N°29 | AVRIL - MAI - JUIN 2019

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

ESP8266 / WEB

Extrayez n'importe quelle donnée du Web pour l'afficher avec des leds

p. 14

ROBOT / PI

Contrôlez vos modèles Lego Mindstorms au joypad à l'aide de BrickPi

p. 92

3D / CAD

Débutez en modélisation 3D avec OpenSCAD

p. 52

C / FPGA

Créez et pilotez vos périphériques matériel en C sur carte ZedBoard

p. 66

Wifi / Domotique / Capteur :

Créez une sonde de température autonome !

p. 38

- Elle procède aux mesures
- Elle stocke les informations
- Elle génère le graphique web

3D / LASER

Transformez votre imprimante 3D en machine de gravure laser

p. 04

RETRO / 8 BITS

Restauration, réparation et modification d'un ordinateur ZX Spectrum

p. 84

L 19338 - 29 - F: 7,90 € - RD

digital.security

COnnected DEvices EXploitation

Formations de hacking hardware et software des objets connectés

CODEX01

4 JOURS
FORMATION EN ANGLAIS

- S'interfacer et communiquer avec un objet connecté
- Déterminer la surface d'attaque d'un objet connecté
- Extraire les micro-logiciels de différentes façons
- Analyser les micro-logiciels
- Identifier des vulnérabilités dans un micro-logiciel et les exploiter
- Conserver un accès résistant aux mises à jour de l'objet compromis

CODEX02 AVANCÉ

5 JOURS
FORMATION EN ANGLAIS

- Contourner les protections contre l'extraction de micro-logiciel
- Obtenir un accès privilégié au système d'un objet connecté
- Analyser un micro-logiciel d'un système ne reposant pas sur un système d'exploitation
- Identifier et exploiter des vulnérabilités applicatives sur architecture ARM
- Réaliser des attaques par canaux auxiliaires afin de contourner des restrictions
- Identifier, analyser et exploiter de multiples protocoles de communications

Les formations seront dispensées par **Damien Cauquil** (**@virtualabs**), expert sécurité spécialisé dans la sécurité des objets connectés depuis plus de 5 ans. Il est par ailleurs l'auteur de plusieurs outils de référence publiés lors de conférences comme DEF CON ou le Chaos Communicaton Congress et a trouvé et communiqué diverses vulnérabilités dans des objets connectés ainsi que dans plusieurs protocoles de communication.

- 10 % pour les 10 premiers inscrits avec le code 7mq9s3

Inscriptions par mail à formations@digital.security

ÉDITO

Le changement est pour bientôt !

Le mot « changement » n'est pas le plus juste car, en vérité, il faut surtout parler d'améliorations et d'ajouts. Je parle, bien entendu, du magazine que vous tenez entre vos mains ou, plus justement, du prochain que vous tiendrez entre vos mains. Enrichie de plusieurs dizaines de pages, la nouvelle formule introduite avec le futur numéro 30 gagnera en diversité en couvrant de nouveaux sujets de façon récurrente, comme la robotique, la sécurité et l'embarqué industriel.

Bien entendu, tout ce qui fait d'*Hackable* ce qu'il est actuellement, en particulier les aspects didactiques et surtout pratiques, restera d'actualité avec, toujours, des réalisations concrètes et un contenu basé essentiellement sur l'expérience et l'expérimentation.

Vous pouvez voir la présente édition comme un numéro de transition puisque vous y trouverez non seulement un article orienté robotique, mais également un papier de P. Kadionik à propos de quelque chose dont nous n'avons encore jamais parlé dans *Hackable* : les FPGA. Voyez cela comme un avant-goût de ce qui occupera un espace dédié supplémentaire dès le prochain numéro.

D'autres évolutions, améliorations, ajouts et surprises vous attendent pour le numéro 30 que vous pourrez découvrir dès le 28 juin chez votre marchand de journaux. D'ici là, je vous laisse découvrir et explorer les sujets du présent numéro qui, je l'espère malgré votre impatience, n'en sera pas moins utile et intéressant.

Enfin, je profite du peu d'espace qui me reste ici pour vous signaler que *Hackable* est désormais également lisible via une application Android et iOS gratuite vous permettant d'acheter les numéros de votre choix et de vous abonner.

Bonne lecture à vous.

D. Denis Bodor

SOMMAIRE

ENCART CONNECT ENCARTÉ DANS LA COUVERTURE

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 : https://www.hackable.fr/ - https://www.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Kathrin Scali
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Abomarque : 09 53 15 21 77

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : 2427-4631
Commission partaire : K92470
Périodicité : trimestriel
Prix de vente : 7,90 €

MIXTE
Papier issu de sources responsables
FSC® C015136

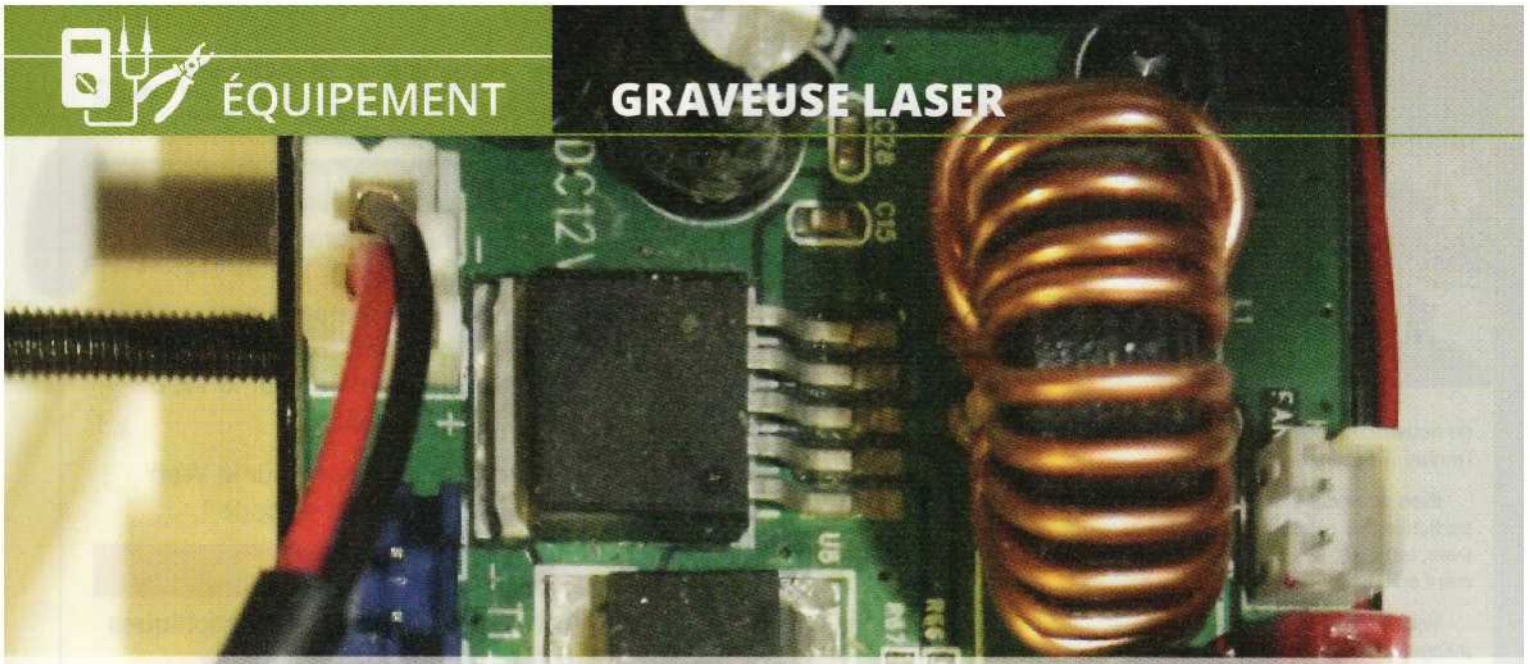
Suivez-nous sur Twitter
@hackablemag

Alsace

À PROPOS DE HACKABLE...

HACKS, HACKERS & HACKABLE

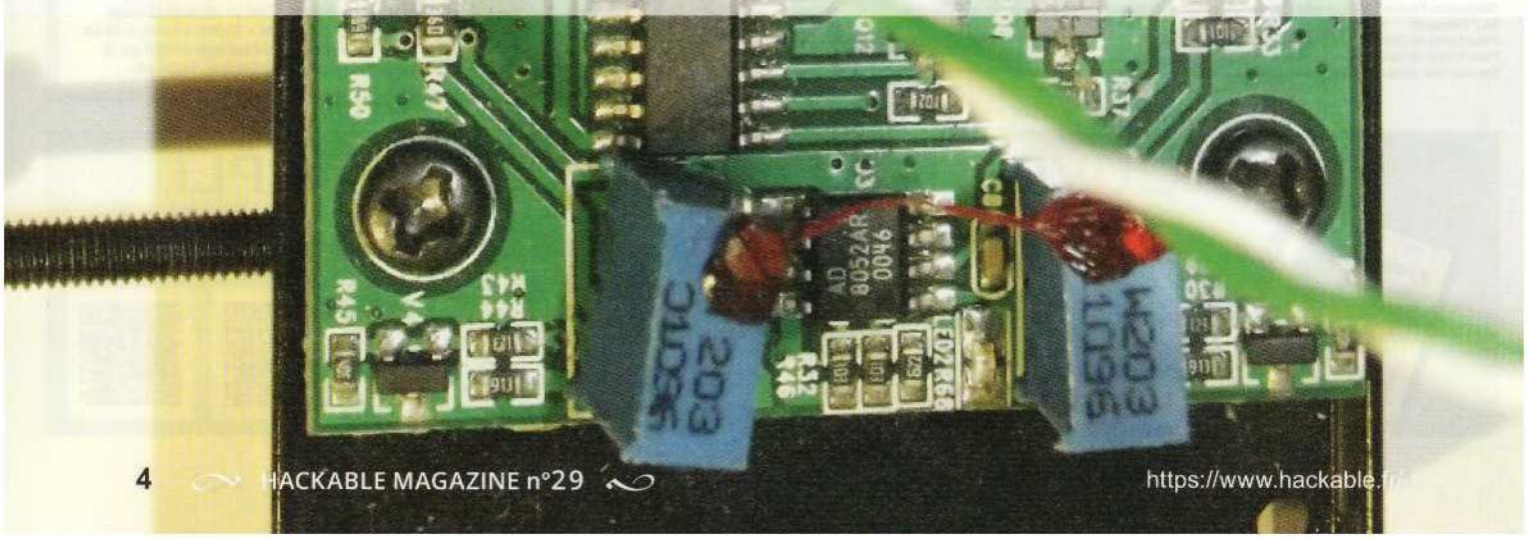
Ce magazine ne traite pas de piratage. Un **hack** est une solution rapide et bricolée pour régler un problème, tantôt élégante, tantôt brouillonne, mais systématiquement créative. Les personnes utilisant ce type de techniques sont appelées **hackers**, quel que soit le domaine technologique. C'est un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



AJOUTER LA FONCTION GRAVEUSE LASER À VOTRE IMPRIMANTE 3D

Jannick Paris

Ça faisait déjà un petit moment que ça me travaillait. J'avais vu de petites graveuses laser pas trop chères (environ 100 €) et je voulais m'en acheter une. Mais ça aurait fait une machine en plus, des câbles en plus et surtout de la place en moins sur mon bureau... Je me devais de trouver une solution.



En y regardant de plus près, on constate que les petites graveuses laser possèdent une mécanique très proche de celle des imprimantes 3D. En fait, une imprimante 3D possède un axe de plus (axe Z), je me suis alors dit : « qui peut le plus peut le moins ». J'ai donc décidé d'ajouter une diode laser sur mon imprimante. Je parle bien d'ajouter, hors de question de perdre la capacité à imprimer des objets. Voyons ensemble toutes les étapes de cette métamorphose.

1. AVERTISSEMENT

Avant toute chose, on va parler sécurité ! On a tous envie de devenir des maîtres Jedi (NDLR : ou Sith) et de s'amuser avec des lasers. Mais il faut garder à l'esprit que ces sources lumineuses sont dangereuses ! Dans mon cas, il s'agira d'un laser d'une puissance de 3W. On peut se dire que ce n'est pas très puissant et que la plupart des éclairages LED que l'on peut trouver en magasin ont des puissances similaires, voire supérieures. Cependant, il ne faut pas juste regarder la puissance,

ce qui va conditionner la dangerosité d'une source lumineuse est sa puissance surfacique. Elle se mesure en W/m². Une lampe LED pour extérieur peut atteindre 20W, mais elle éclaire une surface de plusieurs m², le laser quant à lui ne fait que 3W, mais le point qu'il éclaire fait moins de 1mm².

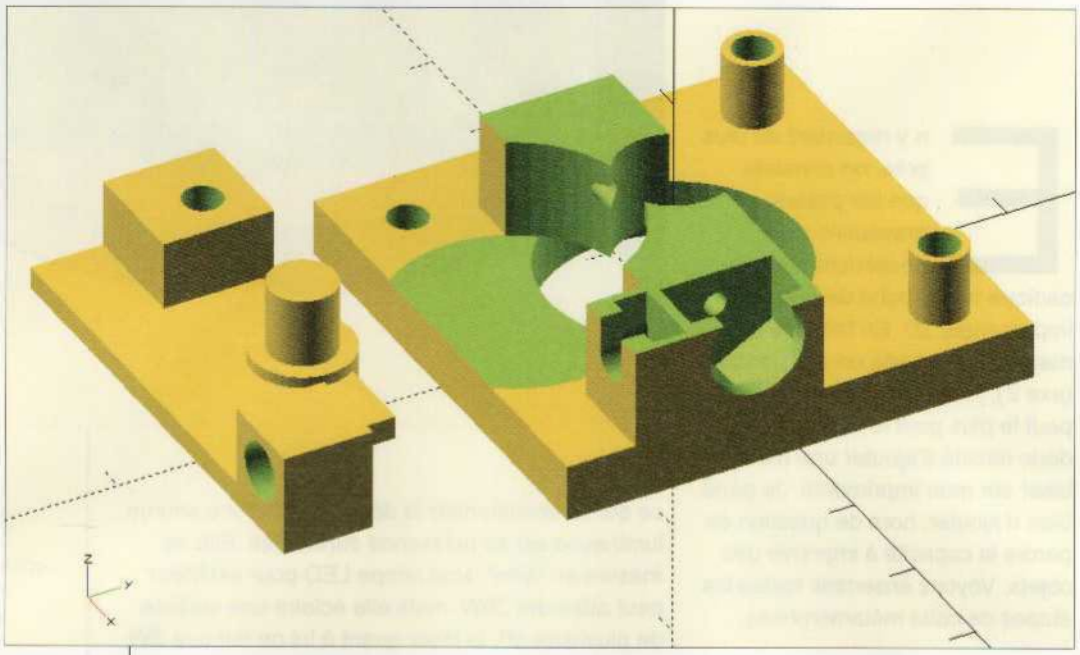
Comparons ça au Soleil. Notre planète se trouve à quelque 150 millions de kilomètres du Soleil, à cette distance la puissance surfacique reçue par la Terre est de l'ordre de 1000W/m² (grosso modo). Avec une telle puissance, des dégâts irréversibles peuvent être causés. Prenons maintenant un laser de 3W sur une surface de 1mm², sachant qu'il y a 1 million de mm² dans un m², on se retrouve face à une puissance surfacique de 3 000 000 W/m² !! Cela signifie qu'au point focal du laser, la lumière sera 3 000 fois plus intense qu'en plein jour ! Je pense que vous comprenez pourquoi il ne faut pas y mettre ni les doigts ni ses yeux !

Les reflets étant également très intenses, pensez à vous équiper de lunettes de protection adaptées à la longueur d'onde de votre laser. J'insiste sur le fait qu'elles doivent être adaptées à la longueur d'onde ! Des lunettes bleues pour un laser bleu ne vous protégeront pas !

2. PROBLÈMES À RÉSOUDRE

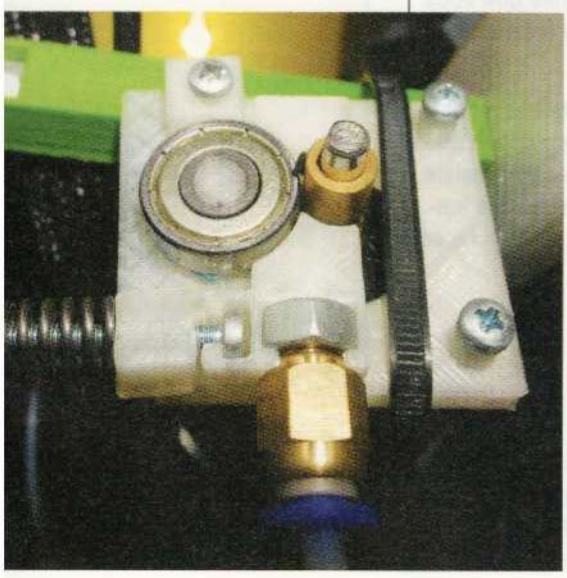
Avant de se lancer tête baissée dans la modification d'une imprimante 3D, il faut bien réfléchir au résultat qu'on souhaite obtenir. Pour ma part, je ne voulais pas transformer mon imprimante en

Le laser que j'utilise est d'une longueur d'onde de 450nm, ce qui correspond à la couleur bleu, les lunettes de protection quant à elles sont de couleur rouge. Vous pouvez vous amuser à regarder toutes les couleurs qui ne contiennent pas de bleu dans votre éditeur graphique préféré. Cela va du vert jusqu'au rouge, en passant par le jaune et l'orange. Le papier millimétré servira à un petit test plus tard.



Les pièces de remplacement : à gauche, la partie qui va permettre de plaquer le roulement contre le filament et à droite, la partie fixée au moteur.

Une fois imprimé et assemblé, voilà le résultat. Le filament entre par l'arrière, passe entre le roulement et la roue d'entraînement pour être poussé dans le raccord pneumatique, dans lequel est inséré un tube en PTFE (polytétrafluoroéthylène, souvent vendu sous les marques déposées : Téflon).



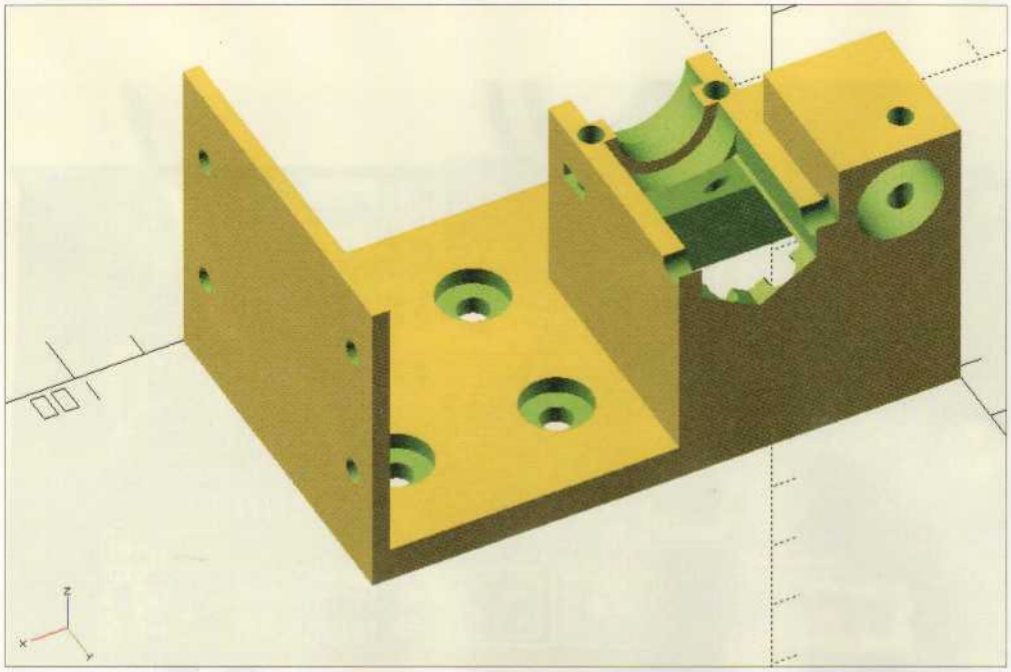
graveuse laser, je voulais lui ajouter cette fonctionnalité. Il faut donc trouver une solution, car encombrer et rajouter du poids sur le chariot X n'est clairement pas une bonne idée.

2.1 Déporter le moteur d'extrusion

Cette partie n'est pas bien compliquée, cependant il est nécessaire de la réaliser avec le plus grand soin. Vu qu'il s'agit de remplacer le système d'extrusion, une erreur sur les nouvelles pièces vous forcera à remonter les anciennes, recalibrer votre imprimante et recommencer... Pour ma part, j'ai choisi la sécurité en me procurant un moteur pas-à-pas identique pour faire mes tests.

De nombreuses imprimantes 3D possèdent leur moteur dédié à l'extrusion monté sur le chariot X, c'est une bonne solution, simple et efficace. De cette façon, il se trouve très près de la tête d'extrusion et peut y pousser le filament directement. Le point négatif, c'est qu'il prend de la place... Place que j'aimerais donner à ma diode laser. Il faut donc chercher une méthode pour pouvoir le déplacer loin de la tête d'extrusion, tout en lui permettant de continuer à pousser le filament à l'intérieur. Même si la solution existe déjà, vous allez voir que ce n'est pas une mince affaire.

Il faut dans un premier temps modéliser une pièce en 3D qui permettra au moteur d'entraîner le filament. De nombreuses versions de ce type de pièce existent sur le site thingiverse.com. Cependant, je n'ai pas trouvé celle qui me convenait, car pour toutes celles que j'ai pu y voir, il fallait visser le raccord pneumatique (je vais détailler ça un peu plus loin) directement dans le plastique. Ce qui a tendance à provoquer un délaminage, voire carrément à casser la pièce. J'ai donc dessiné une pièce dans laquelle on peut insérer un écrou M6, qui va permettre de maintenir correctement ce raccord.



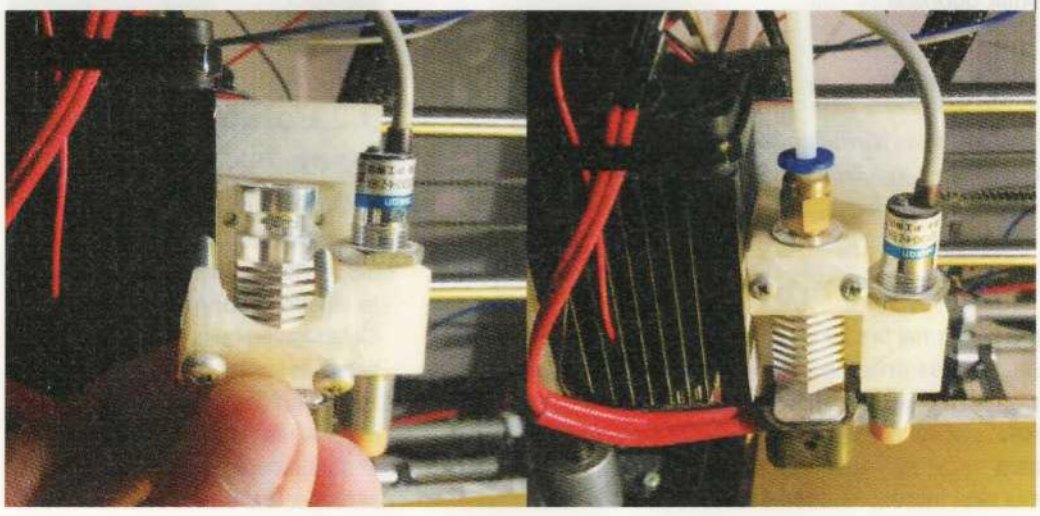
De gauche à droite, on peut voir l'emplacement pour la diode laser, l'empreinte de l'extrudeur hexagon et un trou cylindrique afin d'y glisser le capteur inductif.

En choisissant le bon diamètre pour le raccord et le tube de PTFE, le filament rentre tout juste et glisse à l'intérieur, sans se plier. On peut donc le guider où bon nous semble.

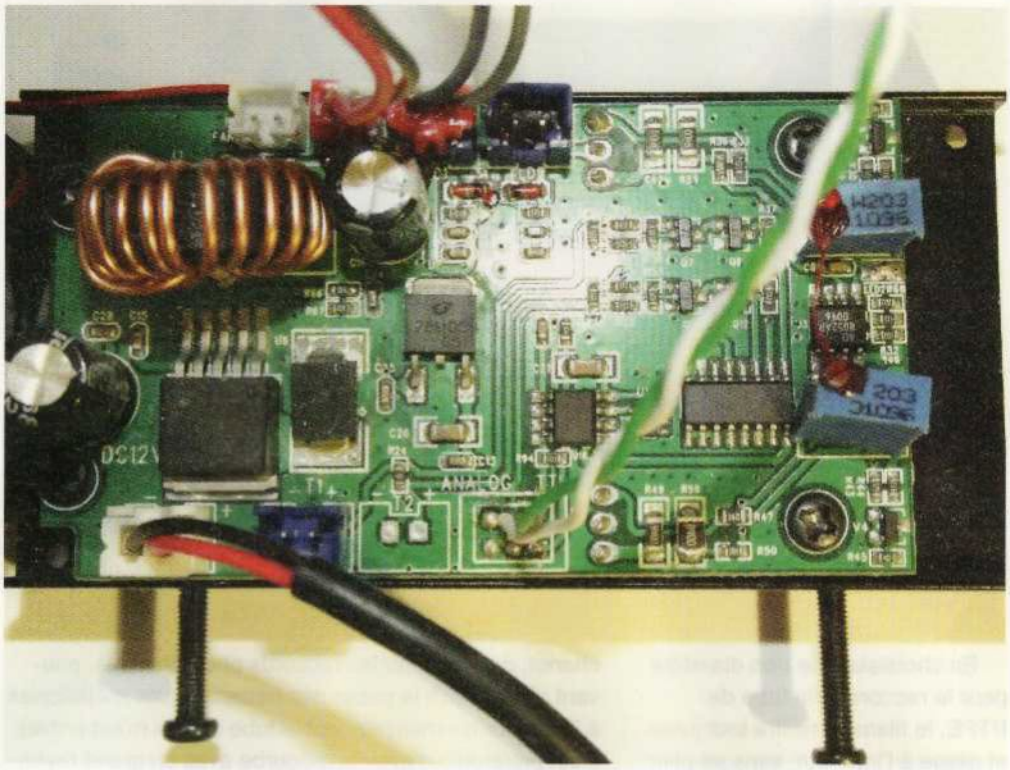
Un point important de cette modification est de laisser au tube de PTFE assez de marge de manœuvre. N'essayez pas d'aller au plus court. Cela aura pour conséquences, lors des mouvements du chariot, de forcer sur les raccords pneumatiques, pouvant aller jusqu'à la casse des petites parties métalliques à l'intérieur qui maintiennent le tube (ce qui m'est arrivé). Il est préférable d'avoir une courbe avec un grand rayon de courbure, ça rendra l'ensemble plus flexible.

2.2 Modélisation d'un nouveau support

Maintenant, il faut modéliser le support qui va tenir la tête d'extrusion, le capteur inductif et la diode laser sur le chariot.



Un petit clip à visser sur l'extrudeur et voilà le résultat final.



Voici l'alimentation fournie avec ma diode laser. Au centre en bas, on voit la notation « ANALOG TTL ». Après une courte investigation, j'ai trouvé la broche à relier à la masse pour allumer le laser.

En utilisant openSCAD, le plus simple est de modéliser chacune des pièces, pour ensuite les soustraire à un bloc. On fera donc apparaître une empreinte de chaque pièce.

Ici par contre, pas moyen de faire un test en dehors de l'imprimante comme pour le paragraphe précédent. Il faut monter la pièce sur le chariot de l'axe X pour essayer. Vérifiez donc à deux fois les cotes de la pièce, avant de démonter quoi que ce soit.

3. PILOTER LE LASER

C'est bien beau d'avoir un laser, mais encore faut-il l'allumer et l'éteindre au bon moment ! Il faut donc trouver une broche de libre sur la carte Ramps, pour agir sur l'alimentation du laser.

3.1 Choix de la broche

Après avoir fait le tour de mon imprimante et cherché les schémas (**https://reprap.org/wiki/File:Arduinomega1-4connectors.png**), j'ai fini par trouver LA réponse ! La broche inutilisée, qui me permettra de piloter le laser et de graver tout ce que je veux, se trouve être... 42 ! Heureusement que je n'ai pas mis sept millions et demi d'années pour trouver.

Pour changer l'état d'une broche avec le firmware Marlin, il faut utiliser le GCode: M42 ! Pour mettre cette broche à l'état 0 ou à l'état 1, voilà les commandes complètes :

```
M42 P42 S0
M42 P42 S255
```

3.2 L'alimentation du laser

Une diode laser est un composant sensible. Tout comme une diode classique, sa courbe caractéristique est une exponentielle. Cela signifie qu'une fois qu'elle sera passante, une variation de la tension d'alimentation, même minime, engendrera une énorme variation de courant traversant la diode. Or, les diodes laser ne

supportent pas les surintensités, même de manière très brève. C'est pour cela qu'il est préférable de choisir une diode laser accompagnée de son alimentation, en faisant attention à la présence de broches TTL. Car ce sont ces broches qui vont nous permettre d'agir sur l'état du laser.

Tout est fonctionnel, cependant trois petits détails m'embêtent. La broche (ANALOG TTL) qui pilote le laser doit être reliée à la masse. Cela signifie que, lorsque je démarrerai mon imprimante, la broche 42 sera dans l'état 0 et le laser sera donc allumé. Deuxième chose, je n'ai aucune idée du courant que va devoir drainer la broche 42. Enfin, si le laser est éteint, comment savoir si mon objet à graver est bien placé ?

Je pourrais prendre chacun des problèmes séparément et trouver à chacun une solution, mais pourquoi faire trois choses alors qu'une seule est suffisante ? Avec deux résistances et un transistor, on peut réaliser un montage en émetteur commun (voir schéma 1).

Ce petit circuit règle tous les problèmes :

- quand l'imprimante démarre, la broche 42 est dans l'état 0, le transistor est donc bloqué. État 0 laser éteint, état 1 laser allumé. C'est quand même plus logique ;
- c'est le transistor qui draine le courant ;
- la résistance de 150kΩ en parallèle du transistor est là pour laisser passer un très faible courant. Vu qu'il s'agit d'une commande analogique, le laser s'allumera très faiblement. Je pourrai donc positionner le faisceau par rapport à l'objet.

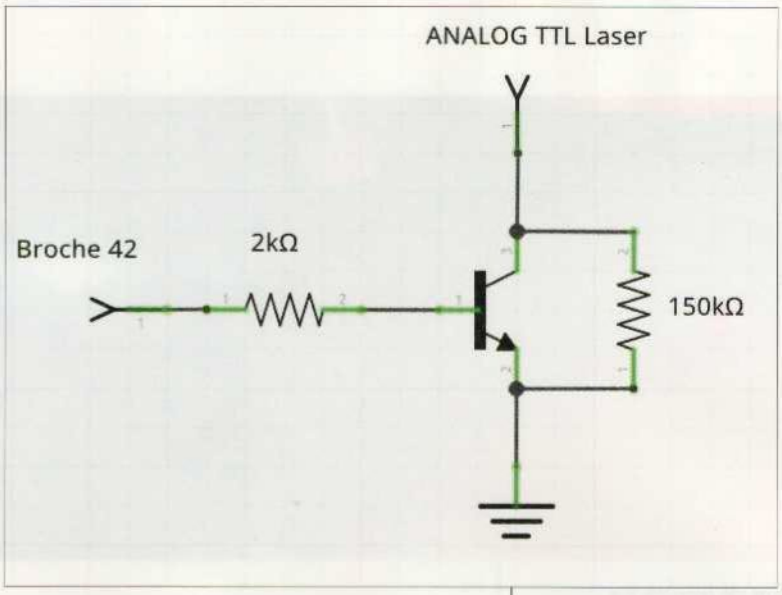
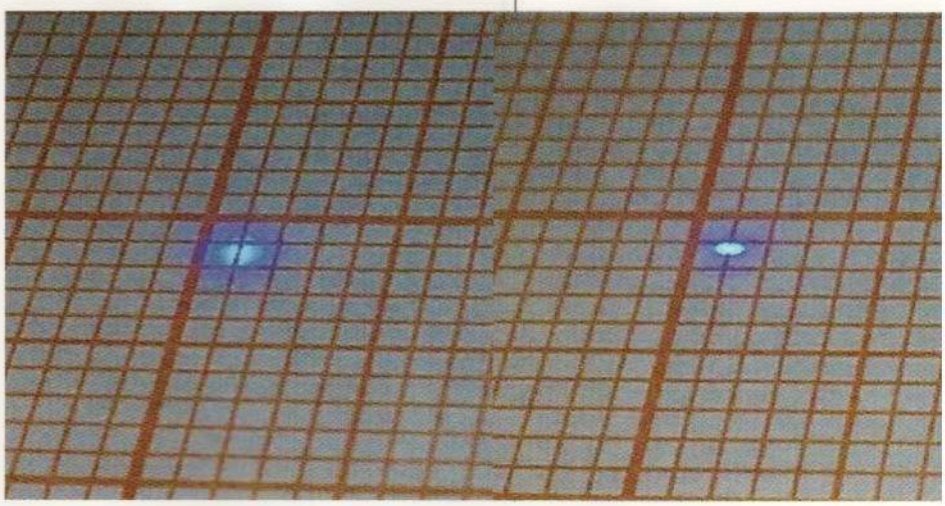


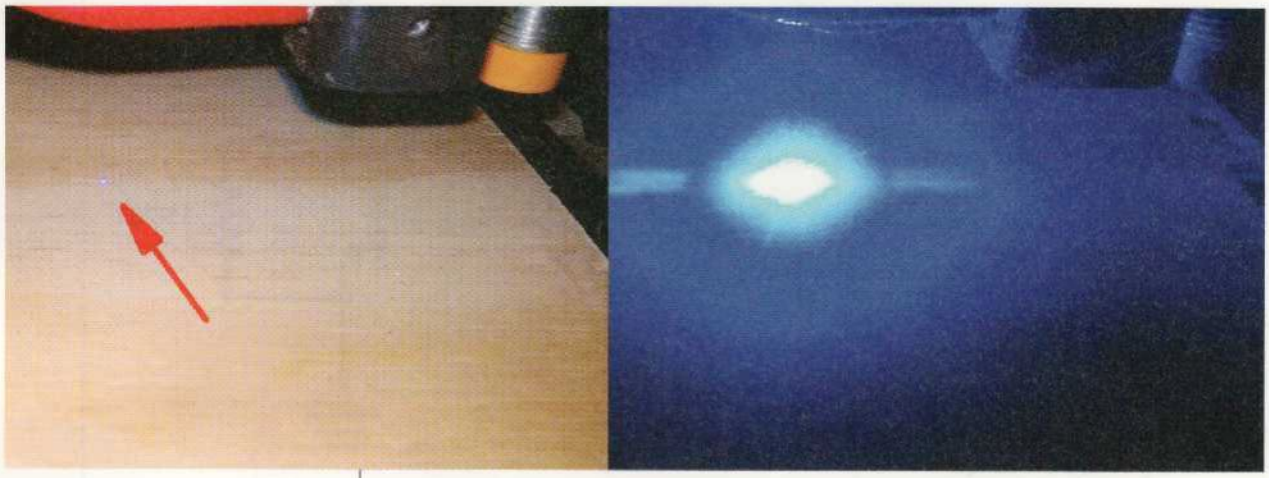
Schéma 1.

4. QUELQUES TESTS

Comme un enfant le soir de Noël, j'ai envie de tester mon nouveau jouet ! Mais sans pour autant négliger la sécurité ! Autant, quand on réalise un petit circuit et qu'on fait des soudures, on peut se brûler, ce n'est pas agréable, ça peut laisser des traces, mais rien de bien grave. Avec un laser, c'est une autre histoire ! La rétine peut être endommagée de manière irréversible ! N'oubliez pas



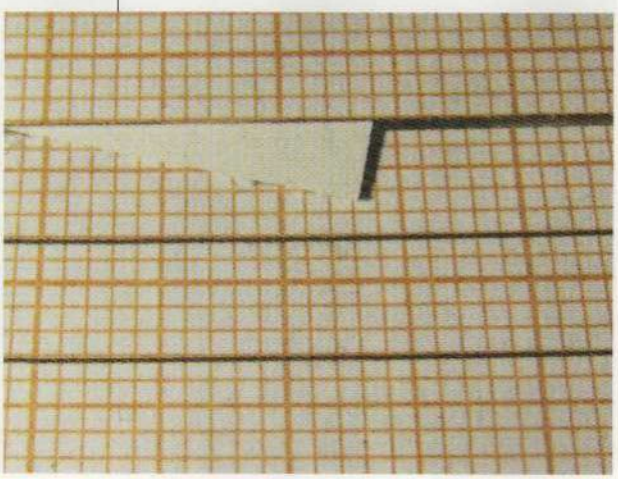
L'avantage d'utiliser une imprimante 3D est qu'on peut jouer sur l'axe Z, pour que l'objet à graver soit pile-poil dans le plan focal du laser. À gauche, le laser était un peu trop haut, on observe une tâche diffuse, à droite après être descendu de quelques millimètres, le point est net.



J'ai essayé de prendre les photos sans bouger, à gauche je vous ai placé une petite flèche, car on a du mal à distinguer le petit point bleu du laser (causé par la résistance de 150kΩ), à droite après la commande : M42 P42 S255. La différence de contraste sur les photos est assez trompeuse, mais sachez que même avec les lunettes de sécurité, ce n'est pas agréable de regarder vers la zone de gravure.

de porter des lunettes de protection adaptées ! Si vous vous lancez à l'aventure, pensez à établir un protocole en utilisant votre machine, histoire de prendre les bonnes habitudes. Pour ma part, je procède ainsi :

- préparation du Gcode ;
- branchement de l'imprimante ;
- positionnement de la pièce ;
- port des lunettes ;
- récupération du Gcode et envoi à l'imprimante ;
- gravure ;
- et seulement à la fin, quand l'imprimante est revenue à l'origine, je retire les lunettes et admire mon œuvre.



Avec un petit script bash et les bonnes commandes, je lui ai demandé de couper du papier millimétré. J'ai glissé une petite carte pour prouver que le papier a bien été coupé. On constate que la coupe est nette et bien inférieure au millimètre.

5. LES CHOSES SÉRIEUSES

Tout fonctionne comme il faut, mais je n'ai pas fait ça pour couper du papier en ligne droite. Le but est de pouvoir prendre une image et de la graver sur un objet. Or, une imprimante 3D ne prend pas en entrée une image. Il va donc falloir convertir l'image à graver en Gcode compréhensible par l'imprimante.

Je vais vous détailler deux manières de faire, mais dans les deux cas, la première étape sera la même, il faut vectoriser l'image que l'on souhaite graver. Le logiciel (libre et gratuit) Inkscape le fait très bien. Rien de compliqué là-dedans. En suivant le tuto : **https://inkscape.org/fr/doc/tutorials/tracing/tutorial-tracing.html**, on arrive au bon résultat sans problème. Une fois que l'image est vectorisée, on a deux possibilités :

- utiliser le plugin « J Tech Photonics Laser Tool » ;
- sauvegarder au format DXF.

Dans le premier cas, on va obtenir directement le Gcode qu'on va pouvoir envoyer à l'imprimante, mais petit bémol, avec des coordonnées absolues. Cela signifie qu'il faut que la position de l'objet à graver corresponde parfaitement avec la position de l'image.

Pour la deuxième, il va nous falloir un autre logiciel (lui aussi libre et gratuit), dxf2gcode. On peut le trouver ici : **https://sourceforge.net/projects/dxf2gcode/** et il porte bien son nom, puisqu'il va nous permettre de passer d'un fichier DXF à du Gcode. Il y a cependant quelques subtilités à régler, avant d'obtenir notre résultat.

Le plus simple est de modifier le fichier : `~/.config/dxf2gcode/postpro_config/postpro_config.cfg`.

```
code_begin_units_mm = G21 ;(Units in millimeters)
code_begin_units_in = G20 ;(Units in inches)
code_begin_prog_abs = G90 ;(Absolute programming)
code_begin_prog_inc = G91 ;(Incremental programming)
code_end = M2 ;(Program end)
comment = %nl(%comment)%nl
```

Ce genre de lignes pose problème, car le firmware Marlin ne considère pas ce qui est entre parenthèses comme étant un commentaire et l'interprétation du Gcode n'aboutira pas. Pour commenter correctement ces lignes, il faut ajouter un `;` :

```
code_begin_units_mm = G21 ;(Units in millimeters)
```

Ce logiciel a été conçu pour générer du Gcode pour des fraiseuses. Or, lorsqu'on travaille avec ce type de machine, un point important est de mettre en sécurité l'outil. Ça consiste simplement à le placer à une hauteur à laquelle on sait qu'il n'y aura aucun obstacle sur sa route. Un autre paramètre important est la profondeur de fraisage. On pourrait croire que cela ne va pas nous intéresser, vu qu'on travaille avec un laser. Mais on va détourner ces paramètres pour piloter le laser et l'allumer au bon moment. Car le Gcode généré par ce logiciel va suivre la structure :

- positionnement de l'outil au début du chemin ;
- descente de l'outil de la hauteur de sécurité ;
- descente de l'outil de la profondeur de fraisage ;
- déplacement le long du chemin ;
- remontée de la profondeur de fraisage ;
- remontée à la hauteur de sécurité.

On va donc détourner le Gcode utilisé pour les descentes et les montées pour actionner le laser. Ce qui va nous donner la structure suivante :

- positionnement du laser au début du chemin ;
- éteindre le laser ;
- allumer le laser ;
- déplacement le long du chemin ;
- allumer le laser ;
- éteindre le laser.

Découpe de tissu. Ça fonctionne très bien, il faut juste faire en sorte qu'il soit bien à plat. S'il y a des plis, il ne seront pas dans le plan focal et la coupe ne sera pas efficace.

Pour y arriver, il faut modifier les lignes :

```
rap_pos_depth = G0 Z%ZE %nl
lin_mov_depth = G1 Z%ZE%nl
```

par :

```
rap_pos_depth = M400 %nlM42 P42 S0%nl
lin_mov_depth = M42 P42 S255%nl
```

Vous remarquerez aussi la présence de la commande M400, qui est là pour spécifier qu'il faut attendre que tous les mouvements soient finis, avant de passer à la commande suivante. Ça nous assure d'être bien positionné avant d'allumer le laser.

On peut également modifier les lignes contenant les codes G1, G2 et G3 pour y ajouter le paramètre F800 (par exemple), ça permettra de définir la vitesse et sera très utile suivant les matériaux qu'on souhaite attaquer. Tous ces paramètres peuvent également être modifiés en mode graphique, dans le menu « Options » et « Postprocessor configuration... ».

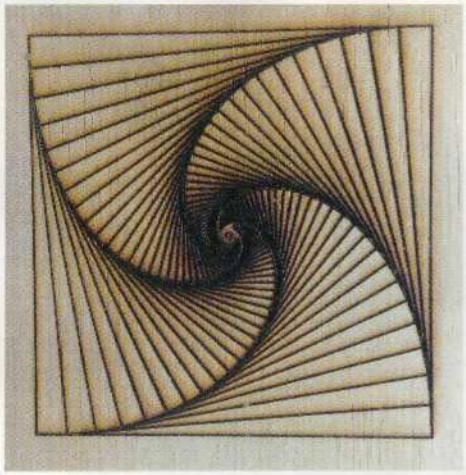
Une fois ceci fait, on va enfin pouvoir traiter notre fichier DXF. Pour ce faire, il suffit de lancer dxf2gcode et d'ouvrir notre image, dans le menu « File », puis « Open... ». Vous verrez alors votre image s'afficher. Bien que non nécessaire, vous pouvez passer par une étape d'optimisation du chemin. Histoire de parcourir le moins de distance et que ça soit plus rapide. Pour ce faire, il faut se rendre dans l'onglet « Layers », cocher tous les chemins ou utiliser le clic droit et choisir « Select all », puis refaire un clic droit et choisir « Optimize route for selection ». Une fois ceci fait, vous pourrez utiliser le menu export pour obtenir votre Gcode, avec « Export Shapes » ou « Optimize and Export Shapes ». On obtient finalement notre Gcode, mais un petit problème subsiste. Le fichier commencera par cette ligne :

```
(Generated with: DXF2GCODE, Version: Py3.6.7 PyQt5.10.1, Date:
$Date: Mon Sep 25 13:57:11 2017 +0200 $)
```

Or, ce n'est pas du Gcode et le firmware Marlin n'arrivera pas à l'interpréter. Il faut donc la commenter (ainsi que les trois autres qui suivent) avec un « ; », ou tout simplement les effacer, car elles ne sont pas nécessaires. J'ai beau chercher, je n'ai pas trouvé une méthode pour les commenter automatiquement... Il faudra donc répéter cette opération à chaque fois...

Si on noircit la surface du plexiglas, on peut alors l'attaquer au laser, mais sur une très faible profondeur.

Avec cette solution, il est possible d'utiliser des coordonnées relatives, cela signifie que l'origine de l'image (le point tout en bas à gauche) sera la position du laser à la lecture du Gcode. Il y a donc moins de contraintes sur le positionnement de l'objet à graver. Il suffit de le placer sur le plateau en faisant attention à ce qu'il soit aligné avec les axes (par exemple, en déplaçant le laser pour voir s'il suit bien



Mes meilleurs résultats ont été obtenus sur du bois, comme ici avec cet emboîtement de carrés.

Après les tests, voici une vraie réalisation. Une petite boîte à bijoux en bois que ma femme et moi avons customisée pour une amie.

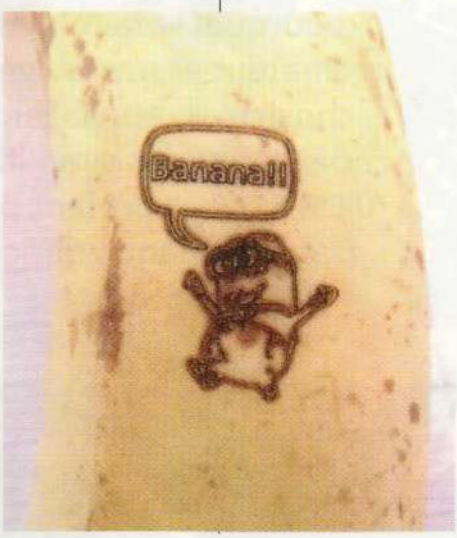
les contours de l'objet) et de positionner le laser là où vous voulez commencer la gravure. Je préfère cette approche, car elle nécessite moins de mesures et donc moins d'erreurs.

6. LES RÉSULTATS

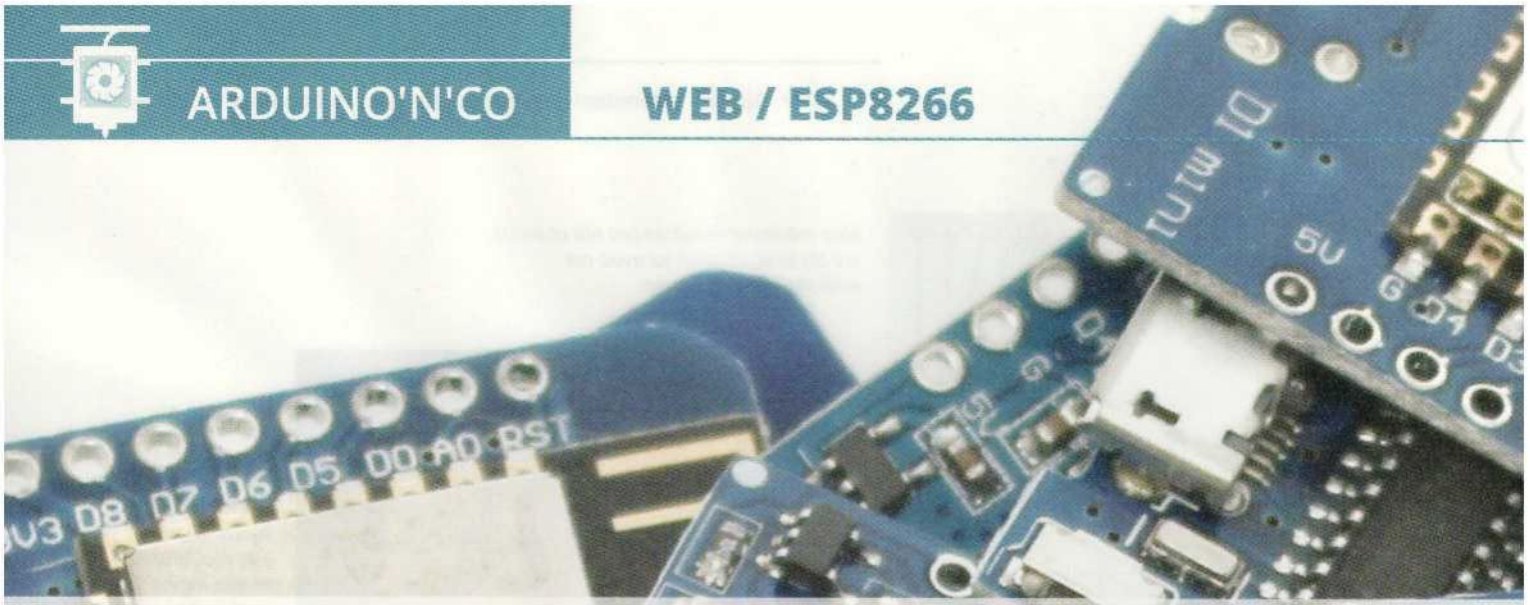
On peut enfin générer des chemins complexes et piloter le laser. Maintenant, on va pouvoir s'amuser ! Je ne vous cache pas que j'ai fait moult tests avant de trouver les bonnes vitesses pour les différents matériaux. Je ne vais pas vous faire une liste de mes paramètres, car cela va dépendre du résultat que vous souhaiterez obtenir. Il faut simplement retenir une chose, plus le laser exposera une zone longtemps, plus la chaleur aura le temps de se propager aux alentours et plutôt que d'avoir un trait fin et net, vous obtiendrez un trait large et flou. Cela signifie que si vous souhaitez entailler un objet en profondeur, il est préférable de faire plusieurs passes, plutôt que le laisser le laser sur un point jusqu'à ce qu'il traverse.

Un laser de 3W est déjà bien dangereux, mais ce n'est pas suffisant pour traverser une feuille d'aluminium ou les quelques microns d'épaisseur de cuivre d'un PCB. Mais les possibilités restent nombreuses, plutôt que de vous les lister, je vous laisse avec quelques photos de mes réalisations. **JP**

Voilà le côté absurde du hack, je suis son père !



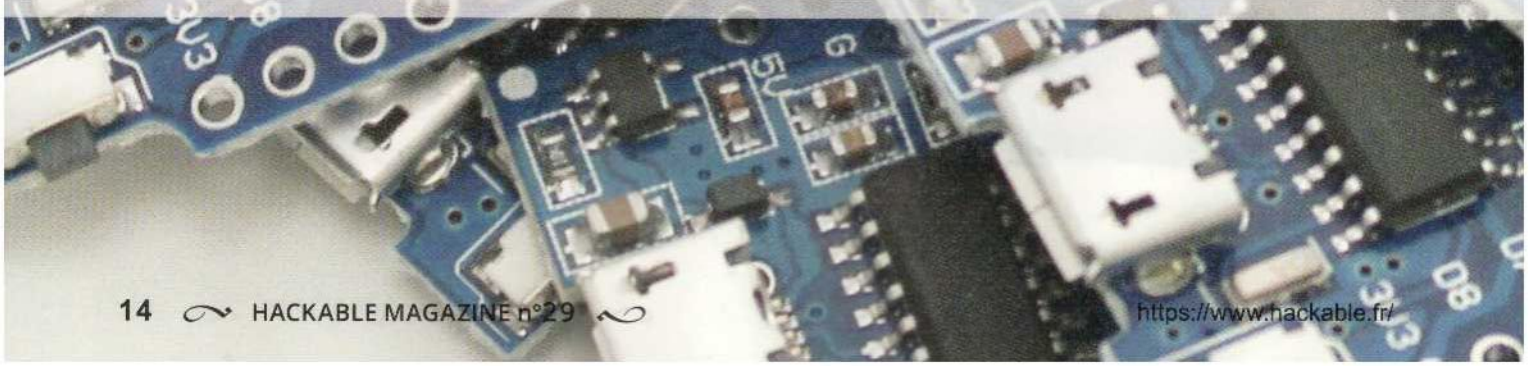
Sur les conseils de Denis, j'ai testé sur une banane parce que... pourquoi pas !



COLLECTEZ DES DONNÉES SUR LE WEB AVEC VOS ESP8266

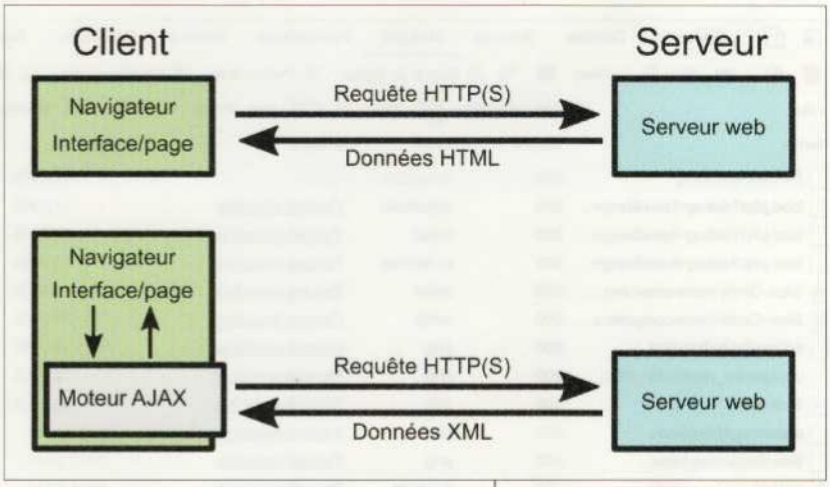
Denis Bodor

Accédez-vous régulièrement à une ou plusieurs pages web spécifiques pour vous informer sur un point précis, comme une valeur numérique, un état ou encore une progression quelconque ? Si tel est le cas, pourquoi le faire manuellement et de façon répétitive avec votre ordinateur et un navigateur web ? L'objet même de la programmation consiste à remplacer les tâches fastidieuses et récurrentes par du code, pour vous simplifier la vie. Les petites cartes connectées comme l'ESP8266 se prêtent parfaitement à ce type d'usage, à condition de franchir chaque difficulté une par une. Et c'est précisément ce que nous allons faire ici...



Automatiser la collecte d'information ou de données sur le web est quelque chose de très courant. Ceci au point que certains prestataires de services et en particulier, les gros acteurs du net, mettent à disposition des interfaces spécifiques à cet usage : des API web. En effet, on peut envisager ces services en ligne comme ayant deux visages : nous avons d'une part ce que vous, utilisateur, voyez et utilisez avec votre navigateur et d'autre part, ce qu'un programme peut utiliser pour accéder aux mêmes services et informations. Ainsi, si nous prenons le cas de Twitter par exemple, nous avons d'un côté le site web fait pour les humains (et assimilés) et de l'autre, une API (*Application Programming Interface* ou interface de programmation applicative) pour les programmes.

Cette seconde interface permet aux développeurs de plus facilement créer des programmes qui peuvent poster un tweet, suivre son audience, lire et traiter les tweets des personnes qu'on suit, etc. Ces API sont généralement très bien documentées, disposent d'exemples concrets d'utilisation (Python, Java, Node.JS, etc.) et nécessitent votre inscription comme développeur pour pouvoir les utiliser. Si l'on s'en tient à cela, tout va bien dans le meilleur des mondes, puisque tous les services en ligne très populaires (Google, Gmail, Twitter, eBay, Facebook, GitHub, etc.) proposent ce genre d'interface.

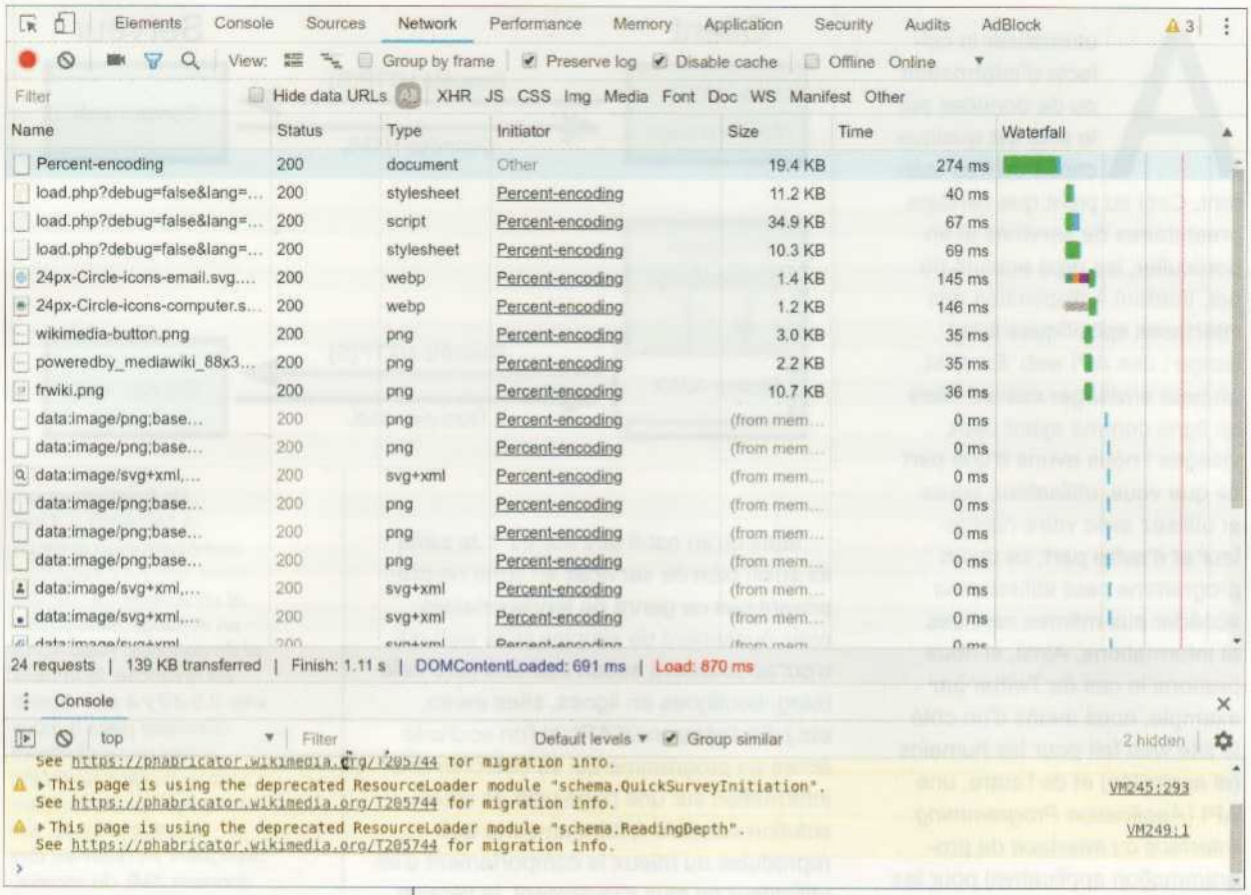


Un fonctionnement « classique » d'une communication entre un client web (le navigateur) et un serveur (le site) est un échange de requêtes et de données. Avec AJAX en revanche, le fameux web 2.0 d'il y a une dizaine d'années (déjà !), nous avons un ou plusieurs scripts, fonctionnant dans le navigateur et modifiant la page affichée, qui dialoguent en recevant des données XML du serveur. Ce type de contenu n'est pas traité dans cet article.

Mais qu'en est-il des autres ? Je parle ici aussi bien de services en ligne ne proposant pas ce genre de fonctionnalités, mais également de simples sites web qui n'ont absolument aucun intérêt à faire cela (blog, boutiques en lignes, sites perso, etc.). En l'absence d'API, si l'on souhaite écrire un programme qui va chercher une information sur une page web, la seule solution consiste à faire en sorte qu'il reproduise au mieux le comportement d'un utilisateur ou plus exactement, la version réduite au strict nécessaire des échanges entre un navigateur et le site web visé.

1. LE PIRE CAS D'ÉCOLE

Comme toujours pour les articles de ce magazine, il ne s'agit pas ici d'une situation hypothétique, mais d'un cas concret de résolution d'un problème bien réel. Tout a commencé par une idée toute simple : récupérer une valeur numérique dans l'interface de gestion de notre boutique en ligne et la présenter sur un afficheur à LED. Rien de bien complexe a priori, puisque cette information s'affiche sur une page HTML, exactement comme n'importe quelle donnée de n'importe quelle page d'un site web.



Les DevTools intégrés à Chromium (et Chrome) permettent de rapidement analyser les échanges entre le navigateur et le site. On peut alors voir apparaître l'ensemble des requêtes effectuées, ainsi que les données obtenues, pour l'affichage complet d'une page. C'est dans cette liste que nous trouverons tous les éléments nécessaires pour notre analyse.

C'était, bien entendu, sans compter un point très important, dès lors que l'on approche un problème de ce genre : ce qui semble aisé dans une tâche manuelle récurrente devient souvent particulièrement complexe une fois déconstruit en une suite d'actions élémentaires, qui doivent être reproduites séquentiellement par un programme. Dans le cas présent, quelque chose d'aussi simple que « se connecter à l'interface de gestion et lire un chiffre » est en réalité une succession d'étapes et d'échanges entre le navigateur web et le serveur. Certaines, comme les redirections, peuvent être sautées et d'autres sont absolument nécessaires.

En plus de cette distinction d'actions, l'exercice est ici compliqué par d'autres mécanismes. L'information devant être obtenue se trouve sur une page qui n'est pas publiquement accessible. Il faut, pour y accéder, se connecter à l'interface de gestion du site qui fonctionne sur la base de PrestaShop, une application Web open source permettant de créer une boutique en ligne. Nous avons donc une page de connexion sur laquelle entrer un nom d'utilisateur et un mot de passe et celle-ci devra être utilisée avec l'ESP8266, afin de nous authentifier avant de pouvoir accéder à ce qui nous intéresse.

Mais ce n'est pas tout, comme il s'agit d'un environnement de production, la connexion doit non seulement se faire en HTTPS (HTTP chiffré utilisant SSL/TLS), mais également via une authentification HTTP préalable avec la méthode dite « Basic ».

Nous avons là sans doute le pire cas qui puisse se poser, car en plus de devoir analyser et reproduire les échanges site/navigateurs, nous ne pourrons utiliser les facilités mises à disposition par la bibliothèque `ESP8266HTTPClient` et en particulier, les méthodes de la classe `HTTPClient` qui ne fonctionnent pas en HTTPS. Cependant, même si tout ceci semble être un horrible cumul de tout ce qui peut nous barrer la route, c'est également un parfait exemple qui pourra se transposer à n'importe quelle autre situation.

Il n'y a qu'un point qui n'est pas présent et aurait complexifié encore davantage les choses : AJAX. Ce terme qualifie une architecture faisant usage de JavaScript et de requêtes de type XMLHttpRequest (AJAX comme *Asynchronous JavaScript And Xml*) pour créer des interfaces Web dynamiques et ergonomiques. Contrairement à une application web classique, une application AJAX (ou Web 2.0) ne repose pas sur un dialogue direct entre le navigateur web et le site (requêtes HTTP et données HTML), mais sur des échanges se faisant entre un code en JavaScript exécuté par le navigateur et le serveur retournant des données au format XML. Le problème qui se pose alors est la difficulté pour une carte comme l'ESP8266 de réellement imiter le comportement d'un navigateur et/ou d'analyser, puis de reproduire les échanges pour les ramener à leur plus simple expression. Le cas typique de ce genre de situation est le site bancaire, nécessitant généralement la saisie d'un identifiant, puis la validation via l'utilisation d'un pavé numérique cliquable à la souris. Ceci est précisément fait pour rendre difficile l'automatisation de l'authentification et, disons-le franchement, tant mieux. Après tout, arriver à programmer son ESP8266 pour accéder ainsi à ses comptes me semble être une très mauvaise idée, en particulier si ça fonctionne...

Le présent article se bornera donc à ne traiter que le cas d'une application web classique, non AJAX. Si vous souhaitez reproduire ce que nous allons étudier, la première chose à faire sera donc de vous assurer de cette compatibilité et de cette indépendance vis-à-vis de JavaScript. Quel que soit le navigateur que vous utilisez (Firefox, Chrome, Chromium), la façon la plus simple de le vérifier est d'installer une extension ou un greffon vous permettant de désactiver/bloquer l'exécution de ces scripts à souhait. Si vous parvenez à vous authentifier sur le site ciblé et à obtenir l'information souhaitée de cette façon, vous pourrez sans doute faire de même avec un ESP8266. Dans le cas de l'applicatif PrestaShop, ceci fonctionne, même si une partie des fonctionnalités ne sont plus au rendez-vous (comme les menus déroulants par exemple).

2. ANALYSER LES ÉCHANGES

Les bibliothèques livrées en standard avec le support ESP8266 pour Arduino permettent de simplifier énormément d'opérations. Malheureusement pour nous, dès lors qu'il s'agit d'une connexion chiffrée, la plupart des classes facilitant ces opérations ne sont plus utilisables. Nous sommes alors obligés de prendre les choses au plus bas niveau : établir la connexion avec le serveur et traiter les données brutes qui nous parviennent. En clair, alors que l'ESP8266 s'occupe du chiffrement seul, nous devons construire les requêtes liées au protocole qui est protégé par SSL/TLS. Dans le cas de HTTPS, il s'agit tout bonnement de HTTP (*L'HyperText Transfer Protocol*) par-dessus SSL/TLS.

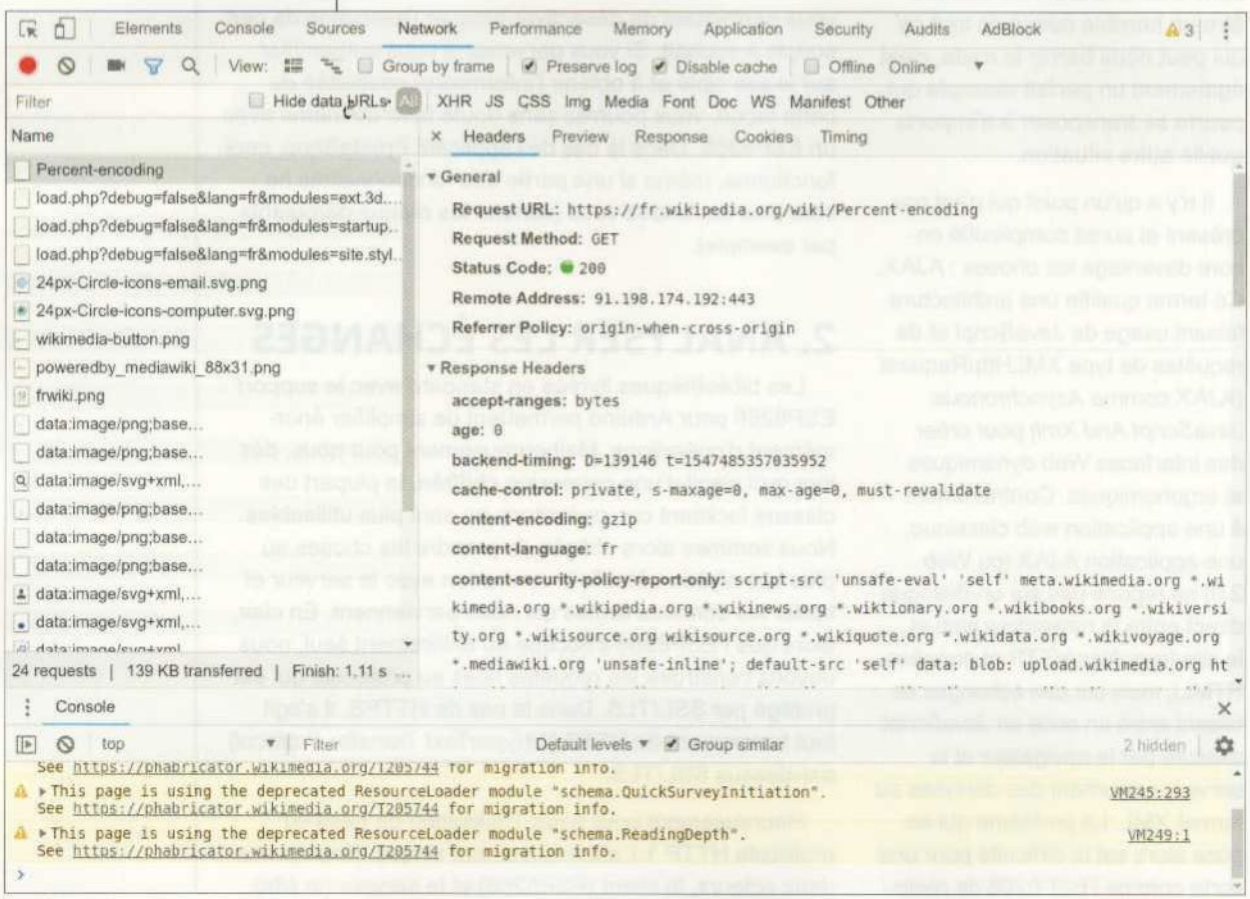
Heureusement pour nous, l'utilisation de base du protocole HTTP 1.1 est relativement simple. Nous avons deux acteurs, le client (ESP8266) et le serveur (le site).

Le client établit la connexion, envoie une requête, le serveur répond et le client fait une nouvelle requête ou termine la connexion. La requête comme la réponse se composent d'un en-tête (métadonnées) et d'un contenu ou corps, séparés par une ligne vide. La toute première ligne de ces messages échangés a une fonction spéciale. Dans le cas d'une requête, c'est une commande (**GET**, **POST**) et pour la réponse, c'est une ligne de statut (avec un code comme 200, 302 ou encore 404).

Ces transactions sont invisibles lorsque vous utilisez un navigateur web et facilitées avec un croquis Arduino pour ESP8266, mais uniquement si la connexion n'est pas chiffrée (SSL/TLS). En cas d'utilisation de HTTPS, vous devez donc construire les requêtes de toutes pièces et analyser les réponses que vous recevrez. Certaines informations importantes pour ce projet sont transmises via l'en-tête et non simplement dans le corps des requêtes, il en va de même pour les réponses. Nous devrons donc travailler sur les deux éléments à la fois.

Pour savoir ce que notre croquis doit envoyer et ce qu'il doit attendre en retour de la part du serveur, nous devons analyser les échanges tels qu'ils se présentent dans le cadre de l'utilisation d'un navigateur.

Chaque requête effectuée vers le serveur peut être inspectée. Il en va de même pour la réponse envoyée par celui-ci. On retrouve alors non seulement les données, mais également les informations d'en-tête, contenant bon nombre des informations dont nous avons besoin pour créer des requêtes et évaluer les réponses avec l'ESP8266.



Heureusement pour nous, les navigateurs modernes offrent un grand nombre de fonctionnalités pour les développeurs. Personnellement, j'utilise Chromium, la déclinaison open source du navigateur de Google et de ce fait, les explications qui vont suivre reposent sur ce logiciel. Des fonctionnalités similaires sont également disponibles avec Firefox de Mozilla et bien entendu, avec Chrome.

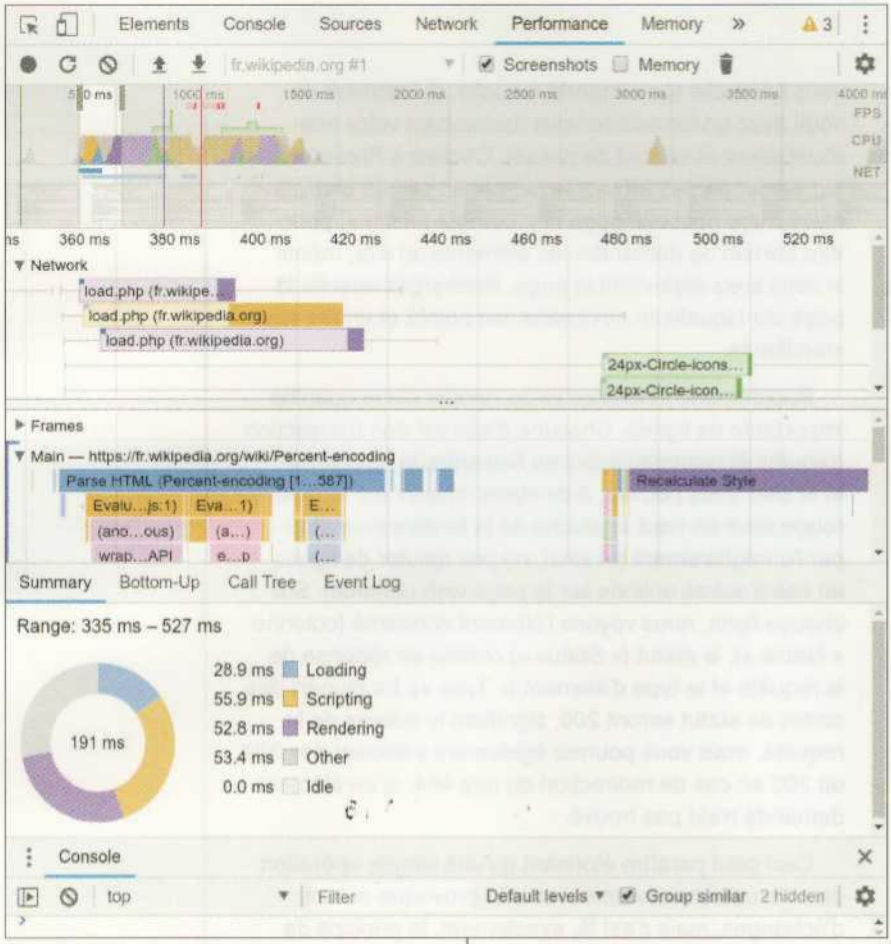
Les fonctionnalités qui nous intéressent ici sont regroupées sous la désignation « DevTools », directement intégrées au navigateur. Pour y accéder, il suffit de vous rendre sur le site ou la page web que vous voulez analyser et activer la fenêtre DevTools, en cliquant avec le bouton droit de la souris sur un élément qui vous intéresse et en choisissant « Inspecter ». Selon votre configuration, les DevTools s'afficheront sur un morceau de la fenêtre en cours d'utilisation ou dans une nouvelle fenêtre. Vous y trouverez une quantité impressionnante de choses permettant d'analyser et même modifier dynamiquement différents éléments de la page affichée dans le navigateur.

Ce qui nous intéresse ici n'est cependant pas vraiment le contenu de la page, les performances, la mémoire utilisée ou encore les scripts exécutés, mais plus précisément les interactions entre le navigateur et le site. Pour cela, les DevTools fournissent une fonctionnalité intéressante à l'onglet « Network ». Rendez-vous simplement à cet endroit, une fois la fenêtre des DevTools affichée et sur la page de connexion qui vous intéresse (peu importe laquelle, du moment où vous avez un formulaire vous demandant votre nom d'utilisateur et un mot de passe). Cochez « Preserve log » pour ne pas effacer le journal en cas de chargement d'une nouvelle page et « Disable cache », pour être certain de demander les éléments au site, même si vous avez déjà visité la page. Rechargez ensuite la page sur laquelle le navigateur est pointé et entrez vos identifiants.

Rapidement, la fenêtre va se remplir d'une quantité importante de lignes. Chacune d'elle est une transaction (requête et réponse) ayant eu lieu entre le navigateur et le site. Vous pouvez, à ce stade, cliquer sur le rond rouge situé en haut à gauche de la fenêtre pour stopper l'enregistrement (et ainsi, ne pas ajouter de lignes en cas d'autres actions sur la page web obtenue). Sur chaque ligne, nous voyons l'élément concerné (colonne « Name »), le statut (« Status ») obtenu en réponse de la requête et le type d'élément (« Type »). La plupart des codes de statut seront 200, signifiant le succès de la requête, mais vous pourrez également y trouver des 301 ou 302 en cas de redirection ou des 404, si un élément demandé n'est pas trouvé.

Ceci peut paraître étonnant qu'une simple opération comme une connexion sur un site provoque autant d'échanges, mais c'est là, exactement, le principe de fonctionnement du Web. Lorsque vous demandez une page, votre navigateur va la recevoir de la part du serveur, mais celle-ci, écrite en HTML, référence d'autres éléments (ou ressources) comme des images, des scripts ou encore des feuilles de style (CSS). Le navigateur va alors automatiquement procéder à de nouvelles requêtes pour obtenir ces éléments et vous afficher la page, telle qu'elle doit être vue.

Bien entendu, nous n'avons pas l'intention d'afficher des pages web et n'aurons donc pas besoin de faire autant de requêtes. La seule chose qui nous intéresse en réalité, ce sont les données ou ressources en HTML. Nous pouvons alors n'afficher qu'une partie des lignes en utilisant un filtre (icône en forme d'entonnoir) et en précisant « mime-type:text/html ». La liste devrait alors se résumer à une poignée de lignes et très certainement trois (ou pas loin). Vous pourrez alors cliquer sur le nom dans la première colonne pour afficher davantage d'informations. Ce qui nous intéresse se trouve à l'onglet « Header ». Là, nous trouvons :



L'enregistrement et l'analyse des échanges entre le navigateur et le site ne sont qu'une fonctionnalité mise à disposition par les DevTools de Chromium à l'attention des développeurs. L'analyseur de performances que l'on peut voir ici en est une autre, il apporte de précieuses informations sur la vitesse de fonctionnement d'un site et la complexité des pages qu'il fournit.

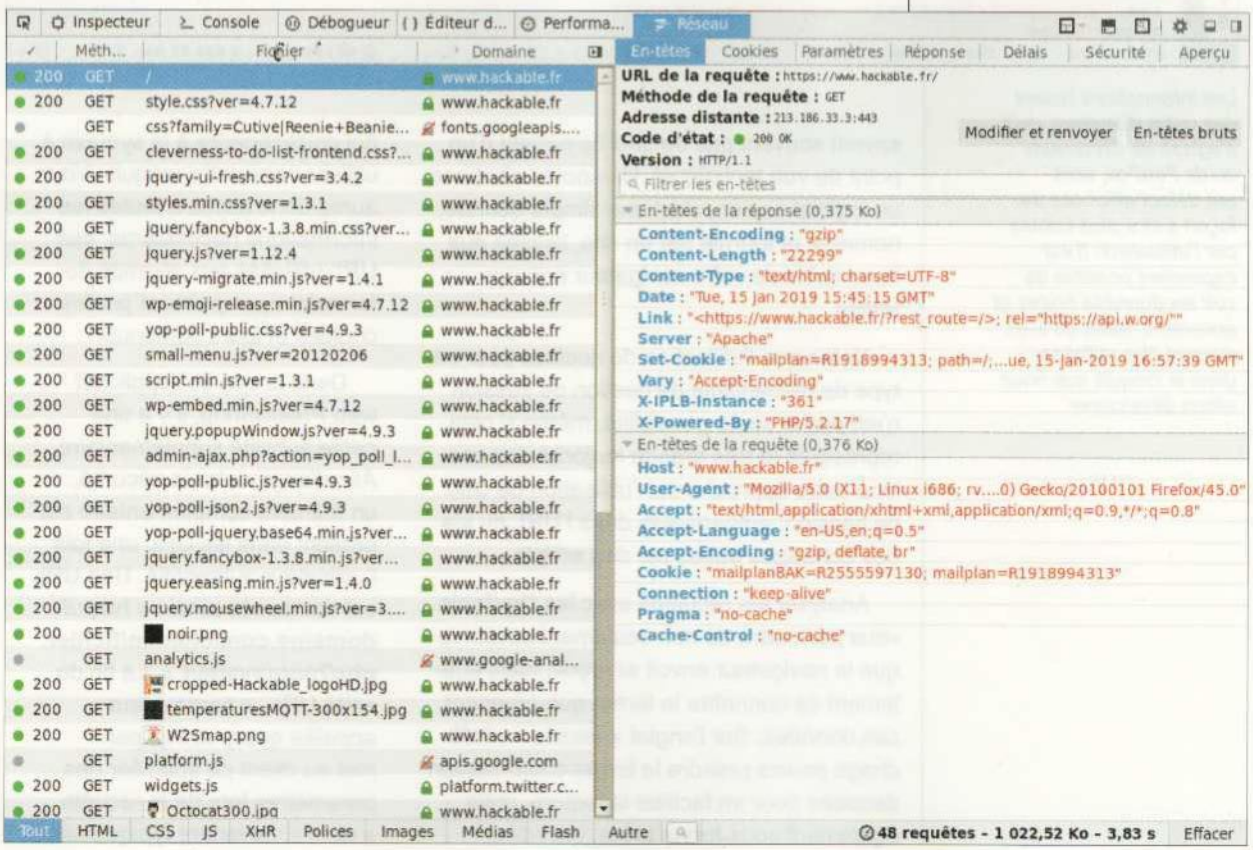
- « General » montre un résumé des informations pour cet échange. On y trouve l'URL concernée, ainsi que la méthode utilisée (généralement, GET pour demander une ressource ou POST pour demander à « modifier » une ressource) ou encore le code lié à la réponse ;
- « Response Headers » est l'en-tête associé à la réponse reçue du serveur. Celui-ci précise, entre autres choses, la taille des données envoyées, leur type ou encore le nom et le contenu d'éventuels cookies (cf. plus loin) ;
- « Request Headers » est l'en-tête de la requête que nous avons envoyé qui contient, par exemple, notre identifiant de navigateur (le *User-Agent*), le type de données que nous acceptons en réponse ou encore les noms des cookies en notre possession et leur contenu.

Vous pouvez tester cela sur n'importe quelle page de n'importe quel site pour observer les échanges au fil de votre navigation. Les cas les plus intéressants pour nous sont ceux de pages de connexion, qui se résument généralement à trois requêtes concernant des pages HTML. Pourquoi trois ? C'est tout simple et lié à la façon dont une connexion ou plus exactement, une authentification via un formulaire HTML, fonctionne :

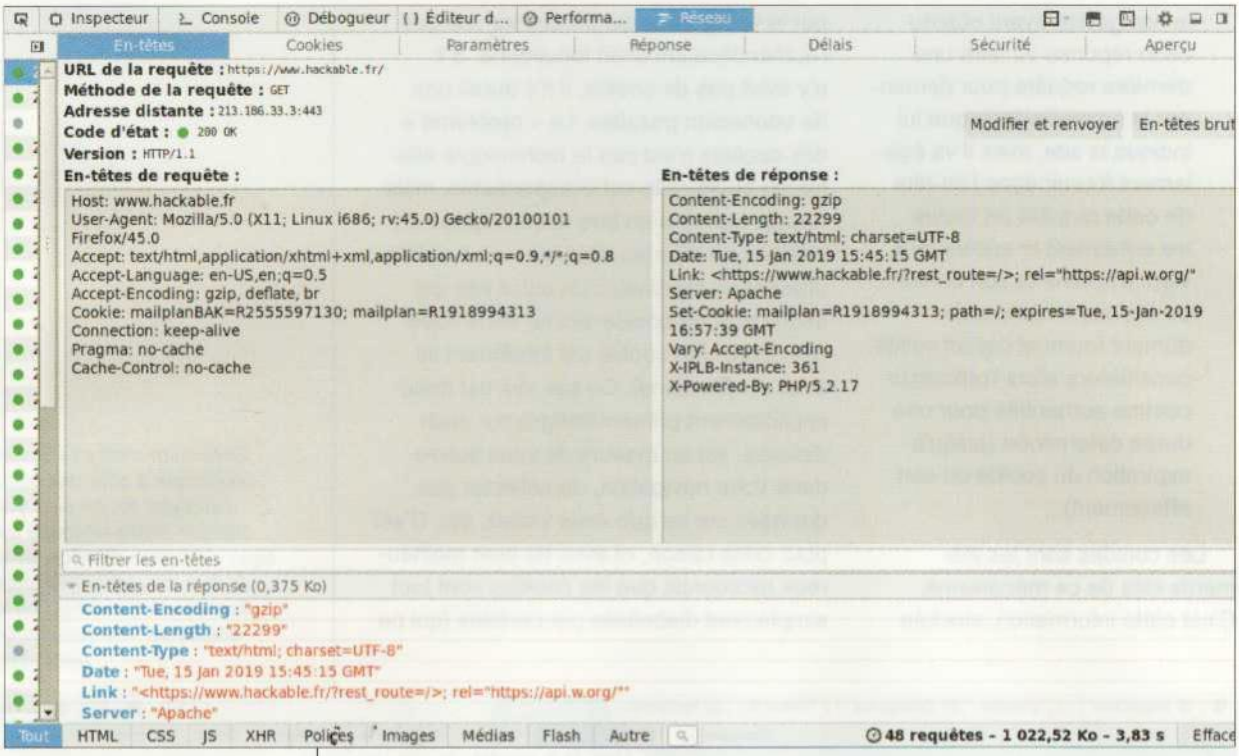
- tout d'abord le navigateur demande, via une requête GET, la page contenant le formulaire HTML permettant à l'utilisateur de préciser son nom et son mot de passe. Le serveur retourne alors la page avec un code 200 ;
- l'utilisateur complète le formulaire avec les informations et valide. Le navigateur produit alors une requête POST pour envoyer ces données au site, qui va répondre avec un code 302. Il s'agit d'une redirection et le site précise dans l'en-tête de sa réponse non seulement où le navigateur doit se rendre pour la suite, mais également un cookie et son contenu ;

- le navigateur ayant obtenu cette réponse va faire une dernière requête pour demander la nouvelle page que lui indique le site, mais il va également fournir dans l'en-tête de cette requête (et toutes les suivantes) le cookie qu'il vient d'obtenir et son contenu. Le site voyant le cookie dûment fourni et l'ayant validé, considérera alors l'utilisateur comme authentifié pour une durée déterminée (jusqu'à expiration du cookie ou son effacement).

Les cookies sont les éléments clés de ce mécanisme. C'est cette information, stockée par le navigateur, qui prouve au site que l'authentification avait fonctionné. S'il n'y avait pas de cookie, il n'y aurait pas de connexion possible. Le « problème » des cookies n'est pas la technologie elle-même puisqu'elle est indispensable, mais l'usage qui peut en être fait. Imaginez un instant que tous les sites que vous visitez utilisent un morceau d'un autre site qui définit et lit un cookie stocké dans votre navigateur (un cookie est forcément lié à un site/domaine). Ce site, qui est donc implicitement présent/intégré sur plein d'autres, est en mesure de vous suivre dans votre navigation, de collecter des données sur ce que vous visitez, etc. C'est pour cette raison, et avec de bien malheureux raccourcis que les cookies sont tout simplement diabolisés par certains (qui ne



Chromium n'est pas le seul navigateur à offrir des outils d'analyse. Ici, on peut voir quelque chose de tout à fait équivalent dans FireFox, avec la liste des échanges entre client et serveur, ainsi que le détail pour une requête en particulier.



Les informations issues des outils d'analyse, qu'il s'agisse de Chromium ou de FireFox, sont par défaut affichées de façon à être plus lisibles par l'utilisateur. Il est cependant possible de voir les données brutes et encodées, telles qu'elles devront être utilisées dans le croquis que nous allons développer.

savent souvent pas de quoi ils parlent d'un point de vue technique). Un cookie n'est pas un logiciel espion, c'est une simple donnée, nommée et fournie par un site, laissée aux bons soins de votre navigateur pour un temps.

Notez que l'utilisation de cookies pour ce type de mécanisme de gestion de session n'est pas la seule possibilité, même si cela représente la très grande majorité des cas. Une autre approche est l'utilisation de clés de session, embarquées dans l'URL ou via des échanges traités par des scripts.

Analyser les requêtes avec les DevTools vous permettra de non seulement savoir ce que le navigateur envoit et reçoit, mais également de connaître la forme que prennent ces données. Sur l'onglet « Header », l'affichage pourra prendre la forme d'information décodée pour en faciliter la lecture, mais également sous forme brute, encodée, qui correspondra à la syntaxe à utiliser dans votre croquis. Vous aurez donc accès à toutes les informations nécessaires, des URL utilisées aux champs du formulaire, en passant par les cookies et leurs contenus.

Dans le cas de l'applicatif web PrestaShop, il y a une petite subtilité supplémentaire. Afin d'augmenter la sécurité, un élément complémentaire est présent dans les URL une fois l'utilisateur authentifié. Une URL est composée ainsi : « **http://domaine.com/chemin/fichier.php?nom=valeur** ». La fin de cette URL, « nom=valeur », appelée *query string*, permet au client de spécifier des paramètres lors de la requête « nom », devenant typiquement

une variable contenant « valeur » côté serveur et pouvant être utilisée pour passer des paramètres. PrestaShop fait usage de ce mécanisme comme de nombreux autres applicatifs web, mais l'utilise également pour augmenter la sécurité, via un paramètre « token » précisant une donnée dont l'intégrité doit être maintenue d'une page à l'autre dans l'interface d'administration. Ce n'est pas un gros problème dans notre cas, puisque la redirection conséquente à l'authentification inclut cette *query string*, mais c'est quelque chose à savoir si vous souhaitez explorer plus avant. Vous ne pouvez pas accéder arbitrairement à des URL pointant dans l'interface d'administration, sans prendre en compte ce token.

En analysant les échanges entre le navigateur et le site dans notre exemple, les choses s'avèrent relativement simples :

- la page de login de l'interface d'administration est accessible via l'URL : **https://boutique.com/admin/index.php?controller=AdminLogin** ;
- s'y trouve le formulaire permettant de spécifier le nom d'utilisateur et le mot de passe qui prendra la forme d'une requête POST sur **https://boutique.com/admin/index.php?controller=AdminLogin** ;
- une authentification réussie via cette requête déclenche une réponse de la part du serveur, avec une redirection (302) vers la page d'accueil d'administration pour cet utilisateur. Un cookie dont le nom débute par `PrestaShop-` est également défini. Si cela échoue, la plage de formulaire est renvoyée avec un message d'erreur ;
- en cas de réussite, il suffit de procéder à une nouvelle requête vers la page qui nous intéresse, en présentant le cookie et en incluant le token dans l'URL, tel qu'il aura été spécifié dans la redirection.

Tout ceci n'est pas apparu comme une évidence immédiatement. C'est l'analyse des échanges et bon nombre d'essais, via un outil en ligne de commandes (`curl`) sous GNU/Linux, qui ont permis d'arriver à cette conclusion. Je vous recommande donc de faire de même et surtout, de vous armer de patience. Si le site visé fonctionne sans JavaScript, il est forcément possible de faire de même. Si ça ne marche pas, c'est donc que vous avez oublié quelque chose et il vous faudra persévérer...

3. ASSEZ POUR LA THÉORIE

Si vous n'êtes pas coutumier des interactions entre un navigateur et un site web, je vous recommande fortement de ne pas foncer directement vers la mise en application de ces échanges. Mieux vaut prendre votre temps en étudiant non seulement les communications entre le navigateur et la page qui vous intéresse, mais aussi et surtout, avec d'autres sites et pages.

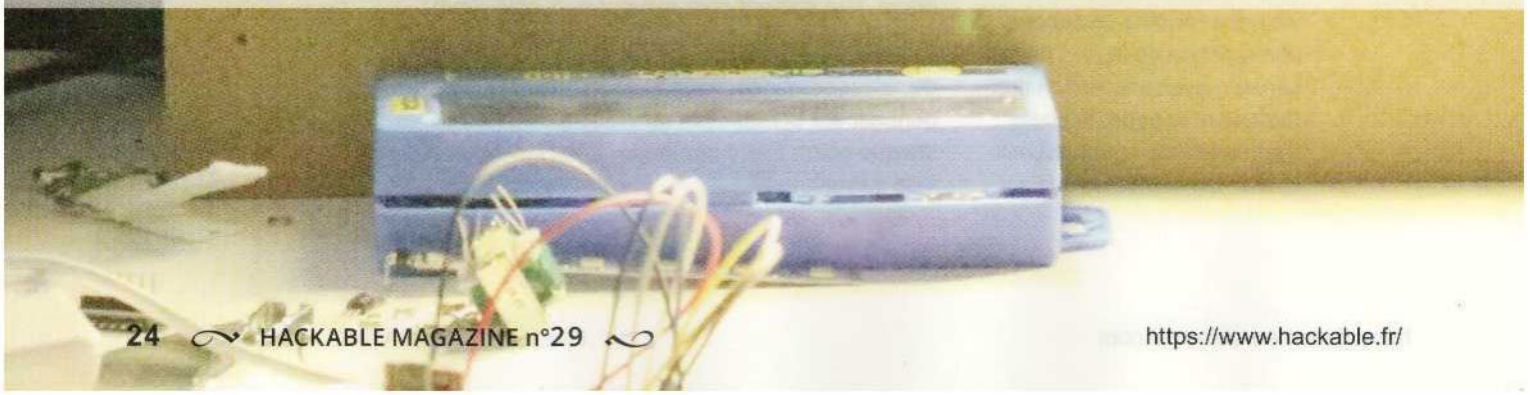
Commencez petit avec des pages simples et un nombre limité de ressources composant les pages. Il est bien plus difficile d'y voir clair lorsqu'on travaille sur une page en HTML 5 regroupant un nombre terrifiant d'images, de feuilles de style et de script. Éventuellement, si le cœur vous en dit et si vous souhaitez intégralement maîtriser votre processus d'apprentissage, il peut être très intéressant d'installer localement un serveur web comme Lighttpd (sur un PC GNU/Linux ou une Raspberry Pi). Ceci vous permettra de monter en complexité petit à petit et d'ajouter des fonctionnalités (formulaire, redirection, authentification, etc.) étape par étape, tout d'abord côté serveur puis côté client.

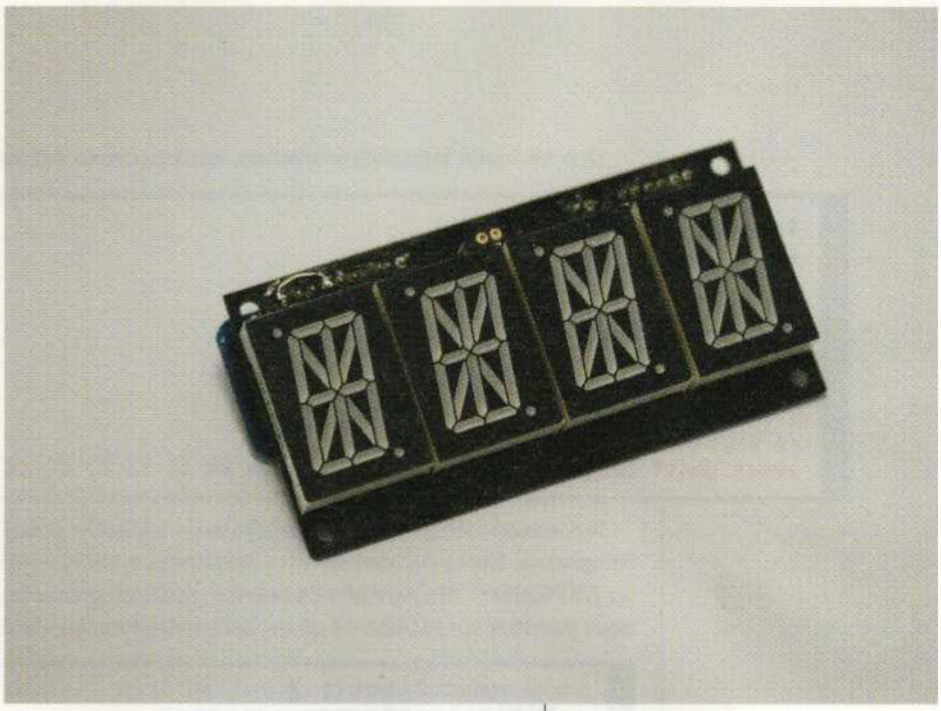
En ce qui concerne notre petit projet, la phase d'analyse terminée, nous n'avons plus qu'à nous pencher sur la rédaction du croquis en traitant chaque point, étape par étape. C'est l'objet de l'article suivant... DB

COLLECTEZ DES DONNÉES SUR LE WEB AVEC VOS ESP8266 : DU CODE !

Denis Bodor

Après avoir vu toute la partie théorique de ce qui fait les échanges entre un navigateur et un site web, et découvert comment espionner et analyser ces communications, il est temps de passer à quelque chose d'un peu plus créatif. Armés de toutes les informations nécessaires, nous pouvons enfin nous pencher sur le code et commencer à obtenir des résultats.





Nous travaillerons ici par étapes, abattant chaque difficulté l'une après l'autre, pour arriver jusqu'au but. Pour rappel, notre objectif est d'extraire des informations numériques de pages d'administration d'une boutique en ligne de type PrestaShop. Au menu, nous devons gérer le HTTPS, passer l'authentification en place, prouver notre identité à l'interface d'administration et arriver à lire une page autrement inaccessible. Sans plus attendre, attelons-nous à la tâche !

1. ÉTABLIR UNE CONNEXION HTTPS

Nous passerons sous silence ici toute la partie standard de connexion au wi-fi, qui se fait tout à fait classiquement, pour nous intéresser uniquement à l'objet de l'article. Notez au passage un conseil qui peut vous être utile : j'ai pris pour habitude de systématiquement, pour chaque projet impliquant un ESP8266, enregistrer dans l'EEPROM émulée le SSID du point d'accès, le mot de passe associé et le nom d'hôte de l'ESP8266 (voir Hackable 26 et l'article sur MQTT). Le croquis de départ vierge pour débuter un projet avec cette carte intègre donc, de base, le code pour lire ces données qui ne sont pas écrasées lors de la reprogrammation de l'ESP8266, se connecter au wi-fi et configurer la mise à jour OTA (voir Hackable 21).

Revenons à notre sujet, pour établir une connexion HTTPS, nous ne nous occupons pas de HTTP, mais nous nous connectons simplement au serveur sur le port 443, en utilisant SSL/TLS. HTTPS est en réalité HTTP sur SSL/TLS. Cette connexion représente la couche chiffrée et ce sera à nous de « parler » HTTP avec le serveur, via ce canal de communication protégé. SSL/TLS n'assure pas que le chiffrement des communications, ce protocole permet également de sécuriser cryptographiquement l'identité des intervenants et en particulier, du serveur. En tant que client, nous pouvons vérifier ce point en inspectant l'empreinte du certificat du serveur (grossièrement, une version condensée de sa carte d'identité).

Cette empreinte du certificat, ou *fingerprint*, peut être obtenue depuis le navigateur, en cliquant au début de l'URL et en cliquant sur « Certificat » (dans Chromium). Dans les détails du certificat utilisé par le site qui apparaissent, repérez l'empreinte SHA-1 et copiez-la dans votre code. D'autres fonctions de hachage cryptographiques peuvent théoriquement être utilisées (comme SHA-256 à la place de SHA-1, si l'empreinte est disponible dans le certificat), mais il est généralement d'usage de prendre la plus concise afin d'économiser la mémoire.

Ce type d'afficheur à LED est très intéressant bien que relativement coûteux (environ 25€ pour ce DSP-0401B de chez Embedded Adventures). Il se pilote à l'aide de quelques connexions, est compatible avec des signaux en 3,3v et permet surtout d'afficher aussi bien des chiffres que du texte sur des afficheurs 16 segments de 35 mm de haut, qui se voient de loin.

Une fois cette information obtenue, nous pouvons débuter notre croquis :

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

// site
const char* host = "boutique.com";
// port IP (défaut HTTPS)
const int httpsPort = 443;

// Empreinte SHA-1 du certificat
const char* fingerprint = "0B 0D 94 6B 4B E8 63 56 C7 CC 4E 47 A2 96 5B B6 27 F5 27 B3";
```

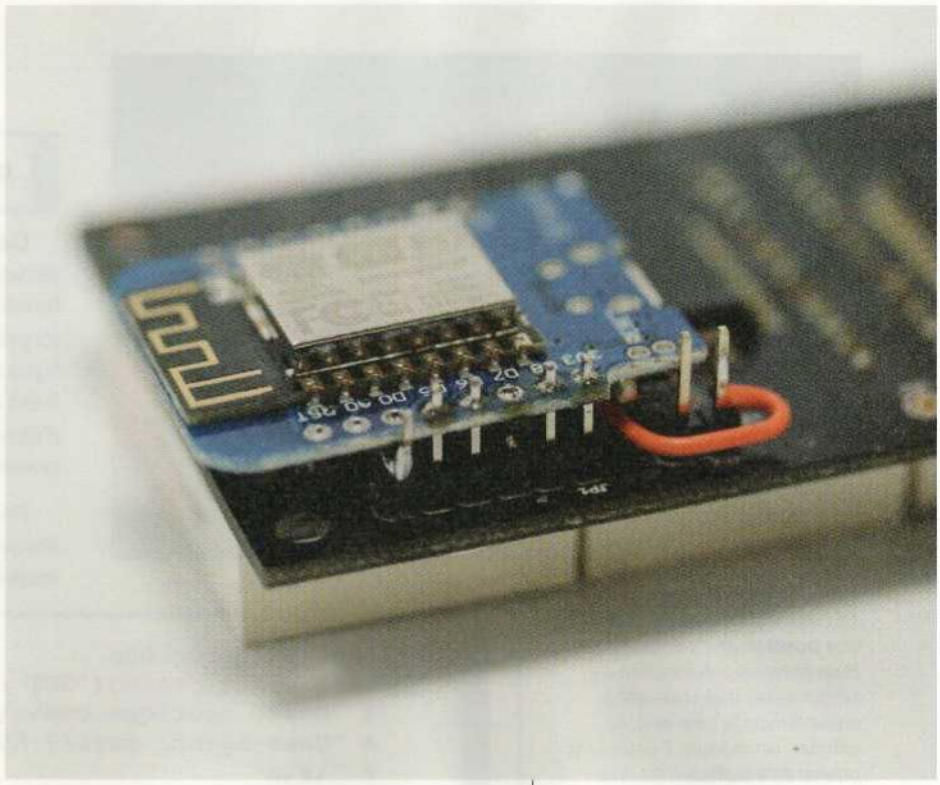
Nous avons ici quelques déclarations de variables pour l'hôte (le site web), le port utilisé et l'empreinte. Notez l'utilisation de la bibliothèque *WiFiClientSecure* en lieu et place de *WiFiClient* ou *ESP8266HTTPClient* afin de pouvoir gérer les connexions SSL/TLS. Nous pouvons ensuite nous pencher sur la base de ce qui sera notre fonction de collecte d'information :

```
void webcollect() {
  WiFiClientSecure client;

  Serial.println(F("Connexion au serveur Web"));
  if (!client.connect(host, httpsPort)) {
    Serial.println(F("Erreur connexion serveur web!"));
    return;
  }

  // vérification certificat SSL/TLS
  if (client.verify(fingerprint, host)) {
    Serial.println(F("Certificat ok"));
  } else {
    Serial.println(F("Mauvais certificat !"));
    client.stop();
    return;
  }

  client.stop();
}
```

Rien de bien extraordinaire ici, nous avons notre fonction `webcollect()` qui établit la connexion avec `client.connect()` en précisant l'hôte et le port. Comme `client` est un objet de type `WiFiClientSecure`, la connexion sera automatiquement en SSL/TLS. Une fois la connexion établie, nous pouvons alors utiliser la méthode `verify()` en spécifiant, en argument, l'empreinte et le nom d'hôte du serveur. Si l'empreinte correspond, nous pourrons nous occuper de la suite et si ce n'est pas le cas, nous terminons la connexion et quittons la fonction.

2. UTILISER L'AUTHENTIFICATION BASIC

Nous sommes maintenant en mesure de communiquer avec le serveur de façon sécurisée. Il ne nous reste donc plus qu'à discuter avec lui en HTTP via ce canal de communication fraîchement établi. Comme nous avons une authentification, la première chose que nous devons pouvoir achever est le fait d'obtenir une page web de la part du serveur.



Un autre avantage des modules d'affichage à LED vendus « tout-fait » est la possibilité de directement intégrer l'ESP8266 à l'arrière. Il faut un peu bricoler avec les connexions, mais on arrive relativement facilement à dissimuler un Wemos D1 mini à l'arrière. Malheureusement, l'alimentation en 5v des segments n'est possible que pour certaines couleurs (rouge et jaune). Pour les autres (vert, blanc et bleu) l'ESP8266 ne pourra fournir la tension demandée.

Le type d'authentification auquel nous avons affaire est dite « HTTP Basic ». Ceci consiste à envoyer un nom d'utilisateur et un mot de passe au serveur dans l'en-tête de la requête. Si ce couple est valide et autorisé, le serveur affiche la page, sinon il retourne un code 401. Votre navigateur fait, encore une fois, cela de façon transparente. En tentant d'accéder à une ressource de ce type, il reçoit en réponse un code 401 et affiche un formulaire vous demandant de spécifier les informations requises. Tant que vous entrerez de mauvaises informations, ce formulaire se réaffichera jusqu'à ce que vous appuyiez sur Echap et, là seulement, vous verrez le message d'erreur incluant le code 401.

Dans notre cas, nous n'allons pas nous amuser à demander ces informations à qui que ce soit et nous les embarquons simplement dans l'en-tête de notre requête. Il est également possible de faire cela avec votre navigateur en utilisant une URL comme **https://utilisateur:mot2passe@site.domaine.com**. Dans ce cas, le formulaire ne s'affichera pas (si les informations sont correctes) et la page demandée sera affichée. Nous ne pouvons pas utiliser cette syntaxe dans notre croquis, car c'est une facilité offerte par le navigateur qui prend les informations, les encode et les utilise pour créer sa requête. Nous, nous allons devoir faire ce travail nous-mêmes.

Une requête HTTP n'est pas quelque chose de bien complexe. Si nous voulons, par exemple, obtenir la page `toto.html` sur le site boutique.com, il nous suffit de formuler la requête suivante :

```
GET /toto.html HTTP/1.1\r\n
Host: boutique.com\r\n
\r\n
```

Nous avons en première ligne la méthode utilisée (`GET`), suivie de la ressource demandée (`/toto.html`) et de la mention du protocole utilisé (`HTTP/1.1`). Chaque ligne se termine systématiquement par les deux caractères spéciaux CR (*Carriage Return*, `0x0D`) et LF (*Line Feed*, `0x0A`). La seconde ligne précise un en-tête spécifiant le nom d'hôte concerné. Ceci est obligatoire avec HTTP version 1.1 et permet à un seul serveur d'héberger plusieurs sites (notion d'hôtes virtuels). La requête ne comportant ici pas de corps, elle se termine simplement par une ligne vide.

Ceci est un strict minimum, mais il est d'usage de préciser d'autres en-têtes comme, par exemple, celui permettant de préciser le client web (ou agent) utilisé :

On voit ici clairement les 16 segments qui composent une position de l'afficheur. Bien entendu, en fonction de l'information que vous allez collecter sur le web et donc afficher, un module 7 segments pourra être suffisant (et plus économique). Il s'avère que dans mon cas, je n'en avais pas sous la main.

```
User-Agent: curl/7.55.0\r\n
```

Ceci peut tantôt être important, si le serveur réagit de façon différente en fonction du navigateur qu'il détecte. J'ai ici précisé l'utilisation de Curl, un outil en ligne de commande pour GNU/Linux, mais il est relativement simple de faire semblant d'être un vrai navigateur, en réutilisant simplement son *User Agent*.

Procéder à ce type de requêtes dans un croquis, une fois la connexion établie, est relativement simple :

```
// Nouvelle requête
client.print(String("GET /toto.html HTTP/1.1\r\n")
  + "Host: boutique.com\r\n"
  + "User-Agent: curl/7.55.0\r\n"
  + "\r\n"
  );
```

Le simple fait d'utiliser la méthode `print()` sur l'objet représentant notre connexion fait l'affaire et il nous suffit de jouer avec la concaténation des chaînes pour obtenir quelque chose de visuellement intelligible. Pour traiter la réponse, une simple boucle fera l'affaire :

```
String line;

// Récupération/affichage en-tête
while(client.connected()) {
  line = client.readStringUntil('\n');
  if (line == "\r") {
    Serial.println(F("en-tête terminé"));
    break;
  }
  line.trim();
  Serial.print("> ");
  Serial.println(line);
}

Serial.println(F("Fermeture connexion"));
client.stop();
```

Ici, nous nous limitons à l'affichage de l'en-tête de la réponse et coupons la connexion immédiatement une fois la ligne vide reçue. Ce code devrait vous renvoyer, dans le moniteur série, quelque chose comme :

```
> HTTP/1.1 200 OK
> Date: Wed, 09 Jan 2019 15:26:06 GMT
> Content-Type: text/html; charset=UTF-8
> Transfer-Encoding: chunked
> Server: Apache
> X-Powered-By: PHP/5.2.17
> Expires: Wed, 11 Jan 1984 05:00:00 GMT
```

« Et mon authentification dans tout ça ? » me direz-vous. Il ne s'agit que d'une information à glisser dans l'en-tête de la requête, mais ceci doit se faire d'une certaine manière. Il s'agit de l'en-tête **Authorization:**, s'utilisant exactement comme **User-Agent:** ou **Host:** et en précisant la méthode d'authentification (ici **Basic**), suivi des informations « utilisateur:mot2passe » encodées en base64.

Vous avez plusieurs solutions pour générer cette information dans le format attendu :

- tout simplement, inspecter les échanges entre le navigateur et le site comme nous l'avons fait précédemment. Cette information est présente dans l'en-tête de la requête et correspond à ce que vous avez saisi dans le formulaire, présenté par le navigateur ;
- utiliser un outil en ligne de commandes sous GNU/Linux, comme **base64** pour obtenir la version encodée. Par exemple, **echo -n "utilisateur:passe" | base64** vous affichera **dXRpbGlzYXRldXI6cGFzc2U=** ;
- utiliser un encodeur/décodeur base64 en ligne qui, en entrant une chaîne de caractères, vous affichera la version encodée. Étant donnée la nature secrète de l'information, je ne recommande pas cette méthode (entrer un mot de passe sur un site, qui n'est pas celui à qui il est destiné, n'est jamais une bonne idée).

Une fois le duo nom d'utilisateur et mot de passe, séparé par un « : », encodé en base64, il vous suffit d'ajouter une ligne dans votre requête (et toutes les suivantes) :

```
// Nouvelle requête
client.print(String("GET /toto.html HTTP/1.1\r\n")
  + "Host: boutique.com\r\n"
  + "User-Agent: curl/7.55.0\r\n"
  + "Authorization: Basic dXRpbGlzYXRldXI6cGFzc2U=\r\n"
  + "\r\n"
  );
```

Et la requête fonctionnera comme s'il n'y avait pas d'authentification nécessaire.

3. POSTER UN FORMULAIRE ET UTILISER LES COOKIES

La méthode HTTP **GET** permet de demander une ressource (page HTML) à un serveur, et c'est ultimement ce que nous ferons pour obtenir l'information que nous souhaitons avoir. Mais pour pouvoir demander cette ressource, il nous faut tout d'abord montrer patte blanche et commencer par poster le formulaire permettant de nous faire accepter par le site. Ceci passe par l'utilisation d'une requête **POST**.

En observant attentivement le contenu de la page proposant le formulaire de connexion, ainsi que les échanges entre le navigateur et le site, nous pouvons assez facilement trouver quelles informations sont envoyées. Un formulaire HTML affiche des zones de saisie qui peuvent avoir plusieurs types (`text`, `password`, `hidden`, `submit`, etc.) mais au final, ceci n'aura pas d'impact sur les données effectivement envoyées. Celles-ci prennent la forme de contenu dans la requête, exactement comme une page HTML est le contenu de la réponse du site. Nous pouvons donc, sur la base de ces informations, construire notre requête comme précédemment :

```
client.print(F("POST /admin/index.php?controller=AdminLogin HTTP/1.1\r\n"
               "Host: boutique.com\r\n"
               "User-Agent: curl/7.55.0\r\n"
               "Authorization: Basic dXRpbGlzYXRldXI6cGFzc2U=\r\n"
               "Content-Type: application/x-www-form-urlencoded\r\n"
               "Content-Length: 59\r\n"
               "\r\n"
               "email=utilisateur%40domaine.fr&passwd=mot2pass&submitLogin="));
```

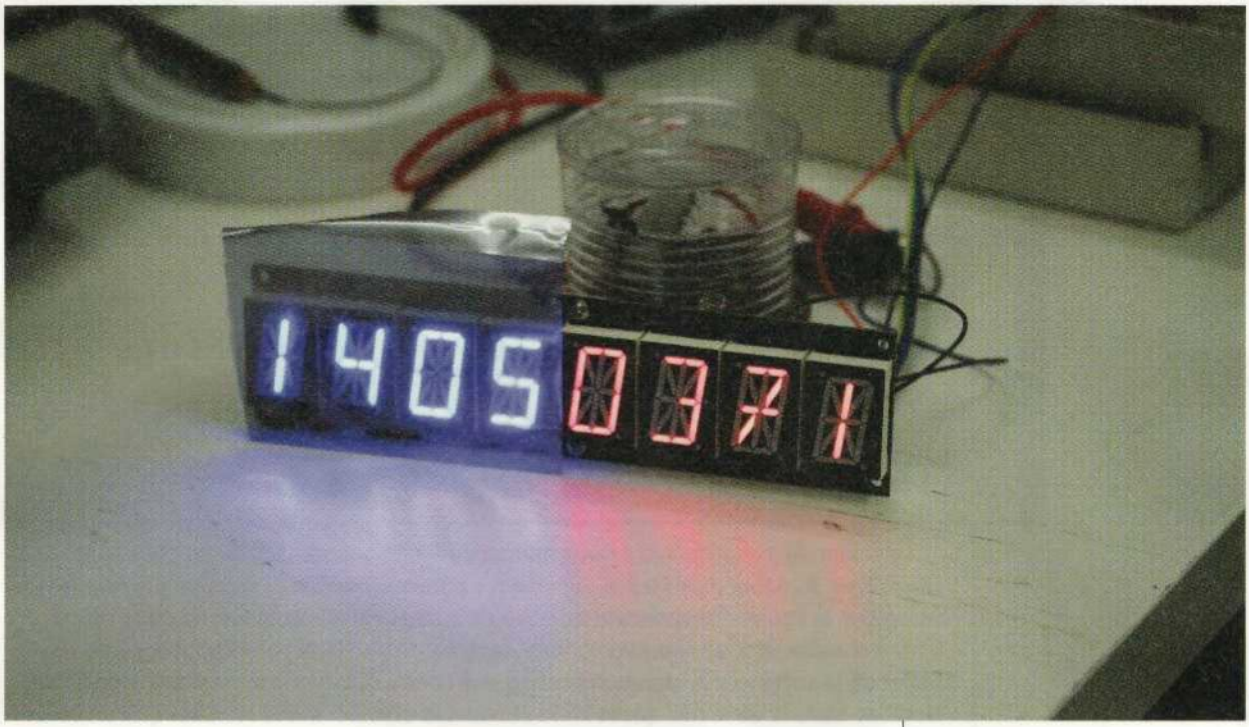
Vous reconnaîtrez sans peine une partie des en-têtes qui sont parfaitement identiques à la requête précédente. Cependant, plusieurs points sont importants :

- nous utilisons une méthode `POST` et non `GET` ;
- nous n'oublions pas de spécifier l'authentification Basic ;
- la ressource concernée embarque ici une *query string* ;
- nous utilisons l'en-tête `Content-Type:` afin de préciser qu'il s'agit du contenu d'un formulaire dans un format particulier ;
- nous devons préciser la taille du contenu que nous envoyons avec l'en-tête `Content-Length:` ;
- et enfin, les données sont passées avec une syntaxe et avec un encodage particulier.

Ce dernier point est très important. Non seulement, nous devons présenter les données sous la forme d'une suite de champs accompagnés de leur valeur (`passwd=mot2pass` par exemple) et séparés par des `&`, mais les informations doivent être partiellement traduites avec un mécanisme d'encodage appelé *percent-encoding* ou *URL encoding*. Suivant cet encodage, certains caractères, dont le `=` et le `&`, ont une signification particulière. Le `&` sert par exemple de séparateur, il ne peut donc pas être présent dans une donnée, sinon il serait interprété comme une délimitation.

En fonction de vos informations à envoyer donc, une traduction (un encodage en réalité) sera nécessaire. On peut voir ici que le `@` de l'adresse mail utilisée comme identifiant d'utilisateur n'est pas présent, mais remplacé par `%40`. Comme avec l'encodage base64, il existe des outils et des services en ligne permettant d'encoder vos informations pour ensuite les copier/coller dans votre croquis. Mais le plus rapide est de tout simplement, une fois encore, récupérer ces informations dans la capture qu'aura fait votre navigateur pour la requête POST.

Dans le cas présent, une authentification réussie aura pour effet une réponse avec un code 302 de la part du serveur, accompagnée de la définition d'un cookie qu'il faudra



Le rendu de ce type de module est du plus bel effet, sachant que les couleurs peuvent être utilisées pour différencier les informations. Leur coût pourra être justifié en partie par le fait de s'en servir pour plus d'une tâche avec, comme ici, l'affichage de l'heure et de la température extérieure, alterné avec celui des données collectées.

ensuite présenter lors de la ou des requêtes suivantes. Si l'authentification échoue, le serveur répondra en envoyant à nouveau la page HTML de connexion. Un code 302 est une redirection et le serveur nous invite à nous rendre sur une nouvelle page. Il n'est en principe pas nécessaire de suivre cette redirection, car dans bien des cas, la réutilisation du cookie sera suffisante pour accéder à n'importe quelle page.

La requête de la partie précédente (le `GET`) n'était qu'un simple test, nous n'avons pas réellement besoin d'accéder à la page de formulaire avant le `POST`. Nous partons ici du principe que le formulaire et donc, les informations à envoyer, ne changeront pas, sauf si l'applicatif web est changé. Si tel était le cas, il faudra certes revoir le script, mais cette solution est plus intéressante qu'un fonctionnement totalement dynamique, qui reviendrait à commencer le développement d'un navigateur.

Notre requête `POST` est donc fiable, il ne nous reste plus qu'à analyser la réponse du serveur et pour cela, nous commençons par déclarer quelques variables :

```
// URL de redirection
String newurl;
// Cookie
String prestacookie;
// ligne de réception
String line;
// Données à récupérer
int nbr1 = 0;
int nbr2 = 0;
```

Comme précédemment, nous allons utiliser une boucle `while()` afin de séquentiellement traiter les lignes de la réponse donnée par le site. Nous ne nous intéressons qu'aux données se

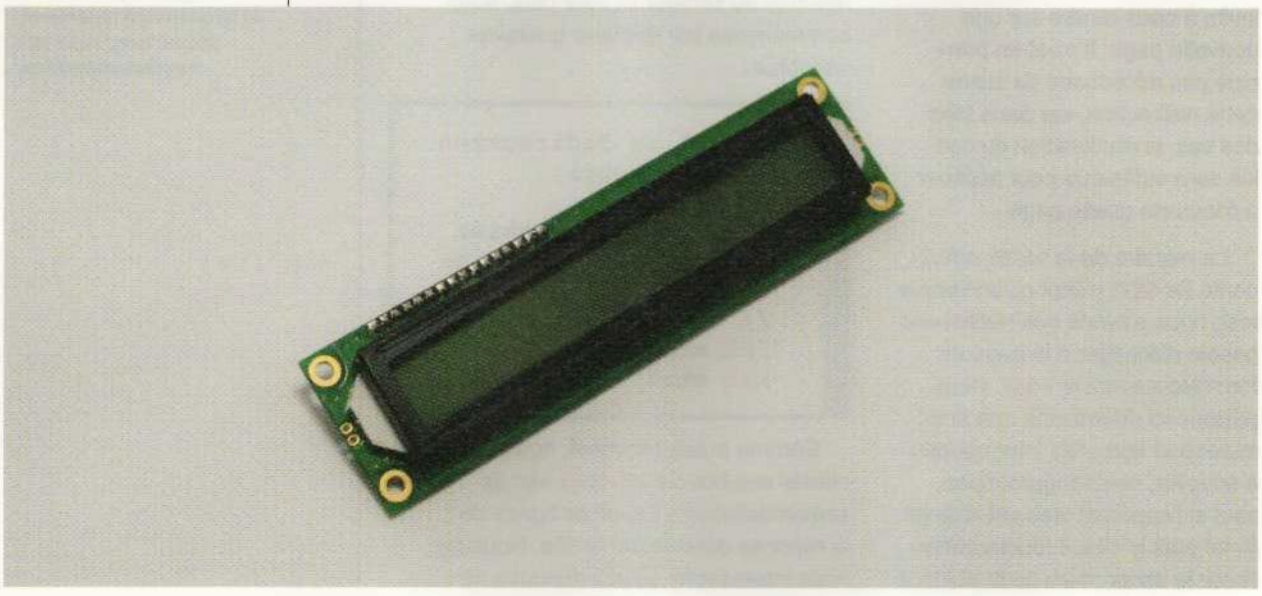
trouvant dans l'en-tête de la réponse, puisque nous sommes censés obtenir une redirection et non des données, en cas de connexion réussie. La première étape consistera à évaluer le code de réponse HTTP :

```
while(client.connected()) {
  line = client.readStringUntil('\n');
  // test code réponse
  if(line.startsWith(F("HTTP/1.1"))) {
    if(!line.startsWith(F("HTTP/1.1 302"))) {
      Serial.println(F("Erreur authentification !"));
      Serial.println(F("Fermeture connexion"));
      client.stop();
      return;
    }
  }
```

La méthode `startsWith()` nous permet de tester la présence d'une chaîne de caractères au début d'une autre. Ainsi, nous pouvons vérifier si, dans un premier temps, il s'agit de la ligne correspondant au code de statut et si tel est le cas, si nous avons bien une redirection temporaire comme espérée (302). Nous sommes obligés de procéder de la sorte, car le standard précise que l'ordre des en-têtes n'est pas significatif. Nous ne savons donc pas quelle ligne contient la donnée attendue. Quoi qu'il en soit, si nous n'avons pas un 302, nous quittons la fonction de collecte après avoir fermé la connexion.

Nous devons également traiter l'URL de redirection. Initialement, mon idée n'était pas de la suivre, mais de simplement extraire le token présent dans l'URL sous la forme d'une *query string* pour le réutiliser à volonté. Il s'avère malheureusement que celui-ci fonctionne comme une sorte de somme de contrôle (ou un condensé) calculée sur

L'afficheur LCD est également une option valable pour ce type d'usage, à condition de se satisfaire de deux limitations importantes : une dimension réduite et la difficulté de trouver un tel module capable de fonctionner avec des signaux 3,3v (ce qui n'est pas le cas de ce modèle).



l'URL complète, le token est ensuite chiffré avec une information d'authentification de l'utilisateur. Ce qu'il faut comprendre ici, c'est tout simplement que le token change pour chaque page et qu'il faudrait donc, en principe, simuler une navigation pour arriver à la page de notre choix. Je rappelle que ceci est une spécificité de l'applicatif web PrestaShop et non un cas courant.

Fort heureusement, dans ce cas précis, nous avons une solution : il est possible, dans l'interface d'administration, de spécifier sa page de départ en tant qu'utilisateur/administrateur (*admin homepage*). Ceci signifie que dès la connexion réussie, PrestaShop redirige l'utilisateur vers une page précise avec, dans l'URL, un token valide. Il nous suffit donc de configurer la page qui nous intéresse pour notre utilisateur et suivre effectivement la redirection. Ce qui simplifie grandement le code sur ce point :

```
// récupération URL de redirection
if(line.startsWith(F("Location: "))) {
  newurl = line.substring(10);
  newurl.trim();
}
```

Enfin, dans l'en-tête de la réponse se trouve également le cookie qui permet au site de savoir, dans les requêtes qui suivent, que nous sommes bien connectés. Nous devons donc le récupérer et le stocker :

```
// récupération cookie
if(line.startsWith(F("Set-Cookie: PrestaShop-"))) {
  int pos = line.indexOf(';',12);
  if(pos>0) {
    prestacookie = line.substring(12,pos);
  }
}
```

Nous y avons échappé pour le token, mais nous devons, pour le cookie, jouer avec les chaînes de caractères. Certes, la classe `String` n'est pas une solution très économe en mémoire (SRAM) et il est possible de faire de même avec des tableaux de caractères C/C++, mais il faut bien avouer qu'avec un ESP8266, les méthodes de cette classe ont souvent plus d'avantages que d'inconvénients. Nous jouons donc du `startsWith()` pour sélectionner la bonne ligne, puis du `indexOf()` pour trouver la fin de la chaîne qui nous intéresse et enfin du `substring()` pour utiliser sa position afin d'obtenir, dans `prestacookie`, une valeur utilisable.

Pour terminer, comme seuls les en-têtes de la réponse nous intéressent, nous pouvons mettre fin à la boucle `while()` avec un `break` pour passer à la suite :

```
    // fin de l'en-tête HTTP
    if (line == "\r") {
      Serial.println(F("en-tête ok"));
      break;
    }
  }
```

4. FAIRE LA REQUÊTE ET EXTRAIRE LES DONNÉES

Nous disposons à présent des éléments nécessaires pour une ultime requête, consistant à demander la page contenant les informations souhaitées. Il existe ici une énième spécificité concernant PrestaShop. En effet, même en configurant une page d'accueil pour l'utilisateur dans l'interface d'administration, l'URL fournie dans la réponse 302 du serveur est incomplète. Lorsqu'on observe les échanges entre un navigateur et le site, il s'avère que l'URL en question, lorsqu'elle est accédée, découle sur une nouvelle redirection avec l'URL finale.

Chose intéressante, la nouvelle URL est exactement la même que la précédente, mais voit une chaîne ajoutée à sa fin. On remarque, pourtant, que le token ne change pas et nous pouvons donc en conclure que tout ce qui est ajouté après la mention du token dans l'URL n'impacte pas sa valeur. C'est une véritable aubaine pour nous, puisque tout ce que nous avons à faire est d'ajouter cette chaîne à l'URL de redirection originale, pour tomber directement sur la bonne page.

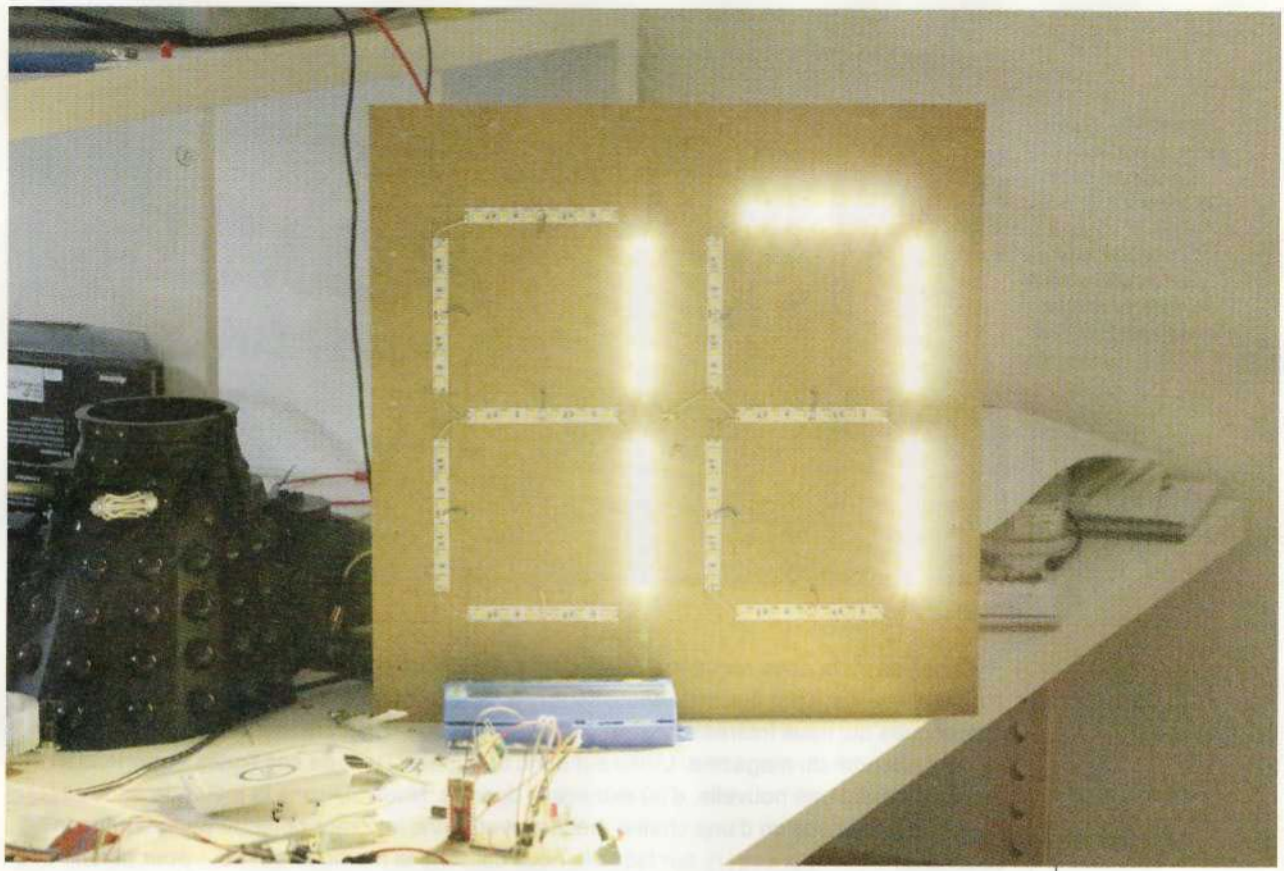
Notre nouvelle requête ressemblera donc à ceci :

```
  client.print(String("GET /admin/") + newurl +
"&ChaineSupplementaire" + " HTTP/1.1\r\n"
    + "Host: boutique.ed-diamond.com\r\n"
    + "User-Agent: curl/7.55.0\r\n"
    + "Authorization: Basic dXRpbGlzYXRldXI6cGFzc2U=\r\n"
    + "Cookie: " + prestacookie + ";\r\n"
    + "\r\n"
    );
```

Je préciserais, à toutes fins utiles, que bien entendu, la requête ici présente a été éditée avant d'être intégrée dans l'article (comme toutes les informations d'authentification). Quoi qu'il en soit, sa structure est identique à celle que vous pourriez vous-même utiliser dans vos développements. On y retrouve les éléments déjà connus, comme `Authorization:`, mais aussi et surtout `Cookie:` qui permet au client de présenter le cookie que le site lui aura précédemment confié.

Nous utilisons ensuite les mêmes techniques pour analyser tout d'abord l'en-tête de la réponse :

```
// Récupération en-tête
while(client.connected()) {
  line = client.readStringUntil('\n');
  if (line == "\r") {
    Serial.println(F("en-tête ok"));
    break;
  }
}
```



Le fait de construire son propre afficheur numérique à partir de rubans de LED, comme nous l'avions fait dans le numéro 11, est une option qui intéressera sans doute les plus manuels. Ici, pas de limitation en termes de dimension des chiffres, mais il faudra être un bon bricoleur pour obtenir un résultat propre.

Puis le corps :

```
// Analyse de la page HTML
while(client.connected()) {
  line = client.readStringUntil('\n');
  if(line.indexOf("Magazine <b>Hackable Magazine") != -1) {
    line = client.readStringUntil('\n');
    line.trim();
    if(!nbr1) {
      nbr1 = line.substring(line.indexOf("<b>")+3).toInt();
    } else {
      nbr2 = line.substring(line.indexOf("<b>")+3).toInt();
    }
  }
  if(nbr1 && nbr2) {
    Serial.println(F("Informations obtenues"));
    break;
  }
}

Serial.println(F("Fermeture connexion"));
client.stop();
```

L'afficheur e-paper tricolore comme celui présenté dans le numéro 27 est sans doute la solution la plus aisée pour un petit projet puisque le matériel combine directement ESP8266 et écran.

L'analyse de la page reçue est totalement spécifique à son contenu et sa structure. Ici, nous traitons ligne par ligne les données HTML obtenues, sachant que les deux valeurs numériques qui nous intéressent sont présentes sur une ligne, précédée d'une autre faisant mention du nom du magazine. L'idée est donc de repérer l'une de ces lignes et d'en obtenir immédiatement une nouvelle, d'où extraire la donnée. Nous utilisons la méthode `indexOf()` pour trouver la position d'une chaîne précise et utilisons la valeur obtenue pour extraire la sous-chaîne (`substring()`), sur laquelle nous utilisons la méthode `toInt()` pour obtenir une valeur entière. Nous partons du principe ici que les deux valeurs qui nous intéressent apparaissent successivement, la première allant dans la variable `nbr1` et la seconde dans `nbr2`. Là encore, si quelque chose change sur la page, il faudra changer le croquis.

Au terme de l'exécution de cette boucle et après la fermeture de la connexion avec `client.stop()`, nous sommes censés avoir dans `nbr1` et `nbr2` des entiers non nuls, correspondants aux données que nous voulions récupérer. Il est alors possible de les utiliser pour un affichage sur le moniteur série ou sur un module connecté à l'ESP8266. Dans mon cas, j'ai fait usage d'un afficheur 16 segments à LED basé sur un TLC5926, vendu par *Embedded Adventures* et pour lequel, j'avais écris une bibliothèque Arduino faisant l'objet d'un article dans le numéro 5 (**https://github.com/Lefinnois/DSP0401B**).

Pour terminer, la fonction `loop()` consiste à simplement déclencher un appel récurrent à la fonction `webcollect()` en fonction du temps écoulé, mesuré avec `millis()` :

```
void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= (MMINUTE*30)) {
    previousMillis = currentMillis;
      webcollect();
  }
  // gestion OTA
  ArduinoOTA.handle();
}
```

Notez l'utilisation d'une macro `MMINUTE` simplifiant la spécification des délais et faisant partie d'une série que j'inclus presque systématiquement à mes croquis Arduino :

```
#define MSECOND  1000
#define MMINUTE  60*MSECOND
#define MHOUR    60*MMINUTE
#define MDAY     24*MHOUR
```

`webcollect()` est appelé une première fois à la fin de la fonction `setup()` puis de façon régulière, toutes les trente minutes dans `loop()`.

5. POUR FINIR

Ce long article présente une situation bien particulière et une démarche jonchée d'embûches, mais qui cependant présente l'avantage d'être très certainement plus complexe que la situation à laquelle vous pourriez être confronté. Dans bien des cas, il n'est pas si difficile d'obtenir l'information voulue, puisque celle-ci se trouvera sur une page visible publiquement. Même en prenant en compte une éventuelle connexion via un formulaire, il devrait vous être relativement facile de faire de même. Bien entendu, chaque site possèdera ses spécificités avec lesquelles vous devrez composer, mais HTTPS, plus l'authentification Basic, plus le formulaire, plus le token, plus les redirections inattendues... forment un mélange relativement rare.

Bien entendu, une approche plus classique est parfois possible avec les mécanismes dédiés au développement que sont les API Web et autres *webservices*. Malheureusement, il faut également bien comprendre que ce n'est pas parce qu'une API est disponible, qu'elle permet forcément de faire ce que vous souhaitez. Si nous prenons l'exemple du moteur de blog WordPress par exemple, une bonne partie des fonctionnalités de base sont utilisables ainsi, mais dans une très grande majorité des cas, ceci ne sera pas valable pour les extensions installées. Les fonctionnalités exposées par une éventuelle API sont totalement dépendantes du bon vouloir des développeurs de cette dernière.

Enfin, je préciserai que, même si la solution consistant à créer un client web comme nous l'avons fait ici reste la voie la plus hasardeuse, c'est aussi celle qui est la moins intrusive. En effet, il n'y a aucun changement à faire sur le site visé, aucune fonctionnalité à activer et aucun greffon à ajouter. Tout se passe comme si l'ESP8266 était une connexion parfaitement normale de la part d'un navigateur. Ceci conviendra donc parfaitement pour un site sensible, où il vaut mieux ne rien toucher... **DB**

CRÉEZ DES CAPTEURS ET DES GRAPHIQUES ENVIRONNEMENTAUX AUTONOMES

Denis Bodor

Coupler une carte Arduino ou ESP8266 à une sonde environnementale (température, hygrométrie, pression) pour collecter des données n'est pas chose compliquée. Si accumuler ces mesures est aisé, il n'en va pas de même pour les représenter. Le plus souvent, même pour une unique sonde, il est nécessaire de mettre en place toute une infrastructure pour obtenir de beaux graphiques. Mais il existe une autre solution : laisser l'ESP8266 représenter les mesures avec du HTML et un peu de JavaScript...

Tout commence généralement bien petit. On branche un DS18B20 ou un module BMP180/BME280 à un ESP8266 et on est tout heureux de voir apparaître les mesures sur le moniteur série. Vient ensuite l'idée de stocker ces données au fil du temps, puis d'en faire de beaux graphiques très faciles à lire et interpréter. Là, les choses se compliquent, car les ESP8266 commencent à envoyer les mesures à un système, en ligne ou local, qui fera fonctionner un serveur Web pour afficher une interface graphique configurable. Au terme du processus, on finit par obtenir un réseau mesh, communiquant les mesures en MQTT via un broker Mosquito à un serveur comme Telegraph, qui les stocke dans une base InfluxDB, elle-même lue par une application web comme Grafana... Ceci est assez typique et le sujet à fait l'objet de précédents articles dans le magazine (numéro 26 et 27). Bref, on passe du petit projet innocent à une vraie cathédrale !

Mais qu'en est-il, si l'on ne veut qu'une seule sonde et ne pas déployer toute une flottille ? MQTT, Telegraph, Grafana... pour un malheureux graphique, cela fait beaucoup de travail, d'investissement et de maintenance. Lorsqu'on sait qu'un simple ESP8266 peut parfaitement supporter un serveur web, la seule chose qui manque à l'appel est finalement la capacité à produire des graphiques présentables sur une page web, sans base de données, sans PC/RPi et sans PHP. Ça tombe bien, cette solution existe, elle s'appelle *Chart.js* et repose sur JavaScript, un langage interprété par tous les navigateurs web.

Les clones de Wemos D1 Mini sont incontestablement mes modules ESP8266 préférés. C'est un excellent compromis entre coût, qualité, nombre de GPIO utilisables, encombrement et modularité. C'est l'un de ces modules qui sera utilisé ici.

1. CE QUE NOUS ALLONS METTRE EN PLACE

Notre idée sera ici de tout réunir sur l'ESP8266 et de tenter de ne reposer que de façon minimale sur une éventuelle connexion Internet. Ce point pourra parfaitement être éliminé pour obtenir quelque chose de totalement déconnecté, à condition d'ajouter un module. Matériellement justement, notre projet sera constitué d'un ESP8266 sous la forme d'un clone de Wemos D1 mini et d'un module BME280 incorporant non seulement le composant Bosch, mais aussi l'électronique complémentaire. La liaison entre l'ESP8266 et le module se fera en i2c à l'aide de 4 connexions (masse, alimentation, SCL et SDA). Comme nous le verrons plus loin, la dépendance à Internet n'existe que pour obtenir la date et l'heure courantes en NTP depuis un serveur public. En ajoutant un module RTC DS1337 ou DS3231, également en i2c, nous n'aurions plus besoin de connexion Internet.

Les choses sont relativement simples côté matériel, mais quelque peu plus étoffées sur l'aspect logiciel. Notre croquis aura pour tâche de régulièrement procéder à la lecture du capteur et stocker ces informations, accompagnées de l'heure courante. Le support de stockage utilisé sera la mémoire SPIFFS utilisable avec tous les ESP8266, comme nous l'avons étudié dans le numéro 28. Cette zone de mémoire, présente dans la flash

associée au composant, qui n'est pas utilisable pour stocker le croquis, contiendra également les données nécessaires à la page HTML que nous présenterons dans le navigateur.

Un grand avantage que j'avais sommairement évoqué, concernant SPIFFS, est le fait de pouvoir stocker des fichiers en vue de s'en servir comme éléments HTML. Ici, nous exploiterons au maximum cet avantage, car les fonctionnalités dont nous avons besoin sont celles utilisées sur des serveurs web « classiques ». Nous allons donc pouvoir utiliser SPIFFS exactement comme le répertoire d'un serveur web et y stocker les pages HTML, les scripts JavaScript, mais aussi les données.

À ce propos justement, il est inconcevable d'envisager l'utilisation de quelque chose se rapprochant d'une base de données sur un ESP8266. Nous nous tournerons donc vers un stockage des mesures dans un fichier format en CSV. Il s'agit d'un simple fichier texte constitué de lignes, elles-mêmes divisées en colonnes séparées par un point-virgule. CSV signifie *Comma-Separated Values*, « valeurs séparées par des virgules » en français et est généralement utilisé pour échanger des données entre tableurs incompatibles. Il est amusant de noter que bien que l'acronyme CSV sous-entende l'utilisation de virgules, le séparateur utilisé n'est absolument pas standardisé, et heureusement. En effet, en France par exemple, la virgule est le caractère utilisé comme séparateur décimal en lieu et place du point britannique utilisé par les anglophones. Elle ne peut donc pas être utilisée comme séparateur de colonne et est alors remplacée dans cet usage par un point-virgule. On parle cependant toujours de « fichiers CSV ».

Vous l'aurez sans doute compris, nous allons devoir travailler sur deux codes : d'une part le croquis Arduino pour ESP8266, chargé de la collecte de données et du fonctionnement du serveur web et de l'autre, un script JavaScript embarqué dans une page HTML, reposant sur quelques bibliothèques/classes tierces et chargé de lire le fichier CSV pour composer un graphique.

La composition de graphiques issus de mesures est applicable à tout types de données provenant de capteurs. Mais l'utilisation qui vient immédiatement à l'esprit est sans le moindre doute la relève de mesures de température, d'humidité relative et de pression, comme celles fournies par ce capteur Bosch BME280.

2. HTML ET JAVASCRIPT

Nous commencerons par la partie HTML/JS pour plusieurs raisons. La première est que ce code est statique, nous allons construire une page web qui ne changera pas, mais qui contient un script chargé de récupérer le fichier CSV qui, lui, a un contenu évoluant dans le temps. Du point de vue de l'ESP8266, les fichiers qui constituent ces pages ne sont que des données, c'est le navigateur utilisé pour accéder à cette page qui va exécuter le code JavaScript et non l'ESP8266. En ce sens, ces données seront placées en SPIFFS via un sous-répertoire `data` du répertoire contenant le croquis Arduino. Il vous faudra, bien entendu, avoir installé le support SPIFFS dans votre environnement Arduino (voir Hackable 28 ou tout simplement **https://github.com/esp8266/arduino-esp8266fs-plugin**).

La seconde raison pour laquelle nous allons travailler sur la partie HTML est motivée par la facilité de mise au point. En effet, il ne s'agit que d'une page web et dans une certaine mesure, il est parfaitement possible de la peaufiner/tester en l'ouvrant simplement

localement avec votre navigateur. Une approche plus complète serait d'installer un serveur web léger (Lighttpd par exemple) ou même Apache sur votre machine de développement (ou une Pi) pour mettre au point le tout. Cette seconde option est celle que j'ai choisi, car elle permet de tester le traitement du fichier CSV (celui-ci est récupéré et analysé par le code JavaScript, ce qui ne fonctionnera pas pour une page ouverte depuis le disque dur, pour d'évidentes raisons de sécurité).

Pour notre page, nous aurons besoin de deux bibliothèques :

- *Chart.js* pour produire le graphique. Ceci est récupérable directement sur **https://github.com/chartjs/Chart.js/releases** sous la forme d'une archive ZIP, dont on ne conservera que les fichiers `Chart.bundle.min.js` et `utils.js` (pour la définition des couleurs).
- *Papa Parse* pour lire le fichier CSV et créer des tableaux (*arrays*) de valeurs en JavaScript. Cette bibliothèque très performante est téléchargeable sur **https://github.com/mholt/PapaParse/releases**. J'ai personnellement opté pour la version 4.6.0 (dernière stable au moment des expérimentations) et nous ne garderons de l'archive ZIP que le fichier `papaparse.min.js`.

La démarche de développement est assez classique, partant d'un des nombreux exemples présentés sur **https://www.chartjs.org/samples/latest** et étoffant le tout de fonctionnalités et ajustements nécessaires. Je ne reprendrai pas ici l'intégralité du contenu du fichier HTML qui est mis à disposition dans le dépôt GitHub du numéro (**https://github.com/Hackable-magazine/Hackable29**), mais me concentrerait simplement sur la logique de fonctionnement de *Chart.js* et les parties spécifiques non présentes dans l'exemple de départ.

La page HTML en tant que telle est excessivement simple et se résume à l'intégration des fichiers JavaScript, la définition de quelques styles et un unique objet :

```
<div id="container" style="width: 100%;">
  <canvas id="canvas"></canvas>
</div>
```

Ce *canvas* sera utilisé par le code JavaScript pour dessiner notre graphique. Mais avant cela, nous devons créer les tableaux de valeurs à partir des données CVS. Pour cela nous utilisons *Papa Parse* :

```
// analyse du fichier CSV
Papa.parse("data.csv", {
  download: true,
  header: true,
  skipEmptyLines: true,
  step: function(row) {
    // remplissage des array
    lab.push(row.data[0].label);
    temp.push(row.data[0].temp);
    hum.push(row.data[0].hum);
    pres.push(row.data[0].pression);
    // max 48 enregs
    if(lab.length > maxn) {
      lab.shift();
      temp.shift();
      hum.shift();
      pres.shift();
    }
  },
  complete: function() {
    console.log("All done!");
  }
});
```

C'est une utilisation tout à fait classique de la bibliothèque et nous récupérons les données du fichier `data.csv` se trouvant au même endroit que la page HTML elle-même (à la racine du site). La première ligne du fichier précise les intitulés des colonnes qui contiennent, dans l'ordre, le libellé (axe X), la température, le pourcentage d'humidité relative et la pression atmosphérique.

Exemple :

```
"label";"temp";"hum";"pression"
"05:58:12 21/01/2019";21.75;29.77;996.38
"06:58:12 21/01/2019";21.68;29.74;996.88
"07:58:12 21/01/2019";21.52;29.80;997.32
"08:58:12 21/01/2019";21.57;29.01;997.63
"10:27:47 21/01/2019";22.44;30.40;998.08
"11:27:47 21/01/2019";22.59;30.76;998.21
"12:27:46 21/01/2019";22.56;30.92;997.94
"13:27:47 21/01/2019";22.56;31.63;997.80
"14:27:47 21/01/2019";22.70;31.53;997.37
[...]
```

Nous utilisons la méthode `push()` pour ajouter les valeurs dans chaque tableau précédemment déclaré avec :

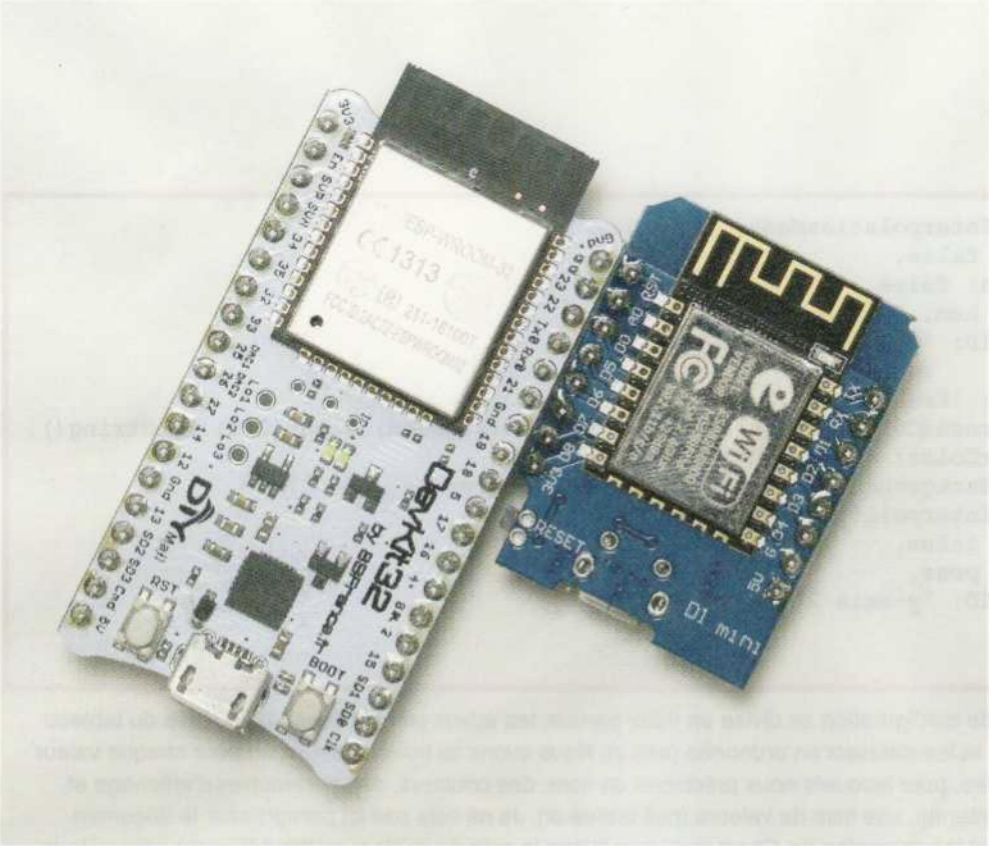
```
// Array pour le graphique
var lab = [];
var temp = [];
var hum = [];
var pres = [];
```

Nous devons cependant prendre en compte un point important. Même avec une mesure toutes les heures, le fichier CSV arrivera relativement rapidement à un volume de données, certes parfaitement gérable par *Papa Parse*, par SPIFFS et par la mémoire de la machine où s'exécute le navigateur, mais qui graphiquement représenté ne sera d'aucune utilité. Afin de ne considérer qu'un nombre réduit de valeurs à la fin du fichier CSV, nous utilisons une condition reposant sur la valeur retournée par la méthode `length()` sur le premier tableau. Ainsi, lorsqu'on arrive à `maxn`, l'ajout d'une valeur provoque automatiquement la suppression d'une valeur plus ancienne (un FIFO, *First In, First Out*).

D'où sort `maxn` ? D'un test vérifiant la présence d'un paramètre, passé sur l'URL utilisée pour accéder à l'ESP8266 :

```
function getQueryVariable(variable){
  var query = window.location.search.substring(1);
  var vars = query.split("&");
  for (var i=0;i<vars.length;i++) {
      var pair = vars[i].split("=");
      if(pair[0] == variable){return pair[1];}
  }
  return(false);
}

// paramètre ?
var maxn = getQueryVariable("maxn");
if(!maxn) maxn = 48;
```

Cette déclaration de fonction et son utilisation sont placées en tout début de script, juste après la déclaration des tableaux. Ceci nous permettra d'accéder à la page en utilisant l'adresse IP de L'ESP8266 pour n'avoir, par défaut, que les 48 dernières valeurs ou si nous



Les codes détaillés dans le présent article devraient être parfaitement applicables à une carte ESP32 (non testée). Néanmoins, la majeure partie du travail étant faite côté navigateur web, le gain en puissance de calcul sur ESP32 ne justifie pas la dépense.

utilisons une URL, comme « **http://adresse/index.html?maxn=123** » par exemple, les 123 dernières valeurs. Il est possible de traiter ce paramètre, appelé une *query string*, de bien d'autres manières et en particulier côté ESP8266, mais mon objectif ici était de garder le croquis Arduino le plus simple possible (il ne fait que servir des fichiers).

Une fois l'appel à *Papa Parse* utilisé, nous avons toutes les données à afficher présentes dans les tableaux `lab[]`, `temp[]`, `hum[]` et `pres[]`. Nous n'avons plus alors qu'à les utiliser pour configurer nos sources de données pour le graphique :

```
var color = Chart.helpers.color;
var barChartData = {
  labels: lab,
  datasets: [{
    label: 'Température',
    backgroundColor: color(window.chartColors.red).alpha(0.5).rgbString(),
    borderColor: window.chartColors.red,
    pointBackgroundColor: window.chartColors.red,
    fill: false,
    cubicInterpolationMode: 'monotone',
    hidden: false,
    data: temp,
    yAxisID: 'y-axis-1',
  }, {
    label: 'Humidité',
    backgroundColor: color(window.chartColors.blue).alpha(0.5).rgbString(),
    borderColor: window.chartColors.blue,
    pointBackgroundColor: window.chartColors.blue,
```

```
      cubicInterpolationMode: 'monotone',
      fill: false,
      hidden: false,
      data: hum,
      yAxisID: 'y-axis-1',
    }, {
      label: 'Pression',
      backgroundColor: color(window.chartColors.green).alpha(0.5).rgbString(),
      borderColor: window.chartColors.green,
      pointBackgroundColor: window.chartColors.green,
      cubicInterpolationMode: 'monotone',
      fill: false,
      data: pres,
      yAxisID: 'y-axis-2',
    }]
  };
```

Cette configuration se divise en deux parties, les *labels* sur abscisse (axe x) tirés du tableau `lab[]` et les *datasets* en ordonnée (axe y). Nous avons ici trois *datasets*, un pour chaque valeur mesurée, pour lesquels nous précisons un nom, des couleurs, des paramètres d'affichage et bien entendu, une liste de valeurs (nos tableaux). Je ne vais pas ici paraphraser la documentation et les exemples de *Chart.js* et vous laisse le soin de jouer avec les différents paramètres et leurs impacts sur le résultat graphique. Notez cependant, le paramètre `yAxisID` qui est différent pour les mesures de pression atmosphérique et nous permet d'obtenir un graphique avec deux échelles différentes en ordonnée (températures et hygrométrie sont dans les mêmes marges de variation, alors que la pression varie entre 950 hPa et 1050 hPa).

Enfin, nous arrivons à la création du graphique avec :

```
  window.onload = function() {
    var ctx = document.getElementById('canvas').getContext('2d');
    window.myBar = new Chart(ctx, {
      type: 'line',
      data: barChartData,
      options: {
        responsive: true,
        legend: {
          position: 'top',
        },
        title: {
          display: true,
          text: 'Station météo 1'
        },
        scales: {
          xAxes: [{
            type: 'time',
            time: {
              format: timeFormat
            }
          }],
```

```
            yAxes: [{
              type: 'linear',
              position: 'left',
              id: 'y-axis-1',
              },{
              type: 'linear',
              position: 'right',
              id: 'y-axis-2',
              gridLines: {
                drawOnChartArea: false,
              }
            }]
          },
          tooltips: {
            enabled: false,
            mode: 'index',
            position: 'nearest',
            custom: customTooltips
          }
        }
      });
    };
```

En dehors des éléments concernant l'aspect général du graphique comme l'emplacement des légendes et textes, les éléments notables sont :

- la configuration de l'axe x en type **time** permettant de produire un graphique où l'espace entre les points n'est pas linéaire, mais dépendant de la donnée temporelle. Ceci nous permet d'accumuler indifféremment des mesures sur des périodes différentes (quarts d'heure, heures, jour, etc.) ;
- pour l'axe y on prend soin de différencier les deux **id** spécifiées précédemment, avec les données de pression figurant à droite ;
- un format de *tooltips* personnalisé est spécifié afin d'obtenir l'affichage d'une petite bulle, résumant les trois valeurs au passage de la souris sur n'importe quel point du graphique. Cette configuration est placée dans un fichier **mytooltips.js** chargé avec l'ensemble des autres scripts en début de fichier.

Notez également **timeFormat** sur l'axe x, précisant le format de date utilisé et déclaré en début de script avec :

```
var timeFormat = 'HH:mm:ss DD/MM/YYYY';
```

Si vous passez un peu de temps sur la page de démonstration de *Chart.js*, vous constaterez qu'il y a énormément de choses à personnaliser sur un tel projet. Notre objectif ici était d'obtenir quelque chose de présentable et relativement fini (et parfaitement satisfaisant, en ce qui me concerne), mais surtout de jeter les bases d'une telle page pour que vous puissiez ensuite l'étoffer avec vos propres préférences et fonctionnalités.

Même si nous avons ici décidé de représenter graphiquement trois mesures en même temps, on peut tout à fait se limiter à un seul type de données. Un simple capteur de température, comme ce DS18b20 interfacé en bus 1-wire et donc utilisable sans problème avec un ESP8266, fera alors parfaitement l'affaire.

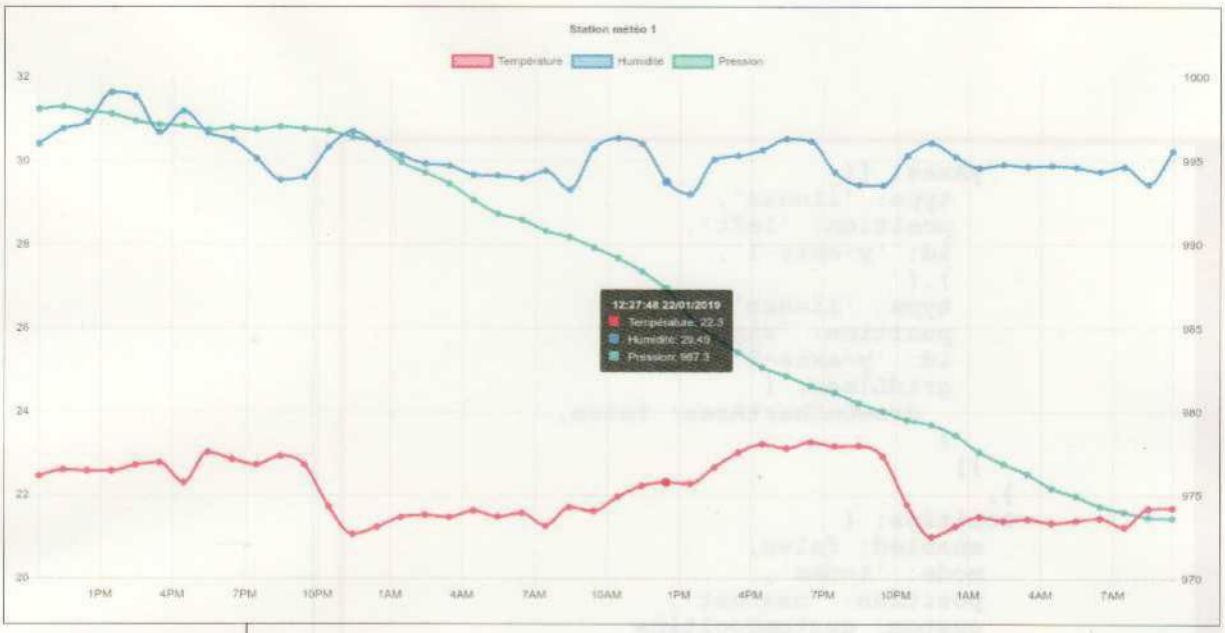


Chart.js permet de réaliser des graphiques d'excellente qualité en JavaScript. Ici, nous avons choisi une représentation avec des courbes, mais d'autres types sont à votre disposition. Notez le tooltip présentant les données numériques au survol de la souris. Cela aussi est totalement personnalisable.

Un dernier point sur les données à placer en SPIFFS concerne le fichier CVS. Nous partirons du principe que ce fichier se fera compléter régulièrement lorsque l'ESP8266 est sous tension. De ce fait, le croquis ne va pas gâcher des lignes de codes à tester la présence du fichier ou encore à éventuellement le créer. Dans le répertoire `data` qui contiendra donc les éléments du serveur web, nous allons trouver les fichiers HTML et JavaScript, mais également un `data.csv` dans sa forme la plus simple, constituée d'une seule ligne (terminée par un caractère LF `0x0A`) :

```
"label";"temp";"hum";"pression"
```

3. LE CROQUIS ARDUINO ESP8266

Tout l'aspect HTML et JavaScript étant traité, nous pouvons passer à la création du croquis pour notre ESP8266. Une fois n'est pas coutume, étant donnée la taille de l'article, je passerai ici sous silence les éléments « standards » du croquis, comme la connexion au point d'accès et l'éventuelle configuration de la mise à jour OTA. Notez cependant que cette dernière fonctionnalité implique automatiquement l'utilisation de mDNS, vous permettant d'accéder à l'ESP8266 par son nom et non uniquement via son adresse IP, depuis un navigateur (pour les plateformes compatibles). Bien entendu, le croquis mis à disposition sur le dépôt GitHub du magazine est complet et couvre donc ces fonctionnalités.

Nous commençons classiquement par inclure tout le nécessaire et définir quelques macros utiles pour la suite :

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <FS.h>
#include <Wire.h>
#include <Adafruit_BME280.h>
#include <NtpClientLib.h>
```

```
#define MSECOND  1000
#define MMINUTE  60*MSECOND
#define MHOUR    60*MMINUTE
#define MDAY     24*MHOUR
```

Nous passons ensuite aux déclarations, avec l'objet représentant notre serveur HTTP et celui permettant d'accéder au capteur Bosch (ainsi que la classique variable pour stocker la valeur de `millis()`) :

```
ESP8266WebServer server(80);
Adafruit_BME280 bme;

unsigned long previousMillis = 0;
```

Le reste se passe dans la fonction `setup()`, où nous commençons par initialiser le système de fichiers SPIFFS :

```
if(!SPIFFS.begin()) {
  Serial.println("Erreur initialisation SPIFFS");
}
```

Ceci fait, nous pouvons désormais accéder aux fichiers qui y sont stockés et associer les différentes URL à chacun d'eux, puis finir par démarrer le serveur HTTP :

```
server.serveStatic("/", SPIFFS, "/index.html");
server.serveStatic("/index.html", SPIFFS, "/index.html");
server.serveStatic("/Chart.bundle.min.js", SPIFFS, "/Chart.bundle.min.js");
server.serveStatic("/data.csv", SPIFFS, "/data.csv");
server.serveStatic("/mytooltips.js", SPIFFS, "/mytooltips.js");
server.serveStatic("/papaparse.min.js", SPIFFS, "/papaparse.min.js");
server.serveStatic("/utils.js", SPIFFS, "/utils.js");
server.begin();
```

Ainsi, les bibliothèques de l'ESP8266 se chargent automatiquement de gérer les requêtes et d'y répondre, sans que nous ayons à écrire de fonctions *callback*. C'est la façon la plus simple de servir des fichiers statiques sur la plateforme et aussi la raison pour laquelle nous gérons la *query string*, éventuellement présente dans l'URL, directement en JavaScript.

Nous passons ensuite à l'initialisation du bus i2c, en spécifiant respectivement les broches utilisées pour les signaux SDA et SCL et vérifions que le capteur est initialisé et accessible :

```
Wire.begin(D2, D1);
if (!bme.begin(0x76)){
  Serial.println(F("Erreur BME280"));
}
```

Notez que si vous souhaitez remplacer l'utilisation d'un serveur NTP pour obtenir l'heure, ici est un bon endroit pour placer l'initialisation d'un module RTC DS1337, également interfacé en i2c.

Dernier point concernant la fonction `setup()`, nous devons initialiser le support de NTP qui, je le rappelle, est un protocole de synchronisation d'horloge. La bibliothèque utilisée, *NtpClientLib*,

tout comme celle pour le capteur Bosch, est installable directement depuis le gestionnaire de bibliothèques. Nous précisons en argument de la méthode `begin()` le serveur NTP à utiliser, notre fuseau horaire (GMT+1), la prise en charge de l'heure d'été/hiver et un décalage temporel à appliquer (ici aucun) :

```
// configuration NTP
NTP.begin("europe.pool.ntp.org", 1, true, 0);
```

Dès cet appel effectué, la bibliothèque se chargera de maintenir la date et l'heure à jour régulièrement. Pour une utilisation plus avancée, des exemples relativement explicites sont livrés avec la bibliothèque.

Ceci conclu notre fonction `setup()` et nous pouvons alors nous pencher sur le cœur du croquis, qui consiste en la lecture des valeurs et la création d'une ligne dans le fichier CSV. Tout ceci prend la forme d'une fonction dédiée :

```
void adddata() {
  char strbuffer[64];
  snprintf(strbuffer, 64, "\"%s\";%.2f;%.2f;%.2f\n",
    NTP.getTimeDateString().c_str(),
    bme.readTemperature(),
    bme.readHumidity(),
    bme.readPressure()/100.0);
  Serial.println(strbuffer);
  File f = SPIFFS.open("/data.csv", "a+");
  if (!f) {
    Serial.println("erreur ouverture fichier!");
  } else {
      f.print(strbuffer);
      f.close();
  }
}
```

Nous commençons par créer un *buffer* destiné à accueillir les chaînes de caractères qui formeront la ligne du fichier, puis utilisons `snprintf()` pour formater toutes nos valeurs et les placer dans ce buffer. La méthode `getTimeDateString()` nous retourne une chaîne contenant la date et l'heure, mais les méthodes de notre objet `bme` retournent des `float`, ceci expliquant respectivement l'utilisation de `%s` et de `%.2f`. Nous tentons alors d'ouvrir le fichier `data.csv` en SPIFFS pour y ajouter la chaîne et en l'absence de problème, appliquons cet ajout avant de fermer le fichier.

Il ne nous reste plus alors qu'à nous occuper de `loop()` :

```
void loop() {
  char strbuffer[64];
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= MHOUR) {
    previousMillis = currentMillis;
    adddata();
  }
  // gestion serveur HTTP
  server.handleClient();
}
```

L'une des fonctionnalités intéressantes fournies par défaut par Chart.js concerne l'interaction avec l'utilisateur. Un simple clic sur une légende permet de masquer et réafficher une partie des données. L'échelle du graphique s'adapte automatiquement, sans rechargement de la page.

Nous reposons sur `millis()` pour régler l'intervalle séparant les enregistrements, en faisant usage des macros définies précédemment. Ici, une heure semble un bon compromis pour des mesures vaguement météorologiques ou environnementales. Bien entendu, selon vos besoins, il pourra être souhaitable de ramener cela à quelques minutes ou de l'étendre à plusieurs heures. Sur ESP8266, `millis()` retourne un *int32*, ce qui correspond, en millisecondes, à un « débordement » (*rollover*) au bout de presque 50 jours. De quoi largement satisfaire à tous types de besoins.

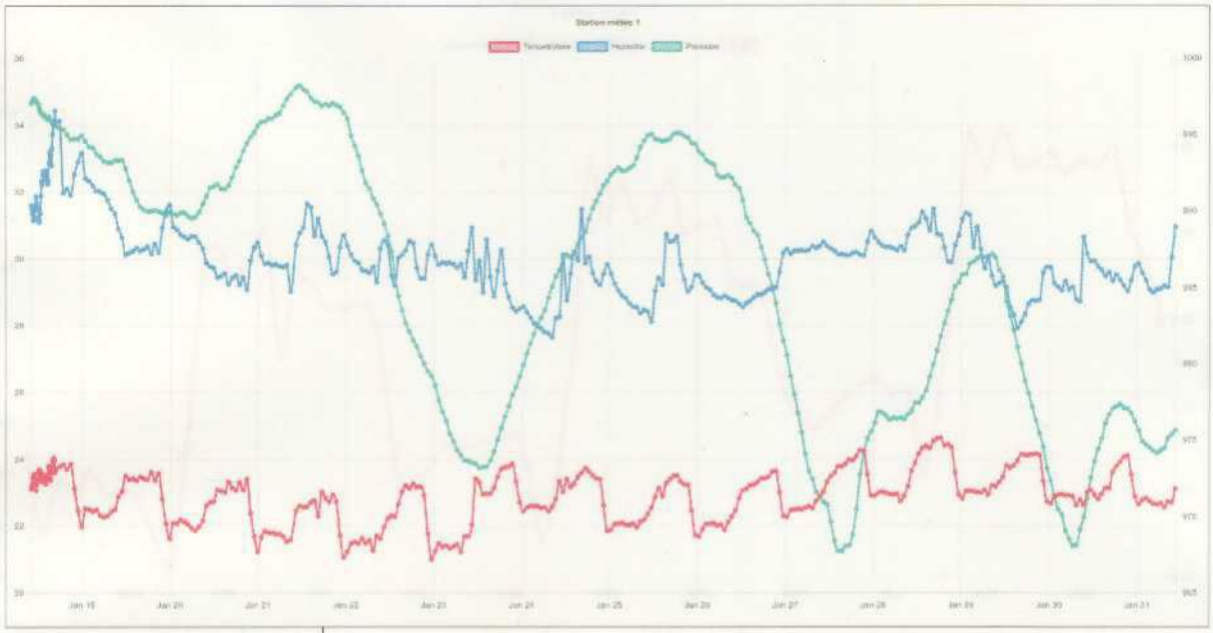
Afin d'assurer le fonctionnement du serveur HTTP, il convient de ne surtout pas oublier l'appel à `server.handleClient()` (ainsi qu'éventuellement à `ArduinoOTA.handle()`) pour assurer le traitement des requêtes. Notez que ce mécanisme interdit implicitement l'utilisation de `delay()` dans `loop()`.

4. ASSEMBLAGE, RÉSULTATS ET ÉVOLUTIONS

Une fois le croquis complété (et je l'espère, personnalisé par vos soins), il ne vous restera plus qu'à tout installer sur votre ESP8266. Mieux vaut commencer par charger les fichiers en SPIFFS avec le menu « Outil » et « ESP8266 Sketch data Upload » afin que tout soit présent pour la première exécution (fichier HTML et CSV). On pourra ensuite charger le croquis lui-même et le laisser commencer à collecter des données.

Au bout de quelque temps, le fait d'accéder à l'adresse IP (ou au nom d'hôte en cas d'utilisation d'OTA/mDNS) provoquera l'affichage du graphique, plus ou moins étoffé.

Notez qu'à des fins de test, rien ne vous interdit de remplir le fichier CSV de données factices, qui se verront complétées au fil du temps par de réelles valeurs. Il vous est également possible de changer la fréquence de mise à jour, même si le fichier contient déjà des données. Enfin, en cas de modification du contenu SPIFFS, sachez que le fichier CSV pourra très facilement être récupéré pour éviter d'en perdre le contenu. Il vous suffira de pointer votre navigateur sur l'adresse de l'ESP8266 en



Quelques lignes de JavaScript nous permettent de prendre en charge un paramètre placé dans l'URL. Celui-ci nous permet de spécifier une quantité de points maximum à considérer et donc, d'afficher nos données sur une période d'une durée arbitrairement choisie.

ajoutant « /data.csv » en fin d'URL et votre navigateur vous proposera le téléchargement (ou affichera le contenu). Ce fichier pourra alors être copié dans le répertoire **data/** du croquis, avant un nouveau chargement en SPIFFS.

En termes d'évolution de ce projet, je dois avouer que je suis relativement satisfait du résultat ici présent. Cette problématique de mise en place d'une sonde environnementale m'a été suggérée par un lecteur ne souhaitant pas déployer toute une infrastructure pour ce qui serait, au final, une « station » météo autonome. Je pense que ce que nous avons là répond assez justement à ce genre de besoins, sachant que le graphique s'affiche tout aussi convenablement sur un PC que sur un smartphone.

Bien entendu, des améliorations restent toujours possibles pour qui veut peaufiner tout cela. On pourrait ainsi envisager de passer via l'URL des paramètres temporels de début et de fin, pour obtenir un graphique sur une plage choisie. Le code JavaScript devra alors analyser la chaîne dans la première colonne du tableau CSV pour intégrer ou non les valeurs obtenues du fichier.

Une autre évolution concerne la densité des informations sur le graphique. Avec une relève toutes les 10 minutes par exemple, l'ensemble deviendrait relativement lourd et illisible, en tentant de tracer les 10 ou 15 derniers jours. Il serait alors intéressant d'ajouter une fonction d'agrégation ou éventuellement de simplement sauter des valeurs intermédiaires, en fonction du nombre de points à afficher. Là aussi, c'est dans le code JavaScript de lecture du fichier qu'il faudrait intervenir.

Enfin, un énorme travail de personnalisation peut être réalisé au niveau du design graphique. *Chart.js* offre énormément d'options de présentation des données et un nombre impressionnant de fonctionnalités de toutes sortes. Nous n'avons fait ici qu'effleurer la surface en personnalisant les *tooltips*, mais il est possible de faire bien plus... **DB**

PROGRAMMATION OBJET ? NON ! CODAGE DES OBJETS AVEC OPENSCAD

Jean-François Rocchini

Depuis quelques années, imprimantes 3D, mais aussi découpeuses laser, fraiseuses CNC, voire découpeuses jet d'eau nous sont accessibles. Ces machines nous permettent de fabriquer facilement et pour un coût raisonnable des objets d'excellente facture. Cependant, il reste la première étape, c'est-à-dire la conception et le dessin, qui passe par l'utilisation d'un logiciel de CAO-DAO. J'ai toujours eu une aversion vis-à-vis de ce type de logiciel, dont les écrans truffés de petites icônes s'apparentent à ceux d'un avion long-courrier. Fort heureusement, il existe une alternative nous permettant, à nous autres pauvres programmeurs, de concevoir et de dessiner des objets en 2D ou en 3D. Il s'agit d'OpenSCAD.

OpenSCAD en deux mots, c'est un pseudo langage et un environnement intégré. Nous le voyons sur la Figure 1.

Cet environnement est composé de 3 parties :

- une partie gauche, l'éditeur de texte, dans lequel nous allons taper les instructions ;
- une partie en haut à droite, dans laquelle s'affichera le dessin 2D ou 3D produit ;
- et enfin, en bas à droite une partie sur laquelle le vérificateur de syntaxe affichera si besoin est les erreurs commises, ainsi que d'éventuelles informations de débogage que vous aurez placé dans votre code.

Un script OpenSCAD est, en fait, une succession d'instructions qui permettent de décrire :

- des objets géométriques élémentaires (2D ou 3D) ;
- leur position dans un espace 2D ou 3D ;
- puis de les faire interagir, afin de que nous puissions concevoir l'objet complexe voulu.

OpenSCAD est bien entendu libre, gratuit et multi-plateforme. Il est inclus dans certaines distributions de Linux et en tout cas disponible en téléchargement sur le site **www.openscad.org**.

1. LA 3D : CONCEPTION ET DESSIN

J'ai choisi de commencer par la conception 3D, c'est à dire la conception dans le but de réaliser l'objet conçu avec une imprimante 3D. En effet, la conception en 3D est plus simple à appréhender que la conception de 2D, puis 3D dans le but de fabriquer un objet à l'aide d'une machine 2D : découpeuse laser, jet d'eau, etc.

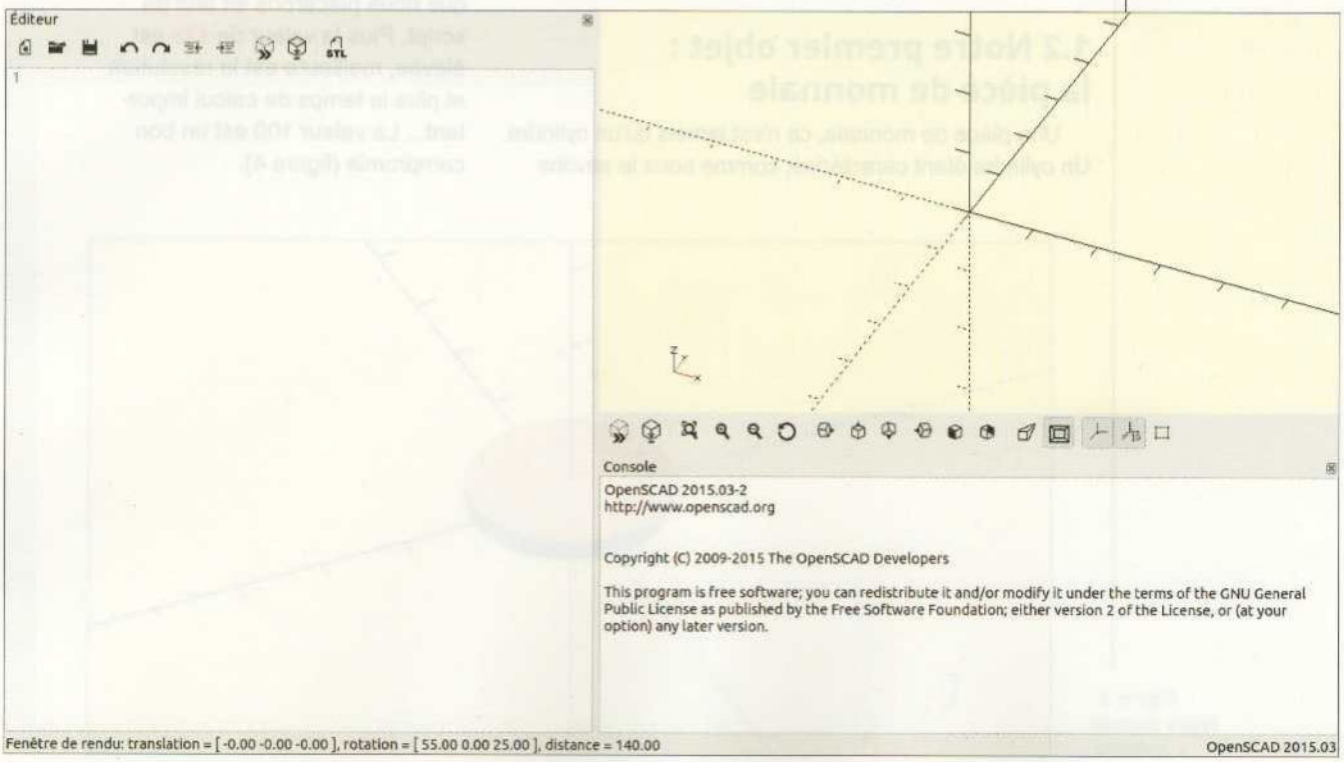


Figure 1 : Vue générale de l'EDI OpenSCAD.

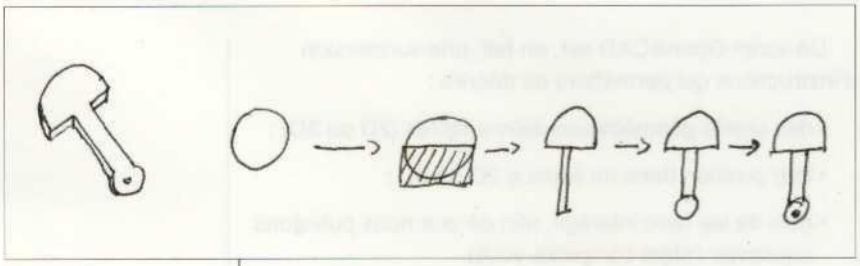


Figure 2 : Vue schématique de la clé et des différentes étapes de réalisation.

1.1 Conception 3D

Afin de ne pas rendre cet article trop aride, nous allons réaliser ensemble étape par étape un petit objet bien utile, qui vous rendra populaire parmi votre famille et vos amis. Nous allons faire une clé qui nous permettra, même en l'absence de pièce, de récupérer un caddy de supermarché ! Et en plus, cette clé est amovible, donc pas de risque de l'oublier dans le caddy.

Je tiens à préciser que je ne suis pas l'inventeur de cet objet sympathique, dont il existe de multiples déclinaisons sur le net.

La conception, ça commence bien souvent par un petit croquis sur un vague bout de papier, voir figure 2.

1.2 Notre premier objet : la pièce de monnaie

Une pièce de monnaie, ce n'est jamais qu'un cylindre. Un cylindre étant caractérisé, comme nous le savons tous, par un diamètre et une hauteur. Pour une pièce de 1 euro : le diamètre est de 23,25 mm et la hauteur de 2,33 mm.

Nous tapons dans l'éditeur de l'EDI :

```
cylinder(d=23.25,h=2.33) ;
```

Attention au « ; » en fin de ligne, mais cela ne doit pas choquer les programmeurs C !

En pressant la touche F5 de notre clavier, nous voyons apparaître notre cylindre (Figure 3).

Certes, mais il est moche... Eh oui, mais nous pouvons améliorer la résolution à l'aide de l'instruction suivante :

```
$fn=100 ;
```

que nous placerons en tête de script. Plus la valeur de `$fn` est élevée, meilleure est la résolution et plus le temps de calcul important... La valeur 100 est un bon compromis (figure 4).

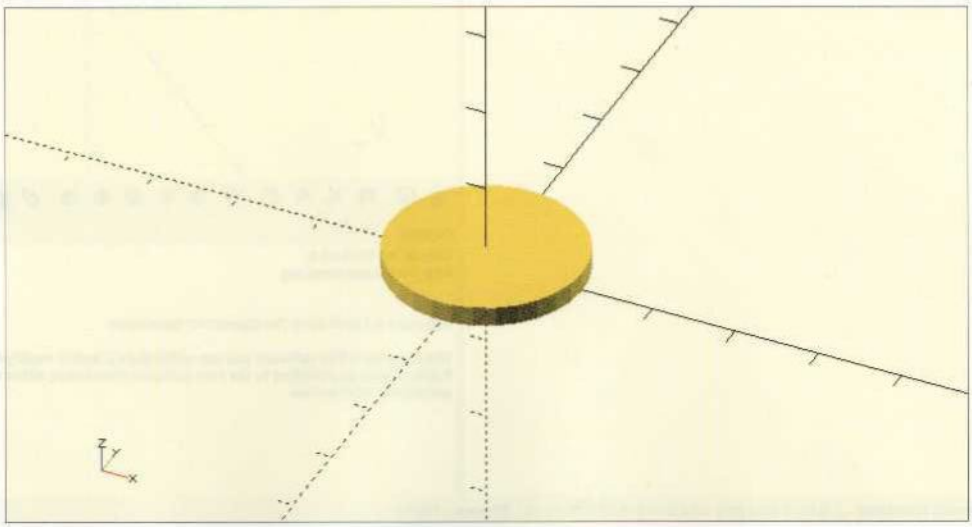


Figure 3 : Notre premier cylindre.

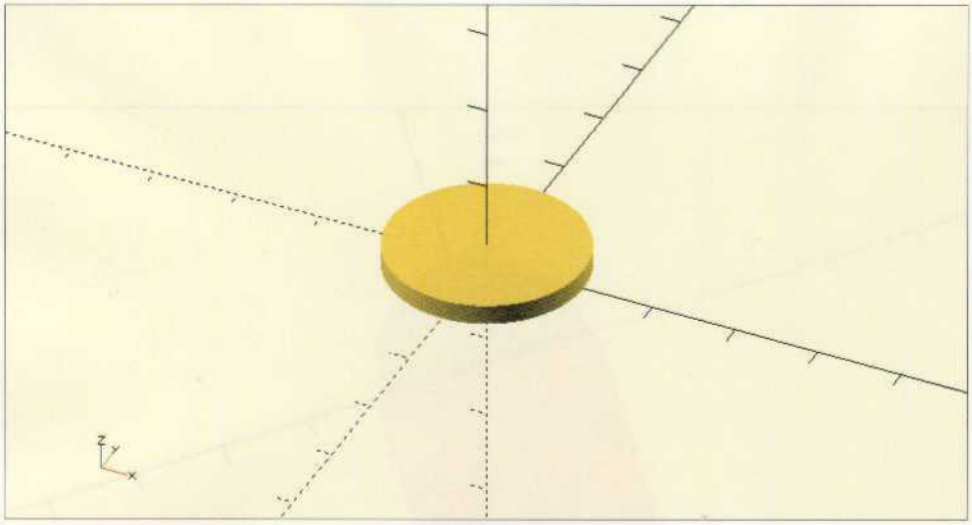


Figure 4 : Notre cylindre avec une meilleure résolution.

1.3 Coupons-la en 2 !

Il nous faut seulement un demi-cylindre. Voyons comment nous pouvons le couper en 2. Pour ce faire, nous allons créer un masque qui viendra se soustraire à notre cylindre. Ce masque sera un parallélépipède rectangle, mais qui est appelé « cube » avec 3 valeurs de longueur d'arête différentes. Que le Dieu (ou le Diable) des maths nous pardonne !

C'est l'opérateur `difference` qui va nous permettre cela :

```
$fn=100;
difference(){
      cylinder(d=23.25,h=2.33);
      cube([25,25,4]);
}
```

Les objets qui suivent le premier objet de la liste viennent en soustraction de celui-ci. C'est bien beau, mais mon masque (largement dimensionné) est mal positionné (figure 5). En effet, par défaut les cylindres sont centrés sur l'origine, tandis que c'est le coin bas gauche des parallélépipèdes rectangles qui y est positionné.

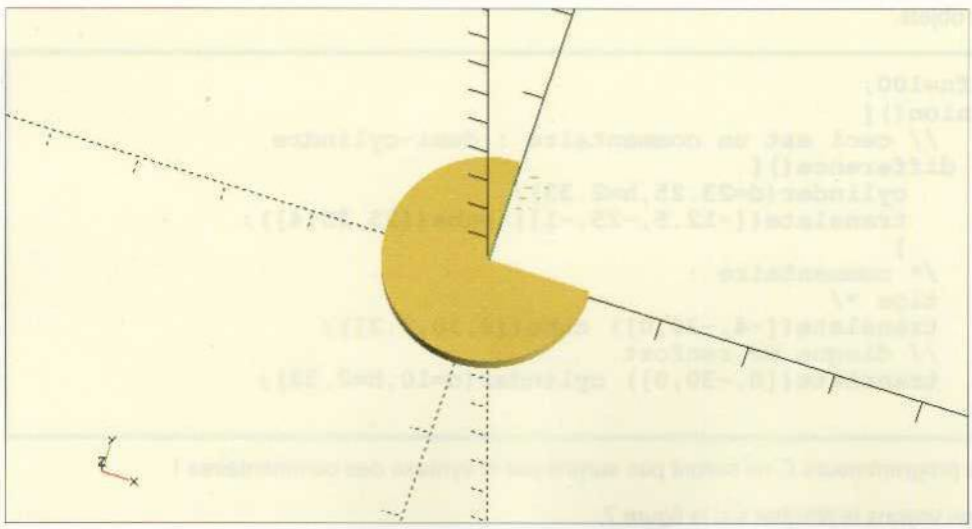


Figure 5 : Le masque n'est pas bien placé.

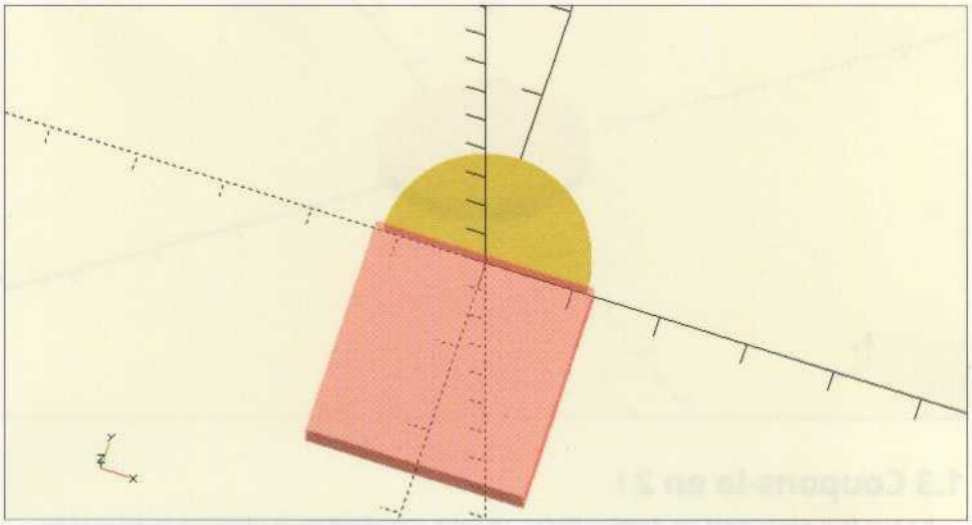


Figure 6 : Le masque est placé correctement. Il est visible grâce au caractère # placé avant.

Il faut déplacer notre masque grâce à l'instruction `translate`.

Les masques sont bien entendu invisibles, mais dans un but de débogage si le caractère « # » est placé juste avant le masque, celui-ci apparaît en mauve translucide.

```
$fn=100;
difference(){
  cylinder(d=23.25,h=2.33);
  translate([-12.5,-25,-1]) #cube([25,25,4]);
}
```

1.4 Et on ajoute la tige et son disque de renfort

Cette tige n'est autre qu'un parallélépipède rectangle que l'on va déplacer, puis fusionner avec notre demi-disque. Quand à son renfort il s'agit d'un disque d'épaisseur identique à celle de la demi-pièce et de la tige. L'opérateur « union » permet la fusion en une seule entité de plusieurs objets.

```
$fn=100;
union(){
   // ceci est un commentaire : demi-cylindre
  difference(){
     cylinder(d=23.25,h=2.33);
     translate([-12.5,-25,-1]) cube([25,25,4]);
     }
   /* commentaire :
   tige */
   translate([-4,-30,0]) cube([8,30,2.3]);
   // disque de renfort
   translate([0,-30,0]) cylinder(d=10,h=2.33);
}
```

Les programmeurs C ne seront pas surpris par la syntaxe des commentaires !

Nous voyons le résultat sur la figure 7.

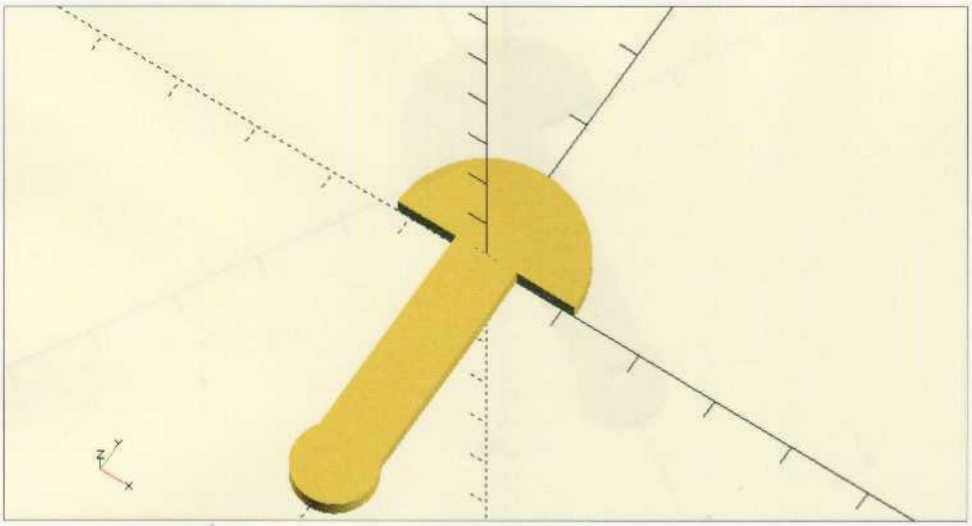


Figure 7 : La tige et le renfort de tige sont ajoutés.

1.5 Perçage du disque de renfort

Comme nous l'avons vu précédemment, pour percer un trou, il suffit de soustraire un cylindre à une forme. Mais encore faut-il le faire au bon endroit !

Si j'exécute le code suivant :

```
$fn=100;
union(){
  difference(){
    cylinder(d=23.25,h=2.33);
    translate([-12.5,-25,-1]) cube([25,25,4]);
    // trou sur le renfort
   translate([0,-30,-1]) #cylinder(d=3,4.3);
}
  translate([-4,-30,0]) cube([8,30,2.33]);
  translate([0,-30,0]) cylinder(d=10,2.33);
}
```

On n'obtient pas de trou sur le renfort, voir figure 8 !

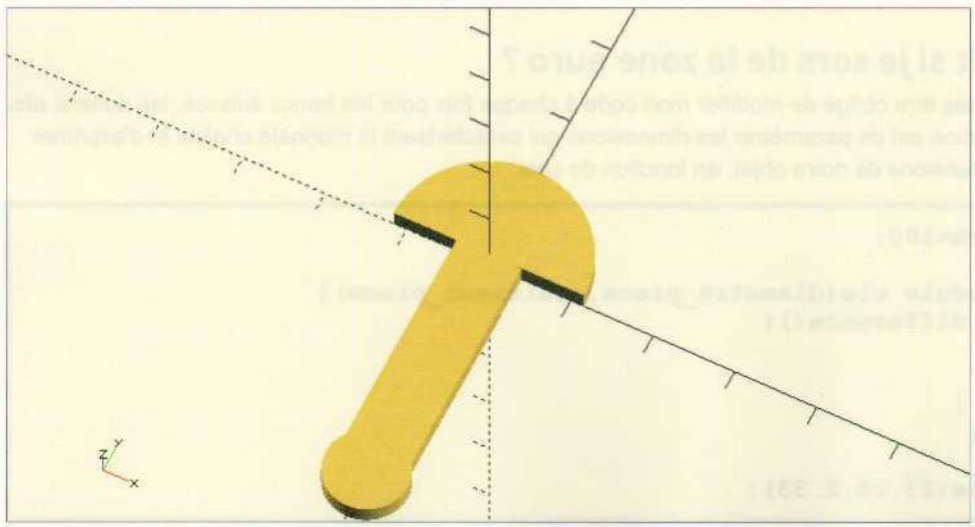


Figure 8 : Zut ! Pas de trou dans le renfort !

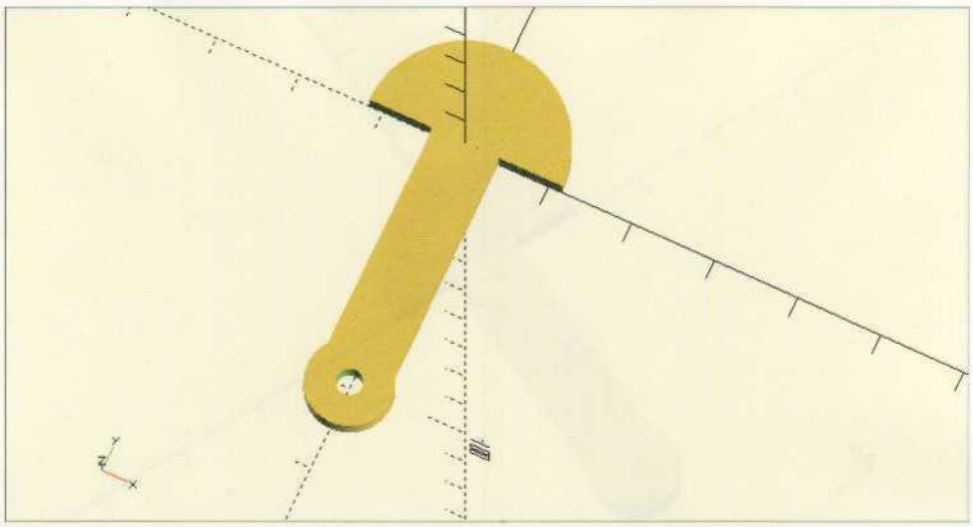


Figure 9 : Ouf c'est bon. La pièce est correcte.

Que s'est-il passé ? Et bien, nous avons voulu percer un trou dans un objet qui n'avait pas encore été créé !

Exécutons le code suivant :

```
$fn=100;
difference(){
   union(){
     difference(){
       cylinder(d=23.25,h=2.33);
       translate([-12.5,-25,-1]) cube([25,25,4]);
     }
     translate([-4,-30,0]) cube([8,30,2.33]);
     translate([0,-30,0]) cylinder(d=10,2.33);
   }
   // code du trou dans le renfort
   translate([0,-30,-1]) cylinder(d=3,4.3);
}
```

Et là c'est bon ! Un mauvais ordre d'emboîtement des `union` et des `difference` est une erreur fréquente chez les débutants (Figure 9)...

1.6 Et si je sors de la zone euro ?

Je vais être obligé de modifier mon code à chaque fois pour les francs suisses, les dollars, etc. La solution est de paramétrer les dimensions qui caractérisent la monnaie choisie et d'exprimer les dimensions de notre objet, en fonction de cela.

```
$fn=100;

module cle(diametre_piece,epaisseur_piece){
  difference(){
.
.
  }
 }

cle(23.25,2.33);
```

Nous avons regroupé notre code en un module, qui est l'analogue d'une procédure dans d'autres langages. Nous passons le diamètre et l'épaisseur en paramètre, lors de l'appel du module.

Si je veux créer une clé en franc suisse pour mon oncle Walter, il me suffit d'appeler le module ainsi :

```
.
.
cle(23.2,1.55) ;
```

Figure 10 : Notre clé en place et prête à être utilisée.

1.7 Et si je veux en faire plusieurs ?

Il existe bien entendu la notion de boucle dans OpenSCAD.

```
$fn=100;

module cle(diametre_piece,epaisseur_piece){
.
.
}
for(i=[0:1:3]){
  translate([25*i,0,0]) cle(23.25,2.33);
}
```

Cette boucle nous permettra d'obtenir 4 clés régulièrement espacées. Ici, le paramètre i débute à 0, puis va jusqu'à 3 inclus, au pas de 1.

1.8 Et maintenant ?

Il ne nous reste plus qu'à presser la touche « F6 » afin de générer le dessin, puis exporter ce dessin sous forme de fichier « STL ». Il nous faudra enfin importer notre fichier STL dans le Slicer qui correspond à notre imprimante 3D, afin de pouvoir l'imprimer. Et obtenir notre clé, voir figure 10. Mais ceci est une autre histoire...

2. LA 2D

Le but de ce petit chapitre est de vous montrer ma façon de faire, afin de pouvoir générer des fichiers de découpe, mais aussi de vérifier la justesse de notre conception. Nous devrons donc pouvoir voir la ou les planches telles qu'elles seront découpées, mais aussi avoir une vue de l'ensemble assemblé en 3D.

2.1 Les figures « primitives » 2D - positionnement et interactions

Le cylindre et le parallélépipède rectangle en 3D auront leurs analogues en 2D : le cercle (`circle`) et le rectangle (`square`). Ils pourront, bien entendu, être repositionnés par `translate` et `rotate` et être modifiés par `union` et `difference`. Attention, on ne pourra faire d'union ou de différence qu'entre objets de même nature : 2D avec 2D et 3D avec 3D !

2.2 Passage de 2D à 3D : l'extrusion

L'opérateur `linear_extrude()` nous permet de faire passer un objet de 2D à 3D. L'épaisseur que l'on veut obtenir sera passée en paramètre.

```
linear_extrude(3) square(3) ;
```

nous permet d'obtenir un cube d'arête de longueur 3.

On peut procéder de même avec un objet complexe 2D codé sous forme de module.

2.3 Structure d'un script de conception et de découpage

La structure sera la suivante :

```
// parties de l'objet en 2D telles qu'elles seront découpées
module module1_2D(){
.
.
}
module module2_2D(){
.
.
}
.
.
// parties de l'objet en 3D telles qu'elles seront assemblées
module module1_3D(){
  linear_extrude(epaisseur) module1_2D() ;
}
module module2_2D(){
  linear_extrude(epaisseur) module2_2D() ;
}
// choix de la vue
vue=0 ;
// vue de l'assemblage en 3D
if (vue==0){
   translate([....]) rotate([...]) module1_3D() ;
   translate([....]) rotate([...]) module2_3D() ;
   .
   .
}
```

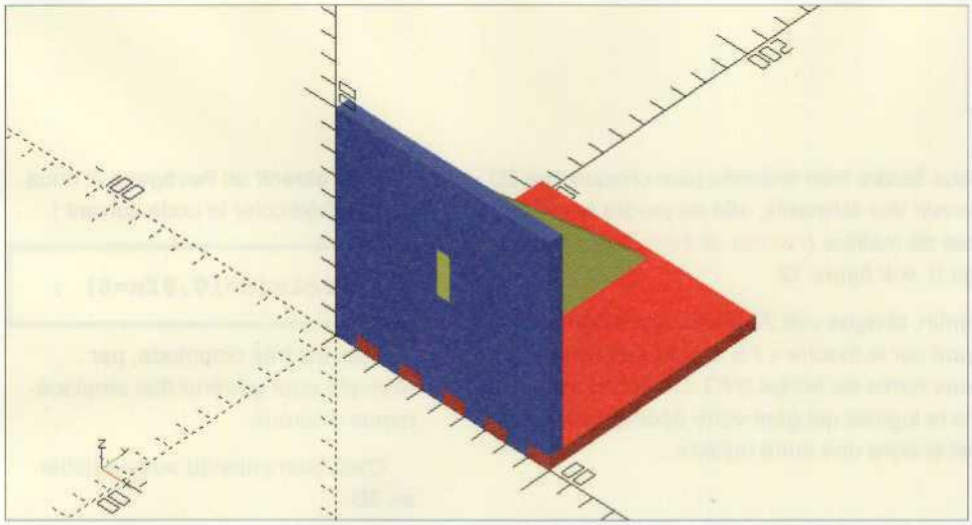


Figure 11 : Exemple de vue d'un assemblage.

```
// vue d'une planche de découpe en 2D
if (vue==1){
    translate([....]) rotate([...]) module1_2D() ;
    translate([....]) rotate([...]) module2_3D() ;
    .
    .
}
.
.
```

L'avantage de procéder ainsi est que chaque partie 3D sera directement dépendante de la partie 2D qui lui correspond. En cas de modification, nous conserverons la cohérence entre 2D et 3D. La vue 0 est, par habitude, la vue en 3D (voir figure 11).

Il peut, selon votre projet, y avoir plusieurs vues 2D. En effet, selon les capacités de votre découpeuse, il est possible que tous les éléments ne tiennent pas sur une seule planche. Vous pouvez aussi avoir besoin d'utiliser des éléments d'épaisseur et/ou de matériaux différents : par exemple, un boîtier en bois avec un couvercle en plastique...

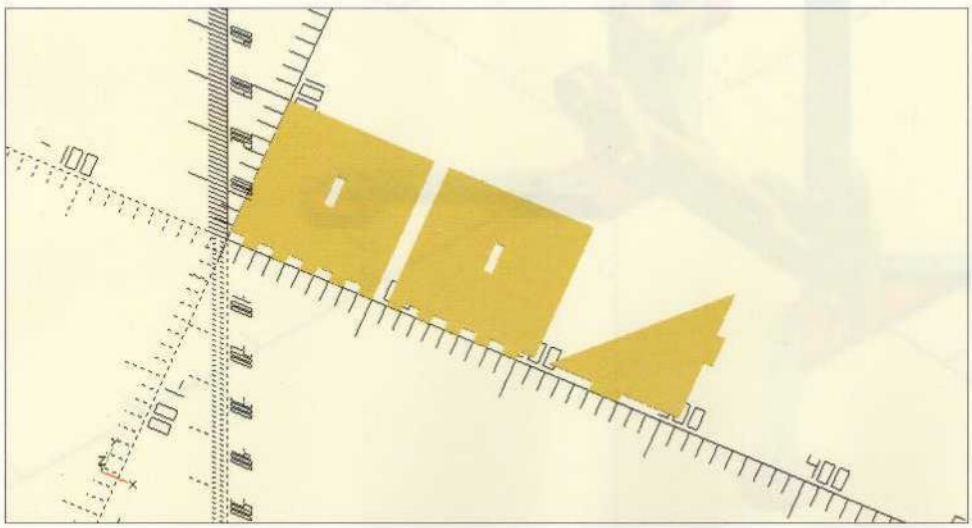


Figure 12 : Exemple de vue d'une planche telle qu'elle sera découpée.

Il vous faudra bien entendu pour chaque vue 2D positionner vos éléments, afin de perdre le moins possible de matière (`rotate` et `translate` sont fait pour ça !), voir figure 12.

Et enfin, chaque vue 2D sera « compilée » en appuyant sur la touche « F6 », puis sauvée en l'exportant sous forme de fichier SVG. Ce fichier sera importé dans le logiciel qui gère votre découpeuse. Mais ceci est encore une autre histoire...

3. QUELQUES TRUCS ET ASTUCES... ET MISE EN GARDE

3.1 Des objets complexes obtenus facilement

On peut, en faisant varier la valeur de `$fn`, faire dériver de façon intéressante les objets de base.

Pour obtenir un hexagone, il nous suffira d'exécuter le code suivant :

```
circle(d=10,$fn=6) ;
```

Ceci est très commode, par exemple pour générer des emplacements d'écrous.

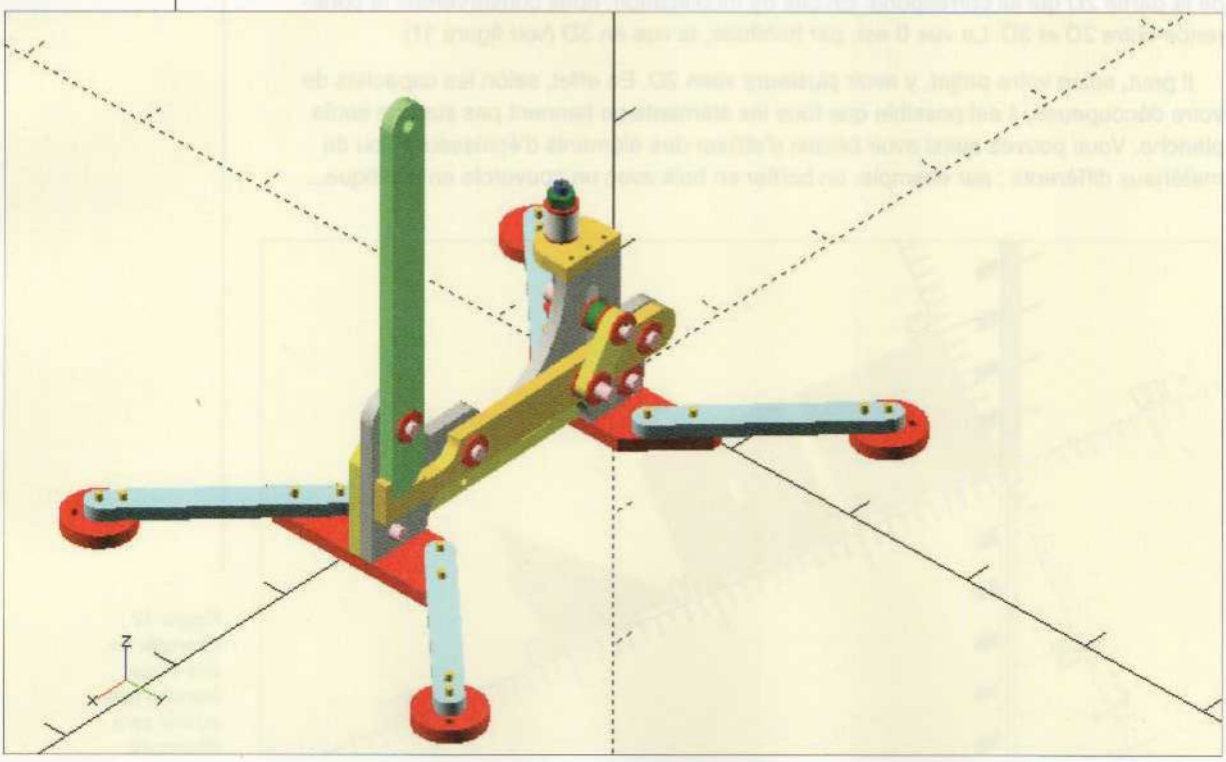
C'est bien entendu aussi valable en 3D :

```
cylinder(d=10,h=30,$fn=6) ;
```

3.2 Unités

OpenSCAD est adimensionnel. Mais il faut bien entendu rester cohérent dans le choix des unités. Il est cependant commode d'utiliser le millimètre. Il faudra faire attention,

Figure 13 : Base de fusée à eau.



lors de l'importation des fichiers STL ou SVG dans les logiciels pilotant les machines, que ceux-ci soient configurés dans la même unité.

3.3 Variables et constantes

J'ai gardé le plus déconcertant pour la fin.

À part les indices de boucle, OpenSCAD n'admet pas (ou du moins en première approximation) de variables, mais seulement des constantes. Donc, attention aux affectations dans les tests classiques: ça ne marche pas.

```
$fn=100;
a=3;
rayon=2;
if (a==3) { rayon=40 ; };
circle(r=rayon);
```

On obtient un cercle de rayon 2 !

Pour obtenir un cercle de rayon 40, il nous faut passer par la forme suivante :

```
$fn=100;
a=3;
rayon=2;
rayon= a==3 ? 40 : 2 ;
circle(r=rayon);
```

Un peu étonnant, mais on s'y fait !

CONCLUSION

J'ai voulu vous montrer, par le présent article, qu'il est aisé pour un programmeur de prendre en main rapidement OpenSCAD. Et d'arriver à concevoir et réaliser des objets y compris complexes, une base de fusée à eau par exemple (voir figure 13 ci-contre).

Donc ça y est, vous n'avez plus d'excuses pour ne pas vous y mettre ! JFR

CRÉEZ SIMPLEMENT VOTRE PÉRIPHÉRIQUE MATÉRIEL AVEC LE LANGAGE C

Patrice Kadionik - Maître de Conférences HDR à l'ENSEIRB-MATMECA

Cet article présente la création et la mise en œuvre d'un périphérique matériel libre sous Linux embarqué pour la carte ZedBoard, utilisant un circuit FPGA Zynq. La synthèse de haut niveau HLS, à l'aide du langage C, sera utilisée ici pour la création du périphérique matériel.

Nous avons décrit dans un précédent article de feu Open Silicium **[1]** la méthodologie de conception conjointe du matériel/logiciel ou *codesign*. Cette méthodologie permet de concevoir en même temps le matériel et le logiciel d'un système embarqué et elle est couramment utilisée pour le développement de système sur silicium ou SoC (*System on Chip*), notamment sur circuit FPGA ou SoPC (*System on Programmable Chip*).

L'usage d'un langage de description de matériel comme VHDL ou Verilog facilite le travail de synthèse, apportant in fine un niveau d'abstraction significatif par rapport à l'approche traditionnelle sous forme de schémas électroniques. Ces langages de description de matériel permettent de travailler au niveau de la fonctionnalité RTL (*Register Transfer Logic*) comme les registres, additionneurs, comparateurs...

Ils sont aujourd'hui concurrencés par d'autres langages pour une synthèse de haut niveau HLS (*High Level Synthesis*), plus proche de l'algorithme et offrant un niveau d'abstraction bien supérieur, pour mieux se focaliser sur la fonctionnalité et sur les flux d'échange de données à synthétiser.

On peut citer comme langages courants pour la synthèse HLS :

- C et C++.
- SystemC. C'est une bibliothèque d'extension de C++.

Lequel choisir ? Si l'on veut focaliser sur l'aspect algorithmique, l'emploi de C ou C++ est approprié. Si l'on veut par contre avoir un contrôle plus strict sur les aspects temporels ou autres flux de données, SystemC est plus approprié.

Cet article abordera l'usage du langage C, langage bien connu par celles et ceux qui font de l'embarqué. Bien sûr, les outils de synthèse HLS mettant en œuvre le langage C ou C++ permettent aussi un contrôle sur le design (*pipelining*, latence...) et dans une moindre mesure, sur les contraintes temporelles par l'emploi de directives de synthèse HLS.

Xilinx **[5]** propose des circuits FPGA puissants, intégrant un processeur *hardcore* de type ARM comme le circuit FPGA Zynq et des outils de synthèse performants. L'outil de synthèse Xilinx Vivado se prête bien au *codesign*, tandis que l'outil Vivado HLS est dédié à la synthèse de haut niveau.

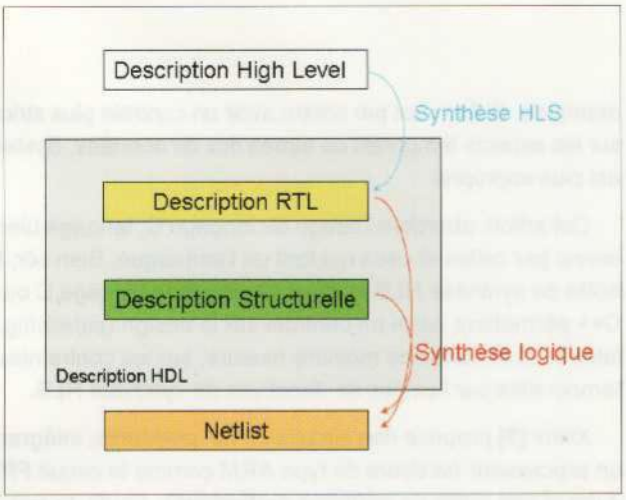
Cet article se propose d'aborder la synthèse HLS, en mettant en œuvre un circuit FPGA Zynq de Xilinx ainsi que ses outils Vivado HLS et Vivado.

On abordera ainsi les points suivants :

- Création d'un périphérique matériel (fonction mathématique) en langage C par l'approche HLS, simulation et cosimulation du bloc IP (*Intellectual Property*) avec l'outil Vivado HLS. Comparaison avec un développement en langage VHDL pur.
- Intégration du bloc IP dans un système SoPC pour un circuit FPGA Zynq avec l'outil Xilinx Vivado. La carte cible utilisée est la carte ZedBoard. Cette partie a déjà été décrite dans l'article **[3]** de *GNU/Linux Magazine* et nous n'en détaillerons ici que les résultats.
- Mise en œuvre de Linux embarqué sur le processeur ARM Cortex-A9 du circuit FPGA Zynq.
- Développement d'un pilote de périphérique (*driver*) sous Linux embarqué en mode utilisateur (*user mode driver*) pour le test du périphérique matériel.

1. LA SYNTHÈSE DE HAUT NIVEAU HLS

La synthèse de haut niveau HLS (*High Level Synthesis*) pour la conception de circuits numériques permet de travailler à un niveau d'abstraction bien supérieur qu'il est possible de faire avec l'approche RTL classique, en utilisant le langage VHDL par exemple. Cela permet au concepteur de s'affranchir des considérations d'implantation, pour ne focaliser que sur



Synthèse HLS vs synthèse logique.

sa fonctionnalité ou sur son algorithme. On partira ainsi d'une description de l'algorithme en langage informatique C, C++ ou SystemC et le synthétiseur HLS produira en sortie une description RTL de l'algorithme en langage VHDL ou Verilog, qui pourra être ensuite synthétisée avec un synthétiseur logique.

La figure ci-dessus décrit le processus de synthèse HLS.

Sur la figure précédente, la description structurelle est une description HDL de plus bas niveau qu'une description RTL, basée sur les opérateurs logiques élémentaires (et, ou, bascules...). Elle est délaissée au profit de la description RTL. La *netlist* permet enfin de générer le fichier de programmation du circuit FPGA.

La synthèse HLS est basée sur 2 opérations importantes :

- L'ordonnancement (*scheduling*) : cette opération permet la translation du code C en opérations RTL. Cela se traduit par la création de la suite des opérations sur les données (*data path*) et du contrôle des opérations (*control path*), généralement à l'aide d'une machine d'états. Les décisions faites à ce niveau sont affectées par le choix de la période d'horloge, le choix de la cible FPGA, mais aussi par les directives de synthèse HLS qu'applique le concepteur.
- La liaison (*binding*) : cette opération permet d'associer les opérations générées par le *scheduling* avec les ressources de la cible FPGA. Il y a aussi une rétroaction possible avec l'opération de *scheduling*.

La synthèse HLS va ainsi traiter 2 aspects importants d'un design :

- L'interface du design : cela concerne les connexions externes au niveau top-level. Il y aura donc une synthèse HLS de l'interface.
- L'algorithme du design : il y aura donc une synthèse HLS de l'algorithme.

Cette synthèse HLS est grandement influencée par 2 paramètres :

- Les contraintes : contraintes sur le *timing* (valeur de la période d'horloge, incertitude sur cette période), éléments technologiques du circuit FPGA (blocs RAM, blocs DSP, etc.)...
- Les directives : pipelining du design, parallélisme du design, choix d'un type d'interface pour le *design*...

L'outil Vivado HLS de Xilinx **[5]** reprend ces grands principes. Le flot de conception avec l'outil Vivado HLS est décrit sur la figure suivante.

Sur cette figure, on peut noter que l'on crée le design en utilisant le langage C (ou C++ ou SystemC). Un banc de test (*test bench*) écrit en langage C doit être développé et peut faire appel à des résultats de référence (*golden references*). Cela permet de simuler et de tester le code source C du *design*. Puis la synthèse HLS est réalisée, conformément aux directives et aux contraintes imposées par le concepteur.

À l'issue de la synthèse HLS, on récupère une description RTL du *design*. On peut alors réaliser une cosimulation C/RTL en 3 étapes :

1 - Le test bench C est exécuté pour créer les stimuli qui serviront à tester le design RTL.

2 - Un test bench RTL est automatiquement généré pour appliquer uniquement les stimuli précédents au design RTL et pour récupérer ensuite les valeurs de sortie.

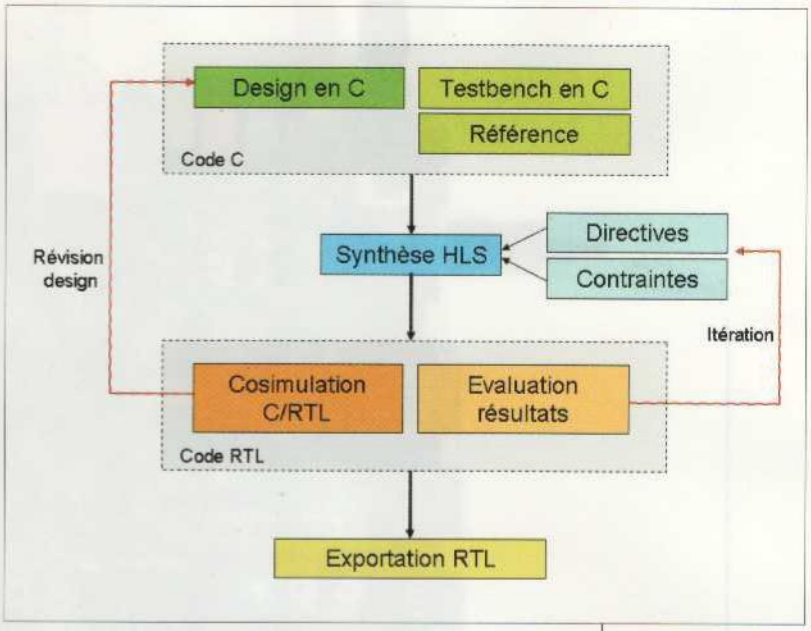
3 - Les valeurs de sortie précédentes sont alors appliquées au test bench C pour vérifier les résultats obtenus avec ceux de référence.

On peut revenir au *design* C en cas de problèmes fonctionnels, mais aussi réitérer la synthèse HLS en jouant sur les directives et sur les contraintes, en fonction des objectifs fixés en termes de surface, de latence, de débit...

Enfin, le design RTL peut être exporté sous forme d'un bloc IP (*Intellectual Property*), pouvant être ensuite importé dans un système SoPC développé avec l'outil Xilinx Vivado.

Les contraintes et les directives permettent d'influencer différents paramètres du design RTL, dont on peut rappeler leur définition :

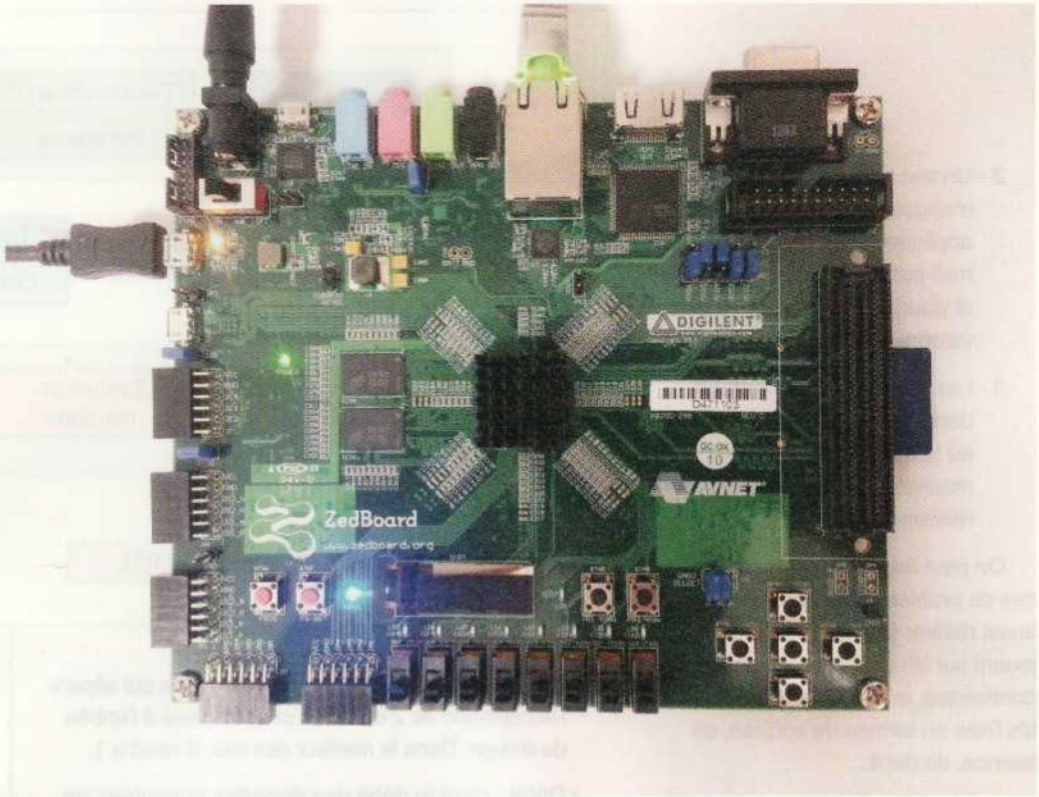
- Latence : temps en nombre de périodes d'horloge ou cycles (un design est toujours une logique synchrone sur front d'horloge) entre les valeurs appliquées à l'entrée du design et les valeurs produites en sortie.
- *Pipelining* : technique permettant d'optimiser le débit des données en entrée, par l'ajout d'éléments mémoire (bascule D) à des endroits bien précis dans le design. Dans le meilleur des cas, le design accepte de nouvelles données à chaque cycle. Dans le cas d'un *pipelining*, Xilinx définit un intervalle d'initialisation II (*Initiation Interval*), qui est le nombre de cycles qui sépare l'acceptation de 2 données consécutives à l'entrée du design. Dans le meilleur des cas, II vaudra 1.
- Débit : c'est le débit des données acceptées en entrée par le design. Si l'on raisonne en temps, c'est le nombre de cycles entre 2 données consécutives à l'entrée du design. S'il n'y a pas de *pipelining* utilisé, le débit en cycles est égal au temps de latence du design. Si l'on utilise la technique de *pipelining*, le débit en cycles est le II.



Flot de conception de Vivado HLS.

L'outil Vivado HLS permet de « jongler » avec ces paramètres grâce aux directives. On pourra en citer quelques-unes et l'auteur renvoie le lecteur à la documentation Xilinx :

- Directives de réduction du temps de latence : `LATENCY` pour spécifier les valeurs min et max de la latence, `UNROLL` pour mettre à plat une boucle algorithmique (au lieu d'exécuter n fois le design interne d'une boucle, on instancie *n* fois le design interne, qui est exécuté une fois (au détriment de la surface de circuit)), `LOOP_FLATEN` pour générer une seule boucle à partir de boucles imbriquées...
- Directives de *pipelining* : `PIPELINE` pour le *pipelining* d'une partie interne du design, `DATAFLOW` pour le *pipelining* du *top level* du *design*...
- Directives pour le choix des interfaces d'E/S.
- Directives pour le choix de l'implantation mémoire des paramètres d'entrée et de sortie du design.



Carte cible ZedBoard.

Vivado HLS a enfin un ordre de priorité dans ses directives :

1 - Atteindre les performances souhaitées : valeur de la période d'horloge, débit.

2 - Minimiser la latence.

3 - Minimiser la surface.

2. CARTE CIBLE ZEDBOARD

La carte utilisée dans cet article est la carte ZedBoard, qui est une carte d'évaluation développée par la société Digilent **[6] [7]** et met en œuvre le circuit FPGA Zynq Z-7020 de Xilinx.

Le circuit FPGA Zynq permet de réaliser un système SoPC et est constitué de logique programmable, couplée à un processeur *hard-core* double cœur ARM Cortex-A9.

La carte Zedboard possède ainsi les éléments suivants :

- Processeur ARM Cortex-A9 à 533 MHz avec 32 Ko de cache L1 et 512 Ko de cache externe L2.
- 512 Mo de RAM DDR3. 32 Mo de mémoire Flash Quad SPI.
- Slot mémoire SD.
- Sorties HDMI (audio et vidéo). Sortie VGA.
- Ethernet (10/100/1000 Mb/s).
- Port de débug ARM DAP (*Debug Access Port*).
- Port USB 2.0 OTG *device* et *host*. Port USB-JTAG. Port USB-UART.
- Connecteur FMC (*FPGA Mezzanine Connector*).
- 5 ports d'extension Pmod.
- 9 LED utilisateur. 8 interrupteurs à glissière. 7 boutons-poussoirs.

L'image ci-dessus présente la carte cible ZedBoard.

3. INSTALLATION DES OUTILS DE DÉVELOPPEMENT

Cette étape a été décrite dans l'article **[2]** et l'auteur encourage le lecteur à s'y référer. La version de Vivado utilisée dans cet article est ainsi la version 2018.2. Une version supérieure de Vivado est tout aussi valable.

L'environnement gratuit Xilinx WebPack **[5]** supporte le composant FPGA Zynq Z-7020 de la carte cible. Le lecteur pourra donc utiliser WebPack pour faire la synthèse HLS et la synthèse du système SoPC de l'article.

On récupérera les *tarballs* du compilateur croisé et du kit de développement SDK (*Software Development Kit*) construit par l'auteur pour la carte cible :

```
host% wget http://kadionik.vvv.enseirb-matmeca.fr/pub/ZedBoard/ZedBoard-hls.tgz
host% wget http://kadionik.vvv.enseirb-matmeca.fr/pub/ZedBoard/arm-2014.05.tgz
```

On installe d'abord le compilateur croisé :

```
host% tar -xvzf arm-2014.05.tgz
host% sudo mv arm-2014.05 /opt
```

Puis, on configure sa variable **PATH** via le fichier **.bash_profile** si l'on utilise le shell *bash* :

```
host% echo "export PATH=/opt/arm-2014.05/bin:$PATH" >> ~/.bash_profile
host% source ~/.bash_profile
```

On installe ensuite le kit de développement pour la carte cible :

```
host% tar -xvzf ZedBoard-hls.tgz
```

On se place dans le répertoire **ZedBoard/**. L'ensemble des manipulations décrites dans cet article sera réalisé à partir de ce répertoire. Les chemins seront donnés par la suite en relatif par rapport à ce répertoire :

```
host% cd ZedBoard
```

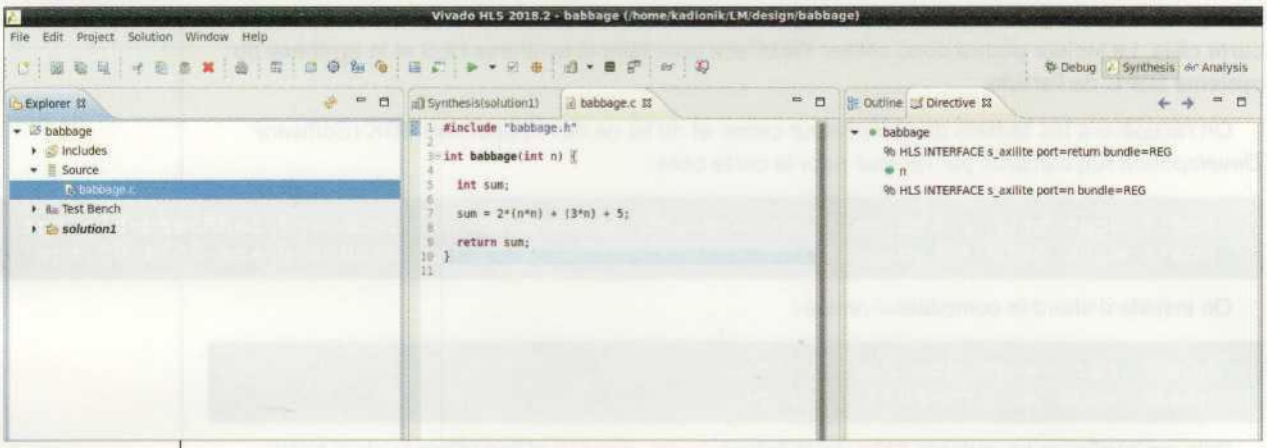
Le *shell* script **mbsdk** fourni dans le kit de développement permet de se placer dans l'environnement de développement Xilinx :

```
#!/bin/bash
# Run this for a Xilinx SDK bash shell
export LM_LICENSE_FILE=2100@localhost

export XIL_IMPACT_USE_LIBUSB=1
export GDK_NATIVE_WINDOWS=1

# Vivado 2018.2
source /opt/Xilinx/Vivado/2018.2/settings64.sh

# set prompt
export PS1="\[\e[32;2m\]\w\[\e[0m\]\n[Xilinx EDK 2018.2]$ "
bash
```

Par la suite, on adoptera les conventions suivantes dans l'article :

• Commande Linux PC hôte pour le développement croisé :

```
host% commande Linux
```



Interface graphique de Vivado HLS.

- Commande Linux PC hôte pour le développement Xilinx :

```
[Xilinx EDK 2018.2]$
```

- Commande *u-boot* :

```
Zynq>
```

- Commande Linux embarqué sur la carte cible ZedBoard :

```
ZedBoard:# commande Linux embarqué
```

4. DÉVELOPPEMENT HLS DU PÉRIPHÉRIQUE MATÉRIEL

À titre d'illustration, nous allons créer avec Vivado HLS le périphérique matériel réalisant l'opération suivante :

$f(n) = 2n^2 + 3n + 5$ avec n entier.

Pour cela, nous allons nous placer dans le répertoire **design/** et lancer Vivado HLS :

```
host% cd design
host% mbsdk
[Xilinx EDK 2018.2]$ vivado_hls -p babbage
```

L'interface graphique de Vivado HLS est très simple et intuitive, comme montré sur la figure ci-dessus.

Le code source C (**babbage.c**) de notre périphérique, saisi dans la zone d'édition, correspond à l'équation mathématique donnée précédemment :

```
#include "babbage.h"
int babbage(int n) {
  int sum;
  sum = 2*(n*n) + (3*n) + 5;
  return sum;
}
```

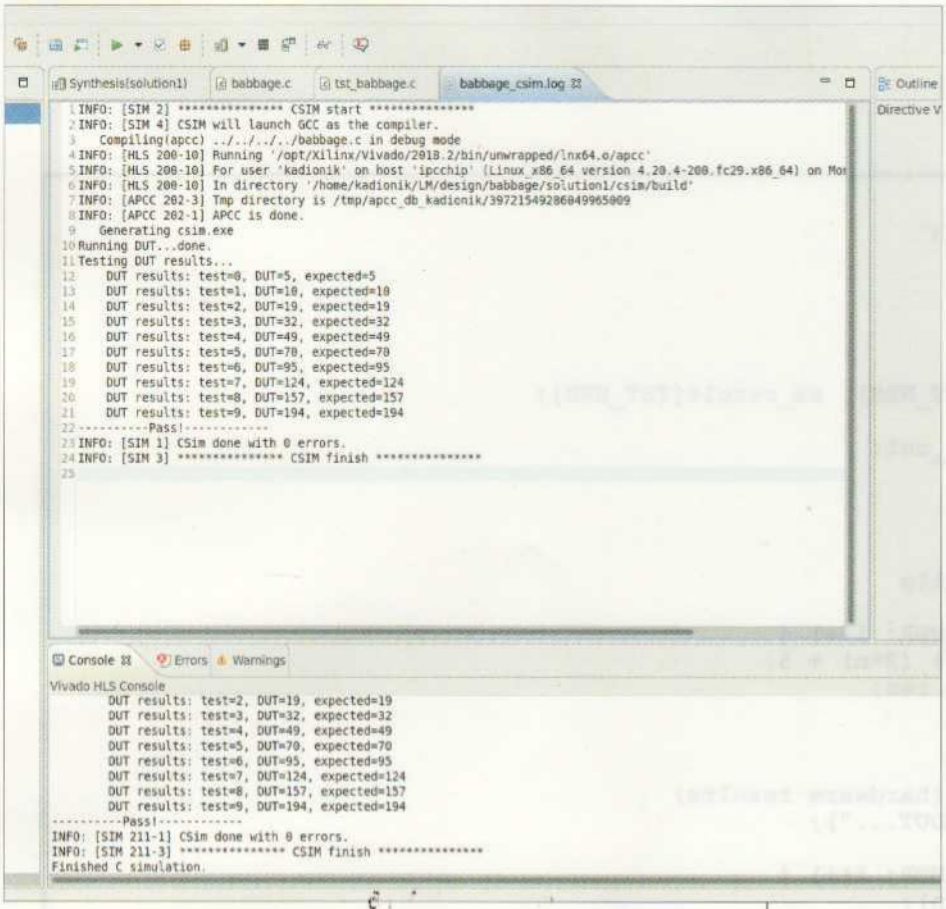
Le fichier **babbage.h** ne contient que le prototype de la fonction C **babbage()**.

Dans l'onglet « *Directive* », nous avons défini le type des paramètres d'E/S, **n** et **sum** de la fonction (**s_axilite**), comme étant accessibles en mode esclave depuis le bus AXI du processeur du système SoPC. Nous n'avons pas utilisé d'autres directives d'optimisation, par souci de simplification de l'exemple.

Nous avons aussi spécifié un fichier *test bench* (**tst_babbage.c**) pour la simulation C et la cosimulation C/RTL :

```
#include <stdio.h>
#include "babbage.h"

#define TST_NBR 10
int main()
{
  int n;
  int hw_result[TST_NBR], sw_result[TST_NBR];
  int res;
  unsigned int err_cnt;
  int i;

  err_cnt = 0;

  // Software results
  n = 0;
  for (i=0; i<TST_NBR; i++) {
    res = 2*(n*n) + (3*n) + 5;
    sw_result[i] = res;
    n++;
  }

  // Call the DUT (hardware results)
  printf("Running DUT...");
  n = 0;
  for (i=0; i<TST_NBR; i++) {
    res = babbage(n);
    hw_result[i] = res;
    n++;
  }
  printf("done.\n");

  // Test the output against expected results
  printf("Testing DUT results...\n");
  for (i = 0; i < TST_NBR; i++) {
    printf("\tDUT results: test=%d, DUT=%d, expected=%d\n", i, hw_result[i],
sw_result[i]);
    if (hw_result[i] != sw_result[i])
      err_cnt++;
  }

  if (err_cnt) {
    printf("----------Fail!-----------\n");
  }
  else {
    printf("----------Pass!-----------\n");
  }
}
```

Le source C du test bench est simple à comprendre. Un test est fait sur 10 valeurs (de 0 à 9 ici). Le tableau `sw_result[]` contient les vraies valeurs à obtenir (*golden references*), tandis que le tableau `hw_result[]` contiendra les valeurs obtenues soit par simulation C du périphérique, soit par cosimulation C/RTL. Notons que Vivado HLS utilise le même fichier *test bench* pour ces 2 tests, ce qui simplifie la vie du concepteur.



Simulation C du design.

La simulation C sera lancée en cliquant sur l'icône « *Run C Simulation* » de la barre d'outils.

La figure ci-dessus donne les résultats de la simulation C. Tout est OK, fort heureusement.

La synthèse HLS est lancée en cliquant sur l'icône « *Run C synthesis* ».

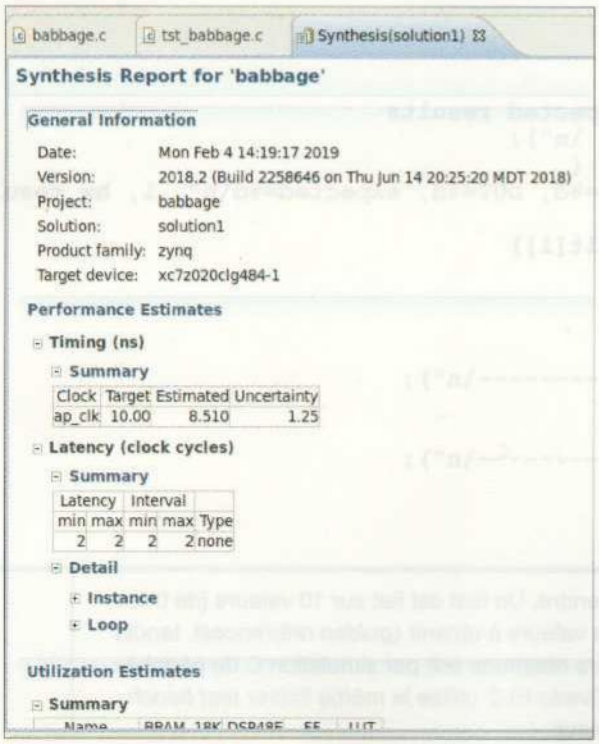
On récupère alors le rapport de synthèse donnant des informations relatives à cette opération (latence, période de l'horloge, éléments logiques utilisés...), comme indiqué sur la figure ci-contre.

La simulation C/RTL sera ensuite lancée pour vérification fonctionnelle de la solution RTL ainsi générée en cliquant sur l'icône « *Run C/RTL Simulation* ». La figure page suivante donne les résultats de la simulation C/RTL. Tout est toujours OK.

Il ne reste plus qu'à exporter le design sous la forme d'un bloc IP en cliquant sur l'icône « *Export RTL* ». Le fichier du bloc IP généré se trouve dans le répertoire `design/babbage/solution1/impl/ip/` et s'appelle ici `xilinx_com_hls_babbage_1_0.zip`. Il pourra être importé par la suite dans le design SoPC.

Nous voyons donc que la synthèse HLS est facile et nous n'avons pas eu à écrire une seule ligne de code VHDL. De plus, le périphérique matériel est automatiquement empaqueté dans un bloc IP prêt à l'emploi. En ayant utilisé la directive `s_axilite`, le bloc IP généré sera géré comme un esclave du bus AXI du processeur.

Comparons maintenant l'approche HLS avec une approche VHDL pure, comme



Résultats de synthèse HLS.

nous l'avions présenté dans l'article **[3]** paru dans GNU/Linux Magazine.

La mise en équation VHDL n'est pas immédiate. Nous utiliserons ainsi la technique de Charles Babbage pour cela (d'où le nom du design).

Nous avons :

$$f(n) = 2n^2 + 3n + 5$$

Nous pouvons remarquer que :

$$f(n) - f(n-1) = 4n + 1$$

Si l'on prend n entier et positif, nous pouvons écrire $f(n)$ de façon récursive :

$$f(n) = 5 \text{ pour } n=0$$

$$f(n) = f(n-1) + 4n + 1 \text{ pour } n>0$$

On peut ensuite recommencer avec $4n + 1$ en posant $g(n) = 4n + 1$

D'où :

$$g(n) - g(n-1) = 4$$

Nous pouvons écrire $g(n)$ de façon récursive :

$$g(n) = 1 \text{ pour } n=0$$

$$g(n) = g(n-1) + 4 \text{ pour } n>1$$

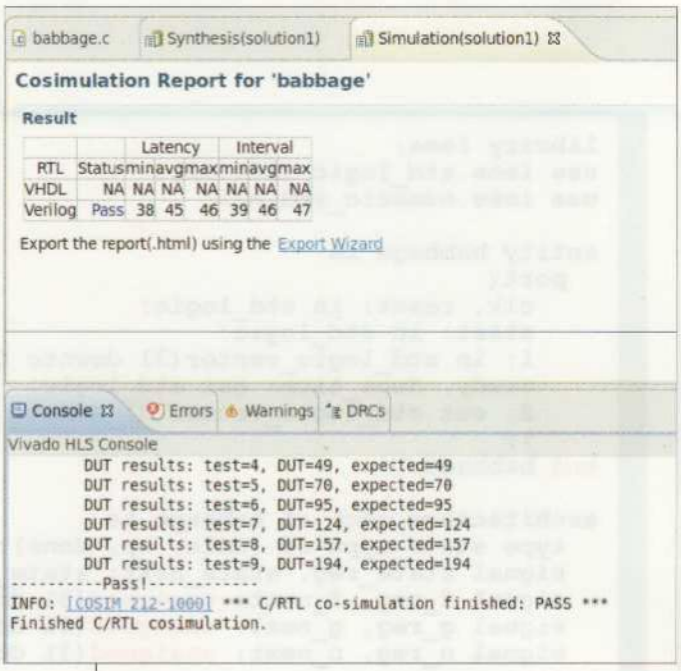
Et alors :

$$f(n) = 5 \text{ pour } n=0$$

$$f(n) = f(n-1) + g(n) = f(n-1) + g(n-1) + 4 \text{ pour } n>0$$

Nous avons ainsi transformé l'équation originelle, mettant en œuvre des multiplications, en un processus récursif qui se prête bien à l'écriture VHDL (registres), basée uniquement sur des additions.

Le code source VHDL correspondant est alors le suivant :



Simulation C/RTL du design.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity babbage is
  port(
    clk, reset: in std_logic;
    start: in std_logic;
    i: in std_logic_vector(31 downto 0);
    ready, done_tick: out std_logic;
    f: out std_logic_vector(31 downto 0)
    );
end babbage;

architecture arch of babbage is
  type state_type is (idle, op, done);
  signal state_reg, state_next: state_type;
  signal f_reg, f_next: unsigned(31 downto 0);
  signal g_reg, g_next: unsigned(31 downto 0);
  signal n_reg, n_next: unsigned(31 downto 0);
begin

-- fsmd state and data registers
  process(clk, reset)
  begin
    if reset='1' then
      state_reg <= idle;
      f_reg <= (others=>'0');
      g_reg <= (others=>'0');
      n_reg <= (others=>'0');
    elsif (clk'event and clk='1') then
      state_reg <= state_next;
      f_reg <= f_next;
      g_reg <= g_next;
      n_reg <= n_next;
    end if;
  end process;

-- fsmd next-state logic
  process(state_reg, n_reg, f_reg, g_reg, start, i)
  begin
    ready <='0';
    done_tick <= '0';
    state_next <= state_reg;
    f_next <= f_reg;
    g_next <= g_reg;
    n_next <= n_reg;

    case state_reg is
      when idle =>
        ready <= '1';
       if start='1' then
          f_next <= to_unsigned(5,32);
          g_next <= to_unsigned(1,32);
          n_next <= unsigned(i);
```

```
          state_next <= op;
        end if;
      when op =>
        if n_reg=0 then
          f_next <= to_unsigned(5,32);
          state_next <= done;
        else
        f_next <= f_reg + g_reg + 4;
        g_next <= g_reg + 4;
        n_next <= n_reg - 1;
        end if;
        if n_reg=1 then
          state_next <= done;
        end if;
      when done =>
        done_tick <= '1';
        state_next <= idle;
    end case;
  end process;

  -- output
  f <= std_logic_vector(f_reg);
end arch;
```

Nous retrouvons une machine d'états et des registres (`n_reg`, `n_next`), (`f_reg`, `f_next`) et (`g_reg`, `g_next`) pour implémenter la récursivité.

À comparer avec l'approche HLS, l'approche VHDL pure est nettement plus complexe et non immédiate, par rapport à l'approche HLS en langage C. Il faut ensuite, comme montré dans l'article **[3]**, incorporer le code VHDL dans le bloc IP à intégrer dans le design SoPC et modifier 2 fichiers VHDL automatiquement générés par Vivado. C'est loin d'être simple, l'approche VHDL permet de tout contrôler comme les timings, mais au prix de quelle énergie !

5. CONCEPTION DU SOPC

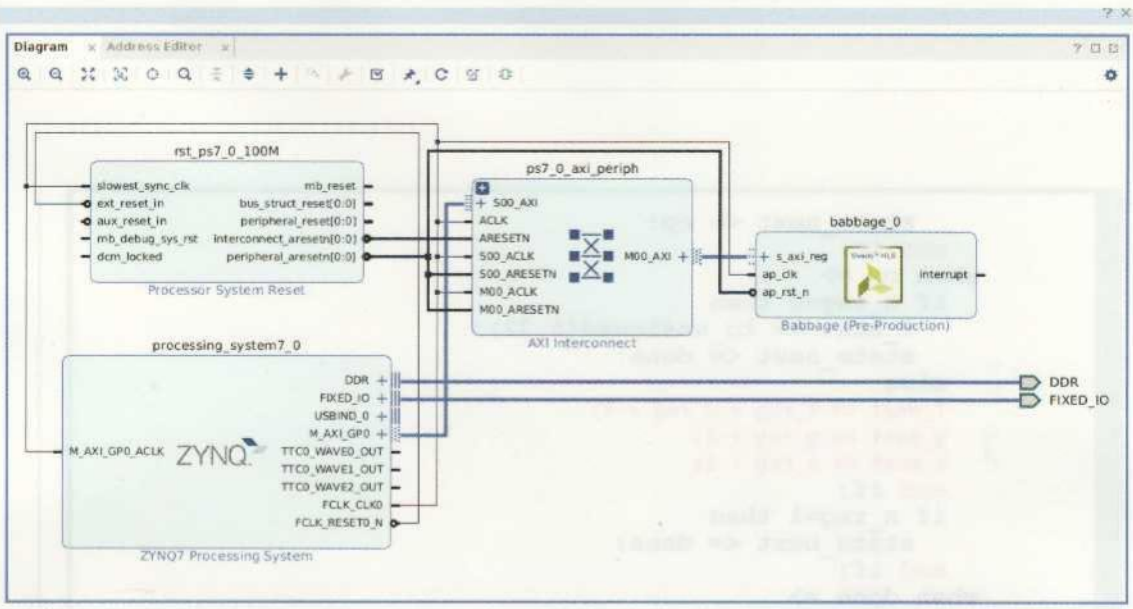
Nous allons partir d'un design de référence pour la carte ZedBoard, dont la construction a été présentée dans l'article **[2]** et auquel nous allons rajouter le bloc IP de notre algorithme généré précédemment par synthèse HLS. Le design de référence se trouve dans le répertoire `design/project_1/`.

On utilise pour cela l'outil Xilinx Vivado :

```
host% mbsdk
[Xilinx EDK 2018.2]$ vivado
```

Nous ouvrons alors le projet Vivado `project_1` sous `design/project_1/`.

Il faut ensuite ajouter l'accès au dépôt contenant le bloc IP de notre périphérique qui se trouve dans le répertoire `design/babbage/solution1/impl/ip/`. Pour cela, on utilise le menu « *Flow Navigator > Project Manager > Settings > IP > Repository* » et l'on ajoute le répertoire `design/babbage/solution1/impl/ip/`.



Design final pour la carte cible ZedBoard.

On ajoute alors au design notre bloc IP appelé **babbage**. On obtient alors le design final suivant, comme montré sur la figure ci-dessus.

En suivant le modus operandi détaillé dans l'article **[1]**, on lance la synthèse du système SoPC pour générer le fichier **.bit**.

On recopie le nouveau fichier de programmation à la place de l'ancien :

```
host% cd ZedBoard
host% cp design/project_1/project_1.runs/impl_1/design_1_wrapper.bit zb.bit
```

Notons enfin que Xilinx propose un atelier de développement logiciel (xsdk) basé sur Eclipse pour pouvoir développer des applications de test *bare metal* du système SoPC. Nous allons le faire plutôt sous Linux embarqué.

6. COMPILATION DU NOYAU LINUX

La marche à suivre a déjà été présentée dans l'article **[1]** et l'on pourra s'y référer pour plus de détails. Nous ne résumerons ici que les principales étapes.

On compile le noyau Linux standard pour la carte cible ZedBoard :

```
host% cd linux
host% ./godefconfig
host% ./go
```

Le fichier image **uImage** du noyau Linux est alors créé dans le répertoire **linux/arch/arm/boot/**.

Le fichier **uImage** est copié dans le répertoire **/tftpboot/** :

```
host% ./goinstall
```

Le fichier Device Tree **devicetree.dtb** fourni est copié dans le répertoire **/tftpboot/**. Sa génération est détaillée dans l'article **[1]** :

```
host% cp design/devicetree.dtb /tftp
```

7. CRÉATION DU SYSTÈME DE FICHIERS ROOT POUR LE NOYAU LINUX

La marche à suivre a déjà été présentée dans l'article **[1]** et l'on pourra s'y référer pour plus de détails. Nous ne résumerons ici que les principales étapes.

La première chose à faire est de créer le système de fichiers *root* squelette `root_fs` pour la carte cible ZedBoard :

```
host% cd ramdisk
host% ./goskel
```

Un système de fichiers *root* minimaliste est construit à la main autour de l'utilitaire *busybox*.

On compile *busybox* :

```
host% cd ramdisk
host% cd busybox
host% ./go
```

On ajoute ensuite le programme de test de notre périphérique.

Le fichier source `babbage.c`, dont le code source est donné ci-après, réalise le test de notre périphérique et il reprend, sous forme d'un driver en mode utilisateur, le programme de test `test_babbage.c` du code source HLS :

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <signal.h>
#include <sys/mman.h>
#include "xparameters.h"
#include "xbabbage_hw.h"

#define TST_NBR 10
#define AP_START 0x01
#define AP_STOP 0x00
#define AP_DONE 0x02
#define SIZE 4096

volatile int *ptr;

start() {
  // Offset registre en octets à traduire en 32 bits
  *(ptr + (XBABBAGE_REG_ADDR_AP_CTRL/4)) = AP_START;
}

stop() {
  *(ptr + (XBABBAGE_REG_ADDR_AP_CTRL/4)) = AP_STOP;
}
```

```
wait() {
  while((*(ptr + (XBABBAGE_REG_ADDR_AP_CTRL/4)) & AP_DONE) != AP_DONE) {
    ;
  }
}

int main(int argc, char **argv) {

  int fd;
  int n;
  int hw_result[TST_NBR], sw_result[TST_NBR];
  int res;
  unsigned int err_cnt;
  int i;

  fd=open("/dev/mem",O_RDWR | O_SYNC);
  if(fd < 0) {
    printf("Failed to open /dev/mem\n");
    exit(-1);
  }
  printf("/dev/mem open OK\n");

  ptr = mmap(0, SIZE, PROT_READ|PROT_WRITE,
    MAP_SHARED, fd, XPAR_BABBAGE_0_S_AXI_REG_BASEADDR);

  if(ptr == (void *)-1) {
    close(fd);
    printf("mmap failed\n");
    exit(-1);
  }
  printf("mmap OK\n");

  stop();

  err_cnt = 0;

  // Software results
  n = 0;
  for (i=0; i<TST_NBR; i++) {
    res = 2*(n*n) + (3*n) + 5;
    sw_result[i] = res;
    n++;
  }

  // Call the DUT (hardware results)
  printf("Running DUT...");
  n = 0;
  for (i=0; i<TST_NBR; i++) {
    *(ptr + (XBABBAGE_REG_ADDR_N_DATA/4)) = n;
    start();
    wait();
    stop();
    hw_result[i] = *(ptr + (XBABBAGE_REG_ADDR_AP_RETURN/4));
    n++;
  }
  printf("done.\n");
```

```
  // Test the output against expected results
  printf("Testing DUT results...\n");
  for (i = 0; i < TST_NBR; i++) {
    printf("\tDUT results: test=%d, DUT=%d, expected=%d\n",
        i, hw_result[i],sw_result[i]);
    if (hw_result[i] != sw_result[i])
      err_cnt++;
  }

  if (err_cnt) {
    printf("----------Fail!-----------\n");
  }
  else {
    printf("----------Pass!-----------\n");

  }
  munmap((void *)ptr, SIZE);
  close(fd);
  exit(0);
}
```

On compile le fichier source **babbage.c** et on installe son exécutable dans le système de fichiers *root* :

```
host% cd tst
host% cd babbage
host% ./go
host% ./goinstall
```

On génère enfin un fichier *RAM disk* **ramdisk.image.gz** à partir du système de fichiers *root* **root_fs** ainsi construit pour la carte cible.

Le fichier **ramdisk.image.gz** sera ensuite transformé en un fichier image **uramdisk.image.gz** par la commande **mkimage**, pour pouvoir être exploité par *u-boot* et que l'on installera dans le répertoire **/tftpboot/** :

```
host% cd ramdisk
host% ./gorootfs
host% ./goinstall
```

8. TEST DU PÉRIPHÉRIQUE SOUS LINUX EMBARQUÉ

Nous avons maintenant sous **/tftpboot/** les 3 fichiers nécessaires pour lancer notre noyau Linux embarqué sur la carte ZedBoard :

- Fichier **uImage** du noyau Linux.
- Fichier *RAM disk* **uramdisk.image.gz** du système de fichiers root.
- Fichier *Device Tree* **devicetree.dtb** pour le noyau Linux.

On programme d'abord le circuit FPGA Zynq :

```
host% cd ZedBoard
host% mbsdk
[Xilinx EDK]$ ./load-design
```

On se connecte ensuite par minicom à *u-boot* de la carte cible ZedBoard :

```
host% minicom -D /dev/ttyACM0
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Aug  4 2017, 01:47:26.
Port /dev/ttyACM0, 15:35:43

Press CTRL-A Z for help on special keys

U-Boot 2017.01-dirty (Dec 14 2018 - 14:35:44 +0100)

Model: Zynq Zed Development Board
Board: Xilinx Zynq
DRAM:  ECC disabled 512 MiB
MMC:   sdhci@e0100000: 0 (SD)
SF: Detected s25fl256s_64k with page size 256 Bytes, erase size 64 KiB, total 3B
In:    serial@e0001000
Out:   serial@e0001000
Err:   serial@e0001000

Model: Zynq Zed Development Board
Board: Xilinx Zynq
Net:   ZYNQ GEM: e000b000, phyaddr 0, interface rgmii-id
eth0: ethernet@e000b000

Hit any key to stop autoboot:  0
Zynq>
```

On utilise enfin la commande *u-boot* suivante pour télécharger par TFTP les 3 fichiers nécessaires puis lancer le noyau Linux :

```
Zynq> run jtagboot
```

On peut alors vérifier le bon fonctionnement de notre périphérique matériel avec l'outil **babbage** :

```
. . .
Freeing unused kernel memory: 1024K
    Hostname       : ZedBoard
    Kernel release : Linux 4.14.0-xilinx-00001-g46173a744b24-dirty
    Kernel version : #15 SMP PREEMPT Tue Feb 5 11:55:42 CET 2019

 Mounting /proc             : [SUCCESS]
 Mounting /sys              : [SUCCESS]
 Mounting /dev              : [SUCCESS]
 Mounting /dev/pts          : [SUCCESS]
 Enabling hot-plug          : [SUCCESS]
 Populating /dev            : [SUCCESS]
 Mounting other filesystems : [SUCCESS]
 Starting telnetd           : [SUCCESS]
 Network configuration      : IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[SUCCESS]

System initialization complete.
```

```
Please press Enter to activate this console.

ZedBoard:/# babbage
/dev/mem open OK
mmap OK
Running DUT...done.
Testing DUT results...
        DUT results: test=0, DUT=5, expected=5
        DUT results: test=1, DUT=10, expected=10
        DUT results: test=2, DUT=19, expected=19
        DUT results: test=3, DUT=32, expected=32
        DUT results: test=4, DUT=49, expected=49
        DUT results: test=5, DUT=70, expected=70
        DUT results: test=6, DUT=95, expected=95
        DUT results: test=7, DUT=124, expected=124
        DUT results: test=8, DUT=157, expected=157
        DUT results: test=9, DUT=194, expected=194
----------Pass!-----------
ZedBoard:/#
```

Nous pouvons remarquer que notre périphérique matériel, développé en C avec l'approche HLS, fonctionne parfaitement sous Linux embarqué.

CONCLUSION

À travers cet article, nous avons pu voir la création d'un périphérique matériel en langage C par l'approche HLS (*High Level Synthesis*).

L'approche HLS permet de s'affranchir d'écrire du code VHDL et permet ainsi à un informaticien de créer simplement son matériel.

Nous avons pu voir sa mise en œuvre avec l'outil Vivado HLS, car nous restons néanmoins tributaires d'outils qui n'existent pas en libre. Il n'existe d'ailleurs pas non plus de circuit FPGA libre.

Ce périphérique écrit en langage C a d'abord été testé par simulation C, puis C/RTL avec l'outil Vivado HLS.

Puis, il a été intégré dans un système SoPC pour être ensuite implanté dans un circuit FPGA Zynq de Xilinx. La mise en œuvre a été réalisée sur la carte de développement ZedBoard.

Le périphérique a enfin été testé sous Linux embarqué sur la carte cible ZedBoard.

Nous avons pu apprécier la facilité de développement par l'approche HLS, par rapport à une approche plus classique tout VHDL.

Après avoir vu la mise en œuvre du *codesign* sous Linux embarqué avec une carte FPGA Zynq ZedBoard **[1]**, la mise en œuvre de Xenomai **[2]**, la création d'un périphérique matériel en langage VHDL **[3]**, ce quatrième et dernier article sur la création d'un périphérique matériel par l'approche HLS clôt (provisoirement) le sujet passionnant du développement conjoint matériel/logiciel.

À vous de jouer maintenant ! **PK**

BIBLIOGRAPHIE

[1] Mise en œuvre de Linux embarqué sur carte FPGA Zynq ZedBoard. Patrice Kadionik. *Open Silicium* n° 17, p. 20-30. Janvier 2016

[2] Mise en œuvre de Xenomai sur carte FPGA Zynq ZedBoard. *Open Silicium* n° 18, p. 61-65. Avril 2016

[3] Créez et pilotez votre périphérique matériel sous Linux embarqué. Patrice Kadionik. *GNU/Linux Magazine* n° 214, p. 52-67. Avril 2018

[4] The Zynq Book. L. Crocket and al. Editions Strathclyde Academia Media. 2014. **http://www.zynqbook.com**

[5] Site de Xilinx : **http://www.xilinx.com**

[6] Carte Digilent ZedBoard : **http://www.digilentinc.com/zedboard**

[7] Site communautaire sur la carte cible ZedBoard : **http://zedboard.org/**

[8] FPGA prototyping by VHDL examples. P. Chu. Éditions Wiley. 2008

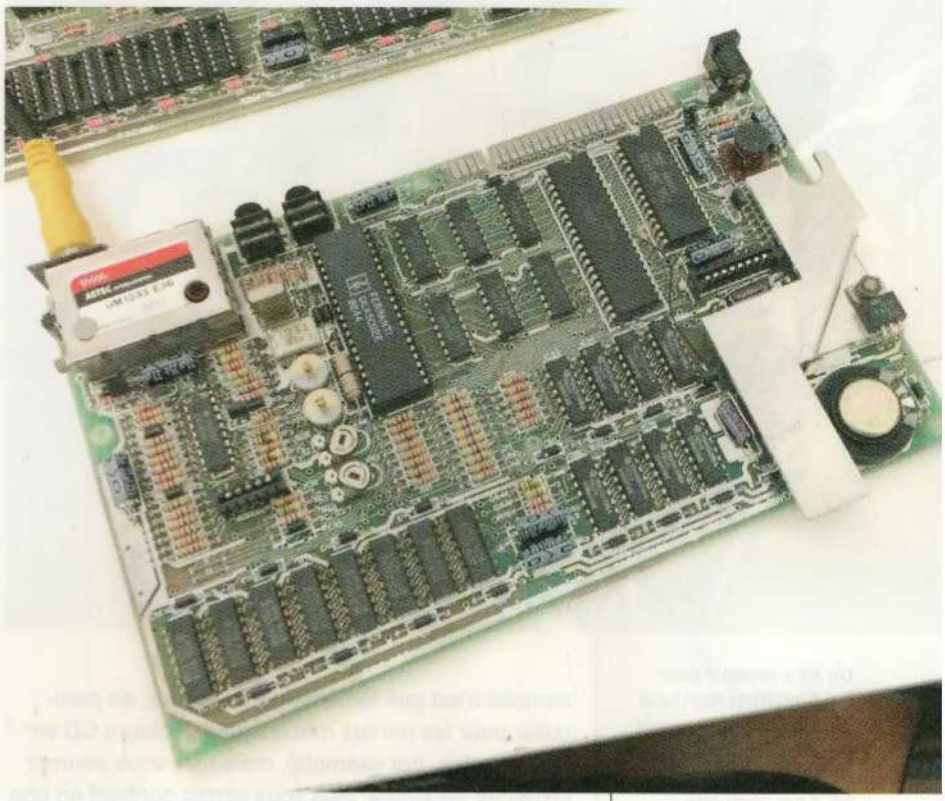
DE LA POUBELLE AU SALON : LE ZX SPECTRUM

Denis Bodor

Lorsqu'on s'intéresse à l'Histoire et plus précisément l'histoire de l'informatique, au point de vouloir posséder l'un de ses morceaux, il y a deux approches possibles : acquérir du matériel testé et fonctionnel ou récupérer ce qui pourrait passer pour un déchet électronique (pour le commun des mortels) et lui redonner vie. N'étant pas du genre à apprécier qu'on fasse les choses à ma place, je préfère largement la seconde option, me permettant par la même occasion d'apprendre et de partager mes découvertes et déboires avec vous...

Cet article sera le premier d'une nouvelle série que j'ai intitulé « De la poubelle au salon ». Chaque article de la série sera indépendant, mais reposera sur le même principe : « ressusciter » ou plus exactement « réanimer » un ordinateur, une console ou un équipement vintage, tout en détaillant les améliorations, les points importants et les techniques de rénovation ou de réparation qui s'y rapportent.

À chaque fois, nous verrons un nouvel équipement, mais bien entendu, la publication de ce type d'articles ne sera pas systématique. Prendre soin d'une machine et tout faire pour lui rendre sa beauté et sa vigueur passées n'est pas une science exacte. Je ne peux donc pas vous promettre si et quand une machine particulière fera l'objet d'un article. Ce que je peux vous dire, en revanche, c'est ce qui constitue en partie la liste des machines pouvant être traitées dans un futur article puisqu'en ma possession : Silicon Graphics (SGI) Indy, Cambridge Z88, Thomson MO5 édition Michel Platini, Macintosh SE/30, Acorn RISC PC, iMac G4 15" (lampe), Commodore 64 (et 128), Atari 2600, Amiga 1200...



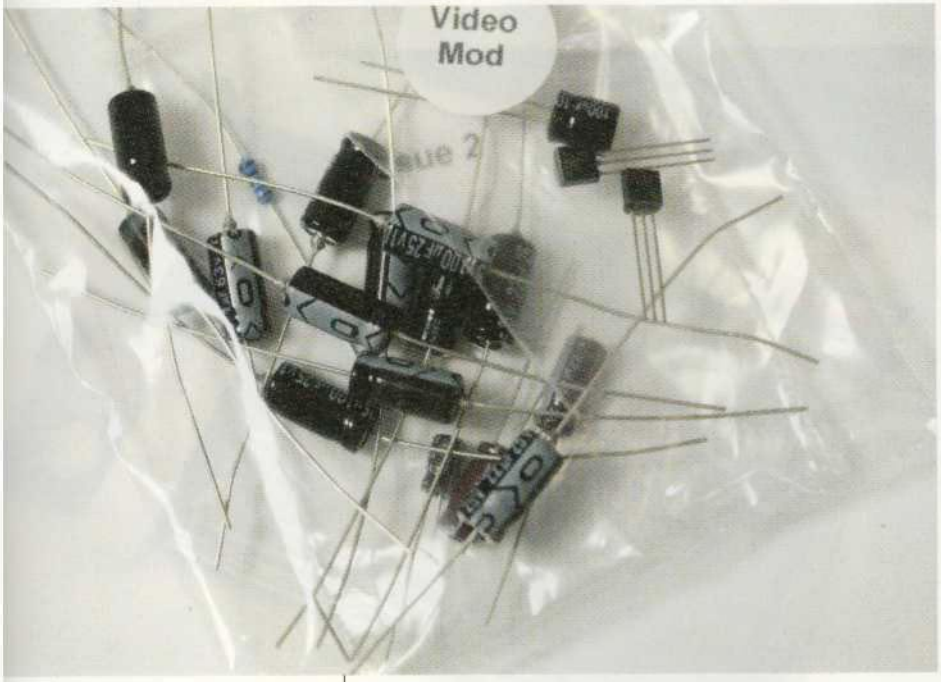
Voici les entrailles d'un ZX Spectrum 48K avec, de gauche à droite et de haut en bas : le modulateur UHF, l'ULA, les CI logiques, le Z80, la ROM, 16 Ko de RAM basse, 32 Ko de RAM haute (optionnelle et composée de 4164 (64 K x 1 bit) partiellement fonctionnels) et le haut-parleur intégré.

1. QUELQUES RÈGLES DE BASES

Avant de nous pencher sur la machine qui nous intéressera pour cet article, il est important de remarquer qu'un certain nombre de « règles » s'applique à n'importe quelle restauration électronique/informatique et sans doute, même à d'autres domaines de restauration (mécanique, automobile, etc.). La plus importante de ces règles est la suivante : **n'alimentez et ne tentez pas de mettre en marche la chose avant de l'avoir inspectée !**

Quelle que soit la provenance de l'équipement ou de la machine, marché aux puces, votre propre grenier, achat en ligne, poubelle du coin... vous ne pouvez pas connaître son état de santé. Que la machine ait ou non l'air en bon état n'a absolument aucune importance. Comme pour une voiture, le fait que la carrosserie soit brillante et bien propre ne vous garantit non seulement pas qu'elle va démarrer en tournant la clé et pas davantage qu'elle ne prendra pas feu quelques secondes après. Pour un ordinateur ou une console, c'est exactement la même chose et vous pourriez provoquer des dégâts très importants et coûteux, à cause d'un composant qui vous aurait coûté quelques centimes si vous l'aviez changé au préalable.

La première chose à faire sera donc de démonter le matériel pour en inspecter chaque centimètre carré. L'objectif de cette inspection minutieuse est de détecter le moindre signe d'usure anormale ou de dommages : corrosion, défauts physiques, joints de soudure décollés ou cassés, traces de brûlures, câbles abîmés ou sectionnés, etc. Un démontage



Un kit « recap » pour ZX Spectrum regroupe tous les condensateurs nécessaires pour un remplacement intégral. Ici, il est complété d'un kit « video mod » permettant de convertir la sortie vidéo UHF en composite de plusieurs façons.

complet n'est pas forcément nécessaire, en particulier pour les parties mécaniques (lecteurs CD sur les consoles, par exemple), mais plus vous pourrez inspecter de zones, plus vous serrez confiant en une mise sous tension en toute sécurité (sans jamais en avoir la garantie absolue).

Le démontage de l'équipement sera également l'occasion de nettoyer le matériel ou simplement de le dépoussiérer correctement. Une fois la partie électronique mise de côté, il ne s'agit finalement le plus souvent que d'une boîte en plastique en plusieurs morceaux, pouvant donc être traitée comme telle. Il existe bien des façons de nettoyer et de prendre soin de ce type de matière, mais dans sa version la plus simple, un peu d'eau chaude et quelques gouttes de produit vaisselle feront déjà un très bon travail.

Et pendant que tout cela sèche, vous pourrez vous pencher sur un autre point très important dans toute restauration : collecter des informations utiles pour vous aider dans votre projet. Quel que soit la machine ou l'équipement concerné, son état ou son âge, il y a fort à parier que vous n'êtes pas le premier à chercher à lui redonner vie. Plus la machine a été populaire par le passé, plus il y a eu de restaurations et plus une importante masse de connaissances aura été réunie. Ceci concerne aussi bien les points faibles du matériel, les techniques de réparation des pannes récurrentes, les symptômes de ces pannes, les différentes versions pouvant exister, les différences entre ces versions, etc.

Chaque matériel possède une histoire et des défauts qui, avec le temps, prennent une importance capitale, mais il est un problème dont le caractère universel ne fait aucun doute : le vieillissement des condensateurs électrolytiques. Ceci est tout simplement **LE** problème, dès lors que l'on parle d'électronique prenant de l'âge. Un condensateur électrolytique, lors de sa fabrication, ne contient pas d'isolant, mais un électrolyte en gelée intercalé entre les parties métalliques. Ce n'est que lorsque le courant passe dans le composant, une première fois, qu'une fine couche d'isolant se forme à la surface du métal et lui donne ses caractéristiques et sa capacité. Mais cet état n'est pas stable et bien qu'étant étanche, ces composants vieillissent mal, se dessèchent ou finissent pas fuir (physiquement ou électriquement).

Ceci est d'autant plus vrai que ces condensateurs électrolytiques sont généralement utilisés dans les circuits de filtrage d'alimentation et donc, à proximité de sources de chaleur qui contribuent à leur détérioration. Les conséquences d'une dégradation d'un ou plusieurs condensateurs de ce type peuvent être catastrophiques. En effet, en l'absence de filtrage, les pics et chutes de tension générés par le circuit d'alimentation peuvent se propager jusqu'aux composants les plus sensibles et les plus difficiles à remplacer (circuit intégré, mémoire, contrôleur, processeur, etc.).

De plus, un condensateur défectueux fuit. Physiquement, si son électrolyte sort du com-

posant, mais surtout électriquement, en laissant passer un courant. Cette fuite de courant, couplée à une augmentation de la résistance série équivalente (ESR), fait chauffer le composant qui va alors se dégrader encore plus vite. Cette chaleur a également un effet mécanique sur le condensateur, qui va alors présenter une déformation caractéristique : la partie supérieure du condensateur sera bombée et gonflée. Attention cependant, un composant dans cet état est par définition à remplacer, mais l'absence de déformation n'est absolument pas un signe de bonne santé pour autant.

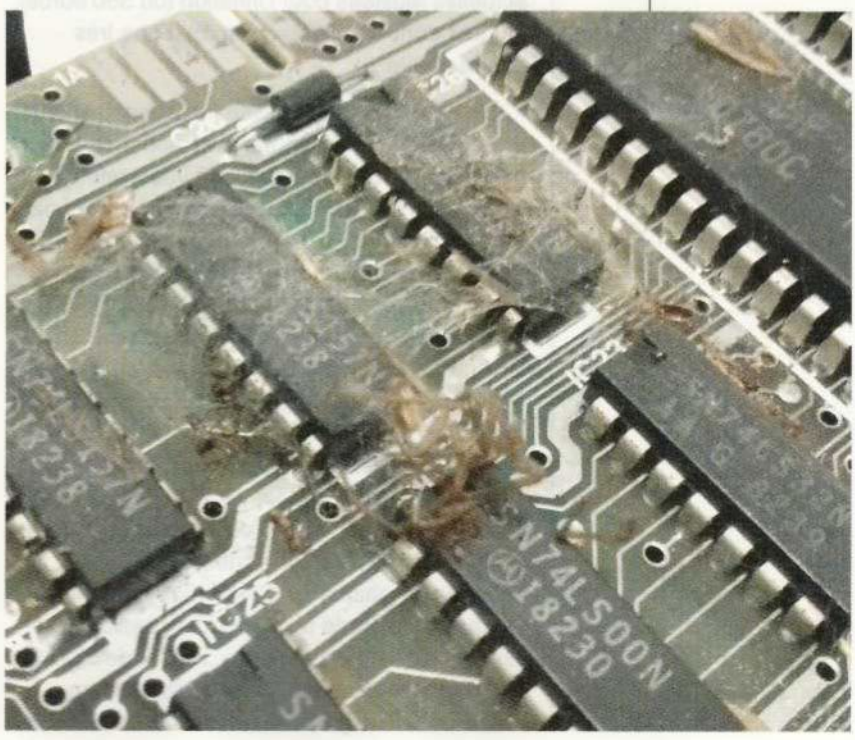
Le rétrocomputing concernant généralement des machines ayant 25 ans ou plus, la question de savoir s'il est nécessaire ou non de remplacer les condensateurs électrolytiques ne se pose même pas. La seule réponse raisonnable est « oui » par défaut. Bien sûr, vous pouvez avoir de la chance et miser sur une simple inspection visuelle pour ne remplacer que les composants gonflés, mais tout dépendra de la machine et de sa rareté en particulier. Un ZX Spectrum, par exemple, n'est pas une pièce rare ni même très difficile à trouver pour moins de 25€. On peut donc se permettre de prendre quelques risques mesurés, ce qui ne serait certainement pas le cas avec quelque chose de bien moins courant, comme une station de travail NeXT, un RISC PC et très certainement pas avec une véritable pièce de collection, comme un Apple Lisa (dont un exemplaire était mis en vente l'année dernière sur eBay pour quelque 50000€ !).

2. LE ZX SPECTRUM

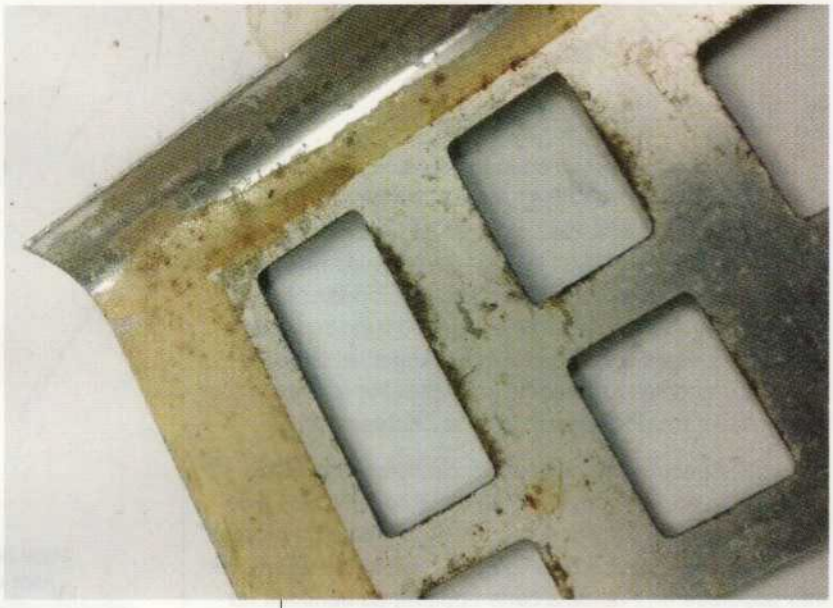
Le ZX Spectrum est un ordinateur personnel (ou familial) britannique des années 80. Celui-ci a commencé à être commercialisé en avril 1982 et fut construit, comme ses prédécesseurs, le ZX80 et le ZX81, par Sinclair Research, une société anglaise basée à Cambridge, fondée par Sir Clive Sinclair. Ce modèle précis arrive en plein boum de l'informatique grand public et vise à corriger les critiques faites aux modèles précédents (en particulier concernant le clavier, qui s'améliore encore avec le ZX Spectrum+).

Le ZX Spectrum se décline principalement en deux versions, avec 16 Ko et 48 Ko de mémoire vive puis plus tard, une version 128 Ko et est construit autour du processeur Z80, cadencé à 3,5 MHz. Le ZX Spectrum n'était pas très populaire en France à cette époque, au bénéfice du Commodore 64, mais aussi du Thomson MO5, massivement présent dans les établissements scolaires en raison du « Plan Informatique pour Tous » initié par le Premier ministre de l'époque, Laurent Fabius.

Dans l'Hexagone, le ZX Spectrum peine à concurrencer les deux gros acteurs que sont Commodore et Amstrad durant les années 80, avec pour conséquence



L'une des joies du rétrocomputing : parfois nous avons, en prime, un cadeau supplémentaire, comme ici sous la forme d'une très belle araignée lyophilisée (sans doute pas d'époque).



L'ancien adhésif double face doit être retiré : coton-tige, acétone, brosse à dents, alcool isopropylique et beaucoup de patience et de délicatesse.

aujourd'hui une faible quantité de machines disponible. Mais outre-Manche, c'est une tout autre histoire, puisque cet ordinateur est littéralement le MO5 anglais. Il est donc très simple de trouver un ZX Spectrum en relativement bon état sur eBay pour quelque 20€ ou moins. Vendu 175 livres sterling en 1982, soit 480 livres actuelles ajustées pour l'inflation (ou 550 euros), on peut donc considérer que c'est une très bonne affaire...

Le ZX Spectrum est assez représentatif des machines de cette époque avec son architecture 8 bits, peu de mémoire, un interpréteur BASIC en ROM et une sortie vidéo passant par un modulateur UHF, permettant la connexion à un téléviseur (PAL, bande 4, canal 35). Côté caractéristiques, nous sommes très loin des fonctionnalités comme celles offertes par un C64 ou un CPC 464. Le mode texte du ZX Spectrum affiche 24 lignes de 32 colonnes avec 16 teintes (8 couleurs normales/claires). Le mode graphique affiche 256 par 192 pixels et est relativement limité, la machine ne disposant pas de gestion matérielle des *sprites* comme le C64. Pire encore, les attributs de couleur en mode graphique sont basés sur une grille découlant du mode texte : il n'est possible d'utiliser que deux couleurs par « zone » de 32×24 pixels, conduisant tantôt à d'inévitables problèmes d'affichage (*attribute clash*).

Côté audio, là encore, l'absence de processeur sonore limite grandement les possibilités et la qualité des jeux, qui doivent alors user d'astuces pour exploiter au maximum les ressources disponibles. L'absence de ports pour les joysticks était compensée par la possibilité d'ajouter une extension matérielle fournissant deux interfaces de ce type (interface 2) ou un port série RS-232 (interface 1). Malgré cela, de nombreux jeux étaient disponibles pour la machine, généralement commercialisés sous forme de cassettes audio qu'il fallait lire avec un magnétophone connecté à l'ordinateur, via l'entrée dédiée. La très grande majorité de ces jeux utilisaient uniquement l'entrée clavier, tous les utilisateurs ne disposant pas de joysticks.

En résumé, le ZX Spectrum était une machine certes accessible, mais nécessitant finalement un budget conséquent, dès lors que la personne souhaitait ajouter les fonctionnalités courantes de l'époque, et ce, pour un résultat qui n'étaient fonctionnellement pas à la hauteur du matériel concurrent. Il n'en reste pas moins que cet ordinateur constitue un morceau d'Histoire important, qui pourra s'ajouter à votre collection pour un budget très raisonnable et peu d'efforts. La machine étant construite autour d'un Z80, elle pourra également constituer une plateforme d'expérimentation intéressante si vous vous intéressez à ce processeur. Les autres ordinateurs 8 bits les plus populaires (C64, Apple II, etc.) utilisent un MOS 6502 (ou 6510) qui est initialement plus complexe à appréhender en assembleur.

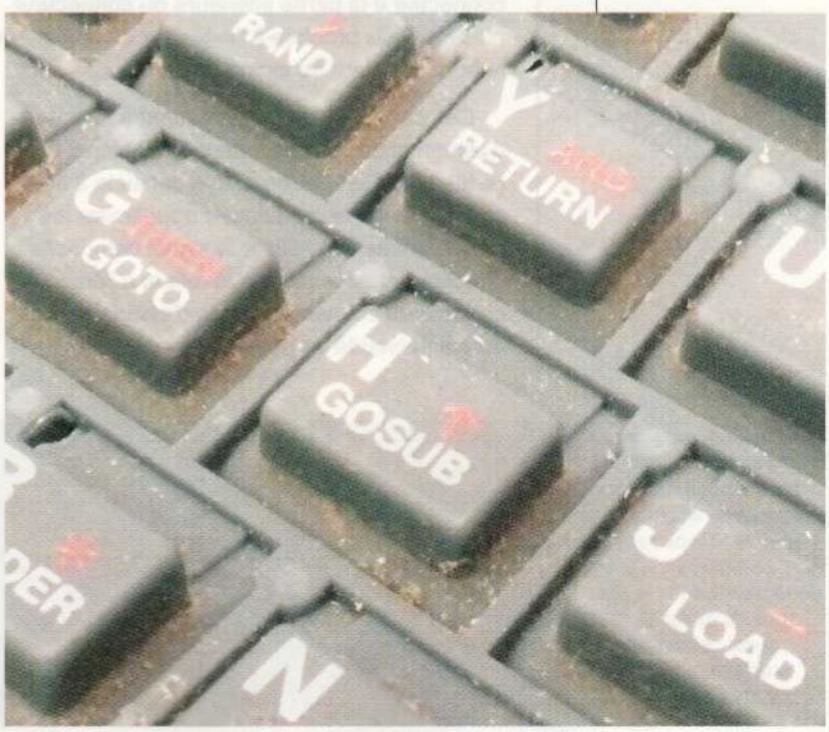
3. RAFRAÎCHIR LA MACHINE

La machine sur laquelle repose cet article est un Sinclair ZX Spectrum 48K, acquis sur eBay auprès d'un vendeur anglais utilisant le pseudonyme *sienna270615* pour 24€. Rapidement arrivé par la poste, l'ordinateur m'a été livré sans alimentation et c'est là le premier point capital qu'il faut connaître, avant de travailler avec un Spectrum : **le connecteur d'alimentation DC 9 volts a une polarité inversée !** Le centre du connecteur est le pôle négatif et l'extérieur le pôle positif. Derrière ce connecteur se trouve un régulateur 7805 qui, en cas d'utilisation d'un bloc d'alimentation classique (9V 1A) avec le positif au centre, à 90% de chance de ne pas survivre à l'erreur. Ceci confirme donc mes recommandations précédentes : ne pas brancher et se documenter.

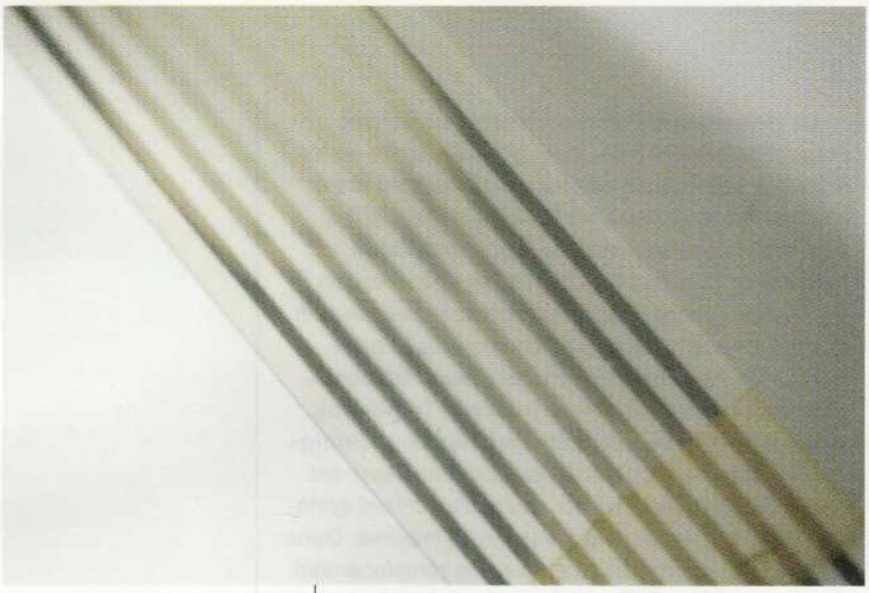
Comme également préconisé par moi-même précédemment, avant même de mettre la machine sous tension, le remplacement systématique des condensateurs électrolytiques doit être fait. Il n'est pas nécessaire d'acheter les composants individuellement, car des vendeurs eBay proposent des kits s'adaptant à toutes les versions de la machine. Le mien m'aura coûté 3,50€ auprès d'un vendeur appelé *phil6581*. À ce prix, j'ai obtenu non seulement un jeu complet de condensateurs (3 x 1 µF, 1 x 4,7 µF, 6 x 22 µF et 2 x 100 µF), mais également de quoi modifier la sortie vidéo de la machine afin d'obtenir un signal composite, en lieu et place du signal UHF (condensateur et transistor).

Le remplacement des condensateurs est une étape relativement aisée pour qui sait manier un fer à souder. Les valeurs de chaque composant étant inscrites, la vérification est facile et on procédera à un remplacement composant par composant. Mieux vaut en effet éviter de risquer une inversion en retirant d'abord tous les composants pour souder ensuite les nouveaux (en cas de doute, cette page résume les valeurs et positions de chaque condensateur en fonction du modèle de machine : **http://blog.retroleum.co.uk/electronics-articles/re-capping-the-spectrum**). La polarité des condensateurs est également évidente à définir, puisqu'un "+" est systématiquement sérigraphié sur le circuit imprimé. Dans mon cas, la seule subtilité aura été le remplacement du condensateur C47, initialement axial, par un modèle radial auquel il faut complètement tordre une des broches afin de la positionner à l'horizontale.

Ceci fait, il est temps de se pencher sur la modification de la sortie vidéo. En effet, avec un équipement moderne comme un téléviseur ou un convertisseur analogique vers HDMI, le seul moyen d'obtenir une image, et donc de vérifier que la machine



En plus de 35 ans, certaines choses ne changent pas. Les claviers étaient, sont et seront toujours de parfaits pièges à crasse.



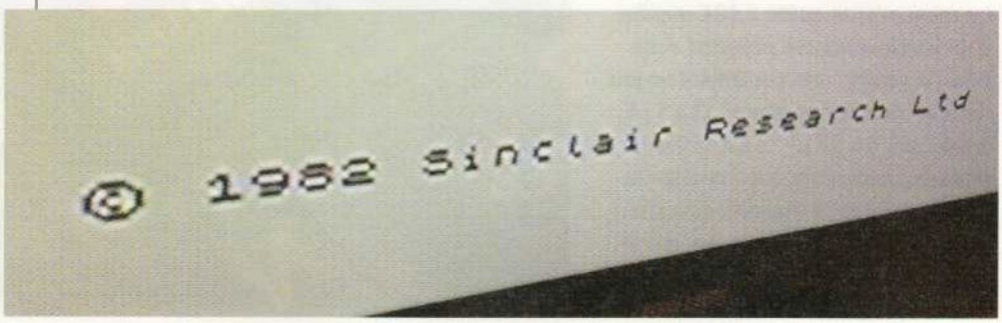
Ceci est le câble plat reliant la membrane du clavier au circuit intégré. C'est LE point faible du ZX Spectrum, car avec le temps, les couches de plastique se désolidarisent et exposent la partie conductrice qui s'oxyde. La connexion finit par se rompre et tout ou partie des touches du clavier ne répondent plus.

fonctionne, est de passer outre le modulateur UHF. Fort heureusement, un signal composite est bel et bien présent dans la machine, mais n'est simplement pas proposé en sortie. L'astuce consiste alors à utiliser le connecteur UHF, en déconnectant le modulateur à l'intérieur du blindage métallique. Une fois le boîtier ouvert, on commencera par dessouder la résistance connectée à la partie intérieure du connecteur, puis les deux broches à l'extérieur du boîtier sur le bord extérieur du circuit imprimé.

Ces deux pattes correspondent à la ligne d'alimentation du modulateur et au signal composite utilisé en entrée. L'idée consiste à ne plus alimenter le modulateur pour limiter les interférences et à utiliser le signal composite directement en sortie, en le reliant à l'âme centrale du connecteur *cinch* (l'extérieur étant connecté/soudé au boîtier et à la masse). La simple connexion est la modification dans sa forme la plus simple, mais il est plus judicieux de placer un condensateur de 100 µF entre le signal et la sortie (avec le « moins » du condensateur côté sortie). Ceci assure une compatibilité améliorée avec les téléviseurs récents. Une autre solution possible consiste à remplacer ce condensateur par un transistor NPN (BC549C) pour *booster* le signal et obtenir une image plus claire et contrastée. Personnellement, la solution à condensateur me convenait parfaitement.

4. RÉPARER CE QUI NE MARCHE PAS

Une fois ce rafraîchissement fait, il ne reste plus qu'à croiser les doigts, connecter la sortie vidéo sur un téléviseur ou un convertisseur et alimenter l'ordinateur en ayant, une fois de plus, vérifié la polarité du bloc d'alimentation. Dans mon cas bonne surprise, l'écran blanc, présentant le texte « *© 1982 Sinclair Research Ltd* » en noir sur la dernière ligne, apparaît relativement clairement. Ceci montre que, dans les grandes



Après une petite heure d'efforts, on arrive à avoir confirmation que la machine est toujours vivante. Si vous voyez ce texte, vous avez généralement fait le plus gros du travail et êtes quasi certain d'avoir correctement investi votre temps et votre argent.

lignes, la machine fonctionne (processeur, ULA, DRAM, etc.) et que la modification de la sortie vidéo est fonctionnelle.

La joie est cependant de courte durée, puisque quelques pressions sur les touches du clavier montrent que seules certaines d'entre elles fonctionnent. La bonne nouvelle est qu'il s'agit là d'un problème connu, découlant de l'usure de la membrane (*délaminage* et oxydation). La mauvaise, en revanche, est que cette membrane se trouve sous le clavier « gomme », lui-même maintenu en place par une « coque » en métal **collé** au boîtier plastique du ZX Spectrum.

Une nouvelle membrane se trouvera facilement sur eBay pour quelques euros. J'ai acheté un exemplaire neuf fabriqué en 2017, auprès d'un vendeur appelé *psycomsbid*, pour 10€. Le vrai problème est le décollage délicat de la partie métallique, car il faut arriver à le retirer sans le déformer. On pourra légèrement chauffer à l'aide d'un sèche-cheveux pour faciliter l'opération, tout en glissant un outil dans les interstices pour faire levier. C'est délicat et long, mais on finit par obtenir gain de cause. On peut ensuite retirer le clavier, puis la membrane pour la remplacer.

On en profitera pour passer le clavier « gomme » sous l'eau avec un peu de produit vaisselle et pendant qu'il sèche, méticuleusement retirer toute trace de colle de la partie métallique. Après un essai sommaire pour confirmer la réparation, il faudra ré-assembler en utilisant de l'adhésif double face, puis refermer le tout pour obtenir un ZX Spectrum comme neuf.

5. POUR FINIR

Le ZX Spectrum est une machine intéressante pour débuter en rétrocomputing, car elle ne vous coûtera pas grand-chose et la plupart des problèmes peuvent être corrigés facilement. Mais il ne faut pas perdre de vue que cette machine avait pour objectif de démocratiser l'informatique et de la rendre accessible au plus grand nombre. De ce fait, l'élément directeur dans sa création a principalement été la réduction des coûts et ceci impacte la qualité et la durabilité. Deux points faibles sont très représentatifs de cet état de fait : l'ULA regroupant en un seul composant bon nombre de fonctionnalités (son, vidéo, gestion DRAM, etc.) et la mémoire. Si l'un de ces points pose problème avec votre machine, il sera plus rentable de simplement la mettre de côté et d'en trouver une autre. Fort heureusement, beaucoup sont en vente et avec un peu de patience, vous trouverez votre bonheur.

En conclusion et de façon tout à fait subjective, je dirai que le ZX Spectrum est loin d'être une pièce maîtresse d'une collection, en termes de fonctionnalités et de ludothèque, mais simplement une machine facile à obtenir, tout en restant un classique du genre. **DB**

CONTRÔLER VOS MODÈLES LEGO AU JOYPAD À L'AIDE DE BRICKPI

Sébastien Colas

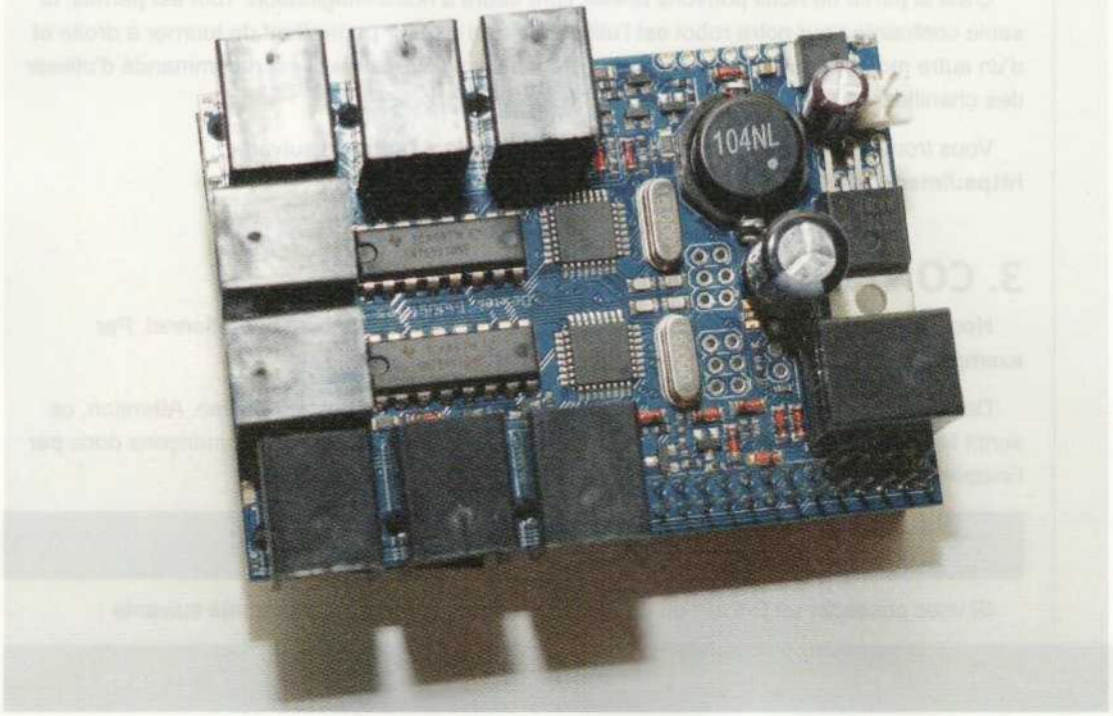
Dexter Industries propose des cartes d'extension pour Raspberry Pi. Dans cet article, nous allons nous intéresser à la carte BrickPi permettant de piloter les moteurs et senseurs Lego MindStorms. Nous verrons comment piloter notre robot Lego à l'aide d'un joypad. Nous développerons notre programme grâce au langage Python.

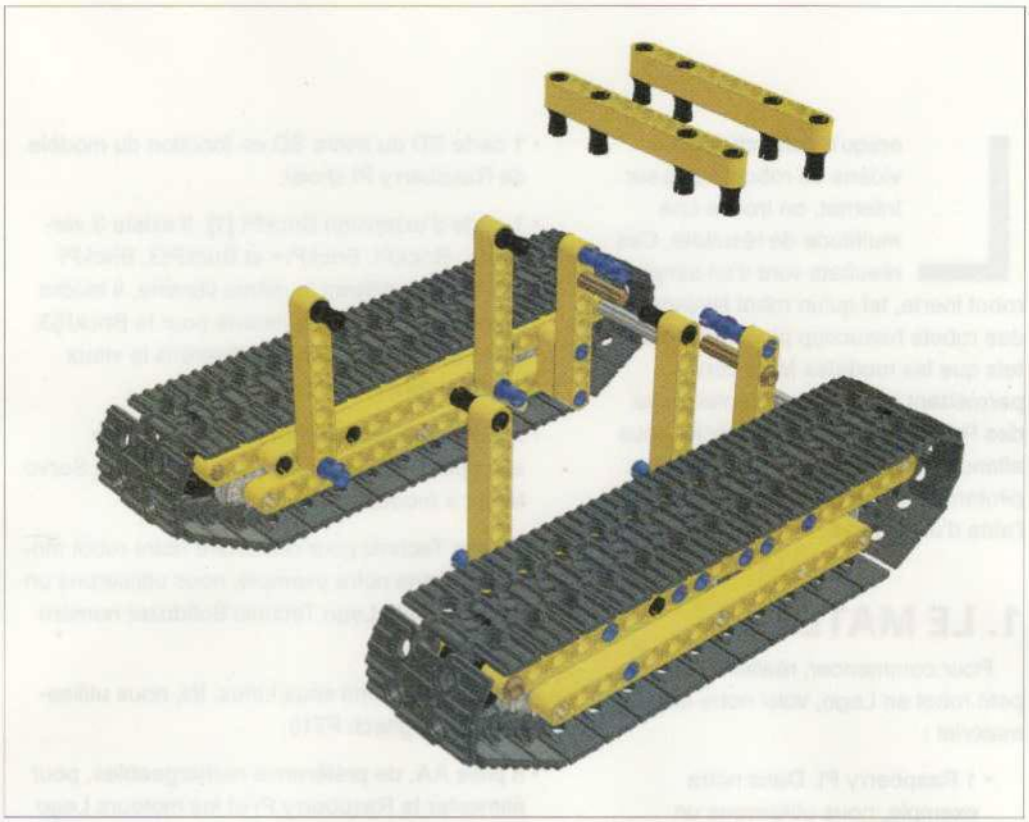
Lorsqu'on cherche des vidéos de robots Lego sur Internet, on trouve une multitude de résultats. Ces résultats vont d'un simple robot inerte, tel qu'un robot Ninjago à des robots beaucoup plus complexes, tels que les modèles Mindstorms permettant par exemple de résoudre des Rubicubes. Dans cet article, nous allons commencer en douceur en pilotant un robot Lego à chenilles à l'aide d'un JoyPad.

1. LE MATÉRIEL

Pour commencer, réalisons un petit robot en Lego, voici notre liste de matériel :

- 1 Raspberry Pi. Dans notre exemple, nous utiliserons un Raspberry Pi Modèle 1 B.
- 1 carte SD ou micro SD en fonction du modèle de Raspberry Pi choisi.
- 1 carte d'extension BrickPi **[1]**. Il existe 3 versions : BrickPi, BrickPi+ et BrickPi3. BrickPi et BrickPi+ utilisent la même librairie, il faudra prendre une nouvelle librairie pour le BrickPi3. Dans notre article, nous utiliserons le vieux modèle : BrickPi.
- 2 moteurs Lego MindStorms. Dans notre exemple, nous prendrons 2 « EV3 Large Servo Motor » modèles 45502.
- 1 Lego Technic pour construire notre robot motorisé. Dans notre exemple, nous utiliserons un vieux modèle Lego Technic Bulldozer numéro 6092224.
- 1 joypad reconnu sous Linux. Ici, nous utiliserons un Logitech F710.
- 8 piles AA, de préférence rechargeables, pour alimenter le Raspberry Pi et les moteurs Lego MindStorms.





2. CONSTRUCTION DU ROBOT LEGO

C'est la partie où nous pouvons laisser libre cours à notre imagination. Tout est permis, la seule contrainte pour notre robot est l'utilisation d'un moteur permettant de tourner à droite et d'un autre moteur pour tourner à gauche. Pour se simplifier la vie, il est recommandé d'utiliser des chenilles.

Vous trouverez sur le site MecaBricks notre modèle à l'adresse suivante :
https://mecabricks.com/en/models/79a8GWewv8w

3. CONFIGURATION LOGICIELLE

Nous supposerons ici que nous disposons d'un système Raspbian opérationnel. Par exemple, la version `2018-11-13-raspbian-stretch-lite.img` [2].

Dexter Industries propose un script d'installation complètement automatisé. Attention, ce script finit par redémarrer le Raspberry Pi. Le script n'installe pas `git`, commençons donc par l'installer :

```
sudo aptitude install git
```

Si vous posséder un BrickPi ou un BrickPi+, il faudra lancer la commande suivante :

```
sudo curl -kL dexterindustries.com/update_brickpi_plus | bash
```

Si vous posséder un BrickPi3, il faudra lancer la commande suivante :

```
sudo curl -kL dexterindustries.com/update_brickpi3 | bash
```

La dernière chose dont nous avons besoin, c'est de la librairie python `inputs` nous permettant de gérer notre joypad :

```
sudo pip install inputs
```

4. PRINCIPE ET FONCTIONNEMENT

Le but du programme que nous allons écrire est très simple : pouvoir piloter notre robot à l'aide d'un joypad.

Nous allons développer notre programme en Python. Il nous faut donc 2 bibliothèques :

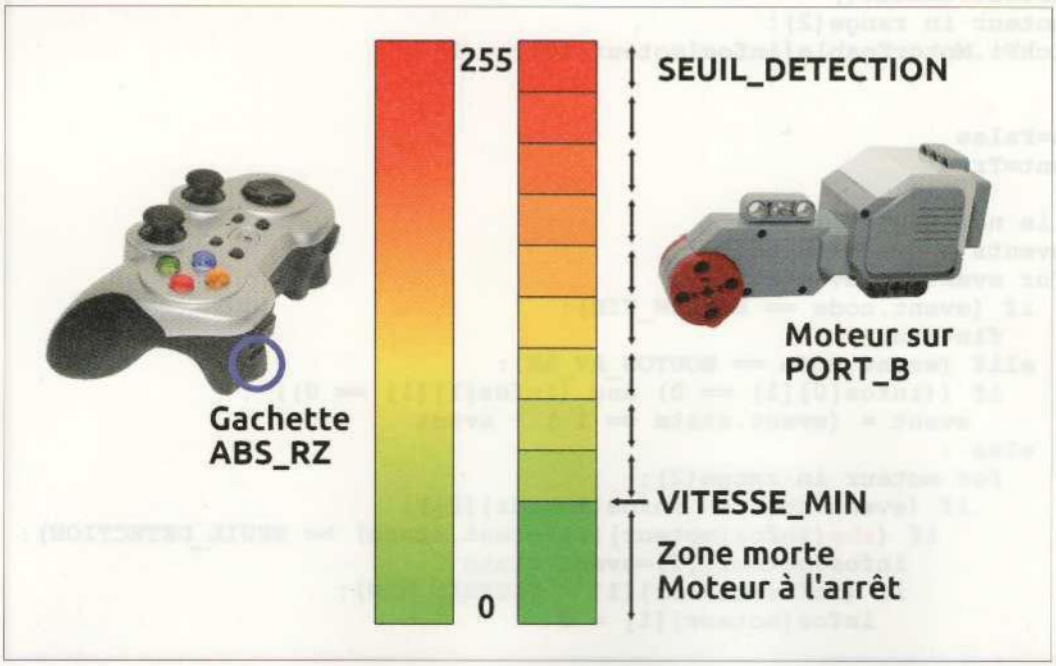
- `inputs` : pour nous permettre de récupérer les événements sur le joypad ;
- `BrickPi` : pour nous permettre de piloter les 2 moteurs de notre robot Lego.

Nous allons piloter le moteur gauche du robot par la gâchette gauche du joypad et le moteur droit par la gâchette droite.

Les valeurs de la pression sur la gâchette se situent dans l'intervalle 0 à 255. Valeurs retournées par la bibliothèque inputs.

Les valeurs d'entrée du moteur se situent entre -255 et +255 avec un moteur à l'arrêt à 0.

Nous pouvons donc transmettre directement les valeurs récupérées du joypad vers les moteurs.



Nous devons prendre en suite en considération 2 paramètres :

- la valeur minimale du moteur pour que celui-ci fasse avancer le robot (variable **VITESSE_MIN**) ;
- les valeurs de vitesse que l'on veut transmettre au robot. Dans l'absolu, nous pourrions tout transmettre, mais le temps de prise en compte de la vitesse par la librairie BrickPi induit de la latence. Nous définissons donc ici un seuil au-dessus duquel nous propagerons les ordres (variable **SEUIL_DETECTION**).

Vous pouvez visualiser tout ceci sur le schéma.

5. PROGRAMMATION

```
01: from inputs import get_gamepad
02: from BrickPi import *
03:
04: VITESSE_MIN=65
05: SEUIL_DETECTION=20
06:
07: BOUTON_FIN    = 'BTN_START'
08: BOUTON_AV_AR = 'BTN_SOUTH'
09:
10: infos = [[PORT_C,0,'ABS_Z'],[PORT_B,0,'ABS_RZ']]
11:
12: BrickPiSetup()
13: BrickPiSetupSensors()
14: BrickPi.Timeout=10000
15: BrickPiSetTimeout()
16: for moteur in range(2):
17:   BrickPi.MotorEnable[infos[moteur][0]] = 1
18:
19: try:
20:   fin=False
21:   avant=True
22:
23:   while not fin:
24:     events = get_gamepad()
25:     for event in events:
26:       if (event.code == BOUTON_FIN):
27:         fin=True
28:       elif (event.code == BOUTON_AV_AR):
29:         if ((infos[0][1] == 0) and (infos[1][1] == 0)) :
30:           avant = (event.state == 1 ) ^ avant
31:       else :
32:         for moteur in range(2):
33:           if (event.code == infos[moteur][2]):
34:             if (abs(infos[moteur][1]-event.state) >= SEUIL_DETECTION):
35:               infos[moteur][1]=event.state
36:               if (infos[moteur][1] < VITESSE_MIN):
37:                 infos[moteur][1] = 0
```

```
38:                 BrickPi.MotorSpeed[infos[moteur][0]] = infos[moteur][1]
* (1 if avant else -1)
39:                 BrickPiUpdateValues()
40:
41: except Exception as e:
42:   print("Joypad non trouvé" )
43:   print(e)
44:
45: for moteur in range(2):
46:   BrickPi.MotorSpeed[infos[moteur][0]] = 0
47:
48: BrickPiUpdateValues()
```

Voici une description du fonctionnement de notre programme.

En lignes 01 et 02, nous chargeons les librairies dont nous avons besoin.

En lignes 04 et 05, définition de la vitesse minimale pour un moteur, ainsi que du seuil de détection.

En lignes 07 et 08, définition des boutons que nous allons utiliser pour mettre fin au programme ou pour basculer de marche avant à marche arrière.

En ligne 10, nous définissons les informations pour faire fonctionner notre robot. Voici la description :

```
[[MoteurGauche,VitesseGauche,JoypadGauche],[MoteurDroite,VitesseDroite,
JoypadDroite]]
```

Des lignes 12 à 17, nous initialisons BrickPi. En ligne 17, nous spécifions où se trouvent les moteurs que nous voulons piloter (dans notre cas `PORT_C` et `PORT_B`).

En ligne 19, nous gérons le cas où il y aurait un problème, par exemple, l'impossibilité de trouver un joypad. Dans un tel cas, nous afficherons un message d'erreur : lignes 41 à 42.

En lignes 20 et 21, nous créons 2 variables, la première **fin** pour déterminer la fin du programme, la seconde **avant** pour savoir si le robot est en marche avant ou en marche arrière.

En ligne 23, nous bouclons tant que la fin du programme n'est pas déclenchée.

En ligne 24, nous récupérons les actions sur le joypad, nous traitons ces actions en ligne 25.

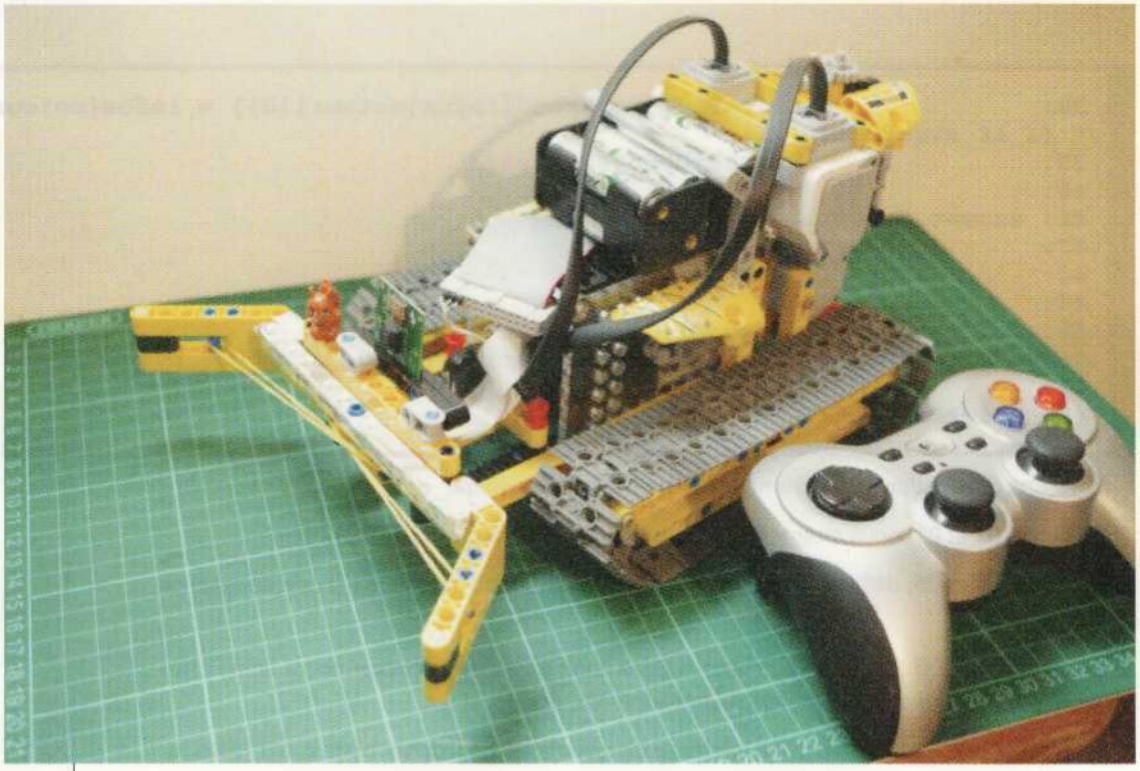
En lignes 26 et 27, nous détectons la fin du programme.

En ligne 28, 29 et 30, nous détectons si l'on passe la marche arrière.

En ligne 32 et 33 pour chacun des deux moteurs, si l'axe de joypad correspondant change de valeur, alors :

Si le seuil de détection a été dépassé (ligne 34), alors :

On enregistre la valeur de la nouvelle vitesse (ligne 35). Si cette valeur est en dessous de la valeur minimale, la vitesse du moteur reste à zéro (ligne 36).



On envoie les informations de vitesse et de sens au moteur et on applique les changements (lignes 38 et 39).

Entre les lignes 45 à 48 il s'agit de la fin du programme. On passe les vitesses de moteur à 0.

On pourra lancer automatiquement le programme au lancement du Raspberry Pi en rajoutant la ligne :

```
python /home/pi/robot.py &
```

dans le fichier `/etc/rc.local`. Il faudra rajouter la ligne juste avant le `exit 0`.

CONCLUSION

Maintenant que nous savons piloter notre robot à l'aide d'un joypad, l'étendue des possibles est infinie :

- rajouter la camera Raspberry à notre Pi ou bien une webcam et piloter notre robot à l'aide de reconnaissance d'image en OpenCV ;
- rajouter des senseurs MindStorms : palpeurs, détecteurs de distance... pour détecter les obstacles ;
- utiliser GPIO pour créer des détecteurs en tout genre : obstacle, distance...

Et pour finir, n'oubliez pas de partager vos créations. **SC**

RÉFÉRENCES

[1] BrickPi sur le site Dexter Industries : **https://www.dexterindustries.com/brickpi/**

[2] Site de téléchargement de RaspBian : **https://www.raspberrypi.org/downloads/raspbian/**

THEY ARE BACK !

LE HACK 19

SAME PLAYER SHOOT AGAIN

6 - 7 JUILLET 2019

CITÉ DES SCIENCES ET DE L'INDUSTRIE

PARIS

WWW.LEHACK.ORG

APRÈS AVOIR ORGANISÉ LA NUIT DU HACK PENDANT 16 ANS, HZV REVIENT AVEC UN NOUVEL ÉVÈNEMENT. LE NOM CHANGE, LES VALEURS RESTENT.