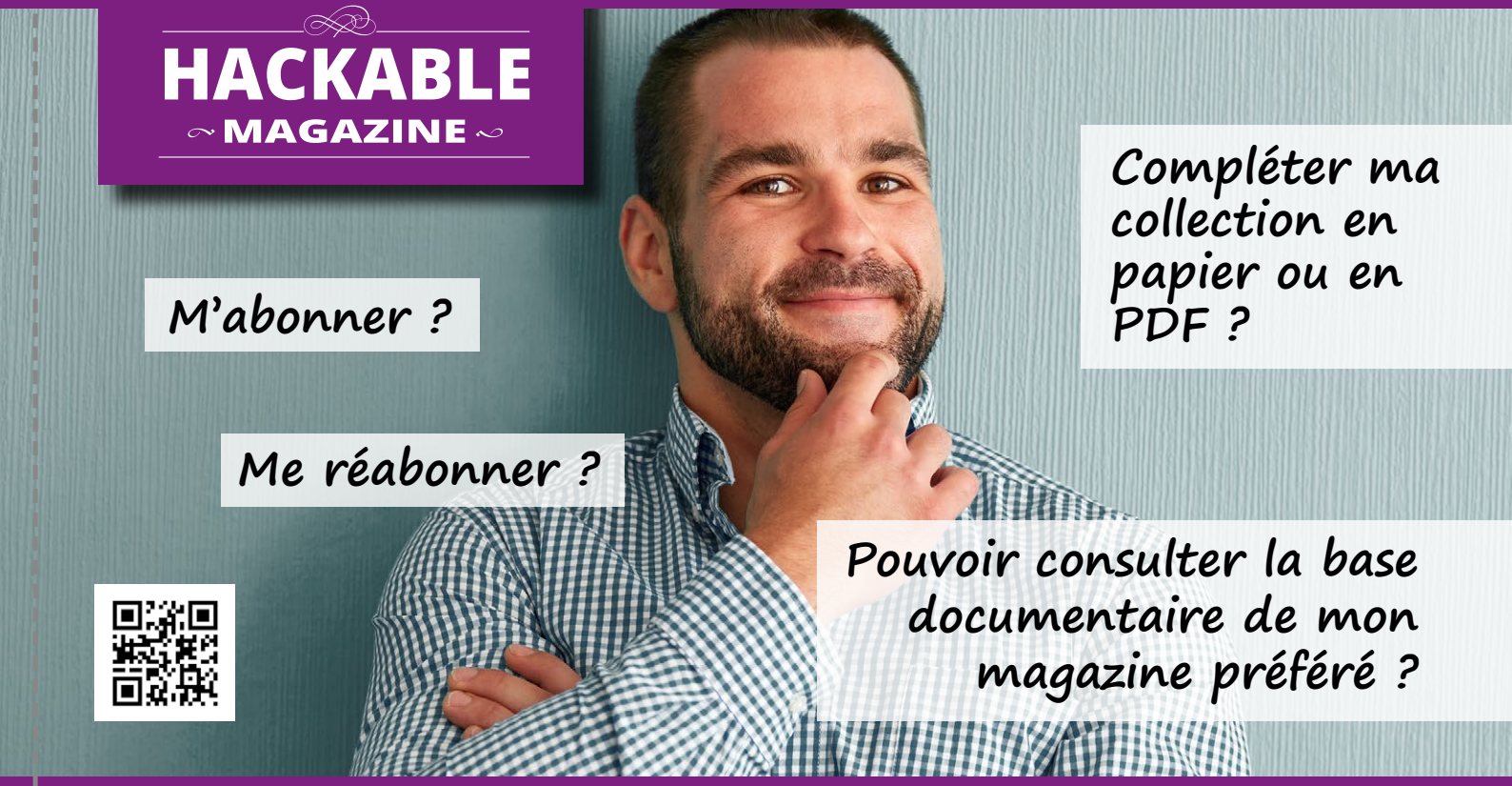
Le condensateur C1 sert à filtrer l'alimentation de la capsule. Dans de nombreuses applications les capsules électrets sont alimentées à l'aide d'une pile, ce qui ne nécessite pas de filtrage. Le condensateur de filtrage ne s'impose donc que dans le cas présent où nous souhaitons alimenter la capsule via la broche MIC-BIAS de l'Orange Pi Zero, prévue pour cet usage. Son rôle est de filtrer les bruits parasites et les ronflements transmis par l'alimentation.

Le condensateur C2 a pour rôle de bloquer le courant d'alimentation continu et de ne laisser passer en sortie que la partie modulée correspondant au signal capté par la capsule.

Le circuit sera relié aux broches suivantes du port 13 broches :

- alimentation : broche MIC-BIAS (10) ;
- sortie : broche MIC1P (11). Notez que la broche MIC1N (12) n'est pas utilisée pour ce montage ;
- masse : broche GND (2).

Abonnez-vous !



C'est simple... c'est possible sur

http://www.ed-diamond.com

... OU 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 : http://**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 :

.....

.....

Bon d'abonnement

CHOISISSEZ VOTRE OFFRE !

Prix TTC en Euros / France Métropolitaine*

Offre	SUPPORT / ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Tarif TTC
HK	6n° Hackable	HK1	39,-	HK13	169,-
LES COUPLAGES AVEC NOS AUTRES MAGAZINES					
i	6n° Hackable + 6n° MISC	i1	79,-	i13	419,-
i+	6n° Hackable + 6n° MISC + 2n° Hors-Série	i+1	99,-	i+13	439,-
J	11n° GNU/Linux Magazine France + 6n° Hackable	J1	105,-	J13	399,-
J+	6n° Hackable + 11n° GNU/Linux Magazine France + 6n° Hors-Série	J+1	159,-	J+13	459,-
K	6n° Hackable + 6n° Linux Pratique	K1	75,-	K13	329,-
K+	6n° Hackable + 6n° Linux Pratique + 3n° Hors-Série	K+1	99,-	K+13	359,-
LA TOTALE DIAMOND !					
L	11n° GLMF + 6n° HK* + 6n° LP + 6n° MISC	L1	189,-	L13	839,-
L+	11n° GLMF + 6n° HS + 6n° HK* + 6n° LP + 3n° HS + 6n° MISC + 2n° HS	L+1	289,-	L+13	939,-

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

Particuliers = CONNECTEZ-VOUS SUR :

http://www.ed-diamond.com

pour consulter toutes les offres !

*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

Professionnels = CONNECTEZ-VOUS SUR :

http://proboutique.ed-diamond.com

pour consulter toutes les offres !

*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

3.2 Ampli et haut-parleur

Comme vu plus haut, les broches power + et - de l'amplificateur PAM8403 sont à relier aux bornes +5V (1) et GND (2) du port 13 broches.

La sortie LINEOUT R ou L (broche 7 ou 8) du connecteur 13 broches de l'Orange Pi Zero est raccordée à la broche input L ou R du PAM8403. La broche input G du PAM8403 est reliée électriquement à la broche power -. Elle est donc déjà de fait reliée à la masse de l'Orange Pi Zero. Pour cette raison, il n'est pas nécessaire de la connecter via un fil supplémentaire.

Les broches rout (sortie droite + et -) ou lout (sortie gauche + et -) sont à raccorder au haut-parleur.

Le choix du canal droit (R) ou gauche (L) n'a aucune importance. Par contre, une fois le choix fait il faut s'y tenir pour l'ensemble du circuit !

3.3 Boutons et leds

Dans le cas de la réutilisation de notre interphone, nous avons voulu conserver ceux qui équipaient d'origine l'interphone. Ceux-ci sont regroupés sur un circuit imprimé sérigraphié, dont sort une nappe de 7 fils aboutissant à un connecteur femelle. Bien sûr il conviendra d'adapter le montage à votre cas particulier. Le principe restera toujours le même, selon que vous réutilisiez un matériel existant, ou que vous choisissiez des composants séparés. Dans ce dernier cas, ou bien en cours de phase d'expérimentation, il vous faut au moins un bouton poussoir et une led.

Le circuit imprimé contient :

- deux boutons (sérigraphiés S3 et S4) connectés en parallèle sur un fil unique, donc en réalité une seule fonction possible. On note la présence d'un emplacement pour un troisième bouton (S1), relié à un fil séparé, mais non équipé ;
- deux leds jaunes (D1 et D2) connectées à un fil unique, destinées à l'éclairage de la platine ;
- une led bicolore rouge/verte (D17), connectée à 3 fils.

Les fonctions des fils sont les suivantes :

Fil	Sérigraphie	Fonction	GPIO
Violet	Boutons S3 et S4	Bouton d'appel	24 [1]
Bleu	Masse		20
Vert	Bouton S1 (non présent)		
Jaune	LED bicolore D17 +		1
Orange	LEDs D1 D2 + (jaune)	Éclairage carte de visite	12
Rouge	LED D17 – rouge		5
Marron	LED D17 - vert		3

Les leds sont montées en série avec des résistances calculées pour une alimentation de 6 volts. Les sorties numériques de l'Orange Pi Zero délivrant 3,3 volts, nous ne prenons aucun risque à ne pas changer la valeur des résistances ! À l'expérience, la perte d'intensité n'a pas posé de problème. Nous avons donc conservé les résistances d'origine.

On remarque que la led bicolore D17 est du type « anode commune ». Cela signifie que la led dispose d'une unique borne plus, et de deux bornes moins (une pour chaque couleur). Cette caractéristique du composant implique un mode d'utilisation inhabituel. Les cathodes (fils rouge et marron) peuvent être connectées à la masse (comme pour une led classique), mais dans ce cas l'activation de la led allumera les deux couleurs en même temps (on obtient de l'orange), sans qu'il soit possible d'obtenir le rouge ou le vert.

[1] Le SoC intègre des résistances de rappel (pull-down) ou de tirage (pull-up) qui peuvent être activées sur les broches du GPIO. Il n'est donc pas nécessaire d'ajouter ce composant pour le bouton d'appel.

La solution consiste à connecter chaque cathode, non pas à la masse, mais à une sortie numérique distincte du GPIO, et à alimenter le composant en connectant l'anode au +3,3 volts. Lorsque la sortie numérique est à l'état haut (+3,3 volts), aucun courant ne circule, et la couleur correspondante est éteinte. Lorsque la même sortie numérique est à l'état bas (0 volt), le courant circule et la couleur est allumée. La logique habituelle est donc inversée, l'état bas allumant la led, et l'état haut l'éteignant. Selon l'état de chacune des deux sorties numériques, quatre états sont donc possibles :

Sortie 1 (cathode couleur rouge)	Sortie 2 (cathode couleur verte)	État led
Haut	Haut	Éteinte
Bas	Haut	Rouge
Haut	Bas	Verte
Bas	Bas	Orange

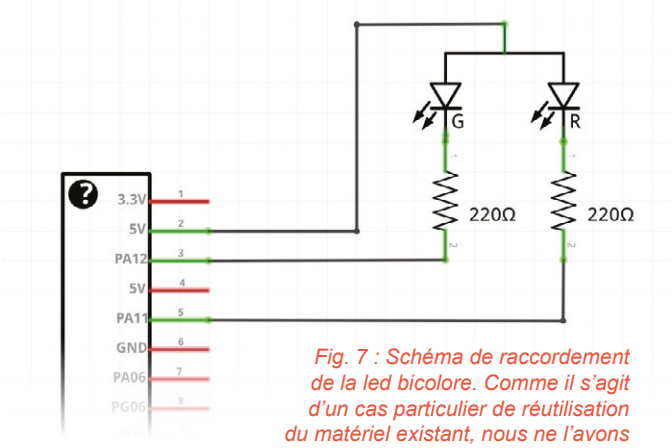


Fig. 7 : Schéma de raccordement de la led bicolore. Comme il s'agit d'un cas particulier de réutilisation du matériel existant, nous ne l'avons pas présenté dans le schéma général présenté plus haut.

3.4 Relais de commande d'ouverture

La commande d'ouverture/fermeture du portail ou du portillon nécessite l'utilisation d'un relais par commande. La partie commande du relais sera reliée à une broche numérique du GPIO, au +5V et à la masse. Le circuit de puissance sera lui connecté au circuit de commande de l'ouvrant (portail ou portillon), lequel doit posséder deux bornes destinées à commander l'ouverture et la fermeture par fermeture du contact pendant une durée d'environ 1 s. Ces deux bornes sont prévues généralement pour connecter une serrure ou un poussoir.

4. PASSONS À LA RÉALISATION PRATIQUE

4.1 Montage sur platine à essai

Avant de passer à la réalisation finale, nous vous recommandons de tester soigneusement l'ensemble de la solution à l'aide d'une platine à essai. Assurez-vous que vous disposez de tous les composants nécessaires, et servez-vous du schéma présenté plus haut (figure 4) pour réaliser le montage.

Une fois le montage effectué, reportez-vous plus loin dans ce numéro à l'article consacré à la partie logicielle et configuration de l'interphone, et vérifiez le bon fonctionnement de votre prototype. Ceci fait, il sera tant de sortir fer à souder, lime et perceuse pour passer à la phase ultime.

4.2 Installation finale

L'interphone était fourni d'origine avec un coffret d'alimentation optionnel (l'appareil pouvant également être utilisé sur piles sans ce coffret). Après avoir dans un premier temps envisagé l'installation de l'Orange Pi Zero et de l'ampli dans la platine de rue, j'ai finalement choisi de n'y laisser que le strict nécessaire (le micro, son circuit d'alimentation et le haut-parleur, cette partie constituant mon « module de commande ») et de détourner le coffret de sa destination originale, pour y déporter l'Orange Pi Zero et l'ampli (le « module de service » !) :

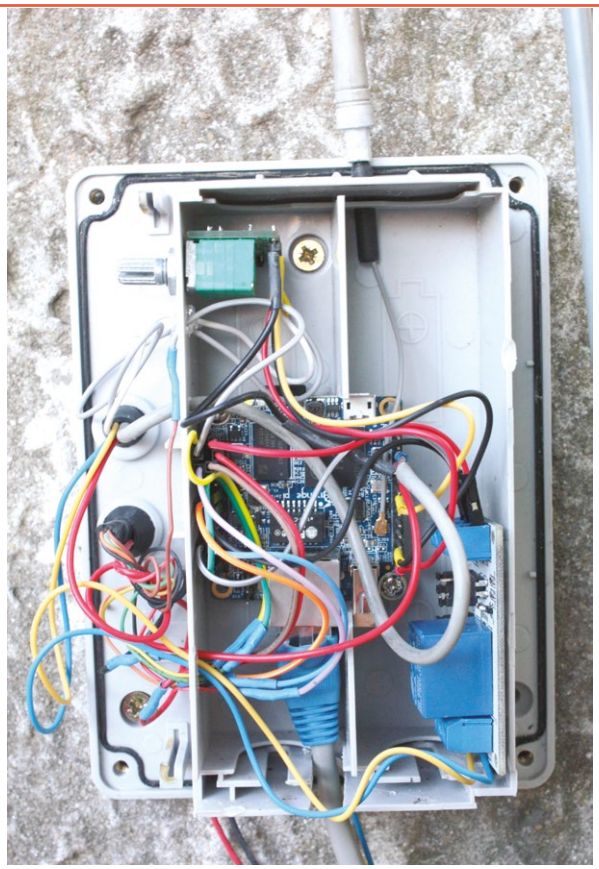


Fig. 8 : Vue de l'intérieur du « module de service ». On remarque l'ampli PAM8403 en haut, fixé à l'aide de son potentiomètre, le relais en bas à droite, et le cabochon de l'antenne wifi en haut du boîtier, qui à l'origine équipait la platine de rue.

convenables pour devenir un jour futur le point de départ d'une station météo connectée, d'autant qu'il recèle de suffisamment d'espace libre pour y installer confortablement capteur de pression, d'humidité et de température !

Bien sûr une situation autre pourrait amener à une réalisation différente. Néanmoins, voici comment se présente l'interphone connecté à notre domicile :

- Fixé au côté extérieur du pilier, le module de commande, contenant le micro électret, son circuit d'alimentation, le haut-parleur, les boutons et les leds.
- Pour des raisons pratiques : malgré sa petite taille, l'intégration de l'Orange Pi Zero dans ce volume contraint prévu pour un autre usage s'est avérée difficile à réaliser, notamment à cause du connecteur RJ45 et de la nécessité d'y faire arriver le câble réseau (tout le câblage devant arriver par l'arrière du boîtier). Le boîtier d'alimentation est bien plus pratique à cet égard : dimensions confortables, joint d'étanchéité, et très facilement aménageable au cutter, à la perceuse ou au Dremel.
- Pour des raisons de dissipation thermique : l'entrée de la maison est orientée plein sud ; il était donc à craindre que l'exposition de la platine bourrée d'électronique au soleil d'été provençal, en l'absence quasi totale d'aération, ne nuise au bon fonctionnement de l'ensemble.
- Pour des raisons de sécurité, il a semblé plus sûr de ne laisser que le strict nécessaire côté rue.
- Enfin, cerise sur le gâteau, le module de service étant installé à l'intérieur, donc sur le côté nord du pilier, à environ 1,60 m de hauteur, il remplit des conditions plus que

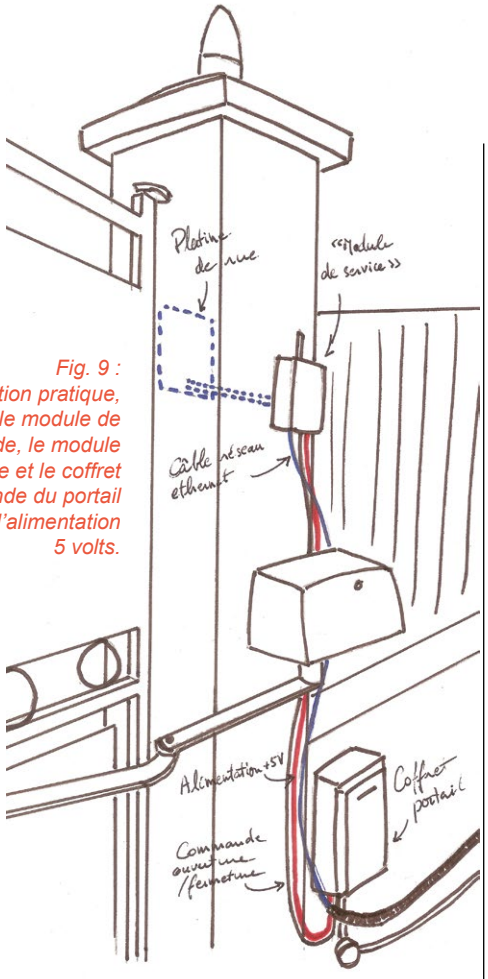
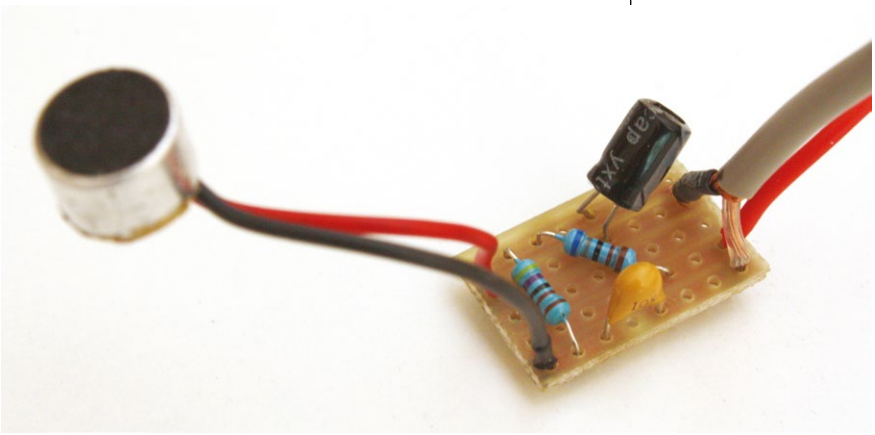


Fig. 9 : La réalisation pratique, avec le module de commande, le module de service et le coffret de commande du portail contenant l'alimentation 5 volts.

Fig. 10 : Le circuit d'alimentation du micro monté sur veroboard. Il faut faire attention à la polarité du micro lors du raccordement. Sur notre micro récupéré, le fil rouge (relié au boîtier) est la masse, et le fil noir l'alimentation et le signal !



• Sur la face intérieure du pilier, en vis-à-vis du module de commande, le module de service, débarrassé de son ancien contenu, et logeant confortablement l'Orange Pi Zero, l'amplificateur PAM8403, ainsi que le relais de commande d'ouverture du portail. Le coffret est connecté au module de commande au moyen d'un câblage traversant le pilier. Du coffret partent également : le câble réseau, qui se raccorde à son autre extrémité au *switch* à l'intérieur de la maison, ainsi que quatre câbles partant vers le coffret électrique du portail, installé à proximité du pilier. Deux câbles relient les bornes 5 volts et masse du GPIO à une alimentation 5 volts qui a été installée à l'intérieur dudit coffret, les deux autres câbles connectent la partie « puissance » du relais au bornier de la carte de commande du portail.

Les figures suivantes (figures 9 et 10) illustrent la réalisation finale.

Lors de la phase de réalisation finale, il est conseillé de souder les fils de connexion aux broches, et de protéger la soudure sous une longueur de gaine thermorétractable. Ceci permettra de fiabiliser l'appareil en sécurisant les connexions dans la durée, et d'éviter de possibles problèmes de faux contacts dus notamment à la corrosion. Assurez-vous de la fiabilité des connexions et de l'absence de courts-circuits, en vous servant au besoin d'un multimètre.

Voici pour la partie matérielle de l'interphone. Il ne nous reste plus qu'à nous pencher sur le système et sa configuration. Ces points sont l'objet de l'article qui suit. **ELB**

RÉFÉRENCES

[1] http://pjsip.org/

[2] https://linux-sunxi.org/Orange_Pi_Zero

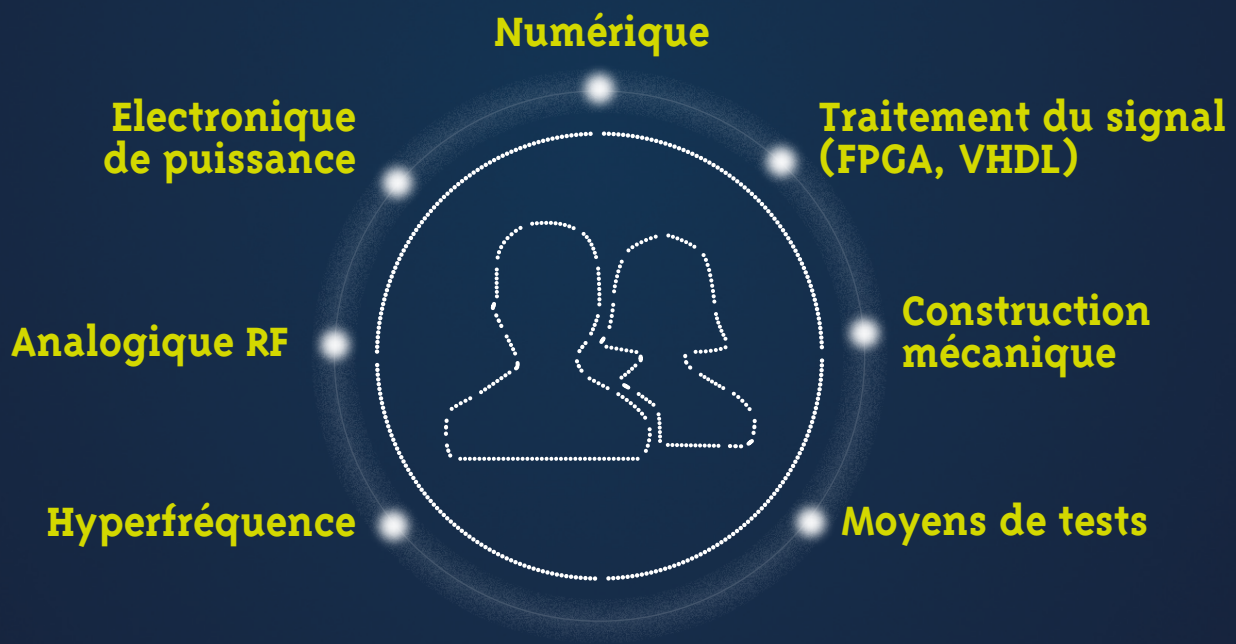
[3] https://www.sonelec-musique.com/electronique_bases_alim_micro_electret.html

[4] http://www.orangepi.org/

TOGETHER WE

EXPLORE A WORLD OF POSSIBILITIES*

À 5 MINUTES DE L'OCÉAN ET EN PLEIN CŒUR D'UN PÔLE D'EXCELLENCE TECHNOLOGIQUE, LE SITE DE **BREST RECRUTE** DANS LES MÉTIERS DE **L'INGÉNIERIE MATÉRIEL.**



Envoyez votre CV à thales-brest-recrute@fr.thalesgroup.com et rejoignez-nous !

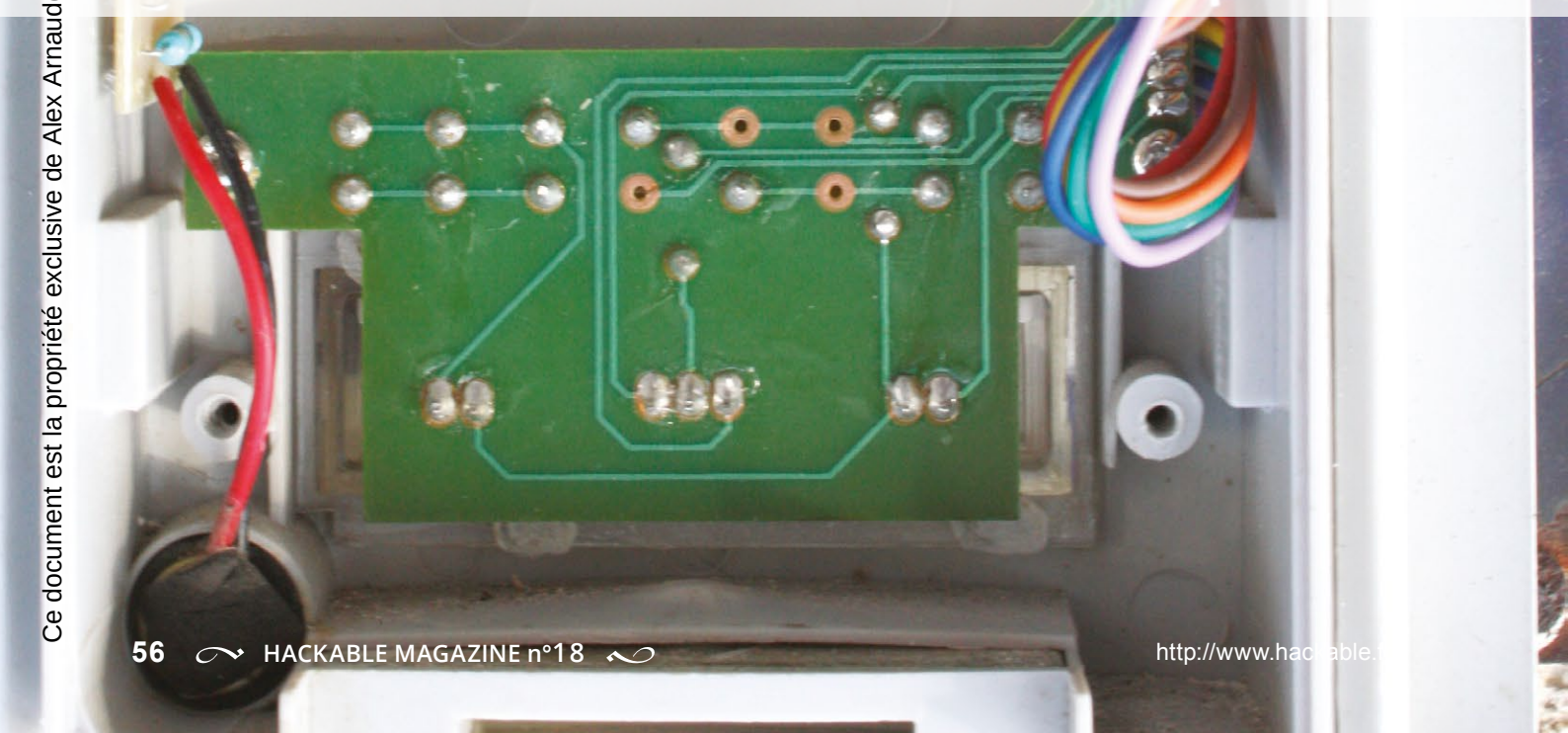
* ENSEMBLE NOUS EXPLORONS UN MONDE DE POSSIBILITÉS

THALES

PORTIER AUDIO CONNECTÉ : INSTALLATION ET CONFIGURATION

Éric Le Bras

Après avoir décrit l'architecture matérielle du portier connecté dans l'article précédent, il est maintenant temps de nous pencher sur son fonctionnement logiciel. Nous allons aborder les composants qui permettent de faire fonctionner le matériel et de gérer la partie communication, l'installation du système, la programmation et la configuration de l'interphone.



Orange Pi est une gamme de cartes de développement open source proposée par le fabricant chinois Xunlong **[3]**. À l'instar des Raspberry Pi, l'Orange Pi Zero nécessite l'installation d'un système d'exploitation sur une carte micro SD. Xunlong, via son site **www.orangepi.org**, propose une section téléchargement avec plusieurs distributions pour chacune de ses cartes. Pour la Pi Zero, pas moins de 5 distributions sont proposées, parmi lesquelles Ubuntu, Debian (serveur ou LXDE) ou Android. Malheureusement, aucun des liens proposés ne m'a permis de récupérer une des images. Soit le lien est mort, soit il amène à une page en chinois. Heureusement, il existe une distribution dédiée aux cartes de développement à base d'ARM, baptisée Armbian **[4]**, qui supporte parfaitement la gamme Orange Pi, et qui jouit d'une bonne réputation dans les forums spécialisés. Une image pour l'Orange Pi Zero étant disponible au moment où je recevais ma carte, c'est donc vers cette distribution que je me suis tourné. Armbian est basée sur Debian Jessie. Utilisateur de Raspbian, vous serez en terrain parfaitement balisé.

Le fonctionnement du portier s'articule autour de plusieurs composants logiciels, dont la figure 1 montre l'organisation.

- **Gestionnaire portier SIP** - C'est le programme Python gérant les fonctionnalités du portier. Le code source est listé et expliqué plus loin dans cet article.

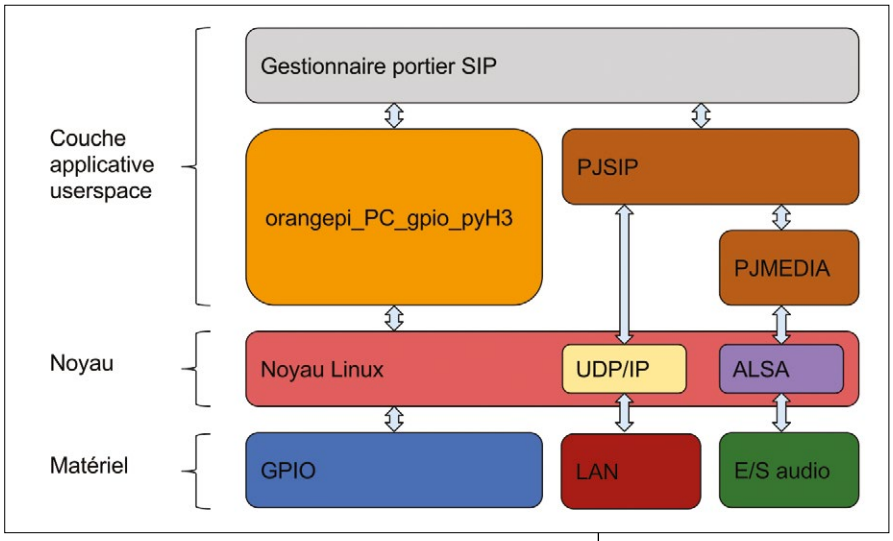


Fig. 1 : Le portier fait intervenir plusieurs couches logicielles, destinées à commander la partie hardware (GPIO, réseau et E/S audio).

- **orangepi_PC_gpio_pyH3 [5]** - Il s'agit d'une librairie Python homologue à la librairie RPi.GPIO, permettant l'accès aux broches du GPIO du *SoC* Allwinner H3. Il faut savoir que le H2+ qui équipe l'Orange Pi Zero est simplement un H3 moins la sortie HDMI 4K et le l'Ethernet Gigabit. Moyennant quoi, le H2+ est totalement compatible avec le H3. C'est d'ailleurs la *datasheet* de ce dernier qui sert de référence pour le H2+ **[2]**. La librairie a été développée pour la carte Orange Pi PC, toutefois on trouve sur GitHub un fork prenant correctement en charge l'Orange Pi Zero. C'est bien sûr cette version que nous utilisons.
- **PJSIP/PJMEDIA [1]** - Ces librairies fournissent le support des protocoles SIP et RTP (via une API nommée `pjsua` pour laquelle un module Python est disponible), ainsi que la gestion des médiums (audio seul dans notre cas, mais les flux vidéos peuvent également être pris en charge).
- **UDP/IP** - Le protocole réseau non connecté, utilisé pour communiquer avec le *trunk* SIP.
- **ALSA** - ALSA est une architecture de gestion des périphériques audios pour Linux. Il s'agit du standard en ce domaine, et la distribution Armbian gère parfaitement cet aspect sans nécessiter de configuration supplémentaire. La couche média de PJSIP s'appuie sur les API exposées par ALSA.

1. INSTALLATION DE L'ORANGE PI ZERO

Vous aurez besoin d'une carte micro SD de 2 Go ou plus. La procédure pour récupérer et installer Armbian sur l'Orange Pi Zero est la suivante :

- Télécharger l'image en se rendant dans la section ***Download*** du site Armbian, onglet ***Server***. Cliquer sur le lien ***Debian Jessie***. L'ensemble des images Armbian est accessible sous : **https://image.armbian.com/**. Pour notre projet, il faut télécharger cette image : **https://image.armbian.com/Armbian_5.24_Orangepizero_Debian_jessie_3.4.113.7z**
- Décompresser l'archive (avec 7-Zip sous Windows ou 7z sous Linux).
- Graver l'image sur la carte micro SD avec Etcher **[6]** (ou **dd**, ou encore *Win32DiskImager*).
- Insérer la carte micro SD dans le lecteur de l'Orange Pi Zero.
- Connecter la carte au réseau local à l'aide d'un câble RJ45. La carte de possédant pas de port HDMI, vous ne disposerez au démarrage que du réseau Ethernet, ou la console série disponible (non expliquée dans cet article) pour communiquer. Le wifi ne sera utilisable qu'après configuration, donc pas au premier démarrage.
- Alimenter la carte par le port micro USB via un PC ou un chargeur de smartphone/tablette. La carte démarre.

Le premier démarrage est assez long (la distribution effectuant des opérations d'initialisation, dont la création d'un fichier de swap et l'extension automatique de la partition pour utiliser la capacité totale de la carte SD). Lors du démarrage, la carte va se voir attribuer une adresse réseau par le serveur DHCP de votre routeur Internet ou de votre box. Pour pouvoir faire quelque chose avec, vous devez impérativement trouver l'adresse réseau qui lui a été attribuée en consultant la liste des appareils connectés sur votre routeur, ou bien en utilisant le programme Unix **nmap** depuis une autre machine. Vous pourriez aussi vous inspirer de l'article *Un afficheur LCD pour connaître l'adresse d'un Raspberry Pi* publié dans le numéro 15 de *Hackable*. Dès que possible, nous vous recommandons d'attribuer une adresse fixe à la carte, soit en configurant la carte en IP fixe, soit, si votre routeur ou votre box le permet, en affectant une IP à l'adresse MAC.

Une fois l'adresse récupérée, l'ouverture d'une session sur la carte se fait via **ssh**, soit depuis un PC Windows au moyen de *PuTTY*, *Cygwin* ou *Ubuntu bash pour Windows 10*, ou depuis un hôte Linux ou un Mac au moyen de la commande **ssh**. Le principe est exactement le même que pour une carte Raspberry Pi, au besoin les articles déjà publiés dans *Hackable* sur le sujet pourront vous aider.

La première connexion doit utiliser le compte *root*, dont le mot de passe par défaut est 1234. La première session déclenche l'exécution automatique d'un script vous demandant de personnaliser ce mot de passe, puis de fournir les informations nécessaires à la création d'un compte d'utilisateur. Lors de vos prochaines sessions, c'est ce compte que vous utiliserez pour vous connecter, et non le compte *root*. Vous pourrez exécuter des commandes nécessitant une élévation de privilèges depuis ce compte, en utilisant la commande **sudo**, ainsi qu'il vous est recommandé lors de cette initialisation.

Comme le script d'initialisation vous y enjoint, déconnectez-vous (Ctrl-D), puis reconnectez-vous avec votre nouveau login.

Mise à niveau du cache **apt-get** :

```
% sudo apt-get update
```

Installation des dernières versions des packages installés :

```
% sudo apt-get dist-upgrade
```

Configuration du fuseau horaire :

```
% sudo dpkg-reconfigure tzdata
```

Si le wifi doit être utilisé, il peut être activé avec cette commande :

```
% sudo nmtui-connect SSID
```

La commande suivante permet de sélectionner le SSID parmi les réseaux détectés :

```
% sudo nmtui-connect
```

Les paramètres wifi sont enregistrés et survivent aux redémarrages. Une mise en garde toutefois : il n'est pas recommandé de rester dans cette configuration si vous comptez exploiter l'interface RJ45. En effet, les interfaces réseau filaires et sans-fils étant toutes les deux activées, se pose un problème de routage : Linux doit choisir entre l'une ou l'autre des deux interfaces actives pour router les paquets sortants. Si c'est l'interface filaire qui est choisie : très bien, c'est probablement ce que vous souhaitez... Mais dans le cas contraire, vos paquets réseau vont voyager par wifi, alors qu'un câble réseau est connecté ! Pire : si la connectivité wifi est mauvaise, il se peut même que vous rencontriez des difficultés à ouvrir une session **ssh**... Vous souhaiteriez que l'interface filaire (**eth0**) soit utilisée par défaut, et l'interface sans fil (**wlan0**) uniquement lorsque la première n'est pas disponible ? Ce n'est pas comme cela que fonctionne le routage par défaut. Notez qu'une telle configuration est possible, mais on sort du cadre de cet article.

Donc, si vous utilisez une connexion réseau RJ45, nous vous recommandons fortement de désactiver le wifi ainsi :

```
% sudo nmcli radio all off
```

...voire de désactiver complètement le démon **NetworkManager**, qui ne gère pas l'interface RJ45 :

```
% sudo systemctl stop NetworkManager
% sudo systemctl disable NetworkManager
```

Votre Pi vous dit merci : vous venez d'économiser le démarrage de 8 process, et de libérer 4 Mo de RAM (et l'interface wifi peut toujours être gérée, comme c'est le cas pour l'interface Ethernet, dans le bon vieux fichier de configuration **/etc/network/interfaces**).

Nous avons maintenant une carte Orange Pi Zero fonctionnelle, avec Armbian installé et configuré. Il est temps de passer à la suite du projet.

2. RÉCUPÉRATION ET COMPILATION DES LIBRAIRIES

Le programme Python qui va gérer l'interphone repose sur plusieurs composants supplémentaires que nous devons au préalable installer. La librairie **orangepi_PC_gpio_pyH3**, comme vue plus haut, permet de manipuler et lire les broches du GPIO. **PJSIP** prend en charge les fonctionnalités de téléphonie (communications et médias). En plus de ces composants, nous aurons besoin des paquets suivants :

- **libasound2-dev** : fichiers de définition pour la librairie ALSA (gestion des périphériques audios).
- **libssl-dev** : fichiers de définition pour la librairie OpenSSL prenant en charge les connexions cryptées SSL/TLS, requises par **PJSIP**. Ce paquet est déjà installé de base avec Armbian.
- **python-dev** : fichiers de définition et outils de développement pour les modules Python. Ce paquet est nécessaire pour compiler **orangepi_PC_gpio_pyH3** et **PJSIP**.

Ces paquets sont installés simplement via **apt-get** :

```
% sudo apt-get install python-dev libasound2-dev
```

Récupération et installation de la librairie **orangepi_PC_gpio_pyH3** :

```
% git clone https://github.com/nvl1109/orangepi_PC_gpio_pyH3.git
% cd orangepi_PC_gpio_pyH3
% sudo python ./setup.py install
```

Récupération, compilation et installation de **PJSIP** :

```
% cd ~
% wget http://www.pjsip.org/release/2.5.5/pjproject-2.5.5.tar.bz2
% tar xjf pjproject-2.5.5.tar.bz2
% cd pjproject-2.5.5
% ./configure --enable-shared
% make dep && make
% sudo make install
```

Installation de l'API Python :

```
% cd pjsip-apps/src/python
% sudo python ./setup.py install
```

Mise à jour du cache des librairies dynamiques :

```
% sudo ldconfig
```

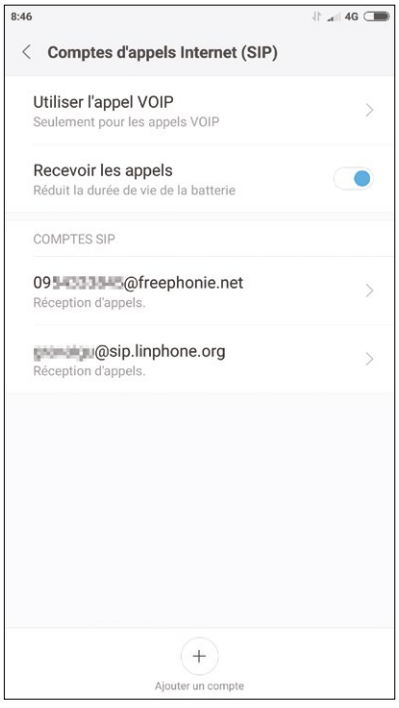


Fig. 2 : La configuration des appels VOIP sur un smartphone sous MIUI8 (Lollipop). Cet écran est accessible via l'application Téléphone, menu Paramètres → Paramètres avancés → Appel VOIP.

2.1 Testons PJSIP

Si vous avez choisi d'utiliser le *trunk* SIP de Free et activé ce service, vous pouvez dès à présent tester votre installation. Si vous utilisez un *trunk* gratuit, il vous faut commencer par configurer votre smartphone pour recevoir les appels SIP. Sous Android, le protocole SIP est déjà présent, il ne reste qu'à le configurer (il est possible, mais pas nécessaire d'installer une application tierce). Pour cela, il vous faut trouver les paramètres VoIP. L'emplacement dépend du constructeur et de la version Android. À titre d'exemple sur mon appareil, le menu se trouve dans l'application ***Téléphone***, sous ***Paramètres***, ***Paramètres avancés***, ***Appel VOIP***. Une fois le menu trouvé, il nous faut activer l'option ***Recevoir les appels***, et ajouter un compte SIP. Les paramètres nécessaires (*nom d'utilisateur*, *mot de passe* et *serveur*) vous ont été fournis lors de la création de votre compte SIP.

Vous pourrez ensuite tester l'installation de PJSIP à l'aide de la commande **pjsua**.

```
% cd ~/pjproject-2.5.5/pjsip-apps/bin
```

Si vous avez activé le service SIP de Free, exécutez cette commande :

```
% ./pjsua-armv7l-unknown-linux-gnueabihf --id sip:09xxxxxxxx@freephonie.net
--registrar sip:freephonie.net --realm "*" --username 09xxxxxxxx --password
yyyyyyyy --reg-timeout 1800 --no-tcp
```

Si vous avez opté pour un compte **Linphone.org**, les paramètres à utiliser sont :

```
% ./pjsua-armv7l-unknown-linux-gnueabihf --id sip:nomutil@sip.linphone.org
-registrar sip:sip.linphone.org --realm "*" --username nomutil --password
yyyyyyyy
```

Les messages échangés entre PJSIP et le *registrar* défilent à l'écran. Appuyez ensuite sur la touche *Entrée* : un menu apparaît, précédé de la liste des comptes. Vérifiez que vous lisez la ligne (l'URI correspond au paramètre **--id** que vous avez entré) :

```
 *[ 1] sip:nomutil@serveursip: 200/OK (expires=1720)
       Online status : Online
```

Si vous voyez le message ci-dessus, vous êtes connecté au *registrar*. Tapez ensuite **m** puis *Entrée*.

Après le prompt *Make call:*, entrez l'URI SIP de votre ligne (en rouge ci-dessous). Attention : le numéro de téléphone à entrer est votre numéro géographique, commençant par 01, 02, 03, 04 ou 05, et non le numéro en 09 fourni par Free :

```
Make call:  sip:04zzzzzzzz@freephonie.net
```

Si vous avec un compte **Linphone.org**, ou pour tester l'appel VoIP sur votre smartphone :

```
Make call:  sip:nomutil@sip.linphone.org
```

Si tout se passe bien, votre téléphone sonne, vous pouvez décrocher ! Appuyez sur les touches du téléphone : si la détection des codes DTMF fonctionne, des messages apparaissent sur la console, indiquant la touche détectée. Si vous ne voyez pas de message, c'est probablement que vous avez appelé votre ligne fixe depuis le service SIP de Free. Vous ne pourrez dans ce cas pas utiliser cette fonctionnalité pour commander l'ouverture (voir plus haut le § « À propos du trunk SIP »).

3. LE PROGRAMME DE GESTION DU PORTIER

La gestion du portier est confiée à un programme Python chargé au démarrage sous forme de service. Le programme gère tous les aspects fonctionnels du portier :

- gestion du bouton d'appel et des leds ;
- gestion de la communication SIP ;
- déclenchement de l'ouverture du portail.

Le programme Python est intégré sous la forme d'un service **systemd**. Ce sera l'occasion d'introduire le système d'amorçage par défaut de Debian Jessie. Pour terminer l'installation de

l'interphone, vous devrez récupérer sur le GitHub du magazine et installer les trois fichiers suivants (expliqués en détail dans les paragraphes ci-dessous) :

- `/usr/local/bin/doorphone.py`
- `/lib/systemd/system/doorphone.service`
- `/etc/default/doorphone`.

3.1 Le fonctionnement du programme

L'interphone est géré par le programme Python `/usr/local/bin/doorphone.py`.

Le programme est destiné à être démarré en tant que service par `systemd` (voir § suivant). Tout ce qui est envoyé par le programme sur la sortie standard par l'instruction `print` sera redirigé vers le journal par `systemd`.

Les modules nécessaires sont importés par ces instructions :

```
import os
import sys
import pjsua as pj
import threading
import signal
import datetime
from pyA20.gpio import gpio
from pyA20.gpio import port
from time import sleep
```

Pour pouvoir accéder au GPIO, le programme doit être exécuté par root. Les lignes suivantes, placées au début du programme, effectuent cette vérification :

```
import os
if not os.getegid() == 0:
    sys.exit('Le script doit être démarré par root')
```

Les ports GPIO utilisés pour gérer les boutons, leds et relais sont déclarés dans des variables :

```
led_yellow = port.PA7
led_red = port.PA11
led_green = port.PA12
relay_0 = port.PA15
relay_1 = port.PA16
call_button = port.PA13
```

Le GPIO est initialisé par le code suivant :

```
# Initialisation du module GPIO
gpio.init()

# Configuration des ports
gpio.setcfg(led_yellow, gpio.OUTPUT)
gpio.setcfg(led_red, gpio.OUTPUT)
gpio.setcfg(led_green, gpio.OUTPUT)
```

```
gpio.setcfg(relay_0, gpio.OUTPUT)
gpio.setcfg(relay_1, gpio.OUTPUT)
gpio.setcfg(call_button, gpio.INPUT)

# Activation de la resistance de tirage (pull-up) pour le bouton
gpio.pullup(call_button, gpio.PULLUP)

# Extinction de la led bicolore (mise à l'état haut, voir l'explication dans
l'article)
gpio.output(led_red, gpio.HIGH)
gpio.output(led_green, gpio.HIGH)

# Eclairage de la carte de visite
gpio.output(led_yellow, gpio.HIGH)

# Initialisation des relais
gpio.output(relay_0, gpio.LOW)
gpio.output(relay_1, gpio.LOW)
```

Notez que la led bicolore (**led_red** et **led_green**), initialisée ici, n'est pas exploitée dans le reste du programme. Son utilisation reste à définir : ouverture du portail ? décrochage du téléphone ? tests, détection d'anomalies ? Il suffit d'ajouter les appels à **gpio.output()** aux bons endroits.

Le code suivant initialise la librairie PJSIP[1], puis crée les objets nécessaires :

- Objet **transport** de type **Transport()** renvoyé par la méthode **create_transport()**.
- Objet **acc_cfg** de type **AccountConfig()**, contenant les informations de connexion au *trunk*. Les informations sont récupérées depuis les variables d'environnement par la méthode **os.getenv()**. Le second argument de cette méthode, si présent, est la valeur par défaut à utiliser en l'absence de la variable d'environnement. Les variables d'environnement sont définies dans le fichier **/etc/default/doorphone** (voir plus loin).

Le paramètre **log_cfg** passé à la méthode **init()**permet de définir un niveau de log, et de fournir la fonction à utiliser pour récupérer les informations tracées par la librairie :

```
LOG_LEVEL=3

# Callback pour le logging de PJSIP
def log_cb(level, str, len):
    print str,
(...)
# Créons l'instance de la librairie
lib = pj.Lib()

try:
    # Initialisation de la librairie PJSIP
    lib.init(log_cfg = pj.LogConfig(level=LOG_LEVEL,
callback=log_cb))

    # Création d'un transport UDP
    transport = lib.create_transport(pj.TransportType.UDP)
```

[1] *L'API Python PJSUA qui est utilisée est documentée sur le site du projet PJSIP à la page http://www.pjsip.org/python/pjsua.htm.*

```
    print "\nListening on", transport.info().host,
    print "port", transport.info().port, "\n"

    # Démarrage de la librairie PJSIP
    lib.start()

    # Création de la configuration du compte
    acc_cfg = pj.AccountConfig()
    acc_cfg.id = os.getenv('SIP_ID')
    acc_cfg.reg_uri = os.getenv('SIP_REGISTRAR')
    acc_cfg.auth_cred = [ pj.AuthCred(os.getenv('SIP_REALM', '*'), os.getenv('SIP_
USERNAME'), os.getenv('SIP_PASSWORD')) ]
    acc_cfg.reg_timeout = int(os.getenv('SIP_REG_TIMEOUT', '300'))

except pj.Error, e:
    print "Exception: " + str(e)
    gpio.output(led_yellow, gpio.LOW)
    lib.destroy()
    lib = None
    sys.exit(1)
```

Le principe de fonctionnement du gestionnaire de portier repose sur la boucle suivante :

- Lecture de l'état du bouton d'appel.
- Si l'état est à 0 (bouton enfoncé), et que l'état précédent était à 1, appel de la fonction **`call_button_handler()`** (voir plus loin).
- Si un appel est en cours, on calcule la durée de l'appel (différence entre l'heure courante et la date de début de l'appel lue dans la variable **`call_start`**) et on la compare à la valeur de **`call_timeout`**. Si la durée de l'appel dépasse la valeur configurée (120 secondes par défaut), on raccroche en appelant la méthode **`call.hangup()`**.
- L'objet **`GracefulKiller()`** permet de détecter la réception d'un signal SIGINT ou SIGTERM (ce dernier est envoyé au process par **`systemd`** lorsqu'il doit être arrêté, par exemple suite à une commande **`systemctl stop`** – voir plus loin). Si la variable **`killer.kill_now`** contient la valeur **`True`**, on sort de la boucle, et on exécute les instructions d'arrêt du service avant de quitter. La classe **`GracefulKiller()`** permet de gérer proprement la réception d'un signal d'arrêt. Elle n'est pas listée ici, mais se trouve dans le source complet.
- Dans le cas contraire, on attend 0,2 seconde puis on réitère au début de la boucle.

```
call_start = None
call_timeout = datetime.timedelta(seconds=int(os.getenv('CALL_TIMEOUT', '120')))
call_nb = 0
bt_prev_state = 1
killer = GracefulKiller()
while True:
    bt_state = gpio.input(call_button)
    if bt_state != bt_prev_state:
        bt_prev_state = bt_state
        if bt_state == 0:  # Le bouton a été enfoncé
            # Effectuer l'appel
```

```
            call_button_handler()
    # Détection du timeout
    if call_start <> None and (datetime.datetime.today() - call_start) > call_timeout:
        call.hangup()
    # SIGINT ou SIGTERM reçu
    if killer.kill_now:
        break  # Sortir de la boucle
    sleep(0.2)

# Arrêt du service
gpio.output(led_yellow, gpio.LOW)  # Extinction des feux
lib.destroy()
lib = None
print "Doorphone shutdown"
sys.exit(0)
```

L'appel est effectué par la fonction **call_button_handler()**. Il se décompose en deux étapes successives :

1. Création du compte. Cette opération effectue l'enregistrement auprès du *registrar*.

2. Création de l'appel.

Dans PJSIP, toutes les opérations qui ont à voir avec l'envoi et la réception de messages, tels que l'enregistrement auprès du *registrar* et la gestion d'un appel, sont asynchrones. Cela signifie que la méthode qui effectue l'opération se termine immédiatement, et que les différents états sont ensuite renvoyés sous la forme de *callbacks*.

C'est la raison pour laquelle, après l'appel de **create_account()**, la méthode **acc_cb.wait()** est appelée, afin que le **make_call()** qui suit ne soit appelé qu'après que l'objet **MyAccountCallback()**ait reçu du serveur la confirmation de l'enregistrement. Dans le cas contraire, le *trunk* refuserait d'effectuer l'appel.

```
def call_button_handler():
    global lib
    global acc
    global call
    global call_nb
    if call_nb == 0:  # Si pas d'appel en cours
        call_nb += 1
        acc_cb = MyAccountCallback()
        acc = lib.create_account(acc_cfg, cb=acc_cb)
        acc_cb.wait()
        # Call user
        call = acc.make_call(os.getenv('SIP_DEST_URI'), MyCallCallback())
```

Les *callbacks* sont des objets, dans lesquels des méthodes sont définies, dans le but de recevoir les notifications d'événements intéressant l'objet principal (le compte, l'appel). Ainsi, dans le gestionnaire de portier, nous avons défini deux *callbacks* : **MyAccountCallback()**, du type **AccountCallback()**, pour les évènements liés au compte, et **MyCallCallback()**, du type **CallCallback()**, pour ceux liés à l'appel.

```
# Callback pour recevoir les évènements du compte
class MyAccountCallback(pj.AccountCallback):
    sem = None

    def __init__(self, account=None):
        pj.AccountCallback.__init__(self, account)

    def wait(self):
        self.sem = threading.Semaphore(0)
        self.sem.acquire()

    def on_reg_state(self):
        if self.sem:
            if self.account.info().reg_status >= 200:
                self.sem.release()
```

- **wait** : Appelée par **call_button_handler()**. Créé un sémaphore initialisé à la valeur 0. L'appel de la méthode **acquire()** est bloquant parce que le sémaphore est à zéro, et sera débloqué par un appel à la méthode **release()** effectué à réception du statut 200 OK (ci-dessous).
- **on_reg_state** : Cette méthode est appelée par l'objet **Account()** lors d'un changement de statut du compte. Si le statut renvoyé par le *registrar* lu dans **reg_status** est 200 (OK), l'appel de **release()** libère l'appel bloquant **acquire()** invoqué dans la méthode **wait()**. Ceci est possible parce que l'objet **Account()**, créé par l'appel à **create_account()** a créé un *thread* distinct du programme principal qui a appelé la méthode **wait()**, du fait du modèle asynchrone de PJSIP. Les méthodes **wait()** et **on_reg_state()** sont donc appelées dans des *threads* parallèles.

```
# Callback pour recevoir les évènements de l'appel
class MyCallCallback(pj.CallCallback):
    def __init__(self, call=None):
        pj.CallCallback.__init__(self, call)

    # Notification when call state has changed
    def on_state(self):
        global acc
        print "Call is ", self.call.info().state_text,
        print "last code =", self.call.info().last_code,
        print "(" + self.call.info().last_reason + ")"
        if self.call.info().state == pj.CallState.DISCONNECTED:
            acc.delete()
            call_start = None
            call_nb -= 1

    # Notification when call's media state has changed.
    def on_media_state(self):
        global lib
        global call_start
        if self.call.info().media_state == pj.MediaState.ACTIVE:
            # Connect the call to sound device
```

```
            call_slot = self.call.info().conf_slot
            lib.conf_connect(call_slot, 0)
            lib.conf_connect(0, call_slot)
            call_start = datetime.datetime.today()

    def on_dtmf_digit(self, digits):
        global relay_0
        global relay_1
        print "DTMF received, digit=", str(digits)
        if digits == "#":
            relay = relay_0
        elif digits == "*":
            relay = relay_1
        else:
            return
        # Ouverture portail ou gâche : fermer le contact pendant 1 s
        gpio.output(relay, gpio.HIGH)
        sleep(1)
        gpio.output(relay, gpio.LOW)
```

- **on_state** : Méthode appelée par l'objet **Call()** lors d'un changement de statut de l'appel. Ici la méthode est utilisée pour supprimer le compte *acc*, et donc se désinscrire du registrar. Comme nous n'effectuons que des appels sortants, nous n'avons pas besoin de rester en ligne une fois l'appel terminé. La variable **call_start** utilisée pour la gestion du *timeout* est réinitialisée. Le compteur d'appels en cours **call_nb** est décrémenté.
- **on_media_state** : Méthode appelée par l'objet **Call()** lors d'un changement de statut du média de l'appel. Le but de cette méthode est, lorsque le média est activé, de connecter le média au périphérique son (entrée micro et sortie ligne). On en profite pour initialiser la variable globale **call_start** avec l'heure système (cette variable sert à chronométrer l'appel, voir plus haut la gestion du *timeout*).
- **on_dtmf_digit** : Méthode appelée par l'objet **Call()** lorsqu'un chiffre DTMF est détecté (appui sur une touche du téléphone par le destinataire de l'appel). Si la touche # ou * est détectée, on ferme le contact du relais 0 ou 1 pendant une seconde puis on le rouvre, pour déclencher l'ouverture du portail ou de la gâche. Attention : certains relais ont une logique inverse : l'état bas (LOW) correspond au relais activé (fermé), l'état haut (HIGH) correspondant à l'état désactivé (ouvert). Si tel est le cas, vous devrez mettre à jour cette méthode, ainsi que l'initialisation de la broche vue plus haut. Notez que certains modules de relais possèdent des cavaliers permettant de choisir le type de comportement souhaité.

3.2 Intégration avec systemd

Depuis l'arrivée de Debian Jessie en 2015, le système d'amorçage par défaut est devenu **systemd**, en remplacement du vénérable **SysV init**.

Sous **systemd**, un service est configuré sous la forme d'un fichier dont le nom se termine par **.service**. Ces fichiers sont constitués de différentes sections (spécifiées entre crochets), chaque section étant constituée d'un ensemble de paires *clé=valeur*. Cette syntaxe n'est pas sans rappeler celle des fichiers **.ini** de Windows, dont elle est indirectement issue, via les fichiers **.desktop** de **Freedesktop.org**.

Voici le fichier **/lib/systemd/system/doorphone.service** servant à démarrer notre gestionnaire de portier :

```
[Unit]
Description=Doorphone manager
After=multi-user.target

[Service]
EnvironmentFile=/etc/default/doorphone
ExecStart=/usr/bin/python -u /usr/local/bin/doorphone.py
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

La clé **After=** permet de spécifier à quel moment le service doit être démarré. **multi-user.target** est un état cible du système (semblable à un *runlevel* sous **SysV init**). Il sert à activer un certain nombre de services système nécessaires dans un environnement non graphique. En gros, il correspond au *runlevel* 3 de l'ancien système : tous les services système sont démarrés, à l'exception du *login* graphique. Cette clé indique donc que le service de gestion du portier doit être démarré après tous les services dépendants de **multi-user.target**.

La clé **EnvironmentFile=** indique un fichier contenant des définitions de variables d'environnement devant être chargées avant le démarrage du programme. En l'occurrence, le fichier **/etc/default/doorphone** contient les informations de connexion au *registrar*, ainsi que d'autres valeurs paramétrables (voir le contenu du fichier plus loin).

La clé **ExecStart=** est suivie de la commande à exécuter pour démarrer le service. L'argument « -u » passé à l'interpréteur **python** force les entrées/sorties standards à être non buffurisées. Grâce à ce paramètre, tout ce qui dans le programme est affiché sur la sortie standard à l'aide de l'instruction **print** et est *logué* dans le journal de **systemd**, accessible par la commande **journalctl**.

Restart=on-abort signifie que le service sera automatiquement redémarré s'il se termine suite à la réception d'un signal non pris en charge par le programme.

La clé **WantedBy=** permet de spécifier une dépendance. Lorsque le service sera activé au *boot* du système, il sera déclenché par la cible **multi-user.target**. **WantedBy=** n'impliquant pas d'ordre de démarrage, pour que notre service soit démarré après les autres services déclenchés par la même cible, la clé **After=** est nécessaire.

Le fichier **doorphone.service** sera placé dans le répertoire : **/lib/systemd/system/**.

La commande suivante doit ensuite être exécutée pour que le service soit chargé automatiquement à chaque démarrage du système :

```
% sudo systemctl enable doorphone.service
```

Les commandes suivantes sont utiles dans les phases de test. Pour afficher l'état du service :

```
% sudo systemctl status doorphone.service
```

Démarrer manuellement le service :

```
% sudo systemctl start doorphone.service
```

Arrêter manuellement le service :

```
% sudo systemctl stop doorphone.service
```

Désactiver le service (il ne sera plus démarré automatiquement) :

```
% sudo systemctl disable doorphone.service
```

Le contenu du fichier de configuration du gestionnaire **/etc/default/doorphone**, lu au lancement du service :

```
# Ce fichier contient les paramètres du gestionnaire de portier doorphone

# URI du compte local
# Freephonie :
# SIP_ID="sip:0123456789@freephonie.net"
# Linphone :
# SIP_ID="sip:nomutil@sip.linphone.org"
SIP_ID="sip:0123456789@provider.net"

# URI du registrar. Exemple : SIP_REGISTRAR="sip:freephonie.net"
# Freephonie :
# SIP_REGISTRAR="sip:freephonie.net"
# Linphone :
# SIP_REGISTRAR="sip:sip.linphone.org"
SIP_REGISTRAR="sip:provider.net"

# Identification du compte SIP
SIP_USERNAME="0123456789"
SIP_PASSWORD="monmotdepasse"
SIP_REALM="*"

# Intervalle de rafraîchissement. Ce paramètre est nécessaire pour Freephonie.
SIP_REG_TIMEOUT=1800

# URI de l'appelé
# Exemple : numéro de téléphone fixe
# SIP_CALL_URI="sip:111111111@freephonie.net;user=phone"
# Exemple : identifiant SIP
# SIP_CALL_URI="sip:nomutil@sip.linphone.org;transport=udp"

# Durée maximale de l'appel, en secondes. Nécessaire si l'appel aboutit sur la
# messagerie.
# Valeur par défaut 120 secondes
CALL_TIMEOUT=120
```

4. TESTONS LE TOUT

Vous êtes arrivés jusque-là, et avez effectué toutes les étapes de montage et d'installation de l'interphone ? Il est temps maintenant de tester tout cela !

Reliez les broches 5V et GND à l'alimentation : l'Orange Pi Zero démarre. Tout d'abord les leds du port RJ45 s'allument, puis au bout d'une dizaine de secondes, la led de statut de la

carte (d'abord fixe, puis en clignotant). Si la led de statut ne s'allume pas, cela signifie que la carte n'a pas réussi à démarrer le système installé sur la carte microSD.

À la fin du démarrage (une poignée de secondes), la led de l'interphone doit s'allumer, signalant que le service a été correctement démarré. Vous pouvez vous connecter via ssh, et vérifier à l'aide de la commande suivante que tout est OK :

```
% sudo systemctl status doorphone.service
```

Sur la console, vous devez voir vers le bas de l'écran une ligne semblable à celle-ci :

```
Feb 26 13:04:09 opzb python[676]: Listening on 127.0.0.1 port 50468
```

L'interphone est pour le moment en attente.

Enfoncez et relâchez le bouton : cette action doit déclencher un appel vers le destinataire paramétré dans le fichier de configuration. Décrochez votre téléphone et vérifiez que la communication est bien établie.

Une fois en communication, appuyez sur la touche # du téléphone. Le relais doit s'enclencher pendant une seconde. Dans le cas contraire, vérifiez avec la commande précédente que le code DTMF a bien été détecté (cela peut ne pas être le cas si vous utilisez le *trunk* de Free et votre téléphone fixe).

En cas de problème, utilisez la commande précédente, ou vérifiez les messages dans le journal de **systemd** à l'aide de :

```
% sudo journalctl -u doorphone.service
```

5. SI VOUS VOULEZ UTILISER UN RASPBERRY PI

Si malgré nos arguments, vous avez choisi d'utiliser une plateforme Raspberry Pi à la place d'Orange Pi (ou si vous ne disposez pas de ce dernier et souhaitez commencer sans attendre), notre projet peut parfaitement s'adapter à la framboise. Vous devrez néanmoins faire attention aux spécificités suivantes.

Les Raspberry Pi ne disposent pas d'entrée audio. Vous devrez pour cela utiliser un périphérique audio USB compatible. A priori le circuit d'alimentation du micro électret reste utilisable avec cette interface. En l'absence de la broche MIC-BIAS, vous pourrez utiliser une broche +3.3V ou +5V. Notez qu'il existe également une carte audio disposant d'une entrée : la Cirrus Logic Audio Card, mais elle monopolise le GPIO et n'est donc pas utilisable pour notre projet. De plus le coût de cette carte d'excellente facture (en gros le prix d'un Raspberry Pi 3), ne la destine clairement pas à cet usage.

Toujours au chapitre électronique, quelques différences sont à prendre en compte. Vous devrez utiliser le jack pour diriger la sortie audio du Raspberry Pi vers l'amplificateur PAM8403. La technique d'alimentation via les broches du GPIO décrite plus haut reste a priori valable. Par ailleurs, pour tester l'état du bouton, une résistance de rappel (pull-down) ou de tirage (pull-up) devra être ajoutée au montage. On est ici en terrain connu et balisé, ce point ayant déjà été traité dans votre magazine.

Fig. 3 : L'interphone à l'essai.Tous les composants d'un téléphone VoIP sont réunis ici, plus le relais qui vient d'être déclenché (led rouge allumée, et message de détection de la touche « # » dans les messages tracés).

Sous Raspbian, PJSIP peut s'installer sous forme de paquets. Un **`apt-cache search pjproject`** renvoie les paquets existants (en version 2.1.0). Cependant l'API Python – dont nous avons besoin – n'est pas disponible sous cette forme dans la distribution stable actuelle. Elle l'est cependant dans *testing* (en version 2.6.0) Vous devrez donc soit passer en *testing*, soit modifier le fichier **`/etc/apt/preferences`** pour pouvoir installer des paquets de *testing* tout en restant en *stable*. À ce sujet, entrez sur votre moteur de recherche les termes « debian testing pinning », ou lisez tout simplement l'article dédié à la gestion des versions de paquets dans le présent numéro.

Notez que ces paquets sont également accessibles sous Armbian (donc sur Orange Pi Zero). Mais en voulant les utiliser, nous nous sommes heurtés à des problèmes de gestion du périphérique audio, ce qui nous a conduits à revenir à la version 2.5.5, et à utiliser la compilation depuis les sources comme expliqué plus haut dans l'article. En cas de difficulté sur Raspberry Pi, cette méthode serait également utilisable.

Enfin, la librairie de gestion du GPIO **`orangepi_PC_gpio_pyH3`** est spécifique aux SoC AllWinner H3 et H2+. Sur Raspberry Pi, vous devrez utiliser la librairie **`RPi.GPIO`**, déjà abordée dans les colonnes de *Hackable*. Les API étant différentes, le code du programme Python devra être adapté.

Notez que, si rien a priori ne s'oppose à ce portage, nous n'avons pas testé cette solution. N'hésitez pas à nous faire part de vos réalisations. Toutes vos remarques, succès et/ou difficultés dans cette entreprise seront recueillis avec bienveillance.

CONCLUSION

Cette réalisation aura été l'occasion de découvrir une nouvelle carte de développement, d'apprendre à utiliser un micro électret, de nous familiariser avec le protocole SIP et la convergence informatique/téléphonie, de comprendre le fonctionnement de **`systemd`**, tout en fabriquant une solution ouverte, maintenable et évolutive.

À l'usage, j'ai vérifié que cette solution fonctionne de façon fiable depuis son installation, et supporte très bien les contraintes liées à l'embarqué (notamment des coupures de courant relativement fréquentes en cette période de rigueur hivernale et d'utilisation fréquente de gros consommateurs tels chauffages d'appoint, four, clim réversible et autre chauffe-eau... Un motif justifié pour un projet personnel de régulation intelligente me direz-vous !). Non seulement la performance de notre portier connecté est sans comparaison avec l'appareil original, mais encore nous avons vite découvert certains avantages, parmi lesquels la possibilité de dater les utilisations de l'interphone dans le journal des appels du téléphone ou de recevoir des messages.

Plusieurs améliorations et évolutions sont déjà envisagées (certaines pourraient mettre à profit des techniques déjà traitées dans *Hackable*) :

- Notifications (par exemple : envoi d'un SMS ou d'un e-mail lorsque le bouton d'appel a été utilisé).
- Détection de clés RFID, ou de séquences de codes DTMF générées par un smartphone.
- Ajout de la vidéo (le protocole SIP ne se limitant pas à la voix).
- Message vocal d'absence (déjà possible avec le *trunk* de Free).
- Réutilisation du combiné intérieur fourni avec le portier d'origine. Il s'agirait de transformer ce combiné – désormais inutilisable – en téléphone VoIP, en utilisant les mêmes composants que pour le portier.
- Redirections sur non-réponse.
- Autres utilisations : contrôle de l'ouverture du portail et alerte en cas d'ouverture anormale, station météo (voir plus haut), borne wifi pour le jardin, voire (soyons fous) : diffusion de musique pendant la période de Noël ! **ELB**

RÉFÉRENCES

[1] http://pjsip.org/

[2] https://linux-sunxi.org/Orange_Pi_Zero

[3] http://www.orangepi.org/

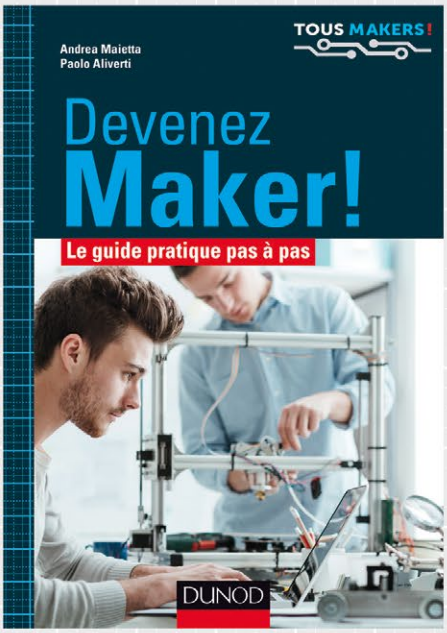
[4] https://www.armbian.com/orange-pi-zero/

[5] https://github.com/nvl1109/orangepi_PC_gpio_pyH3

[6] https://www.etcher.io/

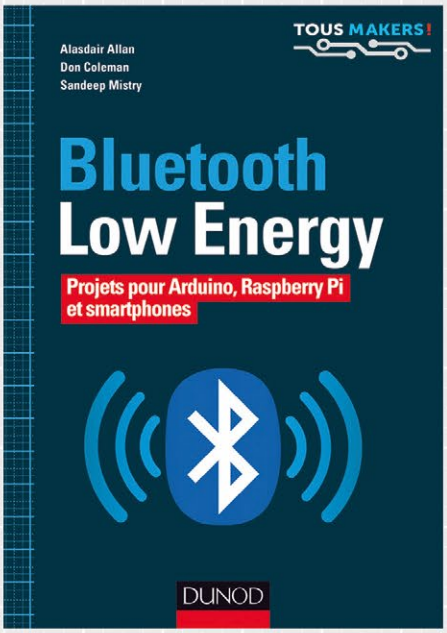
TOUS MAKERS !

RÉVÉLEZ VOTRE POTENTIEL PAR LA CRÉATION



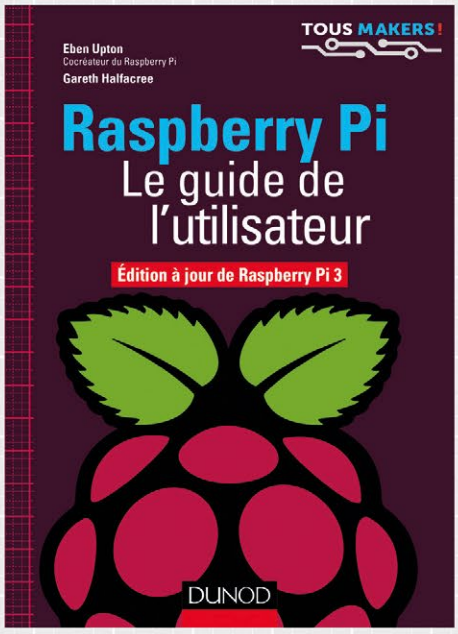
9782100762934, 304 pages, 24,90 €
ANDREA MAIETTA, PAOLO ALIVERTI
Comment transformer vos idées en projets concrets ? Ce livre vous accompagne dans la réalisation de vos premières créations.

9782100758487, 240 pages, 27 €
ALEX ELLIOTT
Le compagnon idéal pour vous guider pas à pas tout au long de la construction de votre drone.

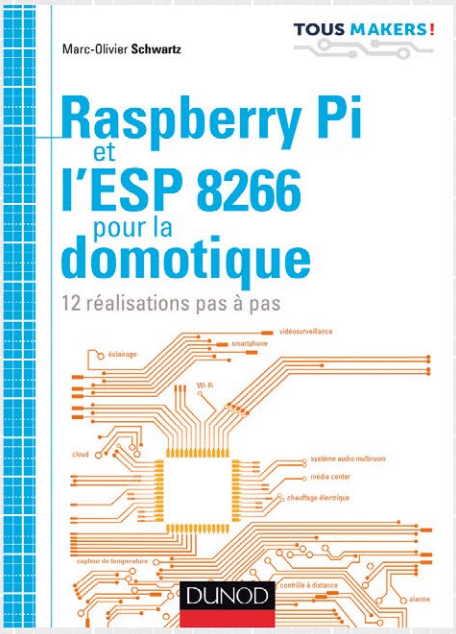


9782100760855, 272 pages, 29 €
ALASDAIR ALLAN ET AL.
Maîtrisez la nouvelle technologie Bluetooth Low Energy en réalisant les différents projets détaillés dans cet ouvrage.

9782100738106, 176 pages, 23 €
PATRICE OGUIC
Un guide pour construire votre machine CNC, idéale pour la gravure et le perçage des circuits imprimés, les maquettistes et les modélistes.



9782100762262, 288 pages, 26,90 €
EBEN UPTON, GARETH HALFACREE
Écrit par le co-créateur du Raspberry Pi, cet ouvrage donne toutes les clés pour tirer le meilleur parti du nano-ordinateur révolutionnaire.



9782100746835, 200 pages, 24,90 €
MARC-OLIVIER SCHWARTZ
Initiez-vous à la domotique avec le Raspberry Pi associé à la puce Wi-Fi ESP 8266 : 12 projets concrets pour rendre votre maison plus « intelligente ».

TOUT LE CATALOGUE SUR **WWW.DUNOD.COM**

DUNOD
ÉDITEUR DE SAVOIRS

AJOUTEZ UNE SORTIE AUDIO BLUETOOTH À VOTRE RASPBERRY PI

Denis Bodor

Comme pour d'autres configurations, le fait de disposer d'une interface graphique facilite grandement la prise en charge de certaines fonctionnalités et/ou le support de certains matériels. Il arrive, cependant, que pour bon nombre de projets, cette interface graphique soit totalement inutile pour la simple et bonne raison que la Pi n'aura pas d'écran connecté en HDMI. Ceci complique un peu les choses lorsqu'il s'agit utiliser uniquement la ligne de commandes, mais cela reste parfaitement faisable. Exemple avec un haut-parleur Bluetooth…

Voici le produit utilisé pour cet article, un SRS-X11 de Sony (59€). Initialement destiné aux smartphones, ce périphérique Bluetooth fait office à la fois d'enceinte et de kit mains-libres. Parfaitement adapté à un usage avec une Raspberry Pi, il permet d'éviter d'utiliser des haut-parleurs de PC, plus encombrants et non autonomes.

Je dois l'avouer, la plupart de mes projets à base de Raspberry Pi ne nécessitent pas d'écran, ni même un clavier USB connecté. L'utilisation de la carte, quel que soit son modèle, est, à mon sens, beaucoup plus rapide et efficace en utilisant la ligne de commandes, généralement au travers d'une connexion distante comme SSH et tantôt, en cas de problème, via une console série. De ce fait, naturellement, je privilégie généralement Raspbian Lite à toutes autres versions du système afin de limiter l'encombrement de la carte micro-SD et la charge du processeur.

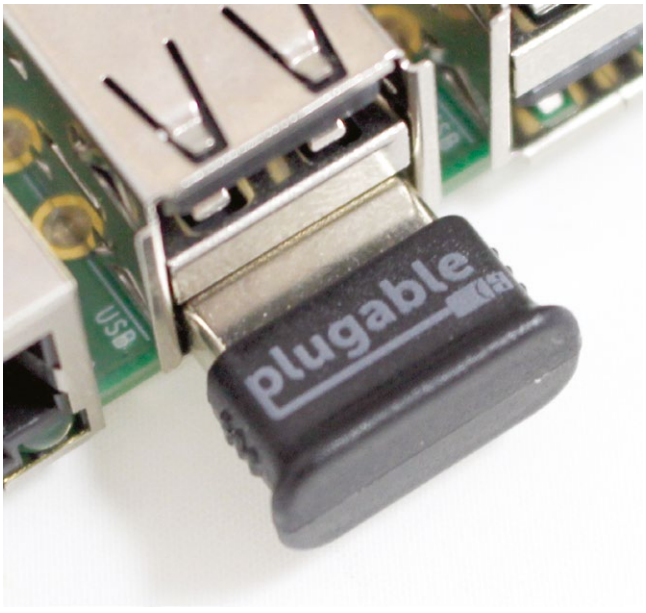
Ce type d'utilisation induit généralement certains petits problèmes et, dans le cas qui nous occupe pour cet article, la nécessité de comprendre comment le système Raspbian est architecturé et les choix qui ont été faits pour les différents éléments qui le composent. Le sujet ici est de se passer de la sortie audio standard de la carte Raspberry Pi pour utiliser, en remplacement, une mini-enceinte Bluetooth comme il en existe des dizaines de modèles... afin que des zozos puissent se balader dans la rue en nous cassant les oreilles (oui, j'ai un côté vieux chnoque qui transparaît de temps en temps).

Le principal intérêt d'utiliser une telle sortie audio Bluetooth, qui est également celle utilisée pour un casque sans fil, est non seulement de fournir une sortie audio de bonne qualité et avec un bon volume sonore sous un format assez compact, mais, par la même occasion, de permettre également l'utilisation de la PWM matérielle nécessaire, par exemple, pour la connexion de leds « intelligentes » type WS2812b (alias NeoPixels) et APA106.

1. BLUETOOTH ET AUDIO

Nous l'avons vu dans les numéros précédents, connecter un appareil Bluetooth à une Raspberry Pi, qu'il s'agisse d'une 3 ou d'un autre modèle équipé d'un adaptateur Bluetooth USB, n'est pas quelque chose de très difficile. Il suffit d'utiliser les bonnes commandes, scanner le voisinage, repérer le périphérique, l'associer et lui faire confiance pour permettre une reconnexion automatique.

Lorsqu'il s'agit d'audio en revanche, les choses ne sont pas aussi simples et la notion de « son » en Bluetooth couvre en réalité pas mal de choses. Il faut donc savoir précisément de quoi on parle. Les périphériques Bluetooth utilisent des profils permettant aux fabricants



La Raspberry PI 3 intègre directement le support du Bluetooth (et du Wifi), mais pour les modèles précédents, il suffit d'ajouter un adaptateur USB Bluetooth qui fournira exactement les mêmes fonctionnalités.

de reposer sur des caractéristiques, des fonctionnalités et des protocoles standardisés, et donc de produire du matériel qui fonctionnera avec d'autres équipements sans nécessiter l'installation de logiciels spécifiques. C'est la raison pour laquelle une enceinte, un casque ou un kit mains-libres d'une marque et d'un modèle donné fonctionnera tout aussi bien avec n'importe quel smartphone ou encore n'importe quelle installation audio Bluetooth (comme celles présentes dans les voitures).

Pour l'audio, il n'y a pas un profil, mais quatre :

- HSP (*HeadSet Profile* ou profil casque) : fournit les fonctionnalités de base pour établir la communication entre un smartphone ou un mobile et un casque ;
- HFP (*Hands Free Profile* ou profil mains-libres) est une amélioration du Profil HSP qui a été conçu pour permettre le contrôle d'un téléphone portable depuis un kit mains-libres dans une voiture par exemple ;
- A2DP (*Advanced Audio Distribution Profile* ou profil de distribution audio avancée) est destiné à la transmission de signaux audios stéréo d'une qualité supérieure au codage mono utilisé avec les profils HSP et HFP qui sont initialement destinés aux simples communications vocales ;
- AVRCP (*Audio/Video Remote Control Profile* ou profil de commande à distance audio/vidéo) est destiné à l'envoi de commandes de lecture (lecture, avance rapide, pause, morceau suivant, etc.) depuis un périphérique comme un casque ou une enceinte vers un appareil comme un smartphone ou une chaîne hifi compatible.

Ces désignations se retrouvent généralement sur les emballages des produits en vente et dans leurs spécifications techniques... quand on les trouve. À ce propos d'ailleurs, je ne saurai que trop vous conseiller de bien vous renseigner sur le Web avant l'achat d'un haut-parleur ou d'un casque Bluetooth.

J'avais fait l'acquisition il y a quelque temps d'un haut-parleur de ce type, de marque DCybel, modèle Mini Drum, et après une fouille méticuleuse du net, n'ai pas réussi à mettre la main sur une notice en PDF. Il semblerait que ce produit ne soit qu'une déclinaison d'un modèle générique, existant dans divers modèles (avec ou sans emplacement micro-SD par exemple), mais que personne n'ait pris la peine de mettre à disposition une documentation correcte. Ni le réel fabricant qui reste inconnu à ce jour (même après démontage de l'objet), ni les distributeurs/revendeurs. L'appareil est donc associé à un smartphone et j'ai fini par jeter l'éponge après avoir tenté, par tous les moyens possibles, de le rendre « découvrable » à nouveau.

Le produit en question étant également utilisable via un connecteur jack comme une enceinte amplifiée, j'ai donc décidé de le mettre (brutalement) de côté

(tout en pestant) pour un autre usage et de jeter (joyeusement cette fois) mon dévolu sur un autre modèle, fabriqué par Sony : un SRX-X11, cubique, avec davantage d'autonomie, de meilleure qualité générale et surtout avec une documentation qui restera accessible pour un bout de temps.

Mais revenons à nos moutons. Ces profils Bluetooth sont parfaitement gérés par la pile BlueZ intégrée dans les systèmes reposant sur Linux, comme Raspbian et toutes les distributions GNU/Linux (mais aussi Android pour les versions qui n'utilisent pas Bluedroid). Mais il ne s'agit là que d'une partie de la solution.

Le système audio de base des distributions GNU/Linux est ALSA pour *Advanced Linux Sound Architecture*. Celui-ci fournit les fonctionnalités audios et MIDI pour l'ensemble du système, des pilotes jusqu'aux outils de configuration et de contrôle. Pour utiliser une sortie audio Bluetooth (A2DP), il faut cependant qu'un lien soit établi entre la gestion Bluetooth et le système audio, et là, se pose un problème : BlueZ depuis sa version 5.0 ne supporte plus ALSA, mais dialogue avec un serveur de son.

Le support ALSA était déjà une avancée notable réglant bon nombre de problèmes concernant la gestion audio dans Linux. Avant son arrivée, ceci était pris en charge par une précédente solution appelée OSS (*Open Sound System*), mais qui possédait alors certaines limitations pénibles (comme le fait qu'une seule application à la fois pouvait émettre du son). Le changement intervint avec l'arrivée du noyau Linux 2.6, mais ceci ne régla qu'une partie des limitations.

L'une des problématiques qu'ALSA ne règle pas concerne la redirection des flux audios. Si l'on cherche, par exemple, à enregistrer le son émis via les haut-parleurs, comme ceux d'une vidéo lue ou d'une conversation en ligne, il faut connecter la sortie audio d'une application à la fois à la sortie physique, mais également à l'entrée d'une autre application, le tout sans que cela ne demande de modification dans les applications en question. Ce besoin d'une gestion plus avancée des flux audios a poussé à la création de choses comme ESD (*Enlightened Sound Daemon*) utilisé par les environnements graphiques Enlightenment et GNOME à une certaine époque. ESD a ensuite cédé sa place à PulseAudio : un serveur de son se plaçant entre les applications et la prise en charge du matériel par le noyau.

PulseAudio est ce qu'on appelle un *middleware*, il prend en charge toute la gestion sonore, l'interconnexion des sources et des sorties et permet même une utilisation en réseau. Ceci est si bien intégré que même les applications reposant uniquement sur ALSA sont prises en charge, via un périphérique virtuel qui renvoie les données à PulseAudio, qui lui-même utilise ALSA pour produire le son. Cependant, PulseAudio n'est **PAS** un support matériel et n'est **PAS** intégré au noyau, cela reste un programme au même titre qu'un éditeur de texte ou qu'un outil en ligne de commandes.

Avec BlueZ 5.0 et supérieur, le support Bluetooth ne dialogue donc plus directement avec ALSA pour fournir un nouveau périphérique audio, mais avec PulseAudio qui, à son tour, se chargera de fournir aux applications des fonctionnalités pour l'utiliser.

Lorsque vous démarrez une Raspberry Pi en mode graphique, PulseAudio est automatiquement lancé pour fournir à toutes les applications, et à l'environnement graphique lui-même, des capacités sonores. C'est également le cas, la plupart du temps, avec des distributions GNU/Linux comme Ubuntu ou Debian.

Vous l'avez sans doute compris, Raspbian Lite n'intègre pas par défaut PulseAudio et pour prendre en charge une enceinte Bluetooth dans cette situation, vous devrez donc non seulement configurer le support Bluetooth de votre système, mais aussi installer/configurer PulseAudio. Ceci fait, votre enceinte Bluetooth deviendra littéralement une « carte son » et vous pourrez vous passer de la sortie standard de la Raspberry Pi.

2. INSTALLATION ET CONFIGURATION

Par défaut dans Raspbian, le support et les outils de gestion Bluetooth sont installés par défaut. PulseAudio en revanche manque à l'appel et la première chose à faire sera donc d'installer les paquets adéquats avec :

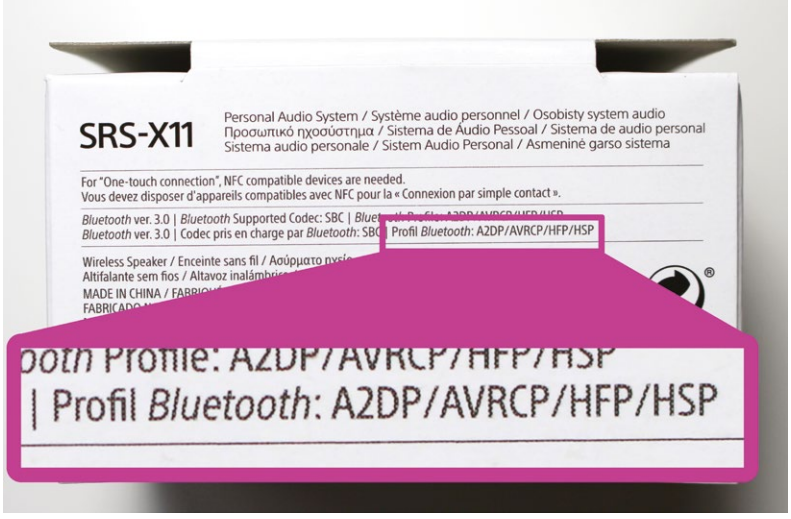
```
$ sudo apt-get install pulseaudio pulseaudio-utils pulseaudio-module-bluetooth
```

Notez que PulseAudio, bien qu'étant un serveur fonctionnant en tâche de fond, n'est pas lancé automatiquement comme un service. Ceci est un point très important et les développeurs de PulseAudio sont très explicites à ce sujet. Il est regrettable de voir trop souvent des tutoriels en ligne passer totalement outre cet état de fait. Depuis la version 0.9.3 (nous en sommes à la 5.0), il est possible de demander à PulseAudio de fonctionner comme un service/démon, dans un mode appelé *system mode*. PulseAudio est conçu pour être lancé une fois « par utilisateur » et sous l'identité de cet utilisateur, le *system mode* implique des risques très importants en termes de sécurité, charge davantage le processeur en raison de la désactivation du partage de mémoire, limite les fonctionnalités et n'est absolument pas conçu pour un tel usage. De plus, il est parfaitement possible de faire sans ce *system mode*...

Lisez attentivement les spécifications techniques du produit Bluetooth que vous comptez acheter. Le profil Bluetooth utilisé pour nos manipulations est A2DP. Le profil HSP n'est pas supporté par PulseAudio en version 5.0 et la version 6.0 n'est pas encore intégrée dans Raspbian stable. Il est peu probable qu'un périphérique comme une enceinte Bluetooth ne supporte pas A2DP, mais restez-y attentif...

Une fois PulseAudio installé, la première étape consiste à associer votre magnifique périphérique Bluetooth. Pour cela, il vous suffit de lancer **`bluetoothctl -a`** après avoir mis le matériel en mode « découvrable » (une pression longue sur le bouton d'allumage avec le SRS-X11) puis, dans l'interface qui vous est proposée :

- vous assurer que le contrôleur Bluetooth est en route avec **`power on`**,
- rechercher de nouveaux périphériques avec **`scan on`**,
- appairer le matériel avec **`paire`** suivi de l'adresse Bluetooth du périphérique détecté (**`FC:A8:9A:1D:8D:39`** par exemple),
- entrer éventuellement un code PIN si le produit le demande,
- faire confiance au périphérique fraîchement appairé avec **`trust`** suivi, encore une fois, de l'adresse Bluetooth,
- et enfin quitter **`bluetoothctl`** avec **`quit`**.



Vous remarquerez sans doute que votre périphérique va rester connecté très brièvement puis se déconnecter. Ceci est parfaitement normal étant donné que nous n'avons absolument rien fait pour maintenir la connexion. PulseAudio n'étant pas lancé, la connexion Bluetooth n'a aucune raison de rester établie.

À présent, éteignez votre enceinte (ou votre casque) et lancez le serveur PulseAudio avec **`pulseaudio --start`**, puis allumez à nouveau votre périphérique (si des messages inquiétants apparaissent c'est normal, ceci n'arrive qu'au premier lancement lors de la création des répertoires). Normalement, celui-ci devra immédiatement se connecter et être utilisable. Vous pouvez vérifier sa prise en charge au niveau Bluetooth avec **`bluetoothctl`** en utilisant la commande **`info`** suivie de l'adresse du périphérique, parmi les lignes affichées, vous devez voir **`Connected: yes`**.

Vous pouvez également vérifier du côté de PulseAudio en utilisant la commande **`pacmd list-cards`**. À ce stade, vous devez voir deux cartes disponibles : la sortie standard de la Raspberry Pi et le périphérique Bluetooth (mention **`%%driver: <module-bluez5-device.c>`** dans la sortie ainsi que le nom du matériel sous **`properties:`**).

Vous pouvez immédiatement tester la sortie en utilisant l'outil **`paplay`** et en faisant suivre cette commande d'un nom de fichier audio (WAV, FLAC, etc. (mais pas MP3, pour cela installez/utilisez **`mpg123`**)). Si tout est bien configuré et installé, le son sera automatiquement transmis au périphérique.

3. FINALISATION ET PEAUFINAGE

Nous n'en avons pas encore fini. Pour l'instant, nous devons encore lancer le serveur PulseAudio manuellement. En effet, avec un environnement graphique ce lancement se fait en même temps que le démarrage de l'interface et donc immédiatement après l'ouverture de session pour un utilisateur (ou l'ouverture automatique en tant que **`pi`**).

Ce problème peut être réglé autrement qu'en utilisant le fameux *system mode*. Il n'est pas nécessaire de lancer PulseAudio au démarrage et pour tout le système, il suffit de le faire au moment où une application demande à utiliser le système audio. Pour faire cela, vous devez changer un paramètre dans le fichier **`/etc/pulse/client.conf`**. Là, modifiez simplement la ligne :

```
; autospawn = yes
```

en

```
autospawn = yes
```

Le point-virgule (**`;`**), comme le dièse (**`#`**), permet de placer des commentaires dans le fichier de configuration, le fait de le retirer en début de ligne permet d'activer la ligne en question.

Il y aura un petit délai lors de la première utilisation d'une application audio, le temps que PulseAudio démarre, mais cela n'arrivera qu'une fois par démarrage

Voici un DCybel Mini Drum (20€) acheté il y a plus d'un an. La notice d'utilisation est introuvable que ce soit chez moi (tout à fait normal) ou sur le Web (parfaitement inacceptable). La bestiole est appairée avec un smartphone et je n'ai pas réussi, sans manuel, à le rendre « découvrable » à nouveau. D'un autre côté, j'ai toujours préféré les cubes aux cylindres et le noir au rouge...

de la Pi. Il est également possible que cela ne fonctionne pas, mais c'est purement fonctionnel : tenter de diffuser du son provoquera bien le lancement de PulseAudio, mais si le périphérique est déjà allumé, la connexion ne se fera pas forcément, car celui-ci peut être en pause et donc non connecté. En éteignant le périphérique et en le rallumant, tout rentrera alors dans l'ordre puisque, là, PulseAudio sera lancé et le matériel ne repassera pas en veille.

Ce point nous amène directement à un autre problème. Avec PulseAudio lancé, il est possible que le matériel coupe la connexion, mais, plus certainement, que PulseAudio le fasse (et le matériel s'éteindra alors automatiquement peu de temps après). En effet, par défaut, un module PulseAudio est automatiquement lancé : **`module-suspend-on-idle`**. Celui-ci désactive une sortie ou une entrée audio si celle-ci n'est pas utilisée un certain temps.

Pour désactiver ce module définitivement, éditez le fichier **`/etc/pulse/default.pa`** et cherchez une ligne **`load-module module-suspend-on-idle`**. Placez ensuite un **`#`** ou un **`;`** en début de ligne, enregistrez le fichier, puis redémarrez PulseAudio avec :

```
$ pulseaudio --kill
$ pulseaudio --start
```

Attention cependant, en faisant cela votre périphérique ne passera jamais en veille et videra donc sa batterie comme si vous écoutiez de la musique. Ceci est peut-être fonction du matériel utilisé, mais dans le cas de mon SRS-X11 c'est bel et bien le cas. Ce n'est pas un gros problème, je ne compte allumer l'enceinte que lorsque j'en aurai l'usage, et penserai à l'éteindre ensuite (ou pas).

Enfin, il ne nous reste plus qu'à désactiver quelque chose qui est clairement devenu inutile : la sortie audio standard de la Pi. Pour cela, éditez le fichier **`/boot/config`** et ajoutez une ligne **`dtparam=audio=off`**. Au prochain démarrage de la carte, la sortie sera désactivée et, éventuellement, il deviendra possible d'utiliser les leds type WS2812b avec la Pi.

Et voila ! Vous avez à présent une Raspberry Pi avec une sortie audio Bluetooth (et une entrée si le matériel possède un micro) parfaitement utilisable, peut encombrante, mobile et, selon le périphérique, de bonne qualité. **DB**

CONNECT ÉVOLUE !

LISEZ CE NUMÉRO ET PLUS DE 15 AUTRES EN LIGNE !

ACTUELLEMENT SUR CONNECT :

- **CE NUMÉRO**
- **et + de 15 autres numéros de Hackable**

+

- **2 numéro Hors-Série de Hackable**

TOUT CELA À PARTIR DE 149 € TTC*/AN !

* Tarif France Métropolitaine

OFFRE DÉCOUVERTE CONNECT

1 MOIS GRATUIT, RÉSERVÉE AUX PROFESSIONNELS

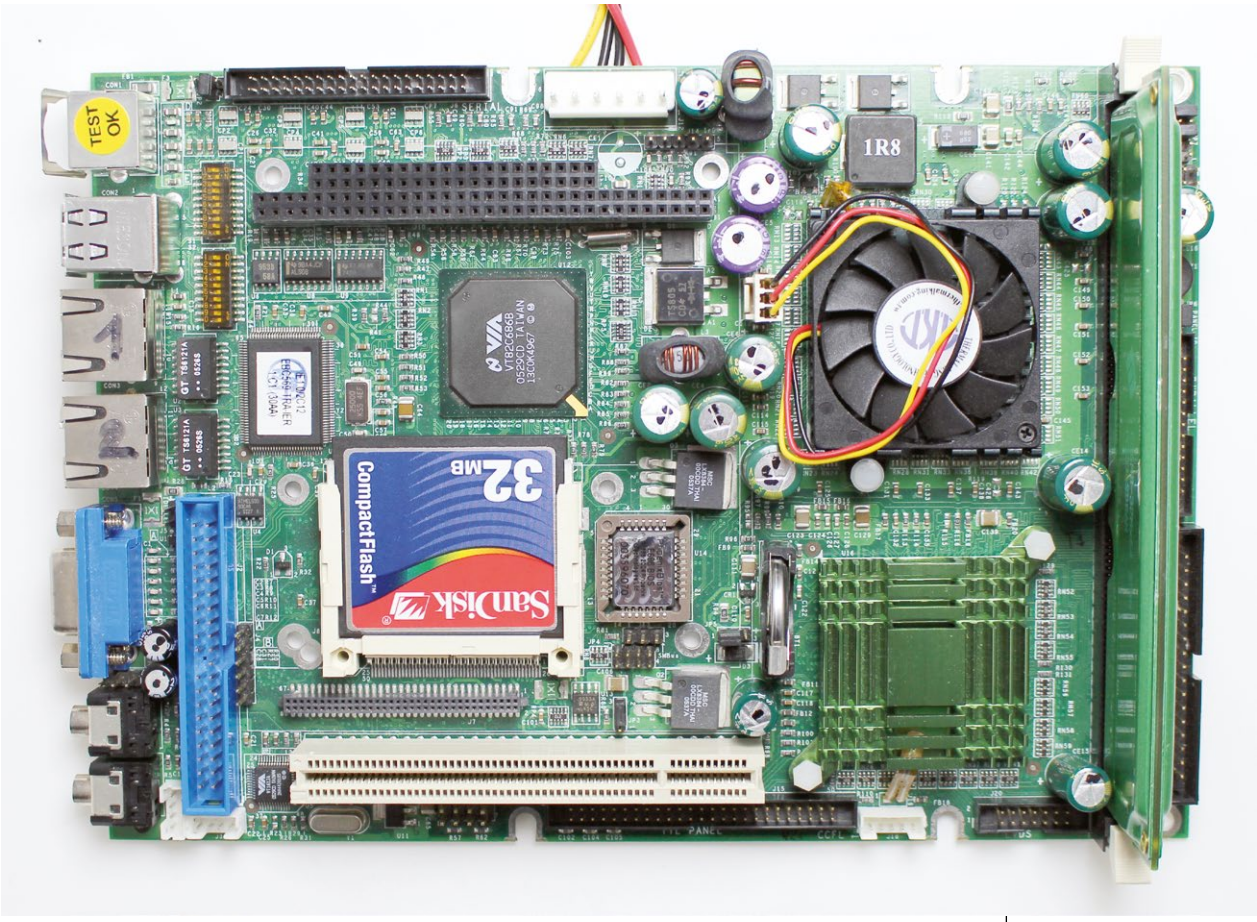
Appelez le 03 67 10 00 28 et donnez le code « HK18 » pour découvrir Connect gratuitement pendant 1 mois !

Pour tous renseignements complémentaires, contactez-nous via notre site internet : **www.ed-diamond.com**, par téléphone : **03 67 10 00 28** ou envoyez-nous un mail à **connect@ed-diamond.com** !

DOS N'EST PAS MORT ! IL VA MÊME TRÈS BIEN !

Denis Bodor

Souvenez-vous, ce n'était pas il y a si longtemps : CONFIG.SYS, MSCDEX, batailler pour avoir 600ko de mémoire conventionnelle, AUTOEXEC.BAT, les bons jeux d'Apogee Software, de LucasArts et de Blizzard, les sharewares en tout genre qu'on achetait sur disquettes, le téléchargement sur des BBS... Ah souvenirs ! Mais tout ceci est-il vraiment un passé définitivement révolu ? Non, pas vraiment. Il est toujours possible d'émuler cet environnement mais, mieux encore, on peut tout simplement recycler un vieux PC et faire renaître le bon vieux temps...

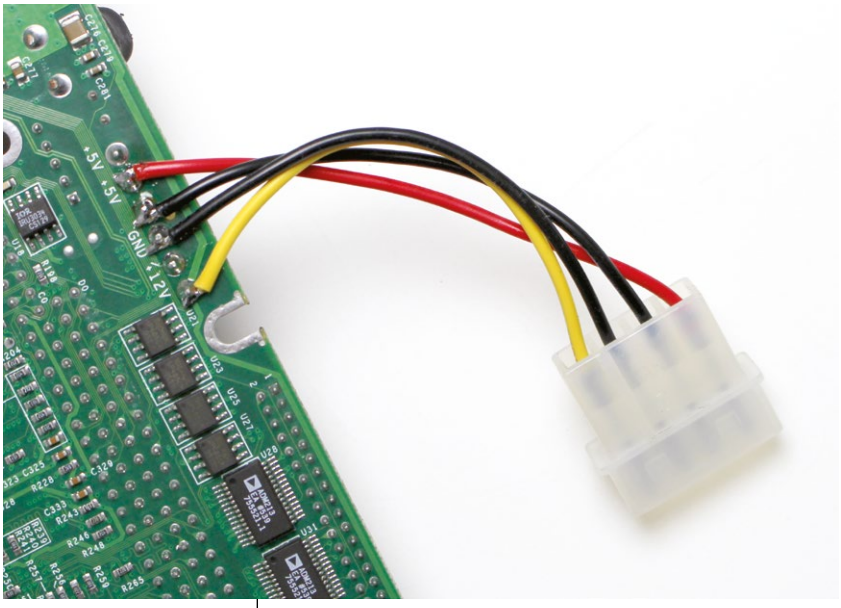


La carte mère utilisée pour ce projet est une ebc569 de Nexcom qui intègre un processeur VIA C3 à 800 Mhz, tous les périphériques nécessaires dont un emplacement pour carte CompactFlash, elle est équipée d'une barrette de 256 Mo de SDRAM PC133.

Je sais ce que vous vous dites : voilà une bien étrange approche dans un magazine traitant surtout de Raspberry Pi et d'électronique numérique en tout genre. Après tout, il existe des solutions permettant à une Pi d'émuler un PC sous MS/DOS, la plus connue est sans le moindre doute DOSBox, s'installant via une simple commande `sudo apt-get install dosbox`.

Seulement voilà, si une telle solution fonctionne relativement bien sur PC, il n'en va pas de même sur une Pi : c'est lent... c'est horriblement lent. Et s'il s'agit de profiter pleinement des bons vieux jeux de l'époque, cette lenteur n'est pas acceptable, en particulier lorsque le premier jeu qui vous vient à l'esprit est ce que j'estime être le chef d'œuvre de la carrière de Jeff Minter : le fantastique, l'incontournable, l'indispensable *Llamatron: 2112* !

Notre objectif sera donc ici de ressusciter de vieux artefacts de PC pour en faire une machine DOS parfaitement utilisable. La notion de « vieux » est ici toute relative, car ceci peut aller du 486 au Celeron, en passant par les Pentium et autres processeurs AMD, Cyrix, etc. Le choix, qui dépendra en premier lieu de ce que vous aurez sous la main ou en stock, est toujours à double tranchant. Plus un matériel sera ancien, plus il sera facile de le faire fonctionner avec un système de la même époque, mais moins vous risquez de trouver de matériel compatible (alimentation, clavier, cartes, mémoire, etc.). Inversement, plus le matériel sera récent, plus vous aurez de chances de mettre la main sur des périphériques utilisables, mais moins il sera probable de trouver des pilotes pour autre chose que Windows 95 ou supérieur. Or là, le but est clairement d'obtenir une machine DOS et non Windows (ce n'est pas la même magie).



La carte mère n'utilise ni un connecteur pour alimentation AT ni ATX, mais accepte directement des tensions de +5V (6A) et +12V (1A). La solution a ici été de souder directement un connecteur d'alimentation Molex permettant la connexion d'une alimentation similaire à un disque dur ou un lecteur CD-ROM.

On ne s'en rend compte que lorsqu'on cherche un matériel précis, mais les périphériques « vintages » commencent à sérieusement prendre de la valeur. Une petite recherche sur eBay pour une carte son ISA 16 bits, par exemple, du type Sound Blaster, testée et en état de marche, vous retourne des annonces affichant des prix entre 50€ et 150€. Si votre quête porte sur du matériel qui, à l'époque déjà, n'était pas à la portée de tous, comme une Gravis Ultrasound (une « GUS » pour les intimes), la tranche de prix sera davantage de 200€ à 450€ ! Les cartes graphiques VGA ISA 16, VLB et PCI semblent moins impactées... pour l'instant (mais une expédition punitive dans mon grenier est d'ores et déjà planifiée pour retrouver, nettoyer, classer et ranger bien proprement toutes ces petites choses qui gagnent clairement en valeur chaque mois qui passe).

1. PHASE 1 : LE MATÉRIEL

Mon idée ici est certes de reconstruire un PC pour satisfaire mon envie de Llamatron mais, pour autant, il n'est pas question de me retrouver avec une tour massive de plus dans mon bureau (à force de tout garder, on finit forcément par manquer de place). Plusieurs options se présentaient à moi, mais j'ai finalement opté pour la plus compacte et intégrée : une carte mère EBC569 de Nexcom normalement destinée à un usage industriel.

C'est un peu surdimensionné, mais elle présente l'avantage de disposer de tout le nécessaire directement intégré :

- processeur VIA C3 Nehemiah à 800 Mhz (équivalent à un Pentium III) ;
- carte graphique VGA Savage4 intégrée au *North Bridge* VIA 8606 ;
- carte audio compatible AC97 *et* Sound Blaster (cf. plus loin dans l'article) intégrée au chipset VT82C686B ;
- deux contrôleurs Ethernet 10/100 RTL 8139C ;
- quatre ports série 16C550 (on sait jamais, ça peut servir) ;
- un unique slot PCI ;
- des connecteurs/contrôleurs disquettes et E-IDE ;
- deux prises USB 1.1 et un connecteur PS/2 ;
- un emplacement pour carte CompactFlash fort sympathique ;
- et tout un tas de connecteurs industriels (LVDS, TTL LCD, PC 104, interface PCI propriétaire, GPIO, etc.).

La carte supporte jusqu'à 512 Mo de SDRAM PC-133 sous la forme d'un unique emplacement DIMM 168, et disposait déjà d'une barrette de 256 Mo lors de son excavation, largement suffisante pour l'usage visé.

La documentation très explicite et complète (c'est l'avantage d'utiliser une carte mère industrielle) précise que le ventilateur

placé sur le processeur n'est pas nécessaire à 400 Mhz et que la circulation d'air du châssis est alors suffisante. Cette configuration cependant se fait à l'aide de cavaliers d'une taille non standard (au pas de 2mm et non de 2,54 mm) et sera faite plus tard, pour l'heure 800 Mhz et le ventilateur feront l'affaire.

De plus, côté alimentation, le connecteur proposé n'a pas besoin de toutes les tensions courantes avec les alimentations AT ou ATX, seuls +5V (6A), +12V (A1) et la masse sont nécessaires. Autre avantage intéressant à ce niveau, il est possible d'utiliser une alimentation AT et ATX, via la prise en charge, directement par la carte, de la broche PWR_ON/PS_ON (le fil vert). Les alimentations ATX, en effet, ne disposent pas d'interrupteur et ne se mettent pas en route automatiquement lors de la connexion du courant (la broche PWR_ON doit être reliée et maintenue à la masse).

La mise en route préalable de la carte mère se fera à l'aide d'une alimentation ATX magnifiquement équipée d'un trombone entre PWR_ON et la masse. Des tests préalables avec une alimentation de laboratoire indiquent qu'en effet, à 800 Mhz, et avec un lecteur CD également alimenté, des pics de courant importants surviennent et font alors chuter la tension de l'alimentation qui n'arrive pas à suivre ce régime. Le connecteur sur la carte mère, composé de 6 broches (+5, +5, GND, GND, GND, +12) n'étant pas très pratique, j'ai alors simplement soudé un connecteur Molex femelle récupéré sur un ventilateur pour processeur (le modèle s'intercalant entre l'alimentation et un disque dur par exemple). Ceci m'a permis d'utiliser une alimentation ATX et de brancher la carte mère comme un disque ou un lecteur CD.

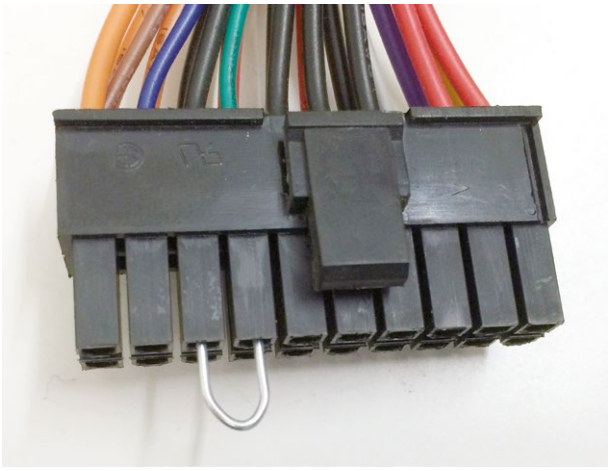
Ainsi, la carte mère, une carte CompactFlash de 32 Mo, une alimentation ATX, un lecteur CD-ROM IDE et un clavier PS/2 et nous voilà fin prêts pour la partie logicielle.

NOTE : La gestion de l'USB sur cette carte mère semble soit avoir un problème, soit être très limitée à la base. L'utilisation d'un clavier USB ailleurs que dans la configuration du BIOS (accessible par la touche Suppr au démarrage) ne fonctionnant pas et l'impossibilité de démarrer sur une clé USB, expliquent le choix du lecteur CD et du clavier PS/2.

2. PHASE 2 : LE SYSTÈME

Pour notre gentille bestiole, nous n'allons pas courir après de vieilles disquettes de MS-DOS, mais opter pour une solution bien plus récente et surtout, toujours maintenue et développée : FreeDOS.

FreeDOS est un système d'exploitation complet, en logiciel libre et 100% compatible avec MS-DOS. Mieux encore il intègre, comme nous le verrons plus loin, des fonctionnalités et des outils que tout utilisateur DOS de l'époque aurait pu rêver d'avoir. Développé depuis 1994 par Jim Hall et les nombreux contributeurs du projet, FreeDOS peut être téléchargé gratuitement et être installé aussi bien dans un émulateur ou une machine virtuelle que sur du matériel bien réel.



Les alimentations ATX ne fonctionnent pas comme un simple bloc d'alimentation standard ou encore une alimentation AT. L'astuce pour les mettre en route consiste à mettre le signal PWR_ON (vert) à la masse. Un simple trombone fera l'affaire le temps de trouver une alimentation d'une taille plus acceptable.

La dernière version en date est la toute récente 1.2 (décembre 2016) qui peut être téléchargée directement sur **http://www.freedos.org/download/** pour trois types d'installation :

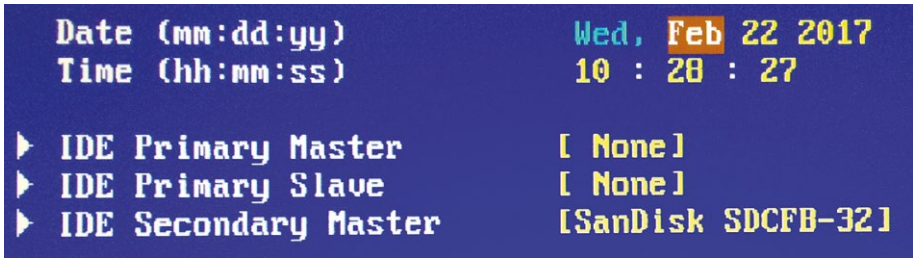
- CD-ROM : C'est la méthode d'installation recommandée. Il suffit de télécharger l'image ISO, la graver sur un support vierge et de démarrer la machine (virtuelle ou non) dessus. En cas de problème de démarrage, une seconde image ISO (*legacy*) est disponible.
- Disquette + CD-ROM : Cette solution est à préconiser si votre machine n'est pas capable de démarrer directement sur un CD (cas typique des lecteurs CD connectés à une carte son par exemple). Là, la solution consiste à télécharger l'image de la disquette, l'enregistrer sur le support avec *win32diskimager* sous Windows ou **dd** sous GNU/Linux, et de graver l'image ISO du CD-ROM. On démarrera alors la machine sur la disquette et l'installation se poursuivra depuis le CD-ROM.
- Support USB : C'est la solution qui semble la plus simple en théorie, mais tout dépendra de votre machine et de sa capacité à démarrer sur une clé USB. Dans mon cas, cette solution n'a malheureusement pas été utilisable. Le BIOS de la machine propose un démarrage sur USB-HDD, USB-FDD, USB-CDROM ou USB-ZIP, mais aucune de ces options n'a permis le démarrage. J'ai donc été obligé de temporairement utiliser un lecteur CD-ROM.

L'installation en elle-même est relativement simple et surtout traduite en français après une simple sélection dans un menu proposé dès le début de la procédure. L'espace disque étant très limité dans mon cas, avec une CompactFlash de seulement 32 Mo (impossible de remettre la main sur les autres en ma possession), la phase finale de l'installation s'est limitée au choix le plus basique avec le minimum d'outils installés.

Notez qu'en cas de partitionnement/formatage d'un disque dur, il vous sera nécessaire de procéder à l'installation en deux fois : démarrage, formatage, redémarrage et installation.

Remarquez également que le choix d'une CompactFlash comme support de stockage n'est pas anodin. Les disques IDE ou E-IDE se font rares et ceux généralement encore disponibles au fond d'un tiroir ont déjà quelques heures de vol. Un disque dur est un périphérique mécanique et ceci suppose une usure non négligeable et une durée de vie limitée. Une bonne solution, si votre PC ne dispose pas d'un emplacement pour une carte CompactFlash, consiste à utiliser un adaptateur IDE/CF qui est souvent un simple circuit imprimé avec supports/connecteurs.

En effet, les cartes Compact Flash disposent de plusieurs modes de fonctionnement dont un appelé *True IDE* qui, comme son nom l'indique, leur permet d'être vus comme des disques. Un adaptateur vous coûtera une poignée d'euros qu'il s'agisse d'un modèle avec une nappe 40 broches ou prenant directement place sur le connecteur IDE mâle de la carte mère. Une alimentation est également tantôt nécessaire



Le BIOS de la carte mère reconnaît la carte CompactFlash comme un disque dur. Ceci n'a rien de bien exceptionnel puisque ce type de carte dispose d'un mode de fonctionnement spécialement prévu pour ce type d'usage.

sous la forme d'un connecteur pour lecteur de disquettes ou de disque dur.

3. PHASE 3 : LES PROBLÈMES

Après l'installation et un premier démarrage, comme on peut s'y attendre, on retrouve tout le charme d'un système DOS. En faisant un peu travailler sa mémoire, les réflexes reviennent naturellement. Inutile ici de revenir sur l'utilisation même de DOS, si vous envisagez de reconstruire un tel système c'est par nostalgie et donc que vous avez une certaine expérience qu'il vous suffira de réveiller.

J'ai déjà évoqué le problème du support USB avec cette carte mère qui n'est pas davantage pris en charge par FreeDOS qu'il ne l'est lors de tests avec une distribution GNU/Linux bootable appelée SystemRescueCd. Le clavier dans ce dernier cas est effectivement opérationnel, mais uniquement une fois pris en charge directement par le noyau Linux. Il semble donc que l'émulation d'un clavier standard par le BIOS ne fonctionne pas.

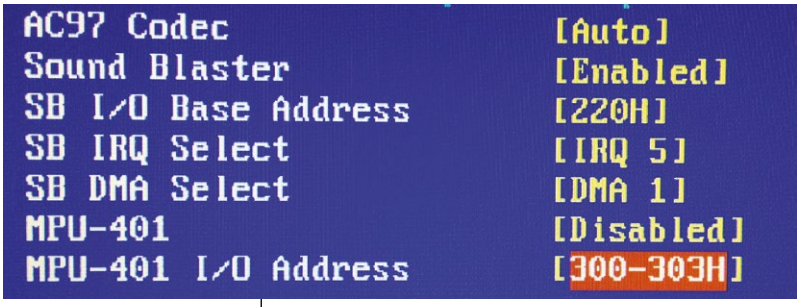
Si votre lecteur CD-ROM est encore physiquement présent après l'installation, celui-ci devrait apparaître comme le lecteur **D:** et pourra être consulté sans le moindre problème. Ceci vous permettra d'utiliser le contenu du CD-ROM de manière à pouvoir installer des outils et programmes supplémentaires si, comme moi, vous n'avez pas opté pour l'installation complète.

Si votre matériel de récupération ne dispose pas d'un emplacement pour carte CompactFlash, il est facile de trouver des petits adaptateurs IDE/CF pour quelques euros vous permettant de faire de même. Les disques IDE ou E-IDE se font rares de nos jours et ceux que l'on peut récupérer au fond d'un placard sont généralement déjà passablement usés et prompts à montrer des signes de faiblesse.

Tout ceci est très intéressant, mais il nous faut nous attaquer à l'objectif principal : jouer. Pour ce faire, il nous faut un support audio. Idéalement, pour ce genre de projet, la solution consiste à utiliser une carte d'époque, mais à moins d'avoir une Sound Blaster ou une carte compatible comme une Sound Galaxy dans un tiroir c'est plus problématique qu'il n'y parait.

Dans mon cas, avec le périphérique audio intégré à la carte mère, les choses se présentaient relativement bien. En effet, la configuration du BIOS proposait justement un fonctionnement compatible Sound Blaster et permettait de spécifier les paramètres habituels : adresse, interruption et DMA (le bon vieux « 220 5 1 »). C'était, cependant, sans compter la notion toute relative de « compatibilité » (c'est comme le *plug'n'play*, la perte de poids sans effort et la peinture monocouche, cela n'existe tout simplement pas).

Après de nombreuses recherches à la fois dans les archives des listes de discussion de FreeDOS et un peu partout sur le Web, il s'avérait que le fameux mode compatible ne



Si vous avez de la chance, et que la carte mère n'est pas trop récente, vous trouverez dans la configuration du BIOS des options permettant d'activer une émulation Sound Blaster. Ceci, dans mon cas, n'a cependant pas été suffisant et implique l'utilisation, en prime, de pilotes DOS relativement difficiles à trouver.

dépendait pas simplement de la configuration du BIOS, mais reposait également sur des programmes devant être lancés au démarrage, chargés d'activer la sortie audio et de servir de « relais » (synthèse FM).

Retrouver ces programmes n'est pas une mince affaire, car même si VIA propose des archives contenant des vieux pilotes DOS pour le chipset VT82C686A/B (**http://download.viatech.com**), ceci ne permet de récupérer qu'un seul des deux programmes, **VIAUDIO.COM**, permettant de corriger le fait que la sortie audio est muette par défaut. C'est finalement sur le forum *vogons.org* (**http://www.vogons.org/viewtopic.php?t=33283**) que la solution s'est faite jour, accompagnée des archives Zip proposés par un utilisateur ayant un problème similaire. Ainsi, en chargeant **VIAUDIO.COM** puis **VIAFMTSR.COM**, enfin la sortie audio est activée et utilisable comme une véritable carte compatible Sound Blaster.

Il suffit alors d'ajouter le lancement de ces deux commandes, après avoir placé les deux fichiers dans **C:\FDOS\BIN**, dans le fichier **AUTOEXEC.BAT** à la racine de **C:**, et le tour est joué.

La leçon à retenir ici est que le mélange de matériel récent et de technologies anciennes est très problématique. Je pense que j'ai eu de la chance, car la nature industrielle de la carte mère implique une utilisation possible avec de vieux applicatifs DOS par soucis de compatibilité avec des logiciels métiers. Ce ne sera peut-être pas le cas avec une plateforme plus récente intégrant un contrôleur audio. Mieux vaut, en principe, travailler avec un matériel de l'ère DOS ou, au moins pré-Pentium III, plus certainement en mesure de proposer des pilotes DOS pour les périphériques audios.

4. PHASE 4 : OH MON DIEU, DOS FAIT ÇA ?!

À ce stade je vous laisse deviner ma satisfaction à pouvoir à nouveau contrôler un lama pour combattre des hordes de boîtes de Coca, de cerveaux et de rouleaux de papier toilette tout en sauvant de pauvres ruminants sans défense. Mais FreeDOS ne fait pas que fournir une solution alternative et en logiciel libre à MS-DOS, il intègre des fonctionnalités modernes permettant d'être plus à l'aise avec cet environnement.

La première avancée notable concerne la gestion de paquets. En effet, l'installeur FreeDOS utilise un mécanisme assez similaire à ce qu'on trouve avec les distributions GNU/Linux. Plutôt que de copier simplement les fichiers sur le disque, il utilise des archives Zip contenant bien plus que les programmes eux-mêmes.

Sur le CD-ROM d'installation se trouve ainsi des dizaines de fichiers Zip installables individuellement à l'aide de la commande **FDNPKG**. Il suffit de s'y promener à l'aide des commandes classiques (**CD**, **DIR**, etc.) pour ensuite installer le paquet de votre choix avec :

```
C:\> FDNPKG install gnubc.zip
doc\gnubc\readme -> C:\FDOS\doc\gnubc\
doc\gnubc\examples\primes.b -> C:\FDOS\doc\gnubc\examples\
doc\gnubc\examples\ckbook.b -> C:\FDOS\doc\gnubc\examples\
doc\gnubc\copying -> C:\FDOS\doc\gnubc\
doc\gnubc\bc.man -> C:\FDOS\doc\gnubc\
bin\bc.exe -> C:\FDOS\bin\
appinfo\gnubc.lsm -> C:\FDOS\appinfo\
Le paquet gnubc a été installé : 7 fichiers extraits, 0 erreurs.
```

Comme avec n'importe quel gestionnaire de paquets, il est également possible de lister ceux déjà installés :

```
C:\> FDNPKG listlocal g
assign 1.4
debug 1.25
defrag 1.3.2
dialog 1.1-20080819
fdnpkg 0.99.4
gnubc 1.02
graphics 2008-07-14
```

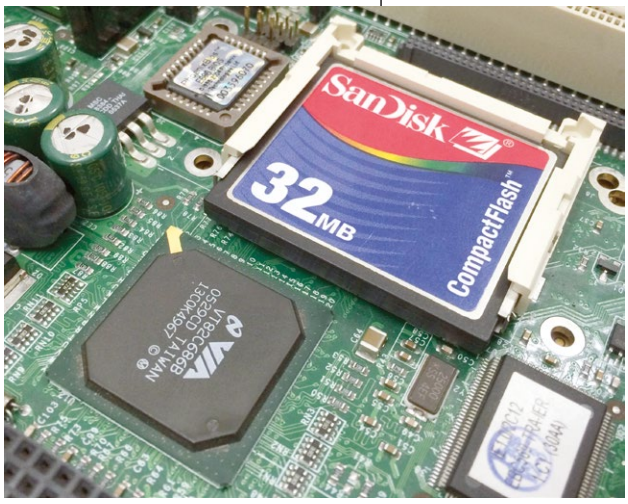
Et, bien sûr, vous pouvez désinstaller l'un d'eux sans problème :

```
C:\> FDNPKG remove gnubc
effacement de C:\FDOS\doc\gnubc\readme en cours
effacement de C:\FDOS\doc\gnubc\examples\primes.b en cours
effacement de C:\FDOS\doc\gnubc\examples\ckbook.b en cours
effacement de C:\FDOS\doc\gnubc\copying en cours
effacement de C:\FDOS\doc\gnubc\bc.man en cours
effacement de C:\FDOS\bin\bc.exe en cours
effacement de C:\FDOS\appinfo\gnubc.lsm en cours
Le paquet gnubc a été enlevé.
```

Mais parmi les évolutions importantes on trouve également le support réseau. Je ne parle pas ici de technologies comme IPX/SPX très en vogue à la belle époque, mais d'un support TCP/IP permettant d'intégrer votre machine DOS dans votre réseau actuel. Tout ceci repose sur le fait de disposer d'un support logiciel adapté.

Dans mon cas, après quelques tentatives et échecs, et pour les contrôleurs Ethernet RTL8139C intégrés, ce support a pris la forme du fichier **RTSPKT.COM**, récupéré sur **http://www.georgpotthast.de/sioux/packet.htm**. Les pilotes pour cartes réseaux sont appelés des *Packet Drivers* et sont constitués de programmes devant être lancés soit manuellement pour essai, soit directement au démarrage depuis **AUTOEXEC.BAT**.

Dans le cas de **RTSPKT.COM** et du fait que ma carte mère dispose de deux interfaces réseau, il m'a tout



Un autre avantage de la carte CompactFlash est sa taille relativement réduite. Ici au premier plan le fameux chipset VIA VT82C686B intégrant, entre autres choses, la carte son compatible AC'97 soi-disant compatible Sound Blaster...

d'abord fallut tâtonner un peu. Avec une seule interface utilisant une puce RTL8139, il suffit de lancer la commande en lui passant en argument une valeur correspondant à l'interruption logicielle à utiliser (typiquement **0x60**). Avec deux interfaces présentes cependant, le pilote ne sait pas quel périphérique utiliser et demande donc de l'aide à l'utilisateur :

```
C:\> RTSPKG 0x60
Packet Driver for Realtek RTL8139 Family PCI/Cardbus Fast Ethernet Network
Interface Cards, Version 3.40
Copyright 2000(c), Realtek Semiconductor Inc.

There are 2 network cards on your main board :
Card 1  SlotNo= 0x11. IRQ= 0xA  I/O= 0xE800  NodeID= 00:10:F3:09:52:D4
Card 2  SlotNo= 0x12. IRQ= 0xB  I/O= 0xEC00  NodeID= 00:10:F3:09:52:D3
Please select a Card No.(1-2):
[...]
```

Pour contourner le problème, il faut alors préciser cette information directement dans les arguments de **RTSPKG** en spécifiant l'interruption logicielle, le numéro du bus et du périphérique (correspondant au **slot** de la précédente commande) :

```
C:\> RTSPKG 0x60 0x00 0x11
Packet Driver for Realtek RTL8139 Family PCI/Cardbus Fast Ethernet Network
Interface Cards, Version 3.40
Copyright 2000(c), Realtek Semiconductor Inc.

Line Speed 100 Mbps
Full Duplex
System: [345]86 processor, PCI bus, Two 8259s
Packet driver software interrupt is 0x60
Interrupt number is 0xA
I/O port is 0xE800
My Ethernet address is 00:10:F3:09:52:D4
```

Ceci ne fait qu'installer le pilote, mais ne fournit pas de connectivité TCP/IP pour autant. Pour cela, il nous faut installer le paquet **FDNET.ZIP** qui fournit le support basique pour le réseau (configuration, client DHCP, etc.) et **MTCP.ZIP** pour disposer d'outils complémentaires (**ping**, FTP, Telnet, etc.). L'installation du paquet **FDNET.ZIP** comprend le fichier **C:\FDOS\BIN\FDNET.BAT** appelé depuis **AUTOEXEC.BAT**. On y retrouve une ligne concernant **RTSPKG** qu'il convient ici de modifier avec les arguments supplémentaires à **0x60**. Bien entendu, si vous n'avez qu'une interface réseau, ceci n'est pas nécessaire. Le fichier **FDNET.BAT** tente de détecter automatiquement l'interface présente à l'aide de l'outil **BERNDPCI.COM** et lance le *Packet Driver* adéquat.

Une fois cette modification faite, et si tout fonctionne correctement, au prochain démarrage du système le pilote sera chargé, l'interface configurée via DHCP et la machine se verra attribuée une adresse IP (par votre box ou un serveur DHCP que vous avez sur votre réseau). À ce propos d'ailleurs, il semblerait que les outils livrés par le paquet **MTCP.ZIP** n'apprécient pas un bail DHCP inférieur à 3600 secondes (une heure). Assurez-vous que votre configuration réseau prenne cette particularité en compte pour cette machine.

Vous avez désormais du réseau et vous pouvez tester cela en vous plaçant dans le répertoire **MTCP** et en utilisant **PING** pour vérifier la connectivité. Mais le meilleur reste à venir, car le gestionnaire de paquets **FDNPKG** est maintenant capable de fonctionner en réseau ! Vous pouvez ainsi

chercher dans les paquets disponibles en ligne avec **FDNPKG search** suivi d'une chaîne de caractères et, bien entendu, en installer un avec **FDNPKG install** suivi d'un nom (sans l'extension **.ZIP**).

Un DOS avec une gestion de paquets, des dépôts et une installation via le net, c'est plus que j'en aurai rêvé il y a... ouh, je préfère ne pas compter, à cette époque j'avais encore des cheveux, beaucoup de cheveux même.

Même les périphériques les plus anodins méritent d'être conservés. Je ne pensais pas réellement avoir besoin d'utiliser ce lecteur de carte à la forme et à la couleur des plus ignobles... et pourtant. Je vous le dis et je vous le répète : ne jetez jamais rien !

Enfin, même si la gestion de paquets et le réseau sont une nouveauté fantastique, il reste une autre fonctionnalité toute aussi intéressante : le support USB. FreeDOS permet en effet de vous donner accès à des périphériques USB comme des supports de stockage. Pour cela, vous devrez installer le paquet **USBDOS** vous fournissant différents outils et/ou pilotes.

Le comportement de ces outils est particulièrement lent sur mon installation, ce qui tend à confirmer un problème matériel, mais cela reste raisonnablement utilisable. La première chose à faire est de charger le support USB UHCI (*Universal Host Controller Interface*) en lançant la commande **USBUHCI**. Vous pourrez alors lister les périphériques présents avec :

```
C:\> USBDEVIC
USBDEVIC 0.05, (C) 2008, Bret E. Johnson.
Program to display information about Devices attached to the USB Host(s).

                              DEVICE ADDRESSES
================================================================================
Host Index:  0  Host Type: UHCI  Bus Type: PCI   IRQ#:  3  Root Hub Ports: 2
Vendor: 1106h = VIA Technologies Inc                       Product: 3038h
================================================================================
                DEVICES                                    INTERFACES
============================================  ==================================
                          L                C  I A                O
ADRS                      o          P     o  n l                w
====    (hex)             S          o BUS  n  t t                n
Test VEND PROD     Sub Pro p USB HUB r POWR f  f I                e     Sub Pro
RWak  ID   ID  Cls Cls col d VER ADR t (mA) g  c n  DESCRIPTION   d Cls Cls col
==== ==== ==== === === === = === === = ==== =  = = ============== = === === ===
  1  1106 3038   9   0   0 . 1.0 ... . s  0 1  0 0*Root Hub       Y   9   0   0
     VIA Technologies Inc

  2  058F 6387   0   0   0 . 2.0   1 1  100 1  0 0*SCSI Trsp Bulk Y   8   6  80
     Alcor Micro Corp
```

Nous voyons ici le hub racine (**1106:3038**) et un support de stockage (**058F:6387**). Nous pouvons alors lancer **USBDRIVE** qui se chargera de détecter les supports présents et traduira leur accès en lettre de lecteur. Ceci prend quelques secondes, mais au bout d'un petit temps d'attente, nous pouvons utiliser l'option **S** pour afficher l'état du programme (qui reste en mémoire, c'est un TSR ou programme résident) :

```
C:\> USBDRIVE S
USBDRIVE 0.18, (C) 2007-2009, Bret E. Johnson.
DOS Driver for up to 8 SCSI-compatible USB Mass Storage Devices.

Initialization Delay: 0 seconds
DOS Version:          7.10
Max Bytes per Sector:  512
Mount FAT32 Volumes:   Yes
Max Sectors per Xfer:    1

   USB INFO                        DISK/LUN                          DOS DRIVE
   =========  ================================================  ==========
I                    R W                                                D
n  H   A  I    INT m r                                                  r
d  o   d  n  L 13h v P            Sect Byte                              i
e  s   d  t  U Dsk M r Num   Num  Per  Per    Total     Approx    v   Approx
x  t   r  f  N Num d t Head  Cyl  Cyl  Sect  Sectors   Capacity  e  Capacity
-  -   -  -  - --- - - ----  ---  ---  ----  ---------  --------  -  --------
0  0   2  0  0 81h . .  255  242   63  512  003B7800h  1995 MB   D:   524 MB

Unused USB Dvc/Intf Indexes: 1-3
Unused INT 13h Disk Numbers: 82h-88h
Unused DOS Drive Letters:    E:-K:

Resident USBDRIVE has been updated with new information.
```

Enfin ! Après tant d'efforts, de recherches, de sueur, d'énervement et de souffrance... vient la récompense et le plaisir de retrouver un jeu qui est toujours aussi amusant, loufoque et déstressant, Llamatron 2112 ! (eh oui, DOOM aussi a été installé, bien entendu)

Nous voyons ici qu'un lecteur **D:** est disponible et qu'il donne accès à un espace de 524 Mo sur une clé USB de 2 Go (la partition ici présente était délibérément plus petite que l'espace disponible afin de tester un éventuel impact sur les performances).

Ce lecteur peut être utilisé comme un disque dur secondaire exactement comme vous le faites pour **C:**...

à la différence, dans mon cas, que ceci est d'une lenteur affligeante. Mais ça fonctionne et peut être bien utile dans certains cas.

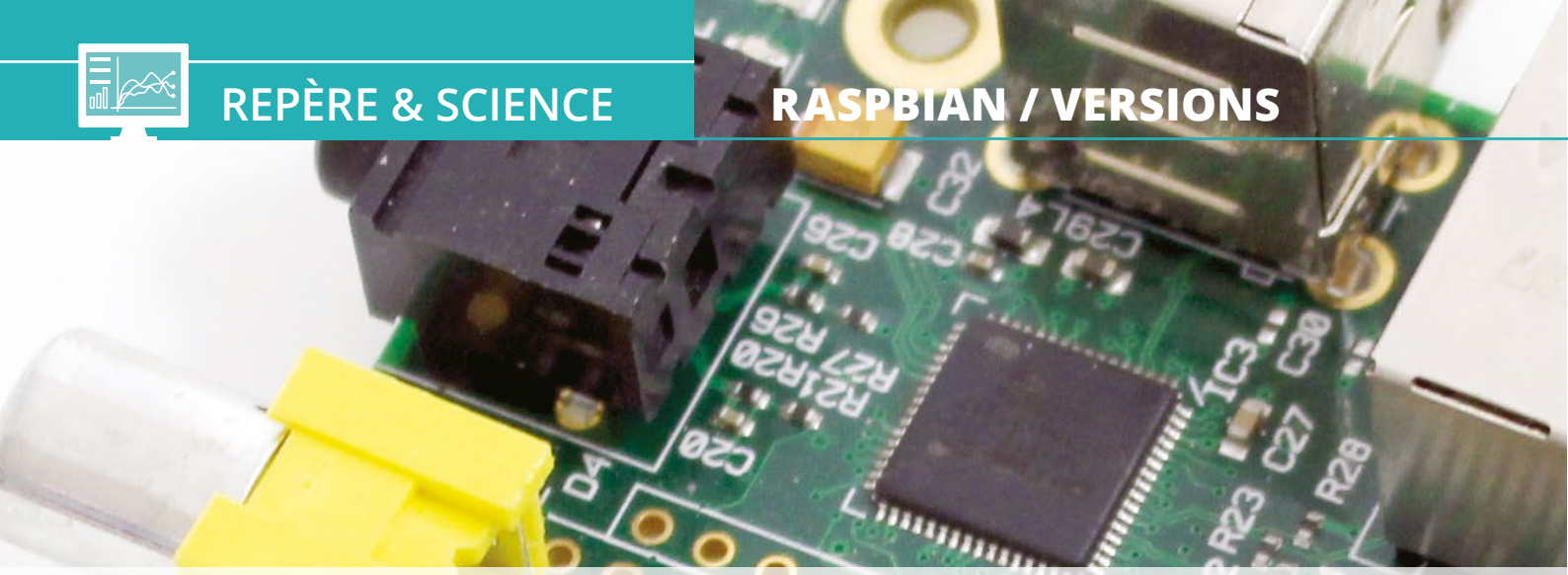
CONCLUSION ET FUTUR

L'objectif initial était de trouver une alternative viable à l'utilisation d'un émulateur afin de profiter de vieux programmes et jeux. Je pense que la démonstration a été clairement faite que, avec un minimum d'efforts, il est parfaitement possible de satisfaire notre nostalgie dans sa teinte la plus originale et surtout sans ralentissement pénible.

Mais nous avons aussi découvert que le bon vieux DOS (et plus exactement FreeDOS) a magnifiquement évolué alors que la plupart des utilisateurs s'en désintéressaient, lui préférant d'horribles interfaces colorées pleines d'icônes et de fenêtres graphiques. Il y aurait encore beaucoup à raconter sur FreeDOS et les logiciels qu'il propose, mais je vous laisserai le plaisir de découvrir cela par vous même. Notez également qu'il s'agit d'un logiciel libre et que les sources (C, C++ et assembleur) de l'ensemble de ces outils sont disponibles et ouverts à l'exploration.

Enfin, précision importante, je tiens à souligner ici que tout ceci ne m'a strictement rien coûté. Je n'ai finalement utilisé que du matériel de récupération que bon nombre de personnes considéreraient comme bon pour le recyclage. Carte mère, CompactFlash, clavier PS/2, écran LCD VGA, alimentation... il ne s'agit que de « restes », de rebuts, de vestiges d'un autre temps.

Bien sûr, à présent que tout fonctionne, il est temps d'investir un peu, opter pour une CompactFlash de plus grande taille, trouver un boîtier sympathique (voire construire un meuble façon borne d'arcade), acheter une alimentation mini-ITX silencieuse... Mais tout ceci est une autre histoire. **DB**



JONGLEZ AVEC LES VERSIONS DE RASPBIAN POUR PROFITER DES APPLICATIONS LES PLUS RÉCENTES

Denis Bodor

Vous devez connaître le syndrome : un logiciel ou un outil dispose des fonctionnalités que vous souhaitez mais, bien entendu, la version disponible dans Raspbian stable n'est pas la bonne, il vous faut la suivante. Très souvent, cette fameuse version tant désirée existe bien, mais elle est dans la future version du système et vous n'avez vraiment pas envie de tout mettre à jour. Pas de problème, rien ne vous empêche de faire vos petits mélanges maison…

Avant de commencer et si vous avez l'habitude de me lire régulièrement, vous devez vous douter qu'il ne sera pas question ici de faire de l'installation sauvage comme une brute épaisse. Jouer du `./configure && make && sudo make install` relève pour moi de la sauvagerie pure et forcer l'installation de paquets n'est pas beaucoup plus élégante. Non, loin de moi l'idée de vous faire faire ce genre de choses, sauf cas d'extrême urgence. Il existe dans Raspbian une solution, tout en finesse, pour bénéficier des paquets dans une version plus avancée que celle proposée par la version stable. Il suffit de judicieusement configurer le système de gestion de paquets APT.

1. UN PEU DE TERMINOLOGIE NE FAIT PAS DE MAL

La distribution Raspbian est un portage de Debian GNU/Linux pour Raspberry Pi. En cela, elle repose exactement sur les mêmes principes, la même logique et les mêmes outils de gestion que Debian GNU/Linux.

Selon cette logique, il existe plusieurs versions d'une distribution :

- La version **stable**, testée et approuvée comme fiable et utilisable en production. À l'heure actuelle, cette version est la 8.0 et possède comme nom de code *Jessie* (la *cowgirl* de *Toy Story* 2 et 3).
- La version **oldstable** (anciennement stable) est tout simplement la précédente version stable. Cette version est la 7.0 et s'appelle *Wheezy* (c'est le pingouin en caoutchouc avec un nœud papillon rouge).
- La version **testing** (en test) qui est en principe assez utilisable pour être testée, mais ne peut être encore considérée comme stable. C'est la future version 9.0 avec, comme nom de code *Stretch* (la pieuvre élastique violette).
- La version **unstable** (instable) est la future version qui pour l'instant n'est pas encore prête à être testée. À cette date, le travail a commencé sur cette version qui **sera** la 10.0 et possède comme nom de code *Buster* (le petit chien qu'Andy reçoit pour Noël dans *Toy Story*, mais qui n'apparaît au spectateur que dans *Toy Story* 2). *Unstable* est aussi *Sid* (le gamin qui aime détruire les jouets dans le premier film et les transforme en monstres). Il faut s'imaginer *Sid* comme une sorte d'incubateur à distribution, un magma dans lequel tout peut changer et tout peut être détruit d'un jour à l'autre. De ce magma, à un moment ou un autre, naîtra la prochaine version *testing* qui sera alors appelée *Buster* et maturera pour devenir, à terme, la version stable. *Sid* en plus d'être le nom d'un personnage de *Toy Story* signifie aussi *Still In Development* (encore en développement) pour bien souligner l'état des paquets qui s'y trouvent et le fait que *unstable* pointe toujours sur *Sid*, contrairement aux autres désignations et noms de code. Notez également qu'il y a « pire » que *Sid*/*unstable* : *experimental*, mais là vous êtes quasiment sûr d'avoir des problèmes...

À un instant donné, il est possible de faire l'association entre nom de code et version (pas le numéro, le nom de la version). Aujourd'hui *stable* est *Jessie*, *testing* est *Stretch*, etc. Mais dès que Debian 9.0 sera publié officiellement, *Jessie* deviendra *oldstable*, *Stretch* sera *stable* et ainsi de suite, sauf *unstable* qui sera toujours *Sid*. Les noms *stable*, *testing* ou encore *unstable* sont souvent désignés comme des « suites » (et tantôt des « catégories ») dans la documentation Debian française, même si souvent on parle de « version » *stable* ou *testing* par exemple.

Pourquoi souligner ce point ? Tout simplement parce que, dans ce qui va suivre et dans les configurations donc nous allons parler, il est possible de désigner les versions par leur nom de code où parle leur version (*stable*, *testing*, *unstable*, *sid*, etc.). En fonction de la désignation que vous utiliserez, vous allez ou non, rester « ancré » avec une version, ou suivre ou non l'évolution des changements de vie des distributions.

2. LES VERSIONS C'EST COMME LA CUISINE, IL FAUT SAVOIR MÉLANGER CORRECTEMENT

Notre objectif ici n'est pas de créer un monstre avec un mélange de tout et n'importe quoi, mais de rester avec une version stable de Raspbian tout en ajoutant, au besoin, des paquets des versions provenant de *testing*.

Configurer la version de votre distribution revient, dans les grandes lignes, à choisir tout simplement d'où viennent les paquets qui la compose. En jetant un œil à la configuration d'une Raspbian fraîchement installée, on trouve dans le fichier `/etc/apt/sources.list` une seule ligne non commentée : **`deb` http://mirrordirector.raspbian.org/raspbian/ `jessie main contrib non-free rpi`**.

Chaque ligne dans ce fichier définit une « source » depuis laquelle le système de gestion de paquets (APT) va récupérer les paquets et les informations les concernant. Ces lignes définissent donc des dépôts où sont stockés ces éléments.

Pour comprendre le sens d'une telle ligne, il faut la découper en détaillant chaque élément :

- **`deb`** désigne le type de dépôt, **`deb`** pour les paquets binaires et **`deb-src`** les sources des paquets, si vous comptez les compiler et construire vous-même.
- **http://mirrordirector.raspbian.org/raspbian/** précise l'URL ou adresse du dépôt. Généralement, il s'agit d'un serveur web (HTTP), mais cela peut également être un serveur FTP.
- **`jessie`** indique la distribution utilisée qui peut être, comme ici, un nom de code, mais peut également être le nom de la catégorie ou suite (*stable*, *testing*, etc.).
- **`main`**, **`contrib`**, **`non-free`** et **`rpi`** indiquent les composants qu'il est possible de prendre dans ce dépôt sous forme de sections. Les paquets sont classés en fonction de leur compatibilité avec les « Directives Debian pour le logiciel libre » (DFSG pour *Debian Free Software Guidelines*). **`main`** est la section regroupant les paquets qui se conforment aux directives, **`contrib`** contient les paquets qui s'y conforment, mais qui ont des dépendances qui ne sont pas dans **`main`** (qui nécessitent donc d'autres paquets en dehors de **`main`**) et enfin **`non-free`** regroupe les paquets qui ne se conforment pas aux DFSG et sont considérés comme n'étant pas des logiciels libres (selon Debian). Ici nous avons, en plus, **`rpi`** qui est une section spécifique à Raspbian, contenant des paquets qui ne concernent que cette plateforme (fut un temps, il s'agissait d'un dépôt distinct).

L'indicateur de distribution ici est très important, car impactant directement le comportement du système de gestion de paquets en cas de mise à disposition d'une nouvelle version de la distribution. En utilisant **`jessie`**, le dépôt utilisé sera toujours le même, même lorsque *Stretch* sera la version stable. Si en revanche vous utilisez **`stable`** pour désigner la distribution, lorsque *Jessie* deviendra *oldstable* et que l'ensemble des paquets de *Stretch* sera regroupé dans le dépôt **`stable`**, vous vous retrouverez avec une ÉNORME mise à jour concernant la quasi-totalité des paquets déjà installés. Ceci n'est pas forcément souhaitable.

Le fichier **`sources.list`** peut contenir plus d'un dépôt et si vous souhaitez pouvoir bénéficier de paquets provenant de *testing* (donc *Stretch* actuellement), vous pouvez ajouter une ligne **`deb` http://mirrordirector.raspbian.org/raspbian/ `stretch main contrib non-free rpi`**.

Après une telle modification, il est nécessaire de mettre à jour la liste des paquets disponibles avec **`sudo apt-get update`**. Ceci fait, votre distribution sera en mesure d'installer et de mettre à jour des paquets provenant de l'actuelle *testing* **MAIS** ce fonctionnement et ce choix sera automatique ! En d'autres termes, maintenant que vous avez accès à deux versions de la distribution Raspbian, en tout logique, le système de gestion de paquets va chercher à installer les paquets les plus récents, des deux distributions !

Si à ce stade vous vous pliez d'un **sudo apt-get dist-upgrade** ou d'un **sudo apt-get upgrade**, tout votre système va passer de *Jessie* à *Stretch* et vous allez donc vous retrouver, si tout se passe bien, avec une Raspbian *testing* en tant que système sur votre Pi. Ce n'est cependant pas l'objectif que nous poursuivons ici et nous devons donc indiquer au système que nous préférons *Jessie* à *Stretch* (même si les pieuvres mauves sont plus amusantes que les cowgirls).

Ceci se fait en configurant justement les préférences d'APT en ajoutant un fichier dans le répertoire **/etc/apt/preferences.d**. Le nom importe peu, mais c'est une bonne habitude de le faire débuter par un chiffre et que le choix du nom soit relativement explicite, **00stable-premier** par exemple. Ce fichier contiendra ceci :

```
# Jessie en priorité maximum
Package: *
Pin: release n=jessie
Pin-Priority: 900

# Stretch en second
Package: *
Pin: release n=stretch
Pin-Priority: 100
```

Nous avons ici trois parties désignant chacune un comportement à adopter. Chaque partie précise un groupe de caractéristiques formant un profil :

- **Package** permet de spécifier les paquets concernés, ici tous (*) ;
- **Pin:** désigne « l'épinglage », on « épingle » la version (*release*) désignée par un nom de code (**n=**), respectivement **jessie** et **stretch** (vous pouvez aussi utiliser **a=** pour spécifier **stable** ou **testing**) ;
- **Pin-Priority:** indique la priorité à utiliser dans la sélection de paquets, plus elle est importante, plus elle « passera » avant une autre, avec ici 900 pour *Jessie* et seulement *100* pour *Stretch*.

L'effet de cette configuration sera d'inverser la logique qui fait qu'un paquet plus récent sera choisi pour l'installation. Cela reste vrai pour *Jessie*, mais les paquets, même avec une version plus importante, de *Stretch* ne seront plus prioritaires. En conséquence, après enregistrement de ce fichier, une commande comme **sudo apt-get upgrade** n'installera plus de paquets provenant de *Stretch*.

Ceci signifie également qu'en utilisant **sudo apt-get install** suivi d'un nom de paquet, c'est la version la plus élevée disponible pour *Jessie* qui sera installée. À titre d'exemple, si nous prenons le paquet **vim**, la version 2:7.4.488-7+deb8u1 est disponible pour *Jessie* et la version 2:8.0.0197-1 pour *Stretch* et ainsi :

```
$ dpkg -l vim
[...]
ii  vim  2:7.4.488-7+deb8u1  armhf Vi IMproved

$ sudo apt-get install vim
[...]
vim est déjà la plus récente version disponible.
0 mis à jour, 0 nouvellement installés,
0 à enlever et 0 non mis à jour.
```

Nous pouvons demander une installation d'une version précise du paquet en faisant suivre son nom d'un **=** et du numéro de version choisie :

```
$ sudo apt-get install vim=2:8.0.0197-1
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Certains paquets ne peuvent être installés.
Ceci peut signifier que vous avez demandé
l'impossible, ou bien, si vous utilisez
la distribution unstable, que certains paquets
n'ont pas encore été créés ou ne sont pas
sortis d'Incoming.
L'information suivante devrait vous aider à
résoudre la situation :
```

```
Les paquets suivants contiennent des dépendances non satisfaites :
vim : Dépend: vim-common (= 2:8.0.0197-1)
          mais 2:7.4.488-7+deb8u1 devra être installé
      Dépend: vim-runtime (= 2:8.0.0197-1)
          mais 2:7.4.488-7+deb8u1 devra être installé
      Dépend: libtinfo5 (>= 6)
          mais 5.9+20140913-1 devra être installé
E: Impossible de corriger les problèmes, des paquets défectueux
  sont en mode " garder en l'état ".
```

Ici, ceci ne fonctionne pas pour une raison très simple : cette version du paquet **vim** nécessite l'installation d'autres paquets, dans une version différente de celles disponibles dans *Jessie*. Ainsi, comme nous n'avons pas donné la priorité à *Stretch*, les versions sélectionnées par défaut sont celles de *Jessie* et non de *Stretch*, et ceci n'est pas compatible.

Nous avons effectivement « demandé l'impossible » et le message d'erreur est parfaitement explicite. Pour régler ce problème, il faut demander à **apt-get** de « viser » (*target*) la bonne distribution, en utilisant l'option **-t** et en précisant **stretch** :

```
$ sudo apt-get install -t stretch vim
[...]
Les NOUVEAUX paquets suivants seront installés :
  xxd
Les paquets suivants seront mis à jour :
  libncurses5 libncursesw5 libtinfo5 vim
  vim-common vim-runtime vim-tiny
[...]
Souhaitez-vous continuer ? [O/n]
```

Là, nous n'avons même plus besoin de préciser la version du paquet à installer, le simple fait d'utiliser **-t** place **apt-get** dans un contexte où il pourra installer le paquet et ses dépendances en considérant uniquement *Stretch*/. Bien entendu, une confirmation vous sera demandée et il est important de bien comprendre qu'un retour en arrière est impossible (ou du moins sera souvent très très problématique).

Le fait d'utiliser des préférences personnalisées pour APT et de pouvoir accéder aux paquets de deux versions de la distribution peut être très avantageux. Tantôt pour régler un problème et parfois simplement par curiosité, comme ici avec Vim 8.0 et tester les nouvelles fonctionnalités disponibles. Il faut cependant utiliser cela avec prudence et parcimonie, car si *testing* est appelé ainsi, et non *stable*, ce n'est pas par hasard. Les paquets qui s'y trouvent n'ont pas tous atteint le niveau de qualité requis pour être considérés comme stables. Les éventuels problèmes relèvent donc de VOTRE responsabilité.

Bien entendu tout ce que je viens de décrire ici est applicable pour *unstable*. Il est donc possible de configurer votre distribution pour bénéficier de toutes les versions possibles, tout en gérant les priorités de façon raisonnable et en installant les paquets en connaissance de cause. Il est cependant fortement déconseillé d'utiliser des paquets d'*unstable* pour un projet pour des raisons évidentes. En revanche, sur une Pi de test, cela peut être très intéressant... **DB**

INNO ROBO EVENT

16-18 MAI 2017

PARIS FRANCE

WHERE INNOVATION GROWS

Plongez au **cœur** de l'**innovation robotique**

Chercheurs, créateurs et utilisateurs vous dévoilent le monde de demain

UNE APPROCHE HUMAINE DE LA ROBOTIQUE

SMART HOMES

INDUSTRY 4.0 & SUPPLY CHAIN SERVICES

MEDICAL & HEALTH

TECHNOLOGIES & FORESIGHT

SMART CITIES

FIELD ROBOTICS

HOSPITALITY RETAIL & TOURISM

www.**innorobo**.com

YouTube theinnorobo

Maker Faire® Paris

9 > 11 juin 2017

Cité des Sciences et de l'Industrie

Appel aux Makers

JUSQU'AU 30 AVRIL

Maker Faire est à la fois une fête de la science, une foire populaire et un événement de référence pour l'innovation. Ce concept regroupe stands de démonstration, ateliers de découverte, spectacles et conférences autour des thèmes de la créativité, de la fabrication, et des cultures Do It Yourself.
Aujourd'hui, plus de 200 éditions réunissent dans 38 pays **des communautés de passionnés, experts ou débutants, qui partagent l'envie de créer, fabriquer et apprendre les uns des autres.**
La 4ème édition de **Maker Faire Paris** se tiendra à la Cité des Sciences et de l'Industrie.

Vous souhaitez participer ?
Rejoignez la communauté des Makers en vous inscrivant à notre appel !

Clôture des inscriptions le 15 avril 2017 à minuit !

Un événement
Make: makezine.com

Présenté par
LEROY MERLIN ...et vos envies prennent Vie !

Partenaire
cité des sciences & de l'industrie

Inscription sur
paris.makerfaire.com