N°1 | JUILLET-AOÛT 2014

HACKABLE MAGAZINE

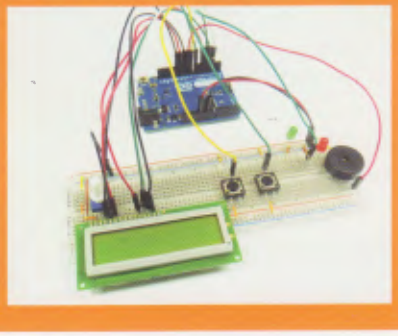
DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

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

ARDUINO

Ardu-sonnette : A-t-on sonné en votre absence ?

p. 68



RASPBERRY PI

Votre Pi toujours à l'heure grâce à une horloge temps réel DS1338

p. 56

B.A.BA

Comprendre et utiliser la Loi de Ohm

p. 46

ARDUINO

Transformez votre Arduino en programmeur de microcontrôleurs

p. 14

REJOIGNEZ LA RÉVOLUTION ARDUINO !

Tout pour s'équiper, apprendre et démarrer ! p. 22

HACK

Démontons, explorons et étudions une e-cigarette

p. 88

OUTILS

Choisir et acheter sa station de soudage sans se tromper

p. 08

L 19338 - 1 - F: 7,90 € - RD

ACTUELLEMENT À DÉCOUVRIR !

LINUX SUR ARM CORTEX-M4 STM32F429 !

OPEN SILICIUM N°11

DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

boutique.ed-diamond.com

ÉDITO

Exploratrice ! Explorateur !

Vous qui venez de prendre ce magazine en main, voilà ce que vous êtes !

Soyez rassuré, vous voici en bonne compagnie car nous le sommes aussi et le clamons haut et fort. Beaucoup sont ceux qui ne cherchent plus à savoir et se contentent d'utiliser ce que l'industrie leur met entre les mains, ignorant les mystères et la magie qui se cache derrière les apparences. Fort heureusement, il semblerait que vous n'en fassiez pas partie, la preuve vous venez de prendre un magazine à la couverture remplie d'assemblage farfelus de fils, de cartes et de composants... Vous voici pris la main dans le sac !

Vous êtes-vous déjà arrêté devant une porte automatique vous demandant comment tout cela fonctionne ? Votre regard est-il attiré par ces panneaux d'affichage lumineux suscitant en vous nombre de questions ? Vous arrive-t-il de vous interroger sur ce qui est mis en oeuvre avec cette carte qui vous permet d'accumuler des points « fidélité » ou vous donne accès à votre piscine municipale ? Vous questionnez-vous sur la façon dont le bus signal son retard estimé ou comment votre smartphone fait pour connaître sa position au mètre près ?

Il existe un monde derrière le monde et vous le savez. C'est ce qui vous pousse à chercher à comprendre et à vouloir expérimenter par vous-même. Nous ne sommes plus une majorité à fonctionner ainsi mais ceci ne signifie pas pour autant que nous ne sommes qu'une poignée, bien au contraire. Alors même que la plupart consomme sans comprendre et sans même se demander « comment », « pourquoi » et « qu'est-ce », il en est qui ne veule pas simplement qu'on leur livre la technologie clé en main. Ils veulent, nous voulons, vous voulez y prendre part ! Je sais, exposé ainsi, cela peut paraître un peu hollywoodien mais il n'y a pas de pilule rouge ou bleu, ni même de Morpheus ou de Trinity (des agents par contre, il y en a partout). Tout est bien plus simple qu'au cinéma, le lapin blanc, c'est votre curiosité.

Paradoxalement, c'est en ces temps où la technologie est glissée furtivement partout, de plus en plus insidieuse et complexe, qu'il est devenu aisé de devenir un initié. Grâce à des initiatives indépendantes, des projets universitaires et tantôt des actions d'industriels importants, aujourd'hui plus que jamais, chacun peu, s'il le souhaite, s'éveiller aux mystères qui se cachent à l'envers du décore.

SUITE PAGE 4...

Hackable Magazine
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND
STIMULANT DE MATIÈRE GRISE

B.P. 20142 – 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – Fax : 03 67 10 00 21
E-mail : lecteurs@hackable.fr
Service commercial : cial@ed-diamond.com
Sites : boutique.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Kathrin Scali
Responsable publicité : Valérie Fréchard,
Tél. : 03 67 10 00 27 v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20

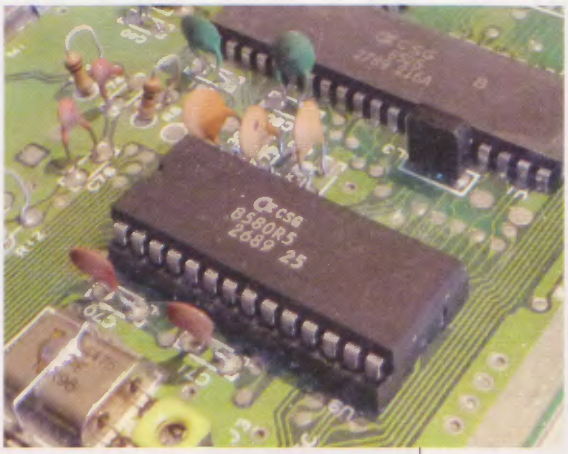
Impression : pva, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12

Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution,
N° ISSN : en cours
Commission paritaire : en cours
Périodicité : bimestriel
Prix de vente : 7,90 €

Suivez-nous sur Twitter
@hackablemag

SOMMAIRE

...SUITE ÉDITO



Gros plan sur la puce SID 8580R5 (successeur du 6581) de mon C64. Le marquage « 2689 » correspond à la date de fabrication, la semaine 26 de l'an 1989. Pure coïncidence, cette puce a été fabriquée il y a précisément 25 ans, fonctionne toujours et peut être utilisée aujourd'hui dans un montage Arduino !

C'est là une opportunité à ne pas manquer car à l'instar de ce qui s'est passé au début des années 80, nous sommes aujourd'hui à la croisé des chemins et nous pouvons choisir ce que nous ferons de cette technologie. L'arrivée de l'ordinateur personnel nous a donné l'opportunité de prendre part à une révolution plutôt que d'en être simplement les témoins. Dans les années 90, c'est la révélation d'Internet au public qui nous a présenté une nouvelle opportunité d'obtenir la maîtrise de l'information et de la technologie.

Depuis la roue a tournée, les ordinateurs personnels d'antan ont laissé place à des PC de plus en plus difficile à contrôler. L'Internet d'AOL, de GeoCities, des newsgroups et des connexions par modem chez des fournisseurs locaux sont devenus l'Internet omniprésent des gros FAI et opérateurs de téléphonie mobile, fief de mégastructures commerciales du nom de Google, Facebook, Apple, Microsoft, Amazon...

L'électronique numérique d'aujourd'hui nous offre les mêmes alternatives et il ne tient qu'à nous de faire le bon choix, et de faire de cette opportunité une réussite pour nous-même, en gardant, cette fois, la maîtrise et le caractère humain, social et collaboratif propre à cette (r) évolution.

Pourquoi prendre le temps et investir argent et énergie dans une telle exploration ? Cette question appelle en vérité bien des réponses. La première qui vient à l'esprit tient en quelques mots : « parce que c'est plaisant » (le terme « jouissif » convient mieux, mais je ne veux pas vous faire peur). Le propre du genre humain est sa capacité à apprendre et transposer une expérience hors de son contexte afin de mettre en oeuvre le savoir ainsi acquit dans une autre situation. Nous sommes conçu pour cela dès la naissance et même si certains perdent une grande partie de cette capacité, c'est un des mécanismes qui nous définit. Notre cerveau a donc pour habitude de nous récompenser lorsque nous achevons quelque chose ou que nous apprenons (aaah, la dopamine). Le plaisir est donc l'un des facteurs les plus importants.

Une autre réponse, qui découle naturellement de la première, est le bon sens lui-même. On ne sait pleinement se servir de quelque chose que lorsqu'on en comprend réellement le fonctionnement. Mais au delà de l'usage lui-même, nous avons d'autres savoirs très utiles qui en découlent. En sachant comment fonctionne quelque chose vous devenez à même de le modifier pour l'adapter à vos besoins et vos envies spécifiques. Mieux encore, vous devenez capable, en cas de changement dans le fonctionnement, de faire en sorte que l'objet retrouve son comportement initial. En d'autres termes, si ça ne marche plus, vous voici capable de le réparer, tout simplement parce que vous comprenez comment tout cela marche ! Ainsi, comme vous devenez à même de contrôler le fonctionnement de ce qui vous entour. Vous gagnez en autonomie et en esprit critique. En augmentant vos connaissances technologiques et en expérimentant par vous-même, vous vous détachez du mécanisme consistant à devoir faire appel à une tierce personne pour vos besoins technologiques. Mieux encore, vous disposer des outils pour évaluer une technologie. Vous n'êtes plus obligés de reposer sur des mécanismes naïfs (faute d'autres données tangibles) consistant à se dire que le TrucPhone 5 est forcément mieux que le TrucPhone 4 parce qu'il possède un capteur de noisette de 12 méga-écureuil au lieu de 8 et de ce fait que vous **devez** en changer (le TrucPhone 5 est mieux, c'est sûr, ils le disent dans les pubs).

Il n'y a donc pas que l'aspect ludique derrière le fait de faire clignoter une led avec un Arduino, il y a tout une chaîne de causalité qui se met en place, vous rendant de plus en plus autonome et prompt à détecter des anomalies ou des défauts de conception. Vous gagnez littéralement en force car, comme on le sait, la connaissance c'est le pouvoir et qu'avec maîtrise du savoir vient la maîtrise de son environnement.

Au delà, de la « vérification » des technologies que vous « consommez », nous avons la confiance que vous pouvez leur accorder. Des initiatives comme Arduino, Raspberry Pi ou d'autres, en vous donnant l'occasion de créer vos objets intelligents et vos propres morceaux de technologie, constituent

Ci-dessus, la manière dont le constructeur (Cherry) voyait sa souris Gentix illuminated. À droite, comment moi je préfère ma souris Gentix. Entre les deux, un coup de tournevis, un fer à souder et quelques leds.

Élément indispensable pour apprendre à ranger ce qui traîne et avoir plein de choses à réparer : un ou plusieurs animaux de compagnie. Je vous présente monsieur Copper, grand amateur de jouets électroniques en tous genres et de dispersion de composants.

des briques de base vous permettant de construire un environnement numérique dans lequel vous pouvez avoir confiance. Pourquoi ? Tout simplement parce que c'est vous qui aurez créé cet environnement et pouvez le changer comme vous l'entendez.

Enfin, comme l'a dit Arthur C. Clarke, les déchets solides ne sont qu'une matière première que nous sommes trop stupides pour pouvoir utiliser. C'est vrai au sens général mais aussi et surtout dans le domaine des équipements électroniques qui nous entourent. Nos vieux PC, téléphones mobiles, téléviseurs, fours micro-onde, lave-linge... deviennent des DEEE (Déchets d'Équipements Électriques et Électroniques) au rythme de quelques 20 kg par personne et par an en France. Des DEEE ? Non ! Il n'existe rien de tel. C'est juste un gaspillage de composants et de matière première que nous ne savons pas utiliser.
Je vous le dis comme je le fais moi-même : récupérez tout ce que vous êtes en mesure de récupérer. Et le reste, gardez-le quand même. Ce que vous ne savez réutiliser aujourd'hui, demain, peut-être vous servira. Vous vous en serez reconnaissant plus tard et les pays « poubelles » où transitent et finissent la majorité de nos DEEE vous en seront reconnaissant de suite.

*Recette de décoration :
Une sculpture fantasy, des coups de meuleuse, un hack pour les yeux translucides, quelques leds, un module Bluetooth, un microcontrôleur et de l'imagination. Résultat : une tête de dragon au-dessus de la porte d'entrée avec des yeux lumineux multicolores programmables en Bluetooth. Amusant, impressionnant et surtout unique !*

Vous voulez un exemple ?
Le Commodore 64 est le modèle d'ordinateur qui a été le plus vendu au monde. Il contient une puce, le MOS 6581 également appelé SID qui, éventuellement couplée à un Arduino par exemple, permet de créer un synthétiseur de sons très ludique et pédagogique. Le MOS 6851 n'est plus fabriqué, c'est une puce de légende et ces anciens composants sont actuellement vendu sur eBay quelques 30 euros. 17 à 25 millions de Commodore 64 ont été vendus et la majorité d'entre eux ont été détruit ou sont entrain de pourrir sur une pile de détritus en Afrique. Et vous, si vous n'en possédez pas et que vous souhaitez construire un tel synthétiseur, vous allez devoir débourser 30 euros pour le SID ou 50 euros pour C64 complet, le tout sans savoir s'il fonctionne...

En vérité, je vous le dis, on nous a menti. Oh pas directement mais l'idée ne s'est pas installée d'elle-même dans nos têtes. Cette idée qui, une fois énoncée à haute voix, est pourtant complètement folle : « les équipements électroniques sont des biens consommables ». On achète et on utilise jusqu'au prochain modèle à la mode et puis on jette. Plus ce cycle est court, plus le marché se porte bien, plus on perd de matière première et moins on est capable de comprendre et de réparer. Un appareil électronique n'est **PAS** un consommable. Nous ne sommes donc pas des consommateurs de technologie. Nous achetons des matériels et nous en faisons ce qui nous chante, parce que ce sont des **biens**, pas du papier toilette, du savons ou du concentré de tomate. Avec un peu de patience et des connaissances de base, changer un écran cassé sur un smartphone est à la porté d'un enfant de 12 ans, en France comme en Chine.

C'est tout cela qui constitue l'esprit et l'âme de ce mouvement et de cet engouement actuel pour l'électronique numérique. Hackable magazine en est le reflet et partage ces valeurs. La technologie est à nous tous et il est largement temps d'en reprendre possession et de s'amuser ! Et puisque l'occasion aujourd'hui se présente, saisissons-là !

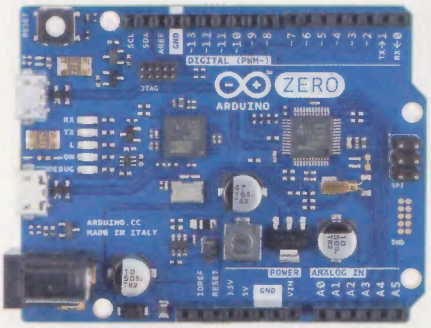
Si vous avez des remarques, des attentes, des idées ou des critiques, n'hésitez pas à nous en faire part en nous envoyant un petit message à **lecteurs@hackable.fr**. ■

Denis Bodor

ARDUINO ZERO : LA NOUVELLE GÉNÉRATION D'ARDUINO

Découlant d'un partenaria entre Arduino et Atmel (constructeur des microcontrôleurs qui équipe les Arduino), arrivera bientôt chez les revendeurs une nouvelle carte : l'Arduino Zero.

Alors que les classiques du genre reposent sur un microcontrôleur Atmel AVR, la Zero utilise une nouvelle génération de puce de chez Atmel ATSAMD21G18 qui, comme son nom l'indique fait partie non plus de la gamme AVR mais de la famille SAMD21. Ce composant est un coeur ARM Cortex-M0 32-bit à 48 Mhz comprenant 256 Ko de flash et 32 Ko de SRAM (et 16 Ko d'EEPROM). Bien plus puissant et riche en ressources qu'un AVR 8-bit !



La carte, comme l'Arduino Due, fonctionne en 3,3 volts et dispose de 6 entrées analogiques (sur les 14 disponibles sur le microcontrôleur), 14 sorties (dont 12 avec PWM) mais aussi et surtout un convertisseur numérique/analogique (DAC) ce qui peut être plutôt amusant, car avec une telle puissance, on peut envisager des applications audio intéressantes. La carte dispose également de deux connecteurs USB, un pour la programmation et l'autre pour les croquis de l'utilisateur.

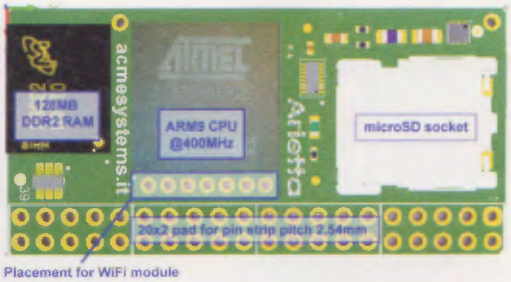
Il semblerait que cette carte soit destinée, à terme, à remplacer la classique Arduino UNO et est, à l'heure actuelle, la plus populaire de toutes. Ce renouvellement espéré explique peut-être le nom de la carte. La participation d'Atmel dans la conception est également un changement important puisque le constructeur n'a jamais été historiquement impliqué directement dans le projet.

Aucun prix et aucune date n'est encore annoncé officiellement. On ne peut qu'espérer qu'il avoisine ceux d'une Arduino Uno car la concurrence, en terme de fonctionnalité, commence à se faire très présente et pressante.

► **http://arduino.cc/en/Main/ArduinoBoardZero** ▌

MADE IN ITALY... AUSSI

La nouvelle carte de la société italienne ACME systems, l'Arietta G25, devrait être disponible au moment où vous lirez ces lignes. Initialement proposée en pré-commande à environ 10 euros, son prix définitif est maintenant de 20 ou 30 euros selon la version et la quantité de mémoire (128 ou 256 Mo).



À ce prix vous disposez d'un module construit autour de la puce Atmel AT91SAM9G25 à 900 Mhz avec 128 ou 256 Mo de mémoire DDR2. Contrairement à la Raspberry Pi, il ne s'agit pas d'un nano-ordinateur, mais d'une plateforme à faible coût destinée aux Makers. L'idée est davantage de voir la chose comme un super-Arduino que comme une Pi car il n'y a pas de sortie vidéo (HDMI) ni d'interface réseau (Ethernet).

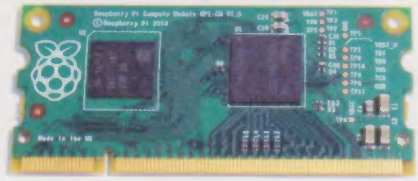
Les principaux avantages de cette carte, en dehors de son faible coût résident dans le support de Linux en version standard (et non une version modifiée par le fabricant), une copieuse documentation de la part d'Atmel (quelques 1200 pages) et une ouverture complète des sources logicielles (bootloader et pilotes inclus).

Nous sommes impatient de mettre la main sur le produit car nous avions déjà beaucoup apprécié la précédente carte d'ACME, l'Aria G25 même si celle-ci était bien moins évidente à utiliser. Elle se présentait, en effet, sous la forme d'un module de 4 cm de côté qu'il fallait souder sur un circuit "maison" ou sur la carte mère proposée par ACME Systems.

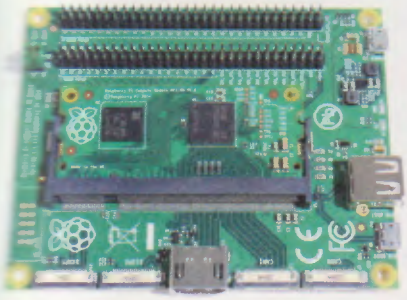
L'Arietta, quant à elle, est directement utilisable, un peu à la manière d'un Arduino avec une série de connecteurs facilement accessibles.

► **http://www.acmesystems.it/arietta** ▌

RASPBERRY PI COMPUTE MODULE : COMME UNE RASPBERRY PI II



Nul doute que la Raspberry Pi a définitivement et profondément changé le domaine de l'informatique pour tous et la fondation éponyme semble s'apprêter à vouloir faire de même avec le monde de l'embarqué. En effet, la nouvelle carte Raspberry Pi, nommée Raspberry Pi Compute Module, repose sur un cahier des charges bien différent de l'initiative de départ.



Cette nouvelle carte se compose de deux éléments, un module construit autour d'un processeur (SoC) Broadcom BCM2835 intégrant 512 Mo de mémoire mais surtout de la flash permettant d'installer le système sans avoir recours à une carte SD ou microSD. Cette espace supplémentaire, qui n'existe pas sur la Raspberry Pi, est de 4 Go et permettra donc d'embarqué le système dans un ensemble fermé. Le module se présente sous la forme d'un circuit se plaçant dans un *slot* au format SODIMM, le même que celui des emplacements pour RAM DDR2 des ordinateurs portables (non, le module ne s'insère pas dans un tel emplacement, c'est juste un format de connecteur courant).

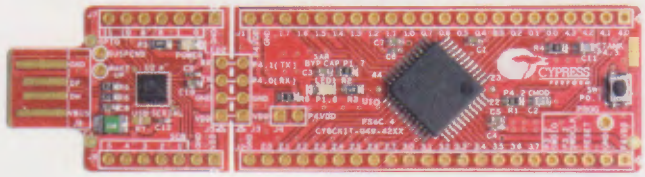
Le module est destiné à s'enficher sur une carte d'accueil qui propose alors l'alimentation et la connectivité USB, HDMI, etc. C'est le *Compute Module IO Board*. Celui-ci propose bien plus d'entrés/sorties que l'actuelle Raspeberry Pi et l'objectif est donc évident. Il s'agit de proposer une solution plus professionnelle autour du concept Raspberry Pi et de donner l'opportunité à ce produit de pénétrer le marché de l'embarqué et de l'informatique industrielle. La séparation module/carte permettra ainsi la construction d'autres cartes d'accueil par des initiatives communautaires ou industrielles. En tant que particulier isolé, il reste cependant peu probable d'arriver au stade de la production d'une carte d'accueil, la connectique SODIMM impliquant la réalisation de circuits d'une finesse particulière.

Il est probable qu'au moment de la parution de ce magazine le module comme la carte d'accueil (formant le "*Raspberry Pi Compute Module Development Kit*") soient déjà disponible chez RS et element14 au prix annoncé de $30 (sans doute transformé par la magie du commerce international en 30 € HT, soit 36 € TTC).

Le duo pourra remplacer avantageusement une Raspberry Pi classique pour qui manque cruellement de ports d'entré/sortie et nous ne manquerons pas de revenir sur le sujet de manière plus pratique en mettant la petite bête à l'épreuve...

► **http://www.raspberrypi.org/raspberry-pi-compute-module-new-product/** ▌

CYPRESS : ENCORE UN CONSTRUCTEUR QUI ENTRE DANS LA DANSE

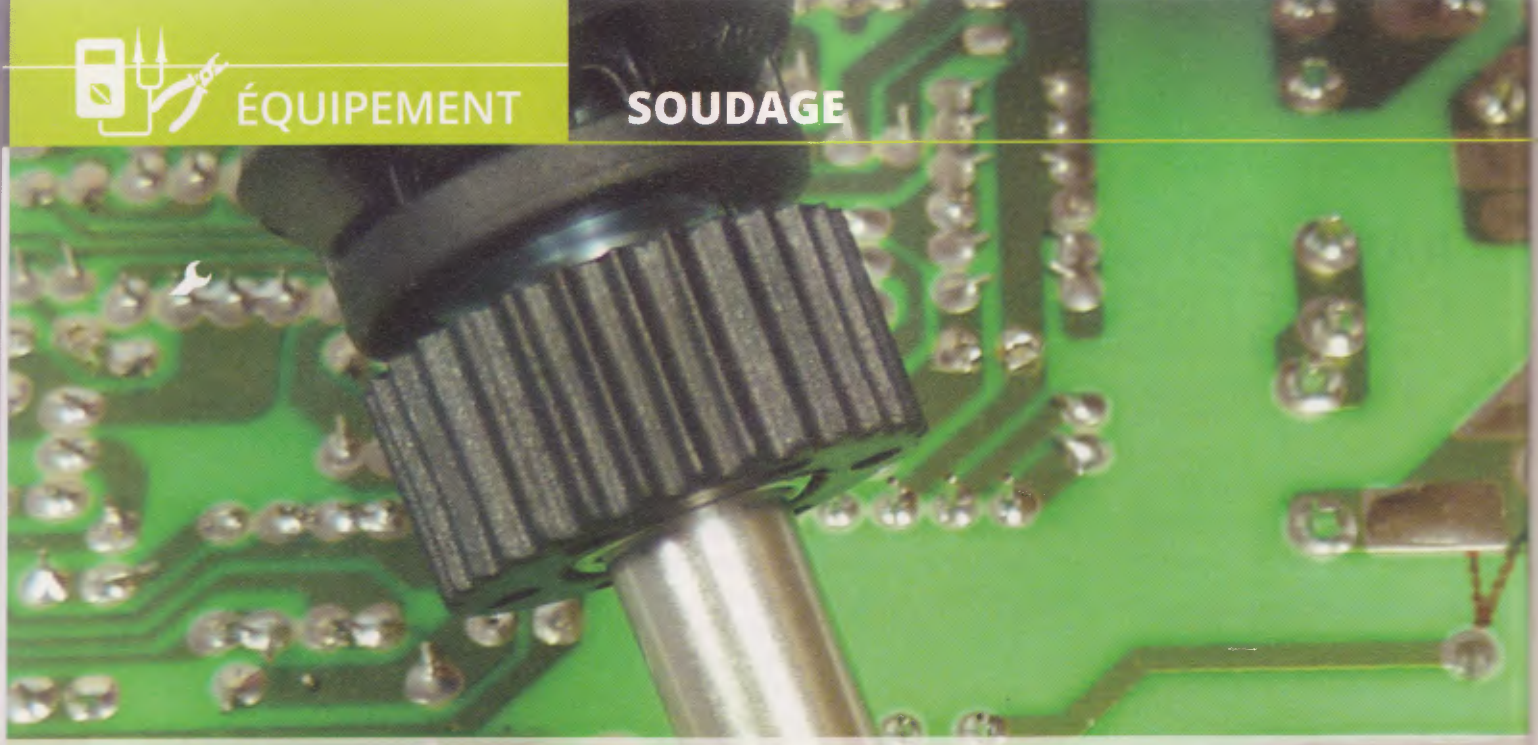


Cypress n'est sans doute pas très connu des hobbyistes mais c'est l'un des acteurs majeurs du monde des microcontrôleurs qui, de plus, intègre un certain nombre de spécificités dans ses produits comme, par exemple, des éléments analogiques (opamp) reconfigurables. Cypress semble décidé à ne pas rater la vague *uino qui déferle actuellement dans le nom de l'électronique.

Pour preuve, le PSoC CY8CKIT-049, un tout nouveau produit débarquant sur le marché, intégrant un microcontrôleur CY8C42xx, digne représentant de la famille PSoC 4 : ARM Cortex-M0 à 48 Mhz, 32 Ko de flash, 4Ko de SRAM, ADC, une quarantaine d'entrées/sorties. Le tout avec une connexion USB/série (CY7C65211) et un bootloader embarqué de base. Le kit est donc autonome et ne nécessite pas de programmeur et vendu... 4 euros TTC ! Oui, vous lisez bien, 4 euros.

Utilisable avec l'environnement de développement PSoC Creator du constructeur, ceci en fait, sauf erreur de notre par, tout simplement le kit de développement et d'évaluation ARM Cortex-M (32 bits) le plus accessible qui soit. Nous en reparlerons très certainement dans un prochain numéro.

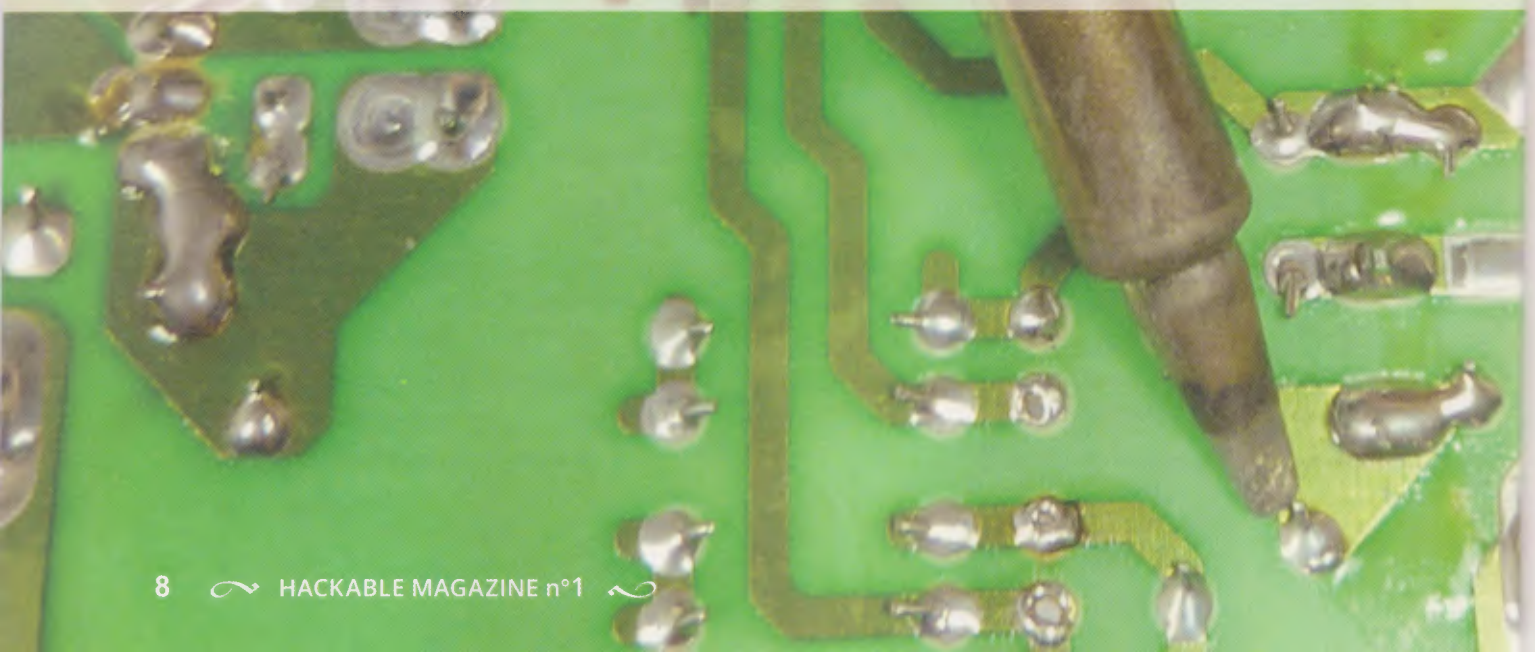
► **http://www.cypress.com/?rID=92146** ▌



LA STATION DE SOUDAGE : VOTRE AMIE POUR LA VIE !

Denis Bodor

Les breadboards et les modules c'est bien, surtout pour expérimenter et prototyper. Mais arrive un moment où le fer à souder devient un compagnon indispensable, aussi bien pour assembler que pour désassembler ou réparer. Contrairement à une idée reçue, bien souder n'est pas difficile, bien au contraire. Il est même relativement difficile de mal souder, dès lors qu'on dispose des bons outils.



Un fer à souder ou plus exactement une station de soudage est un outil indispensable à l'électronicien. Il permet d'assembler ses kits, de récupérer des composants, de faire des réparations, de modifier le matériel, et bien plus encore. Avec le multimètre et le petit outillage (tournevis, pinces, etc) c'est l'un des « indispensable », des *must-have*... De ce fait, c'est un matériel qu'il ne faut pas choisir au hasard. Du moins si vous ne comptez pas en changer tous les 2 ans.

1. LE BUDGET

Parlons peu, parlons bien : une station de soudage convenable vous coûtera entre 150 et 300 euros. C'est un investissement et c'est très clairement ainsi qu'il faut le voir.

Mais pour ce prix, on parle d'un matériel que vous garderez et utiliserez pendant 25 ans ou plus. Le calcul est vite fait, si vous optez pour du matériel bas de gamme à 30 euros, en dehors de son niveau de qualité médiocre, vous pourrez espérer vous en servir quelques années avant qu'il ne montre des signes de faiblesse ou ne vous rende fou par son comportement douteux. Vous en achèterez alors un autre, du même type et ainsi de suite. Finalement, après ne nombreuses années de tourment, vous vous rendrez compte qu'il était bien plus économique de faire un investissement initial plus important dans un matériel que vous pouvez presque garder à vie !

Nous parlerons ici de **station de soudage** et non de **fer à souder**. Un fer à souder est un outil simple qui se connecte directement à une prise de courant ~230V et qui généralement ne dispose d'aucun contrôle de température. Le fer à souder est un outil générique souvent plus adapté à des travaux d'électricité que d'électronique. Son domaine est la soudure de câbles, les prises, de grosses connectiques et autres éléments du même genre, mais pas vraiment de composants sur un circuit imprimé. Si vous ne comptez pas investir le budget adéquate dans une station de soudage digne de ce nom, pour diverses raisons, vous pouvez éventuellement vous rabattre sur un simple fer à souder de marque (mon JBC-30s m'a rendu de fiers services durant des années). Si votre budget est limité, mieux vaut choisir un bon fer de marque qu'une station de soudage chinoise bas de gamme au même prix.

Une station de soudage est un outil qui dispose d'un ajustement de température et est un ensemble régulé et contrôlé. Il ne s'agit pas d'une simple résistance chauffante branchée au courant domestique. Une station intègre de l'électronique qui gère et régule précisément son comportement.

L'importance de choisir une marque réputée tient également dans la disponibilité des pièces à long terme et de la compatibilité entre l'appareil et ses accessoires. Une station de soudage est un instrument de précision et un matériel de qualité résulte d'études précises en termes de thermodynamique et de choix des matériaux. La sécurité est également un élément clé. Il n'est pas rare de voir du matériel bas de gamme faire l'impasse sur des principes fondamentaux en terme d'isolation ou de protection. Ceci est particulièrement valable pour le matériel en provenance directe d'Asie vendu via des sites d'enchères en ligne.

L'étain de bonne qualité, avec la station et la panne est l'un des trois éléments qui font le succès d'une soudure réussie. Ici de l'étain 60% étain / 40% plomb avec un coeur résine (flux décapant) de 0,5 mm de diamètre, de chez Kester.

Le type et l'état exemplaire d'une panne bonne à mettre à la poubelle. Ce modèle de panne à pointe conique fine est presque totalement inutile pour des travaux d'électronique courants.

La tresse à dessouder, votre partenaire idéal pour récupérer des composants et/ou réparer des circuits. Son utilité est de retirer l'étain de soudure par capilarité. Une autre solution est la pompe à dessouder, on aime ou on n'aime pas.

Achetez votre matériel chez un revendeur connu, ayant « pignon sur rue » et fournissant une garantie, car vous aurez ainsi l'assurance qu'il s'agit de matériel original et non d'une contrefaçon à la durée de vie limitée et dont l'utilisation peut s'avérer dangereuse pour votre matériel et pour vous.

Parmi les marques réputées on trouve Hekko, Antex, Weller, Metcal et bien d'autres (j'en oublis sans doute, personnellement j'utilise principalement un Weller WSD-81 au bureau comme à titre personnel).

2. LES CARACTÉRISTIQUES IMPORTANTES

Le travail principal d'une station de soudage est de faire chauffer un élément appelé **panne** de manière à faire fondre un alliage à base d'étain (qu'on nomme généralement simplement « étain ») qui sert à réaliser un brasage qu'on désigne souvent par le terme **soudure**. En toute logique, c'est donc cette capacité à produire de la chaleur et celle de la transférer qui sont les points clés d'une bonne soudure et donc d'un bon matériel.

On parle généralement de capacité thermique pour quantifier « la possibilité qu'a un corps d'absorber ou restituer de l'énergie par échange thermique » (merci Wikipédia). Le brasage consiste à chauffer les éléments qu'on souhaite assembler pour ensuite y faire fondre l'étain. Il y a donc un transfère d'énergie thermique entre l'outil et les éléments (pattes des composants et circuit). L'outil doit être en mesure de transférer cette énergie dans un volume suffisant pour que la température de l'ensemble ne chute pas et rende ainsi la fusion de l'étain trop difficile. Il faut que le fer accumule de l'énergie thermique et la renouvelle de manière optimale.

Mais le transfère de cette énergie dépend aussi et surtout du point de contact ou, dans les faits, de la **panne** elle-même (le bout du fer). Fût un temps, ces pannes étaient simplement faites de cuivre car conduisant très bien la chaleur. Cependant, la montée en température favorise à corrosion (oxydation) et aujourd'hui une bonne panne est un élément d'ingénierie de pointe, composée d'un coeur faisant office de réservoir à chaleur et de nombreuses couches de matériaux divers (céramiques, alliages, etc).

La panne d'un fer est l'élément clé pour une bonne soudure. Une station hors de prix, d'excellente qualité, avec un fer équipé d'une panne de mauvaise facture ou usée découlera sur des mauvaises soudures. Inversement, un fer médiocre, équipé d'une panne de haute qualité pourra, dans certaines conditions, vous permettre des soudures parfaites. L'idéale est, bien entendu, un bon outil avec une bonne panne.

La forme de la panne est également très importante. Tantôt le matériel acheté arrive équipé d'une panne conique à pointe fine. Ce type de panne, pour un usage général est totalement sans intérêt. Préférez les pannes plates avec une largeur d'environ 2 mm, de préférence en plusieurs exemplaires et surtout correspondant à votre marque et modèle de station de soudage. En effet, d'une part les pannes d'une marque ne sont généralement pas mécaniquement compatibles avec une autre mais, de plus, le couple fer/panne doit former un ensemble cohérent et symbiotique. Utilisez toujours des

L'une des nombreuses stations de soudage très populaires, de bonne qualité et d'un fabricant réputé : une Weller WSD 81 version Europe.

pannes de marque, jamais des compatibles. Généralement le budget pour ce genre d'accessoires est de l'ordre de 5 à 10 euros pièce.

Une bonne station et de bonnes pannes représentent ensemble 60% d'une soudure réussie.

3. LES CONSOMMABLES

Le troisième élément clé pour de bonnes soudures est l'étain. Malgré le nom généralement utilisé, l'étain à souder n'est pas composé uniquement d'étain. Il s'agit généralement d'un mélange de 60% d'étain et de 40% de plomb. Certains types d'étain à souder contiennent également de petites quantités de cuivre, d'argent, de bismuth ou encore d'antimoine. Il se peut aussi que les proportions étain/plomb soit sensiblement différents (63%/37% par exemple).

Le point de fusion de l'étain à souder se trouve entre 180°C et 190°C même si on règle généralement la température de la panne entre 350°C et 400°C pour la plupart des travaux de soudure. Ceci est justement en rapport avec la capacité thermique et l'énergie transférée au point de soudure. Notez au passage que la plupart des composants disposent dans leur documentation d'une température et d'une durée maximum de soudure définie par le constructeur. Même si 350°C est une valeur générique, il est toujours de bon ton de se pencher sur la documentation pour des composants particuliers.

Comme pour les pannes et l'équipement lui-même, il convient de ne pas utiliser de produits bas de gamme ou de provenance douteuse. Il existe un certain nombre de marques connues généralement à plébisciter : Kester et Multicore sont deux d'entre elles.

Deux autres éléments importants sont à prendre en compte pour le choix de ce consommable :

- Le diamètre du fil : Le but est de contrôler avec précision la quantité d'étain appliquée pour souder. Il apparaît donc évident que plus le fil est fin, plus il est aisé de précisément jauger la quantité que l'on utilise. Evitez comme la peste les fils de 1 mm de diamètre ou plus, vous allez faire des « pâtés » sauf si vous comptez uniquement souder des gros transformateurs et autres choses énormes du même genre. 0,5 mm est un format qui convient généralement à la plupart des usages.
- Utilisez du fil dit *« resin core »* ou « avec flux ». Le flux est un décapant réagissant avec la chaleur. Son travail est de nettoyer les éléments soudés avant que l'étain n'y coule. Il faut savoir, en effet, que plus le temps passe plus l'oxygène et d'autres éléments vont corroder les vias, les pistes et surtout les pattes des composants. En essayant de les souder simplement avec de l'étain pur, le contact ne sera pas optimal et la soudure risque ne pas se comporter de la manière attendue sur la durée.

Concernant la présence de plomb dans l'étain, il est important de signaler que les directives RoHS (*Restriction of Hazardous Substances*) imposées en Europe interdisent ou limite l'utilisation de plomb dans une vaste majorité de produits et d'équipements électroniques. Les industriels, depuis 2006, secteur par secteur se voient interdire l'utilisation de cet élément pour des raisons sanitaires. D'un point de vue de l'électronicien amateur et du hobbyiste, et de façon purement pratique, on tâchera d'éviter les étains sans plomb ayant une température de fusion et un comportement différent qui nécessitent une panne compatible dite *Lead Free Solder*. Précisons toutefois que RoHS ne regroupe pas que des directives anti-plomb mais concerne un ensemble d'éléments potentiellement dangereux pour la santé comme le chrome, le cadmium ou le mercure.

4. AUTRES CONSOMMABLES ET ACCESSOIRES

Bien que l'étain de soudure contienne du flux permettant de décaper la surface d'une piste ou d'une patte d'un composant, tantôt, cela n'est pas suffisant. Il est donc possible d'acheter du flux de soudure sous différentes formes. La plus pratique à utiliser est le fluxpen se présentant comme un feutre pour tableau blanc. Ceci permet de tout simplement appliquer du flux sur un élément juste avant d'y placer la panne du fer. Sous la chaleur le flux va s'activer et nettoyer la surface, la préparant idéalement pour l'arriver de l'étain.

Lors de la soudure, il n'est pas rare que la panne du fer se charge d'une masse d'étain importante. Or pour bien souder, l'étain doit être appliqué sur les éléments et non sur la panne du fer. Il est donc important de garder la panne propre. Pour la nettoyer régulièrement, deux solutions : éponge métallique ou l'éponge classique mouillée. La première est une pelotte de fil de métal tressé. On y plonge la panne, l'excédent d'étain se dépose dans l'éponge et la panne ressort propre. L'éponge mouillée est plus économique et permet de frotter la panne pour en retirer l'étain. Certains préciseront cependant qu'une éponge mouillée à température ambiante (~23°C) mise en contacte avec une panne à 350°C ou 400°C provoque un choc thermique qui, à répétition, fera que la panne s'usera plus rapidement. On remarquera toutefois que certains grands constructeurs réputés livrent leur station de soudure avec une éponge et un emplacement à cet effet. On peut donc estimer raisonnablement que le choix est mûrement réfléchi de leur part.

Le petit outillage accompagnant généralement des travaux de soudure consiste au minimum en une pince coupante de bonne qualité. On choisira de préférence une « pince coupante de côté » présentant des tranchants le plus à raz possible du bord de la pince. Ceci permet de couper avec précision les pattes des composants et ce, le plus proche possible du point de soudure. Une astuce intéressante à ce propos est l'utilisation d'un couple-ongle qui, s'il est de bonne qualité offrira une alternative tout à fait valable pour ce qui de travaux. Une pince de bonne qualité vous coûtera tout de même une vingtaine d'euros et, comme il s'agit d'un élément mécanique, elle perdra forcément de son tranchant dans la durée.

Le support pour circuit imprimé est un élément qui dépendra complètement de vos préférences. Certains adorent ce genre d'accessoires permettant de caler un circuit de façon à pouvoir y travailler. D'autres en revanche, comme moi, préfèrent avoir davantage de liberté de mouvement et travaillent donc simplement avec le circuit posé à plat sur le plan de travail.

Une loupe vous sera sans doute nécessaire à un moment ou un autre. Ce type d'accessoire a deux usages : il permet d'inspecter les soudures et l'état des pistes et composants, mais il permet aussi de travailler avec précision. Si vous débuter en électronique, l'inspection de vos soudures sera le seul usage de votre loupe. Plus tard, en vous intéressant à des composants dans un format plus réduit, s'ajoutera alors la nécessité de travailler à la loupe. Pour l'inspection, il n'est pas nécessaire d'utiliser un grossissement très important ou un matériel hors de prix. Un grossissement de x4 ou x8 fera amplement l'affaire. Si vous disposez du budget et comptez travailler sur des circuits fins, la loupe binoculaire sera un outillage de choix. Cependant, une lampe-loupe à 50 euros devrait déjà être suffisante dans la plupart des cas.

Si votre objectif est le dessoudage, la réparation et la récupération de composants, il est recommandé de vous outiller en conséquence. La tresse à dessouder est du fil de cuivre fin tressé qui une fois chauffée sur un point de soudure absorbera l'étain en fusion par capillarité. Choisissez de préférence de la tresse de qualité et de marque, les alternatives bas de gamme composées d'alliages économiques pauvres en cuivre ont une moins bonne conductivité thermique et donnent donc de moins bons résultats à température équivalente.

La pompe à dessouder utilise un autre principe de fonctionnement. Celle-ci se présente comme une seringue munie d'un ressort et d'un déclencheur. On enclenche la pompe en pressant le piston et en compressant le ressort. Après avoir fait fondre l'étain, on place là l'embout de la pompe et en la déclenchant, le ressort tire le piston, aspirant rapidement l'étain. Tout comme avec le support à circuit, ce sont vos préférences qui détermineront l'outil et la technique à utiliser. Certains ne jurent que par la pompe, d'autres que par la tresse et d'autres encore utilisent l'un et l'autre selon le circuit et la soudure à travailler.

Une panne plate Weller au bout d'un fer d'une station Weller. La correspondance de marque station/panne est un des points capitaux pour bien souder/braser.

5. PROTECTION

Les travaux de soudure ne sont pas dangereux en eux-mêmes, pour peu que l'on respecte des règles simples de sécurité. En premier lieu, dégagez votre espace de travail. Une panne à 350°C ne doit entrer en contact ni avec ce qui traîne sur votre plan de travail ni avec vous-même, ce qui est relativement désagréable. Posez votre fer sur son support lorsque vous ne l'utilisez pas et réservez-lui un « espace chaud » bien dégagé tout autour. Vous vous brûlerez, c'est un fait. On se brûle tous au moins une fois pour que la leçon soit apprise. A vous de voir si vous préférez respecter les règles et vous brûler le bout du doigt ou au contraire que le fer brûlant glisse et vous tombe sur les genoux (dans le meilleur des cas).

L'étain de soudure c'est du métal en fusion, du flux, et selon l'état de surface (oxyde, etc) d'autres éléments et composés. Il est possible que ces éléments réagissent et qu'il y ait des projections. Ce n'est pas bien grave, l'épiderme humaine est assez résistante mais ce n'est pas le cas d'autres parties du corps. Je pense naturellement aux yeux. Si vous faites une soudure de précision avec le nez à 5 cm du circuit, la dernière chose que vous souhaitez c'est qu'une projection d'étain ou de flux vous arrive dans l'oeil. Dans la mesure du possible donc, utilisez des lunettes de protection !

Lors d'une soudure, le flux agit avec la température, il en découle des émanations de vapeurs qui ne doivent pas être directement inhalées. Travaillez dans un environnement ventiler ou mieux encore, utilisez un petit ventilateur de bureau pour dissiper les vapeurs. Inutile de produire un courant d'air important, le but est simplement d'avoir un petit flux d'air permettant d'empêcher les vapeurs de stagner sous votre nez.

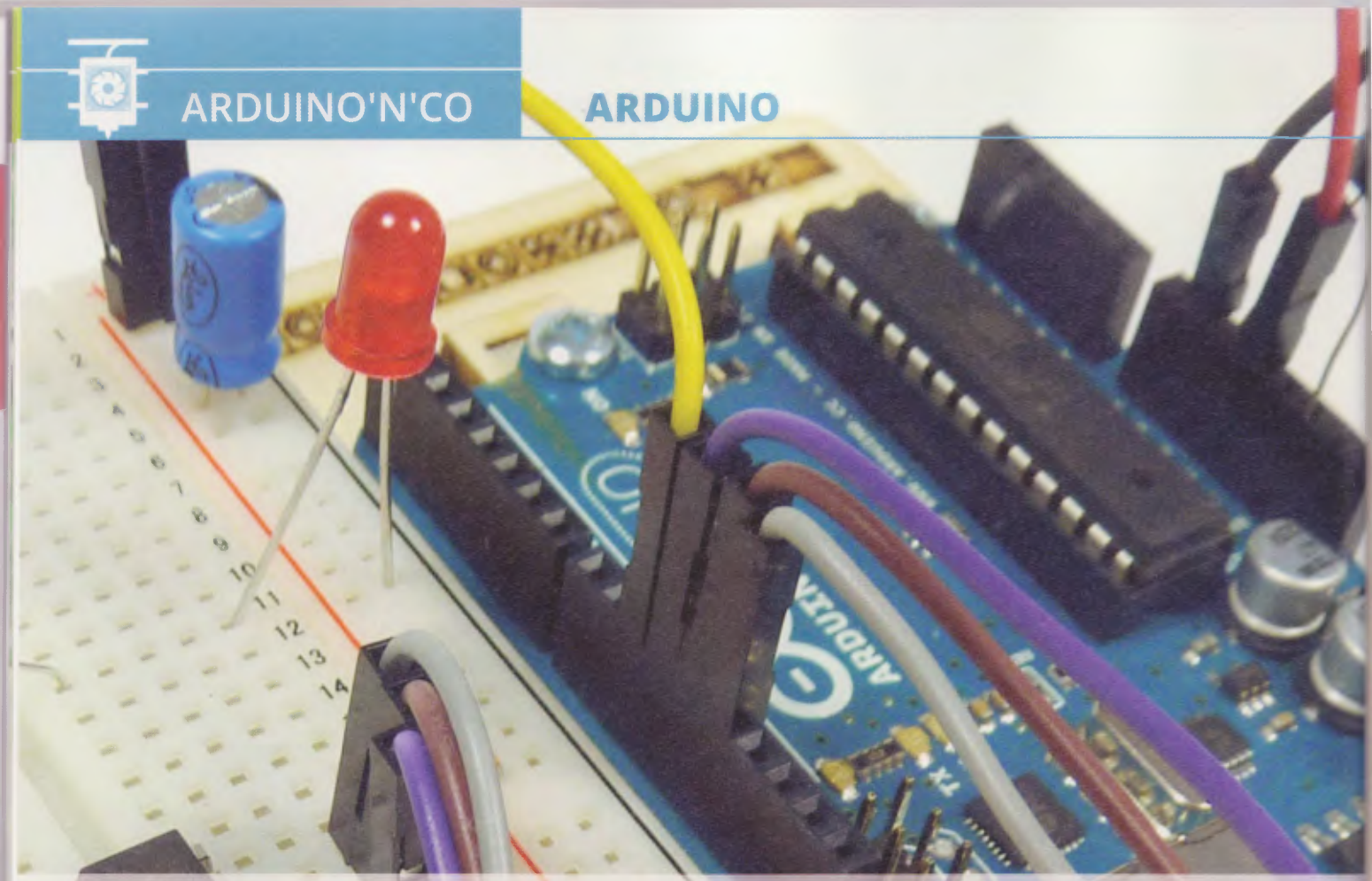
Enfin, n'oubliez pas, le plomb contenu dans l'étain est un matériaux toxique. Le plomb ne peut pas être assimiler par simple contact, il n'y a donc pas de risque à manipuler l'étain de soudure à mains nues. En revanche, il est très toxique par ingestion ce qui signifie, bien entendu, qu'il est très dangereux d'en consommer. Plus sérieusement, deux habitudes sont à prendre impérativement : ne mettez pas le fil d'étain en bouche et lavez-vous les mains. Si la première recommandation est évidente, on oubli souvent la seconde or, après vos travaux de soudure il est probable qu'à un moment ou un autre vous mettrez les doigts à votre bouche ou encore que vous risquez de transférer des résidus sur votre quatre-heure, votre (e)cigarette, votre tasse à café, etc. Lavez-vous les mains !

CONCLUSION

Nous avons exposé ici, dans les grandes lignes ce qu'il faut savoir pour s'équiper correctement pour bien souder/dessouder. Comptez un budget de 150 euros et dès lors 30% du travail sera fait par votre station, 30% par une panne propre et de bonne qualité et 30% par de l'étain de soudure digne de ce nom. Il ne vous reste que 10% du boulot, qui consiste simplement à ne pas oublier la règle de base : on chauffe et on pose l'étain sur les éléments à souder et non sur la panne.

Terminons avec un conseil qui s'avérera commun à presque tous les outils de l'électronicien : dans la mesure du possible, prévoyez votre équipement en double. Dessouder des composants montés en surface est d'une facilité déconcertante avec deux stations de soudage et ce n'est la qu'un des avantages. Il en va de même pour les multimètres (mesurer tension et courant en même temps) ou encore pour les oscilloscopes (un vieux modèle analogique + un oscilloscope moderne digital).

L'électronique comme beaucoup d'autres domaines et activités coûte de l'argent et n'est un véritable plaisir qu'avec un équipement qui vous permet de vous concentrer sur ce que vous faites. Si vous le pouvez, équipez-vous correctement et dans tous les cas, évitez les clones, les contrefaçons et les pseudo-équipement très bas de gammes ou sans marque. Dans ce cas, votre argent ne sera pas investis pour rien et l'électronique vous apportera tout le plaisir que vous en attendez. DB



UN ARDUINO DE LA TAILLE D'UNE CACAHUÈTE !

La grande majorité des cartes Arduino repose sur un microcontrôleur Atmel AVR et, dans le cas de l'Arduino Uno, un ATmega328p. Mais ce composant n'est de loin pas le seul de sa gamme (megaAVR) et encore moins le seul microcontrôleur 8 bits d'Atmel. Ainsi, ce qu'il est possible de faire avec l'Arduino est, dans une certaine mesure, adaptable à un microcontrôleur plus modeste, comme le petit ATtiny85. Chose que nous allons découvrir à présent.

Le microcontrôleur AVR de l'Arduino Uno n'est pas le seul composant actif de la carte. Elle est également équipée d'un ATmega16U2 permettant l'interface avec l'USB et donc la connexion au PC (sur d'autres Arduino c'est une puce FTDI qui accompli cette tâche). D'autres composants sont également ajoutés pour réguler le courant alimentant l'ensemble, ainsi que quelques condensateurs et résistances. Sur le côté de la carte vous avez peut-être remarqué le connecteur marqué ICSP pour *In Circuit Serial Programming* (programmation série in situ).

Celui-ci sert à connecter un programmeur comme l'Atmel AVRISP mkII, l'Atmel AVR Dragon ou n'importe quel autre programmeur d'AVRISP ou compatible. En effet, il faut savoir qu'un microcontrôleur comme un AVR est un composant disposant d'une interface de programmation mais qui nécessite un équipement qui n'est pas intégré à la carte. Pour palier à cela, Arduino choisi d'utiliser un morceau de programme qui peut démarrer avant n'importe quel croquis pour prendre la programmation en charge, un bootloader. Ceci évite d'avoir recourt à un matériel supplémentaire ou à augmenter le coût de fabrication de la carte en ajoutant d'autres composants. Cependant, si l'on souhaite utiliser l'intégralité de la mémoire du microcontrôleur, on se passera de bootloader et les Arduino disposent pour cela du connecteur adapté. Ceci permet également, en cas de problème avec le microcontrôleur, de le remplacer par un composant neuf et d'y installer le bootloader Arduino.

Si vous achetez un Atmel AVR Atmega328p chez un fournisseur ou détaillant de matériel électronique, celui-ci vous arrivera vierge et donc sans bootloader. D'autres AVR ne disposent même pas de la fonctionnalité permettant d'installer un tel programme. Dans ce genre de situation, le seul moyen de programmer le microcontrôleur est d'utiliser l'interface ICSP.

Précisons que, bien avant l'arrivé de l'Arduino, une communauté importante de développeurs pour Atmel AVR existait déjà. En effet, à l'aide d'un éditeur de code, d'un compilateur adapté et d'un programmeur, il est possible depuis longtemps de s'initier à la programmation de ces fantastiques composants. Arduino a rendu les choses encore plus accessibles en fournissant un environnement ultra-simple et en minimisant les besoins en termes d'achat de matériel.

1. UN ARDUINO SANS ARDUINO ?

Il est donc parfaitement possible de programmer le microcontrôleur d'un Arduino en utilisant un tel programmeur et le connecteur ICSP. Mais il est également possible de programmer, par exemple un Arduino Uno, de manière à ce qu'il se transforme en programmeur ISP (l'un des protocoles utilisés par Atmel pour l'ICSP). Pourquoi ? Simplement pour pouvoir programmer un autre microcontrôleur placé sur une *breadboard* et ce, depuis l'environnement Arduino avec toutes les facilités de développement qu'il propose. Oui, il est possible d'utiliser des croquis avec un nombre impressionnant de microcontrôleurs AVR et ce sans le reste de la carte Arduino.

Il y a plusieurs motivations justifiant cela :

- Le prix : une carte Arduino coûte environs 24 euros. Un microcontrôleur ATmega328p vous coûtera seulement un peu plus de 3 euros. Il vous faudra certes une *breadboard*, quelques composants (condensateurs, quartz, etc), une source d'alimentation 5 volts et un module USB/série pour obtenir l'équivalent d'un Arduino, mais ce n'est pas une absolue nécessité. Un AVR seul sur une *breadboard* fonctionne parfaitement si l'on n' a pas besoin de la puissance/précision (quartz) d'un Arduino et du moniteur série.
- La taille : regardez le microcontrôleur d'un Arduino Uno, il occupe en lui-même très peu de place. Et cela n'est qu'un des formats existants (PDIP-28), il en existe de bien moins volumineux, comme le SOIC, qui reste utilisable par des électroniciens amateurs qui souhaitent se lancer dans la réalisation de circuits imprimés (le format ou *package* VQFN est un peu plus problématique). En utilisant un petit AVR et avec de la patience il est ainsi possible de réaliser des montages aussi minuscules que discrets.
- L'expérience et la diversité : sortir des sentiers battus est l'un des meilleurs moyens d'apprendre de nouvelles choses. Utiliser l'environnement Arduino avec autre chose qu'une carte Arduino est un bon début. L'autre voie consistant à ne pas utiliser l'environnement dédié et les facilités qui vont avec. De plus, les microcontrôleurs utilisés par les Arduino officiels sont peu diversifiés :

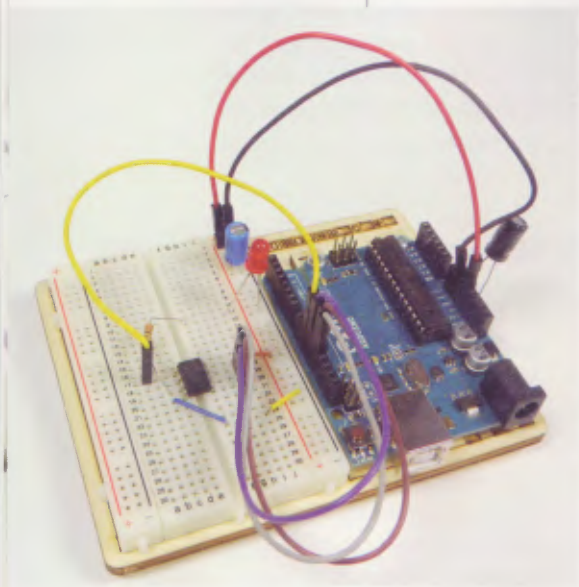
ATmega168, ATmega328P, ATmega1280, ATmega2560, ATmega32U4 et Atmel SAM3X8E.
Ce dernier est un particulier puisqu'il ne s'agit pas de la gamme AVR 8 bits mais d'un SAM3 32 bits à coeur ARM Cortex-M3. En utilisant votre Arduino comme programmeur ISP, vous vous ouvrez donc à une masse importante d'autres microcontrôleurs.

Précisons tout de même que les explications qui vont suivre peuvent en partie s'appliquer à tous les AVR 8 bits (ce qui exclus les XMEGA et les UC3 utilisant une interface PDI, TPI ou JTAG et non l'ISP). Deux choses sont à prendre en compte :

- Le microcontrôleur doit pouvoir être programmer avec le protocole ISP,
- et si l'on souhaite utiliser l'environnement Arduino pour créer des croquis pour ce microcontrôleur, il faut que celui-ci dispose des ressources et fonctionnalités nécessaires et en particulier d'une quantité de flash et de mémoire (SRAM) suffisante pour accueillir vos programmes.

Enfin et il est très utile de le préciser, un microcontrôleur seul sur une *breadboard* n'est PAS un Arduino. Même en créant un circuit et en ajoutant tout ce qu'il faut pour être identique aux cartes originelles, cela ne sera dans aucun cas une carte Arduino. Il n'y a que les Arduino qui sont des Arduino. C'est une question de marque et de droit d'utilisation d'un nom commercial. On a tendance à l'oublier mais en utilisant un Arduino, l'activité à laquelle vous vous livrez est de la programmation sur microcontrôleur et c'est un domaine qui s'étend bien au delà du simple Arduino.

Une carte Arduino peut être facilement transformée en programmeur ISP pour téléverser un croquis dans un microcontrôleur AVR comme ceux qui équipent les Arduino, mais aussi et surtout des modèles plus petits, faciles à intégrer dans des montages « maison », comme ceux de la famille ATtiny.



2. TRANSFORMER L'ARDUINO EN PROGRAMMEUR

Cette étape est simplissime puisque qu'il suffit de téléverser un croquis dans l'Arduino Uno. Mieux encore, il ne vous sera pas nécessaire de le récupérer sur le net puisque celui-ci est livré parmi les exemples de croquis. Passez donc simplement par Fichier -> Exemples -> ArduinoISP pour ouvrir le croquis de Randall Bohn. Remarquez que parmi les premières lignes du code, la connexion est clairement décrite dans les commentaires.

Un connecteur ISP utilise six connecteurs :

- VTG : c'est la ligne d'alimentation fournie par la cible programmée, ceci signifie que la cible doit être alimentée,
- GND : la masse,
- /RES : la ligne de *reset* pour provoquer le redémarrage de la cible,
- SCK : la ligne d'horloge pour cadencer l'envoi et la réception des données,
- MISO (*master in - slave out*) : la ligne où circulent les données de la cible (*slave*) vers le programmeur (*master*),
- MOSI (*master out - slave in*) : la ligne de données du programmeur (master) à destination de la cible (*slave*).

Dans le croquis nous voyons que :

- RST = broche 10,
- MOSI = broche 11,
- MISO = broche 12,
- SCL = broche 13.

Notez que ceci concerne les Arduino non « mega » et donc notre Arduino Uno. Les Arduino Mega et Arduino Mega 2560 utilisent d'autres broches (respectivement 53, 51, 20 et 52).

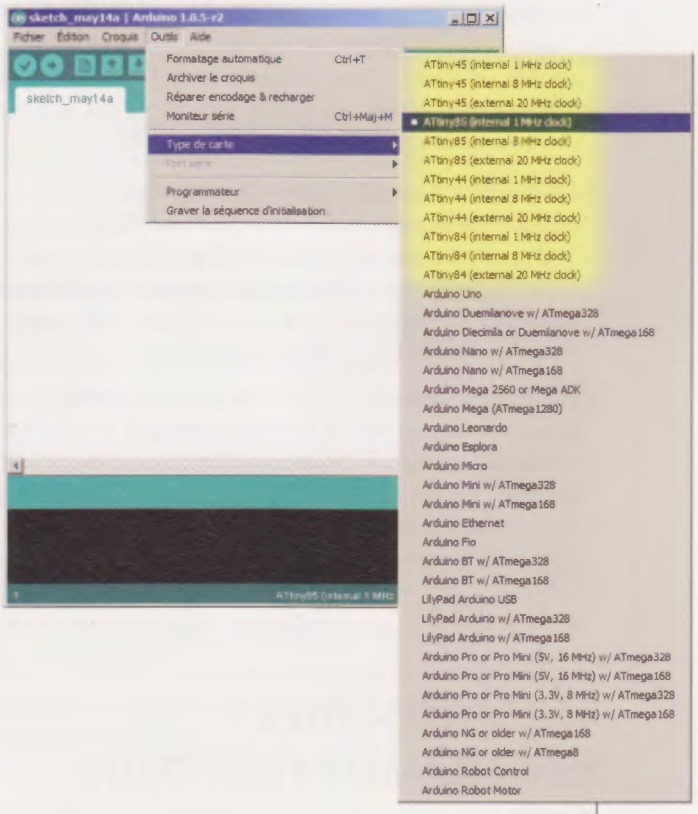
Une fois votre croquis compilé et téléversé dans votre Arduino, vous êtes en possession d'un programmeur ISP utilisable.

3. MODIFICATION DE L'ENVIRONNEMENT ARDUINO

Si vous regardez dans Outils -> Type de carte, vous ne trouverez aucune trace de petits microcontrôleurs comme l'ATtiny85. En revanche, l'entrée Outils -> Programmateur propose bien « Arduino as ISP » qui correspond, comme son nom l'indique, à un Arduino utilisé comme programmeur ISP. Précisons, si c'est nécessaire, que toutes ces manipulations ne présentent aucun risque particulier. Si vous souhaitez, par la suite, reprogrammer votre Arduino avec un autre croquis exemple ou l'une de vos réalisations, il vous suffira de choisir le bon modèle de carte (ici Uno) et le programmeur « AVRISP mkII ». Notez au passage l'utilisation absolument horrible du terme « programmateur » dans les menus de l'environnement Arduino. Comme aiment à le préciser les développeurs : « un programmeur c'est pour le code, un programmateur c'est pour le lave-linge ! ».

Pour pouvoir programmer un microcontrôleur différent de ceux proposés par l'environnement, il va être nécessaire de télécharger et d'installer un élément externe. Rendez-vous sur **https://github.com/damellis/attiny/** et choisissez « Download ZIP » pour récupérer ce qui nous intéresse.

Enregistrer le fichier `attiny-master.zip` obtenu dans un emplacement temporaire quelconque. Il vous faudra ensuite connaître l'emplacement de votre carnet de croquis. Rendez-vous dans les préférences de l'éditeur Arduino pour connaître le chemin exacte. Généralement, il s'agit du répertoire `Arduino` dans le répertoire `Documents` de l'utilisateur courant.

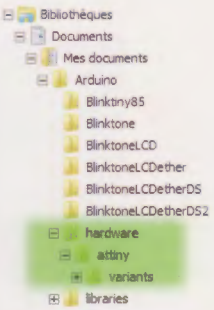


Dès redémarrage de l'environnement Arduino, de nouvelles « cartes » sont à votre disposition.

Quittez l'environnement et double-cliquez sur `attiny-master.zip`. Une fenêtre s'ouvre vous présentant son contenu, un répertoire `attiny-master`. Ce n'est *pas* le répertoire que vous devez copier ! Entrez dans ce dossier pour trouver `attiny`. Là, sélectionner ce dossier, copier-le puis placez vous dans le carnet de croquis.

Si vous avez déjà utilisez Arduino pour créer vos propres projets, il est probable que se trouve dans cet emplacement un certain nombre de dossiers. Créez alors un dossier `hardware` et double-cliquez pour l'ouvrir. Enfin, collez `attiny` à cet endroit. Vous devez donc avoir `hardware` dans votre carnet de croquis et ce dossier doit contenir `attiny` qui, lui-même comprend un fichier `boards.txt` et un dossier `variant`.

Lancez maintenant votre environnement Arduino. Dans Outils -> Type de carte vous devez à présent trouver : ATtiny45, ATtiny85, ATtiny44 et ATtiny84, décliné dans différentes versions faisant référence à la vitesse de l'horloge qui cadence le composant :

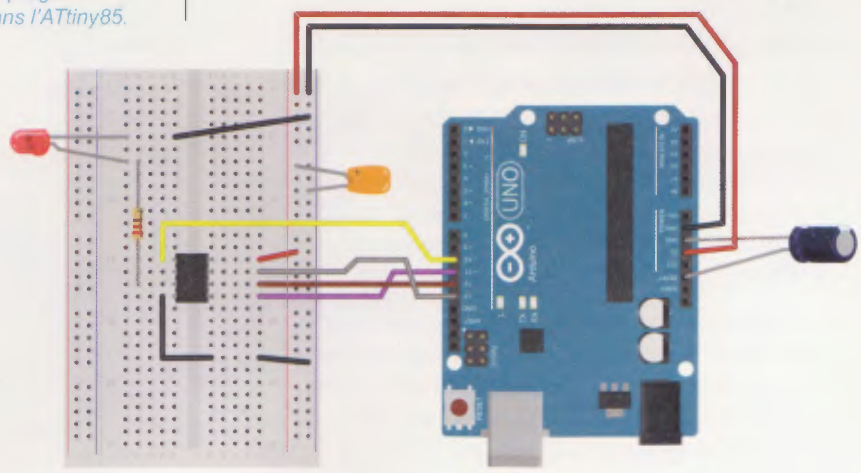


L'arborescence de votre carnet de croquis une fois les nouveaux fichiers installés.

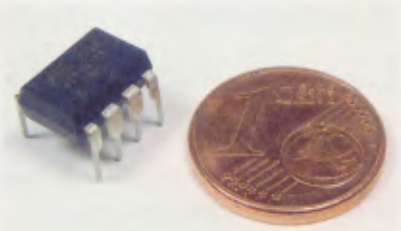
- *internal 1 Mhz clock* : c'est la valeur par défaut d'un ATtiny. Celui-ci n'a pas besoin de source d'horloge externe pour fonctionner, ce qui permet d'utiliser l'ensemble des broches disponibles (8 broches en tout, moins une masse, un *reset* et une pour l'alimentation, reste 5)
- *internal 8 Mhz clock* : 1 Mhz c'est un peu faible, en particulier pour certaines fonctionnalités (bibliothèque SoftwareSerial par exemple) et il est possible de reconfigurer le microcontrôleur pour utiliser une fréquence de 8 Mhz, toujours en interne et donc sans composants supplémentaires (cf plus loin dans l'article).
- *external 20 Mhz clock* : l'horloge interne d'un ATtiny est précise mais pas autant qu'un oscillateur à quartz. Il est possible de pousser la vitesse du microcontrôleur à 20 Mhz en utilisant un tel composant (plus deux condensateurs céramiques à la valeur adaptée, soit 12 à 22 pF). Cependant, vous perdrez alors l'usage de deux autres broches du microcontrôleur (en restera plus que 3). Là encore il sera nécessaire de reconfigurer le composant.

4. CONNEXION ET PROGRAMMATION D'UN ATTINY85

Il est maintenant temps de connecter l'Arduino à un microcontrôleur AVR ATtiny85 afin de pouvoir le programmer. L'Arduino fournira l'alimentation à la *breadboard* et donc à l'ATtiny (broche 4 à la masse et 8 au +5 volts).



Connexion d'un Arduino Uno programmé avec ArduinoISP et prêt à téléverser un programme dans l'ATtiny85.



L'ATtiny85, 8 Ko de Flash, 512 octets de SRAM, un ADC, un USI, des sorties PWM. Tout cela dans un tout petit petit boitier PDIP-8.

Dans le montage illustré ici, nous avons, bien entendu, la connexion des quatre liaisons ISP de base avec la correspondance suivante :

- Broche 1 de l'ATtiny (/RST) sur 10,
- broche 5 (MOSI) sur 11,
- broche 6 (MISO) sur 12,
- et broche 7 (SCK) sur 13.

C'est exactement ce qui se trouve en commentaire dans le croquis ArduinoISP. Nous avons également ajouté une résistance de 220 Ohms et une led connectés à la broche 3 de l'ATtiny afin de pouvoir rapidement procéder à un premier test. Le composant qui se trouve à droite de la *breadboard* est un condensateur (tantale-goutte) de filtrage de 10µF (une valeur supérieure fera également l'affaire). Ceci est optionnel et permet simplement de s'assurer que le courant fourni au microcontrôleur est « propre ». Enfin, point très important, sur l'Arduino lui-même, notez le positionnement, là aussi, d'un condensateur (10µF ou plus) entre la masse et la broche reset (avec le + sur le reset).

Ce condensateur est très important car il va empêcher le redémarrage de l'Arduino au

moment du téléversement du croquis. En effet, l'environnement Arduino, à ce moment, redémarre la carte pour arriver sur le bootloader qui, discutant avec le PC via USB va recevoir le croquis compilé et l'écrire dans la mémoire du microcontrôleur de l'Arduino. Or, nous ne voulons pas programmer la carte Arduino mais l'ATtiny, il ne faut donc pas que l'Arduino redémarre mais que le croquis ArduinoISP dialogue avec le PC pour programmer l'ATtiny.

Un microcontrôleur AVR comme celui équipant l'Ardiuno Uno redémarre si on met la broche RESET à la masse. Ceci signifie donc qu'en fonctionnement « normal » nous avons +5 volts sur cette broche. Le condensateur qu'on place entre ici et la masse va donc se charger et lorsque l'environnement Arduino va tenter de faire un reset, c'est le condensateur qui fournira temporairement +5 volts sur la broche, la carte ne redémarrera donc pas. Techniquement, vous pouvez également placer une résistance de 10 Kohm, par exemple, entre le reset et +5 volts, si vous n'avez pas de condensateur sous la main. C'est ce qu'on appel une résistance de rappel au +5 volts.

5. TÉLÉVERSEMENT DANS L'ATTINY

Pour programmer l'ATtiny85, vous utiliserez l'environnement Arduino. Ouvrez l'exemple Blink et commencez par modifier le croquis en changeant le numéro de la broche utilisée pour la led. Initialement 13, celle-ci sera changée en 4 car, bien entendu, l'ATtiny ne dispose pas de broche 13. Voici la correspondance et les caractéristiques des broches :

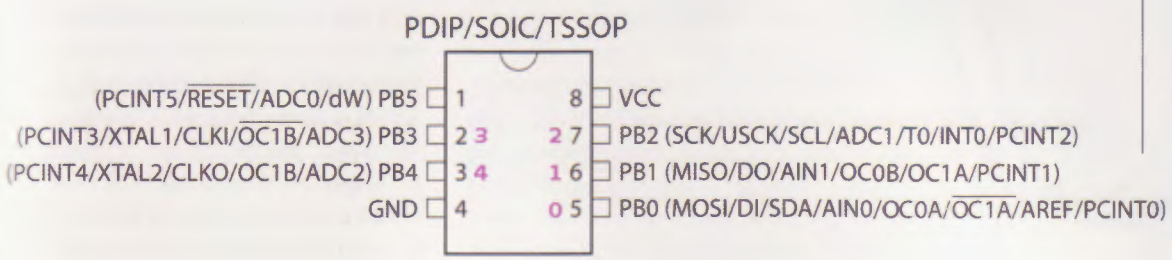
- 5 : broche Arduino 0, sortie analogique/digital (PWM) et référence pour les entrées analogiques,
- 6 : broche Arduino 1, sortie analogique/digital (PWM),
- 7 : broche Arduino 2, Entrée analogique 1,
- 2 : broche Arduino 3, Entrée analogique 3,
- 1 : Broche Arduino 4, Entrée analogique 2.

Assurez-vous ensuite de choisir le bon « type de carte », ATtiny85 (internal 1 Mhz), le programmeur « Arduino as ISP » et enfin le port série adéquate (le même que celui pour l'Arduino lui-même). Il ne vous reste plus qu'à téléverser le croquis et vous devez alors voir clignoter la led en guise de récompense.

6. MODIFIER LA VITESSE DE L'ATTINY

Ceci est généralement invisible pour les utilisateurs d'Arduino, mais il faut savoir que les microcontrôleurs Atmel AVR sont configurables. Ils possèdent à cet effet un ensemble de *fuses* qui permet de définir différent paramètres :

- Activation/désactivation de l'interface ISP,
- Activation/désactivation du reset,



Description des broches d'un ATtiny85 avec, en magenta, les numéros des ports Arduino correspondants (source documentation/ datasheet Atmel).

- Effacement ou non de l'EEPROM en même temps que la flash,
- Détection de défaillance d'alimentation,
- Activation/désactivation du *watchdog*,
- et surtout, choix de la source d'horloge.

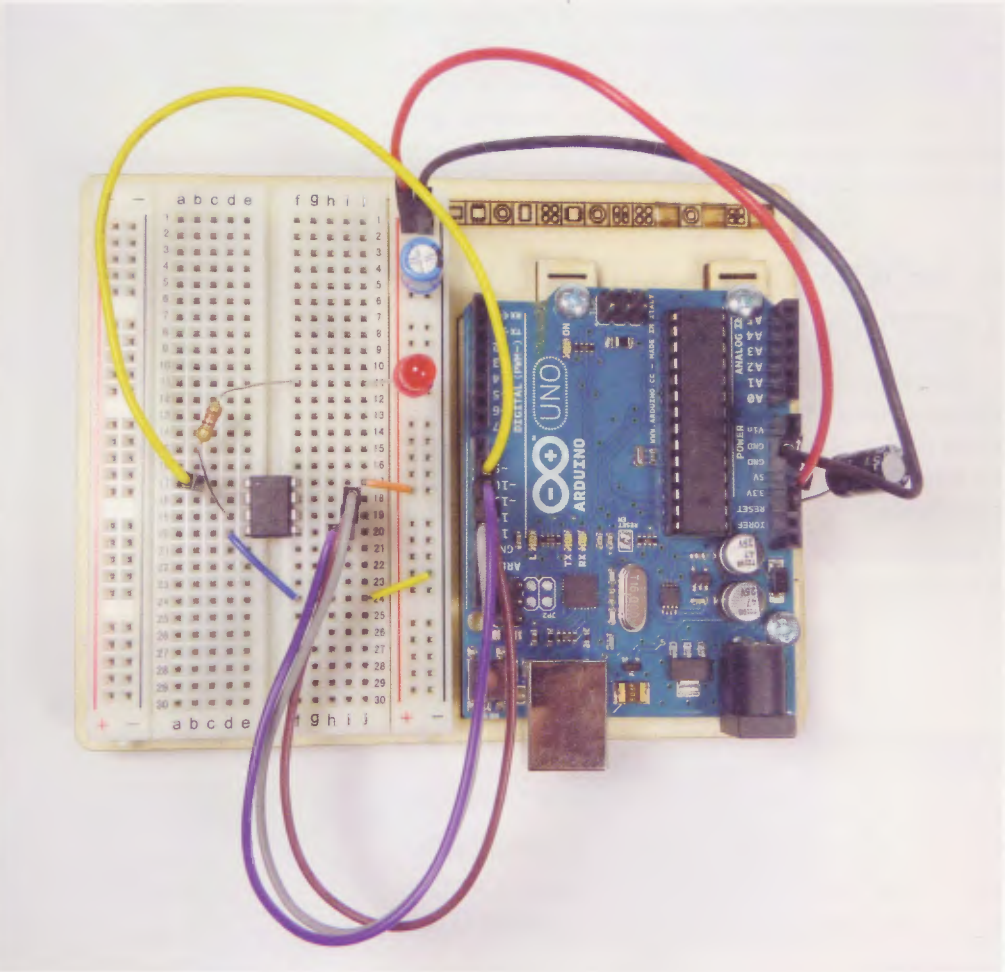
C'est ce dernier point qui nous intéresse mais, ne vous inquiétez pas, il ne vous sera pas nécessaire d'apprendre à manipuler ces *fuses* car les personnes ayant ajouté le support ATtiny ont pensé à tout. Comme expliqué précédemment, dans votre nouvelle liste de cartes Arduino accessible par le menu Outils se trouvent trois ATtiny85 : 1 Mhz, 8 Mhz et 20 Mhz. Pour changer la configuration de votre microcontrôleur, il faut procéder comme suit :

- Choisir le type de « carte » qui vous intéresse,
- utiliser le menu Outils -> Graver la séquence d'initialisation,
- et téléverser à nouveau votre croquis dans l'ATtiny.

Plusieurs choses importantes sont à savoir. La première est que, contrairement à ce que laisse entendre la désignation du menu, ceci ne va pas graver de séquence d'initialisation (le bootloader Arduino) mais simplement changer la configuration des *fuses* de votre ATtiny. L'ATtiny85 peut supporter un bootloader mais dans le contexte actuel cela ne présente pas vraiment d'intérêt.

La configuration de la source d'horloge et donc de la vitesse n'est à faire qu'une seule fois par microcontrôleur. La configuration perdure au fil des redémarrages et même des téléversements de croquis. Il est cependant recommander de repérer d'une manière ou d'une autre quel composant utilise telle ou telle fréquence ou, du moins, de les conserver séparément pour ne pas vous mélanger les pinceaux.

1 et 8 Mhz correspondent à des fréquences que l'ATtiny génère de façon interne, sans composants externes supplémentaires. Vous pouvez à loisir passer de l'une à l'autre. En revanche, le passage à 20 Mhz nécessite un oscillateur à quartz et deux condensateurs céramiques.



L'assemblage final de l'Arduino Uno et de la breadboard où est placé l'ATtiny85 à programmer.

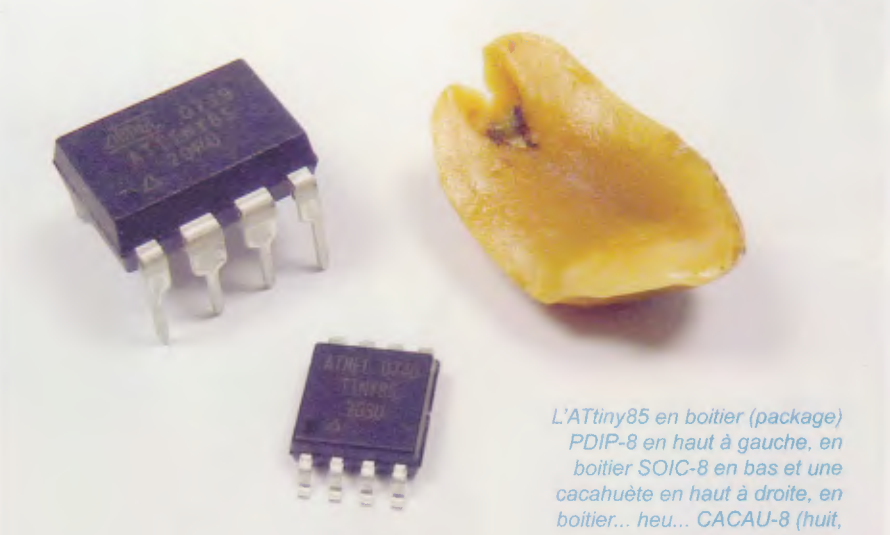
Or si ces éléments ne sont pas présent une fois le microcontrôleur configuré, il ne fonctionnera pas (pas de source d'horloge). En d'autres termes, faites très attention à ce que vous faites car si vous configurez une source d'horloge à quartz alors que vous n'en disposez pas, il vous faudra un vrai programmeur ISP pour reconfigurer l'ATtiny.

Enfin, une fois votre ATTiny configuré à 8 Mhz par exemple, il est important de ne pas oublier de choisir la bonne version dans l'environnement Arduino avant de compiler et téléverser vos croquis. La vitesse de fonctionnement du microcontrôleur impacte de calcul des délais. Si votre croquis s'attend à un fonctionnement à 8 Mhz alors que l'ATtiny est configuré à 1 Mhz, un délai d'une seconde deviendra un délai de 8 secondes. Encore une fois, soyez ordonné, rangez vos composants correctement ou marquez-les (une petite touche de Typex sur le dessous fera parfaitement l'affaire).

7. LE MOT DE LA FIN : À PROPOS DES LIMITES

Vous vous en doutez, il ne vous sera pas possible de faire autant de choses avec un ATtiny85 qu'avec l'ATmega328p d'un Arduino Uno. Ainsi, les fonctions utilisables se limiteront à :

- `pinMode()`
- `digitalWrite()`
- `digitalRead()`
- `analogRead()`
- `analogWrite()`
- `shiftOut()`



L'ATtiny85 en boitier (package) PDIP-8 en haut à gauche, en boitier SOIC-8 en bas et une cacahuète en haut à droite, en boitier... heu... CACAU-8 (huit, eight, huète... caca-U-eight... désolé).

- `pulseIn()`
- `millis()`
- `micros()`
- `delay()`
- `delayMicroseconds()`
- la bibliothèque `SoftwareSerial`

Ceci vous ouvre cependant un certain nombre de perspectives, en particulier en raison de la petite taille de l'ATtiny et de son faible coût. A titre d'exemple, l'ATtiny85-20PU chez Mouser est vendu 1,30 euros TTC pièce.

Pour exploiter davantage les ressources de ce petit microcontrôleur, il sera ensuite nécessaire de vous détacher de l'environnement Arduino et de la notion de croquis pour programmer « sans filet » avec la documentation Atmel sous le coude (234 pages de pure technique en anglais) mais c'est une toute autre histoire.

Ce que nous venons de faire avec un ATiny85 est également possible avec bien d'autres AVR. Rien ne vous empêche d'acheter un lot d'ATmega328p et de les utiliser de la sorte. L'ATtiny13 peut également être utilisé ainsi que le sympathique ATtiny2343 (ou 2313). Vous n'avez que l'embarra du choix. Enfin, et cela méritera sans doute un article dans un prochain numéro, la bibliothèque TinyWire permet à un ATtiny85 d'utiliser son interface i2c/TWI et donc de s'interfacer avec un certain nombre de modules comme des horloges (DS1307). En vérité, c'est tout un nouveau terrain de jeu qui s'offre à vous, avec des heures de plaisir et de manque de sommeil à la *clé*... **DB**

L'ARDUINO STARTER KIT : UN PACK POUR DÉBUTER RAPIDEMENT

La solution la plus aisée pour faire ses premiers pas avec l'Arduino est d'acquérir un kit d'initiation et, pourquoi pas, le « Arduino Starter Kit » officiel. Dans la boîte : un Arduino Uno, une collection de composants électroniques, quelques accessoires, ainsi qu'un guide explicatif.

1. C'EST NOËL !

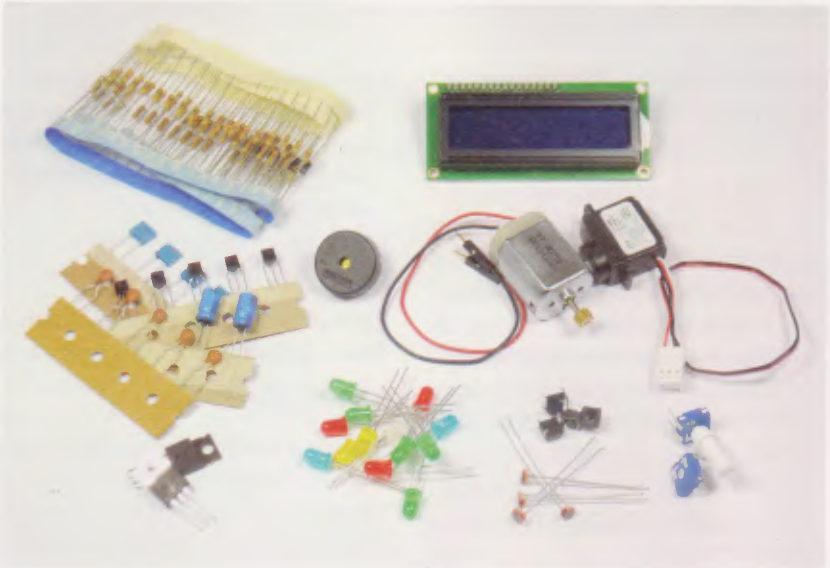
Ce kit coûte environ 90€ et est celui officiellement recommandé pour débuter (pas par nous, par le projet Arduino). Avec la popularité croissante d'Arduino, des kits de découverte ont fait leur apparition de toutes parts. L'idée est de réunir en un seul produit un lot de composants et d'accessoires permettant de simplement s'initier à l'électronique sans avoir à tout acheter séparément chez un détaillant. Notez tout de même qu'un tel kit à une durée de vie limitée, car vous en viendrez forcément à compléter votre « garde manger » de composants par une succession de commandes en tout genre (et ce sera Noël à chaque fois). Le caractère officiel de ce kit est l'une des raisons qui nous ont poussé à en faire la présentation.

Le premier contact est plutôt positif, car le packaging est particulièrement attrayant et soigné. Chaque chose est convenablement rangée dans sa boîte. On prend plaisir à déballer chacun de ces petits paquets, comme au matin de Noël. Aucune déception non plus au niveau du contenu : il y a absolument tout ce qu'il faut pour commencer à s'amuser. En bonus, un petit support en bois pour fixer correctement votre Arduino et travailler dans de bonnes conditions (selon vos préférences).

Vous y trouverez ainsi :

- un ou une Arduino Uno (non qu'il y ait des versions mâle et femelle, c'est simplement que certains préfèrent « un » et d'autres « une ». Mais en réalité c'est comme « le » ou « la » Nutella, le problème n'est pas sensé se poser puisque c'est un nom propre, c'est donc « Arduino » et « Nutella » tout court, en principe),
- une *breadboard* (ou « platine à essai »), autrement dit, un support sur lequel vous pouvez enficher vos composants électroniques pour réaliser un circuit sans soudure),
- un support en bois prédécoupé, à monter, pour fixer correctement votre « station » de travail,
- un écran LCD alpha-numérique standard,
- un micro-servomoteur ainsi qu'un moteur DC,
- un sachet de divers composants électroniques (condensateurs, leds de plusieurs couleurs, câbles et cavaliers de tailles diverses, diodes, photorésistances, potentiomètres, boutons-poussoirs, résistances, capteur de température...)
- un câble USB.

Le manuel (en anglais) qui accompagne le kit est très bien fait, très didactique. Il commence par rappeler quelques bases d'électronique en détaillant le rôle de chacun des composants. Il comporte également une section expliquant comment installer, puis utiliser l'IDE



Une partie des composants livrés avec le kit de démarrage : écran LCD, résistances, condensateurs, transistors, leds, boutons poussoir, potentiomètres rotatifs, régulateurs de tension, LDR, buzzer, servomoteur miniature, moteur... Autant d'éléments qui permettent de réaliser des dizaines de montages pour apprendre et se familiariser avec l'électronique et le monde de l'Arduino.

(l'environnement de développement) sous Windows et Mac OS X (GNU/Linux, comme souvent, est oublié).

Vous trouverez dans ce manuel 15 projets à réaliser avec le contenu du kit, de difficulté croissante ; les montages électroniques et le code des programmes (les croquis ou *sketchs*) sont très bien commentés. De petits pictogrammes viennent souligner les points importants des projets, donner des explications supplémentaires ou vous invitent à aller plus loin en suggérant quelques pistes à explorer.

Un glossaire en fin d'ouvrage est là pour combler d'éventuelles lacunes. Enfin, plusieurs pages libres permettent de griffonner quelques notes si besoin.

Bref, c'est très intéressant... si on est anglophone. Pour les francophones, pas de panique, on est là et la communauté Arduino aussi :)

2. PRÉPARATION

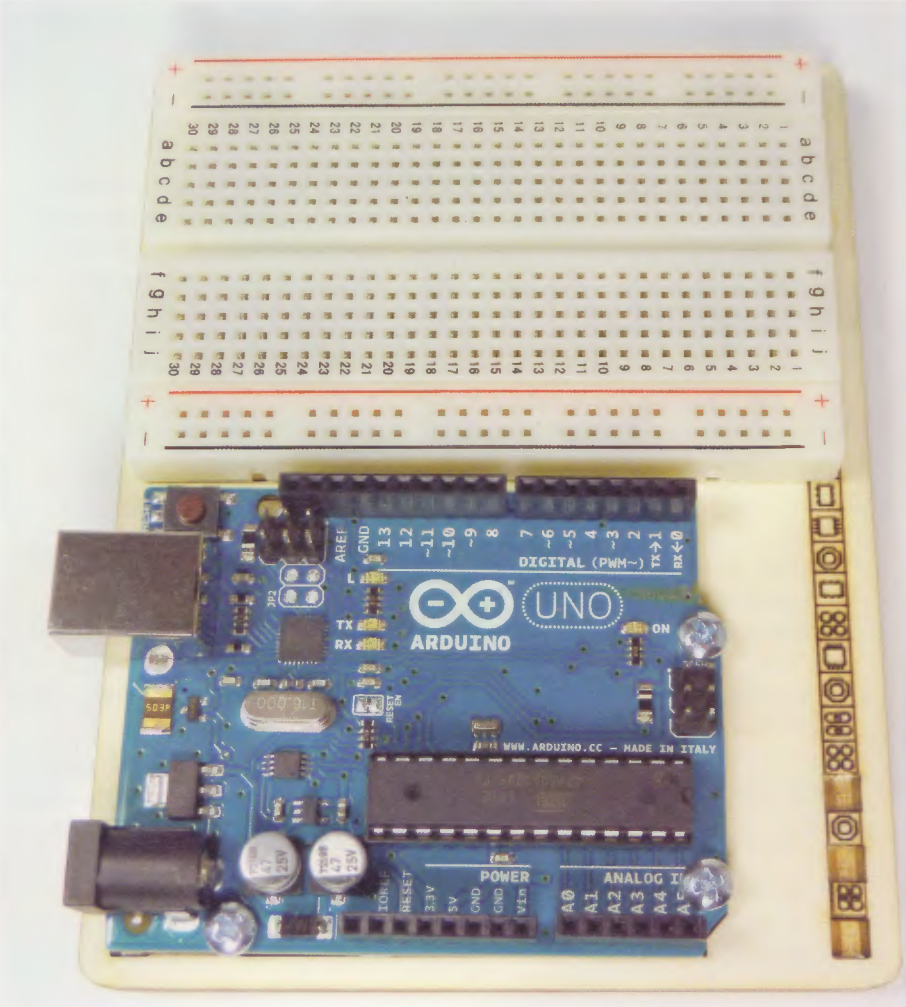
Comme indiqué dans le manuel, on commence par monter l'Arduino sur le support en bois fourni dans le kit.

- On détache tous les morceaux prédécoupés - en tâchant de n'en perdre aucun, puisqu'à priori ils seront utiles pour certains des projets.
- On utilise les 4 pièces en forme de « E » majuscule comme pieds pour le support. Attention : il s'agit d'un détail mais le schéma qui illustre l'insertion des 4 pieds du support est faux ; il faut les insérer dans l'autre sens (par en-dessous) si l'on tient le support en bois comme indiqué sur l'image.
- On fixe la carte sur le support à l'aide des trois petits boulons et écrous fournis (ils se trouvent dans le sachet de composants électroniques).
- On détache la pellicule de protection qui se trouve sous la *breadboard*, puis on la colle juste à côté de l'Arduino.

Et on obtient finalement :

3. LA BREADBOARD

Egalement appelée *protoboard* ou platine à essais, la *breadboard* est le support sur lequel vous allez créer vos premiers circuits. Le nom *breadboard*, littéralement « planche à pain », provient historiquement d'une des premières solutions permettant de simplement créer un assemblage de composants.



Ceci fait, vous êtes prêt à réaliser vos premiers montages. Mais avant cela, un petit retour sur la *breadboard*...

A cette époque reculée, on avait trouvé une astuce consistant à garnir une planche à pain en bois de clous pour y fixer les composants.

Cette technique était également utilisée en ingénierie pour préparer des maquettes des travaux et assemblages mécaniques.

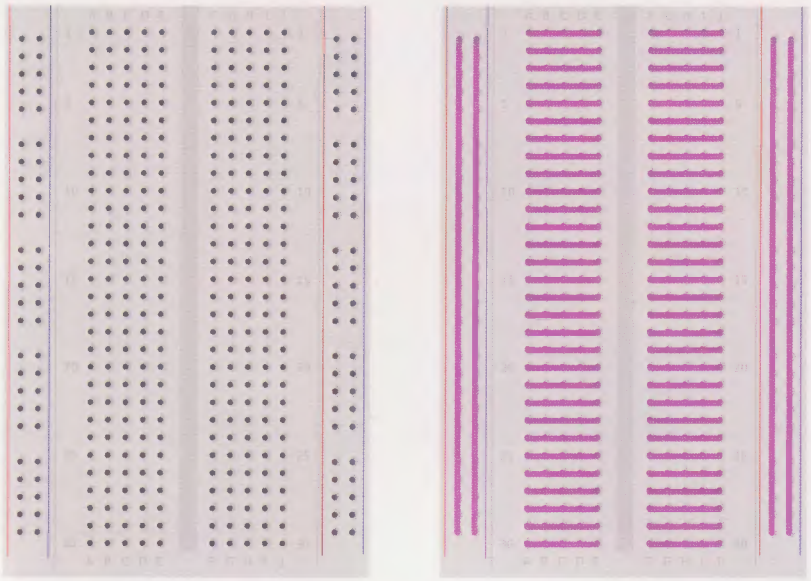
Depuis, la *breadboad* a bien évoluée et est devenue le support de prédilection pour créer des circuits sans soudure. Elle se présente comme une grille de connecteurs permettant d'enficher des composants. Il est possible de placer des composants partout mais il faut savoir que certaines lignes et colonnes de la grille sont interconnectés.

Avec une *breadboard* de 400 points standard (*half+*) nous avons deux séries de 5 colonnes libellées de « a » à « e » et de « f » à « j », ainsi que 30 lignes qui connectent les colonnes entre elles. Ce type de *breadboard* dispose également de deux « rails » d'alimentation dont les connecteurs sont liés verticalement.

Mais le plus simple pour comprendre est encore un dessin :

L'idée est alors de placer les composants en prenant en compte les connexions déjà existantes et en établissant les liaisons supplémentaires via des câbles/fils (*jump wires* en anglais) ou les pattes des composants eux-mêmes.

Précisons que la *breadboard* est une solution qui a ses limites. Même si pour des montages simples vous ne rencontrerez aucun problèmes particuliers, il peut arriver que des connexions ainsi réalisées ne soient pas assez « franches » pour assurer le fonctionnement optimal d'un circuit sensible (vitesse, fréquence, résonance, etc). Les connecteurs, la proximité des « pistes » ou la structure même de la *breadboard* peuvent provoquer des phénomènes électriques perturbant le montage. Il faut alors se tourner vers la réalisation d'un vrai circuit. Mais ceci n'entre pas en ligne de compte dans la plupart des projets initiaux.



Une breadboard dite « half+ » (demi-taille avec des rampes d'alimentation sur les côtés) et en superposition (à droite) une schématisation des connexions internes.

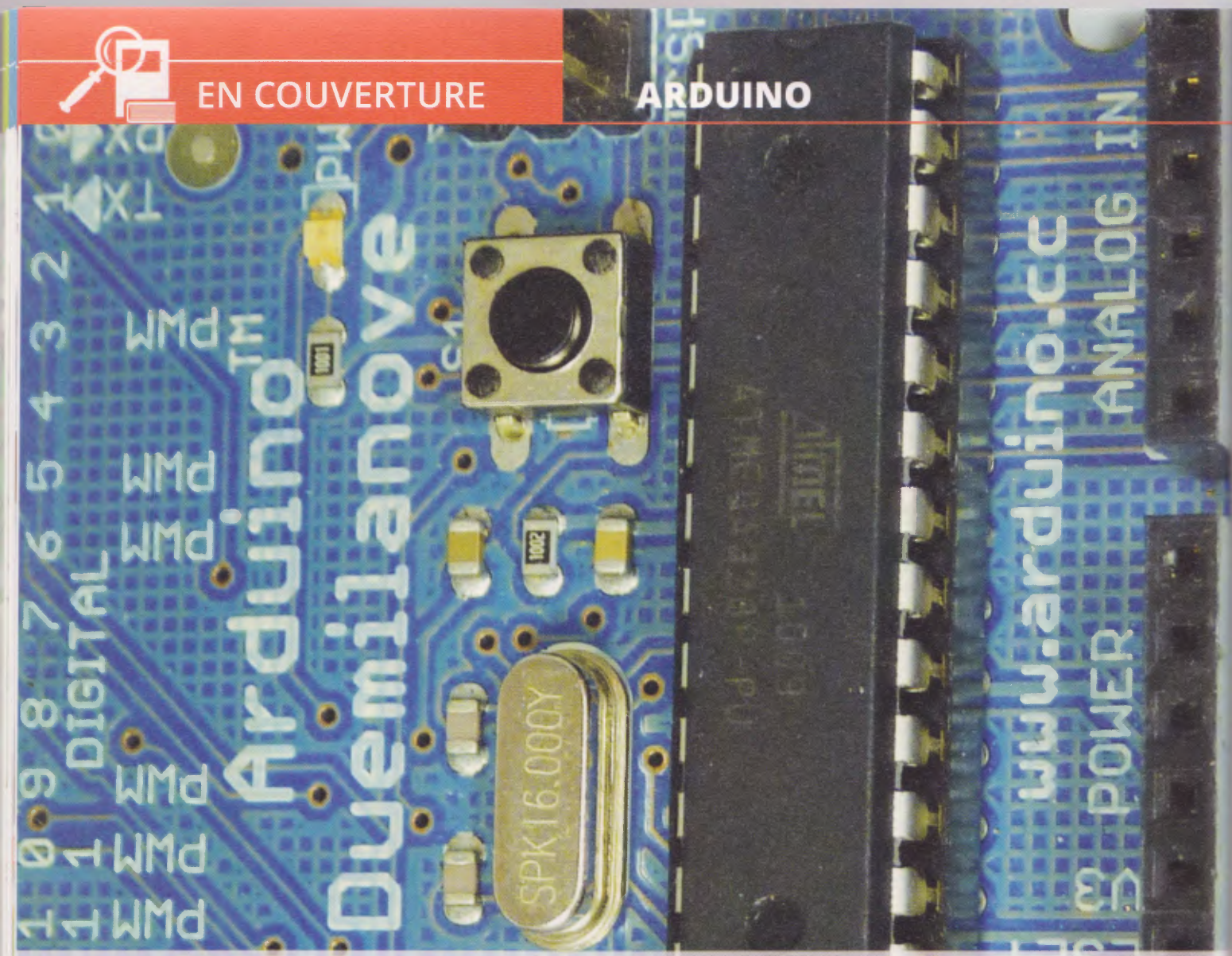
4. PRÊT À DÉMARRER !

Voilà, tout est prêt. Vous avez identifié et trié tous les composants, compulsé fébrilement les pages du manuel, il est temps de vous y mettre ! Un conseil : ne brûlez pas les étapes, même si les projets présentés à la fin du manuel semblent plus intéressants... Si vous souhaitez bien comprendre comment fonctionne cette petite carte et surtout, apprendre à concevoir des croquis par vous même, mieux vaut procéder pas-à-pas.

En conclusion, ce kit pour débutants peut être d'une très grande utilité... si vous êtes anglais ou américain. Dans le cas contraire, il ne permet de régler que le problème principal de tous débutants : réunir un peu tout ce qu'il faut pour réaliser quelque chose qui marche.

Au bout de quelque temps, à force d'acheter des composants, vous réunirez une magnifique collection. Il faut dire que les composants passifs comme les résistances et les condensateurs, par exemple, et une grande partie des actifs (opamps, transistors, diodes, etc) ont des prix qui se chiffrent en centimes ou vendu dans des quantités qui vous mettent à l'abri du besoin pendant très longtemps.

FB-DB



ARDUINO : UN PROJET, UNE RÉVOLUTION ET UNE GAMME DE CARTES

Né il y a presque 10 ans, dans un contexte tout à fait trivial, l'Arduino a tout bonnement révolutionné l'électronique de loisir et ce, à l'échelle mondiale. Aujourd'hui très populaire, il est devenu « la brique » incontournable de tous ceux qui souhaitent s'initier à l'électronique en toute simplicité. Une bonne imagination et un peu de temps libre suffisent désormais pour débuter et créer tout un tas de projets aussi amusants qu'utiles.

1. DES RACINES ITALIENNES

L'Arduino est né en 2005, dans le cadre d'un projet d'étudiants, à l'I*nteraction Design Institute Ivrea* (IDII) en Italie. À l'époque, Massimo Banzi, architecte logiciel et aujourd'hui co-fondateur du projet Arduino, fut engagé par l'institut pour enseigner de nouvelles méthodes de « design interactif »

L'origine du nom de cette petite carte remonte à bien longtemps... Pour la petite histoire, en l'an 1002, le roi Arduin d'Ivrée régna pendant un temps sur l'Italie après avoir pris le pouvoir de façon plus ou moins légitime, il fut donc rapidement détrôné par l'empereur Henri II. Aujourd'hui, un bar porte son nom à Ivrée, pour honorer sa mémoire. Il se trouve que ce bar était le point de chute de Massimo Banzi, qui nomma donc son projet... Arduino.

Initialement c'est un kit microcontrôleur BASIC Stamp de Parallax qui fut utilisé à l'époque pour la conception de la carte; toutefois, celui-ci présentait deux inconvénients majeurs : il n'était pas assez puissant pour réaliser les projets de certains élèves et les coûts atteignaient les $100 (bien trop cher pour des étudiants). En outre, le cahier des charges initial imposait que le matériel puisse être utilisé avec un Mac, plateforme très utilisé parmi les designers de l'IDII. C'est alors que Banzi décida de concevoir une carte qui réponde à tous leurs besoins à partir de zéro.

Fort heureusement, un collègue de Massimo venait justement de développer un langage de programmation adapté aux designers, nommé Processing. Ce dernier devint très vite populaire, car il ne demandait pas de compétences techniques particulières en programmation et s'accompagnait d'un environnement très facile à utiliser. Banzi s'est donc demandé s'il était possible de créer des outils logiciels similaires pour programmer un microcontrôleur.

Massimo Banzi, 2007. Photo : Matt Piddulph, publiée sous licence CC BY-SA 2.0.

Un étudiant du programme, Hernando Barragan, entreprit donc des recherches en ce sens et développa un prototype de plateforme nommé Wiring, qui comprenait une interface graphique sympa et un circuit imprimé prêt-à-l'emploi. Un projet plutôt prometteur donc, mais Banzi souhaitait quelque chose d'encore plus simple à prendre en main et surtout, bon marché. Son leitmotiv : la démocratisation de l'ingénierie.

L'ouverture de leur projet apparu comme une évidence. En outre, l'IDII, par manque de subventions, fut finalement obligé de fermer ses portes et les membres du projet craignirent alors que le « futur Arduino » tombe dans l'oubli ou entre de mauvaises mains. C'est la raison pour laquelle ils décidèrent de libérer rapidement les sources aussi bien du logiciel que du matériel. De ce fait, l'Arduino a fait office de précurseur dans l'application de la philosophie open source dans le domaine du matériel. On parle à présent d'open-hardware.

C'est ainsi qu'un modeste projet d'étudiants bouleversa le milieu de l'électronique dans le monde entier, en rendant la technologie plus accessible. Aujourd'hui, vous pouvez acquérir une carte Arduino pour 25 euros ou la construire vous-même sans avoir de droit et de licence à payer : tous les schémas et les codes sources nécessaires sont diffusés sous licence libre.

Depuis, le projet Arduino a prit de l'ampleur, la gamme de cartes c'est étendue et l'ensemble a gagné en qualité et en maturité, aussi bien d'un point de vue du matériel que des processus de fabrication mais également de l'aspect logiciel.

Aujourd'hui, l'Arduino est devenu presque un réflexe dès lors qu'on souhaite s'initier ou explorer le monde de l'électronique numérique. Des centaines de milliers de cartes ont ainsi été vendues à travers le monde, ainsi que des modules additionnels s'enfichant directement sur les cartes et appelées *shields* (de *to shield*, le verbe « couvrir » en anglais, rien a voir avec Nick Fury et Phil Coulson).

Arduino a permis de démocratiser ce domaine et surtout d'en facilité l'accès mais il ne l'a pas créé. Depuis très longtemps, des amateurs éclairés utilisent des microcontrôleurs pour leurs activités. Il y a plus d'une quinzaine d'années, nombreux sont ceux qui ont débutés avec le PIC 16F84 de Microchip (ah, les cours de Bigonoff, nostalgie, nostalgie) ou encore un des composants de la famille AT90 d'Atmel et pour les plus audacieux, le 68HC11 de Motorola. Il fallait cependant à l'époque disposer de matériel, construire ses cartes et trouver les éléments logiciels (compilateurs, exemples, docs). Voilà précisément ce qu'a simplifié Arduino qui, en un seul duo carte/environnement, vous permet de démarrer rapidement et sans prérequis.

2. L'ARDUINO À LA LOUPE

2.1 Le cœur de la carte : le microcontrôleur

Le composant principal de la carte Arduino est un microcontrôleur de marque Atmel (ATmega328, ATmega2560, ATmega168 ou ATmega8). Un microcontrôleur, c'est comme un ordinateur complet sur une seule puce. Il comprend donc un processeur, de la mémoire, une horloge interne, des périphériques et des ports d'entrée/sortie. Le microcontrôleur est un composant désormais très courant. On en retrouve partout, du réveil au réfrigérateur en passant par les voitures, les friteuses, les lave-linges, les chaînes hifi, les eCigarettes, les ascenseurs, les stations de lavage... Tout ce qui nécessite une « logique » pour fonctionner intègre aujourd'hui un ou plusieurs microcontrôleurs.

Un microcontrôleur se compose de plusieurs éléments. Parmi les plus importants, on trouve :

- Le processeur (ou CPU pour *Central Processing Unit*) est là, comme dans votre PC, pour décoder et exécuter des instructions qu'il reçoit, gérer les entrées/sorties et réagir aux événements.
- la mémoire de programme, qui accueille le programme à exécuter. Il s'agit aujourd'hui de mémoire de type flash mais fut un temps on trouvait surtout des EPROM (programmable une seule fois) ou des UVPROM (effaçable par exposition aux ultraviolets). D'autres microcontrôleurs n'avait pas de mémoire programmable mais juste un petit bout de code (en ROM) et il fallait utiliser une mémoire externe au composant.
- la mémoire vive destinée à stocker des données variables, qui changent au cours de l'exécution du programme. Là encore, c'est maintenant courant mais les premiers microcontrôleurs se limitaient à l'utilisation de toutes petites zones dans le processeur lui-même, appelés registres. La mémoire vive était, comme avec votre PC, externe au microcontrôleur.
- les périphériques intégrés au composant et offrants toute une gamme de fonctionnalités. On y trouve des *timer* (sortes de métronomes), des interfaces, des convertisseurs, etc.
- les ports d'entrée/sortie qui offrent une possibilité de connecter le microcontrôleur avec le monde extérieur, de lui ajouter des périphériques de toutes sortes ou d'interagir avec l'utilisateur (leds, écran, moteur, haut-parleur, etc).

Au final, le microcontrôleur n'est pas si différent d'un PC. Imaginez simplement prendre le processeur, la mémoire, le disque dur et les différents périphériques qu'il comprend (carte son, interface, etc), miniaturiser l'ensemble de façon drastique et réunir tout cela sur une puce d'un millimètre de côté. Et vous avez un microcontrôleur !

2.2 Les autres composants

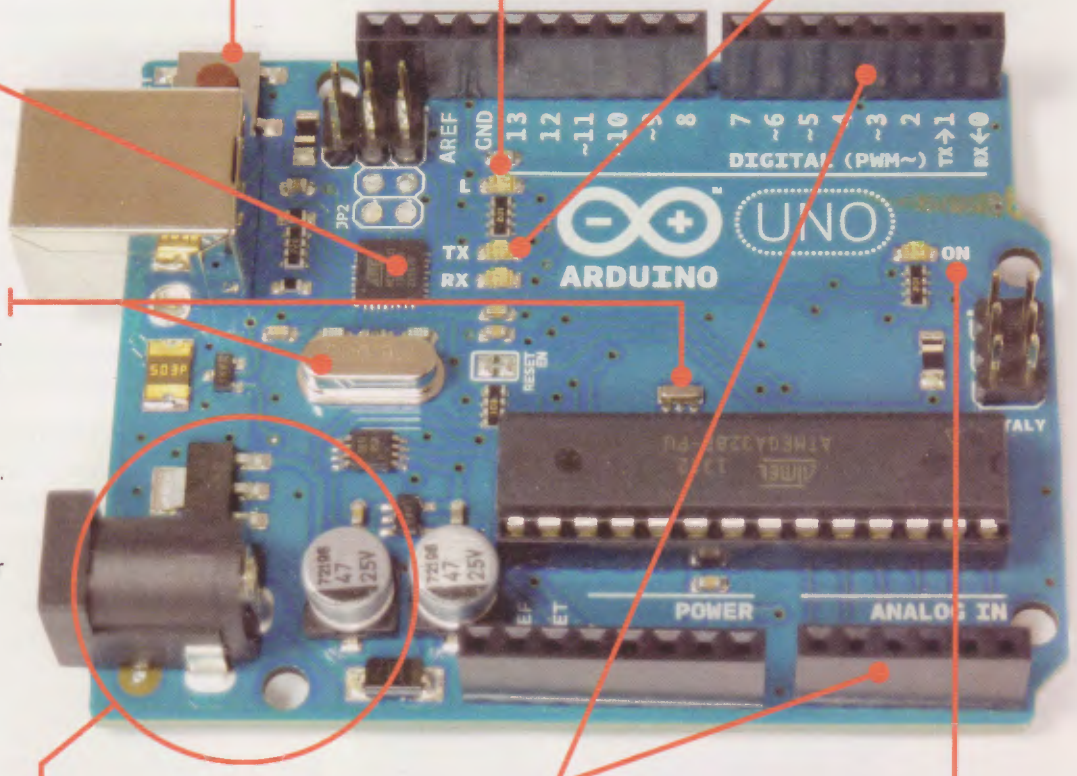
L'illustration ci-contre fait un tour d'horizon de ce qui compose une Arduino Uno.

1 Le composant permettant de servir de traducteur pour la connexion USB avec le PC. Certains Arduino utilisent pour cela une puce FTDI, les modèles plus récents accomplissent le même travail avec un autre microcontrôleur Atmel assez proche de celui utilisé comme base pour la carte elle-même.

5 Bouton poussoir permettant le redémarrage de la carte et donc du microcontrôleur.

7 La led L est un témoin lumineux utilisable par le programme qui fonctionne dans le microcontrôleur. Celle-ci offre à l'utilisateur la possibilité de tester, voir de se servir de la carte sans composant externe (du moins dans un premier temps).

3 Les leds RX/TX sont des petits témoins lumineux permettant de visualiser une activité aussi bien lors de la programmation (téléversement) que de l'utilisation de l'Arduino.



2 L'oscillateur à quartz produit un signal périodique, à une fréquence fixe et surtout précise. Il sert à cadencer le fonctionnement du processeur et des périphériques du microcontrôleur, comme un métronome. Notez ici la présence de deux quartz, un pour le microcontrôleur de base et un autre pour celui qui permet l'interface avec le PC.

6 Régulateur de tension et circuit d'alimentation. Une carte Arduino peut être alimentée par le port USB ou un bloc d'alimentation dédié. Cette partie de la carte gère à la fois le basculement d'une source sur une autre mais permet également de « nettoyer » le courant afin qu'il n'y ait pas de variation de tension, chose que n'apprécie que très moyennement un microcontrôleur.

8 Connecteurs d'entrée/sortie. Ils sont reliés directement aux ports d'entrée/sortie du microcontrôleur et permettent toutes sortes d'utilisation et de connexions avec des composants, des circuits et des modules. L'agencement des connecteurs et leurs nomenclatures est identique sur la plupart des cartes Arduino. De ce fait les modules additionnels appelés *shields* sont les mêmes pour tous les Arduino.

4 La led d'alimentation est directement lié au circuit d'alimentation de la carte. Elle permet à l'utilisateur de savoir que la carte est sous tension et qu'il est donc peu recommandé d'y connecter quoi que ce soit à ce moment.

Insistons quelques instants sur les ports et donc les connecteurs femelles à votre disposition sur les côtés d'une carte Arduino. Il en existe plusieurs types :

- Entrées et sorties digitales : le programme ou croquis enregistré dans la carte Arduino permet de placer chacune de ces broches ou lignes à la masse (état bas) ou de leur faire fournir une tension fixe (état haut), lorsqu'elles sont utilisées en sortie. En entrée, c'est l'inverse, le croquis va lire l'état d'une broche et pouvoir savoir si elle est branchée à la masse ou non.
- Sorties dites analogiques : alors qu'une broche digitale ne sait avoir que deux états, haut et bas, une sortie analogique peut, en principe, présenter n'importe quelle tension. Dans la pratique, ce n'est pas réellement le cas car les microcontrôleurs utilisés par la quasi totalité des Arduino ne savent simplement pas le faire. Pour obtenir quelque chose d'approchant, on utilise une technique appelée PWM qui consiste, grossièrement, à alterner les deux états possibles tellement vite qu'il semble s'agir d'un signal analogique. Un coup « on », un coup « off » et on peut dire que c'est « à moitié on ». Un coup « on », trois coup « off » et nous voici à 25% « on », etc. Toutes les sorties ne savent cependant pas le faire, celles le pouvant sont marquées d'un « ~ » avant leur numéro.
- Les entrées analogiques : là, il ne s'agit pas de tricher, l'Arduino est vraiment capable de mesurer une tension sur certaines broches, marquée « A » suivi d'un chiffre. Là, on pourra appliquer une tension entre 0 et 5 volts et le croquis fonctionnant dans la carte pourra traduire cela en valeur numérique.
- Les broches à fonction alternatives ou spéciales. L'ensemble des broches est utilisable en entrée ou en sortie digitale mais certaines d'entre-elles disposent de fonctionnalitées supplémentaires activables en utilisant des morceaux de croquis spécifiques appelés bibliothèques. Ainsi certaines broches peuvent communiquer avec des modules et *shields* (SPI, i2c, etc) ou encore provoquer une réaction de la part du croquis en fonctionnement (notion d'interruption). Notez au passage que la PWM est un mode de fonctionnement alternatif en soi.

Encore une fois, même si cela peut paraître complexe au premier abord, tout est grandement simplifié par l'environnement Arduino. En effet, la direction des broches et leurs éventuels changement d'état ne sont que l'affaire d'une petite instruction à placer dans le croquis, le tout avec une syntaxe facile à retenir. Les fonctions alternatives, quant à elles, sont presque totalement transparentes pour l'utilisateur puisque dans la plupart des cas ceci se résume à l'utilisation d'une entrée dans un menu (pour insérer la bibliothèque à utiliser).

3. LE CÔTÉ LOGICIEL

Arduino se compose de deux parties, une carte et aussi et surtout un logiciel à installer sur votre PC ou votre Mac pour écrire des croquis ou *sketchs* en anglais et les envoyer au microcontrôleur de la carte Arduino. Le logiciel utilisé est écrit en Java et est, dans les grandes lignes, le même pour Windows, Mac OS X et GNU/Linux (du moins la partie visible). C'est un environnement de développement ou IDE pour *Integrated Development Environment*.

Le langage utilisé pour programmer votre carte est inspiré de Wiring, lui-même tiré d'une autre langage appelé Processing (anciennement Proce55ing ou p5 tantôt pour les intimes). Processing comme Wiring et le langage Arduino se veulent simples et rapides à apprendre. Pourtant, si l'on y regarde de près, ils se base sur l'un des langages de programmation parmi les plus difficiles à maîtriser (non qu'il soit réellement complexe mais du fait qu'il fasse grandement reposer le bon fonctionnement des programmes sur le développeur lui-même).

Arduino a réussi le tour de force de permettre à vous, utilisateurs, de rapidement apprendre à écrire un croquis fonctionnel et à ensuite en savoir de plus en plus sans nécessairement vous heurter à des problèmes insurmontables. C'est donc un langage pédagogique qui, plus tard, vous permettra de basculer sur des langages plus courants dans le milieu de l'électronique numérique, comme le C ou le C++.

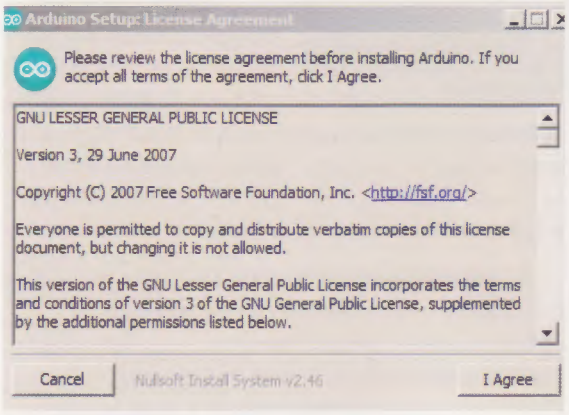
IDE Arduino est un environnement intégré. Il permet la saisie du code mais également sa transformation en quelque chose que le microcontrôleur de l'Arduino peut comprendre. On parle, pour ce type de langage, de compilation, la transformation du code source que vous comprenez, en code machine (ou langage machine) que l'Arduino comprend. L'environnement permet également de procéder au transfère du croquis compilé vers la carte. On parle alors de téléversement.

Bien entendu, l'IDE Arduino n'est pas livré avec votre ordinateur et vous devrez le récupérer sur le site Web officiel : **http://arduino.cc/en/Main/Software**. Là, choisissez la version qui correspond au système que vous utilisez et téléchargez simplement l'ensemble.

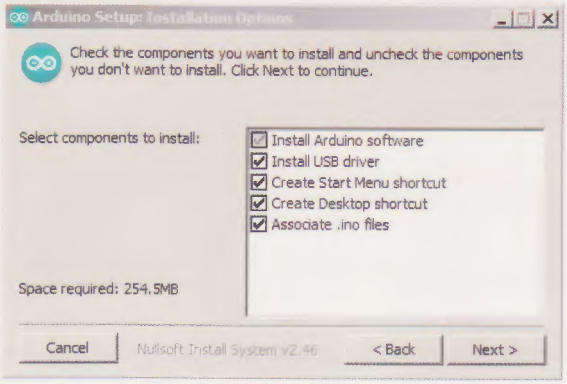
3.1 Installation pas-à-pas sous Windows

Récupérez tout d'abord le fichier exécutable à l'adresse citée précédemment (`arduino-1.0.5-r2-windows.exe`). Puis, comme d'habitude, lancez l'installation via un double-clic sur le fichier.

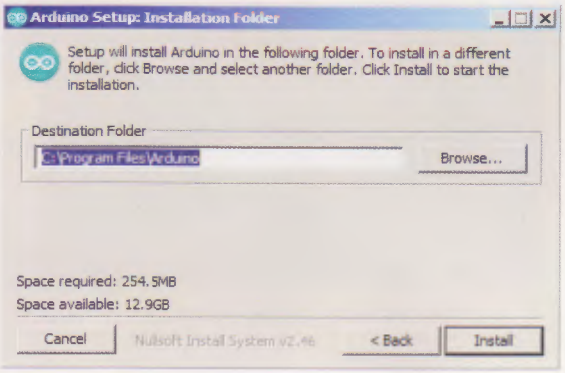
Après avoir lu puis accepté les termes de la licence GNU GPL, vous êtes invités à sélectionner les composants que vous souhaitez installer :



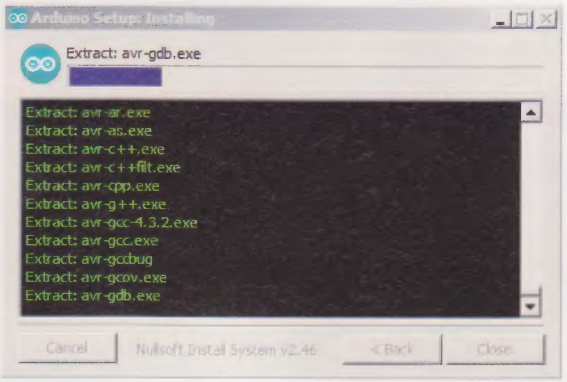
L'ensemble est sélectionné par défaut, à savoir le logiciel lui-même (obligatoire), le pilote USB, les raccourcis pour lancer le logiciel depuis le menu et le bureau, ainsi que l'association avec les fichiers `.ino` qui contiennent les croquis Arduino.



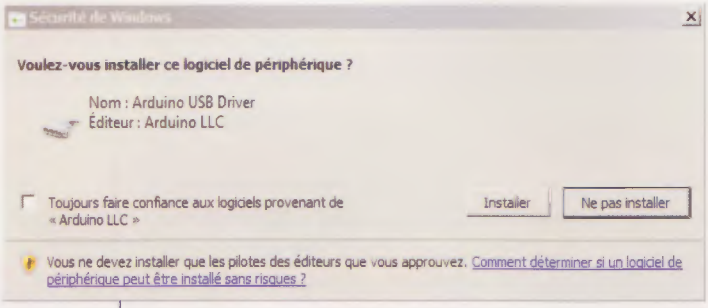
Ceci fait, l'étape suivante vous permet de renseigner le chemin vers le répertoire d'installation. Par défaut, il s'agit de `C:\Program Files\Arduino` :



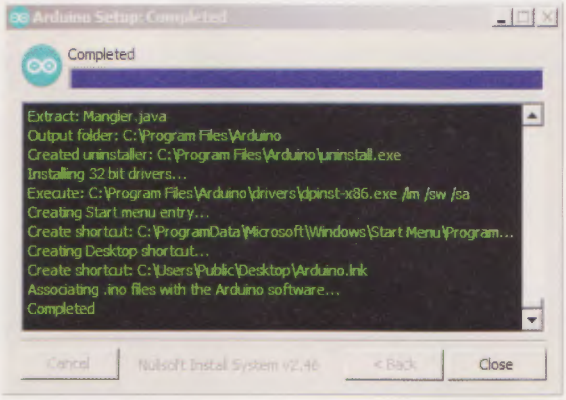
Validez cette emplacement par défaut et l'installation se lance :



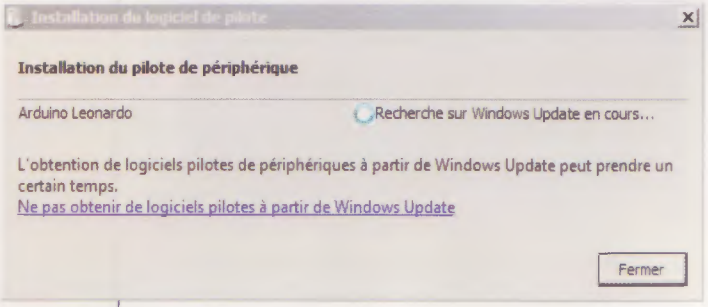
A un certain moment durant l'installation des fichiers sur votre ordinateur, on vous demandera s'il faut installer le « logiciel de périphérique » (un pilote dans la langue de Windows) :



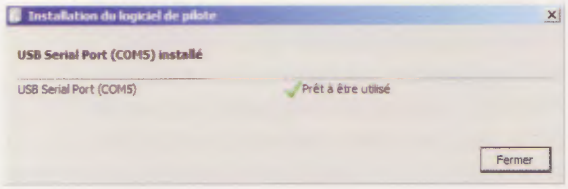
C'est un pilote signé et vous pouvez simplement valider l'installation qui pourra alors se terminer sans encombre :



Connectez ensuite votre carte Arduino à l'ordinateur à l'aide du câble USB, le témoin d'alimentation s'allume. Windows va alors détecter la carte et procéder à l'installation :



Comme le pilote a été précédemment installé, cette étape ne devrait poser aucun problème et se terminer avec succès :



Et voilà, il n'y a rien de plus à faire. Vous voici prêt pour l'aventure. Notez que Windows verra chaque carte Arduino comme un nouveau périphérique et installera le pilote automatiquement à chaque fois. Il en découle que chaque carte aura un port série (COM) associée et qu'il faudra, à chaque fois, vérifier ce point dans l'environnement de développement avant téléversement d'un croquis dans une carte fraîchement connectée. Si vous n'avez qu'une seule carte, vous n'avez à vous en soucier qu'une seule fois.

3.2 Installation sur Mac

L'environnement de développement Arduino existe en version Mac OS X. L'utilisateur Apple sera sans doute un peu surpris de constater que cette application est fournie sous la forme d'un fichier Zip (compressé) et non d'image disque (dmg) comme c'est généralement le cas sur cette plateforme.

Après téléchargement, vous devrez donc double-cliquer sur le fichier `arduino-1.0.5-macosx.zip` là où vous l'avez enregistré. L'utilitaire d'archive va alors le traiter et extraire l'application Arduino. Il ne vous restera plus qu'à prendre l'icône et à la glisser dans le dossier Application de votre Mac et, pourquoi pas, ensuite la glisser dans votre Dock.

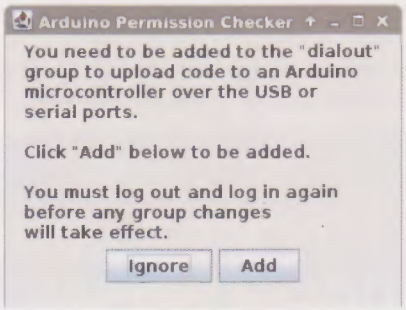
Au premier lancement, votre Mac vous rappellera qu'il s'agit d'une application provenant d'un téléchargement depuis Internet et vous demandera confirmation avant de l'ouvrir, par sécurité. Choisissez bien entendu « Ouvrir » et vous voici dans l'environnement Arduino.

Comme avec d'autres systèmes, une carte Arduino apparaîtra en tant que port série avec un nom comme `tty.usbserial` ou comme un modem nommée `tty.usbmodem`. Dans les deux

cas, le nom du port est suivi d'un numéro de série propre à chaque carte. La même remarque s'applique ici que celle faite concernant Windows : vous devrez vous assurer dans l'environnement de développement, via Outils -> Port série, que le bon port est sélectionné, puisqu'il change en fonction des cartes que vous branchez.

3.3 Installation sous Linux

Sous Ubuntu Linux, l'installation de l'IDE est on ne peut plus aisée : une simple sélection dans la logithèque suivi d'une installation et le tour est joué ! Ceci sera également valable pour de nombreuses autres distributions GNU/Linux puisque, comme l'IDE est open source, il est généralement intégré dans les projets de distribution sans aucune difficulté.



Au premier lancement, un message apparaît pour vous inviter à vous ajouter à un certain groupe d'utilisateurs du système afin de pouvoir téléverser vos croquis dans une carte Arduino. Le groupe en question est `dialout` sous Ubuntu. Le fait de vous ajouter en tant qu'utilisateur dans ce groupe vous donne le droit d'accéder au port (série) où est connectée la carte et surtout d'y écrire. Notez que pour que ce changement soit pris en compte, il vous sera nécessaire de vous déconnecter de votre session utilisateur, puis de vous reconnecter.

4. UN PETIT TOUR DANS L'IDE

L'IDE Arduino est écrit en Java et dans les grandes lignes et en tout point identique quelque soit le système que vous utilisez (merci le caractère multi-plateforme de Java).

Si l'interface devait ne pas être en français au premier lancement, rendez-vous dans les préférences du logiciel pour y remédier (menu File -> Preferences) et renseignez votre langue dans l'un des menus déroulants. Il sera ensuite nécessaire de quitter et relancer l'application Arduino.

Comme vous pouvez le constater, l'environnement Arduino est très épuré (voir simpliste diront certains). C'est totalement délibéré puisque le choix du projet est justement de garder les choses simples de manière à ce que vous puissiez vous concentrer sur le langage et vos croquis, sans avoir à vous perdre dans des dizaines de menus et fenêtres de configuration.

Sous la barre de menu se trouve une barre comprenant quelques boutons avec, de gauche à droite :

- « Vérifier » : un clic sur ce bouton et le croquis présent dans la fenêtre sera transformer en code pour la carte Arduino. En d'autres termes il sera compilé, traduit pour la machine... ou pas. Si vous avez fait des erreurs dans votre croquis, la vérification échouera et une erreur vous sera retournée.

- « Téléverser » : vous permet d'envoyer le croquis compilé dans l'Arduino. Si nécessaire, une vérification aura lieu. Seule une compilation réussie permettra de téléverser le croquis compilé. Tant qu'il restera des erreurs, vous ne pourrez donc pas tester le code sur la carte.
- « Nouveau » : ouvre une nouvelle fenêtre pour un nouveau croquis.
- « Charger » : charge un croquis existant depuis votre carnet de croquis (qui n'est qu'un dossier dans vos documents) ou depuis les exemples livrés avec l'environnement de développement.
- « Enregistrer » : sauvegarde le croquis en cours.
- « Moniteur série » : une carte Arduino peut communiquer avec l'ordinateur avec lequel vous travailler. Plus exactement, vous pouvez faire en sorte qu'un de vos croquis accepte et envoi des informations à votre ordinateur pour vérification ou tout simplement pour vous offrir une interface pour le contrôler. Un clic sur ce bouton vous ouvrira une nouvelle fenêtre contenant un sorte de système de messagerie instantanée avec votre carte Arduino.

La prise en main de l'environnement s'arrête presque à la connaissance de ces boutons. Un seul autre élément important est le choix de votre carte Arduino dans l'environnement. Le même langage est utilisé pour toutes les cartes Arduino (et même certaines autres comme la Launchpad de Ti et son environnement Energia) mais les cartes elles-mêmes sont différentes. Techniquement, ce n'est pas vraiment votre affaire mais celle du compilateur. Tout ce que vous avez à faire est d'indiquer votre modèle de carte à l'environnement. Pour ce faire, passez par le menu Outil -> Type de carte et faites votre choix.

Enfin, la carte une fois connectée apparaît comme un port série. A chaque fois que vous brancherez une nouvelle carte, vous disposerez d'un nouveau port (du moins sous Windows). Il faudra donc faire un petit tour dans Outils -> Port série et sélectionner celui correspondant à la carte. Généralement, avec un PC moderne il est peu probable qu'un autre port série existe en dehors de celui proposé par l'Arduino. Vous n'en trouverez donc sans doute qu'un seul dans ce menu. Notez que dans le coin inférieur droit de la fenêtre de l'IDE, le port actuellement utilisé est noté. Il peut ne pas correspondre à celui de la carte si vous en avez débranché une pour en brancher une autre.

Dans le ***menu Fichier > Exemples*** (ou en passant par le bouton « Charger »), vous trouverez de nombreux exemples très intéressants pour apprendre à vous servir de votre carte Arduino. De plus, via l'entrée « 10.StarterKit », vous retrouverez les exemples associés au kit de découverte détaillé dans l'article précédent. N'hésitez pas à fouiller et ouvrir de nombreux exemples de croquis pour vous imprégner de la logique de l'ensemble et explorer les fonctionnalités une à une.

CONCLUSION

Cette courte présentation d'Arduino n'est qu'une base pour débuter. C'est en forgeant qu'on devient forgeron dit-on. Pour le travail du métal, c'est difficile de savoir si l'adage est toujours valide mais en ce qui concerne l'électronique numérique moderne et surtout la programmation (de croquis ou de n'importe quoi d'autre) c'est bel et bien le cas. C'est en faisant qu'on apprend !

Vous voici, comme bien d'autres, les bienvenues dans le monde qui se cache derrière ce que la plupart des gens utilisent tous les jours sans même s'en rendre compte, celui des microcontrôleur, des circuits, des montages et des composants. Chaussez vos bottes de randonnées car la route qui se déroule sous vos pieds est longue, semée de surprises et de petits problèmes croustillants à régler. Bienvenue dans le monde en plein ébullition de l'Arduino ! **FB-DB**

ABONNEZ-VOUS !

6 NOS **39*** €/AN

*Prix France Métropolitaine

CONSULTEZ L'ENSEMBLE DE NOS OFFRES SUR :

boutique.ed-diamond.com

DISPONIBLE EN VERSION PAPIER ET/OU PDF

Pour plus d'informations, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

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

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

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

HACKABLE MAGAZINE
DÉMONTEZ | COMPRENEZ | ADAPTEZ | PARTAGEZ

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

Vos remarques :

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

DÉCOUVRIR ET APPRENDRE LE LANGAGE ARDUINO

Une carte Arduino est une sorte d'ordinateur miniature très simpliste et comme tout ordinateur celle-ci se programme. Le composant principal de l'Arduino, le microcontrôleur, exécute des programmes que vous avez écrit en utilisant un langage particulier. Nous allons découvrir maintenant à quoi ressemble ce langage, comment l'utiliser et apprendre à ordonner à l'Arduino ce qu'il doit faire.

Peut-être avez-vous connu les premiers ordinateurs personnels ou micro-ordinateur. En un temps reculé, bien avant l'arrivé des PC dans le bureau ou le salon de nos logis, le paysage de l'informatique personnel était bien différent. Les passionnés utilisaient des ordinateurs incompatibles les uns avec les autres, chaque fabricant suivant son bonhomme de chemin et tentant d'écraser la concurrence. Des ordinateurs comme le Commodore 64/128, le Thomson TO7 ou MO5, le Sinclair ZX80/ZX81/Spectrum ou l'Amstrad CPC464 se partageaient le marché. Toutes ces machines avaient en commun, entre autres choses, le fait qu'à la mise sous tension vous vous retrouviez avec un écran vous invitant à programmer, généralement en Basic ou à saisir une instruction dans un langage précis pour le faire réagir. Ces ordinateurs arrivaient chez vous presque vides, on ne pouvait s'en servir qu'en programmant. Une autre particularité de l'époque était que ces machines ne pouvaient exécuter qu'un seul programme à la fois. La notion de système d'exploitation était réservé à des ordinateurs plus puissants et celle de *desktop* graphique quasi-inexistante.

L'Arduino c'est un peu la même chose. La carte ne sait faire qu'une chose à la fois et vous devez la programmer pour vous en servir. On notera d'ailleurs avec amusement que le microcontrôleur qui anime l'Arduino est bien plus puissant, sur certains points, que l'ordinateur le plus vendu au monde de toute l'histoire de l'informatique, le Commodore 64.

Notez que dans ce qui va suivre, un certain nombre de notions est simplifié afin de faciliter la compréhension globale. Un puriste y verra des points qui mériteraient davantage d'explications, voir des inexactitudes. Nous avons choisi ici de ne pas être exhaustif et de rester le plus général possible de manière à ne pas saturer cet article d'introduction d'informations qui ne prendront un sens que plus tard dans la découverte de la programmation Arduino. La démarche est parfaitement délibérée et le risque de sur-simplification connu.

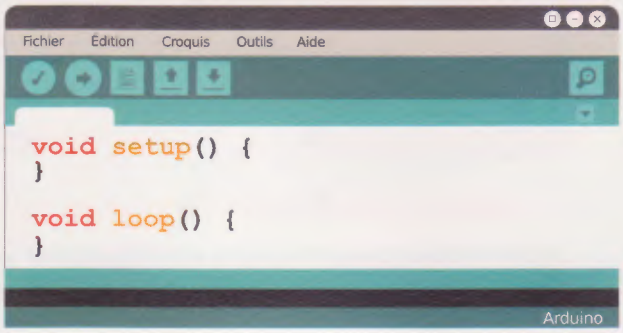
1. DANS LES GRANDES LIGNES ET AVANT DE COMMENCER

Votre programme qui trouvera sa place dans la mémoire de l'Arduino est appelé un croquis ou *sketch* en anglais. Cependant, il ne peut pas être copier tel quel. Il doit être transformé de manière à ce que ce qui est compréhensible pour vous, le code du croquis, le devienne pour l'Arduino ou plus exactement pour le microcontrôleur qui l'équipe. Cette transformation passe par plusieurs étapes qui sont communément regroupées sous le terme « compilation ». Votre croquis est donc compilé puis téléversé (uploadé) dans la mémoire de la carte. Après un *reset* ou redémarrage, ce programme sera automatiquement exécuté par l'Arduino. Le redémarrage est d'ailleurs provoqué automatiquement après le téléversement.

La compilation est déclenchée dans le logiciel Arduino (on parle d'environnement de développement ou d'IDE pour *Integrated Development Environment*) en cliquant sur le bouton « Vérifier » et le chargement dans la mémoire de l'Arduino connecté via le bouton « Téléverser » qui procédera également à une compilation du croquis avant l'opération.

Une chose importante qui peut paraître surprenante, même si on programme déjà sur PC par exemple, tient dans le fait qu'un programme fonctionne en permanence. Plus exactement, le microcontrôleur exécute toujours un programme et ne s'arrête jamais. Votre ordinateur personnel fait la même chose et si vous avez l'impression qu'il ne fait rien, ce n'est qu'une illusion car le processeur est toujours entrain de faire fonctionner un programme. Avec l'Arduino, c'est bien plus visible qu'avec un PC car il n'y a pas de système d'exploitation, c'est donc votre programme qui fonctionne directement, en boucle, encore et encore.

C'est un élément pédagogique crucial et c'est pourquoi la structure même d'un croquis Arduino présente cela de manière explicite. Voici le croquis Arduino le plus simple possible :



Nous n'entrerons pas dans le détail immédiatement. Attachons-nous ici à l'ensemble pour en comprendre la structure. Nous avons deux **fonctions**, `setup()` et `loop()`. Le code que nous allons écrire trouvera sa place entre `{` et `}`. C'est la **portée** des fonctions, ce qui en défini le début et la fin. La différence entre `setup()` et `loop()` est la suivante :

- `setup()` contient un morceau de programme qui sera **exécuter une seule fois** lorsque l'Arduino sera mis sous tension ou redémarré. *setup* signifie mise en place ou configuration. C'est l'objet de cette fonction : régler des points en début de programme, configurer des éléments spécifiques et définir la configuration. Voyez cela comme le code de démarrage.

- `loop()` est très différent. *loop* signifie boucle et le code qui sera placé là sera exécuté... en boucle. Ligne après ligne, tout ce qui se trouve dans la portée de `loop()` est exécuté. Arrivé au `}` de fin, l'Arduino recommencera et exécutera une autre fois le contenu de la fonction, encore et encore...

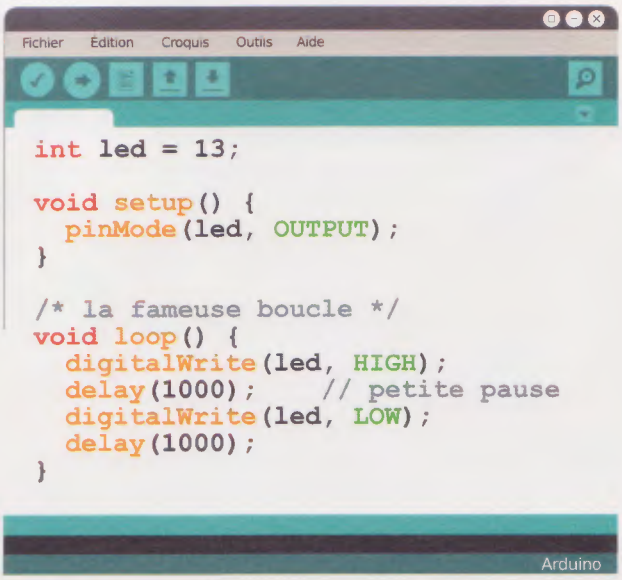
Cette séparation vous permet de ne pas avoir à penser à créer vous-même une boucle et vous ne risquez donc pas d'oublier de le faire (oublier `setup` ou `loop` provoquerait une erreur). Comm.e nous l'avons dit précédemment, un processeur, aussi bien celui de votre PC que celui dans le microcontrôleur de l'Arduino, ne peut rester sans rien faire. Si cela arrivait, il s'arrêterait et ne pourrait plus se réveiller sauf par un *reset*.

Vous pouvez parfaitement saisir le croquis minimaliste, le compiler et le téléverser dans l'Arduino. Il ne fait rien, ni au démarrage ni ensuite, mais va le faire... en boucle ! `loop()` sera appelé indéfiniment, même s'il n'y a rien a exécuter.

2. MON PREMIER PROGRAMME

2.1 Aspect généraux

A présent que nous connaissons dans les grandes lignes de quoi est fait un croquis, il est temps d'écrire notre premier programme vraiment utile. Le voici :



Vous devez reconnaître immédiatement nos chères fonctions `setup()` et `loop()` mais qui, cette fois, contiennent des lignes de codes. Nous avons également une ligne qui se trouve en dehors des portées de ces deux fonctions, nous y reviendrons plus tard mais avant cela, parlons des commentaires.

Un croquis Arduino est constitué d'une suite d'instructions. Chaque terme, chaque mot a un sens et possède une signification. Si vous utilisez une instruction que l'environnement Arduino ne comprend pas, le croquis ne sera pas compilé et une erreur sera signalée. De la même manière, s'il manque des éléments à votre programme, qu'un terme est mal « orthographié » où que vous faites référence à quelque chose qui n'existe pas, une erreur empêchera la compilation.

Il nous est cependant permit de placer du texte dans le croquis afin de faire office de notes, de remarques ou de mémos : ce sont des commentaires. Il y a deux manières de placer un commentaire dans un croquis :

- Placer sur une ligne deux barres de fraction, `//` : tout ce qui suit jusqu'à la fin de la ligne ne sera pas considéré comme du code et tout simplement ignoré lors de la compilation,

- mettre notre texte entre `/*` et `*/` : dans ce cas, peu importe le nombre de ligne, tout ce qui se trouve entre `/*` et `*/` sera ignoré.

Les commentaires permettent non seulement de placer du texte explicatif dans le croquis mais également de désactiver certains morceaux de code. En effet, si nous voulons temporairement que l'environnement ignore une ligne, pour un essai, il nous suffit de la faire précéder de //. S'il s'agit de tout un bloc de code à désactiver, on utilisera généralement /* sur une ligne avant ce morceau de code et */ après.

Plusieurs occurrences d'espace et de tabulation ne compte que comme une seule séparation. De la même manière, les sauts de lignes n'ont, le plus souvent, aucune importance. Ceci :

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

est absolument identique à cela :

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

ou encore à cela :

```
void

setup(){pinMode(led,
      OUTPUT)
  ;
}
```

Pour la compilation, ceci ne fait aucune différence mais pour nous oui. La façon de placer les instructions impacte la lisibilité du croquis. Il existe plusieurs écoles dans ce domaine, plusieurs façons d'écrire du code lisible. De manière générale, on essai de faire en sorte de rendre la porté de chaque fonction évidente, d'où le fait de faire précéder les lignes d'espaces pour les décaler par rapport au reste. Cette pratique se nomme **indentation**. Une autre bonne habitude consiste à utiliser des espaces dans les lignes pour clairement faire apparaître les termes qui les composent.

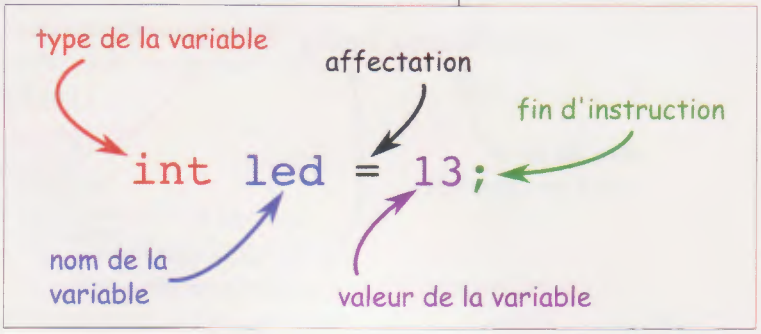
Enfin, dernier point important à propos de la syntaxe en générale, les **instructions** se terminent par un point-virgule. C'est le caractère qui marque la fin d'une instruction et non le saut de ligne. C'est un peu comme le point dans une phrase qui marque la fin de cette dernière.

2.2 Les instructions utilisées

Nous avons utilisé ici plusieurs instructions qui constituent des *ordres* que nous donnons dans notre croquis, des directives que le compilateur va traduire et transformer en quelque chose que l'Arduino peut comprendre. Nous allons découper notre croquis de manière à les étudier en détail.

```
int led = 13;
```

Exemple de déclaration de variable avec affectation.



Cette première ligne, à l'allure bien innocente, contient bien des choses à apprendre et à connaître. Nous avons là ce qu'on appel une « déclaration d'une variable globale avec une affectation ». Pas de panique, pour comprendre découpons davantage. Commençons par transformer cela en deux lignes donnant un résultat équivalent :

```
int led;
led = 13;
```

Nous avons maintenant en première ligne une déclaration de variable. Un ordinateur, comme l'Arduino, stock des « choses » dans sa mémoire. Si nous vulgarisons le concept, une variable est une « case » mémoire avec une étiquette marquée d'un nom, comme un récipient (ou un bocal à anchois) sur une étagère. Le langage de l'Arduino, qui est « dérivé » du C/C++, est statiquement typé. Ceci signifie que vos récipients ne peuvent pas stocker n'importe quoi n'importe comment. Si vous dites avoir un bocal à anchois, vous n'avez pas le droit de mettre des cornichons dedans, c'est un bocal pour les anchois (il est possible de forcer mais oublions cela pour l'instant).

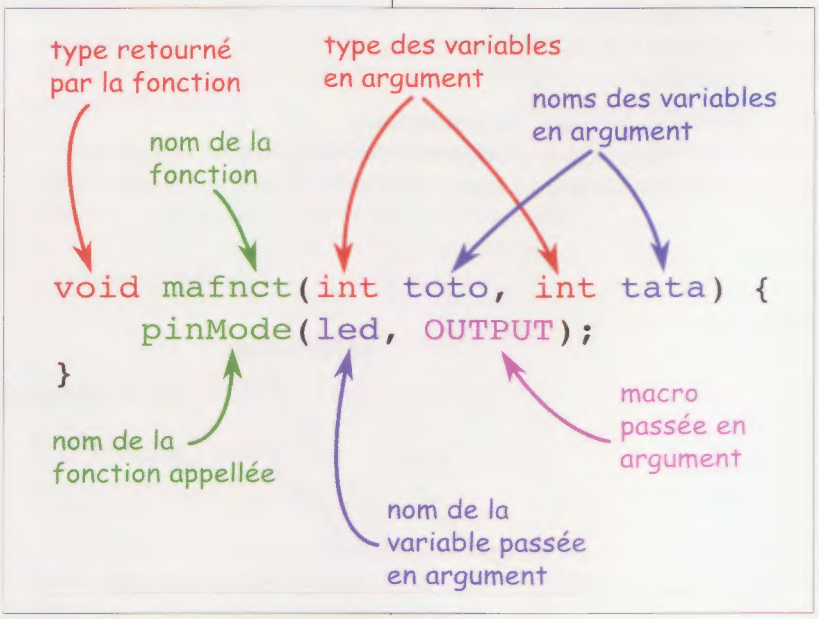
Ici nous avons **déclaré** une variable appelée `led` censée contenir une valeur numérique entière (`int` pour *integer*). Nous venons de créer un récipient à nombres entiers (sans virgule).

La ligne du dessous, nous remplissons le récipient avec une valeur : nous **affectons** la valeur `13` à `led` avec le symbole `=`. Dès lors, nous pouvons utiliser `led` comme s'il s'agissait de la valeur `13`. La seule différence avec la ligne originale tient dans le fait que les deux opérations ont lieu en même temps, en une seule instruction.

Peut-être vous demandez-vous ce qui se passerait si nous ne faisions pas l'affectation. La variable `led` contiendrait-elle zéro ? Dans ce cas précis oui mais c'est une particularité car il s'agit d'une **variable gobale**. Comme elle est déclarée en dehors de la porté d'une fonction, elle est valable pour tout le programme et utilisable partout. Lorsqu'une variable de type `int` (par exemple) est déclarée globale, elle est automatiquement initialisée à `0`.

Il n'en va pas de même si nous faisons cette déclaration dans `setup()` par exemple. Là `led` n'existerait QUE dans `setup()` mais pas en dehors et surtout, aucune valeur ne sera affecté automatiquement. Que contiendra alors `led` ? On parle de contenu indéterminé (*undefined*) ou, en d'autres termes, on ne sait pas. Bien entendu, une variable ne peut pas ne rien contenir, il s'y trouvera forcément quelque chose, mais certainement

Exemple de déclaration et d'utilisation de fonctions.



pas ce qu'on souhaite puisque nous n'y avons rien mis explicitement. Cela peut être un bout de mémoire quelconque, le contenu d'une autre variable ou même un bout du programme lui-même... On ne sait pas, mais ça ne doit pas arriver ! La compilation n'entraînera pas d'erreur car le programmeur (c'est vous ça) est censé savoir ce qu'il fait. Le compilateur va tout de même vous faire part de son étonnement sous la forme d'un avertissement (genre *"warning: 'xxx' is used uninitialized in this function"*) mais, si dans les préférences de l'environnement Arduino vous n'avez pas choisi d'afficher les résultats détaillés pendant la compilation, vous ne le verrez même pas. Je le répète donc : ça ne doit pas arriver ! Jamais ! Les variables non initialisées c'est le mal !

Profitons-en pour signaler une source d'erreur tantôt courante chez les débutants : la variable `led` n'est pas la variable `Led` ou `LED`. Le langage utilisé par l'Arduino est sensible à la casse de caractères. On évitera cependant de trop jouer sur ce point car même s'il peut être pratique d'avoir une variable `led` et une autre `Led` c'est là clairement une source de confusions et de bugs dans vos croquis. Il ne vous est pas possible non plus d'utiliser des noms comme `int`, `void`, etc.

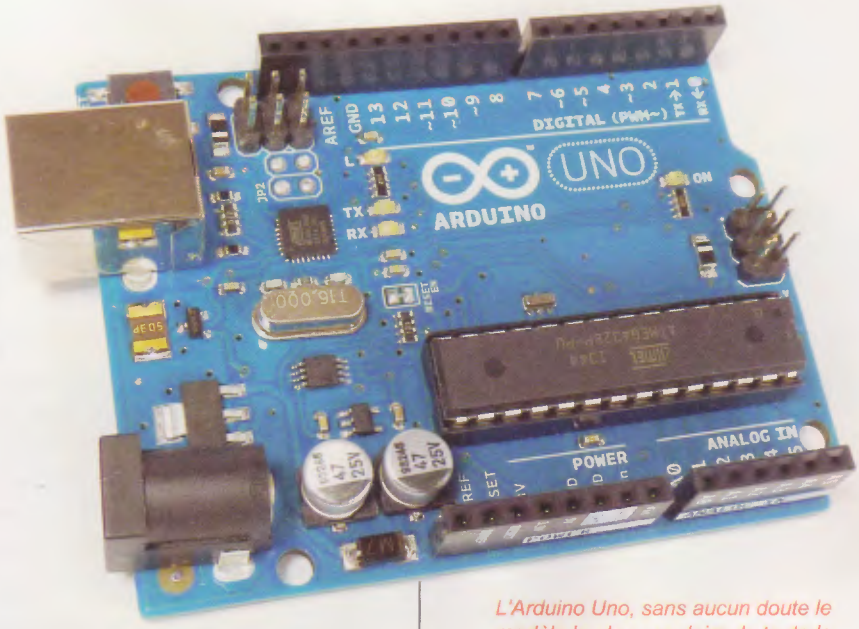
Poursuivons maintenant avec la suite du croquis de démonstration :

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

Nous avons déjà parlé de `setup()` précédemment mais il est temps de nous pencher sur la question sérieusement. Ceci est une déclaration de fonction. Le principe est un peu le même que celui des variables puisque nous « annonçons » que nous définissons une fonction. Vous pouvez voir les fonctions comme des commandes ou des instructions (elles en sont en fait), sauf qu'ici nous créons l'instruction. Elle possède un nom, `setup`, mais également un type. Ici, son type est `void` qui signifie « vide » ou « rien ». Le type d'une fonction est en réalité le type de donnée qu'elle retourne après son exécution. Imaginez que vous fabriquez un lutin magique (si, si). Votre lutin a pour seule tâche de prendre un bocal à anchois (décidément) et de compter les anchois qu'il contient. Lorsque vous voulez compter les anchois, vous ordonnez au lutin de compter le contenu d'un bocal et de vous dire le nombre d'anchois qui s'y trouve. Si notre lutin était une fonction, il ressemblerait à ceci :

```
int lutin() {
  int nombre;
  nombre = compte(12);
  return nombre;
}
```

et vous l'appelleriez ainsi :



L'Arduino Uno, sans aucun doute le modèle le plus populaire de toute la famille. Ici, l'une des dernières versions (R3) avec son ATmega328p et un second microcontrôleur AVR juste au-dessus du quartz de 16 Mhz qui remplace la puce FTDI des précédentes versions. Au menu, 32 Ko de flash, 2 Ko de SRAM et 1 Ko d'EEPROM. C'est la carte par excellence pour débuter avec Arduino.

L'Arduino Mega2560 est l'une des plus « grosses » cartes de la famille Arduino 8 bits. Avec ses 256 Ko de flash et 8 Ko de SRAM elle permet de créer des croquis énormes et d'utiliser un nombre incroyable de fonctionnalités et de bibliothèques. Remarquez également le nombre de ports et de broches à votre disposition pour vos montages.

Voici l'une des cartes les plus récentes, l'Arduino Leonardo. Utilisant un ATmega32u4 (32 Ko de flash et 2,5 Ko de SRAM), elle est plus ou moins équivalente à la Uno en termes de fonctionnalités. La principale différence avec les modèles plus anciens est l'utilisation d'une puce Atmel qui prend en charge l'USB. Ceci évite d'utiliser un composant supplémentaire (FTDI ou Atmel) pour la connexion au PC, mais offre également la possibilité de faire apparaître la carte comme une souris, un clavier ou un port série dans vos croquis.

```
int quantite;
quantite = lutin();
```

Dans `quantite` se trouverait alors le nombre d'anchois et le lutin disparaîtrait. En cas de second appel, un autre exemplaire du lutin serait appelé et retournerai également une valeur. L'instruction `return` quitte la fonction et défini la valeur retournée par la fonction, les derniers mots du lutin avant de disparaître en somme. Notez que `nombre` dans le corps de la fonction `lutin()`, n'existe qu'à cet endroit, dans le lutin, car la variable est déclarée **localement** et non globalement. A chaque invocation, `nombre` est « créé » et utilisé, puis disparaît avec le lutin.

Nous pouvons même aller plus loin et créer un lutin générique, capable de compter n'importe quel bocal :

```
int lutin(int num_bocal) {
  int nombre;
  nombre = compte(num_bocal);
  return nombre;
}
```

et nous l'appellerons ainsi :

```
int quantite;
quantite = lutin(12);
```

Entre les parenthèses, nous fournissons un **argument** à la fonction. Cet argument est ce que le lutin attend pour exécuter sa tâche. Nous en avons ici défini un seul argument qui doit être une valeur numérique entière (`int`) mais il est possible d'en utiliser plusieurs, qui sont alors séparés par des virgules. L'argument de type `int` que nous avons appelé `num_bocal` n'existe que dans la fonction, c'est une variable locale et il n'est pas nécessaire de faire d'autres déclarations (si nous tentons d'ailleurs de déclarer une variable `num_bocal` ceci provoquera une erreur). Bien entendu, comme l'argument attendu est d'un certain type, en utiliser un autre provoquera une erreur.

Revenons maintenant à notre morceau de croquis :

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

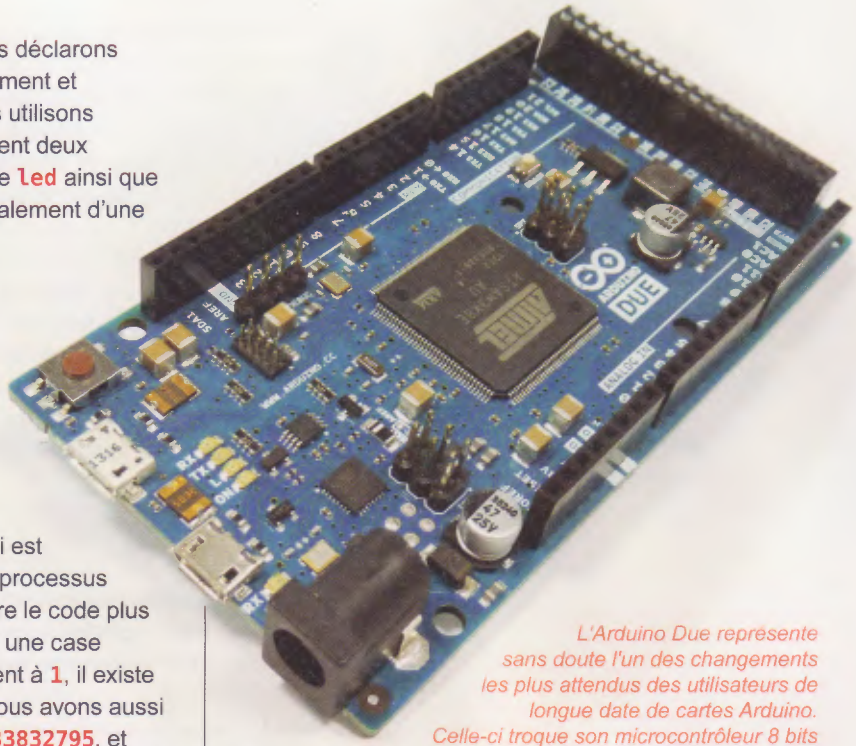
Nous pouvons dire maintenant que nous déclarons la fonction `setup` qui ne prend aucun argument et ne retourne rien. Dans cette fonction, nous utilisons une fonction `pinMode` qui prend en argument deux paramètres. Ici nous utilisons notre variable `led` ainsi que `OUTPUT`. On pourrait croire qu'il s'agit là également d'une variable mais il s'agit d'une macro.

Une macro n'a de sens que pour le compilateur lui-même. Voyez cela comme une sorte d'alias. Dès que le compilateur « verra » `OUTPUT` il va le remplacer automatiquement par quelque chose qui est défini dans l'environnement Arduino. Ceci peut ressembler à une variable mais c'est très différent. Une macro est quelque chose qui est remplacé dans le croquis au tout début du processus de compilation. C'est une manière de rendre le code plus lisible et/ou plus concis mais en aucun cas une case mémoire, un bocal. Ici `OUTPUT` est équivalent à `1`, il existe aussi `INPUT` qui est `0` par exemple, mais nous avons aussi `PI` qui est `3.14159265358979323846264338327795`, et beaucoup d'autres encore. Voici un bon exemple, comme la macro est définie, il est bien plus simple d'utiliser `PI` dès qu'on souhaite écrire `3.14159265358979323846264338327795`. C'est là l'un des principaux intérêts des macros.

Poursuivons enfin avec le reste du croquis :

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

Nous déclarons la fonction `loop` qui ne retourne rien et ne prend aucun argument. Dans cette fonction nous appelons la fonction `digitalWrite` avec deux arguments, toujours notre variable `led` et une autre macro, `HIGH`. Notez que les macros peuvent avoir n'importe quel nom du moment qu'il est unique et ce aussi bien en majuscule qu'en minuscule. L'utilisation des majuscules est un choix arbitraire ou plus exactement un style de programmation (*coding style*). Il est ainsi d'usage de donner des noms en majuscules aux macros de manière à lever l'ambiguïté quand à leur nature. Respectez cette habitude, utilisez des noms de variables en minuscule et des noms de macros (si vous venez à en définir) en majuscule.



L'Arduino Due représente sans doute l'un des changements les plus attendus des utilisateurs de longue date de cartes Arduino. Celle-ci troque son microcontrôleur 8 bits pour un monstrueux Atmel SAM3X8E à coeur ARM Cortex-M3. À la clé, un nombre incroyable de fonctionnalités supplémentaires, une puissance de calcul très importante (84 MHz), 512 Ko de flash et 96 Ko de SRAM. Elle est, bien entendu, plus chère et présente des problèmes de compatibilité électrique puisqu'elle ne supporte pas le 5 volts (cela endommagerait la carte), mais uniquement 3,3 volts. Cette nouvelle génération de carte ne semble actuellement largement pas aussi populaire que la simple Arduino Uno, bien moins puissante.

L'Arduino Nano est physiquement bien plus petite que le reste de la famille, mais totalement équivalente en termes de capacités et de fonctionnalités que, par exemple, la Uno puisqu'elle utilise le même microcontrôleur. Remarquez la différence dans les broches utilisées. Ici, la carte est faite pour s'enficher sur une breadboard et dispose donc de connecteurs mâles sur le dessous.

`delay` est également un appel à une fonction mais cette fois celle-ci ne prend qu'un seul argument. On spécifie ici la valeur 1000 mais, bien entendu rien ne vous empêche de déclarer une variable et de lui affecter cette valeur pour ensuite l'utiliser en argument de `delay`. Cependant, n'utilisez pas des variables lorsque cela n'est pas nécessaire. Le compilateur fera de son mieux pour économiser les ressources et en particulier la mémoire car, dans le microcontrôleur de l'Arduino Uno par exemple, il n'y a que 2 Ko de mémoire. Ceci peut paraître sans importance avec un PC par exemple et ses 4, 16 ou 32 Go de mémoire, mais celle-ci est toujours précieuse. Une variable de type `int` par exemple consommera 2 octets et avec de petit croquis ce ne sera pas un problème que d'en utiliser à tout va. Prenez de bonnes résolutions au plus vite car arrivera forcément un moment où vous manquerez de place et vous devrez alors revoir vos habitudes : économisez la mémoire !

2.3 Liens avec le matériel : mais que fait ce programme ?

Ce second croquis exemple n'est pas comme le premier, il fait réellement quelque chose : il fait clignoter la led de l'Arduino. Aucun composant supplémentaire n'est nécessaire, la carte elle-même est suffisante. Rappelons le croquis :

```
int led = 13;

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

/* la fameuse boucle */
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);     // petite pause
  digitalWrite(led, LOW);
  delay(1000);
}
```

Notre variable `led` contient `13`. Cette valeur est un numéro de broche de l'Arduino. Or la broche 13 est également connectée à la led de la carte via une résistance. Dans la fonction `setup()` appelée une seule fois après un *reset*, nous utilisons la fonction `pinMode()` qui détermine le mode opératoire de la broche. Celle-ci peut être une entrée, `INPUT` ou une sortie `OUTPUT`. La fonction prend en argument un numéro de broche et un mode. Ici, la broche 13 est donc passée en sortie, nous pourrons alors y écrire.

Dans la fonction `loop()` nous utilisons deux fonctions. `digitalWrite()` permet d'écrire sur une broche. Nous pouvons ainsi passer cette dernière à l'état haut, `HIGH` pour qu'elle affiche +5 Volts ou bas, `LOW` pour qu'elle soit à 0 Volts, la masse. `delay()` est une fonction qui prend en argument une durée en millisecondes (millième de seconde). Ici nous lui passons 1000 pour 1000 millisecondes soit 1 seconde.

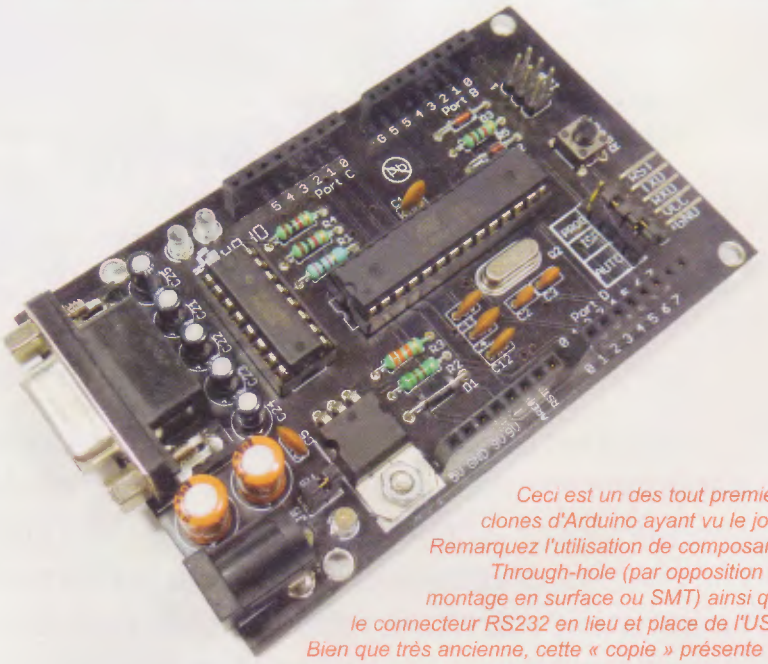
Notre fonction va donc passer la broche 13 à l'état haut, ce qui allumera la led. Elle marquera ensuite une pause d'une seconde puis passera la broche à l'état bas. La led n'est plus alimentée et donc s'éteint. Enfin, nous marquons à nouveau une pause d'une seconde.

Comme `loop()` est appelé en boucle infinie, notre led va s'allumer, s'éteindre, s'allumer... et ainsi de suite : elle clignote. Félicitation, vous venez d'écrire votre premier croquis qui fait réellement quelque chose avec l'Arduino !

3. PAS POUR CONCLURE...

Vous venez de suivre un cours accélérer et très sommaire sur le B.A.BA de l'Arduino et l'écriture de croquis. Vous avez la base mais il y a encore vraiment beaucoup de choses à découvrir, car nous n'avons fait qu'effleurer la surface. Avec le temps, l'expérience et grâce aux articles de ce magazine, vous en apprendrez de plus en plus sur la manière d'écrire vos croquis. Avec un Arduino, vous pouvez faire toutes sortes de choses, faire clignoter une led bien sûr, 100 leds pourquoi pas, actionner des moteurs, lire des températures, afficher du texte ou des images, surveiller des capteurs, aller sur le net, communiquer avec votre PC, jouer de la musique, lire une carte SD, etc. Tout cela très simplement, en connectant des modules et en utilisant des morceaux de codes d'autres programmeurs (des bibliothèques) simplifiant l'usage de ce que vous connecter à votre ou vos Arduino. Il n'y a quasiment aucune limite si ce n'est celles imposées par votre imagination, votre temps, votre ténacité... et les lois de la physique ! **DB**

Voici un clone tel qu'il en existe un certain nombre. Cette carte est vendu sur eBay et bien qu'incluant un ATmega328p comme l'Arduino Uno, elle présente un certain nombre de caractéristiques qui découlent directement d'une tentative du fabricant de minimiser les coûts de production : les condensateurs sont bas de gamme, la sélection de la source de courant se fait par cavalier, etc. La carte reste bien entendu ici tout à fait utilisable, mais d'autres clones sont bien plus douteux.



Ceci est un des tout premiers clones d'Arduino ayant vu le jour. Remarquez l'utilisation de composants Through-hole (par opposition au montage en surface ou SMT) ainsi que le connecteur RS232 en lieu et place de l'USB. Bien que très ancienne, cette « copie » présente un certain nombre d'avantages pour les bricoleurs parmi lesquels la facilité de remplacement des composants et surtout la robustesse de l'ensemble. Le plus amusant est qu'elle est toujours parfaitement utilisable avec l'environnement de développement Arduino actuel pour peu que l'on dispose d'un port série ou d'un adaptateur USB/série.

LOI D'OHM : TOUTE RÉSISTANCE EST FUTILE

... si elle n'est pas correctement calculée. L'univers est régi par des lois et ces lois s'appliquent implacablement sans moyen d'y échapper. Ces lois de la physique, contrairement à celles des hommes, n'ont pas été promulguées ou décidées, elles ont été découvertes. Les lois de la thermodynamique expliquent pourquoi et comment votre café refroidis, celle de la gravitation vous dit pourquoi votre tartine tombe et celles de l'optique géométrique pourquoi il est utile de mettre vos lunettes en vous levant le matin. En électricité et en électronique, il existe également des lois. L'une d'elle est plus incontournable que les autres : la loi d'Ohm.

L'électricité, élément indispensable de la vie aussi bien au sens biologique du terme que pour votre confort quotidien. Ici, dans les pays industrialisés, nous ne pourrions quasiment plus vivre sans électricité et ce au point que nous avons généralement tendance à l'oublier. Regardez autour de vous et imaginez ce qui ne fonctionnerait plus si le courant venait à disparaître, ce à quoi vous n'aurez plus accès. La maîtrise de l'électricité, issu d'une succession de découvertes, de recherches et d'expérimentations à conduit à la découverte de principes et de lois qui définissent le comportement de cette énergie. Élément si vital pour nous, l'électricité mérite qu'on sache à quelles règles elle obéit.

Cet article explique l'une de ces lois, fondamentale et indispensable à toutes personnes qui souhaite utiliser l'électricité et donc à vous qui avez fait le choix de vous intéresser ou vous passionner pour l'électronique.

1. QU'EST CE QUE L'ÉLECTRICITÉ ?

Avant toutes choses, il est important de définir précisément de quoi on parle. Il est souvent étonnant de constater que, même si l'électricité est partout, la grande majorité des personnes n'en connaissent pas la vrai nature et le sens des mots qui permettent de la décrire.

L'électricité, du mot grec *elektron*, est un phénomène physique. C'est le mouvement des charges électriques dans la matière. Une telle charge est une propriété fondamentale de la matière et il en existe deux formes : positive et négative. Les grecs anciens connaissaient déjà ce phénomène, découvert en frottant de l'ambre jaune sur de la fourrure (*elektron* signifie d'ailleurs ambre jaune). Plus tard, au XVIIIème siècle, on fait des progrès et on distingue l'électricité vitreuse et l'électricité résineuse et peu après, Benjamin Franklin a la brillante idée d'imager l'électricité comme un fluide. Ainsi il parle d'un courant qui se forme entre une matière qui possède trop peu de ce fluide et trop de fluide. Il décide que l'électricité vitreuse obtenue en frottant du verre sur de la soie est la charge positive et l'ambre frotté sur la fourrure se charge négativement.

Ce choix a été fait arbitrairement car les connaissances de l'époque ne permettaient pas d'en savoir davantage. Les conséquences de ce choix ont toujours des répercussions aujourd'hui car en réalité le flux peut avoir deux causes différentes : le déplacement de charges négatives sous forme d'électrons ou le déplacement de charges positives sous forme de protons. A notre échelle, le plus souvent, ce sont les électrons qui circulent dans un conducteur (métallique) comme les pistes d'un circuit imprimé. Ceux-ci, étant des porteurs de charges négatives, se déplacent donc du négatif au positif mais on parle cependant de courant positif allant du positif au négatif, car on préfère imaginer le courant comme positif. On utilise les termes de « courant conventionnel » en référence au modèle de Franklin dans lequel le courant est constitué de l'écoulement de particules positives même ci ce n'est pas le cas dans la réalité. Après tout, ce n'est pas bien grave, des charges négatives se déplaçant dans un sens, c'est un peu la même chose que des charges positives se déplaçant dans l'autre, du moment que tout le monde est d'accord...

2. L'INCONTOURNABLE MÉTAPHORE

L'idée de Benjamin Franklin d'imaginer l'électricité comme un fluide à ses limites mais cela reste toujours un excellent point de départ. De plus, comme la plupart des termes utilisés se basent sur cette idée, cela semble bien plus intuitif.

La métaphore du fluide ou de l'électricité s'écoulant comme de l'eau est très pratique. Ainsi, on peut imaginer le **courant** comme un fluide qui se déplace du positif au négatif. Le conducteur électrique devient un tuyau dans lequel le courant s'écoule et qui possède certaines propriétés.

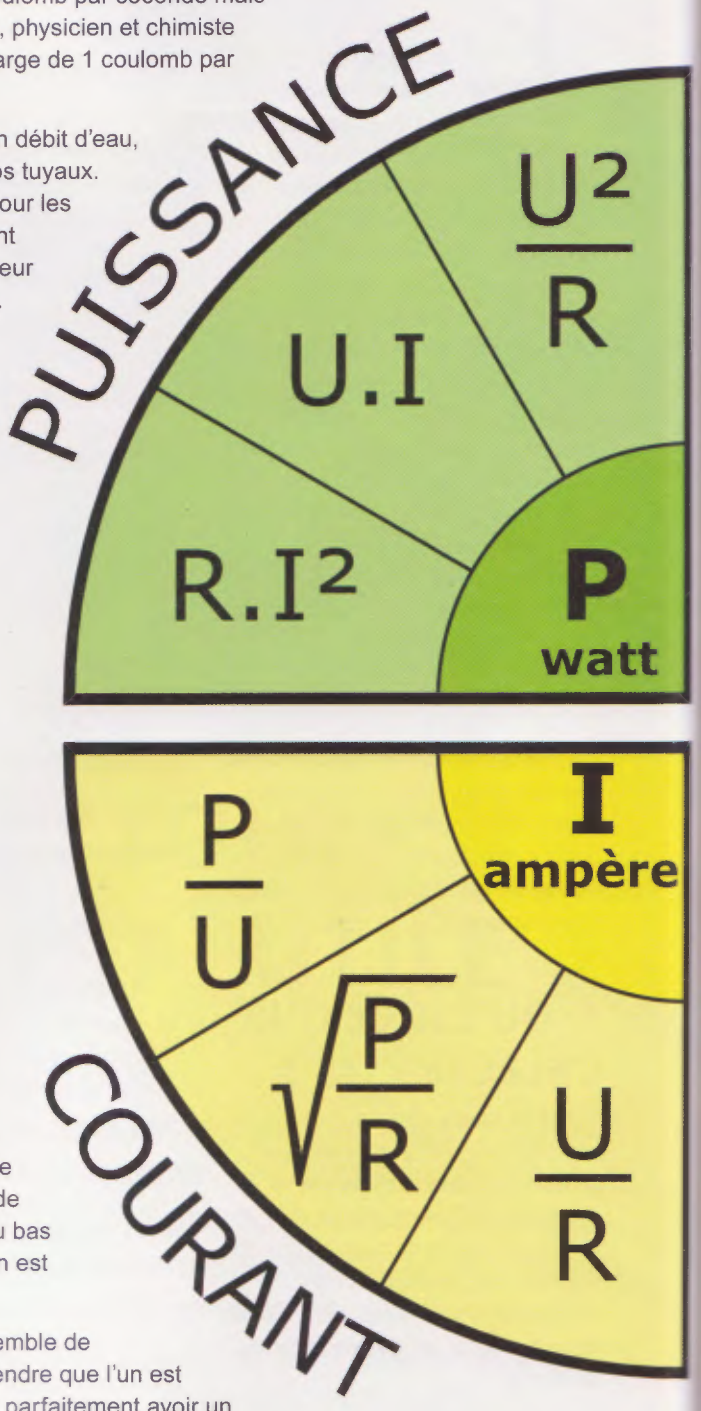
2.1 Courant

Le **courant** comme celui d'une rivière correspond à un débit mais, en lieu et place de mètres-cube d'eau sur une certaine durée par seconde, nous avons un nombre de charges par seconde sous forme de coulomb

par seconde (le nom de l'unité vient du physicien français Charles-Augustin Coulomb). Pour simplifier les choses, on utilise rarement le coulomb par seconde mais l'**ampère** (de André-Marie Ampère, mathématicien, physicien et chimiste français) qui correspond au déplacement d'une charge de 1 coulomb par seconde. Le symbole de l'ampère est **A**.

On parle d'intensité du courant et tout comme un débit d'eau, pour transporter beaucoup de courant il faut de gros tuyaux. Ainsi, par exemple, le courant maximum autorisé pour les prises électriques standards est de 16 A. Le courant maximum que peut fournir une prise USB d'ordinateur est de 500 mA (milli-ampère soit 1/1000 d'ampère).

De la même manière que l'eau dans des canaux, le courant circule dans un circuit. Imaginez bien cela comme un écoulement. Si aucun courant ne circule, il n'y a pas de charges électriques qui se déplacent et donc pas de courant.



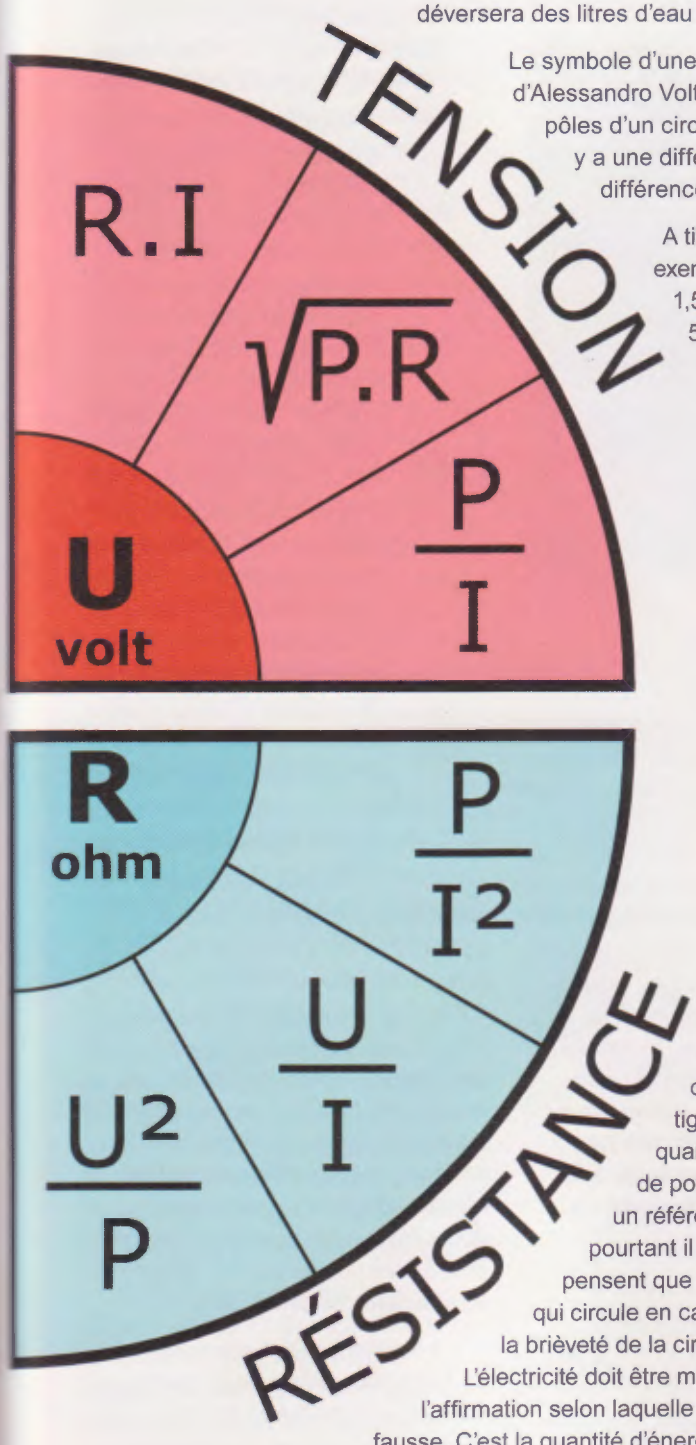
2.2 Tension

Pour poursuivre avec l'analogie des fluides, il existe une autre propriété très importante qu'est la **différence de potentiel** ou **tension**. Imaginez un tuyau reliant un réservoir d'eau placé au sommet d'une colline à un autre se trouvant au bas de cette dernière. En ouvrant le robinet une certaine quantité de liquide va s'écouler avec un débit déterminé par le diamètre du tuyau. Mais un paramètre est à prendre en compte : la pression.

La pression de l'eau peut servir à imager la notion de tension électrique. Il est préférable cependant, dans un premier temps, de vous contraindre à utiliser les termes « différence de potentiel » plutôt que « tension » car l'illustration est plus évidente. Entre le réservoir plein en haut de la colline et celui vide au bas, il y a une différence de potentiel. L'eau *veut* descendre dans le réservoir du bas et plus le premier réservoir est haut plus la pression est importante.

Même si tension et courant sont liés par un ensemble de relations décrites par des lois, il est aisé de comprendre que l'un est relativement détaché de l'autre. Ainsi, vous pouvez parfaitement avoir un tuyau avec une pression énorme, impossible à boucher du pouce, mais dont

ne s'écoule l'eau que goutte à goutte. Inversement, vous pouvez avoir une situation dans laquelle il est aisé de boucher l'embout du tuyau mais qui une fois libéré déversera des litres d'eau en quelques minutes.



Le symbole d'une tension est **U** et son unité est le **volt** (du nom d'Alessandro Volta, physicien italien) de symbole **V**. Ainsi, aux pôles d'un circuit (entre deux points chargés différemment) il y a une différence de charges entre un côté et l'autre, une différence de potentiel.

A titre d'exemple, une pile alcaline type AAA par exemple, présente une tension à ses bornes de 1,5 volts. Votre Arduino utilise des tensions de 5 volts entre une sortie active et la masse. A ce propos, une différence de potentielle, mesurée par un voltmètre se fait, entre deux bornes ou pôles. Il est donc possible de mesurer une tension n'importe où dans un circuit. Ce qu'on nomme « masse » est un simple point de référence même si, dans la quasi-totalité des cas, le potentiel électrique de la masse est la référence à 0 volts du circuit. La législation française défini la masse ainsi : partie conductrice d'un matériel électrique susceptible d'être touchée par une personne, qui n'est normalement pas sous tension. On mesure ainsi généralement une différence de potentiel ou une tension par rapport à la masse, qui est commune entre les différentes parties d'un circuit. La masse de votre Arduino connecté à votre PC est la même que la masse du PC lui-même.

Contrairement à une idée répandue selon laquelle c'est l'intensité du courant qui est dangereuse et non la tension, il faut bien comprendre que ce n'est pas le cas. En effet, une tige en verre frottée sur de la soie accumule une quantité très importante de charges. Une différence de potentiel énorme existe donc entre cette tige et un référentiel non chargé (ou chargé inversement) mais pourtant il n'est pas dangereux de la manipuler. Beaucoup pensent que c'est parce qu'il n'y a pas beaucoup de courant qui circule en cas d'établissement d'un circuit mais c'est plutôt la brièveté de la circulation du courant qui est à mettre en cause. L'électricité doit être manipulée avec précaution dans tous les cas et l'affirmation selon laquelle c'est le courant qui tue et non la tension est fausse. C'est la quantité d'énergie totale qui peut présenter un danger.

3. ENERGIE ET PUISSANCE

Nous avons un courant qui circule dans un circuit et une différence de potentiel entre deux points. Mais à quelle quantité d'énergie ou à quel flux énergétique total avons-nous affaire ?

On utilise généralement le joule comme unité pour quantifier l'énergie ou le travail, tantôt également les calories. La relation entre les deux unités est : 1 calorie = 4,1855 joules. Une calorie est, par exemple, la quantité d'énergie pour augmenter la température d'un gramme d'eau de un degré. Mais cette unité correspond à une quantité d'énergie et non une puissance ou un flux. Elle est donc rarement utilisée en électricité et électronique.

Pour une puissance, symbole **P**, on utilise une unité appelée un **watt** (de James Watt, ingénieur écossais) notée **W**. En appliquant une puissance d'un watt pendant une seconde on obtient un joule. Ceci signifie, dans un monde parfait, qu'en appliquant une puissance de 4,1855 watts pendant une seconde, vous pouvez faire augmenter la température d'un gramme d'eau d'un degré.

En regardant votre facture d'électricité (pas trop longtemps, c'est déprimant) vous verrez des quantités spécifiées en kWh ou kilowatt-heure. 1 kWh est équivalent à 1000 Wh (watt-heure) soit une puissance de 1000 watts utilisée pendant une heure. En faisant quelques calculs simples, ceci nous donne 3 600 000 joules, soit 860 112 calories.

Trois résistances de type, d'âge et de modèle différents telles qu'on en trouve dans tous les appareils électroniques qui nous entourent.

Vous l'avez compris, si votre fournisseur d'électricité vous facture des kilowatt-heure c'est qu'il existe une relation entre le courant, la tension et la puissance. Et fort heureusement cette relation est d'une simplicité exemplaire puisqu'elle se résume à la formule P = U x I. La puissance en watts est égale à la tension en volts fois l'intensité du courant en ampères.

A titre d'exemple, si votre carte Arduino est alimentée en faisant parcourir un courant de 40 mA (milliampère) à une tension de 5 volts (la tension du port USB), celle-ci utilisera une puissance de 0,04*5 = 0,20 watts. Laissez-là connectée ainsi un jour entier et vous aurez consommé 4,80 Wh ou 0,0048 kWh, ce qui vous coûtera environ 0,00067 euros (à 14 centimes le kWh, dont 36% de taxes).

Vous pouvez maintenant vous amuser à faire plein de calculs pour connaître la quantité d'énergie que votre aspirateur ou votre PC consomme et combien il vous coûte mais la notion de puissance est également importante à plus petite échelle. Certains composants, par exemple, comme les résistances, sont conçues pour une puissance maximum (1/4 de watt le plus souvent). Si vous ne respectez pas ces spécifications, le composant ne se comportera plus comme prévu, il risque de surchauffer, de se dégrader... bref, vous allez aux devants de problèmes.

3.1 Résistance

Poursuivons dans la métaphore hydraulique. Que se passe-t-il lorsque vous écrasez un tuyau dans lequel circule de l'eau ? Réponse : le débit diminue car vous limiter la quantité d'eau qui peut circuler. Avec l'électricité c'est exactement la même chose mais pour limiter l'intensité du courant on utilise une **résistance**. C'est un composant que vous avez sans doute déjà vu car présent sous bien des formes un peu partout dans nos équipements électroniques.

La résistance tient son nom du fait qu'elle résiste au passage du courant. Certains matériaux sont dit conducteurs, d'autres sont

diélectriques et ne peuvent pas conduire le courant électrique (se sont des isolants). Cependant, tous les conducteurs ne se valent pas et possèdent tous une certaine résistance. Seuls des matériaux bien particulier peuvent laisser passer le courant sans aucun résistance mais uniquement dans des conditions bien spécifiques. Ce sont les supraconducteurs et il ne s'utilisent généralement qu'à des températures très basse (entre -140°C et -238°C) et tantôt dans des conditions de pression atmosphérique contrôlées. Sans cela, ils présentent une résistance et n'ont donc rien d'exceptionnel.

Les matériaux conducteurs présentent donc tous une certaine résistance. Dans le cas du cuivre utilisé pour les pistes des circuits imprimés, celle-ci est négligeable mais d'autres matériaux, comme le graphite, résiste davantage au passage du courant. Pour quantifier la résistance notée **R**, l'unité utilisée est le **ohm** et son symbole est le **Ω** (la lettre grec oméga). Le mot ohm fait référence au nom du physicien allemand Georg Simon Ohm.

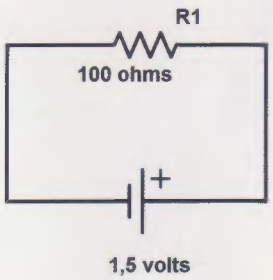
Avec une résistance, on peut limiter la quantité de courant et la formule de calcul est, là aussi, très simple : U = R x I (parfois noté U=RI ou U=R.I). En d'autres termes, la tension en volts est égale à la résistance en ohms multipliée par l'intensité du courant en ampère.

Cette formule résume la loi de Ohm qui nous dit que « la différence de potentiel aux bornes d'une résistance est proportionnelle à l'intensité du courant électrique qui la traverse ». En d'autres termes, quand un courant passe dans une résistance, il se forme une différence de potentiel entre l'entrée et la sortie, et cette tension est déterminée par la valeur de la résistance et le courant qui passe. Plus l'intensité est grande, plus la différence de potentielle est importante pour une résistance fixe.

Les potentiomètres rotatifs, comme il en existe pour contrôler le volume d'enceintes multimédia par exemple, sont des résistances variables. En tournant l'axe central, on déplace un point de contact sur une surface. Comme la résistance est fonction de la distance, la valeur de cette résistance varie.

Là encore la métaphore hydraulique nous aide. Quand vous pincez un tuyau, vous réduisez le débit mais il se forme une différence de pression entre l'un et l'autre côté du point pincé.

Imaginons le circuit suivant :



La résistance utilisée est de 100 ohms et la tension appliquée à ses bornes, provenant par exemple d'une pile alcaline, est de 1,5 volts. Question : quel est l'intensité du courant qui circule ? La réponse en chiffres :

U = R x I

donc :

U / R = I

et

1,5 / 100 = I = 0,015

Le courant qui passe a donc une intensité de 0,015 ampères, soit 15 mA. L'énergie en oeuvre peut également être calculée puisque nous savons déjà que P = U x I.

On voit clairement sur ces panneaux à leds conçus pour une alimentation en 5 volts que les résistances sont de tailles différentes. La résistance du panneau de gauche comportant 9 leds doit dissiper bien moins d'énergie que celle du panneau de droite avec ses 24 leds.

De ce fait :

P = U x I

P = 1,5 x 0,015 = 0,0225

Ce circuit consomme donc 0,0225 watts ou 22,5 mW.

Peut-être vous demandez-vous où va l'énergie ? L'un des principes de base de la thermodynamique (encore une loi), le principe de conservation de l'énergie, nous dit que l'énergie est toujours conservée et ne peut ni être créé ni disparaître. Ces 22,5 mW ne disparaissent donc pas, ils sont transformés d'énergie électrique en... énergie thermique. La résistance va dissiper la puissance par effet Joule qui prend corps sous la forme de chaleur. La formule de calcul de la puissance dissipée par la résistance est P = R x I^2, soit : la puissance en watts est égale à la résistance en ohms fois l'intensité du courant en ampères au carrée. Ce qui est logique car :

U = R x I

et

P = U x I

donc, en remplaçant le U de la seconde formule par son équivalence :

P = (R x I) x I

ou

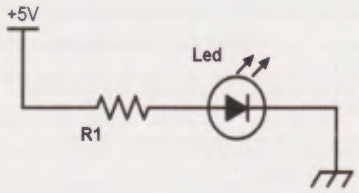
P = R x I x I = R x I^2

4. UN CAS PRATIQUE POUR ASSEOIR TOUT CELA

Voyons maintenant un cas concret pour bien ancrer ces principes simples mais relativement théoriques. Une carte Arduino utilise une tension

de 5 volts par rapport à la masse lorsqu'une de ses sorties est passée à l'état haut (sortie active). Si vous souhaitez connecter une led, vous ne pouvez pas simplement la brancher entre la sortie en question et la masse car ce composant possède des spécifications bien précises.

Une led rouge plus ou moins courante peut être décrite par le constructeur comme nécessitant pour fonctionner un courant de 12 mA (*DC forward current* dans la documentation) et à ses bornes apparaîtra alors une tension de 1,4 volts (*Forward voltage* noté Vf). Pour une led, la tension est une caractéristique et le courant une spécification. Nous devons donc mettre une résistance de manière à limiter le courant qui passe dans le circuit et donc dans la led. Pour cela nous devons calculer sa valeur.

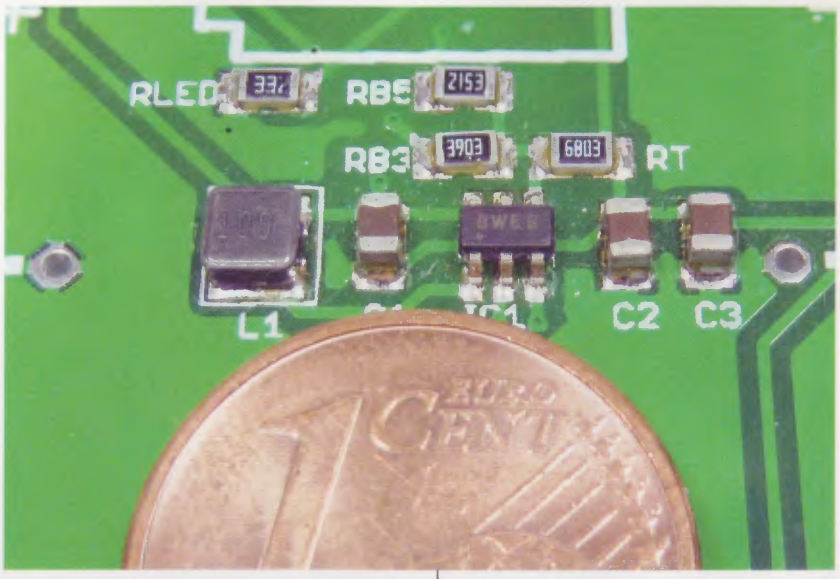


Nous avons une tension de 5 volts entre la sortie de l'Arduino et la masse commune. Mais nous savons que la led à ses bornes fera apparaître une tension de 1,4 volts. Entre la sortie de l'Arduino et la broche positive de la led nous aurons donc forcément une tension de 5-1,4 = 3,6 volts. Cette même tension qui se trouvera donc aux bornes de la résistance. Nous avons donc tout ce qu'il faut pour calculer la valeur de notre résistance :

U = R x I

3,6 = R x 0,012

R = 3,6 / 0,012 = 300



Voici un format très courant de résistance dit CMS, tel qu'on en trouve dans tous nos équipements. Des composants de cette taille sont difficiles à souder manuellement. Oui, c'est une pièce d'un centime d'euros.

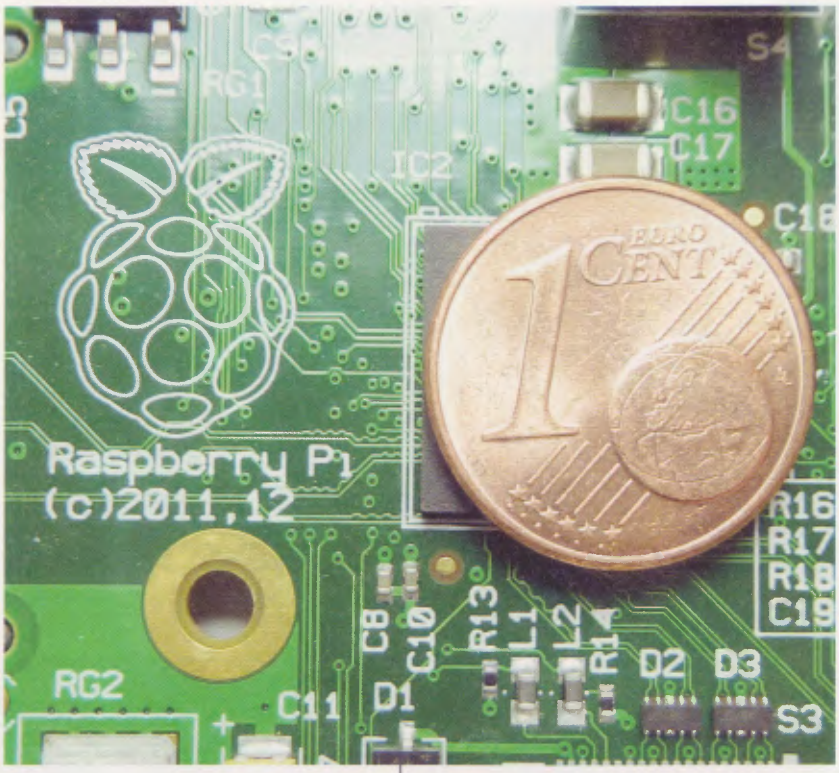
Les résistances d'une valeur de 300 ohms ne sont pas courantes, par contre, 330 ohms oui. Nous décidons donc d'utiliser une résistance un peu plus importante, ce qui aura pour conséquence de laisser passer dans le circuit moins de courant. Nous pouvons d'ailleurs le calculer également :

U = R x I

3,6 = 330 x I

3,6 / 330 = I = 0.0109

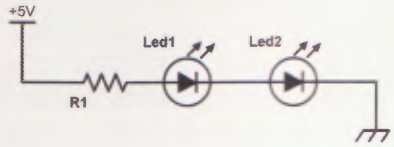
11 mA à la place de 12 mA ce n'est pas grave. Il est toujours préférable de choisir de faire passer moins de courant que plus. Les documentations des leds par exemple, parlent de « maximum absolut » (*absolute maximum rating*). Certaines valeurs ne doivent pas être dépassées et d'autres oui mais durant un laps de temps très réduit (*peak forward current*, courant en crête). Bien que les leds soient rarement des composants très sensibles, il est toujours de bon ton de lire la documentation que vous trouverez en générale facilement (mais en anglais) sur le Web en cherchant simplement la marque et le modèle.



Voici une photo macro d'une carte Raspberry Pi avec, sur le bas de l'image, une résistance à l'emplacement marquée R14 (entre L1 L2 et D2 D3). À cette échelle, il n'est plus possible de souder/dessouder ce type de composant à la main, ce sont donc des machines qui placent les composants (robot pick-and-place) et des fours qui permettent de les souder (Reflow soldering).

5. JUSTE UN PEU PLUS LOIN

Nous allons terminer avec une situation un peu plus complexe. Cette fois nous ne voulons pas connecter une led sur la sortie mais deux. Il existe bien des manières de le faire mais les trois plus courantes sont les suivantes.



Une première solution est de mettre les deux leds l'une à la suite de l'autre, en **série**. Nous savons qu'une led fait apparaître une tension de 1,4 volts, chacune. Ceci veut dire qu'entre la première broche de la première led et la seconde broche de la seconde nous aurons le double, 2,8 volts. Sachant cela, nous pouvons utiliser à nouveau la loi de Ohm pour les calculs :

U = R x I

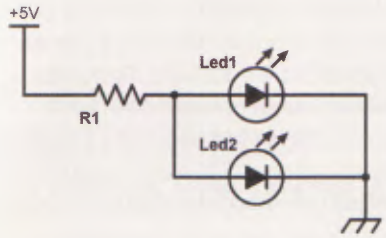
U = 5 - (1,4 x 2) = 2,2

2,2 = R x 0,012

R = 2,2 / 0,012 = 183

Là encore une résistance de 183 ohms n'est pas monnaie courante, on se rabat donc sur une valeur standard supérieure, 220 ohms. Nous aurons 2,2 volts divisé par 220 ohms, ce qui nous donne une intensité de courant de 10 mA. Il est important de relever que le courant qui circule est le même dans tout le circuit. Ce sont les mêmes 10 mA qui vont traverser la résistance, la première led et la seconde. C'est un flot de courant et c'est, je le répète, le même qui passe dans tout le circuit. Les tensions elles, selon l'endroit où on les mesure, sont différentes. Entre la broche de l'Arduino et la masse c'est 5 volts, entre le point résistance/led et la masse c'est 2,8 volts, etc.

Une autre façon de connecter les deux leds consiste à les brancher ensemble, en parallèle et de relier le tout à la masse et à la sortie de l'Arduino avec une résistance.



Mais là, les choses sont bien différentes car à l'instar d'une rivière qui se divise en deux, le courant de départ

est partagé dans les deux branches. Si nous voulons que chaque led ait une intensité de courant de 12 mA qui la traverse, ceci signifie qu'il nous faut le double avant l'embranchement. En revanche, la tension elle sera toujours la même puisque l'une et l'autre led font apparaître la même différence de potentiel. Nous avons bien 1,4 volts aux bornes des leds mais notre résistance doit maintenant laisser passer 12 mA fois deux, soit 24 mA. Ceci étant, il ne reste qu'à faire les calculs :

U = R x I

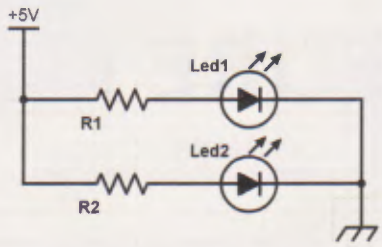
U = 5 - 1,4 = 3,6

3,6 = R x (0.012 + 0.012)

R = 3,6 / (0,024) = 150

150 est une valeur standard pour une résistance, nous pouvons simplement l'utiliser et le tour est joué. Dans la totalité du circuit nous auront 24 mA qui circuleront en tout. Mais dans chaque branche, entre le point de séparation et la masse, ne circulera que 12 mA par led.

Enfin, il existe encore une autre façon de faire, nous pouvons, en effet, utiliser une résistance par led et brancher les deux ensembles en parallèle.



Nous avons maintenant deux résistances à calculer. Chaque led fait apparaître une tension de 1,4 volts mais l'intensité du courant est pour chacune d'elle de 12 mA. Nous avons tout bonnement le premier circuit mono-led en double et de ce fait les mêmes calculs s'appliquent. Nous utiliserons donc des résistances de 330 ohms.

Peut être vous demandez-vous en quoi ce circuit est différent si ce n'est de nous faire utiliser une résistance en plus. La réponse est la puissance dissipé par les résistances. Dans le précédent circuit 24 mA passent par la résistance de 150 ohms. Celle-ci dissipe donc :

P = R x I^2

P = 150 x 0,024 x 0,024 = 0,0864

86,4 milliwatts, ce qui est encore acceptable pour une résistance standard conçu pour dissiper au maximum 1/4 de watts soit 0,250 watts ou 250 mW. Dans le dernier circuit nous avons deux résistances qui chacunes dissipent P = R x I^2 soit 330 x 0.012 x 0.012 = 0,047520 watts ou 47,52 mW. C'est nettement moins.

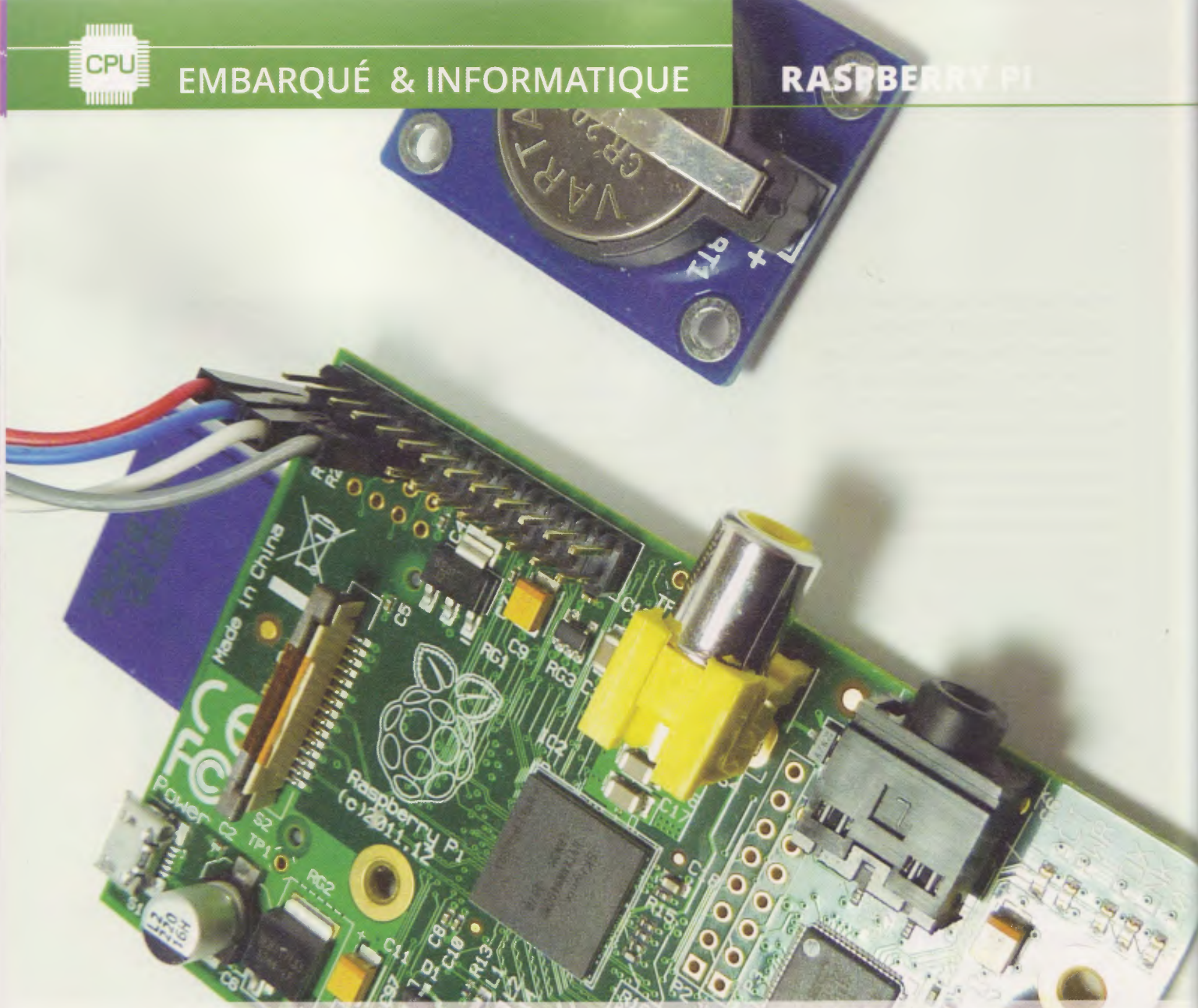
Si vous commencez à multiplier les leds, ceci va devenir un problème (en dehors du fait que l'Arduino ne peut pas fournir un tel courant par broche). Avec 10 leds en parallèle l'intensité du courant doit être de 120 mA passant par une seule et unique résistance, or la puissance dissipé sera alors 3,6*0,12*0,12 = 0,432 soit 432 mW ce qui est largement supérieur au maximum de 250 mW. La résistance va chauffer et se faisant elle va alors changer ses caractéristiques et être littéralement moins résistante. Elle va laisser passer plus de courant qui traversera également les leds. L'ensemble va chauffer et on sort des spécifications du constructeur. Les composants, résistance et leds, seront soumis à des courants et des tensions qui ne sont pas ceux attendu et ils vont se dégrader. En utilisant plusieurs résistances, on réparti la dissipation et chacune d'elle à moins de travail et reste donc dans les normes imposées. Voilà pourquoi il peut être intéressant de procéder de la sorte.

6. POUR FINIR

En parlant de jouer avec les limites, précisons qu'une résistance qui chauffe est un problème sérieux mais loin du pire. Faire dissiper 2 watts à une résistance prévue pour 250 mW et il sera question de risques d'incendie. D'autres composants peuvent même exploser et ce sans arriver dans des tensions et des courants dangereux pour l'homme. Une led connectée dans un PC à une alimentation 12 volts sans résistance, l'alimentation pouvant fournir plusieurs ampères, va dissiper d'un coup 30 watts, ce qui peut vaporiser le semi-conducteur qui est à l'intérieur, créer une pression interne impossible à contenir et faire exploser le composant de manière potentiellement violente (projections de plastique presque garanties).

Mais l'électronique c'est comme le bricolage ou le jardinage, en respectant les lois et les règles il n'y a pas de problème. Et je suis persuadé que vous n'êtes pas du genre à mettre votre pied ou votre main sous une tondeuse en marche et donc à bien faire vos calculs et vos devoirs avant de connecter des composants ou modules entre eux...

Mère nature c'est un peu comme Judge Dredd (de 95 pas le remake pathétique de 2012). Elle a établie des règles claires, si vous les respectez, vous ferez des merveilles. Si vous tentez de tricher, elle se montrera sans pitié et ne vous fera pas de cadeau. La loi c'est elle :) **DB**



VOTRE RASPBERRY PI TOUJOURS À L'HEURE !

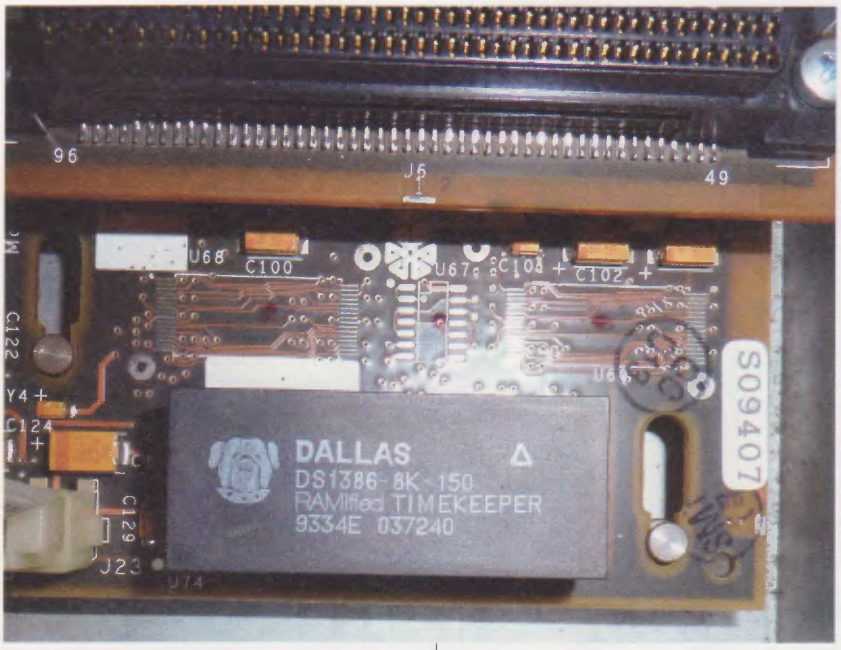
La carte Raspberry Pi, outre sa popularité sans cesse croissante, possède un certain nombre d'avantages parmi lesquelles sa simplicité de prise en main. Malheureusement elle possède également un certain nombre de limitations face à la concurrence, entre autre celle de la BeagleBone Black. Mais les deux cartes ont en commun leur manque cruel d'horloge temps réel. Voici comment régler ce petit problème pour moins de 10 euros.

Vous n'avez pas pu le rater ! Lorsque vous débranchez votre Raspberry Pi de sa source d'alimentation et la reconnectez, celle-ci à besoin du réseau pour être à l'heure. Dans le cas contraire, elle utilisera la date de construction de la distribution utilisée (07/01/2014 dans nos essais avec Raspbian). Plusieurs solutions s'offrent alors à vous comme spécifier la date vous-même ou encore reposer sur le réseau pour que le système « aille trouver comme un grand l'heure qu'il est » (utilisation d'un protocole appelé NTP). Dans les deux cas, en coupant l'alimentation, l'heure enregistrée et décomptée au fil du temps qui passe sera perdue.

Lorsqu'on a du réseau, ceci peut passer inaperçu puisque, selon la configuration, NTP peut être utilisé à chaque démarrage pour mettre l'heure à jour. Mais dans la vraie vie on a pas toujours accès à Internet pour tous les projets et il serait bien agréable que notre nano-PC dispose d'une « montre »...

1. RTC OU HORLOGE TEMPS RÉEL

Ne vous êtes-vous jamais demandé comment votre PC faisait pour connaître l'heure et faire en sorte que le système, à chaque démarrage, même après avoir été déconnecté du courant, soit toujours à l'heure ? Ceci est possible car il intègre une horloge, opérant sur batterie la plupart du temps, fonctionnant alors même que la machine est éteinte.



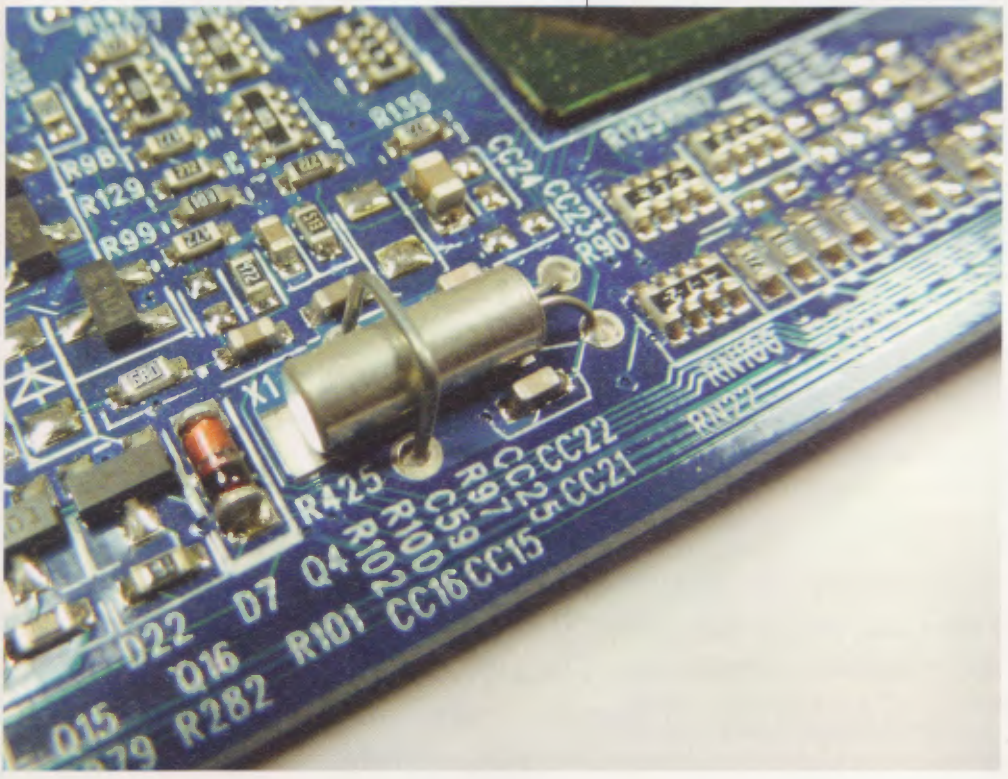
L'horloge temps réel permet à un PC de gérer le temps à l'échelle humaine. Ici, une RTC présente dans une station SGI Indy (1993-1997). De nos jours, la RTC n'est plus un composant à part dans nos ordinateurs, mais une fonctionnalité intégrée dans le chipset de la carte mère (southbridge).

Cette fonctionnalité bien agréable fait en réalité partie d'un ensemble de choses regroupé sous l'appellation « horloge temps réel » ou *real-time clock* en anglais et communément abrégé « RTC ».

Un ordinateur dispose souvent de plusieurs sources de signaux permettant de cadencer le fonctionnement de chaque composant. On parle d'horloge ou de signaux d'horloge. Un peu comme un métronome donnant le rythme, ces horloges égrainent le temps qui passe sous la forme d'impulsions électriques. La plus connue et citée de ces horloges est celle du processeur lui-même, rythmant l'exécution des instructions et donc apportant la notion de vitesse du processeur.

Les choses sont plus complexes aujourd'hui qu'elles ne l'étaient il y a encore quelques années. La vitesse croissante des processeurs a fixé la fréquence des horloges processeurs

> *« Ils [les ports] sont en 3,3V TTL. [...] Il y a des problèmes de durée de vie en 5V - vous pouvez ajouter une diode pour baisser un peu la tension jusqu'à 4,3V. 5V ne grillera pas la puce, mais cela réduira notablement sa durée de vie »*



Exemple de quartz équipant une carte mère moderne. Celui-ci marqué X1, placé à proximité de la puce southbridge (ici, une VIA VT8235) gérant l'audio, l'USB, l'Ethernet, le contrôleur IDE et surtout, intégrant une RTC.

au alentour des 4 Ghz, soit 4 milliards d'impulsions par seconde.
Tout comme les technologies de gravure des processeurs atteint les limites de la physique, la fréquence des horloges possède elle aussi ses limites imposées par la nature de notre univers. Voilà pourquoi, à l'heure actuelle, la seule manière de gagner en puissance passe par le nombre de coeurs d'un CPU et de *threads* que chacun d'eux peut exécuter. Mais le processeur n'est qu'un élément parmi d'autres qu'il faut cadencer. Un ordinateur utilise donc généralement plusieurs horloges pour plusieurs tâches et plusieurs éléments (CPU, périphérique, bus, etc).

Contrairement à une horloge processeur, une horloge temps réel n'a pas pour objectif d'aller de plus en plus vite. Le but poursuivi est l'exactitude et une parfaite correspondance avec les unités temporelles utilisées par nous humain. Une horloge temps réel ou RTC n'a pas besoin d'utiliser une fréquence importante pour ce travail. En fait, la plupart des RTC utilisent une fréquence de 32,768 KHz ou 32768 oscillations par seconde obtenue en utilisant une source très précise : un quartz (plus exactement un résonateur à quartz).

Pourquoi 32768 Hz ?
Simplement parce que cela représente 2^15 oscillations par seconde soit une valeur qu'il est possible de stocker sur 16 bits, facilement manipulable en binaire et donc idéale pour qu'un ordinateur puisse « savoir » ce qu'est une seconde.

Voici le véritable sens du mot RTC. Cependant, de manière générale, le fait que cette horloge inclue également des fonctionnalités permettant un fonctionnement prolongé sur une source d'alimentation à faible courant, à quelque peu associé le terme RTC avec la notion d'horloge autonome. En effet, sur un ordinateur personnel, même s'il ne n'agit plus aujourd'hui d'un composant spécifique soudé sur la carte mère, la RTC est reliée aux bornes d'une pile bouton (souvent de type CR2032) afin de fonctionner même si la machine n'est pas alimentée.

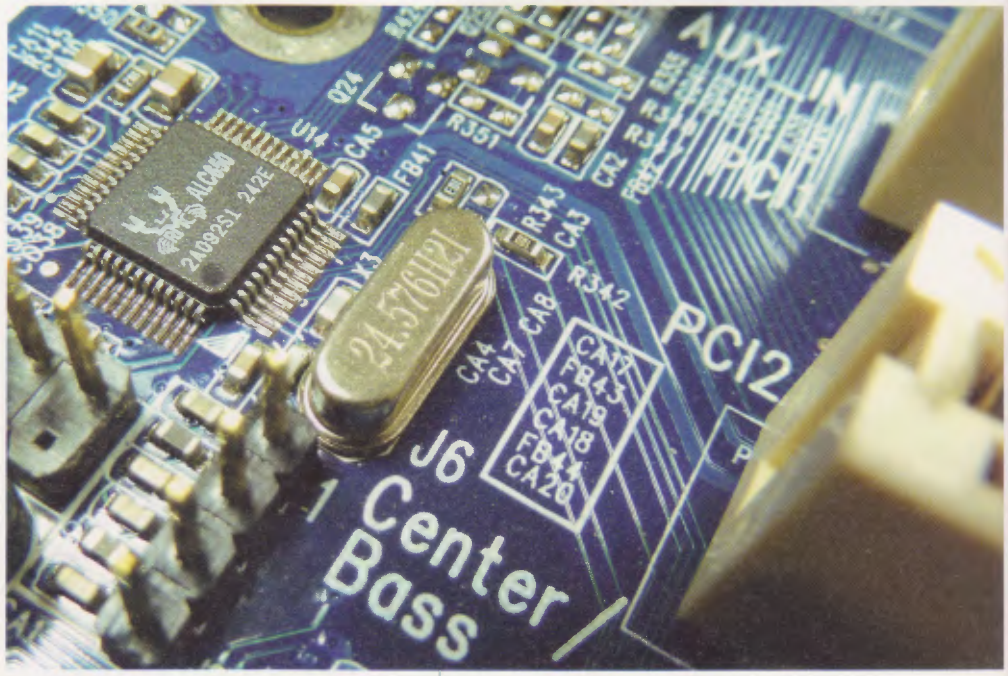
Cette pile fournie également du courant (ou du moins le faisait fut un temps) à une mémoire permettant au PC de se souvenir de sa configuration au plus bas niveau (BIOS). Voilà pourquoi il était courant de retirer la pile pour « remettre à zéro » la configuration et, par la même occasion, réinitialiser la date au « début de l'époque » soit le 1er janvier 1970.

2. UNE TENSION QUI MONTE !

Un problème récurrent dans la communauté Raspberry Pi concerne un sujet capital pour les débutants et potentiellement préoccupant : les niveaux de tension. Soyons clair, les niveaux de tension utilisés par la Raspberry Pi sont 0/3,3V et la carte n'est pas tolérante au 5V.

Vous pouvez retourner la question dans tous les sens, tenter de tricher, flirter avec les limites mais lorsqu'un constructeur précise des niveaux logiques 0/3,3 volts il faut, en principe, respecter ces spécifications. C'est exactement comme pour la documentation d'une led, un tel composant est conçu pour être alimenté avec un courant bien spécifique qui fait alors apparaître une tension elle aussi bien précise à ces bornes, le tout à une température comprise dans une certaine plage. C'est ainsi que le constructeur à conçu le composant et pour un usage dans ces conditions là.

Les GPIOs d'une carte Raspberry Pi sont prévues pour fonctionner en 0/3,3V (c'est du LVTTL pour *Low-voltage TTL*) et même s'il existe des astuces (via l'utilisation de diodes par exemple) permettant ainsi d'interfacer des composants fonctionnant avec des niveaux 0-5V, ceci n'est pas une approche convenable, quand bien même 4,3V serait dans les marges de tolérance. Au mieux il faudra utiliser des composants spécifiques permettant la conversion de manière sûre et fiable.

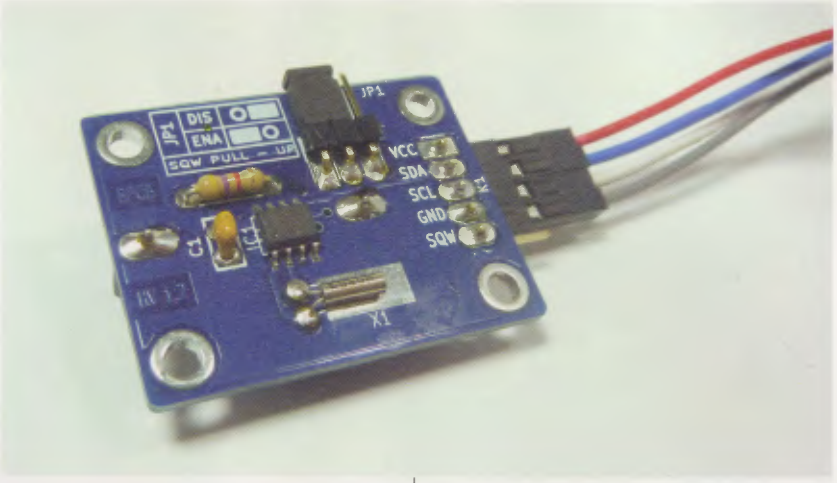


Le quartz de la RTC d'un PC n'est pas le seul d'une carte mère. Ici, le quartz marqué X3 et utilisant une fréquence de 25,576 Mhz est rattaché à une puce ALC650. Il s'agit d'un CODEC audio Realtek. Cette fréquence est typiquement utilisée pour les systèmes audio, puisque correspondant à 512 fois 48 Khz, qui est une fréquence d'échantillonnage très utilisée.

Voilà ce que précise Liz Upton (épouse de Eben Upton, directeur technique chez Broadcom et architecte de la RPi) dans les forums officiels (à défaut de documentation technique (*datasheet*) complète utilisable) :

"*They're 3V3 TTL. [...] There are lifetime issues with 5V – you might want to add a diode to drop the voltage a bit to 4V3. 5V won't fry the chip, but it will decrease its lifetime noticeably.*"

En français : « Ils [les ports] sont en 3,3V TTL. [...]Il y a des problèmes de durée de vie en 5V - vous pouvez ajouter une diode pour baisser un peu la tension jusqu'à 4,3V. 5V ne grillera pas la puce, mais cela réduira notablement sa durée de vie ».



Un module RTC comme il en existe des centaines. Ceux de bonnes factures sont facilement reconnaissables : condensateur de filtrage, résistance de rappel, marquage des broches, trous de fixation, soudures propres... On distingue au centre le composant DS1338 ainsi que, juste en dessous, le quartz cadençant la RTC.

Ainsi, clairement, le constructeur attend que vous utilisez ces niveaux de tension et aucun autre. Dans le cas contraire, vous sortez des spécifications et le comportement devient indéterminé. Peut-être que cela fonctionnera, peut-être pas. Peut-être que cela marchera un temps, puis de manière aléatoire avant de ne plus marcher du tout. Il ne s'agit pas ici de hack mais de jouer avec le feu selon des conditions qui sont totalement variables.

Donc oui, certains utilisateurs parlent de connecter un composant appelé DS1307 qui est sensé être utilisé en 5V (même avec un maximum absolut fixé à 7V) avec plus ou moins de bidouille pour que cela fonctionne (et tantôt rien d'autre qu'une connexion directe). Pour notre part, nous choisissons de respecter les règles de l'art dans le domaine de l'électronique et de l'embarqué qui consistent à suivre les spécifications. En particulier lorsque l'utilisation d'un autre composant, lui pleinement compatible, n'entraîne aucune difficulté ou coût supplémentaire (le DS1338 par exemple).

Avec le temps et l'expérience, on apprend que les spécifications, qu'il s'agisse de tension, de plage de température en fonctionnement, de temporisation, de distance de câblage ou encore de température lors de la soudure, sont des choses qu'il est généralement raisonnable de respecter. Même si c'est très amusant de faire s'allumer une led verte en rouge en poussant le courant fourni, soyez sur que ça ne dure généralement pas très longtemps :)

3. MODULE ET CONNEXION

Vous l'avez compris, nous allons ajouter un module électronique connecté à la Raspberry Pi. Ce module comportera une RTC sous la forme du composant Maxim-Dallas DS1338-33 ou, tout simplement un DS1338. Il s'agit d'une horloge temps réel basse consommation à laquelle est ajouté un quartz de 32,768 Khz, quelques autres composants passifs et surtout un support pour pile bouton CR2032. Il existe bien d'autres composants de ce type, offrant les mêmes fonctionnalités. Nous avons ici choisi ce modèle en particulier car il est compatible avec le très courant DS1307 qui ne fonctionne qu'en 5V.

Pour que notre Rasberry Pi, ou tout autre montage électronique, puisse programmer l'heure initiale et surtout lire l'heure actuelle, le composant dispose d'une interface spécifique en i2c. Sans entrer dans trop de détails, en électronique numérique et embarqué, trois interfaces d'interconnexion se partagent la vedette : SPI, CAN et i2c. Chacune dispose de caractéristiques particulières qui les rendent plus utilisées dans un domaine plutôt qu'un autre. L'i2c est très courant pour les RTC et différents capteurs. CAN est

très utilisé en technologie automobile et SPI est présent dans un peu tous les autres domaines.

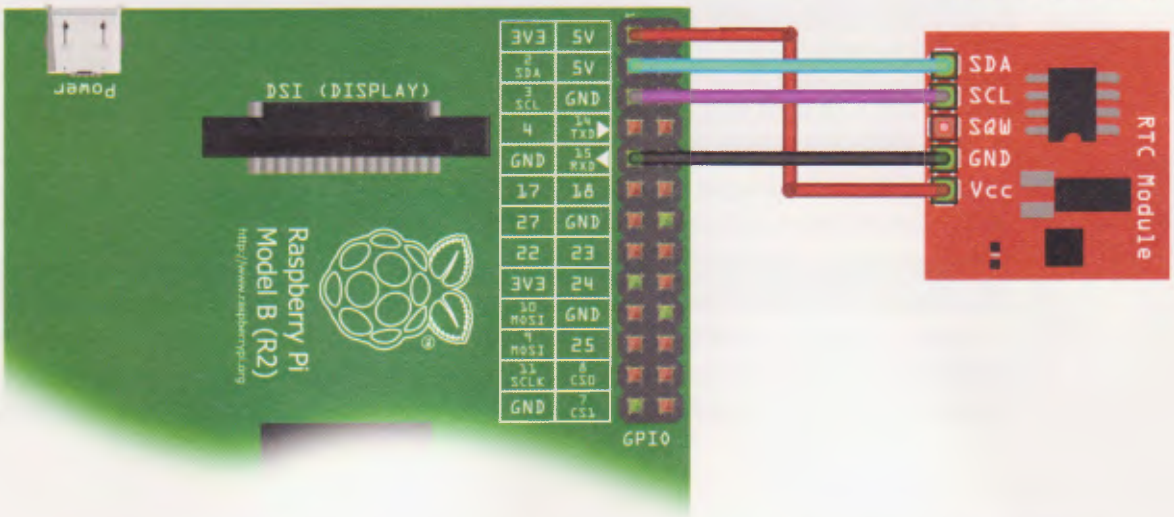
L'i2c est un bus. Ceci signifie que sur les mêmes liaisons peuvent se trouver plusieurs composants i2c, chacun ayant un numéro permettant de l'identifier. Un peu comme les numéros de maison dans une rue. Selon le composant, ce numéro est configurable ou non et ce dans un certaine mesure. L'i2c pour *Inter Integrated Circuit* est une invention de la société Philips (maintenant NXP) il y a plus de 30 ans de cela. Depuis l'i2c est devenu une norme et bien des constructeurs l'utilisent à la fois pour leurs composants et leurs systèmes, tantôt sous le nom TWI (*Two Wires Interface* - interface à deux fils).

Sur un bus i2c, il peut y avoir plusieurs maîtres et plusieurs esclaves mais les échanges se font toujours à l'initiative du maître. Dans le cas qui nous occupe, le maître sera la Raspberry Pi et le seul esclave (pour l'instant) sera le DS1338. Parmi le peu de connecteurs à notre disposition sur le nano-ordinateur nous disposons, en effet, d'une interface i2c (et une SPI aussi). Une telle interface n'utilise que peu de broches :

- SDA pour les données,
- SCL pour le signal d'horloge (rien à voir avec l'horloge temps réel, il s'agit ici du signal qui cadence la communication),
- une alimentation (ici 3,3V),
- et la masse (qui doit impérativement être commune à tous les composants du bus).

Mais trêve de théorie ! Notre module provient d'un vendeur eBay (cesar_dias) et a été acheté en deux exemplaires pour moins de 10 euros. L'enveloppe est arrivée rapidement (du Portugal) sous la forme de deux petits sachets anti-statiques contenant une feuille explicative et deux connecteurs (un droit et un coudé) à souder sur le module. Des modules équivalents sont vendus par Sparkfun (~11€) et d'autres boutiques en ligne dans le même ordre de prix, si vous n'êtes pas très à l'aise ou rassurer par les sites d'enchères en ligne. Une seule remarque négative concernant cet achat : les modules étaient livrés équipés de piles CR2032 à la date de péremption inconnue et surtout placées dans le même sachet que les connecteurs à souder, pouvant donc potentiellement entraîner des courts-circuits. Les deux modules achetés cependant n'ont semble-t-il pas souffert durant le transport.

La connexion du module RTC à la Raspberry Pi ne demande pas de précaution particulière.



La connexion entre la Raspberry Pi et un de ces modules est relativement simple puisque qu'il suffit de connecter :

- broche 1 de la RPi (3,3V) à Vcc sur le module,
- broche 3 (SDA) sur SDA du module, c'est la GPIO 2 en temps normal,
- broche 5 (SCL) sur SCL, la GPIO 3,
- et enfin broche 9 (Masse/GND) sur la masse (GND) du module.

La broche SQW, permettant par exemple la sortie d'un signal carré généré par le DS1338, n'est pas connecté. Pour tout dire, cette sortie, présente sur de nombreuses RTC n'est que rarement utilisée dans des applications basiques comme l'intégration d'une RTC dans un montage ou sur une carte comme la RPi.

4. ESSAI ET VÉRIFICATION

Une fois la connexion établie, le système ne va pas faire le reste du travail à votre place. Nous allons, dans un premier temps, nous assurer que tout fonctionne. Par défaut, du moins avec la configuration telle qu'installée par Raspbian Debian Wheezy du 7 janvier 2014, les broches qui nous intéressent ne sont pas configurées pour servir en tant que bus i2c mais comme simples entrées/sorties. Pour changer cela nous devons commencer par demander au système de charger un module noyau :

```
$ sudo modprobe i2c-bcm2708
```

Nous pourrons alors voir apparaître dans les messages système que celui-ci a bien pris en compte notre ordre :

```
$ dmesg | tail
[ 4112.398106] bcm2708_i2c_init_pinmode(0,0)
[ 4112.398139] bcm2708_i2c_init_pinmode(0,1)
[ 4112.401936] bcm2708_i2c bcm2708_i2c.0: BSC0
   Controller at 0x20205000 (irq 79) (baudrate 100000)
[ 4112.402089] bcm2708_i2c_init_pinmode(1,2)
[ 4112.402108] bcm2708_i2c_init_pinmode(1,3)
[ 4112.402366] bcm2708_i2c bcm2708_i2c.1: BSC1
   Controller at 0x20804000 (irq 79) (baudrate 100000)
```

Nous avons ici, à présent, deux bus i2c disponibles. L'un correspond aux broches qui nous intéressent, l'autre est interne et plus difficilement accessible. En fonction de la version de la Respberry Pi en votre possession, le bus accessible n'est pas le même. Pour les premières cartes c'était le bus 0, pour les révisions 2 (marquées 2011.12), c'est le 1. Remarquez dans la sortie que le mode de fonctionnement des entrée/sorties concernées est changé (**pinmode**).

Pour pouvoir vérifier sur le système que tout ceci fonctionne, nous allons utiliser un outil appelé **i2cdetect** qu'il nous faudra préalablement installer avec :

```
$ sudo apt-get install i2c-tools
```

A ce stade, normalement, le noyau du système à pris en charge les bus i2c mais nous n'y avons pas pour autant accès. Linux dispose d'une fonctionnalité spécifique pour cela, appelée **i2cdev**. Dans les systèmes d'exploitation actuels (par opposition aux vieilleries comme MS/DOS)

il y a une différenciation entre ce que le noyau peut faire et ce que les programmes (même lancés par le super-utilisateur `root`) ont le droit d'utiliser. Le module que nous avons chargé ajoute des fonctionnalités au noyau et possède des privilèges particuliers. Afin qu'un programme comme `i2cdetect` puisse accéder aux bus, et donc au matériel, un autre module doit être chargé. Son travail est de servir de lien entre la gestion du matériel en espace noyau et ce qu'on appel l'espace utilisateur. Avec un système comme GNU/Linux, ce lien est généralement fait par l'intermédiaire du contenu de `/proc`, `/sys` et/ou `/dev`. Chargeons donc le module :

```
$ sudo modprobe i2c-dev

$ ls /dev/i2c-*
/dev/i2c-0  /dev/i2c-1
```

Comme vous le voyez, les bus 0 et 1 ont prit la forme de pseudo-fichiers dans `/dev`. Nous pouvons maintenant utiliser notre outil :

```
$ i2cdetect -l
i2c-0   unknown  bcm2708_i2c.0  N/A
i2c-1   unknown  bcm2708_i2c.1  N/A
```

L'option `-l` nous permet de lister les bus et effectivement, nous en voyons deux. En fonction de votre modèle de Raspberry Pi, il vous suffira d'utiliser `i2cdetect` suivi du numéro de bus que vous voulez scruter :

```
$ sudo i2cdetect 1
WARNING! This program can confuse your I2C bus,
   cause data loss and worse!
I will probe file /dev/i2c-1.
I will probe address range 0x03-0x77.
Continue? [Y/n]
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

La commande, après nous avoir prévenu d'éventuels problèmes, nous demande de confirmer l'opération. Il n'existe pas, en effet, de manière sûr, de scanner un bus i2c. La solution utilisée par `i2cdetect` est tout simplement de tenter d'accéder à chaque numéro sur le bus pour voir si quelque chose répond. Dans le cas présent ce n'est pas un problème mais il peut arriver que cette interrogation brutale soit mal interprétée par les composants du bus qui peuvent alors réagir comme s'il s'agissait d'un ordre qui leur est transmis. Il n'y a pas d'équivalent à un `ping` sur un bus i2c, ni de « y'a quelqu'un ? », il faut entamer un dialogue et voir si ça répond.

Ici, `i2cdetect` fait toutes les adresses de 0 à 77 et un seul esclave répond à l'adresse 68, c'est notre DS1338. Celui-ci est donc parfaitement vu par le système et nous pouvons passer à la suite.

A ce stade notre RTC est accessible mais nous n'avons pas d'élément logiciel permettant de le contrôler. Il nous faut un pilote. Celui-ci existe dans le système et s'appel `rtc_ds1307`. Ne soyez

pas étonné du nom utilisé, il n'est pas rare que différents matériels très similaires soient pris en charge par un unique pilote. C'est le cas du DS1338, piloté par le même module permettant de gérer le DS1307 (et bien d'autres RTC).

Nous pouvons donc charger le module :

```
$ sudo modprobe rtc_ds1307
```

Et il ne se passe strictement rien ! Nous l'avons dit plus haut, il existe des composants i2c pouvant prendre l'adresse qu'ils choisissent (généralement via des mises à la masse ou à la tension d'alimentation de certaines broches) et scanner un bus peut poser des problèmes. Nous avons donc bien un pilote chargé mais celui-ci ne connait pas l'adresse du composant qu'il doit gérer. Nous pouvons l'aider en le précisant manuellement :

```
$ sudo bash -c 'echo "ds1307 0x68" > /sys/class/i2c-adapter/
i2c-1/new_device'
```

Ne soyez pas effrayés par la commande, c'est juste une astuce à connaître. Nous pourrions tout aussi bien faire :

```
$ sudo -s
# echo "ds1307 0x68" > /sys/class/i2c-adapter/i2c-1/new_device
# exit
$
```

L'utilisation de `sudo bash -c` permet de lancer des commandes complexes utilisant des redirections en tant que super-utilisateur. La commande `echo` est utilisée pour afficher un texte mais nous redirigeons ce texte vers `/sys/class/i2c-adapter/i2c-1/new_device`. Le fichier `new_device` n'est pas destiné à être lu mais uniquement à recevoir une ligne composée d'un nom de pilote et d'une adresse sur le bus i2c. Remarquez également que le chemin utilisé concerne `/sys` et que `i2c-1` concerne le bus 1 (`i2c-0` dans l'autre cas).

Notre Raspberry Pi connectée au module RTC alimenté par une pile bouton de type CR2032, suffisante pour faire fonctionner l'horloge durant des mois sinon des années et ainsi tenir l'heure à jour même la Raspberry Pi éteinte.

Nous venons donc d'associer l'adresse `0x68` au pilote `rtc-ds1307` et le résultat est immédiat. Ainsi en affichant les messages système :

```
$ dmesg | tail
[ 9842.022166] rtc-ds1307 1-0068: rtc core: registered ds1307 as rtc0
[ 9842.022216] rtc-ds1307 1-0068: 56 bytes nvram
[ 9842.022276] i2c i2c-1: new_device: Instantiated device ds1307 at 0x68
```

Notre DS1337 est pris en charge et est maintenant utilisé par le système pour servir de première horloge temps réel sous la forme du fichier `/dev/rtc0`. Mais ceci ne signifie pas pour autant que le système va s'en servir. Pour lire et écrire l'heure dans la RTC, un outil spécifique existe, c'est `hwclock` pour *HardWare Clock*. Il n'est en rien spécifique au DS1338 ou compatible mais se place un cran au dessus et utilise `/dev/rtc0`. Ainsi, ce même outil fonctionnera sur tous types de systèmes, le noyau se chargeant de faire le lien avec le matériel effectivement utilisé en guise de RTC.

Nous pouvons nous en servir pour enregistrer une heure dans la RTC :

```
$ sudo hwclock --set --date="05/02/2014 10:38:00"
```

Notez que la date est spécifiée au format américain sous la forme mois/jour/année. Quelques secondes plus tard, nous pouvons lire la RTC et nous obtenons :

```
$ sudo hwclock -r
ven. 02 mai 2014 10:38:14 CEST -0.866652 seconds
```

L'heure à bien été enregistrée et quelques secondes se sont bien écoulées entre l'enregistrement initial et le moment présent. Tout fonctionne à merveille. Deux autres options sont très utilisées avec `hwclock` :

- `--systohc` ou `-w` copiant l'heure système vers la RTC,
- `--hctosys` ou `-s` procédant à l'opération inverse, RTC vers heure système.

Il nous faut maintenant configurer le système afin qu'il procède à ces deux opérations au bon moment et ce de manière automatique.

5. CONFIGURATION DÉFINITIVE

Les tutoriels, billets de blog et autres messages dans les forums apportent généralement des explications et détaillent une démarche basique basée sur les deux options que nous venons de voir. Malheureusement, les auteurs ont généralement une connaissance assez superficielle du fonctionnement d'un système GNU/Linux et font généralement l'impasse sur des mécanismes déjà en place pour faciliter l'intégration du support d'une RTC.

Voici ce qui est généralement conseillé :

- Ajouter `i2c-bcm2708` et `rtc-ds1307` dans `/etc/modules` afin que les deux modules soient chargés au démarrage,
- modifier `/etc/rc.local` pour ajouter la commande `echo` vu précédemment et déclarer le DS1338 sur le bus,
- également ajouter dans ce fichier la commande `hwclock -s` pour régler la date système en fonction de celle inscrite dans la RTC.

Dans les grandes lignes, ça marche, mais c'est complètement brouillon, voir « bricolo ». Mieux vaut respecter la structure du système et en particulier les mécanismes déjà en place. Ainsi, il faut savoir qu'une fois le module `i2c-bcm2708` chargé, un service particulier est prêt à réagir à l'apparition d'un nouveau composant sur le bus. Ce service s'appel `udev` et il chargera automatiquement le module `rtc-ds1307` dès que l'on déclarera le composant sur le bus. Il n'est donc pas nécessaire de charger le module.

Mais `udev` fait beaucoup plus encore. Celui-ci opère selon des règles définies dans le système et éventuellement par l'utilisateur (dans `/etc/udev/rules.d`). Ainsi, ce service permet, entre autres choses, d'exécuter une commande lors de l'apparition d'un périphérique et c'est précisément le cas pour `/dev/rtc0` via le lancement de `/lib/udev/hwclock-set`.

Comme vous pouvez le voir, un certain nombre de choses est donc déjà en place pour notre ajout matériel. Il n'est pas utile de lancer nous même `hwclock -s`, ni même de charger le module `rtc-ds1307`. En revanche, dans l'installation par défaut de Raspbian, un certain nombre de choses est à retirer. Ainsi, un paquet `fake-hwclock` est installé par défaut, permettant de palier à l'absence de RTC et enregistrant régulièrement l'heure dans le système pour éviter un retour brutal en 1970 en cas d'indisponibilité du réseau pour la récupération de l'heure via NTP. Comme nous disposons à présent d'une RTC, ce paquet n'est plus utile et nous pouvons donc le retirer avec :

```
$ sudo apt-get remove --purge fake-hwclock
```

En résumé, tout ce que nous avons à faire c'est dire au système de :

- charger le module permettant le support i2c,
- déclarer notre DS1338 sur le bus,
- et, à l'apparition de la RTC, récupérer l'heure et mettre celle du système à jour.

La configuration tient ainsi à peu de chose. Nous commençons donc par modifier `/etc/modules` pour ajouter une ligne contenant `i2c-bcm2708`. Ceci aura pour effet de charger automatiquement le module au démarrage du système.

La seconde étape consiste à nous pencher sur le fichier `/etc/rc.local`. Là encore, ce contenu est exécuté au démarrage (un peu plus tard), c'est là que nous pouvons ajouter notre déclaration. Une version basique de cette modification pourrait être de simplement ajouter la ligne suivante au fichier (juste avant `exit 0`) :

```
echo "ds1307 0x68" > /sys/class/i2c-adapter/i2c-1/new_device
```

Une version plus complète et plus sûre est la suivante :

```
I2CDIR=/sys/class/i2c-adapter/i2c-1
if [ -d "$I2CDIR" ]; then
  echo "ds1307 0x68" > $I2CDIR/new_device
fi
```

Le chemin est stocké dans une variable `I2CDIR` et nous l'utilisons pour tester si un répertoire de ce nom existe. S'il n'existe pas, c'est que le pilote pour le support i2c ne s'est pas chargé ou qu'un problème a été rencontré. Si le répertoire existe, en revanche, nous pouvons utiliser notre commande pour déclarer le composant sur le bus.

Au moment du démarrage, ceci aura pour effet de réveiller `udev` qui va alors automatiquement charger le module correspondant et donc faire apparaître `/dev/rtc0` qui dès lors sera utilisable par le système.

Mais ce n'est pas tout. La commande utilisée par `udev` lors de l'apparition de la RTC est prévue, à la base, pour d'autres systèmes, qui prennent en charge la RTC bien plus tôt dans le processus de démarrage. Ainsi, la commande utilisée par `udev` est `/lib/udev/hwclock-set`. Il s'agit d'un script que nous devons modifier pour qu'il copie l'heure de la RTC dans le système. Celui-ci, par défaut ne fait qu'ajuster l'heure système dans le noyau en fonction du fuseau horaire car, sur PC par exemple, celle-ci est déjà copier depuis la RTC à ce moment.

Nous devons donc éditer le fichier `/lib/udev/hwclock-set` de :

```
if [ yes = "$BADYEAR" ] ; then
    /sbin/hwclock --rtc=$dev --systz --badyear
else
    /sbin/hwclock --rtc=$dev --systz
fi
```

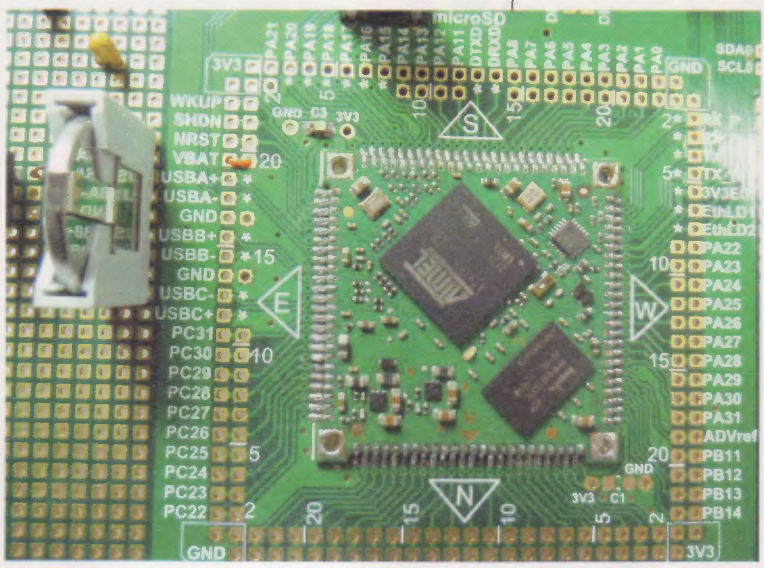
en :

```
if [ yes = "$BADYEAR" ] ; then
    /sbin/hwclock --rtc=$dev -s --badyear
    /sbin/hwclock --rtc=$dev --systz --badyear
else
    /sbin/hwclock --rtc=$dev -s
    /sbin/hwclock --rtc=$dev --systz
fi
```

Nous ajoutons simplement les lignes permettant la copie de l'heure depuis la RTC. Dernière modification, nous devons préciser à l'ensemble du système que nous disposons d'une RTC matérielle. Editons le fichier `/etc/default/hwclock` et précisons `HWCLOCKACCESS=yes`.

La dernière étape de nos manipulations consiste à définir l'heure système. En effet, lors du prochain redémarrage ou arrêt du système, l'heure système sera automatiquement copiée dans la RTC par l'intermédiaire du script `/etc/init.d/hwclock.sh`, il est donc impératif qu'elle soit exacte.

A ce stade, vous pouvez également désinstaller le paquet `ntp` et installer `ntpdate` qui est un simple client permettant de récupérer l'heure ponctuellement depuis un serveur officiel et de mettre à jour l'heure système. Si vous avez du réseau avec votre Raspberry Pi à ce moment, vous pouvez utiliser la commande `ntpdate-debian` juste avant de redémarrer.



Un certain nombre de cartes n'ont pas besoin d'une RTC additionnelle, car celle-ci est déjà présente dans la puce principale (le SoC). Ici, un module AriaG25 d'ACME Systems avec, à gauche, l'emplacement pour la pile CR2032.

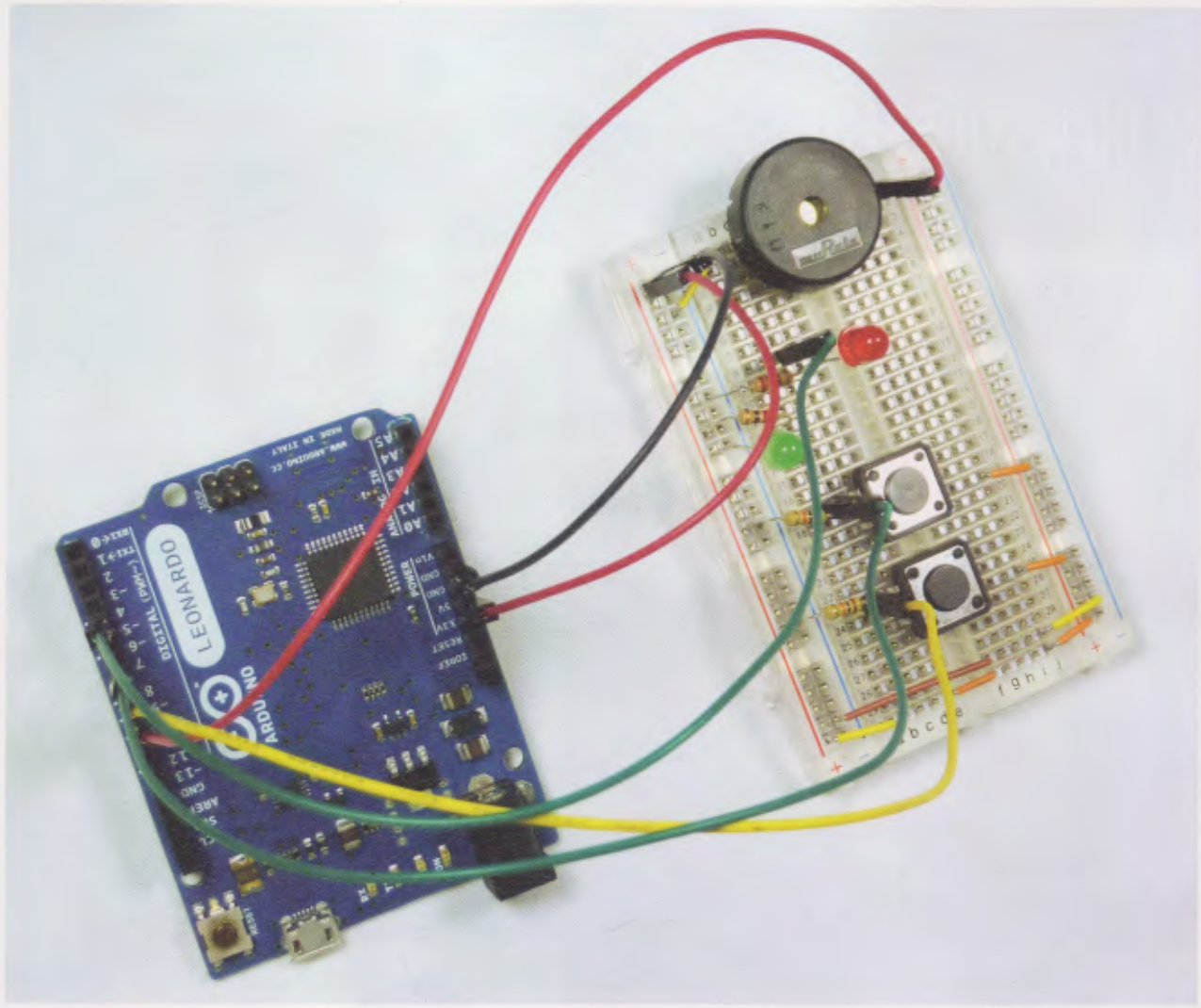
6. POUR CONCLURE

Nous venons de le voir, palier aux manques fonctionnelles de la Raspberry Pi n'est pas quelque chose de très difficile dans le cas de la RTC. Pour une poignée d'euros vous pouvez obtenir une carte bien plus autonome qui pourra alors être utilisée pour des projets sans connectivité Internet. DB

L'ARDU-SONNETTE INTELLIGENTE : EST-ON PASSÉ EN VOTRE ABSENCE ?

Quelqu'un devait passer chez vous, mais vous avez dû sortir faire une course. À votre retour, une question vous taraude l'esprit : quelqu'un est-il passé quand je n'étais pas là ? Cette personne repassera-t-elle plus tard ? Vais-je avoir l'air d'un désespéré si j'appelle pour poser la question ?! Ah si seulement vous aviez votre Alfred à vous, ou mieux encore, un adorable Jarvis dévoué... Attendez voir ! Mais vous n'avez qu'à le fabriquer !

NIVEAU	TEMPS	BUDGET
	20 minutes	environ 26 € (Arduino inclus)



Il y a des choses qui n'existent qu'au cinéma. C'est le cas des milliardaires volant dans des armures avec pour compagnon une intelligence artificielle qui comprend absolument tout ce qu'on lui dit. Dans la vraie vie, les choses sont plus terre-à-terre mais aussi, en réfléchissant un peu, bien plus simples. Que diriez-vous de fabriquer cette sonnette intelligente, capable de vous dire si on a sonné à votre porte en votre absence et même le nombre de fois où cela est arrivé ? Le tout avec un Arduino, quelques composants, deux leds et deux boutons...

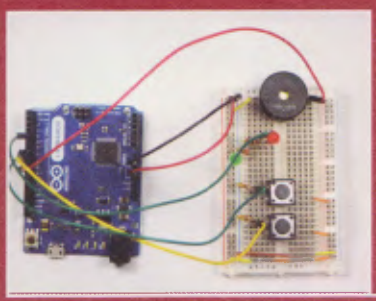
LE PRINCIPE

Notre sonnette intelligente fonctionnera de la manière suivante : à chaque fois que quelqu'un sonne à votre porte, l'Arduino l'enregistre en mémoire et vous le signale par une led qui s'en trouve allumée. Comme notre sonnette doit être intelligente, nous allons faire en sorte que plusieurs coups répétés sur le bouton poussoir ne compte pas pour autant de personnes ayant souhaité vous rendre visite. Si quelqu'un sonne, un délai configurable devra s'écouler entre deux coups de sonnette pour que notre montage considère qu'il s'agit d'une nouvelle visite. Ainsi, qu'il s'agisse de deux tentatives ou plus de vous « réveiller », ou d'une personne insistant longuement pour que vous lui ouvriez, il ne sera comptabilisé qu'une seule visite. De retour dans votre nid douillet, une simple pression sur un bouton et la led témoin clignotera autant de fois que vous aurez eu de visiteurs en votre absence.

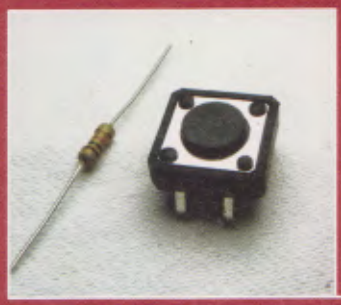
CE QU'IL VOUS FAUT

- une carte Arduino : peu importe laquelle, ce que nous allons lui demander dans un premier temps est à la portée de tous les modèles ;

- un bouton poussoir pour la sonnette, ainsi qu'une résistance de 10 KOhms ;



- une platine à essai (*breadboard*) et des cordons de connexion ;



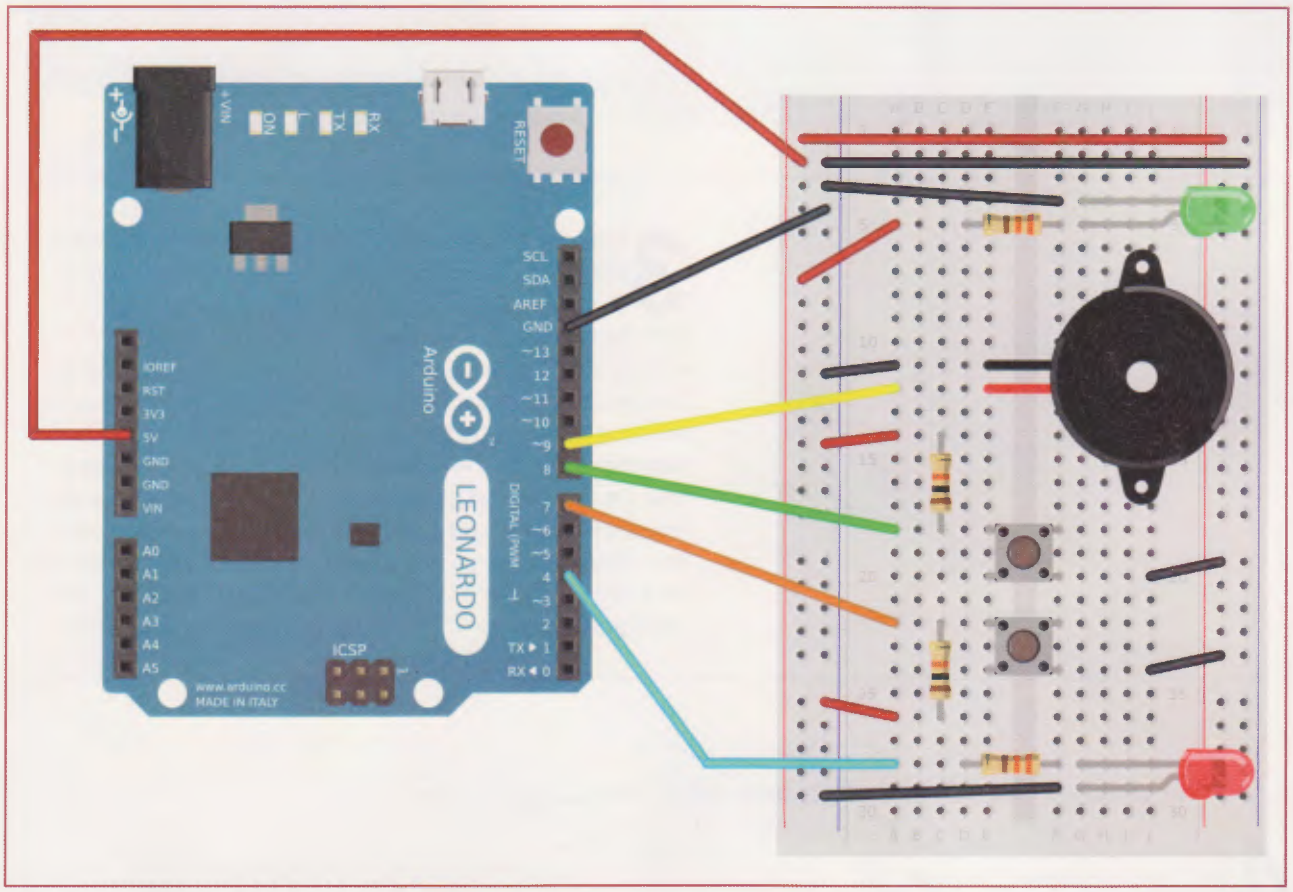
- un autre bouton poussoir et une résistance de 10 KOhms pour la gestion et l'interface ;

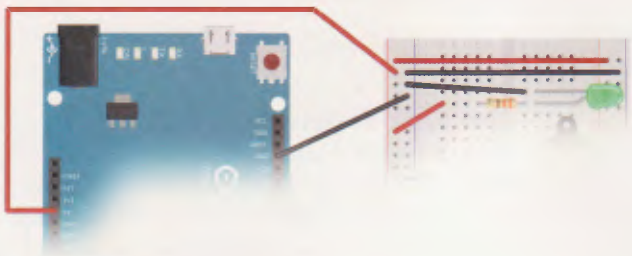
- une led et sa résistance de 330 Ohms pour communiquer avec nous : vous pouvez utiliser la led incluse à l'Arduino ;

- un buzzer qui fera office de signalisation (pour quand vous êtes présent).

- une led et sa résistance de 330 Ohms en témoin d'alimentation ;

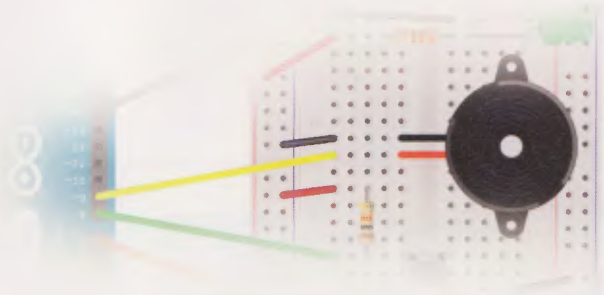
LE MONTAGE

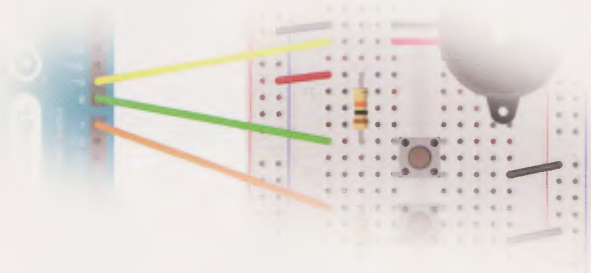




1 Sur la platine, nous commençons par relier l'alimentation depuis la boche 5V de l'Arduino, ainsi que la masse. Ainsi, nous pourrons alimenter tous les composants sur la platine à essai. Nous ajoutons une led, ainsi connectée à cette alimentation par l'intermédiaire d'une résistance de 330 Ohms ou 470 Ohms. Ceci nous permettra de voir que le montage est sous tension.

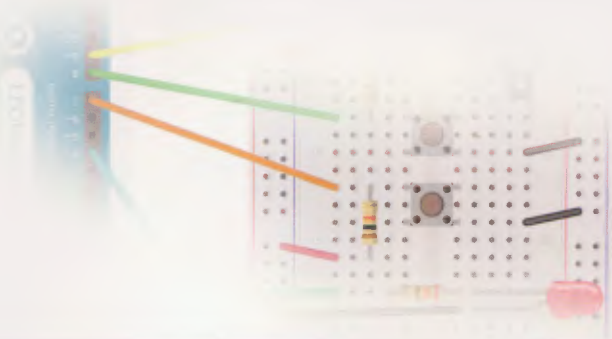
2 Nous connectons ensuite le buzzer piézoélectrique sur la broche 9 de l'Arduino et à la masse. Il faut que cette sortie puisse être utilisée de manière « analogique ». Le buzzer est comme un petit haut-parleur ; de lui-même, il ne génère pas de bruit, mais doit obtenir un signal qu'il transformera en son.





3 Nous passons ensuite à la connexion du premier bouton à la broche 8 qui sera une entrée. C'est notre bouton de commande. Notez l'utilisation de la résistance de 10 KOhms. Dans ce montage, la résistance est connectée entre la masse et l'entrée de l'Arduino qui lira donc par défaut l'entrée comme étant au niveau bas (0, la masse). En revanche, lorsqu'on appuie sur le bouton, une liaison est faite entre l'alimentation (Vcc +5V) et l'entrée, mettant cette dernière à l'état haut. Ce montage est généralement appelé une résistance de rappel à la masse. Il est également possible d'assembler cela dans l'ordre inverse, de manière à obtenir une résistance de rappel à Vcc (la tension d'alimentation), mais ceci présente le désavantage de consommer du courant (très peu) en permanence.

4 Le second bouton correspond à celui de la sonnette. Il est relié à la broche 7 avec un montage identique au précédent. Lorsqu'un visiteur appuiera, l'Arduino saura qu'il doit sonner et gérer l'événement.



5 Enfin, une led rouge est connectée à la broche 4 par l'intermédiaire d'une résistance de 330 Ohms ou 470 Ohms. C'est notre témoin lumineux indiquant qu'il s'est passé quelque chose et que notre intervention est nécessaire.

LE CROQUIS ARDUINO

Comme vous pouvez le voir, le croquis est relativement concis et n'utilise pas de bibliothèques particulières.

Fichier Édition Croquis Outils Aide

```
int led = 4;
int buzzer = 9;
int btsonnette = 8;
int btok = 7;

int active = 0;
unsigned long delai = 300000;
unsigned long previousMillis = 0;

void sonne() {
  tone(buzzer,411);
  delay(150);
  tone(buzzer,611);
  delay(150);
  noTone(buzzer);
}

void setup() {
  pinMode(led, OUTPUT);
  pinMode(btsonnette, INPUT);
  pinMode(btok, INPUT);
}

void loop() {
  if(digitalRead(btsonnette)) {
    unsigned long currentMillis = millis();
    if(((unsigned long)(currentMillis - previousMillis) >= delai) | !active) {
      active++;
    }
    previousMillis = currentMillis;
    sonne();
  }
  if(digitalRead(btok)) {
    digitalWrite(led,LOW);
    delay(1000);
    for(int i=0; i < active; i++) {
      digitalWrite(led,HIGH);
      delay(350);
      digitalWrite(led,LOW);
      delay(350);
    }
    active=0;
  }
  if(active)
    digitalWrite(led,HIGH);
}
```

Arduino

À PROPOS DU CROQUIS

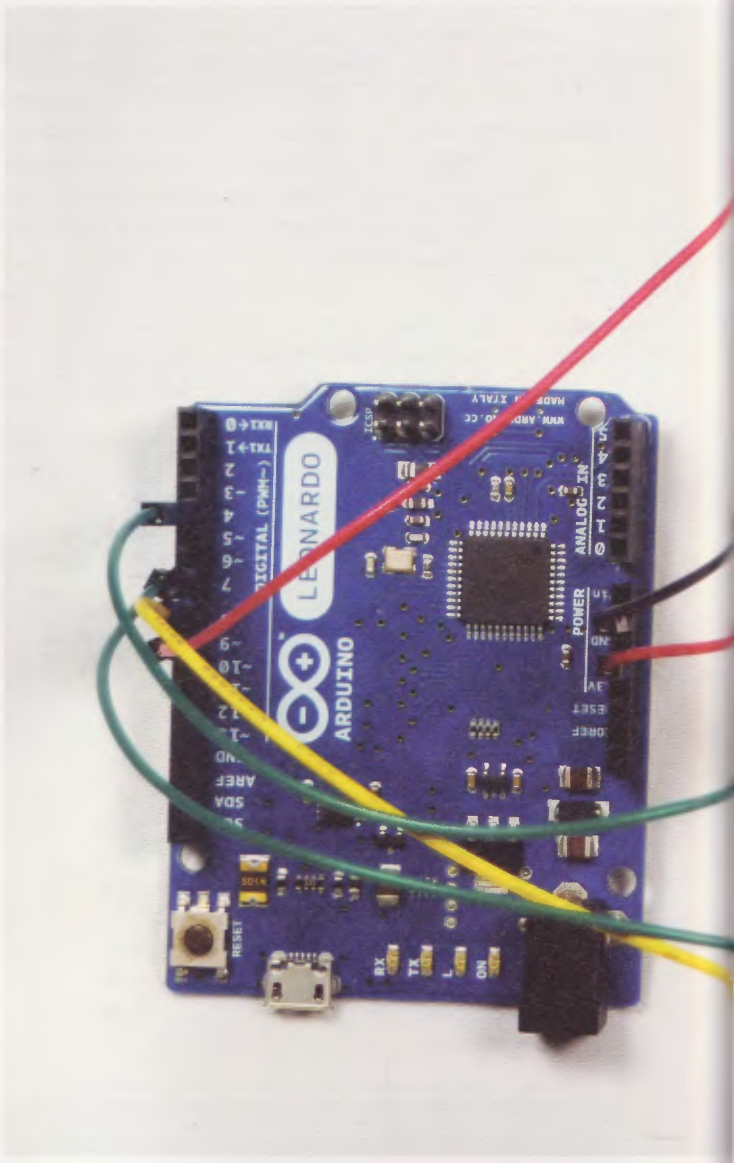
Notre croquis n'utilise que des fonctions standards livrées avec votre environnement Arduino. L'architecture générale consiste à gérer trois types de situations :

- La pression sur le bouton sonnette désigné par `btsonnette` comme « Bouton Sonnette »,
- La pression sur le bouton de contrôle et de remise à zéro que nous avons appelé `btok`,
- Le comportement à adopter si quelqu'un a utilisé, au moins une fois, la sonnette.

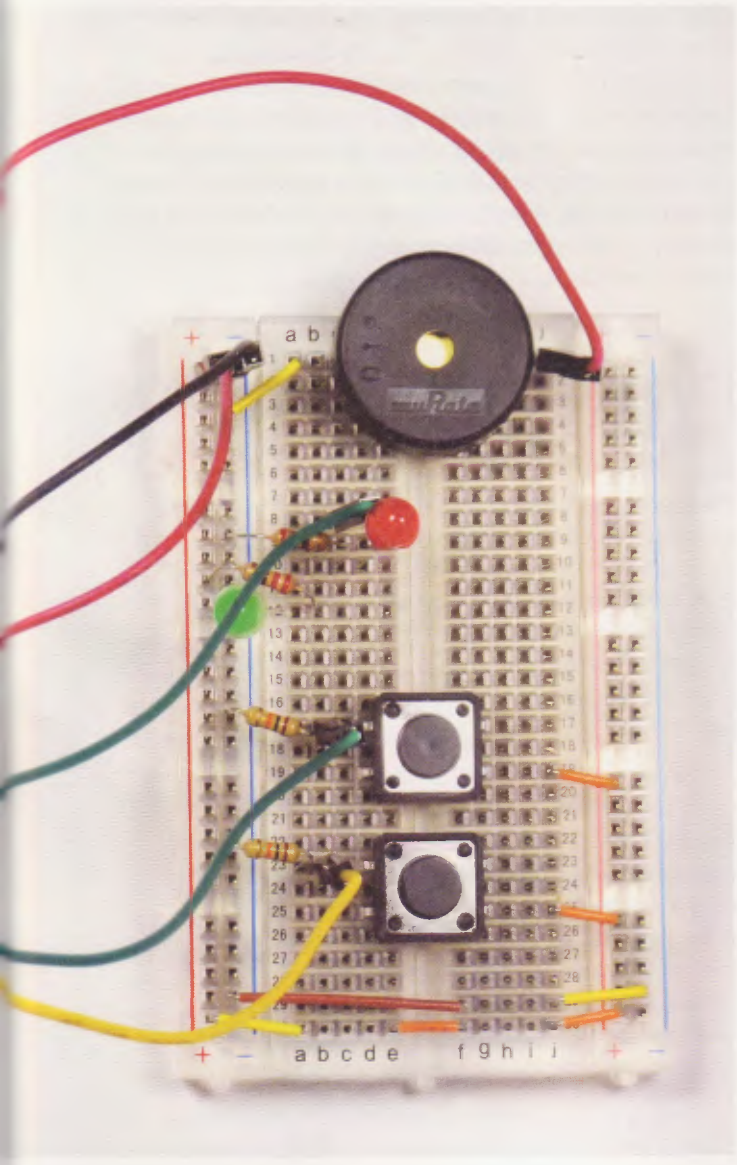
Commençons par le plus simple, à savoir ce que nous allons faire dans cette dernière situation. Cela consiste simplement à passer la sortie où est connectée la led de notification à l'état haut. Celle-ci s'en trouve alimentée et s'allume donc pour nous signaler que nous avons quelque chose à faire. Cette situation est provoquée par le fait qu'une valeur supérieure à zéro se trouve dans `active`.

Voyons maintenant comment `active` change d'état. Cela arrive lorsque le test sur l'entrée correspondant au bouton sonnette retourne une valeur positive et donc, lorsque quelqu'un appuie sur le bouton. Comme nous ne voulons pas que chaque coup de sonnette compte comme une visite, nous nous assurons de ne toucher à `active` que si une certaine durée s'est écoulée (cf. plus bas). Si c'est le cas, nous ajoutons 1 dans `active`, en d'autres termes, nous incrémentons la variable (`active++` est équivalent à `active=active+1` mais bien plus court à écrire). Nous n'oublions pas également de faire produire un son à notre Arduino en appelant la fonction `sonne()`. Nous avons préféré ici sortir le code correspondant sous la forme d'une fonction, afin de plus facilement modifier cette génération de son (cf. plus bas).

Enfin, nous avons le cas d'une pression sur le bouton de contrôle. La situation est la suivante : vous revenez chez vous et la led de notification est allumée indiquant le passage d'un visiteur. Vous appuyez sur le bouton, l'Arduino va éteindre la led, puis la faire clignoter autant de fois que vous aurez eu de visites, c'est-à-dire autant de fois que la valeur présente dans `active`. On part ici du principe que l'utilisateur est attentif et ne regardera pas ailleurs durant le décompte. En effet, dès que les clignotements cessent, `active` est mis à zéro, prêt pour attendre un premier coup de sonnette.



Ceci est la version minimaliste d'une interface homme-machine (ou IHM). On notera que si la sonnette intelligente est mise en place de façon constante, il faudra également appuyer sur le bouton de contrôle avant de quitter les lieux pour remettre le compteur de visites à zéro (si nous avons eu des visites en étant présent). Ce n'est pas cher payé et nous économisons un bouton supplémentaire pour activer/désactiver le mode « surveillance ».



GESTION DU TEMPS

Un sympathique petit problème se pose en écrivant le croquis : comment faire en sorte que plusieurs coups de sonnette, même espacés de plusieurs dizaines de secondes, ne comptent que comme une seule visite ? La solution réside dans la gestion d'un délai durant lequel un autre coup de sonnette provoquera bien la génération d'un son, mais ne changera pas la valeur de `active`.

Les délais dans le langage de l'Arduino sont généralement obtenus avec `delay(x)` où `x` est une valeur en millisecondes (un millième de seconde). Cependant, lorsqu'on utilise ceci, l'Arduino ne va rien faire si ce n'est attendre le temps qu'on lui aura indiqué. Ceci bloque donc le déroulement du programme. Dans notre cas, ce n'est pas utilisable, car une autre pression sur le bouton de la sonnette DOIT faire réagir l'Arduino.

La solution consiste à utiliser `milli()` qui, lorsqu'il est appelé, renvoie le temps écoulé depuis le démarrage de l'Arduino en millisecondes. À chaque appel, nous obtenons une valeur différente en fonction du temps qui passe. Un rapide calcul nous indique que la valeur retournée, qui est sur 32 bits, peut être au maximum de 4294967296 millisecondes (2 puissance 16), soit un peu moins de 49 jours. Que se passe-t-il à ce moment-là ? Simple, la valeur revient à zéro, puis continue d'augmenter. En anglais on parle de *rollover*.

Contrairement à ce qu'on peut penser, dans bien des cas, ceci n'est pas un problème... À partir du moment où on travaille bien avec des valeurs non signées, qui ne peuvent pas être négatives.

Revoici notre morceau de croquis. Nous commençons par définir une variable `delai` contenant le délai minimum entre deux coups de sonnette à 30000 millisecondes, soit 5 minutes.

Puis, nous utilisons une autre variable que nous appelons `previousMillis`, la valeur précédente de `millis()`, que nous initialisons à zéro :

```
unsigned long delai = 300000;
unsigned long previousMillis = 0;
```

Plus loin, dans `loop()`, voilà ce que nous faisons si quelqu'un sonne :

```
if(digitalRead(btsonnette)) {
  unsigned long currentMillis = millis();
  if(((unsigned long)(currentMillis - previousMillis) >= delai) | !active) {
    active++;
  }
  previousMillis = currentMillis;
  sonne();
}
```

Nous commençons par stocker la valeur de `millis()` dans une autre variable appelée `currentMillis`. C'est l'instant présent.

S'ensuit une condition qui va augmenter `active` soit :

- si `active` est actuellement à zéro et donc personne n'a encore sonné,
- si la différence entre l'instant présent et le précédent moment stocké est plus grand que notre délai minimum.

Notez l'utilisation de `(unsigned long)` avant le calcul. Ceci permet de forcer le résultat à ne pas être signé et donc, de ne jamais obtenir de valeur négative. Même si `currentMillis` est plus petit que `previousMillis`, nous obtenons un positif qui peut être comparé à `delai` et contient une différence que nous pouvons utiliser. En fonction de cette comparaison, nous augmentons `active` et, dans tous les cas, nous nous souvenons du moment du dernier coup de sonnette qui n'est autre que celui que nous venons de gérer, `previousMillis = currentMillis`. Ah oui, nous appelons ensuite la fonction qui permet de faire sonner l'Arduino !

Le résultat est exactement celui attendu. Même si une personne reste plantée devant notre porte une bonne heure ou plus et sonne à des intervalles de moins de 5 minutes, nous ne compterons qu'une visite (visite qui selon moi est très certainement à éviter dans ce cas précis). Nous pouvons ajuster ce délai en changeant simplement la valeur de `delai` et donc adapter le montage aux comportements de nos visiteurs.

Peut-être vous demandez-vous pourquoi donc nous testons la valeur dans `active`, et si tel est le cas, c'est très bien. En l'absence de ce test, qui a donc des conséquences équivalentes au dépassement du délai inter-sonnerie, nous obtiendrions un comportement non souhaité. En effet, à la mise sous tension de l'Arduino, le compteur derrière `millis()` se met en marche et notre comparaison se fera entre cette valeur et le contenu de `previousMillis`. Or, au démarrage, `previousMillis` vaut zéro et le calcul, tant que 5 minutes après mise en marche ne se sont pas écoulées, nous donne une valeur inférieure à `delai`. Ceci signifie donc que, après mise en marche, si quelqu'un sonne, avant que 5 minutes ne soient passées, nous ne compterions pas la visite, ce qui serait regrettable...

JOUER DE LA MUSIQUE

Une autre particularité de ce montage est d'utiliser les fonctions `tone()` et `noTone()`, afin de respectivement générer un son et le couper. `tone()` s'utilise en spécifiant tout d'abord la sortie utilisée (ici, `buzzer` est la sortie 9 de l'Arduino) et une fréquence en hertz (Hz). Cette fréquence correspond à un nombre d'oscillations par seconde, ce qui n'est absolument pas spécifique au son mais est un principe général de physique. Il s'agit du nombre de répétitions d'un phénomène par seconde.

Dans le cas de l'Arduino, on parle de signal carré, car la seule chose que savent faire la plupart des Arduino est de mettre une sortie à l'état haut ou bas. En changeant cet état plusieurs fois en un laps de temps donné, on observe donc une suite haut, bas, haut, bas, haut, bas, etc. La durée à l'état haut et à l'état bas est identique, on parle de rapport cyclique de 50% (moitié/moitié). Si, pendant une seconde, nous avons une demi-seconde à l'état haut, puis une demi-seconde à l'état bas, nous avons une fréquence de 1 Hertz.

Ce qu'il y a de magnifique dans ce phénomène, c'est ce qui se passe si on utilise un tel signal pour

piloter un haut-parleur ou un buzzer. La fréquence est alors utilisée pour faire vibrer soit une membrane en carton (haut-parleur), soit une rondelle métallique (buzzer). Ceci provoque un déplacement dans l'air qui est alors successivement comprimé et décomprimé. Cette succession compression/décompression est appelée un son. L'oreille humaine est capable de détecter ces variations à des fréquences allant de 20 Hz à 20 kHz (20000 Hz).

Tout ce que nous avons donc à faire pour produire un son est de générer une fréquence dans cette plage. C'est très exactement ce que fait `tone()`. Dans notre croquis, nous avons choisi de ne pas intégrer les lignes utilisant `tone()` directement dans le corps de `loop()`, afin d'éventuellement permettre de créer plusieurs sonneries différentes. Notre `sonne()` se présente ainsi :

```
void sonne() {
  tone(buzzer,411);
  delay(150);
  tone(buzzer,611);
  delay(150);
  noTone(buzzer);
}
```

Nous avons une génération de 411 Hz pendant 150 millisecondes, puis de 611 Hz pendant la même durée avant, finalement, de couper le son. Comme la fonction est utilisée à chaque fois qu'une pression sur le bouton de la sonnette est détectée, elle est appelée de manière répétée tant que dure la pression.

Il est possible de composer toutes sortes de successions de fréquences, et donc de notes, de cette manière. Il faut cependant garder à l'esprit que si la mélodie est longue, elle durera bien plus longtemps que la pression sur le bouton. C'est généralement le même comportement que l'on constate avec les carillons à multiples mélodies vendus dans le commerce. DB

CE QUE NOUS AVONS APPRIS

Cette réalisation bien qu'assez basique comprend une collection non négligeable de principes de base pour la prise en main d'une carte Arduino.

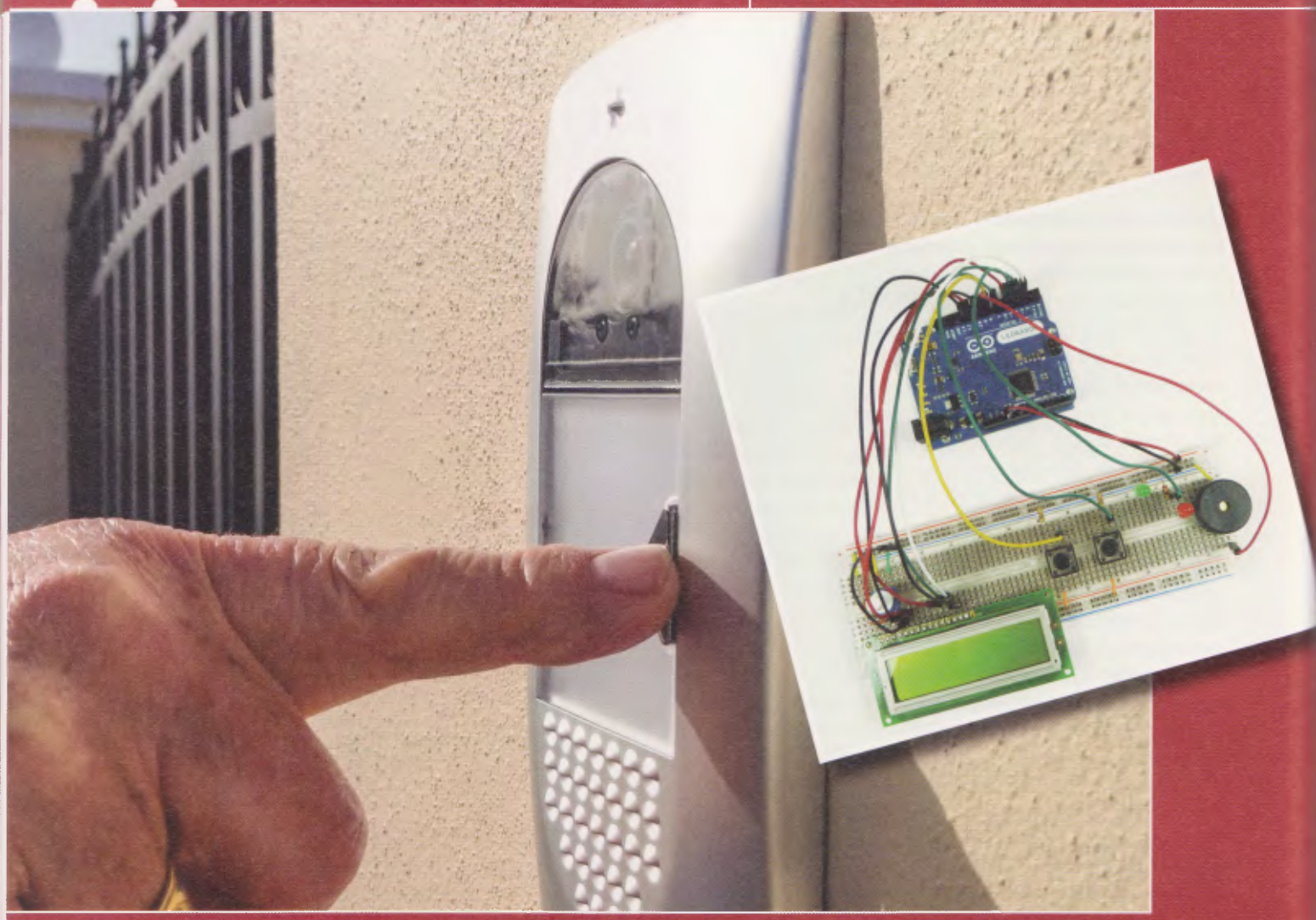
Nous avons ainsi utilisé la sortie analogique de la carte sous la forme d'une fonction permettant de moduler une fréquence et d'obtenir un son via la connexion d'un buzzer piézoélectrique.

Nous avons également utilisé les broches en entrée digitale afin de détecter la pression sur les boutons et avons ainsi fait connaissance avec le principe électronique de base des résistances de rappel. Le B.A.BA de L'Arduino qui consiste à utiliser les sorties digitales a également été couvert via la mise en oeuvre d'un simple témoin lumineux.

Mais l'élément le plus important de cette réalisation est sans conteste l'utilisation des fonctions Arduino permettant la gestion du temps. Ceci aussi bien avec la fonction basique delay() mais surtout avec la notion de Millis qui nous a permit d'appréhender un concept fondamental de l'écriture de croquis. Cette gestion des temporisations est capitale afin de pouvoir structurer vos croquis de façon à ce qu'ils ne restent pas « coincé » à attendre un événement.

Nous avons également aborder brièvement l'utilité de créer ses fonctions de manière à rendre notre code plus modulaire et lui permettre d'évoluer plus facilement en fonction de vos envies et besoins.

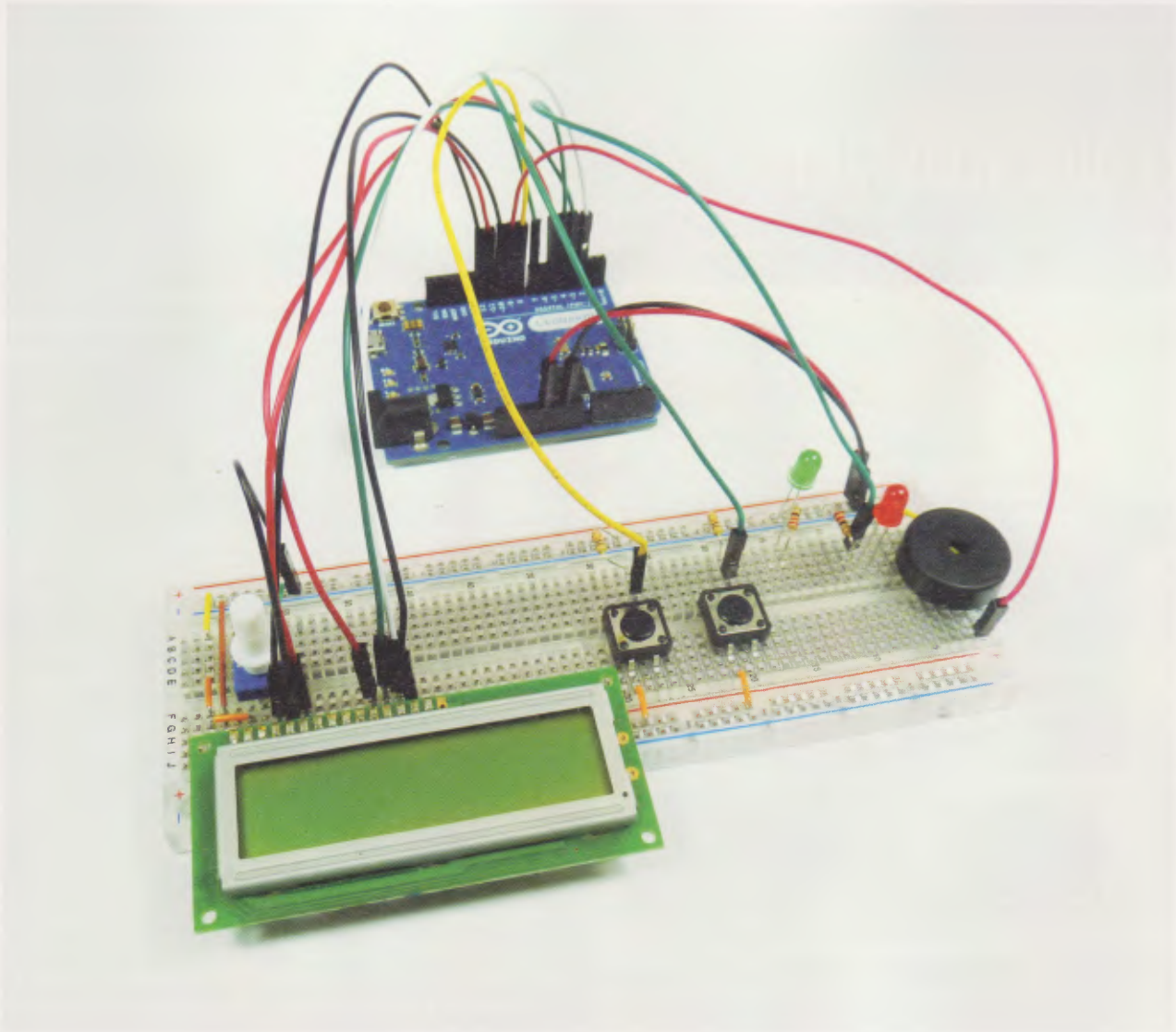
Enfin, d'un point de vue plus générique, ce montage exprime clairement le fait qu'il n'est pas nécessaire de mettre en oeuvre une masse importante de composants et que le point fondamentale reste l'imagination. Si l'idée est originale, le croquis et le montage ne sont pas nécessairement complexes. Soyez simplement imaginatif dans vos réalisations !



L'ARDU-SONNETTE INTELLIGENTE : AJOUTONS UN ÉCRAN !

Notre sonnette intelligente fait déjà un travail tout à fait digne d'un produit commercial. Nous pouvons cependant encore l'améliorer, en particulier en fournissant à l'utilisateur une interface un peu plus étoffée qu'une simple led qui clignote. Que diriez-vous d'adjoindre à notre montage un écran LCD ?

NIVEAU	TEMPS	BUDGET
	10 minutes	environ 10 €



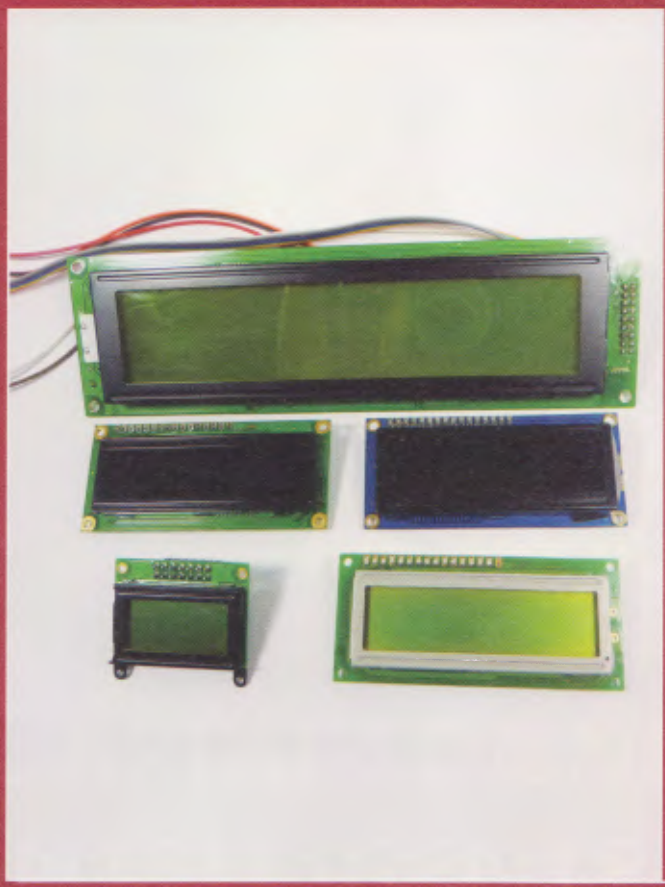
Prenons la peine de relever un fait intéressant : il ne semble pas exister de sonnette de ce type sur le marché. Après une rapide recherche, force est de constater qu'aucun constructeur n'a apparemment pensé à équiper une sonnette rudimentaire d'un petit microcontrôleur comme celui qui équipe les Arduino et d'un témoin lumineux. Avec un peu plus de recherches, il serait même possible d'envisager n'avoir qu'un seul bouton (les industriels aiment économiser sur les parties mécaniques, généralement plus coûteuses et moins fiables que des composants électroniques). Quoi qu'il en soit, le concept est désormais largement établi, nous pouvons nous attacher à le compléter et surtout le rendre plus attrayant.

LE PRINCIPE

Pour l'heure, notre sonnette s'exprime assez peu. En lui ajoutant un écran LCD capable d'afficher du texte, nous pouvons nous dispenser de l'indicateur lumineux donnant le décompte de visites et donc limiter son usage. Notre led de notification reste en place, mais ne sert plus qu'à attirer l'attention si au moins une visite a eu lieu. Le reste des informations sera affiché directement sur l'écran.

CE QU'IL VOUS FAUT

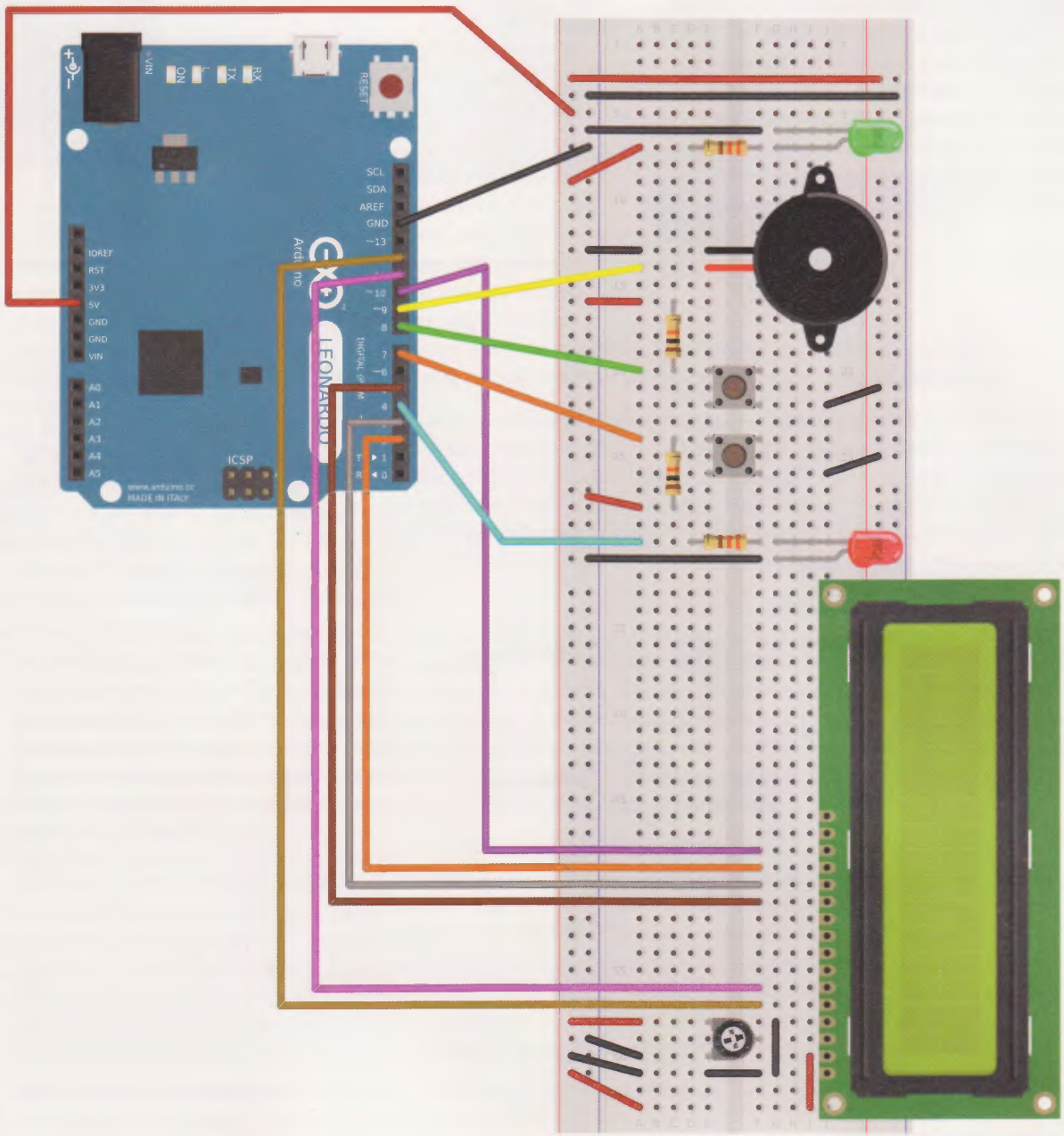
Nous ne retirerons aucun élément du projet et ne ferons qu'en ajouter :



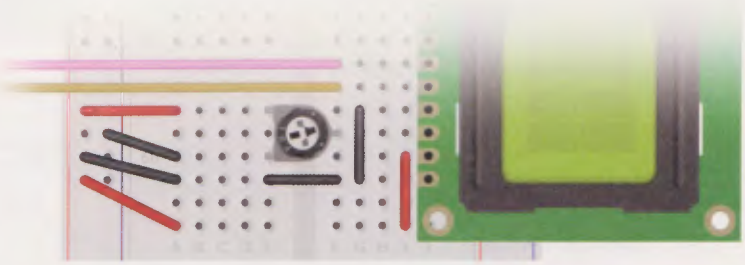
- Un **afficheur LCD alphanumérique** de 2 fois 16 lignes. Il existe bien des modèles d'afficheurs. Certains sont à cristaux liquides ou LCD (*Liquid Crystal Display*), parfois rétro-éclairés, d'autres sont OLED, bien plus lisibles et n'ayant pas besoin de rétro-éclairage ; il en existe même sous forme d'afficheurs fluorescents, ou VFD en anglais pour *Vacuum Fluorescent Display* qui étaient, jusqu'à l'arrivée des OLED, la seule solution pour obtenir un affichage lumineux vraiment contrasté et surtout capable de fonctionner à des températures négatives (la plupart des afficheurs LCD économiques se comportent mal dans ce type de situation). Le modèle le plus souvent utilisé est dit « compatible HD44780 », en référence au composant initial d'Hitachi permettant de gérer l'affichage des caractères sur l'écran (qui est en réalité composé de pixels). De nombreux clones de ce contrôleur existent, comme le KS0070B de Samsung ou le SPLC780A1 de SunPlus, mais ils fonctionnent tous de la même manière et surtout, avec la même connectique. Précisons enfin qu'il existe bien des déclinaisons en fonction de la taille, de la couleur et/ou du nombre de lignes et du nombre de caractères par ligne : 2×8, 1×16, 2×16, 4×20, 4×40, etc. Pour les grands afficheurs, plusieurs contrôleurs compatibles HD44780 sont utilisés, gérant chacun une partie de l'écran.

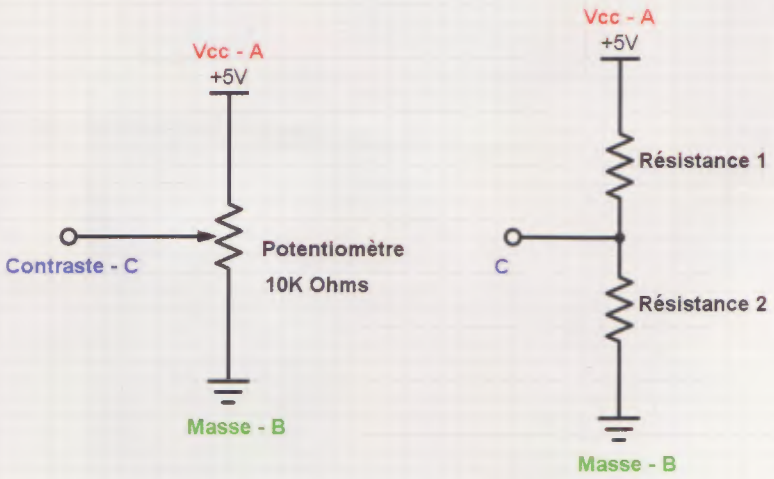
- Un **potentiomètre rotatif**, également appelé **résistance variable** de 10 KOhms. C'est un composant très courant qui peut prendre plusieurs formes, mais ce modèle est le plus courant et donc celui généralement présent dans les kits de découverte Arduino.

LE MONTAGE


VIN
ON
L
TX
RX
RESET
SCL
SDA
AREF
GND
~13
IOREF
RST
3V3
5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5
ICSP
Arduino
LEONARDO
MADE IN ITALY

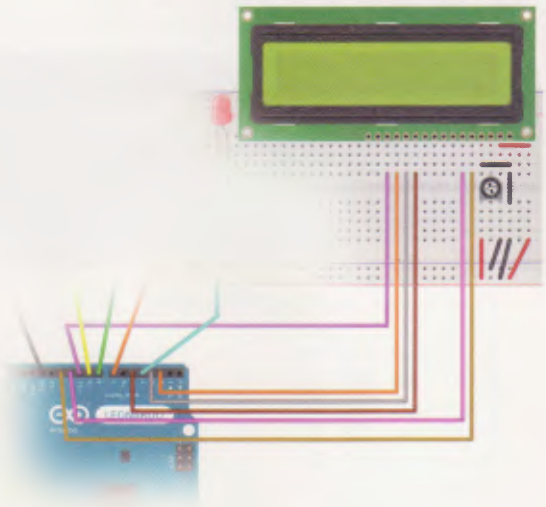
1 Nous connectons tout d'abord l'afficheur à la masse (broche 1) et l'alimentation +5V (broche 2). La troisième broche de l'afficheur reçoit une tension comprise entre 0V (la masse) et +5V et l'utilise pour déterminer le contraste. Comme en fonction de la marque et du modèle de l'afficheur ce réglage peut varier, tout comme en fonction de vos préférences, nous ne pouvons pas utiliser une paire de résistances pour définir de manière fixe le contraste. On utilise alors un potentiomètre ou résistance variable avec une broche sur la masse, la broche opposée sur la tension d'alimentation et la troisième sur la broche 3 de l'afficheur LCD.





2 Un potentiomètre, comme l'illustre le schéma ci-contre (à gauche) peut être vu comme une résistance d'une valeur fixe entre deux broches (ici A et B), mais qui dispose d'un élément mobile qui peut être déplacé vers l'une ou l'autre extrémité (C). Si l'on place la partie mobile au milieu, la résistance entre C et A est identique à celle entre B et C, et la somme des deux est égale à la résistance entre A et B. Ce type de montage, tout comme s'il était fait avec deux résistances (à droite), est appelé **un diviseur de tension**, car de la même manière, en appliquant une tension entre A et B, on constate que la somme des tensions entre A et C, et C et B est égale à celle entre A et B. En réglant le potentiomètre, on peut ainsi faire varier la tension entre A et C, et C et B, de zéro à la tension présente entre A et B. Dans notre montage, ceci permet de régler la tension entre la broche 3 de l'afficheur et la masse. Cette tension est utilisée pour déterminer le contraste. Notez que dans le cas d'un afficheur OLED ou VFD, il n'y a pas de réglage de contraste et ce montage est alors inutile.

3 Un afficheur LCD compatible HD44780 peut être contrôlé avec un minimum de 6 connecteurs, et jusqu'à 11 connecteurs. Le brochage est le suivant :



- 4 : *RS* (*Register Select*) : permet de choisir si on souhaite envoyer des commandes (0) ou des données (1) à l'afficheur. Une commande est, par exemple, l'ordre de placer le curseur à un certain endroit, ou encore celui d'effacer l'écran. Par opposition, des données sont des caractères à afficher. C'est l'Arduino qui pilotera cette broche afin de configurer l'afficheur et lui faire afficher vos messages. Ici, nous relions *RS* à la broche 12.
- 5 : *RW* (*ReadWrite*) : offre un choix entre écrire des données (0) et lire des données. L'afficheur dispose de mémoires qui sont utilisées pour stocker les caractères affichés, mais aussi pour personnaliser 8 symboles/lettres qui ne sont pas déjà gérées. Il est possible de lire ou d'écrire dans ces mémoires. Ici, comme nous faisons une utilisation simple de l'afficheur, nous n'avons pas besoin d'y lire quoi que ce soit, juste d'envoyer des données. Nous n'avons donc pas l'usage de *RW* que nous connectons simplement à la masse, de manière à ce que l'afficheur sache que nous souhaitons toujours écrire.
- 6 : *EN* (*Enable*) : c'est la validation des informations qui sont présentées sur les autres broches. Le principe de fonctionnement consiste à définir l'état des autres lignes et, quand tout est prêt, à valider l'opération. C'est un peu la touche [Entrée] de l'afficheur compatible HD44780. Cette broche de validation est très utile si vous pilotez plusieurs afficheurs. Vous pouvez en effet utiliser les mêmes connecteurs côté Arduino pour l'ensemble des autres broches et avoir un *EN* par afficheur. Ainsi, l'afficheur que vous « validez » est celui qui obéira aux ordres émis. Comme précédemment, c'est notre Arduino qui validera les données lorsqu'elles seront prêtes pour l'envoi. *EN* est connecté à la broche 11 de l'Arduino.
- 7 - 14 : *D0-D7* (*Data*) : pour envoyer des données (ou en lire), on utilisera ces broches pour gérer soit 8 bits d'un coup, soit 4 bits puis 4 autres bits. Il existe deux modes de communication : 8 bits ou 4 bits. L'avantage d'utiliser de concert les broches 7 à 14 est le fait d'envoyer les données plus vite, octet par octet, mais on utilise alors 8 broches. L'autre solution est d'utiliser le mode 4 bits qui économise la moitié des broches, mais oblige à envoyer nos données en deux fois, par tranches de 4 bits. Dans ce mode, on ne connecte pas les broches 7 à 10 de l'afficheur et n'utilisons que les broches 11 à 14, qui seront connectées respectivement aux broches 5, 3, 2 et 10 de l'Arduino. Ne vous inquiétez pas, l'Arduino s'occupe de toute la configuration !
- 15 - 16 : anode (+) et cathode (masse) pour le rétro-éclairage. Certains afficheurs disposent d'une ou plusieurs leds placées derrière ou sur la tranche de l'écran pour illuminer l'affichage. La manière de connecter l'alimentation de cet éclairage dépend de l'afficheur, mais le plus souvent, la broche 15 est le plus et la 16 la masse. La façon d'alimenter cette partie dépend également du modèle d'afficheur. Certains utilisent directement +5V ; d'autres nécessitent une résistance pour limiter le courant. D'autres encore demandent une tension plus importante, car il s'agit d'un rétro-éclairage électroluminescent, plus homogène, mais qui fonctionne de manière très différente des leds. Là, vous n'avez pas le choix, il faudra consulter la documentation du constructeur. Bien entendu, le plus simple et le plus économique est encore d'utiliser un afficheur réflectif, sans rétro-éclairage et donc sans broches 15/16.

Tout ce que nous avons à faire à présent, c'est donc de choisir des broches de l'Arduino qui seront connectées à *RS*, *EN*, *D4*, *D5*, *D6* et *D7*, et surtout nous en souvenir...

LE CROQUIS ARDUINO

Fichier Édition Croquis Outils Aide

```
#include <LiquidCrystal.h>

#define RS 12
#define EN 11
#define D4 5
#define D5 3
#define D6 2
#define D7 10

int led = 4;
int buzzer = 9;
int btsonnette = 8;
int btok = 7;
int active = 0;
unsigned long delai = 300000;
unsigned long previousMillis = 0;

LiquidCrystal lcd(RS,EN,D4,D5,D6,D7);

void setup() {
  pinMode(led, OUTPUT);
  pinMode(btsonnette, INPUT);
  pinMode(btok, INPUT);
  lcd.begin(16,2);
  lcd.print("En attente...");
  lcd.setCursor(0,1);
  lcd.print("Aucune visite");
}

void sonne() {
  tone(buzzer,411);
  delay(150);
  tone(buzzer,611);
  delay(150);
  noTone(buzzer);
}

void loop() {
  if(digitalRead(btsonnette)) {
    unsigned long currentMillis = millis();
    if(((unsigned long)(currentMillis - previousMillis) >= delai) | !active) {
      active++;
      lcd.noDisplay();
      lcd.clear();
      lcd.print("En attente...");
```

```
      lcd.setCursor(0,1);
      lcd.print(active,DEC);
      lcd.print(" visite");
      if(active>1)
        lcd.print("s");
      lcd.display();
    }
    previousMillis = currentMillis;
    sonne();
  }
  if(digitalRead(btok)) {
    digitalWrite(led,LOW);
    lcd.noDisplay();
    lcd.clear();
    lcd.print("En attente...");
    lcd.setCursor(0,1);
    lcd.print("Aucune visite");
    lcd.display();
    active=0;
  }
  if(active)
```

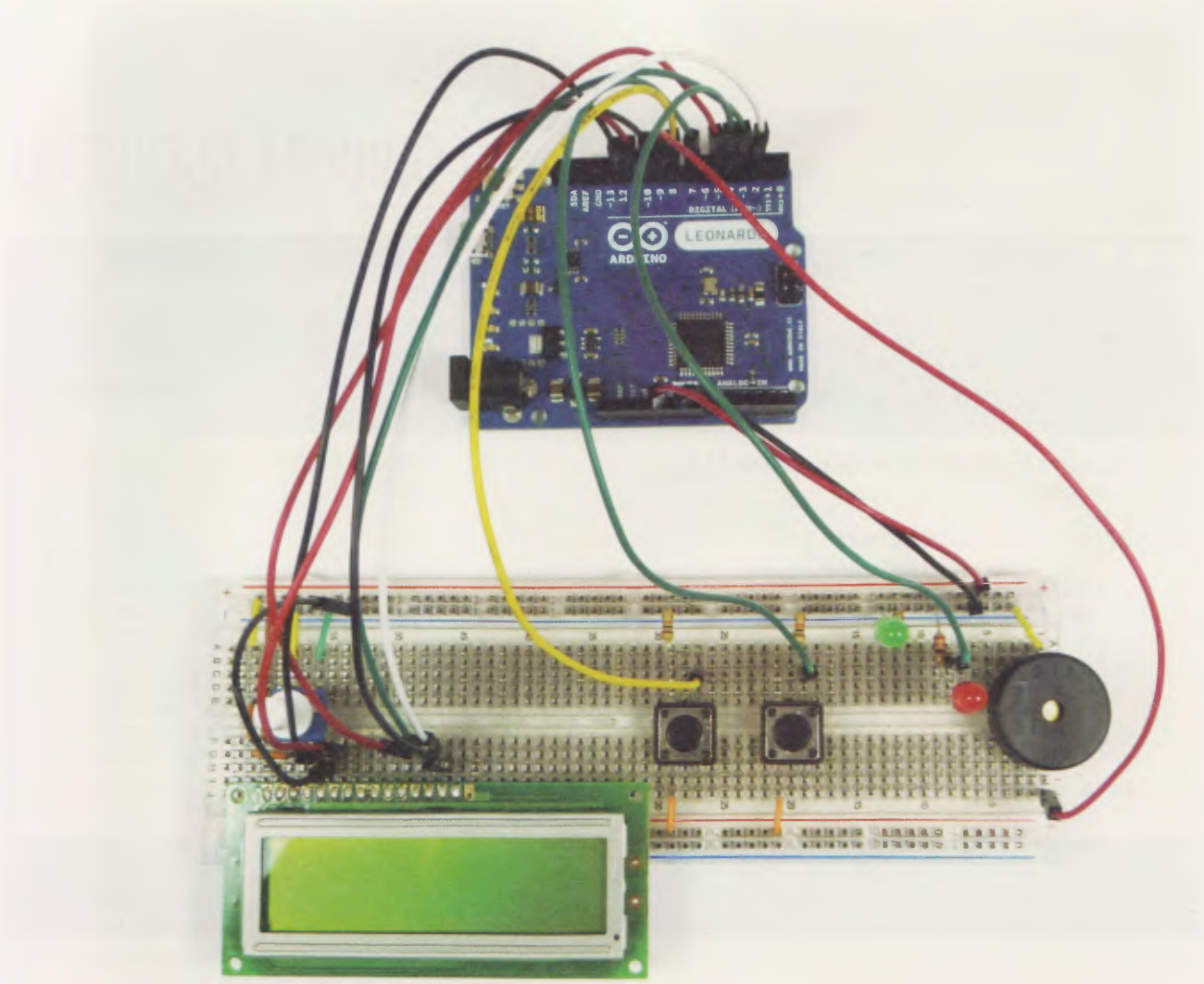
Arduino

À PROPOS DU CROQUIS

La grande majorité de notre croquis n'a pas changé, ni d'ailleurs son principe de fonctionnement. Ont simplement été ajoutés les éléments permettant de piloter l'afficheur LCD. Notre led de notification s'allume toujours pour signifier au moins une visite, mais nous n'avons plus besoin du clignotement pour décompter le nombre de visiteurs. Comme le fonctionnement de l'afficheur est statique, une fois les données envoyées et affichées, celles-ci restent en place, nous ne faisons que changer l'affichage. Il n'est pas utile de le rafraîchir de quelque manière que se soit.

PILOTAGE DE L'ÉCRAN

Le pilotage d'un afficheur LCD alphanumérique type HD44780 ou compatible est quelque chose de pris en charge de base par l'environnement Arduino. Il n'est pas nécessaire d'installer un module ou une bibliothèque tierce. Pour un projet en cours, l'ajout des fonctionnalités concernant l'afficheur LCD se limite généralement à passer par le menu « Croquis », « Importer bibliothèque » et « LiquidCrystal ». Ceci aura pour effet d'ajouter automatiquement la ligne suivante dans votre croquis :



```
#include <LiquidCrystal.h>
```

Bien entendu, ce n'est pas tout, car vous pouvez brancher les connecteurs de l'afficheur dans l'ordre qui vous chante en fonction des broches déjà utilisées sur l'Arduino. Pour simplifier la syntaxe et éviter de jongler mentalement avec des valeurs numériques, nous utilisons ici une spécificité du langage hérité de la syntaxe C/C++ : la définition de macros. Ainsi nous utilisons :

```
#define RS 12
#define EN 11
#define D4 5
#define D5 3
#define D6 2
#define D7 10
```

Dès lors, dans le croquis nous pouvons, par exemple, utiliser `RS` en lieu et place de la valeur `12`.

Notez cependant que ceci n'est pas une variable, mais quelque chose que l'environnement Arduino (le compilateur en particulier) va remplacer à la volée avant de vérifier/compiler le croquis. Ceci ne consomme pas de mémoire, mais n'est qu'une définition de synonyme.

Grâce à cela, nous pouvons utiliser ces « alias » pour procéder à la configuration de l'afficheur avec :

```
LiquidCrystal lcd(RS,EN,D4,D5,D6,D7);
```

Vous conviendrez que c'est bien plus lisible que `LiquidCrystal lcd(12,11,5,3,2,10)`... À ce stade, l'écran est utilisable et dans `setup()` nous pouvons alors spécifier sa taille avec `lcd.begin(colonnes, lignes)` et afficher un message sur la première ligne (par défaut), placer le curseur sur le premier caractère de la seconde ligne et écrire un second message :

```
lcd.begin(16,2);
lcd.print("En attente...");
lcd.setCursor(0,1);
lcd.print("Aucune visite");
```

N'oubliez pas, en informatique on compte à partir de zéro, `lcd.setCursor(0,1)` c'est donc la position 0 de la ligne 1 et ainsi, le premier caractère de la seconde ligne. Comme vous le voyez, l'utilisation même de la bibliothèque LiquidCrystal est très simple.

Voici ce que nous utilisons dès que quelqu'un sonne (ou du moins, que le nombre de visites change) :

```
lcd.noDisplay();
lcd.clear();
lcd.print("En attente...");
lcd.setCursor(0,1);
lcd.print(active,DEC);
lcd.print(" visite");
if(active>1)
  lcd.print("s");
lcd.display();
```

Tout ce code n'est pas nécessaire, mais nous nous efforçons ici de rendre l'affichage le plus agréable possible. Ainsi, nous évitons tout d'abord des effets parasites indésirables découlant de la mise à jour de l'affichage. Pour cela, nous utilisons tout d'abord `lcd.noDisplay()` pour éteindre l'écran (son contenu devient invisible, mais rien n'est effacé). Nous pouvons alors effacer l'écran, réinscrire le message de la première ligne, déplacer le curseur en début de seconde ligne et utiliser une syntaxe étendue de `lcd.print()` où nous utilisons notre variable `active` et `DEC` pour demander un affichage décimal (en base dix, par opposition à `HEX` pour hexadécimal par exemple, ou `BIN` pour binaire). Ceci aura pour effet d'écrire la valeur de `active` à l'écran et nous n'avons plus, ensuite, qu'à ajouter une espace et le mot « visite ».

Notez les deux dernières lignes de ce code. C'est ce genre de choses qui fait toute la différence entre une interface simplement pratique et une interface réellement élégante. Si « visite » est précédé d'un chiffre plus grand que deux, nous ajouterons un « s ». Cela nous « coûte » deux lignes de code, mais c'est tellement plus raffiné que de simplement utiliser « visite(s) »... DB

CE QUE NOUS AVONS APPRIS

L'élément central de cette évolution de la sonnette intelligente est l'écran LCD. Nous avons vu que la prise en charge d'un tel matériel est très simple, tout autant que la connexion du module à l'Arduino.

Ces informations sur l'utilisation d'un écran compatible HD44780 très courant seront réutilisables dans de nombreux autres montages.

Par la même occasion nous avons eu le plaisir de constater que les bibliothèques bien pensées sont écrites de manière générique sans imposé à l'utilisateur une connexion sur des broches spécifiques. Il existe une quantité impressionnante de bibliothèques et seules quelques unes sont intégrées par défaut dans l'environnement de développement Arduino. Bien entendu, toutes ne sont pas de cette qualité et il faut également savoir qu'il n'est pas rare de trouver plusieurs bibliothèques pour un même composant, certaines plus évoluée ou de meilleures facture que d'autres.

Enfin, nous avons appris qu'il ne suffit pas de reposer sur une bibliothèque pour qu'un montage se comporte comme on l'entend. Dans ce cas précis, nous avons choisi de suspendre l'affichage lorsque nous mettions l'écran à jour afin d'éviter d'obtenir des artefacts déplaisant. Ceci aurai put être prise en charge par la bibliothèque mais généralement celles-ci se contente de fournir les fonctions élémentaires de contrôle du matériel. Là encore ce n'est pas une vérité universelle, puisque certaines offre uniquement des « facilités » sans aucune prise en charge shield ou de composants.

Enfin, il est intéressant de constater qu'en ajoutant un seul élément à un montage, il peut arriver que sont intérêt en soit littéralement décuplé. C'est le cas d'un écran LCD en mode texte puisque pour quelques euros, une gamme complète de réalisations et d'évolutions deviennent possibles. Sans compter, bien sûr, l'effet obtenu, tout le monde n'ayant pas une sonnette à afficheur LCD chez soi.

Le circuit du mod exposé. On distingue le bouton de contrôle, les deux leds de rétro-éclairage de ce dernier ainsi que le mystérieux microcontrôleur marqué Q3 dont la sérigraphie a été meulée. Les fils rouge et noir vont directement sur la batterie li-ion et les vert et bleu au contrôle de tension sur le bas du mod.

E-CIGARETTE : RIEN DE MAGIQUE, QUE DU TECHNIQUE

Denis Bodor

Les cigarettes électroniques font fureurs actuellement et ce n'est sans doute pas près de s'arrêter. Des boutiques poussent comme des champignons partout en France et dans le monde. Mais qu'est-ce donc qu'une cigarette électronique ? La première réponse qui nous vient à l'esprit est « une belle occasion de faire quelque chose qu'on ne peut pas faire avec une cigarette : la démonter ! »

Le principe de fonctionnement d'une « vrai » cigarette est ultra-simple. Le tabac est soumis à une source de chaleur et se produit une combustion. Lorsque le fumeur « tire » une bouffée, il aspire l'air environnant via le tube de papier qui canalise le flux. L'apport d'oxygène contenu dans l'air attise la combustion et le fumeur inhale la fumée produite et tout ce qu'elle contient, dont la nicotine, le goudron, etc.

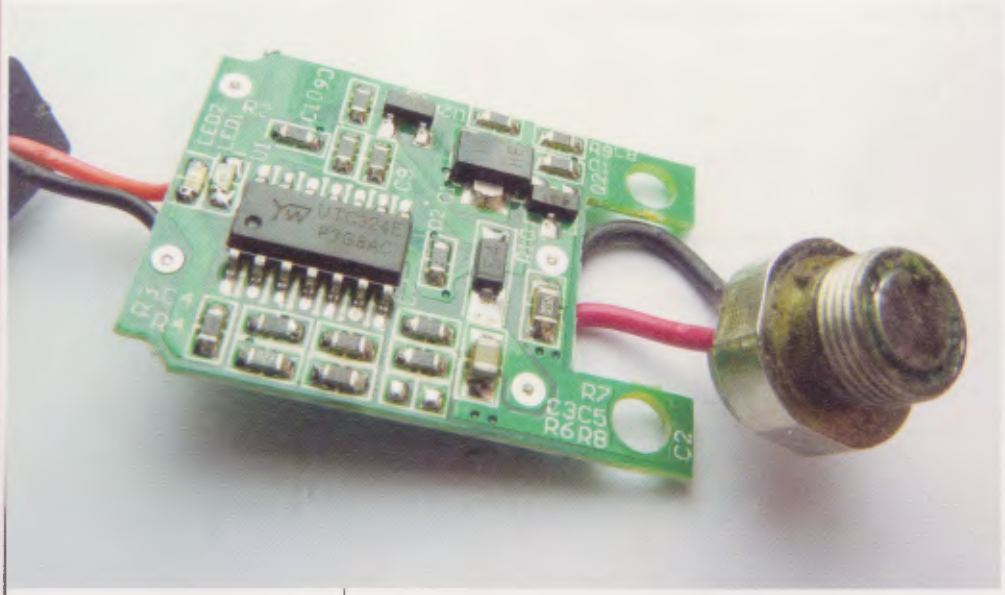
La cigarette électronique utilise un principe similaire mais différent. Ce n'est plus une masse compacte de tabac qui contient la nicotine mais un liquide composé de propylène glycol (PG) et/ou de glycérine végétale (VG), de 0 à 3,6 % de nicotine et d'arômes généralement alimentaires. Ce liquide est chauffé afin d'être vaporiser et inhalé. Contrairement à ce qu'on raconte généralement il ne s'agit pas à proprement parler de vapeur puisqu'au sens strict du terme, la vapeur est la forme gazeuse d'un corps à l'état solide ou liquide. Il s'agit plus exactement d'un aérosol ou « brouillard » composé de fines particules à l'état liquide en suspension dans le milieux gazeux qu'est l'air ambiant. Le chauffage du liquide est effectué à l'aide d'une résistance, construite avec un fil résistif en alliage de différents métaux (généralement nickel et chrome, du nichrome). Le liquide est conduit vers la résistance par capillarité grâce à une mèche en fibre de silice.

Bien entendu, une cigarette électronique ou e-cigarette est également l'occasion pour les « marketeux » d'inventer tout un tas de nouveaux termes bien plus sexy et vendeurs que les désignations réelles :

- le e-liquide est le liquide vaporisé,
- la résistance chauffante montée est appelée atomiseur, cartomiseur ou clearomiseur,
- le réservoir contenant le liquide est la cartouche ou le *tank*,
- la partie électronique contenant l'alimentation est le mod,
- la e-cigarette est l'ensemble du montage.

Bien entendu, s'est développé tout un écosystème aux noms farfelus et les amateurs aiment à disserter sur les forums de quel clearomiseur est le plus adapté à un mod donné et ce avec le e-liquide qui va bien. Notez au passage que vous ne devez voir dans ces propos absolument aucune animosité de ma part envers les utilisateurs de cigarettes électroniques, étant moi-même utilisateur de la chose. Je suis juste assez peu tolérant à l'invention de termes dont le seul but est de pousser au communautarisme et à tenter de faire des « vapoteurs » une sorte de caste pouvant se sentir valorisée et qui serait donc plus prompt à servir de relais marketing (si vous travaillez dans un service marketing, un conseil, montez un collectif, réunissez vos compères, trouvez-vous une île loin de tout et restez-y. Vous pourrez inventer plein de termes amusants et tenter de vous manipuler entre vous).

Un « clearomiseur » usagé. On remarque la simplicié de l'ensemble avec, comme composant principal, la mèche en fibre de silice et les contacts de la résistance. Comme on peut le constater, l'ensemble est largement dégradé du fait de son utilisation.



Le circuit présent dans le « chargeur » qui n'en est pas vraiment un. La puce, un UTC324, est un quadruple op-amp, apparemment destiné à servir de comparateur pilotant la LED1, témoin de charge. L'ensemble forme surtout un simple système de régulation.

1. ASSEZ DISSERTÉ ! OPÉRATION DÉMONTAGE

Notre victime sera ici, un *eGo-CTwist* fabriqué par *Joyetech* équipé d'un *aClear variable Clearomizer* de *Vivi Nova* (sérieux les gars, une île sympa juste pour vous). Nous étudierons la bête étage par étage afin d'en comprendre le fonctionnement et voir si, d'aventure, il ne serait pas possible de faire quelques modifications ou de se sortir de situations stressantes (type « plus de batterie et je vais devoir attendre que ça charge »).

1.1 Le réservoir et la résistance

Sans doute la partie la plus mystérieuse où l'étrange processus de vaporisation à lieu. L'ensemble réservoir/mèche est conçu pour être démonté. L'objet central est, en effet amovible car le chauffage récurrent de la résistance et de la fibre de silice dégrade l'ensemble et peut donc être remplacé pour un coût inférieur à l'ensemble. Bref, c'est un consommable comme le liquide lui-même. On constate que l'ensemble des réservoirs fonctionnent de la même manière, en utilisant comme conducteur la structure elle-même avec la masse sur la partie externe et le pôle positif au centre. Les deux étant séparés par un matériaux diélectrique (non conducteur).

En démontant davantage on constate que l'ensemble est d'une simplicité extrême. Le fil résistif enserre la fibre (mèche) en s'enroulant autour. On peut supposer que celui-ci est couvert d'une protection afin d'éviter que des contacts ne se forment et ne réduisent la résistance. Celle-ci n'est pas soudée mais tenu en contact sur le bord de la structure et en son centre, sans doute pour des raisons de coût de fabrication. L'ensemble est simplement vissé dans le réservoir qui assure ainsi la conduction des deux pôles.

De ce fait lorsque vous achetez une mèche de rechange, vous achetez en réalité la résistance, ce qui explique le marquage qui se trouve sur la partie métallique et qui indique une valeur en ohms. On constate également qu'il n'y a rien de bien complexe et de très technologique là dedans...

1.2 L'électronique d'alimentation

Voilà qui est plus intéressant. Avec un peu d'acharnement, on parvient à démonter le support de fixation du « mod ». On découvre alors un petit circuit double face regroupant quelques composants. L'un deux est difficile à identifier soit parce que la colle utilisée a effacé le marquage, soit parce que celui-ci à été délibérément effacé (par meulage). C'est une pratique courante dans certains équipements électroniques de fabricants qui ne souhaitent pas voir leurs concurrents en apprendre un peu trop sur les technologies utilisées. Il arrive également que certains produits, surtout venant d'Asie, intègrent des

composants n'ayant pas passés tous les tests qualités. Le constructeur peut alors trouver très judicieux d'en dissimuler la source.

Le composant TPS63020 est une puce Texas Instruments. Avec un peu de recherche sur le Web, on fini par trouver sa documentation qui le décrit comme un *« High Efficiency Single Inductor Buck-Boost Converter with 4A Switch »*. Voilà qui ne rigole pas ! Plus sérieusement, il s'agit d'un circuit de contrôle d'alimentation permettant de fournir du courant à une tension précise indépendamment de celle qu'on lui fournit. Il faut en effet savoir qu'une batterie, comme une pile, est un élément qui ne délivre pas une tension constante. Au fil de la baisse de l'énergie encore stockée dans l'élément, la tension varie et ce, selon une courbe non linéaire. La documentation du TPS63020 précise, entre autre, une tension de sortie de 1,2 à 5,5 volts et ce indépendamment de la tension en entrée qui peut être entre 1,8 et 5,5 volts. On remarque également que le fabricant annonce une efficacité de 96%, ce qui signifie que la conversion et le contrôle de l'alimentation ne coûte que 4% du courant qui circule. Enfin, la documentation précise que les applications typiques de ce composant adresse précisément les problématiques d'alimentation des appareils mobiles (smartphone, mediaplayer, équipement médical, etc) alimenté par des batteries NiCad, NiMh, Lithium et même les piles alcalines (ça c'est très intéressant).

On en déduit donc qu'en dehors du contrôle d'alimentation du fil résistif il ne reste qu'une chose qu'il faut prendre en charge : le chargement de la batterie et la signalisation de son état via les deux leds situées sur le circuit et éclairant le bouton par l'arrière. En toute logique, si le contrôle de charge de la batterie n'est pas du ressort de l'adaptateur de charge, le composant « masqué » ne peut servir qu'à cela. Il peut s'agir d'une puce dédiée mais c'est plus certainement, du fait du contrôle des leds (PWM et clignotement de notification), un microcontrôleur contenant un firmware, exactement comme un Arduino. Ceci rejoint la sérigraphie du circuit qui précise « STW boot 8 pin », ce qui regroupe là des termes qu'on jugera assez en rapport avec un microcontrôleur.

1.3 Le reste du mod

Derrière le circuit de contrôle se trouve la cellule de batterie lithium-ion (Li-ion) qui occupe la majorité de l'équipement. Nous n'avons ici qu'une seule cellule or on sait que la tension nominale de ce type de batterie est entre 3,6 et 3,7 volts, ce qui est totalement dans les marges du TPS63020. Il est intéressant de savoir qu'une batterie lithium-ion vieillira moins vite lorsqu'elle est rechargé par des recharges partielles que lorsqu'elle subie des cycles complets de décharge/charge. En d'autres termes, il est préférable de ne pas attendre que la batterie se vide totalement pour la recharger. Il est également probable que

Une mèche neuve. On distingue clairement le fil résistif entourant la fibre de silice. C'est cet élément qui chauffe et vaporise le liquide remontant par capillarité. Un principe de fonctionnement d'une simplicité déconcertante...

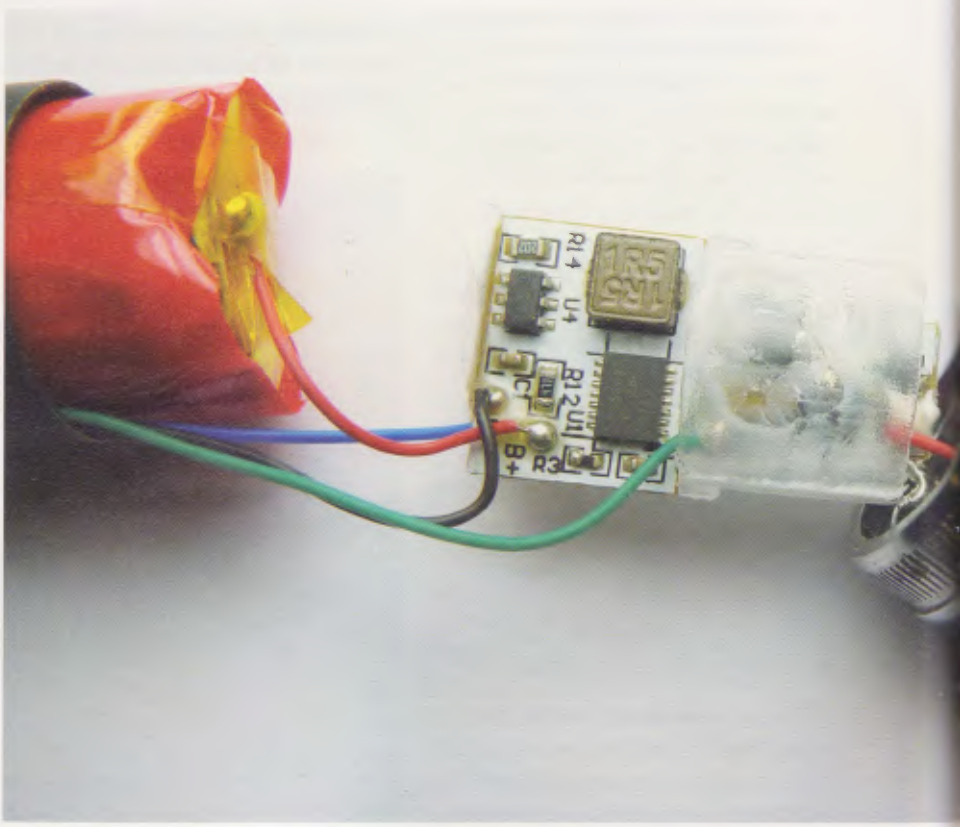
le microcontrôleur veille à cela et ne permet plus l'utilisation de l'appareil si la tension aux bornes de la batterie descend en dessous d'un certain seuil.

Mais un problème plus important est à noter et cela ne se limite pas aux cigarettes électroniques. Une batterie lithium-ion, ou autre, n'est pas un élément qui résiste au temps. Il s'use et de ce fait vieillira bien plus vite que l'électronique de contrôle où même les parties mécaniques de l'appareil. De ce fait, lorsque celui-ci ne fonctionnera plus parce que la batterie sera vétuste, le reste du matériel lui, aura encore de long mois (ou d'années) de fonctionnement devant lui. Oui, vous allez changer de mod, uniquement parce que l'élément le plus faible est inutilisable. Cette remarque s'applique également à n'importe quel appareil alimenté de la sorte : smartphones, players MP3, GPS, et même certains ordinateurs portables.

Le constructeur, bien entendu, le sait parfaitement mais il ne voit aucun problème à vendre un nouvel exemplaire de son produit ou une version plus récente, simplement parce que votre batterie est morte. Dans bien des cas, non seulement l'appareil ne permet pas de changer la batterie seule mais nécessite un démontage et donc l'annulation de la garantie. Plus problématique, il arrive même que le démontage de l'équipement soit rendu très difficile par le design du produit lui-même. Vous arriverez peut-être à l'ouvrir et remplacer l'élément en fin de vie mais il est moins probable que le remontage de l'ensemble conduise à un produit en parfait état. Pour ce type de démontage « sauvage » les anglophones ont un terme dédié : « crack-open ». Le fait de concevoir un produit dans ce sens, sachant pertinemment qu'il possède une durée de vie fixée par un des composants à un nom, c'est l'obsolescence programmée. Le produit est conçu pour devenir défectueux après une certaine durée. Chose qu'il ne faut pas confondre avec le MTBF pour *Mean Time Between Failures* ou le MTTF *Mean Time To Failure* qui indique une valeur temporelle moyenne qui informe sur la fiabilité d'un produit, tel que déclaré par un fabricant. L'obsolescence programmée, lorsqu'elle existe, ne figure pas dans les documentations...

S'il y a un conseil à donner concernant ce type d'équipement c'est tout simplement de les éviter comme la peste. Il faut TOUJOURS préférer les produits qui permettent de remplacer la batterie qu'il s'agisse d'un smartphone ou d'une cigarette électronique. Bien entendu, généralement ces produits modulaires sont haut de gamme et donc plus cher (c'est le cas, chez le même fabricant que notre produit de test, avec les modèles eVic et eMode).

Enfin, terminons par la partie la plus basse du produit, nous trouvons le sélecteur de tension. Celui-ci permet de régler la quantité de « vapeur » produite lorsqu'on appui sur le bouton de déclenchement de la cigarette. Il s'agit ni plus ni moins que d'un potentiomètre rotatif directement connecté au circuit



de contrôle et pilotant donc le régulateur TPS63020. La mesure de ce potentiomètre révèle qu'il possède une valeur de 92,3 Kohm et qu'il doit donc s'agir d'un modèle courant de 100K.

1.4 Le chargeur

Chose amusante, le « chargeur », bien plus aisé à démonter que la cigarette électronique elle-même, ne semble pas disposer de la logique nécessaire à la charge d'une batterie Li-ion. Ceci ne fait que confirmer nos soupçons concernant la puce inconnue trouvée précédemment. C'est le mod lui-même qui gère la charge. Le circuit du chargeur, qui n'en est donc pas un, est construit autour d'un UTC324.

Il s'agit d'un quadruple amplificateur opérationnel (ou op-amp pour les intimes), un circuit très utilisé en électronique analogique. Ici, cet amplificateur différentiel semble être utilisé pour piloter les deux leds témoins d'alimentation (rouge et vert) qui permettent à l'utilisateur de savoir où en est la charge de la batterie. Un op-amp est un composant magnifique (pour qui aime l'analogique). Dans les grandes lignes, son travail est de se débrouiller pour que ses deux entrées présentent la même tension et ce, en pilotant sa sortie. Il est souvent utilisé en guise de comparateur mais possède une palette d'applications absolument monstrueuse. C'est un composant classique, au point qu'on le retrouve dans de nombreux kits de découverte Arduino. L'UTC324 en intègre quatre mais, à y regarder de plus près, tous ne sont pas utilisés ici.

2. QUELQUES INTERROGATIONS

Ayant fait un tour d'horizon de cet ensemble, on en conclu que, finalement, une cigarette électronique n'a vraiment rien de très complexe. Ni dans son principe de fonctionnement, ni dans la conception d'un produit classique dans le domaine. On remarque également que cette simplicité explique la multiplication des produits bas de gamme. En effet, il est aisé pour un constructeur chinois de produire des matériels à très faible coût en utilisant des composants très bas de gamme ou de provenance douteuse. J'ai personnellement fait l'expérience déplaisante, dans un tout autre domaine, de ce type de « stratégie commerciale » avec une batterie pour laptop. Au démontage de la bête au fonctionnement très douteux, il s'est avéré que, si le boîtier et l'électronique interne était bel et bien de marque (IBM en l'occurrence), les cellules de la batterie, quant à elles, étaient de provenance, de qualité et d'age variables. Certaines n'étaient pas connectées et présentes uniquement pour apporter du poids à l'ensemble. Le circuit de contrôle avait même été modifié pour prendre en compte cette honteuse modification... Avec les cigarettes électroniques, c'est la même chose, la porte est grande ouverte à la contrefaçon.

Mais nos questions portent également sur le produit de marque et sa conception. Si l'on peut comprendre l'intégration totale de la batterie pour éviter d'avoir à manufacturer des parties mécaniques (support de batterie, embout vissable, etc) pourquoi avoir choisi de créer un

De l'autre côté du circuit de contrôle du mod, on distingue la puce TPS63020 à l'emplacement U1. Celui-ci assure la régulation de l'alimentation de la résistance chauffante et l'augmentation de tension. La cellule li-ion ne peut fournir que 3,7 volts, mais l'utilisateur peut régler une tension de 3,3 à 4,8 volts. C'est cette puce Texas instruments qui fait le travail.

« chargeur » et non d'intégrer un connecteur USB directement sur le mod (avec l'op-amp et le reste des composants) ? Certains diront sans doute « pour pouvoir vendre des chargeurs » et on ne peut que reconnaître que c'est là l'une des possibilités.

Une autre question dans le même sens concerne l'alimentation elle-même. Si un connecteur avait été proposé, il eut été possible de charger la batterie et d'utiliser la cigarette en même temps. Là encore, les plus suspicieux n'hésiteront pas à affirmer que cela ne peut que pousser à l'achat d'un second mod complet pour que l'un soit utilisé alors que le second est en charge.

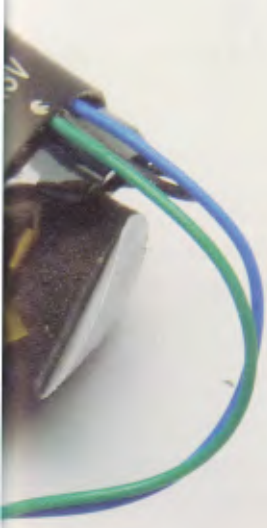
Le matériel est perfectible, c'est un fait. Mais peut-être s'agit-il là d'une logique marketing en dehors de nos compétences. Un mod de ce type coûte environs 25 euros. Le fabricant a peut-être estimé qu'augmenter le coût et donc le prix de 10% à 30% n'était pas une stratégie valable pour le marché. Mais on ne peut s'empêcher de penser qu'il n'y a pas vraiment de fautif à cet état de fait. En effet, si les utilisateurs étaient davantage au fait du fonctionnement de la technologie, il est très probable qu'ils accepteraient volontiers de payer plus cher pour un produit avec une durée de vie plus importante ou plus polyvalent quant à sa source d'alimentation. Mais encore faudrait-il que tout le monde sache ce que signifie « Li-ion », « cycle de charge/décharge » ou même simplement « volt ».

Un dernier point est plus dérangeant. Lorsqu'on parle de charge de batterie, il est indispensable de parler de sécurité. La charge d'un accu doit être non seulement contrôlée mais surveillée. Il n'est ainsi pas rare de trouver dans les batteries destinées aux appareils numériques plus de deux connecteurs. Il y a, bien entendu les pôles positif et négatif mais très souvent aussi une broche pour une sonde de température. Dans le cas du matériel qui nous occupe ici, il ne semble y avoir aucun contrôle de ce type. Certes la technologie li-ion est bien moins dangereuse que la lithium-ion polymère ou Li-Po mais cette absence de sécurité n'est pas très rassurante.

3. UN PETIT HACK POUR LA ROUTE ?

Avant de jouer avec le produit nous avons procédé à un petit test et quelques mesures. Il s'avère dans les faits que si la tension de la batterie tombe en dessous de 3,2 volts, les leds derrière le bouton poussoir clignotent signifiant qu'il faut recharger le mod. Ainsi, bien que le régulateur permettant de monter la tension jusqu'à 4,8 volts soit en mesure de gérer une source de 1,8 volts, une limite est fixée par le constructeur. Ceci rejoint notre petite remarque sur les cycles de charges/décharges des éléments li-ion.

Malheureusement, ceci veut également dire qu'il ne nous sera pas possible de simplement remplacer la cellule li-ion par une paire de piles AA par exemple, fournissant en série quelques 3 volts. Ce qu'il est possible de faire, en revanche, c'est de simplement passer outre le circuit de contrôle et d'alimenter la résistance chauffante avec une source contrôlée, type alimentation de laboratoire. Il est également possible d'envisager de reproduire intégralement le comportement du circuit mais en ne prenant en compte que la partie alimentation et non la charge. Bien entendu, nous n'obtiendrons pas le même niveau d'efficacité que le circuit originel. Il existe bien des solutions à base de régulateur LM317 ou LM338, permettant d'obtenir ce type de résultat. Il s'agit, pour tout dire, exactement de la même logique



L'utilisateur contrôle la tension utilisée sur le fil résistif via une molette au bas du mod. Il s'agit d'un simple potentiomètre rotatif de 100 Kohms qui contrôle le TPS63020.

que celle permettant de fabriquer une alimentation régulée (l'un des incontournables projets de tout électronicien débutant). Il faut garder toutefois à l'esprit que le courant utilisé est assez important. Une mesure lors de l'utilisation effective de la cigarette électronique nous montre par exemple que pas moins de 1,27 ampères circulent entre la batterie et le circuit, avec un réglage sur 3,3 volts. En poussant au delà de 4 volts, on dépasse les 2 ampères. Ceci fait tout de même 8 watts, ce qui est non négligeable et avec ce genre de puissance, quelques pour cents dissipés en chaleur pour la conversion est une dépense d'énergie aussi importante qu'inutile. Donc, oui, avec un régulateur comme un LM317 et une pile 9 volts, vous pouvez arriver à une solution « de secours » mais avec une efficacité relativement médiocre.

Mais le hack plus intéressant reste purement mécanique. En effet, la cellule intégrée dans le mod n'est pas différente de n'importe quelle cellule li-ion. Rien ne nous empêche donc d'utiliser une batterie type Samsung ICR18650-26F tel qu'on en trouve dans la cigarette électronique haut de gamme eVic. Il nous suffit de créer un nouvel assemblage en éliminant la batterie soudée et en la remplaçant par un support pour la nouvelle. Notez qu'il existe quantité de batteries de ce type, disposant de plusieurs formes et méthodes de connexion. Le modèle cité ici est, par exemple, vendu par RS au prix d'environ 11 euros pièce. On peut donc, à loisir remplacer l'élément de manière à palier aux manques de modularité du produit initial.

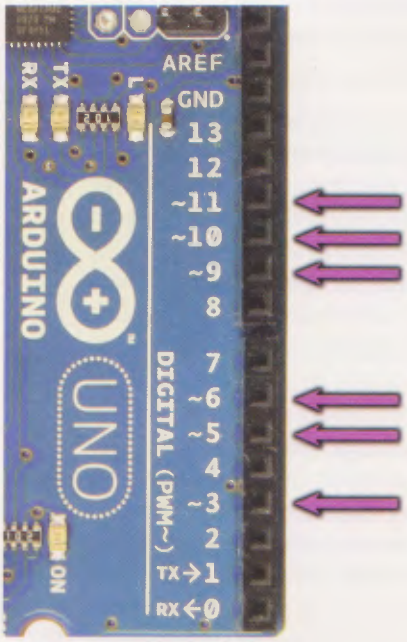
De plus, avec une telle approche, il sera bien plus aisé de charger d'avance quelques cellules de manière à n'avoir, finalement, qu'à changer cet élément. On peut également aller plus loin et trouver une approche moins « portable », en utilisant, par exemple des packs comme il en existe en modélisme.

Enfin, on ne peut s'empêcher de penser à une solution plus écolo basée sur l'utilisation d'un chargeur sur la base d'une dynamo à main comme il en existe pour les smartphones. Bien entendu, l'idée de charger le mod avec un tel chargeur à connecteur USB est une approche basique mais il n'est finalement pas nécessaire de recharger intégralement la batterie (sachant qu'en plus, tourner une manivelle pendant trois heures n'est certainement pas très amusant). Une petite manivelle sur le mod, couplé à un super-condensateur est très certainement à la porté d'un bon bricoleur et encore plus à celle d'un fabriquant un rien intelligent. Trente secondes d'effort physique minimum pour quelques minutes de vapotage semble être non seulement intéressant mais en plus très tendance, non ? DB

L'alimentation du « chargeur » est un bloc secteur relativement standard équipé d'un port USB pouvant fournir 500 mA sous 5 volts.

COMPRENDRE LA PWM

Comment obtenir un signal analogique à partir d'une sortie numérique. En d'autres termes comment, par exemple avec un interrupteur mural, faire en sorte de varier l'intensité lumineuse de l'ampoule qu'il contrôle ? Cette situation très courante en électronique numérique se règle avec une réponse qui tient en trois lettres, c'est la PWM !



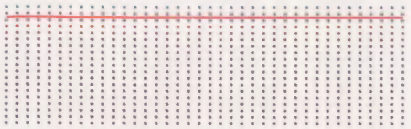
Certaines sorties d'une carte Arduino peuvent produire un signal PWM et même être reconfigurées pour ajuster la fréquence utilisée pour la modulation.

La modulation de largeur d'impulsions (MLI) plus communément appelée PWM, pour *Pulse-Width modulation*, peut être décrite académiquement comme une technique permettant de synthétiser des signaux continus à l'aide de circuits à fonctionnement tout-ou-rien. On parle d'états discrets dès lors qu'il s'agit de mettre en oeuvre des valeurs bien précises, ici allumé ou éteint, *on* et *off*, oui et non, haut et bas, etc. Voilà, merci d'avoir lu cet article et bon courage...

1. DÉMYSTIFIONS

Après une introduction telle que celle qui vient d'être faite, la PWM n'est pas forcément quelque chose qui donnes très envie, n'est-ce pas ? Pourtant, si vous êtes un utilisateur Arduino, il est fort probable que vous en ayez déjà fait usage. En effet, les sorties "analogiques" utilisées avec `analogWrite()` utilisent cette technique. En connectant une led à un port ainsi contrôlé, vous avez réellement l'impression de pouvoir choisir l'intensité lumineuse de la led comme si vous faisiez varier la tension ou le courant dans une ampoule. En réalité, ce n'est vraiment pas le cas. La quasi totalité des cartes Arduino ne possèdent pas de véritable sortie analogique permettant de définir arbitrairement une tension par rapport à la masse. L'Arduino Due le peut grâce à son DAC (*Digital to Analog Converter*) deux canaux mais les autres cartes n'offrent que des sorties PWM.

Ce que fait en réalité la sortie dite analogique consiste à faire clignoter la led mais de manière si rapide que, avec vos pauvres yeux d'humains (certains animaux sont vraiment mieux équipé que nous) vous ne percevez pas la succession de petites tranches de temps où la led est allumée puis éteinte. Ce clignotement est contrôlé par l'Arduino et, côté logiciels, ceci revient à choisir une valeur en 0 et 255.



rapport cyclique : 100%

Si notre pizza découpée en 16 parts dispose d'olives sur chaque part, le rapport cyclique est de 100%.

Disons le de suite et tout à faire clairement car c'est un erreur courante pour qui découvre la PWM : cette valeur ne correspond pas à la vitesse de clignotement de la led ! En d'autres termes, ce nombre que vous spécifiez n'a strictement rien à voir avec la fréquence utilisée. En PWM la fréquence est fixe (répétez-le à haute voix en faisant 27 tours sur vous-même).

Le meilleur moyen d'imaginer la PWM est d'imager son fonctionnement. Prenons donc une pizza (mais pas une *calzone* ça ne fonctionne pas) que nous faisons tourner sur elle-même. Nous fixons sa vitesse de rotation à 10 tours par seconde et ceci restera invariable. Divisons ensuite notre pizza en 16 parts égales. Il ne nous reste plus qu'à utiliser l'élément clé de l'ensemble : les olives (noires ou vertes, peu importe).

En plaçant une olive sur toutes les parts de la pizza et en observant l'ensemble en rotation depuis le bord de celle-ci, on peut dire que notre pizza présente donc 16*10 olives par seconde. On ne voit que des olives et notre pizza est composée de 100% de parts avec olives.

Si maintenant nous retirons la moitié des olives sur toute la partie gauche de la pizza, nous conservons 8 parts avec des olives. Cette fois, l'observateur va voir passer une succession de 8 parts avec olives, 8 parts sans, 8 parts avec, 8 part sans... Soit une alternance de groupes de 8 parts avec et sans, 10 fois par seconde. En une seconde il aura vu passer 80 parts de chaque type et donc un rapport de 50/50. Notre pizza tourne toujours à la même vitesse, seul le nombre de parts avec et sans olives change. On dit que le rapport cyclique est de 50%. Comme le dit Wikipédia, c'est le "*ratio entre la durée du phénomène sur une période et la durée de cette même période*". La durée de la période est de 1/10 de seconde (1 tour de pizza) et la durée du phénomène (la présence d'olive) est de la moitié de cette période, soit 1/20 de seconde.

Si, à présent, nous retirons encore davantage d'olives pour n'en laisser que 4 sur le premier quart de la pizza. L'observateur va voir passer 12 parts sans olives puis 4 parts avec et ce successivement pendant toute une seconde, soit 10 rotations de la pizza. Le rapport cyclique est maintenant de 25%, un quart de la pizza est avec olives. La vitesse de rotation n'a toujours pas changé.

La PWM d'une carte Arduino fonctionne de la même manière à la différence qu'il ne s'agit pas d'une pizza découpée en parts mais d'un bloc de temps découpé en 256 morceaux qui peuvent correspondre soit à la sortie active (+5V ou +3V selon la carte), soit à la sortie à la masse.

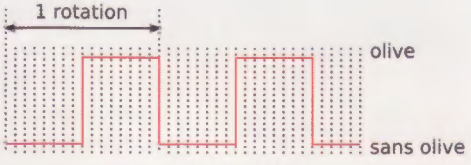
Pour notre pizza, la fréquence utilisée est de 10 Hz puisqu'elle opère 10 rotations complètes à chaque seconde. Elle possède une résolution de 16 tranches. Comme ces tranches peuvent être numérotées de 0 à 15, il nous suffit de 4 bits pour stocker cette numérotation. On dit qu'on dispose donc d'une PWM à une fréquence de 10 Hz avec une résolution de 4 bits.

2. PWM ET ARDUINO

La résolution de la plupart des Arduino (ceux basés sur l'Atmel ATmega328 par exemple) est de 8 bits, ce qui nous donne nos 256 tranches et donc la valeur de 0 à 255 qu'il est possible d'utiliser avec `analogWrite()`. La fréquence utilisée dépend à la fois de la configuration et des sorties utilisées. Par défaut, nous avons la configuration suivante :

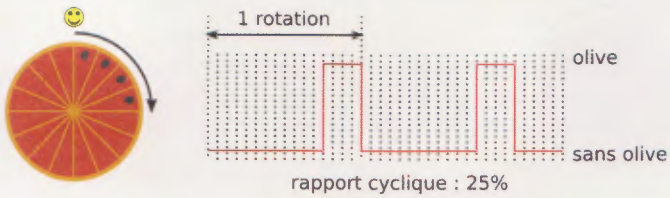
- broches 5 et 6 : 976,5625 Hz (via le timer 0),
- broches 9 et 10 : 490,20 Hz (timer 1),
- broches 11 et 3 : 490,20 Hz (timer 2).

Les *timers* sont des périphériques intégrés dans le microcontrôleur permettant, entre autre, de générer la PWM. Ce n'est pas le seul moyen de le faire puisque le système est, dans l'ensemble, relativement simple. Rien ne vous interdit de créer un croquis pour produire de la PWM de manière totalement logicielle. L'avantage dans le fait de disposer de timers pour cela est de décharger le processeur de cette tâche. Les timers, une fois programmés, fonctionne seuls et ne ralentissent pas votre code.



rapport cyclique : 50%

Avec seulement la moitié des parts comportant une olive, le ratio entre les parts avec et sans est de 50%, c'est notre rapport cyclique.



Un rapport cyclique ou duty cycle en anglais de 25% indique que dans la succession de parts de pizza 1/4 seulement possèdent une olive.

Notez qu'il est possible de changer ces fréquences même si, pour un usage courant, elles ont été choisies pour être utiles pour la plus vaste gamme d'applications. Il est ainsi possible de pousser la fréquence jusqu'à 62,5 KHz ou 32,372 kHz en reconfigurant les timers dans la fonction `setup()` de vos croquis. Ceci nécessite cependant une connaissance avancée des registres de configuration des microcontrôleurs AVR équipant les Arduino et, plus important encore, cela impacte le fonctionnement d'autres fonctions comme `millis()` et `delay()`. Celles-ci utilisent, en effet, les timers pour mesurer le temps. En augmentant ou réduisant la fréquence du timer 0, les valeurs utilisées avec ces fonctions ne seront plus des millisecondes et il faudra donc prendre ces changements en compte dans vos croquis. Nous reviendrons sur le sujet dans un article dédié plus tard.

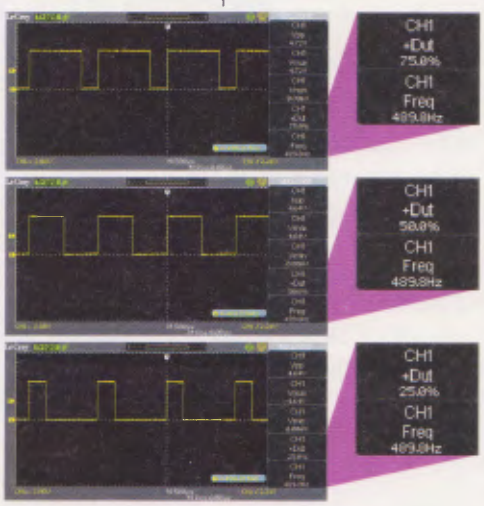
3. À QUOI PEUT SERVIR LA PWM ?

Il existe beaucoup d'éléments, de composants ou de périphériques qui nécessitent pour fonctionner une entrée analogique ou en d'autres termes qui veulent qu'on présente à leurs bornes une tension arbitrairement choisie et non simplement 0 ou 5 volts (tout-ou-rien). Le cas de la led variant d'intensité est un peu particulier car celle-ci ne demande pas ce genre de choses. Elle fonctionne avec un courant donné et présente une tension fixe à ses bornes. Ce n'est pas un composant analogique. La PWM est utilisée pour varier l'intensité lumineuse mais en se basant sur une limitation de la capacité de la vision humaine.

Des choses comme des moteurs à courant continu (DC) ou des périphériques audio peuvent être contrôlés par une carte ne disposant pas de sortie analogique, si celle-ci utilise la PWM. En modulant la largeur d'impulsion on arrive, par exemple, à varier la vitesse d'un moteur en ne lui fournissant pas continuellement du courant. C'est là un domaine où la PWM est largement utilisée car, d'un point de vu mécanique, il est peu judicieux de démarrer un moteur à pleine vitesse. On utilise alors la PWM pour progressivement accélérer le mouvement lors du démarrage et décélérer ensuite lors de l'arrêt. Les ventilateurs présents dans un PC utilisent également la PWM pour varier la vitesse de rotation en fonction de la température des éléments à refroidir et en mesurant continuellement cette vitesse de rotation.

Il est important de garder à l'esprit qu'une sortie en PWM n'est pas une sortie analogique. Cela reste une succession d'états discrets mais il est possible, à l'aide d'un montage simple, d'utiliser la PWM pour produire un signal analogique. On utilise généralement pour cela un réseau RC (ou LC) composé d'une résistance (ou d'une bobine) et d'un condensateur. Celui-ci fait alors office de filtre passe-bas qui laissera passer les basses fréquences (les changements lents) mais atténue les hautes fréquences (celles utilisées pas la PWM). Grâce à ce filtrage, il devient possible d'utiliser une sortie tout-ou-rien en PWM pour moduler un signal comme un sinus qui peut alors être amplifié pour une multitude d'usages (audio, émission radio, réception radio, modulation/démodulation, etc).

La conversion PWM/analogique sort du cadre de ce simple article de présentation de la PWM mais nous reviendrons sur le sujet dans un prochain numéro en construisant une véritable sortie analogique à partir d'une carte Arduino. Vous savez maintenant ce que signifie la PWM et comment elle fonctionne, une brique de plus à mettre à votre actif. DB

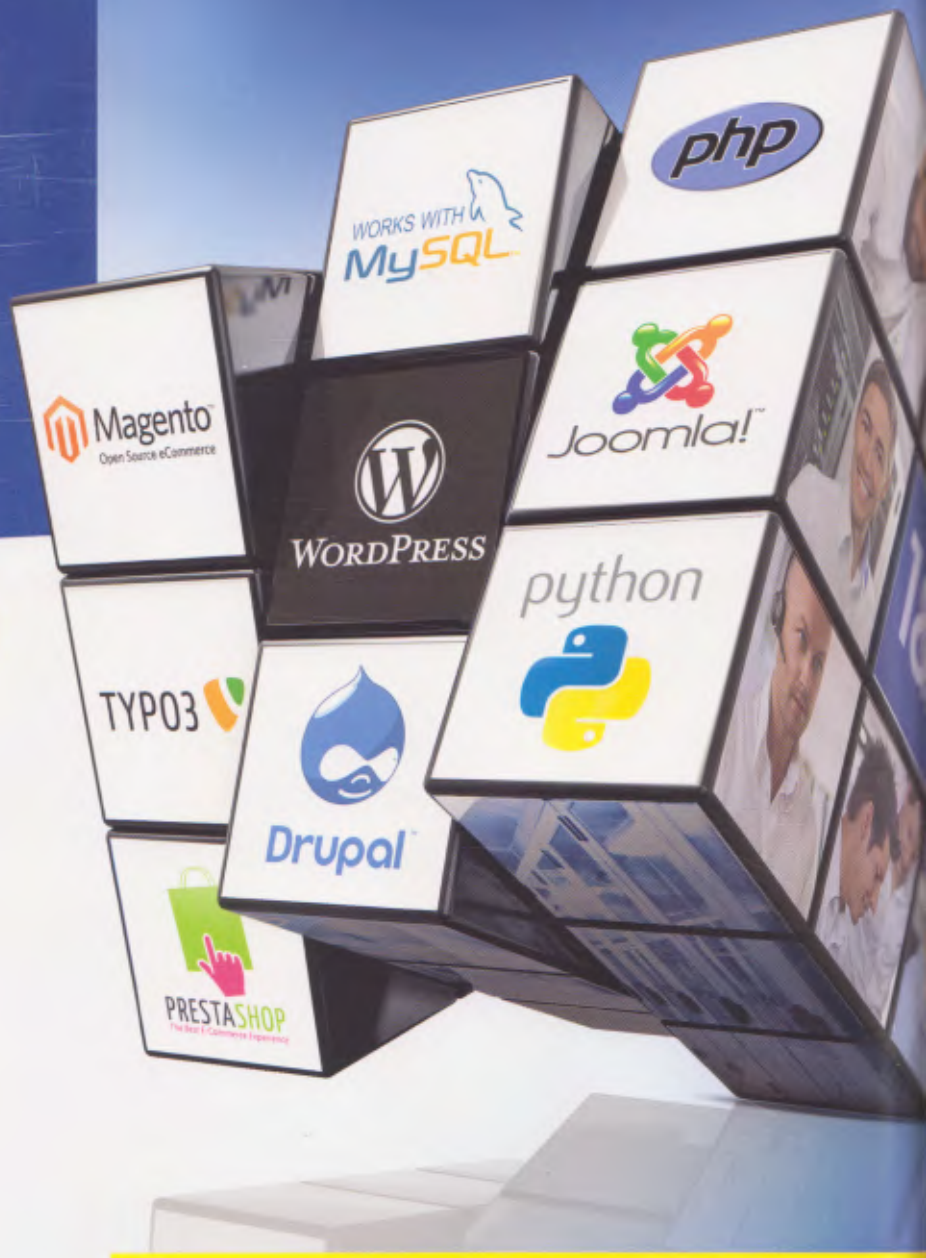


Le signal issu d'une carte Arduino Uno (port 3) vu à l'oscilloscope. On distingue clairement le fonctionnement de la PWM avec des rapports cycliques de 75%, 50% et 25% configurés via la fonction analogWrite() avec des valeurs passées respectivement à 192, 128 et 64. Notez la fréquence fixe, invariablement à environ 490 Hz.

DOMAINES | MAIL | **HÉBERGEMENT** | E-COMMERCE | SERVEURS

NEW HOSTING

AVEC LES MEILLEURES APPLICATIONS !



WordPress & 140 Apps populaires

- Support dédié 24/7 assuré par les experts 1&1
- **Versions d'évaluation disponibles**
- Notifications de sécurité : mises à jour, patchs...
- Performance optimale : 2 Go de RAM garantis
- Service complet pour 140 Apps & CMS réputés : WordPress, Joomla!™, Drupal™, TYPO3, Magento®

Outils de référence

- **PHP 5.5**, Perl, Python, Ruby
- 1&1 Mobile Website Builder
- NetObjects Fusion® 2013 – 1&1 Edition

Marketing efficace

- 1&1 Référencement Pro
- 1&1 Newsletter Tool
- 1&1 WebStat
- Crédits Facebook®

Infrastructure High-Tech

- Disponibilité maximale grâce à la **géo-redondance**
- Connectivité > 300 Gbits/s
- 1&1 CDN powered by CloudFlare® (23 PoPs)

Tout inclus

- Nom de domaine : .fr, .com, .net, .eu, .org, .info
- Ressources illimitées : espace Web, trafic, comptes email et bases de données MySQL
- Système d'exploitation Windows ou Linux

PACKS COMPLETS POUR PROFESSIONNELS

À partir de **0,99** € HT/mois*

1 MOIS POUR ESSAYER

1 CLIC POUR CHANGER DE PACK

1 APPEL UN EXPERT VOUS RÉPOND

0970 808 911
(appel non surtaxé)

1&1

1and1.fr

*30 jours « satisfait ou 100 % remboursé ». Les packs hébergement 1&1 New Hosting sont à partir de 0,99 € HT/mois (1,19 € TTC) la 1ère année au lieu de 2,99 € HT/mois (3,59 € TTC) pour un engagement de 12 mois. À l'issue des 12 premiers mois, les prix habituels s'appliquent. Offres sans durée minimale d'engagement également disponibles. Offres à durée limitée et soumises à conditions détaillées disponibles sur 1and1.fr.