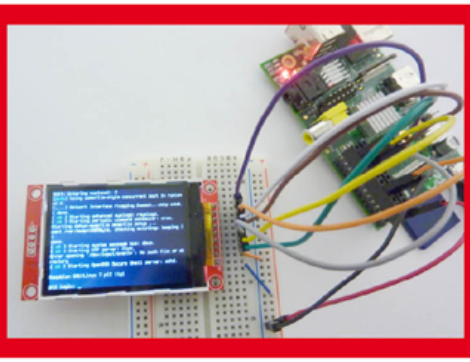
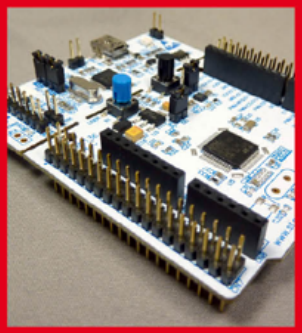
N°3 | NOVEMBRE-DÉCEMBRE 2014

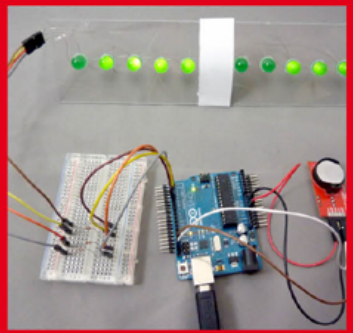
HACKABLE MAGAZINE

DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

France METRO : 7,90 € – CH : 13 CHF – BEL/PORT.CONT : 8,90 € – DOM TOM : 8,50 € – CAN : 14 $ cad – TUNISIE : 18 TND – MAR : 100 MAD







L 19338 - 3 - F: 7,90 € - RD

ÉDITO

Ah la dépendance à la technologie !

Certaines mésaventures donnent l'occasion de faire des éditos. Celle présentement entrain de se dérouler en ce mardi soir 22h23 se résume en une attente dans une voiture (« wagon » c'est pour les marchandises) du TER 96239 bloqué en gare parce qu'un arbre est tombé sur un caténaire, coupant ainsi l'alimentation de la ligne entre Sélestat et Colmar (et au passage, « Touche pas à mon Alsace ! »). C'est donc un édito en direct live.

Voici donc qu'un gros morceau de cellulose a réussi à bloquer trois TER, un Intercité et un TGV pendant déjà plus de 2h30, avec pour seule aide, la gravité et un petit coup de vent. Mais ce n'est pas tout. Cette masse végétale a également réussi, dans le même temps, à provoquer des réactions de survie typique de l'espèce dominante de la planète (nous, en principe). Parmi ces réactions, la première que j'ai pu observer a été la recherche d'une connexion stable à Internet (« rhooo, mais il est pourri le réseau ici »). Le temps passant des tentatives se sont multipliées pour trouver une source de nutriment... pour les smartphones. En pleine démarche anthropologique amateur j'ai alors assisté à des approches primitives de négociation autour des sources (de courant) : « Vous avez un chargeur ? », « Il y a une prise ici ? » et même « puis-je me brancher sur votre ordinateur pour recharger ? »

Ce n'est finalement qu'après le passage d'une vague de désespoir technologique profonde que certains ont alors pris la peine de prendre en compte certains besoins qui leur semblaient secondaires : manger, boire, trouver les toilettes...

L'autre point notable qui m'a quelque peu surpris était la difficulté avérée qu'avait l'information à circuler. Il semblerait qu'en l'absence prolongée de certains médias numériques, l'information ne puisse plus garder son intégrité très longtemps. La branche s'est ainsi transformée en arbre puis en deux arbres, le caténaire s'est changé en container, les ouvriers œuvrant au changement nocturne des traverses en réparateurs d'un dégât provoqué à quelques kilomètres de là ou encore un simple rappel des passagers du TGV en une invitation à passer une nuit à l'hôtel. Force est de constater que les copier/coller sans ordinateurs ne fonctionnent plus vraiment...

Au moment où j'écris ces mots maintenant, le train est à nouveau en marche. D'ici peu tout le monde ici présent pourra à nouveau échanger et discuter numériquement à propos de cette éprouvante mésaventure, sans pour autant relever le point le plus marquant et le plus instructif. Je me permettrais de le résumer par une simple question : « quelle durée de vie peut espérer un habitant ordinaire d'un pays industrialisé si, subitement, cette fantastique toile numérique venait à brutalement ne plus fonctionner ? »

Je vais arriver chez moi d'ici quelques minutes, mais je garderai de cet épisode une déduction importante : lorsqu'on parle de technologie, il y a dépendance et dépendance. L'une d'elles, celle insidieusement distillée dans ces pages, est réciproque et mutuellement profitable. L'autre, par contre, s'apparente plus à de l'esclavage et c'est celle que j'ai majoritairement vue à l'œuvre ce soir, presque mêlée d'une panique difficilement contenue chez certains alors que moi-même étais relativement serein (certes, j'avais dans mon sac un bon livre, mon laptop, quelques cartes, un paquet de câbles, un mobile avec une autonomie indécente comparée à celle d'un smartphone et un énorme morceau d'emmental).

Voyez-vous, lorsqu'on n'est pas curieux de comprendre et d'explorer, on finit enchaîné et contrôlé par nos dépendances. La plupart des personnes que j'ai vues ce soir ne savent pas « faire sans » mais, plus important encore, ne savent pas « faire avec » non plus. Et là, ça fait vraiment peur !

Accessoirement, voici une idée pour la SNCF (non, je ne vais pas critiquer ou râler, je suis maintenant chez moi au chaud avec mes chats, avec certes plus de 3h de retard, mais là dehors des gens vont travailler sans doute une bonne partie de la nuit sur la voie à réparer les dégâts dans le froid pour que je puisse prendre mon TER demain matin) : installez des prises de courant équipées de monnayeur dans les voitures. En réglant l'aspect légal (tout le monde ne peut pas revendre du courant), je pense qu'il y a là une opportunité très intéressante ;)

Denis Bodor

SOMMAIRE

Hackable Magazine
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIÈRE GRISE

B.P. 20142 – 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – Fax : 03 67 10 00 21
E-mail : lecteurs@hackable.fr
Service commercial : cial@ed-diamond.com
Sites : boutique.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

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : en cours
Commission paritaire : K92470
Périodicité : bimestrielle
Prix de vente :
7,90 €

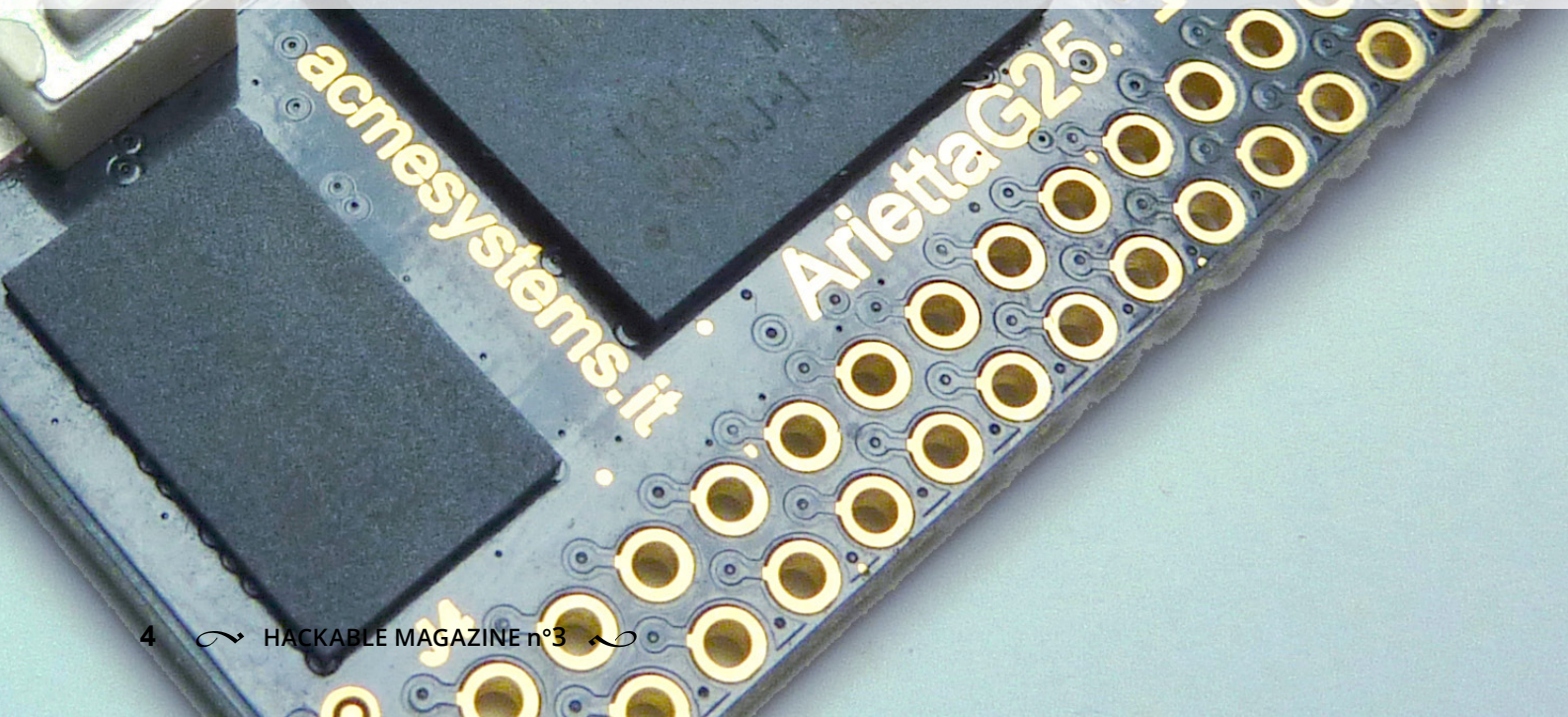
MIXTE
Papier issu de sources responsables
FSC® C015136

Suivez-nous sur Twitter
@hackablemag

ACME ARIETTA G25 : UN NANO-ORDINATEUR DE LA TAILLE D'UNE GOMME

Denis Bodor

On n'arrête pas le progrès, la preuve tout devient plus petit : les ordinateurs, les supports de stockage, le QI nécessaire pour suivre ce qui passe à la télé, les composants et bien entendu les plateformes qui permettent de faire fonctionner Linux. Dans la catégorie « petit bidule furieusement intéressant », le sujet qui nous intéresse ici est le module Arietta G25 du fabricant italien ACME Systems. Au menu, processeur 400 Mhz et 128 Mo de mémoire au format 5 x 2,5 cm…



L'Arietta G25 n'entre pas dans la même catégorie qu'une carte comme la Raspberry Pi ou la BeagleBone Black mais doit être vue comme une version survitaminée d'une Arduino Yùn ou une Intel Galileo. La principale différence entre l'une et l'autre catégorie concerne surtout le système d'affichage. L'ACME Arietta G25 comme la Galileo ne comprend pas de sortie HDMI, VGA ou composite. Ces systèmes sont destinés à être utilisé en *headless*, sans moniteur/clavier, au travers d'une liaison réseau ou série.

Pour certains projets, cette particularité est tout à fait appréciable car cela impacte directement la taille de la carte mais aussi son prix. L'Arietta G25 ne coûte que 20 euros qui est le prix d'une Arduino Leonardo par exemple. Ceci à la différence près que :

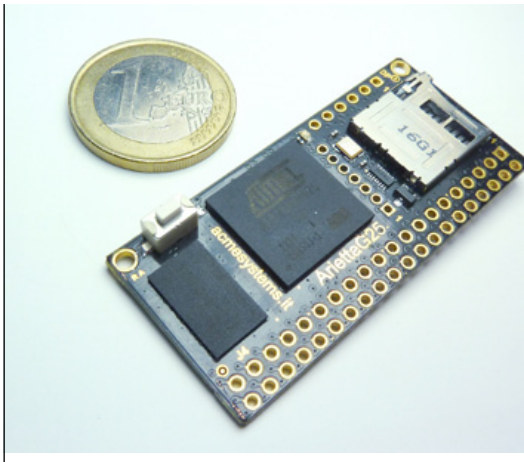
- L'Arietta G25 est trois fois plus petite,
- fait fonctionner un système Linux Debian très similaire au Raspbian de la Raspberry Pi,
- dispose d'un emplacement pour une micro SD jusqu'à 32 Go,
- peut accueillir en option un module Wifi 802.11 b/g/n (7 euro ou 14 avec l'antenne),
- comporte 29 GPIOs facilement accessibles, plus 3 bus USB, un connecteur pour une pile bouton pour la RTC et les habituelles fonctions (reset, 3,3 volts, 5 volts, masse).

Ces ports d'entrée/sortie représentent un point très important de cette carte car, selon leur configuration, on peut disposer ainsi de :

- 3 ports série (en plus de la console série),
- 2 bus i2c,
- un bus SPI avec jusqu'à 4 lignes CS,
- 4 sorties « analogiques » en PWM,
- 4 entrées analogiques (ADC 4 canaux),
- une sortie audio numéro (I2S),
- un bus 1-wire.

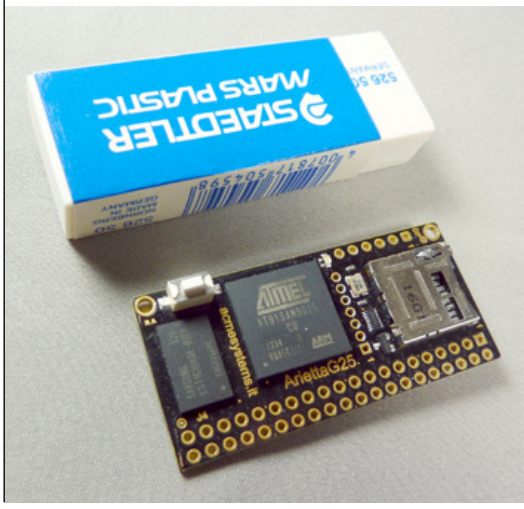
Bien sûr, il n'est pas possible d'activer toutes ces fonctionnalités en même temps (ADC et PWM par exemple sont sur les mêmes broches) et ceci consomme des entrées/sorties mais c'est tout à fait intéressant en terme de polyvalence pour des projets précis.

Vous l'avez compris, ce type de carte est un peu plus « technique » que ce qu'offre une Raspberry Pi et s'adresse donc à des utilisateurs ayant déjà pour habitude de chercher à exploiter à fond leur matériel. L'Arietta G25 reste cependant accessible et est d'ailleurs conçue dans ce sens à l'attention des « makers ». Sans pour autant devoir être un dieu en programmation ou un kaïd de Linux, quelques notions et une certaine expérience de la ligne de commandes seront indispensables. D'autres modules et cartes chez le constructeur sont bien plus orientés vers les applications industrielles comme l'Aria G25 qui utilise le même « cœur » Atmel ou la nouvelle ACQUA A5 (Atmel SAMA5D3x). J'en parle avec entrain car j'étais déjà utilisateur de l'Aria ainsi que les précédentes cartes comme la Fox G20 ou la vieille ACME Fox à processeur ETRAX de chez Axis (100 Mhz et 16 Mo de RAM pour plus de 150 euros mais ça, c'était il y a presque 10 ans).



L'Arietta G25 des italiens d'ACME Systems : processeur, mémoire, emplacement micro SD. Le tout sur un adorable petit circuit avec une myriade de GPIO

La preuve en image : l'Arietta G25 a bien la taille d'une gomme !



Côté ressource processeur c'est un SoC (*System on a Chip*) Atmel qui forme la base de la carte et plus précisément un AT91SAM9G25 à cœur ARM9 à 400Mhz. Celui-ci est accompagné de 128 Mo de mémoire (il existe une version 256 Mo à 30 euros). Ce SoC repose sur une architecture ARM926EJ-S utilisant un jeu d'instructions ARMv5TEJ (armel pour Debian) alors que le SoC Broadcom d'une Pi par exemple utilise de l'armv6l (armhf) et l'AM3358 (Cortex A8) de la Beaglebone Black de l'ARMv7-A (je ne parle même pas des monstres comme les Allwinner à multiples cœurs des Cubieboard). Derrière ce charabia technique il faut comprendre que le processeur de l'Arietta G25, avec ses 128 Mo de RAM est bien en dessous d'une carte comme la Raspberry Pi ou d'autres cartes de dernière génération. Elle ne se place d'ailleurs pas comme une concurrente mais comme quelque chose entre une Pi et une Arduino Yùn en terme de puissance.

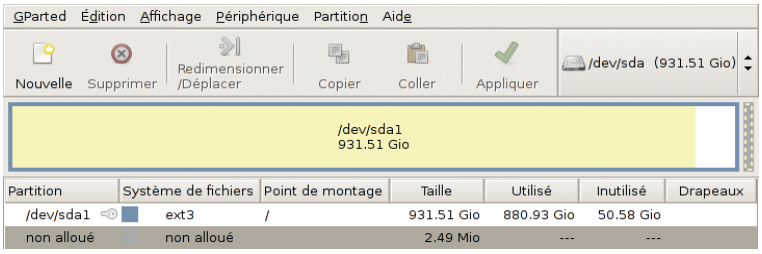
Enfin, un point que nous considérons comme très important, il n'est fait usage d'aucun élément propriétaire qu'il s'agisse du bootloader (le programme de démarrage comme le **`bootcode.bin`** sur la Pi) ou des éléments dans le noyau Linux. De plus, la documentation conséquente, typique d'Atmel et d'autres « fondeurs », est également un point appréciable car même si vous-même ne développez pas des fonctionnalités directement en rapport avec le processeur, elle permet aux développeurs Linux de faire leur travail efficacement (on remarquera d'ailleurs au passage qu'Atmel contribue activement au support et au développement du noyau).

1. PRÉPARATION DE LA MICRO SD

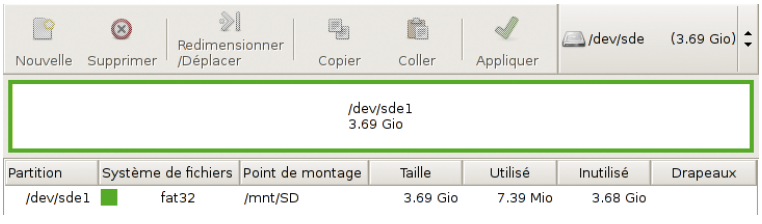
1.1 Créer les partitions

Tout comme La Raspberry Pi et contrairement à la Beaglebonne Black, l'Arietta G25 ne dispose pas de mémoire de stockage interne. Le système et les données trouvent donc leurs places sur une micro SD. Mais comme nous l'avons dit l'Arietta s'adresse à des utilisateurs un peu plus acharnés, la confection de la carte contenant le système est donc un peu plus technique que la simple copie d'une image sur une SD avec Win32DiskImager. Il sera nécessaire d'utiliser une machine sous GNU/Linux pour installer tout cela avec, dans l'ordre, la création des partitions, le montage de chaque système de fichiers, la copie des données et le démontage. Tout ceci devra être fait à la main.

Il vous faudra une carte micro SD de 2 Go minimum pour pouvoir installer le système de base proposé par ACME Systems. Nous partirons ici du principe que la carte en question est neuve et comporte donc déjà une partition au format FAT ou NTFS. Placez la carte dans un lecteur USB que vous connecterez à la machine ou utilisez l'emplacement adéquate si votre ordinateur en possède un. Lancez ensuite le logiciel GParted en tant que super-utilisateur (**`sudo gparted`** par exemple en ligne de commandes). Celui-ci va scanner les disques et partitions en présence puis vous afficher quelque chose comme ceci :



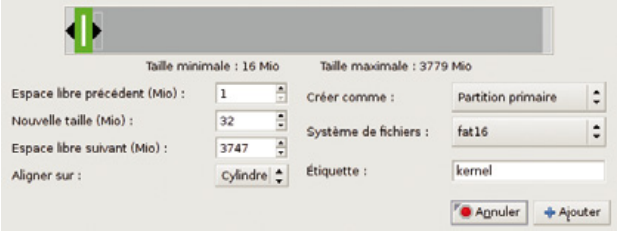
C'est là qu'il faut faire très attention ! Vous ne devez en aucun cas toucher à un autre disque que celui qui correspond à la micro SD. Dans GParted, vous pouvez utiliser le menu « GParted », « Périphériques » et choisir le bon disque ou encore, dans la barre de bouton, à droite, cliquer sur la liste des disques pour choisir celui qui vous intéresse. Les disques sont désignés par le pseudo-fichier correspondant dans **`/dev`** avec des noms comme **`sda`**, **`sdb`**, etc. Il est fait mention de leur taille ce qui facilite grandement le repérage du bon support.



Ici notre micro SD placée dans un adaptateur USB est `/dev/sde`. On constate que le support de 3,69 Go contient une partition en FAT32 qui occupe l'ensemble de l'espace disponible. C'est classique des cartes neuves. Vous pouvez cliquer sur la partition puis sur le bouton « Supprimer ». Les changements ne seront effectués que lorsque vous cliquerez sur le bouton « Appliquer ». Vous pouvez donc en toute sécurité procéder à l'ensemble des changements, vérifier que tout est correct pour enfin, en dernière étape, déclencher toutes les opérations qui modifieront effectivement le contenu du support.

Sélectionnez l'espace non alloué puis cliquez sur « Nouvelle » pour ajouter une partition. Le système que nous allons installer sur la carte est configuré pour utiliser 4 partitions. La première est, comme avec une Raspberry Pi, celle qui accueillera le programme de démarrage (*bootloader* de second niveau) et le noyau Linux. La suivante sera le système lui-même (*root filesystem*) puis une partition pour des données utilisateur et enfin une partition de swap (mémoire virtuelle).

La première partition que vous allez créer fait 32 Mo, commence au début du support, est de type FAT16 et a un label « kernel ». Ce qui nous donne, dans l'interface de GParted :

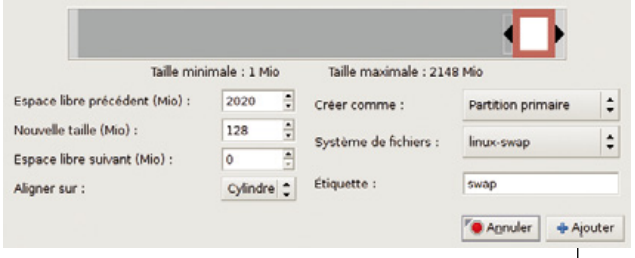


Vous ne pouvez pas choisir de changer ces partitions sans modifier les fichiers de configuration du système. Le noyau par exemple s'attendra à trouver le système sur la seconde partition. Mais vous pouvez changer leurs tailles respectives. Les indications données par ACME concernent une micro SD largement plus petite que notre carte de 4 Go. Nous allons donc changer la procédure du constructeur de manière à tirer partie de la totalité de l'espace que vous utilisiez un support de 4 Go, 8 Go, 16 Go ou 32 Go.

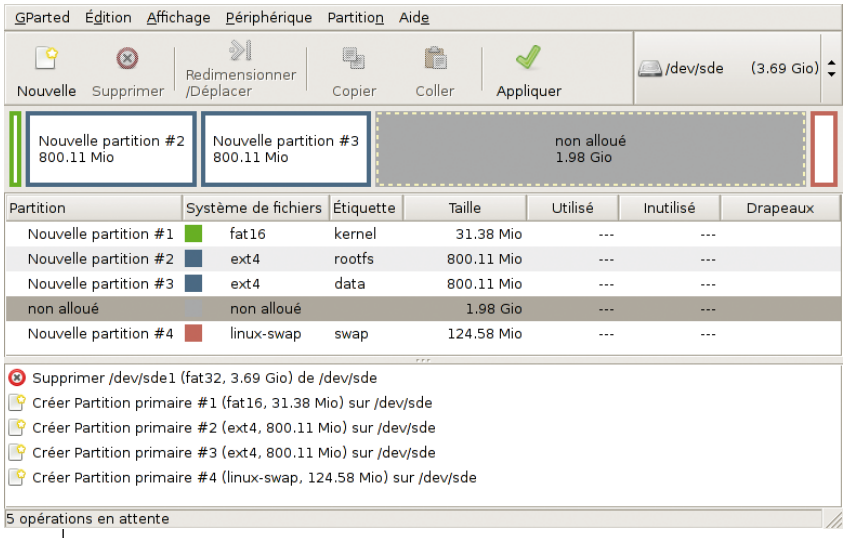
Ainsi nous devons un peu jongler pour conserver l'ordre des partitions tout en adaptant les tailles à nos besoins et sans avoir à faire de calculs. Nous allons donc créer la seconde partition à 800 Mo ainsi (type ext4, label « rootfs ») :

Puis nous passons à la troisième partition de 800 Mo également (type ext4, label « data ») :

Enfin, nous ajoutons la partition de swap mais prenons soin de la déplacer en fin du support en entrant tous les paramètres (taille 128 Mo, type linux-swap et label « swap ») et, avant de valider, il suffit de prendre le bloc et le glisser complètement à gauche dans la partie supérieure de la fenêtre :

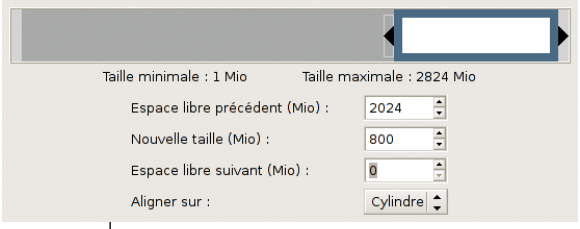


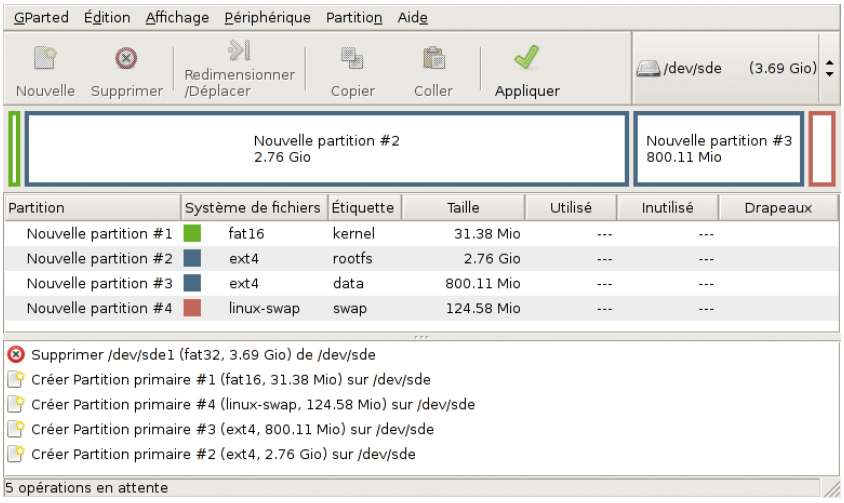
Le résultat qui s'affiche dans l'interface de GParted nous montre les trois premières partitions, un trou non alloué et les 128 Mo de zone de swap :



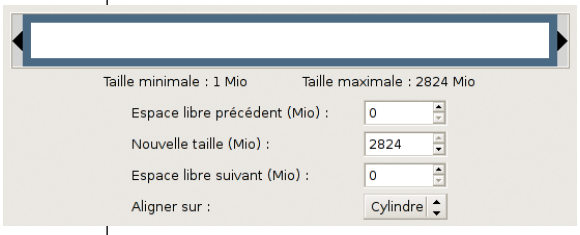
SD. Notre partition FAT16 fait toujours 32Mo, la partition de données 800 Mo et le swap de 128 Mo mais l'espace pour le système est passé de 800 Mo au maximum utilisable sur notre support, soit 2,76 Go. Nous n'avons plus qu'à appliquer les changements. Remarquez que ceux-ci sont listés dans la partie inférieure de l'interface de GParted, c'est la liste de choses à faire.

Ce que nous allons faire maintenant c'est déplacer la partition « data » pour la glisser tout contre celle de swap. Sélectionnez cette partition et cliquez sur le bouton « Redimensionner/Déplacer ». Prenez la partition dans la partie supérieure et glissez-là à droite :



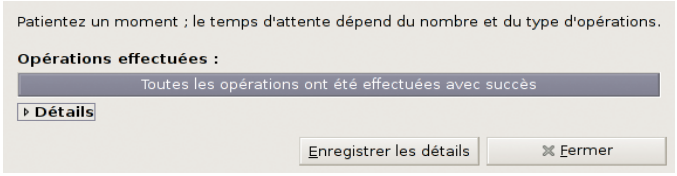


Notre espace non alloué est remonté d'un cran et se trouve maintenant entre les partitions 2 et 3 (« rootfs » et « data »). Vous pouvez alors sélectionner la seconde partition et utiliser le même bouton. Mais cette fois nous n'allons pas glisser la partition mais utiliser la petite poignée de droite pour l'étendre au maximum :



Dès la modification validée, nous avons à l'écran une image fidèle de l'organisation qui sera appliquée à la micro

Vérifiez que vous utilisez bien le bon support et cliquez simplement sur le bouton « Appliquer ». Confirmez l'action et Gparted applique les changements pour de bon, ce qui durera plus ou moins longtemps en fonction de la taille et de la vitesse du support. Gparted confirmera la bonne marche de l'ensemble par un petit message :



Bien entendu, strictement rien ne vous empêche de créer les partitions en indiquant des tailles que vous aurez calculées. La répartition que vous choisirez ne dépend que de vous et de deux critères importants : la première partition en FAT16 ne peut dépasser 2 Go et la partition de swap (la dernière) devrait faire au moins la taille de la mémoire. Le premier élément est dépendant du format FAT16 lui-même mais gardez à l'esprit que cette zone

ne sert qu'à stocker le noyau et d'autres petites choses, inutile de dépasser 64 Mo pour cela. La zone de swap permet au système de copier une partie de la mémoire sur le support pour libérer de la « vraie » RAM si le système est trop chargé. Sur un système embarqué, cela ne devrait même pas arriver car si la mémoire vient à manquer de manière récurrente ce n'est pas l'utilisation du swap qui arrangera la situation. En revanche, cette zone peut être utilisée pour mettre le système en hibernation profonde en copiant l'intégralité du contenu de la mémoire sur la zone de swap avant de s'éteindre. Au prochain démarrage, l'opération inverse est réalisée et le fonctionnement reprend son cours. C'est ainsi que cela fonctionne sur les ordinateurs portables sous GNU/Linux par exemple et, en installant quelques éléments logiciels, on peut faire de même sur n'importe quel nano-ordinateur ou module comme la Raspberry Pi ou l'Arietta G25.

1.2 Installer le système

Ceci va être un peu plus technique puisqu'il s'agit de désarchiver un système complet sur une carte micro SD ou plus exactement sur la partition « rootfs ». Là encore, il vous faudra utiliser GNU/Linux et en particulier la ligne de commandes, en tant que super-utilisateur. Les commandes utilisées restent relativement simples et nous allons voir ensemble comment réaliser tout cela étape par étape.

Si vous n'avez pas retiré la micro SD ou le lecteur USB du PC vous ayant servi pour le partitionnement et formatage de la carte, vous connaissez déjà les désignations précises de chaque partition. Dans notre cas il s'agit de **/dev/sde1** pour la partition FAT16 et **/dev/sde2** pour la racine du système. Si vous avez le moindre doute, il vous suffira d'utiliser la commande **dmesg** pour voir les messages du système dans lesquels doivent figurer ceux de la détection des partitions. Vérifiez plus d'une fois car si vous utilisez les commandes qui vont suivre sur le mauvais disque, vous allez endommager votre système.

Notez également qu'en fonction de la distribution que vous utilisez, il est parfaitement possible que le contenu de la micro SD soit directement accessible dans le gestionnaire de fichiers. Ceci signifie alors que le système à tout naturellement monté (mis à disposition pour vous) le contenu des différentes partitions et que vous n'avez donc pas besoin de le faire vous-même. Ubuntu, par exemple, fait cela. Dès l'insertion de la carte ou connexion du lecteur USB, vous aurez accès aux partitions via des sous-répertoires dans **/media**. Ces sous-répertoires sont nommés en fonction du label utilisé lors du formatage. Ainsi vous devriez trouver **/media/kernel**, **/media/rootfs** et **media/data**. En ligne de commandes, il vous suffit d'utiliser la commande **mount** et parmi les informations affichées, vous trouverez les lignes correspondantes.

Mais avant toutes choses, pointez votre navigateur web sur **http://www.acmesystems.it/binary_repository**. Là vous trouverez un lien « *Download from here these files* » qui vous enverra sur une page vous proposant de télécharger les fichiers **kernel.tar.bz2** et **rootfs.tar.bz2**. Il s'agit respectivement de ce qu'il faut décompresser sur la première partition (« kernel ») et la seconde (« rootfs »). Placez ces fichiers dans votre répertoire personnel.

Si votre système ne « monte » pas automatiquement les partitions de la micro SD, vous devez le faire vous-même. Pour commencer, créez deux répertoires dans votre répertoire personnel :

```
% mkdir kernel
% mkdir rootfs
```

Puis utilisez la commande **mount** pour rendre accessible ces partitions via les répertoires en question :

```
% sudo mount /dev/sde1 ./kernel
% sudo mount /dev/sde2 ./rootfs
```

La commande **mount** prend en argument le fichier qui représente la partition et le répertoire de destination. Vous devrez donc faire correspondre **sde1** et **sde2** avec les bonnes désignations pour votre système.

Il ne vous reste plus ensuite qu'à copier le contenu des archives dans les bons répertoires. On commence par le noyau et les autres éléments qui l'accompagnent :

```
% sudo tar --no-same-owner \
  -mxjvf kernel.tar.bz2 -C
/media/kernel
./acme-arietta.dtb
./boot256.bin
./boot.bin
./zImage
```

L'option `--no-same-owner` et `-m` sont là surtout pour éviter de désagréables messages d'erreur. Nous plaçons des éléments sur un système de fichiers qui n'a pas la moindre idée de ce qu'est le propriétaire d'un fichier puisque qu'il ne gère pas ce genre de choses. `tar` va donc « râler » en constatant qu'il ne peut pas définir ces caractéristiques. L'option `-C` en revanche est très importante car elle désigne l'endroit où placer les fichiers. Ici nous avons spécifié `/media/kernel` car c'est le répertoire correspondant sur Ubuntu. Bien entendu, si vous avez vous-même monté les partitions, c'est alors `./kernel` qu'il faut utiliser.

On passe directement au reste du système en faisant de même avec l'archive `rootfs.tar.bz2` :

```
% sudo tar -xvjpSf rootfs.tar.bz2
-C /media/rootfs
[...]
plein de lignes
plein plein plein !
[...]
```

Ici, inutile d'utiliser les options pour éviter les messages d'erreur. Le système de fichiers cible n'est pas en FAT16 mais en ext4 et il FAUT justement que les caractéristiques de chaque fichier soient correctement définies pour que le système fonctionne. Cette étape mettra un certain temps car il n'y a peut-être pas beaucoup de volume (environ 500 Mo) mais de nombreux fichiers.

Si vous avez vous-même monté les partitions, il est temps de les démonter :

```
% sudo umount kernel
% sudo umount rootfs
```

Ces deux commandes aussi peuvent prendre du temps car même si la copie semble terminée, ce n'est pas le cas. Il y a un cache en place dans le système vous permettant de rapidement reprendre la main lors des copies, par exemple. Mais si l'on veut détacher les données du système, alors celles-ci doivent être inscrites sur le support pour qu'il puisse être physiquement déconnecté. Il en va de même avec la fonction d'éjection que vous utiliserez si le système a monté tout seul les partitions. Surtout ne cliquez pas et retirez sans attendre le support, vous risquez de perdre une partie de ce qui devait être écrit sur la micro SD. Attendez que le support disparaisse de l'interface graphique.

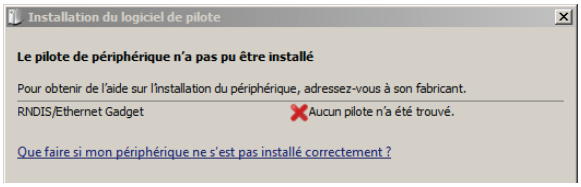
Si tout c'est bien passé et que vous avez suivi scrupuleusement ces indications, alors vous voici propriétaire d'une carte micro SD prête à être glissée dans l'emplacement dédié de l'Arietta G25.

2. PREMIER DÉMARRAGE

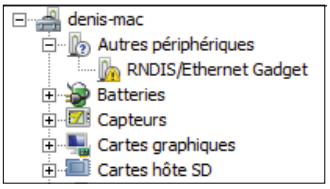
Sans écran, clavier ou câble de liaison série, comment dialoguer avec le système et nous assurer que tout fonctionne ? La réponse tient dans le connecteur USB de l'Arietta. Contrairement à ce que vous trouvez sur la Raspberry Pi, il ne s'agit pas là seulement d'un connecteur d'alimentation. En utilisant un câble USB comme celui pour votre smartphone (micro USB) et en connectant ainsi l'Arietta à votre PC, vous n'allez pas seulement alimenter le module, vous allez également voir apparaître une nouvelle interface réseau. Là encore, contrairement à ce qui existe avec la Raspberry Pi, la carte d'ACME dispose d'une interface USB OTG permettant au système de devenir un périphérique USB. Ceci signifie aussi que pour l'instant elle va apparaître comme une interface réseau mais qu'il est possible d'en faire un disque USB, un port série, une imprimante, un clavier/souris (HID) ou même un périphérique MIDI.

2.1 Depuis Windows 7

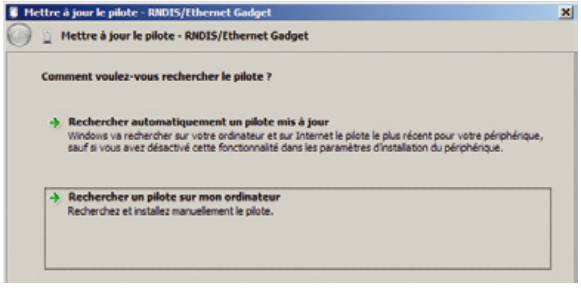
Une fois n'est pas coutume, sous Windows les choses risquent d'être un peu compliquées car il est fort peu probable que votre système trouve tout seul le pilote qui convient pour utiliser cette interface réseau un peu particulière. Pourtant, Windows sait parfaitement l'utiliser si on lui montre comment il doit faire. Ainsi à la connexion du câble USB, un périphérique « RNDIS/Ethernet Gadget » est bien détecté et Windows cherche un pilote. Après un long moment, il fini par vous dire qu'il n'a rien trouvé à installer et que vous devez vous débrouiller seul :



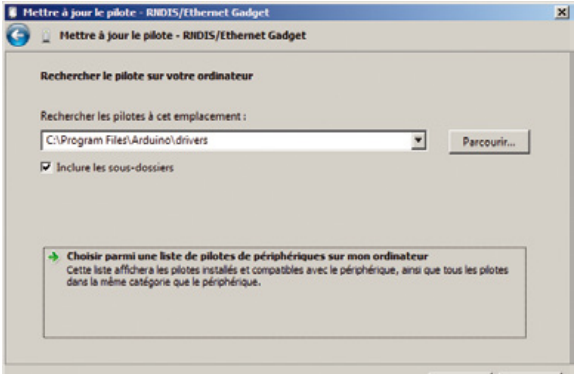
Rendez-vous donc dans le gestionnaire de périphérique où, bien entendu, le périphérique est marqué d'un symbole signifiant un problème. Cliquez avec le bouton droit et choisissez « Mettre à jour le pilote » :



Dans la fenêtre qui apparaît, deux choix sont proposés : la recherche automatique ou la sélection manuelle. La première solution n'ayant déjà rien donnée, choisissez « Rechercher un pilote sur mon ordinateur » :

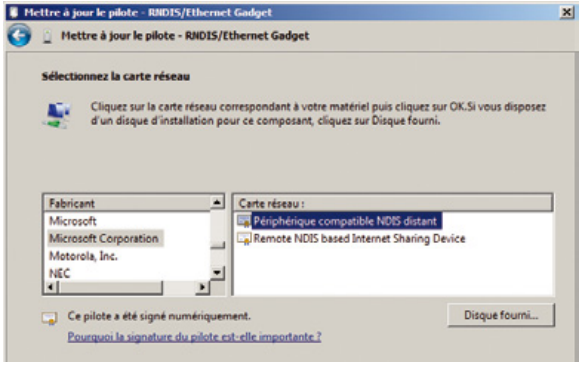


Dans la fenêtre suivante vous pouvez soit définir un emplacement pour le pilote, soit choisir parmi une liste, option que nous choisissons :

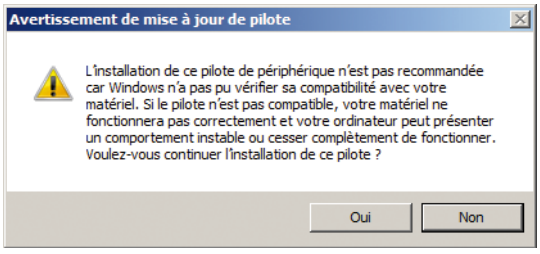


Dans la liste qui s'affiche, trouvez et sélectionnez « Cartes réseau » :

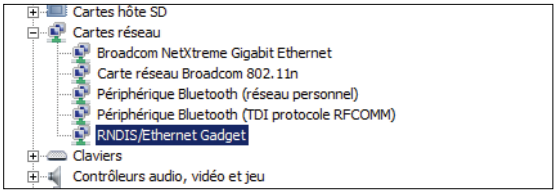
Parcourez la liste des fabricants jusqu'à « Microsoft Corporation » et dans la partie droite « Périphérique compatible NDIS distant » :



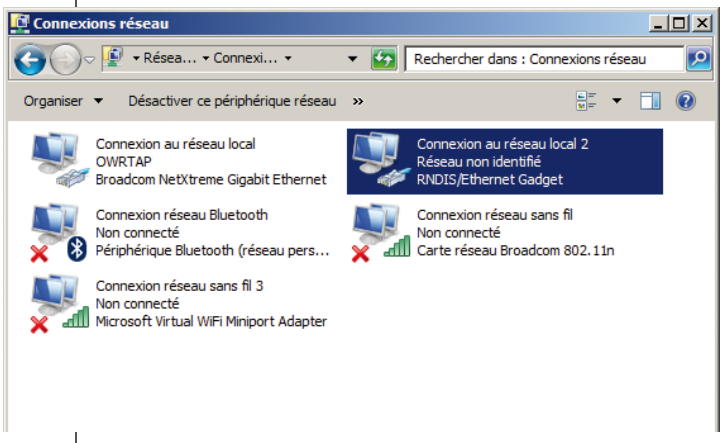
Bien entendu, Windows étant persuadé d'être plus malin que vous, il vous affiche un avertissement dès ce choix validé. Vous allez provoquer la fin du monde, attirer la foudre et votre ordinateur pourra présenter un comportement instable (plus que d'habitude, je suppose). Même pas peur ! On sait parfaitement ce qu'on fait, merci :



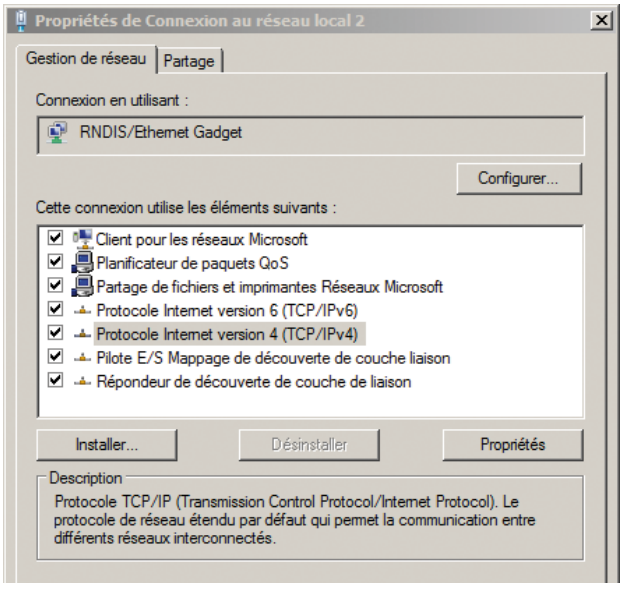
Naturellement, dans le gestionnaire de périphériques, notre matériel apparaît alors correctement :



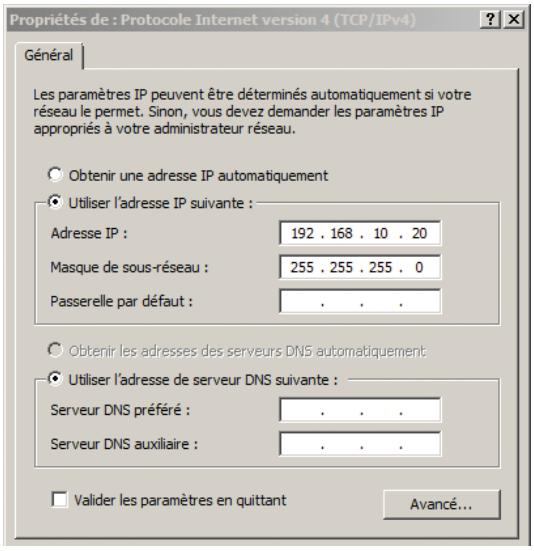
En faisant un tour dans le panneau de configuration à la rubrique « Centre réseaux et partage » puis « Modifier les paramètres de la carte » (par exemple), on constate également l'apparition de notre nouvelle interface. Elle est gérée comme n'importe quelle autre interface de ce type et de ce fait, par défaut, attend une configuration automatique.



Comme il ne s'agit que d'un lien entre deux systèmes, ceci n'existe pas et il faut alors configurer le réseau à la main. Par défaut le système sur la micro SD de l'Arietta G25 utilise l'adresse 192.168.10.10. Si nous configurons notre interface sous Windows en 192.168.10.20, par exemple, les deux systèmes seront sur le même réseau et pourront communiquer. Cliquez avec le bouton droit et choisissez d'afficher les propriétés de l'interface :

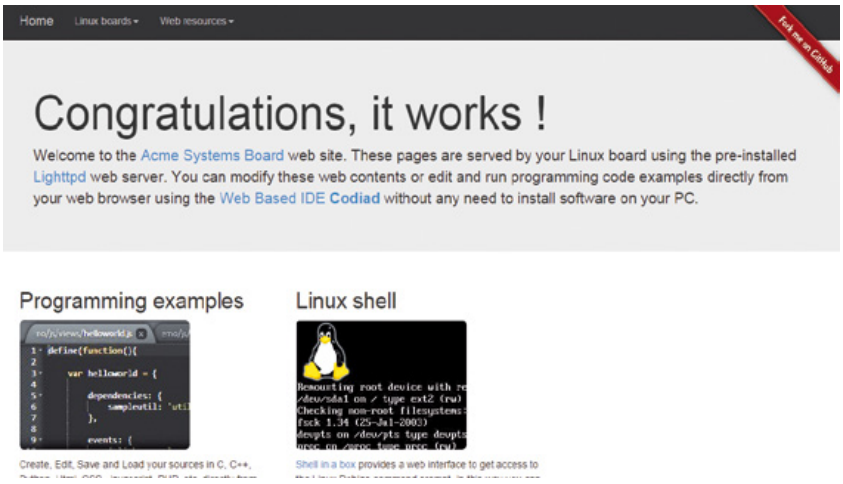


Dans la liste des éléments utilisés par la connexion, choisissez « Protocole Internet version 4 (TCP/IPv4) » et cliquez sur le bouton « Propriétés » :



Vous pourrez alors passer de « Obtenir une adresse IP automatiquement » à « Utiliser l'adresse IP suivante » pour ensuite remplir les différents champs comme sur notre capture. Validez avec « OK » chaque fenêtre et voici votre configuration terminée. Notez que cette configuration, étrangement, est lié au port USB où vous avez branché le câble. Si plus tard vous débranchez l'Arietta et la rebranchez sur le même port, la configuration sera utilisée. Si vous la branchez sur un autre port, il faudra tout recommencer, y compris l'installation du pilote (Ah, la logique Windows).

A ce stade le câble USB reliant votre PC au module Arietta G25 fait office de liaison réseau et vous pouvez utiliser un simple navigateur pour vérifier la bonne marche de l'ensemble. Pointez simplement celui-ci sur **http://192.168.10.10** et le serveur web intégré à l'Arietta répondra en affichant ceci :

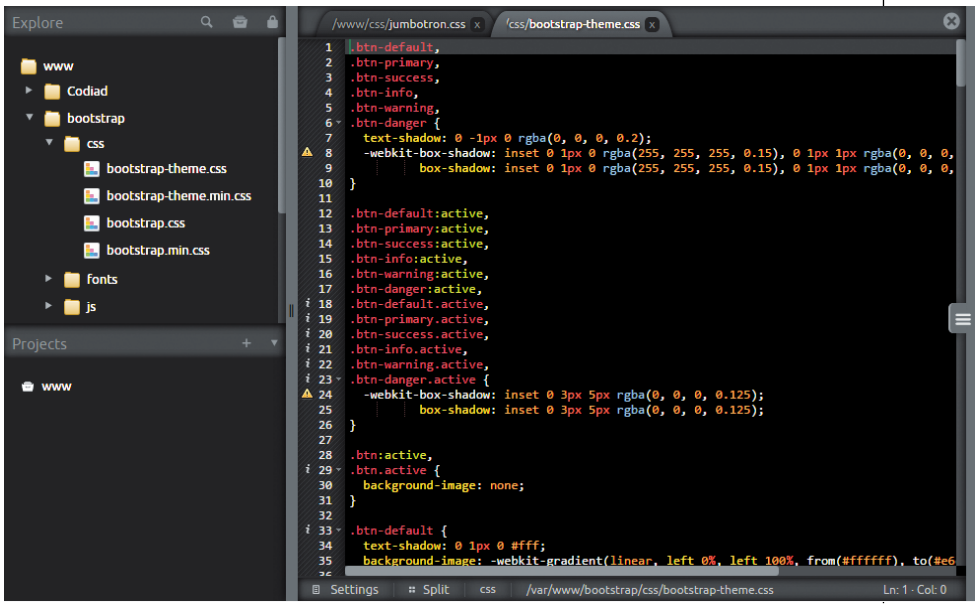


ACME Systems a bien fait les choses pour les débutants. Cette interface web permet par exemple d'accéder directement à la ligne de commandes grâce à *Shell In a Box*. Vous obtiendrez ainsi un accès au shell Linux de l'Arietta comme avec une console série ou une connexion SSH. Utilisez simplement le nom d'utilisateur **acme** et le mot de passe **acmesystems** et vous pourrez taper des commandes. Pour passer en super-utilisateur **root**, utilisez **su** (**sudo** n'est pas installé) et saisissez le mot de passe du **root** qui est également **acmesystems** par défaut.

La page web propose également une autre fonctionnalité intéressante sous la forme de l'environnement Codiad. Il s'agit, tout simplement, d'un environnement de développement (IDE) vous permettant de créer du code HTML, CSS, Javascript, PHP, Python, C ou C++ directement sur l'Arietta et ce dans un navigateur web. Vous pouvez vous y connecter avec le login **root** et le mot de passe **acmesystems** (notez que ce compte n'est pas celui de l'utilisateur du même nom, si vous changer le mot de passe **root**, ce sera toujours **acmesystems** qu'il faudra utiliser dans Codiad) (voir Figure ci-dessous).

Par défaut le projet ouvert est celui du site embarqué sur l'Arietta et donc la page web que vous voyez en utilisant votre navigateur.

Vous pouvez également accéder à l'Arietta via SSH en vous connectant avec un outil appelé PuTTY à l'adresse 192.168.10.10 ou avec **ssh** si vous avez installé les outils Cygwin sur votre Windows.



2.2 Depuis Linux

Avec GNU/Linux, les choses sont un peu plus simples puisque dès la connexion et le démarrage complet de l'Arietta, le système détecte automatiquement la présence d'une nouvelle interface réseau. Ceci apparaît clairement dans les messages système :

```
new high-speed USB device number 45 using ehci-pci
New USB device found, idVendor=0525,
idProduct=a4a2
New USB device strings: Mfr=1, Product=2,
SerialNumber=0
Product: RNDIS/Ethernet Gadget
Manufacturer: Linux 3.16.1+ with atmel_usba_udc
cdc_eem 2-2.4.4:1.0 usb0: register 'cdc_eem' at
  usb-0000:00:1d.7-2.4.4, CDC EEM Device,
66:4c:95:af:72:d8
```

ainsi que dans la liste des interfaces réseaux :

```
% cat /proc/net/dev
Inter-|   Receive
face  | bytes    packets errs drop[...]
    lo:   15155      80    0    0[...]
  usb0:       0       0    0    0[...]
  eth1:       0       0    0    0[...]
  eth0: 127740283   91519  0    0[...]
```

usb0 est notre nouvelle interface et si nous voulons que tout fonctionne rapidement, il nous suffit d'utiliser la commande **`sudo ifconfig usb0 192.168.10.20`** ce qui aura pour effet de configurer l'interface **`usb0`** avec l'adresse IP 192.168.10.20, rendant ainsi accessible le système à l'autre bout du câble qui, lui, utilise l'adresse 1925.168.10.10.

Dès cette commande validée, il suffit d'utiliser un navigateur web pour se rendre sur http:/192.168.10.10 et ainsi avoir accès à une ligne de commandes via *Shell in a box* et à l'environnement de développement Codiad. On pourra également utiliser la commande **`ssh`** avec **`ssh root@192.168.10.10`** (mot de passe par défaut **`acmesystems`**) pour avoir accès en tant que super-utilisateur.

Sous GNU/Linux, il est également bien plus simple de donner un accès à internet à l'Arietta de cette manière (via le câble USB). Il s'agit, en quelques commandes, de dire à votre système de tout simplement relayer les communications comme si elles venaient de votre PC. Ceci se fait avec ces commandes saisies en tant que super-utilisateur :

```
% sudo -s
% echo 1 > /proc/sys/net/ipv4/ip_forward
% iptables -t nat -A POSTROUTING \
-o eth0 -j MASQUERADE
```

Notez que c'est là une façon très basique de procéder à ce relais de connexion, le but n'étant pas de configurer l'ensemble de manière définitive mais de rapidement fournir à l'Arietta une connectivité Internet. Dès lors vous pourrez utiliser la ligne de commandes pour, par exemple, mettre à jour le système, ou installer de nouveaux paquets (avec **`apt-get`** comme sur une Pi).

3. ALLER UN PEU PLUS LOIN ?

Il y a encore beaucoup de choses à dire sur l'Arietta G25 et son utilisation mais cet article est d'ors et déjà très (trop ?) long et technique. Ajoutons simplement une méthode permettant d'intégrer le support Wifi si vous avez retenu l'option d'acquérir une Arietta accompagnée du module USB Wifi RaLink RT5370N.

Tout comme pour le reste de la configuration, tout cela se fera à la main via la ligne de commandes, directement sur le système de l'Arietta. Les éléments logiciels sont déjà présents dans le système mais vous devrez définir la configuration adéquate. Vous devrez, tout d'abord, créer un fichier contenant les informations vous permettant de vous connecter à votre point d'accès Wifi. Créez un fichier, avec **`nano`** par exemple, appelé **`maison.conf`**, dans **`/etc/wpa_supplicant`** et contenant :

Comparaison de taille entre un Arduino Leonardo et nos Arietta. Ce n'est pas la taille qui compte mais les fonctionnalités et la manière de s'en servir

```
network={
  ssid="le_ssid_de_votre_AP"
  psk="la_cle_wpa"
  priority=5
}
```

WPA supplicant qui gère les mécanismes d'authentification Wifi sous Linux est maintenant capable de se connecter au point d'accès **le_ssid_de_votre_AP** avec le mot de passe **la_cle_wpa**. Mais ce n'est pas tout, nous devons également dire au système qu'il doit utiliser cette connexion. Ceci se fait en éditant le fichier **/etc/network/interfaces** qui contient déjà les informations du support réseau par USB :

```
auto lo
iface lo inet loopback
  pre-up modprobe g_ether

auto usb0
  iface usb0 inet static
  address 192.168.10.10
  netmask 255.255.255.0
  gateway 192.168.10.20
```

Ajoutez alors, ceci dans le fichier :

```
auto wlan0
iface wlan0 inet dhcp
  wpa-conf /etc/wpa_supplicant/maison.conf
```

Afin de ne pas travailler sans filet en cas de problème, ne supprimez pas immédiatement les lignes concernant l'USB mais placez simplement un **#** devant la ligne **gateway 192.168.10.20**.

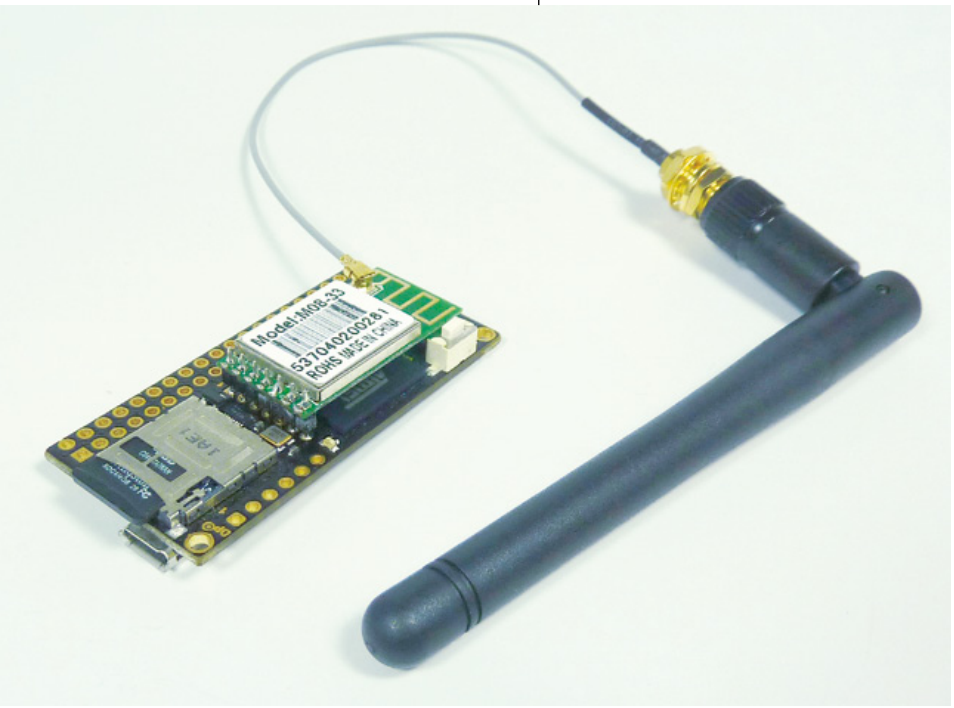
Ceci évitera au système de tenter de passer par le PC connecté pour atteindre Internet. Il passera alors via le Wifi et votre box, tout en laissant intact l'accès via l'adresse 192.168.10.10 (serveur web, *Shell in a box*, accès SSH).

4. POUR CONCLURE, PUISQU'IL FAUT

Pour poursuivre l'aventure et votre futur histoire d'amour avec cette chère Arietta, le plus simple est encore de simplement pointer votre navigateur web sur le site d'ACME Systems (**http://www.acmesystems.it/arietta**). Les différents tutoriels sont en anglais et requièrent une certaine expérience dans l'utilisation de Linux et de la ligne de commandes. Vous y trouverez des explications sur l'utilisation des différents ports du module ainsi que ces exemples concrets de mise en oeuvre, allant jusqu'à la création d'un système Debian complet à partir de rien.

Comme nous l'avons signalé d'entrée de jeu, l'Arietta G25 reste une plateforme certes économique mais relativement difficile à appréhender pour un débutant n'ayant pas fait tout d'abord ses classes sur une carte plus accessible comme la Raspberry Pi. ACME Systems reste une société produisant des cartes et des modules avant tout destinés aux professionnels et aux industriels, mais avec un vrai esprit d'ouverture vers les bidouilleurs et les hobbyistes. L'Arietta est ainsi présentée comme un « *Tiny, cheap and easy ARM9@400Mhz Linux Embedded module for DIY and Maker cultures* » mais elle est également susceptible d'intéresser des industriels et des bureaux d'étude.

On réalise alors qu'il n'est pas si aisé de rendre accessible une plateforme au plus grand nombre, quelque soit leur niveau d'expérience dans le domaine. L'esprit très « pro » et « tech » d'ACME transparaît ainsi malgré les efforts fait pour essayer de faciliter au maximum la prise en main de leur nouveau produit. « Chose qui n'est pas un mal » diront certains et il est vrai qu'on ne peut pas tout avoir, la simplicité et la technicité. C'est une question de choix et de cahier des charges. **DB**



Une Arietta G25 équipée de son module Wifi soudé avec antenne externe. La carte ne possédant pas de port Ethernet, l'ajout du Wifi est relativement indispensable pour la plupart des projets.

DÉCOUVREZ NOS NOUVELLES OFFRES D'ABONNEMENTS !

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

boutique.ed-diamond.com

1 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 ?

ACCÈS À LA BASE DOCUMENTAIRE

Disponible en 2015 !

EXEMPLE

6N° HACKABLE (PAPIER + PDF 1 LECTEUR)
+ ACCÈS 1 AN BASE DOCUMENTAIRE OS (1 CONNEXION)

TARIF NORMAL	ÉCONOMIE	OFFRE SPÉCIALE
196,20 €	77,80 €	119,-€

2 LES COUPLAGES AVEC NOS AUTRES MAGAZINES :

EXEMPLE

6N° HACKABLE (PAPIER)
+ 4N° OPEN SILICIUM (PAPIER)

TARIF NORMAL	ÉCONOMIE	OFFRE SPÉCIALE
83,40 €	18,40 €	65,-€

EXEMPLE

6N° LINUX PRATIQUE (PAPIER)
+ 11N° GNU/LINUX MAGAZINE (PAPIER) + 6N° MISC (PAPIER)
+ 6N° HACKABLE (PAPIER) + 4N° OPEN SILICIUM (PAPIER)

TARIF NORMAL	ÉCONOMIE	OFFRE SPÉCIALE
271,12 €	71,12 €	200,-€

MBED : UNE APPROCHE UN PEU DIFFÉRENTE D'ARDUINO

D. B.

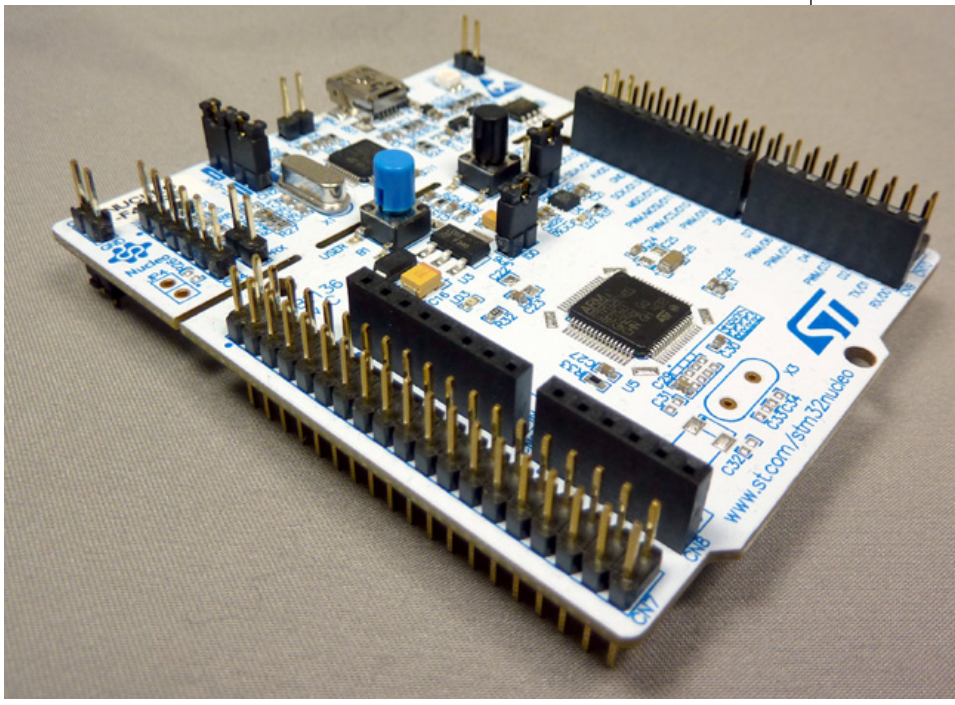
Arduino a conquis bien des utilisateurs et est entrain de chambouler bien des domaines où jusqu'alors étaient utilisés des circuits et des montages dédiés, fabriqués en petite quantité et donc bien plus onéreux qu'une plateforme très polyvalente. Ceci n'a pas manqué de susciter de l'intérêt de la part de l'ensemble des fabricants voulant, eux aussi, leurs « écosystèmuino ». Une voie possible est la copie pure et simple du concept jusqu'au look de l'IDE. Une autre consiste à comprendre les fondements du phénomène et d'en créer une évolution. La plateforme mbed a opté pour cette solution.

Arduino sera toujours Arduino et ce, pour deux raisons principales. La première est tout simplement que cette initiative a été la première à se populariser massivement et à avoir su capter l'intérêt des utilisateurs. C'est un bien et un mal. En effet, cela permet certes de former une communauté autour de critères techniques fixes et stables mais limite les tentatives d'explorations en dehors des sentiers battus. Ce contre-coup a un effet sur l'ensemble de l'écosystème de l'électronique numérique hobbyiste mais aussi sur Arduino lui-même. L'introduction de l'Arduino Due n'a pas déchaîné les passions, les utilisateurs préférant, pour leurs projets, miser sur une valeur sûre comme l'Arduino Uno, pourtant largement plus limité que la nouvelle Due en terme de puissance, de ressources et de fonctionnalités. Espérons que la future carte Arduino Zero ne connaisse pas le même destin d'outsider que la Due. On ne peut pas éternellement rester dans un univers 8 bits avec 32 Ko de flash et 2 Ko de RAM. Le moment viendra où les AVR coûteront plus cher à produire que des microcontrôleurs à cœur ARM Cortex-M.

La seconde raison est plus « administrative » puisqu'il s'agit du fait que le terme « Arduino » ne peut tout simplement pas légalement être utilisé pour un projet autre qu'Arduino, simple question de marque déposée. N'en soyez pas étonné, il en va de même pour Firefox. Le nom d'un « produit » véhicule le niveau de qualité que ses développeurs y placent. Si tout le monde peut faire des navigateurs ou des cartes et les appeler à loisir Arduino ou Firefox comment ferions-nous pour savoir qui fait quoi ? Même Linux est une marque déposée (par Linus Torvalds) et Richard Stallman lui-même n'a pas vu un quelconque problème en cela.

Cependant, ceci n'empêche pas l'existence de cartes et d'environnements de développement (IDE) très similaires comme :

- Freeduino, Seeeduino, Roboduino, InduinoX, Diavolino... La liste est longue et il s'agit généralement de cartes reposant sur le même microcontrôleur Atmel AVR, complétées tantôt d'éléments habituellement proposés sous forme de *shields* (Ethernet, Wifi, Bluetooth, etc) et parfaitement compatibles avec l'environnement Arduino.
- Texas Instrument et ses *Launchpads* pouvant reposer sur l'IDE Energia qui reprend les bibliothèques standards (*Wiring*) utilisées par Arduino et adaptées aux microcontrôleurs de Ti comme la série MSP430. L'environnement est très similaire en dehors de la couleur dominante qui passe du vert/bleu au rouge (logique puisque c'est une adaptation des sources originales).
- Leaflabs Maple est une carte utilisant un microcontrôleur STM32 mais également un environnement basé sur celui de l'Arduino. Cette fois l'environnement est vert pomme et là aussi le code a été adapté pour que la majorité des croquis puisse être utilisé.



- chipKIT Uno32, chipKIT Max32 ou encore Pinguino sont des cartes utilisant des composants PIC de Microchip (principal concurrent d'Atmel et ses AVR équipant les Arduino). Là encore le principe est le même avec une adaptation des éléments logiciels (Wiring) et de l'environnement de programmation. Mais cette fois pas de couleur dominante (du moins pour le Pinguino), l'interface est grise mais largement étoffée d'un grand nombre de fonctionnalités et de boutons... multicolores.

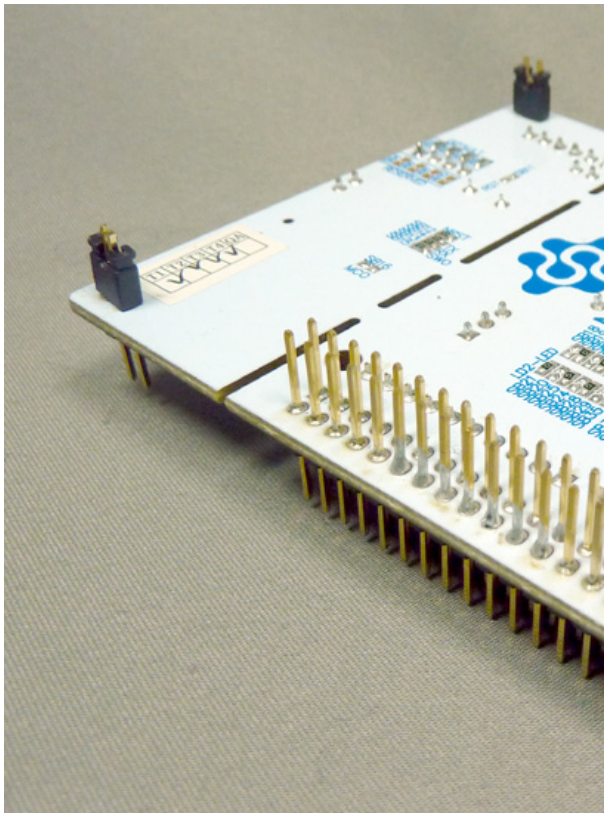
D'autres cartes, comme la Freedom de Freescale ou certains modèles d'Atmel ou STMicroelectronics, ne se préoccupent pas de créer un environnement de développement accessible mais intègrent une série de connecteurs dis « compatible Arduino » permettant d'utiliser directement les *shields* existant. Là, la compatibilité est purement matérielle mais l'idée d'approcher un secteur fertil est belle et bien présente.

En 2012, Renesas, autre fabricant important dans le milieu de l'électronique, introduit au Japon un produit dans le même esprit : La Sakura Board. Celle-ci est peu connue en occident comme bon nombre de produits qui ne sont tout simplement diffusés qu'auprès du public insulaire (pour les produits « technologiques » le marché japonais se suffit souvent à lui même). En dehors de son aspect incroyablement nippon et *girly* (à quand une famille de PokéBoard incluant une PikapikaBoard jaune ?) et de ses fonctionnalités très intéressantes (96 Mhz, 1 Mo de flash, USB, Ethernet), cette carte innove avec un principe ingénieux. Pourquoi devoir utiliser un système de programmation complexe (avec ou sans bootloader) si on peut simplement voir la carte comme une clé USB ? En d'autres termes, pourquoi ne pas s'arranger pour que la carte apparaisse comme un disque externe sur lequel copier le programme compilé comme un fichier ? Et c'est sans doute là toute la force du concept, parce que dans ce cas précis, plus rien n'interdit alors de mettre l'environnement de développement dans un navigateur web, la dépendance vis-à-vis de l'ordinateur utilisé disparaît tout simplement ! N'importe quelle machine capable de lire un disque USB et de se connecter à un site web devient une option pour programmer. Le markéteux fanatique hurlera sans doute hystériquement « c'est la magie du cloud ! » avant de retourner conspirer avec le prince des ténèbres pour vous vendre quelque chose dont vous n'avez pas besoin.

Ce principe de programmation est également celui de la plateforme mbed, à une différence près : La Sakura Board est une carte et un environnement, mbed c'est aujourd'hui un seul environnement web mais quelques 37 modèles de cartes compatibles, issues de 13 fabricants et reposant sur une demi douzaine de types de microcontrôleurs (tous basés sur une architecture ARM Cortex-M). On notera au passage que le SDK du projet est open source (depuis février 2013) et que l'ensemble est directement chapeauté/géré par ARM.

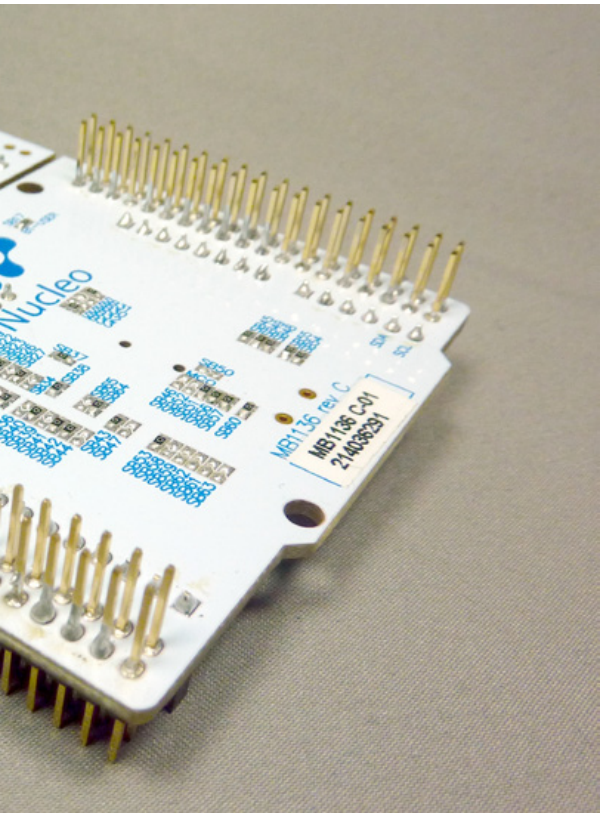
1. MBED FINALEMENT C'EST QUOI ?

Grossièrement et pour résumer les choses à l'aspect purement pratique, mbed est un environnement de développement qui prend place dans votre navigateur web. Vous pouvez donc y gérer vos projets et y écrire votre code. Si vous êtes déjà utilisateur de plateforme Arduino, le dépaysement ne sera pas trop important. Il vous faudra cependant oublier la notion de « croquis » car mbed se conforme aux principes de programmation standards. Il n'y a donc pas de `setup()`



et de `loop()` mais un unique `main()`, mais en dehors de ce point particulier le reste est majoritairement du C/C++.

La particularité de mbed est de ne nécessiter absolument aucune installation sur votre machine, sauf si vous comptez mettre à jour votre carte compatible mbed. Les fichiers qui composent un projet n'existent que « dans le cloud ». Certes ceci peut paraître peu rassurant, exactement comme pour d'autres services (images, vidéos, documents, mails, etc), mais présente un avantage non négligeable : où que vous soyez, un simple nom d'utilisateur et un mot de passe vous permettent de reprendre vos programmes tels que vous les avez laissés lors de la session précédente. Vous pouvez donc à loisir commencer un projet sur une machine pour le poursuivre sur une autre, même si celle-ci est celle d'un ami ou une machine publique (la connexion est sécurisée via HTTPS).



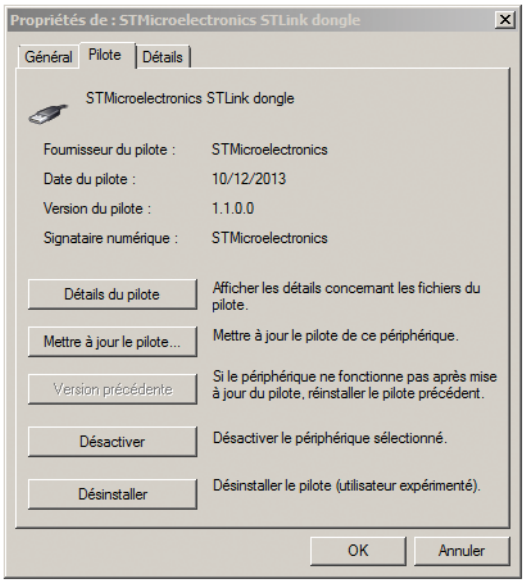
Une fois le code prêt, la compilation se passe également du côté du serveur web. Vous n'avez qu'à déclencher la construction et un firmware pour votre carte au format binaire (`.bin`) vous est proposé au téléchargement. Si vous avez connecté votre carte au PC, celle-ci apparaît comme un disque USB et vous n'avez qu'à y copier le fichier. La carte se programme et votre code est exécuté.

Mais mbed est également une communauté. En effet, comme le code de chaque utilisateur se trouve stocké côté serveur, il devient très facile de le partager. Ainsi, un grand nombre d'utilisateurs, une fois leurs projets terminés et stable, décident de le mettre à disposition de tous. Ceci peut prendre la forme de programmes complets ou de bibliothèques permettant de simplement ajouter des fonctionnalités identiques à vos propres projets. Pour l'heure quelques 300 bibliothèques sont ainsi disponibles et presque 8000 programmes. Ce système de partage communautaire est totalement intégré à la plateforme et vous pouvez tout aussi simplement importer le travail d'un autre utilisateur que diffuser le votre, directement dans l'interface de développement. S'ajoute à cela une gestion de versions vous permettant de faire évoluer et de corriger vos programmes ainsi qu'un système de commentaires complet facilitant la communication entre utilisateurs.

2. UN EXEMPLE AVEC LA NUCLEO F401RE

Pour faire connaissance avec tout cela, la meilleure solution est encore la pratique. Quelques 37 cartes sont actuellement supportées par mbed et nous avons du faire un choix. Celui-ci s'est porté sur une carte Nucleo F401RE de STMicroelectronics pour une raison très simple : nous l'avions déjà, car acquise il y a quelques mois dès lors que sa disponibilité fut annoncée (qui a dit « achat compulsif » ?). Cette carte utilise un microcontrôleur puissant basé sur de l'ARM Cortex-M4, le STM32F401RET6. Au menu, processeur à 84 Mhz avec FPU (calcul en virgule flottante), 512 Ko de mémoire flash, 96 Ko de SRAM, convertisseur analogique/numérique 10 canaux, 10 timers, 3 bus i2c, 3 USART (ports série), 4 bus SPI, accès SDIO, port USB 2.0 et vraiment beaucoup d'entrées/sorties, le tout pour quelques 12 euros TTC.

La carte propose un connecteur compatible Arduino permettant d'utiliser des *shields* ainsi qu'une autre série de broches, mâles cette fois, appelée *Morpho header* proposant les mêmes signaux auxquels s'ajoutent ceux inexistant dans le monde Arduino. Enfin, la



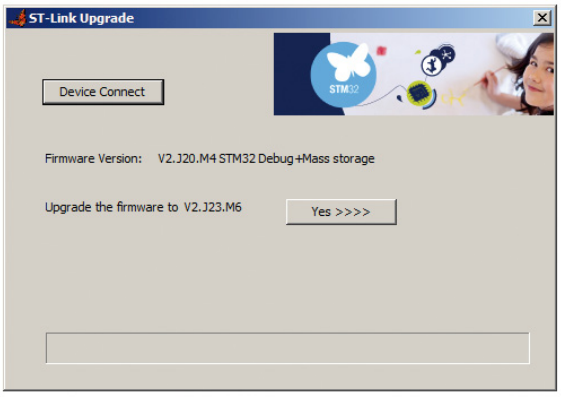
Ce n'est qu'en installant le pilote ST-Link que vous pourrez accéder à toutes les interfaces de la carte Nucleo et ainsi procéder à sa mise à jour

carte une fois connectée à un ordinateur apparaît non seulement comme un périphérique de stockage typique de mbed mais également comme un port série (pour la communication comme avec un Arduino) mais surtout, en plus, comme une interface ST-Link. Cette dernière est celle d'un programmeur pour microcontrôleur STM32 pouvant être utilisée par des outils professionnels et/ou open source pour manipuler les données en mémoire flash et pour inspecter l'exécution d'un programme. Les cartes de la famille Nucleo s'adressent ainsi aussi bien aux utilisateurs de mbed qu'aux développeurs souhaitant utiliser des outils standards comme GCC et OpenOCD.

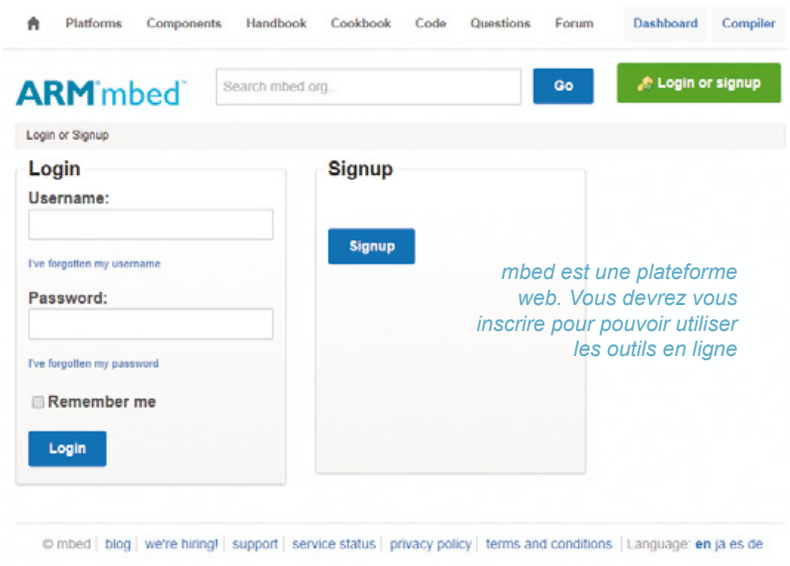
Toutes ces interfaces pour développeurs sont regroupées physiquement sur la carte sous la forme d'une zone construite autour d'un microcontrôleur STM32F103. En tant qu'utilisateur mbed, ce n'est pas sensé être votre affaire puisque vous vous intéresserez normalement à l'autre côté de la carte permettant d'utiliser les ressources du STM32F401. Cependant, la première chose à faire avant de jouer les explorateurs est de mettre à jour ces interfaces et donc le firmware du STM32F103. Ceci devra se faire sous Windows puisque les outils mis à disposition par ST n'existent pas pour les autres systèmes (ou alors il faut jongler).

Pointez votre navigateur sur **http://developer.mbed.org/teams/st/wiki/ST-Link-Driver** afin de récupérer le pilote ST-Link. En effet, bien que la carte apparaisse dès sa connexion comme un disque USB, l'interface de programmation ST-Link n'est pas prise en charge par défaut par Windows. Le fichier **`stlinknucleodriversigned.zip`** devra être désarchivé afin de pouvoir lancer **`dpinst_x86.exe`** ou **`dpinst_amd64.exe`** selon que vous ayez un Windows 32 ou 64 bits. Une fois les pilotes installés la connexion de la carte affichera toujours un disque de 512 Ko mais le gestionnaire de périphérique fera également apparaître un « STLink dongle ».

Il faudra alors vous rendre sur **http://developer.mbed.org/teams/ST/wiki/Nucleo-Firmware** pour récupérer la dernière version du firmware (V2.J23.M6 à cette date) sous la forme, là encore, d'un fichier Zip (**`stlinkupgradev2j23m6.zip`**). Cette archive contient le programme **`ST-LinkUpgrade.exe`** qu'il vous suffira de lancer. L'interface très simpliste se limite à deux boutons. Cliquez tout d'abord sur « *Device Connect* » pour que l'outil puisse accéder au périphérique (le F103 au lieu du F401) et vous afficher la version du firmware actuellement en place. Après quelques secondes et une déconnexion/connexion automatique du périphérique (qui change de mode), les informations devraient s'afficher et vous pourrez cliquer sur le bouton « *Yes>>>>>* » pour mettre à jour. Une barre de progression indique l'état de



L'outil ST-LinkUpgrade permet de mettre à jour automatiquement le firmware de la partie développeur de la carte, corrigeant ainsi un certain nombre de bugs parfois problématiques (liaison série instable par exemple)



l'opération avant d'afficher simplement « *Upgrade is successful* ». Bravo, votre Nucleo est à jour. Notez qu'à la validation du message la carte va encore une fois se déconnecter et se reconnecter, vous présentant alors le message d'exécution automatique habituel à la connexion d'un lecteur externe sous Windows.

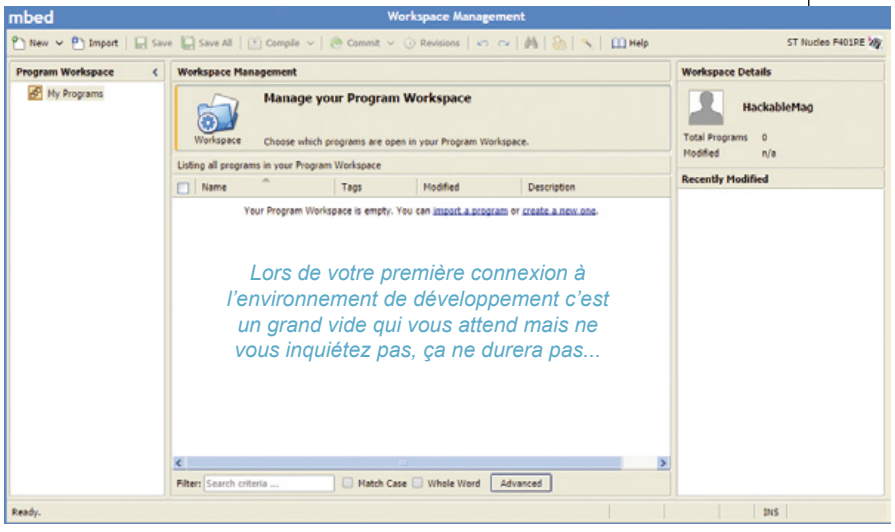
Vous voici prêt à utiliser votre carte compatible mbed. Dans le disque ayant fait son apparition se trouve un fichier **`mbed.htm`** sur lequel il vous suffit de double-cliquer pour lancer le navigateur par défaut. Ceci vous redirigera automatiquement sur la page de connexion de **http://developer.mbed.org**, où vous pourrez choisir « *Signup* » afin de vous créer un compte.

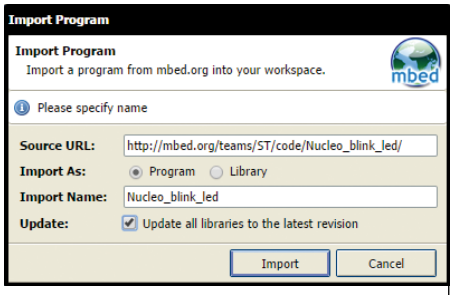
Sur la page d'inscription vous serez invité à entrer une adresse mail, un nom d'utilisateur, un mot de passe et vos nom et prénom. Notez que le type de carte que vous utilisez est automatiquement défini puisqu'il est spécifié dans le fichier HTML sur la carte (ceci ressemble même à un numéro de série, je ne suis pas sûr d'apprécier). Il vous faudra accepter les conditions d'utilisation après en avoir, bien entendu, studieusement pris connaissance (oui, je sais, mais en principe il faut lire le texte). Vous serez automatiquement connecté à votre compte après sa création mais devrez confirmer votre adresse mail en cliquant sur le lien se trouvant dans le message envoyé par mbed.

Il ne vous reste plus qu'à cliquer en haut à droite sur « *Compiler* » pour accéder à votre environnement de développement. Vous noterez que l'interface est entièrement en anglais ce qui peut s'avérer problématique pour certains utilisateurs. Cependant l'ensemble de l'environnement est assez basique et il est donc possible de s'en sortir aisément sans maîtriser la langue de Shakespeare (expression consacrée mais stupide puisque l'anglais de cet auteur était bien différent de l'anglais moderne).

Le principe de fonctionnement de l'environnement est assez similaire à n'importe quel application du même genre (type Eclipse par exemple) mais un peu plus riche que l'environnement Arduino. La notion d'espace de travail (*workplace* en anglais) est celle d'Eclipse. Vous retrouvez ainsi l'ensemble de vos projets sur la partie gauche de l'interface. La partie centrale est dédiée à l'élément le plus important (code, importation, listes, recherches) et celle de droite aux informations complémentaires.

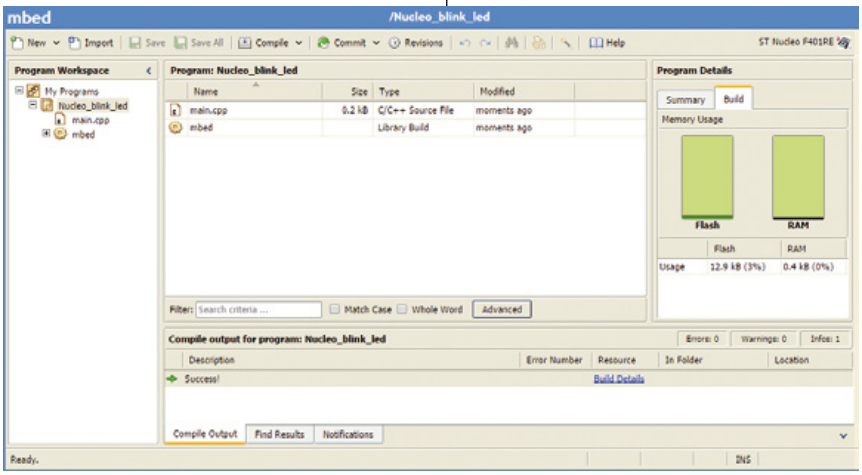
A ce stade, vous n'avez rien dans votre espace de travail et vous pouvez, au choix, importer un





Il est toujours de bon ton de mettre à jour les bibliothèques en important un programme d'un autre utilisateur afin de les faire correspondre avec le modèle de carte que vous possédez et également profiter des derniers correctifs des bibliothèques

Après compilation d'un code, il est possible de cliquer sur « Build Details » pour avoir un aperçu de l'occupation de la mémoire flash et de la RAM. Une fonctionnalité bien pratique qui mériterait d'être ajoutée dans l'environnement Arduino



projet d'un autre utilisateur ou en créer un nouveau. Pour notre prise en main, nous allons simplement piocher dans ce qui existe. Cliquez alors simplement sur le bouton « *Import* » pour faire apparaître le « *Import Wizard* » listant l'ensemble des projets partagés par d'autres. Parmi les premières entrées de la liste se trouve *Nucleo_blink_led* de la *Team ST* (je ne sais pas pourquoi mais *Team ST* ça m'a fait penser à Pokemon). Il vous suffit de sélectionner l'entrée et cliquer sur le gros bouton « *Import* ». Une fenêtre de confirmation apparaît vous permettant entre autre de changer le nom du projet. Cochez la case « *Update all libraries to the latest revision* » et cliquez simplement sur « *Import* ».

Le projet est automatiquement intégré à votre espace de travail et vous pouvez alors éditer le code en cliquant sur **`main.cpp`**. Comme vous pouvez le constater, ce simple code est facilement intelligible si vous êtes coutumier d'Arduino (après tout ceci reste du C++). Pour programmer votre carte mbed, rien de plus simple, cliquez sur le bouton « *Compile* » et après quelques secondes votre navigateur vous propose de télécharger un fichier. Assurez-vous que l'emplacement de sauvegarde est bien votre carte (ici « NUCLEO (F:) ») et enregistrez le fichier. Immédiatement, la led bicolore proche du connecteur USB doit s'activer signifiant un accès et une fraction de seconde après, la led LD2 va clignoter à intervalles réguliers comme l'ordonne le programme. Vous venez de « flasher » votre premier programme dans une carte mbed !

C'est aussi simple que cela et justement là toute la force de mbed. Vous pouvez vous concentrer pleinement sur vos projets sans avoir à vous soucier de l'installation d'outils et des éventuels problèmes de compatibilité avec le système que vous utilisez. Mieux encore, vous pouvez encore accélérer les choses en utilisant les touches de raccourcis comme [ctrl]+D pour compiler et télécharger ou encore [ctrl]+B pour simplement compiler sans téléchargement.

3. A PROPOS DU CODE

En dehors de l'aspect fonctionnel de l'interface, le code à fournir dans l'environnement mbed est sensiblement différent de celui utilisé par Arduino. Rappelons toutefois qu'un croquis Arduino n'est rien d'autre qu'un code C/C++ et que la syntaxe générale est identique, même si elle se trouve légèrement simplifiée via quelques astuces.

Alors qu'un croquis Arduino prend la forme suivante :

```
void setup() {
  // code lancé une fois au
  // démarrage de la carte
}

void loop() {
  // code répété en boucle
  // au fil du programme
}
```

le code équivalent d'un projet mbed ressemblera à ceci :

```
#include "mbed.h"

int main() {
  // code lancé une fois
  while(1) {
    // code répété en boucle
  }
}
```

C'est transparent pour l'utilisateur Arduino mais ce qui est effectivement envoyé à une carte Uno par exemple ce sera quelque chose comme :

```
int main() {
  setup();

  for (;;) {
    loop();
  }

}
```

Le passage de l'un à l'autre environnement n'est pas grand chose et tient en une simplification presque enfantine : un programme standard en C/C++ commence toujours par un appel à la fonction **`main()`** que le programmeur doit créer. Le **`main()`** d'un croquis Arduino est dissimulé à l'utilisateur mais existe et appelle **`setup()`** une fois, puis **`loop()`** à répétition. Arduino cache simplement **`main()`** afin que l'utilisateur n'ait pas à se soucier, lors de ses premiers pas, de la structure d'un tel code mais puisse immédiatement et instinctivement faire la distinction entre l'initialisation et la boucle principale. Cela peut être très déroutant pour un débutant, même programmeur sur PC, de se dire que le code dans un microcontrôleur n'est jamais à l'arrêt et une erreur courante de ne pas concevoir un programme pour qu'il tourne en boucle à l'infini. En supprimant **`main()`**, Arduino réduit à néant la probabilité d'un tel oubli.

Avec mbed, les choses sont un peu plus... classiques. C'est au programmeur (vous) que revient la responsabilité de créer la fonction **`main()`** et tout ce qu'elle implique, mais une fois le principe acquis l'affaire est dans le sac pour de bon. Tout le reste est absolument identique, de la déclaration de variables à leur initialisation en passant par les appels de fonctions ou la syntaxe des conditions et des boucles.

Côté bibliothèques les choses sont également très similaires mais à la fois plus denses et plus simples. En effet, tout est réuni au même endroit et ajouter une bibliothèque à l'un de vos projets se résume à sélectionner votre programme et à utiliser le bouton « *Import* ». Dans la fenêtre qui s'affiche, assurez-vous d'être sur l'onglet « *Libraries* » et faites simplement votre petit marché.

4. UN VRAI EXEMPLE POUR ASSEOIR TOUT CELA

Ajoutons un peu de pratique pour installer toutes ces bonnes choses dans notre esprit. Chose intéressante, vous pouvez commencer alors même que vous n'avez pas de carte compatible mbed ou qu'elle est encore entre les mains du gentil facteur de la poste. Lancez votre environnement de développement et utilisez « *New* », « *New Program* » et choisissez comme « *Template* » l'entrée « *Empty Program* » pour un projet vide. Choisissez un nom pour votre projet et cliquez « *OK* ».

Votre nouveau programme apparaît dans l'espace de travail. Sélectionnez-le et choisissez « *New* » puis « *New file* » puis taper **`main.cpp`**. Dans l'éditeur, tapez le code suivant :

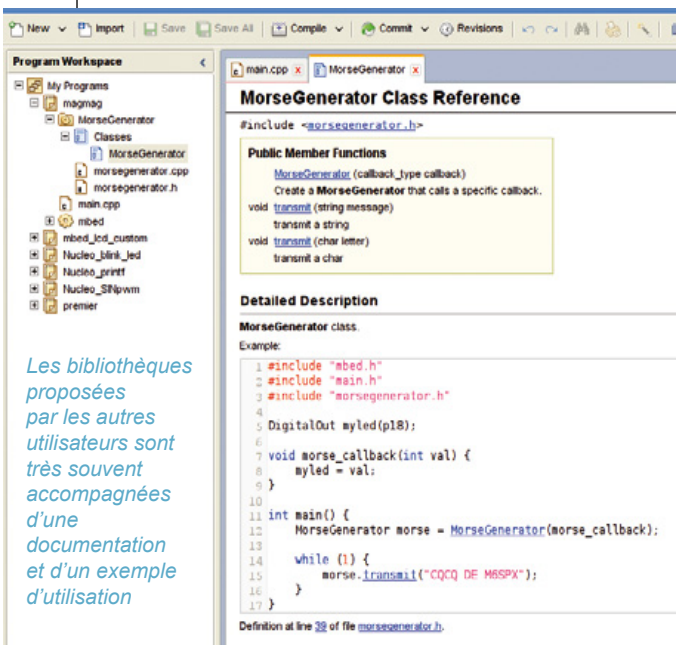
```
#include "mbed.h"

int main() {
    while(1) {
    }
}
```

Puis, tentez une compilation sans téléchargement ([ctrl]+B). Ça ne marche pas ! Mais ceci est parfaitement

normal. Une bibliothèque fournit les fonctions de base de mbed, c'est pourquoi nous avons inclus **mbed.h** en début de code. Cependant, le compilateur ne retrouve pas ce fichier car il n'est pas présent dans votre projet. Comme l'erreur est classique, en bas de la fenêtre, au niveau du message d'erreur, se trouve un petit bouton « *Fix it!* » (« Corriger ça ! »). En cliquant, l'environnement va tenter de déterminer d'où peut provenir un tel fichier et vous propose une liste de bibliothèques qu'il est possible d'ajouter dans votre projet. La bibliothèque qu'il nous faut est la première, « mbed ». Ajoutez-là et automatiquement le code se compile et vous est proposé au téléchargement (ça c'est un bug, puisque nous avions déclenché une simple construction).

Utilisez maintenant le bouton « *Import* » puis l'onglet « *Libraries* » et dans la zone de recherche entrez « morse ». Dans la liste, vous devez trouver « MorseGenerator » de Stephen Paulger. Sélectionnez la ligne et importez (n'oubliez pas de cocher « *Update* », c'est souvent une bonne idée). La bibliothèque est ajoutée à votre projet.

En parcourant l'arborescence, vous pourrez constater que Stephen a prévu une documentation pour sa bibliothèque. Elle décrit comment utiliser les fonctions qu'elle met à disposition et fournit un exemple.



Les bibliothèques proposées par les autres utilisateurs sont très souvent accompagnées d'une documentation et d'un exemple d'utilisation

Pour intégrer ces éléments dans votre **main.cpp**, vous pouvez copier/coller ces morceaux d'exemples (il semblerait que Chrome pour GNU/Linux ne permette pas la sélection du texte, ceci fonctionne pourtant avec Chrome pour Windows), pour arriver à ceci :

```
#include "mbed.h"
#include "morsegenerator.h"

// configuration de LED1
// en sortie "morseLED"
DigitalOut morseLED(LED1);

// fonction à appeler pour
// produire du morse
void morsecallback(int val) {
  // on change l'état de la led
  // en fonction de la valeur
  // de la variable val
  morseLED = val;
}

// fonction principale
int main() {
  // on déclare le générateur
  // en précisant la fonction
  // a utiliser générer le signal
  MorseGenerator morse =
    MorseGenerator(morsecallback);

  // boucle infinie
  while(1) {
    // appel de la méthode transmit
    // de l'objet morse avec le
    // texte à envoyer en morse
    morse.transmit("SOS SOS SOS");

    // on attend 3 seconde
    wait(3);
  }
}
```

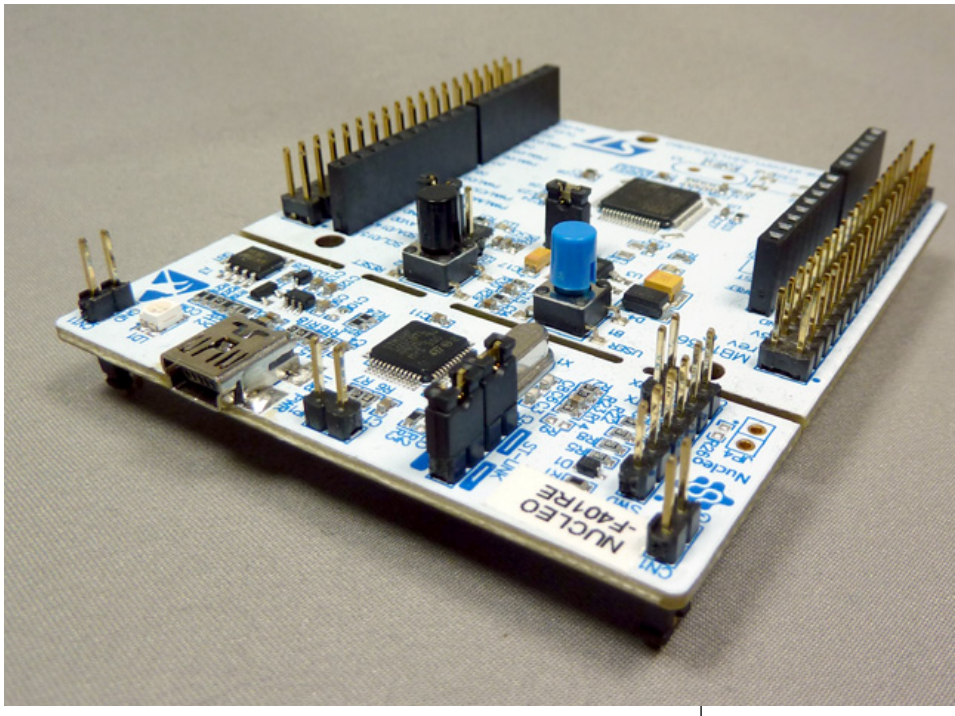
On compile avec [ctrl]+D pour récupérer le firmware binaire qu'on enregistre sur la carte et après quelques secondes on peut la voir émettre un triste SOS (trois court, trois long, trois court) avec sa led verte (non, relier la sortie de la carte à un projecteur en bord de mer n'est absolument PAS une bonne idée).

CONCLUSION

Comme vous pouvez le voir, faire du vrai C/C++ avec mbed n'est pas vraiment plus compliqué qu'utiliser Arduino pour des projets simples. Bien sûr, ceci se complique quelque peu lorsqu'on souhaite faire des

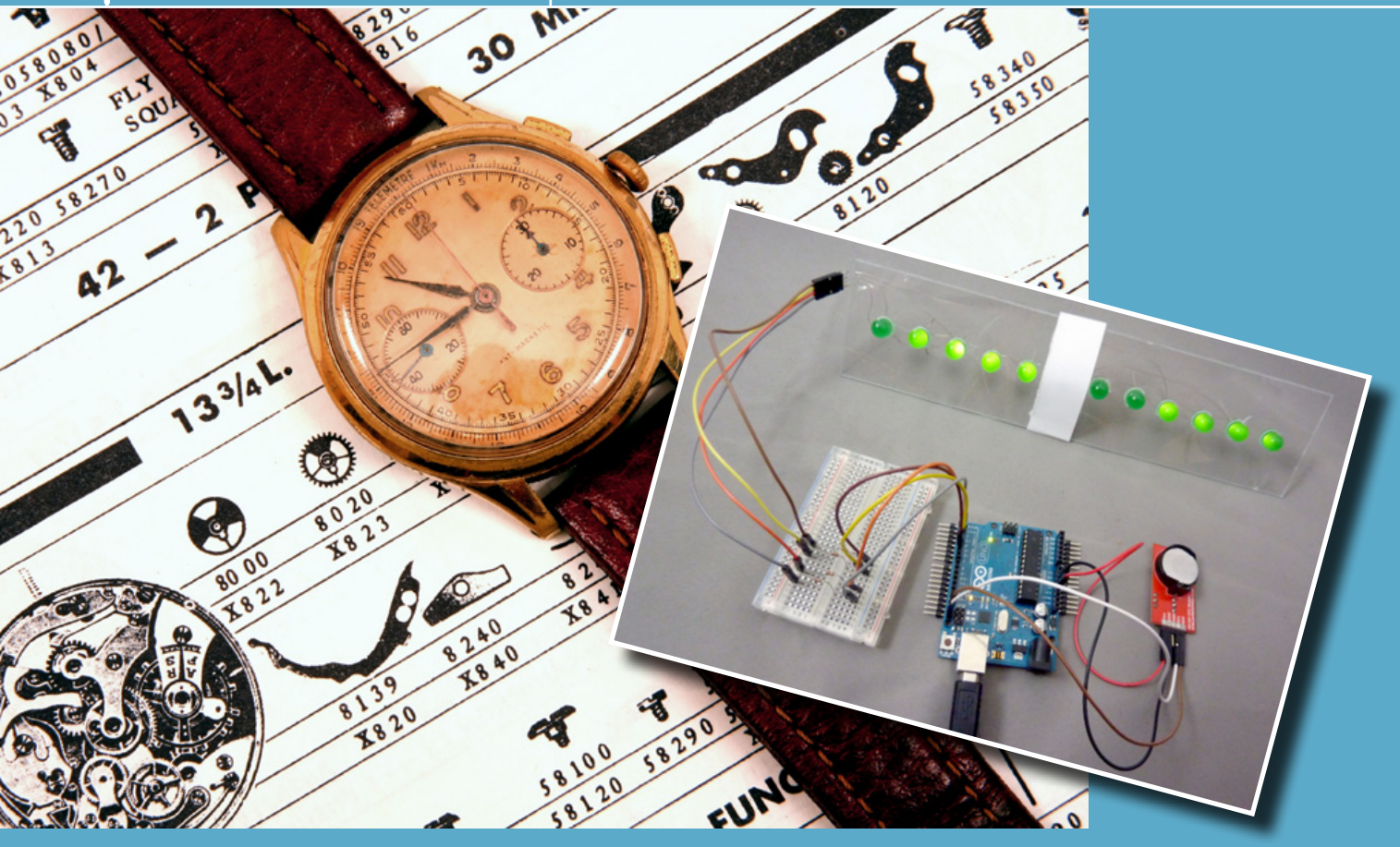
choses un peu plus avancées. Fort heureusement la popularité grandissante de mbed provoque une démultiplication des projets publiés et des bibliothèques disponibles. Certains aspects de la plateforme mbed sont très intéressants, à commencer par le prix des cartes qui proposent des ressources très importantes pour une fraction du coût d'une simple Arduino Uno. Le caractère ubiquitaire des développements est également un point très agréable. Ainsi rien ne vous empêche de débuter un projet au bureau sous Windows, en poursuivre le développement sur le coin d'une table au restaurant avec votre Mac en 3G/4G/LTE/HSPA+/Wifi, pour enfin le finir chez vous, tard dans la nuit, sur votre PC GNU/Linux... Il est même possible d'utiliser le navigateur d'une tablette si vous êtes assez courageux (avec un clavier Bluetooth) ou masochiste (sans clavier Bluetooth sur une tablette 7 pouces).

Du côté des limitations et des points gênants, on retrouve tous les griefs liés aux services « dans le cloud » et en particulier l'absolue nécessité d'avoir une connectivité Internet (les usagés du TGV voient de quoi je parle). L'autre problème tient dans le fait qu'on n'a strictement aucun contrôle de la plateforme elle-même aussi bien en terme de sécurité du fonctionnement qu'au niveau de la pérennité de la solution. Certes, c'est ARM qui se trouve derrière mais cela reste un simple service qui peut disparaître du jour au lendemain ou pire, devenir subitement payant. Il est toujours possible d'exporter ses projets et d'en faire une sauvegarde mais ceci n'efface pas tous les doutes.



Les plans ont été annoncés pour une future version de la plateforme avec, parmi les nouveautés, une possibilité de travail *offline*, en ligne de commandes. De plus la solution devrait énormément s'étoffer en intégrant beaucoup de nouvelles fonctionnalités allant jusqu'à l'intégration optionnel d'un système d'exploitation complet (RTOS). Le résultat final n'est pas attendu avant une bonne année de développement mais d'ici là des versions alpha et beta devraient être diffusées. mbed se place sur un terrain sensiblement différent de celui d'Arduino couvrant un public allant de l'utilisateur avec une certaine expérience jusqu'aux professionnels.

Le fait qu'il s'agisse d'une plateforme sponsorisée par ARM et ses partenaires semble garantir un succès, tout comme l'aspect innovant de l'initiative. Mais seul le temps et la maturation du projet assureront l'étendu effective de ce succès. Quoi qu'il en soit, nous avons là d'ors et déjà une solution relativement sympathique pour qui souhaite sortir du sentier battu d'Arduino (que dis-je, battu... pavé, macadamisé et carrelé !) et découvrir de nouveaux horizons. Arduino occupe massivement le terrain mais mbed reste un outsider mais l'histoire de l'informatique et de l'électronique a déjà apporté la preuve que rien n'était joué d'avance. Mieux vaut garder des options sur tout ce qui bouge... **DB**

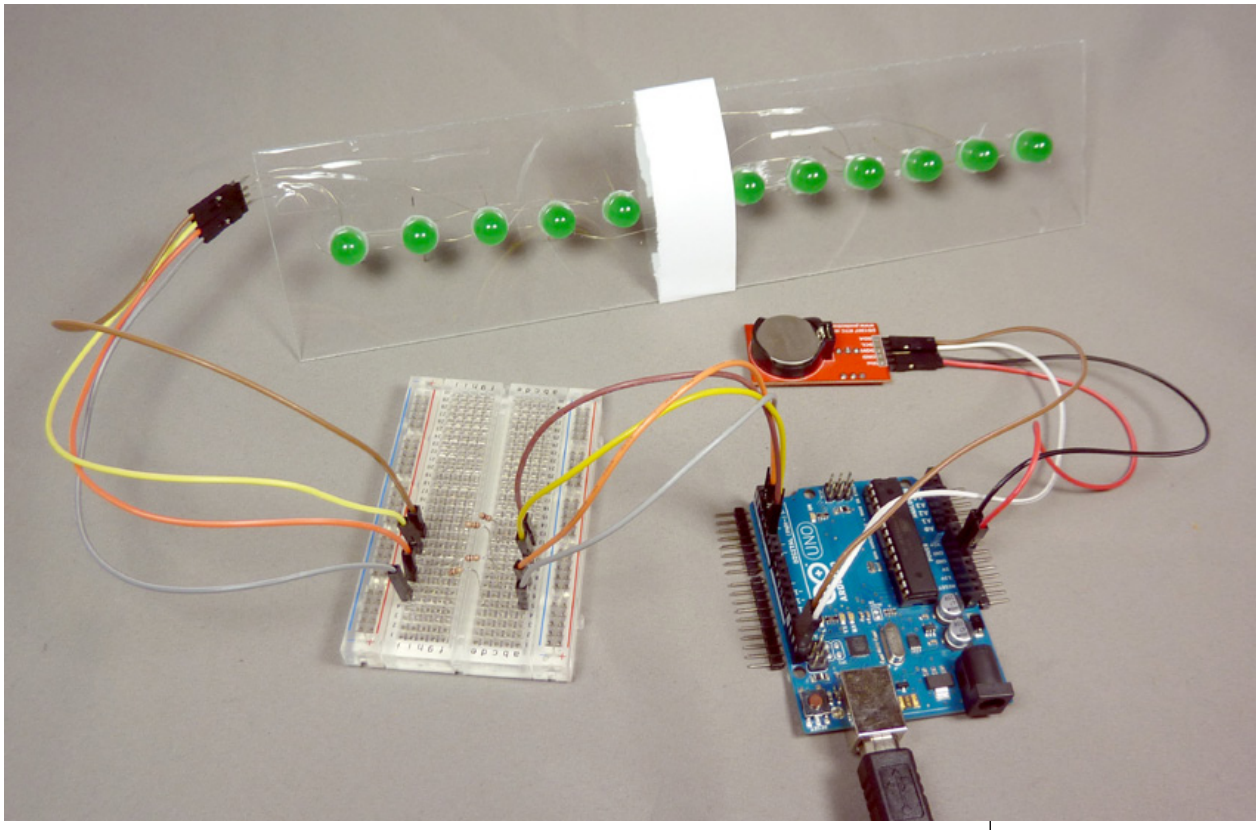


L'HORLOGE BINAIRE : UNE MONTRE PAS COMME LES AUTRES

D. B.

Il y a 10 types de personnes, celles qui comprennent la notation binaire et celles qui ne la comprennent pas. Désolé, mais ceci était une boutade tout simplement incontournable étant donné les explications qui vont suivre. En effet, le sujet qui nous intéresse ici consiste précisément à créer une horloge que seuls les « initiés » sont en mesure de lire et comprendre. Il s'agit de la classique, l'obligatoire et la merveilleuse horloge binaire.

NIVEAU	TEMPS	BUDGET
	25 minutes	environ 30 € (Arduino inclus)



Le montage complet avec les 12 leds montées sur un support, la platine à essais avec les résistances, la carte Arduino Uno et le module DS1307 avec sa pile bouton.

Plantons le décor, voulez-vous ? Dans le monde de tout un chacun, il existe principalement deux types de systèmes de représentation de l'heure. Ces représentations prennent la forme de montres à aiguilles ou de montres digitales. Ces dernières indiquent l'heure avec des chiffres, généralement sous la forme « hh:mm:ss ». Mais nous autres programmeurs savons que les valeurs affichées en base 10 ne sont qu'une façon de présenter les choses. Ainsi, ce qui paraît naturel pour un humain, sous la forme de chiffres de 0 à 9, ne l'est principalement que pour deux raisons : parce que depuis qu'on est tout petit on nous explique que c'est ainsi qu'il faut compter et, apparemment, parce qu'on a 10 doigts (les poulpes comptent en octal). Les ordinateurs eux, n'ayant pas de doigts, ne comptent pas ainsi. Ils ne connaissent que deux choses, les 0 et les 1, NON ou OUI, *on* ou *off*, VRAI ou FAUX... Ce qui ne les empêche pas de faire des merveilles.

En tant que programmeurs nous sommes obligés de parler la langue des machines et donc de faire intimement connaissance avec le système binaire et l'algèbre de Boole, car ces concepts sont essentiels aussi bien en informatique qu'en électronique... numérique. Quoi de plus pratique alors pour entraîner notre cerveau que de manipuler une représentation binaire à chaque instant avec quelque chose d'aussi trivial que de s'enquérir de l'heure qu'il est ? Sachant qu'en plus cela aura la fâcheuse tendance de jeter dans la confusion la plus profonde tous ceux qui ne s'adonnent pas aux mêmes exercices mentaux et qui ne voient alors que des lumières là où nous voyons l'heure du goûter approcher (ceci dit, comme l'avait précisé le grand Bob Widlar, « *every idiot can count to one* »).

LE PRINCIPE

Il existe bien des manières de créer une montre avec une carte Arduino et nous allons choisir ici celle que nous pensons être la plus pédagogique et celle contenant un maximum d'éléments réutilisables pour d'autres projets. Une horloge se compose généralement de deux parties plus ou moins dissociées. La première est une base de temps ou en d'autres termes, un système capable de mesurer le temps qui s'écoule. Une carte comme l'Arduino dispose de plusieurs périphériques internes permettant de mesurer une durée, tous cadencés sur la base d'une oscillation qui rythme également son fonctionnement. Cette cadence est donnée par un composant intégré à la carte : un quartz.

Ceci peut paraître relativement simple et il peut être tentant de vouloir utiliser ce « métronome » pour obtenir une source fiable de la mesure du temps qui passe. En réalité, cette source n'est pas si fiable que cela et même des fonctions dédiées comme `millis()` ne disposent pas de la précision qu'on est en droit d'attendre d'une horloge qui fonctionnerait des jours entiers. Énormément de paramètres entrent en ligne de compte lorsqu'on tente d'utiliser une horloge système comme base de temps. C'est entre autres pour cette raison que des microcontrôleurs un peu plus fournis en ressources disposent d'un périphérique dédié utilisant une source d'horloge différente à une fréquence plus adaptée (32,768 kHz). Ce n'est pas une question de puissance, votre PC fait de même en intégrant une horloge dite « temps réel » ou RTC pour *Real-Time Clock*.

Ce fait d'utiliser un périphérique dédié évite bien des problèmes et il n'est pas nécessaire alors de concevoir les programmes de manière à ne pas perturber la mesure du temps. Ceci présente un autre très gros avantage. En effet, comme le périphérique dispose de sa propre horloge, il devient relativement indépendant du système et peut donc fonctionner de manière totalement autonome. Ajoutez une pile ou une batterie et voici un système en mesure d'égrainer les secondes même lorsque le système principal est à l'arrêt.

L'autre élément d'une horloge ou d'une montre est l'interface utilisateur. C'est la partie qui présente simplement l'information. Nous venons de le voir ensemble, il est de bon ton de détacher la mesure de l'affichage. Grâce à l'utilisation d'une RTC, nous avons tout loisir d'utiliser pleinement les ressources du système pour donner l'information sans avoir à nous soucier du temps que prendront ces différentes opérations. Mais ce n'est pas pour autant que nous devons gâcher de précieuses ressources. Beaucoup d'éléments sont rares lorsqu'on développe sur carte Arduino, mais l'un d'entre eux fait souvent cruellement défaut : le nombre d'entrées/sorties.

Entrons dans le vif du sujet en dévoilant une partie de nos plans : nous allons créer une horloge présentant les heures et les minutes sous forme binaire et plus exactement 5 bits pour la première valeur et 6 bits pour la seconde. Chaque bit étant représenté par une led allumée ou éteinte, nous aurons besoin de piloter un minimum de 11 leds. L'Arduino Uno proposant quelques 14 broches en sortie plus 6 en entrée analogique, ceci nous donne un total de 20 broches utilisables en tout et pour tout. Nous sommes en dessous de ce maximum avec nos 11 leds. Mais que se passerait-il si nous souhaitons afficher également les secondes, le jour et le mois ou encore si nous désirons tout simplement nous servir des broches pour un autre usage, complémentaire à la fonction d'horloge ?

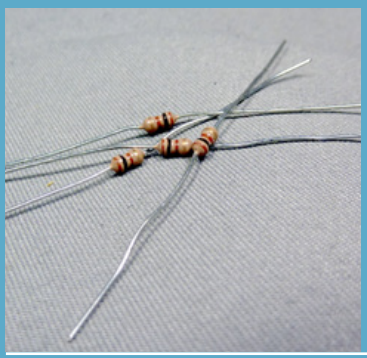
La problématique qui se pose à nous se résume donc en une simple question : comment piloter un maximum de leds avec un minimum de broches ? La réponse ultime qui ne repose pas sur l'utilisation d'un composant dédié porte un nom : le charlieplexing. Cette technique particulière a été pour la première fois exposée par Charlie Allen sous la forme d'une *application note* de son employeur *Maxim Integrated* il y a environ 20 ans. Comme son nom l'indique, il s'agit d'une solution de multiplexage permettant donc de partager une ressource (un nombre limité de broches) entre plusieurs « utilisateurs » (les leds).

La façon de traiter ici ces deux éléments qui composent notre horloge a été choisie de manière à ce que vous puissiez les réutiliser pour vos propres projets. Une RTC peut en effet être d'une grande utilité pour une vaste majorité de montages et la technique du charlieplexing vous permettra de piloter non seulement de simples leds, mais également un grand nombre de systèmes d'affichage comme les afficheurs 7 segments par exemple. Ceci de façon bien plus économique que via des circuits intégrés dédiés comme le décodeur BCD 4511 ou encore le contrôleur de matrice de leds MAX7219.

CE QU'IL VOUS FAUT

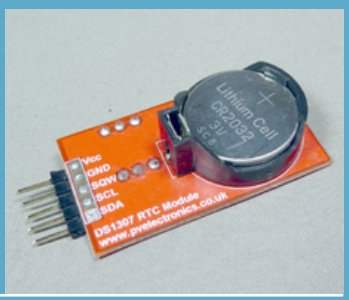
- Une carte Arduino disposant d'une interface i2c/TWI. Les générations actuelles de cartes comme la Uno Rev3 disposent de broches dédiées marquées SCL et SDA. Les cartes plus anciennes comme la Duemilanove ne proposent pas ces sorties, mais le bus est tout de même accessible via les broches A4 (SDA) et A5 (SCL). Vous l'avez compris, lorsqu'elles sont utilisées en mode i2c/TWI, ces broches ne peuvent servir d'entrées analogiques.

- 12 leds du type et de la couleur qui vous convient le mieux. Il est de bon ton d'utiliser des leds proposant une forte luminosité, car en raison du multiplexage seules certaines d'entre elles sont actives à un moment donné. C'est la persistance rétinienne qui donne l'impression que 11 leds sont actives en même temps, mais il en découle une perte significative d'intensité lumineuse. Nous construirons notre montage avec 12 leds, mais seules 11 seront utilisées pour l'affichage de l'heure et des minutes. Nous partons ici du principe que vous serez amené à utiliser vos leds pour d'autres réalisations et qu'il serait dommage de retirer le caractère générique du montage en charlieplexing.



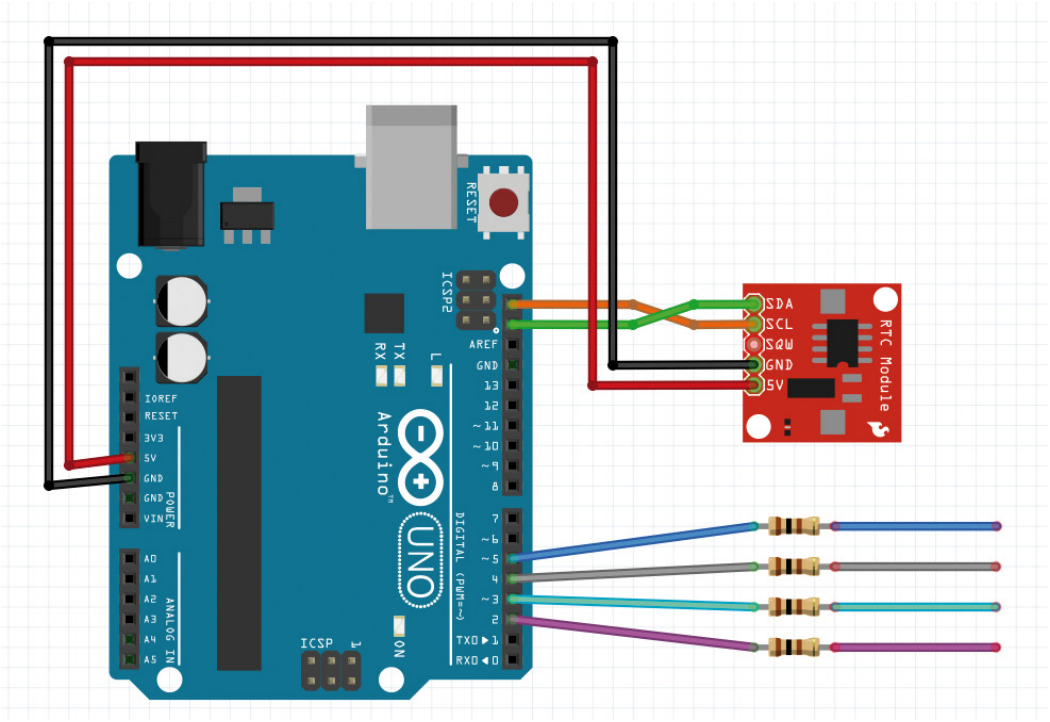
- Quatre résistances de 100 ohms. Nous utilisons pour notre exemple 12 leds vertes de 10 mm normalement pilotées avec un courant de l'ordre de 20 mA. Un rapide calcul vous indique que soit ces leds présentent une tension bien étrange à leurs bornes, soit nos résistances sont quelque peu en dessous de leur valeur normale. C'est la seconde option qui est la bonne, car il faut savoir qu'une led possède deux valeurs maximum concernant le courant qui la traverse. Une valeur pour un courant continu (*DC Forward Current*) et une valeur pour une alimentation ponctuelle (*Peak Forward Current*) qui peut être plus de 5 fois supérieure en fonction du rapport cyclique et de la durée d'une impulsion. Tout ceci est, bien entendu, décrit dans la documentation du fabricant.

- Un module RTC utilisant un DS1307. Nous avons déjà abordé le sujet dans un numéro précédent (Hackable #1) avec un article sur l'utilisation d'un tel module avec la Raspberry Pi (DS1338). Les DS1307 et DS1338 se différencient principalement par leurs tensions d'alimentation, mais sont très similaires du point de vue de leur utilisation. Le DS1307 est bien plus facile à trouver aussi bien dans des boutiques en ligne que sur des sites d'enchères, mais il ne peut fonctionner qu'avec une tension entre 4,5 et 5,5 volts, ce qui interdit son utilisation avec la Pi qui est en 3,3V.



- Du fil électrique et/ou une platine à essais avec une flopée de câbles. Connecter 12 leds en utilisant 4 broches de l'Arduino implique paradoxalement une belle salade de câbles sur une platine à essais. En effet, comme nous le verrons un peu plus loin, l'assemblage est relativement logique et linéaire si l'on dispose de quatre longues lignes de connexion. Malheureusement, ce n'est généralement pas le cas sur les platines et il faut alors composer pour arriver au même résultat. Il est bien plus simple d'attaquer ici directement au fer à souder après avoir positionné les 12 leds sur un support en inversant alternativement la polarité.

LE MONTAGE



L'assemblage de la réalisation en lui-même ne pose pas de réelles difficultés puisque cela se résume à attacher le module en i2c et à utiliser quatre sorties de l'Arduino avec de simples résistances. C'est malheureusement l'une des réalisations où, étrangement, sur le papier les choses sont très simples, mais où dans la pratique l'interconnexion prend rapidement des tournures de labyrinthe. Réjouissez-vous, généralement c'est l'inverse, on bute sur l'aspect théorique, mais ensuite la pratique passe comme une lettre par la poste...

LE DS1307

Nous ne nous étendrons pas outre mesure sur ce composant/module relativement simple d'usage. Notre Arduino Uno Rev3 dispose de deux broches libellées SCL et SDA correspondant au bus i2c ou TWI.

i2c est une désignation utilisée par son créateur NXP/Philips alors que TWI est l'acronyme de *Two Wire Interface* (« interface à deux fils », oui c'est super recherché comme nom), mais les deux termes qualifient techniquement le même bus. Celui-ci, en plus d'une alimentation et d'une masse est effectivement un bus à deux fils. Nous avons SCL (*Serial Clock Line*) qui véhicule le signal d'horloge et SDA (*Serial DAta line*) pour la communication des données. Il n'y a pas de rôle fixe maître/esclave en i2c tel que c'est le cas en SPI (voir l'article sur l'écran LCD dans le présent numéro). Chaque équipement sur le bus peut être maître ou esclave et il peut y avoir plusieurs maîtres, mais la communication a toujours lieu entre un maître et plusieurs esclaves, et toujours à l'initiative du maître.

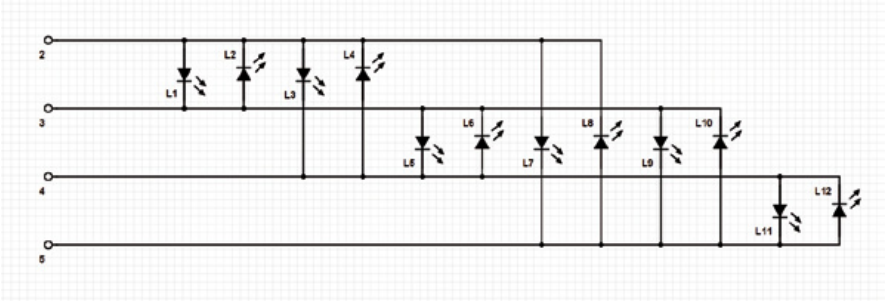
Un élément pratique à retenir sur le bus i2c/TWI est le fait que les deux lignes/signaux sont bidirectionnels. Il n'y a pas de MISO/MOSI comme en SPI et ceci implique un fonctionnement particulier au niveau électrique. En effet, sur un bus i2c les périphériques sont connectés par des lignes dites à collecteur ouvert. Les équipements n'imposent donc pas un niveau de tension lorsqu'ils n'utilisent pas le bus. C'est pour cette raison que la présence de résistances de rappel à VDD (la tension correspondant au niveau logique haut) est nécessaire, puisque aucun composant n'impose un niveau de tension. La totalité des modules RTC DS1307 et DS1338 que nous avons eus entre les mains était équipée de ces résistances.

La connexion d'un module utilisant un DS1307 ou un DS1338 se fait en reliant simplement les lignes entre elles, masse/masse, Vcc/+5V, SDA/SDA et SCL/SCL.

Notez qu'il est très peu probable que vous receviez vos modules équipés de piles et c'est très bien ainsi. En effet, les piles boutons comme celle du type CR2032 ont une durée de vie limitée ainsi qu'une date de péremption. Il est peu recommandé d'acheter ce type de piles sur des sites ne possédant pas une certaine renommée. Mieux vaudra, le plus souvent, vous tourner vers une boutique de hifi du coin ou simplement vers le rayon bricolage du supermarché. Vous aurez ainsi tout loisir de choisir une pile de marque et de consulter sa date limite d'utilisation qui doit figurer sur l'emballage. Cette pile permet de faire fonctionner le DS1307 même en l'absence d'alimentation par le montage. En d'autres termes, vous programmerez l'heure une fois et le module poursuivra le compte du temps qui passe comme un grand. Bien entendu, en l'absence de pile, la déconnexion de l'alimentation rime avec remise à zéro...

LE CHARLIEPLEXING

Voici la partie la plus amusante du projet. La technique du charlieplexing ne fonctionne qu'avec des composants polarisés et le principe de fonctionnement est relativement simple. Voici notre schéma de connexion :



À gauche, nous avons les broches ou « sorties » de l'Arduino. Vous allez rapidement comprendre pourquoi le terme « sortie » n'est pas tout à fait adapté. Considérons en premier lieu les broches 2 et 3 ainsi que les leds L1 et L2 :

- si 2 est en sortie à l'état haut et 3 en entrée, L1 s'allume,
- si 2 est en entrée et 3 en sortie à l'état haut, L2 s'allume.

Ajoutons mentalement maintenant la broche 4 dans notre petit jeu. Les deux premières affirmations restent exactes, mais s'ajoutent maintenant :

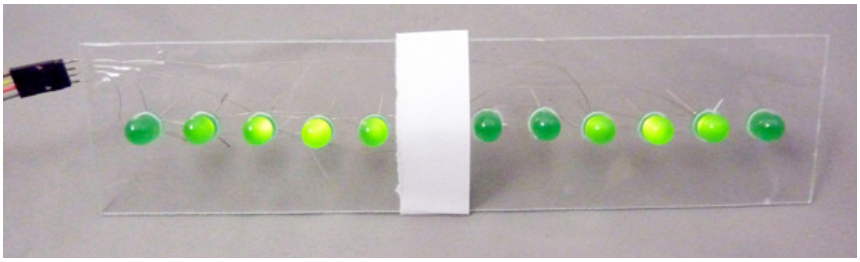
- si 2 est à haut et 4 en entrée, L3 s'allume,
- si 2 est en entrée et 4 à haut, L4 s'allume,
- si 3 est à haut et 4 en entrée, L5 s'allume,
- et enfin si 4 est à l'état haut et 3 en entrée, c'est L6 qui s'allume.

En résumé, en jouant sur la polarité des leds et l'état de nos entrées/sorties, en ajoutant une seule broche nous ajoutons le contrôle sur quatre nouvelles leds. Ainsi, en ajoutant encore une broche, nous voici en mesure de piloter 12 leds avec seulement 4 entrées/sorties. Techniquement, avec *n* broches, nous pouvons piloter n^2-n leds (ici, n=4 et donc 4*4-4=12). C'est magique !

Magique ? Non et miraculeux encore moins. Comme dans la vie, en électronique on a rien sans rien ou, autrement dit, ce qu'on gagne d'un côté on le perd par ailleurs. Ici, nous gagnons le fait de grandement limiter le nombre d'entrées/sorties de l'Arduino, mais en contrepartie il n'est pas possible, loin de là, d'allumer toutes les leds en même temps. Dans l'absolu, en ayant un port en entrée et trois en sortie par exemple, seule trois d'entre elles peuvent se trouver alimentées au maximum. Et ce n'est qu'un cas de figure particulier.

La solution consiste alors, comme pour d'autres solutions de multiplexage de leds, à alterner l'alimentation de chacune d'elles de manière à créer l'illusion qu'elles soient toutes actives. La persistance rétinienne faisant le reste, à un taux de rafraîchissement élevé, on a l'impression que toutes les leds sont allumées mais se produit alors un phénomène similaire à la PWM. En effet, le ratio de temps où une led est sous tension par rapport au temps ou elle se trouve éteinte n'est pas très important et on perd énormément en intensité lumineuse.

Un autre problème découlant du multiplexage concerne le courant qui circule dans le circuit. Nous pouvons augmenter la luminosité de l'affichage en faisant passer plus de courant tout en respectant les indications du constructeur mais, comme nous n'avons qu'un nombre limité de résistances, ceci implique que le courant est le même pour toutes les leds. Ceci pose un problème majeur s'il s'agit d'utiliser des



Cette photo présente deux choses capitales : il est 15h14 et je ne suis absolument pas capable de percer 12 trous de 10 mm en ligne droite sur un support en polycarbonate.

composants de modèles différents. Il n'est pas possible, sauf à utiliser des leds avec exactement les mêmes caractéristiques, d'utiliser ce type de technique avec un affichage multicolore.

Ajoutons à cela la complexité croissante du circuit à mesure qu'on ajoute des entrées/sorties et donc des leds à contrôler. L'ensemble devient proportionnellement de plus en plus complexe, difficile à construire et surtout problématique lorsqu'un dysfonctionnement survient. Qu'il s'agisse d'un problème de connexion ou du mauvais fonctionnement d'une led ou d'une résistance, identifier la cause d'un comportement étrange prend souvent plus de temps que de reconstruire le circuit dans son ensemble.

Quoi qu'il en soit, ces concessions sont parfois parfaitement justifiées, car le charlieplexing permet surtout de limiter le nombre de composants et en particulier de circuits logiques. Il existe bien des manières de piloter un grand nombre de leds via des registres à décalages, des latchs, des décodeurs BCD ou encore des circuits spécialisés, mais aucune n'est aussi économique matériellement qu'une poignée de résistances.

LE CROQUIS

Fichier Édition Croquis Outils Aide

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

#include <Chaplex.h>

// définition des broches pour les leds
byte ctrlpins[] = {2,3,4,5};

// déclaration du charlieplexing
Chaplex myCharlie(ctrlpins, 4);

// représentation des connexions par paires
// sous la forme {anode,cathode}
// les nombres désignent les broches utilisées par leur
// position dans ctrlpins[] et non les numéros des broches
// de l'Arduino
// Exemple : 0 = broche en position 0 = 2
//           2 = broche en position 2 = 4
// On déclare toutes les connexions telles qu'elles existent
// physiquement. L'ordre dans lequel on les déclare est
// l'ordre des leds
```

```
charlieLed myLeds[12]={{0,1},
                       {1,0},
                       {0,2},
                       {2,0},
                       {1,2},
                       {2,1},
                       {0,3},
                       {3,0},
                       {1,3},
                       {3,1},
                       {2,3},
                       {3,2}};

// Ceci est la valeur de départ du compteur/timer
// 256-16MHz/1024/78 = environ 5 ms
byte timer2TCNT2 = 178;

void setup() {
  // on passe toutes les leds à OFF
  for (int i=0; i< 12; i++) {
    myCharlie.ledWrite(myLeds[i], OFF);
  }

  // configuration du timer2
  noInterrupts();
  TCCR2A = 0;
  TCCR2B = 0;
  TCNT2 = timer2TCNT2;
  TCCR2B |= (1 << CS22) | (1 << CS21) | (1 << CS20);
  TIMSK2 |= (1 << TOIE2);
  interrupts();
}

// routine appelée à chaque fois que le
// compteur/timer déborde.
ISR(TIMER2_OVF_vect) {
  myCharlie.outRow();
  TCNT2 = timer2TCNT2;
}

void loop() {
  // variable pour stocker l'heure
  tmElements_t tm;
  // lecture du DS1307
  if (RTC.read(tm)) {
    // on remplit le tableau en fonction de la valeur
    // de l'heure et des minutes
    myCharlie.ledWrite(myLeds[0], (tm.Hour & 16));
    myCharlie.ledWrite(myLeds[1], (tm.Hour & 8));
    myCharlie.ledWrite(myLeds[2], (tm.Hour & 4));
    myCharlie.ledWrite(myLeds[3], (tm.Hour & 2));
    myCharlie.ledWrite(myLeds[4], (tm.Hour & 1));

    // on saute la led L6 toujours à OFF
    myCharlie.ledWrite(myLeds[5], OFF);

    myCharlie.ledWrite(myLeds[6], (tm.Minute & 32));
    myCharlie.ledWrite(myLeds[7], (tm.Minute & 16));
    myCharlie.ledWrite(myLeds[8], (tm.Minute & 8));
    myCharlie.ledWrite(myLeds[9], (tm.Minute & 4));
    myCharlie.ledWrite(myLeds[10], (tm.Minute & 2));
    myCharlie.ledWrite(myLeds[11], (tm.Minute & 1));
  }
  // titi pause
  delay(500);
}
```

Arduino

À PROPOS DU CROQUIS

INSTALLATION DES BIBLIOTHÈQUES

Trois bibliothèques doivent être installées. Les deux premières permettent la gestion structurée du temps et l'accès au DS1307 (ou DS1338). Ces bibliothèques écrites et maintenues par *Michael Margolis* sont tantôt faussement considérées comme celles de *Paul Stoffregen* (le créateur des cartes/modules Teensy) qui en fait grandement la promotion.

Deux options s'offrent ainsi à vous pour l'installation. Soit vous passez par le site pjrc.com en prenant bien garde à télécharger la dernière version de la bibliothèque `Time` (1.4 de septembre dernier), car la 1.3 pose problème avec la dernière version de l'IDE Arduino. Soit vous rendez à César ce qui appartient à César (par Toutatis !) et pointez votre navigateur sur **https://github.com/michaelmargolis/arduino_time** pour récupérer le fichier Zip contenant les bibliothèques `Time`, `DS1307RTC` et `TimeAlarms`. Après désarchivage et copie des éléments dans votre répertoire `libraries`, vous devez obtenir trois sous-répertoires : `Time`, `DS1307RTC` et `TimeAlarms`. Au lancement de l'environnement de développement Arduino, vous devez normalement retrouver, dans les menus, des exemples pour chacun des trois éléments.

Le chalieplexing, quant à lui, sera l'affaire d'une autre bibliothèque : `Chaplex`. Il s'agit d'une évolution par Stefan Götze de la bibliothèque `Charlieplex` d'Alexander Brevig. Le code est un peu ancien, mais fonctionne toujours à merveille. Vous pourrez récupérer la bibliothèque sur **http://code.google.com/p/yacll/** et procéder au désarchivage du fichier `Chaplex.zip` dans votre sous-répertoire `libraries`. Là encore, après redémarrage de l'environnement de développement, vous devez trouver les exemples de croquis associés.

PILOTAGE DU DS1307

Si vous venez de fraîchement recevoir votre module RTC DS1307 ou d'y placer la pile bouton, il est plus que probable que l'horloge ne soit pas configurée. Une fois l'ensemble connecté à votre Arduino, le plus simple est d'ouvrir dans l'IDE l'exemple `SetTime` livré avec la bibliothèque `DS1307RTC`. Ce croquis permet de très rapidement configurer date et heure sur le DS1307 en fonction de celles de votre machine de développement. Ouvrez l'exemple, compilez et chargez le croquis dans votre carte Arduino.

Ceci fait, cliquez immédiatement sur l'icône du moniteur série pour voir normalement s'afficher un message du type :

```
DS1307 configured Time=10:57:54, Date=Oct 21 2014
```

Ceci confirme la configuration des informations dans la RTC. Dès lors, celle-ci est à l'heure et tant que la pile n'est ni retirée ni vide vous n'aurez plus besoin d'y écrire. Vous pourrez également utiliser l'autre exemple livré avec la bibliothèque, `ReadTest`, pour vous assurer que les lectures ne posent aucun problème. Là encore, le moniteur série vous donnera les informations que vous espérez :

```
DS1307RTC Read Test
-------------------
Ok, Time = 11:01:28, Date (D/M/Y) = 21/10/2014
Ok, Time = 11:01:29, Date (D/M/Y) = 21/10/2014
Ok, Time = 11:01:30, Date (D/M/Y) = 21/10/2014
```

L'utilisation de la bibliothèque et la lecture des données qui nous intéressent est relativement simple. Il nous suffit d'inclure les bibliothèques :

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

Puis de demander une lecture des informations :

```
tmElements_t tm;
if (RTC.read(tm)) {
  // des trucs si ça marche
}
```

La fonction `RTC.read()` prend en argument une variable de type `tmElements_t` qui est une structure contenant énormément

d'informations. On retrouve ainsi des valeurs numériques non signées sur 8 bits directement accessibles :

- `tm.Second` : le nombre de secondes,
- `tm.Minute` : les minutes,
- `tm.Hour` : l'heure au format 24h,
- `tm.Wday` : le jour de la semaine avec dimanche = 1,
- `tm.Day` : le jour dans le mois,
- `tm.Month` : le numéro du mois,
- `tm.Year` : le nombre d'années depuis 1970.

Notez que notre croquis ne prend pas en compte le fait qu'il puisse y avoir une erreur. Si `RTC.read()` retourne 0 nous bouclons simplement dans `loop()`. Jetez un œil à l'exemple `ReadTest` pour plus d'informations.

LE CHARLIEPLEXING AVEC CHAPLEX

La bibliothèque `Chaplex` est un peu plus complexe à utiliser, car il faut en comprendre la logique interne, mais aussi la façon optimale de l'utiliser. Après notre description des limitations du charlieplexing, vous devez avoir compris que la vitesse à laquelle nous rafraîchissons l'affichage est un point critique. Mais avant de parler des performances, voyons comment configurer la base du multiplexage.

Avant toute chose, déclarons les éléments nécessaires, à commencer par l'inclusion de la bibliothèque :

```
#include <Chaplex.h>

byte ctrlpins[] = {2,3,4,5};
Chaplex myCharlie(ctrlpins, 4);
```

Nous stockons dans le tableau `ctrlpins[]` la liste des broches utilisées. Attention, l'ordre prend toute son importance ici, car ces broches seront ensuite désignées, non par leur numéro Arduino, mais par leur position dans le tableau (en commençant avec la position 0). Nous déclarons ensuite notre objet `myCharlie` en utilisant un constructeur (C++ oblige) et en lui passant le tableau ainsi que le nombre de broches utilisées.

Il nous faut ensuite lister toutes les connexions :

```
charlieLed myLeds[12]=
{{0,1},{1,0},{0,2},{2,0},{1,2},{2,1},
{0,3},{3,0},{1,3},{3,1},{2,3},{3,2}};
```

Il s'agit d'une succession de désignations des broches par rapport au tableau `ctrlpins[]` avec, à chaque fois, l'anode et la cathode. `{0,1}` désigne ainsi la première led, avec son anode (+) connectée à la broche 0 du tableau (donc la 2) et sa cathode (-) connectée à la broche 1 du tableau (donc la 3). Et ainsi de suite en suivant le schéma donné en début d'article... Petite astuce mnémotechnique au passage, pour vous souvenir de la signification anode/cathode : ÂNE-node et CAThode, un âne est plus grand qu'un *cat* (chat), l'anode est donc le « plus » (normalement c'est cathode=coq, mais je préfère les chats que les trucs pleins de plumes qui hurlent comme des malades aux premières lueurs du jour, surtout en été).

Ceci étant configuré, il devient possible d'utiliser la méthode suivante pour définir l'état de chaque led :

```
myCharlie.ledWrite(myLeds[i], OFF);
```

où `i` est la position de la led dans le tableau `myLeds[]`. Notez que ceci ne change pas l'état effectif des leds connectées. Le système fonctionne comme une mémoire tampon. On décide de l'état `ON` ou `OFF` de chaque led du tableau et seulement ensuite on appelle `myCharlie.outRow()` pour répercuter les changements.

TIMER, INTERRUPTION ET ISR

La place nous manque pour traiter en détail l'utilisation des interruptions, nous ne fournirons donc ici que les bases. Une interruption est « quelque chose qui interrompt le fonctionnement d'un programme ». Il existe énormément de sources d'interruptions dans le microcontrôleur équipant une carte Arduino. Certaines

sont matérielles, d'autres logicielles. Selon la configuration que nous choisissons d'utiliser, il est possible de provoquer l'exécution de quelques lignes de code lorsqu'une interruption survient : c'est une routine d'interruption ou ISR (*Interrupt Service Routine*). Notre croquis comporte une telle ISR, déclarée comme ceci :

```
ISR(TIMER2_OVF_vect) {
  myCharlie.outRow();
  TCNT2 = timer2TCNT2;
}
```

La routine utilise **myCharlie.outRow()** qui « pousse » les données sur l'état des leds sur l'afficheur multiplexé. On ne choisit pas les noms des ISR, car celles-ci sont fixes est correspondent en réalité à un numéro ou vecteur d'interruption. L'ISR que nous avons écrite est celle correspondant au vecteur de l'interruption déclenchée lorsque le compteur du *timer* 2 déborde.

Un *timer* est comme une petite horloge interne qui compte en permanence. Vous pouvez configurer le ou les *timers* d'un Atmel AVR pour compter de plusieurs façons. Dans la fonction **setup()** nous configurons le *timer* 2 ainsi :

```
// désactivation des interruptions
noInterrupts();

// mise à zéro de la configuration
// du timer2 via les registres A et B
TCCR2A = 0;
TCCR2B = 0;

// le compteur est initialisé avec une
// valeur fixe, ici 178. Le compteur va
// jusqu'à 255, valeur à laquelle il déborde.
// En débutant à 178 ceci nous donne environ
// une interruption toutes les 5 ms
TCNT2 = timer2TCNT2;

// On règle le fonctionnement du timer non
// pas sur la cadence de l'horloge mais sur une
// fraction de cette fréquence (/1024)
TCCR2B |= (1<<CS22)|(1<<CS21)|(1<<CS20);

// On configure le fait de déclencher une
// interruption lorsque le compteur déborde.
// Lorsque le compteur est à 255 et qu'on ajoute
// 1, celui-ci revient à 0 et notre ISR est
// appelée
TIMSK2 |= (1 << TOIE2);

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

C'est un peu « costaud », mais nous reviendrons sur le sujet plus en détail dans un prochain numéro. Deux choses sont à savoir sur ce type de gestion d'interruption et sur les *timers*. Premièrement, une interruption comme son nom l'indique va interrompre le fonctionnement d'un programme pour exécuter l'ISR. Ainsi, même en plein dans un **delay()** la routine sera appelée. L'autre élément important à ne surtout jamais oublier est le fait que les *timers* peuvent être utilisés par des bibliothèques et qu'il y a donc un risque de conflit entre ce type d'utilisation et les configurations présentes dans ces bibliothèques. Dans le cas qui nous occupe ceci n'est pas un problème puisque notre croquis reste relativement modeste et surtout ne fait pas usage de sortie dites analogiques. Le *timer* 2 par exemple est celui utilisé pour piloter les broches 3 et 11 en PWM.

AMÉLIORATION DU CROQUIS

Je ne doute pas que certains lecteurs en voyant une succession de lignes comme **myCharlie.ledWrite(myLeds...** ont dû avoir envie de hurler. Rassurez-vous, c'était délibéré.

Pourquoi ? Tout simplement pour montrer qu'aligner ainsi des lignes identiques pique littéralement les yeux d'un programmeur. La répétition et les copier/coller de code c'est le mal (ou alors vous êtes un dieu du code machine et savez très exactement ce que vous faites et surtout ce que le compilateur va faire). La répétition est le

travail de la machine, pas du développeur. Observons donc attentivement ce que fait ce morceau précis du code. Nous utilisons un *ET* logique pour obtenir une valeur à partir de l'heure et des minutes. Si cette valeur est 0 (`OFF`) la led est éteinte, sinon elle est allumée.

Exemple : `tm.Hour` vaut 9. Si nous voulons savoir s'il faut allumer la led correspondant au 8 (4ème bit) nous faisons 9 ET 8 = 8, car 01001 (9) ET 01000 (8) = 01000 (8). En code, cela donne `9 & 8 == 8`. Le résultat n'est pas 0 et donc la led est `ON`. Nous pouvons également utiliser une syntaxe plus compréhensible avec `(1 << 3)`, soit la valeur 1 qu'on décale trois fois vers la gauche en binaire : 00010 (un fois), 00100 (deux fois), 01000 (trois fois) = 8.

Nous pourrions donc écrire nos lignes ainsi :

```
myCharlie.ledWrite(myLeds[0],
(tm.Hour & (1<<4)));
myCharlie.ledWrite(myLeds[1],
(tm.Hour & (1<<3)));
myCharlie.ledWrite(myLeds[2],
(tm.Hour & (1<<2)));
myCharlie.ledWrite(myLeds[3],
(tm.Hour & (1<<1)));
myCharlie.ledWrite(myLeds[4],
(tm.Hour & (1<<0)));
```

Ce qui est plus lisible, mais pas beaucoup mieux. Mais on distingue plus facilement la manière d'éviter la répétition. Nous avons d'un côté un index qui augmente (0 à 4) et de l'autre un décalage qui diminue (4 à 0). Avec un peu de gymnastique, on peut donc transformer cela en une boucle :

```
for(int i=0; i<5; i++) {
  myCharlie.ledWrite(myLeds[4-i],
(tm.Hour & (1<<i)));
}
```

Ce qui nous donne :

- 1er tour : i=0, `myLeds[4-0]`, `1<<0` = 1,
- 2ème tour : i=1, `myLeds[4-1]`, `1<<1` = 2,
- 3ème tour : i=2, `myLeds[4-2]`, `1<<2` = 4,
- 4ème tour : i=3, `myLeds[4-3]`, `1<<3` = 8,
- 5ème tour : i=4, `myLeds[4-4]`, `1<<4` = 16,
- i>5, fini ! **DB**

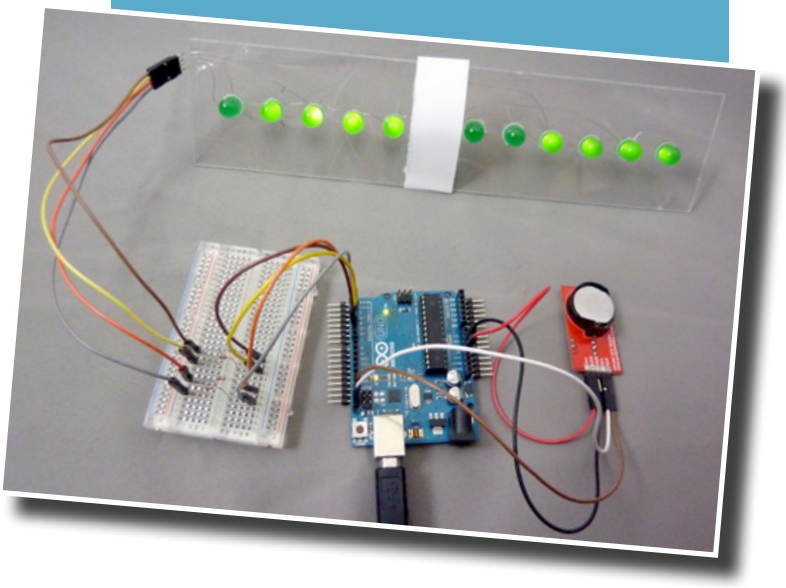
CE QUE NOUS AVONS APPRIS

Nous avons appris plein de choses ici. Nous savons utiliser une RTC DS1307 pour ajouter une gestion du temps relativement stable pour n'importe lequel de nos projets. « relativement », car le DS1307 est sensible à la température, un équivalent plus stable est le DS3231. Vous vous souvenez de la sonnette intelligente du premier numéro de Hackable ? Et si vous ajoutiez l'heure du dernier coup de sonnette sur l'afficheur LCD ?

Nous savons également piloter une myriade de leds avec seulement quelques sorties de la carte Arduino tout en ayant bien pris en compte les limites de la technique de charlieplexing. Mieux encore, je vous recommande très fortement la lecture du code de la bibliothèque `Chaplex` et ses exemples. Il est relativement succinct, mais très enrichissant.

Nous avons également vu brièvement ce qu'est une interruption, une ISR, un *timer* ainsi que la manière de réduire une douzaine d'ignobles copier/coller de code en deux jolies boucles sexy.

Mais plus amusant encore, nous avons maintenant une belle horloge qui non seulement nous permet de nous entraîner à décoder des valeurs binaires de tête mais, en plus, risque de laisser vos amis dubitatifs sinon admiratifs... ou méfiants quant à votre santé mentale et/ou vos tendances masochistes.



CONTRÔLEZ VOTRE ARDUINO AVEC LA TÉLÉCOMMANDE DE VOTRE TV

Denis Bodor

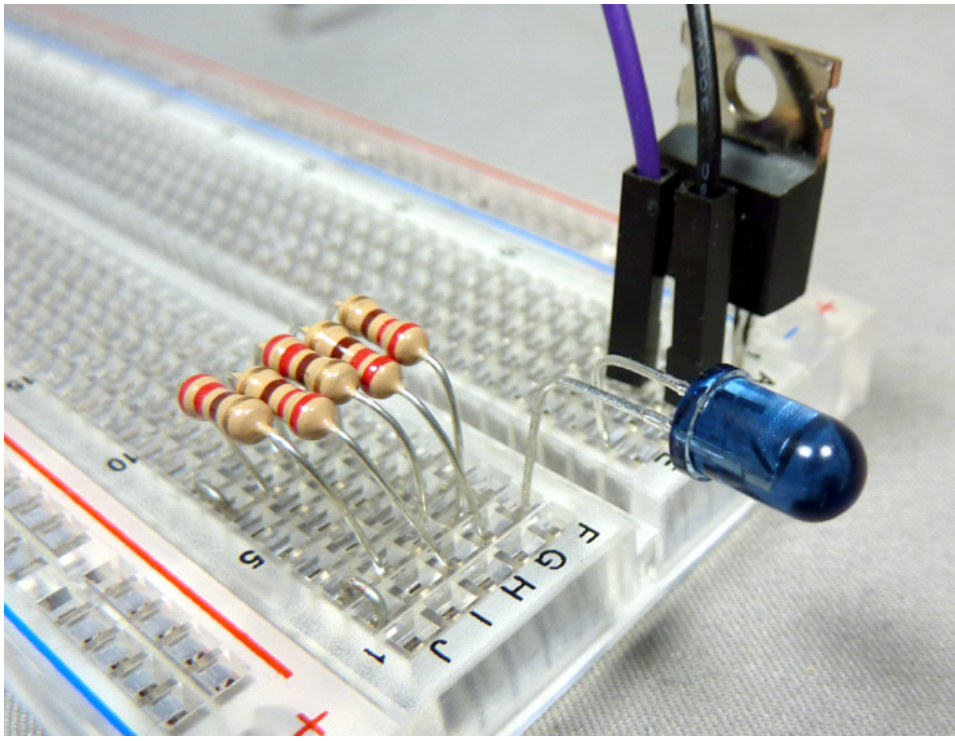
De nos jours, certains rares appareils hifi utilisent des télécommandes Bluetooth mais la grande majorité repose encore sur la bonne vieille télécommande infrarouge, en particulier la quasi totalité des téléviseurs et lecteurs DVD de salon. Ce qu'il y a d'amusant avec cette technologie c'est qu'avec une carte Arduino et quelques simples petits composants, on peut être à la fois émetteur et/ou récepteur de ces signaux.

NIVEAU	TEMPS	BUDGET
	15 minutes	environ 60 € (Arduino inclus)

Vous l'avez sans doute compris mais je vais le dire autrement juste pour le plaisir : pour quelques euros (et par « quelques » je veux dire 2 ou 3) vous pouvez télécommander votre Arduino mais aussi l'utiliser comme télécommande « maison ». L'une comme l'autre utilisation ouvre des perspectives absolument intéressantes tant au niveau purement sérieux que pour faire de bonnes blagues bien lourdes (du type la TV qui s'éteint ou la lecture du DVD qui saute un chapitre toute seule au hasard).

Contrairement à ce qu'on pourrait penser, l'histoire de la télécommande pour appareil hifi ne commence pas par l'utilisation de l'infrarouge mais avec, tout d'abord, une commande filaire déportée puis l'utilisation de communications à fréquences. Il y a même eu une tentative d'utilisation d'un signal acoustique grâce à un système piézoélectrique reposant sur une fréquence juste au delà de la capacité de perception humaine. Les ultrasons produits n'étaient cependant par très compatibles avec les animaux. Les chiens en particulier n'appréciaient pas vraiment le système. La notion même de contrôle à distance par ondes radio a été mise en évidence (et en pratique) par un certain *Nikola Tesla* (hé oui, encore lui) et sa démonstration de bateau télécommandé (brevet US 613809).

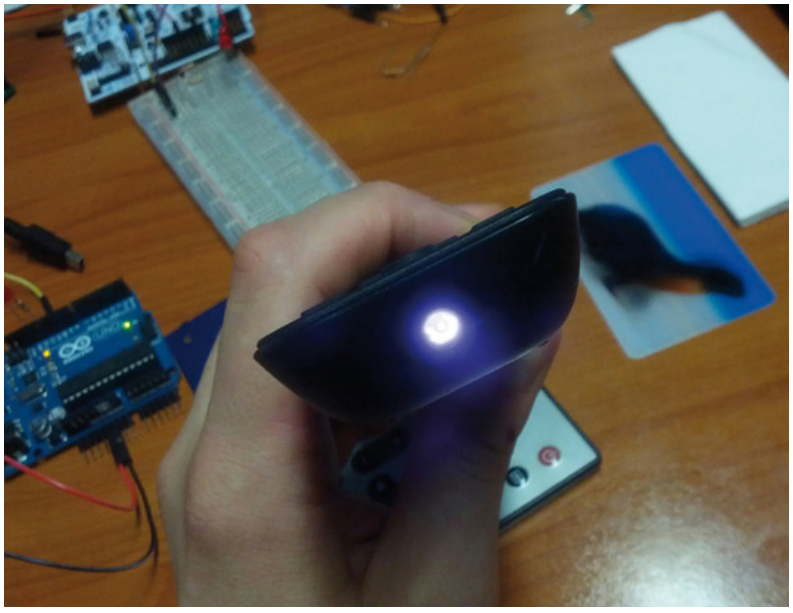
Après diverses tentatives fonctionnelles mais peu adaptées (trop chers, pas assez fiables, trop complexes) de la part des constructeurs de TV, une solution simple a vue le jour, permettant enfin à tous les robert-de-canapé de ne plus avoir à faire un effort titanesque pour se lever et changer de chaîne : la télécommande infrarouge. Étonnamment, la solution est née à la fin des années 70 en raison d'un besoin lié non pas au téléviseur mais au produit annexe qu'était le récepteur teletext. Alors que la TV n'avait besoin que de quelques signaux de contrôle, le teletext prévu pour afficher des messages sur des dizaines de pages, avait besoin de quelque chose de plus complet que « suivant »/« précédent ». La société ITT fut l'une des premières à généraliser le système infrarouge et un protocole a même été nommé en conséquence. D'autres constructeurs ont suivi le mouvement, certains avec des protocoles identiques et d'autres préférant jouer en solitaire.



La télécommande de la TV a bouleversé la perception de l'appareil en créant une impression d'interactivité pour le téléspectateur, puisque celui-ci pouvait commencer à zapper « efficacement ». La plupart des documents décrivant l'histoire de la télévision parle de réelle interaction mais, personnellement, je m'en tiendrais à « l'illusion d'interactivité ». En effet, même si le téléspectateur ne regarde plus une chaîne par simple paresse de se lever, ses libertés d'interaction sont très limitées, tout comme un mouton qui a l'illusion d'être libre d'aller où il veut... dans son enclot. Au cas où vous en doutiez, le téléviseur ne fait plus parti de mon habitat naturel depuis longtemps, ce qui n'empêche pas, vous allez le constater, de s'amuser avec celle des autres et de recycler les télécommandes pour un autre usage.

LE PRINCIPE

Le principe de base du fonctionnement de n'importe quelle télécommande infrarouge est, tout simplement, d'émettre un rayonnement dans la longueur d'ondes de l'infrarouge proche (PIR) qui sera capter par un récepteur qui réagira en conséquence. Le principe est simple mais lorsqu'on parle de télécommande ceci ne suffit pas. Les infrarouges s'étendent des longueurs d'ondes de 700 nm à 1 mm. La gamme est découpée en trois grandes bandes spectrales (ISO), « proche », « moyen » et « lointain » ou en trois domaines (CIE), IR-A, IR-B et IR-C. Les télécommandes fonctionnent généralement sur des longueurs d'ondes de l'ordre de 940 nm. Retenir cette valeur n'est pas anodin car c'est une caractéristique que doivent respecter émetteur (led) et récepteur pour fonctionner correctement.



Au centre, la led IR en activité sur une télécommande Philips telle que vue par l'optique d'un smartphone Galaxy Nexus. A l'arrière plan, à gauche l'Arduino Uno d'expérimentation pour cet article, au centre une ST Nucleo F401RE, et à droite une image animée d'une otarie (?).

Les humains ne peuvent pas voir les infrarouges, les serpents oui, mais nous on a des pouces préhenseurs et on sait fabriquer des outils pour traiter et utiliser ces rayonnements (et toc !). Ainsi, votre smartphone ou votre appareil photo numérique peut capter le rayonnement infrarouge et vous l'afficher à l'écran. Si vous n'en n'avez jamais fait l'expérience, prenez simplement une télécommande classique, pointez-là vers l'objectif en appuyant sur un bouton. Vous devriez voir la led au bout de la télécommande se mettre à clignoter frénétiquement. La couleur que vous voyez, une sorte de violet/rose, n'est PAS de l'infrarouge. Le rayonnement est capté par le détecteur CMOS derrière l'objectif et reproduit sur l'écran LCD mais avec de la lumière visible. Vous voyez une interprétation du signal, rien de plus. Les infrarouges ne sont pas violet/rose.

Mais ce n'est pas tout. Même avec la bonne longueur d'ondes, il n'est pas possible de commander un téléviseur en faisant simplement clignoter une led. Afin d'éviter les interférences et les sources de perturbation, le signal envoyé et reçu par le système est modulé sur une certaine fréquence. Ainsi, lorsque la led de la télécommande est active, elle clignote à une fréquence de 36, 38 ou 40 Khz. Ce signal, qu'on appelle porteuse, est le seul (en principe) qui est correctement capté par le récepteur. Cette porteuse est modulée de façon à envoyer des informations numériques (des 0 et des 1). C'est un peu comme siffler un message. Vous produisez une seule note, et donc une fréquence, mais générez une mélodie en alternant les sons et les silences pour former le message.

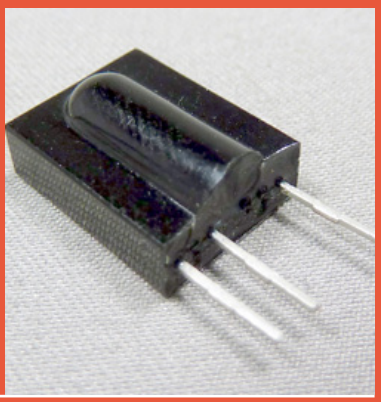
L'alternance que siffle une télécommande répond à un protocole choisi par un constructeur ou un autre. Il en existe une poignée mais les plus connus/utilisés sont, dans leurs ordres d'importance RC-5 (Philips), RC-6 (Philips), JVC, SONY, NEC, PANASONIC, LG.

Ainsi, jouer avec les signaux infrarouges dans le cadre de ce qui va suivre repose sur trois variables clés : une longueur d'ondes, une fréquence porteuse (*carrier frequency*) et un protocole.

CE QU'IL VOUS FAUT

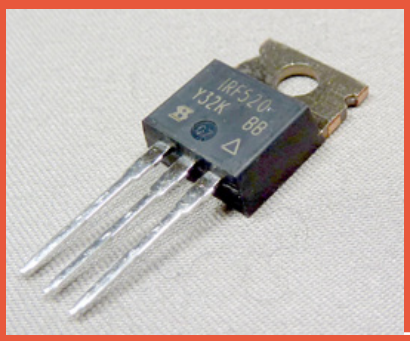
- Une carte Arduino : peu importe le modèle de carte, les seules choses dont nous aurons besoin c'est une entrée numérique, un lien série pour afficher des messages avec le moniteur de l'environnement de développement et éventuellement une ou plusieurs sorties. Tout ceci devrait même être facilement transposable à des « clones » comme Launchpad/Energia.

- Un récepteur TSOP1736 ou TSOP1738 ou équivalent. TSOP* est plus ou moins générique lorsqu'on parle de réception infrarouge (il ne faut pas confondre avec *Thin Small Outline Package* qui est un format de composant). C'est un capteur intelligent et non un composant passif comme une photorésistance ou LDR (voir article sur la boite à lettres qui tweet du numéro 2). Un TSOP1736, prévu pour une porteuse à 36 Khz par exemple, comprend un récepteur IR, un circuit de contrôle de gain, un filtre, un démodulateur et finalement un transistor pour fournir les données reçues sur la sortie. Les TSOP1738 et 1736 se différencient par la fréquence de la porteuse utilisée pour la démodulation. Notez qu'il s'agit d'une fréquence optimale, un 1736 pourra recevoir les signaux en 38 Khz, mais pas aussi bien qu'un 1738. TSOP1736 et 1738 sont obsolètes et remplacés chez le fabriquant Vishay par exemple, par les TSOP31236 et TSOP31238 qui fonctionnent de la même manière. Attention lors de votre achat de composants. Sur eBay par exemple, certains n'hésitent pas à en proposer à plus de 4 euros (+ 3 euros de port) alors que le composant, chez un fournisseur standard coûte autour d'un euro (Mouser 0,689 HT, Farnell 1,17 HT, RS 1,06 HT). De plus, via une source inconnue vous n'êtes pas sûr de recevoir un produit de qualité comme ceux d'un fabricant réputé comme Vishay.



- Une led IR TSAL6200 : des leds IR il en existe des quantités incroyables se déclinant en longueur d'ondes, courant d'alimentation, angle de diffusion, puissance, vitesse de commutation... Pour une valeur sûre, pas de miracle, ce modèle de led est celui présenté dans la documentation des récepteurs TSOP et est donc parfaitement adapté à l'envoi de signaux du type qui nous intéresse. La même remarque ici s'applique que celle pour le récepteur : faites attention aux prix. Nous avons acquis 20 exemplaires sur eBay pour un total de 2,90 € (+4,90€ de port), ce qui nous donne un prix de 0,39 € la led. Chez Mouser elle est à 0,457 HT, chez RS à 0,112 HT par 50 et chez Farnell à 0,28 HT par 5. Nous avons eu de la chance puisque selon les quantités et le coût du port le calcul est plus ou moins favorable au vendeur eBay. Une led sera accompagnée d'une résistance de 39 ohms permettant une alimentation sous 5V avec un courant de 96 mA.

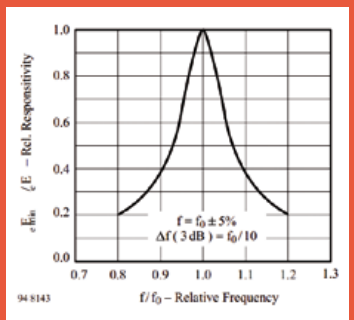
...CE QU'IL VOUS FAUT



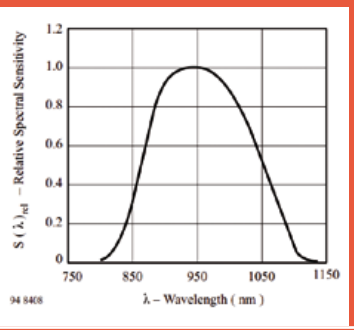
- un MOSFET IRF520 : une carte Arduino n'est pas faite pour fournir le courant que demande la led (100 mA), chaque sortie étant limitée à 40 mA. Nous devons donc utiliser une autre source d'alimentation, celle de l'Arduino (VCC) et un composant de contrôle simple : un MOSFET. C'est ce type de composant que nous avons utilisé dans le numéro précédent pour contrôler l'éclairage de la boite à lettres, car il est en mesure de piloter un courant de plus de 6,5A sous 100V (à 25-150°C) ce qui est largement suffisant pour une led ou une myriade de leds.

- Une ou des leds standards accompagnées de leurs résistances de 220, 330 ou 470 Ohms selon la couleur et leurs spécifications.

- Une télécommande IR : ancien matériel, récupération, marché aux puces, Espoir, Emmaüs ou simple télécommande d'un produit peu utilisé, peu importe... La seule chose qui compte c'est que le matériel fonctionne et utilise un protocole connu par la bibliothèque que nous allons utiliser. La probabilité est énorme, je ne suis JAMAIS tombé sur une télécommande fonctionnelle utilisant un protocole étrange et/ou inconnu (des télécommandes muettes, oui, plein).

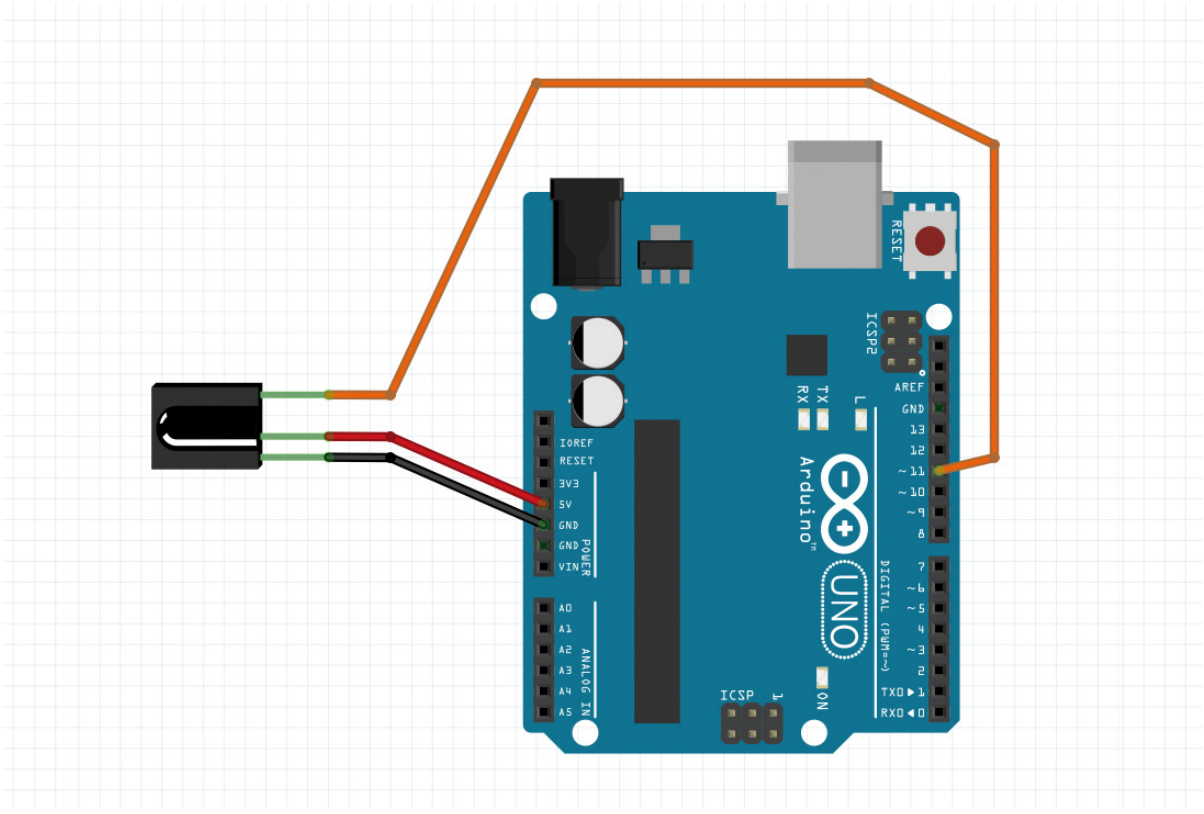


- Un récepteur TSOP est « calé » sur une fréquence porteuse mais fonctionnera avec une fréquence proche. La qualité de réception ne sera simplement pas optimale - source documentation Vishay des TSOP17xx



- La réception idéale se fera avec un rayonnement IR d'une longueur d'ondes de 940 nm. Avant et après, la capacité de réception du signal chute rapidement - source documentation Vishay des TSOP17xx

LE MONTAGE 1 : ON COLLECTE



Cet article ne contient pas un montage mais trois car nous allons travailler par étapes : la collecte d'informations, la réaction de l'Arduino en fonction des messages d'une télécommande et enfin l'envoi de signaux.

Chaque télécommande, en fonction du modèle de matériel qu'elle contrôle et de son fabriquant, utilise des codes différents par boutons. Certaines télécommandes utilisent RC-5, d'autres RC-6 mais certaines peuvent également utiliser un autre protocole connu ou inconnu. Généralement les fabricants occidentaux et surtout chinois essaient de rester dans les clous. Ce n'est pas pour vous faire plaisir mais simplement parce qu'en réception il existe des circuits et des morceaux de programmes compatibles RC-5, RC-6, etc. Ceci évite de devoir travailler sur un nouveau protocole et d'investir de l'argent en R&D. Ainsi, ce sont surtout les « grosses marques » qui se permettent d'avoir leurs propres protocoles pour tous leurs produits (Sony par exemple).

Bien entendu, afin d'éviter qu'une télécommande n'impacte le fonctionnement d'un autre appareil, chaque bouton envoi un message ou code différent. Pour que notre montage puisse lui-aussi réagir comme le matériel d'origine, il nous faut connaître les fameux codes. C'est ainsi que fonctionne également les télécommandes universelles qui, en plus d'être équipées d'une led pour émettre, sont pourvues d'un récepteur de type TSOP (ou d'un récepteur simple accompagné d'un processeur qui analyse le signal). Il suffit de placer la nouvelle télécommande en mode apprentissage et de lire et copier ce qui est envoyé par le matériel d'origine.

La connexion du TSOP1736 à l'Arduino est simplissime puisque cela consiste à relier la masse à la masse, VCC à la broche 5V de l'Arduino et la broche restante, la sortie du TSOP, à la broche 11 utilisée en entrée. Et voilà notre récepteur prêt à servir.

LE CROQUIS 1


```
#include <IRremote.h>

int RECV_PIN = 11;

IRrecv irrecv(RECV_PIN);
decode_results results;

void setup() {
  Serial.begin(9600);
  irrecv.enableIRIn();
}

void loop() {
  if (irrecv.decode(&results)) {
    switch(results.decode_type) {
      case UNKNOWN:
        Serial.print("UNKNOWN: ");
        break;
      case RC6:
        Serial.print("RC6: ");
        break;
      case RC5:
        Serial.print("RC5: ");
        break;
      case NEC:
        Serial.print("NEC: ");
        break;
      case SONY:
        Serial.print("SONY: ");
        break;
      case LG:
        Serial.print("LG: ");
        break;
      case JVC:
        Serial.print("JVC: ");
        break;
      case PANASONIC:
        Serial.print("PANASONIC: ");
        Serial.print(results.panasonicAddress,HEX);
        Serial.print(" - ");
        break;
      default:
        Serial.print("???: ");
    }

    Serial.print(results.value, HEX);
    Serial.print(" ");
    Serial.print(results.bits, DEC);
    Serial.println(" bits");
    irrecv.resume();
  }
}
```

À PROPOS DU CROQUIS 1

Avant toutes choses, pour utiliser ce croquis (et les suivants) il vous faut installer la bibliothèque *Arduino-IRremote* de Ken Shirriff. Pointez simplement votre navigateur sur **https://github.com/shirriff/Arduino-IRremote** et utilisez le bouton sur la droite « *Download Zip* » pour la télécharger. Dans le Zip se trouve un répertoire **`Arduino-IRremote-master`** que vous devrez copier dans le sous-répertoire **`libraries`** de votre dossier des croquis Arduino en le renommant **`IRremote`**.

Une fois le croquis compilé et chargé dans l'Arduino, les signaux envoyés par une télécommande vers le TSOP seront décodés et des messages s'afficheront dans le moniteur série :

```
RC6: 800F8423 36 bits
RC6: 800F8423 36 bits
RC6: 800F8423 36 bits
???: FFFFFFFF 0 bits
RC6: 800F8423 36 bits
RC6: 800F8423 36 bits
RC6: 800F0412 36 bits
RC6: 800F0412 36 bits
RC6: 1000A 20 bits
???: FFFFFFFF 0 bits
RC6: F 20 bits
RC6: F 20 bits
UNKNOWN: 58D4190A 32 bits
UNKNOWN: 4BD03788 32 bits
UNKNOWN: 25AE7EE0 32 bits
```

Nous avons ici des codes RC-6 36 bits d'une télécommande Toshiba et des codes RC-6 20 bits d'un matériel Philips. Remarquez les lignes **`UNKNOWN`** et **`???`** qui sont des signaux parasites reçus et décodés mais qui ne correspondent à aucun protocole connu par la bibliothèque. Il peut s'agir de n'importe quoi, une réverbération du signal de la télécommande sur les murs ou autres, des signaux sur la même longueur d'ondes provenant d'ailleurs ou tout simplement de messages partiels ou endommagés envoyés par la télécommande.

Pour faire fonctionner la bibliothèque de Ken, il faut tout d'abord l'inclure dans notre code. C'est l'objet de la première ligne. Pour qu'elle puisse ensuite être utilisée, nous devons définir la broche à utiliser en entrée pour recevoir les données envoyées par le TSOP. Ici c'est la broche 11 :

```
int RECV_PIN = 11;
IRrecv irrecv(RECV_PIN);
```

La bibliothèque s'occupe de la majeure partie du travail et stocke le résultat dans une variable complexe, une structure d'un type particulier. Nous devons donc définir cette variable avec :

```
decode_results results;
```

`results` est destiné à contenir ces informations structurés, c'est une variable de type **`decode_results`**. C'est tout ce qu'il nous faut en terme de

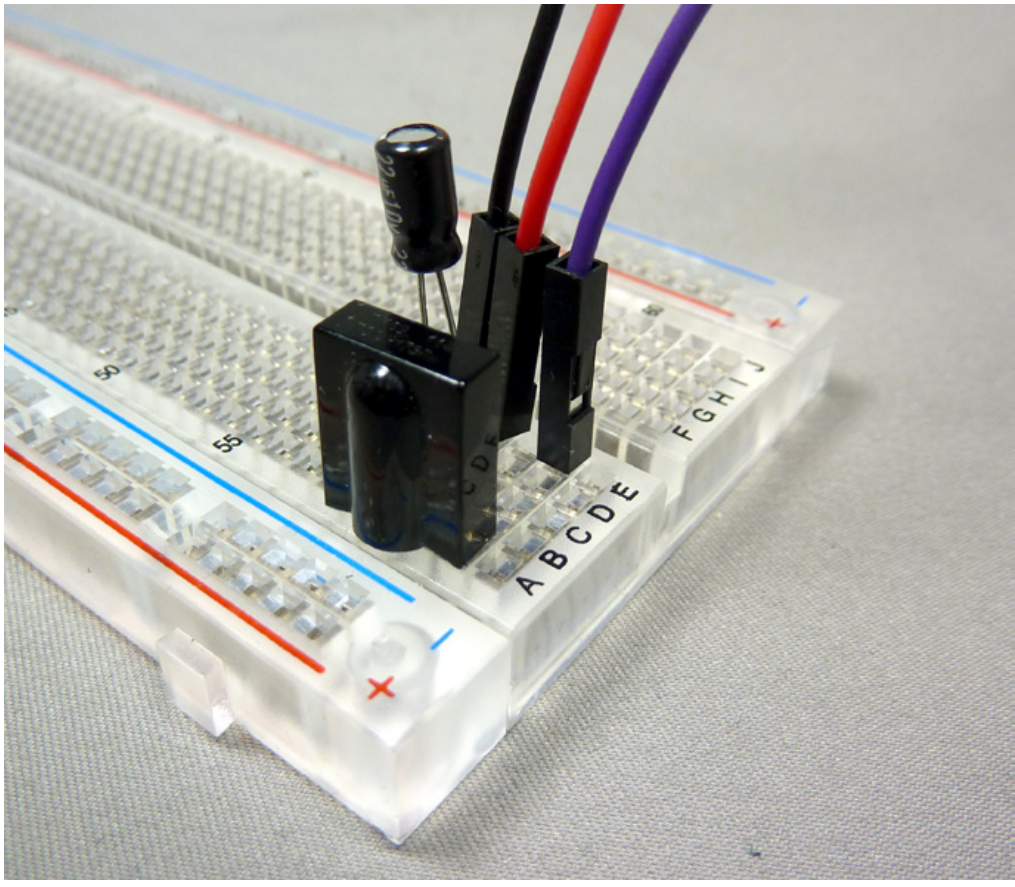
variables globales, nous pouvons passer à la fonction **setup()**. Celle-ci configure la communication série en 9600 bps afin qu'on puisse envoyer des messages au moniteur et met en marche la réception de signaux IR :

```
Serial.begin(9600);
irrecv.enableIRIn();
```

Nous pouvons passer à notre boucle principale, la fonction **loop()**. Le principe de fonctionnement de la bibliothèque de Ken est le suivant : dès qu'un signal est reçu, un appel à la méthode **decode** retourne une valeur non nulle. La méthode prend en argument un pointeur vers notre structure. Sans entrer dans des considérations complexes sur cette notion, la méthode a besoin de savoir où elle peut mettre les données décodées s'il y en a. En C/C++ un « où » est synonyme d'une adresse en mémoire, un endroit, un pointeur vers cet endroit. **results** est le nom de notre variable/structure, c'est l'étiquette du bocal où mettre les données, ce n'est pas ce qu'attend la méthode. Elle veut un « où » pas un « qui ». Pour spécifier le bon argument, l'endroit où est **results**, on fait simplement précéder le nom de la variable de **&** en C/C++.
En résumé :

- **results** : le bocal à données,
- **&results** : l'endroit où est le bocal.

Nous pouvons maintenant comprendre le sens de l'instruction suivante :



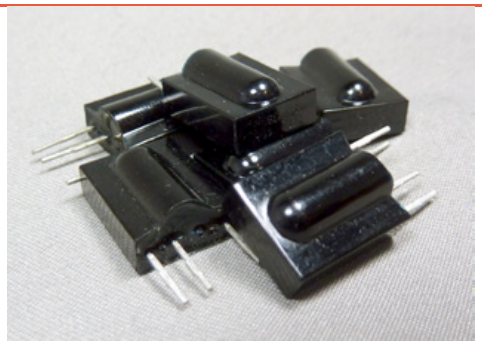
```
if (irrecv.decode(&results))
```

qui peut se traduire en « si l'appel à la méthode **decode** de **irrecv** à qui je passe l'adresse de **results** me retourne quelque chose qui n'est pas zéro alors... ». Ou, en d'autres termes, si on tente de décoder un message et que l'opération réussie alors **results** contiendra des données que nous pouvons manipuler.

Dans la porté du **if**, nous nous amusons tout d'abord à tenter d'extraire le type de message que nous avons reçu. Celui-ci est stocké dans **results.decode_type**. Nous pourrions utiliser des **if** pour cela :

```
if(results.decode_type == RC5)
  Serial.print("RC5: ");
if(results.decode_type == RC6)
  Serial.print("RC6: ");
[...]
```

et ainsi de suite... Mais ceci n'est pas très beau. Pour ce genre de situations où une variable peut avoir une valeur parmi plusieurs, le langage C/C++ permet d'utiliser une syntaxe spécifique : le **switch**/**case**. Cette syntaxe est la suivante :



```
switch(variable) {
  case valeur1:
    // faire un truc
    break; // on sort du test
  case valeur2:
    // faire un autre truc
    break; // on sort du test
  case valeur3:
    // faire un encore un autre truc
    break; // on sort du test
  default:
    // faire un truc si rien d'autre
    // ne correspondait
}
```

On *switch* sur `variable` et pour chaque *case* on défini une action à mener. `break` permet de stopper le processus. Ainsi, si `variable` est/vaut `valeur1`, on s'arrête là. En l'absence de `break` dans ce cas, l'action pour `valeur2` sera également accomplie. Ceci peut être intéressant dans certaines situations mais pas ici.

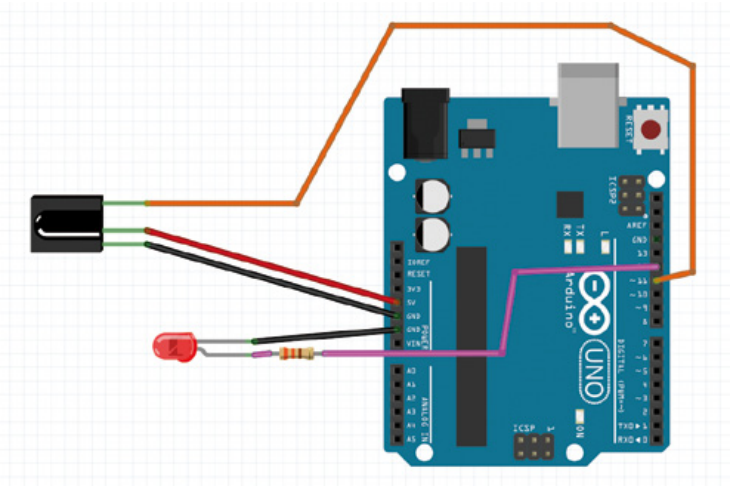
Une fois qu'on a affiché sur le moniteur série le type de message reçu, il ne nous reste plus qu'à présenter son contenu. Pour se faire on utilise un autre membre de la structure : `results.value`, la valeur du message (car celui-ci est numérique). On demande un affichage de la valeur en notation hexadécimale. Et puis tant qu'on y est, on affiche également le nombre de bits du message, contenu dans le membre `results.bits`.

Notez le cas particulier du protocole Panasonic qui découpe les données en une partie adresse et une partie valeur.

Enfin, et c'est à ne surtout pas oublier, nous devons utiliser `irrecv.resume()` afin de pouvoir signaler la fin du traitement et recevoir le message suivant.

Le but de la collecte est de vous créer un dictionnaire établissant la correspondance bouton/code en testant méticuleusement chacun d'entre eux et en copiant/collant les codes dans un coin. Notez qu'il faut procéder avec soin et patience et que la plupart des télécommandes récentes utilisent deux codes par bouton. Exemple, avec la télécommande Toshiba une première pression unique sur un bouton envoie `800F045B` mais la seconde utilisation envoie `800F845B`. Une troisième pression envoie à nouveau le premier code, etc. Seul un bit change entre les deux codes et le bouton se comporte comme une bascule.

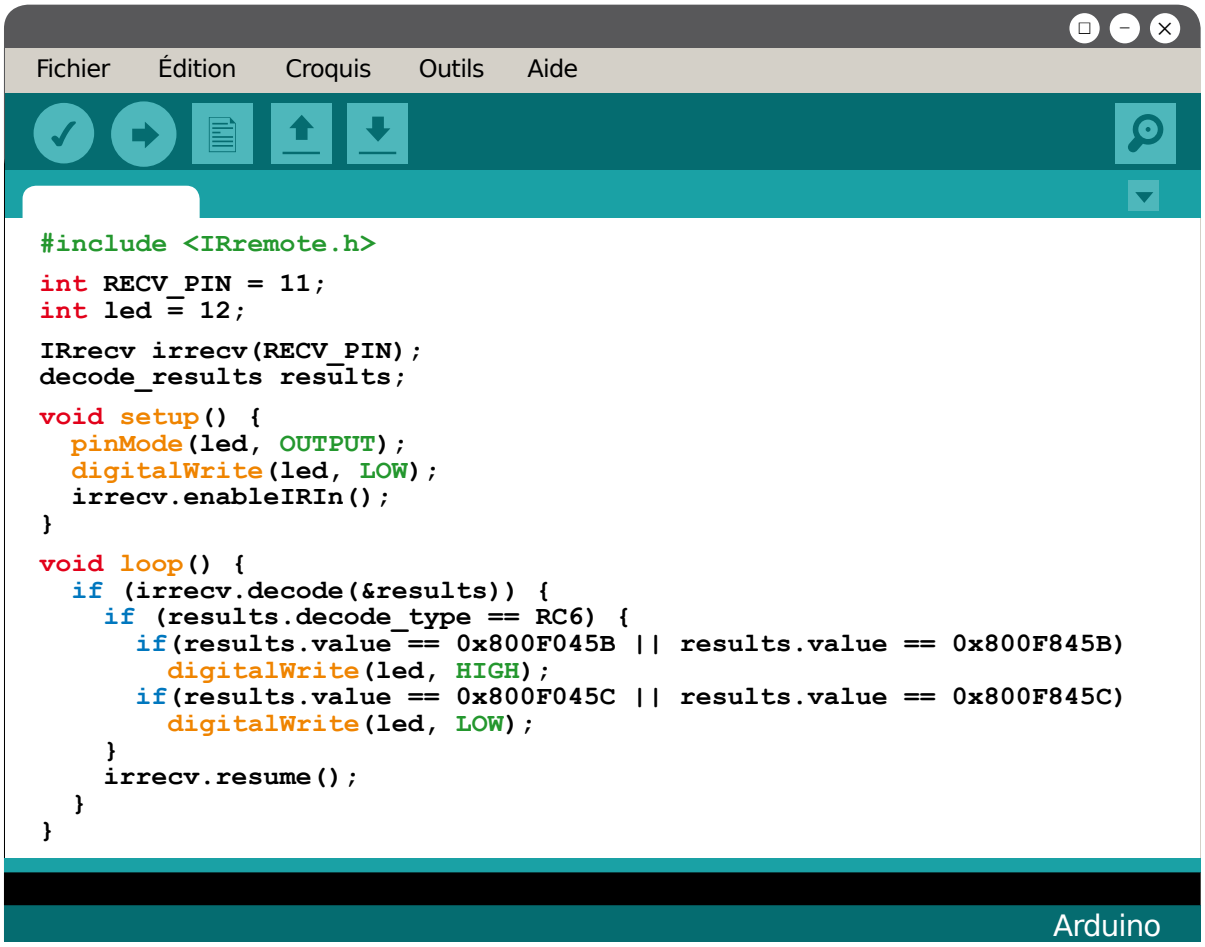
LE MONTAGE 2 : ON REÇOIT



Matériellement la configuration ne change pas beaucoup puisque la seule chose que nous ajoutons est une simple led et sa résistance. Peut-être vous demandez-vous pourquoi ne pas simplement utiliser la led « L » soudée sur l'Arduino Uno. C'est purement fonctionnel. En effet, puisque nous comptons changer l'état de la led en fonction de l'utilisation des boutons de la télécommande, autant tester la porté de l'ensemble. Il est donc préférable que le témoin lumineux puisse être visible de loin.

Notre led sera connectée à la broche 12. Un choix totalement arbitraire.

LE CROQUIS 2




```
#include <IRremote.h>

int RECV_PIN = 11;
int led = 12;

IRrecv irrecv(RECV_PIN);
decode_results results;

void setup() {
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
  irrecv.enableIRIn();
}

void loop() {
  if (irrecv.decode(&results)) {
    if (results.decode_type == RC6) {
      if(results.value == 0x800F045B || results.value == 0x800F845B)
        digitalWrite(led, HIGH);
      if(results.value == 0x800F045C || results.value == 0x800F845C)
        digitalWrite(led, LOW);
    }
    irrecv.resume();
  }
}
```

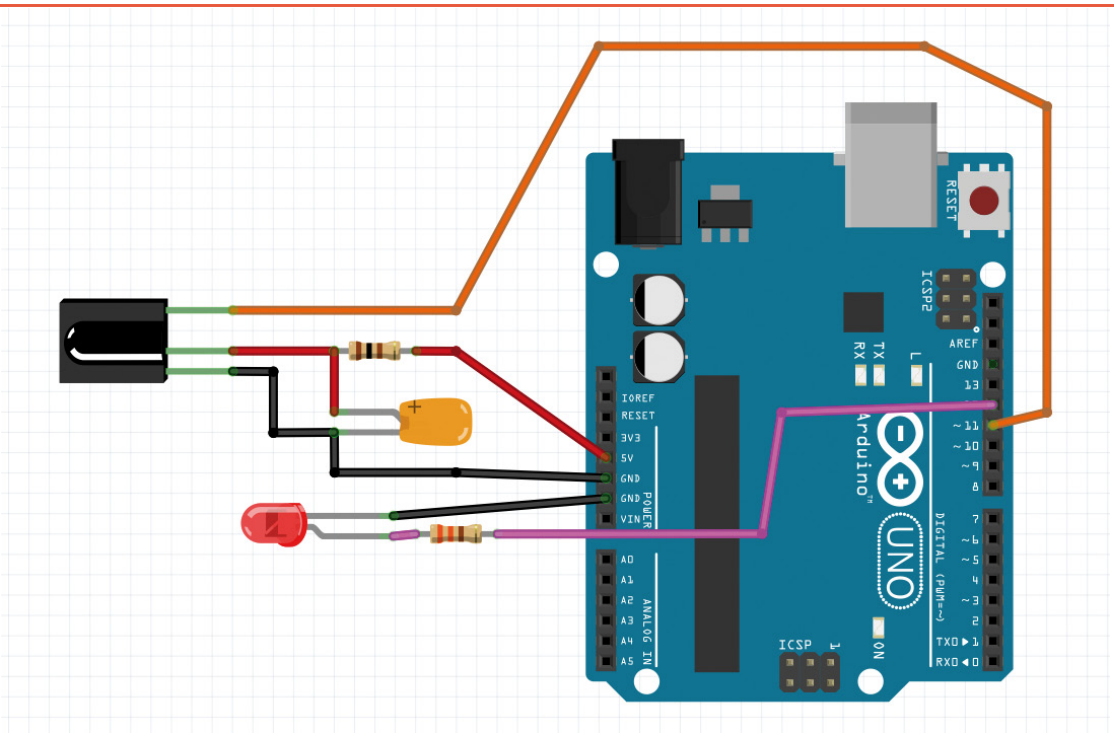
À PROPOS DU CROQUIS 2

Nous avons choisi d'utiliser deux boutons particuliers de la télécommande, l'un est rouge, l'autre est vert. En utilisant le croquis précédent, nous avons conclus que :

- le bouton rouge utilise alternativement les codes **800F045B** et **800F845B**,
- le bouton vert alternativement les codes **800F045C** et **800F845C**.

Cette télécommande utilise justement le principe de bascule dont nous venons de parler. Une pression sur un bouton envoi un code, la seconde un autre. En revanche, en pressant un bouton de manière continue c'est le même code qui est envoyé à répétition. Comme nous influençons l'état de la led en sortie en fonction de deux boutons, nous prenons simplement les deux codes en compte. Notre `if` contient donc les caractères `||` signifiant un OU logique. Ceci nous évite d'utiliser deux conditions `if` à chaque fois.

Petite précision qui peut être importante, nous avons conditionné l'interprétation des messages/codes au fait que leur type soit celui de la télécommande (RC-6 ici). Ceci nous permet de limiter non seulement le traitement de l'information aux seules conditions



La documentation du TSOP recommande d'ajouter un condensateur et une résistance afin de nettoyer la tension d'alimentation et d'éviter que des variations parasites ne perturbent le composant.

qui nous intéressent mais également les interférences. S'il ne s'agit pas de RC-6 nous passons simplement notre tour.

Le croquis dans son ensemble est très simple et nous pouvons facilement l'étendre. Nous pouvons ainsi traiter et utiliser n'importe quel code de la télécommande, non seulement pour piloter n'importe quelle sortie mais également n'importe quelle autre fonctionnalité que nous souhaitons programmer.

Précisons tout de même que l'alternance des codes peut devenir un problème. En effet, imaginez que nous ayons le croquis suivant mais que nous ajoutons les deux codes d'un autre bouton, mais cette fois en changeant l'état de la led à `HIGH` pour l'un et `LOW` pour l'autre. Appelons ce bouton *play*. Imaginez ensuite le scénario suivant :

- *play* (code 1) : `HIGH`
- *play* (code 2) : `LOW`
- rouge : `HIGH`
- vert : `LOW`

Ça à l'air de marcher. Poursuivons :

- *play* (code 1) : `HIGH`
- vert : `LOW`
- *play* (code 2) : `LOW`

Aïe ! On était en droit de s'attendre à ce que le bouton *play* rallume la led. L'approche est mauvaise puisque l'état de la bascule n'est pas dépendant du tout de l'état de la led. La télécommande n'a aucune connaissance de cela, pour elle le bouton *play* envoi simplement alternativement l'un et l'autre code, point.

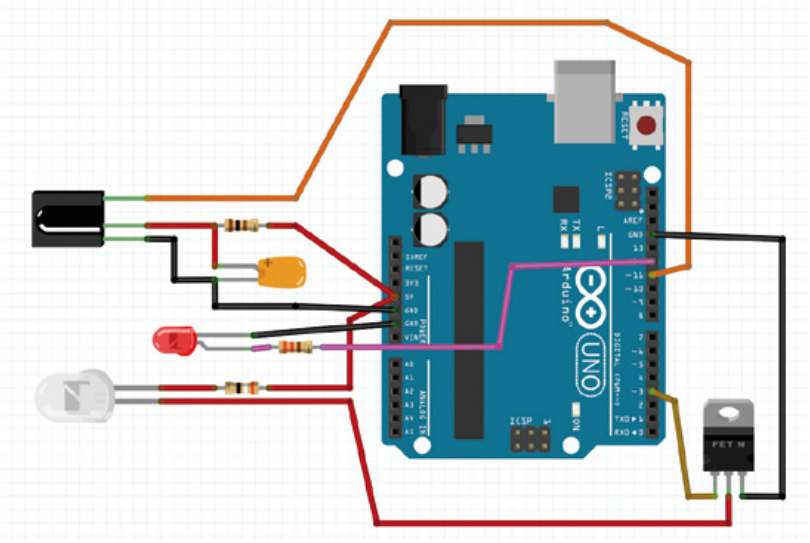
En réalité, il faut effectivement prendre en compte les deux codes du bouton et gérer la bascule côté Arduino :

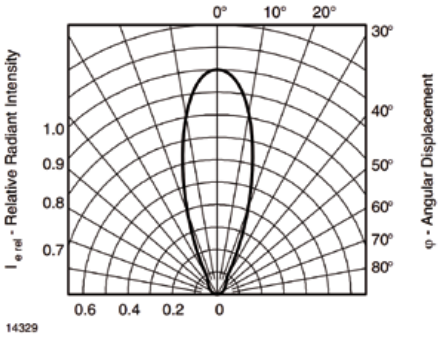
```
if(results.value == 0x800F840C ||
results.value == 0x800F040C) {
  digitalWrite(led, !digitalRead(led));
  delay(500);
}
```

Un `digitalRead()` en argument d'un `digitalWrite()` ? Pas exactement, il faut prendre en compte le caractère `!` dans la ligne. Celui-ci inverse la valeur. Ainsi, si `digitalRead(led)` retourne `LOW` (`0`), c'est `HIGH` (`1`) qui est utilisé avec `digitalWrite()`. Si c'est `HIGH` qui est lu, c'est `LOW` qui est utilisé. Cette simple ligne prend l'état de la sortie et l'inverse.

Pour en terminer avec la partie réception, sachez que si vous rencontrer des problèmes étranges comme une réception importante de codes et de messages d'un type inconnu, il peut s'agir d'un problème de « bruit » sur l'alimentation du TSOP. La documentation Vishay recommande d'ajouter une résistance de 100 ohms sur la ligne d'alimentation (+5V) ainsi qu'un condensateur de 4,7µF entre l'alimentation et la masse afin de réduire les parasites qui peuvent perturber le fonctionnement interne du TSOP. Ceci nous donne donc une version améliorée du montage.

LE MONTAGE 3 : ON ENVOI





L'angle de diffusion de la led TSAL6200 et très réduit. Ceci est parfaitement adapté pour une télécommande qui est un outil directionnel mais pas nécessairement pour d'autres usages.

Ok, je le reconnais, ça se complique un peu. Mais en se concentrant uniquement sur les éléments ajoutés ce n'est pas grand chose. Nous avons ajouté la led TSAL6200 et une résistance 39 Ohm afin de fournir environ 100 mA en 5V. Une sortie de l'Arduino n'est pas en mesure de fournir autant de courant. Celui-ci provient donc de la broche 5V. Nous devons donc contrôler l'état de la led avec une sortie et plus exactement la broche 3. Cette sortie ne peut être choisie arbitrairement car elle est définie directement dans la bibliothèque de Ken. En effet, pour pouvoir avec précision générer la porteuse de 36 Khz, le développeur a décidé d'utiliser la PWM (pseudo sortie analogique, cf article sur le sujet dans Hackable 1).

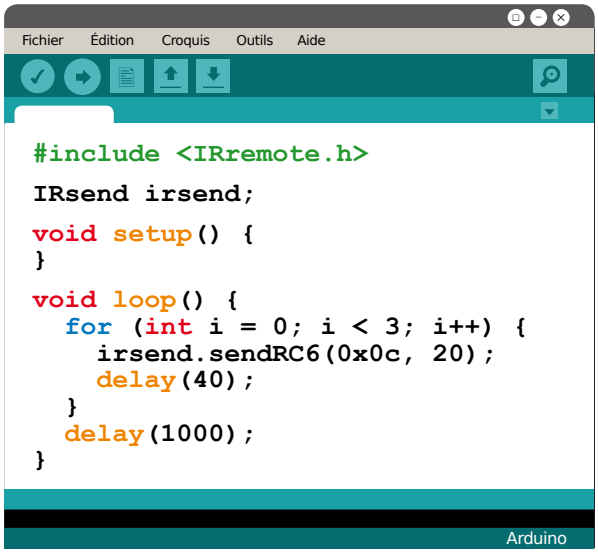
Pour contrôler la led avec la sortie 3, nous utilisons un MOSFET N de modèle IRF520. C'est un composant très courant et facile à trouver. Celui-ci possède trois broches : source, drain et grille. Il permet, en jouant sur la tension appliquée entre la grille et le drain, de laisser ou non passer un courant entre le drain et la source.

Il nous suffit donc de connecter la led et sa résistance à +5V puis à la broche drain et la source à la masse. On relie ensuite la broche grille à la sortie 3 de l'Arduino pour contrôler le passage du courant. Si la sortie est à l'état haut (`HIGH`) le courant drain/source passe et la led s'allume. Si la sortie est à l'état bas, la led s'éteint. L'IRF520 est parfaitement adapté à cette tâche, ce qui n'est pas forcément le cas d'autres MOSFET car la vitesse de commutation est importante. 36 Khz c'est 36000 cycles par seconde, soit un cycle toutes les 27 microsecondes. Ce MOSFET a une vitesse de commutation de l'ordre de 70 nanosecondes maximum. Il n'y a donc aucune chance que notre fréquence puisse être trop importante.

Ce montage utilisant un MOSFET est non seulement nécessaire étant donné le courant à fournir à la led mais présente, aussi et surtout, un gros avantage. Il est possible d'alimenter de cette façon n'importe quel circuit jusqu'à 100V et 9A. Rien ne nous interdirait donc d'utiliser plusieurs leds, voir une quantité très importante si nécessaire.

Pourquoi plusieurs leds ? Ceci ne peut qu'augmenter le rayon d'action mais surtout assurer une large diffusion du signal. La TSAL6200 émet son rayonnement de manière très directionnelle avec un angle de diffusion de l'ordre d'une vingtaine de degrés. Il faut donc pointer correctement la led vers l'appareil à contrôler. Si vous souhaitez émettre un signal de manière omnidirectionnelle, vous n'avez d'autre solution que de multiplier les leds. Idéalement pour 360°, il vous en faudra 24, ce qui fait beaucoup de courant. Mais 2,4 A est quelque chose que l'IRF520 est parfaitement capable de piloter (avec un petit radiateur en plus et surtout une alimentation autre que celle fournie par l'Arduino). Bien entendu la réflexion du signal IR sur les murs améliorera encore les choses dans une pièce standard.

LE CROQUIS 3



À PROPOS DU CROQUIS 3

Quoi ? C'est tout ? Hé oui, il n'y a rien de plus à faire pour envoyer, par exemple le code `0000C` en RC-6 (bouton *power* sur notre télécommande Philips). La bibliothèque de Ken se charge de tout le reste. Tout ce que vous avez à faire est de déclarer un objet `IRsend` et ensuite d'utiliser sa méthode `sendRC6` pour envoyer un code avec ce protocole. Les arguments à utiliser sont le message à envoyer (le code) et sa taille en bit. Ces deux informations proviennent directement de notre phase de collecte de codes.

D'autres protocoles/méthodes sont utilisables comme `sendNEC`, `sendSony`, `sendRC5`, `sendDISH`, `sendSharp`, `sendPanasonic`, `sendJVC` ou `sendSAMSUNG`.

En RC-6, le protocole veut que le code soit envoyé trois fois avec un délai de 40 ms. Nous ajoutons donc une boucle `for` basée sur la variable `i` afin de répéter trois fois l'émission. Nous marquons une pause de 1 seconde et répétons toute la procédure. Notre croquis va donc envoyer le signal du bouton *power* toutes les secondes au téléviseur. Bien entendu, cela fonctionne sans problème, dès la mise sous tension du montage, la TV, si elle est allumée, s'éteint immédiatement et part dans un cycle allumage/extinction sans fin. Je vous laisse le soin d'aller plus loin et d'imaginer une utilisation plus pratique d'un tel système. **DB**

CE QUE NOUS AVONS APPRIS

Nous avons appris énormément avec cet article. Tout d'abord nous avons acquis une connaissance du fonctionnement des systèmes utilisés par la plupart des matériels hifi/TV pour offrir à l'utilisateur un certain confort. Nous avons constaté que la quasi-totalité des constructeurs faisait logiquement usage de la technologie la plus fiable et économique mais que rien ne nous empêchait d'à la fois interpréter les signaux et les copier. Il n'y a strictement aucune sécurité ou protection d'aucune sorte. Implicitement nous avons appris comment fonctionnaient les télécommandes dites universelles et sommes maintenant parfaitement capable d'en construire une très facilement.

Les possibilités qui s'offrent maintenant à nous sont assez excitantes puisque nous pouvons faire en sorte que nos montages soient télécommandables. Nous pouvons également automatiser le contrôle des appareils à distance en incluant l'émission de signaux dans un système domotique basé sur Arduino. Vous connaissez certainement des personnes qui trouvent intelligent de laisser leur TV allumée durant leur vacances afin de faire croire que leur maison est occupée, afin de dissuader les voleurs. Ceux-ci ne sont cependant pas dupes. Un téléviseur allumé 24h/24 associé à deux ou trois autres indices renseignent rapidement sur la vacuité d'une habitation en faisant simplement l'effort malsain d'une petite surveillance régulière. En revanche, allumer et éteindre le téléviseur à distance, ainsi que quelques lampes, en fonction d'heures précises et en intégrant un paramètre aléatoire sera bien plus efficace...

Et ce n'est là qu'une idée parmi tant d'autres. Couplez réception avec le TSOP et émission avec une led TSAL6200 et vous obtenez par exemple un répétiteur pour augmenter la porté d'une télécommande. Vous pouvez également recycler une vieille télécommande inutilisée pour associer à certains boutons l'émission des codes d'extinction de plusieurs appareils. En d'autres termes, construire une télécommande universelle qui pilote, en même temps, plusieurs périphériques. Imaginez... un seul bouton pour tout éteindre en allant vous coucher...

VOS COLLÈGUES VOUS ÉNERVENT ?

CONTRE-ATTAQUEZ AVEC ARDUINO, UNE TÉLÉCOMMANDE ET UN LIVRE POUR ENFANTS !

D. B.

La nature est équilibre et harmonie. Afin de rétablir l'ordre dans l'univers, vous n'avez pas vraiment le choix. Si vos collèges vous rendent fou, la nature exige que vous leurs rendiez la pareille. Nous sommes là pour vous y aider parce que, pour nous, l'équilibre universel c'est important... Tout autant que le plaisir de rendre chèvre quelqu'un avec des bruits répétitifs, stupides et infantiles d'animaux et ce, à distance...

Que cela soit clair, ceci est un hack absolument ignoble réalisé en 15 minutes sur le coin d'un bureau. C'est d'ailleurs là tout l'esprit de l'exercice et surtout pas de faire quelque chose de bien propre et réfléchi.

Le point de départ de l'opération est une simple visite pour des raisons purement alimentaires dans un magasin *discount* (Lidl pour ne pas le nommer) à quelques minutes à pied de la rédaction. Alors que la plupart des clients visitent ce lieu et perçoivent les produits comme le commerçant l'attend, vous et moi savons qu'il existe une autre réalité. Elle se trouve juste derrière le monde qu'on superpose à notre regard pour nous empêcher de voir la vérité. La Matrice ?! Non, absolument pas, juste une myriade d'opportunités pour qui souhaite bidouiller. Ainsi la plupart des gens, lorsqu'ils voient « *Mon livre sonore : Les bébés animaux* » dans un rayon, interprètent cela comme un livre sonore « *Les bébés animaux* ». Nous, nous y voyons l'occasion de nous demander comment remplacer les petits boutons qui déclenchent les « coin-coin » et autres « miaou-miaou » par quelque chose de plus amusant, comme une carte Arduino et une télécommande par exemple.

La lecture de l'ouvrage s'avérant relativement rapide et assez peu intéressante (« Piou ! Piou ! Piou ! Le caneton ») nous pouvons porter toute notre attention sur le boitier à sa droite.

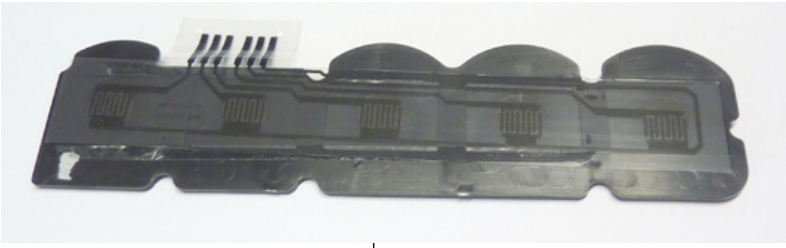
Comme on peut s'y attendre, le boitier plastique est simplement collé sur la couverture cartonnée et derrière se trouve une belle collection de vis qu'il suffira de retirer pour accéder à la partie qui nous intéresse. C'est une chance, très souvent ce genre de produit est entièrement collé, parfois même sans possibilité de remplacer les piles (qui coûtent souvent plus chez que le produit).

La partie électronique est simplement collée au livre. La vis en façade n'est là que pour les piles, les autres sont à l'arrière.

Le module est alimenté par trois piles LR44 reliées en série afin de fournir 4,5 volts. On serait tenté de se dire que 4,5V n'est pas très éloigné de 5V et que, peut-être, la puce est conçue pour accepter des tensions dans une plage comme 3,3V à 6V relativement courante. Il serait dommage toutefois de risquer la destruction de la puce dont nous n'aurons jamais la documentation. Le temps nous manque, nous avons une bonne blague à faire...

Nous nous en tiendrons donc à l'alimentation par les piles prévues à cet effet. En observant le circuit et surtout le système de détection

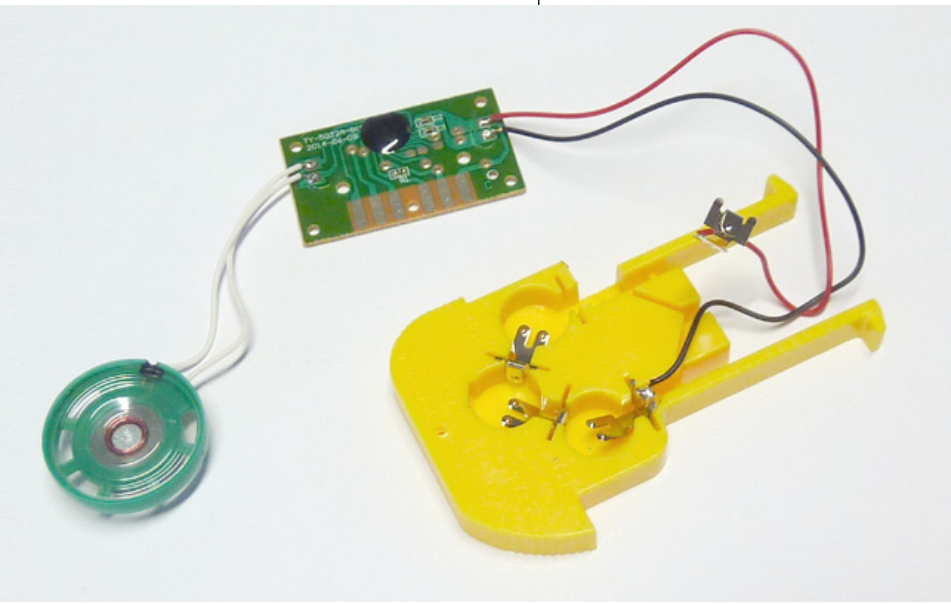
Le bidule est alimenté par trois piles boutons type LR44 à 1,5V connectées en série. L'électronique est donc alimenté en 4,5V.



Il n'y pas pas de boutons physiques coûteux mais deux membranes plastiques couvertes d'un revêtement conducteur, en appuyant on met une ligne à +4,5V

de pression sur les images, nous remarquons que l'activation des sons est provoquée par la mise à VCC (la tension d'alimentation) de cinq broches du module principal. Le choix du système de déclenchement de la part du constructeur n'est pas innocent mais purement économique. On retrouve ce type de membrane plastique dans énormément de produits. Le principe est simple, deux films plastiques sont peints avec une encre conductive faisant office de pistes. En appliquant une pression, les deux parties du circuit sont en contact et le courant passe. On retrouve ce type de choses un peu partout, des claviers aux calculatrices en passant par la plupart des petits appareils électroniques. Il ne s'agit pas nécessairement d'une technique bas de gamme, tout dépend de la qualité du matériel mais dans la plupart des cas, ceci coûtera moins cher que des interrupteurs ou boutons poussoirs mécaniques.

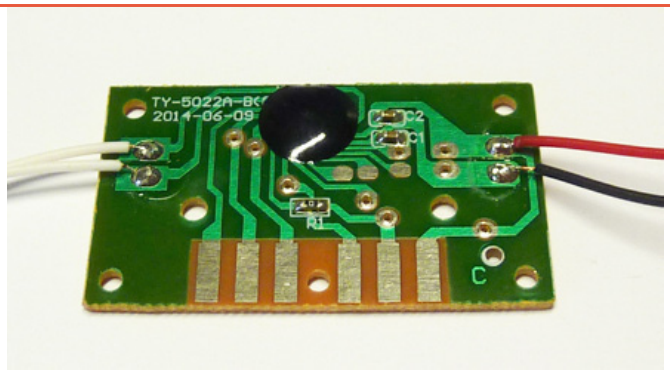
Comme on peut s'en douter, le contenu du boitier est très simple et se résume à un petit circuit imprimé et quelques fils. Heureusement, rien n'est collé ou thermocollé et nous avons facilement accès à tout ce qui nous intéresse.



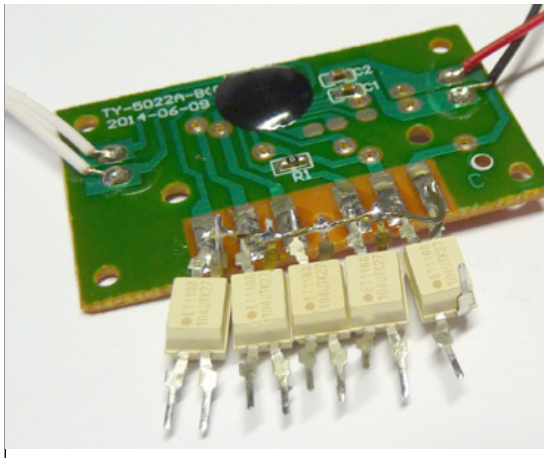
Comme nous décidons de ne pas alimenter le module en 5V, ceci signifie alors que nous ne pouvons pas non plus connecter directement les sorties de l'Arduino sur les broches. Nous ne pouvons savoir si le fait d'appliquer un niveau logique de +5V peut ou non endommager la puce sonore. Généralement les niveaux logiques hauts ne dépassent pas VCC.

L'idée est donc de totalement isoler électriquement le module de l'Arduino. Pour ce faire, il existe un composant dédié : l'optocoupleur (ou photocoupleur). Derrière ce joli nom se cache un duo led + phototransistor. Ce dernier fonctionne comme un transistor à la différence que la base, qui contrôle le passage du courant entre le collecteur et l'émetteur, ne reçoit pas du courant mais de la lumière ou de l'infrarouge. Les deux éléments sont réunis dans un boitier mais peuvent également être groupés en plusieurs exemplaires (2, 4, 8, voir plus). Il n'y a pas de contact direct entre l'un et l'autre coté, on parle donc d'isolation galvanique, ce qui est adapté pour beaucoup d'applications où l'on ne souhaite pas avoir une masse commune. Ceci peut être souhaitable pour le pilotage de hautes tensions mais également pour éviter les parasites pouvant se propager d'un circuit à un autre. Ainsi dans le cas des instruments de musique électroniques, à interfaces MIDI par exemple, l'optocoupleur est indispensable.

En fouillant un peu dans le fatras qui sert de stock de composants



L'ensemble du circuit est assez basique et classique : l'alimentation, le haut-parleur, les contacts pour les touches, une résistance de 0 ohms (c'est juste un pont), deux condensateurs et un circuit logique noyé dans la résine



Étant donné qu'il est totalement illusoire d'espérer trouver une documentation pour la puce, mieux vaut ne pas tenter le diable et laisser son autonomie électrique à l'ensemble : on se tourne donc vers l'option « optocoupleurs »

accumulés au fil des années, on finit rapidement par mettre la main sur quelques optocoupleurs TCET1108. Un tel composant n'est pas juste un interrupteur, il s'agit d'un phototransistor, ce qui implique un fonctionnement particulier similaire à celui d'un transistor. Ce n'est ni le lieu ni le moment de s'occuper de la théorie des transistors pour le moment (ceci fera l'objet d'un article que certains trouveront sans doute pénible, dans l'un des prochains numéros). Pour l'instant la seule chose qu'il faut savoir c'est qu'un transistor est piloté avec du courant contrairement au MOSFET qui est piloté par une tension. Un phototransistor en commutation (tout ou rien) est conducteur si le courant qui circule au travers de la led intégrée est supérieur au courant qui traverse le circuit en sortie, divisé par une valeur magique : le CTR (*Current Transfer Ratio* ou Rapport de Transfert en Courant (RTC) en français).

Dans le cas du composant que nous avons trouvé, nous avons un CTR de 130% à 260%. Pour commuter (saturer) le phototransistor il nous suffit d'y faire passer un courant supérieur à 260% du courant circulant « de l'autre côté ». Or, cela nous pouvons le mesurer facilement avec notre multimètre. Il suffit de mesurer le courant qui passe entre l'alimentation et une des broches en entrée. Résultat : 0,004 mA soit 4 µA. Cela ne va pas être très difficile à dépasser, tout en restant dans la valeur maximum utilisable d'après la documentation du TCET1108 (60 mA).

Les sorties de l'Arduino étant en +5V et la documentation nous indiquant que la tension qui apparaît aux bornes de la led (Vf) est de 1,25V (1,6V max), on peut faire un simple calcul de la loi d'Ohm :

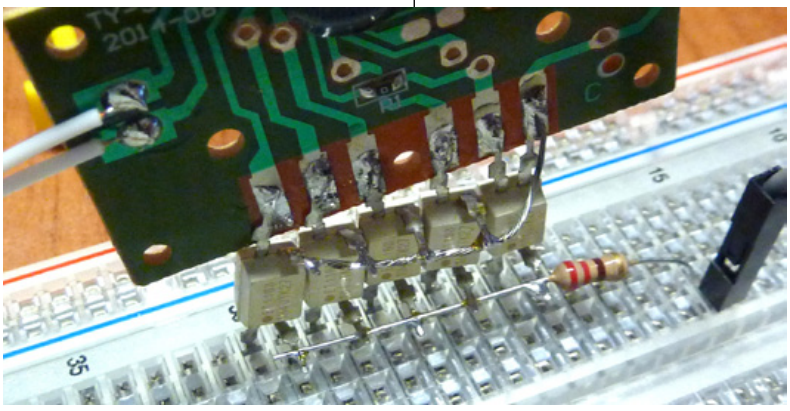
```
U = R.I
U = 5 - 1.25 = 3.75
3.75 = R * 0.06
R = 3.75 / 0.06 = 62,5 ohms
```

Mais nous n'avons pas besoin d'arriver à 60 mA puisqu'il nous suffit d'arriver 260% de 4 µA. Voyons ce que ça donne avec une simple résistance de 220 ohms :

```
U = R.I
U = 5 - 1.25 = 3.75
3.75 = 220 * I
3.75/220 = I = 0.017 = 17 mA
```

Magnifique ! Une résistance identique à celle qu'on utilise pour une led classique fera largement l'affaire. Ça marchera même avec 330 ohm ou 470.

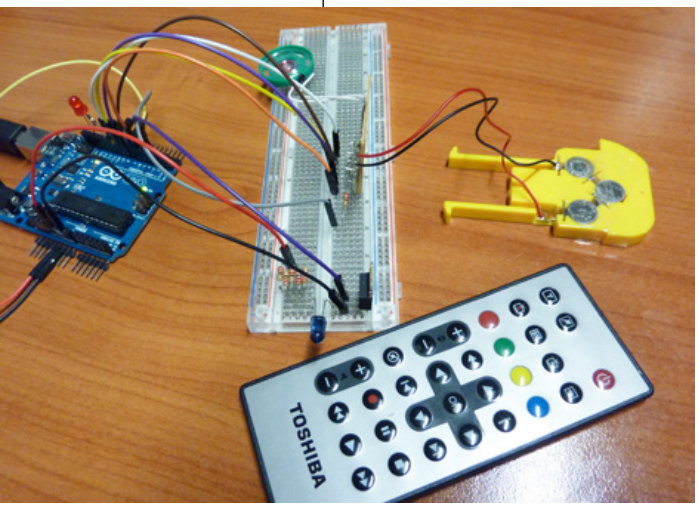
L'ajout soigné des optocoupleurs a été totalement prévu, calculé et optimisé pour une insertion sur une platine à essais... ou c'est juste un coup de chance



Nous pouvons maintenant nous amuser à souder directement les optocoupleurs, côté phototransistor, sur le module sonore puis relier toutes les broches allant à la masse (de l'Arduino) avec une unique résistance de 220 ohm.

Nous obtenons donc cinq lignes que nous pouvons directement relier aux sorties de l'Arduino pour simuler des pressions sur ce qui était initialement de petites images d'animaux.

Comme on est un peu pressé, l'assemblage et la soudure rapide sont assez peu soignés mais l'important c'est de rapidement générer des bruits stupides, non de produire un magnifique montage durable. Et puis, à y regarder de plus près, et la mauvaise foi aidant, nos soudures ne sont pas plus laides que les gros pâtés fait à l'usine pour les connecteurs du haut-parleur et de l'alimentation...



Le montage dans son ensemble : on a simplement ajouté le module à l'Arduino d'expérimentation en reliant la masse et 5 sorties pour contrôler les optocoupleurs

1. LE CROQUIS VITE FAIT

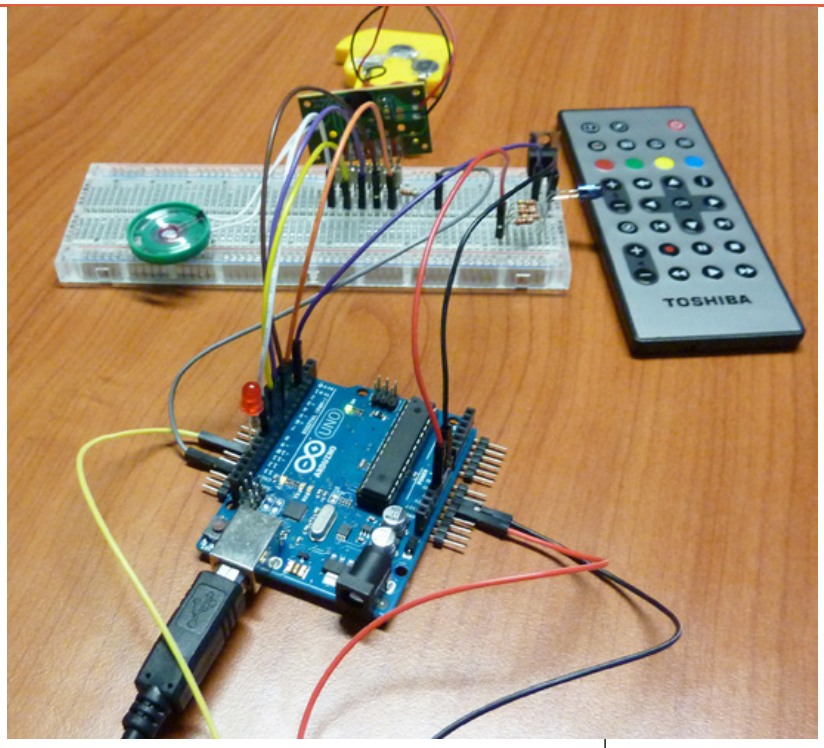
```
#include <IRremote.h>

int RECV_PIN = 11;

int chien = 10;
int chat = 9;
int canard = 8;
int cheval = 7;
int chevre = 6;

IRrecv irrecv(RECV_PIN);
decode_results results;

void setup() {
  pinMode(chien, OUTPUT);
  pinMode(chat, OUTPUT);
  pinMode(canard, OUTPUT);
  pinMode(cheval, OUTPUT);
  pinMode(chevre, OUTPUT);
  irrecv.enableIRIn();
}

void loop() {
  if (irrecv.decode(&results)) {
    if (results.decode_type == RC6) {
      switch(results.value) {
        case 0x800F045B:
        case 0x800F845B:
          digitalWrite(chien, HIGH);
          break;
        case 0x800F045C:
        case 0x800F845C:
          digitalWrite(chat, HIGH);
          break;
        case 0x800F045D:
        case 0x800F845D:
          digitalWrite(canard, HIGH);
          break;
        case 0x800F045E:
        case 0x800F845E:
          digitalWrite(cheval, HIGH);
          break;
        case 0x800F0422:
        case 0x800F8422:
          digitalWrite(chevre, HIGH);
          break;
      }
      delay(10);
      digitalWrite(chien,  LOW);
      digitalWrite(chat,   LOW);
      digitalWrite(canard, LOW);
      digitalWrite(cheval, LOW);
      digitalWrite(chevre, LOW);
    }
    irrecv.resume();
  }
}
```

Il n'y a ici rien de nouveau par rapport aux explications données dans l'article précédent. Nous utilisons un `switch`/`case` en plaçant judicieusement les `break` de manière à traiter les deux codes RC-6 de chaque bouton. Notre impulsion sur les connecteurs du module ne durera qu'un peu plus de 10 ms, ce qui est plus ou moins équivalent à l'utilisation de la membrane initialement prévue à cet effet (estimation totalement empirique).

On programme l'Arduino et quelques pressions sur les boutons de la télécommande démontrent le bon fonctionnement des calculs, des soudures et du croquis. Nous voici prêt pour cacher l'ensemble dans un bureau ou une salle de réunion et de guetter le moment propice pour déclencher discrètement un « miaou » ou un « bééééé » ! Il sera même possible de rendre l'ensemble encore plus discret et mobile en alimentant l'Arduino via le connecteur 2.1 mm noir avec une pile 9V standard.

2. EXTENSIONS POSSIBLES

L'une des suites possibles de cet article est, éventuellement, de faire les choses un peu plus proprement en transformant le module du livre sonore en un véritable module, sur plaque pastillée par exemple et en choisissant les composants de manière un peu moins aléatoire. Il peut être intéressant également de s'intéresser de plus près à l'alimentation du module en cherchant à savoir si oui ou non nous pouvons l'alimenter en +5V. Ceci nécessite une alimentation de laboratoire afin de surveiller le courant consommé et un outil pour observer la température de la puce, les temps de réponse, etc. C'est une opération à risque. Nous pouvons également utiliser un régulateur LDO (*Low-DropOut*) comme un TPS71745 qui nous permettrait d'alimenter le module en 4,5V à partir des 5V fournis par l'Arduino.

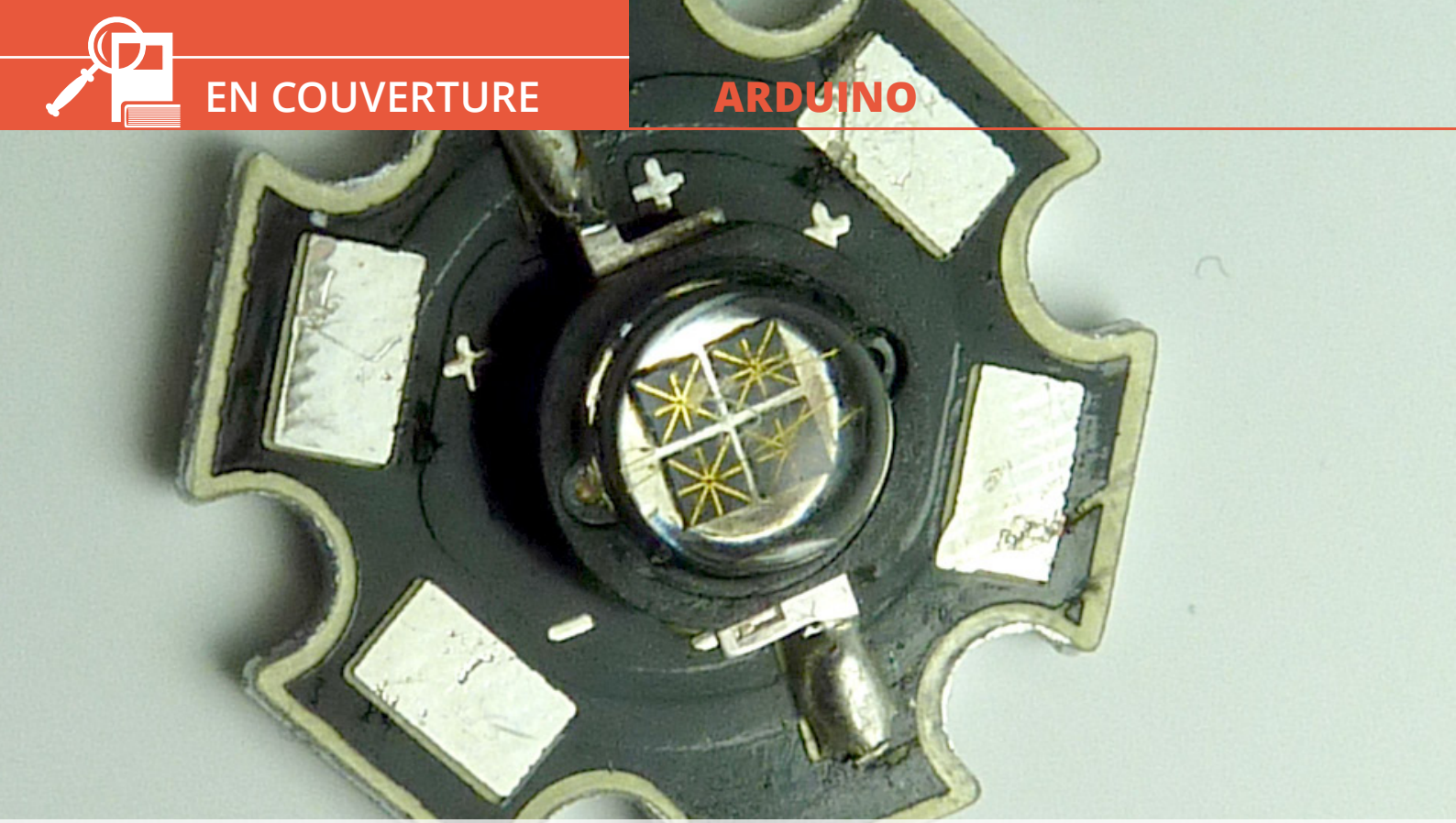


Voilà une joyeuse salade de câbles dans laquelle nous n'aurons jamais besoin de nous y retrouver. Au bout de quelques centaines de « coin-coin » et de « bééé-bééé » il est possible que les collègues ne trouvent plus cela très drôle et cherchent à réduire votre intégrité physique... L'Arduino pourra alors servir à autre chose

Mais à quoi bon... Le montage n'est pas assez générique pour justifier tant d'efforts et, dans l'état, sa durée de vie est très limitée, proportionnelle à la patience de vos collègues ou amis.

On peut cependant envisager quelques évolutions rapides comme l'ajout (via récupération ou construction) d'un amplificateur permettant de rendre les effusions sonores encore plus énervantes. On pourra également s'intéresser à d'autres systèmes de déclenchement comme l'utilisation de modules/shields Ethernet, Bluetooth, Wifi, RF ou encore en couplant le montage à un capteur de mouvement PIR acheté pour l'occasion ou récupéré sur une lumière d'appoint automatique.

Quoi qu'il en soit, l'objectif premier était ici de mettre l'accent sur quelques points dont la perception d'un produit non pas pour ce qu'il est mais pour ce qu'il peut devenir, l'intérêt d'avoir une approche créative dans ses expérimentations et l'importance de ne jamais rien jeter. Il y a toujours des choses à récupérer et à réutiliser, même si à un instant *t* elles ne semblent pas présenter le moindre intérêt... **DB**



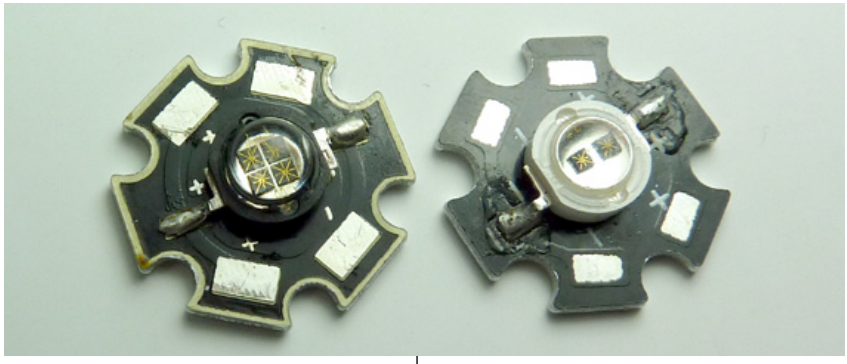
« LES SEULES BONNES TV QUE J'AI VU ÉTAIENT DES TV ÉTEINTES »

Denis Bodor

Cette phrase honteusement adaptée de celle généralement attribuée au lieutenant-colonel Custer (mais apparemment l'oeuvre du général Sheridan) résume assez bien la raison d'être de l'expérimentation qui va suivre. Il existe plusieurs raisons motivant l'extinction d'un téléviseur, qu'il s'agisse du sien ou de celui de quelqu'un d'autre. Mais cette motivation n'est pas la clé ici, seul l'objectif nous importera : éteindre toutes les TV dans un rayon le plus large possible.

Bien entendu, nous prenons ici le cas le plus démonstratif. Il sera parfaitement possible, avec un peu de patience et de ténacité, d'appliquer les mêmes principes pour couper le son, changer les réglages, zapper, etc. Le principe est relativement simple puisque, comme vous l'avez compris en lisant le premier article du dossier, les appareils Hifi comme les TV se pilotent via un signal infrarouge modulé sur une certaine fréquence et selon un certain protocole. L'idée est donc, tout simplement, d'envoyer les codes ou messages de mise hors tension de tous les téléviseurs connus, les uns à la suite des autres. L'effet est radical, dans la majorité des cas, quelque soit le modèle, l'appareil s'il est en fonction, s'éteint purement et simplement.

Le concept est celui inventé par *Mitch Altman* et concrétisé sous la forme d'un matériel vendu assemblé par sa société *Cornfield Electronics* : le *TV-B-Gone* (pour « TV be gone », en français « la TV s'en est allée »). Sa motivation initiale est résumée par sa citation présente en première page de son site : "*You can use TV-B-Gone to control access to television for philosophical or practical reasons, or simply to have fun !*" (« Vous pouvez utiliser le TV-B-Gone pour contrôler l'accès à la TV pour des raisons philosophiques ou pratiques, ou juste pour vous amuser »). Adafruit commercialise également un kit TV-B-Gone pour un peu moins de $20 basé sur un Atmel ATtiny85 mais c'est *Ken Shirriff* (oui, le même que celui qui a créé la bibliothèque que nous avons précédemment utilisée) qui a porté le code sur la plateforme Arduino.



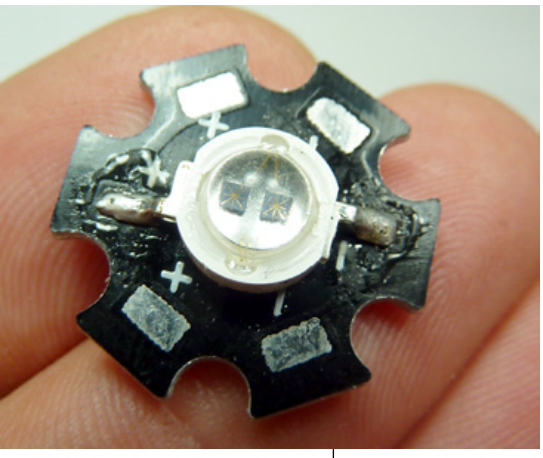
Les leds de puissance qu'il s'agisse de lumière visible, d'UV ou d'IR, sont facilement reconnaissables à leur support de fixation. A gauche une led IR 850 nm (éclairage pour caméra) et à droite 940 nm (éclairage pour équipement spéciaux mais aussi la bonne longueur d'ondes pour les télécommandes)

Grâce à lui, il est possible de transformer n'importe quelle carte Arduino en TV-B-Gone très facilement ou presque. Le kit Adafruit utilise quatre leds IR : deux IR333-A et deux IR333C/H0/L10. Les premières agissent sur une longueur d'ondes de 940 nm avec un angle de 20° et un courant de 100 mA. Les secondes également sur 940 nm et avec un courant identique ne se différencient que par un angle de diffusion plus large, de 40°. La combinaison des deux types permet d'atteindre une distance de contrôle théorique de quelques 45 mètres ! Est-ce suffisant ? Pas pour Hackable. Nous avons besoin de plus de puissance !

1. LE MATÉRIEL UTILISÉ

Le montage se base, bien entendu sur le croquis de Ken disponible sur son site (et GitHub) et la pièce centrale sera donc une carte Arduino Uno. Mais l'élément critique si l'on cherche à maximiser la portée, la diffusion et la puissance reste avant tout la ou les leds utilisées. Nous voulons surtout un maximum de puissance et, de préférence un angle de diffusion le plus large possible. Ce dernier point est dépendant de l'objectif de la réalisation et deux voies

La led sur 850 nm n'est pas adaptée pour l'émulation de télécommande mais elle fera office très efficacement de système d'éclairage pour les caméras sensibles aux IR. Notez les 4 « chips » formant la led et les connexions internes de toute beauté. Il ne nous est pas possible de vous le montrer en raison de la sensibilité aux IR des APN, mais ce type de leds émet une faible lueur rouge. Ceci provient du fait qu'une led est totalement incapable d'émettre sur une longueur d'ondes unique. Une partie du rayonnement émis déborde sur les longueurs d'ondes voisines. Et proche du 850 nm nous avons 780 nm, le « rouge extrême » perceptible pas nos yeux.



Notre led de puissance de 1 watts utilisée pour notre montage. On distingue clairement les deux « chips » dans le composant ainsi que les résidus de flux provenant de la soudure sur le support mais on pardonne cela à 5€/pièce. D'autres points de soudure sont à notre disposition pour connecter la led à son alimentation.

sont possibles. Soit nous voulons atteindre une portée maximum et devons focaliser l'émission sous forme de rayon avec un angle de diffusion le plus étroit possible, soit nous souhaitons obtenir une télécommande universelle quasi-omnidirectionnelle nous évitant de pointer notre périphérique vers l'appareil à contrôler.

Dans le premier cas, on pourrait s'intéresser à des projecteurs infrarouge ou des lasers en prenant bien soin de s'assurer de la longueur d'ondes utilisée car un certain nombre de ces équipements sont destinés à l'éclairage infrarouge pour des caméras de vision nocturne (dans les 850 nm). Dans le second, il existe une certaine gamme de leds dites « de puissance » offrant un angle de diffusion de l'ordre de 140°. Qu'elles soient destinées à une utilisation pour de l'éclairage (lumière blanche), de la signalisation (couleurs) ou à agir dans le domaine des UV ou de l'infrarouge, les leds de puissance sont facilement reconnaissables.

Elles se présentent généralement montées sur un circuit miniature de forme hexagonale avec une base en aluminium ayant plusieurs usages : fixation, connexion et surtout dissipation thermique. En effet, si nous prenons en exemple les leds que nous avons achetées auprès d'un vendeur allemand sur Ebay (KT-elektronic), celles-ci doivent être alimentées avec un courant de 700 mA (800 mA maximum) et présentent une tension à leurs bornes de 1,6 V. Un rapide calcul nous indique une puissance consommée de 0,7 A fois 1,6 V, soit 1,12 watts.

Mais ne vous y trompez pas, cette puissance n'est pas celle de l'émission IR de la led. Avec une led blanche ou de couleur, nous disposons d'une unité de mesure de la puissance effective : le lumen. Celle-ci permet de connaître la puissance lumineuse ou l'intensité du flux lumineux. Un lumen (*lm*) est l'intensité lumineuse de 1 candela (*cd*) émise dans un angle solide de 1 stéradian (*sr*). Le stéradian est grossièrement à la 3D ce que le radian est à la 2D. Une unité plus connue du grand public pour l'éclairement lumineux est le lux (*lx*) qui vaut (1 cd * 1 sr) / 1m². Pour asseoir ces notions, ajoutons que l'œil humain possède un champ de « vision précise » d'environ 1/2 stéradian (physiologiquement parlant la perception humaine est un peu plus complexe que ce simple nombre).

Malheureusement, tout cela ne concerne que la lumière visible et non les UV ou les infrarouges. Les candelas et les lumens sont ajustés en fonction de la sensibilité de l'œil humain. Ainsi à la question « *quelle est l'intensité lumineuse d'une led infrarouge* », la seule réponse possible est 0 mcd (zéro milli-candela). Ceci signifie aussi que si vous tombez sur un vendeur qui vous annonce un produit type « *led IR de 2000 mcd* », passez simplement votre chemin, il ne sait clairement pas de quoi il parle.

Ainsi la plupart des documentations techniques indique un nombre de milliwatts par stéradian (*mW/sr*). Avec des leds IR de signalisation classique, le flux énergétique est de l'ordre de 30 à 70 mW/sr (à 100 mA). Celle de notre led de puissance serait apparemment de l'ordre de 300 mW/sr. Je dis « apparemment » car nous n'avons pas de véritable documentation (*datasheet*) et baserons cette estimation sur une autre documentation pour une led à un *chip* aux caractéristiques similaires mais multipliés par deux puisque notre composant possède deux *chips* reliés en parallèle.

Le point important dans tout cela est le fait que la puissance consommée par la led n'est pas intégralement convertie en émission infrarouge. En réalité, plus de la moitié de la puissance injectée dans la led est perdue en énergie thermique. Ainsi, notre led de puissance consommant joyeusement plus d'un watt va chauffer et va devoir dissiper 500 mw au minimum de chaleur. Voilà pourquoi ce type de led est soudé sur un support spécifique et qu'il est impératif

de la placer sur un radiateur afin d'éviter tout problème. Bien entendu, dans le cas d'un signal ponctuel comme l'envoi d'un code RC-6, ce n'est pas un gros problème mais en entamant une séquence d'envoi de centaines de codes, la montée en température sera bien réelle et nous devrons la prendre en compte.

Fort heureusement, les vendeurs distribuant des leds de puissance mettent généralement aussi en vente des systèmes de dissipation thermique, généralement sous la forme de gros radiateurs en aluminium sur lesquels il est possible de maintenir le support avec des vis.

Là, le jeu est le même que pour l'achat des leds avec souvent en bonus des traductions amusantes venues tout droit de Shenzhen dans les descriptifs :

- « *Complètement Refroidissement sans VENTILATEURS POUR LA lumière pendentif LED 24 heures 365 jours* »
- « *Monter verser la lumière de LED 10W* »

2. LA PROBLÉMATIQUE DE L'ALIMENTATION

Obtenir un courant de 700 mA sous 1,6 V n'est pas quelque chose d'évident. Vous pouvez bien sûr construire votre alimentation de A à Z. C'est très pédagogique et vous acquerrez de cette manière une bonne expérience en électronique analogique. Malheureusement ceci vous prendra pas mal de temps, chose que vous n'avez pas nécessairement à votre disposition.

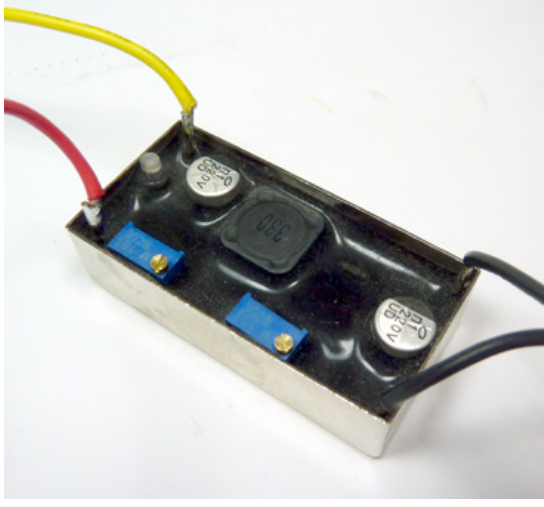
Fort heureusement, les leds de puissance étant de plus en plus populaires pour les systèmes d'éclairage ou de signalisation, on trouve aujourd'hui des alimentations dédiés. Ces circuits spécialisés sont généralement vendus en tant que tel mais également sous d'autres désignations comme « convertisseur DC/DC », « *buck power supply* » ou encore *DC led drivers*.

Ces alimentations n'utilisent pas de convertisseurs linéaires comme le très connu LM7805 ou le MC33269 équipant certaines cartes Arduino mais des convertisseurs Buck utilisant le mécanisme d'alimentation à découpage. Le principal avantage des alimentations à découpage est d'offrir une grande efficacité. En effet, alors qu'un régulateur linéaire se débarrasse du trop plein d'énergie sous forme de chaleur, l'alimentation à découpage opère d'une façon très différente. Le courant est haché en petits morceaux par un mécanisme de PWM. Le rapport cyclique permet de ne prendre qu'une partie du courant qui est ensuite passé dans un filtre pour obtenir une tension moyenne. Des circuits dédiés, comme le LM2596 de Ti, intègrent toute cette mécanique en intégrant en plus une surveillance constante du courant et de la tension en sortie afin d'éviter les problèmes (chute de tension), garantir la sécurité (surchauffe) et limiter au maximum le bruit en sortie.

On trouve sur les sites d'enchères en ligne ce type de produit pour quelques 5 euros. Généralement construit autour d'un LM2596 ils sont vendus à la fois comme alimentation pour les leds de puissant mais également comme circuit de charge d'accu lithium-ion. Les caractéristiques importantes à prendre en compte lors de l'achat sont les suivantes :

- Tension acceptée en entrée (7V-35V),
- tension réglable en sortie (1,25V-30V),
- courant réglable en sortie (0A-3A).

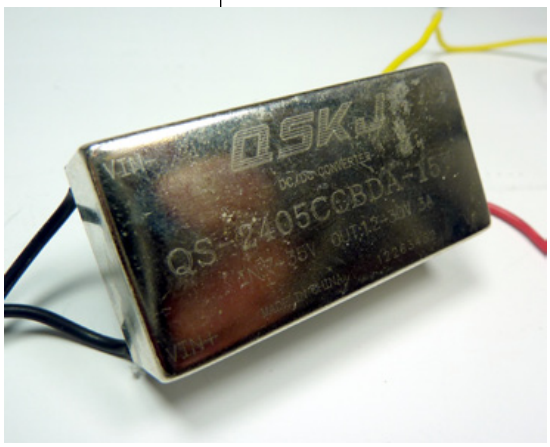
Attention lors de votre achat. Selon la provenance du matériel, il peut



L'un des régulateurs que nous avons acquis pour l'occasion se présente sous la forme d'un circuit placé dans une boite métallique noyée de résine. L'agencement des composants cependant laisse clairement penser que l'architecture sous-jacente doit être très proche, sinon identique, aux autres circuits nus qu'on trouve un peu partout sur le web. A droite nous avons l'entrée et à gauche la sortie. Deux potentiomètres multi-tour permettent de régler la tension et le courant en sortie.

y avoir tromperie sur la marchandise. En effet, comme l'a montré *Julian Ilett* sur sa chaîne Youtube, le LM2596 existe en plusieurs versions dont une, LM2596HV, est susceptible d'accepter jusqu'à 60 volts en entrée. Certains fabricants cependant n'hésitent pas à utiliser des LM2596S sérigraphiés LM2596HV afin de vendre leur matériel plus cher. Bien entendu le composant ne peut pas fonctionner avec une tension de 60V et est parfaitement incapable de réguler le courant et la tension en sortie de manière acceptable. Nous ne sommes pas réellement impacté par ce type de fraudes car, dans la plupart des cas un simple bloc secteur de 9V ou 12V sera utilisé pour obtenir nos 700 mA à 1,6 volts (voir une pile 9V). Il sera cependant judicieux avant achat de consulter les avis et notes attribuées aux vendeurs et de faire une petite recherche sur le web pour s'assurer que celui-ci ne se soit pas fait « épinglé » sur un forum ou un autre...

Ne vous faites cependant pas d'illusions. Les valeurs maximums pour ce type de produits bas de gamme ne sont PAS A PRENDRE AU PIED DE LA LETTRE ! Plusieurs utilisateurs ont sacrifié leurs exemplaires pour démontrer, vidéos à l'appui, qu'en essayant de pousser le module vers ses limites, cela fini généralement mal. Une sortie en 30V 2,5A pendant moins d'une minute, par exemple, peut faire gripper le régulateur à près de 70°C qui, sans dissipation, conduit à sa destruction pur et simple. D'autres vidéos montrent un fonctionnement correcte en restant loin des valeurs maximales avec tantôt des comportements étranges, comme la sortie à plus de 5A et une montée rapide en température en ajustant le potentiomètre de réglage de la tension au minimum. Avec une sortie réglée juste un peu au dessus (~0,05V), aucun problème...



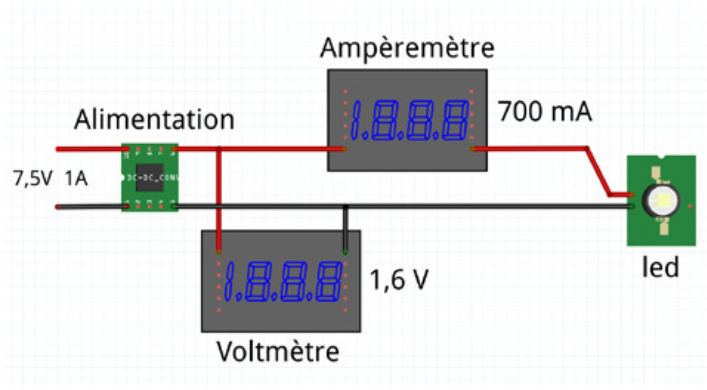
La coque de l'alimentation a un double usage. Elle permet de protéger physiquement l'électronique et assure la dissipation thermique ce qui permet, peut-être, d'éviter les problèmes de destruction du régulateur qui se trouve sur le circuit interne.

Il est intéressant de remarquer que ce type de produits n'est pas vendu que sur Ebay mais un peu partout dans des boutiques en ligne. Instinctivement, on aurait tendance à se méfier de la vente direct via les sites d'enchères mais en vérité le système d'évaluation et de commentaires joue généralement très bien son rôle alors qu'il n'y a aucune équivalence pour une boutique web ne présentant souvent que les avis positifs, parfois ne provenant même pas de réels acheteurs.

Sans vouloir faire la promotion d'un système de paiement aux commissions pénibles côté vendeur, il faut avouer que l'utilisation de Paypal, côté client, est également rassurante. En effet, pour en avoir fait les frais personnellement lors de la vente d'un de mes ordinateurs d'occasion il y a quelques années, le client à tout loisir de provoquer, en ouvrant un « litige », de gros problèmes pour le vendeur, même pour 24h de retard dans l'expédition (suspension de compte vendeur, mise en veille du compte Paypal et déclenchement de toute une procédure de sortie de litige riche en paperasse en tout genre). Les vendeurs une fois mis au pied du mur deviennent donc très conciliant s'ils ont quelque chose à se reprocher.

3. PROCÉDURE DE RÉGLAGE DE L'ALIMENTATION

Pour rendre les choses les plus simples et les moins pénibles possibles, il est préférable ici d'utiliser deux multimètres. Le premier permettra de régler la tension en sortie aux bornes de la led et le second mesurera le courant traversant le circuit. L'utilisation de deux appareils de mesure n'est pas une obligation mais ceci vous évitera de brancher/débrancher sans cesse l'ensemble pour procéder aux réglages de l'une et l'autre valeur.



Notre convertisseur de tension est équipé d'une led bicolore vert/rouge. Bien entendu, la documentation du module se limite à sa simple description sur une page Ebay. Ce n'est qu'en expérimentant qu'on se rend compte de l'effet de chaque potentiomètre et de l'indication donnée par la led. Lorsque celle-ci est verte, le convertisseur est en limitation de tension et le courant qui circule dans le circuit en sortie est inférieur à la limite imposée par le réglage. Lorsque la led est rouge, le module est passé en limitation de courant. Ceci signifie que, potentiellement, davantage de courant pourrait circuler mais que celui-ci est limité par le régulateur du circuit.

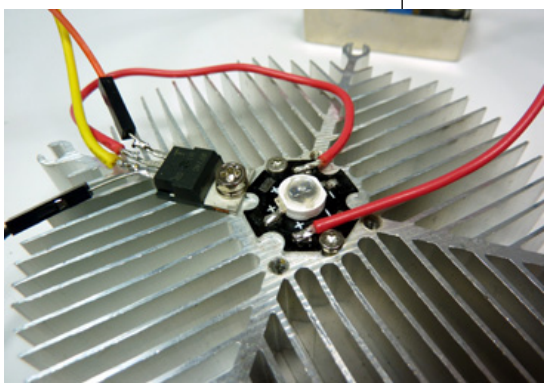
Notre alimentation utilise une seule led et deux potentiomètres mais d'autres modèles comportent deux leds et trois potentiomètres. Ce potentiomètre supplémentaire équipe généralement les modules d'alimentation pouvant être utilisés comme chargeur d'accu lithium-ion. Sommairement, pour charger ce type d'accumulateurs on utilise une tension constante et un courant limité. La phase de charge commence avec la stabilisation de la tension et une baisse progressive du courant qui circule dans l'accumulateur. En dessous d'un certain seuil de courant, l'accumulateur est considéré comme chargé et on peut alors appliquer une charge d'entretien (*floating voltage*) destinée à maintenir la charge afin que l'accumulateur ne s'auto-décharge pas. Le troisième potentiomètre permet de régler la tension utilisée pour cette charge d'entretien, à priori, en fraction de la tension de charge.

La procédure même de réglage consiste simplement à ajuster les potentiomètres au niveau minimum (sens anti-horaire) puis de brancher les multimètres :

- Celui en ampèremètre est en série du circuit de sortie,
- celui en voltmètre est en parallèle.

Ne branchez pas la led. Il est recommandé pour la première mise sous tension de laisser le circuit ouvert et de s'assurer que la tension est bien proche de zéro. Il existe des équipements permettant de simuler une charge dans un circuit. Il peut s'agir d'un ensemble de résistances configurées de manière à limiter le passage du courant à une valeur déterminée, ou d'un équipement plus complexe permettant de régler une valeur en ampères arbitrairement choisie. En l'absence de ce type de matériel ou, simplement, de grosse résistance pouvant dissiper plusieurs watts, nous ne pouvons que faire confiance au fabricant de l'alimentation en connectant la led si la tension mesurée est effectivement acceptablement basse.

On réglera ensuite les potentiomètres d'ajustement de tension et de courant de manière à obtenir les valeurs adaptées à la led. Vous remarquerez peut-être que la tension ne sera pas exactement celle indiquée par la documentation ou la description de la led. Il existe des marges de tolérance et une probabilité d'erreur dans les informations fournies. N'oubliez pas le principe de fonctionnement d'une led, qui est pilotée en courant et non en tension. Ceci signifie qu'en fournissant le courant adéquate à une led, une tension apparaît à ses bornes, et non l'inverse. L'élément important est le courant qui circule et il est parfaitement normal que le module d'alimentation passe donc en limitation de courant (led en rouge avec notre exemplaire).



Le radiateur est sans doute un peu surdimensionné pour les 500 mW que nous devrons dissiper mais il s'adapte mécaniquement parfaitement au support. Nous l'utilisons également pour fixer le MOSFET piloté par l'Arduino et contrôlant le fonctionnement de la led.

Une fois l'alimentation parfaitement réglée, vous pouvez débrancher la led et assembler le circuit pour placer le MOSFET comme nous l'avons fait avec la led TSAL6200 dans le premier article. Étant donné la puissance en présence (1,6V x 0,7A > 1W) il est fortement recommandé de fixer un radiateur au MOSFET. Dans notre cas, le dissipateur thermique utilisé pour la led étant quelque peu surdimensionné, nous avons décidé de faire d'une pierre deux coups en le fixant tout simplement au même support.

4. MONTAGE

Le montage en lui-même est strictement identique à celui utilisé dans le premier article pour notre première émission de code RC-6. Ceci est vrai jusqu'à la broche utilisée en sortie de l'Arduino pour contrôler le MOSFET IRF520. La seule différence provient donc de la led et de son alimentation. Nous ne prenons plus les +5V directement sur l'Arduino mais utilisons la tension fournie par l'alimentation dédiée à led. Celle-ci passe par la led puis le drain du MOSFET et retourne par la source à la broche négative de l'alimentation. La grille du MOSFET est alors reliée à la broche 3 de l'Arduino. Nous n'oublions pas, enfin, de connecter la source du MOSFET à la masse de l'Arduino. En effet le fonctionnement même du MOSFET repose sur une comparaison de tensions. Sans cette connexion il n'y aurait pas de différence de potentiel entre la grille et la source.

On ajoutera entre la broche 13 de l'Arduino et la masse une led et sa résistance de 470 ohms (ou 220 ou 330) permettant de servir de témoin d'activité. Un bouton poussoir quelconque sera également placé entre la masse et la broche 2. Son utilisation déclenchera l'envoi des codes IR. Et enfin, la broche 5 permet, en laissant le croquis de Ken dans l'état, de choisir entre l'envoi de codes pour l'Amérique du nord et l'Asie (non connecté) et l'Europe, le moyen orient, l'Australie et la nouvelle Zélande (connexion à la masse via une résistance de 10K ohms).

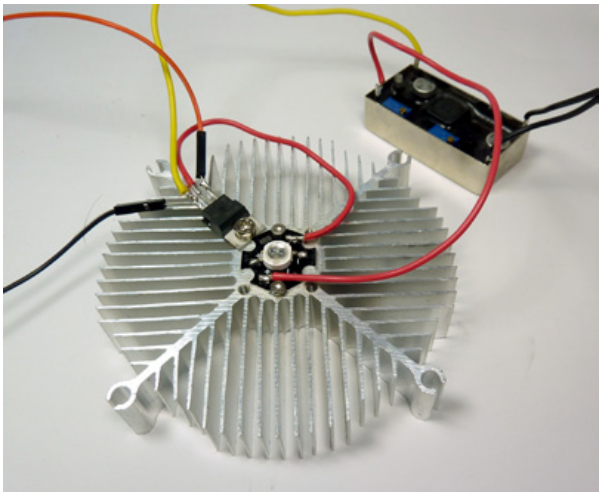
5. UTILISATION DU CROQUIS TV-B-GONE

(ATTENTION : ce qui va suivre est une analyse d'un bug dans le code et cette partie est relativement technique. Si cela ne vous intéresse pas, sautez à la section suivante qui explique simplement les modifications à apporter sans plus d'explications).

Sur le papier les choses sont très simples au niveau du croquis. Dans la réalité, la grande majorité des utilisateurs n'arrivent pas à faire fonctionner l'ensemble. Pourquoi ? Hé bien tout simplement parce que le code n'a pas été mis à jour depuis 4 ans et qu'entre temps, il y a eu un changement majeure de version du côté de l'environnement Arduino.

L'arrivé de la version 1.0 de l'environnement en novembre 2012 ainsi que la succession de versions jusqu'à la 1.0.6 de septembre dernier ont impliqué des changements à la fois dans la manière dont les croquis sont traités mais aussi compilés. Les versions récentes du compilateur AVR GCC manipulent les données de manière un peu plus strictes et en particulier celles devant trouver leur place non pas dans la mémoire vive (SRAM) mais en flash.

C'est une astuce de programmation qui n'a rien d'exceptionnelle. Elle consiste à placer dans



L'ensemble du montage est relativement clair. L'alimentation est connectée à l'anode de la led, et la cathode est branchée sur le drain du MOSFET. La source de ce dernier, fil jaune, se connecte à la borne négative de l'alimentation ainsi qu'à la masse de l'Arduino. Enfin, la grille, fil orange, est reliée à la broche 3 de l'Arduino

NOUVEAU ! NOUVEAU ! NOUVEAU ! NOUVEAU !

ABONNEZ-VOUS!

DÉCOUVREZ NOS NOUVELLES OFFRES !

LE CLASSIQUE !

VERSION PAPIER

Retrouvez tous les 2 mois votre bimestriel favori en papier dans votre boîte à lettres !

Envie de lire tous les 2 mois votre magazine sur votre tablette ou votre ordinateur ? Voici l'abonnement PDF !

Attention, les offres mentionnées dans la page suivante sont réservées aux particuliers, professionnels n'hésitez pas à nous contacter à : abopro@ed-diamond.com

L'ÉLECTRONIQUE !

VERSION PDF

boutique.ed-diamond.com

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

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

Découvrez une sélection d'offres page suivante

FAITES DES ÉCONOMIES !

Venez découvrir toutes nos offres de couplage avec :

LINUX MAGAZINE / FRANCE

GNU LINUX PRATIQUE ESSENTIEL — DÉCOUVRIR, COMPRENDRE ET UTILISER LINUX

MISC Multi-System & Internet Security Cookbook

Open Silicium

sur :

boutique.ed-diamond.com

Veuillez cocher ci-dessous l'offre souhaitée, et veuillez renvoyer l'ensemble du coupon à l'adresse indiquée page précédente avec votre règlement. Vous pouvez également directement vous abonner sur notre site internet !

HACKABLE MAGAZINE

Merci de cocher ci-dessous l'offre souhaitée...

	TARIF NORMAL	ÉCONOMIE	TARIF SPÉCIAL*
☐ PAPIER — HK 6N°	47,40	8,40	39,-€
☐ PDF** — HK 6N°	47,40	8,40	39,-€
☐ BASE DOCUMENTAIRE** — ACCÈS 1 AN			Disponible septembre 2015
☐ PAPIER HK 6N° + PDF** HK 6N°	94,80	35,80	59,-€

HACKABLE MAGAZINE + Open Silicium

Merci de cocher ci-dessous l'offre souhaitée...

	TARIF NORMAL	ÉCONOMIE	TARIF SPÉCIAL*
☐ PAPIER — HK 6N° + OSI 4N°	83,40	18,40	65,-€
☐ PDF** — HK 6N° + OSI 4N°	83,40	18,40	65,-€
☐ PAPIER HK 6N° + OSI 4N° + BASE DOCU OSI** ACCÈS 1 AN BD OSI	113,40	28,40	85,-€
☐ PAPIER HK 6N° + OSI 4N° + PDF** HK 6N° + OSI 4N°	166,80	72,80	94,-€
☐ PAPIER + PDF** + BASE DOCUMENTAIRE OSI** — OSI 4N° + HK 6N° + OSI 4N° + HK 6N° + ACCÈS 1 AN BASE DOCU OSI	196,80	77,80	119,-€

*Prix TTC France Métro. Pour les tarifs hors France Métro et étranger, veuillez consulter notre boutique en ligne !

**Offre réservée aux particuliers, professionnels contactez-nous par mail : abopro@ed-diamond.com

la mémoire flash, normalement utilisée pour le programme, les variables qui ne changent pas (leur contenu est constant). Dans le cas présent, le croquis de Ken intègre une quantité phénoménale de codes IR permettant d'éteindre presque tous les téléviseurs, toutes marques et tous modèles confondus. Ces données sont utilisées par le programme mais, bien entendu, ne changent pas au cours de l'exécution de ce dernier. Il n'y a donc pas de raison de consommer la précieuse mémoire vive inutilement. D'ailleurs, les 2 Ko de SRAM disponibles avec un ATmega328 comme celui qui équipe une carte Arduino Uno ne seraient tout simplement pas suffisant pour cela.

Le mot-clé **PROGMEM** permet de demander au compilateur de placer une variable en mémoire flash et il doit être normalement utilisé avec des types de données spécifiques avec Arduino. On utilise ensuite, dans le cœur du programme, des fonctions spéciales permettant de retrouver les données et de les utiliser. Bien entendu, ceci prend tout son sens non pas pour stocker quelques valeurs numériques mais pour des masses importantes de données ou des chaînes de caractères importantes.

Le problème qui se pose avec **PROGMEM** est son lien étroit avec le compilateur et la syntaxe qu'il impose pour utiliser ce type de technique. L'abstraction dont on bénéficie généralement avec Arduino disparaît et on arrive de plein pied dans le domaine de la « vraie » programmation AVR, sans filet et sans indulgence de la part des outils de développement.

Il en découle des problèmes relatifs à la syntaxe à utiliser et aux mécanismes de compilation intégrés par l'environnement Arduino. En d'autres termes, comme c'est un point particulier, presque hors du champ d'Arduino, les programmeurs doivent maintenir leurs codes et croquis à jour afin de suivre les évolutions. Bien entendu, ce n'est souvent pas le cas et tout cela fini par ne plus fonctionner au fil du temps. Ainsi, en récupérant le croquis Arduino de Ken sur :

- **https://github.com/shirriff/Arduino-TV-B-Gone**
- ou sur **http://www.righto.com/2010/11/improved-arduino-tv-b-gone.html**

et en tentant d'ouvrir et vérifier le croquis dans l'environnement, on obtient une erreur :

```
WORLDcodes.cpp:8506:37: error: variable 'NApowerCodes'
must be const in order to be put into read-only section
by means of '__attribute__((progmem))'
```

Notez également que les fonctions particulières utilisées pour l'Arduino Uno provoquent un problème bien plus grave avec l'Arduino Leonardo. Nous avons constaté, en effet, que la version 1.0.5 de l'environnement, sur distribution Debian GNU/Linux posait problème même en corrigeant le code concernant l'utilisation de **PROGMEM**. Ceci n'a cependant pas été testé, faute de temps, avec un environnement 1.0.6 et/ou sous Windows ou Mac OS X.

Le message d'erreur à propos du fichier **WORLDcodes.cpp** concerne l'utilisation de **PROGMEM** (qui est en fait l'attribut **_ATTR_PROGMEM__**). Le compilateur nous dit qu'en lignes 8506 et 8646 nous lui signifions vouloir placer **NApowerCodes** et **EUpowerCodes** en mémoire de programmation mais que pour que cela soit valable, il nous demande de déclarer ces variables avec **const**. Or, voilà le code incriminé :

```
const struct IrCode *NApowerCodes[] PROGMEM = {
  &code_na000Code,
  &code_na001Code,
```

Il y a donc bien un **const** mais cela se complique un peu car en réalité, nous avons un tableau de pointeurs qui contient donc des adresses vers d'autres données également placées en mémoire flash. Et ce n'est pas tout, le croquis est coupé en morceaux. Nous avons ainsi :

- **TVB.pde** qui est le croquis du programme principal,
- **WORLDcodes.cpp** qui contient les codes pour tous les téléviseurs,
- **main.h** qui est un fichier d'entête permettant de définir quelques paramètres et faisant le lien entre les autres éléments.

Dans **TVB.pde** on trouve ainsi une autre déclaration :

```
extern PGM_P *NApowerCodes[] PROGMEM;
extern PGM_P *EUpowerCodes[] PROGMEM;
```

Et c'est précisément de cela qu'il s'agit, une *déclaration*. Dans `WORLDcodes.cpp` nous avons une *déclaration* **ET** une *définition* des deux variables mais comme elles sont utilisées dans `TVB.pde`, elles doivent également y être *déclarées* (mais pas définies puisqu'elles le sont déjà par ailleurs). Bien entendu, il nous faudra aussi corriger le problème de l'utilisation de `const` ici.

La correction est simple à partir du moment où l'on comprend ce que veut dire ceci :

```
const struct IrCode *NApowerCodes[] PROGMEM = {
```

qui se traduit en français par : « *je déclare un tableau qui ne change pas et qui contient des pointeurs vers des données sous forme de structures de type* `IrCode`*, le tout doit trouver place dans la flash, svp* ». Voyez-vous pourquoi le compilateur n'est pas content ? Il manque effectivement un `const` car nous n'avons pas dit que les pointeurs stockés ne changent pas non plus. La version corrigée est :

```
const struct IrCode * const NApowerCodes[] PROGMEM = {
```

Voilà ! Maintenant, tout est `const` (constant) et le compilateur est heureux... sur ce point là. Maintenant, il râle sur `TVB.pde` :

```
TVB.pde:78:30: error: variable 'NApowerCodes'
must be const in order to be put into read-only
section by means of '__attribute__((progmem))'
```

Même pas peur, c'est le même problème et donc la même correction et on change :

```
extern PGM_P *NApowerCodes[] PROGMEM;
```

en

```
extern PGM_P * const NApowerCodes[] PROGMEM;
```

Et là :

```
TVB.pde:381: référence indéfinie vers « NApowerCodes »
TVB.pde:381: référence indéfinie vers « EUpowerCodes »
```

Mais pourquoi ? Et puis d'abord, c'est quoi ce `PGM_P` ? Et où est l'autre `const` ?

Du calme. Pour comprendre il faut fouiller dans la documentation de l'AVR Libc, l'un des éléments sur lesquels est construit tout l'environnement Arduino. Là, on apprend que `PGM_P` n'est qu'une macro pour `const char *` (comme `HIGH` est une macro pour `1`). Il est donc précisément là notre second `const`. De ce côté tout va bien.

Par contre, il y a deux petits problèmes :

- La déclaration de `NApowerCodes` et `EUpowerCodes` n'a rien à faire dans `TVB.pde`. Sa place est dans `main.h` car les fichiers d'entête sont fait pour cela (en C++ on ne rigole pas avec ça).
- `NApowerCodes` et `EUpowerCodes` ne sont pas des tableaux de pointeurs vers des `char` mais vers des structures `IrCode`.

Il faut faire un peu de ménage car ce qui était toléré par le compilateur des anciennes versions de l'environnement Arduino ne l'est plus et c'est à nous programmeur de mettre tout cela en ordre...

6. AAAAAHHHH ! IL EST MALADE ! MAIS JE DOIS FAIRE QUOI ALORS ?!

Si vous avez sauté à cette section de l'article directement, vous venez de rater de croustillantes explications sur les problèmes soulevés par l'utilisation de la mémoire flash, les conséquences de la non mise à jour d'un projet et surtout, ce qu'il en coûte de jouer avec les fonctionnalités avancées des briques qui composent Arduino.

Vous l'avez compris, le code original de Ken de TV-B-Gone pour Arduino ne fonctionne pas. Pour le corriger, récupérez-le sur **http://www.righto.com/2010/11/improved-arduino-tv-b-gone.html** sous la forme d'une archive `arduino-tv-b-gone-1.2.zip`. Décompressez-là dans votre répertoire à croquis (*sketchbook*) puis lancez l'environnement Arduino et ouvrez le projet.

Trois onglets vous sont proposés : **TVB.pde**, **WORLDcodes.cpp** et **main.h**. Dans **WORLDcodes.cpp**, à la ligne 8506 changez :

```
const struct IrCode *NApowerCodes[] PROGMEM = {
```

en

```
const struct IrCode * const NApowerCodes[] PROGMEM = {
```

et, ligne 8646 :

```
const struct IrCode *EUpowerCodes[] PROGMEM = {
```

en

```
const struct IrCode * const EUpowerCodes[] PROGMEM = {
```

Dans **TVB.pde**, en début de fichier, trouvez et supprimez les lignes :

```
extern PGM_P *NApowerCodes[] PROGMEM;
extern PGM_P *EUpowerCodes[] PROGMEM;
extern uint8_t num_NAcodes, num_EUcodes;
```

Enfin, dans **main.h**, en fin de fichier, ajoutez les lignes :

```
extern const struct IrCode * const NApowerCodes[] PROGMEM;
extern const struct IrCode * const EUpowerCodes[] PROGMEM;
extern uint8_t num_NAcodes, num_EUcodes;
```

Enregistrez les modifications, vérifiez le code et programmez votre Arduino !

Si vous êtes vraiment impatient et ne voulez même pas corriger rapidement le croquis de Ken, oui, vous trouverez une version corrigée sur le GitHub du magazine...

7. LE RÉSULTAT

Sans surprise en alimentant l'ensemble, en reliant la broche 5 à la masse pour utiliser les « messages » européens et après une pression sur un bouton, les codes d'arrêt d'une centaine de téléviseurs sont envoyés. L'utilisation de la led de puissance apporte des résultats étonnants. Durant nos essais, nous sommes arrivés à éteindre une TV Philips depuis l'extérieur d'un bâtiment en nous plaçant de l'autre côté de la rue. Il est d'ailleurs probable que d'autres TV du voisinage aient été éteintes accidentellement par la même occasion.

De la même manière avec une telle puissance d'émission, la led placée dans un bureau a permis d'éteindre cette même TV placée dans une autre pièce au bout d'un couloir. Le fait que les murs de nos bureaux soient blancs n'est sans doute pas sans rapport avec une telle efficacité. En effet, les signaux infrarouges se sont sans doute réfléchis à plusieurs reprises jusqu'à atteindre le téléviseur. Et ce avec une intensité encore largement suffisante pour que le code puisse être interprété.

Avec ce montage il faut cependant garder à l'esprit deux ou trois choses :

- Il est peut-être illégal, là où vous vous trouvez, de diffuser ainsi un signal hors d'une enceinte privé.
- Les codes sont envoyés les uns à la suite des autres et il est possible qu'un code corresponde effectivement au bouton d'arrêt mais qu'un autre ait une fonction différente (« chaîne + » par exemple) remettant alors le téléviseur en route.
- Si un téléviseur est en veille, l'envoi du code d'arrêt peut le mettre en fonction ce qui est exactement l'effet opposé à celui recherché.
- L'utilisation d'une led aussi puissante envoyant des codes avec une porteuse à 36 Khz ou 38 Khz aura tendance à « couvrir » ou du moins perturber les autres signaux IR et ainsi rendre les télécommandes inutilisables.
- Adapter le montage pour une utilisation sur batterie 9V et s'en servir dans au rayon Hifi d'une grande surface doit être terriblement amusant mais sans doute aussi une très mauvaise idée. Ne vous laissez pas emporter par la fièvre de l'IR car n'oubliez pas, avec un grand pouvoir arrivent... des gens qui vous disent que vous avez des responsabilités. Plus sérieusement, ne faites pas n'importe quoi. **DB**

CYGWIN : VOUS REPRENDREZ BIEN UN PEU DE LINUX DANS VOTRE WINDOWS...

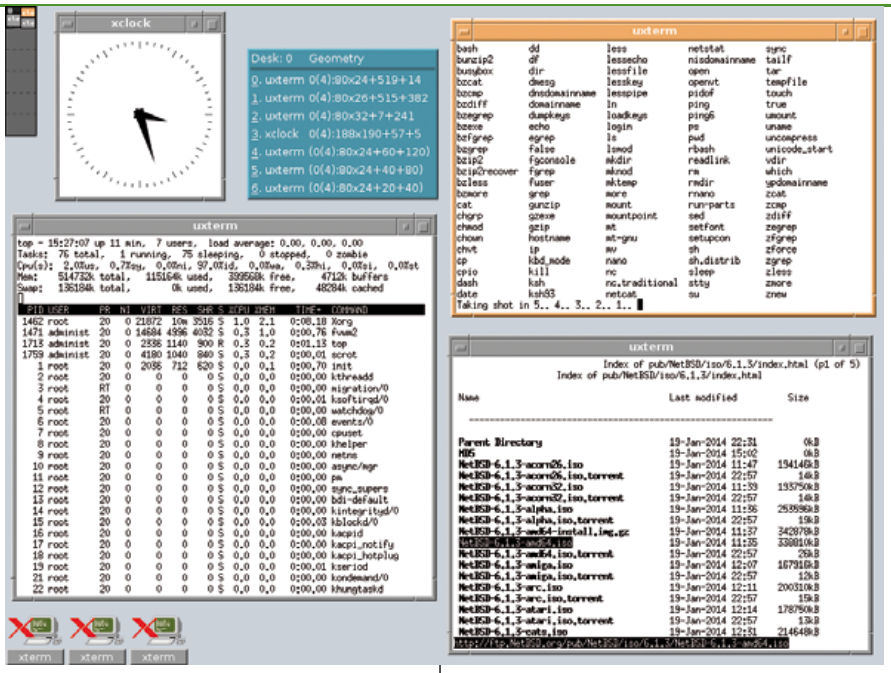
D. B.

Le projet initial peut paraître tordu puisque sous le nom Cygwin se cache ni plus ni moins que l'idée de disposer d'outils « Unix-like » sous Windows. L'arrivée de la Raspberry Pi et d'autres nombreuses cartes fonctionnant sous GNU/Linux donne de nouvelles raisons d'utiliser Cygwin. Voyons tout cela ensemble...

Mais pourquoi diable vouloir quelque chose d'aussi spartiate que des outils en ligne de commandes venant du monde Unix sous Windows qui est un havre de graphisme, d'ergonomie et de convivialité (sic) ?

Commençons par le début. Les philosophies d'Unix, Mac OS et Windows ont toujours été très différentes, chacun de ces systèmes adressant spécifiquement certains besoins et certaines préférences des utilisateurs. Initialement, et sur certains points c'est toujours valable actuellement, chaque système a des carences particulières qui peuvent être comblées par ses congénères. Ainsi on a pu voir se rapprocher les différents concepts les uns des autres. Mac OS est devenu Mac OS X, héritant les caractéristiques du vieux système Mac OS 9 (comme le *Finder*) mais aussi et surtout de NeXTstep/OpenStep et ses fondations Unix/BSD. GNU/Linux a adopté de plus en plus de comportements propres à Windows avec des interfaces graphiques plus « lisses » et moins austères. Et Windows a allègrement pioché dans Mac OS X avec Vista tout en bénéficiant, dès 1995, des avantages du projet Cygwin. Précisons que ce dernier est un logiciel libre tout comme les différents éléments qu'il est susceptible d'installer.

Ainsi, pour répondre à la question initiale, depuis le début du projet, les utilisateurs intéressés par Cygwin sont principalement des utilisateurs de systèmes Unix, dont GNU/Linux, Solaris, les différents BSD, AIX, Irix, etc. Ces utilisateurs qui se servent intensivement des outils en ligne de commandes et bénéficient des avantages qu'ils offrent, en utilisant Windows, se trouvent restreints et contraints dans un système qui n'est pas parfait (aucun système ne l'est). Le fait est qu'en ayant l'habitude d'un système, il s'avère très difficile d'en utiliser un nouveau, du moins tout aussi pleinement que le système de départ. Pour pallier à cela, Cygwin fournit tout ce qu'il faut pour que l'utilisateur Unix retrouve ses petits sous Windows et puisse garder ses réflexes durement acquis. Ajoutons à cela que, d'un point de vue de la programmation, Cygwin permet aux développeurs de créer des outils pour Windows en utilisant des standards très proches de ceux d'Unix. Nous n'entrerons pas dans le détail de cet aspect particulier mais sachez que des choses aussi simples que d'accéder au réseau et d'y lire ou écrire des données se fait très différemment d'un système à un autre. Cygwin n'est donc pas qu'un jouet ou qu'un élément de confort, en entreprise, il permet surtout d'utiliser au mieux les compétences existantes et donc de gagner en efficacité, d'un point de vue de la gestion des ressources humaines.



Le GNU/Linux le plus utilisé, la distribution Ubuntu, est aujourd'hui bien différente des premières versions du système qui étaient utilisables, certes, mais principalement par des personnes connaissant déjà les systèmes Unix ou souhaitant disposer d'un tel système pour leur propre usage. Ici, l'une des plus anciennes interfaces graphiques ou gestionnaires de fenêtres, FVWM, créé en 1994. C'était ça, Linux, dans ce temps là.

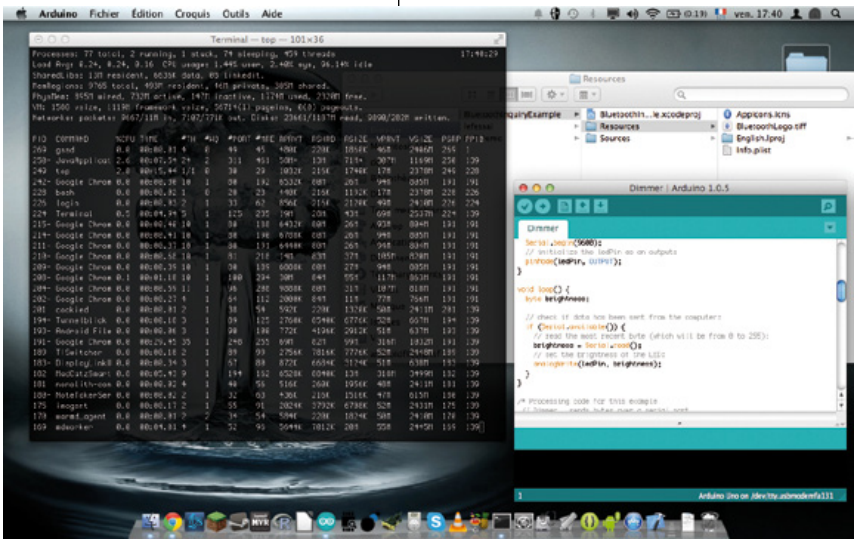
Faisons une petite parenthèse de l'autre côté du monde informatique, en précisant que les utilisateurs ayant découvert Mac OS X dès 2001 ont eu accès aux fonctionnalités que nous allons détailler. En effet, le Mac OS X peut être vu comme une adaptation de NeXTstep, le système utilisé sur les ordinateurs construit par NeXT, la société fondée par S. Jobs peu après son départ/éviction d'Apple en 1985. Or NeXTstep est basé sur un système Unix/BSD et Mac OS X hérite de bon nombre de fonctionnalités qui s'y trouvait, dont une ligne de commandes, les outils Unix courants et une partie de l'architecture générale. Ainsi, et cela va sans doute ravir les Apple-fan qui nous lisent, ce qu'un utilisateur Windows obtiendra avec Cygwin, les utilisateurs Mac en dispose depuis plus de 10 ans sans rien faire...

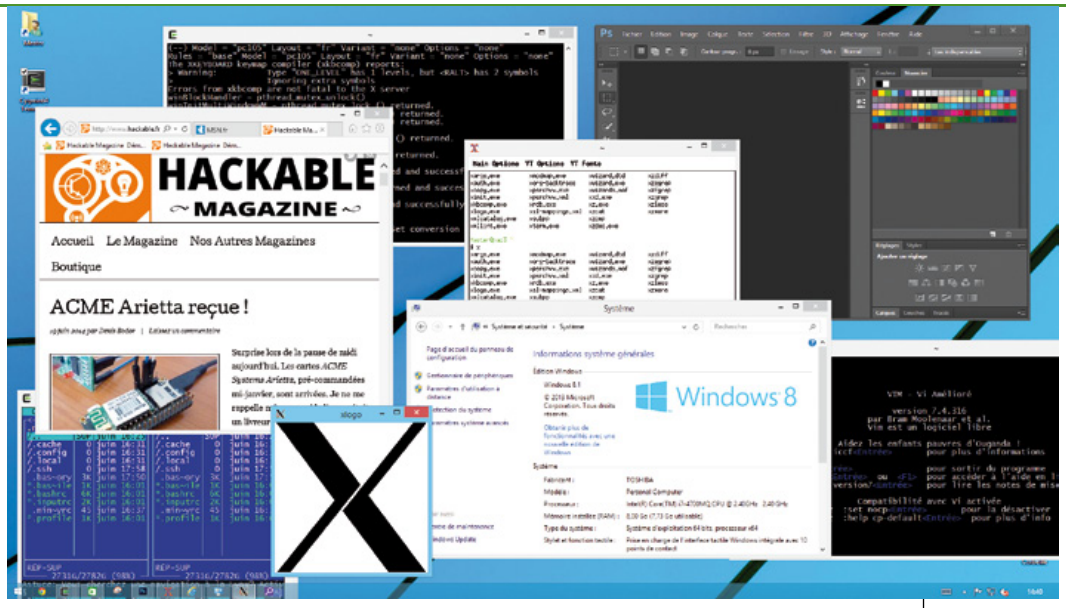
Terminons cette introduction en précisant clairement ce qu'est Cygwin : il s'agit d'un projet et d'une couche de compatibilité. Il se compose principalement de trois choses :

- Une DLL : Si vous êtes coutumier de longue date de Windows, le terme ne doit pas vous être inconnu (`vbrun300.dll` doit sans doute vous rappeler de vieux souvenirs). Une DLL est un morceau de code utilisé par plusieurs applications. Comme avec une boîte à outils, différents programmes peuvent utiliser le code contenu dans la DLL, ce qui évite à chacun d'entre eux de devoir intégrer ce code lui-même. Sous GNU/Linux ce sont des `.so` et sous Mac OS X, des `.dylib`. La DLL ou bibliothèque Cygwin fourni les outils permettant aux programmes Unix de fonctionner sous Windows.
- Un ensemble d'applications, de programmes et de bibliothèques (plus de 3200 en tout) : Ceux-ci sont identiques à ceux qu'on trouve sur les différents systèmes Unix actuels. Ainsi, par exemple, le shell également appelé interpréteur de commandes, qui est votre « agent de liaison » avec le système. Celui installé par Cygwin s'appelle Bash, le même que celui utilisé par GNU/Linux, Mac OS X ou votre Raspberry Pi. Lorsque je dis « le même », c'est vraiment le même ! Il a simplement été compilé pour être utilisable avec Cygwin comme il l'est dans les autres systèmes. Il en va de même pour toute une collection d'outils et d'applications ainsi mises à disposition par le projet.
- Un installateur graphique permettant de facilement installer les outils dont on a besoin en plus de la fameuse DLL. Il ne s'agit pas d'un simple système d'installation comme il en existe pour les applications Windows courantes. Il permet également de mettre à jour les différentes applications, d'en installer de nouvelles et de les désinstaller.

Enfin, ce que Cygwin n'est pas : **il ne s'agit pas d'un émulateur**. Cygwin n'est pas une solution pour exécuter des programmes GNU/Linux ou Mac OS X dans Windows. Vous ne pouvez donc

Apple, depuis l'arrivée de Mac OS X en 2001, livre ses Mac avec un système incluant une ligne de commandes et de nombreux outils qu'on trouve également sous GNU/Linux, BSD, Solaris, etc. Ceci est un héritage du système NeXTstep





Un Windows 8 comme vous ne l'avez sans doute jamais vu. Non, il ne s'agit pas de fenêtres d'un émulateur mais de ce qu'on peut obtenir en installant Cygwin : le meilleur de deux mondes très différents

pas simplement copier un programme depuis un de ces systèmes et tenter de le lancer sous Windows, même avec Cygwin d'installé. Vous retrouvez dans Cygwin les mêmes applications que dans un GNU/Linux, mais elles ont été recompilées à partir des sources, c'est à dire reconstruites spécialement pour que Windows puisse lancer Windows/Cygwin. Exactement le même principe que lorsque vous compilez un croquis Arduino pour une carte Uno, celui-ci ne fonctionnera pas sur une Launchpad de Ti, même si le croquis est absolument identique, il faudra le recompiler avec l'environnement Energia.

1. POURQUOI CYGWIN PEUT VOUS INTÉRESSER ?

En dehors du fait de bénéficier du confort d'utilisation d'une vraie ligne de commandes et d'une interface de programmation Unix qui sont, somme toute, des points importants pour des spécialistes, Cygwin présente des avantages pour l'utilisateur Windows « standard » et le bidouilleur. En effet, l'arrivée des nano-ordinateurs sous GNU/Linux comme la Raspberry Pi, la BeagleBone ou la récente ACME Arietta, implique déjà l'utilisation d'un système de type Unix.

Les ressources limitées de ces ordinateurs font qu'il n'est pas vraiment raisonnable d'utiliser des outils graphiques lourds comme on en trouve dans les distributions GNU/Linux courantes sur PC. Il devient très intéressant d'apprendre à utiliser la ligne de commandes pour des tâches courantes, la modification de la configuration ou l'exploration du système.

Disposer d'outils similaires sinon identiques dans son Windows peut donc être une manière d'apprendre plus vite. Il n'est pas possible d'installer Windows sur la Raspberry Pi mais il est possible d'installer un peu d'Unix dans Windows et ce, sans rien casser et sans avoir à installer deux systèmes sur son PC (ce qu'on appel un dual-boot). Avec Cygwin vous pouvez utiliser votre Windows en ligne de commandes tout comme vous le faites avec votre Raspberry Pi, par exemple.

Mais un autre point est encore plus avantageux ! Le GNU/Linux installé sur votre nano-ordinateur ou votre PC dispose de fonctionnalités particulières qui ne sont pas présentes par défaut dans Windows. L'une d'entres elles s'appelle SSH. SSH permet de vous connecter vers et depuis une machine en ligne de commandes à travers le réseau. Il existe une solution pour Windows, PuTTY, qui permet de vous

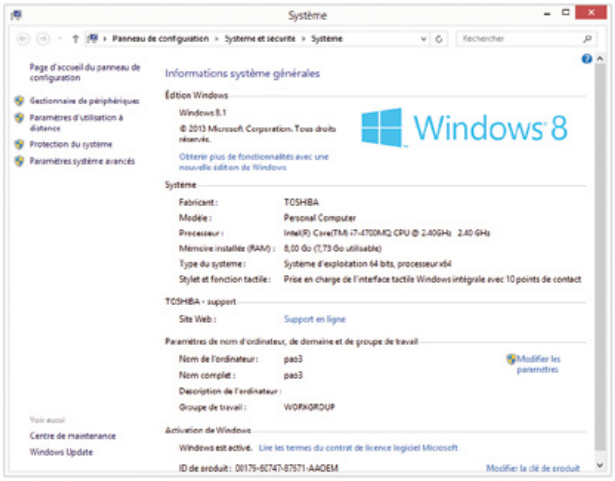
connecter depuis Windows vers votre GNU/Linux mais elle n'offre pas tous les avantages d'un vrai OpenSSH. Les outils de Cygwin vont beaucoup plus loin puisqu'il vous devient possible depuis une Raspberry Pi, de vous connecter à votre Windows en ligne de commandes, avec un environnement très similaire. Et SSH ne se limite pas à cela. Ce n'est là qu'un exemple car tous les autres outils sont également présents dans Cygwin comme ceux permettant de transférer des fichiers, de tester le réseau, d'écrire des petits scripts, de faire du Python, d'installer un serveur Web, de manipuler ou convertir des fichiers, etc.

Avec Cygwin d'installé sous Windows, vous vous retrouvez donc avec le plus parfait moyen de communication carte/PC qui soit tout simplement parce que, des deux côtés, ça parle la même langue...

2. INSTALLATION DE CYGWIN

L'installation de Cygwin a été prévue pour être la plus simple possible... du point de vue d'un utilisateur Unix. On parlait de mondes et de philosophies différentes en début d'article, en voici la parfaite démonstration. Ne vous attendez donc pas à un système d'installation comme on en trouve pour la plupart des applications Windows (vous savez, clic, suivant, suivant, suivant, Terminé, sans rien lire).

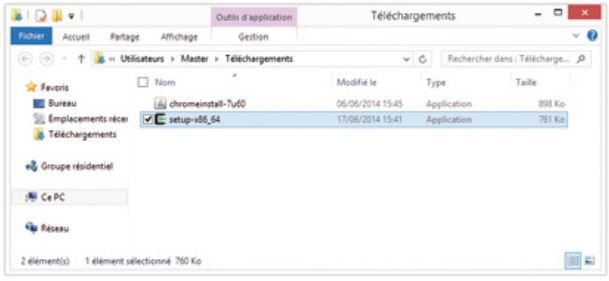
Nous avons ici choisi de détailler l'installation sur Windows 8.1. Il s'agit de la version la plus récente du système de Microsoft, pour certains la pire depuis Vista, pour d'autres la meilleure depuis XP. Quoi qu'il en soit, ce qui va être détaillé ici est applicable aussi bien à Windows 8.1, qu'à 8 ou encore 7 ou XP. Mais avant toutes choses, vous devez savoir si votre système est 32 bits ou 64 afin d'installer la version la plus adaptée de Cygwin. Un petit tour dans le Panneau de configuration, à la section « Système et sécurité » et « Système » et vous apprendrez ce qu'il faut savoir sur votre Windows :



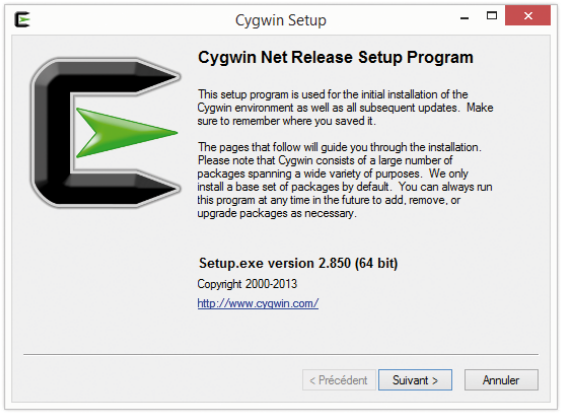
La ligne « Type de système » nous apprend ici que nous sommes sur une machine 64 bits avec un système 64 bits. La question qui se pose alors est la suivante : 32 ou 64 bits ? Une application 64 bits ne fonctionne que sur une machine 64 bits, mais une application 32 bits fonctionnera, en principe, sur du 64 bits aussi bien que sur du 32 bits. Avec Cygwin, vous devez peser le pour et le contre : l'intérêt du 64 bits est en premier lieu de pouvoir accéder à plus de mémoire, le désavantage tient dans le fait que tous les outils n'existent pas en version 64 bits (environs 200 éléments manquent à l'appel). Il y a donc des applications que vous ne trouverez pas dans cette version et vous ne pourrez donc pas les installer. Une recommandation simple : si vous ne cherchez pas à utiliser Cygwin pour des applications particulièrement gourmandes en mémoire (typiquement des applications scientifiques), préférez la diversité et le choix, optez pour le 32 bits.

Nous pouvons maintenant nous rendre sur **http://cygwin.org** pour télécharger le programme dans la bonne version : `setup-x86.exe` ou `setup-x86_64.exe` respectivement pour 32 ou 64 bits.

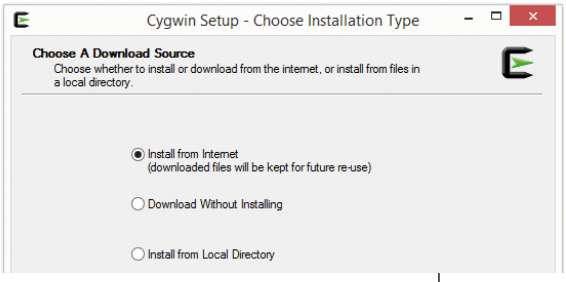
Après téléchargement, lancez le programme qui se trouve normalement dans le dossier des téléchargements (ou à l'endroit où vous l'avez placé) et confirmez l'exécution. Il sera judicieux, plus tard, d'éventuellement le déplacer car c'est ce même programme qui gère les mises à jour et l'installation/désinstallation de composants :



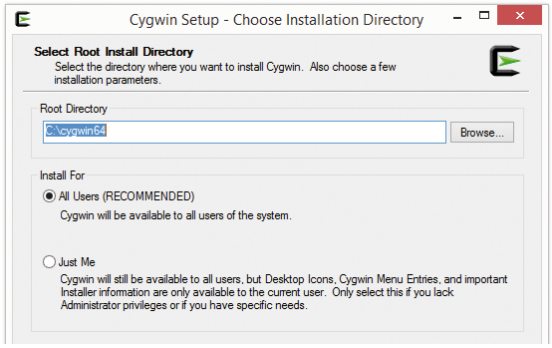
Une fois le lancement confirmé après le message d'alerte de Windows (ouh, le programme vient de la jungle Internet, il faut faire attention !), vous voici avec l'écran d'accueil de l'installateur Cygwin. Notez qu'un numéro de version est spécifié (ici 2.850) ainsi que la plateforme 32/64 bits pour laquelle l'installeur est conçu :

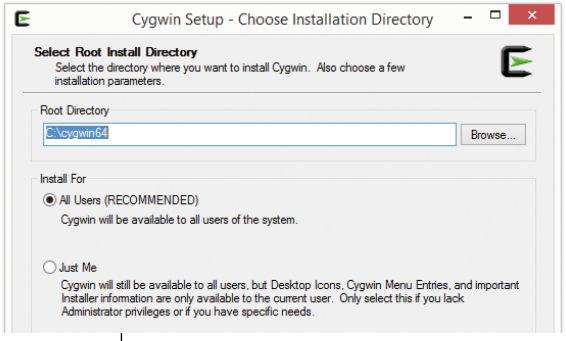


Après avoir cliqué sur « Suivant », le programme vous demande de choisir la source pour l'installation. Vous avez le choix entre une installation depuis le net (*Install from Internet*), le téléchargement seulement (*Download Without Installing*) ou installation depuis un répertoire (*Install from Directory*). Vous pouvez, par exemple, télécharger les éléments pour ensuite les copier sur une clé USB puis relancer l'installation sur une autre machine en installant depuis un répertoire. Pratique pour une machine sans connexion à Internet. Ici nous optons pour l'installation classique depuis Internet :

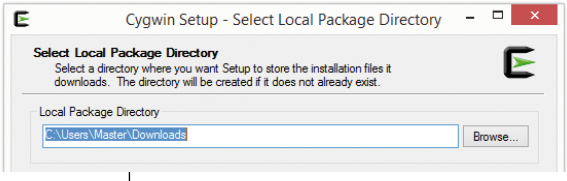


Vient ensuite le choix du répertoire d'installation. Par défaut, il s'agit de `C:\cygwin64` pour la version 64 bits et `C:\cygwin` pour la version 32 bits. C'est là que seront installés les différents fichiers, dans un répertoire bien séparé du reste du système et avec une arborescence ressemblant fortement à une installation GNU/Linux tel que celle d'une distribution ou d'une Raspberry Pi :

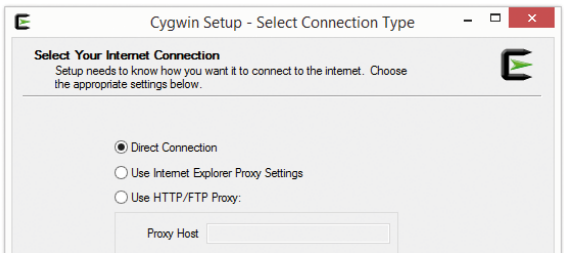




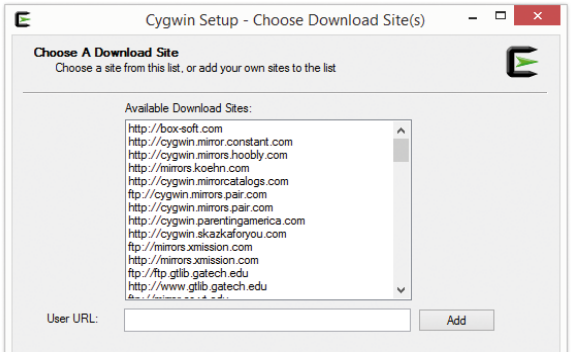
Après validation, l'installeur vous demandera de choisir un dossier pour stocker les éléments téléchargés. Par défaut c'est le dossier « Téléchargements » de votre dossier personnel. Dans la fenêtre s'affiche le vrai nom du répertoire. Ne vous inquiétez donc pas de lire **`Download`** à la place de « Téléchargement » (**`Desktop`** est « Bureau », **`Pictures`** est « Images », **`Music`** est « Musique », etc). Sachez également que les fichiers ne seront pas copiés en vrac, un sous dossier spécifique à Cygwin sera créé et votre dossier de téléchargement restera bien propre :



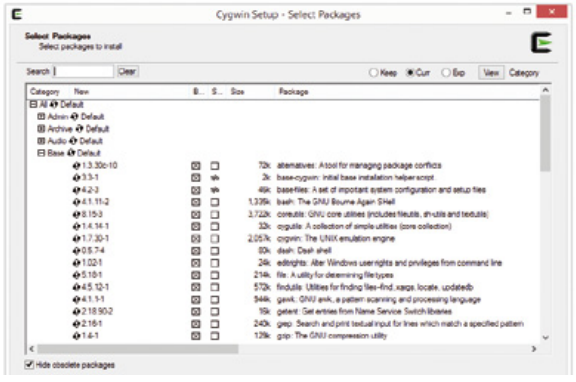
Afin d'accélérer les téléchargements sur un réseau d'entreprise, par exemple, il est possible d'utiliser un proxy. C'est une sorte de serveur qui sert de cache pour les accès à internet. Si vous ne savez pas ce que c'est ou qu'une telle chose n'est pas présente sur votre réseau, laissez simplement le choix sur « Direct connection » :



Cygwin ne date pas d'hier et n'est pas un petit projet anodin. Pour accélérer les installations, tout un nombre de serveurs ont été créés partout dans le monde avec une copie à jour des éléments pouvant être installés. On appelle cela des serveurs miroirs ou tout simplement des miroirs. Dans la longue liste qui se présente à vous, choisissez un serveur proche de chez vous, dans votre pays ou dans un pays frontalier. Il ne semble pas y avoir de miroir français dans la liste, nous sommes en Alsace, nous choisissons donc un miroir en Allemagne :



Après validation, l'installeur va récupérer la liste des éléments disponibles et l'utiliser pour vous présenter une interface où vous pouvez choisir ce que vous souhaitez installer :



Une sélection par défaut est déjà faite avec les éléments de base indispensables. Vous avez donc automatiquement un shell (l'interpréteur de commandes), les outils qui l'accompagne et les bibliothèques pour que tout cela fonctionne. Parcourez la liste ou utilisez la fonction de recherche pour ensuite cliquer pour sélectionner les éléments complémentaires à installer. Ce système est un peu particulier pour Windows.

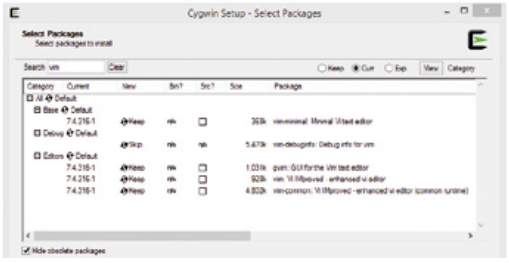
L'interface est la même que vous ayez déjà installé Cygwin ou non. Voyez cela comme un gestionnaire de paquets comme on en trouve sous GNU/Linux.

Lors de la première installation, la fenêtre vous présente les colonnes :

- « Category » : la section dans laquelle est placé l'élément. Le tout est organisé par thème,
- « New » : le numéro de version actuellement disponible sur le serveur,
- « Bin? » (ratatiné en « B... ») : case à cocher pour une installation du « binaire », l'élément dans sa version directement utilisable,
- « Src? » (« S... ») : case à cocher pour l'installation des sources, ceci n'est intéressant que si vous comptez modifier les éléments et les recompiler vous-même,
- « Size » : la taille de l'élément,
- « Package » : le descriptif de l'élément.

Tout ce que vous avez à faire pour installer des éléments est de cocher les cases dans la colonne « Bin? ». Il vous sera possible, par la suite de lancer à nouveau l'installeur pour compléter ou modifier votre installation. Rien n'est figé.

La fenêtre qui se présente dans ce cas est un peu différente :



Une nouvelle colonne fait son apparition, « Current ». Elle contient le numéro de version de l'élément déjà installé sur votre ordinateur ou rien, s'il n'est pas encore installé. Vous pourrez cliquer dans la colonne « New » pour décider ce que vous souhaitez faire. Si la ligne correspond à un élément installé, chaque clic permettra de choisir entre :

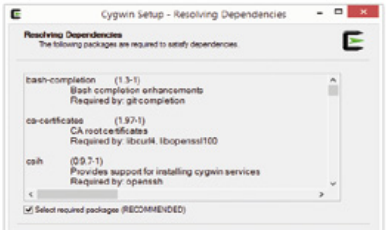
- « Keep » : garder ce qui est installé,
- « Uninstall » : désinstaller l'élément,
- « Reinstall » : supprimer et réinstaller,
- « Source » : pour installer également les sources (effet identique à un clic sur la case à cocher de la colonne « Src? »),
- un numéro de version : pour choisir une version spécifique à installer s'il en existe plusieurs.

Avec une ligne concernant un élément non installé, vous aurez le choix entre :

- « Skip » : pour ne rien faire et sauter la question,
- un numéro de version : pour choisir la version à installer.

Le principe est donc de faire votre petit marché dans cette liste de 3200 lignes pour ensuite cliquer sur « suivant ».

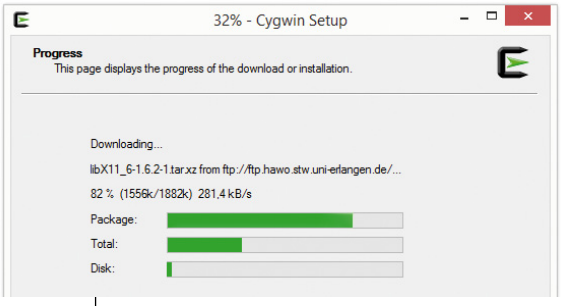
Il est plus que probable que la fenêtre suivante s'affiche alors :



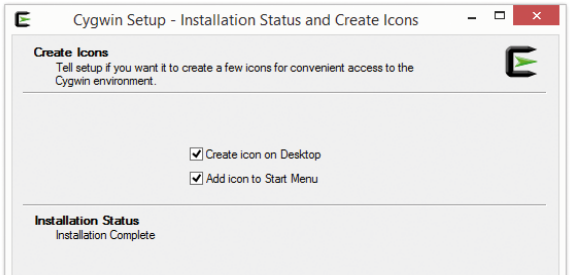
Tout comme l'ensemble des éléments nécessite la DLL Cygwin pour fonctionner, certains éléments en nécessitent d'autres pour être utilisés. Le principe est le même. Si l'outil A et l'outil B, par exemple, doivent manipuler des images, il est plus intelligent pour les programmeurs de placer tout ce qui concerne cette gestion d'images dans une bibliothèque qu'on appellera C. Les outils A et B, plutôt que de tout faire en double, reposeront simplement sur C. Ceci les rend dépendants de C mais pour améliorer cette gestion d'image, il suffira de changer C sans toucher aux travaux sur A et B. Ce type de choses existe également sous Windows mais c'est bien plus présent dans les systèmes Unix comme GNU/Linux. On parle alors de dépendances et de gestion de dépendances.

Pour que vous n'ayez pas vous-même à gérer tout cela, Cygwin, comme les distributions GNU/Linux, le fait à votre place. C'est la résolution de dépendances. Inutile donc de devoir courir après « qui a besoin de quoi » à la main, c'est automatique mais Cygwin vous signal, tout de même, les résultats de son travail en vous affichant cette fenêtre. Il arrive tantôt que des dépendances en cascade provoque l'installation indirect de dizaines d'éléments en raison de la sélection d'un seul d'entre eux (exemple typique, ImageMagick, un ensemble d'outils de manipulation d'images). De nos jours ce n'est pas bien grave puisqu'on dispose généralement d'espace disque à foison. Mais sur une petite machine avec un petit disque il peut être important de prêter une attention toute particulière à cette fenêtre.

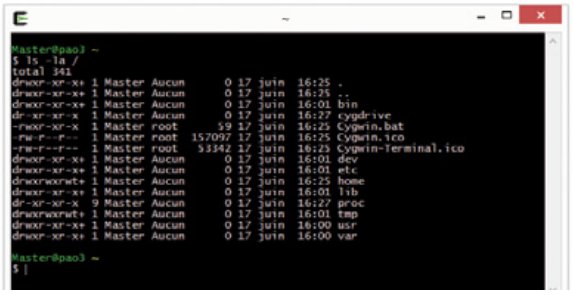
Dès l'étape de résolution de dépendances passée en cliquant sur « Suivant », l'installation effective débute :



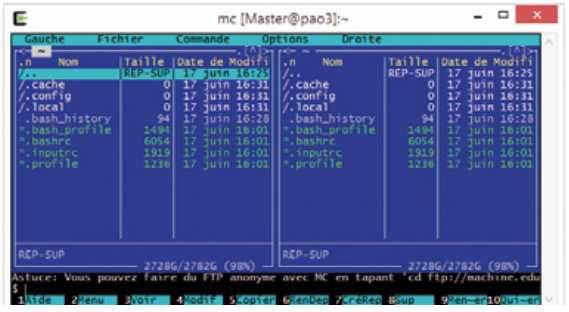
Après téléchargement et installation de tous les composants, l'installeur vous propose d'installer une icône sur le bureau ainsi que dans le menu démarrer (même s'il n'existe plus vraiment dans Windows 8) :



Il vous suffira maintenant de cliquer sur l'icône sur le bureau pour lancer « Cygwin Terminal ». Dans les faits, ceci lance une fenêtre de terminal pour taper des commandes. Il s'agit d'un accès au shell, exactement comme avec GNU/Linux ou Mac OS X et son application Terminal :



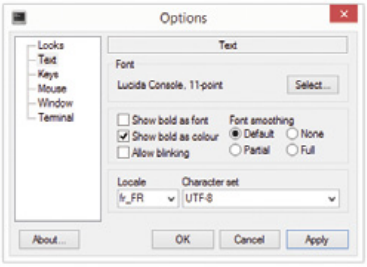
Tout ce que vous avez choisi d'installer est directement disponible. Outils, commandes, programmes... tout est là. Si, par exemple, vous aviez, fût un temps, l'habitude d'utiliser Norton Commander sous MS/DOS, vous serez ravi d'apprendre qu'il existe un clone, Midnight Commander, parfaitement similaire mais ajoutant bon nombre de fonctionnalités :



Bien entendu, tout l'intérêt de l'installation de Cygwin réside principalement dans l'utilisation de la ligne de commandes. Si vous avez choisi d'installer OpenSSH vous pouvez immédiatement contacter votre Raspberry Pi en utilisant la commande **ssh** suivie du nom d'utilisateur, d'un **@** et de l'adresse IP de la carte. Si un accès SSH est disponible (vous l'avez activé via **raspi-config**), le mot de passe de l'utilisateur sera demandé et après saisie et validation, vous obtiendrez une ligne de commandes sur la carte. Exactement comme si vous aviez lancé un terminal sur celle-ci !

Si l'aspect de la fenêtre Cygwin ne vous convient pas (police trop petite), cliquez avec le bouton droit

de la souris sur le haut de la fenêtre et choisissez « Options ». Vous pourrez alors régler tout un tas de paramètres, dont la taille des caractères :



A présent, prenez le temps de faire le tour du propriétaire, d'ajuster le nombre d'éléments installés et de constater par vous-même les différences qu'il peut y avoir entre Cygwin et un système GNU/Linux comme celui d'un nano-ordinateur. Si vous êtes coutumier de la ligne de commandes sur ce genre de plateforme, vous remarquerez rapidement que, même si toute la partie « utilisateur » est parfaitement identique, d'autres éléments plus spécifiques sont absents ou différents. Tout ce qui est intimement lié au système (et au noyau en particulier) est naturellement impacté, absent ou émulé (le contenu de `/proc` par exemple).

Dans le même ordre d'idées, comme le but initial de Cygwin est de créer plus de confort pour l'utilisateur, profitez-en pour personnaliser l'environnement comme vous le feriez (ou l'avez déjà fait) avec votre nano-ordinateur. Qu'il s'agisse du shell, des alias ou des configurations de différents outils (Vim, Emacs, MC, etc), tout ceci est identique à un véritable système Unix ou GNU/Linux. Installez-vous confortablement.

Enfin, pour finir avec cette partie exploration, comprenez bien que vous n'avez pas à faire à un système Unix mais bien à Windows. Même si le répertoire personnel est différent de celui de Windows, que tout est basé sur une fausse racine (`/` est en réalité le répertoire d'installation de Cygwin) et que votre vraie racine `C:\` est représentée en `/cygdrive/c`, il s'agit bel et bien de Windows. Le système n'a pas changé, vous disposez simplement d'une nouvelle manière de l'utiliser. Ainsi, si vous lancez la commande `calc`, c'est le fichier `/cygdrive/c/WINDOWS/system32/calc/calc.exe` (et donc `C:\WINDOWS\system32\calc\calc.exe`) qui est utilisé et bel et bien la calculatrice incluse avec Windows. Le principe est simple, le shell cherchera d'abord une commande dans Cygwin et s'il ne la trouve pas, cherchera dans le reste de l'installation Windows. C'est le principe de fonctionnement du `$PATH` qu'on retrouve sous Unix et sous MS/DOS (en fait, c'est `$PATH`). Donc, si vous vous posez la question, oui, il est possible de lancer n'importe quel programme Windows depuis la ligne de commandes. C'est normal, puisque **les outils installés par Cygwin sont des programmes Windows**..

3. ET LES MISES À JOUR ?

Les mises à jour des paquets installés pas Cygwin sont fréquentes, le projet est très actif. Pour procéder à une mise à jour, il vous suffit de lancer l'installeur comme si vous souhaitiez installer de nouveaux éléments. Par défaut, les éléments disponibles à l'installation dont le numéro de version est supérieur à celle d'un élément installé sont automatiquement sélectionnés pour installation. Ainsi, il vous suffit de lancer l'installeur et de cliquer sur « Suivant ».

Si vous souhaitez voir l'étendu de cette mise à jour avant installation, un bouton « View » se trouve à votre disposition en haut à droite de la fenêtre de sélection des éléments. Cliquez dessus jusqu'à ce qu'à sa droite s'affiche « Pending » (en attente). Vous obtiendrez la liste de ce qui doit être installé.

Ce bouton permet de changer le mode de visualisation :

- « Category » : l'arborescence classique pour explorer la liste,
- « Full » : la liste complète de tout ce qui existe,
- « Pending » : en attente d'installation,
- « Up To Date » : ce qui est à jour, autrement dit tout ce qui est installé sur votre machine,
- « Not Installed » : la liste totale à l'exclusion de ce qui est déjà installé. DB

UN ÉCRAN À 5 EUROS POUR VOTRE RASPBERRY PI

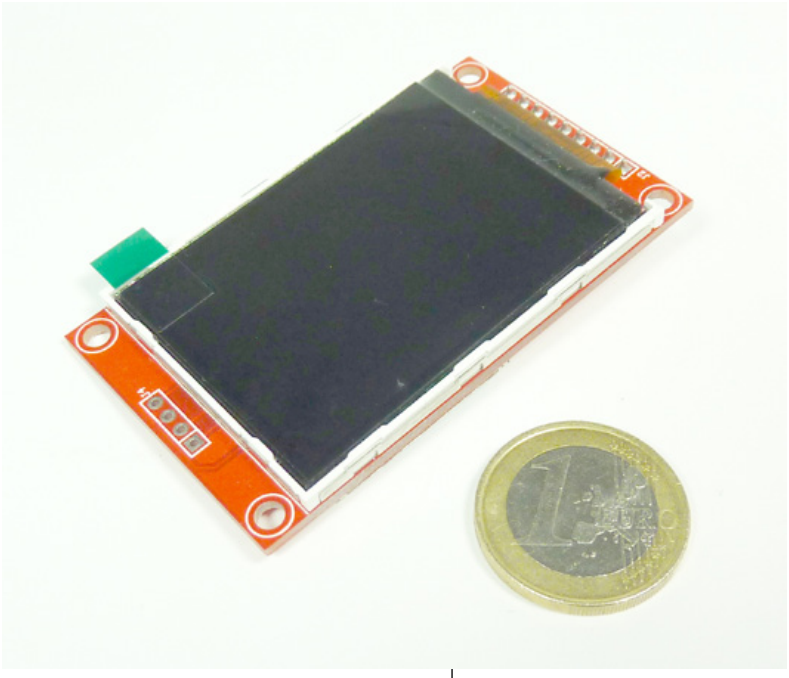
D. B.

Il y a bien des solutions pour équiper sa Pi d'un écran de taille réduite mais le budget est généralement assez conséquent. Même si la carte peut, en effet, être simplement connectée à un moniteur HDMI ou via une prise RCA/composite, il y a une solution qui s'adaptera aussi bien à un projet à base de Raspberry Pi que d'Arduino. Comble du bonheur, nos amis chinois étant de la partie, elle est extrêmement économique.

Coupons court aux suppositions. Nous parlons ici d'un écran LCD TFT très économique et bas de gamme, de taille réduite (moins de 6 cm de diagonale) et ayant une résolution très faible (320×240). Ceci ne permettra pas de raisonnablement faire de votre Raspberry Pi un ordinateur portable. Ce n'est d'ailleurs pas l'objectif. Il s'agit plutôt de fournir à la carte un système d'affichage adapté aux projets devant fournir des informations à un utilisateur comme pour de la domotique par exemple, un contrôle de température ou encore un GPS fait maison. Les performances sont loin d'être optimales mais il faut garder le sens des proportions. On parle ici d'investir moins d'un sixième du prix du nano-ordinateur. Ceci sera cependant largement suffisant pour diverses activités pouvant aller jusqu'au jeu. Je pense naturellement à l'émulation de consoles d'arcade ou de vieux ordinateurs personnels pour lesquels une résolution de 320×240 pixels était déjà bien au delà de leur capacité de l'époque.

1. DE QUEL ÉCRAN PARLE-T-ON ?

L'écran présenté en photo dans cet article n'est qu'une option, il est distribué sous de nombreux noms et les fabricants se clonent entre eux. Notre module d'affichage a été acheté sur eBay auprès du vendeur appelé *Czb Electronic* sous la désignation « *2.2" Serial SPI TFT LCD Display Module 240×320 Chip ILI9340C PCB Adapter SD Card* ». Vendu 4,24 euros (port gratuit depuis Shenzhen), autant vous dire que nous en avons pris 10 d'un coup misant sur le fait que la description était fidèle à la réalité.



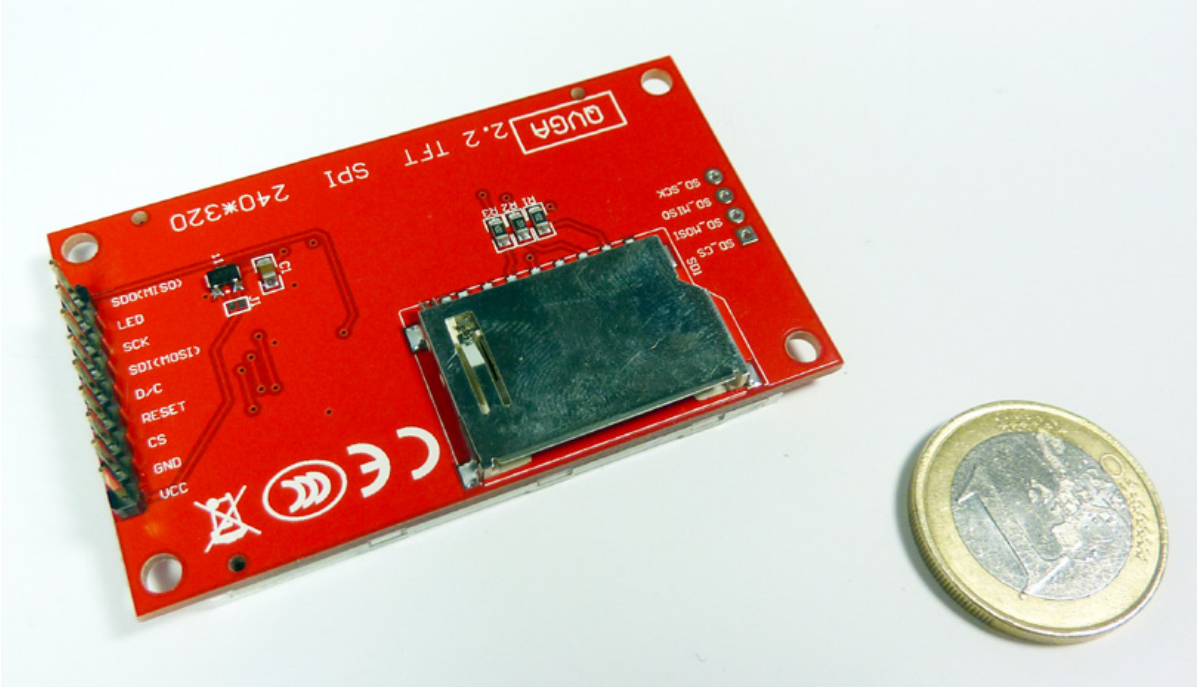
L'écran LCD tel qu'il vous arrivera tout droit de Chine. Tâchez de laisser le film protecteur le plus longtemps possible sur l'écran pour les différentes manipulations avant l'installation finale. Le revêtement plastique qui le recouvre se raye très facilement.

Ce module comprend un afficheur LCD TFT de 2,2 pouces de diagonale piloté par un contrôleur ILI9340C, le tout monté sur un circuit imprimé disposant d'une série de broches soudées. Comme à l'usine ils ont dû remarquer qu'il restait un peu de place, il s'est vu également équipé d'un emplacement pour carte SD avec des connecteurs à part non soudés. Nous ne parlerons pas de ce point particulier dans cet article et la présence de cette emplacement n'impacte en rien la connectique ou le fonctionnement de l'écran lui-même.

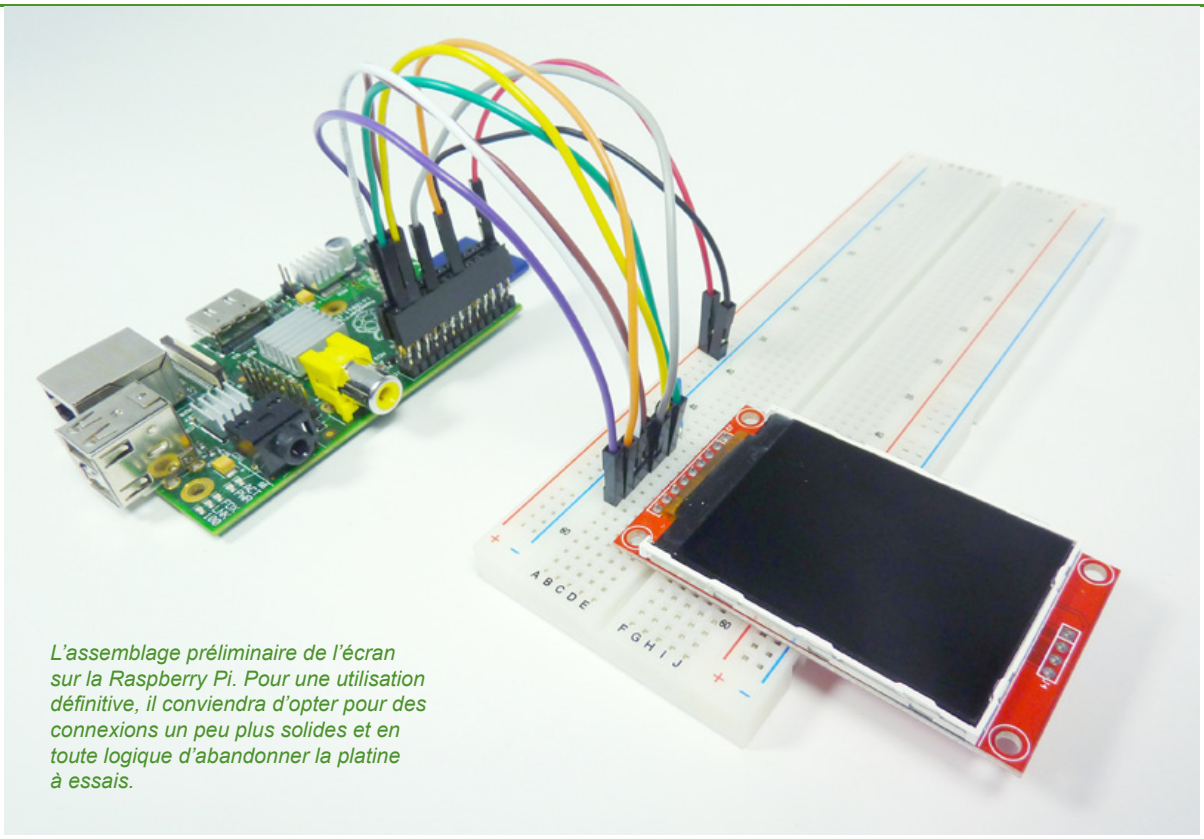
Plusieurs éléments sont importants dans la description d'un tel objet. Nous avons d'une part la dimension et la résolution de l'écran, la première en pouces (2,54 cm) et la seconde en quantité de pixels horizontaux et verticaux. Ces éléments sont facilement évaluables. Autre caractéristique de l'écran, le nombre de couleurs est souvent spécifié, avec ici « *262K/65K* ». Avec des écrans bas de gamme et économiques, ne vous attendez pas à avoir un affichage 24 bits dit *Truecolor*. 65536 couleurs c'est déjà très bien.

L'écran est piloté par un contrôleur appelé *driver*, *Driver IC* ou *controller* dans les annonces. Celui-ci sert d'interface entre le monde extérieur et l'affichage. En effet, un écran LCD est une matrice permettant de contrôler le passage de la lumière provenant d'une source placée à l'arrière de l'écran. Il faut trois points de jonction de la matrice pour composer un pixel, une jonction par couleur (rouge, vert et bleu). Pour ce module, il faut donc contrôler 230400 points de jonction en manipulant les lignes et les colonnes permettant d'y accéder. Inutile donc d'espérer piloter cela de manière logicielle depuis une Raspberry Pi ou une Arduino. Un circuit spécialisé est dédié à cette tâche et celui-ci reçoit ces ordres de l'extérieur via un protocole spécifique. La liaison peut être de type série comme ici avec du SPI ou parallèle ce qui est bien plus rapide mais consomme bien plus de ports d'entrée/sortie (GPIO).

Le type de liaison est une chose, le « langage » utilisé sur cette liaison en est une autre. C'est là que le contrôleur revêt toute son importance puisqu'il est impératif que le protocole soit pris en charge par le pilote



Au dos du circuit se trouve la connectique permettant la liaison avec la Pi ou toute autre carte capable de piloter un bus SPI et quelques entrées/sorties. On distingue ici également l'emplacement pour carte SD disposant, à droite, de ses propres connexions, sans broches soudées.



L'assemblage préliminaire de l'écran sur la Raspberry Pi. Pour une utilisation définitive, il conviendra d'opter pour des connexions un peu plus solides et en toute logique d'abandonner la platine à essais.

(Raspberry Pi) ou la bibliothèque (Arduino) que vous pourrez utiliser. Si ce n'est pas le cas, il faudrait développer ce support et lire, comprendre et implémenter le protocole décrit dans la documentation du fabricant (si elle existe). C'est une tâche difficile et longue qui permet dans un premier temps d'arriver à afficher quelque chose sur l'écran. Ensuite, il faut encore améliorer son support de manière à ce que l'affichage soit le plus rapide possible tout en utilisant le moins de ressources. Ce n'est qu'ensuite que le support peut être alors considéré comme implémenté correctement. Bref, c'est une tâche titanesque, nécessitant d'importantes compétences en développement et la garantie de passer un certain nombre de nuits blanches.

Ici, nous sommes en présence d'un contrôleur ILI9340C et nous avons acheté ce module pour cette raison précise. Celui-ci est pris en charge par le pilote FBTFT développé par *Noralf Tronnes* (alias Notro) qui en supporte de nombreux autres. Profitons-en pour tirer notre chapeau à ce développeur pour son travail exceptionnel (c'est lui qui a passé les fameuses nuits blanches à notre place).

Notez que Adafruit commercialise un module similaire (**http://www.adafruit.com/products/1480**) de même résolution et dimension, et utilisant le même contrôleur au prix de... $25 (20 euros). Certes, le module intègre une puce permettant l'utilisation en 5V et on peut supposer que les critères de fabrications et la qualité des composants sont sensiblement supérieurs, mais à ce prix là on se paie 4 exemplaires en provenance directe de Shenzhen. Le tout en pouvant utiliser le support Arduino développé par Adafruit sans le moindre problème (en dehors de la compatibilité 5V puisque le module chinois lui ne tolère que du 3,3V).

Pour conclure cette partie, on remarquera que le pilote de Notro supporte également les écrans des téléphones portables Nokia 3310. Son prix cependant est plus élevé que notre présent module, sans doute parce que l'époque où Nokia fabriquait LE mobile à posséder est bel et bien derrière nous (certains diraient « le temps où Nokia faisait des téléphones et non des mini-tablette-PC-téléphone sous Windows »). Ceci dit, si vous avez l'une de ses antiquités sous la main, rien ne vous empêche de le désosser et d'en extraire l'écran pour disposer, sur votre Pi, d'un fantastique affichage monochrome de 84×40 pixels. Regardez sur Youtube, quelques bidouilleurs ont poussé le vice jusqu'à utiliser ce type d'écrans en ligne de commandes avec leur Pi. Il faut de très bon yeux mais ça marche !

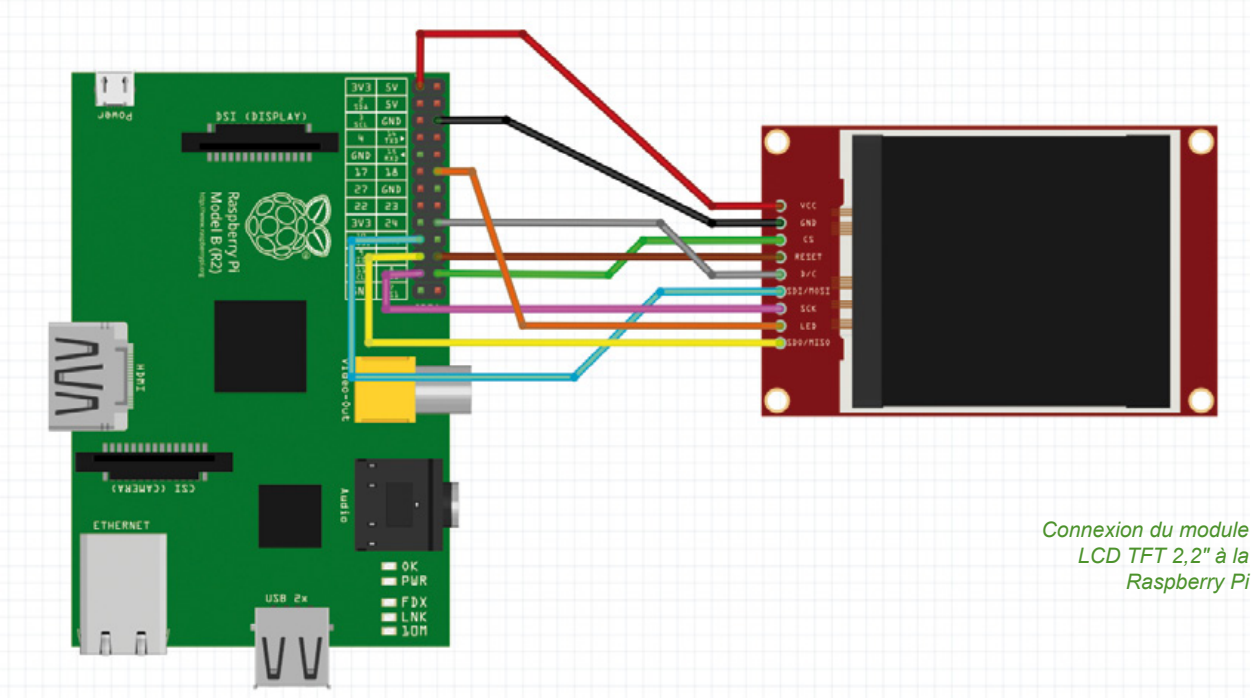
2. CONNEXION

Nous l'avons dit, le module est interfacé en SPI. Il s'agit d'un bus série très courant qui rend l'écran utilisable avec une vaste gamme de cartes dont la Pi et les Arduino. Un bus SPI (pour *Serial Peripheral Interface*) utilise 4 signaux ou lignes :

- SCLK ou CLK pour *clock* (horloge) est le signal qui cadence la communication,
- MOSI pour *Master Out Slave In* ou SDI pour *Seriat Data In* transporte les données du maître vers l'esclave,
- MISO pour *Master In Slave Out* ou SDO pour *Serial Data Out* assure le transport des informations dans le sens inverse (notez que MOSI/MISO est plus explicite que SDI/SDO qui dépend de l'endroit où on se place),
- SS ou CS pour *Slave Select* ou *Chip Select* permet au maître (celui qui contrôle la communication) de désigner l'esclave. Ceci permet d'utiliser les mêmes lignes SCLK, MOSI et MISO pour connecter plusieurs esclaves sous forme de bus, le maître choisissant à qui parler spécifiquement en utilisant SS/CS. Il y a, dans ce cas, autant de lignes CS que d'esclaves mais ceci évite d'avoir 4 lignes par périphérique (plus la masse et l'alimentation qui sont également communes).

La Raspberry Pi dispose d'un bus SPI accessible via le connecteur P1 mais il ne vous suffira pas de relier simplement ces quatre signaux, l'alimentation et la masse. L'écran utilise d'autres lignes pour être correctement contrôlé. Ainsi, l'interconnexion complète sera la suivante :

- VCC sur 3.3V (broche 17) : c'est l'alimentation qui, avec ce module spécifique est en 3,3 volts (NE LE BRANCHEZ PAS SUR DU 5 VOLTS sans vous être assuré trois fois que le module dispose d'un régulateur adapté, ce qui n'est pas le cas avec ce modèle précis),
- GND sur GND (broche 20) : l'incontournable masse,
- CS sur CE0 (broche 24) : la ligne de sélection,



Connexion du module LCD TFT 2,2" à la Raspberry Pi

- RESET sur GPIO 25 (broche 22) : le signal permettant la réinitialisation de l'afficheur,
- D/C sur GPIO 24 (broche 18) : une ligne permettant de choisir si on envoi des données d'affichage ou des commandes au contrôleur ILI9340C,
- SDI/MOSI sur MOSI (broche 19) : la communication de la Pi au module,
- SCK sur SCLK (broche 23) : horloge,
- LED sur GPIO_18 (broche 12) : c'est l'alimentation du rétro-éclairage obtenu via les 4 leds blanches placées derrière l'écran LCD. La documentation du pilote de Notro précise une connexion sur le GPIO 18 ce qui permet de contrôler le rétro-éclairage de manière logicielle mais vous pouvez également l'alimenter séparément si vous disposez des caractéristiques précises d'alimentation ou du matériel permettant de les mesurer. Une documentation d'un autre module très similaire précise une connexion des 4 leds en parallèle avec un courant de 15mA par led et une tension de 3,2V (3,4 max). Il est donc possible de connecter cette broche à une alimentation 3,3V avec une petite résistance pour limiter le courant (voir d'utiliser un MOSFET pour que la Pi contrôle tout de même l'alimentation) mais cela dépend totalement de chaque module. Pour l'heure le pilote ne permet qu'un rétro-éclairage tout-ou-rien. Il n'est pas possible de régler l'intensité, sauf en bidouillant.
- SDO/MISO sur MISO (broche 21) : la communication du module à la Pi.

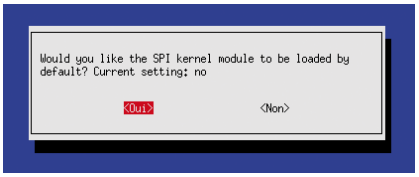
Vérifiez les connexions pour vous assurer que tout est en ordre et démarrez votre Raspberry Pi comme d'habitude. Il est temps de passer à la configuration logicielle.

3. CONFIGURATION

Pour faire fonctionner la Raspberry Pi avec ce module ou un autre modèle compatible, il est nécessaire d'utiliser un pilote particulier. Avec un système GNU/Linux les pilotes sont des composants du noyau et peuvent être présents sous deux formes : totalement intégrés au noyau ou sous forme de modules qu'il est possible de charger/décharger en fonction des besoins. D'une manière ou d'une autre il sera donc nécessaire de mettre à jour le noyau du système de votre Pi. Pas d'inquiétude, cela reste relativement simple à condition de suivre correctement une procédure claire.

Le développeur ayant créé le support pour ces écrans LCD a non seulement fait les choses correctement mais a également simplifié la procédure d'installation. Cependant, avant de procéder à cette installation, il est nécessaire de s'assurer qu'une fonctionnalité est correctement activée dans la configuration de la Pi : le bus SPI.

Deux solutions sont disponibles pour cela. Soit vous lancez l'outil **raspi-config** avec **sudo** pour parcourir les menus et vous assurer que le support SPI est activé (« *Advanced Options* », « *SPI* »), soit vous éditez (avec **sudo nano**) le fichier **/etc/modprobe.d/raspi-blacklist.conf** où vous devez vous assurer que la ligne **blacklist spi-bcm2708** est soit absente, soit débute par un **#** afin que le module ne soit pas interdit de chargement (blacklisté).



Activation du bus SPI dans raspi-config

Dans l'un ou l'autre cas, vous devrez redémarrer le système pour prendre ces changements en compte. Vous pourrez alors vous assurer que le support SPI est activé avec la commande :

```
$ dmesg | grep spi
[5.366697] bcm2708_spi bcm2708_spi.0:
   master is unqueued, this is deprecated
[5.547757] bcm2708_spi bcm2708_spi.0:
   SPI Controller at 0x20204000 (irq 80)
```

ou encore

```
$ cat /proc/interrupts | grep spi
80: 0   ARMCTRL  bcm2708_spi.0
```

Vous pouvez, à présent, mettre à jour votre Raspberry Pi avec les éléments développés par Notro grâce à la commande :

```
$ sudo REPO_URI=https://github.com/notro/rpi-firmware rpi-update
[...]
*** Running pre-install script
     Work around rpi-update issue #106
     rm -rf //lib/modules/3.12.25+/kernel
*** Updating firmware
*** Updating kernel modules
*** depmod 3.12.25+
*** Updating VideoCore libraries
*** Using HardFP libraries
*** Updating SDK
*** Running ldconfig
*** Storing current firmware revision
*** Running post-install script
     /lib/firmware
*** Deleting downloaded files
*** Syncing changes to disk
*** If no errors appeared, your firmware
was successfully updated to
237f5ea618d34c1539f3cf2c61e4c20f06e89649
*** A reboot is needed to activate the new firmware
```

Ceci a pour effet de télécharger un nouveau noyau ainsi que les modules qui l'accompagnent intégrant bien entendu le support des écrans LCD développés par Notro. Comme le précise le message, il est nécessaire de redémarrer la Raspberry Pi pour que les changements prennent effet.

4. CHARGEMENT ET ESSAIS

Nous disposons, après ce démarrage, d'une nouvelle version de la base du système avec un noyau à jour et des pilotes adaptés. Le pilote lui-même est capable de gérer plusieurs afficheurs de tailles et de résolutions différentes, reposant sur un nombre tout aussi important de contrôleurs spécifiques. Pour piloter notre écran nous devons donc charger le pilote en spécifiant le bon modèle.

Bien entendu, le module chinois économique n'est pas directement désigné mais il existe un modèle strictement équivalent vendu par Adafruit : **adafruit22a**. Nous pouvons donc faire notre premier essai en chargeant le module et en spécifiant ce modèle :

```
$ sudo modprobe fbtft_device \
  name=adafruit22a verbose=1
```

L'option **name** permet de préciser le modèle et **verbose** indique que des informations complémentaires doivent être retournées. Normalement, dès le chargement du module, l'écran doit s'allumer, tout en restant vide et noir. En revanche, le noyau nous indique la configuration utilisée et la bonne marche de la manipulation :

```
$ dmesg | tail
[...]
fbtft_device: Deleting spi0.0
fbtft_device:  GPIOS used by 'adafruit22a':
```

```
fbtft_device:    'reset' = GPIO25
fbtft_device:    'dc' = GPIO24
fbtft_device:    'led' = GPIO18
graphics fb1: fb_ili9340 frame buffer,
  240x320, 150 KiB video memory,
  4 KiB DMA buffer memory, fps=20,
  spi0.0 at 32 MHz
```

Le système utilise un mécanisme appelé *framebuffer* qui, grossièrement, est une représentation en mémoire des pixels qui composent l'écran. La Raspberry Pi dispose déjà d'un périphérique de ce type, accessible via **/dev/fb0**, c'est l'affichage connecté à la prise HDMI de la carte. La commande **fbset** nous retourne ses caractéristiques :

```
$ fbset
mode "656x416"
    geometry 656 416 656 416 16
    timings 0 0 0 0 0 0 0
    rgba 5/11,6/5,5/0,0/16
endmode
```

Nous pouvons faire de même pour **/dev/fb1** :

```
$ fbset -fb /dev/fb1
mode "240x320"
    geometry 240 320 240 320 16
    timings 0 0 0 0 0 0 0
    nonstd 1
    rgba 5/11,6/5,5/0,0/0
endmode
```

Grâce à la magie de la philosophie UNIX qui veut que « tout est fichier » dans le système, nous pouvons très simplement tester le fonctionnement de l'écran. Comme le *framebuffer* est un fichier (**/dev/fb1**), il suffit d'y écrire des données aléatoires pour qu'une myriade de pixels de couleurs variables s'affiche :

```
$ cat /dev/urandom > /dev/fb1
cat: erreur d'écriture: Aucun
espace
  disponible sur le périphérique
```

L'erreur qui apparaît est parfaitement normale puisque cette commande prend des données aléatoires dans **/dev/urandom** qui est une source sans fin et les copie dans **/dev/fb1**. A un moment, l'écran est rempli de pixels, il n'y a plus de place et la commande **cat** s'arrête. Sur l'écran, nous voyons apparaître quelque chose comme ceci :

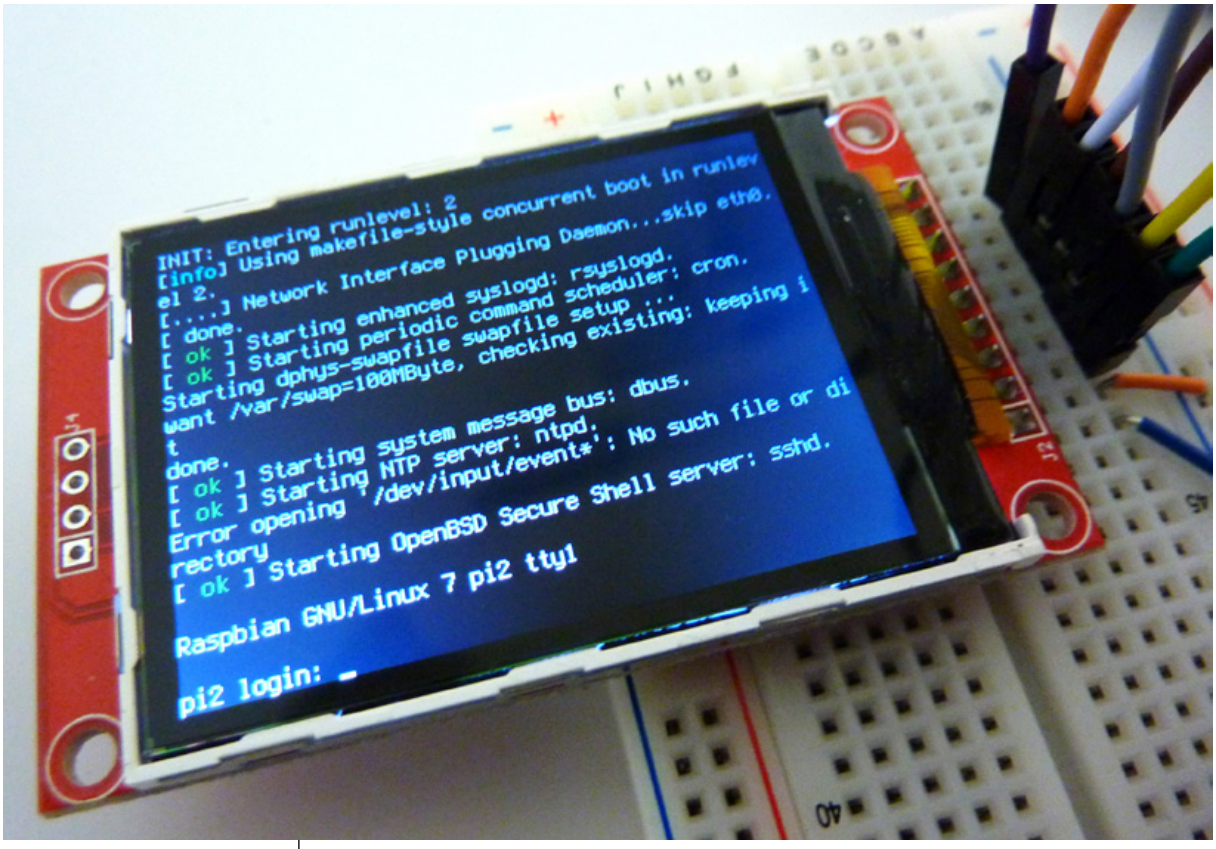
Premier essai de l'écran, un remplissage aléatoire de pixels

5. ÇA MARCHE, ON PEUT CONFIGURER LE SYSTÈME

Nous avons un périphérique d'affichage fonctionnel supplémentaire et nous pouvons donc nous en servir comme écran pour le système. Dans un premier temps, nous devons ajouter le chargement du module dans la configuration de manière à ce qu'il soit utilisé automatiquement au démarrage. Pour cela rien de plus simple, il nous suffit d'ajouter quelques lignes dans le fichier **/etc/modules** :

```
fbtft dma
fbtft_device name=adafruit22a
verbose=0
```

Nous connaissons déjà la seconde ligne puisque nous venons de l'utiliser en direct. Notez que nous avons mis **verbose** à **0** pour éviter d'afficher les messages d'informations. La première ligne charge le module **fbtft** qui est la base du pilote et nous lui passons l'option **dma**. Le DMA pour *Direct Memory Access* est une technologie permettant d'accélérer les échanges entre périphériques en évitant au maximum



Moment de bonheur intense après quelques redémarrages du système. Un choix de police judicieux pour ce minuscule écran le rend parfaitement utilisable en ajoutant simplement un clavier USB

l'intervention du processeur. Cette option permet donc d'utiliser les fonctions DMA du bus SPI de la Raspberry Pi afin d'accélérer l'envoi des données à l'écran.

Nous pouvons également augmenter la vitesse du bus SPI lui-même qui par défaut est à 32 Mhz en utilisant, pour le module **`fbtft_device`** l'option **`speed=48000000`** (48 Mhz). Enfin, plus intéressant encore, il est possible de « tourner » le *framebuffer* avec l'option **`rotate=90`**. Ainsi, l'orientation de l'écran passe du mode portrait (240×320) au mode paysage (320×240).

Le périphérique sera présent au démarrage mais ce n'est pas pour autant que le système y affichera quoi que ce soit. Pour que cela fonctionne et que Linux affiche les informations de démarrage et une console sur l'écran, nous devons nous pencher sur les options contenues dans le fichier **`cmdline.txt`** présent sur la partition FAT de la carte SD. Ce fichier est accessible via **`/boot/cmdline.txt`** et il nous suffit de l'éditer pour ajouter, à la fin de la ligne, les options **`fbcon=map:10 fbcon=font:VGA8x8`**.

Ceci nous permet de préciser que nous souhaitons une console via ce nouveau *framebuffer* tout en spécifiant la police de caractères à utiliser. Une fois le fichier enregistré, il nous suffit de redémarrer la Pi et voir défiler adorablement les messages de démarrage sur ce tout petit écran. Mieux encore, nous sommes, comme sur le grand

écran, invité à entrer notre nom d'utilisateur et notre mot de passe et pouvons donc maintenant utiliser le nano-ordinateur comme nous le ferions habituellement mais avec un nano-écran et de méga-bons yeux.

Vous pouvez bien sûr jouer avec ces options. Le paramètre **font** est évident mais **map** est un peu plus délicat. Cette option permet d'établir la correspondance entre le terminal (tty) et le périphérique *framebuffer*. Ici nous utilisons **10** ce qui signifie que le premier terminal (tty1) correspond à **fb1** et le second (tty2) à **fb0** (la sortie HDMI). Et le tty3 ? C'est simple, on boucle, et ce sera donc **fb1**, puis tty4=**fb0**, etc. Si vous n'avez que faire de la sortie HDMI, utilisez simplement **map:1** et tous les terminaux seront associés au mini écran (vous pouvez basculer d'un tty à l'autre avec [alt]+[F1], [alt]+[F2], [alt]+[F3], etc). Notez au passage qu'une option **rotate:** existe mais elle ne concerne que la rotation de la console, non du *framebuffer* lui-même qui est géré par le pilote.

6. JOUONS !

Voir ainsi démarrer le système est déjà relativement jouissif mais il est possible d'améliorer les choses et de pousser un peu plus loin encore. Dans un premier temps, vous remarquerez sans doute que l'affichage n'est pas très propre en mode console. La faible résolution n'est pas idéale et ne s'accommode pas vraiment bien de la police par défaut utilisée. Nous pouvons régler rapidement le problème en reconfigurant le

```
[info] Using makefile-style concurrent boot in runlev
el 2.
[....] Network Interface Plugging Daemon...skip eth0.
[ done.
[....] Starting enhanced syslogd: rsyslogdError openi
ng '/dev/input/event*': No such file or directory
. ok
[ ok ] Starting periodic command scheduler: cron.
[ ok ] Starting system message bus: dbus.
Starting dphys-swapfile swapfile setup ...
want /var/swap=100MByte, checking existing: keeping i
t
done.
[ ok ] Starting NTP server: ntpd.
[ ok ] Starting OpenBSD Secure Shell server: sshd.
My IP address is 192.168.10.110

Raspbian GNU/Linux 7 pi2 tty1

pi2 login:
```

Notre console après redémarrage, utilisant une police Terminus 6×12, petit mais lisible

paquet **console-setup** avec la commande **sudo dpkg-reconfigure console-setup**. Dans les choix qui vous sont proposés, laissez les valeurs déjà en place jusqu'à l'étape du choix de la police et faites votre sélection. Nous avons constaté qu'avec ce module, la police la plus adaptée était Terminus en 6×12. Cette modification sera enregistrée et activée dès le prochain démarrage du système.

Le mode console c'est bien amusant mais on ne peut pas avoir le bureau graphique comme avec un grand écran en HDMI, n'est-ce pas ? Comme dirait Hassan Cehef, si, si, bien sûr, c'est possible, ça me fait plaisir (ah la belle époque des nuls) ! Et cela ne présente pas vraiment de difficulté car il suffit de se pencher sur le fichier **/usr/share/X11/xorg.conf.d/99-fbturbo.conf** que vous éditerez avec **sudo nano** par exemple pour changer la ligne

```
Option  "fbdev" "/dev/fb0"
```

en

```
Option  "fbdev" "/dev/fb1"
```

Le lancement de la commande **startx** depuis la console ou une configuration pour un démarrage en mode graphique via **raspi-config** et vous voici avec un bureau X sur votre écran miniature. Ce n'est

certes pas facile à utiliser mais cela vous permettra alors de lancer les applications en mode graphique qui ne peuvent se servir directement du *framebuffer*.

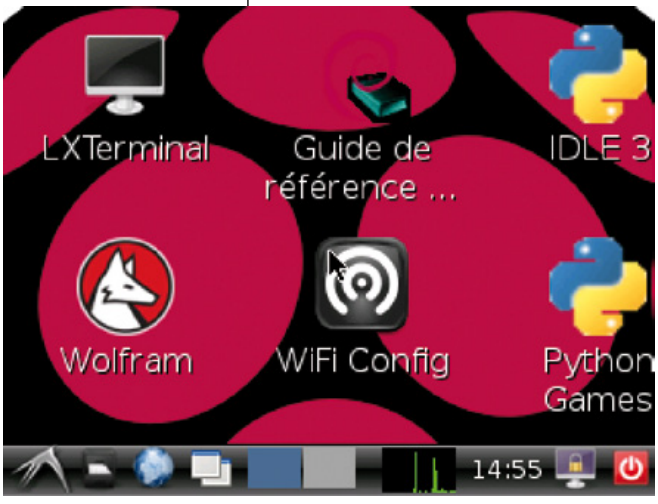
Dans la catégorie « j'utilise du matériel surdimensionné », si vous voulez transformer votre Pi en horloge amusante essayez ceci : installez **xdaliclock**, créez un fichier **.xinitrc** contenant une ligne **xdaliclock -fullscreen -nocycle -builtin1 -24 -fg yellow** et démarrer X avec **startx -- -nocursor** (attention au double tiret permettant de séparer les options de **startx** de celles que lui-même passe à **X**). Vous voici propriétaire d'une magnifique horloge avec un effet de morphing à chaque seconde (un CPU à 700 Mhz avec 512 Mo de RAM pour donner l'heure, c'est presque une Apple Watch). Et surtout n'oubliez pas que vous avez créé ce fichier qui lance automatiquement des commandes au démarrage du serveur graphique, parce que sinon vous risquez de vite devenir fou en cherchant pourquoi votre bureau n'apparaît plus quoi que vous fassiez.

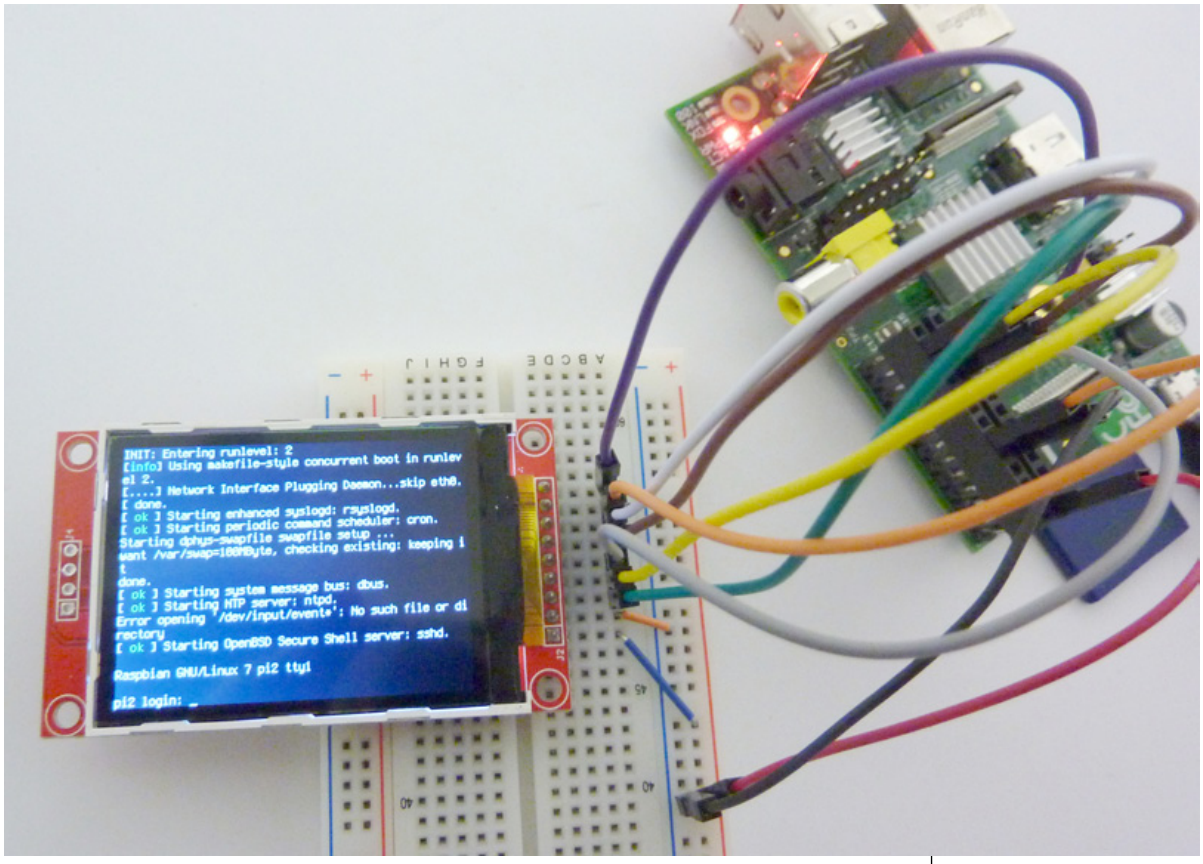
Mais certaines applications sont parfaitement en mesure de se passer de X. C'est le cas par exemple du lecteur multimédia MPlayer (paquet **mplayer**) qui sera alors parfaitement capable de lire une vidéo directement en console. Il suffit pour cela d'utiliser les bonnes options : **mplayer -vo fbdev2:/dev/fb1 -vf scale=320:-3 video.mp4**. **-vo** permet de spécifier le pilote Mplayer pour l'affichage en *framebuffer* ainsi que le fichier qui le représente (**/dev/fb1**). On précise également un filtre permettant de redimensionner la vidéo à la taille de l'écran, ici en mode paysage, et on accompagne le tout du nom de fichier à lire et l'affaire est dans le sac. Bien entendu, il est préférable de traiter les vidéos et les ré-encoder à la dimension de l'écran plutôt que de laisser le processeur de la Raspberry Pi faire cet éprouvant travail.

Il en va de même pour l'affichage d'images avec **fbi** (paquet éponyme) et voici que votre Raspberry Pi se transforme en cadre photo numérique en usant de quelques options : **fbi -a -noverbose -t 3 img/***. Ceci aura pour effet d'afficher en boucle les images présentes dans le sous-répertoire **img/**, avec un redimensionnement automatique (**-a**), pas de texte (**-noverbose**) et une pause de 3 secondes entre les images (**-t**).

En voyant les images qui illustrent cet article peut-être vous demandez-vous comment nous avons réussi à obtenir une telle qualité photographique. Il ne s'agit en réalité pas de photos mais de captures d'écrans. Celles-ci sont très facile à faire puisque, je le rappelle, un *framebuffer* n'est rien d'autre qu'un espace en mémoire. On utilise alors un outil appelé **fbcat** pour copier le contenu de **/dev/fb1** vers un fichier qui sera automatiquement au format PPM avec

Hé oui, un démarrage en mode graphique est parfaitement possible et c'est absolument mignon





Vue d'ensemble du montage en fonction sur la platine à essai. Rien ne vous empêche de connecter l'écran directement à la Raspberry Pi. Ici notre carte cobaye ayant déjà été passablement torturée, nous avons dû jongler avec l'inéluctable problématique des connecteurs mâles/femelles (vous aussi avez déjà remarqué qu'on a systématiquement jamais les bons câbles sous la main ?)

`fbcat /dev/fb1 > capture.ppm`. On convertira ensuite l'image soit avec Gimp après transfère du fichier sur un PC, soit directement avec un outil de la suite d'ImageMagick (paquet **`imagemagick`**) via **`convert capture.ppm capture.png`**.

7. POUR FINIR

Dès lors qu'on dispose d'un tel système d'affichage, les options offertes s'en trouvent démultipliées, tout comme les applications possibles. Au delà de l'aspect purement expérimental et du plaisir de voir s'afficher des messages et des interfaces similaires à ce qu'on peut voir sur un écran 17" ou 19" en HDMI, un écran de 320×240 pixels si peu coûteux permettra d'ajouter un vrai plus à n'importe quel projet. Même une simple base collectant des données s'en trouve améliorée dès lors qu'on peut surveiller d'un simple coup d'œil ce qui se passe. Il en va de même pour les projets domotiques, robotiques, serveurs multimédia, etc.

Bien sûr, l'utilisation qui est généralement faite de ce type d'écrans tourne autour de l'émulation de machines d'arcades ou de consoles de jeux portables (type Gameboy). Les réalisations qu'on peut trouver sur le web fonctionnent, le plus souvent, avec une distribution GNU/Linux spécialisée et basée sur Raspbian. La logique est similaire à celle permettant la réalisation d'une borne d'arcade à une échelle « standard » puisque l'affichage se résume à l'utilisation d'un *framebuffer* ou d'un serveur X exactement comme on le ferai avec l'écran en HDMI.

Nous reviendrons très certainement dans le futur sur ce sujet qui peut être intéressant indépendamment de l'affichage lui-même. **DB**

HACK RAPIDE : MISE À JOUR D'UNE SOURIS LUMINEUSE

D. B.

Parfois on achète des périphériques, matériels et équipements non seulement parce qu'on en a besoin mais aussi parce qu'ils sont beaux et ont un petit quelque chose en plus. Pourtant, même si cela suffit à déclencher un achat, on ne peut s'empêcher d'entendre une petite voix dans notre tête qui nous rappelle que le produit n'est ni unique ni très exactement tel qu'on l'aurait souhaité.

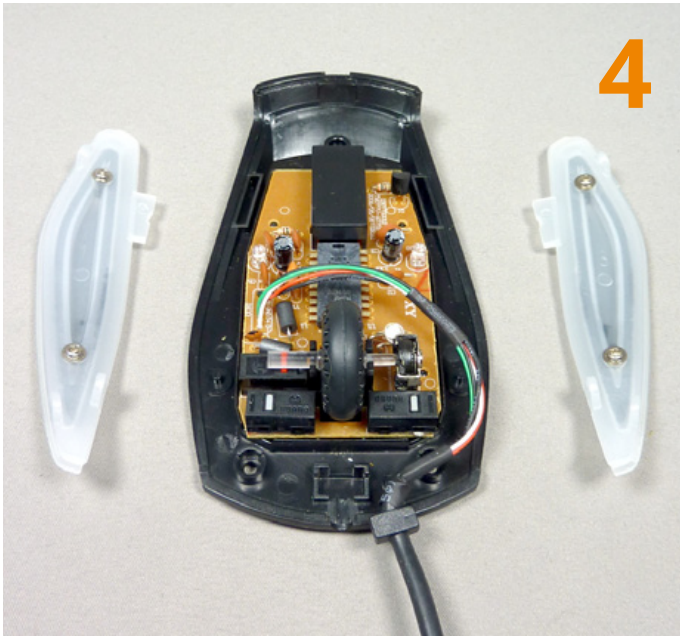
Une souris super-geek qui fait de la lumière c'est bien. Mais voilà, si seulement cette lumière était rouge, rose, verte... bref, en n'importe quoi d'autre que la couleur que le constructeur à choisi (blanc). C'est un syndrome courant et un exemple parfait est assez bien résumé sur le site Tokyoflash Japan qui semble prendre un malin plaisir à éviter systématiquement les combinaisons led/boitier qui me plaisent (mais pourquoi la Kisai RPM n'existe pas en rouge/chrome façon Cylon ? Pourquoi ?!). Mais revenons à notre souris, un autre problème pourrait être résumé par le simple fait d'entendre la simple exclamation : « *Elle est trop chouette ta souris ! C'est une Cherry ? Je vais acheter la même* ». Et vous voilà entrain de ruiner le reste votre journée à regarder votre souris en vous demandant ce que vous lui trouviez de si exceptionnel puisque le premier venu peut vous voler une partie du plaisir de la posséder.

Par contre en prenant ce design qui vous plaisait tant et en l'adaptant à votre envie initiale, les choses sont bien différentes. Non seulement vous avez un objet personnalisé et unique (ou presque) mais en plus ce même commentaire d'un collègue ou ami suscitera des sentiments très différents. Sentiment qu'on pourrait là encore résumer avec une phrase : « *ouais, bin, bonne chance pour trouver ce modèle, niark, niark, niark* ».

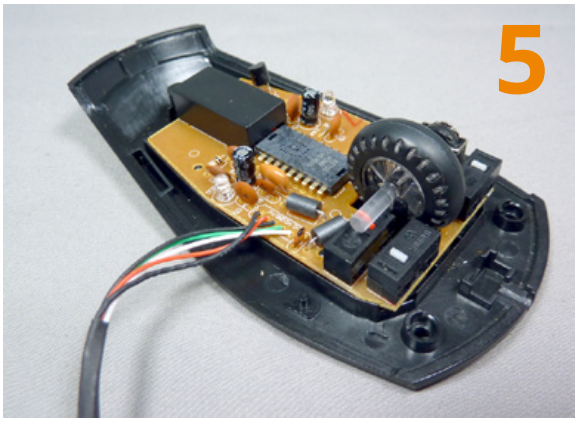
Notre « victime » est une souris optique filaire USB de marque Cherry et le modèle est Gentix Illuminated Mouse. Ce n'est pas une souris de gamer type Razer Ouroboros à 130€ mais juste un périphérique de PC standard à 12€ avec un petit plus sous la forme d'un éclairage interne blanc.

La souris émet de la lumière à la fois par ses côtés translucides et par la molette. Deux intensités lumineuses différentes alternent lors de l'utilisation : forte en action et légèrement réduite après moins d'une demi seconde sans bouger. Comme le montre cette photo, la lumière n'est vraiment visible que dans un environnement sombre. Avec un éclairage de bureau adapté pour le travail la lumière interne est à peine visible.

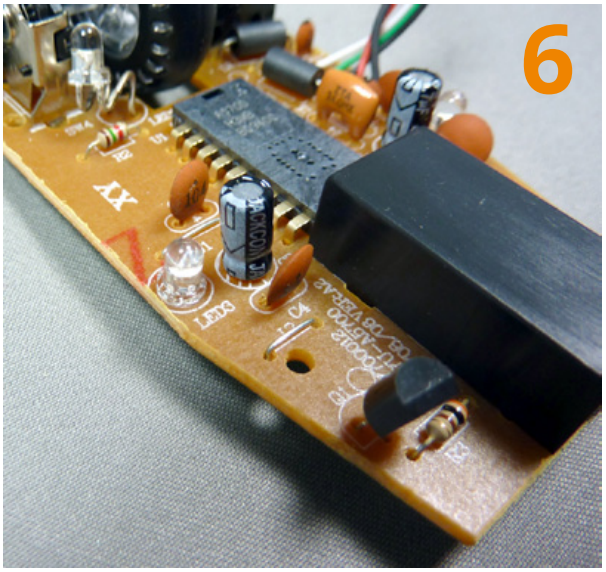
Pour découvrir les entrailles de la bête, comme de coutume avec un produit fini, le jeu consiste principalement à trouver où sont les vis. Dans ce cas-ci, deux sont placées sous les patins autocollant en téflon et une se trouve sous l'étiquette « Q.C. Passed » apposé après le contrôle qualité en sortie d'usine. On prend un vicieux plaisir à percer l'étiquette annulant ainsi la garantie (les vrais bidouilleurs n'utilisent jamais la garantie)



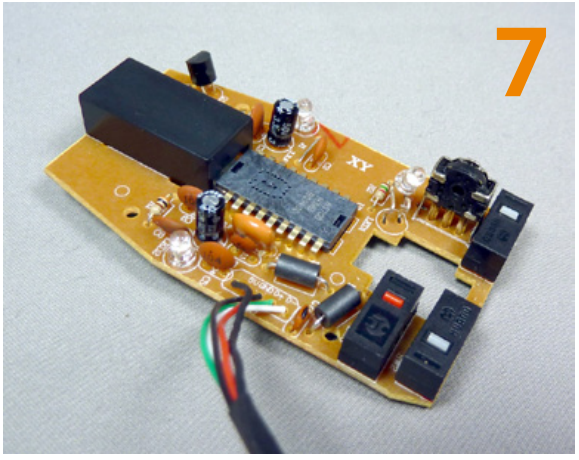
Les vis retirées, on peut simplement soulever la partie supérieure pour découvrir l'électronique du périphérique. Le circuit n'est ni collé, ni clipsé, ni visé. Ceci nous facilitera grandement la tâche, la colle c'est le mal.



On peut, à présent, profiter de l'occasion pour nettoyer l'intérieur et les quelques 2 Kg de poils de chat qui ont trouvés place un peu partout (merci Lithium, Carbon et Copper pour votre production incessante de phanères aux changements de saisons). L'ensemble du circuit est assez simple : laser, capteur, une route encodeuse, trois boutons poussoirs, des leds et des composants passifs.

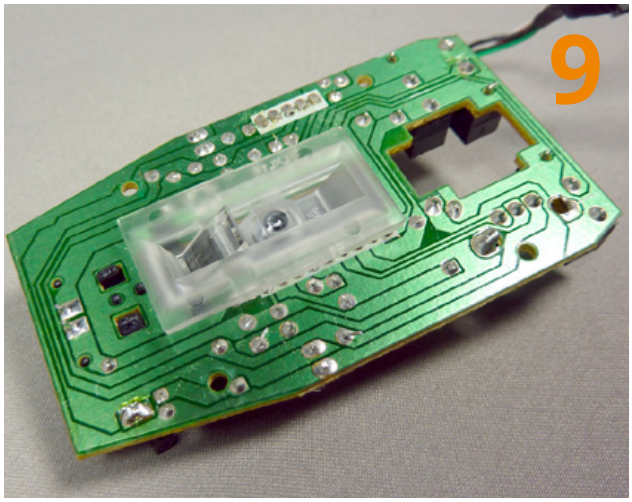


Voilà précisément ce qui nous intéresse. L'une des trois leds blanches permettant l'éclairage intérieur de la souris. Il y en a une de chaque coté et une autre orientée vers la molette. On remarque au premier plan un transistor PNP 2N3906 qui semble clairement contrôler ces leds. Nous n'avons pas besoin de nous pencher sur ce composant puisque nous ne comptons pas modifier les caractéristiques électriques du circuit.

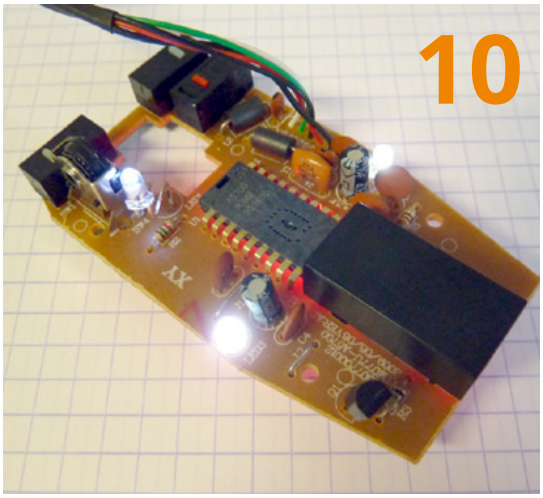


Après extraction sans difficulté du circuit de l'intérieur du boitier, on peut avoir une vue d'ensemble un peu plus satisfaisante. La manière dont la led de la molette a été montée est assez intéressante et mieux vaut le mémoriser car nous devrons procéder de même avec le composant de remplacement. Remarquez les boutons (switch) de marque Huano. Ceux-ci semblent détestés par certains gamers parce qu'ils demandent trop de pression du doigt et réduisent la vitesse dans les RTS, même si cela est appréciable pour les FPS. D'après la même source (sur overclock.net) les Huano permettent facilement 66 clics en 10s alors qu'une autre marque, Omrons, en permet 10 de plus dans le même temps (vous voyez, il n'y a pas qu'en électronique qu'il existe des passionnés obsessionnels).

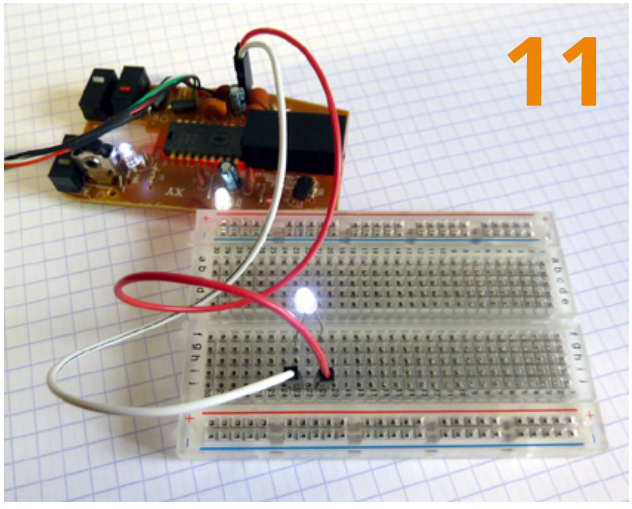
La molette peut être retirée de la roue codeuse pour faciliter les manipulations. Ingénieusement, celle-ci est constituée de matière plastique transparente diffusant la lumière mais est gainée d'une matière plus souple évitant de glisser lors de son utilisation. L'axe de rotation présente un « jeu » afin de pouvoir cliquer (troisième bouton de la souris).



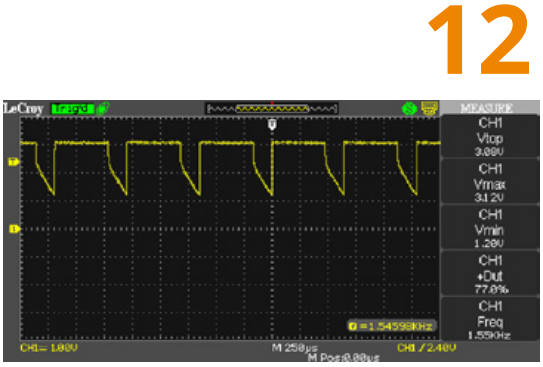
En observant les pistes du circuit on en apprend également beaucoup. Nos trois leds sont connectées en parallèle directement à la masse USB et au transistor. Les soudures sont propres et il sera aisé de les dessouder.



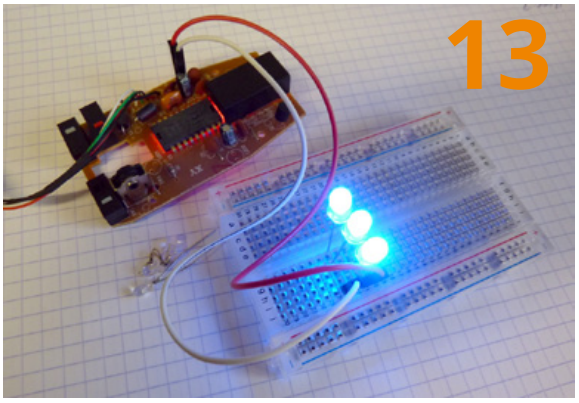
La mise sous tension de la souris via un bloc d'alimentation USB tel que ceux utilisés pour recharger les smartphones nous permettra de travailler en « live » sur le circuit sans risquer de faire réagir inutilement un PC. On s'interdira toutefois de dessouder/souder quoi que ce soit avec le circuit sous tension. Ceci peut paraître logique mais rappelons-le à tout hasard, comme le fait de travailler avec un environnement bien dégagé afin d'éviter tout court-circuits avec des composants qui traînent.



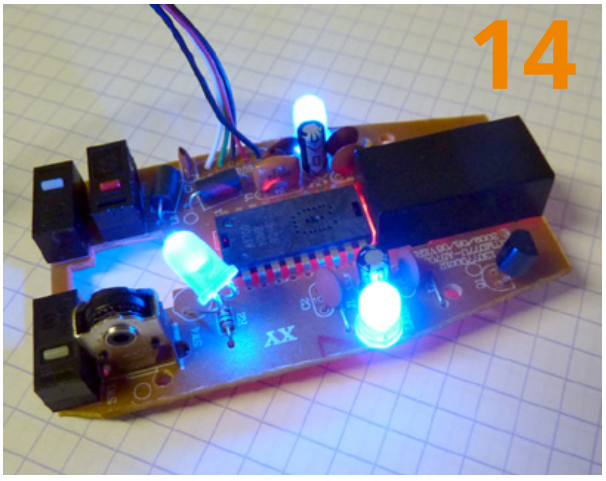
Nous ne pouvons pas faire n'importe quoi et notre premier mouvement sera de dessouder une led pour la déporter sur une platine à essais. Cela nous permettra de procéder à quelques mesures et en particulier la tension aux bornes de la led mais aussi le courant qui y circule. Ces mesures nous apprennent que la souris en activité, les leds utilisent un courant de 23 mA à 3V, ce qui est relativement classique pour des leds blanches.



En branchant un oscilloscope aux bornes de la led, on apprend également qu'en pleine activité celle-ci est alimentée continuellement. En revanche dans le mode d'intensité réduite, il s'agit de PWM ou quelque chose d'approchant. La mesure nous montre une fréquence de 1,55 Khz avec un rapport cyclique de 77%. Notez cependant que le signal n'est pas carré et que la tension ne descend pas en dessous de 1,2V. Il n'est même pas exacte de parler de vraie PWM en réalité.



Après une petite fouille dans le stock de leds offrant un flux lumineux important, on met la main sur des sympathiques leds bleues aux caractéristiques très similaires mais au format 5mm en lieu et place des modèles 3 mm d'origine. Si vous reproduisez ce genre de hack, la clé est de tout simplement acquérir ou trouver des leds avec un courant identique (« If » dans les docs) et une tension (« Vf ») avec une marge correspondante. On s'empresse ensuite de dessouder les deux autres leds et de tester l'ensemble sur la platine à essais avec les trois nouvelles leds en parallèle.



Comme l'essai sur platine est convainquant, il ne nous reste plus qu'à souder les nouvelles leds aux bons emplacements, en faisant très attention de respecter la polarité en suivant les pistes. Une fois l'ensemble en place, on remet la souris sous tension et on profite du spectacle. La seule partie difficile consiste à diriger la lumière dans molette de manière optimale. Ceci fait, il ne reste plus qu'à tout assembler.

Le résultat est assez sympathique même si, selon moi, cette seconde modification est moins plaisante que la première opérée sur mon premier exemplaire de la souris et agrémenté de leds rouges. C'est une simple question de goût. La lumière est dans tous les cas bien plus intense qu'avec la version originale et, au moins, ce n'est pas aussi neutre que du blanc. Un choix judicieux des leds permettra un rendement plus important et donc, à tension et courant égales, l'ensemble dégagera plus de lumière, mais coûtera sans doute plus cher qu'une fouille dans le « bocal à led ».

Et nous voici enfin propriétaire d'une souris unique qui, disons-le franchement, a carrément plus de classe que celle du fabricant. Le seul problème viendra de votre entourage (encore). En effet, vous serez sans doute bien incapable de vous retenir de signaler que c'est là l'un de vos hacks. En conséquence de quoi vous aurez droit à : « Tu m'en fais une rose ? », « Moi j'en veux une violette », « Moi une jaune ! »... jusqu'au fatidique « On peut changer la couleur ? ». Et là, vous n'aurez d'autre choix que de commencer à penser à la manière d'intégrer dans la souris un gadget lumineux RVB USB pilotable par PC et un mini hub... **DB**

7ème Forum International de la Cybersécurité

20 ET 21 JANVIER 2015

LILLE GRAND PALAIS

PLUS D'INFORMATIONS SUR

www.forum-fic.com

L'événement européen de référence sur la cybersécurité

FIC 2015

Cybersécurité et Transformation Numérique

contact@forum-fic.com

Un évènement co-financé par Région Nord-Pas de Calais et organisé par