N°14 | SEPT. - OCT. 2016

HACKABLE MAGAZINE

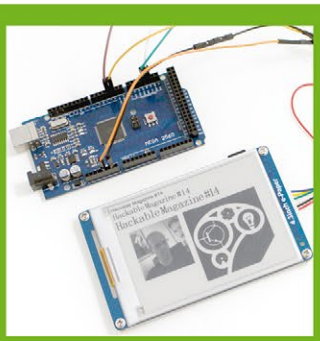
DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

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

ARDUINO

Apprenez à piloter un module à papier électronique pour afficher textes et images

p. 26



ACTU / CARTE

Prise en main et test du CHIP, le nano-ordinateur concurrent de la Pi ... ou pas

p. 14

RASPBERRY PI

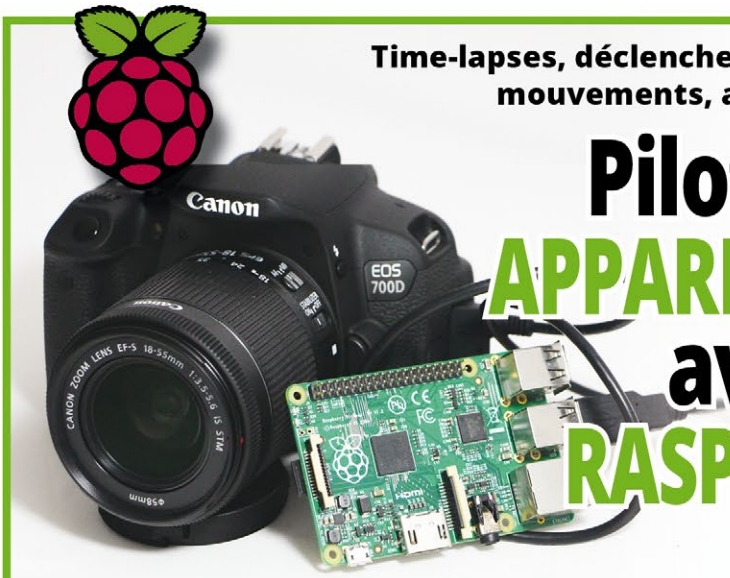
Faites de votre Pi un point d'accès Wifi bridge et personnalisez son nom

p. 86

ALIMENTATION

Apprenez à alimenter correctement et en toute sécurité vos projets à base d'Arduino

p. 94



Time-lapses, déclenchements par capteur de mouvements, annotation d'images...

Pilotez votre APPAREIL PHOTO avec votre RASPBERRY PI !

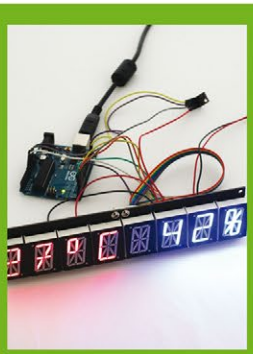
p. 50

Canon, Nikon, Panasonic, Sony... Reflex, hybride, compact, bridge... plus de 1800 modèles supportés

ARDUINO

Créez un thermomètre hygromètre pour améliorer votre confort et votre environnement

p. 38

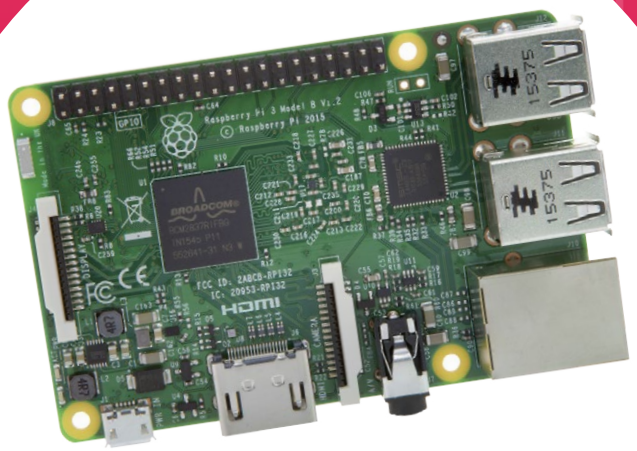


ACTU / SYSTÈME

Découvrez Tizen, le système au cœur de la smartwatch Samsung Gear S2 et du smartphone Z3

p. 04

L 19338 - 14 - F: 7,90 € - RD



Raspberry Pi 3

KUBii

Votre boutique On-line

Raspberry Pi

Avec Raspberry Pi, une multitude de fonctionnalités s'offrent à vous ...

Découvrez l'univers du **Raspberry Pi**
et plus de **250 accessoires !**

Code KDO : **HACKABLE**

5€ pour 100€ d'achat

www.kubii.fr

ÉDITO

Attrapez-les tous !

À moins de vivre dans une grotte et totalement déconnecté du monde, vous avez sans doute du observer des comportements étranges autour de vous : des utilisateurs de smartphones l'œil rivé sur l'écran semblant errer, perdus, avant de subitement changer de direction ou se précipiter quelques dizaines de mètres plus loin, parfois en témoignant de façon vive et bruyante une certaine excitation. Ne vous inquiétez pas, c'est normal, ce syndrome s'appelle Pokemon Go...

Je ne suis pas fan de Pokemon, mais il y a dans ce phénomène, ou cette mode, quelque chose qui m'a attiré : la possibilité d'utiliser ce système pour des projets et des montages amusants. Nous avons là une sorte de système de géocaching virtuel utilisant une masse de données impressionnante, et ce avec un aspect interactif intéressant. Bref, un terreau propice à la bidouille.

Seulement voilà, la société éditrice du jeu, Niantic, ne semble pas apprécier du tout qu'on joue avec ses jeux d'une autre manière que celle initialement prévue (même si c'est bien plus drôle que d'attraper des créatures trimbalant un poireau). Ainsi, bien qu'il soit techniquement possible de se construire, par exemple, un détecteur de Pokemon à base de Raspberry Pi, ceci est totalement contraire aux conditions d'utilisation (TOS) et peut vous valoir une suspension de votre compte utilisateur, sinon pire encore.

Nous avons donc une communauté de développeurs « rebelles », essayant par tous les moyens de créer une interface de programmation (une API) ou d'utiliser celle, non publique, déjà existante et, en face, un éditeur essayant, par tous les moyens aussi, de restreindre son utilisation, voire de sanctionner ceux qui s'en servent. Personnellement, je vois cela comme un beau gâchis de temps, d'énergie, d'efforts et de ressources...

Bien entendu, on peut parfaitement comprendre que l'objectif poursuivi est tout simplement d'empêcher les joueurs de tricher, mais dans ce cas, pourquoi ne pas disposer d'une API interne réservée au jeu et en proposer une autre, publique, fournissant simplement des informations basiques, accessibles après enregistrement ? Après tout, c'est ce que font déjà d'autres sociétés et en particulier Google, Twitter et Facebook pour leurs services...

Je crois que des choses m'échapperont toujours concernant les choix et la politique de certains acteurs du monde super-connecté dans lequel nous vivons. L'ouverture, la standardisation et la proximité avec les développeurs/hackers les plus inventifs sont autant de principes qui ont, pourtant, déjà fait leurs preuves à maintes reprises...

Devant un tel manque d'ouverture, la réaction la plus intelligente est donc de laisser les ténèbres, l'obstination et l'obscurantisme là où ils sont et passer paisiblement son chemin... (en pestant)

Denis Bodor

Hackable Magazine

est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIÈRE GRISE

10, Place de la Cathédrale - 68000 Colmar
Tél. : 03 67 10 00 20 – Fax : 03 67 10 00 21
E-mail : lecteurs@hackable.fr
Service commercial : cial@ed-diamond.com
Sites : www.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Kathrin Scali
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
Service des ventes : Abomarque : 09 53 15 21 77
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : 2427-4631
Commission paritaire : K92470
Périodicité : bimestriel
Prix de vente : 7,90 €

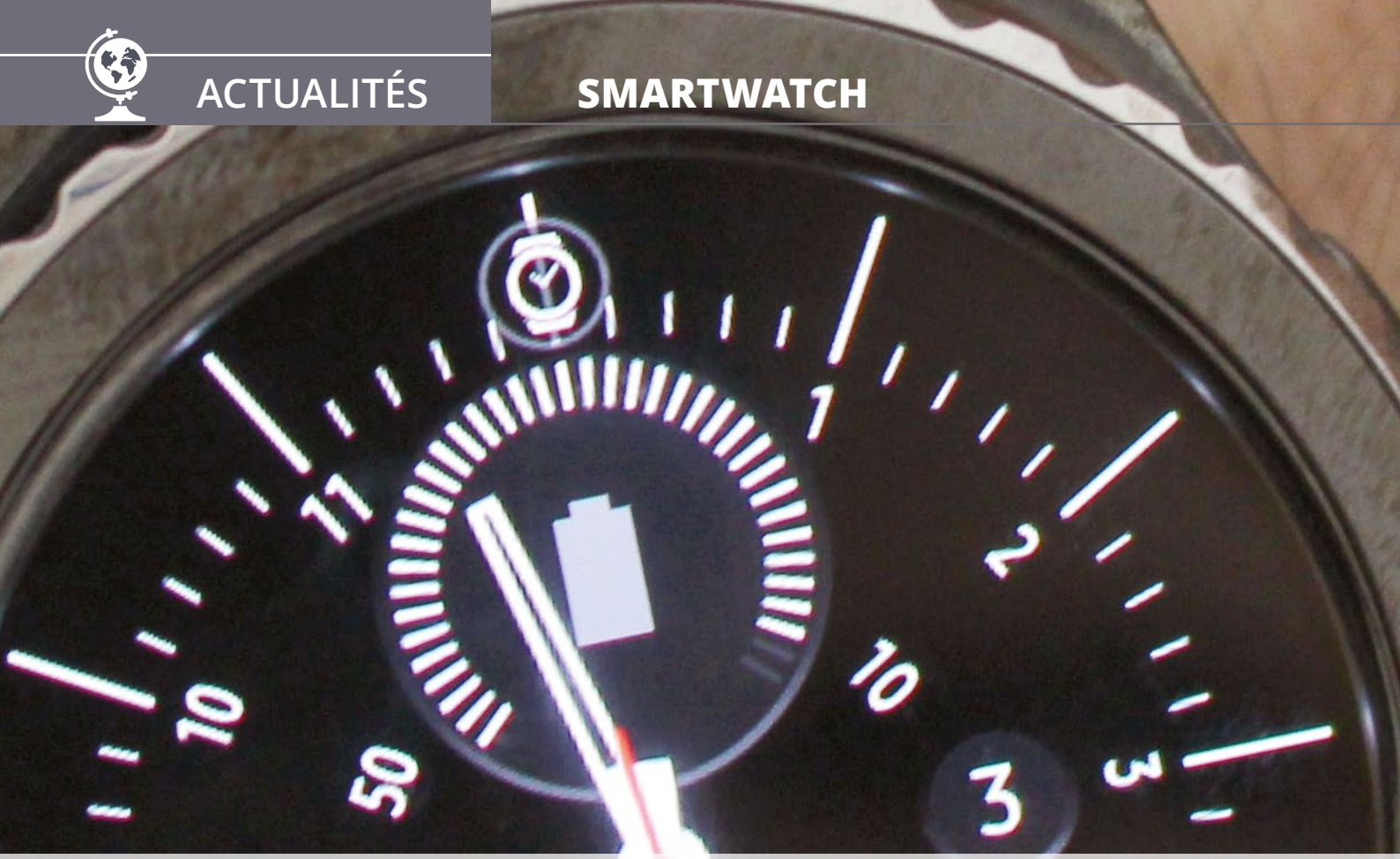
Suivez-nous sur Twitter
@hackablemag

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



TIZEN, L'AUTRE PLATEFORME MOBILE OPEN SOURCE...

Denis Bodor

... quand Samsung décidera de mettre vraiment ce système en avant et qu'il y aura des applications. On entend parler de Tizen depuis des années, mais même si des produits existent effectivement, Samsung reste très (trop) discret sur le sujet. Alors que la smartwatch Gear S2 3G est arrivée sur le marché cet été, il est temps de jeter un œil au système qui l'anime. Non ce n'est pas Android, mais Tizen...

Que trouvons-nous sur le marché du smartphone et des tablettes actuellement ? Deux OS se partagent clairement le terrain de chasse (le gibier c'est nous, mais on dit « client » pour pas qu'on s'effarouche) : iOS et Android. En dehors de ces vedettes figurent quelques outsiders ou dinosaures, mais également un système qui a un fort potentiel : Tizen.

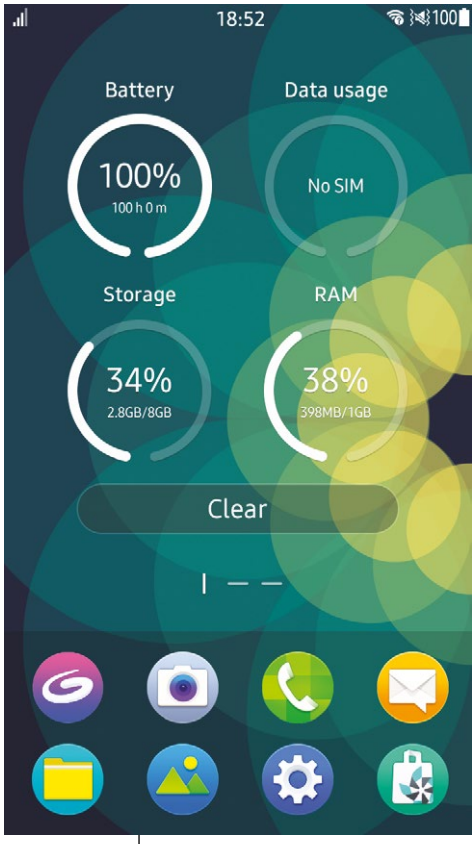
La genèse de Tizen remonte à MeeGo, lui-même né de la fusion du projet Maemo initié par Nokia et du Moblin d'Intel. Cela remonte à quelques 7 ans maintenant, mais certains se souviendront avec nostalgie de leur Nokia N810 ou N900. Depuis, les choses ont bien changé et Android occupe aujourd'hui plus de 80% des unités utilisées. Derrière les 15% d'iOS de l'iPhone, plusieurs « petits » systèmes se partagent les miettes qui restent. Parmi eux se trouve Tizen, un système open source (comme Android), basé sur Linux (comme Android), et destiné à fonctionner sur smartphones, smartwatchs, équipements automobiles, appareils photo numériques, téléviseurs, etc. (oui, comme Android).

Tizen est le successeur de la plateforme LiMo (LInux MObile) dont MeeGo était une implémentation, mais n'est pas pour autant un dérivé de MeeGo, mais en réalité d'une implémentation de référence réalisée par Samsung dans le cadre de la plateforme LiMo. La fondation LiMo est à présent devenue l'association Tizen regroupant, entre autres, Samsung, Intel, Huawei, Fujitsu, NEC, Panasonic, Orange ou encore Vodafone, et toutes les références pointant initialement sur LiMo sont maintenant redirigées vers Tizen.

Bien que partageant techniquement des origines et des bases communes, les visions propres à Android et Tizen sont très différentes. Sous Android, vous développez principalement vos applications avec un environnement dédié appelé Android Studio et ce en Java. Avec Tizen, Java n'est pas de la partie et les applications peuvent être écrites en HTML5 plus d'autres technologies Web (JavaScript, jQuery, etc.) ou sous forme d'applications natives en C.

Autre détail intéressant, il n'est pas nécessaire d'utiliser l'environnement graphique de développement (IDE) pour créer ses applications. Il est parfaitement possible, dans la grande tradition du développement UNIX/Linux d'utiliser l'éditeur de son choix et la ligne de commandes, sans le moindre problème. On reconnaît là, sans le moindre doute, la trace de développeurs de la vieille école et l'influence évidente de gens comme Carsten Haitzler, alias Rasterman, créateur des *Enlightenment Foundation Libraries* (EFL) et du gestionnaire de fenêtres Enlightenment (si vous ne connaissez pas, installez le paquet **`e17`** sur votre Pi, ainsi que **`terminology`** et vous aurez une très bonne idée de ce que permettent de faire les EFL).

L'aspect technique plus proche d'un système GNU/Linux standard, une « culture Linux » plus présente et une utilisation



L'interface de Tizen ne déroutera pas un utilisateur d'Android ou d'iPhone. Ergonomiquement parlant c'est une sorte de mélange s'inspirant des deux autres systèmes. Certains points sont sans doute perfectibles, mais dans l'ensemble c'est avant tout une question de goût.

Le Samsung Z3 n'est pas commercialisé en Europe et est réservé, comme le Z1, au marché indien où il est distribué depuis octobre 2015. On remarquera la présence d'un accumulateur amovible et d'un double emplacement SIM, deux caractéristiques très rares chez nous.

massive de standards et de projets ouverts comme la GNU libC, X11, Wayland, ConnMan (gestion réseau), oFono (téléphonie) ou Smack (sécurité et isolation des applications web) en font, à mon sens, quelque chose de bien plus attrayant qu'Android et assez proche de la liberté technique qu'on retrouve avec une Raspberry Pi.

1. LE POURQUOI DU COMMENT

Il y a quelques années, j'avais débuté la rédaction un article sur Tizen et la façon de développer des applications pour cet environnement. Le projet d'article s'est cependant rapidement essoufflé tant l'installation des outils et du kit de développement (SDK) était pénible. Il était littéralement nécessaire de bidouiller dans tous les coins pour ne serait-ce qu'obtenir une installation fonctionnelle.

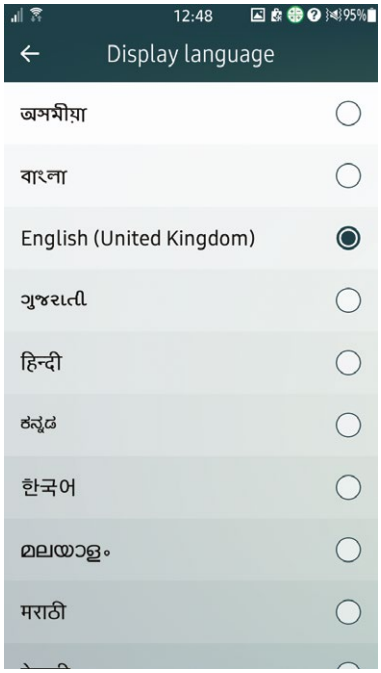
Dernièrement cependant, mon attention a été attirée par ce que je considère comme un gadget amusant : une montre Samsung Gear S2.

Mais mon intérêt premier n'était pas encore Tizen, mais simplement d'utiliser une smartwatch d'une taille raisonnable (pas une horloge comtoise plus large que mon poignet) m'offrant l'opportunité d'y installer des applications « maison » par la suite. Après une prise en main des plus pénibles en raison du fait qu'il s'agissait d'une édition « Asia » et que je ne possède pas/plus de smartphone, et après avoir dû utiliser le smartphone Samsung d'une collègue, j'ai décidé de jeter un œil sans attendre à cette opportunité de développement. Et là, surprise ! Les développeurs de Tizen et du SDK ont fait un sacré bout de chemin !

Non seulement l'environnement de développement (**https://developer.tizen.org/development/tools/download**) est disponible à la fois pour Windows, Mac OS X (ou « macOS » comme il faut dire maintenant) et Ubuntu, mais ce en 32 et 64 bits... et avec, au choix, une interface graphique ou en ligne de commandes. L'installation se passe sans le moindre problème (avec un système GNU/Linux Debian) et l'environnement (basé sur Eclipse) permet de rapidement créer un « HelloWorld ». Certes, l'installation d'applications maison sur la montre passe par l'enregistrement de certificats en ligne, à la fois pour le périphérique et le programmeur, selon une procédure relativement simple, mais dont on se passerait volontiers, mais on arrive, en une après-midi à créer sa première application vaguement utile (en l'occurrence un widget affichant l'adresse IP obtenue par la montre via wifi).

La Samsung Gear S2 est l'un des rares périphériques Tizen disponibles en Europe. Ici le modèle « classic » passe presque pour une montre standard contrairement aux autres modèles du marché aux cadrans démesurés comme la Moto 360.

L'élément clé ici est la possibilité de développer à la fois des applications web HTML/JS, mais aussi, et surtout des applications dites natives, en C et reposant sur un environnement de programmation relativement simple. En ce qui me concerne, c'est en particulier ce point qui fait de Tizen quelque chose de réellement intéressant puisqu'on se trouve dans une logique de programmation que je pourrai qualifier de « classique ». Ainsi, même si vous n'êtes pas coutumier de l'environnement Tizen, ceci n'est pas très éloigné de ce qu'il faut connaître pour créer une application C sous Linux utilisant les EFL, qui n'est somme toute par très différent de la programmation C/C++ qu'on trouve sur bien des plateformes PC, Arduino, Launchpad, RPi/WiringPi, etc.

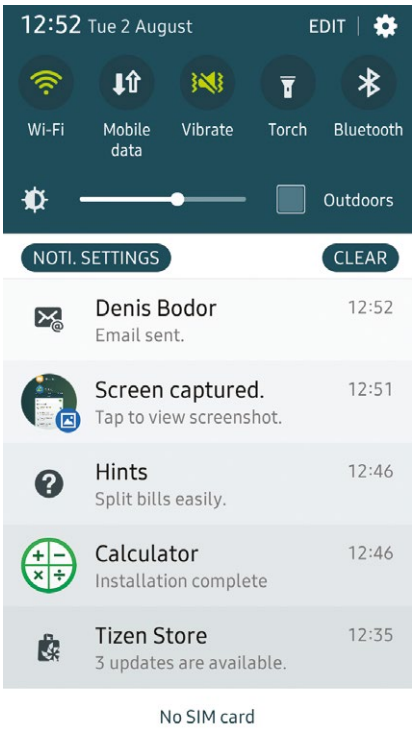


Le principal problème dans l'achat d'un smartphone destiné à un marché étranger est la liste de langues supportées. Ici, la seule option possible est clairement l'anglais.

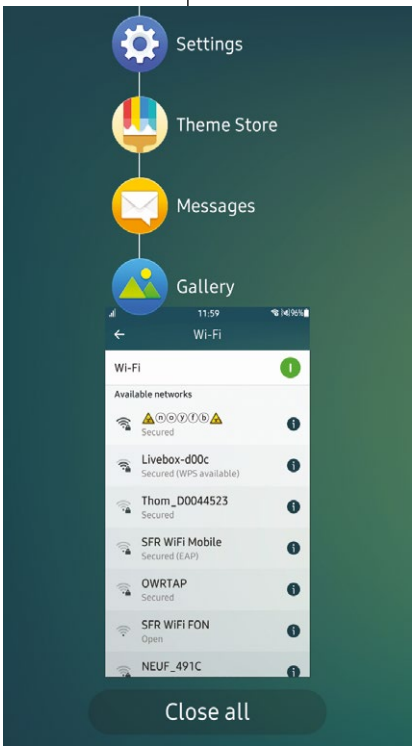
Ceci est très différent de la philosophie de développement sur Android ou iOS avec des langages comme Java ou Objectif-C qui peuvent paraître « extraterrestres » par rapport aux classiques que sont C, C++ ou Python. Cela fait de Tizen un système qui me semble très intéressant pour débuter le développement d'applications mobiles même si deux importants problèmes sont au rendez-vous si vous vous lancez dans l'aventure...

2. DISPONIBILITÉ DU MATÉRIEL

Ayant commencé à jouer avec le SDK, l'environnement, les exemples et le matériel lui-même, on en vient naturellement à vouloir voir plus grand. Ma première expérience m'a donc conduit à la recherche d'un périphérique utilisant Tizen pour étendre mes possibilités de développement. Et c'est là que le problème le plus important se fait jour : il y a, dans les faits, très peu de périphériques disponibles.



L'écran de notification déroulable depuis le haut est très proche de ce qu'on trouve sous Android si ce n'est que les raccourcis sont configurables.



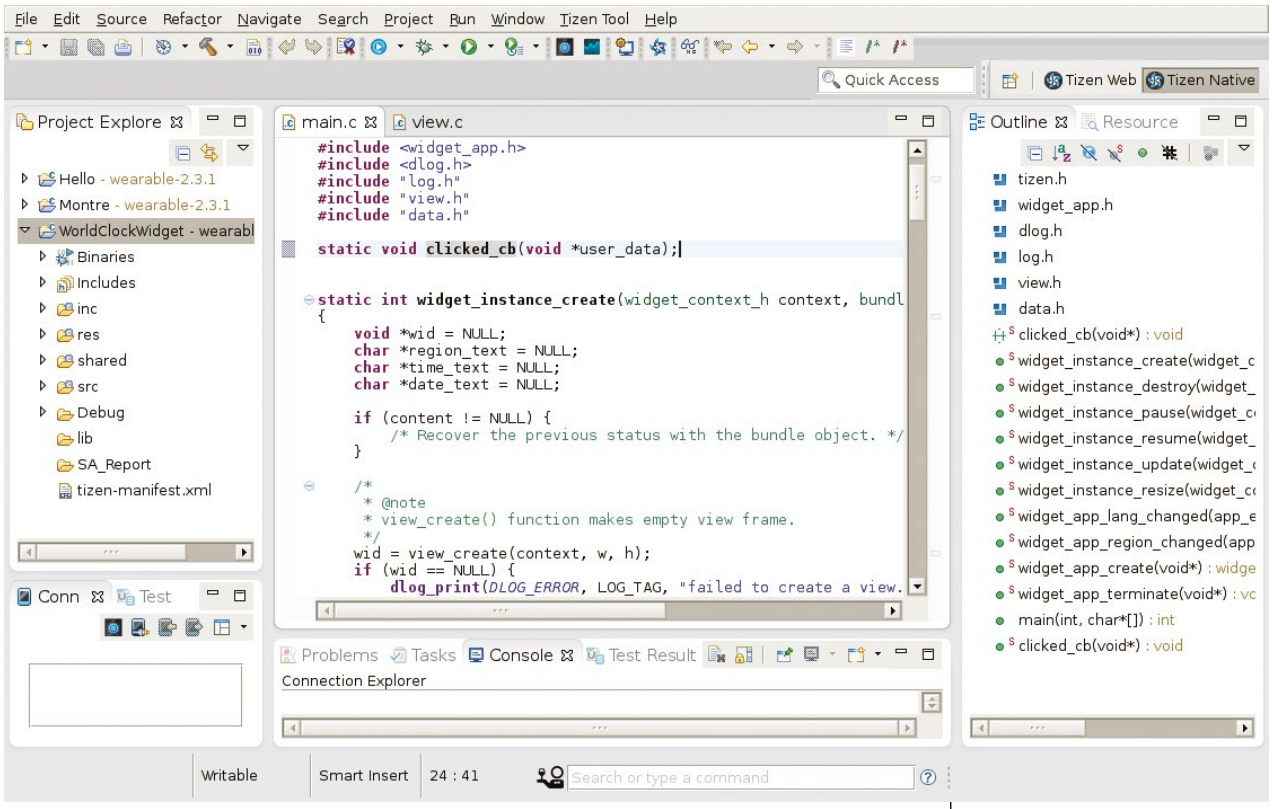
Tizen est multitâche, une pression maintenue sur le bouton physique en façade liste les applications en cours d'exécution qu'on peut alors terminer en les glissant sur le côté.

Nous avons tout d'abord les smartwatchs comme Gear 2, Gear S, Gear S2, Gear Fit et la prochaine Gear S3 (en septembre 2016 théoriquement). C'est sans le moindre doute le périphérique mobile le plus facile à acquérir puisque la plupart des modèles sont disponibles chez les distributeurs de produits électroniques. Les téléviseurs Samsung représentent également une option puisque tous les modèles fabriqués par Samsung depuis 2015 fonctionnent sous Tizen. Étant cependant totalement anti-TV (et anti-taxe-TV) depuis plus de 10 ans, je ne vais toutefois pas céder pour le seul plaisir de développer pour un périphérique qui ne m'intéresse pas.

Certains appareils photo numériques comme le NX1 ou le NX500 peuvent être une option intéressante, à condition d'y mettre le prix et en partant du principe qu'on ne possède pas déjà un appareil hybride ou reflex digne de ce nom. L'option du Gear 360, un appareil photo/vidéo 360° pourrait être tentée puisque l'objet pourra avoir un usage « normal » original, mais le fait qu'il ne dispose pas d'interface utilisateur (écran) limite grandement son usage en guise de plateforme de développement.

Si on écarte les produits exotiques que sont les réfrigérateurs ou encore l'adaptateur Samsung OBD II pour votre voiture, on en arrive tout naturellement à la solution qu'on envisagerait en premier lieu : le smartphone. Et là, on se heurte à un gros problème, sinon à un mur, car même s'il existe effectivement deux modèles de mobiles sous Tizen que sont le Z1 (01/2015) et le Z3 (10/2015), ceux-ci ne sont pas disponibles en Occident. En effet, ces produits sont réservés au marché indien (plus le Bangladesh et le Népal) et sont extrêmement difficiles à obtenir en Europe. Ce qui est d'autant plus frustrant que le prix annoncé de 5590 roupies indiennes équivaut à environ 75 euros.

L'obtention d'un tel périphérique relève donc soit d'une chasse au trésor, soit de la mise en œuvre de contacts internationaux qui vous permettraient d'acquérir l'objet (l'option de deux fois 8h de vol Paris - New Delhi pour un budget à 4 chiffres n'est pas raisonnable). La seule option restante consiste donc à fouiller les sites comme eBay ou Amazon et d'attendre la bonne occasion. Chose que j'ai fait pour finalement mettre la main sur un Z3 quasi neuf à quelques 230€, sur eBay, en provenance d'Allemagne.

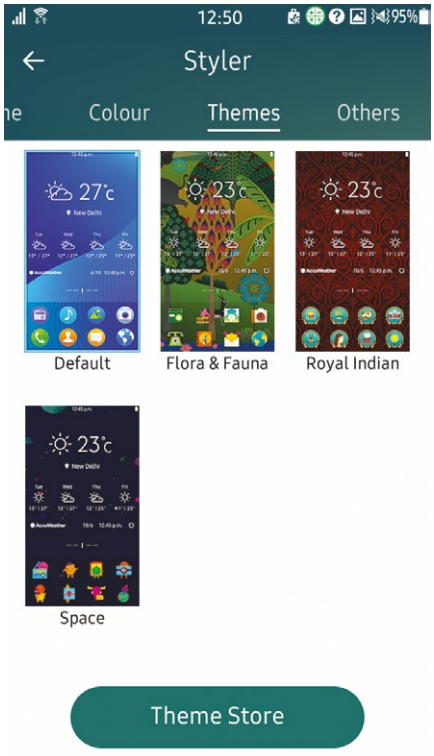


Achat que je ne regrette pas, car les spécifications, même au prix payé sont tout à fait sympathiques :

- livré avec Tizen 2.3 et mise à jour vers 2.4,
- processeur quatre cœurs à 1,3 Ghz,
- processeur graphique Mali-400,
- 1 Go de RAM,
- 8 Go de stockage interne,
- un emplacement microSD (jusqu'à 128 Go),
- deux emplacements SIM,
- écran Super AMOLED 5 pouces 720*1280 pixels,
- caméra 8 MP avec flash à l'arrière + 5 MP en façade,
- connectivité Wifi 802.11 b/g/n + Bluetooth 4.0 LE,
- radio FM intégrée,
- géolocalisation GPS + GLONASS,
- et finalement accumulateur amovible de 2600 mAh (oui, oui, amovible !).

On ne peut s'empêcher de se demander pourquoi diable un tel produit n'existe pas pour le marché occidental qui doit supporter une déferlante d'accumulateurs intégrés et de disparition progressive des emplacements microSD, le tout à des prix qui font froid dans le dos. Certes le Z3 n'est pas 4G, mais il serait loin d'être le seul modèle d'entrée de gamme à se limiter au GSM/HSPA. Autre limitation de ce téléphone dans l'état, le nombre de langues disponibles aussi bien pour l'interface que pour le clavier est très limité : seul l'anglais est utilisable puisqu'aucune autre langue occidentale n'est installée.

Le principal intérêt de Tizen est sans conteste l'environnement de développement disponible pour Windows, Mac et Linux, permettant de développer des applications « maison » soit en HTML/JS, soit en C. Et ce, au choix, avec l'environnement graphique ci-contre basé sur Eclipse, soit en ligne de commandes avec son éditeur de code préféré (Vim donc).



En termes d'interface la grosse spécificité de Tizen est la personnalisation. Il est possible de modifier presque tous les éléments graphiques du système et une importante quantité de thèmes sont disponibles.

On ne peut qu'espérer un prochain changement qui selon les rumeurs devrait succéder à l'annonce de la disponibilité de Tizen 3.0 en septembre. Je me permettrai cependant d'émettre la plus grande réserve concernant l'arrivée d'un Z2 ou d'un Z5 en Europe même si la version 3.0 du système risque effectivement d'être disponible. En effet, fin 2015 bon nombre de sites couvrant l'actualité des smartphones ont annoncé l'arrivé imminente du Z3 dans 11 pays d'Europe, dont la France, sans pour autant que cela soit effectivement suivi des faits.

3. CARENCE D'APPLICATIONS

En termes de critiques pour les rares personnes ayant eu l'occasion de tester et commenter l'utilisation d'un Samsung Z3 par exemple, le verdict est souvent le même : bon appareil, interface très personnalisable, mais bien trop peu d'applications.

Bien entendu, le système est encore relativement jeune (en particulier la 2.3) et quelques critiques ont également été faites sur l'ergonomie générale, l'utilisation du bouton physique ou encore l'absence de rotation de l'écran d'accueil, mais ceci est avant tout une affaire de goût. Ce qui ne l'est pas en revanche c'est le millier d'applications disponibles dans le *Tizen Store*, face à la concurrence que sont

La smartwatch Gear S2 intègre à l'arrière un capteur permettant de mesurer le rythme cardiaque. C'est l'une des fonctionnalités de fitness intégrée qui devrait ravir les sportifs...

Le Z3 possède les caractéristiques d'un smartphone de moyenne gamme avec une finition exemplaire tout en restant sobre. Il ne lui manque que deux choses : la 4G et une fonctionnalité NFC.

Android et iOS où ce nombre s'exprime en millions. Il faut cependant mitiger cette comparaison en rappelant que le Play Store de Google, tout comme l'App Store d'Apple, sont littéralement infestés d'applications inutiles ou redondantes. Une simple recherche de « lampe torche » et vous en aurez la parfaite démonstration.

Certes l'offre d'applications gratuites comme payantes permet une certaine diversité et dans la masse on trouve parfois des petites merveilles. Mais la question n'est pas de savoir quelle plateforme a le plus d'applications disponibles, mais laquelle a les bonnes applications. Et c'est là que réellement Tizen est à la traîne, car la plupart des applications phares manquent à l'appel pour séduire l'utilisateur lambda (Facebook Messenger, Instagram, WhatsApp et Dropbox sont disponibles dans le *store*, mais ne sont pas suffisants pour séduire le grand public).

Si comme moi, vous préférez voir dans votre smartphone ou smartwatch une plateforme de développement pour des projets personnels, ce problème n'entre pas vraiment en ligne de compte. Au pire, le navigateur, le client mail et l'outil de messagerie intégrés suffiront amplement pour un usage courant. Mais ce n'est pas là un usage qu'on peut considérer comme « normal » ou digne de pouvoir s'adresser à un marché concurrentiel aux leaders du domaine.

L'absence d'un nombre important d'applications et surtout d'applications vedettes est donc bel et bien un énorme handicap pour Tizen.

L'écriture d'applications et de widgets, comme ici le fruit de ma première tentative de programmation, est relativement aisée pour qui connaît un minimum le C ou HTML5. L'environnement fournit bon nombre d'exemples dont on peut s'inspirer et on a ainsi très vite le pied à l'étrier.

CONCLUSION

La plateforme Tizen peut se prêter à une utilisation bien plus dans l'esprit « bidouille » que ne le permettent Android ou iOS. C'est avant tout une question de philosophie globale et d'utilisation des technologies en œuvre. Il y a bien entendu un travail d'apprentissage non négligeable à prévoir pour comprendre et utiliser l'environnement de développement et la logique des EFL (dans le cas d'un développement natif), mais ceci me semble bien plus accessible que les deux principaux systèmes du marché.

Le manque d'applications découle à mon sens directement de la non-disponibilité du matériel. Bien sûr les « grosses applications » que sont Facebook, Instagram, Twitter, Snapchat, WhatsApp, etc. jouent un rôle important pour l'utilisateur, mais un écosystème d'applications en tout genre n'en est pas moins important. Or celles-ci sont principalement développées par des particuliers ou des petites entreprises qui, sans pouvoir mettre la main facilement sur un appareil compatible, ne peuvent tout simplement rien créer. Personne ne se lance dans le développement d'une véritable application sur la base d'un simple émulateur...

Ma conclusion toute personnelle est donc que malgré les difficultés rencontrées, Tizen est une plateforme furieusement intéressante que je continuerai d'explorer et un système que je recommande si, comme moi, vous aimez fouiller et vous amuser à créer des applications en fonction de vos besoins. Mais à moins d'une évolution dans la disponibilité du matériel suite à l'arrivée de Tizen 3.0, la très faible part de marché de ce système ne justifierait pas que je m'étende davantage sur le sujet ici, ce que je regrette énormément...

À noter enfin que le blog de l'*Open Source Group* de Samsung parle d'un portage en cours de Tizen sur Raspberry Pi 2, mais toutes les informations, images, fichiers et sources datent de 2015 et les travaux ne semblent pas avoir avancé depuis. Peut-être que cela reviendra au goût du jour d'ici la fin de l'année et que le projet redémarrera avec la finalisation de Tizen 3.0. Adresser ainsi le marché de l'électronique de loisir serait une excellente carte à jouer de la part de Samsung.

Tizen a un potentiel de « bidouillabilité » énorme et pourrait devenir le compagnon idéal pour quelqu'un usant de Raspberry Pi et d'Arduino pour se confectionner son environnement électronique. Croisons les doigts, avec un peu de chance lorsque vous lirez ceci Samsung se sera plié de quelques annonces sympathiques et les choses pourront enfin bouger dans ce sens... **DB**

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

HACKABLE MAGAZINE

PROFESSIONNELS !

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...

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

www.ed-diamond.com

PDF COLLECTIFS PRO

Prix TTC en Euros / France Métropolitaine

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

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

ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO OPEN SILICIUM

Prix TTC en Euros / France Métropolitaine

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

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

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

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

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

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

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

HACKABLE MAGAZINE

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

Vos remarques :

.....

.....

LE C.H.I.P : LE TUEUR DE FRAMBOISES ?

Denis Bodor

J'avoue que ce titre a tout de l'accroche racoleuse du tabloïd le plus pathétique, mais c'est ainsi qu'a plus ou moins été perçu le CHIP lors de son annonce. En effet, ceci remonte à une date où le Raspberry Pi Zero n'était même pas encore une rumeur et l'argument le plus percutant était alors tout simplement « le premier ordinateur au monde à 9 dollars ». Le CHIP est depuis passé de la phase de financement participatif et celle de la livraison aux contributeurs Kickstarter, à celle de la précommande avec une production en masse et un début de livraison prévue pour octobre. Le moment parfait pour vous faire découvrir la bête…

Le CHIP de *Next thin Co.* (abrégé NTC) a été annoncé à grand fracas lors du lancement de sa campagne de financement participatif (ou en anglais *crowdfounding*) en mai 2015. Avec un objectif de 50000$, ce sont finalement quelques 2 millions de dollars qui ont été engagés par près de 40000 contributeurs avec des premières livraisons fin 2015 et durant le premier semestre 2016. Parmi ces contributeurs se trouvait votre dévoué rédacteur en chef.

Les « récompenses » pour différentes contributions étaient constituées de plusieurs éléments :

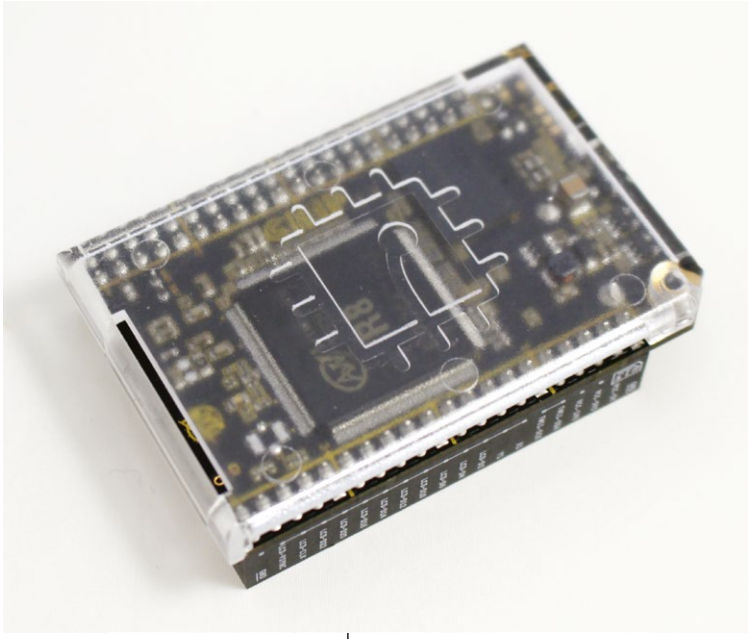
- le CHIP lui-même, se présentant comme un ordinateur mono-carte de 40x65mm ;
- un jeu d'accessoires comprenant un accu LiPo, et deux cartes additionnelles appelées DIP permettant d'ajouter une sortie VGA ou HDMI ;
- le PocketCHIP, une sorte de station d'accueil pour le CHIP fournissant un écran tactile LCD 4,3 pouces, un clavier (QWERTY) et un accu, le tout dans un boîtier plastique robuste.

L'ensemble de ces éléments est maintenant disponible en précommande sur **https://getchip.com/pages/store** : CHIP (9$), PocketCHIP (69$), DIP HDMI (15$) et DIP VGA (10$). Montants auxquels il faut, bien entendu, ajouter le port (quelques 6$) et surtout les frais de douane incluant, entre autres, 20% de TVA. Instinctivement, on a naturellement envie de comparer le CHIP au Raspberry Pi Zero, annoncé à quelques 5$ (mais qui en réalité s'affiche à quelques 16€ en France). Cependant, on prendra garde de ne pas tout mélanger, car même si la tentation est grande, il ne s'agit pas du tout du même type de jouet...

Le CHIP de Next thing Co. est un minuscule ordinateur intégrant processeur ARM, 512 Mo de mémoire vive, Wifi, Bluetooth 4.0, périphériques et mémoire flash de 4 Go permettant de stocker un système GNU/Linux et de faire fonctionner l'ensemble.

1. LE CHIP

Comme la Raspberry Pi, le CHIP (oui, j'ai décidé que le CHIP était mâle) est un ordinateur mono-carte intégrant processeur, mémoires, périphériques, alimentation et une myriade d'entrées/sorties. Comme la Pi, il est alimenté en 5V par un



L'élément central du CHIP est le SoC Allwinner R8, regroupant le processeur ARM Cortex-A8 à 1 Ghz, un processeur graphique Mali 400 et un processeur vidéo Cedar Engine (encodage/décodage).

connecteur USB et fait fonctionner un système GNU/Linux Debian (Raspbian étant une adaptation de Debian). Mais dans les grandes lignes, la comparaison s'arrête là.

Au cœur du CHIP se trouve le processeur ou plus exactement le SoC (*System On a Chip*) Allwinner R8 regroupant un processeur ARM Cortex-A8 1Ghz, un processeur vidéo *Cedar Engine* (VPU) et un processeur graphique Mali 400 (GPU). Ceci s'accompagne de 512 Mo de mémoire vive et surtout de 4Go de flash. C'est là un point important, car le CHIP n'utilise pas de SD ou de microSD pour accueillir le système et les données, mais une mémoire interne de taille fixe. Ceci signifie qu'il n'est pas possible d'étendre cette mémoire puisqu'elle est soudée sur le CHIP, mais d'un autre côté, qu'il n'est pas nécessaire non plus d'acquérir une carte SD/microSD supplémentaire. Ce type d'architecture est bien plus courant dans le monde des systèmes embarqués que le fait d'avoir recours à un stockage externe. Cela impacte directement la façon de travailler avec la plateforme.

Une autre différence notable du CHIP face à une Raspberry Pi (à l'exception de la Pi 3) est l'intégration du Wifi et du Bluetooth 4.0. Là encore ceci implique qu'il n'est pas nécessaire d'ajouter des périphériques USB pour obtenir ce type de connectivité. Parlant d'USB d'ailleurs, le CHIP ne dispose que d'un seul port, comme la Pi Zéro, mais le connecteur servant à l'alimentation est un vrai port USB OTG. Ceci signifie que la carte peut, par exemple, être configurée pour devenir un périphérique USB comme une interface série, Ethernet ou autre (voir *Hackable n°11* et son article sur l'USB gadget expérimental de la Pi Zero).

Une autre spécificité propre au CHIP est la présence d'un contrôleur d'alimentation (AXP209) capable de supporter un accumulateur LiPo. Un connecteur JST 2.0 est directement intégré et permet ainsi de connecter un accumulateur (une cellule) qui sera automatiquement utilisé lorsque l'alimentation est débranchée et rechargé lorsqu'elle est présente. Ceci vous permettra de créer, sans accessoire annexe, un projet autonome. L'accu livré est marqué « 11.1 Wh » correspondant à 3000 mAh (Wh*1000/tension, soit 11,1*1000/3,7=3000), ce qui est suffisant pour alimenter un CHIP ou un PocketCHIP durant quelques 5h. Bien entendu, rien ne vous empêche d'opter pour un accu de plus grande capacité si vous le souhaitez, du moment qu'il s'agit bien d'un accu LiPo en 3,7V (valeur standard) incluant, bien sûr, un circuit de protection. La taille d'un CHIP, associé au fait qu'il n'y ait pas de partie mécanique (emplacement SD) à ce support rendent la carte très utile pour des projets demandant autonomie et intégration.

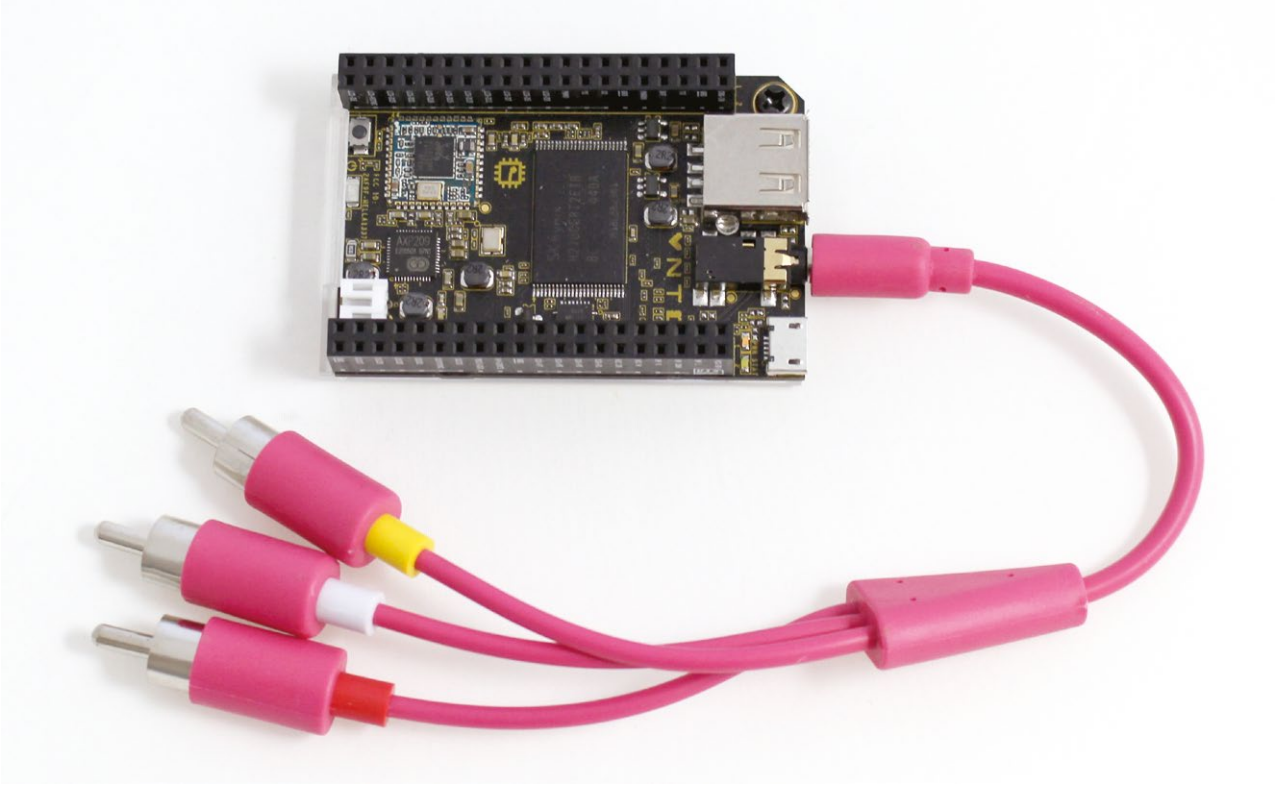
2. LES DIP

Avant toute chose, précisons que ces cartes additionnelles ne sont supportées qu'avec la version dite Debian 4.4 du système. Les CHIP ayant été commandés via la campagne Kickstarter et livrés fin 2015, sont normalement équipés du système Debian 4.3 ne supportant pas encore les DIP (qui n'ont été livrés qu'en mai). Les CHIP précommandés et qui seront livrés à partir d'octobre seront normalement équipés de Debian 4.4 et il ne sera pas nécessaire de faire de mise à jour. Si vous avez déjà un CHIP ou que vous en trouviez un sous Debian 4.3, il vous faudra d'abord passer à 4.4. Nous traiterons de la procédure simplifiée un peu plus loin dans l'article.

Les DIP sont au CHIP ce que les shields sont à Arduino. Comme le précise la documentation du CHIP, les 80 broches disponibles sur la carte en deux séries de deux rangées de 20 broches permettent de connecter toutes sortes de circuits. On y trouve ainsi de quoi commander un écran LCD parallèle, une console série (UART), diverses tensions, des masses, les sorties d'un ADC, l'audio, l'interface pour une surface tactile résistive, deux bus i2c (TWI), un SPI et huit GPIO.

Ce format et une partie des signaux sont directement utilisés par les DIP afin de, pour l'instant, proposer des sorties vidéos VGA et HDMI. Le CHIP permet, par défaut de sortir un signal vidéo via ce qui semble être une simple prise jack audio. Il s'agit en réalité d'un connecteur TRRS (pour *Tip-Ring-Ring-Sleeve*) 3,5mm fournissant une masse, l'audio droite et gauche, et un signal vidéo composite (prise RCA jaune sur les téléviseurs).

Le CHIP ne dispose pas de port VGA ou HDMI par défaut et ne propose qu'une sortie composite RCA. Parfaitement adapté pour une utilisation autonome, ceci peut être vu comme un handicap majeur face aux différents modèles de Raspberry Pi proposant tous une sortie vidéo numérique par défaut.



Vous l'avez compris, le CHIP seul ne propose donc pas de sortie directement compatible avec un moniteur, mais uniquement un signal analogique, soit en 640×480 (NTSC), soit en 720×576 (PAL). Un peu comme les premiers ordinateurs familiaux type Commodore 64 ou Thomson MO5.

Chaque DIP intègre à la manière des HAT Raspberry Pi, une EEPROM 1-wire comprenant un numéro d'identification (VID/PID) lu au démarrage du système. En fonction de cette identification, le système va alors utiliser, tout comme la Pi, un *overlay* du *device tree* pour prendre en charge et initialiser le périphérique. Il y a pour l'heure un seul VID (*Vendor IO*), `0x009d011a`, celui de *Next Thing Co*, et trois PID (*Product ID*) :

- `0x00000001` : le PocketCHIP,
- `0x00000002` : le DIP VGA,
- `0x00000003` : le DIP HDMI.

Toutes les informations concernant le fonctionnement et la conception des DIP étant ouvertes et publiques (**http://docs.getchip.com/dip.html** et **https://github.com/NextThingCo**), on peut supposer que d'autres modèles pourraient apparaître bientôt de façon indépendante de NTC.

Le PocketCHIP est une « station d'accueil » pour le CHIP, intégrant un écran 4,3 pouces en 480×272 pixels, un clavier et un accu pour un fonctionnement autonome.

3. POCKETCHIP

Le PocketCHIP est un gros DIP dans lequel s'insère un CHIP. Celui-ci fournit un écran LCD 4,3 pouces tactile (résistif) avec une résolution de seulement 480×272 pixels. Il embarque également un accu LiPo (le même que celui vendu par NTC) et dispose d'un clavier à membrane avec un agencement QWERTY. L'ensemble forme une unité autonome permettant d'avoir sur soi un système compact entièrement utilisable avec toutes les sorties et broches utiles placées au-dessus de l'écran (SPI, i2c, GPIO, etc.).

Je ne vous cacherais pas que son utilisation relève davantage du gadget que d'un ensemble ergonomique agréable à utiliser. Certes, certaines choses ont bien été pensées comme la prise en main, l'interface graphique intégrée au système ou encore le petit trou pour un stylo permettant de faire tenir le PocketCHIP debout, mais la résolution limite énormément les possibilités, tout comme le fait que l'écran tactile soit résistif et donc non multi-touch.

L'argument de vente du PocketCHIP est très orienté jeu avec, de base, l'environnement de création PICO-8 permettant de créer des jeux minimalistes. PICO-8 se présente comme un environnement de jeu, mais également une plateforme de création (musique, animation, etc.) que ses créateurs désignent comme une *Fantasy Console*. Personnellement, je ne vois pas vraiment d'intérêt dans ce type de choses même si le concept est intéressant (des programmes encodés dans les PNG, pourquoi pas), en particulier lorsqu'il s'agit d'un logiciel propriétaire vendu normalement pour quelques 15$.

Le système proposé pour le PocketCHIP n'est cependant pas intimement lié à PICO-8 et repose sur une architecture GNU/Linux Debian parfaitement standard, avec gestion de paquets, shell, outils, serveur X, etc. Rien ne vous empêchera donc de lancer des applications X parfaitement classiques si ce n'est, encore une fois, la faible résolution proposée. On trouvera cependant dans ce DIP un véritable intérêt lorsqu'il s'agira d'utiliser, par exemple, Python et PyGame pour créer de toutes pièces ses propres jeux, ou encore développer ses propres applications graphiques adaptées à l'écran.

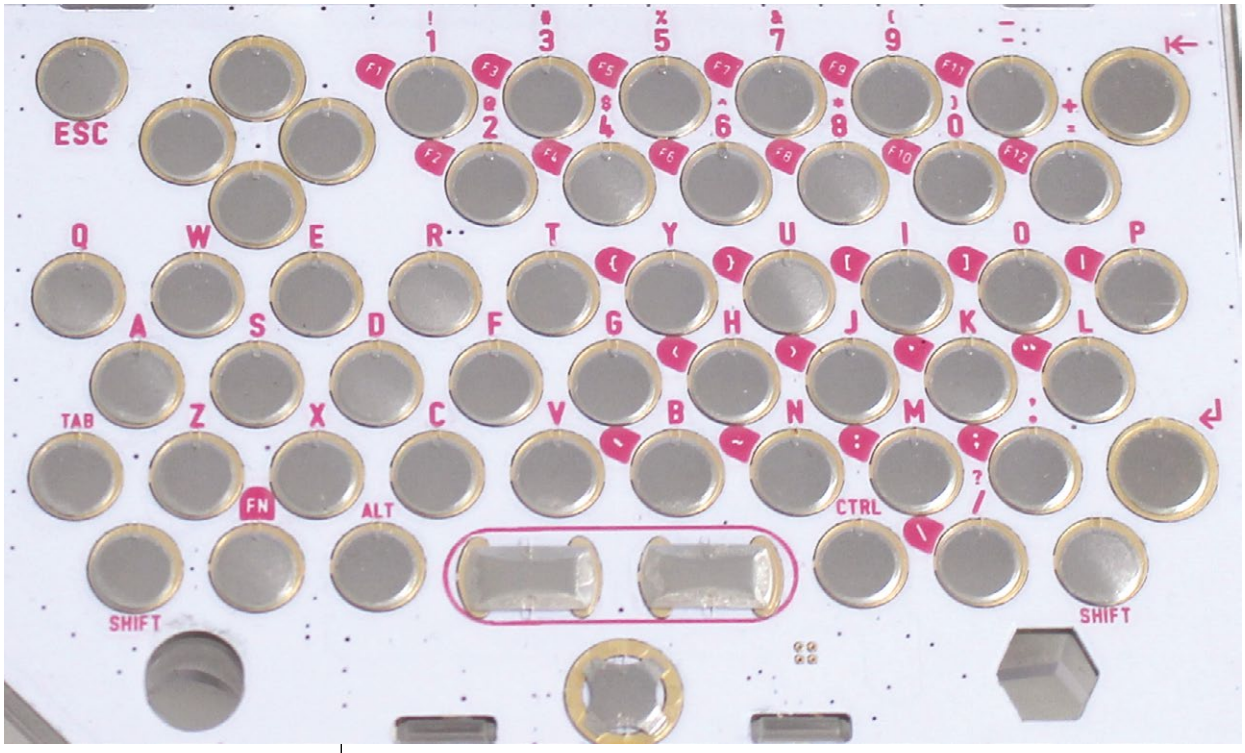
4. MISE À JOUR DU CHIP

Comme je l'ai dit, le CHIP ne démarre pas depuis une carte SD ou microSD, mais à partir de sa mémoire flash intégrée. Une fois le système en marche, vous pourrez mettre à jour la distribution Debian exactement comme avec une Pi (`apt-get update` et `apt-get dist-upgrade`). Mais pour faire évoluer l'ensemble du système en une fois, vous ne pouvez pas, comme avec une Pi, tout simplement créer une nouvelle carte SD/microSD puis démarrer dessus. Vous devez reflasher le système en connectant le CHIP à un PC/Mac alors qu'il se trouve dans un mode spécial, appelé FEL, donnant accès à la mémoire flash.

Le CHIP s'insère directement à l'arrière du PocketCHIP rendant l'ensemble des broches inaccessibles. Celles-ci sont heureusement déportées au-dessus de l'écran, permettant ainsi la connexion de montages et de circuits faits maison.

La procédure recommandée est la même quel que soit le système que vous aller utiliser (Windows, Mac, GNU/Linux) et se fera par l'intermédiaire du navigateur Google Chrome ou Chromium. Cependant, les préparatifs sont différents, car chaque système gère le périphérique dans le mode FEL de façon différente.

Sous Windows, la première chose à faire, avant même de connecter le CHIP consiste à installer un pilote spécifique, disponible via la page **http://docs.getchip.com/chip.html#instructions**. Ce fichier, `InstallDriver2.exe`, devra être installé après téléchargement, tout simplement en l'exécutant puis en redémarrant la machine.



Le clavier du PocketCHIP est formé d'un ensemble de pastilles métalliques tenues en place avec un film adhésif. L'organisation est naturellement QWERTY et un certain nombre de touches manquent à l'appel (verrouillage majuscules, insert, suppr). C'est une solution utilisable pour saisir quelques commandes ou URL, mais il est difficile d'y voir une solution ergonomique utilisable sérieusement.

Notez que la procédure de mise à jour ou d'installation d'un système sur le CHIP, lorsqu'elle est faite sous Windows peut ne pas fonctionner correctement du premier coup. Ceci provient à la fois de la façon dont fonctionne le CHIP et celle dont Windows gère les pilotes. Lorsque le CHIP est connecté en mode FEL, il démarre à partir d'un code spécifique placé en mémoire morte (ROM). Windows doit reconnaître le périphérique dans cet état. Cependant, le processus de mise à jour ne va utiliser ce mode que pour un court instant avant de basculer le CHIP en mode Fastboot pour la mise à jour. Là, Windows voit une déconnexion du périphérique et une reconnexion d'un autre avec des identifiants différents et doit donc basculer sur un autre pilote qu'il devra installer. La première tentative échoue généralement parce que ce second pilote (« *C.H.I.P Flashing Mode* ») n'est pas installé en même temps que le premier.

Il faut également ajouter à cela quelques autres problèmes potentiels :

- la qualité du USB câble utilisé : changez de câble ;
- le fait de brancher le CHIP sur un port USB 3 : connectez-le sur un port USB2 ou via un hub USB2.

Passez-moi l'expression, mais ne vous étonnez donc pas de « galérer », la gestion de périphériques et de pilotes sous Windows étant fidèle à elle-même, c'est parfaitement normal (triste, mais normal).

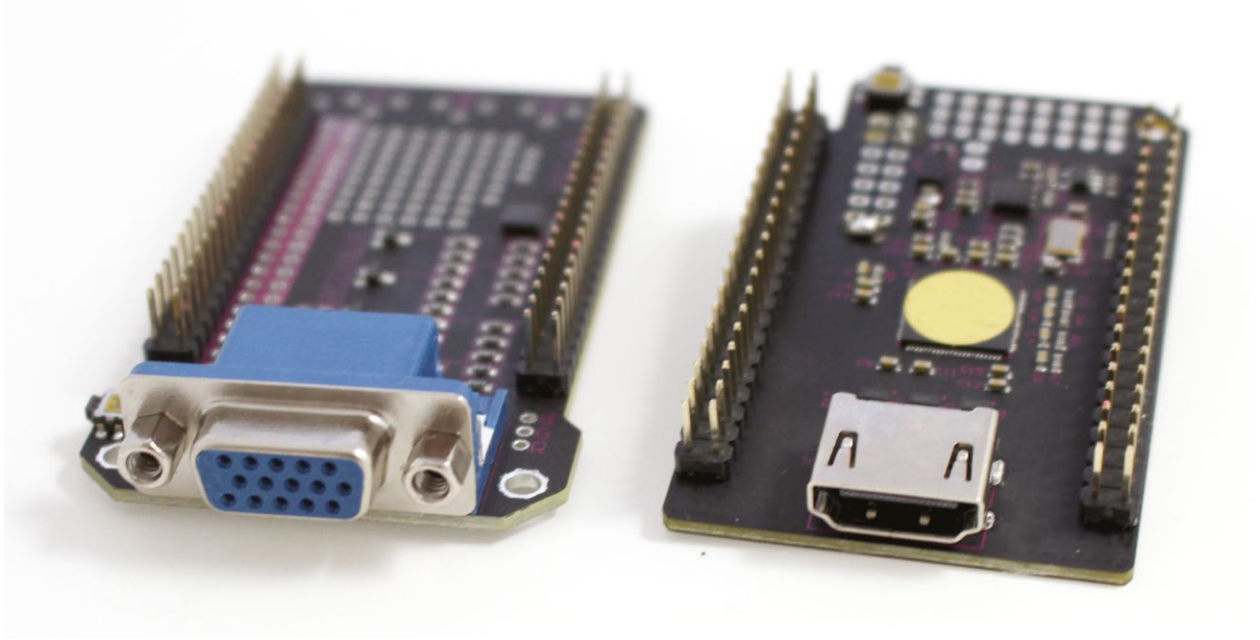
Sous GNU/Linux, comme sur Mac, il n'y a pas de pilote à installer, car le système reconnaît le périphérique. Sous Linux, il faudra cependant que l'utilisateur courant dispose des permissions

adéquates pour accéder au matériel. Il vous faudra éditer un nouveau fichier, que nous appellerons **99-allwinner.rules**, placé dans **/etc/udev/rules.d**. Ce fichier va ajouter une règle udev qui ajustera les droits au moment de la connexion et de la détection du matériel.

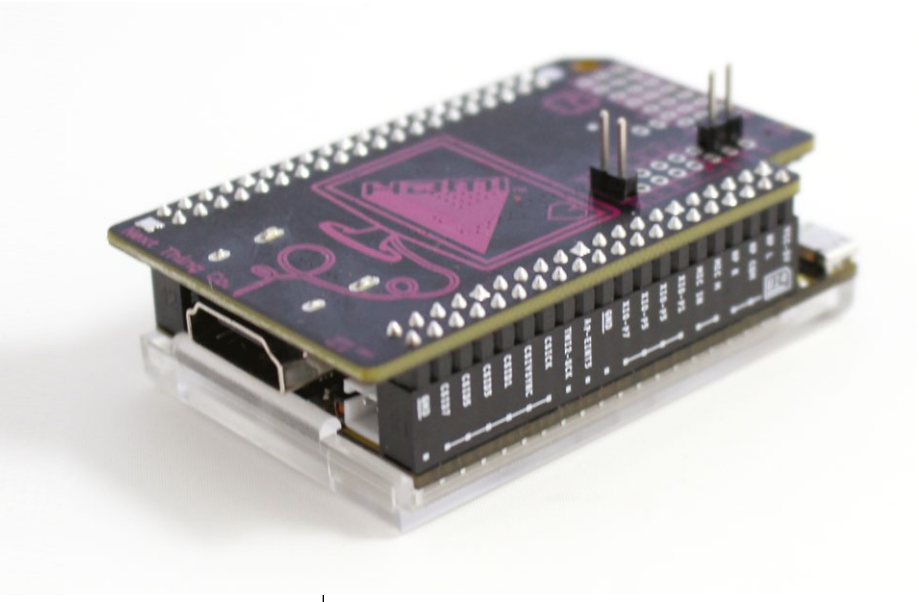
```
# Permissions pour CHIP
SUBSYSTEM=="usb", ATTRS{idVendor}=="1f3a", ATTRS{idProduct}=="efe8",\
   GROUP="plugdev", MODE="0660" SYMLINK+="usb-chip"
SUBSYSTEM=="usb", ATTRS{idVendor}=="18d1", ATTRS{idProduct}=="1010",\
   GROUP="plugdev", MODE="0660" SYMLINK+="usb-chip-fastboot"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1f3a", ATTRS{idProduct}=="1010",\
   GROUP="plugdev", MODE="0660" SYMLINK+="usb-chip-fastboot"
SUBSYSTEM=="usb", ATTRS{idVendor}=="067b", ATTRS{idProduct}=="2303",\
   GROUP="plugdev", MODE="0660" SYMLINK+="usb-serial-adapter"
```

Ceci aura pour effet de donner les bonnes permissions à tous les utilisateurs du groupe **plugdev** dont l'utilisateur par défaut fait généralement partie. Il faudra ensuite rafraîchir la configuration du service udev avec **sudo udevadm control --reload-rules**.

Ceci fait, quelle que soit la plateforme, l'étape suivante consistera à connecter un câble entre l'une des broches GND (masse) et celle marquée FEL, puis à brancher le CHIP en USB à un PC ou un Mac. Prenez ensuite votre navigateur Google Chrome et pointez-le sur **http://flash.getchip.com**. Là, une page vous proposera de choisir le système que vous désirez enregistrer dans la flash du CHIP et il ne vous restera plus qu'à suivre les indications à l'écran. La procédure, lorsqu'elle fonctionne, est relativement longue, mais vous obtiendrez à terme un CHIP avec un système à jour capable ensuite de prendre en charge les DIP et donc l'éventuelle sortie HDMI.



Les DIP sont des cartes additionnelles permettant d'ajouter des fonctionnalités matérielles au CHIP. Ici à gauche le DIP VGA et à droite le DIP HDMI. Ces deux ajouts constituent les seuls moyens pour connecter le CHIP à un moniteur PC.



Le DIP HDMI fournit une solution pour faire du CHIP une station de travail miniature, mais il faudra ajouter un hub USB afin de pouvoir utiliser un clavier et une souris. Un certain nombre de broches sont déportées sur le DIP, mais l'ensemble des signaux ne sera plus présent/utilisable. À l'heure actuelle, seul le signal vidéo est présent en HDMI, pour l'audio, il faudra attendre ou utiliser la sortie jack 3,5mm.

5. CONFIGURATION ET FRANCISATION

Quel que soit le système utilisé, configurer son CHIP pour le plier à notre identité toute française n'est pas très différent de ce qu'on peut faire avec une carte Raspberry Pi. Avant toutes choses, on obtiendra une ligne de commandes (console série, terminal dans l'interface graphique ou via Wifi/SSH). On pourra alors utiliser, tant bien que mal, le clavier pour saisir les commandes :

```
$ sudo apt-get install locales $ sudo dpkg-reconfigure locales
```

Ceci aura pour effet d'installer les éléments propres à la configuration des *locales* ou « paramètres régionaux » en bon français. La dernière commande vous présente une liste où vous choisirez **`fr_FR ISO-8859-1`**, **`fr_FR.UTF-8 UTF-8`** et **`fr_FR@euro ISO-8859-15`**, pour ensuite sur l'écran suivant, définir **`fr_FR.UTF-8`** comme *locale* par défaut. Le fait d'installer les deux autres les rend simplement disponibles en cas de connexion SSH depuis une machine les utilisant (une ancienne installation GNU/Linux par exemple).

Ces manipulations vous permettront dans un premier temps d'obtenir les messages et des traductions françaises pour de nombreux logiciels et commandes, mais ce n'est pas tout. Il vous faudra ensuite configurer votre clavier USB (ne le faites pas pour un pocketCHIP), avec **`sudo dpkg-reconfigure console-data`**. Dans la liste, sélectionnez ***Choisir un codage clavier pour votre architecture*** (notez que ça parle français à présent). Enfin, vous pourrez enchaîner sur **`sudo dpkg-reconfigure keyboard-configuration`** et naviguer dans les écrans pour sélectionner le type de clavier que vous souhaitez utiliser.

De là on peut ensuite se plier d'un **`sudo apt-get update`** et d'un **`sudo apt-get dist-upgrade`** pour mettre l'ensemble du système à jour. Ce après quoi j'enchaîne généralement avec un **`sudo apt-get install vim bash-completion screen mc exuberant-ctags`** et un **`sudo apt-get purge vim-tiny`** avant de copier, depuis mon PC, mon arborescence **`.vim*`** (configuration Vim) et les fichiers **`.inputrc`** (recherche dans l'historique avec les flèches haut/bas) et **`.screenrc`**. Je dispose alors d'une ligne de commandes exactement comme je l'aime, un véritable éditeur de texte/code/configuration, un démultiplexeur de terminaux et un gestionnaire de fichiers en mode texte.

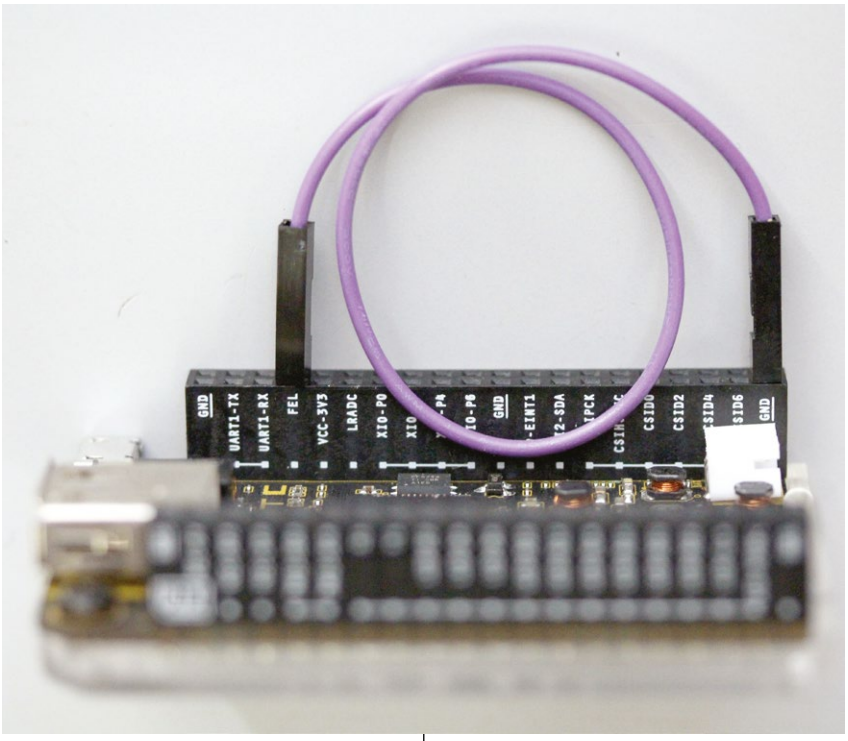
Quelle que soit la sortie vidéo utilisée (HDMI, VGA ou composite), le système Debian GNU/Linux propose une interface graphique reposant sur XFCE. Différentes applications permettent alors de l'utiliser comme un ordinateur standard, dans les limites imposées par les 512 Mo de mémoire vive.

CONCLUSION, AVIS ET RÉPONSE

Je vais être direct : le CHIP ne va pas remplacer les Raspberry Pi, ni même les concurrencer.

Ce n'est pas une question de qualité ou de technicité, ni même de puissance ou de fonctionnalités. Les deux plateformes n'adressent simplement pas les mêmes utilisateurs. Pour tout dire, je ne sais pas exactement qui sont les utilisateurs visés par *Next Thing Co.*, car derrière un aspect accessible et beaucoup d'efforts sincères pour satisfaire les amateurs et utilisateurs de Pi, le CHIP ressemble davantage à une plateforme pour l'embarqué qu'à un nano-ordinateur mono-carte.

Nous avons là un système avec de la flash intégrée, un bootloader U-Boot absolument merveilleux et un support noyau de grande qualité (d'ailleurs développé par les français de Free Electrons). L'absence de sortie VGA ou HDMI est en parfaite cohérence vis-à-vis d'une telle approche hyper technique. Les différentes interfaces proposées, l'intégration du contrôleur de charge LiPO, l'USB OTG dès la première version, tout comme le support des *overlay* du *device tree*,

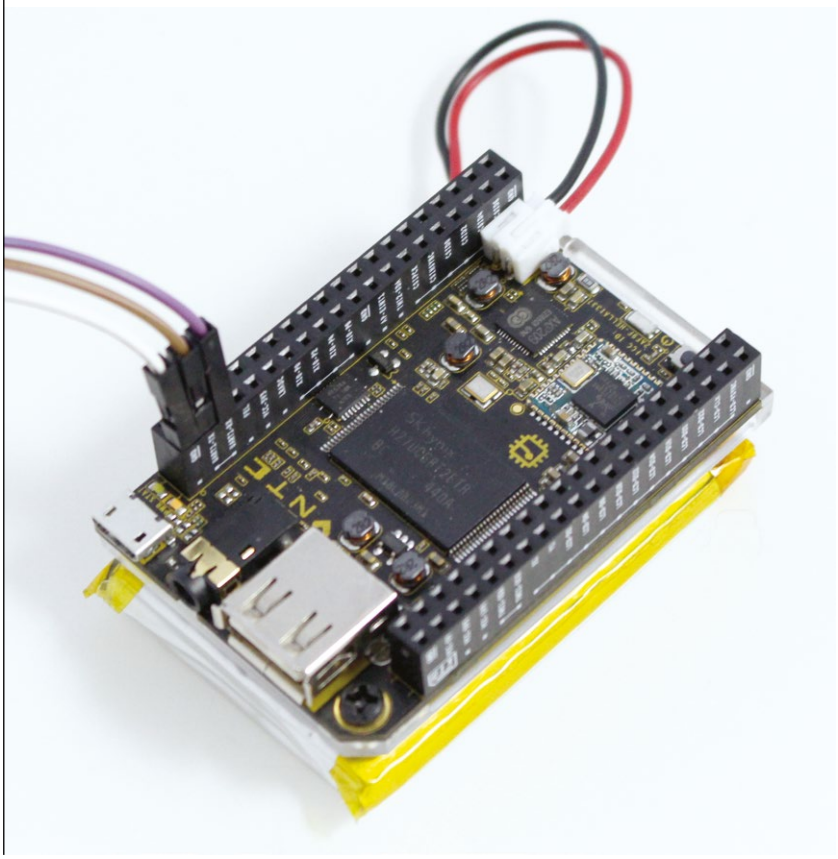


Le CHIP ne dispose pas d'un emplacement SD ou microSD. Le système trouve place dans la mémoire flash intégrée et pourra être changé en démarrant le CHIP en mode FEL via la mise à la masse de la broche éponyme. Il faudra ensuite utiliser la connexion USB via un PC ou un Mac pour charger un nouveau système grâce à une application installée dans Google Chrome (ou via les outils du SDK).

tout semble décrire un matériel destiné aux professionnels ou aux amateurs éclairés souhaitant une approche plus industrielle.

Mais en parallèle à ces caractéristiques, nous avons le PocketCHIP, les DIP, PICO-8 ou encore le système de mise à jour via navigateur web. Autant d'éléments qui vont clairement dans le sens de l'utilisateur hobbyiste.

J'apprécie grandement le CHIP lui-même. Compact, fonctionnant sur accu, intégrant l'USB OTG... c'est une plateforme de développement très intéressante et économique. Les DIP et le PocketCHIP, en revanche me laissent totalement indifférent, même s'il a



Après différents essais, tests, et expérimentations, voici ce qui m'apparaît être la seule configuration réellement intéressante du CHIP : un système embarqué autonome, contrôlé via une console série (ou SSH), susceptible d'être utilisé comme excellente base pour un projet de développement d'un objet connecté. Nous sommes ici dans le registre du système embarqué ou de la plateforme de développement, davantage que dans celui occupé par la famille Raspberry Pi.

été amusant de les tester, ce ne sont à mes yeux que de simples gadgets qui se retrouveront dans un tiroir assez rapidement après quelques essais avec des codes « maison ».

Si l'on s'en tient à l'offre elle-même, on constate que le CHIP seul n'est alors en rien concurrent à la Raspberry Pi, bien installée et satisfaisant pleinement les besoins de ses utilisateurs. Beaucoup trop de limitations sont présentes dans le cas du CHIP. Avoir recours à un DIP pour l'affichage sur un écran moderne est très restrictif. Le caractère indispensable d'un hub USB pour une utilisation desktop l'est tout autant. Dans ce cadre d'utilisation, l'absence d'un support de stockage amovible rend difficiles les modifications du système ou simplement sa réinitialisation. L'outil de flashage apporte une solution, mais elle n'est en rien comparable avec la simplicité offerte par le fait de mettre une microSD dans un lecteur et d'écraser son contenu. Et enfin, le concept du pocketCHIP bien que très amusant se solde finalement en une seule et unique frustration se résumant à deux valeurs : 480 et 272.

Voici donc ma conclusion, je reconnais ne pas être tendre avec le matériel mais, non, le n'ième « Raspberry Pi Killer » supposé n'en est pas un et se résume à une excellente plateforme de développement et d'expérimentation pour Linux embarqué (le vrai, avec U-Boot, sous-système MTD, UBIFS, USB OTG, etc.). Ainsi, si vous souhaitez plonger dans les méandres du système, développer, personnaliser la plateforme ou créer un objet connecté... ce sera un excellent choix. Si en revanche vous voulez voir dans le CHIP un simple ordinateur miniature pour des projets accessibles, une Raspberry Pi est un bien meilleur choix. **DB**

UN AFFICHAGE SUR PAPIER ÉLECTRONIQUE POUR VOTRE ARDUINO

Denis Bodor

Les solutions pour afficher du texte et des graphismes avec une carte Arduino sont légion : afficheurs LCD alphanumériques, matrices de leds, écrans TFT, modules Oled, etc. Le papier électronique utilisé majoritairement par les liseuses comme l'Amazon Kindle ou la Kobo de la Fnac possède l'aspect d'un papier plastifié tout en permettant un affichage dynamique. Il existe quelques modules d'affichage de ce type pouvant être utilisés pour vos projets, le plus souvent de taille réduite. Celui testé et utilisé ici est un modèle conséquent avec une diagonale de 115 mm (4,3"), de quoi satisfaire ses envies d'originalité...

Le principal intérêt du papier électronique est son aspect : il ressemble à une impression laser ou jet d'encre sur papier. Le support est donc totalement réflectif par opposition aux écrans LCD, TFT ou Oled émettant de la lumière. L'affichage est donc plus « naturel » et moins fatiguant à la lecture.
Un autre avantage est l'aspect statique de l'affichage. La technologie utilisée consiste à changer l'état des pixels, dans un sens ou dans l'autre, et c'est uniquement ce changement qui nécessite de l'énergie. Lorsqu'il n'est plus alimenté, l'écran conserve donc l'état qui lui a été donné précédemment, contrairement à d'autres systèmes où une coupure d'alimentation est synonyme d'effacement. L'affichage sur papier électronique est donc très économe en énergie, un autre avantage important pour les liseuses.

1. LA TECHNOLOGIE DU PAPIER ÉLECTRONIQUE

La technologie du papier électronique, ou *e-paper* en anglais, n'est pas récente. Les premières expérimentations dans ce domaine remontent aux années 70 puis le concept a été amélioré dans les années 90, gagnant en résolution et en fonctionnalités. Ce n'est que début 2000 que les périphériques de lecture électronique utilisant cette technologie font leur apparition et il faudra encore attendre une bonne dizaine d'années avant de voir leur popularité croître au point de devenir véritablement un objet du quotidien.

Le principe de fonctionnement de la plupart de ces écrans repose sur l'électrophorèse, consistant à séparer des particules en fonction de leur charge électrique.
La base d'un tel système d'affichage est un ensemble de capsules contenant des particules sphériques d'un diamètre de quelques microns, en suspension dans un liquide transparent. Les billes en suspension sont chargées négativement ou positivement et colorées en blanc ou en noir. Les capsules sont prises en sandwich dans un substrat couvert d'électrodes transparentes. En appliquant une charge électrostatique de part et d'autre du support, les particules ou billes changent de position laissant alors apparaître une couleur ou l'autre selon la polarité.

Une fois le déplacement terminé, il n'est pas nécessaire de maintenir la charge électrostatique. En l'absence de celle-ci, les particules conservent tout simplement leurs positions. En théorie, ce mécanisme fonctionne à merveille, dans la réalité il n'en va pas tout à fait de même. La densité de capsules dans le substrat, la proximité des unes par rapport aux autres, la présence de charges électrostatiques rémanentes et l'inertie mécanique des billes font qu'un effet visuel gênant apparaît : des images fantômes.

En effet, une image précédemment affichée en noir sur fond blanc, par exemple, redessinée en blanc, laissera entrevoir le tracé précédent. De la même manière, un tracé en noir puis le basculement de tous les pixels de la zone dans cette même couleur fera, là aussi, apparaître une trace fantôme. Cet état de fait rend très difficile un rafraîchissement partiel de l'écran et la technique utilisée dans la plupart des cas consiste alors à non seulement redessiner entièrement l'écran à chaque changement, mais également à basculer tout l'écran en noir puis en blanc avant un nouveau tracé, parfois même plusieurs fois de suite. Ceci force les billes à reprendre une position uniforme avant tout nouvel affichage, un peu comme secouer un récipient pour faire descendre les petits éléments et remonter les plus gros.

En fonction de la qualité et la technologie utilisée par le papier électronique, ces opérations de rafraîchissement peuvent être plus ou moins optimisées en étant

Le module d'affichage se présente sous la forme d'un circuit imprimé de 75 x 118 mm où un écran de 4,3 pouces d'une résolution de 800×600 pixels occupe la majeure partie de la surface. Largement suffisant pour envisager de nombreuses applications...

faites zone par zone, mais un rafraîchissement complet s'avère tout de même nécessaire régulièrement. Vous comprendrez alors qu'il n'est pas vraiment possible d'utiliser ce type d'affichage de la même façon qu'un écran TFT ou Oled et que l'affichage d'animations est tout simplement hors de portée.

2. ÉCRAN WAVESHARE 4,3 POUCES

La technologie *e-paper* parfois appelée EPD (*E-Paper Display*) bien que très courante pour les liseuses et de plus en plus pour les systèmes d'affichage de prix en magasins, est encore relativement rare lorsqu'il s'agit d'électronique hobbyiste. Quelques modules existent proposant des écrans de plus ou moins grande taille, mais leur coût reste généralement important face à des équivalences en LCD TFT.

Le modèle choisi et testé ici est construit et vendu par la société WaveShare sous la désignation « *4.3inch e-Paper* », mais se retrouve sur les sites d'enchères et les boutiques en ligne sous de nombreux autres intitulés. L'objet testé était vendu par *eckstein_komponente* sur eBay sous le nom « *4.3 inch 800 x 600 e-Paper LCD serial interface electronic paper display WS92019* » au prix de 68€. J'ai opté pour cette offre, même si le mot « LCD » dans la désignation était peu encourageant, car le vendeur se trouvant en Allemagne, j'étais assuré d'obtenir le produit assez rapidement. On peut cependant trouver le même objet, via des vendeurs de Shenzhen pour quelques 50€ port offert si l'on accepte de patienter quelques semaines pour la livraison.

Le modèle est facilement reconnaissable avec un écran de

taille conséquente (4,3" ou 115mm de diagonale), son circuit imprimé bleu, une sérigraphie « 4.3inch e-Paper » sur la tranche et le logo Waveshare à l'arrière. Il est peu probable que vous puissiez le confondre avec un autre modèle tant ces produits sont peu courants...

Ce module d'affichage est bien plus qu'un simple écran. Il intègre en effet un microcontrôleur STM32F103, 1 Mo de mémoire (SRAM), 128 Mo de mémoire de stockage (NAND), un support microSD et une interface série. Piloter un écran e-paper n'est pas chose facile et le microcontrôleur STM32 se charge donc de le faire à votre place, exactement comme un circuit intégré HD4480 ou compatible le fait pour un afficheur LCD alphanumérique. Mais ce n'est pas tout, ce microcontrôleur prend également en charge des primitives graphiques permettant d'activer des pixels, tracer des lignes, dessiner des formes pleines ou vides, afficher des images ou écrire du texte.

Il est également possible d'utiliser des fichiers présents sur la carte microSD (appelée TF dans la documentation) ou dans la mémoire de stockage de 128 Mo ainsi que de procéder à des copies de la carte vers la mémoire. Les fichiers en question peuvent être des images au format BMP ou des polices de caractères dans un format qui semble non standard.

C'est donc davantage un système complet qu'un simple écran, et celui-ci communique avec un protocole particulier via une connexion série en 115200 bps. La connexion avec une carte Arduino est relativement simple même si le câble fourni propose des connecteurs femelles alors que les cartes Arduino également (du moins les versions officielles). Le brochage est le suivant :

- 1 RST (bleu) : la ligne permettant de réinitialiser le module qui est généralement laissée non connectée ;
- 2 WAKE_UP (jaune) : le signal permettant de réveiller le module si celui-ci est passé en veille de façon logicielle pour réduire la consommation de courant ;
- 3 DIN (vert) : réception des données série, cette ligne doit être connectée à la broche TX de la carte Arduino ;
- 4 DOUT (blanc) : envoi des données série transmises par le module, à connecter à la broche RX de l'Arduino ;
- 5 GND (noir) : masse ;
- 6 VCC (rouge) : tension d'alimentation entre 3,3V et 5V. Cette tension sera également celle du niveau logique haut pour les autres broches. Le module est donc utilisable aussi bien avec les Arduino en 5V comme la UNO et en 3,3V comme la Due, mais également avec une Raspberry Pi (il existe un module Python développé par un programmeur indépendant de WaveShare : **https://github.com/yy502/ePaperDisplay**).

3. UTILISATION DE L'ÉCRAN

La connexion Arduino/module sera la suivante : VCC sur 5V, GND avec GND, DIN sur TX, DOUT sur RX, WAKE_UP sur 2 et RST sur 3. Bien que la documentation précise que la broche RST soit plus ou moins inutile, celle-ci est effectivement configurée et utilisée dans la bibliothèque. Ce qui nous amène au principal problème de ce produit : une bibliothèque passablement mal écrite intégrant un certain nombre d'éléments « en dur » dans le code.

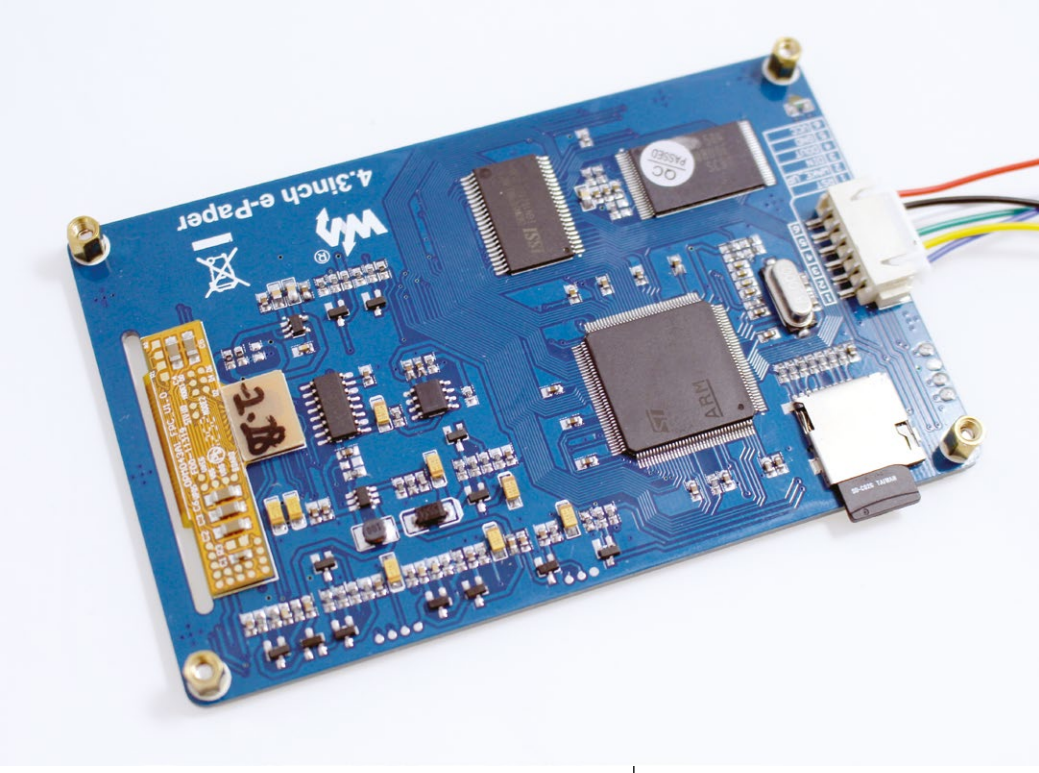
Cette bibliothèque est disponible directement sur le wiki de WaveShare sous la forme d'une archive 7z : **http://www.waveshare.com/wiki/File:4.3inch-e-Paper-Code.7z**. Ce fichier contient des codes et fichiers pour les plateformes ST Nucleo F103RB (mbed), STM32 Open103Z de WaveShare et Arduino, ainsi que ce qui semble être le binaire du firmware pour le STM32 du module (sans les sources). Vous trouverez ainsi, une fois l'archive décompressée, un répertoire **4_3inch-ePaper-demo/Arduino-epd** contenant un croquis **Arduino-epd.ino** ainsi qu'un sous-répertoire **epd** comprenant les deux fichiers qui forment la bibliothèque (**epd.cpp** et **epd.h**).

Il vous faudra alors copier le répertoire **epd** et son contenu dans le répertoire **librairies** de votre carnet de croquis pour ensuite, éventuellement ouvrir le fichier **Arduino-epd.ino**, le compiler et le charger dans votre Arduino. Si tout se passe correctement, le module devrait alors afficher une série de démonstrations graphiques avant de terminer par quelques lignes de texte et passer en veille.

La mise en œuvre d'un premier croquis « maison » est relativement simple au regard de ce que fournit la bibliothèque :

```
#include <epd.h>

void setup() {
  // initialisation du module
  epd_init();
  epd_wakeup();

  // utilisation de la mémoire intégrée
  epd_set_memory(MEM_NAND);

  // on dessine en noir
  epd_set_color(BLACK, WHITE);

  // effacement de l'écran
  epd_clear();

  // choix de la police
  epd_set_en_font(ASCII64);

  // écriture de texte
  epd_disp_string("Hackable Magazine #14", 0, 80);

  // mise à jour de l'affichage
  epd_udpate();

  // mise en veille
  epd_enter_stopmode();
}

void loop() {
}
```

Le développeur ayant une certaine expérience remarquera sans peine quelques petites choses dans l'exemple officiel fourni, indiquant clairement que le code n'est pas initialement développé pour Arduino, mais adapté d'un code générique en C. L'utilisation de **void setup(void)** en lieu et place de **void setup()**, par exemple, est typique du C, car ces deux

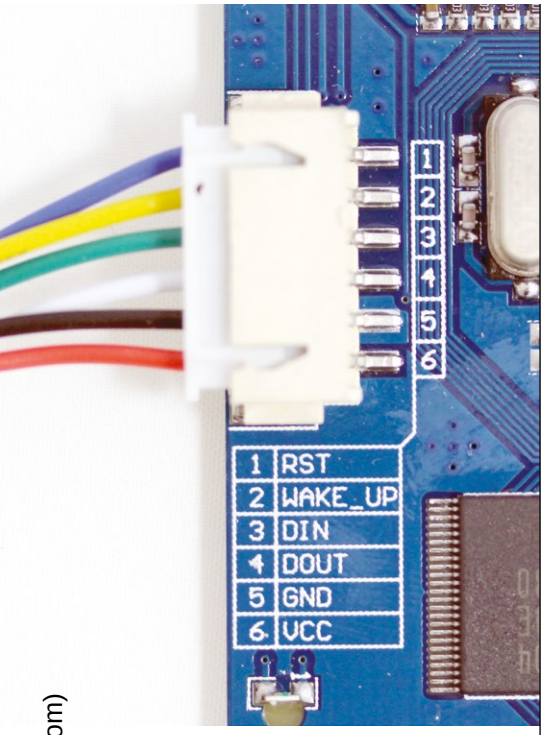


L'électronique de contrôle se trouve à l'arrière du module où l'écran à proprement parler est connecté via un flatflex. La qualité de l'ensemble est bien supérieure à celle d'autres modules d'affichage (LCD, TFT, etc.) généralement utilisés, il s'agit clairement d'un produit initialement destiné à une utilisation industrielle.

syntaxes n'ont pas le même sens alors qu'en C++, le langage sous-jacent de l'environnement Arduino (compilateur `avr-g++`), elles sont strictement identiques. De plus, les fonctions proposées sont... des fonctions et non des méthodes (comme `Serial.println()`), il ne s'agit pas de programmation orientée objet comme ce que l'on trouve généralement pour une bibliothèque Arduino digne de ce nom.

Ce n'est pas tout, en regardant du côté du fichier `epd.cpp`, on remarque également un certain nombre de choses gênantes :

- un commentaire précise que l'environnement utilisé est RealView MDK-ARM, une solution propriétaire de la société Keil appartenant à ARM et non un environnement dédié à Arduino ou aux microcontrôleurs Atmel ;
- le fichier intègre une mention « Copyright 2005-2014, WaveShare - All Rights Reserved » sans préciser de licence, il n'est donc pas légal de redistribuer ce code ;
- les broches 2 et 3 correspondent à `wake_up` et `reset`, il n'est pas possible d'utiliser d'autres broches sans modifier la bibliothèque ;
- la bibliothèque utilise par défaut `Serial` correspondant au seul port série sur une UNO et au premier sur Mega 2560 alors que ce port est déjà utilisé pour le moniteur série et pour la programmation de la carte. Il est ainsi nécessaire de déconnecter le câble blanc du module avant de programmer la carte pour éviter d'envoyer des données non valides à l'écran. En cas d'utilisation d'un autre port ou de la bibliothèque *SoftwareSerial*, il faut modifier le code de la bibliothèque fournie.
- La bibliothèque déclare un tableau de 512 octets (`unsigned char`) alors que Arduino UNO, par exemple, n'en possède que 2Ko soit 1/4 de la SRAM disponible, pour composer et stocker les données à envoyer au module. Mais la seule fonction faisant usage d'un tel volume de données est `epd_disp_string()`, qui ne vérifie d'ailleurs pas la taille de la chaîne de caractères qui lui est passée en argument.
- La réponse du module, généralement `"OK"` n'est jamais vérifiée, le port série n'est utilisé qu'en écriture. Techniquement, la broche DOUT sur RX est totalement inutile.



Interface et alimentation sont regroupées sur un connecteur à 6 broches et le module est livré avec le câble adapté, mais qui malheureusement, à l'autre bout, possède des connecteurs femelles : parfait pour une Raspberry Pi ou une carte Arduino Nano, mais pénible avec une UNO, ou une Mega2560...

- De façon générale, l'ensemble de la bibliothèque n'est pas optimisé et n'utilise aucune technique permettant d'économiser de la mémoire (comme le stockage des chaînes en flash).

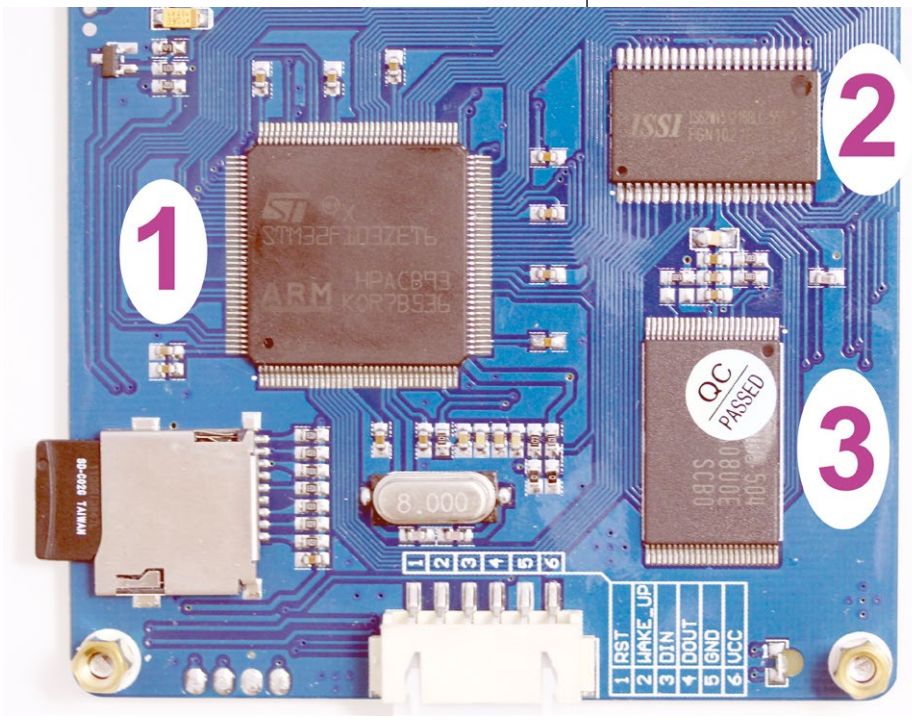
Comme vous pouvez le voir, la bibliothèque fournie par WaveShare n'est rien autre qu'une modeste démonstration du fonctionnement du protocole à utiliser avec le module et pour ainsi dire, un exemple assez brouillon et décevant des fonctionnalités du produit.

Quoi qu'il en soit, voici un résumé des fonctions disponibles, puisque la documentation officielle ne couvre pas le sujet :

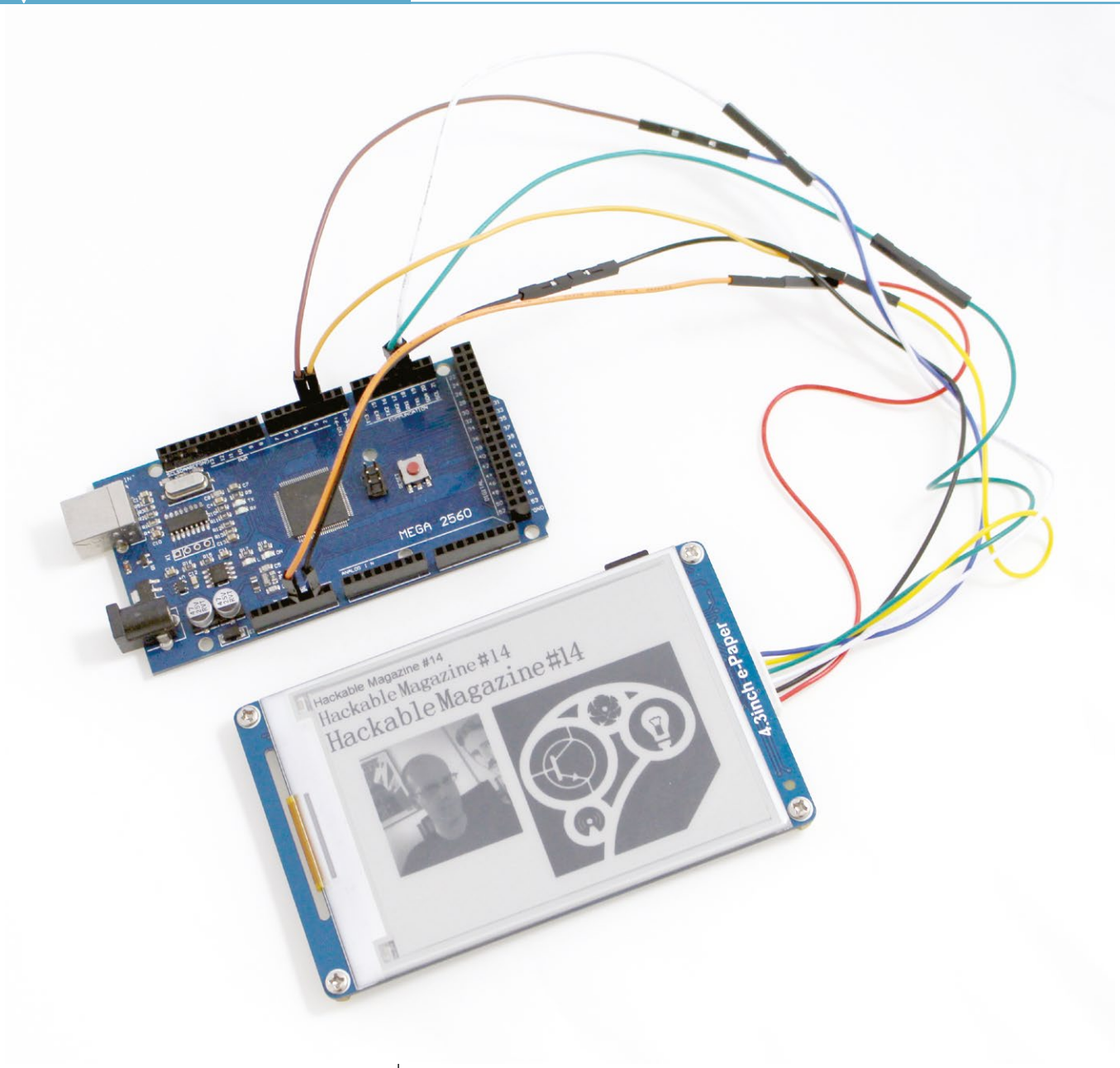
- **`epd_init()`** : initialise le module, à appeler avant toutes communications,
- **`epd_reset()`** : réinitialise le module via un changement d'état sur la broche RST (3),
- **`epd_wakeup()`** : réveille le module s'il est passé en veille, via la broche WAKE_UP (2),
- **`epd_enter_stopmode(void)`** : passe le module en veille, celui-ci ne répondra à aucune commande sans d'abord utiliser **`epd_wakeup()`**,
- **`epd_handshake()`** : vérifie simplement la communication, le module doit répondre **`"OK"`**, mais la bibliothèque ne s'en assure pas,
- **`epd_set_memory(mode)`** : détermine si le module doit lire les données depuis la microSD (**`MEM_TF`**) ou la mémoire interne (**`MEM_NAND`**),
- **`epd_screen_rotation(mode)`** : définit l'orientation de l'affichage, soit normal (**`EPD_NORMAL`**), soit tourné à 180° (**`EPD_INVERSION`**). La position 0:0 se trouve en haut à gauche de l'écran par défaut (affichage normal) et en bas à droite en inversé.
- **`epd_clear()`** : efface la mémoire de l'écran (ceci n'a aucun impact sur l'affichage tant que **`epd_udpate()`** n'est pas utilisé). La couleur utilisée pour l'effacement est celle d'arrière-plan (**`epd_set_color(couleur, couleur_fond)`**),
- **`epd_udpate()`** : répercute les modifications de l'affichage sur l'écran. La logique ici est de composer un écran avec les primitives graphiques puis de « pousser » ces données sur l'affichage. Avant appel à cette fonction, l'image est uniquement présente dans la mémoire volatile de l'afficheur (*framebuffer*).
- **`epd_load_pic()`** : permet de charger tous les fichiers graphiques présents sur la microSD dans la mémoire NAND interne (l'espace dédié est effacé puis les fichiers sont copiés), l'espace disponible pour ces données est de 80 Mo.
- **`epd_load_font()`** : permet de copier les fichiers de polices présents sur la microSD dans la mémoire interne (idem, effacement puis copie), l'espace disponible est de 48 Mo. Cette fonction ainsi que **`epd_load_pic()`**

ne vérifient pas la présence de fichiers sur la microSD, si la carte est vide, la zone correspondante en mémoire interne est simplement effacée.

- **`epd_set_color(couleur, couleur_fond)`** : définit la couleur d'avant-plan et d'arrière-plan pour les tracés. Les couleurs utilisables sont **`WHITE`** (blanc), **`GRAY`** (gris), **`DARK_GRAY`** (gris foncé) et **`BLACK`** (noir).
- **`epd_set_en_font(police)`** : définit la police de caractères à utiliser pour l'affichage de textes anglais entre **`ASCII32`**, **`ASCII48`** et **`ASCII64`**. La valeur numérique dans le nom de la macro correspond à la hauteur du texte en pixels (notez que les caractères accentués ne sont pas disponibles).
- **`epd_set_ch_font(police)`** : définit la police de caractères pour l'affichage de texte en chinois simplifié avec le jeu de caractères GBK, entre **`GBK32`**, **`GBK48`** et **`GBK64`**. Notez que les polices ASCII sont toujours disponibles, mais que les polices chinoises doivent se trouver en mémoire interne ou sur microSD pour pouvoir être utilisées. Les deux jeux de polices sont disponibles, avec des BMP de démonstration, sur le wiki WaveShare.
- **`epd_draw_pixel(x0, y0)`** : active un pixel avec la couleur de premier plan aux coordonnées x0:y0,
- **`epd_draw_line(x0, y0, x1, y1)`** : trace une ligne entre le point x0:y0 et x1:y1,
- **`epd_fill_rect(x0, y0, x1, y1)`** : dessine un rectangle rempli de la position x0:y0 (coin supérieur gauche) et x1:y1 (coin inférieur droit),
- **`epd_draw_circle(x0, y0, r)`** : trace un cercle avec pour centre x0:y0 et un rayon *r*,
- **`epd_fill_circle(x0, y0, r)`** : trace un disque (plein) avec pour centre x0:y0 et un rayon *r*,
- **`epd_draw_triangle(x0, y0, x1, y1, x2, y2)`** : trace un triangle, avec ses sommets placés aux points x0:y0, x1:y1 et x2:y2,



Le pilotage de l'écran repose sur l'utilisation de trois éléments : un microcontrôleur STM32 (1) chargé de la gestion et l'interprétation des commandes, de la mémoire SRAM (2) pour stocker l'image présentée à l'écran (framebuffer) et 128 Mo de mémoire flash NAND (3) pour les fichiers graphiques BMP (80 Mo) et les polices de caractères (48 Mo).



La connexion du module à une carte Arduino ne présente matériellement pas de difficulté, mais il faudra modifier la bibliothèque fournie par le constructeur si l'on souhaite utiliser un autre port série que celui proposé par défaut, comme ici avec Serial1 sur un clone d'Arduino Mega 2560.

- **`epd_fill_triangle(x0, y0, x1, y1, x2, y2)`** : trace un triangle plein, avec ses sommets placés aux points x0:y0, x1:y1 et x2:y2,
- **`epd_disp_char(caractere, x0, y0)`** : affiche un caractère unique dans la police précédemment sélectionnée aux coordonnées x0:y0 (coin supérieur gauche),
- **`epd_disp_string(chaine, x0, y0)`** : affiche une chaîne de caractères à la position x0:y0 (coin supérieur gauche),
- **`epd_disp_bitmap(chaine, x0, y0)`** : affiche un dessin chargé depuis un fichier BMP en mémoire interne ou sur microSD (choisi par **`epd_set_memory(mode)`**) à la position x0:y0 (coin supérieur gauche).

Le protocole documenté liste d'autres commandes possibles, comme la définition de la vitesse de communication ou encore la lecture du type de mémoire sélectionnée. Certaines sont présentes dans le fichier `edp.cpp`, mais pas dans `epd.h`, et d'autres ne sont tout simplement pas prises en charge par la bibliothèque.

4. UN MOT À PROPOS DES IMAGES ET DES POLICES

Le module est en mesure de soulager la mémoire de votre carte Arduino en vous permettant de stocker les images à afficher sous la forme de fichiers graphiques au format BMP. Plusieurs solutions existent vous permettant de stocker ces fichiers. Vous pouvez soit les utiliser directement depuis la microSD en sélectionnant ce support avec `epd_set_memory(MEM_TF)`, soit les placer sur la microSD et utiliser la fonction `epd_load_pic()` pour les transférer de la microSD vers la mémoire interne. Attention cependant, cette opération n'ajoute pas des images, mais efface la mémoire interne dédiée de 80 Mo puis copie l'intégralité des fichiers BMP de la microSD. Si vous avez en mémoire interne des données qui ne sont pas sur la microSD, ces fichiers seront donc effacés.

Un certain nombre de restrictions sont également présentes :

- la microSD doit être formatée en FAT32 ;
- le nom des fichiers doit être composé de lettres majuscules ASCII uniquement ;
- l'extension des fichiers doit être `.BMP` ;
- le nom complet des fichiers ne peut pas faire plus de 10 caractères, `.BMP` inclus, ce qui ne vous laisse que 6 caractères à définir ;
- les images doivent utiliser une palette de couleurs indexée sur 1 bit (2 couleurs) ou 2 bits (4 couleurs).
- les images ne peuvent pas avoir une taille supérieure à celle de l'écran, 800 par 600 pixels.

Il est donc nécessaire d'utiliser un logiciel de retouche d'images pour convertir n'importe quel autre format et/ou nombre de couleurs. La documentation WaveShare recommande l'utilisation de MS Paint et de leur utilitaire uC GUI BitmapConvert. Personnellement, je n'ai aucune affinité avec un outil aussi simpliste que Paint et aucune confiance quant à l'exécution d'un programme dont l'origine n'est pas claire (et disponible sans les sources).

Ma solution consiste donc tout simplement à utiliser Gimp, logiciel libre disponible pour GNU/Linux, Windows et Mac OS X (il paraît qu'on dit macOS maintenant). Pour les images à convertir en deux couleurs (noir et blanc), la technique est relativement simple : ouvrir l'image, adapter ses dimensions avec ***Image*** et ***Echelle et taille de l'image*** puis convertir les couleurs via ***Image***, ***Mode***, ***Couleurs indexées***. Dans la fenêtre qui s'affiche, choisir ***Utiliser la palette noir & blanc (1-bit)*** et cliquer sur le bouton ***Convertir*** avant d'exporter le résultat au format BMP.

Par défaut, l'image sera radicalement transformée, tous les pixels ayant une certaine valeur limite passant en noir et les autres en blanc. Il sera cependant possible, dans le cas d'une photo, d'utiliser un tramage permettant d'avoir un rendu plus fidèle à l'original. Il suffit de choisir ***Floyd-Steinberg*** et Gimp fera de son mieux pour obtenir un résultat optimal.

L'écran e-paper peut cependant faire mieux que cela. Nous avons en effet non pas juste du noir et du blanc, mais deux teintes de gris intermédiaires à notre disposition. Nous pouvons donc convertir l'image avec chaque pixel encodé sur deux bits et non un seul. Pour cela, il est possible de préciser dans la fenêtre de conversion de mode l'option ***Générer une palette optimale*** en spécifiant 4 en guise de nombre de couleurs, et ce après avoir passé l'image originale en niveau de gris via le menu ***Couleurs*** et ***Désaturer***.

Une version plus avancée de cette technique consiste à créer une palette de couleurs spécifique en passant par le menu ***Fenêtres***, ***Fenêtres ancrables*** et ***Palette***. Dans la fenêtre qu'y s'affiche, vous présentant une liste de palettes existantes, cliquez sur l'icône représentant une feuille vierge en bas, pour créer une nouvelle palette. Là, précisez 4 dans ***Colonnes*** et ajouter 4 couleurs avec la même icône que précédemment. Il vous suffira alors de double-cliquer sur chaque couleur pour la définir en notation HTML : ***000000***, ***555555***, ***AAAAAA*** et ***FFFFFF***. Donnez ensuite un nom à votre nouvelle palette et enregistrez-la en cliquant sur l'icône représentant une petite disquette. Retournez ensuite dans l'outil de changement de mode en couleurs indexées et optez pour ***Utiliser une palette personnalisée***, sélectionnez votre palette dans la liste, décochez ***Enlever les couleurs non utilisées de la palette*** et procédez à la conversion avec ou sans tramage. L'image est convertie en 4 niveaux de gris et vous pouvez l'exporter en BMP.

Une fois le fichier placé sur la microSD puis éventuellement transféré en mémoire interne avec `epd_load_pic()`, ou utilisé sur place en précisant `epd_set_memory(MEM_TF)`, il ne vous restera plus qu'à l'afficher avec, par exemple, `epd_disp_bitmap("HLOGO.BMP", 350, 165)`. Vous pouvez placer les images où bon vous semble, même si celles-ci dépassent de l'écran.

Notez également que l'ordre d'utilisation des primitives graphiques est important, les différents éléments tracés et dessinés s'ajoutent les uns sur les autres et il n'y a pas de notion de transparence (le texte apparaîtra toujours avec un fond blanc, ou noir, de la hauteur de la police précisée). En utilisant judicieusement différents éléments graphiques ainsi, vous pourrez composer un écran complet constitué d'images, de formes et de textes.

CONCLUSION ET AVIS

« Frustrant », c'est le mot qui me vient instinctivement à l'esprit lorsque je tente de formaliser un avis sur ce module. Matériellement, l'objet est vraiment sympathique et de bonne facture, mais son potentiel n'est clairement pas exploité. Nous avons vu que le code fourni en guise de bibliothèque Arduino mériterait une réécriture complète même s'il semble évident qu'il ne s'agit que d'un simple exemple fonctionnel d'utilisation. Mais on sent réellement qu'il aurait été possible de faire bien davantage sur l'aspect logiciel au sens large du terme.

En effet, c'est le microcontrôleur STM32 intégré sur le module qui gère l'ensemble des fonctions disponibles. Il est tout à fait légitime de penser qu'avec un ARM Cortex-M3 cadencé à 72 Mhz, 512 Ko de flash et 64 Ko de mémoire, ce ne sont pas les ressources qui manquent pour étendre les fonctionnalités (le fichier `4_3inch_ePaper.hex` censé être le firmware du STM32, une fois converti en binaire avec `objcopy`, fait moins de 100 Ko) et propose des choses comme :

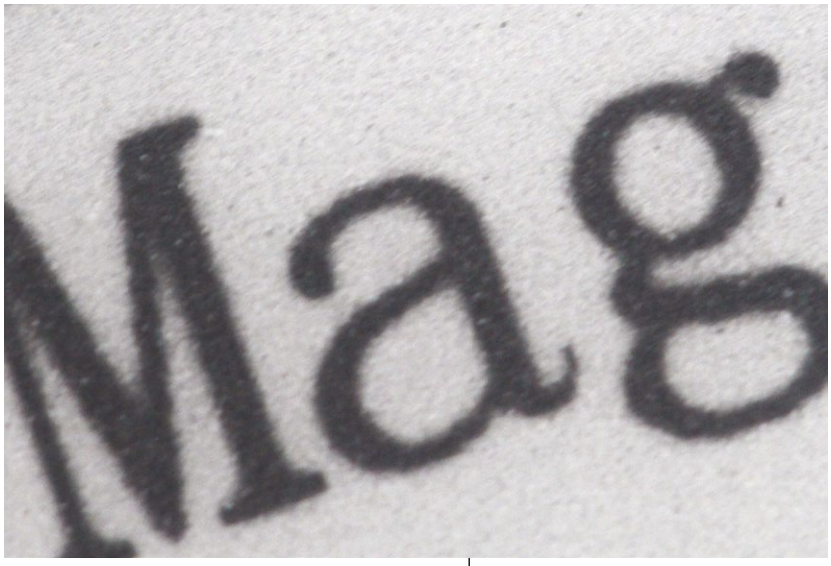
- Le *clipping*, consistant à n'afficher qu'une partie d'une image en précisant les coordonnées de la zone à utiliser. Ceci est une technique très intéressante permettant de réunir plusieurs éléments en un seul fichier graphique pour n'en utiliser que certaines zones. Idéal pour un jeu d'icônes par exemple.
- Un rafraîchissement partiel de l'écran, permettant d'avoir ainsi un affichage plus rapide, exactement comme pour les liseuses.
- Des primitives graphiques un peu plus évoluées que le modeste ensemble proposé, avec des choses comme des cercles et disques partiels (en précisant un angle d'ouverture), un choix de l'épaisseur de ligne lors des tracés ou encore des formes plus complexes comme des boîtes à coins arrondis.

Mais le problème principal reste le manque d'ouverture. Une carence en documentation

complète et en exemples de qualité est une chose récurrente avec ce genre de modules, mais ici de nombreux développeurs seraient sans doute prêts à étoffer très largement le firmware dans un effort communautaire si celui-ci était disponible sous la forme de code source. Mais ce firmware est mis à disposition sous une forme inutilisable en termes d'évolution ou de modification, chose qui va de paire avec l'absence de permission explicite ou de licence dans le code de démonstration fourni, mais également avec l'utilisation d'un format obscur de polices de caractères.

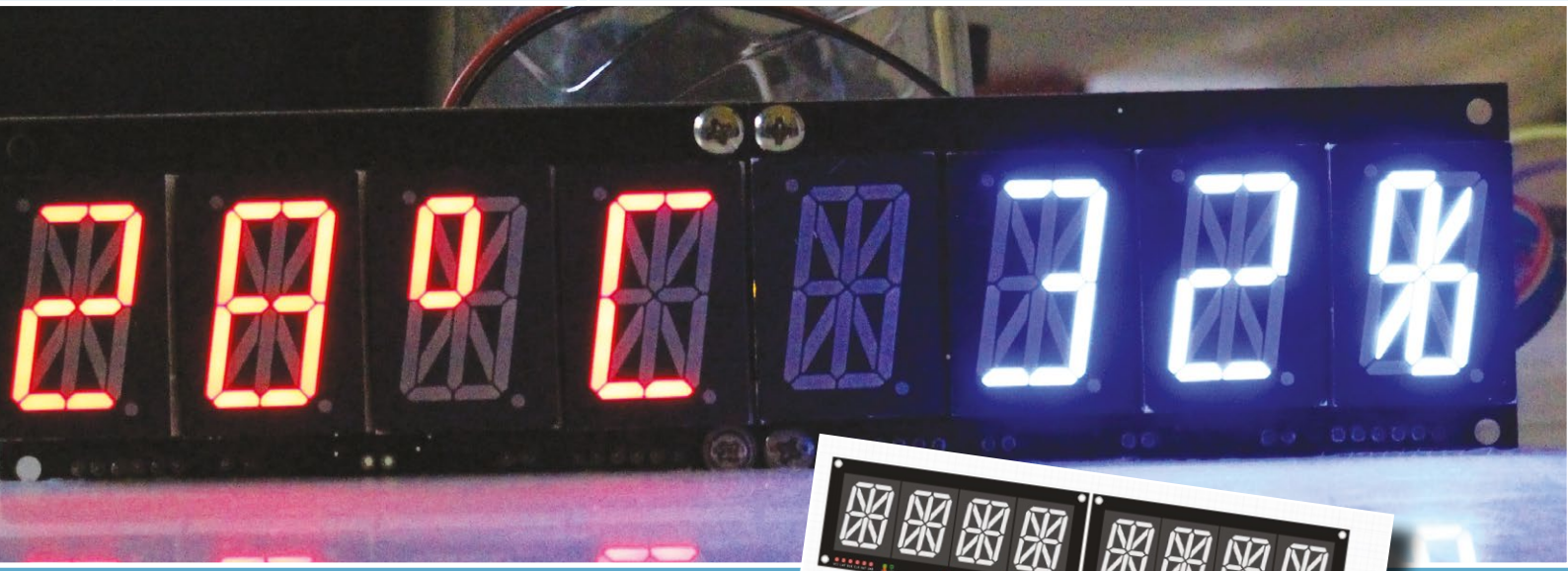
Les fichiers à l'extension `.FON` semblent bien constitués de pixels et on reconnaît des formes en tentant de convertir le format avec des outils comme ceux d'ImageMagick, mais l'absence de documentation rend les choses très difficiles puisqu'il s'agit de polices proportionnelles (un « i » occupe moins d'espace horizontal qu'un « w », cette information est forcément encodée dans le fichier pour chaque symbole). Trois tailles de polices de caractères est déjà très limitatif, mais bien moins que le fait de ne pas disposer de caractères accentués. Là encore, une certaine ouverture technique permettrait la création d'un jeu bien plus complet de polices, voire la création d'un outil de conversion utilisant un format standard comme TrueType.

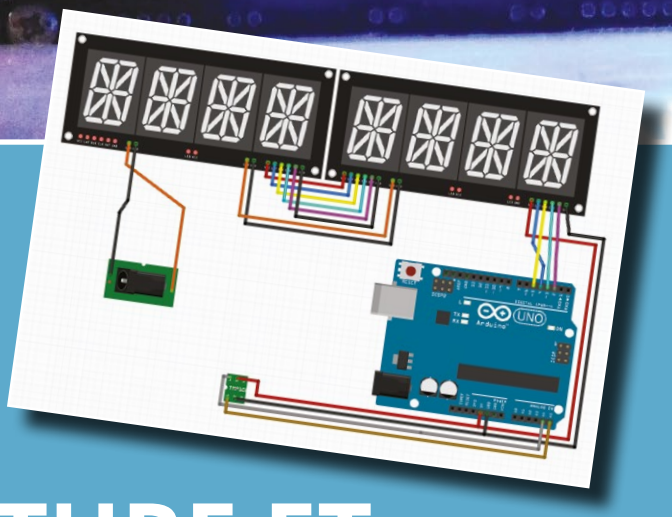
Certes le module permet de disposer d'un affichage peu conventionnel dans le domaine et il est possible d'user d'astuces pour contourner tous ces problèmes, mais il est tout à fait regrettable de voir une telle potentialité laissée ainsi inexploitée. Il ne fait aucun doute que le matériel vaut son prix certes, c'est tout bonnement dommage...



Gros plan sur l'affichage. On devine à peine les minuscules capsules contenant les particules en suspension permettant de former les pixels ainsi que l'aspect "granuleux" de l'affichage qui n'est pas sans rappeler une impression laser sur papier standard (ce qui est précisément l'objet de cette technologie).

Note de dernière minute : toutes les expérimentations pour cet article ont été menées en utilisant un clone d'Arduino Mega 2560 et en alimentant le module via sa broche 5V sans le moindre problème. Cependant, 48h après la finalisation de la rédaction, le module ne semble plus pouvoir être alimenté de cette façon en raison de pics de courant dépassant 1A, et nécessite donc l'utilisation d'une alimentation propre (l'Arduino n'étant pas en mesure de fournir un tel courant). Ce comportement étant intermittent, on peut supposer que le circuit d'alimentation de mon module ou de l'écran e-paper comporte un problème. Quoi qu'il en soit, si votre carte Arduino redémarre toute seule et/ou que l'intensité de la led du module fluctue, tentez d'alimenter le module avec une source distincte capable de fournir davantage de courant. **DB**





CRÉEZ UN MONITEUR DE TEMPÉRATURE ET D'HYGROMÉTRIE

Denis Bodor

Avoir une idée précise des conditions intérieures (ou extérieures) d'un emplacement est parfois simplement intéressant, mais tantôt très important. La température et l'humidité relative, par exemple, sont utiles pour le confort intérieur, mais également capitales selon ses activités comme la culture en serre, l'élevage d'animaux, les loisirs créatifs, l'affinage et/ou la fermentation de certains aliments ou encore, dans mon cas précis, le travail d'un matériau vivant comme le bois. Voici donc un projet d'afficheur combiné température/ hygrométrie sur base Arduino…

NIVEAU	TEMPS	BUDGET
	20 minutes	85 €

Dans mon cas tout à fait personnel à moi-même tout seul, et ayant acquis il y a quelques mois un petit local à rénover pour le transformer en atelier/labo, je me suis heurté à une constatation malheureuse : le lieu est humide, voire très humide. En raison d'une installation d'évacuation d'eaux de pluie quasi-médiévale (mais de façon évidente, pas du tout antédiluvienne), en cas de précipitations, l'eau s'infiltre dans le sol et remonte plus tard à l'intérieur du local sous forme de vapeur. Une nouvelle installation avec une réfection du puits perdu devrait régler le problème à terme, mais le lieu devant devenir mon terrain de jeu multidisciplinaire incluant le travail du bois, j'ai décidé de garder un œil sur l'humidité ambiante, de manière à éventuellement agir (aérer) en cas de dépassement d'un certain seuil critique.

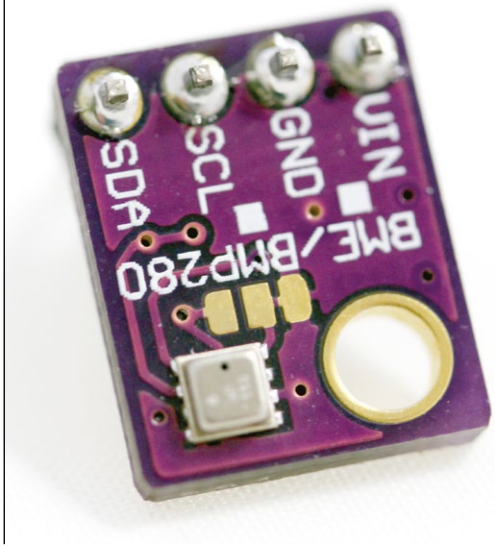
Il existe des accessoires tout faits pour ce genre de choses, sous la forme de petits modules généralement associés à une station météo, mais lorsque l'affichage est présent, il est ridiculement petit. Or je souhaite, d'un coup d'œil, avoir l'information que je désire. De plus, à terme, l'idée sera de déclencher une aération automatique (type VMC) ou un chauffage (pour l'hiver) pour tenter de baisser le taux d'humidité automatiquement.

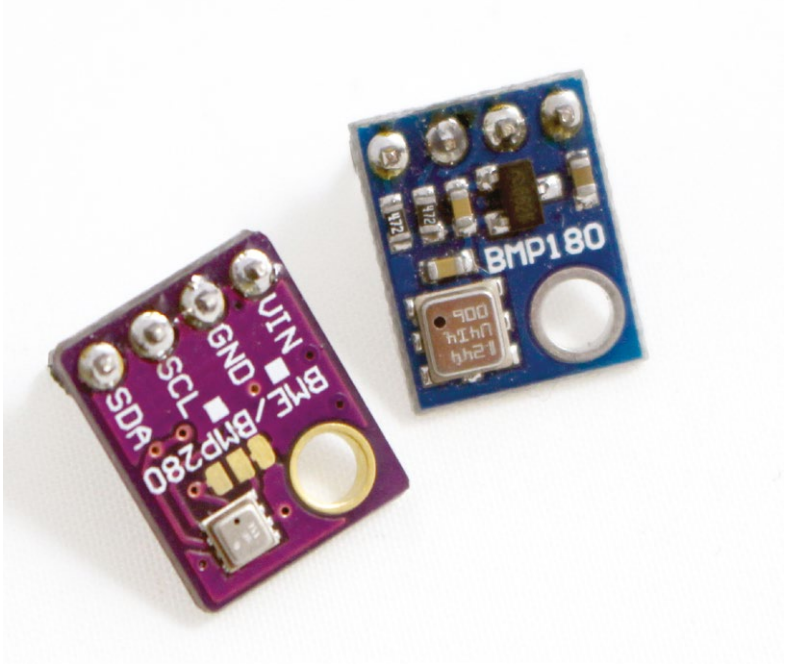
La notion d'humidité relative ou de degré d'hygrométrie est souvent mal comprise pour qui n'a pas fait l'effort de se renseigner. Cette valeur s'exprime en pourcentage et on serait tenté de lire un « 42% » comme signifiant qu'il y a 42% d'humidité ou d'eau dans l'air (ce qui ne veut pas dire grand-chose). En réalité, cette indication exprime le « rapport de la pression partielle de la vapeur d'eau contenue dans l'air sur la pression de vapeur saturante à la même température ». Merci Wikipédia, mais comme tantôt, en terminant de lire la réponse, on finit par ne plus comprendre sa propre question...

Sommairement, le degré d'hygrométrie indique la quantité d'humidité dans l'air **par rapport à son maximum**. Lorsque l'air est chaud, il peut contenir davantage d'humidité, lorsqu'il est froid il peut en contenir moins. C'est pour cette raison que lorsque vous prenez une douche, de la condensation se forme sur la fenêtre de votre salle de bain : la vapeur d'eau contenue dans l'air se condense, car la température sur la vitre est plus basse que dans la pièce.

Un taux d'humidité relative de 42% correspondra donc, non pas à une mesure, mais à l'indication « nous sommes à 42% du maximum de vapeur d'eau que peut contenir l'air à la température actuelle ». À 100%, une partie de cette eau va commencer à se condenser. Mais également, sans changer la quantité absolue d'eau dans l'air, si la température descend, le degré d'hygrométrie va augmenter. L'air peut contenir moins de vapeur d'eau et on approchera donc proportionnellement du maximum.

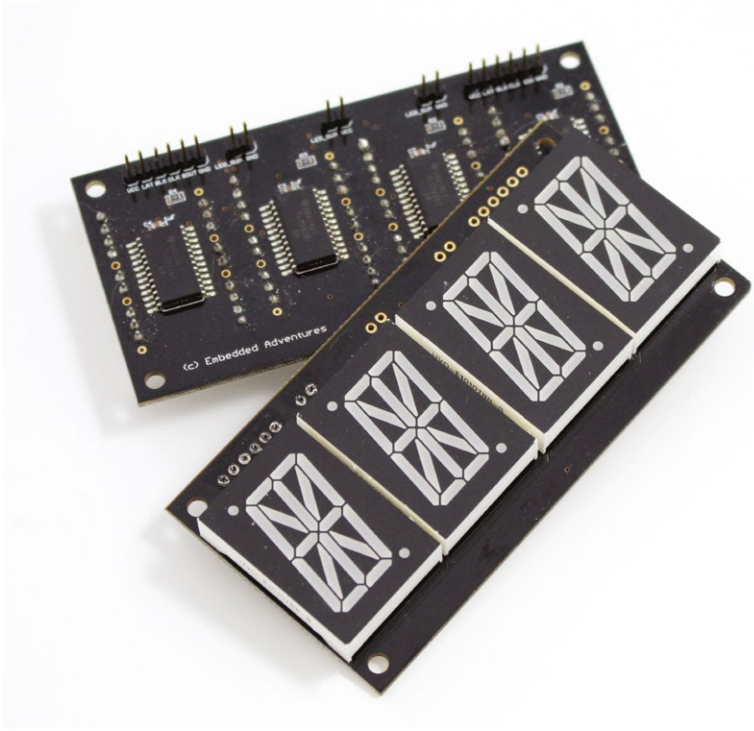
Cette toute petite chose de 1cm de côté est capable de mesurer la température, la pression atmosphérique et le taux d'hygrométrie de l'air. Remarquez les trois « pads » à souder permettant de choisir l'adresse du composant sur le bus i2c entre, par défaut 0x76 et, par coupure entre les deux pads de droite et soudure entre les deux de gauche, 0x77.





Deux modules reposant sur des circuits intégrés Bosch avec à gauche le BME280 utilisé ici et à droite le BMP180 utilisé dans un précédent article sur la mesure de pression et la création d'un baromètre Arduino.

Les modules d'affichage d'Embedded Adventures sont composés de 4 afficheurs 16 segments montés sur un circuit avec, à l'arrière les 4 TCL5926 permettant leur pilotage avec un nombre réduit de broches. Ceux-ci permettent d'afficher textes et chiffres dans différentes couleurs selon le modèle, mais coûtent tout de même le prix d'une carte Raspberry Pi 3.



C'est aussi pour ça qu'en chauffant une pièce, on réduit le degré d'hygrométrie. L'air est proportionnellement moins humide et voit alors son pouvoir séchant augmenter. Dans le cas de mon atelier/labo, il n'est pas question qu'il y ait condensation, ce qui risque d'arriver en particulier en hiver. Le fait de chauffer l'endroit n'est pas juste une affaire de confort, l'électronique ne fait généralement pas bon ménage avec l'eau...

LE PRINCIPE

Dans les grandes lignes, la version de base de ce projet est relativement simple. Nous souhaitons utiliser un module nous permettant de mesurer la température et le degré d'hygrométrie, puis nous servir de ces valeurs pour les afficher sur un dispositif relativement lisible à une distance de quelques mètres. Cette mesure sera effectuée à intervalles réguliers de façon à afficher les valeurs quasiment en temps réel.

On gardera à l'esprit une évolution possible du montage permettant le pilotage d'une VMC, d'un ventilateur ou d'un chauffage (électrique a priori) permettant éventuellement d'influer sur l'humidité relative soit en chassant l'air humide, en faisant entrer de l'air sec ou en augmentant globalement la température de la pièce. Le contexte ici est celui décrit plus haut, un atelier, mais ceci pourra aisément être adapté à une pièce à vivre, un vivarium ou encore une serre.

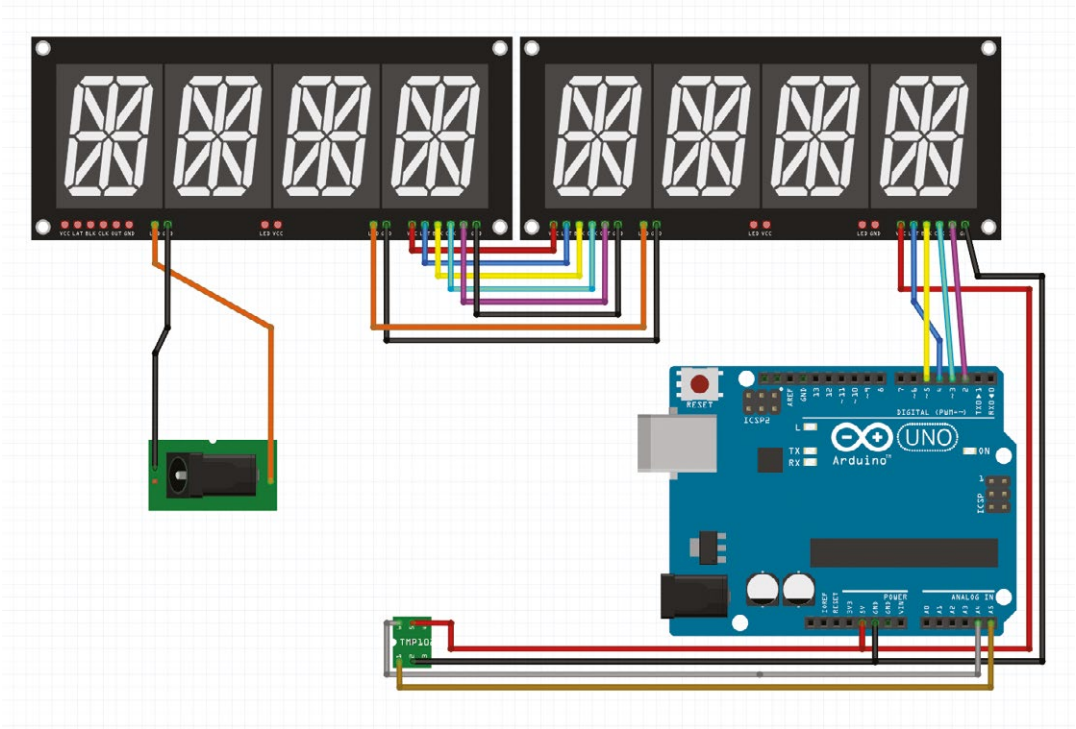
CE QU'IL VOUS FAUT

- **Carte Arduino : ce projet ne nécessite pas de fonctions particulières propres à un modèle spécifique de carte Arduino.**

- **Un module capteur température/hygrométrie/pression BME280. Nous avons traité précédemment dans le magazine le capteur BMP180 combinant mesure de température et de pression. Ceci est une version sensiblement plus complète offrant, en plus, la mesure du degré hygrométrique. Le composant actif de ce type de module, le BME280 lui-même, est un circuit intégré fabriqué par Bosch et pouvant être interfacé en SPI ou en i2c. La quasi-totalité des modules utilise la configuration i2c. Le module acheté, sur eBay auprès d'un vendeur appelé czb6721960, m'aura coûté moins de 5€ port offert. Il comprend le BME280, un régulateur de tension, un circuit pour adapter les niveaux de tension et quelques composants passifs (résistances).**

- **Deux modules de 4 afficheurs 16 segments pilotés par TLC5926, fabriqués et vendus par *Embedded Adventures* pour quelques 35€ pièce, port inclus. Il existe 5 modèles différents se distinguant par la couleur de l'affichage (vert, jaune, rouge, bleu et blanc). Pour le projet, j'ai opté pour un module rouge pour la température et un bleu pour l'hygrométrie. Ces modules sont construits autour de quatre exemplaires du circuit intégré TLC5926 faisant office de registre à décalage. Le principe consiste à stocker dans leur mémoire l'état des 16 segments permettant de former un symbole sur chacun des 4 afficheurs à leds. Les 4 TLC5926 sont chaînés entre eux de manière à prendre en compte l'état de 64 segments (4*16). Dans notre montage, deux modules forment, de la même manière, une chaîne de 8 TLC5926 pilotant les 128 segments. Un TLC5926 peut être alimenté avec une tension de 3,3 à 5 volts, mais les afficheurs à leds nécessitent une tension et un courant dépendant de leur couleur. Le TLC5926 est en mesure de réguler le courant adéquat pour chaque afficheur, mais il doit recevoir une tension suffisante pour le faire (3,8V pour rouge, 4,3V pour jaune et vert, 6,6V pour bleu et blanc). Le module dispose d'une broche VCC pour alimenter le TLC5926 et une autre broche, LED_SUP, pour les leds.**

- **Bloc d'alimentation 7,5V 1A. Ceci est une spécificité découlant du choix des couleurs des afficheurs. Le fait d'utiliser des leds bleues implique une tension minimum d'environ 7 volts. Si nous nous contentons d'utiliser des leds rouges, une alimentation 5V serait suffisante, mais dans un cas comme dans l'autre, la carte Arduino ne peut pas fournir le courant nécessaire pour quelques 128 leds. Une alimentation supplémentaire est donc indispensable. Notez que plus la tension délivrée sera éloignée de la valeur attendue par les leds, plus les TLC5926 devront dissiper de chaleur pour ajuster le courant. Alimenter le montage en 12V ou 15V est donc à éviter, à moins de prévoir un système de refroidissement (radiateur ou radiateur+ventilateur). Quoi qu'il en soit, le TLC5926 intègre une protection et se mettra automatiquement en arrêt en cas de dépassement de la température limite de fonctionnement. Petite remarque pratique au passage : pensez à éloigner le capteur de l'afficheur pour ne pas fausser les mesures.**



LE MONTAGE

Le montage nécessite beaucoup de connexions, mais est, dans les grandes lignes relativement simple. En premier lieu, nous avons le module BME280 qui est connecté en i2c à la carte Arduino. En plus de la masse et de la tension d'alimentation de 5 volts, deux lignes sont utilisées : SDA pour les données et SCL pour le signal d'horloge.

Notez que ces broches, côté Arduino, sont communes avec les broches d'entrées analogiques A4 (SDA) et A5 (SCL). On peut donc utiliser l'une et l'autre broche sans changement du croquis. Ici, pour rendre le dessin du montage visuellement plus clair, j'ai utilisé A4 et A5, mais dans la réalité, dans la salade de câbles générée, ce sont bien les broches libellées SDA et SCL qui sont utilisées (celles proches du connecteur USB).

La paire d'afficheurs demande un peu plus d'attention. Les modules de chez *Embedded Adventures* possèdent quelques 18 broches, mais plusieurs d'entre elles sont interconnectées. Nous avons ainsi 4 masses (GND), trois tensions d'alimentation pour les circuits intégrés TLC5926 (VCC), et trois tensions d'alimentation pour les leds (LED_SUP). Ce n'est pas tout, pour piloter les 8 TLC5926, nous utilisons également des broches communes à tous les composants et qui sont présentes deux fois sur le module :

- LAT est un signal permettant de valider les données envoyées et les mémoriser dans les TLC5926 ;
- CLK est le signal d'horloge permettant l'envoi des bits ;
- BLK est une broche utilisée pour activer ou désactiver les leds. Cette broche est reliée à une sortie « analogique » (PWM) de l'Arduino, nous permettant ainsi de régler une intensité lumineuse.

Restent alors deux broches : SIN correspondant à la réception des bits et SOUT pour l'émission. La logique utilisée par les TLC5926 consiste à recevoir les 16 bits concernant le premier afficheur 16 segments. Lorsque le bit suivant arrive, le premier qui a été envoyé « déborde » sur le second TLC5926 qui le considère comme son premier bit. En continuant ainsi de « pousser » des bits sur le premier TLC5926, le second circuit intégré finit lui aussi par laisser « déborder » des bits sur le troisième TLC5926 et ainsi de suite. Comme nous connectons le SOUT du premier module au SIN du second, ce débordement de bits se propage également aux 4 TLC5926 du second module. Au final, en envoyant 128 bits, nous définissons l'état des 128 segments des afficheurs et il ne nous reste plus qu'à utiliser la broche LAT pour enregistrer ces états dans l'ensemble des TLC5926.

LE CROQUIS

Fichier Édition Croquis Outils Aide

```
#include <DSP0401B.h>
#include <Wire.h>
#include <Adafruit_BME280.h>

#define SDI_PIN 2   // données
#define CLK_PIN 3   // horloge
#define LE_PIN  4   // validation
#define OE_PIN  5   // état affichage

// objet pour l'afficheur
DSP0401B mydisp;
// objet pour le capteur
Adafruit_BME280 bme;

void setup() {
  // 2 afficheurs en chaîne (8 caractères)
  mydisp.begin(2, SDI_PIN, CLK_PIN, LE_PIN, OE_PIN);
  // réglage luminosité
  mydisp.brightness(35);

  // initialisation capteur BME280, adresse 0x76
  if (!bme.begin(0x76)){
    while(1) {
      // erreur BME180, affichage message
      // boucle infinie
      mydisp.slidetext("BME180 ERROR",170);
    }
  }
}

void loop() {
  // tableau pour le texte (8 caractères + \0 de fin)
  char texte[9];
  // température
  int temp = (int)bme.readTemperature();
  // humidité relative
  int hygro= (int)bme.readHumidity();

  // composition du texte à afficher
  snprintf(texte, 9, "%2d#C%3d%%", temp, hygro);

  // affichage
  mydisp.sendtext(texte);

  // pause 1s
  delay(1000);
}
```

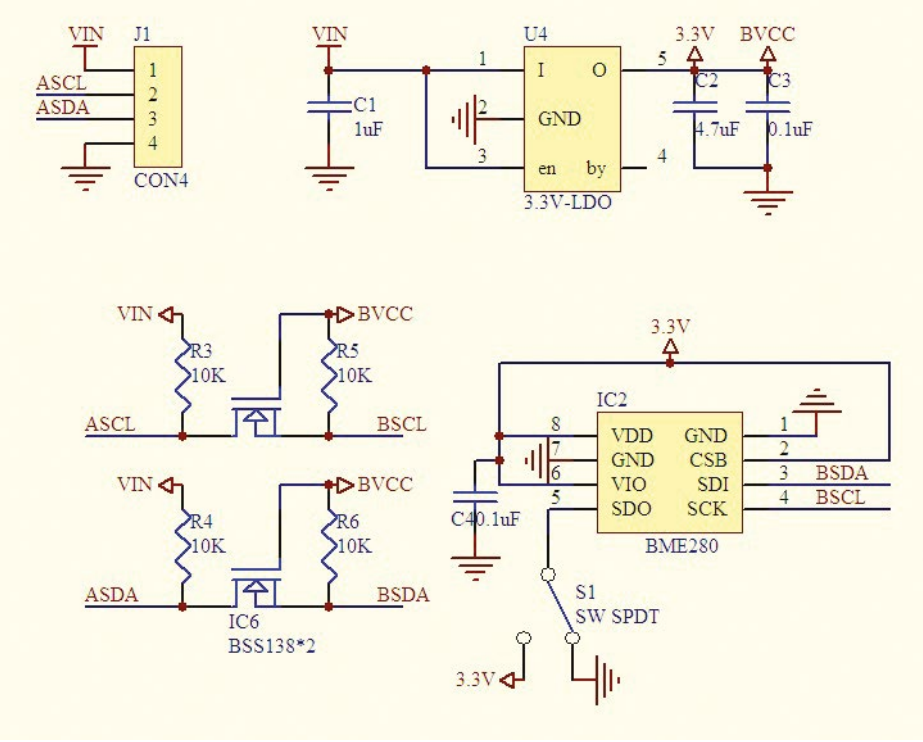
Arduino

À PROPOS DU CROQUIS

Le sujet principal du numéro 5 de *Hackable* concernait l'utilisation directe du registre à décalage TLC5926 et des modules d'affichage utilisés ici. En conclusion de nos manipulations de l'époque, nous avons produit une bibliothèque Arduino permettant de faciliter l'utilisation de ces afficheurs. Nous ne reviendrons donc pas sur le sujet du fonctionnement en détail du pilotage de l'afficheur et des TLC5926. Il vous suffira, en effet, d'installer la bibliothèque en question, téléchargeable ici : **https://github.com/Lefinnois/DSP0401B**. Placez simplement le répertoire `DSP0401B` et son contenu dans le répertoire des bibliothèques (`libraries`) de votre carnet de croquis et le tour sera joué.

Le module capteur quant à lui sera piloté par une autre bibliothèque, développée par AdaFruit et installable normalement depuis le gestionnaire de bibliothèques de votre environnement Arduino : `Adafruit_BME280`. Notez qu'il existe également deux autres bibliothèques, respectivement développées par SparkFun et un développeur indépendant (Tyler Glenn), mais je n'ai pas fait d'essais spécifiques, les résultats apportés par `Adafruit_BME280` me convenant parfaitement.

Une fois l'objet représentant le capteur instancié avec `Adafruit_BME280 bme`, nous pouvons utiliser ce dernier pour initialiser le capteur via sa méthode `begin()` en spécifiant, en argument, l'adresse du BME280 sur le bus i2c. Le composant peut utiliser deux adresses, `0x76` ou `0x77`, configurées en fonction de l'état de sa broche SDO. Sur le module testé, cette broche est connectée



Le vendeur eBay des modules capteurs fournit un schéma dans la description. On constate que le BME280 se trouve en réalité derrière un adaptateur de niveau de tension et est alimenté en 3,3V par un régulateur (LDO). Ceci contredit la description du produit qui indique une tension d'alimentation entre 1,8V et 5V, ainsi qu'un fonctionnement en i2c ou SPI. Ces données concernent en réalité le BME280 lui-même et non le circuit sur lequel il se trouve.

à la masse et l'adresse est donc **0x76**. Nous pourrions cependant changer cela en coupant la piste et en ajoutant un point de soudure à l'endroit dédié à cet effet. SDO serait encore connecté à la tension d'alimentation et l'adresse serait changée en **0x77**. Ceci permet d'utiliser de concert deux modules sur le même bus i2c d'une seule carte Arduino (par exemple, pour mesurer la température et l'hygrométrie intérieures et extérieures).

La récupération des valeurs dans le croquis se fait ensuite en utilisant les méthodes **readTemperature()** et **readHumidity()**, retournant toutes deux des valeurs en virgules flottantes (type **float**). Le fait de placer un **(int)** nous permet de convertir cela en entier puisque nous ne disposons pas de suffisamment de place pour afficher un nombre à virgule (et nous n'avons pas de point sur les afficheurs). Cette technique s'appelle le *casting* et on dit qu'on *cast* un type en un autre, ou qu'on force une conversion de type.

Mais la partie la plus intéressante de ce croquis tourne autour de l'utilisation de la fonction **snprintf()**. Cette fonction issue d'une famille complète incluant **vprintf**, **sprintf** ou encore **printf** que vous connaissez sans doute, permet d'obtenir une conversion vers une sortie formatée. Dans les grandes lignes, il s'agit de prendre des données statiques et issues de variables et de produire un contenu adapté à nos besoins.

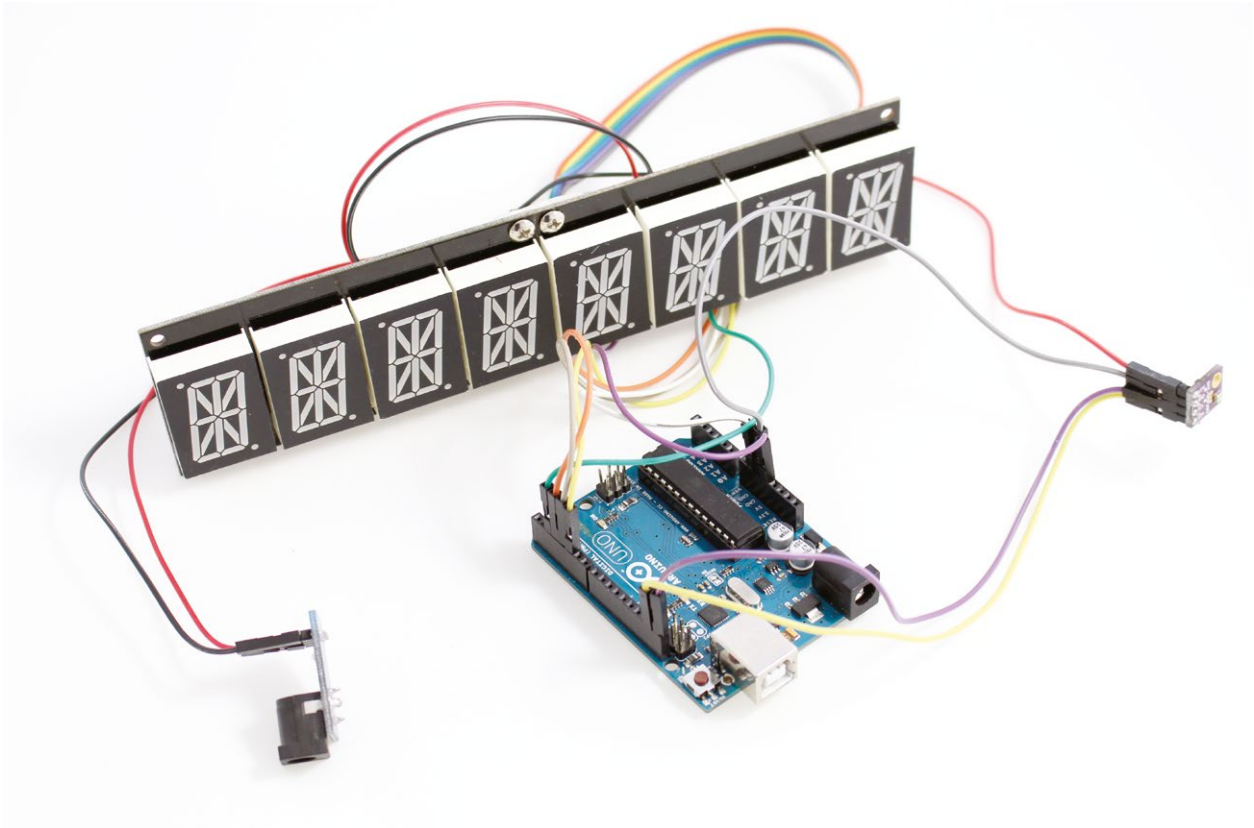
Pour comprendre l'utilité de cette fonction dans notre croquis, le plus simple est de tenter de ne pas se servir de **snprintf()**. Pour afficher le texte souhaité sur notre afficheur, nous avons simplement à notre disposition deux entiers (**int**) et nous devons produire une chaîne de 8 caractères. De plus, nous voulons que la valeur pour la température et le degré d'hygrométrie soient sur deux positions et accompagnés respectivement de « °C » et « % ». Enfin pour ne rien gâcher, nous voulons également que le degré d'humidité, indiqué par définition sur 3 caractères, soit aligné à droite (nous nous contentons de 2 chiffres pour la température, car à plus de 99°C je pense qu'on se rendrait compte du problème sans avoir besoin de l'afficheur).

Pour obtenir nos caractères à partir de valeurs entières, nous pouvons faire la chose suivante :

```
char texte[9] = "42#C 56%";
[...]
  texte[0]=(temp/10)+48;
  texte[1]=temp-((temp/10)*10)+48;
  texte[5]=(hygro/10)+48;
  texte[6]=hygro-((hygro/10)*10)+48;
```

Nous avons un tableau de 9 **char**, c'est notre chaîne de caractères qui par définition se termine toujours par **\0** pour marquer la fin de la chaîne. C'est la position 8 du tableau. Nous déclarons le tableau en l'initialisant avec une chaîne possédant déjà un texte statique avec les symboles aux bonnes positions. Notez l'astuce consistant à utiliser un « # » pour obtenir un « ° » sur l'afficheur, chose qui est intégrée à la bibliothèque **DSP0401B** (un « ° » dans l'IDE Arduino n'est pas un symbole ASCII 7 bits, mais un caractère UTF-8 sur 16 bits).

Il nous reste donc à compléter les positions 0 à 8. Pour définir un caractère à partir d'une valeur, nous n'avons qu'à utiliser la table ASCII. Là, on se rend compte que le caractère « 0 » correspond à la valeur 48, le « 1 » à 49 et ainsi de suite jusqu'à « 9 » étant 57. De ce fait, pour obtenir un « 0 » à partir d'un 0, il nous suffit d'ajouter 48. Il en va de même pour tous les autres chiffres.



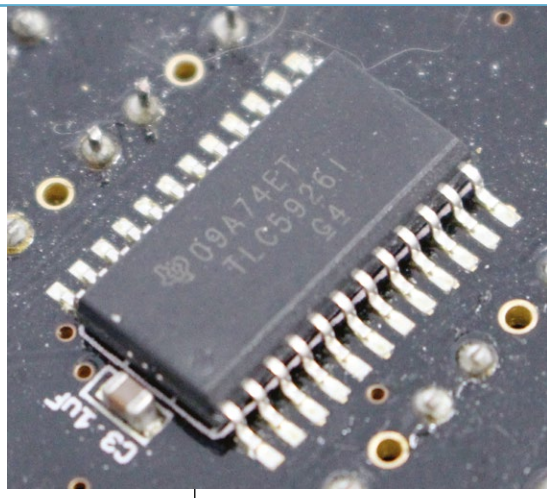
Notre montage final est riche en connexions diverses. Lors de l'installation définitive, cette connectique fragile sera remplacée par des câbles soudés et la carte Arduino très probablement changée par un modèle chinois plus économique (ou pourquoi pas un ATmega328 seul).

En position 0 de notre tableau, correspondant au caractère le plus à gauche sur l'afficheur, nous avons les dizaines de la valeur en degrés Celsius. Il nous suffit donc de prendre notre valeur entière de température, de la diviser par 10 puis d'ajouter 48 pour obtenir le caractère correspondant. Pour les unités, c'est tout aussi simple puisque nous prenons l'entier, lui retranchons ses dizaines et obtenons alors une valeur entre 0 et 9 à laquelle nous ajoutons 48 pour obtenir un nouveau caractère. Enfin, nous procédons de même pour le degré d'hygrométrie.

Ça marche, mais nous avons cependant un petit problème : ce n'est pas aussi joli que nous l'avions souhaité. Si la température est de 5°C, par exemple, **temp** divisé par 10 nous donne 0 et provoquera l'affichage « 05°C », ce qui n'est pas très lisible dans ce cas précis.

Nous pouvons alors modifier nos lignes de façon à utiliser un espace à la place d'un « 0 » sous certaines conditions :

```
char texte[9] = "42#C 56%";
[...]
  texte[0]= (temp<10) ? ' ' : (temp/10)+48;
  texte[1]=temp-((temp/10)*10)+48;
  texte[5]= (hygro<10) ? ' ' : (hygro/10)+48;
  texte[6]=hygro-((hygro/10)*10)+48;
```



Le registre à décalage TLC5926 est l'élément central des modules d'affichage. C'est avec ces composants que l'Arduino va dialoguer pour définir l'état de chaque segment de l'affichage et ainsi former des lettres et des chiffres visibles et lisibles de loin.

La condition est ici spécifiée sous la forme d'un opérateur ternaire. La première ligne pourrait être remplacée en :

```
if(temp<10)
  texte[0] = ' ';
else
  texte[0] = (temp/10)+48;
```

Il est simplement plus concis d'utiliser la syntaxe **condition ? si vrai : si faux;**. Le résultat sera, pour une température de 5°C et une humidité relative de 42%, une chaîne produite en **" 5#C 42%"**.

Vous me direz, c'est beaucoup de code pour une simple présentation et/ou un formatage d'une chaîne de caractères, il doit y avoir quelque chose de plus élégant. Et vous avez raison, la famille de fonctions **printf()** est précisément là pour ça. Mais attention, le gain en souplesse a un coût.

Notre **snprintf()** prend en argument un pointeur sur la variable cible (**texte** est un pointeur, **texte[0]** est l'élément à la première position), le nombre d'octets que la fonction doit produire au maximum (avec le **\0** de fin de chaîne), une constante définissant le format à utiliser et la ou les variables dont on fait usage pour le formatage.

Ici, la constante de formatage dit : « un entier signé sur deux chiffres » (**%2d**), « les caractères # et C », « un entier signé sur 3 chiffres » (**%3d**) et « le caractère % ». Le fait d'utiliser **%d** nous permet, pour la température, d'afficher une valeur négative, chose que nous ne prenons pas en compte avec nos codes précédents. Attention toutefois, le **2** de **%2d** est le nombre **minimum** de chiffres, pas maximum. Ceci signifie qu'avec une température de -10°C, le « C » débordera sur l'afficheur bleu et comme nous avons précisé 3 chiffres pour l'humidité, le « % » sera alors supprimé.

Une solution possible consiste à ne plus afficher de « C » du tout et d'utiliser 3 chiffres pour la température :

```
snprintf(texte, 9, "%3d#%3d%%", temp, hygro);
```

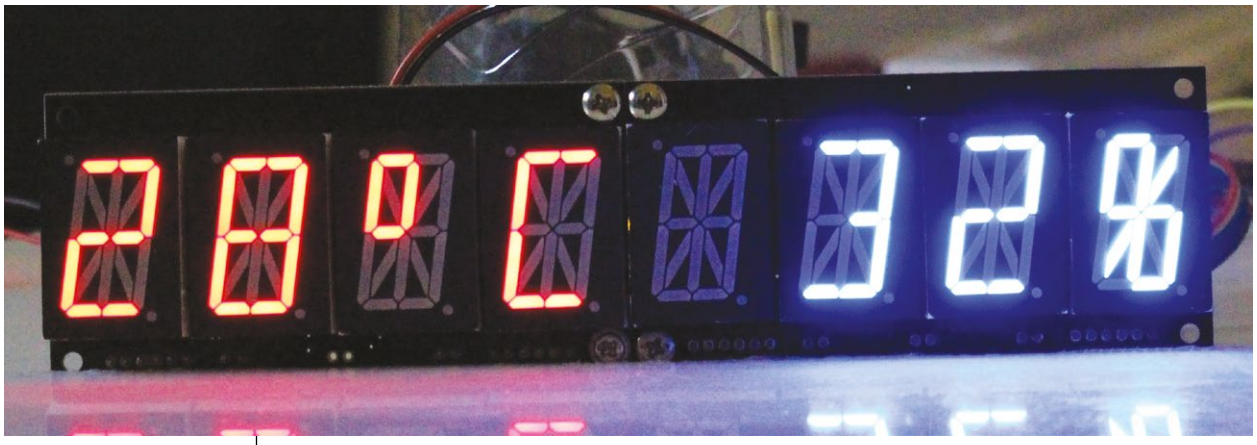
ou garder le « C », mais réduire le degré d'hygrométrie à 2 chiffres (le « % » restera, mais le « C » sera bleu à partir de -10°C) :

```
snprintf(texte, 9, "%2d#C%3d%%", temp, hygro);
```

Et si vraiment, vraiment, vous voulez faire les choses le mieux possible, vous pouvez utiliser :

```
snprintf(texte, 9, "%2d%s%3d%%",temp, (temp<-9) ? "#" : "#C", hygro);
```

Nous ajoutons ici un spécificateur de format **%s** permettant de spécifier une chaîne de caractères. Celle-ci est alors ajoutée dans les arguments sous la forme d'un opérateur ternaire et sera **"#"** si la température est inférieure à -9°C et **"#C"** si elle est égale ou supérieure à -9°C. En d'autres termes, s'il y a de la place pour afficher « °C » on le fait, et



Une fois en route, le montage va toutes les secondes mesurer la température ambiante et le degré d'hygrométrique puis mettre à jour l'affichage. L'intensité lumineuse des afficheurs 16 segments peut être contrôlée. Une évolution possible du projet pourra être une mesure de la luminosité avec une simple LDR sur une entrée analogique pour adapter automatiquement l'intensité lumineuse et réduire à la fois la consommation de courant et la température des TLC5926.

sinon on se contente de « ° » pour tenir dans les 4 caractères rouges.

Tout ceci est bien joli, mais il faut maintenant parler du coût. Le formatage manuel de l'affichage produit un croquis binaire de 8612 octets (26%), consommant 298 octets de mémoire vive (14%). Le fait d'utiliser `snprintf()` augmente ces quantités à 9892 octets (30%) et 314 octets de mémoire (15%). Dans le cas présent, ce n'est pas un gros problème, nous perdons 1280 octets de flash (sur 32Ko pour UNO) et 16 octets de SRAM (sur 2Ko), mais nous ne les utilisons de toute façon pas. Mais pour certains projets, plus d'un Ko de flash peut être une quantité non négligeable et il faudra se passer de ce type de fonctions pour faire tenir le croquis dans la mémoire de l'Arduino. Pensez-y...

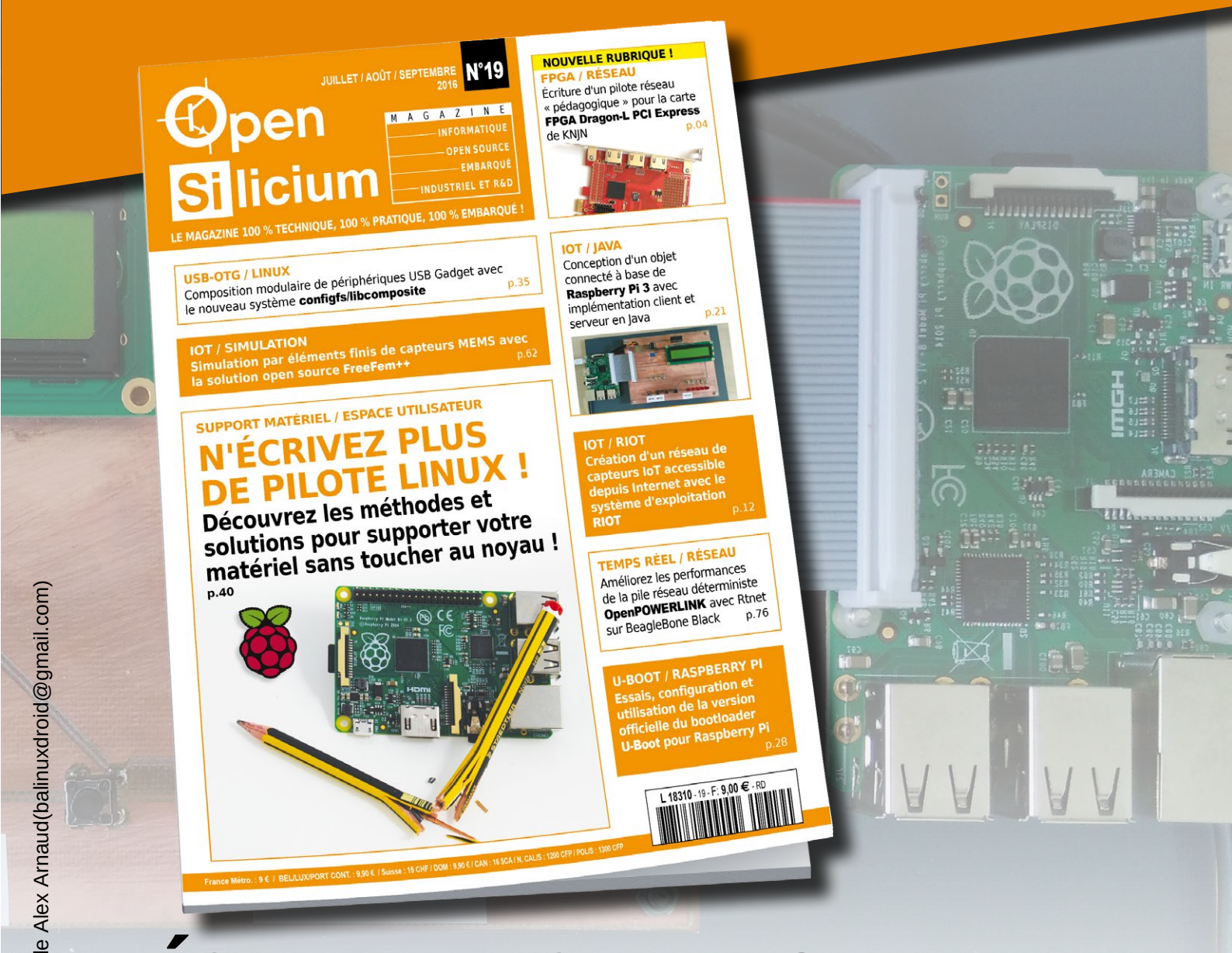
CONCLUSION

Je tiens à préciser ici que l'utilisation des deux modules d'affichage de chez *Embedded Adventures* représente une part très importante du budget consacré à ce projet. Des alternatives existent puisqu'il est possible d'utiliser des afficheurs 16 segments en utilisant d'autres registres à décalage sous la forme d'un circuit « fait maison ». Il est également possible de se passer de l'affichage alphanumérique et se contenter de chiffres. Des modules utilisant des afficheurs 7 segments sont bien moins chers.

Une autre voie si l'on veut conserver un aspect bien visible des valeurs mesurées, mais en faisant une concession sur la précision, consiste à utiliser des leds WS2812b, comme sur la réglette décrite dans le précédent numéro. En segmentant la plage de température et d'humidité relative, il est possible de se limiter à quelques 10 ou 20 leds, en alternant entre un affichage rouge et bleu, par exemple. Nous pourrons même ajouter une indication de la pression avec une troisième couleur sans dépenser un euro de plus.

Enfin, tout dépend également du contexte d'utilisation de ce système. Pour une serre ou une pièce de petite dimension, l'utilisation d'un afficheur LCD HD44780 ou d'un écran Oled peut être une option valable. Il faudra simplement se déplacer pour lire les valeurs, ce qui n'est pas toujours un problème. Depuis 13 numéros de *Hackable*, nous avons, je pense, eu l'occasion de voir bon nombre de solutions permettant d'afficher une valeur. À vous de faire travailler votre imagination si vous souhaitez quelque chose d'original... **DB**

ACTUELLEMENT DISPONIBLE
OPEN SILICIUM N°19 !



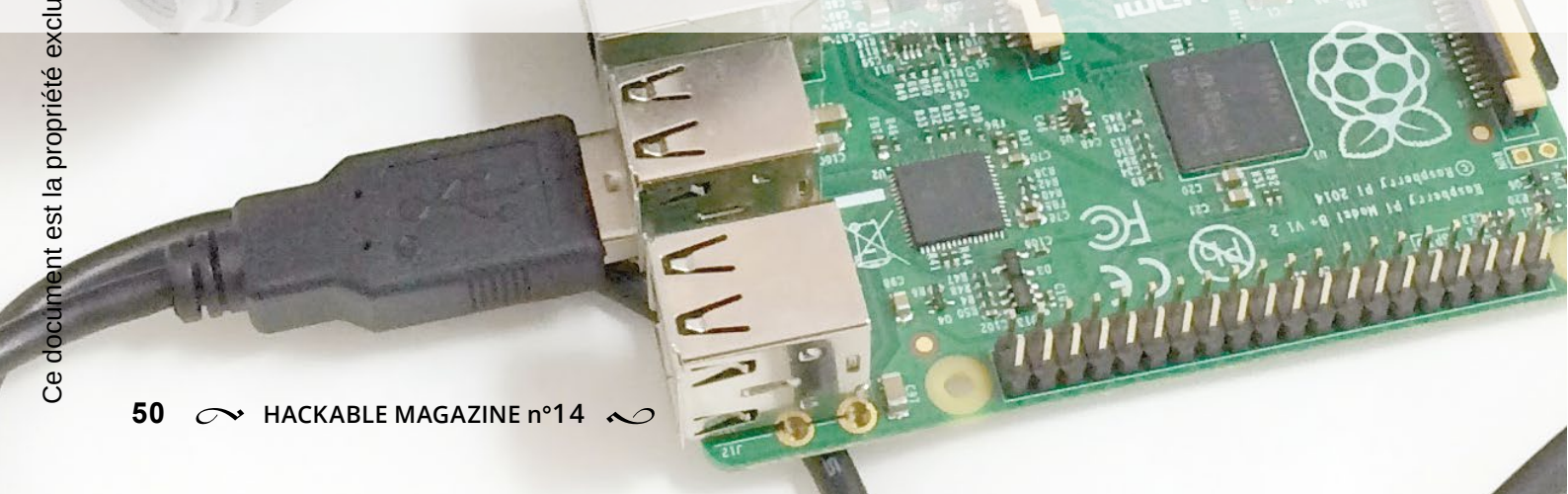
N'ÉCRIVEZ PLUS DE PILOTE LINUX !
DÉCOUVREZ LES MÉTHODES ET SOLUTIONS POUR SUPPORTER VOTRE MATÉRIEL SANS TOUCHER AU NOYAU !

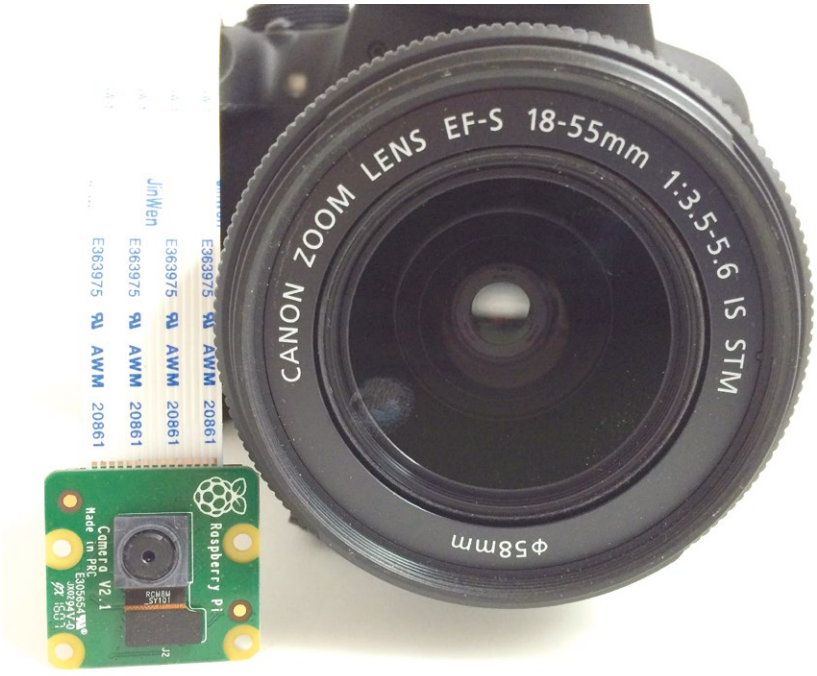
NE LE MANQUEZ PAS
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
www.ed-diamond.com

CONTRÔLEZ VOTRE APPAREIL PHOTO NUMÉRIQUE AVEC VOTRE PI

Denis Bodor

Le fait de disposer d'un module caméra pour la Raspberry Pi est un avantage certain et permet de faire beaucoup de choses. On est cependant très loin de ce que peut offrir un appareil reflex numérique, même de moyenne gamme, en termes de qualité d'image. D'un autre côté les firmwares des appareils photo manquent souvent de souplesse et de fonctionnalités, en particulier pour l'automatisation des prises de vue. La solution pour profiter du meilleur des deux mondes est simple : piloter son APN avec une Pi.





L'utilisation du module caméra de la Raspberry Pi permet d'automatiser énormément d'opérations pour les prises de vue, mais il faut se rendre à l'évidence, une lentille de 1mm ne fait pas le poids face à un objectif de reflex ou d'hybride. Avoir plein de mégapixels c'est une chose, le « trou » par lequel ils entrent en est une autre...

L'exemple le plus évident pour comprendre l'intérêt d'un contrôle automatisé d'un appareil photo moyen ou haut de gamme (typiquement un reflex numérique) est le *time-lapse*. Cette technique consiste à prendre une photo à chaque intervalle de temps prédéfini et de combiner tous les clichés en une vidéo ou une image unique. Un autre exemple est la prise d'une série de clichés avec pour chacun un temps d'exposition ou une sensibilité ISO différente, pour ensuite combiner les photos pour créer une image à grande gamme dynamique (ou HDRI).

Il est très rare que ce type de fonctionnalité soit intégré dans le logiciel de l'appareil photo, même pour les modèles les plus coûteux. L'une des solutions consiste à utiliser, par exemple, un déclencheur programmable à brancher à l'appareil ou encore, pour certains modèles, utiliser un firmware alternatif en open source comme le fantastique Magic Lantern (pour Canon EOS, **http://www.magiclantern.fm/features.html**).

Une solution plus économique et bien plus dans l'esprit du magazine est de tout simplement connecter l'appareil photo à une Raspberry Pi en USB et d'utiliser un programme en ligne de commandes appelé gPhoto pour choisir les caractéristiques de la ou ou des prises de vue et, bien entendu, prendre des photos à volonté. Notez qu'il existe également des applications Android permettant de faire de même, comme *DSLR Controller* de *Chainfire*, mais qui sont payantes et surtout nécessitent un smartphone ou une tablette supportant l'USB hôte.

La solution Raspberry Pi + câble USB standard est donc non seulement la plus économique, mais également la plus souple.

1. MATÉRIEL UTILISÉ POUR L'ARTICLE

gPhoto supporte quelques 1828 modèles différents d'appareils photo numériques dans sa version 2.5.4 actuellement disponible pour Raspbian. Les explications qui vont suivre sont valables pour n'importe quel modèle de Raspberry Pi et également pour un simple PC sous GNU/Linux.

Côté appareil photo, notre victime innocente sera un Canon EOS 700D connecté à l'aide d'un câble USB mini type B mâle vers type A mâle. Bien entendu, le type de câble USB nécessaire sera dépendant du modèle d'appareil. Notez qu'ici j'utilise un modèle

relativement coûteux (~550€), mais gPhoto ne se limite pas à ce type de matériel. Un vieux compact Nikon Coolpix 885 sera parfaitement pris en charge par exemple.

La première chose à faire avant de vous lancer sera donc de vérifier que votre matériel est bien supporté. Après avoir installé l'utilitaire avec **sudo apt-get install gphoto2**, une simple commande fera l'affaire :

```
$ gphoto2 --list-cameras | grep -i "EOS 7"
        " Canon EOS 700D "
        " Canon EOS 70D "
        " Canon EOS 7D "
```

L'option **--list-cameras** affiche l'ensemble des modèles pouvant être utilisés et nous redirigeons ce résultat avec **|** vers la commande **grep** afin de n'afficher que les lignes contenant le texte **"EOS 7"**. Si votre matériel apparaît à l'écran, c'est très bon signe, car cela signifie qu'il est au moins en partie supporté. Certains appareils utilisent le protocole MTP, d'autres PTP et d'autres encore une solution spécifique. Actuellement, PTP semble être devenu la norme et ce protocole est donc commun à tous les appareils récents, seules les spécificités de certains modèles peuvent ne pas être supportées par gPhoto.

L'étape suivante consiste à connecter l'appareil à la Raspberry Pi puis à l'allumer. Le système devrait alors détecter le périphérique comme en témoignent les messages du noyau :

```
$ dmesg | tail
usb 1-1.2: new high-speed USB device number 10 using dwc_otg
usb 1-1.2: New USB device found, idVendor=04a9, idProduct=3272
usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1.2: Product: Canon Digital Camera
usb 1-1.2: Manufacturer: Canon Inc.
```

Il ne vous reste plus ensuite qu'à vous assurer que gPhoto « voit » effectivement votre appareil avec :

```
$ gphoto2 --auto-detect
Modèle                Port
----------------------------------
Canon EOS 700D        usb:001,010
```

On constate ici qu'un Canon EOS 700D est connecté sur le bus USB 001 en tant que périphérique numéro 010. Vous pourrez également obtenir des informations complémentaires en utilisant **gphoto2 --summary**, comme le numéro de série, la version du firmware, etc.

2. PREMIÈRES MANIPULATIONS

Instinctivement, dès la connexion, la première chose qu'on a envie de tester est la prise de clichés. Inutile de tourner autour du pot au risque d'être frustré, la manipulation est d'une extrême simplicité :

```
$ gphoto2 --capture-image-and-download
Le nouveau fichier est à l'emplacement /capt0000.jpg de l'appareil
Enregistrement du fichier en capt0000.jpg
Effacement du fichier /capt0000.jpg de l'appareil
Suppression de " capt0000.jpg " du répertoire " / "...
```

L'option **--capture-image-and-download**, comme son nom l'indique, va provoquer le déclenchement de l'appareil photo, l'enregistrement de l'image, son téléchargement via USB sous le nom **capt0000.jpg** puis sa suppression de l'appareil. Attention cependant, selon les appareils l'image ne sera pas enregistrée puis supprimée sur le support d'enregistrement, mais en mémoire vive. Avec mon EOS 700D par exemple, la configuration par défaut utilise la mémoire SDRAM de l'APN, ce qui peut être déroutant avec d'autres commandes comme :

```
$ gphoto2 --capture-image
Le nouveau fichier est à l'emplacement /capt0000.jpg de l'appareil
```

Contrairement à ce que le message laisse penser, on ne retrouve pas de **capt0000.jpg** sur la carte SD, car ce fichier est présent uniquement en mémoire vive de l'appareil. Ainsi, pour prendre des clichés et conserver les images comme lorsqu'on utilise classiquement l'appareil, il faut changer la configuration avec :

```
$ gphoto2 --get-config capturetarget
Label: Cible d'acquisition
Type: RADIO
Current: Mémoire interne
Choice: 0 Mémoire interne
Choice: 1 Carte mémoire

$ gphoto2 --set-config capturetarget=1

$ gphoto2 --get-config capturetarget
Label: Cible d'acquisition
Type: RADIO
Current: Carte mémoire
Choice: 0 Mémoire interne
Choice: 1 Carte mémoire
```

La première commande liste le contenu de l'option **capturetarget** déterminant ce que l'APN doit faire de l'image. On voit clairement (ligne **Current:**) que le choix en place est **Mémoire interne**. La seconde commande, utilisant l'option **--set-config** change cela en **1**, **Carte mémoire**, et la dernière commande nous confirme le changement. À présent, en utilisant **--capture-image**, le résultat est bien différent :

```
$ gphoto2 --capture-image
Le nouveau fichier est à l'emplacement
  /store_00020001/DCIM/100CANON/IMG_0994.JPG de l'appareil
```

L'outil ne parle plus de **capt0000.jpg**, mais de **IMG_0994.JPG**, un nom de fichier bien plus classique. De plus, le chemin complet où est enregistré le fichier sur l'APN correspond également à un emplacement identique à celui utilisé lors de clichés pris manuellement. Notez que **/store_00020001** n'est pas vraiment un répertoire, mais le « point de montage »

La connexion entre l'appareil photo et la Raspberry Pi se fait à l'aide d'un simple câble USB. Ceci est une fonction standard des appareils qui ne risque pas d'endommager quoi que ce soit. Ici, j'ai fait mes tests avec un Canon EOS 700D, mais plus de 1200 modèles sont compatibles avec l'outil gPhoto, y compris des APN compacts, des modèles d'entrée de gamme ou des appareils ayant un bon nombre d'années.

par lequel le logiciel de l'appareil accède à la carte SD.

Une autre option utilisable avec certains appareils est **`--trigger-capture`** permettant de déclencher une prise de vue sans autre forme de procès. Cette option, lorsque disponible, comme pour l'EOS 700D, est sensiblement plus rapide que **`--capture-image`**.

Notez que le fait de modifier la valeur de l'option **`capturetarget`** perdure entre deux mises en route de l'appareil, et est même conservé lors d'un changement d'accu. Ceci peut être important dans le sens où l'utilisation de la mémoire interne est intéressante pour **`--capture-image-and-download`** (plus rapide), mais que l'espace de stockage SD l'est davantage pour **`--capture-image`**. Il sera donc de bon ton de vérifier la configuration avant de lancer une série de prises de vue.

3. GÉRER LES FICHIERS

Récupérer systématiquement le cliché pris sous la forme d'un fichier JPEG n'est pas forcément quelque chose de souhaitable tant d'un point de vue de l'organisation que du temps de traitement lors de chaque déclenchement. Heureusement, il est donc possible de laisser les fichiers sur l'appareil, mais gPhoto fournit également un certain nombre de commandes de gestion intéressantes.

La première d'entre elles est celle permettant de lister le contenu de l'appareil :

```
$ gphoto2 --list-files
Le dossier " / " ne contient aucun fichier.
Le dossier " /store_00020001 " ne contient aucun fichier.
Le dossier " /store_00020001/DCIM " ne contient aucun fichier.
Le dossier " /store_00020001/DCIM/100CANON " contient
10 fichiers.
#1     IMG_0992.JPG                rd  4947 KB image/jpeg
#2     IMG_0993.JPG                rd  4977 KB image/jpeg
#3     IMG_0994.JPG                rd  5138 KB image/jpeg
#4     IMG_0995.JPG                rd  5056 KB image/jpeg
#5     IMG_0996.JPG                rd  5033 KB image/jpeg
```

```
#6      IMG_0997.JPG                rd  4941 KB image/jpeg
#7      IMG_0998.JPG                rd  4946 KB image/jpeg
#8      IMG_0999.JPG                rd  4931 KB image/jpeg
#9      IMG_1000.JPG                rd  4403 KB image/jpeg
#10     IMG_1001.JPG                rd  4297 KB image/jpeg
Le dossier " /store_00020001/MISC " ne contient aucun fichier.
```

--list-files peut également être réduit en l'option courte **-L** (consultez la page de manuel pour en savoir plus avec **man gphoto2**). Nous voyons ici qu'il y a 10 JPEG placés dans un répertoire **DCIM/100CANON** sur la carte SD (**/store_00020001/**). Ces 10 fichiers sont affichés par leur nom avec une indication de leur taille et de leur type, mais aussi, et surtout précédés d'un numéro en début de ligne.

Il est possible alors de récupérer un ou plusieurs fichiers en les désignant de cette façon :

```
$ gphoto2 --get-file 2
Téléchargement de " IMG_0993.JPG " depuis le répertoire
   " /store_00020001/DCIM/100CANON "...
Enregistrement du fichier en IMG_0993.JPG
```

Bien entendu, récupérer un unique fichier n'a pas de sens et il est bien plus intéressant d'en « télécharger » une série en spécifiant une plage spécifique comme :

```
$ gphoto2 --get-file 4-9
Téléchargement de " IMG_0995.JPG " depuis le répertoire
  " /store_00020001/DCIM/100CANON "...
Enregistrement du fichier en IMG_0995.JPG
Téléchargement de " IMG_0996.JPG " depuis le répertoire
  " /store_00020001/DCIM/100CANON "...
Enregistrement du fichier en IMG_0996.JPG
Téléchargement de " IMG_0997.JPG " depuis le répertoire
  " /store_00020001/DCIM/100CANON "...
Enregistrement du fichier en IMG_0997.JPG
Téléchargement de " IMG_0998.JPG " depuis le répertoire
  " /store_00020001/DCIM/100CANON "...
Enregistrement du fichier en IMG_0998.JPG
Téléchargement de " IMG_0999.JPG " depuis le répertoire
  " /store_00020001/DCIM/100CANON "...
Enregistrement du fichier en IMG_0999.JPG
Téléchargement de " IMG_1000.JPG " depuis le répertoire
  " /store_00020001/DCIM/100CANON "...
Enregistrement du fichier en IMG_1000.JPG
```

Nous récupérons ici les fichiers 4 à 9 de la précédente liste obtenue avec **--list-files**. La version courte de l'option est **-p**. Enfin, vous pouvez également, tout simplement, récupérer l'ensemble des fichiers en une fois avec **--get-all-files**. Fort pratique si vous avez votre APN sur un trépied et n'avez vraiment pas envie de jongler avec la carte SD.

4. RÉGLER LA PRISE DE VUE

La méthode de configuration des paramètres d'une prise de vue peut paraître surprenante au premier regard, mais est relativement facile à maîtriser. En effet, les différentes choses qui peuvent être réglées prennent la forme d'une arborescence et de pseudo-fichiers côté appareil photo. Ainsi pour lister toutes les informations lisibles et définissables, on commence par afficher cette liste de pseudo-fichiers :

```
$ gphoto2 --list-config
/main/actions/syncdatetimeutc
/main/actions/syncdatetime
/main/actions/uilock
/main/actions/autofocusdrive
/main/actions/manualfocusdrive
/main/actions/cancelautofocus
/main/actions/eoszoom
/main/actions/eoszoomposition
/main/actions/viewfinder
/main/actions/eosremoterelease
[...]
/main/capturesettings/aperture
/main/capturesettings/shutterspeed
/main/capturesettings/meteringmode
/main/capturesettings/bracketmode
/main/capturesettings/aeb
/main/other/d402
/main/other/d407
/main/other/d406
/main/other/d303
/main/other/5001
```

La liste est longue, mais, en plus du simple nom du pseudo-fichier, il vous suffit de lire son contenu pour savoir précisément de quoi il en retourne. Exemple :

```
$ gphoto2 --get-config whitebalance
Label: Balance des blancs
Type: RADIO
Current: Automatique
Choice: 0 Automatique
Choice: 1 Lumière naturelle
Choice: 2 Ombre
Choice: 3 Nuageux
Choice: 4 Tungstène
Choice: 5 Fluorescent
Choice: 6 Flash
Choice: 7 Manuel
```

whitebalance permet donc de régler la balance des blancs. Pour régler un paramètre, on utilise **--set-config** suivi du nom du paramètre (pseudo-fichier), d'un signe égal et du réglage souhaité. Il y a cependant une subtilité puisque ce qui est attendu est une valeur. Or certains réglages se présentent ainsi :

```
$ gphoto2 --get-config iso
Label: Vitesse ISO
Type: RADIO
Current: Automatique
Choice: 0 Automatique
Choice: 1 100
Choice: 2 200
Choice: 3 400
Choice: 4 800
Choice: 5 1600
Choice: 6 3200
Choice: 7 6400
Choice: 8 12800
```

Nous avons là une liste de choix indexés et une valeur. Le choix 3, par exemple, correspond à une sensibilité de 400 ISO. On pourrait penser qu'il suffit de faire **`gphoto2 --set-config iso=3`**, mais il n'en est rien. Si une valeur peut être précisée, alors il faut utiliser la valeur avec **`-set-config`** et non le numéro du choix.

L'un des principaux intérêts du fait de piloter un appareil photo avec une carte comme la Pi est de pouvoir créer des time-lapses avec un intervalle de temps important (en heures). Dans ce genre de situations, l'usage d'une alimentation sur accu n'est pas une bonne idée puisque si l'appareil passe en veille il n'est plus accessible via USB, de ce fait il faut désactiver la mise en veille et l'autonomie s'en trouve drastiquement réduite. La solution est donc de basculer sur une alimentation « secteur » pour laisser l'appareil en marche en permanence.

Le plus simple dans ce cas est d'oublier **-set-config** et de s'en remettre à deux autres options moins ambigües :

- **--set-config-value** pour spécifier ici **gphoto2 --set-config-value iso=400**,
- ou **--set-config-index** pour renseigner **gphoto2 --set-config-index iso=3**.

Dans les deux cas, l'ISO sera réglée sur 400. Ceci prend tout son sens lorsqu'il s'agit par exemple de la vitesse d'obturation (**shutterspeed**). Un **1** pourrait être « une seconde » (choix 16 dans la liste) ou le choix 1, soit 30 secondes. Utiliser **--set-config-value** et **--set-config-index** vous évitera de faire comme moi, vous mélanger les pinceaux et perdre votre temps pour rien.

Notez, à propos de ces réglages, que la liste dépend du mode dans lequel se trouve l'appareil photo. L'EOS 700D par exemple, avec le sélecteur en mode A+ ne proposera naturellement pas certains réglages accessibles en mode « M » (manuel) puisqu'ils sont automatiques. C'est logique, mais là encore c'est quelque chose auquel il faut être attentif pour éviter de perdre son temps (et de pester pour rien).

5. DEUX PETITS EXEMPLES

Commençons par l'incontournable *time-lapse* consistant à prendre des clichés à intervalles réguliers pour pouvoir observer un phénomène qui échappe normalement à nos sens (comme la pousse d'une plante par exemple). gPhoto rend cela très facile avec des options qui ne sont pas sans rappeler celles de **raspistill**, l'outil de prise de vue du module caméra Raspberry Pi. Ainsi pour faire une série de 5 clichés espacés de 10 secondes, on se pliera simplement de :

```
$ gphoto2 --capture-image-and-download \
--frames 5 --interval 10 --filename capture%05n.jpg
```

Voici le genre de résultats qu'on peut obtenir en combinant plusieurs clichés d'une scène ayant une grande gamme dynamique (HDR) avec des temps d'expositions différents. La première photo utilise un temps d'exposition de 1/12 de secondes, on voit l'extérieur, mais l'intérieur est sous-exposé. La seconde photo avec un temps d'exposition d'une seconde nous permet de distinguer les détails dans la pièce, mais l'extérieur est sur-exposé. En combinant plusieurs clichés, on obtient la dernière image, nous permettant de voir l'ensemble de la scène. L'outil utilisé pour la fusion est enfuse, disponible dans Raspbian.

`--frames` précise le nombre d'images et **`--interval`** le temps en secondes à attendre entre deux clichés. **`--filename`** nous permet de préciser la façon de nommer les fichiers qui sont obtenus par **`--capture-image-and-download`**. Ici, nous utilisons **`%n`** pour obtenir le numéro dans la séquence. Le **`05`**, permettant de former **`%05n`**, précise une valeur sur 5 chiffres avec un préfixe composé de zéros. Nous obtenons ainsi nos cinq fichiers nommés de **`capture00001.jpg`** à **`capture00005.jpg`**. Il existe d'autres « marqueurs » permettant de nommer de façon judicieuse les fichiers. **`--filename capture%Y-%m-%d_%H:%M:%S.jpg`**, par exemple, inclura dans le nom la date et l'heure avec un format permettant facilement le tri numérique.

Le délai entre les clichés, même exprimé en secondes permet de faire bien plus que ce que le logiciel interne par défaut (Magic Lantern étant ce genre de fonctions) ou même un accessoire annexe peut proposer. Une chose cependant doit être gardée à l'esprit : l'autonomie de l'appareil. Dans bien des cas, un adaptateur secteur est indispensable et on prendra soin de régler le délai avant extinction sur une durée infinie. Personnellement, j'ai choisi l'ACK-E8 officiel de Canon pour l'EOS 700D. Cela m'a coûté quelques 70€, mais j'ai préféré ne pas prendre le risque d'endommager un reflex numérique à plus de 500€ en optant pour un clone chinois de qualité douteuse à 20€. Il y a suffisamment de vidéos sur YouTube comparant des alimentations d'un point de vue technique pour savoir que prix, qualité et sécurité sont intimement liés dans ce domaine.

Une dernière astuce importante pour vos time-lapses : utilisez des réglages manuels et en particulier pour la balance des blancs afin d'éviter un effet de scintillement sur la vidéo finale que vous génèrerez avec **`avconv -r 24 -i capture%05d.jpg video.mp4`** (paquet **`libav-tools`**).

Un autre exemple intéressant est le *bracketing*. C'est une technique de photo numérique permettant d'obtenir plusieurs clichés d'une scène en faisant varier un réglage au choix. L'une des utilisations de cette technique consiste à jouer sur la sensibilité ou le temps d'exposition et ainsi pouvoir, avec un logiciel de retouche photo, combiner les clichés pour créer une image contenant à la fois les éléments qui pourraient être sous-exposés sur certaines photos et sur-exposés sur d'autres.

Là, ce n'est pas aussi simple que pour le time-lapse, mais c'est relativement compréhensible :

```
$ gphoto2 --set-config-value shutterspeed=0.6 --capture-image-and-download \
--set-config-value shutterspeed=1/8 --capture-image-and-download \
--set-config-value shutterspeed=1/30 --capture-image-and-download \
--set-config-value shutterspeed=1/80 --capture-image-and-download \
--set-config-value shutterspeed=1/400 --capture-image-and-download \
--set-config-value shutterspeed=1/1000 --capture-image-and-download \
--set-config-value shutterspeed=1/3200 --capture-image-and-download \
--filename capture%05n.jpg
```

En une seule longue commande, nous modifions les réglages du temps d'exposition (par leurs valeurs) et provoquons plusieurs prises de vue. Comme il s'agit d'une seule « session » de l'utilitaire, nous pouvons faire usage de l'option **`--filename`** pour qu'il nomme tout seul les fichiers obtenus lors de la récupération. Nous obtenons alors les fichiers **`capture00001.jpg`** à **`capture00007.jpg`** ayant tous un temps d'exposition différent :

```
$ exif *.jpg | grep "Temps d'exposition"
Temps d'exposition  |1/2 sec.
Temps d'exposition  |1/8 sec.
Temps d'exposition  |1/30 sec.
Temps d'exposition  |1/80 sec.
Temps d'exposition  |1/400 sec.
Temps d'exposition  |1/1000 sec.
Temps d'exposition  |1/3200 sec.
```

La commande **`exif`** permet de lister les métadonnées d'un JPEG. Celle-ci s'installe sous la forme d'un paquet du même nom avec **`apt-get`**. Il ne vous restera ensuite plus qu'à combiner les clichés avec votre logiciel préféré.

POUR CONCLURE

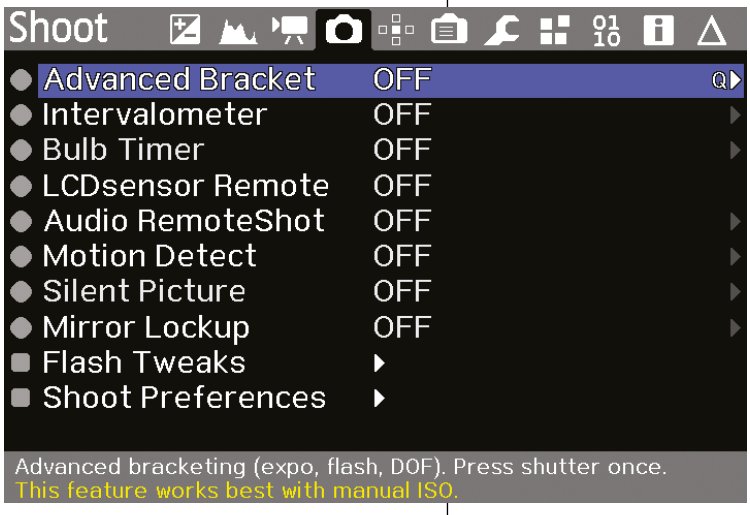
En cas de problème ou de fonctionnement étrange, il est recommandé de jeter un œil à la page **http://gphoto.org/news** avant de dire, demander ou de faire quoi que ce soit. gPhoto est un outil qui ne date pas d'hier, mais l'existence d'un bug pouvant impacter un appareil récent est toujours possible (c'est un développement perpétuel). L'exemple parfait concerne justement mon EOS 700D qui parfois ne répond plus après une suite de manipulations (réglages répétés de la vitesse d'obturation). Il s'avère que c'est là quelque chose de corrigé avec libgphoto2 2.5.6 et la page précise « *Canon EOS: crash on 700D fixed* » (« Canon EOS: crash sur 700D corrigé »). La version disponible dans Raspbian

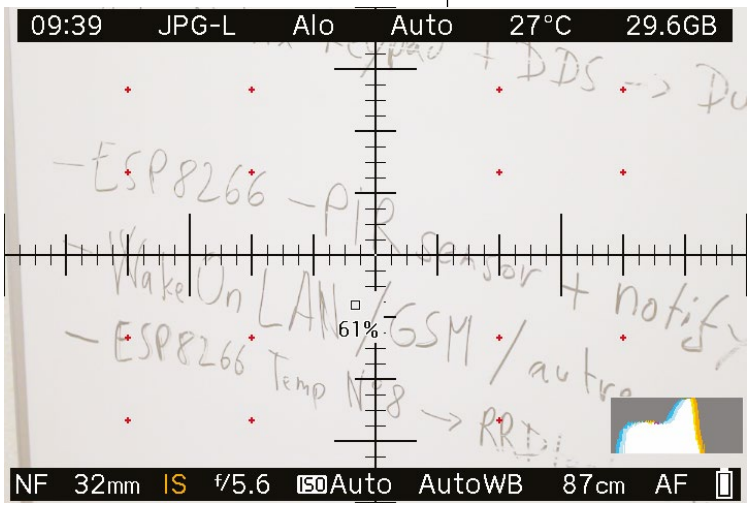
étant la 2.5.4, il n'y a que peu de solutions possibles : patienter le temps de voir venir une mise à jour, ou recompiler libgphoto2 et gPhoto à partir des sources Debian comme nous l'avions fait dans *Hackable n°9* pour obtenir une version récente de RTL power sous forme de paquet. Une autre solution pour obtenir des paquets plus récents est d'ajouter un dépôt dans **/etc/apt/sources.list** et en particulier la ligne :

```
deb http://http.us.debian.org/debian/ testing non-free contrib main
```

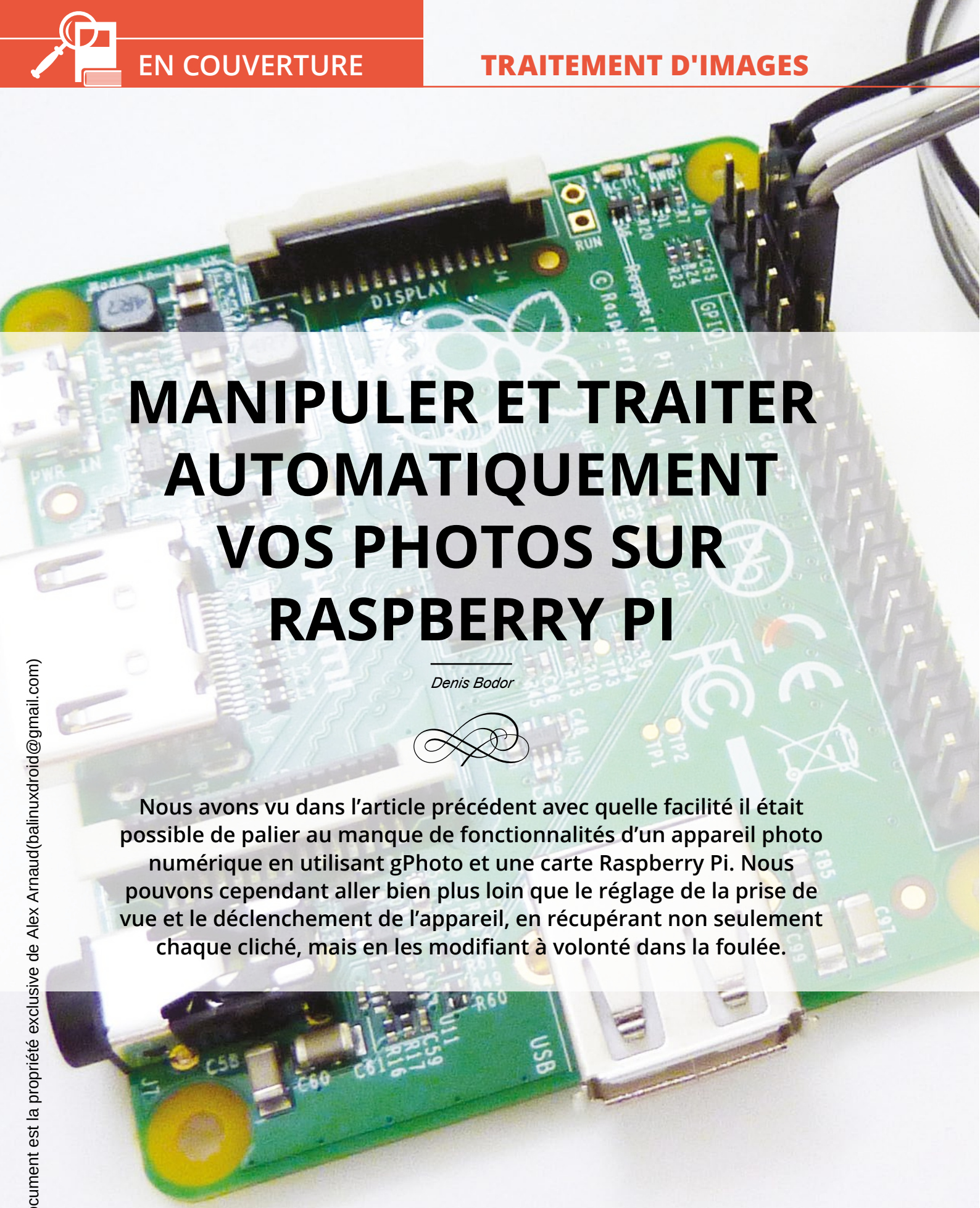
Ceci vous donnera accès à la version *testing* de la prochaine version de la distribution et donc à des outils et applications plus récents, mais en phase de test (et donc considérés comme non suffisamment stables pour constituer une version finale de Raspbian pour l'instant). Là, vous aurez accès à la version 2.5.10 de gPhoto en plus de la 2.5.4, mais attention, ce faisant vous commencerez à mélanger des paquets de la distribution stable et ceux de testing sur votre système.

Une fois la configuration en place et gPhoto bien en main, le panel de possibilités est énorme. Le fait d'utiliser une Raspberry Pi nous permet également de voir encore plus loin en utilisant par exemple un détecteur de mouvements pour déclencher une prise de vue ou encore de contrôler le déplacement de l'appareil sur un rail ou une barre de travelling motorisée pour ajouter un peu de mouvement aux time-lapses. Nous reviendrons certainement sur le sujet à un moment pour parler de tout ça... **DB**





Magic Lantern est un applicatif qui s'ajoute au logiciel interne (firmware) des appareils Canon EOS et fournit une myriade de fonctionnalités supplémentaires en fonction du modèle d'appareil, comme le bracketing, le bulb timer, un intervallomètre de précision, déclenchement sur mise au point (pas pour le 650D/700D), contrôle précis des FPS pour les vidéos, etc. Si vous avez un EOS compatible, cela vaut vraiment la peine d'essayer et cela complètera parfaitement une utilisation avec la Raspberry Pi.



MANIPULER ET TRAITER AUTOMATIQUEMENT VOS PHOTOS SUR RASPBERRY PI

Denis Bodor

Nous avons vu dans l'article précédent avec quelle facilité il était possible de palier au manque de fonctionnalités d'un appareil photo numérique en utilisant gPhoto et une carte Raspberry Pi. Nous pouvons cependant aller bien plus loin que le réglage de la prise de vue et le déclenchement de l'appareil, en récupérant non seulement chaque cliché, mais en les modifiant à volonté dans la foulée.

Une partie des explications qui vont suivre sont applicables non seulement à l'utilisation d'une carte Raspberry Pi (ou d'un PC) pour déclencher et obtenir des clichés depuis un appareil photo numérique (APN), mais également à n'importe quel fichier graphique d'ores et déjà présent sur vos supports de stockage ou même obtenu par d'autres moyens (capture, téléchargement, etc.). Ce que je vous propose de découvrir ici est le monde fantastique de la retouche et manipulation d'images automatisée.

Si vous avez l'habitude de travailler avec des données graphiques, il est très probable que vous fassiez usage d'un logiciel comme Photoshop ou The Gimp. Ce type d'applications repose sur l'interactivité, vous modifiez l'image au fur et à mesure de vos besoins et, une fois le résultat attendu obtenu, vous enregistrez votre travail. Ces logiciels permettent souvent d'automatiser certaines tâches, comme changer la taille d'un groupe de photos, via des scripts qui sont généralement conçus par vous-même ou des tiers et lancés par l'application elle-même.

Ici, nous allons utiliser des outils qui, à la base, ne sont pas du tout conçus pour une utilisation interactive, mais uniquement pour le traitement par lot (*batch* en anglais) et ce, en ligne de commandes. Ceci peut paraître effrayant au premier abord, mais cela est bien plus simple dès lors qu'on prend le temps d'en comprendre la logique. La récompense à la clé pour vos efforts sera la possibilité pour vous d'automatiser intégralement, non seulement les prises de vue, mais également n'importe quelle modification sur le ou les clichés obtenus. Je ne parle pas simplement ici de déclencher manuellement l'exécution d'un script pour traiter un lot d'images, mais bel et bien de laisser la Raspberry Pi faire tout le travail à votre place.

1. IMAGEMAGICK

Peu semblent le savoir, mais ImageMagick est initialement un programme développé au sein de la société DuPont. En 1987, le développeur John Cristy reçoit une demande de la part de son supérieur afin de trouver une solution permettant d'afficher des images de 16 millions de couleurs (24 bits) sur un écran 256 couleurs. Il se tourne alors vers Usenet (les newsgroups) pour chercher de l'aide et Paul Raveling de l'université de Californie du Sud (USC ISI) lui indique un code tout prêt disponible sur les serveurs de l'université, permettant de faire la conversion 24 bits vers 256 couleurs (8 bis indexés). Dès lors, John n'a de cesse d'améliorer le code qu'il développe sur cette base, au fil des demandes des chercheurs et biologistes de DuPont, puis fini par rendre public le résultat sur Usenet en août 1990. Considérant cela comme un juste retour des choses, John reçoit l'aval de la direction de DuPont et ImageMagick voit officiellement le jour sous la forme d'un logiciel libre.

Depuis, DuPont a transféré les droits sur le logiciel à la fondation à but non lucratif *ImageMagick Studio LLC* et une énorme communauté de développeurs et d'utilisateurs s'est développée. ImageMagick est aujourd'hui non seulement une suite d'outils très stables et matures, mais également une bibliothèque utilisée dans de nombreux autres projets.

Pour installer ImageMagick sur votre Pi, il vous suffira d'utiliser la commande `sudo apt-get install imagemagick` et vous aurez alors accès à une dizaine de programmes permettant toutes sortes d'opérations :

- `identify` permet d'avoir des informations précises sur une image (format, taille, métadonnées, etc.) ;
- `display` affiche l'image avec une interface sommaire ;
- `convert` convertit entre plusieurs formats d'images tout en permettant toutes sortes de modifications (redimensionnement, découpage, marquage, flous, tracés, etc.) ;

- **import** permet de faire des captures d'écrans ;
- **compare** compare mathématiquement et visuellement des images pour mettre en évidence les différences ;
- **mogrify** modifie une image à la manière de **convert**, mais en travaillant directement sur l'original ;
- **animate** affiche une animation composée d'une image (GIF par exemple) ou d'une séquence d'images ;
- **composite** permet de composer des images en en mélangeant plusieurs autres ;
- **conjure** interprète et exécute un script écrit dans le langage propre à ImageMagick, le MSL ou *Magick Scripting Language* ;
- **stream** permet de traiter les pixels d'une image ou d'une partie d'une image sous la forme d'un flux de données (très pratique pour des images géantes) ;
- **montage** regroupe des images pour former une nouvelle image comme un patchwork.

Nous n'allons pas ici traiter de tous ces outils, ceci demanderait tout un hors-série, sinon plusieurs, tant ce qu'il est possible de faire est incroyablement vaste. Moi-même je n'utilise que rarement toutes ces commandes et m'en tiens, dans 90% des cas, à **identify**, **display**, **import** et surtout **convert**.

2. POUR SE FAIRE LA MAIN

L'outil **convert** vous permettra de faire énormément de choses grâce à ces quelques 230 options, mais seules quelques-unes sont fréquemment utiles. Mais nous allons commencer par quelque chose de plus simple, en obtenant tout d'abord des informations sur un fichier :

```
% identify image1.jpg
image1.jpg JPEG 5184x3456 5184x3456+0+0 8-bit
DirectClass 6.587MB 0.000u 0:00.000
```

identify utilisé en passant un simple nom de fichier va tenter d'analyser l'image et vous afficher les informations les plus intéressantes. Ici, nous voyons qu'il s'agit d'un format JPEG et que l'image fait 5184 par 3456 pixels. Vous pouvez également utiliser l'option **-verbose** pour obtenir une liste complète de caractéristiques allant du format au nombre de couleurs en passant par l'espace colorimétrique, le niveau de qualité de la compression, la résolution en DPI, la moyenne de la valeur des pixels, etc.

Notre image fait donc 5184 par 3456 et nous allons commencer par l'utiliser pour produire une image de plus petite taille avec :

```
% convert image1.jpg -geometry 1000 image2.jpg

% identify image2.jpg
image2.jpg JPEG 1000x667 1000x667+0+0 8-bit
DirectClass 521KB 0.000u 0:00.000
```

L'option **-geometry** de **convert** nous permet de spécifier une nouvelle taille pour produire le fichier **image2.jpg**. Ici, l'ordre des options importe peu, mais ce ne sera pas forcément le cas

lors de l'utilisation d'options en plus grand nombre. Il est bon de prendre l'habitude de spécifier les arguments sur la ligne de commandes à la manière d'une phrase : « converti image1.jpg avec la géométrie 1000 pour créer image2.jpg ».

Notez que l'image résultante a une dimension de 1000 par 667 pixels. **convert** s'est, en effet, débrouillé seul pour déterminer que la valeur passée en argument de **-geometry** était la largeur de l'image souhaitée et a calculé proportionnellement sa hauteur. Il fait cela tellement bien que si nous spécifions :

```
% convert image1.jpg -geometry 1000x1000 image3.jpg
```

nous obtenons :

```
% identify image3.jpg
image3.jpg JPEG 1000x667 1000x667+0+0 8-bit
DirectClass 521KB 0.000u 0:00.000
```

L'intégrité de l'image est un point important dans ImageMagick et le fait de spécifier une taille qui ne correspond pas au bon ratio hauteur/largeur pourrait produire un résultat non attendu (avec une perte de données). Si vous voulez vraiment que l'outil fasse strictement ce que vous lui demandez, il faut ajouter un **!** pour forcer l'opération :

```
% convert image1.jpg -geometry 1000x1000!  image4.jpg

% identify image4.jpg
image4.jpg JPEG 1000x1000 1000x1000+0+0 8-bit
DirectClass 720KB 0.000u 0:00.000
```

Bien entendu, l'image s'en trouve déformée, mais c'est vous qui avez insisté... Si vous souhaitez que la valeur que vous passez soit la hauteur et non la largeur et que **convert** respecte de la même manière le ratio original, il vous suffit de spécifier la seconde dimension. Ainsi, **-geometry x1000** produira une image avec une hauteur de 1000 pixels (le x1000 étant la partie droite d'une syntaxe « largeur**x**hauteur »).

Enfin, autre valeur intéressante, vous pouvez également spécifier une proportion. **-geometry 10%** va alors produire une image de 10% de sa taille originale.

Une autre manipulation courante consiste non pas à changer la taille d'une image, mais son format. **convert** fait cela très facilement puisqu'il vous suffit de spécifier l'extension adéquate du fichier cible. Exemple :

```
% identify image2.jpg
image2.jpg JPEG 1000x667 1000x667+0+0 8-bit
DirectClass 521KB 0.000u 0:00.000

% convert image2.jpg image2.png

% identify image2.png
image2.png PNG 1000x667 1000x667+0+0 8-bit
DirectClass 1.698MB 0.000u 0:00.000
```

En dehors de l'extension du fichier cible, il vous est possible de spécifier un format avec son nom suivi d'un double point. La commande `convert image2.jpg png:image2.toto` produira le même résultat que précédemment, tout en spécifiant le format qui ne peut être déduit par `convert` grâce à l'extension donnée.

Beaucoup d'autres modifications peuvent être apportées aux images avec `convert` :

- inverser les couleurs avec `-negate` ;
- convertir un espace de couleur à un autre comme sRGB vers niveau de gris avec `-colorspace Gray` ;
- réduire le nombre de couleurs avec par exemple `-colors 256` ;
- tourner une image d'un certain nombre de degrés avec `-rotate` ;
- appliquer des flous avec `-radial-blur`, `-blur` ou `-gaussian-blur` ;
- découper un morceau d'image avec `-crop` ;
- etc.

Je vous recommande fortement la lecture de la page **http://www.imagemagick.org/Usage/**. Même si elle est en anglais, les exemples donnés sont relativement visuels et vous permettront au moins de vous faire une idée des différentes options et des conséquences de leur utilisation.

3. UN PEU DE SCRIPT SHELL

Pour l'heure, nous n'avons travaillé qu'avec un seul fichier image et bien qu'ImageMagick dispose d'un langage qui lui est propre, c'est généralement au shell qu'on confie l'interprétation de scripts. Si vous êtes débutant sous GNU/Linux grâce à la Raspberry Pi, vous ne le savez peut-être pas, mais la ligne de commandes que vous utilisez régulièrement est un shell, et plus exactement le shell Bash.

Un shell ou interpréteur de commandes, est le programme qui traduit ce que vous tapez au clavier et détermine ce qu'il faut en faire. Ceci est ce qu'on appelle une utilisation interactive du shell, mais il est également possible de réunir un lot de commandes dans un fichier et de demander au shell de l'interpréter : c'est un script. Les commandes dans ce script peuvent être des programmes, des instructions propres au shell lui-même ou encore des boucles, des initialisations de variables, des conditions... Ainsi en plus d'être votre interpréteur de commandes, le shell Bash est également un langage de programmation, ou plus exactement, un langage de script.

Ce n'est pas tout. Même si l'on trouve généralement ce type d'utilisation dans un script, ou en d'autres termes dans un fichier, il est également possible d'utiliser ces fonctionnalités directement en ligne de commandes pour, par exemple, lancer un appel à un programme à l'intérieur d'une boucle.

Voici un exemple simple. Soit une série de fichiers dans le répertoire courant :

```
% ls
tizen_S2_arriere.jpg  tizen_S2_ip.jpg
tizen_S2.jpg  tizen_Z3_arriere.jpg  tizen_Z3.jpg
```

Nous souhaitons, pour chacun de ces fichiers, afficher son type à l'aide de la commande `file`, ainsi :

```
% file tizen_S2.jpg
tizen_S2.jpg: JPEG image data, JFIF standard 1.01
```

Vous pouvez répéter la commande autant de fois qu'il y a de fichiers, ou vous pouvez demander au shell de boucler sur la liste :

```
% for i in *.jpg; do file $i; done
tizen_S2_arriere.jpg: JPEG image data, JFIF standard 1.01
tizen_S2_ip.jpg: JPEG image data, JFIF standard 1.01
tizen_S2.jpg: JPEG image data, JFIF standard 1.01
tizen_Z3_arriere.jpg: JPEG image data, JFIF standard 1.01
tizen_Z3.jpg: JPEG image data, JFIF standard 1.01
```

La syntaxe utilisée, ici sur une ligne, est plus facile à comprendre présentée ainsi (comme dans un script) :

HACKABLE MAGAZINE

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS !

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

www.ed-diamond.com

LES COUPLAGES PAR SUPPORT :

VERSION
PAPIER

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

VERSION
PDF

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

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

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

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

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

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

HACKABLE MAGAZINE

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

Vos remarques :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC HACKABLE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Prix en Euros / France Métropolitaine

Offre	SUPPORT / ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF 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.

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

```
for i in *.jpg
do
  file $i
done
```

Les **;** utilisés sur une seule ligne permettent de marquer la fin de chaque instruction (comme en C), mais ils ne sont pas nécessaires lorsqu'on utilise plusieurs lignes. Nous avons ici une boucle **for** plaçant dans la variable **i** chaque élément de la liste ***.jpg** (les fichiers avec une extension **.jpg** du répertoire courant). Dans le corps de la boucle (**do**), la variable référencée **$i** contiendra donc, à chaque tour, un nom de fichier qu'il sera possible d'utiliser pour une commande, ici **file**. La portée de la boucle elle-même est définie par les directives **do** et **done**.

Bien entendu, nous pouvons utiliser n'importe quelle commande ou suite de commandes dans notre boucle, y compris celles fournies par ImageMagick :

```
% for i in *.jpg; do identify $i; done
tizen_S2_arriere.jpg JPEG 3695x3359 3695x3359+0+0 8-bit DirectClass 4.19MB 0.370u 0:00.380
tizen_S2_ip.jpg JPEG 2596x2689 2596x2689+0+0 8-bit DirectClass 2.02MB 0.180u 0:00.190
tizen_S2.jpg JPEG 3613x3019 3613x3019+0+0 8-bit DirectClass 3.968MB 0.340u 0:00.329
tizen_Z3_arriere.jpg JPEG 3680x3234 3680x3234+0+0 8-bit DirectClass 2.966MB 0.300u 0:00.300
tizen_Z3.jpg JPEG 3547x3195 3547x3195+0+0 8-bit DirectClass 2.925MB 0.280u 0:00.279
```

Imaginez maintenant cette même série de fichiers placée dans un répertoire dans lequel nous ajoutons un sous-répertoire **petit/**. En utilisant un éditeur de texte (Vim, Nano, Emacs, etc.), nous pouvons écrire le script suivant :

```
#!/bin/bash

for i in *.jpg
do
  convert -geometry 500 $i petit/$i
done
```

Notre boucle est identique à la précédente, mais nous utilisons **convert** pour changer la taille de chaque image et placer le résultat dans le sous-répertoire **petit/**, avec le même nom de fichier. La première ligne du script est appelée un *shebang* (**#!**) et permet au système de savoir qu'il s'agit d'un script et qu'il doit être interprété par un programme particulier (ici **/bin/bash**, le shell Bash). Ceci nous permet d'enregistrer, par exemple, le script dans un fichier **monscript** puis de le rendre exécutable avec **chmod +x monscript**. Ainsi, nous pourrons le lancer avec **./monscript** et le système l'interprètera avec **/bin/bash** sans que nous ayons à faire **sh monscript** en ligne de commandes.

L'exécution de ce script va boucler sur tous les fichiers **.jpg** du répertoire courant, les redimensionner à la taille demandée et enregistrer le résultat dans **petit/**. Nous obtenons donc une version réduite des images de tout un répertoire en une seule commande.

Bien entendu, ceci fonctionne avec n'importe quelle transformation mise à disposition par **convert** et nous pouvons même convertir des fichiers d'un format à un autre. Ceci demande une petite astuce puisque **convert** travaille par défaut en fonction de l'extension du fichier et, même si nous utilisons par exemple **png:** nous nous retrouverions avec des données PNG dans un fichier avec une extension **.jpg** (certains logiciels n'aiment pas du tout ça).

Pour manipuler le nom de fichier se trouvant dans **$i** nous devons faire appel à **basename**, une commande spécialement conçue pour retirer un nom de répertoire et une extension d'un nom de fichier. La syntaxe est la suivante : **basename fichier.ext ext** qui aura pour effet de retirer « ext » et retourner le nom « fichier. ». En utilisant l'apostrophe inverse (AltGr+7), nous pouvons utiliser ce résultat comme un argument/élément d'une commande.

Ainsi, si nous faisons **convert fichier.jpg `basename fichier.jpg jpg`png**, **basename** va retourner « fichier. », texte que nous réutilisons en le complétant de « png » pour former « fichier.png » et ainsi former la commande **convert fichier.jpg fichier.png**. En appliquant cette syntaxe et cette commande **basename** non pas à un nom de fichier statique, mais à celui contenu dans une variable, nous obtenons **convert $i `basename $i jpg`png**. Chose que nous pouvons alors facilement adapter dans un script :

```
#!/bin/bash

for i in *.jpg
do
  echo "Convertion $i"
  convert -geometry 500 $i petit/`basename $i jpg`png
done
```

Ici, nous utilisons donc **convert** pour redimensionner tous les fichiers du répertoire courant, un par un, en plaçant le résultat dans le répertoire **petit/** et en changeant l'extension en **.png** et donc le format utilisé par **convert**. Nous en profitons pour ajouter un affichage permettant de suivre la progression, les outils d'ImageMagick étant silencieux par défaut (typique de la philosophie UNIX : si un programme n'a rien à signaler ou d'avertissement à donner, il doit rester silencieux). L'exécution du script nous donne donc :

```
% ./script.sh
Convertion tizen_S2_arriere.jpg
Convertion tizen_S2_ip.jpg
Convertion tizen_S2.jpg
Convertion tizen_Z3_arriere.jpg
Convertion tizen_Z3.jpg

% ls petit/
tizen_S2_arriere.png  tizen_S2_ip.png  tizen_S2.png
tizen_Z3_arriere.png  tizen_Z3.png

% identify petit/tizen_Z3.png
petit/tizen_Z3.png PNG 500x450 500x450+0+0 8-bit
  DirectClass 400KB 0.000u 0:00.000
```

Nous avons bel et bien transformé tous nos fichiers JPEG du répertoire courant en des versions plus petites, placées dans **petit/** et au format PNG. Bingo !

Poussons encore davantage la personnalisation de ce script. Imaginez maintenant que vous ne sachiez pas à l'avance dans quel format ni dans quelle taille vous souhaiterez convertir vos fichiers. Ces informations peuvent être passées au script lors de son invocation. Modifiez le script en ajoutant ceci après la ligne *shebang* :

```
if [ -z "$TAILLE" ]
then
  TAILLE=500
fi

if [ -z "$EXT" ]
then
  EXT="png"
fi
```

Nous avons ici deux conditions **if** qui testent si les variables spécifiées (**$TAILLE** et **$EXT**) ont une taille nulle. Si tel est le cas, c'est que les variables ne sont pas initialisées et nous leur affectons donc une valeur par défaut, respectivement 500 et « png ».

Nous modifions également le corps de la boucle ainsi :

```
  echo "Convertion $i en $TAILLE et format $EXT"
  convert -geometry $TAILLE $i petit/'basename $i jpg'$EXT
```

REMARQUE

Notez qu'il est relativement rare d'utiliser cette technique pour passer des arguments à un script. J'ai choisi ici cette méthode de façon délibérée pour enchaîner sur l'utilisation de gPhoto et sa manière d'utiliser des scripts.

En temps normal, on utilisera plutôt les variables `$1`, `$2`... désignant respectivement le premier argument, le second argument... passés après le nom du script (qui est contenu dans `$0`). Le test et l'initialisation prendraient alors cette forme :

```
if [ -z "$1" ]
then
  TAILLE=500
else
  TAILLE=$1
fi

if [ -z "$2" ]
then
  EXT="png"
else
  EXT=$2
fi
```

Et l'appel du script serait : `./script.sh 850 bmp` pour une conversion avec une largeur de 850 pixels et un format cible BMP.

Nous utilisons les valeurs dans **$TAILLE** et **$EXT** pour afficher la progression et comme arguments pour l'appel à **convert**. À présent, si nous lançons le script comme précédemment, nous obtenons :

```
% ./script.sh
Convertion tizen_S2_arriere.jpg en 500 et format png
Convertion tizen_S2_ip.jpg en 500 et format png
Convertion tizen_S2.jpg en 500 et format png
Convertion tizen_Z3_arriere.jpg en 500 et format png
Convertion tizen_Z3.jpg en 500 et format png
```

Les variables ne sont pas initialisées et les valeurs par défaut, 500 et « png » sont alors utilisées. Mais si nous lançons la commande ainsi :

```
% TAILLE=850 EXT=bmp ./script.sh
Convertion tizen_S2_arriere.jpg en 850 et format bmp
Convertion tizen_S2_ip.jpg en 850 et format bmp
Convertion tizen_S2.jpg en 850 et format bmp
Convertion tizen_Z3_arriere.jpg en 850 et format bmp
Convertion tizen_Z3.jpg en 850 et format bmp
```

Les variables renseignées en début de ligne, qui sont appelées **variables d'environnements**, existent dans le script et n'ont pas une taille nulle. Elles sont alors utilisées pour les commandes dans la boucle. Nous pouvons ainsi choisir, au moment de l'invocation du script, la taille et le format cible.

4. L'OPTION DES SCRIPTS AVEC GPHOTO

Nous avons maintenant quelques connaissances de base en script shell et en utilisation des outils ImageMagick. Il est donc temps de lier cela avec gPhoto et en particulier son option **--hook-script**. Celle-ci permet d'associer l'exécution d'un script aux différentes phases de traitement des actions de gPhoto. Le script doit se conformer à un certain fonctionnement, car gPhoto va lui passer une ou deux variables d'environnement, exactement comme nous l'avons fait pour **$TAILLE** et **$EXT** à l'instant.

Les variables passées sont **ACTION** et **ARGUMENT** avec les valeurs possibles pouvant être les suivantes :

- **$ACTION** = "init" : gPhoto vient d'être lancé, mais n'a encore accompli aucune action. Si le script arrête son exécution en retournant une valeur différente de zéro (avec **exit 1** par exemple), gPhoto n'ira pas plus loin et abandonnera l'opération. Ceci peut être très intéressant pour, par exemple, tester la présence d'un répertoire de destination pour les images ou s'assurer d'un espace disque suffisant avant d'aller plus loin.
- **$ACTION** = "start" : gPhoto a fini d'analyser sa ligne de commandes et est sur le point d'exécuter les ordres donnés.
- **$ACTION** = "download" : gPhoto vient de télécharger un fichier depuis l'appareil photo et l'a enregistré sous un nom qu'il aura placé dans la variable **$ARGUMENT**. C'est ici que nous pourrons alors déclencher différentes opérations sur l'image qui vient d'être obtenue.
- **$ACTION** = "stop" : gPhoto a fini son travail et est sur le point de quitter. C'est le moment de faire un brin de ménage ou de provoquer une opération concernant l'ensemble des fichiers qui ont été téléchargés (comme créer une archive et/ou l'envoyer par mail, par exemple).

En spécifiant l'option **--hook-script** suivi d'un nom de script, gPhoto va donc l'appeler en initialisant ses variables en fonction de la progression de son travail. Notre objectif sera donc d'écrire un script qui, en fonction du contenu de la variable d'environnement **$ACTION**, procèdera à différentes opérations. Nous pourrions utiliser ici des **if** pour tester le contenu de **$ACTION**, mais une condition se prête bien davantage à ce genre d'utilisation : un switch/case :

```
#!/bin/bash

case "$ACTION" in
"init")
  echo ">>> Initialisation"
  ;;
"start")
  echo ">>> Execution des commandes"
  ;;
"download")
  echo ">>> $ARGUMENT a été telechargé."
  ;;
"stop")
  echo ">>> Traitement gPhoto terminé"
  ;;
esac
```

case va évaluer le contenu de **$ACTION** et appliquer automatiquement les instructions placées dans la portée adéquate, délimitée par le contenu attendu entre guillemets et **;;**. La fin du switch/case est bornée par un **esac** (**case** à l'envers, tout comme **fi** marque la fin d'un **if**). Nous avons donc ici différents affichages de texte avec la commande **echo** en fonction du contenu de **$ACTION**.

Il ne nous reste alors plus qu'à tester le script avec :

```
% gphoto2 --capture-image-and-download --hook-script ./script.sh
>>> Initialisation
>>> Execution des commandes
Le nouveau fichier est à l'emplacement /capt0000.jpg de l'appareil
Enregistrement du fichier en capt0000.jpg
>>> capt0000.jpg a été telechargé.
Effacement du fichier /capt0000.jpg de l'appareil
>>> Traitement gPhoto terminé
```

Pour chaque équivalence du contenu de **$ACTION**, notre script nous informe de la progression ainsi que du nom du fichier obtenu. Notez cependant que la sortie du script se mélange avec les messages provenant directement de gPhoto. Nous pouvons réduire la présence de ce texte parasite en ajoutant l'option **--quiet** à notre ligne de commandes :

```
% gphoto2 --quiet --capture-image-and-download --hook-script ./script.sh
>>> Initialisation
>>> Execution des commandes
/capt0000.jpg
>>> capt0000.jpg a été telechargé.
>>> Traitement gPhoto terminé
```

Notez bien que ce fonctionnement est valable pour une « session » ou une exécution complète de gPhoto. Ceci signifie que si nous lui demandons de prendre une série de clichés, le script nous affichera bien une seule initialisation et une exécution de commandes, mais autant de mentions de fichiers téléchargés qu'il y aura de clichés pris. Ceci est donc particulièrement bien adapté à la prise de photos à intervalles réguliers. Pour asseoir tout cela, nous allons immédiatement mettre en pratique cette théorie...

5. UN CAS PRATIQUE : HORODATER LES CLICHÉS D'UN TIME LAPSE

Notre objectif ici sera d'utiliser gPhoto pour prendre des clichés avec un intervalle de 10s (cette valeur n'est pas importante). Après chaque photo, nous voulons que l'image récupérée soit automatiquement redimensionnée avec une largeur de 800 pixels, placée dans un répertoire particulier et marquée de la date et l'heure de la prise de vue.

La série d'images pourra être ensuite convertie en vidéo avec `avconv` comme nous l'avons vu dans l'article précédent et dans l'article sur le module raspicam dans le numéro de juillet/août.

L'utilisation d'un script avec l'option `--hook-script` étant à présent maîtrisée, vous l'avez compris, tout ce que nous avons à faire est de spécifier la bonne commande pour l'action « download ». L'outil à mettre en œuvre est encore une fois `convert` et nous connaissons déjà l'option `-geometry` permettant de redimensionner l'image.

Nous avons besoin de deux choses supplémentaires :

- un moyen d'écrire du texte sur chaque image, de façon lisible et propre ;
- extraire la date et l'heure de la prise du cliché depuis les fichiers.

`convert` fournit l'option `-annotate` permettant d'ajouter un test tout en précisant son inclinaison et son décalage par rapport à un point de référence.

L'inclinaison ici ne nous intéresse pas, mais le fait de placer le texte à une certaine distance d'un bord ou d'un coin nous sera bien pratique. Nous commencerons donc par définir ce point de référence via l'option `-gravity` pouvant prendre en argument :

- `NorthWest` : en haut à gauche,
- `North` : en haut au centre,
- `NorthEast` : en haut à droite,
- `West` : à gauche au centre,
- `Center` : au centre de l'image,
- `East` : à droite au centre,
- `SouthWest` : en bas à gauche,
- `South` : en bas au centre,
- `SouthEast` : en bas à droite.

Nos options seront alors `-gravity SouthEast -annotate +10+10 "un texte"`, les `+10` signifiant respectivement à 10 pixels sur X (largeur) et Y (hauteur) du bord de l'image. Mais nous voulons également que le texte soit lisible et nous choisissons alors une couleur de remplissage qui sera du blanc avec `-fill white` ainsi qu'une taille de caractères adaptée, `-pointsize 24`.

Afin de nous assurer que le texte en blanc soit bien visible quelle que soit l'image, nous allons placer à l'arrière un fond translucide en spécifiant l'option `-undercolor` avec comme argument une valeur RGBA (Rouge Vert Bleu Alpha) à `#000000B0` (pas de rouge, pas de vert, pas de bleu et un canal alpha à `B0` sur un maximum de `FF`). La notation utilisée est celle du langage HTML avec, en plus, deux caractères pour la transparence avec `00` pour totalement transparent et `FF` pour une opacité complète.

Notre liste d'options est maintenant : `-geometry 800 -gravity SouthEast -undercolor '#000000B0' -fill white -pointsize 24 -annotate +10+10 "un texte"` nous permettant donc d'avoir : un texte blanc en police 24 points à 10 pixels du bord inférieur droit sur un fond noir translucide. Notez que les options sont traitées dans l'ordre, et de ce fait que l'inscription du texte aura lieu **après** le redimensionnement de l'image.

Il ne nous reste plus qu'à déterminer le texte à utiliser. Un fichier JPEG tel que ceux provenant des appareils photo numériques embarque des métadonnées incluant le type d'appareil, les conditions de prise de vue, parfois des coordonnées GPS, une vignette... mais aussi et surtout la date et l'heure du cliché. Attention, il ne s'agit pas là de la date de création ou de modification du fichier, mais bien celle de la prise de vue.

Pour obtenir les informations embarquées dans le fichier, il vous suffit d'utiliser la commande **exif** provenant du paquet du même nom. Voici quelques informations issues d'un JPEG provenant de mon EOS 700D :

```
% exif capt0000.jpg
--------------------+-------------------------------
Marqueur            |Valeur
--------------------+-------------------------------
Constructeur        |Canon
Modèle              |Canon EOS 700D
Orientation         |Top-left
Date et heure       |2016:08:17 12:02:03
Temps d'exposition  |1/200 sec.
F-Number            |f/9,0
ISO Speed Ratings   |100
Version d'exif      |Version d'exif 2.3
Shutter Speed       |7,62 EV (1/197 sec.)
Ouverture           |6,38 EV (f/9,1)
Longueur focale     |46,0 mm
Mode d'exposition   |Exposition automatique
Balance des blancs  |Balance des blancs automatique
```

Nous ne voulons cependant pas récupérer toutes ces informations, mais une seule. Nous devons donc utiliser l'option **-t** en spécifiant le tag (l'information) qui nous intéresse. Pour obtenir la liste des tags EXIF présents dans un fichier, il nous suffit de faire **exif -l fichier.jpg** et trouver celui qui nous intéresse. Ici, ce sera « *0x9003 Date and Time (Original)* ». Il ne nous reste plus qu'à l'utiliser pour obtenir la métadonnée espérée :

```
% exif -t 0x9003 capt0000.jpg
EXIF entry 'Date and Time (Original)' (0x9003, 'DateTimeOriginal') exists in IFD 'EXIF':
Tag: 0x9003 ('DateTimeOriginal')
  Format: 2 ('ASCII')
  Components: 20
  Size: 20
  Value: 2016:08:17 12:02:03
```

Voilà qui n'est pas tout à fait formaté comme nous l'attendions. Heureusement, **exif** possède un mode beaucoup plus approprié via l'option **-m** ou **--machine-readable** :

```
% exif -m -t 0x9003 capt0000.jpg
2016:08:17 12:02:03
```

En utilisant l'apostrophe inversée comme nous l'avons fait pour **basename**, nous pouvons alors utiliser cette sortie comme texte pour l'option **-annotate** de **convert**. Voici donc notre script final :

```
#!/bin/bash

#exit 1

case "$ACTION" in
"init")
  echo ">>> Initialisation"
  ;;
"start")
  echo ">>> Execution des commandes"
  ;;
"download")
  echo ">>> $ARGUMENT a été téléchargé."
  convert $ARGUMENT -geometry 800 -gravity SouthEast \
    -undercolor '#000000B0' -fill white -pointsize 24 -annotate +10+10 \
    " `exif -m -t 0x9003 $ARGUMENT` " petit/$ARGUMENT
  ;;
"stop")
  echo ">>> Traitement gPhoto terminé"
  echo ">>> `ls petit/*.jpg|wc -l` fichiers traité(s) dans petit/"
  ;;
esac
```

Il vous suffira de l'enregistrer dans un répertoire vide, sous le nom **script.sh** par exemple, avant de vous plier d'un **chmod +x script.sh**, puis de créer un répertoire **petit/** avant de lancer la commande :

```
% gphoto2 --quiet --capture-image-and-download --frames 5 \
  --interval 10 --hook-script ./script.sh
>>> Initialisation
>>> Execution des commandes
/capt0000.jpg
>>> capt0000.jpg a été téléchargé.
/capt0001.jpg
>>> capt0001.jpg a été téléchargé.
/capt0002.jpg
>>> capt0002.jpg a été téléchargé.
/capt0003.jpg
>>> capt0003.jpg a été téléchargé.
/capt0004.jpg
>>> capt0004.jpg a été téléchargé.
>>> Traitement gPhoto terminé
>>> 5 fichiers traité(s) dans petit/
```

gphoto est ici utilisé pour prendre une photo toutes les 10 secondes jusqu'à atteindre un nombre de 5 clichés. Bien entendu, dans une situation réelle, il s'agira plutôt d'une image toutes les 30s, minutes voire 30mn, en fonction de la scène photographiée (coucher de soleil, déplacement des nuages, journée complète, croissance d'une plante, fonte des neiges, évolution du feuillage d'une forêt, etc.).

Le résultat du traitement avec `convert` ressemblera à ceci :

Je vous invite fortement à consulter la page web **http://www.imagemagick.org/Usage** pour trouver des variations intéressantes sur ce thème et vous découvrirez que l'étendue des possibilités est absolument incroyable.

POUR FINIR

Nous pourrions pousser encore plus loin le développement de ce script et ajouter sans cesse de nouvelles fonctionnalités. Créer des déclinaisons dans différents formats de chaque cliché, renommer les fichiers, créer des archives par jour, composer une mosaïque d'images, envoyer les fichiers via une liaison réseau (`scp`), sauvegarder régulièrement sur clé USB...

Nous pourrions également rendre le script plus générique en lui faisant prendre des arguments dans un fichier de configuration (avec l'instruction `source`) ou en restructurant entièrement le script de façon à placer toutes les commandes et paramètres dans des variables réunies en début de fichier pour une édition rapide.

La qualité d'un appareil reflex numérique couplé à la souplesse d'une Raspberry Pi, la puissance de gPhoto et la richesse d'ImageMagick, le tout lié ensemble grâce aux fonctionnalités du shell Bash, nous ouvre des horizons inaccessibles aux photographes les plus créatifs. Mais c'est aussi une source d'enrichissement personnel tant au niveau de la masse de connaissances que vous allez acquérir, qu'en termes de qualité humaine comme la patience et la méticulosité. En effet, rien n'est plus frustrant que de mettre en place un tel système, le laisser faire son travail alors que les conditions sont optimales pour, à terme après des jours, voire des semaines de prise de vue, se rendre compte qu'un petit réglage aura gâché toute la collection d'images...

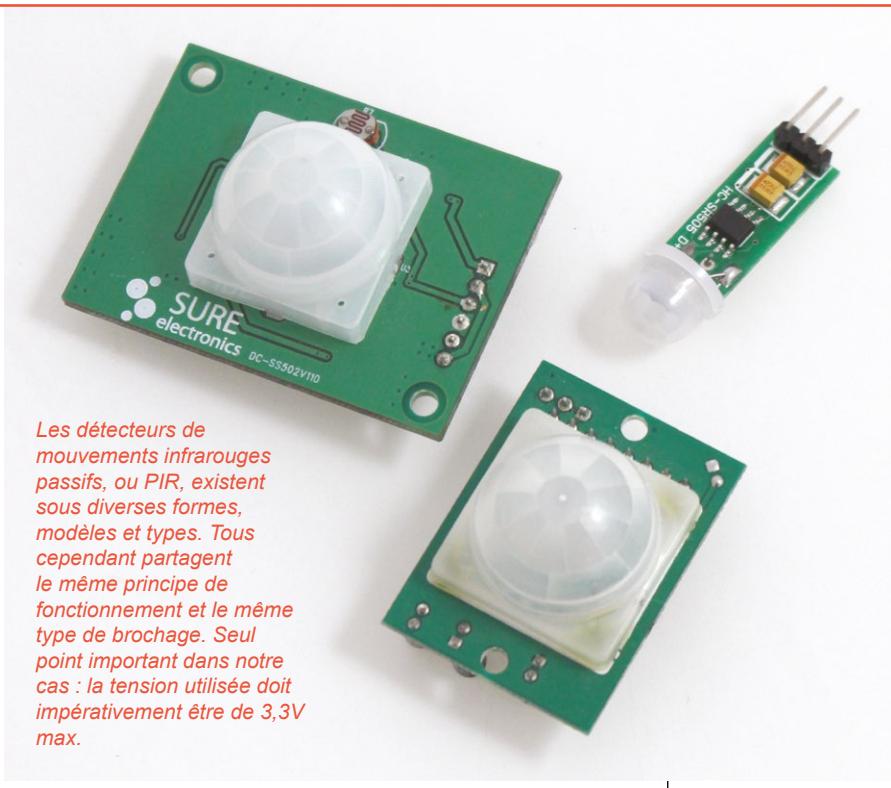
Voici quelques conseils qui, je l'espère vous aideront dans vos expérimentations :

- pensez à l'alimentation de l'appareil et préférez dans la mesure du possible une alimentation secteur et non sur batterie ;
- testez, re-testez et re-testez encore vos scripts, vos commandes et vos options avant de démarrer votre prise de vue définitive ;
- faites des copies et des sauvegardes de toutes vos images, et assurez-vous de ne jamais rien écraser ;
- préférez les réglages manuels de l'appareil photo (balance des blancs, ISO, mise au point, etc.), tout ce qui est automatique est susceptible de varier et de donner des résultats catastrophiques une fois converti en vidéo ;
- assurez-vous, par tous les moyens possibles, que personne ne touche à votre équipement ou ne risque de le déplacer par inadvertance ;
- choisissez le bon intervalle en fonction de la scène photographiée, dans le doute mieux vaut avoir trop de clichés que pas assez ;
- conservez toujours les clichés originaux en haute résolution ;
- organisez-vous et classez vos séries de fichiers par thèmes par date. **DB**

PRENEZ DES CLICHÉS AUTOMATIQUEMENT EN CAS DE DÉTECTION DE MOUVEMENT

Denis Bodor

Nous avons exploré plusieurs possibilités d'utilisation d'une Raspberry Pi connectée à un appareil photo numérique et il y a certainement encore des centaines d'options à couvrir, mais il en reste une qui me parait tout simplement incontournable : déclencher une prise de vue en fonction d'un évènement externe détecté par la Raspberry Pi. Notre exemple ici sera la détection d'un mouvement par un capteur infrarouge.



Les détecteurs de mouvements infrarouges passifs, ou PIR, existent sous diverses formes, modèles et types. Tous cependant partagent le même principe de fonctionnement et le même type de brochage. Seul point important dans notre cas : la tension utilisée doit impérativement être de 3,3V max.

Nous savons à présent piloter un appareil photo numérique depuis la Pi en ligne de commandes grâce à gPhoto et même traiter automatiquement les images ainsi capturées. Nous pouvons faire des séries de clichés en fonction de paramètres temporels, mais qu'en est-il de la possibilité de prendre des clichés en fonction d'une source ponctuelle d'évènement ? Par exemple :

- le déclenchement d'un capteur par contact ;
- la détection d'un bruit ;
- la coupure d'un faisceau lumineux (ou infrarouge) ;
- la détection d'un objet avec un capteur ultrason ;
- ou encore la captation d'un mouvement par un détecteur infrarouge.

Sur papier, le principe est simple, la Raspberry Pi, d'une façon ou d'une autre, détecte quelque chose et ce quelque chose doit provoquer la prise d'un cliché. En pratique, les choses sont plus délicates qu'il n'y paraît, mais la Pi, comme gPhoto, nous offre des solutions parfaitement adaptées.

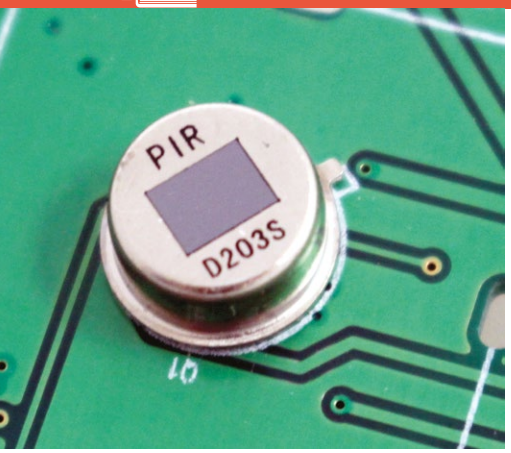
Notre exemple ici consistera en l'utilisation d'un module de détection infrarouge passif ou *Passive InfraRed sensor* en anglais, généralement écourté en *PIR sensor*.

1. LES CAPTEURS PIR

Ce type de capteur est très courant, ils sont généralement utilisés, associés à un système quelconque d'éclairage pour déclencher la mise en route à la moindre détection de mouvement. Leur principe de fonctionnent est relativement simple et se base sur un capteur pyroélectrique constitué d'un matériau créant une différence de potentiels en étant exposé à un changement de température. À l'instar des cristaux piézoélectriques qui génèrent une telle différence de potentiels en cas de déformation mécanique, le capteur a pour objectif de signaler un changement de température et plus généralement l'arrivée d'une source de chaleur comme un corps humain.

Le système est dit « passif », car il n'y a pas d'émission infrarouge de la part du module. Celui-ci ne fait que capter un rayonnement émis par une source, ce qui signifie donc qu'un objet à la même température que son environnement ne déclenchera pas de détection.

Le capteur pyroélectrique se compose généralement de deux éléments connectés en série et avec des polarités opposées. Les deux éléments, percevant chacun la chaleur ambiante, vont tous deux afficher une différence de potentiels identique, mais s'annulant l'une l'autre. Le capteur ne réagira donc pas en cas de variation globale de la température, mais uniquement si l'un des deux éléments perçoit de la chaleur, mais l'autre non. C'est précisément pour cette raison que le capteur est placé derrière une lentille concentrant le rayonnement infrarouge sur l'une ou l'autre partie du capteur. Cette lentille à facette nous parait blanche, mais elle est en réalité transparente aux infrarouges dans la gamme de longueurs d'ondes (entre 8 et 15 micromètres) à laquelle réagit le capteur. Elle sert également de filtre pour le capteur.



Sous la lentille en plastique blanc se cache le capteur pyroélectrique. Celui-ci est généralement trop complexe à utiliser seul (électronique analogique) et s'accompagne d'un circuit intégré simplifiant sa mise en œuvre.

Pour que le système fonctionne, il faut donc mesurer une différence de potentiel aux bornes du capteur, amplifier cette différence et déterminer un seuil minimum indiquant une détection effective. Vous n'aurez pas à faire tout cela par vous-même, car un circuit spécialisé existe et est intégré presque à l'ensemble des modules de détection, qu'il s'agisse de circuits pour hobbyistes ou de produits manufacturés comme des éclairages automatiques. Ce circuit est le BISS001. Très complet, il prend en charge non seulement le capteur, mais également un ensemble de réglages ainsi que le pilotage de relais. Le circuit lui-même est en mesure de fonctionner avec une tension entre +3V et +6V, mais ceci est généralement un paramètre qui découle du module et de son origine.

Une simple recherche sur eBay ou sur Amazon des termes « PIR sensor » et vous aurez droit à une liste incroyablement longue d'offres en tous genres, avec des prix ne dépassant généralement pas 4€ pièce, port offert, et une moyenne aux alentours de 1,5€. Il vous faudra cependant faire très attention aux spécifications techniques données, car nous avons l'intention de faire fonctionner ce module avec une Raspberry Pi. Il ne faut donc en aucun cas que la tension de sortie dépasse 3,3V, l'alimentation du module pouvant être 3,3V ou 5V. Je vous recommande d'ailleurs, à réception du ou des modules que vous commanderez, de vérifier, à l'aide d'un multimètre, la tension existante entre la masse et le signal de sortie en cas de détection, et ce avant toute connexion à une broche en entrée de la Raspberry Pi.

Ces modules proposent entre 3 et 6 broches, mais seules 3 nous intéressent ici :

- la masse ;
- la tension d'alimentation ;
- le signal en sortie généralement libellé OUT, SOUT ou VOUT.

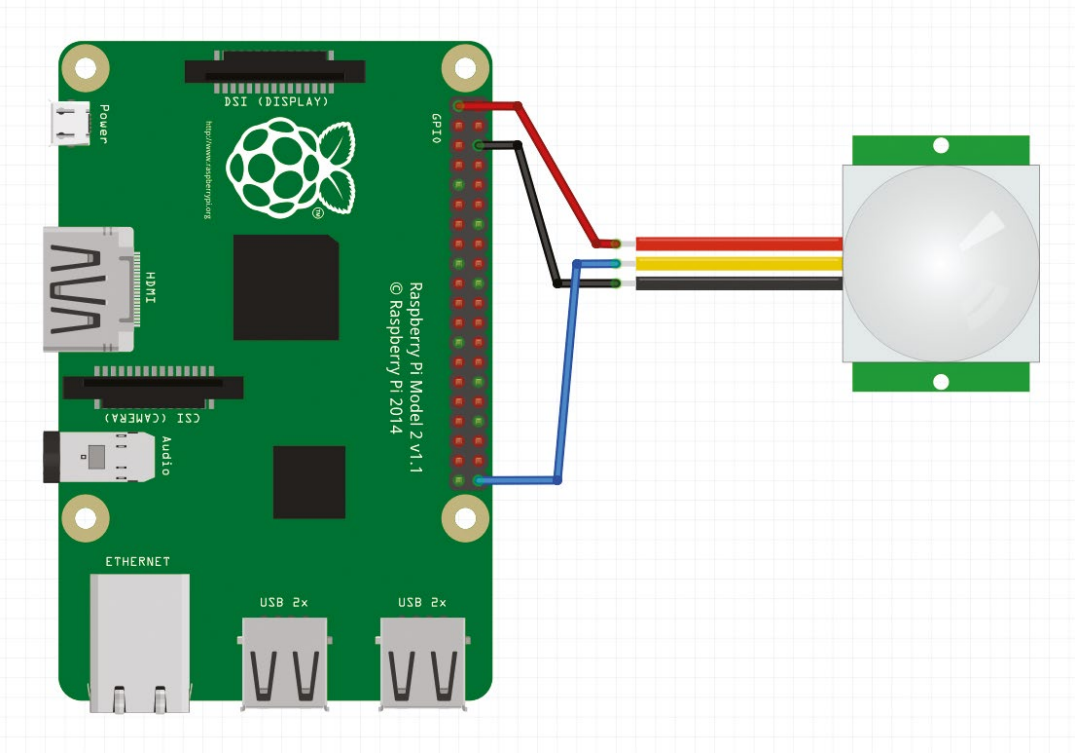
Vous pourrez trouver également :

- une seconde tension d'alimentation connectée à un régulateur et acceptant entre 5V et 20V selon le modèle. Dans ce cas, cette broche est marquée VCC et celle de l'alimentation avec une mention de la tension attendue comme 3.3V par exemple ;
- deux connexions pour alimenter directement les bobines d'un relais qui sera déclenché en cas de détection.

De plus, certains modules comportent des réglages ou des fonctions supplémentaires :

- une photorésistance (LDR) activable par cavalier ou micro-interrupteur, permettant de limiter la détection de mouvement en l'absence de lumière. Ici, il s'agit principalement de piloter un éclairage et ce genre de modules dispose donc souvent des connexions pour un relais ;
- un potentiomètre pour régler la sensibilité de la détection sans unité de mesure spécifique (c'est souvent une affaire de tâtonnement) ;
- un potentiomètre réglant la durée de commutation d'un relais. Là encore, ceci va de pair avec la présence des broches dédiées et de la LDR.

Enfin, la plupart des modèles permettent de configurer via un cavalier, un interrupteur ou un pont à souder, le mode de fonctionnement au choix entre « retriggerable » et « non retriggerable ». En mode retriggerable, la détection d'un évènement va provoquer un changement d'état (bas à haut) sur la sortie et une détection consécutive va prolonger le maintien de cet état. Ce n'est que lorsqu'aucune détection n'aura plus lieu que l'état de la sortie va alors à nouveau changer d'état (haut à bas). Tant qu'il y aura des détections dans un court délai, la sortie restera à l'état haut et l'ensemble des détections sera considéré comme un seul évènement.



En mode non retriggerable, une détection provoquera un changement d'état (bas à haut) en sortie puis un retour à l'état standard (bas). Une détection consécutive va alors provoquer un nouveau changement d'état, une temporisation et un nouveau retour à l'état standard. Un lot de détections sera donc perçu comme une succession d'évènements.

En fonction des besoins, l'un ou l'autre mode sera préféré. Pour un éclairage, par exemple, il n'est pas souhaitable de voir la lampe changer sans cesse d'état, plongeant régulièrement la personne dans le noir même en cas de mouvement...

Pour résumer, afin de bien choisir votre ou vos modules PIR, ceux-ci doivent pouvoir être alimentés en 3,3V ou 5V, fournir un signal de sortie impérativement en 3,3V et disposer d'un moyen de choisir le mode d'utilisation.

Le modèle utilisé ici est référencé SS502V110 et fabriqué par Sure Electronics, mais j'ai également testé un modèle plus petit de fabrication inconnue marqué HC-SR505, sans le moindre problème. Dans tous les cas, et je le répète : alimentez votre module en 5V ou en 3,3V et testez avec un multimètre la tension présente entre la sortie et la masse. Celle-ci ne doit en aucun cas être supérieure à 3,3V !

2. ASSEMBLAGE DES COMPOSANTS

La connexion du module PIR à une Raspberry Pi est d'une simplicité déconcertante. Selon que le module soit en 5V ou en 3,3V, connectez l'alimentation du module sur la broche 2 ou 1 de la Pi. La masse sera connectée à l'une des masses disponibles : 6, 9, 14, 20, 25, 30, 34 ou 39. Enfin, la sortie du module sera connectée sur l'une des broches GPIO de la carte. Le numéro de la broche importe peu, du moment que cela ne rentre pas en conflit avec d'autres fonctions de broches qui pourraient être configurées selon vos besoins :

- console série : 8 et 10 ;
- SPI : 19, 21, 13, 24 et 26 (pour un écran LCD par exemple) ;
- i2c : 3 et 5 (pour une RTC ou un capteur de température).

3. CODE ET COMMANDES

3.1 Premier essai

Nous allons, dans un premier temps, tâcher de détecter le changement d'état du module depuis la carte Raspberry Pi. L'une des façons de faire consiste à scruter en permanence l'état de la broche sur laquelle est connectée la sortie du module et à afficher un message si l'état est haut (à 3,3V). Cette technique s'appelle le *polling* :

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

import time
import RPi.GPIO as GPIO

# on utilise les numéros de broches
GPIO.setmode(GPIO.BOARD)
# broche 40 en entrée
GPIO.setup(40, GPIO.IN)

# boucle sans fin
while True:
    # la broche est a l'état haut ?
    if GPIO.input(40):
        # oui, on affiche un message
        print "Mouvement Détecté"
    # dodo pendant 1 seconde
    time.sleep(1)
```

Ce script Python très simple ne fait pas grand-chose. Il est constitué d'une simple boucle infinie, testant toutes les secondes si la broche est à l'état haut. Si tel est le cas, un message est affiché. On enregistrera ce code dans un fichier, **motion.py** par exemple, avant de changer les permissions pour le rendre exécutable avec **chmod +x motion.py**.

Il nous suffira ensuite de le lancer avec **./motion.py** et de passer la main devant le capteur PIR pour voir s'afficher, normalement, les messages de détection. Notez qu'avec les dernières versions de Raspbian, il n'est plus nécessaire de lancer ce genre de scripts via **sudo**, l'utilisateur **pi** par défaut ayant les droits adéquats pour accéder aux GPIO sans devenir temporairement super-utilisateur (**root**).

Une autre approche pour obtenir le même résultat consiste à ne pas scruter l'état de la broche, mais de laisser le système nous signaler un changement d'état et de réagir en conséquence.

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

# modules
import time
import RPi.GPIO as GPIO

# Déclenchement gPhoto
def detection(channel):
    # Affichage du message
    print "Mouvement Détecté"

# Configuration de la broche
GPIO.setmode(GPIO.BOARD)
GPIO.setup(40, GPIO.IN)

# Installation de la gestion de
# changement d'état
GPIO.add_event_detect(40, GPIO.
RISING, callback=detection)

# Message
print "En attente de détection..."

# Boucle sans fin
while True:
    time.sleep(10)
```

Ici, nous utilisons la méthode **add_event_detect()** permettant d'associer une fonction à un évènement. Cette méthode prend en argument la broche à surveiller (ici **40**), l'évènement avec ici un changement d'état de bas à haut (**RISING**) et le nom de la fonction à appeler (le *callback*). La fonction est déclarée un peu plus haut dans le script et ne fait, encore une fois qu'afficher un message.

Cette approche nous permet de sortir de la boucle tous les éléments de test et de reposer entièrement sur des fonctions appelées indépendamment. Cela tombe bien, car nous allons également aborder la notion de signaux qui est une forme d'évènements et donc quelque chose qui se traite de cette façon.

3.2 Ajoutons gPhoto

Il y a une fonction de gPhoto que nous n'avons pas encore abordé. En effet, celui-ci est en mesure de déclencher une prise de vue à la réception d'un signal. Les signaux dans un système UNIX comme Raspbian forment une méthode de communication entre processus. Lorsque vous utilisez CTRL+C par exemple pour interrompre le fonctionnement d'une commande, le système va suspendre le fonctionnement de ce programme et lui envoyer le signal SIGINT. Si celui-ci n'y répond pas, le système va simplement arrêter le processus en question.

Il existe toute une gamme de signaux ayant une signification bien précise dans tout le système, mais deux en particulier sont laissés à la discrétion des programmeurs pour leur création : SIGUSR1 et SIGUSR2. Dans le cas de gPhoto, le signal SIGUSR1 est utilisé pour déclencher une prise de vue sous certaines conditions. Lorsque gPhoto est en mode *time-lapse* par exemple, il est possible de le forcer, durant les temps d'attente, à prendre un cliché en lui envoyant un SIGUSR1. Pour cela, nous devons connaître le numéro de processus, ou PID, étant associé au gPhoto lancé. Numéro que vous pouvez récupérer avec la commande **`pidof`** suivie du nom du programme. Le signal est alors envoyé avec la commande **`kill -SIGUSR1`** suivie du PID en question. Il est possible de combiner les deux commandes avec **`kill -SIGUSR1 `pidof gphoto2``** ou tout simplement en utilisant la commande **`killall -SIGUSR1 gphoto2`**.

On pourra ainsi lancer gPhoto avec **`gphoto2 --interval -1 --capture-image-and-download`**, de façon à le laisser en attente avec un intervalle de prise de vue négatif et donc infini. Un premier cliché sera cependant pris par défaut dès le lancement de la commande, mais seul un SIGUSR1 en provoquera un autre.

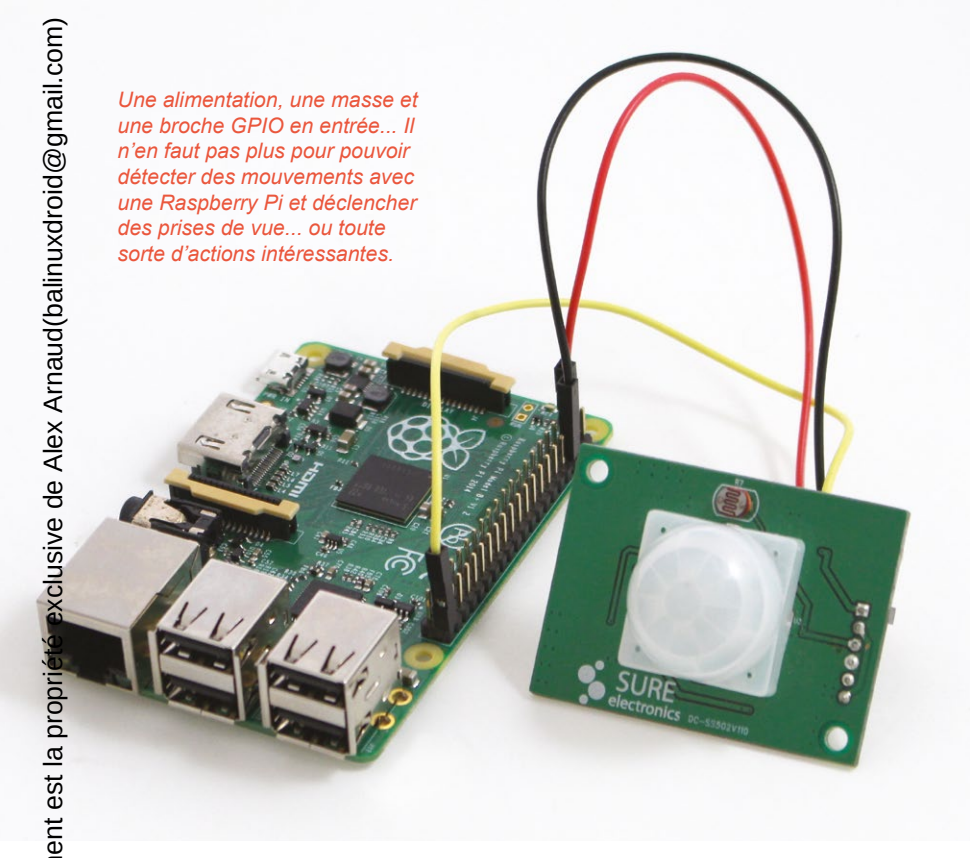
Tout ce que nous avons à faire est donc de lancer gPhoto puis d'envoyer un signal SIGUSR1 avec notre script si un mouvement est détecté par le capteur PIR. Nous allons cependant faire un peu plus en ajoutant une date et heure dans le message, tester la présence de gPhoto en mémoire et abandonner l'opération si ce n'est pas le cas et, nous aussi, gérer la réception d'un signal de façon à ce qu'un CTRL+C arrête non seulement notre script, mais également le processus gPhoto.

La version finale de notre script sera donc :

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

# modules
from subprocess import check_output
  #0 client-session (t4 r0 i0/0 o0/0 fd 4/5 cc -1)
import RPi.GPIO as GPIO

# Fonction pour quitter avec CTRL+C
def quitter(sig, frame):
    print "On quitte !"
    # Arrêt de gPhoto
    os.kill(gphotopid, signal.SIGTERM)
    # On quitte
    sys.exit()

# Déclenchement gPhoto
def detection(channel):
    # Obtention de l'heure système
    localtime = time.strftime('%x %X')
    # Affichage du message de capture
    print localtime,"- Mouvement Détecté ! Envoi SIGUSR1 à", gphotopid
    # Envoi du signal à gPhoto
    os.kill(gphotopid, signal.SIGUSR1)

# Configuration de la broche
GPIO.setmode(GPIO.BOARD)
GPIO.setup(40, GPIO.IN)

# Recherche du processus gPhoto en mémoire
try:
    gphotopid = int(check_output(["pidof","gphoto2"]))
```

```
except Exception:
    print "Pas de PID gPhoto trouvé"
    sys.exit()

# Prêt à travailler
print "gPhoto présent. PID = ", gphotopid

# Installation du gestionnaire pour CTRL+C
signal.signal(signal.SIGINT, quitter)

# Installation de la gestion de changement d'état
GPIO.add_event_detect(40, GPIO.RISING, callback=detection)

# Message
print "En attente de détection..."

# Boucle sans fin
while True:
    time.sleep(10)
```

Pour récupérer le PID de gPhoto, nous utilisons tout simplement la commande **pidof** présente dans le système et convertissons le résultat en valeur numérique. La technique utilisée est celle du **try**/**catch** permettant de tenter une opération et de réagir à une éventuelle erreur (exception). Si tel est le cas, ceci signifie que gPhoto n'est pas lancé et nous arrêtons là l'exécution du script.



Une alimentation, une masse et une broche GPIO en entrée... Il n'en faut pas plus pour pouvoir détecter des mouvements avec une Raspberry Pi et déclencher des prises de vue... ou toute sorte d'actions intéressantes.

Autre point intéressant du script, nous mettons en place un gestionnaire de signaux pour « attraper » un SIGINT découlant d'un CTRL+C. Si ce signal arrive, c'est la fonction associée **quitter()** qui s'en chargera et utilisera le PID de gPhoto pour lui envoyer un SIGTERM pour le « terminer » avant de stopper l'exécution du script.

Enfin, dernier ajout pour un minimum de suivi des opérations, nous récupérons l'heure courante dans la fonction **detection()** afin de faire précéder le message de détection de ces données utiles.

Il ne nous reste plus qu'à enregistrer tout cela, lancer gPhoto comme détaillé précédemment puis à lancer notre script :

```
$ ./motion2.py
gPhoto présent. PID =  3346
En attente de détection...
08/18/16 12:00:50 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:01:08 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:01:23 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:01:38 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:02:10 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:02:27 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:02:44 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:03:14 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:03:29 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:03:46 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
08/18/16 12:04:00 - Mouvement Détecté ! Envoi SIGUSR1 à 3346
```

À chaque détection par le capteur PIR, une photo sera prise automatiquement. Et le plus admirable dans tout cela, c'est qu'il nous est parfaitement possible d'utiliser les informations du précédent article afin de traiter automatiquement ces clichés, au fur et à mesure qu'ils sont faits, afin de les formater, les dater et les archiver...

4. LIMITATIONS ET PERSPECTIVES

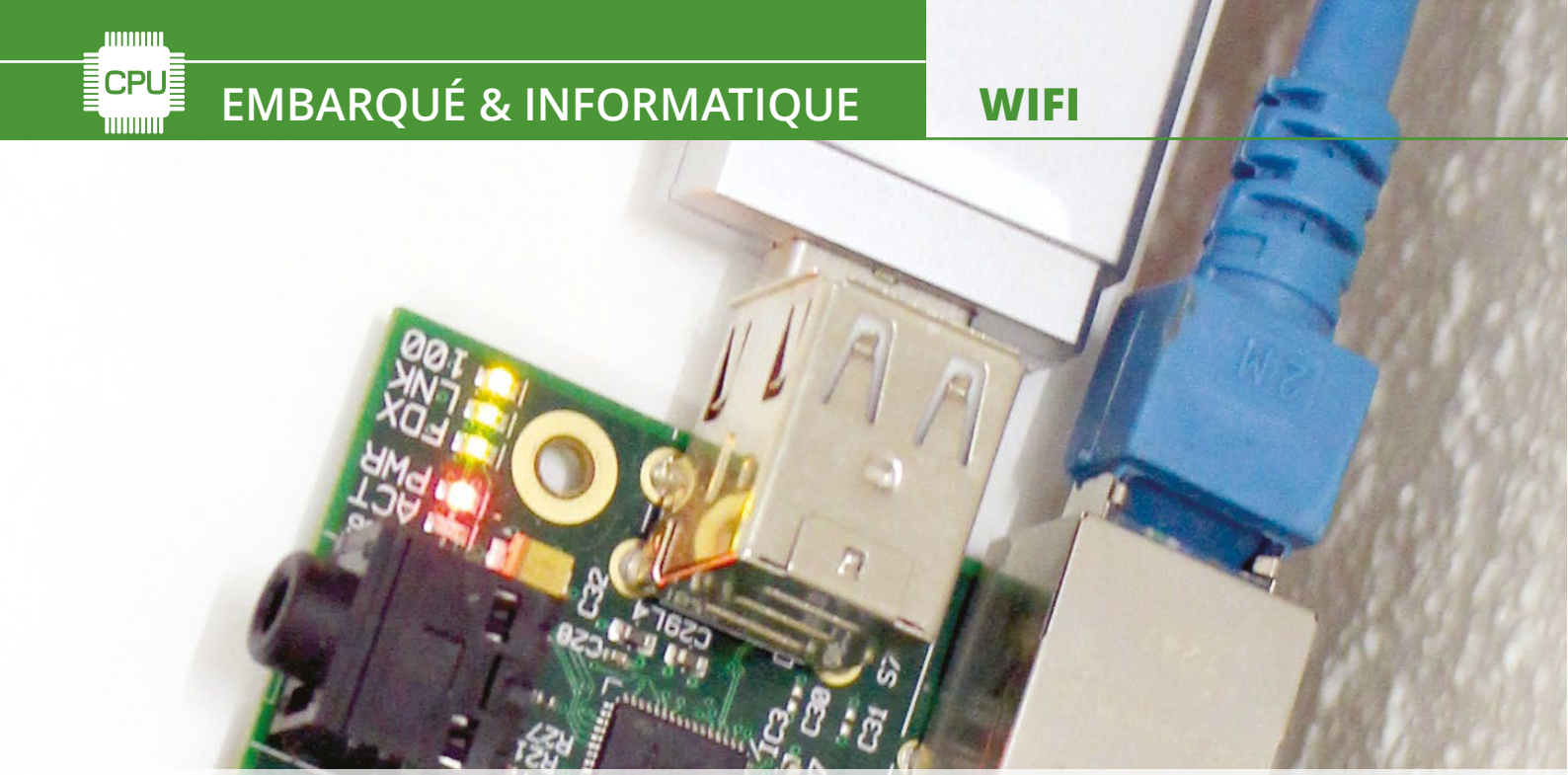
Les résultats dépendront de vos objectifs concernant cette technique. Le délai de propagation de l'évènement, par exemple, peut être problématique. Si par exemple, vous souhaitez capturer des clichés d'animaux sauvages (rongeur, oiseaux, etc.), la vitesse à laquelle la photo sera prise peut être insuffisante. On peut difficilement réduire le délai d'exécution du script et on tentera surtout de traiter le problème côté APN. La prise de vue automatique induit forcément un délai et un passage en manuel pourra être une bonne approche. Le fait de stocker la photo sur l'appareil, et d'utiliser **`--trigger-capture`** plutôt que **`--capture-image-and-download`** aidera aussi grandement.

Toujours dans le créneau de la photographie animalière, il pourra être souhaitable d'opter pour une configuration en retriggerable ou non retriggerable, sachant que dans le cas d'un reflex numérique le bruit du miroir lors de la prise de vue risque de faire fuir votre sujet. Il pourrait être ainsi souhaitable d'induire délibérément un délai entre la détection de mouvement et la prise de vue, pour s'assurer de l'immobilité de l'animal photographié. Mais ceci n'est en rien une science exacte...

Enfin, le capteur PIR n'est pas la seule façon de détecter une présence. D'autres modules peuvent ainsi être utilisés, voire plusieurs modules et capteurs de concert, permettant au script de décider de la façon et du moment de prendre des photos. On peut également envisager de changer les paramètres de prise de vue de l'APN en fonction du moment dans une journée ou des conditions environnementales (luminosité, température, etc.).

Il y a suffisamment de ports sur une Raspberry Pi pour créer un système de détection et de décision complet pour bien des usages. La limite, comme souvent, n'est que votre imagination... **DB**

Le dôme de plastique des modules est en réalité une lentille permettant de concentrer les rayonnements infrarouges sur le capteur pyroélectrique. Ceci nous paraît opaque ou translucide, mais est en réalité totalement transparent aux IR de la bonne longueur d'onde...



VOTRE RASPBERRY PI EN PONT WIFI/ETHERNET

Denis Bodor

Créer un point d'accès wifi avec une Raspberry Pi est quelque chose de relativement simple. L'objectif est généralement de fournir une connectivité vers la Pi et éventuellement d'en faire un point d'accès vers Internet pour des périphériques sans fil. La méthode généralement décrite consiste à créer un réseau Wifi d'une part et se connecter à un réseau filaire différent de l'autre. Il existe cependant une autre solution : créer un pont ou bridge en anglais...

Un point d'accès wifi classique se compose généralement de plusieurs éléments : un outil permettant l'authentification des connexions, un serveur distribuant des adresses IP et une solution pour faire transiter et filtrer les données du réseau Wifi vers le réseau filaire et inversement. Une installation réseau standard cependant comprend généralement déjà un accès à Internet sous la forme d'une Box ou d'un routeur câble. Le fait d'installer un point d'accès de cette manière revient donc à créer un second réseau local et à le connecter avec celui déjà existant. On a donc une interconnexion de deux réseaux distincts et une passerelle/routeur entre les deux.

Pour bien comprendre l'intérêt d'utiliser une autre méthode, plantons un peu le décor pour que vous ayez une idée de la situation et de certaines préférences qui sont les miennes. Avant toutes choses, je pars du principe qu'un réseau filaire est, par définition, mieux qu'un réseau wifi. Ainsi, la très grande majorité des connexions réseau sont filaires et ce n'est que pour quelques périphériques que je suis dans l'obligation de me rabattre sur le wifi, car ils ne disposent pas de connectivité Ethernet.

Sur un tel réseau, les adresses des machines sont attribuées individuellement de façon non dynamique. Cela permet non seulement de garder une correspondance entre machines (interface Ethernet) et adresses, mais également de limiter les connexions aux seules machines déjà connues. Ce n'est pas une vraie mesure de sécurité, mais cela permet de conserver un minimum d'ordre et surtout de me connecter aux machines via une simple adresse fixe (sans avoir recours à des protocoles de résolution comme DNS, mDNS, etc.).

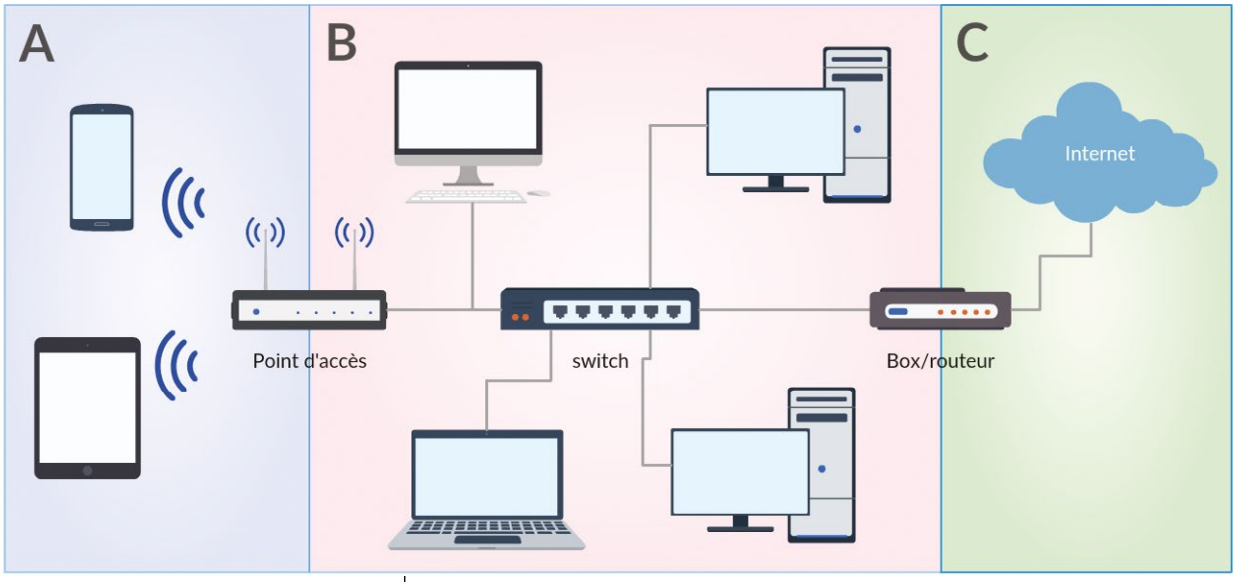
Dans un tel scénario, l'ajout d'un point d'accès wifi (ou même plusieurs) disposant de son propre réseau n'est pas forcément le plus logique. Il est plus simple d'avoir un seul serveur (DHCP) distribuant les adresses et les configurations IP, regroupant les adresses pour toutes les machines du réseau, filaire et wifi. Il n'existe qu'un endroit où cet « annuaire » existe et où l'on peut facilement apporter des modifications. Il faut donc, en principe, qu'un périphérique wifi fasse partie du même réseau que les machines connectées en Ethernet et de ce fait les rende accessibles de la même manière d'un point à l'autre du réseau.

Le point d'accès ne doit alors pas être un point de connexion entre deux réseaux, mais un simple pont ne faisant que relayer les données d'un média (sans fil) à un autre (filaire). Il ne s'agit plus de deux réseaux, mais de deux segments d'un même réseau et le point de jonction est un pont ou *bridge* en anglais.

1. PRÉLIMINAIRE

Nous partons ici du principe que, dans un premier temps, la Raspberry Pi n'aura pour objectif que de servir de point d'accès et de bridge vers le réseau filaire. La carte utilisée ici est une Pi 1 ou en d'autres termes le tout premier modèle disponible (avec carte SD et connecteur RCA vidéo). Celle-ci sera équipée d'un adaptateur wifi Linksys WUSB54GC presque tout aussi ancien.

La carte SD utilisée a été fraîchement effacée et initialisée avec une image récente de Raspbian Jessie Lite 4.4 (du 27/05/2016). Aucune fonctionnalité spécifique n'est nécessaire et le tout a été configuré sans interface graphique, avec un minimum de mémoire pour le processeur graphique et quelques outils standards facilitant la configuration (Vim, Midnight Commander, etc.).



L'architecture générale d'un réseau local avec un accès Internet et un point d'accès wifi peut être le suivant. Nous avons en B le réseau local filaire, en C Internet (le reste du monde) derrière un routeur ou une box et en A, le plus souvent, un autre réseau en wifi. Le point d'accès s'occupe de faire la liaison entre les machines des réseaux A et B via une technique appelée NAT. Mais si le point d'accès est configuré en bridge, A et B forment alors un même réseau et ne sont que deux segments de ce dernier. Exactement comme si le point d'accès était un simple hub/switch Ethernet, les périphériques wifi et les machines du LAN sont sur un même réseau IP.

Après un premier redémarrage post configuration (locales, fuseau horaire, etc.), le système a été mis à jour avec **`sudo apt-get update`** et **`sudo apt-get dist-upgrade`** et enfin, les paquets suivants ont été installés :

- **`hostapd`** pour la gestion du point d'accès ;
- **`bridge-utils`** pour la configuration en tant que bridge.

Il conviendra également de supprimer le paquet préinstallé **`avahi-daemon`**, un outil de configuration automatique utilisant les protocoles Zeroconf. Ceci ne nous est d'aucune utilité ici et risque même de perturber nos manipulations. Il en va de même pour **`dhcpcd5`**.

2. CRÉER LE POINT D'ACCÈS WIFI

La première étape consiste à configurer le point d'accès. Le nom de l'interface wifi dépendra du matériel utilisé. Ici, l'adaptateur Linksys utilise une puce Ralink rt73 et un firmware est nécessaire pour le faire fonctionner. Celui-ci, sous la forme du fichier **`/lib/firmware/rt73.bin`** est fourni par le paquet **`firmware-ralink`** installé par défaut avec Raspbian comme bien d'autres firmwares. Nous n'avons donc rien à faire si ce n'est de connecter le périphérique et laisser le système faire son travail. Il suffira alors d'utiliser la commande **`sudo ifconfig -a`** (ou **`ip link show`**) pour obtenir la liste des interfaces réseaux disponibles et repérer celle correspondante, ici **`wlan0`**.

On pourra ensuite se tourner vers l'écriture du fichier **`/etc/hostapd/hostapd.conf`** :

```
# interface à utiliser
interface=wlan0
# pilote
driver=nl80211
# SSID en UTF8
utf8_ssid=1
# nom de l'AP
ssid=coucouAP
# mode Wifi
hw_mode=g
# canal à utiliser
channel=3
# on accepte toutes les MAC
macaddr_acl=0
# on accepte que OSA
auth_algs=1
# on ne cache pas le SSID
ignore_broadcast_ssid=0
# chiffrement WPA2
wpa=2
# la phrase secrète
wpa_passphrase=monmot2passe
# gestion PSK
wpa_key_mgmt=WPA-PSK
# chiffrement TKIP pour WPA
wpa_pairwise=TKIP
# chiffrement CCMP pour WPA2
rsn_pairwise=CCMP
# config en bridge
bridge=br0
```

Cette configuration est très similaire à celle déjà utilisée dans les articles de *Hackable n°6* (point d'accès Tor) et 8 (base de collecte de sondes de température). Les commentaires du fichier parlent d'eux-mêmes. La ligne très importante est **`bridge=br0`**. Celle-ci indique au démon HostAP qu'il doit ajouter cette interface dans le bridge **`br0`** après sa configuration.

Enfin, pour que le service HostAP se lance correctement au démarrage de la Pi, nous devons éditer le fichier **`/etc/default/hostapd`** et spécifier le chemin et le nom du fichier de configuration à utiliser :

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Ceci fait, vous pourrez (re)démarrer le démon HostAP avec **`sudo service hostapd start`** et votre point d'accès sera créé. Il devrait être immédiatement visible depuis un périphérique wifi. La connexion après saisie du mot de passe devrait également fonctionner, mais le périphérique n'obtiendra pas d'adresse et vous recevrez alors une erreur. C'est normal, nous n'avons absolument rien installé permettant de répondre à une requête DHCP, utilisée pour obtenir la configuration réseau et l'adresse IP. Ceci ne sera pas une tâche pour la Raspberry Pi puisque l'idée est précisément de connecter entre elles les interfaces **`eth0`** et **`wlan0`** pour former un unique réseau « physique ».

3. CONFIGURER LE BRIDGE

La notion de pont ou de bridge est relativement simple puisqu'il suffit d'imaginer cela comme un switch réseau logiciel. Ainsi, de la même manière que vous connectez plusieurs câbles Ethernet sur un switch physique pour relier entre elles des machines, un bridge connecte plusieurs interfaces réseau d'une seule et même machine. Ce qui circule sur une interface sera propagé à l'autre et inversement.

Tout ceci se configurera dans le fichier **/etc/network/interfaces** :

```
#interface à activer
auto lo br0

# interface virtuelle loopback
iface lo inet loopback

# interface Ethernet
iface eth0 inet manual

# interface Wifi
iface wlan0 inet manual

# Le bridge
iface br0 inet dhcp
        bridge_ports eth0
        bridge_hw b8:27:eb:5c:06:c5
        # wlan0 est ajouté par hostapd
```

Les interfaces **eth0** et **wlan0** sont présentes, mais on précise une configuration manuelle. Aucune adresse IP ou configuration réseau via DHCP n'est configurée, ces interfaces ne sont là que pour être des composants du bridge. Celui-ci dispose de sa propre interface sous le nom **br0** qui est configurée pour obtenir une adresse via DHCP. L'adresse matérielle (MAC) utilisée pour **br0** sera l'adresse la plus petite de toutes les interfaces composant le bridge.

Ceci peut être une source de problèmes dans certaines situations et mieux vaut limiter les risques. La solution consiste à utiliser **bridge_hw** en spécifiant l'adresse matérielle de l'interface **eth0** (qu'on peut obtenir avec **ifconfig** ou **ip link**). L'interface représentant le bridge ne peut prendre qu'une adresse matérielle de l'une des interfaces qui le compose, vous ne pouvez pas mettre n'importe quoi et devez impérativement copier/coller ce que vous retourne **ifconfig** ou **ip link**.

Notez que dans la configuration, nous utilisons **bridge_ports** qu'avec **eth0** et non en spécifiant toutes les interfaces utilisées comme on le fera pour *bridger* **eth0** et **eth1** sur un PC par exemple. HostAP ajoute en effet lui-même **wlan0** lorsque le point d'accès est configuré.

Après redémarrage, tout sera automatiquement mis en place : le point d'accès sera configuré, le bridge sera créé et une adresse IP ainsi qu'une configuration réseau seront demandées via DHCP. Le serveur DHCP (spécifique ou celui d'une box) va déterminer la configuration puis répondre à la requête et l'interface sera alors configurée. Vous pouvez jeter un œil à la configuration du bridge avec la commande :

```
pi@raspberrypi:~ $ brctl show
bridge name   bridge id           STP enabled  interfaces
br0           8000.b827eb5c06c5   no           eth0
                                               wlan0
pi@raspberrypi:~ $ ifconfig
br0       Link encap:Ethernet  HWaddr b8:27:eb:5c:06:c5
          inet adr:192.168.10.160  Bcast:192.168.10.255  Masque:255.255.255.0
          adr inet6: fe80::ba27:ebff:fe5c:6c5/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:188 errors:0 dropped:0 overruns:0 frame:0
          TX packets:114 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:19588 (19.1 KiB)  TX bytes:15441 (15.0 KiB)

eth0      Link encap:Ethernet  HWaddr b8:27:eb:5c:06:c5
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:193 errors:0 dropped:0 overruns:0 frame:0
          TX packets:114 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:19818 (19.3 KiB)  TX bytes:16753 (16.3 KiB)

lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:Hôte
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 00:22:6b:a9:32:90
          adr inet6: fe80::222:6bff:fea9:3290/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:80 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:0 (0.0 B)  TX bytes:10197 (9.9 KiB)
```

Si, à présent, vous vous connectez en wifi à votre point d'accès, votre périphérique dialoguera avec HostAP pour négocier la connexion chiffrée. Une fois authentifié, il enverra une requête DHCP pour obtenir la configuration réseau. Celle-ci passera du wifi au réseau filaire et se propagera sur votre réseau local pour finalement arriver au serveur DHCP qui va la traiter. Votre périphérique wifi fera alors partie du même réseau local.

La différence avec un point d'accès fournissant une adresse IP lui-même est le niveau de fonctionnement de la passerelle. Un bridge fonctionne au niveau de la couche « liaison » (2) alors que l'autre solution fonctionne sur la couche « réseau » (3) et implique de manipuler les paquets avec Netfilter (**iptables**) et faire ce qu'on appelle du NAT (*Network Address Translation* ou traduction d'adresse réseau).

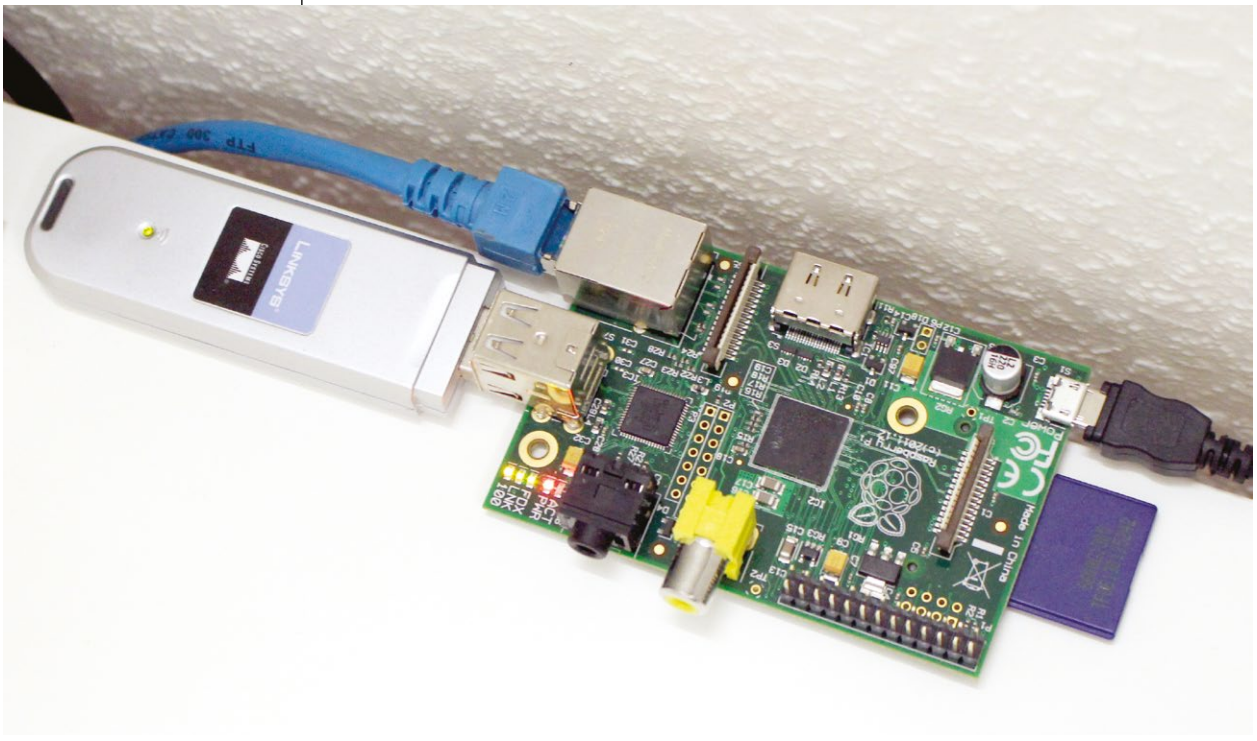
Ceci signifie également que le périphérique wifi utilisé sera directement accessible sans autre forme de procédé depuis n'importe quelle machine du réseau local. Si ce périphérique est une autre carte Raspberry Pi, par exemple, il suffira d'utiliser son adresse IP pour s'y connecter en SSH et obtenir une ligne de commandes.

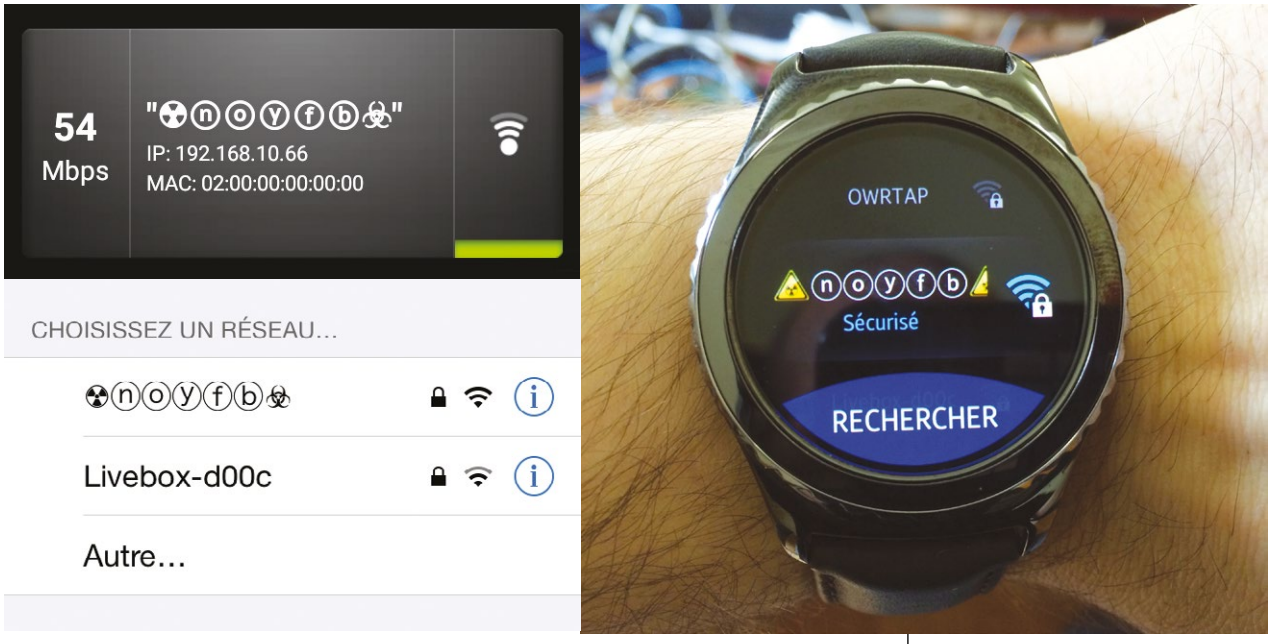
Dédier une Raspberry Pi à ce type d'usage peut sembler être du gâchis. Mais en y réfléchissant bien, un point d'accès wifi capable de fonctionner en bridge vous coûtera plus cher qu'une vieille Pi de première génération et un adaptateur USB wifi plus vieux encore. De plus, rien ne vous empêche par la suite d'ajouter d'autres services pour compléter l'installation ou de multiplier les points d'accès avec des configurations distinctes (certains adaptateurs peuvent supporter plusieurs SSID avec HostAP).

4. PETIT BONUS : METTRE DES ICÔNES DANS LE NOM DU POINT D'ACCÈS

Nous avons plusieurs fois parlé dans ces pages de points d'accès wifi, mais saviez-vous qu'il est possible d'utiliser des symboles dans le nom du point d'accès et non simplement du texte ? En effet, le SSID peut être composé de jusqu'à 32 caractères UTF-8. Or l'ensemble des caractères UTF-8, déjà utilisés par plus de 86% des sites Internet, contient bien plus que des signes pour différentes langues. On y trouve aussi de sympathiques pictogrammes, des icônes et des emoji/émoticônes (**https://fr.wikipedia.org/wiki/Emoji**).

Pour la petite histoire, je me suis rendu compte de cela d'une façon fort amusante après avoir regardé un épisode de Doctor Who et plus exactement l'épisode 6 de la saison 7, « *The Bells of Saint John* » où de mystérieux points d'accès capturent et téléchargent l'esprit des gens dans un cloud. Les points d'accès sont nommés





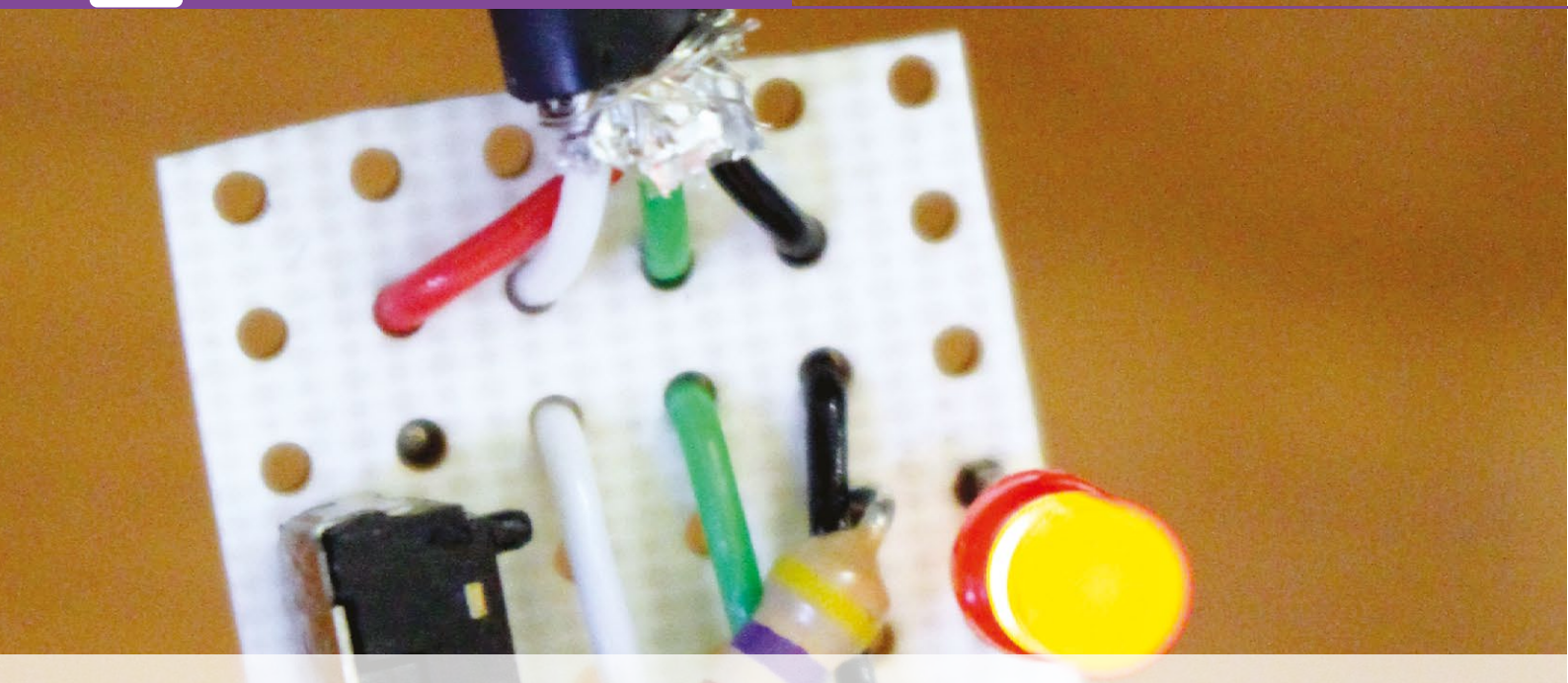
Voici quelques exemples d'affichage d'un SSID (nom de point d'accès) utilisant des caractères UTF-8. Sur une smartwatch Gear S2 sous Tizen, avec un iPhone ou encore avec un widget sur Android. Remarquez que si les symboles sont relativement proches, chaque système les affiche de façon sensiblement différente. Avec un matériel ne sachant pas afficher de l'UTF-8, le nom cependant apparaîtra comme une longue série de caractères cryptiques et vides de sens...

par des symboles géométriques et tout naturellement je me suis demandé s'il n'était pas possible de faire de même (utiliser des symboles, non capturer l'esprit des gens). Et il se trouve que je n'étais pas le seul à me poser la question, il est effectivement possible de nommer son point d'accès avec toute une gamme de caractères amusants !

Pour faire cela, il vous suffit d'éditer le fichier de configuration de HostAP et d'y placer les caractères UTF-8 de votre choix. Bien entendu, il faut que votre éditeur de texte soit compatible UTF-8 et vous pourrez alors très simplement copier/coller des symboles depuis, par exemple, votre navigateur web (connexion en SSH, *locales* configurées en `fr_FR.UTF-8`, Vim et copier/coller à la souris fonctionne très bien).

La seule chose à ne pas oublier est d'ajouter une ligne `utf8_ssid=1` dans le fichier de configuration et vous pourrez laisser libre cours à votre imagination. En fonction de la plateforme cliente, certains caractères apparaîtront même sous la forme de petites icônes en couleur. J'ai fait des tests avec une tablette Nvidia Shield sous Android 6, un iPhone 5, une Gear S2 et un Samsung Z3 sous Tizen et le rendu est systématiquement impressionnant !

Certains utilisateurs semblent ne pas arriver à saisir directement les caractères dans le fichier de configuration. Une alternative possible pour arriver au même résultat consiste à spécifier la valeur hexadécimale de chaque octet de chaque caractère en le faisant précéder de `\x`. `\xe2\x98\x94` produira ainsi le caractère U+2614, ou en d'autres termes l'icône d'un parapluie sous la pluie. C'est plus fastidieux, mais si vous rencontrez des difficultés, ce sera la voie à suivre. **DB**



INTÉGRER L'ARDUINO DANS UNE RÉGLETTE LUMINEUSE : ATTENTION À L'ALIMENTATION !

Denis Bodor

Dans le numéro précédent, nous avons vu comment modifier une réglette afin de remplacer d'ennuyeuses leds blanches par 72 leds « intelligentes » RVB de façon à pouvoir piloter l'état de chaque led individuellement et ainsi créer des animations avec une carte Arduino. Une évolution possible du projet consiste à utiliser une carte Arduino Nano ou Leonardo Micro pour l'intégrer directement dans la réglette. Mais attention ! Une carte Arduino ne s'alimente pas n'importe comment !

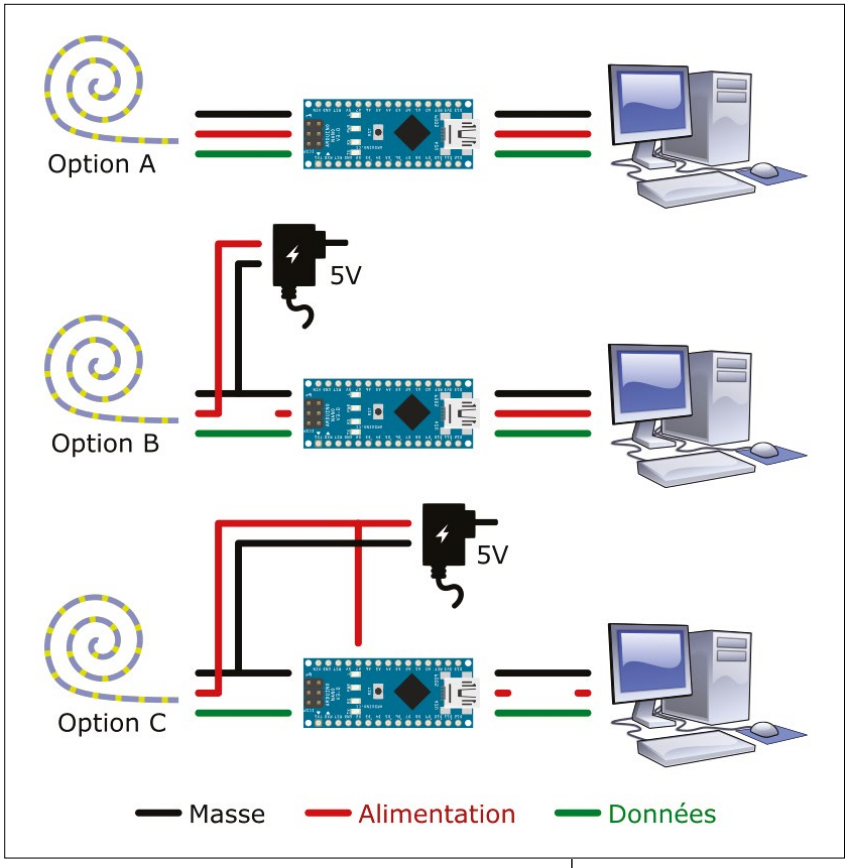
Notre projet initial, en dehors de la modification de la réglette elle-même, consistait tout simplement à contrôler 72 leds WS2812b assemblées en ruban avec une Arduino UNO. Le ruban demande beaucoup de courant et ne peut être alimenté par la broche +5V de la carte Arduino. Il dispose donc d'une alimentation propre qui n'est pas reliée à la carte. Seule la masse est commune au ruban, à la carte et à la connexion USB (c'est normal, c'est la masse, la référence 0V de tout le circuit).

En cherchant à intégrer une carte Arduino de faible dimension dans la réglette, comme une Nano, un problème pratique se pose : si la carte est dans l'objet, mieux vaut éviter d'avoir recours à deux alimentations électriques, une pour le ruban et une pour la carte. Il faut donc trouver un moyen d'alimenter les deux parties avec une seule source.

1. COMMENT PEUT ÊTRE ALIMENTÉE UNE CARTE ARDUINO

Si nous prenons l'exemple de l'Arduino UNO, nous avons plusieurs connexions permettant d'alimenter le microcontrôleur qu'elle intègre :

- Le port USB. C'est l'alimentation sans doute la plus utilisée qui permet à la fois de fournir du courant à la carte et de communiquer avec elle, aussi bien pour la programmer que pour faire dialoguer un croquis avec, par exemple, le moniteur série de l'environnement de développement. Le courant théorique maximum via cette connexion est de 500mA.
- Le connecteur JACK. Celui-ci est destiné à la connexion d'une alimentation externe sous la forme d'un bloc secteur fournissant un courant continu entre 250mA et 1A avec une tension située entre 9 et 12V. En principe, la plage de tensions se situe entre 7 et 20V, mais la documentation recommande toutefois 9-12V. Derrière ce connecteur se trouvent une diode de protection (assurant contre l'inversion de polarité) et un régulateur de tension fournissant le courant au reste de la carte. Alimenter la carte via ce connecteur permet de disposer de plus de courant pour vos montages utilisant alors la broche 5V pour la connexion de modules et composants externes. Il reste possible d'utiliser le connecteur USB avec ce type d'alimentation, aussi bien pour la programmation que pour



Voici trois façons envisageables d'alimenter des leds WS2812b et une carte Arduino. L'option A repose sur le fait d'alimenter la carte via USB avec un PC, ceci n'est utilisable que si le nombre de leds est très faible (< 8). L'option B permet d'alimenter les leds avec un adaptateur secteur alors que l'Arduino est alimenté via USB. Notez que la broche 5V de l'Arduino n'est PAS connectée aux leds. Enfin, l'option C est celle retenue ici. Aucune alimentation n'est fournie via USB et les leds comme la carte sont alimentées par le bloc secteur.

Une carte Arduino Nano est suffisamment petite pour pouvoir se glisser dans la réglette. L'alimentation unique pour les leds et la carte nous permettent de faire fonctionner le croquis sans connexion USB. Notez le gros rectangle noir en bas, il s'agit d'une gaine thermo-rétractable isolante dans laquelle sera ensuite glissée l'Arduino avant d'être collée dans la réglette.

le moniteur série. Notez que sur les Arduino disposant de ce connecteur, un circuit spécifique, composé d'un amplificateur opérationnel (LM358) et d'un MOSFET, coupe automatiquement l'alimentation USB si une tension est présente entre la diode et le régulateur (point Vin).

- La broche Vin. Cette broche est connectée à un point situé avant le régulateur, mais après la diode de protection. Elle possède deux usages. Comme entrée pour l'alimentation pour fournir du courant au régulateur indépendamment du connecteur JACK, et comme sortie pour fournir du courant à la tension appliquée sur le connecteur JACK, moins la chute de tension de la diode dans le sens passant (sens direct). En cas d'alimentation via cette broche, il est absolument hors de question de connecter une alimentation sur la prise JACK. Le circuit de protection coupant l'alimentation USB en cas de tension sur Vin n'est présent que sur les Arduino disposant d'un connecteur JACK, ce qui n'est pas le cas, par exemple, sur Arduino Nano qui se satisfait d'une diode Schottky entre la tension d'alimentation USB et la tension d'alimentation 5V de toute la carte (diode qui peut être absente sur certains clones chinois à bas prix).
- La broche 5V. Celle-ci est connectée directement à la ou les broches d'alimentation du microcontrôleur. Il n'y a aucune protection à ce niveau. Comme il s'agit d'un point de connexion se trouvant après le régulateur, c'est un accès direct au « bus » d'alimentation 5V de l'ensemble de la carte. Cette broche, comme Vin, peut être utilisée à la fois pour fournir du courant à des composants externes ou pour alimenter la carte, mais avec une tension bien régulée.

Notez que la broche marquée 3.3V ne doit en aucun cas être utilisée pour alimenter la carte ou le microcontrôleur. Cette tension, sur beaucoup de cartes Arduino, est

fournie directement par un second régulateur et permet uniquement d'alimenter des composants en 3,3V. C'est une très mauvaise idée d'appliquer une tension sur la sortie de bon nombre de régulateurs.

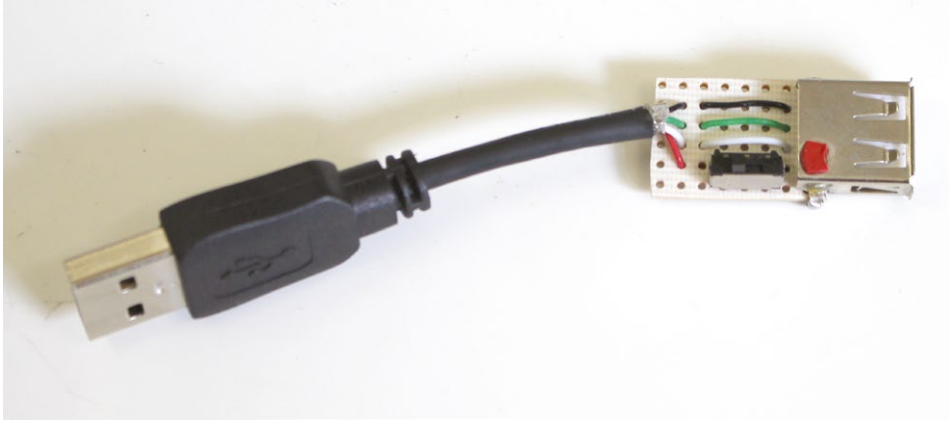
2. RETOUR À LA RÉGLETTE ET AU RUBAN DE LEDS

En résumé, sur une Arduino UNO, on peut utiliser l'alimentation USB ou le connecteur JACK sans problème lors de la programmation de la carte. Sur une Nano en revanche nous n'avons pas de JACK, mais uniquement la broche Vin. Cependant, ceci ne nous est d'aucune utilité puisque le ruban de leds doit être alimenté en 5V. Ceci est insuffisant pour Vin puisque la tension en entrée du régulateur doit être supérieure, souvent d'un minimum de 1V, à la tension en sortie.

Il ne nous reste donc plus que l'USB et la broche 5V pour régler notre problème. Notre objectif est d'alimenter les leds **ET** la carte avec une seule source en 5V fournissant suffisamment de courant pour l'ensemble des WS2812b. La tension d'alimentation via USB, nommée VUSB, n'est pas accessible sous la forme d'une broche et ceci serait une mauvaise idée, car le PC fournit également une tension de +5V via l'USB.
De façon générale, en l'absence d'une raison spécifique et parfaitement légitime, on ne connecte **jamais** les sorties de plusieurs sources d'alimentation entre elles.

Voici donc que la solution apparaît d'elle-même, nous n'avons plus que la broche 5V dans notre épuisette, ou en d'autres termes, le bus d'alimentation lui-même. Ceci implique deux choses importantes :

- Il faut que le courant soit « propre » et la tension bien régulée, ce qui implique généralement l'utilisation d'un ou plusieurs condensateurs de filtrage/découplage entre la tension d'alimentation et la masse, en plus d'une source fiable et stable. C'est quelque chose qu'il est déjà important de mettre en place lorsqu'on alimente une ou plusieurs WS2812b et cela ne changera donc pas notre circuit et nos besoins.
- Il ne faut en aucun cas que les tensions 5V et VUSB se « mélangent ». Sur une carte Arduino Nano officielle, ceci est garanti par la fameuse diode Schottky (temps de commutation très court et seuil de tension directe très bas), mais j'ai déjà eu le loisir, ou la déplaisante surprise, de constater que cette diode n'est pas toujours présente sur les clones les moins chers. De plus, ceci ne fait que protéger le PC d'une surtension côté montage et non



Pour nous assurer de l'absence de problème avec ce montage et éventuellement d'autres, un adaptateur USB mâle/femelle type A est assemblé avec, sur la ligne d'alimentation, un interrupteur permettant au choix de fournir ou non du courant au périphérique.

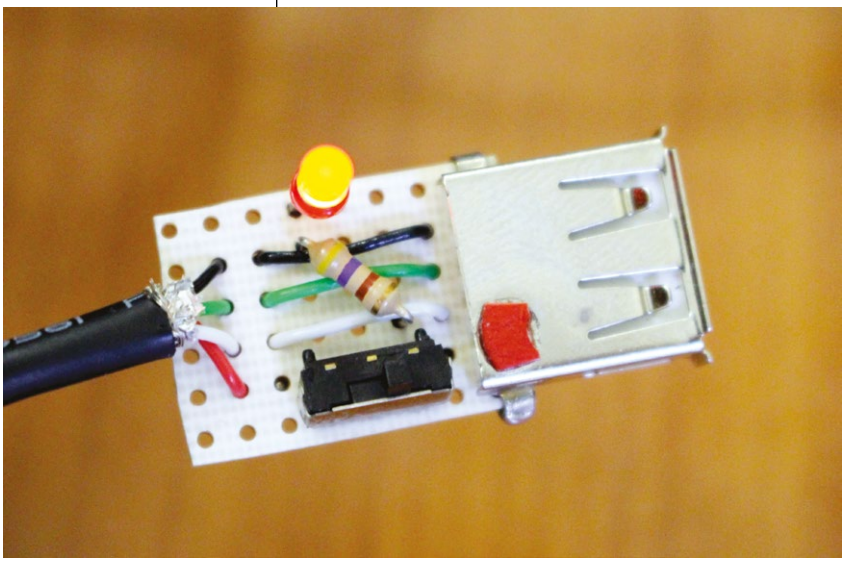
l'inverse étant donné l'orientation de la diode (et certains câbles USB non blindés sont de véritables antennes).

Il faut également prendre en compte un autre point : en n'alimentant pas le montage avec le bloc d'alimentation et en connectant la carte Arduino en USB, celle-ci va tenter d'alimenter le ruban de leds, mais sera incapable de fournir le courant adéquat.

La solution que j'ai donc adoptée est la suivante : alimenter le montage complet (ruban et Nano) avec le bloc d'alimentation de 5V/10A et utiliser un câble USB modifié pour connecter ponctuellement l'ensemble au PC pour une mise à jour du croquis. La modification du câble USB est fort simple puisqu'il suffit de couper le connecteur d'alimentation (rouge) qui s'y trouve.

En poussant l'idée un rien plus avant, j'en suis arrivé à assembler un petit adaptateur USB A mâle vers USB A femelle équipé d'un interrupteur sur la ligne d'alimentation me permettant d'alimenter ou non un périphérique USB en fonction de mes besoins. Ceci permet alors d'utiliser n'importe quel câble USB adapté à la carte Arduino utilisée (type B pour la UNO, type mini-B pour la Nano ou type micro-B pour la Leonardo). Une évolution suivante de l'adaptateur intègre, de plus, une résistance de 470 ohms et une led rouge entre la masse et la ligne d'alimentation après l'interrupteur, permettant de n'avoir aucun doute quant à la coupure de la ligne d'alimentation (je me connais, alors je suis prudent).

Taraudé par l'existence d'un risque de mauvaise manipulation, je n'ai pu m'empêcher d'ajouter une résistance et une led à l'adaptateur, connectés entre la masse et l'alimentation côté périphérique. Ainsi je sais, avant de brancher quoi que ce soit, si le périphérique sera ou non alimenté via USB.



En conséquence, le fonctionnement est le suivant :

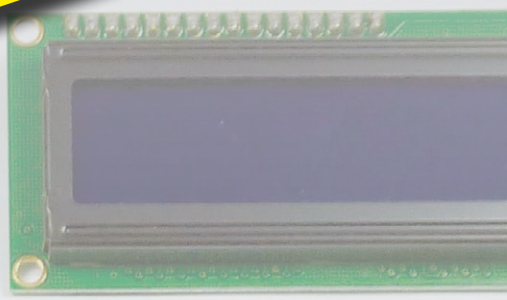
- une seule et unique alimentation fournie le courant pour les leds et la carte Arduino et le projet est autonome ;
- le montage ne peut être programmé en USB que s'il est sous tension et alimenté avec le bloc de 10A ;
- je peux également déconnecter l'alimentation du projet, débrancher le ruban et utiliser un câble USB standard si nécessaire.

3. UN DERNIER MOT

Ce projet est loin d'être terminé. La monopolisation d'une carte Arduino dans le seul but d'animer les leds revient à délibérément ignorer les entrées/sorties honteusement inutilisées. En ajoutant une RTC, la barre pourra donner l'heure. Un capteur de température ou d'humidité fera de cette barre un indicateur de confort original. Utiliser une entrée analogique nous permettra d'échantillonner un signal audio et créer un vumètre. Et enfin, connecter un module Wifi ou Bluetooth nous offrirait un contrôle à distance depuis un PC, un smartphone ou une smartwatch. À suivre donc... **DB**

TOUJOURS DISPONIBLE !

LE 1ER HORS-SÉRIE DE HACKABLE !



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

TOUJOURS DISPONIBLE SUR : www.ed-diamond.com

DOUANES & DROITS INDIRECTS

INFORMATICIENS

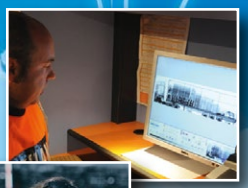
REJOIGNEZ LA DOUANE !

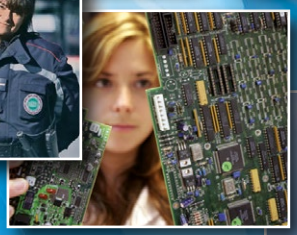
Mettez vos talents au service de la protection du territoire.

RÉSEAU INTERNATIONAL DE LA DOUANE

PROTÉGER LES CITOYENS ET L'ENVIRONNEMENT

VALIDATION





PARTICIPER AU FINANCEMENT DES SERVICES PUBLICS

RECHERCHE

ACCOMPAGNER LES ACTEURS DU COMMERCE MONDIAL

#LaDouaneRecrute
douane.gouv.fr / recrutement

Sur iOS et Android :
douanefrance.mobi

@douane_france

Contactez Infos Douane Service
0811 20 44 44 Service 0,06 € / min + prix appel

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DES FINANCES ET DES COMPTES PUBLICS