N°12 | MAI - JUIN 2016

HACKABLE MAGAZINE

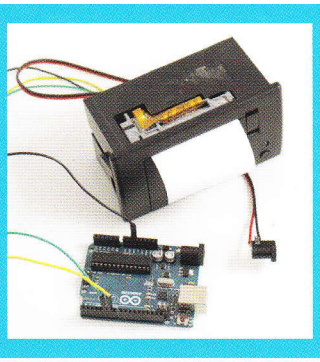
DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

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

ARDUINO

Imprimez facilement textes, code-barres et images sur une mini imprimante thermique

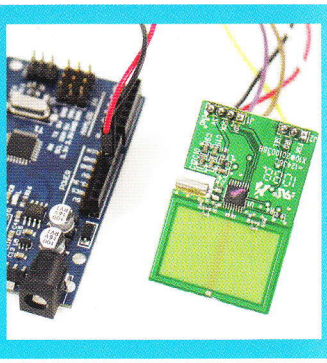
p. 04



RADIO / HACK

Analysez les signaux d'une télécommande 433 Mhz et reproduisez son fonctionnement

p. 52



RASPBERRY PI

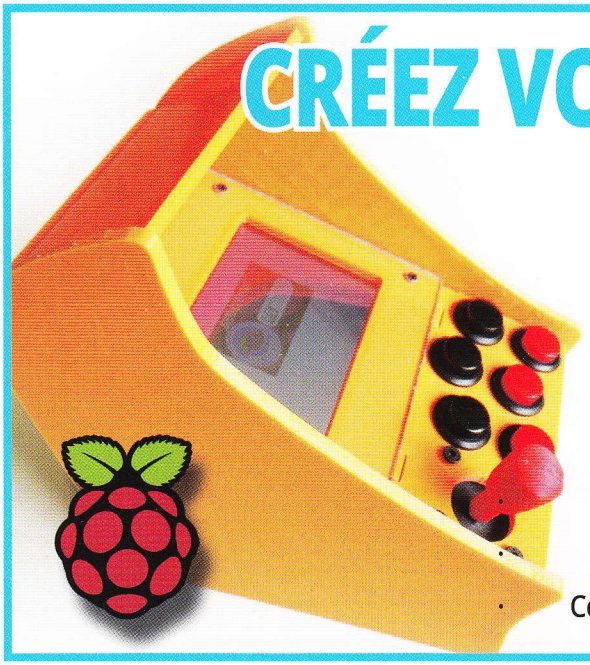
Mettez à jour votre système sans connexion internet grâce à une simple clé USB

p. 88

PROGRAMMATION

Utilisez judicieusement les constantes et macros dans vos croquis Arduino

p. 84



CRÉEZ VOTRE BORNE D'ARCADE !

... miniature à base de Raspberry Pi

p. 28

Construisez le boîtier

Ajoutez un mini écran LCD

Intégrez joystick et boutons

Configurez le système et les émulateurs

ARDUINO

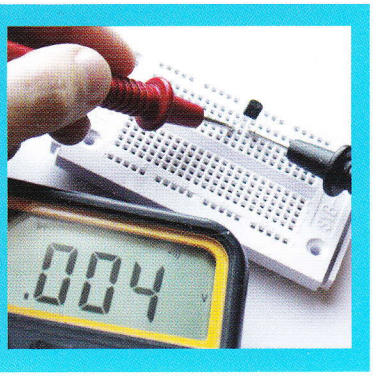
Générez, produisez et imprimez des QR codes avec une carte Arduino UNO

p. 16

COMPOSANTS

Comprenez le fonctionnement et l'utilisation des indispensables MOSFET

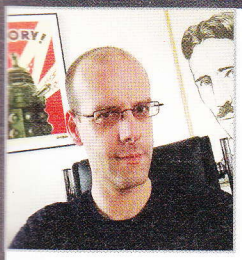
p. 70



L 19338 - 12 - F: 7,90 € - RD


NUIT DU HACK XIV
2/3 JUILLET 2016
HOTEL NEW YORK
CONVENTION CENTER
DISNEYLAND PARIS
WWW.NUITDUHACK.COM

ÉDITO



Jouer peut prendre bien des formes !

Il est même parfois possible de combiner plusieurs types de jeux en une seule activité ou une série d'activités. Et c'est exactement ce que vous propose de faire l'article principal de ce numéro. En effet, au-delà de l'objet qu'est une mini borne d'arcade, offrant grâce à la Raspberry Pi un nombre incroyable de jeux, nous avons l'aspect ludique de la construction et de la configuration du système, l'article couvrant la réalisation dans son ensemble.

Mais on peut également voir cela dans le sens inverse. Personnellement, je ne compte plus les projets entamés et aboutis qui, une fois fonctionnels, voient leur avenir se résumer soit à un fonctionnement sans aucune intervention, soit pire encore, à un abandon dans un coin jusqu'à finir recyclé pour un autre projet. Il est souvent très amusant de conduire un projet à terme, en « butant » sur des problèmes, des imprévus et en trouvant pour chaque déconvenue une solution rapide, efficace et tantôt élégante. Mais une fois l'ensemble fonctionnel et utilisable, la magie semble se dissiper et on part souvent en quête d'une autre aventure technique, et de sa prochaine dose de dopamine.

Avec une borne d'arcade, ou toute autre réalisation à destination ludique, les choses sont différentes et la magie reste entière, car après des heures de construction viennent les heures de jeu...

Mais, à bien y regarder, tout ce numéro est, je trouve, bourré de jeux : analyser et reproduire le fonctionnement d'un émetteur « domotique », s'amuser à imprimer avec une carte Arduino, pousser l'Arduino UNO dans ses extrêmes en générant des QR codes, explorer le monde des MOSFET, garder sa Raspberry Pi à jour sans Internet... Autant d'activités qui permettent de s'amuser et se divertir... et donc de jouer !

Sans plus attendre, je vous laisse découvrir tout cela par vous-même dans les pages qui suivent et vous souhaite, ce faisant, d'excellentes parties !

Denis Bodor

Hackable Magazine
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIÈRE GRISE

10, Place de la Cathédrale - 68000 Colmar
Tél. : 03 67 10 00 20 – Fax : 03 67 10 00 21
E-mail : lecteurs@hackable.fr
Service commercial : cial@ed-diamond.com
Sites : www.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Kathrin Scali
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12

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

Suivez-nous sur Twitter
@hackablemag

À PROPOS DE HACKABLE...

HACKS, HACKERS & HACKABLE

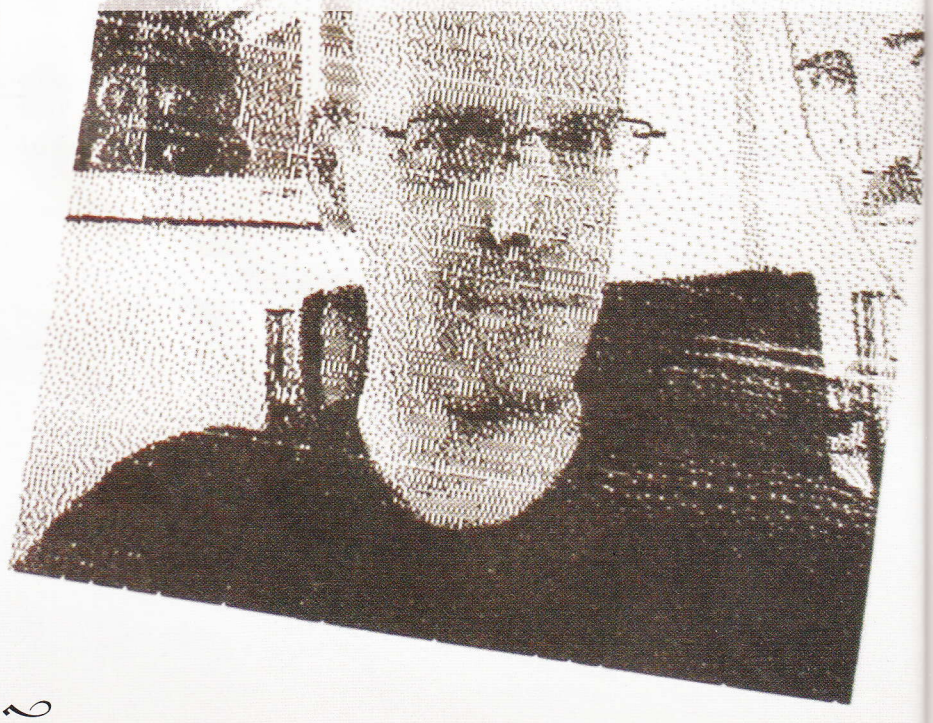
Ce magazine ne traite pas de piratage. Un **hack** est une solution rapide et bricolée pour régler un problème, tantôt élégante, tantôt brouillonne, mais systématiquement créative. Les personnes utilisant ce type de techniques sont appelées **hackers**, quel que soit le domaine technologique. C'est un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

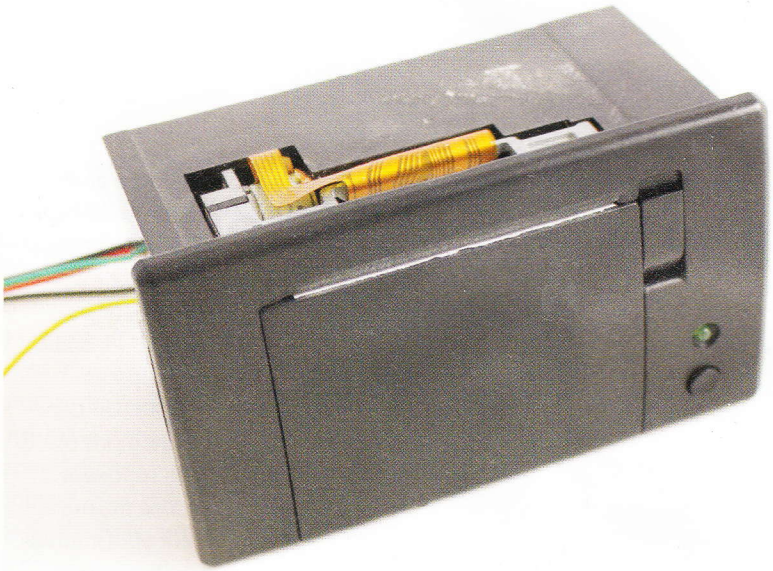
SOMMAIRE

CONTRÔLEZ UNE IMPRIMANTE THERMIQUE AVEC UNE CARTE ARDUINO

Denis Bodor

Ce ne sera sans doute pas une surprise si je vous dis que l'impression de documents est une activité en perte de vitesse. Le fait de transcrire une information sur un support papier présente cependant un certain nombre d'avantages face au « tout électronique ». Il peut donc être très intéressant de fournir à un projet une certaine capacité d'impression et c'est exactement le propos de cet article...





Il existe de nombreux modèles disponibles dans les boutiques en ligne et sur les sites d'enchères et de ventes directes. Celui-ci est physiquement très similaire au produit vendu par Adafruit et, dans les grandes lignes, s'avère relativement compatible.

Ce n'est ni le lieu ni le moment de nous lancer dans une dissertation sur l'intérêt du papier, de l'impression et les avantages de la « matérialisation » de documents et d'informations. Le fait est, tout simplement, que le papier et les informations imprimées sont toujours très présents sous bien des formes dans notre quotidien. Le pourquoi tient généralement en trois points : le coût, l'universalité et la durée de rétention de l'information. En d'autres termes, un écrit sur un bout de papier, c'est pas cher, ça ne demande rien d'autre que des yeux (et un cerveau) pour être utilisé et ça reste longtemps lisible.

Tout le monde parle du livre et de la presse mais, à bien y regarder, l'impression est partout, des titres de transport aux tickets de caisse en passant par les reçus bancaires, les additions de restaurants, les billets de spectacles, etc. Le média est encore tellement présent qu'il existe même des formes d'impressions produites par des machines pour des machines, nous excluant totalement, nous pauvres humains, des échanges par papier interposé (code-barre, QR code, etc.).

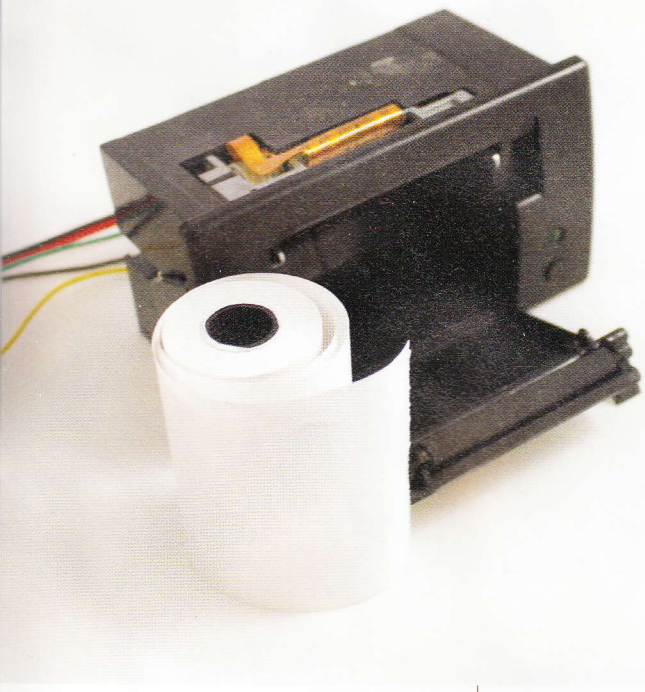
Les techniques et méthodes d'impression sont diverses et nombreuses, mais lorsqu'il s'agit d'imprimer une information sur un support de petite taille, ayant une durée de vie limitée (quelques mois), une technique est plus utilisée que les autres : l'impression sur papier thermique.

Cette technique d'impression n'utilise pas d'encre ou de pigment qui serait déposé sur le papier, mais un support spécial appelé « papier thermique ». Celui-ci est constitué de deux couches de produits chimiques, déposés sur un support de fibres de cellulose (du papier). Nous avons d'une part un leuco-colorant pouvant avoir deux formes, une colorée et une transparente, ainsi qu'une couche de révélateur. Lorsqu'on applique une source de chaleur suffisante, le révélateur devient liquide et active le leuco-colorant qui passe alors de l'état transparent à coloré (noir).

Une imprimante thermique se résume alors à un dispositif consistant à chauffer avec précision des points sur le papier thermique afin d'un dessiner un motif, des caractères, des lignes, des points, etc.

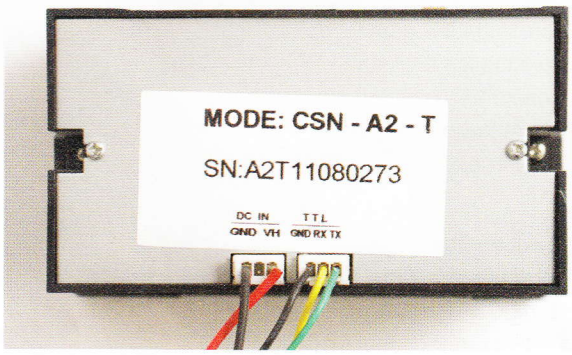
L'avantage de cette technique d'impression est évident : il n'y a qu'un seul consommable. Contrairement à l'impression à jet d'encre, matricielle ou laser qui nécessite du papier et un pigment sous une forme quelconque (liquide, ruban imprégné, poudre/toner), ici nous n'avons que le papier thermique. Le périphérique d'impression lui-même s'en trouve également simplifié puisque les éléments mécaniques sont réduits au minimum. La tête d'impression faisant généralement la largeur du papier, la seule partie mobile est celle nécessaire à l'entraînement du support.

Le contrecoup cependant est également facile à deviner : on ne peut pas imprimer sur n'importe quel papier. Les télécopieurs (fax) utilisaient initialement du papier thermique, mais ceux-ci ont, il y a



Le seul consommable de ce type d'imprimante est le papier thermique. Il n'y a pas d'encre ou de toner à ajouter, mais l'impression ne peut se faire que sur un papier spécial, heureusement facile à trouver en papeterie par exemple.

Quel que soit le modèle, le matériel est généralement livré avec sa connectique et un système de fixation. À gauche, nous avons l'alimentation et à droite l'interface permettant de piloter et contrôler l'imprimante via une liaison série à 9600 bps ou 19200 bps.



1. CHOIX DE L'IMPRIMANTE

Il existe bien des types d'imprimantes thermiques vendues sur des boutiques en ligne comme Adafruit (ou ses revendeurs). Des sites d'enchères en ligne comme eBay ou de ventes directes comme AliExpress, permettent également d'obtenir ce genre de matériel à moindre coût.

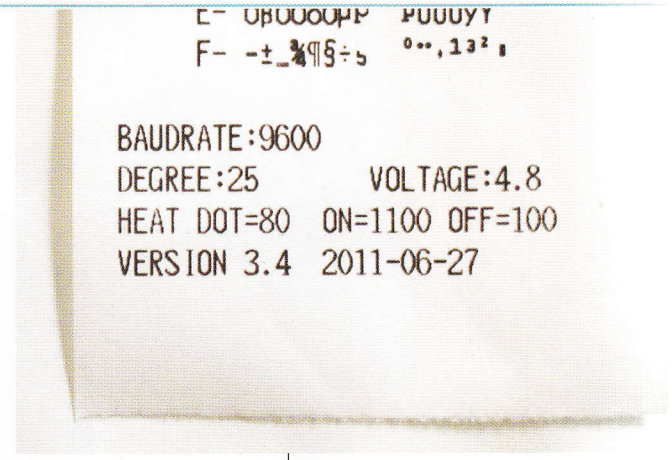
Le périphérique choisi sera généralement de petite taille et compatible avec des rouleaux de papier thermique standards de 57 mm de largeur (laise) et de 39 mm de diamètre. Ce genre de consommables se trouve facilement dans les papeteries et autres magasins vendant du matériel de bureau.

La partie critique dans le choix de l'imprimante concerne, bien entendu, son plusieurs années, évolué pour basculer sur l'impression laser ou jet d'encre, afin de réduire les coûts des consommables. Un certain nombre d'activités reposent encore massivement sur l'impression thermique, en particulier pour les petits formats (étiquettes, tickets, reçus, etc.).

interface. De plus en plus de modèles utilisent une connectique USB. Certains proposent également un port parallèle (comme sur les anciennes imprimantes PC) ou encore du Bluetooth (pour les smartphones et tablettes Android). Le modèle à utiliser ici pour une connexion à une carte Arduino nécessite une interface série TTL (0-5V). Le modèle vendu par Adafruit, prenant la forme d'un pavé noir, se retrouve sous diverses désignations et références sur le Web, tout en étant facilement reconnaissable.

En terme de connectique, il est livré avec un ensemble de câbles permettant l'alimentation et l'échange de données :

- rouge : alimentation 5V (5-9V pour certains modèles, lisez bien le descriptif du produit). La tension n'est pas un problème puisque des « blocs secteur » de ce type sont monnaie courante, mais il n'en va pas de même pour le courant qui peut monter jusqu'à 2 ampères (voir plus selon la configuration) lors de l'impression. En cas de problème, la première chose à vérifier sera donc la capacité de l'alimentation à fournir suffisamment de courant. Il est, bien entendu, hors de question ici d'alimenter l'imprimante via la broche « 5V » d'une carte Arduino.
- noir : la masse, commune au périphérique, à l'alimentation et la carte Arduino. Cette imprimante dispose de deux fils noirs de masse, un à relier au bloc d'alimentation et l'autre à l'Arduino.



En maintenant enfoncé le bouton en façade de l'imprimante lors de la connexion de l'alimentation, un test d'impression est déclenché. Ceci permet non seulement de valider la puissance de l'alimentation utilisée, mais apporte également diverses informations utiles sur le matériel, dont la vitesse de communication. Ici, « BAUDRATE:9600 ».

- vert : c'est le TX de l'imprimante ou en d'autres termes la ligne lui permettant d'émettre des données. Elle devra être reliée sur une broche RX côté Arduino. Attention, encore une fois lisez bien la description de l'objet que vous commandez, la plupart de ces imprimantes utilisent des niveaux de tensions dits TTL, mais certaines utilisent le standard RS232 (+/-12V) incompatible avec une carte Arduino.
- jaune : la ligne RX permet à l'imprimante de recevoir des données, qu'il s'agisse des informations à directement imprimer ou des paramètres de configuration. Cette ligne est reliée sur la broche TX côté Arduino.

Il est possible de rapidement tester le bon fonctionnement de l'imprimante, et donc de l'alimentation, en provoquant un test automatique. Le périphérique dispose d'un bouton permettant de faire avancer le papier (*feed*). Si celui-ci est maintenu enfoncé tout en connectant l'alimentation, une impression de test est provoquée, résumant les caractères imprimables ainsi que les informations sur le matériel. C'est aussi de cette manière que vous obtiendrez un paramètre très important pour la mise en œuvre du périphérique : la vitesse de communication série qui peut être de 9600 bps ou 19200 bps selon le modèle et le firmware utilisés par le produit.

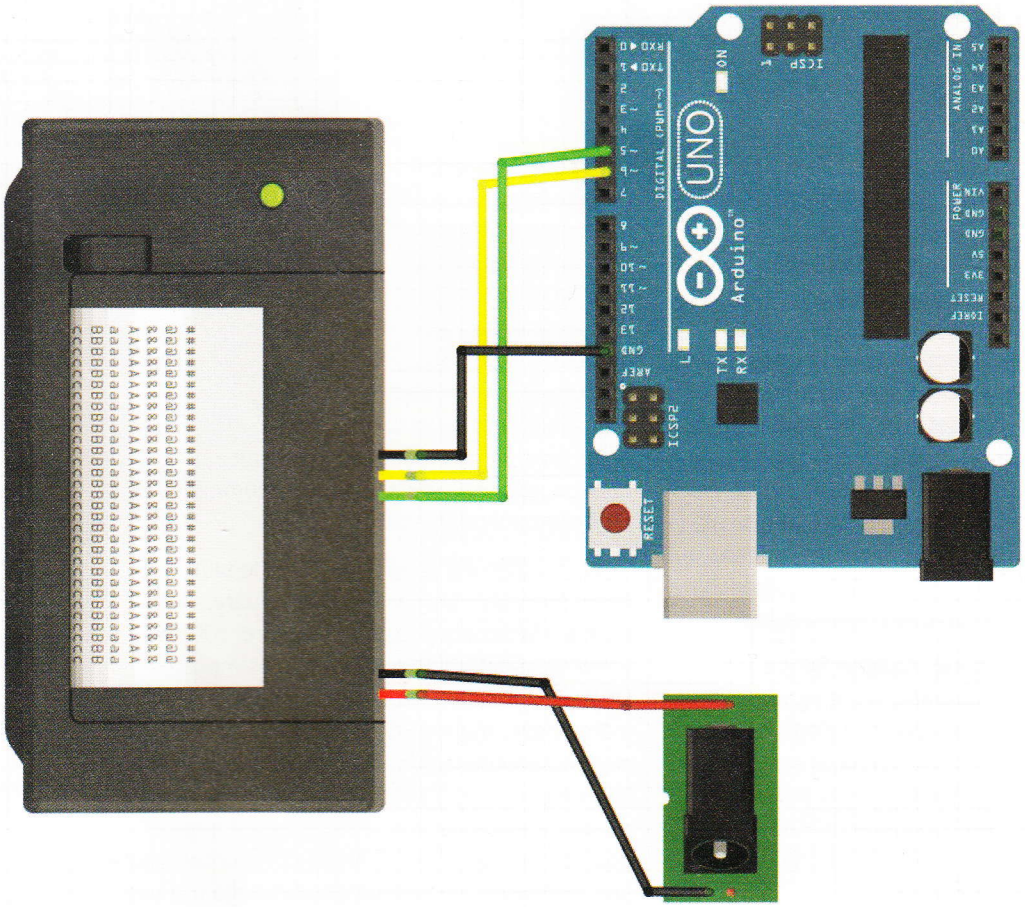
Ce type d'imprimante vous coûtera dans les 40€ même s'il est possible en fouillant des sites comme AliExpress d'en trouver à moins de 25€, mais avec un descriptif des plus sommaires et donc énormément de doutes quant aux caractéristiques du produit. Si vous voulez une solution fiable et sans surprise, mieux vaut opter pour un matériel plus cher, mais réputé comme celui d'Adafruit (ou vendu comme étant explicitement compatible Arduino).

Ce modèle est sans doute celui le plus facile (et le plus sûr) à mettre en œuvre, mais il est parfaitement possible d'explorer d'autres voies. Il semblerait, en effet, que les modèles USB et Bluetooth reposent peu ou prou sur la même base, mais en étant équipés d'interfaces supplémentaires, elles-mêmes connectées en interne sur un port série TTL. Il peut donc être envisageable d'accéder à l'interface d'origine pour la connexion avec l'Arduino, mais cela reste une probabilité et quelque chose que je n'ai pas (encore) personnellement testé.

Ce qu'il conviendra d'éviter, en revanche, c'est le recyclage d'un système d'impression provenant d'un vieux terminal de paiement (TPE) ou d'une caisse enregistreuse. Dans tous les périphériques de ce type que j'ai eu le loisir de démonter, la tête d'impression de l'imprimante était directement pilotée par un circuit intégré prenant également en charge le reste du périphérique (bouton, écran, etc.). À cœur vaillant rien d'impossible, certes, mais réutiliser directement la tête d'impression sans interface est clairement une tâche complexe et coûteuse en temps et en expérimentations, et à l'issue incertaine.

2. LE MONTAGE

Connecter l'imprimante à une carte Arduino est d'une extrême simplicité. Ici, la carte UNO ne disposant que d'un seul port série déjà utilisé par le moniteur, le dialogue avec l'imprimante devra se faire via des broches d'entrée/sortie standards. La bibliothèque « SoftwareSerial », déjà



incluse à l'environnement, nous permettra donc d'utiliser n'importe quelles broches pour l'imprimante comme un port série.

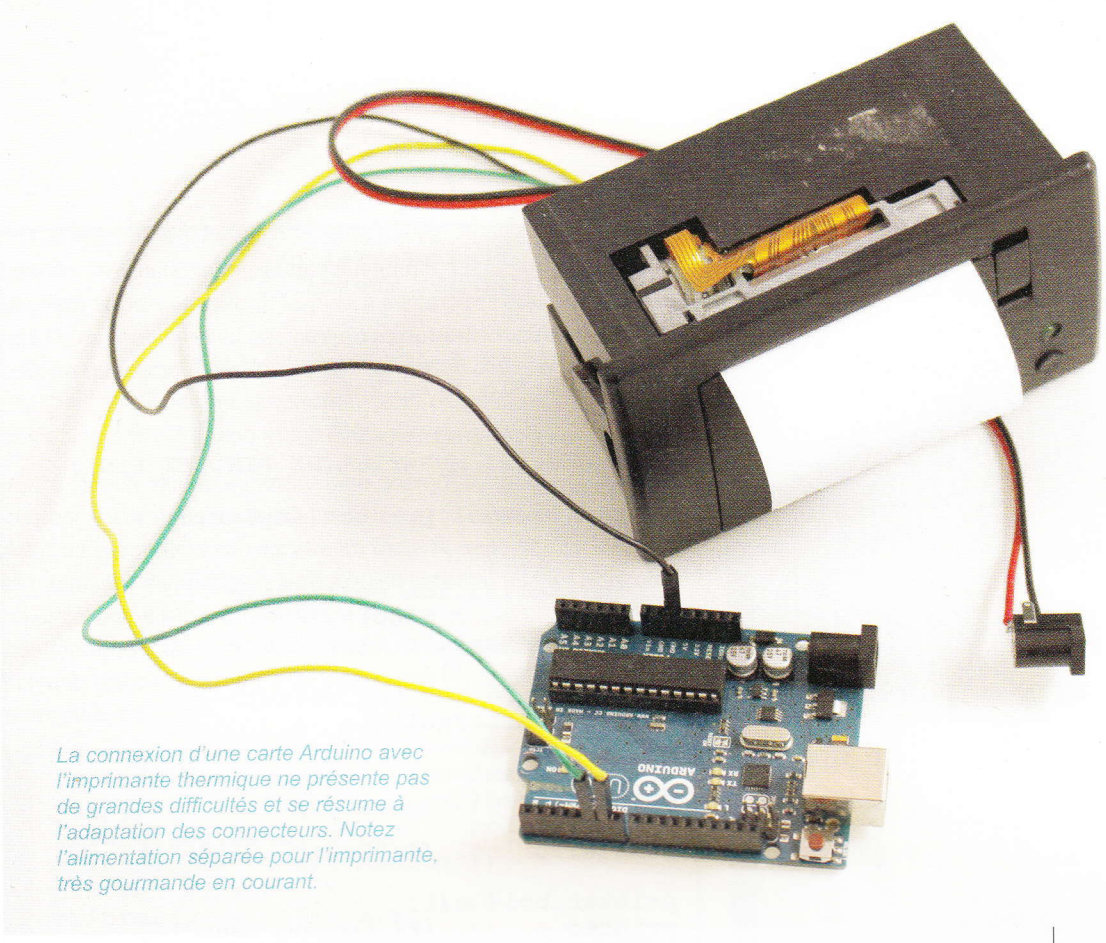
Le choix, totalement arbitraire, sera ici d'utiliser la broche 6 pour l'envoi de données (fil jaune) et la broche 5 pour la réception (vert). La masse du connecteur marqué « TTL » sera connectée à l'une des masses présentes sur la carte Arduino. Remarquez que dans la quasi-totalité des cas, quelle que soit la provenance de l'imprimante, il est fort probable qu'un léger bricolage soit nécessaire. En effet, le câblage livré avec l'imprimante et plus exactement les connecteurs, ne correspondent pas aux besoins des cartes Arduino, pas plus d'ailleurs qu'à ceux du connecteur d'alimentation. Soudures, dominos, connexions sauvages... il vous faudra adapter la connectique à votre besoin d'une manière ou d'une autre.

Notez enfin que, dans le cas d'une utilisation d'une carte comme une Arduino Mega2560 ou une Leonardo, vous disposez de plus d'un port série physique. Il n'y a cependant pas d'impératif à ce niveau, car la vitesse de communication restant relativement faible (19200 ou 9600 bps), la bibliothèque « SoftwareSerial » s'en sort parfaitement bien. Le fait d'utiliser un port série physique n'apporte donc pas grand-chose si ce n'est une économie substantielle de mémoire, tantôt souhaitable (voir article suivant sur la génération et l'impression de QR codes).

3. PREMIER ESSAI

L'imprimante communique via le port série en distinguant deux types de messages :

- d'une part, les données brutes qui sont utilisées pour être imprimées. Ces lignes se terminant par un caractère



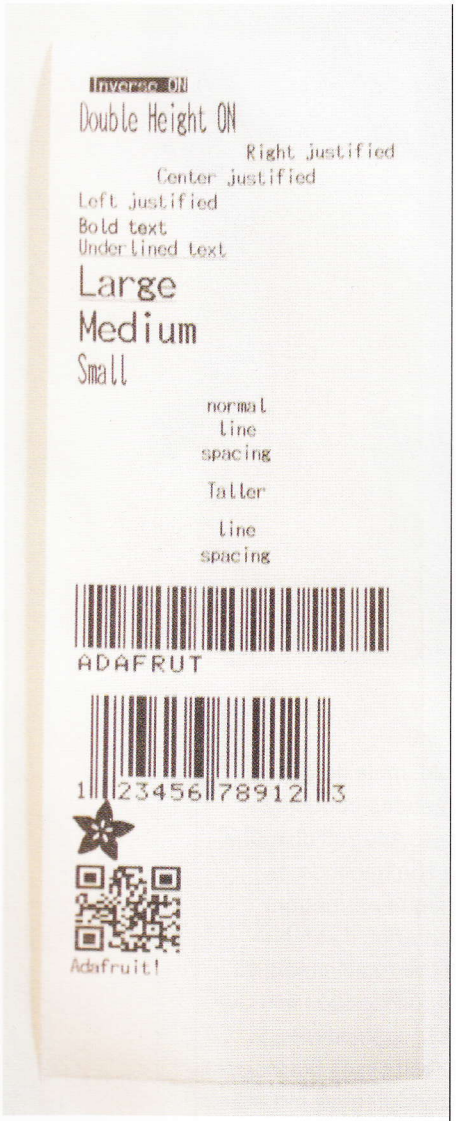
La connexion d'une carte Arduino avec l'imprimante thermique ne présente pas de grandes difficultés et se résume à l'adaptation des connecteurs. Notez l'alimentation séparée pour l'imprimante, très gourmande en courant.

LF (*Line Feed*, `0x0A` dans la table ASCII) sont donc du texte à imprimer.

- d'autre part, des commandes de configuration spécifiques interprétées par l'imprimante et permettant de formater du texte, d'en changer l'aspect (gras, taille, etc.), d'imprimer des données particulières comme des codes barres ou des images, et de configurer le comportement même du système d'impression (temps de chauffe, vitesse d'impression, etc.). Ces messages de configuration commencent tous avec un caractère particulier non imprimable (`0x1B`, `0x1D` ou `0x12` correspondant respectivement aux caractères « ESC », « GS » et « DC2 »).

Le protocole est relativement simple, mais demande une lecture complète du jeu de commandes et de multiples essais. Heureusement pour nous, Adafruit distribue ce type de périphérique et, comme à son habitude, la société met à disposition une bibliothèque permettant de facilement prendre en charge le matériel. Il semblerait que le produit vendu par Adafruit soit relativement courant et, ce faisant, que la bibliothèque fonctionne parfaitement avec des imprimantes à l'aspect identique, mais à moindre coût. Les deux modèles utilisés pour cet article ne proviennent pas de chez Adafruit, mais ont été achetés sur eBay il y a plusieurs années. En dehors de la vitesse de communication (9600 bps au lieu de 19200) et de quelques fonctionnalités absentes comme le contrôle de l'espacement de caractères, tout a fonctionné sans le moindre problème (il semble s'agir physiquement du même matériel, mais avec un firmware d'une version plus ancienne).

La bibliothèque est installable très facilement via le gestionnaire intégré dans l'environnement Arduino, sous le nom « Adafruit Thermal Printer Library » en version 1.1.0. Une fois installée, on pourra très rapidement procéder à un premier essai :



Le résultat du test d'impression de la bibliothèque « Adafruit Thermal Printer Library » présente les diverses fonctionnalités disponibles et permet également de valider la compatibilité de votre matériel.

```
#include "SoftwareSerial.h"
#include "Adafruit_Thermal.h"

// définition des broches
#define TX_PIN 6 // jaune
#define RX_PIN 5 // vert

// port série logiciel
SoftwareSerial mySerial(RX_PIN, TX_PIN);
// imprimante
Adafruit_Thermal printer(&mySerial);

void setup() {
  // initialisation port série
  mySerial.begin(9600);
  // initialisation imprimante
  printer.begin();
  // configuration par défaut
  printer.setDefault();

  // défilement une ligne
  printer.feed(1);
  // impression
  printer.println(F("Coucou Monde"));

  printer.boldOn();
  printer.println(F("Coucou GRAS"));
  printer.boldOff();

  printer.doubleWidthOn();
  printer.println(F("Double Largeur"));
  printer.doubleHeightOn();
  printer.println(F("Double double"));
  printer.doubleWidthOff();
  printer.println(F("Double Hauteur"));
  printer.doubleHeightOff();

  printer.justify('R');
  printer.println(F("Droite"));
  printer.justify('C');
  printer.println(F("Centre"));
  printer.justify('L');
  printer.println(F("Gauche"));

  // on regroupe quelques attributs
  printer.justify('C');
  printer.doubleHeightOn();
  printer.doubleWidthOn();
  printer.boldOn();
  printer.inverseOn();
  printer.println(F("Inverse GROS !"));
  printer.feed(4);
}

void loop() {
  // vide, on ne fait rien en boucle
}
```

Les commentaires dans le code devraient être suffisants pour détailler le fonctionnement de l'ensemble. La logique consiste à déclarer le port série logiciel puis à instancier l'objet représentant l'imprimante en lui passant en argument un pointeur vers le port en question. Dans le cas de l'utilisation d'un port matériel, `&mySerial` deviendra simplement `Serial1` ou `Serial2` dans le cas d'une carte Arduino Mega2560, par exemple.

Ce simple exemple montre déjà quelques caractéristiques intéressantes du matériel. Ceux livrés avec la bibliothèque Adafruit permettent de prendre en main le reste des fonctions et méthodes disponibles. Notez qu'en fonction de l'imprimante, de son origine et du firmware qui s'y trouve, tout ne fonctionnera pas forcément de la manière attendue. À titre d'exemple, le code suivant, tiré de l'exemple « A_printertest », ne fonctionne pas correctement :

```
printer.setSize('L');
printer.println(F("Large"));
printer.setSize('M');
printer.println(F("Medium"));
printer.setSize('S');
printer.println(F("Small"));
```

Dans le résultat, il n'y a pas de différences visuelles entre le texte en *Large* et en *Medium*, et *Small* n'est pas dans la taille par défaut. Il semblerait que les commandes de configuration ne soient pas toutes prises en compte par mon imprimante, ce qui provoque des problèmes d'interprétation (*Large* par exemple est souligné alors que le `printer.underlineOff()` présent juste avant ces lignes désactive normalement explicitement le soulignement). Il est donc important, après le test d'initialisation (bouton + alimentation) de charger le croquis exemple « A_printertest » afin de vérifier ce qui marche et ce qui ne marche pas avec votre matériel.

Exemple d'impression de codes barres dans différents formats, tel que produit par le croquis de démonstration livré avec la bibliothèque Adafruit. De façon générale ce type d'impression ne sera intéressant que pour une utilisation très spécifique.

4. IMPRESSION DE CODES BARRES

Les codes barres sont un moyen d'encoder une information sous la forme d'une série de barres verticales d'épaisseurs arbitrairement définies. Ceci permet, à l'aide d'un lecteur adapté (douchette ou smartphone) de récupérer l'information et d'accélérer sa saisie. Il existe bien des standards dans ce domaine et la plupart sont pris en charge par la bibliothèque Adafruit : UPC_A, UPC_E, EAN8, CODE39, I25, CODEBAR, CODE93, CODE128, CODE11, MSI et EAN13. Ce dernier, l'EAN13 (pour

Voici un exemple de code barre EAN13. Pour en vérifier la génération, il vous suffit de prendre un produit au hasard, utiliser les 13 chiffres s'y trouvant et comparer le résultat à l'original.

European Article Numbering à 13 chiffres), est le format que l'on trouve sur les produits distribués en Europe et permettant de leur attribuer un numéro unique.

La génération de codes barres est directement prise en charge par l'imprimante (et non par la bibliothèque) et là encore, en raison de variations de versions du firmware, il est possible de constater des différences de comportement d'un matériel à l'autre. Cependant, d'après nos essais, l'EAN13 semble parfaitement pris en charge avec le modèle testé, même si l'utilisation pratique reste très limitée. L'exemple « C_barcodes » de la bibliothèque vous permettra cependant de procéder à un test pour vous assurer du fonctionnement de l'imprimante pour quelques standards courants.

Une version plus moderne d'encodage sous la forme de codes barres, en deux dimensions cette fois, est le très populaire QR code, lisible par les smartphones et les tablettes. Ceci n'est pas pris en charge directement par l'imprimante mais, avec un peu de ténacité, peut être généré et imprimé avec un croquis Arduino (voir article suivant sur le sujet).

5. IMPRESSION D'IMAGES

Il est possible d'imprimer aussi bien du texte que des codes barres avec ce type d'imprimante, mais également des images en lui fournissant, tout simplement la couleur (noir ou blanc) des différents pixels qui la compose sous forme de données binaires. Gardez cependant à l'esprit qu'une imprimante thermique de ce type est destinée avant tout à imprimer des reçus et des tickets. Un télécopieur d'ancienne génération utilise également le système d'impression thermique, mais sa conception est sensiblement différente. La capacité d'impression d'images pour notre imprimante se limite normalement à un logo d'une société ou d'un commerçant, mais en aucun cas n'est prévue pour imprimer une énorme image.

Quelle que soit la qualité du papier utilisé ou la configuration en place, il est donc tout à fait normal que sur des images de plus d'une centaine de points de large (sur les 384 disponibles sur la largeur d'impression) et d'autant de points de haut, vous obteniez des zones mal imprimées ou un manque de précision. En effet, la tête d'impression étant constituée d'éléments chauffants, l'inertie thermique peut conduire à des « ratés » sous forme de « bavures » et autres zones partiellement imprimées.

La bibliothèque Adafruit fournit une méthode dédiée à l'impression des images, stockée en mémoire vive (SRAM), mais plus judicieusement en mémoire de programmation (flash). Bien que l'image soit monochrome, la densité d'impression (environ 200 points par pouce, soit 8 points par millimètre) provoque une consommation non négligeable de mémoire. À titre indicatif, une image carrée de 2 cm de côté sera composée de 160 points par 160 points, soit 25600 points en tout, ce qui occupera plus de 3 Ko de flash (sur les 32 Ko disponibles avec une carte Arduino UNO).

L'impression d'une image se fait très simplement avec par exemple :

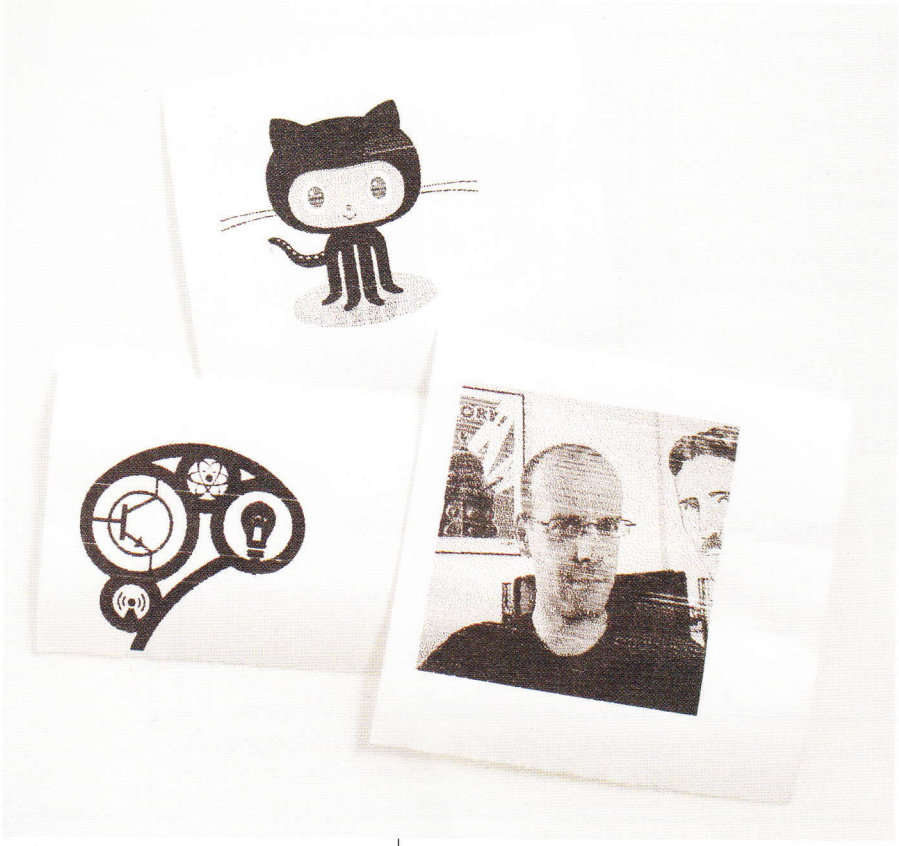
```
#include "Adafruit_Thermal.h"
#include "image.h"
#include "SoftwareSerial.h"

#define TX_PIN 6 // jaune
#define RX_PIN 5 // vert

SoftwareSerial mySerial(RX_PIN, TX_PIN);
Adafruit_Thermal printer(&mySerial);

void setup() {

  mySerial.begin(9600);
  printer.begin(225);
  printer.setDefault();

  // impression image
  printer.printBitmap(image_width, image_height, image_data);

  printer.feed(2);
}

void loop() {
}
```

`image` est un pointeur sur un tableau de `uint8_t` présent dans un fichier d'entête (`.h`) intégré dans votre projet :

```
#ifndef _image_h_
#define _image_h_

#define image_width 300
#define image_height 314
static const uint8_t PROGMEM image_data[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x80, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20,
    0x84, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00,
...
    0xff, 0xff, 0xff, 0x7f, 0xff, 0xff, 0xff, 0xff, 0x7f, 0xff, 0xff, 0xff,
    0xff, 0xf7, 0x7f, 0xff, 0xfb, 0xff, 0xff, 0xff, 0xff, 0xbf, 0xff, 0xff,
    0xff, 0xff, 0xbf, 0x0f };
#endif
```

Chaque octet de ce tableau représente 8 points pouvant être noirs (`1`) ou blanc (`0`). La dimension de l'image en hauteur et en largeur est spécifiée sous la forme des macros `image_height` et `image_width`. Les trois éléments (hauteur, largeur, pointeur) sont ensuite directement utilisés par la méthode `printBitmap()` qui en l'absence de quatrième argument considère que l'image est en mémoire de programmation (`PROGMEM`).

Vous l'avez compris, pour gérer ce type d'images, qui n'est qu'une succession de bits, il est avant tout nécessaire de convertir un format connu (JPEG, PNG, etc.) en une représentation dite *raw data* (données brutes).



L'impression d'images de taille importante n'est pas la spécialité de ce type de matériel et les résultats peuvent être très aléatoires. Il est préférable de considérer cette fonctionnalité comme un moyen d'ajouter des petites décorations (logos, icônes, symboles, etc.) accompagnant des données textuelles.

Il existe plusieurs solutions plus ou moins pratiques et/ou rapides pour cela, mais la plus simple consiste à reposer sur l'utilisation d'un croquis Processing livré avec les fichiers de la bibliothèque Adafruit. Vous devrez donc installer Processing puis ouvrir le fichier `bitmapImageConvert.pde` pour la conversion. Là, lors de l'exécution du croquis, une boite de dialogue vous demandera de choisir un fichier graphique et dès validation, celui-ci sera converti en fichier `.h`, sous le même nom et dans le même dossier que le fichier d'origine. Il vous suffira alors, dans l'environnement Arduino, d'ajouter ce fichier au projet via le menu ***Croquis***, ***Ajouter un fichier...***, puis d'ajouter une ligne `#include` dans votre code.

Pour des résultats optimaux, l'image de départ doit être en noir et blanc (pas de couleur ou de niveau de gris) et, idéalement, d'une taille multiple de 8 points, bien entendu inférieure aux 384 points maximums de l'imprimante en largeur. La conversion d'une image couleur en noir et blanc pourra se faire très simplement avec un outil de retouche graphique comme GIMP (disponible sous Windows, Mac OS X et GNU/Linux).

CONCLUSION

Les idées de réalisation, dès lors qu'on peut ainsi transférer une information sur un support papier, ne manquent pas. Liste de choses à faire, résumé des tweets, prévisions météo, tickets, reçus... la liste n'en finit pas. Mieux encore, il est parfaitement possible de mixer plusieurs projets. Vous vous souvenez de la « serrure » à tag RFID du numéro précédent ou de nos expérimentations NFC dans *Hackable n°10* ? Imaginez alors ce qu'il est possible de faire en associant un tel « marqueur » et une solution d'impression comme celle-ci... De quoi très facilement, par exemple, envisager des applications pratiques pour un marathon, un parcours d'orientation ou encore du géocaching ! **DB**

ACTUELLEMENT DISPONIBLE

OPEN SILICIUM N°18 !

CONCEVEZ VOS OBJETS CONNECTÉS

...AVEC DES LOGICIELS LIBRES !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

GÉNÉREZ ET IMPRIMEZ DES QR CODES AVEC UNE CARTE ARDUINO

Denis Bodor

Les QR codes ou codes QR sont maintenant omniprésents dans notre quotidien. On les retrouve des affiches publicitaires aux catalogues, en passant par les titres de transport, les livres, les autocollants, les véhicules... Dès lors qu'il s'agit de fournir une longue chaîne de caractères, comme l'URL d'une page web, d'un média imprimé à un périphérique électronique, le QR code est LA solution. Une modeste carte Arduino est en mesure de générer un tel code, même si cela demande quelques efforts...

Précisons immédiatement que ce dont il sera question dans cet article n'est pas la simple impression d'un QR code déjà existant dans la mémoire du microcontrôleur. Il n'est pas question de simplement imprimer une image produite par ailleurs. Ceci est bien trop facile et surtout n'est pas vraiment utile. Non, ce qui nous intéresse est de **générer** un QR code, et donc, à partir d'une information dynamiquement produite par la carte Arduino, de créer l'image correspondante.

L'exercice est plus délicat qu'il n'y paraît, car produire une telle image demande une quantité non négligeable de mémoire, or une carte comme une Arduino UNO et son microcontrôleur ATmega328 ne dispose que de 32 Ko de flash pour les programmes et de 2 Ko de mémoire vive (SRAM). Il serait illusoire d'espérer générer n'importe quel type de QR code (voir plus loin) et nous serons obligé de limiter nos attentes ainsi que les possibilités du croquis final.

Notre objectif de démonstration consistera à réutiliser du code existant, l'adapter à la plateforme Arduino puis à écrire un croquis capable d'accepter une chaîne de caractères via le moniteur série pour ensuite imprimer le QR code correspondant sur une imprimante thermique série. Ceci ajoute une difficulté supplémentaire car, bien que les images imprimables soient monochromes et donc composées d'un ensemble de pixels n'occupant qu'un bit, nous devons produire un QR code suffisamment grand pour pouvoir être lu par un smartphone. Comme l'image est composée dynamiquement, tout ceci doit prendre place en mémoire vive, dont nous n'avons que 2048 octets !

L'image initialement générée avec un QR code en version 9 ne fait que 53 pixels de côté. L'impression sans mise à l'échelle est bien plus simple, mais le résultat est tout simplement inutilisable, car faisant un malheureux centimètre de haut et de large, parfaitement illisible avec un smartphone.

La solution devra donc reposer sur une astuce consistant à imprimer non pas une image, mais une succession de lignes d'un seul pixel de haut. Nous limitons ainsi la taille de l'image et l'espace mémoire sera le même pour chaque ligne, réinitialisé et réutilisé.

1. À PROPOS DES QR CODES

Les QR codes sont une forme de codes barres en deux dimensions normalisés (ISO/CEI 18004) et librement utilisables. Contrairement aux codes barres unidimensionnels tels que ceux que l'on trouve sur les emballages de produits (norme EAN-13), les QR codes permettent de stocker une quantité importante d'informations. En contrepartie de cet avantage, on trouve le fait de devoir disposer d'un périphérique plus complexe pour lire et décoder l'image, mais la popularisation des smartphones et des tablettes ces dernières années fait que (presque) tout le monde dispose aujourd'hui d'un lecteur adapté.

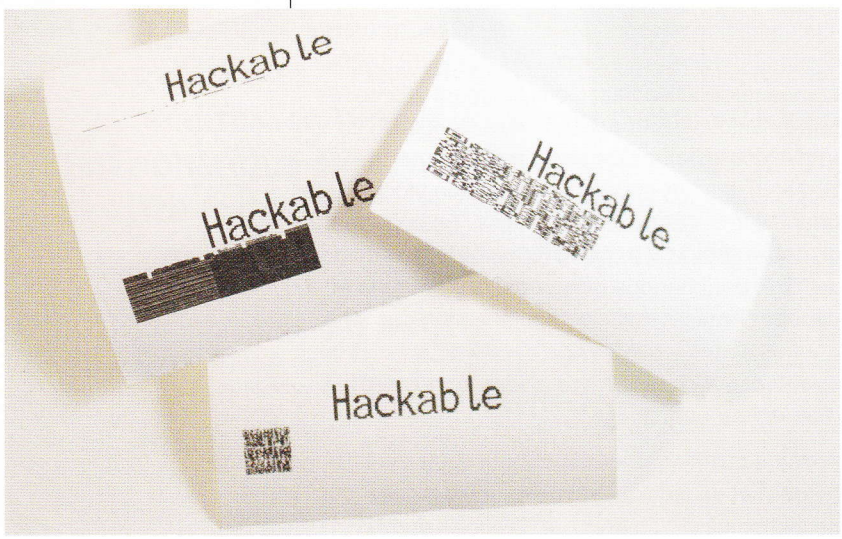
Alors qu'un code-barre comme EAN-13 ne permet de stocker qu'une quantité limitée de symboles (13 chiffres), le QR code permet

d'encoder des informations plus conséquentes et de déclencher des actions en fonction de ces informations :

- une URL ou adresse web permet de pointer automatiquement un navigateur sur une page précise ;
- des coordonnées géographiques immédiatement positionnables sur une carte ;
- des informations de connexion (SSID et mot de passe) pour se connecter facilement à un point d'accès Wifi ;
- un numéro de téléphone pour déclencher un appel ou l'envoi d'un SMS/MMS ;
- les coordonnées d'une personne (VCard) afin d'enregistrer rapidement un nouveau contact ;
- etc.

QR code n'est pas le seul standard existant, mais celui-ci a su s'imposer face à la concurrence (Datamatrix/Flashcode, par exemple) auprès du grand public. Le principe de fonctionnement, exposé sommairement, est relativement simple. Il s'agit d'utiliser une information textuelle et de l'encoder sous la forme d'un tableau de points, noirs ou blancs. Pour faciliter le décodage et la lecture, des repères sont ajoutés ainsi que des pixels complémentaires permettant l'utilisation d'un mécanisme de correction d'erreur (code de Reed-Solomon).

Avant d'arriver à un résultat utilisable et surtout lisible avec un smartphone, un certain nombre d'essais ont été nécessaires. Quel que soit le projet, il est rare que tout fonctionne du premier coup...



En effet, un QR code est destiné à être imprimé sur un support susceptible d'être endommagé. La norme prévoit donc qu'une partie plus ou moins importante de l'image puisse être illisible ou comporte des erreurs, sans que l'intégrité de l'information encodée ne soit remise en cause. Ce mécanisme utilise un système reposant sur la redondance de l'information. C'est un élément que nous devons prendre en compte pour notre projet, car plus il y a de redondance, moins nous avons de place pour l'information. Ou en d'autres termes, pour une information donnée, plus le QR code est résistant, plus l'image devra avoir de pixels.

On distingue quatre niveaux de correction d'erreurs :

- L : environ 7% de redondance dans l'information encodée ;
- M : environ 15% ;
- Q : environ 25% ;
- H : environ 30%.

En plus de ce mécanisme de correction, il faut également distinguer des « versions ». Le terme est trompeur, car il n'a rien à voir avec la notion de « versions » pour un logiciel. Il s'agit plutôt de déclinaisons du standard concernant la configuration et le nombre de modules (pixels) qui composent un QR code. Il y a 40 versions utilisables en fonction de la taille de l'information à encoder,

tout en prenant en compte le niveau de correction d'erreur souhaité. Chaque version peut, en effet, être déclinée en correction L, M, Q ou H.

Ainsi pour un QR code en version 1, de 21×21 pixels/modules, on peut encoder respectivement 25 (L), 20 (M), 16 (Q) ou 10 (H) caractères alphanumériques. En version 40, ces nombres augmentent jusqu'à 4296, 3391, 2420 et 1852 caractères. Nous avons donc 160 combinaisons possibles et autant de capacités différentes de stockage pour des QR codes allant de 21×21 à 177×177 pixels/modules de côté. Vous pouvez retrouver un tableau récapitulatif complet sur **http://www.qrcode.com/en/about/version.html** (le site de Denso Wave, inventeur du QR code).

Dans des conditions optimales, la logique de création d'un QR code commence par la sélection d'un niveau de correction d'erreur dépendant des conditions d'utilisation du code ou de la fragilité de son support. C'est ensuite en fonction du type et de la taille des données à encoder qu'on choisit la version à utiliser pour obtenir un QR code d'une taille optimale et adaptée.

Ceci sous-entend donc qu'un générateur de QR code puisse gérer les 40 versions utilisables. Lorsqu'on dispose de mémoire à profusion comme c'est le cas sur PC, un smartphone ou une Raspberry Pi, ceci n'est pas un problème. Mais dans le cas d'une carte Arduino Uno avec ses 2048 malheureux octets de SRAM, c'est une tout autre affaire, en particulier lorsqu'on utilise déjà une partie de cette mémoire pour un port série logiciel (bibliothèque *SoftwareSerial*) et le support de l'imprimante thermique. Un rapide calcul nous montre clairement nos limites absolues. Si nous prenons simplement un QR code en version 40, la taille des données encodables est déjà deux fois supérieure à celle de la mémoire vive, tout comme celle de l'image qui en résulterait (177*177/8, soit environ 3916 octets). Et comme nous souhaitons générer dynamiquement nos QR codes, il n'est pas question d'utiliser la mémoire flash comme c'est le cas généralement avec les chaînes de caractères immuables (constantes).

Le choix d'une version et d'un niveau de correction cependant n'est limitatif que concernant le nombre **maximum** de caractères qu'il est possible d'encoder, et non le minimum. Autrement dit, en choisissant, par exemple, une version 9 et un niveau de correction M nous pouvons encoder 262 caractères, mais également moins. Certes, le QR code résultant sera plus grand que nécessaire, mais nous n'avons plus qu'une seule déclinaison à gérer et non 160.

2. RÉUTILISER UN CODE EXISTANT

Implémenter et programmer un générateur de QR code à partir de rien n'est pas une option envisageable ici. Ceci supposerait en effet, de lire et d'assimiler l'ensemble de la norme et d'écrire un croquis en conséquence tout en gardant à l'esprit les limitations propres à la plateforme Arduino. Traiter un tel sujet occuperait au moins l'ensemble des pages de ce numéro, sinon plus (sans compter les migraines pour vous comme pour moi).

Il existe un certain nombre de bibliothèques Arduino permettant la génération de QR codes, mais celles-ci se limitent à la version 1 permettant donc de stocker que jusqu'à 25 caractères alphanumériques. Ceci est bien en dessous de nos attentes, car l'idée est, en premier lieu, d'être relativement générique et d'encoder, par exemple, des URL (avec une limite de 255 caractères pour un maximum de libertés).

Heureusement pour nous, Tom Zerucha (alias tz) a développé un code très compact permettant de générer des QR codes dans 40 versions et avec les quatre niveaux de correction décrits dans la norme, et l'a mis à disposition sous licence GNU GPL v3, sur GitHub : **https://github.com/tz1/qrduino**). Cependant, contrairement à ce que le

Contrairement à d'autres projets, ici chaque tentative rime avec « utilisation de consommables ». Mieux vaut donc prévoir un stock raisonnable pour ne pas se retrouver « coincé » faute de papier...

nom « qrduino » laisse entendre, il ne s'agit pas à proprement parler de croquis Arduino, mais plus exactement d'un code C compilable sur PC (GNU/Linux), à destination du PC lui-même ou d'un microcontrôleur Atmel AVR. L'AVR est le composant au cœur des Arduino, mais il ne s'agit pas du tout du même environnement de programmation.

L'exercice consistera donc à utiliser le code de ce développeur et à l'intégrer dans un projet en utilisant l'environnement de développement Arduino. Ceci peut paraître être un problème, mais en réalité c'est une opportunité inespérée de voir qu'il est parfaitement possible d'adapter le travail d'un programmeur utilisant d'autres outils, de façon à faciliter ses propres développements.

Tom Zerucha a conçu son code de façon particulière. La majorité de ce dernier est utilisé à la fois pour la version PC et la version Atmel AVR. La subtile différence réside à la fois dans quelques fichiers utilisés dans un cas et non dans l'autre, mais aussi, et surtout par le fait que la version PC peut produire un QR code de n'importe quelle version alors que la déclinaison AVR nécessite un fichier supplémentaire, généré par un programme annexe, qui détermine une seule version de QR code et ce avec un seul niveau de correction d'erreur.

Pour réutiliser ces travaux dans un environnement Arduino, la première chose à faire est donc, sous GNU/Linux (PC ou Raspberry Pi) de récupérer le code de Tom et produire ce fichier particulier. Pour cela, pointez votre navigateur web sur **https://github.com/tz1/qrduino** et récupérez les sources sous la forme d'une archive ZIP. Sous GNU/Linux, décompressez le fichier et vous devez alors obtenir un répertoire `qrduino-master` contenant :

```
COPYING  dofbit.c  ecctable.h  imageproc/
lcd.c  Makefile  Makefile.avr  qrbits.h
qrenc.c  qrencode.c  qrencode.h  qrframe.c
qrjpeg.c  README  regtest.sh
```

À ce stade, si vous utilisez la commande `make`, vous produirez un outil de génération de QR code :

```
% make
gcc -Wall -g -O6   -c -o qrencode.o qrencode.c
gcc -Wall -g -O6   -c -o qrenc.o qrenc.c
gcc -Wall -g -O6   -c -o qrframe.o qrframe.c
gcc   qrencode.o qrenc.o qrframe.o   -o qrencode
gcc -Wall -g -O6   -c -o qrjpeg.o qrjpeg.c
gcc   qrjpeg.o qrencode.o qrframe.o   -o qrjpeg

% echo -n "http://hackable.fr" | ./qrencode 1 2
P1
21 21
1 1 1 1 1 1 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1
1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1
1 0 1 1 1 0 1 0 0 0 1 1 0 0 1 0 1 1 1 0 1
1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 0 1 1 1 0 1
1 0 1 1 1 0 1 0 0 1 0 0 1 0 1 0 1 1 1 0 1
1 0 0 0 0 0 1 0 1 1 1 1 1 0 1 0 0 0 0 0 1
1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
1 0 0 1 0 1 1 0 1 1 0 0 0 1 0 1 0 0 0 0 0
0 0 0 1 1 1 0 1 0 0 0 1 0 1 1 1 1 0 0 0 1
0 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1
0 0 1 0 0 1 0 0 0 0 1 0 1 0 1 0 1 1 0 1 1
1 1 1 1 1 0 1 1 0 0 1 0 1 0 0 1 0 1 0 0 0
0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 1
1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 1 1 0
1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0
1 0 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 1 0 0 0
1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 0 1 1
1 0 1 1 1 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1
1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 0 1 0 0 0
1 1 1 1 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0 1 0
```

L'outil **qrencode** affiche une représentation textuelle des pixels/modules du QR code sous forme de 0 et de 1. Il prend en paramètre respectivement la version à utiliser entre 1 et 40, ainsi que le niveau de correction entre 1 et 4 (correspondants à L, M, Q ou H). Bien entendu, cet outil n'a qu'un intérêt secondaire pour nous.

Pour produire un programme équivalent pour Atmel AVR, Tom utilise un « environnement » en ligne de commandes et plus particulièrement le compilateur **avr-gcc** seul (le même qui est intégré dans l'environnement Arduino). Comme nous ne souhaitons pas utiliser cette méthode, nous devons un peu bricoler.

Nous commençons donc par créer l'outil qui nous permettra de produire un fichier supplémentaire définissant la version de QR code et le niveau de correction que nous souhaitons pour notre projet :

```
% make dofbit
gcc -Wall -g -O6   -c -o dofbit.o dofbit.c
gcc   dofbit.o qrframe.o   -o dofbit
```

Nous avons maintenant à notre disposition la commande **dofbit**. Celle-ci prend deux arguments : la version entre 1 et 40, et le niveau de correction entre 1 et 4. Pour nos essais, nous avons utilisé la version 9 et le niveau M, et donc utilisé **./dofbit 9 2**. Cette commande affiche

directement le code produit sur l'écran. Pour obtenir un fichier, nous redirigeons cela vers un fichier avec `./dofbit 9 2 > dofbit92.c` et obtenons donc le fichier `dofbit92.c`. Le nom du fichier importe peu, mais il est de bon ton de le nommer de façon à s'y retrouver par la suite.

Le fonctionnement du code de Tom est particulier et optimisé pour économiser un maximum de mémoire. Il ne fait cependant pas de miracle et les éventuels problèmes ne sont pas faciles à détecter. En effet, une partie des données du QR code est précalculée, mais le code alloue dynamiquement de la mémoire. Ceci signifie que le message indiquant l'espace mémoire utilisé dans l'environnement Arduino ne doit pas être pris au pied de la lettre, car même s'il semble rester quelques 600 octets de mémoire vive disponibles, ce n'est pas le cas lorsque le croquis fonctionne sur la carte. Nos tests ont montré qu'en fonction du projet et de sa complexité, un QR code en version 10 est plus ou moins le maximum utilisable. N'hésitez pas à consulter le tableau récapitulatif sur **qrcode.com** pour ajuster cela en fonction de vos besoins spécifiques (et n'oubliez pas, qui peut le plus peut le moins).

Le code importé et légèrement modifié est intégré sous forme de fichiers complémentaires qui apparaissent comme des onglets dans l'environnement de développement Arduino. L'ensemble sera compilé pour former le code (binaire) programmé dans la carte.

Une fois votre fichier généré, créez un nouveau croquis dans l'environnement Arduino puis intégrez les fichiers suivants via le menu ***Croquis > Ajouter un fichier*** :

- le fichier que vous avez généré (ici `dofbit92.c`),
- `qrbits.h`,
- `qrencode.h`,
- et `qrencode.c` que vous devrez ensuite renommer `qrencode.cpp` (l'environnement traite les fichiers C++ et C de façon différente et bien que le code soit en C, il faut en changer l'extension pour que la compilation fonctionne).

Ces différents fichiers sont alors copiés automatiquement dans votre projet et apparaissent

sous la forme d'onglets. Il nous reste une modification à apporter pour que nous puissions utiliser tout cela avec notre croquis. La façon dont le compilateur traite les constantes stockées en mémoire flash est différente aujourd'hui de celle d'usage au moment où Tom a écrit son code.

Dans le fichier/onglet **qrencode.c**, repérez les lignes :

```
extern unsigned char framebase[] PROGMEM;
extern unsigned char framask[] PROGMEM;
```

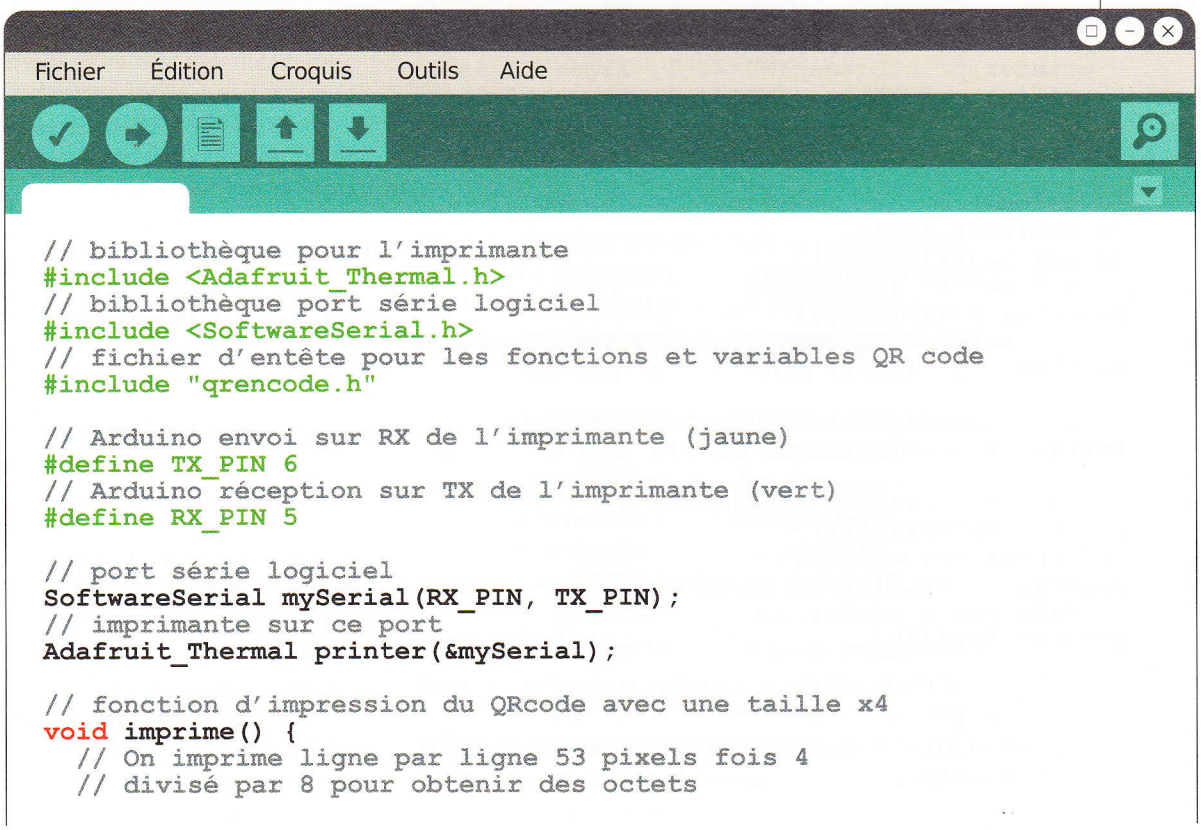
Et ajoutez le qualificateur **const** pour obtenir :

```
extern const unsigned char framebase[] PROGMEM;
extern const unsigned char framask[] PROGMEM;
```

Les versions actuelles du compilateur GCC demandent, en effet, que les « variables » en mémoire de programmation (**PROGMEM**, la flash) soient qualifiées de constantes. C'est l'objet de cette modification.

À ce stade, votre croquis doit pouvoir être construit sans erreur, même s'il ne fait rien pour l'instant.

3. LE CROQUIS FINAL




```
// bibliothèque pour l'imprimante
#include <Adafruit_Thermal.h>
// bibliothèque port série logiciel
#include <SoftwareSerial.h>
// fichier d'entête pour les fonctions et variables QR code
#include "qrencode.h"

// Arduino envoi sur RX de l'imprimante (jaune)
#define TX_PIN 6
// Arduino réception sur TX de l'imprimante (vert)
#define RX_PIN 5

// port série logiciel
SoftwareSerial mySerial(RX_PIN, TX_PIN);
// imprimante sur ce port
Adafruit_Thermal printer(&mySerial);

// fonction d'impression du QRcode avec une taille x4
void imprime() {
  // On imprime ligne par ligne 53 pixels fois 4
  // divisé par 8 pour obtenir des octets
```

```
  // 53*4/8 = 26,5 = 27 octets
  uint8_t ligne[27];

  // variable pour la position en octets
  int pix;

  // on boucle ligne par ligne
  for(int y=0; y<53; y++) {
    // réinitialisation compteur de pixels par ligne
    pix=0;
    // réinitialisation du tableau d'octets (image)
    memset(ligne,0,sizeof(ligne));

    // on boucle sur les pixels par saut de deux
    for(int x=0; x<53; x=x+2) {
      //Un octet de l'image est composé de 2 pixels originaux, répétés 4 fois
      // pixel x
      ligne[pix]|= QRBIT(x,y) << 7;
      ligne[pix]|= QRBIT(x,y) << 6;
      ligne[pix]|= QRBIT(x,y) << 5;
      ligne[pix]|= QRBIT(x,y) << 4;
      // pixel suivant
      ligne[pix]|= QRBIT(x+1,y) << 3;
      ligne[pix]|= QRBIT(x+1,y) << 2;
      ligne[pix]|= QRBIT(x+1,y) << 1;
      ligne[pix]|= QRBIT(x+1,y);
      // octet suivant
      pix++;
    }
    // on imprime la ligne/image 4 fois
    printer.printBitmap(WD*4, 1, ligne, false);
    printer.printBitmap(WD*4, 1, ligne, false);
    printer.printBitmap(WD*4, 1, ligne, false);
    printer.printBitmap(WD*4, 1, ligne, false);
  }
}

void setup() {
  // moniteur série
  Serial.begin(115200);
  // softserial imprimante
  mySerial.begin(9600);
  // initialisation avec temps de chauffe à 200
  printer.begin(200);

  // petit message
  Serial.println(F("GO GO GO"));

  // réinitialisation imprimante
  printer.setDefault();
  // impression message
  printer.println(F("GO GO GO"));
  // défilement papier
  printer.feed(2);
}

void loop() {
  // objet pour la chaîne de caractères reçue
  String saisie;
```

```
  Serial.println("Texte svp ?");

  // On attend l'arrivée de la ligne
  while (Serial.available()==0);

  // lecture du texte
  saisie=Serial.readString();
  // copie dans le tampon pour le QRcode
  saisie.toCharArray((char *)strinbuf, 371);
  // encodage
  qrencode();

  // on imprime le texte
  printer.println(saisie);
  // un espace
  printer.feed(1);
  // impression du QRcode
  imprime();
  // un grand espace
  printer.feed(3);
}
```

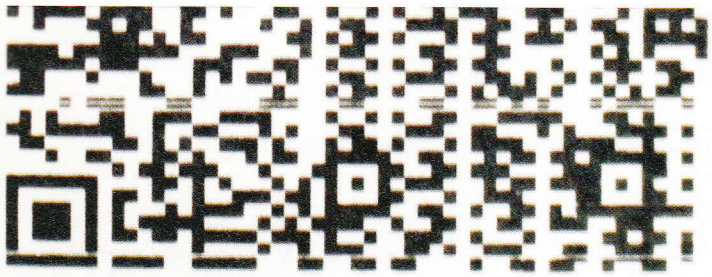
Arduino

Précisions ici quelques points concernant l'utilisation du code de Tom, qui maintenant, d'un point de vue Arduino, est une « bibliothèque » directement intégrée dans notre projet. Les fichiers intégrés mettent à notre disposition une variable `strinbuf`, un pointeur vers un tableau de 371 `char` ou en d'autres termes une chaîne de caractères. Celle-ci est déclarée dans notre `dofbit92.c` (le fichier généré) et dépend de la version du QR code choisie et du niveau de correction utilisé.

Cette chaîne doit être initialisée avec les données qui devront être encodées par la fonction `qrencode()`. Le résultat se trouvera dans un autre tableau, également déclaré dans `dofbit92.c`, `qrframe[]`, mais nous ne l'utiliserons pas directement. Pour récupérer la liste et l'état (noir ou blanc) de chaque pixel/module, nous devons utiliser la macro `QRBIT(x,y)` avec en argument la position du pixel dans l'axe X et Y. Pour parcourir l'image, nous devons donc connaître sa taille. Celle-ci est stockée dans la variable `WD`. Je n'apprécie pas les noms de variables en majuscules, préférant réserver cela aux macros, mais ce n'est pas mon code, c'est celui de Tom. Ceci fait aussi partie des choses à prendre en compte (ou à apprendre à tolérer) lorsqu'on réutilise un code tiers.

Poursuivons sur cette génération de QR codes en parlant de la fonction `loop()`. Il s'agit là d'un traitement de saisie depuis le moniteur série relativement classique. Nous attendons qu'une ligne soit disponible avec `Serial.available()` puis, dès que la valeur retournée est différente de zéro, nous récupérons les données dans un objet `saisi` de type `String`.

Pour transférer ces données dans la zone pointée par `strinbuf`, nous utilisons la méthode `toCharArray` de notre objet `saisie` en lui passant en argument un pointeur vers l'emplacement où copier les données et la taille de cet emplacement (oui, j'aurais pu utiliser `sizeof()`, mais je tenais à rester explicite). Remarquez le `(char *)` devant `strinbuf`. Ceci s'appelle du *casting* et permet de forcer un type de données en un autre. En effet, dans `dofbit92.c`, `strinbuf` est un pointeur vers un `unsigned char`, mais `toCharArray` attend un pointeur vers un `char`. Ce casting nous permet de dire au compilateur que tout va bien, nous savons ce que nous faisons.



La qualité d'impression sur du papier thermique est dépendante d'un grand nombre de paramètres. Certains sont réglables directement dans la configuration de l'imprimante, d'autres découlent tout simplement de l'âge du consommable. Ici on voit clairement que l'impression est loin d'être parfaite (le rouleau de papier doit avoir dans les 8 ans).

Après cette copie, il nous suffit d'invoquer `qrencode()` pour générer l'image, imprimer le texte reçu, faire un saut de ligne puis appeler notre fonction d'impression avant de refaire un saut, plus important.

4. UN MOT SUR LA FONCTION D'IMPRESSION

Le QR code, une fois généré par la fonction `qrencode()` est stocké sous forme binaire dans un tableau `qrframe[]` en SRAM. Il est techniquement possible d'utiliser directement ces données avec la méthode `printBitmap()` de l'objet `Adafruit_Thermal` (`printer`). Les données sont organisées dans un format tout à fait compatible avec cette méthode permettant d'imprimer une image. Le problème cependant est que l'image fait 53 pixels de côté et qu'une fois imprimée à cette taille il est totalement impossible de la scanner avec un smartphone.

Nous devons donc agrandir l'image avant impression. Un petit problème cependant nous bloque dans cette démarche : l'Arduino UNO n'a plus suffisamment de mémoire (SRAM) pour stocker une image d'une taille raisonnable (4 fois plus grande par exemple, soit 11236 pixels ou quelques 1400 octets).

Il faut donc ruser et la solution adoptée ici consiste à découper le QR code en 53 images de 1 pixel de haut, qui seront alors imprimées les unes à la suite des autres (fois 4). Le gain de mémoire est important, car nous pouvons utiliser le même espace mémoire pour chaque image, la seconde écrasant les données de la première et ainsi de suite.

Multiplier la taille par quatre verticalement n'est pas difficile puisqu'il nous suffit d'imprimer la même image quatre fois de suite. Horizontalement cependant c'est une tout autre affaire. Les données des images à imprimer sont matérialisées par une série d'octets dont les bits représentent les données brutes de l'image (les pixels). Deux arguments, en plus du pointeur vers les données, sont passés à la méthode `printBitmap()` précisant la hauteur et la largeur de l'image. Enfin, un quatrième argument précise si les données se trouvent en mémoire vive (SRAM) ou en mémoire de programme (flash).

La macro `QRBIT(x,y)` nous permet de connaître la valeur de chaque pixel qui peut être soit 1 pour noir, soit 0 pour blanc et nous pouvons donc parcourir l'image pour composer nous-mêmes une version agrandie. Chaque octet de nos données d'image sera composé de quatre fois la valeur d'un pixel puis quatre fois la valeur du suivant.

Nos « tranches » d'images doivent faire 53×4 bits soit, arrondis à l'octet supérieur, 27 octets. Nous construisons alors une simple boucle pour traiter les données d'une tranche :

```
uint8_t ligne[27];
pix=0;

for(int x=0; x<53; x=x+2) {
  ligne[pix]= QRBIT(x,y) << 7;
  ligne[pix]|= QRBIT(x,y) << 6;
  ligne[pix]|= QRBIT(x,y) << 5;
  ligne[pix]|= QRBIT(x,y) << 4;
  ligne[pix]|= QRBIT(x+1,y) << 3;
  ligne[pix]|= QRBIT(x+1,y) << 2;
  ligne[pix]|= QRBIT(x+1,y) << 1;
  ligne[pix]|= QRBIT(x+1,y);
  pix++;
}
```

Nous composons chaque octet en utilisant le bit récupéré par `QRBIT(x,y)` et le décalons grâce à l'opérateur `<<` à la bonne place dans l'octet. Notez l'utilisation de `|=` permettant un OU logique avec la valeur déjà présente dans `ligne[pix]`. Ces quelques lignes pourraient être regroupées, de façon moins lisible, en

```
ligne[pix]=
(QRBIT(x,y) << 7) | (QRBIT(x,y) << 6) |
(QRBIT(x,y) << 5) | (QRBIT(x,y) << 4) |
(QRBIT(x+1,y) << 3) | (QRBIT(x+1,y) << 2) |
(QRBIT(x+1,y) << 1) | QRBIT(x+1,y);
```

.

Pour traiter toutes les 53 lignes du QR code, il nous suffit d'inclure cela dans une autre boucle, en incrémentant la variable `y` correspondant à la position verticale du pixel lu. Et voilà comment on peut imprimer une image de 11236 pixels avec seulement 2 Ko de SRAM et trois bibliothèques qui ne sont pas spécialement économes en mémoire vive.

Ceci devrait être facilement adaptable à un autre support d'affichage, comme un écran OLED ou LCD par exemple, comme ceux déjà présentés dans le précédent numéro du magazine.

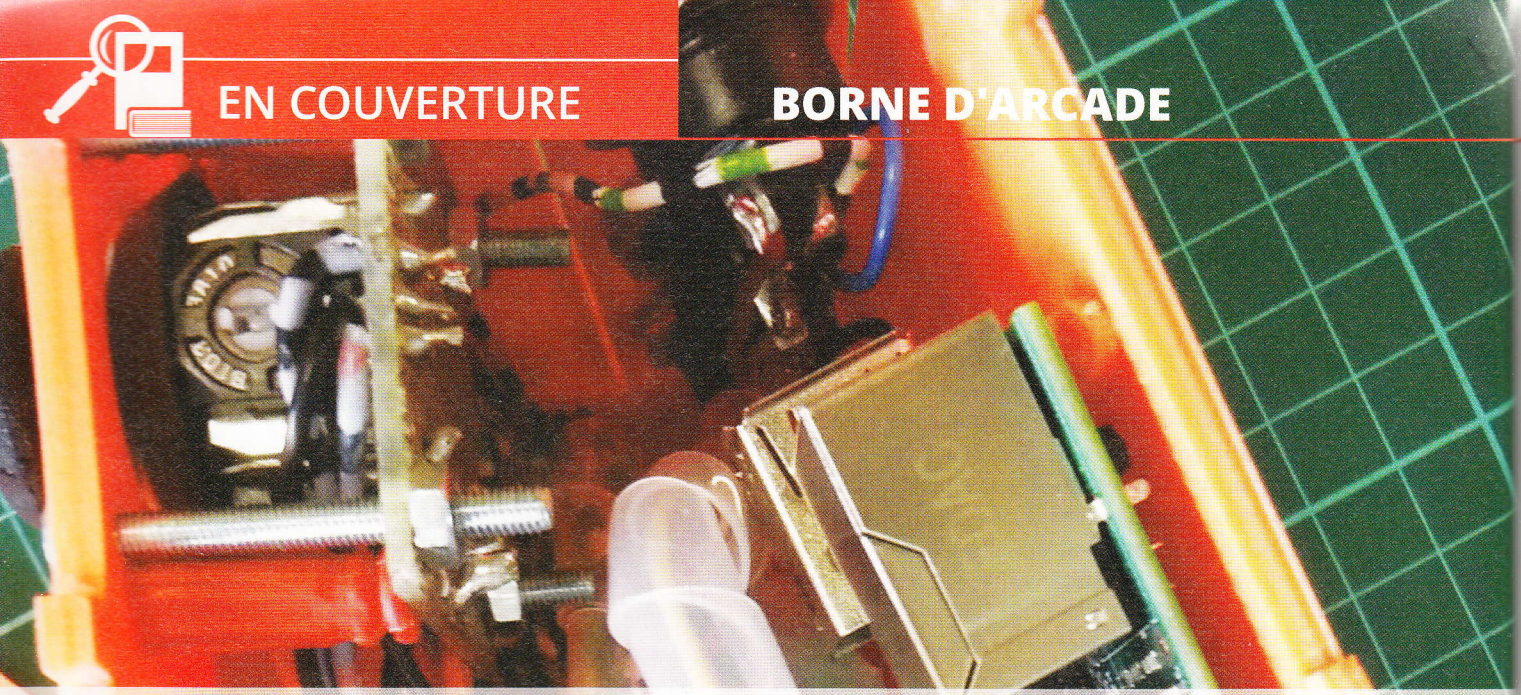
Voici le résultat final : un QR code directement généré par la carte Arduino à partir des données envoyées via le moniteur série et agrandi par un facteur 4 avant impression.

5. PERSPECTIVES ET AUTRES UTILISATIONS

Nous avons choisi ici un exemple relativement simple, mais déjà très intéressant. En effet, grâce à notre croquis et la carte Arduino, nous avons transformé l'imprimante thermique, déjà capable d'imprimer des codes barres unidimensionnels, en imprimante de QR codes. Nous pouvons donc non seulement nous en servir en lui envoyant du simple texte, mais également modifier notre croquis pour générer de façon autonome une série d'images imprimées. Une application pratique pourra, par exemple, être la génération de coupons de réduction pour une boutique en ligne sous la forme d'URL à usage unique.

En poussant les choses un peu plus loin, on pourrait également transposer ce concept sur une autre plateforme comme un ESP8266 et obtenir une solution d'impression en Wifi. La même chose pourrait être envisagé avec l'ajout d'un module Bluetooth par exemple.

Enfin, je terminerai en précisant que le choix de la carte Arduino, une UNO, était parfaitement délibéré afin de s'imposer une contrainte technique « amusante ». Pour des projets plus complets, il serait judicieux de basculer sur une Arduino Mega 2560 offrant plusieurs ports série matériels, 256 Ko de mémoire flash et surtout 8 Ko de mémoire vive (SRAM), donnant bien plus de libertés d'action... DB



DIY : CRÉEZ VOTRE MICRO BORNE D'ARCADE DE A À Z

Yann Morère

Le but de cet article est de présenter la fabrication complète d'une micro borne d'arcade : de la conception de la caisse jusqu'à l'intégration de la partie informatique. Pour cela, on utilisera l'impression 3D ainsi qu'un Raspberry Pi et Raspicade pour la partie logicielle.

Comme vous l'avez sans doute remarqué, le rétrogaming est de plus en plus présent dans les revues et sur Internet. Il existe maintenant de nombreux salons et rencontres de passionnés autour de ces jeux qui ont bercé notre enfance (enfin la mienne en tout cas). L'arcade et les anciennes consoles par l'intermédiaire de l'émulation sont à nouveau très prisées. De plus, l'arrivée du Raspberry Pi a encore donné un nouvel élan à cette tendance en proposant des solutions d'émulation très bas coût par l'intermédiaire des nombreuses distributions dédiées à l'arcade disponibles pour ce petit ordinateur. Parmi elles, on retrouve Retropie **[1]**, Piplay **[2]**, et du côté français la très bonne Recalbox **[3]** (ma préférée), Happi **[4]** et aussi Raspicade **[5]** (développée par votre serviteur ;-)).

L'intérêt majeur de l'utilisation d'un Raspberry Pi par rapport à une solution basée sur une architecture PC est l'encombrement et la consommation énergétique. Bien sûr, il faudra être conscient de la faible puissance de calcul, et ne pas vouloir émuler des systèmes/consoles trop récents. On pourra ainsi réaliser des intégrations de très petites tailles. On trouve d'ailleurs de nombreux projets concernant la fabrication de consoles portables à base de notre petite framboise : le projet Super Pi Boy présente un clone de Gameboy **[6]**, la Super Game Pi d'Adaftuit **[7]**, le projet Raspberry Gear **[8]** ainsi que bien d'autres sur le site Thingiverse **[9]**.

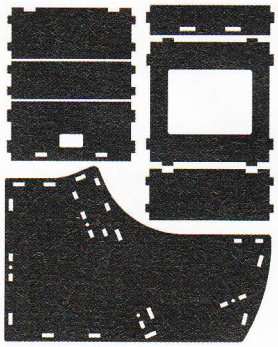
Figure 1

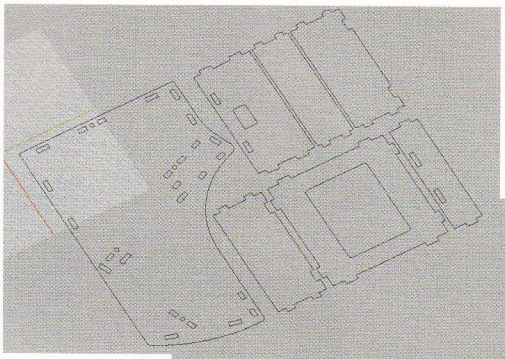
L'idée d'une micro borne d'arcade est venue lorsque j'ai redécouvert dans mon grenier le jeu Epoch Galaxy II de mon enfance (Figure 1). Il s'agit d'un clone du jeu d'arcade Galaxian de type « *Table Top* » produit en 1981-82.

Pourquoi ne pas faire une petite borne d'arcade de cette taille avec tous les jeux de cette époque ? Avant de poursuivre, je me suis tourné vers Internet et j'ai constaté que je n'étais pas le seul à avoir eu cette idée. Adafruit propose sa Cupcade **[10]**, le site Instructables **[11]** propose une micro borne de la taille d'une canette de 33cl. La micro borne qui ressemble sans doute le plus à celles des années 80 est sans doute la Star Force Pi **[12]** issue d'un financement participatif. Alors, pourquoi créer sa propre borne ? Pour le plaisir, mais aussi parce que ces projets souffrent de certaines limitations. Par exemple, la Cupcade ne permet de jouer qu'à des jeux 2 boutons et l'émulateur est limité à Mame. Impossible de jouer à des jeux nécessitant 4 ou 6 boutons. La mini borne Instructables est bien trop petite pour permettre une vraie jouabilité.

Le prix de revient d'un tel jouet dépasse facilement les 100 €. Outre le plaisir de construire « from scratch » une micro borne, il est important de minimiser, si possible le coût. Ayant une imprimante 3D à disposition, le plus rentable sera de concevoir la caisse de la borne et ensuite de l'imprimer. On peut aussi se rapprocher d'un Fablab ou d'un imprimeur en ligne pour la réalisation de la caisse.

Cependant, certains matériels seront incontournables : l'écran TFT, les contrôles (joysticks et boutons) le Raspberry Pi et l'alimentation. Là encore, nous aurons le choix entre les produits supportés, documentés, mais assez chers de chez Adafruit, ou des équivalents chinois bien moins chers, mais sans documentation et réel support.





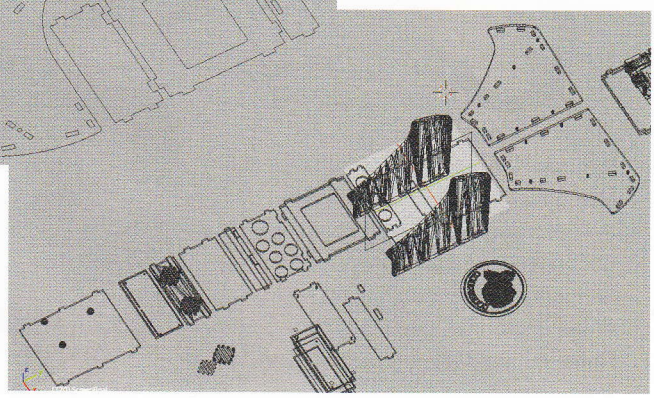


Figure 2

Dans cet article, nous utiliserons les matériels suivants pour la création de 2 bornes :

- 1 Raspberry Pi 512 rev B et son alimentation ;
- 1 Raspberry Pi B+ et son alimentation ;
- 1 écran Adafruit PiTFT assemblé 320x240 2.8" **[13]** ;
- 1 écran Waveshare 3"2 320x240 **[14]** ;
- 2 joysticks analogiques pour Xbox 360/PS2, que l'on modifiera **[15]** ;
- 18 boutons **[16]** ;
- 2 amplificateurs audio stéréo 5V **[17]**.

On récupérera les potentiomètres de réglage de volume ainsi que les haut-parleurs sur 2 vieux casques micro. Il vous faudra aussi, 2 cartes SD de 8Go, un bon fer à souder, de la patience... beaucoup de patience.

Note

Le nouveau Pi Zéro semble un candidat idéal pour ce genre de projet. Cependant, à l'heure de l'écriture de ces lignes, je n'ai toujours pas réussi à m'en procurer un, car il est continuellement en rupture de stock.

Les 2 bornes seront équipées d'un joystick 4 directions ainsi que 9 boutons : 6 boutons pour le jeu, un bouton de crédit, un bouton de « Start » et un bouton pour quitter les émulateurs.

La première borne utilisera l'écran TFT Adafruit, le Raspberry Pi 512 rev B ainsi qu'un petit module pour multiplexer les entrées sur l'interface GPIO. En effet, l'écran TFT utilise le bus SPI et réquisitionne de fait 7 GPIOs. En conséquence, il n'en reste plus assez pour récupérer tous les contrôles de notre panel arcade. Ce petit module utilise un composant MCP23017 sur le bus I2C et permet de multiplexer les entrées à l'aide d'un programme résident pikeyd **[18]**.

Note

Pourquoi ne pas avoir utilisé un Pi2 bien plus puissant pour ce type de projet ? Parce que je voulais trouver une utilisation pour ces 2 anciennes versions afin de limiter au maximum le budget. Cependant, il est tout à fait possible d'utiliser un Pi2 ou un Pi zéro, la marche à suivre sera identique à celle du B+.

Note

L'ensemble des manipulations de cet article peut aussi être réalisé sur une distribution Debian ou MiniBian.

Le seconde borne utilisera l'écran Waveshare. Le Raspberry Pi B+ embarquant des GPIOs supplémentaires, il ne sera pas nécessaire d'utiliser un module additionnel. Nous allons seulement utiliser le connecteur GPIO pour récupérer l'ensemble de nos contrôles à l'aide du programme `retrogame` **[35]** que nous aurons modifié pour la prise en compte de l'ensemble des boutons et de l'écran utilisant lui aussi le bus SPI **[19]**.

Les 2 bornes utiliseront une distribution Raspicade (version spécifique Pi1) que l'on adaptera pour la prise en charge de l'écran et des contrôles.

Passons maintenant à la conception de la caisse de la borne d'arcade.

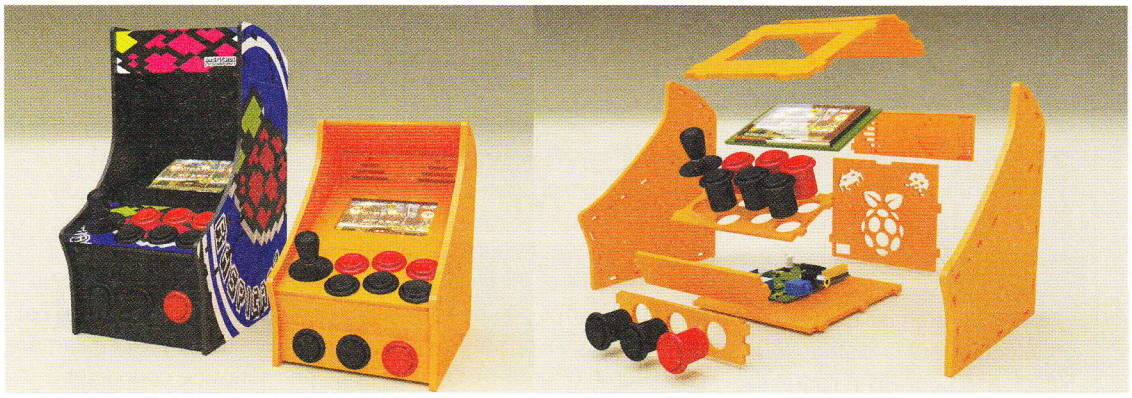


Figure 3

1. CONCEPTION DE LA CAISSE

Pour la réalisation de la forme générale de la borne, j'ai utilisé comme base celle de la Cupcade **[18]**. Bien sûr, il a fallu modéliser en 3D à l'aide de Blender l'ensemble des pièces qui la compose à partir des plans 2D. La conception a dû être modifiée afin de prendre en compte les contraintes de dimensions maximales d'impression.

La technique de modélisation est assez simple. On part des formes 2D de la borne issue du fichier PDF de la référence **[20]** que l'on convertira au format SVG. On importe ensuite le fichier généré dans Blender.

Le passage à la 3D se fait à l'aide d'une simple extrusion. Il faudra ensuite modifier la forme et les emplacements des boutons. On n'oubliera pas de prévoir l'emplacement de fixation du Raspberry Pi.

La figure 2 (ci-contre) résume l'ensemble des opérations.

La figure 3 montre la version initiale et la version modifiée pour l'impression ainsi qu'une vue éclatée montrant la disposition des composants : Raspberry Pi, écran et bouton.

Ensuite, il ne reste plus qu'à vérifier nos différentes pièces à l'aide de la Toolbox 3D Print avant l'export au format STL qui nous servira pour l'impression 3D. Je vous renvoie vers l'article dédié à l'impression 3D à l'aide d'outils libres paru dans *Linux Pratique n°91* pour plus d'informations.

Dans cette première version, j'ai laissé les lumières des ergots de fixations traversantes sur les panneaux latéraux. À la suite du montage de la première caisse, j'ai constaté que l'on pouvait les laisser borgnes. La seconde version est donc bien plus jolie comme montré sur la figure 4.

L'ensemble des fichiers nécessaires à l'impression 3D est disponible aux adresses **[21]** et **[22]**.

Passons maintenant à la configuration de notre matériel informatique pour notre première borne.

Note

Je vous déconseille l'utilisation de l'ABS pour l'impression des panneaux latéraux de la borne. En effet, sur ces grandes surfaces, l'ABS se rétracte beaucoup et la pièce est beaucoup trop déformée pour être utilisable.

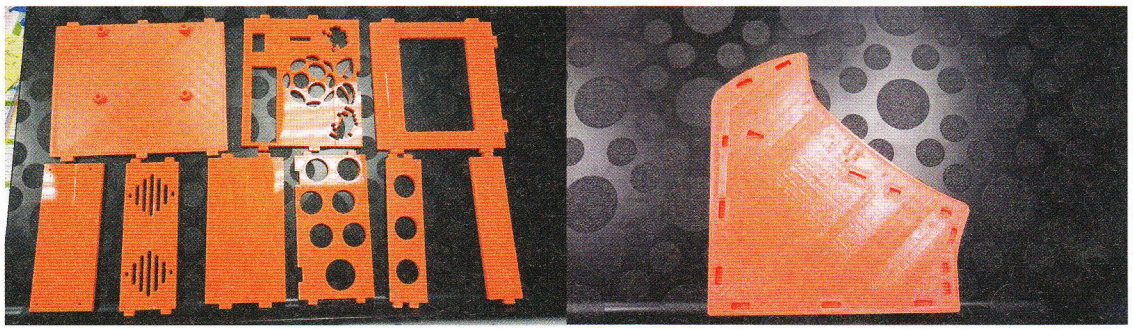
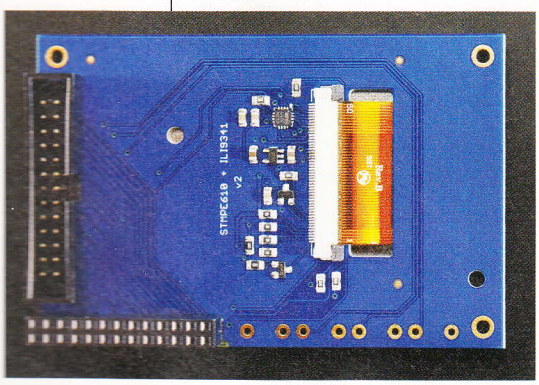


Figure 4

2. VERSION 1 : BORNE À BASE DE PI 512 REV B

Dans un premier temps, on téléchargera la distribution Raspicade pour Pi1 à l'adresse **[23]**. Après avoir créé l'image de la carte SD, on connecte l'écran sur le port GPIO du Raspberry Pi. L'écran n'étant pas encore supporté, on branchera un écran sur le port HDMI (TV ou écran VGA à l'aide d'un convertisseur) et un clavier. Vous pouvez vous affranchir de tout cela en branchant le Raspberry Pi sur le réseau de votre box et réaliser les manipulations à distance à l'aide d'une connexion SSH. Pour la distribution Raspicade, l'identifiant est `pi` et le mot de passe `pi`. Cet utilisateur fait aussi partie du groupe `sudo` et vous permet de réaliser les tâches d'administration. Je vous conseille la lecture de la documentation en français disponible à l'adresse **[24]** pour mieux comprendre la distribution et son fonctionnement.

Figure 5



2.1 Configuration pour le support de l'écran TFT

On va commencer par ajouter le support de l'écran PiTFT d'Adafruit. Cet écran possède une interface tactile, mais nous ne l'utiliserons pas dans la suite. Un des gros avantages de cet écran est qu'il possède un report du connecteur GPIO à l'arrière de l'écran (Figure 5). Cela va nous permettre de connecter facilement notre module de multiplexage. Pour des raisons de coûts de fabrication, ce type de connecteur ne sera pas présent sur l'écran Waweshare.

La distribution Raspicade ne supporte pas en standard l'écran PiTFT. On va utiliser les paquets fournis par Adafruit pour le support. Cela va modifier le noyau et cassera le support du module XinMo (module arcade spécifique avec un support nécessitant de patcher le noyau). Ce n'est pas vraiment grave, car nous ne l'utilisons pas dans la suite.

Note

> Le support des écrans TFT sur Raspberry Pi est maintenant mature, et on peut maintenant configurer l'écran sans changer de noyau. Par facilité, on utilise ici les paquets réalisés par Adafruit.

La mise en œuvre de la configuration de l'écran est bien documentée sur le site Adafruit **[25]**. Voici les étapes de l'installation.

On commence par installer un nouveau noyau qui prend en charge l'écran PiTFT 2.8 résistif ainsi qu'un utilitaire de configuration :

```
$ curl -SLs https://apt.adafruit.com/add | sudo bash
$ sudo apt-get install -y raspberrypi-bootloader
$ sudo apt-get -y install adafruit-pitft-helper
```

Ensuite, on édite le fichier `/boot/config.txt` et on y ajoute les lignes suivantes :

```
[pi1]
device_tree=bcm2708-rpi-b-plus.dtb
[pi2]
device_tree=bcm2709-rpi-2-b.dtb
[all]
dtparam=spi=on
dtparam=i2c1=on
dtparam=i2c_arm=on
dtoverlay=pitft28r,rotate=90,speed=32000000,fps=20
```

La variable « rotate » ordonne au pilote de réaliser une rotation de l'écran de 0 90 180 ou 270 degrés :

- `0` est le mode « portrait » (hauteur plus grande que largeur), et le bas de l'écran est près des prises USB ;
- `90` est le mode « paysage » et le bas de l'écran est proche de la sortie son jack 3"5 ;
- `180` est le mode « portrait » avec le haut de l'écran près des prises USB ;
- `270` est le mode « paysage » et le haut de l'écran est proche de la sortie son jack 3"5.

La variable `speed` indique au pilote la vitesse de pilotage de l'écran. 32MHz (32000000) est un bon début, mais si vous avez des problèmes de stabilité d'images ou de rétroéclairage, il faudra régler empiriquement cette valeur. On peut essayer 16MHz (16000000) et augmenter progressivement dans le cas où l'écran est déporté du Pi. En effet, j'ai noté des problèmes de stabilité d'image liés à la longueur de nappe pour la connexion entre le port GPIO du Pi et celui de l'écran.

On peut ensuite redémarrer le Raspberry Pi. Normalement, l'écran PiTFT s'allume blanc puis repasse au noir, c'est le signe qu'il est reconnu par le système. Oui, je sais, c'est un peu bizarre...

L'écran tactile n'est pas configuré, mais nous n'en aurons pas besoin dans la suite. Si vous désirez l'activer, je vous renvoie vers l'adresse **[26]**.

Afin de gagner en performance, Raspicade n'utilise pas de serveur graphique pour l'émulation, mais directement le framebuffer. On aura donc exclusivement un usage console.

Afin d'avoir un affichage lisible, on réalise la configuration de la police texte qui sera utilisée par le système. Pour cela, on reconfigure le paquet `console-setup` :

```
$ sudo dpkg-reconfigure console-setup
```

Dans les menus affichés, sélectionnez ***UTF-8***, puis ***Guess optimal character set***, puis ***Terminus*** et finalement, ***6x12 (framebuffer only)***.

Ensuite, on lance la configuration de l'écran à l'aide du programme `adafruit-pitft-helper` en spécifiant le type de notre écran 2,8" capacitif :

```
$ sudo adafruit-pitft-helper -t 28r
```

Il est possible d'utiliser aussi un écran 2,2" HAT avec l'option `22`, un écran 2,8" capacitif avec l'option `28c`.

À la question concernant l'utilisation de la console, il faut répondre non. En effet, afin de pouvoir conserver un affichage graphique en mode framebuffer (pour l'affichage d'EmulationStation et des jeux émulés), nous allons faire croire au Raspberry Pi qu'il utilise toujours la sortie HDMI, et nous allons utiliser un programme qui va copier le framebuffer HDMI `/dev/fb0` vers le framebuffer de l'écran TFT `/dev/fb1` sur le port SPI du GPIO.

Ce programme se nomme `fbcp` et se trouve sur GitHub. On l'installe de la manière suivante :

```
$ sudo apt-get install cmake
$ git clone https://github.com/tasanakorn/rpi-fbcp
$ cd rpi-fbcp/
$ mkdir build
$ cd build/
$ cmake ..
$ make
$ sudo install fbcp /usr/local/bin/fbcp
```

Afin que le programme se lance automatiquement au démarrage, permettant ainsi d'utiliser seulement l'écran TFT, nous allons créer un nouveau service par l'intermédiaire du fichier `/etc/init.d/afbcp` suivant :

```
#!/bin/bash
### BEGIN INIT INFO
# Provides:          afbcp
# Required-Start:
# Required-Stop:
# Should-Start:
```

```
# Default-Start:     S
# Default-Stop:
# Short-Description: Start framebuffer copy utility
# Description:       Start framebuffer copy utility
### END INIT INFO

do_start () {

    /usr/local/bin/fbcp &
    exit 0
}

case "$1" in
  start|"")
    do_start
    ;;
  restart|reload|force-reload)
    echo "Error: argument '$1' not supported" >&2
    exit 3
    ;;
  stop)
    # No-op
    ;;
  status)
    exit 0
    ;;
  *)
    echo "Usage: afbcp [start|stop]" >&2
    exit 3
    ;;
esac

:
```

Après avoir rendu le script exécutable, on active le service dans le système :

```
$ sudo insserv  /etc/init.d/afbcp
```

On pourra le désactiver simplement par la commande :

```
$ sudo insserv -r /etc/init.d/afbcp
```

On édite ensuite le fichier **/boot/config.txt** afin de renseigner la résolution de l'affichage en 320x240 :

```
hdmi_force_hotplug=1
hdmi_cvt=320 240 60 1 0 0 0
hdmi_group=2
hdmi_mode=87
dtoverlay=pitft28r,rotate=90,speed=80000000,fps=60
```

Voici les valeurs finales fonctionnelles après l'overclockage :

```
# --- added by adafruit-pitft-helper Wed May  6 05:51:00 CEST 2015 ---
[pi1]
device_tree=bcm2708-rpi-b-plus.dtb
```

```
[pi2]
device_tree=bcm2709-rpi-2-b.dtb
[all]
dtparam=spi=on
dtparam=i2c1=on
dtparam=i2c_arm=on
# working config
# 900 Mhz -> 48000000
# 1Ghz -> 35000000
#modification of the speed
dtoverlay=pitft28r,rotate=90,speed=22000000,fps=60
# --- end adafruit-pitft-helper Wed May  6 05:51:00 CEST 2015 ---
#resolution configuration
hdmi_force_hotplug=1
hdmi_cvt=320 240 60 1 0 0 0
hdmi_group=2
hdmi_mode=87
```

Tout est maintenant prêt, au prochain redémarrage, votre écran TFT devrait vous afficher la vidéo de Raspicade au démarrage, puis EmulationStation (Figure 6).

Figure 6

Une remarque importante est à faire ici. Suivant la longueur de votre nappe entre le GPIO et l'écran, l'overclockage utilisé pour le Raspberry Pi, l'affichage peut devenir instable. Cela vient de la vitesse de transfert imposée sur le bus SPI. L'instabilité peut se matérialiser par une image déformée, ou encore une luminosité qui oscille. Dans les 2 cas, il faudra baisser la vitesse de transfert et trouver empiriquement celle qui convient à votre configuration.

Un réglage fonctionnel pour un overclockage à 1,1Ghz du Pi 512 rev B est le suivant :

```
#overclocking extreme
arm_freq=1100
core_freq=550
sdram_freq=600
over_voltage=8
over_voltage_sdram=6
force_turbo=1

#resolution configuration
hdmi_force_hotplug=1
hdmi_cvt=320 240 60 1 0 0 0
hdmi_group=2
hdmi_mode=87
# working config
# 900 Mhz -> 48000000
# 1Ghz -> 35000000
#modification of the speed
dtoverlay=pitft28r,rotate=90,speed=22000000,fps=60
```

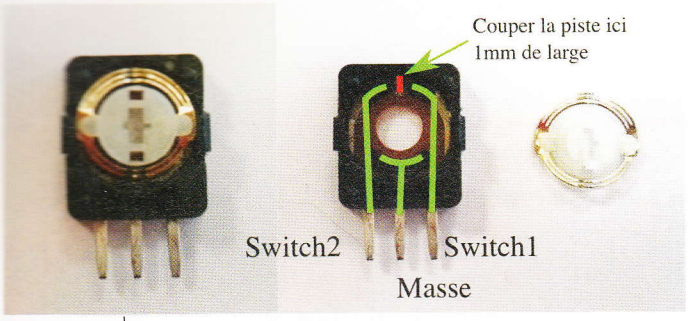


Figure 7

Notre écran est maintenant fonctionnel, passons à la prise en charge de nos contrôles arcade.

2.2 Modification du joystick analogique

Avant de réaliser le câblage sur le port GPIO de tous nos contrôles, intéressons-nous aux joysticks. Les plus attentifs d'entre vous auront remarqué que les joysticks utilisés sont analogiques, alors que le joystick arcade utilise des contacts binaires (lamelles ou micro switchs) pour les directions. Il va falloir transformer nos joysticks analogiques de manière à ce qu'ils se comportent comme leurs homologues à contact.

2 solutions sont possibles : la première consiste à utiliser le module joystick vendu par Adafruit **[27]** qui embarque une conversion analogique/numérique (à base du comparateur de tension LM339) en plus d'un amplificateur audio ou d'en créer un similaire. L'autre solution consiste à « hacker » les potentiomètres du joystick analogique afin qu'ensuite ils se comportent comme un interrupteur 2 états. Ceci est décrit à l'adresse **[28]**.

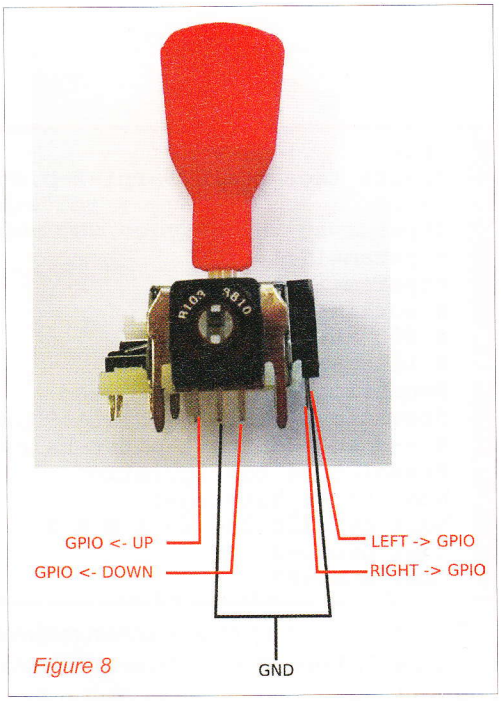


Figure 8

Notre joystick analogique contrôle 2 potentiomètres, un pour chaque direction de déplacement. La tension de sortie dans chaque direction va donc varier en fonction de la position du joystick. Nous désirons obtenir un niveau logique à la place d'une variation de tension pour chaque position (Haut, Bas, Droite et Gauche). On peut bien sûr réaliser cela à l'aide d'un PIC, Arduino ou un comparateur de tension, mais ici nous allons simplement utiliser un cutter.

Il faut commencer par démonter les potentiomètres du joystick (Figure 7). Ensuite, il faut démonter le potentiomètre lui-même afin de faire apparaître les pistes de celui-ci.

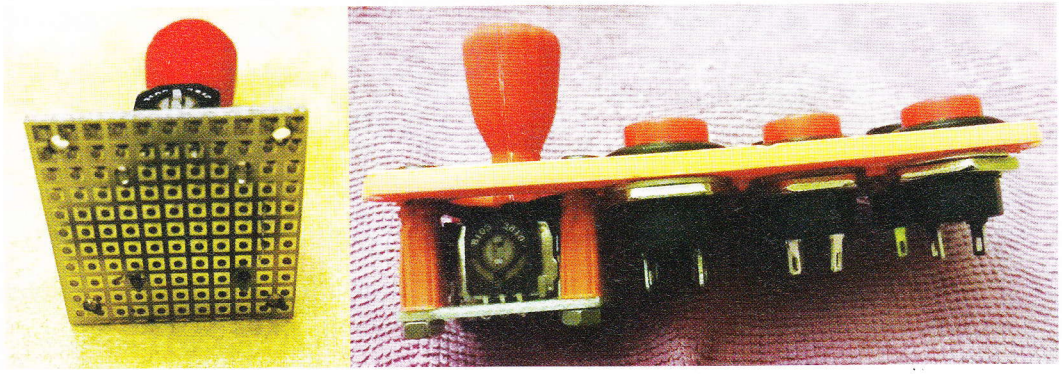


Figure 9

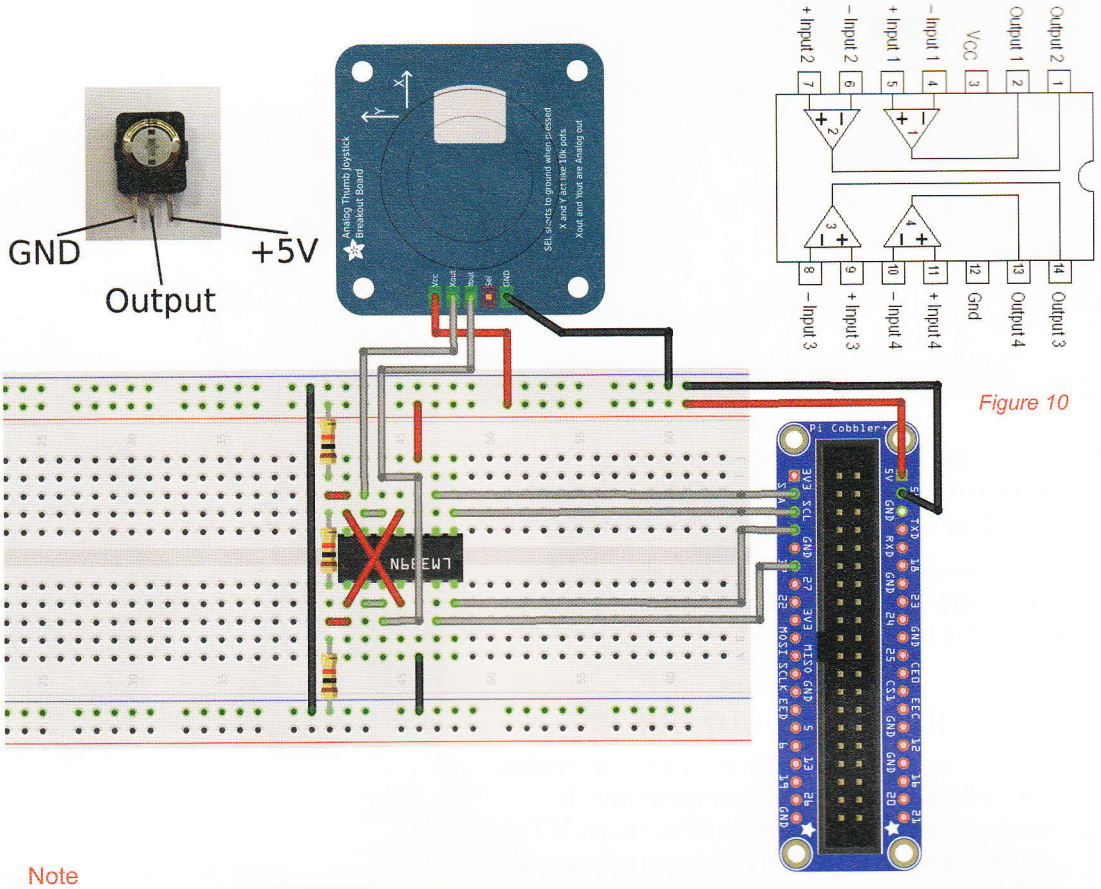


Figure 10

Note

Comme on ne coupe qu'une partie de la piste, il reste une certaine impédance. Avec le multimètre, vous pourrez évaluer les valeurs d'impédance entre les parties droite et gauche. Ce n'est effectivement pas parfait, car lorsque le joystick sera connecté sur le GPIO, cette impédance résiduelle engendrera une petite latence avant le basculement de l'entrée GPIO entre 2 états. Si cette latence est trop importante à votre goût, il faudra utiliser un montage à base de comparateur de tension décrit dans la suite.

On coupe la partie supérieure de la piste à l'aide d'un cutter. Il faut que la séparation entre les deux parties soit assez large afin que le patin métallique du potentiomètre ne pose plus sur la piste lorsqu'il est au repos. En effet, il faut que le circuit soit coupé entre la ligne de masse et les lignes Switch1 et 2. On peut remonter le potentiomètre et vérifier son bon fonctionnement à l'aide d'un multimètre.

On réitère cette opération sur le second potentiomètre. Le câblage sur le GPIO sera celui décrit en figure 8, ci-contre. On n'utilisera pas le bouton situé sous le joystick. On prépare ensuite un morceau de plaque d'essai afin de faire un support. Après avoir soudé le joystick, on monte le tout sur le panel de notre borne (Figure 9).

Dans le cas où vous trouveriez la latence trop importante pour la jouabilité, il est possible de réaliser un montage électronique très simple sur la base du circuit LM339. C'est un circuit qui contient 4 comparateurs de tension et il va nous permettre de transformer les sorties analogiques de notre joystick en sorties numériques. Le schéma du circuit est donné en figure 10. Il s'agit du montage proposé par Adafruit pour sa Cupcade, sans la partie amplification audio **[27]**. Une fois réalisé, on obtient le montage de la figure 11, page suivante.

La sortie en tension de chaque potentiomètre de direction est comparée à 2 valeurs seuil définies par nos résistances de 10kOhms dans un pont diviseur à 3 résistances. Lorsque celle-ci est supérieure au seuil haut (3,4V), la sortie correspondante du comparateur est activée, lorsqu'elle est inférieure au seuil bas (1,6V) l'autre sortie est activée. Cela nous permet de transformer simplement notre signal analogique en signal numérique.

Le câblage des potentiomètres du joystick est différent de celui de la figure 7 puisqu'ils ne seront modifiés. La figure 10 décrit comment les connecter à l'alimentation du Raspberry Pi.

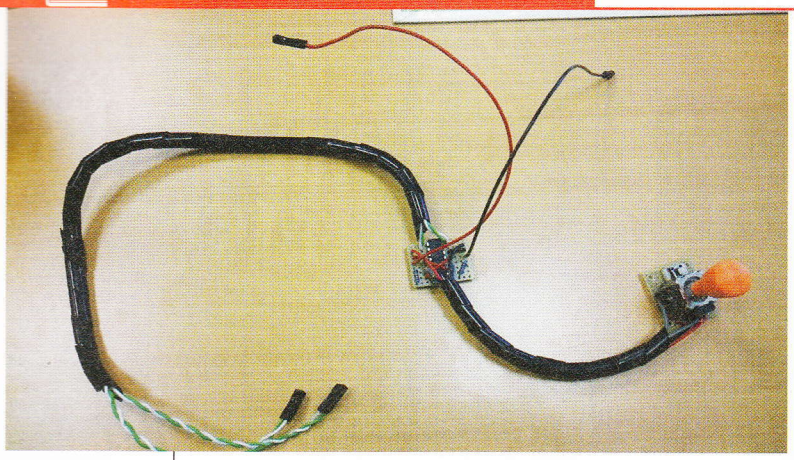


Figure 11

Afin de ne pas endommager le GPIO du Raspberry Pi, on vérifiera qu'il n'y a pas de court-circuit sur notre montage avant de le brancher.

Passons maintenant à la récupération de tous nos contrôles sur le GPIO, à l'aide d'un multiplexeur utilisant le bus I2C.

2.3 Multiplexage sur bus I2C

Comme nous l'avons déjà dit, il n'est pas possible sur un Raspberry Pi de première génération de connecter simultanément sur le GPIO l'écran TFT et un panel arcade regroupant un joystick 4 directions et 9 boutons (6 boutons de jeu, Start P1, Credits et Escape/Quit). En effet, l'écran, qui utilise le port SPI, réquisitionne 7 entrées GPIO. Il n'en reste alors plus que 10 disponibles, et nous en avons besoin de 13. La solution consiste à utiliser un module mcp23017 sur le bus I2C pour multiplexer les entrées. On trouvera à l'adresse **[29]** les fichiers nécessaires à la génération de la petite carte électronique (Figure 12). On peut obtenir le composant MCP23017 en échantillon gratuit auprès de la société Microchip. Nous utiliserons la version à 16 entrées. Dans le cas d'un panel 2 joueurs, on utiliserait le module à 32 entrées.

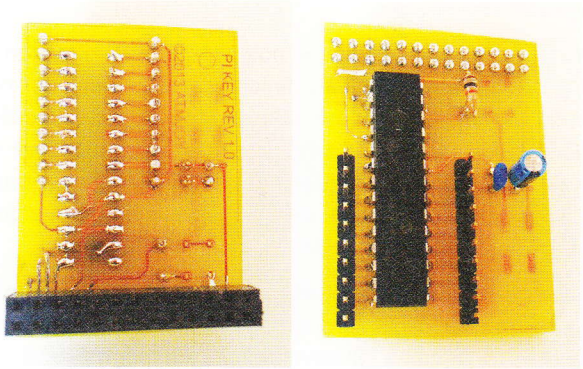
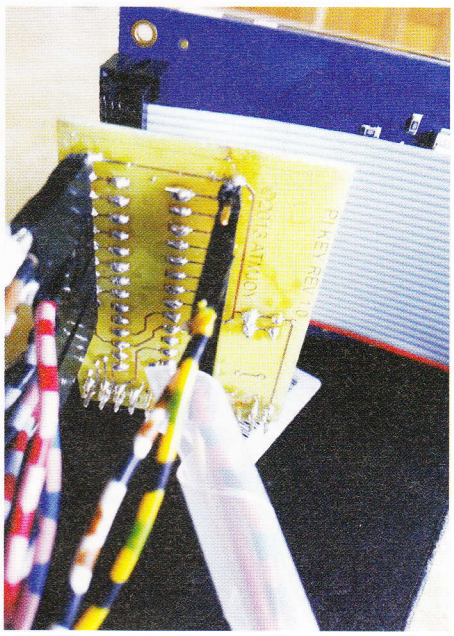


Figure 12

Figure 13



Note

Nous aurions pu nous affranchir du module sur bus I2C en utilisant les GPIO supplémentaires non câblés sur P5 ainsi que ceux du connecteur S5 de la caméra.

Les composants nécessaires en plus des connecteurs pour la version 16 entrées sont : 1 MCP230147, 1 résistance 10K Ohms, 1 condensateur 10 µF et 1 condensateur 100 nF.

On peut le brancher sur le GPIO délocalisé de l'écran comme montré en figure 13. Le câblage de l'I2C ne rentre pas en conflit avec le câblage du bus SPI utilisé par l'écran.

On utilise alors le démon **pikeyd** **[30]** déjà présent et configuré dans Raspicade. Ce programme résident permet de transformer notre panel arcade en clavier. Ainsi l'appui sur un bouton enverra une lettre à l'émulateur comme si l'on avait appuyé sur une touche du clavier. La configuration utilise les touches par défaut de l'émulateur MAME. Il suffit ensuite d'activer le démon **pikeyd** au démarrage de la distribution en éditant le fichier **/etc/rc.local** comme suit :

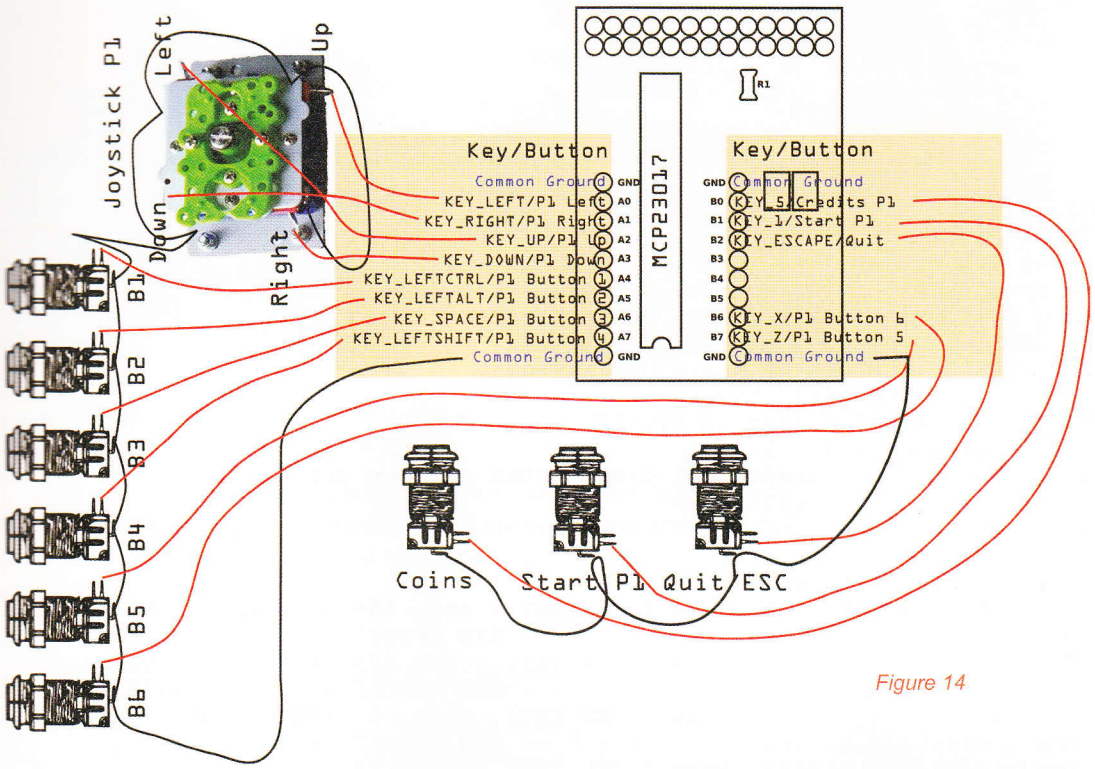


Figure 14

```
#Uncomment to launch pikeyd deamon
to support mcp modules
/home/pi/pikeyd/pikeyd -d
```

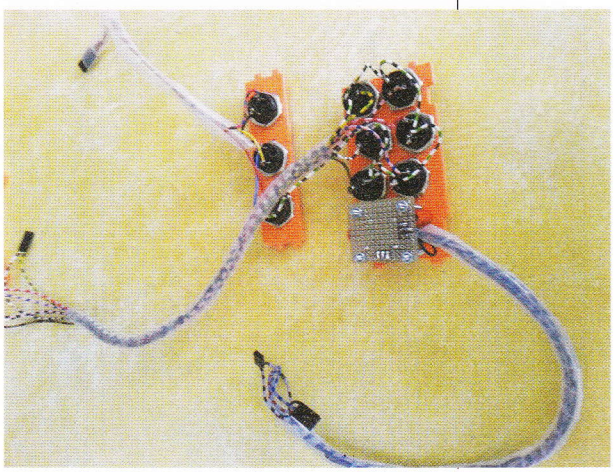
Le fichier de configuration se nomme **/etc/pikeyd.conf** :

```
MCP23017 has two 8-bit banks
# interrupt GPIO4
XIO_A              4/0x20/MCP23017A
XIO_B              4/0x20/MCP23017B
# port A
KEY_LEFT           XIO_A:0
KEY_RIGHT          XIO_A:1
KEY_UP             XIO_A:2
KEY_DOWN           XIO_A:3
KEY_LEFTCTRL       XIO_A:4
KEY_LEFTALT        XIO_A:5
KEY_SPACE          XIO_A:6
KEY_LEFTSHIFT      XIO_A:7
# port B
KEY_Z              XIO_B:0
KEY_X              XIO_B:1
KEY_C              XIO_B:2
KEY_V              XIO_B:3
KEY_BACKSPACE        XIO_B:4
KEY_ESC              XIO_B:5
KEY_1              XIO_B:6
KEY_5              XIO_B:7
```

On câblera ensuite les boutons sur les connecteurs du module à l'aide du schéma suivant de la figure 14. Ce dernier correspond à la configuration effectuée dans le fichier **pikeyd.conf**.

Le câblage d'un panel arcade est assez simple. La seule chose un peu délicate est peut-être la tresse de masse sur l'ensemble des composants du panel (Figure 15).

Figure 15



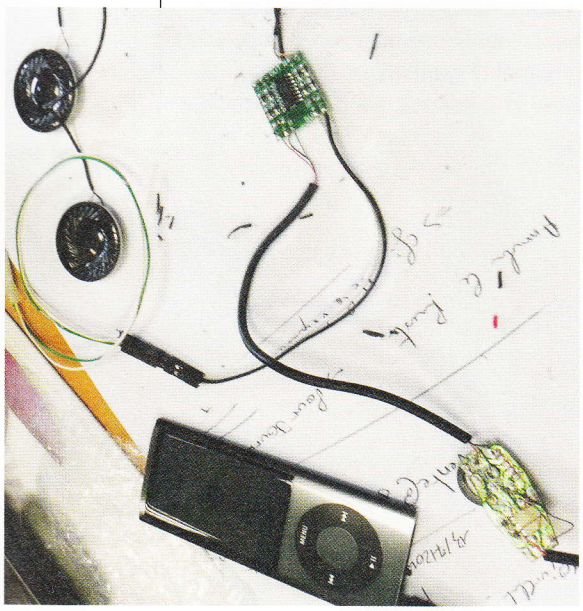
Une fois le panel câblé complètement, il sera possible de tester son bon fonctionnement à l'aide de la commande **evtest** dans une console. La commande s'utilise comme suit :

```
pi@raspicade ~ $ evtest
No device specified, trying to scan all of /dev/input/event*
Not running as root, no devices may be available.
Available devices:
/dev/input/event0: CHICONY HP Basic USB Keyboard
/dev/input/event1: pikeydx32-input
Select the device event number [0-1]: 1
Input driver version is 1.0.1
Input device ID: bus 0x3 vendor 0x1 product 0x1 version 0x1
Input device name: "pikeydx32-input"
Supported events:
.....
Testing ... (interrupt to exit)
Event: time 1401556172.000297, type 1 (EV_KEY), code 105 (KEY_LEFT), value 1
Event: time 1401556172.000297, -------------- SYN_REPORT ------------
Event: time 1401556172.131992, type 1 (EV_KEY), code 105 (KEY_LEFT), value 0
Event: time 1401556172.131992, -------------- SYN_REPORT ------------
Event: time 1401556193.163464, type 1 (EV_KEY), code 106 (KEY_RIGHT), value 1
Event: time 1401556193.163464, -------------- SYN_REPORT ------------
Event: time 1401556193.283563, type 1 (EV_KEY), code 106 (KEY_RIGHT), value 0
Event: time 1401556193.283563, -------------- SYN_REPORT ------------
[...]
```

Un appui et relâchement de boutons envoie un événement et une valeur. L'appui sur chaque bouton et direction du joystick vous permettra de contrôler le bon fonctionnement de notre panel.

Passons maintenant à la partie amplification audio.

Figure 16



2.4 Amplification audio et montage final

Pour des raisons de coût, on récupérera les haut-parleurs sur un casque micro PC usagé. Ceci nous permet aussi de récupérer le potentiomètre de volume de petite taille qui sera tout à fait adapté à notre micro borne (Figure 16).

Pour la partie amplification, je me suis tourné vers un amplificateur 3W/5V trouvé sur eBay pour moins de 2€ frais d'envoi inclus **[17]**. Il n'y a pas de difficultés particulières ici, on branchera cette partie audio sur le Raspberry Pi à l'aide d'une fiche Jack 3.5.

Ensuite le plus difficile est de faire rentrer tous les composants dans la petite boîte de notre borne (Figure 17).

Afin de finaliser le montage, vous aurez besoin de visserie pour maintenir l'écran, le joystick ainsi que les haut-parleurs. La colle à chaud est une bonne alliée pour fixer proprement le bouton du potentiomètre de volume sur la partie arrière de la borne.

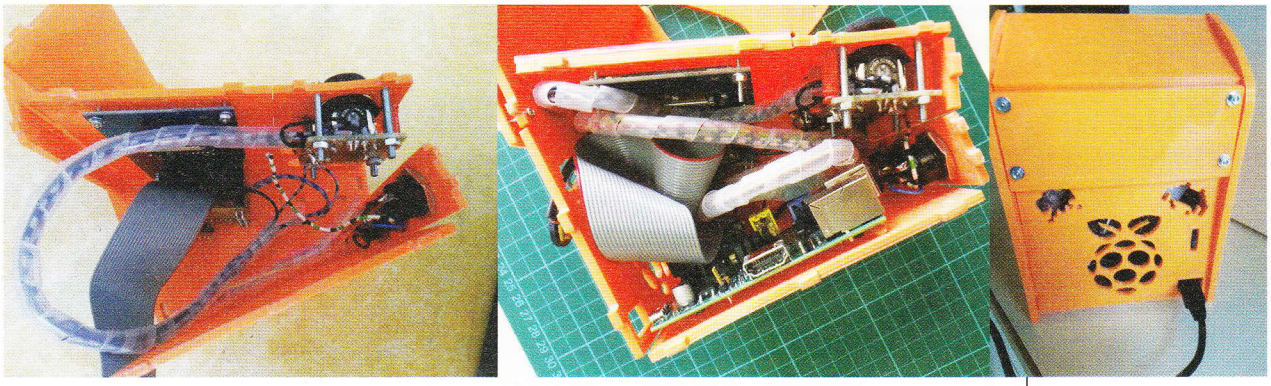


Figure 17

Le « dustwasher » du joystick peut être facilement réalisé à l'aide d'un petit morceau de gaine thermorétractable (Figure 18).

Vous trouverez une vidéo de la borne en fonctionnement à l'adresse **[31]**. Nous pouvons maintenant passer à la version 2, qui sera esthétiquement un peu plus jolie, car les lumières des ergots de fixations seront borgnes.

Figure 18

3. VERSION 2 : BORNE À BASE DE PI B+

Cette seconde borne utilisera un Raspberry Pi B+, possédera un écran peu documenté acheté sur un site chinois et utilisera le port GPIO pour l'ensemble des entrées nécessaires à nos contrôles arcade.

3.1 Configuration pour le support de l'écran TFT

Les petits écrans LCD pour Raspberry Pi sont maintenant légion. On en trouve partout sur Internet à des tarifs allant du simple au quadruple. On peut considérer que le prix d'achat reflète plus le support et la facilité de mise en œuvre que la qualité du produit lui-même. Dans le cas d'un bas coût, vous aurez au mieux une image d'un système prête à être écrite sur une carte SD et qui vous permettra de faire fonctionner votre écran rapidement. Cependant, vous n'aurez pas le choix de la distribution et il vous sera impossible de l'exploiter avec un autre système. De plus, l'image sera la plupart du temps remplie de logiciels inutiles installés par le fabricant. Mais tout n'est pas perdu, grâce au développeur Notro **[32]** qui réalise les pilotes linux framebuffer pour les petits écrans TFT : FBTFT. Son travail a été intégré dans les noyaux récents et le support des écrans est directement accessible.

L'écran utilisé pour notre projet est un Waveshare 3.2" TFT (*thin-film transistor*) LCD. Il est supporté dans les dernières versions du noyau par les pilotes fbtft **[33]**. Nous utiliserons la configuration sous le nom « waveshare32b ». Si vous avez un écran différent de celui-ci, vérifiez qu'il est supporté par le pilote en consultant la page **[33]**, sinon il vous faudra récupérer les informations précises pour sa configuration spécifique.

Les spécifications de cet écran sont les suivantes :

LCD Type	TFT
LCD Interface	SPI
Touch Screen Type	Resistive
Touch Screen Controller	XPT2046
Colors	65536
Backlight	LED
Resolution	320*240 (Pixel)
Aspect Ratio	4:3

On remarque qu'il utilise le bus SPI et monopolise les broches suivantes sur le GPIO du Raspberry Pi :

PIN NO.	SYMBOL	DESCRIPTION
1, 17	3.3V	Power positive (3.3V power input)
2, 4	5V	Power positive (5V power input)
6, 9, 14, 20, 25	GND	Ground
11	TP_IRQ	Touch Panel interrupt, low level while the Touch Panel detects touching
12	KEY1	Key
13	RST	Reset
15	LCD_RS	LCD instruction control, Instruction/Data Register selection
16	KEY2	Key
18	KEY3	Key
19	LCD_SI / TP_SI	SPI data input of LCD/Touch Panel
21	TP_SO	SPI data output of Touch Panel
23	LCD_SCK / TP_SCK	SPI clock of LCD/Touch Panel
24	LCD_CS	LCD chip selection, low active
26	TP_CS	Touch Panel chip selection, low active

Comme sur la borne précédente, nous ne configurerons pas l'écran tactile. Cependant, il est supporté et configurable. Je vous renvoie à l'adresse **[34]** pour plus de détails à ce sujet.

Ici, la configuration de l'écran sera manuelle. En effet, il n'y a pas, comme dans le cas de l'écran Adafruit, d'utilitaire spécifique pour faire le travail à notre place. Nous aurons plusieurs fichiers à modifier : `cmdline.txt`, `config.txt` et `/etc/modules`.

On commence par le fichier `/boot/cmdline.txt`. On ajoute la configuration de l'écran à la ligne existante :

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 root=/dev/mmcblk0p2
rootfstype=ext4 elevator=deadline rootwait console=tty2 loglevel=3 logo.
nologo quiet fbtft_device.custom fbtft_device.name=waveshare32b fbtft_
device.gpios=dc:22,reset:27 fbtft_device.bgr=1 fbtft_device.speed=48000000
fbcon=map:10 fbcon=font:ProFont6x11 logo.nologo dma.dmachans=0x7f35
console=tty1 consoleblank=0 fbtft_device.fps=60 fbtft_device.rotate=90
```

HACKABLE MAGAZINE

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS !

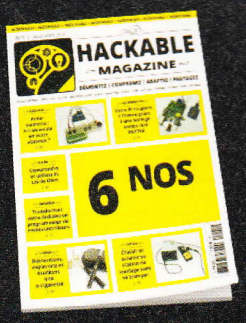
PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

www.ed-diamond.com

LES COUPLAGES PAR SUPPORT :

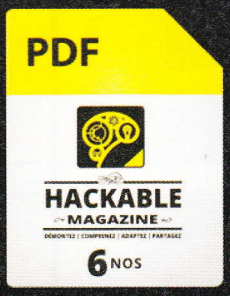
VERSION
PAPIER

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



VERSION
PDF

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



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

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

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

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

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

HACKABLE MAGAZINE

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

Vos remarques :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC HACKABLE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Prix en Euros / France Métropolitaine

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

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

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

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

Cet ajout permet de passer les paramètres de configuration de notre écran au module noyau `btft_device` au chargement du système.

On commence par la variable `fbtft_device.custom` qui permet de spécifier une configuration pour un écran non supporté nativement par le module. La variable `fbtft_device.name` permet ensuite de spécifier le pilote à utiliser.

L'option `btft_device.gpios` permet de spécifier les GPIO et leurs utilisations. Ici, « dc - Data/Command » (ou RS) se trouve sur le GPIO 22 (broche 13 du tableau précédent), et le « Reset » sur le 27 (broche 15 du tableau).

L'option `fbtft_device.bgr` définit le bit BGR qui permet d'inverser les couleurs rouge et bleu de l'écran.

L'option `fbtft_device.speed` définit la vitesse du bus SPI en hertz.

L'option `fbtft_device.rotate` définit le sens d'affichage de l'écran par une rotation dans le sens horaire : `0` = 0°, `1` = 90°, `2` = 180° et `3` = 270°.

L'option `fbtft_device.fps` fixe le nombre d'images par seconde. Ici 60, pour un rafraîchissement de l'écran à 60Hz, la valeur par défaut est 20.

L'option `fbcon=map:10`, permet d'affecter une séquence de framebuffer aux consoles disponibles sur le système. La séquence `10` sera répétée sur l'ensemble des consoles. Ainsi, la console 1 (tty1) sera affichée sur le framebuffer 1, la console 2 (tty2) sur le frambuffer 0, la console 3 (tty3) sur le framebuffer 1 et ainsi de suite. Comme dans le cas de l'écran Adafruit, nous allons copier le framebuffer 0 sur le framebuffer 1 de l'écran TFT, on commence donc la séquence par le framebuffer 1.

L'option `console=tty1` fixe la console affichée par défaut. L'affichage se fera donc sur l'écran TFT qui est sur le framebuffer 1. Si nous avions inversé l'option `fbcon=map:01`, il aurait fallu choisir la console 2 (tty2) par défaut pour afficher sur l'écran TFT.

L'option `fbcon=font:ProFont6x11` permet de définir la police de caractères affichée dans la console.

L'option `dma.dmachans=0x7f35` permet de spécifier un canal DMA libre pour l'écran.

Dans notre cas, l'écran est supporté et on peut s'affranchir de la configuration spécifique.

On passe ensuite à l'édition du fichier `config.txt` pour la configuration de la résolution de notre écran. On y ajoute les lignes suivantes :

```
gpu_mem=256
#same resolution for hdmi and tft
hdmi_force_hotplug=1
hdmi_cvt=320 240 60 1 0 0 0
hdmi_group=2
hdmi_mode=87
#overclocking
arm_freq=1000
core_freq=500
sdram_freq=500
over_voltage=2
#spi for tft
dtparam=spi=on
```

La directive `gpu_mem` permet de spécifier la quantité de mémoire allouée à la partie graphique. Elle est importante, car EmulationStation requiert une grande quantité de mémoire pour l'affichage des images des jeux lors de leur présentation.

L'option `hdmi_force_hotplug=1` force l'utilisation de la sortie HDMI même si celle-ci n'est pas branchée.

Les options `hdmi_cvt=320 240 60 1 0 0 0`, `hdmi_group=2` et `hdmi_mode=87` permettent de configurer la sortie HDMI (framebuffer 0) à la bonne résolution (320X240).

Les options `arm_freq=1000`, `core_freq=500`, `sdram_freq=500`, `over_voltage=2` réalisent l'« overclocking » du système.

Finalement, l'option `dtparam=spi=on` active le bus SPI sur le Raspberry Pi. Cette dernière option peut aussi être configurée par l'intermédiaire du programme `raspi-config`.

Il ne nous reste plus qu'à configurer le fichier `/etc/modules`. On y ajoute les lignes suivantes avec une valeur de départ pour la vitesse du bus SPI :

```
fbtft dma
fbtft_device name=waveshare32b gpios=dc:22,reset:27 speed=48000000
```

Avec cette vitesse de transfert, il est possible que certaines animations, notamment dans EmulationStation vous paraissent saccadées. Il faut alors augmenter progressivement la vitesse. Voici la valeur maximale qui a fonctionné pour moi :

```
fbtft dma
fbtft_device name=waveshare32b gpios=dc:22,reset:27 speed=82000000
```

L'écran est maintenant configuré, mais il reste à réaliser la copie du framebuffer 0 (sortie HDMI) vers le framebuffer 1 (écran TFT en SPI). La procédure utilisant le programme `fbcp` est identique à celle décrite dans le paragraphe 2.1.

Après cette étape, votre écran est fonctionnel. Passons maintenant au support de nos contrôles arcade par le GPIO.

3.2 Modification du programme retrogame

Pour la réalisation pratique du panneau de contrôle (Panel Arcade), la procédure est identique à celle décrite dans les pages précédentes. La différence réside dans la gestion des contrôles. Nous disposons d'un Raspberry Pi B+ qui comprend un connecteur GPIO de 40 broches. Ce dernier contient assez de connexions pour gérer l'écran TFT sur le bus SPI et l'ensemble des entrées de notre panel Arcade.

Dans la suite, nous allons utiliser le programme `retrogame` d'Adafruit **[35]** qui permet de gérer les entrées du GPIO comme s'il s'agissait d'appuis sur les touches d'un clavier. Grâce à ce programme, l'appui d'un bouton de notre panel se transformera par l'envoi d'un caractère comme s'il était issu d'un clavier. Il sera donc possible de piloter les émulateurs par cette technique.

Cependant, le programme initial ne gère que 8 contrôles : les 4 directions du joystick et 4 boutons. De plus, il ne prend pas en compte les spécificités de notre écran au niveau des connexions sur le port GPIO.

Nous devons donc le modifier afin de respecter le schéma de câblage donné en figure 19.

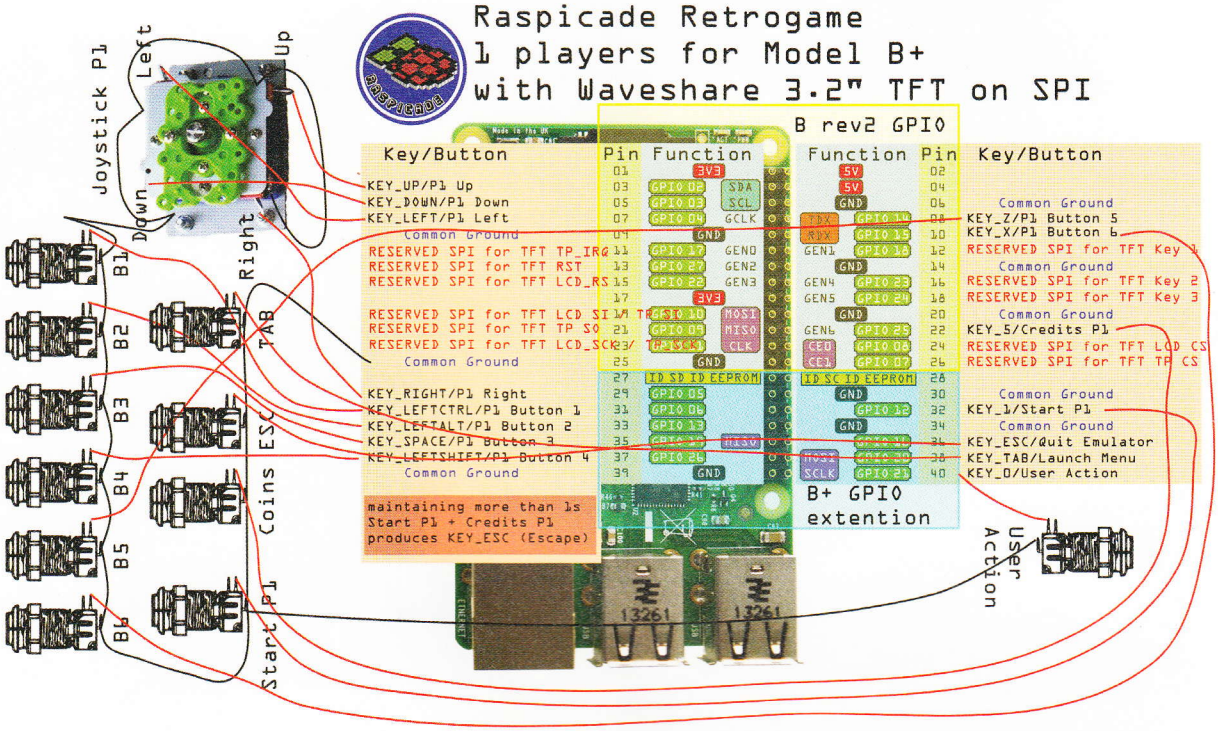


Figure 19

Pour cela, on réalise les modifications suivantes dans le fichier **retrogame.c** :

```
  ioStandard[] = {
// This pin/key table is used if an Adafruit PiTFT display
// is detected (e.g. Cupcade or PiGRRL).
// Input   Output (from /usr/include/linux/input.h)
//Player 1 config
{  2,     KEY_UP        },   // Up
{  3,     KEY_DOWN      },   // Down
{  4,     KEY_LEFT      },   // Left Joystick (4 pins)
{  5,     KEY_RIGHT     },   // Right
{  6,     KEY_LEFTCTRL  },   // Button 1
{ 13,     KEY_LEFTALT   },   // Button 2
{ 19,     KEY_SPACE     },   // Button 3
{ 26,     KEY_LEFTSHIFT  },   // Button 4
{ 14,     KEY_Z  },   // Button 5
{ 15,     KEY_X  },   // Button 6
{ 25,     KEY_1  },   // Button Start P1
{ 12,     KEY_5  },   // Button Coins/Credits P1
// Button to halt system on pin 15 -> sudo halt
{ 16,    KEY_ESC },    // Button to exit emulators and programs
{ 20,     KEY_TAB },    // Button to show mame menuCoins/Credits P2
{ 21,    KEY_0 },    // Button to user action
// For credit/start/etc., use USB keyboard or add more buttons.
{  -1,     -1           } }; // END OF LIST, DO NOT CHANGE
```

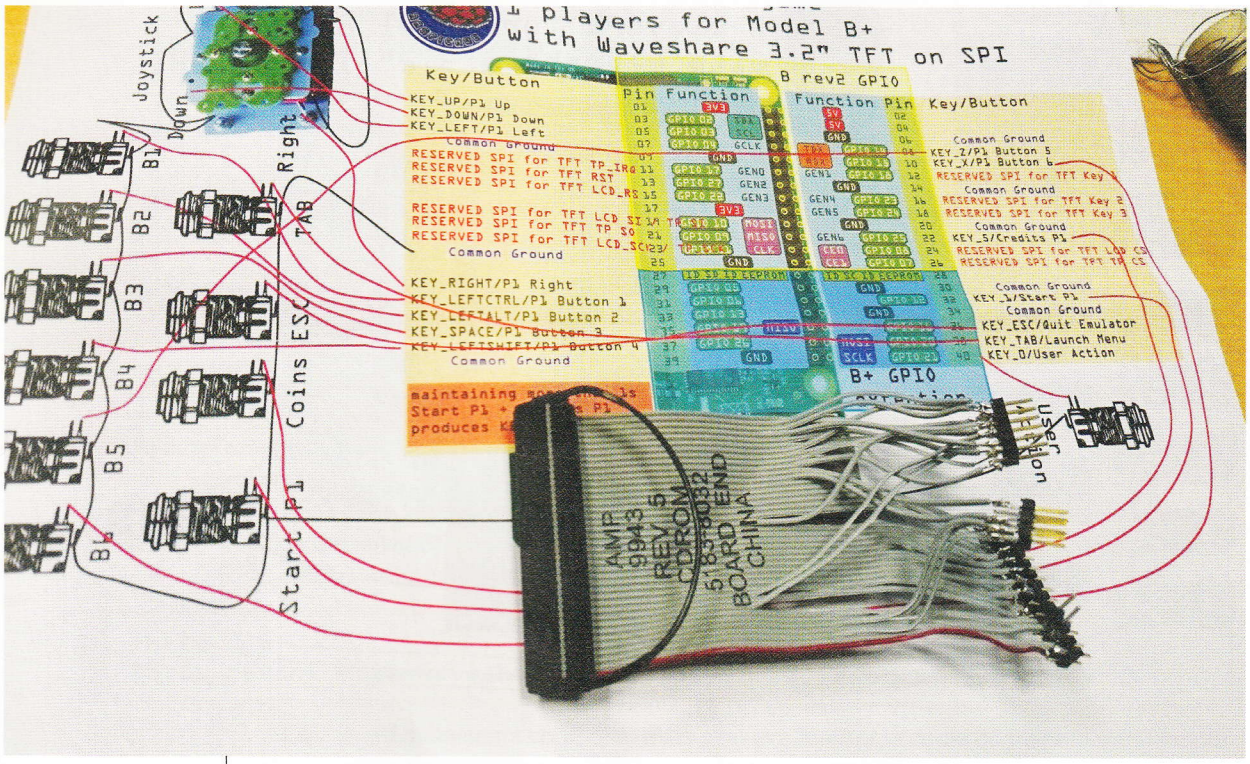


Figure 20

Il s'agit essentiellement de la réaffectation des broches du connecteur GPIO. Le programme complet est disponible ainsi que sa documentation à l'adresse **[36]**.

Une fois le panel complètement câblé, le programme **retrogame** installé, compilé et en exécution (lire la page **[36]** pour plus d'informations à ce sujet), on peut tester son bon fonctionnement à l'aide de la commande **evtest**.

Concernant la partie audio, les mêmes composants ont été utilisés, la procédure est donc identique à la précédente.

3.3 Montage et finitions

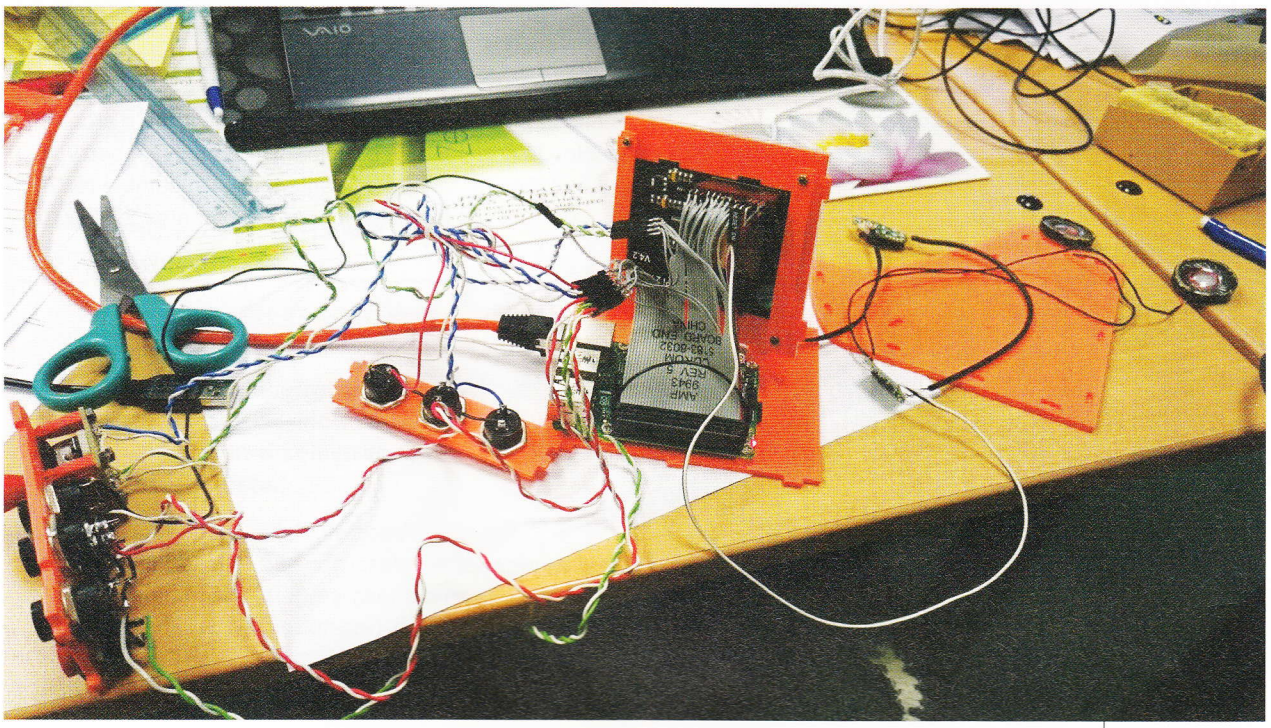
Pour le montage, je vous renvoie vers la page **[37]** qui regroupe un grand nombre de photos retraçant l'assemblage des différents composants.

Il faudra encore une fois vous armer d'un bon fer à souder et de beaucoup de patience... En effet, il va falloir séparer les fils d'une nappe IDE 40 fils afin de créer la connectique pour l'écran TFT et celle des contrôles du panel comme montré en figure 20.

Il est important de conserver une longueur de nappe la plus courte possible. En effet, la longueur des connexions influe fortement sur la stabilité de l'affichage sur l'écran TFT.

Une fois le tout connecté, il faut vérifier l'ensemble du câblage à l'aide du Raspberry Pi en fonctionnement (Figure 21).

Figure 21



Ensuite, on ferme la caisse de la petite borne après avoir tout fait entrer à l'intérieur (Figure 22).

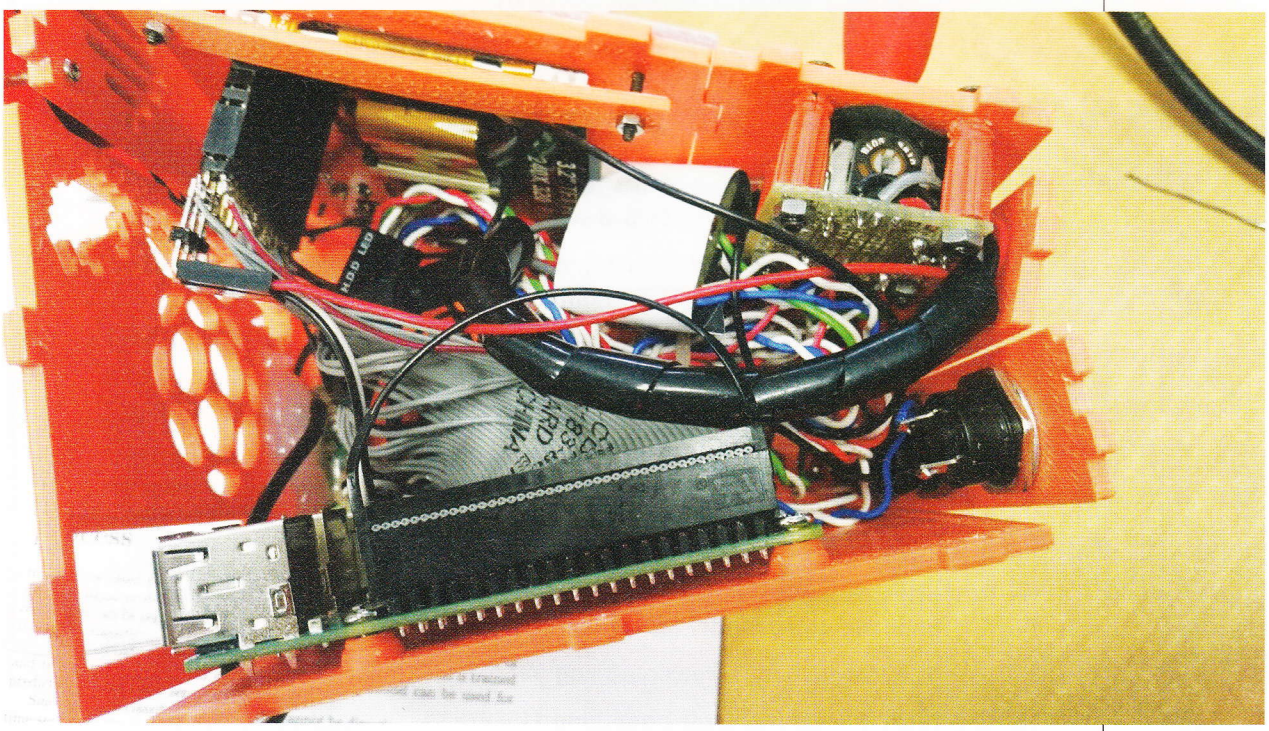


Figure 22

On rebranche le tout et c'est parti pour une partie effrénée de Galaga ou 1942...

CONCLUSION

Nous avons terminé cet article traitant de la création complète d'une petite borne d'arcade autour du Raspberry Pi. Mis à part les quelques composants principaux (écran, Raspberry Pi), tout a été réalisé avec des moyens assez réduits. Malgré sa petite taille, l'ensemble reste jouable, et très facilement transportable... Alors, à vos fers à souder et imprimantes 3D ! YM

RÉFÉRENCES

[1] http://blog.petrockblock.com/retropie/

[2] http://piplay.org/

[3] http://www.recalbox.com/

[4] http://happi-game-center.com/

[5] https://sourceforge.net/projects/raspicade/

[6] https://superpiboy.wordpress.com/

[7] https://learn.adafruit.com/super-game-pi

[8] https://learn.adafruit.com/raspberry-gear

[9] http://www.thingiverse.com/search/page:1?q=handheld+game&sa=

[10] https://www.adafruit.com/products/1783

[11] http://www.instructables.com/id/Micro-Raspberry-Pi-arcade-cabinet/

[12] http://www.starforcepi.com/overview.html

[13] https://www.adafruit.com/products/1601

[14] http://www.banggood.com/3_2-Inch-TFT-LCD-Display-Module-Touch-Screen-For-Raspberry-Pi-B-B-A-p-1011516.html

[15] http://www.banggood.com/Original-3D-Analog-Joystick-Controller-Module-For-Xbox-360-PS2-p-931300.html

[16] http://www.banggood.com/3A-250V-Off-on-Non-locking-Momentary-Push-Button-Switch-p-915925.html

[17] http://www.ebay.fr/itm/Module-amplificateur-audio-PAM8403-2x3W-classe-D-2-canaux-DC-5V-/201516186755?hash=item2eeb4cfc83

[18] https://github.com/mmoller2k/pikeyd

[19] https://github.com/ian57/Nanocab-Pi1-Pi2-Retrogame

[20] https://learn.adafruit.com/system/assets/assets/000/017/715/original/cupcade-cab-art.pdf?1404184503

[21] http://www.thingiverse.com/thing:926435

[22] http://www.thingiverse.com/thing:1158875

[23] https://sourceforge.net/projects/raspicade/files/Pi1-B512-B%2B/raspicade-samba-gamepads-keyboard-pikeyd_ready-B-B%2B-GPIOready-ES-20150513.img.gz/download

[24] https://sourceforge.net/p/raspicade/wiki/HowTo/

[25] https://learn.adafruit.com/adafruit-pitft-28-inch-resistive-touchscreen-display-raspberry-pi/software-installation

[26] https://learn.adafruit.com/adafruit-pitft-28-inch-resistive-touchscreen-display-raspberry-pi/touchscreen-install-and-calibrate

[27] https://learn.adafruit.com/cupcade-raspberry-pi-micro-mini-arcade-game-cabinet/build-interface-circuit

[28] http://www.instructables.com/id/Simple-Analog-to-Digital-joystick-conversion/

[29] http://atmjoy.com/pikeyd-pi-gpio-keyboard-daemon-i2c-mcp3017/

[30] https://github.com/mmoller2k/pikeyd

[31] https://youtu.be/WCU2KrV0N-M

[32] https://github.com/notro/fbtft

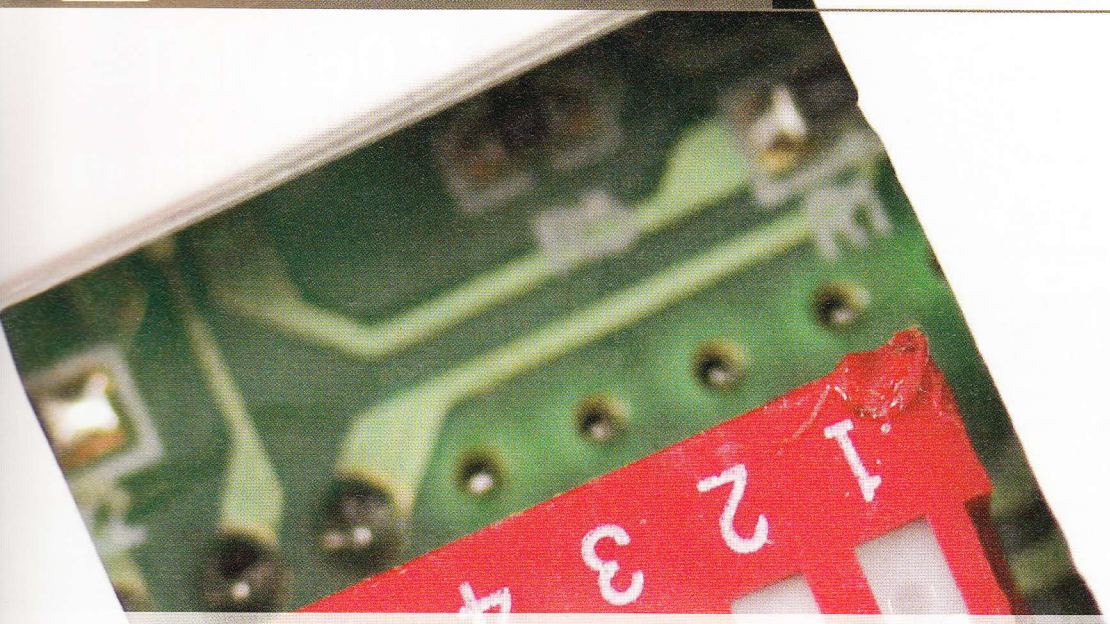
[33] https://github.com/notro/fbtft/blob/master/fbtft_device.c

[34] http://www.circuitbasics.com/setup-lcd-touchscreen-raspberry-pi/

[35] https://github.com/adafruit/Adafruit-Retrogame

[36] https://github.com/ian57/Nanocab-Pi1-Pi2-Retrogame

[37] http://www.gamoover.net/Forums/index.php?topic=34507.0



REMPLACER UNE TÉLÉCOMMANDE À FRÉQUENCE PAR UN MONTAGE ARDUINO : COLLECTE DES DONNÉES

Denis Bodor

Nous avons déjà par le passé, dans les pages de ce magazine, fait connaissance avec le domaine des transmissions par radiofréquences en « écoutant », par exemple, une télécommande de garage afin d'en déduire la configuration. Nous allons pousser ici cela un cran plus loin et voir qu'avec une carte Arduino complétée d'un module d'émission 433 Mhz, il est parfaitement possible de remplacer une ou plusieurs télécommandes génériques de ce type.

Les expérimentations ayant conduit à la réalisation de cet article ont débuté par hasard. Comme cela arrive souvent, en cherchant tout autre chose, je suis tombé sur les entrailles d'un autre équipement, en l'occurrence d'une télécommande désossée. Une fois encore, le bon vieux principe du « ne jette pas, un jour ça servira » s'est pleinement justifié. En effet, ce reste de télécommande était composé de deux circuits de formes et de fabrications fort différentes : une base comprenant les contacts pour les boutons, un microcontrôleur inconnu (type COB ou *Chip On Board*, sous un dôme de résine) et les supports de piles, et un module soudé provenant clairement d'une source différente, avec un circuit intégré, un oscillateur à quartz, une antenne et quelques composants passifs (condensateurs et résistances).

Mais ce fut surtout la sérigraphie des circuits imprimés, au point d'interconnexion, qui constitua l'élément déclencheur : FSK, NIRQ, NSEL, SDI, SCK, BATT, GND. Il s'agissait donc d'un module émetteur interfacé en SPI, piloté par le microcontrôleur de la base et donc susceptible d'être utilisé séparément. Un rapide coup d'œil au circuit intégré indiqua qu'il s'agissait d'un Si4021 de chez Silicon Labs, et plus précisément un émetteur multi-canaux FSK pouvant travailler sur les bandes ISM 315 Mhz, 433 Mhz, 868 Mhz et 915 Mhz.

ISM pour « Industriel, Scientifique et Médical », correspond aux bandes de fréquences ne nécessitant pas de licence préalable. La bande ISM 433 Mhz en particulier, en Europe, couvre les fréquences entre 433,05 Mhz et 434,79 MHz et est généralement utilisée pour bon nombre d'applications du carillon sans fil aux capteurs de température des stations météo et réveils matin en passant par les télécommandes pour prises de courant, les jouets télécommandés, les baby phones... Partant du principe que la fréquence utilisée devait très certainement être 433 Mhz et donc l'antenne adaptée en conséquence, il était clair que ce circuit pouvait être réutilisé.

À l'origine de cet article se trouve ceci, les restes d'une télécommande désossée aux origines plus que mystérieuses. Deux éléments attirent immédiatement l'attention : le module vert est clairement d'une fabrication différente de la base et la sérigraphie des deux circuits ne laisse presque planer aucun doute, il s'agit d'une interface SPI.

La documentation technique du Si4021 recommande une utilisation conjointe avec le Si4020, récepteur du même fabricant, tout en précisant que le module est parfaitement adapté à la transmission d'un signal en FSK et OOK. FSK, pour *Frequency-Shift Keying* est une modulation par déplacement de fréquence où une information est codée de manière à ce que le signal varie entre des fréquences prédéterminées. Une pour signifier un 0 et une autre pour un 1, par exemple.

OOK, pour *On-Off Keying*, en revanche, ou ASK/OOK, est une modulation d'amplitude, beaucoup plus simple à mettre en œuvre puisque c'est la simple présence ou non d'une émission qui encode l'information à transmettre (exactement

Le plus souvent, il n'est même pas utile de parcourir plusieurs bandes de fréquences pour trouver le signal émis par le matériel. La fréquence est directement spécifiée sur la télécommande, sur les récepteurs ou sur les deux. Ici, il est clairement fait mention des 433,92 Mhz utilisés.

comme pour le code morse). C'est souvent cette modulation qui est utilisée pour les télécommandes, qu'il s'agisse de portes de garage ou de prises de courant, en particulier pour les systèmes d'entrée de gamme.

Le si4021 demande une configuration particulière puisque c'est un circuit intégré très complet se chargeant de la génération de fréquences, de la synthèse du signal, de l'amplification, et même de quelques fonctions particulières permettant de l'utiliser comme un composant autonome (mode EEPROM). Son successeur le si4432 va encore plus loin en faisant office d'émetteur-récepteur (*transceiver*), davantage orienté vers la communication de données.

Dans le cadre de cet article cependant, il est préférable, à défaut de module équivalent, de s'orienter vers des émetteurs 433 Mhz très économiques (~3€). En effet, l'objectif ici est d'utiliser l'ASK/OOK, ce qui se résume à contrôler l'émission du signal en fonction de l'état d'une broche, et ce avec des délais précis. Les modules économiques, vendus généralement en paires émetteur/récepteur, utilisent précisément ce principe de fonctionnement (et celui-là uniquement, contrairement au si4021).

Si vous disposez d'un module à Si4021 (récupéré ou acheté), sa configuration détaillée ici vous concernera. Dans le cas d'un module économique, il vous suffira de sauter cette partie, le reste de l'article étant parfaitement utilisable tel quel. Le Si4432 en revanche, disponible par exemple sous forme de modules sur eBay pour quelques 10€, ne sera pas traité, car fonctionnellement bien au-delà de nos besoins.

1. RTL-SDR : PETIT RAPPEL

Le périphérique de réception utilisé dans la suite de mes explications a déjà fait l'objet de plusieurs articles dans le magazine. Il s'agit d'un équipement USB initialement destiné à la réception de la TNT (DVB-T). Cependant, comme il est uniquement constitué d'un récepteur/tuner et d'un convertisseur analogique/numérique, il peut être utilisé pour recevoir toutes sortes d'émissions radio, des stations FM commerciales aux *pagers* (POCSAG) en passant par les signaux de télécommandes, les informations transmises par les avions en vol (ADSB) ou encore des images provenant de satellites météo (NOAA).

Cette utilisation est appelée radio logicielle, car la majeure partie du traitement de signal est faite par un ordinateur, par opposition à la radio « classique » nécessitant un équipement dédié souvent très coûteux. Le composant principal de ces périphériques USB est le circuit intégré Realtek RTL2832U, vastement pris en charge par un grand nombre d'applications maintenant très matures et simples à utiliser comme GQRX (Mac/Linux), SDR# (Windows), HDSDR (Windows), Linrad (Linux), SDR Touch (Android), etc. Ceci, bien sûr, en plus du classique, incontournable et très pédagogique GNU Radio.

Ce type de produit est très courant et se retrouve aussi bien en supermarché qu'en magasin de bricolage ou d'aménagement intérieur. Ce modèle spécifique a été acheté il y a plusieurs années au Monoprix du coin pour une poignée d'euros.

Dans ce contexte d'utilisation, ces récepteurs sont généralement désignés sous le nom d'équipement RTL-SDR et fournissent une alternative très économique (~20€) à des appareils plus complets, mais plus coûteux comme HackRF, SDRPlay, BladeRF ou AirSpy. Notez également que, depuis plusieurs mois maintenant, il n'est plus nécessaire de prendre le risque de choisir un récepteur DVB-T qui pourrait s'avérer incompatible. En effet, des versions « garanties » RTL-SDR sont désormais disponibles, de meilleure facture et totalement compatibles, auprès de détaillants de matériels radioamateur.

Bien qu'il existe des logiciels très riches et intéressants sous Windows, dès lors qu'il s'agit d'utiliser des petits utilitaires spécifiques, la plateforme de choix reste GNU/Linux. Il pourra s'agir tout aussi bien d'un PC sous Debian ou Ubuntu, ou d'un système Raspbian sur une Raspberry Pi. Dans les deux cas, il conviendra d'installer les paquets adéquats avec :

```
$ sudo apt-get install rtl-sdr librtlsdr-dev cmake git
```

`librtlsdr0` installe une configuration permettant d'interdire le chargement automatique du module noyau permettant la prise en charge du récepteur comme tuner DVB-T (`/etc/modprobe.d/rtl-sdr-blacklist.conf`) ainsi qu'une autre, autorisant les utilisateurs standards (sans `sudo`) à utiliser le matériel (`/lib/udev/rules.d/60-librtlsdr0.rules`). Il est donc recommandé d'installer ce paquet **avant** la connexion du périphérique et juste après un démarrage. Ceci évitera que les modules soient chargés automatiquement, vous bloquant l'accès au matériel. Si nous avez branché le récepteur avant l'installation des paquets, redémarrez simplement votre Raspberry Pi (barbare, mais plus simple que de décharger les modules à la main avec `rmmod`).

Remarquez que nous installons également ici `git` et `cmake` permettant respectivement de récupérer les sources d'un utilitaire dont nous aurons besoin par la suite et de configurer ces mêmes sources pour leur compilation.

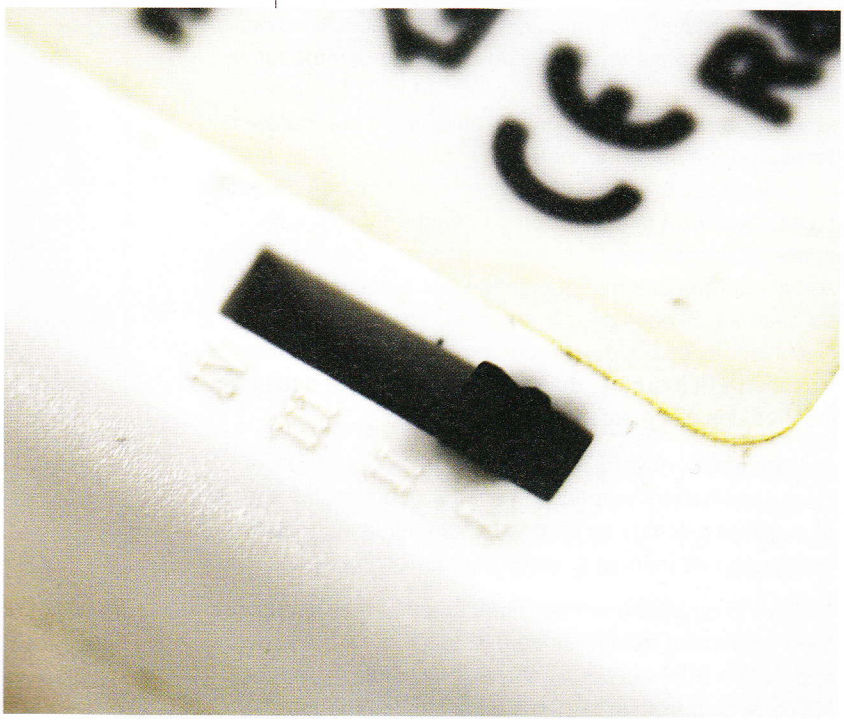
Sur PC ou avec une Pi utilisant une interface graphique, il est également recommandé d'installer `gqrx-sdr`, une application SDR qui pourra s'avérer fort utile en cas de difficulté pour trouver le signal de la télécommande.

2. ANALYSE DES PRISES RADIOCOMMANDÉES

Notre objectif ici consistera à découvrir l'information émise par une télécommande de prise électrique afin de l'utiliser avec notre module émetteur 433 Mhz. Ce genre de produit est relativement courant et on en trouve aussi bien en grande surface que dans les magasins de bricolage. Leur logique de fonctionnement est identique à celle de la télécommande Cardin dont nous avons parlé dans le numéro 6. Un message est encodé puis transmis sous la forme d'un signal modulé en amplitude. Ce message est démodulé puis décodé par les prises et celle correspondante change d'état (allumée/éteinte). Le message contient l'information sur la prise cible et le bouton pressé.

Il est généralement possible de régler la télécommande et les différents récepteurs sur un « canal » qui, en réalité, n'a rien à voir avec la fréquence utilisée, mais est une simple information embarquée dans le message. Ceci évite que vous et votre voisin ne contrôliez les mêmes appareils par inadvertance.

Les télécommandes destinées au contrôle de prises électriques disposent souvent d'un sélecteur permettant de choisir un « canal ». Il ne s'agit généralement pas d'une option de sélection d'une fréquence différente, mais d'une configuration impactant le message encodé qui est émis. Toutes les télécommandes utilisent donc en réalité la même fréquence, quel que soit le « canal » choisi.



Il existe plusieurs façons d'analyser le signal émis par la télécommande (cf. article sur la télécommande Cardin dans le numéro 6). Une utilisation combinée de GQRX, GNU Radio Companion et Audacity permet généralement d'arriver à ses fins par la voie que je qualifierais de pédagogique. Une solution plus rapide cependant consiste à utiliser un outil développé par Benjamin Larsson (alias merbanan) : *rtl_433*.

C'est Steve Markgraf qui a initialement diffusé sa création sur GitHub mars 2012, mais Benjamin a repris le flambeau et intégré les nombreuses contributions d'autres développeurs. Aujourd'hui, *rtl_433* est en mesure de détecter et interpréter près d'une cinquantaine de types de messages provenant d'émetteurs comme des télécommandes de prises électriques, des capteurs de température, des moniteurs de consommation de courant ou encore des stations météo.

De plus, *rtl_433* est en mesure de facilement afficher des informations d'analyse sur les messages captés par le récepteur RTL-SDR et c'est précisément ce dont nous allons nous servir. Pour installer l'outil, il nous suffit de procéder comme suit.

Nous commençons par récupérer les sources depuis GitHub en les plaçant dans un répertoire dédié :

```
$ mkdir SDR
$ cd SDR
$ git clone https://github.com/merbanan/rtl_433.git
```

Nous nous plaçons dans le répertoire des sources, créons un répertoire pour la construction/compilation du programme et utilisons `cmake` puis `make` :

```
$ cd rtl_433
$ mkdir build
$ cd build

$ cmake ../
$ make
[...]
Scanning dependencies of target data-test
[100%] Building C object tests/CMakeFiles/data-test.dir/data-test.c.o
Linking C executable data-test
[100%] Built target data-test
```

Dans le sous-répertoire `src/`, nous trouvons alors l'utilitaire directement utilisable sans installation :

```
$ cd src
$ ./rtl_433
[...]
Found 1 device(s):
  0:  Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Exact sample rate is: 250000.000414 Hz
Sample rate set to 250000.
Bit detection level set to 8000.
Tuner gain set to Auto.
Reading samples in async mode...
Tuned to 433920000 Hz.
```

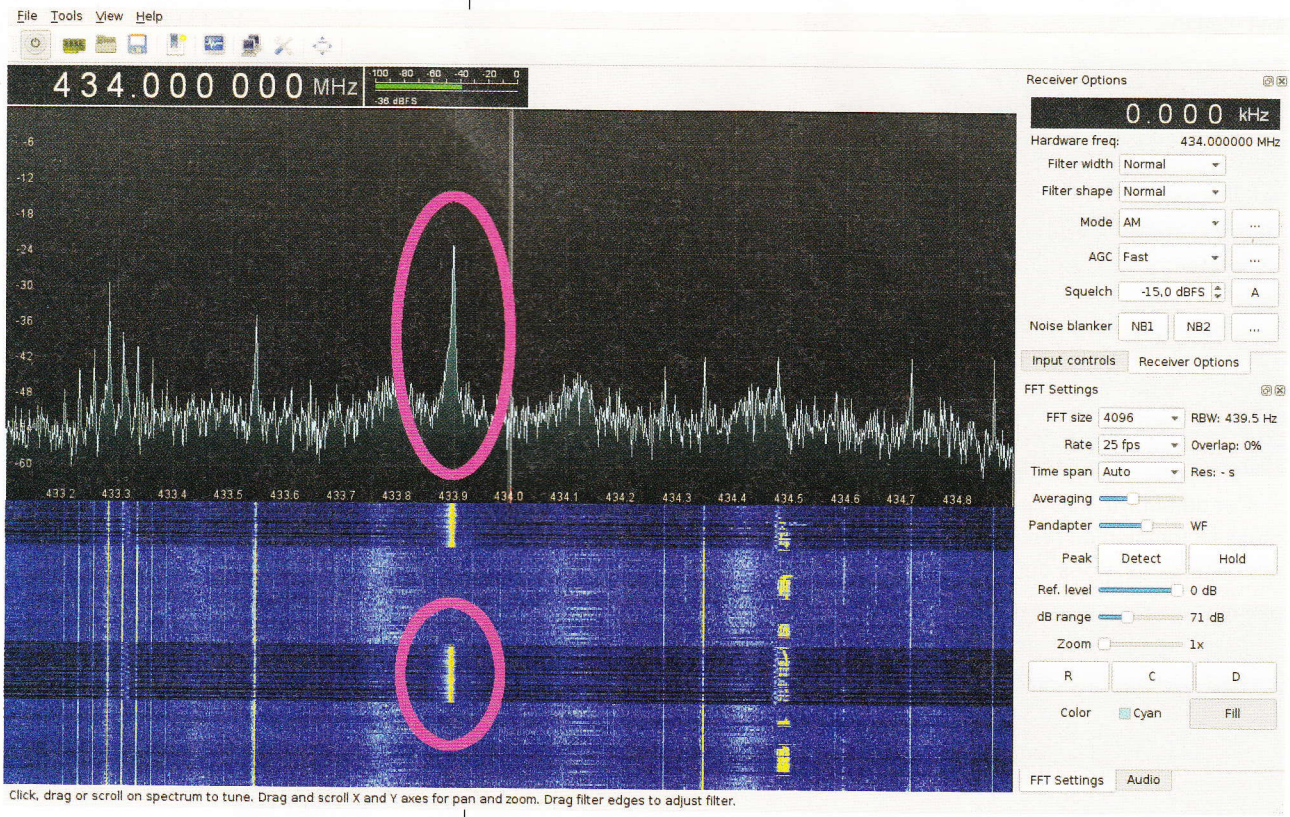
Par défaut, *rtl_433* se « cale » sur 433,92 Mhz et attend d'éventuels messages puis les interprète en fonction du résultat d'une détection automatique. Si nous appuyons sur un bouton de la télécommande, nous obtenons :

```
:        Waveman Switch Transmitter
        id:      0
        channel:         4
        button:  3
        state:   off
Generic remote keypress / sensor
ID 16bit = 0x1515
CMD 8bit = 0x55
TRISTATE = 0FFF0FFFFFFF
Generic remote keypress / sensor
ID 16bit = 0x1515
```

Le duo RTL-SDR et GQRX est bien utile, car même s'il ne permet pas de décoder ou d'interpréter le signal de la télécommande, il apporte son lot d'informations utiles comme la fréquence utilisée, la puissance du signal, ou encore le type de modulation à l'œuvre.

```
CMD 8bit = 0x57
TRISTATE = 0FFF0FFFFFF1
Generic remote keypress / sensor
ID 16bit = 0x1555
CMD 8bit = 0x57
TRISTATE = 0FFFFFFFFFF1
Generic remote keypress / sensor
ID 16bit = 0x1555
CMD 8bit = 0x57
TRISTATE = 0FFFFFFFFFF1
```

La détection automatique n'est pas parfaite et il est possible, comme ici, que plusieurs périphériques puissent correspondre au message reçu (*Waveman Switch Transmitter* et *Generic remote keypress*). En effet, pour ce genre de produit, il n'est pas rare que le même circuit soit utilisé sous plusieurs marques et modèles différents. Notre télécommande n'est pas de la marque Waveman, mais pourtant elle est détectée comme telle.



Mais ce qui nous intéresse davantage est le message lui-même et nous pouvons alors utiliser l'option **-A** pour déclencher l'analyseur d'impulsion :

```
Detected OOK package
:        Waveman Switch Transmitter
        id:      O
        channel:         3
        button:  4
        state:   off
Generic remote keypress / sensor
ID 16bit = 0x1551
CMD 8bit = 0x55
TRISTATE = 0FFFFF0FFFFF
Analyzing pulses...
Total count:   25,  width: 10319           (41.3 ms)
Pulse width distribution:
[ 0] count:   15,  width:   105 [104;107]  ( 420 us)
[ 1] count:   10,  width:   318 [317;319]  (1272 us)
Gap width distribution:
[ 0] count:   14,  width:   320 [319;321]  (1280 us)
[ 1] count:   10,  width:   107 [106;109]  ( 428 us)
Pulse period distribution:
[ 0] count:   24,  width:   425 [424;427]  (1700 us)
Level estimates [high, low]:  15915,    800
Frequency offsets [F1, F2]:   -5513,      0 (-21.0 kHz, +0.0 kHz)
Guessing modulation: Pulse Width Modulation with fixed period
Attempting demodulation... short_limit: 211,
  long_limit: 322, reset_limit: 322, demod_arg: 0
pulse_demod_pwm(): Analyzer Device
bitbuffer:: Number of rows: 1
[00] {25} ea ae aa 80 : 11101010 10101110 10101010 1
```

Énormément d'informations arrivent à l'écran et en particulier le décalage par rapport à la fréquence de base (ici -21 KHz), les différents délais utilisés et une représentation binaire du message. Nous savons donc maintenant, sans avoir à utiliser une application SDR comme GQRX que la télécommande utilise en réalité la fréquence 433,899 Mhz. Ceci n'est pas critique, car le plus souvent les récepteurs sont très tolérants aux décalages de fréquences.

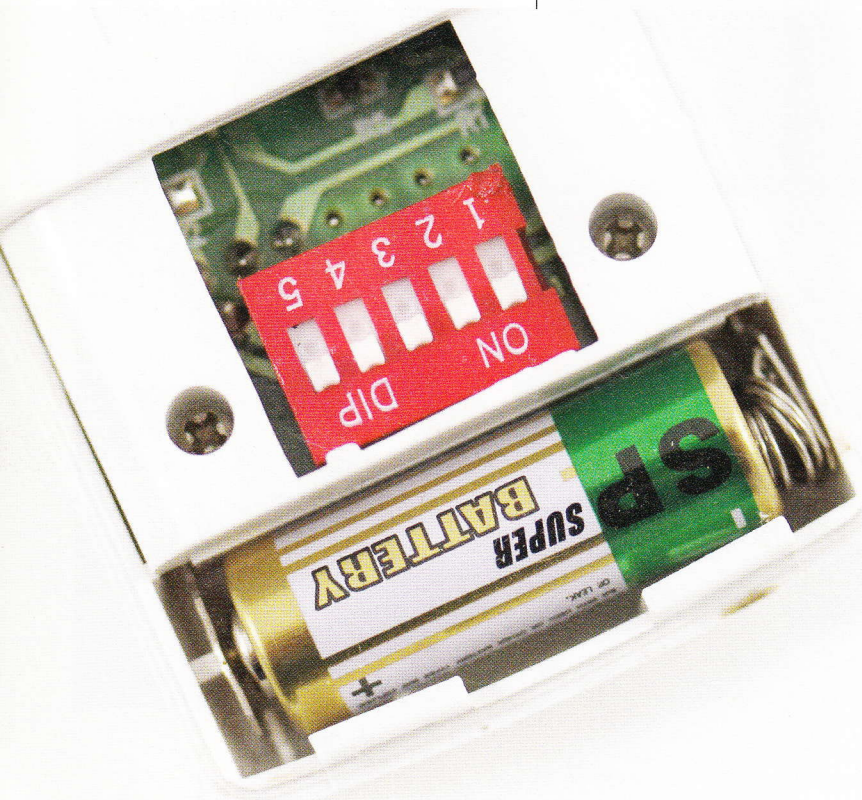
Nous utilisons ensuite l'option **-a** pour l'analyse de message (et désactiver l'interprétation) :

```
*** signal_start = 1901548, signal_end = 2027204
signal_len = 125656,  pulses = 200
Iteration 1. t: 216    min: 110 (118)    max: 323 (82)    delta 5
Iteration 2. t: 216    min: 110 (118)    max: 323 (82)    delta 0
Pulse coding: Short pulse length 110 - Long pulse length 323

Short distance: 102, long distance: 315, packet distance: 3291

p_limit: 216
bitbuffer:: Number of rows: 8
[00] {25} 15 51 55 00 : 00010101 01010001 01010101 0
[01] {25} 15 51 55 00 : 00010101 01010001 01010101 0
[02] {25} 15 51 55 00 : 00010101 01010001 01010101 0
```

```
[03] {25} 15 51 55 00 : 00010101 01010001 01010101 0
[04] {25} 15 51 55 00 : 00010101 01010001 01010101 0
[05] {25} 15 51 55 00 : 00010101 01010001 01010101 0
[06] {25} 15 51 55 00 : 00010101 01010001 01010101 0
[07] {25} 15 55 57 00 : 00010101 01010101 01010111 0
```

La sortie est plus compacte et surtout, la répétition du message due à une longue pression sur un bouton ne sature pas l'écran d'informations. Nous voyons ici clairement le message binaire envoyé, correspondant au bouton pressé. Pour collecter l'ensemble des données, il nous suffit alors de méticuleusement utiliser chaque bouton et copier/coller le message correspondant pour une utilisation ultérieure. Par acquis de conscience, on peut également répéter l'opération en configurant l'émetteur sur chacun des trois autres canaux pour obtenir un « lexique » complet, soit 32 messages en tout (4 prises, 2 boutons par prise et 4 canaux).

Des modèles plus récents de prises télécommandées utilisent une configuration plus poussée. Ici, sur la télécommande elle-même on peut voir une série de 5 micro-interrupteurs permettant de configurer le canal à utiliser. La pile placée juste en dessous est assez courante pour ce type d'équipement, un type A23 fournissant 12V. Ce modèle particulier (FHT-7901) utilise une modulation qu'il n'a pas été possible de reproduire avec un module de récupération, mais rtl_433 a toutefois correctement décodé les messages.



3. PRÊT POUR LA SUITE ?

Les différents messages récupérés vont nous permettre de rejouer ces transmissions, mais contrairement au simple rejeu consistant à enregistrer un signal et l'émettre à nouveau, nous allons moduler l'information en utilisant une carte Arduino et le module émetteur.

La technique consistant à enregistrer un signal et le rejouer n'est, en effet, utilisable qu'à condition de disposer d'un appareil SDR capable de recevoir et d'émettre, comme c'est le cas pour le HackRF One de Michael Ossmann ou son clone économique, le HackRF Blue. Dans notre cas, avec un simple récepteur DVB-T, nous ne pouvons pas réutiliser un signal enregistré. C'est possible dans l'absolu, mais ce serait bien plus compliqué que de nous baser sur les messages binaires. Technique traitée dans l'article qui suit. DB

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

HACKABLE MAGAZINE

PROFESSIONNELS !

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...

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

www.ed-diamond.com

PDF COLLECTIFS PRO

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

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

ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO OPEN SILICIUM

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

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

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

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

HACKABLE MAGAZINE

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

Vos remarques :

.....

.....

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

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

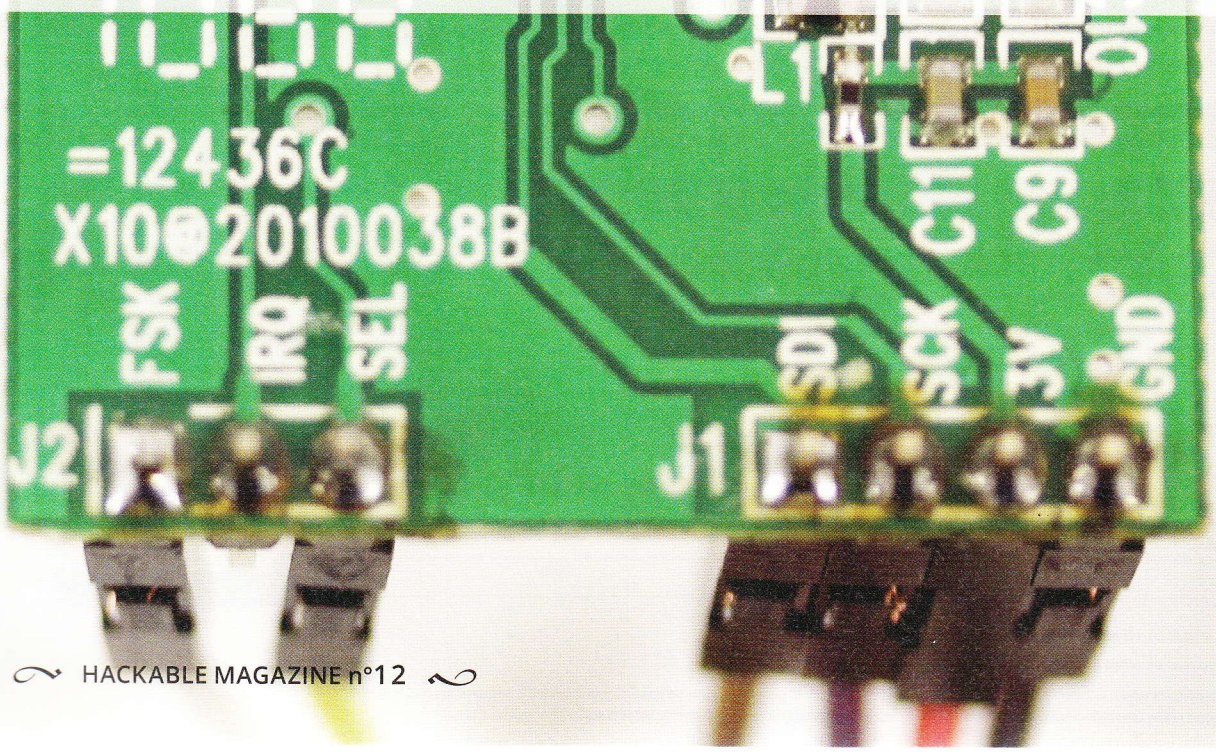
En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : boutique.ed-diamond.com/content/3-conditions-generales-de-ventes et reconnais que ces conditions de vente me sont opposables.

Prix TTC en Euros / France Métropolitaine

REMPLACER UNE TÉLÉCOMMANDE À FRÉQUENCE PAR UN MONTAGE ARDUINO : ÉMULATION ET ÉMISSION

Denis Bodor

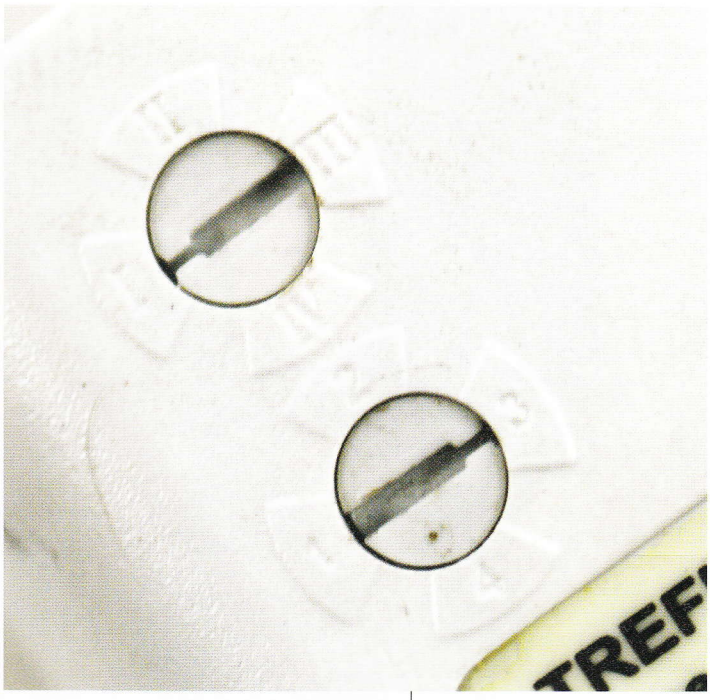
L'article précédent nous a permis de découvrir l'utilisation d'un récepteur RTL-SDR et de l'outil rtl_433 afin de capter, décoder et analyser le signal d'une télécommande générique 433 Mhz. Il est temps à présent de réutiliser les informations collectées pour émuler cette même télécommande avec notre module émetteur et une carte Arduino.



Le module récupéré depuis les « restes » de télécommande est alors dessoudé puis équipé de broches pour en faciliter l'utilisation avec une carte Arduino. La sérigraphie à la fois sur le circuit et le module est la suivante :

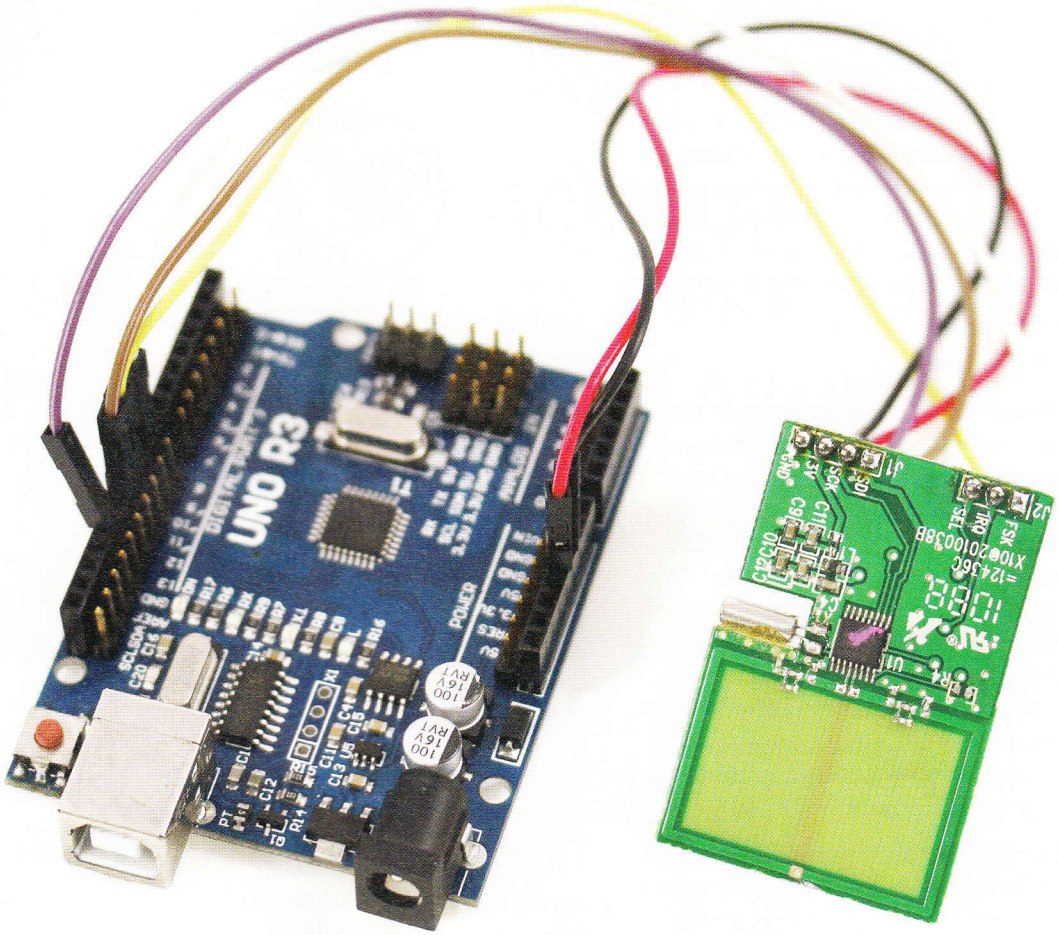
- GND : la masse ;
- BATT/3V : la tension d'alimentation. Entre 2,2V et 5,4V d'après la documentation de Silicon Labs. Le marquage « 3V » correspond en réalité à la broche VDD du Si4021 et fait référence à la source d'alimentation de la télécommande originale, 2 piles boutons CR2032 en parallèle (attention : il convient de suivre le tracé des pistes pour s'assurer que la broche du module est effectivement **directement** reliée à celle du circuit intégré et que l'alimentation n'est donc pas prise en charge par un régulateur quelconque) ;
- SCK : le signal d'horloge pour la communication avec le module. Le Si4021 est un composant utilisant une interface SPI ;
- SDI : la ligne pour la transmission de données. C'est MOSI (*Master Out - Slave In*) dans la nomenclature SPI : du maître vers l'esclave ;
- NSEL/SEL : correspond au signal /CS permettant d'adresser le module SPI lorsque cette broche est à la masse ;
- NIRQ/IRQ : permet d'obtenir des informations depuis le module comme le contenu du registre d'état (non utilisé ici). Cette broche est un mélange entre une ligne d'interruption et un équivalent à MISO (*Master In - Slave Out*) ;
- FSK : la broche permettant de fournir les données à transmettre par la voie des ondes.

Cette dernière broche est particulière. Le Si4021 peut fonctionner de deux façons dans notre type d'utilisation : soit avec une modulation FSK et dans ce cas les données sont encodées après avoir été fournies sur cette broche, soit avec une modulation ASK/OOK et la mise à l'état haut de cette broche provoque l'émission sur la fréquence configurée. Nous avons donc une interface SPI pour la configuration du module et la broche FSK pour le contrôle.



Les prises télécommandées testées ne possèdent pas une configuration fixe, mais disposent de deux sélecteurs, un pour le « canal » (en haut) et un autre pour le numéro de la prise elle-même (en bas), correspondant au numéro d'une paire de boutons sur la télécommande. Il est donc possible, par exemple, de configurer plusieurs prises sur la même paire de boutons.

Il est important ici de relever un point très important lorsqu'il s'agit de récupérer et réutiliser des éléments électroniques : dans cette situation, vous utilisez un circuit destiné à une utilisation spécifique sur laquelle vous n'avez pas toutes les informations. Dans le cas présent, la broche FSK n'était tout simplement pas connectée au reste du circuit et, après une observation méticuleuse, un pont était fait sur le module lui-même entre cette broche et la masse, à l'aide d'une résistance 0 Ohm. Toute tentative pour mettre FSK à +5V provoquait donc un court-circuit et il a fallu dessouder la



Relier le module émetteur à une carte Arduino ne présente pas de difficulté particulière, il ne s'agit que d'une liaison SPI tout à fait classique.

fameuse résistance pour utiliser cette broche. Moralité : observez attentivement ce que vous récupérez et utilisez votre multimètre pour détecter ce genre de problème **AVANT** de tenter une mise en œuvre. En d'autres termes, étudiez et analysez au mieux le circuit avant de faire n'importe quel branchement.

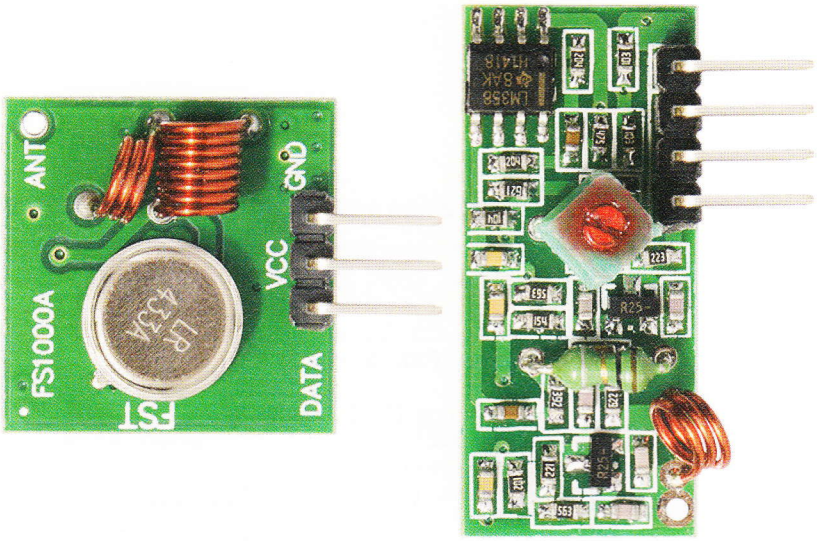
La connexion avec une carte Arduino est relativement simple :

- SCK sur la broche 13,
- SDI sur la 11 (MISO),
- NSEL sur la 10,
- FSK sur la 8,
- GND sur l'une des masses disponibles,
- BATT sur +5V.

Le reste se passe dans la partie logicielle. Le Si4021 dispose d'une documentation (*datasheet*) relativement facile à trouver sur le Web, d'une trentaine de pages (en anglais, bien sûr), contenant tout le nécessaire pour utiliser le composant.

Dans le cas de l'utilisation d'un module 433,92 Mhz économique acheté en ligne, il n'y a pas de connexion SPI ni de configuration. Ces modules émetteurs sont relativement simplistes et disposent de trois broches : GND pour la masse, VCC pour l'alimentation +5V et DATA (tantôt libellé « ATAD ») pour provoquer

l'émission du signal. Cette broche contrôle donc tout simplement la modulation ASK/OOK. Il suffit de piloter DATA avec une sortie quelconque de la carte Arduino pour obtenir un fonctionnement identique sans aucune configuration préalable. Vous êtes cependant, bien entendu, « coincé » sur une fréquence précise, impossible à ajuster, sauf à modifier physiquement le module.



L'utilisation d'un module intégrant un circuit intégré comme le Si4021 n'est pas une absolue nécessité et notre configuration est totalement équivalente à la mise en œuvre de modules plus simples et surtout plus économiques (à l'achat) comme ceux-ci, vendus en paire émetteur (en haut) et récepteur (en bas).

1. CONFIGURATION DU MODULE SI4021

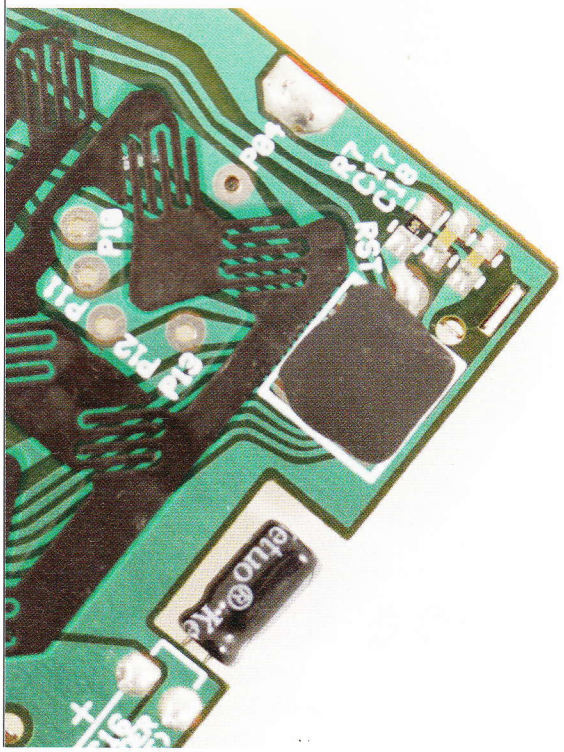
Le circuit intégré Si4021 permet bien plus que ce que nous cherchons à faire ici. Nous devons néanmoins le configurer pour notre usage spécifique, car son état après mise sous tension n'est pas celui attendu. Cette configuration se fait en envoyant des données (commandes + arguments) via la liaison SPI sous la forme de valeurs sur 8 ou 16 bits (un ou deux octets). Je ne rentrerai pas ici dans le détail de toutes les valeurs possibles des bits de ces octets, car cela reviendrait à paraphraser la documentation de Silicon Labs.

La configuration du Si4021 aura lieu dans la fonction `setup()` de notre croquis. Nous devons, dans l'ordre :

- sélectionner la bande de fréquences qui nous intéresse entre 315, 433, 868 et 915 ;
- régler la fréquence précise de l'émission sur 433,92 Mhz ;
- configurer l'utilisation de la modulation OOK et l'amplification souhaitée ;
- et finalement activer chaque élément interne du Si4021 nécessaire à l'émission.

Au terme de cette configuration, le fait de mettre FSK à +5V provoquera l'émission et l'interrompra si cette broche est à la masse. Nous pourrons donc moduler le signal à notre convenance en utilisant cette unique broche. Notez qu'en ne configurant pas la modulation ASK/OOK, les signaux arrivant sur la broche FSK sont utilisés par défaut pour la modulation par déplacement de fréquence (FSK).

Sur le circuit de base de la mystérieuse télécommande de récupération se trouve ce qui semble être un microcontrôleur sous une goutte de résine. C'est ce composant qui était chargé de piloter le module d'émission et de prendre en charge la lecture des boutons. Il n'y a ici, malheureusement rien à récupérer pour une utilisation ultérieure.



La première chose à faire consiste à configurer la connexion SPI :

```
#include <SPI.h>

#define CS 10
#define FSK 8

void setup() {
  // configuration ports en sortie
  pinMode(CS, OUTPUT);
  pinMode(FSK, OUTPUT);

  // CS à l'état haut
  // le module n'est PAS "sélectionné" et
  // ne traite donc PAS les éventuelles données SPI
  digitalWrite(CS, HIGH);

  // Configuration SPI
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV16);
  SPI.setDataMode(SPI_MODE0);
  SPI.setBitOrder(MSBFIRST);
```

À présent, nous pouvons dialoguer avec le Si4021 et nous commençons par choisir la bande de fréquence de 433 Mhz, ainsi que la configuration de l'oscillateur :

```
  // Configuration :
  // bande de fréquence sur 433 Mhz
  // sortie horloge à 10 MHz
  digitalWrite(CS, LOW);
  SPI.transfer(B10001111);
  SPI.transfer(B00001000);
  digitalWrite(CS, HIGH);
```

Il faut ensuite régler la fréquence utilisée. Sur les 16 bits envoyés, nous avons la commande **1010** suivie de 12 bits spécifiant une valeur entre 96 et 3903, utilisée dans une formule de calcul détaillée dans la documentation : **10 Mhz * C1 * (C2 + F/4000)**. **F** est notre valeur sur 12 bits. **C1** et **C2** dépendent de la bande utilisée. Pour la bande 433 Mhz, les valeurs sont respectivement 1 et 43. Notre calcul est donc : **10*1*(43+1568/4000) = 433.92 Mhz** et la valeur 1568 est utilisée pour les fameux 12 bits (4 dans le premier octet et 8 dans le second) :

```
  // configuration de la fréquence sur 433.92 MHz
  digitalWrite(CS, LOW);
  // 011000100000 = 1568
  // 10*1*(43+1568/4000) = 433.92 Mhz
  SPI.transfer(B10100110);
  SPI.transfer(B00100000);
  digitalWrite(CS, HIGH);
```

Nous pouvons alors passer à la configuration du type de modulation et au réglage de l'amplification. La commande est **1011** et les arguments sont 1 bit pour choisir OOK et 3 bits pour l'amplification/atténuation entre 0 et -21 dB :

```
    // Configuration de l'émission
    digitalWrite(CS, LOW);
    // 1011 + ook p2 p1 p0
    SPI.transfer(B10111000); // OOK + power
    digitalWrite(CS, HIGH);
```

La configuration étant maintenant complète, il ne nous reste plus qu'à activer les différents éléments du circuit intégré (oscillateur, synthétiseur, amplificateur) :

```
    // Configuration de l'alimentation
    digitalWrite(CS, LOW);
    SPI.transfer(B11000000);
    // a1 a0 ex es ea eb et dc
    // ex = 1 = crystal oscillator enabled
    // es = 1 = synthesizer enabled
    // ea = 1 = power amplifier enabled
    // dc = 0 = enable clock output buffer
    SPI.transfer(B00111000);
    digitalWrite(CS, HIGH);
}
```

À ce stade, nous pouvons très simplement tester le fonctionnement en ajoutant une fonction `loop()` vide et en nous amusant à mettre la broche FSK à +5V manuellement. En réglant notre récepteur RTL-SDR, avec GQRX par exemple, sur 434 Mhz, nous devons voir une émission sur la fréquence configurée.

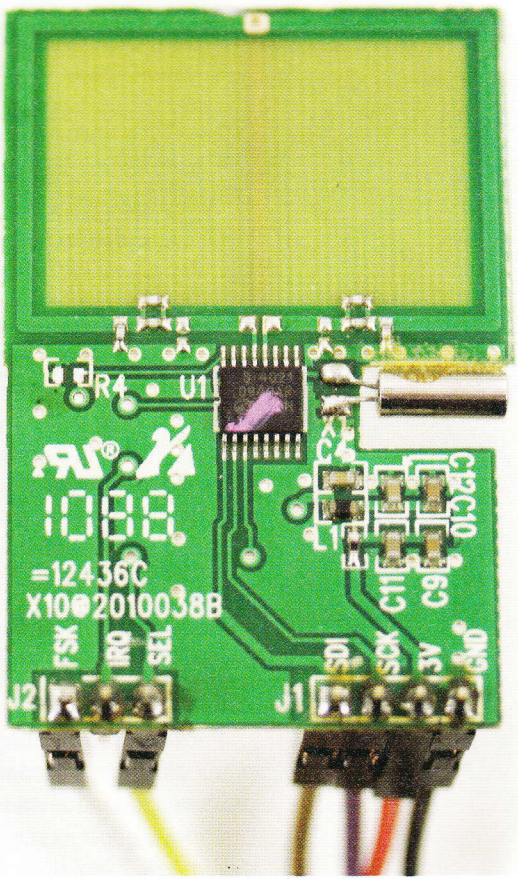
2. ÉMISSION DES MESSAGES ET TEST

Nous disposons maintenant d'un émetteur contrôlable avec une simple sortie de la carte Arduino, mais encore faut-il pouvoir moduler le signal de façon à ce qu'il puisse être interprété par le récepteur. En d'autres termes, nous devons trouver un moyen de convertir les suites de bits collectées avec *rtl_433* en une série d'impulsions envoyées sur la broche FSK et de ce fait, émises par la voie des ondes grâce au module.

Heureusement pour nous, ce travail a déjà été fait par Suat Özgür (alias sui77) sous la forme d'une bibliothèque Arduino : *rc-switch*. Son code peut être récupéré sur **https://github.com/sui77/rc-switch** et intégré comme n'importe quelle bibliothèque dans votre environnement Arduino (il faut le faire manuellement, *rc-switch* n'étant pas disponible via le gestionnaire de bibliothèques).

Une fois la bibliothèque installée, on commence par l'intégrer au croquis et par créer un objet correspondant :

Le module d'émission est un ensemble relativement simple : on y trouve le Si4021 de Silicon Labs, un oscillateur à quartz de 10 Mhz, quelques condensateurs et résistances. La partie supérieure du circuit forme une antenne-cadre, une solution peu coûteuse et relativement courante pour ce genre d'applications.



```
#include <RCSwitch.h>

RCSwitch monSwitch = RCSwitch();
```

`monSwitch` est maintenant une instance de `RCSwitch` nous permettant de configurer et moduler nos informations/messages. Plusieurs méthodes permettent de configurer la façon de moduler le signal :

```
  // Option : durée de l'impulsion
  // mySwitch.setPulseLength(320);
  // Option : protocole, 1 fonctionne la plupart du temps
  monSwitch.setProtocol(1);
  // Option : nombre de répétitions du message
  monSwitch.setRepeatTransmit(10);

  monSwitch.enableTransmit(FSK);
```

Les valeurs par défaut ont parfaitement fonctionné dans mon cas, mais selon la prise à commander vous devrez peut-être essayer d'autres combinaisons sur la base des informations retournées par *rtl_433*.

Il ne nous reste plus qu'à faire le test, dans la fonction `loop()` en envoyant alternativement un message d'allumage puis un autre d'extinction à différents intervalles de temps :

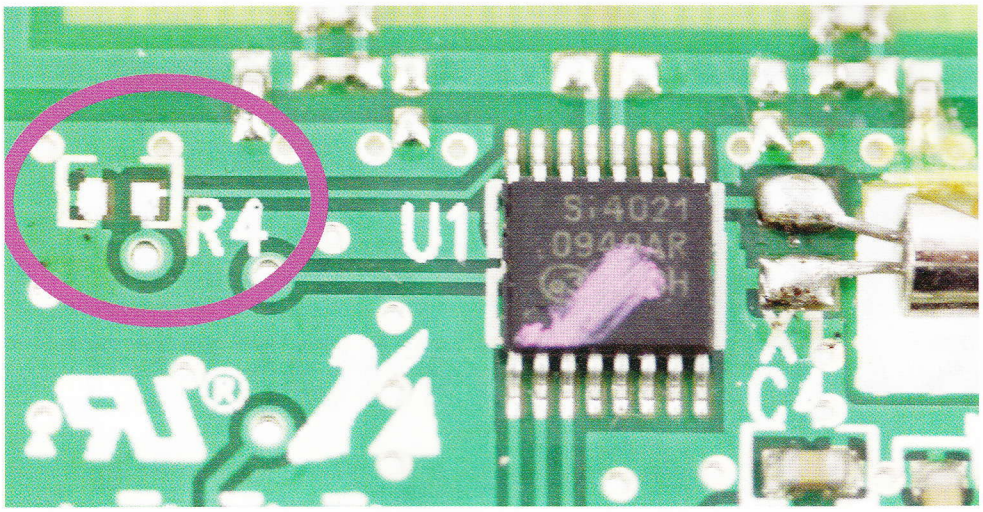
```
void loop() {
  //  canal 1 bouton 3 ON
  monSwitch.send("000101010101000101010101");
  delay(1000);
  //  canal 1 bouton 3 OFF
  monSwitch.send("000101010101000101010100");
  delay(2000);
}
```

À ce stade, la prise concernée (bon canal, bon bouton) doit réagir en conséquence et démontrer la viabilité du montage. En cas de problèmes, utilisez à nouveau *rtl_433* en mode analyse afin de comparer l'émission de la vraie télécommande avec celle de votre montage/croquis. Aidez-vous éventuellement de GQRX ou SDR# pour valider la fréquence utilisée et la puissance du signal.

3. PERSPECTIVES ET UTILISATION

Nous pouvons maintenant facilement développer notre croquis ou intégrer ce code de test dans n'importe quel projet et ainsi piloter notre installation électrique de façon automatique pour l'usage qui nous plaira.

Parmi les idées qui viennent naturellement à l'esprit, nous avons en premier lieu le fait de réunir plusieurs télécommandes en une seule et donc de prendre en charge des messages provenant de modèles et de marques différentes de télécommande. En effet, les produits que l'on trouve généralement ne permettent souvent de piloter que quatre prises. Pour en contrôler davantage, il faut donc deux ensembles différents et donc deux télécommandes réglées sur deux



Voici le cœur du module récupéré, le Si4021 de Silicon Labs. Notez l'emplacement noté R4 sur la gauche qui initialement était celui d'une résistance de 0 ohm (un simple pont) connectant la broche FSK à la masse et nous empêchant ainsi d'utiliser cette ligne pour contrôler l'émission du signal. Il a donc fallu dessouder ce composant pour utiliser le module.

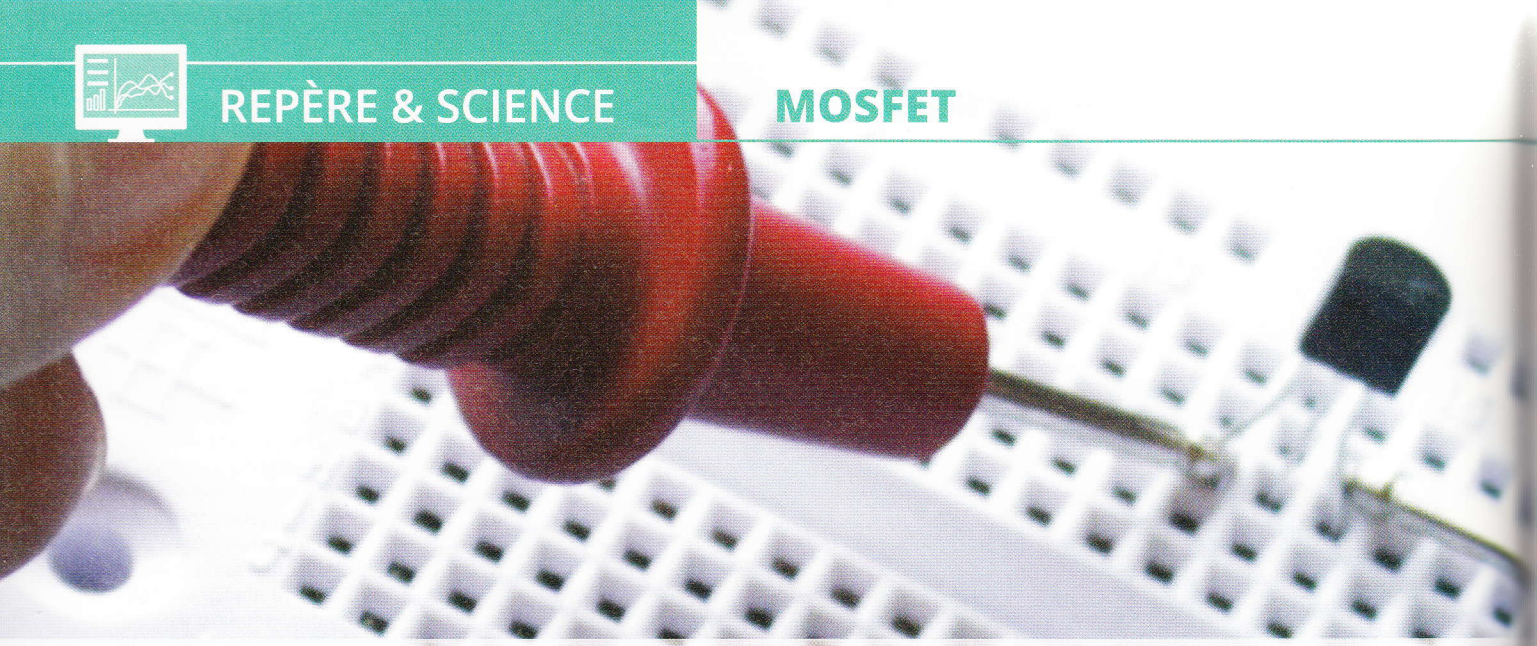
canaux différents. Ce problème ne se pose pas avec un émetteur et une carte Arduino, nous pouvons très simplement émettre les messages de 32 boutons (4 canaux, 4 prises, deux boutons par prise) à partir d'un seul croquis.

Sur cette base, nous pouvons également envisager une automatisation plus poussée en ajoutant une horloge temps réel (RTC) comme un module DS1307. Ceci nous permettra de piloter les prises et donc différents appareils et lampes en fonction de plages horaires pour simuler une présence. Idéal pour les vacances et ainsi éviter les visites d'éventuels cambrioleurs.

Enfin, dans le cas précis de la mise en œuvre du circuit intégré Si4021, on peut également imaginer le développement d'une bibliothèque complète pour faciliter la prise en charge du composant et, pourquoi pas, imaginer une interface permettant de configurer l'ensemble de façon générique avec un jeu de commandes via le moniteur série. Il en résulterait la création d'un émetteur polyvalent configurable, sorte de couteau suisse de l'émission de signaux ASK/OOK ou FSK pilotable avec un terminal série, Processing ou encore un code Python fait maison.

Dans le même ordre d'idée de développement générique, on peut également imaginer se pencher vers d'autres produits que de simples télécommandes ou carillons. Sur la base du code de *rtl_433,* il est parfaitement possible de créer, par exemple, de toutes pièces un capteur de température qui sera compatible avec une station météo du commerce. On peut alors envisager, pourquoi pas, de tenter de mettre en place un projet complet visant à fournir l'équivalent de *rtl_433* en « écriture » plutôt qu'en « lecture » des ondes.

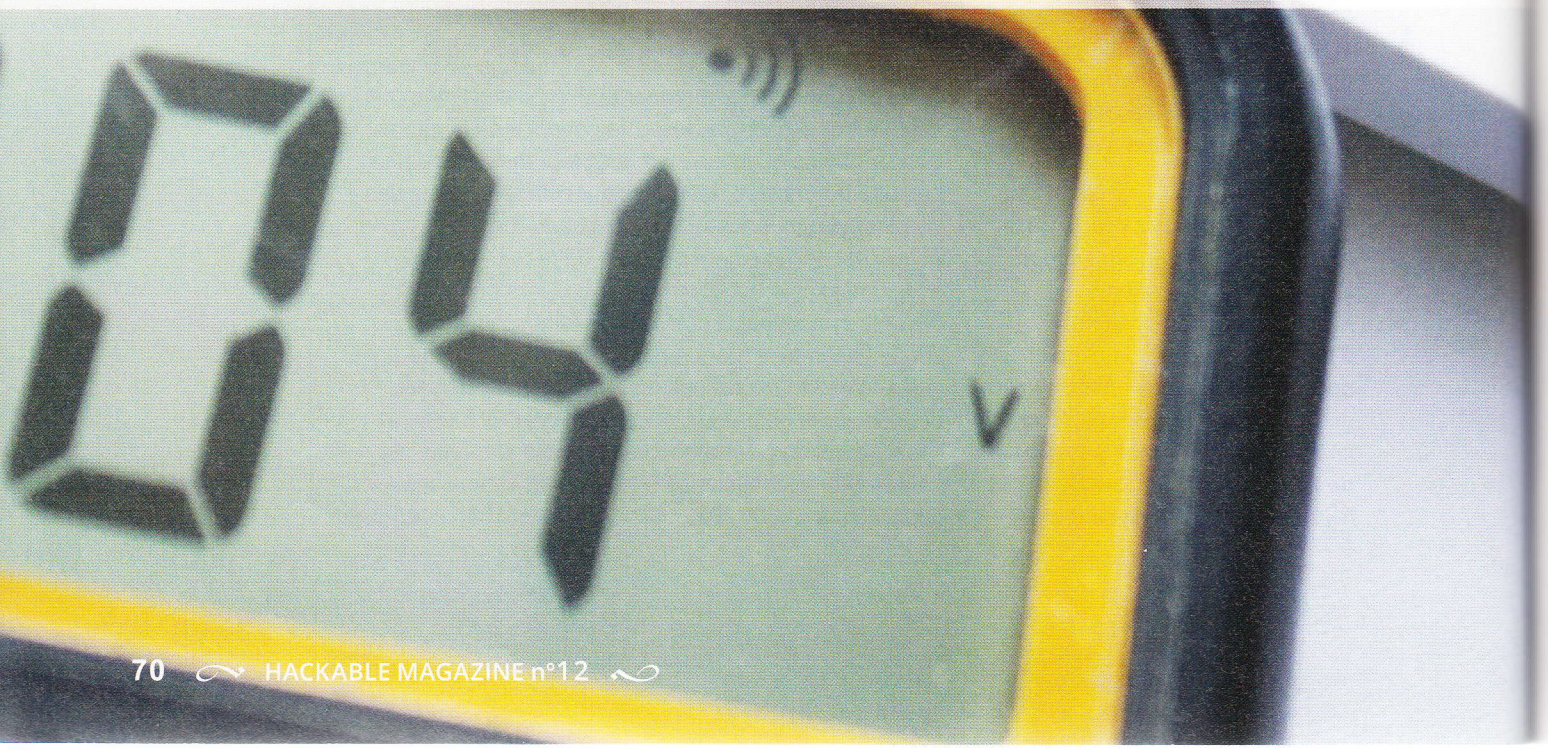
Je préciserai pour terminer que, même si l'idée semble très amusante, tester la patience ou les nerfs de ses voisins en interférant avec leur installation n'est pas une bonne idée. Même si tout cela reste dans des bandes de fréquences utilisables sans licence (ISM) et qu'il est parfaitement possible de faire de même avec une simple télécommande, dès lors qu'il s'agit d'émettre un signal, il convient de faire preuve de bon sens et de la plus grande prudence. L'accès à une telle technologie n'est pas un motif pour se montrer déraisonnable, bien au contraire. Procédez à vos essais dans un environnement contrôlé et tâchez de ne pas interférer avec des équipements qui ne vous appartiennent pas. **DB**



NOS AMIS LES MOSFET

Yann Guidon
[Amateur de méta-commentaires]

Ne vous contentez plus des transistors bipolaires ! Les MOSFET sont une de ces merveilles modernes dont vous auriez tort de vous passer. Nous allons aborder les côtés pratiques de leur utilisation au quotidien car, si vous ne l'avez pas déjà fait, vous les adopterez après avoir lu cet article. Ils résoudront certains de vos problèmes et rendront vos montages plus efficaces, alors examinons leurs caractéristiques.



Aujourd'hui, les MOSFET sont partout. C'est un composant fabriqué en quantités industrielles comme les autres donc il est devenu abordable, comme son précurseur bipolaire. On s'en procure facilement, une pièce coûte environ 10 centimes et parfois à peine un centime en achetant des lots en ligne : 10$ pour 1000 pièces, port compris, c'est proche de ce qu'on appelle de la « poussière électronique », car si vous en laissez tomber un, ça ne vaut presque pas la peine de se baisser pour le ramasser.

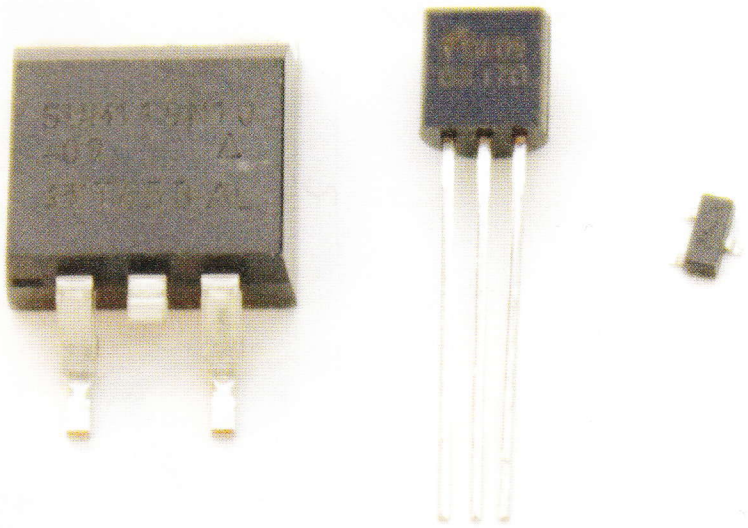


Fig. 1 : Trois tailles et puissances de MOSFET à canal N. À gauche : SUM110N-09 en boîtier TO263 (100V, 110A, 10mOhm à 10Vgs). Au milieu : BS170 en boîtier TO92 (60V, 0,5A, 1 Ohm). À droite : 2N7002 en boîtier SOT23 (60V, 0,1A à 0,3A).

1. DESSINE-MOI UN MOSFET...

Un MOSFET est un composant qui permet de commuter du courant, c'est-à-dire laisser passer plus ou moins d'électrons entre deux électrodes en fonction d'une troisième. Il peut être vu comme un interrupteur (tel un relais) ou une résistance variable.

Le courant commuté maximal dépend de la taille du boîtier et d'autres caractéristiques, fournies par le constructeur. La référence du MOSFET est choisie en fonction du type d'application, et il existe une très grande variété de modèles. Nous allons nous concentrer sur les applications de faible puissance, avec les boîtiers TO92 ou SOT23 qui sont les moins chers.

- Les références les plus courantes en boîtier TO92 (avec des pattes, qu'on peut planter dans une platine sans soudure) sont BS170 pour la version canal N (équivalent du NPN), et BS250 pour le canal P (le complémentaire du canal N). Le 2N7000 est une autre référence courante, similaire au BS170, mais avec un brochage différent.
- Les équivalents en boîtiers SOT23 (boîtier de 3mm à monter en surface) sont BSS138 (N) et BSS84 (P). À notre niveau, le 2N7002 est plus ou moins équivalent au BSS138.

On trouve des MOSFET plus gros et plus puissants, capables de supporter des centaines de volts et/ou des centaines d'ampères. À l'extrême opposé, les transistors de la famille CMOS (version miniaturisée des MOSFET) constituent aujourd'hui l'essentiel des circuits des ordinateurs actuels et une puce peut en intégrer des milliards. En raison de leur infime dimension, ils ne peuvent commuter qu'un courant ridiculement faible et ne peuvent pas supporter plus de 2V, mais en échange, ils sont hyper rapides.

2. LA BÊTE À TROIS PATTES

Un transistor MOSFET discret, tel qu'on l'achète à l'unité, présente trois broches, comme son cousin bipolaire :

- La source (équivalent de l'émetteur d'un transistor bipolaire) est habituellement connectée directement à la source d'alimentation.
- Le drain (équivalent du collecteur) est habituellement connecté au circuit à commuter.
- La grille, qui correspond à la base, contrôle la résistance entre la source et le drain.

On pourrait donc presque remplacer un bipolaire par un MOSFET. Cependant, contrairement au bipolaire, aucun courant ne passe au travers de la grille d'un MOSFET*, car c'est un type de transistor à grille isolée.

* contrairement à ce que suggère la page 90 de *Hackable n°7*

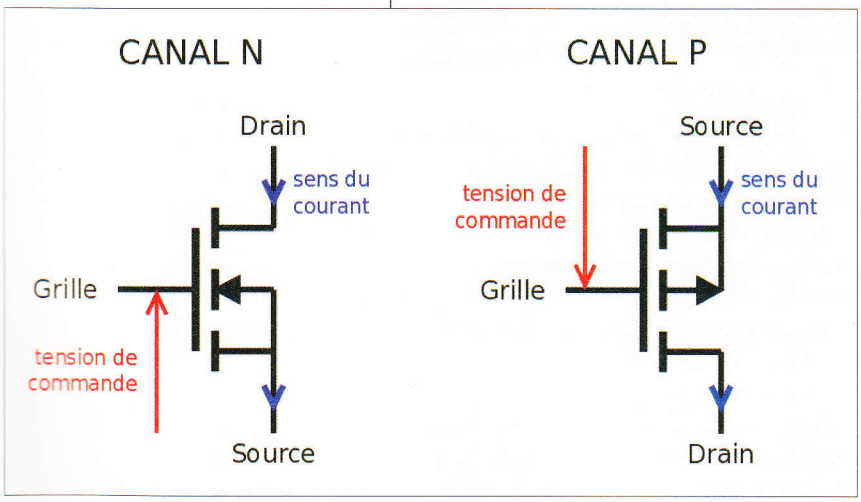


Fig. 2 : À gauche, le symbole d'un MOSFET canal N, à droite le symbole du canal P (une version miroir ou complémentaire). Leur fonctionnement est inversé, mais on les représente normalement avec le sens du courant qui va vers le bas (de la tension la plus élevée à la plus faible). La flèche de tension du canal P est orientée vers le bas, car il travaille avec une tension négative.

3. ANATOMIE D'UN MOSFET

Le terme MOSFET signifie en anglais *Metal Oxide Semiconductor Field Effect Transistor*. Commençons par l'examiner à partir de la fin.

- Déjà, cela indique que c'est un transistor et c'est réellement une résistance variable. Le mot « transistor » est la contraction de « *Transfer Resistor* » (résistance de transfert) et un transistor bipolaire se comporte plus comme un amplificateur de courant.
- *Field Effect* signifie *effet de champ*. C'est un champ électrique qui contrôle la résistance du composant. Un peu comme les antiques tubes à vide, les MOSFET sont donc contrôlés avec une tension, par opposition aux transistors bipolaires qui sont contrôlés par un courant traversant leur base. Ils ont donc l'avantage de consommer moins d'énergie et ils peuvent amplifier avec un gain très supérieur. Ils ne dissipent presque pas d'énergie lorsqu'ils sont saturés, même s'il faut *une certaine énergie* pour y arriver.
- *Metal Oxide Semiconductor* représente la succession des couches qui composent le transistor. Une couche de métal (conducteur) est déposée sur une très fine couche d'oxyde (isolant) reposant sur un semi-conducteur (habituellement à base de silice dopé d'impuretés). Dans les circuits

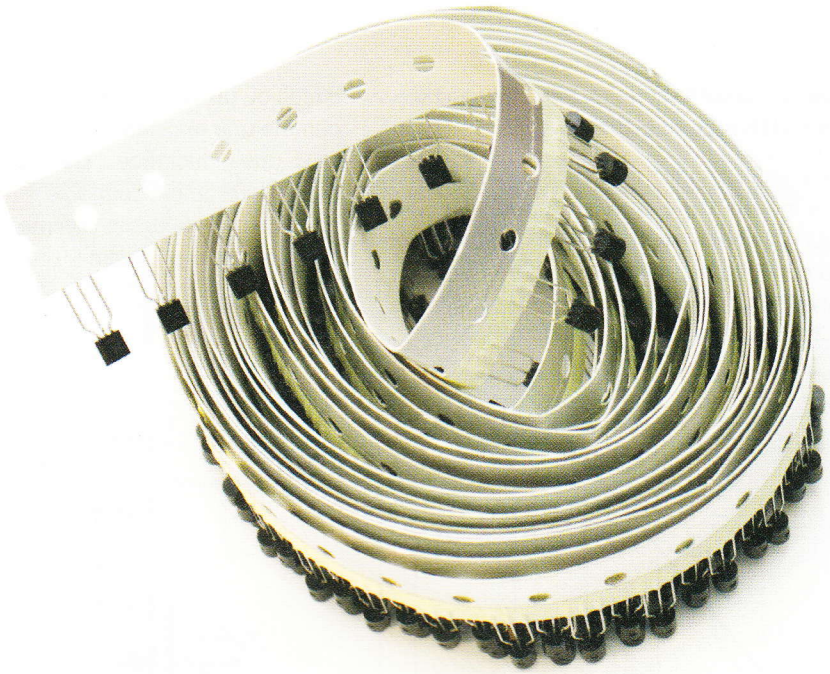


Fig. 3 : Ces BS170 sont conditionnés sur une surface conductrice pour éviter l'accumulation de charges électrostatiques qui endommageraient le composant.

intégrés, la couche métallique est aujourd'hui remplacée par du silicium polycristallin, mais le terme *Metal* est resté.

Plus la couche isolante est fine, plus le champ électrique est intense au niveau du semi-conducteur, et donc plus le transistor sera sensible et il commutera à une plus faible tension. En contrepartie, cela le rend très sensible aux surtensions, en particulier aux décharges d'électricité statique. L'isolant est fragile et il arrive souvent qu'un MOSFET claque pour un oui ou pour un non !

Cela peut se manifester par un court-circuit interne, par exemple, ou une résistance anormale. Pour réduire les chances de destruction du circuit, assurez-vous d'avoir observé toutes les précautions anti-ESD (*ElectroStatic Discharge*) possibles. Pour commencer, restez connecté à la terre et évitez les vêtements et les chaussures synthétiques. Leurs frottements peuvent générer des tensions hallucinantes, surtout par temps sec !

4. UTILISER UN MOSFET COMME UNE RÉSISTANCE VARIABLE

Prenons un 2N7000 (Canal N) et branchons-le tel que le montre le dessin suivant.

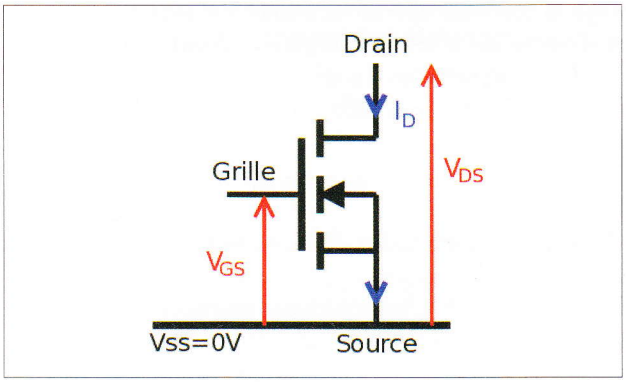


Fig. 4 : Circuit de test des caractéristiques d'un MOSFET N.

La tension entre la source et le drain (notée V_{DS}) est fixée par une alimentation de laboratoire à 10V.

La tension de grille (entre la grille et le drain, notée V_{GS}) est ensuite variée (au moyen d'une autre alimentation) et on mesure le courant qui est débité par l'alimentation 10V. Selon la documentation du constructeur, on devrait obtenir une courbe comme celle-ci :

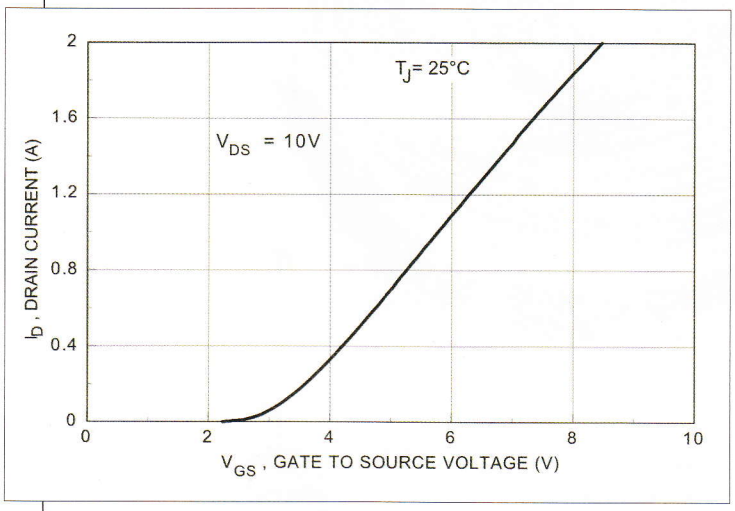


Fig. 5 : Courbe de transfert d'un MOSFET-N pour Vds constant, le courant est une fonction de Vgs.

Si la tension de grille augmente trop, le courant deviendra trop important, faisant chauffer le composant. Contrairement à un transistor bipolaire, la résistance d'un MOSFET augmente aussi lorsque le composant est chaud, ce qui augmente encore plus la dissipation et le processus risque de s'emballer, provocant la destruction du composant.

En fait, c'est normal que le composant souffre, car ce montage le connecte directement entre les deux bornes d'une alimentation et toute l'énergie est absorbée par le composant. Cela s'appelle, en d'autres termes, un court-circuit. Comme quoi il ne faut pas recopier bêtement les circuits des datasheets.

5. UTILISER UN MOSFET COMME INTERRUPTEUR

Bien qu'un MOSFET soit capable d'absorber plus ou moins de puissance, il est préférable que cette énergie soit transmise à une charge qui la consomme : un autre circuit ou un autre composant comme une résistance, une LED, une inductance... Dans ce cas, la charge est connectée en série avec le drain, comme dans le schéma suivant :

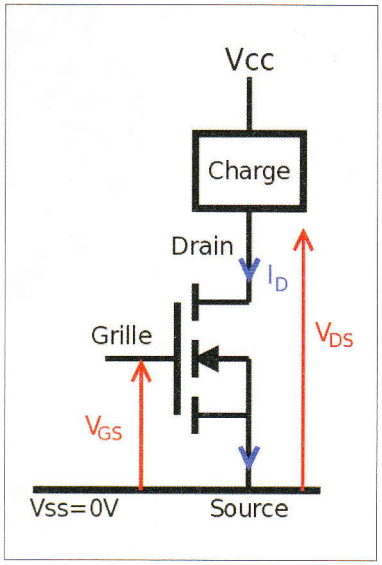


Fig. 6 : En réduisant considérablement sa résistance, le MOSFET permet d'alimenter efficacement une charge résistive.

La commutation avec un canal N vers le 0V est appelée « Low Side ». Les MOSFET N sont plus efficaces et moins chers que les MOSFET P, mais il n'est pas toujours possible ou raisonnable de couper le 0V, cela peut créer de nombreux soucis pour le reste du circuit.

Pour commuter l'alimentation positive et conserver la masse commune, donc en montage « High Side », il faut utiliser un MOSFET canal P, dont la source se retrouvera connectée à V_{CC}.

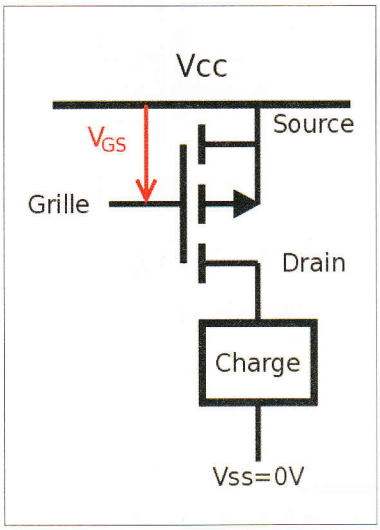


Fig. 7 : Un MOSFET P permet de conserver une masse commune pour tout le montage.

6. LES TROIS RÉGIMES DU MOSFET

Ce qui nous intéresse alors est la tension du drain (V_{DS}), car le courant n'est plus critique, maintenant que la charge encaisse la différence de tension (donc consomme l'énergie). Tant que le MOSFET est beaucoup plus conducteur que la charge, la tension à ses bornes reste faible, et comme la puissance dissipée est le produit de la tension et du courant, alors il ne court pas de risque.

Selon la tension entre la grille et la source, le MOSFET sera alors considéré comme étant dans un état bloqué, ohmique ou saturé. Les tensions varient d'un modèle à l'autre, même d'un composant à l'autre en raison des tolérances de fabrication, et bien sûr des caractéristiques de la charge du drain. Les transitions d'un état à l'autre ne sont pas non plus très précises, mais on parle de ces trois régimes de fonctionnement pour simplifier les calculs.

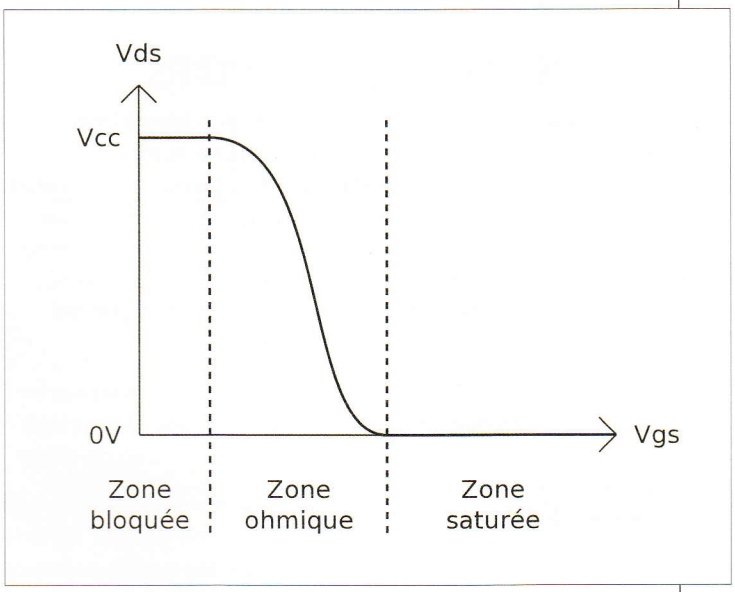


Fig. 8 : Les trois régimes de fonctionnement dépendent de la tension de grille et de la résistance en série sur le drain.

- Régime bloqué : à moins de 2V, le MOSFET ne laisse quasiment pas d'électron passer. La résistance entre la source et le drain est presque infinie. Cela correspond à un interrupteur ouvert et la charge n'est plus alimentée.
- Régime ohmique ou résistif : autour de quelques volts, la résistance est inversement proportionnelle à la tension de grille. Le MOSFET se comporte alors comme une résistance, ajustable grâce à cette tension. Le transistor subit à la fois une tension et un courant et peut donc chauffer.
- Régime saturé : vers environ 10V, quand l'augmentation de la tension de grille ne se traduit plus par une réduction de V_{DS}, le transistor est saturé. La résistance équivalente peut être une dizaine ou centaine de milliohms selon les modèles, certains peuvent même réduire leur résistance au point d'être considérés comme un conducteur normal. Cela correspond à un interrupteur fermé et la charge est complètement alimentée.

Évidemment, les valeurs indicatives données ci-dessus doivent être vérifiées pour chaque type de MOSFET que vous utiliserez, certains sont optimisés pour de très basses tensions de fonctionnement, d'autres pour de très forts courants...

7. LES EFFETS PARASITES

Nous avons vu que le MOSFET est une bête à trois pattes. Sous le capot, par contre, il y a une autre zone conductrice dans le sandwich semi-conducteur, qui poserait des soucis si on la laissait flotter. Dans la plupart des cas, cette zone est raccordée à une des électrodes, ce qui rend le composant asymétrique. Sans cette connexion interne, le drain et la source seraient tout à fait interchangeables (comme sur un FET unijonction tel le BF245).

Cependant, la connexion a un autre effet encore plus important. Elle met en contact une zone dopée N avec une zone dopée P, ce qui crée... une diode. La tension de chute dans le sens passant est similaire à une diode classique comme la 1N4148 (entre 0,6V et 0,7V).

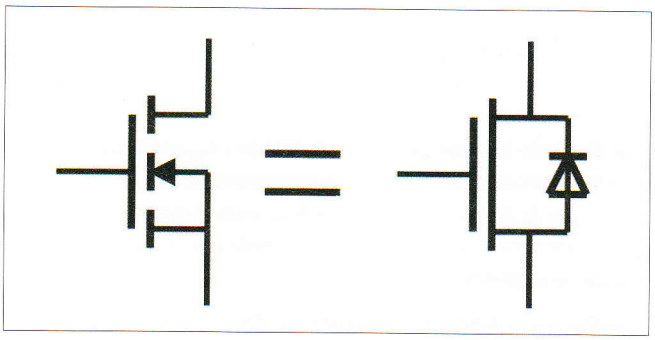


Fig. 9 : Un transistor MOSFET à trois broches se comporte comme un transistor FET à grille isolée, en parallèle avec une diode dite intrinsèque, entre la source et le drain.

Cette diode intrinsèque, ou parasite, a des avantages et des inconvénients à garder en tête. Il faut se rappeler que le courant peut passer dans l'autre sens et perturber un montage. D'un autre côté, elle peut rendre service comme on le verra plus loin.

Pour les rares cas où le courant doit être bloqué dans les deux sens, je n'ai jamais trouvé de MOSFET à 4 broches (sauf pour des applications trop spécifiques dans les hautes fréquences), mais il existe une astuce consistant à mettre deux MOSFET tête-bêche, afin que leurs diodes s'annulent.

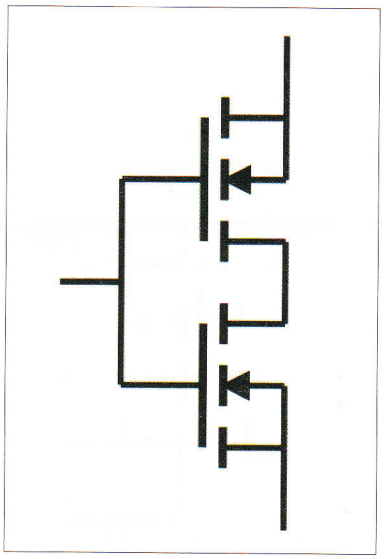


Fig. 10 : Un commutateur idéal est constitué de deux MOSFET identiques, connectés en miroir.

7.1 Capacité de la grille

En électronique, deux électrodes parallèles séparées par un isolant, cela s'appelle un condensateur. De par sa structure, un MOSFET contient un tel dispositif, entre la grille et la zone semi-conductrice.

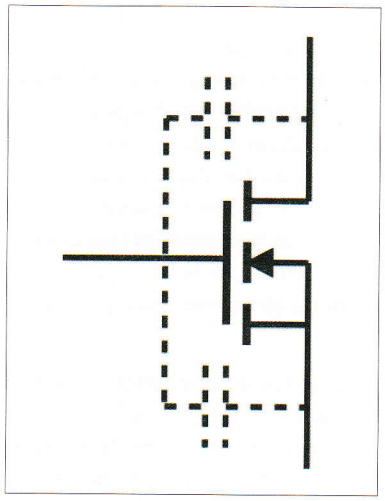


Fig. 11 : La structure des MOSFET crée aussi des capacités parasites.

Cela a de nombreux effets, plus ou moins désirables, le plus important étant une sorte d'effet mémoire : le MOSFET conserve l'état passant ou bloqué si on déconnecte la grille. Le courant de fuite est habituellement négligeable et l'état peut persister *un certain temps* (pour de plus amples précisions, regardez le sketch du fût du canon de Fernand Reynaux).

C'est cet effet qui est mis à profit dans les mémoires dynamiques : la grille est chargée à un potentiel (haut ou bas) et on peut lire l'état en mesurant la résistance. Ce type de transistor est extrêmement petit et le courant de fuite non négligeable, il faut donc relire et recharger la grille plusieurs milliers de fois par seconde (c'est les fameux cycles de rafraîchissement). La plupart des ordinateurs utilisent ce type de composant (souvent assemblés en barrettes) pour constituer leur mémoire de travail.

L'effet mémoire peut créer un problème si le MOSFET est contrôlé par une sortie à 3 états, comme une broche d'entrée-sortie de microcontrôleur. Supposons un instant que vous alimentez un circuit au travers d'un MOSFET N : sa grille « flottera » (et captera tout parasite passant par là) tant que la broche de contrôle du MCU n'est pas à un état défini. Cela se produit à la mise sous tension (quelques millisecondes avant que l'état ne soit configuré par le programme) ou durant une phase de *reset*. Ne vous étonnez pas si votre montage est alimenté transitoirement, ou même s'il reste allumé !

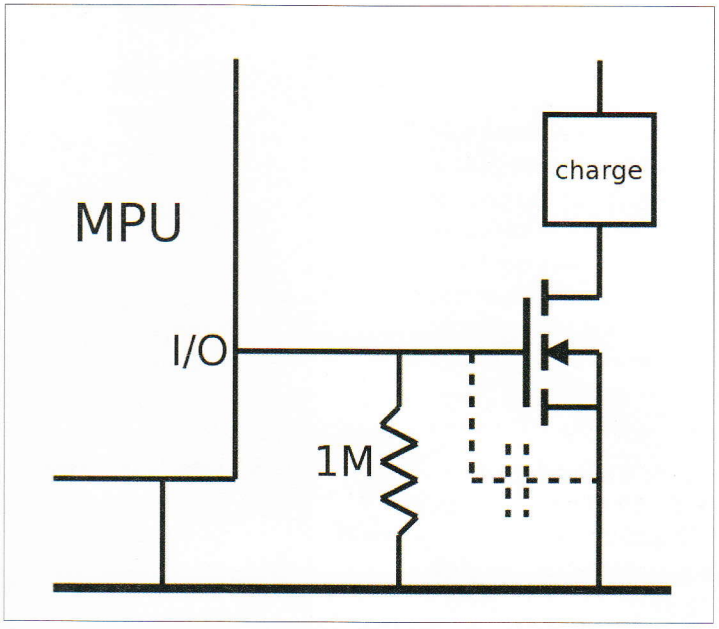


Fig. 12 : Lorsque la sortie 3 états est flottante, la résistance de tirage décharge la capacité parasite de la grille, ce qui déconnecte la charge.

La solution consiste à définir cet état par défaut en ajoutant une petite résistance de très forte valeur entre la grille et la source. Typiquement, 1MΩ suffit pour couper le MOSFET quand sa grille n'est plus à un état défini.

Avec l'habitude, cette résistance de tirage devient presque le pendant de l'indispensable résistance de grille d'un transistor bipolaire, avec l'avantage qu'il n'est pas nécessaire de la calculer, on met « environ 1MΩ » tout simplement.

7.2 Pilote de grille pour MOSFET de puissance

Plus un MOSFET doit laisser passer de courant, plus la surface de semi-conducteur et de grille est importante. Si on compte l'isolant, cela forme un condensateur dont la valeur peut atteindre plusieurs nanofarads, et encore plus si on ajoute les autres effets parasites. Par exemple, la capacitance du BS170 est environ 30pF, mais celle du SUM110 peut atteindre 7nF.

Pour les montages qui commutent à très haute fréquence, il faut en plus tenir compte de *l'effet Miller*, qui se traduit pour nous par des forts courants pour charger la grille, lors des transitions entre l'état passant et l'état isolant. Les transistors de puissance, afin d'être aussi efficaces que possible et dissiper le moins d'énergie, doivent être complètement saturés, à des tensions de 10V ou plus. Certains peuvent réduire leur résistance à quelques milliohms, mais cela

nécessite de pomper beaucoup d'électrons de ou vers la grille. Cela se transforme en un fort courant instantané, car plus la transition est lente, plus le transistor sera en régime résistif longtemps et dissipera de l'énergie. C'est très important pour les alimentations à découpage, car il faut hacher le courant à plusieurs dizaines ou centaines de kilohertz !

Des circuits intégrés, appelés « MOSFET driver » ou « power-MOSFET gate driver », ont été conçus pour piloter la grille avec des pics de courant suffisants pour compenser l'effet de Miller et améliorer le temps de commutation. Les pilotes sous forme de circuits intégrés sont souvent très efficaces et ont de nombreux avantages, qui peuvent justifier de les utiliser seuls.

Par exemple, si vous devez contrôler un circuit de puissance moyenne (moins d'un ampère), avec des commutations très rapides (ne durant pas plus que quelques dizaines de nanosecondes), mais la tension de fonctionnement est plus élevée que votre circuit de contrôle, un driver de MOSFET est tout indiqué. C'est une interface très intéressante entre une commande en basse tension et une charge en moyenne tension.

8. RALENTIR LA COMMUTATION POUR ADOUCIR LES TRANSITIONS

Nous avons déjà parlé de la nécessité de commuter un MOSFET aussi vite que possible. Cela réduit les pertes en diminuant la durée du

COMMENT TESTER UN MOSFET INCONNU ?

Supposons que vous venez de trouver un MOSFET de type totalement inconnu. Est-il de type P ou N ? Quel est son brochage ? Pour le savoir, et vérifier que le datasheet correspond bien, mettons à profit les deux effets parasites que nous venons de voir.

Puisqu'il n'y a que 3 broches, 6 combinaisons sont possibles. On élimine facilement les mauvaises avec quelques tests au moyen d'un multimètre réglé sur le calibre « diode ».

Pour trouver la source et le drain, il suffit de trouver une chute de tension d'environ 0,6V, correspondant à la diode parasite. Cela implique 3 paires de tests, un test dans chaque sens (car la diode n'apparaît que dans un sens).

Cependant, à cause de l'effet mémoire, si on charge la grille (une chance sur deux), alors le transistor devient passant : on mesure donc une résistance très faible, ou une tension presque nulle, entre la source et le drain. On ne peut pas savoir le sens (puisque c'est symétrique), car la diode parasite est court-circuitée.

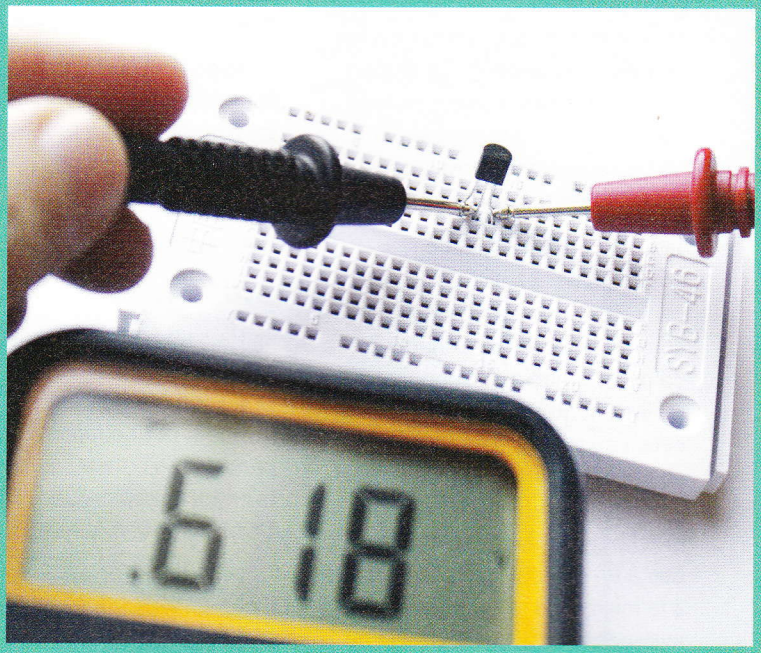


Fig. a : Deux pattes sont court-circuitées par le contact simultané d'une sonde. L'autre patte est amenée à une tension positive par l'autre sonde. On mesure 0,618V avec le calibre diode, on déduit que la sonde positive est la source.

Il est donc important, entre chaque test, de court-circuiter les 3 broches entre elles, pour décharger la grille et rendre le transistor bloqué.

Pour trouver le type, c'est simple : regardez si le transistor devient passant si vous chargez la grille positivement ou négativement (inversez le sens des pointes de mesure).

Ainsi, avec un peu d'astuce et de déduction, on peut retrouver le brochage et le type d'un MOSFET avec une demi-douzaine de mesures.

Voici le résumé de la procédure pour trouver la source et le drain :

- Étiqueter les broches inconnues avec les noms A, B et C,
- Court-circuiter A, B et C,
- Mesurer A – B,
- Mesurer B – A,
- Court-circuiter A, B et C,

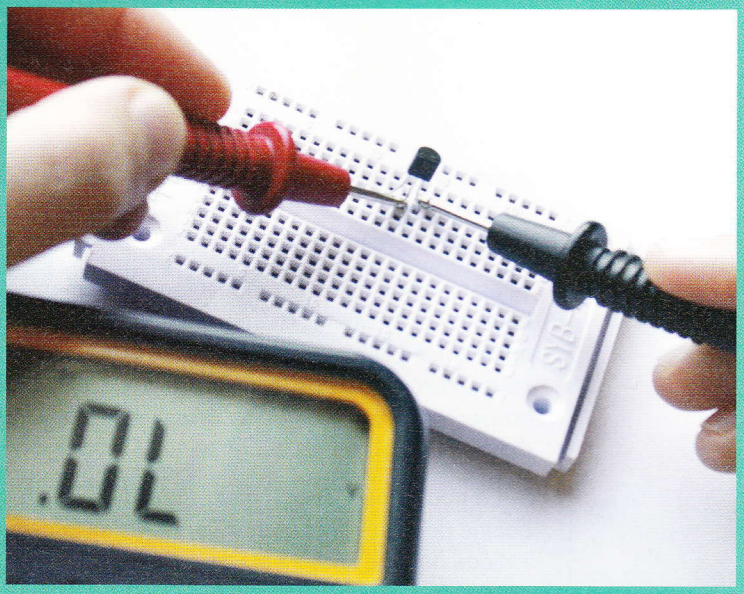


Fig. b : La sonde négative est connectée à la source, la sonde positive à une des deux autres pattes. Aucun courant ne circule, mais cela charge la grille.

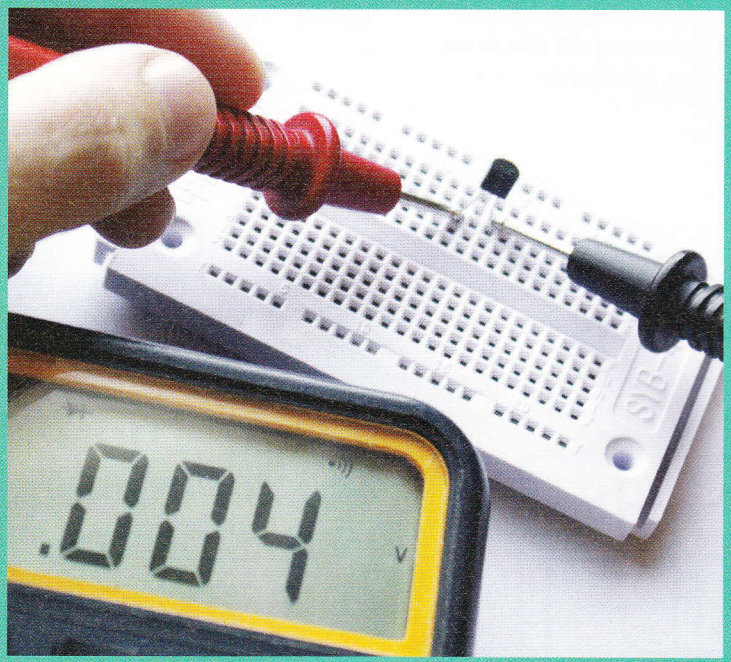


Fig. c : La sonde négative est toujours connectée à la source, la sonde positive à la dernière patte. Le courant passe avec une faible résistance donc la dernière patte est bien le drain. Si on court-circuite la source et la grille avec l'autre sonde, la résistance doit devenir infinie.

- Mesurer C - B,
- Mesurer B - C,
- Court-circuiter A, B et C,
- Mesurer A - C,
- Mesurer C - A.

Une de ces mesures donne 0,6V environ : la source est alors connectée à la sonde positive et le drain à la sonde négative. Par déduction, la broche non testée est la grille.

Si vous trouvez une chute de tension de 0,6V à deux endroits alors c'est probablement un transistor bipolaire...

Ensuite, connectez la sonde + à la grille et la sonde - à la source, puis mesurez la chute de tension entre la source et le drain (la sonde + est déplacée sur l'autre broche). La chute de tension doit être quasiment nulle pour un MOSFET canal N, au lieu d'une très forte résistance.

Pour vérifier si c'est un canal P, inverser les sondes + et - : - sur la grille, + à la source, puis - sur le drain, qui donne une tension nulle (résistance nulle).

régime ohmique. Cependant les transitions extrêmement rapides ont d'autres effets, comme la génération de parasites à haute fréquence qu'il faut filtrer...

La situation est exacerbée lorsqu'on alimente une bobine, par exemple l'électro-aimant d'un relais. Électroniquement, on la modélise par une forte inductance : la mise sous tension n'est pas un souci et le courant augmente doucement, mais l'arrêt crée un pic de tension inverse et dangereux.

La tradition consiste à mettre une diode aux bornes de la bobine pour que le courant inverse recircule dedans. Cette « diode de roue libre » est une solution du type « soigner » au lieu de « prévenir ».

Une autre solution existe : mettre un condensateur en parallèle de la bobine, pour absorber le pic et lisser la tension. Le condensateur doit être dimensionné proportionnellement à la bobine, ce qui augmente les coûts et l'encombrement.

Lorsqu'une bobine est commandée par un MOSFET, celui-ci contient une diode qui peut faire office de diode de roue libre, qui injecte l'impulsion négative vers le rail d'alimentation. Certains MOSFET de puissance ont une diode intrinsèque optimisée pour passer un fort courant, ou même une diode Schottky en parallèle.

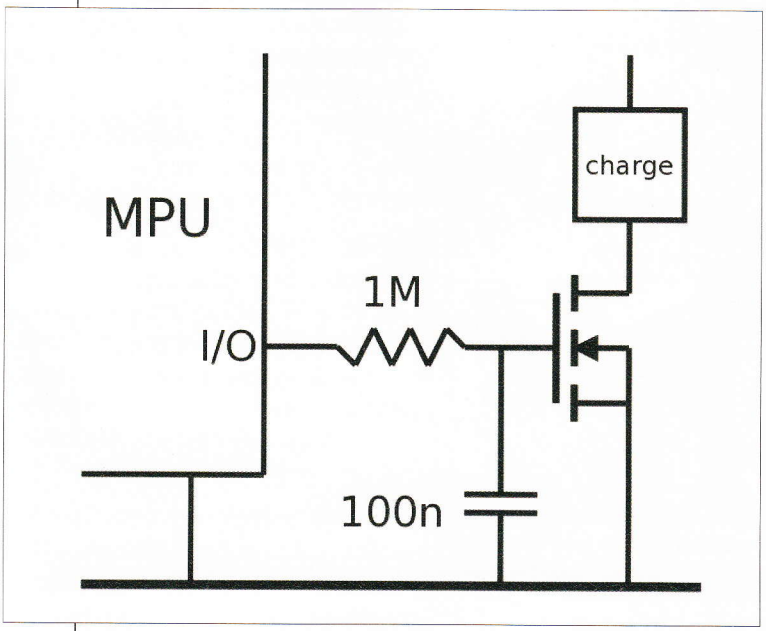


Fig. 13 : Une simple cellule RC ralentit la commutation.

Mais pour prévenir, au lieu de guérir, il faut réfléchir autrement. L'énergie du pic de courant inverse est aussi proportionnelle à la vitesse de variation du courant. Pour réduire le pic, il faut donc ralentir la variation. Le gros condensateur en aval peut être remplacé par un minuscule condensateur en amont, au niveau de la commande de la grille.

Le ralentissement augmente la durée de régime ohmique, mais cela reste transitoire donc tout à fait acceptable pour un transistor suffisamment bien dimensionné, si la commutation n'est pas trop répétitive.

La durée de la transition est facile à calculer avec la formule approximative t=R×C. Dans l'exemple ci-dessus, 1MΩ×100nF = 10^6×10^-7 = 0,1s environ. Ces valeurs sont disponibles en composants CMS ultraminiatures et économiques. Si le MOSFET de contrôle est « gros », comme celui à gauche de la première photo, il n'y a même pas besoin d'ajouter un condensateur en parallèle, une très forte résistance (10MΩ ou plus) peut suffire.

9. PROTECTION CONTRE LES INVERSIONS DE POLARITÉ

Lorsque vous concevez l'alimentation de votre circuit, il vous arrive de penser à la protéger avec une diode, afin qu'aucune mauvaise manipulation ne l'endommage. Même avec un

détrompeur, on n'est pas à l'abri d'une alimentation branchée à l'envers, par exemple (ça ne sent pas que le vécu...).

Mais voilà, la diode va ajouter 0,7V de chute à l'alimentation, donc cette dernière doit être dimensionnée en conséquence. Si votre alimentation est fixée à 5V, vous n'aurez plus que 4,3V à disposition. Et cette chute n'est pas juste en tension : cela dissipe de la puissance, pour rien.

Heureusement, il existe des diodes à faible chute, en particulier les diodes Schottky, avec une perte plus faible : environ 0,5V au lieu de 0,7, mais cela change en fonction des modèles, de la tension inverse supportée et bien sûr du courant. C'est un peu mieux, mais ce n'est toujours pas une vraie solution.

Un MOSFET peut avoir une résistance presque nulle et peut remplacer une diode. Le coût unitaire est plus élevé, mais la chute de tension est négligeable (si les bons paramètres sont choisis). Et le montage est très simple !

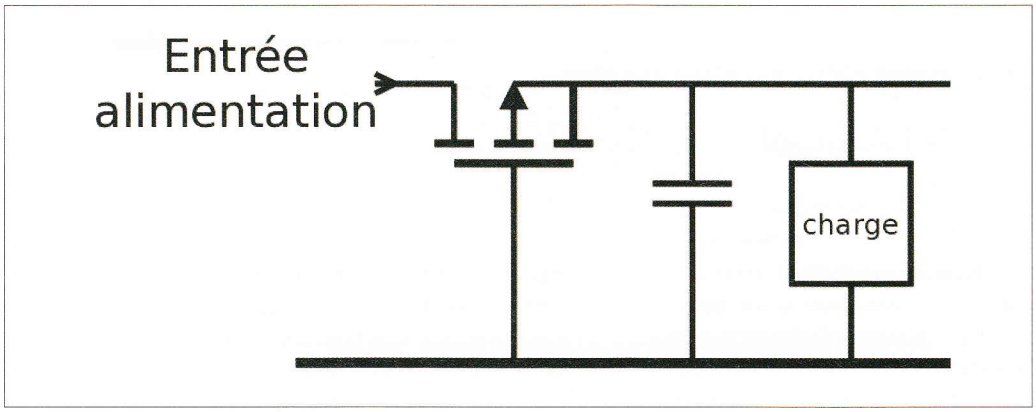


Fig. 14 : Un MOSFET peut aussi être utilisé comme une « super diode ».

Ce montage n'est pas très intuitif, mais il est très astucieux. En effet, il a l'air de fonctionner à l'envers par rapport aux montages que nous avons vus jusque là. La source est à un potentiel inférieur au drain lorsque l'alimentation est branchée correctement, le courant circule dans le sens inverse de l'utilisation normale pour un MOSFET P.

Pour mieux comprendre, souvenez-vous que le drain et la source sont interchangeables. La polarité est habituellement définie par le sens de la diode intrinsèque, mais ce n'est pas une obligation. Examinons ce qu'il se passe si l'alimentation est connectée alors que le circuit est déchargé.

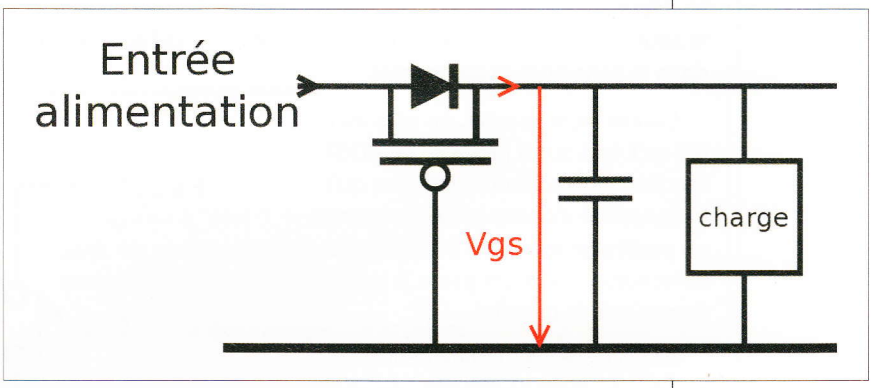


Fig. 15 : La symétrie du MOSFET ainsi que sa diode intrinsèque permettent de protéger contre les tensions négatives.

Comme le condensateur est vide, la tension V_{GS} est nulle et donc le transistor est bloqué. Mais si on fournit une alimentation positive, la diode laisse passer du courant, qui charge ensuite le condensateur. Lorsque la tension a suffisamment augmenté, le MOSFET devient totalement

passant et la charge peut alors consommer autant de courant qu'elle veut, avec une perte minimale.

Au contraire, si la tension d'entrée est négative, la diode interne ne permet pas de faire passer le courant. Le transistor ne devient pas non plus passant, car la tension est dans le mauvais sens. Le circuit est protégé tant que la tension de grille ne dépasse pas la limite supportée par le composant.

Le montage fonctionne aussi bien en version « *High Side* » (Canal P, sur l'illustration ci-dessus) qu'en « *Low Side* » avec un canal N. Il faut faire attention au sens du drain et de la source, en orientant la diode intrinsèque dans le sens normal du courant.

L'autre inconvénient, en plus du prix, est qu'il faut aussi protéger le MOSFET des décharges électrostatiques qu'il est possible d'observer sur la source, en particulier lorsqu'on branche une alimentation qui n'est pas à la même masse que le circuit.

Enfin, le circuit protège contre les inversions, mais pas contre une chute de tension : le transistor saturé laisse passer le courant dans l'autre sens et si l'alimentation baisse, cela baissera aussi la tension du condensateur de filtrage. Une électronique plus sophistiquée est nécessaire pour transformer le MOSFET en une vraie diode idéale.

10. COUPER L'ALIMENTATION D'UN MONTAGE

Les circuits intégrés actuels, tels les microcontrôleurs, se vantent d'une consommation très très faible au repos, afin de prolonger la durée d'utilisation des appareils portables. Un appareil éteint ou en veille ne devrait rien consommer et préserver la charge de la batterie.

Cependant un système électronique ne se résume pas à un microcontrôleur et peu de composants ont un mode de mise en veille. Il faut alors couper leur alimentation, soit sélectivement (au moyen de MOSFET P par exemple) soit en coupant tout, tout simplement.

C'est ce que nous allons voir avec un MOSFET N, qui a une résistance plus faible qu'un MOSFET P équivalent. Dans le système suivant, le microcontrôleur commande la mise sous tension et l'extinction, ce qui coupe toute l'alimentation une fois que le programme à exécuter est terminé.

Tout d'abord, il faut allumer le MOSFET. Pour éviter l'allumage intempestif, sa grille doit être tirée vers la source par une résistance de forte valeur, comme nous avons vu précédemment.

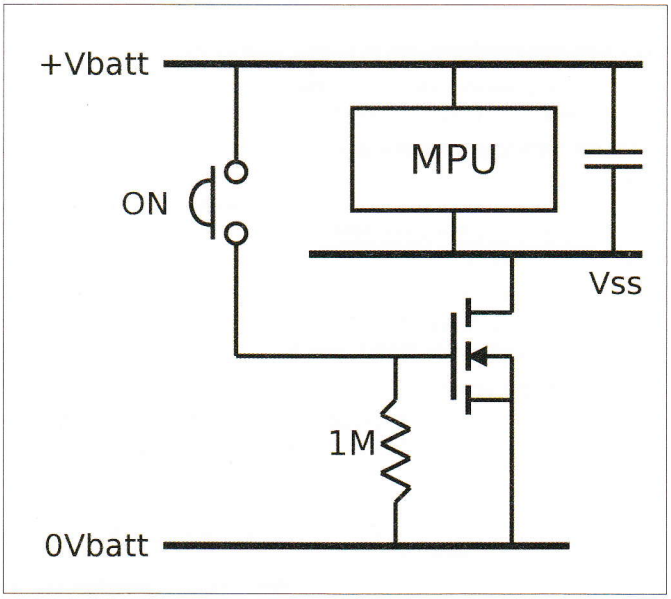


Fig. 16 : Un bouton poussoir permet d'allumer le montage, mais il faut ensuite le garder allumé...

Une fois le montage mis sous tension, il faut le maintenir dans cet état, car il s'éteindra dès que l'impulsion (qui peut être un contact mécanique fugitif ou tout autre capteur) sera terminée. Le microcontrôleur doit prendre le relais et forcer la grille à l'état haut avec une broche d'entrée-sortie. Cependant, si la broche est connectée directement à la grille, de nouveaux problèmes se posent :

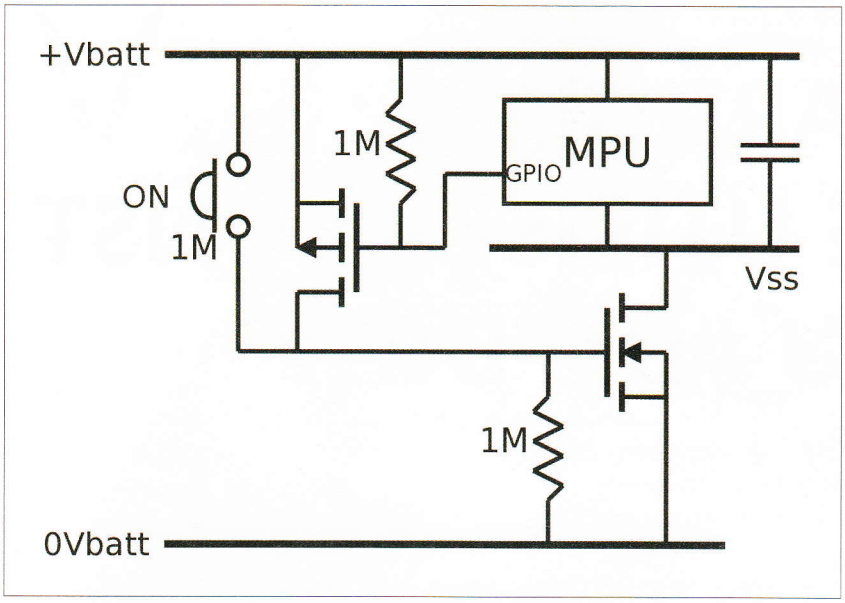


Fig. 17 : Un deuxième MOSFET, beaucoup plus petit, remplace le bouton poussoir durant le fonctionnement du circuit.

- Si la broche est forcée à 0 alors que l'impulsion force la grille à **+Vbatt**, un conflit électrique se produit et peut endommager la broche.
- Si la broche passe à 0, alors le MOSFET coupe, ce qui remonte le potentiel du niveau '0' du microcontrôleur. Cela se transforme en '1' sur la grille du MOSFET qui redevient passant, ou ne quitte pas cet état.

En fait, on voudrait créer un autre contact en parallèle du premier pour résoudre tous les conflits. Et justement, les MOSFET le font très bien : un MOSFET canal P peut connecter la grille du MOSFET N vers l'alimentation positive.

Le programme du microcontrôleur se déroule ainsi :

- Dès la mise sous tension, la broche d'entrée-sortie est configurée à l'état 0, soit 0V, et le MOSFET P devient passant, en parallèle du bouton poussoir, ce qui bloque l'alimentation à l'état allumé.
- Le microcontrôleur remplit sa fonction.
- Pour éteindre l'alimentation, la broche GPIO passe à l'état 1 ou peut flotter. La grille du MOSFET P remonte à **+Vbatt**, celle du MOSFET N est tirée à 0 par la résistance, ce qui déconnecte le **Vss** du circuit. Le microcontrôleur doit alors vider le condensateur dans une boucle infinie qui consomme de l'énergie.

Une fois déconnecté, le montage ne consomme plus rien du tout.

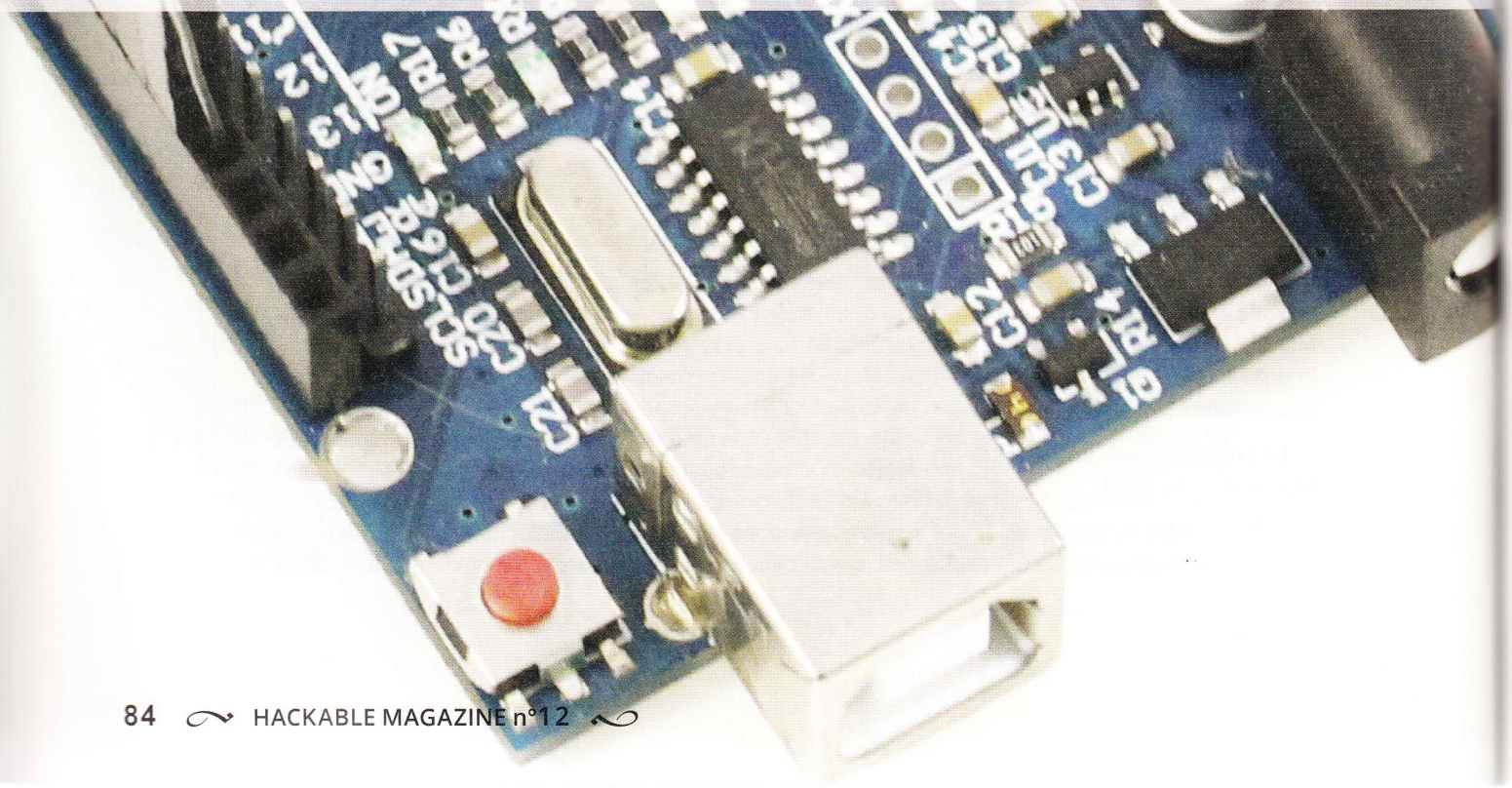
CONCLUSION

Nous venons de survoler quelques aspects théoriques des MOSFET sous l'angle de l'application pratique. Leur nom peut paraître barbare et leurs propriétés surprennent au début, mais les MOSFET sont vraiment nos amis ! Nous n'utiliserons qu'une petite partie de leurs propriétés, mais vous savez maintenant qu'ils peuvent vous rendre de grands services. Pour vous aider à vous y retrouver quand vous vous lancerez seul, le prochain numéro vous expliquera comment décrypter le jargon technique et les abréviations obscures que contiennent les *datasheets*, ces documents denses et exhaustifs, mais incontournables. YG

ARDUINO : DOIS-JE UTILISER CONST OU #DEFINE ?

Denis Bodor

Ce que soit dans les bibliothèques ou dans les croquis partagés entre amateurs du domaine, lorsqu'il s'agit de définir une valeur invariable, deux solutions s'offrent à vous : utiliser #define pour définir une macro ou déclarer une variable avec la directive const comme « constante ». Quelle solution choisir ? L'une est elle meilleure que l'autre ? La réponse est simple… Ça dépend.



Voici ce qu'on appelle une « réponse de Normand ». Notez au passage que l'expression semble provenir d'une ancienne loi normande offrant une possibilité de rétractation sur un jour à compter de la signature d'un accord marchand. Personnellement, je préfère la déclinaison anglaise de l'expression, « une réponse de politicien », qui a le mérite de ne pas être péjorative, mais tout simplement pragmatique...

Mais revenons à nos moutons (non, je ne vous ferai pas le coup deux fois) : **#define** ou **const** ? Pour répondre à cette question, la première chose à faire est de savoir de quoi on parle.

1. #DEFINE

Cette instruction ne concerne pas le compilateur qui va transformer le code que vous écrivez en une série d'instructions en langage machine pour le processeur. Celle-ci est traitée par le préprocesseur C qui opère avant la compilation. Ce préprocesseur gère également les autres directives débutant par **#** comme **#include** utilisé pour inclure un fichier d'entête (**.h**) dans un programme.

Une macro définie et utilisée ainsi :

```
#define SORTIE  7
digitalWrite(SORTIE, HIGH);
```

ne sera pas présente dans le code passé au compilateur qui verra tout simplement cela :

```
digitalWrite(7, 0x1);
```

Oui, **HIGH** est aussi une macro. Ce type de définition est donc un simple alias. Et cela ne se limite pas à une valeur, vous pouvez définir une macro pour opérer n'importe quel remplacement avant la phase de compilation. Ainsi, ceci fonctionnera très bien :

```
#define TOUJOURS for( ; ; )

TOUJOURS {
  Serial.println("Coucou");
  delay(1000);
}
```

Il est même possible de passer des arguments aux macros et donc de demander au préprocesseur de travailler un peu plus :

```
#define PLUSMILLE(N) ((N) + 1000)

int valeur = PLUSMILLE(142);
```

valeur contiendra alors 1142.

2. CONST

Cette directive ne s'utilise pas seule, ce n'est pas un type, mais un qualificateur permettant de signifier au compilateur que la variable déclarée ne doit pas changer. Ceci par exemple est interdit :

```
const int toto = 12;

toto = 42;
```

Vous serez immédiatement rappelé à l'ordre :

```
essai.ino: In function 'void setup()':
essai.ino:68:8: error: assignment of
read-only variable 'toto'
assignment of read-only variable 'toto'
```

Ou en bon français : « Soyez gentil de ne pas faire n'importe quoi, il n'est pas question que je change une variable que vous avez déclarée vous-même en lecture seule ».

Ce qualificateur est donc très utile pour éviter les erreurs. Ainsi, une variable que vous savez pertinemment ne jamais devoir changer sera déclarée en **const** et le compilateur vous assistera pour repérer d'éventuelles erreurs. Mais ce n'est pas tout, **const** donne également une indication au compilateur sur la nature d'une variable et donc sur la façon dont il peut optimiser le code pour vous faire économiser de la mémoire vive.

3. ET MA RÉPONSE ALORS ?

Il n'y a pas de réponse, mais uniquement des arguments pour l'un et pour l'autre alors que les deux ne s'opposent pas réellement. L'une des raisons généralement avancées au bénéfice de l'utilisation de **const** est le fait qu'une macro n'a pas de type et peut donc être source d'erreurs et de confusion. En réalité, les choses ne sont pas si évidentes. Voici un exemple :

```
const char TOTO[]= "coucou";

int zou(int val) {
  return val*2;
}

void setup() {
  zou(TOTO);
}
```

La fonction **zou()** attend en argument une variable de type **int**, mais nous avons une variable **TOTO** qui est un tableau de caractères. Le compilateur nous signale l'erreur :

```
essai_def.ino: In function 'void setup()':
essai_def.ino:12:11: error: invalid conversion
from 'const char*' to 'int' [-fpermissive]
essai_def.ino:7:5: error: initializing
argument 1 of 'int zou(int)' [-fpermissive]
invalid conversion from 'const char*' to 'int'
[-fpermissive]
```

zou() veut un **int** et nous lui donnons autre chose. Le même code utilisant un **#define** :

```
essai_def.ino: In function 'void setup()':
essai_def.ino:12:11: error: invalid conversion
from 'const char*' to 'int' [-fpermissive]
essai_def.ino:7:5: error:   initializing
argument 1 of 'int zou(int)' [-fpermissive]
invalid conversion from 'const char*' to 'int'
[-fpermissive]
```

nous retourne exactement la même erreur. Certes, le préprocesseur ne s'occupe pas du type, mais le compilateur réagit de la même manière. Les problèmes peuvent survenir dans certains cas particuliers en programmation C++, dans le cas d'une surcharge par exemple, avec des types qui peuvent paraître similaires au compilateur. C'est le cas par exemple du type **boolean** et du type **int**. Avec une déclaration :

```
const boolean vrai = 1;
```

il n'y a pas d'ambiguïté, **true** ne sera pas pris pour un **int** comme ce serait le cas avec :

```
#define vrai 1
```

Mais là encore, c'est une question d'utilisation et d'interprétation car, pour régler le problème il suffirait de faire :

```
#define vrai (boolean)1
```

Un autre cas peut sembler aller contre l'utilisation de **#define** au bénéfice de **const**. Imaginez que vous ayez, dans une bibliothèque, une ligne comme :

```
#define DEBUG
```

Puis, dans votre croquis, pour votre usage personnel, vous utilisez quelque chose comme :

```
const boolean DEBUG = 1;
```

Vous obtiendrez une erreur. Mais ceci est également vrai avec d'autres formes de doubles déclarations, et peut être réglé avec d'autres instructions à destination du préprocesseur comme **#ifdef**, **#ifndef** et **#undef**.

Enfin, nous avons l'argument de l'erreur facilement détectée comme :

```
int laled = 13;

if(laled = 7) {};
```

Là, le compilateur ne dira rien alors que vous avez une assignation dans un **if()** et non un test (**==**). En déclarant votre **laled** ainsi, l'erreur sera signalée :

```
const int laled = 13;
```

Mais cela n'est pas un argument contre **#define**, car le résultat sera le même avec :

```
#define laled 13
```

Seul le message change :

- « *assignment of read-only variable 'laled'* » avec **const** : « vous essayez d'assigner une valeur à une variable en lecture seule » ;
- et « *lvalue required as left operand of assignment* » dans le second : « l'opérande de gauche doit être un emplacement, une variable (*lvalue*) ».

De façon générale cependant, on s'entend à dire qu'une déclaration avec **const** est *type-safe*, car le compilateur connaît le type de la variable, et *scope-safe*, car un **#define** est forcément global (dans une unité source d'un fichier objet) alors qu'une variable peut être locale (à une fonction). Mais ce dernier point est presque rhétorique, car le principe même d'une constante est d'être invariable, jusqu'à son existence même, et ce pour toutes les fonctions. C'est une question de point de vue, mais personnellement si **LED_VERTE** est **7**, parce que j'ai l'anode d'une led verte connectée à la broche 7 d'une carte Arduino, c'est tout aussi vrai dans **setup()** que dans **loop()**...

CONCLUSION : ÇA DÉPEND DE VOUS

En effet, tout cela repose sur la compréhension de **#define** et de **const** ainsi que de votre rigueur de programmeur. Il suffit donc d'utiliser l'une et l'autre directive, en sachant bien à quoi elles servent, à qui elles s'adressent (préprocesseur ou compilateur) et comment elles sont traitées dans la chaîne de compilation. S'il faut retenir deux choses de tout cela, ce serait :

- **#define** est l'équivalent d'un « rechercher/remplacer » dans l'éditeur avant la compilation, alors que **const** qualifie une variable ;
- vous devez faire attention à la manière dont vous nommez les choses !

Ce n'est pas une obligation, mais de façon générale, et depuis des décennies, les macros sont entièrement nommées en majuscules par convention alors que les noms de variables ne le sont jamais. Les conventions existent pour une bonne raison et même si vous avez un esprit rebelle, tenez-vous-y... ou vous finirez par vous en mordre les doigts car, un jour ou l'autre, vous retomberez sur l'un de vos propres codes et ce choix vous jouera alors des tours.

Les conventions existent au-delà de la façon de nommer une macro ou une variable et dépendent de ce sur quoi vous travaillez et avec qui vous travaillez. L'utilisation des macros pour certaines choses entre également dans ce type de conventions afin que les programmeurs puissent immédiatement comprendre ce qu'ils voient en regardant un code. Il n'y a pas d'argument absolu pour ou contre **#define**, ou pour ou contre l'utilisation systématique de **const**, et en particulier pas des choses comme « *#define date d'avant const* », « *tout le monde utilise const* », « *const est mieux que #define* »...

La seule réponse valable à la question est donc : ça dépend de votre code, des circonstances, de l'utilisation que vous en faites et de vos préférences.

Note : et pour ceux sur le point de hurler que dans la documentation officielle (**https://www.arduino.cc/en/Reference/Const**) il est dit « *In general const is preferred over #define for defining constants* », je les invite à consulter le fichier **hardware/arduino/avr/variants/standard/pins_arduino.h** :

```
#define LED_BUILTIN 13
```

DB

GARDEZ UNE RASPBERRY PI À JOUR SANS INTERNET

Denis Bodor

Voilà un titre bien accrocheur, mais il n'est pas question ici de comprendre « sans Internet » dans le sens « aucune connexion nulle part ». En effet, les paquets de la distribution Raspbian doivent toujours être téléchargés à un moment ou un autre... mais pas forcément par la Raspberry Pi elle-même, qui peut être alors ponctuellement mise à jour ou se voir ajouter des applications et outils sans connexion directe au net.

Notre scénario initial est relativement simple : vous disposez d'une Raspberry Pi d'un modèle quelconque qui se trouve installée dans un ensemble fini comme un projet abouti et « en production » (capteur, centrale, robot, etc.). Votre installation ne permet aucune connexion filaire ou Wifi à un quelconque réseau (ou est connecté à un réseau qui lui-même n'a pas accès à Internet). La question qui se pose alors est : comment mettre à jour le système et avoir l'opportunité, si nécessaire d'ajouter outils, applications, services ou bibliothèques ? La réponse est simple et tient en quelques mots : un miroir Raspbian et une clé USB.

Une clé USB de 128 Go comme celle-ci vous coûtera actuellement une centaine d'euros et vous assurera d'avoir suffisamment d'espace, à la fois pour le miroir et pour des données complémentaires ou sauvegardes. L'option 64 Go fonctionnera également, mais il est probable que cela ne soit plus suffisant d'ici quelque temps, les distributions GNU/Linux ayant tendance à prendre de plus en plus de poids au fil des versions...

1. LE MENU

Le système équipant généralement les Raspberry Pi est une distribution GNU/Linux appelée Raspbian basée sur Debian GNU/Linux. Comme évoqué à plusieurs reprises dans le magazine, le rôle d'une distribution GNU/Linux est de regrouper un ensemble de logiciels et bibliothèques et de structurer le tout de façon à permettre l'installation, la désinstallation et la mise à jour de ces éléments, de façon structurée, cohérente et simple pour l'utilisateur. Avec Raspbian comme avec quasiment toutes les distributions GNU/Linux, l'ensemble des éléments prend la forme de paquets mis à disposition sur des serveurs web appelés dépôts (*repository*). Lorsque l'utilisateur souhaite installer ou mettre à jour un élément, il utilise un outil du système de gestion de paquets pour le télécharger et procéder à son installation.

Il est possible de configurer ce système de gestion de paquets de façon à utiliser différents dépôts pour obtenir des logiciels provenant de plusieurs sources. C'est le cas par défaut avec Raspbian. En effet, nous avons d'une part un dépôt défini dans le fichier `/etc/apt/sources.list` pour les paquets propres à Raspbian provenant de **http://mirrordirector.raspbian.org/raspbian/** (`mirrordirector` est une solution pour équilibrer les téléchargements de façon à ne pas saturer un unique serveur) et d'autre part, dans `/etc/apt/sources.list.d/raspi.list`, un autre dépôt, **http://archive.raspberrypi.org/debian/**, géré par la fondation de manière indépendante à Raspbian.

Chacun de ces dépôts contient une série de paquets sous forme de fichiers `.deb`, mais également un index (`Packages`) contenant diverses informations utiles sur chaque paquet (description, dépendances, version, catégorie, etc.).

Mais cette configuration, reposant sur l'utilisation de différents serveurs web (HTTP), peut également faire usage d'autres médias pour le stockage des paquets, comme un CD-ROM, un DVD, un serveur sur votre réseau local ou encore une clé USB... à condition que l'espace de stockage soit adapté.

L'idée est donc relativement simple : maintenir une copie à jour de tous les dépôts utiles pour une Raspberry Pi sur un support USB à partir d'un système connecté au net, puis utiliser ce même support pour mettre à jour et/ou installer des paquets sur une ou plusieurs Raspberry Pi non connectées.

Notez que cette technique permet également d'isoler physiquement un système d'Internet, ce qui peut être intéressant pour en améliorer la sécurité.

2. ON MET LE COUVERT !

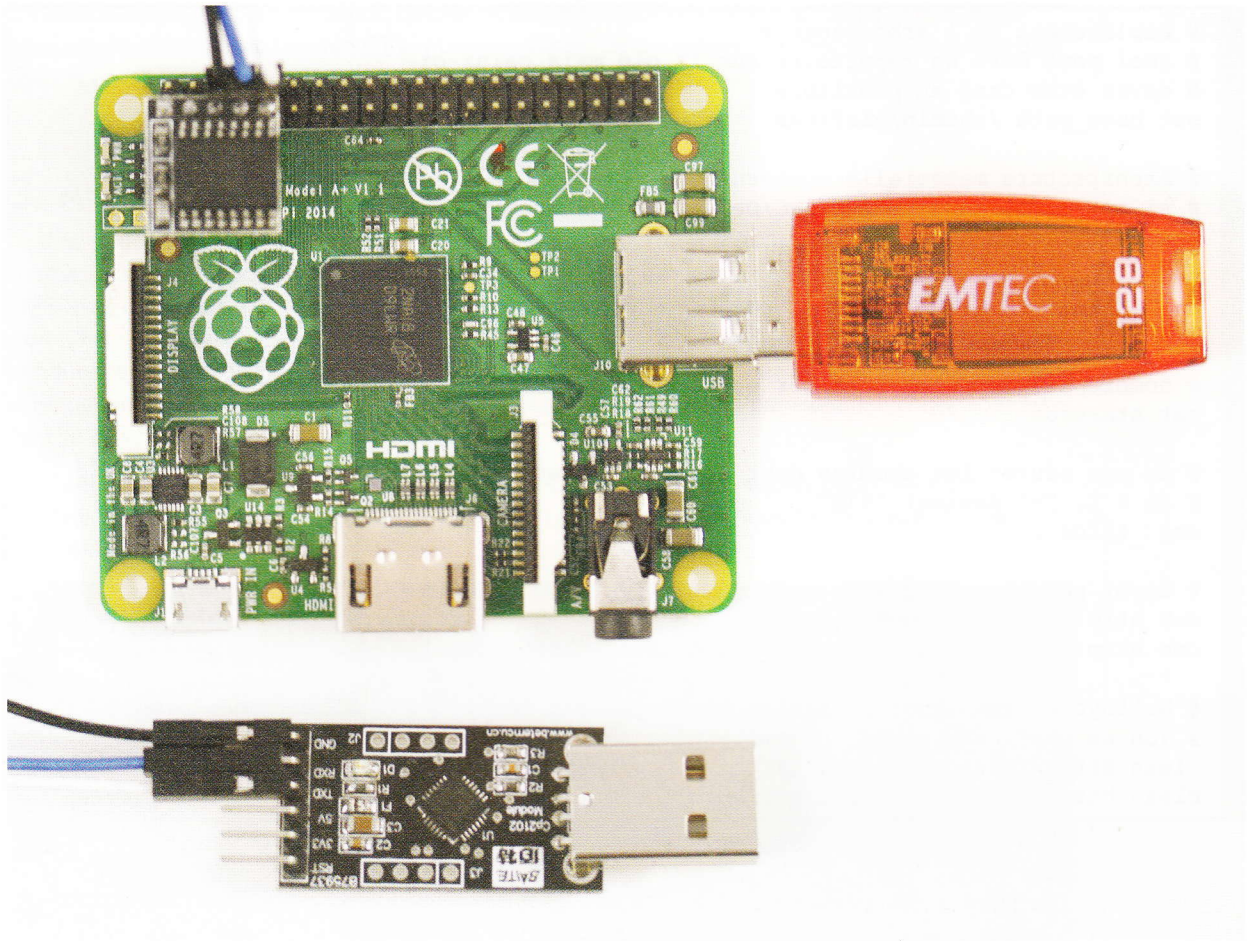
Avant de débuter, comprenez bien qu'il s'agit de maintenir une copie locale de l'ensemble des paquets pour Raspberry Pi (de Raspbian et de la fondation), aussi bien ceux que vous comptez installer mais, en plus grande quantité, ceux que vous n'utiliserez peut-être jamais. Nous parlons ici de respectivement 60 Go et 1,1 Go, à télécharger au moins une fois entièrement et à stocker sur un support amovible (disque dur ou clé USB). Ce volume devant être tenu à jour régulièrement pour correspondre aux fichiers présents sur les serveurs, il n'est pas impossible que d'ici quelques mois, il se voit majoré de centaines de Mo.

Nous baserons nos explications sur l'utilisation des deux dépôts par défaut, mais vous pouvez, bien entendu, également en ajouter d'autres dans le processus et ainsi consommer davantage d'espace. Il pourra être également intéressant de maintenir une copie des paquets « sources » si vous trouvez une utilité à reconstruire ou modifier des paquets.

Ainsi, à cette date, si vous optez pour les deux dépôts par défaut, une clé USB de 64 Go devrait suffire, mais il est plus raisonnable d'en choisir une de 128 Go pour éviter un second achat d'ici quelque temps et surtout, soit un nouveau téléchargement intégral, soit une copie de l'ensemble d'une clé à une autre (ce qui risque de prendre autant de temps).

L'outil utilisé pour obtenir une copie de l'intégralité des dépôts visés est `apt-mirror` (paquet éponyme). Ceci sous-entend donc la mise en œuvre d'un système GNU/Linux Debian ou compatible. Il peut s'agir d'un PC sous Debian GNU/Linux, Ubuntu ou Mint ou encore d'une machine virtuelle fonctionnant dans un autre système comme Windows ou Mac OS X. Une dernière solution pourra être une autre Raspberry Pi bien sûr ! C'est un très bon choix, ne serait-ce qu'en raison de consommation électrique durant le téléchargement. Oui, quelques 62 Go à télécharger sous forme d'une multitude de petits fichiers est quelque chose qui peut prendre une nuit. Mieux vaut donc laisser « tourner » une Pi qu'un PC de quelques 700 watts.

Quels que soient le système et la plateforme choisis, vous pourrez installer `apt-mirror`. Celui-ci est normalement pleinement intégré au système et, une fois configuré, tiendra régulièrement à jour la copie miroir du ou des dépôts sélectionnés. Une utilisation courante de cet outil consiste, tout simplement, à disposer d'une copie de tous les paquets disponibles localement, comme sur un PC portable, et donc de ne pas avoir besoin d'une connexion internet permanente pour installer des logiciels. `apt-mirror` peut aussi être utilisé pour créer un serveur web faisant office de dépôt pour tout un réseau (maintien du miroir la nuit et installation rapide le jour au travers du réseau local).



Une Raspberry Pi A+, une clé USB et une horloge DS3231/ DS1337 (RTC) et vous voici fin prêt pour vous passer ponctuellement d'Internet et gérer votre système de façon totalement autonome, sans connexion Ethernet ou Wifi !

Dans notre cas, il s'agira plutôt d'une utilisation ponctuelle suivant la logique : on insère la clé USB, on synchronise la copie locale avec les dépôts officiels, on retire la clé et on l'utilise avec une ou plusieurs Raspberry Pi. Mettre à jour un miroir déjà créé ne demande généralement pas beaucoup de temps, en particulier si l'on procède à cette synchronisation une fois par semaine ou par mois. Notez que ce qui va suivre marchera tout aussi bien pour une Raspberry Pi que pour un PC ou un Mac sous Debian, Ubuntu ou Mint. Il ne s'agit que d'utiliser les bons dépôts.

Une autre spécificité de notre présente utilisation concerne l'endroit où sont stockées les données : directement sur support USB et non dans le cœur du système (`/var/spool/apt-mirror` par défaut). Ceci implique donc que la clé USB soit connectée et accessible par le système au moment où nous lançons `apt-mirror` pour synchroniser les fichiers.

Une fois `apt-mirror` installé, il nous faudra nous pencher sur son fichier de configuration, `/etc/apt/mirror.list`, et nous en revoyons le contenu ainsi :

```
# Emplacement de l'arborescence
# ceci peut être un répertoire sur la clé mais celui-ci
# devra être créé au préalable
set base_path /chemin/clef/usb

# Architecture matérielle concernée
# si apt-mirror est utilisé PC/Debian ou une machine virtuelle
set defaultarch armhf

# Ne pas lancer les scripts de fin
set run_postmirror 0

# nombre de processus à lancer en même temps
set nthreads     20

# ne pas sauter les chemins dont le nom contient un tilde (~)
# si à 1, "~" devient "%7E"
set _tilde 0

# dépôt pour lesquels maintenir une copie
deb http://mirrordirector.raspbian.org/raspbian/ jessie main contrib non-free rpi
deb http://archive.raspberrypi.org/debian/ jessie main ui

# Nettoyer après synchronisation en supprimant les paquets locaux obsolètes
# (on ne garde pas ce qui n'existe plus sur les dépôts distants)
clean http://mirrordirector.raspbian.org/raspbian/ jessie main contrib non-free rpi
clean http://archive.raspberrypi.org/debian/ jessie main ui
```

Deux lignes sont extrêmement importantes :

- **base_path** précise l'endroit où sont placés les éléments du miroir lui-même, mais également des répertoires servant à la gestion du miroir. On trouvera là, un sous-répertoire **mirror/** contenant un répertoire par dépôt synchronisé, un sous-répertoire **skel/** pour les fichiers temporaires et un sous-répertoire **var/** pour les journaux d'activité et les sommes de contrôle (pour assurer l'intégrité des données) ;
- **defaultarch** qui précise pour quelle architecture nous souhaitons un miroir des paquets. Par défaut, c'est l'architecture identique à celle où est lancée la commande qui est utilisée (**armhf** sur une Raspberry Pi, mais **i386** ou **amd64** sur PC/Mac). En lançant **apt-mirror** sur une Raspberry Pi, cette ligne sera inutile puisque nous sommes sur une architecture **armhf**, mais sur PC/Mac ou avec une machine virtuelle elle sera nécessaire, pour ne pas récupérer les paquets « PC » (architecture x86), mais bien ceux pour les Pi (ARM).

En fin de fichier de configuration, nous trouvons les lignes débutant par **deb** et **clean** et reprenant littéralement le contenu des fichiers **/etc/apt/sources.list** et **/etc/apt/sources.list.d/raspi.list** dans une installation Raspbian par défaut. En effet, les dépôts ainsi spécifiés dans la distribution contiennent forcément l'ensemble des paquets installables et c'est, bien entendu, précisément tout cela que nous comptons récupérer.

Notez la présence des lignes **clean** qui évitent d'encombrer notre copie locale d'anciens fichiers. En effet, les paquets possèdent un numéro de version directement dans le nom de fichier.

En cas de mise à jour sur un dépôt, l'ancien fichier laisse place au nouveau. Sans `clean`, nous nous retrouverions avec l'ancien ET le nouveau fichier, ce qui est encombrant et pas vraiment utile dans la plupart des cas.

3. À TABLE !

Il ne nous reste plus qu'à lancer la commande après nous être assuré du chemin correspondant à la clé USB et des permissions s'y rapportant. Notez qu'il est question ici d'utiliser la clé USB avec un système GNU/Linux d'un côté comme de l'autre. L'idée est donc de formater le support dans un format propre à ce système pour éviter des soucis. Les clés USB sont généralement vendues formatées en FAT ou en FAT32 pour Windows, et même si Mac OS X et GNU/Linux s'en arrangent bien, il est préférable ici de la reformater en ext2/ext3/ext4 par exemple (l'outil Gparted fera ça très bien sous GNU/Linux).

Étant donné que nous ne travaillons qu'avec des répertoires accessibles par l'utilisateur courant, nous n'avons pas de problème de gestion de permissions, nous n'avons qu'à exécuter la commande :

```
% apt-mirror
Downloading 60 index files using 20 threads...
Begin time: Sun Dec 20 09:42:54 2015
[20]... [19]... [18]... [17]... [16]... [15]... [14]...
[13]... [12]... [11]... [10]... [9]... [8]... [7]...
[6]... [5]... [4]... [3]... [2]... [1]... [0]...
End time: Sun Dec 20 09:43:24 2015

Processing tranlation indexes: [TT]

Downloading 0 translation files using 0 threads...
Begin time: Sun Dec 20 09:43:24 2015
[0]...
End time: Sun Dec 20 09:43:24 2015

Processing indexes: [PP]

60.1 GiB will be downloaded into archive.
Downloading 46429 archive files using 20 threads...
Begin time: Sun Dec 20 09:43:30 2015
[20]...
```

L'outil calcule automatiquement la masse de données à télécharger et la manière de procéder. Notez que ceci aura pour effet d'accaparer presque toutes les ressources systèmes et en particulier la connexion Internet et les accès disque/USB. Vous pourrez suivre cela avec la commande `sudo iftop` (du paquet du même nom). La durée de l'opération sera dépendante du débit Internet et de la rapidité de votre clé ou disque USB. Je vous recommande cependant de lancer cela, par exemple, avant d'aller vous coucher afin de ne pas gêner votre utilisation courante de la connexion internet même par d'autres machines (mon fils en a fait les frais alors qu'il jouait à « Assassin's Creed IV » sur la PS3. D'un autre côté, il harponnait des baleines... ce n'est que justice).

Vous pouvez interrompre l'exécution de la commande avec le raccourci Ctrl+C. Vous ne serez pas obligé de reprendre l'ensemble du téléchargement. Lors du prochain lancement de **apt-mirror**, l'outil va analyser les dépôts distants et la copie locale puis compléter l'opération en vous affichant les informations en conséquence :

```
26.8 GiB will be downloaded into archive.
Downloading 9390 archive files using 20 threads...
```

Une fois le processus terminé, **apt-mirror** s'arrête avec quelque chose comme :

```
[...]
End time: Mon Dec 21 17:55:54 2015
0 bytes in 0 files and 0 directories can be freed.
Run /home/denis/MIRROR/var/clean.sh for this purpose.
```

Notez le message concernant **clean.sh**, le script de nettoyage, qui s'affiche indépendamment de l'état des fichiers à nettoyer. Ceci est un simple message d'information qui ne nous concerne pas puisque nous avons prévu ce nettoyage directement dans la configuration du fichier **/etc/apt/mirror.list**.

Après cette synchronisation, nous nous retrouvons avec les répertoires **mirror**, **skel** et **var** sur la clé USB (ou dans un sous-répertoire si tel est le choix que vous avez fait). Il nous suffira alors d'éjecter (démonter) la clé pour nous tourner vers la Raspberry Pi à mettre à jour. Plus tard, pour vous assurer que votre miroir est synchronisé, reconnectez la clé, vérifiez que les chemins soient bons et lancez à nouveau **apt-mirror**, plus ou moins régulièrement. Cette manipulation prendra bien moins de temps, puisque la majeure partie des téléchargements effectués sera toujours valable (et votre fils pourra continuer jouer au pirate sans être dérangé).

4. ON SE TOURNE VERS LA RASPBERRY PI

4.1 Un peu de préparation

Contrairement à un système Debian, Mint ou Ubuntu, Raspbian est bien plus simple et bien plus orientée « ligne de commandes ». Contrairement à ces distributions PC/Mac, avec Raspbian Jessie Lite en particulier (choix idéal pour une Pi intégrée dans un projet), la connexion d'une clé USB ne provoque pas automatiquement la mise en place d'un accès à cette dernière. La connexion génère simplement la détection d'un nouveau disque et l'utilisateur doit « monter » le système de fichiers qui s'y trouve. Celui-ci apparaît alors comme un contenu de répertoire. Une fois l'accès devenu inutile, on « démonte » le système de fichiers qui n'est alors plus accessible par le système et on peut retirer la clé.

Le problème qui se pose généralement est l'identification de la clé, car le nom du disque est dépendant de l'ordre dans lequel on branche plusieurs clés ou disque. **/dev/sda1** est la première partition du premier disque de ce type, **/dev/sdb1** est la première partition du second disque, etc. Mais en branchant l'une ou l'autre clé, la première sera toujours **/dev/sda** et la première partition **/dev/sda1**.

Il y a une solution : les disques sont également identifiés par un numéro unique (UUID), un label, un chemin ou un identifiant. L'UUID peut alors être utilisé comme un **/dev/sda1** ou un **/dev/sdb1**, mais sans risque de confusion, il suffit d'utiliser **/dev/disk/by-uuid/ee290474-ffbe-4f5d-a4e0-06bf55c96f00** par exemple.

Pour nous simplifier la vie, nous allons donc nous servir de cela pour procéder à une configuration du système qui facilitera l'accès ponctuel à la clé USB. Dans un premier temps, connectez la clé à la Pi puis affichez les derniers messages système :

```
[491353.587152] usb-storage 1-1.4.1:1.0: USB Mass Storage device detected
[491353.601405] scsi host1: usb-storage 1-1.4.1:1.0
[491354.598849] scsi 1:0:0:0: Direct-Access     ICYBOX   IB-226StUE2-Wh   2.27 PQ: 0 ANSI: 4
[491354.603960] sd 1:0:0:0: [sda] 1953525168 512-byte logical blocks: (1.00 TB/931 GiB)
[491354.604811] sd 1:0:0:0: [sda] Write Protect is off
[491354.604859] sd 1:0:0:0: [sda] Mode Sense: 23 00 00 00
[491354.605955] sd 1:0:0:0: [sda] No Caching mode page found
[491354.606000] sd 1:0:0:0: [sda] Assuming drive cache: write through
[491354.607084] sd 1:0:0:0: Attached scsi generic sg0 type 0
[491354.654770]  sda: sda1
[491354.660399] sd 1:0:0:0: [sda] Attached SCSI disk
```

Ici, par exemple, un disque USB (*USB Mass Storage device detected*) a été détecté. Nous apprenons qu'il est représenté par **sda** et que sa taille est de 1 To. Sur l'avant-dernière ligne, nous voyons également que ce disque **sda** possède une seule partition **sda1**. Nous savons donc que, à ce moment précis, ce qui nous intéresse se trouve sur **/dev/sda1**. Nous jetons ensuite un œil à **/dev/disk/by-uuid** :

```
$ ls -l /dev/disk/by-uuid/
total 0
lrwxrwxrwx 1 root root 15 déc.  18 16:40 7771-B0BB -> ../../mmcblk0p1
lrwxrwxrwx 1 root root 15 déc.  18 16:40 c7f58a52-6b71-4cea-9338-65f3b8af27bf ->
../../mmcblk0p2
lrwxrwxrwx 1 root root 10 déc.  24 09:09 ee290474-ffbe-4f5d-a4e0-06bf55c96f00 ->
../../sda1
```

Les deux premières lignes **mmcblk0p1** et **mmcblk0p2** correspondent respectivement à la première et la seconde partition de la carte microSD (ou SD) de la Raspberry Pi. Mais nous voyons également que **ee290474-ffbe-4f5d-a4e0-06bf55c96f00** est **/dev/sda1**. Dès lors, nous pouvons utiliser **/dev/disk/by-uuid/ee290474-ffbe-4f5d-a4e0-06bf55c96f00** comme s'il s'agissait de **/dev/sda1**, avec une différence majeure : l'UUID est fixe et indépendant de l'ordre de connexion des disques ou clés USB.

Nous créons alors un répertoire **/mnt/MIRROR** avec :

```
$ sudo mkdir /mnt/MIRROR
```

Notre objectif est de rentre le contenu de la clé (ou du disque) accessible par ce répertoire. Nous pourrions utiliser les commandes **mount** et **umount** en spécifiant tous les paramètres, mais il y a plus simple. Nous éditons le fichier **/etc/fstab** pour ajouter une ligne (avec **sudo nano** par exemple) :

```
/dev/disk/by-uuid/ee290474-ffbe-4f5d-a4e0-06bf55c96f00    /mnt/MIRROR
auto users,noauto 0 0
```

Ceci définit que le répertoire **/mnt/MIRROR** est lié à **/dev/disk/by-uuid/ee290474-ffbe-4f5d-a4e0-06bf55c96f00**, que le système de fichiers est auto-détecté, que le montage n'est pas automatique, mais qu'il peut être déclenché par un utilisateur normal (non **root**) et qu'il n'est pas nécessaire de vérifier ce disque au démarrage. Ceci fait, tout ce que nous avons à faire pour accéder au disque est de le connecter et d'utiliser **mount /mnt/MIRROR**. Le contenu apparaîtra alors dans ce répertoire. Lorsque nous aurons fini de l'utiliser, il suffira de faire **umount /mnt/MIRROR** et de le débrancher. Si nous branchons une autre clé USB, celle-ci aura donc un UUID différent et cela ne marchera pas. Notre répertoire est désormais intimement lié à cette clé et à elle seule.

4.2 Configuration du système de gestion de paquets

Par défaut, comme nous l'avons vu en début d'article, le système va chercher les paquets et les informations les concernant sur le net. Nous ne voulons plus qu'il adopte un tel comportement. La première chose à faire est donc de « désactiver » cela en éditant les fichiers **/etc/apt/sources.list** et **/etc/apt/sources.list.d/raspi.list** et en plaçant un **#** en début des lignes qui n'en ont pas. Ceci a pour effet de mettre ces lignes en commentaire et donc de faire en sorte qu'elles ne soient plus utilisées.

À ce stade, un **sudo apt-get update** ne fera plus rien du tout. L'étape suivante consistera donc à ajouter une autre source pour obtenir les paquets et plus précisément la clé ou le disque USB que nous avons préparé. Pour cela, nous allons ajouter un fichier **/etc/apt/sources.list.d/mirror.list** contenant deux lignes :

```
deb file:/mnt/MIRROR/MIRROR/mirror/mirrordirector.raspbian.org/
raspbian jessie main contrib non-free rpi
deb file:/mnt/MIRROR/MIRROR/mirror/archive.raspberrypi.org/debian
jessie main ui
```

Notez que **deb http:** est remplacé ici par **deb file:**, nous ne référençons pas des sources du web (HTTP), mais des fichiers locaux. Autre point important, les chemins spécifiés ne sont pas exactement identiques à ceux de la configuration de départ. Il s'agit des répertoires contenant les sous-répertoires **dists/** et **pool/** de chaque miroir. Le système de gestion de paquets utilise ces deux répertoires pour fonctionner et il lui faut l'emplacement précis où il peut les trouver. Le reste de la ligne est identique à la configuration par défaut avec **jessie** qui précise la « suite » à utiliser, puis les différents composants qui nous intéressent, à commencer par **main**, la base de la distribution. Ces éléments ayant été utilisés pour créer le miroir à partir des sources par défaut, nous utilisons simplement les mêmes ici.

4.3 Utilisation

Tout est prêt, il ne nous reste plus qu'à faire un essai. Nous commençons donc par connecter et « monter » la clé USB :

```
$ mount /mnt/MIRROR
```

Nos dépôts locaux étant à présent accessibles par le système, nous pouvons mettre à jour la liste des paquets :

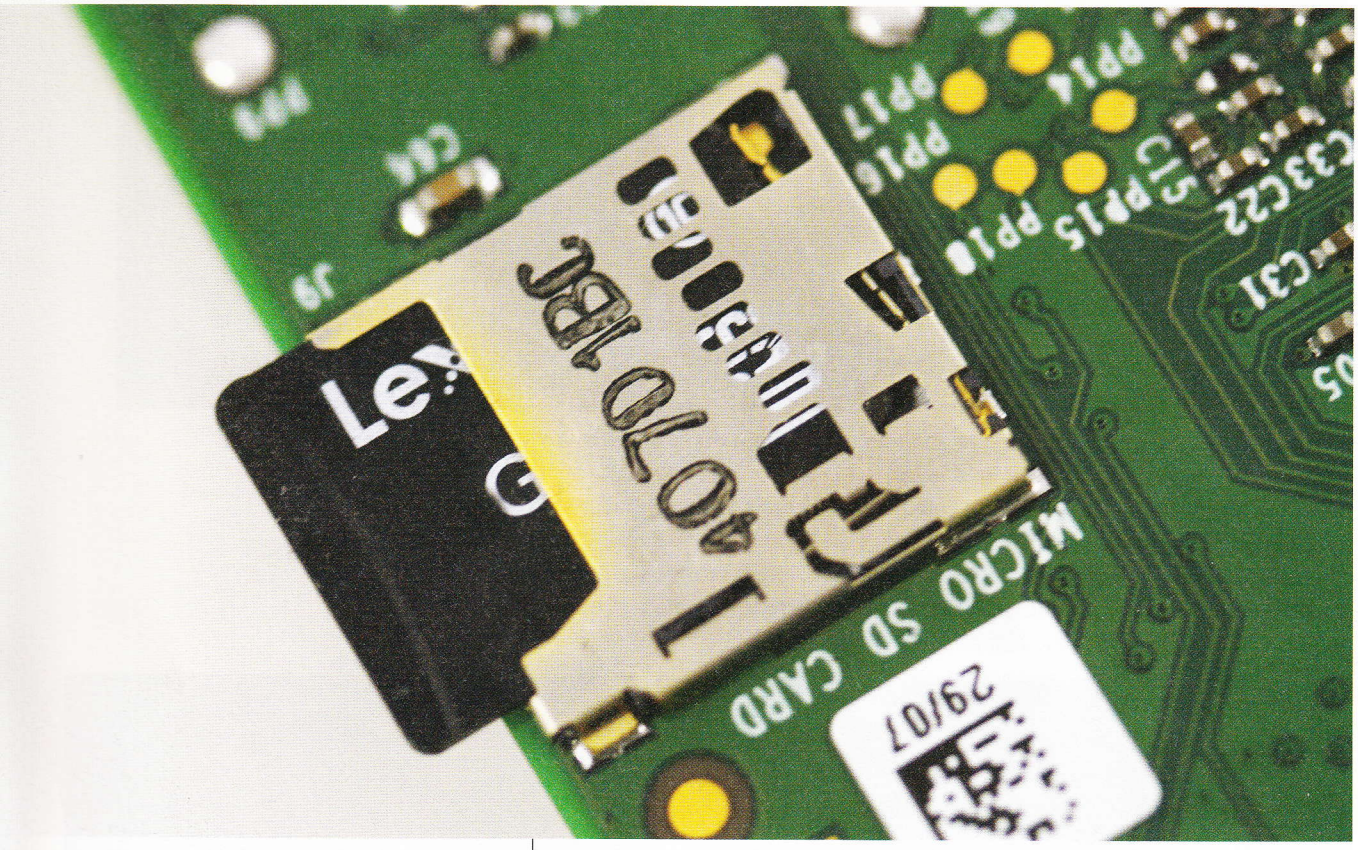
```
$ sudo apt-get update
Réception de : 1 file: jessie InRelease [13,3 kB]
Réception de : 2 file: jessie InRelease [15,0 kB]
Ign file: jessie/main Translation-fr_FR
Ign file: jessie/main Translation-fr
[...]
Ign file: jessie/rpi Translation-fr
Ign file: jessie/rpi Translation-en
Lecture des listes de paquets... Fait
```

À partir de ce moment, la liste des paquets est intégrée au système et à jour. Pour rechercher des paquets, par exemple, avec `apt-cache search` nous n'avons plus besoin de la clé. Ceci est également valable pour obtenir des informations (`apt-cache show`) ou supprimer des paquets avec `apt-get remove` (à moins que la suppression n'implique l'installation d'autres paquets). Pour installer ou mettre à jour en revanche, la clé USB sera nécessaire. Nous pouvons d'ailleurs faire un simple test en installant un paquet quelconque (ici la calculatrice GNU `bc`) :

```
$ sudo apt-get install bc
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les NOUVEAUX paquets suivants seront installés :
  bc
0 mis à jour, 1 nouvellement installés, 0 à enlever et 16 non mis à jour.
Il est nécessaire de prendre 0 o/96,3 ko dans les archives.
Après cette opération, 195 ko d'espace disque supplémentaires seront utilisés.
Sélection du paquet bc précédemment désélectionné.
(Lecture de la base de données... 30578 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de .../b/bc/bc_1.06.95-9_armhf.deb ...
Dépaquetage de bc (1.06.95-9) ...
Traitement des actions différées (" triggers ") pour man-db (2.7.0.2-5) ...
Traitement des actions différées (" triggers ") pour install-info (5.2.0.dfsg.1-6) ...
Paramétrage de bc (1.06.95-9) ...
```

Bien entendu, cela fonctionne parfaitement, sans connexion et très rapidement. Si la clé n'est pas accessible, cela ne casse rien, mais nous obtenons simplement un message d'erreur nous précisant que le paquet est introuvable. Cela fonctionnera tout aussi bien avec une mise à jour avec `sudo apt-get upgrade` et `sudo apt-get dist-upgrade`. Une fois nos opérations d'installation et de maintenance terminées, nous pouvons démonter la clé et la retirer :

```
$ umount /mnt/MIRROR
```



Une solution alternative à la clé USB consiste à utiliser directement la carte micro SD. Il faut bien entendu opter pour une capacité adéquate (128 Go) et, de préférence, réserver une partition pour le stockage des paquets. Plus compacte et libérant le port USB pour un autre usage, cette technique oblige cependant l'arrêt de la Raspberry Pi pour la mise à jour du miroir.

5. POUR FINIR

Il existe d'autres façons de faire, aussi bien pour la création du contenu de la clé que pour la configuration du système, mais celle-ci est, je trouve, la plus simple et la plus pédagogique. La première plateforme qui vient en tête pour une telle utilisation est bien entendu la toute récente Raspberry Pi Zero ou encore la Raspberry Pi A+. Ces deux cartes ne disposent pas de connecteur réseau Ethernet et selon l'utilisation visée n'ont pas nécessairement besoin d'être équipées d'un adaptateur Wifi. On peut donc envisager une configuration vraiment minimale avec une console série pour les interactions avec le système et notre clé USB pour les mises à jour.

En dehors de la gestion de Raspberry Pi in situ, cette technique peut également être intéressante pour une utilisation non sédentaire. Il suffit de garder son miroir à jour et il devient possible de tester et d'explorer Raspbian où qu'on se trouve... comme par exemple dans un TGV, en pleine nature, à la plage, au restaurant... Notez cependant le risque de probable perturbation de vos relations sociales, peu de gens comprenant l'intérêt de jouer avec une Raspberry Pi à table, en particulier si il ou elle essaie d'avoir une conversation avec vous. DB


Maker Faire® Paris
Présenté par Leroy Merlin
30 avril & 1er mai 2016
Paris expo porte de Versailles - Foire de Paris
APPEL AUX MAKERS !
Nous recherchons
Bricoleurs ★ Ingénieurs
Amateurs ⚓ Hackers ⚓ Geeks
Chercheurs ★ Inventeurs
Entrepreneurs ★ Rêveurs
Inscription sur
makerfaireparis.com

INNOROBO PARIS FRANCE

24-26 MAI 2016

co-produit

INSIDE 3D PRINTING™ CONFERENCE & EXPO

SMART CITIES

MEDICAL & HEALTH

SMART HOMES

FACTORIES OF THE FUTURE

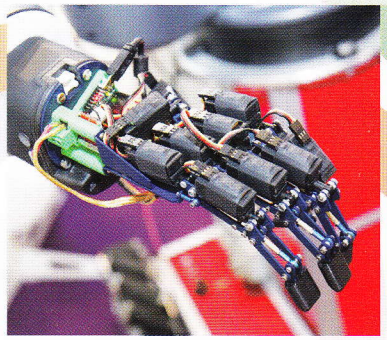
SERVICE TO INDUSTRIES

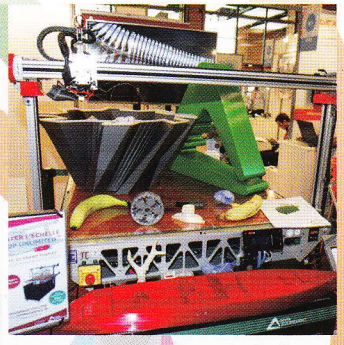
FIELD ROBOTICS

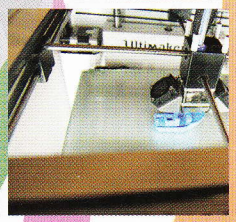
TECHNOLOGIES & FORESIGHT

Passionnés d'électronique, de mécanique et de nouvelles technologies ?

De l'impression 3D à l'intelligence artificielle, les technologies emblématiques font leur show, **ne les manquez pas !**







Ouverture aux professionels les mardi et jeudi

Ouverture au grand public et aux étudiants le mercredi toute la journée

YouTube theinnorobo

Pour la première fois INNOROBO est à PARIS. Toutes les info sur :

www.innorobo.com

risingmedia

PRODUIT PAR INNOECHO