N°21 | NOVEMBRE - DÉCEMBRE 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

VINTAGE / MINITEL

## Redonnez vie à un Minitel 1B et affichez textes et images en vidéotex avec votre Arduino

p. 10

CHARGEUR / SANS FIL

## Utilisez la technologie Qi pour alimenter/charger vos montages comme des smartphones

p. 72

RÉSEAU / ZEROCONF

## Contactez vos montages réseau par leur nom grâce à mDNS sur Raspberry Pi et ESP8266

p. 60

ESP8266 / OTA

## Oubliez l'USB et mettez directement à jour le code de vos montages via Wifi avec l'OTA

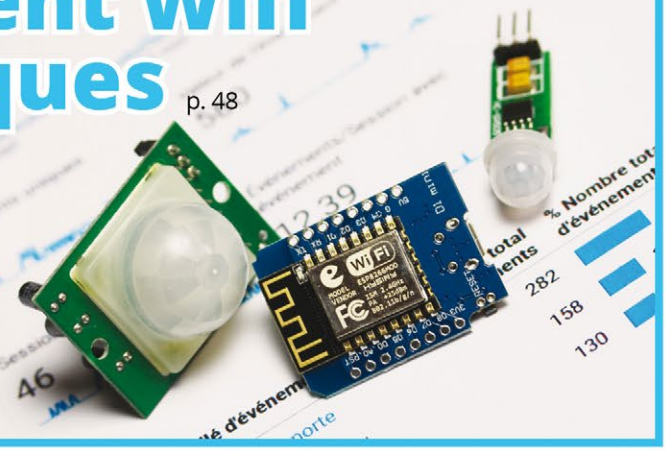
p. 38

ESP8266 / Google / Domotique

## Créez des capteurs de mouvement Wifi économiques

p. 48

- Utilisez un capteur de mouvement infrarouge
- Collectez et envoyez les informations sur le net
- Enregistrez les évènements dans Google Analytics
- Traitez votre environnement physique comme un site web



WEARABLE / DIY

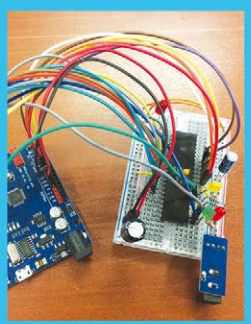
## Créez votre t-shirt interactif avec une carte Arduino Lilipad, des leds et un peu de couture

p. 26

RÉTRO / Z80

## Simulez la mémoire pour votre Z80 avec votre carte Arduino et écrivez votre premier code assembleur

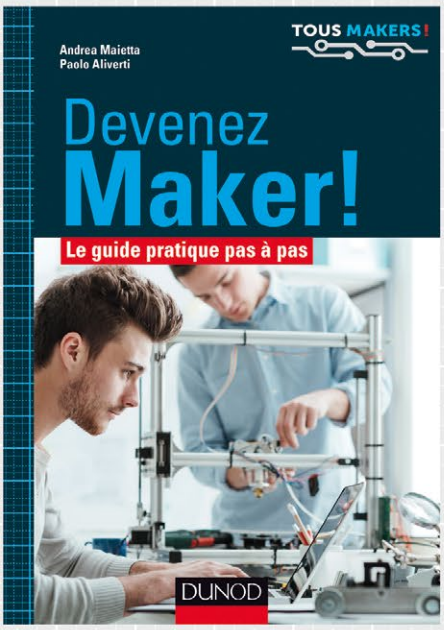
p. 82



L 19338 - 21 - F: 7,90 € - RD

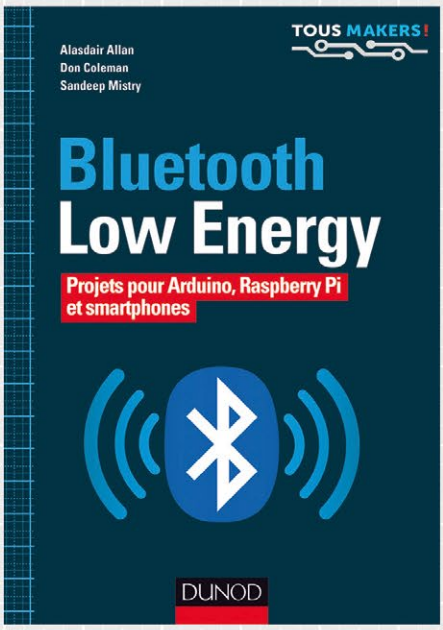
# TOUS MAKERS !

## RÉVÉLEZ VOTRE POTENTIEL PAR LA CRÉATION



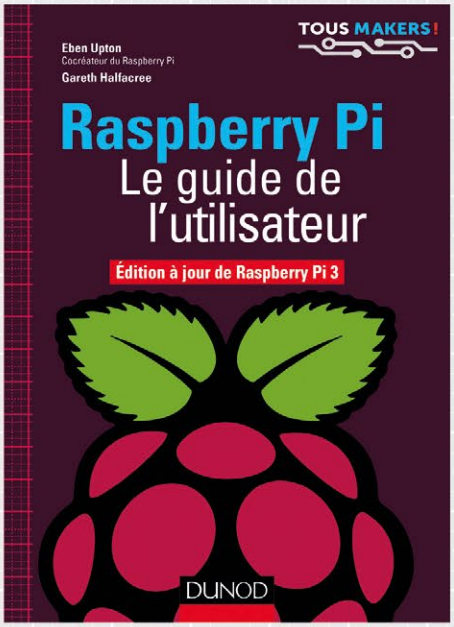

9782100762934, 304 pages, 24,90 €
ANDREA MAIETTA, PAOLO ALIVERTI
Comment transformer vos idées en projets concrets ? Ce livre vous accompagne dans la réalisation de vos premières créations.


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

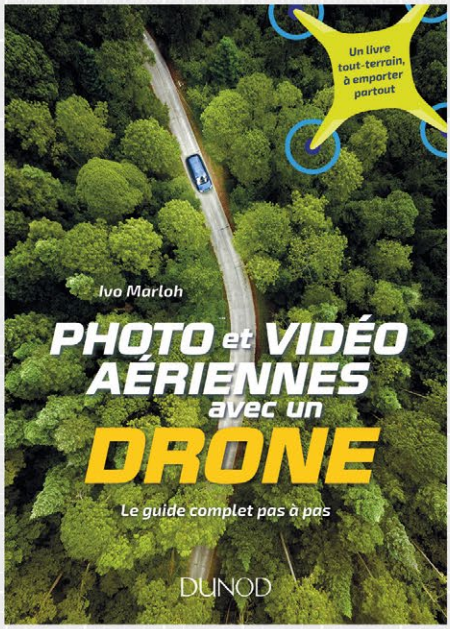



9782100760855, 272 pages, 29 €
ALASDAIR ALLAN ET AL.
Maîtrisez la nouvelle technologie Bluetooth Low Energy en réalisant les différents projets détaillés dans cet ouvrage.


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.




9782100757992, 160 pages, 19,90 €
IVO MARLOH
Toutes les clés pour filmer et photographier avec son drone en toute sérénité.



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

DUNOD
ÉDITEUR DE SAVOIRS

# ÉDITO

Le passé commence à valoir cher !

Lorsque cela fait près de 25 ans qu'on nage dans la technologie en suivant les marées, les remous et les courants, parfois, des choses vous échappent. On évolue en même temps que les produits, les logiciels et les ressources qu'on utilise et sans petites piqûres de rappel de temps en temps, on en arrive, comme beaucoup, à prendre pour argent comptant tout ce qui se trouve à notre disposition aujourd'hui.

Une façon de relativiser consiste à se pencher sur de vieux matériels et de vieux systèmes. Chose que j'ai dernièrement fait en offrant à ma station Silicon Graphics Indy un écran via un adaptateur 13w3/VGA et en y installant IRIX 6.5. La réalité m'est apparue, après maintes reconfigurations, en pouvant finalement lancer Netscape Navigator 4.79 : la quasi-totalité des sites sont tout bonnement illisibles ou inaccessibles, au mieux ils sont tellement « lourds » qu'ils mettent une éternité à s'afficher.

Une autre façon consiste à faire un tour sur eBay dans la section « Informatique vintage » et d'observer les prix. Les ordinateurs des années 80/90 valent cher, parfois très cher, mais ce n'est rien comparé aux composants, périphériques et pièces détachées pour ces machines. Et lorsqu'on arrive à des éléments plus spécifiques, comme par exemple un hub Ethernet proposant RJ45 (10BASE-T) **et** BNC (10BASE2), la solution la plus simple pour connecter une machine 10BASE2 à un réseau moderne, cela devient tout simplement surréaliste !

Le phénomène est fascinant. Des machines « inaccessibles » à leur époque sont temporairement devenues obsolètes et presque sans valeur, et sont maintenant en passe de devenir, en compagnie des machines et matériels d'antan plus populaires, à nouveau inaccessibles. Les portes du temps semblent se refermer sur cette période de l'histoire informatique, mais cette fois dans le bon sens du terme (pour « ceux qui gardent »).

Si vous en doutez, je vous conseille, comme je l'ai conseillé à ce lecteur qui m'a contacté et ayant un IBM PC XT complet dans son garage, de faire un tour dans les annonces de ventes et les enchères. Toutes ces merveilles n'ont plus QUE de la valeur sentimentale, les premières heures de l'informatique personnelle pour tous sont entrées dans leur phase « pièces de collection ». Et je trouve que c'est une très bonne chose...

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 : https://www.hackable.fr/
https://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

Alsace

## 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.

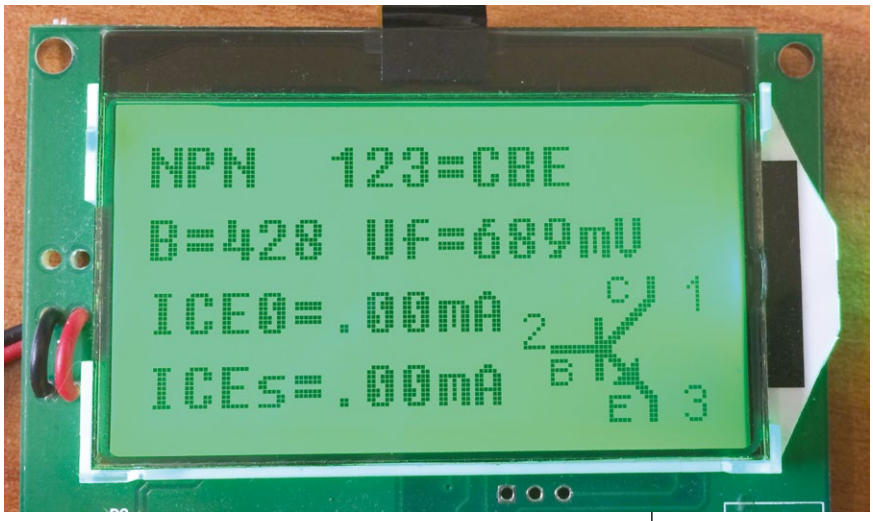


# SOMMAIRE

# UN TESTEUR UNIVERSEL DE COMPOSANTS POUR 15€ ?

*Denis Bodor*

**Il y a deux éléments dans cette question qui sont troublants : le mot « universel » et le prix. Certes, un équipement comme un multimètre est un outil capable de tester et mesurer plusieurs types de caractéristiques, mais il n'en est pas pour autant « universel ». Et, à 15€, on ne peut que supposer que le matériel en question ne sera sans doute pas un modèle de précision ou de qualité. Instinctivement, la réponse la plus logique semble donc être « non ». Pourtant, dans ce cas précis, on peut mitiger sa réponse…**




Ce genre de testeur se trouve un peu partout en vente sur le net, des boutiques en ligne aux sites d'enchère en passant pas les classiques plateformes de vente directe depuis les lieux de production. « Testeur de composants », « testeur universel », « testeur de transistors »... la désignation importe peu puisque chacun semble y mettre son grain de sel tant sur la désignation que sur l'adaptation du projet d'origine. La recette « magique » pour ce type de matériel est cependant toujours la même : un microcontrôleur Atmel couplé à un afficheur LCD (alphanumérique ou graphique), quelques composants passifs et actifs et un support ZIF permettant de placer/retirer facilement des composants.

L'idée, elle aussi, est toujours la même : le code embarqué dans le microcontrôleur utilise ses entrées/sorties pour tester n'importe quel composant qui lui est présenté, il en déduit le type, le brochage et mesure les valeurs de ses caractéristiques.

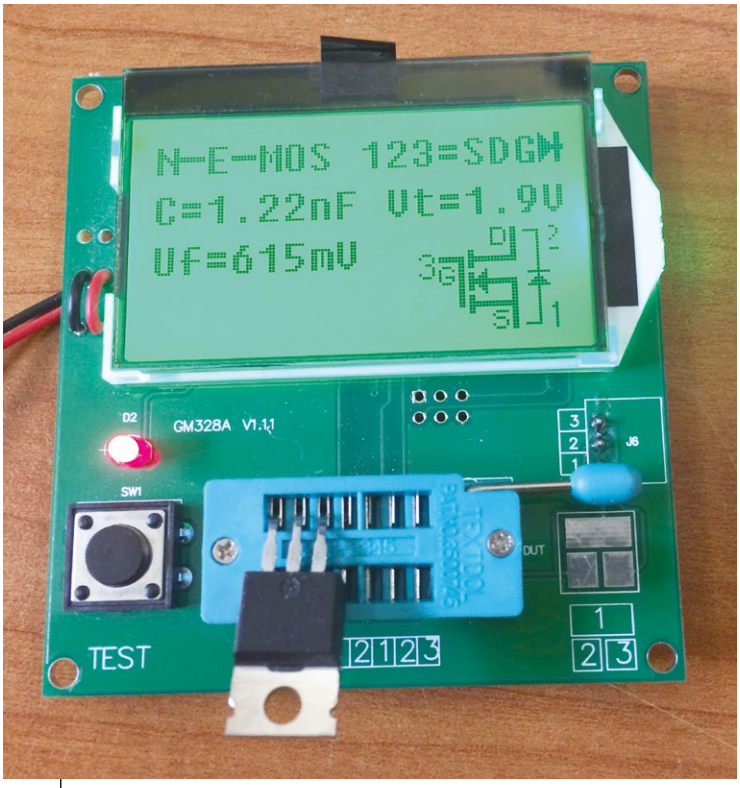
## 1. GM328A, L'IDÉE ORIGINALE

Le testeur de composants que l'on trouve sur presque tous les sites, d'Amazon à Banggood en passant par eBay ou Tindie, semble s'être diversifié au point ne plus plus vraiment voir un quelconque ancêtre commun. Du moins c'est ce que certains fabricants espèrent en effaçant toutes traces d'un quelconque héritage et présentant presque systématiquement le produit comme une création originale.

En réalité, 99% de ces « testeurs » sont des évolutions des travaux de Markus Frejek, repris par Karl-Heinz Kübbeler et enfin Markus Reschke jusqu'à ce jour. Ce qui était à l'origine un « Transistortester AVR » a évolué au fil des années et des améliorations, en un testeur universel de composants ne se limitant plus aux transistors et incluant le test de condensateurs, de résistances, de FET, de triacs, de thyristors, de bobines, de diodes, etc.

Retracer l'historique d'un tel projet, en particulier lorsque les constructeurs ayant réutilisé le circuit et le code dans le but évident d'en masquer l'origine, n'est pas facile, mais on arrive finalement sur **mikrocontroller.net** afin de rendre à César ce qui appartient à César, et saluer le travail des vrais créateurs de l'outil. Légalement parlant, le créateur original du projet, M. Frejek n'a pas spécifié de licence d'utilisation tout en précisant qu'il s'agissait d'open source et qu'une utilisation commerciale nécessitait une prise de contact préalable. C'est K.H. Kübbeler et M. Reschke qui, n'arrivant pas à contacter M. Frejek et ayant réécrit une très vaste partie du code, ont décidé en 2016 que la licence serait l'EUPL v1.1, une licence de logiciel libre, approuvée par la Commission européenne et compatible GPL, OSL, Eclipse, etc.

Les utilisateurs les plus curieux pourront donc remonter aux sources en pointant leur navigateur sur **https://www.mikrocontroller.net/articles/AVR_Transistortester** afin de récupérer les sources du firmware, ce qui leur permettra éventuellement de faire évoluer leur matériel, sous réserve d'identifier clairement ce que le fabricant aura changé dans le concept et le circuit.




## 2. IMPORTANT : CECI NE REMPLACE PAS UN MULTIMÈTRE !

L'utilisateur ou le hobbyiste ayant une certaine expérience du domaine n'aura sans doute pas besoin qu'on lui rappelle, mais exactement comme cela est précisé dans les sources de Markus Reschke, un tel produit n'est **pas une solution de remplacement pour un multimètre numérique** ou tout autre vrai équipement de mesure électronique.

C'est un testeur de composant qui, comme nous le verrons, possède ses qualités et son utilité, mais en aucun cas un appareil de mesure fiable. Aucun produit découlant de ce projet, même lorsqu'il est joliment présenté dans un boîtier, ne possède ne serait-ce qu'un semblant de protection. Le produit ne survivra certainement pas à des tensions de plus de 5 volts et n'est absolument pas destiné à voir connecter à ses broches une quelconque source de courant.

Répétons-le donc clairement : **ceci n'est pas un substitut pour un multimètre de qualité**, c'est un testeur de composants dont le travail consiste à identifier ces derniers et procéder à un certain nombre de mesures approximatives.

## 3. TESTS ET UTILISATION

Pour les tests, j'ai acquis deux déclinaisons du testeur, via eBay :

- Un modèle vendu sous la désignation « *All-in-1 Component Tester Transistor Diode Capacitance ESR Meter Inductance BoBo* », par « d-light-factory » au prix de 12,68€ port inclus.
- Un autre, désigné par « *Component Tester Transistor Diode Capacitance ESR Meter Mosfet NPN PNP Inductanc* » par « haiy_95inliao », un peu plus cher à 18,53€, port inclus.

Les deux modèles se présentent de la même manière et possèdent certaines caractéristiques identiques. Il s'agit de matériel basé sur un microcontrôleur Atmel ATmega328P à 8 Mhz, comprenant un écran LCD graphique de 64×128 pixels rétro-éclairé, un support ZIF permettant d'enficher les composants, une alimentation par pile 9V, d'une série de trois contacts pour les composants de surface, une led témoin d'alimentation et un emplacement pour souder un connecteur 2×6 broches ICSP (pour reprogrammer l'ATmega328P).

Les deux versions (et il en existe bien d'autres) se différencient surtout par le circuit lui-même. Bien qu'assez similaires dans les grandes lignes, on voit clairement que des choix différents ont été faits. Le modèle le plus cher, avec un circuit imprimé rouge, possède un bouton sup-

plémentaire permettant d'éteindre le dispositif, place l'écran sur une carte fille tenue par quatre entretoises en métal, propose un emplacement pour un connecteur d'alimentation et, de façon générale semble de meilleure qualité et plus robuste.

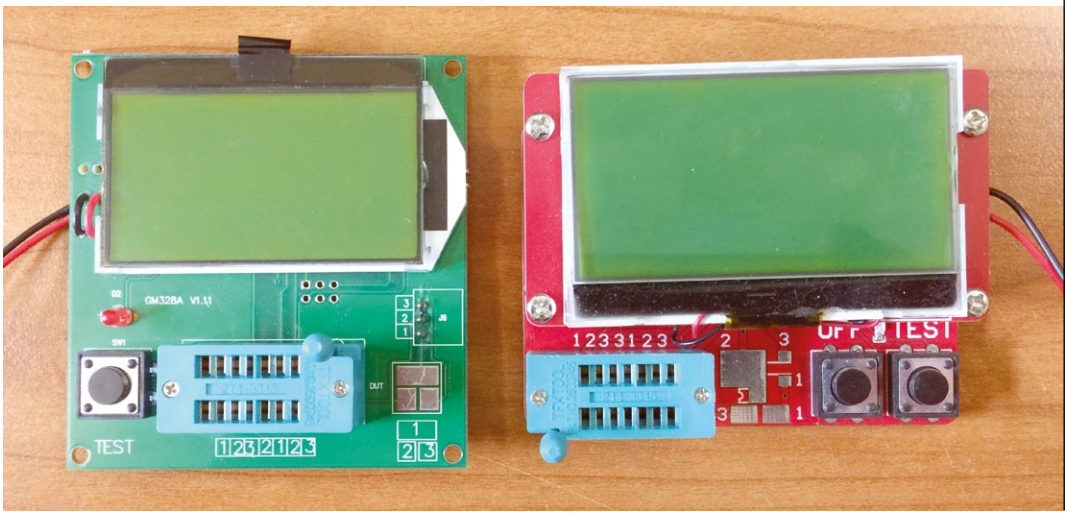
Le premier modèle, quant à lui, semble plus proche du concept original, ne proposant qu'un seul bouton servant à la mise en fonction et au test des composants. L'extinction repose donc entièrement sur l'arrêt automatique ou la déconnexion de la pile. L'afficheur LCD est soudé directement sur le circuit imprimé dans une position qui est relativement fragile. On notera également que le circuit imprimé porte la mention GM328A et qu'un des messages s'affichant à l'écran fait mention de **mikrocontroller.net**.

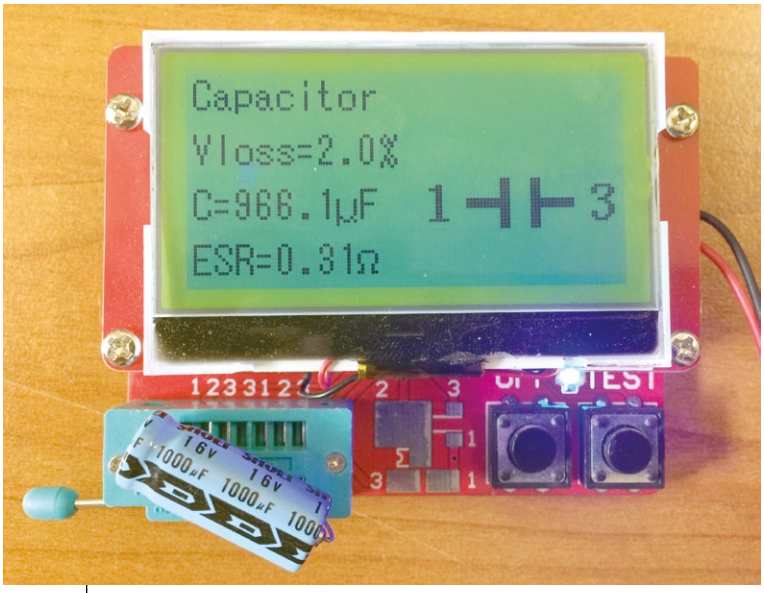
Côté firmware, nous avons également une grosse différence. Le matériel au circuit rouge bien que de meilleure qualité fait l'impasse sur certaines fonctionnalités complémentaires présentes dans les évolutions récentes du code, et ce malgré la mention dans la description du produit de « latest M328 version of the software, more functions » (« dernière version du logiciel M328, plus de fonctions ») tout en précisant « 2013 » par ci et « 2014 » par là (la dernière version libre, 1.29m, date en réalité de juin 2017).

La première impression laissée par ces produits n'est pas forcément très engageante. On voit clairement qu'il s'agit d'une reprise approximative du projet original pour en faire un matériel pouvant être produit à bas coût et en grande quantité. Dans les ceux cas, aucun effort n'est réellement fait pour valoriser l'origine du concept ou permettre une quelconque mise à jour aisée.

Dans la pratique, les deux modèles se valent (et pour cause, le code est mature et bon). Il suffit ainsi de placer un composant sur le support ZIF et de presser le bouton « TEST ». Après une brève initialisation et l'affichage d'un message, le montage teste le composant pour en déterminer le type puis affiche les résultats. Le support ZIF possède 14 connecteurs, mais il ne s'agit que d'une répétition des 3 lignes disponibles. Un marquage reprend la nomenclature qui sera ensuite réutilisée à l'écran pour désigner les points de connexion. Ceci permet de placer un composant où qu'on veuille en fonction de son format tout en utilisant toujours les broches 1 et 2, ou 1, 2 et 3.

Si nous plaçons, par exemple, une led entre 1 et 2 et procédons au test, un petit dessin d'une diode apparaît nous permettant déjà de déterminer rapidement anode






et cathode, si les deux pattes ont la même taille (suite à récupération par exemple). Dans le cas d'une led, le courant direct ne peut être déterminé/deviné, mais la tension, elle, est mesurée, ainsi que la capacité.

Pour les composants comme les transistors, cela s'avère encore plus intéressant car, là aussi, le brochage est détecté sans peine. Le testeur ne vous dira pas de quel composant il s'agit, mais vous donnera cependant des caractéristiques intéressantes. En plaçant ainsi un transistor NPN BC547b assez courant et standard, les deux modèles nous indiquent plus ou moins les mêmes choses :

- c'est un transistor NPN (noté « BJT-NPN » ou « NPN » à l'écran) ;
- la base est sur 2, le collecteur sur 1 et l'émetteur sur 3 ;
- le gain (noté « B » ou « hFE ») est de 428 (entre 125 et 900 dans la documentation) ;
- la tension de base/émetteur (notée « Vf ») est de 694mV ou 0,694V (Vbe est donné entre 0,55 et 0,72 dans la doc).

Une diode nous donnera également des résultats intéressants en nous précisant bien entendu la polarité, mais également la tension directe, comme par exemple 0,684V sur la diode 1N4007 testée, ce qui est tout à fait dans les spécifications.

Les résistances donneront un résultat similaire en indiquant bien entendu la valeur en ohms avec une précision tout à fait acceptable. Un point très intéressant ici, il est possible de mesurer deux résistances à la fois en les plaçant entre 1 et 2, et entre 2 et 3. Ceci permettra également de mesurer facilement un potentiomètre. Il aurait été intéressant, en plus de ces informations très factuelles, que le rapport proportionnel soit également affiché en simulant un montage en diviseur de tension, nous évitant ainsi de faire des calculs.

Les condensateurs peuvent également être testés, mais il faudra faire très attention à leur éventuelle charge, car le testeur pourrait ne pas survivre le cas échéant. Court-circuitez donc systématiquement vos condensateurs avant de les tester. La polarité n'est pas détectée/affichée avec les condensateurs électrolytiques par exemple, mais la valeur en farad est mesurée ainsi que l'ESR. Pour rappel, l'ESR ou *Equivalent Series Resistance* (en bon français « résistance série équivalente ») est une représentation de la résistance d'un composant, généralement sur un courant alternatif.

L'ESR est un peu valeur de la différence entre théorie et pratique dans le cas d'un condensateur. Idéalement, un tel composant ne présente pas de résistance, mais étant composé de matériaux qui présentent une résistance finie, dans les faits, il présente effectivement une équivalence de résistance. Cette valeur généralement mesurée en dixième ou centième d'ohm, varie en fonction de la tension et de la capacité du condensateur. Là où cela devient très intéressant, et en particulier dans le cas des condensateurs électrolytiques, c'est qu'en vieillissant ou en étant régulièrement exposés à une source de chaleur, l'ESR augmente.

Qui dit résistance, dit généralement dissipation thermique et donc condition favorable à l'augmentation de l'ESR, jusqu'à détruire le composant qui souvent est gonflé sinon ouvert.

Tester l'ESR d'un condensateur électrolytique permet donc de vérifier son état. Si vous faites des réparations ou de la récupération, et sachant que ces condensateurs sont généralement la source de tous les maux, le testeur pourra vous être utile. Attention cependant : il n'est pas question de l'utiliser avec des composants *in situ* et surtout, cela ne remplacera pas un appareil dédié digne de ce nom (un RLC mètre ou *LCR meter* en anglais). Mais si une ESR importante est affichée (supérieure à 1/4 d'ohm), le composant est certainement bon pour la poubelle.

## 4. MAIS EST-CE QUE ÇA VAUT LE COUP ?

Ou plus exactement, « le coût », dans ce cas précis. La réponse est affirmative, mais uniquement comme équipement d'appoint et certainement pas comme solution de remplacement pour un bon multimètre. Ces testeurs sont vraiment très pratiques dans le cadre de ce pour quoi ils sont conçus : identifier et tester des composants. Voyez cela comme un chausse-pied, un testeur vous donnant des bases pour ensuite compléter, le cas échéant, en procédant à des mesures plus précises avec un appareil spécialisé.

À titre personnel, l'intérêt réside principalement dans le test des transistors et des MOSFET, composants actifs que j'utilise le plus souvent. Pour les condensateurs, l'ESR et les bobines, j'ai préféré opter pour un RLC mètre (Keysight U1733C), mais ceci est un choix tout personnel, principalement motivé par le fait que je fais de plus en plus de restauration d'ordinateurs vintages...

Le mot de la fin sera donc « oui ». Oui, cela vaut le coup d'investir 15€ pour ce type de produit. **DB**

# UTILISEZ UN MINITEL COMME ÉCRAN POUR VOTRE ARDUINO

*Denis Bodor*

**Le Médium Interactif par Numérisation d'Information Téléphonique, ou Minitel, rappellera sans doute bien des souvenirs à nombreux d'entre vous : son utilisation, son clavier mécanique, les services proposés, les 3615 suivis d'un prénom féminin, le bruit caractéristique de la porteuse du modem et le sifflement éprouvant de l'alimentation du tube cathodique… Le Minitel, avec ses 30 ans de service, est un artefact de la culture et de la technologie française, mais annoncer sa mort en 2012 était prématuré. Ce vénérable équipement peut toujours être utilisé… avec une carte Arduino.**

Je me souviens de l'été 1998, alors que je bidouillais et écrivais les articles du tout premier numéro de *Linux Magazine France* (on a ajouté le « GNU/ » ensuite), je découvrais les petits secrets du Minitel 1B et sa prise péri-informatique. Il était possible de l'utiliser comme un terminal pour son PC sous GNU/Linux, en utilisant une séquence spéciale de touches afin de le faire basculer en 80 colonnes tout en supportant une plus haute vitesse de communication (Fnct+T puis A, Fnct+T puis E et Fnct+P puis 4).

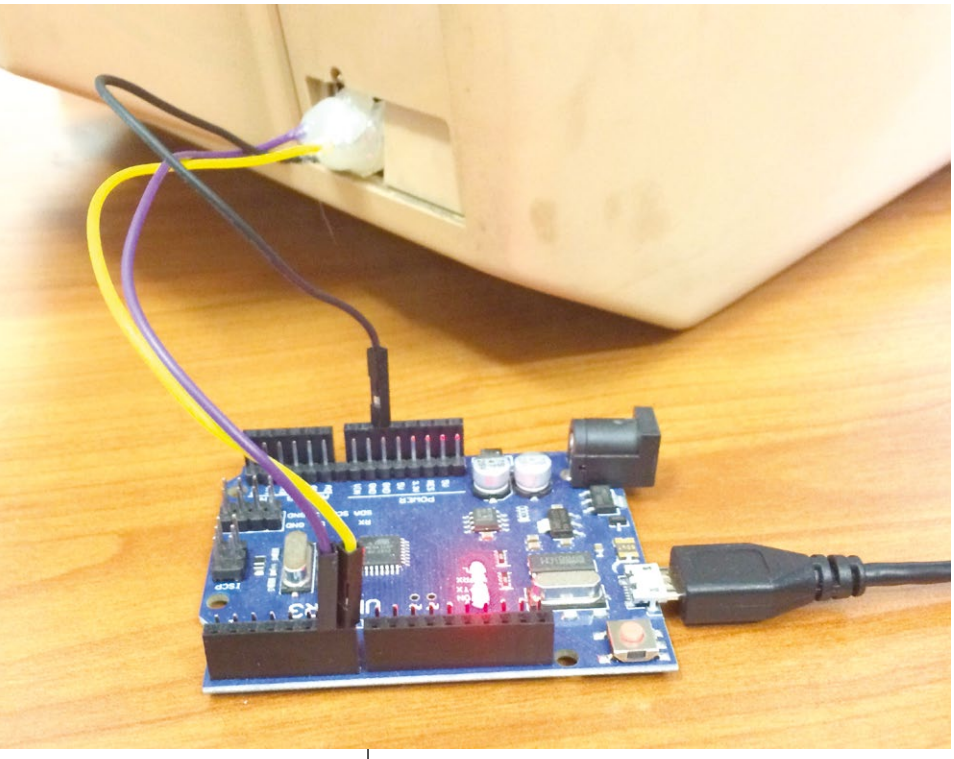
À l'époque, l'USB et les cartes Arduino n'existaient pas. Les PC étaient équipés de ports série RS-232 avec des niveaux de tension incompatibles avec le Minitel (5V) et il fallait assembler un petit circuit de conversion (avec des transistors ou un MAX232). C'est au détour d'un marché aux puces l'été dernier et en découvrant un Minitel sur un étal, que l'idée m'est venue, « *il n'y a plus besoin de conversion, le minitel et une carte Arduino sont en 5V* », et que la décision fut prise, « *Ok, si c'est un 1B et que la personne me le fait à 20€ ou moins, je le prends* ».

Et ce fut le cas. Je rentrais donc chez moi, tout heureux de mon acquisition et de pouvoir vérifier, par la même occasion, une certaine théorie auditive. En effet, l'alimentation THT du tube cathodique de l'appareil émettait souvent un sifflement à une fréquence très élevée. Or cette fréquence est à la limite des capacités auditives humaines et, en prenant de l'âge, elle devient imperceptible sinon totalement inaudible. Et effectivement, mon Minitel ne siffle plus en ce qui me concerne. Donc, soit je suis tombé sur un Minitel qui ne siffle pas, soit je n'arrive plus à entendre cette fréquence du haut de mes 43 ans. Dans les deux cas, cela me va très bien !


*Voici un Minitel 1B Telic Alcatel en provenance directe du marché aux puces du coin. Il a besoin d'un petit nettoyage, mais fonctionne parfaitement et pourra servir à la fois de terminal série en 80 colonnes et de système d'affichage Vidéotex pour n'importe quel projet à base d'Arduino ou de Raspberry Pi.*

## 1. UN PETIT MOT SUR LE MINITEL

Je ne m'attarderai pas sur le sujet, la page Wikipédia sur le Minitel (**https://fr.wikipedia.org/wiki/Minitel**) décrit de façon très complète l'histoire et la technologie utilisée. Il est cependant amusant de noter qu'à la fin des années 70 certains n'hésitaient pas à annoncer la



*La connexion de la carte Arduino passe par la prise péri-informatique située à l'arrière du Minitel. Attention, tous les modèles n'en possèdent pas, mais celui le plus facile à trouver à ce jour, le Minitel 1B, en est heureusement équipé.*

mort du papier devant l'arrivée d'un nouveau type de support : la télématique, une véritable révolution technologique. Mais le papier est toujours là et reste le seul support durable, pratique, capable de retenir une information sur une très longue période, et ce indépendamment d'une source d'énergie ou d'une technologie complexe. Bref, on a pas encore trouvé mieux pour « enregistrer » du texte et des dessins pour un sacré bout de temps. Finalement, le Minitel n'a pas tué le papier, mais Internet, lui, a bel et bien tué le Minitel, sans toutefois, lui non plus, tuer le papier.

Après de nombreuses expérimentations, en Bretagne, en Île-de-France ou encore en Alsace, le Minitel est finalement lancé commercialement à l'échelle nationale au début des années 80 par les PTT. Il se connecte, via le réseau téléphonique, à un réseau appelé Télétel fonctionnant en Vidéotex. Dans l'absolu, le Minitel n'est qu'un équipement permettant de communiquer en Vidéotex, norme Antiope, sur le réseau Télétel, mais rapidement tout le monde a fini par parler de « services Minitel », de « serveur Minitel » et de « Minitel » comme d'un ensemble. C'est en réalité le Vidéotex qui permet l'envoi de pages (écrans) composées de texte et de caractères semi-graphiques et donc l'utilisation du service.

Un Minitel est un appareil relativement simple composé d'un écran cathodique, d'une alimentation, d'un modem V.23 et d'un circuit basé sur le processeur Intel 8052, une RAM pour la mémoire de l'affichage et une ROM contenant la forme des caractères. Contrairement à un terminal série, qui généralement n'utilise que les symboles des lettres (plus quelques autres), le Vidéotex utilise un mode alpha-mosaïque où on peut afficher non seulement les caractères ASCII, mais également des symboles d'un alphabet graphique ou mosaïque. Cet alphabet est composé de caractères de deux colonnes de points de 6 lignes, regroupant ainsi les 64 combinaisons possibles de remplissage de cette « grille ».

C'est grâce à cela qu'il était possible d'avoir des pages présentant un graphisme, aujourd'hui assez rudimentaire, s'affichant en réalité sur un écran de 25 lignes de 40 colonnes (et donc de 75 lignes de 80 pixels). Cette économie d'un vrai mode graphique a permis de produire des terminaux peu chers tout en s'offrant au passage de la place pour un peu de couleur. Un Minitel est capable d'afficher 8 couleurs même si celles-ci n'apparaissent que comme des niveaux de gris sur l'écran noir et blanc (du plus clair au plus foncé : blanc, jaune, cyan, vert, bleu, rouge, magenta et noir). Il est amusant de noter qu'avec la popularisation des modems peu coûteux et des émulateurs Minitel pour PC, bon nombre de services ont été obligé de réviser leurs pages, car ce qui pouvait être du plus bel effet

en 8 niveaux de gris sur un Minitel devenait absolument horrible en couleur dans un émulateur.

Il existe plusieurs modèles de Minitel ayant des fonctionnalités et des caractéristiques sensiblement différentes, mais le plus commun est sans doute le Telic d'Alcatel (qui signifie, au passage, « ALsacienne de Constructions Atomiques, de Télécommunications et d'ÉLectronique »), norme Minitel 1B (avec touches Fnct et Ctrl). Ce petit « B » était important pour moi à l'époque, car il signifiait « bi-standard » et caractérisait le fait de pouvoir passer l'appareil en 80 colonnes (mode VT52) et son modem « retourné » pour émettre en 1200 bps et recevoir en 75 bps et non simplement l'inverse (norme V.23). Mieux encore, il disposait d'une prise DIN 5 à l'arrière permettant de le connecter à un PC pour échanger des données directement (sans modem), et ce à 4800 bps. Tout ceci faisant qu'il était et qu'il est toujours possible de l'utiliser assez simplement comme terminal pour une machine sous GNU/Linux ou BSD.

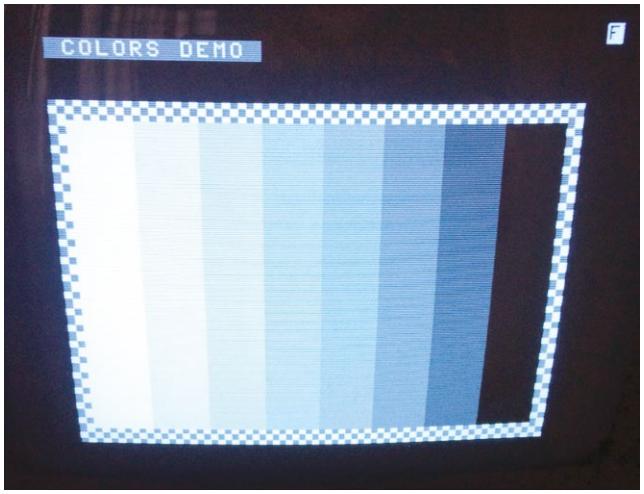
La production de Minitel s'est arrêtée en 2012, tout comme le réseau/service Télétel lui-même, le 30 juin 2012. Quelques 6,5 millions de Minitels ont été produits et utilisés, et bien qu'une campagne de démantèlement et de recyclage des vieux appareils ait été mise en place (assez tardivement semble-t-il), il n'est guère étonnant de retrouver ces matériels, souvent en très bon état et parfaitement fonctionnels, dans les brocantes, les marchés aux puces ou même simplement au fond d'un placard dans un bureau.

En ce qui me concerne, le recyclage n'est à envisager qu'à la condition que le matériel ne puisse plus être utilisé. Dans le cas du Minitel, certes 6,5 millions de terminaux ne peuvent pas tous trouver une seconde vie, mais il me paraît important de garder au moins un exemplaire en bonne condition, et d'en faire quelque chose. Après tout c'est un morceau d'histoire de notre culture qui, bien que maintenant remisé dans les oubliettes lugubres de la technologie, mérite de trôner en bonne place dans mon musée personnel au même titre que mes C64/128, ma NeXTstation Turbo, ma SGI Indy ou mon Acorn RiscPC (et non, ce n'est pas un tas de vieux trucs ou un capharnaüm technologique, c'est un « musée personnel », na !).

## 2. UN MINITEL ? MAIS POUR QUOI FAIRE ?

Le Minitel, et en particulier le 1B, peut servir à énormément de choses aujourd'hui. L'utilisation la plus courante est sans le moindre doute celle en tant que terminal pour un PC GNU/Linux ou une Raspberry Pi. Cette pratique étant relativement courante et mise en œuvre de longue date, elle est très vastement documentée, en français sur le Web, et je trouve qu'il serait déplacé et redondant d'en faire étalage ici.

À l'autre extrême, si je puis dire, nous avons, par exemple, la superbe réalisation de Pierre Genet qui a littéralement désossé son Minitel pour replacer la carte logique par un circuit de sa composition à base de LM1881,




*Le Minitel est capable d'afficher plusieurs niveaux de gris qui correspondent en réalité à 8 couleurs. Ce croquis de démonstration, livré avec la bibliothèque dédiée, nous permet de constater que ceci est parfaitement pris en charge.*

le transformant en moniteur vidéo noir et blanc (**http://www.cfp-radio.com/realisations/rea48/minitel-01.html**). Une fois doté d'une telle entrée composite, doublé d'un circuit audio, l'écran est parfaitement capable d'afficher de la vidéo provenant d'un lecteur multimédia, d'un ordinateur familial des années 80 ou même d'une Raspberry Pi. L'opération est cependant assez complexe, risquée (on joue avec quelque chose pilotant environ 25000 Volts) et implique la réalisation d'un circuit complet. C'est un peu lourd pour un article.

Le Minitel 1B est capable de servir de terminal pour bien des projets puisqu'il suffit de le passer en 80 colonnes et de changer la vitesse de communication via une séquence de touches, pour pouvoir s'en servir pour tout et n'importe quoi capable de communiquer via une liaison série TTL (0-5V). Ceci ne présente donc pas vraiment d'originalité puisqu'au final cela se résume à la connexion de deux fils (plus la masse) et une utilisation en lieu et place du moniteur série Arduino. C'est trop facile !

Il est bien plus intéressant de laisser le Minitel en mode Vidéotex et d'utiliser ses spécificités historiques, et je pense en particulier au mode alphamosaïque, avec dans l'idée de composer des pages mêlant texte et graphisme. La nature des informations à afficher sera laissée à votre entière discrétion, mais on imagine sans peine bien des usages (moniteur de température, horloge, borne interactive, etc.). Après tout, le Minitel n'est rien d'autre qu'un écran piloté via une liaison série, exactement comme on pourrait le faire avec un écran LCD en SPI ou encore avec un écran série Nextion (voir *Hackable n°19*).

Concernant la communication série via la prise péri-informatique DIN 5 située à l'arrière du Minitel, voici ce que disent les « Spécifications Techniques d'Utilisation » du Minitel 1B des PTT (facile à trouver en ligne, en PDF) :

```
1.2 Niveaux électriques
Les niveaux électriques de la prise sont compatibles avec le niveau TTL
collecteur ouvert:
- un niveau de tension supérieur ou égal à 2,5V présenté sur une entrée
(Rx ou PT) sera interprété comme un état logique 1;
- un niveau de tension inférieur ou égal à 0,4V présenté sur une entrée
(Rx ou PT) sera interprété comme un état logique 0.
[...]

2 Caractéristiques des liaisons
Le module prise assure des échanges bidirectionnels simultanés en
asynchrone à une vitesse programmable, dont la valeur en standard est de
1200 bauds.

2.1.1 Signaux Ts et Rx
La réception (signal Rx) et l'émission (signal Tx) des données par le
Minitel s'effectuent par des liaisons du type série asynchrone. Le format
des signaux est fixe avec 8 bits de données plus un bit de parité paire.
```

Ceci signifie deux choses importantes :

- une carte Arduino 5V (ou tolérante aux 5V) pourra parfaitement communiquer directement avec le Minitel, que ce soit via le port série physique ou avec la bibliothèque `SoftwareSerial` ;
- le format des données va nous poser problème, car la configuration par défaut sur Arduino est 8N1 : 8 bits de données, pas de parité et un bit de stop.

Ce second point est cependant gérable sans avoir à jouer avec des configurations avancées des registres du microcontrôleur et/ou avec la modification de la bibliothèque `SoftwareSerial`. La documentation des PTT ne le précise pas, mais il y a également un bit de stop et le format est donc 7E1 (7 bits, parité paire, 1 bit de stop). Autrement dit, l'un des bits de données en 8N1 est le bit de parité en 7E1, nous pouvons de ce fait le considérer manuellement à l'envoi comme à la réception, sans toucher à la configuration.

Et c'est précisément en creusant de ce côté que je suis tombé sur une bibliothèque Arduino écrite par Jérôme Saint-Clair (alias « 01010101 ») du *Graffiti Research Lab France* de Paris. Cette bibliothèque, nommée simplement `Minitel`, permet de faire précisément ce que je souhaitais et bien plus encore, afficher des pages Vidéotex sur l'écran du Minitel. Cette bibliothèque est le résultat d'un projet de son auteur consistant à capturer des images d'une webcam avec Processing pour les envoyer sur le Minitel (**http://graffitiresearchlab.fr**).

La vidéo de démonstration date d'environ 3 ans et le dépôt GitHub de la bibliothèque (**https://github.com/01010101/Minitel**) affiche des changements et mises à jour s'étalant entre mai 2013 et avril 2016. Bien que tout ceci soit relativement récent (version 0.2), il semblerait que le développement ait ralenti, sinon stoppé. En effet, c'est en apportant quelques modifications à ce code, concernant la configuration de la vitesse de communication, que je me suis rendu compte qu'une fonctionnalité similaire avait été proposée en janvier 2017 (via une *pull request* dans le jargon GitHub), sans pour autant trouver un écho, favorable ou non.

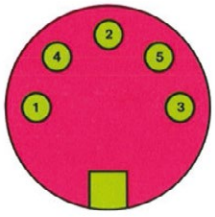
J'ai donc décidé de forker le dépôt GitHub et y ai inclus mon code de configuration de la vitesse, ainsi que quelques autres modifications (détection des caractéristiques du Minitel connecté). Pour la suite de cet article, cependant, vous pourrez donc utiliser aussi bien la bibliothèque proposée sur **https://github.com/01010101/Minitel** que sur **https://github.com/Lefinnois/Minitel**, puisque nous nous en tiendrons aux fonctionnalités de base.

*Page 68 du document « minitel 1B Spécification Techniques d'Utilisation » des PTT se trouve la description de la prise DIN 5 située à l'arrière de l'appareil, derrière une adorable petite trappe coulissante. Les broches utilisées pour la connexion à la carte Arduino UNO sont la masse (2), Rx (1) sur 6 et Tx (3) sur 7.*

**1.1 Prise mécanique**

La prise péri-informatique est du type DIN 5 broches femelle sur laquelle sont disponibles les signaux suivants :

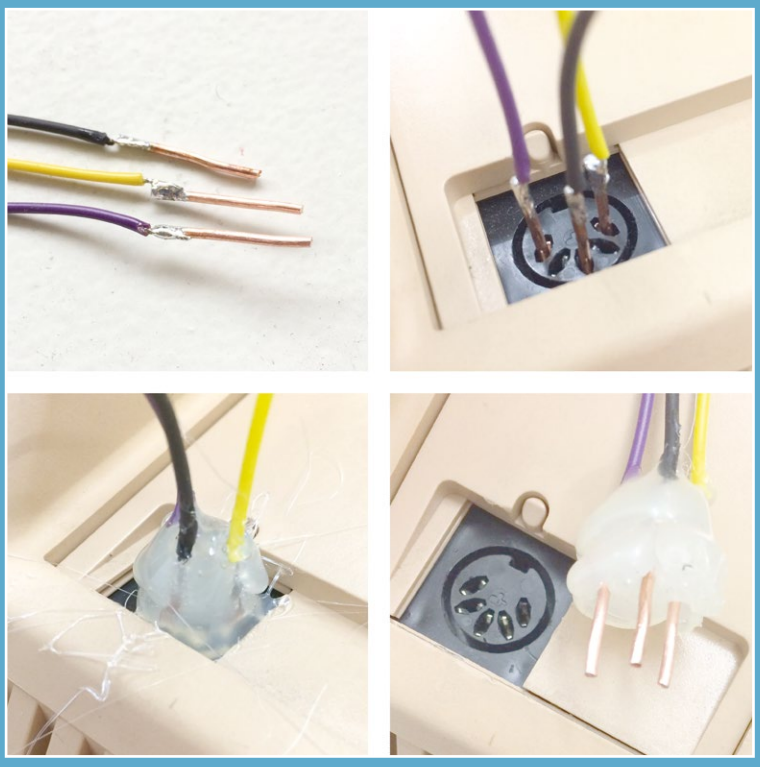
- **broche 1 :** réception des données par le terminal (signal Rx) ;
- **broche 2 :** masse ;
- **broche 3 :** émission de données par le terminal (signal Tx) ;
- **broche 4 :** périphérique en transmission (signal PT) ;
- **broche 5 :** sortie alimentation disponible pour les périphériques. Cette fonction n'est pas disponible sur les versions dont l'identification porte les références Cu2 à Cu4 incluses.




Prise femelle vue de face

La bibliothèque de Jérôme Saint-Clair repose sur `SoftwareSerial` et permet donc d'utiliser n'importe quelles broches de la carte Arduino en guise de ligne Rx et Tx. Par défaut, le Rx du Minitel est connecté à la broche 6 et Tx à la 7, les masses sont bien entendu mises en commun. L'aspect matériel s'arrête là, tout le reste est une question de code.

## PAS DE CONNECTEUR ? JE SORS LE PISTOLET À COLLE !



Voici une technique qui pourrait sans doute vous être utile à l'occasion et dont je me sers tantôt lorsque je ne trouve pas ou n'ai pas sous la main de connecteur adapté pour une prise femelle. C'est celle que j'ai utilisé avec mon Minitel en attendant l'arrivée de mes connecteurs DIN 5 par la poste. À ma connaissance, elle n'a jamais été décrite nulle part et je dois avouer que je ne suis pas peu fier de ma trouvaille :)

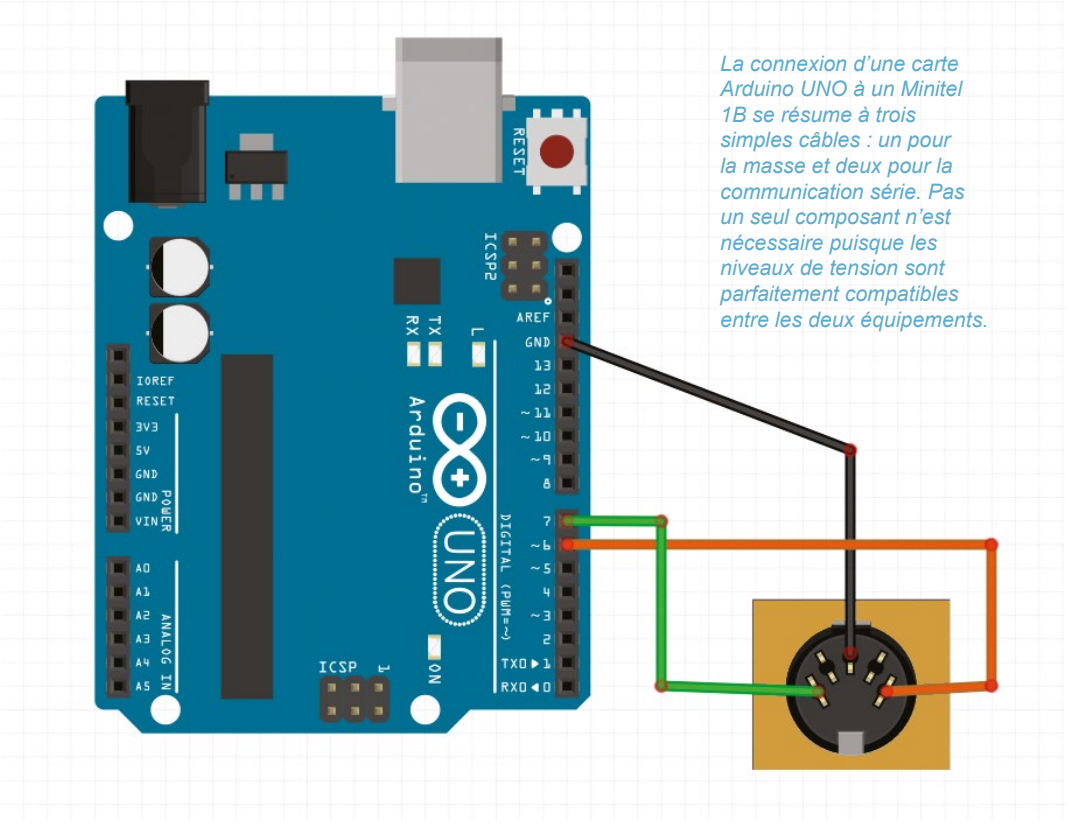
L'idée est la suivante : un connecteur mâle n'est rien d'autre qu'une série de broches maintenues dans un certain agencement par un support généralement en plastique. Pour obtenir un connecteur mâle, il suffit donc d'utiliser la prise femelle comme moule pour y placer les connecteurs et combler les interstices avec quelque chose qui soit temporairement fluide, mais se solidifie en un temps raisonnable : de la colle de pistolet à colle.

La réelle difficulté réside généralement dans la recherche de broches à la bonne taille. Dans le cas de la prise DIN 5 du Minitel, jumpers, trombones et câbles rigides pour platine à essais (voir le numéro précédent) étaient tous d'un diamètre insuffisant. Cela s'est terminé avec un brin de cuivre émaillé provenant d'un transformateur, qu'il a fallu débarrasser de son vernis isolant par grattage.

Une fois ces sections de brins soudées à des câbles/jumpers pour platine à essais, il a suffi de les placer dans les trous correspondants de la prise DIN femelle du Minitel et noyer le tout dans la colle. Il faut faire attention à correctement remplir l'espace entre les broches tout en évitant de saturer les bords, ce qui rendrait ensuite l'extraction difficile. On procède en plusieurs fois de façon à former un connecteur moulé autour des broches sans trop laisser de vide. Je recommande également de faire un petit essai en posant une petite goutte, de façon à s'assurer que la colle froide se retire relativement facilement du support.

Il suffit ensuite d'attendre que tout cela refroidisse pour délicatement, mais tantôt avec insistance, décoller le tout de la prise femelle en s'aidant, par exemple, d'un tournevis plat. On ébavure ensuite au cutter les coulures superficielles et les fils de colle tirés de-ci de-là, et le tour est joué. Vous obtenez un connecteur, certes très laid, mais parfaitement fonctionnel.

Je suppose qu'il est possible d'améliorer la technique en utilisant un « fond » en papier pour éviter que la colle n'adhère à la prise et, éventuellement, faire un petit coffrage en carton ou en plastique pour obtenir une forme bien cylindrique. Cependant, la forme « vieux chewing-gum mâché » ne m'a jamais posé de problème.




*La connexion d'une carte Arduino UNO à un Minitel 1B se résume à trois simples câbles : un pour la masse et deux pour la communication série. Pas un seul composant n'est nécessaire puisque les niveaux de tension sont parfaitement compatibles entre les deux équipements.*

## 3. COMMUNIQUONS UN PEU AVEC LE MINITEL

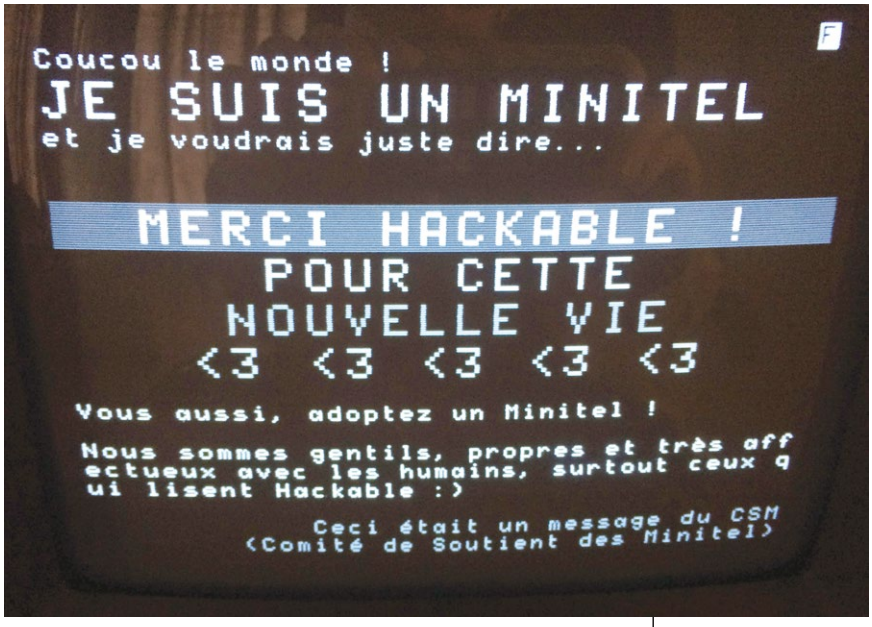
La connexion de la carte Arduino au Minitel est simplissime et une fois le périphérique sous tension, nous n'avons plus qu'à envoyer nos instructions pour contrôler l'affichage. Grâce à la bibliothèque de Jérôme Saint-Clair, nous n'avons pas besoin de nous soucier du format de données ni même de la configuration. Il suffit de télécharger la bibliothèque depuis GitHub et la copier dans le répertoire **libraries** de son carnet de croquis. On a alors accès à un certain nombre d'exemples, dont **MinitelDemo** présentant l'ensemble des fonctionnalités.

Nous verrons la partie graphique un peu plus loin, mais dans un premier temps, penchons-nous tout d'abord sur le texte avec un premier croquis :

```
#include <SoftwareSerial.h>
#include <Minitel.h>

//Minitel m(6,7,4800);
// Objet représentant la connexion avec les valeurs
// par défaut
Minitel m;

void setup() {
  // efface l'écran
  m.clearScreen();

  // mode texte
  m.textMode();
  // couleur avant-plan
  m.textColor(WHITE);
  // couleur arrière-plan
  m.bgColor(BLACK);
```

```
  // texte à la position courante
  m.text("Coucou le monde !");
  // taille élargie en hauteur et largeur
  m.charSize(SIZE_DOUBLE);
  // texte à une position donnée
  m.text("JE SUIS UN MINITEL", 1, 3);
  // taille normale du texte
  m.charSize(SIZE_NORMAL);
  // déplacement du curseur
  m.moveCursorTo(1, 4);
  // affiche le texte à la nouvelle position
  m.text("et je voudrais juste dire...");

  m.charSize(SIZE_DOUBLE);
  m.textColor(WHITE);
  m.bgColor(MAGENTA);
  m.text("  MERCI HACKABLE !  ", 1, 8);
  m.bgColor(BLACK);
  m.textColor(CYAN);
  m.text("     POUR CETTE", 1, 10);
  m.textColor(BLUE);
  m.text("    NOUVELLE VIE", 1, 12);
  m.textColor(GREEN);
  m.text("   <3 <3 <3 <3 <3", 1, 14);

  m.charSize(SIZE_NORMAL);
  m.textColor(WHITE);
  m.text("Vous aussi, adoptez un Minitel !", 1, 16);
  // l'affichage de texte déplace le curseur
  // Si on précise une position
  m.text("Nous sommes gentils, propres et très ", 1, 18);
  // le texte est mis au bout
  m.text("affectueux avec les humains, surtout ceux ");
  // et revient à la ligne automatiquement
  m.text("qui lisent Hackable :)");

  // mais il faut savoir où il s'arrête pour ne pas
  // écraser le texte déjà affiché et écrire ensuite
  // au bon endroit
  m.textColor(BLUE);
  m.text("            Ceci était un message du CSM", 1, 22);
  m.text("        (Comité de Soutien des Minitel)", 1, 23);
}

void loop() {
  m.moveCursorTo(1, 4);
  delay(2000);
}
```

Pour commencer à utiliser le Minitel comme afficheur, nous n'avons qu'à déclarer un objet de type **Minitel**, ici appelé **m**. On utilisera ensuite les méthodes de cet objet pour contrôler l'affichage. L'instanciation de **m** déclenche automatiquement la configuration du port série logiciel avec les broches 6 et 7 par défaut (mais aussi de **Serial** qui n'est pourtant pas nécessaire, j'ai retiré cela dans ma version). Il est cependant possible de préciser d'autres broches, par exemple avec **Minitel m(8,9);**.




*Une fois la connexion réalisée, envoyer du texte sur l'écran du minitel est un jeu d'enfant. Il est possible de choisir la taille du texte (parmi 4 options) ainsi que la couleur d'arrière et d'avant-plan (et même de faire une faute d'orthographe).*

Le Minitel utilise sa mémoire vive comme tampon d'affichage et un curseur peut être déplacé en spécifiant une colonne et une ligne. Par défaut, le curseur ne s'affiche pas, mais il est possible de l'activer avec **`cursor()`** et le désactiver avec **`noCursor()`** (ou **`cursor(true)`** et **`cursor(false)`**). Le texte qui sera alors envoyé à l'écran sera tout simplement placé à la position du curseur. Nous voyons dans ce croquis que la méthode **`text()`** peut prendre en argument le texte à écrire, mais aussi, éventuellement la position sous la forme de deux valeurs numériques. Utiliser **`moveCursorTo()`** ou spécifier les coordonnées en argument de **`text()`** est une affaire de goût, les deux fonctionnent.

Il faut cependant faire attention à une chose importante : lorsque vous changez la taille des caractères (avec **`charSize()`**), la hauteur de ces derniers peut être double (**`SIZE_DOUBLE`** ou **`SIZE_DOUBLE_HEIGHT`**) et cette dimension s'étend vers la ligne précédente. En d'autres termes, si vous placez un texte de taille **`SIZE_NORMAL`** en ligne 5, par exemple, le texte en **`SIZE_DOUBLE`** ou **`SIZE_DOUBLE_HEIGHT`** devra être positionné en 7 pour ne pas écraser la ligne 5.

Bien que l'affichage soit en noir et blanc, il vous est possible de préciser, avant l'envoi d'un texte, une couleur d'avant-plan (**`textColor()`**) et d'arrière-plan (**`bgColor()`**) parmi les valeurs **`BLACK`**, **`RED`**, **`GREEN`**, **`YELLOW`**, **`MAGENTA`**, **`BLUE`**, **`CYAN`** et **`WHITE`**. Cette configuration reste active tant que vous ne rechangez pas les couleurs. Notez cependant que l'envoi d'un tel attribut « consomme » un caractère. C'est pour cette raison qu'il faut placer un espace en début de la chaîne de caractères devant adopter cet attribut. Ceci est malheureusement la conséquence d'un choix technique, car le code envoyé pour changer la couleur, par exemple, occupe la place d'un caractère. Il faut faire avec...

Enfin, contrairement aux exemples livrés avec **`Minitel`**, nous avons ici tout placé dans la fonction **`setup()`** pour n'afficher/envoyer qu'une fois la page à l'écran. En l'absence d'activité, l'écran du Minitel passe en veille afin de ménager le revêtement interne du tube cathodique (image fantôme). Les exemples redessinent entièrement l'écran toutes les deux secondes mais, comme avec notre croquis, on peut simplement déplacer le curseur dans **`loop()`** pour obtenir le même résultat et garder éveillé l'écran sans perdre du temps à dessiner.

## 4. UN PEU D'AIDE DE PYTHON POUR LES IMAGES

Afficher des graphiques en mode alphamosaïque n'est, en principe, pas difficile du tout. Le fonctionnement même du système et la façon dont cela est géré par la bibliothèque **`Minitel`** rendent l'affichage d'un caractère graphique des plus aisé. Il suffit de passer en mode graphique et d'utiliser la méthode adéquate pour envoyer le caractère :

```
m.graphicMode();
m.moveCursorTo(10, 5);
m.graphic("001111");
```


*En commençant à travailler avec le mode alphamosaïque, on se rapproche des pages Vidéotex utilisées à l'époque. Voici donc notre solution pour remplacer Internet, les blogs, la boutique en ligne : le 3615 HACKABLE (sous réserve de validation par le gérant des Éditions Diamond, c'est pas gagné).*

*Parmi les exemples livrés avec la bibliothèque Arduino, on retrouve des pages Vidéotex qui rappelleront sans doute des souvenirs à certains et certaines...*


Les méthodes permettant le changement de couleur d'avant et d'arrière-plan sont les mêmes que celles utilisées pour le texte. La méthode `graphic()` prend en argument une chaîne de caractères représentant, de gauche à droite, la liste des pixels à allumer (1) ou éteindre (0) sur une grille de 2×3, en commençant par la première colonne de la première ligne et s'étendant de gauche à droite et de haut en bas. `"101010"` donnera donc un caractère avec les trois pixels de gauche à la couleur d'avant-plan et les trois de droite à celle d'arrière-plan. `"110011"` allumera ainsi les deux pixels du haut ainsi que les deux du bas. Vous avez compris le principe...

Le problème n'est pas technique, mais pratique. Pour afficher une image composée ainsi de, disons, 30 pixels sur 30 pixels, il faudra donc utiliser `graphic()` 150 fois (30/2 fois 30/3). On pourra cependant accélérer l'affichage et simplifier les choses en utilisant la méthode `repeat()` pour demander au Minitel de répéter le dernier caractère le nombre de fois passé en argument. Afficher une ligne de 30 pixels (et donc de 15 caractères) identiques ainsi peut être fait en spécifiant le caractère avec `graphic()` une fois et en utilisant `repeat(14)` avant de passer à la ligne suivante avec `moveCursorTo()`. `repeat()` fonctionne également avec les caractères du mode texte.

Cette astuce/accélération toutefois ne rend pas la tâche véritablement plus facile. Une autre solution, mais nous privant de l'utilisation de `repeat()` consiste

*La solution la plus « propre » pour utiliser la prise péri-informatique consiste à faire usage d'un connecteur DIN 5 mâle à souder. Mais ceci n'empêche pas de bricoler une solution temporaire en attendant que le matériel arrive...*

à utiliser la méthode **textByte()** permettant de spécifier une donnée à envoyer directement au Minitel. En plaçant la valeur de chaque caractère dans un tableau, on peut créer une boucle envoyant l'image complète, caractère après caractère, ligne après ligne. Le résultat est plus compact dans le croquis puisqu'il suffit de déclarer une variable (un tableau de **char**) et surtout, il devient possible de facilement stocker plusieurs images pour les afficher à souhait au moment et à l'endroit voulu (sans avoir des croquis de centaines de lignes de code).

La question qui se pose naturellement ensuite est celle concernant la manipulation de ces images et plus exactement, comment obtenir cette liste de valeurs à partir d'un format d'image plus simple à modifier. C'est là qu'on fera intervenir Python et le module PIL. Ceci pourrait être réalisé avec n'importe quel langage (Java, Processing, C, Perl), mais il faut avouer que Python est bien pratique pour rapidement coder ou prototyper quelque chose sur un coin de table (ou un coin de Raspberry Pi en l'occurrence).

L'idée est donc de créer une image en noir et blanc avec un logiciel adapté (Gimp, PhotoShop, Paint, etc.) tout en s'assurant de correspondre aux besoins du système alphamosaïque :

- image en noir et blanc,
- largeur maximum de 40 colonnes de 2 pixels,
- hauteur maximum de 24 lignes de 3 pixels,
- le nombre de pixels en largeur doit être pair,
- le nombre de pixels en hauteur doit être multiple de 3.

Python permettrait de manipuler une image d'une taille et d'un format quelconque de manière à la convertir selon ces critères, tout comme il est possible d'utiliser, sur une Pi, des outils comme ceux d'ImageMagick pour procéder à la conversion avant de faire intervenir notre script Python. L'idée étant de montrer qu'il est relativement aisé de programmer rapidement un script pour accomplir cette tâche, je suis parti du principe qu'en créant une image d'une dimension maximum de 80×72 pixels, qui a tout d'une icône donc, on devra de toute façon ajuster un certain nombre de pixels à la main. N'espérez pas pouvoir convertir une photo couleur de votre smartphone et obtenir un résultat satisfaisant en laissant simplement faire des outils ou des algorithmes.

Voici donc le script Python :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import sys
import os
from PIL import Image

# test argument
if len(sys.argv) != 2:
    sys.exit("Veuillez préciser un nom de fichier (PNG, JPEG, etc)")
```

```
# essai ouverture image
try:
    im = Image.open(sys.argv[1])
    pix = im.load()
except:
    sys.exit("Impossible d'ouvrir le fichier image")

# largeur et hauteur en pixels
width, height = im.size

# nom de la variable dans le croquis
nom = os.path.splitext(sys.argv[1])[0]
# vérifications
if width/2 > 40:
    sys.exit("Erreur : L'image doit avoir 80 pixels de large maximum")
if height/3 > 24:
    sys.exit("Erreur : L'image doit avoir 72 pixels de haut maximum")
if width % 2:
    sys.exit("Erreur : L'image doit avoir une largeur multiple de 2")
if height % 3:
    sys.exit("Erreur : L'image doit avoir une hauteur multiple de 3")

print "// Image: " + nom + "(" + str(width) + "x" + str(height) + ")"

x = 0
y = 0

print "#define L_" + nom + "    " + str(width/2)
print "#define H_" + nom + "    " + str(height/3)

print "\nPROGMEM const byte " + nom + "[]={"
while y < height:
    print "  ",
    x = 0
    while x < width:
        val = 32
        val +=  1 if pix[x,y] > 0 else 0;
        val +=  2 if pix[x+1,y] > 0 else 0;
        val +=  4 if pix[x,y+1] > 0 else 0;
        val +=  8 if pix[x+1,y+1] > 0 else 0;
        val += 16 if pix[x,y+2] > 0 else 0;
        val += 32 if pix[x+1,y+2] > 0 else 0;
        if y > (height-4) and x > (width-3):
            print val,
        else:
            print str(val) + ",",
        x += 2;
    print ""
    y += 3;
print "};"
```

Dans les grandes lignes, le fonctionnement est relativement simple puisque, après les vérifications d'usage, on parcourt simplement les pixels toutes les deux colonnes et toutes les trois lignes. De là, on inspecte les 5 pixels adjacents pour en déduire un caractère de 2×6 et encoder cela en une valeur. Le résultat est directement affiché à l'écran en ligne de commandes sous la forme d'une déclaration d'un tableau en C (le nom de la variable est tiré du nom de fichier de l'image) :

```
$ ./img2gcharTAB.py logo.png
// Image: logo(64x60)
#define L_logo    32
#define H_logo    20

PROGMEM const byte logo[]={
   32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, ...
   32, 32, 32, 32, 32, 32, 32, 32, 64, 80, 92, 94, ...
   32, 32, 32, 32, 32, 32, 88, 94, 95, 95, 47, 47, ...
   32, 32, 32, 32, 88, 95, 95, 95, 95, 95, 95, 93, ...
   32, 32, 64, 94, 95, 63, 39, 35, 35, 35, 75, 47, ...
   32, 72, 95, 63, 33, 32, 64, 80, 60, 68, 90, 32, ...
   64, 95, 63, 33, 32, 56, 83, 32, 80, 46, 35, 43, ...
   74, 95, 53, 32, 74, 33, 95, 39, 33, 32, 32, 32, ...
   74, 95, 61, 44, 95, 44, 95, 32, 32, 32, 32, 32, ...
   74, 95, 53, 32, 74, 48, 95, 68, 80, 48, 32, 32, ...
   32, 95, 95, 48, 32, 43, 80, 32, 47, 45, 80, 56, ...
   32, 34, 95, 95, 84, 32, 32, 35, 35, 35, 75, 32, ...
   32, 32, 34, 79, 95, 93, 92, 80, 80, 80, 90, 92, ...
   32, 32, 32, 32, 34, 95, 95, 95, 95, 95, 95, 95, ...
   32, 32, 32, 32, 90, 95, 55, 33, 32, 66, 79, 95, ...
   32, 32, 32, 32, 95, 95, 74, 52, 92, 72, 55, 95, ...
   32, 32, 32, 32, 79, 95, 86, 64, 92, 48, 90, 95, ...
   32, 32, 32, 32, 34, 79, 95, 95, 95, 95, 95, 95, ...
   32, 32, 32, 32, 32, 32, 35, 43, 47, 95, 95, 55, ...
   32, 32, 32, 32, 32, 32, 32, 32, 72, 95, 63, 32, ...
};
```

Il suffira ensuite de copier/coller cela (ou rediriger la sortir avec **>**) dans un **.h** qu'on intègrera à notre croquis Arduino dans un nouvel onglet et on pourra alors se pencher sur la fonction d'affichage :

```
void afficheImg(const byte *data, int l, int h, int posx, int posy) {
  for (int y=0; y<h; y++) {
    m.moveCursorTo(posx, posy+y);
    for (int x=0; x<l; x++) {
      m.textByte(pgm_read_byte_near(data+(y*l)+x));
    }
  }
}
```

Afin d'économiser au maximum la mémoire vivre de la carte, notez le mot-clé **PROGMEM** forçant le stockage de ces données en mémoire flash, ainsi que l'utilisation de **pgm_read_byte_near()** pour les lire. La fonction **afficheImg()** prend en argument un pointeur sur le début des données, la largeur en colonnes, la hauteur en lignes, ainsi que la position sur l'axe horizontal et vertical. Il ne s'agit bien entendu pas de pixels, mais de caractères alpha-mosaïques. On pourra ensuite dans **setup()** ou **loop()** utiliser tout simplement :

```
m.graphicMode();
m.textColor(CYAN);
m.bgColor(RED);
afficheImg(logo, L_logo, H_logo, 5, 3);
```

Ceci affichera l'image stockée dans **logo[]** à partir de la colonne 5 de la ligne 3. Il ne vous reste donc plus qu'à vous amuser à créer et convertir des images ainsi, mélanger cela avec un peu de texte pour composer et obtenir de belles pages Vidéotex pour vos projets.


*On peut réaliser toutes sortes de pages en peu de temps à condition d'utiliser les bons outils et la bonne approche. Un script Python permettant de créer une image alphamosaïque à partir d'un fichier graphique prend, certes, du temps à être développé, mais accélère grandement la suite des opérations.*

## POUR FINIR

Cet article déjà fort long ouvre la voie à bon nombre d'améliorations. Nous n'avons, en effet, pas parlé de l'utilisation du clavier, mais les exemples livrés avec la bibliothèque Minitel devraient rendre cela limpide. Le script Python est aussi une piste à poursuivre, en incluant éventuellement un prétraitement des images, mais aussi la conversion de 8 niveaux de gris en couleurs du Minitel pour améliorer le rendu. Notez cependant que le mode alphamosaïque travaille avec des caractères de 2×6 et bien qu'il soit possible de changer de couleur d'un caractère (2×3 pixels) à l'autre, un résultat parfait ne pourra être obtenu.

Enfin, la bibliothèque aussi peut être améliorée en incluant des fonctionnalités supplémentaires. Il est par exemple possible de faire en sorte que la vitesse de communication sur la prise péri-informatique soit augmentée (jusqu'à 4800 bps pour un Minitel 1B, soit 4 fois plus vite que la normale), et ce sans avoir à utiliser une séquence de touches au clavier. Ceci est intégré à ma version de la bibliothèque, il suffit de spécifier la vitesse souhaitée en troisième argument lors de la déclaration de l'objet de type `Minitel` (`Minitel m(6,7,4800);`).

Autre fonction également ajoutée, la détection ou plus exactement l'interrogation de la ROM du Minitel qui annonce alors son type, son constructeur et diverses caractéristiques. Ceci peut être obtenu en invoquant la méthode `getMinitelInfo()` comme le montre un nouvel exemple ajouté, appelé `MinitelInfo`. D'autres améliorations sont certainement encore possibles.

Enfin, il y a tout l'aspect matériel qu'il est possible d'explorer. On peut envisager d'intégrer un module Bluetooth dans l'appareil et donc ne plus avoir à utiliser de liaison série filaire. On peut également s'écarter des cartes Arduino et utiliser un ESP8266 pour apporter le Wifi à ce vénérable équipement. Et, bien entendu, sous réserve de faire attention au niveau de tension, ou de tout simplement utiliser un convertisseur USB/série, rien ne vous empêche d'utiliser ou d'intégrer une Pi. Frédéric BISSON, alias Zigazou, a développé un module PyMinitel pour Python (**https://github.com/Zigazou/PyMinitel**) qui offre plus ou moins les mêmes fonctionnalités que la bibliothèque pour Arduino.

Et si tant est qu'il soit nécessaire de le préciser, si vous trouvez un Minitel, gardez-le ! La campagne de recyclage a provoqué un effet inattendu : ces matériels commencent à se faire rares et donc cher. Un Minitel 1 ou 1B Telic peut toujours être trouvé pour environ 20€, mais un Minitel 2 vous coûtera déjà un peu moins de 60€ (cher payé pour atteindre 9600 bps). Les prix montent ensuite rapidement avec des modèles moins courants comme les Minitel NFZ 300 ou les Minitel 2 de Philips. Le Minitel est maintenant une pièce de collection, traitez le vôtre avec respect et il vous le rendra au centuple... **DB**

# ACTUELLEMENT DISPONIBLE

## LINUX PRATIQUE N°104


# INITIEZ-VOUS AUX BITCOINS & CRYPTO-MONNAIES !

## NE LE MANQUEZ PAS

**CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :**

**https://www.ed-diamond.com**

# FABRIQUEZ VOTRE T-SHIRT INTERACTIF AVEC UN LILYPAD ARDUINO

*Axelle Apvrille*

**Vous avez sans doute déjà vu des montres connectées, des bracelets sportifs connectés, mais le T-shirt connecté, là, quelle classe pour un(e) informaticien(ne) ! Il y a moyen d'allier vos penchants de « hacker » à vos talents artistiques. Mais cela peut aussi être l'occasion de quelques frustrations : j'ai failli jeter le T-shirt par la fenêtre (ou à la poubelle) au moins 3 fois. Alors si vous envisagez de vous lancer (dans le projet, pas par la fenêtre), lisez cet article !**

Lorsque Sparkfun **[1]** a sorti une gamme de composants électroniques portables sur soi (« wearable »), cela m'a tout de suite intéressée. L'occasion de faire quelque chose de joli, de geek et d'original ! Cette gamme, baptisée Lilypad, est faite de composants électroniques que l'on peut coudre sur un vêtement avec du fil conducteur, et que l'on peut également passer (délicatement quand même) au lave-linge. Il existe de nombreux composants : des LEDs, boutons, capteurs et même des Arduino.

Je me suis alors mise en tête de faire un T-shirt « intelligent », qui utilise divers composants, sans tout de même viser trop compliqué pour une première réalisation. Mon objectif est d'avoir un T-shirt qui relève la température ambiante et l'affiche de façon illustrative sur un thermomètre gradué à l'aide de plusieurs LEDs. Ainsi, rien qu'en regardant le T-shirt, on peut savoir qu'il fait par exemple entre 15 et 20°C.

## 1. ÉLÉMENTS

C'est l'heure des courses (cf. tableau 1) ! Il nous faut un capteur de température, des LEDs pour le retour visuel et un microcontrôleur pour piloter l'affichage des LEDs en fonction de la température.

Pour relier les composants entre eux, c'est la spécificité de la couture connectée, nous utilisons du *fil conducteur*. Comme tout fil, le fil conducteur est distribué en bobine, mais il est hélas plus cher, de couleur métallisée uniquement, mais bien sûr c'est un excellent conducteur !

*Figure 1 : Vue rapprochée du capteur de température Lilypad et de deux boutons Lilypad. Les composants sont cousus avec du fil conducteur.*

Les composants de la gamme Lilypad, eux, sont soudés sur des mini circuits sur lesquels les entrées/sorties et alimentation sont reliées à des trous bordés de métal conducteur. L'idée est alors de coudre autour de ces trous afin d'une part de fixer le mini circuit sur le vêtement, mais aussi de connecter électriquement les circuits. Les fils doivent aussi être protégés entre eux. Pour cela, j'ai utilisé des bandes thermocollantes.

Pour alimenter l'ensemble des circuits, il faut aussi bien entendu une alimentation. J'aborderai ce point plus tard dans l'article, car j'ai rencontré de nombreux déboires.

| MATÉRIEL | PRIX INDICATIF |
|---|---|
| Lilypad Arduino USB **[2]** ATmega32U4 | 27.00 € |
| Capteur de température Lilypad MCP9700 | 5.40 € |
| Fil conducteur en bobine de 9 mètres (en fait, il faut bien ça) | 3.25 € |
| 5 LEDs Lilypad | 5.40 € les 5 |
| 2 boutons poussoirs Lilypad (voir sections 3 et 4) | 2 x 1.70 € |
| Batterie e-Textile, ex. : **https://www.sparkfun.com/products/13112** (voir section 9) | 5.00 € |
| Bande thermocollante (voir section 10) | 3.00 € |
| T-shirt | 5.00 € |
| **Total** | **57.45 €** |

*Tableau 1 : Matériel indispensable à la réalisation du T-shirt connecté.*

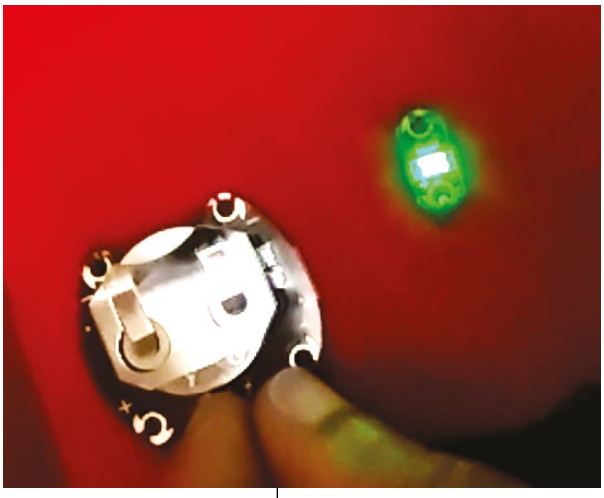


*Figure 2 : Première étape : la LED est cousue sur un tissu d'essai. La pile bouton est insérée dans un porte-pile bouton Lilypad que j'avais.*

Enfin, pour tester avant de coudre, il est pratique d'avoir des pinces crocodiles - à moins que vous ne préfériez utiliser une platine à essai avec des fils scotchés aux composants. Et bien sûr, il vous faut un T-shirt.

Faut-il être un as de la couture pour faire ce T-shirt connecté ? Non ! D'ailleurs, je suis une très mauvaise couturière. Il faut juste savoir faire le *point avant* **[3]**, et arrêter des fils. C'est suffisant pour un T-shirt de base. Vos compétences éventuelles supplémentaires vous permettront sans doute des bonus artistiques !

| AUTRES OUTILS UTILES | IMPORTANCE |
|---|---|
| Aiguille à relativement gros chas | Indispensable |
| Ciseaux | Indispensable |
| Dé à coudre | Optionnel |
| Découd-vite | Quasi indispensable |
| Épingles | Quasi indispensable |
| Feutrine, tissus, colle à tissus, éléments décoratifs | Dépend du projet exact réalisé |
| Fer à souder | Dépend du projet exact réalisé |
| Machine à coudre | Dépend du projet exact réalisé |
| Pinces crocodiles | Optionnel, mais pratique |
| Voltmètre | Indispensable |

*Tableau 2 : Autre matériel ou outils utiles.*

## 2. ÉTAPES DE MONTAGE

À moins d'être très expérimenté en couture connectée, coudre tous les composants pour tester ensuite est tout sauf une bonne idée, même pour un projet qui ne paraît pas compliqué. Au contraire, il vaut mieux procéder par petites étapes incrémentales et tout vérifier petit à petit :

1. Vérification de l'**allumage d'une LED** (voir Figure 2). Sur un bout de tissu sans importance, j'ai cousu une LED que j'ai alimentée par une pile bouton. Cela a permis de m'essayer au fil conducteur.
2. **Lecture de la température** en connectant le capteur de température au Lilypad Arduino USB. J'ai fait cette étape sans aucune couture, en reliant les composants avec des pinces crocodiles. On relie le + du capteur au + de l'Arduino, le – au – et le S (signal) du capteur à, par exemple, A5 sur l'Arduino. Enfin, j'alimente l'Arduino par le port USB. Je détaillerai un peu plus loin exactement comment lire la température.
3. **Dessin du circuit** complet, afin de savoir où coudre sur le T-shirt (voir Figure 3). J'ai effectué mon croquis à l'aide de Fritzing **[4]**.
4. **Couture** des éléments sur le T-shirt.

5. **Test et résolution des problèmes** – toujours présents sur un tel système : nous verrons par la suite ceux que j'ai rencontrés, et comment les résoudre.

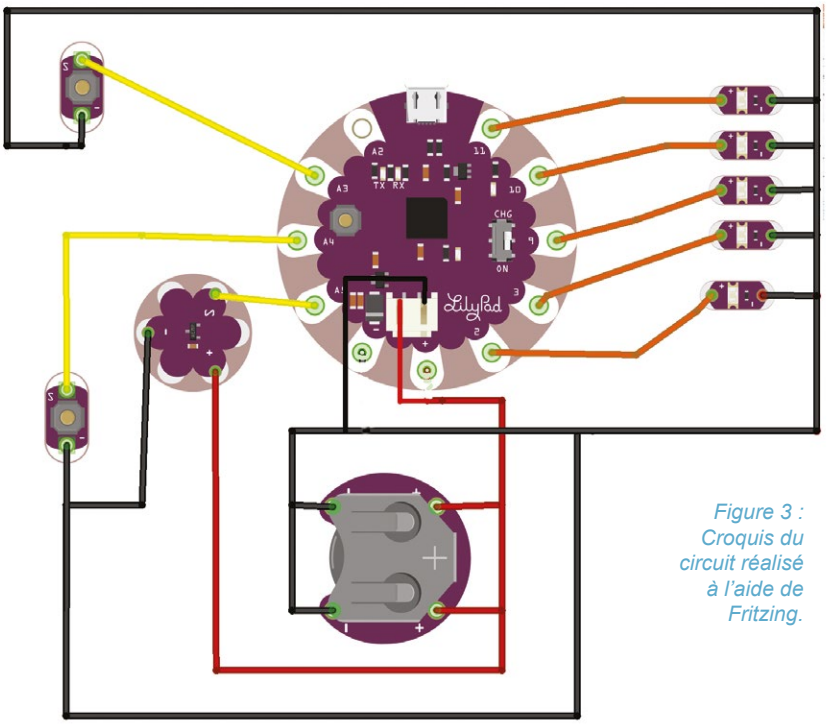


*Figure 3 : Croquis du circuit réalisé à l'aide de Fritzing.*

## 3. LIRE LA TEMPÉRATURE

La documentation du capteur de température **[5]** nous explique que le voltage du signal S du capteur est fonction de la température. Précisément, le capteur est censé fournir 0.5V à 0°C, 0.75V à 25°C etc., c'est-à-dire 10mV par degré Celcius.

Le signal S du capteur est relié, dans mon cas, à la broche analogique A5 du Lilypad Arduino USB. Pour lire la tension, on utilise la fonction `analogRead()`. Celle-ci convertit l'intervalle 0 - 3.3V (le Lilypad Arduino USB fonctionne en 3.3V **[2]**) en un intervalle 0-1023. Nous avons donc 3.3 / 1024 = 0.003223 volts par unité.

Dans un croquis Arduino, la lecture de la température se fait donc ainsi :

```
int sensorPin = A5;
sensorValue = analogRead(sensorPin);
degreesC = ((sensorValue * 0.003223) - 0.5) * 100 ;
```

En essayant le circuit avec les pinces crocodiles, on se rend vite compte que la théorie est quelque peu éloignée de la pratique. Il y a à la fois un problème de *fluctuation des données* (la température lue varie d'une seconde à l'autre) et un problème de *précision* du capteur. Un capteur de température de meilleure qualité serait bien entendu une solution, mais la gamme Lilypad n'en propose qu'un seul, donc à moins de prendre un capteur de température en dehors de cette gamme et de le coudre sur le tissu, il n'y a pas d'alternative de ce côté. J'opte plutôt de tirer parti du capteur que j'ai :

1. Lisser les variations de température en effectuant 10 relevés de température et en prenant la moyenne comme résultat.
2. *Calibrer* le capteur en jouant sur le coefficient 0.003223 pour ajuster la température. Par exemple, si la température lue par le capteur est un peu basse par rapport à la réalité, je peux reprogrammer le Lilypad Arduino avec un coefficient légèrement plus élevé pour lire une température plus proche de la réalité.

Oui, mais reprogrammer le Lilypad Arduino USB nécessite un ordinateur à portée de main. On a beau être des geeks, on ne se promène pas avec un ordinateur attaché à son T-shirt ! Comment faire ? J'ai choisi d'ajouter deux boutons à mon circuit : un bouton signifiera « augmenter le

coefficient » (et donc la température) et l'autre « diminuer le coefficient ». Ces deux boutons me permettront d'ajuster la température lue sur le T-shirt en fonction d'une température de référence (station météo par exemple).

## 4. BOUTONS

Les boutons poussoirs Lilypad ont une borne « - », que je connecte à la masse, et une borne signal (S) que je relie à une broche analogique du Lilypad Arduino USB (ex : A3).

Pour savoir si un bouton est enfoncé ou pas, on utilise la fonction **digitalRead()** :

```
int buttonPlus = A3;
int buttonValue = digitalRead(buttonPlus);
if (buttonValue == LOW) {
  Serial.print("Bouton + appuyé");
} else {
  Serial.println("Bouton + pas appuyé");
}
```

Suis-je trop binaire ? Je pensais que cela suffisait. Mais, sans rien d'autre, non, cela ne marche pas tout à fait. Lorsqu'on appuie sur le bouton, tout va bien, cela fonctionne comme prévu : le signal est relié à la masse, et on obtient donc un signal bas (**LOW**). Le problème survient lorsque le bouton n'est pas appuyé. Dans ce cas-là, le signal n'est relié à rien, et renvoie donc une valeur assez aléatoire qui peut tout aussi bien être bas (**LOW**) que haut (**HIGH**)...

La solution est d'ajouter au circuit une grosse résistance, pour que le signal lu soit haut (**HIGH**) lorsque le bouton n'est pas enfoncé. N'ayez pas peur, pas besoin d'ajouter une résistance à votre T-shirt, le Lilypad Arduino USB contient de telles résistances, pour les activer il suffit de préciser le mode **INPUT_PULLUP** :

```
void setup()
{
  pinMode(buttonPin, INPUT_PULLUP);
  ...
}
```

## 5. ALLUMER LES LEDS

Allumer ou éteindre une Lilypad LED ne présente aucune difficulté, donc je ne m'y attarderai pas. Ces LEDs sont soudées sur de petites plaques avec une borne « - » et une borne « + ». Dans mon cas, j'ai connecté la borne « + » de mes 5 LEDs respectivement aux sorties digitales 2, 3, 9, 10, et 11, et la borne « - » à une masse commune (voir Figure 3).

Dans un croquis , on allume une LED avec **digitalWrite(led, HIGH)** et on l'éteint avec **digitalWrite(led, LOW)**.

Pour mon T-shirt, j'allume progressivement les LEDs en fonction de la température. Entre 10 et 15°C, je n'allume que la première LED, entre 15 et 20°C j'allume les deux premières et ainsi de suite. À plus de 30°C, j'allume donc toutes les LEDs (j'habite dans le Sud...). L'allumage progressif des LEDs permet de simuler le mercure d'un thermomètre, avec une granularité moindre, mais un plus bel effet ;-)

Le croquis final utilisé pour mon T-shirt connecté peut être téléchargé depuis **[10]**.

## 6. TRACER LES COUTURES

Si vous avez tout connecté avec des pinces crocodiles (ou à l'aide d'une platine d'essais), vous avez probablement un beau tas de fils. Pas question de faire pareil sur le T-shirt !

Premièrement, il faut absolument **limiter la quantité de fil conducteur** à utiliser, que ce soit pour le côté esthétique, le côté financier ou la conductivité. Le fil conducteur a une faible résistance, certes, mais à force cela finit par bloquer le courant.

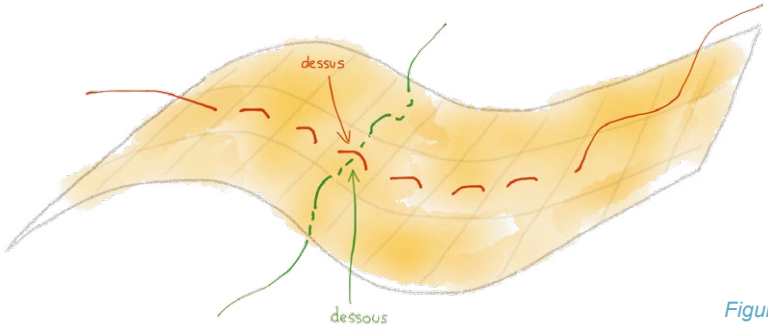



*Figure 4 : Si on ne peut éviter un croisement de fils, il faut se servir du tissu comme isolant et passer un fil au-dessus, l'autre au-dessous.*

Deuxièmement, il faut forcément **éviter les croisements** avec du fil conducteur à moins de créer des courts circuits !

Pour ce point, j'ai tenté d'utiliser Fritzing. Outre tracer des circuits, le logiciel permet également de trouver le meilleur chemin pour relier les composants entre eux. C'est la fonction d'auto-routage. Je ne suis pas totalement arrivée à une solution satisfaisante, donc j'ai terminé à la main. Je suis arrivée à un point où le chemin des fils était assez simple, mais il était difficile d'éviter un croisement, à moins d'ajouter beaucoup plus de fil.

Comment gérer un croisement ? C'est une question importante, mais impossible de trouver une explication dans les tutoriaux ! Si le tissu du T-shirt est suffisamment épais, il suffit de faire attention au moment de la couture de faire passer un fil au-dessus du T-shirt et un fil en dessous. Le tissu du T-shirt sert d'isolant entre les deux. Si le tissu de votre T-shirt n'est pas assez épais (un peu percé ?!), alors il vous faudra ajouter un petit morceau de feutrine à l'endroit de la connexion.

## 7. COUTURE !

Ça y est ! Il est temps de passer à la couture. Sélectionnez une aiguille suffisamment grosse pour pouvoir passer le fil conducteur dans le chas, mais la plus fine possible pour moins marquer votre tissu. Vous remarquerez tout de suite que le fil conducteur est un peu plus difficile à manier que le fil « normal », car il est plus épais et moins souple. Si vous envisagez de l'utiliser à la machine à coudre, je vous conseille vivement de faire un essai, car son manque de souplesse, son épaisseur et sa faible élasticité peuvent poser problème. Personnellement, j'ai fait toutes les coutures à la main : c'est plus long, mais je maîtrise mieux ce que je fais au niveau des connexions.

J'ai agrémenté le circuit de quelques éléments de décoration (un pingouin sur la banquise), mais ils n'ont aucune fonction électronique. Vous pouvez laisser libre

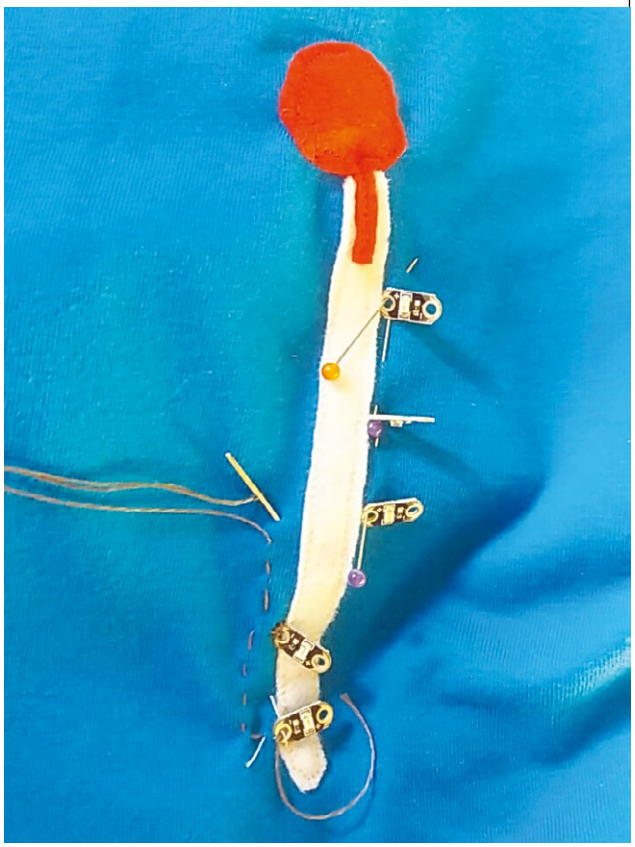


*Figure 5 : Couture des LEDs. J'ai d'abord cousu le motif d'arrière-plan, puis j'ai épinglé les LEDs que j'ai cousues progressivement à la main.*

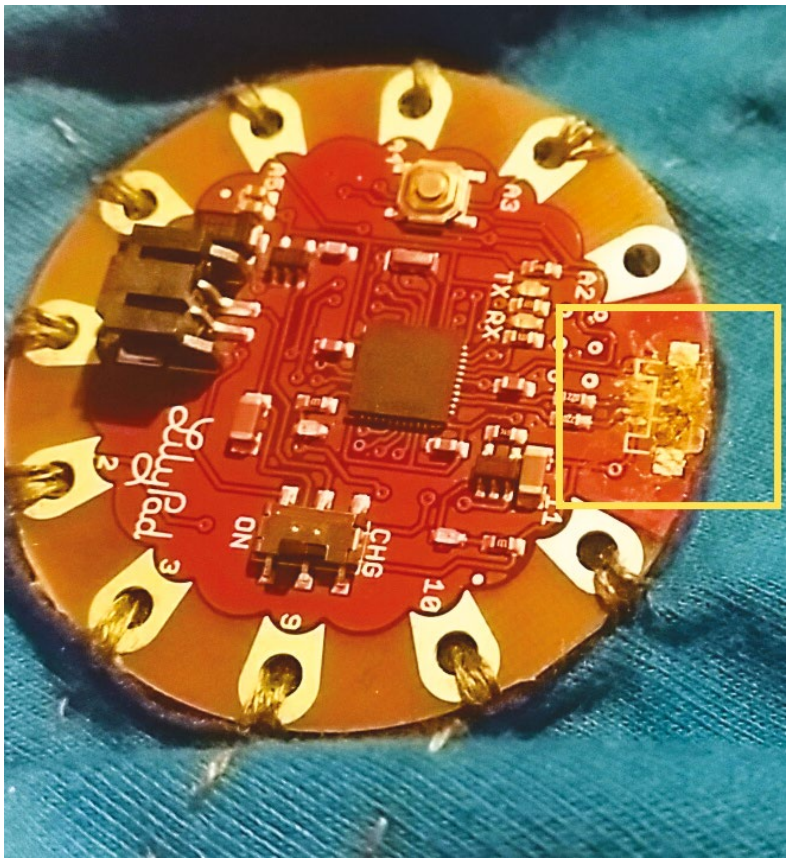



*Figure 6 : Le connecteur USB de mon Lilypad Arduino USB a lâché !*

*Figure 7 : Découd-vite en action. Malgré son nom, cela reste toujours lent !*

cours à vos talents artistiques. Niveau confort, je conseille de coudre de la feutrine (ou tout autre tissu de votre choix) sous les plus gros éléments électroniques, notamment le Lilypad Arduino USB et l'alimentation. C'est plus agréable à porter, car cela évite d'avoir l'impression d'avoir un circuit imprimé sur la peau.

## 8. QUELQUES DIFFICULTÉS INATTENDUES AVEC LE LILYPAD

Depuis le début, mon Lilypad Arduino USB avait un connecteur USB un peu étroit et je bataillais chaque fois que je devais le connecter. Devinez ? Au bout d'un moment, il a cassé ! Mon mari m'a aidée et nous avons essayé de ressouder le connecteur. Cela a marché quelque temps, puis cela s'est abîmé à nouveau et m'a valu de belles crises de nerfs, car suivant la position du câble USB le transfert de données se faisait plus ou moins bien. Puis, un peu plus tard, cela a complètement lâché, et cette fois-ci nous ne sommes pas arrivés à réparer la bête.

J'ai dû racheter un Lilypad Arduino USB, découdre l'ancien armée d'un découd-vite et recoudre le nouveau. C'est beaucoup de temps perdu, et puis attention, lorsqu'on remplace l'Arduino, il faut faire refaire les contacts, c'est-à-dire *rattraper l'ancien fil conducteur*, faire un nœud autour – pour assurer une bonne connexion - et coudre jusqu'au nouvel Arduino.

**Moralité de l'histoire** : ne cousez jamais un élément électronique vaguement défectueux sur un vêtement connecté. Aussi, essayez de limiter les tests en USB qui ont tendance à tirer sur les fils... ou sur le port USB.

## 9. ALIMENTATION

Le choix de l'alimentation dépend des besoins. Malheureusement, aucune documentation ne précise comment alimenter le Lilypad Arduino USB de façon efficace, et en fonction des composants connectés. Comme j'avais vu quelques exemples **[6]** avec porte-pile bouton Lilypad et LEDs Lilypad, j'ai tenté avec un porte-pile bouton. Et puis après tout, on pourrait estimer que les éléments de la gamme Lilypad sont faits pour fonctionner entre eux, non ? J'étais quand même perplexe qu'une pile bouton (CR2032) suffise à alimenter les LEDs *et* l'Arduino. Je vois d'ici quelques électroniciens avertis hausser les épaules : effectivement, je n'aurais même pas dû tenter, c'était idiot, cela n'avait aucune chance de marcher (et ça ne marchait pas).

Conservez le porte-pile bouton pour éclairer quelques LEDs, mais n'envisagez pas de l'utiliser pour un Lilypad Arduino.

Oui, mais quoi utiliser alors ? Je décide d'opter pour un porte-pile AAA. J'en trouve un en France à 16 euros, mais je trouve le prix vraiment exagéré pour un porte-pile. J'en achète un à 2 euros sur un site marchand chinois. Les spécifications du porte-pile **[7]** précisent qu'on peut insérer une pile de tension entre 1.2 et 5V et que cela fournira 5V à l'Arduino, grâce à un convertisseur boost NCP1402.

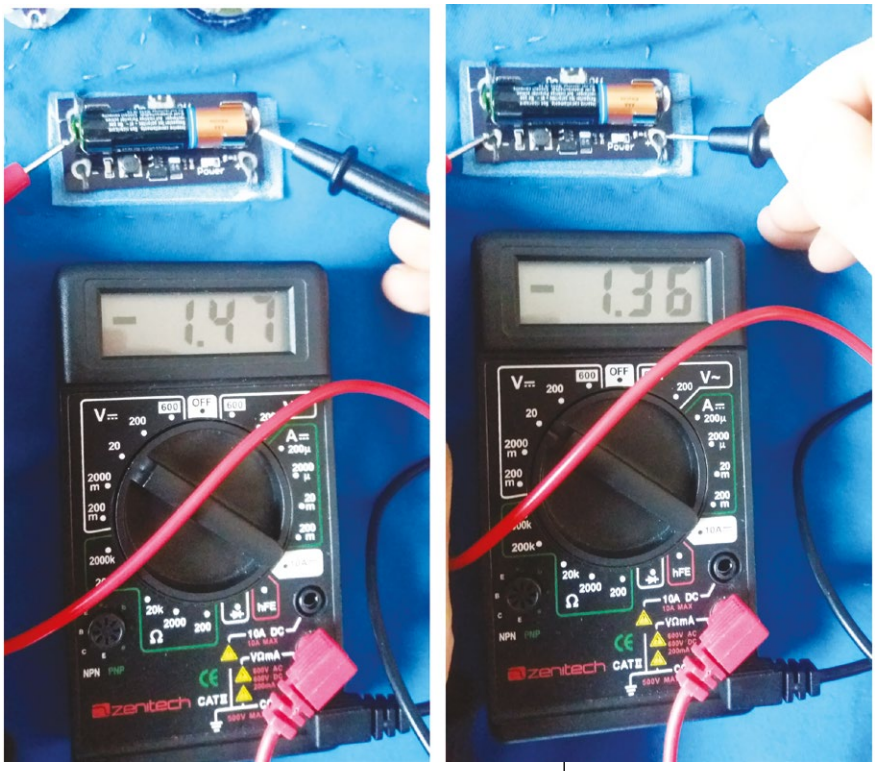



*Figure 8 : Sur l'image de gauche, on voit que ma pile AAA fournit 1.47V. Sur l'image de droite, on voit que le porte-pile ne fournit en sortie que 1.36V et pas les 5V attendus... Le convertisseur boost est défectueux (ou absent...).*

Assez bêtement, je remplace le porte-pile bouton par le porte-pile AAA sans le tester. Évidemment, ça ne marche pas. Je teste le circuit : le composant que j'ai reçu fournit 1.5V, pas 5V : pas de quoi alimenter le Lilypad Arduino USB (voir Figure 8). Sans doute est-il défectueux ? Me voici de retour à la case départ.

**Moralité** : testez les composants électroniques lorsque vous les recevez, surtout d'un site lointain (smiley).

Comment alimenter le Lilypad Arduino USB ?! D'autres tutoriels Sparkfun **[8]** montrent des Lilypad Arduino alimentés par des batteries LiPo, et c'est la méthode à retenir. Par chance, nous disposions chez nous d'une vieille batterie inutilisée d'hélicoptère radio-commandé, et j'ai souhaité la réutiliser. Le connecteur n'était pas le bon (il y en a tellement de différents...) alors nous avons bricolé une connexion : j'ai soudé deux fils mâle-mâle respectivement au connecteur JST + et – du Lilypad Arduino USB, et laissé l'autre extrémité libre.

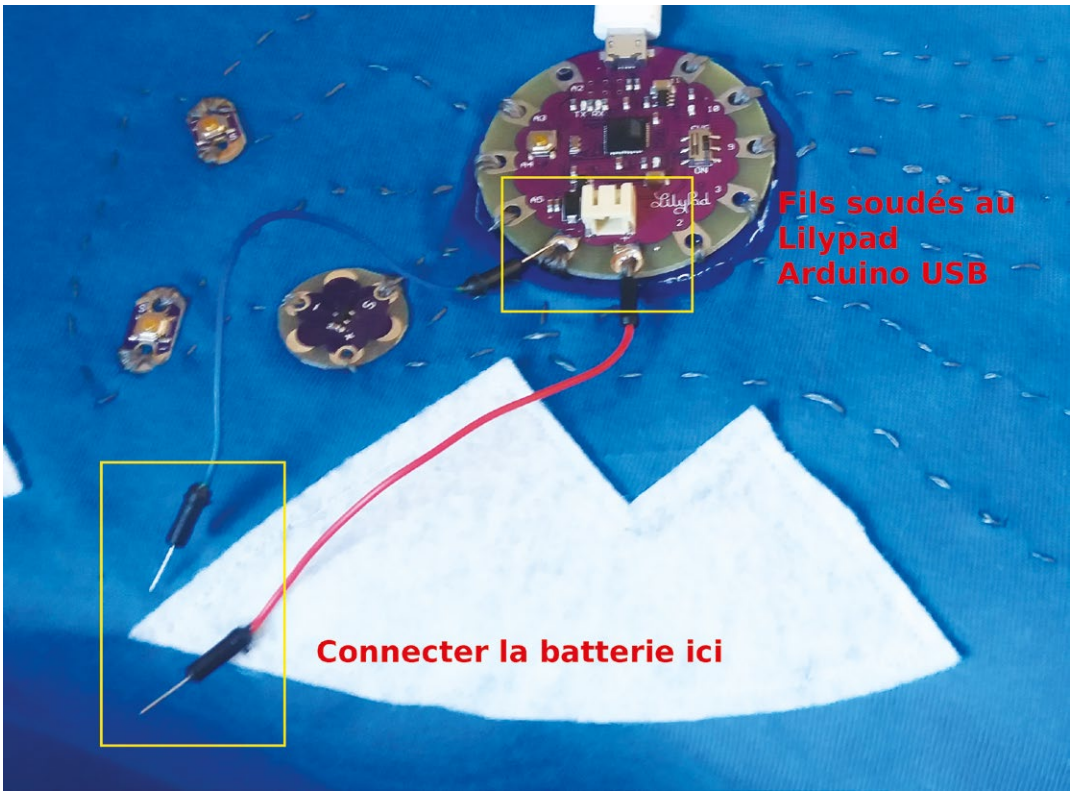



*Figure 9 : Oups ! Je me suis trompée encore une fois ! J'ai soudé sur le + et le – du Lilypad Arduino USB. Mais ce n'est pas bon ! En regardant le schéma du composant, on voit que le + n'est pas une entrée, mais une sortie. Il faut connecter la batterie sur le + et le - du connecteur JST qui est juste au-dessus.*

Lorsque je souhaite alimenter l'Arduino, j'insère le fil + dans le connecteur rouge de ma batterie, et le fil – dans le connecteur noir de ma batterie, et hop, ça démarre ! Inversement, je débranche les fils pour déconnecter la batterie. Si vous préférez, vous pouvez ajouter un interrupteur.

Enfin, un avantage d'une batterie LiPo : le rapport poids/puissance bien plus favorable que des piles AAA classiques. Ainsi, le T-shirt se plie beaucoup moins pour supporter le poids de la batterie.

## 10. EST-CE QUE JE JETTE LE T-SHIRT PAR LA FENÊTRE ?

Ce n'est pas terminé. J'ai encore rencontré au moins trois problèmes notables :

- **Le coup du mauvais câble USB**. Avec ce câble, l'Arduino s'allumait, mais le transfert du programme ne fonctionnait pas. J'aurais passé moins de temps à me rendre compte que c'était un problème matériel si rien n'avait fonctionné du tout. Si vous avez des messages comme ceux-ci dans la console du logiciel Arduino, pensez tout simplement à changer de câble...

```
avrdude: butterfly_recv(): programmer is not responding
avrdude: butterfly_recv(): programmer is not responding
Found programmer: Id = "ï¿½"; type = ,
    Software Version = .; Hardware Version = #.
avrdude: butterfly_recv(): programmer is not responding
```

- **Les erreurs de couture**. Cela paraît simple, puis plus ça avance, et plus il y a de fils, et plus les erreurs arrivent. Par exemple, la photo ci-contre montre la couture du capteur de température. Est-ce que vous remarquez le problème dans la Figure 10 ? La solution est expliquée dans la légende de l'image. C'est un cas typique d'une erreur de couture. Malheureusement, il est presque certain que vous en ferez. La seule solution, c'est de **vérifier au voltmètre où le courant passe**, méticuleusement, jusqu'à isoler le problème.
- **Les courts circuits**. Un T-shirt n'est pas une planche rigide : lorsque vous le portez, le tissu bouge, se plie. Que se passe-t-il si en se pliant deux portions de fil conducteur se touchent ? Vous pouvez très bien accidentellement avoir une connexion entre le – et le + et donc un beau court circuit ! C'est bête, personne n'en parle, mais il faut y penser ! Il faut isoler chaque fil avec les autres. La solution la plus simple, c'est d'acheter des bandes thermocollantes (ce qu'on utilise par exemple pour les ourlets rapides) et de passer cela au dos du T-shirt au fer à repasser (voir Figure 11). Les fils sont alors « tenus » à leur place, et si le tissu du T-shirt se plie, il n'y aura pas contact entre fil conducteur, mais entre deux zones de bande thermocollante, ce qui ne pose aucun problème. De plus, à porter,


*Figure 10 : Le « - » du capteur de température est relié au « - » du bouton de gauche (ça c'est bon). Mais ensuite, le « - » du bouton n'est relié à rien ! Il faut le relier à la masse. Ce qui est simple sur un breadboard devient beaucoup plus compliqué avec du fil cousu !*

*Figure 11 : Bandes thermocollantes, posées au fer à repasser au dos du T-shirt afin d'éviter des courts circuits accidentels lorsque le T-shirt se plisse et différents fils conducteurs se touchent. J'ai mis plusieurs bandes pour mieux isoler certains fils.*

*Figure 12 : Le T-shirt et son heureux propriétaire !*

certains trouvent plus agréable d'avoir contre soi le tissu de la bande thermocollante plutôt que le fil conducteur (un peu rêche).

## 11. RÉSULTAT FINAL ET CONCLUSION

Vous pouvez contempler le résultat final avec son heureux propriétaire ;-)

Vous avez tous les éléments en main pour réaliser votre propre vêtement connecté. Si vous habitez en région parisienne, vous pouvez également contacter l'ElectroLab **[9]** : certains membres font de la belle couture connectée et ont même proposé des activités à la Nuit du Hack fin juin.

### À RETENIR

- **Tester tous les composants un à un avant de les coudre. Ne pas risquer de coudre un composant « légèrement défectueux ».**
- **Si, à la couture, le fil conducteur doit se croiser, faire passer un fil au-dessus du tissu et l'autre en dessous, voire insérer un morceau de tissu supplémentaire (feutrine par exemple) entre les deux.**
- **Mettre aussi peu de fil conducteur que possible. Cela évite des erreurs de couture, et aussi le problème de résistance naturelle de tout fil, même s'il est moindre pour un fil conducteur.**
- **Isoler les connexions de fil conducteur les uns par rapport aux autres avec une bande thermocollante.**
- **Coudre le fil conducteur fermement, serré, autour des trous de connexions de composants – et non pas lâche – pour que le contact se fasse bien.**
- **Alimenter le Lilypad Arduino USB avec une batterie LiPo.**
- **Le capteur de température Lilypad n'est pas précis et doit être calibré.**

*Figure 13 : Vue rapprochée de la réalisation finale. Le pingouin se trouve sur la banquise et cache l'Arduino. On peut néanmoins y accéder en soulevant ses pieds. La baleine supporte la batterie, et la température est affichée à droite par les LEDs. Le capteur de température et les boutons poussoirs pour le calibrage ne sont pas cachés et se trouvent à gauche.*

Quelques améliorations pour mes prochaines réalisations :

- Les LEDs se voient très bien de nuit, ou par temps gris, mais pas bien en plein soleil. C'est dommage pour un T-shirt qu'on porte a priori quand il fait beau... Il faudrait trouver des LEDs plus puissantes.
- Une des LEDs s'éclaire plus que les autres. Je n'ai pas encore compris pourquoi. La tension délivrée par l'Arduino est plus élevée pour cette LED, alors que la tension devrait être la même pour toutes les LEDs. Un défaut du Lilypad ?

Enfin, il existe depuis une autre gamme de composants portables sur soi, appelée Flora et commercialisée par Adafruit. Le principe d'utilisation devrait être sensiblement le même. À tester lors d'une prochaine réalisation ? **AA**

## LIENS


**[1] https://www.sparkfun.com/categories/135**

**[2] https://www.arduino.cc/en/Main/ArduinoBoardLilyPadUSB**

**[3] http://www.bluemarguerite.com/Loisirs-creatifs/techniques-3871-point-avant.deco**

**[4]** Fritzing : **http://fritzing.org/home/**

**[5]** TMP35, TMP36, TMP37 : **http://cdn.sparkfun.com/datasheets/Sensors/Temp/TMP35_36_37.pdf**

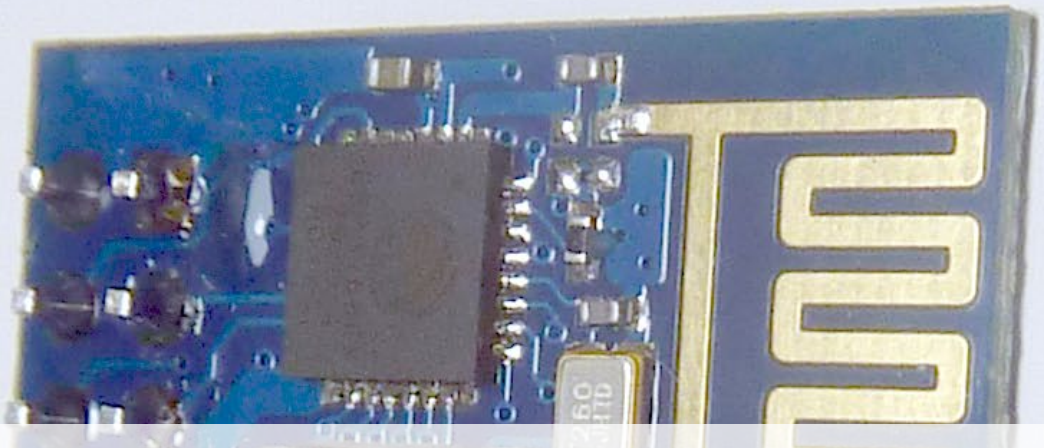
**[6] https://learn.sparkfun.com/tutorials/ldk-experiment-3-buttons-and-switches**

**[7] https://cdn.sparkfun.com/datasheets/E-Textiles/Lilypad/Lilypad_PowerSupply-v18.pdf**

**[8] https://learn.sparkfun.com/tutorials/lilypad-pixel-board-hookup-guide**

**[9] http://www.electrolab.fr/**

**[10]** Code source du croquis pour le Lilypad Arduino USB : **https://git.framasoft.org/axellec/hackable**



# OTA : PROGRAMMEZ VOTRE ESP8266 EN WIFI !

*Denis Bodor*

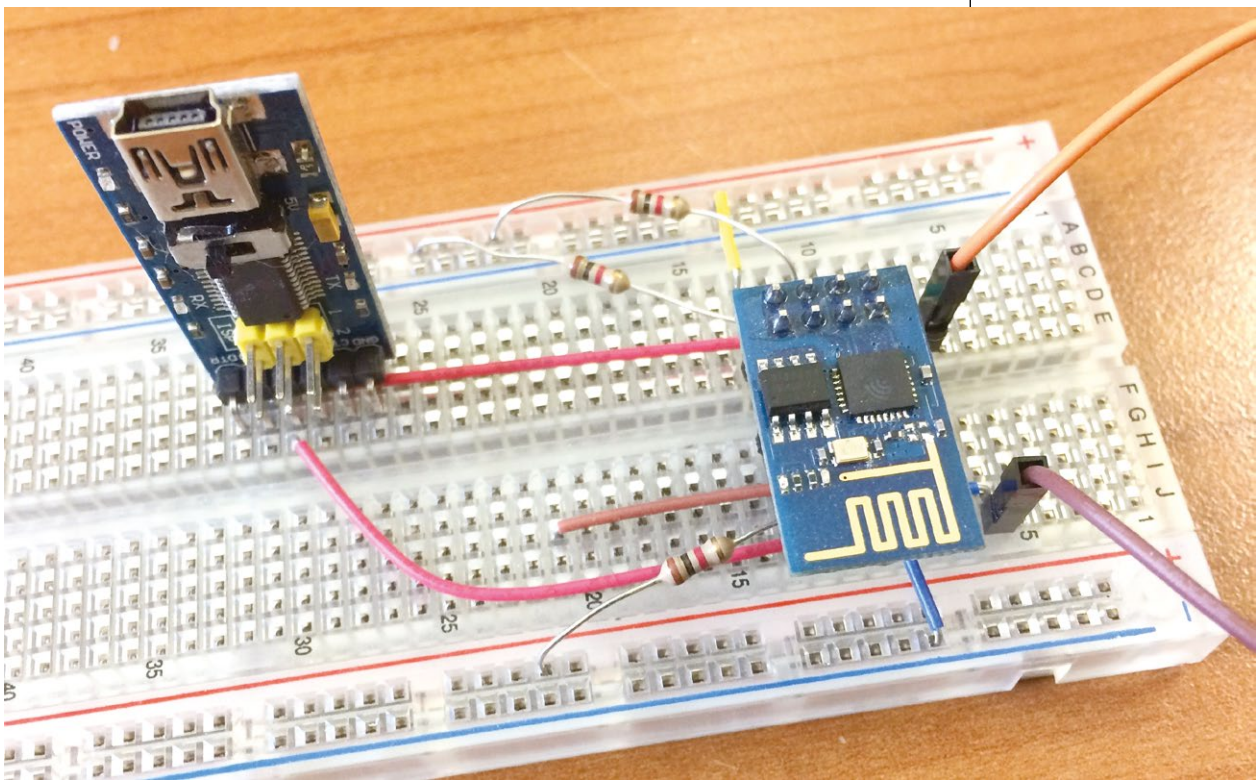
**L'ESP8266, en particulier depuis l'arrivée d'une collection massive et diversifiée de cartes et modules en faisant usage, est une plateforme compatible Arduino absolument fantastique. Plus on fouille pour apprendre à s'en servir du mieux possible, plus on trouve de fonctionnalités totalement fascinantes qui deviennent vite indispensables. Celle dont il sera question ici n'est autre que la possibilité de charger votre croquis dans la mémoire, non plus avec un câble, mais à distance, en Wifi !**

Le fait de pouvoir ainsi reprogrammer une carte via une connexion sans fil est appelé une mise à jour OTA, pour *Over The Air* (« par les airs »). Ce terme est directement hérité de l'univers des smartphones où, bien entendu, une mise à jour du système se fait par un téléchargement et une installation par le smartphone lui-même qui, en un sens, se met donc à jour tout seul comme un grand. Notez qu'il n'en a pas toujours été ainsi et qu'il est toujours possible, avec certains smartphones, de mettre à jour le firmware (le système) via une connexion USB, un démarrage dans un mode spécial (*recovery* ou *download*), et un logiciel dédié (Odin ou Heimdall pour les Samsung, par exemple).

Si vous connaissez la plateforme ESP8266, vous savez sans doute que les cartes utilisant cette puce de chez *Espressif Systems*, s'utilisent exactement comme des cartes Arduino. On les branche simplement à son ordinateur et, après ajout des éléments nécessaires dans l'environnement via le gestionnaire de cartes, on transfère le croquis compilé dans sa mémoire. Par rapport aux toutes premières itérations de ces cartes, initialement vendues comme des modules Wifi pour Arduino, ceci représente déjà un grand pas en avant puisqu'il était nécessaire, en ce temps, d'assembler un circuit à la main et de démarrer l'ESP8266 dans un mode particulier en mettant un connecteur à la masse tout en provoquant la réinitialisation.

Cette époque est de loin révolue, même si cela n'était le standard d'il y a seulement une paire d'années. Les cartes ont évolué,

*Avant l'arrivée de cartes comme la Wemos D1 et d'autres modèles complets, programmer l'ESP8266 d'un module ESP-01 se résumait à ceci : le module monté sur une platine, un adaptateur USB/série, des résistances de rappel, des câbles, des jongleries pour changer de mode... C'était il y a environ deux ans, mais cela parait être une tout autre époque tant les choses ont changé depuis.*



se sont diversifiées et se sont étoffées en fonctionnalités comme en ressources. Une utilisation plus simple, plus de broches utilisables, davantage de mémoire flash, une intégration d'un port USB pour la programmation et, surtout, une amélioration drastique du support dans l'environnement Arduino, ont porté l'ESP8266 au rang de super-Arduino économique et connecté.

Cependant, même si ces facilités d'utilisation sont une véritable bénédiction, il est possible d'aller encore plus loin. Lorsqu'on travaille sur un projet impliquant du Wifi, on cherche généralement à avoir une certaine autonomie vis-à-vis du PC/Mac et donc de rendre le projet accessible plus universellement qu'avec une connexion USB, ou du moins, plus connecté qu'avec « un fil à la patte ». En phase de mise au point du croquis, il est généralement assez pénible de devoir laisser connecté le montage ou le reconnecter à chaque mise à jour. Même une fois en place, un projet peut évoluer et si cela implique une flottille d'ESP8266 utilisés comme capteurs de toutes sortes, ce mode de fonctionnement est synonyme de désinstallation, connexion au PC, mise à jour, déconnexion et réinstallation, autant de fois qu'il y a de capteurs.

Une autre approche possible, mais apparemment trop peu connue (une fois qu'on en fait l'expérience on ne souhaite plus aimer que cela), consiste à utiliser l'OTA se résumant, sommairement, à contacter l'ESP8266 en Wifi, lui envoyer le nouveau croquis compilé et lui demander de mettre à jour le contenu de sa mémoire avant de redémarrer. Ceci est bien plus pratique, plus rapide et surtout plus souple qu'une liaison USB classique.

## 1. PRÉPARATIFS ET CHOSES À SAVOIR

Tout le nécessaire pour réaliser un téléchargement (ou « téléversement ») de croquis en OTA est directement intégré dans l'environnement Arduino dès lors qu'on installe le support pour les plateformes/cartes ESP8266. Pour rappel, ceci passe par un petit tour dans les préférences afin d'ajouter une URL pour le gestionnaire de cartes supplémentaires (**http://arduino.esp8266.com/stable/package_esp8266com_index.json**) et par l'installation du support « *esp8266 by ESP8266 community* » via le gestionnaire de cartes (menu ***Outils***). La dernière version stable en date pour ce support est la 2.3.0.

Pour que la mise à jour OTA fonctionne cependant, il faudra que votre système (Windows, macOS ou GNU/Linux) dispose du langage Python en version 2.7. D'après la documentation (version 2.0.0), Python 3.5 ne conviendra pas. Les utilisateurs Windows devront s'assurer que, lors de l'installation du langage sur leur machine,

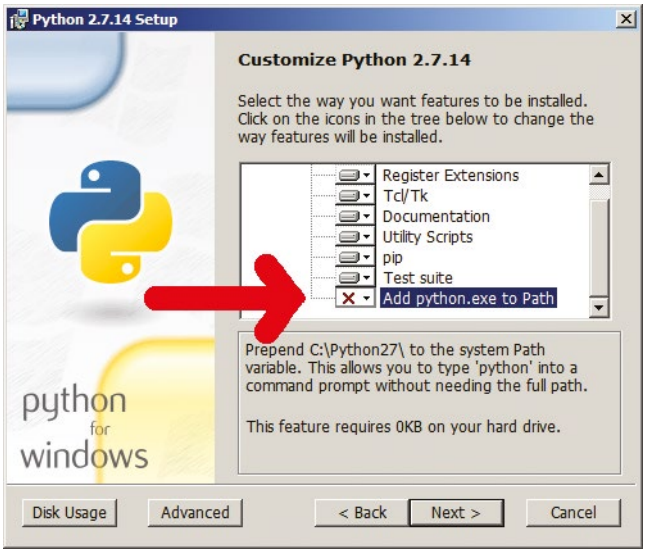

*Supporter les plateformes ESP8266 dans son environnement Arduino se résume à une édition des préférences puis à l'installation du support via le gestionnaire de cartes.*

l'exécutable `python.exe` soit configuré dans les chemins de recherche. Il s'agit d'une simple option d'installation/configuration (***Add python.exe to Path***), mais il faut tout de même penser à l'activer pour que tout puisse fonctionner. Pour installer Python dans votre Windows, allez simplement sur **https://www.python.org/download** et téléchargez la dernière version 2.7 (2.7.14) en fichier d'installation MSI. Double-cliquez sur le fichier et suivez les instructions.

Le téléchargement OTA vers l'ESP8266 se fait via l'exécution du script `packages/esp8266/hardware/esp8266/2.3.0/tools/espota.py`, mais ceci se fait de manière tout à fait transparente, directement depuis l'environnement Arduino. Il est cependant toujours bon de savoir qui fait quoi sur sa propre machine...

Le mode de fonctionnement de la programmation OTA d'un ESP8266 implique principalement deux choses importantes :

- D'une part l'ESP8266 doit être connecté au réseau local, mais la détection de la carte par l'environnement (IDE) Arduino se fera via mDNS-SD (voir article sur le sujet dans le présent numéro). Il ne vous sera pas nécessaire de connaître l'adresse IP de la carte, mais il est important, à mon avis, de comprendre le fonctionnement de mDNS pour régler d'éventuels problèmes. La ou les cartes sont trouvées parce qu'elles exposent un service non standard, « Arduino », sur TCP (`avahi-browse -t _arduino._tcp`).




*Pour que la mise à jour OTA fonctionne correctement sous Windows depuis l'environnement Arduino, vous devrez installer Python 2.7 et ne pas oublier d'ajouter le programme dans le chemin de recherche par défaut, car le script qui envoie le croquis à l'ESP8266 via le réseau est écrit en Python et l'interpréteur doit être trouvé automatiquement.*

- Toutes les cartes ne fonctionnent pas, et je pense en particulier à l'ESP-01, car il faut suffisamment de mémoire flash pour à la fois le croquis en train de fonctionner et le croquis à programmer, reçu via le réseau. 512 Ko de flash (4 Mb) ne sont pas suffisants et bien que le croquis initial fonctionnera sans problème, une tentative de mise à jour provoquera une erreur.

Enfin, même s'il peut sembler superflu de le dire, la mise à jour OTA n'est possible que sur une carte faisant fonctionner un croquis prenant en charge la fonctionnalité. Ce n'est pas une option ou une fonctionnalité de la plateforme, mais quelque chose que vous devez intégrer dans vos croquis et prendre en compte. De ce fait, une première programmation « filaire » en USB sera donc nécessaire, mais il faut également garder à l'esprit qu'en programmant en OTA un croquis qui ne contient pas les routines OTA adéquates, toute programmation OTA deviendra impossible par la suite.

## 2. UNE BASE DE CROQUIS SUPPORTANT L'OTA

Vous l'avez compris, l'OTA n'est pas le problème de la plateforme, mais celle de votre croquis. Pour approcher le sujet, le plus simple est donc de construire un croquis de base sur lequel reposeront tous vos futurs projets. Celui-ci fera le strict minimum : configurer la gestion de l'OTA et prendre en charge d'éventuelles demandes de mise à jour, tout en permettant un fonctionnement presque classique du croquis.

Le croquis débute de manière très classique par l'inclusion des fichiers d'entête (« bibliothèques » au sens Arduino) et la déclaration de quelques variables globales :

```
#include <ESP8266WiFi.h>
#include <ArduinoOTA.h>

// SSID du point d'accès
const char* ssid = "ssid_AP_wifi";
// mot de passe wifi
const char* password = "mot2passeWIFI";
// nom d'hôte (pour mDNS)
const char* hostString = "espOTAtest2";
// mot de passe pour l'OTA
const char* otapass = "123456";

// gestion du temps pour loop()
unsigned long previousMillis = 0;
// gestion du temps pour calcul de la durée de la MaJ
unsigned long otamillis;
```

On retrouve les informations standards indispensables comme le SSID du point d'accès et le mot de passe associé, mais également un nom d'hôte qui sera utilisé par l'environnement Arduino pour procéder à la mise à jour OTA ainsi qu'un mot de passe qu'il sera nécessaire de saisir pour provoquer cette mise à jour. Notez que ce dernier point est capital en terme de sécurité, si vous ne voulez pas que n'importe qui ayant un accès au réseau local puisse reprogrammer votre ou vos ESP8266.

Les deux dernières variables globales déclarées seront utilisées d'une part pour afficher un message en boucle sur le moniteur série afin de montrer que l'attente d'une mise à jour n'est pas bloquante en utilisant `millis()` et pour afficher, en fin de mise à jour, la durée totale de l'opération. Ceci est purement cosmétique, mais donne un côté très « pro » et « travaillé » au projet.

On pourrait envisager une intégration de l'OTA dans notre croquis de façon plus « diffuse », mais je préfère personnellement toujours bien séparer les fonctionnalités, ne serait-ce que pour les transposer plus facilement d'un croquis à un autre. De ce fait, j'ai décidé de réunir l'ensemble du code en rapport avec la configuration de l'OTA dans une fonction, qu'il me suffira d'appeler au moment opportun depuis `setup()`.

Cette fonction, appelée `confOTA()`, se charge tout d'abord de configurer quelques éléments simples à partir des variables précédemment déclarées et initialisées, en utilisant simplement les méthodes de la classe `ArduinoOTA` :

```
void confOTA() {
  // Port 8266 (défaut)
  ArduinoOTA.setPort(8266);

  // Hostname défaut : esp8266-[ChipID]
  ArduinoOTA.setHostname(hostString);

  // mot de passe pour OTA
  ArduinoOTA.setPassword(otapass);
```

Notez au passage que le port utilisé ici a sa valeur par défaut (la même que si cette ligne n'était pas présente). C'est juste une façon de se donner l'opportunité, plus tard, de changer facilement d'avis. Vient ensuite une partie un peu plus difficile à appréhender si vous n'avez pas l'habitude de ce genre de syntaxe en C++ : un adaptateur générique de fonction. Derrière ce nom inquiétant se cache, tout simplement, une solution permettant de passer toute une fonction en argument d'une méthode. Exemple, avec notre première utilisation :

```
// lancé au début de la MaJ
ArduinoOTA.onStart([]() {
  Serial.println("/!\\ MaJ OTA");
  otamillis=millis();
});
```

La syntaxe peut paraître confuse, mais le code se trouvant dans la portée (entre **{** et **}**) est tout simplement celui appelé automatiquement en début de mise à jour OTA. Un fonctionnement similaire à celui utilisé pour les routines d'interruption (avec **attachInterrupt()**) aurait pu être utilisé puisque le code de deux lignes que nous utilisons ici se résume à une fonction et qu'il existe donc un pointeur utilisable vers celle-ci. Cette syntaxe cependant, reposant sur la notion de *template* en C++ présente un avantage intéressant dans le fait de réunir l'ensemble du code en un seul endroit, sans jongler avec des noms de fonction, et ce avec les arguments attendus. Enfin, comme c'est la solution prévue par les développeurs du support ESP8266 pour Arduino, il serait malvenu de chercher à faire autrement juste pour le principe, tout en ajoutant de la complexité.

La déclaration suivante est assez similaire puisqu'il s'agit du code appelé en fin de mise à jour, utilisant **otamillis** pour calculer la durée écoulée :

```
// lancé en fin MaJ
ArduinoOTA.onEnd([]() {
  Serial.print("\n/!\\ MaJ terminee en ");
  Serial.print((millis()-otamillis)/1000.0);
  Serial.println(" secondes");
});
```

Mais les choses deviennent plus touffues avec le code notifiant de la progression de la mise à jour :

```
// lancé lors de la progression de la MaJ
ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
  Serial.printf("Progression: %u%%\r", (progress / (total / 100)));
});
```

Ici, le code utilise deux variables (**progress** et **total**) qui lui sont passées en argument et qui sont ensuite utilisées pour calculer et afficher un pourcentage de progression. Ici, nous allons afficher ce pourcentage de façon répétée (à chaque appel à cette fonction dite de *callback*) en retournant simplement en début de ligne (**\r** équivalent à un CR ou *Carriage Return*) et en écrivant par dessus celle existant précédemment. Bien entendu, à vous de voir si cet affichage vous convient, avec **progress** et **total** il est parfaitement possible de créer une barre de progression de la même manière. Ceci peut être un exercice intéressant pour se faire la main...

Il nous reste une fonction à écrire et celle-ci est chargée de gérer d'éventuelles erreurs. Elle prend en argument un code d'erreur correspondant à un code défini dans **ArduinoOTA.h** qu'il nous suffit de tester pour afficher un message en conséquence :

```
// En cas d'erreur
ArduinoOTA.onError([](ota_error_t error) {
  Serial.printf("Erreur[%u]: ", error);
  switch(error) {
    // erreur d'authentification, mauvais mot de passe OTA
    case OTA_AUTH_ERROR:      Serial.println("OTA_AUTH_ERROR");
                              break;
    // erreur lors du démarrage de la MaJ (flash insuffisante)
    case OTA_BEGIN_ERROR:     Serial.println("OTA_BEGIN_ERROR");
                              break;
    // impossible de se connecter à l'IDE Arduino
    case OTA_CONNECT_ERROR:   Serial.println("OTA_CONNECT_ERROR");
                              break;
    // erreur de réception des données
    case OTA_RECEIVE_ERROR:   Serial.println("OTA_RECEIVE_ERROR");
                              break;
    // erreur lors de la confirmation de MaJ
    case OTA_END_ERROR:       Serial.println("OTA_END_ERROR");
                              break;
    // erreur inconnue
    default:                  Serial.println("Erreur inconnue");
  }
});
```

Le code de démonstration proposé par le support ESP8266 Arduino faisait usage de **if/else if**, mais je préfère cette syntaxe, reposant sur **switch()/case**, bien plus lisible.

Enfin, nous touchons à la fin de notre fonction **confOTA()** et ne devons surtout pas oublier d'initialiser la fonctionnalité OTA qui prendra alors en compte tous nos paramètres :

```
  // Activation fonctionnalité OTA
  ArduinoOTA.begin();
}
```

Nous pouvons nous pencher maintenant sur la fonction **setup()** chargée d'établir la connexion Wifi et bien entendu d'appeler notre fonction de configuration :

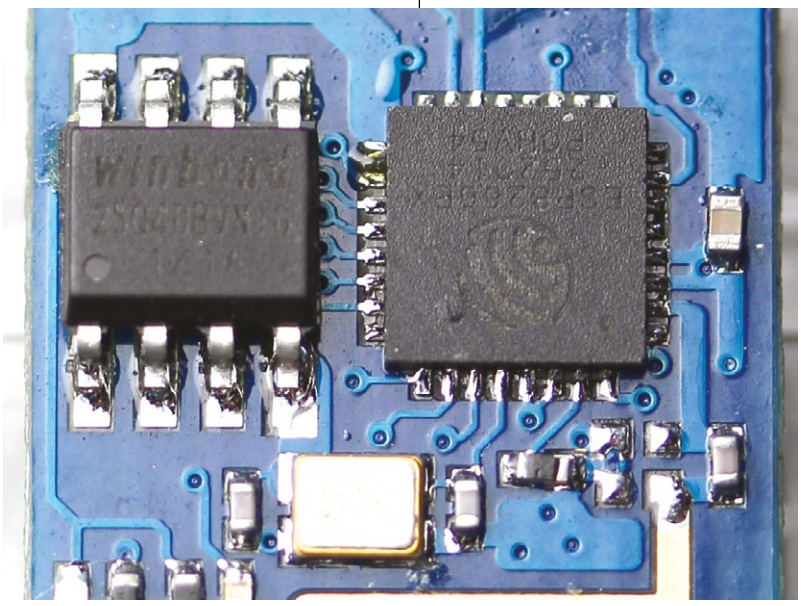
```
void setup() {
  // moniteur série
  Serial.begin(115200);
  // démarrage
  Serial.println("\r\nBoot...");

  // mode Wifi client
  WiFi.mode(WIFI_STA);
  // connexion
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    // impossible de se connecter au point d'accès
    // reboot après 5s
    Serial.println("Erreur connexion Wifi ! Reboot...");
    delay(5000);
    ESP.restart();
  }
```

```
  // configuration OTA
  confOTA();

  // tout est prêt,on affiche notre IP
  Serial.print("Adresse IP: ");
  Serial.println(WiFi.localIP());
}
```

Rien de bien extraordinaire ici. Comme le croquis ne fait rien de particulier, si ce n'est attendre une mise à jour, nous n'avons que peu de choses à configurer/initialiser. Nous pouvons alors conclure avec la fonction **loop()** qui, elle aussi, sera relativement simpliste :

```
void loop() {
  unsigned long currentMillis = millis();

  // 10s de passées ?
  if (currentMillis - previousMillis >= 10000) {
    previousMillis = currentMillis;
    Serial.println("pouet !");
  }

  // gestion OTA
  ArduinoOTA.handle();
}
```

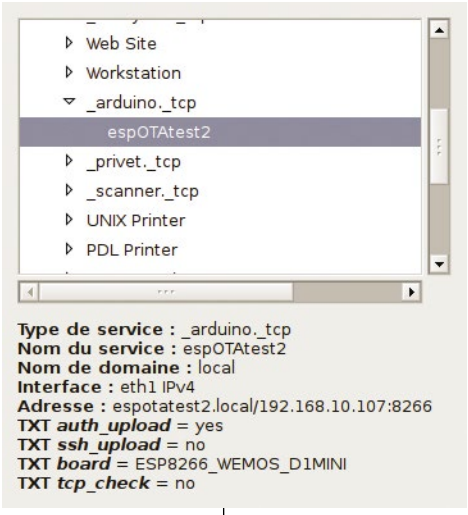
La ligne la plus importante, et la raison pour laquelle nous n'utilisons pas **delay()** est bien entendu l'appel à **ArduinoOTA.handle()** qui permet de régulièrement vérifier qu'il n'y ait pas une demande de mise à jour OTA en provenance du réseau. En l'absence de cette ligne ou si cet appel est bloqué par l'exécution d'une boucle ou d'un délai, l'ESP8266 ne sera pas en mesure de recevoir la demande et d'exécuter l'opération. C'est un point auquel il faudra être très attentif dans vos développements car, en cas d'erreur, vous serez alors à nouveau obligé d'utiliser la connexion USB pour reprogrammer la mémoire de la carte.



*Les modules ESP-01 ne disposent que de 512 Ko de mémoire flash. Nous voyons ici à droite l'ESP8266 et à gauche la mémoire SPI Winbond 25Q40BV. Avec beaucoup de patience et des doigts de fée, il est possible de dessouder ce composant au format SOIC-8 pour le remplacer par un W25Q32BV (sensiblement plus large) proposant 8 fois plus d'espace et ainsi mettre à jour votre ESP-01 pour supporter la programmation OTA. De quoi donner une nouvelle vie à ces modules pour une pincée d'euros (environ 7€ pour 10 exemplaires sur eBay).*

## 3. METTRE À JOUR EN OTA

Une fois ce croquis programmé, via USB, dans la carte ESP8266, celui-ci doit se connecter à votre réseau en Wifi. Du fait de l'utilisation de mDNS




*Le programme Avahi Discover, utilisable sur Raspberry Pi et toute machine GNU/Linux, propose une interface graphique vous permettant de facilement obtenir des informations sur un hôte et donc sur l'ESP8266 proposant le service de mise à jour OTA. Il existe également des outils de ce type pour Windows, macOS, Android, iOS, etc.*

avec l'OTA , la carte devrait être facilement détectable aussi bien via son nom que par le service qu'elle propose. Vous pouvez, depuis une Raspberry Pi ou un PC GNU/Linux, utiliser la commande **avahi-browse** pour vérifier cet état de fait :

```
$ avahi-browse -t _arduino._tcp
+   eth1 IPv4 espOTAtest2  _arduino._tcp local
```

Nous avons là un hôte appelé **espOTAtest2** proposant le service qui nous intéresse et son adresse IP est :

```
$ avahi-resolve -4 -n espOTAtest2.local
espotatest2.local  192.168.10.107
```

Dans l'environnement Arduino, le menu ***Outils > Port*** devrait vous proposer, en plus des ports série habituels, une entrée composée du nom d'hôte de l'ESP8266 et de son adresse IP. Il vous suffit de la sélectionner pour, ensuite, procéder à la programmation avec Ctrl+u, l'icône ou le menu ***Croquis > Téléverser***.

Une fois le changement de port effectué, vous ne pourrez plus utiliser le moniteur série. Le choix du port étant commun pour le moniteur et l'interface de programmation. Si vous souhaitez voir les messages de la carte ESP8266, encore connectée au PC/mac, vous devrez utiliser une autre application « terminal série » en précisant le port USB/série correspondant à la carte, ou basculer d'un port à l'autre à chaque fois.

Ceci peut paraître contraignant, mais on peut supposer qu'en phase de mise au point d'un croquis, l'ESP8266 sera de toute façon connecté à la machine de développement. On bénéficie cependant, tout de même, d'une certaine souplesse supplémentaire, car le téléchargement du croquis est bien plus rapide qu'avec la connexion USB. La logique de développement optimale consiste à utiliser un ESP8266 pour la mise au point et, une fois le croquis validé, le déployer en OTA sur les différentes cartes accessibles uniquement via le réseau.

## CONCLUSION

Une fois qu'on a testé ce type de fonctionnalité, il est relativement difficile de revenir à la méthode purement USB tant l'OTA est souple est rapide. Il ne faut cependant pas perdre de vue deux choses importantes :

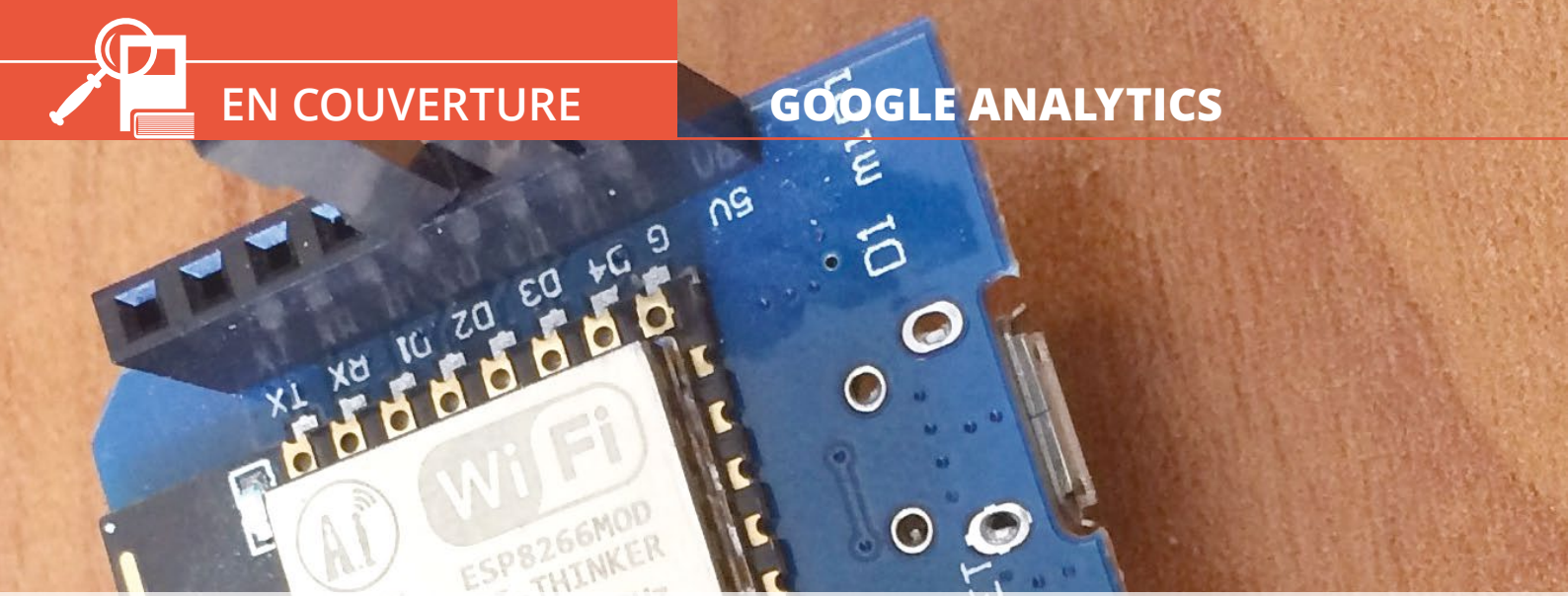
- Vos projets et donc vos croquis devront toujours prendre en compte cette fonctionnalité et ceci impactera forcément votre façon de programmer. Il vous faudra en permanence, et quoi qu'il arrive, laisser l'opportunité à **ArduinoOTA.handle()** de s'exécuter régulièrement.
- L'utilisation de l'OTA a un coût en termes d'occupation mémoire. Le simple croquis que j'ai présenté ici, une fois compilé, occupera quelques 240 Ko, soit près d'un quart de la mémoire flash disponible pour les programmes avec une carte Wemos D1 mini. Et il faut, bien entendu, le double de cet espace pour pouvoir procéder à

une mise à jour. L'ESP8266, du moins la plupart des cartes récentes, ne manque pas de mémoire et il est difficile d'imaginer se retrouver à l'étroit comme avec une Arduino UNO par exemple, mais ceci reste possible.

La mise à jour OTA via l'environnement de développement (IDE) Arduino n'est pas la seule approche proposée. Il est également possible d'utiliser un navigateur web en prenant comme base les deux exemples livrés avec le support ESP8266 : `WebUpdater` et `SecureWebUpdater`. Ceux-ci permettent d'envoyer, avec un navigateur, un fichier binaire résultant de la compilation d'un croquis devant être enregistré dans la mémoire de l'ESP8266.

J'ai délibérément omis de traiter de cette approche car, bien que tout à fait amusante, elle ne me semble pas très pratique (en dehors de situations bien spécifiques). En ce qui me concerne, pour un usage courant, le fait d'extraire le binaire pour utiliser une autre application (navigateur) pour programmer la carte ne m'apporte aucun bénéfice concret, mais uniquement davantage de manipulations. On peut supposer que cela pourrait être intéressant afin de permettre la mise à jour via un tiers : développer une nouvelle version du croquis dans l'IDE, produire un binaire, l'envoyer par mail à quelqu'un et lui demander de procéder à l'installation via OTA/Web. Si une telle situation vous concerne, jetez simplement un œil aux deux exemples en question, ils sont relativement explicites et clairs sur la base des explications données ici.

Une autre voie qui n'a pas été explorée ici consiste à utiliser une entrée de l'ESP8266 pour provoquer la mise en route d'un mode OTA. Il est en effet relativement simple dans `setup()` de tester l'état d'une broche et, en fonction du résultat, exécuter le croquis « normal » ou basculer dans une boucle appelant `ArduinoOTA.handle()` à répétition. En développant encore davantage et en utilisant un bouton, on pourrait ainsi redémarrer un ESP8266 en mode OTA et donc imposer une action physique pour permettre les mises à jour. **DB**



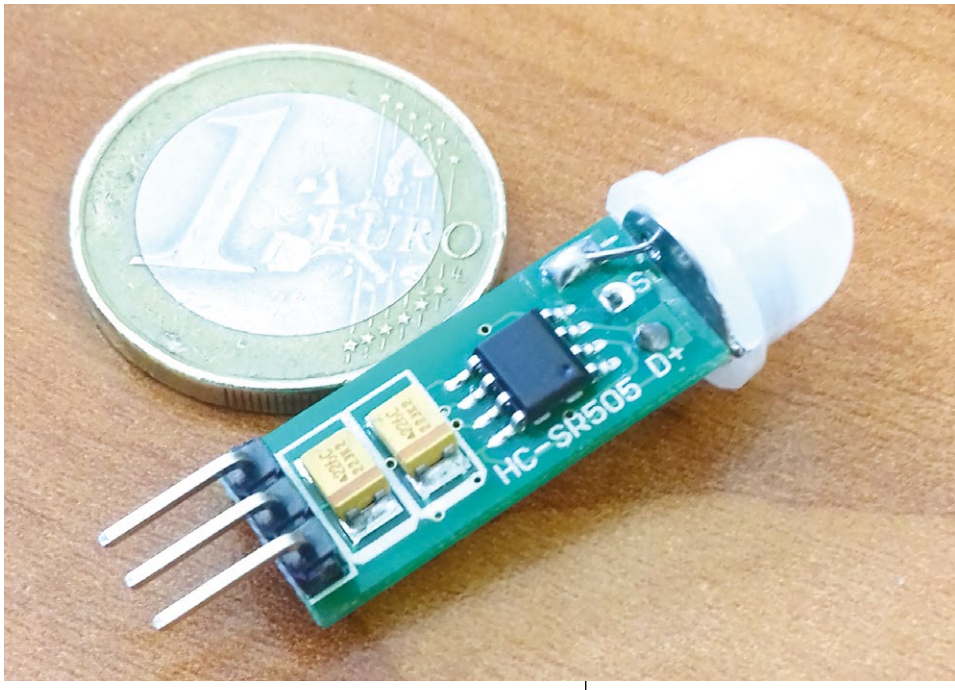

# UTILISEZ GOOGLE ANALYTICS POUR LA SURVEILLANCE PHYSIQUE

*Denis Bodor*

**Si vous disposez d'un site web, quel qu'il soit, vous connaissez sans doute Google Analytics. En incluant un petit bout de JavaScript dans vos pages HTML, vous pouvez ainsi obtenir énormément d'informations statistiques intéressantes, comme le nombre de visites, les pages les plus vues, la vitesse du site ou encore, pour les boutiques en ligne, les commandes et le chiffre d'affaires corrélés avec les visites. Mais saviez-vous que cet outil ne se limite pas qu'au Web ? Avec un peu de code et une carte connectée, il est possible d'envoyer des informations du monde réel pour étoffer vos statistiques…**

Imaginons un scénario tout simple : vous gérez une échoppe de produits du terroir type « épicerie fine », mais vous disposez également d'une boutique en ligne pour vendre vos produits. Côté web, grâce à Google Analytics, vous pouvez tout savoir du fonctionnement de votre site, le trafic, le nombre de pages vues, le rapport entre les visites et les ventes, les horaires de fréquentation minimum et maximum, les pages les plus populaires, etc. Ne serait-il pas fantastique d'avoir également ce genre d'informations pour votre boutique physique ? Mieux encore, d'avoir au même endroit les statistiques des deux boutiques ?

Voilà précisément ce que je vous propose de faire dans cet article. Bien entendu, les informations ne pourront pas être exactement similaires entre une boutique web et son alter ego physique. Le web permet par exemple de suivre un visiteur précisément, même si l'anonymat de la personne est garanti. On peut donc savoir d'où vient la personne (accès direct, chercher, autre site), quelles pages elle a visitées et si ceci a conduit à une vente ou non. Dans le cas d'un magasin bel et bien réel, il ne vous est pas possible de tracer le comportement d'un visiteur, les données sont plus générales et concerneront donc « la clientèle » au sens le plus général du terme (à moins bien entendu de taguer vos clients comme des bovins, mais je ne pense pas que ceci soit très convenable, légal ou vraiment effectivement bon pour vos affaires, les gens ont tendance à ne pas rester dans une boutique quand on leur met un piercing jaune à une oreille).

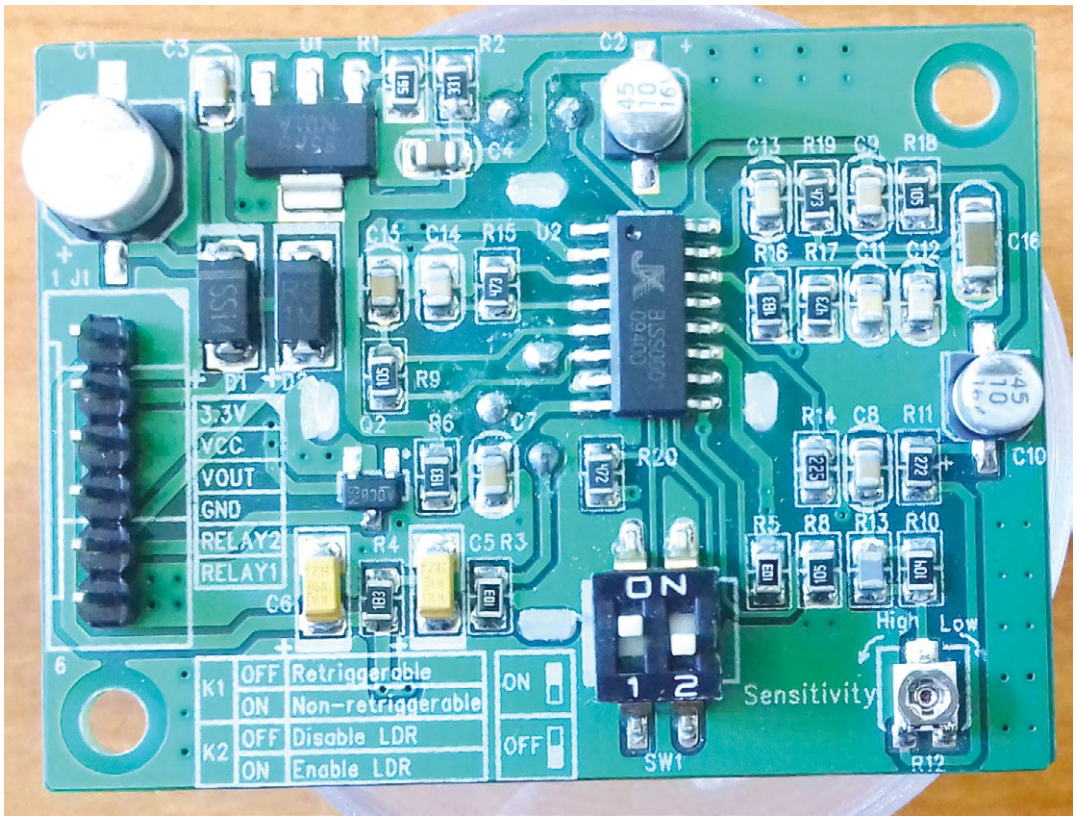



*Certains modèles de capteurs infrarouges passifs de mouvements sont réduits au strict minimum avec, comme bénéfice, une taille très réduite. Avec aucun réglage possible et une alimentation entre 4,5V et 20V, l'utilisation de ce modèle s'avère économique et simple, à défaut d'être adaptable à toutes les situations.*

Pour ce projet, nous utiliserons une carte à base d'ESP8266 et plus exactement une Wemos D1 mini, coûtant moins de 5€ et fournissant par défaut une connectivité Wifi. Les ESP8266 sont maintenant devenus aussi populaires que les cartes Arduino et se programment via l'environnement Arduino tout aussi simplement. De nombreuses cartes à base d'ESP8266 existent et toutes pourront faire l'affaire ici (sauf peut-être l'ESP-01 qui est désormais presque obsolète, car bien plus pénible à mettre en œuvre tout en ne proposant qu'une ridicule quantité de broches utilisables).

## 1. COMMENT FONCTIONNE GOOGLE ANALYTICS ?

Google Analytics n'a pas accès à la configuration des sites web, il ne fonctionne donc pas comme d'autres systèmes, tantôt plus anciens, qui analysent les journaux d'activité d'un serveur (AWStats, Webalize ou Piwik, par exemple). Google Analytics utilise une autre approche, mais




*D'autres capteurs, comme ce module fabriqué par SURE Electronics sous la référence SS502, proposent bien plus de réglages et de fonctionnalités comme le pilotage de relais, la détection jour/nuit pour la surveillance ou encore le réglage de la sensibilité du capteur PIR.*

qui ne lui est pas exclusive (Piwik peut également le faire), consistant à intégrer sur chaque page d'un site, un petit élément (en JavaScript) qui va automatiquement envoyer des informations à Google dès que ces pages sont affichées/accédées. Ce n'est donc pas votre serveur web qui envoie les données, mais le navigateur des visiteurs.

Ceci implique deux choses liées entre elles, qui ont du bon et du mauvais :

- vous pouvez vous-même envoyer des informations statistiques à la place du navigateur de vos visiteurs ;
- et n'importe qui d'autre peut également le faire à leur place.

Ceci peut paraître surprenant, car cela signifie qu'il est relativement aisé de littéralement « pourrir » les statistiques avec de fausses informations. Mais, à y bien réfléchir, cela ne serait qu'une forme automatisée d'un rechargement continu des pages avec un navigateur. Si quelqu'un de vraiment têtu et résolu à vous gâcher vos journées passe le plus clair de son temps sur votre site, en utilisant plein de navigateurs différents, débranchant et rebranchant sa box régulièrement pour changer d'adresse IP, le résultat sera presque le même : vos statistiques ne voudront plus rien dire. Ce n'est donc pas vraiment un problème de sécurité, mais une caractéristique du fonctionnement même de ce genre de collecte d'informations (une solution basée sur l'analyse de journaux d'un serveur ne résoudrait pas non plus le problème).

Le point qui nous intéresse ici est donc le premier, envoyer directement des données à Google Analytics, ne concernant pas une quelconque activité sur le Web, mais celle ayant lieu dans la vie bien réelle de tous les jours.

Ce que fait le morceau de code JavaScript intégré dans les pages

d'un site n'est pas très compliqué, il collecte les informations et envoie une requête aux serveurs de Google pour qu'elles soient archivées. Vous, de votre côté, pouvez ensuite consulter le résultat du traitement de ces informations, résumé sous la forme d'une page web garnie de graphiques, de rapports, d'analyses et de tout un tas d'autres représentations assez travaillées.

Le code en JavaScript est générique, c'est le même pour tous les sites souhaitant utiliser Google Analytics et il peut être configuré pour différentes choses. Les données collectées sont cependant systématiquement associées des « ID de suivi » identifiant votre ou vos sites. Celui-ci se présente sous la forme d'un code débutant par « UA- » suivi d'une série de chiffres. Dans le jargon de Google Analytics, un utilisateur possède un *compte*, auquel est rattaché une ou plusieurs *propriétés* qui, elles-mêmes peuvent avoir différentes *vues*.

Le compte n'est pas un compte Google (Gmail, smartphone, Play, etc.), mais un compte Google Analytics, c'est un point d'accès au service qui représente le niveau d'organisation le plus supérieur. Il peut s'agir, par exemple, d'une société ou d'une entité ayant plusieurs sites.

Une propriété désigne généralement un site web, mais il peut également s'agir d'une application mobile ou même d'une borne d'accès comme un point de vente électronique (selon la doc de Google bien qu'aucune autre information ne soit donnée sur ce point). L'ID de suivi est associé à une propriété et l'interface fournie un « code de suivi » qu'on peut directement copier/coller sur son site (le morceau de JavaScript avec l'ID de suivi intégré).

Une vue n'a rien à voir directement avec la collecte des données, c'est un point d'accès aux rapports les concernant. Lorsque vous créez une propriété, automatiquement, une vue par défaut est créée. Celle-ci est non filtrée est affichera toutes les données relatives au site, mais il vous est possible de créer d'autres vues, plus spécifiques et filtrées avec un sous-ensemble des informations (affichées uniquement à partir de la date de création de la vue en question).

Les utilisateurs du service, ceux avec un compte Google, ont des permissions spécifiques pour un compte, une propriété et/ou une vue. L'administrateur (vous) pouvez choisir qui peut gérer les utilisateurs, modifier la configuration (tâches administratives), collaborer en manipulant des éléments partagés ou simplement lire et analyser les informations. Ceci permet de faciliter la gestion du travail collaboratif en limitant au minimum les actions (et donc les bêtises) de certains utilisateurs. Ceci s'avère particulièrement pratique dans le cadre des vues puisqu'il est possible ainsi de fournir une vue filtrée et restreinte à certains collaborateurs et non à d'autres.

Je ne m'étendrai pas davantage sur le sujet, car il existe bien des documentations, tutoriels et vidéos en ligne décrivant la façon de configurer et d'utiliser Google Analytics. Une recommandation cependant, je vous déconseille d'acheter un ouvrage sur le sujet en espérant y trouver toutes les réponses. L'interface et les fonctionnalités de Google Analytics sont connues pour changer assez fréquemment, rendant souvent tout ou partie d'un ouvrage presque inutile.

Pour mettre en œuvre les explications qui vont suivre, il vous faudra donc un compte Google, un accès à Google Analytics (**https://analytics.google.com**), créer un compte au sens Google Analytics du terme et finalement créer une propriété. Et là, si vous n'avez pas de site web, vous allez rencontrer un petit problème : le système demande une « URL par défaut » et donc un nom de domaine pour le site.

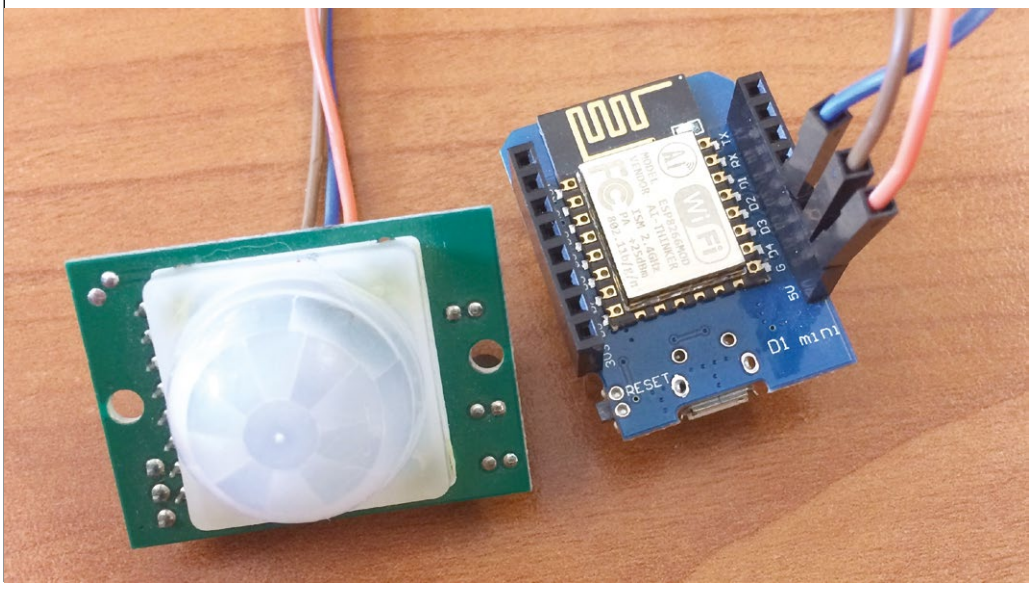
Bien que cette URL ne soit pas utilisée en pratique pour la collecte de données, je ne saurai vous recommander de mettre quelque chose au hasard, car cela

ne serait pas techniquement correct. Vous êtes censé posséder (ou pouvoir administrer) le site pointé par l'URL et devriez donc avoir, en principe, la main sur le site en question. Il existe plusieurs solutions pour disposer d'une URL légitime : acheter un nom de domaine, utiliser une plateforme de blog permettant quelque comme chose « votrenom.domaine.com » (Blogger, Blogspot ou Wordpress.com par exemple), passer par un fournisseur de sous-domaine, utiliser un site de partage de photos, etc. Certaines solutions sont gratuites, d'autres payantes (avec en prime un site ou un blog avec lequel jouer), l'important est simplement d'avoir le droit d'utiliser une URL légitimement.

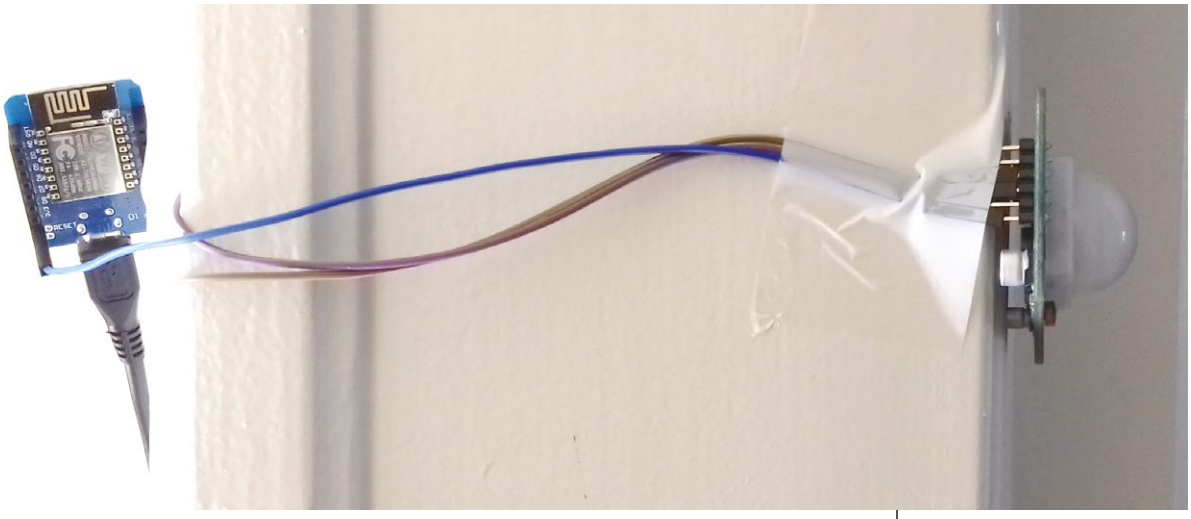
Une fois la propriété créée vous obtiendrez l'ID de suivi dont j'ai parlé précédemment. Le code JavaScript à intégrer dans le site ne nous intéressera pas ici puisque nous ne travaillerons pas avec un vrai site. Notre projet utilisera une fonctionnalité particulière de Google Analytics appelée *Measurement Protocol*. Celle-ci permet, en utilisant simplement l'ID de suivi, d'envoyer des données à Google Analytics via une simple requête HTTP **POST**. J'ai bien dit HTTP, il n'est donc pas nécessaire de vous soucier du support SSL/TLS (pour HTTPS) comme c'est le cas pour d'autres services en ligne (comme Twitter par exemple).

## 2. UN MONTAGE QUI PERMET D'ENVOYER DES DONNÉES À GOOGLE

Certes, dit comme ça cela peut faire peur, en particulier en raison du climat de surveillance global qu'on peut estimer parfaitement réel (je ne rentrerai pas dans ce débat, ce n'est pas le sujet). Quoi qu'il en soit, ici, il ne s'agit pas d'envoyer d'informations nominatives, loin de là. Outre le fait qu'il est strictement interdit de stocker des informations personnelles via Google Analytics (mais uniquement des identifiants), nous n'aurons pas l'occasion ici d'avoir ne serait-ce que ces données en notre possession. Nous ne verrons que des mouvements parfaitement anonymes.




*La connexion du module de détection infrarouge avec la carte ESP8266 Wemos D1 mini (ou toute autre carte) se limite à trois connexions, deux pour l'alimentation 5V et un pour le signal de détection de mouvement.*



*Voici l'installation hautement technologique d'un premier capteur PIR wifi utilisé pour les tests lors de la rédaction de cet article. Les choses se sont encore grandement améliorées par la suite avec l'utilisation de punaises sur un tableau en liège... Hé oui, on ne se refuse rien à la rédaction de Hackable, et certainement pas d'utiliser les méthodes « MacGyver » (le vrai, pas celui du reboot pathétique de 2016) !*

L'idée du présent projet repose sur une détection de mouvement au moyen d'un capteur infrarouge. D'autres biais peuvent être envisagés de la même façon avec un simple contact, une barrière laser, une photodiode, un capteur de pression, un contacteur au mercure, un capteur magnétique (comme ceux équipant les fenêtres pour les systèmes d'alarme), un capteur de vibration, un détecteur à ultrason, etc.

Dans le cas des capteurs de mouvement infrarouges, la mise en œuvre est relativement simple. Ces modules, appelés également capteurs PIR, existent dans différentes versions, incluant plus ou moins de fonctionnalités. Dans sa déclinaison la plus simple, le module s'alimente par deux broches (masse et tension d'alimentation) et propose une sortie mise à l'état haut lorsqu'un mouvement a été détecté en face du capteur. Les modèles plus complets ajoutent généralement des ports pour piloter directement des relais, un réglage de la sensibilité et une photorésistance (ou LDR pour *Light-Dependent Resistor*) dont l'utilisation peut être activée ou non afin de ne capter les mouvements que dans l'obscurité (typique d'une utilisation pour un éclairage automatique).

Il y a deux choses à savoir concernant ces modules :

- L'alimentation n'est pas toujours de 5V. En fonction du modèle, il est important de lire la documentation, qu'elle soit directement fournie par le vendeur ou qu'elle doive être trouvée sur le Web. Certains modules disposent d'un régulateur de tension acceptant entre 4,5V et 20V, d'autres demandent 12V et d'autres encore proposent une broche particulière pour une alimentation en 3,3V.
- Le déclenchement est tantôt configurable selon deux modes distincts. Le mode dit « normal » provoque un changement d'état à chaque détection. La sortie du module passe de l'état bas à l'état haut pour signifier une détection, mais si le mouvement continue face au capteur, le basculement de l'état va continuer. L'autre mode appelé « *retrigger* » (redéclenchement) change ce comportement. Une première détection passera l'état de la sortie à haut et ceci ne changera pas tant qu'un mouvement sera détecté. Ces deux modes sont également parfois appelés « *Repeat Trigger* » (déclenchements répétés) et « Single Trigger » (déclenchement unique).

Ce dernier point est important dans notre projet, car nous allons surveiller les changements d'état et les compter. Il est donc plus intéressant, à mon avis, d'utiliser le mode « *retrigger* » où des mouvements continus devant le capteur ne comptent que pour un seul événement. Bien entendu, cela dépend également du délai qu'utilise le module avant de repasser la sortie à l'état bas dans un mode ou l'autre. Notez que les modules les moins coûteux (et les plus petits) ne proposent aucun réglage et sont souvent par défaut dans ce mode.

Vous l'aurez compris, nous n'aurons besoin ici que d'une seule broche de la carte ESP8266 et j'ai arbitrairement choisi celle libellée D3. La raison en est fort simple, elle se trouve juste à côté de D4 (qui est connecté à la led sur la carte), elle-même se trouvant à côté de la masse et de la broche 5V, les deux fournissant l'alimentation directement au module de détection PIR.

Pour prendre en charge le changement d'état sur D3, nous utiliserons une interruption. Nous commencerons donc par écrire notre routine qui, je le rappelle, se doit d'être concise :

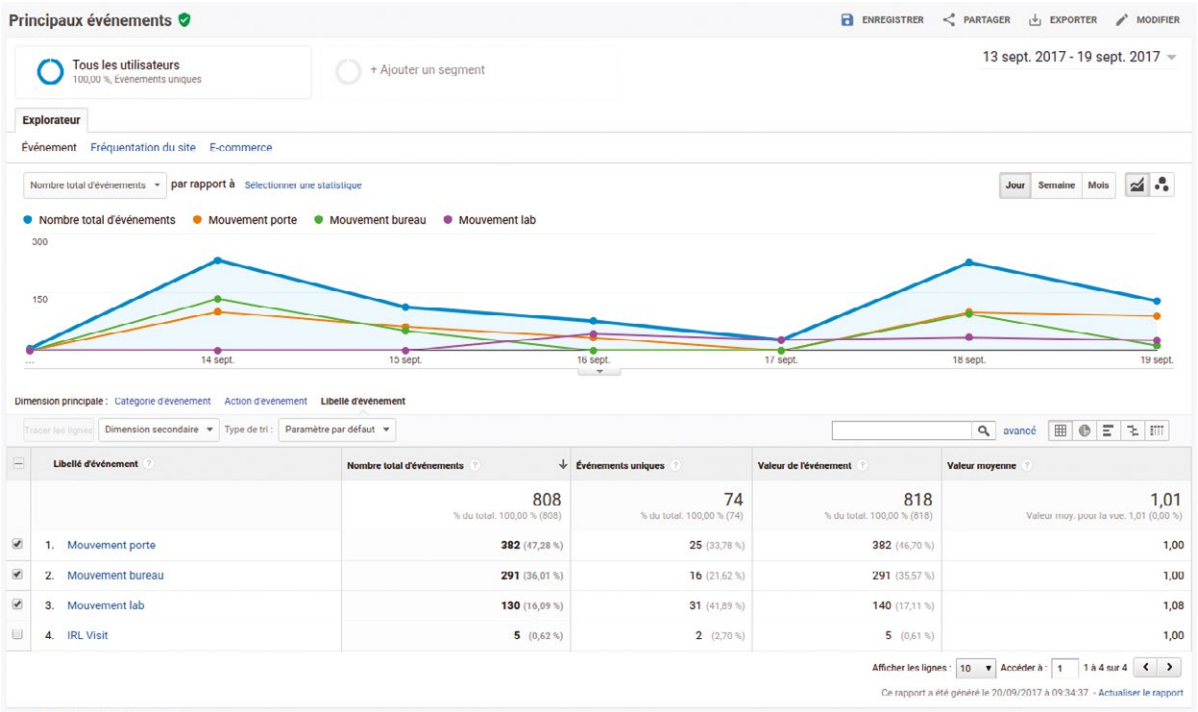
```
volatile int compteur = 0;

void IRint() {
  digitalWrite(D4, LOW);
  compteur++;
}
```

C'est ensuite, dans la fonction **setup()** que nous configurerons l'appel à cette routine en cas d'un passage de l'état bas à l'état haut sur D3 :

```
// broche D3 en entrée
pinMode(D3, INPUT_PULLUP);

// association avec la routine
attachInterrupt(digitalPinToInterrupt(D3), IRint, RISING);

// activation des interruptions
sei();
```

Dès que le changement d'état aura lieu, le fonctionnement normal du croquis sera interrompu et la routine sera exécutée, incrémentant la variable **compteur** et allumant la led de la carte (notez l'inversion de la logique, la led s'allume en mettant D4 à la masse). Pour agir en fonction de cette valeur, nous n'avons qu'à faire un test dans la fonction **loop()**, à intervalle régulier :

```
unsigned long previousMillis = 0;

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= 10000) {
    previousMillis = currentMillis;
    if(compteur) {
      Serial.print("compteur= ");
      Serial.println(compteur);
      compteur=0;
      digitalWrite(D4, HIGH);
    }
  }
}
```




*Après quelques jours de fonctionnement de trois capteurs Wifi installés, il devient intéressant d'observer les données. Notez la mention de « IRL Visit » en bas de la capture, provenant d'un premier test et bien entendu impossible à supprimer de Google Analytics...*

Nous pourrions faire de même avec un `delay(10000)` et donc sans utiliser `millis()`, mais mieux vaut prévoir pour plus tard et directement permettre de facilement étendre le croquis par la suite. `delay()` est simple à utiliser, mais bloque le fonctionnement normal du croquis (sauf pour l'interruption bien entendu) et ceci pourrait rendre difficile l'ajout de fonctionnalités. Quoi qu'il en soit, ici, toutes les 10 secondes nous testons `compteur` et si son contenu est non nul, nous affichons sa valeur et plaçons D4 à la masse afin d'éteindre la led qui aura été allumée dans la première exécution de la routine d'interruption.

Nous avons à présent la base de notre croquis permettant d'obtenir l'information souhaitée depuis le capteur PIR. Il ne nous reste plus qu'à écrire le reste du croquis comprenant la connexion à notre point d'accès Wifi et surtout la fonction qui va envoyer les données à Google.

## 3. ÊTRE UN CLIENT WEB EN TOUTE SIMPLICITÉ

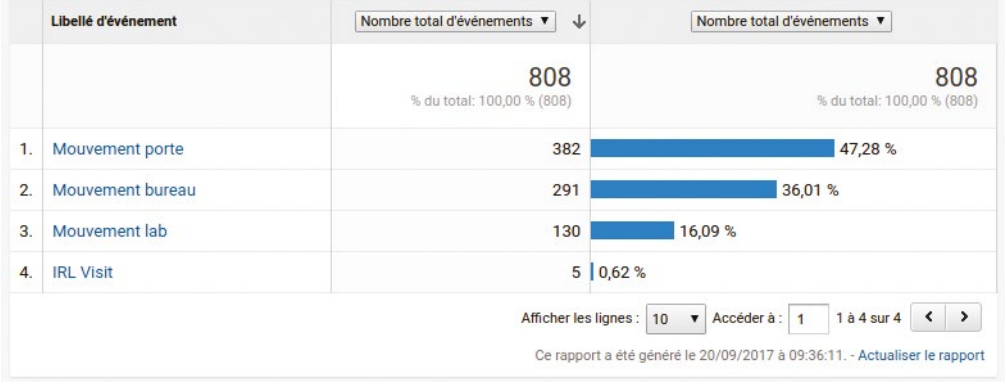
Lorsqu'on cherche sur le net comment agir en tant que client web avec une carte à base d'ESP8266, on trouve généralement des croquis gérant cela au plus bas niveau. Ces exemples et réalisations établissent une connexion avec un serveur, composent toute la requête « à la main » et analysent la réponse obtenue après envoi. C'est très surprenant, car il existe une approche bien plus simple. En effet, le support ESP8266 pour Arduino comprend une bibliothèque spécialement développée pour gérer les requêtes HTTP : `ESP8266HTTPClient.h`.

Il est alors totalement inutile de faire tout le travail nous-mêmes puisqu'il suffit de reposer sur les fonctions et méthodes prévues à cet effet, et donc d'éviter toute la partie consistant à gérer et disséquer la réponse (et les codes HTTP) envoyée par le serveur. Toutes les manipulations de chaînes de caractères sont ainsi faites automatiquement pour nous.

Le « *Measurement Protocol* » nous permettant de fournir nos données à Google Analytics n'est pas très complexe. Il s'agit simplement d'envoyer une requête **POST** comme celle générée lorsque vous validez un formulaire dans un navigateur web. Ici, cependant, nous sautons une étape et construisons directement la requête sous la forme d'une URL composée de façon tout à fait standard vers **http://www.google-analytics.com/collect**. Il suffit de spécifier les différents paramètres, séparés d'un **&** :

- **v=** : la version du protocole utilisé (toujours **v=1** ici) ;
- **tid=** : l'ID de suivi récupéré sur Google Analytics au format **UA-xxxxxxxx-x** (où les **x** sont des chiffres) ;
- **cid=** : l'ID client correspondant normalement à un identifiant unique de visiteur/client, mais nous n'en avons pas l'usage ici. Ce paramètre est cependant indispensable si on ne précise pas un UID, avec **uid=**, permettant d'identifier un utilisateur (notez que l'UID ne doit **jamais** être une information personnelle identifiable) ;
- **t=** : le type de *hit* ou, autrement dit, d'action ayant provoqué l'envoi de données. Nous utiliserons ici le type **event**, un événement, mais il peut également s'agir, par exemple, de **pageview** pour l'affichage d'une page web ou de **transaction** pour une transaction en commerce électronique ;
- **ec=** : ce paramètre, comme les suivants, est lié au type **event**, il s'agit d'un « mot » précisant la catégorie de l'événement qui a déclenché l'envoi. Nous utiliserons le terme « physique » comme il s'agit d'un événement de « la vraie vie » ;
- **ea=** : précise l'action qui vient de se dérouler. Comme nous utilisons un capteur de mouvement PIR, le mot « mouvement » paraît assez judicieux ;
- **el=** : c'est le label (ou libellé) caractérisant l'événement et, là encore, le choix est totalement arbitraire. Personnellement, pour ce type d'usage c'est ici que je précise l'endroit où se trouve le capteur : « bureau », « couloir », « porte », « entrée », « lab », etc. ;
- **ev=** : précise, enfin, une valeur pour événement, qui peut être, comme nous allons le voir dans le code qui va suivre, la valeur de **compteur** ou, autrement dit, le nombre de déclenchements du capteur sur l'intervalle écoulé (10s).

*Différentes représentations peuvent être choisies pour l'affichage des données collectées et analysées. Ici nous voyons la répartition des mouvements captés sous la forme de barres avec un pourcentage, mais l'affichage « camembert » est également possible.*



Remarquez que la catégorie, l'action et le label d'un événement sont déterminés directement dans la requête sans avoir besoin d'une quelconque configuration côté Google Analytics. Si vous laissez fonctionner votre montage pendant une semaine avec une catégorie « physque » et remarquez ensuite la faute de frappe, pour la corriger en « physique », Google Analytics enregistrera, depuis le début des envois, deux catégories, « physque » et « physique » . Et il n'est, bien entendu, pas possible de corriger des données déjà envoyées...

Sur cette base, nous pouvons donc écrire notre fonction provoquant à la fois la connexion au serveur Google, l'envoi de la requête et l'obtention du code retour HTTP pour vérification :

```
int envoiGA(int nbr) {
  // objet représentant la requête
  HTTPClient http;

  // valeur de retour, par défaut en erreur
  int ret = -1;

  // la requête POST
  // (sur deux lignes pour les besoins de la mise en page)
  String eventData = "v=1&tid=UA-xxxxxxxx-x&cid=0000&t=event&"
                     "ec=physique&ea=mouvement&el=Mouvement lab&ev=";
  // plus la valeur du compteur
  eventData += nbr;

  Serial.print("Envoi GA: ");
  Serial.println(eventData);

  // sommes-nous bien connectés ?
  if(WiFi.status()== WL_CONNECTED) {
    // oui, début de l'envoi
    http.begin("http://www.google-analytics.com/collect");
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    // récupération de la réponse
    int httpCode = http.POST(eventData);

    Serial.print("Reponse GA: ");
    Serial.println(httpCode);
    http.end();

    // 200 signifie "succès de la requête"
    if(httpCode == 200) {
      // on retourne 0
      ret = 0;
    }
  }

  // si on arrive ici, il y a eu un problème
  // et on retourne -1
  return ret;
}
```

Nous utilisons ici un type « événement », puisque cela semble le choix le plus naturel, mais il est possible d'adapter cette fonction à d'autres usages (comme le commerce par exemple) en utilisant un autre type. Référez-vous à la documentation Google sur le « *Measurement Protocol* » pour en savoir plus : **https://developers.google.com/analytics/devguides/collection/protocol/v1/devguide**.

Il ne nous reste plus, à ce stade, qu'à assembler le tout en ajoutant, dans **setup**, de quoi nous connecter en Wifi :

```
const char* ssid = "SSID_AP_WIFI";
const char* password = "mot_de_passe";

void setup() {
  uint8_t i = 0;

  pinMode(D4, OUTPUT);
  digitalWrite(D4, HIGH);

  Serial.begin(115200);

  // connexion Wifi client (STAtion)
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.print("Connexion à ");
  Serial.println(ssid);
  while (WiFi.status() != WL_CONNECTED && i++ < 20) delay(500);
  if(i == 21){
    Serial.print("Erreur de connexion sur "); Serial.println(ssid);
    while (1) {
      // erreur DHCP
    }
  }

  // Affichage adresse
  Serial.print("adresse IP: ");
  Serial.println(WiFi.localIP());
[...]
```

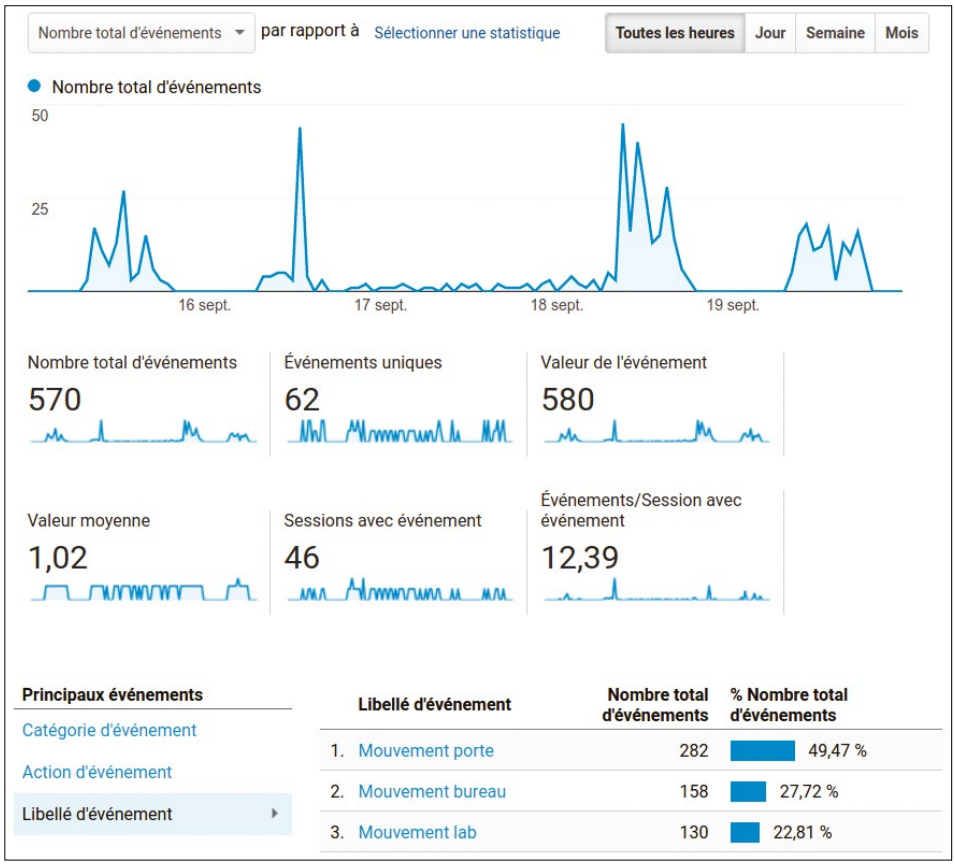
Le tout, sans oublier d'intégrer l'appel à la fonction **envoiGA()** dans **loop()**, juste avant de remettre à 0 la variable **compteur**.

## 4. RÉSULTATS ET ÉVOLUTIONS POSSIBLES

Les premiers essais du montage se feront sans doute loin de son point d'installation, afin de simplement vérifier que les mouvements sont correctement détectés, que le croquis les prend effectivement en compte et que les données soient correctement envoyées au serveur Google. Cette dernière phase du développement est facilitée par une fonction intéressante de Google Analytics permettant de voir en « temps réel » l'activité d'un site.

Rendez-vous donc sur **https://analytics.google.com** et dans le menu ***Temps réel*** choisissez ***Événements***. Avec un décalage d'une seconde ou deux, un mouvement devrait provoquer l'allumage de la led sur la carte, l'affichage des messages sur le moniteur série et, peu après, l'apparition d'un pic d'activité sur l'interface Google Analytics.

Malheureusement, il ne vous sera pas possible de voir le label de l'événement en question puisqu'il n'est pas affiché dans cette vue. Il faudra alors installer le ou les capteurs et commencer à réellement collecter des données pour enfin, après quelques jours, vous rendre dans ***Comportement***, ***Événements***, ***Principaux événements*** et afficher le graphique par ***Libellé d'événement***. Bien entendu, plus le temps passera plus les informations seront intéressantes.




*Certains graphiques d'une vue sont plus précis que d'autres et offrent une résolution horaire et non simplement par jour, semaine ou mois. Il n'est cependant pas possible de tracer simultanément les différents libellés ici. Google Analytics est très étoffé et ne cesse d'évoluer, cela sera peut-être possible dans un futur plus ou moins proche.*

Il est possible de grandement étendre le fonctionnement de ce montage, avec toutes sortes de capteurs. Certaines informations cependant ne sont pas adaptées pour ce type de collecte. Je pense par exemple à des relevés de températures, de pression ou d'humidité relative. Il est, bien entendu possible de tricher en utilisant d'autres types d'informations, le montant d'une transaction, par exemple, est une valeur en virgule flottante et des degrés pourraient se substituer aux euros. Il est également possible de simplement multiplier par 100 une valeur pour obtenir un entier et l'utiliser comme valeur d'événement, mais là encore, ce n'est pas vraiment correct. Après tout, une mesure récurrente n'est pas vraiment un événement...

Il y a donc clairement des limites avec ce type d'utilisation de Google Analytics, mais dès lors qu'il s'agit de « trafic physique », pouvant être le pendant des visites sur un site web, il n'y a aucun problème. Dans le scénario de l'épicerie ou de la boutique ayant pignon sur rue, le fait de considérer l'établissement comme un site de e-commerce peut être poussé encore davantage. La technologie utilisée par les caisses enregistreuses ne m'est pas familière, mais il doit exister une possibilité de collecter des données et les considérer comme des transactions. Après tout, l'objet même d'une caisse électronique est bien de faciliter la comptabilité et donc d'enregistrer des transactions. Ceci **doit** être faisable d'une façon ou d'une autre.

Une autre utilisation possible pourrait impliquer la mise en œuvre de tags NFC ou RFID permettant à l'ESP8266 de lire un identifiant stocké et donc d'avoir une information plus fiable sur les « clients » ou les « utilisateurs ». On peut également imaginer interfacer un équipement existant comme une machine à café, un distributeur automatique de friandises ou même des installations sanitaires (WC, lavabo, etc.). Les idées ne manquent pas et encore une fois, l'ESP8266 démontre clairement qu'il mérite les mêmes honneurs que les cartes Arduino ou Launchpad en termes de souplesse d'utilisation et de richesse en fonctionnalités, sinon davantage... **DB**

# mDNS OU COMMENT RETROUVER FACILEMENT VOTRE PI

*Denis Bodor*

**… sur le réseau, mais aussi n'importe quelle autre machine sans avoir son adresse IP. Vous connaissez le problème, soit vous attribuez une adresse statique à chaque machine de votre réseau avec une configuration dans votre box ou votre propre serveur DHCP, soit l'attribution d'adresse est dynamique. Dans les deux cas, il faut connaître l'adresse du système auquel vous voulez vous connecter. Il existe cependant un mécanisme appelé Multicast DNS (mDNS) qui vous permet d'éviter cela. Jetons donc un œil…**

Avant toutes choses, précisons que tout ceci n'est ni nouveau ni même à configurer sur votre Pi (ou plus exactement dans Raspbian). mDNS est un protocole qui ne date pas d'hier, mais, en plus, ceci est pris en charge par défaut par votre Pi, ainsi que votre système pour PC/Mac qu'il s'agisse de Windows, macOS ou GNU/Linux. Il s'agit donc ici de simplement savoir que cela existe, comprendre comment cela fonctionne et quelles sont les implications pour votre réseau.

## 1. CONNAÎTRE LA BONNE ADRESSE

Le problème de l'adressage est présent depuis que les réseaux eux-mêmes existent. Chaque machine doit être identifiée par une adresse afin que les autres, sur le même réseau puissent y accéder. La solution la plus ancienne et la plus simple pour régler ce problème est l'adressage purement statique : on configure chaque machine avec une adresse unique et on se souvient simplement de toutes les correspondances. Ceci implique donc que pour chaque nouvelle machine du réseau, il est nécessaire de configurer une nouvelle adresse différente de celles déjà utilisées. Bien entendu, si on veut changer l'adresse, il faut alors changer la configuration du système. Rien n'est centralisé.

Avec un réseau susceptible de changer fréquemment cela devient vite pénible et le système a donc été automatisé. Il faut savoir, en effet, que l'adresse IP d'une machine (ou plus exactement d'une interface réseau d'une machine) peut être configurée arbitrairement, mais que les interfaces réseau (Ethernet, ATM, Token Ring, Wifi, etc.) disposent d'une autre adresse, fixe, physique : l'adresse MAC (pour *Media Access Control*). C'est un identifiant unique stocké dans une interface réseau dont la valeur est fixée par le constructeur et est réputée unique. Celle-ci prend la forme de 48 bits (6 octets) représentés sous la forme de 6 valeurs hexadécimales séparées par un double-point, exemple : **b8:27:eb:ba:b5:02**. Vous pouvez connaître l'adresse MAC des interfaces de votre Raspberry Pi en utilisant la commande **ifconfig** ou **ip addr** :

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:ba:b5:02
          inet adr:192.168.10.234  Bcast:192.168.10.255
Masque:255.255.255.0
          adr inet6: fe80::e3e2:84e8:e981:c648/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:178270 errors:0 dropped:21912 overruns:0 frame:0
          TX packets:16639 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:38201219 (36.4 MiB)  TX bytes:1916257 (1.8 MiB)
```

Grâce à cette adresse MAC, il est possible de créer une correspondance avec une adresse IP en mettant en œuvre un protocole particulier appelé DHCP. En effet, lorsque deux machines discutent entre elles, elles le font via des signaux électriques transmis sur un câble réseau (ou dans les airs). Du point de vue du matériel, la notion d'adresse IP n'existe pas. En utilisant une adresse IP pour contacter une machine inconnue, le système commence par demander l'adresse MAC de cette dernière avec un autre protocole : ARP (*Address Resolution Protocol*). Celui-ci tient alors à jour une table de correspondance entre adresse MAC et adresse IP.

DHCP est autre chose, c'est un « protocole de configuration dynamique des hôtes » (ou *Dynamic Host Configuration Protocol* en anglais, DHCP). Celui-ci sert, pour une machine, à connaître son adresse IP et à configurer son système automatiquement. Au démarrage par exemple, votre


*Tous les modèles de Raspberry Pi ne disposent pas d'un port Ethernet, mais ceci reste, à mon avis, la solution la plus fiable et stable pour installer votre Pi dans votre réseau local. Personnellement, ce n'est que lorsque je n'ai pas le choix que j'opte pour le Wifi, mais dans les deux cas, filaire ou non, mDNS est parfaitement utilisable.*

Raspberry Pi va envoyer une requête DHCP demandant « voici mon adresse MAC, quelle est mon adresse IP ? ». Si un serveur DHCP est présent sur le réseau, celui-ci peut répondre et fournir une adresse à la Pi : « Ok, voici ton adresse IP » (dans les faits, c'est tous les paramètres réseau qui peuvent être ainsi transmis).

Le serveur DHCP a pour tâche de faire en sorte que chaque machine ait une adresse IP différente sur le réseau. Deux cas de figure sont généralement envisagés : distribuer les adresses séquentiellement, le premier qui demande est le premier servi, ou disposer d'une configuration où une liste d'adresses MAC établit les correspondances avec les adresses IP. Dans le cas d'une box ADSL, le plus souvent, la première solution est utilisée et le premier ordinateur qui demande une adresse obtient la première disponible.

Ce style de configuration implique, bien entendu, que tout est fait dynamiquement, mais aussi qu'une même machine pourra avoir une adresse différente selon l'ordre de démarrage des différents systèmes. En d'autres termes, il est tout à fait possible qu'une Pi soit accessible à un moment avec une adresse puis, quelques jours après, après plusieurs démarrages, avec une autre.

Ce système est loin d'être parfait car, soit vous ne savez jamais avec certitude quelle adresse a votre PI, soit, si vous optez pour une liste IP/MAC, vous êtes obligé de reconfigurer le serveur DHCP pour chaque nouvelle machine ou carte (et vous êtes toujours obligé de vous souvenir de l'adresse IP). C'est mieux qu'une configuration statique par système, mais c'est perfectible.

## 2. DE L'ADRESSE VERS LE NOM

Demander à un humain « normal » de devoir se souvenir de quelque chose comme **192.168.10.234** n'est pas beaucoup plus facile qu'avec **b8:27:eb:ba:b5:02**. Ainsi, pour accéder à une machine sur Internet, on utilise rarement une adresse, mais plutôt un nom. Ce nom, associé à un identifiant de protocole et un emplacement, forme une URL (*Uniform Resource Locator*).

L'URL « **https://boutique.ed-diamond.com/en-kiosque/1269-hackable-magazine-20.html** » par exemple, référence le fichier **1269-hackable-magazine-20.html**, dans le « répertoire » (chemin) **en-kiosque**, sur la machine nommée **boutique** du domaine **ed-diamond.com** accédé via le protocole HTTPS. Cette machine est cependant accessible en réalité par une adresse IP : 213.162.55.165.

Il y a donc un mécanisme en place pour associer un nom complet de machine avec une adresse IP. Ce nom complet, composé du nom d'hôte de la machine (**boutique**) et du domaine (**ed-diamond.com**) est appelé un nom de domaine pleinement qualifié ou FQDN en anglais (pour *Fully Qualified Domain Name*). Et, quelque part, un serveur a donné

à votre machine l'adresse IP correspondante, un serveur DNS (*Domain Name System*).

Je n'entrerai pas dans le détail ici, mais m'en tiendrais à vulgariser le concept comme précédemment. Un DNS est un service fournissant, entre autres choses, une adresse IP en fonction d'un FQDN. Comme le système est global à tout Internet, il n'existe pas qu'un seul serveur DNS, mais toute une tripotée constituée selon une hiérarchie bien précise et les informations sont également dupliquées d'un serveur à l'autre selon des règles strictes pour assurer le fonctionnement de l'ensemble même si un ou plusieurs serveurs tombent en panne.

Lorsque vous entrez une URL dans votre navigateur il y a tout un lot de mécanismes qui se mettent en place, impliquant des serveurs partout dans le monde, dialoguant avec des protocoles complexes, à la vitesse de la lumière (ou presque), tout ça pour que vous puissiez voir des vidéos de chatons sur votre écran (ou d'autres choses). Mais ceci résout le problème initial : il n'est plus nécessaire de mémoriser des adresses IP et surtout, la correspondance peut être changée à volonté avec un minimum d'actions.

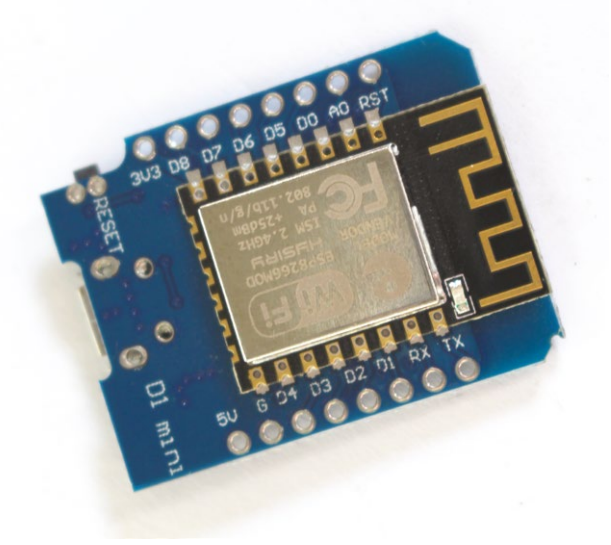
En revanche, mettre un tel système en place, un serveur DNS local pour votre réseau, bien que parfaitement possible, est loin d'être aisé et demanderait des efforts non négligeables pour configurer tout cela de manière synchronisée avec le serveur DHCP. Le jeu n'en vaut tout simplement pas la chandelle, d'autant qu'autre chose est déjà en place et bien plus simple à utiliser...

## 3. LA SOLUTION : MDNS

Idée toute bête : et si chaque machine du réseau donnait elle-même la correspondance entre son nom et son adresse IP ? Ce mécanisme serait assez similaire à DHCP finalement, de la même manière qu'une machine peut hurler « c'est quoi mon IP ? » en espérant une réponse, elle pourrait tout aussi bien crier « c'est quoi l'IP de XXX ? » en attendant la réponse de l'intéressé. J'utilise délibérément les termes « hurler » et « crier » ici puisque ces messages ne sont pas adressés à une machine en particulier, mais diffusés ou multidiffusés sur tout le réseau et n'importe quelle machine peut les recevoir.

Cette approche, totalement différente de la résolution de noms DNS, est celle proposée par le protocole mDNS qui fait parti d'un ensemble, plus large, de protocoles appelés Zeroconf. Cet ensemble a été créé pour permettre de simplifier la création automatique d'un réseau local (non Internet donc) et plus particulièrement d'un réseau domestique (par opposition à un réseau « professionnel »). On y trouve mDNS pour remplacer DNS, mais également IPv4LL/APIPA pour remplacer DHCP, DNS-SD pour trouver les services (partages, impression, etc.) ou encore NAT-PMP permettant de rendre accessible une machine depuis l'extérieur du réseau local (mécanisme NAT).

Avec Zeroconf, il devient possible pour un utilisateur n'ayant aucune connaissance particulière en réseau,




*Un petit module comme celui-ci, utilisé avec l'environnement Arduino, est une alternative économique, puissante et connectée aux cartes officielles Arduino à base de microcontrôleurs Atmel AVR. Il existe quelques subtilités dans leur utilisation comparée aux Arduino, mais, dans l'ensemble, la transition d'une plateforme à l'autre est relativement aisée.*

de simplement brancher une machine, que ce soit un ordinateur ou une imprimante réseau par exemple, et d'immédiatement l'utiliser. Ce type de simplicité est généralement très apprécié des utilisateurs Apple et ce n'est donc pas une surprise qu'une des premières mises en œuvre de Zeroconf ait été faite chez la firme à la pomme, sous le nom de « Rendez-vous », devenu ensuite « Bonjour ».

L'ensemble des autres systèmes a rapidement emboîté le pas que ce soit Windows ou GNU/Linux. Le système Raspbian de votre Pi dispose de ces fonctionnalités, et en particulier de mDNS, via la bibliothèque Avahi, implémentant presque tous les protocoles Zeroconf. Celle-ci s'accompagne d'un démon (un processus fonctionnant en permanence en mémoire) assurant les services Zeroconf et donc toute la mécanique nécessaire. Avahi est installée par défaut dans Raspbian et, sans que vous le sachiez, fait déjà ce travail pour vous.

Dans la très grande majorité des cas, dès lors que vous branchez votre Raspberry Pi au réseau et la démarrez, celle-ci va répondre aux demandes mDNS qui la concernent. Le nom de votre système sur le réseau sera son nom d'hôte (« raspberrypi » par défaut) associé au domaine « .local ». Vous pourrez accéder à votre système depuis n'importe quelle machine utilisant également les protocoles Zeroconf et en particulier mDNS. Pour cela, il vous suffit d'accéder à **raspberrypi.local** :

```
% ping raspberrypi.local
PING raspberrypi.local (192.168.10.234) 56(84) bytes of data.
64 bytes from 192.168.10.234: icmp_seq=1 ttl=64 time=0.335 ms
64 bytes from 192.168.10.234: icmp_seq=2 ttl=64 time=0.434 ms
64 bytes from 192.168.10.234: icmp_seq=3 ttl=64 time=0.387 ms
[...]
```

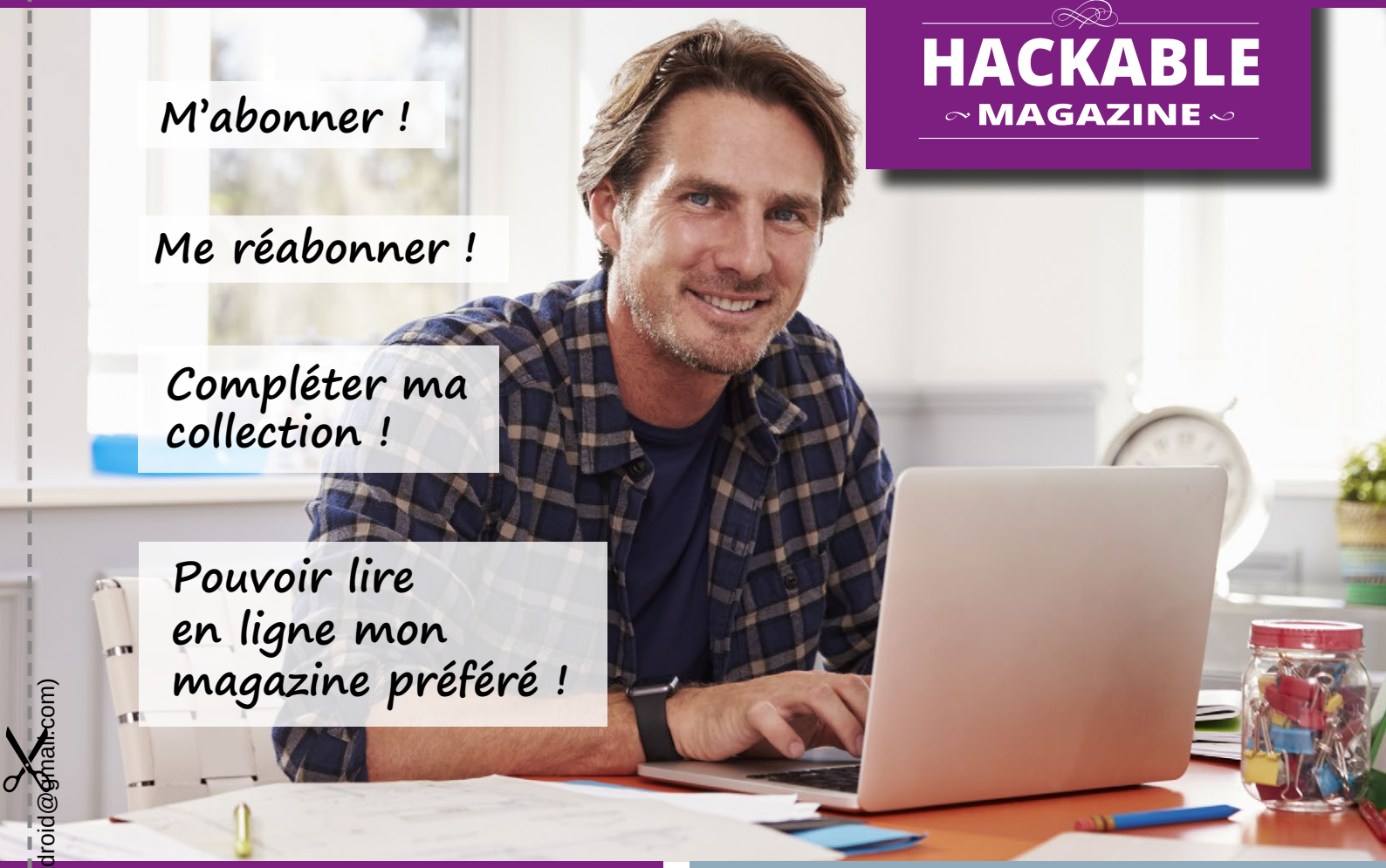
Inutile de connaître l'adresse de la Pi, ce nom est suffisant, et ce quelle que soit l'adresse attribuée par le serveur DHCP intégré à votre box ADSL ou à votre routeur Internet.

Il est cependant de bon ton de changer le nom d'hôte de votre Pi en utilisant **sudo raspi-config**, car comme celui-ci est défini par défaut de manière identique pour tous les systèmes Raspbian, toutes les Pi du réseau s'appellent donc **raspberrypi.local**. Ceci explique également pourquoi le serveur SSH (connexion en ligne de commandes à distance) est désactivé, car il deviendrait excessivement facile de prendre le contrôle d'une Pi non reconfigurée : **ssh pi@raspberrypi.local**, mot de passe **raspberry**, **sudo** et vous voilà le maître des lieux ! Ceci ne signifie pas pour autant que SSH soit dangereux ou risqué, mais il est important de changer le nom d'hôte et surtout le mot de passe de l'utilisateur **pi**, dès le premier démarrage.

Comme nous venons de le voir avec la commande **ping**, un système GNU/Linux comme Raspbian est parfaitement en mesure d'utiliser un nom avec un domaine **.local**. Ceci est rendu possible par l'utilisation du paquet **libbss-mdns** qui ajoute la résolution mDNS aux fonctionnalités GNU NSS (*Name Service Switch*). NSS est un système modulaire offrant à (presque) tous les outils du système les fonctionnalités de résolution de nom. En ajoutant **libbss-mdns** comme greffon, immédiatement et sans plus de changement, tout le système utilise mDNS. Il existe bien d'autres greffons pour plusieurs autres protocoles pouvant offrir une résolution de nom (Winbind, LDAP, fichier DB, Docker, etc.).

Cependant, pour simplement trouver l'adresse d'une machine sans la contacter directement, le plus logique est d'utiliser les outils Avahi (paquet **avahi-utils**), comme par exemple :

# Abonnez-vous !






## PARTICULIERS,

➡ Rendez-vous sur :

**www.ed-diamond.com**

pour consulter toutes les offres !

➡ ...ou renvoyez-nous le document au verso complété !

## PROFESSIONNELS,

➡ Rendez-vous sur :

**proboutique.ed-diamond.com**

pour consulter toutes les offres dédiées !

➡ ...ou renvoyez-nous le document au verso complété !

# VOICI LES OFFRES D'ABONNEMENT AVEC HACKABLE !

## CHOISISSEZ VOTRE OFFRE ! Prix TTC en Euros / France Métropolitaine*

| Offre | ABONNEMENT | PAPIER Réf | PAPIER Tarif TTC |
|---|---|---|---|
| HK | 6n° HK | ☐ HK1 | 39 € |
| | **LES COUPLAGES AVEC NOS AUTRES MAGAZINES** | | |
| I | 6n° HK + 6n° MISC | ☐ I1 | 79 € |
| I+ | 6n° HK + 6n° MISC + 2n° HS | ☐ I+1 | 99 € |
| J | 6n° HK + 11n° GLMF | ☐ J1 | 105 € |
| J+ | 6n° HK + 11n° GLMF + 6n° HS | ☐ J+1 | 159 € |
| K | 6n° HK + 6n° LP | ☐ K1 | 75 € |
| K+ | 6n° HK + 6n° LP + 3n° HS | ☐ K+1 | 99 € |
| L | 6n° HK + 6n° LP + 11n° GLMF + 6n° MISC | ☐ L1 | 189 € |
| L+ | 6n° HK + 6n° LP + 3n° HS + 11n° GLMF + 6n° HS + 6n° MISC + 2n° HS | ☐ L+1 | 289 € |

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

**J'indique l'offre si différente que celles ci-dessus :**

**J'indique la somme due (Total) : €**

**Je choisis de régler par :**

❑ **Chèque bancaire ou postal à l'ordre des Éditions Diamond (uniquement France et DOM TOM)**

❑ **Pour les règlements par virements, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20**

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

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

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

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

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : 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 :**

.....

.....



## RETROUVEZ TOUTES NOS OFFRES SUR : www.ed-diamond.com !

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

```
$ avahi-resolve -4 -n raspberrypiled.local
192.168.10.234
```

Ici nous apprenons que la machine **raspberrypiled** a pour adresse IPv4 **192.168.10.234**. Bien entendu, si cela venait à changer par la suite, car le serveur DHCP lui attribut une autre adresse, ceci serait immédiatement pris en compte.

Inversement, il est possible de connaître le nom d'une machine à partir de son adresse avec :

```
$ avahi-resolve -a 192.168.10.234
192.168.10.234  raspberrypiled.local
```

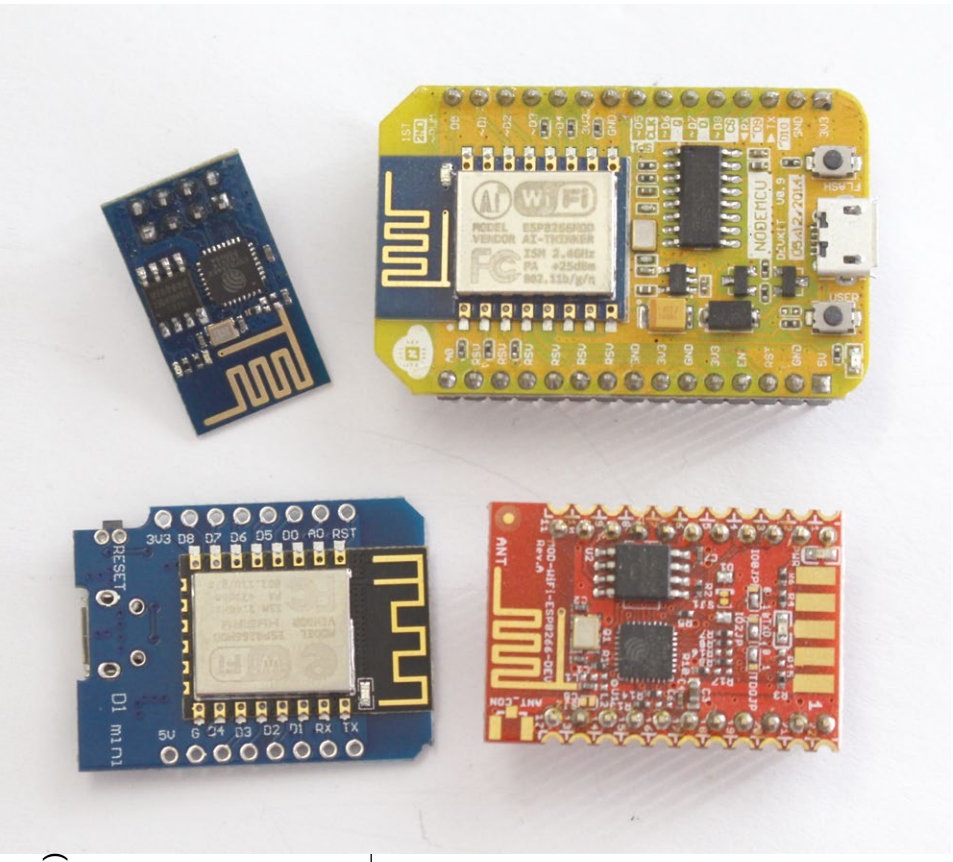
Sur cette base, on peut rapidement écrire un petit script intéressant permettant de faire le tour d'un réseau **192.168.10.*** (ou **192.168.10.0/255.255.255.0**, ou encore **192.168.10.0/24** en notation CIDR) :

```
#!/bin/bash

# permet les ctrl+c
trap "echo Exited!; exit;" SIGINT SIGTERM

# on boucle sur les IP possibles
for i in {1..254}
do
   # en ligne ?
   ping -c 1 192.168.10.$i > /dev/null
   if [ $? -eq 0 ]; then
      # oui, on le dit...
      echo -n "node 192.168.10.$i is UP  : "
      # et on cherhe le nom via mDNS
      avahi-resolve -a 192.168.10.$i
   fi
done
```

Il suffit alors de lancer ce script avec **sh nom_du_fichier.sh** pour le voir tester les 254 adresses possibles, nous dire si ces machines répondent et, le cas échéant, nous afficher leur nom via une résolution inverse mDNS. Ce genre de choses peut également être obtenu avec l'outil **avahi-browse** qui est un peu plus bavard puisqu'il retourne également les services disponibles sur les machines trouvées (avec **avahi-browse -l -a -t** par exemple). On peut même rechercher sur le réseau, par exemple les imprimantes réseau IPPS (**avahi-browse -t _ipps._tcp**), les serveurs web (**avahi-browse -t _http._tcp**) ou encore les machines offrant SSH (**avahi-browse -t _ssh._tcp**), en fournissant simplement le nom d'un type de service selon la nomenclature DNS-SD (*DNS-Service Discovery*).

## 4. PAS SEULEMENT SUR PC ET RASPBERRY PI

Vous conviendrez sans doute avec moi que ce mécanisme mDNS est fort intéressant. Même sans entrer dans le détail de la configuration et en s'en tenant à l'installation par défaut, ceci est furieusement pratique. Mais qu'en est-il d'autres plateformes pouvant fonctionner sur le réseau ? Je pense en particulier aux modules ESP8266, s'utilisant exactement comme des Arduino, mais



*Les modules ESP8266 ont débuté leur carrière timidement avec ce qu'on appelle maintenant l'ESP-01 (en haut à gauche), initialement prévu pour être utilisé en complément d'une carte à microcontrôleur afin de fournir une connectivité Wifi. Avec le temps et un énorme effort principalement communautaire, ces modules existent à présent avec une multitude de formes, de prix, de tailles et de fonctionnalités intégrées.*

avec connectivité Wifi et faciles à acquérir pour quelques sous.

Nous avons précédemment utilisé ce type de plateformes dans le numéro 18 avec un montage indépendant, se connectant au point d'accès Wifi et permettant de démarrer ou éteindre (brutalement) un PC ou tout autre équipement électronique, en simulant simplement une pression sur le bouton de mise sous tension. Selon la configuration réseau utilisée, il est parfaitement possible que le montage ne se voit pas toujours attribuer la même adresse IP. L'usage de mDNS peut donc grandement faciliter les choses.

Or justement, les bibliothèques dédiées à la gestion du Wifi dans l'environnement Arduino pour cette plateforme (ESP8266, NodeMCU, Wemos, ESPduino, etc.) intègrent également une prise en charge du protocole mDNS. Il vous suffit pour cela d'inclure **ESP8266mDNS.h** dans votre croquis et de configurer quelques options. Le code présenté, en plusieurs morceaux, ci-après n'est pas complet, il ne s'agit que des éléments à ajouter dans votre propre croquis. Vous trouverez cependant un croquis complet d'un projet de serveur accessible par la commande **telnet** de votre Raspberry Pi (ou d'un PC sous GNU/Linux) dans le dépôt GitHub du numéro. Il s'agit d'une version simplifiée du croquis du projet présenté dans le numéro 18.

Pour utiliser mDNS avec l'ESP8266, on commence simplement par inclure les deux bibliothèques (« fichiers d'entête » en réalité) suivantes :

```
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
```

Assez classiquement, on déclare et initialise des variables nous permettant de stocker les paramètres importants :

```
// le nom du point d'accès à utiliser
const char* ssid = "mon_ssid";

// le mot de passe correspondant
const char* password = "phrase2passe";

// le nom d'hôte de notre ESP8266
const char* hostString = "ESPtelnet";
```

Comme avec n'importe quelle configuration Wifi cliente, on enchaîne sur la tentative de connexion dans la fonction **setup()**, juste après avoir défini le nom d'hôte qu'utilisera l'ESP8266 :

```
// définir le nom d'hôte de l'ESP8266
WiFi.hostname(hostString);

// établissement de la connexion Wifi cliente
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);

Serial.print("\nConnexion à ");
Serial.println(ssid);
while (WiFi.status() != WL_CONNECTED && i++ < 20) delay(500);
if(i == 21){
  Serial.print("Erreur de connexion sur ");
  Serial.println(ssid);
  while (1) {
    // erreur DHCP ou pas de réponse
  }
}

// Affichage adresse
printIPAddress();
```

Le mode choisi, **WIFI_STA**, fait de nous un client Wifi et non un point d'accès. Il est important de spécifier cela, car dans le cas contraire nous pourrions être client **et** point d'accès, et donc visible de n'importe quel client Wifi comme un smartphone. La connexion à proprement parler est déclenchée avec **begin()** et nous partons dans une boucle de 20 vérifications espacées d'une demi-seconde. Ceci a pour but de laisser la connexion s'établir, mais une fois le temps écoulé, si **status()** retourne autre chose que **WL_CONNECTED** informant d'une connexion correcte, nous estimons qu'il y a un problème. Dans ce cas, une boucle infinie **while()** bloque le croquis et tout s'arrête.

Dans le cas contraire, nous avons une connexion et avons reçu une adresse IP. Nous pouvons alors enchaîner sur la partie mDNS avec **MDNS.begin()** en passant en argument le nom d'hôte arbitrairement choisi. La valeur renvoyée par la méthode nous informe du bon déroulé de l'opération :

```
// démarrage mDNS-SD
if (!MDNS.begin(hostString)) {
  Serial.println("Erreur configuration mDNS!");
} else {
  Serial.println("Répondeur mDNS démarré");
  // ajout service disponible ici
  MDNS.addService("telnet", "tcp", 23); // Announce esp tcp service on port 23
}
```

Si tout va pour le mieux, nous utilisons **MDNS.addService()** pour préciser le type de service que nous fournissons. mDNS est la partie « résolution d'adresse », mais c'est mDNS-SD qui permet, en plus, d'informer de la liste des services que nous proposons. Ici, nous basant sur le projet de *Hackable n°18*, nous avons un serveur Telnet, sur le port 23, permettant d'accepter des commandes. Nous précisons donc en argument le nom du service, le protocole (TCP) et le port utilisé.

Notez que le nom d'hôte que nous utilisons ici est un choix, basé sur la fonction du montage utilisé. Le plus souvent, pour simplifier les choses, on compose ce nom d'une partie arbitraire et d'une autre, variable, découlant de l'identifiant 32 bits de la puce ESP8266 :

```
char hostString[16] = {0};

[...]

sprintf(hostString, "ESP_%06X", ESP.getChipId());
Serial.print("Hostname: ");
Serial.println(hostString);
```

Il est amusant de remarquer qu'on se retrouve là presque exactement dans la même situation que celle qui nous a poussés, au départ, à faire usage de mDNS. Nous n'avons plus d'adresse IP à retenir, mais devons finalement faire de même avec le nom d'hôte. Personnellement, je préfère donner un nom d'hôte totalement arbitraire en fonction du projet ou de son utilité, et ne pas me baser sur des identifiants numériques. À ma charge de ne pas utiliser deux fois le même nom d'hôte, mais je trouve cela bien plus explicite.

Quoi qu'il en soit, une fois mDNS et mDNS-SD configurés, il ne nous reste plus alors qu'à lancer le serveur pour terminer notre fonction `setup()` :

```
// Lancement serveur pour accepter les connexions
server.begin();
server.setNoDelay(true);
```

Bien entendu, ce serveur a été configuré en déclarant `WiFiServer server(23);` en début de croquis et la fonction `loop()` aura pour travail d'interpréter les communications sur ce dernier. Ceci n'est pas l'objet du présent article et reviendrais à honteusement paraphraser mon propre article du numéro 18. De plus, cette partie du croquis est totalement dépendante du projet. Ici c'est Telnet et le port 23, mais il peut tout aussi bien s'agir d'un serveur web et donc de HTTP et du port 80. N'oubliez simplement pas de configurer mDNS-SD en conséquence...

Une fois le croquis chargé et l'ESP8266 redémarré, automatiquement il se connectera au point d'accès Wifi (jusque-là rien d'exceptionnel), mais il lancera également son support mDNS, nous permettant ainsi de le trouver rapidement. Nous pouvons par exemple demander « qui propose Telnet ? » :

```
$ avahi-browse -t _telnet._tcp
+ eth0 IPv4 ESPtelnet     Telnet Remote Terminal local
```

De la même façon, à partir du nom d'hôte choisi et complété de `.local`, nous pouvons obtenir l'adresse IP du montage :

```
$ avahi-resolve -4 -n ESPtelnet.local
esptelnet.local 192.168.10.40
```

...et utiliser `ping` pour vérifier sa disponibilité :

```
$ ping -c 2 ESPtelnet.local
PING ESPtelnet.local (192.168.10.40) 56(84) bytes of data.
64 bytes from 192.168.10.40: icmp_seq=1 ttl=128 time=4.68 ms
64 bytes from 192.168.10.40: icmp_seq=2 ttl=128 time=102 ms

--- ESPtelnet.local ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 4.682/53.576/102.471/48.895 ms
```

...ou tout simplement nous connecter rapidement via `telnet ESPtelnet.local`. Plus besoin de se souvenir d'une adresse IP qui peut, en plus, potentiellement changer en cas de redémarrage, nos montages réseau/wifi sont bien plus facilement utilisables. Que du bonheur !

## CONCLUSION

mDNS est une solution très intéressante qui peut grandement vous simplifier la vie. Mais cette simplicité a un coût puisque cela expose directement toutes vos créations sur le réseau local. Certes celles-ci le sont en principe tout autant avec un serveur DHCP (ou même avec des adresses attribuées statiquement), mais il faut reconnaître que le mélange du nom d'hôte par défaut avec un nom d'utilisateur par défaut, un mot de passe par défaut et Avahi utilisé par défaut n'est pas vraiment une bonne idée. Rien d'étonnant donc que le serveur SSH ne soit pas activé par défaut avec Raspbian.

Personnellement, j'aurai cependant préféré une autre solution, adoptée sur d'autres ordinateurs monocartes : forcer la création d'un nouvel utilisateur avec son mot de passe dès le premier démarrage du système (ou du moins la mise en marche du script pour le faire, à chaque démarrage, tant que le problème est là). C'est certes plus contraignant pour l'utilisateur débutant, mais un minimum de sécurité n'a pas de prix. Et puis c'est bien plus intelligent que de permettre ce que bon nombre d'utilisateurs font finalement avec la Pi : ne rien changer et activer SSH en suivant un tutoriel ou un commentaire sur un site web, persuadé que cela ne changera rien d'important.

Personnellement, jusqu'à présent mon réflexe lors du premier démarrage d'une Pi, ou d'une installation toute fraîche d'un GNU/Linux sur PC, se résumait à `sudo apt-get purge avahi-daemon`, arguant mentalement (et stupidement) du fait que « c'était pas là avant, donc ça sert à rien ». C'est en réalité le support mDNS sur ESP8266 qui m'a fait changer d'avis. La moralité de l'histoire sera donc de ne jamais rien écarter définitivement sur la base de ses habitudes et de ses besoins évalués à un instant donné. À chaque problème sa solution, et le problème d'un moment est parfois la solution d'un autre... DB

*Un groupe de capteurs de température assemblés à partir d'un adaptateur USB, une sonde DS18B20 et un module ESP2866 (ESP-01 ici), comme ceux dont nous avions parlé dans le numéro 8 de Hackable, peut bénéficier grandement des fonctionnalités offertes par mDNS. Inutile alors pour le système collectant les données de connaître la liste complète des capteurs, il lui suffit de se baser sur un préfixe utilisé pour les noms d'hôtes et éventuellement un service associé, scanner le réseau et le tour est joué !*

# UTILISEZ UNE ALIMENTATION SANS FIL POUR VOS PROJETS

*Denis Bodor*

**Ah la transmission d'énergie sans fil, toute une aventure, toute une mythologie, tout un labyrinthe où certains se perdent parfois en divagations… jusqu'à l'arrivée d'un standard de fait : le Qi. Recharger son smartphone en le posant simplement sur un présentoir devient petit à petit quelque chose de tout à fait normal. Mais, voyez-vous, quand une telle technologie se popularise et que les prix s'effondrent, je ne pense pas à ce que les constructeurs veulent que je pense. Non, je me demande immédiatement : « mais comment utiliser cela dans mes projets ? ». Et parfois, comme ici, ça marche plutôt bien… selon ce qu'on demande.**

Surtout, ne cherchez pas « transmission d'énergie sans fil » sur le Web, vous risquez de tomber tout autant sur des informations très sérieuses que sur des choses qui frôlent le mysticisme le plus étrange. Car, voyez-vous, là où technologie et légende se mélangent il y a souvent un nom qui revient plus fréquemment que les autres : Nikola Tesla. Que voulez-vous, quand vous êtes un brillant inventeur, avec un certain goût pour la mise en scène et des ambitions à la hauteur de votre savoir et de votre imagination, les gens ont tendance, bien longtemps après votre trépas, à vous prêter des capacités et des actes que vous-même n'auriez pas même osé sous-entendre.

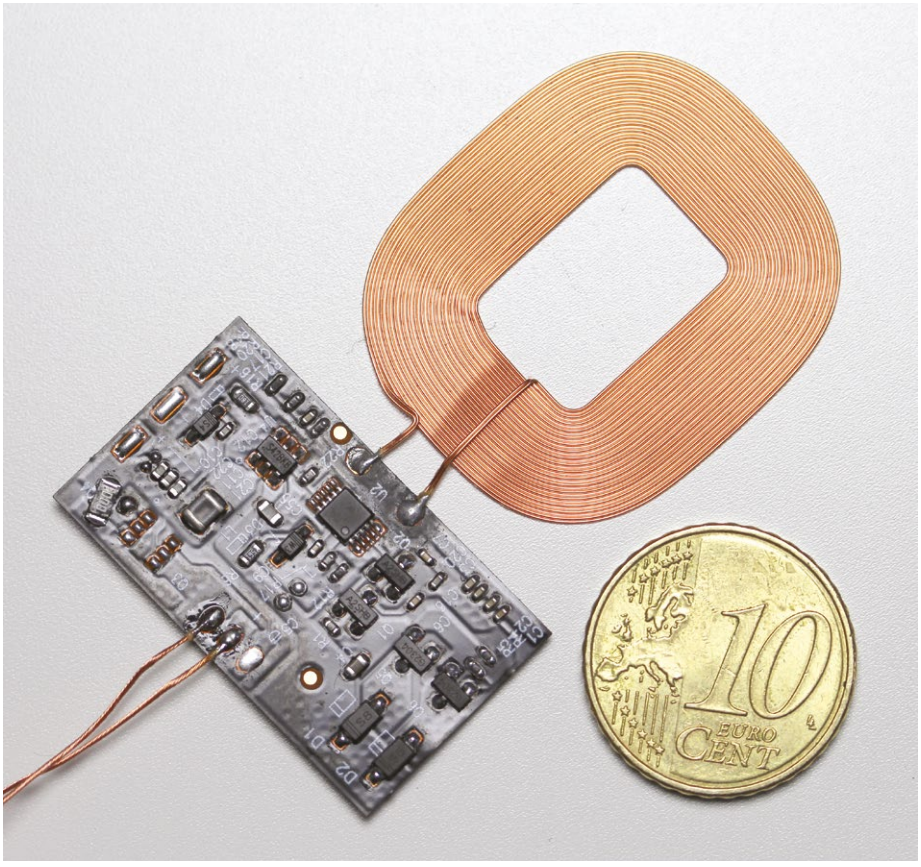
Mais les faits (et les brevets) sont là, ce cher Nikola Tesla a effectivement fait grandement progresser bien des domaines grâce à ses travaux et en particulier ceux sur la transmission d'énergie sans fil. Le principe en lui-même est fort simple :

- faites passer un courant dans un conducteur et un champ magnétique se forme ;
- coupez l'alimentation et le champ magnétique s'effondre ;
- répétez l'opération et placez un autre conducteur à proximité ;
- par induction, un courant apparaît sans ce second conducteur ;
- vous venez de transmettre de l'énergie sans utiliser de support.


*Les récepteurs Qi à intégrer dans un smartphone (comme ici pour un Samsung S4) se glissent juste sous la coque arrière de l'appareil. À droite, on peut voir que ceci se résume à peu de choses : une bobine et un circuit relativement simple.*

Cette base a donné naissance à la communication radio, des tous premiers postes de transmission jusqu'au Wifi, LoRa, Bluetooth, TNT, et j'en passe. Mais contrairement à ce type de procédés, visant à transmettre un signal qui devra être amplifié pour être utilisable, il existe une autre approche dont le but est d'alimenter un dispositif par induction.

Là encore, sur le papier, le principe est des plus évidents. En faisant circuler un courant alternatif dans une bobine, un champ électromagnétique est créé. Une bobine à proximité, dans ce champ magnétique, va alors voir apparaître dans son conducteur un courant alternatif : c'est un effet de l'induction électromagnétique. Il est donc possible, sans support, de transmettre de l'énergie d'une bobine à une autre. Ce phénomène a été découvert il y a fort longtemps et est utilisé, lui aussi, un peu




*Étant donné la structure du circuit et la présence de points de soudure « + » et « - » un peu partout, il est fort probable que, quel que soit le modèle de smartphone cible, ce soit effectivement le même circuit (et probablement la même bobine) qui est utilisé pour tous les récepteurs. On voit ici deux points de connexion utilisés et, sur la droite, trois autres (deux « + » et un « - »). Il y en a autant à l'arrière du circuit...*

partout, des transformateurs aux moteurs en passant par les tags RFID et ces nouveaux fameux systèmes permettant de recharger son smartphone en le posant simplement sur un support.

Dans la pratique, les choses ne sont cependant pas si simples. Bien qu'il soit possible de très facilement obtenir un résultat avec deux bobines bricolées sur un coin de table, il faut faire la différence entre « transmettre de l'énergie » et « transmettre de l'énergie efficacement ». Tout un tas de phénomènes entrent en ligne de compte dans ce domaine et en n'en maîtrisant pas pleinement les effets, un tel montage dissipera très facilement la plupart de l'énergie fournie en chaleur ou en émissions inutiles. Si près de 100 ans se sont écoulés entre la construction de la tour de Wardenclyffe par Nikola Tesla pour ses expérimentations et l'arrivée des chargeurs de smartphone, ce n'est pas par hasard.

Contrairement à la transmission de signaux radio, et bien que les technologies soient très proches (et se recoupent), le critère le plus important dans la transmission d'énergie sans fil est le rendement ou l'efficacité du système. Le but est d'avoir le moins de perte possible et donc de faire en sorte qu'un maximum de courant envoyé dans l'émetteur soit induit dans le récepteur. Dans le cas des chargeurs de smartphone, il faut, de plus, que l'encombrement soit réduit au minimum, car personne ne veut d'une énorme bobine alourdissant et doublant la taille de son mobile dernier cri.

## 1. LA TECHNOLOGIE QI

Avant toutes choses et pour ne pas passer pour un abruti en société, il faut savoir que « Qi » se prononce « tchi » du mandarin/chinois signifiant approximativement « énergie de la vie et de l'univers » (il n'y a pas vraiment de

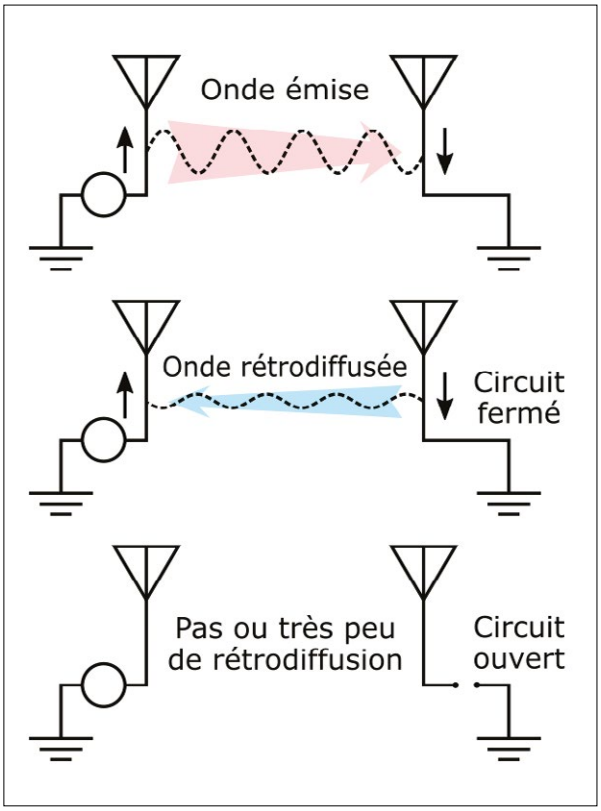
traduction ou d'équivalence avec un concept occidental si ce n'est peut-être « la force » de tonton Georges). Ceci devrait merveilleusement compléter la remise en place d'un interlocuteur vous ayant déjà égratigné le tympan d'un « **la** wifi » en quémandant le mot de passe associé parce que, voyez-vous, il est « trop trop geek » !

Quoi qu'il en soit, Qi est une technologie créée il y a une dizaine d'années par le *Wireless Power Consortium* (ou WPC) sous la forme d'une liste de spécifications techniques. Publiées et diffusées gratuitement en 2009, ces spécifications forment une norme sous l'intitulé « *Qi low power specification* » (« norme de faible puissance Qi »). Ceci, et le fait que le consortium regroupe quelques 220 sociétés parmi lesquelles Samsung, Nokia, Qualcomm, Apple, LG, HTC, IKEA (oui, oui), Texas Instruments, ASUS, Panasonic ou Sony, a permis de populariser Qi au point qu'il s'agit à présent d'un standard pleinement adopté (même Apple ne fait pas bande à part avec son dernier iPhone 8, c'est dire). Notez tout de même au passage que, bien que les spécifications soient disponibles, une partie des implémentations sont brevetées et que rejoindre le consortium consiste en une adhésion payante. Il ne s'agit donc pas vraiment d'un standard ouvert.

La norme décrit une architecture générale ainsi que des points très précis devant être respectés. Il n'est déjà pas aisé d'arriver à un rendement satisfaisant étant donné les contraintes techniques, mais les smartphones ont, de plus, des besoins spécifiques. Hors de question donc de faire dans l'à-peu-près et de risquer que l'utilisateur ne transforme involontairement son périphérique en chauffage d'appoint ou que le chargeur fasse office d'émetteur radio.

Un « chargeur » Qi ne se résume donc pas à une bobine, mais intègre toute une partie électronique de contrôle et de commande permettant d'ajuster la puissance délivrée en fonction de ce que le smartphone demande. Cette communication est unidirectionnelle et fonctionne sur le principe de rétrodiffusion (ou *backscatter* en anglais).

Lorsqu'une onde est transmise par un émetteur, celle-ci induit un courant dans le récepteur, mais une partie de l'énergie est « réfléchie » en fonction de la charge électrique du récepteur. La figure ci-dessous montre comment le récepteur peut utiliser ce phénomène pour moduler un signal de façon à transmettre des données à l'émetteur. C'est précisément ce qui se passe avec les chargeur Qi.




*Principe de fonctionnement général de la rétrodiffusion. En haut, un émetteur envoie une onde induisant un courant dans l'antenne/bobine du récepteur. Avec une charge électrique importante, comme un court circuit dans la seconde illustration, une onde rétrodiffusée se propage en sens inverse et peut être détectée par l'émetteur. Dans la dernière illustration, le circuit est ouvert et très peu ou pas de rétrodiffusion apparaît. De ce fait, en jouant sur la charge électrique, le récepteur peut moduler un signal à destination de l'émetteur et donc transmettre des informations.*

Ce mode de fonctionnement n'a rien de nouveau et est également utilisé par les tags RFID, par exemple. Là, cependant, la communication est bidirectionnelle puisqu'en plus d'alimenter le tag, l'émetteur peut envoyer des données en modulant le signal. Le flux inverse, par contre, repose également sur le phénomène de rétrodiffusion contrôlée.

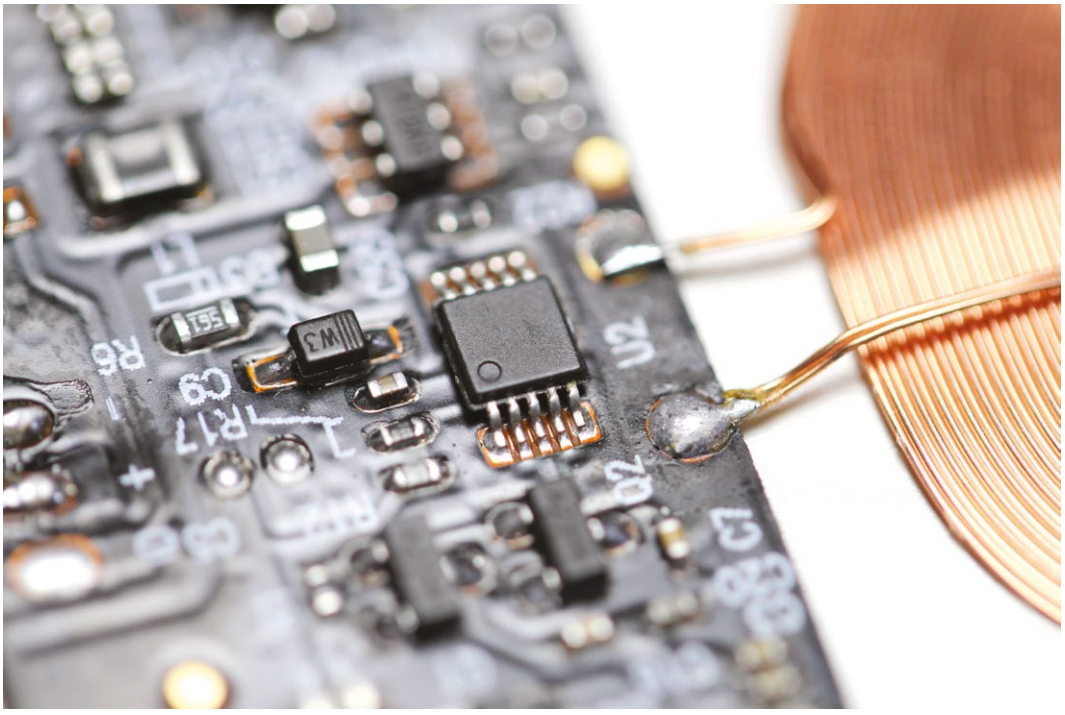
En technologie Qi, c'est ainsi le récepteur qui détermine le courant qu'il souhaite recevoir et transmet cette information à l'émetteur. Cette communication utilise un signal modulé en amplitude (ASK) pour transmettre des séries de bits formant des paquets structurés. Ceux-ci sont composés d'un préambule, d'un entête, d'un message et d'une somme de contrôle, selon un format décrit dans les spécifications Qi.

Si vous voulez en savoir plus, une très intéressante note applicative de NXP/Freescale décrit le fonctionnement en détail sur une vingtaine de pages (document référence AN4701). Cependant, en ce qui nous concerne, le point important est ici de comprendre qu'il ne suffit pas de créer une bobine pour alimenter ses projets Arduino (ou Raspberry Pi) via un chargeur Qi. Ceci ne fonctionnera tout simplement pas, puisque dans le protocole de communication se trouve toute une phase de « négociation » où le récepteur doit fixer l'intensité du courant dont il a besoin.

Vous comprendrez aisément, grâce à toutes ces explications, que créer un récepteur Qi n'est pas simple et très certainement pas une partie de plaisir. Ce qu'on peut faire, en revanche, c'est utiliser quelque chose d'existant et donc profiter de la popularité de Qi, aussi bien pour la disponibilité du matériel que pour le coût réduit de mise en œuvre...

## 2. BRICOLONS QI POUR NOS PROJETS

Tous les smartphones ne sont pas compatibles Qi par défaut. En effet, seul un nombre réduit de modèles récents disposent de cette fonctionnalité. En revanche, un certain nombre d'autres

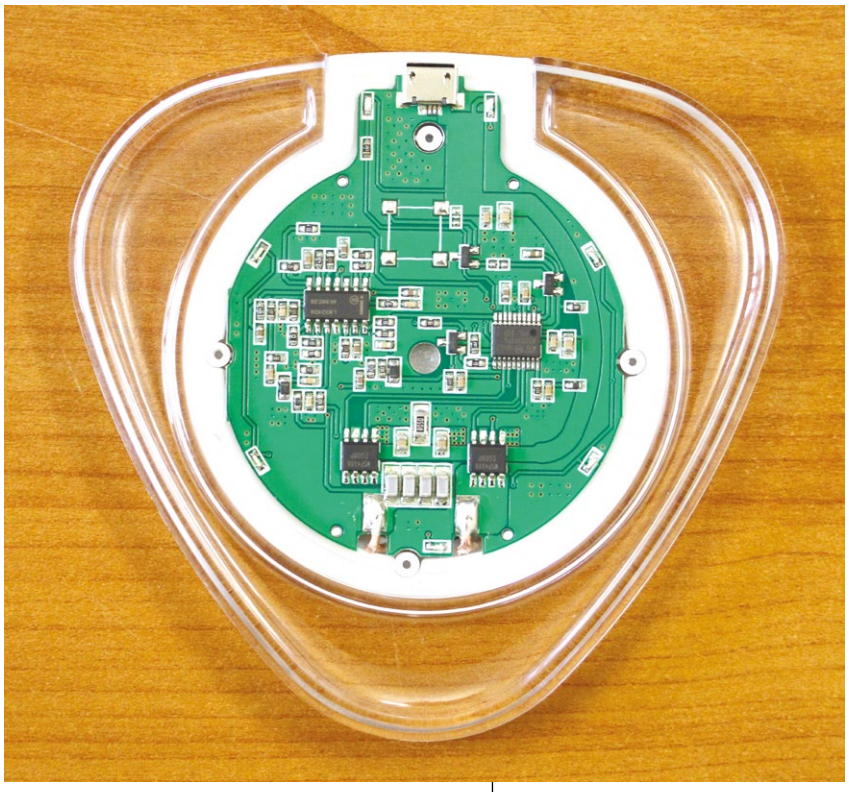



*Le seul circuit intégré présent est ce mystérieux composant à l'emplacement U2, sans le moindre marquage permettant de l'identifier. Il peut tout aussi bien s'agir d'un circuit spécialisé Qi, un clone contrefait ou, tout simplement, un microcontrôleur implémentant le protocole décrit dans les spécifications Qi.*

modèles peuvent se voir équipés d'une coque optionnelle intégrant une antenne/bobine et/ou un circuit permettant la gestion du protocole Qi. En cherchant un peu sur le Web, on constate que bon nombre de ces coques ne sont en réalité qu'un bout de plastique doublé d'un « récepteur Qi » fournissant simplement une tension d'alimentation standard.

Ces modules de réception sont également distribués séparément pour un usage sensiblement différent : ils s'intègrent entre l'appareil et la coque existante et, là encore, ils proposent deux simples connecteurs d'alimentation. Enfin, certains fabricants d'accessoires se sont lancés dans une brèche laissée vacante par les marques, celle des smartphones ne disposant ni d'une option de changement de coque ni d'un connecteur dédié pour y brancher un module récepteur. La connexion se fait alors, tout simplement, via le connecteur micro-USB standard du smartphone.

Voilà exactement le genre de choses qui met la puce à l'oreille, car s'il est possible de modifier un smartphone pour lui ajouter un récepteur ayant un connecteur micro USB mâle, il doit être possible d'alimenter n'importe quel périphérique disposant d'un connecteur micro USB femelle, et donc une carte Arduino, ESP8266 ou encore Ti Launchpad. Dans l'absolu, il est tout simplement possible d'alimenter n'importe quel périphérique USB de la sorte, c'est-à-dire n'importe quel projet acceptant +5V +/-5%, soit entre 4,75V et 5,25V (selon les spécifications USB 1.1 et 2.0).



*Un peu plus de composants côté chargeur Qi et pour cause, il faut piloter toute la partie électronique de puissance permettant d'émettre. En plus du contrôleur Qi, on reconnaît immédiatement deux doubles MOSFET directement reliés à la bobine/antenne en spirale située à l'arrière du circuit imprimé, ainsi qu'un quadruple amplificateur opérationnel (opamp). L'architecture du circuit dans son ensemble est très similaire au kit d'évaluation STEVAL-ISB027V1 de chez STMicroelectronics... mais 20 fois moins cher.*

Côté chargeur, le standard étant... un standard, n'importe quel périphérique compatible Qi pourra, en toute logique, alimenter n'importe quel appareil Qi. Bien entendu, on retrouve alors aussi bien des périphériques de marque avoisinant les 75€ et les super entrées de gamme à 3€ sur eBay en provenance directe de Shenzhen. Une petite recherche sur le Web et vous voici devant une pléthore d'options. Personnellement, entre les marques qui gonflent les prix et les offres super économiques avec des composants parfois douteux, je vise généralement l'entre-deux avec une petite préférence pour les produits dont la description est la plus complète et juste (et éventuellement la proximité du vendeur si je suis pressé).

Au moment où j'écris cet article, deux modèles ont été commandés : le premier à 14€ de marque SOAIY en provenance de Birmingham (Royaume-Uni) et un second, de marque FANTASY, en plusieurs exemplaires, car à seulement 3€, livré (un jour)

depuis Shenzhen (Chine). À cette date, seul le matériel venant d'Angleterre est arrivé, mais des tests complémentaires ont également été faits avec un chargeur Samsung d'une collègue, modèle EP-NG930 (le chargeur, pas la collègue).

Notez que ces équipements sont généralement fournis sans bloc d'alimentation puisqu'en principe vous êtes censé utiliser celui de votre smartphone. Tous les modèles que j'ai eu le loisir d'avoir en main ou pour lesquels les spécifications détaillées étaient accessibles utilisaient un connecteur micro USB type B (comme la Raspberry Pi par exemple) et nécessitaient une alimentation de 2A en 5V. Certains modèles proposent tantôt, en plus, un connecteur d'alimentation en 9V, mais ceci reste très rare.

Le « chargeur » n'est donc pas un problème et se trouvera sans la moindre difficulté. Il n'en va pas forcément de même pour les récepteurs qui eux ne se trouveront que sur les sites comme Ebay, Banggood, DealExtreme, Alibaba, etc. Là, mieux vaut être prudent et ne pas forcément miser sur l'aspect « local » car, bien des fois, les vendeurs compatriotes ne sont pas nécessairement les plus sérieux. Dans ce cas précis et dans mon empressement, j'ai donc commandé à la fois auprès d'un vendeur de Gateshead (Royaume-Uni) des récepteurs pour Samsung S4 à ~3€ pièces et des modèles génériques micro USB à trois fois ce prix, auprès d'un vendeur bien de chez nous (« neodis_com »), espérant les recevoir en premier. J'ai finalement reçu en quelques jours les récepteurs d'outre-Manche et le vendeur français... a supprimé l'annonce et son compte peu après la vente.

Bien entendu, un litige a été ouvert sur eBay et, du fait de sa démarche, mon paiement m'a été automatiquement remboursé en quelques jours. Le fait est cependant que ce n'est pas la première fois que ce genre de choses m'arrivent et c'est systématiquement avec des vendeurs français, italiens, espagnols... mais jamais avec des transactions impliquant des vendeurs situés en Asie, en Angleterre, en Allemagne ou en Pologne. Sur ma base statistique de quelques 700 transactions s'étalant sur bien des années et bien qu'essayant de ne pas tirer de conclusion hâtives, je suis maintenant résolu à me montrer d'une méfiance et d'une prudence sélective des plus poussées (en clair, pour les vendeurs de certains pays, qui n'ont pas 100% d'évaluations positives et un minimum de 1000 évaluations, je passe mon tour). C'est triste, mais les statistiques et les probabilités ne pardonnent pas.

En raison de cette mésaventure donc, nous nous pencherons uniquement sur les récepteurs non génériques et non USB. Ceci n'est pas très grave puisqu'il n'y aurait de toute façon pas grand-chose à dire, les manipulations se limitant à une simple connexion. Il est bien plus intéressant de regarder de plus près quelque chose de moins facile à utiliser.

Le récepteur se présente comme un rectangle de 43mm x 60mm muni d'une excroissance de 20mm disposant de deux points de connexion non libellés. L'ensemble fait moins d'un millimètre d'épaisseur et donne la perception d'une qualité assez moyenne. Pris en sandwich entre deux films plastiques autocollants qui peinent à rester ensemble, se trouve une bobine plate rectangulaire et un circuit imprimé souple regroupant quelques transistors, composants passifs et un circuit intégré à 10 broches sans marquage (sans le moindre doute un composant dédié au Qi).

Deux approches sont possibles avec ce genre de produits :

- placer le récepteur sur un support de charge et utiliser un multimètre pour déterminer la polarité en mesurant simplement la tension en sortie ;
- ou, tout simplement, dépecer l'objet.

Cette seconde solution est certainement celle à préférer, car elle donne l'opportunité de découvrir l'intérieur du récepteur et de constater qu'il existe plus d'un point de connexion (quatre paires +/- en tout sur le modèle

testé). On constate aussi que la contrainte de finesse du récepteur a poussé le fabricant à utiliser du fil de cuivre émaillé de faible diamètre, ce qui n'est pas très rassurant étant donné que le récepteur devrait être en mesure de fournir quelques 800 mA en 5V (4 watts).

Il faut se montrer très délicat durant l'opération de démontage. Un premier film plastique est relativement facile à retirer, découvrant la bobine et le circuit interne. Cependant, la bobine est placée sur un support en mousse adhésive, lui-même collé à l'autre film. Toute la difficulté réside donc dans le fait de décoller la bobine sans la déformer, mais avec un peu de délicatesse cela est tout à fait possible.

## 3. EST-CE UNE BONNE SOLUTION POUR VOS PROJETS ?

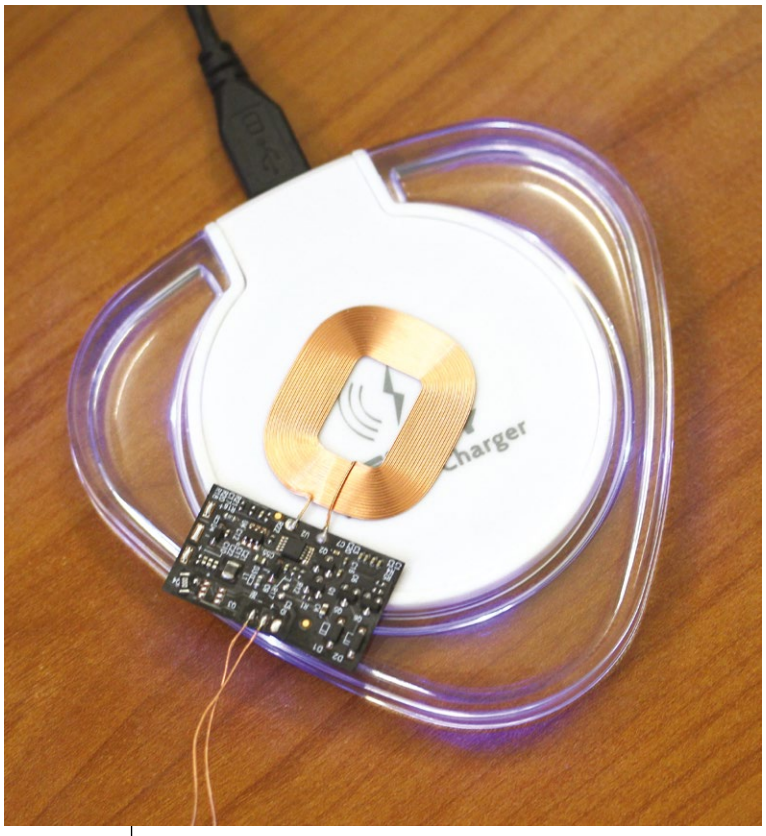
Annonçons-le sans détour, bien que l'absence de connexion/déconnexion soit très intéressante, l'efficacité de ce type de dispositif est loin d'être satisfaisante. En regroupant les informations fournies par d'autres utilisateurs/bidouilleurs sur le Web avec mes propres mesures, une efficacité moyenne d'environ 60% à 70% est le mieux qu'on puisse espérer. En d'autres termes, pour fournir 300 mA à un montage maison, le chargeur utilisera quelques 500 mA. Rien d'étonnant donc au fait que ces chargeurs demandent généralement un bloc d'alimentation capable de fournir 2A en 5V.


*On peut clairement le constater dès lors qu'on mesure le courant entrant dans le chargeur et celui fourni par le récepteur, l'efficacité de ce type de solution n'est pas très impressionnante. La Raspberry Pi, connectée ici au récepteur, utilise 270mA, mais le chargeur, lui, en « pompe » 450 du bloc d'alimentation. Nous avons donc presque un watt qui disparaît dans la nature (littéralement, puisque cette perte est dissipée en chaleur).*

Pour rappel, une carte Raspberry Pi, selon le modèle utilisé, est susceptible de consommer jusqu'à 800 mA (sans aucun périphérique USB connecté), ce qui peut nous amener à la limite maximale des chargeurs et récepteurs Qi :

Pi inactive :

- Raspberry Pi 2 B : 420mA ;
- Raspberry Pi B+ : 230 à 240mA ;
- Raspberry Pi B : 320 à 330mA ;
- Raspberry Pi A+ : 180 à 240mA ;
- Raspberry Pi A : 120 à 140mA ;
- Raspberry Pi Zero : 60 à 70mA ;
- Raspberry Pi 3 B : 280 à 320mA.



*Comme on peut s'y attendre, la version « épluchée » sur récepteur Qi fonctionne parfaitement une fois placée sur le chargeur/émetteur. On voit ici que ce dernier s'illumine en bleu indiquant une « charge ». En l'absence de récepteur, il prend une couleur rouge et lorsque la charge est terminée, il s'éclaire en vert.*

Pi en pleine charge processeur :

- Raspberry Pi 2 B : 450 à 650mA ;
- Rasbperry Pi B+ : 300 à 600mA ;
- Raspberry Pi B : 380 à 450mA ;
- Raspberry Pi A+ : 200 à 300mA ;
- Raspberry Pi A : 170 à 300mA ;
- Raspberry Pi Zero : 120 à 150mA ;
- Raspberry Pi 3 B : 500 à 800mA.

Il est également important de remarquer que le standard Qi est conçu pour la charge de batterie et non plus une alimentation directe. Ceci implique qu'il peut y avoir des micro-coupures susceptibles de provoquer le redémarrage de la carte que vous choisirez d'alimenter. C'est donc généralement une très bonne idée que de placer un ou plusieurs condensateurs d'une capacité suffisante (1mF ou plus) en parallèle à la sortie, exactement comme on le ferait pour construire une alimentation.

À quoi bon alors peut servir ce type de bricolage ? La réponse est toute simple : à exactement la même chose que ce pour quoi la technologie Qi est prévue à l'origine, charger des batteries.

Bien qu'il soit parfaitement possible d'imaginer un montage à base d'Arduino, d'ESP8266 ou de Raspberry Pi Zero se mettant en route dès qu'on le place sur le chargeur, il est bien plus légitime d'y associer une batterie ou un accumulateur et ainsi rendre le chargement plus facile. Dans cette optique, je vois trois solutions utilisables :

- Opter pour une plateforme/carte disposant déjà d'un contrôleur de charge et d'un connecteur pour accumulateur LiPo/LiIon/LiFePo (au format JST par exemple). Il s'agit généralement de cartes à base d'ESP8266 ou d'Atmel AVR, compatibles avec l'environnement Arduino. Il suffit alors de connecter le récepteur Qi en parallèle à l'alimentation USB ou d'adapter un connecteur USB de récupération.
- Utiliser une batterie d'appoint pour smartphone comme celle que nous avions étudiée dans le numéro 15. Là encore, en modifiant l'appareil ou en adaptant un connecteur d'alimentation, il devient possible de charger la batterie en la posant simplement, comme un smartphone, sur le chargeur.
- Mettre en œuvre un circuit ou une carte spécialisée dans la gestion de charge, comme par exemple le LiPoRider (~20€) que j'avais présenté dans *Hackable n°10*. Ce type de circuit accompagné d'un accu LiPo fonctionne exactement comme la batterie citée précédemment en permettant une charge via USB, la connexion d'un accumulateur de taille et capacité de son choix, mais également la connexion d'un panneau solaire. Une alternative plus économique peut être la combinaison d'un contrôleur de charge LiPo

TP4056 (~1,5€), d'un accu de qualité type 18650 de 2500mAh (~9€) et d'un convertisseur Boost ou *Step-Up boost converter* en anglais (~1€), transformant les 3,7V-2,5V de l'accu en 5V.

Dans les trois cas, tout repose sur l'assemblage final du projet et donc dans un certain savoir-faire manuel pour arriver à un résultat acceptable et présentable. Une chose importante à ne pas perdre de vue est la contrainte liée au couplage inductif : il s'agit d'antennes. Ceci signifie que la proximité d'une surface conductrice peut perturber ou stopper le transfert d'énergie, en particulier dans le cas du fer. Cet élément parasite n'a pas besoin de se trouver entre le chargeur et le récepteur, le simple fait d'être dans la portée du champ magnétique est suffisant. Placez/collez votre récepteur directement sur le boîtier métallique de votre projet, par exemple, et la charge a toutes les chances de ne pas se faire. Le problème est exactement le même que celui rencontré par certains utilisateurs de smartphones équipés d'une coque partiellement métallique, sans doute très jolie, mais peu compatible Qi.

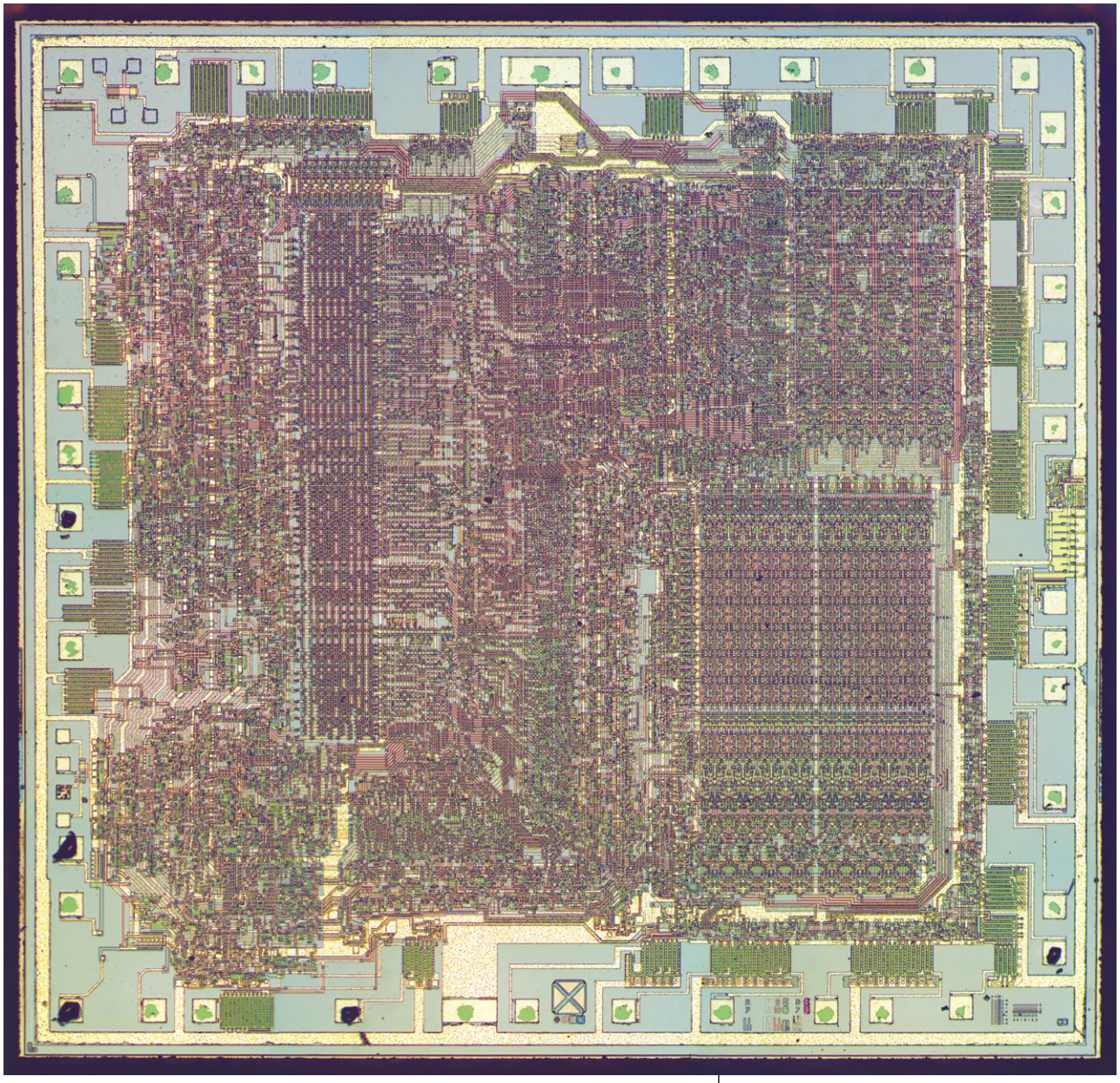
En conclusion, je pense qu'il est juste de dire que l'utilité de Qi pour des montages à base de microcontrôleurs ou d'ordinateurs mono-carte se limite à des cas bien spécifiques, impliquant généralement déjà un fonctionnement autonome via un accumulateur. En dehors de ces cas bien bornés, où effectivement Qi apport un vrai avantage, et même si la réalisation est assez économique, alimenter un projet ainsi n'a pas vraiment de sens.

Ce qui est possible, en revanche, c'est de tout simplement trouver des appareils normalement chargés via une connexion filaire et de les convertir en Qi (console portable, batterie d'appoint, GPS de randonnée, éclairage rechargeable, enceintes Bluetooth pour smartphone, cigarette électronique, etc.). Ceci peut être, de plus, un exercice fort intéressant et très instructif.... **DB**

# CRÉEZ VOTRE ORDINATEUR 8 BITS SUR PLATINE À ESSAIS : LA MÉMOIRE

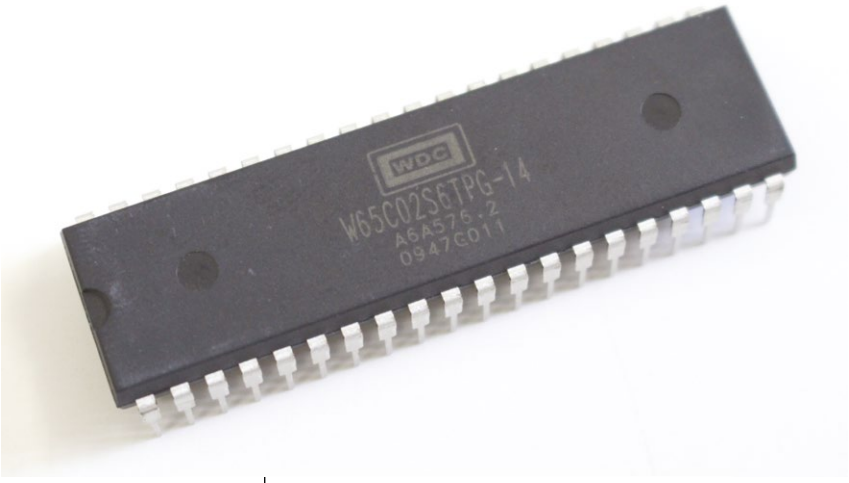
*Denis Bodor*

**Nous avons vu dans l'article précédent sur le sujet qu'il n'est pas très difficile, mais pourtant furieusement intéressant, d'animer un processeur 8 bits à l'aide de simples résistances, quelques leds et une carte Arduino. Il est temps à présent de pousser l'expérience un peu plus loin et donc d'en apprendre davantage sur le fonctionnement d'une technologie que nous utilisons quotidiennement sans vraiment nous en rendre compte. Faisons donc un pas de plus dans la construction de notre ordinateur 8 bits en fournissant au processeur Z80 ce qu'il pensera être de la mémoire.**



*Le processeur Zilog Z80 a été conçu comme compatible avec l'architecture Intel 8080, mais en ajoutant un lot important d'améliorations. Celui-ci a vu le jour en juillet 1976 et regroupe 8500 transistors sur une puce de 18mm² avec une gravure de 4000 nm (milliardièmes de mètres). À titre de comparaison, un AMD Ryzen 8 cœurs compte 4,8 milliards de transistors sur une surface de 192mm² avec une finesse de gravure de 14nm...*

Contrairement au microcontrôleur Atmel ATmega328P d'une carte comme l'Arduino UNO, un processeur ne dispose pas de mémoire, ni de périphériques comme un port série, un bus SPI ou même simplement de broches dont l'état peut être arbitrairement contrôlable (GPIO). Le processeur n'est là que pour exécuter des instructions et communiquer avec d'autres éléments via des bus. Un ensemble processeur + mémoire + périphériques, lorsqu'il est regroupé dans un seul circuit intégré, est appelé microcontrôleur. Lorsque ces éléments sont distincts en revanche, le processeur seul ne peut rien faire. Il lui faut au minimum un signal d'horloge




*Le Z80 n'est pas le seul microprocesseur à pouvoir être utilisé pour construire un ordinateur. Le 6502 peut également être utilisé, mais il convient de bien se documenter sur le sujet en raison de certaines limitations techniques importantes. Le WDC W65C02S présenté ici est une version CMOS du 6502 pouvant être, entre autres choses, cadencé arbitrairement comme le Z80. Ce n'est pas le cas, par exemple, de modèles NMOS plus anciens.*

pour être cadencé et des instructions à exécuter, et celles-ci se trouvent bien entendu dans une mémoire.

L'étape logique qui nous occupe ici, et qui consiste donc à fournir de la mémoire au processeur, passe généralement par l'utilisation d'un composant programmable comme une mémoire morte ou en anglais *Read-Only Memory* ou « ROM ». Ce terme générique couvre souvent un ensemble de composants de types différents :

- Les vrais ROM dont le contenu est défini lors de leur fabrication et qui est non inaltérable.
- Les PROM pour *Programmable Read-Only Memory* qui sont des composants vendus vierges, mais programmables par l'utilisateur **une seule fois**. On parle également tantôt de PROM ou d'EPROM OTP pour *One-Time Programmable* (programmable une fois).
- Les EPROM pour *Erasable Programmable Read-Only Memory* qui fonctionnent comme les PROM, mais peuvent toutefois être effacées pour être ensuite reprogrammées. Notez bien qu'il ne peut s'agir d'un effacement partiel, l'ensemble de la mémoire est effacé en une fois. Lorsque l'effacement se fait par exposition du composant à un rayonnement ultraviolet via une petite « fenêtre » transparente en quartz, on parle d'UVPROM. Certains composants de ce type peuvent être des EPROM OTP dans le sens où il n'est pas possible de les exposer aux UV car, bien que la technologie soit la même, ils ne disposent pas de « fenêtre » transparente pour les effacer.
- Les EEPROM forment le type actuellement le plus utilisé, car ils dispensent l'utilisateur de la phase d'exposition aux UV par exemple. La mémoire peut être effacée et réécrite électriquement, d'où le nom de *Electrically Erasable Programmable Read-Only Memory*.
- Les NVRAM pour *Non-Volatile Random-Access Memory* ou mémoires vives non volatiles. Il s'agit là d'un terme générique, car la notion de « mémoire vive » souvent associée à l'acronyme « RAM » sous-entend une mémoire qui ne garde pas son contenu sans alimentation (une mémoire volatile donc), mais également la possibilité d'un accès direct/aléatoire aux données par opposition à un accès séquentiel. Une EEPROM dont le contenu est accessible de cette façon, et non via un bus SPI ou i2c par exemple, est donc une NVRAM. Pour ajouter à la confusion, il n'est pas rare que des composants à accès séquentiel embarqués dans des PC portables par exemple, soient tantôt aussi appelés NVRAM, alors qu'il s'agit en réalité d'EEPROM i2c.

Traditionnellement, lorsqu'on construit un ordinateur « maison », on s'oriente vers des UVPROM,

en particulier pour le côté vintage de ces composants. On peut également utiliser des EEPROM qui fonctionneront de la même manière mais, dans les deux cas, ceci implique l'achat ou la construction d'un programmeur, un appareil permettant d'effacer la mémoire et de la programmer à nouveau. Le budget n'est pas forcément important, mais il faut également prendre en considération l'aspect pratique. En effet, cette mémoire est destinée à contenir les instructions et les données que le processeur devra utiliser. Ceci signifie donc qu'en mettant au point un programme, on doit pouvoir retirer le composant du projet pour le mettre dans le programmeur (via un support ZIF le plus souvent). De plus, le cycle « retrait, effacement, programmation, insertion, test » devient vite pénible, en particulier lorsqu'on commence à peine à assimiler les instructions utilisables par le processeur, leurs significations et leurs usages.

À des fins pédagogiques il y a donc bien mieux...

## 1. UNE CARTE ARDUINO EN GUISE DE MÉMOIRE

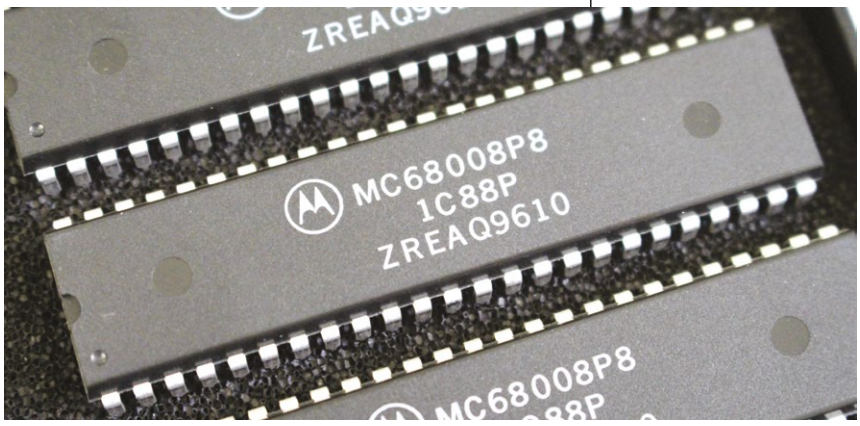
Une autre approche possible, qui était jusqu'à il y a quelques années inimaginable, consiste à ne pas utiliser une véritable mémoire (UVPROM, EEPROM ou autre), mais à simuler ce composant à l'aide d'un microcontrôleur et en particulier celui d'une carte Arduino. L'objectif est alors de connecter le processeur, ici le Z80, à la carte pour lui faire croire qu'il a affaire à une vraie mémoire.

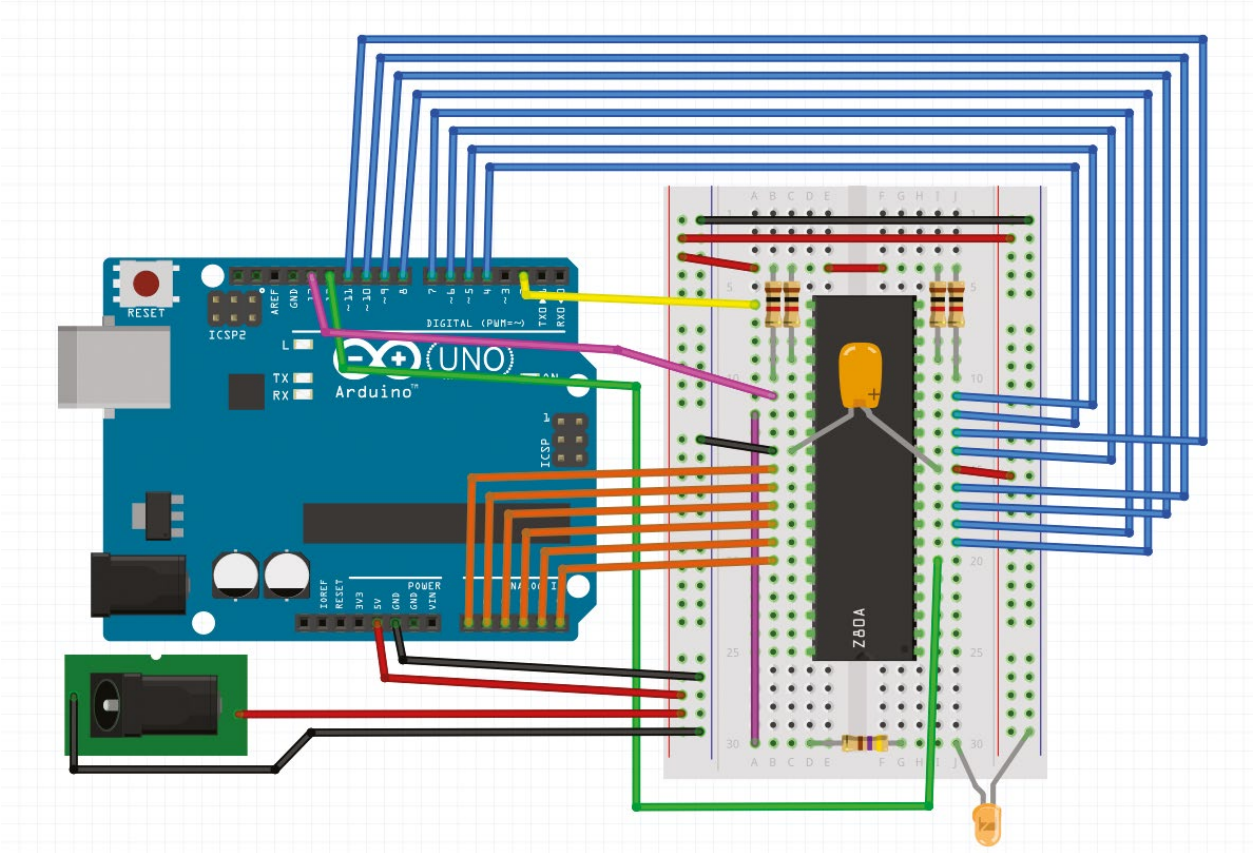
En théorie, l'opération est relativement simple puisqu'il nous suffit, côté Arduino, de déterminer l'adresse voulant être lue par le processeur en observant l'état des 16 broches du bus d'adresse et de fournir en retour une valeur sur 8 bits via le bus de données. Il y a juste un petit problème... Les broches d'une carte Arduino utilisées pour cette simulation de mémoire seraient :

- 16 bits/broche pour le bus d'adresse ;
- 8 bits/broche pour le bus de données ;
- /RD pour demander à la mémoire de présenter les données sur le bus. Normalement connecté à la broche /OE (*Output Enable*) de la mémoire qui déclenche alors le changement d'état sur le bus de données ;
- /MREQ pour *Memory REQuest* indique que le bus d'adresse présente une adresse utilisable. Normalement connecté à la broche /CE (*Chip Enable*) de la mémoire pour lui signifier de prendre en compte l'adresse ;
- phi, le signal d'horloge généré par la carte Arduino ;
- /RESET permettant de réinitialiser le processeur.

Il nous faut donc en principe 28 broches en tout côté carte Arduino, 18 en entrée et 10 autres en sortie. Notez que le bus de données est censé

*Si vous voulez pousser vos expérimentations plus loin avec un processeur plus puissant et plus rapide, le Motorola 68008 est généralement un bon choix. Il s'agit d'un modèle particulier de 68000 avec un bus de données de seulement 8 bits, mais une architecture 8/16/32 bits ainsi qu'un bus d'adresse de 20 bits. Avec ce genre de processeurs, il devient possible d'envisager de faire fonctionner une version spéciale de Linux (uClinux pour m68k).*






*En reliant les bus d'adresse et de données à une carte Arduino UNO, on peut très simplement expérimenter en émulant la mémoire normalement attachée au Z80, mais nous sommes drastiquement limités par le nombre de broches utilisables. Dans cet état, nous ne pouvons que simuler une mémoire de 64 octets pour fournir des instructions au processeur.*

être bidirectionnel, mais qu'ici nous ne laissons pas le Z80 écrire dans la pseudo mémoire, nous émulons une ROM (EPROM ou UVPROM). Pour émuler une RAM et donc permettre l'écriture en mémoire par le Z80, il faudrait également prendre en compte, en plus, le signal /RW indiquant que le bus de données présente un octet à écrire à l'adresse spécifiée par le bus d'adresse, et surtout, gérer ces broches comme des sorties et des entrées (nous verrons cela plus tard).

Le problème qui se pose tient au fait qu'il existe peu de cartes Arduino en mesure de fournir autant de broches. Deux solutions se présentent alors à nous, soit utiliser une carte Arduino Mega 2560 avec 54 entrées/sorties, qui par la même occasion aurait également assez de mémoire pour émuler les 64 Ko adressables par le Z80, soit limiter le bus d'adresse au maximum utilisable avec les broches restantes.

Dans ce second cas de figure, nous pouvons envisager un brochage comme :

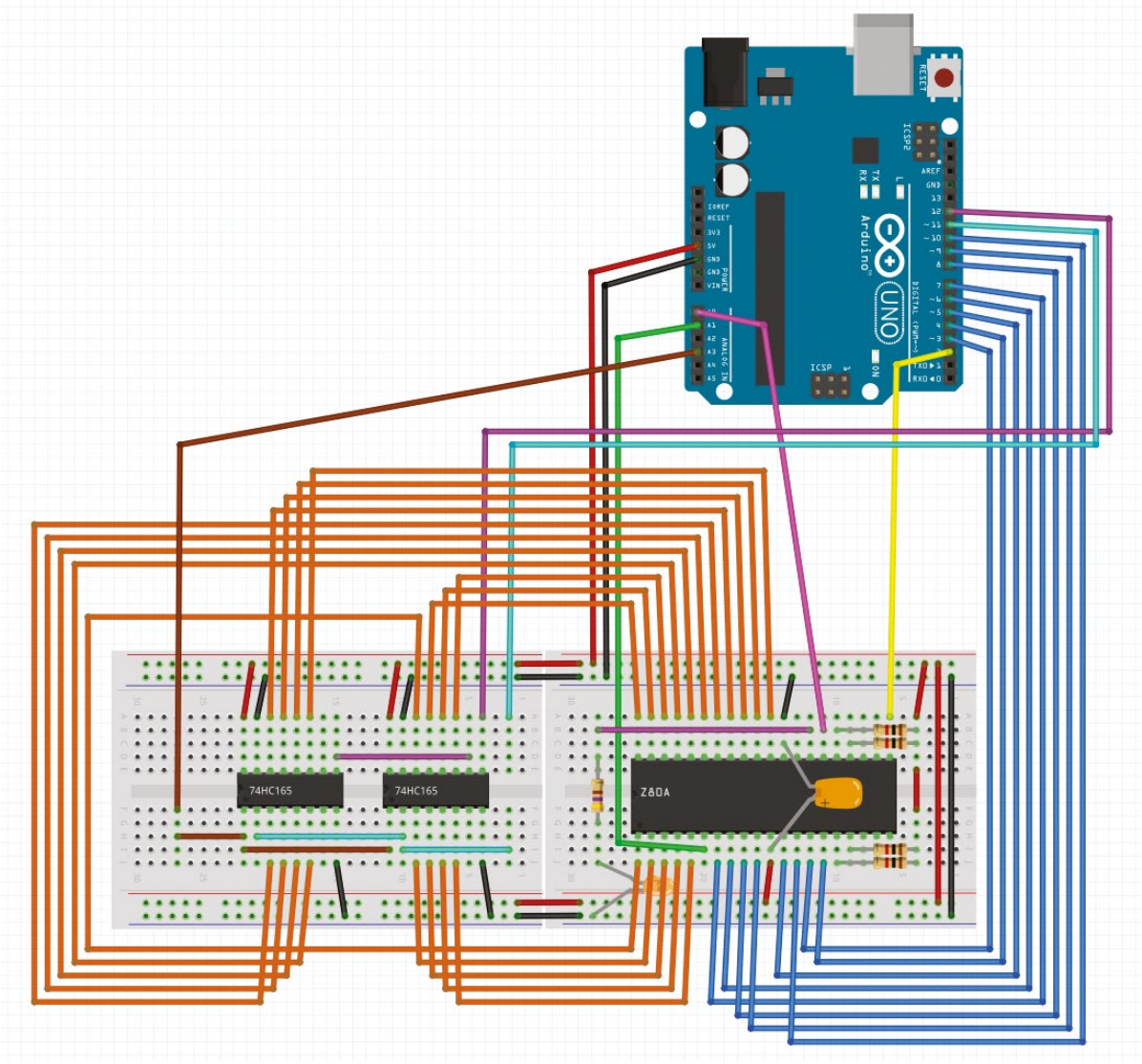
- 4 à 11 : bus de données, D0 à D7 ;
- 2 pour /RD ;
- 3 pour /MREQ (optionnel ici) ;
- 12 pour /RESET ;
- 13 pour l'horloge (phi) ;
- A0 à A5 : 6 bits du bus d'adresse, A0 à A6.

Ici, l'ensemble des broches d'une carte Arduino UNO est utilisé et nous nous retrouvons avec un bus d'adresse de 6 bits, soit un espace adressable de 2 puissance 6, ou 64 octets maximum. Ce n'est pas Byzance, mais c'est plus que suffisant pour écrire un premier programme simple.

Une autre solution existe et consiste à faire intervenir un circuit logique comme le 74HCT165.

Celui-ci fonctionne comme un registre à décalage standard (cf. *Hackable n°5*), mais dans le sens inverse. On présente un état sur ses broches en entrée, on déclenche une lecture par un changement d'état sur une broche (/PL) et on récupère les informations bit par bit. En chaînant deux de ces composants, il devient alors possible de lire les 16 bits du bus d'adresse avec seulement 3 broches de la carte Arduino (5 en réalité, 11-MOSI ne peut être utilisé pour autre chose et 10-SS n'est utilisable qu'en sortie). Le problème de la taille mémoire reste cependant entier, même en n'émulant qu'une mémoire en lecture seule, il n'y a, de toute façon, pas assez de flash dans l'ATmega238P d'une carte Arduino UNO pour gérer 64 Ko de mémoire.

*En utilisant deux circuits logiques 74HC165, nous pouvons faire en sorte que la carte Arduino puisse lire l'état de toutes les lignes du bus d'adresse du Z80. Ceci implique cependant deux composants supplémentaires, une adaptation du croquis et surtout une véritable salade de câbles. On peut alors théoriquement gérer 64 Ko (2^16) de mémoire, mais ne pouvons pour autant pas simuler un tel espace faute d'espace flash côté Arduino.*



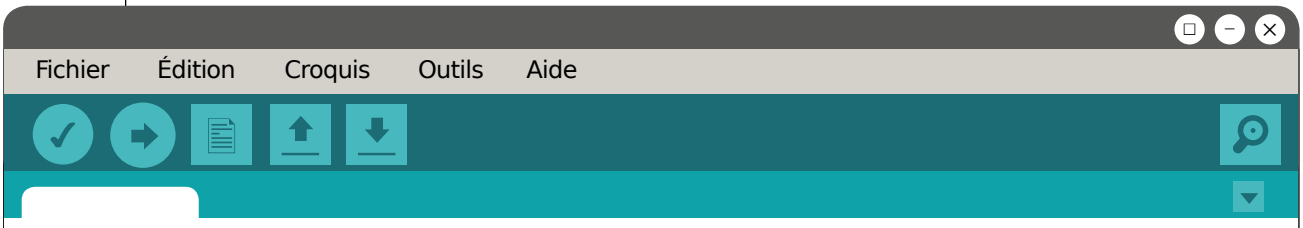
## 2. LE CROQUIS ARDUINO

Sans plus attendre, plongeons dans le code qui, bien que relativement long n'est pas très complexe et n'implique finalement que des changements d'états et des lectures de broches ainsi que des opérations binaires.

Notre stratégie sera la suivante : nous cadençons simplement le Z80 avec une impulsion d'horloge par tour de boucle **loop()** et nous réagissons à un changement d'état en attachant une interruption sur la broche 2 connectée à /RD. Cette routine sera appelée automatiquement dès que l'état de la broche passe de haut à bas (flanc descendant, **FALLING**). Comme une telle routine doit rester concise, nous ne procédons pas à la lecture de l'adresse directement, mais changeons simplement la valeur d'une variable (**doread**) qui sera prise en compte dans la boucle principale ensuite.

Si cela arrive, nous lisons l'adresse fournie par le Z80, et l'utilisons pour obtenir un octet dans un tableau de **byte** faisant office de mémoire (**mem[]**). Enfin, nous présentons cet octet sur le bus de données avant de réinitialiser la valeur de **doread** et donner le signal d'horloge suivant.

Notez qu'une autre approche aurait été possible en se passant de l'interruption, mais cela demande de prendre en compte une information capitale : un cycle d'horloge ne correspond pas à un cycle machine. Certaines instructions comme **NOP** utilisent 4 cycles d'horloges et d'autres comme **JP** en utilisent 10. Nous ne pouvons donc pas afficher nos informations à chaque cycle d'horloge et donc à chaque tour dans **loop()**. Pour contourner ce problème, il faudrait alors garder en mémoire la précédente adresse traitée et ne réagir que si la nouvelle est différente. J'ai fait le choix de ne présenter ici que la première solution pour réduire la taille de l'article, mais aussi parce que je trouve l'approche avec interruption plus élégante (la version sans interruption est disponible avec les autres codes du numéro sur GitHub) :




```
#define B_CLOCK  12  // horloge
#define B_RESET  13  // réinitialisation
#define B_MREQ   3   // requête mémoire
#define B_RD     2   // lecture

#define BITS_ADDR  6  // nombre de bits d'adresse
#define CLKDELAY  60  // délai horloge

// Bus d'adresse
//                  bit       0  1  2  3  4  5
int pinsAddress[BITS_ADDR] = {A0,A1,A2,A3,A4,A5};

// Bus de données
//              bit   0 1 2 3 4 5  6  7
int pinsData[8]     = {4,5,6,7,8,9,10,11};

// Variable pour la routine d'interruption
volatile int doread=0;
```

```
// Routine d'interruption
void readISR() {
  doread=1;
}

// Fonction horloge
void doClock(unsigned int n) {
  for(int i=0; i<n; i++) {
    digitalWrite(B_CLOCK, HIGH);
    delay(CLKDELAY);
    digitalWrite(B_CLOCK, LOW);
    delay(CLKDELAY);
  }
}

// Réinitialisation
void doReset() {
    digitalWrite(B_RESET, LOW);
    doClock(10);
    digitalWrite(B_RESET, HIGH);
}

// Lecture de l'adresse
unsigned int getaddr() {
  unsigned int addr = 0;

  for(int pin=0; pin<BITS_ADDR; pin++) {
      if(digitalRead( pinsAddress[pin] ) == HIGH )
        addr |= (1 << pin);
  }

  return addr;
}

// Définition des bits de données
void setData(unsigned char data) {
    for(int pin=0; pin<8; pin++) {
      if(data & 1)
        digitalWrite(pinsData[pin], HIGH);
      else
        digitalWrite(pinsData[pin], LOW);
      data >>= 1;
    }
}

// Configuration
void setup() {
  pinMode(B_CLOCK, OUTPUT);
  pinMode(B_RESET, OUTPUT);
  pinMode(B_MREQ, INPUT);
  pinMode(B_RD, INPUT);

  // Tout le bus de données en sortie
  for( int pin = 0; pin < 8; pin++ ) {
    pinMode(pinsData[pin], OUTPUT);
  }

  // Tout le bus d'adresse en entrée
  for( int pin = 0; pin<BITS_ADDR; pin++ ) {
    pinMode(pinsAddress[pin], INPUT);
  }
```

```
  // Activation moniteur série
  Serial.begin(115200);
  delay(25);

  // Reset du Z80
  Serial.println("Reset Z80");
  doReset();
  delay(10);

  // On attache l'interruption
  // Un changement d'état haut vers bas appelle notre routine
  attachInterrupt(digitalPinToInterrupt(B_RD), readISR, FALLING);
}

// Mémoire pour notre Z80
unsigned char mem[] = {
  0xc3, 0x04, 0x00, 0x00, 0x00, 0x00, 0xc3, 0x00, 0x00
};

// Boucle principale
void loop() {
  // La routine d'interruption a été appelée,
  // nous avons du travail à faire
  if(doread) {
    // Récupération de l'adresse
    unsigned int addr = getaddr();
    // Affichage
    Serial.print("Adresse: 0x");
    if(addr < 0x10)   Serial.print("0");
    if(addr < 0x100)  Serial.print("0");
    if(addr < 0x1000) Serial.print("0");
    Serial.print(addr, HEX);

    // Si l'adresse est valide (dans le tableau mem)
    if(addr >= 0 && addr < sizeof(mem)) {
      // On "pousse" les données correspondantes sur le bus
      setData(mem[addr]);
      // Et on affiche le tout
      Serial.print(" [0x");
      if(mem[addr] < 0x10)   Serial.print("0");
      Serial.print(mem[addr], HEX);
      Serial.println("]");
    } else {
      // Nous sommes hors de l'espace mémoire émulé
      // On renvoie 0x00, l'instruction NOP
      setData(0x00);
    }
    // On confirme la prise en charge
    doread=0;
  }

  // Impulsion horloge
  doClock(1);
}
```

Arduino

Pour travailler sereinement, nous avons donc ici quatre fonctions :

- `doClock()` : envoie une impulsion sur la broche phi du Z80 en guise de signal d'horloge ;
- `doReset()` : provoque la réinitialisation « propre » du Z80 en mettant la broche RESET à la masse pendant 10 cycles d'horloge ;
- `getaddr()` : récupère l'adresse sur 6 bits lus sur les broches A0 à A5, et donc A0 à A5 du Z80, et la retourne ;
- `setData()` : définit l'état des broches correspondant aux 8 bits de données en fonction d'un octet passé en argument.

La partie la plus importante ici est, bien entendu, `mem[]`, notre tableau de `unsigned char` (ou `byte`) constituant notre « fausse » mémoire. Comme nous n'utilisons que 6 des 16 lignes d'adresses, nous ne pouvons gérer que 64 octets de mémoire. Nous prévoyons d'ailleurs un test dans la boucle principale afin de nous assurer que nous ne « sortons » pas du tableau durant l'exécution.

Il ne nous reste donc plus qu'à remplir le tableau avec des instructions à destination du Z80 et un premier essai pourrait être de simplement y placer 64 fois `0x00` correspondant à 64 occurrences de l'instruction `NOP`, *No OPeration*, ne rien faire. Mais il est plus intéressant de voir un véritable programme fonctionner, chose que nous allons traiter maintenant...

## 3. UN PEU D'ASSEMBLEUR POUR LA ROUTE ?

Nous connaissez déjà `NOP`, l'instruction qui ne fait rien, si ce n'est dire au processeur que l'instruction est valide et qu'il doit simplement passer à la suivante. Nous allons donc en introduire une nouvelle pour ajouter un peu de piquant : `JP` ou `JP **` comme spécifié dans la documentation du processeur Z80.

« JP » signifie *JumP*, un saut. Mais il existe plusieurs types de sauts parmi les instructions du Z80. Celle qui nous intéresse est un **saut inconditionnel** ayant pour *opcode* `0xc3`. L'instruction complète est constituée de 3 octets : l'opcode `0xc3`, la partie basse de l'adresse (8 bits de poids faible) et la partie haute (8 bits de poids fort).

Le bus d'adresse du Z80 est sur 16 bits, mais il ne s'agit que d'un processeur 8 bits et donc incapable de gérer des valeurs supérieures à 8 bits (le bus de données et la quasi-totalité des registres). Pour préciser l'adresse du saut ou, en d'autres termes, la prochaine adresse à lire pour obtenir un opcode, nous devons donc le faire en deux parties. L'ordre dans lequel ces octets sont présentés et traités est déterminé par l'architecture du processeur et donc le choix du constructeur. On parle d'*endianness* (une traduction possible du terme est « boutisme », mais il faut reconnaître que parler d'architectures « petit-boutistes » et « gros-boutistes » est tout simplement affreux).

Le Zilog Z80, comme les processeurs Intel x86 et compatibles, sont des architectures little-endian dans lesquelles, par exemple, la valeur `0x12345678` sera stockée en mémoire sous la forme des octets `0x78`, `0x56`, `0x34` puis `0x12`. Les 8 bits qui « pèsent » le plus dans la valeur 32 bits qu'est `0x12345678` sont stockés en premier. On dit que l'octet de poids le plus fort (MSB ou *Most Significant Byte*) est stocké à une adresse inférieure à l'octet de poids le plus faible (LSB pour *Least Significant Byte*).

Ainsi, lorsqu'on passe les deux octets après `0xC3`, on doit fournir d'abord l'octet de poids faible (LSB) et ensuite l'octet de poids fort (MSB), car c'est ainsi que le processeur attend ces informations. Avec une architecture big-endian, cette adresse se présentera de façon plus « naturelle » où `0x1234` serait `0x12` et `0x43`. C'est un point à prendre en considération si vous considérez expérimenter avec autre chose qu'un Z80 (ou un MOS 6502/6501). Le Motorola 68000 par exemple est big-endian.

Avec nos deux instructions à notre actif, vous pouvons donc imaginer un programme simpliste tel que :

- sauter à une adresse ;
- faire quelques `NOP` ;
- sauter à l'adresse de départ.

Nous avons là une boucle infinie qui, en assembleur pourrait être quelque chose comme :

```
JP 0x0004
NOP
NOP
NOP
JP 0x0000
```

Ce qui en termes de code machine nous donnerait `0xc3, 0x04, 0x00, 0x00, 0x00, 0x00, 0xc3, 0x00, 0x00` car :

```
0x0000: 0xc3 0x04 0x00   JP 0x0004
0x0003: 0x00             NOP
0x0004: 0x00             NOP
0x0005: 0x00             NOP
0x0006: 0xc3 0x00 0x00   JP 0x0000
```

Le croquis Arduino présenté précédemment possède justement ces 9 octets (5 instructions) au début du tableau `mem[]` et son exécution affiche donc sur le moniteur série la sortie suivante :

```
Reset Z80
Adresse: 0x0000 [0xC3]
Adresse: 0x0001 [0x04]
Adresse: 0x0002 [0x00]
Adresse: 0x0004 [0x00]
Adresse: 0x0005 [0x00]
Adresse: 0x0006 [0xC3]
Adresse: 0x0007 [0x00]
Adresse: 0x0008 [0x00]
Adresse: 0x0000 [0xC3]
Adresse: 0x0001 [0x04]
Adresse: 0x0002 [0x00]
Adresse: 0x0004 [0x00]
Adresse: 0x0005 [0x00]
Adresse: 0x0006 [0xC3]
Adresse: 0x0007 [0x00]
Adresse: 0x0008 [0x00]
```

On retrouve là les adresses correspondant à notre code machine assemblé à la main et les différents octets retournés au Z80 entre crochets. Celui-ci les traite exactement comme prévu et le programme boucle comme attendu. Si vous connectez des leds aux broches /RD, /M1 et phi vous pourrez d'ailleurs observer la différence d'activité dans la lecture des octets, le premier cycle machine et le signal d'horloge.

C'est le moment de s'autocongratuler : bravo, vous avez écrit et exécuté votre premier vrai programme en assembleur pour un processeur inventé il y a plus de 40 ans ! Et en plus il fonctionne !

## 4. DONNER LE SALE BOULOT À LA MACHINE

Le petit côté vintage dans le fait d'assembler à la main un programme est amusant mais, il faut le reconnaître, cela peut vite devenir pénible. En faire l'expérience avec un code aussi simple que celui que nous avons testé ici vous donne déjà une petite idée de la subtile torture mentale quotidienne vécue par nos aïeux, pionniers de l'informatique.

Comme le dit l'agent Smith, ne confiez jamais à un humain le travail d'une machine. On peut donc simplifier notre tâche en faisant intervenir une Raspberry Pi (ou un PC sous GNU/Linux) pour ne plus avoir à assembler le code à la main. Nous pouvons écrire de l'assembleur et utiliser un assembleur pour processeur Z80 afin de faire faire la traduction à notre place. Ceci nous épargne principalement deux activités : penser en termes d'adresses et réfléchir en little-endian.

# ACTUELLEMENT DISPONIBLE

## LINUX PRATIQUE HORS-SÉRIE N°40 !


# NE LE MANQUEZ PAS

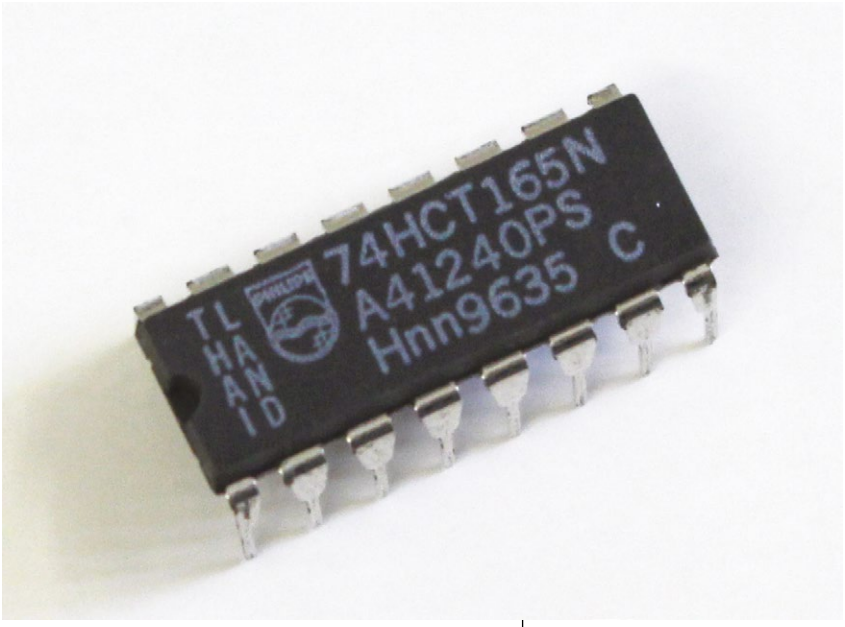
## CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

## https://www.ed-diamond.com

En lieu et place des adresses pour les sauts par exemple, nous pouvons utiliser des étiquettes (ou labels) pour référencer des endroits spécifiques du code et de la mémoire. Nous pouvons également ajouter des commentaires, des macros et surtout, l'assembleur se charge de l'endianness, nous laissant la possibilité de spécifier les valeurs 16 bits de façon plus lisible (pour nous simples humains).

Le Z80 est un processeur bien connu, ayant une communauté d'utilisateurs très importante et surtout il est encore utilisé dans une version plus moderne dans certaines calculatrices graphiques/scientifiques (*Texas Instruments* en particulier). Il n'y a donc rien d'étonnant à trouver plus d'un assembleur pour ce processeur, y compris parmi les paquets disponibles par défaut dans votre système Raspbian (et donc aussi Debian sur PC) :

- **crasm** 1.8 : un assembleur pouvant générer du code binaire pour les processeurs 6800, 6801, 6803, 6502, 65C02 et Z80 ;
- **pasmo** 0.5.3 : un assembleur plus jeune, décrit par l'auteur comme simple d'utilisation (??) ;
- **z80asm** 1.8 : un classique du genre avec un code stable, mais une documentation assez réduite ;
- **binutils-z80** 2.24.28 : les très connus et utilisés utilitaires binaires GNU pour la cible Z80 incluant tous les outils classiques.




*Le circuit logique 74*165 est un registre à décalage, mais à l'opposé d'un composant comme le 74*595 permettant de piloter des sorties, celui-ci permet de lire des entrées. Ainsi un seul 74HC595 piloté via 3 broches vous permettra d'ajouter 8 sorties à votre carte Arduino, mais le 74HC165, lui, fera de même pour lire 8 entrées. De plus, l'un comme l'autre peuvent être chaînés pour multiplier à volonté les sorties ou entrées, toujours avec seulement 3 broches de contrôle.*

Le choix est relativement difficile, car c'est avant tout une affaire de goût. **z80asm** est souvent mentionné sur le Web, mais la documentation officielle est presque inexistante. **crasm** présente l'avantage de supporter plusieurs processeurs 8 bits, mais produit uniquement des fichiers textes (Intel Hex) peu pratiques pour notre usage. Les *binutils*... eh bien, ce sont les *binutils*... Puissants, riches, fonctionnels et... parfois difficiles à prendre en main. Et enfin, **pasmo** fait le travail tout simplement, mais est encore un peu jeune.

Pour quelque chose d'aussi simple que le fait de reproduire notre exemple, tous ces compilateurs pourront faire l'affaire avec un petit moins pour les *binutils* plus difficiles à mettre en œuvre. Vous trouverez sur le dépôt GitHub du numéro des exemples pour les quatre assembleurs cités ici, mais je ne détaillerai, dans la suite, que l'utilisation de **z80asm** (un choix totalement arbitraire, sinon aléatoire).

Pour installer **z80asm** sur votre Raspberry Pi, rien de plus simple, il suffit de vous plier d'un **sudo apt-get install z80asm** et le tour est joué. Il vous faudra également un bon éditeur de texte/code comme Vim, Emacs ou

Nano qui est installé par défaut (message subliminal totalement discret et subtil : mettez-vous à Vim), ainsi que l'outil **xxd** pour convertir le résultat en quelque chose d'utilisable directement dans votre croquis.

Il ne vous reste plus maintenant qu'à écrire votre premier programme en assembleur, qui sera presque identique au précédent et nous permettra de valider le résultat :

```
org 0x0000

; un commentaire

.INIT:  JP .PLOP
        NOP
.PLOP:  NOP
        NOP
        JP .INIT
```

La directive **org** n'est pas destinée au processeur, mais à l'assembleur lui-même. Elle permet de spécifier l'adresse de départ du code, ici 0x0000. Ce sera donc le point de référence pour que l'assembleur puisse travailler. Notre programme débute par une étiquette (**.INIT:**) marquant l'adresse courante. Ceci permet de nous y référer ensuite comme s'il s'agissait d'une adresse (un peu comme une macro en C). **.PLOP:** est également une telle étiquette qui nous sert immédiatement pour faire notre premier saut. Le programme se poursuit donc avec deux **NOP** puis un saut vers l'adresse représentée par **.INIT**.

Pour assembler le programme, nous l'enregistrons dans un fichier (**premier.asm**) et appelons :

```
$ z80asm -o premier.bin premier.asm --list=premier.lst
```

La commande prend en argument **-o** suivi d'un nom de fichier pour stocker le résultat, le fichier source assembleur et une option **--list=** très intéressante puisqu'elle permet d'obtenir un *listing* du code assemblé :

```
# File premier.asm
0000                    org 0x0000
0000
0000                    ; un commentaire
0000
0000 c3 04 00           .INIT:  JP .PLOP
0003 00                         NOP
0004 00                 .PLOP:  NOP
0005 00                         NOP
0006 c3 00 00                   JP .INIT
# End of file premier.asm
0009
```

L'assembleur décrit ici son travail. On y retrouve, bien entendu, notre code, mais également la traduction en code machine ainsi que les adresses utilisées. À ce stade déjà, vous devez reconnaître les octets que nous avons déjà utilisés dans le croquis Arduino. Mais nous pouvons pousser plus loin encore, en utilisant **xxd** pour produire un morceau de code en C à partir du fichier **premier.bin** résultant de l'assemblage :

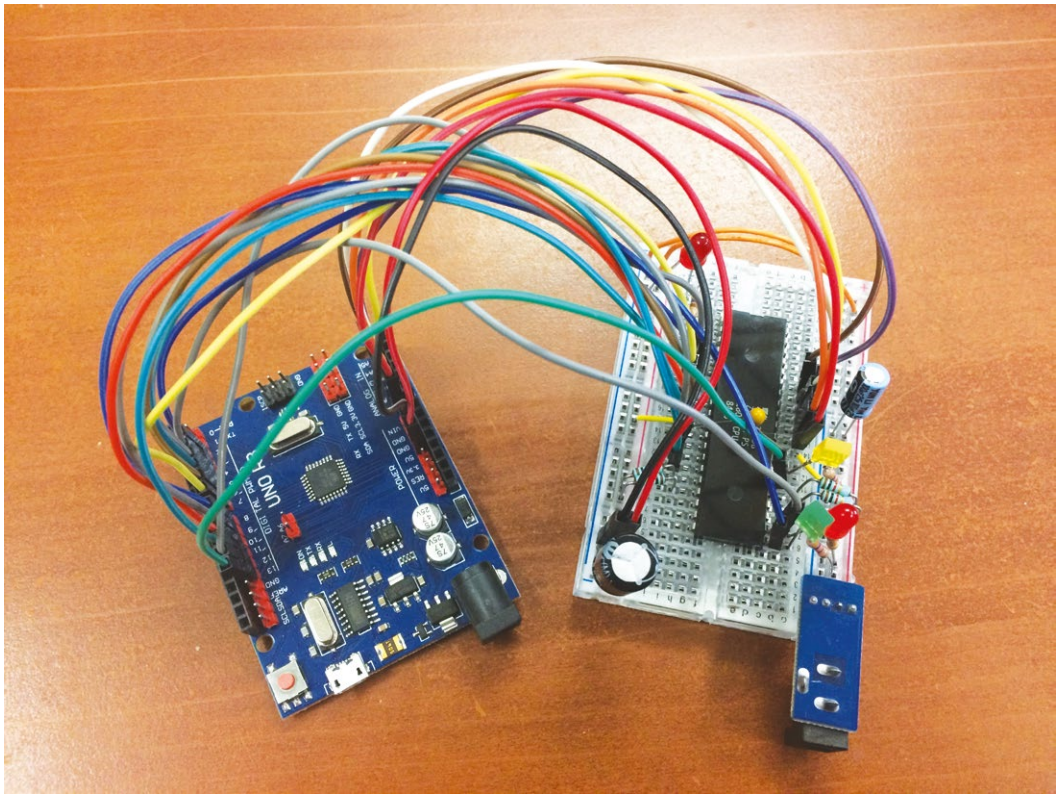
```
$ xxd -i premier.bin
unsigned char premier_bin[] = {
  0xc3, 0x04, 0x00, 0x00, 0x00, 0x00, 0xc3, 0x00, 0x00
};
unsigned int premier_bin_len = 9;
```

Nous avons là une déclaration de variable tout à fait standard et parfaitement conforme à ce que nous avions fait à la main, à l'octet près ! Le nom de variable utilisé est celui du fichier (avec le `.` remplacé par un _). `premier_bin_len`, précisant la taille des données, ne nous est pas utile ici (nous utilisons `sizeof()`). Il nous suffit alors de copier/coller ce code dans notre croquis, de modifier le nom de la variable et nous obtenons le même résultat que précédemment.

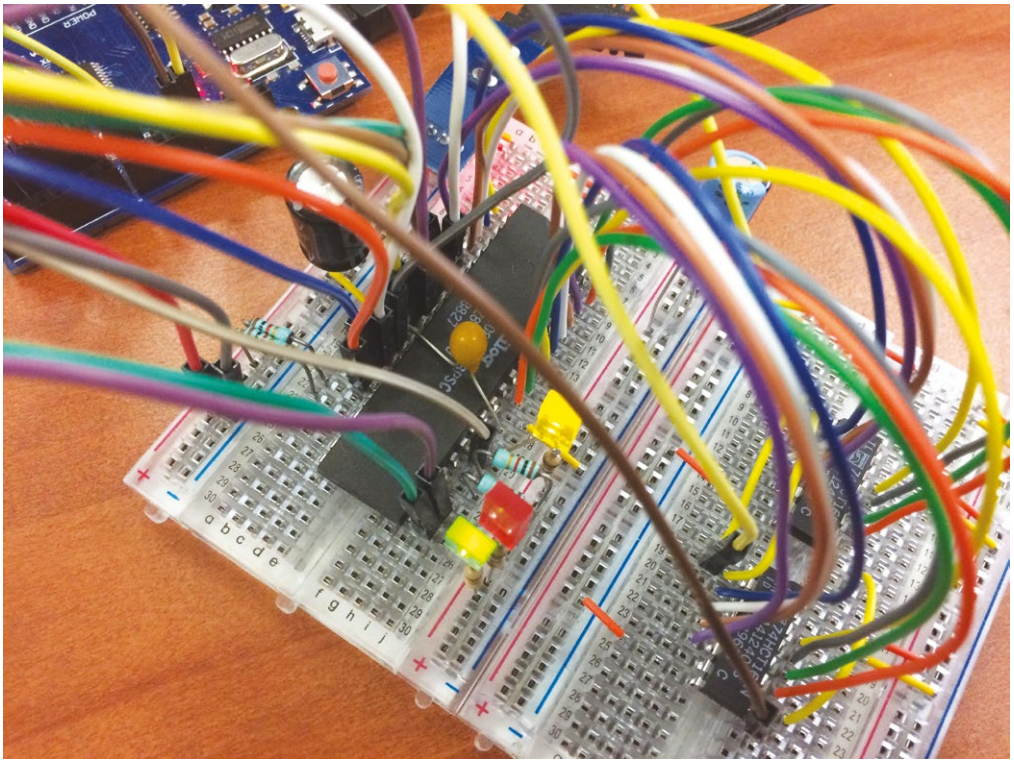
Nous venons d'utiliser un assembleur Z80 sur Raspberry Pi pour produire un code machine pour notre plateforme d'expérimentation et celui-ci s'exécute à merveille.

## POUR FINIR (TEMPORAIREMENT)

Avec cet article, nous avons avancé d'un grand pas dans notre projet, ainsi que de quelques années dans l'histoire de l'informatique. Certes, c'est encore loin d'être un ordinateur complet ou même un équivalent à une carte Arduino, mais nous nous rapprochons petit à petit de l'objectif. Nous pouvons maintenant écrire des codes en assembleur et laisser les outils faire le travail, ce qui n'est pas sans rappeler une approche relativement moderne.



*Nous utilisons ici l'intégralité des broches (GPIO) disponibles d'une carte Arduino UNO et même si nous ne pouvons simuler que 64 octets de mémoire (6 lignes du bus d'adresse sur les 16 disponibles), cela fait déjà beaucoup de connexions et donc autant de sources d'erreurs ou de faux contacts.*



*Une variation du montage utilise deux 74HC165 qui donnent accès à l'intégralité des 16 bits d'adresse. Côté Arduino, la lecture d'une adresse se fait alors via seulement 3 broches, mais il faut tout de même connecter le Z80 aux 74HC165... et nous n'avons pas assez de mémoire pour simuler l'ensemble des 64 Ko adressables.*

Je ne sais pas encore ce qu'il y aura de prévu pour le prochain article. Sur cette base, on peut commencer à travailler avec des périphériques comme un port série (SIO) ou un timer (CTC) par exemple. On peut également améliorer notre concept de base en utilisant les 74HC165 pour gérer les 16 bits d'adresses, basculer sur une carte Arduino Mega 2560 ou ajouter une EEPROM i2c en guise d'espace de stockage (pour simuler davantage de ROM). Enfin, on peut également décider de prendre en charge une mémoire vive et donc de permettre les écritures en mémoire par exemple...

Ce dernier point est important, car qui dit « mémoire vive » dit « possibilités d'utilisation d'une pile exécution » (*stack*), une zone mémoire dédiée au stockage des adresses pour l'appel de routines (**CALL** et **RET** en assembleur). Le processeur Z80 possède tout ce qu'il faut pour cela (registre SP) et c'est une brique préliminaire indispensable pour aller plus loin que l'assembleur : envisager d'écrire des programmes en C !

Nous verrons bien ce qui m'inspire le plus dans toutes ces pistes, mais d'ici là, je vous recommande de faire vos propres essais, en particulier en assembleur. Car même si nous finirons avec un langage de plus haut niveau comme le C, la compréhension de l'assembleur reste quelque chose, dans ce domaine, qui est très important. C'est un style de programmation très particulier, mais qui aide vraiment à comprendre et assimiler des concepts que vous utilisez déjà sans le savoir avec vos cartes Arduino. Qui sait, peut-être y prendrez-vous goût et tenterez également de faire de l'assembleur avec vos cartes Arduino, ce qui est quelque chose de tout à fait possible, mais cela est une tout autre histoire... et un tout autre article. **DB**

Robobox

# LA BOX

## QUI FAIT NAITRE LES ROBOTICIENS DE DEMAIN

WWW.ROBOBOX.FR

dès 19,99€ / mois !

CONSTRUISEZ CHAQUE MOIS UN NOUVEAU ROBOT
ET FORMEZ VOUS A LA ROBOTIQUE