N°15 | NOV. - DÉC. 2016

HACKABLE MAGAZINE

DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

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

ARDUINO / LEDS

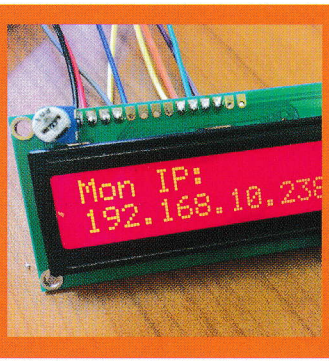
Recyclez une vieille lanterne et créez une décoration magique qui réagit au toucher

p. 32

RASPBERRY PI

Connectez un module LCD à votre Raspberry Pi pour afficher son adresse au démarrage

p. 66



PROJETS / PARTAGE

Mettez de l'ordre dans vos projets et partagez vos créations avec Git et GitHub

p. 74

AUTONOMIE

Faire fonctionner vos montages sur batterie externe pour smartphone ? Pas si simple !

p. 90

Modules Bluetooth : Contrôlez votre Arduino depuis votre Smartphone

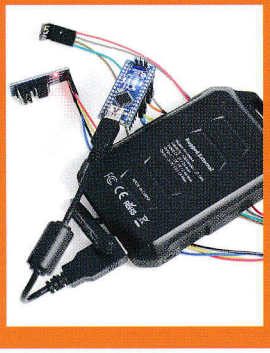
p. 44

- Connectez et configurez vos modules Bluetooth
- Pilotez votre éclairage multicolore
- Activez/désactivez n'importe quel matériel avec des relais

ARDUINO

Créez un enregistreur autonome pour collecter température, humidité et pression

p. 18



RASPBERRY PI

Contrôlez facilement votre alimentation de laboratoire à distance avec votre Pi

p. 06

L 19338 - 15 - F: 7,90 € - RD

TOGETHER WE
ARE PUSHING THE BOUNDARIES
OF SECURITY*

thalesgroup.com/careers

I CYBERSECURITY

#ThalesCyber

La ferme des animaux vous évoque davantage de croustillants binaires à vous mettre sous la dent plutôt qu'un lieu bucolique ?

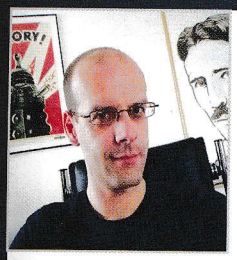
Aucun IOC ne vous échappe et vous avez envie de vous engager dans des investigations à grande échelle ?

Vous avez une soif insatiable d'uid 0 et avoir une longue root devant vous ne vous fait pas peur ?

Alors rejoignez-nous !
La porte (non dérobée) est ici !

THALES
Together • Smarter • Safer

ÉDITO



Y'a des jours comme ça...

Hé oui, tous les jours ne se valent pas. Il y a les jours où tout fonctionne à la perfection, les projets aboutissent, des bibliothèques abscons deviennent limpides, son *pull request* (cf. article sur Git et GitHub) est accepté par les développeurs d'un raytracer mythique avec plus de 25 ans d'histoire (POV-Ray) et son code C « maison » permettant d'émettre avec un HackRF One qui refusait obstinément de fonctionner, fait des merveilles suite à une sorte d'illumination/révélation trigonométrique. Et puis il y a les autres jours...

Comme celui où l'on apprend que Texas Instruments lance une carte destinée à l'éducation, se branchant directement sur trois modèles de calculatrices de la marque (TI-83, TI84 et Nspire). Le TI-Innovator Hub semble très amusant et permet effectivement de contrôler toutes sortes de sorties et de capteurs directement depuis des programmes écrits sur la calculatrice.

Plein d'entrain on se heurte à une première déception en apprenant que la calculatrice TI-84 Plus CE (celle qui attisait ma convoitise) n'existe apparemment pas sur le marché français (on passe étrangement de la TI-83 à la TI-89 sur le site). Dommage, car j'avais déjà récupéré un compilateur C fort sympathique et l'idée d'animer un processeur Z80, héritier d'une lignée presque aussi vieille que moi, me rendait déjà tout chose...

Allez, ce n'est pas grave, il me reste l'option de la TI-Nspire CX. Exit du Z80, un ARM c'est pas mal non plus côté histoire tout en permettant une programmation en Basic (beurk), en Lua, ainsi qu'en C/C++ et même en assembleur ARM... à condition d'installer un « jailbreak » appelé Ndless ?!? Tiens, c'est nouveau, je ne savais pas que les calculatrices aussi étaient devenues des horreurs où l'utilisateur, qui a pourtant ...

suite page 4

Hackable Magazine
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIÈRE GRISE

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

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

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

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

Suivez-nous sur Twitter
@hackablemag

À PROPOS DE HACKABLE...

HACKS, HACKERS & HACKABLE

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

SOMMAIRE

suite ÉDITO

...acheté tout à fait légitimement le matériel et donc en principe obtenu le droit de l'utiliser comme bon lui semble, doit « pirater » son achat pour en jouir pleinement. Très bien, je prends note, et classe les calculatrices dans la même catégorie que les iPhone à jailbreaker et les smartphones Android à rooter... Déprimant...

À ce stade bien que ma motivation soit déjà largement entamée, je poursuis néanmoins mes investigations. Ce TI-Innovator Hub repose sur le microcontrôleur MSP432 et ressemble fortement au Ti Launchpad MSP432 (récemment passé du noir au rouge). Ça tombe bien, le TI-Innovator est introuvable à la vente et j'ai déjà un Launchpad MSP432. Il me suffit donc de pointer mon navigateur sur l'endroit où Texas met à disposition les sources du code à placer dans la carte et... Tiens, je ne trouve pas de firmware pour ce matériel, même déjà compilé, c'est étrange.

Insistons...

Après quelques longues minutes de recherches et grâce aux informations collectées par les passionnés de TIplanet (**https://tiplanet.org/forum/**), voici que je tombe sur **https://education.ti.com/ide/stem/**. Chic alors, je vais pouvoir enregistrer dans mon Launchpad le firmware de... Comment ça je dois installer une extension dans mon navigateur web ? Une extension qui peut communiquer avec les sites Ti et des applications natives de mon système ? Certainement pas ! Et je devrais installer sur mon système le *TI Cloud Agent Application* qu'on me propose de télécharger sous la forme d'un programme compilé ! Qui en plus est en 64 bits alors que ma présente machine est en configuration 32 bits ?!

Non, non, messieurs. Je n'installe pas de programme dont je ne suis pas sûr du comportement, ni même d'extensions et je ne vais certainement pas m'amuser (sic) à me lancer maintenant dans une migration 32/64 bits, tout ça parce que vous n'êtes pas fichu de mettre quelque part une pauvre archive avec les sources d'un firmware qui ne contient certainement pas de secrets industriels que la diabolique concurrence risque de vous dérober. D'autant que l'idée initiale consistant à brancher une carte pédagogique à une calculatrice Ti a été implémenté il y a fort longtemps par Christopher Mitchell et sa bibliothèque ArTICL, fonctionnant sur Arduino **ET** MSP432, disponible sur GitHub sous licence BSD ! Entre nous, cela ne m'étonnerait qu'à moitié de trouver des morceaux de ArTICL dans le firmware du TI-Innovator Hub, la licence BSD le permettant sans le moindre problème légal.

C'est précisément à ce stade que, écœuré, je décide de jeter l'éponge. Et dire que j'étais prêt à acheter une calculatrice à presque 150€ alors que ma Casio fx8500g de 1991 marche encore très bien...

Au final, la question n'est pas de savoir à quoi joue Ti, si le marché de l'éducation nécessite des politiques particulières ou si ce fameux firmware est disponible ou non sous une forme acceptable et bidouillable, caché au fin fond d'un serveur demandant une inscription. Non, la vraie question consiste à se demander pourquoi diable quelque chose qui est destiné à être un support ou un accessoire pédagogique ne mérite pas l'adoption d'un comportement plus favorable à ceux qui sont les plus demandeurs de connaissances et de savoir : vous, moi, nous, les élèves, les professeurs, les parents...

Il y a là quelque chose de totalement paradoxal, voire de foncièrement malsain. L'apprentissage à un domaine est, à mon sens, aussi et surtout l'assurance qu'on pourra explorer et exploiter ce même domaine autant qu'on le désire, sans autre restriction que sa propre volonté d'apprendre et sa capacité à assimiler les concepts nouveaux. En d'autres termes, on est naturellement plus enclin à apprendre à correctement choisir des citrons, dès lors qu'on sait qu'on pourra à terme les presser jusqu'à la dernière goutte, et non simplement admirer leurs belles nuances de jaunes et de verts.

Alors bien sûr, tous les élèves ne vont pas, loin de là, passer du stade « *j'allume une led avec ma calculatrice avec la ligne de code qu'on me dit de taper* » à « *je vais réécrire tout le support, les pilotes et le firmware pour maîtriser la chaîne complète* ». Mais, et c'est un énorme « mais », je pense qu'il est capital d'avoir le sentiment, sinon l'assurance, dès le départ de pouvoir le faire si on le désire.

La pédagogie se résume à un ensemble de méthodes et pratiques visant à transmettre une connaissance, un savoir ou un savoir-faire, et ceci implique le fait d'avoir des perspectives et des libertés. Si, en revanche, il s'agit d'inculquer des pratiques devant être reproduites dans un contexte borné et avec une séquence prédéfinie, ceci s'appelle de la programmation ou, dans le cas d'un être vivant, du dressage.

Le passage de l'un à l'autre tient en peu de choses, mais favoriser un cadre spécifique par rapport à une liberté d'exploration individuelle est sans le moindre doute ce qui fait la différence entre enseignement et dressage. Le simple fait de rendre difficile l'accès à la technologie sous-jacente d'un équipement comme le TI-Innovator Hub sous la forme d'outils adaptés et de sources pour le firmware est quelque chose de parfaitement déplorable et contre-productif.

Même si les informations ne sont peut-être pas réellement sciemment dissimulées, mais simplement éparpillées et rendues, de ce fait, difficiles à collecter, cela forme une barrière qu'il est bien trop facile de considérer comme insurmontable. Même pour moi, qui suis généralement prompt à être motivé par la technologie, par passion, et non un élève de lycée ayant généralement d'autres divertissements, ceci est tout bonnement pesant. Dans l'état, il me semble bien que ce TI-Innovator Hub n'est qu'une boîte noire qui est bien loin de pouvoir faire fantasmer la prochaine génération d'ingénieurs, de programmeurs et d'électroniciens. C'est triste. Dommage et triste...

Mais il serait regrettable de conclure cet édito sur une aussi déplorable constatation, aussi je profite de l'espace qu'il reste (comment ? On me dit à l'infographie qu'il ne reste pas de place ? Mais si voyons, il suffit de réduire la taille de la police), pour vous annoncer que le premier hors-série du magazine, paru cet été, revient chez votre marchand de journaux à partir du 1er décembre. Voici donc une seconde occasion de mettre la main dessus si vous ou quelqu'un de votre entourage souhaite être initié au monde de l'Arduino et de la programmation (la vraie, en C/C++). Il pourra être, par exemple, glissé discrètement sous le sapin accompagné d'une ou deux cartes et de quelques composants pour passer un réveillon tout clignotant en famille, en découvrant une plateforme **ouverte** offrant des perspectives **vraiment** illimitées ;)

Denis Bodor

COMPRENDRE, UTILISER ET ADMINISTRER LINUX

GNU LINUX PRATIQUE ESSENTIEL

À partir du 23 décembre, Linux Pratique change... ...découvrez sa

Nouvelle formule !

+ de Raspberry Pi

+ de tutoriels

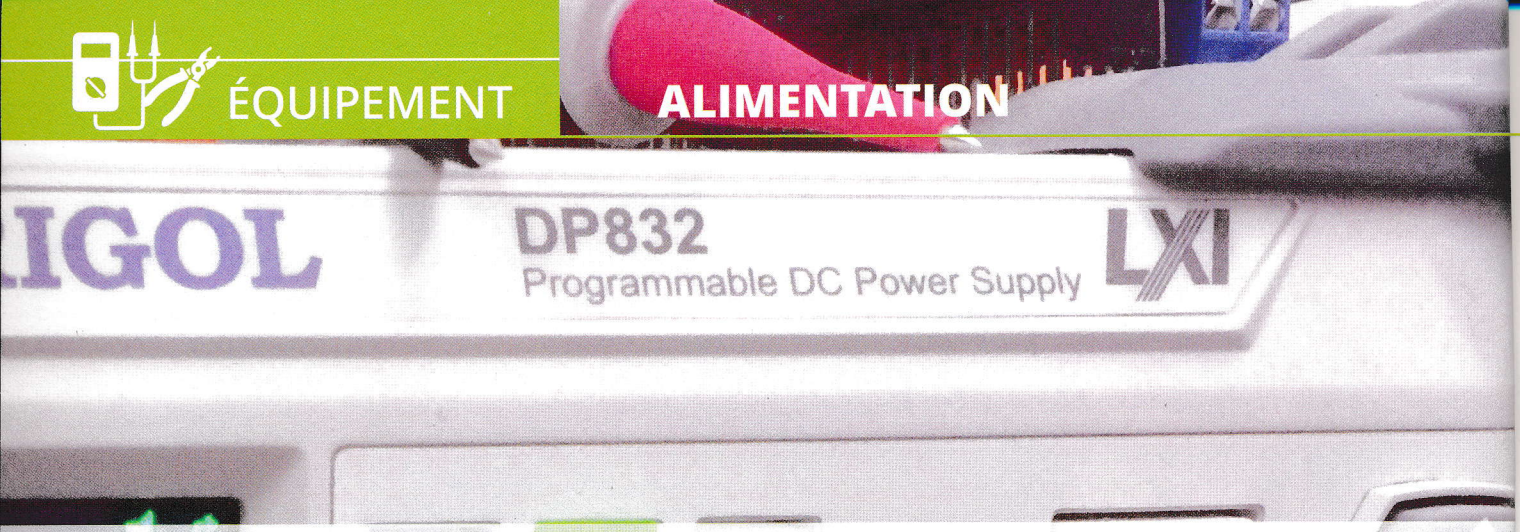
+ d'initiation à la programmation

+ de logithèque

Les nouvelles rubriques de votre magazine :

- Cahier Raspberry Pi & débutant Linux
- Programmation & scripts
- Logithèque & applicatif
- Mobilité & objets connectés
- Système & personnalisation
- Web & réseau
- Terminal & ligne de commandes
- Entreprise & organisation
- Réflexion & société ...

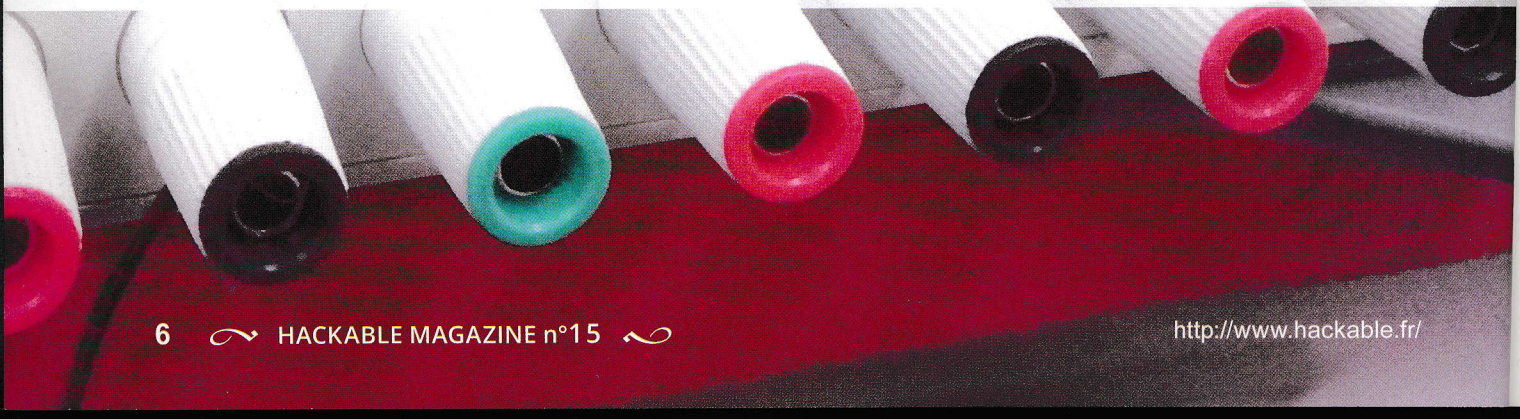
COMPRENEZ, UTILISEZ & ADMINISTREZ LINUX SUR PC, MAC & RASPBERRY PI AVEC LINUX PRATIQUE !



CONTRÔLEZ VOTRE ALIMENTATION DE LABORATOIRE AVEC VOTRE RASPBERRY PI

Denis Bodor

Dans un lointain numéro du magazine, nous avions abordé les alimentations de laboratoire, leur utilité, les critères de sélection d'un tel matériel et les fonctionnalités pouvant être intéressantes pour une utilisation en amateur. Les produits récents proposent des fonctionnalités de prise de contrôle et de mesures par un système informatique (PC, Mac, Pi, etc.) via différentes connexions, tantôt en standard, tantôt en option. La prise en charge de ces fonctionnalités est relativement simple, sans applications spécifiques, avec juste un brin de Python depuis votre Raspberry Pi...

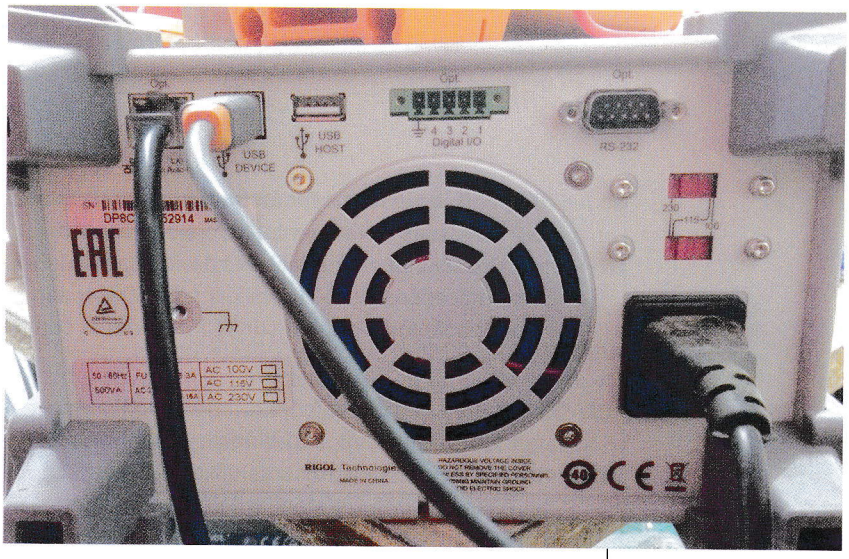


Si vous vous êtes équipé d'une alimentation de laboratoire ou d'un oscilloscope d'entrée de gamme (ou pas) récent, vous avez sans doute remarqué qu'il disposait d'un connecteur USB (type B le plus souvent) et/ou d'un connecteur Ethernet pour réseau filaire, ainsi que, peut-être d'un port série et un connecteur semblant être un croisement entre celui d'une ancienne imprimante parallèle et d'un joystick PC (vous savez, du temps où on avait des cartes son et un port parallèle).

Toute cette connectique, se trouvant généralement à l'arrière de l'appareil, est destinée à permettre la connexion avec d'autres appareils et instruments ou avec un système informatique. Chose que nous allons précisément découvrir ici.

Mais avant de commencer et rentrer dans le vif du sujet, voyons ensemble ce qui vous attend si vous tentez d'appréhender le monde de la communication entre appareils (multimètres de laboratoire, oscilloscopes, alimentation programmable, générateur de signaux, analyseur de spectre, etc.) ou entre appareils et ordinateurs ou, en d'autres termes, le monde de l'instrumentation des équipements de mesure et de laboratoire.

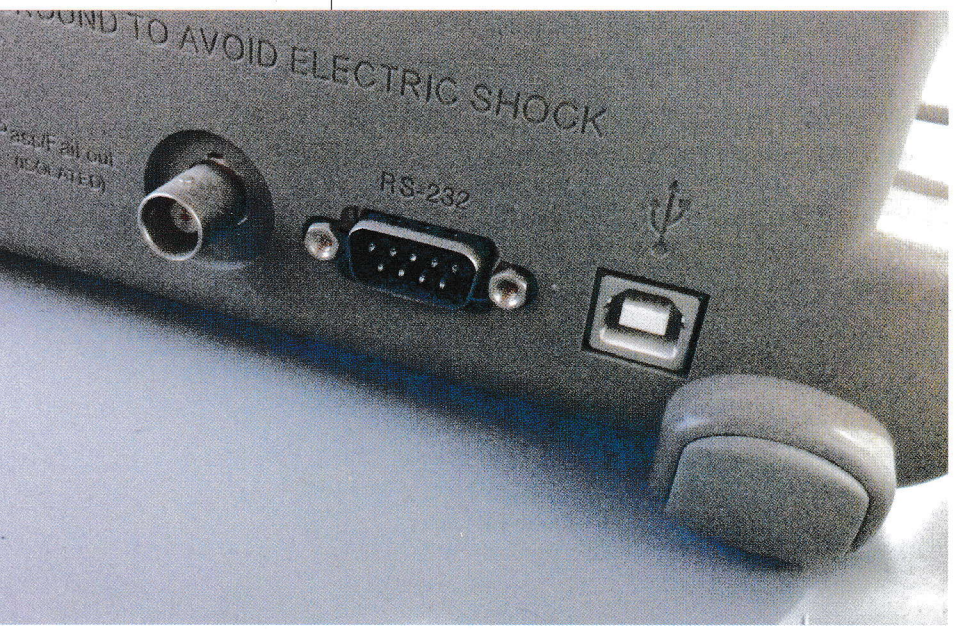
Pour faire connaissance avec un nouveau monde ou un nouveau territoire, il faut généralement avant tout apprivoiser le jargon d'usage. Voici donc celui du domaine visé :



On trouve à l'arrière des instruments de mesure de laboratoire toute une connectique permettant la liaison avec un système informatique comme une Pi ou un autre instrument. Ici l'alimentation DP832 propose l'USB en standard et le réseau, le port série et les E/S en options payantes.

- VISA (*Virtual Instrument Software Architecture*) : standard définissant les spécifications pour communiquer avec un instrument au travers d'interfaces comme GPIB ou VXI. C'est également une interface de programmation (API) de haut niveau permettant d'écrire des programmes communiquant avec les instruments. Grâce à l'API VISA, un programmeur peut écrire un logiciel sans se soucier, par exemple, du type de connexion utilisé (USB, réseau, GPIB, etc.) ou du système d'exploitation sur lequel fonctionnera son programme.
- GPIB (*General Purpose Interface Bus*) : une norme définissant le fonctionnement d'un bus de communication initialement conçu par Hewlett-Packard pour réaliser des systèmes automatisés de tests d'équipements. Cette norme, standardisée sous le nom IEEE-488 dans les années 70, définit entre autres choses un format de connecteur 24 broches qu'on retrouve encore sur de nombreux instruments. Le langage de communication utilisé sur un bus GPIB est le SCPI.
- SCPI (*Standard Commands for Programmable Instrumentation*) : définit une norme de langage permettant de contrôler un instrument de mesure tel un oscilloscope ou un fréquencemètre. La norme décrit le langage dans les grandes lignes et les constructeurs d'instruments définissent souvent leurs propres instructions. Pour simplifier les choses, les fabricants fournissent alors un pilote écrit avec l'API VISA qui permet aux programmeurs de n'avoir à se soucier ni de la connectique utilisée (GPIB, USB, réseau) ni des instructions spécifiques.

- VXI (*VME eXtensions for Instrumentation*) : standard très vaste définissant à la fois des aspects mécaniques, électriques et des protocoles de communications. VXI-11 (parfois noté VXI11) est un protocole fonctionnant sur TCP/IP pour la communication via le réseau.
- LXI (*LAN eXtensions for Instrumentation*) : un standard définissant un protocole de communication pour les instruments utilisant Ethernet (le réseau filaire). Ce standard détaille la façon d'utiliser Ethernet pour cette communication et les échanges entre instruments.
- IVI (*Interchangeable Virtual Instrument*) : pour permettre de faire communiquer les instruments LXI, le standard impose que chaque matériel doit avoir un pilote IVI. Celui-ci est créé en utilisant une interface de programmation IVI basée sur Microsoft COM, une bibliothèque C ou le framework .NET. Le fait d'utiliser un pilote IVI est censé éviter l'utilisation directe de SCPI et simplifier l'interopérabilité entre instruments et applications.
- USBTMC (*USB Test and Measurement Class*) : c'est une classe de périphériques USB. Les périphériques USB peuvent se diviser en deux catégories, ceux devant disposer d'un pilote spécifique au produit et ceux faisant partie d'une « catégorie » ou classe de produit, qui sont alors pris en charge par un pilote générique. Les souris, claviers, supports de stockage USB, webcams, imprimantes... sont tous une classe de périphériques. C'est pour cette raison que vous pouvez brancher un clavier USB sur n'importe quelle machine avec n'importe quel système et directement l'utiliser. USBTMC est également une classe de périphériques, destinée aux instruments de test et de mesure. Un périphérique conforme aux spécifications de la classe USBTMC fonctionnera donc exactement comme un autre produit également conforme, sans avoir à utiliser de pilotes USB spécifiques.



Ces connectiques, le plus souvent USB, ne sont pas spécifiques aux alimentations. Nous avons ici l'arrière d'un oscilloscope Rigol DS1052e proposant également l'USB et un port série.

Bien. Si vous avez tenu jusqu'ici, vous devez soit être totalement perdu dans le qui-fait-quoi, soit avoir un mal de tête conséquent (comme moi). Le fait est que derrière toutes ces appellations, ces standards et ces normes, se cache une véritable usine à gaz terminologique. C'est généralement ce qui arrive lorsque plusieurs constructeurs se mettent d'accord pour faire communiquer leurs produits tout en essayant de garder leurs spécificités. Il en résulte des consortiums, des alliances, des fondations et surtout des tonnes et des tonnes de spécifications parfaitement imbitables et totalement sédatives, ayant pour objectif, certes de garantir l'interopérabilité, mais aussi et surtout de vendre de la certification à tout va. L'industrie adore les clubs privés et les onéreuses cartes de membre...

Pour ma part, je préfère, au risque de simplifier outre mesure les choses et devenir inexact, envisager une approche plus pratique. Je simplifierai donc les choses ainsi :

- Les équipements de mesures répondent à un ensemble de normes et de standards, les constructeurs se sont mis d'accord sur une architecture et l'ont appelé VISA.
- Pour concrétiser cette architecture, un certain nombre de connexions sont disponibles : des ports séries, des connecteurs GPIB, de l'USB avec USBTMC et du réseau avec VXI11/LXI/IVI.
- Tout ce petit monde parle un patois précis appelé SCPI, qui semble cependant trop compliqué pour faire discuter tous les intervenants parce que chacun y a mis son grain de sel.
- Pour que les programmeurs n'aient pas à apprendre le dialecte SCPI de chaque matériel, les constructeurs fournissent des briques logicielles standardisées permettant de construire des applications plus facilement puisqu'il leur suffit de posséder la bonne brique pour parler au bon matériel.

En ce qui nous concerne, nous pouvons éliminer une partie de ces éléments : les briques logicielles sont faites pour les applications en C ou reposant sur COM ou .NET, nous allons simplement utiliser un module Python et lire la documentation du matériel.

Le standard VISA ne nous intéresse pas vraiment, tout ce que nous voulons c'est un moyen de connecter la Pi à un appareil et pour cela, il suffit généralement d'observer ses connecteurs.

Ce qui en résumé nous donne : retourner l'appareil pour regarder s'il y a un connecteur USB ou Ethernet et trouver les instructions SCPI adéquates dans la documentation du fabricant. Nous verrons en fin d'article qu'il est possible de simplifier les choses, mais ceci repose sur l'existence d'un bout de code pour votre matériel spécifique.

1. NOTRE VICTIME : L'ALIMENTATION RIGOL DP832

Rigol est un fabricant très en vogue dans le monde de l'électronique hobbyiste, car cette société produit du matériel peu cher de relative bonne facture. Les oscilloscopes DS1054Z ou le précédent modèle, le DS1052E, offrent des fonctionnalités très intéressantes tout en étant dans une tranche de prix raisonnable pour un particulier (respectivement quelques 330€ et 410€). Un équipement de laboratoire reste cependant un investissement important et il en va de même pour une alimentation programmable comme le DP832 dont nous avons déjà parlé dans le numéro 4 du magazine.

Notez que je n'ai pas d'affinités particulières avec la marque Rigol et encore moins d'arrangement. Les explications qui vont suivre peuvent être appliquées à n'importe quel équipement, alimentation ou autre, de n'importe quelle marque. Il se trouve simplement que le matériel que j'avais sélectionné au moment de remplacer mon alimentation Protek 3305-L se trouvait être le DP832 de Rigol.

Je ne vais pas m'étendre sur le sujet, mais il existe une spécificité concernant le matériel Rigol : certaines fonctionnalités sont activables en saisissant un code spécifique à chaque équipement, en fonction de son numéro de série. Le DP832 par exemple dispose matériellement de connexions série et réseau, mais ces fonctions ne sont utilisables que durant une période donnée de démonstration. Il faut ensuite acheter les options pour les débloquer définitivement. Il se trouve cependant que des informations existent sur le Web permettant d'utiliser ce mécanisme. Au dire de la plupart

des sites et vidéos YouTube concernant cet état de fait et devant l'absence de toute action de la part de Rigol, certains sont arrivés à la conclusion qu'une certaine tolérance existait chez le constructeur. Mais, même si tel était le cas et que Rigol ne s'intéresse pas aux amateurs qui de toute façon n'auraient pas acheté les options en question (voire s'en sert comme argument de vente auprès des hobbyistes), le fait d'activer une fonctionnalité sans en acquérir le droit reste une démarche que je ne peux me permettre de recommander, pour des raisons évidentes. Si vous avez vraiment besoin d'une de ces options, comme la résolution augmentée ou l'option LAN+série, dirigez-vous vers un détaillant Rigol et achetez-la.

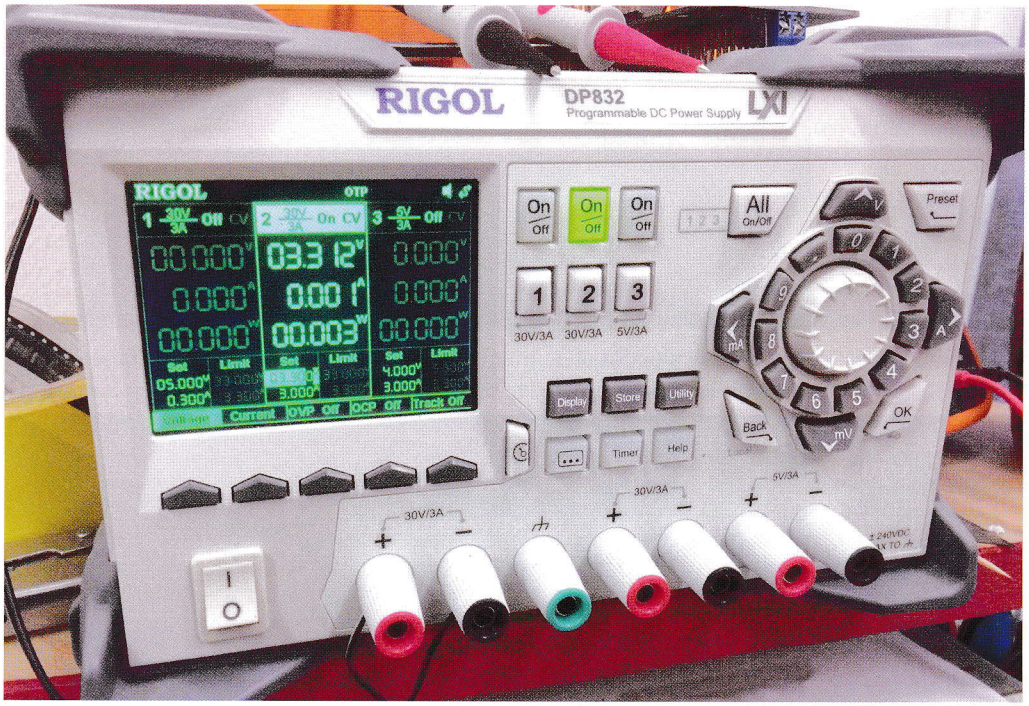
Toujours à propos de Rigol et de l'alimentation DP832, sachez que des problèmes peuvent exister concernant la communication USB avec certaines versions du logiciel intégré au matériel (*firwmare*). Mon appareil possédait à l'origine une version 1.11 du firmware et il est possible de mettre à jour vers la plus récente, 1.14 d'avril 2015 qui semble améliorer grandement la stabilité de la connectique USB. Le firmware n'est pas disponible directement au téléchargement et le constructeur ne recommande pas la mise à jour, car la procédure est risquée. Pour obtenir la mise à jour, il faudra vous rendre sur le site **rigolna.com**, trouver la fiche produit de votre appareil, aller sur l'onglet ***Software*** et cliquer sur l'icône permettant de faire une demande (***Request the Latest Firmware***). Là vous devrez indiquer vos informations personnelles, le type et modèle de matériel, son numéro de série et la version actuelle du firmware installé. Les fichiers et une documentation PDF vous seront alors normalement envoyés par mail. En ce qui me concerne, la mise à jour n'a pas posé de problème particulier.

2. PRISE EN CHARGE AVEC LA RASPBERRY PI

Avec l'alimentation Rigol DP832, l'utilisation du port USB (USBTMC) est possible par défaut, mais la connexion Ethernet (LXI) et série nécessite l'achat d'une option. Nous allons donc principalement nous pencher sur la connexion disponible gracieusement pour tous. Dès la connexion de l'alimentation à la Raspberry Pi via un câble USB type A vers type B, le système nous informe de la détection d'un périphérique (**`dmesg | tail`**) :

```
usb 1-4: new high-speed USB device number 15 using ehci-pci
usb 1-4: config 1 interface 0 altsetting 0 bulk endpoint 0x82 has invalid maxpacket 64
usb 1-4: config 1 interface 0 altsetting 0 bulk endpoint 0x3 has invalid maxpacket 64
usb 1-4: New USB device found, idVendor=1ab1, idProduct=0e11
usb 1-4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-4: Product: DP800 Serials
usb 1-4: Manufacturer: Rigol Technologies.
usb 1-4: SerialNumber: DP8C163952914
```

Nous avons ici un périphérique USB identifié par les ID vendeur et produit **`0x1ab1:0x0e11`** avec un numéro de série **`DP8C163952914`**. Par défaut, le noyau Linux de la Raspberry Pi ne supporte pas USBTMC, car le pilote (module noyau) n'est pas présent. Ceci n'est pas un problème, car le module Python que nous allons utiliser ne repose pas sur ce support, mais dialogue avec le périphérique directement. Si, dans le futur, le noyau Linux de la Pi était livré avec le pilote, la connexion devrait provoquer l'apparition d'une entrée **`usbtmc0`** dans **`/dev`**, comme c'est le cas par exemple sur PC. Une autre solution que l'attente pourrait être de recompiler un noyau pour votre Pi, avec cette fonctionnalité en module, comme détaillé dans l'article du numéro 10.



L'alimentation DP832 est, à mon sens et malgré son interface utilisateur relativement peu ergonomique au niveau du pavé numérique sur la droite, un produit fort agréable et de bonne qualité. Cela reste cependant un investissement conséquent puisqu'un tel instrument vous coûtera tout de même quelques 400€.

Pour accéder au matériel, nous allons avoir besoin d'un module développé par un groupe de programmeurs et disponible sur GitHub à l'adresse **https://github.com/python-ivi/python-usbtmc**. Ce groupe propose trois modules : **python-usbtmc** pour la communication USB, **python-vxi11** pour le réseau (VXI11/LXI) et **python-ivi**, une implémentation en Python non officielle de IVI intégrant un certain nombre de pilotes reposant sur **python-usbtmc** et **python-vxi11**.

Pour utiliser ces modules, nous procédons ainsi après installation, si nécessaire, du paquet **git** :

```
% mkdir IVI
% cd IVI
% git clone https://github.com/python-ivi/python-usbtmc.git
% git clone https://github.com/python-ivi/python-vxi11.git
% git clone https://github.com/python-ivi/python-ivi.git
% mkdir essai
% cd essai
% cp -r ../python-ivi/ivi .
% cp -r ../python-usbtmc/usbtmc .
% cp -r ../python-vxi11/vxi11 .
```

Nous nous retrouvons ainsi avec les modules copiés dans un répertoire **essai** (**~/IVI/essai**) où il nous suffira de nous placer pour faire nos essais. Ces téléchargements ne sont cependant pas suffisants. Ces modules ont besoin de NumPy que nous installerons avec : **sudo apt-get install python-numpy**.

Reste ensuite le problème de l'accès au port USB. Raspbian met à disposition un paquet **python-usb** (module **pyusb**), mais celui-ci n'est disponible qu'en version 0.4.3-1 alors que le module Python a besoin d'une version bien plus récente, comme celle disponible sur PC (1.0.0-1). Nous devons donc installer ce module séparément avec l'installeur **pip** :

L'alimentation DP832 dispose en réalité de deux ports USB. Celui de gauche nous permet de contrôler l'instrument avec une Raspberry Pi ou un ordinateur et un peu de code Python, et celui de droite permet la connexion d'une clé USB pour le stockage et l'exportation de fichiers.

```
% sudo apt-get install python-pip
% sudo pip install pyusb
Downloading/unpacking pyusb
  Downloading PyUSB-1.0.0.tar.gz (52kB): 52kB downloaded
  Running setup.py (path:/tmp/pip-build-H74ivo/pyusb/setup.py) egg_info
  for package pyusb
Installing collected packages: pyusb
  Running setup.py install for pyusb
Successfully installed pyusb
Cleaning up...
```

Nous y sommes presque. Pour l'heure, le périphérique connecté à la Pi ne peut être « écrit » qu'avec les permissions du super-utilisateur via **sudo**. Ceci n'est pas très pratique et nous obligerait à lancer Python avec ces mêmes permissions. Pour faciliter les choses et pouvoir accéder librement au périphérique USB, nous allons modifier la configuration et en particulier ajouter une règle udev.

Pour cela, nous devons créer un fichier **/etc/udev/rules.d/86-usbtmc.rules** contenant une simple ligne :

```
SUBSYSTEMS=="usb", ACTION=="add", ATTRS{idVendor}=="1ab1",
ATTRS{idProduct}=="0e11", MODE="0660", GROUP="plugdev"
```

Ceci a pour effet de préciser que le périphérique USB **0x1ab1:0x0e11**, lors de sa connexion, doit appartenir au groupe **plugdev** et que les membres de ce groupe peuvent lire et écrire sur le périphérique. Comme l'utilisateur standard de la Pi, **pi**, fait partie par défaut du groupe **plugdev**, nous n'aurons plus besoin de **sudo**. Après ajout du fichier, il ne faudra pas oublier de redémarrer udev avec **sudo /etc/init.d/udev restart**, puis débrancher/rebrancher le périphérique.

3. ACCÉDONS AU MATÉRIEL

À présent que tout est correctement installé, nous pouvons invoquer simplement Python dans le répertoire où nous avons copié les modules. Il nous suffit donc de lancer **python** qui nous accueille en mode interactif :

```
% python
Python 2.7.9 (default, Mar  8 2015, 00:52:26)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Nous pouvons alors importer le module **usbtmc**, créer un objet représentant notre matériel en spécifiant ses identifiants USB, et l'interroger en utilisant une instruction SCPI :

```
>>> import usbtmc
>>> instr =  usbtmc.Instrument(0x1ab1,0x0e11)
>>> print(instr.ask("*IDN?"))
RIGOL TECHNOLOGIES,DP832,DP8C163952914,00.01.14
```

L'appareil nous répond gentiment avec la marque, le modèle, le numéro de série et la version du firmware. La méthode **ask** que nous venons d'utiliser permet de directement envoyer des instructions SCPI à l'appareil. ***IDN?** est une instruction impliquant un retour d'information, elle se termine par un point d'interrogation et signifie *identify*. Nous demandons au matériel de s'identifier.

Certaines instructions sont communes à l'ensemble des matériels alors que d'autres sont spécifiques à un type (alimentation, oscilloscopes, etc.) et enfin certaines ne sont utilisables que par un périphérique de marque et/ou de modèle spécifique. On trouve généralement une liste exhaustive de ces instructions dans la documentation du fabricant avec, dans notre cas, le « *Programming Guide* » de Rigol destiné aux alimentations de la série DP800.

La méthode **ask** attend une réponse de la part du matériel, mais ce n'est pas le cas pour toutes les instructions SCPI que nous pouvons utiliser. Certaines sont des ordres donnés à l'appareil, comme par exemple définir la tension sur une des sorties de l'alimentation. Ces ordres n'impliquent aucune réponse et si nous utilisons **ask**, nous obtenons une erreur. Exemple :

```
>>> print(instr.ask(":SOUR1:VOLT 1.5"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
[...]
    raise USBError(_strerror(ret), ret, _libusb_errno[ret])
usb.core.USBError: [Errno 110] Operation timed out
```

« Operation timed out » nous indique que le temps d'attente pour la réponse est écoulé, mais l'ordre a bien été donné :

```
>>> print(instr.ask(":SOUR1:VOLT?"))
1.50
```

Pour émettre un tel ordre, il faut donc utiliser **write** et non **ask**, ainsi :

```
>>> instr.write(":SOUR1:VOLT 3.3")
>>> print(instr.ask(":SOUR1:VOLT?"))
3.30
```

L'alimentation DP832 dispose de trois sorties numérotées, en syntaxe SCPI, de **SOUR1** à **SOUR3**. Il s'agit là d'abréviations de *source*. Les documentations spécifient généralement cela en désignant en majuscule le terme réduit à son strict minimum, par exemple « SOURce ». Dans notre cas, les commandes suivantes sont équivalentes et, bien entendu, retournent le même résultat :

```
>>> print(instr.ask(":SOUR1:VOLT?"))
3.30
>>> print(instr.ask(":SOURCE1:VOLT?"))
3.30
```

```
>>> print(instr.ask(":SOURce1:VOLT?"))
3.30
>>> print(instr.ask(":SOURc1:VOLT?"))
3.30
```

Il en va de même pour d'autres termes, comme « VOLTage », « CURRent », « OUTPut », « MEASure » ou encore « POWEr ». La syntaxe générale SCPI repose sur une symbolique de niveau. Ainsi, dans « :SOUR1:VOLT », nous avons une racine « : », « SOUR » désignant la source concernée à un premier niveau et « VOLT » à un second niveau, car séparé de « SOUR » par un « : ». Un ordre se termine par une directive ou un argument et une demande par un point d'interrogation. Il est également possible de spécifier plusieurs instructions en les séparant par un point-virgule. Exemple :

```
>>> print(instr.ask(":SOUR1:VOLT 7.2;:SOUR1:VOLT?"))
7.20
```

Nous avons ici deux instructions. La première fixe la tension pour la sortie à 7,2 volts et la seconde lit la valeur actuellement définie. Notez que la tension est spécifiée en argument de l'ensemble de la commande, séparée par un espace. Ceci sera important par la suite.

Une autre commande utilisable pour obtenir un résultat équivalent est « APPLy », aussi bien pour procéder au réglage que pour lire les valeurs courantes :

```
>>> instr.write(":APPL CH1,5,0.3")
>>> print(instr.ask(":APPL? CH1"))
CH1:30V/3A,5.00,0.30
```

La première ligne règle ou applique un réglage sur la sortie 1 avec une tension de 5V et un courant de 0,3A et la seconde lit ces mêmes valeurs. L'utilisation de « SOURce » ou « APPLy » est une question de préférence, avec comme petit avantage pour la seconde, le fait qu'on obtient non seulement les réglages actuels, mais également les caractéristiques de la sortie en question :

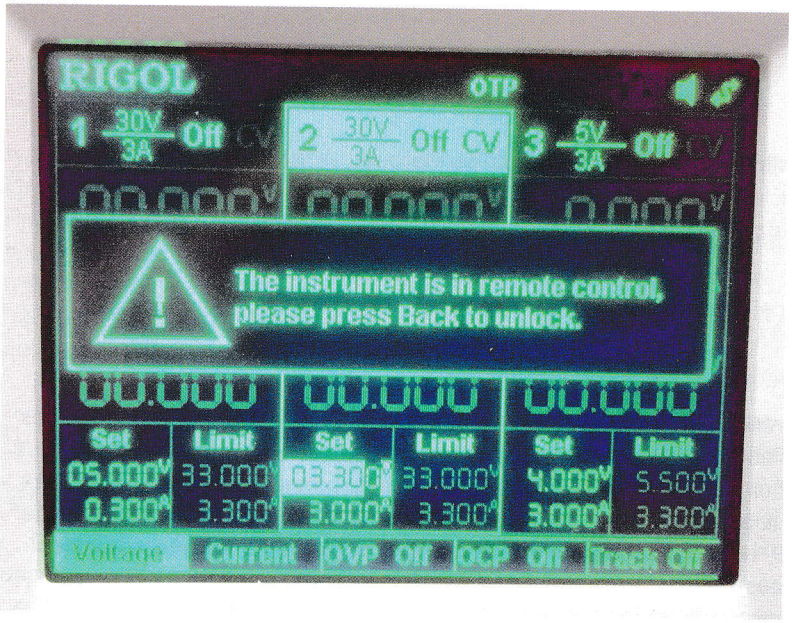
```
>>> print(instr.ask(":APPL? CH1"))
CH1:30V/3A,5.00,0.30
>>> print(instr.ask(":APPL? CH2"))
CH2:30V/3A,5.00,3.00
>>> print(instr.ask(":APPL? CH3"))
CH3:5V/3A,4.00,3.00
```

On voit ici effectivement que l'alimentation DP832 propose sur les sorties ou canaux 1 et 2, jusqu'à 30V et 3A, et sur la sortie 3 uniquement un maximum de 5V.

Il est important que comprendre que les instructions « :SOUR1:VOLT » et « :SOUR1:CURR » (pour le courant) correspondent, tout comme avec « APPL », aux réglages tels qu'on peut les ajuster sur le matériel lui-même et non de valeurs mesurées. Ainsi, si je fais :

```
>>> print(instr.ask(":SOUR1:VOLT 3.3;:SOUR1:CURR 1.2;:SOUR1:VOLT?;:SOUR1:CURR?"))
3.30;1.20
```

Je paramètre une tension de 3,3V avec un courant maximum de 1,2A et lis simplement ces deux valeurs de réglage qui sont donc configurées, mais non mesurées. Pour mesurer la tension, le courant et la puissance sur cette même sortie, il faudra utiliser :



Dès lors que l'instrument est pris en charge à distance, que ce soit en USB ou via le réseau, celui-ci cesse de pouvoir être utilisé de façon standard. Un avertissement est affiché à l'écran dès lors qu'on tente toute manipulation. Notez l'icône dans le coin supérieur droit de l'écran signifiant cet état de fait. En pressant le bouton « back », il est possible de reprendre le contrôle, mais la connexion USB deviendra alors inopérante.

```
>>> print(instr.ask(":MEAS:ALL? CH1"))
0.00,0.00,0.00
```

Bien entendu, les valeurs retournées sont à zéro, car la sortie n'est pas activée, il n'y a donc ni tension ni courant à mesurer. Remarquez la mention de « CH1 » qui peut sembler peu conforme à la syntaxe précédente. En réalité, ceci est tout à fait cohérent, car il s'agit d'un argument, exactement comme la tension spécifiée précédemment. Nous avons donc la commande **:MEAS:ALL?** et l'argument **CH1** désignant la sortie/canal (*CHannel* en anglais). Pour obtenir une mesure spécifique, nous pouvons remplacer **ALL** par :

```
>>> print(instr.ask(":MEAS:VOLT? CH1"))
0.00
>>> print(instr.ask(":MEAS:CURR? CH1"))
0.00
>>> print(instr.ask(":MEAS:POWE? CH1"))
0.00
```

et avoir respectivement, la tension, le courant ou la puissance mesurée.

Enfin, nous pouvons activer les sorties comme bon nous semble et, dans la foulée, mesurer à nouveau les valeurs :

```
>>> instr.write(":OUTP CH1,ON")
>>> print(instr.ask(":OUTP? CH1"))
ON
>>> print(instr.ask(":MEAS:ALL? CH1"))
5.00,0.00,0.00
```

OUTP ou **OUTPut** prend en argument le numéro de la sortie et l'état, pouvant être **ON** ou **OFF**.

Cette dernière commande complète cette introduction à SCPI. Comme précisé précédemment, les commandes disponibles dépendent du matériel, du constructeur et du modèle de périphérique. Un jeu de commandes bien plus conséquent sera ainsi disponible avec un oscilloscope pour la simple raison que ce type de matériel permet bien plus de réglages et de mesures. Certaines de

ces commandes sont identiques entre les modèles et les marques. Ainsi, ce qui vient d'être détaillé fonctionnera, dans les grandes lignes, avec d'autres alimentations disposant d'un port USB. Tout ce que vous avez à faire est de parcourir la documentation dédiée.

4. VOUS N'AIMEZ PAS SCPI ? PAS DE PROBLÈME, IL Y A IVI !

Pour comprendre l'intérêt de IVI, présentons les choses naturellement. Prenez l'ensemble des commandes SCPI utilisables avec un matériel et organisez-les dans un fichier. Prenez ensuite les mêmes éléments pour un autre modèle de matériel et faites de même. Enfin, inventez un ensemble de commandes qui, en spécifiant le bon matériel va automatiquement faire correspondre la ou les commandes SCPI adéquates. Peu importe le matériel utilisé, les commandes sont les mêmes du moment que vous précisez le matériel à utiliser. Bravo, vous venez de créer deux pilotes IVI pour deux matériels différents !

C'est précisément ce qu'ont fait, de façon plus technique et sérieuse, les créateurs du module **python-ivi** et ce avec un grand nombre de modèles de périphériques de marques diverses : Agilent, Tektronics, LeCroy, DiCon ou encore, bien sûr, Rigol. Nous avons donc une façon unique et unifiée de contrôler tous les matériels supportés, que ce soit en USB (**python-usbtmc**), via le réseau (**python-vxi11**), par un port série (pySerial) ou encore avec un connecteur GPIB (linux-gpib).

Ici, nous allons utiliser l'alimentation DP832 en USB comme précédemment, mais sans utiliser SCPI directement. Comprenez bien que **python-ivi** forme un ensemble de pilotes et une interface uniforme, mais que la communication effective repose tout de même sur un module dédié. Autrement dit, nous avons toujours besoin de **python-usbtmc**.

L'utilisation avec IVI débute par le chargement du module puis par la création d'un objet représentant le périphérique :

```
>>> import ivi
>>> alim = ivi.rigol.rigolDP832("USB::0x1ab1::0x0e11::DP8C163952914::INSTR")
>>> print(alim.identity.instrument_serial_number)
DP8C163952914
```

On précise les identifiants USB du matériel ainsi que le numéro de série (apparaissant avec **dmesg** juste après la connexion à la Pi). Dès lors, **alim** peut être utilisé en invoquant différentes méthodes et en affichant certains de ses attributs, comme son numéro de série, pour vérification. J'ai constaté que tantôt une erreur apparaissait à ce stade, mais une nouvelle occurrence de la commande **print()** affiche effectivement l'information et plus aucun problème ne se pose ensuite.

Comme précédemment, nous pouvons spécifier une tension et activer la sortie pour enfin procéder à la mesure effective :

```
>>> alim.outputs[1].voltage_level = 3.3
>>> alim.outputs[1].enabled = True
>>> print(alim.outputs[1].measure('voltage'))
3.31
```

Non seulement ceci est plus simple qu'en utilisant les commandes SCPI, mais ces méthodes fonctionneront exactement de la même manière avec une autre alimentation ou même un autre

type de connexion (LAN, série, GPIB). C'est en cela que réside l'intérêt d'IVI puisque vous n'avez finalement qu'un seul script Python à écrire et pouvez passer d'un matériel à l'autre en ne changeant qu'une seule ligne.

5. LE RESTE, C'EST À VOUS DE LE CRÉER

En ayant accès à un instrument depuis une Pi ou un ordinateur, il devient possible d'en faire presque totalement ce qui vous chante. Procéder à des mesures récurrentes, déporter l'affichage, créer des bancs de tests... tout ce qu'il est possible d'imaginer dès lors que vous pouvez remplacer vos interventions par un script.

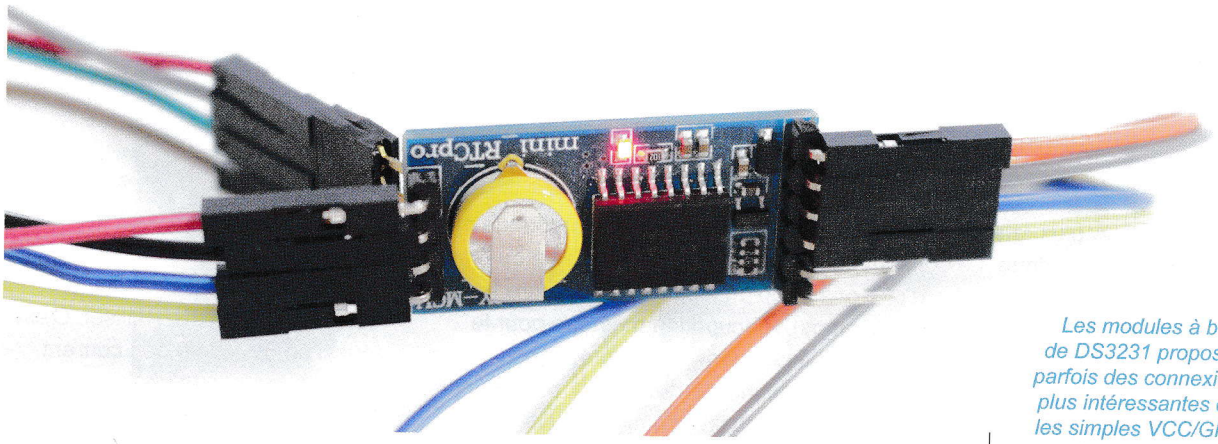
Mieux encore, grâce aux entrées/sorties de la Raspberry Pi, vous êtes maintenant en mesure de coupler ces fonctionnalités avec des montages ou la prise en charge de composants ou de modules. Ici, il s'agit d'une alimentation, mais tout ceci est parfaitement applicable à, par exemple, un oscilloscope.

Je terminerai en précisant que si vous avez du matériel à disposition et des compétences en programmation Python, votre aide sera sans le moindre doute la bienvenue. Les développeurs de Python IVI ne peuvent pas tester leur création sur tous les instruments disponibles et le nombre de matériels supportés est donc directement lié au nombre de contributeurs. Il n'est pas nécessaire de disposer de qualifications particulières en électronique pour ajouter un pilote, vous devez juste avoir le matériel sous la main, une connaissance de Python et un peu de temps à votre disposition. Je pense que c'est un projet qui mériterait un coup de pouce... DB

CRÉEZ UN ENREGISTREUR AUTONOME DE TEMPÉRATURE, D'HUMIDITÉ ET DE PRESSION

Denis Bodor

Arduino permet toutes sortes d'assemblages de capteurs permettant de remonter, en direct, tout autant d'informations sur un environnement et, par exemple, les afficher dans le moniteur série. Ceci est parfait pour des mesures immédiates ou dans un laps de temps réduit, mais pour des surveillances de plus longue durée, il faut travailler sans ordinateur et avec un support de stockage pour les données. Au programme ici : construire une telle « sonde » autonome.



Les modules à base de DS3231 proposent parfois des connexions plus intéressantes que les simples VCC/GND/SDA/SCL. Ici nous avons deux broches par signal du bus i2c nous permettant de « repartir » du module vers un autre, mais également une broche INT/SQW permettant éventuellement d'utiliser la fonction d'interruption liée à l'alarme paramétrable dans le DS3231.

Imaginez la situation : vous devez relever une série de températures et autres caractéristiques environnementales sur une période relativement longue, des heures, des jours, des semaines... Il pourra s'agir, par exemple, d'une pièce ou d'un local isolé, sans courant, ou encore d'un jardin, d'un garage, d'une serre... L'objectif est de collecter des données à intervalles réguliers, les stocker dans le montage puis, une fois la période de mesure passée, récupérer tout cela et produire un magnifique graphique (car tout le monde aime les graphiques).

Une carte Arduino basée sur un Atmel AVR dispose d'une mémoire de stockage accessible en lecture comme en écriture depuis un croquis. Nous ne pouvons, en effet pas utiliser la mémoire flash comme nous le faisons pour les chaînes de caractères via la macro `F()`, car cela ne permet qu'une lecture (seul le *bootloader* peut écrire dans la flash, car le microcontrôleur est alors dans un mode spécial). L'EEPROM intégrée au microcontrôleur, en revanche, est spécialement destinée à ce type d'usage et permet de stocker des données qui perdureront entre les différentes périodes de mise en marche.

Malheureusement, cette mémoire est bien trop petite pour nos objectifs. L'ATmega328P d'une carte Arduino Uno ou Nano, par exemple, ne dispose que de 1024 octets d'espace EEPROM. Même la carte Arduino Mega 2560 et son ATmega2560 ne proposent pas plus de 4 Ko.

Un rapide calcul nous montre que nous serons rapidement à l'étroit. Nous devons stocker une date/heure (`time_t`) et trois valeurs en virgules flottantes (`float`) pour la température, l'humidité relative et la pression atmosphérique. Chacun de ces éléments occupe 4 octets, soit un total de 16 octets par enregistrement. Avec un malheureux kilooctet d'EEPROM, nous pouvons stocker royalement une soixantaine de relevés. Avec une mesure toutes les secondes, cela nous donne tout au plus une minute et même si nous faisons des concessions en passant à une mesure toutes les 10 minutes, nous n'avons pas de quoi stocker une journée complète de mesures...

Pour régler le problème, nous devons utiliser un stockage externe. Plusieurs solutions sont à notre disposition parmi lesquelles le stockage sur carte SD et l'utilisation d'une EEPROM externe i2c. L'option SD est séduisante de par le rapport coût/volume qu'elle propose. Il faudra cependant prendre en considération le fait que le support SD occupe une taille non négligeable dans la mémoire de l'Arduino, que des problèmes de compatibilité ne sont pas à exclure, que certaines SD et microSD peuvent utiliser jusqu'à 150 mA (le maximum pour une UNO étant 200 mA) et surtout que ceci suppose des contraintes mécaniques qui ne sont pas tolérables dans tous les environnements (vibrations, etc.).

L'option de l'EEPROM externe en revanche, avec une taille pouvant atteindre 256 Ko (mais

plus raisonnablement 32 Ko) ne posera aucun problème mécanique. La consommation se limitera à quelques 2 ou 3 mA maximum lors d'une opération de lecture ou d'écriture (moins de 10µA le reste du temps), l'occupation mémoire pour le croquis restera limitée et finalement, aucun problème de compatibilité ne perturbera nos plans. Certes 32 Ko, par exemple, nous permettra de ne stocker qu'un peu plus de 2000 enregistrements, mais ceci représente tout de même quelques 30 minutes avec une cadence de mesure par seconde, plus d'une journée avec une mesure par minute et plus d'un mois avec deux enregistrements par heure... Il y a de quoi faire et rien ne vous empêchera d'utiliser plusieurs EEPROM.

Enfin, notez que la plupart des EEPROM répondent à des caractéristiques impressionnantes en termes d'endurance. Le modèle utilisé dans cet article est donné pour un million de cycles lecture/écriture et une durée de rétention de 40 ans ! La comparaison avec une carte SD est relativement difficile, dans le sens où le support intègre un contrôleur chargé de répartir physiquement les opérations dans la mémoire, mais sachez que la limite se situe généralement autour des 100000 cycles par secteur pour une carte de bonne qualité, et une rétention de données entre 5 et 10 ans.

1. COMPOSITION DE NOTRE ENREGISTREUR

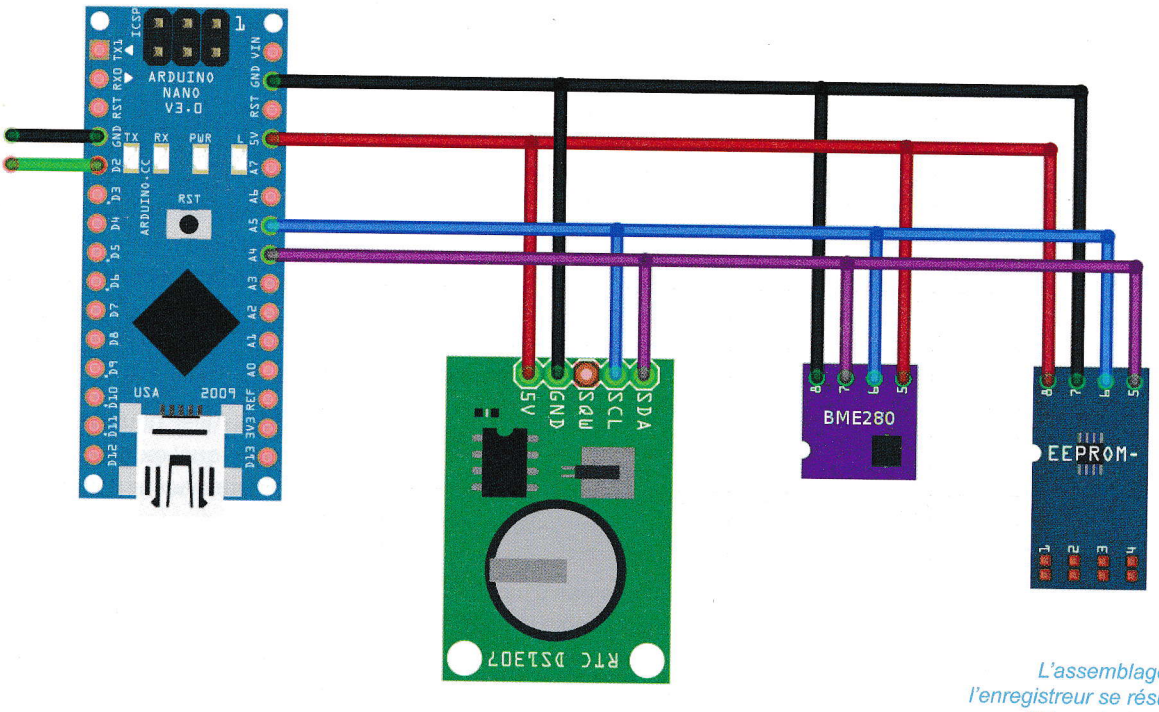
L'idée, vous l'avez compris, est ici d'utiliser une EEPROM externe de 32 Ko et plus exactement un composant nommé AT24C256 de chez Atmel. Je n'ai pas de préférence particulière pour ce modèle, il s'agit simplement d'une excellente affaire conclue sur eBay. La bibliothèque que nous allons utiliser, *extEEPROM* de *Jack Christensen*, est compatible avec de nombreux modèles (sauf rares exceptions) d'EEPROM série i2c, de tailles comprises entre 2 kbits (256 octets) et 2048 kbits (256 Ko).

Concernant la bonne affaire, j'ai acquis mes EEPROM auprès d'un vendeur appelé *echoii_mall*, mais il serait plus juste de parler de modules. En effet l'EEPROM elle-même, un composant au format DIP-8, est montée sur un circuit imprimé disposant d'un emplacement, une série de cavaliers pour choisir l'adresse de l'EEPROM, 4 connecteurs au pas de 2,54 mm, une led et quelques résistances. Le tout pour un prix, port offert, de... 1€ ! Ceci n'a rien de spécifique à ce vendeur, des offres très similaires (à quelques centimes près) pullulent sur eBay.

Ce module EEPROM est interfacé en i2c, un bus présent sur toutes les cartes Arduino et relativement facile à utiliser. C'est aussi un protocole très courant pour toutes sortes de capteurs. Nous allons donc ajouter à notre montage un module BME280 composé d'un capteur Bosch du même nom, proposant la mesure de la température, de l'humidité relative et de la pression. Chose que nous complèterons avec un autre module i2c nous fournissant une horloge temps réel (RTC), un DS3231. Ceci pourra être remplacé sans problème par un module équivalent à base de DS1307 ou de DS1338, car nous n'utiliserons ni la sonde de température intégrée dans le DS3231 (trop peu précise) ni sa fonction d'alarme (pour l'instant).

Tout cela sera connecté à une carte Arduino Nano à l'aide de deux simples broches (en plus de la masse et l'alimentation), merci le bus i2c. Ceci pourra, bien entendu être complété d'autres capteurs interfacés en i2c et je pense tout particulièrement aux modules à base de SI1145 de SiLabs permettant de mesurer les rayonnements infrarouges, ultraviolets et en lumière visible. Je n'ai malheureusement pas encore reçu ces modules et, bien évidemment, il n'est pas question de parler ici de ce que je n'ai pas eu l'occasion d'avoir entre les mains...

Les variations sont presque infinies puisque le principe même est de pouvoir connecter des données et les stocker, horodater, dans l'EEPROM. Tant que



L'assemblage de l'enregistreur se résume au câblage de tous les modules sur le bus i2c de l'Arduino. La difficulté réside davantage dans la salade de câbles potentielle qu'on obtient en pratique que dans la logique de connexion.

l'ensemble pourra fonctionner de façon autonome et sur une longue durée, tout ira pour le mieux.

À propos de durée justement, nous ne sommes pas limités uniquement par l'espace de stockage disponible, mais également par la source d'alimentation et son autonomie. Fort heureusement, une grosse demande fait chuter les prix en ce moment pour les batteries externes permettant de recharger les smartphones et tablettes (en raison, semble-t-il, d'un engouement récent pour la chasse aux créatures imaginaires à enfermer dans des boules bicolores). Inutile donc de chercher une solution complexe d'alimentation, un « pack » de ce type, fournissant 5V via un connecteur USB A fera parfaitement l'affaire. Des modèles étanches et robustes de quelques 10000mAh, relativement compacts, se trouvent un peu partout pour une trentaine d'euros (voire moins de 10€ pour les petites capacités). Avec une consommation mesurée de quelques 40mA sur le montage de test, on parle ici de quelques 10 jours de fonctionnement continu.

Au total, notre budget sera donc :

- Arduino Nano (clone) : 2,50€ ;
- capteur BME280 : 4,50€ ;
- module EEPROM AT24C256 : 1€ ;
- module RTC DS3231 : 1€ ;
- pack batterie RavPower RP-PB044 10050 mAh : 33€.

Ce qui nous donne un total de 42€, tout en considérant que la batterie pourra servir à bien d'autres projets et bien d'autres usages (voire pour recharger un smartphone si nécessaire). On parle donc ici en réalité d'un montage à moins de 10€.

2. DE L'ORDRE ET DE LA DISCIPLINE

Interconnecter tout cela n'est pas un problème, mais prendre le temps de réfléchir à la logique du croquis avant de se lancer dans l'aventure est indispensable.

Nous aurons à stocker en EEPROM des enregistrements composés d'une date/heure, une température en degrés Celsius, un taux d'humidité relative en pourcentage et une pression en hectopascals (hPa). L'espace occupé par ces différentes données dépend des valeurs retournées par les capteurs. Dans les trois cas, il s'agit d'un **float** (4 octets). Nous pourrions convertir cela en entier (**int**), voire en **byte** et réduire drastiquement l'espace occupé (de 12 à 3). Nous perdrons cependant en précision, et ce seront donc des **float**.

L'horodatage quant à lui peut sembler être un problème, en particulier si nous partons dans l'idée de vouloir stocker une chaîne comme **"30/08/2016 14:37:43"**. Heureusement, c'est là un problème qui ne date pas d'hier et une solution simple consiste à utiliser une variable de type **time_t** qui n'est rien d'autre qu'un **unsigned long**, un entier long non signé de 4 octets, destiné à contenir un nombre de secondes depuis le 1er janvier 1970 à 0 heure.

Nous avons donc 4 variables dont les valeurs doivent successivement être placées en EEPROM. Pour faciliter les choses, une approche consiste à créer une structure avec :

```
struct MesInfo {
  time_t horo;
  float temp;
  float hygro;
  float pression;
};
```

Dès lors, nous pourrons déclarer une variable avec cette structure en guise de type et y mettre des valeurs :

```
MesInfo mesdata;

mesdata.horo = 1472561252; // 30/08/2016 12:47:32
mesdata.temp = 29.45F,
mesdata.hygro = 10.2F,
mesdata.pression = 990.0F;
```

Il nous suffira ensuite d'enregistrer **mesdata** en EEPROM pour stocker, d'un coup, toutes ces valeurs. L'opération inverse se fera de la même façon en lisant les données en EEPROM en quantité suffisante (16 octets) pour remplir la structure et ensuite individuellement obtenir les valeurs.

Un problème subsiste : comment savoir où écrire dans l'EEPROM ? S'il n'était pas question de tolérer que le montage puisse être éteint tantôt, il nous suffirait d'utiliser un compteur et écrire, à chaque fois, 16 octets plus loin en EEPROM. Ceci pourrait fonctionner, mais comment, alors, récupérer nos données en fin de « campagne » de mesures ? La moindre interruption d'alimentation et le compteur est perdu...

Nous allons donc enregistrer la valeur du compteur dans l'EEPROM, à la toute première position. Ainsi, pour connaître à n'importe quel moment le nombre d'enregistrements, il nous suffira de lire les deux premiers octets de la mémoire (**int** = 16 bits = 2 octets). Avec 32 Ko d'EEPROM, nous pouvons avoir 2047 enregistrements ((2^15-2)/16) et ces deux octets seront donc suffisants.

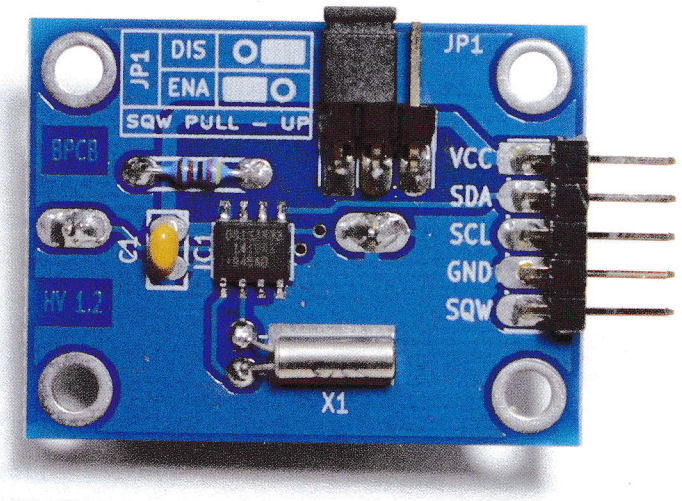
Pour une EEPROM vide de tout enregistrement, le compteur en début d'EEPROM sera donc à 0. La première écriture se fera donc après les 2 octets du compteur et s'étalera sur 16 octets, puis le compteur sera lu, incrémenté et réécrit. L'écriture suivante provoquera la lecture du compteur (qui sera à 1) et la position utilisée sera alors 2+(taille d'un enregistrement x compteur). L'écriture faite, le compteur sera incrémenté et mis à jour en EEPROM. Les écritures suivantes utiliseront le même mécanisme et le compteur en EEPROM contiendra toujours le nombre d'enregistrements, qui multiplié par 16 (taille d'un enregistrement) et additionné de 2 (la taille du compteur) nous donnera l'emplacement de la prochaine écriture possible.

Le cahier des charges général de notre croquis/montage sera le suivant :

- À intervalle régulier, nous procèderons à la relève des capteurs et à l'enregistrement des données successives.

- Cette opération ne sera possible que si une des broches de l'Arduino est mise à la masse. Nous pouvons ainsi suspendre l'enregistrement en fonction de l'état de la broche.
- Via le moniteur série, nous pourrons donner des ordres au croquis pour voir l'état du compteur, lister les données enregistrées et effacer l'EEPROM.
- L'effacement ne sera autorisé que si la fameuse broche est à la masse et donc l'enregistrement suspendu.
- Si l'enregistrement en EEPROM provoque une erreur (typiquement, une tentative d'écriture au-delà de l'espace disponible), nous ne touchons pas au compteur en EEPROM. L'erreur se répètera, mais la cohérence des données ne sera pas corrompue.

Dans le cas présent, avec l'Arduino Nano, la broche 2 se trouve juste à côté d'une broche GND (masse). En configurant cette broche en entrée avec résistance de rappel à la tension d'alimentation (`INPUT_PULLUP`), nous pouvons utiliser un cavalier pour la mettre à la masse. Le cavalier en place, l'enregistrement sera suspendu et l'effacement autorisé. Cette logique de fonctionnement nous permettra de gérer la mémoire (lecture et effacement) en connectant le montage à un PC, tout en suspendant les mesures, puis de brancher le montage à la batterie pour enfin, retirer le cavalier et autoriser les mesures.



3. UTILISATION DE L'EEPROM AVEC EXTEEPROM

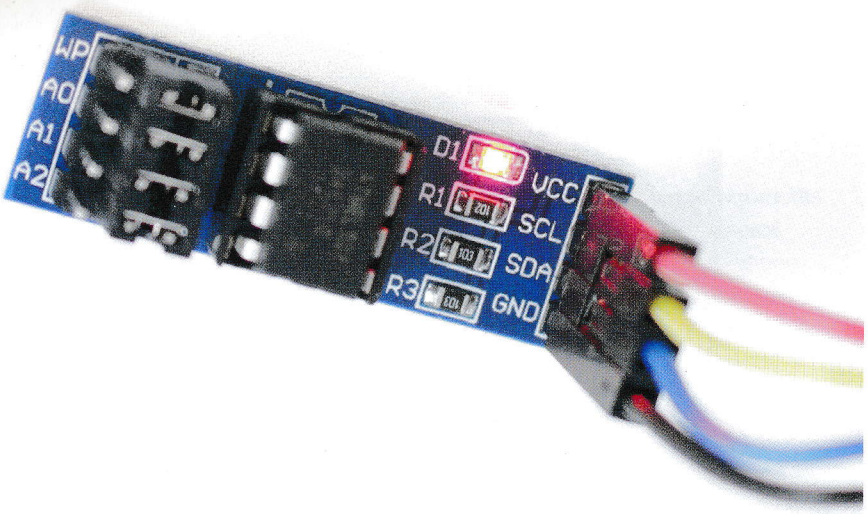
La bibliothèque extEEPROM est directement installable depuis l'environnement Arduino, via le gestionnaire de bibliothèques. On l'inclura simplement en compagnie de la bibliothèque *Wire* permettant l'utilisation du bus i2c puis on créera l'objet correspondant :

```
#include <Wire.h>
#include <extEEPROM.h>

extEEPROM eep(kbits_256, 1, 64);
```

Dès lors, nous pourrons utiliser `eep` pour accéder au composant. Les arguments utilisés sont respectivement la taille de l'EEPROM (entre `kbits_2` et `kbits_2048`), le nombre d'EEPROM présentes et la taille de page utilisée. Ce dernier paramètre n'est pas un choix, mais une caractéristique du composant qu'on trouvera dans sa documentation (*datasheet*) sous le terme *page size* ou décrit plus explicitement (« *The 128K/256K is internally organized as 256/512 pages of 64-bytes each* » dans la documentation de l'AT24C256, par exemple).

Notez qu'il est possible d'avoir plusieurs EEPROM sur le bus i2c et donc de spécifier le nombre correspondant dans la déclaration. La

Pour notre projet, un DS3231 n'est pas impératif, n'importe quelle horloge temps réel en i2c disposant d'une bibliothèque Arduino fera l'affaire. Comme ici, ce module DS1338, déclinaison 5V/3.3V du classique DS1307.



C'est incroyable ce qu'on peut trouver en ligne pour un malheureux euro. Ce module non seulement propose une EEPROM AT24C256, mais celle-ci est placée sur un support DIP-8 ce qui signifie que nous pouvons, si nécessaire, retirer l'EEPROM et la remplacer par un modèle de plus grande capacité sans le moindre problème. Notez la présence des cavaliers sur le côté permettant de configurer l'adresse de l'EEPROM sur le bus et d'interdire physiquement l'écriture.

bibliothèque part du principe qu'il s'agit d'EEPROM strictement identiques et disposant d'adresses correctement configurées avec les cavaliers et/ou les broches (A0 à A2, ou A1) du composant.

Avant d'utiliser l'EEPROM en lecture ou écriture, nous devons l'initialiser avec la méthode **begin()**, dans notre fonction **setup()** :

```
// initialisation gestion EEPROM
if (eep.begin(twiClock400kHz)) {
  Serial.print(F("extEEPROM erreur"));
  while (1);
}
```

L'argument passé spécifie la vitesse de communication entre **twiClock100kHz** et **twiClock400kHz**. Une vitesse plus importante de la communication réduira les temps de réponse, mais vous devrez vous assurer que celle-ci est compatible avec l'ensemble des composants sur le bus. De plus, Jack Christensen recommande d'initialiser l'EEPROM en dernier, après l'initialisation des autres composants sur le bus, afin de s'assurer que les autres bibliothèques utilisées ne soient pas impactées par le changement de vitesse. Une autre solution pour s'assurer d'une compatibilité est de tout simplement laisser le bus à 100 kHz.

Concernant les opérations de lecture et d'écriture, la bibliothèque extEEPROM propose différentes méthodes ou plutôt différentes utilisations de ces méthodes. Celles que nous choisirons d'utiliser ici prennent en argument l'adresse en EEPROM à utiliser, un pointeur sur les données à considérer et le nombre d'octets à traiter.

Comme nous utilisons une structure, c'est bien entendu le pointeur sur **mesdata** qui sera utilisé. Pour obtenir la taille de ces données, nous disposons de la fonction **sizeof()**, prenant en argument un type ou une expression et retournant sa taille en octets. Personnellement, j'ai une préférence pour l'utilisation d'une expression, comme **sizeof(mesdata)**, plutôt que d'un type comme **sizof(MesInfo)**, car cela limite les erreurs et risques de confusion.

Pour écrire nos données en EEPROM, nous utiliserons donc quelque chose comme :

```
eep.write(sizeof(compteur)+sizeof(mesdata)*compteur, (byte *) &mesdata,
sizeof(mesdata));
```

Ceci demande quelques explications. **compteur** contient la valeur précédemment lue à la première position de l'EEPROM et **mesdata** les informations à enregistrer. Nous précisons pour la méthode **write()** une adresse ou position d'écriture correspondant à la taille du compteur (**sizeof(compteur)**) plus la taille des données (**sizeof(mesdata)**) multipliée par le nombre d'enregistrements déjà en mémoire. Comme expliqué précédemment, ceci nous donne la position à laquelle nous pouvons écrire sans rien écraser.

Le second argument de `write()` doit être un pointeur sur la position des données à écrire. L'EEPROM ne sait gérer que des octets et non d'autres types. Il doit donc s'agir d'un pointeur sur la première position d'une série d'octets, c'est pourquoi nous *castons* avec `(byte *)` ou en d'autres termes, forçons le type de données pointées. `mesdata` est notre variable de type `MesInfo` (notre structure), or `write()` demande non pas la variable, mais l'adresse de la variable, un pointeur. Si `mesdata` est notre variable, `&mesdata` est l'adresse mémoire (SRAM) à laquelle les données se trouvent.

Enfin, `write()` a besoin de la quantité d'octets devant être écrits et, là encore, nous utilisons `sizeof()` pour le lui fournir.

Pour la lecture, cela se passera exactement de la même manière :

```
for(int i=0; i<compteur; i++) {
  eep.read(sizeof(compteur)+i*sizeof(mesdata), (byte *) &mesdata, sizeof(mesdata));
}
```

Nous bouclons sur la valeur présente dans `compteur`, précédemment lue depuis l'EEPROM, et utilisons `read()` avec les mêmes arguments. Cette fois, c'est la quantité d'octets lus dans l'EEPROM qui seront placés dans notre variable comme des octets. L'EEPROM n'a que faire du type utilisé et c'est une fois le transfert terminé que nous trouverons alors dans `mesdata` les informations souhaitées.

4. RELEVÉ DES MESURES

La relève des données des capteurs et de l'heure fournie par le module RTC est relativement simple. Il nous suffit d'utiliser les bonnes fonctions des bonnes bibliothèques :

```
#include <DS3232RTC.h>
#include <Adafruit_BME280.h>
#include <Time.h>

Adafruit_BME280 bme;
```

Celle d'Adafruit est directement récupérable via le gestionnaire de bibliothèques et celle permettant l'utilisation du DS3231 devra être installée manuellement après téléchargement sur **https://github.com/JChristensen/DS3232RTC**.

Celle-ci ne demande pas d'initialisation particulière, mais l'heure dans le composant devrait être mise à jour une première fois via l'exemple fourni avec la bibliothèque avant de pouvoir être utilisable. Bien entendu, si vous mettez en œuvre un module DS1307 ou DS1338, ou tout simplement une autre bibliothèque, vous trouverez facilement votre bonheur dans le gestionnaire de bibliothèques en cherchant simplement « RTC » et en adaptant votre croquis en conséquence.

En ce qui concerne le capteur BME280, quelques lignes dans la fonction `setup()` suffiront pour l'initialisation :

```
// initialisation capteur BME280
if (!bme.begin(0x76)){
  Serial.print(F("BME280 erreur"));
  while (1);
}
```

Les mesures et relèves seront directement faites dans une fonction **enreg()** :

```
// Enregistrement d'un relevé
void enreg() {
  // code retour de l'écriture
  int ret = 0;

  // lecture du compteur
  int compteur = 0;
  eep.read(0, (byte *) &compteur, sizeof(compteur));

  // remplissage de notre structure
  mesdata.horo=RTC.get();
  mesdata.temp=bme.readTemperature();
  mesdata.hygro=bme.readHumidity();
  mesdata.pression=bme.readPressure()/100.0F;

  // écriture en EEPROM
  ret = eep.write(sizeof(compteur)+sizeof(mesdata)*compteur, (byte *)
&mesdata, sizeof(mesdata));

  // Une erreur ?
  if(ret != 0)
    return;

  // incrémentation du compteur
  compteur++;
  // mise à jour du compteur
  eep.write(0, (byte *) &compteur, sizeof(compteur));
}
```

Celle-ci reprend les explications précédentes concernant extEEPROM ainsi que la logique de stockage en EEPROM et l'utilisation de notre structure. Notez que nous testons la valeur retournée par **eep.write()**. Si celle-ci est différente de 0 c'est qu'une erreur est survenue et très certainement une erreur sur la position d'écriture en EEPROM. Si tel est le cas, c'est qu'il n'y a plus de place et nous arrêtons simplement à cet endroit, sans mettre le compteur à jour.

5. AFFICHAGE DES DONNÉES EN EEPROM

L'affichage des données collectées et enregistrées en EEPROM se résume à une lecture du compteur et une boucle pour récupérer le contenu de tous les enregistrements disponibles. Tout ceci est alors formaté sous la forme d'une ligne par enregistrement avec les différentes valeurs séparées par des points-virgules. C'est là un format, appelé CSV, qui nous permettra de copier/coller dans un fichier ce qu'affiche le moniteur série, puis d'ouvrir ce fichier avec un tableur comme LibreOffice.

```
// fonction d'affichage des données
void dump() {
  int compteur;
```

```
    // lecture du compteur à la position 0 de l'EEPROM
    eep.read(0, (byte *) &compteur, sizeof(compteur));

    if(!compteur) {
      Serial.println("Pas de data");
      return;
    }

    // On boucle sur les enregistrements
    for(int i=0; i<compteur; i++) {
      eep.read(sizeof(compteur)+i*sizeof(mesdata), (byte *) &mesdata, sizeof(mesdata));

      // Affichage
      print2digits(day(mesdata.horo), Serial);
      Serial.print("/");
      print2digits(month(mesdata.horo), Serial);
      Serial.print("/");
      print2digits(year(mesdata.horo), Serial);
      Serial.print(" ");
      print2digits(hour(mesdata.horo), Serial);
      Serial.print(":");
      print2digits(minute(mesdata.horo), Serial);
      Serial.print(":");
      print2digits(second(mesdata.horo), Serial);
      Serial.print(";");
      Serial.print(mesdata.temp);
      Serial.print(";");
      Serial.print(mesdata.hygro);
      Serial.print(";");
      Serial.print(mesdata.pression);
      Serial.println("");
    }
}
```

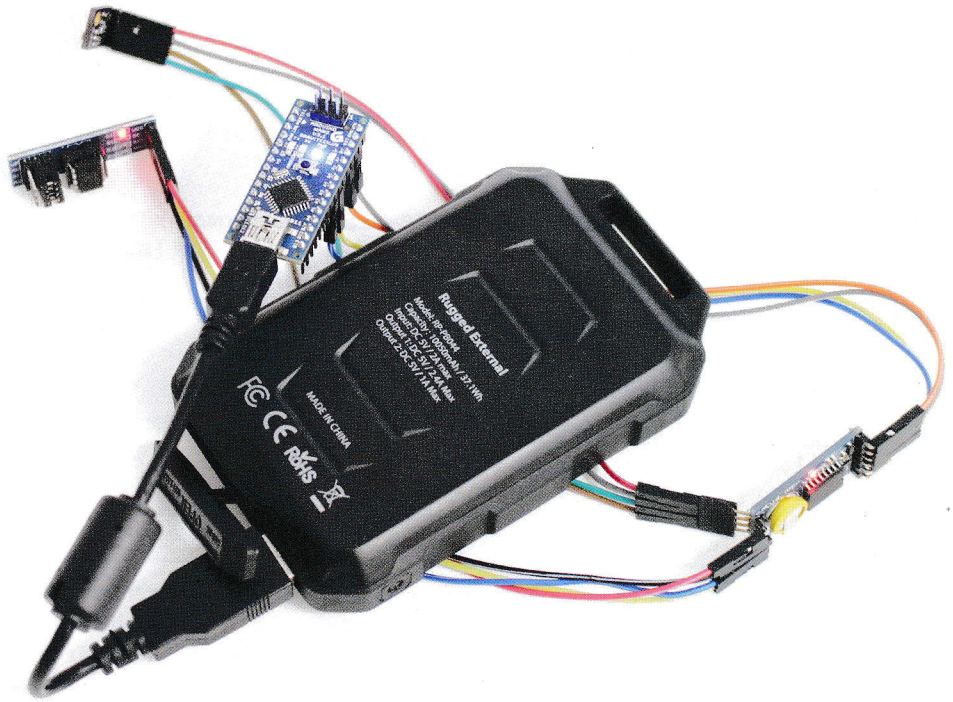
Notez l'utilisation des fonctions fournies par **Time.h**, comme **month()**, permettant d'extraire d'une variable de type **time_t** les éléments dont nous avons besoin. De plus, afin d'avoir des valeurs numériques de l'heure visuellement agréables et alignées, nous concoctons une petite fonction dédiée :

```
// Affichage d'une valeur sur deux chiffres
void print2digits(int val, Stream &sortie) {
  // si <10 alors on préfixe d'un 0
  sortie.print(val < 10 ? "0" : "");
  sortie.print(val, DEC);
}
```

6. GESTION DES COMMANDES ET DE LA COMMUNICATION

Pour permettre l'affichage de la valeur du compteur et l'effacement de l'EEPROM, nous ajoutons deux autres fonctions. La première ne fait que lire la valeur stockée en première position en EEPROM et la retourner via le port série :

```
// Affichage de la valeur du compteur
// C'est le numéro du dernier enregistrement en mémoire
void printcompteur() {
  int compteur;
  eep.read(0, (byte *) &compteur, sizeof(compteur));
  Serial.print("Valeur actuelle du compteur: ");
  Serial.println(compteur);
}
```



En version d'essai, l'ensemble n'est pas très robuste, mais cela est bien suffisant pour procéder aux tests et affiner le fonctionnement du croquis. Pour une utilisation « sur le terrain », il faudra cependant faire l'effort de remplacer tous ces câbles pour arriver à quelque chose de plus compact, soudé sur une plaque pastillée par exemple.

La seconde concerne l'effacement, qui n'en est pas vraiment un. Nous n'avons pas besoin d'écraser les données de toute l'EEPROM et adoptons exactement le même mécanisme que celui utilisé par votre PC pour son disque dur : nous nous contentons de modifier l'information qui décrit le nombre d'enregistrements présents :

```
// Effacement de l'EEROM
void reinit() {
  int compteur;
  eep.read(0, (byte *) &compteur, sizeof(compteur));
  Serial.print("Valeur actuelle du compteur: ");
  Serial.println(compteur);

  // Uniquement si broche 2 à la masse : cavalier en place
  if(!digitalRead(2)) {
```

```
    Serial.println("Effacement compteur");
    eep.write(0, (byte *) &compteur, sizeof(compteur));
    eep.read(0, (byte *) &compteur, sizeof(compteur));
    Serial.print("Valeur actuelle du compteur: ");
    Serial.println(compteur);
  } else {
    Serial.println("Effacement interdit ! Mauvais mode: mesures en cours");
    Serial.println("Placez le cavalier pour autoriser l'effacement");
  }
}
```

Notez que tout cela est conditionné par le fait que la broche 2 de l'Arduino soit mise à la masse ou en d'autres termes qu'un cavalier soit placé entre 2 et GND.

Il ne nous reste plus maintenant qu'à créer **setup()** dont la majeure partie du contenu nous est déjà familière :

```
// configuration
void setup() {
  Serial.begin(115200);
  while (!Serial) {}

  // Cavalier
  pinMode(2, INPUT_PULLUP);

  // initialisation capteur BME280
  if (!bme.begin(0x76)){
    Serial.print(F("BME280 erreur"));
    while (1);
  }

  // initialisation gestion EEPROM
  if (eep.begin(twiClock400kHz)) {
    Serial.print(F("extEEPROM erreur"));
    while (1);
  }
}
```

Et enfin, nous pencher sur la boucle principale :

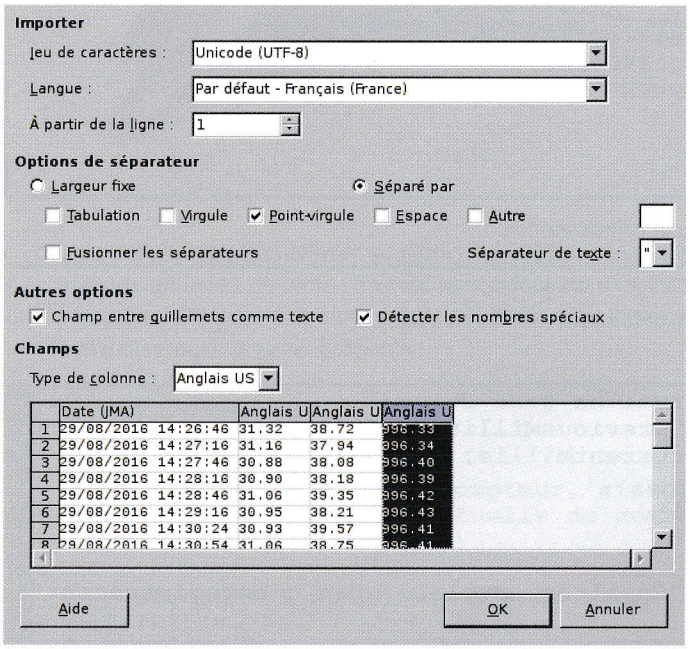
```
void loop() {
  unsigned long currentMillis = millis();
  String commande;

  // Gestion des intervalles sans delay()
  if (currentMillis - previousMillis >= 60000) {
    previousMillis = currentMillis;
    // Cavalier ?
    if(digitalRead(2))
      enreg();
    else
      Serial.println("Cavalier en place: pas d'enregistrement");
  }
```

```
  // récupération d'une chaîne via le moniteur série
  while (Serial.available() > 0) {
    commande = Serial.readStringUntil('\n');
  }

  // Analyse des commandes
  // affichage des données
  if(commande == "dump")
    dump();
  // affichage du compteur
  if(commande == "compteur")
    printcompteur();
  // Effacement EEPROM
  if(commande == "reinit")
    reinit();
}
```

La fonction **millis()** est utilisée pour gérer la fréquence des mesures et permet de ne pas bloquer l'exécution du croquis (voir exemple *BlinkWithoutDelay* dans l'environnement Arduino). La lecture du port série est facilitée par la méthode **readStringUntil()** nous permettant d'obtenir très simplement toute une ligne envoyée par le moniteur série. Il nous suffit ensuite de comparer les chaînes et d'appeler au besoin les fonctions que nous avons créées pour chaque commande.

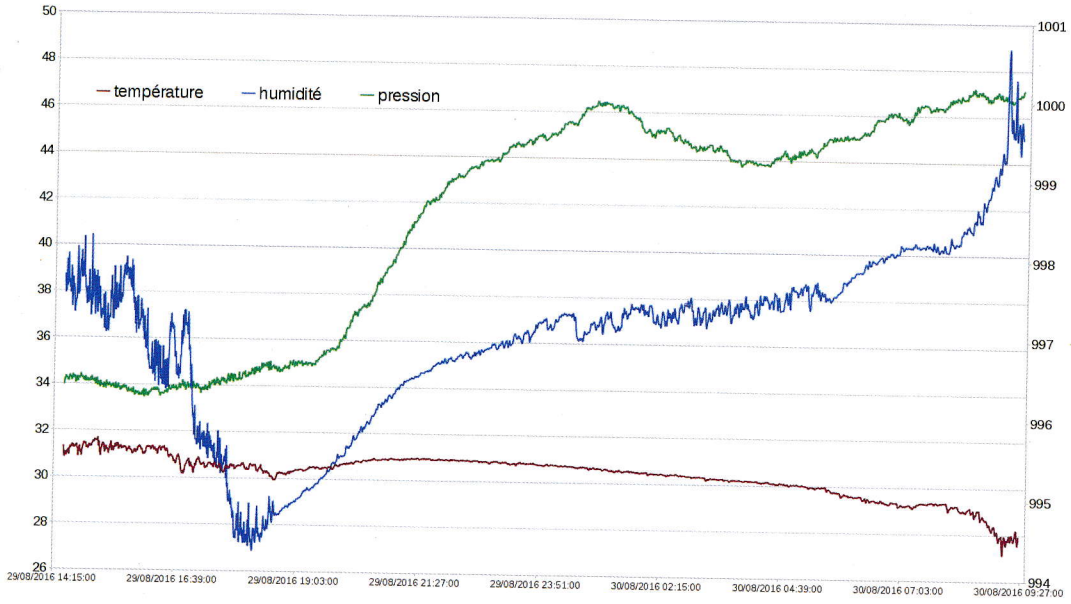
7. C'ÉTAIT LONG, MAIS ÇA POURRAIT ÊTRE PIRE

Cet article a une taille relativement conséquente malgré le fait de ne pas être entré outre mesure dans le détail de chaque morceau de croquis. L'objectif, en dehors de la construction de l'enregistreur lui-même et de l'utilisation d'une EEPROM externe était également de mettre en avant le processus de conception dans son ensemble.



L'importation des données CSV dans un tableur comme celui de LibreOffice nécessite d'ajuster et spécifier quelques paramètres comme le format de date, le séparateur de colonne ou encore le caractère utilisé pour les décimales dans les valeurs numériques.

Certains projets démarrent avec peu de choses comme un BME280 et une carte Arduino, puis s'étoffent petit à petit avant de devenir vraiment complets et riches en fonctionnalités. Il est alors nécessaire de ne pas se laisser emporter en empilant des masses de codes sans structure. Mieux vaut réfléchir posément au concept, borner les fonctionnalités et évaluer les différentes options. Ensuite et seulement ensuite, avec une idée claire de l'ensemble, on pourra sereinement commencer la conception du croquis en respectant la logique établie.



Après avoir bataillé avec LibreOffice quelques minutes, on finit par obtenir un graphique plutôt acceptable. Notez la chute du taux d'hygrométrie tout au long de la chaude après-midi avant de repartir à la hausse en même temps que la pression au cours de la nuit, ainsi que la correspondance entre la baisse de la température et le pic d'humidité au matin. Ici, l'enregistreur était posé dans un bureau non loin d'une fenêtre ouverte exposée sud-sud-ouest.

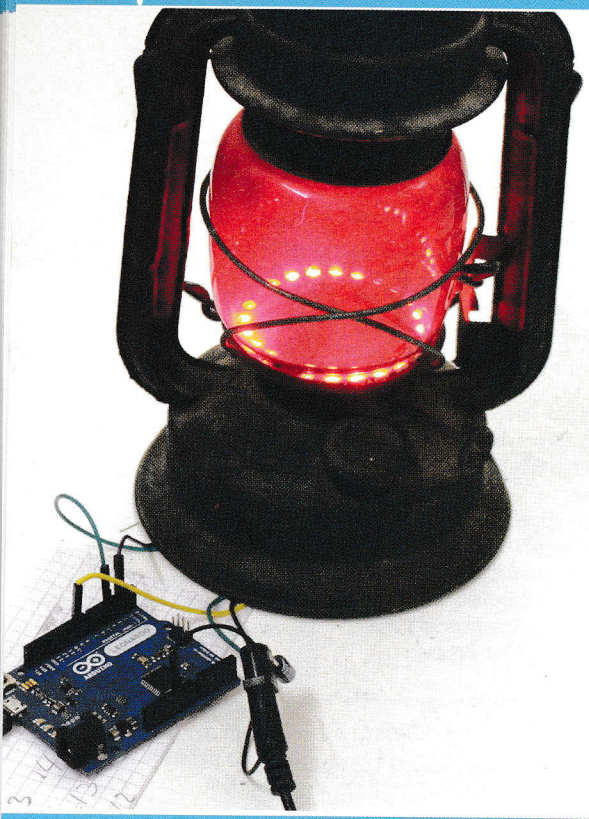
Il ne faudra pas, non plus, hésiter à prendre le temps de remettre en question le concept et si nécessaire, changer de cap et revoir l'ensemble du croquis, même s'il est déjà fonctionnel. Enfin, il est également important de garder à l'esprit dès le départ les possibilités d'évolutions légitimes et ne pas poser de barrières qui plus tard limiteront nos modifications. Ici, ajouter un capteur est relativement aisé, car nous utilisons une structure et des tailles déterminées dynamiquement. Si nous avions stocké les données une par une, il en serait tout autrement... Intégrer le déclenchement d'une action, en revanche, sortira du cadre du projet et demandera une refonte quasi-complète, car ceci n'a jamais été initialement prévu (pour moi, c'est même un projet et un montage totalement différent).

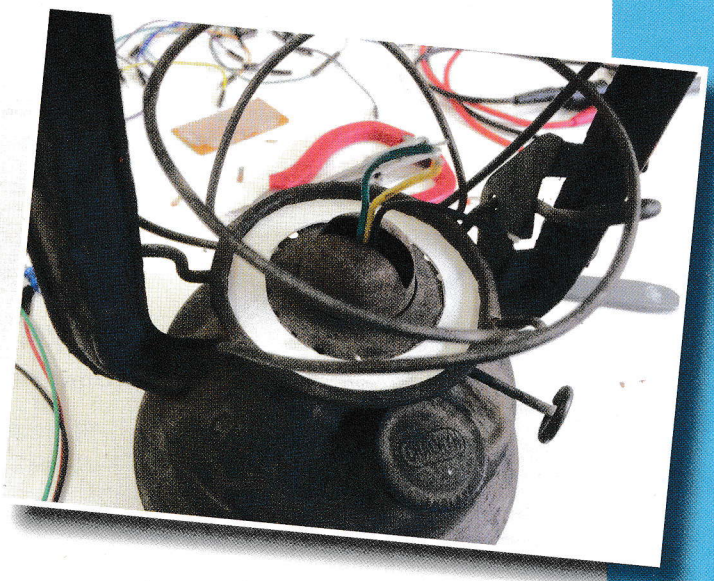
Parmi les évolutions possibles que j'envisage d'ores et déjà, nous avons l'ajout d'un capteur lumière/UV/IR SI1145 (qui est présentement en chemin depuis Shenzhen), mais également d'un module INA219, interfacé en i2c et permettant la mesure d'une tension et d'un courant. Ceci, en remplaçant la batterie par des piles (ou en bricolant le pack de batterie), permettra de déterminer l'autonomie du montage et cesser les mesures si le niveau est trop faible. Nous pourrons également inclure au besoin ces données dans les enregistrements pour suivre la consommation.

Une autre option sera d'ajouter tout de même un support pour carte SD. Non pour le stockage en temps réel des mesures, mais pour facilement exporter les données de l'EEPROM vers une SD, via une simple pression sur un bouton par exemple. La carte pourra ensuite être lue directement sur un PC pour générer un graphique. Il est également possible d'envisager une copie récurrente de l'EEPROM vers la SD après un certain temps ou un certain nombre d'enregistrements, et ce avant de vider l'EEPROM pour poursuivre les mesures.

Enfin, et c'est la raison principale du choix du module DS3231, une refonte du croquis permettra d'utiliser l'alarme du composant pour réveiller l'Arduino qu'on aura mis en sommeil après une mesure. Ceci pourra grandement augmenter l'autonomie de l'ensemble fonctionnant sur batterie, mais, là encore, sort du cadre du présent article.

Le croquis final de ce projet, présenté ici en petits morceaux, est disponible en téléchargement sur le GitHub du magazine (**https://github.com/Hackable-magazine**), très certainement dans une version plus avancée ou complète. **DB**





CRÉEZ UNE LANTERNE QUI RÉAGIT AU TOUCHER

Denis Bodor

Je suis un grand amateur de marchés aux puces. Lorsqu'on s'y balade, non en voyant les choses pour ce qu'elles sont, mais pour ce qu'elles peuvent être ou devenir, c'est un véritable festival et il faut alors littéralement se retenir d'acheter à tout va, car le temps de réalisation est forcément limité. Dans le cas présent, une vieille lanterne a attiré ma convoitise et est devenue mienne pour trois malheureux euros. L'objectif : en faire un objet de décoration intégrant des leds dont le comportement sera contrôlé par le toucher...

NIVEAU	TEMPS	BUDGET
	10 minutes	40 €

La première surprise en rentrant chez moi avec la lanterne (et plein d'autres choses) dans mon cabas fut de découvrir l'affaire que je venais de faire. Apparemment cette marque de lanternes est très prisée des collectionneurs et/ou des amateurs de décorations vintages. Ce modèle à huile de paraffine, le Chalwyn Tropic, se négocie entre 30€ et 60€ en fonction de l'état de conservation de l'objet. De plus, à en juger par le résidu de suie couvrant le tout et le verre teinté rouge, il semblerait que ma lanterne ait été utilisée dans le domaine ferroviaire. En cherchant un peu sur le Web, il s'avère que Chalwyn était, dans les années 40, un fabricant anglais réputé pour quatre modèles de lanternes, *Far East*, *Tempest*, *Pilot* et *Tropic*, ce dernier étant utilisé par l'armée anglaise.

Il n'est pas possible de magiquement connaître l'histoire de l'objet ainsi arrivé entre mes mains, mais je caresse l'idée que celui-ci accompagnait peut-être les troupes anglaises de libération en Alsace et aura poursuivi sa carrière dans une gare locale. Une chose est sûre, nous avons donc là quelque chose de plus de 70 ans, qui mérite bien plus que de traîner dans un grenier ou sur l'étal d'un marché aux puces. Les puristes diront sans doute qu'une rénovation en bonne et due forme est de mise, mais je n'ai aucun usage d'un tel risque d'incendie probable. Je préfère catapulter cette lanterne dans le 21ème siècle et lui donner une seconde vie plutôt que d'en faire un objet de musée ou un éclairage d'appoint qui ne sera jamais utilisé.

Le but ici est donc d'agrémenter cette lanterne d'un nouveau système d'éclairage à base de leds tout en conservant son esthétique d'origine. De plus, je tiens à conserver un aspect mystérieux au résultat final, voire « magique ». L'éclairage de la lanterne ne sera donc pas contrôlé par un interrupteur, un bouton poussoir ou toutes autres choses trop visibles, mais par le simple fait de toucher sa surface métallique.

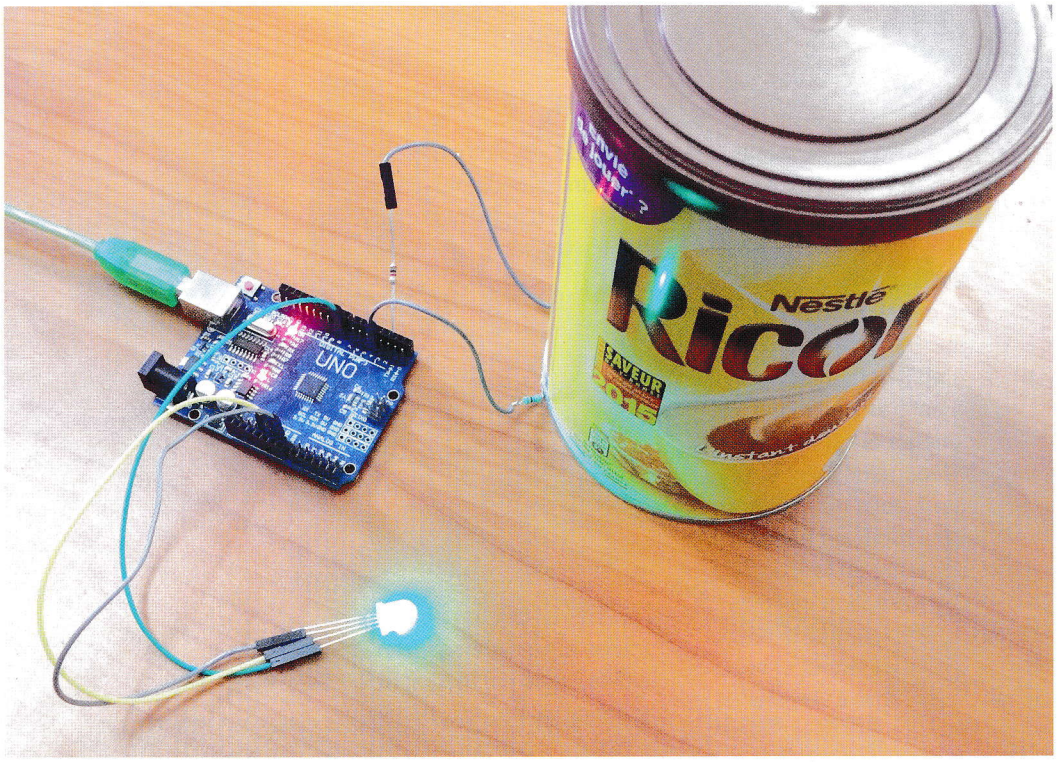
Cet objet a 70 ans et ne méritait certainement pas d'être abandonné honteusement sur l'étal d'un marché aux puces. Heureusement, je suis passé par là et le promets à une nouvelle vie dans le monde moderne, après un petit rafraîchissement électronique...

LE PRINCIPE

Nous voulons intégrer des leds dans la lanterne et allons ici opter pour les fameuses leds « intelligentes » WS2812b également appelées tantôt NeoPixels. L'idée est de pouvoir illuminer correctement l'intérieur de la lanterne et créer des effets intéressants en jouant sur la couleur et/ou l'intensité de chaque led avec un minimum de connectique vers la carte Arduino. Les leds WS2812b se prêtent parfaitement à cet usage et, coup de chance, il existe un montage en anneau composé de 24 leds d'une taille s'ajustant parfaitement à la base du verre de la lanterne.

La forme et la taille de l'assemblage de leds est totalement dépendant de ce à quoi il sera adapté, de l'effet recherché et de la quantité de lumière souhaitée. Ici, c'est un anneau de 24 leds, mais des déclinaisons en 16 et 12 leds existent. Il est également possible d'utiliser ou d'assembler individuellement les leds en quantité plus ou moins importante. Encore une fois, tout dépend du résultat visé.

Mais l'éclairage en lui-même n'est pas ici l'élément le plus intéressant ou le plus important. Nous



Avant de travailler sur l'objet avec lequel il n'est pas question de se rater, il est de bon ton de faire quelques essais. Ici une boîte vide de Ricoré fera parfaitement l'affaire afin de faire connaissance avec les subtilités de la détection capacitive.

partons dans l'idée de ne pas trop modifier physiquement l'aspect de la lanterne (au point de la laisser un peu crasseuse) et éliminons donc d'office l'option consistant à ajouter un horrible bouton. Nous allons plutôt utiliser une propriété physique de notre corps : sa capacité électrique.

Vous ne le savez peut-être pas, mais dans l'ensemble vous êtes un grand condensateur. Je ne dis pas ça pour être méchant, c'est juste une réalité physique, puisque votre corps est capable d'accumuler des charges électriques. C'est précisément cette caractéristique qui est utilisée par les écrans tactiles modernes qui fonctionnent parfaitement bien avec votre doigt, mais pas avec un stylet en plastique (écrans résistifs).

Votre smartphone va donc mesurer en permanence la capacité à la surface de l'écran. Il en va de même pour certains boutons tactiles non mécaniques (interrupteurs, plaques vitrocéramique, etc.). Cette technologie présente de nombreux avantages parmi lesquels l'absence d'usure, la résistance aux intempéries, l'intégration ou encore la robustesse ou la liberté de design.

Le principe de fonctionnement, tel que nous allons le mettre en œuvre est fort simple : on fait passer un courant au travers une résistance de grande valeur et on observe la tension à sa sortie qui est également connectée à une surface conductrice. En l'absence de capacité et dans un monde parfait, dès lors qu'on appliquera un courant, on mesurera une différence de potentiel à la sortie de la résistance (qui est forcément reliée vers la masse puisqu'on mesure son état). Mais le monde n'est pas parfait et une certaine capacité négligeable est déjà présente.

Cependant, en touchant ou en nous approchant de la surface conductrice, nous allons faire intervenir notre propre capacité dans le circuit. Qui dit résistance et condensateur dit réseau RC et un délai proportionnel à la capacité va être introduit. En d'autres termes, en appliquant une tension en entrée de la résistance, le changement en sortie ne sera pas immédiat. Plus la capacité sera grande, plus le délai sera long. Nous pouvons donc mesurer la présence d'un corps humain à l'aide d'une sortie de l'Arduino, une résistance et une entrée.

CE QU'IL VOUS FAUT

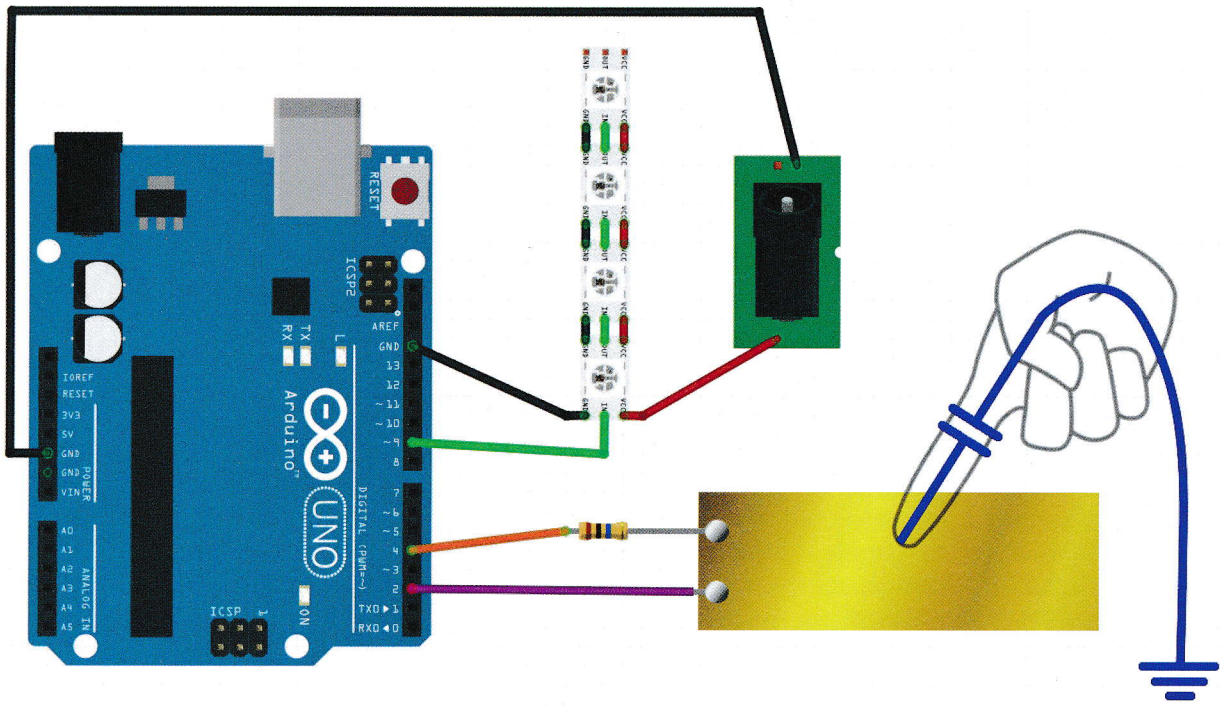
- Une carte Arduino. N'importe quelle carte en 5V fera l'affaire (UNO, Nano, Micro, Mega, etc.). Les entrées/sorties en œuvre sont numériques et utilisées par paire pour chaque surface de contact. Nous n'avons ici besoin que de deux broches pour cet usage. Le fait que la carte fonctionne en 5V n'a aucun rapport avec la détection, mais concerne la compatibilité avec les WS2812b.

- Des leds WS2812b. Il y a plus d'une façon de faire en fonction de l'effet à obtenir. Dans ce cas précis, j'utilise un anneau de 24 leds WS2812b prenant place tout autour du support de verre de la lanterne. Selon votre projet, une seule WS2812b peut suffire ou encore simplement une ou plusieurs leds standards contrôlées par un MOSFET. L'idée est de simplement éclairer votre objet en fonction des détections de contact qui sont faites.

- Une résistance entre 100 Kohms et 50 Mohms. Comme détaillé précédemment, le principe consiste à connecter une résistance de forte valeur entre deux broches de la carte Arduino. Celle-ci représente le paramètre fixe du circuit par opposition à la capacité qui est variable et dépendra de nombreux facteurs (distance, surface de contact, conductivité de la peau, etc.). J'utilise ici une résistance de 10 Mohms formant un bon compromis. Plus la résistance sera importante, plus le courant sera limité et donc, plus le délai de chargement du condensateur (vous) sera important. Le choix de la valeur de résistance finale dépendra du résultat obtenu lors des essais avec votre surface de détection et donc de l'objet métallique utilisé.

- Un objet métallique comme une lanterne ou n'importe quelle autre forme (boîte de Ricoré, papier alu, conserve, morceau de cuivre...). Ce paramètre est fixe ou variable selon votre projet. Ici, l'objectif est de rendre sensitive la lanterne, il n'est donc pas question d'utiliser autre chose. Si vous comptez simplement expérimenter avec la sensibilité au toucher, un morceau de métal fera l'affaire. Notez que cela ne fonctionnera qu'avec des conducteurs, mais que ceux-ci peuvent être nus, peints, vernis ou sales. La distance entre vous et le conducteur, constituée par une fine pellicule diélectrique, forme également un condensateur avec une capacité qui se combinera à la vôtre.

- Une alimentation adaptée pour les leds. Ceci dépend totalement du nombre et du type de leds utilisées. Une ou deux WS2812b pourront parfaitement être alimentées directement par une carte Arduino, ainsi qu'une dizaine de leds standards et un MOSFET. Si en revanche vous souhaitez plus de luminosité, comme moi, et optez pour un nombre conséquent de leds, il faudra les alimenter de façon distincte avec un courant suffisant. Un bloc d'alimentation dédié sera alors indispensable. 24 WS2812b contenant chacune 3 leds nous donne un peu moins de 1,5A en 5V lorsque toutes les leds sont allumées. Un bloc d'alimentation comme ceux pour smartphone fera l'affaire.



LE MONTAGE

Physiquement, le montage est relativement simple. Nous avons d'une part la connexion de la série de leds WS2812b reliée à la broche 9 de la carte et d'autre part la surface tactile connectée à la broche 4 via une résistance de 10 Mohms et à la broche 2 directement. Le choix des broches que ce soit pour les leds ou la surface conductive est totalement arbitraire.

Je n'ai ici représenté que 4 WS2812b en série, mais le principe est exactement le même quel que soit leur nombre. Pour rappel, ce sympathique composant se compose de 4 broches : alimentation +5V, masse, données en entrée et données en sortie. La couleur RVB que doit prendre chaque led est déterminée par une succession d'états sur la broche de données en entrée. S'il y a plus de données que celles permettant de contrôler une led, l'ensemble des informations est « poussé » sur les leds suivantes via la sortie de données. Une seule broche de l'Arduino est donc suffisante pour contrôler un nombre arbitraire de leds WS2812b.

L'anneau de 24 leds suit exactement le même schéma de connexions en série et propose les mêmes broches qu'une WS2812b seule, y compris la sortie de données puisqu'il est ainsi possible de chaîner plusieurs anneaux si nécessaire.

Le principe de fonctionnement de la bibliothèque que nous allons utiliser pour la détection capacitive consiste à mettre à l'état haut la broche 4 puis chronométrer le temps nécessaire pour lire ce même état sur la broche 2. En plaçant notre doigt en contact avec la surface métallique, nous ajoutons un condensateur comme représenté sur le schéma. Ce faisant, le délai de lecture s'en trouvera prolongé et le croquis dans l'Arduino saura qu'un contact est établi. La résistance de 10 Mohms réduisant le courant circulant devrait être suffisante pour la plupart des usages et objets. En augmentant sa valeur, nous serons en mesure de détecter de plus faibles capacités, ce qui résulte généralement en une détection à une distance plus importante. Cependant, comme vous le verrez plus loin, c'est quelque chose qui sera également pris en charge et ajusté dans le croquis.

LE CROQUIS

Fichier Édition Croquis Outils Aide

```
#include <WS2812FX.h>
#include <CapacitiveSensor.h>
#include <EEPROM.h>
#include <avr/wdt.h>

#define NBRLED 24
#define LEDP 9

WS2812FX ws2812fx = WS2812FX(NBRLED, LEDP, NEO_GRB + NEO_KHZ800);

CapacitiveSensor lanterne = CapacitiveSensor(4,2);

unsigned long preumsMillis = 0;
int compteur;
uint8_t mode;

void setup(){
   pinMode(LED_BUILTIN, OUTPUT);

   lanterne.set_CS_AutocaL_Millis(5000);
   Serial.begin(115200);

   // lecture du dernier mode en eeprom
   EEPROM.get(0, mode);

   // initialisation leds
   ws2812fx.init();
   ws2812fx.setBrightness(255);
   ws2812fx.setSpeed(500);
   ws2812fx.setColor(255, 0, 0);
   // donnée en eeprom valide ?
   if(mode <  MODE_COUNT)
     ws2812fx.setMode(mode);
   else
     ws2812fx.setMode(FX_MODE_STATIC);
   ws2812fx.start();
}

void loop(){
    long val =  lanterne.capacitiveSensor(15);

    unsigned long currentMillis = millis();

    ws2812fx.service();

    if(compteur) {
```

```
        // 2s de passées ?
        if(currentMillis - preumsMillis >= 2000)  {
          // oui, on prend en compte le compteur
          Serial.println(compteur);
          switch(compteur) {
            case 1:
                 mode = FX_MODE_STATIC;
                 break;
            case 2:
                 mode = FX_MODE_FADE;
                 break;
            case 3:
                 mode = FX_MODE_FIREWORKS;
                 break;
            case 4:
                 mode = FX_MODE_SPARKLE;
                 break;
            case 5:
                 mode = FX_MODE_RUNNING_LIGHTS;
                 break;
            default:
                 // bloqué !
                 wdt_enable(WDTO_15MS);
                 while(1) {};
          }
          ws2812fx.setMode(mode);
          EEPROM.put(0, mode);
          // réinitialisation
          compteur=0;
        } else {
          // dans la seconde
          if(val > 6000) {
            Serial.println(val);
            digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
            compteur++;
            delay(180); // debounce
          }
        }
      } else {
        // compteur à 0
        if(val > 6000) {
          // premier contact
          Serial.println(val);
          preumsMillis = currentMillis;
          compteur++;
          digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
          delay(180); // debounce
        }
      }
}
```

Arduino

À PROPOS DU CROQUIS

Le premier point important de ce croquis concerne la bibliothèque *CapacitiveSensor* de Paul Badger. Celle-ci est disponible via le gestionnaire de bibliothèques de l'environnement Arduino et s'installera donc très facilement. Pour l'utiliser, il vous suffit d'inclure la bibliothèque et de créer un objet représentant la surface, en spécifiant les deux broches utilisées : celle en sortie où est branchée la résistance et celle en entrée connectée directement à la surface conductrice.

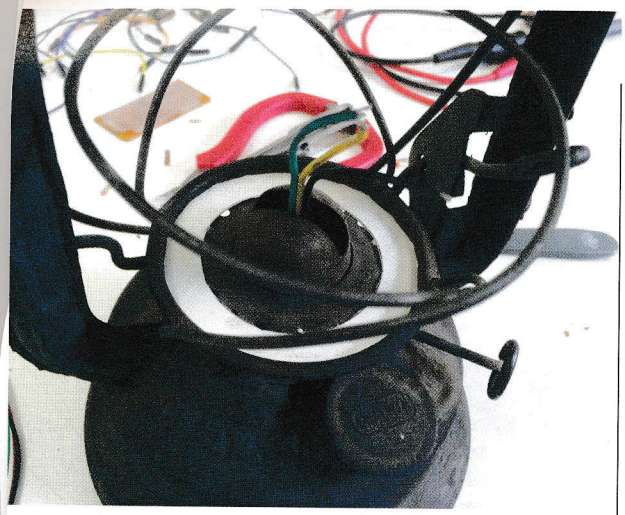
Le réservoir de la lanterne sera l'emplacement idéal de la carte Arduino et, dans un premier temps, le seul moyen de faire passer le câblage discrètement. 15 petites minutes de travail au disque diamanté et beaucoup de bruit plus tard, et l'affaire est dans le sac.

Notre utilisation de cette bibliothèque est relativement noyée dans le reste du croquis. Voici donc une version simplifiée de son utilisation, tirée de l'exemple qui l'accompagne :

```
#include <CapacitiveSensor.h>

CapacitiveSensor surface = CapacitiveSensor(4,2);

void setup(){
   Serial.begin(115200);
}

void loop(){
    long start = millis();
    long valeur =  surface.capacitiveSensor(30);

    Serial.print(millis() - start);
    Serial.print("\t");
    Serial.print(valeur);

    delay(10);
}
```

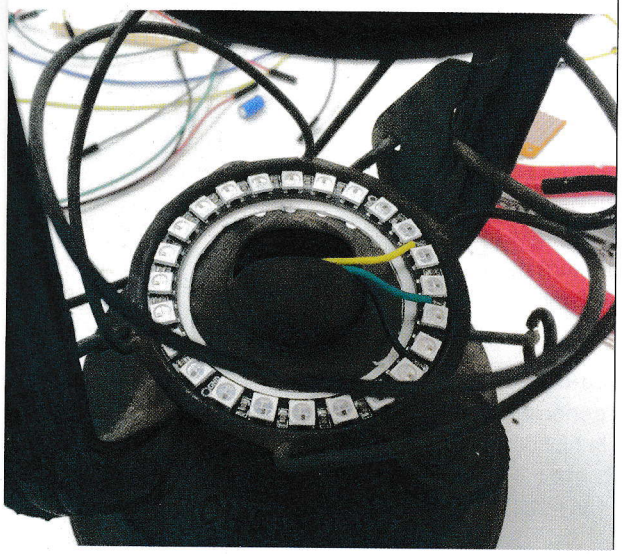
La valeur que nous récupérons dans **valeur** est totalement arbitraire et ne correspond à aucune unité de mesure particulière. Celle-ci est fournie, à chaque tour dans **loop()**, par la méthode **capacitiveSensor()** prenant en argument le nombre d'échantillons mesurés. Plus ce nombre est grand, plus la mesure sera précise, mais plus elle prendra de temps. Je vous invite à faire l'essai de ce mini croquis afin de comprendre la relation entre cet argument et les valeurs obtenues avec votre surface.

Pour notre projet, nous ne voulons pas simplement une valeur limite au-delà de laquelle allumer la lanterne. L'autre bibliothèque utilisée, *WS2812FX*, permet de générer différents effets avec les leds WS5812b. *WS2812FX*, de Harm Aldick, repose sur la bibliothèque *NeoPixel* d'Adafruit. Cette dernière est installable via le gestionnaire de bibliothèques, mais *WS2812FX* devra être récupéré sur **https://github.com/kitesurfer1404/WS2812FX** puis installé à la main.



Seulement trois câbles sont nécessaires pour contrôler l'anneau de leds WS2812b : une masse, une alimentation 5V et un signal permettant de contrôler individuellement la couleur et l'intensité lumineuse de chaque led. Un anneau en carton est ici ajouté afin d'éviter tout problème de contact entre le circuit et le métal de la lanterne.

Un véritable coup de chance : l'anneau de leds a une taille parfaitement adaptée au support du verre de la lanterne. Il a simplement été nécessaire de dessouder les trois connecteurs et d'acrobatiquement les ressouder une fois l'anneau en place. Pas besoin de colle, le verre est directement pressé contre les leds à l'aide d'un mécanisme à ressort dans le haut de la lanterne.



WS2812FX propose quelques 44 effets différents et nous l'avons déjà utilisé dans le numéro 13 pour le projet de réglette lumineuse. L'idée est ici de compter le nombre de contacts consécutifs entre un doigt et la lanterne et d'utiliser ce nombre pour choisir un effet à appliquer aux leds. La simple valeur retournée par `capacitiveSensor()` n'est donc pas suffisante et nous devons construire une logique spécifique pour pouvoir compter le nombre de dépassements d'une valeur limite (ici 6000) durant une tranche de temps donnée.

La logique est la suivante :

- On démarre avec un compteur de contact à zéro.
- Au premier contact détecté, lorsqu'une mesure est supérieure à 6000, on incrémente le compteur, qui passe à 1 et on sauvegarde la valeur de `millis()`.
- Lors d'un nouveau contact et si le compteur n'est pas à zéro, on incrémente uniquement si le délai écoulé depuis le premier contact n'est pas supérieur à 2 secondes. Ceci aura pour effet de comptabiliser tous les contacts dans la période de 2 secondes.
- Si le délai de 2 secondes est écoulé, on prend en compte le décompte de contact pour appliquer l'effet correspondant et on repasse la valeur du compteur à zéro, prêt pour une nouvelle séquence. On stocke également le numéro de l'effet correspondant dans l'EEPROM interne de l'ARDUINO de manière à ce qu'en cas de réinitialisation ou de coupure de l'alimentation, on revienne automatiquement au dernier effet utilisé.
- Si le nombre de contacts ne correspond pas à un effet, c'est soit que l'utilisateur (nous) a tapoté plus qu'il ne faut, soit qu'un problème est survenu et nous redémarrons la carte Arduino.

Ce dernier point découle d'une constatation durant mes essais. Il semblerait qu'en fonction de certaines conditions que je ne suis pas arrivé à clairement déterminer, la capacité détectée se mettait à fluctuer de façon incontrôlée. Pour régler le problème, j'ai donc décidé de prendre en compte cette éventualité en partant du principe que si le compte de contact ne correspond pas à quelque chose d'attendu, c'est que le problème s'est manifesté.

Un *reset* de la carte Arduino semble régler le problème en attendant que je puisse explorer ce phénomène qui semble aléatoire. Il existe plusieurs façons de réinitialiser le microcontrôleur AVR d'une carte Arduino :

- Connecter une sortie à la broche « RESET » de la carte avec un condensateur et mettre cette ligne à la masse pour forcer une réinitialisation. Cette approche consomme une broche supplémentaire de la carte et n'est pas vraiment une solution 100% logicielle.

- Utiliser une instruction en assembleur pour faire un saut à l'adresse 0 de la mémoire flash où se trouve le début du programme en mémoire. Voilà quelque chose qui est purement logiciel, mais qui ne correspond pas à une véritable réinitialisation car, par exemple, les entrées/sorties ne se retrouvent pas dans un état identique à celui d'une véritable initialisation.
- Utiliser le *watchdog* intégré dans le microcontrôleur. Il s'agit d'une fonctionnalité permettant justement de détecter et régler un problème comme un gel du croquis ou une boucle sans fin non souhaitée. Le principe consiste à avoir, en tâche de fond, un décompte qui, s'il arrive à 0, réinitialise le microcontrôleur. Il faut donc régulièrement signaler au *watchdog* que le programme est toujours actif pour éviter une réinitialisation.

C'est donc cette dernière solution qui est utilisée ici de façon un peu détournée puisqu'on met en route le *watchdog* avec `wdt_enable(WDTO_15MS)` juste avant de partir dans une boucle infinie (`while(1) {}`). 15 millisecondes plus tard, le *watchdog* estimant que le programme est bloqué, le microcontrôleur est automatiquement redémarré proprement. Nous obtenons notre reset forcé.

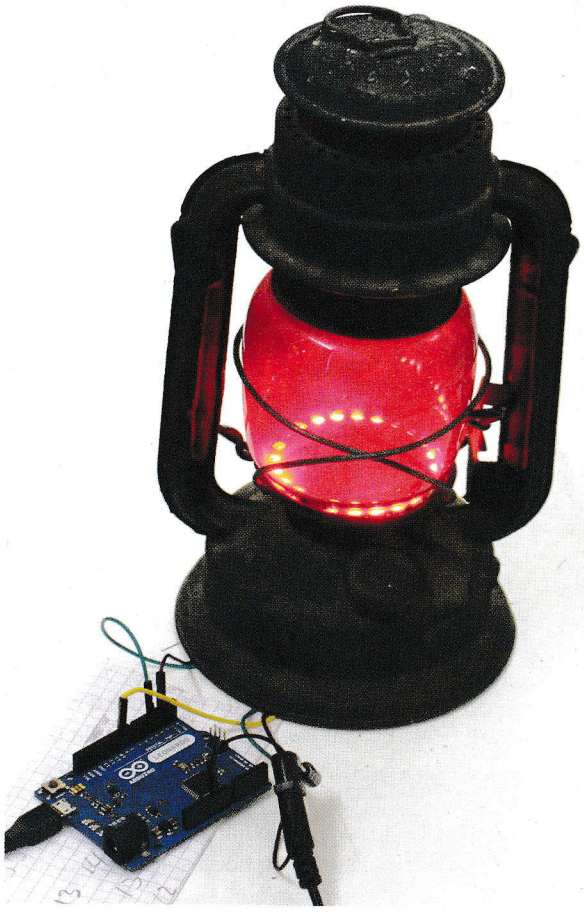
En ce qui concerne l'enregistrement en EEPROM, nous utilisons simplement les fonctions `EEPROM.put()` et `EEPROM.get()` permettant respectivement de stocker une valeur et la lire. Les deux paramètres passés en argument de ces fonctions sont l'adresse concernée en EEPROM et le nom de la variable contenant la valeur à utiliser.

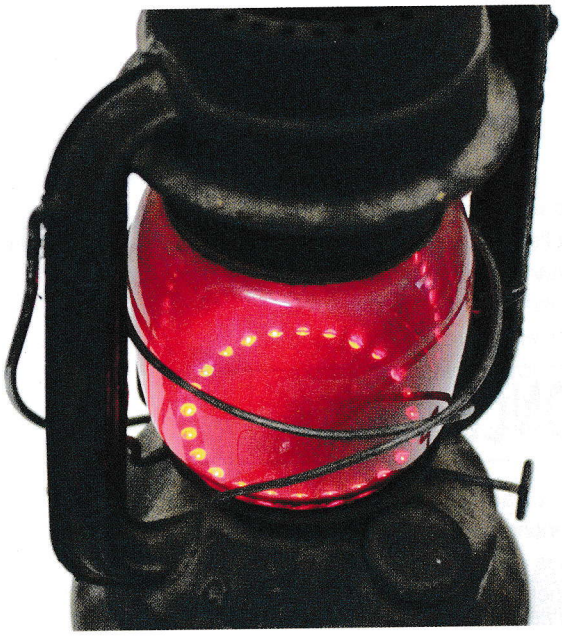
CONCLUSION

Nous arrivons au terme de cet article qui finalement ne présente pas vraiment de difficultés en termes de programmation. La partie délicate en revanche réside dans la réalisation pratique, et ce avec la connexion de la surface aux broches de la carte. Souder un câble ou une résistance à une surface métallique nécessite généralement un fer à souder et/ou une station de soudage de bonne qualité. En effet, la soudure elle-même consiste à faire fondre de l'étain sur les parties à souder, ce qui implique une certaine température. Si votre fer chinois d'entrée de gamme n'est pas capable de faire chauffer le support aussi vite qu'il dissipe la chaleur, vous risquez de passer un moment fort pénible. Je vous recommande également de traiter la surface au papier de verre et surtout d'utiliser du flux, qui une fois chauffé par la panne du fer, va nettoyer la surface, éliminer les traces d'oxyde et faciliter l'opération. Poncez/grattez/limez, mettez du flux, chauffez la surface et appliquez une quantité généreuse d'étain. Étamez également le câble ou la patte du composant et ensuite seulement, soudez.

Vous remarquerez que le croquis présenté ici définit une couleur pour

Une fois l'assemblage terminé, c'est l'heure des réglages et le moment d'affiner le croquis de façon à trouver et ajuster au mieux les différents effets proposés par la bibliothèque WS2812FX. Le fait que le verre soit rouge limite grandement les possibilités en termes de couleur, mais on peut se rattraper sur l'aspect animation. À terme, la carte Leonardo sera remplacée par une Arduino Nano, placée dans le réservoir.





Ce projet était une excellente occasion de jouer avec la bibliothèque CapacitiveSensor, mais une autre voie possible aurait été d'utiliser la molette de réglage permettant initialement d'ajuster la hauteur de la mèche. Une fois reliée à un commutateur rotatif intégré dans la lanterne, la molette pourrait permettre le choix de l'effet et la mise en route de la lanterne.

les leds avec `ws2812fx.setColor(255, 0, 0)` et je n'ai utilisé que des effets monochromes parmi ceux proposés par *WS-2812FX*. Le verre rouge équipant ma lanterne interdit presque totalement l'utilisation d'une autre couleur tant le rendu obtenu est fade dans le cas contraire. Peut-être aurais-je l'occasion de mettre la main sur une autre lanterne de ce genre avec un verre transparent permettant d'exploiter toutes les couleurs des leds. Si tel est le cas, je pense que j'en profiterai pour dépolir la surface intérieure avec un produit qu'on trouve généralement dans les magasins de loisirs créatifs, destiné à la gravure chimique à l'aide d'un pochoir en vinyle.

Enfin, si vous rencontrez des problèmes de stabilité concernant la détection des contacts, la première chose à vérifier c'est la mise à la terre. Lorsque vous programmez votre carte Arduino, le câble USB possède un blindage généralement relié d'une façon ou d'une autre au châssis du PC, lui-même connecté à la terre. Sauf, bien entendu, si vous utilisez un portable fonctionnant sur batterie. La mesure de la capacité dépend fortement de la connexion à la masse de la carte Arduino. En fonctionnement autonome (sans PC), cette référence doit être conservée pour que les mesures soient stables. Une solution peut être de relier la masse de la carte à la terre, une autre consiste à utiliser un large support conducteur connecté à la terre et placé sous l'objet, mais isolé de ce dernier.

Dans le même ordre d'idée, il peut être intéressant de protéger l'entrée de l'Arduino (la broche sans la résistance) à l'aide d'une résistance de 1 Kohm environ. Ceci limitera le risque d'endommager l'entrée de la carte si vous touchez la surface en étant chargé d'électricité statique (cas typique des semelles en caoutchouc sur la moquette, ou du pull en synthétique sur les cheveux).

Pour améliorer la stabilité, l'auteur de la bibliothèque *CapacitiveSensor* recommande également d'éventuellement ajouter un condensateur entre 20 et 400 pF en parallèle à la capacité à détecter (vous). En d'autres termes, ajouter un condensateur entre l'entrée et la terre/masse. Je n'ai cependant pas testé cette amélioration puisque dans l'ensemble les valeurs lues étaient cohérentes (en dehors du fameux problème nécessitant un redémarrage). Notez également que l'environnement joue un rôle important. Une bouteille d'eau placée à proximité du montage, par exemple, perturbe les mesures en introduisant une capacité parasite.

Gardez à l'esprit que cette détection capacitive reste, bien qu'utilisable, relativement simpliste. Depuis quelques années, il existe des circuits intégrés spécialisés pour ce type d'applications et on retrouve ces technologies directement embarquées dans les microcontrôleurs. C'est le cas, par exemple, de l'Atmel ATmega328PB intégrant la technologie QTouch sous la forme d'un périphérique dédié, le *QTouch Peripheral Touch Controller*.

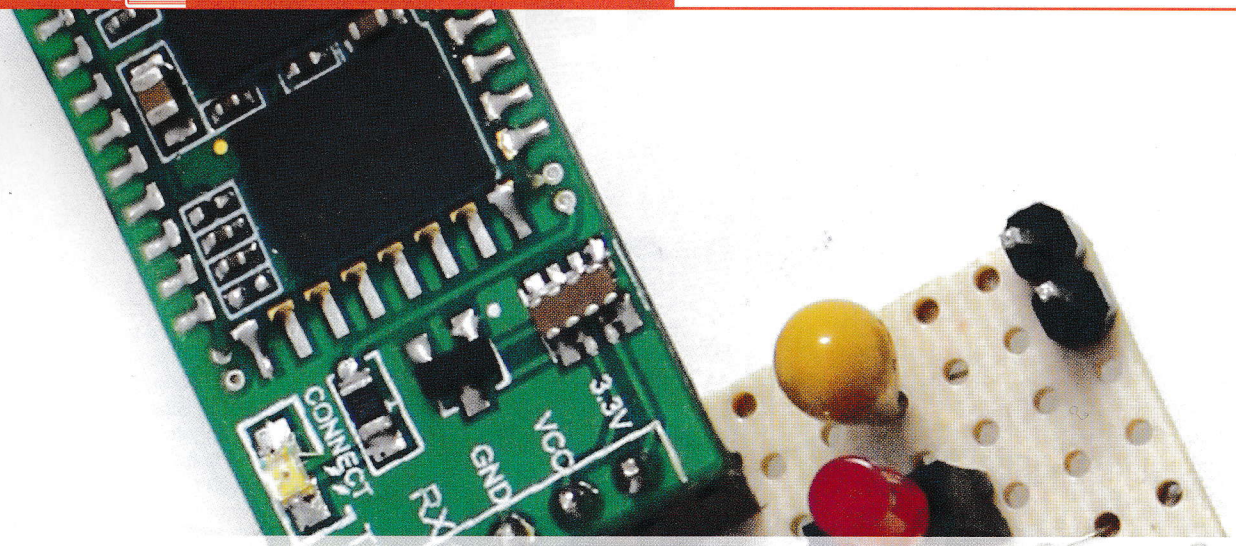
Enfin, bien que la bibliothèque *CapacitiveSensor* soit très populaire et présente l'avantage de pouvoir être utilisée sur des broches standards, il faut savoir qu'une autre approche est possible via l'utilisation des entrées analogiques. Trois bibliothèques explorent cette voie : *QTouchADCArduino* de *Jens Geisler*, *AnalogTouch* de *NicoHood* et *ADCTouch* de *Martin Pittermann*. Si vous n'arrivez pas à un résultat acceptable avec la méthode présentée ici, n'hésitez pas à tenter ces autres solutions. **DB**

IL EST DE RETOUR
CHEZ VOTRE MARCHAND DE JOURNAUX !

HACKABLE HORS-SÉRIE N°1

VOUS UTILISEZ DÉJÀ LA RASPBERRY PI ? ...OU PAS METTEZ-VOUS À L'ARDUINO !

À PARTIR DU 1ER DÉCEMBRE
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
http://www.ed-diamond.com



CONNECTEZ VOTRE ARDUINO EN BLUETOOTH : CONFIGURATION DU MODULE

Denis Bodor

Le Bluetooth est partout, du PC portable au smartphone en passant par les montres connectées et même les vieux téléphones mobiles. Mais c'est également une technologie relativement simple à mettre en œuvre dans un projet Arduino et une excellente solution pour faire communiquer différents modules et cartes. Voyons ensemble comment ajouter rapidement une telle connexion Bluetooth pour obtenir une communication série sans fil...

Lorsqu'il s'agit de créer une communication sans fil entre un montage quelconque et un « système plus intelligent » (PC, Mac, smartphone, Raspberry Pi, etc.), les choix possibles sont légion : Wifi, modules radio (APC220, nRF24, etc.), ZigBee... Chaque technologie possède ses avantages et ses inconvénients en termes de facilité de mise en œuvre, de consommation d'énergie, de portée, de vitesse ou encore de coût. Sans oublier un autre aspect en ce qui nous concerne : l'omniprésence de la technologie.

Là, il ne reste finalement plus que deux options : le Wifi et le Bluetooth. La première est séduisante puisque qui dit « Wifi » qui forcément « réseau » et donc, d'une manière ou d'une autre, une connexion au reste du monde, que ce soit le vôtre (LAN) ou le grand monde d'Internet. En revanche, ajouter du Wifi à un projet Arduino, ou même du réseau filaire Ethernet, implique le fait de jongler avec des protocoles, des bibliothèques et des concepts parfois délicats à appréhender. Même en remplaçant la carte Arduino par un ESP8266, dont nous avons parlé dans le numéro 7 et qui se programme directement via l'environnement Arduino, il faut tout de même assimiler quelques concepts (HTTP, MQTT, IP, DNS, SSDP, etc.) avant d'arriver à obtenir un résultat, une communication sous forme de question/réponse.

Dans le cas du Bluetooth en revanche, les choses sont bien plus simples à notre échelle. Certes, il s'agit d'une pile de protocoles comprenant bien des subtilités, mais tout ceci peut être facilement masqué pour arriver à l'essentiel : communiquer avec votre montage exactement comme vous le feriez avec le moniteur série de l'IDE Arduino. En effet, avec les modules très courants dont nous allons parler, il n'y a strictement rien à ajouter côté croquis pour bénéficier du Bluetooth et ajouter cette connectivité dans vos projets.

IMPORTANT !

BLUETOOTH, LE, BLE, CLASSIC, 4.0, 2.1, EDR...

Afin de vous éviter au maximum une crise de nerfs, il est une chose qu'il est capital de comprendre : le Bluetooth est un standard et il y a des versions à ce standard.

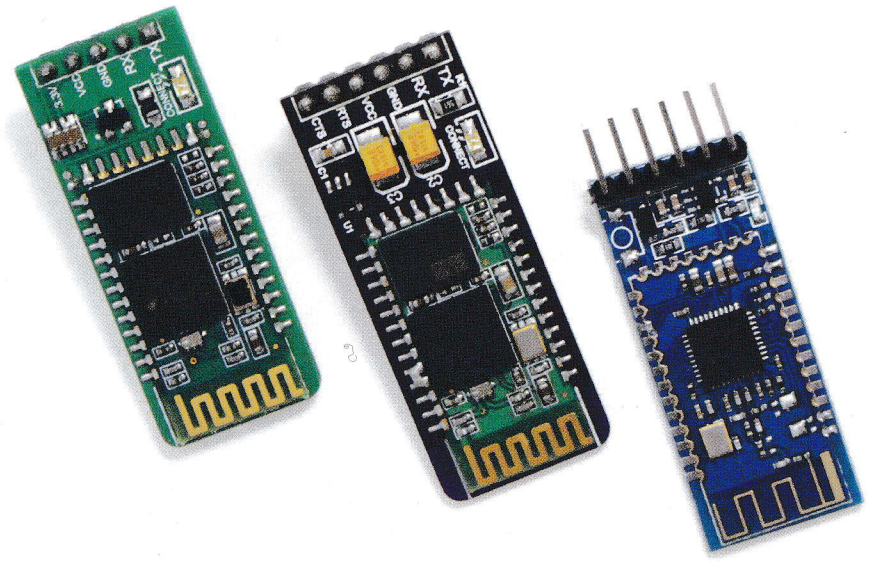
Mais plus important encore « Bluetooth » ne veut pas dire « Bluetooth LE » (alias BLE, « Bluetooth Smart » ou encore « Bluetooth Low Energy ») !

La version 4.0 du standard, rendue publique en 2010, inclut les protocoles *Classic Bluetooth*, *Bluetooth Low Energy* (réduit en « BLE » ou « Bluetooth LE », et commercialement désigné par « Bluetooth Smart ») et *Bluetooth High Speed*. Ceci signifie que le standard 4.0 (et ultérieur) détaille le fonctionnement de ces protocoles, mais ne les impose pas. Autrement dit, ce n'est pas parce qu'un périphérique ou un smartphone est Bluetooth 4.0 qu'il est forcément compatible *Bluetooth Low Energy* ou *Classic Bluetooth*. Ce qui est juste, en revanche, c'est de dire qu'un périphérique *Bluetooth Low Energy* est forcément compatible Bluetooth 4.0 ou supérieur (puisque c'est à partir de cette version du standard que le protocole a vu le jour).

Ainsi, si votre smartphone est capable de « voir » à la fois les périphériques Bluetooth un peu anciens et le dernier cri en Bluetooth LE, c'est parce qu'il supporte à la fois *Classic Bluetooth* et *Bluetooth Low Energy*. En revanche, un appareil ne supportant que le Bluetooth 3.0 ou le Bluetooth 2.1+EDR sera totalement incapable de voir un périphérique *Bluetooth Low Energy*. Il en va de même pour un appareil effectivement Bluetooth 4.0 ou supérieur, mais non compatible *Bluetooth Low Energy*.

En résumé, nous avons d'une part le *Classic Bluetooth* et toutes les versions du standard d'avant 4.0, et d'autre part le *Bluetooth Low Energy*. Ces deux mondes sont incompatibles au niveau le plus bas et ne peuvent communiquer entre eux.

Dans cet article, nous parlons de Bluetooth et donc, pour les périphériques Bluetooth 4.0 ou supérieurs, de *Classic Bluetooth*. Non de *Bluetooth Low Energy*, de BLE, ou de Bluetooth Smart.



Voici différents modules Bluetooth tels qu'on peut en trouver sur de nombreux sites de vente. À gauche, un module HC-05 sur un support intégrant un régulateur de tension, au centre un modèle similaire, mais plus ancien comprenant des broches RTS/CTS pour le contrôle de flux et à droite un module Bluetooth Smart HM-10 de piètre qualité.

1. LES MODULES BLUETOOTH

Une simple recherche de « Bluetooth module » ou « Bluetooth Arduino » sur un site comme eBay, Banggood, Tindie, DealExtreme (DX) ou Alibaba, et vous serez submergé d'offres en tous genres vous proposant des modules à quelques 3€ ou 4€ vous promettant monts et merveilles pour ajouter du Bluetooth à vos projets. Ceux-ci se présentent généralement sous deux formes : un circuit minuscule possédant de nombreux connecteurs sur trois côtés et une version plus utilisable intégrant ce même circuit, mais soudé sur un support proposant un connecteur au pas de 2,54mm avec entre 4 et 6 broches.

Il va de soi qu'il vaut mieux éviter la première déclinaison, car très délicate à mettre en œuvre et pas vraiment moins chère. En plus de servir de support de connexion, le circuit imprimé de la version avec connecteur intègre généralement un régulateur de tension ainsi qu'une led accompagnée d'une résistance. Au cœur du module, souvent désigné sous le nom de HC-05, se trouve une puce Bluecore 4 de *Cambridge Silicon Radio* (CSR), alimentée en 1,8V. Le circuit intègre, en plus de ce composant une mémoire flash ainsi qu'un autre régulateur de tension. De ce fait, le module de base, sur son support peut être alimenté directement en 5 volts par une carte Arduino, par exemple.

Attention cependant, en fonction du modèle que vous allez acquérir, il pourra être nécessaire d'utiliser des résistances lors de la connexion à la carte Arduino. En effet, même si le module peut être alimenté en 5V, la communication elle-même se fait avec des tensions 0/3,3V. Parlant de communication, ces modules utilisent tous une communication série. Nous avons donc toujours au minimum 4 connexions :

- Vcc : la tension d'alimentation,
- GND : la masse,
- RX : la broche de réception à connecter au TX d'une carte Arduino,
- TX : la broche d'émission à connecter au RX.

À ce stade, une certaine confusion peut régner concernant votre achat. Soit les résistances permettant l'adaptation de tension sont directement présentes sur le support, soit il faudra les prévoir séparément. Certains modèles de modules comprennent également un circuit intégré chargé de la conversion de tension, mais ils sont relativement rares (et plus chers).

Ce n'est pas tout, comme nous allons le voir un peu plus loin, ces modules disposent de deux modes de fonctionnement. Nous avons d'une part le mode de configuration (ou mode commande) et de l'autre celui de fonctionnement normal. Le premier mode permet de définir les paramètres de communication du module (vitesse, code, nom, etc.) alors que le second se contente de servir de relais entre la com-

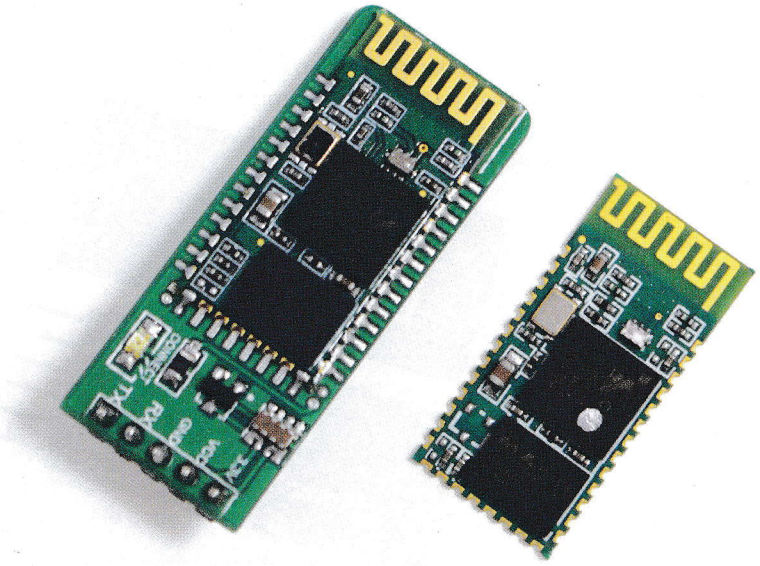
munication série et le Bluetooth : tous les messages qui sont reçus en Bluetooth sont émis sur la broche TX et tout ce qui arrive de la broche RX est envoyé en Bluetooth.

Pour passer d'un mode à l'autre et permettre la configuration, l'une des broches du module (le tout petit) doit être reliée à la tension d'alimentation (3,3V) lors de sa mise en marche. En fonction du module choisi, ceci peut être prévu sur le support sous la forme d'un cavalier, un bouton poussoir, un interrupteur ou... rien du tout. Cela se complique encore davantage lorsqu'il s'agit de savoir précisément quelle broche doit être utilisée (souvent désignée par « KEY »).

Les modules généralement désignés sous le nom HC-05 utilisent la broche 34 (PIO11) pour la sélection du mode alors que les modules désignés par HC-06 utilisent la broche 26 (PIO3). Physiquement, les deux modules sont strictement identiques au composant près. Ce qui change en revanche c'est le code enregistré dans leur mémoire flash (le firmware) : les HC-05 sont capables de prendre le rôle de maître ou d'esclave, alors que les HC-06 ne peuvent être qu'esclave ou en d'autres termes, ne peuvent qu'accepter des connexions et non les initier. Ceci n'est cependant pas très important, car dans la plupart des cas, un projet Arduino sera un périphérique Bluetooth attendant des connexions et ne cherchera de toute façon pas à se connecter de lui-même.

Le module idéal sera donc un HC-05 sur un support adapté et disposant d'un interrupteur pouvant permettre la sélection de mode. Il est important de le préciser, car certains modules sont incomplets ou incompatibles :

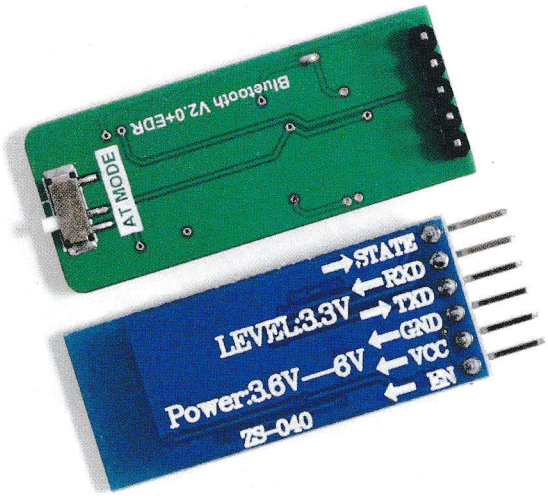
- un HC-05 sur un support pour HC-06 : le bouton, s'il est présent (généralement pas) est connecté à la mauvaise broche (PIO3 au lieu de PIO11) et n'aura aucun effet. Généralement, les supports pour HC-06 sont bleus avec une sérigraphie au verso indiquant l'usage des broches ;
- un HC-06 sur un support pour HC-05 : même problème, même conséquence ;
- un HC-05 ou HC-06 sur un support ne proposant pas de sélecteur de mode : là vous avez deux solutions, soit l'utiliser avec les paramètres par défaut (vitesse, nom, code), soit bricoler pour pouvoir activer le mode de configuration en reliant la bonne broche à la tension d'alimentation, du module (3,3v) non du support (5V). Ceci peut se faire en soudant délicatement un câble sur la broche et en la reliant au moment opportun à la bonne tension.



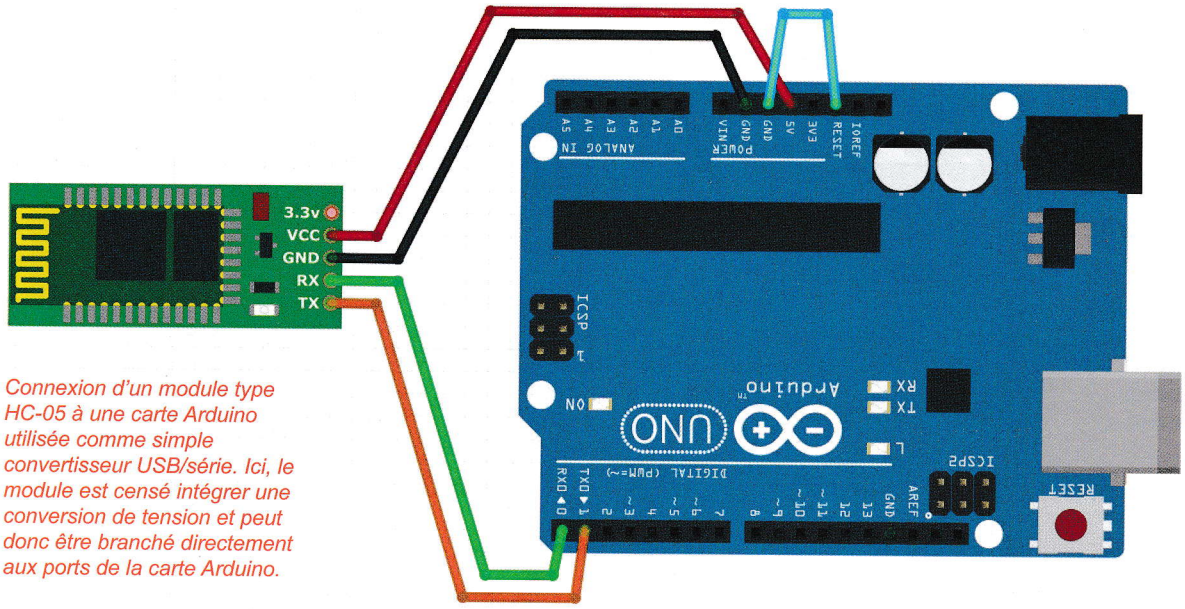
Un module Bluetooth au sens strict du terme est le circuit présenté à droite. Il intègre le contrôleur CSR (carré) et de la mémoire flash (rectangle). Afin de rendre son utilisation plus facile, celui-ci est généralement vendu monté/soudé sur un support avec une connectique plus pratique, un régulateur de tension et une led (à gauche).

Il peut être rapidement difficile de s'y retrouver dans les objets et être plus ou moins sûr de ce qu'on va recevoir. Voici quelques conseils pour vous éviter une déception :

- Évitez comme la peste toutes les annonces ou descriptions qui parlent de RS232. RS232 est une norme pour les connexions séries qui utilisent +/-12v et étaient présentes sur les PC il y a une quinzaine d'années. Le vendeur qui parle de RS232 pour ce type de module ne sait clairement pas de quoi il parle.
- Évitez les modules HC-05 ou HC-06 ne proposant que 4 broches et ne disposant pas de bouton poussoir ou d'interrupteur. La seule solution pour configurer ces modules sera de souder un câble à la minuscule broche permettant l'activation du bon mode.
- À défaut d'interrupteur de sélection de mode, orientez vos préférences vers les modules proposant une broche « KEY ». Il y a de fortes chances que cette broche soit effectivement reliée à celle permettant le changement de mode.
- Un module désigné par HC-05, mais dont la photo présente un support bleu où tous les composants ne sont pas présents a toutes les chances d'être effectivement un HC-05, mais sur un support prévu pour HC-06. Même en soudant un bouton sur le support, vous ne connecterez très probablement pas la bonne broche à la tension d'alimentation.
- Évitez les annonces qui proclament vendre des modules « HC-05/HC-06 ». Cela montre que le vendeur ne fait que copier/coller des termes qu'il ne comprend pas et vous recevrez quelque chose que vous n'attendez certainement pas. Un module est soit un HC-05, soit un HC-06, mais pas les deux.
- Évitez les modules HM-10 ou HM-11, il s'agit de modules Bluetooth LE et vous ne pourrez probablement pas les utiliser avec tous vos équipements Bluetooth. Notez qu'ils ne sont pas foncièrement mauvais, mais il s'agit d'une technologie totalement différente de celle dont nous allons parler.
- Si le vendeur précise « Raspberry Pi » dans l'intitulé de son annonce, passez votre chemin, c'est un attrape-nigaud. Ce type de module peut effectivement se connecter à une Pi mais, passez-moi l'expression, c'est totalement stupide. Il existe des *dongles* USB Bluetooth 2.0+EDR pour le tiers du prix d'un module (1€), bien plus faciles à utiliser, directement compatibles avec Linux et ne nécessitant pas de se passer de la console série de la Pi. Le seul intérêt d'une connexion d'un tel module à une Pi est de « déporter » la console série en Bluetooth, mais c'est là quelque chose de totalement annexe.
- Si le vendeur présente dans son annonce un schéma du support, c'est très bon signe.



Les modules Bluetooth HC-05 et HC-06 se configurent via des commandes AT, après les avoir basculés dans un mode spécial. Certains supports de module permettent cette opération via une broche, un cavalier, un bouton ou, comme ici, avec un micro-interrupteur (en vert). D'autres supports de modules n'offrent aucune possibilité de configuration (celui en bleu). Il faut alors soit bidouiller, soit simplement utiliser la configuration par défaut.



Connexion d'un module type HC-05 à une carte Arduino utilisée comme simple convertisseur USB/série. Ici, le module est censé intégrer une conversion de tension et peut donc être branché directement aux ports de la carte Arduino.

Il aura au moins fait l'effort de fournir un maximum d'informations et il est très probable que le matériel corresponde effectivement au schéma.

- Si vous êtes agile de vos petits doigts, il peut être plus sûr d'acheter séparément le module lui-même et le support. Ceci ne sera pas forcément plus économique et vous prendra un certain temps pour l'assemblage, mais vous saurez avec davantage de précision de quoi il en retourne.
- Préférez, bien entendu, les vendeurs avec une bonne réputation, de nombreuses et bonnes évaluations et des bons commentaires des acheteurs. Préférez également les produits affichant une quantité déjà vendue importante. Généralement, lorsqu'un vendeur se rend compte du mécontentement des acheteurs, il supprime l'annonce et en crée une nouvelle pour le même produit. Les annonces avec beaucoup de ventes à leur actif sont donc généralement un bon signe.

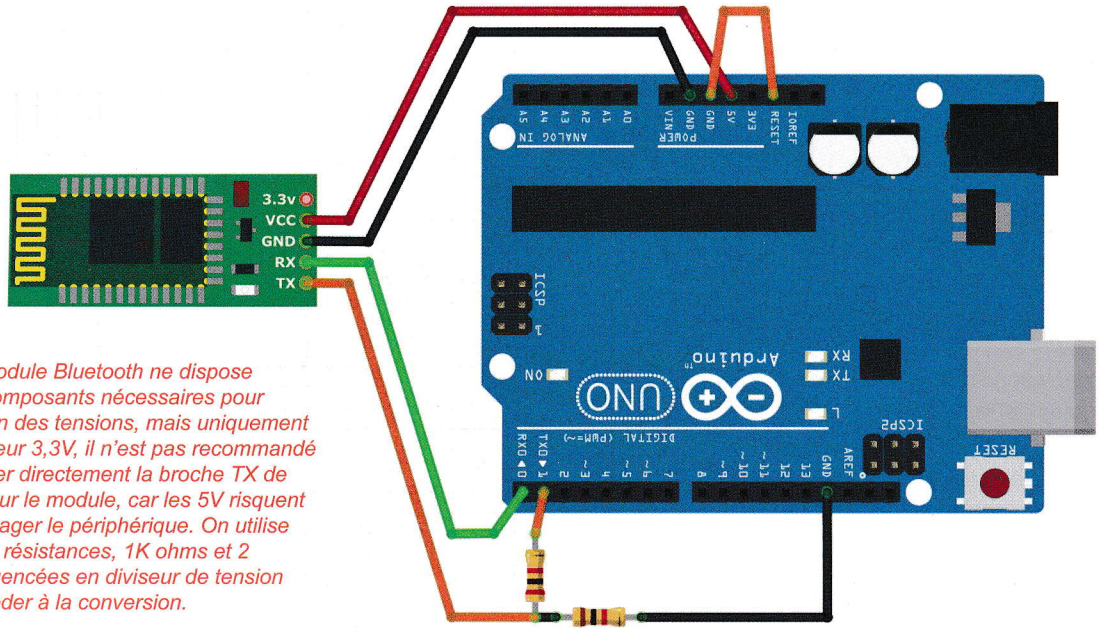
À ce stade, vous l'avez très certainement compris, quoi que vous fassiez, il n'est pas certain que vous obteniez le matériel correspondant à vos attentes. En ce qui me concerne, je dispose de ce type de modules depuis fort longtemps et en avait fait l'acquisition auprès d'un vendeur appelé Wide.hk. Le firmware qu'ils contiennent date de 2009 et correspond à un HC-05. Ils présentent cependant l'avantage de posséder un micro-interrupteur permettant la sélection du mode, chose relativement rare dans les produits actuellement vendus.

Le dernier conseil que je vous donnerai concernant l'achat consiste à ne pas acheter en trop grande quantité auprès d'un unique vendeur (à moins d'être à 100% sûr). Comme généralement le port est offert, il est préférable d'acheter un module chez trois vendeurs plutôt que trois modules chez un vendeur. C'est une question de probabilités, tant au niveau du délai de livraison que de la qualité du produit réceptionné. Rien ne vous empêchera, ensuite, une fois un module testé, d'en commander d'autres exemplaires via la même annonce.

2. CONFIGURATION DU MODULE : LE MODE COMMANDE

Le module Bluetooth arrive généralement avec une configuration par défaut qu'il est préférable de changer : la vitesse de communication, le nom sous lequel le module apparaît en Bluetooth et le code PIN permettant l'appairage (ou le jumelage).

Pour procéder à cette configuration, il faut passer le module en mode commande et utiliser des instructions spécifiques



Si votre module Bluetooth ne dispose pas des composants nécessaires pour l'adaptation des tensions, mais uniquement un régulateur 3,3V, il n'est pas recommandé de brancher directement la broche TX de l'Arduino sur le module, car les 5V risquent d'endommager le périphérique. On utilise alors deux résistances, 1K ohms et 2 Kohms, agencées en diviseur de tension pour procéder à la conversion.

permettant de changer ses paramètres. La communication se fait directement avec le module via la liaison série. Lorsque le module est démarré de façon classique, les données qui lui sont envoyées sont transmises via Bluetooth, mais lorsqu'il se trouve en mode commande, ces données sont des instructions qu'il accepte et traite.

Il existe plusieurs façons d'établir cette communication avec le module. Celle que je trouve la plus aisée et qui ne nécessite aucun élément/équipement supplémentaire, consiste à tout simplement utiliser la carte Arduino comme convertisseur USB/série. En effet, le microcontrôleur d'une carte comme l'Arduino UNO communique avec le PC/Mac et le moniteur série via une communication série au travers d'un convertisseur USB/série intégré (FTDI ou un ATmega16U2). Mais cette ligne de communication est également connectée aux broches TX et RX de la carte elle-même. De ce fait, en désactivant le microcontrôleur en maintenant la broche reset à la masse, on peut établir une connexion directe entre le convertisseur USB/série et un module connecté aux broches RX et TX.

Un simple câble connectant GND et RESET sur la carte la transforme donc en simple convertisseur USB/série, mais maintenir le bouton reset enfoncé, avec une pince à linge par exemple, fonctionnera tout aussi bien. Il ne vous restera plus qu'à connecter le module Bluetooth sur les broches 5V, GND, RX et TX puis utiliser le moniteur série de l'environnement Arduino pour parler au module, passé pour l'occasion en mode commande.

Notez qu'en fonction du module, il peut être risqué de connecter sa broche de réception de données (notée RX en principe) à la carte Arduino. En effet, si le module ne convertit pas les tensions sur les broches RX/TX, la carte va utiliser 5V alors que le module attend 3,3V. Personnellement, je n'ai jamais eu de problème de ce côté, mais en principe, ce n'est pas correct (pour tout dire, je viens tout juste de me rendre compte que j'ai toujours utilisé ces modules, depuis des années, en 0/5V alors qu'ils ne sont censés supporter que 0/3,3V).

Pour faire les choses proprement, la solution la plus rapide consiste à créer un diviseur de tension à l'aide de résistances :

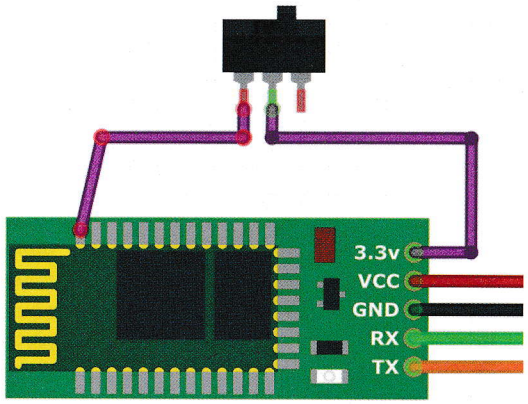
- connectez la sortie de la carte Arduino à une résistance de 1 Kohm (R1) ;
- branchez l'autre borne de la résistance à la broche RX du module ;
- et enfin, connectez cette même borne à la masse via une résistance de 2 Kohms (R2), ou deux fois 1 Kohm en série.

Vous obtenez un diviseur de tension où la différence de potentiel entre la broche du module et la masse sera égale à U*R2/(R1+R2), soit 5*2000/(1000+2000) = 3,33V, dès lors que la sortie de l'Arduino sera à 5V. Dans le sens inverse, vous n'avez rien à faire puisque le module utilise 3,3V comme tension à l'état haut, ce qui est parfaitement dans les tolérances de l'Arduino qui y verra donc le même état.

Une fois le module alimenté et démarré en mode commande, ouvrez le moniteur série, réglez la vitesse sur « 38400 bauds » et les fins de lignes en « Les deux, NL et CR ». À ce stade, vous devez être en mesure d'envoyer la première commande, **AT** et la réponse retournée par le module sera **OK**. Si tel n'est pas le cas, vérifiez différents points :

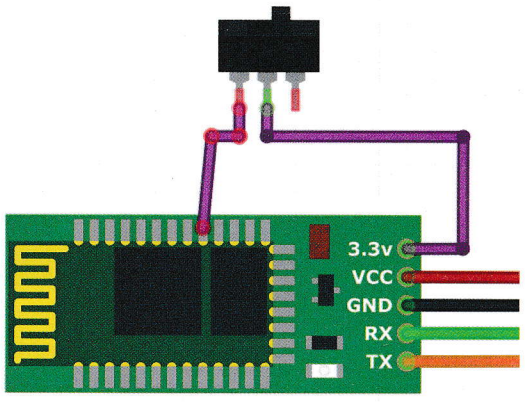
- Le module est-il bien connecté ? La sérigraphie sur le module et les mentions « RX » et « TX » sont parfois inversées. Elles peuvent signifier l'utilité de la broche elle-même au regard du module ou de la connexion. Dans mon cas par exemple avec mes HC-05, « TX » fait référence à la broche permettant d'envoyer des données en Bluetooth (c'est en principe le RX du module lui-même) et est donc connectée au « TX » de la carte Arduino. Ce n'est pas nécessairement votre cas.
- Le module est-il bien en mode commande ? Avec un module HS-05, la led présente sur le support du module clignote de façon continue en mode commande : 1 seconde allumée, 1 seconde éteinte. Lorsque le module est démarré en mode standard, la led clignote rapidement, et si le module est en mode standard et qu'un périphérique Bluetooth y est connecté, la led pulse brièvement une fois toutes les secondes.
- La bonne vitesse est-elle sélectionnée ? La vitesse de communication en mode commande dépend des choix du fabricant et est indépendante de la vitesse en mode standard que vous pouvez configurer. Tous les modules HC-05 que j'ai eus entre les mains utilisaient 38400 bps, mais c'est peut-être différent avec le modèle en votre possession (en fonction de la version du firmware en flash). Essayez de trouver une documentation spécifique à votre module ou testez toutes les vitesses possibles et probables : 9600, 19200, 38400, 57600, 115200, 230400 ou 460800.
- La carte Arduino est-elle effectivement en plein reset grâce au câble ou au bouton poussoir maintenu enfoncé ?

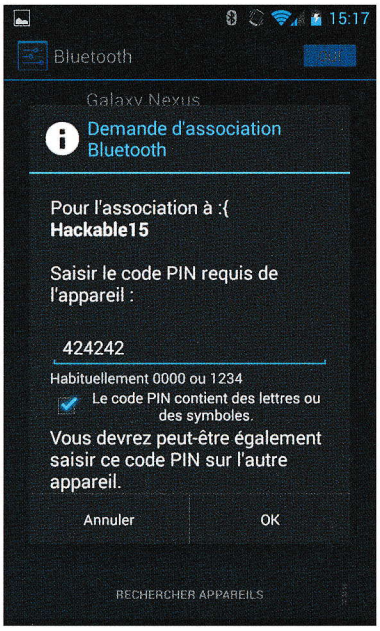
Si vous obtenez la réponse **OK** de la part du module, vous êtes prêt pour la configuration. Les commandes à utiliser débutent toutes par **AT** à l'instar de celles utilisées il y a bien



Si le circuit support du module Bluetooth HC-05 ne permet pas, par une broche, un interrupteur ou un bouton, de passer en mode configuration/commande, un petit bidouillage fera l'affaire. Il suffit de relier la broche 34 du module (PIO11) à la tension d'alimentation 3,3V lors de sa mise en route.

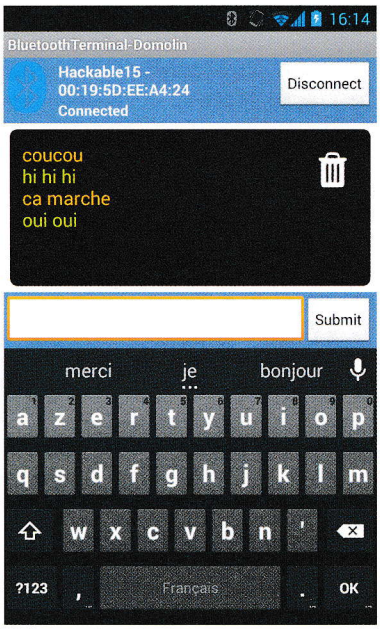
Le passage en mode commande sur un module HC-06 fonctionne de la même manière qu'avec un HC-05, seule la broche utilisée est différente. Ici, c'est la broche 26 (PIO3) qu'il faudra mettre à 3,3V pour démarrer le module dans le bon mode.





Une fois configuré, le module Bluetooth pourra être appairé (ou jumelé, ou associé) comme n'importe quel périphérique. Il vous suffira de saisir le code PIN défini lors de la configuration et le tour sera joué.

Une application comme Bluetooth Terminal de Domolin Bolivia, disponible sur Google Play, vous permettra de rapidement valider le fonctionnement de la liaison. Tout ce que vous taperez côté Android apparaîtra côté Arduino dans le moniteur série et inversement.



longtemps avec les modems analogiques (commandes Hayes) ou encore les modules ESP8266 utilisés avec leur firmware d'origine. Ces commandes permettent de définir des paramètres gérant son comportement. Nous n'allons ici couvrir qu'un seul cas d'utilisation où le module a un rôle d'esclave en Bluetooth : il est en attente de connexion, mais n'initie pas de communications lui-même.

Les commandes se divisent en deux catégories : celles ne réglant pas un paramètre, mais ordonnant une action, et celles permettant de consulter et de définir des valeurs de configuration. Un exemple de commande déclenchant une action est :

```
AT+VERSION
+VERSION:1.0-20090818
OK
```

La commande envoyée est **AT+VERSION** et la réponse débutant par un **+** répète l'ordre, suivi de la réponse effective : **1.0-20090818**. La fin d'une réponse est toujours marquée par **OK**. Si un problème survient pour une commande, le module répondra **ERROR:(0)** en lieu et place de **OK** (le **0** correspond à un code d'erreur, mais celui-ci ne semble documenté nulle part).

Une autre commande de ce type permet d'obtenir l'adresse Bluetooth matérielle (MAC) du module :

```
AT+ADDR
+ADDR:19:5d:eea424
OK
```

Cette adresse ne peut être changée et est unique à chaque périphérique Bluetooth alors que son nom, par exemple, peut être modifié. L'adresse est ici présentée de manière un peu « brouillonne » et se transcrit en **19:5d:ee:a4:24** ou en **19:5D:EE:A4:24**, c'est ainsi qu'elle apparaîtra pour un appareil procédant à une recherche de périphériques Bluetooth.

Les commandes permettant le réglage se terminent soit par un **?** pour une consultation de la valeur en cours, soit par **=** suivi d'une nouvelle valeur pour un réglage. Commençons pas consulter le nom actuellement configuré :

```
AT+NAME?
+NAME:WIDE_HK
OK
```

Pour définir un nouveau nom puis vérifier l'enregistrement :

```
AT+NAME=Hackable15
OK

AT+NAME?
+NAME:Hackable15
OK
```

Nous pouvons également faire de même avec le code PIN permettant l'appairage :

```
AT+PSWD?
+PSWD:1234
OK

AT+PSWD=424242
OK

AT+PSWD?
+PSWD:424242
OK
```

La commande **AT+ROLE?** permet de connaître le rôle que doit prendre le module dans une relation Bluetooth. **0** pour esclave et **1** pour maître. Avec un module

HC-06 ne permettant que le rôle d'esclave, cette commande n'existe pas.

Et enfin, nous avons la configuration de la vitesse de communication en mode standard :

```
AT+UART?
+UART:9600,0,0
OK
```

Cette commande permet de configurer non seulement la vitesse entre la carte Arduino et le module, mais également le format de données. Nous avons là trois paramètres séparés par des virgules : vitesse, bit de stop, parité. Un croquis de carte Arduino utilise généralement quelque chose comme `Serial.begin(9600)` en ne précisant que la vitesse, mais le format de données peut également être spécifié. Par défaut, il sera de 8 bits de données, pas de parité et un bit de stop (alias 8N1). Dans le cas du module nous n'avons pas le choix du nombre de bits de données, mais pouvons régler le reste. Ainsi pour 19200, pas de parité et un bit de stop nous ferons :

```
AT+UART=19200,1,0
OK

AT+UART?
+UART:19200,1,0
OK
```

Une dernière chose devra sans doute être configurée, en particulier si vous rencontrez des problèmes de connexion avec certains appareils. Chaque périphérique Bluetooth annonce un certain type lorsqu'il est scanné, c'est ce qui permet aux smartphones, par exemple, d'afficher une petite icône lors d'une recherche Bluetooth (casque, PC, etc.). Ce type est défini par une classe de périphérique et de service ou *Class of Device/Service* (CoD) en anglais. Il s'agit d'une valeur qu'il est possible de configurer par une commande `AT` et qui est souvent à `0x000000` par défaut. Certains appareils n'aiment pas que la classe soit à zéro et refusent de lister le périphérique.

Pour définir cette valeur, il vous suffit d'aller sur un site comme **http://www.ampedrftech.com/cod.htm**, et de choisir la classe la plus proche de ce que vous comptez faire avec le module. Cochez les cases de votre choix sur la page, affichez la valeur, puis passez-la avec la commande `AT+CLASS=` :

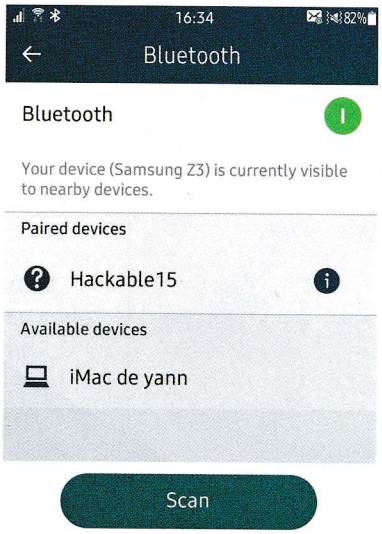
```
AT+CLASS?
+CLASS:0
OK

AT+CLASS=0C0800
OK

AT+CLASS?
+CLASS:c0800
OK
```

`0C0800` définit un périphérique proposant des services de rendu (*Rendering*) et de capture (*Capturing*) et étant un jouet (*Toy*) de type non défini (*Undefined*). Généralement, cette valeur satisfait la plupart des appareils et smartphones.

Notre module est maintenant configuré :

- nom : « Hackable15 »,
- code PIN : « 424242 »,
- vitesse : 19200 8N1.

Il suffit alors de déconnecter le module de son alimentation, repasser en mode standard en le reconnectant sans forcer le mode commande (soit en basculant l'interrupteur s'il est présent, soit en ne connectant plus la broche adéquate à 3,3V). Nous sommes prêts pour le premier essai.



Cette communication Bluetooth marchera sur la plupart des périphériques supportant Classic Bluetooth ou Bluetooth inférieur à 4.0, et même sous Tizen (mais il n'y a pas encore d'application pour s'en servir). Seuls les iPod/iPad/iPhone seront incapables de prendre en charge ces modules parce que la firme à la grande pomme a décidé de ne pas supporter le profil Bluetooth associé (SPP). En même temps, ce n'est pas étonnant, ce sont les mêmes qui ont décidé dernièrement que la prise audio jack était obsolète et n'avait plus rien à faire sur un smartphone...

L'antenne intégrée aux modules sous la forme d'une piste de cuivre est souvent suffisante pour des besoins courants. Pour augmenter la portée cependant, rien n'empêche de bricoler un peu et d'ajouter un connecteur RP-SMA et une « vraie » antenne...

3. UN PREMIER ESSAI AVEC ANDROID ET LE PROBLÈME D'IOS

Pour nous assurer que la configuration est appliquée et que nous pouvons communiquer avec le module en Bluetooth, nous n'allons pas utiliser de croquis. La configuration matérielle actuelle, avec le module directement connecté au moniteur série, sera largement suffisante (n'oubliez pas cependant de changer la vitesse en celle configurée précédemment). Il nous suffit alors de prendre un smartphone ou une tablette Android compatible Classic Bluetooth ou Bluetooth inférieur à 4.0 et de trouver la bonne application.

Pourquoi forcément un périphérique Android ? C'est simple, les iPhone et iPad ne peuvent pas communiquer avec le module. Ceci découle d'un choix d'Apple concernant les profils Bluetooth supportés par leurs périphériques. Nous avons parlé précédemment des normes Bluetooth, mais nous n'avons couvert qu'une partie de ce qui existe : les *Bluetooth Core Specifications*. En plus de cela, nous trouvons la notion de profils déterminant le type de communication supporté par un périphérique Bluetooth. En l'occurrence, un iPhone ou un iPad supporte les profils suivants (dixit la page du support Apple, **https://support.apple.com/fr-fr/HT204387**) :

- HFP (*Hands-Free Profile*) : le profil pour les kits mains libres ;
- PBAP (*Phone Book Access Profile*) : accès aux annuaires, carnet d'adresses et contacts ;
- A2DP (*Advanced Audio Distribution Profile*) : diffusion audio ;
- AVRCP (*Audio/Video Remote Control Profile*) : contrôle de diffusion audio/vidéo (télécommande) ;
- PAN (*Personal Area Networking*) : réseau personnel ;
- HID (*Human Interface Device*) : interface homme/machine (clavier, souris, etc.) ;
- MAP (*Message Access Profile*) : messagerie.

Le problème est que nos sympathiques petits modules utilisent le profil pour lequel ils sont faits : *Serial Port Profile* ou SPP, un profil permettant d'émuler une communication série via Bluetooth. Il semblerait que le choix de ne pas supporter SPP soit en rapport avec le contrôle des périphériques autorisés (au sens industriel du terme) et le programme de licences *Made For iPod* (MFi) permettant à Apple de choisir qui a le droit de fabriquer des accessoires pour leurs périphériques. Le fait que les produits Apple ne soient pas très bidouille-compatibles n'est pas nouveau, mais dans ce cas précis, c'est tout simplement éliminatoire. Si vous avez un iPod, un iPhone ou un iPad, vous devrez vous rabattre sur la solution du Bluetooth LE avec des modules comme le HM-10 par exemple, au fonctionnement assez différent de ceux traités ici.

Sur un smartphone Android, dès le module démarré en mode standard, celui-ci devrait être visible dans les préférences Bluetooth lors d'une recherche (scan). Celui-ci devra apparaître soit avec son nom, soit avec son adresse matérielle. Vous pourrez alors le sélectionner pour procéder à l'appairage et entrer le code PIN que vous aurez défini précédemment. Ceci fait et validé, votre smartphone pourra se connecter au module sans autre forme de procès.

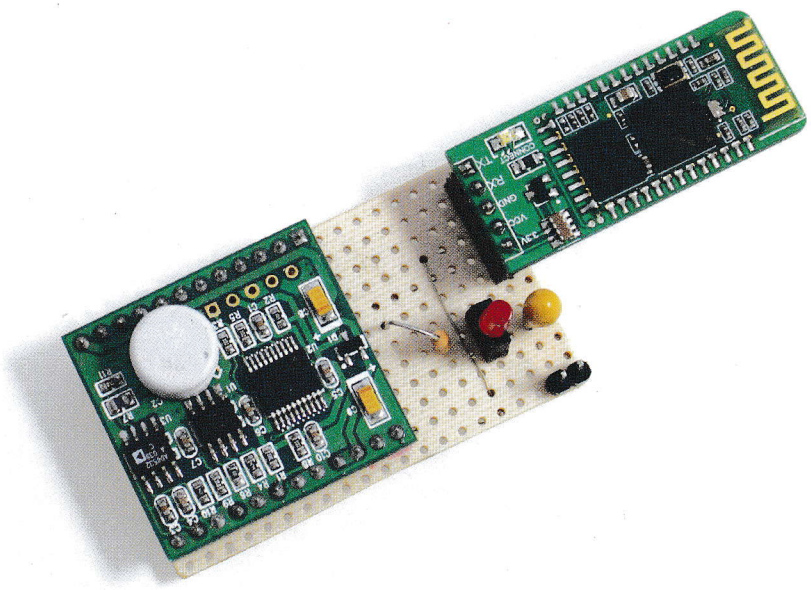
Pour procéder au test, vous devrez utiliser une application faisant office de moniteur série exactement comme celui de l'IDE Arduino. Une simple recherche de « Bluetooth terminal » sur Google Play devrait vous afficher une sélection relativement conséquente. Après plusieurs tests d'applications, ma préférence s'est arrêtée sur « Bluetooth Terminal » de Domolin Bolivia. L'application est modeste, mais fait parfaitement son travail. Il vous suffira alors de choisir le périphérique Bluetooth dans la liste, vous connecter et envoyer des messages. Tout ce que vous validerez côté Android devra apparaître dans le moniteur série Arduino et inversement. Ceci validera l'état de la communication et le fonctionnement de l'ensemble.

CONCLUSION TEMPORAIRE

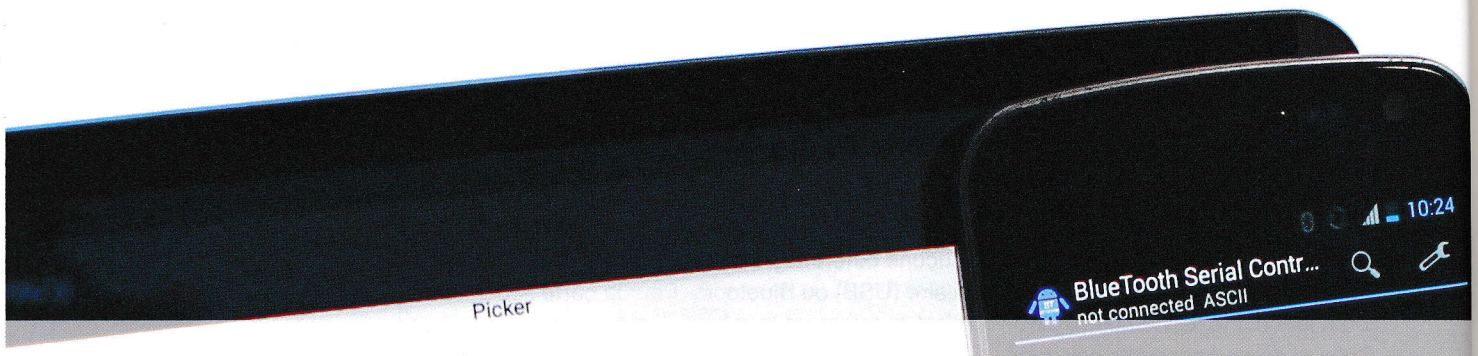
Notez qu'il est déjà possible de créer un projet sur cette base. Tout ce que vous aurez à faire sera de développer un croquis exactement de la même manière que s'il s'agissait d'interpréter des ordres provenant du moniteur série. Cela ne fait absolument aucune différence qu'il s'agisse d'une liaison filaire (USB) ou Bluetooth. Pour la carte Arduino, c'est exactement la même chose.

Notez toutefois que le fait d'avoir le module Bluetooth connecté au seul port série d'une carte comme l'Arduino UNO peut poser problème lors de l'enregistrement du croquis. Mieux vaut débrancher le module, programmer la carte, puis le rebrancher. Une autre solution sera d'utiliser la bibliothèque *SoftwareSerial* pour connecter le module sur d'autres broches de la carte et réserver le vrai port série pour la programmation et la mise au point du croquis.

Vous conviendrez avec moi que le fait d'envoyer des commandes ainsi à votre croquis peut être très amusant, mais pas vraiment ergonomique, et encore moins impressionnant si vous comptez faire la démonstration de vos projets. Nous allons régler ce petit problème dans l'article qui suit et, par la même occasion, user justement de *SoftwareSerial* pour nos communications... **DB**



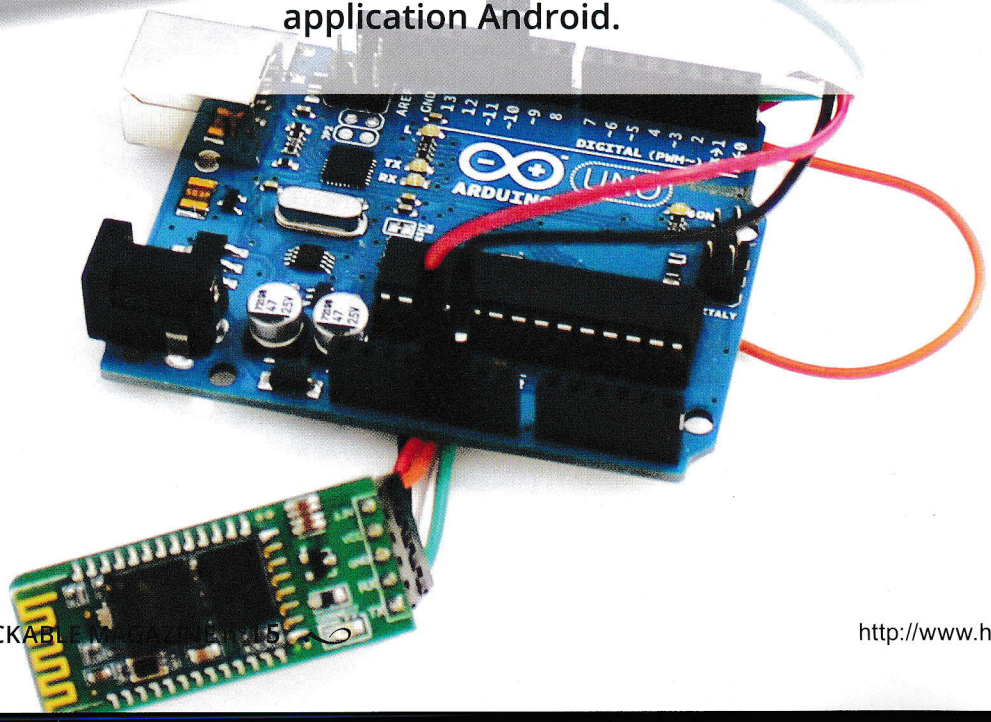
Le fait que ces modules Bluetooth soient de simples convertisseurs Bluetooth/série présente un avantage intéressant : il n'est pas toujours utile de faire intervenir une carte Arduino. Ici, le module sur la gauche est un capteur de température et d'humidité relative communiquant via une liaison série. Pour créer un capteur Bluetooth, il suffit de configurer le module Bluetooth avec la bonne vitesse et de relier les deux en ajoutant une alimentation.



BLUETOOTH : PILOTEZ VOTRE ARDUINO AVEC VOTRE SMARTPHONE

Denis Bodor

À présent que nous avons fait connaissance avec les modules Bluetooth, il est temps de passer aux choses sérieuses ou du moins, aux choses plus utiles. La connectivité Bluetooth est disponible sur bien des systèmes, des PC aux Raspberry Pi, mais la plateforme la plus intéressante est sans l'ombre d'un doute le smartphone. Voyons donc ensemble comment commander un projet Arduino depuis une application Android.



Précisons de suite ce que cet article n'expliquera pas : le développement d'une application Android. En effet, même s'il est techniquement possible à tout un chacun de créer une application pour son smartphone, ce n'est pas quelque chose qui peut être fait aussi rapidement et aussi facilement qu'un croquis Arduino. Android Studio, l'environnement de développement proposé gratuitement par Google pour développer des applications Android, n'est pas conçu dans un but pédagogique, comme c'est le cas d'Arduino, mais dans le but de... développer des applications.

Bien sûr, la création d'un premier programme, le fameux « Hello World », est quelque chose de relativement rapide et vous aurez, en quelques minutes votre propre application vous faisant « coucou » sur votre smartphone, après avoir suivi un petit tutoriel en ligne. Le problème n'est pas là, mais dans l'étape suivante : sauter le pas et concevoir, avec un langage que vous ne connaissez peut-être pas encore, une application complète avec plusieurs écrans (ou « vues » dans le monde Android), des interactions avec l'utilisateur et le système, ou encore disposant d'une ergonomie digne de ce nom.

La seule création d'une telle application, même simple, détaillée de manière intelligible, nécessiterait un magazine complet (chose de nous avons d'ailleurs fait aux Éditions Diamond, sous la forme d'un hors-série de *GNU/Linux Magazine* (n°82)), car il faudrait aborder beaucoup de concepts et de principes de développement qui n'ont rien de didactique.

Fort heureusement, il ne nous sera pas nécessaire de créer une application pour envoyer des ordres en Bluetooth à notre projet Arduino. En effet, un certain nombre de programmeurs, coutumiers de ce genre de pratiques, proposent leurs applications sur Google Play et il nous suffit de les utiliser en programmant uniquement la partie Arduino.

Bien entendu, si vous désirez une interface spécifique ou souhaitez explorer des domaines ou des fonctionnalités qui n'ont pas été déjà couverts, vous n'aurez d'autres choix que de vous initier à la programmation Android. Il en va de même si vous souhaitez développer une application iOS, à condition bien entendu, d'écarter l'option des modules que nous avons précédemment configurés et de vous lancer dans le monde du Bluetooth Low Energy (puisque les iPhone et iPad ne sont pas compatibles avec les modules Bluetooth utilisant un profil SPP).

1. CONTRÔLEZ LA COULEUR DE LEDS

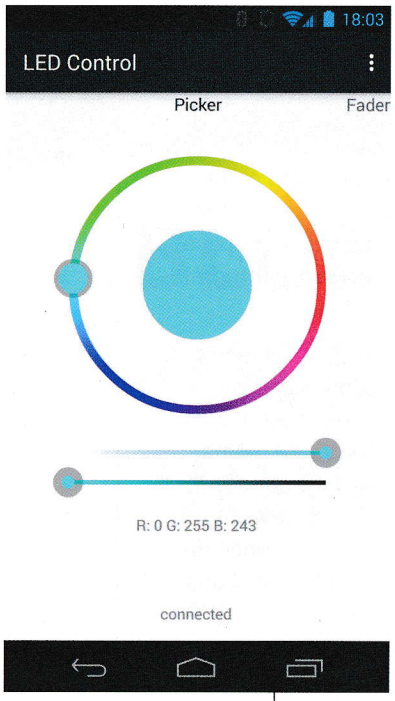
La première application que nous allons utiliser est *LED Control* de Michael Fennel. Celle-ci est disponible gratuitement sur Google Play, mais ne propose pas d'accès à son code source (si vous aviez dans l'idée de l'améliorer). Cette application permet de contrôler la couleur d'une ou plusieurs leds connectées à un montage équipé d'un module Bluetooth SPP comme ceux que j'ai décrits précédemment.

Bien que le développeur ayant créé l'application ne semble pas vouloir exposer son fonctionnement interne, il met à disposition un croquis d'exemple sur GitHub ainsi qu'une documentation succincte détaillant le protocole utilisé (**https://github.com/fennel-labs/LED-control/wiki/protocol**). En effet, l'application, après connexion en Bluetooth, envoie des données dans un format précis définissant la couleur que doivent prendre la ou les leds à contrôler.

Pour obtenir LED Control, flashez ce code.

Le format des messages toujours sur 6 octets est le suivant :

- Octet 0 : numéro ou adresse de la led à contrôler (pour une utilisation future, à l'heure actuelle c'est toujours 0) ;



L'application LED Control de Michael Fennel vous permettra de contrôler la couleur d'une ou plusieurs leds, voire d'un éclairage complet, depuis votre smartphone Android.

- Octet 1 : mode de fonctionnement, 0 pour le mode « monochrome » et 1 pour le mode « fondu » (animation) ;
- Octet 2 à 4 : dépend du mode. En mode « monochrome », ces trois octets définissent respectivement les valeurs de rouge, de vert et de bleu entre 0 et 255. En mode « fondu », l'octet 2 précise la vitesse et l'octet 3 l'état de l'animation (0 = oui, 1 = non). L'octet 4 est alors inutilisé ;
- Octet 5 : le caractère `;` utilisé pour marquer la fin du message.

L'application présente deux écrans différents : un sélecteur de couleur (*Picker*) et un contrôleur d'animation (*Fader*). Le sélecteur se présente comme un anneau de couleur où l'on peut déplacer un curseur. Au centre, une zone circulaire permet d'alterner la couleur entre celle sélectionnée, le blanc et le noir (led éteinte). Deux curseurs supplémentaires, placés en dessous de l'anneau, permettent de régler la saturation et la valeur de la couleur actuelle (du blanc à la couleur et de la couleur au noir).

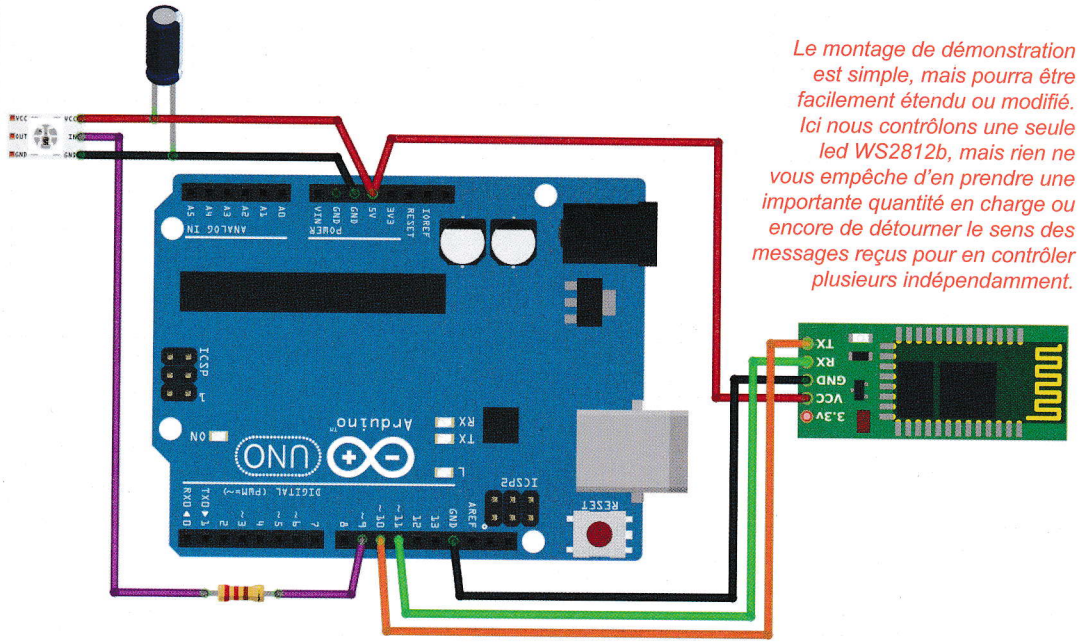
Le second écran permet de régler une vitesse entre 0 et 100% (correspondant aux 0 à 255 dans les messages) à l'aide d'un curseur horizontal et l'état de l'animation (on/off) à l'aide d'un bouton.

Notez que tout ceci forme les paramètres non variables avec lesquels nous allons travailler. La logique première veut que le réglage de la couleur corresponde effectivement à celle de la ou des leds connectées à notre carte Arduino et que le contrôle d'animation fasse varier continuellement l'intensité de chaque composante de couleur (RVB). Cependant, dans l'absolu, vous n'êtes pas du tout obligé de respecter ces principes. L'écran de contrôle de l'animation peut parfaitement servir pour un choix d'effet (de la bibliothèque WS-2812FX, par exemple), en décodant arbitrairement le fait que 10% de la vitesse correspond à un certain effet, 15% à un autre, 20% un troisième, etc. Vous ne contrôlez pas la relation application/message, mais vous pouvez faire ce que vous voulez avec la relation message/action côté Arduino.

Michael Fennel fournit un croquis Arduino d'exemple matérialisant sa vision de cette relation. Nous allons ici procéder de même, mais avec une approche plus simple et plus épurée afin de démontrer la facilité qu'offre l'utilisation d'un module Bluetooth pour vos projets.

Le montage de démonstration sera relativement simple puisque nous allons tout bonnement connecter le module Bluetooth à deux broches arbitrairement choisies de la carte Arduino, en plus de l'alimentation 5V et de la masse. Nous n'allons, en effet, pas utiliser le port série de la carte, mais la bibliothèque *SoftwareSerial* nous permettant de mettre en place ce type de liaison de façon logicielle. Ceci fonctionnera exactement comme `Serial()` tout en conservant la communication avec l'IDE Arduino et son moniteur série pour peaufiner nos réalisations.

En ce qui concerne la led que nous allons contrôler, le choix le plus simple consiste à utiliser une WS2812b (alias NeoPixel). Le croquis de démonstration de Michael Fennel utilise des sorties analogiques (PWM) et une led RVB standard, mais ceci n'a pas vraiment d'importance. Le croquis que nous allons utiliser pourra être facilement adapté à presque n'importe quel usage et n'importe quel composant. Notre unique WS2812b (qui est en réalité un clone au format 8 mm translucide ici) sera connecté à la broche 9



Le montage de démonstration est simple, mais pourra être facilement étendu ou modifié. Ici nous contrôlons une seule led WS2812b, mais rien ne vous empêche d'en prendre une importante quantité en charge ou encore de détourner le sens des messages reçus pour en contrôler plusieurs indépendamment.

en plus des 5V et de la masse. Là encore, c'est un choix arbitraire sans importance. Pour correctement faire les choses, il est recommandé d'ajouter un condensateur de quelques 100µF entre l'alimentation et la masse, ainsi qu'une résistance entre la broche et l'entrée de données de la led, en particulier si votre environnement de travail est riche en perturbations électrostatiques.

Il ne nous reste plus, à présent, qu'à rédiger le croquis qui sera extrêmement court. Nous utilisons les bibliothèques *SoftwareSerial* pour la création d'un port série supplémentaire et *Adafruit_NeoPixel* pour la prise en charge de la led WS2812b.

La création des objets représentants les deux éléments se fera respectivement avec :

- `SoftwareSerial btmodule(10, 11)` qui nous permet d'obtenir un `btmodule` utilisable exactement comme `Serial`. Nous passons en argument les broches correspondants aux signaux RX et au TX.
- `Adafruit_NeoPixel ws = Adafruit_NeoPixel(1, 9, NEO_RGB + NEO_KHZ800)` qui rendra `ws` utilisable dans le reste du croquis. Là, les arguments sont un peu plus nombreux avec, dans l'ordre, le nombre de leds (`1`), la broche utilisée (`9`) et une combinaison de macros précisant le type d'encodage des valeurs RVB (`NEO_RGB` pour notre led 8mm, mais `NEO_GRB` pour une vraie WS2812b) et la vitesse de communication (`NEO_KHZ800` pour 800 khz).

Nous pouvons ensuite initialiser l'ensemble (port série, port série logiciel et led) dans la fonction `setup()` avant de spécifier une couleur noire pour la led. En effet, le modèle utilisé s'illumine en bleu dès sa mise sous tension, ce qui n'est pas très agréable.

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

// Déclaration NeoPixel
// NEO_RGB pour la led 8mm
// NEO_GRB pour une vraie WS2812b
Adafruit_NeoPixel ws = Adafruit_NeoPixel(1, 9,
NEO_RGB + NEO_KHZ800);

// Déclaration port série logiciel
SoftwareSerial btmodule(10, 11); // RX, TX

void setup() {
  // moniteur série en 115200 bps
  // (utile pour débugger)
  Serial.begin(115200);
  while (!Serial);

  // port logiciel en 19200
  // vitesse configurée avec AT+UART=19200
  btmodule.begin(19200);
```

```
  // initialisation NeoPixel
  ws.begin();
  // noir
  ws.setPixelColor(0, ws.Color(0,0,0));
  // affichage
  ws.show();
}

void loop() {
  // Des données à lire ?
  if (btmodule.available()) {
    // Oui, on récupère la commande
    String commande = btmodule.readStringUntil(';');
    // on a 5 octets (sans le ";") ?
    if(commande.length() == 5) {
      // oui, c'est une commande valide
      // le second octet est une commande de couleur ?
      if(commande.charAt(1) == 0x00) {
        // oui, on donne au NeoPixel la couleur demandée
        ws.setPixelColor(0, ws.Color(commande.charAt(2),commande.
charAt(3),commande.charAt(4)));
        ws.show();
      }
    }
  }
}
```

Il ne reste ensuite plus qu'à gérer les messages arrivant dans la fonction `loop()` en vérifiant tout d'abord la présence de données reçues, avec `btmodule.available()`. Si tel est le cas, nous utilisons la méthode `readStringUntil()` en précisant comme argument le caractère marquant la fin d'un message (`;`). Si l'opération réussie, `commande` contiendra le message en question, débarrassé de son point-virgule.

Celui-ci, pour être valide, devra avoir une taille de 5 caractères/octets. Si ce n'est pas le cas, le message est corrompu et inutilisable et nous arrêtons là. Dans le cas contraire, nous avons un message à traiter. `charAt()` peut alors être utilisé pour vérifier s'il s'agit d'une commande de changement de couleur tel que décrit dans le protocole de Michael Fennel, avec la valeur `0` comme second octet.

À ce stade, nous sommes donc relativement confiants quant à l'état et la nature du message et nous utilisons encore une fois `charAt()` pour extraire les octets correspondants à chaque valeur de couleur et les utiliser directement pour changer les valeurs utilisées par la led. Enfin, nous « poussons » ces changements avec `show()`.

Le test sera fort simple après enregistrement du croquis dans la carte Arduino : nous utilisons l'application de Michael Fennel, via le menu ***Connection Settings*** pour choisir notre périphérique Bluetooth que nous avons précédemment appairé et jouons avec les contrôles et curseurs à l'écran. En temps réel, la couleur de la led devrait changer selon notre bon plaisir.

Je n'ai pas ici pris en charge la fonction d'animation (message avec le second octet à 1), mais on peut facilement imaginer ajouter une simple condition et, par exemple, remplacer tous les appels aux fonctions et méthodes de *Adafruit_NeoPixel* par celles de *WS2812FX*, qui elles-mêmes utilisent *Adafruit_NeoPixel*, et définir un ou des effets en conséquence.

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

http://www.ed-diamond.com

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

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

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

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

En renvoyant 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 :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC HACKABLE
POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Prix en Euros / France Métropolitaine

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

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

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

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :
http://www.ed-diamond.com pour consulter toutes les offres !

2. CONTRÔLER DES RELAIS OU D'AUTRES PÉRIPHÉRIQUES

Une autre application que je trouve très intéressante (gratuite et sans pub), car très polyvalente est *BlueTooth Serial Controller* de NEXT PROTOTYPES. Celle-ci peut sembler complexe au premier regard, mais elle permet de faire des merveilles. Là, nous sommes dans une situation différente de celle de l'application précédente qui était très spécialisée. *BlueTooth Serial Controller* vous propose d'associer les boutons d'un pavé de 9 à 25 boutons sur l'écran avec l'envoi de messages de votre choix.

Après installation de l'application, on fera un tour dans les préférences et commencera par activer le mode « Orientation/Paysage » (pour un smartphone) ainsi que le mode « 9 BUTTON MODE » afin de masquer le second pavé de 16 boutons.

Pour une utilisation en bascule, comme pour le contrôle de relais, la logique sera la suivante : l'application propose 5 écrans de contrôle (*controllers*) différents nommés de A à E. Par défaut, on n'utilise qu'un seul contrôleur en envoyant des messages lors du contact avec un bouton, mais ici, nous voulons une bascule. Nous allons donc configurer le contrôleur A avec les messages à envoyer pour activer le ou les relais, et le contrôleur B pour les messages désactivant ces relais.

Nous allons donc commencer par faire un tour, toujours dans les préférences, dans le menu ***BUTTON/Name***, ce qui nous conduit à un autre écran de configuration. Là se trouve une longue liste de 26 boutons que nous pouvons nommer comme bon nous semble. Très originalement, nous allons libeller nos 9 boutons en R1 à R9 (en mode 9 boutons, les boutons 10 à 25 n'ont pas d'importance).

Nous revenons ensuite en arrière pour tapoter l'entrée ***BUTTON/Commande***. Une liste très similaire s'affiche nous proposant de saisir, pour chaque bouton, un message à envoyer. Pour la démonstration, nous nous contenterons de configurer le premier bouton en lui associant le message « r1on ».

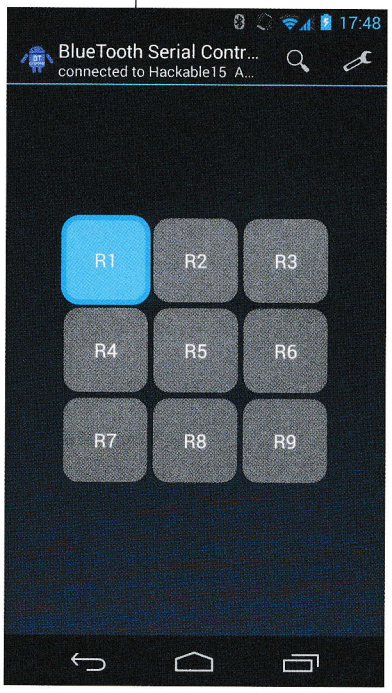
Nous revenons ensuite à l'écran précédent et défilons jusqu'à l'entrée ***ON/OFF Mode*** qui nous conduit à un autre écran de configuration. Là, activez ***Enable ON/OFF Mode*** et juste dessus tapez sur ***Another Command's Source*** et choisissez ***Controller B***. Ceci paramètre l'application pour le mode bascule : en activant un bouton, nous envoyons le message définit précédemment, mais en désactivant ce même bouton, nous comptons envoyer le message du contrôleur B et non A.

De ce fait, revenez à l'écran précédent et, tout en haut des options, passez sur le contrôleur B, puis retournez sur ***BUTTON/Command***. Vous constaterez que le message associé au bouton 1 est la valeur par défaut (« def ») et non plus « r1on ». C'est normal, nous configurons le contrôleur B et non le A. Changez la configuration pour envoyer le message « r1off » cette fois puis, enfin, revenez à l'écran précédent pour rebasculer sur la contrôleur A.

Pour terminer, quittez la configuration pour afficher l'écran principal avec les 9 boutons. Un « clic » sur le bouton 1, nommé R1 enverra « r1on » tout en l'activant (il devient bleu) et un second « clic » désactivera le bouton (redevenant gris) tout en envoyant « r1off » (du contrôleur B). Il ne nous reste alors plus qu'à créer le croquis Arduino correspondant.

Pour obtenir BlueTooth Serial Controller, flashez ce code.

BlueTooth Serial Controller est une application Android offrant la possibilité de configurer l'effet d'une série de boutons présentés à l'écran. La configuration est un peu touffue, mais l'étendue des possibilités est très importante puisque c'est vous qui décidez de la nature et la forme des messages envoyés en Bluetooth.



Celui-ci ne sera que très légèrement différent du précédent, car il nous suffit d'éliminer toutes traces du support pour la led et d'utiliser la broche 9 pour contrôler un relais (ou autre chose comme une led pour test) :

```
#include <SoftwareSerial.h>

// Déclaration port série logiciel
SoftwareSerial btmodule(10, 11); // RX, TX

void setup() {
  // moniteur série en 115200 bps
  // (utile pour débugger)
  Serial.begin(115200);
  while (!Serial);

  // port logiciel en 19200
  // vitesse configurée avec AT+UART=19200
  btmodule.begin(19200);

  pinMode(9, OUTPUT);
  digitalWrite(9, HIGH);
}

void loop() {
  // Des données à lire ?
  if (btmodule.available()) {
    // Oui, on récupère la commande jusqu'à LF
    String commande = btmodule.readStringUntil('\n');
    if(commande.length() > 0) {
      // on a une commande
      if(commande == "r1on")
        digitalWrite(9, LOW);
      if(commande == "r1off")
        digitalWrite(9, HIGH);
    }
  }
}
```

On retrouve ici le **readStringUntil()**, mais cette fois avec **\n** en argument, qui est le caractère « nouvelle ligne », alias LF. Il est possible de choisir, dans la configuration de l'application, ce marqueur de fin de ligne (par défaut) ou CR (**\r**) ou CR et LF. Nous testons également si la chaîne obtenue est vide (en cas de problème) et finalement, on active ou désactive la sortie en fonction du message « r1on » ou « r1off ». Notez au passage que le module relais utilisé ici étant contrôlé par un transistor PNP SS8550, la sortie est inversée.

Nous n'avons ici configuré qu'un bouton, une paire de messages et un relais mais, vous l'aurez compris, une simple carte Arduino, avec cette application, pourra très facilement contrôler 9 relais de la même manière, avec une interface relativement sympathique.

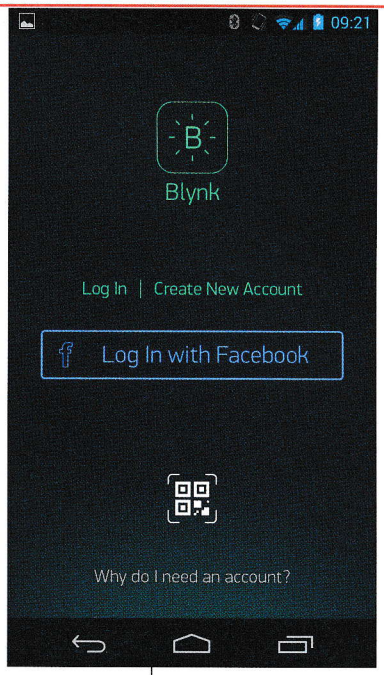
Si vous procédez à une recherche sur Google Play, avec « Bluetooth relay », vous trouverez une quantité impressionnante d'applications. La plupart sont trop restrictives, peu configurables ou, disons-le franchement, carrément moches et boguées. Après en avoir testé une bonne douzaine, j'ai retenu celle-ci pour sa simplicité (relative) et sa souplesse. Mais il est possible qu'il en existe une plus personnalisable encore.

3. PLUS LOIN : UN MOT À PROPOS D'ARDUDROID, DE BLYNK ET MIT APP INVENTOR 2

Dans ma tentative de sélections d'applications démonstratives, je suis tombé sur ce qu'on pourrait appeler des classiques qui semblent plébiscitées par bon nombre d'utilisateurs, mais que j'ai écartées pour des raisons personnelles et/ou des préférences politico-philosophiques.

ArduDroid de Hazim Bitar, par exemple, est très sympathique, car cette application permet, à l'aide d'un croquis dédié, de prendre le contrôle sur toutes les broches d'une carte Arduino. En entrée comme en sortie, y compris en analogique. Le problème cependant, est que cela ne reste intéressant que pour des tests et non pour un usage courant ou une démonstration grandiloquente de vos capacités techniques (vous savez le « *non, mais moi je contrôle tout mon appart avec mon téléphone* », « *oui, c'est moi qui ai tout fait* »). Contrôler les broches n'est pas la même chose que de choisir une couleur sur un écran...

Une autre application très très à la mode est Blynk. Le concept est en effet extrêmement séduisant puisque l'interface de l'application est totalement modulaire et graphiquement très agréable. C'est un système complet fonctionnant aussi bien en Bluetooth qu'en Wifi et ce avec des cartes Arduino ou des choses plus musclées comme une Raspberry Pi. Jauges, graphiques, curseurs, boutons, afficheurs, voyants... tout y est ! Mais voilà, une application qui me demande de me connecter ou créer un compte pour utiliser mon propre matériel a une fâcheuse tendance à me mettre les nerfs à vif, tout autant qu'une liste longue comme le bras de permissions demandées par l'application (incluant un accès à la caméra, aux données GPS, et aux achats « Achats via l'application »).



Blynk est une application non seulement très populaire, mais de très bonne qualité. Cependant, lorsque je suis accueilli par un écran me demandant de me connecter à une infrastructure et donc de créer un compte pour quelque chose d'aussi simple que de contrôler une carte Arduino en Bluetooth, j'ai le réflexe d'immédiatement fermer l'application et la désinstaller...

Certes, il est possible d'installer son propre serveur (en Java) pour se passer de celui de Blynk, mais c'est une approche que je n'apprécie pas même si, oui, en effet, ceci rend votre projet accessible de n'importe où, et qu'on se trouve aujourd'hui dans un monde de cloud, d'objets connectés (IoT) et aussi de fuites de données privées... Peut-être reviendrai-je sur le sujet d'ici quelque temps, mais pour l'heure le « *Connectez-vous avec Facebook* » en écran d'accueil de l'application m'a clairement effarouché.

Enfin, nous avons, non pas une application, mais une solution pour créer facilement des applications Android : MIT App Inventor 2. Je dois avouer que là, ma réticence est plus technique et fortement en lien avec ma vision de ce qu'est la programmation. MIT App Inventor 2 se présente sous la forme d'une application à installer sur son smartphone et d'un environnement de développement en ligne (sur le Web) permettant de créer des applications puis de les tester et les déployer sur son smartphone. Je suis déjà réticent quant à une telle architecture, puisque forcément il faut créer un compte, mais cela se cumule avec la façon que propose cet environnement de créer une application : non pas un langage, mais une représentation visuelle du programme façon Scratch (créé également au MIT Media Lab). Trouvez-moi vieux jeu si vous voulez, mais j'ai cette fâcheuse tendance à penser que le fait d'agencer des pièces de puzzle n'est pas « programmer ». Une imbrication de pièces multicolores peut effectivement être très attrayante, voire inspiratrice, pour une jeune audience mais, selon moi, ce n'est pas une solution viable à long terme pour créer un programme lisible, facile à modifier et à maintenir.

Apprendre à utiliser Java et l'environnement de développement Android demande du temps et des efforts, certes, mais cet apprentissage sera de toute façon nécessaire lorsqu'il s'agira de passer d'une solution « puzzle » à la création d'une vraie application. **DB**

CONNECTEZ UN MODULE LCD À VOTRE RASPBERRY PI POUR AFFICHER SON ADRESSE RÉSEAU

Denis Bodor

Pour certains projets, un système d'affichage tel un écran HDMI, qu'il s'agisse d'un moniteur PC ou d'un petit écran 7 pouces, est parfois surdimensionné. Même un écran TFT de quelques pouces de diagonale est plus qu'il n'en faut s'il s'agit simplement d'afficher un état ou une adresse réseau. Dans ce genre de situations, un afficheur LCD alphanumérique est bien suffisant. Il faudra simplement prendre quelques précautions pour ne pas risquer d'endommager votre très chère framboise...

L'écran LCD alphanumérique fait partie de la boite à outils standard de l'utilisateur Arduino, car il s'agit d'une solution simple et efficace pour afficher des informations basiques, voire pour créer une interface utilisateur simpliste. Ce type de modules repose, dans la très grande majorité des cas, sur un circuit intégré HD44780 ou compatible. Son utilisation est donc quelque chose de standard et relativement facile à mettre en œuvre. Seul point noir au tableau, ces modules sont le plus souvent faits pour fonctionner en 5V alors que la Raspberry Pi, elle, utilise une tension de 3,3V.

Il y a cependant une astuce, car le problème ne se pose réellement qu'en cas de lecture sur les broches de la carte. Il ne faut, en effet, jamais imposer une tension supérieure au maximum sur les broches d'une Pi (elle n'est pas tolérante au 5V). L'inverse en revanche, à savoir utiliser une tension 3,3V là ou 5V sont attendus, fonctionnera parfaitement bien. L'astuce consiste donc à faire en sorte que la Raspberry Pi soit la seule à contrôler les tensions et les états et que le module d'affichage reste muet comme une carpe (ou compatible).

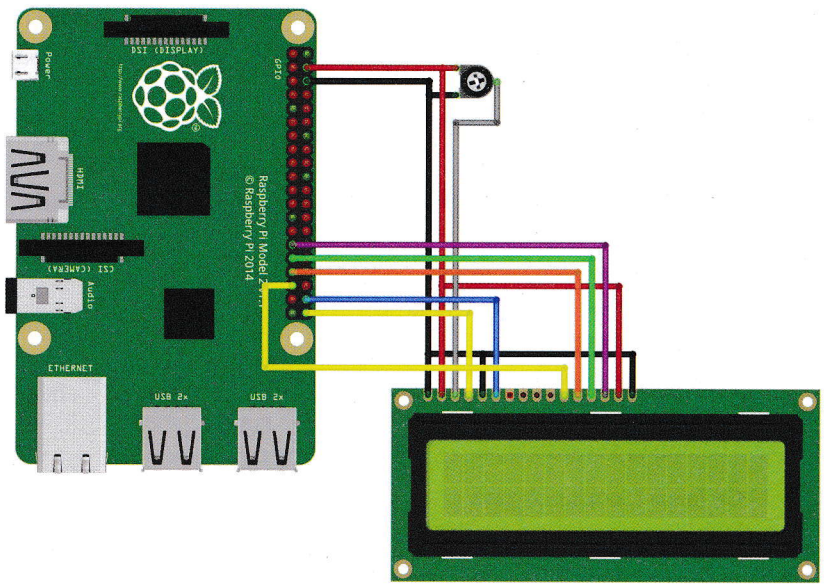
Les ordres envoyés par la Raspberry Pi, quant à eux, seront bel et bien reçus et acceptés par le module d'affichage car, quelle que soit la technologie utilisée par le circuit intégré contrôlant l'affichage, les niveaux de tensions utilisés seront bien dans les plages acceptées. En technologie TTL, la plus ancienne, un niveau logique est considéré comme bas avec une tension entre 0V et 0,8V. Pour un niveau logique haut, ce sera entre 2V et la tension d'alimentation, notée VCC (qui en TTL est obligatoirement en 5V +/- 10%). Dans le cas de la technologie CMOS en revanche, un niveau logique bas correspond à une tension entre 0V et 1/3 de la tension d'alimentation (noté VDD) et un niveau logique haut doit correspondre à une tension entre 2/3 de VDD et VDD. Avec nos signaux 3,3V donc, nous arrivons tout juste aux 2/3 de 5V (+/- la tolérance).

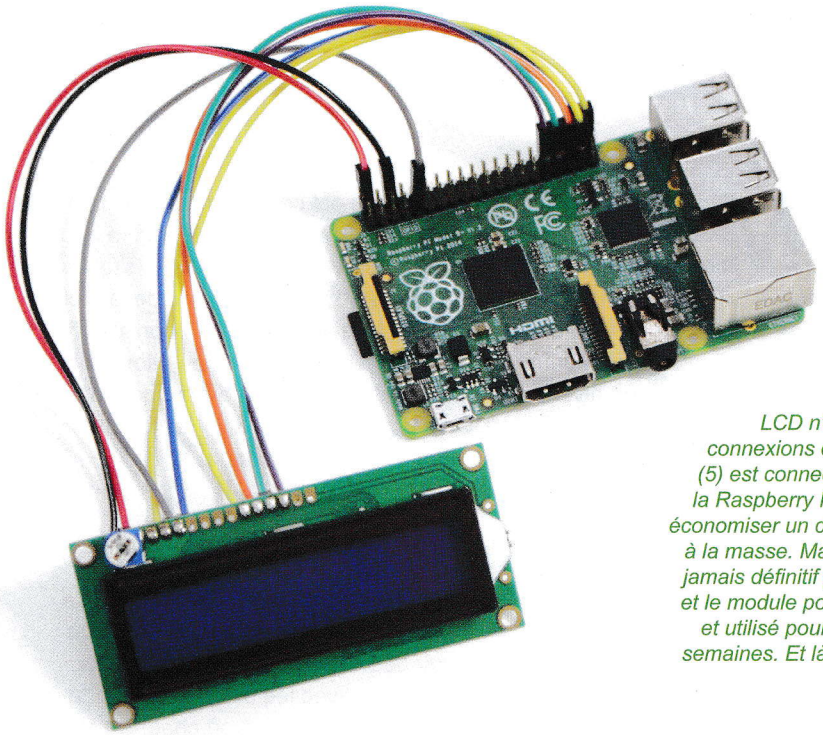
1. CONNEXION DE L'AFFICHEUR

L'afficheur LCD alphanumérique sera ici configuré en mode 4 bits afin de minimiser le nombre de connexions. En effet, ce type de module peut recevoir des ordres sous forme d'octets, soit envoyés en parallèle sur 8 bits avec les broches DATA de 0 à 7, soit sur 4 bits avec les broches de 4 à 7 où chaque octet sera donc coupé en deux et envoyé en deux fois.

Le point important ici tient en deux éléments :

- bel et bien alimenter le module en 5V,
- et attacher la broche 5 (R/W) déterminant la lecture ou l'écriture sur le module, à la masse. De cette façon, nous empêchons physiquement le module d'être utilisé en lecture et il n'imposera pas de tension sur les broches DATA (ou DB).





La connexion du module LCD n'est pas trop gourmande en connexions et câbles. Ici la broche R/W (5) est connectée à l'une des masses de la Raspberry Pi et nous pourrions encore économiser un câble en soudant cette ligne à la masse. Mais sait-on jamais, rien n'est jamais définitif pour ce genre de montage, et le module pourrait bien être déconnecté et utilisé pour autre chose d'ici quelques semaines. Et là j'aurai peut-être besoin de R/W...

Nous tentons, dans la mesure du possible de regrouper les connexions dans une certaine zone du connecteur de la Raspberry Pi, tout en évitant d'utiliser les broches avec des fonctions alternatives (série, SPI, i2c, etc.).

Le schéma de connexion peut sembler touffu, mais en réalité, il est grandement possible de simplifier les choses en établissant des ponts directement sur le module lui-même :

- attacher les connexions à la masse ensemble ;
- relier les deux connecteurs d'alimentation (module et leds de rétroéclairage) ;
- souder directement le potentiomètre de réglage du contraste sur le module.

Ce faisant, nous pouvons réduire les connexions à 8 câbles (alimentation, masse, RS, E, DATA 4 à 7).

Pour rappel, la nomenclature d'un module HD44780 ou compatible est :

- 1/GND : la masse ;
- 2/VCC : l'alimentation ;
- 3/VE : réglage du contraste ;
- 4/RS : sélection commande/données (*Register Select*) ;
- 5/RW : lecture/écriture (*Read/Write*) ;
- 6/E : validation données (*Enable*) ;
- 7 : DATA0 ;
- 8 : DATA1 ;
- 9 : DATA2 ;
- 10 : DATA3 ;
- 11 : DATA4 ;
- 12 : DATA5 ;
- 13 : DATA6 ;
- 14 : DATA7 ;
- 15 : led + ;
- 16 : led -.

2. UTILISATION AVEC PYTHON

Plusieurs solutions permettent de piloter le module d'affichage. Nous pouvons utiliser un programme dédié ou encore un module Python comme celui proposé par Adafruit. Je trouve cependant que cela est un peu démesuré et induit l'installation d'éléments qui ne sont pas strictement nécessaires. La voie choisie sera donc celle du contrôle direct de l'afficheur à l'aide d'un script Python sans avoir recours à des ressources extérieures. Ceci est parfaitement envisageable, car le module est relativement simple à piloter et nous évite de devoir installer quoi que ce soit d'autre. Tout ce dont nous avons besoin c'est le module `RPi.GPIO` présent par défaut dans Raspbian qu'il s'agisse de la version standard ou *Lite*.

Notre cas pratique sera ici d'écrire un petit script récupérant l'adresse IP associée à la connexion filaire de la carte (généralement fournie par une box ou un routeur/serveur DHCP) et l'affichant sur le module LCD de façon à rapidement connaître l'état de la connexion et l'adresse pour se connecter à distance (SSH).

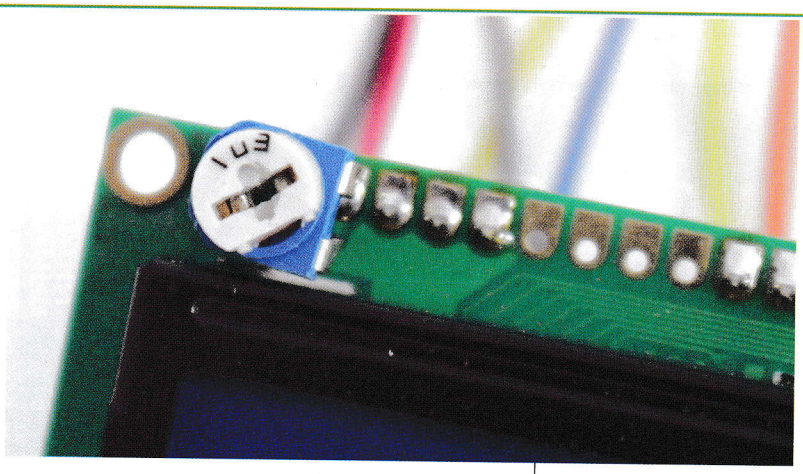
Voici notre script :

```
#!/usr/bin/python

# modules utiles
import RPi.GPIO as GPIO
import netifaces
from time import sleep

# création classe
class HD44780:
    # constructeur
    def __init__(self, pin_rs=40, pin_e=38, pins_db=[37, 35, 33, 31]):
        self.pin_rs = pin_rs
        self.pin_e = pin_e
        self.pins_db = pins_db

        # configuration des broches/GPIO
        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(self.pin_e, GPIO.OUT)
        GPIO.setup(self.pin_rs, GPIO.OUT)
        for pin in self.pins_db:
            GPIO.setup(pin, GPIO.OUT)

        self.clear()

    # Effacement afficheur
    def clear(self):
        self.cmd(0x33) # init 8 bits
        self.cmd(0x32) # init 8 bits confirmation
        self.cmd(0x28) # 4 bits - 2 lignes
        self.cmd(0x0C) # Pas de curseur
        self.cmd(0x06) # Incrémentation curseur
        self.cmd(0x01) # Efface écran

    # Envoi de commandes
    def cmd(self, bits, char_mode=False):
        sleep(0.005)
        bits=bin(bits)[2:].zfill(8)

        GPIO.output(self.pin_rs, char_mode)

        for pin in self.pins_db:
            GPIO.output(pin, False)

        for i in range(4):
            if bits[i] == "1":
                GPIO.output(self.pins_db[::-1][i], True)

        GPIO.output(self.pin_e, True)
        GPIO.output(self.pin_e, False)
```

```
        for pin in self.pins_db:
            GPIO.output(pin, False)

        for i in range(4,8):
            if bits[i] == "1":
                GPIO.output(self.pins_db[::-1][i-4], True)

        GPIO.output(self.pin_e, True)
        GPIO.output(self.pin_e, False)

    # Envoi de texte
    def message(self, text):
        for char in text:
            if char == '\n':
                self.cmd(0xC0) # second ligne
            else:
                self.cmd(ord(char),True)

    # Destruction objet
    def __del__(self):
        GPIO.cleanup()

# Utilisation comme script et non module
if __name__ == '__main__':
    lcd = HD44780()
    lcd.clear()
    lcd.message("Mon IP:\n%s" % \
    netifaces.ifaddresses('eth0')[netifaces.AF_INET][0]['addr'])
```

Notre script a une double utilité. C'est à la fois un outil nous permettant d'afficher l'adresse IP et une classe Python éventuellement utilisable pour d'autres scripts. Nous définissons ainsi un certain nombre de méthodes :

- **`__init__()`** est le constructeur qui a pour tâche de configurer les broches (GPIO) utilisées, avec une numérotation correspondant aux numéros de broches (**`BOARD`**) ;
- **`clear()`** se charge à la fois de l'initialisation du module et de l'effacement de l'écran. Nous devons envoyer un certain nombre de commandes au module, selon une technique décrite sur la page Wikipédia du HD44780. L'astuce consiste à pouvoir nous assurer que, dans tous les cas, nous passerons en mode 4 bits, quel que soit l'état dans lequel se trouve le module (8 bits, en plein milieu d'un octet, ou déjà en 4 bits) ;
- **`cmd()`** est la méthode permettant d'effectivement envoyer des commandes et des données à l'afficheur (si **`char_mode`** est **`True`**). C'est elle qui va changer l'état des broches par série de 4 bits et valider les données avec la broche *Enable* ;
- **`message()`** nous permet d'envoyer du texte tout en interceptant un éventuel défilement de ligne (**`\n`**) pour basculer sur la seconde ligne de l'afficheur. Cet envoi se fait en utilisant la méthode **`cmd()`** ;
- et enfin **`__del__`** est le destructeur qui se charge, lorsque l'objet est détruit (typiquement la fin du script), de signaler au système que les broches ne sont plus utilisées. Sans cela, un message d'avertissement serait affiché à chaque exécution (puisque les broches n'auront pas été explicitement « libérées »).



En soudant directement un potentiomètre de 10 Kohms sur le module, non seulement cela rend l'ensemble plus compact, mais en plus, nous économisons trois câbles et sans doute une platine à essais.

La majeure partie de ce script consiste en la création d'une classe **HD44780** permettant ainsi de créer des objets de ce type et d'utiliser leurs méthodes. C'est précisément ce que nous faisons sur les cinq dernières lignes du script en testant le contenu de **__name__**. Si ce contenu est importé dans un script Python (comme un module), cette variable contiendra le nom du module. Mais si c'est l'interpréteur qui exécute ce code, **__name__** contiendra **__main__**. Ceci permet de considérer du code uniquement si le script est exécuté directement et non importé dans un autre script.

Dans ce morceau de code, nous instancions un objet **lcd** de classe **HD44780** avec les valeurs par défaut correspondant à nos connexions. Nous utilisons ensuite **clear()** pour initialiser, configurer et effacer l'affichage puis nous affichons un texte composé de « Mon IP: », un saut de ligne, et l'adresse IP de l'interface filaire de la Raspberry Pi. Notez que nous spécifions ici **eth0**, mais que cela fonctionnerait exactement de la même manière avec une interface réseau Wifi comme **wlan0**.

Bien entendu, au lancement du script (après enregistrement sous le nom **LCDIP.py** et avec les permissions ajustées via **chmod +x LCDIP.py**), notre module s'anime et affiche le texte spécifié. Si tel n'est pas le cas, vérifiez les connexions ainsi que les valeurs des broches utilisées dans le script et guettez un éventuel message d'erreur de Python.

Pour éviter d'éventuels problèmes de lecture de l'adresse, nous pouvons modifier le script et en particulier l'appel à **message()** ainsi :

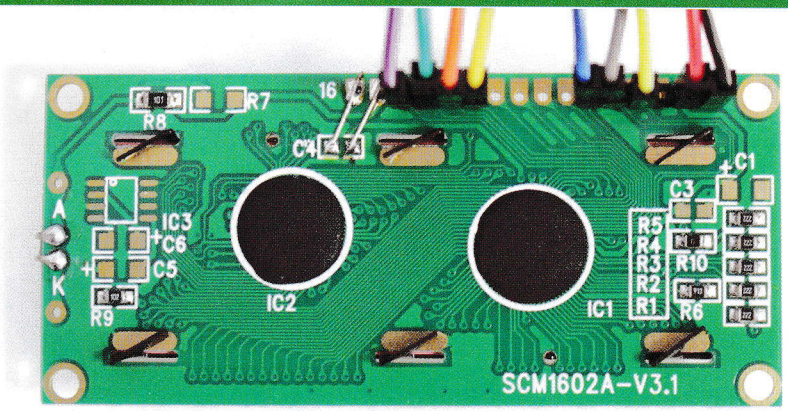
```
try:
    lcd.message("Mon IP:\n%s" %  \
    netifaces.ifaddresses('eth1')[netifaces.AF_INET][0]['addr'])
except:
    lcd.message("Mon IP:\nERREUR !")
```

De cette façon, en cas d'interface réseau inexistante par exemple, le problème sera signifié directement sur l'afficheur et non sous la forme d'un message en ligne de commandes.

3. INTÉGRER NOTRE SCRIPT AU SYSTÈME

Nous disposons maintenant d'un script de taille raisonnable capable de nous afficher l'information souhaitée via un simple appel ponctuel. L'idée n'est cependant pas de devoir lancer nous-mêmes ce script, car si nous pouvions le faire, c'est que nous aurions la main sur le système et aurions par la même occasion d'autres moyens d'obtenir une adresse IP. L'objectif se résume dans le fait que ce script doit être lancé automatiquement au démarrage du système.

Ceci cependant ne peut pas se faire n'importe quand. En effet, lorsque le système GNU/Linux de votre Pi démarre, une série de programmes sont lancés. Certains ne font qu'un passage éclair alors que d'autres restent en mémoire et fonctionnent en « tâche de fond », ce sont des démons (ce qui, en principe, est une mauvaise traduction de *daemon*, un terme



En plus de la soudure du potentiomètre, il est possible de s'amuser à chercher une masse et une tension d'alimentation sur le module à l'aide d'un multimètre. Ici l'emplacement C4 était vide et proposait justement ces tensions. Deux petits ponts avec des pattes de composants et voilà encore deux broches joyeusement économisées !

choisi par les chercheurs du projet MAC du MIT dans les années 60, davantage en rapport avec la mythologie hellénistique grecque et les entités spirituelles œuvrant en coulisse, qu'avec la notion chrétienne de démons).

Cette organisation complexe de lancement de programmes et de script de toutes sortes est historiquement la tâche du premier programme lancé après le noyau : **`init`** et plus précisément l'init SysV (System 5). Aujourd'hui, les systèmes GNU/Linux récents ou à jour, comme celui utilisé par votre Raspberry Pi, reposent sur un autre système d'initialisation, appelé *systemd*, composé non pas d'un seul programme, mais de plusieurs dizaines.

Alors que la mise en place d'un script permettant le lancement d'une simple commande au démarrage était d'une simplicité enfantine avec le bon vieux init SysV, l'opération qui nous occupe ici est sensiblement plus complexe (oui, je suis un rien nostalgique et vois dans systemd une véritable usine à gaz). Ainsi, pour ajouter notre lancement de **`LCDIP.py`** dans la configuration, nous devons avant toutes choses créer un fichier décrivant un « service », que nous appellerons **`/etc/systemd/system/lcdip.service`** :

```
[Unit]
Wants=network-online.target
After=network.target network-online.target ssh.service

[Service]
Type=oneshot
ExecStart=/usr/local/bin/LCDIP.py

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

Un tel « service » est défini par une unité, elle-même décrite en détail dans ce fichier. Celui-ci est divisé en trois sections, **`[Unit]`**, **`[Service]`** et **`[Install]`**. La première section décrit l'unité elle-même et en particulier ses dépendances vis-à-vis des autres unités. Ici, nous voulons que notre unité *veuille* **`network-online.target`**, signifiant que le réseau est configuré, fonctionnel et en ligne.

Notre unité sera donc utilisée après cette configuration ainsi qu'après la mise en route du service SSH (il n'y a aucune raison d'afficher l'adresse de la Pi si la connexion distante avec SSH n'est pas encore possible).

La seconde section décrit le service qui ici consiste en un simple appel à un programme (**`oneshot`**) avec comme commande à utiliser le chemin complet vers notre script Python.

Enfin, nous avons l'imbrication avec le reste des unités lorsque nous installerons ce service. La dernière ligne du fichier provoque, à ce moment, une dépendance de **`multi-user.target`** envers notre unité (**`lcdip.service`**). De ce fait, lors de l'installation dans le système, **`multi-user.target`** *voudra* **`lcdip.service`**.

Pour tester ce service, nous devons enfin copier **`LCDIP.py`** dans **`/usr/local/bin`** avec **`sudo cp LCDIP.py /usr/local/bin`**, car ceci ne peut être fait qu'avec les permissions du super-utilisateur.

À ce stade, nous avons configuré le service et pouvons l'utiliser, mais celui-ci n'est pas installé dans le système, rien ne se passera en redémarrant la Pi. Nous pouvons cependant tester le fonctionnement de notre configuration avec :

```
$ sudo systemctl start lcdip
```

Aucun message ne devrait s'afficher, mais l'écran du module devrait présenter le message attendu ou, au moins, le rafraîchir (vous pouvez éventuellement déconnecter/reconnecter la broche d'alimentation du module avant de valider la commande).

Notre service fait son travail correctement, nous n'avons plus qu'à l'installer officiellement avec :

```
$ sudo systemctl enable lcdip
Created symlink from
/etc/systemd/system/multi-user.target.wants/lcdip.service
to /etc/systemd/system/lcdip.service.
```

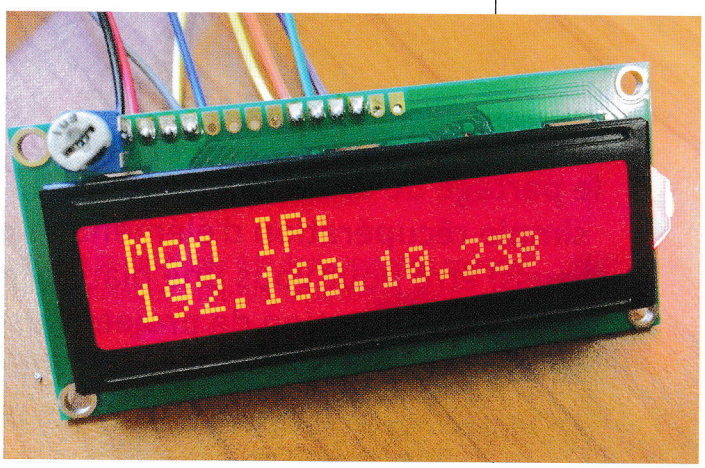
Comme le message l'indique, cette intégration se fera via une création d'un lien symbolique directement dans `/etc/systemd/system` et lors du prochain démarrage, notre `LCDIP.py` sera lancé dès les autres services nécessaires activés. Un simple coup d'œil à l'afficheur nous permettra ainsi de connaître l'adresse pour nous connecter à la Pi.

CONCLUSION

Le but de cet article, en plus de son aspect pratique, était surtout d'avoir l'occasion de montrer qu'en comprenant bien le fonctionnement des modules utilisés, nous pouvons parfaitement adapter leur comportement à nos besoins. Certes, l'afficheur à base de HD44780 n'est pas utilisé au maximum de son potentiel du fait de se priver de lecture et de limiter l'envoi de données au mode 4 broches, mais ça marche. Si vous êtes comme moi et vivez votre passion pleinement, je ne doute pas un instant que vous devez avoir en votre possession de tels modules en plusieurs exemplaires.

À quoi bon en acquérir de nouveaux, compatibles 3,3V, si un peu de bidouille vous permet d'utiliser les mêmes que ceux que vous mettez en œuvre pour vos projets Arduino ? Ce n'est pas tant une question de budget, étant donné le prix sans cesse en baisse de tous ces éléments, que le fait de passer commande et de devoir attendre en trépignant à côté de la boîte à lettres. Lorsqu'on a une idée en tête, mieux vaut la mettre en pratique le plus vite possible, car « délai de livraison » rime souvent avec « passage à autre chose ». Vous le savez aussi bien que moi, un projet en appelant un autre, lorsque les modules arrivent enfin, il n'est pas rare qu'on ne sache même plus pourquoi ils ont été commandés ou que l'excitation soit tout simplement retombée, et qu'autre chose nous trotte dans la tête...

Une dernière remarque à propos du script et de la non-utilisation d'un module Python comme ceux proposés en ligne : la compacité du code et la non-dépendance à ces modules ont leurs avantages, mais elles ont aussi leurs inconvénients. Ici, notre script est conçu pour traiter uniquement un afficheur de deux lignes de 16 caractères. Pour l'adapter à un 4×20, il faudra modifier le code et changer la procédure d'initialisation. Bien entendu, cela dépendra fortement de l'utilisation finale du module et du script. Dans ce cas précis, avec une adresse IP, faisant forcément 15 caractères au maximum, augmenter la taille de l'afficheur n'a rien d'intéressant. La seule évolution possible sera d'opter pour un affichage LCD d'une ligne de 16 caractères. Chose qui ne demande qu'une très légère modification, que je vous laisserai le plaisir d'expérimenter par vous-même. Après tout, adapter et plier un montage à sa volonté, c'est avant tout apprendre... DB



Les afficheurs LCD de deux lignes de 16 caractères, avec un rétroéclairage rouge, sont mes préférés. Ici la photo ne rend pas fidèlement compte des couleurs effectives de l'affichage. À l'œil nu, le fond est bien plus sombre et les caractères bien moins jaunes...

PARTAGEZ VOS PROJETS ET VOS CRÉATIONS SUR GITHUB

Denis Bodor

Vous connaissez sans doute GitHub, le site communautaire permettant le partage de code, de croquis et de bibliothèques. Mais au-delà de la simple récupération d'éléments proposés par d'autres, GitHub peut aussi être pour vous une plateforme pour partager vos travaux et aider à l'amélioration de projets existants. Mais pour cela, il faut savoir se servir de Git, un outil fantastique qui pourra vous aider pour vos propres projets...

GitHub est l'endroit rêvé pour qui souhaite apprendre ou simplement réutiliser tout ou partie du travail d'autrui. C'est un site d'hébergement et de gestion de codes, de toutes sortes de codes. Le magazine par exemple dispose de sa page sur GitHub et vous y retrouverez à chaque numéro tous les éléments et fichiers en relation avec les articles qui s'y trouvent. Il est ainsi possible de récupérer les croquis Arduino, les scripts Bash et Python ou encore les fichiers de configuration trop longs à ressaisir à la main.

Il est ainsi possible de simplement aller sur le site, choisir un numéro de magazine et récupérer un fichier ou une archive complète pour un numéro donné. Ceci ressemble à un simple site de téléchargement, mais ne vous y trompez pas, c'est bien plus que cela. GitHub est une interface web pour quelque chose de vraiment plus puissant : Git, créé par Linus Torvalds (oui, oui, le Linus de Linux) et Junio Hamano en 2005, pour remplacer un autre outil, BitKeeper, alors en place pour le développement du noyau Linux et dont l'utilisation gracieuse n'avait plus cours.

GitHub n'est pas le seul site à proposer ce type d'hébergement, mais c'est le plus populaire. De la même façon, Git, n'est pas non plus le seul de sa catégorie, avec comme « confrère » CVS, Mercurial, Darcs, Bazaar ou SVN, mais là encore c'est le plus utilisé.

Git est ce que l'on appelle un logiciel de gestion de versions. Illustrer l'utilité d'un tel outil n'est pas difficile et vous n'avez qu'à observer votre manière de gérer vos fichiers. Avant même de parler de collaboration et de développement en communauté, un outil comme Git répond à un besoin simple : vous travaillez sur un croquis Arduino ou un code Python pour votre bien aimée Pi. Au fil du temps, vous ajoutez des lignes, en supprimez, en modifiez. Comme vous avez peur de perdre l'ancienne version, lors des sauvegardes importantes, vous en modifiez le nom. `capteuruv`, `capteuruv2`, `capteuruv_ok`, `capteuruv_lcd`, `capteuruv_ok`, ah zut, existe déjà, `capteuruv_ok2` alors... etc.

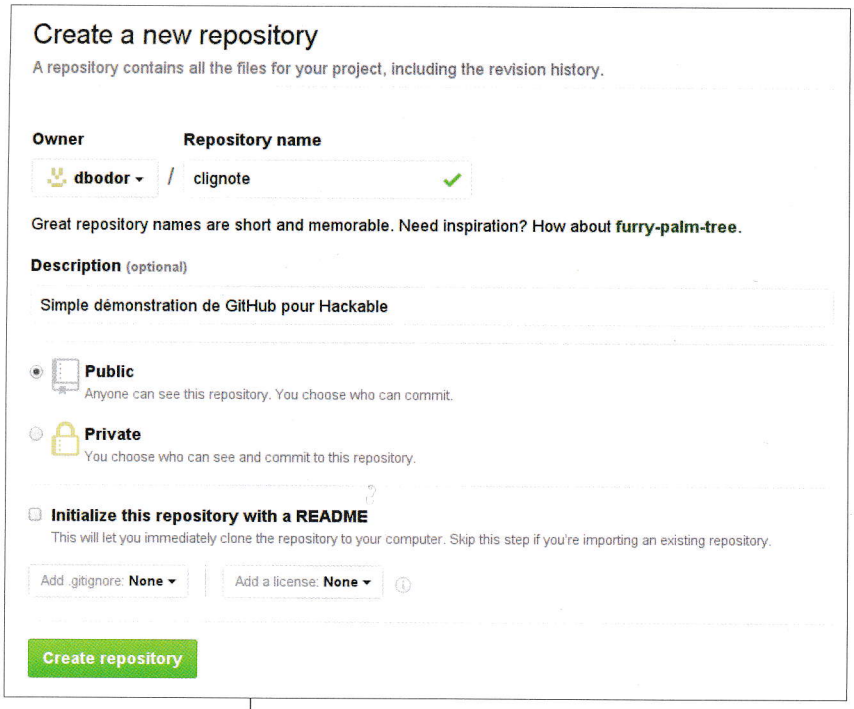
Et là, c'est le drame ! Quelque chose qui marchait avant ne fonctionne plus et la réponse est... quelque part dans une pile de fichiers. Vos seules pistes sont des noms de fichiers tordus, les dates et heures de modifications et une baguette de sourcier achetée au marché aux puces la semaine dernière pour faire plaisir à pépé... C'est pas gagné !

Ne vous inquiétez pas, voilà bien quelque chose qu'on a tous fait et qu'on fait encore tantôt, avec des croquis, des scripts, des images, des documents... Pour des petites choses, cela peut passer, mais lorsqu'il s'agit de gérer les fichiers de projets comme le noyau Linux, le navigateur Firefox ou une suite bureautique comme LibreOffice, ce n'est pas possible. Ces projets utilisent des outils pour régler le problème, et si c'est bon pour eux, c'est bon pour nous !

Git permet, plutôt que de créer 25 copies d'un même projet, de tenir un suivi des changements et modifications à votre place et avec peu d'intervention de votre part. Les changements sont archivés, datés et gérés. Vous pouvez revenir en arrière, comparer les changements, créer des versions parallèles... et finalement partager votre travail en ligne et même recevoir des contributions d'autres personnes ayant amélioré votre création.

1. GITHUB, TÉLÉCHARGEMENT, COMPTE ET TOUT CE QU'IL FAUT SAVOIR

Cet article ne va pas se substituer à la documentation en ligne ou aux tutoriels déjà présents sur GitHub, mais simplement vous donner les bases pour démarrer. Sachez également qu'un compte GitHub, ou sur un autre site du même type comme BitBucket, n'est pas une obligation pour se servir et profiter de Git. C'est simplement une façon de placer vos codes et scripts sur un serveur distant, de rendre publics vos travaux et de participer à d'autres projets.



La création d'un dépôt se fait très facilement sur GitHub. Il suffit de spécifier un nom, une description courte et le type de dépôt choisi. Dès l'instant suivant sa création, il sera prêt à être utilisé, cloné, synchronisé...

Avant toutes choses, il faudra vous créer un compte GitHub. Cette opération est gratuite et nécessite la saisie de quelques informations de base comme un pseudo, une adresse mail, etc. Vous pourrez ensuite choisir le type d'hébergement que vous souhaitez. Deux solutions sont possibles : un nombre illimité de dépôts (*repositories*) publics gratuits ou un nombre illimité de dépôts privés et publics (au choix, au cas par cas) pour 6,23€ par mois.

Un dépôt est un emplacement unique pris en charge par les fonctionnalités de suivi de Git. *Hackable*, par exemple, dispose d'un dépôt par numéro du magazine et les modifications qui s'y rapportent sont gérées par numéro. Partez du principe qu'un projet = un dépôt.

Un dépôt public est visible de tous et figurera dans la liste de vos dépôts attachés à votre compte. Seuls vous et les personnes que vous autorisez pourrez apporter des modifications dans ces dépôts, mais vos fichiers seront lisibles et téléchargeables. Inversement, un dépôt privé ne sera visible que par vous ou par ceux que vous aurez désignés. Ils sont généralement utilisés pour des projets personnels ou restreints à une équipe, qui ne doivent pas être visibles de tous. Encore une fois, le fait d'opter pour la gratuité et donc uniquement des dépôts publics ne vous empêche en rien d'utiliser Git. Mais lorsque vous utiliserez GitHub, vos fichiers et projets seront disponibles et accessibles à tous.

Personnellement, je pense que la meilleure façon de faire est de rendre publics ses travaux, même modestes. Non seulement ceci permet d'aider votre prochain, mais en plus, cela vous permet de profiter de l'expérience des autres programmeurs qui pourront vous signaler des erreurs ou proposer des améliorations et corrections.

BitBucket utilise une approche différente. Les dépôts peuvent être privés ou publics gratuitement, mais les dépôts privés ne peuvent avoir plus de 5 utilisateurs. Au-delà, c'est payant. Enfin, GitLab, un projet en logiciel libre, propose également de l'hébergement, gratuit et illimité, mais avec une version « *Enterprise Edition* » incluant une assistance technique et des fonctionnalités professionnelles supplémentaires pour $39 par an et par utilisateur. GitHub est cependant le site le plus populaire, en particulier pour les projets publics, et c'est la raison pour laquelle je base cet article sur ce site en particulier (et parce que le logo est adorable, un *octocat*, mi-chat, mi-pieuvre).

Une fois votre inscription faite, vous recevrez un e-mail vous permettant de confirmer votre adresse mail en cliquant sur le lien inclus. Ce faisant, vous arriverez sur une page vous permettant de lire le guide du débutant (en anglais) ou démarrer un nouveau projet. Vous pourrez également agrémenter votre profil de différentes informations (nom, photo, adresse de site, biographie, emplacement géographique, etc.).

2. UN TOUT PETIT PEU DE JARGON

Vous voici maintenant propriétaire d'un compte GitHub. Ceci vous permet de faire plus que vous ne pouviez en visitant le site anonymement. Vous pouvez, par exemple, faire une copie d'un dépôt public d'un autre utilisateur. Ceci s'appelle un **fork**, vous **forkez** le projet de quelqu'un d'autre.

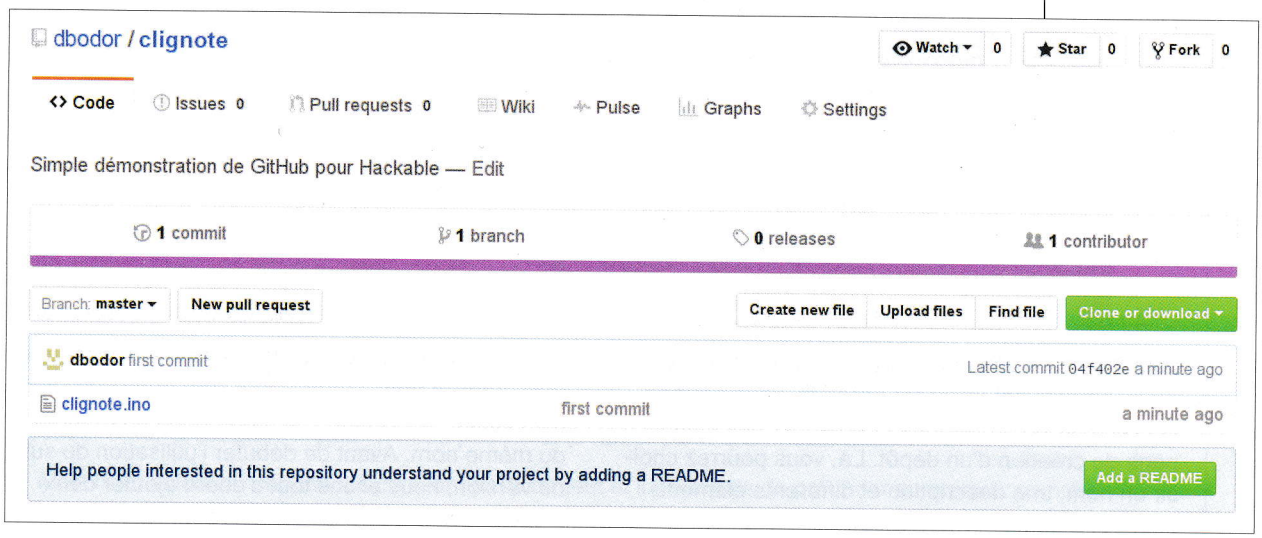
Pourquoi faire une telle chose ? Ceci vous permet dans un premier temps d'avoir votre copie à vous du projet sur GitHub. Une fois la copie effectuée sur votre ordinateur, via ce qu'on appelle un **clone**, vous pourrez y faire des modifications tout en utilisant le système de gestion de versions. Votre copie apparaîtra sur le site comme un **fork** du dépôt original et les utilisateurs sauront que c'est une version « personnelle » d'un projet existant, qui poursuit une vie parallèle.

Chaque modification enregistrée dans votre dépôt est appelée un **commit**. La logique est la suivante : vous changez le contenu d'un ou plusieurs fichiers et lorsque vous estimez qu'il est temps de faire une « sauvegarde », vous **commitez** ces changements, accompagnés d'un message décrivant les modifications. À chaque **commit**, une trace est conservé par Git, incluant les modifications apportées, la date et heure, les fichiers concernés, etc. À n'importe quel moment, vous pouvez voir les différences entre les **commits** ou entre un **commit** et l'état actuel des fichiers. Un lot de différences de ce type est généralement désigné par le terme **diff**.

Si vos modifications sur le dépôt concernent des améliorations ou des corrections de bugs, sans doute souhaiterez-vous les partager avec l'auteur original. Une fois vos changements **commités** dans votre dépôt sur votre ordinateur, vous mettrez à jour la copie sur votre compte GitHub : vous poussez les changements sur le dépôt géré par le site ou en d'autres termes, vous **pushez** les modifications dans le dépôt distant. À ce stade, vous pourrez signaler votre intention au propriétaire du dépôt originalement **forké**, en faisant un **pull request** (horriblement traduit en « demande de tirage »). Ceci est une notification l'invitant à prendre (**pull**) vos modifications et à les répercuter dans son dépôt.

Lors d'un **pull request** et avant que les modifications ne soient répercutées, vous pourrez discuter avec lui des changements apportés directement sur

Dès lors qu'on aura poussé les changements depuis notre dépôt vers celui présent sur GitHub, notre projet sera visible. Ceci comprend non seulement les fichiers qui le composent, mais également l'historique des modifications, les commentaires de commit, etc.



GitHub et selon l'issue de la discussion, vos modifications seront acceptées ou non. Il vous sera également possible d'apporter des changements consécutifs à la discussion, en faisant des modifications, les **commitant** dans votre dépôt et les **pushant** sur GitHub.

Il est également possible que dans vos interactions sur GitHub, quelqu'un vous signale un problème (**issue** en anglais). Le fait de posséder un compte GitHub vous permet de faire de même dans le projet de quelqu'un. Le site prend en charge le suivi de ces signalements, jusqu'à leur résolution et propose ainsi, par dépôt une liste de problèmes à résoudre d'une façon ou d'une autre. Le fait de lister les problèmes d'autres projets est aussi l'occasion pour vous de les corriger ou de participer à la conversation.

Idéalement, si vous utilisez par exemple une bibliothèque Arduino récupérée sur GitHub et rencontrez un problème déjà signalé, mais non corrigé, vous pouvez **forker** le dépôt, modifier la bibliothèque pour régler le problème, **commiter** et **pusher** les changements pour enfin faire un **pull request** afin que votre contribution aide les autres utilisateurs. Ces « problèmes » peuvent être des bugs, des souhaits d'utilisateurs ou simplement des questions posées par quelqu'un qui rencontre des difficultés.

Le mécanisme de **fork** et de **pull request** est la base de l'aspect collaboratif sur GitHub.

3. PRÉLIMINAIRES

Une fois votre compte GitHub actif, deux options s'offrent à vous : créer un dépôt ou forker le dépôt de quelqu'un. Dans les deux cas, cela aura pour effet de créer un dépôt dans votre compte GitHub. Pour un fork, un simple clic sur le bouton en haut à droite sur la page d'un projet fera l'affaire. Vous serez ensuite automatiquement redirigé vers votre page pour ce dépôt.

Pour la création d'un nouveau dépôt, toujours en haut à droite d'une page, vous disposez d'une icône « + ». En cliquant sur cette dernière, vous pourrez choisir « *New repository* » et arriverez alors sur la page de création d'un dépôt. Là, vous pourrez choisir un nom, une description et différents éléments pour votre dépôt.

Le nom du dépôt correspondra généralement au nom du répertoire lorsque vous le clonerez de GitHub sur votre ordinateur (ou Pi). Il est donc de bon ton de choisir un nom court, simple à retenir, et dans le cas d'un croquis Arduino, du même nom que le fichier `.ino` que vous placerez à l'intérieur du répertoire.

Pour notre exemple, nous allons créer un dépôt `clignote` destiné à un croquis Arduino ultra-simpliste. Une fois le dépôt créé, il sera vide, mais néanmoins existant et ce sera le point de départ de nos manipulations.

Dernier point avant de commencer, même si vous n'utilisez pas Linux, je vous recommande de lire la partie suivante, car j'y détaillerai des principes et des mécanismes qui s'appliqueront également dans la partie traitant de Windows.

4. GIT SOUS LINUX OU SUR RASPBERRY PI

Sous Linux, que ce soit sur la Raspberry Pi ou sur PC, le plus simple pour profiter de Git est de tout simplement utiliser la commande `git` généralement installée via le paquet du même nom. En guise d'exemple, nous allons utiliser un croquis Arduino très simple puisqu'il s'agit de celui permettant de faire clignoter la led intégrée à la carte :

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Ce croquis sera enregistré dans le carnet de croquis, sous le nom `clignote` et sera donc constitué d'un fichier `clignote.ino` placé dans un répertoire du même nom. Avant de débuter l'utilisation du suivi de version, nous allons tout d'abord ajouter deux éléments de configuration à notre installation de

Git en précisant un nom et une adresse mail. Ceci permettra de signer les commits automatiquement sans que Git ne cherche à déterminer seul (et mal) les informations à partir du nom d'utilisateur et celui du système :

```
% git config --global user.name dbodor
% git config --global user.email "dbodor@hackable.fr"
```

Ces informations seront automatiquement stockées dans un fichier **.gitconfig** dans votre répertoire personnel :

```
[user]
        name = dbodor
        email = dbodor@hackable.fr
```

Il vous sera possible d'ajouter d'autres éléments dans ce fichier de configuration pour personnaliser le comportement de Git sur la machine.

Une fois votre croquis enregistré, nous pouvons nous pencher sur sa prise en charge avec Git. À ce stade, nous avons un simple répertoire contenant un fichier, ainsi qu'un dépôt Git vide sur GitHub. La première chose à faire pour profiter des services de Git est de transformer notre répertoire en dépôt Git :

```
% cd emplacement/croquis/arduino

% git init
Dépôt Git vide initialisé dans /home/denis/sketchbook/clignote/.git/
```

Un sous-répertoire **.git** sera créé contenant toutes les informations utiles pour le suivi de versions. Ce répertoire sera totalement invisible pour l'environnement Arduino et ne perturbera en rien son fonctionnement.

Nous avons à présent un dépôt Git, mais nous n'avons pas spécifié, parmi son contenu, quels sont les fichiers à prendre en compte. Nous ajoutons (*add*) donc explicitement un fichier à gérer avec :

```
% git add clignote.ino
```

À tout moment, nous pouvons invoquer **git status** pour connaître l'état du dépôt :

```
% git status
Sur la branche master
Validation initiale
Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)
        nouveau fichier: clignote.ino
```

Ici, Git nous indique qu'un fichier a été ajouté depuis l'initialisation du dépôt. Aucun changement n'a encore été validé (ou commité), nous avons simplement indiqué à Git la présence d'un nouveau fichier à suivre. Celui-ci est alors indexé et nous pourrions le retirer ou le désindexer avec **git rm**.

Mais l'état actuel du dépôt nous convient parfaitement et il est temps de procéder au premier commit. Ce sera la première borne de la vie de notre projet et de notre dépôt. Pour commiter, nous utilisons :

```
% git commit -m "first commit"
[master (commit racine) 04f402e] first commit
1 file changed, 10 insertions(+)
create mode 100644 clignote.ino
```

L'option `-m` nous permet d'ajouter un message de commit détaillant les changements. Pour des dépôts qui ne quitteront jamais votre système, il n'y a pas de problème à spécifier des messages dans votre langue natale. Internet cependant parle majoritairement anglais et si vous souhaitez être explicite quant à la vie de votre projet et permettre à d'autres personnes de contribuer, il est préférable d'utiliser la langue de Shakespeare (même si cela passe par Google Traduction, c'est mieux que rien). Traditionnellement, le premier commit est généralement accompagné du message « *first commit* » puisqu'il n'y a pas grand-chose à détailler.

À ce stade, votre répertoire ou dépôt Git fonctionne parfaitement et gère les fichiers que vous avez ajoutés. Vous pouvez par exemple lister les commits :

```
% git log
commit 04f402e9aebe6fd2ebad69cecdb37c4dd79f522f
Author: dbodor <dbodor@hackable.fr>
Date:   Wed Sep 14 13:51:02 2016 +0200
    first commit
```

Mais ceci n'existe que sur votre système. Sur GitHub, vous disposez d'un autre dépôt pour ce projet, mais les deux ne sont pas encore reliés. Le principe de fonctionnement de Git repose sur le fait que la gestion soit décentralisée. Un serveur comme celui de GitHub n'est pas nécessaire. Ce n'est d'ailleurs même pas un serveur, c'est un endroit où réside et vit une autre « instance » du dépôt. Cette décentralisation permet de voir chaque clone d'un dépôt comme une copie indépendante et tous les dépôts sont égaux.

Pour ouvrir votre projet au public et le rendre accessible de n'importe où, il faut synchroniser votre copie locale avec celle sur GitHub. Nous allons pusher les changements depuis le dépôt local vers le dépôt GitHub. Pour ce faire, nous devons indiquer à notre dépôt local l'existence d'un dépôt distant (*remote*) :

```
% git remote add origin https://github.com/dbodor/clignote.git
```

Nous ajoutons (**add**) un dépôt distant (**remote**) appelé **origin** et ayant pour URL **https://github.com/dbodor/clignote.git**, celle-là même fournie par GitHub au moment de la création du dépôt via le site. Cette ligne ne fait que configurer la présence du dépôt distant, il n'y a pas de connexion. Pour répercuter les changements, nous devons pusher :

```
% git push -u origin master
Username for 'https://github.com': dbodor
Password for 'https://dbodor@github.com':
Counting objects: 3, done.
Delta compression using up to 16 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 297 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
```

```
To https://github.com/dbodor/clignote.git
* [new branch]      master -> master
La branche master est paramétrée pour suivre la branche distante master
depuis origin.
```

La commande **push** permet de mettre à jour le dépôt distant par rapport à ce qui est commité dans l'état actuel du dépôt local. Nous spécifions la destination par son nom, **origin** et précisons la branche à utiliser.

La notion de branche dans la vie d'un dépôt permet, tout comme avec la métaphore arboricole, de travailler dans plusieurs directions pour un projet, avec des séries de modifications bornées à une seule branche. Vous pouvez par exemple décider que vos travaux en cours ne concernent que des corrections et améliorations minimes et en même temps créer une branche spécifique à l'ajout d'une fonctionnalité. Lorsque vous serez satisfait de votre ajout et que tout fonctionne, vous pourrez fusionner (*merge*) les changements et faire avancer d'un coup la branche principale.

C'est là une utilisation de Git plus avancée que ce que ne couvre cet article d'introduction. Mais sachez qu'un dépôt a toujours au minimum une branche, qu'on appelle traditionnellement **master**, car c'est la branche maîtresse ou principale.

À présent, notre projet est pris en charge dans un dépôt Git local et sur GitHub et nous pouvons poursuivre son développement. Nous allons donc changer le croquis en remplaçant l'utilisation du numéro de broche par une macro :

```
#define LED  13

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, HIGH);
  delay(1000);
  digitalWrite(LED, LOW);
  delay(1000);
}
```

Le croquis est enregistré, cela fonctionne et la led clignote. Mais qu'en dit Git ?

```
% git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler les modifications
dans la copie de travail)
        modifié:         clignote.ino
aucune modification n'a été ajoutée à la validation (utilisez "git add" ou
"git commit -a")
```

Le fichier **clignote.ino** a changé et Git l'a remarqué. Il nous précise d'ailleurs que ces changements n'ont pas encore été pris en compte. Mais quels sont ces changements exactement ?

```
% git diff
diff --git a/clignote.ino b/clignote.ino
index 0fac73d..8aa21bc 100644
--- a/clignote.ino
+++ b/clignote.ino
@@ -1,10 +1,12 @@
+#define LED  13
+
 void setup() {
-  pinMode(13, OUTPUT);
+  pinMode(LED, OUTPUT);
 }

 void loop() {
-  digitalWrite(13, HIGH);
+  digitalWrite(LED, HIGH);
   delay(1000);
-  digitalWrite(13, LOW);
+  digitalWrite(LED, LOW);
   delay(1000);
 }
```

Les changements d'un commit à un autre peuvent directement être affichés dans l'interface web GitHub de façon explicite. On voit ici apparaître clairement le remplacement du numéro de broche dans notre croquis Arduino par une macro.

`git diff` nous affiche automatiquement les différences entre le ou les fichiers actuels et ceux du précédent commit. Les lignes précédées d'un « - » sont celles supprimées et celles avec un « + », celles ajoutées. Nous avons une vue résumée de ce qui a changé... un diff. Si cela venait à ne pas nous convenir, nous pourrions revenir en arrière avec un simple **`git checkout`** (il faudra cependant fermer le croquis dans l'IDE Arduino et le réouvrir).

Tout ceci paraît en ordre, nous pouvons donc valider les changements, commiter cette nouvelle version et pousser tout cela sur GitHub :

```
% git add clignote.ino

% git commit -m "LED pin as macro"
[master 398e399] LED pin as macro
1 file changed, 5 insertions(+), 3 deletions(-)

% git push -u origin master
```

Notre second commit apparaît naturellement dans le journal :

```
% git log
commit 398e39998a3848a903edfdcc96bd5566188d4ec7
Author: dbodor <dbodor@hackable.fr>
Date:   Wed Sep 14 13:55:16 2016 +0200
    LED pin as macro

commit 04f402e9aebe6fd2ebad69cecdb37c4dd79f522f
Author: dbodor <dbodor@hackable.fr>
Date:   Wed Sep 14 13:51:02 2016 +0200
    first commit
```

Ces changements ainsi que tout l'historique des commits apparaîtront également dans GitHub puisque les deux dépôts à ce stade sont strictement identiques après le push.

Si d'aventure, vous supprimez malencontreusement le répertoire **clignote**, ce n'est pas grave, vous ne perdez rien, la preuve :

```
% cd ..
% rm -rf clignote

% git clone https://github.com/dbodor/clignote.git
Clonage dans 'clignote'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 1), reused 6 (delta 1), pack-reused 0
Unpacking objects: 100% (6/6), done.
Vérification de la connectivité... fait.

% git log
commit 398e39998a3848a903edfdcc96bd5566188d4ec7
Author: dbodor <dbodor@hackable.fr>
Date:   Wed Sep 14 13:55:16 2016 +0200
    LED pin as macro

commit 04f402e9aebe6fd2ebad69cecdb37c4dd79f522f
Author: dbodor <dbodor@hackable.fr>
Date:   Wed Sep 14 13:51:02 2016 +0200
    first commit
```

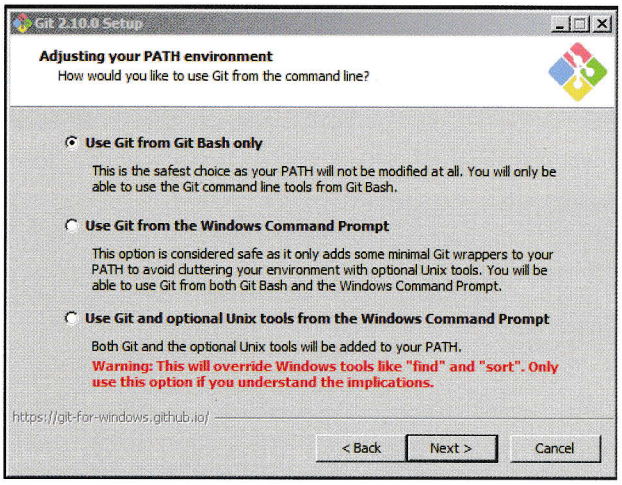
Tout est là, les commits, les messages, les modifications, les dates... tout ! D'une simple commande, `git clone`, vous pouvez créer un clone du dépôt GitHub et poursuivre vos travaux, que ce soit sur votre machine ou une autre, peu importe, tous les clones se valent. Si vous poursuivez vos modifications sur un autre dépôt, puis poussez ces changements sous forme de commit sur GitHub, puis revenez sur l'ordinateur de départ, un simple `git pull` mettra le dépôt local à jour.

De la même façon, vous pouvez parfaitement continuer vos travaux sans Internet, dans le train par exemple ou sur un banc public dans un parc. Faites évoluer votre projet, procédez à de nombreux commits pour bien organiser tout cela puis, de retour chez vous, poussez simplement tous ces changements sur GitHub avec un `git push origin master` et le tour est joué !

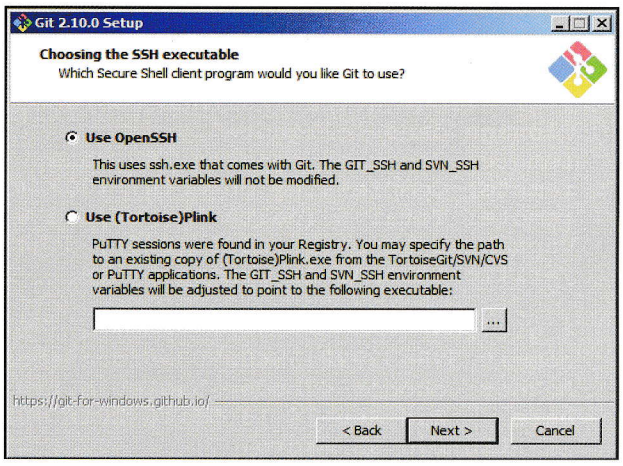
5. ON CONTINUE SOUS WINDOWS

L'utilisation de Git ne se limite pas au système Linux qu'il soit installé sur PC ou sur Raspberry Pi. Il existe en effet plusieurs versions de Git pour Windows. Celle que nous allons utiliser ici est téléchargeable sur **https://git-for-windows.github.io**. Celle-ci présente l'avantage de pouvoir être utilisée aussi bien en ligne de commandes qu'avec une interface graphique. De plus, pour une utilisation plus poussée, elle intègre par défaut les outils nécessaires, comme SSH permettant de s'authentifier sur GitHub en utilisant un mécanisme de clé et non via l'utilisation d'un nom d'utilisateur et d'un mot de passe à saisir à chaque push.

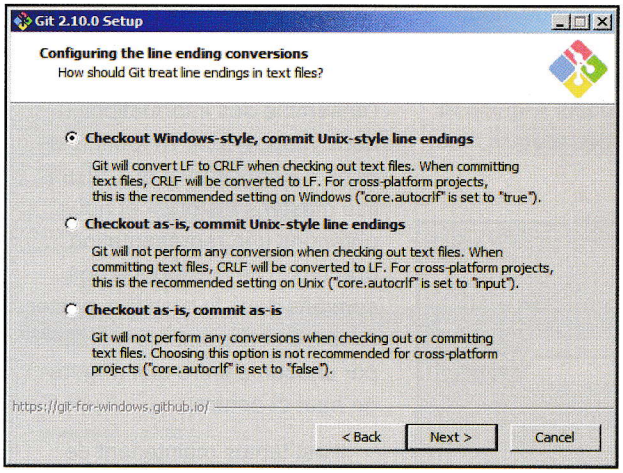
L'installation de Git pour Windows peut être déroutante puisqu'un certain nombre de questions (en anglais) ne sont pas forcément claires ou intelligibles pour une personne n'ayant jamais utilisé un autre système. Git pour Windows s'accompagne d'un interpréteur de commandes, différent de `cmd.exe` ou du PowerShell, et plus précisément Bash, celui-là même présent dans les systèmes Linux. Les questions que pose l'installateur concernent ces éléments et la façon d'intégrer Git dans Windows :



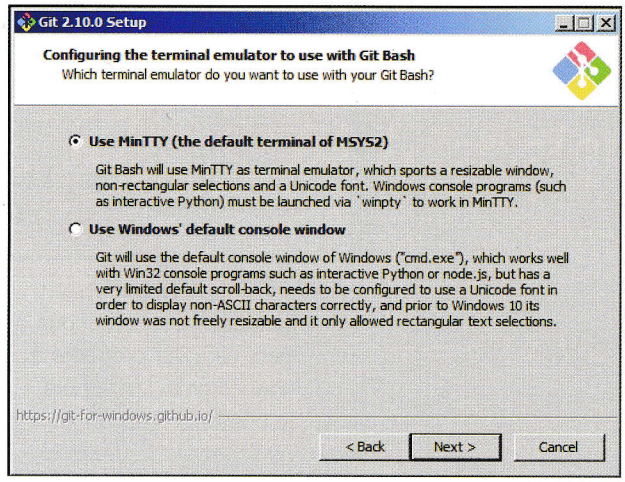
Git installe également Bash, mais si vous souhaitez utiliser Git avec la ligne de commandes classique de Windows, l'installeur doit modifier le système et en particulier le chemin de recherche des programmes. Personnellement, je n'aime pas que des applications changent les paramètres de Windows et je choisis la première option : Git ne sera utilisable en ligne de commandes qu'à travers Bash, l'interpréteur installé avec Git.



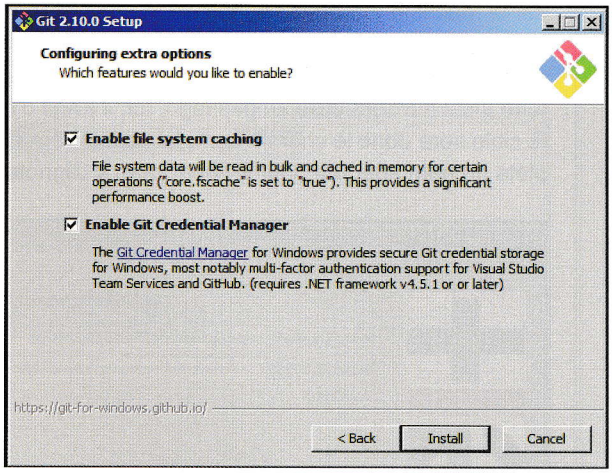
Une façon d'assurer un maximum de sécurité consiste à utiliser SSH en lieu et place de la saisie de login et mot de passe. Git installe un client SSH, mais vous pouvez également choisir d'utiliser une autre application. Le SSH intégré est généralement une bonne solution si vous n'utilisez que Git comme gestionnaire de versions.



Windows ne gère pas les caractères de fin de ligne de la même façon que les systèmes de type Unix comme Linux. Avec ces systèmes, les fins de ligne sont marquées avec le caractère LF (saut de ligne) alors que sous Windows ce sont deux caractères qui sont utilisés : CRLF (retour chariot et saut de ligne). Le problème est double, car vous n'avez pas envie de récupérer des lignes avec un format différent et inversement si vous mettez des CRLF partout en contribuant à un dépôt vous risquez de vous attirer les foudres du programmeur en face. Heureusement, Git peut faire la conversion à votre place automatiquement de façon à avoir des CRLF dans votre Windows, mais des LF en commitant.



L'interpréteur n'est qu'un programme prenant en compte vos commandes, ce n'est pas une application avec une fenêtre. Pour cela, on utilise un émulateur de terminal. Si vous comptez utiliser la ligne de commandes, ici, vous avez le choix entre celui intégré avec Git, MinTTY et celui proposé en standard par Windows (la console qui va avec cmd.exe). Celui accompagnant Git étant plus riche en fonctionnalités, c'est généralement le choix le plus intéressant.



Enfin, l'installateur vous demande s'il doit activer des options supplémentaires dont le cache permettant d'accélérer grandement les opérations et un gestionnaire d'accréditation permettant l'utilisation de l'authentification multi-facteur dans Visual Studio et avec GitHub. Les deux options sont activées par défaut et cela nous convient parfaitement.

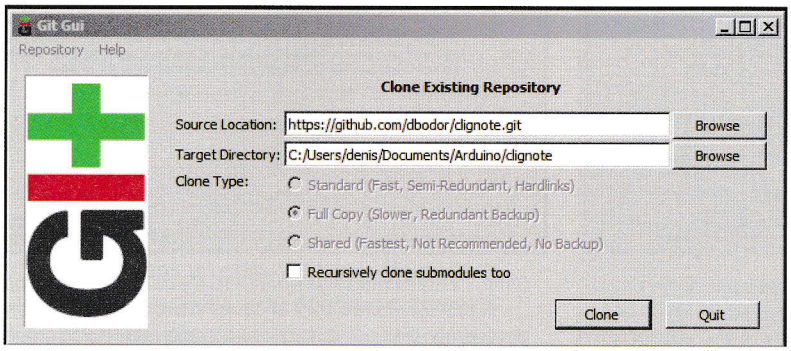
Et voilà ! Votre Git pour Windows est installé. Dans le menu ***Démarrer***, vous retrouverez un dossier Git avec :

- Git Bash : pour utiliser Git en ligne de commandes avec un interpréteur Bash comme sous Linux ;
- Git CMD : pour faire de même, mais avec la ligne de commandes classique de Windows ;
- Git GUI : l'interface graphique permettant de gérer Git de façon un peu plus... clic-clic.

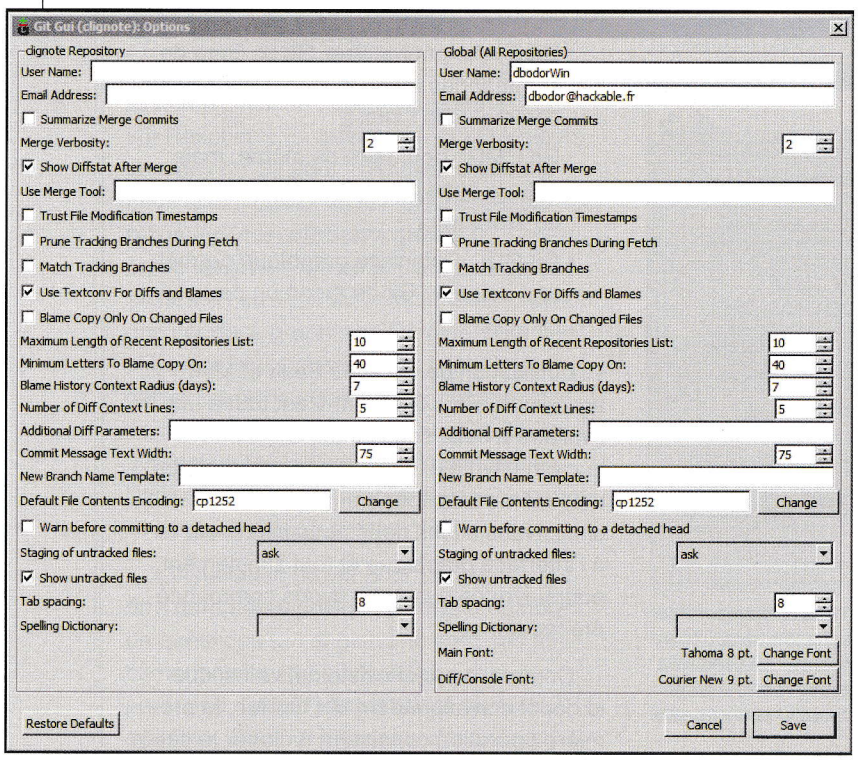
L'utilisation de Git avec Bash et MinTTY, accessible via « Git Bash » est tout à fait similaire à celle sous Linux. On retrouve la même syntaxe, les mêmes commandes et le même fonctionnement général du shell. Nous allons donc nous intéresser ici à l'interface graphique et tout simplement poursuivre ce que nous avons commencé précédemment.

Comme le projet existe déjà ainsi que le dépôt correspondant sur GitHub, la première opération consistera à cloner le dépôt

sur la machine Windows. À l'invocation de l'interface graphique, nous trouvons directement une entrée « *Clone Existing Repository* » nous permettant de spécifier la source, sur GitHub, et la destination, un répertoire local qui sera créé au passage. Ici, il s'agit d'un croquis Arduino et la cible sera donc le chemin complet vers le répertoire des croquis, complété du nom adéquat pour le répertoire, « clignote » :



Avant tout changement dans le dépôt qui vient d'être cloné et de la même façon que nous avons dû le faire sous Linux, nous précisons à Git un nom d'utilisateur et une adresse mail qui seront utilisés globalement. Les options sont configurables via le menu « Edit » et « Options », dans la fenêtre qui s'ouvre juste après le clonage :

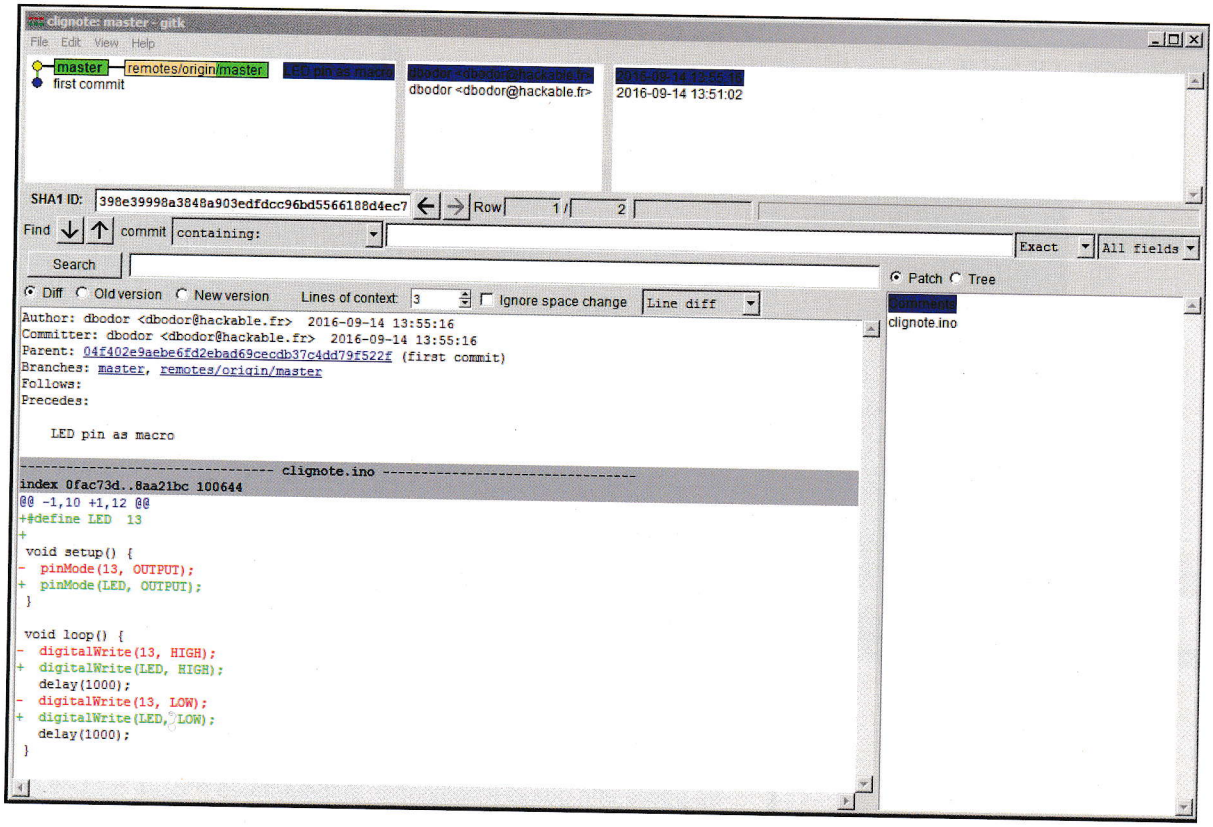


Nous possédons maintenant une copie du dépôt et comme promis, on retrouve l'ensemble des informations qui s'y rapportent. Un petit tour dans « Repository » et « Visualize master's history » et nous obtenons une représentation graphique de l'historique des commits dans la branche master avec les changements effectués par chacun d'entre eux (Figure en haut ci-contre).

Il est temps maintenant de procéder à quelques modifications du croquis depuis l'environnement Arduino. Nous décidons de définir une macro correspondant à la durée utilisée par `delay()` et modifions le code dans ce sens :

```
#define LED  13
#define DUREE 1000

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, HIGH);
  delay(DUREE);
  digitalWrite(LED, LOW);
  delay(DUREE);
}
```

Après enregistrement et vérification, nous pouvons nous tourner vers Git pour procéder à un nouveau commit. Un simple clic sur le bouton « Rescan » procède à l'analyse du dépôt et à l'affichage des changements. Dans la zone supérieure gauche figurent les fichiers ayant changés, mais non choisis pour le commit (*unstaged change*), un clic sur le ou les fichiers aura pour effet de les prendre en



Enseignants, Lycées, Écoles, Universités...

HACKABLE MAGAZINE

Besoin de ressources pédagogiques ?

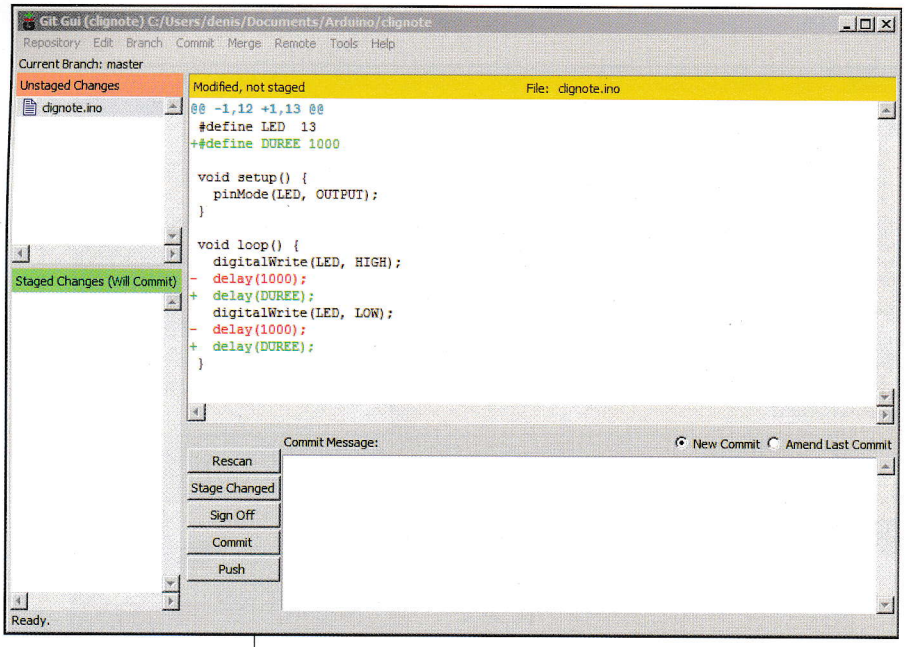
...Permettre à mes élèves de consulter la base documentaire ?

C'est possible ! Rendez-vous sur :

http://proboutique.ed-diamond.com

pour consulter les offres !

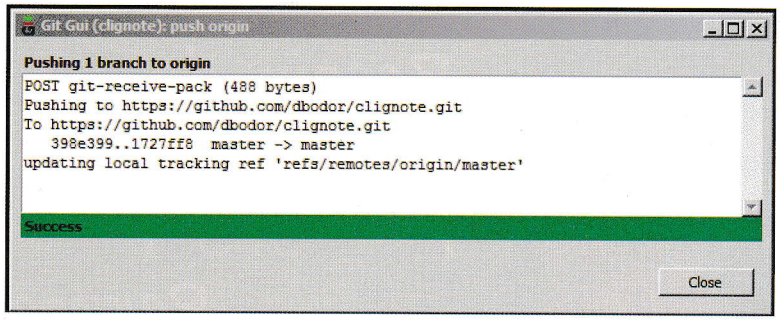
N'hésitez pas à nous contacter pour un devis personnalisé par e-mail : abopro@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20



considération pour votre commit. C'est l'équivalent du **`git add`** que nous avons vu précédemment. À partir de là, il vous suffira d'ajouter le message de commit dans la zone inférieure droite et de cliquer sur le bouton « Commit » (Figure ci-contre).

Nous venons de commiter les changements, mais nous devons encore pousser tout cela sur le dépôt GitHub. Un simple clic sur le bouton « Push » provoquera cette action. Il n'est pas utile de spécifier le dépôt distant puisque nous l'avons paramétré précédemment avec **`git remote`** et que ceci est maintenant présent dans la configuration du dépôt lui-même. Vous serez alors invité, comme en ligne de commandes, à saisir votre nom d'utilisateur GitHub et son mot de passe afin que la modification soit autorisée (Figure ci-contre).

Après l'échange entre votre ordinateur et GitHub et l'application des changements, un message de confirmation vous sera affiché :



Bravo ! Vous venez de procéder à votre premier commit et push depuis Windows. De retour sur votre machine Linux, dans le dépôt correspondant, un simple **`git pull`** répercutera les changements depuis le dépôt GitHub sur le dépôt local :

```
% git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
Depuis https://github.com/dbodor/clignote
   398e399..1727ff8  master     -> origin/master
Mise à jour 398e399..1727ff8
Fast-forward
clignote.ino | 5 +++--
```

```
 1 file changed, 3 insertions(+), 2 deletions(-)

% git log
commit 1727ff8e5166657c161f4a3d53f6e722b4690aa0
Author: dbodorWin <dbodor@hackable.fr>
Date:   Wed Sep 14 16:10:39 2016 +0200
    blink delay as macro

commit 398e39998a3848a903edfdcc96bd5566188d4ec7
Author: dbodor <dbodor@hackable.fr>
Date:   Wed Sep 14 13:55:16 2016 +0200
    LED pin as macro

commit 04f402e9aebe6fd2ebad69cecdb37c4dd79f522f
Author: dbodor <dbodor@hackable.fr>
Date:   Wed Sep 14 13:51:02 2016 +0200
    first commit
```

Comme attendu, on retrouve, dans le journal, le dernier commit fait sous Windows. Chose qui apparaîtra, bien entendu, également via l'interface web de GitHub.

POUR CONCLURE

Cette simple introduction à Git et à un service comme GitHub ne fait apparaître que l'aspect le plus simple de l'utilisation de Git, mais offre déjà une vision claire des services que peut rendre un système de gestion de versions pour vos projets. Fini les renommages de fichiers, les oublis, les pertes de code... En reposant sur Git, vous confiez toutes ces problématiques à un outil et pouvez donc vous concentrer sur ce qui est le plus important : votre projet et votre code.

Prendre en main et vous habituer à utiliser Git vous permettra de vous simplifier la vie, mais aussi de partager vos connaissances, vos travaux et de participer aux projets d'autres programmeurs en leur proposant des modifications, des corrections et des améliorations. Ce sera également l'occasion d'apprendre en recevant des contributions ou parfois, en vous faisant remettre en place, car votre modification casse tout ou ne correspond pas à la façon de faire du projet.

Ce genre de chose n'est ni une erreur ni un problème. C'est certes parfois humiliant ou gênant, mais c'est surtout une occasion d'apprendre. N'ayez pas peur de montrer votre code, publiez tôt et publiez souvent. À terme, cela ne peut vous être que bénéfique... DB

TECHSHOP RECRUTE DES ACCOMPAGNATEURS(TRICES) PROJET ("DREAM CONSULTANT")* À LILLE ET PARIS

Prise de Mission : début 2017

*consultant de rêve

> TechShop est un atelier collaboratif de fabrication de plus de 2000 m² dédié à l'innovation et à la créativité et destiné à la communauté des "makers" (bricoleurs, entrepreneurs, étudiants, retraités, créateurs...).

Plus d'info sur : www.techshoplm.fr

LE PROFIL :

Dynamique, ingénieux, curieux, le Dream Consultant est au centre de l'expérience TechShop. Avec un sens exceptionnel du service, de l'écoute et de la relation interpersonnelle, il accompagne les membres dans la réalisation de leurs projets et il s'occupe de la maintenance des lieux.

Pour plus d'infos, contactez-nous à l'adresse suivante : **job@techshoplm.fr**

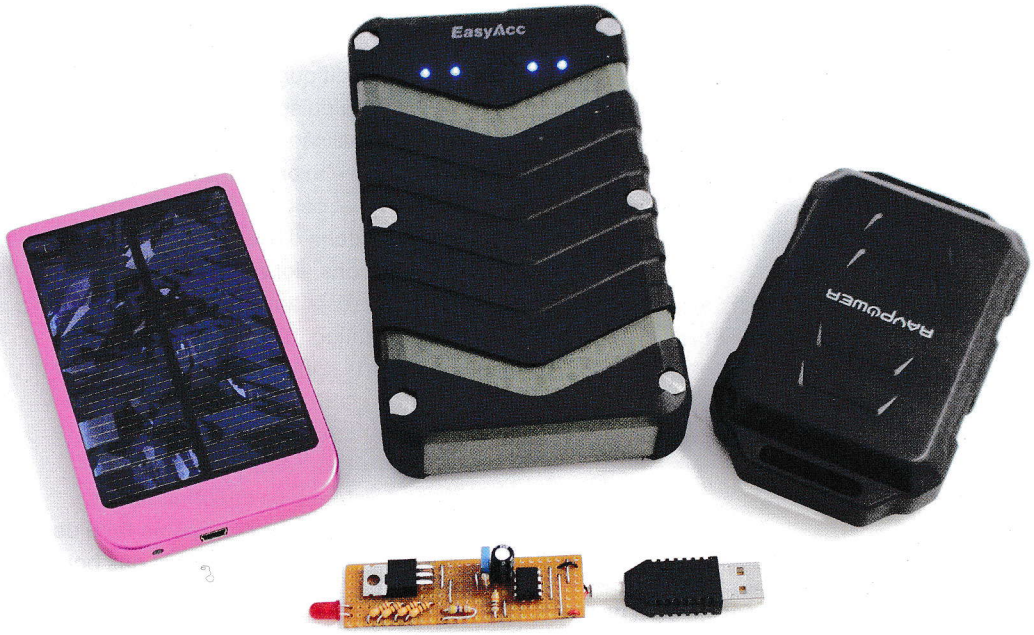
Suivez-nous sur **Facebook** et **Twitter**

TechShop ATELIERS LEROY MERLIN

VOS PROJETS ARDUINO ET RASPBERRY PI SUR UNE BATTERIE EXTERNE POUR SMARTPHONE. PAS SI SIMPLE...

Denis Bodor

En voilà une drôle d'idée d'article ! Après tout, qu'est-ce qui pourrait bien mériter des explications si ce n'est de simplement brancher son montage sur le ou les ports USB de la batterie externe ? La réponse est simple : avec une batterie de bonne qualité, cela ne marchera qu'une paire de minutes... puis le courant sera coupé. Si vous avez déjà essayé et eu quelques déboires de ce genre, voici le pourquoi du comment et surtout la solution au problème !



Une petite collection de batteries externes pour recharger les smartphones avec à gauche une version solaire chinoise de soi-disant 2600mAh (plutôt 1500mAh dans les faits), au centre une grosse bête EasyAcc PB20000TP de 20000mAh et à droite une RavPower RP-PB044 de 10050mAh. Comme on peut s'y attendre, les deux batteries de droite ont une fonction d'arrêt automatique et celle de gauche non.

Les batteries externes pour smartphones et tablettes, tantôt appelées *power bank*, sont très à la mode. Leur principe d'utilisation est relativement simple : vous chargez la batterie chez vous puis l'emportez. Lorsque votre smartphone donne des signes de faiblesse du fait d'une utilisation intensive (généralement du type 4G + écran + GPS = Pokemon), il vous suffit de le brancher sur la batterie via un câble USB et celle-ci le recharge comme le ferait un adaptateur secteur classique.

1. LE PROBLÈME

Instinctivement, et du fait des prix qui ne cessent de chuter pour ce type d'accessoires, on aurait tendance à penser qu'il s'agit là d'une solution parfaite pour alimenter un montage à base d'Arduino ou de Raspberry Pi. Après tout, ces batteries peuvent fournir un courant non négligeable avec une tension de 5V régulée, et ce via un ou plusieurs connecteurs USB type A. Pas besoin de contrôler la décharge ni la charge des accus, et les capacités proposées, de l'ordre de quelques 10000 mAh satisferont la plupart des besoins... Mais, et c'est un gros « mais », la plupart du temps, on rencontre un problème majeur : ces batteries sont conçues pour fournir un courant important à un smartphone et se mettent en arrêt dès lors que le smartphone en question a fini de charger. Cette détection se fait, dans la batterie externe, en surveillant l'intensité du courant qui circule sur le ou les ports USB.

En d'autres termes, si vous connectez un montage à base d'Arduino et de quelques capteurs, il y a de fortes chances qu'après quelques minutes ou moins, le circuit d'alimentation de la batterie décide que rien n'est branché et qu'il peut s'éteindre. Bien entendu, tous les modèles de batteries externes ne le font pas, mais ce sont malheureusement celles qui sont les mieux conçues et de meilleure qualité qui intègrent ce type de fonctionnalités. Il y a des fabricants qui n'hésitent pas à créer des produits avec de beaux boîtiers, mais un contenu plus que douteux, sans aucune sécurité ou intelligence embarquée, et ceux-là on préfèrera bien entendu les éviter. C'est toujours plus appréciable d'avoir effectivement à disposition la capacité marquée sur le produit et de limiter les risques d'incendie (oui, j'ai eu entre les mains un tel produit, intégrant des cellules usagées recyclées d'un portable et... du sable).

Une Raspberry Pi « nue » consommera entre 180 et 500 mA en fonction du modèle. Pour une carte Arduino, ce sera une vingtaine de milliampères, mais augmenter délibérément la consommation n'est pas la solution. En effet, le principe même de l'utilisation d'une batterie est d'assurer au montage une autonomie la plus importante possible. Ajouter une grosse résistance nous assurant une consommation constante de 300 mA, en plus du montage, est totalement contreproductif. Pire encore, en fonctionnement sur batterie, nous tenterons forcément de réduire au maximum la consommation, ce qui dans certains cas pourrait nous faire passer sous la barre du milliampère en veille, un niveau clairement en deçà de ce qui peut être considéré comme une consommation valable du point de vue de la batterie externe.

2. LA SOLUTION

Les batteries externes pour smartphones ne sont pas juste des accumulateurs joyeusement soudés entre eux avec, en plus, un simple circuit de protection (parfois oui, mais si vous aimez votre smartphone, passez votre tour). Il s'agit en réalité de systèmes intelligents embarquant un microcontrôleur exécutant un programme de gestion. Celui-ci a pour tâche de contrôler la charge et décharge des accumulateurs, le bon fonctionnement de l'ensemble, et bien entendu l'optimisation de la consommation.

Un test facile consiste à allumer le boîtier et simplement d'attendre que celui-ci s'arrête. Bien entendu, en fonction du modèle, il n'est pas impossible que les voyants notifiant de l'état de charge s'éteignent bien avant l'arrêt de la fourniture effective du courant. Une simple led accompagnée d'une résistance de 470 ohms, une mesure au multimètre ou tout simplement la connexion d'une carte Arduino suffiront pour en avoir le cœur net. Généralement, le délai d'extinction est de l'ordre d'une à deux minutes. Sachant cela, vous avez déjà la première information importante : l'intervalle après lequel le boîtier va décider de jeter l'éponge.

L'autre élément à déterminer est le courant minimum nécessaire qui doit circuler pour que le circuit dans la batterie considère qu'il est en train de charger un smartphone ou une tablette. Il s'agit du seuil en dessous duquel la batterie pense que rien n'est connecté. Pour déterminer cette valeur, nous sommes obligés de tâtonner. Plus proche nous serons de la valeur limite moins nous consommerons effectivement de courant avec notre montage. Pour procéder aux essais, il existe plusieurs méthodes, mais toutes reposent sur le fait de consommer effectivement un courant

Ainsi, nous pouvons utiliser des résistances à placer entre la tension d'alimentation USB et la masse. La bonne vieille loi d'Ohm nous dit que U = R x I. « U » est connu puisque c'est la tension normalisée du bus USB, 5V. Nous n'avons donc qu'à choisir « I » et calculer : U/I = R. Si nous décidons d'utiliser 200 mA, nous devrons alors mettre en œuvre une résistance de 25 ohms. Mais attention ! Les résistances standards sont prévues pour dissiper 1/4 de watt. Avec un courant de 200 mA en 5V, nous arrivons à P=UxI, 5*0,2=1 ampère. La résistance va chauffer, fumer, sentir et peut-être même faire de la lumière... ce qui n'est pas bon signe.

Il faudra donc utiliser plusieurs résistances en parallèle pour que chacune d'elles dissipe une puissance acceptable. Le calcul de la résistance totale avec un montage en parallèle répond à la formule 1/(1/R1+1/R2+1/R3...). Avec 6 résistances de 150 ohms, nous arrivons donc à 1/(1/150×6)=25 ohms. Chacune des résistances devra dissiper une puissance dépendant du courant qui y circule, soit 5/150=0,033 ou 33 mA, ce qui nous donne 5 x 0,033 = 165 mW. Nous sommes sous le 1/4 de watt, tout va bien.

Une autre solution est d'utiliser des leds. Une led rouge standard utilise un courant entre 10 et 20 mA, et fait apparaître à ses bornes une tension de 1,8V. Ainsi, nous pouvons utiliser une simple résistance de quelques 330 ohms, et multiplier le circuit en parallèle jusqu'à consommer le courant visé. Il sera sans doute plus judicieux d'opter pour des leds de forte puissance (bleues, blanches, infrarouges) afin d'en réduire le

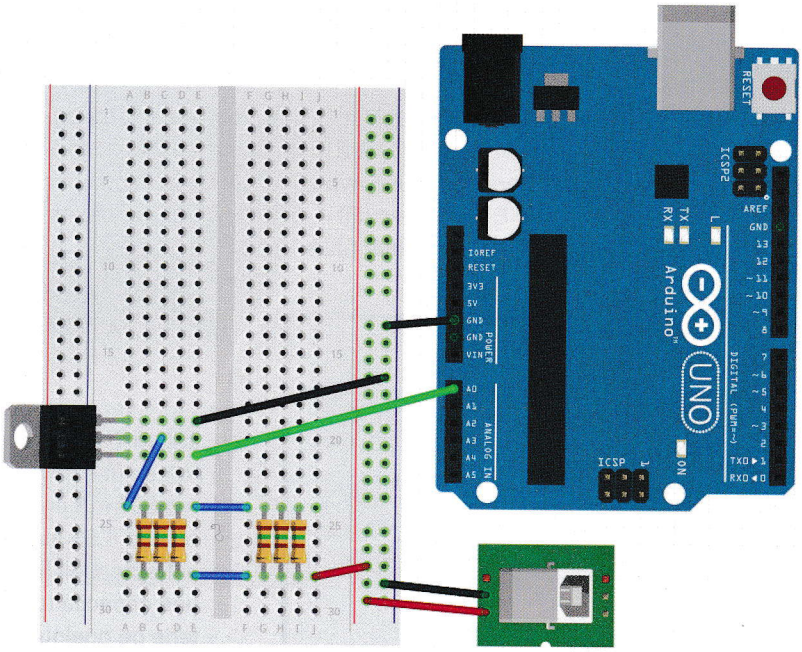


Figure 1 : Ce montage simple nous permet de faire consommer quelques 200mA à la batterie externe pour une durée déterminée et avec un intervalle choisi.

nombre. Cette option cependant reste relativement désavantageuse, car plus coûteuse que de simples résistances et relativement voyante selon le montage qui sera utilisé avec la batterie au terme des essais. L'objectif n'est pas d'avoir une lampe de poche, mais juste une batterie externe qui reste allumée.

Le test en lui-même est relativement simple : brancher le montage destiné à consommer du courant et vérifier si la batterie décide toujours de s'éteindre au bout du délai constaté précédemment. Si c'est le cas, il faut augmenter le courant, sinon, le réduire légèrement.

Une fois la valeur optimale trouvée, il ne nous manque plus qu'une information pour compléter les recherches : le temps durant lequel cette consommation doit être présente pour être validée par le système dans la batterie externe. Une pulsation d'une paire de millisecondes pourrait ne pas être suffisante pour que la batterie « voit » la consommation et la considère comme valide pour réinitialiser le délai avant extinction.

Là, nous arrivons dans un domaine où il devient difficile de tester manuellement différentes valeurs, car il n'est pas envisageable d'arriver, par simple contact ou avec un bouton poussoir, à connecter le circuit 10, 50 ou 200 ms. Nous devons faire intervenir un montage « maison ».

3. UN MONTAGE POUR COLLECTER LES INFORMATIONS

Notre objectif sera de rapidement assembler quelques composants pour ensuite, à l'aide d'un croquis Arduino tout aussi simple, laisser ou non passer le courant dans les résistances et ainsi trouver la temporisation nécessaire pour garder la batterie externe active en continu.

Ce montage va donc faire intervenir notre assemblage de résistances et/ou de leds, mais le rendre contrôlable par l'une des sorties de la carte Arduino. La technique est exactement la même que si vous deviez contrôler n'importe quel circuit qui demande davantage de courant que ne peut en fournir une sortie de la carte. Il y a plus d'une façon de faire, mais celle que je préfère et trouve la plus simple consiste à mettre en œuvre un MOSFET-N et plus précisément un IRL520 parfaitement adapté aux signaux 0/5V de l'Arduino.

Nous allons donc connecter l'ensemble des résistances (la charge) entre les 5V de la batterie et le drain du MOSFET, puis connecter la source à la masse (qui sera commune avec l'Arduino). Pour activer/désactiver le MOSFET, il nous suffit de connecter sa dernière broche, la grille, à la sortie que nous allons contrôler. Le croquis sera très basique :

```
void setup() {
  pinMode(A0, OUTPUT);
}

void loop() {
  // activation
  digitalWrite(13, HIGH);
  // 200ms
  delay(400);

  // désactivation
  digitalWrite(13, LOW);
  delay(25000);
}
```

Il ne reste plus qu'à partir dans une boucle modification/compilation/chargement/essai en jouant sur le premier délai jusqu'à descendre au minimum auquel la batterie cesse de voir la consommation comme valide. On pourra, par la même occasion, tâtonner de la même façon avec intervalle entre les pulsations (second délai) mais, cette fois en augmentant la temporisation.

Plus la durée d'activation du MOSFET sera courte et plus le délai entre ses activations sera long, plus nous économiserons de courant et rallongerons l'autonomie du futur projet fonctionnant sur batterie.

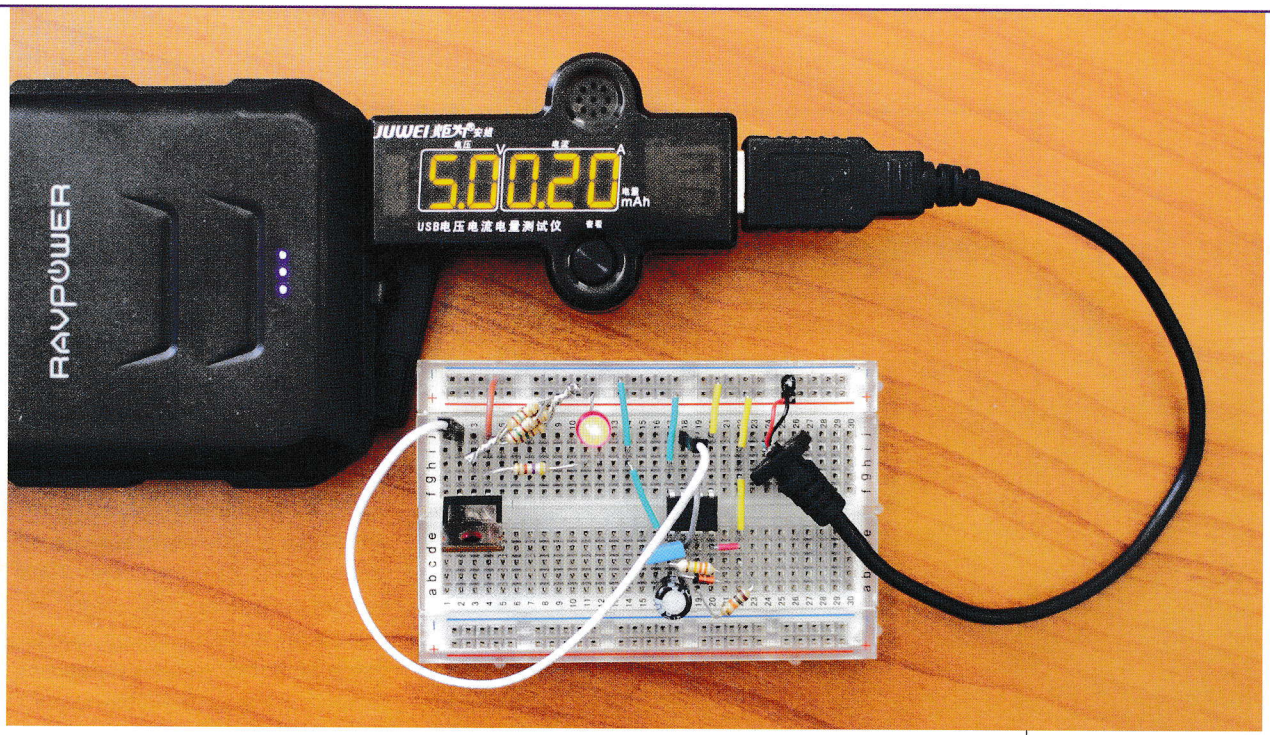
Ces mesures dépendront totalement de la batterie externe que vous testerez. Certaines demandent une consommation courte sur des intervalles longs, d'autres l'inverse, et d'autres encore semblent choisir un intervalle en fonction de la durée de la période où le courant passe, avec, semble-t-il, quelques paliers. L'objectif n'est cependant pas d'analyser le fonctionnement du circuit dans la batterie, mais simplement de trouver des temporisations qui donnent un équilibre satisfaisant entre consommation totale et activation permanente.

Dans mon cas, avec les trois batteries testées, il semble qu'une impulsion de quelques 800 ms toutes les 24 secondes satisfait les besoins des différents circuits de gestion d'alimentation. Partant de cette constatation, il sera facile de déterminer approximativement combien nous « coûtera » un montage de maintien en éveil de la batterie :

- une impulsion de 0,8s toutes les 24s nous donne 2,41 impulsions par minute et donc 145 par heure ;
- les 145 impulsions en question nous donnent une durée totale de 118,4 secondes sur une heure ;
- 118,4 secondes à consommer 200 mA reviennent à 23600 mA en une seconde, ou environ 393 mA en une minute ou encore 6,55 mA en une heure, autrement dit 6,55 mAh sur une capacité totale de la batterie de quelques 10000 mAh.

L'autonomie de la batterie ne s'en trouvera donc pas drastiquement réduite et nous pouvons atteindre notre objectif. Deux solutions s'offrent maintenant à nous : prévoir dans chaque montage sur batterie une telle temporisation et un tel circuit, ou créer un circuit dédié qui se branchera en parallèle sur la batterie.

La première solution est tentante, car elle ne nous coûterait qu'un MOSFET et quelques résistances. Mais ceci signifiera de prévoir obligatoirement dans chaque montage ce type de fonctionnalité. Je n'aime pas cette idée et préfère grandement garder les choses bien séparées. Dans le cas d'un montage ou d'un circuit autonome devant accomplir une tâche périodique, il peut être rapidement pénible de devoir prendre en compte la problématique du maintien en éveil de la batterie. Mes montages autonomes se limiteront à leur objectif premier et un autre circuit se chargera de la batterie.



Voici la version « platine à essais » de notre montage de maintien en éveil de la batterie avec, en prime, un petit adaptateur USB fort pratique permettant d'afficher la tension et le courant en sortie (moins de 3€ sur eBay/Banggood). Ici, le montage est actif et laisse circuler quelques 200mA dans les résistances. Exactement comme calculé/simulé et du premier coup... fait rare et d'autant plus plaisant.

4. UN CIRCUIT POUR PASSER DE LA THÉORIE À LA PRATIQUE

La problématique des batteries externes qui s'éteignent n'est pas nouvelle et en fouillant sur le Web à la recherche d'informations je suis tombé sur un billet de *Paul Stoffregen*, le créateur des cartes Teensy, sur **dorkbotpdx.org**. De façon fort amusante, son approche était identique à la mienne (mais avec 3 ans d'avance) en testant la batterie avec une de ses Teensy et non une carte Arduino.

Sa solution, également indépendante du montage à utiliser sur batterie, repose sur la mise en œuvre de deux transistors sous la forme d'un oscillateur très simple. En fonction de la charge et décharge d'un condensateur en montage RC, la consommation de quelques millisecondes intervient toutes les 1,4 secondes. Plusieurs points cependant ne me conviennent pas dans ce circuit : la mise au point d'un montage à transistors peut vite devenir compliquée (voire pénible), la circulation du courant ne se fait pas sous la forme d'états discrets, mais fluctue, ce qui rend difficile le calcul du courant effectivement consommé et enfin, chaque modèle de transistor ayant ses propres caractéristiques, il faudra refaire les calculs et les tests pour chaque combinaison (et là je parle de vous, et non de moi).

Pour autant, et même si cela est relativement tentant, il n'est pas question non plus de mettre en œuvre un microcontrôleur pour une tâche aussi simple. Il existe un circuit intégré parfaitement adapté pour créer un oscillateur : le NE555, et celui-ci ne vous coûtera qu'une vingtaine de centimes. Ajoutez au 555 deux résistances, un condensateur, et dans notre cas une diode et vous voici en mesure de faire exactement ce dont nous avons besoin : envoyer une impulsion d'une largeur définie à intervalles réguliers que nous configurons comme bon nous semble.

Le 555 est un circuit qui a plus de 45 ans et qui est toujours largement utilisé, c'est littéralement un circuit à tout faire. Notre utilisation reposera sur sa configuration astable avec une petite astuce. En effet, ce dont nous avons besoin se rapproche fortement d'une PWM ou modulation en largeur d'impulsion. Le problème cependant c'est que le 555 ne permet pas un rapport cyclique de moins de 50% (moins de temps à l'état haut qu'à l'état bas), à moins bien sûr, de trouver

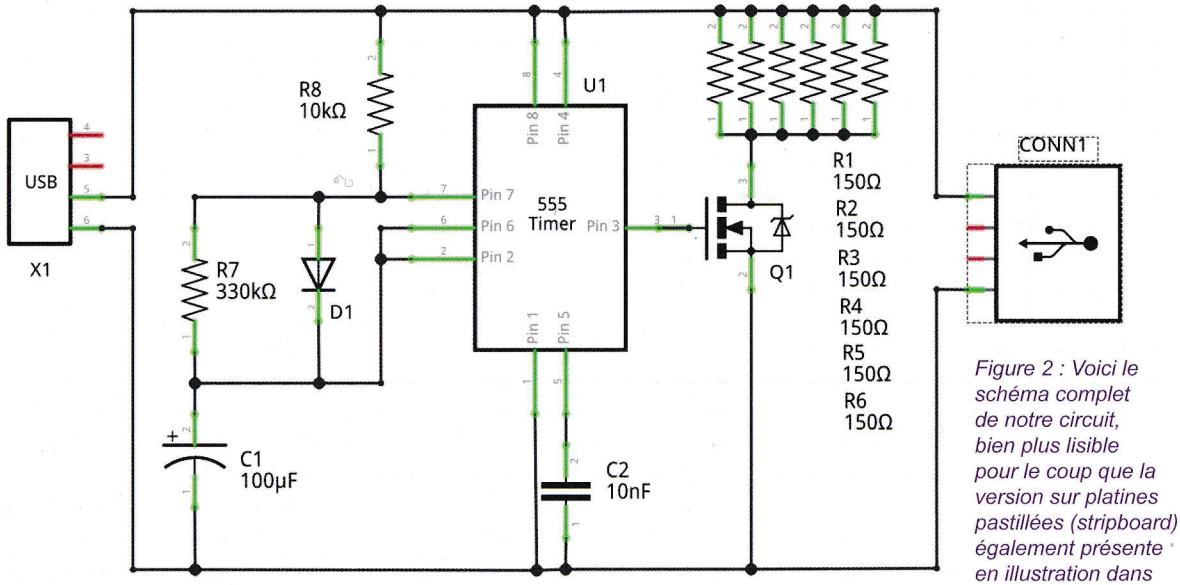


Figure 2 : Voici le schéma complet de notre circuit, bien plus lisible pour le coup que la version sur platines pastillées (stripboard) également présente en illustration dans cet article.

un moyen d'inverser sa sortie. Si j'ai choisi la solution à base de 555, ce n'est pas pour ajouter un transistor en plus et ajouter un circuit logique comme un inverseur ou une NAND serait « déplacé »...

Le fonctionnement du 555 repose sur le chargement/déchargement contrôlé d'un condensateur à l'aide de deux résistances. L'astuce consiste donc à ajouter une diode de façon à passer outre l'une des résistances durant le chargement et ne l'utiliser que pour la décharge du condensateur.

Le schéma en figure 2 montre notre montage final autour du 555. Les résistances R1 à R6 sont celles destinées à régler le courant devant circuler et la connexion à la masse est contrôlée par le MOSFET IRL520 en Q1. Toute la partie de droite n'intervient aucunement dans la configuration de l'oscillateur qui repose entièrement sur R8, R7 et C1. C'est en jouant sur ces valeurs et en particulier celles des résistances qu'on pourra choisir la durée des états haut et bas de la sortie connectée à la grille du MOSFET.

Pour rappel, les broches du 555 sont :

- GND : la masse,
- TRIG : la gâchette qui déclenche la temporisation si la tension est inférieure à 1/3 de Vcc,
- OUT : le signal en sortie,
- RESET/RST : la remise à zéro qui stoppe la temporisation,
- CONT/CV : accès à la tension de référence interne (à 2/3 de Vcc),
- THRES/THRS : seuil qui détermine la fin de la temporisation lorsque la tension est supérieure à 2/3 de Vcc,
- DISCH/DIS : connexion du condensateur à décharger,
- VCC : tension d'alimentation.

R8 est ici l'équivalent de Ra (ou R1) telle qu'on la trouve généralement désignée dans la documentation du 555. R7 est Rb et C1 est C.

Les formules de calcul classiques pour le 555 dans cette configuration astable sont :

- la fréquence : 1,44 / ((Ra + 2 x Rb) x C) ;
- le rapport cyclique : (Ra + Rb) / (Ra + 2 x Rb).

Cependant, comme on a ajouté la diode, ici une 1N4148, en parallèle à Rb, ceci change la donne et nous permet d'utiliser plutôt :

- Durée état haut : 0,67 x Ra x C ;
- Durée état bas : 0,67 x Rb x C.

Dans notre cas, ceci nous donne :

- État haut : 0,67 x 10 kohms x 100 µF = 0,67 x 10000 x 0,0001 = 0,67 secondes = 670 ms ;
- État bas : 0,67 x 330 kohms x 100 µF = 0,67 x 330000 x 0,0001 = 22 secondes.

Comme les résistances standards ont une tolérance de 5%, nous pourrons avoir un état haut entre 636 et 703 millisecondes et un état bas entre 21 et 23 secondes. Bien entendu, il faut également prendre en compte la tolérance du condensateur électrolytique, de l'ordre de 10 à 20%, ce qui élargit encore les variations des plages de temps. Il n'est donc pas impossible qu'avec les mêmes valeurs de composants vous ayez des délais relativement différents. L'important cependant est d'arriver à rester dans ce qu'accepte la batterie externe.

5. RÉALISATION ET MISE EN PLACE

Une fois l'aspect théorique et les valeurs pour votre batterie externe déterminés, vous pourrez passer à la réalisation. L'assemblage du circuit sur plaque pastillée ou même platine à essais est plus facile à faire qu'à représenter. Avec un peu de patience et de logique, et en structurant le circuit avec la tension d'alimentation en haut et la masse en bas, on arrive facilement à obtenir quelque chose de compact et propre (Figure 4, page suivante).

En ce qui concerne la connectique, le plus simple est d'utiliser une rallonge USB (type A mâle d'un côté, femelle de l'autre), pour la couper en deux et placer notre circuit au milieu. On pourra éventuellement ajouter une résistance et une led entre la tension d'alimentation et le drain du MOSFET (en parallèle aux résistances) si l'on souhaite avoir un témoin de fonctionnement. La consommation supplémentaire de 10 ou 20 mA ne changera pas grand-chose.

Une autre option est utilisable avec les batteries externes à double connecteur USB : brancher notre circuit à base de 555 sur un port et le montage autonome sur l'autre. Dans mon cas, la consommation de courant sur un port maintenait également l'alimentation sur le second. Je ne peux cependant garantir ce fonctionnement avec tous les modèles de batterie puisque techniquement rien n'empêche un constructeur de gérer indépendamment les deux ports.

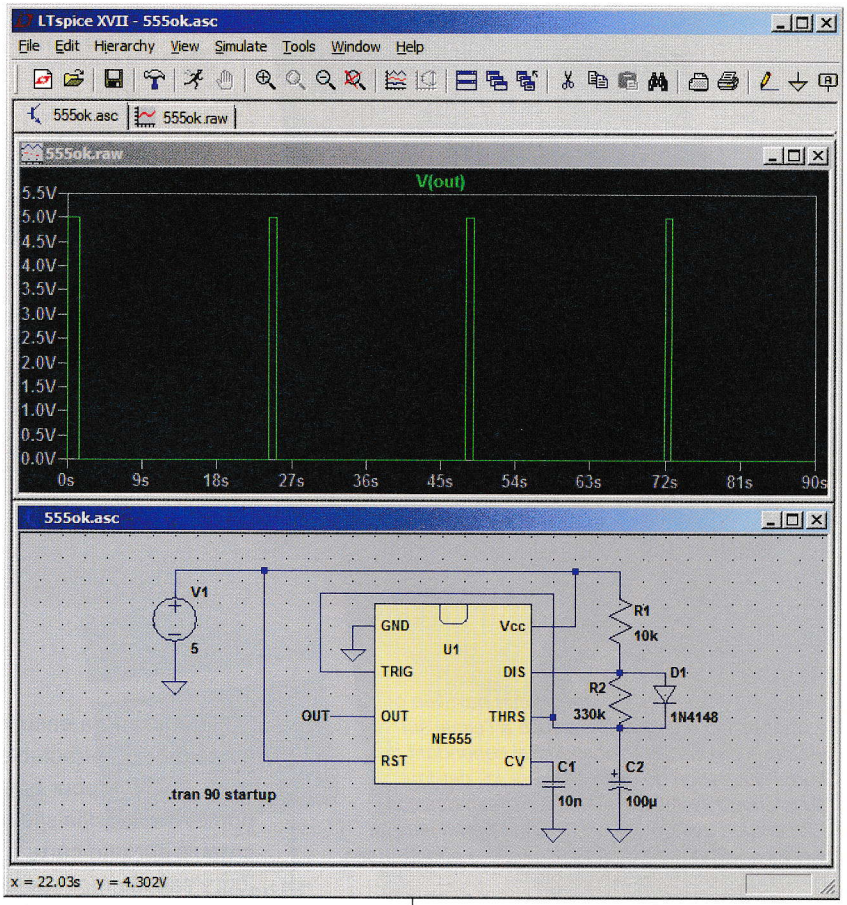


Figure 3 : Il existe un outil gratuit installable sous Windows, permettant de procéder à une simulation de circuits comme celui-ci : LTspice de Linear Technology. Assez spartiate et peu ergonomique, il permet cependant de procéder à des essais rapides de différentes valeurs de résistances et de condensateurs, sans avoir à faire les calculs manuellement ou à procéder à des tests physiques.

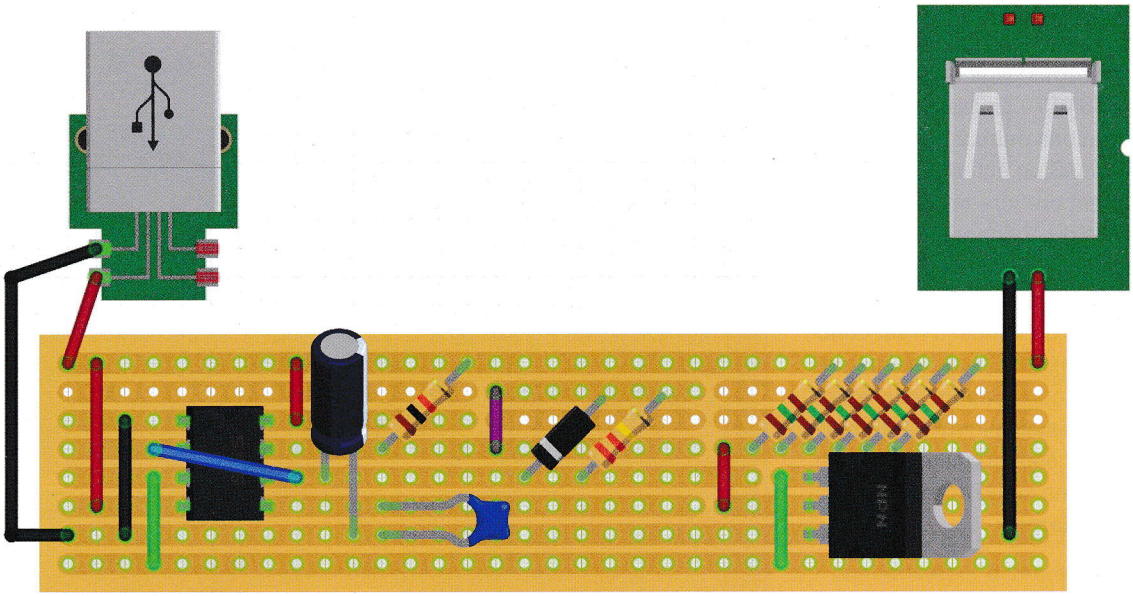
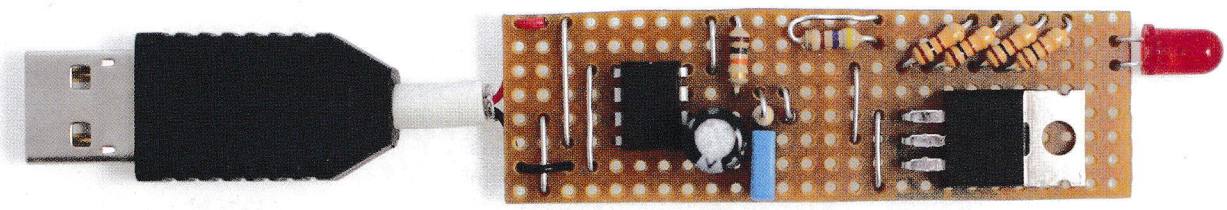


Figure 4 : L'assemblage sur plaque pastillée ou à bandes fait généralement suite à l'essai sur platine à essais. Je m'abstiendrai cependant de faire l'illustration de cette dernière ici, car tout simplement illisible. Comme dans bien des cas, l'illustration avec le logiciel Fritzing prend plus de temps que l'assemblage et la soudure effective des composants...

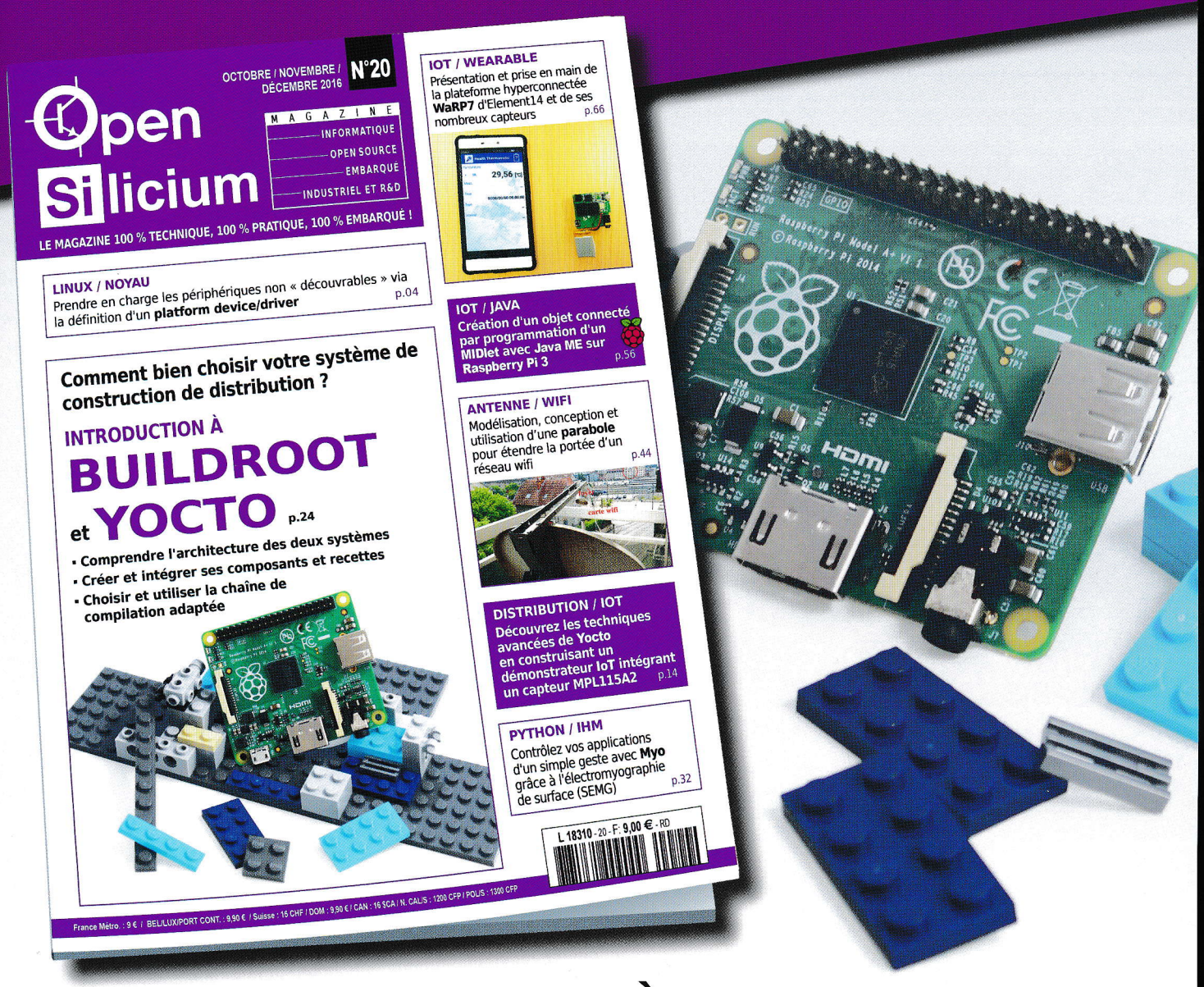
Le montage final (ou presque) sur plaque à bandes (stripboard). Je considère ceci comme une première version, équipée en prime d'une led témoin. Les prochaines générations utiliseront des composants en surface (CMS), un circuit imprimé maison et peut-être des potentiomètres permettant ajuster les temporisations au cas pas cas.

Il nous suffira ensuite de protéger le circuit, avec de la gaine thermorétractable par exemple, et d'utiliser le câble ainsi amélioré pour brancher le montage devant être autonome. J'ai ainsi laissé fonctionner sur batterie un simple croquis sur une carte Arduino Nano pendant plusieurs jours sans la moindre coupure d'alimentation de la part de la batterie. Mieux encore, après presque une semaine de fonctionnement, les quatre leds placées sur la batterie, affichaient toujours une charge entre 80 et 100% (les 4 leds allumées). Ni le circuit de maintien en éveil, ni le montage Arduino ne semblent avoir un impact notable sur la batterie et promettent ainsi une capacité de fonctionnement autonome qui laisse rêveur... **DB**



ACTUELLEMENT DISPONIBLE

OPEN SILICIUM N°20 !



COMMENT BIEN CHOISIR VOTRE SYSTÈME DE CONSTRUCTION DE DISTRIBUTION ? INTRODUCTION À

BUILDROOT & YOCTO

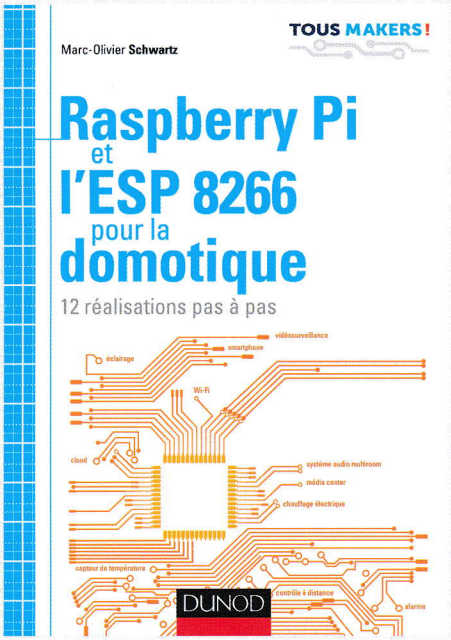
NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

TOUS MAKERS !

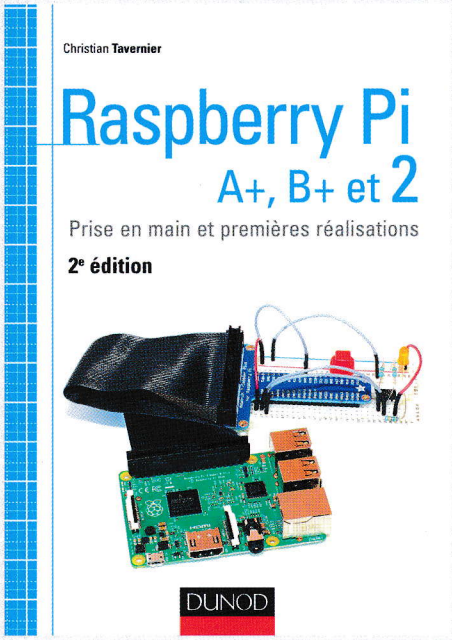
RÉVÉLEZ VOTRE POTENTIEL PAR LA CRÉATION



9782100746835, 200 p., 24,90 €

MARC-OLIVIER SCHWARTZ

Initiez-vous à la domotique avec le Raspberry Pi associé à la puce Wi-Fi ESP 8266 : 12 projets concrets pour rendre votre maison plus « intelligente ».



9782100742639, 352 p., 24,90 €

CHRISTIAN TAVERNIER

Cet ouvrage décrit les différentes versions de Raspberry Pi et explique comment le configurer et le paramétrer correctement.

9782100738106, 192 p., 22 €

PATRICE OGUIC

Un guide pas à pas pour construire votre machine CNC, idéale pour la gravure et le perçage des circuits imprimés, les maquettistes et les modélistes.

9782100742561, 160 p., 19,90 €

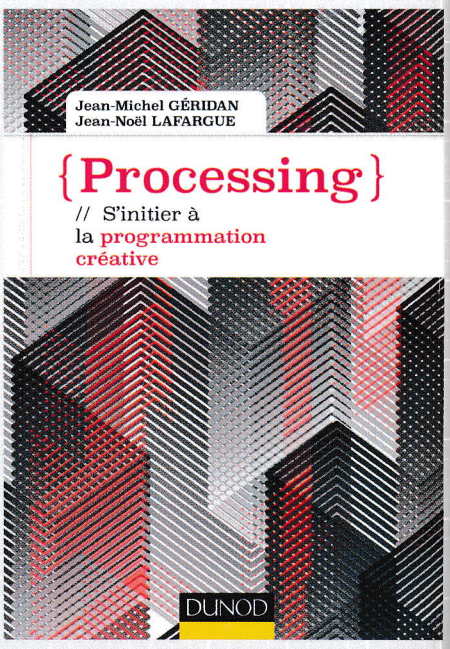
PASCAL LIÉGEOIS

Réalisez un drone terrestre radiocommandé, contrôlable à vue ou à distance, et pouvant embarquer un appareil photo ou une caméra.

9782100746842, 160 p., 19,50 €

FRANÇOIS BRUMENT, MAËLLE CAMPAGNOLI

À travers 70 réalisations, cet ouvrage présente les nombreuses possibilités de l'impression 3D, afin de comprendre en quoi elle va tout changer.



9782100737840, 280 p., 29 €

JEAN-MICHEL GÉRIDAN, JEAN-NOËL LAFARGUE

Tout savoir sur Processing, le logiciel open source pour les créateurs qui veulent produire des installations interactives.

TOUT LE CATALOGUE SUR **WWW.DUNOD.COM/TOUS-MAKERS**

DUNOD
ÉDITEUR DE SAVOIRS