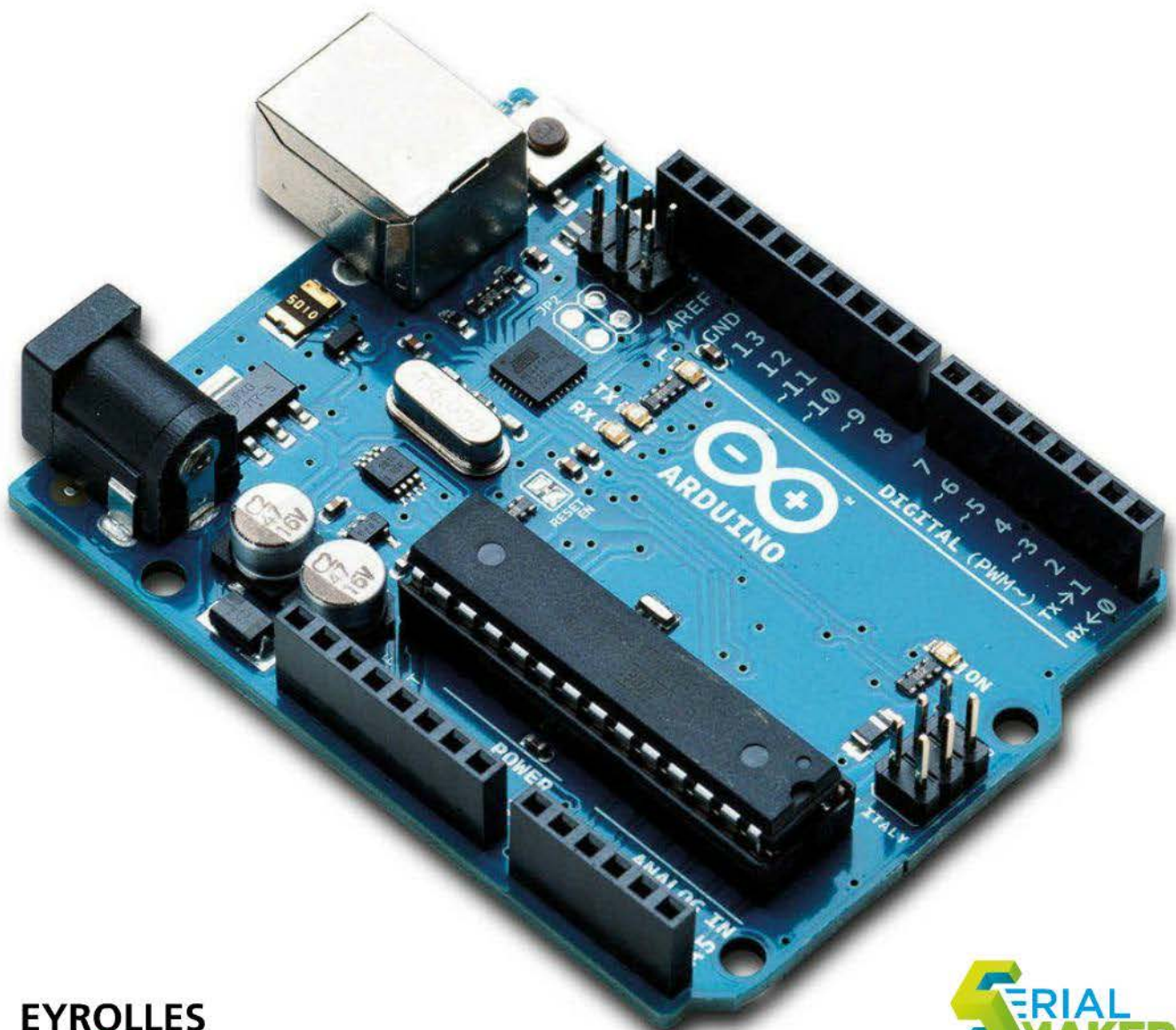


LE GRAND LIVRE D'ARDUINO

Erik Bartmann

2^e édition



22 montages à réaliser avec Arduino

Avec son petit microcontrôleur hautement performant et facilement programmable, la plate-forme libre Arduino a révolutionné le mouvement *Do It Yourself*. Se couplant aisément avec d'autres composants (écrans LCD, capteurs, moteurs...), elle est devenue aujourd'hui un élément indispensable dans de nombreux dispositifs électroniques. Sa simplicité d'utilisation, l'étendue de ses applications et son prix modique ont conquis un large public d'amateurs et de professionnels : passionnés d'électronique, designers, ingénieurs, musiciens...

Remarquable par son approche pédagogique, cet ouvrage de référence vous fera découvrir le formidable potentiel d'Arduino, en vous délivrant un peu de théorie et surtout beaucoup de pratique avec ses 22 montages à réaliser. Mise à jour avec les dernières évolutions d'Arduino, cette deuxième édition s'est enrichie de deux nouveaux chapitres et de projets à monter avec un Raspberry Pi ou une carte Yún.

À qui s'adresse ce livre ?

Aux électroniciens, bricoleurs, bidouilleurs, hobbyistes, ingénieurs, designers, artistes, makers...

Dans ce livre, vous apprendrez notamment à :

- créer un séquenceur de lumière
 - fabriquer un afficheur LCD
 - commander un moteur pas-à-pas
 - réaliser un shield
-

Sur www.serialmakers.com

Téléchargez le code source des sketches Arduino présentés dans cet ouvrage.

Électronicien de formation, **Erik Bartmann** est aujourd'hui développeur pour le principal fournisseur européen d'infrastructures informatiques. Passionné d'électronique depuis toujours, il est l'auteur de plusieurs ouvrages sur Arduino, Processing et le Raspberry Pi, parus aux éditions O'Reilly.

O'REILLY®

LE GRAND LIVRE
D'ARDUINO

CHEZ LE MÊME ÉDITEUR

Dans la collection « Serial Makers »

C. PLATT. – **L'électronique en pratique.**

N°13507, 2013, 344 pages.

C. BOSQUÉ, O. NOOR et L. RICARD. – **FabLabs, etc.** *Les nouveaux lieux de fabrication numérique.*

N°13938, 2015, 216 pages.

M. RICHARDSON et S. WALLACE. – **À la découverte du Raspberry Pi.**

N°13747, 2013, 176 pages.

B. PETTIS, A. KAZIUNAS FRANCE et J. SHERGILL. – **Imprimer en 3D avec la MakerBot.**

N°13748, 2013, 226 pages.

M. BERCHON. – **L'impression 3D (2^e édition).**

N°13946, 2014, 232 pages.

R. JOBARD. – **Les drones.** *La nouvelle révolution.*

N°13976, 2014, 190 pages.

J. BOYER. – **Réparez vous-même vos appareils électroniques.**

N°13936, 2014, 384 pages.

Erik Bartmann

LE GRAND LIVRE
D'ARDUINO

Deuxième édition

EYROLLES



ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Authorized French translation of the German edition of *Die elektronische Welt mit Arduino entdecken, 2.Auflage* by Erik Bartmann, ISBN 978-3-95561-115-6 © 2014 by O'Reilly Verlag GmbH & Co. KG. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Traduction autorisée de l'ouvrage en langue allemande intitulé *Die elektronische Welt mit Arduino entdecken, 2.Auflage* d'Erik Bartmann (ISBN : 978-3-95561-115-6)

Adapté de l'allemand par Danielle Lafarge et Patrick Chantereau

L'éditeur remercie vivement Olivier Decelle pour sa validation technique de l'ouvrage.

Avant-propos

Il est frappant de constater qu'aujourd'hui, nous sommes entourés d'objets préfabriqués qu'il n'est pas possible ou presque de modifier. Ajouté à cela, les médias (journaux, télévision, Internet...) qui nous assènent certaines pseudo-vérités que beaucoup tiennent pour acquises sans chercher plus loin. D'où le risque d'une mise sous tutelle insidieuse de l'individu, que nous ne devons absolument pas sous-estimer. Nous sommes très peu sollicités pour comprendre notre environnement quotidien : dans ce contexte, que reste-t-il de notre créativité ? En lisant ces lignes, vous vous demandez peut-être quel est le rapport avec Arduino et si vous avez en main le livre qu'il vous faut. Pas d'inquiétude, si j'ai choisi de débiter cet avant-propos par ce texte quelque peu provocateur mais somme toute réaliste, c'est parce que cet ouvrage va vous donner les moyens d'exprimer toute votre créativité.

L'électronique est un vaste domaine, idéal pour donner libre cours à son imagination, selon son envie et son humeur. C'est pourquoi cet ouvrage ne se contente pas de présenter des montages prêts à l'emploi. Certes, il en faut, mais son but premier est de fournir des pistes au lecteur pour qu'il développe ses propres circuits. Les kits de montage électrique, à assembler selon un « schéma F », sont d'emblée efficaces et attrayants, et garantissent que tout fonctionne comme son inventeur l'a souhaité. Mais soyons honnêtes : monter un tel circuit constitue-t-il un exploit remarquable ? Certainement pas. En se présentant comme un creuset d'idées, ce livre va beaucoup plus loin et favorisera l'inventivité de tout amateur d'électronique.

Cela étant, un petit coup de pouce sera nécessaire au départ, car pour développer vos propres projets Arduino, vous devrez d'abord en connaître les bases. Mais ce processus est tout à fait normal : pour

apprendre à marcher, courir, lire ou écrire, nous avons dû aussi faire appel à l'aide d'autrui.

Arduino est un circuit imprimé open source. Vous connaissez déjà sûrement ce qualificatif pour des logiciels gratuits, où chacun peut prendre part au développement du projet et y apporter sa contribution. Ce mode de collaboration, réunissant beaucoup de personnes intéressées et engagées, possède un fort potentiel et fait clairement avancer les choses. Les résultats parlent d'eux-mêmes et n'ont rien à envier à ceux des projets commerciaux.

Sous ce nom Arduino se cachent non seulement du matériel mais aussi un logiciel. On parle alors de *Physical Computing*, qui désigne la construction de systèmes interactifs permettant de connecter le monde physique à celui des ordinateurs. Le monde dans lequel nous vivons est considéré comme un système analogique, alors que les ordinateurs agissent dans un environnement numérique ne connaissant que les états logiques 0 et 1. C'est à nous, individus créatifs, qu'il appartient d'établir une liaison entre ces deux mondes et de montrer par des actions et des faits de quoi nous sommes capables.

Cet ouvrage traite de deux thématiques fondamentales, dont nous ne pourrions nous affranchir :

- l'électronique (composants et fonctions) ;
- le microcontrôleur (la carte Arduino).

Naturellement, tout livre étant limité en volume, ces deux thèmes n'ont pu être traités ici de manière exhaustive. Si vous souhaitez en savoir davantage, la littérature existante sur ce sujet est abondante, sans compter toutes les informations disponibles sur Internet. Cet ouvrage se veut juste être l'instigateur, voire le déclencheur, d'une irrépressible soif d'apprendre. Je serai donc très heureux si j'en suis un peu pour quelque chose. Mais concentrez-vous d'abord sur l'ouvrage que vous avez dans les mains.

Au début, tout sera simple et facile au point que certains se demanderont quel est l'intérêt de faire clignoter une diode. Pourtant, soyez assuré que ce petit montage est lui aussi une pierre de l'édifice. N'oubliez pas qu'une phrase est composée de lettres qui, prises isolément, ne veulent pas dire grand-chose, mais qui, une fois assemblées, nous permettent de communiquer. En électronique, c'est le même principe : tout circuit n'est ni plus ni moins qu'une association judicieuse de composants électriques.

Nouveautés de la deuxième édition

Depuis la sortie de la première édition de mon livre sur Arduino, les choses ont quelque peu évolué : de nouvelles cartes Arduino sont apparues sur le marché, ouvrant de nouvelles voies créatives. J'en présenterai quelques-unes au chapitre 2 en détaillant leurs avantages et inconvénients.

Il n'est pas facile pour Arduino de s'affirmer alors que vient d'arriver l'ordinateur monocarte Raspberry Pi. Beaucoup d'entre nous se sont déjà laissé séduire par les promesses de ce nano-ordinateur qui, avec son système d'exploitation Linux et son connecteur GPIO, offre des possibilités infinies. Pourtant, je ne pense pas qu'il puisse détrôner la carte Arduino, qui possède deux atouts majeurs : d'une part, elle est spécifiquement destinée à la réalisation de montages électroniques et, d'autre part, elle est très facile à programmer au moyen du langage C/C++. D'ailleurs, dans le domaine du *prototypage* – le montage rapide de circuits interactifs –, Arduino a toujours une longueur d'avance. Vous avez une idée, vous attrapez votre carte Arduino et vous commencez à programmer et à connecter des composants.

Toutefois, il n'est pas étonnant que de nombreux fans d'Arduino se sentent menacés par le Raspberry Pi qui fait figure d'étranger ou d'intrus. Mais il me semble qu'il se base sur une tout autre approche. Il est davantage destiné aux débutants en informatique qui souhaitent s'initier à différents langages et découvrir les principes fondamentaux de la programmation. Il convient parfaitement pour un premier contact avec un mini-ordinateur qui, au fond, n'a rien à envier à un véritable ordinateur : comme ses aînés, il est doté de ports pour un clavier, une souris, un écran, une connexion réseau ou des périphériques USB, de mémoire et d'un processeur.

Arduino, au contraire, a été conçu pour offrir aux bricoleurs un outil leur permettant de parvenir rapidement à un résultat exploitable. L'interactivité est et demeure au premier plan. La carte Arduino est imbattable dans le domaine de la collecte de données de capteurs devant être traitées quasiment en temps réel pour déclencher certaines (ré)actions, comme le pilotage d'actionneurs (moteurs, servomoteurs).

Aujourd'hui, on a tendance à adopter des positions très tranchées. Tout est soit bon, soit mauvais, noir ou blanc. L'esprit de compétition domine. Pourtant, pourquoi les deux environnements ne pourraient-ils pas coexister ? Grâce au Raspberry Pi, Arduino pourrait étendre le

champ de ses possibilités créatives en repoussant encore les limites au point que tout, ou presque, serait envisageable. Cela dit, nous nous intéresserons essentiellement à Arduino dans ce livre, sinon il aurait fallu lui donner un autre titre.

Ah, j'allais oublier un détail. Il existe une nouvelle carte Arduino dotée d'un processeur additionnel sur lequel tourne une distribution Linux. Il s'agit de l'Arduino Yún. Comme vous pouvez le constater, les développeurs d'Arduino ne se reposent pas sur leurs lauriers et ils ont senti le vent tourner ! Cette nouvelle carte allie la technologie Arduino existante au système d'exploitation Linux : les deux univers peuvent communiquer pour échanger des données ou des informations. Je parlerai aussi de cette nouvelle carte dans ce livre.

Cette deuxième édition s'est également étoffée d'un nouveau chapitre sur le logiciel de prototypage Fritzing, et de trois nouveaux montages exploitant le Raspberry Pi et la carte Yún.

Structure de l'ouvrage

Vous allez certainement vite remarquer que le style employé dans cet ouvrage diffère un peu de ce qu'on peut lire habituellement. En effet, j'ai opté pour un ton familier et complice. Au fil des pages, je vous ai donné le rôle d'un candide qui pose çà et là des questions, qui vous sembleront pertinentes dans certains cas, et stupides dans d'autres – mais c'est totalement voulu. En raison de la clarté et de la simplicité de certains sujets, on se refuse parfois à poser des questions de peur d'être ridicule. Dans ce livre, vous ne connaîtrez pas ce sentiment puisque, par chance, quelqu'un d'autre posera ces questions à votre place !

Par ailleurs, j'ai préféré ne pas vous confronter dès le début de l'ouvrage aux principes de l'électronique et de la programmation Arduino, car cela aurait donné au livre une approche trop scolaire que je souhaitais précisément éviter. Aussi, les thématiques seront abordées en temps voulu et intégrées dans des exemples. Vous ne disposerez ainsi que du strict nécessaire au moment précis de l'apprentissage. En outre, les principales instructions sont regroupées à la fin du livre dans un référentiel que vous pourrez consulter à tout moment si besoin.

La structure de chaque chapitre suit un déroulement relativement classique. Je commence par présenter les différentes thématiques qui y seront traitées, afin que vous ayez un aperçu de ce qui vous attend, puis j'entre dans le vif du sujet en développant et analysant le thème

proprement dit. Je clos enfin le chapitre par un récapitulatif des domaines abordés, ce qui permettra de renforcer encore un peu plus les connaissances acquises.

La plupart des langages de programmation étant d'origine américaine, toutes les instructions seront en anglais. Naturellement, j'expliquerai tous les termes qui le méritent.

Voici comment sont présentés la plupart des 22 montages proposés dans l'ouvrage. Vous y trouverez, dans l'ordre :

- les composants nécessaires ;
- le code du programme ;
- la revue de code (l'analyse du code) ;
- le schéma électrique ;
- la conception des circuits ;
- les problèmes couramment rencontrés (et que faire si cela ne marche pas du premier coup ?) ;
- un récapitulatif de ce que nous avons appris ;
- un exercice complémentaire pour approfondir la thématique.

Certaines expériences sont accompagnées de relevés d'oscilloscope ou d'enregistrements d'analyseur logique visant à mieux faire comprendre le parcours du signal.



Pour aller plus loin

Dans ce type d'encadré, vous trouverez des informations utiles, des astuces et des conseils liés au thème abordé. Je vous fournirai aussi des mots-clés pour continuer votre recherche sur Google, ainsi que certaines adresses Internet incontournables (comme des fiches techniques de composants électroniques), les autres étant susceptibles de changer ou même de disparaître dans un futur proche.



Attention !

Lorsque vous rencontrerez cet encadré, lisez-le attentivement afin de garantir le succès de votre montage.

Pour certains montages, je proposerai des solutions *Quick & Dirty* qui pourront surprendre à première vue. Mais elles seront suivies d'une variante améliorée qui devrait vous faire dire : « Tiens, ça marche aussi comme ça et même pas mal du tout ! Cette solution est encore meilleure. » Si c'est le cas, alors j'aurai atteint le but que je m'étais fixé. Sinon, ce n'est pas grave : tous les chemins mènent à Rome...

Code source de l'ouvrage

Le code des sketches Arduino présentés dans cet ouvrage est disponible à l'adresse <http://www.serialmakers.com/livres/le-grand-livre-arduino>. Vous y trouverez également des compléments et des liens utiles.

Prérequis

Le seul prérequis personnel est d'aimer le bricolage et les expériences. Nul besoin d'être un *freak* de l'électronique ou un expert en informatique pour comprendre et reproduire les montages de ce livre.

Nous commencerons en douceur afin que chacun puisse suivre. Ne vous mettez pas de pression, le premier objectif de cet ouvrage est de vous distraire !

Composants nécessaires

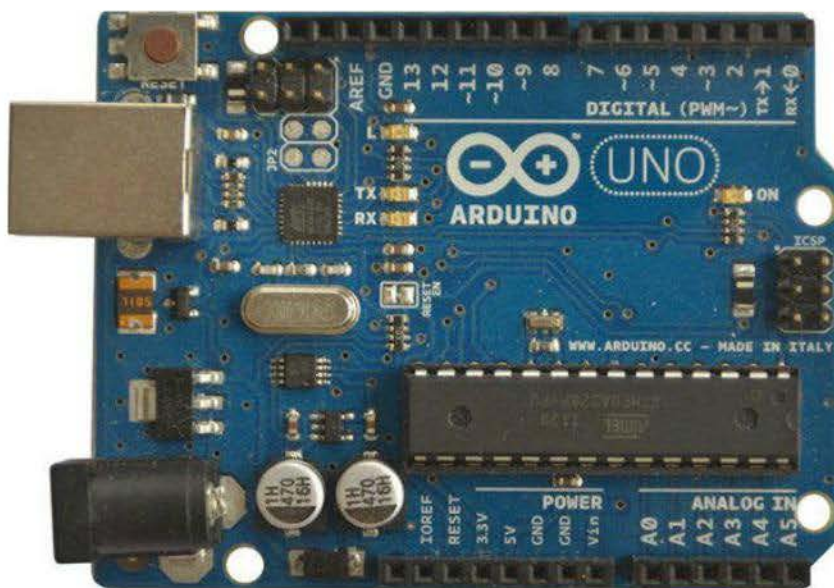
Notre carte Arduino est certes bien sympathique, et nous apprécions que tout y soit si bien pensé et si petit. Mais il faudra quand même passer à l'étape suivante et connaître tout ce qu'il est possible d'y raccorder. Si vous n'avez pas l'habitude de manipuler des composants électroniques (résistances, condensateurs, transistors, diodes...), pas d'inquiétude. Chacun fera l'objet d'une description détaillée, afin que vous sachiez comment il réagit individuellement et au sein du circuit. Pour chaque montage, j'indiquerai en outre la liste des composants nécessaires. Naturellement, l'élément-clé de tout circuit sera toujours la carte Arduino, mais je ne la mentionnerai pas forcément de manière explicite. À ceux qui se demandent combien coûte une carte Arduino et s'ils pourront conserver leur train de vie après un tel achat, je répondrai : « Yes, you can ! » Elle est très bon marché, aux alentours de 25 euros.

Dans tous les exemples, j'utilise la carte Arduino Uno, la plus populaire. Je présenterai aussi la carte Arduino Yún qui utilise Linux, mais qui coûte tout de même la bagatelle de 60 € environ.

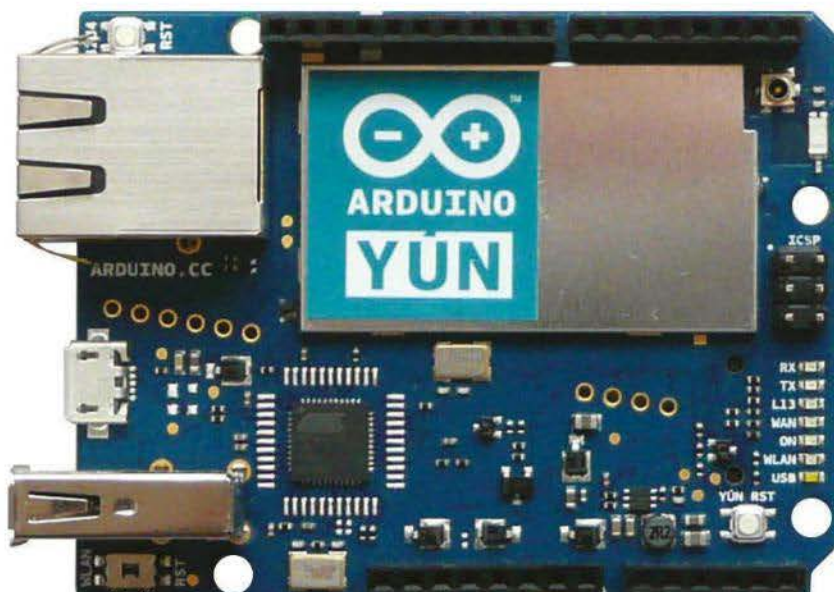
Dans tout ce livre, j'ai veillé à n'utiliser aucun composant rare, sophistiqué ou coûteux. D'ailleurs, vous qui ne jetez sûrement rien, vous avez peut-être encore à la cave ou au grenier des appareils électroniques usagés (scanner, imprimante, lecteur de DVD, magnétoscope, radio, etc.) que vous pourrez démonter pour récupérer divers

composants. Mais attention, assurez-vous que les appareils soient toujours débranchés avant de les ouvrir : faute de quoi, vous risqueriez de nous quitter avant la fin de l'ouvrage !

Toutes les expériences sont réalisées avec une tension d'alimentation de 5 ou 12 V.



◀ **Figure 1**
La carte Arduino Uno



◀ **Figure 2**
La carte Arduino Yún

Recommandations

Arduino étant une carte permettant de réaliser toutes sortes d'expériences en y branchant des composants et des câbles d'une part, et l'erreur étant humaine d'autre part, je réclame ici toute votre attention.

La carte est directement reliée à l'ordinateur via l'interface USB. Autrement dit, une faute d'inattention comme un court-circuit sur la carte peut dans le pire des cas nuire à votre machine, notamment au port USB, et endommager la carte mère. Pour éviter cela, branchez un concentrateur USB entre l'ordinateur et la carte Arduino. Avec quatre ports, il coûte souvent moins de 10 €.

Par ailleurs, la carte Arduino présente beaucoup de contacts sur sa face inférieure ; ce sont des points de soudure par lesquels les composants sont fixés et reliés entre eux. Ils sont évidemment conducteurs et particulièrement sujets à des interconnexions indésirables. Dans le pire des cas, la loi de Murphy s'applique et vous créez un court-circuit. Je sais de quoi je parle, car j'ai déjà « fusillé » plusieurs cartes de cette manière. Aussi, tirez parti des fautes des autres pour faire mieux ! Même si bien sûr, vous avez aussi le droit à l'erreur, qui est une étape obligée dans le déroulement d'un apprentissage réussi.

Figure 3 ▶
Utilisez de préférence un
concentrateur USB pour raccorder
le microcontrôleur Arduino
à l'ordinateur.





◀ **Figure 4**
La carte Arduino Uno vue
de dessous

Si vous posez la carte sur un support métallique ou sur une table pleine de fils dénudés, c'est le court-circuit assuré. Pensez-y le jour venu, sinon je vous souhaite bien du plaisir.

J'en profite ici pour vous donner un autre conseil. Peut-être avez-vous déjà remarqué les quatre trous forés de 3 mm de diamètre sur la carte Arduino. Ils ne sont pas là pour une meilleure ventilation locale de la carte, mais pour tout autre chose. Pour que la face soudée ne repose pas directement sur le support de travail et n'entre pas, comme nous l'avons dit, en contact avec des matériaux conducteurs, vous pouvez y insérer des tampons en caoutchouc ou écarteurs pour plaques conductrices. Ils garantissent un espace de sécurité entre carte et support, protégeant ainsi de tout court-circuit.

Malgré tout, je vous recommande de demeurer prudent. Les circuits électroniques, et en particulier les circuits intégrés avec microprocesseur, sont très sensibles aux décharges électrostatiques (ESD). Marcher sur un tapis avec certaines chaussures peut ainsi charger par frottement le corps en électricité statique, et un courant très fort peut ensuite circuler brièvement au contact d'un composant électronique. Le composant est alors généralement grillé. Avant d'approcher un microcontrôleur, vous devez donc vous assurer que vous n'êtes pas chargé. Le simple fait de toucher un tuyau de chauffage à nu permet de décharger cette énergie. Restez vigilant.

Remerciements

Impossible pour moi de clore cet avant-propos sans remercier ma famille, mes amis, et d'une manière générale, tous ceux qui m'ont aidé dans la genèse de cet ouvrage. Libre à vous de sauter ce passage !



Lorsque j'ai écrit mon premier livre sur le langage de programmation Processing (paru chez O'Reilly) – j'en entends qui disent : « Voilà qu'il fait de la publicité maintenant... » –, cela m'a fait du bien et du mal à la fois. Du bien, parce que j'ai pu atteindre, de mon vivant, le but que je m'étais fixé dans la vie : écrire un manuel de programmation. Mais aussi du mal, parce que j'ai dû vivre en marge de ma famille pendant un long moment. Mais heureusement, elle l'a compris et m'a apporté de temps en temps de quoi me nourrir, si bien que je n'en garde pas d'importantes séquelles, ni physiques ni spirituelles.

Peu avant la publication de ce premier livre, j'avais confié à mon éditeur, Volker Bombien, combien j'appréciais le microcontrôleur Arduino. Ce n'est pas tombé dans l'oreille d'un sourd : il a aussitôt sauté sur l'occasion et je lui en suis très reconnaissant. L'intérêt que je portais à l'électronique dans mes jeunes années, qui stagnait jusque-là, m'a soudainement repris pour ne plus me lâcher. Il faut dire que tout ce que nous pouvons entreprendre aujourd'hui en électronique n'était encore qu'un rêve il y a trente ans.

Je remercie également ma famille, qui s'est certainement demandé : « Le voilà encore qui se coupe de nous, peut-être est-ce de notre faute ? » Un grand merci aussi au validateur technique fribbe aka Holger Lübker, qui m'avait déjà aidé sur mon livre sur le Raspberry Pi. Je remercie par ailleurs Andreas Riedenauer, de la société Inteltek, qui a relu mon manuscrit et m'a éclairé de ses lumières. Je ne saurai oublier ma correctrice Dorothée Leidig qui a débarrassé mon manuscrit de ses fautes et qui en a fait un ouvrage lisible. Un grand merci pour votre aide ! Vous êtes des travailleurs de l'ombre, un peu comme les souffleurs au théâtre. On ne vous voit jamais de visu, mais votre action se fait sentir sur la qualité de l'ouvrage. Vous êtes indispensables et incontournables !

Pour finir, je vous présente votre guide, qui se prénomme Ardu. Il sera présent à vos côtés tout au long de ce livre et posera les questions que personne n'ose poser.

Cool mec ! J'ai hâte de voir ça. On va s'en occuper, nous, du bébé...
euh... de la carte Arduino, pas vrai ?

Bien sûr, Ardus !

Il est désormais temps que je vous abandonne à votre destin et que je
me retire sur la pointe des pieds.

Amusez-vous bien avec votre carte Arduino !

Eni Bartouille



Table des matières

Partie I : Les bases

Chapitre 1 : Qu'est-ce qu'un microcontrôleur ?	3
À quoi sert-il ?	4
Structure d'un microcontrôleur	5
Chapitre 2 : La famille Arduino	11
Les différentes cartes Arduino	11
Arduino Uno	12
Arduino Leonardo	13
Arduino Mega 2560	15
Arduino Esplora	16
Boarduino V2.0	18
Arduino Nano	20
Arduino LilyPad	21
Arduino Due	22
Arduino Yún	24
Chapitre 3 : La carte Arduino	27
L'alimentation électrique	31
Les modes de communication	32
Les langages de programmation C/C++	34
Comment puis-je programmer une carte Arduino ?	36
L'environnement de développement d'Arduino	48
La communication par port	59
Ordre et obéissance	63

Chapitre 4 : Les bases de l'électronique	67
Vous avez dit électronique ?	67
Principaux composants électroniques	78
Autres composants	101
Chapitre 5 : Circuits électroniques simples	115
Les circuits résistifs	115
Les circuits capacitifs (avec condensateurs)	123
Les circuits avec transistors	125
Chapitre 6 : Fritzing	131
L'interface du logiciel	132
Le Fritzing Creator Kit	149
Chapitre 7 : L'assemblage des composants	153
Qu'est-ce qu'une carte ?	153
La plaque d'essais sans soudure (breadboard)	155
Les câbles et leurs caractéristiques	157
Les cavaliers flexibles	160
Test de continuité avec un multimètre	163
Chapitre 8 : Le matériel utile	165
Pinces diverses	165
Pince à dénuder	166
Tournevis	166
Extracteur de circuit intégré	168
Troisième main	168
Multimètre numérique	169
Oscilloscope	171
Alimentation externe	173
Gabarit de pliage pour résistances	176
Fer à souder et soudure à l'étain	178
Pompe à dessouder	179
EEBoard	180
Chapitre 9 : Les bases de la programmation	185
Qu'est-ce qu'un programme ou sketch ?	185
Que signifie traitement des données ?	187
Structure d'un sketch Arduino	204
Combien de temps dure un sketch sur la carte ?	206

Chapitre 10 : Programmation de la carte Arduino	211
Les ports numériques	211
Les ports analogiques	213
L'interface série	219

Partie II : Les montages

Montage 1 : Le premier sketch	221
Le phare "Hello World"	221
Composants nécessaires	222
Code du sketch	223
Revue de code	224
Schéma	227
Réalisation du circuit	227
Problèmes courants	229
Qu'avez-vous appris ?	229
Exercice complémentaire	230
Montage 2 : Interrogation d'un capteur	231
Appuyez sur le bouton	231
Code du sketch	232
Revue de code	233
Schéma	236
Réalisation du circuit	240
Problèmes courants	241
Autres possibilités pour des niveaux d'entrée définis	242
Qu'avez-vous appris ?	245
Exercice complémentaire	245
Montage 3 : Clignotement avec gestion des intervalles	247
Appuyez sur le bouton-poussoir et il réagit	247
Composants nécessaires	248
Code du sketch	248
Revue de code	250
Schéma	255
Réalisation du circuit	255
Problèmes courants	256
Qu'avez-vous appris ?	256
Exercice complémentaire	257

Montage 4 : Le bouton-poussoir récalcitrant	259
Une histoire de rebond	259
Composants nécessaires	261
Code du sketch	262
Revue de code	262
Schéma	265
Autres possibilités de compenser le rebond	266
Réalisation du circuit	268
Problèmes courants	268
Qu'avez-vous appris ?	269
Exercice complémentaire	269
Astuce	270
Montage 5 : Le séquenceur de lumière	271
Qu'est-ce qu'un séquenceur de lumière ?	271
Composants nécessaires	273
Code du sketch	273
Revue de code	274
Schéma	281
Réalisation du circuit	281
Problèmes courants	282
Qu'avez-vous appris ?	282
Exercice complémentaire	283
Montage 6 : Extension de port	285
Le registre à décalage	285
Composants nécessaires	288
Code du sketch	289
Revue de code	290
Schéma	293
Réalisation du circuit	294
Extension du sketch : première partie	295
Extension du sketch : deuxième partie	299
Problèmes courants	301
Qu'avez-vous appris ?	302
Exercice complémentaire	302
Montage 7 : La machine à états	305
Des feux de circulation	305
Composants nécessaires	307

Code du sketch	307
Revue de code	308
Schéma	310
Réalisation du circuit	311
Sketch élargi (circuit interactif pour feux de circulation)	311
Autre sketch élargi	318
Problèmes courants	324
Qu'avez-vous appris ?	324
Exercice complémentaire	325
Cadeau !	326

Montage 8 : Le dé électronique 327

Qu'est-ce qu'un dé électronique ?	327
Composants nécessaires	329
Code du sketch	329
Revue de code	330
Schéma	335
Réalisation du circuit	336
Que pouvons-nous encore améliorer ?	338
Problèmes courants	345
Qu'avez-vous appris ?	346
Exercice complémentaire	346

Montage 9 : Comment créer une bibliothèque ? 347

Les bibliothèques	347
Qu'est-ce qu'une bibliothèque exactement ?	348
En quoi les bibliothèques sont-elles utiles ?	349
Que signifie programmation orientée objet ?	350
Construction d'une classe	353
Une classe a besoin d'aide	355
Une classe devient un objet	356
Initialiser un objet : qu'est-ce qu'un constructeur ?	357
La surcharge	358
La bibliothèque-dé	359
Utilisation de la bibliothèque	365
Qu'avez-vous appris ?	367

Montage 10 : Des détecteurs de lumière 369

Comment fonctionne un détecteur de lumière ?	369
Composants nécessaires	370
Code du sketch	370

Revue de code	371
Schéma	372
Réalisation du circuit	375
Nous devenons communicatifs.	375
Arduino l'émetteur	377
Processing le récepteur.	378
Problèmes courants	381
Qu'avez-vous appris ?	382
Exercice complémentaire	382

Montage 11 : L'afficheur sept segments 383

Qu'est-ce qu'un afficheur sept segments ?	383
Composants nécessaires	386
Code du sketch	386
Revue de code	387
Schéma	389
Réalisation du circuit	389
Sketch amélioré	390
Problèmes courants	395
Qu'avez-vous appris ?	395
Exercice complémentaire	395

Montage 12 : Le clavier numérique 397

Qu'est-ce qu'un clavier numérique ?	397
Composants nécessaires	399
Réflexions préliminaires	399
Code du sketch	402
Schéma	409
Réalisation du shield	410
Problèmes courants	411
Qu'avez-vous appris ?	411
Exercice complémentaire	412

Montage 13 : Un afficheur alphanumérique 413

Qu'est-ce qu'un afficheur LCD ?	413
Composants nécessaires	415
Remarque préliminaire sur l'utilisation de l'afficheur LCD	415
Code du sketch	418
Revue de code	419
Schéma	421

Réalisation du circuit	422
Jeu : deviner un nombre	422
Problèmes courants.....	429
Qu'avez-vous appris ?	429
Exercice complémentaire	429

Montage 14 : Le moteur pas-à-pas..... 431

Encore plus de mouvement	431
Composants nécessaires	435
Code du sketch	436
Revue de code	437
Problèmes courants.....	440
Qu'avez-vous appris ?	441
Exercice complémentaire	441

Montage 15 : La température 443

Chaud ou froid ?	443
Comment peut-on mesurer la température ?	444
Composants nécessaires	445
Code du sketch Arduino	445
Revue de code Arduino	446
Revue de code Processing	446
Schéma	447
Sketch élargi (maintenant avec tout le reste)	448
Composants nécessaires	450
Problèmes courants.....	456
Qu'avez-vous appris ?	456
Exercice complémentaire	457

Montage 16 : Le son et plus encore..... 459

Y a pas le son ?	459
Composants nécessaires	460
Code du sketch	460
Revue de code	461
Réalisation du circuit	463
Sketch élargi : jeu de la séquence des couleurs	464
Composants nécessaires	465

Montage 17 : Communication réseau..... 473

Qu'est-ce qu'un réseau ?	473
Composants nécessaires	478

Code du sketch	480
Revue de code	482
Problèmes courants	487
Qu'avez-vous appris ?	488
Exercice complémentaire	488

Montage 18 : Numérique appelle analogique..... 489

Comment convertir des signaux numériques en signaux analogiques ?	489
Composants nécessaires	492
Réflexions préliminaires	492
Code du sketch	493
Revue de code	493
Schéma	494
Réalisation du shield	495
Commande du registre de port	495
Problèmes courants	501
Qu'avez-vous appris ?	502
Exercice complémentaire	502

Montage 19 : Interactions entre Arduino et Raspberry Pi 503

Réveillons l'Arduino sommeillant dans tout Raspberry Pi	503
Installation de l'IDE Arduino sur le Raspberry Pi	505
Firmata	506
Préparation de l'Arduino	508
Préparations du Raspberry Pi	508
Commande par MLI	512
Commande d'un servomoteur	515
Interrogation d'un bouton-poussoir	516
Interrogation d'un port analogique	517
Liaison série entre le Raspberry Pi et l'Arduino	519

Montage 20 : Temboo et la carte Yún – API Twitter..... 523

Composants nécessaires	524
Temboo	524
Votre compte Twitter	528
De retour dans Temboo	532
Au tour de la Yún	536
Qu'avez-vous appris ?	547
Exercice complémentaire	547

Montage 21 : Temboo et la carte Yún – Tableur Google	549
Composants nécessaires	550
Google Docs	550
Procédure pas à pas	551
Qu'avez-vous appris ?	559
Exercice complémentaire	559
Montage 22 : Réalisation d'un shield	561
Shield de prototypage fait maison	562
De quoi avons-nous besoin ?	562
Bon sang, rien ne va !	564
Premier exemple d'application	566
Composants nécessaires	567
Code du sketch	568
Réalisation du shield	568
Annexe : Référentiel des instructions	571
Structure d'un sketch	571
Structures de contrôle	572
Boucles	573
Constantes importantes	574
Fonctions	576
Directives de prétraitement	579
Index	581

Partie I

Les bases

Qu'est-ce qu'un microcontrôleur ?

Si vous n'y tenez plus et que vous voulez déjà brancher votre Arduino et procéder à la première expérimentation, vous pouvez sauter allègrement ce chapitre, qui décrit les principes du microcontrôleur, et y revenir peut-être plus tard.

Un microcontrôleur est un circuit intégré (ou IC, *Integrated Circuit*), qui rassemble sur une puce plusieurs éléments complexes dans un espace réduit. Au temps des pionniers de l'électronique, on soudait un grand nombre de composants encombrants, tels que les transistors, les résistances ou les condensateurs, sur des cartes plus ou moins grandes. Aujourd'hui, tout peut loger dans un petit boîtier en plastique noir muni d'un certain nombre de broches. Ces dernières sont les connexions du circuit intégré au moyen desquelles s'effectue la communication. La figure 1-1 montre un microcontrôleur ATmega328, qu'on trouve sur la carte Arduino.



◀ **Figure 1-1**
Microcontrôleur ATmega328
(source : Atmel)

Avec ses dimensions réduites, il dispose pourtant d'une grande puissance de calcul. En fait, il suffit de le souder sur une carte et de le mettre sous tension pour pouvoir l'utiliser. Certes, il manque encore quelques composants (par exemple, des stabilisateurs de tension, des

connexions pour la programmation, et d'autres sur lesquels nous reviendrons plus tard), mais il est cependant sous cette forme déjà (presque) prêt à l'emploi.

À quoi sert-il ?

Maintenant, vous vous demandez peut-être à quoi sert un microcontrôleur et ce qu'on peut faire avec. À cela, je peux répondre que les possibilités sont innombrables et dépendent uniquement de votre créativité.

Les microcontrôleurs jouent un rôle prépondérant dans les domaines suivants – cette liste est loin d'être exhaustive et sert surtout à se faire une idée des diverses possibilités d'utilisation.

- Fonctions de surveillance dans des environnements critiques, par exemple dans des cages thoraciques (température, humidité, fréquence cardiaque, pression sanguine du prématuré...).
- Commande de chauffage : contrôle de la température externe ou interne pour le chauffage optimal de locaux.
- Stimulateurs cardiaques : surveillance de la fréquence cardiaque et, le cas échéant, stimulation du cœur.
- Appareils ménagers : par exemple, commande par programme enregistré dans des lave-linge ou lave-vaisselle modernes.
- Électronique de loisirs : lecteurs MP3, téléphones portables, appareils photo...
- Robotique : par exemple, commande de robots industriels pour le montage de pièces automobiles.

Cette liste peut ainsi se poursuivre à l'infini, mais nous pouvons d'ores et déjà remarquer une chose : les microcontrôleurs perçoivent des influences extérieures par le biais de capteurs, les traitent en interne à l'aide d'un programme, puis envoient des ordres de commande correspondants vers l'extérieur. Ils font donc preuve d'une certaine intelligence, qui dépend bien évidemment du programme mis en œuvre. Un microcontrôleur peut assurer des fonctions de mesure, de commande et de régulation.

Regardons maintenant de plus près le fonctionnement d'une boucle de régulation. Elle se compose d'un processus en boucle fermée comportant une perturbation. Un capteur transmet cette dernière au microcontrôleur qui réagit alors en fonction de son programme.

Imaginez le scénario suivant. Vous vous trouvez au sein du système de contrôle de chauffage qui régule la température de votre local de travail. Le capteur dit au microcontrôleur : « Dis donc, il fait plutôt chaud dans le local de travail ! » Le microcontrôleur réagit alors en régulant la température. Le chauffage apporte donc moins d'énergie sous forme de chaleur dans le local. Le capteur le remarque et dit au microcontrôleur : « La température est maintenant celle souhaitée, soit 20 °C. » De l'air froid provenant de l'extérieur rentre petit à petit. Le capteur donne l'alerte et dit au microcontrôleur : « Il commence à faire froid ici et mon bonhomme va tomber malade, il faut faire quelque chose ! » Le microcontrôleur augmente la température en conséquence. Vous voyez, c'est un jeu de ping-pong : ici, en l'occurrence, une boucle de régulation qui réagit à des influences perturbatrices extérieures liées à des variations de température.

Structure d'un microcontrôleur

Venons-en maintenant à la structure générale d'un microcontrôleur et regardons les différents composants de la puce.

Stop, j'ai déjà une question ! Vous nous avez dit tout à l'heure que le microcontrôleur était déjà prêt à l'emploi. Mais où se trouve son programme et où stocke-t-il ses données ? Vous avez sûrement oublié de parler des modules de stockage qui doivent encore être raccordés.



Bonne question, mais vous ne savez pas encore tout sur notre microcontrôleur ! Pris comme tel – et c'est ce que nous faisons –, c'est un ordinateur complet sur un espace réduit au maximum, avec donc les éléments suivants :

- unité centrale (CPU) ;
- mémoire de travail ;
- mémoire de données ;
- horloge interne ;
- ports d'entrée et de sortie.

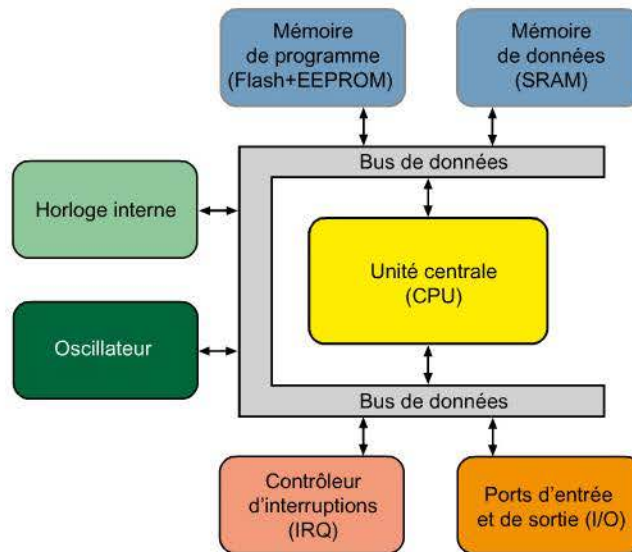
Un microcontrôleur se divise grossièrement en trois parties :

- unité centrale (CPU) ;
- mémoires (ROM et RAM) ;
- ports d'entrée et de sortie.

L'horloge interne, ou l'oscillateur qui permet de piloter l'unité centrale, a été laissée de côté pour le moment. Les éléments qui composent un microcontrôleur sont comparables aux périphériques d'un ordinateur. La différence réside dans le fait que les trois parties citées précédemment sont intégrées au microcontrôleur. Elles se trouvent toutes dans le même boîtier, ce qui est plus simple et plus compact.

Jetons maintenant un coup d'œil au schéma fonctionnel de notre microcontrôleur.

Figure 1-2 ►
Schéma fonctionnel
d'un microcontrôleur



Vous vous demandez maintenant ce que signifient les différents blocs dans le schéma et quelle est leur fonction exacte, pas vrai ?

L'unité centrale (CPU)

L'élément le plus important dans un microcontrôleur est l'unité centrale, appelée également CPU (*Central Processing Unit*). Sa fonction principale consiste à décoder et à exécuter des instructions. Elle peut adresser des mémoires, gérer des entrées ou sorties et réagir à des interruptions (*interrupts*). Une interruption (IRQ, ou *Interrupt Request*) est un signal qui demande au CPU d'interrompre un cycle de calcul en cours pour pouvoir réagir à un certain événement.

Le bus de données

Le bus de données sert à transporter les données d'un bloc à un autre. Par exemple, le CPU demande des données provenant de la mémoire, qui sont prises en charge par le bus et immédiatement mises à disposi-

tion pour traitement. Lorsque le résultat du calcul est disponible, il est à nouveau transféré sur le bus et transmis à un port de sortie qui, par exemple, pilote un moteur de robot pour atteindre un but précis. Cette structure de bus est une autoroute de données utilisable en commun par tous ceux qui sont desservis.



◀ **Figure 1-3**

Sur l'autoroute des données :
« Prochain arrêt, mémoire ! »

Les zones de mémoire

En principe, il existe deux types de mémoires d'un microcontrôleur :

- la mémoire de programme ;
- la mémoire de données.

La première accueille le programme que le CPU doit exploiter, alors que la seconde est utilisée pour gérer les résultats de calcul du moment.

Il y a un problème quelque part. Quand j'éteins mon ordinateur, tous les programmes qui se trouvent dans la mémoire s'effacent et je dois les recharger depuis mon disque dur pour pouvoir travailler avec.

C'est vrai et c'est en cela que la mémoire de programme d'un microcontrôleur est particulière. Un microcontrôleur n'a bien sûr pas de disque dur, mais il garde son programme en mémoire en l'absence de tension d'alimentation externe. Un type de mémoire particulier est utilisé à cet effet, qu'on appelle mémoire flash. Comme son nom l'indique, c'est une mémoire non volatile, c'est-à-dire que bits et octets ne « s'envolent » pas en cas de coupure d'alimentation et restent disponibles.

Vous avez déjà utilisé cette forme de mémoire des milliers de fois sur votre ordinateur. Le BIOS est hébergé dans une mémoire flash de



type EEPROM, et ses données peuvent être écrasées au besoin par l'insertion d'une nouvelle version. On dit alors que le BIOS est « reflashé ». Le contraire se produit dans les mémoires de données dites SRAM. Ces dispositifs sont volatils et les données mémorisées sont perdues dès qu'il y a coupure d'alimentation. Mais rien de grave puisque ces dernières ne sont nécessaires que lorsque le programme est exécuté. Quand le microcontrôleur est sans courant, il n'a rien besoin de calculer. Mais la mémoire SRAM a sur la mémoire flash un avantage très important : elle offre un accès plus rapide.

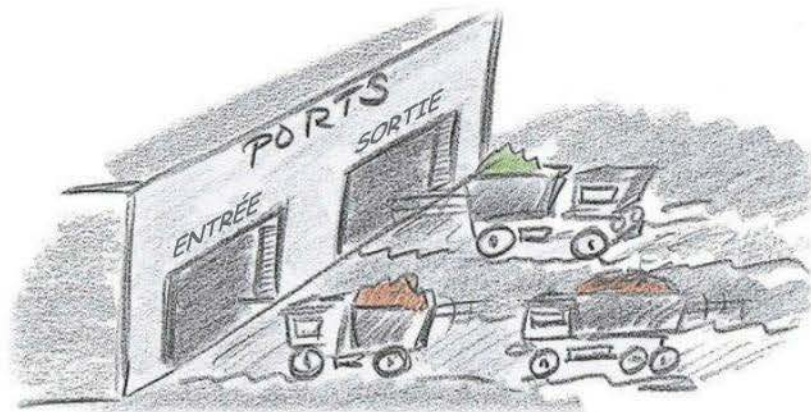
Les ports d'entrées-sorties

Les ports d'entrées-sorties sont les connexions qui relient le microcontrôleur au monde extérieur. Ils constituent une interface à laquelle la périphérie peut être connectée. On entend par périphérie tout ce qui peut être avantageusement raccordé à l'interface. Il peut s'agir, par exemple, des composants électroniques ou électriques suivants :

- LED (diode électroluminescente) ;
- bouton-poussoir ;
- commutateur ;
- LDR (*Light Dependant Resistor* ou photorésistance) ;
- transistor ;
- résistance ;
- haut-parleur ou élément piézoélectrique ;
- etc.

Nous reviendrons sur ces éléments et aussi sur leur raccordement aux différents ports.

Figure 1-4 ►
Ports d'entrée et de sortie



En principe, les ports d'entrées-sorties sont soit numériques, soit analogiques – nous verrons plus loin ce qui les différencie.

Le contrôleur d'interruption

Un microcontrôleur est équipé d'un contrôleur dit d'interruption. Qu'est-ce que c'est et à quoi sert-il ? Imaginez le scénario suivant.

Vous allez vous coucher le soir et vous voulez vous lever le lendemain à 6 h 00 pour avoir le temps de vous laver, de prendre votre petit déjeuner et d'arriver à l'heure à votre travail. Mais comment faire ? Deux possibilités s'offrent à vous, qui donnent des résultats différents.

- **Cas n° 1.** Vous réglez votre réveil sur 6 h 00 avant d'aller vous coucher. Vous pouvez ainsi dormir tranquille, sans la crainte d'oublier de vous réveiller le lendemain. Le réveil se déclenche à l'heure souhaitée, et vous arrivez frais et dispo sur votre lieu de travail.
- **Cas n° 2.** Vous allez vous coucher et, comme vous n'avez pas de réveil, vous vous relevez toutes les 30 minutes pour vérifier l'heure qu'il est pour ne pas rater le moment de vous lever. À 6 h 00 du matin, vous êtes crevé et incapable de travailler car vous avez passé votre nuit à vous réveiller.

Je pense que pour tout le monde le choix est vite fait : le cas n° 1 est préférable et plus économe en ressources.

Transposons maintenant cet exemple à notre microcontrôleur. Un commutateur surveillant l'état d'une vanne est raccordé au port d'entrée numérique. Notre microcontrôleur pourrait ainsi être programmé pour interroger l'état du commutateur à intervalles brefs et réguliers. Cette interrogation cyclique appelée *polling* (ou interrogation) est, dans ce cas, plutôt inefficace car l'unité centrale est sollicitée pour rien. Une surveillance par interruption s'avère ici bien plus avantageuse. L'unité centrale suit le cours normal de son programme et ne réagit que si une interruption se produit. Le travail de fond s'interrompt un court instant et bascule dans une routine d'interruption (ISR, ou *Interrupt Service Routine*). Celle-ci contient des instructions qui indiquent l'action à effectuer. Après cela, on revient au travail de fond et à l'endroit précis où l'interruption s'est produite, comme si rien ne s'était passé.



Arduino est-il un microcontrôleur ?

Ce chapitre est consacré aux principes généraux d'un microcontrôleur. Vous en connaissez maintenant les principaux composants (unité centrale, mémoires, ports) et vous savez à quoi ils servent. Mais vous êtes désormais en droit de vous poser la question suivante : « Arduino est-il un microcontrôleur à proprement parler ? » La réponse est oui, sans aucun doute ! Il possède bien tous les composants dont nous avons parlé et les réunit en son sein. Mais il cohabite aussi avec d'autres composants sur une carte compacte, dont nous allons parler dans le prochain chapitre.



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- microcontrôleur ;
- microcontrôleur AVR ;
- Atmel.

La famille Arduino

Les différentes cartes Arduino

Les cartes Arduino doivent satisfaire des exigences diverses et variées. Certains utilisateurs souhaiteront effectuer du prototypage et tester de nouveaux montages ou idées, la carte Arduino Uno disposant d'assez d'entrées-sorties pour les projets d'envergure raisonnable. De part sa taille, la carte devrait aussi pouvoir être utilisée à l'avenir dans des projets plus ambitieux. D'autres auront besoin d'un grand nombre de ports afin de pouvoir raccorder de nombreux capteurs ou actionneurs. Une troisième catégorie d'utilisateurs ne cherchera qu'à attirer l'attention et à transmettre des signaux à leurs semblables à l'aide de diodes clignotantes. Ces exemples ne constituent qu'une partie des attentes que les *makers* auront vis-à-vis de leur carte Arduino.

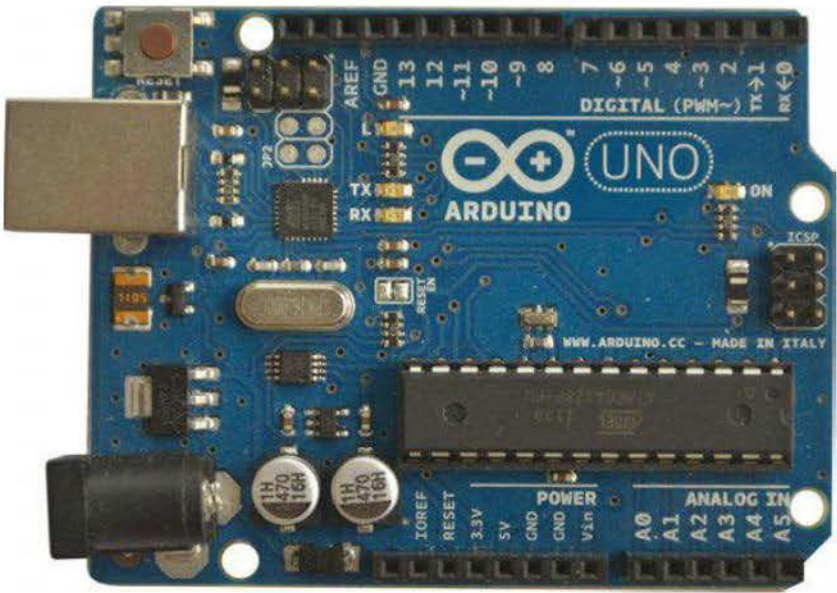
La forme, la taille ou les possibilités de connexion jouent un rôle décisif dans le choix de la carte adaptée. C'est pourquoi les développeurs d'Arduino ont mis au point un vaste choix de cartes à micro-contrôleur afin que chacun trouve le modèle qui réponde à ses besoins.

Dans ce chapitre, nous allons donc passer en revue les principaux membres de la famille Arduino. Même si nous ne détaillerons pas tous les modèles, cette présentation vous sera certainement utile lorsque vous devrez choisir votre carte tout en ménageant votre portefeuille. Pourquoi se ruiner en achetant la carte originale quand de nombreux fabricants proposent des clones dont les fonctionnalités sont similaires ? Je me limiterai (le plus souvent) aux modèles d'origine, mais libre à vous de choisir une autre voie. Je vais commencer ce petit tour d'horizon par la pièce maîtresse.

Arduino Uno

À mes yeux, la carte indispensable pour débiter demeure la carte Arduino Uno dont le prix est très abordable. Elle s’est hissée au rang de standard de fait et elle convient parfaitement à tous ceux qui veulent faire leurs premières armes avec un microcontrôleur. C’est une valeur sûre.

Figure 2-1 ►
La carte Arduino Uno



Voici un résumé des spécifications de la carte Arduino Uno :

Tableau 2-1 ►
La carte Arduino Uno

Catégorie	Valeur
Microcontrôleur	ATmega 328
Fréquence d'horloge	16 MHz
Tension de service	5 V
Tension d'entrée (recommandée)	7–12 V
Tension d'entrée (limites)	6-20 V
Ports numériques	14 entrées et sorties (6 sorties commutables en MLI)
Ports-analogiques	6 entrées analogiques
Courant maxi. par broche d'E/S (c.c.)	40 mA
Courant maxi. par broche 3,3 V	50 mA
Mémoire	32 Ko Flash, 2 Ko SRAM, 1 Ko EEPROM
Chargeur d'amorçage	0,5 Ko (en mémoire Flash)
Interface	USB
Dimensions	6,86 cm × 5,3 cm
Prix (approximatif)	24 €

Examinons maintenant les avantages et les inconvénients de la carte :

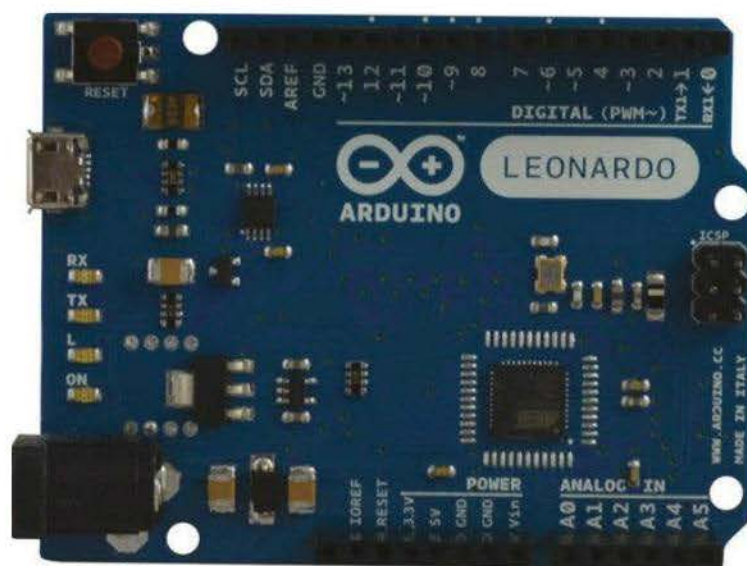
- Avantages
 - carte Arduino par excellence pour laquelle de nombreux exemples de montage sont disponibles sur Internet ;
 - nombre suffisant de broches d'entrées-sorties pour les projets élémentaires ;
 - vaste choix de *shields* ;
 - bon marché
- Inconvénients
 - nombre insuffisant de broches d'entrées-sorties pour les projets ambitieux ;
 - la mémoire disponible risque d'être un peu juste pour les gros projets ;
 - ne peut pas être utilisée comme hôte USB pour simuler un clavier ou une souris, par exemple (voir Arduino Leonardo).

Vous trouverez de plus amples informations à la page <http://arduino.cc/en/Main/arduinoBoardUno>.

Arduino Leonardo

La carte Arduino Leonardo est considérée comme le successeur officiel de l'Arduino Uno. Elle est équipée du microcontrôleur ATmega 32U4 qui communique aussi via l'interface USB, de sorte qu'un processeur supplémentaire est inutile. La carte peut être programmée en tant qu'hôte USB pour lequel différentes classes de clavier et souris sont disponibles. D'ailleurs, la carte Arduino Leonardo peut aussi être programmée en tant que HID (*Human Interface Device*). En plus du port matériel UART, auquel le microcontrôleur a accès depuis la classe *Serial1*, un port COM virtuel (**VCP** : **Virtual COM-Port**) est accessible via USB et peut être sollicité par le biais de *Serial*.

Figure 2-2 ▶
La carte Arduino Leonardo



Voici un résumé des spécifications de la carte Arduino Leonardo :

Tableau 2-2 ▶
La carte Arduino Leonardo

Catégorie	Valeur
Microcontrôleur	ATmega 32U4
Fréquence d'horloge	16 MHz
Tension de service	5 V
Tension d'entrée (recommandée)	7-12 V
Tension d'entrée (limites)	6-20 V
Ports numériques	20 entrées et sorties (7 sorties commutables en MLI)
Ports analogiques	12 entrées analogiques
Courant maxi. par broche d'E/S (c.c.)	40 mA
Courant maxi. par broche 3,3 V	50 mA
Mémoire	32 Ko Flash, 2,5 Ko SRAM, 1 Ko EEPROM
Chargeur d'amorçage	4 Ko (en mémoire Flash)
Interface	USB
Dimensions	6,86 cm × 5,3 cm
Prix (approximatif)	20 €

Examinons les avantages et les inconvénients de la carte :

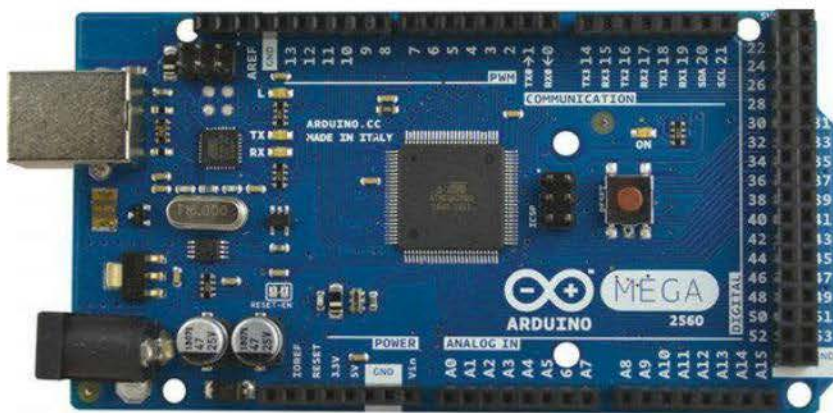
- Avantages
 - port COM virtuel supplémentaire ;
 - broche MLI de plus que sur l'Arduino Uno ;
 - 6 entrées analogiques supplémentaires par rapport à la carte Arduino Uno, sur les broches numériques 4, 6, 8, 9, 10 et 12 ;

- 6 ports numériques supplémentaires par rapport à la carte Arduino Uno, sur les broches analogiques A0 à A5 ;
- meilleur marché que l'Arduino Uno.
- Inconvénients
 - les entrées analogiques supplémentaires sont bloquées en cas d'utilisation des broches numériques, car elles sont utilisées au même moment ;
 - les entrées numériques supplémentaires sont bloquées en cas d'utilisation des broches analogiques, car elles sont utilisées au même moment ;

Vous trouverez de plus amples informations à la page <http://arduino.cc/en/Main/ArduinoBoardLeonardo>.

Arduino Mega 2560

J'ai mentionné plus haut que le nombre de broches d'E/S pouvait être insuffisant pour certains projets. Pour y remédier, vous pouvez envisager de passer à la carte Arduino Mega 2560. Vous ne manquerez pas de remarquer sur la figure 2-3 les nombreuses rangées de prises qui viennent compléter les ports d'entrée-sortie. La carte est inévitablement plus grande que l'Arduino Uno.



◀ **Figure 2-3**
La carte Arduino Mega 2560

Voici un résumé des spécifications de la carte Arduino Mega 2560 :

Catégorie	Valeur
Microcontrôleur	ATmega 2560
Fréquence d'horloge	16 MHz
Tension de service	5 V
Tension d'entrée (recommandée)	7-12 V
Tension d'entrée (limites)	6-20 V

◀ **Tableau 2-3**
La carte Arduino Mega 2560

Tableau 2-3 (suite) ►
La carte Arduino Mega 2560

Ports numériques	54 entrées et sorties (15 sorties commutables en MLI)
Ports analogiques	16 entrées analogiques
Courant maxi. par broche d'E/S (c.c.)	40 mA
Courant maxi. par broche 3,3 V	50 mA
Mémoire	256 Ko Flash, 8 Ko SRAM, 4 Ko EEPROM
Chargeur d'amorçage	8 Ko (en mémoire Flash)
Interface	USB
Dimensions	10,16 cm × 5,3 cm
Prix (approximatif)	47 €

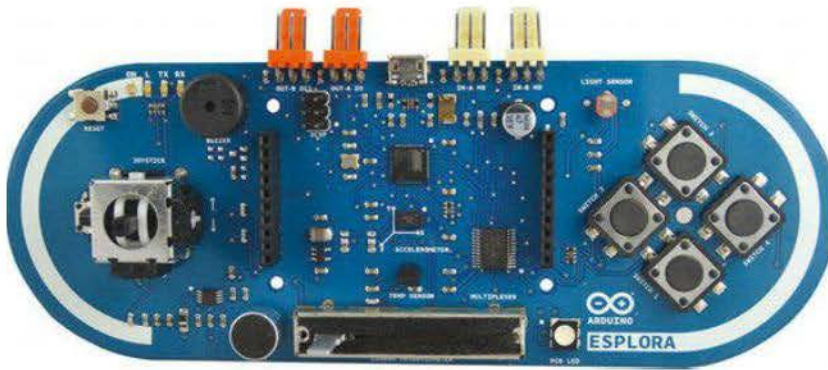
Examinons maintenant les avantages et les inconvénients de la carte :

- Avantages
 - nombreuses entrées et sorties pour raccorder des capteurs ou des actionneurs ;
 - capacité de mémoire suffisante pour les gros projets ;
 - plus de broches UART (4 ports de communication série) ;
 - plus de broches MLI (15 sorties numériques peuvent être utilisées comme MLI) ;
 - compatible avec la plupart des shields conçus pour l'Arduino Uno, par exemple ;
 - il existe des shields spéciaux pour le prototypage qui, en raison de leur surface supérieure, peuvent recevoir plus de composants ;
 - de nombreux schémas et exemples sont disponibles sur Internet.
- Inconvénients
 - facteur de forme plus élevé que pour l'Arduino Uno, par exemple ;
 - deux fois plus chère que l'Arduino Uno.

Vous trouverez de plus amples informations à la page <http://arduino.cc/en/Main/arduinoBoardMega2560>.

Arduino Esplora

L'Arduino Esplora est une carte sur laquelle est basée l'Arduino Leonardo. Si vous l'examinez de plus près, vous constaterez qu'elle est dotée d'un certain nombre de capteurs, qui ne sont pas présents sur la carte Arduino Uno ni sur d'autres cartes.



◀ **Figure 2-4**
La carte Arduino Esplora

Les capteurs suivants sont préinstallés :

- un joystick avec un bouton-poussoir central ;
- un microphone ;
- un potentiomètre (linéaire) ;
- un capteur de température ;
- un accéléromètre 3 axes (x, y et z) ;
- quatre boutons-poussoir disposés en diamant ;
- un capteur de lumière.

La carte comporte aussi les sorties suivantes :

- un buzzer ;
- une LED RVB.

Comme vous pouvez le remarquer, cette carte est donc équipée de nombreux capteurs que vous devrez acheter en plus si vous choisissez un autre modèle Arduino. Sur le bord supérieur, vous trouverez également :

- deux entrées Tinker pour relier des modules de capteurs Tinker-Kit à l'aide des connecteurs 3 broches (broches jaunes)
- deux sorties Tinker pour relier des modules d'actionneurs Tinker-Kit à l'aide des connecteurs 3 broches (broches oranges)
- un connecteur pour écran couleur TFT avec lecteur de carte SD utilisant le protocole SPI

Voici un résumé des spécifications de la carte Arduino Esplora :

Catégorie	Valeur
Microcontrôleur	ATmega 32U4
Fréquence d'horloge	16 MHz
Tension de service	5 V

◀ **Tableau 2-4**
La carte Arduino Esplora

Tableau 2-4 (suite) ►
La carte Arduino Esplora

Mémoire	32 Ko Flash, 2,5 Ko SRAM, 1 Ko EEPROM
Chargeur d'amorçage	4 Ko (en mémoire Flash)
Interface	USB
Dimensions	16,51 cm × 6 cm
Prix (approximatif)	50 €

Examinons maintenant les avantages et les inconvénients de la carte :

- Avantages
 - nombreux capteurs préinstallés ;
 - connecteurs pour modules TinkerKit.
- Inconvénients
 - il n'est pas possible d'utiliser des shields ;
 - possibilités d'extension limitées ;
 - relativement peu de circuits ou d'exemples de montages disponibles sur Internet (par rapport aux cartes Arduino Uno ou Arduino Mega 2560) ;
 - prix assez élevé.

Vous trouverez de plus amples informations à la page <http://arduino.cc/en/Guide/ArduinoEsplora>.

Boarduino V2.0

Si vous envisagez d'acheter une carte Boarduino, sachez que vous devez avoir un fer à souder sous la main, car elle est livrée en pièces détachées à assembler soi-même, à savoir :

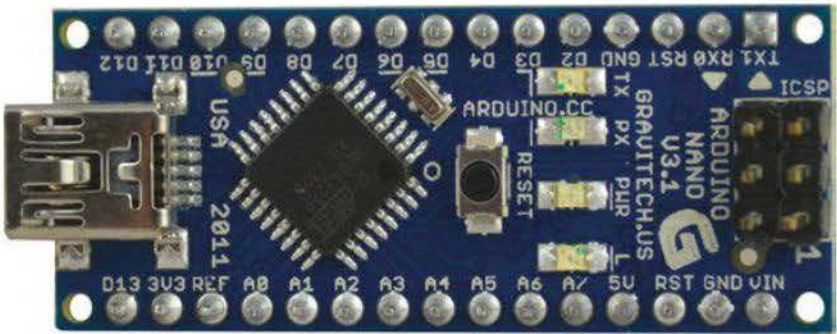
- un microcontrôleur ;
- un support de circuit à 28 broches ;
- une carte ;
- des connecteurs.

J'avais annoncé que je ne présenterai que des modèles Arduino originaux, mais je vais tout de même faire une exception. Comme la plupart des clones, celui-ci est compatible avec Arduino. Il a été conçu pour être monté sur une plaque d'essais sans soudure. Vous trouverez des notes d'assemblage détaillées à la page <http://learn.adafruit.com/boarduino-kits/usb-boarduino-assembly>.

Arduino Nano

La carte Arduino Nano possède des connecteurs au dos qui permettent de l'enficher facilement sur une plaque d'essais, ce qui évite d'avoir recours à des cavaliers flexibles, comme pour l'Arduino Uno. Ne vous laissez pas abuser par les dimensions de cette minicarte dont les performances n'ont (presque) rien à envier à l'Arduino Uno.

Figure 2-6 ►
La carte Arduino Nano



Voici un résumé des spécifications de la carte Arduino Nano :

Tableau 2-6 ►
La carte Arduino Nano

Catégorie	Valeur
Microcontrôleur	ATmega 168 ou 328
Fréquence d'horloge	16 MHz
Tension de service	5 V
Tension d'entrée (recommandée)	7-12 V
Tension d'entrée (limites)	6-20 V
Ports numériques	14 entrées et sorties (6 sorties commutables en MLI)
Ports analogiques	8 entrées analogiques
Courant maxi. par broche d'E/S (c.c.)	40 mA
Mémoire	ATmega 168 : 16 Ko mémoire Flash 1 Ko SRAM 512 octets d'EEPROM ATmega 328 : 32 Ko mémoire Flash 2 Ko SRAM 1 Ko EEPROM
Chargeur d'amorçage	2 Ko (en mémoire Flash)
Interface	USB
Dimensions	1,9 cm × 4,3 cm
Prix (approximatif)	40 €

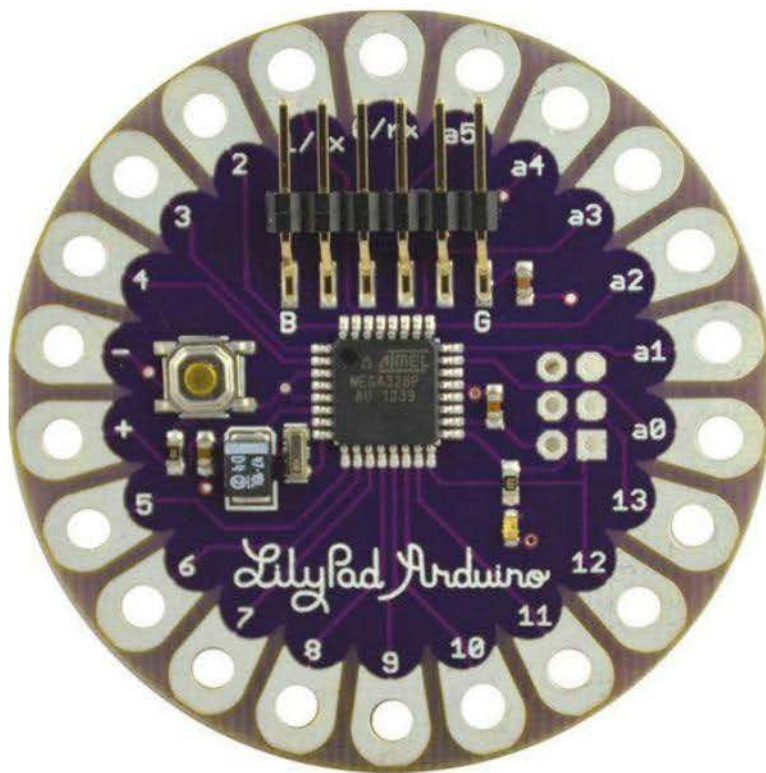
Examinons maintenant les avantages et les inconvénients de la carte :

- Avantages
 - encombrement réduit ;
 - peut être enfichée directement sur la plaque d'essais.
- Inconvénients
 - il n'est pas possible d'utiliser des shields.

Vous trouverez de plus amples informations à la page <http://arduino.cc/en/Main/ArduinoBoardNano>.

Arduino LilyPad

La plateforme Arduino LilyPad est destinée aux plus créatifs d'entre nous qui veulent coudre des circuits électroniques sur leurs vêtements, par exemple. Les raccordements électriques ne se font pas par câbles, mais par des fils conducteurs. D'après le fabricant, le composant est lavable. Toutefois, je vous déconseille de le passer en machine. Lavez-le délicatement à la main avec un détergent doux. Autre précaution : pensez bien à débrancher l'alimentation électrique avant le nettoyage !



◀ **Figure 2-7**
La carte Arduino LilyPad

Voici un résumé des spécifications de la carte Arduino LilyPad :

Tableau 2-7 ►
La carte Arduino LilyPad

Catégorie	Valeur
Microcontrôleur	ATmega 168 V ou 328 V
Fréquence d'horloge	8 MHz
Tension de service	2,7-5,5 V
Tension d'entrée (recommandée)	2,7-5,5 V
Ports numériques	14 entrées et sorties (6 sorties commutables en MLI)
Ports analogiques	6 entrées analogiques
Courant maxi. par broche d'E/S (c.c.)	40 mA
Mémoire	16 Ko Flash, 1 Ko SRAM, 512 octets EEPROM
Chargeur d'amorçage	2 Ko (en mémoire Flash)
Dimensions	5 cm de diamètre
Prix (approximatif)	20 €

Examinons maintenant les avantages et les inconvénients de la carte :

- Avantages
 - *fringable* : carte extraplate conçue pour être intégrée à des vêtements.
- Inconvénients
 - la programmation ne se fait pas par USB, mais par un module adaptateur FTDI (5 V). Vous trouverez plus d'infos sur ce module en saisissant les critères de recherche suivants : *LilyPad FTDI Basic Breakout - 5V DEV-10275*
 - Comme les raccordements électriques peuvent uniquement être soudés sur la plateforme LilyPad, cette carte se prête davantage aux montages qui ne nécessitent pas d'ajustements fréquents.

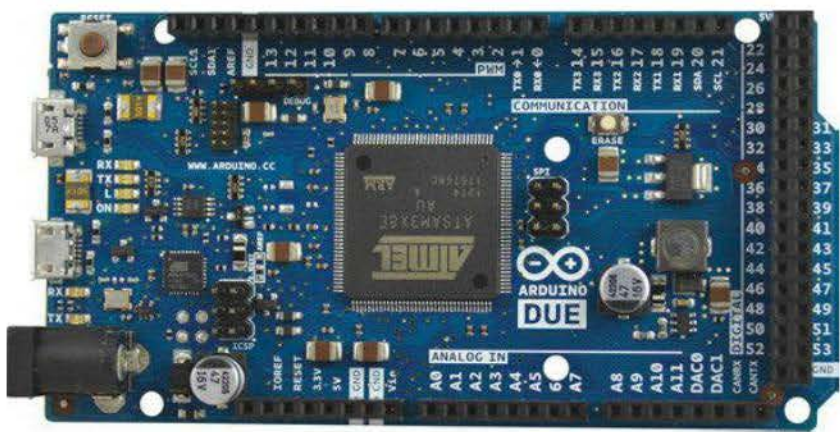
Vous trouverez de plus amples informations à la page <http://arduino.cc/en/Main/ArduinoBoardLilyPad>.

Arduino Due

L'Arduino Due est la première carte Arduino équipée d'un processeur 32 bits. La fréquence d'horloge de 84 MHz permet de réaliser des calculs complexes en un temps record. De plus, les programmes les plus lourds disposent désormais d'une capacité de mémoire suffisante, ce qui évite d'avoir à faire attention au moindre octet, comme sur une Arduino Uno. Comme toutes les cartes Arduino antérieures utilisent une tension d'entrée de 5 V, des problèmes risquent de se poser si l'on ne sait pas que les entrées de la carte Arduino Due sont limitées à une tension de 3,3 V. La carte risque d'être irrémédiablement détruite.

Au lieu de la résolution 8 bits habituelle (*analogWrite(0...255)*), les 12 broches numériques, qui peuvent être utilisées comme sorties MLI, ont désormais une résolution étendue à 12 bits (*analogWrite(0...4095)*), qui peut être changée par *analogWriteResolution(12)*.

Encore quelques mots sur les protocoles des interfaces. En plus d’I²C, TWI (*Two-Wire-Interface*) et SPI (*Serial-Peripheral-Interface*), on trouve aussi le bus CAN (*Controller Area Network*) qui est utilisé aussi bien dans l’automatisation de maquettes de chemin de fer que pour la mise en réseau de systèmes divers ou d’organes de commande dans les automobiles. L’interface USB OTG permet aussi de raccorder une souris, un clavier ou un smartphone.



◀ **Figure 2-8**
La carte Arduino Due

Voici un résumé des spécifications de la carte Arduino Due :

Catégorie	Valeur
Microcontrôleur	ATmega SAM3X8E basé sur une architecture ARM Cortex M3 à 32 bits
Fréquence d'horloge	84 MHz
Tension de service	3,3 V (attention, ce n'est pas du 5 V !)
Tension d'entrée (recommandée)	7-12 V
Tension d'entrée (limite)	6-20 V
Ports numériques	54 entrées et sorties (12 sorties commutables en MLI)
Ports analogiques	12 entrées analogiques
Courant maxi. par broche 3,3 V (c.c.)	800 mA
Courant maxi. par broche 5 V (c.c.)	800 mA
Mémoire	512 Ko Flash, 94 Ko SRAM (2 bancs : 64 Ko + 32 Ko), 512 octets EEPROM
Dimensions	10,2 cm × 5,3 cm
Prix (approximatif)	47 €

◀ **Tableau 2-8**
La carte Arduino Due

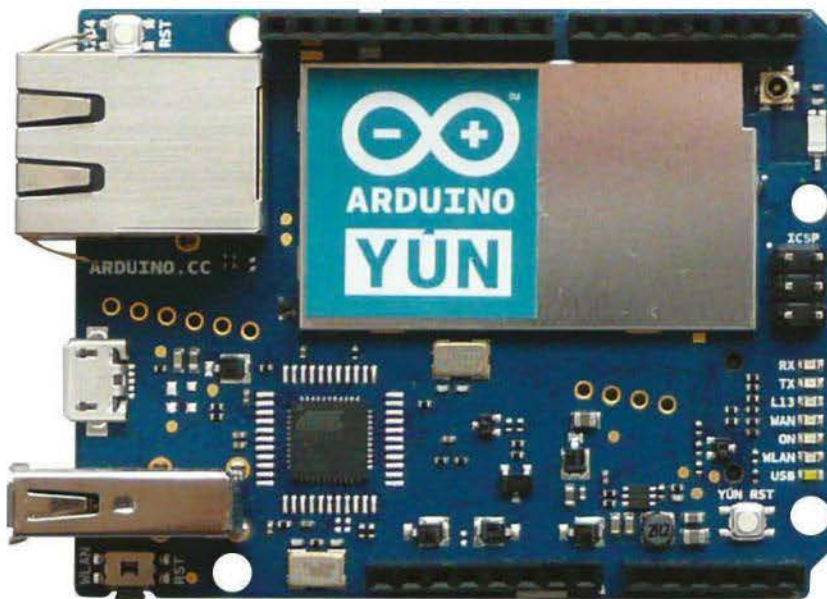
Examinons maintenant les avantages et les inconvénients de la carte :

- Avantages
 - processeur ARM 32 bits ;
 - nombreuses entrées et sorties pour raccorder des capteurs ou des actionneurs ;
 - broches UART étendues (4 ports de communication série matériels) ;
 - connexion USB-OTG (*On-The-Go*) ;
 - deux CNA (convertisseur numérique-analogique) qui peuvent être utilisés pour générer des signaux audio, par exemple ;
 - deux bus TWI (*Two Wire Interface*) ;
 - peut recevoir des shields Arduino à condition qu'ils fonctionnent en 3,3 V et qu'ils soient conformes avec le brochage de l'Arduino 1.0 (à vérifier absolument).
- Inconvénients
 - Tension de fonctionnement de 3,3 V !
 - Ne pas utiliser de shields Arduino qui fonctionnent en 5 V.

Vous trouverez de plus amples informations à la page <http://arduino.cc/de/Main/ArduinoBoardDue>.

Arduino Yún

La carte Arduino Yún est la première à être équipée de deux processeurs. Du côté Arduino, il y a un microcontrôleur de type *ATmega 32U4* qui se charge de l'exécution des sketches. De l'autre, il y a la machine Linux. Eh oui, vous avez bien lu : c'est un processeur du type *Atheros AR9331* fonctionnant sous *Linino*, une distribution Linux basée sur *OpenWRT*. Les deux couches – comme je me plais à les appeler – sont interconnectées par un *bridge* afin d'échanger des informations ou des données. Ce bridge est un logiciel qui peut être utilisé par les deux processeurs.



◀ **Figure 2-9**
La carte Arduino Yún

Voici un résumé des spécifications de la carte Arduino Yún :

Microcontrôleur AVR :

Catégorie	Valeur
Microcontrôleur	ATmega 32U4
Fréquence d'horloge	16 MHz
Tension de service	5 V
Tension d'entrée	5 V
Ports numériques	20 entrées et sorties (7 sorties commutables en MLI)
Ports analogiques	12 entrées analogiques
Courant maxi. par broche 3,3 V (c.c.)	50 mA
Courant maxi. par broche 5 V (c.c.)	40 mA
Mémoire	32 Ko Flash, 2,5 Ko SRAM, 1 Ko EEPROM
Dimensions	7 cm × 5,3 cm
Prix (approximatif)	62 €

◀ **Tableau 2-9**
La carte Arduino Yún
(microcontrôleur AVR)

Processeur Linux :

Catégorie	Valeur
Processeur	Atheros AR9331
Architecture/fréquence d'horloge	MIPS @400MHz
Tension de service	3,3 V
Ethernet	IEEE 802.3 10/100 Mbit/s

◀ **Tableau 2-10**
La carte Arduino Yún
(processeur Linux)

Tableau 2-10 (suite) ►
La carte Arduino Yún
(processeur Linux)

Wi-Fi	EEE 802.11b/g/n
USB Type-A	2.0 Host/Device
Lecteur de cartes	Pour Micro-SD uniquement
Mémoire	64 Mo DDR2 RAM, 16 Mo Flash

Examinons maintenant les avantages et les inconvénients de la carte :

- Avantages
 - combinaison d’une carte Arduino standard basée sur une carte Leonardo avec un microprocesseur Linux (système d’exploitation Linino basé sur OpenWRT) ;
 - module Wi-Fi préinstallé ;
 - interface Ethernet préinstallée ;
 - les environnements Arduino et Linux peuvent échanger des données ou des informations via un bridge ;
 - nombreuses possibilités d’extension par le biais d’API préprogrammées disponibles gratuitement sur le site Temboo (<https://temboo.com/>). Voir le montage n° 20. Il s’agit notamment de collections de services web, comme Twitter ou Google+, facilement accessibles via une couche d’abstraction normalisée à l’aide de fonctions homogènes. Les bibliothèques disponibles réunissent plus d’une centaine d’API ;
 - possibilités d’extension logicielle par carte micro SD.
- Inconvénients
 - plus énergivore de sorte que le port USB 2.0 peut vite atteindre ses limites.

Vous trouverez de plus amples informations à la page <http://arduino.cc/en/Main/ArduinoBoardYun>.

La carte Arduino

Je vais commencer par vous présenter la pièce maîtresse de tous les montages de ce livre : la carte à microcontrôleur Arduino.



◀ **Figure 3-1**
La carte à microcontrôleur Arduino

Sur cette image, vous ne pouvez évidemment pas vous rendre compte à quel point les dimensions de la carte Arduino sont réduites (environ 7 cm de large et 5 cm de long) ; elle est vraiment très maniable, tient sans problème dans une main et s'avère vraiment compacte.

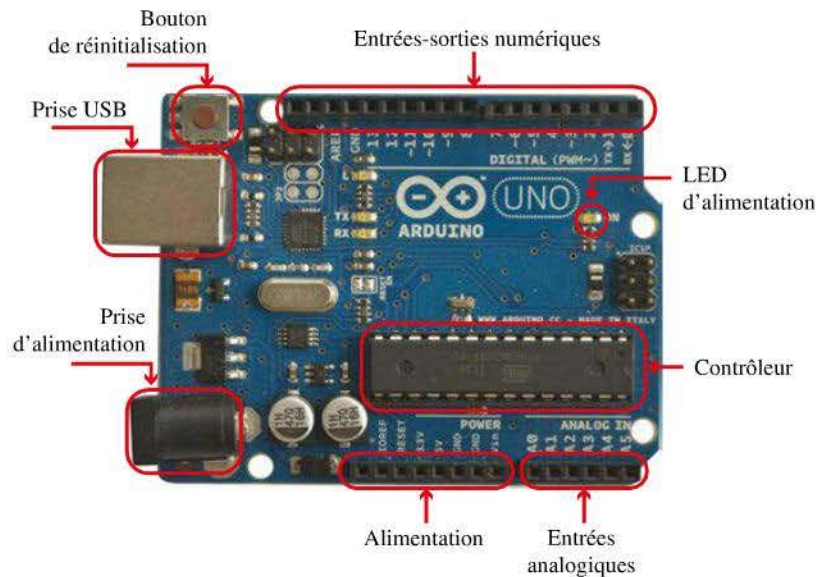
Au fil des années, plusieurs cartes Arduino ont été développées. J'ai choisi de prendre l'exemple de la Uno, car c'est la plus populaire.

Les composants les plus divers y sont reconnaissables (voir figure 3-2), lesquels feront l'objet d'une explication détaillée. Certains penseront certainement qu'on ne peut pas concevoir quelque chose de sérieux sur une surface aussi réduite. Mais grâce à la miniaturisation des composants ces dernières années, ce qui nécessitait auparavant cinq puces électroniques n'en requiert aujourd'hui plus qu'une.

Le plus gros élément qui saute directement aux yeux est le microcontrôleur proprement dit. Il est de type ATmega328.

J'ai choisi la carte Arduino Uno R3 car, même si d'autres modèles ont suivi depuis, comme Arduino Due ou Arduino Yún, elle s'est quasiment hissée au rang de standard. Elle convient tout particulièrement à ceux qui veulent faire leurs premiers pas dans le monde des microcontrôleurs.

Figure 3-2 ►
Que trouve-t-on
sur la carte Arduino ?



Ces éléments sont les plus importants de la carte Arduino mais, bien entendu, cela ne veut pas dire que les autres sont à négliger.

Voici les principales caractéristiques de la carte Arduino :

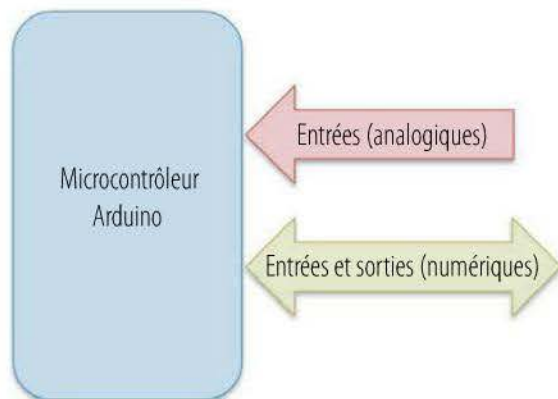
- microcontrôleur ATmega328 ;
- tension de service 5 V ;
- 14 entrées et sorties numériques (6 sorties commutables en MLI) ;
- 6 entrées analogiques (résolution 10 bits) ;
- 32 Ko de mémoire flash (0,5 Ko occupé par le chargeur d'amorçage ou *bootloader*) ;

- 2 Ko de SRAM ;
- 1 Ko d'EEPROM ;
- fréquence d'horloge 16 MHz ;
- interface USB.

Une question simple me vient à l'esprit. Quand je regarde la carte, je me demande ce qu'elle peut bien avoir de spécial. Puis-je communiquer d'une manière quelconque avec le microcontrôleur et, si oui, comment ?



Comme vous pouvez le constater, un certain nombre d'entrées ou de sorties sont disponibles pour communiquer avec la carte Arduino. Elles constituent l'interface avec le monde extérieur et permettent d'échanger des données avec le microcontrôleur, comme l'indique le schéma 3-3.



◀ **Figure 3-3**
Entrées et sorties
de la carte Arduino

Le microcontrôleur Arduino, représenté en bleu à gauche, peut communiquer avec nous via certaines interfaces. Certains ports servent d'entrées (flèche rose), et d'autres d'entrées et de sorties (flèche verte). Un port est ici un chemin d'accès défini au microcontrôleur, pratiquement une porte vers l'intérieur qu'il est possible d'actionner.

Vous apercevez également des réglettes de raccordement noires sur ses bords supérieur et inférieur.



Pas si vite ! Quelque chose ne colle pas. Notre microcontrôleur doit avoir des ports d'entrée et de sortie aussi bien analogiques que numériques. Or, je ne vois sur ce schéma que des entrées comme ports analogiques. Où sont les sorties ?

Bien observé, Arduus ! Mais le schéma est tout à fait correct. La raison en est la suivante et fera l'objet d'une explication plus détaillée : notre carte Arduino n'est pas équipée de sorties analogiques séparées. Cela peut paraître bizarre au premier abord, mais certaines broches numériques sont détournées de leur destination première et servent de sorties analogiques.

Maintenant, vous devez vous demander comment tout cela fonctionne ! Voici donc un avant-goût de ce qui sera expliqué dans la section « Que signifie MLI ? » du chapitre 10, page 214, consacrée à la modulation de largeur d'impulsion. Il s'agit d'un procédé dans lequel le signal présente des phases à niveau haut et des phases à niveau bas plus ou moins longues. Si la phase à niveau haut, dans laquelle le courant circule, est plus longue que celle à niveau bas, une lampe branchée par exemple sur la broche correspondante éclairera visiblement plus fort que si la phase à niveau bas était la plus longue. Plus d'énergie sera donc apportée en un temps donné sous forme de courant électrique. À cause de la persistance rétinienne de notre œil, nous ne pouvons différencier des événements changeant rapidement que sous certaines conditions, et un certain retard se produit aussi lorsque la lampe passe de l'état allumé à celui éteint, et réciproquement. Cela m'a tout l'air d'une tension de sortie qui se modifie, bizarre non ?

En tout cas, ce mode de gestion des ports présente d'emblée un inconvénient. Quand vous utilisez une ou plusieurs sorties analogiques, c'est au détriment de la disponibilité des ports numériques – il y en a alors d'autant moins à disposition – mais cela ne saurait nous gêner outre mesure, car nous n'atteignons pratiquement pas les limites de la carte. De ce fait, nous n'avons pas de restriction sur les montages expérimentaux à tester.



Une question encore, avant que vous ne poursuiviez sur votre lancée : qu'est-ce que le chargeur d'amorçage (ou *bootloader*) que vous avez mentionné dans l'énumération des caractéristiques de la carte Arduino ?

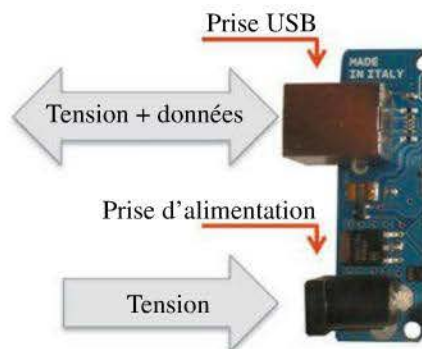
Ah oui Arduus, j'allais oublier ! Un chargeur d'amorçage est un petit logiciel qui a sa place dans une certaine zone de la mémoire flash du microcontrôleur et assure le chargement du programme proprement dit. Normalement, un microcontrôleur reçoit son programme de travail d'un matériel informatique supplémentaire, par exemple un

programmeur ISP (*In System Programming*). Le chargeur d'amorçage évite cela, ce qui rend le téléchargement du logiciel vraiment facile. Sitôt dans la mémoire de travail du contrôleur, le programme de travail est exécuté. Si jamais vous deviez changer, pour une raison quelconque, votre microcontrôleur ATmega328 sur la carte, le nouveau circuit ne saurait pas ce qu'il doit faire car le chargeur d'amorçage n'est pas chargé par défaut. Cette procédure peut être menée au moyen de différents procédés que je ne peux pas expliquer ici faute de place. Cependant, vous trouverez sur Internet suffisamment d'informations pour vous permettre d'installer le chargeur d'amorçage approprié au microcontrôleur.

L'alimentation électrique

Notre carte Arduino doit être alimentée en énergie pour pouvoir travailler.

Cette alimentation peut s'effectuer tout d'abord via l'interface USB qui relie la carte à l'ordinateur – ce chemin sert aussi à l'échange de données entre la carte et l'ordinateur. En phase de développement avec votre Arduino, la connexion USB va servir d'alimentation primaire de la carte. La seconde possibilité consiste à brancher une batterie ou un bloc secteur au connecteur, appelé prise jack. Vous pouvez, par exemple, employer cette variante si vous avez construit un engin manœuvrable, commandé par la carte Arduino. Le véhicule doit pouvoir évoluer librement dans l'espace, sans câble. En effet, l'utilisation d'un câble USB, généralement trop court, limiterait alors la mobilité de l'engin. L'emploi d'une batterie rend le dispositif autonome.



◀ **Figure 3-4**
Alimentation
de la carte Arduino

Je vous montre ici les différentes prises. Attention, elles ne peuvent pas être interverties, car elles ont des formes et des fonctions différentes.

Prise USB



Prise jack



Dès qu'il s'agit de courant ou de tension, il convient de consulter le tableau 3-1.

Tableau 3-1 ►
Valeurs de courant
ou de tension

Catégorie	Valeur
Tension de service	5 V (DC)
Alimentation depuis l'extérieur (recommandée)	7-12 V (DC)
Alimentation depuis l'extérieur (valeur limite)	6-20 V (DC)
Courant continu par broche (maximal)	40 mA

DC : Direct Courant ou courant continu

L'interface USB peut fournir un courant maximal de 500 mA ; c'est en principe suffisant pour réaliser la plupart des circuits d'essai de ce livre. Elle est protégée contre les courts-circuits et les courants forts grâce à un polyfusible. Mais attention, cela ne doit pas vous empêcher de construire votre circuit avec le plus grand soin.

Rappelez-vous ce que j'ai dit dans l'introduction sur le concentrateur USB et ne perdez jamais cela de vue (voir page XII).

Les modes de communication

Un microcontrôleur de type Arduino a déjà beaucoup de connexions qu'il s'agit de bien distinguer.

Le port USB

Sans le port USB, vous ne seriez pas en mesure d'initialiser une communication.

Le travail avec la carte Arduino peut se diviser en deux étapes : le temps consacré à la mise en œuvre du montage et celui dédié à la programmation, appelé phase de développement (*design time*).

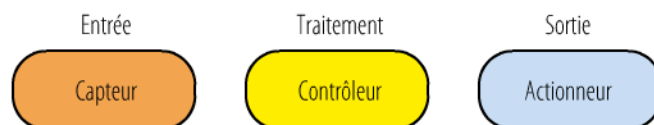
La programmation s'effectue dans un environnement de développement que vous allez apprendre à connaître très rapidement. C'est dans cet environnement que vous allez saisir le programme créé par vos soins pour le transmettre au microcontrôleur. Si tout s'est bien passé, le temps de l'exécution (*runtime*) commence. Vous n'avez pas besoin de dire explicitement au microcontrôleur : « Maintenant mon ami, vous vous mettez au travail ! » Il démarre en effet immédiatement après avoir reçu toutes les instructions de votre part. Vous pouvez en outre échanger des données avec votre ordinateur via le port USB. Nous verrons plus tard comment cela fonctionne.

Les ports d'entrée ou de sortie (E/S)

Les ports E/S représentent l'interface du microcontrôleur. Il existe plusieurs chemins ou canaux pour échanger des données, comme chez l'Homme avec les yeux, les oreilles et la bouche. Il se produit, grâce et par le biais de ces canaux de communication, une interaction avec l'environnement.

Votre carte Arduino utilise des données provenant de capteurs (par exemple, de température, de lumière ou d'humidité) pour réagir en conséquence et entreprendre des actions appropriées. Elle peut aussi activer des dispositifs lumineux et sonores, ou agir sur des actionneurs (moteurs et capteurs).

Vous avez certainement compris que nous avons affaire à deux types de signaux de commande. Des capteurs fournissent des données, et des actionneurs convertissent des grandeurs d'entrée en grandeurs de sortie. Ce processus se déroule selon le principe ETS (Entrée, Traitement, Sortie).



◀ **Figure 3-5**
Le principe ETS

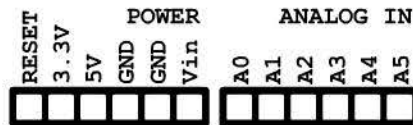
Où se trouvent ces ports d'entrée et de sortie sur la carte Arduino ? Quand vous la tenez de manière à pouvoir lire l'inscription « UNO », vous verrez les ports d'entrée et de sortie sur le bord supérieur (un bloc de dix broches et un autre de huit broches).



◀ **Figure 3-6**
Entrées et sorties numériques

Il est bien entendu important de savoir pour chaque port quelle est son adresse afin de pouvoir communiquer avec lui au cours de la programmation. C'est pour cela que chaque broche possède un numéro. La numérotation de la première broche commence par 0 – en programmation, presque toutes les numérotations commencent par 0. Sous certains chiffres se trouve un tilde (~) indiquant que la broche peut être commutée en sortie analogique. Il s'agit là d'une broche MLI (rappelez-vous la modulation de largeur d'impulsion ; on y reviendra un peu plus loin). À l'extrémité inférieure de la carte se situent les ports d'alimentation (à gauche) et les ports d'entrée analogiques (un bloc de huit broches et un autre de six broches).

Figure 3-7 ►
Alimentation et entrées
analogiques



La numérotation du port analogique commence par 0 mais, cette fois-ci, en partant de la gauche.



Pour aller plus loin

Avant de câbler les différentes broches, orientez-vous toujours à l'aide des désignations correspondantes qui se trouvent soit au-dessus, soit en dessous. Les broches étant très proches les unes des autres, vous risquez vite de mal les lire, voire carrément de vous tromper en câblant la broche voisine de gauche ou de droite. Cela peut être très grave si vous reliez deux broches avoisinantes ou plus, car vous allez alors provoquer un court-circuit. Dans ce cas, il se peut qu'un élément ou deux se mettent éventuellement à fumer dans le circuit. Le mieux est donc de lire les barrettes à la verticale à partir du haut, car une lecture en biais et de côté est risquée.

Attention, ne câblez jamais une carte sous tension alimentée par le port USB. Prenez le temps de bien câbler le circuit – une ligne mal câblée peut endommager la carte –, et évitez de penser sans cesse à l'étape suivante, à savoir le test du circuit. Il est impératif de rester concentré sur son travail et tout ira pour le mieux.

Les langages de programmation C/C++

Pour que la communication avec la carte Arduino se déroule sans problème, les développeurs doivent convenir d'une base de langage, afin qu'ils puissent se comprendre entre eux et exploiter un flux d'informations. C'est la même chose que lorsque vous allez à l'étranger et que vous ne maîtrisez pas la langue. Dans ce cas, vous

devez vous adapter d'une façon ou d'une autre, peu importe la manière (gestes...).

Le microcontrôleur ne connaît à son niveau d'interprétation que le langage machine, appelé aussi code natif, composé exclusivement de valeurs numériques. Il est très difficile à comprendre, car nous avons appris tout petit à échanger à l'aide de mots et de phrases, et non de valeurs numériques. Nous devons donc trouver un moyen de pouvoir communiquer de manière compréhensible avec le microcontrôleur. C'est pourquoi un environnement de développement traduisant les commandes dans un langage dit évolué – autrement dit, se présentant sous une forme semblable à notre langage – a été créé. Pour autant, nous ne sommes pas plus avancés puisque le microcontrôleur ne comprend pas ce langage. En effet, il manque une sorte de traducteur servant de lien entre lui et l'environnement de développement. C'est le rôle du compilateur qui convertit un programme écrit en langage évolué en un langage cible compréhensible par le destinataire (ici, le CPU, ou *Central Processing Unit*, de notre microcontrôleur).



◀ **Figure 3-8**
Le compilateur sert de traducteur.

Presque tous les langages de programmation font appel au vocabulaire anglais ; nous n'avons donc pas d'autre choix que de nous y mettre. Une autre étape de traduction sera donc nécessaire, mais je pense que l'anglais scolaire suffira ici dans la plupart des cas. Les instructions – autrement dit, les ordres – que l'environnement de développement comprend sont concises et semblables à celles du langage militaire, et représentent ce qu'il faut faire.

Micro ! Branchez la lampe au port 13, exécution !

Ne vous en faites pas, elles vous seront enseignées au fur et à mesure. Comme l'indique fort justement le titre de cette section, C et C++ sont également des langages évolués. Aujourd'hui, tous les programmes professionnels sont écrits en C/C++ ou dans des langages apparentés tels que C# ou Java, qui ont tous une forme de syntaxe similaire.

À tous les programmeurs qui s'offusquent de ne pas voir ici leur langage favori, je tiens à préciser que cela ne signifie pas que je considère tous les autres langages (et il y en a beaucoup) comme non



professionnels. Nous en restons ici à C/C++ car Arduino, tout comme le compilateur, dispose d'une partie de la fonctionnalité des langages C/CC++. Ainsi, ceux qui ont déjà programmé avec ne se sentiront pas perdus, et nous ferons en sorte que les autres se sentent eux aussi rapidement à l'aise. Par ailleurs, beaucoup d'autres packs de développement avec microcontrôleur utilisent des compilateurs compatibles C/C++ ; autrement dit, l'étude de ces langages va bientôt se révéler utile. Mais concentrons-nous maintenant sur Arduino.



Je voudrais bien voir un peu de code maintenant. Allez, juste un exemple pour voir, d'accord ?

En voilà un qui ne sait pas attendre ! Voici juste un exemple simple que nous retrouverons bientôt de toute façon :

```
int ledPin = 13;           //Déclarer une variable
                           //initialisée à 13

void setup(){
  pinMode(ledPin, OUTPUT); //Broche numérique 13 comme sortie
}

void loop(){
  digitalWrite(ledPin, HIGH); //LED au niveau haut (5 V)
  delay(1000) ;               //Attendre une seconde
  digitalWrite(ledPin, LOW);  //LED au niveau bas (0 V)
  delay(1000);                //Attendre une seconde
}
```

Dans cet exemple, vous faites clignoter une diode branchée à la broche de sortie numérique 13. Ne me dites pas que vous voulez déjà essayer, car je n'ai même pas encore expliqué les principes de l'installation du pilote ! Pour cela, vous devez attendre et, avant tout, configurer correctement l'environnement de développement. On y va ?

Comment puis-je programmer une carte Arduino ?

Comme je l'ai dit déjà, nous disposons, pour la programmation du microcontrôleur Arduino, d'un environnement de développement – appelé également IDE (*Integrated Development Environment*) –, au moyen duquel on entre directement en contact avec la carte et on charge le programme dans le microcontrôleur. Un programme est

appelé *sketch* dans le contexte Arduino, qu'on peut traduire approximativement par esquisse. À l'avenir, nous parlerons donc de sketches pour désigner les programmes Arduino.

Pour toucher un large public avec Arduino, des environnements de développement qui se ressemblent ont été créés pour les plates-formes les plus diverses. Le système d'exploitation le plus connu et le plus répandu est Windows. C'est pourquoi j'ai décidé de développer tous les sketches de cet ouvrage sous Windows – ce qui ne veut pas dire, bien évidemment, que les autres plates-formes sont mauvaises ! Les différentes versions pour les systèmes d'exploitation suivants sont disponibles sur le site Internet <http://www.arduino.cc/en/Main/Software> :

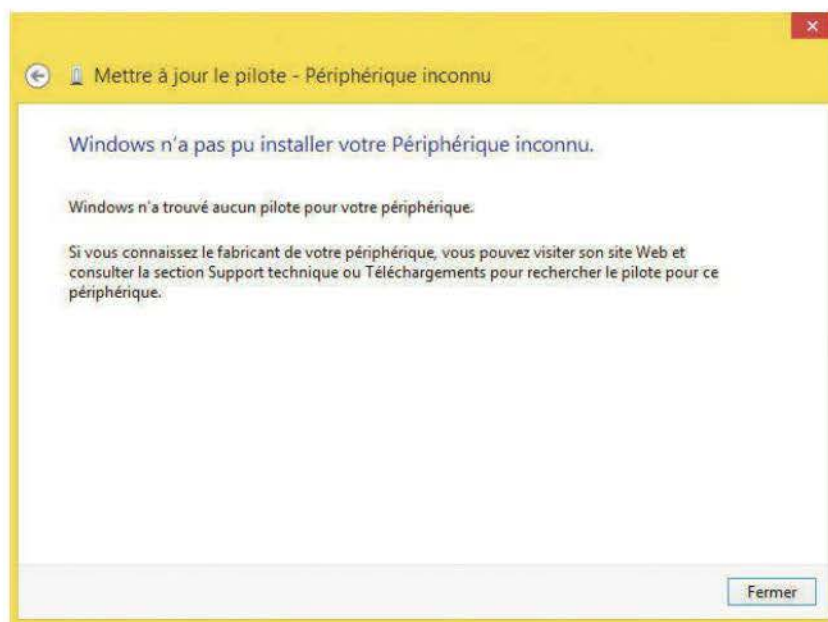
- Windows ;
- Mac OS X ;
- Linux (32 bits) .

Vous y trouverez également des *Release Notes* (ou notes de validation) contenant des informations importantes sur la version de l'IDE concernée. Il est question, par exemple, de nouvelles caractéristiques ou d'erreurs éliminées qui se sont produites dans la version précédente. Il est intéressant d'y jeter un coup d'œil.

Installation de l'environnement de développement, pilote inclus

J'en ai déjà tellement dit sur l'environnement de développement qu'il est temps maintenant de le voir d'un peu plus près. Je commencerai par Windows (en traitant l'installation sous Windows 7, mais l'opération demeure similaire avec une autre version de Windows), puis je passerai à Mac OS X et terminerai par Linux. Cet ordre n'est évidemment pas lié à la valeur de ces systèmes d'exploitation. Attention, si vous connectez votre carte Arduino Uno à votre ordinateur sans avoir préalablement installé l'environnement de développement, le système d'exploitation recherchera automatiquement le pilote correspondant et la tentative échouera, accompagnée du message d'erreur suivant.

Figure 3-9 ►
Le pilote Arduino
est introuvable.



Les versions de l'environnement de développement évoluent vite et vous devez régulièrement procéder à des mises à jour. Quand vous lirez ce livre, il est possible que la version décrite soit déjà dépassée, mais cela ne signifie pas pour autant que les sketches présentés ne fonctionnent plus : la rétrocompatibilité est généralement assurée.

Figure 3-10 ►
L'environnement de
développement Arduino pour les
différents systèmes d'exploitation



Installation sous Windows 7

Étape 1

Pour l'installation sous Windows, j'ai choisi l'option *Windows Installer* qui installe automatiquement les pilotes requis sur l'ordinateur.



◀ **Figure 3-11**
Téléchargement du fichier
d'installation pour Windows

Cliquez sur le bouton *Enregistrer le fichier* pour enregistrer le fichier d'installation sur votre ordinateur.

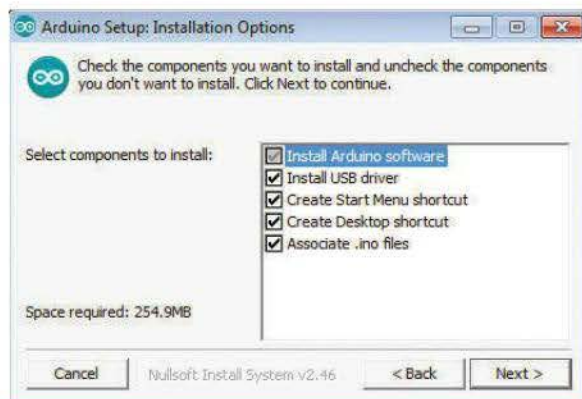
Étape 2

Ouvrez le fichier d'installation et suivez les instructions qui s'affichent dans les boîtes de dialogue. Commencez par cliquer sur le bouton *I Agree*.



◀ **Figure 3-12**
Acceptation du contrat de licence

Vous voyez alors la liste des composants qui seront installés. Cliquez sur le bouton *Next*.



◀ **Figure 3-13**
Choix des composants à installer

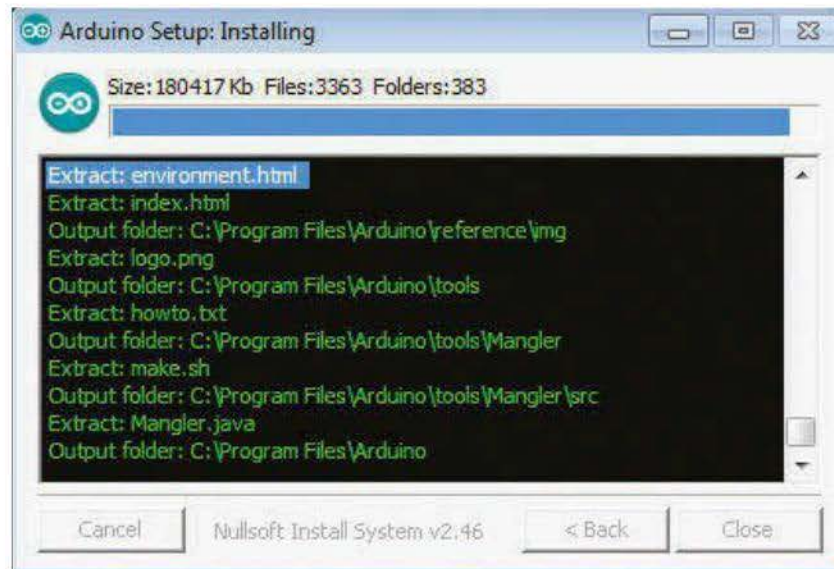
Vous pouvez ensuite changer d'emplacement pour le dossier d'installation. Cliquez sur *Install* pour valider le chemin d'accès proposé.

Figure 3-14 ►
Choix de l'emplacement
des fichiers du programme

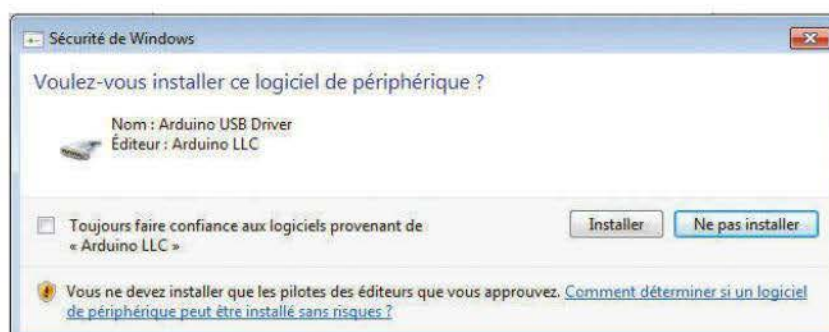


L'installation commence. Vous pouvez en suivre la progression grâce à une barre.

Figure 3-15 ►
Progression de l'installation

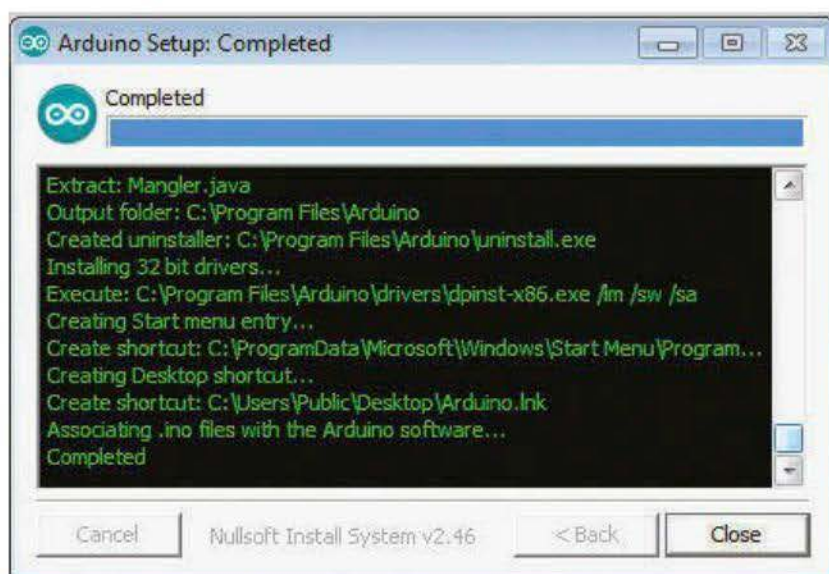


Un message de sécurité de Windows surgit en cours d'installation. Ne vous laissez pas troubler. Fermez la boîte de dialogue en cliquant sur *Installer*. Ce message apparaît lorsque l'éditeur du pilote est inconnu. Ici, vous ne courez aucun risque.



◀ **Figure 3-16**
Message de sécurité de Windows

Une fois l'installation terminée, cliquez sur le bouton *Close*.



◀ **Figure 3-17**
L'installation est terminée.

Étape 3

Lorsque vous raccordez votre carte Arduino Uno à votre ordinateur via un câble USB, elle doit désormais apparaître dans le gestionnaire de périphériques. La carte Uno n'est pas livrée avec un câble USB. Par conséquent, pensez aussi à vous en procurer un, sinon la carte ne présentera pas grand intérêt.

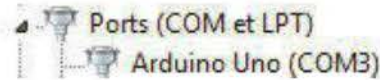


◀ **Figure 3-18**
Câble USB pour relier la carte Arduino à l'ordinateur

La prise de type B est reliée à la carte, et celle de type A à l'ordinateur. Rappelez-vous ce que j'ai mentionné dans l'avant-propos : l'utilisation d'un concentrateur USB est conseillée.

Allez maintenant dans le gestionnaire de périphériques de l'ordinateur. Pour cela, cliquez droit sur l'icône du Bureau, puis sélectionnez Gestion. Ouvrez le gestionnaire de périphériques. Dans l'arborescence qui apparaît, vous trouverez une entrée sous *Ports (COM & LPT)*.

Figure 3-19 ►
La carte Arduino Uno est reconnue.



Vous pouvez maintenant ouvrir l'environnement de développement depuis le menu Démarrer de Windows.

Installation sous Mac OS X (Mavericks)

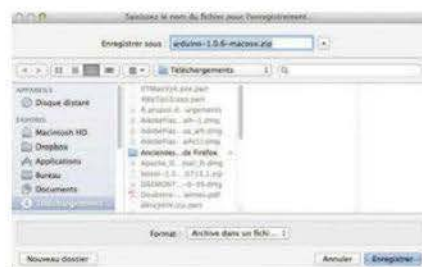
Pour installer l'environnement de développement Arduino sous Mac OS X, ouvrez la page Internet correspondante dans votre navigateur Safari.

Figure 3-20 ►
La page de téléchargement
du logiciel Arduino dans Safari



Cliquez sur le lien *Mac OS X* et enregistrez le fichier. Une fois le téléchargement terminé, le fichier se trouve dans le dossier *Téléchargements*.

Figure 3-21 ►
Le dossier Téléchargements
dans le Finder



Étape 1

Pour lancer l'environnement de développement Arduino, dézippez le fichier téléchargé et sur le fichier *Arduino*. Un message de sécurité apparaît pour vous signaler que le fichier a été téléchargé sur Internet. Cette question ne réapparaîtra plus lorsque vous aurez accepté d'ouvrir le programme.



◀ **Figure 3-22**

Message de sécurité affiché au démarrage de l'application

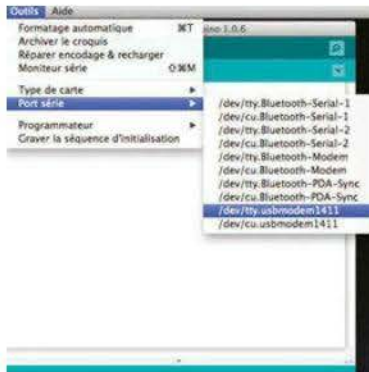
Cliquez sur le bouton *Ouvrir*. Pour accéder plus rapidement à l'interface, placez le fichier téléchargé dans le dossier *Programmes*.

Étape 2

Connectez votre carte Arduino Uno à votre Mac à l'aide de son câble USB. Avant de commencer, vous devez encore configurer deux choses sur lesquelles je reviendrai dans la suite de ce chapitre :

- via quel port l'Arduino est-elle raccordée au Mac ?
- de quelle carte Arduino s'agit-il ?

Dans le menu *Outils>Port série*, sélectionnez l'entrée */dev/tty.usbmodem14441*.



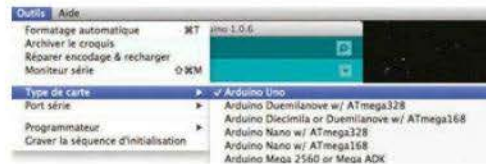
◀ **Figure 3-23**

Sélection du port USB

Si cette entrée n'est pas proposée, débranchez le câble USB de la carte Arduino de votre Mac et parcourez la liste des ports disponibles. Puis branchez à nouveau le câble USB et examinez la liste. L'entrée qui s'est ajoutée est la bonne.

Il ne vous reste plus choisir la bonne carte Arduino. Affichez la liste des cartes Arduino prises en charge dans le menu *Outils>Type de carte* et sélectionnez l'entrée *Arduino Uno*.

Figure 3-24 ►
Sélection de la carte Arduino Uno



L'environnement de développement est prêt et vous pouvez commencer !

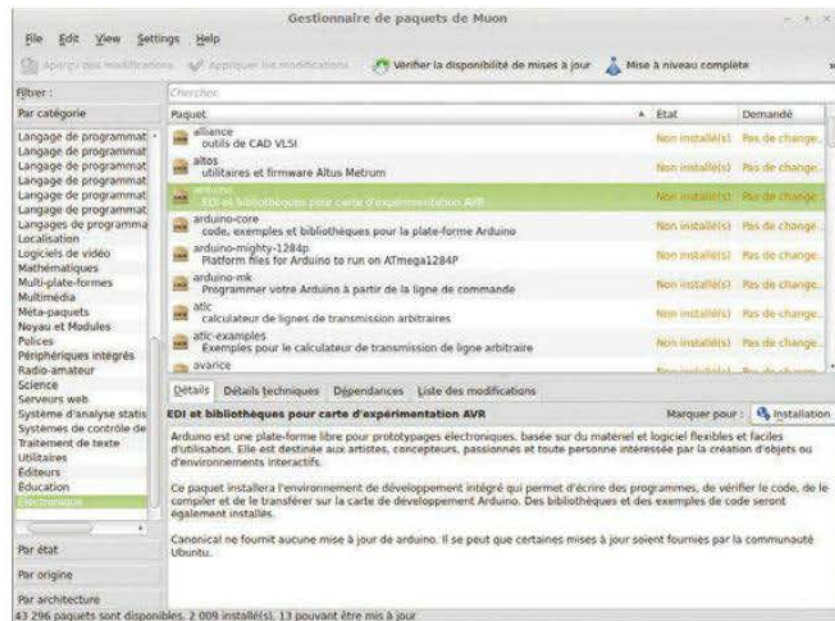
Installation sous Ubuntu

Un pack d'installation de la version Arduino 1.0.5 existe déjà pour la version KUbuntu 13.04 de Linux. Je vous montre ici une méthode d'installation simple par le biais de Muon. J'ai installé KUbuntu sur mon ordinateur. Il s'agit d'Ubuntu avec l'environnement de travail KDE au lieu de Gnome.

Étape 1

Sélectionnez *Muon* dans le menu *Applications*>*Système* et cliquez sur la catégorie *Électronique*, à gauche.

Figure 3-25 ►
L'interface de Muon sous KUbuntu



L'environnement de développement d'Arduino apparaît et vous pouvez démarrer son installation en cliquant sur le bouton *Installer*.

Étape 2

L'installation du pack modifie aussi d'autres fichiers.

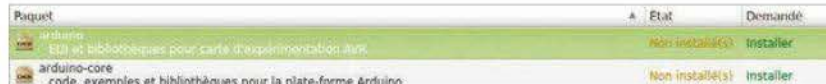


◀ **Figure 3-26**
Les modifications supplémentaires

Cliquez sur *OK*.

Étape 3

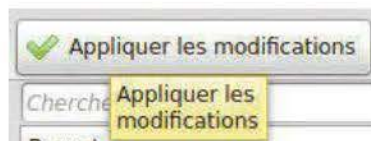
Tous les packs devant être installés sont affichés.



◀ **Figure 3-27**
Les packs sont présélectionnés.

Étape 4

Pour démarrer l'installation du pack, il ne vous reste plus qu'à cliquer sur le bouton *Appliquer les modifications* dans la partie supérieure de la fenêtre.

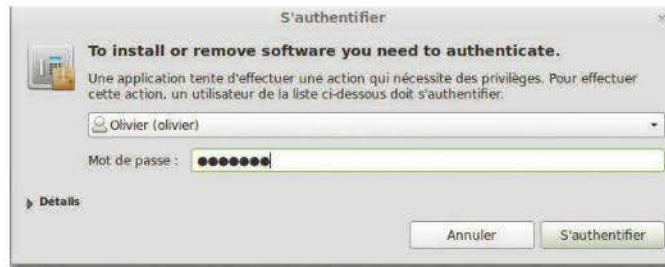


◀ **Figure 3-28**
Démarrage de l'installation

Étape 5

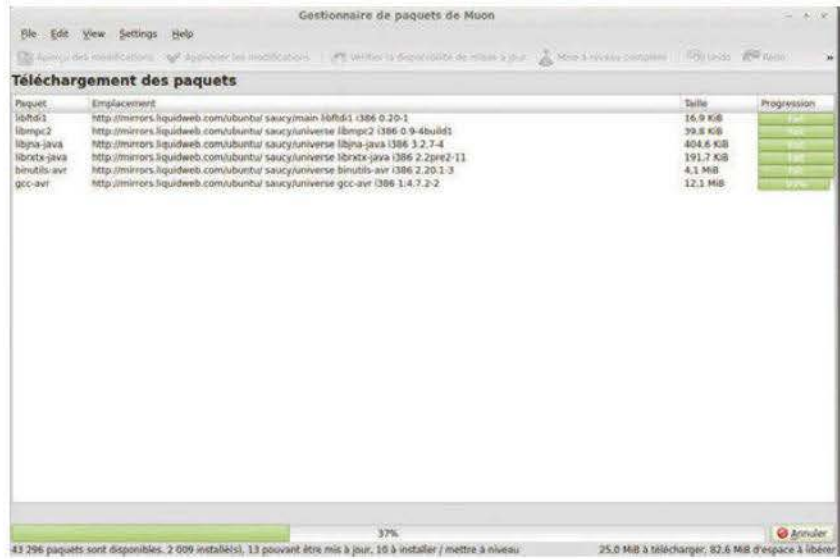
La gestion des logiciels étant la chasse gardée de l'administrateur, vous devez saisir le mot de passe correspondant.

Figure 3-29 ►
Saisie du mot de passe racine



L'installation du pack commence enfin.

Figure 3-30 ►
L'installation commence.



Étape 6

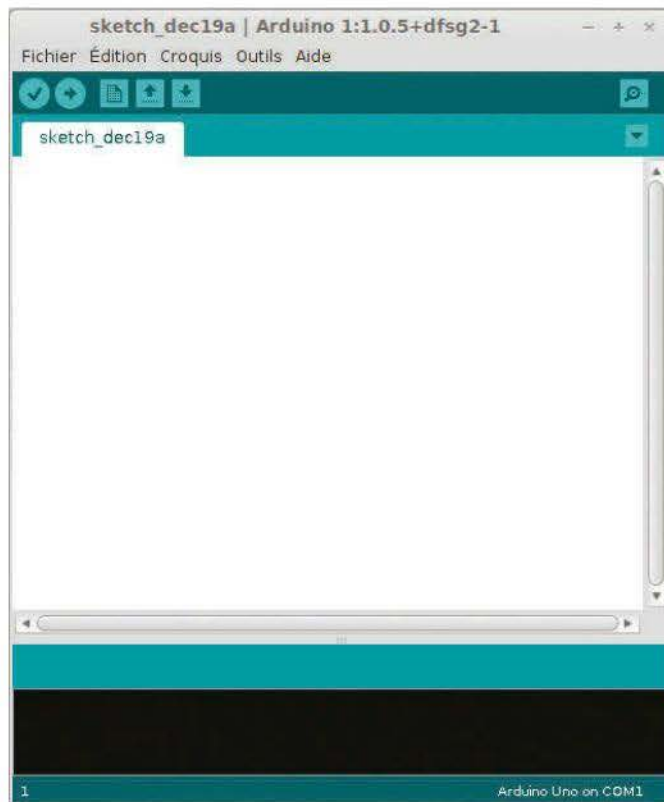
Pour pouvoir communiquer avec la carte, l'environnement de développement Arduino doit faire partie du groupe *Dialout*. Cliquez sur le bouton *Add* pour l'y autoriser.

Figure 3-31 ►
Ajout au groupe Dialout



Étape 7

Tout est prêt maintenant et vous pouvez lancer l'environnement de développement Arduino depuis la commande *Applications>Électronique*.



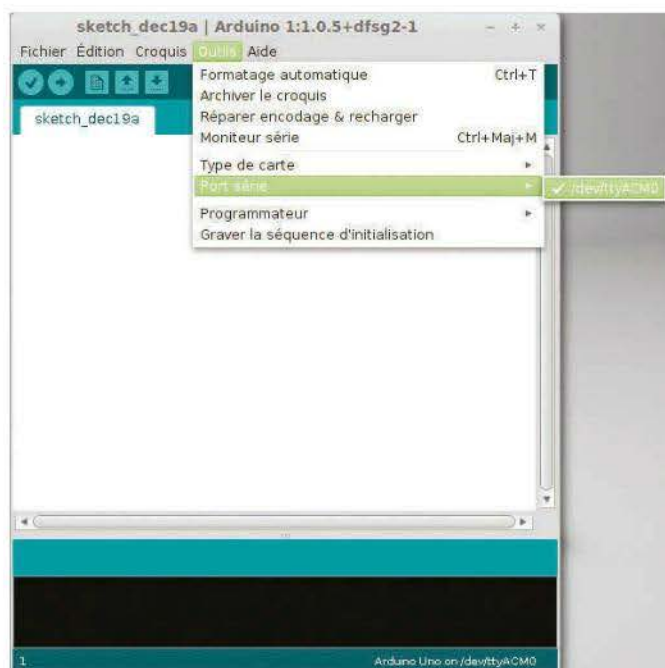
◀ **Figure 3-32**
L'environnement
de développement Arduino

Si vous jetez un œil en bas à droite de la fenêtre de l'environnement de développement, vous pouvez y lire la mention *Arduino Uno on COM1*. Ce n'est pas tout à fait exact, car la dénomination COM n'est pas utilisée sous Linux pour un port série. Vous devez donc sélectionner le bon port.

Étape 8

Dans le menu *Outils>Port série*, sélectionnez l'entrée */dev/ttyACM0*.

Figure 3-33 ►
Affectation du bon port



Attention !

Il arrive parfois que la commande *Port série* soit grisée, ce qui vous empêche d'affecter un nouveau port. Réinitialisez Linux – manœuvre cependant rarement nécessaire – et vous pourrez ensuite sélectionner le port série adapté.

L'environnement de développement d'Arduino

Qu'est-ce qu'un environnement de développement et que peut-on faire avec ? Eh bien, il permet, au développeur ou à l'expert Arduino que vous allez bientôt être, de transposer ses idées de programmation sur des objets matériels (*hardware*), avec comme composant principal la carte Arduino, à laquelle peuvent être raccordés les éléments électroniques ou électriques les plus divers. Ce sont là des choses simples, mais *hard* par leur structure, d'où le terme *hardware* employé.

Seulement, à quoi sert ce matériel si on ne lui dit pas ce qu'il doit faire ? Il y a en effet quelque chose qui manque, et ce quelque chose, c'est le logiciel (*software*), le monde des données et des programmes – ou des *sketches* dans le cas d'Arduino. Le logiciel est

soft, autrement dit immatériel, à moins de l'imprimer sur papier. Il permet au matériel d'interpréter et d'exécuter des instructions.

Le hard et le soft forment une entité indissociable, ils ne sont rien l'un sans l'autre.

Lancement de l'environnement de développement

Venons-en maintenant aux choses concrètes. Le démarrage de l'environnement de développement, que j'appellerai dorénavant IDE, est proche. Sous Windows, vous y accédez depuis le menu du bouton Démarrer dans lequel vous reconnaîtrez l'icône spécifique d'Arduino.

La fenêtre suivante apparaît au démarrage.



◀ **Figure 3-34**

La fenêtre IDE vide (sous Windows)

En l'observant de plus près, vous pourrez remarquer différentes zones, dans lesquelles il se passera peut-être quelque chose plus tard... Nous allons les passer toutes en revue, en partant du haut de la fenêtre vers le bas.

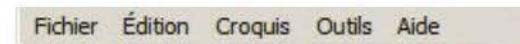
La barre de titre

La barre de titre, qui se situe tout en haut de la fenêtre, comprend deux informations :

- le nom du sketch (ici, `sketch_sep22a`). Il est attribué automatiquement et commence toujours par `sketch`. Viennent ensuite le mois, le jour et une lettre prise dans l'ordre, entre `a` et `z`, dans le cas où d'autres sketches seraient créés ce jour-là. Notre sketch a été créé le 19 décembre, dans sa première version ;
- le numéro de version de l'IDE Arduino (ici, la version 1.0.6), qui augmentera au fil du temps dès que des erreurs auront été éliminées ou de nouvelles fonctions ajoutées.

La barre de menus

Dans la barre de menus, vous trouverez les différents menus grâce auxquels vous pourrez appeler certaines fonctions de l'IDE.



La barre d'icônes

La barre d'icônes se situe sous celle des menus.



La zone des onglets

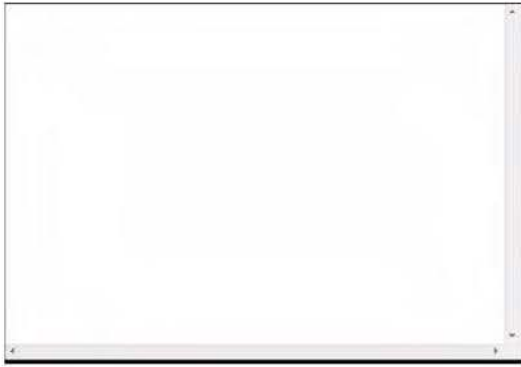
La zone des onglets indique combien de fichiers de code source font partie du projet Arduino actuellement ouvert.

Pour le moment, seul un onglet ayant pour nom `sketch_dec19a` apparaît. Cependant, d'autres peuvent être ajoutés au gré de la programmation. Pour cela, il faut se servir de l'icône située sur le côté droit.



L'éditeur

C'est le cœur de l'IDE. La zone d'édition, qui est pour le moment encore complètement vierge, est le lieu central où vous pouvez étaler vos idées. Vous y saisissez le code source, ainsi que les instructions qui conduiront le microcontrôleur à faire ce que vous voulez.



La ligne d'information

La ligne d'information vous renseigne sur certaines actions entreprises par l'IDE. Tout est en anglais naturellement.

Par exemple, si vous avez enregistré avec succès un sketch sur le disque dur, c'est ici que vous en êtes informé. En outre, si, par exemple, le compilateur a détecté une erreur de saisie dans le sketch lors de la transcription, vous êtes prévenu par un message. D'autres détails sur les erreurs détectées s'affichent dans la fenêtre de messagerie (voir la capture précédente).



Done Saving

La fenêtre de messagerie

L'IDE vous fournit, dans la fenêtre de messagerie, tout un tas d'informations :

- sur le transfert d'un sketch sur la carte Arduino (succès ou échec) ;
- sur les activités de traduction du compilateur (succès ou échec) ;
- sur le moniteur série (succès ou port COM non trouvé).

La ligne d'état

La ligne d'état indique soit le numéro de ligne du curseur (ici, ligne 3) :



3

Arduino Uno on COM3

soit une zone (ici, lignes 1 à 4) :



1 - 4









Arduino Uno on COM3


À droite, vous pouvez voir en plus le nom de votre carte Arduino et le port COM utilisé par votre interface série.

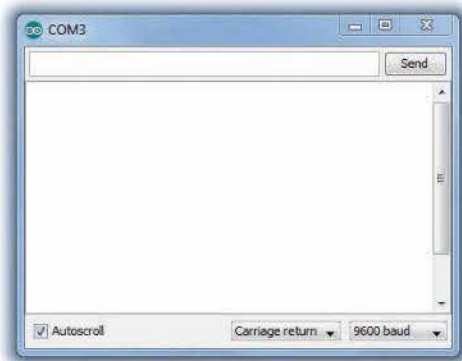
La barre d'icônes en détail

À force d'utiliser quotidiennement l'IDE, vous vous apercevrez que la barre d'icônes est votre compagnon le plus précieux. Même si la barre ne contient pas beaucoup d'icônes, il vous faut néanmoins en maîtriser les fonctionnalités.

Tableau 3-2 ▶
Fonctions des icônes
de la barre d'icônes

icône	Fonction
	Pour vérifier la syntaxe du sketch qui se trouve dans l'éditeur (<i>Verify</i> signifie ici contrôler la syntaxe) et compiler le programme.  Une barre horizontale s'affiche au début de la vérification/compilation, laquelle indique la progression. Si aucune erreur n'est constatée, l'opération se termine par le message <code>Done Compiling</code> . Dans la fenêtre d'édition se trouve une indication relative aux besoins en mémoire du sketch. 
	Pour créer un nouveau sketch. Souvenez-vous que l'IDE ne peut gérer qu'un seul sketch à la fois. Si vous en démarrez un nouveau, n'oubliez surtout pas d'enregistrer l'ancien, faute de quoi vous perdrez toutes les informations.
	Tous les sketches sont consignés dans un livre de sketches qui se trouve dans le répertoire <code>C:\Utilisateur\<Nom d'utilisateur>\Mes documents\Arduino</code> . Le nom d'utilisateur à saisir est le vôtre. Cette icône sert à charger un sketch enregistré sur le disque dur dans l'IDE. Elle vous permet aussi d'accéder aux nombreux exemples de sketches livrés avec l'IDE. Regardez-les car ils peuvent vous aider.
	Pour enregistrer votre sketch sur un support de données. L'enregistrement s'effectue par défaut dans le répertoire du livre de sketches mentionné plus haut.
	Pour transmettre le sketch compilé avec succès sur la carte Arduino dans le microcontrôleur. Pendant ledit téléchargement du sketch, voici ce qui se produit – sur la carte se trouvent des petites diodes lumineuses qui vous tiennent au courant de certaines activités. 
	– LED L : reliée à la broche 13. Elle s'allume brièvement quand la transmission commence.

Icône	Fonction
	<ul style="list-style-type: none"> – LED TX : ligne émettrice de l'interface série de la carte. Elle clignote pendant la transmission. – LED RX : ligne réceptrice de l'interface série de la carte. Elle clignote pendant la transmission. <p>La ligne émettrice (TX) est matériellement reliée à la broche numérique 1, et la ligne réceptrice (RX) à la broche numérique 0.</p>
	Le moniteur série peut être ouvert avec cette icône. Une boîte de dialogue ressemblant à un terminal s'ouvre.



Dans le champ de saisie supérieur, vous pouvez entrer des commandes qui seront envoyées à la carte quand vous appuierez sur la touche *Send*. La zone centrale de la fenêtre est consacrée aux données envoyées par la carte via l'interface série. Certaines valeurs auxquelles vous vous intéressez peuvent y être affichées. Dans la partie inférieure, vous pouvez, grâce à une liste déroulante à droite, régler la vitesse de transmission (*baud*) qui doit correspondre à la valeur que vous avez employée pour programmer le sketch. Si ces valeurs ne correspondent pas, aucune communication n'est possible.



Pour aller plus loin

Dans le cas où vous auriez oublié la fonction qui se cache derrière l'une de ces six icônes, il vous suffit de passer la souris devant l'une d'elles et de regarder à droite de la barre d'icônes pour y lire sa signification.

L'éditeur en détail

L'éditeur, dans lequel vous saisissez votre code source, vous assistera à maintes reprises dans la programmation. La figure 3-35 vous présente le contenu de la fenêtre : il s'agit d'un code source qu'il est inutile de chercher à comprendre pour l'instant. Il s'agit simplement de montrer comment et sous quelle forme celui-ci est représenté.

Tableau 3-2 (suite)

Fonctions des icônes de la barre d'icônes

Figure 3-35 ►
Code source d'un sketch Arduino

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Bonjour mon ami Arduino");
}
```

Quelles sont les caractéristiques qui vous sautent tout de suite aux yeux ? En voici un petit résumé.

1. L'IDE est capable de faire ressortir certains mots en couleurs dans l'éditeur. Lesquels ?
2. Les caractères sont plus ou moins gras selon les mots.
3. Certains éléments ressortent plus particulièrement. Il s'agit ici de l'accolade finale.
4. La représentation du code source obéit à une certaine hiérarchisation visuelle. Certaines zones sont plus décalées à droite que d'autres. Bien évidemment, ce n'est pas pour rien, ni seulement pour faire beau : tout a une raison d'être.

Point 1

Certains mots, appelés également mots-clés, apparaissent en couleurs. Il s'agit de noms réservés qui ont été, par exemple, attribués à des instructions. Notre environnement de développement ou le compilateur dispose d'un certain vocabulaire que nous pouvons utiliser pour programmer notre sketch. Quand on saisit un mot (ou un mot-clé) qu'il connaît, l'IDE réagit en le faisant aussitôt ressortir en couleurs.

Dans le cas présent, les mots-clés apparaissent toujours en orange. Ainsi, vous conserverez une meilleure vue d'ensemble et vous pourrez visualiser immédiatement si une instruction a été mal orthographiée. En effet, si tel est le cas, elle n'apparaîtra pas dans la couleur appropriée.

Point 2

L'IDE représente en gras certains mots reconnus en tant que mots-clés. Il s'agit ici, par exemple, des mots `setup` et `loop`, qui sont appelés à jouer un rôle fondamental dans un sketch. Ce sont des noms de fonctions. Pour le moment, peu importe ce que c'est exactement et ce qu'ils veulent dire, disons simplement qu'ils sont en gras pour mieux attirer l'attention.

Point 3

Les instructions sont toujours présentées par blocs dans l'environnement de programmation IDE. Cela signifie que les instructions affichées l'une en dessous de l'autre font partie d'un bloc d'exécution, signalé par une paire d'accolades : l'accolade initiale marque le début du bloc, et l'accolade finale la fin. Ces deux accolades sont indissociables : si l'une des deux vient à manquer, il s'ensuit obligatoirement une erreur car la structure du bloc n'est pas complète.

Si vous placez le curseur derrière une accolade, l'autre accolade de la paire se retrouve automatiquement encadrée. Sur la figure 3-35, on le remarque pour la fonction `setup` : j'ai placé le curseur derrière l'accolade initiale et l'accolade finale correspondante s'est alors dotée d'un cadre. Ceci est également valable pour les parenthèses. Nous reviendrons bien entendu plus tard sur la différence entre accolades et parenthèses.

Point 4

Dans un bloc d'exécution, le code source est généralement décalé à droite par rapport au bloc ou libellé du bloc proprement dit. Ainsi, la vue d'ensemble est bien meilleure et la recherche d'erreurs en est facilitée. Cette distinction visuelle permet par ailleurs de mieux différencier les blocs quand il y en a plusieurs.

Bien entendu, rien ne vous empêche d'écrire l'intégralité du code source sur une seule ligne. Même si le compilateur ne détecterait aucune erreur de syntaxe, la vue d'ensemble serait néanmoins catastrophique. De même, vous pourriez aligner toutes les lignes de code à gauche, mais le style de programmation ne serait alors pas terrible. Notez qu'il existe une option pour indenter automatiquement le code, via *Tools>Auto format*.



Pour aller plus loin

Si vous avez déjà fait de la programmation avec un environnement de développement dans un autre langage, par exemple C#, vous trouverez à coup sûr que l'environnement de développement Arduino est bien différent. La configuration est ici beaucoup plus spartiate et ne possède pas toutes les fonctions des autres IDE – cela est bien évidemment volontaire. Les développeurs de la carte Arduino ont tenu à ce que facilité et simplicité riment aussi avec maniement et programmation du logiciel.

Beaucoup reculent dès qu'il s'agit de domaines compliqués propres au monde technique, tels que microcontrôleur ou programmation, car ils les jugent beaucoup trop difficiles et ne sont pas sûrs d'y arriver. Ne vous faites pas de souci, vous allez y arriver. Laissez-vous seulement surprendre et séduire par le charme de la carte Arduino !

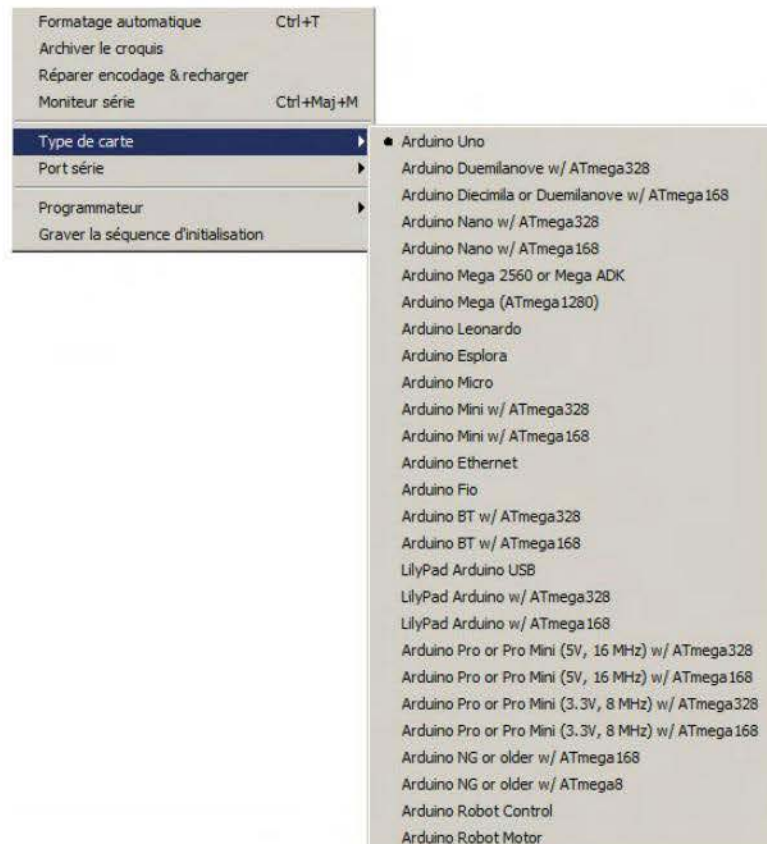
Transmission du sketch sur la carte Arduino

Une fois votre sketch programmé, vérifié et compilé avec succès, les choses deviennent sérieuses. Il s'agit maintenant de le transmettre au microcontrôleur. Néanmoins, une petite chose n'a pas encore été dite. Du fait qu'il existe sur le marché des cartes Arduino très diverses, qui toutes diffèrent plus ou moins par le matériel mais sont alimentées en données par un seul IDE, vous devez effectuer un réglage de base. Ce n'est pas bien compliqué. Avant tout, connectez votre carte Arduino à votre ordinateur

Sélection de la carte Arduino

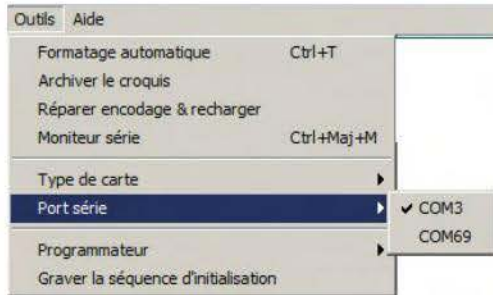
Choisissez donc l'option *Type de carte* dans le menu *Outils* pour obtenir la liste de toutes les cartes prises en charge par l'IDE. Comme vous travaillez avec la dernière carte Uno, il vous faut sélectionner la première entrée de la liste, qui est déjà cochée dans mon cas, car j'avais effectué au préalable le réglage.

Figure 3-36 ►
Sélection de votre carte Arduino
dans l'IDE



Sélection du port série

Sélectionnez maintenant le port COM pour l'interface série à l'aide de la commande *Outils>Port série*. La liste comporte plusieurs entrées parmi lesquelles vous devez choisir la bonne. Au besoin, jetez un coup d'œil dans le gestionnaire de périphériques pour vous aider dans votre choix.



◀ **Figure 3-37**
Sélection du port série dans l'IDE

Le port COM3, sur lequel ma carte est branchée, a été détecté. Avez-vous compris ?

Non, c'est le contraire ! Vous avez du mal à vous exprimer. D'un côté, vous parlez d'une interface série et d'un port COM, et de l'autre, vous reliez la carte à l'ordinateur via une prise USB. Ce sont pourtant deux choses complètement différentes, non ?



Vous avez évidemment raison et j'allais oublier d'en parler. Heureusement que vous êtes attentif ! Les anciennes cartes Arduino possèdent effectivement encore une interface série (RS232) sous la forme d'un connecteur D-sub à 9 broches, relié à l'ordinateur via un câble série. Les ordinateurs récents disposent tous d'une prise USB, ce qui rend peu à peu l'interface série inutile ; d'autres n'ont carrément plus aucune possibilité de connexion série standard. Cependant, le traitement interne suppose un composant série. Alors comment faire ? La carte Arduino dispose, entre autres, de son propre petit microcontrôleur de type ATMEGA8U2-MU, programmé pour servir de convertisseur USB série. Une carte plus ancienne, appelée Duemilanove, était déjà pourvue d'un circuit FTDI qui remplissait la même fonction. La nouvelle carte présente les avantages suivants :

- elle a des temps de latence plus courts (temps entre une action et une réaction attendue) ;
- elle est programmable ;
- elle peut se connecter en tant que clavier USB sur le système.

Dans la variante Linux, il n'y a pas de port COM mais une entrée de type :

`/dev/ttyACMo`

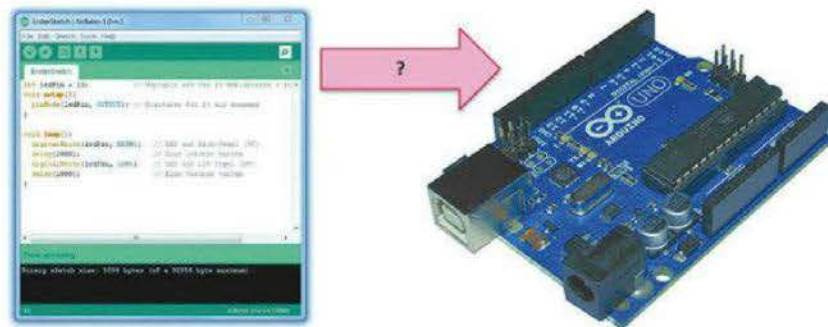
`dev` est l'abréviation de *device* (appareil en français). Vous trouverez d'autres informations sur Internet.



Pouvez-vous m'expliquer un peu ce qui se passe lors de la transmission du code du sketch sur la carte Arduino ? Ou la question est-elle un peu prématurée ?

Non, Arduus, la question n'est pas du tout prématurée et a le mérite d'être posée. Je vous ai déjà parlé un peu de l'environnement de développement, du compilateur et des langages de programmation C/C++. Certains écoutent sans rien dire, vous au moins, vous posez des questions, et c'est bien !

Figure 3-38 ►
Que se passe-t-il en arrière-plan lors de la transmission du sketch sur la carte Arduino ?



La transmission du sketch s'effectue en quatre étapes.

Étape 1

Une vérification du code du sketch est faite par l'environnement de développement, afin de garantir que la syntaxe C/C++ est correcte.

Étape 2

Le code est ensuite envoyé au compilateur (`avr-gcc`), qui le transcrit en un langage lisible par le microcontrôleur : c'est le langage machine.

Étape 3

Le code compilé fusionne avec certaines bibliothèques Arduino qui apportent les fonctionnalités de base, ce qui aboutit à la création d'un

fichier au format Intel HEX. Il s'agit d'un fichier texte qui contient des informations binaires pour le microcontrôleur. Voici un court extrait du premier sketch, dont vous avez déjà eu un avant-goût.

```
ErsterSketch.cpp.hex
1 :10000000C9461000C947E000C947E000C947E0095
2 :10001000C947E000C947E000C947E000C947E0068
3 :10002000C947E000C947E000C947E000C947E0058
4 :10003000C947E000C947E000C947E000C947E0048
5 :10004000C949D000C947E000C947E000C947E0019
6 :10005000C947E000C947E000C947E000C947E0028
7 :10006000C947E000C947E0000000002400270009
8 :100070002A0000000000250028002B0000000000DE
9 :1000800023002600290004040404040404040202DA
```

◀ **Figure 3-39**
Extrait d'un fichier Intel HEX

Le microcontrôleur comprend ce format, car c'est son *Native Language*, c'est-à-dire sa langue maternelle.

Étape 4

Le chargeur d'amorçage (*bootloader*) transmet le fichier Intel HEX, via USB, à la mémoire flash du microcontrôleur. Ledit processus de téléchargement – donc la transmission sur la carte – est assuré par le programme `avrdude`, qui fait partie intégrante de l'installation Arduino. Vous le trouverez sous `arduino\hardware\tools\avr\bin`. D'autres informations sur les paramètres à entrer lors de l'appel sont disponibles sur Internet.

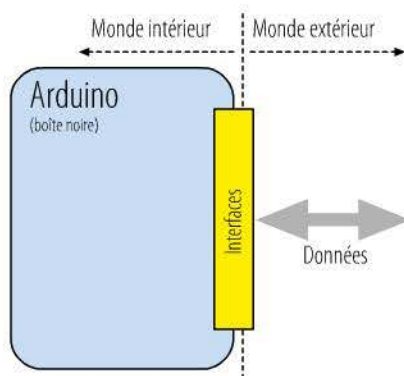
La communication par port

Jusqu'ici, vous ne connaissez de la communication avec votre carte Arduino que le côté programmation. Un sketch est programmé par vous et envoyé sur la carte via le port USB. Immédiatement après son chargement, le sketch s'exécute et traite les données. Mais ces données doivent ensuite parvenir au microcontrôleur via des interfaces, sous forme de valeurs fournies par des capteurs. Si besoin, elles seront enfin renvoyées plus tard vers l'extérieur, pour commander un moteur par exemple. Ce sujet a été juste évoqué au début du chapitre, dans les explications sur les ports analogiques et numériques.

Vous avez dit interfaces ?

Le terme interface est déjà revenu si souvent dans le livre qu'il est temps à présent d'en donner une définition claire et valable. Une interface permet de faire communiquer un système fermé avec le monde extérieur. Voyons à ce sujet la figure suivante.

Figure 3-40 ►
Les interfaces sont des dispositifs de
liaison entre deux mondes voisins.



Avec un pied dans le monde intérieur et un autre dans le monde extérieur, l'interface maintient ainsi le contact entre les deux systèmes, qui s'échangent mutuellement des informations sous forme de données. Dans ce contexte, votre carte Arduino pourrait être assimilée à une boîte noire, car il n'est pas utile de connaître en détail les différents éléments et fonctions qui la constituent.

Qu'est-ce qu'une boîte noire ?

Une boîte noire est un système plus ou moins complexe dont la structure interne n'est pas accessible de l'extérieur à cause de son encapsulation. Mais ici, tout ce qui nous intéresse en tant qu'utilisateur, c'est ce que la boîte noire est capable de faire et comment nous pouvons la commander. Aussi est-elle accompagnée d'une description détaillée de ses interfaces, dans laquelle les fonctionnalités sont expliquées. Votre carte Arduino peut être comparée à une telle boîte.

Ce livre va vous aider à comprendre le fonctionnement des interfaces, leurs particularités et leur comportement.



Bah, nous verrons bien !

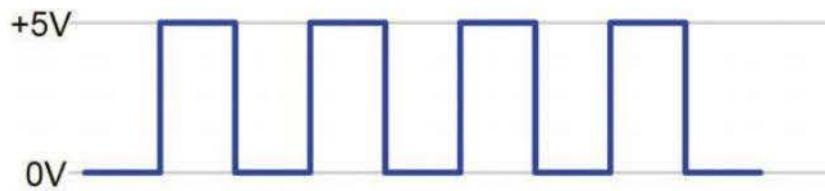
Quelle est la différence entre le numérique et l'analogique ?

Puisque nous parlons de la boîte noire et de la communication par port, et que notre carte Arduino est dotée de ports numériques et analogiques, c'est l'occasion d'expliquer les différences entre ces deux modes.

Le mode numérique (ou digital, du latin *digitus* signifiant « doigt ») fait appel à deux états bien définis :

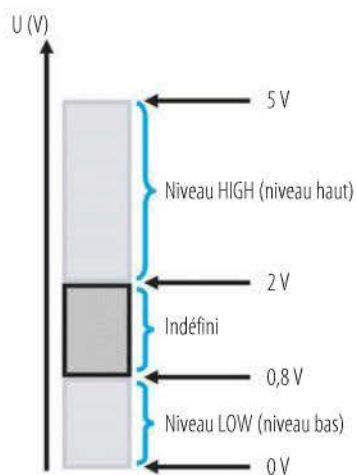
- le niveau LOW ou niveau bas (forme abrégée L ou 0) ;
- le niveau HIGH ou niveau haut (forme abrégée H ou 1).

Vous voyez ici un signal de type numérique.



◀ **Figure 3-41**
Évolution d'un signal numérique
(signal rectangulaire)

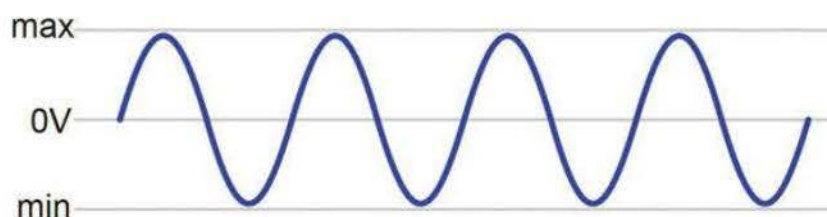
Des valeurs de tension peuvent être attribuées à ces deux états logiques. Dans le cas présent, il s'agit de la logique +5 V pour les signaux numériques. Qu'est-ce que cela signifie ? Dans la technique numérique, des niveaux de tension sont affectés aux états binaires. En principe, la valeur de tension 0 V correspond à la valeur LOW (niveau bas) binaire et la valeur de tension +5 V à la valeur HIGH (niveau haut) binaire. Étant donné qu'il peut y avoir de légers écarts dus aux différentes tolérances des composants, il est nécessaire de définir une plage de tolérance pour les états logiques. Si nous ne mesurons que +4,5 V au lieu de +5 V, ce serait un niveau LOW au sens strict du terme. C'est pour cette raison que des plages de tolérance ont été créées avec les valeurs suivantes.



◀ **Figure 3-42**
Plages de tolérance

Les signaux analogiques ont, quant à eux, une tout autre caractéristique. Avec le temps, ils évoluent de façon continue entre deux valeurs extrêmes (un maximum et un minimum).

Figure 3-43 ►
Évolution d'un signal analogique
(signal sinusoïdal)



Nos exemples porteront sur ces deux types de signaux.

L'entrée (INPUT)

Un flux d'informations peut circuler de façon bidirectionnelle et produire ainsi un échange de données. La carte Arduino dispose de ports qui ont des comportements différents. Il faut naturellement faire encore ici la distinction entre numérique et analogique. Commençons par les entrées.

Entrées numériques

Les entrées numériques de la carte sont alimentées par des capteurs à caractéristique numérique.

Le capteur numérique le plus simple est l'interrupteur. Il peut être soit ouvert et, dans ce cas, il ne délivre aucun signal (niveau LOW ou niveau bas), soit fermé et il délivre un signal (niveau HIGH ou niveau haut). Il en va de même pour un transistor employé en commutation électronique, qui fournit un signal de type numérique.

Entrées analogiques

Les entrées analogiques de la carte peuvent être alimentées par des capteurs qui ont des caractéristiques aussi bien analogiques que numériques.

Prenons le cas d'un capteur de température, dont la résistance varie en fonction de la température ambiante. Ce capteur délivre à l'entrée une certaine tension, dont la valeur peut permettre de calculer la température réelle. Chaque valeur de tension sera traduite en valeur de température et pourra éventuellement être affichée ou servir à commander un ventilateur pour assurer un meilleur refroidissement.

La sortie (OUTPUT)

Ce qui rentre doit sortir d'une manière ou d'une autre ; c'est dans la nature des choses. Il est donc logique que la carte Arduino soit pourvue d'un certain nombre de sorties, à l'aide desquelles des

commandes ou affichages sont exécutés. En entrée, on parle de capteur et, en sortie, d'actionneur comme un moteur ou un relais.

Sorties numériques

Vous pouvez, par exemple, utiliser les sorties numériques pour raccorder des indicateurs de signaux optiques qui reflètent des états internes. Il s'agit en général de diodes électroluminescentes (ou LED, *Light Emitting Diode*), qui sont connectées aux broches en question à travers une résistance appropriée. Bien entendu, une sortie numérique peut aussi commander un transistor, qui pourra piloter une charge plus importante que ce que le port Arduino serait à même de faire.

Sorties analogiques

Sur votre carte Arduino, les sorties analogiques ne sont pas une mince affaire. Il n'existe pas de port dédié, autrement dit configuré pour cet usage. Certains ports numériques prennent quasiment la fonction en charge et simulent un signal analogique généré par la modulation de largeur d'impulsion (MLI). Vous en saurez plus au moment de programmer une sortie analogique.

Ordre et obéissance

Sans logiciel, votre magnifique matériel informatique ne servirait à rien. Seul un logiciel intelligent lui donnera vie et lui permettra d'accomplir les tâches pour lesquelles il a été conçu. Ces dernières doivent cependant être confiées à votre microcontrôleur Arduino.

Fais ce que je te dis

La communication est assurée par des instructions données au microcontrôleur, que ce dernier comprend en raison de sa spécification et convertit en actions correspondantes. Voici une instruction pour bien comprendre de quoi je parle. Le sens n'a ici aucune importance.

```
pinMode(13, OUTPUT);
```

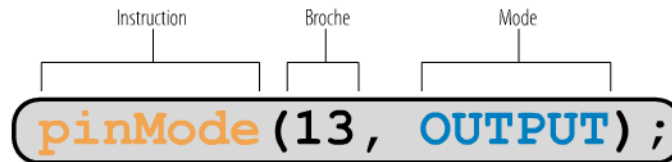
Si vous tapez cette instruction dans l'environnement de développement, la surbrillance de syntaxe entre en action et les mots-clés reconnus s'affichent en couleurs, instructions incluses. La vue d'ensemble est meilleure et vous voyez tout de suite si, par exemple, une instruction a été mal écrite.

Écrivez la ligne suivante.

```
pinMode(13, OUTPUT);
```

L'instruction n'est pas reconnue et elle est écrite en noir, ce qui doit vous mettre la puce à l'oreille. Par ailleurs, la structure de l'instruction `pinMode` mérite que l'on s'y attarde. À sa suite, vous pouvez voir des parenthèses. Ce sont des *arguments* qui sont transmis lors de l'exécution de l'instruction, à la manière d'un sac dans lequel vous enveloppez des objets pour les emporter.

Figure 3-44 ►
L'instruction `pinMode`



Les arguments sont des informations supplémentaires dont une instruction a besoin. Ici, ils indiquent que le port 13 doit servir de sortie.

Nous avons cependant oublié quelque chose de décisif : chaque instruction doit se terminer par un point-virgule. C'est le signe, pour le compilateur, que l'instruction est terminée et qu'une autre suit le cas échéant. Même si toutes les instructions n'ont pas besoin d'arguments, la paire de parenthèses reste nécessaire – dans ce cas, il n'y a simplement rien entre les deux.

Respectez aussi toujours les minuscules et les majuscules. Tout comme dans les langages de programmation C/C++, cette distinction est importante. De tels langages sont dits *case sensitive* ; autrement dit, `pinMode` n'est pas égal à `pinmode` !

Que se passe-t-il si une instruction a été mal formulée ?

Une instruction envoyée au microcontrôleur est toujours exécutée à moins qu'elle ait été mal rédigée. C'est pourquoi, vous devez vous familiariser avec le vocabulaire du microcontrôleur ou de l'environnement de développement – qui est apparenté à C++ – et le maîtriser. Bien évidemment, cela ne viendra pas du jour au lendemain !

C'est comme une langue étrangère : plus vous pratiquez, plus vite vous la maîtrisez. Si, par exemple, dans un e-mail à un interlocuteur étranger, vous orthographiez mal un mot, il se peut que le destinataire

comprenez quand même le sens de votre phrase. Avec un ordinateur, c'est différent : il ne veut rien savoir. Soit vous vous exprimez clairement et utilisez le mode d'écriture exact, soit il refuse catégoriquement l'instruction et se met en grève. Comment peut-il savoir ce que vous voulez dire ? On ne saurait lui prêter cette intelligence... Si une instruction est mal écrite ou si les minuscules et majuscules n'ont pas été respectées, une erreur se produit au niveau du compilateur. Par chance, on sait dans la plupart des cas de quoi il s'agit. On a en effet des indications sur l'endroit et la cause de cette erreur.

Les erreurs sont de trois types :

- erreurs de syntaxe ;
- erreurs logiques ;
- erreurs chronologiques.

L'erreur de syntaxe

Par chance, le compilateur détecte les erreurs de syntaxe. Elles sont par ailleurs faciles à localiser.

Regardons maintenant le message d'erreur suivant.



```
'pinmode' was not declared in this scope

rumpf.cpp: In function 'void setup()':
rumpf.pde:-1: error: 'pinmode' was not declared in this scope
```

J'ai écrit `pinMode` tout en minuscules. Bien évidemment, il s'agit d'une erreur que le compilateur a remarquée. En conséquence, il signale qu'il ne connaît pas l'instruction `pinmode`.

L'erreur logique

Les erreurs logiques sont plus problématiques car plus insidieuses. Elles ne donnent pas lieu à un message d'erreur puisque tout est en ordre du côté des instructions. Pourtant, quelque chose ne va pas : le sketch programmé ne se déroule pas comme vous l'aviez imaginé.

Le compilateur n'est ici pas en cause. Il peut s'agir, par exemple, d'une formule ou d'une valeur erronée que vous avez saisie quelque part, ou d'un port nécessairement de sortie dont vous avez fait une entrée. Les sources d'erreurs sont nombreuses et variées.



Nous verrons comment détecter ces erreurs quand nous aborderons le thème du déverminage. Il s'agit en l'occurrence d'une méthode qui sert à trouver des erreurs dans le programme.

L'erreur chronologique

Une erreur chronologique est un problème qui affecte d'abord la durée d'exécution du sketch. Tout est en ordre au niveau de la syntaxe et du compilateur, mais une bombe cachée attend son heure pour exploser. Tout peut très bien aller un temps et vous pensez alors que c'est bon, et un beau jour, plus rien...



Voici un exemple tiré du monde Windows. Imaginez que vous ayez stocké votre collection de musique MP3 sur un disque externe D : . Un programme de musique y accède régulièrement et exécute les morceaux stockés. Tout marche à merveille quand soudain, pour une raison quelconque, le disque ne répond plus, soit parce qu'il est en panne, soit que le câble USB est débranché. Le programme essaie toujours d'accéder aux fichiers de musique, car le programmeur, négligent, n'a pas cru bon de doter l'appel au lecteur d'un traitement des erreurs. L'accès demandé n'aboutit pas et le programme est irrémédiablement muet. Cela semble tiré par les cheveux, mais plus d'un programme réagit en s'interrompant simplement au lieu d'émettre un message d'erreur. Ces interruptions intempestives ont de quoi irriter.



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- Arduino ;
- Freeduino ;
- Arduino Projects.

Les bases de l'électronique

Pour utiliser une carte Arduino, il est préférable d'avoir quelques connaissances en électronique. Ce chapitre en rappelle les principales notions :

- les grands principes de l'électronique ;
- les notions de courant, tension et résistance ;
- la loi d'Ohm ;
- le circuit fermé ;
- les composants passifs et actifs ;
- les principaux composants électriques et électroniques ;
- le circuit intégré.

Mais si vous connaissez déjà les bases de l'électronique, vous pouvez passer au chapitre suivant.

Vous avez dit électronique ?

On entend souvent dire que notre monde ne serait pas ce qu'il est aujourd'hui sans l'électronique, qu'on retrouve désormais dans tous les domaines. Mais, en définitive, que sait-on vraiment de l'électronique ?

Dans le terme électronique, on retrouve le mot « électron ». Les électrons circulent dans un conducteur, par exemple un fil de cuivre, et leur mouvement donne naissance à un courant électrique. Le but est de faire suivre à ce dernier certains trajets, d'établir ou d'interrompre sa circulation, autrement dit d'en garder le contrôle. Quand on y parvient, il est possible de réaliser des choses fantastiques.

Ici, on maîtrise un phénomène qu'on ne voit pas à l'œil nu et qui n'est reconnaissable qu'aux effets produits. Nous modélisons les processus

les plus divers, que nous commandons ou contrôlons à volonté. Sur un espace microscopique, nous amenons les électrons à effectuer les trajets souhaités. En fait, c'est ça l'électronique.

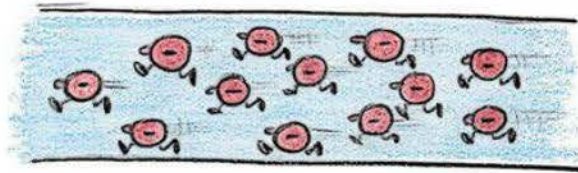
Le flux d'électrons

Les électrons sont d'infimes particules élémentaires qui circulent à vitesse élevée dans un conducteur. Voici quelques-unes de ces propriétés caractéristiques :

- charge négative ($-1,602176 \times 10^{-19} \text{ C}$) ;
- pratiquement sans masse ($9,109382 \times 10^{-31} \text{ kg}$) ;
- stable (durée de vie supérieure à 10^{24} ans).

Je n'ai pas hésité à filmer pour vous, à l'aide d'une caméra ultraperfectionnée, un conducteur traversé par un courant pour que vous puissiez visualiser ces minuscules particules (voir figure 4-1). Elles vont toutes dans le même sens et sont responsables de la circulation du courant.

Figure 4-1 ►
Électrons parcourant
un conducteur en cuivre

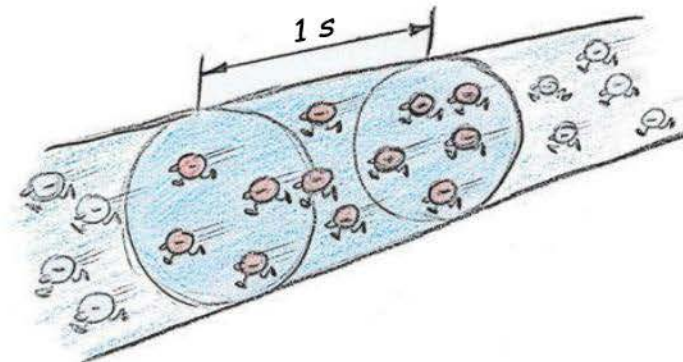


Quand j'ai mentionné la charge négative de l'électron, vous aurez peut-être remarqué que la valeur $-1,602176 \times 10^{-19} \text{ C}$ était très petite. L'unité de mesure est le coulomb (C). La charge Q circulant en un temps donné à travers un conducteur ayant une certaine section est exprimée à l'aide de la relation suivante :

$$Q = I \cdot t$$

Il s'agit du produit entre l'intensité du courant I (en ampères) et le temps t (en secondes).

Figure 4-2 ►
Parcours à travers un conducteur en
cuivre sur une durée d'1 seconde



Sur cette vue de la migration des électrons à travers un conducteur en cuivre, j'ai marqué un tronçon que les électrons parcourent en 1 seconde. On peut retenir qu'une charge d'un coulomb a été transportée lorsqu'il y a eu le passage d'un courant d'un ampère pendant 1 seconde.

J'ai déjà utilisé tellement de fois le mot courant que je me dois maintenant d'en dire plus sur cette grandeur physique.

Le courant

Comme vous avez pu le voir dans la dernière formule, la charge et le courant sont reliés entre eux. On peut assimiler le courant à la circulation d'une charge électrique. Plus la charge circulant par unité de temps est importante, plus le courant électrique I est élevé.

$$I = \frac{Q}{t}$$

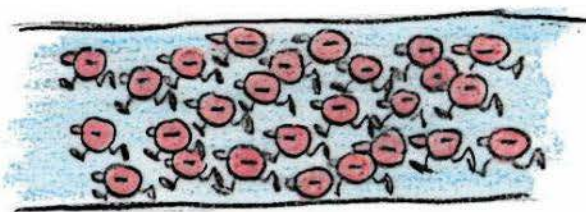
La figure 4-3 représente un faible flux d'électrons. Seuls quelques porteurs de charge circulent par unité de temps ; on aura donc un faible courant.



◀ **Figure 4-3**

Faible flux d'électrons : un petit nombre d'électrons génèrent peu de courant électrique.

À l'inverse, la figure 4-4 représente beaucoup de porteurs de charge circulant par unité de temps, qui donnent naissance à un fort courant dans le conducteur.



◀ **Figure 4-4**

Flux d'électrons élevé : un grand nombre d'électrons génèrent un courant électrique important.

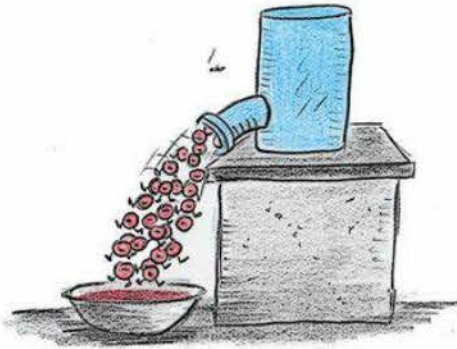
L'intensité de courant I est mesurée en ampères (A) ; 1 ampère constitue une très forte intensité pour des microcontrôleurs. La charge maximale d'une sortie numérique de votre carte Arduino est de

l'ordre de 40 mA (ou 40 milliampères). Un milliampère représente un millième d'ampère ($1\,000\text{ mA} = 1\text{ A}$).

La tension

Si l'on jette un œil aux dessins précédents schématisant des électrons parcourant un conducteur, il y a quelque chose que nous ne prenons pas en considération : pourquoi se déplacent-ils ? Notre monde est ainsi fait : chaque action a sa raison ou sa finalité correspondante ; nos gestes sont toujours commandés ou motivés par quelque chose. Il en va de même pour les électrons : ils vont tous dans le même sens et vers le même but. Une force motrice doit par conséquent agir. La comparaison de ce phénomène avec de l'eau qui s'écoule est très parlante. C'est cette analogie que j'ai utilisée dans ce chapitre.

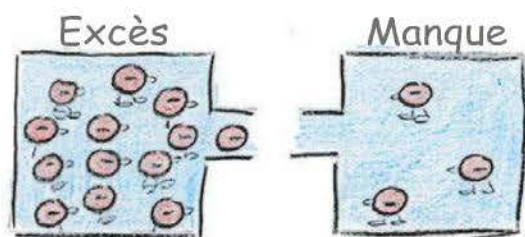
Figure 4-5 ►
Les électrons se déplacent en raison d'une différence de potentiel.



Ici, je devrais plutôt parler de différence de charge et non de différence de potentiel. Les charges électriques tendent toujours à compenser les différences de charge.

Prenons, par exemple, une pile chargée. Elle a deux bornes (ou pôles) entre lesquelles il existe une différence de charge. L'une de ces bornes présente un excès de charge et l'autre un manque de charge. Faute de liaison électrique entre les deux pôles, aucune charge ne peut se déplacer pour rétablir l'équilibre, si bien qu'aucun courant ne circule.

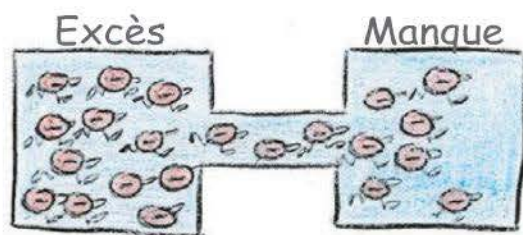
La tension U se mesure en volts (V) et sert à estimer la différence de potentiel entre deux points.



◀ **Figure 4-6**

La différence de charge ne peut être compensée, faute de liaison.

L'absence de liaison entre les deux potentiels empêche toute égalisation des charges et aucun courant ne circule.



◀ **Figure 4-7**

Une compensation de la différence de charge se produit.

C'est seulement quand une liaison est établie que les porteurs de charge peuvent se déplacer et qu'un courant peut circuler.

Combien de temps circule le courant jusqu'à ce que le côté gauche soit vide et le côté droit plein ?

Le courant circule tant que la charge n'est pas équilibrée, autrement dit jusqu'à ce que les deux pôles aient autant de porteurs de charge l'un que l'autre. Si tous les électrons étaient passés du côté droit, il y aurait eu de nouveau un déséquilibre, et l'opération se serait répétée dans l'autre sens. Par ailleurs, une séparation de charge ne peut être rétablie que par un apport d'énergie après une égalisation de charge. Or, ce n'est pas le cas, et c'est donc aussi pour cette raison qu'une pile normale est vide après une égalisation de charge.

J'ai souvent entendu dire qu'il existe différentes formes de courant. Il y a le courant continu et le courant alternatif. Pourrais-je en savoir plus là-dessus ?

Bien sûr ! Votre carte Arduino fonctionne avec du courant continu, caractérisé par une intensité et une direction qui ne varient pas au fil du temps. En électronique, ce type de courant est symbolisé par les lettres DC (*Direct Current*) ou, en France, par CC (courant continu). La figure 4-8 montre l'évolution d'une tension continue au cours du



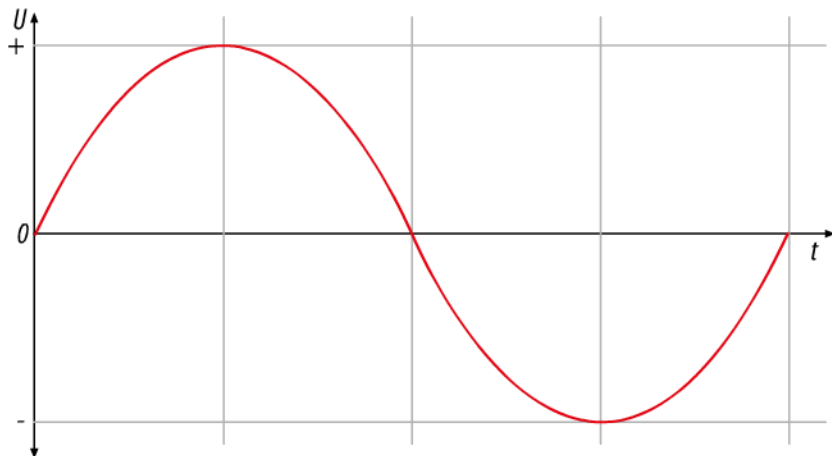
temps. Le courant alternatif est, quant à lui, noté AC (*Alternating Current*) ou CA (courant alternatif).

Figure 4-8 ►
Évolution d'une tension continue
au cours du temps



L'axe horizontal des abscisses (X) représente le temps t et l'axe vertical des ordonnées (Y) indique la tension U . On voit que la valeur de tension ne varie pas au cours du temps. Analysons maintenant l'évolution de la tension alternative, représentée ici par une sinusoïde.

Figure 4-9 ►
Évolution d'une tension alternative
au cours du temps



La valeur de la tension varie en permanence et oscille entre deux valeurs limites, l'une positive et l'autre négative. Notez que la lettre U indique qu'on a affaire à une tension. Sachez aussi qu'il existe, dans la plupart des cas, une relation de proportionnalité entre la tension et le courant.

La notion de résistance

Les électrons circulant dans un conducteur peuvent avoir plus ou moins de mal à arriver au bout. Sur leur chemin, ils rencontrent en effet des résistances très différentes les unes des autres. On peut établir une classification des matériaux en fonction de leur conductibilité :

- isolants (très haute résistance). Par exemple : la céramique ;
- mauvais conducteurs (haute résistance). Par exemple : le verre ;
- bons conducteurs (faible résistance). Par exemple : le cuivre ;
- très bons conducteurs (supraconductivité à de très basses températures où la résistance électrique tend vers 0) ;
- semi-conducteurs (la résistance peut être commandée). Par exemple : silicium ou germanium.

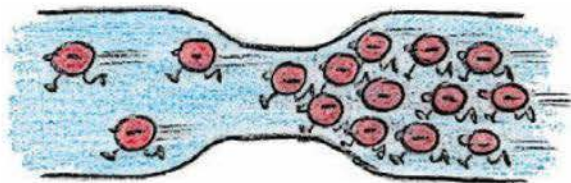
Il existe ainsi deux grandeurs électriques, inversement proportionnelles : la résistance R – dont la valeur est exprimée en ohms (Ω) ; elle se déduit à partir du code couleur peint directement sur son corps – et la conductance G .

Plus la résistance est élevée, plus la conductance est faible, et inversement. La relation qui lie une résistance à une conductance est donnée par :

$$R = \frac{1}{G}$$

La résistance est la valeur inverse de la conductance. Une résistance élevée s'oppose au passage des électrons ; on peut la comparer à un goulet d'étranglement.

Lorsque la résistance est élevée, le courant qui la traverse est faible. Imaginez que vous couriez sur une surface lisse. Avancer ne devrait pas vous poser trop de problèmes. En revanche, si vous essayez de courir dans du sable en gardant la même allure, c'est fatigant. Vous dépensez de l'énergie sous forme de chaleur et votre vitesse diminue. Il en va de même pour des électrons qui doivent traverser par exemple du verre (isolant) au lieu du cuivre (conducteur).



◀ **Figure 4-10**
Résistance freinant le flux
d'électrons

Le frottement plus intense des électrons, par exemple sur la paroi externe ou bien entre eux, dégage une énergie sous forme de chaleur que la résistance transmet à l'extérieur.

Il existe plusieurs types de résistances (par exemple, à couche de carbone ou à couche métallique). Dans la plupart des circuits électroniques, elles sont utilisées pour limiter l'intensité des courants.



Lorsque j'examine de plus près le schéma du flux d'électrons, j'ai l'impression que les électrons sur le côté droit avancent plus vite que ceux du côté gauche.

Je comprends bien ce que tu veux dire et je sais ce qui a pu te faire arriver à cette conclusion erronée. Le courant qui circule dans un circuit fermé est toujours le même. Certes, il est influencé par la résistance illustrée ici. Mais, au final, le courant avant ou après la résistance est toujours le même. À chaque unité de temps, le même nombre d'électrons passe dans le conducteur ou dans la résistance. Mais tu as bien fait de poser la question.

Nous avons maintenant fait le tour de toutes les grandeurs électriques nécessaires à la compréhension d'une loi très importante, la loi d'Ohm.

La loi d'Ohm

La loi d'Ohm décrit le rapport entre la tension U aux bornes d'une résistance R et le courant I qui la traverse. Voici la relation qui la définit :

$$R = \frac{U}{I}$$

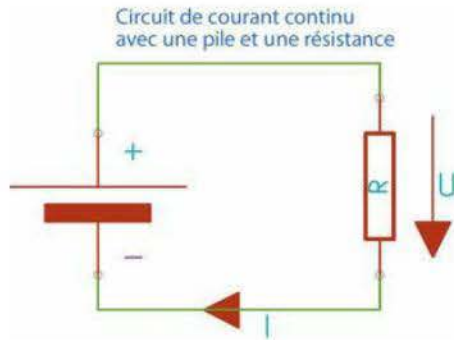
La résistance s'obtient en divisant la tension par le courant ; comme dit précédemment, elle s'exprime en ohms (Ω) et elle est désignée par la lettre R .

Nous utiliserons cette loi pour calculer la résistance de limitation pour une diode électroluminescente qui ne saurait fonctionner sans elle.

Le circuit fermé

Vous savez maintenant qu'une circulation de courant ne peut aboutir qu'à condition que le circuit soit fermé et qu'une force électromotrice

agisse. C'est le cas aussi pour les électrons et également pour des molécules d'eau par exemple. Étudions le schéma d'un circuit simple.



◀ **Figure 4-11**
Circuit fermé simple avec une pile et une résistance

Sur le côté gauche du plan de câblage se trouve une source de tension continue sous la forme d'une pile dont les deux pôles + et - sont raccordés à une résistance. Le circuit est ainsi fermé et un courant I peut circuler dès lors que la pile est chargée. Cette circulation de courant engendre une certaine chute de tension U aux bornes de la résistance R . Je vais maintenant expliquer les rapports qui existent entre U , R et I .

Ces grandeurs sont précisément celles de la loi d'Ohm. Je suppose que nous allons l'appliquer, non ?

Tout juste Arduus ! Nous allons nous livrer à un petit exercice avec les valeurs suivantes.

- La tension U de la pile est de 9 V.
- La résistance R a une valeur de 1 000 Ω (1 000 Ω = 1 k Ω = 1 kilo-ohm).

Question : quelle est la valeur du courant I qui traverse la résistance et bien entendu la pile ?

Si on tire I de la formule :

$$R = \frac{U}{I}$$

on obtient :

$$I = \frac{U}{R}$$

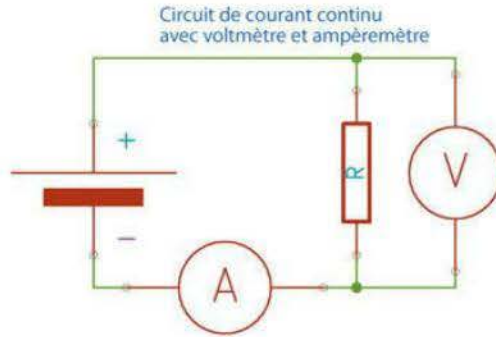


En remplaçant les différentes grandeurs par leur valeur numérique, on obtient le résultat suivant :

$$I = \frac{U}{R} = \frac{9 \text{ V}}{1\,000 \, \Omega} = 0,009 \text{ A} = 9 \text{ mA}$$

Un courant I de 9 mA circule donc dans le circuit. Si vous avez monté un tel circuit, cette valeur peut être mesurée grâce à un multimètre. La mesure d'une tension s'effectue en branchant cet appareil en parallèle sur le composant et la mesure d'un courant I s'obtient par sa mise en série (en position ampèremètre) avec le composant en question.

Figure 4-12 ►
Mesure des valeurs du courant
et de la tension

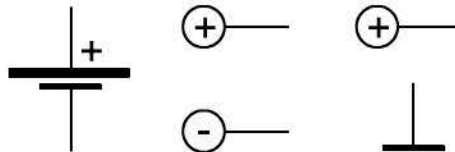


Chaque conducteur a bien une certaine résistance, donc probablement aussi l'ampèremètre. La mesure de l'intensité du courant n'en est-elle pas faussée ?

Bien dit Ardus ! C'est vrai et c'est pour cette raison que les appareils de mesure réglés sur la mesure de l'intensité du courant ont une résistance interne très faible. Le résultat de la mesure n'est alors pratiquement pas influencé.

Dans les circuits représentés, j'ai utilisé le symbole de la pile pour indiquer la source de tension. Cependant, il en existe d'autres variantes selon les schémas (voir figure 4-13).

Figure 4-13 ►
Divers symboles de sources
de tension



Le symbole de gauche représente une pile ; celui du centre est aussi bien utilisé pour des piles que pour d'autres alimentations en courant continu ; sur celui de droite, le pôle négatif a été remplacé par le symbole de masse. Ce dernier est surtout employé dans les schémas

complexes, pour éviter d'avoir à représenter le fil de la borne négative sur l'ensemble du circuit.

Nous reviendrons plus loin dans ce chapitre sur les circuits électroniques de base, dont je souhaite encore préciser certains détails. Je pense qu'il est maintenant temps de vous débrouiller un peu tout seul. Ne craignez rien, la solution de l'énigme vous sera donnée d'ici la fin de ce chapitre !

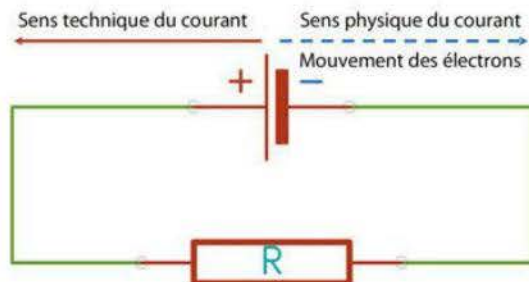


Attention !

En électronique, il existe deux conventions de sens du courant, qui sont opposées. Vous devez par conséquent savoir en quoi réside la différence.

N'importe quoi ! Voilà maintenant que les électrons peuvent choisir dans quel sens ils parcourent le conducteur. C'est l'anarchie !

Non Arduus, vous pouvez être tranquille car le courant ne circule en vérité que dans un seul sens. La cause de cette confusion – si on peut l'appeler ainsi – remonte à l'ignorance de nos précurseurs. Avant que les scientifiques ne puissent se faire une idée exacte de la théorie du mouvement des électrons, ils avaient tout bonnement décrété que le pôle positif présentait un excès d'électrons et le pôle négatif un manque. Sur la base de cette définition, les électrons émigrent du pôle positif vers le pôle négatif quand une liaison conductrice est établie entre les deux pôles. Les recherches ultérieures devaient apporter la lumière : les électrons nous donnaient tort et circulaient précisément dans le sens opposé. Mais comme une mauvaise habitude ne se perd pas si facilement et puisque tout avait été fait jusqu'alors en dépit du bon sens, une parade a été trouvée : le sens incorrect de jadis serait appelé sens technique du courant. Le sens correct actuel prendrait le nom de sens physique du courant, qui indique le déplacement proprement dit des électrons.



Principaux composants électroniques

Le composant électronique le plus simple est la résistance, mais il en existe beaucoup d'autres. Dans ce chapitre, nous nous limiterons aux composants de base, qui peuvent être rangés en deux catégories : les composants passifs et les composants actifs.

Composants passifs et actifs

Composants passifs

Les composants passifs sont des composants qui n'ont aucune action d'amplification sur le signal concerné. Dans cette catégorie, on trouve :

- les résistances ;
- les condensateurs ;
- les inductances (bobines).

Composants actifs

Les composants actifs ont de l'influence sur un signal. Ils peuvent agir, par exemple, sur son amplitude en l'augmentant. En voici quelques-uns :

- les transistors ;
- les thyristors ;
- les photocoupleurs.

La résistance fixe

La résistance fixe, appelée plus simplement résistance, est un composant dont la valeur est constante (même si elle demeure sensible aux variations de température).

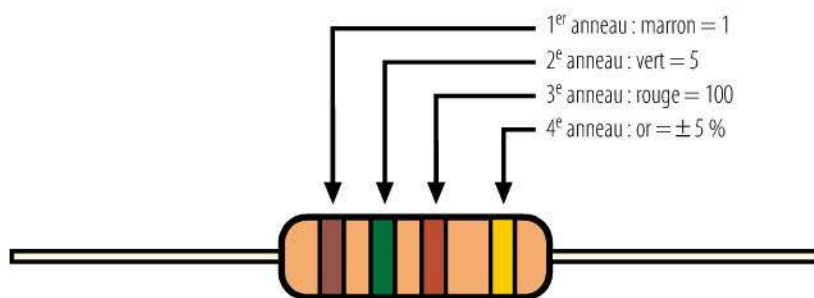
Un code couleur a été défini pour connaître la valeur d'une résistance : en effet, elles sont trop petites pour recevoir un marquage explicite et elles ont par ailleurs des tailles différentes. Sachez que la taille d'une résistance est fonction de la puissance maximale qu'elle peut dissiper.



◀ **Figure 4-14**
Plusieurs types de résistances

Quand on débute, il semble souvent compliqué de déterminer la valeur de la résistance, et on ne sait pas toujours dans quel sens il faut lire les anneaux de couleur. Aussi vais-je vous donner quelques astuces pour y arriver.

Compte tenu de la précision du processus de fabrication, la valeur réelle d'une résistance peut légèrement différer de celle indiquée sur le composant. Aussi les anneaux précisant la valeur de la résistance sont complétés par un anneau dit de tolérance, argenté ou doré. L'anneau de tolérance se situe à droite des trois anneaux de couleur traduisant la valeur de la résistance (voir figure 4-15).



◀ **Figure 4-15**
Détermination de la valeur de la résistance au moyen du code couleur

Voici la valeur de la résistance obtenue quand on écrit ces chiffres les uns à côté des autres.

1 ^{er} chiffre	2 ^e chiffre	Multiplicateur	Tolérance	Valeur
1	5	100	$\pm 5\%$	$1\,500\,\Omega = 1,5K$

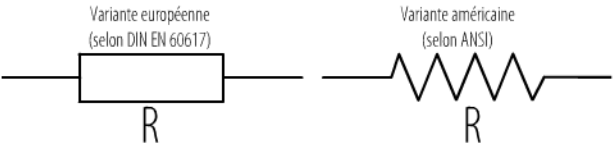
Le tableau 4-1 donne tous les codes couleurs avec leurs valeurs correspondantes.

Tableau 4-1 ►
Code couleur pour les résistances

Couleur	1 ^{er} anneau (1 ^{er} chiffre)	2 ^e anneau (2 ^e chiffre)	3 ^e anneau (multiplicateur)	4 ^e anneau (tolérance)
Noir	x	0	$10^0 = 1$	
Marron	1	1	$10^1 = 10$	$\pm 1 \%$
Rouge	2	2	$10^2 = 100$	$\pm 2 \%$
Orange	3	3	$10^3 = 1\,000$	
Jaune	4	4	$10^4 = 10\,000$	
Vert	5	5	$10^5 = 100\,000$	$\pm 0,5 \%$
Bleu	6	6	$10^6 = 1\,000\,000$	$\pm 0,25 \%$
Violet	7	7	$10^7 = 10\,000\,000$	$\pm 0,1 \%$
Gris	8	8	$10^8 = 100\,000\,000$	$\pm 0,05 \%$
Blanc	9	9	$10^9 = 1\,000\,000\,000$	
Or			$10^{-1} = 0,1$	$\pm 5 \%$
Argent			$10^{-2} = 0,01$	$\pm 10 \%$

Voici comment sont représentées les résistances dans les schémas électriques.

Figure 4-16 ►
Symboles de la résistance fixe dans un schéma électrique



- Il peut s'agir d'un rectangle selon la représentation DIN (*Deutsche Industrie Norm*) avec les raccordements électriques à gauche et à droite. La valeur de la résistance peut se trouver soit carrément dans le symbole, soit juste au-dessus ou en dessous.
- Il peut aussi s'agir de la variante américaine selon l'ANSI (*American National Standards Institute*), dans laquelle la résistance est représentée par une ligne en zigzag. Ce symbole date du temps où les résistances consistaient en un enroulement de fil plus ou moins épais. Le symbole de l'ohm est en principe absent et seul le nombre est indiqué si la valeur est inférieure à 1 k Ω (1 000 ohms), suivi éventuellement d'un K pour kilo si la valeur est supérieure ou égale à 1 k Ω ou d'un M pour méga si la valeur est supérieure ou égale à 1 M Ω . Voici quelques exemples.

Tableau 4-2 ►
Différentes valeurs de résistance

Valeur	Marquage
330 Ω	330
1 000 Ω	1K
4 700 Ω	4,7K ou 4K7
2,2 M Ω	2,2M

Pour que la dissipation de la puissance P ne devienne pas un problème, celle-ci peut être calculée à l'aide de la formule suivante :

$$P = U \cdot I$$

L'unité de puissance est le watt (W). Les résistances que nous utilisons pour nos expérimentations sont toutes à couche de carbone aggloméré, avec une dissipation d'énergie admise de $1/4$ de watt.

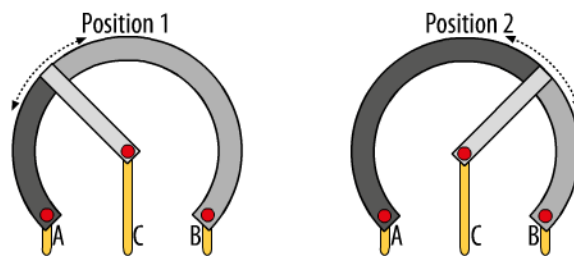
La résistance variable

Outre les résistances fixes, il existe toute une gamme de résistances variables. Par exemple, pour régler le volume de votre radio, vous avez besoin d'une résistance dont la valeur varie en fonction de l'angle de rotation de l'axe de commande.

Résistance ajustable et potentiomètre

Il existe deux sortes de résistances réglables manuellement : la résistance ajustable et le potentiomètre (ou potard en langage familier) – la valeur de leur résistance varie en fonction de la rotation de leur axe mobile. En principe, elles fonctionnent toutes deux de la même manière.

La figure 4-17 montre le schéma de ce type de dispositif. Sur un support non conducteur se trouve une couche de matériau résistant présentant des contacts (A et B) à ses deux extrémités. Entre ces deux contacts, il y a une résistance de valeur fixe. Pour que cette dernière puisse être modifiée, on a un troisième contact mobile (C) capable de se déplacer dans les deux sens sur la couche résistante. Entre ce curseur C et l'un des contacts (A ou B), la valeur de la résistance est variable.



◀ **Figure 4-17**

Schéma d'une résistance ajustable ou d'un potentiomètre dans deux positions différentes

Dans la position 1, la résistance entre les points A et C est inférieure à celle entre les points C et B . Dans la position 2 en revanche, le curseur se trouve plus loin vers la droite ; la valeur de résistance entre les points A et C a augmenté et celle entre les points C et B a diminué.

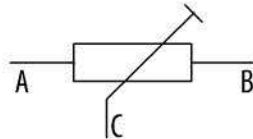
La résistance ajustable

La résistance ajustable sert de résistance à réglage définitif. La plupart du temps, elle est soudée directement sur une carte pour être ajustée à l'aide d'un petit tournevis d'horloger pour calibrer le circuit. Généralement, la valeur de la résistance n'est plus jamais modifiée par la suite.



Figure 4-18 ►

Symbole de la résistance ajustable



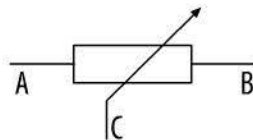
Le potentiomètre

Le potentiomètre est une résistance réglable qui permet d'ajuster, par exemple, le volume d'une radio ou bien l'intensité d'un dispositif lumineux. Son curseur mobile peut être déplacé au moyen d'un axe sortant du boîtier. Il est donc très facile de régler manuellement la valeur de la résistance.



Figure 4-19 ►

Symbole du potentiomètre



La photorésistance

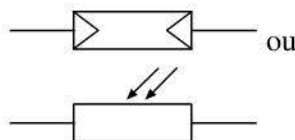
La photorésistance est également appelée LDR (*Light Depending Resistor*). La valeur de la résistance de ce composant évolue en fonction de l'intensité lumineuse qui lui est appliquée. Plus la lumière incidente est forte, plus sa résistance est faible.

La photorésistance va nous permettre de faire des essais intéressants sur un servomoteur, qui sera censé suivre une source lumineuse et se diriger toujours vers le point le plus lumineux.

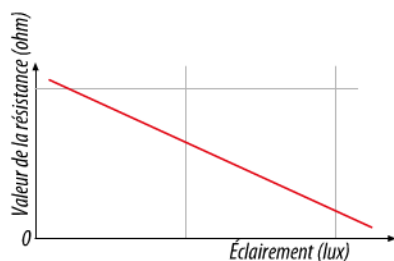


Figure 4-20 ►

Symboles de la photorésistance



La courbe caractéristique d'une LDR traduit son comportement en fonction de la variation de son éclairement. L'éclairement s'exprime en lux (lx).



◀ **Figure 4-21**

Courbe caractéristique d'une LDR

Les domaines d'application d'une LDR sont variés. En voici quelques-uns :

- comme interrupteur crépusculaire pour déclencher une source lumineuse supplémentaire à la nuit tombante telle que des réverbères ou un éclairage intérieur de véhicule ;
- pour mesurer l'intensité lumineuse avant de prendre des photos ;
- comme capteur dans des barrières lumineuses pour, par exemple, des portes d'ascenseur ou des contrôles d'accès à des zones de sécurité.

La plage de résistance de la LDR dépend du matériau employé et présente approximativement une résistance dans l'obscurité comprise entre 1 et 10 M Ω . Une intensité d'éclairement de 1 000 lux (lx) environ engendre une résistance comprise entre 75 et 300 Ω .

La résistance thermosensible ou thermistance

La valeur de la résistance thermosensible varie en fonction de la température. Il existe deux types de thermistances :

- NTC (*Negative Temperature Coefficient* ou résistance à coefficient de température négatif) ;
- PTC (*Positive Temperature Coefficient* ou résistance à coefficient de température positif).

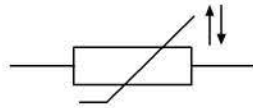


NTC

La résistance d'une thermistance NTC diminue (et la conductivité augmente) lorsque la température augmente.

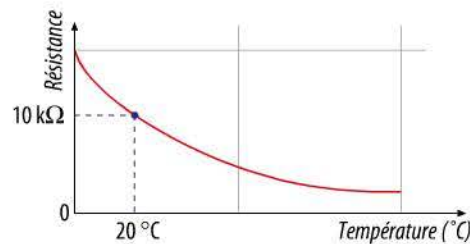
Sa forme ressemble à celle d'un condensateur céramique, ce qui peut parfois donner lieu à des confusions. Une inscription, par exemple 4K7, laisse pourtant présumer de la valeur de résistance, et l'appellation « Thermistor NTC 4K7 » permet de l'identifier formellement.

Figure 4-22 ►
Symbole de la thermistance NTC



Une NTC se reconnaît à sa courbe caractéristique.

Figure 4-23 ►
Courbe caractéristique
d'une thermistance NTC

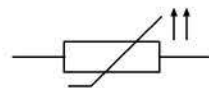


On remarque qu'il s'agit bel et bien d'une courbe et non d'une droite comme pour la LDR. La principale caractéristique de cette résistance est sa valeur de référence, R_{20} , qui est annoncée à une température ambiante de 20 °C. Sur la courbe, j'ai opté pour une valeur fictive de 10 kΩ.

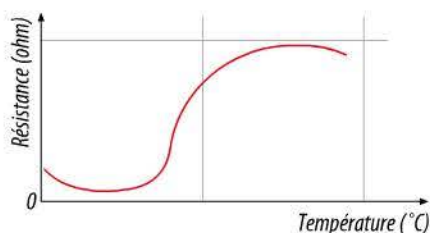
PTC

La résistance de la thermistance PTC varie inversement à celle d'une thermistance NTC. La conductance d'une PTC diminue (et la résistance augmente) en fonction de la température.

Figure 4-24 ►
Symbole de la PTC
(thermistance PTC)



La courbe caractéristique d'une PTC est l'inverse de celle de la NTC, avec quelques particularités supplémentaires. En effet, elle peut posséder un minimum dans la région des températures basses et un maximum dans celle des températures élevées.



◀ **Figure 4-25**
Courbe caractéristique d'une PTC

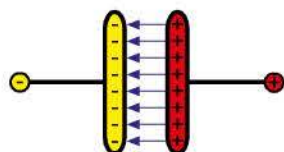
Le tableau 4-3 résume le comportement des deux types de résistances sensibles à la température (NTC et PTC).

Type	Température	Résistance	Courant
NTC	↑	↓	↑
	↓	↑	↓
PTC	↑	↑	↓
	↓	↓	↑

◀ **Tableau 4-3**
Comportement des NTC et PTC à différentes températures

Le condensateur

Le condensateur est un composant formé de deux plaques conductrices parallèles, séparées par un diélectrique. Si, par exemple, une tension continue est appliquée entre les deux plaques, il se crée entre elles un champ électrique.



◀ **Figure 4-26**
Champ électrique (représenté par des flèches) entre les deux plaques d'un condensateur

Les deux plaques se trouvent à une certaine distance l'une de l'autre et sont isolées entre elles par une couche isolante : le diélectrique. Une fois le condensateur chargé, le champ électrique persiste même si l'alimentation est coupée. Les deux plaques emmagasinent donc la quantité de charge fournie Q , qui est exprimée en coulomb (C).

$$Q = I \cdot t$$

Dans ce cas, le condensateur se comporte comme une pile chargée.



Attention !

Un condensateur chargé ne doit jamais être court-circuité et doit toujours être déchargé au moyen d'une résistance appropriée.

La quantité de charge que le condensateur peut emmagasiner dépend de deux facteurs :

- sa capacité totale C , qui est mesurée en farads (F) ;
- la tension d'alimentation U qui lui est appliquée.

Nous pouvons retenir que la quantité de charge Q d'un condensateur est d'autant plus grande que la capacité ou la tension est élevée. La formule suivante relie les trois grandeurs :

$$Q = C \cdot U$$

Exemple de calcul : soit un condensateur avec une capacité C de $3,3 \mu\text{F}$, auquel on applique une alimentation de 9 V . Quelle est la charge totale Q ?

$$Q = C \cdot U = 3,3 \mu\text{F} \cdot 9 \text{ V} = 2,97 \cdot 10^{-5} \text{ C}$$

La capacité d'un condensateur est en règle générale bien inférieure à un farad. Sa valeur se situe plutôt dans les plages suivantes :

- μF (10^{-6}) – microfarad ;
- nF (10^{-9}) – nanofarad ;
- pF (10^{-12}) – picofarad.

Il existe différents types de condensateurs ; en voici quelques-uns.

Condensateurs non polarisés

- Condensateur céramique
- Condensateur à film plastique
- Condensateur à film papier métallisé

Condensateurs polarisés

- Condensateurs électrolytiques

Sont représentés ci-contre un condensateur électrolytique (à gauche) et un condensateur céramique (à droite). Comme vous pouvez le voir, la différence de taille est énorme.

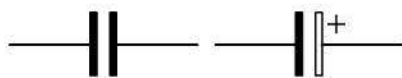


Les condensateurs non polarisés peuvent servir aussi bien dans des circuits de courant continu que dans des circuits de courant alternatif, alors que les condensateurs polarisés comme le condensateur électrolytique ne peuvent être employés que dans des circuits de courant continu et sous polarité correcte.

Le mode de fonctionnement des condensateurs me paraît clair, mais je ne vois pas bien où et à quoi ils peuvent servir.

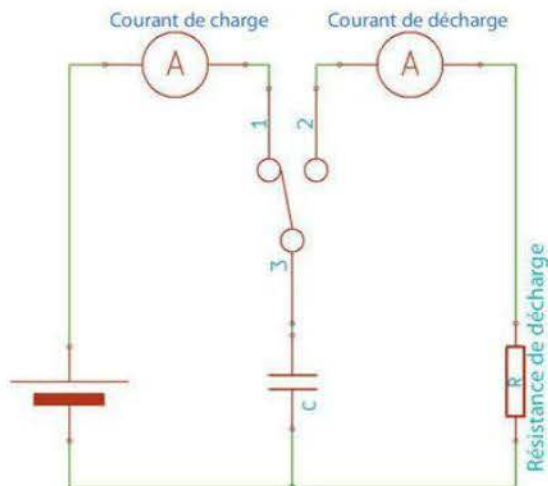
Les domaines d'utilisation sont extrêmement variés. En voici quelques-uns :

- pour lisser ou stabiliser une tension. Quand, par exemple, un composant complexe comme le circuit intégré a besoin d'une alimentation stable pour éviter de perdre des données, un condensateur est connecté entre les bornes + et - du boîtier du circuit. Ceci dans le but de maintenir une tension constante d'alimentation rendant ainsi imperceptible toute fluctuation de l'alimentation générale ;
- pour coupler des circuits à plusieurs étages pour des circuits de minuterie qui, au bout d'un certain temps, ferment ou ouvrent un contact de relais ;
- pour des timers, qui envoient des impulsions à intervalles réguliers à une sortie.



◀ **Figure 4-27**
Symboles représentant un condensateur normal (à gauche) et un condensateur polarisé électrolytique (à droite)

Voyons un peu comment se comporte un condensateur relié à une pile (voir figure 4-28).

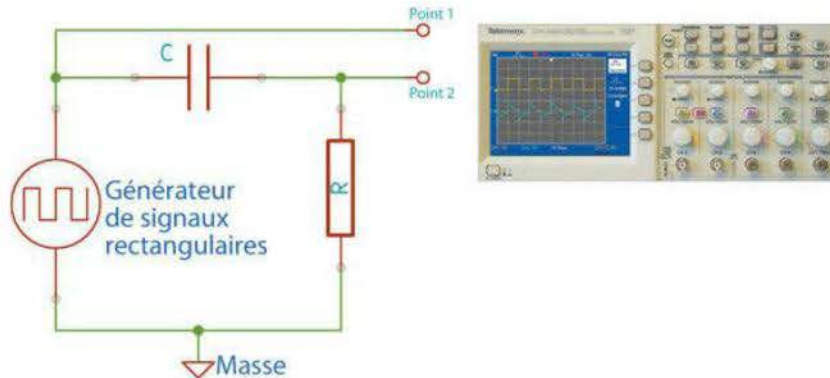


◀ **Figure 4-28**
Circuit de charge et de décharge d'un condensateur

Dans ce circuit, un condensateur est chargé par une pile quand l'inverseur est en position 1. Quand il est dans la position 2, le condensateur C est court-circuité par la résistance R et commence à se

décharger. On peut ainsi mesurer aussi bien le courant de charge que celui de décharge à l'aide des deux ampèremètres. Tout ceci n'est pour vous que pure théorie, aussi ai-je conçu un circuit où l'opération de commutation est automatique et électronique. La source de tension n'est pas une pile mais un générateur de signaux rectangulaires. La tension évolue à intervalles réguliers entre deux valeurs U_{max} et 0 V.

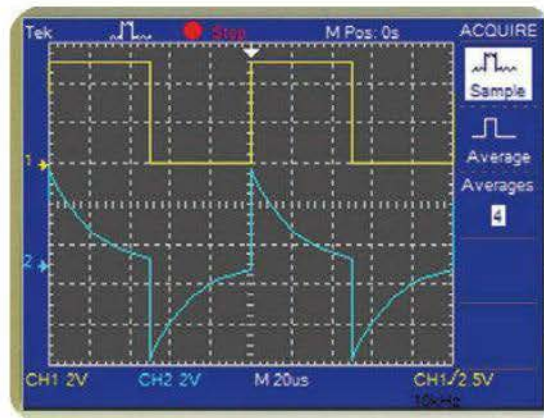
Figure 4-29 ►
Charge et décharge
d'un condensateur au moyen
d'un générateur de signaux
rectangulaires



J'ai relié aux deux points de mesure 1 et 2 un oscilloscope à deux canaux qui vont permettre de représenter les variations des tensions aux divers points. Le point de mesure 1 est relié au canal 1 (courbe jaune) et se situe directement à la sortie du générateur de signaux rectangulaires. Le point de mesure 2 est relié au canal 2 (courbe bleue) et présente les variations de la tension aux bornes de la résistance R (sortie du condensateur C).

Voyons maintenant de quelle manière un signal rectangulaire traverse le condensateur. La figure 4-30 vous montre les courbes d'évolution des tensions aux points 1 et 2.

Figure 4-30 ►
Signaux à l'entrée et à la sortie
du condensateur (avec Multisim)



Comment interpréter l'oscillogramme ? Quand le niveau de tension d'entrée passe de 0 à 5 V (courbe jaune), le condensateur transmet

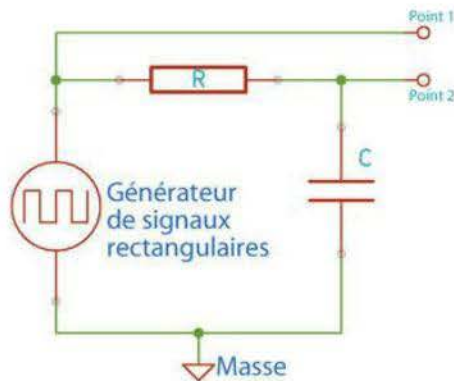
instantanément cette variation (courbe bleue). Ce dernier est assimilé à un court-circuit pour les variations rapides. Lorsque le niveau de la tension d'entrée est à 5 V pendant une certaine durée, le condensateur se charge et sa résistance augmente. Vous voyez que la courbe bleue du bas s'aplatit progressivement et revient presque à 0 V.

Un condensateur chargé constitue un circuit ouvert pour le courant continu. Le condensateur ne laisse pas passer le courant continu. Quand le signal d'entrée passe de 5 V à 0 V, le condensateur transmet cette variation vers la sortie et commence à se décharger à travers la résistance. Cette fois, le courant qui circule est dans le sens inverse de celui de la charge (courbe bleue). Le courant de décharge va diminuer jusqu'à pratiquement atteindre la valeur zéro. La tension aux bornes de la résistance, représentée par la courbe bleue, se rapproche aussi de 0 V. Le cycle ainsi décrit va se répéter.

Attendez, quelque chose ne va pas ! Vous avez dit que le condensateur se charge avec le temps, or la courbe bleue bondit de 0 V au niveau maximum quand le signal d'entrée passe à 5 V. Comment cela se fait-il ?

Bien raisonné ! C'est vrai qu'il y a de quoi s'y perdre. Regardez bien la construction du circuit. Le signal bleu en question est la tension aux bornes de la résistance ; elle varie au rythme du courant qui passe à travers le condensateur.

La courbe bleue ne représente pas la tension aux bornes du condensateur. Pour s'en assurer, il nous faut modifier légèrement le circuit. Permutons simplement la résistance et le condensateur. On obtient ainsi le circuit suivant.



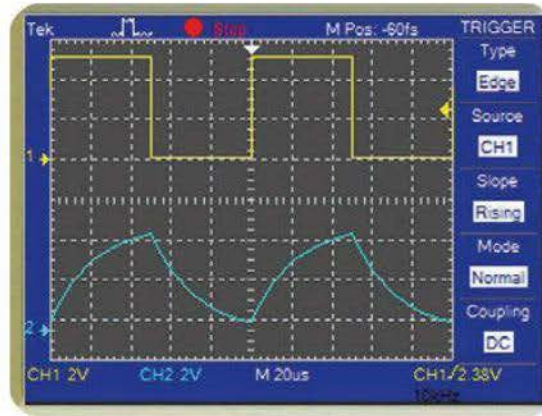
◀ **Figure 4-31**
Charge et décharge
d'un condensateur à l'aide
d'un générateur de signaux
rectangulaires

On voit que la résistance R fait office de résistance de charge et que la tension est prise aux bornes du condensateur C . L'oscillogramme



suisant montre bien plus clairement l'opération de charge et de décharge du condensateur.

Figure 4-32 ►
Tension de charge du condensateur
(avec Multisim)

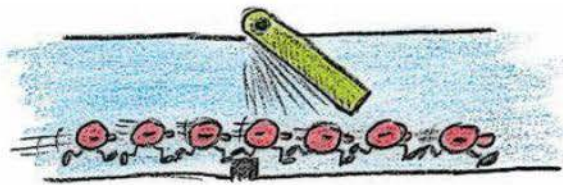


Si le signal rectangulaire passe de 0 V à sa valeur maximale 5 V, le condensateur se charge à travers la résistance R . Cela prend bien sûr du temps : la courbe bleue ne se rapproche en effet que lentement de la valeur voulue de 5 V. Lorsque l'amplitude du signal rectangulaire retombe à 0 V, le condensateur commence à se décharger à travers la résistance en partant de la valeur finale de la tension de charge. La tension va chuter pour essayer d'atteindre 0 V. Au moment où le signal rectangulaire d'entrée repasse à la valeur 5 V, le condensateur va recommencer à se charger. Le cycle ainsi décrit va se répéter.

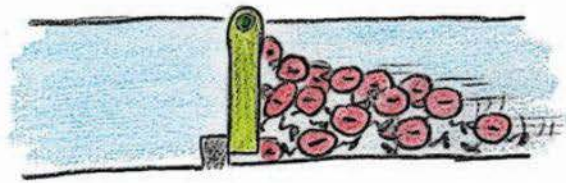
La diode

Une diode est un composant conçu à base de semi-conducteurs (silicium ou germanium). Elle a la propriété de ne laisser passer le courant que dans un seul sens (sens passant). Dans le sens inverse, le courant qui circule à travers une diode est pratiquement nul. Ce comportement électrique fait penser à une soupape de chambre à air : l'air de la pompe entre, mais aucun air ne ressort.

Figure 4-33 ►
Électrons traversant
une diode dans le sens passant



On voit que les électrons n'ont aucun mal à traverser la diode. Le clapet interne s'ouvre et les électrons circulent sans problème. Les suivants n'auront pas cette chance...



◀ **Figure 4-34**

Électrons tentant de traverser la diode dans le sens non passant

Le clapet ne s'ouvre pas dans le sens souhaité, et on se bouscule au *checkpoint* (ou point de contrôle), car rien ne bouge.

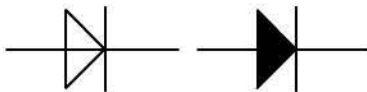
La forme et la couleur des diodes sont des plus variées. Voici deux exemples.



Le sens dans lequel la diode est passante a une énorme importance, et la présence d'un marquage sur le corps du composant est indispensable. Il ne s'agit pas cette fois-ci d'un code couleur, mais d'un trait plus ou moins épais avec une inscription dessus. Les deux polarités de la diode portent également des noms différents :

- l'anode ;
- la cathode.

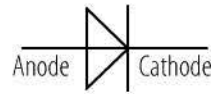
Une diode au silicium est polarisée dans le sens passant quand la différence de tension entre l'anode et la cathode est supérieure à +0,7 V.



◀ **Figure 4-35**

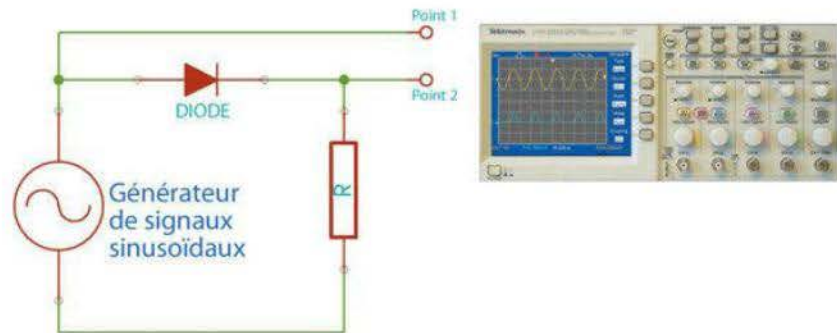
Symboles de la diode, version en contour à gauche et version pleine à droite

Mais où se situe l'anode par rapport à la cathode ? Voilà le moyen mnémotechnique que j'ai trouvé pour m'en souvenir : la cathode commence en allemand par la lettre K (*Kathode*). Un trait vertical sur le schéma de la diode indique la position de la cathode. Physiquement, cette dernière se repère par l'anneau inscrit directement sur le corps de la diode.



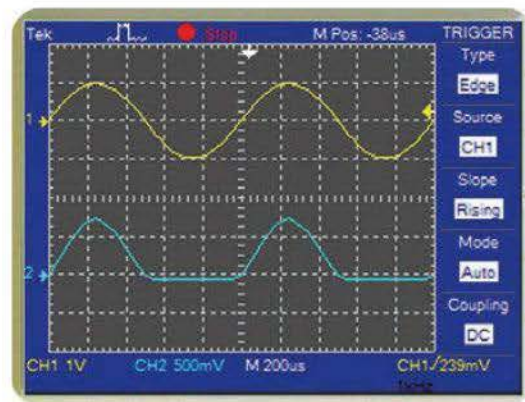
Facile à se rappeler, non ? Voyons maintenant un peu comment fonctionne la diode dans un circuit. J'utilise à l'entrée de cette dernière non pas un signal rectangulaire mais un signal sinusoïdal, présentant aussi bien des valeurs de tension positives que négatives. Le circuit doit vous être familier maintenant.

Figure 4-36 ►
Circuit pour commander une diode
au moyen d'un générateur
de signaux sinusoïdaux



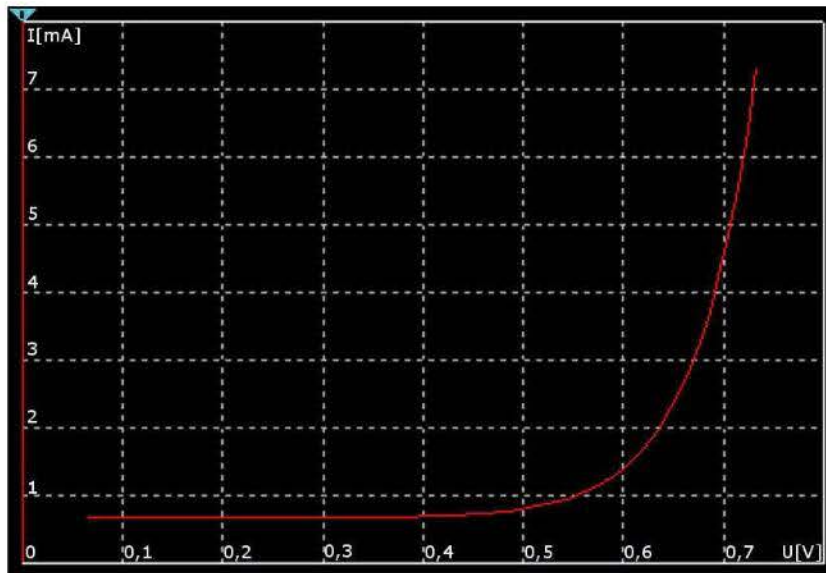
L'entrée de la diode – donc l'anode – est reliée à la sortie du générateur de signaux sinusoïdaux. Ce point de liaison est représenté par la courbe jaune dans l'oscillogramme. La sortie – donc la cathode – est figurée par la courbe bleue.

Figure 4-37 ►
Entrée et sortie d'une diode
(avec Multisim)



Le signal d'entrée en jaune forme une belle courbe sinusoïdale. La diode au silicium ne laissant cependant passer que des signaux positifs $> +0,7$ V bloque les signaux négatifs. La courbe de sortie (en bleu) ne représente que l'alternance positive de la sinusoïde. Lorsque la tension à l'entrée est négative, la tension de sortie est nulle (diode bloquée).

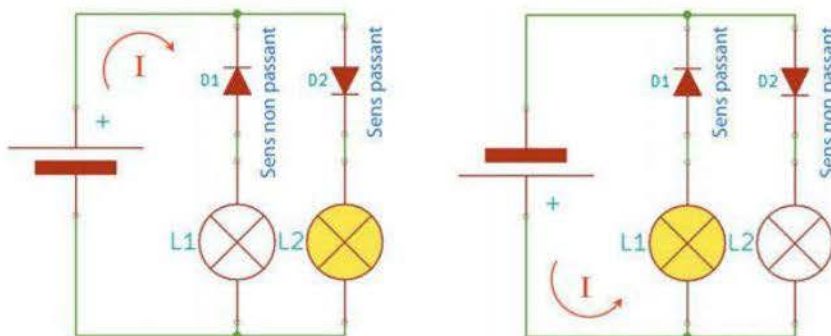
Avant d'en finir avec la diode, jetons un coup d'œil sur la caractéristique tension-courant. Cette courbe montre à partir de quelle tension d'entrée le courant se met à traverser la diode, et la diode à être conductrice. On ne détecte la présence d'un courant de conduction qu'à partir de +0,5 V environ, et qui augmente très rapidement à partir de +0,7 V.



◀ **Figure 4-38**

Courbe caractéristique tension-courant d'une diode au silicium avec Multisim

Les deux circuits très simples ci-après montrent le mode de fonctionnement décrit à l'instant pour une vanne électronique. Ils se composent de deux diodes et de deux lampes alimentées par une pile.



◀ **Figure 4-39**

Sens passant et non passant de diodes dans deux circuits à lampes

Circuit de gauche

Le pôle positif de la pile est relié à l'anode de la diode D2, qui est polarisée dans le sens passant et laisse passer le courant. La lampe L2 s'allume. La diode D1 est bloquée, car sa cathode est reliée au pôle positif de la pile. La lampe L1 reste éteinte.

Circuit de droite

La polarité de la pile est permutée et le pôle positif se trouve en bas ; les rapports de polarité sont inversés. Le pôle positif de la pile est appliqué à l'anode de la diode D1 et la lampe L1 s'allume. La diode D2 est bloquée, car le pôle positif se trouve connecté à sa cathode. La lampe L2 reste éteinte.

Vous vous demandez peut-être maintenant à quoi servent de tels composants. Les domaines d'application sont multiples. En voici quelques-uns :

- redressement de courant alternatif ;
- stabilisation de tension ;
- diode de roue libre (protection contre la surtension aux bornes d'une inductance lors de l'arrêt, par exemple d'un moteur).

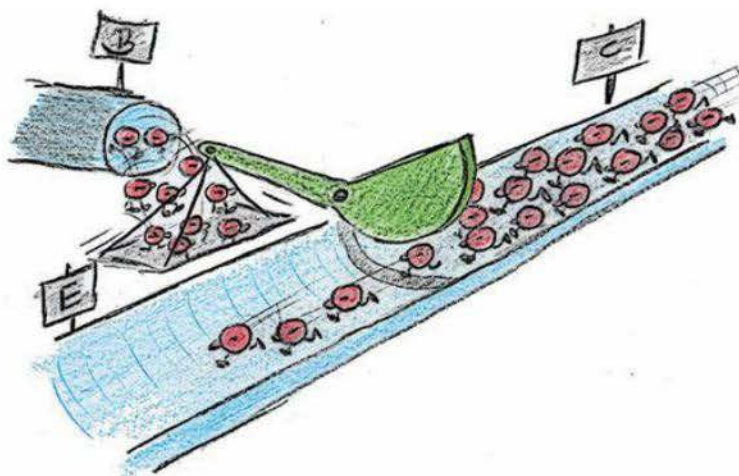
Il existe de nombreux types de diodes, par exemple des Zener ou à effet tunnel. Les énumérer toutes ici et expliquer leurs différences feraient exploser le nombre de pages de ce livre ! Je vous renvoie par conséquent à la littérature électronique spécialisée ou à Internet.

Le transistor

Venons-en maintenant à un composant électronique très intéressant qui a contribué tout d'abord au développement des circuits intégrés : le transistor. Il est le premier composant électronique à faire partie de la catégorie des composants actifs. Il s'agit d'un dispositif conçu à partir de semi-conducteurs, qui peut être utilisé aussi bien comme commutateur électronique que comme amplificateur.

Dans la plupart des cas, le transistor possède trois pattes. Il en existe de nombreuses variantes, avec des formes, tailles et couleurs différentes.

Figure 4-41 ►
Électrons traversant un transistor



La figure 4-43 présente l'intérieur d'un transistor NPN (voir page 98) relié au pôle positif de la source de tension par la connexion B. Une lettre est attribuée à chaque patte pour pouvoir différencier les connexions d'un transistor.

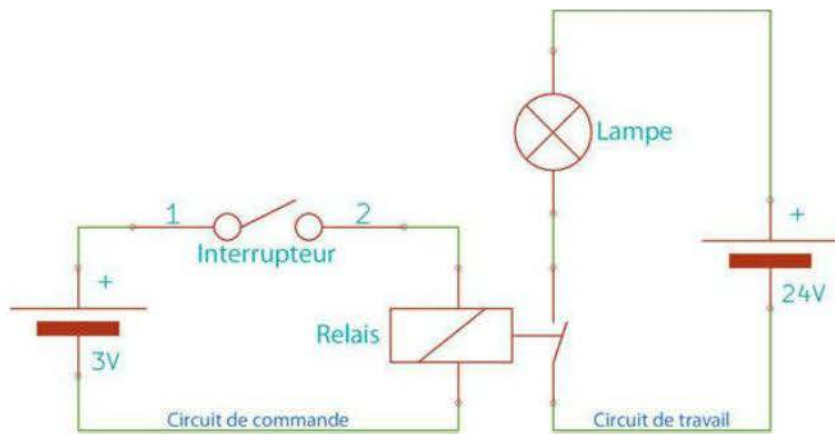
- B pour base ;
- C pour collecteur ;
- E pour émetteur.

La figure 4-41 présente comment le flot d'électrons circule entre le collecteur et l'émetteur. C'est le circuit de travail, qui permet, par exemple, de commander d'autres dispositifs (lampes, relais et même moteurs). On voit aussi le courant passer par la base ; c'est le courant de commande, qui régule par son intensité le courant de travail. Sachez qu'un courant de commande très faible peut donner naissance à un courant de travail relativement élevé. Ce comportement est appelé amplification.



Je ne vois pas très bien la différence entre circuit de commande et circuit de travail. Pourquoi existe-t-il tout à coup deux circuits de courant ? Je pensais qu'il n'y en avait toujours qu'un seul.

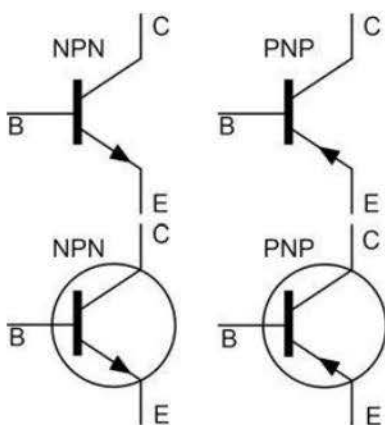
Pour mieux comprendre, voici le principe à l'aide d'un simple circuit conventionnel avec des composants électriques.



◀ **Figure 4-42**
Circuits de travail et de commande
avec des composants électriques

À gauche se trouve le circuit qui commande le relais via un commutateur. Pour l'instant, sachez simplement que le relais est un composant électromécanique qui ferme un contact quand une tension est appliquée ; une alimentation de 3 V est suffisante pour le commander. À droite se trouve le circuit de travail permettant d'allumer une lampe de 24 V. Les contacts de travail du relais ferment ce circuit de courant quand le commutateur est fermé et la lampe s'allume. On part du principe que le courant du circuit de commande est plus faible que celui du circuit de travail.

Vous voyez qu'ici on travaille avec deux circuits de courant distincts. Appliquons maintenant ce mode de fonctionnement au transistor. Je vous montre avant le schéma de branchement de ce dernier. De même qu'on trouve deux types de transistors, il existe aussi deux symboles différents.



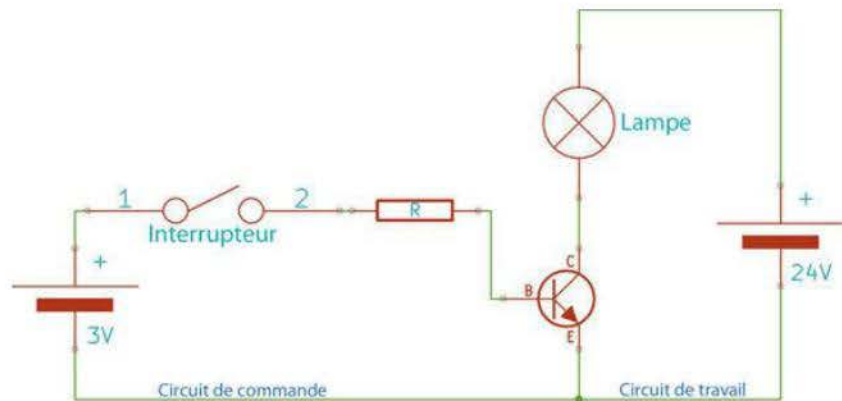
◀ **Figure 4-43**
Symboles du transistor, sans cercle
en haut et avec cercle en bas

Un transistor présente trois couches de silicium superposées, les deux couches extérieures étant toujours identiques. Les différences entre NPN et PNP résident dans l'agencement de ces couches.

- Sur le transistor NPN, les couches N se trouvent à l'extérieur et composent le collecteur et l'émetteur, et la couche P du milieu constitue la base. Le fonctionnement normal d'un tel transistor est obtenu lorsque le potentiel base-émetteur est de $+0,7\text{ V}$ au minimum. À cette condition, il y aura du courant au niveau du collecteur et de l'émetteur.
- En revanche, le fonctionnement normal d'un transistor PNP est obtenu lorsque le potentiel base-émetteur est négatif et a une valeur maximale de $-0,7\text{ V}$.

Je peux maintenant vous montrer le principe des circuits de commande et de travail.

Figure 4-44 ►
Circuits de commande et de travail
avec des composants électroniques



Le relais a été remplacé par un transistor NPN commandé positivement (lorsque l'interrupteur est fermé) via une résistance série R . Cette dernière est absolument nécessaire, car un courant de base trop fort provoque une surchauffe et risque de détruire le transistor. Même si les circuits de commande et de travail ont une masse commune, on continue néanmoins de parler de deux circuits de courant distincts.



Observons maintenant de plus près un transistor. J'ai opté pour le type BC557C (voir figure 4-45). Il s'agit d'un transistor PNP, qui correspond dans la configuration NPN au transistor BC547C. Comme le montre la vue d'ensemble des transistors, leurs boîtiers ont des formes très diverses. Ici, il s'agit d'un transistor vraiment universel, approprié pour des petits circuits amplificateurs ou des applications de commutation. Il est logé dans un boîtier de type TO-92 en plastique. Ces deux types de transistors ont le même brochage.



◀ **Figure 4-45**

Brochage des transistors BC547C
et BC557C (vue de dessous)



Pour aller plus loin

Vous trouverez sur Internet toutes les informations utiles sur les transistors et tous les autres composants cités dans ce livre.

Le circuit intégré

Cette miniaturisation s'est bien sûr faite en plusieurs étapes. Tout a commencé avec la découverte du transistor, qui a permis aux développeurs de loger plusieurs circuits dans des espaces beaucoup plus réduits. Dans les premiers temps, des circuits plus ou moins complexes utilisaient les tubes. Ceux-ci étaient bien plus gros qu'un transistor et réclamaient donc plus de puissance. Plus tard, on a placé des quantités énormes de transistors sur des circuits imprimés pour pouvoir concentrer en un lieu une fonction électronique complexe, ce qui a abouti à la longue à des accumulations de cartes. Par la suite, quelqu'un a eu l'idée de mettre plusieurs composants discrets, tels que transistors, résistances et condensateurs, sur une puce au silicium de quelques millimètres carrés. Le circuit intégré (ou IC pour *Integrated Circuit*) était né.

- Années 1960 : environ deux douzaines de transistors par puce (3 mm^2).
- Années 1970 : environ deux milliers de transistors par puce (8 mm^2).
- Années 1980 : quelques centaines de milliers de transistors par puce (20 mm^2).
- Aujourd'hui : plusieurs milliards de transistors par puce.

Le microcontrôleur ATTiny13 avec ses 8 pattes de raccordement est un exemple parlant. Il s'agit d'un véritable mini-ordinateur avec tout ce que cela comporte (unité arithmétique, mémoires, ports d'entrée et de sortie, etc.). Il y a quelques décennies, un ordinateur de cette complexité aurait nécessité un nombre incroyable d'eurocartes à base de composants discrets (dimensions : $160 \times 100 \text{ mm}$).

Figure 4-46 ►
Le microcontrôleur ATtiny13 dans
un boîtier DIP de la société Atmel



Attention !

Dès l'introduction, je vous ai averti du risque encouru par les circuits intégrés en cas de charge statique. Par exemple, marcher sur un tapis en polyester charge votre corps en énergie électrostatique qui peut alors se décharger à tout moment sous la forme d'un éclair. La décharge peut atteindre facilement 30 000 volts et avoir raison des transistors les plus solides. Une mise à la terre, en touchant par exemple un tuyau de chauffage non peint ou un contact de sûreté, est donc conseillée.

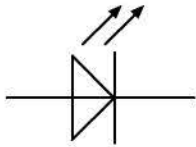
La LED

Une diode électroluminescente – appelée aussi LED (*Light Emitting Diode*) – est un composant semi-conducteur qui émet de la lumière à une certaine longueur d'onde en fonction du matériau semi-conducteur employé. Le sens du courant est important car la LED n'émet de la lumière que dans le sens passant. La LED n'est pas détériorée en cas de polarité inversée ; elle reste simplement éteinte. Il faut impérativement veiller à ce qu'une LED soit toujours accompagnée d'une résistance série correctement calculée. Faute de quoi, sa luminosité sera étonnamment intense la première fois et ensuite plus rien. Les diodes électroluminescentes ont des formes et des couleurs variées.

Figure 4-47 ►
Plusieurs types de diodes
électroluminescentes



Tout comme une diode normale, la diode électroluminescente présente deux contacts appelés anode et cathode. Son symbole est similaire, avec seulement deux flèches en plus indiquant la lumière émise.



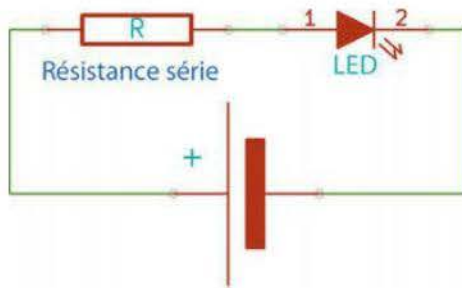
◀ **Figure 4-48**

Symbole de la diode électroluminescente

Dans la vue suivante, on peut voir qu'une patte de raccordement est plus courte que l'autre, pour mieux différencier l'anode de la cathode. Le fil le plus long est l'anode.



Pour qu'une LED puisse s'allumer, l'anode doit être raccordée au pôle positif et la cathode au pôle négatif. La vue suivante montre un circuit simple pour commander une LED.



◀ **Figure 4-49**

Polarisation d'une LED à travers une résistance série

Autres composants

Les éléments de circuits mentionnés jusqu'à présent font tous partie de la famille des composants électroniques. Je vais maintenant vous présenter quelques composants électriques.

L'interrupteur

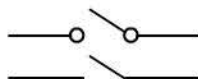
Le courant ne circule que si on a un circuit fermé et que les électrons peuvent circuler librement. Pour pouvoir influencer sur ce circuit de l'extérieur, vous devez, par exemple, installer un interrupteur. Il s'agit d'un dispositif qui ouvre ou ferme un contact. Vous trouverez une multitude de versions d'interrupteurs munis de deux ou de plusieurs contacts.

Figure 4-50 ►
Divers types d'interrupteurs



L'interrupteur le plus simple possède deux contacts. Il peut être représenté par différents symboles.

Figure 4-51 ►
Symboles de l'interrupteur



L'état de l'interrupteur peut être qualifié de stable. S'il a été actionné, il reste dans cette position jusqu'à ce qu'il le soit de nouveau.

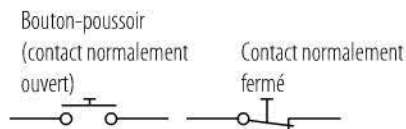
Le bouton-poussoir

Le bouton-poussoir est un proche parent de l'interrupteur ; il influe aussi sur la circulation du courant. S'il n'est pas actionné, le circuit est en principe interrompu. Je dis « en principe », car il existe aussi des boutons-poussoir qui sont fermés tant qu'ils ne sont pas actionnés et qui interrompent le circuit quand ils le sont. On les appelle alors contacts normalement fermés.

Figure 4-52 ►
Divers types de boutons-poussoir



Le symbole du bouton-poussoir ressemble à celui de l'interrupteur. Il existe cependant quelques petites différences qui ont leur importance et qui ne doivent pas être ignorées.



◀ **Figure 4-53**

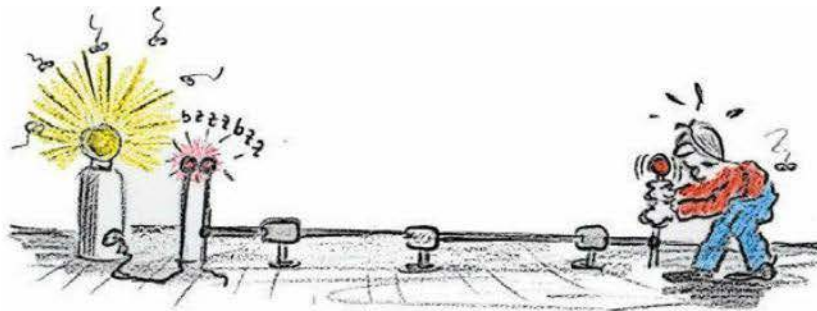
Symboles du bouton-poussoir et du contact normalement fermé

L'état d'un bouton-poussoir est qualifié de non stable. Si vous appuyez dessus, le contact se ferme et le courant peut circuler. Si vous le relâchez, le contact revient dans sa position initiale et la circulation du courant est de nouveau interrompue.

Dans nos montages, nous utiliserons très souvent des boutons-poussoir, plus rarement des interrupteurs. Mon modèle préféré est le bouton-poussoir miniature, qui peut être soudé directement sur la carte.

Le relais

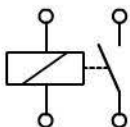
Je vous ai déjà parlé du relais dans l'introduction sur le transistor. Je souhaite revenir ici plus précisément sur ce composant. Un relais n'est en fait rien d'autre qu'un interrupteur ou un inverseur que vous pouvez actionner à distance. La figure 4-54 montre un travailleur fermant un contact à distance du temps où il n'y avait pas encore de relais.



◀ **Figure 4-54**

Interrupteur à distance d'un autre temps

Un relais peut être représenté par différents symboles.



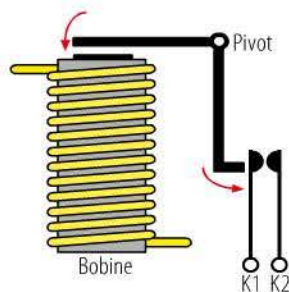
◀ **Figure 4-55**

Symboles du relais (avec un contact de travail)



J'ai ouvert ici un relais pour pouvoir observer de plus près sa structure interne. Sur le côté gauche se trouve la bobine, dont le cœur est constitué d'un noyau en fer permettant de mieux canaliser les lignes de champ magnétique. Si un courant circule dans la bobine, la palette mobile est attirée et pousse les contacts de travail vers la droite, qui peuvent tout aussi bien être ouverts que fermés. La vue schématisée suivante montre comment la palette est attirée vers le bas, fermant ainsi un contact.

Figure 4-56 ▶
Schéma d'un relais



Si la palette mobile est attirée vers le bas, elle ferme les deux contacts K1 et K2. Utilisé d'une certaine manière, un relais peut également servir d'amplificateur, si cela s'avère souhaitable. Un courant faible circulant dans la bobine peut, à condition que les contacts du relais aient les bonnes dimensions, commander un courant beaucoup plus important.



Attention !

Ne jamais brancher un relais directement sur une sortie de la carte Arduino. Il passerait à coup sûr beaucoup plus de courant que ce que la sortie peut délivrer, endommageant le microcontrôleur. Vous verrez plus tard comment un relais peut être commandé.

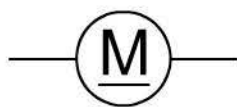
Le moteur

Je pense que vous savez ce qu'est un moteur. Bien entendu, nous ne parlerons pas d'un moteur à combustion, dont le carburant serait par exemple du gazole, mais d'un moteur électrique. Il s'agit ici d'un assemblage qui transforme l'énergie électrique en énergie motrice.



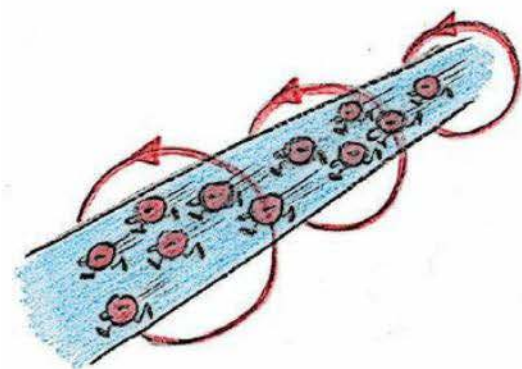
◀ **Figure 4-57**
Plusieurs types de moteurs
électriques

Il existe des moteurs de toutes tailles et pour toutes les gammes de tension. Ils sont tout aussi bien fabriqués pour du courant continu que pour du courant alternatif.



◀ **Figure 4-58**
Symbole du moteur à courant
continu

Concentrons-nous sur le courant continu. Un moteur à courant continu comprend un élément fixe, représenté par l'aimant, et un élément mobile : la bobine. Celle-ci est montée de manière à pivoter sur un axe. Si un courant est envoyé par le biais d'un conducteur, un champ magnétique se crée autour de celui-ci. Il sera d'autant plus élevé que la longueur de fil est concentrée sur une certaine zone. Cela explique les nombreux tours de fil autour de la bobine.

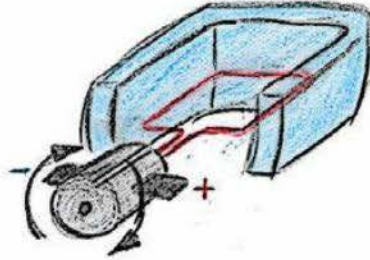


◀ **Figure 4-59**
Conducteur dans lequel circule
un courant.

La figure 4-59 représente un conducteur parcouru par des électrons circulant dans un sens. Les cercles rouges indiquent les lignes de champ magnétique générées par le courant. Si nous approchons maintenant une aiguille de boussole du conducteur fixe, elle réagit en se tournant dans le sens des lignes de champ magnétique. Lignes de champ magnétique du fil et aiguille de boussole subissent une interaction de forces. Si nous

utilisons à la place un aimant fixe renfermant un conducteur mobile, la force induite provoque un mouvement du fil.

Figure 4-60 ►
Schéma très simplifié d'un moteur
à courant alternatif



La figure 4-60 montre une seule spire de conducteur en rouge, pouvant tourner librement dans un aimant permanent en bleu. Si on fait maintenant passer un courant dans le fil, les champs magnétiques de ce dernier réagissent avec ceux de l'aimant. Le fil tourne le long de l'axe. Le rotor gris en deux parties, sur lequel est fixé le fil, inverse la polarité de ce dernier après une rotation à 180° et fait circuler le courant en sens inverse. Le champ magnétique créé dans le fil revient alors à la polarité précédente (du fait de la rotation de 180° et de la commutation du courant) et provoque une nouvelle rotation du fil à 180° avec une nouvelle inversion de polarité. Ce changement permanent du champ magnétique assure un mouvement rotatif du fil avec le moteur.

Pour amplifier les forces entre les deux champs magnétiques, un moteur comporte évidemment de nombreuses spires de fil conducteur qui forment une bobine, et développe une certaine force pendant la rotation. Le pilotage d'un moteur exigeant un peu plus de courant que ce que peut fournir une seule sortie du microcontrôleur, un transistor est nécessaire pour assurer l'amplification. Nous verrons bientôt comment ça marche.

Couper l'alimentation du moteur pose cependant un problème non négligeable : la bobine induit elle-même, après l'arrêt du courant d'alimentation, une surtension (auto-induction) susceptible d'endommager le microcontrôleur ou le transistor du fait de son niveau et du sens de circulation inversé. Nous verrons comment résoudre ce problème lorsque nous traiterons de la diode de roue libre.

Le moteur pas-à-pas

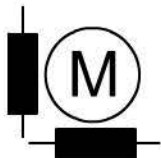
Un moteur normal tourne jusqu'à ce que le courant qui l'alimente soit coupé, puis il effectue encore quelques tours emporté par son élan. Il finit par s'arrêter dans une position à coup sûr imprévisible. Ce

comportement n'est bien sûr pas souhaité quand il s'agit d'atteindre précisément certaines positions bien déterminées, les unes à la suite des autres. Pour cela, il faut employer un type de moteur spécial : le moteur pas-à-pas. Peut-être avez-vous déjà vu des automates industriels qui servent à monter des éléments de carrosserie pour les souder ensemble à l'endroit précis où ils doivent l'être ? La position doit alors être rigoureusement exacte car tout doit ensuite se raccorder. Ces automates sont actionnés par des moteurs pas-à-pas, tout comme les scanners à plat et les tables traçantes.



◀ **Figure 4-61**
Plusieurs types de moteurs pas-à-pas

Les moteurs pas-à-pas ont plus de deux fils de raccordement. La plupart du temps, un tel moteur est représenté avec deux bobines, mais son symbole peut varier.

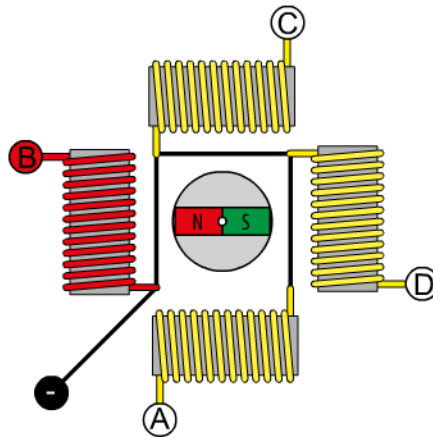


◀ **Figure 4-62**
Symbole du moteur pas-à-pas

Pour que ce moteur puisse passer par certaines positions, il doit présenter une construction interne qui l'amène à s'arrêter à certains endroits. Cela ne se faisant pas avec des moyens mécaniques (par exemple, une roue dentée qui se bloquerait à un endroit pendant la rotation), il doit y avoir une solution électrique. Quand on fixe un aimant sur un axe et positionne des bobines tout autour, l'aimant se tourne vers la bobine parcourue par le courant et ne bouge plus ensuite. Un moteur pas-à-pas fonctionne selon ce principe. J'ai choisi

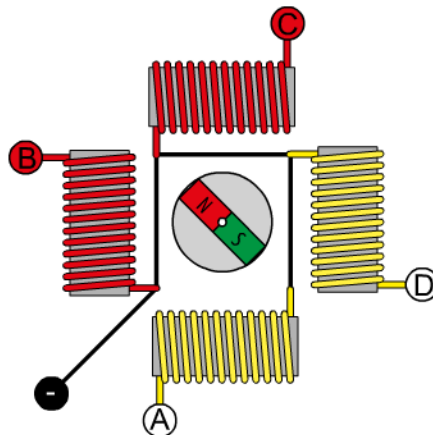
pour simplifier un moteur avec 4 bobines et une commande simple, dont le positionnement est par conséquent approximatif.

Figure 4-63 ►
Représentation schématique
d'un moteur pas-à-pas avec
4 bobines ou positions



Au centre se trouve l'aimant pivotant, entouré des 4 bobines. Elles sont toutes reliées à la masse par l'une de leurs deux extrémités. Pour expliquer le fonctionnement du moteur pas-à-pas, un courant est appliqué à la bobine B. L'aimant se tourne dans la direction de cette dernière pour ne plus bouger. Si une seule bobine à la fois est alimentée en courant, quatre positions différentes (de 90° chacune) peuvent être obtenues tout au plus. Mais si deux bobines voisines sont alimentées en même temps, la palette mobile s'arrête entre les deux. La précision est donc plus élevée.

Figure 4-64 ►
Excitation simultanée
de plusieurs bobines



Il est maintenant possible de travailler avec des pas de 45° au lieu de 90° . Pour que la position adoptée reste stable, la ou les bobines doivent être alimentées en courant jusqu'à ce qu'une nouvelle direc-

tion soit ordonnée. Pour que le moteur tourne dans le sens des aiguilles d'une montre, les broches des bobines doivent être branchées dans le bon ordre.

Commençons, par exemple, par la bobine B : B/BC/C/CD/D/DA/A/AB/B/, etc.

Eh là, pas si vite ! J'ai regardé de plus près les moteurs pas-à-pas et j'ai remarqué qu'ils disposent tous de 4 broches, sauf un qui en a 5. Pourquoi ?

Vous avez pris une loupe pour voir ça ? Mais, c'est vrai, Arduus ! Vous avez tout à fait raison. Il existe deux types de moteurs pas-à-pas :

- unipolaires (5 ou 6 broches) ;
- bipolaires (4 broches).

Les premiers sont plus faciles à commander, car le courant circule toujours dans le même sens dans les bobines. C'est pour cette raison que j'ai pris comme exemple un moteur unipolaire pas-à-pas.



Le servomoteur

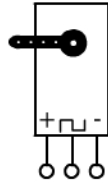
Les modèles réduits d'avions ou de bateaux disposent de petits servomoteurs pour commander des fonctions les plus diverses, comme la vitesse ou le cap. Il s'agit de petits moteurs à courant continu qui sont équipés de 3 broches et dont le positionnement est commandé par une modulation de largeur d'impulsions (MLI). Vous en saurez plus dans le chapitre 10 sur la programmation de la carte Arduino.



◀ **Figure 4-65**
Deux types de servomoteurs

Voici comment peut être représenté un servomoteur (voir figure 4-66).

Figure 4-66 ►
Symbole du servomoteur



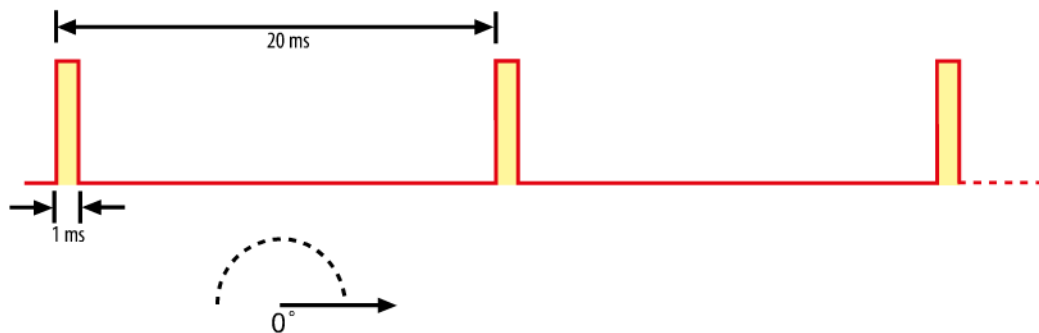
Qu'est-ce au juste que la MLI ? Un servomoteur non modifié a en principe un rayon d'action allant de 0° à 180° et ne peut pas pivoter à 360° comme un moteur. Le pivotement du servomoteur est commandé par un signal rectangulaire avec des spécifications particulières.

Durée d'une période

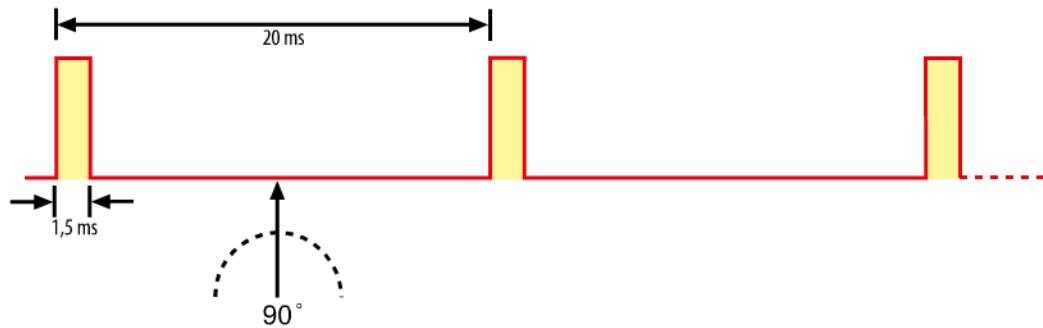
La durée T d'une période est constante et vaut 20 ms.

Durée d'impulsion

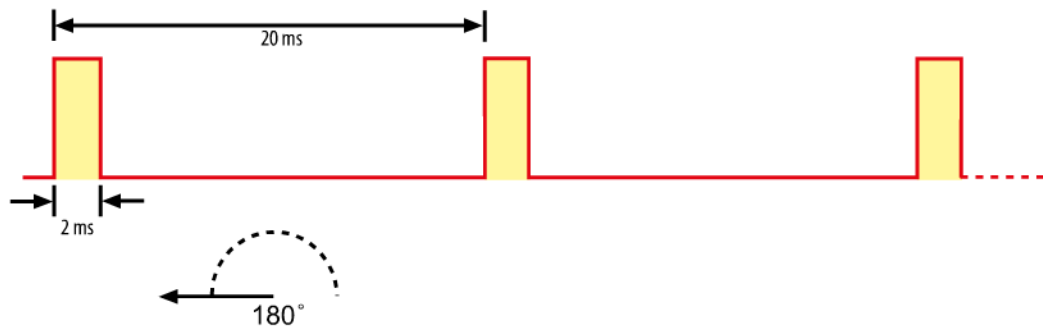
La durée de l'impulsion doit être comprise entre 1 ms (butée de droite) et 2 ms (butée de gauche). Les figures ci-après présentent trois positions de servomoteur avec les signaux de commande périodiques correspondants.



Avec une durée d'impulsion de 1 ms, le servomoteur est amené sur la butée de droite, qui correspond à un angle de 0° .



Avec une durée d'impulsion de 1,5 ms, le servomoteur peut occuper la position centrale, qui correspond à un angle de 90° .

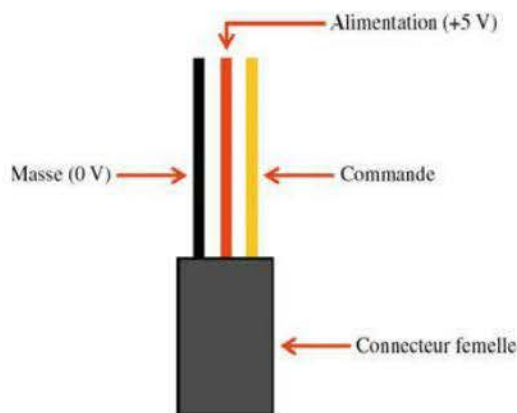


Avec une durée d'impulsion de 2 ms, le servomoteur est amené sur la butée de gauche, ce qui correspond à un angle de 180° .

Vous avez maintenant une vague idée de ce qu'est la MLI. La largeur (ou durée d'impulsion) permet de commander un composant électronique comme le servomoteur. La même méthode peut être utilisée pour gérer la luminosité, par exemple dans le cas de diodes électroluminescentes.

Les valeurs peuvent certes varier en fonction des servomoteurs, mais le principe reste le même. Plus besoin de se triturer la tête pour savoir comment et avec quelles valeurs commander le servomoteur, car d'autres développeurs s'en sont déjà chargés et nous pouvons utiliser leur savoir. Il existe des codes sources prêts à l'emploi que nous pouvons incorporer dans notre montage. Vous verrez bientôt comment tout cela fonctionne. Le positionnement n'étant ordonné que par un seul signal de commande, le servomoteur n'a pas beaucoup de broches.

Figure 4-67 ►
Brochage d'un servomoteur



Le buzzer piézoélectrique

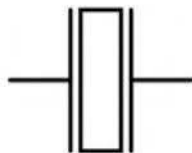
Je souhaite conclure ce chapitre sur l'électronique en vous présentant le buzzer piézoélectrique.

Figure 4-68 ►
Buzzer piézoélectrique



Sa forme est assez bizarre et on a du mal à croire que ce composant puisse faire du bruit. Il renferme un cristal qui se met à vibrer quand une tension alternative est appliquée. Cet effet appelé piézoélectrique se produit quand des forces (pression ou déformation) sont exercées sur certains matériaux ; une tension électrique est alors mesurable. Le buzzer piézoélectrique agit de façon inverse : quand une tension alternative est appliquée, une déformation régulière, perçue comme une vibration, se produit et met en mouvement les molécules d'air, ce qui est perçu comme un son. Pour qu'il soit plus fort, le mieux est de coller le buzzer piézoélectrique sur un support vibrant, de sorte que les vibrations émises soient transmises et amplifiées.

Figure 4-69 ►
Symbole du buzzer piézoélectrique



Après la lecture de ce chapitre, vous disposez maintenant d'une bonne vue d'ensemble des différents composants électroniques. Certes, il y aurait encore beaucoup de choses à dire, mais j'ai choisi d'en rester là dans ce livre. Si vous souhaitez en savoir davantage, je vous invite à consulter des ouvrages spécialisés.

Circuits électroniques simples

Au sommaire de ce chapitre :

- les circuits résistifs (montages en série et en parallèle) ;
- le diviseur de tension ;
- les circuits capacitifs (montages en série et en parallèle) ;
- les circuits avec transistors.

Puisque vous connaissez maintenant les bases de l'électronique grâce au chapitre précédent, passons à l'étape logique suivante qui consiste à associer plusieurs composants pour en faire un circuit. Pour que les débuts soient moins difficiles, je vais vous montrer quelques circuits de base ne nécessitant, pour la plupart, que très peu de composants. Dans la seconde partie du livre constituée de 19 montages Arduino, la complexité augmentera au fil des projets, mais vous pourrez toujours les construire en suivant les principes exposés ici. Ce chapitre ne se veut pas un précis des circuits électroniques de base, mais se focalise sur la compréhension des projets Arduino. Vous trouverez le cas échéant des explications plus détaillées dans les montages eux-mêmes. Ne vous inquiétez pas, vous saurez tout ce qu'il faut savoir le moment venu.

Les circuits résistifs

Une résistance dans un circuit électrique agit comme un limiteur de courant. La traversée de la résistance est rendue plus ou moins difficile aux électrons qui assaillent ce composant. Le principe est facile à comprendre.

Imaginez un grand nombre de personnes venues voir un concert, obligées de passer par une petite entrée de 2 mètres de large pour pénétrer dans la salle de spectacle. Les corps se touchent si bien que le flux des spectateurs se ralentit. Bien entendu, les gens ont tendance à transpirer et beaucoup de chaleur est émise. On avance moins vite que si l'entrée faisait par exemple 10 mètres de large.

Montages en série et en parallèle

Que se passe-t-il quand plusieurs résistances sont reliées selon une certaine configuration ? Cela devrait influencer d'une manière ou d'une autre sur la résistance totale. Prenons deux exemples : le montage en série d'une part et le montage en parallèle d'autre part.

Le montage en série

Quand deux résistances ou plus sont montées l'une derrière l'autre, on parle d'un montage en série. Il est dans la nature des choses que la résistance totale augmente à mesure que le nombre de résistances individuelles placées l'une derrière l'autre augmente. La résistance totale est en l'occurrence égale à la somme des résistances individuelles. Supposons maintenant que les 3 résistances suivantes soient montées l'une derrière l'autre :



La résistance totale se calcule comme suit :

$$R_{\text{totale}} = R_1 + R_2 + R_3 = 1K + 2K + 1,5K = 4,5K$$

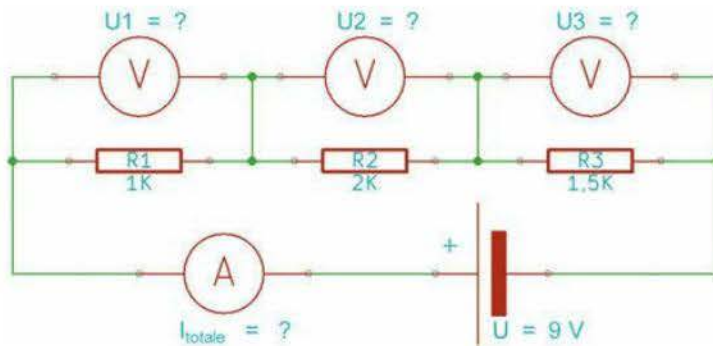
J'aurais aimé avoir votre avis sur le courant qui passe par les résistances. Comment se comportent-elles avec lui selon vous ? Partons du principe que le courant passe de gauche à droite dans les résistances.



Le courant devrait alors faiblir derrière chaque résistance. Plus je mesure loin à droite derrière chaque résistance, plus le courant est faible.

Non Arduus, ce n'est pas tout à fait vrai. La première partie de ta réponse est juste car chaque résistance diminue la circulation du courant. Pour autant, un seul courant – partout identique – est mesu-

nable sur l'ensemble du circuit électrique. Voyons cela dans un circuit.



◀ **Figure 5-1**
Montage en série de 3 résistances
dans un circuit électrique

Quelles valeurs sont connues dans ce circuit et lesquelles sont inconnues et doivent être calculées ?

Connues : U , R_1 , R_2 et R_3

Inconnues : I_{totale} , U_1 , U_2 et U_3

Sachant que le courant I est partout le même, vous pouvez utiliser la formule suivante :

$$I_{totale} = \frac{U}{R_{totale}} = \frac{U}{R_1 + R_2 + R_3}$$

Si vous appliquez les valeurs, vous obtenez le résultat suivant :

$$I_{totale} = \frac{9 \text{ V}}{1\text{K} + 2\text{K} + 1,5\text{K}} = 2 \text{ mA}$$

Maintenant que vous avez calculé un courant $I = 2 \text{ mA}$ passant par tous les composants, vous pouvez calculer la chute de tension à chaque résistance. La formule communément utilisée est la suivante :

$$U = R \cdot I$$

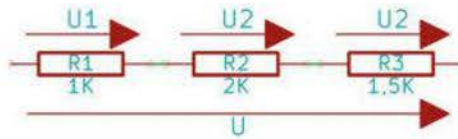
L'équation est donc la suivante :

$$U_1 = 1\text{K} \cdot 2 \text{ mA} = 2 \text{ V}$$

$$U_2 = 2\text{K} \cdot 2 \text{ mA} = 4 \text{ V}$$

$$U_3 = 1,5\text{K} \cdot 2 \text{ mA} = 3 \text{ V}$$

Si vous additionnez toutes les tensions partielles (U_1 , U_2 , U_3) vous obtenez à nouveau la tension totale U .



La chute de tension aux bornes d'un composant est indiquée par une flèche et va dans le sens du courant du plus (+) vers le moins (-).

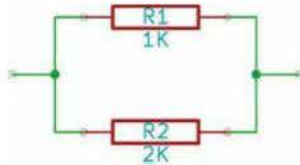


Pour aller plus loin

C'est la résistance présentant la valeur la plus élevée qui connaît également la plus forte chute de tension.

Le montage en parallèle

Un montage en parallèle consiste à placer deux ou plusieurs composants côte à côte. Le courant qui arrive dans un montage de ce type se scinde alors en plusieurs branches.



Il se comporte comme un cours d'eau qui se divise à un endroit pour se reformer quelques kilomètres plus loin. La résistance totale se calcule comme suit :

$$\frac{1}{R_{totale}} = \frac{1}{R_1} + \frac{1}{R_2} = \frac{1}{1K} + \frac{1}{2K}$$

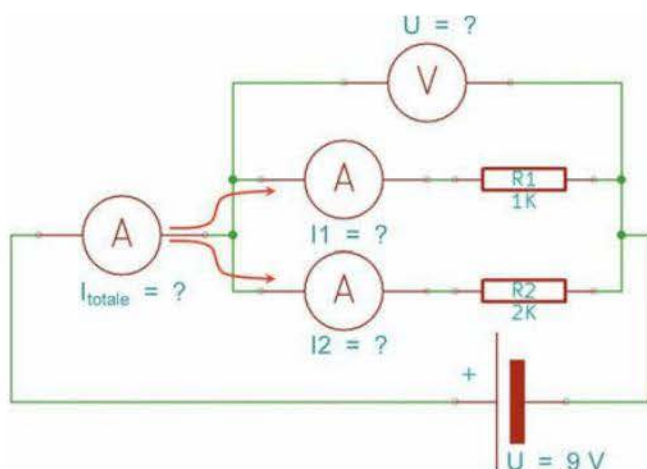
Le résultat pour la résistance totale R_{totale} est le suivant :

$$R_{totale} = 666,67 \, \Omega$$

Si plus de deux résistances sont montées en parallèle, vous devez rallonger la formule à concurrence du nombre de termes correspondants de la somme.

$$\frac{1}{R_{totale}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \dots + \frac{1}{R_n}$$

Un circuit avec deux résistances montées en parallèle se présente comme suit :



◀ **Figure 5-2**
Montage en parallèle
de deux résistances

Dans ce circuit, un faible courant parcourt évidemment aussi l'appareil qui mesure la tension via les résistances, mais laissons cela de côté pour l'instant. Quelles valeurs sont connues dans ce circuit et lesquelles sont inconnues et doivent être calculées ?

Connues : U , R_1 et R_2

Inconnues : I_{totale} , I_1 et I_2

Une résistance totale de $666,67 \Omega$ a déjà été calculée. Celle-ci vous permet de déterminer très simplement l'intensité du courant total I_{totale} avant sa ramification. Je vous redonne la formule :

$$I = \frac{U}{R}$$

Ce qui donne :

$$I_{totale} = \frac{9 \text{ V}}{666,67 \Omega} = 13,5 \text{ mA}$$

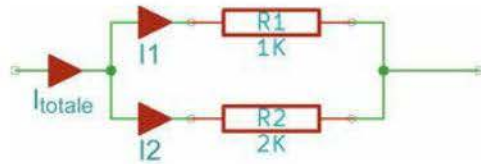
Comment calculer cependant les courants partiels I_1 et I_2 ? Rien de plus simple puisque vous connaissez la résistance de chaque branche et la tension présente aux bornes de chaque résistance.

Quand des composants sont montés en parallèle, la chute de tension est la même pour chacun d'eux. Nous avons ici affaire à une pile de 9 V. Effectuons le calcul :

$$I_1 = \frac{9 \text{ V}}{1 \text{ K}} = 9 \text{ mA}$$

$$I_2 = \frac{9 \text{ V}}{2 \text{ K}} = 4,5 \text{ mA}$$

Si vous additionnez les deux courants partiels I_1 et I_2 , qu'obtenez-vous à votre avis ? Exact, le courant total.



Ce qui se ramifie en amont (donc à gauche sur la figure) finit par se rejoindre et forme la somme des parties.



Pour aller plus loin

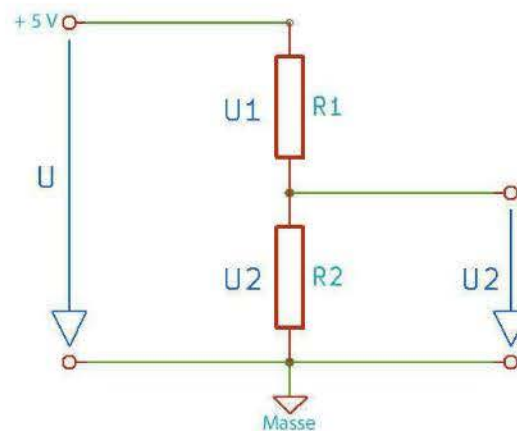
Si plusieurs résistances sont montées en parallèle, la résistance totale est inférieure à la plus petite résistance individuelle.

Voici une astuce concernant les tailles de résistance. Si vous montez deux résistances de même valeur en parallèle, la résistance totale est exactement égale à la moitié de chaque résistance. Faites le calcul pour vérifier.

Le diviseur de tension

Dans beaucoup de cas, on ne souhaite pas utiliser la pleine tension de service de +5 V pour alimenter divers composants. Maintenant que vous savez que des résistances sont utilisées par exemple pour diminuer des courants, je voudrais vous montrer un circuit qui ressemble au montage de résistances en série. Le circuit en question est appelé diviseur de tension non chargé.

Figure 5-3 ►
Diviseur de tension non chargé



Sur le côté gauche, la tension d'alimentation $U = +5\text{ V}$ a été appliquée aux deux résistances R_1 et R_2 . Sur le côté droit se trouve une mesure de tension U_2 , aux bornes de la résistance R_2 . Une tension est pour ainsi dire prélevée entre les deux résistances. Une partie de la tension d'alimentation chute à travers R_1 et l'autre à travers R_2 . La formule suivante peut être utilisée pour calculer la tension U_2 :

$$U_2 = \frac{R_2}{R_1 + R_2} \cdot U$$

Eh là pas si vite ! Expliquez-moi seulement comment vous en êtes venu à cette formule ? J'ai du mal à comprendre.

D'accord Ardu, pas de problème. Je peux rendre cette équation compréhensible au moyen d'une équation de proportionnalité. Je confronte les résistances correspondantes à la tension présente à leurs bornes. La tension U est appliquée aux résistances R_1 et R_2 et la tension U_2 uniquement à la résistance R_2 . L'équation de proportionnalité suivante peut donc être posée :

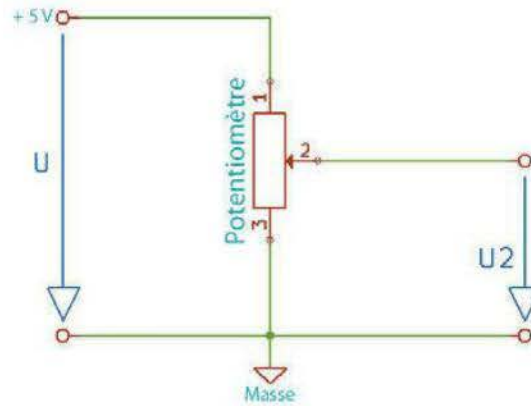
$$\frac{U}{R_1 + R_2} = \frac{U_2}{R_2}$$

En tirant U_2 de cette équation, on obtient la formule ci-dessus. Nous souhaitons cependant configurer le circuit d'une manière aussi flexible que possible et non pas changer les résistances pour chaque valeur de tension U_2 souhaitée. C'est pour cette raison que nous utilisons un composant qui nous permet d'adapter rapidement la valeur de la résistance à nos attentes.

Vous connaissez déjà ce composant : c'est le potentiomètre. Il dispose de trois bornes et d'un bouton central rotatif permettant d'ajuster la valeur de la résistance dans les limites données. La borne centrale est reliée en interne au curseur. La résistance peut ainsi être ajustée par le réglage du potentiomètre. Regardez la figure 5-4. Elle montre un schéma de câblage d'un potentiomètre qui ressemble beaucoup au circuit du diviseur de tension.

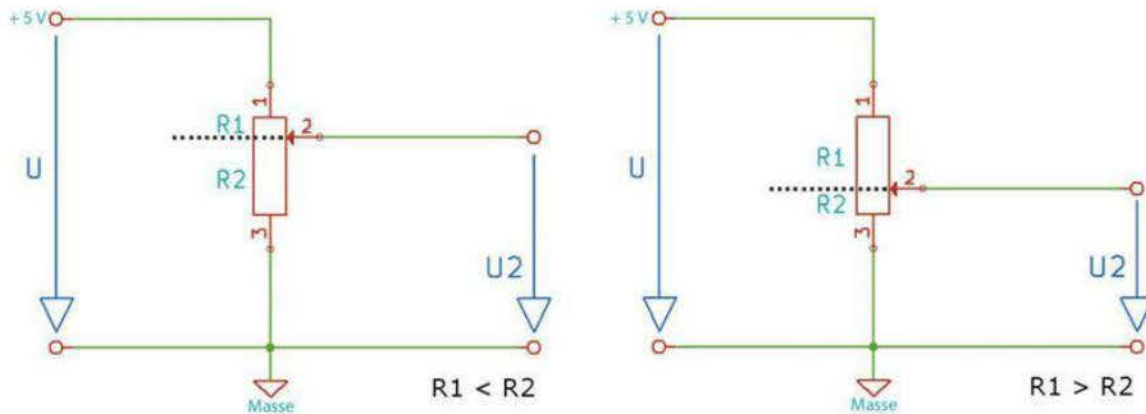


Figure 5-4 ►
Diviseur de tension variable
par potentiomètre



Le curseur du potentiomètre est la broche 2 dans le circuit. Plus le curseur va vers le haut, plus la valeur de résistance entre la broche 1 et la broche 2 diminue, tandis qu'elle augmente en proportion entre la broche 2 et la broche 3. Le potentiomètre peut être vu comme deux résistances qui se modifient, avec un curseur tenant lieu de diviseur qui fractionne les deux résistances. La figure 5-5 montre le comportement du potentiomètre et les résistances R_1 et R_2 qui en résultent.

Figure 5-5 ▼
Diviseur de tension variable
par potentiomètre



Dans le circuit de gauche, la résistance R_1 est inférieure à R_2 . Autrement dit, la tension la plus élevée est mesurée aux bornes de R_2 , laquelle se trouve être également la tension de sortie U_2 . C'est logique car si le curseur du potentiomètre sur la broche 2 continue de monter, il finit par toucher la tension d'alimentation de +5 V qui est alors disponible à la sortie. Inversement, la tension de sortie diminue à mesure que le curseur du potentiomètre descend vers la masse. Une fois celle-ci atteinte, la tension est de 0 V à la sortie. Nous nous en servons par exemple pour alimenter les entrées analogiques du microcontrôleur avec des valeurs de tension variables qui, par

exemple, baissent par le biais d'une LDR ou NTC. Vous avez oublié la signification de ces sigles ? Vous trouverez toutes les réponses dans le chapitre précédent !

Les circuits capacitifs (avec condensateurs)

Les condensateurs servent à stocker une charge et se comportent comme une discontinuité dans un circuit de courant continu. Seul un courant de charge circule pendant le cycle de chargement, lequel diminue à mesure que le condensateur se charge. Ce dernier finit en revanche par constituer une barrière infranchissable pour les électrons.

Montages en série et en parallèle

Des condensateurs peuvent, tout comme les résistances, faire partie de diverses configurations. Ne travaillant pour le moment qu'en courant continu, nous nous concentrerons ici sur la capacité et non pas sur la résistance. Eh oui, un condensateur présente également une résistance, qui dépend de la fréquence pour le courant alternatif ! Les condensateurs réagissent pour ce qui est de leurs capacités en tout point inversement à ce que font les résistances avec leurs valeurs dans des montages en série ou en parallèle.

Le montage en série

Si vous branchez deux condensateurs ou plus en série et si voulez déterminer la capacité totale, vous pouvez utiliser la formule permettant de calculer la résistance totale dans un montage en parallèle.



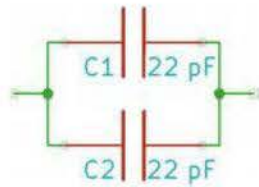
La formule pour calculer la capacité totale est la suivante :

$$\frac{1}{C_{totale}} = \frac{1}{C_1} + \frac{1}{C_2} = \frac{1}{22\text{pF}} + \frac{1}{33\text{pF}}$$

$$\text{donc } C_{totale} = 13,2 \text{ pF}$$

Le montage en parallèle

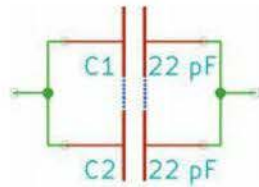
Si deux condensateurs ou plus sont branchés en parallèle, la formule du montage en série pour des résistances peut servir à calculer la capacité totale.



La formule pour calculer la capacité totale est la suivante :

$$C_{\text{totale}} = C_1 + C_2$$

Le montage de ces deux condensateurs en parallèle est facile à comprendre et on voit tout de suite pourquoi la capacité totale est la somme des deux capacités partielles. Les plaques des condensateurs sont simplement reliées entre elles par les points bleus. Les plaques ont été agrandies de sorte qu'une capacité équivalant à la somme des capacités des deux condensateurs a été créée.



Le résultat serait dans ce cas :

$$C_{\text{totale}} = C_1 + C_2 = 22 \text{ pF} + 22 \text{ pF} = 44 \text{ pF}$$

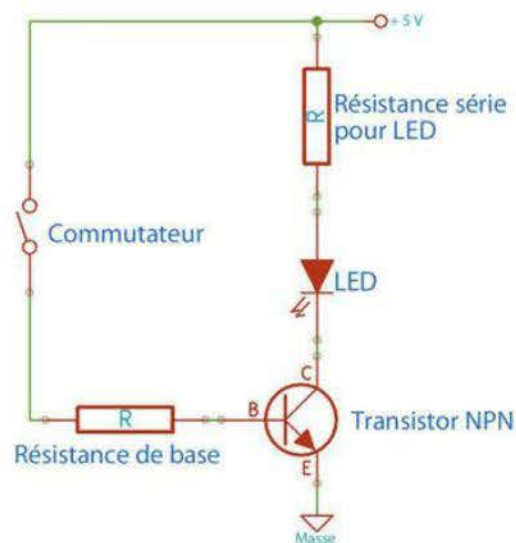
Le comportement des condensateurs étant ce qu'il est en courant continu, je ne vois pas très bien en quoi ces composants sont utiles.

Vous vous souvenez peut-être de certains passages du chapitre précédent sur les bases de l'électronique, où j'explique que les condensateurs sont entre autres utilisés pour lisser et stabiliser la tension. Parlons brièvement de la stabilisation. Quand un microprocesseur doit alimenter les multiples consommateurs raccordés à ses nombreuses sorties (tels que diodes électroluminescentes ou moteurs, qui peuvent tous être activés en même temps), la tension d'alimentation peut connaître de courts effondrements. Aussi des condensateurs de filtrage sont utilisés pour que l'alimentation du microcontrôleur

n'en souffre pas directement et ne puisse pas engendrer une sous-alimentation qui le conduise à interrompre sa tâche ou à se réinitialiser. Ils sont connectés aux deux prises V_{CC} (*Voltage of Common Collector*, soit tension d'alimentation positive) et masse du contrôleur et sont placés le plus près possible de ces broches. Un condensateur électrolytique de 100 μF par exemple emmagasine le courant et maintient la tension un moment en cas d'interruption. Il s'agit pratiquement d'une ASI (alimentation sans interruption) du domaine de la milliseconde.

Les circuits avec transistors

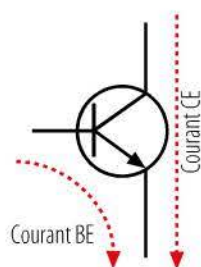
Les transistors peuvent être aussi bien des éléments de commutation que des amplificateurs. Le circuit à transistor le plus simple comporte une résistance de base et une charge avec une résistance en série dans le circuit de collecteur, et sert de commutateur électronique sans contact. Le transistor servira principalement de commutateur, aussi je m'abstiendrai d'expliquer son utilisation comme amplificateur pour des raisons de place.



◀ **Figure 5-6**
Transistor NPN servant
de commutateur

Ce montage possède à la fois un circuit de commande (à gauche de la base) et un circuit de travail (à droite de la base). Voyons à présent ces deux circuits de plus près.

Figure 5-7 ►
Le courant de commande
et le courant de travail
circulent ensemble
dans le transistor.



Le courant de commande I_B effectue le trajet base-émetteur (BE) du transistor tandis que le courant de travail I_C effectue le trajet collecteur-émetteur (CE). Sans vouloir en dire plus sur l'utilisation du transistor comme amplificateur, la formule suivante peut être intéressante car elle permet de calculer l'amplification en courant, désignée ici par la lettre β :

$$\beta = \frac{I_C}{I_B} = \frac{300 \text{ mA}}{50 \text{ }\mu\text{A}} = 6\,000$$

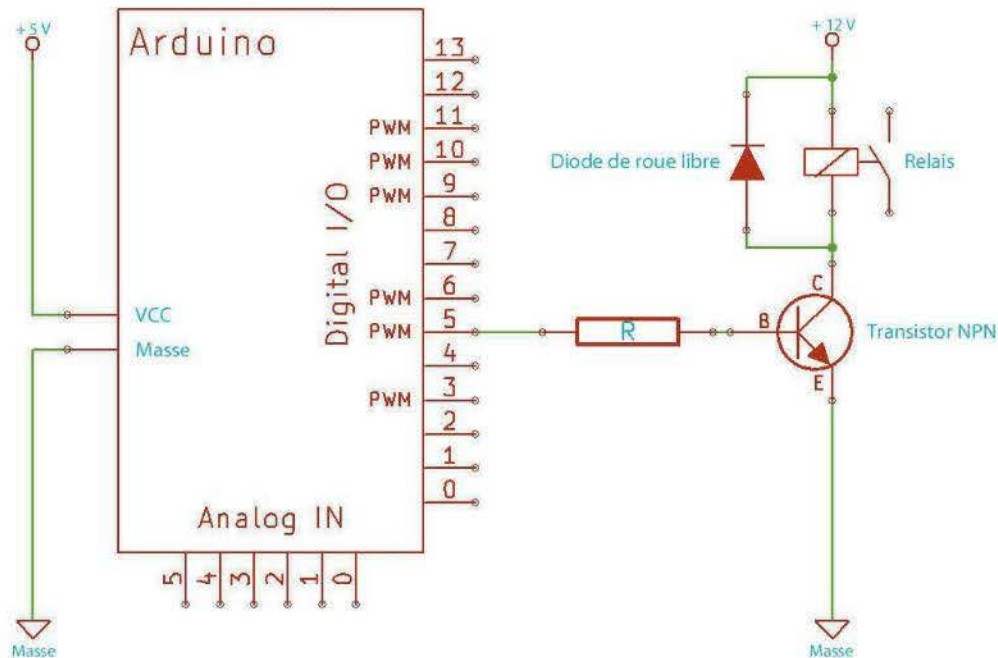
Avec les valeurs de courant collecteur et de courant de base utilisées dans cet exemple, on trouve un facteur d'amplification en courant $\beta = 6\,000$. Dans beaucoup de fiches techniques, le facteur d'amplification en courant β est également appelé hFE. L'amplification préserve en quelque sorte la broche de sortie du microcontrôleur, qui ne doit fournir qu'un faible courant pour commander une charge plus grande (par exemple un relais, un moteur ou une lampe) réclamant bien plus de courant pour que le composant concerné puisse travailler correctement. Quand vous fermez l'interrupteur, la tension d'alimentation de +5 V environ est appliquée sur la résistance de base. La tension base-émetteur atteint +0,7 V environ et le transistor conduit, si bien que le circuit collecteur-émetteur, bloqué jusque-là, voit son impédance diminuer très fortement et le courant de collecteur peut circuler dans la charge.



Quand je regarde ce circuit, je me demande pourquoi la diode électroluminescente est commandée par un transistor et non pas directement par l'interrupteur. Y a-t-il une raison ?

Que dire Ardu... Il est vrai que ce circuit ne sert qu'à montrer comment courant de commande et courant de travail coopèrent. Tout ceci est un peu surdimensionné et n'est pas impérativement nécessaire pour commander une simple diode électroluminescente. Si en revanche vous avez une utilisation qui consomme un courant très

important, que la sortie du microcontrôleur n'est pas capable d'alimenter, il vous faut un circuit du genre de celui décrit ici. Rappelez-vous les spécifications de notre microcontrôleur, il y est dit qu'une seule sortie ne peut fournir que 40 mA au maximum. Au-delà, le contrôleur s'abîme. Vous avez peut-être un relais requérant une tension de 12 V dans votre bric-à-brac. La carte Arduino ne pouvant fournir que 5 V au maximum, il y a comme un problème. Cependant, rien ne vous oblige à avoir une seule source de courant. Vous pouvez utiliser deux circuits d'alimentation séparés. Voici un exemple :



Qu'est-ce qui saute aux yeux ? Nous avons à gauche l'alimentation en +5 V de la carte Arduino et à droite celle en +12 V du relais. Toutes deux sont des sources de courant indépendantes et autonomes qui doivent pourtant avoir un potentiel de masse commun. Les deux points indiqués par Masse sur le schéma sont reliés entre eux.

▲ Figure 5-8
Microcontrôleur Arduino commandant un relais via un transistor (circuit de commande)



Attention !

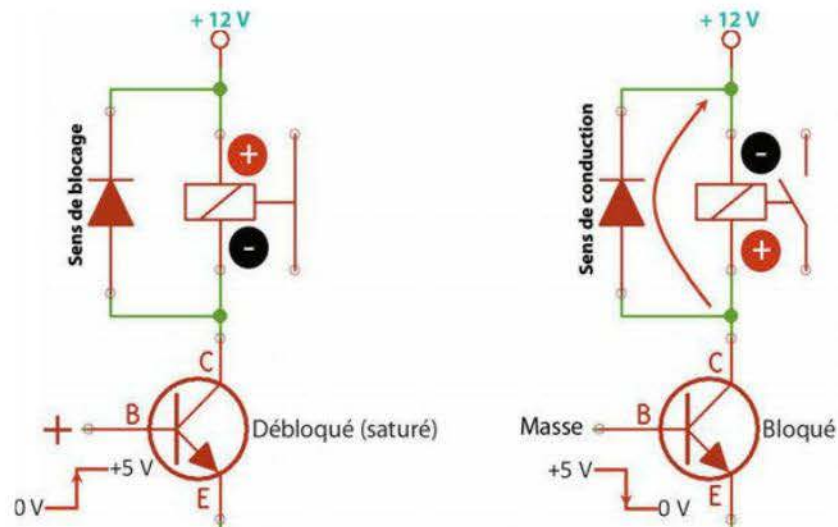
Vous ne devez en aucun cas – je dis bien en aucun cas – relier les deux points d'alimentation en +5 V et +12 V entre eux ! Clash assuré et microcontrôleur au minimum grillé.



C'est cette diode en parallèle au relais appelée diode de roue libre qui me gêne. À quoi sert-elle au juste ?

Il me faut ici approfondir un peu, Arthus. Pour pouvoir agir et pour que les contacts se ferment à la circulation du courant, un relais a besoin qu'une bobine crée un champ magnétique et déplace une palette mobile. En électronique, une bobine est également désignée par inductance. Cette dernière a une propriété particulière. Quand un courant parcourt le très long fil de la bobine, il crée un champ magnétique. Rien de nouveau jusqu'ici. Ce champ magnétique non seulement attire la palette mobile, mais induit également une tension dans la bobine elle-même. Ce processus est appelé auto-induction. La bobine se rebelle en quelque sorte, car la tension induite est orientée de manière à s'opposer au courant qui l'a provoquée. Si j'alimente une bobine en courant, la tension auto-induite essaie de contrarier la tension en question. Cette dernière ne se crée que lentement. Si par contre je coupe à nouveau le courant, la variation rapide du champ magnétique génère une tension induite qui s'oppose à la baisse de tension et s'avère plusieurs fois plus élevée que la tension initiale. C'est là tout le problème. La commutation avec le léger retard ne constitue pas un risque pour le circuit et ses composants. Lors de l'arrêt en revanche, l'effet secondaire extrêmement néfaste de la pointe de tension excessive ($> 100\text{ V}$) doit être impérativement évité pour que le circuit ne grille pas. Faute de quoi, les chances de survie du transistor sont réellement minces. Aussi une diode est-elle connectée en parallèle au relais pour écrêter la pointe de tension et dériver le courant vers la source.

Figure 5-9 ►
Diode de roue libre préservant
le transistor des surtensions



Quand le transistor du schéma de gauche est débloqué, l'excitation du relais tarde un peu et les potentiels indiqués se mettent en place au niveau de la diode : le plus à la cathode et le moins à l'anode. Autrement dit, la diode est bloquée et le circuit se comporte comme si elle n'était pas là. Si par contre la base du transistor est reliée à la masse, celui-ci se bloque et les potentiels indiqués apparaissent du fait de la variation du champ magnétique de la bobine : le plus à l'anode et le moins à la cathode. La diode travaille dans le sens de la conduction et dérive le courant vers l'alimentation. Le transistor est préservé.

Que serait l'univers du prototypage, c'est-à-dire l'assemblage rapide de composants logiciels et matériels, sans le logiciel open source Fritzing ? Il serait bien démuni ! Vous arrive-t-il aussi souvent qu'à moi de construire un circuit électronique avec votre carte Arduino, à l'aide d'une plaque d'essais et de divers composants raccordés par des cavaliers, en vue de produire un prototype pour votre projet ? La programmation par le biais d'un sketch sur mesure donne vie au montage. C'est ce qui s'appelle le *Physical Computing*. Une fois le travail terminé, le tout fonctionne à votre convenance et vous êtes content de vous. Pour les projets assez ou très ambitieux, vous aimeriez sans doute pouvoir conserver le circuit pour votre usage personnel ou pour la postérité. En d'autres termes, vous aimeriez archiver votre œuvre. Lorsque le projet n'est pas voué à une courte vie sur la plaque d'essais, mais qu'il doit servir durablement au quotidien, il est conseillé et indiqué de produire une carte sous la forme d'un circuit imprimé professionnel.

Dans ce domaine, Fritzing vous apporte la solution. Ce logiciel open source, qui a été développé à l'université des sciences appliquées de Potsdam, en Allemagne, permet de concevoir des circuits en un tour de main, de les documenter, de créer un schéma électrique et même de faire fabriquer un circuit imprimé à partir de ces informations. Si vous voulez avoir un aperçu du potentiel de Fritzing pour le prototypage, consultez le site <http://fritzing.org/projects/>. Vous y trouverez de nombreuses sources d'inspiration. Plusieurs circuits présentés dans ce livre ont été construits à l'aide du logiciel Fritzing.

L'interface du logiciel

L'interface de Fritzing étant très intuitive, vous vous familiariserez rapidement avec ce logiciel. Mais avant de commencer, vous devez le télécharger depuis la page <http://fritzing.org/download/>. Il existe des versions pour Windows, Mac OS X et Linux. La procédure d'installation est décrite en détail. La figure suivante présente l'interface du logiciel dans lequel un projet Fritzing a été ouvert.

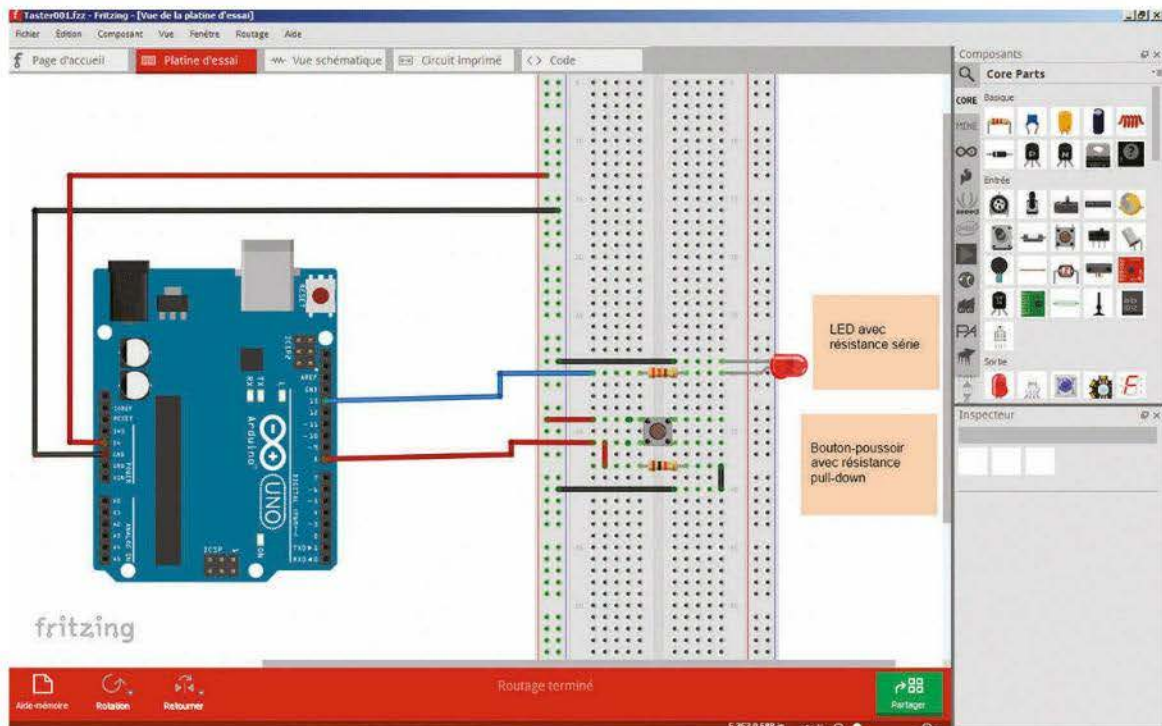


Figure 6-1 ▲
Interface de Fritzing
avec le projet
du bouton-poussoir

Ce projet contient déjà quelques composants que vous rencontrerez probablement souvent à l'avenir. Il s'agit, notamment, de la pièce maîtresse de tous les montages de ce livre : la carte à microcontrôleur Arduino. Elle est raccordée par des connexions en couleur à la plaque d'essais blanche sur laquelle se trouvent, par ailleurs, des composants comme des résistances, un bouton-poussoir et une diode rouge. Pour garantir le bon fonctionnement du circuit, ses composants sont évidemment aussi raccordés les uns aux autres par des lignes colorées. Pour commenter le fonctionnement du circuit, vous pouvez ajouter des remarques sous la forme de post-its jaunes, comme ceux que vous collez sans doute sur votre écran.

La construction ou l'extension du circuit s'effectue par cliquer-glisser depuis une bibliothèque de composants classés dans différentes caté-

gories, accessibles à l'aide d'onglets sur le côté droit de la fenêtre. Il suffit de faire glisser le composant voulu et de le placer sur la breadboard. Vous trouverez des rubriques réunissant les composants de base, comme les résistances, les condensateurs, les transistors, les diodes, les LED, etc. Il y a aussi des rubriques spécialisées qui contiennent tous les modèles de cartes Arduino existants, ainsi que les principaux shields. D'autres onglets permettent également d'accéder aux nombreux shields de la marque Sparkfun Electronics, ainsi qu'à d'autres composants. Le choix de composants électroniques prêts à l'emploi est impressionnant et il devrait répondre à tous vos besoins ou presque.

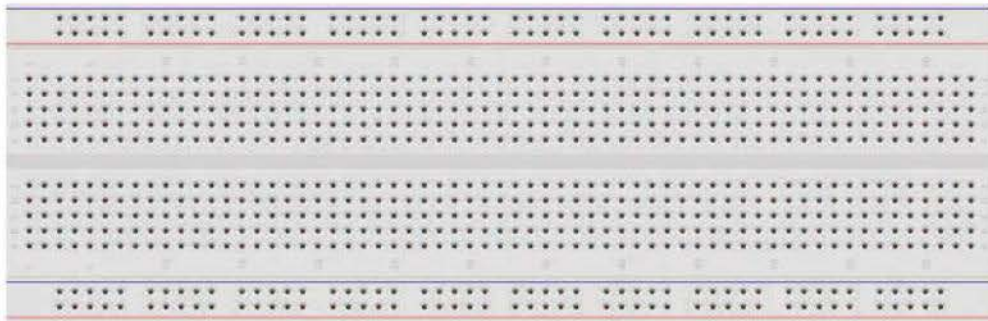
Si une pièce particulière est absente de la bibliothèque, vous pouvez l'y ajouter en suivant les instructions. Puis vous pourrez même la mettre à disposition de la communauté toute entière afin que d'autres développeurs puissent aussi en profiter. D'ailleurs, c'est un peu le mot d'ordre qui devrait réunir tous les bricoleurs amateurs ou passionnés : développez et construisez, puis mettez votre projet à disposition pour favoriser le transfert de connaissances. Vos montages n'auraient pas grand intérêt si vous vous contentiez de bricoler dans votre coin. Il serait dommage que le reste du monde ne puisse pas en profiter.

Revenons à l'interface de Fritzing. Outre la page d'accueil qui vous permet d'accéder rapidement à vos derniers sketches, la fenêtre se divise en trois parties :

- plaque d'essais ;
- vue schématique ;
- circuit imprimé.

Platine d'essai

C'est depuis cet onglet que vous pouvez construire votre circuit sur ordinateur. Une plaque d'essais vide apparaît toujours au démarrage de Fritzing. Vous pouvez vous servir de cette base de départ pour venir y enficher des composants. Il existe différentes tailles de breadboards parmi lesquelles vous pouvez choisir celle qui correspond à vos besoins. La taille *Full+* qui est proposée par défaut est un bon début (voir page suivante).



Comme chaque composant sélectionné est doté d'un certain nombre de propriétés, telles que la taille, l'orientation, la couleur, etc., qui sont affichées sur le côté droit, vous pouvez modifier les dimensions de la plaque après l'avoir sélectionnée.

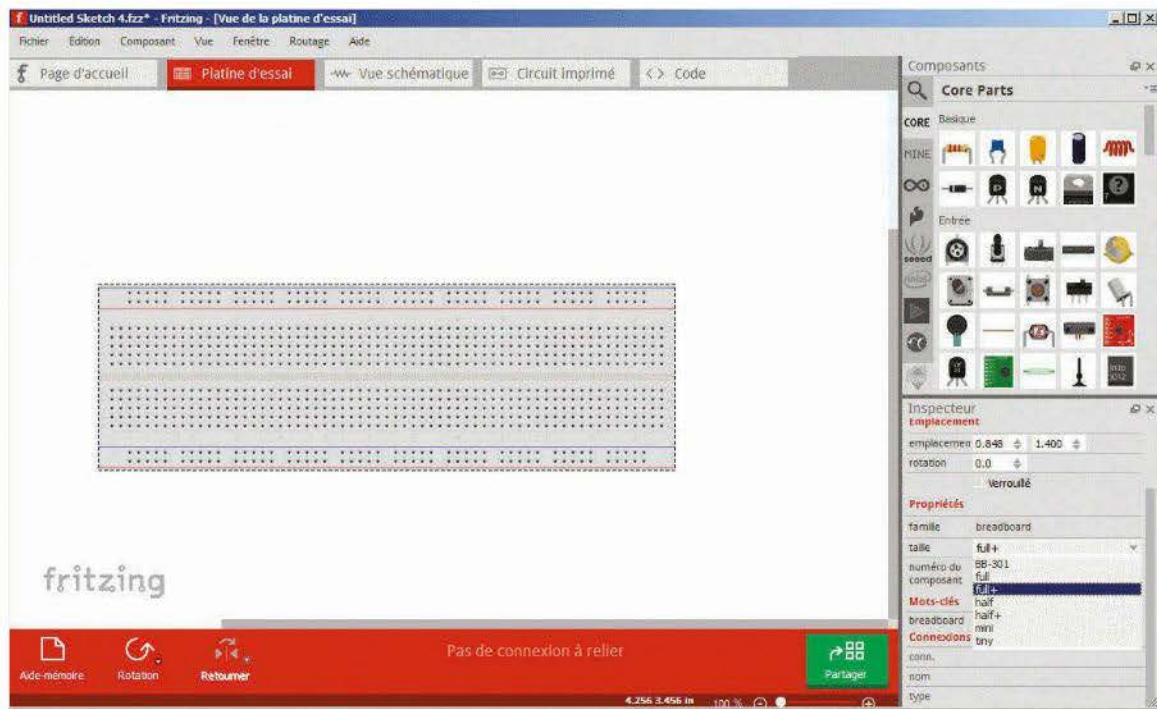
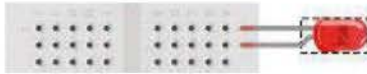


Figure 6-2 ▲
Choix de la taille de la plaque
d'essais

Je ne peux malheureusement pas décrire toutes les fonctionnalités en détail. Mais, à titre d'exemple, j'aimerais vous présenter la réalisation d'un petit projet de commande d'une LED au moyen d'un bouton-poussoir. J'ai réduit la breadboard à la taille *Tiny* et j'ai pris une LED dans la bibliothèque *Basique*. Les deux pattes de raccordement doivent être mises en contact avec les trous dans la plaque afin

d'établir une liaison conductrice. Tant que les extrémités des pattes ne chevauchent pas les trous, il n'y a pas de liaison électrique.



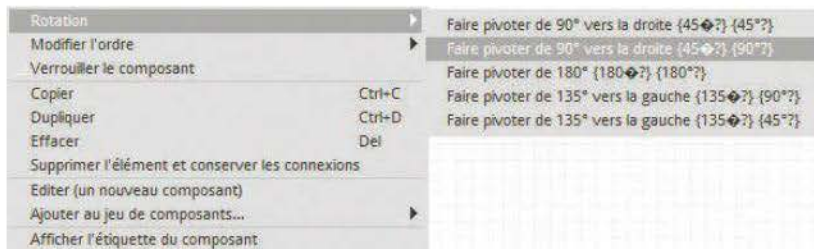
Lorsque je fais glisser la LED sur les trous, l'extrémité des pattes change de couleur afin de signaler que la LED est en place.



Lorsque je relâche le bouton de la souris, la liaison électrique est établie. Comme vous pouvez le voir, toutes les prises reliées les unes aux autres sur la plaque sont affichées en vert en fonction de l'endroit où la LED a été branchée.



Tous les autres composants fonctionnent sur le même principe que la résistance mise en place ici. Après avoir cliqué-glissé et positionné la résistance série, j'ai corrigé sa valeur de résistance initiale en réglant une valeur de 330 Ω . Ce réglage s'effectue aussi dans le panneau d'informations du composant sélectionné. Ensuite, j'ai cliqué-glissé la carte Arduino Uno à partir de la rubrique *Arduino*. Je l'ai fait pivoter à l'aide des commandes du menu contextuel.



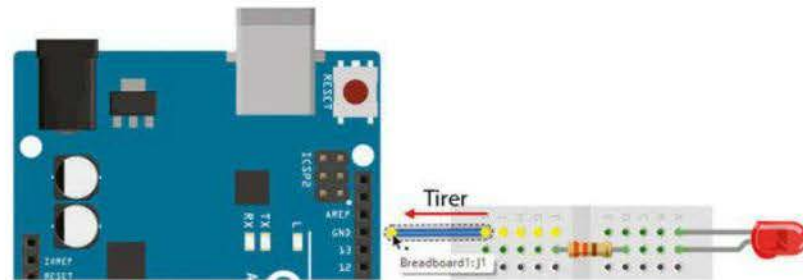
C'est encore plus simple et plus rapide à l'aide du bouton Rotation qui se trouve dans la barre d'outils rouge le long du bord inférieur de la fenêtre :



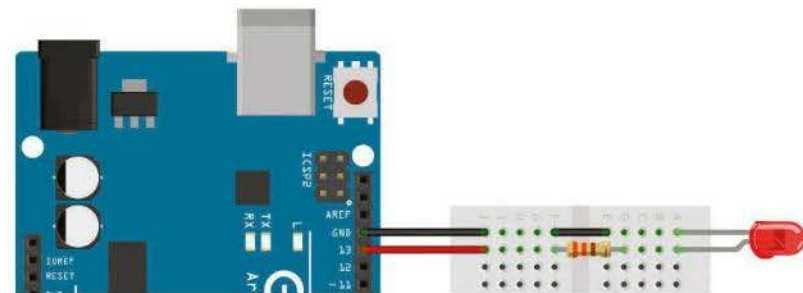
Vous pouvez cliquer directement sur le bouton pour appliquer immédiatement une rotation de 90° dans le sens horaire. Ou bien cliquez sur le petit triangle à droite du bouton.



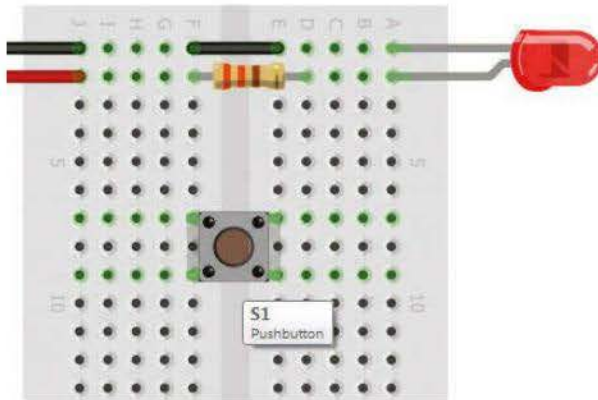
Le menu qui apparaît contient les mêmes options que proposées dans le menu contextuel. Ensuite, je relie la plaque d'essais à la carte Arduino. Je clique sur un trou dans la plaque et je maintiens le bouton de la souris enfoncé, puis je fais glisser la souris jusqu'à un port de la carte Arduino.



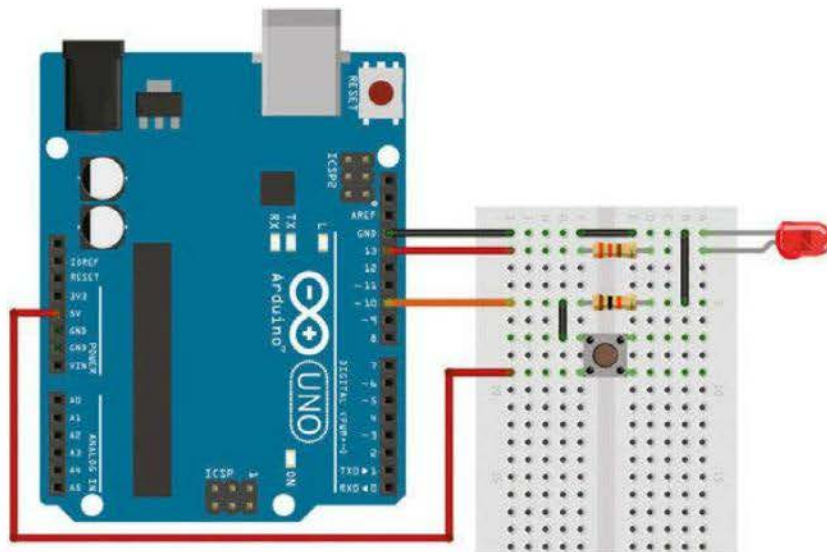
Les prises reliées électriquement s'affichent en jaune. Ainsi, vous obtenez la connexion de la LED et de la résistance série illustrée ci-après :



Comme vous avez pu le remarquer, j'ai changé la couleur des fils de raccordement : le fil de masse est noir, le fil de commande est rouge. Cela s'effectue dans le panneau d'information ou depuis le menu contextuel. Ensuite, je voudrais relier le bouton-poussoir avec une résistance pull-down à une entrée numérique. Le bouton intitulé *Pushbutton* se trouve aussi dans la rubrique *Core Parts*. Faites-le glisser sur la plaque à l'emplacement illustré.



Vous devez le faire pivoter de 90°, car son orientation initiale ne convient pas. Ensuite, nous allons raccorder la résistance pull-down au bouton-poussoir. Faites glisser une autre résistance que vous réglez sur 10 kΩ. Procédez au câblage en reprenant les couleurs et les positions illustrées ci-après.



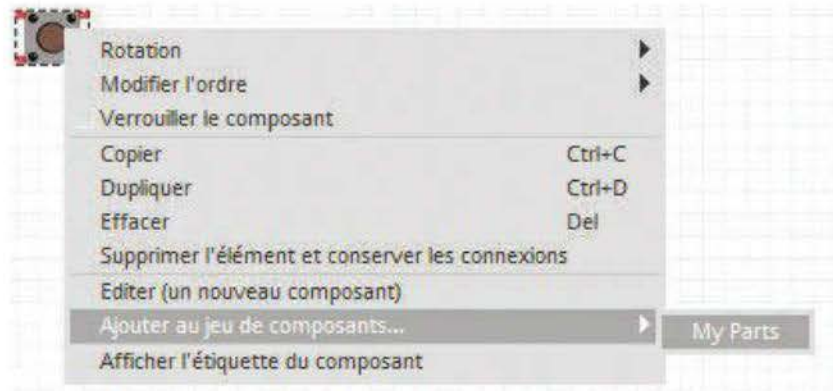
Ne vous inquiétez pas si vous ne reconnaissez pas ce montage ou si vous n'en comprenez pas le sens : je reviendrai plus loin sur le rôle d'une résistance pull-down. Je me contenterai de dire ici que lorsque le bouton n'est pas enfoncé, le potentiel de la masse est transmis à la broche d'entrée numérique 10 en passant par la résistance de 10 kΩ. Le niveau y est donc bas (LOW). Lorsque vous appuyez sur le bouton, le courant de 5 V est transmis à la broche 10 et le niveau est donc haut (HIGH). La résistance pull-down a pour rôle, d'une part,

d'éviter que ne se produise un court-circuit entre les broches GND et 5 V au moment où vous appuyez sur le bouton et, d'autre part, d'envoyer un niveau bas sur la broche 10 quand le bouton n'est pas enfoncé. En numérique, il n'y a rien de pire qu'une entrée ouverte qui représente un niveau indéfini.

Nous allons maintenant pouvoir examiner le schéma électrique du circuit que nous avons construit jusqu'ici sur la plaque d'essai. Le schéma électrique est généré automatiquement en arrière-plan au fur et à mesure que vous ajoutez des composants.

Petite astuce utile pour améliorer votre confort quand vous travaillez avec Fritzing : si vous utilisez régulièrement les mêmes composants électroniques dans vos projets, vous en aurez assez de les rechercher constamment parmi les différentes catégories. Parmi les nombreuses catégories disponibles sur le côté droit, vous en trouverez une intitulée *Mine – My Parts*. C'est là que vous pouvez ajouter les composants que vous utilisez fréquemment afin de les retrouver rapidement. Comment faire ? C'est très simple : cliquez sur un composant à l'aide du bouton droit de la souris et sélectionnez la commande *Ajouter au jeu de composants>My Parts* dans le menu contextuel.

Figure 6-3 ►
Ajout du composant
à la catégorie My Parts



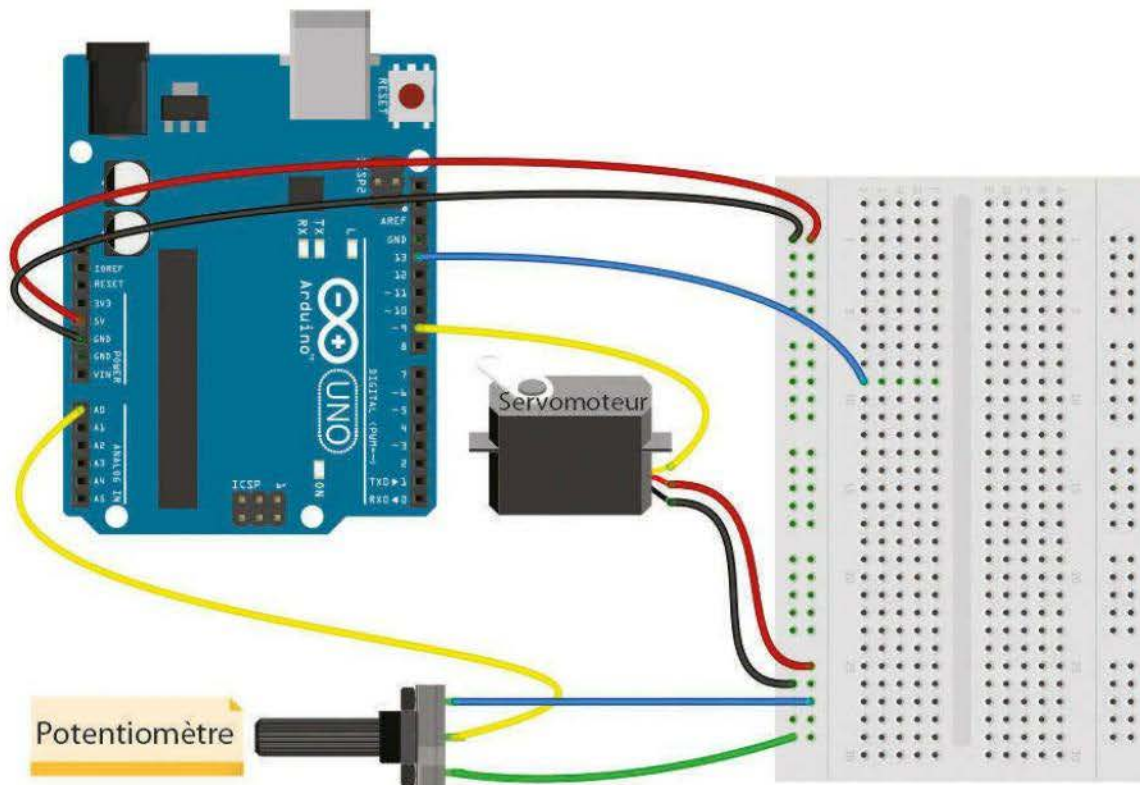
Je l'ai fait pour les composants utilisés dans ce projet et la catégorie *My Parts* se remplit peu à peu.

Figure 6-4 ►
Composants réunis
dans la catégorie My Parts



Il est désormais très facile de récupérer des composants dont je me sers souvent en les faisant glisser depuis ma catégorie personnelle.

Avant de passer au point suivant, j'aimerais vous présenter une autre méthode de raccordement très intéressante. Jusque là, les câbles cheminaient à angles droits, comme dans la majorité des schémas électriques. Mais pour que le projet paraisse plus réaliste, vous pouvez aussi créer des raccordements courbes en procédant comme suit :



▲ **Figure 6-5**
Raccordement courbes

La fonction des fils conducteurs courbes est désactivée par défaut. Lorsque vous avez établi une liaison et que vous voulez qu'elle fasse un coude à un emplacement précis, pointez à l'endroit voulu. Le pointeur de la souris se transforme :



◀ **Figure 6-6**
Pointeur doté d'une icône de coude

Créez un coude en faisant glisser la souris tout en maintenant le bouton gauche enfoncé. Pour obtenir un câble courbe, vous devez

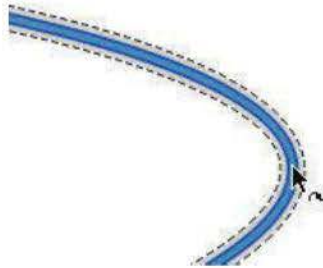
appuyer sur la touche Ctrl avant de cliquer sur le fil. Le pointeur de la souris se transforme :

Figure 6-7 ►
Pointeur doté d'une icône de courbe



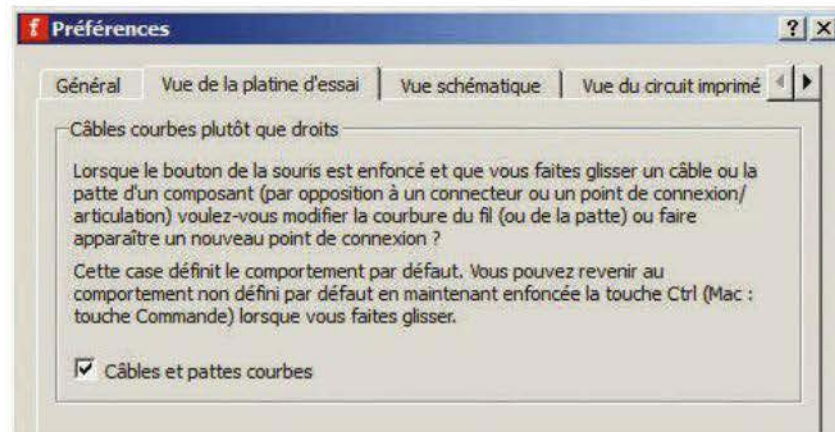
Vous pouvez maintenant couder le câble.

Figure 6-8 ►
Câble coudé



Vous pouvez activer cette fonction en permanence dans les Préférences qui sont accessibles depuis le menu *Édition* :

Figure 6-9 ►
Activation des câbles et pattes courbes



Lorsque vous cochez la case *Câbles et pattes courbes*, les câbles auront toujours une forme courbe. Cette option s'applique aussi à la vue schématique et à celle du circuit imprimé.

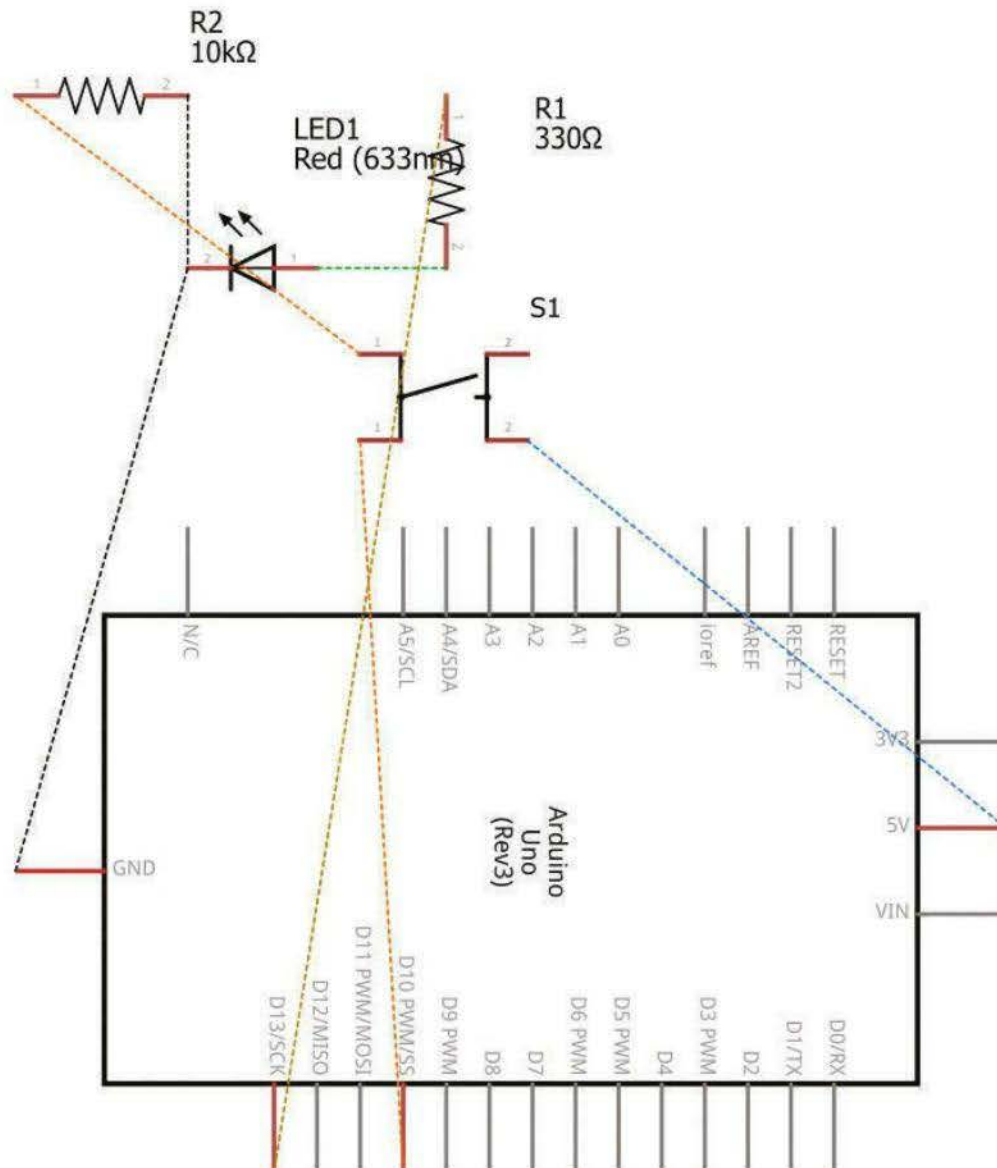
Vue schématique

Vous pouvez accéder à la vue schématique en cliquant sur l'onglet correspondant dans la partie supérieure de l'interface.

Cette vue présente tous les raccordements que vous venez d'établir sur la plaque d'essais. Vous serez frappé par le désordre qui semble y régner, mais ne vous inquiétez pas, car nous allons y remédier.

Pour commencer, je fais pivoter le symbole électrique de la carte Arduino afin qu'il corresponde à l'orientation choisie dans la vue de la platine d'essai. Ensuite, je positionne les composants d'une façon qui me paraît à la fois logique et claire. C'est évidemment une question de préférence personnelle.

▼ **Figure 6-10**
Vue schématique assez désordonnée d'un projet de bouton-poussoir



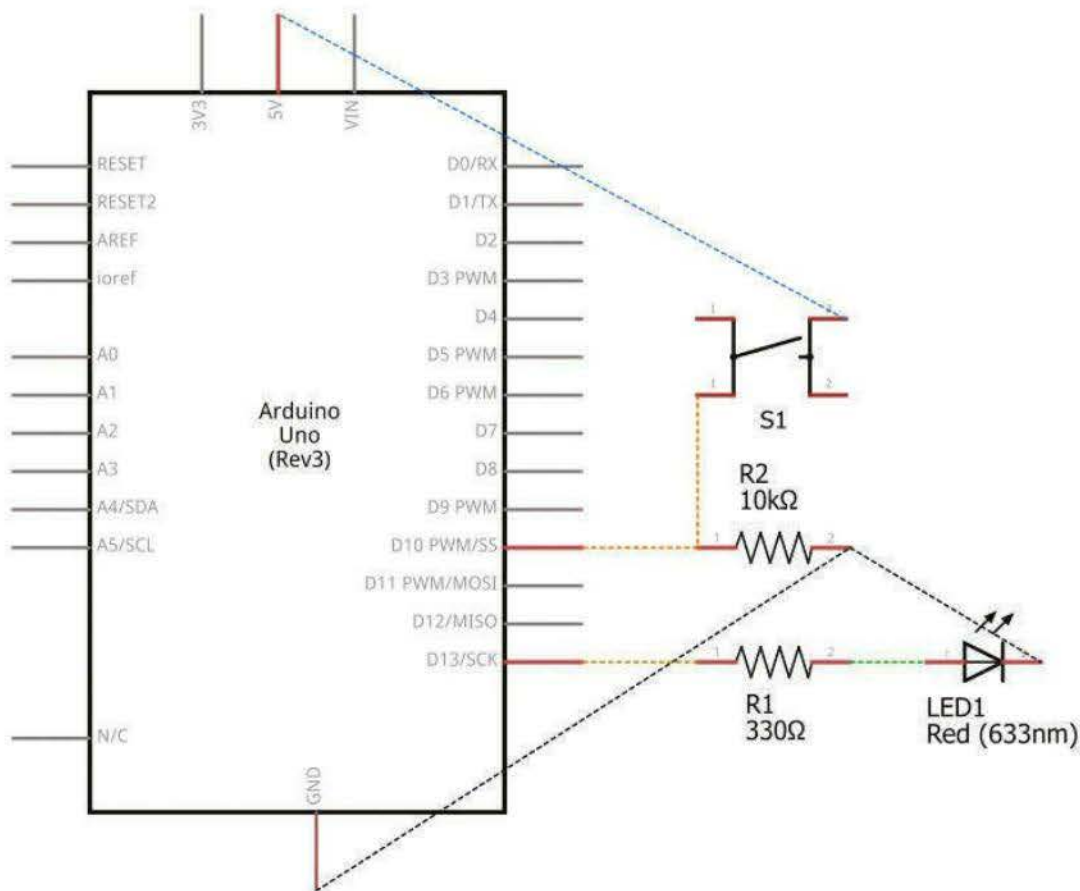
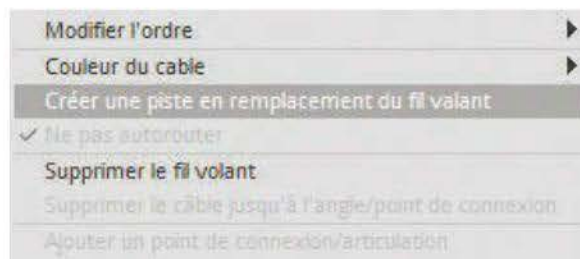


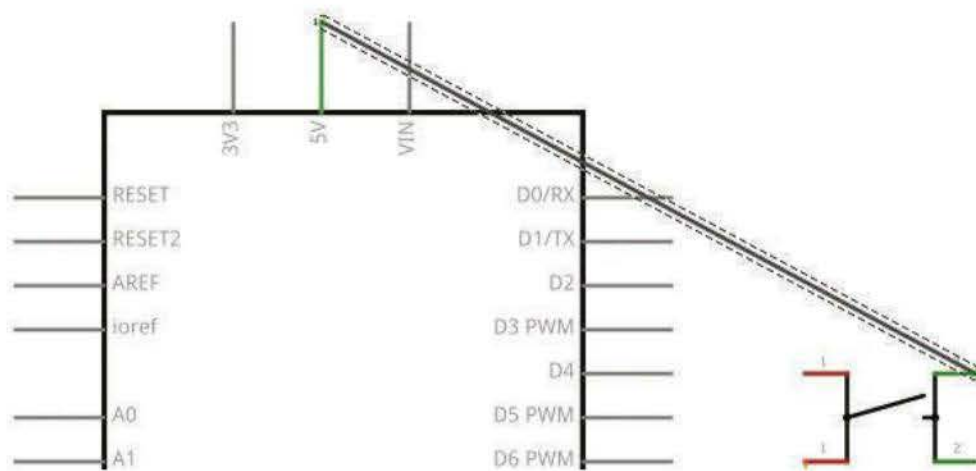
Figure 6-11 ▲
Vue schématique mieux rangée
d'un projet de bouton-poussoir

Les liaisons que vous reconnaissez ici sous la forme de lignes en pointillés correspondent au plus court chemin entre les contacts. Cela s'appelle un *chevelu*. Lorsque vous ouvrez le menu contextuel d'un fil du chevelu, vous pouvez voir les commandes suivantes :



La commande *Créer une piste en remplacement du fil volant* paraît très prometteuse, car, plus tard, nous aimerions que les différentes pistes relient électriquement tous les composants sur un circuit imprimé. Je vais vous présenter la procédure pour le fil qui relie la

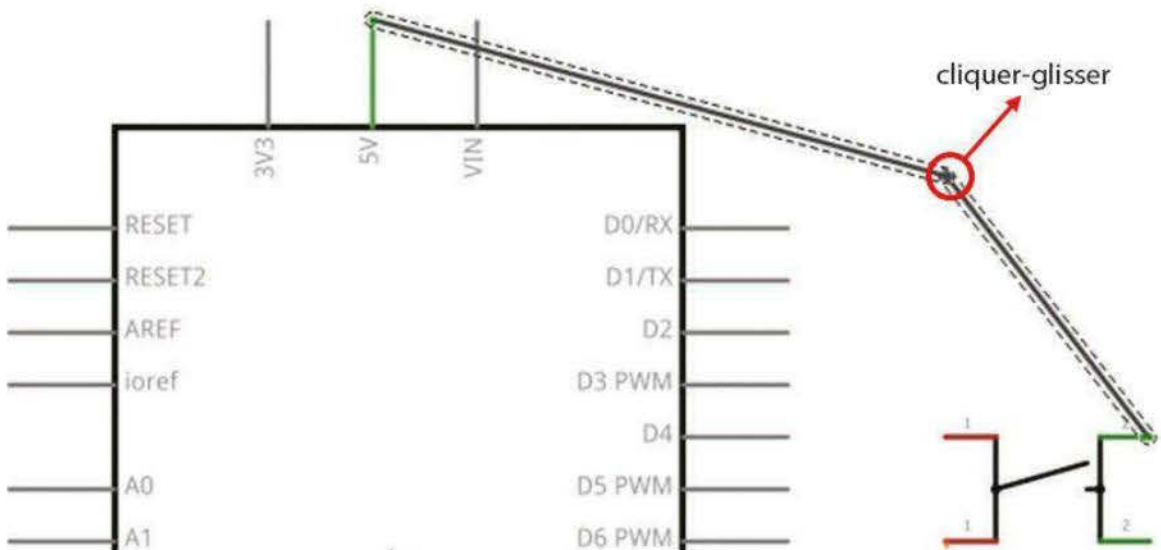
broche 5V à l'une des pattes du bouton-poussoir. Pour créer une piste en un tour de main, il suffit de double-cliquer sur le chevelu correspondant. Une fois la piste créée, la vue est la suivante.



▲ **Figure 6-12**
Enchevêtrement transformé en piste

Vous pouvez maintenant voir une ligne continue bordée de deux lignes pointillées. Sur une vue schématique, il est préférable qu'une piste ait un tracé rectiligne avec des changements de direction à angle droit, même s'il y a des exceptions. Cliquez sur une piste et insérez un point de flexion en faisant glisser la souris tout en maintenant le bouton gauche enfoncé. Vous pouvez déplacer le point de flexion, toujours en maintenant le bouton de la souris enfoncé.

▼ **Figure 6-13**
Ajout d'un point de flexion



Il est aussi très facile d'ajuster la position du point de flexion après avoir relâché le bouton de la souris. Pour supprimer un point de flexion, il suffit de double-cliquer dessus. Sur la figure suivante, j'ai ajusté la position de toutes les pistes à l'aide de points de flexion.

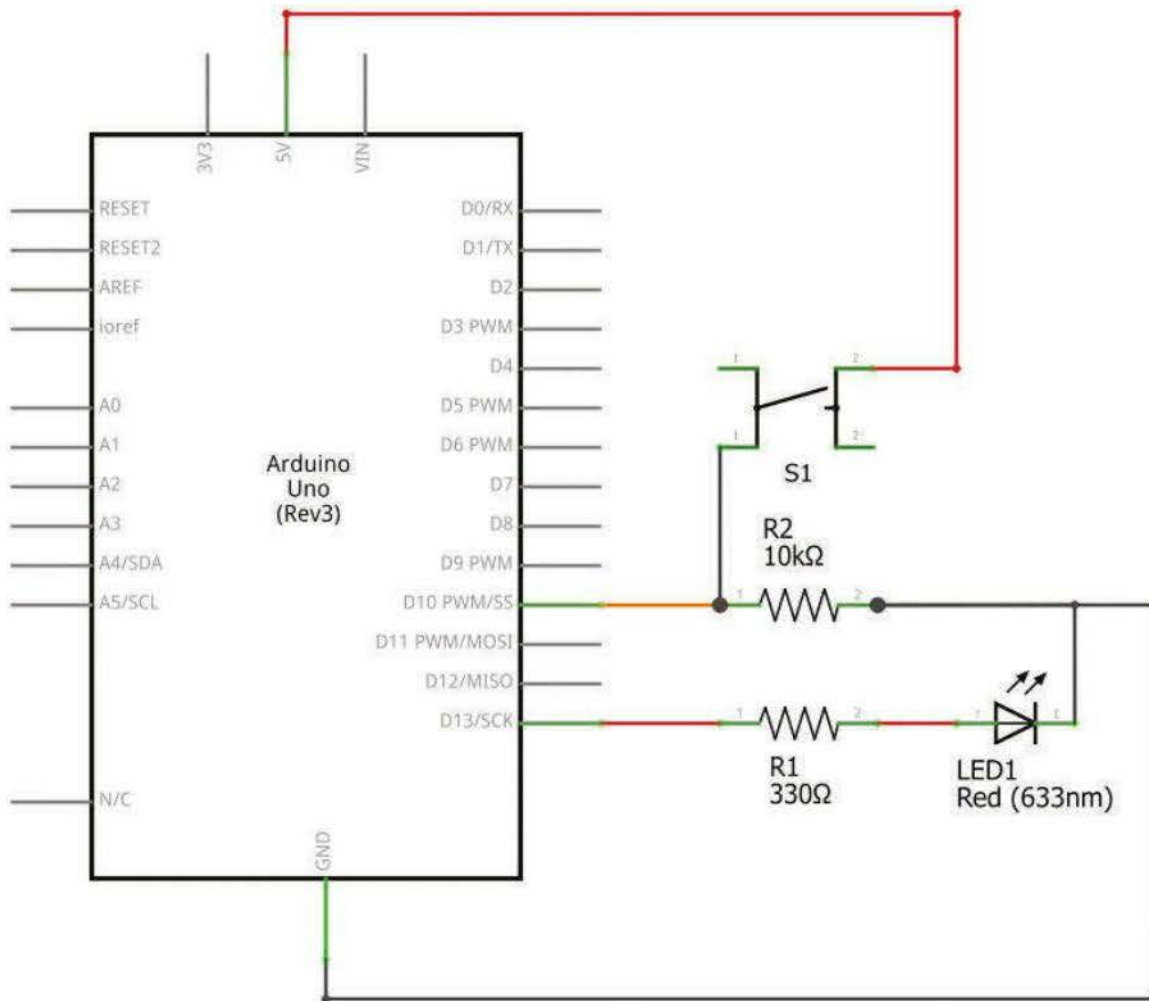


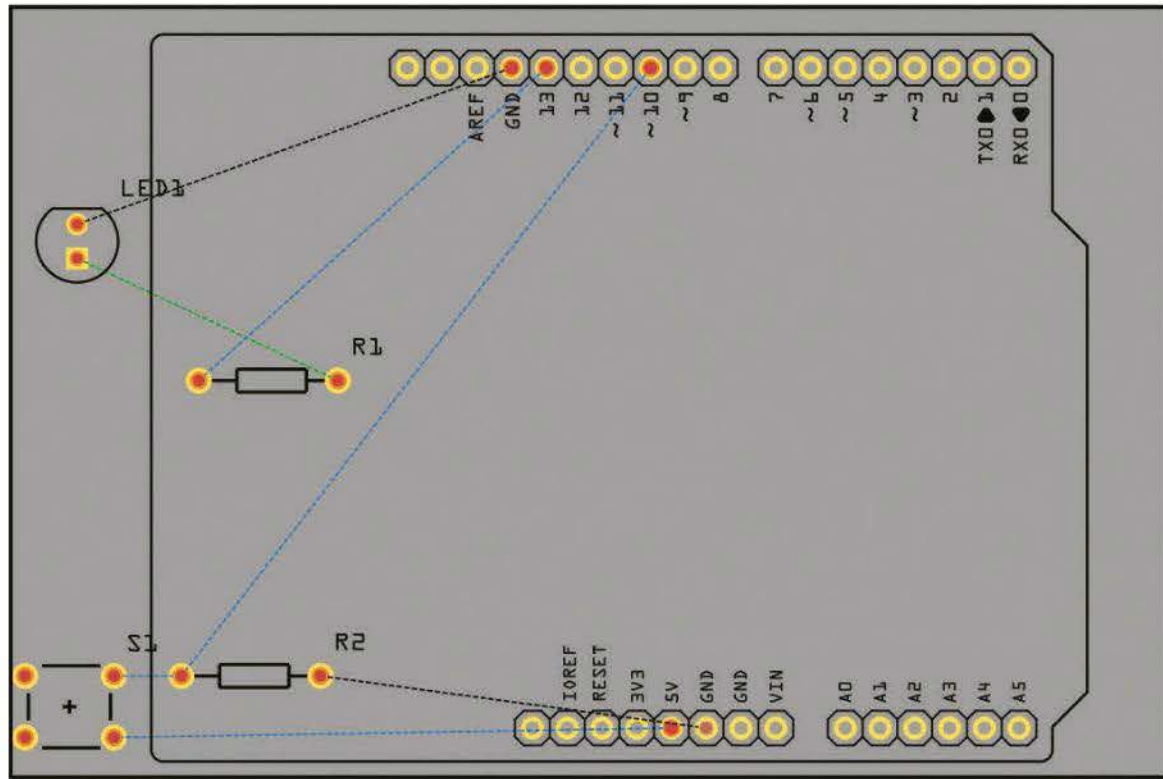
Figure 6-14 ▲ Nous en arrivons donc au troisième onglet : la vue du circuit imprimé.
Vue schématique terminée

Circuit imprimé

Vous pouvez accéder à la vue du circuit imprimé en cliquant sur l'onglet correspondant dans la partie supérieure de l'interface.



La vue du circuit imprimé présente aussi les raccordements que nous venons de créer sous la forme de cheminements. Toutefois, la vue montre les composants et la carte Arduino vue de dessus avec toutes ses broches, ainsi que les éléments et leurs raccordements. Là aussi, les liaisons vous paraîtront emmêlées et il va falloir y remédier.



Vous constaterez que les composants sont très mal placés, deux d'entre eux ne se trouvant pas même sur le circuit imprimé (ou *PCB*, *Printed Circuit Board*). Nous allons faire un peu de ménage par cliquer-glisser pour réorganiser les composants. Si vous en aviez disposé un grand nombre sur la plaque d'essais, il vous faudra peut-être essayer différentes dispositions avant d'aboutir à un résultat satisfaisant. La figure suivante a déjà meilleure allure, même si elle ne compte que quatre composants et qu'il n'y a donc vraiment pas de quoi crier au génie.

▲ **Figure 6-15**

Vue assez désordonnée du circuit imprimé

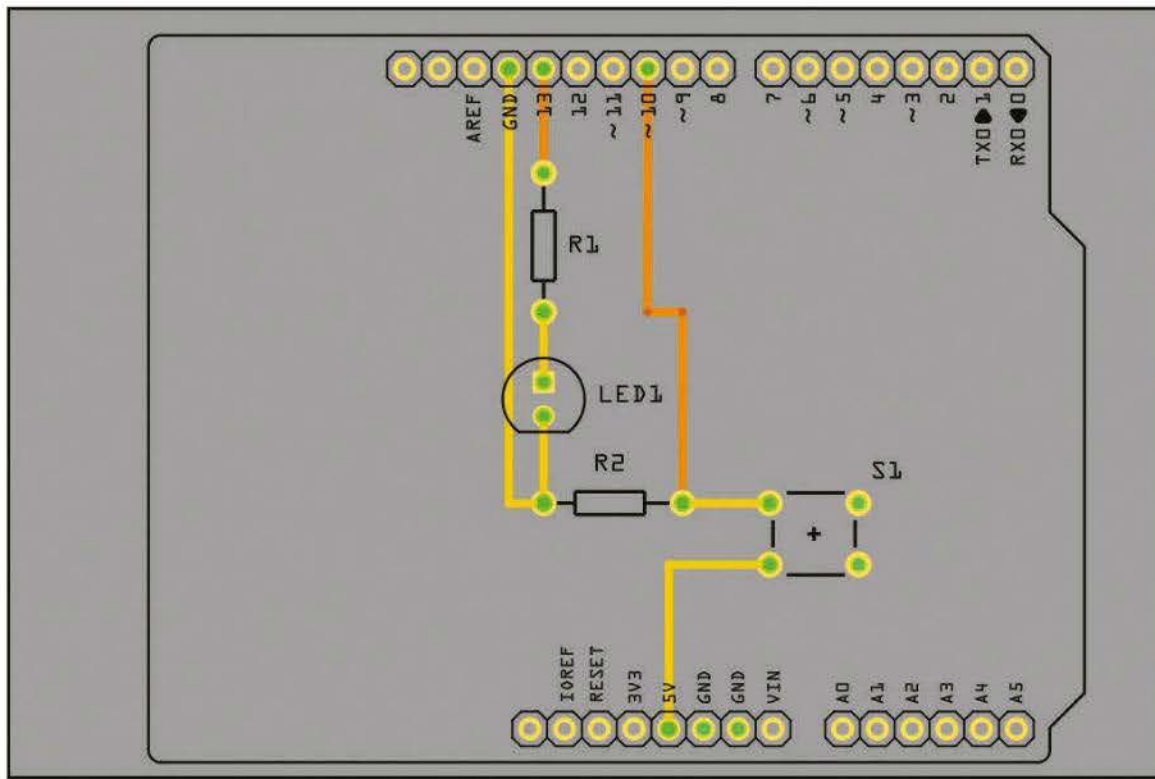


Figure 6-16 ▲
Vue du circuit imprimé
avec la disposition finale
des composants

J'aimerais en profiter pour vous signaler que les circuits imprimés ont évidemment une vue de dessus et une vue de dessous pouvant recevoir des pistes de cuivre. Dans cet exemple rudimentaire, une couche suffit (couche de cuivre de dessous), car les composants peuvent être disposés de telle façon que les pistes conductrices ne présentent pas d'intersections. Qu'est-ce qui vous saute aux yeux lorsque vous observez ces pistes conductrices ? Elles ont différentes couleurs. Certaines sont jaunes et d'autres orange. Il doit bien y avoir une raison à cette différence. Eh bien, c'est lié aux deux faces d'un circuit imprimé.

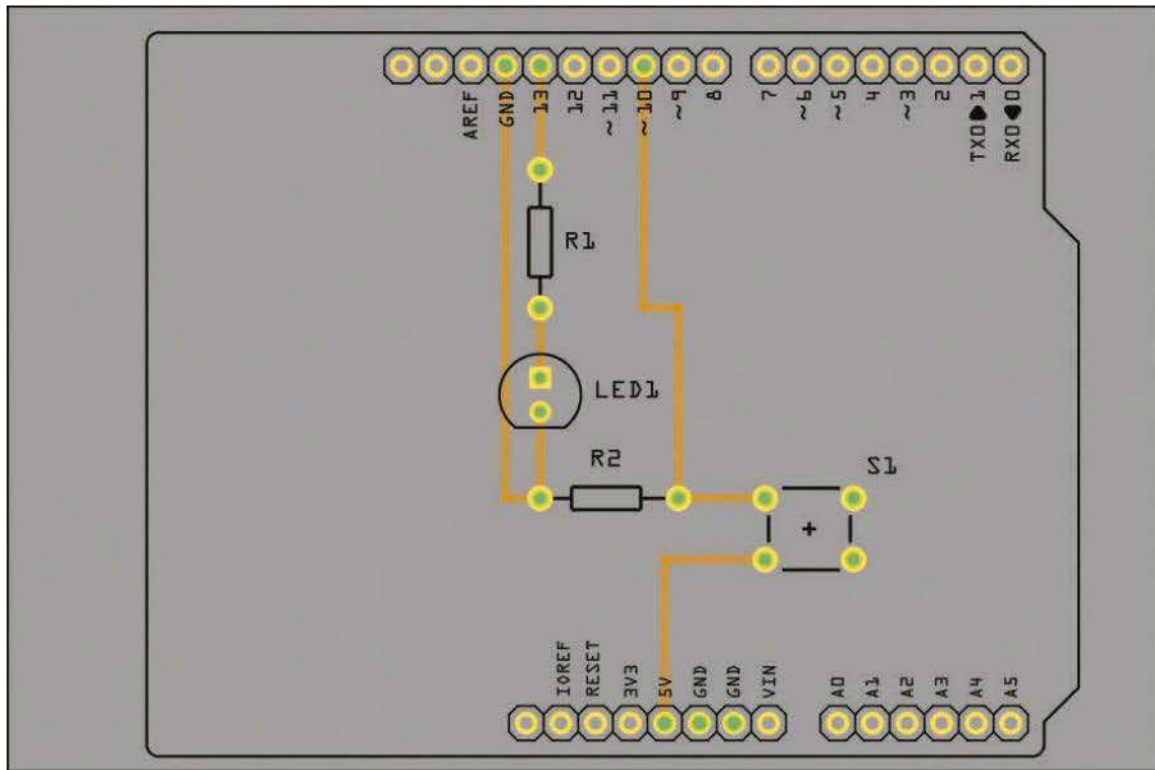
- jaune : les pistes conductrices se trouvent sur la couche supérieure ;
- orange : les pistes conductrices se trouvent sur la couche inférieure.

Dans cet exemple, les deux couches sont utilisées, ce qui est totalement infondé. La commande *Routage>Autoroutage* vous permet d'essayer des propositions de routage pour le cheminement des pistes conductrices. Si le résultat n'est pas satisfaisant, vous devrez intervenir manuellement. Nous voulons déplacer sur la face inférieure les

pistes conductrices qui sont signalées en jaune et qui se trouvent donc sur la face supérieure. Procédez comme suit.

- Sélectionnez successivement les différentes pistes conductrices jaunes en maintenant la touche Ctrl enfoncée (vous pouvez bien sûr aussi sélectionner individuellement chaque piste et activer la commande Déplacer vers l'autre côté de la carte dans le menu contextuel).
- Sélectionnez la commande *Routage>Déplacer vers l'autre côté de la carte*.

Le résultat est le suivant :



Lorsque vous êtes satisfait de votre circuit imprimé, vous pouvez le faire fabriquer par le *Fritzing Fab*. Pour obtenir un devis approximatif, sélectionnez la commande *Routage>Estimation des coûts avec Fritzing Fab*. Vous pourrez aussi connaître les tarifs en vigueur en survolant le bouton *Faire fabriquer* qui se situe dans le coin inférieur droit de la fenêtre.

▲ **Figure 6-17**

Vue du circuit imprimé avec les pistes conductrices sur la face inférieure

Figure 6-18 ►
Le bouton Faire fabriquer affiche des informations sur le coût de fabrication.

Commandez votre circuit imprimé sur Fritzing Fab

La surface totale du circuit imprimé 1 dans cette esquisse est de 47.78 cm² (7.4 in²). Utilisez Fritzing Fab pour produire un circuit imprimé à partir de votre esquisse. Profitez de notre remise quantitative :

Exemplaires	1	2	5	10
Prix par carte	33.45 €	28.67 €	19.11 €	19.11 €
Prix	33.45 €	57.34 €	95.57 €	191.15 €

Routage terminé

Faire fabriquer

Partager

Les tarifs de fabrication évoluent en fonction du nombre de cartes produites.

Le shield Arduino créé dans cet exemple utilise le facteur d'encombrement d'une carte Arduino Uno. Vous pouvez évidemment aussi fabriquer des shields adaptés aux autres cartes de la famille. Cliquez sur la platine pour la sélectionner. Ensuite, vous pouvez choisir d'autres formes de shields dans le panneau d'informations.

Figure 6-19 ►
Choix du type de shield Arduino

Propriétés

famille	microcontroller board (arduino)
type	Arduino UNO (Rev3) - ICSP
numéro du composant	Arduino MEGA ADK (Rev3) (icsp) Arduino Micro (Rev3) Arduino Mini (Rev5) Arduino Nano (3.0)
Mots-clés	Arduino Nano (3.0) ICSP
rev3, uno,	Arduino Nano v2.3
Connexion	Arduino Pro
conn.	Arduino Pro Mini (Rev13) Arduino UNO (Rev3)
nom	Arduino UNO (Rev3) - ICSP
type	

Le Fritzing Creator Kit

Pour les plus impatientes, Fritzing propose même un kit qui contient tout le nécessaire pour débiter dans l'univers du microcontrôleur avec une carte Arduino. Le kit contient au choix une carte Arduino Uno ou une carte Arduino Mega.



◀ **Figure 6-20**

Le Fritzing Creator Kit
avec une carte Arduino Uno

Ce kit bien pensé est tout sauf un fourre-tout de composants réunis à la hâte comme on en trouve à la pelle sur Internet. Il s'agit d'un ensemble harmonieux et mûrement réfléchi. À l'ouverture de la boîte, tout est bien rangé et à sa place, ce qui illustre l'amour du détail que l'on retrouve aussi dans toutes les expériences proposées.

Le kit contient également un manuel de 136 pages qui constitue une introduction à l'utilisation créative des composants électroniques avec la carte Arduino. Le livre aborde aussi bien les notions analogiques et numériques essentielles en Arduino que des connaissances plus avancées, comme la communication avec Processing, la commande des moteurs à courant continu et des servomoteurs, etc. Vous serez étonné de tout ce que vous pourrez réaliser à l'aide de ce kit !

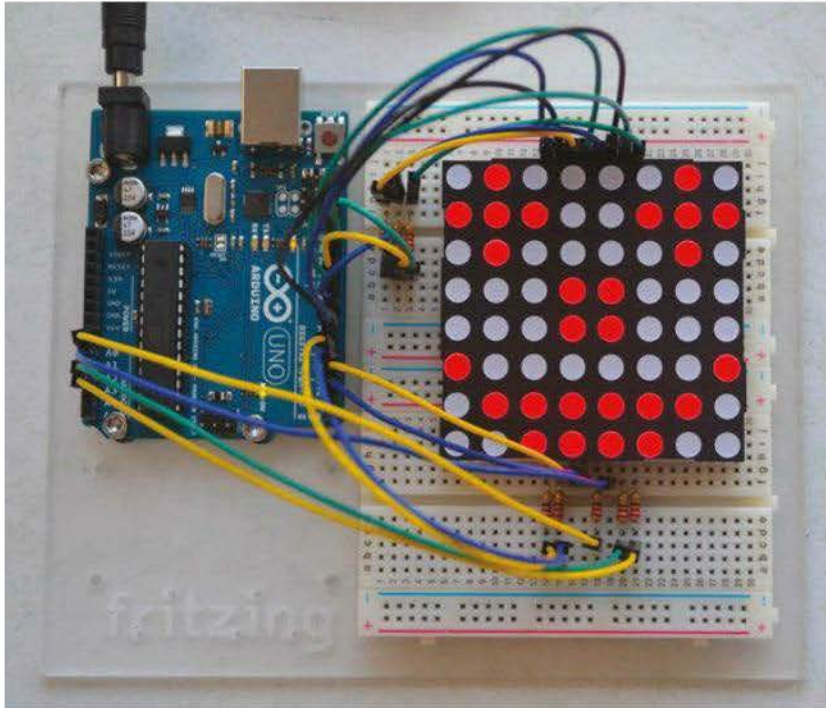
Figure 6-21 ▶
Le Fritzing Creator Kit ouvert



Figure 6-22 ▶
Le manuel associé



J'ai immédiatement été attiré par l'expérience proposée avec la matrice de LED. Pour éviter que les composants ne s'égarent dans la nature pendant la construction, le kit contient une plaque pouvant servir de support à la carte Arduino et à deux plaques d'essais. Votre expérience n'en sera qu'améliorée. En outre, comme vous disposez d'une bonne vue d'ensemble, vous résoudrez très rapidement vos erreurs éventuelles.



◀ **Figure 6-23**
Expérience de la matrice de LED

Le Fritzing Creator Kit réunit les composants suivants :

- manuel
- modèles en papier
- carte Arduino Uno R3 ou Mega
- matrice d'affichage à LED
- 2 plaques d'essais sans soudure
- plaque support
- câble USB
- capteur de lumière (LDR)
- potentiomètres
- interrupteur à inclinaison
- 2 boutons-poussoir

- LED RGB
- buzzer piézo
- 22 LED
- servomoteur
- moteur à courant continu (4,5 V à 9 V)
- résistances : 10 de 220 Ω , 10 de 100 k Ω
- transistor MOS-FET
- 24 cavaliers courts
- 14 cavaliers longs
- câbles d'extension
- clip pour pile de 9 V
- circuit L293D pont-H pour contrôle de moteur
- 3 vis + entretoise
- clé Allen (hexagonale)



Pour aller plus loin

Vous trouverez plus d'informations sur le Fritzing Creator Kit sur Internet :

<http://creatorkit.fritzing.org/>

Si vous ne voulez pas saisir manuellement les sketches des différentes expériences, vous pouvez les télécharger à l'adresse suivante :

<http://fritzing.org/creatorkit-code/>

Bon divertissement avec Fritzing !

L'assemblage des composants

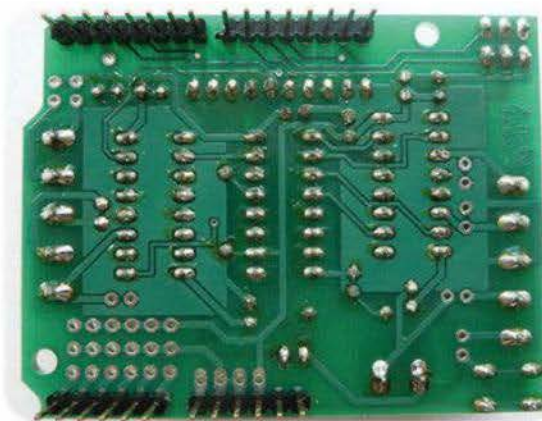
Les choses deviennent sérieuses et vous découvrez le matériel que vous pourrez bientôt utiliser. Vous connaissez maintenant les composants électroniques de base et il vous reste à savoir où et comment fixer ou relier entre eux les composants. Posons-nous les questions suivantes.

- Qu'entend-on par cartes ?
- Qu'est-ce qu'une plaque d'essais sans soudure, appelée également breadboard ?
- Qu'entend-on par cavaliers flexibles et à quoi servent-ils ?
- Peut-on éventuellement fabriquer ces cavaliers soi-même et à moindre coût ?

Qu'est-ce qu'une carte ?

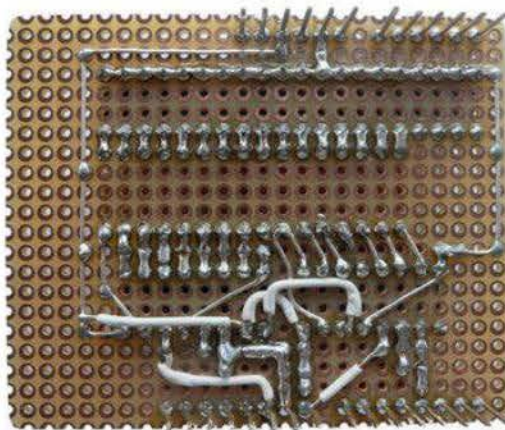
De nos jours, on utilise une carte pour pérenniser un circuit. Une carte est une mince plaque de quelque 2 mm d'épaisseur, fabriquée dans un matériau tel que bakélite ou époxy, qui sert de support à divers composants. Il existe différents types de cartes. Des pistes conductrices, permettant de raccorder électriquement entre eux les différents composants, sont gravées sur la face inférieure ou la face supérieure des cartes professionnelles. La figure 7-1 montre une carte de ce type.

Figure 7-1 ►
Carte professionnelle avec face
inférieure gravée
(Shield motor Arduino)



Vous pouvez bien entendu, avec l'équipement adéquat et une certaine adresse, arriver à ces résultats chez vous, mais cela demande beaucoup de travail. Pour aller plus vite, ce qui ne veut pas dire que le résultat ne sera pas bon et ne fonctionnera pas au bout du compte, vous pouvez utiliser une carte pour montage expérimental. Elle présente un grand nombre de trous perforés à intervalles normalisés (2,54 mm, en général), ce qui permet de placer plus librement les composants. Vous devez naturellement remplacer les pistes conductrices manquantes par des fils de liaison.

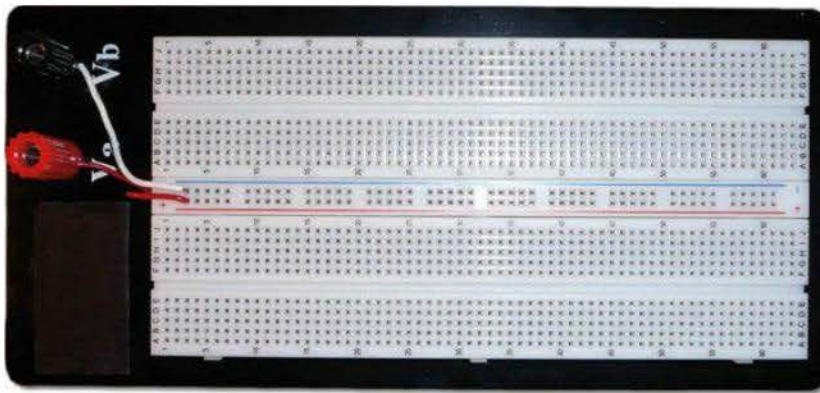
Figure 7-2 ►
Carte semi-professionnelle avec fils
de liaison ajoutés à la main



En prenant le temps de bien faire les choses, vous pouvez arriver en gros à ce résultat. Pour ceux que cela rebute, et je sais d'expérience que les novices se contentent au début de faire des expérimentations et ne souhaitent pas se lancer immédiatement dans la fabrication d'une carte qui risque de prendre du temps, il existe une solution plaisante qui permet de s'épargner travail et poussière.

La plaque d'essais sans soudure (breadboard)

La plaque d'essais sans soudure, également appelée breadboard, accueille des composants électriques et électroniques qui peuvent être reliés entre eux par des cavaliers flexibles. Les professionnels l'utilisent eux aussi pour vérifier ou corriger la capacité de fonctionnement des nouveaux circuits avant d'envisager leur fabrication en série sur des cartes gravées à l'avance.



◀ **Figure 7-3**
Plaque d'essais vue de dessus
(en position latérale de sécurité)

Cette plaque présente une multitude de petites prises femelles sur lesquelles composants et cavaliers peuvent être branchés à raison d'un seul branchement par prise.

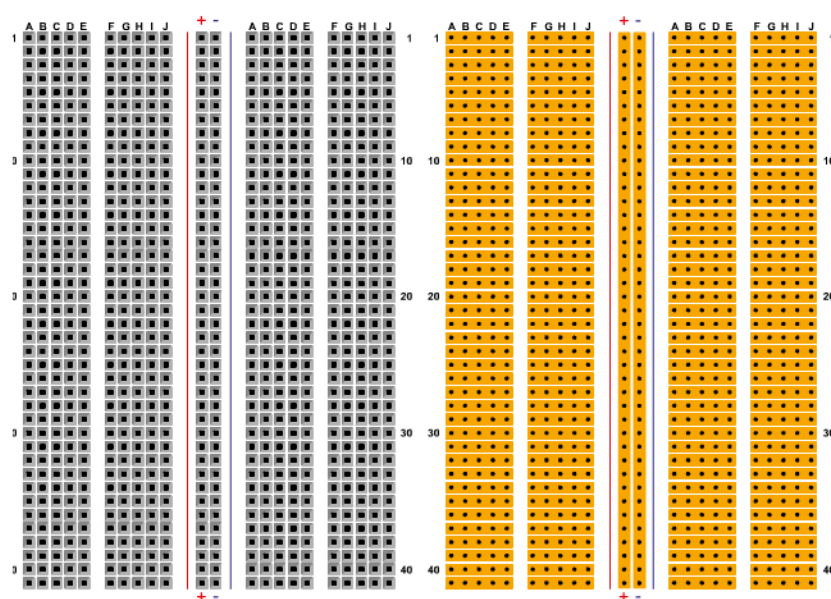
Si un seul branchement par prise est possible, comment fait-on pour relier les composants entre eux ? Je ne comprends pas très bien.

Beaucoup de prises sont reliées entre elles à l'intérieur de la plaque (en général, par groupes de cinq). Si l'une d'elles est utilisée, par un fil inséré dans le trou correspondant, celles qui lui sont liées sont donc disponibles pour être connectées à ce fil.

Vous avez ainsi en principe toujours assez de branchements libres pour établir les liaisons nécessaires. Reste à savoir comment ces liaisons invisibles sont agencées à l'intérieur de la carte. Voyez plutôt : les deux images de la figure 7-4 montrent une plaque d'essais vue de l'extérieur et de l'intérieur.

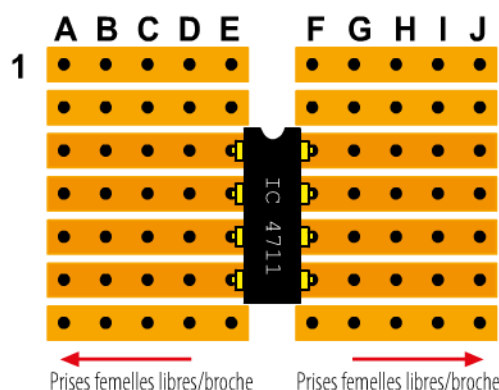


Figure 7-4 ►
Plaque d'essais vue de l'extérieur
(à gauche) et de l'intérieur
(à droite)



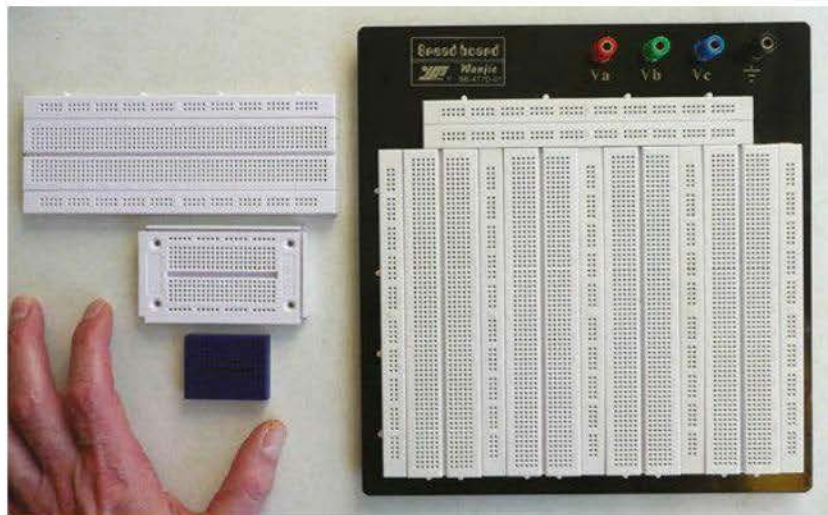
Si vous pouviez mettre les deux images l'une sur l'autre, vous verriez précisément quelles prises ont une liaison conductrice. Mais je pense que vous pouvez déjà vous faire ici une idée de ce qui va ensemble. Dans chacune des rangées (1 à 41), les prises A à E et F à J constituent un bloc conducteur. Les deux rangées de prises verticales au centre (+ et -) sont à disposition pour une alimentation éventuellement nécessaire à plusieurs endroits. Je vais maintenant brancher un composant à plusieurs broches sur la plaque pour que vous compreniez mieux l'avantage desdites liaisons internes.

Figure 7-5 ►
Circuit intégré sur une plaque
d'essais



Ce futur circuit intégré à 8 pattes est placé dans l'espace légèrement plus large entre les blocs de liaison A à E et F à J. Chaque broche dispose ainsi, à gauche comme à droite, de 4 prises supplémentaires

qui sont en liaison électrique avec elle. Vous pouvez y brancher d'autres composants ainsi que des fils. Il existe une quantité de plaques d'essais diverses dont la taille varie en fonction de l'usage.



◀ **Figure 7-6**
Les tailles varient entre assez petites et relativement grandes.



Attention !

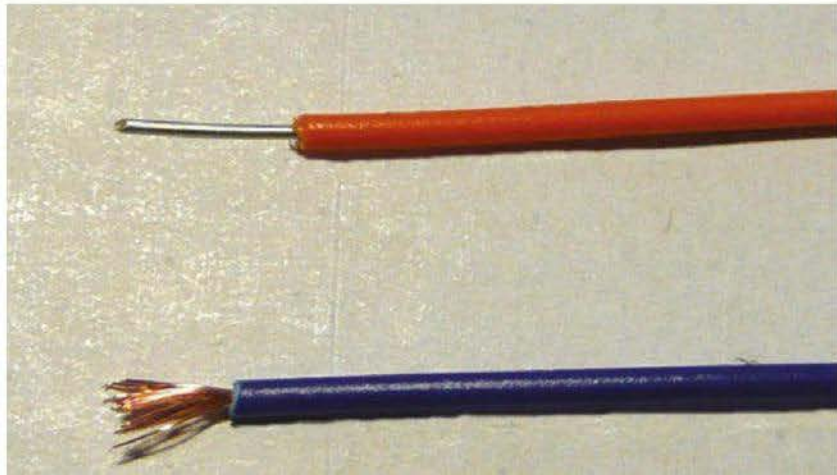
Il existe des plaques dont les barrettes de prises verticales ou *power-rails* sont électriquement interrompues au centre. Si vous n'êtes pas sûr d'avoir acheté la bonne carte, vous pouvez tester la continuité avec un multimètre en effectuant une mesure entre la broche la plus haute et la broche la plus basse d'une barrette de prises verticale. En l'absence de liaison, et si vous avez impérativement besoin d'une liaison électrique permanente, utilisez un cavalier.



Les câbles et leurs caractéristiques

J'aimerais profiter de cette occasion pour vous présenter les câbles employés. Examinons les deux câbles de la figure 7-7.

Figure 7-7 ►
Deux types de câbles

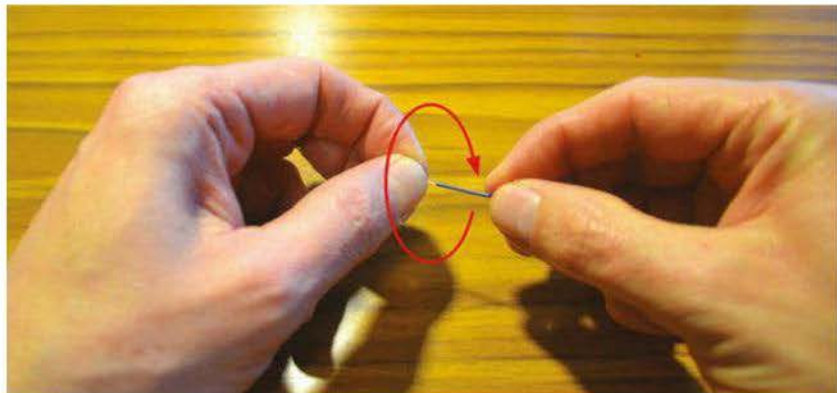


Ne me dites pas que l'un est orange et l'autre bleu. La différence apparaît clairement lorsqu'on les dénude, c'est-à-dire quand on retire leur gaine en plastique. Le câble orange, qui ne comporte qu'un seul fil, est un fil à âme pleine rigide. Son diamètre est généralement de 0,5 mm. Son usage est très répandu pour les raccordements sur les plaques de prototypage. Mais ce câble ne convient pas pour y raccorder des composants externes, comme les potentiomètres, car il n'est pas assez flexible. À la longue, il risque de se débrancher ou de rompre.

Le câble bleu, qui comporte plusieurs fils, est un câble multibrin. Son avantage par rapport au précédent est évident : plus souple, il peut être courbé ou déformé à volonté. Attention, si vous utilisez le câble dénudé comme sur la figure 7-7, les brins risquent de provoquer un court-circuit avec les contacts voisins. C'est pourquoi ces câbles ne sont pas utilisés tels quels, comme vous le verrez avec les cavaliers flexibles : ils doivent être étamés.

Tout d'abord, les brins sont entortillés comme sur la figure 7-8.

Figure 7-8 ►
Entortillage des brins



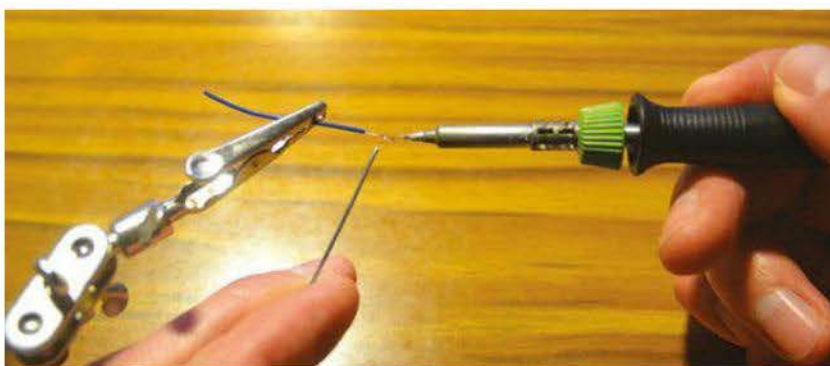
Prenez le câble en tenant l'extrémité dénudée entre le pouce et l'index de la main gauche. Ensuite, faites tourner l'autre extrémité entre le pouce et l'index de la main droite dans le sens horaire. Les brins s'entortillent comme illustré sur la figure 7-9.



◀ **Figure 7-9**

Les brins non étamés sont entortillés.

Les brins entortillés demeurent très fragiles et risquent de se dénouer sous l'effet d'une sollicitation mécanique. Pour y remédier, les brins entortillés sont donc étamés. Maintenez le câble en place au moyen d'une « troisième main », comme sur la figure 7-10, en veillant à ne pas endommager sa gaine par une pression excessive.



◀ **Figure 7-10**

Étamage à l'aide d'une troisième main

Rapprochez l'étain de soudure et la pointe du fer à souder chaud du câble entortillé de façon à ce que l'étain enveloppe immédiatement les brins lors d'un bref contact. Comme la soudure demande un peu de pratique, attendez-vous à devoir vous exercer avec quelques fils avant d'obtenir un résultat satisfaisant. Comme cela ne saurait tarder, ne vous laissez pas décourager par vos premiers échecs. La vitesse et la quantité d'étain apporté jouent un rôle décisif. Comparons les deux essais de la figure 7-11.

Figure 7-11 ►
Une bonne et une mauvaise
soudures

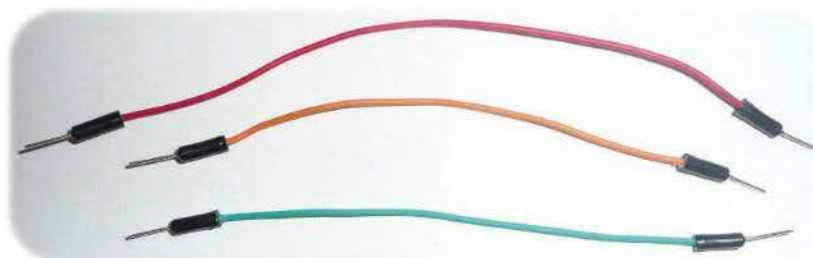


Le câble du haut, qui est plus long, a été correctement étamé, comme vous pouvez le voir d'après la répartition uniforme de l'étain. La soudure a légèrement augmenté le diamètre des brins, mais de façon à peine perceptible. En revanche, le câble du bas, plus court, a été recouvert avec beaucoup trop d'étain, ce qui a provoqué ce renflement important. Il n'y a qu'une chose à faire : pratiquer ! Comme je l'ai déjà dit, le risque de rupture est plus réduit avec les câbles flexibles qu'avec les câbles rigides. Lorsque vous aurez fabriqué quelques câbles comme décrit précédemment, pensez à tester leur continuité, surtout si vous comptez les utiliser pour des montages complexes sur un shield. Je vous montrerai un peu plus loin comment tester la continuité avec un multimètre.

Les cavaliers flexibles

Pour relier les composants à une plaque de prototypage, on utilise des cavaliers flexibles.

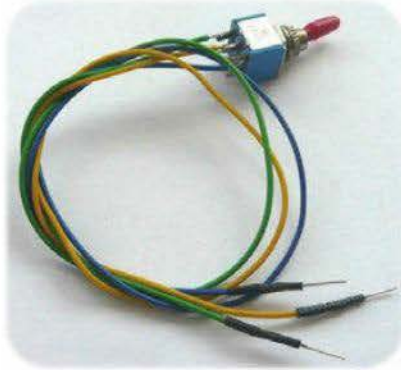
Figure 7-12 ►
Cavaliers flexibles du commerce
(pas chers)



Ils ont des couleurs et longueurs diverses et sont – je mentirais si je disais le contraire – de qualité médiocre. Ils sont cependant suffisants pour un débutant et on peut s'en procurer facilement chez un fournisseur de matériel électronique. On les appelle aussi câbles de liaison (*Low Cost Jumper Wires*).

Peut-on les fabriquer soi-même ?

Oui, j'ai moi-même fabriqué quantité de cavaliers, ce qui demande peu de choses. Avantage : on peut prendre un fil flexible – également appelé fil à brins multiples – de la section et de la couleur qu'on veut et bien entendu de la longueur qui convient. La figure 7-13 montre un commutateur sur lequel j'ai soudé trois cavaliers flexibles de ma fabrication.



◀ **Figure 7-13**

Cavaliers flexibles fabriqués
par moi-même sur un commutateur

Vous pouvez bien sûr en équiper tous les composants tels que potentiomètres, moteurs, servomoteurs et moteurs pas-à-pas. La mise en œuvre est plus souple et le temps passé à fabriquer les cavaliers ou les fils ne sera pas perdu !

Matériaux nécessaires pour fabriquer des cavaliers flexibles :

- fil de cuivre argenté (0,6 mm) ;
- fil à brins multiples de diamètre au choix (0,5 mm² maxi) ;
- gaine thermorétractable 3:1 (1,5/0,5).



◀ **Figure 7-14**

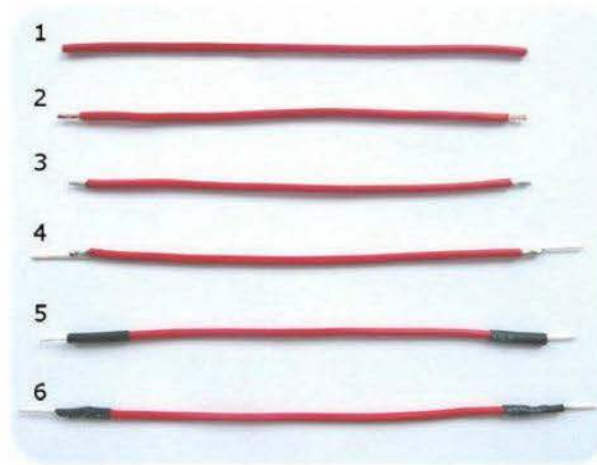
Matériaux nécessaires pour
fabriquer des cavaliers flexibles

Outils nécessaires à la fabrication des cavaliers flexibles :

- briquet ;
- fer à souder ;
- soudure à l'étain ;
- pince coupante diagonale et pince à dénuder le cas échéant.

Voici les différentes étapes de la fabrication.

Figure 7-15 ►
Les différentes phases
de fabrication d'un cavalier flexible



Étape 1

Coupez le fil multibrin à la longueur souhaitée.

Étape 2

Dénudez les deux extrémités du fil sur environ 0,5 cm.

Étape 3

Étamez les deux extrémités du fil.

Étape 4

Soudez le fil de cuivre argenté aux extrémités du fil multibrin.

Étape 5

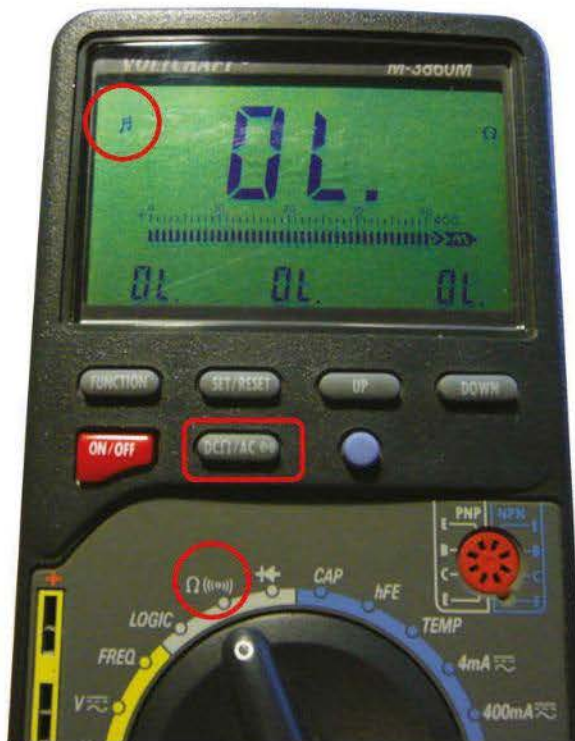
Enfilez les deux morceaux de gaine thermorétractable (1 cm environ) aux deux extrémités de manière à recouvrir à la fois les zones soudées et une partie de l'isolant du fil multibrin.

Étape 6

Chauffez avec un briquet 3 à 4 secondes les deux morceaux de gaine thermorétractable de manière à ce qu'ils se rétractent et épousent la forme du fil. Ne passez pas la flamme trop près de la gaine, faute de quoi elle cuit et n'a pas le temps de se rétracter.

Test de continuité avec un multimètre

Examinons un multimètre de plus près. Même s'il en existe de nombreux modèles, ils fonctionnent tous plus ou moins de la même manière. Sur la figure 7-16, j'ai entouré les principaux boutons et réglages de celui que j'utilise :



◀ **Figure 7-16**
Multimètre

La première étape consiste à définir la grandeur électrique à mesurer à l'aide du gros bouton rond. Comme ici nous voulons mesurer une résistance électrique, cela correspond sur mon appareil au symbole de l'ohm et du signal sonore. Lorsque les fiches ne sont pas reliées ensemble, on a théoriquement une résistance infinie entre les deux pôles, ce qui est signalé par *OL*. Si je rapproche les deux fiches en

créant un contact électrique, une résistance plus faible est affichée. Lorsqu'on utilise un multimètre pour effectuer un test de continuité, un signal sonore est émis, en plus de l'affichage de la résistance quand la liaison électrique a une résistance proche de 0 ohm. Sur certains modèles de multimètres, cette fonction doit d'abord être activée. Sur mon appareil, je dois appuyer sur le bouton encadré en rouge afin qu'un symbole de signal sonore apparaisse dans le coin supérieur gauche de l'écran. Si un signal sonore retentit quand je relie les fiches, cela signifie que la liaison électrique existe et qu'il y a donc bien continuité. Quand vous construirez vos propres shields pour votre carte Arduino, un multimètre sera indispensable pour localiser les défaillances.

Quand vous effectuez un test de continuité, vérifiez que le circuit a préalablement été déconnecté de la source de tension. Sinon, cela risque d'endommager le multimètre.

Le matériel utile

Les bons outils faisant les bons ouvriers, je ne peux que vous recommander le matériel mentionné ci-après. Évitez seulement de racheter les outils que vous avez déjà !

Si vous avez déjà vu à quoi ressemble un laboratoire d'électronique professionnel ou un atelier d'électronique, et si le domaine vous intéresse vraiment, vous avez sûrement eu du mal à contenir votre enthousiasme. La diversité des appareils de mesure avec leurs nombreux câbles colorés et la multiplicité des outils laissent le profane à la fois perplexe et émerveillé. C'est du moins ce que j'ai ressenti quand mon père m'a montré son poste de travail. Il travaillait alors sur l'une des nombreuses souffleries du Centre aéronautique et spatial allemand.

Je classe les outils présentés ici en deux catégories :

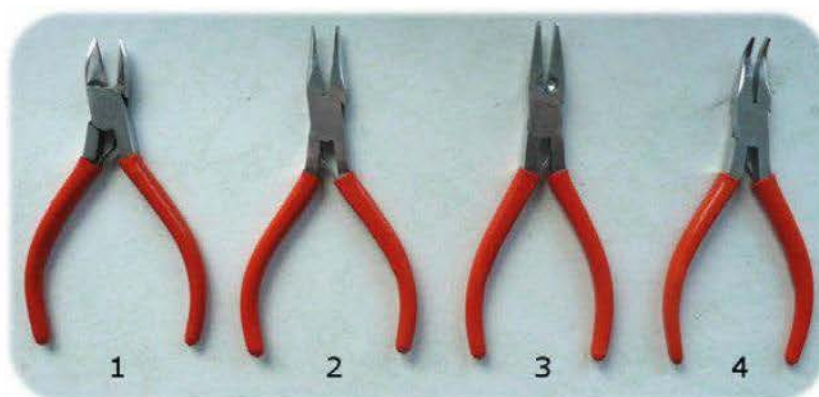
- Catégorie 1 : les *must have!*
Outils nécessaires pour votre travail.
- Catégorie 2 : les *nice to have!*
Outils non indispensables, mais qui peuvent faciliter votre travail.

Pinces diverses

La figure 8-1 présente les pinces qui vous seront les plus utiles :

- la pince coupante diagonale pour couper les fils ;
- la pince à becs longs pour tenir et placer des petits composants ;
- la pince universelle, pour tenir un objet avec plus de force ;
- la pince à becs coudés, permettant de tenir un élément caché ou peu accessible.

Figure 8-1 ►
Pincettes diverses



Ce jeu de pincettes fait à mon avis partie de la catégorie *Must have!*

Pince à dénuder

Cette pince permet de dénuder plus facilement les fils. Une pince coupante diagonale peut convenir, mais vous risquez de couper le fil si vous appuyez trop fort.

Figure 8-2 ►
Pince à dénuder



Cette pince, selon moi, relève de la catégorie *Nice to have!*

Tournevis

Les petits tournevis d'horloger sont parfaits pour fixer les fils aux bornes à vis, comme vous pouvez le voir sur la photo suivante.



Attention !

Les tournevis d'horloger ne sont pas isolés et sont donc conducteurs car entièrement métalliques. Vous ne devez en principe travailler sur un circuit que si celui-ci est vraiment hors tension.



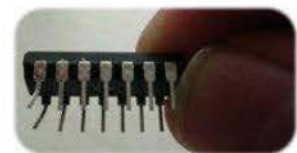
◀ **Figure 8-3**

Jeu de tournevis d'horloger

Si vous avez fixé un circuit intégré sur une plaque d'essais mais que vous souhaitez l'enlever sans tordre ses pattes de raccordement à 90° et risquer de les casser, vous pouvez utiliser un tournevis d'horloger approprié.

Si vous utilisez seulement vos doigts, vous risquez de tordre les pattes, comme sur la photo ci-contre.

Vous devez donc toujours être très soigneux avec les pattes fragiles d'un circuit intégré. Si ce qui arrive ici ne se produit qu'une ou deux fois, cela n'est pas grave. Mais des pattes de raccordement trop sollicitées peuvent vite lâcher. Les tournevis d'horloger font assurément partie de la catégorie *Must have!*.



Extracteur de circuit intégré

L'effet de levier étant ce qu'il est, détacher un circuit intégré d'une plaque d'essais ne devrait quand-même pas être chose impossible avec un tournevis. Un électronicien pur jus utilisera toutefois un outil spécial qui ne coûte pas très cher. Ce dernier ressemble à la pince à sucre de nos grands-mères et peut effectivement servir à cela faute d'autre chose. Il a cependant été conçu au départ pour détacher un circuit intégré (par exemple d'une plaque d'essais). Je dirais qu'il fait partie de la catégorie *Nice to have!* Il n'est en effet pas indispensable car d'autres moyens peuvent être utilisés à condition de procéder avec précaution.

Figure 8-4 ►
Extracteur de circuit intégré



Troisième main

Si vous avez déjà fait de la soudure, vous aurez probablement été confronté au problème suivant : d'une main, vous tenez la carte, de l'autre, l'étain, et de la troisième... Ah, mince ! Il vous faudrait une troisième main pour tenir le fer à souder. Comme vous n'êtes pas un mutant, vous pouvez toujours bloquer la carte contre un objet lourd, comme un livre. Mais la stabilité n'est pas garantie. Pourquoi ne vous serviriez-vous pas d'une véritable « troisième main » ? Il s'agit d'un outil très utile pour le travail de précision, car il vous laisse les mains libres.

Vous pouvez fixer la carte au moyen de deux pinces, tandis que la loupe vous permet d'avoir une vue grossie de la zone à souder. Vous avez ainsi les deux mains libres et vous pouvez placer précisément vos points de soudure aux endroits requis.



◀ **Figure 8-5**
Troisième main

Je me permets simplement de vous recommander d'être prudent avec la loupe. Pas de panique, vous ne risquez pas de vous abîmer les yeux. Mais j'ai bien failli provoquer un accident lorsque j'avais laissé la troisième main sur mon bureau près de la fenêtre sans remarquer que les rayons du soleil passaient à travers la loupe. Le plateau de la table a fini par prendre feu ! Si je n'avais pas été présent, je n'ose pas imaginer ce qui aurait pu se passer. Soyez donc bien prudent et ne laissez pas au soleil cet outil moins inoffensif qu'il n'y paraît.

Multimètre numérique

Un multimètre numérique est un appareil de mesure multiple capable de détecter et mesurer des grandeurs électriques.

Figure 8-6 ►
Multimètres numériques divers



Les appareils du marché offrent une gamme de mesures plus ou moins large. La plupart d'entre eux disposent cependant des fonctionnalités suivantes :

- mesure de la résistance d'un composant ;
- test de continuité d'un circuit (contrôleur sonore de continuité) ;
- mesure de tension continue et de courant continu ;
- mesure de tension alternative et de courant alternatif ;
- détermination de la capacité des condensateurs ;
- test de fonctionnement des transistors.

Comme vous pouvez le constater, ces options sont nombreuses et en principe suffisantes. Cet appareil de mesure fait donc partie de la catégorie *Must have!* Son prix varie en fonction du nombre de fonctionnalités, mais tous permettent généralement de mesurer la résistance, de tester la continuité des circuits et de mesurer le courant ou la tension. Le multimètre le plus simple coûte moins de 10 € et vous permet déjà de bien travailler. Les plus chers disposent naturellement de fonctions supplémentaires qui sont cependant toutes inutiles au débutant – ce qui explique qu'il fasse partie de catégorie *Nice to have!* La décision dépend du porte-monnaie. Si vous voulez investir sur le long terme et si vous avez de la place, vous pouvez acheter un multimètre de table.



◀ **Figure 8-7**
Multimètre numérique de table

Un instrument de ce type présente plusieurs avantages : il est doté d'un écran vertical plus lisible, sa précision est nettement supérieure à celle d'un multimètre basique et il est toujours prêt à l'emploi, car il se branche sur le secteur. Son plus gros atout à mes yeux, c'est que je sais toujours où il est !



Attention !

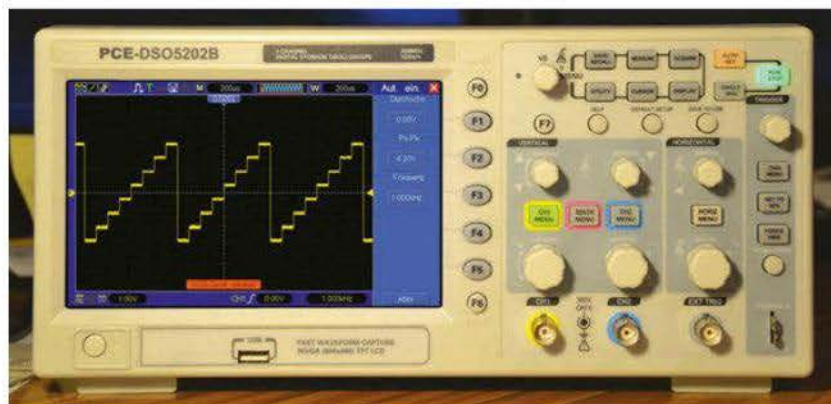
Avant de mesurer quoi ce soit avec votre multimètre, vous devez vous assurer que le bouton est bien réglé sur la grandeur électrique à mesurer. Si, par exemple, après avoir déterminé la résistance d'un composant (une résistance se détermine toujours hors tension) vous mesurez une tension adjacente, oublier de régler le mode de mesure peut endommager le multimètre.

Oscilloscope

L'oscilloscope est d'emblée un appareil de mesure haut de gamme. Il peut par exemple représenter graphiquement des courbes de tension en fonction du temps et permet entre autres de détecter les défaillances avec brio.

Il fait partie de la catégorie *Nice to have!*. Cet appareil est cependant très plaisant à utiliser et certains modèles conçus pour les débutants sont en vente à moins de 300 €.

Figure 8-8 ►
Oscilloscope PCE-DSO5202B

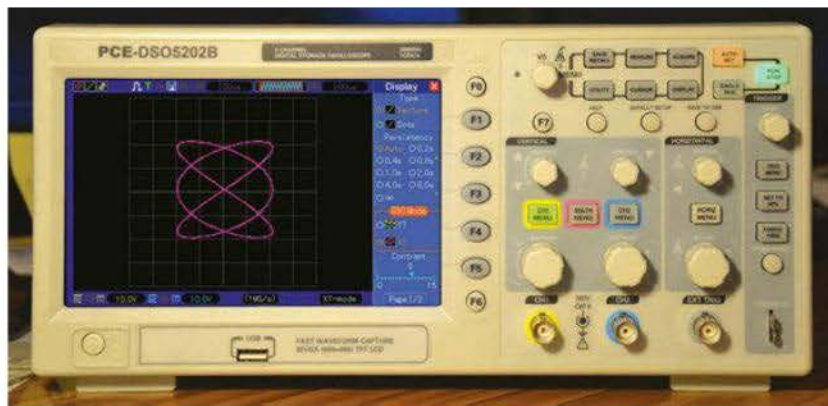


Je serai amené à utiliser plusieurs fois un oscilloscope dans ce livre, pour montrer les courbes d'évolution en fonction du temps des tensions à certains points de mesure d'un circuit. Il s'avère très utile pour faire des démonstrations et aider à comprendre des processus complexes.

Le modèle de la figure 8-8 est un oscilloscope enregistreur de type PCE-DSO5202B à deux canaux d'entrée. Il est doté d'une largeur de bande pouvant atteindre 200 MHz et d'un grand écran couleur de 7 pouces. Voici ses principales caractéristiques :

- Résolution : 800 × 480 pixels
- Nombre de couleurs : 64 000 couleurs
- Fonction réglage automatique (auto set)
- Port USB
- Nombreuses fonctions de mesure automatiques
- Balayage mono-coup (Single-Shot)
- Sensibilité verticale : 2 mV à 5 V/div
- Résolution verticale : 8 bits
- Fonctions mathématiques : addition, soustraction, multiplication, division, analyse FFT, interpolation : $\sin(x)/x$
- Mémoire horizontale : 1 Méc
- Vitesse d'échantillonnage horizontale : 1 Géc/s
- Signal de déclenchement : front, largeur d'impulsion, vidéo, réglage libre du niveau de déclenchement
- Type de déclenchement : auto, normal, single
- Couplage du déclencheur : AC, DC, LF rej, HF rej

Si vous ne comprenez pas bien certaines fonctions de l'oscilloscope, vous n'avez pas nécessairement besoin de vous plonger dans son mode d'emploi, car le fonctionnement de la plupart des boutons est expliqué directement sur l'écran. Je trouve cela extrêmement pratique, même pour les débutants. Si vous voulez reprendre une courbe affichée à l'écran dans une documentation ou une présentation, vous n'êtes pas obligé de la prendre en photo : il vous suffit d'appuyer sur le bouton d'enregistrement pour que l'image soit transférée sur une clé USB branchée sur l'un des ports correspondants. Peut-être avez-vous déjà entendu parler des courbes de Lissajous. Il s'agit d'ellipses pouvant être visualisées en mode XY. Ces figures peuvent également être créées au moyen d'un testeur de composant qui génère la courbe caractéristique d'un composant électronique.



◀ **Figure 8-9**
L'oscilloscope PCE-DS05202B
affichant une courbe de Lissajous

Pour plus d'informations, vous pouvez effectuer une recherche dans Google avec les mots-clés « oscilloscope » et « testeur de composants ».

Alimentation externe

Votre carte Arduino est certes alimentée en courant par un port USB et cela suffit bien pour quelques expérimentations, mais nous touchons à des circuits censés nous permettre par exemple de commander un moteur, lequel a besoin de plus de « jus » pour fonctionner. Dans ce cas, une alimentation externe est indispensable sinon la carte Arduino pourrait être endommagée.

Figure 8-10 ►
Alimentation stabilisée
pour laboratoire



Tout dépend bien sûr de l'usage qu'on veut en faire, mais un bloc secteur coûte en règle générale bien moins cher qu'une alimentation réglable de laboratoire.

Figure 8-11 ►
Bloc secteur



Le bloc secteur présenté ici propose différentes tensions de sortie sélectionnables par le biais d'un petit commutateur rotatif. Les tensions 3 V, 5 V, 6 V, 7,5 V, 9 V et 12 V peuvent être choisies. Une autre caractéristique est le courant maximal que l'alimentation est capable de délivrer. Plus ce courant est élevé, plus l'unité coûte cher. Le courant maximal du bloc secteur présenté ici est de 800 mA tandis que celui de l'alimentation réglable de laboratoire est de 1,5 A. Les prix sont sans limite vers le haut tout comme beaucoup de choses dans la vie. Cette alimentation de laboratoire coûte 140 € environ alors que le bloc secteur ne coûte lui que 15 € environ. La construction suivante vous permet d'alimenter votre carte Arduino avec une pile 9 V monobloc.



◀ **Figure 8-12**
Alimentation par pile de 9 V

Sont nécessaires :

- un clip pour pile de 9 V ;
- une prise de 2,1 mm ;
- une pile de 9 V.

La figure 8-13 montre comment souder le clip de pile et la prise.



◀ **Figure 8-13**
Alimentation par pile de 9 V

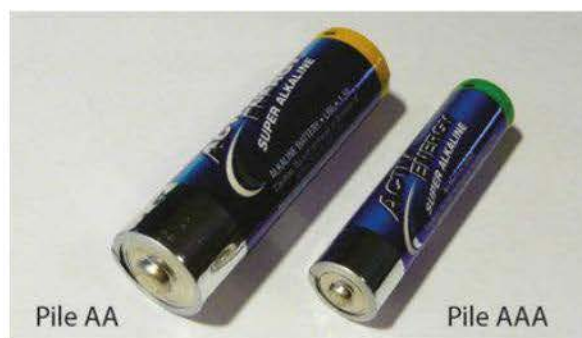
Respectez impérativement la polarité : le pôle plus (+) se trouvant au centre de la prise et le pôle moins (–) sur le manchon métallique extérieur. Contrôlez, pile branchée, la polarité des bornes avec un multimètre avant de brancher la prise sur votre carte Arduino. La photo suivante présente un coupleur de piles.

Figure 8-14 ►
Coupleur de piles



Il existe différents modèles de coupleurs pouvant recevoir deux, quatre, six ou même huit piles de type AA ou AAA.

Figure 8-15 ►
Piles d'1,5 V



Selon le modèle de coupleur de piles choisi, vous disposerez de tensions atteignant 3 V, 6 V, 9 V ou 12 V. Sur celui de la figure 8-14, vous pouvez noter en haut à gauche la présence d'un connecteur pour pile de 9 V. Bien pratique ! Je m'en sers pour raccorder la pile de 9V illustrée plus haut.

Gabarit de pliage pour résistances

Quand j'en suis venu à parler de cet outil dans ce livre, j'ai dû faire des recherches pour avoir son appellation exacte. J'ai fouiné un bon moment sur Google et fini par découvrir que son véritable nom était gabarit de pliage pour résistances. Si quelqu'un me l'avait demandé

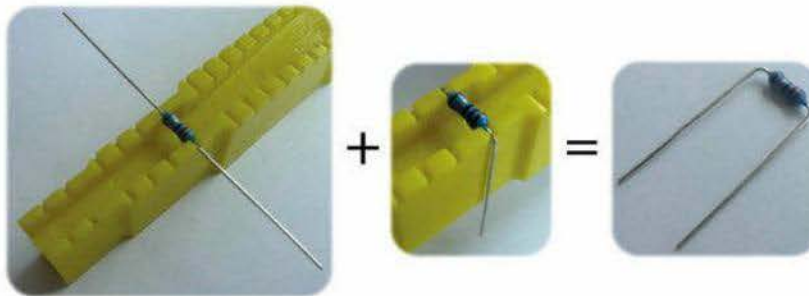
avant... Le fait est qu'il s'agit d'un objet en plastique permettant de plier, non pas la résistance elle-même, mais ses fils de raccordement.



◀ **Figure 8-16**

Gabarit de pliage pour résistances
(ou gabarit de pliage tout court)

Il n'a l'air de rien mais c'est un outil très utile. Il fait pour moi partie de la catégorie *Must have!* Cela peut vite tourner au fiasco si vous essayez de plier au jugé les fils de raccordement pour qu'ils rentrent dans les trous de la plaque d'essais. Je trouve que la fabrication et l'aspect d'une carte confinent à l'art et à l'esthétisme. Ça ressemble à quoi quand les composants sont placés de travers et penchés ? On se dit que le quidam ne s'est pas foulé ou n'avait pas le bon outil. Comme je vous l'ai dit déjà, l'intervalle entre les trous est de 2,54 mm sur une carte standard. Ce gabarit propose également d'autres dimensions de pliage pour résistances (idem pour les diodes). On les place dans des rainures, on plie ensuite les fils de raccordement vers le bas avec les doigts et on a un intervalle bien parallèle entre les fils, lequel est toujours un multiple de l'intervalle entre les trous. Le composant va parfaitement sur la plaque d'essais.



◀ **Figure 8-17**

Posez, pliez, c'est fait !

Vous n'êtes bien sûr pas obligé d'utiliser ce procédé pour placer des composants sur une plaque d'essais puisque le circuit n'y est jamais définitif. Tout ne doit pas être parfait comme sur une carte. Il ne faut pas pour autant faire n'importe quoi car un court-circuit peut vite perturber le fonctionnement du circuit et même griller des composants.

Fer à souder et soudure à l'étain

Un fer à souder est indispensable au bricoleur et fait donc partie pour moi de la catégorie *Must have!*. D'ailleurs, vous en aurez besoin pour réaliser certains montages du livre, à moins d'utiliser une station de soudage. Voici deux stations d'un excellent rapport qualité-prix.

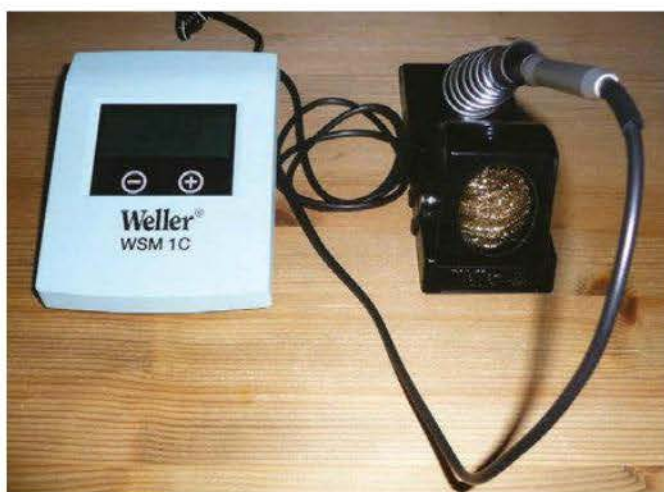
La station de soudage Ersa

Figure 8-18 ►
Station de soudage Ersa
avec différentes pannes



La station de soudage Weller

Figure 8-19 ►
Station de soudage Weller



L'avantage de la station de soudage sur le fer à souder est que la température de la panne est réglable, ce qui peut être vital pour des composants sensibles à la température comme les circuits intégrés. On trouve des fers à souder à partir de 10 €, mais je ne les recommanderai même pas à un débutant en raison de leur mauvaise qualité. Tout dépend évidemment de l'usage que vous en aurez. Si vous comptez en faire un usage intensif et si vous voulez qu'il vous dure longtemps, vous devrez privilégier la qualité.

Fil de soudure



◀ **Figure 8-20**
Bobine de fil de soudure

Pour raccorder électriquement des composants électroniques ou des câbles, on utilise du fil de soudure qui est chauffé à une température d'environ 185 °C pour être liquéfié, puis qui durcit en refroidissant. Cela permet, par exemple, de fixer des circuits électroniques sur une plaque d'essais afin de prévenir les courts-circuits ou les coupures qui se produisent parfois avec un câblage volant. Il existe de nombreux types de fil de soudure qui contiennent une âme décapante pour éliminer les couches d'oxyde pouvant apparaître sur les surfaces à souder. Un agent de brasage peut aussi être acheté et utilisé séparément.

Pompe à dessouder

Enlever un composant préalablement soudé pour une raison quelconque (par exemple composant défectueux ou inadapté) pose problème. Cela peut marcher à la rigueur avec un composant à deux pattes. On chauffe la première soudure avec le fer à souder jusqu'à ce qu'elle soit liquide, puis on tire le composant vers le haut. On procède ensuite de la même manière pour la seconde soudure. Mais lorsque se présente le cas d'un transistor à trois pattes, les choses se compliquent. Quand on chauffe la première soudure, les deux autres

connexions le maintiennent en place et l'extraction est quasi impossible...



Attention !

Si vous chauffez un composant électronique trop longtemps, il risque de surchauffer et donc de s'abîmer. Les semi-conducteurs surtout sont très sensibles à la chaleur !

C'est là que la pompe à dessolder entre en jeu.

Figure 8-21 ►
Pompe à dessolder



Elle ressemble à une seringue, à ceci près que son extrémité antérieure présente une ouverture plus ou moins grande au lieu d'une aiguille. À l'autre bout se trouve un poussoir permettant de faire coulisser le piston qui comprime le ressort dans la pompe. En fin de course le piston se bloque. Si on appuie alors sur le petit bouton, le piston revient brusquement dans sa position initiale, produisant une brève dépression à la pointe de la pompe qui aspire la soudure préalablement liquéfiée et dégage plus ou moins l'endroit soudé. Il faut avoir le coup de main pour actionner la pompe, chauffer la soudure et appuyer sur le déclencheur au bon moment. Le mieux est de vous entraîner sur une vieille carte avec des composants dont vous n'avez plus besoin ou qui sont hors d'usage.

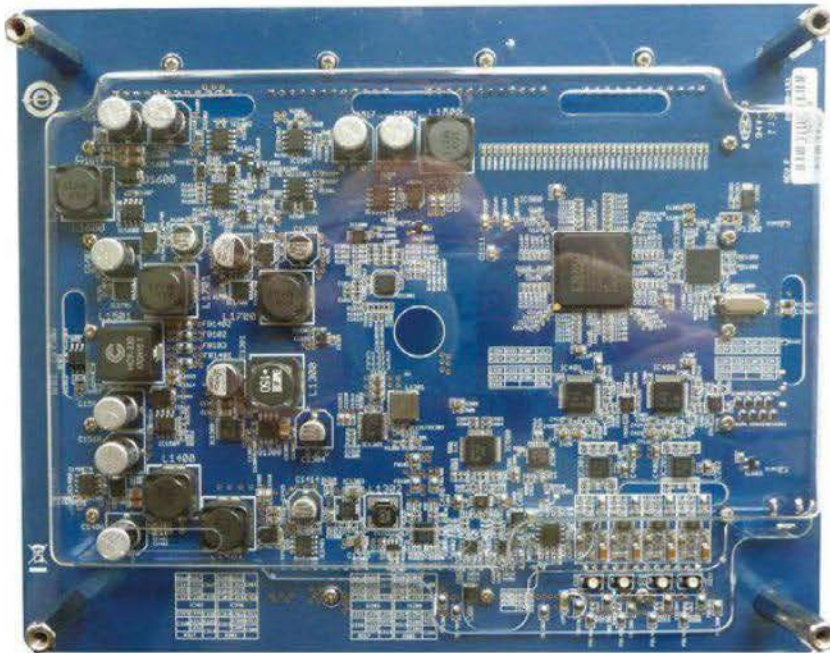
EEBoard

J'aimerais profiter de cette occasion pour vous présenter une plateforme tout-en-un qui réunit plusieurs instruments de mesure électroniques. À première vue, elle ressemble à une plaque d'essais tout ce qu'il y a de plus ordinaire, ce qu'elle est aussi. Elle appartient à la catégorie *Nice to have!*.

Il s'agit de l'EEBoard, abréviation d'Electronics Explorer Board. Elle est fabriquée par la société américaine Digilent. Qu'a-t-elle donc de si spécial ? Pour le comprendre, il faut la retourner pour examiner sa face inférieure (voir figure 8-23).



◀ **Figure 8-22**
Vue de dessus de l'EEBoard



◀ **Figure 8-23**
Vue de dessous de l'EEBoard

Cette plate-forme est donc beaucoup plus complexe qu'il n'y paraît à première vue, car de nombreux composants sont réunis sur une même

carte. Avec l'EEBoard, vous disposez des appareils électroniques suivants :

Oscilloscope numérique 4 voies

- 40 Méch/s (Mega-Samples)
- Convertisseur analogique-numérique 10 bits avec une mémoire de 16 Ko
- Plusieurs modes de déclenchement
- AC/DC avec une tension d'entrée de ± 20 V
- Transformation de Fourier rapide (FFT – *Fast Fourier Transform*)
- Zoom
- Exportation des données dans différents formats de fichiers

Analyseur logique 32 voies

- Signal d'entrée jusqu'à 5 V
- 100 Méch/s (Mega-Samples)
- Mémoire tampon jusqu'à 16 Kéch par entrée
- Horloge interne/externe et trigger
- Exportation des données dans différents formats de fichiers

Générateur de signaux arbitraires 2 voies

- Formes d'ondes standards et personnalisables
- 40 Méch/s (Mega-Samples)
- Convertisseur 14 bits avec une mémoire tampon de 32 Ko
- Bande passante 4 MHz
- Amplitude de 10 V
- Modulation AM/FM

Alimentations et voltmètres

- Tensions fixes 5 V et 3,3 V jusqu'à 2 A
- Tension/Courant programmables de -9 V à $+9$ V et jusqu'à 1,5 A
- 4 voltmètres (impédance d'entrée 1,2 M Ω)
- 2 références de tension ± 10 V

Générateur de signaux numériques 32 voies

- Mémoire tampon jusqu'à 2 Kéch par entrée
- Éditeur de pattern personnalisable

Gestion d'entrées/sorties

- Boutons-poussoirs ;
- Interrupteurs ;
- LED
- Afficheur 7 segments
- Potentiomètre linéaire
- Barres de progression

Reliée à votre PC via un port USB, l'EEBoard s'utilise avec un logiciel fourni qui est compatible Windows. Sur la figure 8-24, vous pouvez visualiser la fonction d'oscilloscope en mode 2 voies qui présente une sinusoïde et la réaction d'une diode.



◀ **Figure 8-24**
Oscilloscope de l'EEBoard

Une présentation détaillée en français du fonctionnement de l'EEBoard est disponible sur le lien <http://www.lextronic.fr/P22880-plate-forme-electronics-explorer.html>.

Les bases de la programmation

Dans le chapitre 3, nous avons déjà un peu abordé la programmation. Je vous ai montré un premier programme, appelé sketch dans la terminologie Arduino, et vous ai donné quelques informations générales sur le langage de programmation C/C++. Mais je ne vous ai pas expliqué ce que programmer signifie. Nous savons qu'il faut un appareil – PC, Mac ou microcontrôleur comme notre carte Arduino – disposant d'une interface pour pouvoir communiquer avec nous. Matériel et logiciel sont dépendants l'un de l'autre et ne peuvent fonctionner qu'ensemble. Seuls les programmes insufflent une certaine forme de vie au matériel et lui permettent de réaliser ce que le programmeur – vous, donc – peut imaginer.

Qu'est-ce qu'un programme ou sketch ?

En programmation, on rencontre généralement deux éléments constitutifs.

Élément de programme n° 1 : l'algorithme

Le sketch est censé accomplir seul une certaine tâche. Pour cela, un algorithme, composé d'un ensemble d'étapes élémentaires nécessaires à cet accomplissement, est créé. Un algorithme est donc une règle de calcul traitée à la manière d'une recette de cuisine.

Imaginez que vous vouliez construire un boîtier en bois pour y loger votre carte Arduino, pour que cela soit un peu plus beau, plus ordonné

et plaise également à vos amis. N'achetez pas de bois sans avoir préparé un plan répondant par exemple aux questions suivantes.

- Quelles sont les dimensions de la caisse ?
- De quelle couleur doit-elle être ?
- Où des ouvertures doivent-elles être pratiquées pour poser par exemple des interrupteurs ou des lampes ?

Après avoir réuni le matériel, passez à la fabrication proprement dite, en procédant étape par étape dans un certain ordre :

- choix des plaques de bois ;
- mise à dimension des plaques de bois ;
- ponçage des bords avec du papier de verre ;
- perçage de certaines plaques de bois, appelées à recevoir des ports ;
- vissage des plaques de bois entre elles ;
- mise en peinture du boîtier ;
- insertion de la carte Arduino et câblage des interrupteurs ou des lampes.

Telles sont les étapes par lesquelles il faut passer pour arriver à vos fins. Il en va de même pour l'algorithme.

Élément de programme n° 2 : les données

Vous avez très certainement soigneusement noté les dimensions sur le plan, de manière à pouvoir les consulter pendant la construction. Il faut faire en sorte que tout coïncide à la fin. Ces dimensions sont comparables aux données d'un sketch. L'algorithme utilise des valeurs temporaires qui l'aident dans son travail de prise en charge des différentes étapes. Il utilise pour cela une technique qui lui permet de stocker des valeurs et de les rappeler par la suite. Les données sont en effet sauvegardées dans des variables et disponibles à tout moment dans la mémoire. Vous en saurez bientôt plus.

Que signifie traitement des données ?

On entend par traitement des données l'utilisation d'un algorithme qui se sert de données en entrée pour en obtenir d'autres, modifie celles-ci par différents calculs et produit des résultats en sortie.

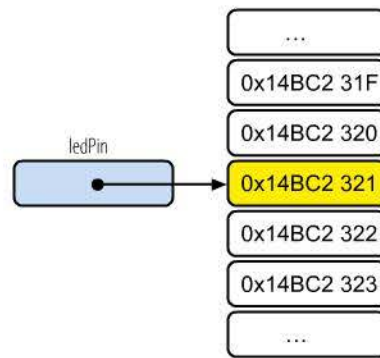


◀ **Figure 9-1**
Le traitement des données

Qu'est-ce qu'une variable ?

Nous avons déjà vu que des données étaient sauvegardées dans des variables. Ces dernières jouent un rôle central dans la programmation et sont utilisées dans le traitement des données pour stocker des informations de tous types. Une variable occupe dans la mémoire une certaine place qu'elle garde libre. L'ordinateur ou le microcontrôleur gère cependant cette mémoire (de travail) selon ses propres méthodes. Tout ceci se fait au moyen de désignations codées que tout un chacun a certainement du mal à retenir. C'est pour cette raison que vous pouvez doter les variables de noms évocateurs qui renvoient en interne aux adresses de mémoire concernées.

Figure 9-2 ►
Variable pointant sur une zone
de la mémoire de travail



Dans cette figure, la variable nommée `ledPin` pointe sur une adresse de début dans la mémoire de travail. Elle peut également être considérée comme une sorte de référence renvoyant à quelque chose de particulier. Dans le chapitre 3, je vous ai présenté un bref sketch contenant entre autres la ligne de code suivante.

```
int ledPin = 13; //Variable déclarée + initialisée avec broche 13
```

Il s'agit ici de l'utilisation d'une variable nommée `ledPin`, à laquelle la valeur numérique 13 a été attribuée. Plus loin dans le sketch, cette variable est évaluée et continue d'être utilisée.



Juste une question : que veut dire `int` devant le nom de la variable ?

Ah oui ! Le terme `int` est l'abréviation du mot *integer*. Il s'agit d'un type de donnée utilisé dans le traitement des données pour caractériser des nombres entiers, ce qui nous amène au point suivant.

Les types de données

Il existe différents types de données. Le microcontrôleur gère ses sketches et ses données dans sa mémoire. Cette mémoire est une zone structurée qui est gérée par des adresses et qui enregistre ou restitue des informations, lesquelles sont stockées sous la forme de 1 et de 0. L'unité de mémoire logique la plus petite est le bit, qui ne peut lui-même stocker que les deux états 1 ou 0. Voyez-le comme une sorte de commutateur électronique pour allumer ou éteindre. Du fait qu'avec un bit seuls deux états peuvent être représentés, plusieurs bits s'avèrent judicieux et nécessaires pour le stockage des données.

8 bits font 1 octet et permettent de stocker $2^8 = 256$ états différents. S'agissant d'un système binaire, la base 2 est utilisée. Avec 8 bits, on peut donc couvrir un domaine de valeurs compris entre 0 et 255. Le

système décimal que nous connaissons a pour base le nombre 10. Voici ce que donnent les poids des différentes positions.

Puissances	10^3	10^2	10^1	10^0
Valeurs	1000	100	10	1
Combinaison de bits	4	7	1	2

◀ **Figure 9-3**

Le système décimal et les poids des 4 premières positions

Vous pouvez bien sûr lire la valeur directement, mais ce n'est pas facile pour une personne qui n'a pas l'habitude du système binaire et doit additionner les différentes positions. On obtient dans ce cas :

$$2 \cdot 10^0 + 1 \cdot 10^1 + 7 \cdot 10^2 + 4 \cdot 10^3 = 4\,712$$

J'ai posé l'addition en commençant par la position de valeur la plus faible et en poursuivant dans l'ordre croissant vers la valeur la plus forte. Mais revenons au type de donnée byte. Le graphique de la figure 9-4 montre les 8 bits d'un octet représentant une certaine valeur décimale.

Puissances	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valeurs	128	64	32	16	8	4	2	1
Combinaison de bits	1	0	0	1	1	1	0	1

◀ **Figure 9-4**

Les 8 bits d'un octet avec leurs valeurs

Chaque position a une valeur particulière. La conversion en nombre décimal résulte également de l'addition des différents poids.

$$1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 + 0 \cdot 2^6 + 1 \cdot 2^7 = 157$$

On obtient ici le nombre 157. Ces 8 bits occupent naturellement un certain espace dans la mémoire, espace qui est nécessaire pour stocker un nombre compris entre 0 et 255. Cela se veut amplement suffisant pour de petites opérations de calcul et c'est pourquoi le type de donnée byte a été créé avec ledit domaine de valeurs.

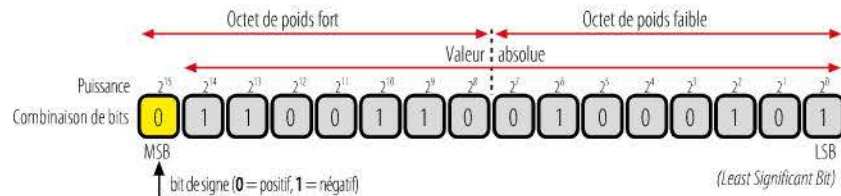
Si en revanche nous voulons travailler avec des valeurs supérieures à 255, nous atteignons les limites du possible. Le type de donnée directement supérieur a donc été créé pour calculer des valeurs plus élevées. Il a pour nom *int*, ce qui est comme nous l'avons dit déjà une forme abrégée du mot *integer*. Deux octets ont tout simplement été associés pour avoir un domaine de valeurs plus étendu à disposition.



Laissez-moi réfléchir : cela ferait $2^{16} = 65\,536$ combinaisons de bits, donc un domaine de valeurs compris entre 0 et 65 535, n'est-ce pas ?

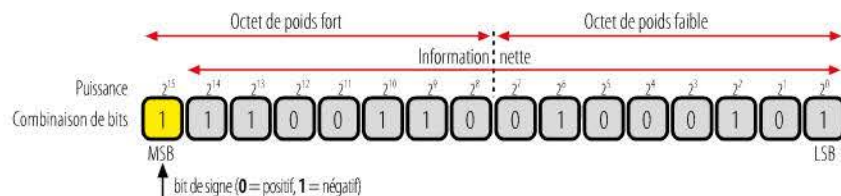
Presque, Ardu ! C'est vrai pour les 65 536 combinaisons de bits mais le domaine de valeurs n'est pas tout à fait celui que vous avez donné. Vous avez oublié qu'il existe des nombres non seulement positifs mais aussi négatifs qui doivent être également pris en compte dans ce type de donnée `int`. Pour cela, si un type de donnée est prévu à la fois pour des nombres positifs et négatifs, un bit spécial est utilisé pour stocker une information de signe, quasiment un drapeau. Ce drapeau est en principe le bit avec le plus fort poids ou **MSB** (*Most Significant Bit*). Il est naturellement dans l'ordre des choses que, pour représenter la valeur particulière, un bit de moins soit disponible pour le stockage. Voyons cela à l'aide de deux exemples. Voici tout d'abord un nombre positif, reconnaissable au fait que le bit de signe a pour valeur 0.

Figure 9-5 ▶
Les 16 bits du type de donnée `int`
(nombre positif)



La combinaison de bits représentée correspond à la valeur décimale +26 181. La même combinaison de bits avec un bit de signe 1 est donc la suivante.

Figure 9-6 ▶
Les 16 bits du type de donnée `int`
(nombre négatif)



Oui, c'est simple. Cette combinaison de bits donne la valeur - 26 181. J'ai compris !



Et vous êtes tombé dans le piège. Votre réponse est fausse. La dernière combinaison de bits n'est pas égale à l'opposé de la valeur +26 181. Un test pourrait le démontrer. Pour trouver la valeur absolue d'une valeur binaire négative, deux étapes sont nécessaires :

- inversion de tous les bits (1 devient 0 et 0 devient 1) ;
- ajout de la valeur 1.



Pour aller plus loin

L'inversion de tous les bits, appelée complément à un, est une opération portant sur des nombres binaires. Si la valeur 1 doit encore être ajoutée pour finir, on appelle l'ensemble du processus complément à deux.

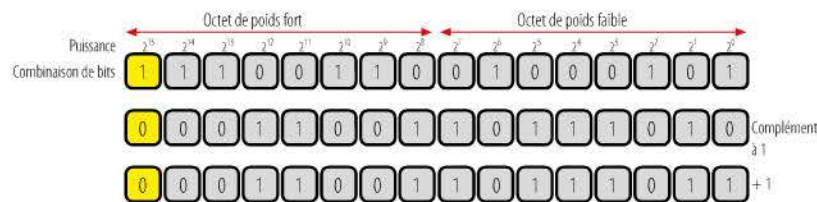
Les règles suivantes s'appliquent pour l'addition des nombres binaires.

A	B	A + B	Report
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

◀ **Tableau 9-1**

Addition d'une position binaire

L'addition des différentes positions est semblable au calcul dans le système décimal que nous connaissons. Voyons cette procédure de plus près pour une autre combinaison binaire.



◀ **Figure 9-7**

Détermination d'un nombre décimal négatif

La combinaison de bits de la dernière ligne donne la valeur décimale +6,587. Il s'agit de la valeur décimale négative de la combinaison de bits de la première ligne. On peut dire d'une autre manière que :

$$1110011001000101 = -6,587$$

Le domaine de valeurs du type de donnée `int` s'étend de -32 768 à +32 767 et s'avère donc bien plus vaste et plus souple que le type de donnée `byte`.

J'ai bien compris pour les nombres négatifs mais je ne vois pas trop l'utilité du complément à deux. Pourquoi nous donnons-nous tout ce mal ? Convertir un nombre positif en nombre négatif devrait suffire si le bit de signe est passé de 0 à 1.

Il est vrai qu'à première vue cela semble inutile. Mais il existe une raison cachée que je vais vous dévoiler. Prenons pour faire simple un nombre de 8 bits qui ne relève pas du type de donnée `byte` précédemment décrit puisque celui-ci ne comportait pas de signe.



Figure 9-8 ►
Nombre binaire positif

Puissances	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Valeur	128	64	32	16	8	4	2	1	
Combinaison de bits	0	1	0	1	0	1	1	0	$= 86_{10}$

Ce nombre positif (le MSB est de 0) représente une valeur décimale de 86. Si vous devez manipuler plusieurs systèmes de numération, il est conseillé de placer en indice la base correspondante après chaque valeur. Supposons maintenant que nous voulions en faire un nombre négatif en passant uniquement le bit de signe de 0 à 1. Le bon résultat serait donc celui de la figure 9-9.

Figure 9-9 ►
Nombre binaire négatif

Puissances	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Valeur	128	64	32	16	8	4	2	1	
Combinaison de bits	1	1	0	1	0	1	1	0	$= 86_{10}$

Si arrivés à ce stade, nous en venons à conclure que le résultat est la valeur négative du nombre positif précédemment indiqué, tout irait bien au début. Cela ne porterait pas à conséquence. Mais dans le traitement des données, les valeurs ne sont pas seulement stockées et affichées. Elles servent aussi à calculer et c'est là que le bât blesse. Supposons que vous vouliez ajouter une valeur – disons +1 –, ce qui voudrait dire que le résultat serait plus élevé de 1 que la valeur initiale. Voyons ce que cela donnerait côté bits.

Figure 9-10 ►
Résultat de l'addition
(pas tout à fait juste !)

Puissances	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Valeur	128	64	32	16	8	4	2	1	
Combinaison de bits	1	1	0	1	0	1	1	0	$= 86_{10}$
Addition +1	0	0	0	0	0	0	0	1	+1
Résultat	1	1	0	1	0	1	1	1	$= 87_{10}$

Rien ne vous frappe ? Malgré l'ajout d'une valeur positive, le résultat a diminué de 1. $-86 + 1 = -87$? On n'est pas près d'en sortir ! Appliquons maintenant à la valeur initiale le complément à un mentionné plus haut. J'en profiterai pour passer directement au problème suivant, qui se pose pour un nombre très particulier : on obtient de toute valeur son pendant négatif en plaçant un signe négatif devant, chiffre 0 inclus. Mais 0 et -0 sont absolument identiques et il n'y a aucune différence arithmétique.

Puissances	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Valeur	128	64	32	16	8	4	2	1	
	0	0	0	0	0	0	0	0	= 0
	1	1	1	1	1	1	1	1	= 0

◀ **Figure 9-11**
Deux combinaisons de bits
pour la même valeur

Impensable car prêtant à confusion ! Aussi la valeur 1 a-t-elle été ajoutée, ce qui a donné en somme le complément à deux. Ce procédé vient justement de vous être décrit pour un nombre de 16 bits. Le tableau 9-2 donne quelques exemples de valeurs positives et négatives.

Valeurs positives	Valeurs négatives
$1_{10} = 00000001_2$	$-1_{10} = 11111111_2$
$64_{10} = 01000000_2$	$-64_{10} = 11000000_2$
$80_{10} = 01010000_2$	$-80_{10} = 10110000_2$

◀ **Tableau 9-2**
Valeurs positives et valeurs
négatives correspondantes

Je voudrais ici vous poser une question : si vous tombiez par hasard sur la combinaison de bits 10110010_2 dans la mémoire et si quelqu'un vous demandait à quelle valeur décimale elle correspond, sauriez-vous lui donner la bonne réponse ?

Bien sûr, pourquoi ? J'ai toutes les informations qu'il faut pour réussir une conversion.

Non, vous n'avez pas encore toutes les informations ! Je vous ai certes montré la combinaison de bits mais pas encore le type de donnée de base. Il existe pourtant d'autres types de données de 16 bits qui peuvent aussi être utilisés puisque le langage de programmation de base est C ou C++. C'est le cas par exemple du type de donnée *unsigned int*, qui est aussi un type de nombre entier mais qui ne peut stocker – comme son nom *unsigned* (non signée) l'indique – que des valeurs positives. J'ajouterai qu'il existe encore d'autres différences, qui peuvent varier de compilateur à compilateur, certains utilisant 2 et d'autres 4 octets pour gérer le type de donnée. Il s'agit dans notre cas de 2 octets, autrement dit le domaine de valeurs est compris entre 0 et +65 535.





Holà, holà ! Je ne vois pourquoi on se donne tout ce mal. Pourquoi ne crée-t-on pas un seul type de donnée suffisamment grand pour contenir toutes les valeurs possibles et imaginables ? Il n'y aurait pas tous ces problèmes avec les différents domaines de valeurs que personne ne peut retenir.

Ainsi selon vous, il faudrait créer un type de donnée dont la taille serait par exemple de 16 octets et avec laquelle on serait paré contre toute éventualité. Voyons plus loin : la mémoire est limitée dans un microcontrôleur et ne peut être agrandie aussi facilement que dans un ordinateur. Chaque petite variable, qui ne doit compter que de 0 à 255, mobiliserait une place 16 fois supérieure. Si vous faites la somme de toutes les variables nécessaires à votre sketch, vous arrivez vite à saturer la mémoire disponible. C'est pour cette raison que divers types de données ont été créés avec différentes tailles ou différents domaines de valeurs, pour avoir un choix approprié à l'utilisation qu'on veut en faire. Avec le temps, vous aurez les principaux domaines de valeurs en tête et n'aurez plus besoin de consulter un tableau. À propos de tableau, voici pour commencer la liste des principaux types de données auxquels vous allez être confronté.

Tableau 9-3 ►
Types de données et domaines
de valeurs correspondants

Type de données	Domaine de valeurs	Taille de donnée	Exemple
byte	0 à 255	1 octet	byte value = 42;
unsigned int	0 à 65 535	2 octets	unsigned int seconds = 46547;
int	-32 768 à 32 767	2 octets	int ticks = -325;
long	-2 ³¹ à 2 ³¹ -1	4 octets	long value = -3457819;
float	-3,4 × 10 ³⁸ à 3,4 × 10 ³⁸	4 octets	float reading = 27.5679;
double	voir float	4 octets	double reading = 27.5679;
boolean	true ou false	1 octet	boolean flag = true ;
char	-128 à 127	1 octet	char mw = 'm';
String	variable	variable	String name = "Erik Bartman";
Array	variable	variable	int pinArray [] = {2,3,4,5};

Presque tous ces types de données seront utilisés dans ce livre, aussi je ne m'appesantirai pas davantage sur le sujet.

J'ai encore une question : que se passe-t-il si, par exemple, j'ai une variable du type de donnée byte et que le maximum de 255 est dépassé quand j'incrémente sa valeur ? Une erreur survient-il ?



On s'y attendrait en effet, mais pourtant rien ne se produit et le contenu de la variable recommence à compter à partir de 0. Il faut néanmoins le savoir, car cela peut aboutir à des erreurs dans l'exécution du sketch, qui ne sont pas faciles à localiser. Par conséquent, gardez bien en tête le domaine de valeurs correspondant au type de données choisi.

Qu'est-ce qu'une boucle ?

Dans un sketch, l'exécution de nombreuses étapes récurrentes peut être nécessaire pour calculer des données. Si ces étapes sont similaires, il n'est pas nécessaire de les écrire en grand nombre les unes en dessous des autres et de les faire exécuter de manière séquentielle. Une structure de programme spéciale a été créée à cet effet, permettant d'exécuter une partie de programme composée d'une ou plusieurs étapes, maintes fois répétées.

Cette structure s'appelle une boucle. Voyons comment elle se construit. On en distingue deux types :

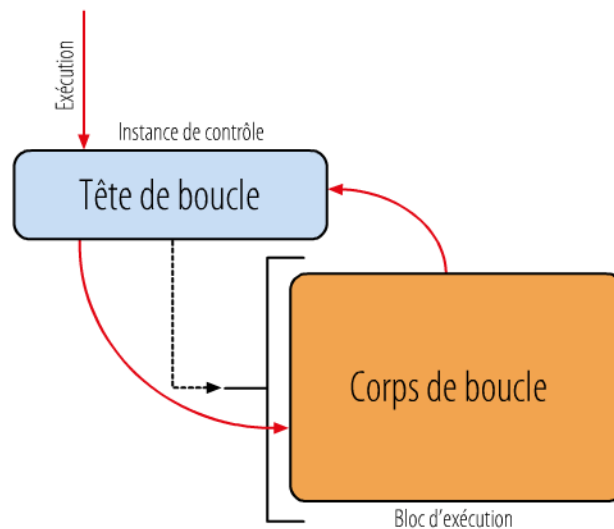
- les boucles avec condition de sortie en tête ;
- les boucles avec condition de sortie en queue.

Ces deux familles de boucles possèdent une instance qui contrôle si la boucle doit être itérée, et de quelle manière elle doit l'être. À cette instance est rattachée une seule instruction ou tout un bloc d'instructions (corps de boucle).

Les boucles avec condition de sortie en tête

Dans les boucles avec condition de sortie en tête, l'instance de contrôle se situe au début de la boucle. L'exécution de la première itération de la boucle dépend de l'évaluation de la condition. Pour les boucles dont la condition se situe en tête, les instructions placées au sein de la boucle peuvent ne pas être exécutées si la condition est fausse dès l'entrée dans la boucle.

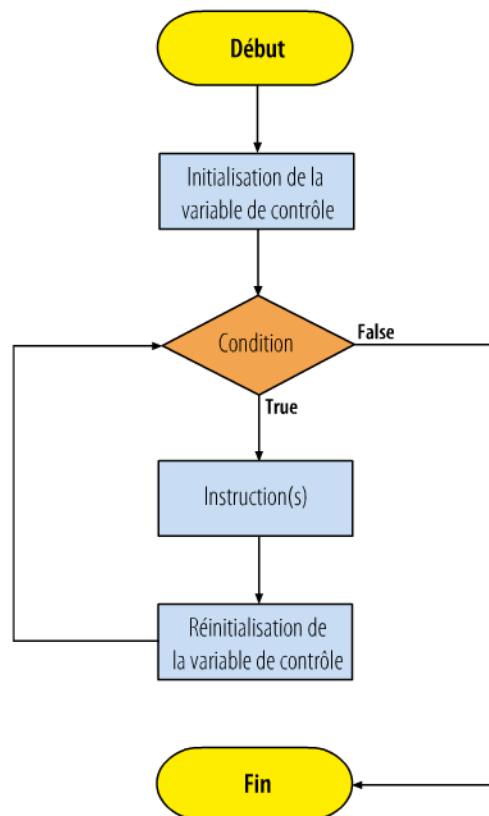
Figure 9-12 ►
Principe d'une boucle
avec condition de sortie en tête



Il existe différents types de boucles avec condition de sortie en tête, qui s'utilisent selon le contexte.

La boucle for

Figure 9-13 ►
Organigramme d'une boucle for



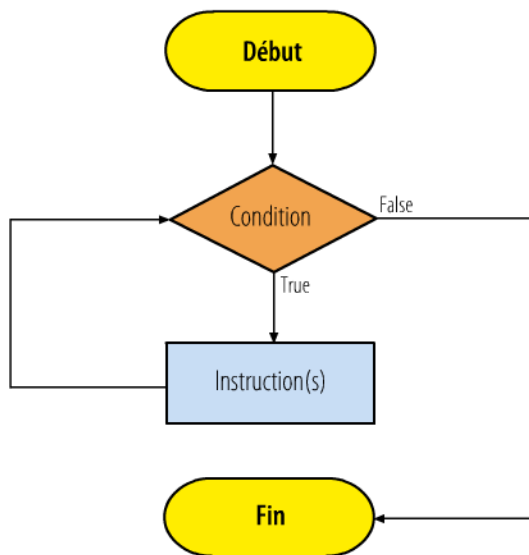
La boucle `for` est toujours utilisée quand on connaît déjà le nombre d'itérations de la boucle avant même de l'appeler. Jetons un coup d'œil à l'organigramme servant à restituer graphiquement le déroulement du programme (voir figure 9-13).

Une variable appelée variable de contrôle est utilisée dans la boucle. Dans la condition, elle est soumise à une évaluation qui décide si et combien de fois la boucle doit être itérée. La valeur de cette variable est généralement modifiée au début de la boucle à chaque nouvelle itération, si bien que la condition d'interruption doit être remplie au bout d'un moment dans la mesure où vous n'avez pas fait de faute de raisonnement. Voici un court exemple, sur lequel nous reviendrons bientôt.

```
for(int i = 0; i < 7; i++)  
    pinMode(ledPin[i], OUTPUT);
```

La boucle `while`

La boucle `while` est utilisée quand on ne sait qu'au moment de l'exécution si et combien de fois la boucle doit être itérée. Si, pendant une itération de boucle, une entrée du microcontrôleur par exemple est constamment interrogée ou surveillée alors qu'une action doit être exécutée à une certaine valeur, cette boucle vous rendra bien service. Voyons maintenant à quoi ressemble l'organigramme.



◀ **Figure 9-14**
Organigramme d'une boucle
`while`

Sur cette boucle, la condition d'interruption se trouve également en tête. En revanche, la variable indiquée dans la condition n'est pas modifiée. Elle doit l'être dans le corps de boucle. En cas d'oubli, on a

affaire à une boucle sans fin d'où on ne sort pas tant que le sketch s'exécute. Voici encore un court exemple.

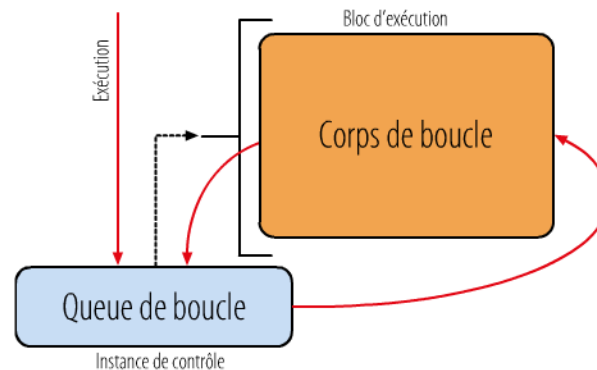
```
while(i > 1) //Instance de contrôle
{
    Serial.println(i);
    i = i - 1;
}
```

Quand vous travaillez avec des valeurs ou des variables de type *float*, par exemple, dans l'instance de contrôle, il peut être très risqué d'attendre un résultat précis à cause de l'imprécision de *float*. La condition d'interruption risque de ne jamais être remplie et le sketch se retrouvera prisonnier d'une boucle sans fin. Au lieu de l'opérateur `==` pour évaluer l'égalité, utilisez de préférence les opérateurs `<=` ou `>=`.

Les boucles avec condition de sortie en queue

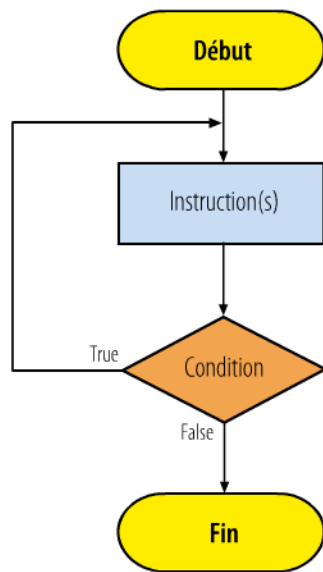
Venons-en maintenant à la boucle avec condition de sortie en queue. On l'appelle ainsi parce que l'instance de contrôle est hébergée en fin de boucle.

Figure 9-15 ►
Principe d'une boucle
avec condition de sortie en queue



On l'appelle boucle `do...while`. La condition étant évaluée seulement à la fin de la boucle, on peut déjà en déduire qu'elle sera exécutée au moins une fois.

Cette boucle est peu utilisée, mais je tenais tout de même à la citer par souci d'exhaustivité. La syntaxe ressemble à celle de la boucle `while`, mais vous remarquerez que l'instance de contrôle est placée en fin de boucle.



◀ **Figure 9-16**
Organigramme d'une boucle
do...while

```

do
{
  Serial.println(i);
  i = i - 1;
} while(i > 1); //Instance de contrôle
  
```

Qu'est-ce qu'une structure de contrôle ?

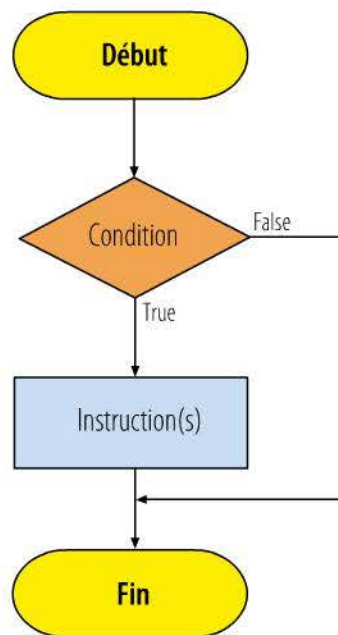
Les instructions ont déjà été abordées au chapitre 2. Elles informent le microcontrôleur de ce qu'il doit faire. Mais un sketch se compose généralement de toute une série d'instructions qui doivent être traitées de manière séquentielle. La carte Arduino présente un certain nombre d'entrées ou de sorties auxquelles vous pouvez raccorder divers composants électriques ou électroniques. Si le microcontrôleur doit réagir à certaines influences extérieures, vous branchez par exemple un capteur sur une entrée. La forme de capteur la plus simple est un interrupteur ou un bouton-poussoir. Quand le contact est fermé, une LED doit s'allumer. Le sketch doit donc être en mesure de prendre une décision. Si l'interrupteur est fermé, la LED est alimentée (LED allumée) ; si l'interrupteur est ouvert, la LED est privée d'alimentation (LED éteinte).



Jetons d'abord un coup d'œil sur l'organigramme qui nous montre comment le déroulement de l'exécution du sketch arrive à certains endroits, où le processus n'est plus linéaire. Le sketch, quand une structure de contrôle est atteinte, se trouve à la croisée des chemins et doit examiner par où il doit continuer.

Une condition lui sert de base décisionnelle, condition qu'il lui faut analyser. Techniquement, une instruction `if` est utilisée. Il s'agit d'une décision `si-alors`.

Figure 9-17 ►
Organigramme d'une structure
de contrôle `if`



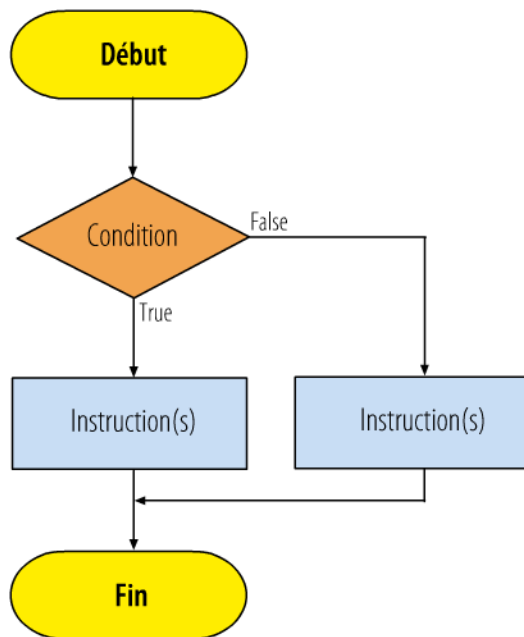
Si la condition a été vérifiée, il s'ensuit l'exécution d'une, voire de plusieurs instructions. Voici encore un court exemple.

```
if(buttonState == HIGH)
    digitalWrite(ledPin, HIGH);
```

Si plusieurs instructions sont exécutées dans une instruction `if`, vous devez constituer un bloc d'instructions avec les paires d'accolades. Il sera alors exécuté en tant qu'unité d'instructions complète.

```
if(buttonState == HIGH)
{
    digitalWrite(ledPin, HIGH);
    Serial.println("HIGH-Level reached.");
}
```

Il existe aussi une forme élargie de la structure de contrôle `if`. Il s'agit d'une décision si-alors-sinon qui résulte d'une instruction `if-else`. La figure 9-18 présente l'organigramme.



◀ **Figure 9-18**
Organigramme d'une structure
de contrôle `if-else`

L'exemple de code suivant vous montre la syntaxe de l'instruction `if-else`.

```
if(buttonState == HIGH)
    digitalWrite(ledPin, HIGH);
else
    digitalWrite(ledPin, LOW);
```

Commentez votre code !

Quand des êtres humains veulent communiquer entre eux, par exemple pour exprimer des sentiments ou transmettre des informations, ils utilisent un langage sous sa forme orale ou écrite. C'est seulement par ce moyen qu'ils peuvent apprendre quelque chose et accroître leur savoir ou leur compréhension. Quand on traite un problème en programmeur et qu'on encode, il est assurément utile de prendre çà et là quelques notes. Il nous vient parfois une fulgurance ou une idée géniale et quelques jours plus tard, nous avons du mal – et c'est souvent le cas pour moi – à nous souvenir exactement de notre raisonnement. Qu'est-ce que j'ai bien pu programmer et pourquoi m'y suis-je pris ainsi et pas autrement ? Chaque programmeur peut bien sûr avoir sa propre stratégie de prise de notes : bloc-notes, dos de prospectus publicitaires, documents Word, etc.

Toutes ces méthodes présentent cependant des inconvénients non négligeables.

- Où ai-je bien pu mettre mes notes ?
- S'agit de la dernière version actualisée ?
- Je n'arrive pas à me relire !
- Comment mettre ces notes à la disposition d'un ami qui s'intéresse également à ma programmation ?

Le problème vient de la séparation du code de programmation et des notes, qui ne forment alors plus un tout. Si les notes sont perdues, vous aurez vraiment beaucoup de mal à tout reconstruire. Imaginez maintenant votre ami, qui n'a absolument aucune idée de ce que vous vouliez faire avec votre code. Mais il existe une autre solution : vous pouvez laisser des remarques et consignes dans le code, et ce à l'endroit précis où elles sont pertinentes. Vous avez ainsi sous la main toutes les informations qui vous sont nécessaires.

Commentaires sur une ligne

Voici un exemple pris dans un programme.

```
int ledPinRedCar = 7;    //La broche 7 commande la LED rouge
int ledPinYellowCar = 6; //La broche 6 commande la LED jaune
int ledPinGreenCar = 5;  //La broche 5 commande la LED verte
...
```

Ici, des variables sont déclarées et initialisées avec une valeur. Des noms évocateurs ont certes été choisis, mais je trouve utile de laisser

encore quelques brèves remarques complémentaires. Derrière la ligne d'instruction est ajouté un commentaire, introduit par deux barres obliques (*slash*). Pourquoi ces barres sont-elles nécessaires ? Le compilateur essaie bien entendu d'interpréter et d'exécuter tous les prétendus ordres qui lui sont donnés. Prenons par exemple le premier commentaire :

La broche 7 commande la LED rouge

Il s'agit des divers éléments d'une phrase que le compilateur ne comprend pas puisque ce ne sont pas des instructions. Cette notation entraînerait une erreur pendant la compilation du code. Mais les deux `//` masquent cette ligne et informent le compilateur que tout ce qui suit les deux traits obliques ne le concerne pas et qu'il peut sans crainte ne pas en tenir compte.

C'est une sorte de pense-bête pour le programmeur, qui n'est même pas fichu de retenir la moindre chose pendant une période prolongée (> 10 minutes). Patience avec lui ! Ce mode d'écriture permet d'introduire un commentaire d'une seule ligne.

Commentaires sur plusieurs lignes

Si, en revanche, vous voulez écrire plusieurs lignes, comme une description succincte de votre sketch, placer deux barres obliques devant chaque ligne peut s'avérer fastidieux. Aussi la variante suivante à plusieurs lignes a-t-elle été créée.

```
/*  
  Auteur: Erik Bartmann  
  Domaine: Contrôle des feux de signalisation  
  Date: 31.10.2013  
*/
```

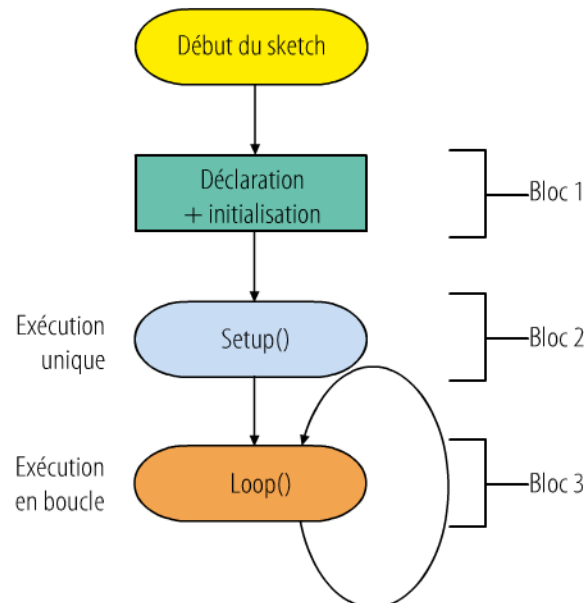
Ce commentaire présente une combinaison de signes introductifs `/*` et une combinaison de signes conclusifs `*/`. Tout ce qui se trouve entre les deux tags (un tag étant une marque utilisée pour identifier des données qui ont une importance particulière) est considéré comme étant du commentaire et ignoré par le compilateur. Tous les commentaires s'affichent en gris dans l'environnement de développement Arduino, de manière à être immédiatement reconnaissables.

Structure d'un sketch Arduino

Si vous voulez écrire un sketch pour votre carte Arduino, vous devez impérativement tenir compte de certaines choses. Pour être exécutable, le sketch doit présenter deux structures de programme techniques qui font partie de la même catégorie. Ce sont les *fonctions*, qui servent quasiment de cadre au sketch. Voyons d'abord ce qu'est au juste une fonction. Nous avons vu jusqu'ici les différentes instructions, qui existent en tant que telles et n'ont pas forcément un rapport entre elles. Or, il est possible de réunir plusieurs instructions en une unité logique et de donner à cette structure un nom évocateur. Vous faites alors appel au nom de la fonction comme pour une instruction ordinaire et toutes les instructions qu'elle contient sont exécutées en bloc.

Arrêtons-nous auparavant sur la manière dont un tel sketch peut se dérouler. Supposons que vous vouliez faire une promenade et emporter certaines choses avec vous. Vous mettez tout dans un sac à dos, puis vous partez. En route, vous fouillez sans cesse dans votre sac pour prendre des forces ou pour vérifier sur la carte que vous êtes encore sur le bon chemin. Au sens figuré, cela se déroule exactement de la même manière dans un sketch. Au début du sketch se produit l'exécution unique d'une certaine action par exemple pour initialiser des variables qui devront être utilisées plus tard. Ensuite, certaines instructions sont alors exécutées en boucle, gardant ainsi le sketch vivant. Jetons un coup d'œil à la structure du sketch, dans laquelle les domaines fondamentaux sont divisés en trois blocs.

Figure 9-19 ►
Structure de sketch fondamentale



Ces blocs sont les suivants.

Bloc 1 : déclaration et initialisation

Dans ce bloc, peuvent être intégrées – si nécessaire – des bibliothèques externes au moyen de l’instruction `#include`. Je vous dirai plus tard comment cela fonctionne. C’est également ici que sont déclarées les variables globales qui sont accessibles et utilisables partout dans le sketch.

Cette déclaration permet de définir le type de donnée de la variable. Lors de l’initialisation en revanche, la variable reçoit une valeur.

Bloc 2 : la fonction setup

La fonction `setup` permet la plupart du temps de configurer les différentes broches du microcontrôleur. On définit ainsi quelles broches doivent servir d’entrées et de sorties. Certaines sont reliées à des capteurs, des boutons-poussoir ou des résistances sensibles à la température, qui amènent des signaux de l’extérieur à une entrée correspondante. D’autres conduisent encore des signaux à des sorties pour commander par exemple un moteur, un servo ou une diode électroluminescente.

Bloc 3 : la fonction loop

La fonction `loop` permet de former une boucle sans fin contenant la logique, au moyen de laquelle des capteurs sont interrogés ou des actionneurs commandés en permanence. Chacune des deux fonctions forme ensemble avec son nom un bloc d’exécution identifié par des accolades `{}`. Celles-ci servent d’éléments délimiteurs pour savoir où la définition de la fonction commence et où elle s’arrête. Le mieux est de vous montrer les corps de fonction d’un sketch exécutable. Il ne se passe peut-être pas grand-chose mais il s’agit d’un véritable sketch.

```
void setup(){
    // Une ou plusieurs instructions
    // ...
}

void loop(){
    // Une ou plusieurs instructions
    // ...
}
```



Ces fonctions doivent-elles porter ces noms ou puis-je leur donner le nom que je veux ? Et que signifie le mot `void` qui se trouve devant chaque fonction ?

Non, les fonctions doivent porter ces noms à la lettre près : elles sont recherchées en début de sketch car elles servent de points de départ et garantissent un début bien défini.

Comment le compilateur saurait-il sinon laquelle ne doit être exécutée qu'une fois et laquelle doit l'être en boucle ? Ces noms sont donc absolument nécessaires. Passons à votre deuxième question concernant la signification du mot `void`. Il s'agit d'un type de donnée indiquant simplement que la fonction ne renvoie aucune valeur à l'appelant. `void` peut être traduit par place vide ou trou. Il signifie non pas 0 mais rien. La structure générale d'une fonction est présentée à la figure 9-20.

Figure 9-20 ▶
Structure générale d'une fonction

```
Type de donnée retournée Nom (paramètre)  
{  
    // Une ou plusieurs instructions  
    return valeur ; //Pas pour le type void  
}
```

Quand une fonction possède le type de donnée `void`, aucune instruction `return` renvoyant une valeur n'est admise. En revanche, si elle présente un autre type de donnée, elle peut renvoyer à l'appelant une valeur qui doit toutefois correspondre à celle du type de donnée indiqué. Vous pouvez même transmettre à une fonction des valeurs qu'elle doit utiliser ensuite. Ces valeurs sont placées à l'intérieur des parenthèses et sont transmises aux variables correspondantes. Dans une définition de fonction, les variables sont appelées paramètres. Les paires de parenthèses doivent être là même s'il n'y a aucune valeur à transmettre, comme pour `setup()` et `loop()`. Elles restent tout simplement vides. Vous apprendrez à composer vos propres fonctions lorsque vous réaliserez les montages de ce livre.

Combien de temps dure un sketch sur la carte ?

Tout sketch mis sur la carte Arduino dans le microcontrôleur est aussitôt exécuté. Il dure tant que la carte est alimentée en courant et que vous n'entrez pas un autre sketch. Si vous coupez l'alimentation,

soit au port USB soit en externe, le sketch s'arrête bien évidemment mais reprend dès que vous rebranchez la carte. Il est conservé hors tension dans la mémoire (flash) du microcontrôleur et n'a pas besoin d'être rechargé.



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- principes de programmation C++ ;
- principe Entrée Traitement Sortie.

Programmation de la carte Arduino

Ce chapitre est consacré aux interfaces de notre carte Arduino. Ce sont des canaux de communication permettant une interaction entre la carte et le monde extérieur. Les thèmes abordés sont les suivants.

- À quoi correspondent les ports numériques ?
- À quoi correspondent les ports analogiques ?
- Qu'est-ce qu'un signal MLI ?

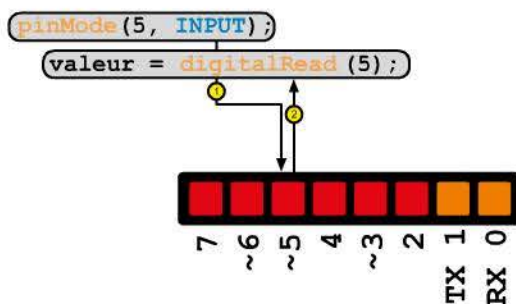
Les ports numériques

Les ports numériques de votre carte Arduino peuvent servir aussi bien d'entrées que de sorties. Mais ça ne veut pas dire pour autant que les broches 0 à 7 sont des entrées et que les broches 8 à 13 des sorties. Chacune des 14 broches numériques mises à disposition peut être configurée individuellement en entrée ou en sortie. Une instruction est utilisée à cet effet, laquelle définit le sens de circulation des données, broche par broche. L'instruction `pinMode` permet de programmer le numéro et le sens des données (INPUT ou OUTPUT) de chaque broche, en fonction de ce dont votre sketch a besoin.

Les entrées numériques

L'instruction `pinMode` est utilisée pour programmer une broche en entrée. La figure 10-1 montre les deux étapes nécessaires à la configuration et à l'interrogation d'une entrée numérique.

Figure 10-1 ►
Configuration et lecture d'une
entrée numérique sur la broche 5

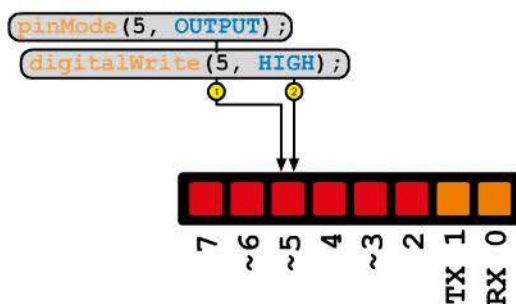


La première étape consiste à configurer la broche 5 en entrée (INPUT) avec `pinMode`, et ce uniquement au sein de la fonction mentionnée pour la première fois dans le chapitre 9. La deuxième étape permet de lire le niveau logique (HIGH ou LOW) de la broche par l'instruction `digitalRead`. Dans cet exemple, il est affecté à la variable `valeur` et peut être traité plus tard.

Les sorties numériques

L'instruction `pinMode` est, bien entendu, également utilisée pour programmer une broche numérique en sortie, mais cette fois avec `OUTPUT` comme deuxième argument. La figure 10-2 montre les deux étapes nécessaires à la configuration et à la définition d'une sortie numérique.

Figure 10-2 ►
Configuration et définition d'une
sortie numérique sur la broche 5



La première étape consiste à configurer la broche 5 en sortie (OUTPUT) avec `pinMode`, et ce uniquement au sein de la fonction `setup` mentionnée pour la première fois dans le chapitre 5. La deuxième étape permet de définir le niveau logique (HIGH ou LOW) de la broche avec l'instruction `digitalWrite`.



Pour aller plus loin

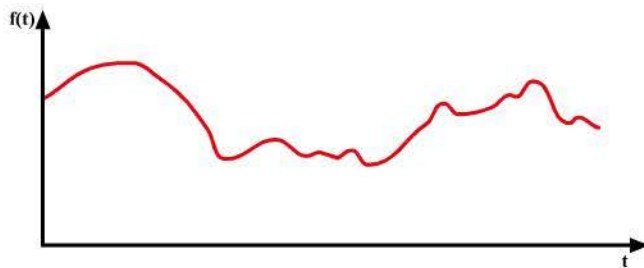
Les deux broches numériques 0 (RX = réception) et 1 (TX = émission) ont une fonction spéciale et sont utilisées par l'interface série. Dans les graphiques, elles sont différenciées par une autre couleur.

Pour éviter les problèmes, je vous déconseille d'utiliser ces deux broches. Elles m'ont déjà posé quelques problèmes, aussi fais-je toujours en sorte de ne pas les utiliser dans mes circuits. Si, faute de ports disponibles, vous envisagez d'en faire quand même usage, il vous faudra débrancher brièvement ces deux connexions lors du chargement du sketch dans le microcontrôleur. Au risque de voir survenir des problèmes empêchant le chargement.

Les ports analogiques

Les entrées analogiques

Les signaux analogiques sont aussi étrangers au microcontrôleur que ne l'est l'intelligence, bien que certains scientifiques prétendent pouvoir donner une forme de personnalité à leurs machines. Voyons maintenant les signaux analogiques de plus près.



◀ **Figure 10-3**
Signal analogique

On voit que leur évolution en fonction du temps présente différentes valeurs comprises entre un minimum et un maximum et qu'il n'y a pas d'échelonnement net comme pour les signaux numériques, où seule une alternative entre niveau HIGH et LOW est possible. Pour faire traiter un signal analogique par un microcontrôleur, il faut une entrée analogique et rien de plus.

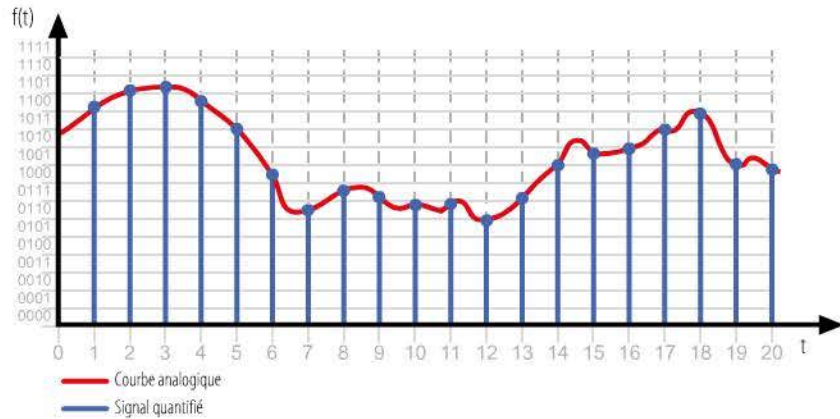
Vous venez de dire que les signaux analogiques peuvent varier de manière continue entre deux limites. Autrement dit, on peut rencontrer n'importe quelle valeur arbitraire lors de mesures à l'entrée analogique, n'est-ce pas ?



En théorie oui, sauf que nous avons affaire ici à un microcontrôleur, qui ne peut traiter que des signaux numériques. Je m'explique : les signaux analogiques sont traités et mémorisés par un circuit spécial appelé convertisseur analogique/numérique (ou convertisseur A/D).

Théoriquement, un signal analogique se compose de variations infiniment petites qui se produisent dans la courbe d'évolution temporelle. Mais comment ces valeurs peuvent-elles être détectées par un microprocesseur ? Voyons maintenant cette courbe de plus près.

Figure 10-4 ►
Numérisation d'un signal
analogique (résolution de 4 bits)



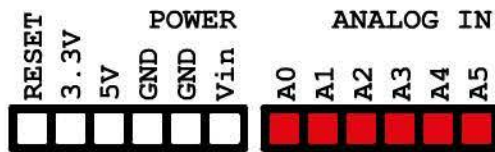
Ce graphique montre le signal analogique représenté par la courbe rouge. La mesure est effectuée à des instants déterminés (numérotés sur l'axe des temps) et à chacun de ces points correspond une valeur binaire (sur l'axe des ordonnées). On peut voir cependant sur le graphique qu'une même valeur numérique peut être affectée à des valeurs analogiques différentes. Il y a même des zones dans lesquelles se retrouvent plusieurs points de mesure analogique. Voyez par exemple les valeurs analogiques correspondant aux points 8 et 9. Elles sont différentes et font pourtant partie de la zone numérique 0111.



Mais pourquoi est-ce ainsi ? Aucune différence n'est faite entre les deux valeurs qui seraient, par conséquent, égales.

Non, Ardur. Comme j'ai pu vous l'indiquer précédemment, un signal analogique présentait des gradations infiniment diverses dans sa courbe d'évolution en fonction du temps. Rien que pour les valeurs minimales, il faudrait une valeur binaire tout aussi infiniment « longue » pour pouvoir reproduire toutes les valeurs. C'est non seulement impossible, mais aussi inutile. Notre microcontrôleur dispose de 10 bits pour la résolution d'un signal analogique. C'est du reste également ce qui a été retenu pour la caractéristique du convertisseur A/D : résolution de 10 bits.

Que signifient ces 10 bits ? Ils permettent d'interpréter $2^{10} = 1\,024$ nombres binaires différents. Si nous prenons comme référence la tension de +5 V disponible sur la carte Arduino (notamment pour l'alimentation), une entrée analogique – il y en a 6 en tout de A0 à A5 – peut traiter des valeurs comprises entre 0 et +5 V.



◀ **Figure 10-5**

Ports analogiques A0 à A5 sur le connecteur à 6 broches (côté droit)

Sur chaque canal, le signal d'entrée est converti en interne en un nombre binaire par un convertisseur A/D et, un canal présentant 1 024 valeurs puisque la résolution est de 10 bits, l'unité de mesure la plus petite – appelée aussi quantum – peut être calculée comme suit :

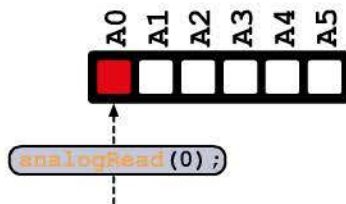
$$\text{Quantum} = \frac{\text{Tension de référence}}{\text{Nombre de valeurs}} = \frac{+5\text{ V}}{1\,024} = 4,88\text{ mV} \approx 4,9\text{ mV}$$



Attention !

Si vous délivrez à une entrée analogique une tension supérieure à +5 V, le microcontrôleur risque d'être complètement détruit ou pour le moins grillé au niveau de ce canal. Veillez à toujours vérifier sous quelles tensions vous travaillez. C'est important si vous utilisez des sources d'alimentation externes telles qu'une pile de 9 V ou un bloc d'alimentation séparé. Pour nos exemples analogiques, je n'utilise toutefois que l'alimentation électrique de la carte, soit +5 V.

L'instruction `analogRead` (numéro de broche) permet de lire la valeur d'un signal présent sur une entrée analogique. Le graphique montre l'interrogation de la broche analogique portant le numéro 0.



◀ **Figure 10-6**

Quelle valeur a la broche analogique 0 ?

Le tableau 10-1 montre quelques exemples de valeurs mesurées à une entrée analogique et sa tension existante réelle.

Tableau 10-1 ►
Valeurs analogiques mesurées
et tensions d'entrée réelles
correspondantes

Valeur analogique mesurée	Valeur correspondante
0	0 V
1	4,9 mV
2	9,8 mV
...	...
1023	5 V

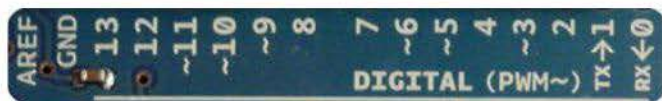
Les sorties analogiques

Vous avez pu voir par vous-même que le microcontrôleur ne disposait d'aucune sortie analogique. Est-ce un défaut ? Ou ont-elles simplement été oubliées ? Sûrement pas ! Je peux d'ores et déjà dire ici qu'aucune sortie n'est dédiée et donc absolument nécessaire aux signaux analogiques. Pourtant, si des sorties analogiques sont à disposition comme le prétend la description de la carte Arduino, il doit bien y avoir une autre solution. Mais laquelle ? Et revoilà la MLI. Trois lettres qui ne vous évoquent rien pour l'instant.

Que signifie MLI ?

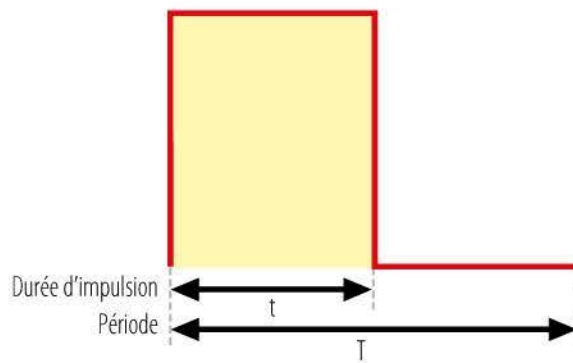
Vous n'allez peut-être pas le croire, mais les prétendues sorties analogiques manquantes se trouvent sur certaines broches numériques. Si vous regardez de plus près la carte Arduino, vous verrez qu'un signe bizarre en zigzag se trouve sur certaines de ces broches. Ce signe est appelé *tilde* et signale les sorties analogiques.

Figure 10-7 ►
Broches analogiques côté
numérique



MLI est le pendant français de PWM (*Pulse-Width-Modulation*), autrement dit modulation de largeur d'impulsion. Vous n'êtes pas plus avancé pour autant...

Un signal MLI est un signal de fréquence et d'amplitude de tension constantes. La seule chose qui varie est le rapport cyclique. Vous allez bientôt savoir de quoi il s'agit.

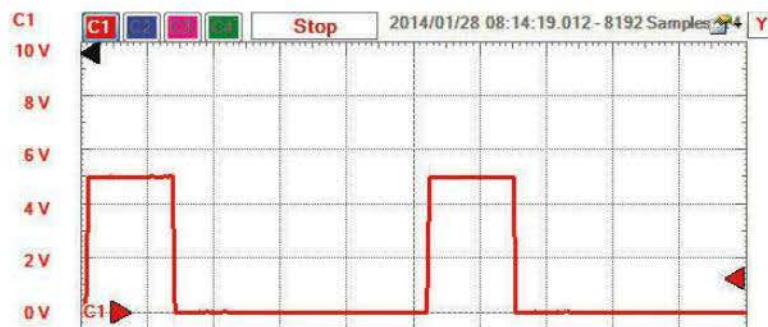


◀ **Figure 10-8**
Durée d'impulsion et durée de période dans la courbe d'évolution temporelle (t varie ; T est constant)

Plus l'impulsion est large, plus l'utilisateur reçoit d'énergie.

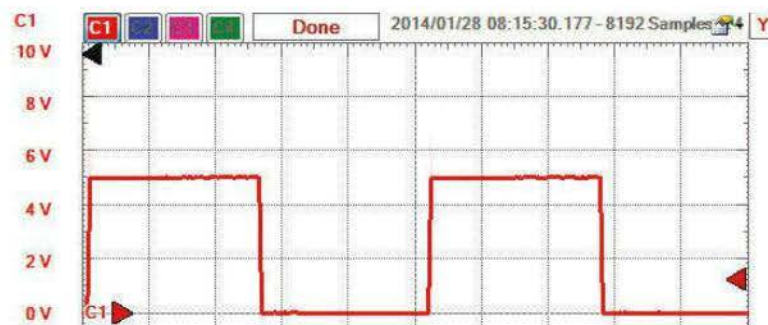
$$\text{Rapport cyclique} = \frac{t}{T}$$

On peut dire aussi que la surface de l'impulsion sert de mesure à l'énergie envoyée. Voici maintenant quatre oscillogrammes pour des rapports cycliques de 25 %, 50 %, 75 % et 100 %.



◀ **Figure 10-9**
Rapport cyclique de 25 %

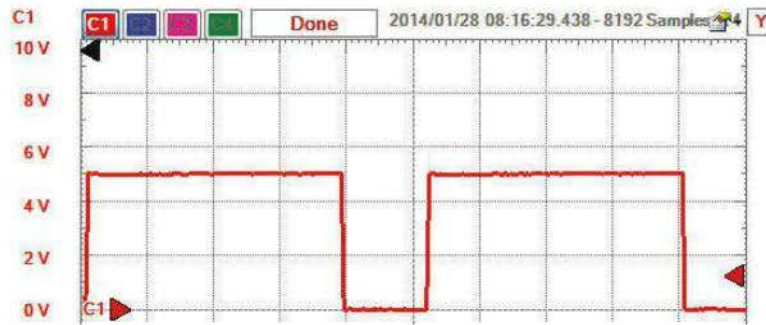
Si une LED était raccordée à la sortie MLI, elle recevrait seulement un quart de l'énergie possible pour éclairer. Elle serait coupée avant même de commencer à éclairer correctement. Autrement dit, elle n'éclairerait que très faiblement.



◀ **Figure 10-10**
Rapport cyclique de 50 %

Dans le cas d'un rapport cyclique de 50 %, le temps de marche est égal au temps d'arrêt au cours d'une période. La LED éclaire bien mieux qu'à 25 % parce qu'elle reçoit plus d'énergie par unité de temps.

Figure 10-11 ▶
Rapport cyclique de 75 %



Dans le cas d'un rapport cyclique de 75 %, le rapport entre temps de marche et temps d'arrêt bascule nettement en faveur du temps de marche, autrement dit la LED éclaire encore mieux qu'à 25 % et 50 %.

◀ **Figure 10-12**
Rapport cyclique proche de 100 %

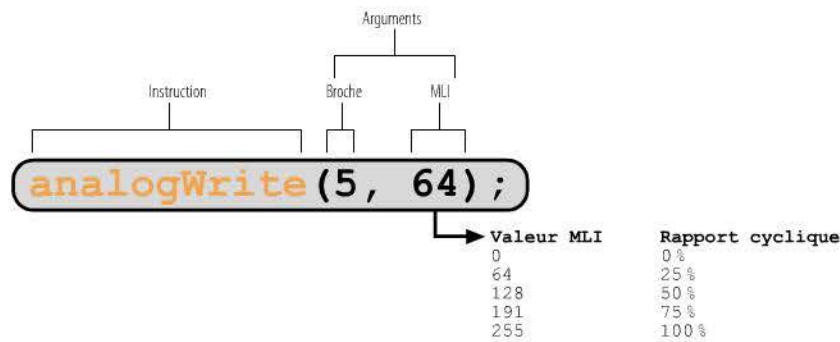


Dans le cas d'un rapport cyclique proche de 100 %, la LED est constamment allumée et semble très lumineuse. Une chose est sûre quand on utilise des sorties analogiques : le nombre de sorties analogiques utilisées diminue d'autant le nombre de broches numériques à disposition.



Et si je veux maintenant utiliser la sortie analogique présente sur une broche numérique, comment fais-je pour lui parler ? Dois-je utiliser l'instruction `digitalWrite` avec peut-être un autre paramètre ?

Bonne question, Arduos ! Pour une sortie analogique au moyen d'un signal MLI, on utilise l'instruction `analogWrite`, qui nécessite de spécifier le numéro de la broche et la valeur à convertir en signal MLI. Nous en reparlerons bien sûr en temps voulu. Cette valeur peut varier entre 0 et 255. La figure 10-13 présente un échantillon des plus marquantes.



◀ **Figure 10-13**
Instruction `analogWrite`
avec quelques exemples de valeurs

Dans cet exemple, la broche MLI n° 5 est sollicitée et la valeur à convertir est 64. Cette valeur permet d'obtenir un signal de rapport cyclique de 25 %. La formule suivante permet de calculer la valeur d'entrée nécessaire pour obtenir le rapport cyclique souhaité.

$$\text{Valeur MLI} = \frac{\text{Rapport cyclique souhaité}}{100 \%} \cdot 255$$

►► Pour aller plus loin

Pour utiliser une sortie analogique, il faut impérativement programmer la broche nécessaire en tant que sortie (OUTPUT) au moyen de l'instruction `pinMode`.

Je tiens évidemment à dire avant de clore le sujet qu'un signal MLI n'est pas un véritable signal analogique au sens propre du terme. S'il vous en faut un à tout prix, vous pouvez brancher un circuit RC à la sortie, lequel constitue un filtre passe-bas permettant de lisser la tension de sortie. Vous trouverez d'autres informations sur Internet à ce sujet.

►► Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- signal MLI ;
- modulation de largeur d'impulsions.

L'interface série

L'interface série, qui – comme je l'ai dit déjà – utilise le port USB, est une autre possibilité d'entrer en contact avec la carte Arduino. Nul besoin ici encore d'un programme externe de terminal pour commu-

niquer car on peut passer par le moniteur série. Ce type de mise en relation avec votre carte Arduino peut être utile dans de nombreux cas :

- pour entrer des données pendant l'exécution d'un sketch ;
- pour sortir des données pendant l'exécution d'un sketch ;
- pour imprimer certaines informations pendant l'exécution d'un sketch pour rechercher des erreurs.

La carte Arduino ne disposant pas d'un appareil de saisie dans sa version standard, l'interface série permet justement de saisir des données à la main en passant par le clavier pour intervenir le cas échéant sur le déroulement du sketch. Mais nous verrons aussi que cette interface peut très bien être utilisée pour créer une base de communication commune à différents programmes ou langages de programmation. Vous apprendrez des possibilités intéressantes : comment contrôler votre carte Arduino par exemple avec une application C#, envoyer des données à celle-ci pour une présentation graphique préparée... Vous pouvez sinon tirer habilement parti du langage de programmation Processing pour qu'il serve de frontal graphique à la carte Arduino. Si votre sketch ne fonctionne pas comme vous l'auriez voulu, utilisez le moniteur série de l'interface comme fenêtre pour imprimer le contenu de variables. Vous pouvez ainsi déterminer et éliminer des erreurs dans le code source. Vous saurez plus tard comment on doit s'y prendre. Le mot technique employé pour cela est *debugging* (ou débogage).

Partie II

Les montages

Le premier sketch

Au sommaire :

- la déclaration et initialisation d'une variable ;
- la programmation d'une broche numérique en sortie (OUTPUT) ;
- l'instruction `pinMode()` ;
- l'instruction `digitalWrite()` ;
- l'instruction `delay()` ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Le phare "Hello World"

Eh bien Arduus, les choses deviennent maintenant sérieuses, sérieuses certes mais pas vraiment difficiles car nous allons avancer doucement. La plupart des manuels sur les langages de programmation commencent par la présentation d'un programme appelé Hello World. Ce programme est généralement le premier que le débutant est amené à découvrir. Il donne un premier aperçu de la syntaxe du nouveau langage de programmation et imprime le texte "Hello World" dans une fenêtre. Le nouveau langage de programmation se présente ainsi à vous et au reste du monde, et semble dire "Eh je suis là ! Utilisez-moi".

Nous avons à présent un petit problème car notre Arduino n'a pas d'écran à l'origine, et donc pas de console de visualisation pour nous

informer. Alors que faire ? Si aucune communication sous une forme écrite n'est possible, alors peut-être l'est-elle avec des signaux optiques ou acoustiques ? Nous optons pour la variante optique car une diode électroluminescente – également appelée LED – se branche sans problème à l'une des sorties numériques et attirera à coup sûr l'attention. J'ai été moi-même très étonné quand ça a marché du premier coup.

Composants nécessaires

L'exemple étant très simple, seules une LED et une résistance série sont nécessaires.



Dans le chapitre 3 sur les principes de base Arduino, je vous ai dit que la carte comportait entre autres quelques LED, dont l'une était reliée directement à la broche numérique 13 et possédait sa propre résistance série. Ceci dit, aucun composant externe ne devrait avoir besoin d'être branché sur la carte.



Cette LED se trouve à gauche près du logo Arduino.



Pour aller plus loin

Quand vous reliez pour la première fois une carte Arduino flambant neuve à votre ordinateur, cette LED incorporée s'allume pendant une seconde. Un premier sketch incluant cette fonctionnalité de base a donc été chargé à l'usine une fois la carte assemblée.

Code du sketch

Le code du sketch est le suivant pour ce premier exemple :

```
int ledPin = 13; //Déclarer + initialiser broche numérique 13
                //comme sortie

void setup(){
  pinMode(ledPin, OUTPUT); //Broche numérique 13 comme sortie
}

void loop(){
  digitalWrite(ledPin, HIGH); //LED au niveau HIGH (5V)
  delay(1000);                //Attendre une seconde
  digitalWrite(ledPin, LOW);  //LED au niveau LOW (0V)
  delay(1000);                //Attendre une seconde
}
```

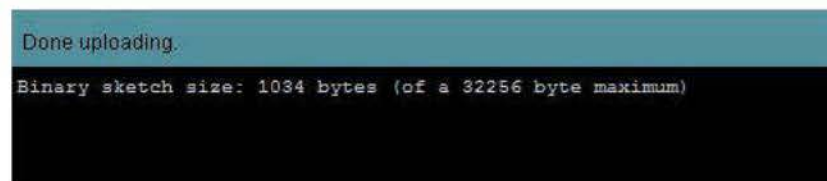
Vous pouvez vérifier le code si vous l'avez transmis dans l'éditeur. Le compilateur essaye alors de le traduire. Le tableau 1-1 indique les deux étapes fondamentales.

Icône	Fonction
	Commencer la vérification par compilation.
	Commencer la transmission au microcontrôleur une fois la compilation terminée.

◀ **Tableau 1-1**

Étapes pour compiler et transmettre

À la fin, un message indique que la transmission a réussi. La mémoire nécessaire y est donnée en octets, de même que la mémoire totale à disposition.



◀ **Figure 1-1**

Message d'état et affichage des informations sur la mémoire

J'ai une question avant que vous ne poursuiviez. Comment doit-on considérer les mots OUTPUT, HIGH ou LOW ? S'agit-il de mots-clés ? L'IDE les fait en tout cas apparaître en couleurs.

Bien vu Arduus ! Il me faut ici développer un peu. Si vous initialisez des variables avec des valeurs dont vous êtes seul, de prime abord, à connaître la signification, les autres personnes appelées à travailler avec le code en question auront certainement du mal à comprendre. Que peut bien vouloir dire par exemple le nombre 42 ? Un tel style de



programmation n'est pour moi pas très convaincant. Il se trouve que nous avons pris précisément le nombre 13 pour désigner la broche de la sortie numérique, mais nous entendons rendre le code de programme un peu plus évocateur à l'avenir. De telles valeurs suspectes survenant dans le code source sont au fait appelées *magic numbers*. Mais revenons aux mots indiqués en couleurs. Il s'agit ici de *constantes*, c'est-à-dire des indicateurs qui, tout comme les variables, ont été initialisés avec une valeur mais ne peuvent plus être modifiés. C'est pourquoi on les appelle des constantes. Ce nom en dit déjà beaucoup plus que n'importe quelle valeur ésotérique. Nous reviendrons bientôt sur les instructions `pinMode` et `digitalWrite` et j'expliquerai alors l'importance de ces constantes.

Revue de code

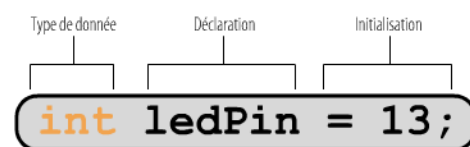
Nous déclarons et initialisons au départ une *variable globale* portant le nom de `ledPin`, dont le type exact de donnée est `int` (*int* = integer) et qui apparaît dans toutes les fonctions avec la valeur 13.

Tableau 1-2 ▶
Variable nécessaire et son objet

Variable	Objet
<code>ledPin</code>	Contient le numéro de broche pour la LED sur la broche de sortie numérique 13.

L'initialisation équivaut à une affectation de valeur avec l'opérateur d'affectation `=`. Déclaration et initialisation occupent ici une ligne. Le mode d'écriture est donc plus court que la variante à deux lignes.

Figure 1-2 ▶
Déclaration et initialisation
de variable



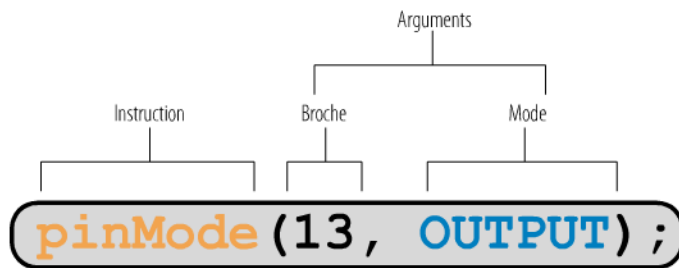
Si vous décidez d'écrire ces deux actions séparément, aucun problème mais les deux lignes ne doivent pas se suivre. Cet exemple produit une erreur :

```
int ledPin; //Déclarer la variable
ledPin = 13; //Initialiser la variable avec valeur 13 -> Erreur !!!
```

La déclaration de la variable globale `ledPin` se fait hors des fonctions `setup` et `loop`. L'initialisation se fait quant à elle dans la fonction `setup` qui n'est appelée qu'une fois. Le code du sketch correct est alors le suivant :

```
int ledPin;      //Déclarer la variable
void setup(){
  ledPin = 13; //Initialiser la variable avec valeur 13
               // ...
}
```

Vous auriez très bien pu aussi opérer sans variable et entrer la valeur 13 partout dans les instructions `pinMode` et `digitalWrite`. Seulement, il y a un inconvénient. Si vous décidez plus tard de changer de broche, vous devez revoir tout le code du sketch pour procéder à tous les changements. C'est fastidieux et surtout générateur d'erreurs. Vous risquez d'oublier un endroit quelconque à éditer et d'avoir ensuite un problème. Ce ne pas très grave dans ce court exemple mais si le sketch était plus long, vous sauriez combien ce rudiment de programmation est vraiment utile. Faisons donc bien les choses dès le début. Tout va bien jusqu'ici ? La fonction `setup` est appelée une seule fois au début pour démarrer le sketch et la broche numérique 13 est programmée en tant que sortie. Revoyons pour ce faire l'instruction `pinMode`.



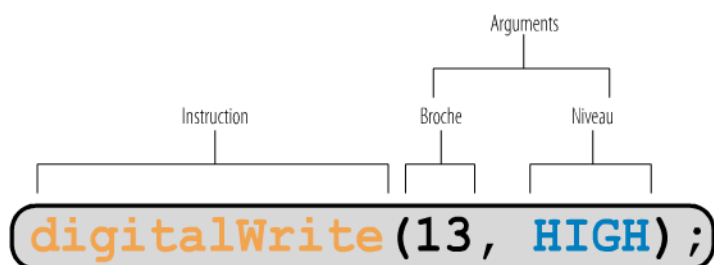
◀ **Figure 1-3**
L'instruction `pinMode`
avec ses arguments

Elle présente deux arguments, le premier pour la broche ou le port à configurer et le deuxième pour définir son comportement comme entrée ou sortie. Mettons que vous vouliez raccorder une LED et que vous ayez besoin pour cela d'une broche de sortie. L'instruction exige deux arguments numériques, le deuxième étant une constante avec une certaine valeur qui définit le mode relatif à la direction des informations. Derrière la constante `OUTPUT` se cache la valeur 1. Que dites-vous donc par conséquent de l'instruction suivante :

```
pinMode(13, 1);
```

On a du mal à comprendre ce qui se passe. La forme initiale est bien plus explicite et on sait tout de suite de quoi il s'agit. Il en va de même pour l'instruction `digitalWrite`, qui présente également deux arguments.

Figure 1-4 ►
L'instruction `digitalWrite`
avec ses arguments



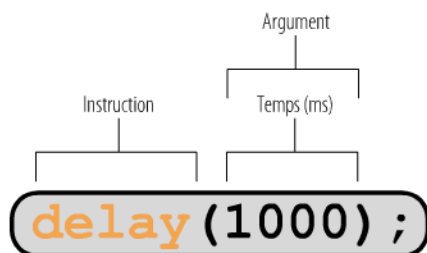
On trouve ici aussi une constante dont le nom est `HIGH`, censée servir d'argument pour un niveau `HIGH` sur la broche 13. Elle est équivalente à la valeur numérique 1. Vous trouverez les valeurs correspondantes dans le tableau 1-3.

Tableau 1-3 ►
Constantes avec valeurs
numériques correspondantes

Constante	Valeur	Explication
<code>INPUT</code>	0	Constante pour l'instruction <code>pinMode</code> (programme la broche en tant qu'entrée)
<code>OUTPUT</code>	1	Constante pour l'instruction <code>pinMode</code> (programme la broche en tant que sortie)
<code>LOW</code>	0	Constante pour l'instruction <code>digitalWrite</code> (met la broche au niveau <code>LOW</code>)
<code>HIGH</code>	1	Constante pour l'instruction <code>digitalWrite</code> (met la broche au niveau <code>HIGH</code>)

La dernière instruction utilisée, `delay`, sert à la temporisation. Elle interrompt l'exécution du sketch pour un temps correspondant à la valeur donnée qui exprime la durée en millisecondes (ms).

Figure 1-5 ►
L'instruction `delay`



La valeur 1 000 signifie une attente de 1 000 ms précisément, soit 1 seconde, avant de continuer.

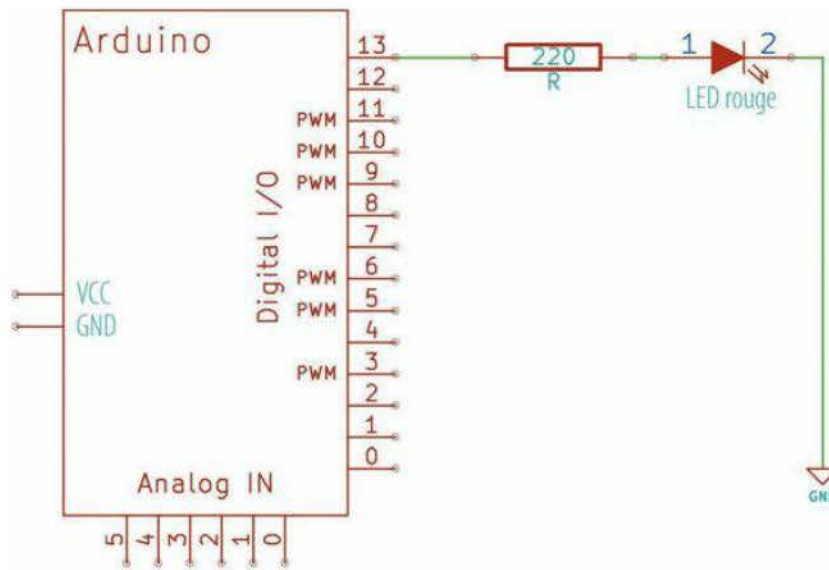
Entrons maintenant plus avant dans le sketch. La fonction `loop`, ici c'est une boucle sans fin, démarre. Voici les différentes étapes de travail.

1. Allumez la LED de la broche 13.
2. Attendez une seconde.

3. Éteignez la LED de la broche 13.
4. Attendez une seconde.
5. Revenez au point 1 puis, recommencez.

Schéma

Le schéma rend les choses plus claires.



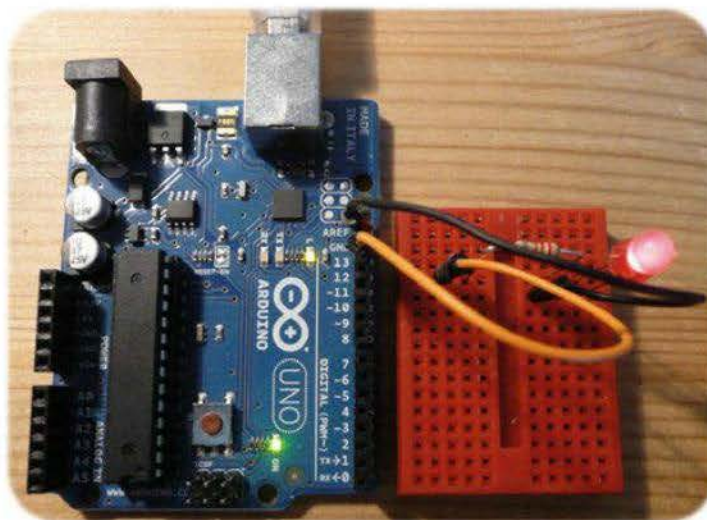
◀ **Figure 1-6**
Carte Arduino avec une LED
sur la broche 13

L'anode (ici, l'électrode 1) de la LED est reliée à la broche 13 via la résistance série, tandis que l'autre extrémité ou cathode (ici, l'électrode 2) de la LED est reliée à la masse de la carte Arduino.

Réalisation du circuit

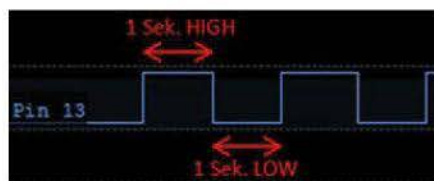
La réalisation du circuit est simple. Toutefois, veillez à ce que la polarité de la LED soit correcte, sinon vous n'obtiendrez qu'une belle LED éteinte. La LED soudée sur la carte clignotera quand même. Une LED mal polarisée n'abîmera rien mais mieux vaut bien faire les choses.

Figure 1-7 ►
LED clignotante servant de phare
pour notre premier sketch



C'est difficile à voir mais en regardant bien, on s'aperçoit que la LED « onboard » clignote en même temps que la LED reliée extérieurement. Les LED sont censées commencer à clignoter aussitôt après la transmission réussie sur la carte. Voyons maintenant de plus près le déroulement chronologique. La LED clignote toutes les deux secondes.

Figure 1-8 ►
Chronogramme



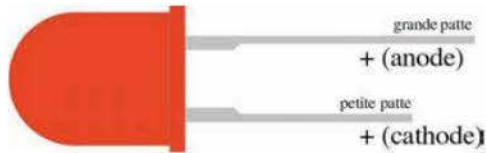
Attention !

On trouve sur Internet des schémas de circuits où une diode électroluminescente est branchée directement entre masse et broche 13. Les deux fiches femelles se trouvant l'une à côté de l'autre, côté broches numériques, il est très aisé de brancher une LED. Je vous mets expressément en garde contre cette variante car la LED est utilisée sans résistance série. Ce n'est pas tant pour la LED mais bel et bien pour votre microcontrôleur que je m'inquiète. J'ai mesuré une fois que l'intensité du courant atteignait 60 mA. Cette valeur est de 50 % au-dessus du maximum et donc assurément trop élevée. Rappelez-vous que le courant admis par une broche numérique du microcontrôleur est de 40 mA au maximum.

Problèmes courants

Si la LED ne s'allume pas, plusieurs choses peuvent en être la cause ainsi que nous l'avons déjà dit.

- La LED peut avoir été mal polarisée. Rappelez-vous les deux différentes connexions d'une LED qui sont l'anode et la cathode.

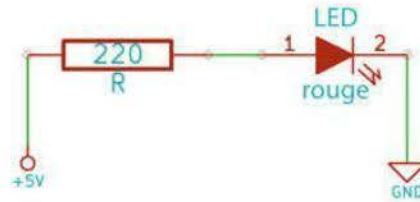


- La LED a peut-être été grillée par une surtension lors des montages précédents. Testez-la avec une résistance sur une source d'alimentation de 5 V.
- Vérifiez les fiches de la barrette de raccordement qui sont reliées à la LED ou à la résistance série. S'agit-il bien de GND et de la broche 13 ?
- Vérifiez le sketch que vous avez entré dans l'éditeur de l'IDE. Peut-être avez-vous oublié une ligne ou commis une erreur ou peut-être le sketch a-t-il mal été transmis ?
- Si la LED qui se trouve sur la carte clignote, le LED branchée doit elle aussi clignoter. Le sketch fonctionne dans ce cas correctement.

Qu'avez-vous appris ?

- Vous avez appris à déclarer et initialiser des variables globales en une ou plusieurs lignes.
- Vous avez déterminé le sens de transmission des données pour une certaine broche comme OUTPUT au moyen de l'instruction `pinMode`, si bien que vous avez pu envoyer, au moyen de l'instruction `digitalWrite`, un signal numérique (HIGH ou LOW) à la sortie où la LED est branchée.
- Vous avez créé un temps d'attente dans l'exécution du sketch au moyen de l'instruction `delay`, si bien que la LED restait allumée ou éteinte un certain temps.
- Vous savez que pour utiliser une LED, il faut une résistance série dimensionnée en conséquence. Vous trouverez ci-après un

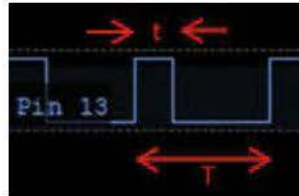
schéma de connexion d'une LED avec une résistance de 220 ohms en série.



Exercice complémentaire

Dans notre premier exercice, je vous propose de modifier le sketch de telle sorte que les temps durant lesquels la LED est allumée ou éteinte soient déterminés par deux variables, afin de pouvoir changer facilement le rapport cyclique. Ce dernier peut être défini, dans le cas d'une suite périodique d'impulsions, par le rapport entre la durée d'une impulsion et la durée de la période. Le résultat est la plupart du temps exprimé en pourcentage. Le chronogramme de la figure 1-9 montre les différentes durées pour t ou T .

Figure 1-9 ►
Chronogramme d'une impulsion



t = durée de l'impulsion

T = durée de la période

La formule pour calculer le rapport cyclique est la suivante :

$$\text{Rapport cyclique} = \frac{t}{T}$$

Programmez le sketch de telle sorte que la LED reste allumée pendant 500 ms et éteinte pendant 1 s. Le rapport cyclique est alors calculé comme suit :

$$\text{Rapport cyclique} = \frac{500 \text{ ms}}{1\,500 \text{ ms}} = 0,33$$

Ceci correspond à un rapport cyclique de 33 %. Par rapport à la durée complète de la période, la LED est allumée pendant 33 % du temps.

Interrogation d'un capteur

Au sommaire :

- la déclaration et l'initialisation de plusieurs variables ;
- la programmation des broches en tant qu'entrée (INPUT) et sortie (OUTPUT) ;
- l'instruction `digitalRead()` ;
- l'utilisation de la structure de contrôle `if-else` ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Appuyez sur le bouton

Dans cet exemple, nous entendons aller à contre-courant et ne pas envoyer, comme dans notre premier sketch, des informations depuis notre carte Arduino vers l'extérieur, mais relier un composant à une broche, interroger l'état du composant et renvoyer celui-ci à une LED raccordée. Le comportement suivant doit être ici de mise :

- bouton-poussoir non enfoncé – LED éteinte ;
- bouton-poussoir enfoncé – LED allumée.

Composants nécessaires



1 LED rouge



1 bouton-poussoir



1 résistance de 10 k Ω



1 résistance de 330 Ω



Plusieurs cavaliers flexibles de couleurs et de longueurs différentes

Code du sketch

Le code du sketch pour cet exemple est le suivant :

```
int ledPin = 13;    //LED en broche 13
int buttonPin = 8;  //Bouton-poussoir en broche 8
int buttonState;    //Variable pour enregistrer l'état du bouton

void setup(){
  pinMode(ledPin, OUTPUT); //Broche LED en tant que sortie
  pinMode(buttonPin, INPUT); //Broche bouton-poussoir en tant
                             //qu'entrée
}

void loop(){
  buttonState = digitalRead(buttonPin);
  if(buttonState == HIGH)
    digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin, LOW);
}
```

Quand vous avez transmis le code, compilez-le comme vous avez appris et envoyez-le au microcontrôleur.



Pour aller plus loin

Une broche numérique opère de manière standard comme entrée et n'a donc pas besoin d'être programmée en tant que telle au moyen de l'instruction `pinMode`. C'est cependant utile pour une meilleure vue d'ensemble. Vous pouvez malgré tout laisser tomber cette étape dans la mesure où votre mémoire est juste et où chaque octet compte.

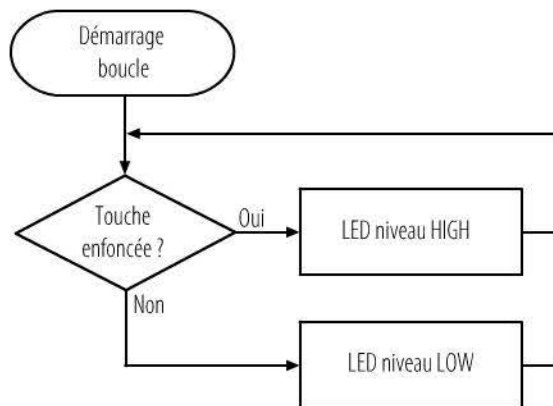
Revue de code

Dans cet exemple, on voit d'emblée qu'on a affaire à plusieurs variables qui doivent être déclarées et initialisées dès le début. Passons-le en revue.

Variable	Objet
<code>ledPin</code>	Contient le numéro de broche pour la LED sur la broche de sortie numérique 13.
<code>buttonPin</code>	Contient le numéro de broche pour le bouton-poussoir sur la broche d'entrée numérique 8.
<code>buttonState</code>	Sert à enregistrer l'état du bouton-poussoir pour une exploitation ultérieure.

◀ **Tableau 2-1**

Variables nécessaires et leur objet



◀ **Figure 2-1**

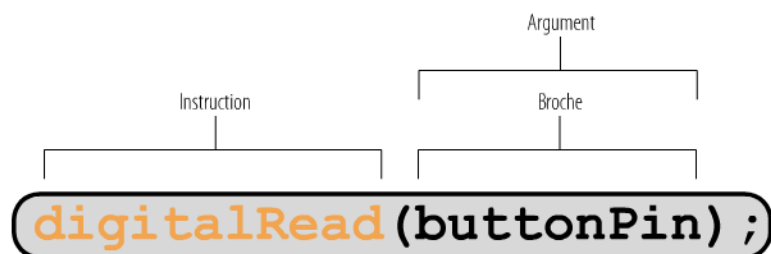
Organigramme pour commander la LED

L'organigramme se lit très facilement. Lorsque l'exécution du sketch arrive à la boucle sans fin `loop`, l'état de la broche du bouton-poussoir est continuellement interrogé et consigné dans la variable `buttonState`. Voici la ligne de code correspondante.

```
buttonState = digitalRead(buttonPin);
```

La variable est donc réinitialisée en permanence, et son comportement varie selon l'état du bouton-poussoir. La syntaxe de l'instruction `digitalRead` est indiquée dans la figure 2-2.

Figure 2-2 ►
L'instruction `digitalRead`



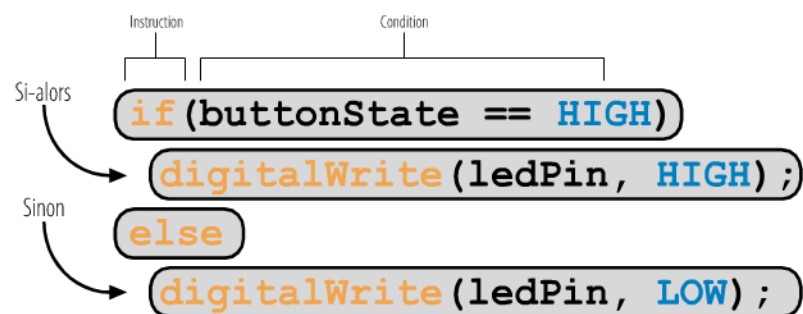
Cette fonction n'est pas seulement appelée, mais nous renvoie également une valeur de retour que nous pouvons mettre à profit. La valeur est transmise à la variable `buttonState` au moyen de l'opérateur d'affectation `=`. Les valeurs possibles peuvent être soit `HIGH`, soit `LOW` et sont ici aussi – comme vous l'avez déjà appris – des constantes qui améliorent la lisibilité. Vous savez maintenant quelles valeurs se cachent derrière, grâce au montage n° 1. L'évaluation est effectuée à la suite de l'interrogation par une structure de contrôle `if-else` (si-alors-sinon).

```

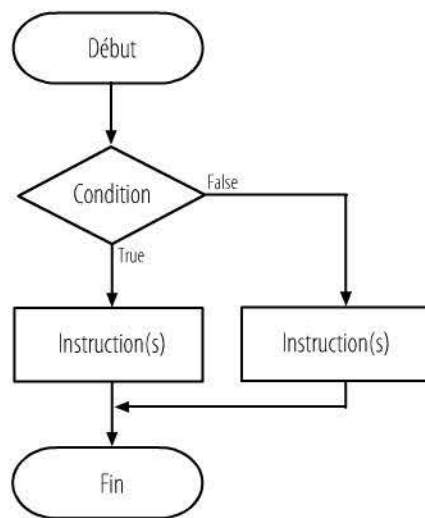
if(buttonState == HIGH)
    digitalWrite(ledPin, HIGH);
else
    digitalWrite(ledPin, LOW);
  
```

L'instruction `if` évalue la condition entre parenthèses, laquelle peut être librement traduite comme suit : « Le contenu de la variable `buttonState` est-il égal à `HIGH` ? Si oui, exécuter la ligne d'instruction qui vient aussitôt après l'instruction `if`. Si non, poursuivre avec l'instruction qui vient après le mot-clé `else`. »

Figure 2-3 ►
Interrogation par structure
de contrôle `if-else`



La figure 2-4 montre le mode de travail de cette structure de contrôle.



◀ **Figure 2-4**
Organigramme pour structure
de contrôle if-else

Il existe une variante plus simple de la structure de contrôle if, dans laquelle la branche else est absente. Nous y reviendrons plus tard. Vous voyez donc que le déroulement d'un programme n'est pas forcément linéaire. On peut y insérer des ramifications qui mettent diverses instructions ou blocs d'instructions à exécution au moyen de mécanismes d'évaluation. Un sketch n'agit pas seulement, mais réagit également à des influences externes, par exemple des signaux de capteur.



Attention !

Une erreur très fréquente chez les débutants consiste à confondre opérateur d'égalité et opérateur d'affectation. L'opérateur d'égalité `==` et l'opérateur d'affectation `=` ont des missions complètement différentes, mais sont souvent utilisés l'un à la place de l'autre. Le pire est que les deux modes d'écriture sont utilisables et valables dans une condition. Voici l'utilisation correcte de l'opérateur d'égalité :

```
if(buttonState == HIGH)
```

Voici maintenant l'utilisation erronée de l'opérateur d'affectation :

```
if(buttonState = HIGH)
```

Comment se fait-il que ce mode d'écriture ne produise pas des erreurs ? C'est très simple : il s'ensuit une affectation de la constante HIGH (valeur numérique 1) à la variable buttonState. 1 n'étant pas une valeur nulle, elle est interprétée comme étant true (vraie). Dans le cas d'une ligne de code `if(true)...`, l'instruction qui suit est toujours exécutée. Une valeur numérique 0 est évaluée comme

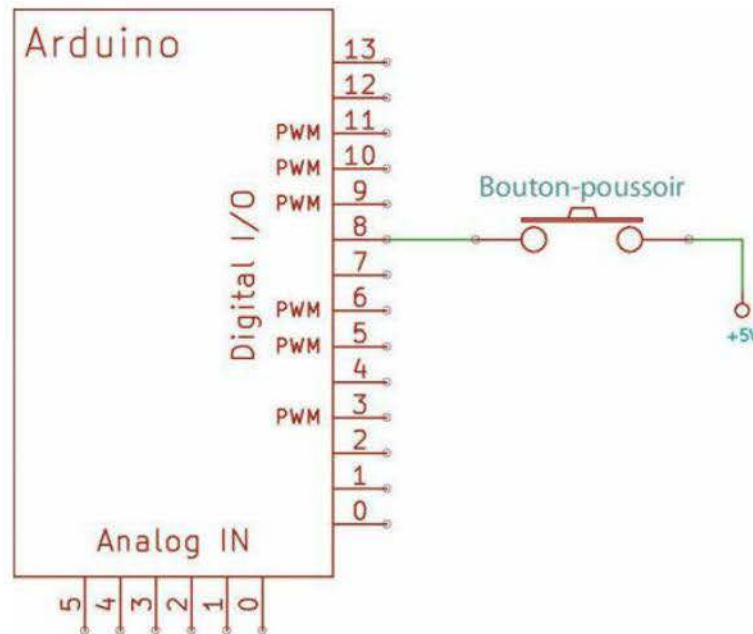
false (fausse) dans C/C++ et toute autre différente de 0 est évaluée comme true.

De telles erreurs sont difficiles à discerner et cela prend toujours énormément de temps.

Schéma

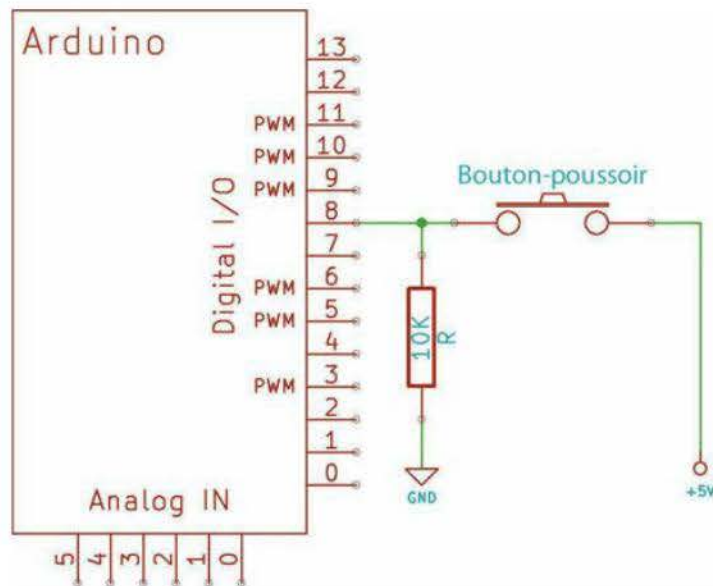
Voyons d'abord le raccordement du bouton-poussoir à l'entrée numérique. Je l'ai branché sur la broche 8, de manière à ce qu'il soit légèrement éloigné de la broche 13. J'aurais bien sûr pu utiliser n'importe quelle autre broche numérique.

Figure 2-5 ►
Carte Arduino avec un bouton-poussoir sur la broche 8



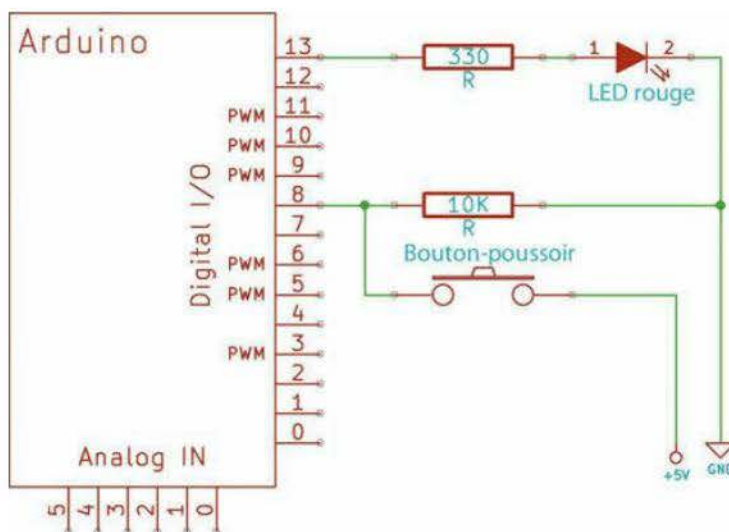
On voit ici que le bouton-poussoir est relié d'un côté à la broche numérique 8 et de l'autre à la tension de service de la carte Arduino, soit 5 V. C'est là tout le problème. Le circuit, tel que vous le voyez ici, ne fonctionne pas comme vous l'auriez peut-être imaginé. Si une entrée n'est alimentée par aucun niveau défini sous une forme HIGH ou LOW, le comportement dépend de facteurs divers, tels que par exemple l'énergie statique provenant de l'environnement ou l'humidité de l'air. Cela ressemble alors plus à un jeu de hasard qu'à un circuit stable. Pour remédier à ce problème, il existe diverses solutions dont certaines vous seront données au fur et à mesure. Une résistance dite *pull-down* est par exemple utilisée. Cette dernière tire littéralement le niveau ou plutôt le potentiel vers le bas. Comme un courant passe également par cette résistance, celle-ci doit être relativement forte.

Le circuit de la figure 2-6 montre cette résistance, qui tire la broche 8 à travers ses 10 k Ω (c'est la valeur empirique souvent utilisée dans la littérature) vers la masse quand le bouton-poussoir n'est pas enfoncé.



◀ **Figure 2-6**
Carte Arduino avec un bouton-
poussoir sur broche 8
et une résistance pull-down

Quand le bouton-poussoir est relâché, l'entrée numérique a donc un niveau LOW défini, clairement reconnu par le logiciel. Si, par contre, le bouton-poussoir est enfoncé, la résistance fait chuter les +5 V de la tension de service. Celle-ci est appliquée directement à la broche 8, qui est alors dotée d'un niveau HIGH défini. Ces connaissances préalables étant acquises, nous pouvons nous consacrer dorénavant au circuit réel.



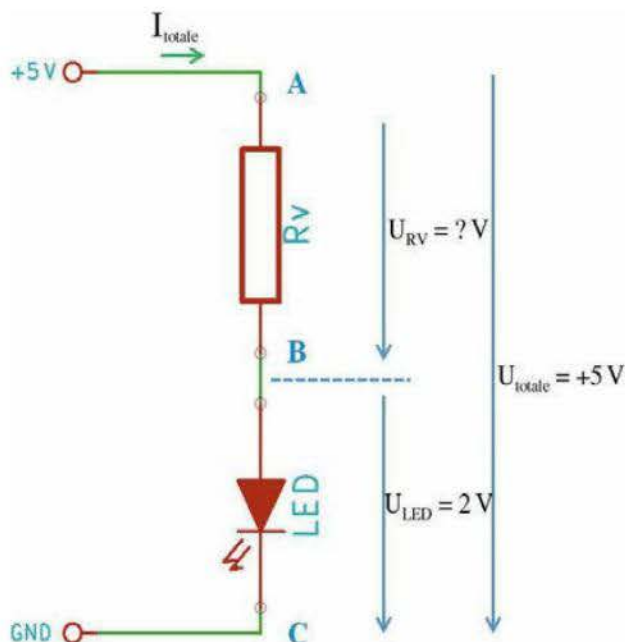
◀ **Figure 2-7**
Carte Arduino avec le circuit
complet pour bouton-poussoir
et LED



Excusez-moi de vous interrompre à nouveau mais quelque chose me turlupine. Dans le montage n° 1, vous avez utilisé une résistance de 220 ohms comme résistance série. Celle utilisée ici fait par contre 330 ohms. Ça ressemble encore à un jeu de hasard. Que dois-je prendre au juste ?

Cette question se justifie et je me dois d'y répondre. Je vais vous montrer comment on calcule une résistance série qui marche bien et ne pose aucun problème dans un circuit. La figure 2-8 montre un circuit avec une LED et une résistance série, ainsi que les valeurs du courant et de la tension correspondantes.

Figure 2-8 ►
LED avec résistance série et valeurs
du courant et de la tension



Pour calculer la valeur d'une résistance, on utilise de nouveau la loi d'Ohm. J'ai déjà adapté la formule générale à la résistance R à déterminer.

$$R = \frac{U}{I}$$

Mais comment déterminer le courant et la tension ? C'est très simple : +5 V sont appliqués à la résistance et à la LED, lesquelles sont montées en série. Cette tension est délivrée par la sortie d'une broche Arduino.

Aux bornes de la LED, on a normalement entre les points B et C une chute de tension d'environ +2 V, selon la LED utilisée et sa couleur. La tension aux bornes de la résistance série – donc entre les points A et B – est par conséquent la différence entre +5 V et +2 V, soit +3 V. Reste à connaître la grandeur du courant qui passe par la résistance et la LED. Rappelez-vous que le courant passant par tous les composants électroniques est le même dans un montage en série. La fiche technique de la carte Arduino nous apprend que le courant maximal fourni par une broche est de 40 mA. Cette valeur ne doit en aucun cas être dépassée, faute de quoi le microcontrôleur peut en souffrir. C'est pourquoi nous limitons le passage du courant en insérant cette résistance série R_V dans le circuit. Il est cependant conseillé, pour plus de sécurité, de ne pas prendre 40 mA, mais une valeur légèrement inférieure. Pour calculer la résistance série, j'utilise ici deux valeurs de courant différentes soit 5 mA et 10 mA, des valeurs situées entre 5 et 30 mA sont courantes pour une LED :

$$R_1 = \frac{U_{totale} - U_{LED}}{I_1} = \frac{5 \text{ V} - 2 \text{ V}}{10 \text{ mA}} = 300 \Omega$$

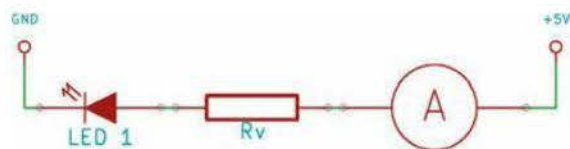
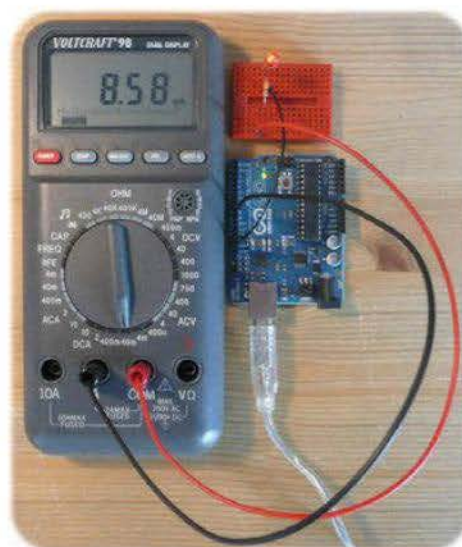
et

$$R_2 = \frac{U_{totale} - U_{LED}}{I_2} = \frac{5 \text{ V} - 2 \text{ V}}{5 \text{ mA}} = 600 \Omega$$

La valeur de la résistance série doit donc être comprise entre 300 et 600 ohms pour que le port de sortie de la carte Arduino ne soit que modérément sollicité. Des valeurs de résistance plus élevées peuvent bien sûr être prises pour limiter davantage le courant, mais cela se traduirait par une baisse de luminosité pour une LED, or vous souhaitez quand-même voir encore qu'elle est allumée. J'ai choisi dans tous les autres circuits une valeur de 330 ohms pour les LED avec résistance série. Toutes les valeurs de résistance possibles ne sont pas fabriquées, mais des E-séries sont proposées avec certaines classes. Lorsque vous achetez des résistances – des assortiments pratiques sont souvent proposés –, vous devez tenir compte également de la puissance dissipée maximale. Des résistances avec une puissance dissipée d' $\frac{1}{4}$ de watt sont ici largement suffisantes. Voici la théorie. Mais il n'est pas question de mesures réelles sur l'objet actif.

J'ai branché un multimètre sur le circuit électrique de la commande de LED pour mesurer le courant.

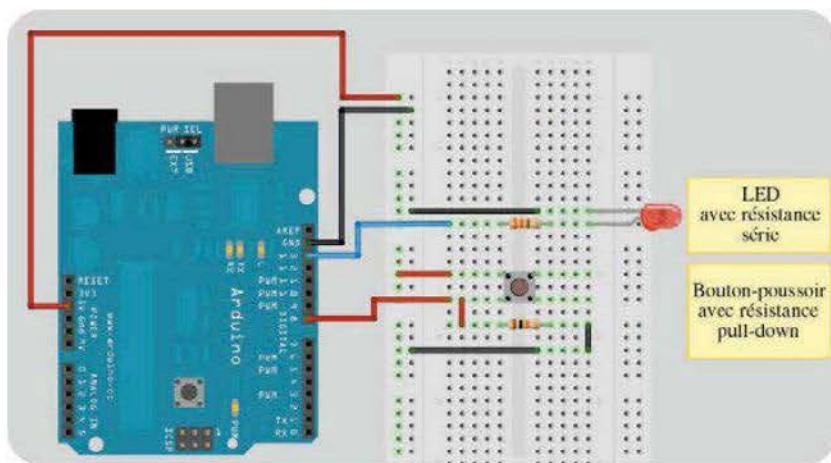
Figure 2-9 ►
Mesure de courant sur circuit
électrique de commande de LED
avec résistance série



J'ai choisi une résistance série de 330 ohms pour limiter le courant à 10 mA maxi. Le multimètre affiche un courant de 8,58 mA, soit pratiquement la valeur annoncée de 10 mA. La différence est due aux tolérances des composants et se veut même un plus faible que prévu.

Réalisation du circuit

La réalisation du circuit est déjà un peu plus complexe, c'est pourquoi nous allons nous servir de Fritzing (voir chapitre 6). Cet outil vraiment très utile est disponible sur le site Internet <http://fritzing.org>. Il va nous aider à construire le circuit et assembler les composants électroniques sur un document de travail. Vous pouvez télécharger ce logiciel gratuitement et l'utiliser pour vos projets.



◀ **Figure 2-10**
Réalisation du circuit avec Fritzing



Pour aller plus loin

Au cas où vous auriez oublié comment les différentes prises femelles d'une plaque d'essais sont reliées entre elles, consultez la section « La plaque d'essais sans soudure (breadboard) » du chapitre 7, page 155.

Problèmes courants

Si la LED ne s'allume pas quand le bouton-poussoir est enfoncé ou si la LED reste allumée, débranchez le port USB de la carte pour plus de sécurité et vérifiez ce qui suit.

- Vos fiches de raccordement sur la maquette correspondent-elles vraiment au circuit ?
- La LED a-t-elle été mise dans le bon sens, autrement dit sa polarité est-elle correcte ?
- Les boutons-poussoir peuvent être à 2 ou 4 connexions. S'il s'agit d'un modèle à 4 connexions, ont-elles été correctement branchées ? Faites, le cas échéant, un essai de continuité avec un multimètre et vérifiez ainsi l'adéquation du bouton-poussoir et des pattes correspondantes.
- Les deux résistances ont-elles bien les bonnes valeurs ou celles-ci ont-elles été interverties par mégarde ?
- Le code du sketch est-il correct ?

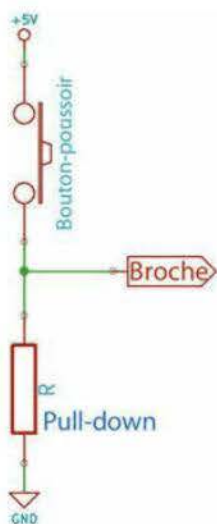
Autres possibilités pour des niveaux d'entrée définis

Avant de clore ce projet, je vais vous montrer d'autres possibilités d'avoir un niveau d'entrée défini sur une broche quand aucun signal ne lui est appliqué depuis l'extérieur. Les trois variantes suivantes sont importantes pour nous.

Avec une résistance pull-down

Vous avez déjà utilisé ce circuit.

Figure 2-11 ►
Circuit avec une résistance pull-down

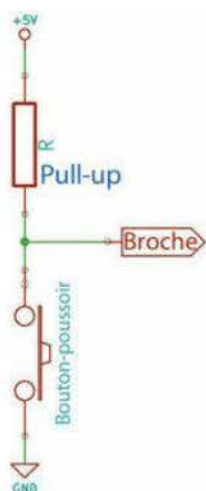


Quand le bouton-poussoir est ouvert, la broche d'entrée de votre microcontrôleur est au potentiel de la masse du fait de la résistance pull-down. Si le bouton-poussoir est fermé, la tension d'alimentation de +5 V est connectée sur la broche d'entrée.

Tableau 2-2 ►
Potentiels de la broche

État du bouton-poussoir	Potentiel de la broche
Ouvert	0 V (masse, niveau LOW)
Fermé	+5 V (tension d'alimentation, niveau HIGH)

Tout ce qui fonctionne avec une résistance connectée à la masse peut aussi être réalisé avec une résistance connectée à la tension d'alimentation. Les potentiels sont alors exactement inversés.



◀ **Figure 2-12**
Circuit avec une résistance pull-up

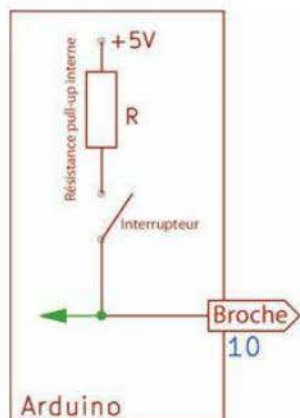
Quand le bouton-poussoir est ouvert, la tension de service de +5 V est appliquée via la résistance *pull-up* à la broche d'entrée de votre microcontrôleur. Si le bouton-poussoir est fermé, la broche est immédiatement reliée au potentiel de masse.

État du bouton-poussoir	Potentiel de la broche
Ouvert	+5 V (tension d'alimentation, niveau HIGH)
Fermé	0 V (masse, niveau LOW)

◀ **Tableau 2-3**
Potentiels de la broche

Avec la résistance pull-up du microcontrôleur

Une résistance pull-down ou pull-up séparée est en fait superflue car votre microcontrôleur dispose déjà de résistances pull-up internes incorporées aux broches numériques, qui peuvent être également mises en service par logiciel le cas échéant. On peut les imaginer comme suit :



◀ **Figure 2-13**
Résistance pull-up interne du microcontrôleur

J'ai choisi dans cet exemple la broche 10, à laquelle par exemple votre bouton-poussoir est relié. On peut voir que la résistance pull-up R relie la broche 10 à la tension d'alimentation de +5 V via un interrupteur électronique. La question est maintenant de savoir comment cet interrupteur peut être fermé pour que la broche présente un niveau HIGH en l'absence de signal d'entrée. Les instructions suivantes sont ici nécessaires :

```
pinMode(pin, INPUT); //Programmer la broche comme entrée
digitalWrite(pin, HIGH); //Activer la résistance pull-up interne
```



Eh là, pas si vite ! Il y a quelque chose qui cloche. Vous programmez une broche en tant qu'entrée parce que vous voulez y raccorder un bouton-poussoir. Jusque-là, ça va. Mais vous envoyez quelque chose avec digitalWrite à cette même broche qui n'a pas été programmée en tant que sortie. Qu'est-ce que ça veut dire ?

C'est ça le truc. La séquence d'instructions en question vous permet d'activer la résistance pull-up interne de 20 k Ω , laquelle fixe le potentiel à +5 V lorsque aucun signal n'est appliqué à l'entrée.



Attention !

Si vous choisissez une des deux variantes (résistance pull-up interne ou externe), vous devrez modifier légèrement votre code. Réfléchissez un peu avant de poursuivre votre lecture. Si le bouton-poussoir est relâché, un niveau LOW est appliqué à l'entrée de la broche quand vous travaillez avec une résistance pull-down. Le test, pour savoir si le bouton-poussoir est enfoncé, s'effectue au moyen de la ligne suivante :

```
if(buttonState == HIGH)
```

Jusqu'ici, tout va bien. Mais si vous travaillez maintenant avec une résistance pull-up, qui génère un signal HIGH quand le bouton-poussoir est ouvert, vous devez écrire la ligne :

```
if(buttonState == LOW)
```

dans laquelle vous avez remplacé HIGH par LOW. Compris ?



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- résistance pull-up ;
- résistance pull-down.

Qu'avez-vous appris ?

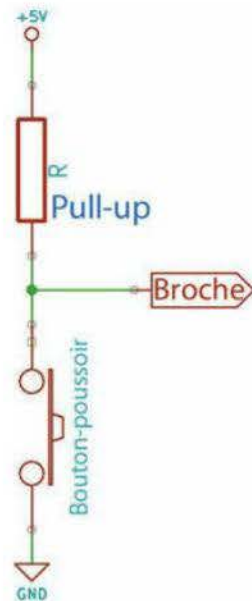
- Vous avez appris à utiliser plusieurs variables qui peuvent servir à diverses choses : déclaration pour broche d'entrée ou de sortie et enregistrement des informations d'état.
- Les broches numériques sont programmées implicitement comme entrées et n'ont pas besoin d'être explicitement programmées en tant que telles.
- Vous avez appris la fonction `digitalRead`, qui renvoie LOW ou HIGH à une entrée numérique en fonction du niveau appliqué. Cette valeur doit être affectée à une variable pour pouvoir être utilisée ultérieurement.
- Vous avez vu comment on peut contrôler le déroulement d'un sketch à l'aide de la structure de contrôle `if-else`.
- Différents schémas vous ont montré comment on représente graphiquement les connexions entre des composants électroniques.
- Une entrée numérique non connectée d'un composant électronique, dont le niveau n'est pas défini (HIGH ou LOW), conduit généralement à un comportement aléatoire et donc imprévisible du circuit.
- Aussi vous ai-je expliqué comment on utilise des résistances pull-down et pull-up, qui imposent un potentiel défini.
- Le microcontrôleur dispose de résistances pull-up internes de 20 k Ω , qui peuvent être activées via le logiciel. Vous pouvez ainsi vous épargner l'ajout de résistances pull-up externes.
- Le calcul d'une résistance série pour une LED ne vous pose maintenant plus aucun problème.
- Si vous n'avez pas lu le chapitre 6, vous avez fait la connaissance de l'outil Fritzing qui vous permet d'obtenir très rapidement des résultats dans la création de circuits par glisser-déposer.

Exercice complémentaire

Dans cet exercice, je souhaiterais vous présenter une tâche consistant à faire un *pull* de niveau numérique. Pull signifie « tirer » et c'est précisément ce que fait une résistance pull-down.

Mais le sens contraire est tout aussi possible. Une résistance pull-up permet de tirer un niveau vers le haut en direction de la tension d'alimentation. Voici à présent un tronçon de circuit que vous connaissez déjà :

Figure 2-14 ►
Résistance pull-up



Programmez votre sketch de telle sorte que le circuit fonctionne comme indiqué. La LED s'allume quand le bouton-poussoir est enfoncé. Elle s'éteint dès qu'il ne l'est plus. Le point *Broche* dans le circuit est ici relié à la broche 8 de votre carte Arduino. La commande de la LED demeure inchangée.

Clignotement avec gestion des intervalles

Au sommaire :

- la déclaration et l'initialisation de plusieurs variables ;
- la programmation de plusieurs broches aussi bien comme entrée (INPUT) que comme sortie (OUTPUT) ;
- l'instruction `digitalRead()` ;
- l'instruction `millis()` ;
- l'utilisation de la structure de contrôle `if-else` ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Appuyez sur le bouton-poussoir et il réagit

Dans notre premier exemple, nous avons vu comment interrompre l'exécution d'un sketch avec la fonction de retardement `delay`. La LED reliée à la broche de sortie numérique 13 clignotait à intervalles réguliers. Un tel circuit ou une telle programmation présente cependant un inconvénient que nous entendons déceler et éliminer. Pour cela, il faut approfondir un peu le circuit clignotant. Que se passerait-il si vous branchiez en plus un bouton-poussoir sur une entrée numérique pour interroger continuellement son état ?

Une LED est censée s'allumer si vous appuyez sur le bouton-poussoir. Peut-être voyez-vous déjà où je veux en venir ? Tant que

l'exécution du sketch est prisonnière de la fonction `delay`, le traitement du code est interrompu et l'entrée numérique ne peut être interrogée. Vous appuyez donc sur le bouton-poussoir et rien ne se passe.

Composants nécessaires



1 LED rouge



1 LED jaune



1 bouton-poussoir



1 résistance de 10 k Ω



2 résistances de 330 Ω



Plusieurs cavaliers flexibles de couleurs et de longueurs différentes

Code du sketch

Le code du sketch suivant ne fonctionne pas comme nous l'aurions voulu.

```
//Le code suivant ne fonctionne pas comme prévu
int ledPinBlink = 13;      //LED clignotante rouge en broche 13
int ledPinButton = 10;     //LED de bouton-poussoir jaune en
                           //broche 10

int buttonPin = 8;         //Bouton-poussoir en broche 8
int buttonState;           //Variable pour enregistrement
                           //de l'état du bouton-poussoir

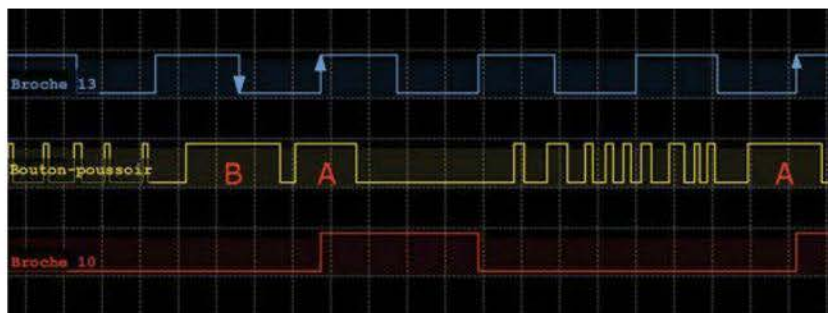
void setup(){
  pinMode(ledPinBlink, OUTPUT); //Broche LED clignotante comme sortie
  pinMode(ledPinButton, OUTPUT); //Broche LED de bouton-poussoir comme
  //sortie
  pinMode(buttonPin, INPUT);    //Broche bouton-poussoir comme entrée
}

void loop(){
  //Faire clignoter la LED clignotante
```

```
digitalWrite(ledPinBlink, HIGH); //LED rouge au niveau HIGH (5 V)
delay(1000);                      //Attendre une seconde
digitalWrite(ledPinBlink, LOW);  //LED rouge au niveau LOW (0 V)
delay(1000);                      //Attendre une seconde
//Interrogation de l'état du bouton-poussoir
buttonState = digitalRead(buttonPin);
if(buttonState == HIGH)
    digitalWrite(ledPinButton, HIGH); //LED jaune au niveau HIGH (5 V)
else
    digitalWrite(ledPinButton, LOW);  //LED jaune au niveau LOW (0 V)
}
```

Il y a quelque chose que je ne comprends pas. L'exécution repasse bien à un moment donné sur la ligne d'interrogation du bouton-poussoir dans la boucle sans fin. L'état est alors bien interrogé correctement.

Vous avez tout compris : l'expression « à un moment donné » sied ici à merveille ! Vous voulez cependant que le traitement du code réagisse à tout moment et pas seulement à un moment donné quand l'exécution reparaît. Les fonctions `delay` entravent bien quasiment la poursuite du sketch. C'est compris ? Je vous montre le comportement sur un chronogramme, où figurent les trois signaux pertinents, ceux de la LED clignotante (broche 13), du bouton-poussoir (broche 8) et de la LED du bouton-poussoir (broche 10).



◀ **Figure 3-1**

Chronogramme des signaux sur les broches 13, 8 et 10

Regardez le signal jaune qui représente l'état du bouton-poussoir. J'ai beau appuyer plusieurs fois sur ce dernier, le signal rouge sur la broche 10 ne réagit pas au début. Si je maintiens cependant le bouton-poussoir enfoncé pendant un temps plus long (à l'endroit signalé par un A), le signal de la broche 10 finit par passer lui aussi au niveau HIGH. Mais pourquoi ne s'est-il rien passé à l'endroit signalé par un B ? Je maintiens pourtant bien aussi le bouton-poussoir enfoncé pendant un temps plus long.

C'est très simple ! Vous avez deux activations de `delay` et la deuxième sert à faire durer le niveau `LOW`. Une fois ce délai écoulé, l'état du bouton-poussoir est interrogé très brièvement, c'est-à-dire précisément au moment où le niveau passe de `LOW` à `HIGH`. Le niveau sur la broche 10 réagit donc toujours au flanc montant (A) et ne réagit pas au flanc descendant (B). C'est simple, non ? Toujours est-il que nous devons ici renoncer à `delay` et choisir une autre solution, comme le montre l'exemple suivant. Ne vous en faites pas pour les lignes de code car nous allons le développer progressivement :

```
int ledPinBlink = 13; //LED clignotante rouge en broche 13
int ledPinButton = 10; //LED de bouton-poussoir jaune en broche 10
int buttonPin = 8; //Bouton-poussoir en broche 8
int buttonState; //Variable pour enregistrement de l'état du
                //bouton-poussoir

int interval = 2000; //Intervalle de temps (2 secondes)
unsigned long prev; //Variable de temps
int ledState = LOW; //Variable d'état pour la LED clignotante

void setup(){
  pinMode(ledPinBlink, OUTPUT); //Broche LED clignotante comme sortie
  pinMode(ledPinButton, OUTPUT); //Broche LED de bouton-poussoir comme
                                //sortie
  pinMode (buttonPin, INPUT); //Broche bouton-poussoir comme entrée
  prev = millis(); //Mémoriser le compteur de temps actuel
}

void loop(){
  //Faire clignoter la LED clignotante via la gestion des intervalles
  if((millis() - prev) > interval){
    prev = millis();
    ledState = !ledState; //Bascule état de la LED
    digitalWrite(ledPinBlink, ledState); //Bascule la LED rouge
  }
  //Interrogation de l'état du bouton-poussoir
  buttonState = digitalRead (buttonPin);
  if(buttonState == HIGH)
    digitalWrite(ledPinButton, HIGH); //LED jaune au niveau HIGH (5 V)
  else
    digitalWrite(ledPinButton, LOW); //LED jaune au niveau LOW (0 V)
}
```

Revue de code

Vous voyez ici que les variables à déclarer et à initialiser au début sont de plus en plus nombreuses. Faisons maintenant un peu le tour.

Variable	Objet
ledPinBlink	Contient le numéro de broche pour la LED sur la sortie numérique broche 13.
ledPinButton	Contient le numéro de broche pour la LED sur la sortie numérique broche 10.
buttonPin	Contient le numéro de broche pour le bouton-poussoir sur l'entrée numérique broche 8.
buttonState	Sert à enregistrer l'état du bouton-poussoir pour une exploitation ultérieure.
interval	Contient la valeur pour la gestion des intervalles.
prev	Enregistre la valeur actuelle de la fonction <code>millis</code> .
ledState	Mémoire l'état de la LED du bouton-poussoir.

◀ **Tableau 3-1**

Variables nécessaires et leur objet

Je commencerai par la gestion des intervalles car c'est le plus important. Le diagramme de la figure 3-2 nous montre une évolution chronologique avec certaines valeurs de temps marquantes. Je dois vous expliquer auparavant certaines choses dans le code source. Il s'agit d'une part de la nouvelle fonction `millis`, qui fournit en retour le temps écoulé depuis le début du sketch actuel en millisecondes. Il faut ici tenir compte de quelque chose d'important. Le type de la donnée de retour est `unsigned long`, donc un type de nombre entier de 32 bits non signé dont le domaine de valeurs s'étend de 0 à 4 294 967 295 ($2^{32} - 1$). Ce domaine de valeurs est aussi vaste parce qu'il doit être en mesure de traiter des valeurs correspondant à une période prolongée (49,71 jours maximum) sans débordement (dépassement de capacité).



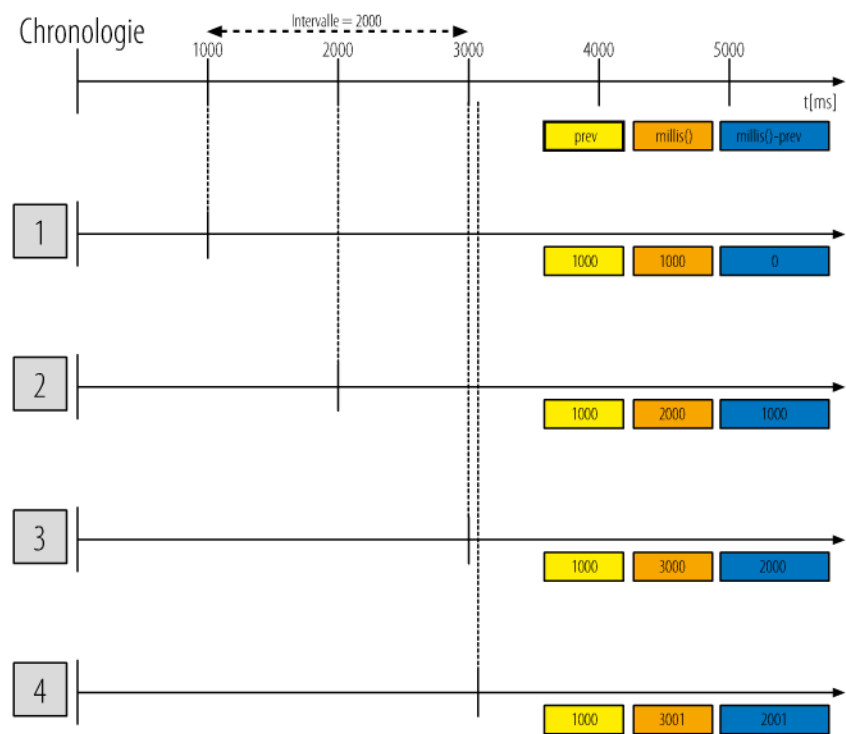
Pour aller plus loin

Un débordement signifie pour des variables que le maximum des valeurs que leur type de données peut traiter a été dépassé et va maintenant recommencer à 0. Pour le type de donnée `byte`, qui présente une donnée de 8 bits et peut, par conséquent, stocker $2^8 = 256$ états (de 0 à 255), un débordement se produit au moment de l'action $255 + 1$. Le type de donnée `byte` n'est plus en mesure de traiter la valeur 256.

Trois autres variables ont été ajoutées par mes soins, dont les rôles sont les suivants :

- `interval` (enregistre en ms le temps applicable à l'intervalle de clignotement) ;
- `prev` (enregistre en ms le temps actuellement écoulé. `prev` vient de *previous* qui signifie précédent) ;
- `ledState` (la LED clignotante est commandée en fonction de l'état HIGH ou LOW).

Figure 3-2 ►
Évolution chronologique
de la gestion des intervalles



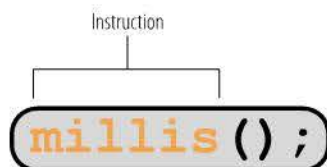
Analysons maintenant le diagramme, dans lequel j'ai pris au hasard des moments marquants pour plus de clarté. Le temps ne s'écoule évidemment pas réellement pendant ces étapes.

Tableau 3-2 ►
Contenu des variables
dans l'évolution chronologique

Moment	Explication
1	Le temps actuel en millisecondes (1 000 dans le cas présent) est chargé dans la variable <code>prev</code> . La chose se produit une seule fois dans la fonction <code>setup</code> . La différence <code>millis() - prev</code> donne la valeur 0 comme résultat. Cette valeur n'est pas supérieure à la valeur d'intervalle 2 000. La condition n'est pas remplie et le bloc <code>if</code> n'est pas exécuté.
2	1 000 ms plus tard, la différence <code>millis() - prev</code> est à nouveau calculée et il est vérifié que le résultat n'est pas supérieur à la valeur d'intervalle 2 000. 1 000 n'étant pas supérieur à 2 000, la condition n'est toujours pas remplie.
3	1 000 ms se sont encore écoulées, la différence <code>millis() - prev</code> est à nouveau calculée et il est vérifié que le résultat n'est pas supérieur à la valeur d'intervalle 2 000. 2 000 n'étant pas supérieur à 2 000, la condition n'est toujours pas remplie.
4	Après une durée de fonctionnement de 3 001 ms, la différence donne cependant une valeur supérieure à la valeur d'intervalle 2 000. La condition est remplie et le bloc <code>if</code> mis à exécution. L'ancienne valeur <code>prev</code> est remplacée par le temps actuel provenant de la fonction <code>millis</code> . L'état de la LED clignotante peut être inversé. Le jeu reprend au début sur la base de la nouvelle valeur temps dans la variable <code>prev</code> .

Pendant tout le déroulement, aucun arrêt n'a été inséré à aucun endroit sous la forme d'une pause dans le code source, si bien que l'interrogation de la broche numérique 8 pour gérer la LED du bouton-poussoir n'a été absolument pas gênée.

Une pression sur le bouton-poussoir est presque simultanément évaluée et affichée. La seule nouvelle instruction, qui s'avère être une fonction délivrant une valeur en retour, se nomme `millis`.



◀ **Figure 3-3**
L'instruction `millis`

On voit qu'il n'accepte aucun argument et présente par conséquent une paire de parenthèses vides. Sa valeur de retour possède le type de donnée `unsigned long`.

Une seule ligne me chagrine encore. Que signifie `ledState = !ledState` et qu'entend-on par bascule ?

Vous allez plus vite que moi dites donc ! Mais qu'à cela ne tienne puisque vous en parlez. La variable `ledState` stocke le niveau qui commande la LED rouge ou plutôt qui est en charge du clignotement (`HIGH` pour allumée et `LOW` pour éteinte). La ligne suivante :

```
digitalWrite(ledPinBlink, ledState);
```

permet de commander la LED. Le clignotement est précisément obtenu par un va-et-vient entre les états `HIGH` et `LOW`. Ce va-et-vient est également appelé bascule. Je vais reformuler la ligne de manière à la rendre peut-être plus claire.

```
if(ledState == LOW)
    ledState = HIGH;
else
    ledState = LOW;
```

La première ligne demande si le contenu de la variable `ledState` est égal à `LOW`. Si oui, il est mis sur `HIGH` ; sinon, il est mis sur `LOW`. Il s'agit également d'une bascule d'état. La variante à une ligne suivante, que j'ai déjà utilisée, est beaucoup plus courte.

```
ledState = !ledState; //Bascule état de la LED
```



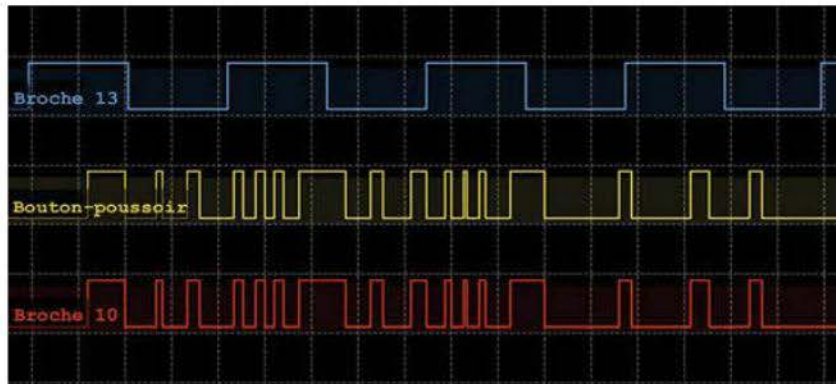
J'utilise ici l'opérateur logique `not`, représenté par le point d'exclamation. Il est souvent utilisé pour des variables booléennes qui ne peuvent accepter que les valeurs logiques `true` ou `false`. Le résultat de l'opérateur `not` est la valeur logique opposée à celle de l'opérande. Ceci est également valable pour les deux niveaux `HIGH` et `LOW`.

Le port 8 est finalement interrogé tout à fait normalement et sans retardement du bouton-poussoir.

```
buttonState = digitalRead(buttonPin);
if(buttonState == HIGH)
    digitalWrite(ledPinButton, HIGH);
else
    digitalWrite(ledPinButton, LOW);
```

Je vous montre à nouveau le comportement sur un chronogramme, dans lequel les trois signaux en question – à savoir la LED clignotante (broche 13), le bouton-poussoir (broche 8) et la LED de bouton-poussoir – sont représentés l'un en dessous de l'autre de la même manière que plus haut :

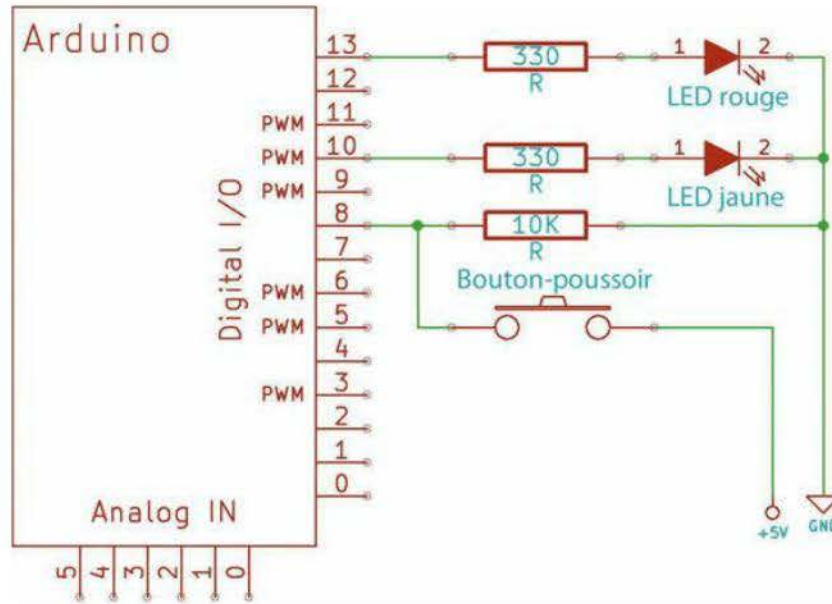
Figure 3-4 ►
Chronogramme des signaux
sur les broches 13, 8 et 10



On voit que le signal bleu représente la LED clignotante sur la broche 13. Si maintenant j'actionne à intervalles réguliers le bouton-poussoir – représenté par le signal jaune – sur la broche 8, le signal rouge de la LED du bouton-poussoir réagit aussitôt. Aucun retard et aucune interruption ne sont à observer. Le comportement du circuit est exactement celui que nous voulions.

Schéma

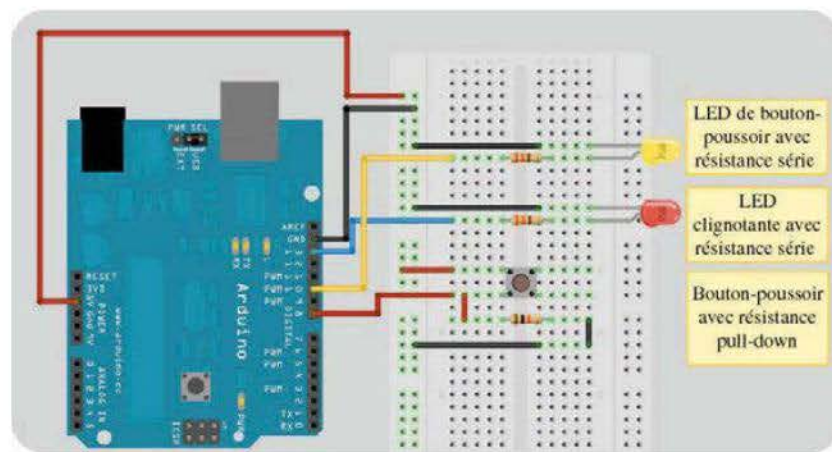
La lecture du schéma ne devrait plus vous poser de problème maintenant. Seule une autre LED, censée réagir quand le bouton-poussoir est enfoncé, a été ajoutée.



◀ **Figure 3-5**
Carte Arduino avec un bouton-poussoir et deux LED

Réalisation du circuit

La plaque d'essais est maintenant un peu plus remplie.



◀ **Figure 3-6**
Construction du circuit avec Fritzing



Pour aller plus loin

Comme vous avez pu le voir dans cette réalisation et aussi dans la précédente, j'utilise des cavaliers flexibles de couleurs différentes. Quand vous composez des circuits sur votre plaque d'essais, je vous conseille d'utiliser également des couleurs différentes. J'ai choisi par exemple le rouge pour la tension d'alimentation et le noir pour la masse. Les autres lignes de signal peuvent être bleues, jaunes ou même rouges.

Il n'y pas de règle précise en la matière, mais vous devriez constituer votre propre système de couleurs afin de conserver une bonne vue d'ensemble. Cela peut être également utile aux personnes extérieures de trouver une maquette conçue proprement.

Problèmes courants

Si la LED ne s'allume pas quand le bouton-poussoir est enfoncé ou si la LED reste allumée, vérifiez ce qui suit.

- Vos fiches de raccordement sur la plaque d'essais correspondent-elles vraiment au schéma ?
- Les LED ont-elles été mises dans le bon sens ? Pensez à la polarité !
- Les boutons-poussoir peuvent être à 2 ou 4 connexions. S'il s'agit d'un modèle à 4 connexions, ont-elles été correctement branchées ? Faites, le cas échéant, un essai de continuité avec un multimètre et vérifiez ainsi l'adéquation du bouton-poussoir et des pattes correspondantes.
- Les deux résistances ont-elles bien les bonnes valeurs ? Ont-elles été éventuellement interverties ?
- Le code du sketch est-il correct ?

Qu'avez-vous appris ?

- Vous savez utiliser plusieurs variables à des fins les plus diverses (déclaration pour broche d'entrée ou de sortie et enregistrement des informations d'état).
- L'instruction `delay` interrompt le déroulement du sketch et instaure une pause, de telle sorte que toutes les instructions subséquentes ne soient pas exécutées tant que le temps d'attente n'est pas écoulé.
- Vous avez appris, à travers la gestion des intervalles avec la fonction `millis`, un moyen permettant de maintenir malgré tout

l'exécution continue du sketch de la boucle sans fin `loop`, de telle sorte que d'autres instructions de la boucle `loop` soient exécutées et qu'une utilisation d'autres capteurs, tels que le bouton-poussoir raccordé, soit possible.

- Vous avez appris à lire divers chronogrammes, qui représentent très bien graphiquement les différents états de niveau dans la courbe d'évolution temporelle.

Exercice complémentaire

Créez simplement un sketch qui allume la LED en cas de pression sur le bouton-poussoir et qui l'éteint en cas de nouvelle pression, et ainsi de suite. Un cas épineux qui fera l'objet du montage suivant. Il se peut que vous rencontriez un problème que nous résoudrons plus tard. Le mot-clé est rebond.

Le bouton-poussoir récalcitrant

Dans ce montage, vous verrez qu'un bouton-poussoir, ou aussi un interrupteur, ne se comporte pas toujours comme vous l'auriez voulu. Prenons pour exemple un bouton-poussoir qui, en théorie, ferme le circuit tant qu'il reste enfoncé, et le rouvre quand il est relâché. Rien de neuf en soi et rien de bien difficile à comprendre. Mais les circuits électroniques, dont le rôle consiste par exemple à déterminer le nombre exact de pressions sur le bouton-poussoir pour une exploitation ultérieure, posent un problème dont on ne peut se douter au départ.

Une histoire de rebond

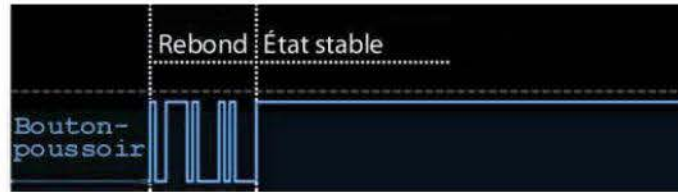
Le mot-clé de ce montage est rebond. Appuyer sur un bouton-poussoir normal et même le maintenir enfoncé revient à fermer le contact mécanique une seule et unique fois dans le bouton-poussoir. Ce n'est pourtant pas le cas la plupart du temps, car le composant en question ouvre et referme plusieurs fois le contact dans un intervalle de temps très court, de l'ordre de la milliseconde. Les surfaces de contact d'un bouton-poussoir ne sont en général pas complètement lisses, et on peut voir de multiples aspérités et impuretés quand on les observe au microscope électronique. Ainsi, les points de contact des matériaux conducteurs ne se touchent pas instantanément et pas durablement au moment du rapprochement. L'effet en question peut aussi être obtenu par vibration ou montage sur ressorts du matériau, le contact étant alors brièvement fermé puis de nouveau ouvert plusieurs fois l'une derrière l'autre lors de la jonction.

Ces impulsions délivrées par le bouton-poussoir sont enregistrées et traitées en bonne et due forme par le microcontrôleur, c'est-à-dire comme si vous appuyiez très vite et très souvent sur le bouton-pous-



soir. Ce comportement est bien sûr gênant et doit être évité d'une manière ou d'une autre. Regardons maintenant le chronogramme de plus près.

Figure 4-1 ►
Bouton-poussoir à rebond



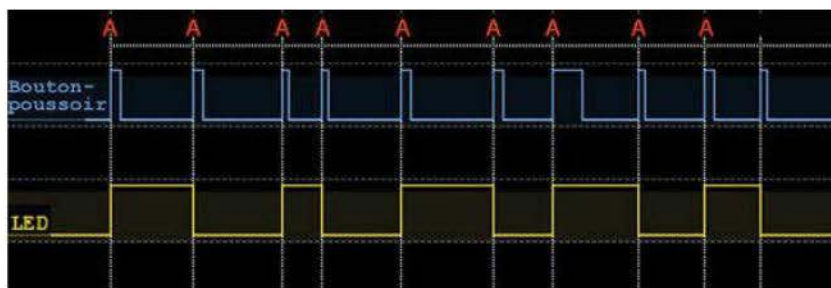
J'ai appuyé une seule fois sur le bouton-poussoir et l'ai ensuite maintenu appuyé, mais celui-ci a interrompu plusieurs fois la liaison souhaitée avant que l'état stable de la connexion ne soit atteint. Cette suite de fermetures et d'ouvertures du circuit jusqu'à ce que le niveau HIGH définitif souhaité soit atteint est appelée rebond. Ce comportement peut aussi se constater dans l'opération inverse. Si je relâche le bouton-poussoir, plusieurs impulsions peuvent éventuellement être générées jusqu'à ce que j'obtienne enfin le niveau LOW souhaité. Le rebond du bouton-poussoir est à peine perceptible voire carrément invisible par l'œil humain et si un circuit censé commander une LED quand le bouton-poussoir est enfoncé était construit, les différentes impulsions se verraient comme un niveau HIGH du fait de la persistance oculaire. Essayons donc une autre solution. Nous pourrions construire un circuit avec un bouton-poussoir sur une entrée numérique et une LED sur une autre sortie numérique.



Mais il n'y a rien de nouveau là-dedans. Qu'est-ce que ça apporte ? Vous venez de dire que ce type de rebond possible ne se voyait pas dans un circuit.

Notre circuit n'est pas le seul composant. Il y a certes le matériel mais il y a aussi le logiciel et nous entendons le configurer de telle sorte que la LED s'allume à la première impulsion. Elle doit s'éteindre à l'impulsion suivante et se rallumer à celle d'après et ainsi de suite. Nous avons donc affaire à une bascule du niveau logique. Si maintenant plusieurs impulsions sont enregistrées par le circuit ou plutôt par le logiciel quand le bouton-poussoir est appuyé, la LED change alors plusieurs fois d'état.

Dans le cas d'un bouton-poussoir sans rebond, les états doivent être tels que représentés dans le diagramme de la figure 4-2.



◀ **Figure 4-2**
Changement du niveau de la LED
pour un appui sur le bouton-
poussoir

On voit que dans le cas de multiples appuis sur le bouton-poussoir (flanc montant), lesquels sont ici indiqués par un A, l'état de la LED bascule. Comment faire pour que le logiciel se charge de l'exécution ? Voyons maintenant la liste des composants.

Composants nécessaires

Pour le circuit suivant, j'ai trouvé un ancien bouton-poussoir dans mon bric-à-brac qui, lui, rebondira à coup sûr énergiquement. Les nouveaux boutons-poussoir disponibles peuvent présenter une protection mécanique contre le rebond avec un point d'appui reconnaissable. Quand on appuie dessus, on peut entendre un léger claquement. Cela indique que le contact a été fermé avec une pression ou une vitesse plus élevée de manière à empêcher ou minimiser le rebond.



1 LED rouge



1 bouton-poussoir (sans protection antirebond)



1 résistance de 330 Ω



1 résistance de 10 k Ω



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Le code du sketch est le suivant pour cet exemple.

```
int buttonPin = 2; //Bouton-poussoir en broche 2

int buttonValue = 0; //Variable pour enregistrer l'état du bouton-
//poussoir
int previousButtonValue = 0; //Variable pour enregistrer l'ancien
//état du bouton-poussoir
int ledPin = 8; //LED en broche 8
int counter = 0; //Variable de compteur

void setup(){
  pinMode(buttonPin, INPUT); //Broche bouton-poussoir comme entrée
  pinMode(ledPin, OUTPUT); //Broche LED comme sortie
}

void loop(){
  buttonValue = digitalRead (buttonPin); //Interrogation du
//bouton-poussoir
//La valeur précédente du bouton-poussoir est-elle différente
//de la valeur actuelle ?
  if(previousButtonValue != buttonValue){
    if(buttonValue == HIGH){
      counter++; //Incrémentation du compteur (+1)
    }
  }
  previousButtonValue = buttonValue; //Sauvegarde de la valeur
//actuelle du bouton-poussoir
  if(counter % 2 == 0) //La variable du compteur est-elle un
//nombre pair ?
    digitalWrite(ledPin,HIGH);
  else
    digitalWrite(ledPin,LOW);
}
```

Le code ne semble pas très compliqué à première vue, mais il est cette fois-ci un peu plus subtil. Vous allez bientôt voir dans quelle mesure.

Revue de code

On commence ici aussi par déclarer et initialiser une série de variables globales.

Variable	Objet
buttonPin	Cette variable contient le numéro de broche pour le bouton-poussoir (2).
buttonValue	Cette variable enregistre l'état du bouton-poussoir.
previousButtonValue	Cette variable sert à enregistrer l'état précédent du bouton-poussoir.
ledPin	Cette variable contient le numéro de broche pour la LED (8).
Counter	Cette variable mémorise les niveaux HIGH de l'état du bouton-poussoir.

◀ **Tableau 4-1**

Variables nécessaires et leur objet

L'initialisation des différentes broches au sein de la fonction `setup` ne nécessitant aucune explication supplémentaire, passons directement à la fonction `loop`. Le niveau sur la broche du bouton-poussoir est continuellement interrogé via la fonction `digitalRead` et sauvegardé dans la variable `buttonValue` :

```
buttonValue = digitalRead(buttonPin) ;
```

La tâche du sketch consiste cependant à détecter chaque appui – représenté par un niveau `HIGH` – sur le bouton-poussoir, et à incrémenter en conséquence une variable de compteur. Normalement, les lignes de code suivantes devraient suffire.

```
void loop(){
    buttonValue = digitalRead(buttonPin); //Interrogation du bouton-
                                         //poussoir

    if(buttonValue == HIGH){
        counter++;                      //Incrémentation du compteur (+1)
    }
    // ...
}
```

Seulement le code comporte une erreur critique. La variable de compteur est incrémentée à chaque nouveau passage de la fonction `loop` quand le bouton-poussoir est enfoncé, et plus vous appuyez longtemps sur le bouton-poussoir, plus la variable est incrémentée. Mais le contenu de la variable ne doit être incrémenté que de 1 quand le bouton-poussoir est enfoncé. Comment faire pour modifier ce comportement du code ? La solution est en fait très simple. Il suffit de sauvegarder temporairement le niveau sur la broche du bouton-poussoir dans une variable après chaque interrogation. La nouvelle valeur est alors comparée à l'ancienne lors de l'interrogation suivante. Si les deux niveaux sont différents, vous devez simplement vérifier que la nouvelle valeur correspond au niveau `HIGH`, car ce sont eux qu'il faut décompter. Le nouveau niveau du moment est ensuite sauvegardé temporairement pour la prochaine comparaison, et tout reprend depuis le début.



Mais si le compteur est incrémenté à chaque appui sur le bouton-poussoir, comment fait-on pour allumer ou éteindre la LED ? Celle-ci doit pourtant s'allumer à chaque 1^{er}, 3^e, 5^e, 7^e appui, etc., et s'éteindre à chaque 2^e, 4^e, 6^e, 8^e appui, etc., sur le bouton-poussoir.

C'est précisément l'approche que nous avons utilisée pour résoudre le problème. Ce qu'il faut c'est évaluer d'une manière ou d'une autre le contenu de la variable du compteur. Ne remarquez-vous rien quand vous regardez les valeurs responsables de l'allumage de la LED ?



J'y suis ! Toutes les valeurs qui doivent faire allumer la LED sont impaires et les autres sont paires.

Exactement, c'est la solution. Il nous faut donc trouver le moyen de programmer quelque chose qui nous permette de tester la parité ou l'impairité d'une valeur. Je vous donne une piste : quand vous divisez un nombre par 2, vous avez un reste nul si le nombre est pair mais vous en avez un non nul si le nombre est impair. Jetons maintenant un coup d'œil au tableau 4-2.

Tableau 4-2 ►
Division de nombres entiers par 2

Division	Résultat et reste de la division	Reste non nul ?
1/2	0 Reste 1	Oui
2/2	1 Reste 0	Non
3/2	1 Reste 1	Oui
4/2	2 Reste 0	Non
5/2	2 Reste 1	Oui
6/2	3 Reste 0	Non

On voit donc que seules des valeurs impaires donnent un reste non nul. Un opérateur spécial est utilisé en programmation pour déterminer le reste. Il s'agit de l'opérateur modulo, qui est représenté par le signe %. La première des lignes de code vérifie si la valeur du compteur est paire ou impaire :

```
if(counter % 2 == 0)    //La variable de compteur est-elle un nombre
                        //pair ?
    digitalWrite(ledPin, HIGH);
else
    digitalWrite(ledPin, LOW);
```

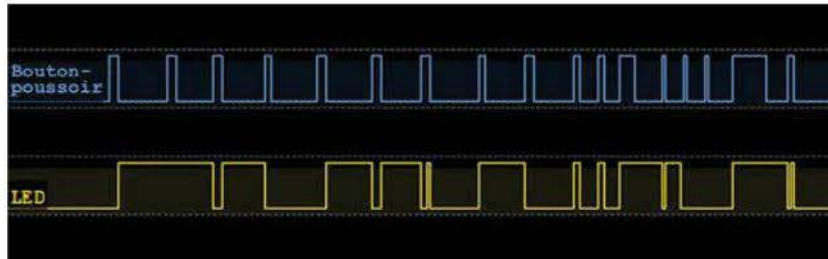
La LED est allumée quand les valeurs sont paires, et éteinte quand elles sont impaires.



Attention !

Les opérandes de l'opérateur modulo % doivent être d'un type de donnée à nombre entier, tel que par exemple `int`, `byte` ou `unsigned int`.

Voyons maintenant comment le circuit se comporte quand nous appuyons plusieurs fois – disons toutes les secondes – sur le bouton-poussoir. Le résultat est présenté dans le chronogramme de la figure 4-3.



◀ **Figure 4-3**

Changement du niveau de la LED pour un appui sur le bouton-poussoir

Ce n'est certainement pas le comportement que nous avions prévu. La LED ne bascule pas au rythme de l'appui sur la touche, mais a le comportement typique d'un bouton-poussoir ou d'un interrupteur à rebond. Que faire pour que le rebond n'ait pas ce genre de conséquence sur le circuit ou le compteur ? Une des solutions consiste à ajouter une temporisation pour diminuer le rebond. Ajoutez simplement un ordre `delay` derrière l'évaluation du compteur :

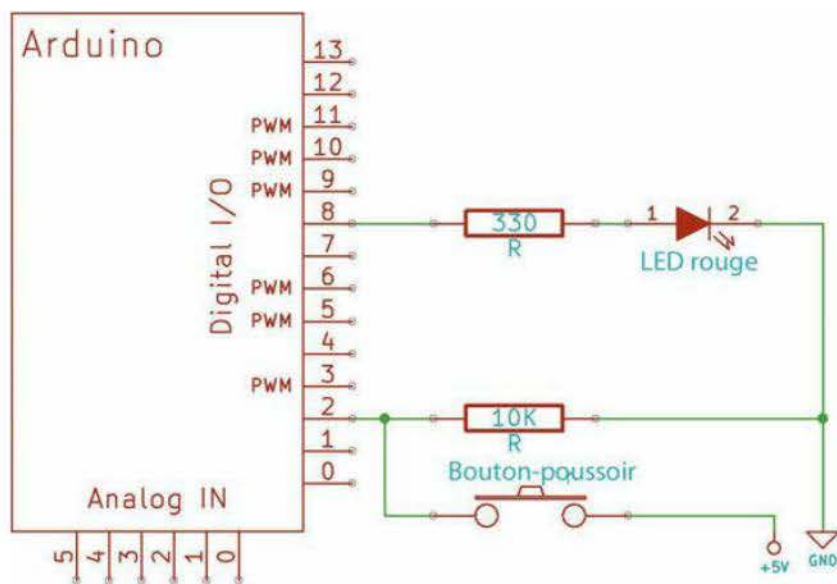
```
if(counter % 2 == 0)
    digitalWrite(ledPin, HIGH);
else
    digitalWrite(ledPin, LOW);
delay(10);           //Attendre 10 ms avant d'interroger à nouveau
                     //le bouton-poussoir
```

J'ai choisi ici une valeur de 10 millisecondes car elle convenait très bien pour mon bouton-poussoir. La valeur correcte ou optimale dépend naturellement toujours de la vitesse à laquelle vous souhaitez actionner le bouton-poussoir plusieurs fois d'affilée, de sorte que le logiciel puisse encore réagir. Essayez différentes valeurs et choisissez celle qui vous convient.

Schéma

Le schéma vous est certainement familier. Le logiciel utilisé est quant à lui légèrement différent.

Figure 4-4 ►
Carte Arduino avec bouton-poussoir
et LED illustrant le rebond



Autres possibilités de compenser le rebond

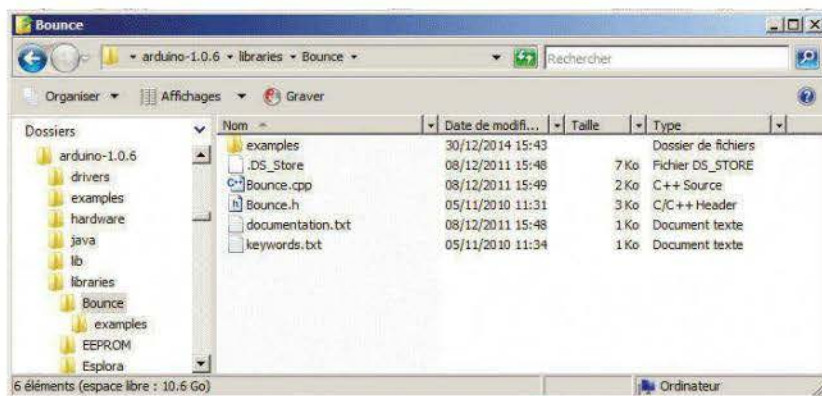
Nous avons jusqu'ici proposé une possibilité de compenser le rebond d'un composant mécanique tel que par exemple le bouton-poussoir. Mais il en existe d'autres :

1. boutons-poussoir spéciaux qui n'ont pas de rebond et disposent d'un point d'appui fixe ;
2. utilisation d'une bibliothèque spécialement prévue à cet effet, appelée Bounce ;
3. par l'ajout d'un petit dispositif matériel basé sur un circuit RC.

Je souhaite m'arrêter un peu sur le point 2. Si le point 3 vous intéresse également, vous trouverez de nombreuses informations sur Internet. Une bibliothèque, également appelée librairie, est un composant logiciel développé par exemple par d'autres programmeurs pour résoudre un problème particulier. Pour ne pas réinventer la roue à chaque fois, le code en question est conditionné sous forme de bibliothèque et mis à disposition des autres utilisateurs. Si ces bibliothèques sont en accès libre – et c'est la plupart du temps le cas pour l'environnement Arduino –, vous pouvez les utiliser sans problème dans votre projet.

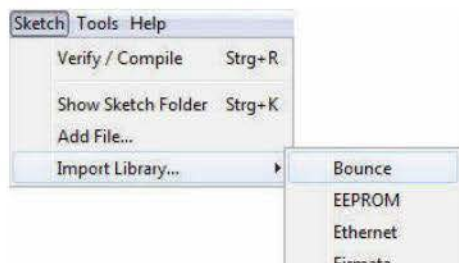
Vous trouverez la bibliothèque Bounce sur <http://www.arduino.cc/playground/Code/Bounce>. Vous pouvez la télécharger sous forme

d'un fichier .zip compressé. Décompressez-le dans le répertoire `arduino-1.x\libraries\`, dans lequel figurent déjà d'autres bibliothèques livrées avec le logiciel Arduino. Vous devez alors obtenir une structure de fichiers comme celle de la figure 4-5.



◀ **Figure 4-5**
Structure de la bibliothèque Bounce

Si maintenant vous programmez le sketch dans lequel vous utilisez cette bibliothèque, vous êtes assisté par l'environnement de développement et obtenez l'aide nécessaire après avoir inséré la bibliothèque dans votre projet. Vous devez d'abord signaler d'une manière ou d'une autre à votre compilateur que vous souhaitez incorporer du code étranger. Utilisez pour cela l'instruction de prétraitement `#include`. Des explications plus précises vous seront bientôt données à ce sujet. Il vous suffit, après avoir décompressé le code dans le répertoire en question, d'ajouter l'instruction `#include` en utilisant les entrées du menu de l'IDE représentées à la figure 4-6.



◀ **Figure 4-6**
Intégration de la bibliothèque Bounce dans votre sketch

Par le menu `Sketch>Import Library`, on peut voir la liste de toutes les bibliothèques disponibles dans le répertoire `libraries`. On choisit l'entrée `Bounce`, ce qui écrit automatiquement la directive de préprocesseur `#include` sur la première ligne de l'éditeur. Après cette ligne, on écrit le code qui peut, par exemple, se présenter comme ce qui suit :


```

#include <Bounce.h>      //Intégrer la bibliothèque Bounce
int ledPin = 12;         //LED en broche 12
int buttonPin = 8;       //Bouton-poussoir en broche 8
int waitTime = 10;       //Temps d'attente = 10 ms
Bounce debouncing = Bounce(buttonPin, waitTime);
                        //Générer un objet Bounce

void setup(){
  pinMode(ledPin, OUTPUT); //Broche de LED comme sortie
  pinMode(buttonPin, INPUT); //Broche de bouton-poussoir comme entrée
}

void loop(){
  debouncing.update();      //Mise à jour de l'antirebond
  int Value = debouncing.read(); //Lecture valeur de mise à jour
  if (Value == HIGH)
    digitalWrite(ledPin, HIGH); //Allumer LED
  else
    digitalWrite(ledPin, LOW);  //Éteindre LED
}

```

Vous saurez bientôt ce qu'est ici un objet. Prenez seulement le code comme il est. Je vous conseille d'utiliser le code que nous avons créé pour le circuit devant basculer la LED à chaque appui sur le bouton-poussoir. Il convient mieux pour vérifier que la bibliothèque Bounce fonctionne bien.



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- rebond bouton-poussoir ;
- antirebond.

Réalisation du circuit

Ayant déjà construit un circuit similaire avec Fritzing (voir montage n° 2), je n'ai pas besoin de le représenter ici.

Problèmes courants

Si la LED ne s'allume pas ou ne bascule pas, plusieurs choses peuvent en être la cause.

- La LED peut avoir été mal polarisée. Rappelez-vous les deux différentes connexions d'une LED que sont l'anode et la cathode.
- La LED est peut-être défectueuse et a été grillée par une surtension lors des montages précédents. Testez-la avec une résistance série sur une source d'alimentation de 5 V.
- Vérifiez les branchements de la LED et des composants sur votre plaque d'essais.
- Vérifiez le sketch que vous avez entré dans l'éditeur de l'IDE. Peut-être avez-vous oublié une ligne ou commis une erreur ou peut-être le sketch a-t-il mal été transmis ?
- Vérifiez le bon fonctionnement du bouton-poussoir utilisé avec un testeur de continuité ou un multimètre.

Qu'avez-vous appris ?

- Vous savez maintenant que des composants mécaniques tels que des boutons-poussoir ou des interrupteurs ne se ferment ou ne s'ouvrent pas immédiatement. Plusieurs brèves interruptions successives peuvent résulter par exemple de tolérances de fabrication, d'impuretés ou de matériel en vibration, avant d'arriver à un état stable. Ce comportement est enregistré et traité comme tel par des circuits électroniques. Si par exemple vous devez compter le nombre d'appuis sur le bouton-poussoir, ces impulsions multiples peuvent s'avérer extrêmement gênantes.
- Ce comportement peut être corrigé de différentes manières :
 - par une solution logicielle (par exemple une stratégie de temporisation lors de l'interrogation du signal d'entrée) ;
 - par une solution matérielle (par exemple un circuit RC).
- Vous savez comment intégrer dans votre sketch une bibliothèque externe créée par d'autres développeurs, et aussi ce qu'est la directive de prétraitement `#include`.

Exercice complémentaire

Dans cet exercice, je voudrais que vous construisiez un circuit commandant plusieurs LED. Disons qu'il doit en avoir au moins 5. Le logiciel doit allumer une nouvelle LED dans la chaîne à chaque appui sur le bouton-poussoir. Le rebond est ainsi bien visible, quand

en effet plusieurs LED s'allument directement lors d'un appui sur le bouton-poussoir. Corrigez alors la programmation de telle sorte que le rebond n'ait plus aucune incidence, et vérifiez-la avec le circuit.

Astuce

Vous pouvez utiliser un bargraphe à LED pour composer une chaîne avec beaucoup de LED commutées l'une derrière l'autre. Il existe plusieurs versions, dans lesquelles les différentes LED sont idéalement logées dans un boîtier.

Certains composants peuvent comporter 10 et même 20 éléments de LED.

Figure 4-7 ►
Bargraphe de type YBG 2000
avec 20 éléments de LED



Une résistance série appropriée est ici aussi impérativement nécessaire.

Le séquenceur de lumière

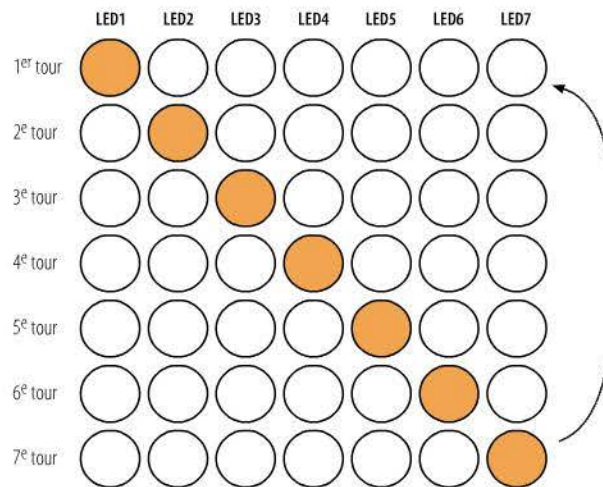
Au sommaire :

- la déclaration et l'initialisation d'un tableau (array) ;
- la programmation de plusieurs broches comme sortie (OUTPUT) ;
- l'utilisation d'une boucle `for` ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Qu'est-ce qu'un séquenceur de lumière ?

Vous maîtrisez maintenant suffisamment les LED pour être en mesure de réaliser des montages où clignotent plusieurs diodes électroluminescentes. Ça n'a l'air de rien dit comme ça, mais ce n'est pas si simple. Nous allons commencer par un séquenceur de lumière, qui commande une par une différentes LED. Dans ce montage, les LED branchées sur les broches numériques devront s'allumer conformément au modèle présenté sur la figure 5-1.

Figure 5-1 ►
Séquence d'allumage des 7 LED



À chaque tour, la LED s'allume une position plus à droite. Arrivé à la fin, le cycle reprend au début. Vous pouvez programmer les diverses broches, qui toutes sont censées servir de sortie, de différentes manières. Dans l'état actuel de vos connaissances, vous devez déclarer sept variables et les initialiser avec les valeurs de broche correspondantes. Ce qui pourrait donner ceci :

```
int ledPin1 = 7;
int ledPin2 = 8;
int ledPin3 = 9;
...
```

Chaque broche doit être ensuite programmée dans la fonction setup avec pinMode comme sortie, ce qui représente aussi un travail de saisie considérable :

```
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
pinMode(ledPin3, OUTPUT);
...
```

Voici donc la solution. Je voudrais vous présenter un type intéressant de variable, capable de mémoriser plusieurs valeurs du même type de donnée sous un même nom.



Vous rigolez ! Comment une variable peut-elle mémoriser plusieurs valeurs sous un seul et même nom ? Et comment dois-je faire pour sauvegarder ou appeler les différentes valeurs ?

Patience ! C'est possible. Cette forme spéciale de variable est appelée tableau (array). On n'y accède pas seulement par son nom évocateur, car une telle variable possède aussi un index. Cet index est un nombre entier incrémenté. Ainsi, les différents éléments du tableau – c'est le nom donné aux valeurs stockées – peuvent être lus ou modifiés. Vous allez voir comment dans le code du sketch ci-après.

Composants nécessaires



7 LED rouges



7 résistances de 330 Ω



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Voici le code du sketch pour commander le séquenceur de lumière à sept LED :

```
int ledPin[] = {7, 8, 9, 10, 11, 12, 13}; //Tableau de LED avec
                                           //numéros des broches

int waitTime = 200; // Pause entre les changements en ms

void setup()
{
  for(int i = 0; i < 7; i++)
    pinMode(ledPin[i], OUTPUT); //Toutes les broches du tableau comme
                                //sorties
}

void loop()
{
  for(int i = 0; i < 7; i++)
  {
    digitalWrite(ledPin[i], HIGH); //Élément du tableau au niveau HIGH
    delay(waitTime);
    digitalWrite(ledPin[i], LOW); //Élément du tableau au niveau LOW
  }
}
```

Revue de code

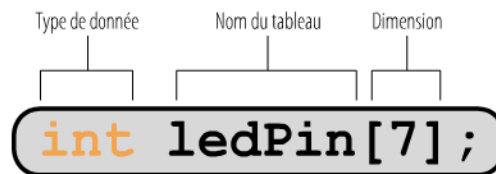
Les variables suivantes sont techniquement nécessaires à notre programmation expérimentale.

Tableau 5-1 ▶
Variables nécessaires et leur objet

Variable	Objet
ledPin	Tableau pour enregistrer les différentes broches sur lesquelles les LED sont branchées.
waitTime	Contient le temps d'attente entre les changements de LED (en ms).

Dans le sketch du séquenceur de lumière, vous rencontrez pour la première fois un tableau et une boucle. Cette dernière est nécessaire pour accéder confortablement aux différents éléments du tableau par le biais des numéros de broche. D'une part les broches sont toutes programmées en tant que sorties, et d'autre part les sorties numériques sont sélectionnées. L'accès à chaque élément se fait par un index et comme la boucle utilisée ici dessert automatiquement un certain domaine de valeurs, cette construction est idéale pour nous. Commençons par la variable de type array. La déclaration ressemble à celle d'une variable normale, à ceci près que le nom doit être suivi d'une paire de crochets.

Figure 5-2 ▶
Déclaration du tableau



- Le type de donnée définit quel type les différents éléments du tableau doivent avoir.
- Le nom du tableau est un nom évocateur pour accéder à la variable.
- Le nombre entre les crochets indique combien d'éléments le tableau doit contenir.

Vous pouvez imaginer un tableau comme un meuble à plusieurs tiroirs. Chaque tiroir est surmonté d'une étiquette portant un numéro d'ordre. Si je vous donne par exemple pour instruction d'ouvrir le tiroir numéro 3 et de regarder ce qu'il y a dedans, les choses sont plutôt claires non ? Il en va de même pour le tableau.

Index	0	1	2	3	4	5	6
Contenu du tableau	0	0	0	0	0	0	0

Tous les éléments de ce tableau ont été implicitement initialisés avec la valeur 0 après la déclaration. L'initialisation peut toutefois être faite de deux manières différentes. Nous avons choisi la manière facile et les valeurs, dont le tableau est censé être pourvu, sont énumérées derrière la déclaration entre deux accolades et séparées par des virgules :

```
int ledPin[] = {7, 8, 9, 10, 11, 12, 13};
```

Sur la base de cette ligne d'instruction, le contenu du tableau est le suivant.

Index	0	1	2	3	4	5	6
Contenu du tableau	7	8	9	10	11	12	13

N'avons-nous pas oublié quelque chose d'important ? Dans la déclaration du tableau, il n'y a rien entre les crochets. La taille du tableau devrait pourtant y être indiquée.

C'est vrai, mais le compilateur connaît déjà dans le cas présent – par les informations fournies pour l'initialisation faite dans la même ligne – le nombre d'éléments. Aussi la dimension du tableau n'a-t-elle pas besoin d'être indiquée. L'initialisation, quelque peu fastidieuse, consiste à affecter explicitement les différentes valeurs à chaque élément du tableau :

```
int ledPin[7];    //Déclaration du tableau avec 7 éléments
void setup()
{
  ledPin[0] = 7;
  ledPin[1] = 8;
  ledPin[2] = 9;
  ledPin[3] = 10;
  ledPin[4] = 11;
  ledPin[5] = 12;
  ledPin[6] = 13;
  // ...
}
```





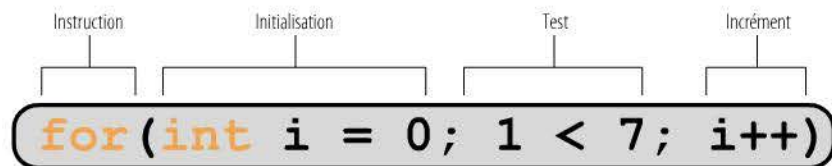
Attention !

L'index du premier élément du tableau est toujours le chiffre 0. Si, par exemple, vous déclarez un tableau de 10 éléments, l'index admis le plus élevé sera le chiffre 9 – soit toujours un de moins que le nombre d'éléments.

Si vous ne vous en tenez pas à cette règle, vous pouvez provoquer une erreur à l'exécution que le compilateur, qui se cache derrière l'environnement de développement, ne détecte ni au moment du développement ni plus tard pendant l'exécution, c'est pourquoi vous devez redoubler d'attention.

Venons-en maintenant à la boucle et regardons la syntaxe de plus près.

Figure 5-3 ►
Boucle for



La boucle introduite par le mot-clé `for` est appelée boucle `for`. Suivent entre parenthèses certaines informations sur les caractéristiques-clés.

- Initialisation : à partir de quelle valeur la boucle doit-elle commencer à compter ?
- Test : jusqu'à combien doit-elle compter ?
- Incrément : de combien la valeur initiale doit-elle être modifiée ?

Ces trois informations déterminent le comportement de la boucle `for` et définissent son comportement au moment de l'appel.



Pour aller plus loin

Une boucle `for` est utilisée la plupart du temps quand on connaît au départ le nombre de fois que certaines instructions doivent être exécutées. Ces caractéristiques sont définies dans ce que l'on appelle l'en-tête de boucle, qui correspond à ce qui est entre parenthèses.

Mais soyons plus concrets. La ligne de code suivante :

```
for(int i = 0; i < 7; i++)
```

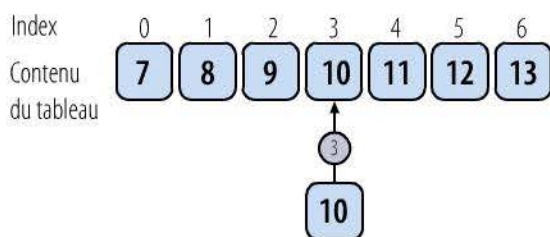
déclare et initialise une variable `i` du type `int` avec la valeur 0. L'indication du type de donnée dans la boucle stipule qu'il s'agit d'une variable locale qui n'existe que tant que la boucle `for` itère, c'est-à-dire suit son cours. La variable `i` est effacée de la mémoire à la sortie de la boucle.

Le nom exact d'une telle variable dans une boucle est « variable de contrôle ». Elle parcourt une certaine zone tant que la condition ($i < 7$) – désignée ici sous le nom de « test » – est remplie. Une mise à jour de la variable est ensuite effectuée selon l'expression de l'incrément. L'expression $i++$ ajoute la valeur 1 à la variable i .

Vous avez utilisé l'expression $i++$. Pouvez-vous m'expliquer exactement ce qu'elle signifie ? Elle doit augmenter la valeur de 1, mais son écriture est étrange.

Les signes $++$ sont un opérateur qui ajoute la valeur 1 au contenu de l'opérande, donc à la variable. Les programmeurs sont paresseux de naissance et font tout pour formuler au plus court ce qui doit être tapé. Quand on pense au nombre de lignes de code qu'un programmeur doit taper dans sa vie, moins il y a de caractères et mieux c'est. Il s'agit aussi à terme de consacrer plus de temps à des choses plus importantes – par exemple encore plus de code – en adoptant un mode d'écriture plus court. Toujours est-il que les deux expressions suivantes ont exactement le même effet : $i++$; et $i = i + 1$;

Deux caractères de moins ont été utilisés, ce qui représente tout de même une économie de 40 %. Mais revenons-en au texte. La variable de contrôle i sert ensuite de variable d'index dans le tableau et traite ainsi l'un après l'autre les différents éléments de ce tableau.



Sur cette capture d'écran d'une itération de la boucle, la variable i présente la valeur 3 et a donc accès au 4^e élément dont le contenu est 10. Autrement dit, toutes les broches consignées dans le tableau `ledPin` sont programmées en tant que sorties dans la fonction `setup` au moyen des deux lignes suivantes :

```
for(int i = 0; i < 7; i++)  
  pinMode(ledPin[i], OUTPUT);
```

Une chose importante encore : si, dans une boucle `for`, il n'y a aucun bloc d'instructions, formé au moyen d'accolades (comme nous en verrons un bientôt dans la fonction `loop`), seule la ligne venant immé-



diatement après la boucle `for` est prise en compte par cette dernière. Le code de la fonction `loop` contient seulement une boucle `for` dont la structure de bloc donne cependant accès à plusieurs instructions :

```
for(int i = 0; i < 7; i++)
{
    digitalWrite(ledPin[i], HIGH); //Élément de tableau au niveau HIGH
    delay(waitTime);
    digitalWrite(ledPin[i], LOW);  //Élément de tableau au niveau LOW
}
```

Je voudrais vous montrer dans un court sketch comment la variable de contrôle `i` est augmentée (incrémentée) :

```
void setup(){
    Serial.begin(9600);          //Configuration de l'interface série
    for(int i = 0; i < 7; i++)
        Serial.println(i);      //Impression sur l'interface série
}

void loop(){/* vide */}
```

Puisque notre Arduino n'a pas de fenêtre d'affichage, nous devons trouver autre chose. L'interface série sur laquelle il est quasiment branché peut nous servir à envoyer des données. L'environnement de développement dispose d'un Serial Monitor capable de recevoir et d'afficher ces données sans problème. Vous pouvez même l'utiliser pour envoyer des données à la carte Arduino. Vous en saurez plus bientôt. Le code initialise par l'instruction suivante :

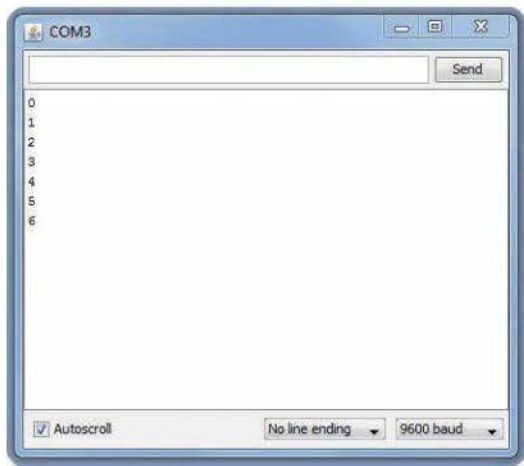
```
Serial.begin(9600);
```

l'interface série avec une vitesse de transmission de 9 600 bauds.

La ligne suivante :

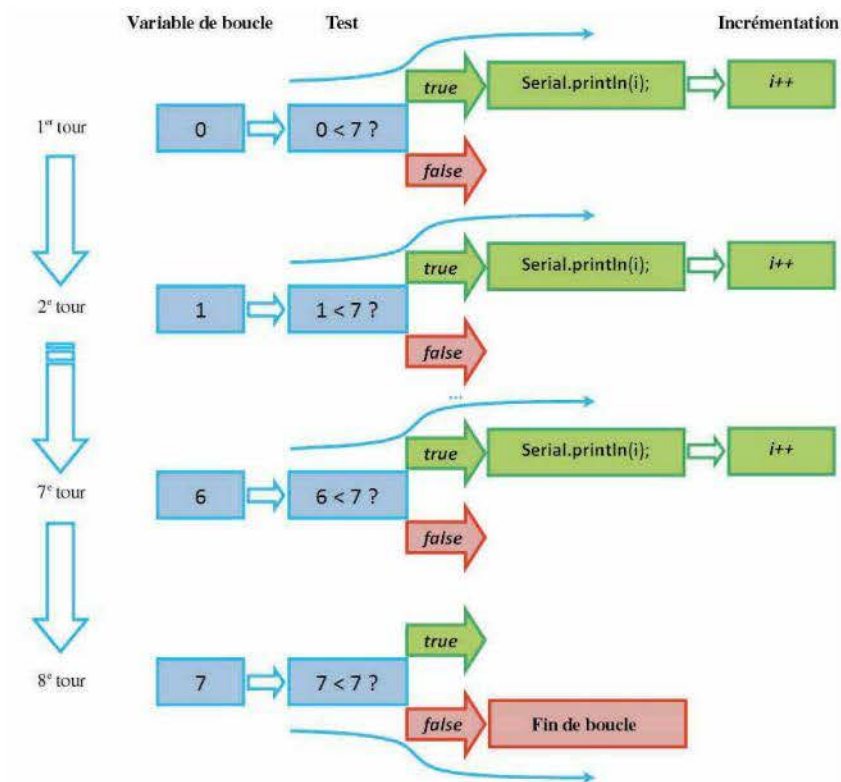
```
Serial.println(i);
```

envoie ensuite au moyen de la fonction `println` la valeur de la variable `i` à l'interface. Il ne vous reste plus qu'à ouvrir le Serial Monitor pour afficher les valeurs de la figure 5-4.



◀ **Figure 5-4**
Impression des valeurs dans le
Serial Monitor

On voit ici comment les valeurs de la variable de contrôle `i`, dont nous avons besoin dans notre sketch pour sélectionner les éléments du tableau, sont imprimées de 0 à 6. J'ai placé le code dans la fonction `setup` pour que la boucle `for` ne soit exécutée qu'une fois et ne s'affiche pas constamment. La figure 5-5 montre de plus près les différents passages de la boucle `for`.



◀ **Figure 5-5**
Comportement de la boucle `for`



Eh là, pas si vite ! Le code de programmation de l'interface série c'est du chinois pour moi. On y trouve `Serial`, `begin` ou encore `println` avec un point entre les deux. Qu'est-ce que ça veut dire ?

Vous aimez bien tout comprendre et ce n'est pas pour me déplaire ! Très bien ! Il me faut maintenant parler de la programmation orientée objet, car elle va me servir à vous expliquer la syntaxe. Nous reviendrons plus tard sur ce mode de programmation puisque C++ est un langage orienté objet (ou OOP sous sa forme abrégée). Ce langage est tourné vers la réalité constituée d'objets réels tels que par exemple table, lampe, ordinateur, barre de céréales, etc. Aussi les programmeurs ont-ils défini un « objet » représentant l'interface série. Ils ont donné à cet objet le nom de `Serial`, et il est utilisé à l'intérieur d'un sketch. Chaque objet possède cependant d'une part certaines caractéristiques (telles que la couleur ou la taille) et d'autre part un ou plusieurs comportements qui définissent ce qu'on peut faire avec cet objet. Dans le cas d'une lampe, le comportement serait par exemple le fait de s'allumer ou de s'éteindre. Mais revenons à notre objet `Serial`. Le comportement de cet objet est géré par de nombreuses fonctions qui sont appelées méthodes en programmation orientée objet (OOP). Deux de ces méthodes vous sont déjà familières : la méthode `begin` qui initialise l'objet `Serial` avec le taux de transmission voulu, et la méthode `println` (*print line* signifie en quelque sorte imprimer et faire un saut de ligne) qui envoie quelque chose sur l'interface série. Le lien entre objet et méthode est assuré par l'opérateur point (.) qui les relie ensemble. Quand je dis par conséquent que `setup` et `loop` sont des fonctions, ce n'est qu'une demi-vérité car il s'agit, à bien y regarder, de méthodes.

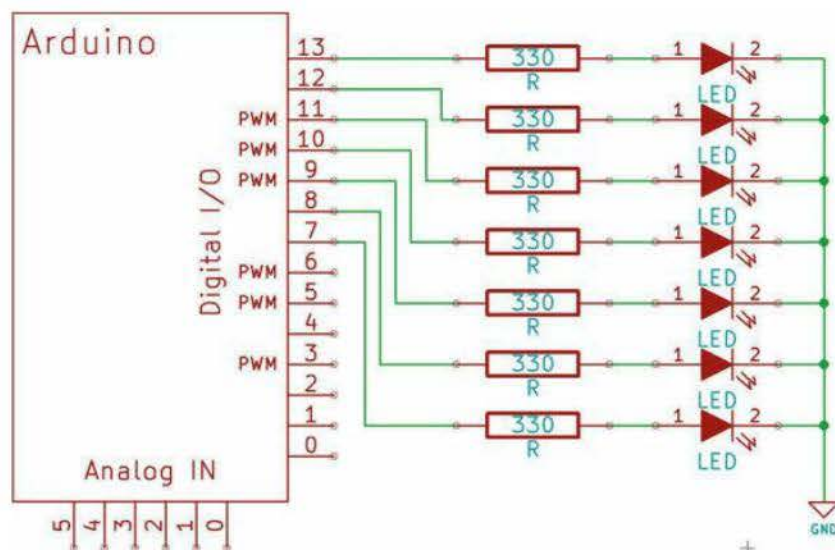


Pour aller plus loin

Vous savez maintenant comment envoyer quelque chose à l'interface série. Vous pouvez vous en servir pour trouver une ou plusieurs erreurs dans un sketch. Si le sketch ne fonctionne pas comme prévu, placez des instructions d'écriture sous forme de `Serial.println()` à divers endroits qui vous paraissent importants dans le code et imprimez certains contenus de variable ou encore des textes. Vous pouvez ainsi savoir ce qui se passe dans votre sketch et pourquoi il ne marche pas bien. Vous devez seulement apprendre à interpréter les données imprimées. Ce n'est pas toujours facile et il faut un peu d'entraînement.

Schéma

Le schéma montre les différentes LED avec leur résistance série de 330 ohms.

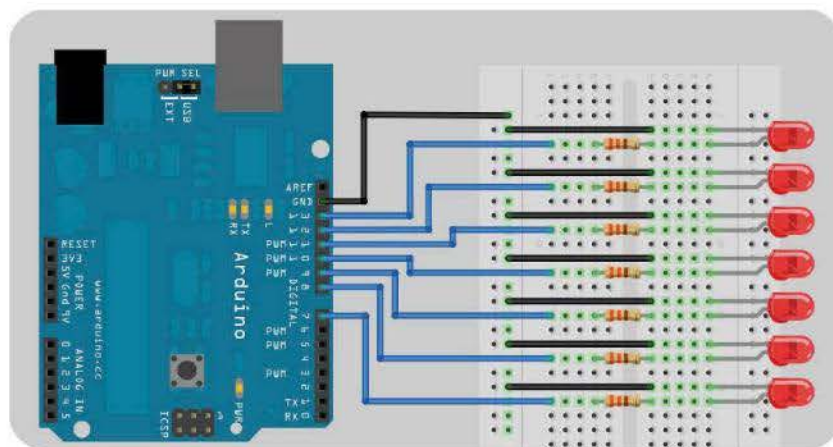


◀ **Figure 5-6**

Carte Arduino commandant 7 LED pour un séquenceur de lumière

Réalisation du circuit

Votre plaque d'essais accueille toujours plus de composants électroniques sous forme de résistances et diodes électroluminescentes.



◀ **Figure 5-7**

Réalisation du circuit de séquenceur de lumière avec Fritzing



Attention !

Quand vous branchez des composants électroniques tout près les uns des autres, comme c'est ici le cas, soyez très attentif car il arrive souvent de se tromper et d'occuper le trou voisin sur la plaque, si bien que le circuit ne fonctionne qu'en partie, voire pas du tout. Cela devient sérieux si vous travaillez avec les lignes d'alimentation et de masse placées l'une à côté de l'autre. Des problèmes peuvent aussi résulter de cavaliers flexibles mal enfoncés dans leur trou, dont les fils conducteurs dénudés ressortent en partie. Des courts-circuits peuvent se produire quand on bouge ces cavaliers, lesquels peuvent tout abîmer. Il faut donc se monter soigneux.

Problèmes courants

Si les LED ne s'allument pas l'une après l'autre, débranchez le port USB de la carte pour plus de sécurité et vérifiez ce qui suit.

- Vos fiches de raccordement sur la plaque correspondent-elles vraiment au circuit ?
- Pas de court-circuit éventuel entre elles ?
- Les LED ont-elles été mises dans le bon sens ? Autrement dit, la polarité est-elle correcte ?
- Les résistances ont-elles bien les bonnes valeurs ?
- Le code du sketch est-il correct ?

Qu'avez-vous appris ?

- Vous avez fait la connaissance d'une forme spéciale de variable vous permettant d'enregistrer plusieurs valeurs d'un même type de donnée. Elle est appelée tableau (`array`). On accède à ses différents éléments au moyen d'un index.
- La boucle `for` vous permet d'exécuter plusieurs fois une ou plusieurs lignes de code. Elle est gérée par une variable de contrôle, active dans la boucle et initialisée avec une certaine valeur initiale. Une condition vous a permis de définir pendant combien de temps la boucle doit s'exécuter. Vous contrôlez ainsi quel domaine de valeurs la variable traite.
- Vous pouvez réunir plusieurs instructions, qui sont ensuite toutes exécutées par exemple dans le cas d'une boucle `for`, en constituant un bloc au moyen de la paire d'accolades.
- La variable de contrôle, dont nous venons de parler, est utilisée pour modifier l'index d'un tableau et accéder ainsi à ses différents éléments.

Exercice complémentaire

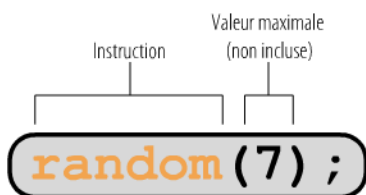
Dans cet exercice, je souhaiterais que vous fassiez clignoter le séquenceur de lumière de différentes manières. Je vous propose :

- toujours dans le même sens, une LED s'allumant à tour de rôle (c'est le montage que vous venez de voir) ;
- dans un sens puis dans l'autre, une ou plusieurs LED s'allumant à tour de rôle ;
- dans les deux sens en même temps (la LED 1 s'allumant en même temps que la LED 7 au premier tour, puis la LED 2 s'allumant au même moment que la LED 6 au deuxième tour et ainsi de suite) ;
- à chaque tour, une LED s'allume de manière aléatoire.

Pour commander une LED au hasard, vous aurez besoin d'une autre fonction que vous ne connaissez pas encore. Elle se nomme `random`, ce qui signifie « aléatoire » ou « au hasard ». Il existe deux syntaxes possibles pour cette fonction.

1^{re} syntaxe

Vous utiliserez la syntaxe suivante pour générer une valeur au hasard dans un domaine compris entre 0 et une limite supérieure :



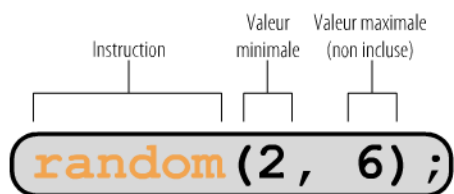
◀ **Figure 5-8**
Instruction `random`
(avec un argument)

Attention, la valeur maximale que vous indiquez sera toujours non incluse. Dans cet exemple, vous générerez ainsi des chiffres au hasard entre 0 et 6 inclus.

2^e syntaxe

Vous utiliserez la syntaxe suivante pour générer une valeur au hasard comprise entre une limite inférieure et une limite supérieure.

Figure 5-9 ►
Instruction `random`
(avec deux arguments)



Cette instruction générera des valeurs comprises entre 2 et 5 inclus, la valeur la plus élevée étant ici aussi exclue. Cette particularité pourra surprendre certains, mais il n'est pas possible de faire autrement.

Extension de port

Au sommaire :

- la déclaration et l'initialisation de plusieurs variables ;
- la programmation de plusieurs broches comme sorties (OUTPUT) ;
- le registre à décalage de type 74HC595 avec 8 sorties ;
- la commande du registre à décalage par trois lignes de la carte Arduino ;
- la définition d'une fonction particulière ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- les autres sketches ;
- l'instruction `shiftOut` ;
- un exercice complémentaire.

Le registre à décalage

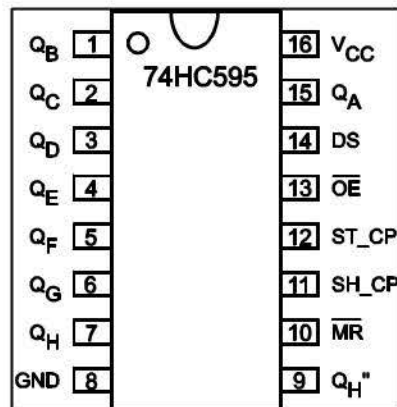
Nous avons vu dans le montage précédent comment programmer la commande des multiples LED d'un séquenceur de lumière. Votre carte Arduino ne disposant que d'un nombre limité de sorties numériques, ces précieuses ressources pourraient finir par vous manquer pour ajouter d'autres LED à votre séquenceur de lumière. Par ailleurs, vous voulez peut-être aussi connecter quelques capteurs sur des entrées numériques.

Vous aurez donc encore moins de broches numériques à disposition. Comment résoudre ce problème ? Il existe plusieurs solutions, en voici une. Je dois utiliser pour cela un registre à décalage. Vous vous demandez certainement ce que c'est et comment il opère. Dans cette expérimentation, un circuit intégré (IC pour *Integrated Circuit*) sera relié pour la première fois à votre carte Arduino. Un registre à décalage est un circuit géré par un signal d'horloge et doté de plusieurs sorties disposées l'une derrière l'autre. À chaque période d'horloge, le niveau présent à l'entrée du registre est transmis à la sortie suivante. Cette information passe ainsi par toutes les sorties existantes.



Le circuit intégré 74HC595, que nous utilisons ici, dispose d'une entrée série par laquelle les données sont entrées et de huit sorties équipées de registres mémoire internes pour conserver les états. Seules trois broches numériques sont nécessaires à l'alimentation, lesquelles fournissent des données au module qui, de son côté, commande ses huit sorties. C'est en soi une économie majeure car le circuit 74HC595 peut être cascader, permettant ainsi une expansion quasi illimitée des sorties numériques. De quoi s'agit-il exactement ? Voyons de plus près les différentes entrées et sorties de ce circuit. La figure 6-1 illustre le brochage du circuit, vu de dessus.

Figure 6-1 ►
Brochage du registre à décalage
74HC595



Le tableau 6-1 récapitule les différentes broches et leur signification.

Broche	Signification
V _{cc}	Tension d'alimentation
GND	Masse 0 V
Q _A -Q _H	Sorties parallèles 1 à 8
Q _H "	Sortie série (entrée pour un deuxième registre à décalage)
MR	Master Reset (actif LOW)
SH_CP	Registre à décalage, entrée d'horloge (Shiftregister clock input)
ST_CP	Registre mémoire, entrée d'horloge (Storageregister clock input)
OE	Activation de la sortie (Output enable/actif LOW)
DS	Entrée série (Serial data input)

◀ **Tableau 6-1**

Signification des broches
du registre à décalage 74HC595

Le mode de fonctionnement du registre à décalage peut se résumer ainsi : quand le niveau à l'entrée d'horloge SH_CP passe de LOW à HIGH, le niveau à l'entrée série DS est lu, transmis à l'un des registres internes et enregistré temporairement. Mais cela ne signifie pas pour autant transmis aux sorties Q_A à Q_H. C'est seulement une impulsion d'horloge à l'entrée ST_CP de LOW à HIGH qui fait transmettre toutes les informations des registres internes aux sorties. C'est utile car ce n'est que quand toutes les informations ont été lues à l'entrée série qu'elles sont censées être détectées aux sorties. Le changement du niveau logique de LOW à HIGH est appelé contrôle par front montant d'horloge, car une action n'est entreprise que quand un changement de niveau se produit de la manière décrite.

Voyons maintenant un peu ce qui se passe dans le registre à décalage...

Voici justement SH_CP au travail. Quand il tourne la pancarte de LOW à HIGH, le candidat potentiel, qui se trouve dans la zone DS-area, passe dans le registre suivant et attend la suite de son voyage vers la sortie.



◀ **Figure 6-2**

SH_CP préparant les données série

La figure 6-3 montre ST_CP en train de faire partir les données des registres internes vers les sorties.

Figure 6-3 ►
ST_CP autorisant le départ
des données des registres
vers les sorties



Quand il tourne la pancarte de LOW à HIGH, les portes des registres internes s'ouvrent et alors seulement les données peuvent trouver le chemin de la sortie. Le procédé croqué ici sera reproduit dans plusieurs sketches pour que vous puissiez voir en direct comment marche le registre à décalage. Nous allons tout faire de A à Z, mais vous verrez à la fin qu'il existe une instruction bien commode pour toutes les actions à entreprendre l'une derrière l'autre, qui vous épargnera bien du travail et vous facilitera les choses.

Composants nécessaires



1 registre à décalage 74HC595



8 LED rouges



8 résistances de 330 Ω



1 résistance de 10 k Ω



1 bouton-poussoir



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Voici le code du sketch pour commander le registre à décalage 74HC595 au moyen de trois lignes de sorties numériques. Les broches suivantes sont nécessaires sur le registre :

- SH_CP (registre à décalage, entrée d'horloge) ;
- ST_CP (registre mémoire, entrée d'horloge) ;
- DS (entrée série pour les données).

Des variables sont affectées aux trois lignes de données, variables auxquelles j'ai donné les noms suivants :

- SH_CP est shiftPin ;
- ST_CP est storagePin ;
- DS est dataPin.

Ce sketch met l'entrée série DS sur HIGH, niveau qui est ensuite transféré dans le registre interne quand l'entrée d'horloge SH_CP du registre à décalage passe de LOW à HIGH. Les sorties sont alors programmées et enregistrées via les registres internes au moyen de l'entrée d'horloge ST_CP du registre mémoire.

```
int shiftPin = 8;      //SH_CP
int storagePin = 9;    //ST_CP
int dataPin = 10;      //DS
void setup(){
  pinMode(shiftPin, OUTPUT);
  pinMode(storagePin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  resetPins();         //Mise de toutes les broches sur LOW
  //Mise de DS sur HIGH pour reprise ultérieure par SH_CP
  digitalWrite(dataPin, HIGH); //DS
  delay(20);           //Brève pause avant traitement
  //Transmission du niveau à DS dans registres mémoire internes
  digitalWrite(shiftPin, HIGH); //SH_CP
  delay(20);           //Brève pause avant traitement
  //Transmission des registres mémoire internes aux sorties
```

```

    digitalWrite(storagePin, HIGH); //ST_CP
    delay(20);
}

void loop(){/* vide */}

//Réinitialisation de toutes les broches → niveau LOW
void resetPins(){
    digitalWrite(shiftPin, LOW);
    digitalWrite(storagePin, LOW);
    digitalWrite(dataPin, LOW);
}

```

Revue de code

Les variables suivantes sont techniquement nécessaires à notre programmation.

Tableau 6-2 ▶
Variables nécessaires et leur objet

Variable	Objet
shiftPin	N° broche SH_CP
storagePin	N° broche ST_CP
dataPin	N° broche DS

Les variables sont d'abord pourvues des informations de broche nécessaires puis toutes les broches sont programmées en tant que sorties au début de la fonction `setup`. Vous rencontrez pour la première fois dans ce montage une fonction écrite par vous-même. Une fonction n'a en soi rien de nouveau pour vous puisque `setup` et `loop` font déjà partie de cette catégorie de structures logicielles. Je souhaite cependant revenir sur le sujet pour en préciser le sens et l'objet. Une fonction peut être considérée comme une sorte de sous-programme qui peut toujours être appelé dans le déroulement normal d'un sketch. Elle est invoquée par son nom et peut tout aussi bien retourner une valeur à l'appelant qu'enregistrer plusieurs valeurs transférées nécessaires au calcul ou au traitement. La structure formelle d'une fonction est la suivante :

Figure 6-4 ▶
Structure de base d'une fonction

```

Type de donnée de retour Nom (paramètre)
{
    //Une ou plusieurs instructions
}

```

La partie entourée est appelée signature de la fonction et représente l'interface formelle avec la fonction. Cette dernière est comparable à une

black box, que vous connaissez déjà. En fait, vous n'avez pas besoin de savoir comment elle fonctionne. Il vous suffit de connaître la structure de l'interface et de savoir sous quelle forme une valeur est retournée le cas échéant. Vous programmez ici bien entendu la fonction elle-même et devez pour le moins connaître la logique qu'elle renferme. Certaines fonctions peuvent également être obtenues par exemple sur Internet, dans la mesure où leur usage n'est pas limité techniquement par une licence, et utilisées dans votre montage. Peu importe de savoir comment elles fonctionnent du moment qu'elles ont été programmées et testées avec succès par d'autres. Le principal est qu'elles fonctionnent ! Mais revenons à notre définition de la fonction. Si elle renvoie une valeur en retour à l'appelant, comme le fait par exemple `digitalRead`, vous devez indiquer le type de donnée en question dans votre fonction.

Supposons que vous vouliez retourner des valeurs qui sont toutes des nombres entiers, le type de donnée est alors `Integer` défini par le mot-clé `int`. Si toutefois aucun retour n'est requis, vous devez le faire savoir par le mot-clé `void` (traduction : *vide*) qui précède déjà les deux fonctions principales `setup` et `loop`.

Excusez-moi mais j'ai une question. Vous avez dit que les fonctions sont toujours invoquées par leur nom. Mais qu'en est-il des deux fonctions `setup` et `loop` ? Pas besoin de stipuler quelque part dans le code qu'elles doivent être appelées et pourtant ça marche. Comment est-ce possible ?

Cette question est pertinente, pourtant ce comportement n'étonne personne la plupart du temps. `setup` et `loop` sont des fonctions systématiques qui sont appelées implicitement. Comme vous l'avez remarqué, vous n'avez pas besoin de vous en occuper.



Pour aller plus loin

Si cela vous intéresse, vous trouverez dans le répertoire d'installation sous `arduino-1.x.y\hardware\arduino\cores\arduino` le fichier `main.cpp` que vous pourrez ouvrir avec un éditeur de texte. Vous verrez alors ce qui suit :


```

1  #define ARDUINO_MAIN
2  #include <Arduino.h>
3
4  int main(void)
5  {
6      init();
7
8      #if defined(USBCON)
9          USB.attach();
10     #endif
11
12     setup();
13
14     for (;;) {
15         loop();
16         if (serialEventRun) serialEventRun();
17     }
18
19     return 0;
20 }

```

La fonction directement appelée en début de programme avec C++ est nommée `main`, comme c'est le cas ici. Elle sert quasiment de point d'accès, pour que le programme sache par quoi il doit commencer. `main` contient plusieurs appels de fonction qui sont traités l'un après l'autre. On y trouve entre autres la fonction `setup` et l'appel de la fonction `loop` dans une boucle sans fin définie par `for(;;)`. Vous reconnaissez certainement les déroulements ou plutôt les relations qui se créent en coulisses au début d'un sketch quand il s'agit d'appeler `setup` ou `loop`.

Si une ou plusieurs variables sont à fournir à votre fonction, celles-ci sont indiquées entre parenthèses derrière son nom, séparées par des virgules, avec leur type de donnée correspondant. Les parenthèses sont nécessaires même s'il n'y a rien entre elles faute de variables. La signature est suivie du corps de fonction, formé par la paire d'accolades. Toutes les instructions, qui se trouvent entre ces deux accolades, font partie de la fonction et sont traitées séquentiellement de haut en bas lors de l'appel. Mais revenons au code. En quoi est-ce utile d'écrire une fonction particulière ? Très simple ! Ça l'est toujours quand les mêmes instructions sont à exécuter plusieurs fois dans le code et c'est ici le cas. Je dois exécuter la suite d'instructions qui suit à divers endroits pour réinitialiser – autrement dit, pour remettre sur `LOW` – les niveaux sur les différentes broches numériques. Sans fonction, le sketch compterait un grand nombre de lignes de code en plus et manquerait donc de clarté.

```

digitalWrite(shiftPin, LOW);
digitalWrite(storagePin, LOW);
digitalWrite(dataPin, LOW);

```



Pour aller plus loin

Le code source, qui revient plusieurs fois avec la même séquence d'instructions dans le sketch, est appelé code redondant ou redondance de code. Le mieux est de le stocker dans une fonction à laquelle vous donnez un nom suffisamment évocateur pour en saisir le sens. Si vous devez procéder à une modification, vous intervenez de manière centrale au sein de la fonction et non pas un peu partout dans le code, ce qui est générateur de bien des erreurs et très chronophage.

Au début du sketch, l'appel de fonction :

```
resetPin() ; //Mettre toutes les broches sur LOW
```

permet de mettre les broches 8, 9 et 10 au niveau LOW.

Le premier signal de niveau HIGH est ensuite appliqué à DS par la ligne :

```
digitalWrite(dataPin, HIGH); //DS
```

Puis une attente de 20 ms s'écoule avant que la ligne suivante ne transmette le niveau HIGH de DS au registre mémoire interne :

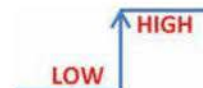
```
digitalWrite(shiftPin, HIGH); //SH_CP
```

Il faut ici tenir compte du fait que ce n'est possible qu'au moyen d'un contrôle par front montant de LOW vers HIGH.

Il n'y a pas encore de transfert en direction du port de sortie. Une nouvelle attente de 20 ms s'écoule, et enfin la ligne suivante déclenche la transmission des registres mémoire internes aux sorties, ce qui revient à commander les LED dans le cas présent :

```
digitalWrite(storagePin, HIGH); //ST_CP
```

Un changement de niveau de LOW à HIGH est ici aussi nécessaire, d'où le recours à la fonction `resetPin` qui permettra plus tard de changer encore le niveau de LOW à HIGH.



Schéma

Le schéma montre les différentes LED avec leurs résistances série de 330 ohms, qui sont commandées par le registre à décalage 74HC595. L'entrée *master reset* de la puce est connectée, à travers la résistance *pull-up*, à la tension d'alimentation +5 V, si bien que le reset ne se déclenche pas tant que le bouton-poussoir n'est pas enfoncé puisque l'entrée MR est active au niveau LOW. On notera la présence d'un trait horizontal au-dessus de MR, qui correspond à une négation. L'entrée

output enabled est également active au niveau LOW et reliée par un fil à la masse car les sorties doivent être toujours actives. Le registre à décalage est commandé par les broches Arduino 8, 9 et 10 avec les fonctions décrites plus haut.

Si vous lancez le sketch, la première LED s'allume immédiatement sur la sortie Q_A car vous avez entré une seule fois 1 dans le registre à décalage. Vous devez actionner non seulement le bouton-poussoir du circuit mais aussi le bouton de reset de la carte Arduino pour effectuer un reset (réinitialisation).

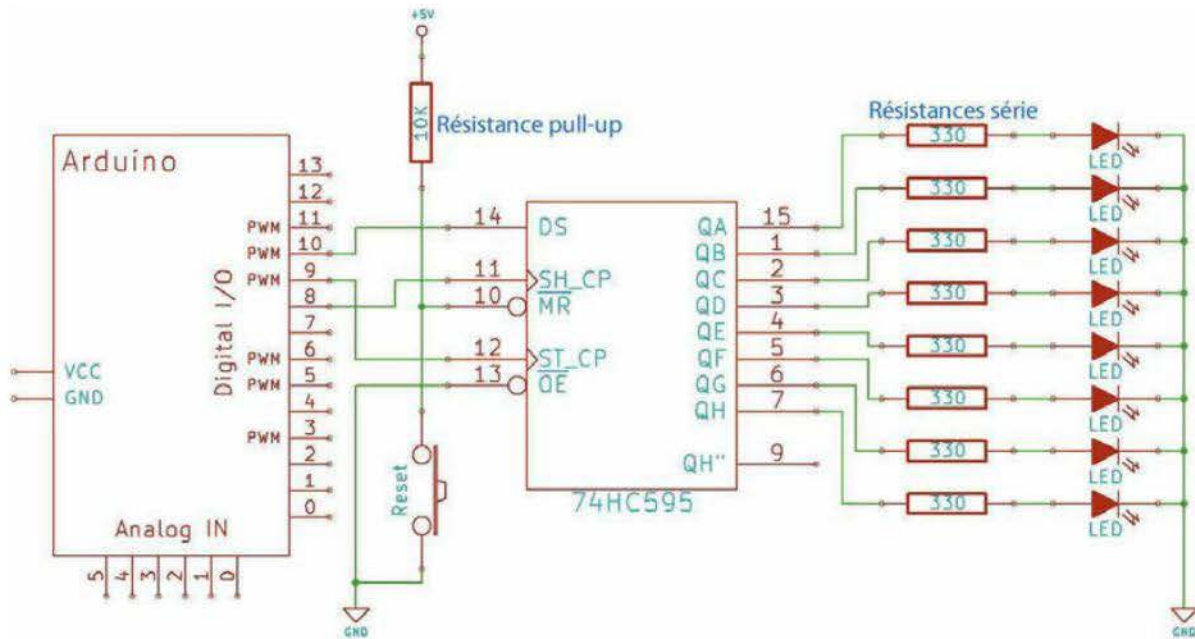
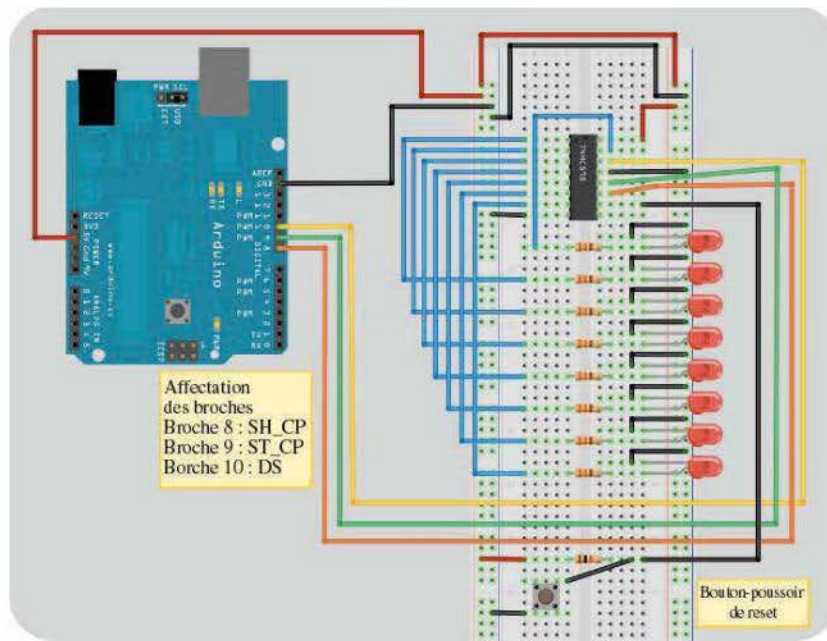


Figure 6-5 ▲
Carte Arduino commandant
le registre à décalage 74HC595
par trois lignes de signaux

Réalisation du circuit

Votre plaque d'essais se remplit et les choses deviennent intéressantes, n'est-ce pas ?



◀ **Figure 6-6**
Réalisation du circuit avec Fritzing

Extension du sketch : première partie

Complétons maintenant un peu le sketch de telle sorte que vous puissiez rentrer plusieurs valeurs dans l'entrée série. Ce n'est encore qu'un degré intermédiaire et non pas la solution finale que je souhaite vous présenter. Ce code doit transmettre au registre à décalage une séquence stockée dans un tableau de données. La construction du circuit reste la même.

```
int shiftPin = 8;           //SH_CP
int storagePin = 9;         //ST_CP
int dataPin = 10;           //DS
int dataArray[] = {1, 0, 1, 0, 1, 1, 0, 1};
void setup(){
  pinMode(shiftPin, OUTPUT);
  pinMode(storagePin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  resetPins();              //Mettre toutes les broches à LOW
  putPins(dataArray);       //Régler les broches sur le tableau
                             //de données
  //Transmission des registres mémoire internes aux sorties
  digitalWrite(storagePin, HIGH); //ST_CP
}
```



```

void loop() { /* vide */}

void resetPins(){
  digitalWrite(shiftPin, LOW);
  digitalWrite(storagePin, LOW);
  digitalWrite(dataPin, LOW);
}

void putPins(int data[]){
  for(int i = 0; i < 8; i++){
    resetPins();
    digitalWrite(dataPin, data[i]); delay (20);
    digitalWrite(shiftPin, HIGH); delay (20);
  }
}

```

Voyons maintenant comment le code accomplit son travail. Tout tourne autour du tableau de données qui contient le modèle de commande des différentes LED. Il s'agit donc de la ligne de déclaration et initialisation suivante :

```
int dataArray[] = {1, 0, 1, 0, 1, 1, 0, 1};
```

Le code lit les éléments du tableau de gauche à droite et rentre les valeurs dans le registre à décalage. Un 1 signifie une LED allumée et un 0 une LED éteinte.



Un moment s'il vous plaît. Vous avez utilisé les valeurs 1 et 0 pour commander les LED. Ne vaut-il pas mieux travailler avec les noms de constante HIGH et LOW ?

J'ai préféré utiliser les valeurs 1 et 0 parce que ce sont elles précisément qui se cachent derrière les constantes HIGH et LOW. Normalement, je ne suis pas pour les *magic numbers* (voir page 224) mais il m'a semblé que dans ce cas, je pouvais faire une exception. 1 et 0 étant aussi les valeurs logiques, vous ne devriez pas avoir trop de mal à comprendre ! Vous pouvez bien entendu écrire à la place de :

```
int dataArray[] = {1, 0, 1, 0, 1, 1, 0, 1};
```

la ligne suivante :

```
int dataArray[] = {HIGH, LOW, HIGH, LOW, HIGH, HIGH, LOW, HIGH};
```

Mais revenons au code et à sa manière d'exploiter le tableau. La chose n'est pas si simple car j'ai ajouté une fonction nommée `putPins`, qui a pour tâche de remplir le registre à décalage. Elle présente un

paramètre de transmission capable d'enregistrer non pas une variable normale mais tout un tableau de données. Il suffit simplement de transmettre le tableau de données comme argument dans la fonction :

```
putPins(dataArray);
```

La fonction est définie comme suit :

```
void putPins(int data[]){  
  for(int i = 0; i < 8; i++){  
    resetPins(); //Remise à zéro des broches et préparation  
                //à la commande par front montant  
    digitalWrite(dataPin, data[i]); delay(20);  
    digitalWrite(shiftPin, HIGH); delay(20);  
  }  
}
```

On peut voir qu'un tableau du type de donnée `int` a été déclaré avec deux crochets dans la signature de la fonction. Au moment où la fonction est appelée, le tableau initial `dataArray` est copié dans `data`, qui est ensuite exploité dans la fonction.

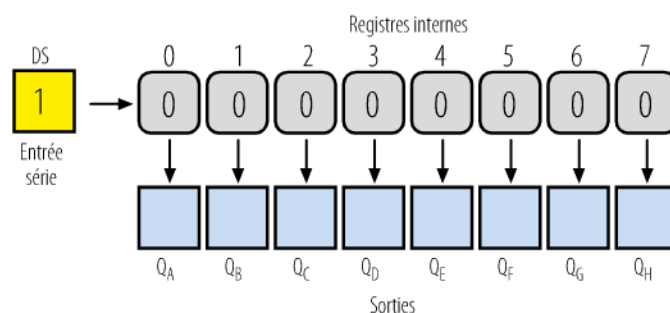
Puis chaque élément du tableau est envoyé dans l'entrée série, au moyen de la boucle `for` (que vous connaissez déjà), via :

```
digitalWrite(dataPin, data[i]);
```

et placé lors de l'étape suivante dans le premier registre interne via :

```
digitalWrite(shiftPin, HIGH);
```

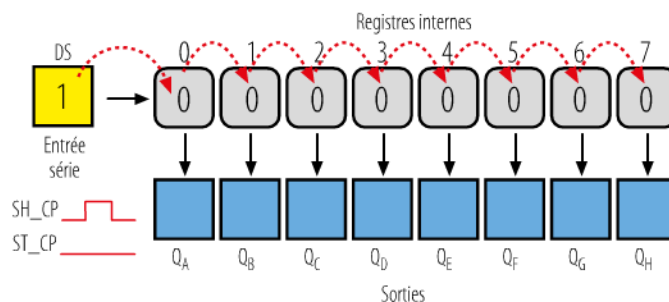
Le tout s'effectue huit fois (de 0 à 7), chaque registre interne transmettant ses valeurs au suivant. Les figures suivantes montrent les choses encore plus clairement.



◀ **Figure 6-7**
Registre à décalage

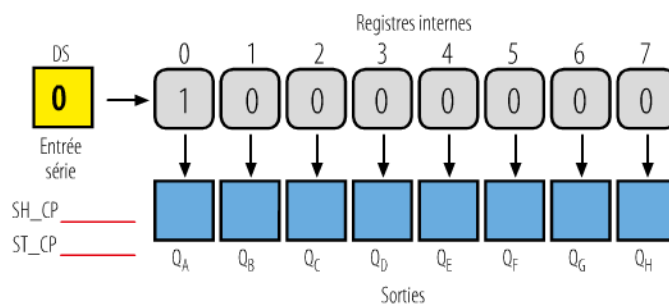
Au début, les registres sont encore tous vides. Un 1 attend toutefois déjà à l'entrée d'être transféré dans le premier registre interne.

Figure 6-8 ►
Registre à décalage pendant
le premier temps SH_CP



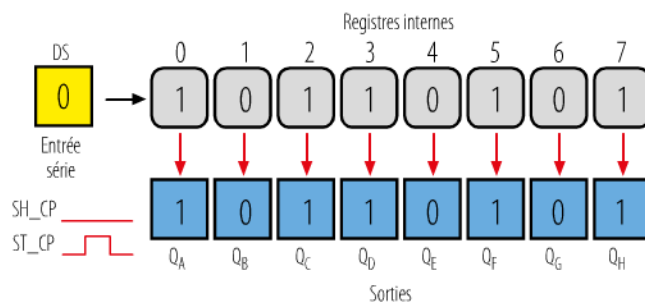
Le 1 qui se trouve à l'entrée série est entré, pendant le front montant de SH_CP, dans le premier registre interne. Les contenus de tous les registres sont décalés d'une position vers la droite. Cette action donne les états illustrés à la figure 6-9.

Figure 6-9 ►
États du registre à décalage
après la première impulsion
de SH_CP



À l'entrée se trouve maintenant un 0 qui sera lui aussi entré à la prochaine impulsion de SH_CP dans le premier registre interne. Mais avant, l'état du septième registre interne sera passé au huitième, le sixième au septième, etc., et enfin le 0 est entré dans le premier registre. Passons directement au moment où toutes les valeurs du tableau ont été entrées, conformément au schéma ci-dessus, dans les registres internes et où l'impulsion ST_CP a recopié les registres vers les sorties.

Figure 6-10 ►
États du registre à décalage
après lecture des valeurs du tableau
et après l'impulsion ST_CP



Les valeurs du tableau lu sont appliquées aux sorties seulement maintenant, la première valeur entrée se trouvant complètement à droite et la dernière complètement à gauche.

Comment fait-on pour inverser le comportement ? Je voudrais maintenant que la première valeur du tableau se trouve complètement à gauche et que la dernière complètement à droite se trouve à la sortie, de telle sorte que l'ordre soit quasiment inversé.



Pas de problème car où les broches ont-elles été déterminées ? Exact, dans la fonction `putPins` ! C'est la boucle `for` qui fixe l'ordre des différentes broches. Celui-ci sera inversé si vous appelez la dernière valeur à la place de la première et la transmettez au registre à décalage. Voici le code modifié de la boucle `for` :

```
for(int i = 7; i >= 0; i--){
    // ...
}
```

Extension du sketch : deuxième partie

Maintenant que vous en savez assez sur le registre à décalage 74HC595, je voudrais vous présenter une instruction spéciale qui vous épargnera du travail. `shiftOut` est vraiment facile à utiliser. Mais je dois auparavant vous livrer quelques informations sur la sauvegarde des données dans l'ordinateur, informations qui sont vraiment importantes pour comprendre le fonctionnement d'un microcontrôleur. Je me sers pour mes réalisations du type de donnée `byte`, dont la taille est de 8 bits et qui peut stocker des valeurs comprises entre 0 et 255. La figure 6-11 montre la valeur décimale 157 sous sa forme binaire 10011101.

Puissances	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valeurs	128	64	32	16	8	4	2	1
Combinaison de bits	0	0	0	1	1	1	0	1

Figure 6-11
Combinaison binaire
pour le nombre entier 157

Si on regarde les puissances de plus près, on voit que la base est le chiffre 2. Nous autres humains comptons en base 10 en raison des dix doigts de nos mains. Les valeurs des différents chiffres d'un nombre sont donc 10^0 , 10^1 , 10^2 , etc. Pour le nombre 157, cela donne $7 \times 10^0 + 5 \times 10^1 + 1 \times 10^2$ qui font bien sûr 157. Le microcontrôleur ne

pouvant cependant stocker que deux états (HIGH et LOW), le système binaire (du latin *binarius*, deux chacun) est fondé sur la base 2. La valeur décimale de ladite combinaison binaire se calcule par conséquent comme suit, en commençant en principe par la valeur – ou bit – la plus petite :

$$1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 + 1 \times 2^6 = 157_{10}$$

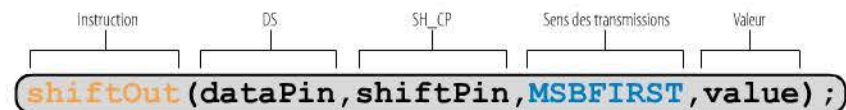


Pour aller plus loin

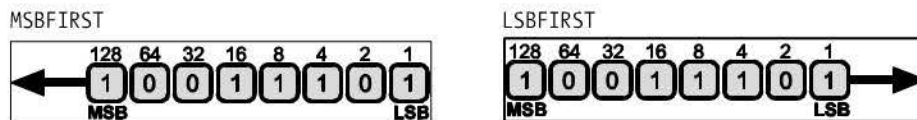
La base figure derrière le nombre pour plus de clarté quand différents systèmes de numération sont utilisés.

Avec un nombre de 8 bits (ou 1 octet), vous pouvez représenter 256 valeurs distinctes (de 0 à 255). Ceci dit, revenons à l'instruction `shiftOut`, qui possède différents paramètres dont nous allons faire le tour.

Figure 6-12 ▶
Instruction `shiftOut`
avec ses nombreux arguments



Les arguments `dataPin`, `shiftPin` ou encore la valeur à transmettre sont censés être clairs. Mais que signifie la constante `MSBFIRST` ? Cet argument permet de définir le sens de transmission des bits. Dans le cas d'un octet, le bit de poids fort est nommé *Most Significant Bit* (MSB) et le bit de poids faible *Least Significant Bit* (LSB). Vous pouvez aussi définir quel bit doit être transféré en premier dans le registre à décalage.



Voici le code complet avec l'instruction `shiftOut`. Le circuit n'a pas non plus besoin ici d'être modifié.

```
int shiftPin = 8;    //SH_CP
int storagePin = 9;  //ST_CP
int dataPin = 10;    //DS
byte value = 157;    //Valeur à transférer
void setup(){
  pinMode(shiftPin, OUTPUT);
  pinMode(storagePin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop(){
```

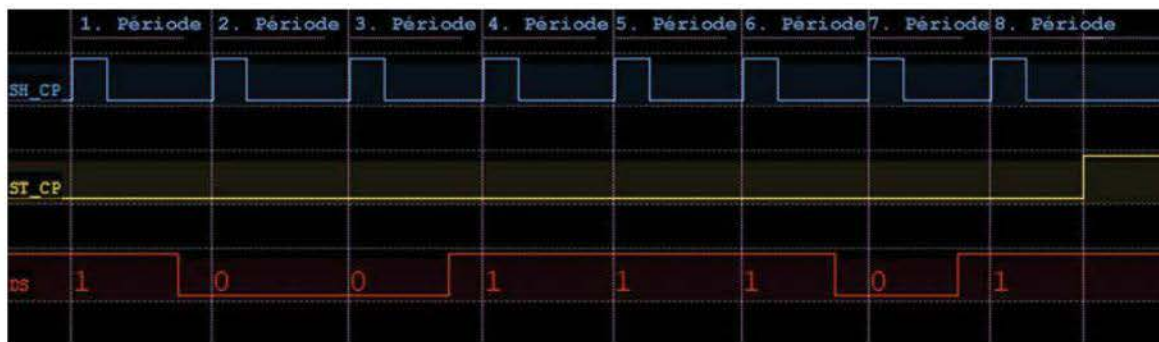
```
digitalWrite(storagePin, LOW);
shiftOut(dataPin, shiftPin, MSBFIRST, value);
digitalWrite(storagePin, HIGH);
delay(20);
}
```



Pour aller plus loin

Vous pouvez saisir directement la combinaison binaire au lieu du nombre décimal 157 lors de l'initialisation des variables et éviter ainsi la conversion. Tapez seulement B10011101. Le préfixe B indique qu'il s'agit d'une combinaison binaire avec laquelle la variable doit être initialisée.

Ce chronogramme vous montre les niveaux des trois lignes de données, les unes par rapport aux autres, pour commander le registre à décalage pendant le déroulement chronologique.



On voit tout en haut le signal d'horloge SH_CP pour la prise en charge des données à l'entrée série DS. À l'issue de la 8^e période, le niveau de ST_CP passe de LOW à HIGH et les données des registres internes sont transmises aux sorties. Essayez avec différentes valeurs et différents sens de transmission pour bien comprendre.

▲ **Figure 6-13**

Chronogramme pour le nombre transféré 157 (B10011101)



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- 75HC595 ;
- 75HC595 fiche technique ;
- 74HC595 datasheet.

Problèmes courants

Si les LED ne s'allument pas l'une après l'autre, débranchez le port USB de la carte pour plus de sécurité et vérifiez ce qui suit.

- Vos fiches de raccordement sur la plaque d'essais correspondent-elles vraiment au circuit ?
- Pas de court-circuit éventuel ?
- Les différentes LED sont-elles correctement branchées ? La polarité est-elle correcte ?
- Les résistances ont-elles bien les bonnes valeurs ?
- Le registre à décalage est-il correctement câblé ? Contrôlez encore une fois tous les raccordements qui sont nombreux.
- Le code du sketch est-il correct ?

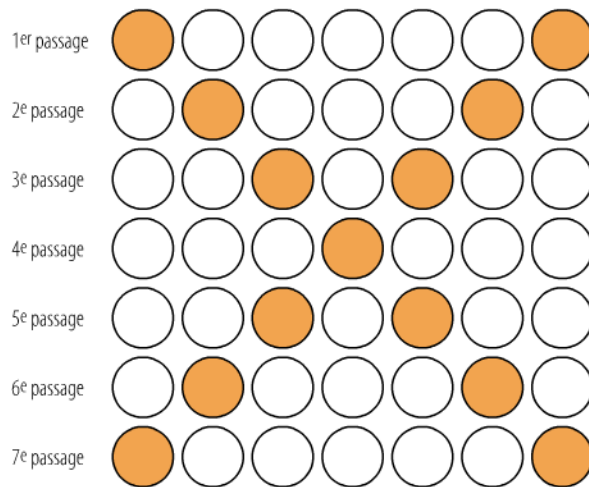
Qu'avez-vous appris ?

- Vous savez ce qu'est un registre à décalage de type 74HC595 avec une entrée série et huit sorties.
- Le premier sketch permet de commander les trois lignes de données SH_CP, ST_CP et DS, les signaux d'horloge étant contrôlés par un front d'horloge montant, autrement dit ne réagissant que quand le niveau passe de LOW à HIGH.
- L'instruction `shiftOut` permet d'envoyer au registre à décalage des combinaisons de bits au moyen de nombres non seulement décimaux mais aussi binaires.
- Vous pouvez initialiser une variable du type de donnée `byte` avec un nombre entier, par exemple 157, ou à l'aide de la combinaison de bits correspondante qui doit être précédée du préfixe `B`, soit par exemple `B10011101`.

Exercice complémentaire

Dans cet exercice complémentaire, je vous propose d'abord de faire s'allumer les LED au gré de toutes les combinaisons de bits possibles entre 00000000 et 11111111.

Puis je vous invite à concevoir différents motifs ou séquences, selon lesquels les LED doivent clignoter. Pour cela, je vous donne l'exemple de la figure 6-14.



◀ **Figure 6-14**
Séquence de LED commandée par
un registre à décalage 74HC595

Le motif du 7^e passage est le même que celui du 1^{er} et la séquence revient au début. Il s'agit de deux LED allumées qui se rapprochent l'une de l'autre pour s'éloigner à nouveau. Vous pouvez répéter ce déroulement trois fois. Toutes les LED doivent clignoter à la fin pendant 1 seconde l'une après l'autre, puis le cycle doit reprendre au début.

La machine à états

Au sommaire :

- la déclaration et l'initialisation de plusieurs variables ;
- la programmation de plusieurs broches comme sortie (OUTPUT) ;
- la programmation d'un port comme entrée (INPUT) ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- le sketch élargi (circuit interactif pour feux de circulation) ;
- un exercice complémentaire.

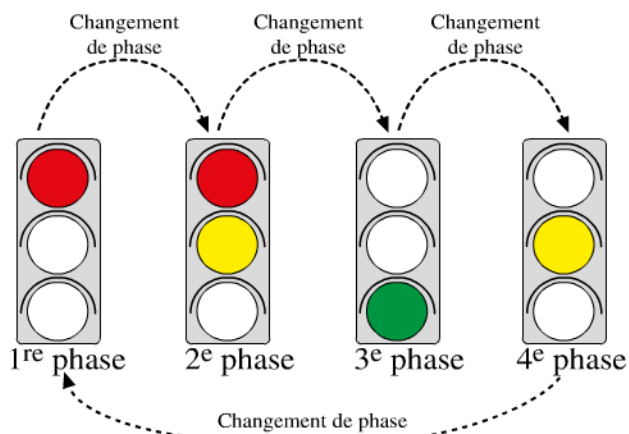
Des feux de circulation

La programmation de feux de circulation est une tâche classique. La plupart du temps, on y croise pour la première fois une *state machine* – son nom complet est *Finite State Machine* ou FSM. Il s'agit d'une machine supportant des états certes différents, mais finis. Voici ce qui caractérise ce modèle :

- l'état ;
- la transition d'état ;
- l'action.

Voyons d'abord les différentes phases de signalisation possibles.

Figure 7-1 ►
États de signalisation
avec changement de phase



Ici, on prévoit 4 phases de signalisation (de 1 à 4), le cycle reprenant ensuite au début. Pour plus de simplicité, nous nous en tiendrons à des feux pour un sens de circulation. La signification des différentes couleurs est bien connue de tous :

- rouge : interdiction de passer ;
- orange : attendre le signal suivant ;
- vert : autorisation de passer.

Chaque phase a une durée d'allumage définie. L'utilisateur de la route doit avoir en effet le temps de comprendre les différentes phases pour réagir en conséquence. Voici les durées d'allumage qui seront utilisées dans notre exemple (voir tableau 7-1), qui n'ont rien à voir avec la réalité. Bien évidemment, il est possible de régler le temps selon vos goûts.

Tableau 7-1 ►
Phases avec durée d'allumage

1 ^{re} phase	2 ^e phase	3 ^e phase	4 ^e phase
Durée : 10 s	Durée : 2 s	Durée : 10 s	Durée : 3 s

Une fois le code transmis, les feux de circulation doivent exécuter les 4 phases énoncées précédemment, puis recommencer au début.

Composants nécessaires



1 LED rouge



1 LED orange



1 LED verte



3 résistances de 330 Ω



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Voici le code du sketch pour commander les feux de circulation :

```
#define DELAY1 10000    //Pause 1, 10 secondes
#define DELAY2 2000     //Pause 2, 2 secondes
#define DELAY3 3000     //Pause 3, 3 secondes
int ledPinRed = 7;      //Broche 7 commande la LED rouge
int ledPinOrange = 6;   //Broche 6 commande la LED orange
int ledPinGreen = 5;    //Broche 5 commande la LED verte
void setup(){
  pinMode(ledPinRed, OUTPUT);    //Broche comme sortie
  pinMode(ledPinOrange, OUTPUT); //Broche comme sortie
  pinMode(ledPinGreen, OUTPUT);  //Broche comme sortie
}

void loop(){
  digitalWrite(ledPinRed, HIGH); //Allumage LED rouge
  delay(DELAY1);                 //Attendre 10 secondes
  digitalWrite(ledPinOrange, HIGH); //Allumage LED orange
  delay(DELAY2);                 //Attendre 2 secondes
  digitalWrite(ledPinRed, LOW);  //Extinction LED rouge
  digitalWrite(ledPinOrange, LOW); //Extinction LED orange
  digitalWrite(ledPinGreen, HIGH); //Allumage LED verte
  delay(DELAY1);                 //Attendre 10 secondes
  digitalWrite(ledPinGreen, LOW); //Extinction LED verte
```



```
digitalWrite(ledPinOrange, HIGH); //Allumage LED orange
delay(DELAY3); //Attendre 3 secondes
digitalWrite(ledPinOrange, LOW); //Extinction LED orange
}
```

Revue de code

Voici les variables qui sont techniquement nécessaires à notre expérimentation.

Tableau 7-2 ▶
Variables nécessaires et leur objet

Variable	Objet
ledPinRed	Commande la LED rouge.
ledPinOrange	Commande la LED orange.
ledPinGreen	Commande la LED verte.

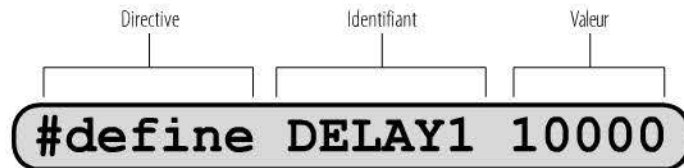


J'ai déjà une question ! Le code du sketch commence par trois lignes dont le contenu m'est complètement inconnu. Que signifient `#define` et le reste de la ligne après ?

Encore une fois, vous allez plus vite que la musique ! En fait, l'instruction `#define` n'est pas une véritable instruction, mais une directive de prétraitement. Rappelez-vous la directive de prétraitement `#include`, reconnaissable au point-virgule manquant en fin de ligne qui caractérise normalement la fin d'une instruction.

Quand le compilateur entreprend de traduire le code source, une partie spéciale de celui-ci appelée préprocesseur suit les directives de prétraitement, qui sont toujours introduites par le signe dièse `#`. Vous croiserez encore d'autres directives de ce type au cours de ce livre. La directive `#define` permet d'utiliser des noms symboliques et des constantes. La syntaxe pour l'utiliser est la suivante (voir figure 7-2).

Figure 7-2 ▶
La directive `#define`



Cette ligne agit comme suit : partout où le compilateur trouve l'identifiant `DELAY1` dans le code du sketch, il le remplace par la valeur `10000`. Vous pouvez vous servir de la directive `#define` partout où vous souhaitez utiliser des constantes dans le code. J'ai déjà soulevé ce problème auparavant : pas de *magic numbers* !

Pourquoi n'avez-vous pas utilisé `#define` partout où des broches ont été définies ? Ce sont pourtant également des constantes qui ne varient plus au cours du sketch.



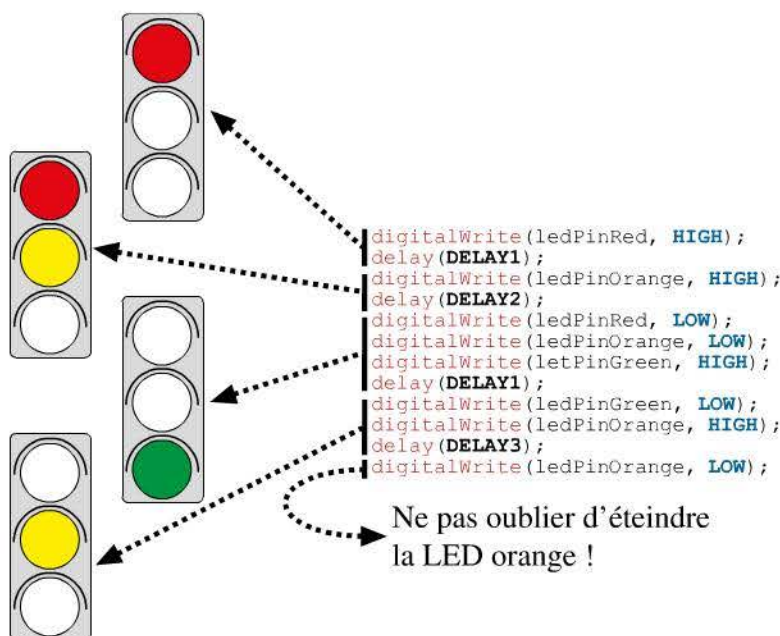
Vous avez raison ! J'aurais pu le faire partout et certains sketches Arduino, que vous trouverez sur Internet, emploient ce mode d'écriture. Au lieu de :

```
int ledPinRed = 7;
```

on peut alors écrire :

```
#define ledPinRed 7
```

Le sketch agit comme avant et cela ne change rien que vous utilisiez la première ou la seconde variante – mais il est important de vous en tenir à celle choisie et de ne pas en changer au gré de votre humeur. Pour ma part, j'emploie dans mon sketch la déclaration et l'initialisation de variable quand il s'agit de broches, et la directive `#define` pour les constantes. Revenons maintenant à notre sketch et voyons comment il fonctionne.



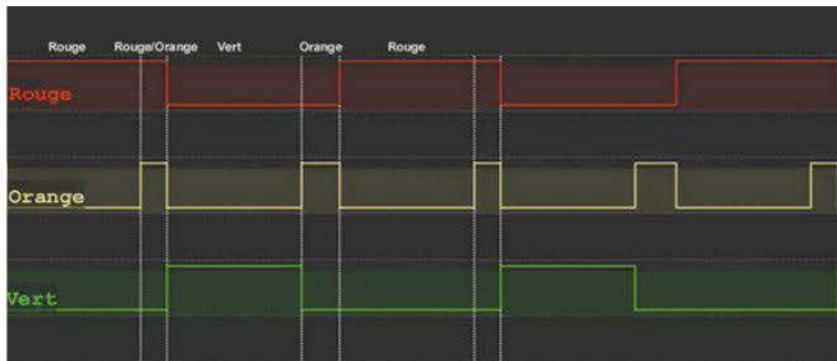
◀ Figure 7-3

Commande des différentes phases de signalisation

Pensez non seulement à allumer les diverses LED, mais aussi à les éteindre lors des différents changements de phase. Au passage de la phase 1 à la phase 2, seule une LED orange vient s'ajouter à la LED rouge. Mais au passage de la phase 2 à la phase 3, veillez à ce que les

LED rouge et orange s'éteignent avant que la LED verte ne s'allume. Jetez un coup d'œil au chronogramme pour voir comment les LED sont allumées à tour de rôle pendant les différentes phases.

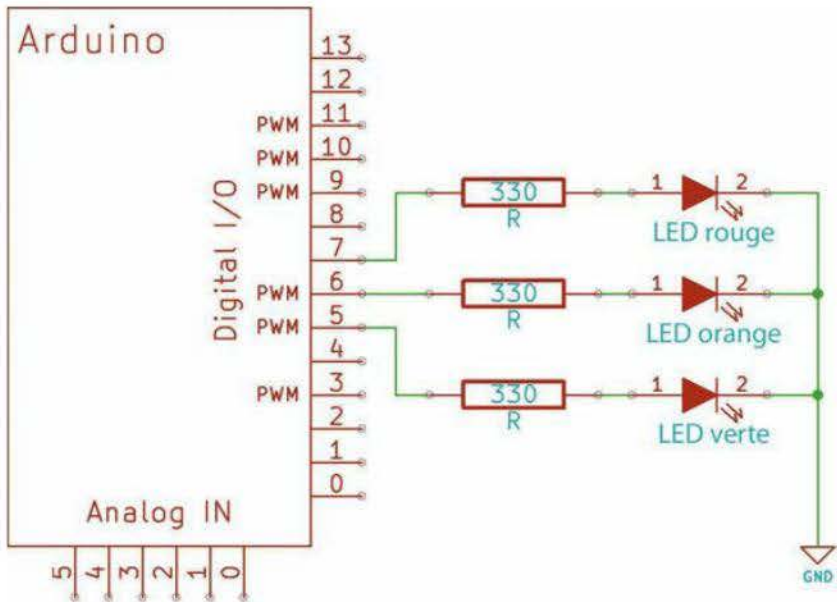
Figure 7-4 ►
Chronogramme des feux
de circulation



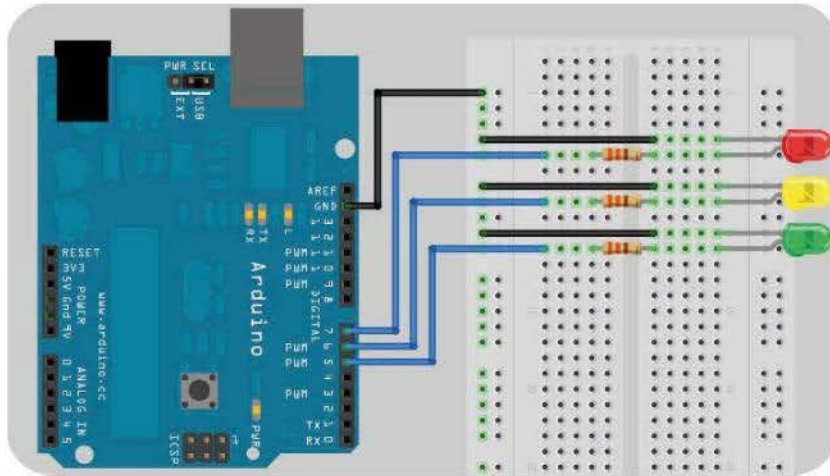
Schéma

Le schéma montre les trois LED de couleur avec leurs résistances de 330 ohms. Les besoins en composants supplémentaires sont légèrement moindres, mais les choses vont bientôt changer.

Figure 7-5 ►
Carte Arduino gérant nos feux
de circulation



Réalisation du circuit



◀ **Figure 7-6**
Réalisation du circuit pour feux
de circulation avec Fritzing

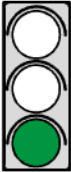

Sketch élargi (circuit interactif pour feux de circulation)

Ce sketch étant relativement simple dans sa programmation et sa construction, modifions un peu les choses.

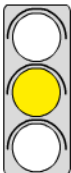

Imaginons maintenant des feux pour piétons installés sur un tronçon droit d'une route nationale. Il n'est pas utile que les phases pour les automobilistes changent sans cesse si aucun piéton ne veut traverser la voie de circulation. Comment les feux doivent-ils fonctionner avec leurs phases ? Quel matériel supplémentaire faut-il et comment faire pour étendre la logique ? Voici ce qu'il faut prendre en compte.

- Si aucun piéton ne se présente pour traverser la route, le feu reste vert pour les automobilistes et rouge pour les piétons.
- Si un piéton appuie sur le bouton gérant les feux pour traverser en toute sécurité, le feu passe à l'orange puis au rouge pour les automobilistes. Le feu passe ensuite au vert pour les piétons. Après un temps prédéfini, le feu repasse au rouge pour les piétons, et le feu rouge passe à l'orange puis au vert pour les automobilistes.

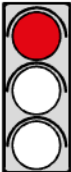

1^{re} phase

Automobiliste	Piéton	Explications
		Ces deux signaux lumineux restent allumés jusqu'à ce qu'un piéton s'approche et appuie sur le bouton gérant les feux. C'est alors seulement que les changements de phase se déclenchent et font en sorte que le feu passe au rouge pour les automobilistes et au vert pour les piétons.

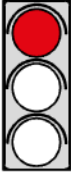

2^e phase

Automobiliste	Piéton	Explications
		Le changement de phase est déclenché par l'appui sur le bouton gérant les feux. Le feu passe à l'orange pour les automobilistes, ce qui signifie qu'il va passer au rouge sous peu. Durée : 3 s

3^e phase

Automobiliste	Piéton	Explications
		Pour des raisons de sécurité, le feu est d'abord rouge pour les automobilistes et pour les piétons. Cela permet aux automobilistes de libérer le cas échéant le passage piétons. Durée : 7 s

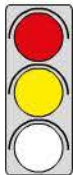

4^e phase

Automobiliste	Piéton	Explications
		Le feu passe au vert pour le piéton après un court moment. Durée : 10 s

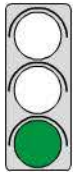

5^e phase

Automobiliste	Piéton	Explications
		Le feu repasse au rouge pour les piétons. Durée : 1 s

6^e phase

Automobiliste	Piéton	Explications
		Le feu passe du rouge à l'orange pour les automobilistes, ce qui les avertit que le feu va bientôt passer au vert. Durée : 2 s

7^e phase

Automobiliste	Piéton	Explications
		Le feu repasse au vert pour les automobilistes et au rouge pour les piétons. Cette dernière phase est semblable à la première. Durée : jusqu'au prochain appui sur le bouton

Composants supplémentaires

Ce sketch élargi nécessite les composants supplémentaires suivants.



1 LED rouge



1 LED verte



2 résistances de 330 Ω



1 résistance de 10 kΩ



1 bouton-poussoir

Le code élargi est alors le suivant :

```

#define DELAYo 10000 //Pause 0, 10 secondes
#define DELAY1 1000 //Pause 1, 1 seconde
#define DELAY2 2000 //Pause 2, 2 secondes
#define DELAY3 3000 //Pause 3, 3 secondes
int ledPinRedDrive = 7; //Broche 7 commande la LED rouge
                        //(feux pour automobilistes)
int ledPinOrangeDrive = 6; //Broche 6 commande la LED orange
                        //(feux pour automobilistes)
int ledPinGreenDrive = 5; //Broche 5 commande la LED verte
                        //(feux pour automobilistes)
int ledPinRedWalk = 3; //Broche 3 commande la LED rouge
                        //(feux pour piétons)
int ledPinGreenWalk = 2; //Broche 2 commande la LED verte
                        //(feux pour piétons)
int buttonPinLight = 8; //Bouton gérant les feux reliés
                        //à la broche 8
int buttonLightValue = LOW; //Variable pour l'état
                        //du bouton gérant les feux

void setup(){
  pinMode(ledPinRedDrive, OUTPUT); //Broche comme sortie
  pinMode(ledPinOrangeDrive, OUTPUT); //Broche comme sortie
  pinMode(ledPinGreenDrive, OUTPUT); //Broche comme sortie
  pinMode(ledPinRedWalk, OUTPUT); //Broche comme sortie
  pinMode(ledPinGreenWalk, OUTPUT); //Broche comme sortie
  pinMode(buttonPinLight, INPUT); //Broche comme entrée
  digitalWrite(ledPinGreenDrive, HIGH); //Valeurs de départ
                                        //(vert pour automobilistes)
  digitalWrite(ledPinRedWalk, HIGH); //Valeurs de départ
                                        //(rouge pour piétons)
}
void loop(){
  //Lire l'état du bouton gérant les feux dans la variable
  buttonLightValue = digitalRead (buttonPinLight);
  //Si bouton appuyé, fonction appelée
  if(buttonLightValue == HIGH)
    lightChange();
}
void lightChange(){
  digitalWrite (ledPinGreenDrive, LOW);

```

```

digitalWrite(ledPinOrangeDrive, HIGH); delay(DELAY3);
digitalWrite(ledPinOrangeDrive, LOW);
digitalWrite(ledPinRedDrive, HIGH); delay(DELAY1);
digitalWrite(ledPinRedWalk, LOW);
digitalWrite(ledPinGreenWalk, HIGH); delay(DELAY0);
digitalWrite(ledPinGreenWalk, LOW);
digitalWrite(ledPinRedWalk, HIGH); delay(DELAY1);
digitalWrite(ledPinOrangeDrive, HIGH); delay(DELAY2);
digitalWrite(ledPinRedDrive, LOW);
digitalWrite(ledPinOrangeDrive, LOW);
digitalWrite(ledPinGreenDrive, HIGH);
}

```

Le nombre de ports nécessaires est passé à 6, mais cela ne signifie pas pour autant que les choses sont devenues plus difficiles. Vous devez seulement soigner un peu plus le câblage et l'affectation des broches. Commençons par les variables à spécifier tout au début du programme.

D'un point de vue logiciel, voici les variables nécessaires à notre programmation expérimentale.

Variable	Objet
ledPinRedDrive	Commande la LED rouge (feux pour automobilistes)
ledPinOrangeDrive	Commande la LED orange (feux pour automobilistes)
ledPinGreenDrive	Commande la LED verte (feux pour automobilistes)
ledPinRedWalk	Commande la LED rouge (feux pour piétons)
ledPinGreenWalk	Commande la LED verte (feux pour piétons)
buttonPinLight	Broche de raccordement du bouton-poussoir des feux pour piétons
buttonLightValue	Enregistre la valeur de l'état du bouton-poussoir

◀ **Tableau 7-3**

Variables nécessaires et leur objet

Les différentes broches sont programmées comme entrées ou sorties, et la valeur de démarrage LOW est attribuée à la variable `buttonLightValue` au sein de la fonction `setup`. Parce qu'aucun changement de phase ne survient tant qu'on n'appuie pas sur le bouton, le circuit doit présenter un état de départ bien défini. Aussi les feux pour automobilistes et piétons sont-ils initialisés par les deux lignes :

```

digitalWrite(ledPinGreenDrive, HIGH);
digitalWrite(ledPinRedWalk, HIGH);

```

Au sein de la fonction `loop`, l'état du bouton-poussoir est constamment interrogé par la fonction `digitalRead` et le résultat est affecté à la variable `buttonLightValue`. L'évaluation intervient immédiatement dans le test de la structure de contrôle `if` :


```
if(buttonLightValue == HIGH)
    lightChange();
```

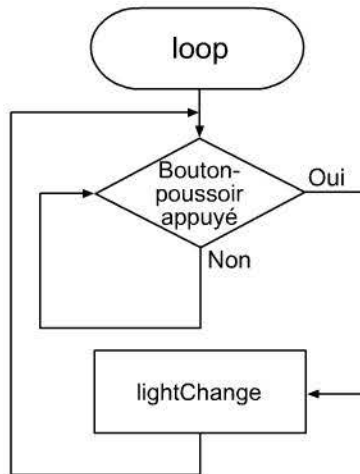
En cas de niveau HIGH, on passe directement à la fonction `lightChange`, qui déclenche alors les changements de phase.



Et que se passe-t-il si j'appuie une deuxième fois sur le bouton-poussoir ? Le déroulement s'en trouve-t-il d'une manière ou d'une autre perturbé ?

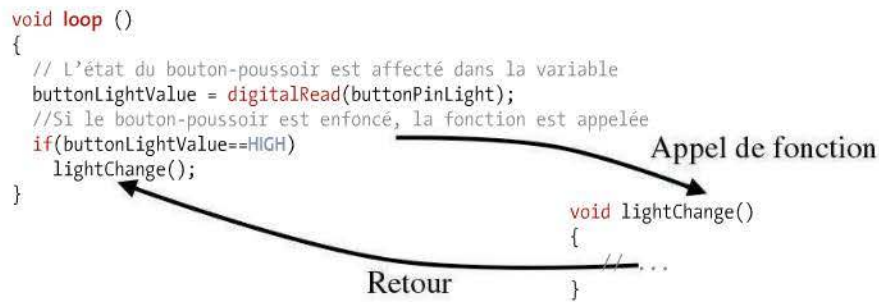
Cette question vient à point nommé. Récapitulons le déroulement du sketch. L'organigramme suivant devrait répondre à votre question.

Figure 7-7 ▶
Appel de la fonction
`lightChange`



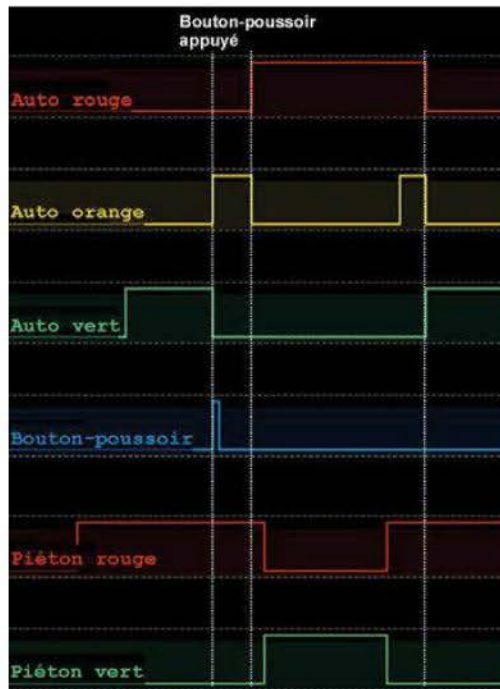
Comme vous pouvez le voir, l'état du bouton-poussoir est constamment interrogé et évalué en début de traitement dans la fonction `loop`. Ce sont les seules étapes de traitement dans cette fonction. Elle n'a donc rien d'autre à faire que d'observer l'état du bouton-poussoir et de bifurquer dans la fonction `lightChange` quand le niveau passe de LOW à HIGH. Une fois la fonction appelée, les divers changements de phase sont initiés et les phases sont maintenues par différents appels de la fonction `delay`.

Nous venons alors tout juste quitter la fonction `loop`. Un nouvel appui sur le bouton-poussoir ne serait donc pas enregistré par la logique, car la fonction `digitalRead` n'est plus constamment appelée. Il ne le serait qu'après avoir quitté la fonction `lightChange`.



◀ **Figure 7-8**
Appel et retour

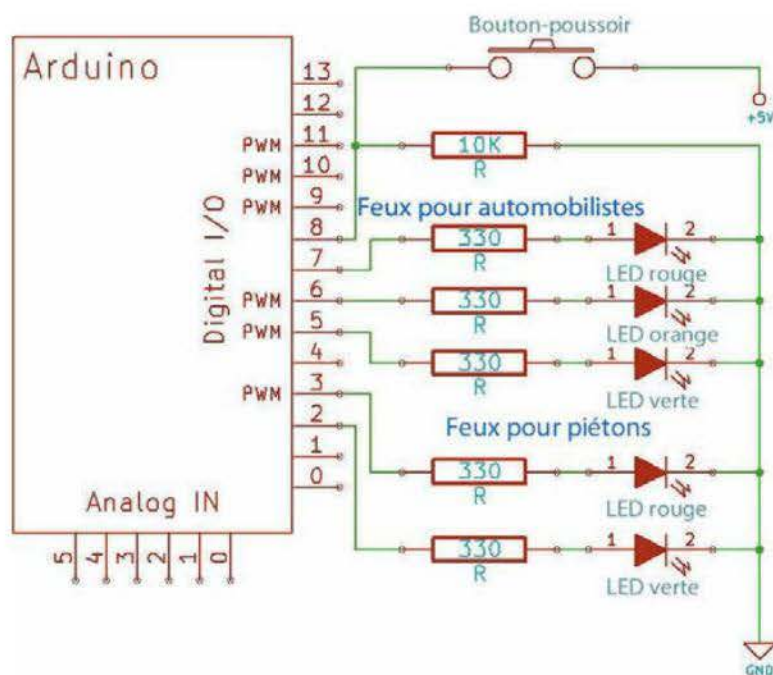
Avant de passer au schéma, voici encore un chronogramme montrant les différentes durées d'allumage l'une par rapport à l'autre. La situation de départ nous montre que le feu est vert pour les automobilistes et rouge pour les piétons. Un piéton ayant l'intention de traverser la route à un endroit supposé plus sûr appuie sur le bouton gérant les feux, ce qui initie les changements de phase.



◀ **Figure 7-9**
Chronogramme des feux interactifs

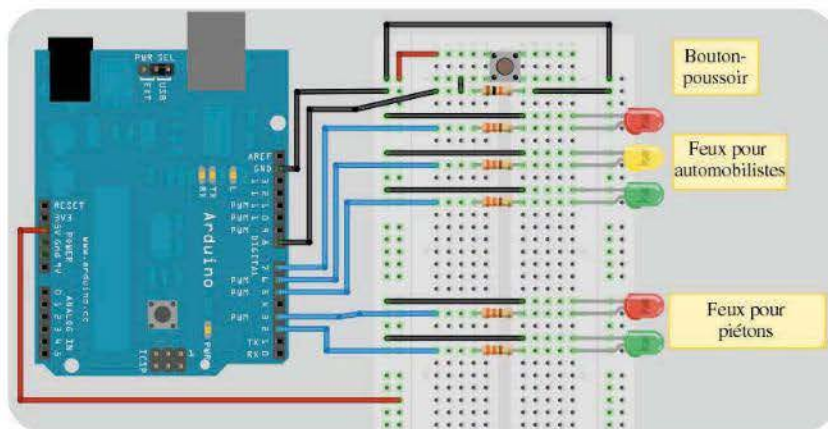
Dans le schéma relatif au dernier sketch, vous pouvez voir les extensions à ajouter pour faire fonctionner le circuit.

Figure 7-10 ►
Circuit interactif avec feux
pour automobilistes et piétons



La construction est alors la suivante sur la plaque d'essais.

Figure 7-11 ►
Réalisation du circuit interactif
pour feux avec Fritzing



Autre sketch élargi

Je vais maintenant modifier encore un peu le sketch gérant les feux de circulation, afin que vous fassiez travailler davantage votre matière grise.

Dans la programmation du circuit pour feux de circulation, j'ai toujours oublié d'éteindre l'une ou l'autre LED lors d'un changement

de phase avant d'allumer la suivante lors du premier essai, ce qui m'a gêné. J'ai donc imaginé de configurer plus simplement l'allumage et l'extinction des LED. Certes, cela demande un peu de préparation mais peut s'avérer utile pour vos montages à venir. Mais avant de commencer, je dois d'abord vous parler un peu des bits et des octets.

Le circuit ne change pas. Ordinateur et carte Arduino stockent toutes les données au niveau le plus bas de la mémoire sous forme de bits et d'octets (8 bits). J'ai déjà abordé ce thème dans le montage n° 6 sur l'extension de port numérique. En voici les grandes lignes (voir figure 7-12).

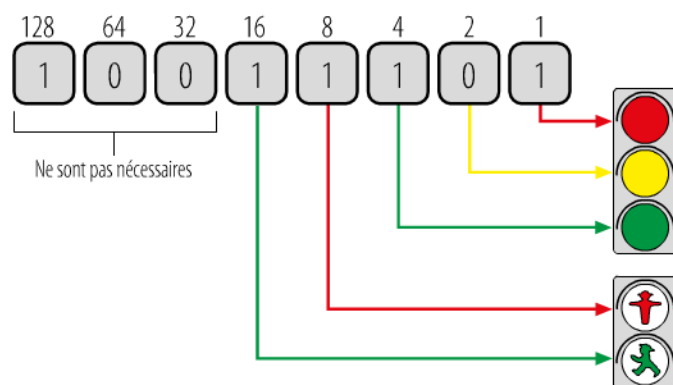
Puissances	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valeur	128	64	32	16	8	4	2	1
Combinaison de bits	1	0	0	1	1	1	0	1

◀ **Figure 7-12**
Combinaison binaire pour le nombre entier 157

Le nombre qui s'écrit en binaire 10011101 s'écrit en décimal :

$$1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^4 + 0 \times 2^5 + 0 \times 2^6 + 1 \times 2^7 = 157_{10}$$

Il suffit maintenant que certains bits de cet octet servent à commander les différentes LED de nos feux de circulation pour pouvoir allumer ou éteindre toutes les LED au moyen d'une seule valeur exprimée en décimal.



◀ **Figure 7-13**
Quel bit pour quelle LED ?

On voit que 5 bits de cet octet suffisent à commander les feux. Mais comment fait-on au juste ? J'ai reporté dans le tableau 7-4 les nombres décimaux correspondants, que j'ai déterminés à partir des différentes phases.

Tableau 7-4 ►
Nombres décimaux
pour commander les LED

	Piéton		Automobiliste			
LED	Verte	Rouge	Verte	Orange	Rouge	Nombre décimal
Poids	$2^3=8$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	
Phase 1	0	1	1	0	0	12
Phase 2	0	1	0	1	0	10
Phase 3	0	1	0	0	1	9
Phase 4	1	0	0	0	1	17
Phase 5	0	1	0	0	1	9
Phase 6	0	1	0	1	1	11

Reste à trouver, à partir des nombres décimaux correspondants, quel bit commande une LED particulière. C'est possible avec l'opérateur ET bit à bit &. Le tableau 7-5 montre que le résultat n'est 1 que si les deux opérandes ont 1 pour valeur.

Tableau 7-5 ►
Opération ET bit à bit

Opérande 1	Opérande 2	Opération ET
0	0	0
0	1	0
1	0	0
1	1	1

Voici un exemple : vérifions que la LED rouge des feux pour piétons s'allume pendant la phase 1 de notre commande pour feux de circulation.

Tableau 7-6 ►
Contrôle de correspondance du bit

	Piéton		Automobiliste			
LED	Verte	Rouge	Verte	Orange	Rouge	Nombre décimal
Poids	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	
Phase 1	0	1	1	0	0	12
Opérande	0	1	0	0	0	8
Résultat	0	1	0	0	0	8

Le deuxième opérande avec la valeur décimale 8 sert en quelque sorte de filtre. Il vérifie, seulement dans la position du bit de poids 2^3 , qu'un 1 se trouve bien dans le premier opérande. C'est le cas dans notre exemple et le résultat est 8. Le tableau 7-7 donne les nombres décimaux pour lesquels des opérations ET bit à bit doivent être effectuées avec les valeurs provenant des différentes phases pour déterminer l'état voulu de la LED.

LED	Valeur du 2 ^e opérande
LED rouge (automobiliste)	1
LED orange (automobiliste)	2
LED verte (automobiliste)	4
LED rouge (piéton)	8
LED verte (piéton)	16

◀ **Tableau 7-7**

Valeurs pour déterminer les bits à 1 ou à 0

Nous nous servons de l'opérateur conditionnel `?` pour la vérification. Il s'agit d'une forme d'évaluation spéciale d'une expression. La syntaxe générale est la suivante (voir figure 7-4).

Condition?Instruction1:Instruction2

◀ **Figure 7-14**

Opérateur conditionnel ?

Quand l'exécution du programme arrive à cette ligne, la condition est d'abord évaluée. Si le résultat est vrai, `Instruction1` est exécutée, sinon c'est `Instruction2` qui l'est. Pour commander toutes les LED avec cette structure, les lignes de code suivantes doivent être écrites, le nombre décimal pour commander les LED étant mémorisé dans la variable `lightValue`.

```
digitalWrite(ledPinRedDrive, (lightValue&1)==1?HIGH:LOW);
digitalWrite(ledPinOrangeDrive, (lightValue&2)==2?HIGH:LOW);
digitalWrite(ledPinGreenDrive, (lightValue&4)==4?HIGH:LOW);
digitalWrite(ledPinRedWalk, (lightValue&8)==8?HIGH:LOW);
digitalWrite(ledPinGreenWalk, (lightValue&16)==16?HIGH:LOW);
```

Ces 5 lignes de code permettent de commander l'état (allumé ou éteint) des 5 LED.

Il y a quelque chose que je ne comprends pas. Comment obtient-on les différentes durées d'allumage des diverses phases de signalisation ? Je ne vois nulle part l'instruction `delay` qui sert à définir les pauses.

Bonne remarque Ardu, aussi ces lignes de code sont-elles insérées dans une fonction à part et complétées par `lightValue` et une deuxième valeur pour la fonction `delay`. Cela donne :

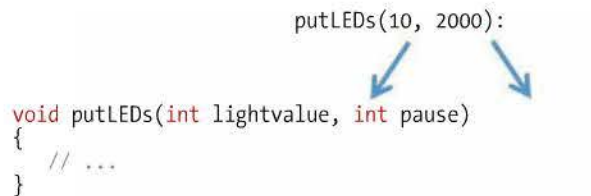
```
void putLEDs(int lightvalue, int pause){
    digitalWrite(ledPinRedDrive, (lightValue&1)= =1?HIGH:LOW);
    digitalWrite(ledPinOrangeDrive, (lightValue&2)==2?HIGH:LOW);
    digitalWrite(ledPinGreenDrive, (lightValue&4)==4?HIGH:LOW);
    digitalWrite(ledPinRedWalk, (lightValue&8)==8?HIGH:LOW);
    digitalWrite(ledPinGreenWalk, (lightValue&16)==16?HIGH:LOW);
    delay(pause);
}
```



Pour commander les différentes phases de signalisation, il ne vous reste plus qu'à appeler cette fonction avec les valeurs correspondantes qui figurent dans le tableau 7-4. Les appels sont alors les suivants :

```
void lightChange(){
    putLEDs(10, 2000);
    putLEDs(9, 1000);
    putLEDs(17, 10000);
    putLEDs(9, 1000);
    putLEDs(11, 2000);
    putLEDs(12, 0);
}
```

On voit que la fonction `putLEDs` est appelée dans la fonction `lightChange`. Regardons maintenant les choses de plus près à l'aide d'un exemple. La fonction présentant plusieurs paramètres, il est certainement utile de savoir dans quel ordre ils sont transmis lors de l'appel.



L'ordre de transmission des arguments `10` et `2000` aux paramètres de la fonction `putLEDs` est exactement celui dans lequel vous les avez écrits entre parenthèses. Les paramètres de la fonction sont définis par les variables locales `lightValue` et `pause`, dans lesquelles les valeurs transmises sont copiées.



Attention !

Respectez impérativement l'ordre des arguments lors de l'appel de la fonction. Faute de quoi, le sketch ne plantera pas dans ce cas, mais le circuit ne réagira pas comme prévu. Il faut donc que :

- le nombre des arguments doit coïncider avec celui des paramètres ;
- les types de données des arguments transmis doivent correspondre à ceux des paramètres ;
- l'ordre doit être respecté lors de l'appel.

Vous avez employé encore une fois l'expression « variable locale ». Je n'ai pas encore bien saisi la différence entre variable locale et variable globale.



Pas de problème ! La différence est toute simple. Les variables globales sont déclarées et initialisées en début de sketch et sont visibles partout, même à l'intérieur des fonctions, pendant le fonctionnement. La ligne de code suivante montre une variable globale de notre sketch :

```
int ledPinRedDrive = 7; //Broche 7 commande la LED rouge (feux pour
                        //automobilistes)

// ...
```

Celle-ci est utilisée plus tard dans la fonction `setup`. Elle y est donc visible et vous pouvez y accéder.

```
void setup( ){
  pinMode(ledPinRedDrive, OUTPUT); //Broche comme sortie
}
```

Les variables locales sont déclarées ou initialisées dans des fonctions ou, par exemple, dans une boucle `for`. Elles ont une durée de vie limitée et ne sont visibles que dans la fonction ou le bloc d'exécution. « Durée de vie » signifie qu'une zone spéciale est mise à la disposition des variables locales lorsque la fonction est appelée dans la mémoire. Une fois la fonction quittée, ces variables ne sont plus utiles et la mémoire est libérée. Une variable locale n'est jamais visible hormis dans la fonction où elle a été déclarée et elle ne peut pas non plus être utilisée depuis l'extérieur.

Bon, j'ai compris. Mais qu'en est-il des valeurs définies avec `#define` au début du sketch ? Quel comportement ont-elles ?



Vous pouvez aussi les considérer comme des définitions globales qui sont visibles et accessibles partout dans le sketch. Maintenant que vous connaissez la directive `#define`, je peux vous dire que des constantes telles que `HIGH`, `LOW`, `INPUT` ou `OUTPUT` – et de nombreuses autres encore – ont été également définies par ces directives.



Pour aller plus loin

Dans le répertoire suivant :

`arduino-1.x.y\hardware\arduino\cores\arduino`

vous trouverez, entre autres, un fichier nommé `Arduino.h`. Ce fichier de l'EDI d'Arduino contient beaucoup de définitions importantes, notamment celles dont je viens de parler. En voici un court extrait :


```

36 #define HIGH 0x1
37 #define LOW 0x0
38
39 #define INPUT 0x0
40 #define OUTPUT 0x1
41
42 #define true 0x1
43 #define false 0x0
44
45 #define PI 3.1415926535897932384626433832795
46 #define HALF_PI 1.5707963267948966192313216916398
47 #define TWO_PI 6.283185307179586476925286766559
48 #define DEG_TO_RAD 0.017453292519943295769236907684886
49 #define RAD_TO_DEG 57.295779513082320876798154814105
50
51 #define SERIAL 0x0
52 #define DISPLAY 0x1
53
54 #define LSBFIRST 0
55 #define MSBFIRST 1

```

Cela ne vous rappelle rien ? Vous en saurez bientôt plus sur ce qu'est un fichier d'en-tête dans le montage n° 9. Contentez-vous pour l'instant de savoir qu'il est intégré par le compilateur dans le projet et que toutes les définitions qu'il contient sont globales et disponibles dans le sketch.

Problèmes courants

Si les LED ne s'allument pas les unes après les autres, débranchez le port USB de la carte pour plus de sécurité et vérifiez ce qui suit.

- Vos fiches de raccordement sur la plaque d'essais correspondent-elles vraiment au circuit ?
- Est-ce qu'il y a un court-circuit éventuel ?
- Les différentes LED sont-elles correctement branchées ? La polarité est-elle correcte ?
- Les résistances ont-elles bien les bonnes valeurs ?
- Le code du sketch est-il correct ?
- Le bouton-poussoir est-il correctement câblé ? Vérifiez encore une fois les contacts en question avec un testeur de continuité.

Qu'avez-vous appris ?

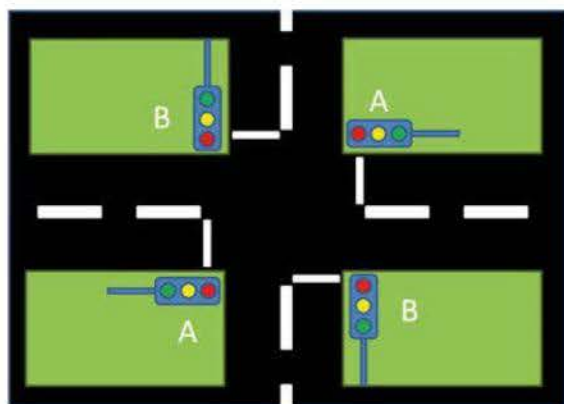
- Vous avez découvert comment évaluer le niveau d'une entrée numérique à l'aide de la fonction `digitalRead`.
- Vous avez réalisé un circuit pour feux de circulation à la fois simple, car initiant automatiquement les différents changements de phase indépendamment des influences externes, mais aussi interactif car réagissant avec un capteur (ici, un bouton-poussoir)

à des impulsions de l'extérieur et déclenchant alors seulement les changements de phase.

- Utiliser la directive de prétraitement `#define` ne devrait plus vous poser de problèmes. Elle est employée la plupart du temps là où des constantes sont définies. Le compilateur remplace partout dans le code le nom de l'identifiant par l'expression correspondante.
- L'opérateur conditionnel `?` peut être employé pour retourner différentes valeurs en fonction de l'évaluation d'une expression. Le mode d'écriture est vraiment compact et n'est pas toujours immédiatement compréhensible.
- Vous avez appris à transmettre plusieurs valeurs à une fonction, et vous savez en détail à quoi il faut faire attention.
- Vous connaissez la différence entre variable locale et variable globale, et vous savez ce que visibilité et durée de vie signifient dans ce cas.

Exercice complémentaire

Réalisez un circuit pour feux de circulation à un croisement. Le dessin suivant peut vous servir de base.

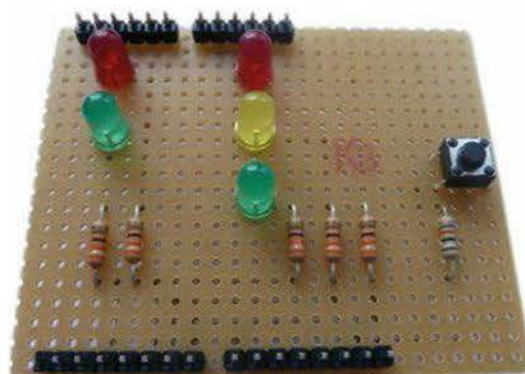


Les paires de feux de circulation A et B doivent être commandées en même temps. Cette fois-ci, il n'y aura pas de feux pour piétons. Veillez à ce que quand un sens passe au rouge, l'autre ne passe pas tout de suite au vert. Un délai de sécurité doit être prévu pour que les automobilistes déjà engagés puissent encore franchir le croisement quand le feu passe du vert au rouge.

Cadeau !

Je vous propose ici une carte à construire facilement, qui vous servira à réaliser le circuit pour feux de circulation avec feux pour piétons.

Figure 7-15 ►
Circuit pour feux de circulation
avec feux pour piétons



Le dé électronique

Au sommaire :

- la déclaration et l'initialisation d'un tableau bidimensionnel ;
- la programmation de plusieurs broches comme sortie (OUTPUT) ;
- la programmation d'un port comme entrée (INPUT) ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire ;
- quelque chose d'intéressant.

Qu'est-ce qu'un dé électronique ?

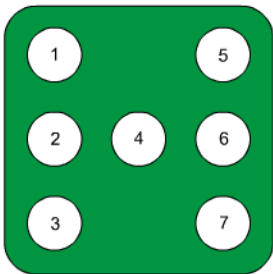
Bien que les derniers montages vous aient donné quelques bases pour programmer la carte Arduino, vous pensez sûrement être encore loin du compte... Aussi allons-nous appliquer, approfondir et élargir nos connaissances en étudiant quelques circuits intéressants. La construction d'un dé électronique est toujours plaisante. Il y a quelques années de cela, quand les microprocesseurs n'existaient pas ou étaient hors de prix, on utilisait plusieurs circuits intégrés. Vous trouverez pour ce faire de nombreuses instructions de bricolage sur Internet. Notre but est ici de commander le dé électronique avec la seule carte Arduino.

Tout le monde a déjà joué aux dés, que ce soit au 421, au 5000 ou au Yams. Aussi notre prochain circuit sera celui d'un dé électronique. Il se compose d'une unité d'affichage avec sept LED et un bouton-



poussoir pour lancer le dé. Voici d’abord la disposition des LED qui est celle des points d’un véritable dé. Les points portent tous un numéro pour mieux se repérer au moment de commander les LED. Le numéro 1 se trouve en haut à gauche, la numérotation se poursuivant vers le bas puis vers la droite jusqu’au numéro 7 qui se trouve en bas à droite.


Figure 8-1 ►
Numérotation des points du dé



Notre construction doit comporter un bouton-poussoir qui lance le dé. Lorsqu’on appuie dessus toutes les LED clignotent irrégulièrement et quand on le relâche, l’affichage s’arrête sur une certaine combinaison de LED, laquelle représente le nombre obtenu. Les différentes combinaisons de points sont répertoriées dans le tableau 8-1.

Tableau 8-1 ►
Quelle LED s’allume
pour quel nombre ?

Dé	Nombre	LED						
		1	2	3	4	5	6	7
	1				✓			
	2	✓						✓
	3	✓			✓			✓
	4	✓		✓		✓		✓
	5	✓		✓	✓	✓		✓

Dé	Nombre	LED						
		1	2	3	4	5	6	7
	6	✓	✓	✓		✓	✓	✓

◀ Tableau 8-1 (suite)

Quelle LED s'allume pour quel nombre ?

Il est certes tout à fait possible de construire le circuit sur votre plaque d'essais, mais ce n'est pas toujours simple compte tenu de la symétrie des LED. Dans un montage à part (le n° 22), nous construirons le circuit sur une carte spéciale appelée *shield*, et que nous brancherons par-dessus la carte Arduino. C'est la manière la plus propre et la plus pratique de fabriquer un dé électronique durable. Mais utilisons d'abord la plaque d'essais. Quel matériel nous faut-il ?

Composants nécessaires



7 LED rouges



7 résistances de 330 Ω



1 résistance de 10 k Ω



1 bouton-poussoir



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Voici le code du sketch pour commander le dé électronique :

```
#define WAITTIME 20
int pips[6][7] = {{0, 0, 0, 1, 0, 0, 0}, //Nombre sorti 1
                  {1, 0, 0, 0, 0, 0, 1}, //Nombre sorti 2
                  {1, 0, 0, 1, 0, 0, 1}, //Nombre sorti 3
                  {1, 0, 1, 0, 1, 0, 1}, //Nombre sorti 4
                  {1, 0, 1, 1, 1, 0, 1}, //Nombre sorti 5
                  {1, 1, 1, 0, 1, 1, 1}}; //Nombre sorti 6
```

```

int pin[] = {2, 3, 4, 5, 6, 7, 8};
int pinOffset = 2;    //Première LED sur broche 2
int buttonPin = 13;   //Bouton-poussoir sur broche 13

void setup(){
    for(int i = 0; i < 7; i++)
        pinMode(pin[i], OUTPUT);
    pinMode(buttonPin, INPUT);
}

void loop(){
    if (digitalRead (buttonPin) == HIGH)
        displayPips(random (1, 7)); //Générer un nombre entre 1 et 6
}

void displayPips(int value){
    for(int i = 0; i < 7; i++)
        digitalWrite(i + pinOffset,(pins[value - 1][i] == 1)?HIGH:LOW);
    delay(WAITTIME);    //Ajouter une courte pause
}

```

Revue de code

Du point de vue logiciel, les variables présentées dans le tableau 8-2 sont nécessaires à notre montage.

Tableau 8-2 ►
Variables nécessaires et leur objet

Variable	Objet
pips	Tableau bidimensionnel contenant les informations sur les LED à commander pour la valeur d’affichage respective
pin	Tableau unidimensionnel contenant les numéros des différentes broches de LED
pinOffset	La première LED ne se trouve pas sur la broche 0. Cette variable contient une valeur de décalage qui définit la position de départ pour une boucle <code>for</code> , afin de commander la première LED et toutes les autres.
buttonPin	Broche de raccordement du bouton-poussoir au dé

La programmation est déjà plus compliquée et nous n’avons pas seulement affaire cette fois à un tableau unidimensionnel comme dans le montage n° 5 sur le séquenceur de lumière. Un tableau bidimensionnel est ici nécessaire pour mémoriser les numéros des LED qui doivent s’allumer en fonction du nombre obtenu. Rappelons-nous encore une fois, par l’intermédiaire de la figure 8-2, comment un tableau unidimensionnel fonctionne et comment nous pouvons y accéder.

Index	0	1	2	3	4	5	6
Contenu du tableau	7	8	9	10	11	12	13

◀ **Figure 8-2**
Tableau unidimensionnel

La déclaration et l'initialisation du tableau sont assurées par la ligne suivante :

```
int ledPin[] = {7, 8, 9, 10, 11, 12, 13};
```

Le tableau contient ici sept éléments. Un tableau unidimensionnel est reconnaissable à sa paire de crochets derrière le nom de variable. On accède à un élément particulier en indiquant l'index entre les crochets. Vous écrivez donc ce qui suit pour accéder au 4^e élément :

```
ledPin[3]
```

N'oubliez pas que l'on compte à partir de 0 ! Un tableau bidimensionnel possède au sens figuré une dimension spatiale de plus, passant ainsi quasiment d'une droite unidimensionnelle à une surface.

		Colonnes (LED)						
		0	1	2	3	4	5	6
Index	0	0	0	0	1	0	0	0
	1	1	0	0	0	0	0	1
	2	1	0	0	1	0	0	1
	3	1	0	1	0	1	0	1
	4	1	0	1	1	1	0	1
	5	1	1	1	0	1	1	1
Lignes (nombre)								

◀ **Figure 8-3**
Tableau bidimensionnel

Il se comporte de la même manière que pour trouver une pièce sur un jeu d'échec. On la localise plus facilement grâce à des coordonnées : par exemple Dame sur D1, D indiquant la colonne et 1 la rangée. Le tableau présenté ici dispose de $6 \times 7 = 42$ éléments. Déclaration et initialisation se font comme d'habitude. Seule la paire de crochets est rajoutée pour la nouvelle dimension.

```
int pips[6][7] = {{0, 0, 0, 1, 0, 0, 0}, //Nombre sorti 1
                  {0, 0, 1, 0, 0, 0, 1}, //Nombre sorti 2
                  {0, 0, 1, 1, 0, 0, 1}, //Nombre sorti 3
                  {1, 0, 1, 0, 1, 0, 1}, //Nombre sorti 4
```



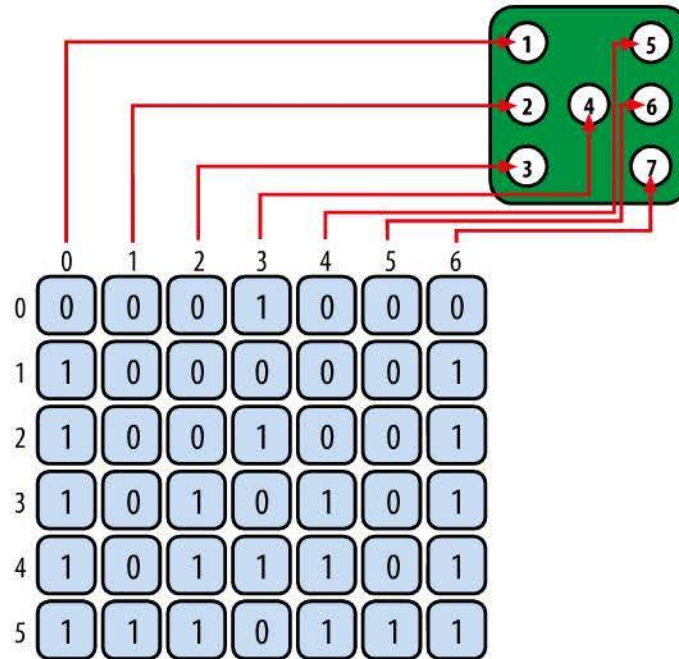
```
{1, 0, 1, 1, 1, 0, 1}, //Nombre sorti 5
{1, 1, 1, 0, 1, 1, 1}}; //Nombre sorti 6
```

La première valeur [6] entre crochets indique le nombre de lignes, le deuxième [7] le nombre de colonnes. La double paire de crochets permet aussi d'accéder à un élément :

```
pips[ligne][colonne]
```

Vous pouvez ainsi procéder ligne par ligne et lire les valeurs de LED correspondantes pour y accéder. La figure 8-4 montre l'affectation des différentes valeurs.

Figure 8-4 ►
Affectation des valeurs
des colonnes du tableau aux LED
correspondantes



Il y a quelque chose que je trouve bizarre. On ne peut pas faire 0 avec un dé. Pourtant le graphique, lui, commence par 0 et finit par 5 à la place de 6. Pouvez-vous m'expliquer ?

La réponse est simple. Vous avez un peu tout mélangé... Ce ne sont pas les points du dé qui sont énumérés mais l'index du tableau. Rappelez-vous que l'index commence toujours à 0 et présente donc un décalage numérique de - 1 par rapport aux points du dé. Voici maintenant un petit sketch qui affiche les contenus du tableau bidimensionnel sur le Serial Monitor :

```
int pips[6][7] = {{0, 0, 0, 1, 0, 0, 0}, //Nombre sorti 1
                  {1, 0, 0, 0, 0, 0, 1}, //Nombre sorti 2
```

```

        {1, 0, 0, 1, 0, 0, 1}, //Nombre sorti 3
        {1, 0, 1, 0, 1, 0, 1}, //Nombre sorti 4
        {1, 0, 1, 1, 1, 0, 1}, //Nombre sorti 5
        {1, 1, 1, 0, 1, 1, 1}}; //Nombre sorti 6

void setup(){
  Serial.begin(9600);
  for(int row = 0; row < 6; row++){
    for(int col = 0; col < 7; col++){
      Serial.print(pips[row][col]);
      Serial.println();
    }
  }
}
void loop(){...}

```

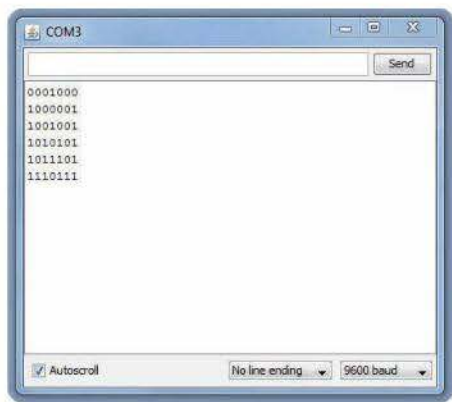
Il s'agit ici de deux boucles `for` imbriquées. La boucle extérieure, qui contient la variable de contrôle `row` (ligne), commence à compter à partir de sa valeur de départ 0. Vient ensuite la boucle intérieure, qui commence elle aussi par la valeur 0 de sa variable de contrôle `col` (colonne). La boucle intérieure doit cependant avoir fini de traiter toutes ses valeurs pour que la boucle extérieure incrémente la sienne.



Pour aller plus loin

Dans le cas de boucles imbriquées l'une dans l'autre, le traitement se fait de l'intérieur vers l'extérieur. Autrement dit, la boucle intérieure doit avoir exécuté tous ses passages avant que la boucle extérieure ne compte un de plus et que la boucle intérieure ne poursuive avec ses passages. Le cycle continue jusqu'à ce que toutes les boucles aient été traitées.

La figure 8-5 présente le contenu du tableau imprimé sur le Serial Monitor.



◀ **Figure 8-5**

Contenu du tableau imprimé ligne par ligne sur le Serial Monitor

Comparez cette impression à l'initialisation du tableau et vous verrez qu'elles coïncident. Passons maintenant à l'analyse du code propre-

ment dite. La fonction `setup` a encore pour tâche d'initialiser les différentes broches :

```
void setup(){
  for(int i = 0; i < 7; i++)
    pinMode(pin[i], OUTPUT);
  pinMode(buttonPin, INPUT);
}
```

Les broches pour commander les LED, broches programmées comme `OUTPUT` dans la fonction `setup`, sont également regroupées dans un tableau. Il n'y a qu'au bouton-poussoir, qui est relié à une entrée numérique, qu'une variable normale est affectée. La tâche principale est encore exécutée par la fonction `loop`.

```
void loop(){
  if(digitalRead(buttonPin) == HIGH)
    displayPips(random (1, 7)); //Générer un nombre entre 1 et 6
}

void displayPips(int value){
  for(int i = 0; i < 7; i++)
    digitalWrite(i + pinOffset, (pips[value-1][i] == 1)?HIGH:LOW);
  delay(WAITTIME);
}
```

Quand le bouton-poussoir est enfoncé, la fonction `displayPips` est appelée. Un chiffre aléatoire compris entre 1 et 6 lui est transmis comme argument. Voyons maintenant de plus près le mode d'exploitation de la fonction. Il s'agit essentiellement d'une boucle `for`, qui commande les différentes LED correspondant au chiffre transmis.

Supposons qu'un 4 soit sorti : la fonction reçoit cette valeur comme argument. La boucle `for` commence son travail. Elle commande les broches et détermine le niveau `HIGH/LOW` nécessaire pour la LED en question :



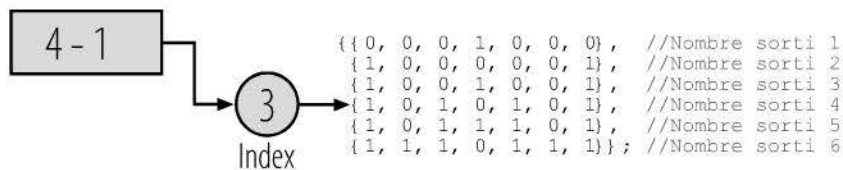
```
for(int i = 0; i < 7; i++)
  digitalWrite ( + pinOffset, (pips[ value - 1][ i ] == 1)?HIGH:LOW);
```

└──────────┘
└──────────┘
Broche de la LED
Niveau HIGH/LOW

C'est la variable de décalage (offset) qui est utilisée, mais je n'ai pas très bien compris à quoi elle sert.

Pas de problème, Arduus ! La variable `pinOffset` a pour valeur 2 et établit que la première broche à traiter se trouve à cette place. La première broche, portant le numéro 0, est `RX` et la deuxième, portant le

numéro 1, est TX. Ces deux broches sont en principe à éviter. La boucle `for` commençant par la valeur 0, la valeur de `pinOffset` lui est rajoutée. Mais revenons à notre exemple, dans lequel un 4 est sorti. La boucle `for` traite la 4^e ligne du tableau pour déterminer les niveaux HIGH/LOW nécessaires. Mais cette valeur doit être diminuée de 1 du fait que l'on commence par la valeur d'index 0.



◀ **Figure 8-6**
Sélection de l'élément du tableau pertinent pour un nombre préalablement sorti

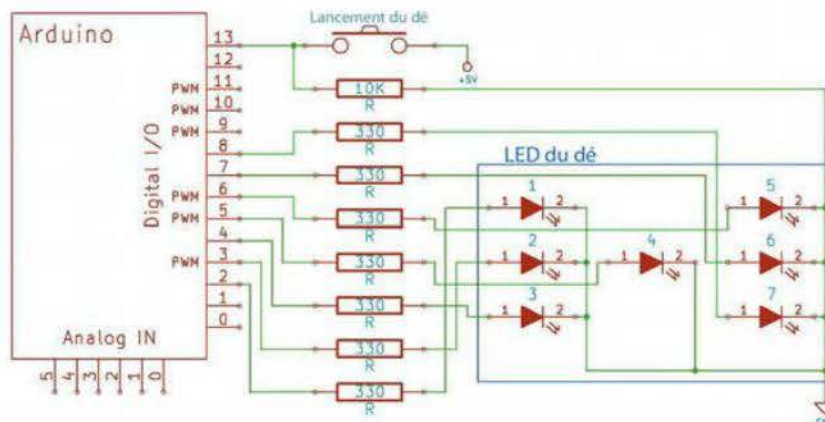
La ligne sélectionnée dans le tableau comporte les valeurs 1, 0, 1, 0, 1, 0, 1 qui sont traitées une à une par la boucle `for`. Ceci est initié par l'expression suivante :

```
(pins[Value - 1][i] == 1)?HIGH:LOW)
```

Celle-ci vérifie que les valeurs sont bien 1 ou 0. Le niveau HIGH est appliqué si c'est 1 et le niveau LOW si c'est 0. Les LED correspondant au nombre sorti sont ainsi activées ou désactivées. Tant que le bouton-poussoir est maintenu enfoncé, un nouveau nombre est déterminé et les LED clignotent toutes très vite l'une derrière l'autre. Une fois le bouton-poussoir relâché, le dernier nombre reste affiché. La constante `WAITTIME` permet de régler la vitesse à laquelle les nombres changent quand le bouton-poussoir est enfoncé, soit ici 20 ms.

Schéma

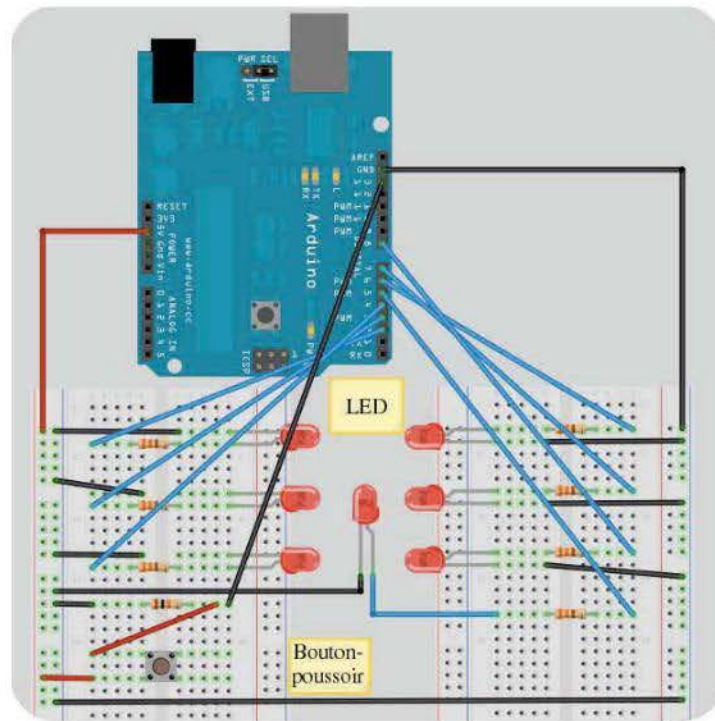
Le schéma montre les 7 LED du dé avec leurs résistances série de 330 Ω et le bouton-poussoir avec sa résistance pull-down.



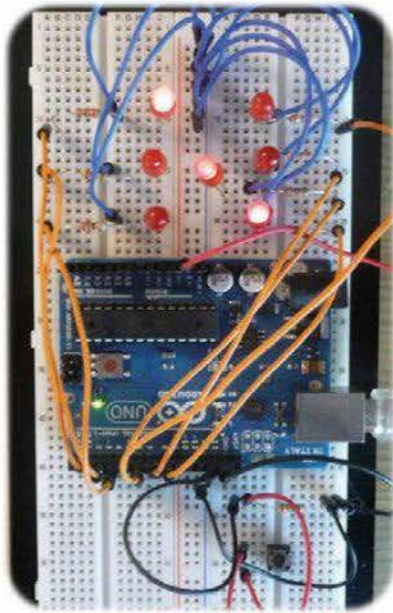
◀ **Figure 8-7**
Carte Arduino commandant chacune des 7 LED de notre dé

Réalisation du circuit

Figure 8-8 ►
Réalisation du dé électronique
avec Fritzing



On voit que j'ai utilisé deux plaques d'essais pour construire ce circuit. Il existe cependant des versions suffisamment larges pour pouvoir placer tous les composants dessus. Faites des essais de disposition car vous n'êtes pas obligé de suivre ce que j'ai fait. À vous de trouver votre propre stratégie. La figure 8-9 montre la construction du circuit sur une seule plaque, mais il a fallu « jongler » un peu pour y arriver. Ceci dit, il devrait très bien fonctionner.



◀ **Figure 8-9**
Réalisation du dé électronique
sur une plaque d'essais

Il m'est venu une idée. Je me suis souvenu du tableau unidimensionnel et j'ai essayé quelque chose. Vous avez dit qu'il est inutile d'écrire la dimension du tableau entre les crochets si celui-ci est initialisé aussitôt dans la même ligne. Le compilateur déduirait alors des valeurs transmises les dimensions que doit avoir le tableau. J'ai donc voulu essayer avec le tableau bidimensionnel mais j'ai eu une erreur.



L'idée n'est pas mauvaise et prouve que vous y pensez et appliquez ce que vous avez appris. Mais les choses ne vont pas si bien avec le tableau bidimensionnel. Si vous omettez toutes les indications sur la taille du tableau et écrivez :

```
int pips[][] = {{0, 0, 0, 1, 0, 0, 0}, //Nombre sorti 1
                {1, 0, 0, 0, 0, 0, 1}, //Nombre sorti 2
                {1, 0, 0, 1, 0, 0, 1}, //Nombre sorti 3
                {1, 0, 1, 0, 1, 0, 1}, //Nombre sorti 4
                {1, 0, 1, 1, 1, 0, 1}, //Nombre sorti 5
                {1, 1, 1, 0, 1, 1, 1}}; //Nombre sorti 6
```

le compilateur renâcle, comme vous avez pu le constater.

Le message d'erreur dit, pour résumer, que dans le cas d'un tableau multidimensionnel, toutes les limites, hormis la première, doivent être indiquées. Vous pouvez donc écrire la ligne suivante :

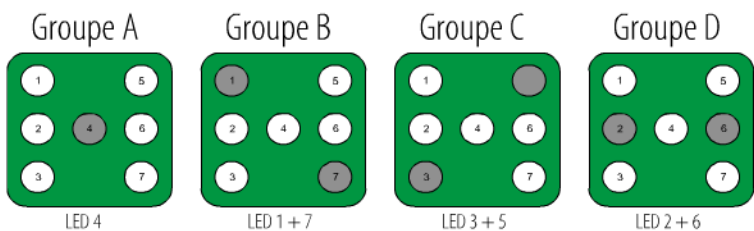
```
int pips[][7] = ...
```

Le compilateur acceptera ce code.

Que pouvons-nous encore améliorer ?

Il est presque toujours possible d'améliorer ou de simplifier les choses. Il vous suffit de prendre un peu de recul et de considérer le projet dans son ensemble. Ne vous creusez pas trop la tête. Les idées viennent souvent quand on est occupé à autre chose. Revenons à notre dé. Si vous regardez les différents points d'un dé pour différentes valeurs, vous remarquerez peut-être quelque chose. Retournez pour ce faire au tableau « Quelle LED s'allume pour quel nombre ? » Question : les huit LED s'allument-elles toutes indépendamment les unes des autres ? Ou se peut-il que certaines forment un groupe ? Question idiote, non ? C'est le cas, bien évidemment : la figure 8-10 montre les différents groupes.

Figure 8-10 ►
Groupes de LED sur le dé électronique



Pris séparément, le groupe A et le groupe B sont utilisables, ce qui est moins le cas pour les groupes C et D.

Quoi qu'il en soit, les configurations souhaitées sont générées par un groupe ou une combinaison de plusieurs groupes. Voyons maintenant lequel ou lesquels des groupes est ou sont concerné(s) par quels points du dé :

Tableau 8-3 ►
Points du dé et groupes de LED

Dé	1	2	3	4	5	6
Groupe A	✓		✓		✓	
Groupe B		✓	✓	✓	✓	✓
Groupe C				✓	✓	✓
Groupe D						✓

Ainsi, nous pouvons contrôler les LED avec 4 lignes de commandes au lieu de 7.

Si je comprends bien, il faut interconnecter deux LED dans les groupes B, C, et D. Ne peut-on pas faire autrement ? Dois-je les câbler en série ou en parallèle ?



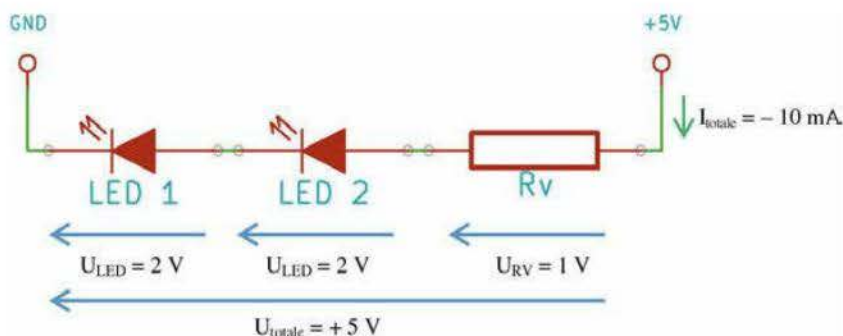
C'est tout à fait ça, Ardu's ! Dans le montage n° 2, nous avons calculé la résistance série pour une LED rouge. Relisez-le si besoin. Si plusieurs LED doivent être commandées, il faut les brancher en série. Il y a environ 2 V aux bornes d'une seule LED rouge, autrement dit la résistance série doit faire chuter 3 V. Deux LED étant ici branchées l'une derrière l'autre, il est possible de déterminer ce qui suit pour la chute de tension aux bornes de la résistance série R_V :

$$U_{RV} = U_{totale} - U_{LED1} - U_{LED2} = +5 \text{ V} - 2 \text{ V} - 2 \text{ V} = 1 \text{ V}$$

1 V doit donc être « grillé » sur la résistance série R_V pour que 2 V subsistent aux bornes de chaque LED. Pour ce qui est du courant, qui circule de la même façon dans tous les composants (rappelez-vous comment le courant se comporte dans un montage en série), je le fixe à 10 mA (10 mA = 0,01 A). On obtient donc les valeurs suivantes dans la formule pour calculer la résistance série :

$$R_V = \frac{U_{totale} - U_{LED1} + LED2}{I} = \frac{5 \text{ V} - 4 \text{ V}}{0,01 \text{ A}} = 100 \Omega$$

Le circuit ressemble à ceci :



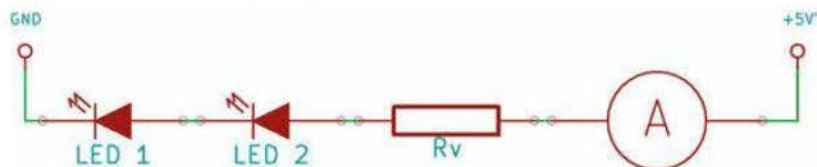
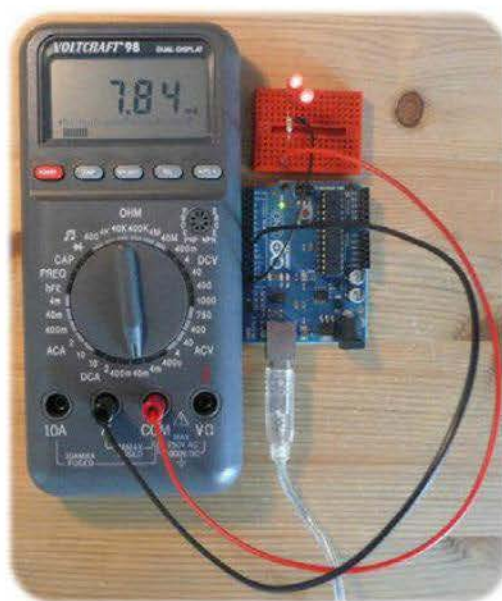
◀ **Figure 8-11**
Deux LED avec une résistance série

■ Attention !

Veillez à ce que les deux LED soient dans le même sens, sinon pas d'allumage. L'anode de la LED 1 est reliée à la cathode de la LED 2.

Ici aussi, j'ai vérifié le calcul de manière pratique pour m'assurer que tout est également en ordre.

Figure 8-12 ►
Mesure du courant sur le circuit
de commande avec deux LED
et une nouvelle résistance



Le courant de 7,84 mA est absolument correct et encore inférieur à la prescription de 10 mA maxi. Deux LED ayant naturellement besoin du double de tension d'alimentation par rapport à une seule, la résistance série doit être plus faible pour que la luminosité des deux LED, soit la même que celle d'une seule LED. Vous pouvez bien sûr utiliser la même résistance de 330 Ω pour tous les groupes A à D, ce qui signifie toutefois en théorie que la luminosité du groupe A sera plus forte avec une seule LED que le reste des groupes.

Passons maintenant à la programmation. Par quoi commencer ? Je vous suggère de revenir au tableau « Points du dé et groupes de LED » et de voir s'il s'en dégage une systématique établissant quel groupe de LED doit être commandé, à quel moment et pour quelles configurations de points du dé. Procédez étape par étape et observez un groupe après l'autre. Vous pouvez les traiter complètement à part l'un de l'autre car la logique de commande se charge ensuite de les rassembler pour un affichage en commun des véritables points du dé. Je vous montre encore une fois de manière simplifiée le groupe A du dernier tableau.

Dé	1	2	3	4	5	6
Groupe A	✓		✓		✓	

Encore un indice : qu'est-ce que les nombres 1, 3 et 5 ont en commun ?

Ce sont tous des nombres impairs.

Oui Arduus ! C'est la solution.

Formulation pour commander le groupe A

Commander le groupe A si le nombre aléatoire déterminé est impair. Passons maintenant au groupe B. Voici l'extrait correspondant du tableau :

Dé	1	2	3	4	5	6
Groupe B		✓	✓	✓	✓	✓

Que constatez-vous ici ?-

Tous les nombres sont concernés sauf le 1.

Bien, Arduus ! Mais à quoi pourrait bien ressembler une formulation que le microcontrôleur comprendrait sans problème ? Une description quelque peu maladroite donnerait ceci : commander le groupe B si le nombre est 2 ou 3 ou 4 ou 5 ou 6. Cherchez là encore le point commun et vous pourrez raccourcir fortement la formulation.

Formulation pour commander le groupe B

Commander le groupe B si le nombre aléatoire déterminé est supérieur à 1. Voyons maintenant le groupe C :

Dé	1	2	3	4	5	6
Groupe C				✓	✓	✓

Vous savez comment faire maintenant, n'est-ce pas ?





Tous les nombres supérieurs à 3 sont concernés.

Oui Ardus !

Formulation pour commander le groupe C

Commander le groupe C si le nombre aléatoire déterminé est supérieur à 3. Passons pour finir au groupe D :

Dé	1	2	3	4	5	6
Groupe D						✓

Plus besoin de vous demander maintenant, n'est-ce pas ?

Formulation pour commander le groupe D

Commander le groupe D si le nombre aléatoire déterminé est égal à 6. Nous pouvons à présent passer à la programmation proprement dite. Vous verrez que cette solution est beaucoup plus simple que l'utilisation d'un tableau. Mais il faut suivre mentalement quelques pistes jusqu'au bout avant de s'apercevoir que 4 broches au lieu de 7 sont nécessaires pour commander les LED. Cela permet cependant d'aborder cette thématique par le côté ludique. Voici le code du sketch pour commander le dé électronique avec moins de lignes de commande :

```
#define WAITTIME 20
int GroupA = 8;    //LED 4
int GroupB = 9;    //LED 1 + 7
int GroupC = 10;   //LED 3 + 5
int GroupD = 11;   //LED 2 + 6
int buttonPin = 13; //Bouton-poussoir à la broche 13
void setup(){
  pinMode(GroupA, OUTPUT);
  pinMode(GroupB, OUTPUT);
  pinMode(GroupC, OUTPUT);
  pinMode(GroupD, OUTPUT);
}

void loop(){
  if(digitalRead(buttonPin) == HIGH)
```



```

    displayPips(random(1, 7)); //Générer un nombre entre 1 et 6
}

void displayPips(int value){
    //Éteindre tous les groupes
    digitalWrite(GroupA, LOW);
    digitalWrite(GroupB, LOW);
    digitalWrite(GroupC, LOW);
    digitalWrite(GroupD, LOW);
    //Commande de tous les groupes
    if(value%2 != 0)    //La valeur est-elle impaire ?
        digitalWrite(GroupA, HIGH);
    if(value > 1)
        digitalWrite(GroupB, HIGH);
    if(value > 3)
        digitalWrite(GroupC, HIGH);
    if(value == 6)
        digitalWrite(GroupD, HIGH);
    delay(WAITTIME);    //Ajouter une courte pause
}

```

Je viens de m'apercevoir que vous avez oublié quelque chose ! Vous avez programmé les broches pour les groupes A à D comme sortie, mais vous avez oublié de définir le bouton-poussoir comme entrée.

C'est vrai, Arduus ! Je n'ai pas programmé cette entrée sur la broche 13 comme entrée. Vous avez raison sur ce point. Mais je n'ai pas non plus oublié puisque toutes les broches numériques sont définies comme entrée de manière standard et n'ont donc pas besoin d'être explicitement programmées en cas d'utilisation appropriée.

Vous pouvez bien entendu le faire partout dans votre sketch car cela aide certainement à comprendre.

J'ai en fait tout compris jusqu'à la ligne dans laquelle il est déterminé si la valeur est impaire. Pouvez-vous m'expliquer s'il vous plaît ?

Bien sûr, Arduus ! L'opérateur % (opérateur modulo) détermine toujours le reste d'une division. Si le nombre est divisible par 2, il s'agit d'un nombre pair. Le reste de la division étant dans ce cas toujours 0. La ligne :

```
if(value%2 != 0)
```

me permet cependant de demander si le reste est différent de 0 pour ainsi commander le groupe A.



Encore une remarque avant d'en venir au circuit : cela ne change pas grand-chose si, au lieu de la résistance série de $100\ \Omega$ calculée pour les groupes B à D, vous utilisez ici les anciennes résistances de $330\ \Omega$. La luminosité semble être la même, mais ce n'est qu'approximatif. La figure 8-13 montre qu'il faut moins de résistances série pour les LED que dans le montage précédent.

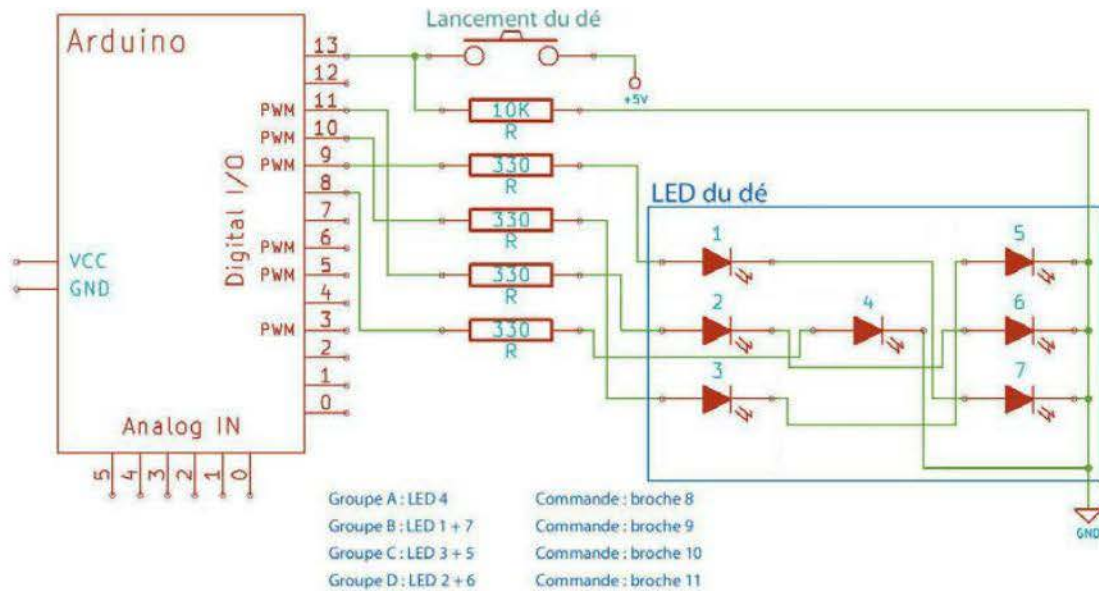
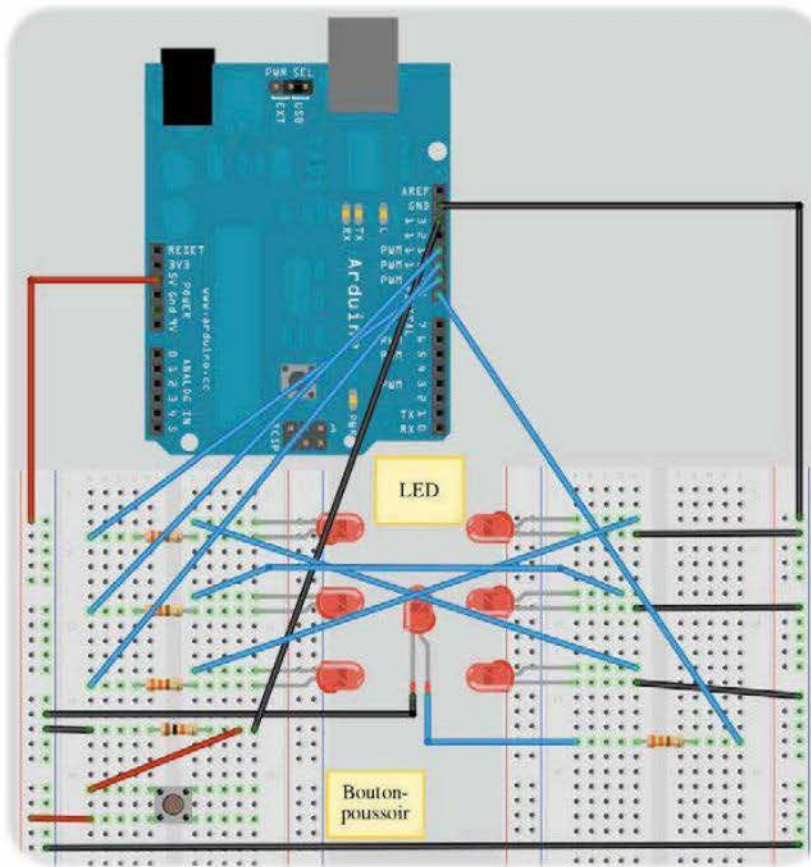


Figure 8-13 ▲
Carte Arduino commandant les 7 LED
de notre dé par groupe de LED

La réalisation sur la plaque d'essais s'avère plus simple parce que les lignes de commande sont moins nombreuses :



◀ **Figure 8-14**
Réalisation du dé électronique
utilisant des groupes de LED
avec Fritzing

Problèmes courants

Si les LED ne se mettent pas à clignoter après avoir appuyé sur le bouton-poussoir ou si les points du dé qui s'affichent sont bizarres, voire incohérents, débranchez le port USB de la carte pour plus de sécurité et vérifiez ce qui suit.

- Vos fiches de raccordement sur la maquette correspondent-elles vraiment au circuit ?
- N'y a-t-il pas un court-circuit éventuel entre elles ?
- Les LED ont-elles été mises dans le bon sens ? Autrement dit, la polarité est-elle correcte ?
- Les résistances ont-elles bien les bonnes valeurs ?
- Le code du sketch est-il correct ?
- Le bouton-poussoir est-il correctement câblé ? Vérifiez encore une fois les contacts en question avec un testeur de continuité.

Qu'avez-vous appris ?

- Vous avez appris dans ce montage comment déclarer et initialiser un tableau bidimensionnel et comment accéder aux différents éléments de ce tableau.
- Vous savez comment imprimer des contenus de variables avec le Serial Monitor pour vérifier l'exactitude de ces valeurs. Vous pouvez ainsi rechercher une erreur et analyser le code en cas de comportement incorrect. Vous devez cependant être sûr que le circuit est correctement câblé, sinon vous chercherez dans le code-source une erreur qui, en réalité, se situe au niveau du matériel. Cela vous évitera de perdre du temps et vous épargnera peut-être même une crise de nerf.
- Vous avez appris comment calculer une résistance série pour deux LED montées en série, de manière à ce que la luminosité demeure pratiquement inchangée.

Exercice complémentaire

L'objet de cet exercice est déjà un peu plus délicat. Vous vous souvenez sûrement du registre à décalage 74HC595 avec ses 8 sorties. Essayez de réaliser un circuit ou de programmer un sketch qui commande un dé électronique au moyen du registre à décalage. Combien de broches numériques économisez-vous avec cette variante ? Est-ce un avantage par rapport à la réalisation avec des groupes de LED ?

Comment créer une bibliothèque ?

Le jour viendra où vos connaissances vous permettront de réaliser des idées à vous, que d'autres n'auront peut-être pas encore eues. Mais peut-être souhaitez-vous aussi améliorer un projet existant parce que votre solution est plus élégante et moins compliquée à transposer. D'innombrables développeurs de logiciels se sont penchés avant vous sur les questions les plus diverses et ont programmé des bibliothèques pour épargner du travail et du temps aux autres développeurs. À travers ce projet, nous allons découvrir les grands principes de ces bibliothèques et leur création. Si le langage de programmation C++ – programmation orientée objet incluse – vous a toujours intéressé, vous allez être servi !

Les bibliothèques

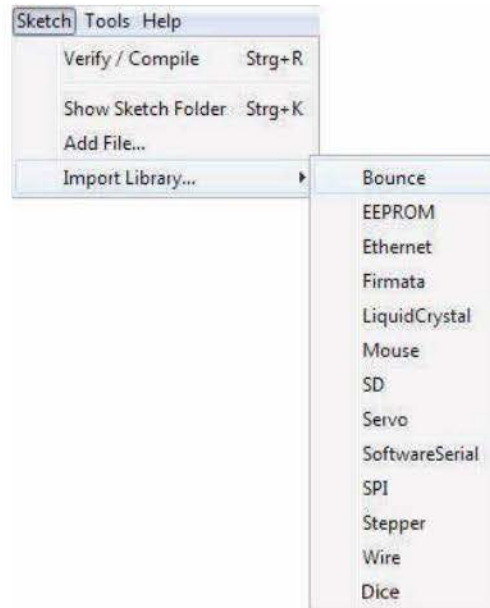
Une fois l'environnement de développement Arduino installé ou plutôt décompressé, vous disposez de quelques bibliothèques maison prêtes à l'emploi, appelées également librairies (*libraries* en anglais). Elles traitent de thèmes intéressants, tels que commander :

- un servomoteur ;
- un moteur pas-à-pas ;
- un écran LCD ;
- une EEPROM externe pour stocker des données...

Ces bibliothèques sont stockées dans le répertoire `libraries` du répertoire d'installation d'Arduino. Vous pouvez utiliser Windows Explorer ou passer par l'environnement de développement Arduino pour savoir quelles sont les bibliothèques disponibles. On y trouve

une entrée de menu spéciale Sketch>Import Library permettant d'afficher la liste correspondante.

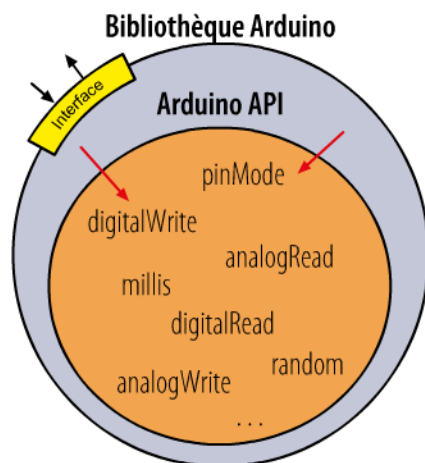
Figure 9-1 ►
Afficher ou importer
des bibliothèques



Les entrées du menu coïncident avec les répertoires du dossier `libraries`. Tout cela est bien beau, mais voyons d'abord comment une bibliothèque Arduino fonctionne et ce que nous pouvons faire avec.

Qu'est-ce qu'une bibliothèque exactement ?

Avant de passer à un exemple concret, vous devez savoir ce qu'est une bibliothèque. J'ai dit déjà qu'elle servait quasiment à emballer et réunir des tâches de programmation plus ou moins complexes en un paquet de programme. La figure 9-2 montre la coopération entre une bibliothèque Arduino et l'API Arduino.



◀ **Figure 9-2**
Comme fonctionne
une bibliothèque Arduino ?

Nous avons affaire à deux couches de programme interdépendantes. Je procède de l'intérieur vers l'extérieur. J'ai appelé la couche interne API Arduino. (API est l'abréviation d'*Application Programming Interface* et une interface vers toutes les instructions Arduino disponibles.) Je n'en ai sélectionné que très peu par manque de place. La couche externe est constituée par la bibliothèque Arduino, qui enveloppe la couche interne. Elle est de ce fait appelée *wrapper* (enveloppe) et se sert de l'API Arduino. Pour pouvoir accéder à la couche wrapper, une interface doit être mise en œuvre car vous entendez bien sûr exploiter la fonctionnalité d'une bibliothèque. Une interface est un portail d'accès à l'intérieur de la bibliothèque, qui est en soi une unité fermée. Le terme technique est encapsulation. Vous allez bientôt voir en détail de quoi il s'agit et en quoi cela concerne le langage de programmation C++.

En quoi les bibliothèques sont-elles utiles ?

Question idiote à laquelle j'ai déjà répondu plusieurs fois. Aussi me contenterai-je ici de vous en rappeler les avantages.

- Pour ne pas avoir à « réinventer la roue » chaque fois, les développeurs ont trouvé un moyen de stocker le code de programme dans une bibliothèque. Beaucoup de programmeurs dans le monde profitent de ces structures logicielles, qu'ils peuvent utiliser sans problème dans leurs propres projets. Le mot-clé est ici réutilisation.
- Une fois testée et débarrassée de ces erreurs, une bibliothèque peut être utilisée sans en connaître les déroulements internes. Sa

fonctionnalité est encapsulée et cachée du monde extérieur. Il suffit au programmeur de savoir utiliser son interface.

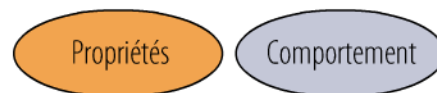
- Son code propre en est d'autant plus clair et plus stable.

Que signifie programmation orientée objet ?

La programmation orientée objet (ou POO) est du chinois pour la plupart des débutants, et peut même réserver maux de tête et nuits blanches à certains. Mais ce n'est pas obligé et j'espère pouvoir y contribuer, je veux dire à votre compréhension et non à vos maux de tête ! Dans le langage de programmation C++, tout est considéré comme objet et ce style – ou paradigme – de programmation s'oriente vers la réalité qui nous entoure. Nous sommes cernés d'innombrables objets qui sont plus ou moins réels et que nous pouvons toucher et observer. Si vous regardez un objet banal de plus près, vous pourrez constater certaines propriétés. Prenons par exemple un dé pour ne pas sortir du sujet. Vous savez déjà comment programmer et construire un dé électronique. Vous avez sûrement, dans l'un de vos jeux de société, un dé quelconque que vous pouvez regarder de plus près. Que pourriez-vous en dire, si vous deviez le décrire le plus précisément possible à un extra-terrestre ?

- À quoi ressemble-t-il ?
- Quelle taille a-t-il ?
- Est-il léger ou lourd ?
- De quelle couleur est-il ?
- A-t-il des points ou des symboles ?
- Quel nombre ou symbole est sorti ?
- Que peut-on faire avec ? (question idiote, non ?)

Les éléments de cette liste peuvent être répartis en deux catégories.



Mais quel élément fait partie de quelle catégorie ? Jetons un coup d'œil au tableau 9-1 puisqu'il s'agit de courant et de tension.

Propriétés	Comportement
Taille	Lancer le dé
Poids	
Couleurs	
Points ou symboles	
Nombre ou symbole sorti	

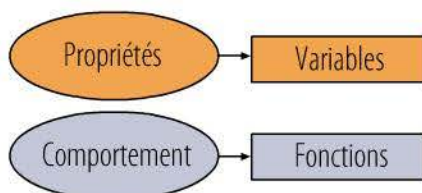
◀ **Tableau 9-1**
Distinction entre propriétés
et comportement

Seuls deux éléments de la liste sont pertinents pour la programmation prévue. Les autres sont certes intéressants, mais sans objet pour un dé électronique. Ces deux éléments sont :

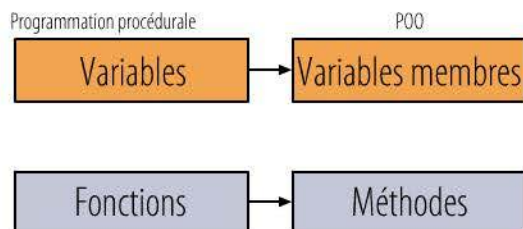
- le nombre de points sorti (état) ;
- lancer le dé (action).

Je n'ai pas la moindre idée de la manière dont on charge des propriétés ou un comportement dans un sketch. Comment fait-on ?

Ça ne pose aucun problème Arduus ! Voyez plutôt sur la figure suivante.



Les propriétés sont consignées dans des variables et le comportement est géré par des fonctions. Mais dans le contexte de la programmation orientée objet, variables et fonctions ont une autre désignation. Pas de quoi paniquer cependant puisque c'est en définitive la même chose.

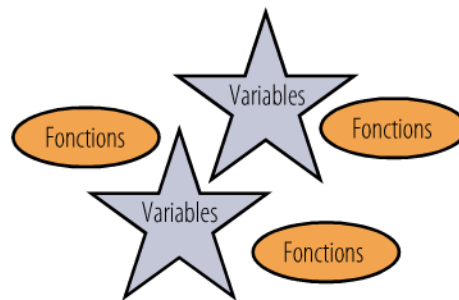


Les variables deviennent des variables membres (en anglais, *fields*) et les fonctions des méthodes (en anglais, *methods*).

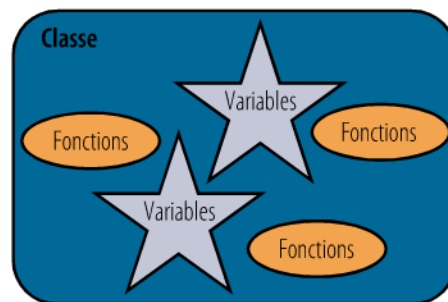
Quelle avancée formidable ! Il suffit de rebaptiser deux éléments d'un programme pour avoir un nouveau – comme vous dites – paradigme de programmation. Le progrès tient à peu de choses, non ?



Allons Ardus, ne soyez pas sarcastique. C'est que je n'ai pas fini. Dans la programmation procédurale que nous connaissons à travers les langages C ou Pascal, des instructions ayant un rapport logique, qui sont nécessaires pour résoudre un problème, sont rassemblées dans ce qu'on appelle des procédures semblables à nos fonctions. Les fonctions opèrent en principe au mieux avec les variables qui leur ont été transmises comme arguments ou, dans le cas défavorable, avec des variables globales qui ont été déclarées au début d'un programme. Celles-ci sont visibles dans tout le programme et chacun peut les modifier à sa convenance. Toutefois, cela comporte certains risques et c'est actuellement, tout bien pesé, la plus mauvaise variante pour traiter des variables ou des données. Variables et fonctions ne forment aucune unité logique et vivent quasiment les unes à côté des autres dans le code, sans avoir aucun rapport direct entre elles.



Venons-en maintenant à la programmation orientée objet. Elle comporte une structure appelée classe. On peut dire pour simplifier que les servent de containers pour des variables membres ou des méthodes.



La classe enveloppe ses membres, appelés *members* dans la POO, à la manière d'un grand manteau. On ne peut en principe accéder aux membres qu'en passant par la classe.

Construction d'une classe

Mais qu'est-ce donc qu'une classe ? Si vous n'avez jamais eu affaire aux langages de programmation C++, Java et même C# pour ne citer que ceux-là, le terme ne vous en dira pas plus qu'un caractère chinois pour moi. Mais la chose est en fait assez facile à comprendre. Si vous regardez encore une fois le dernier graphique, vous verrez qu'une classe a vocation d'entourer et ressemble en quelque sorte à un container. Une classe est définie par le mot-clé `class`, suivi du nom qu'on lui a donné. Suit une paire d'accolades, que vous avez pu voir dans d'autres structures comme une boucle `for` et qui amène la formation d'un bloc. L'accolade finale est suivie d'un point-virgule.

Mot-clé Nom de classe

```
class name{  
};
```

◀ **Figure 9-3**
Définition générale d'une classe

Comme je vous l'ai déjà dit, la classe est composée de différents membres sous forme de variables membres et de méthodes, qui se fondent, selon la définition de cette classe, en une unité. La POO offre diverses possibilités de réglementer l'accès aux membres.

Oui, mais à quoi bon cette réglementation ? Quand je définis une variable ou plutôt une variable membre dans une classe, je veux pouvoir y accéder n'importe quand. À quoi sert cela s'il si je ne peux plus ensuite accéder à la classe ? Ou peut-être ai-je mal compris le principe ?



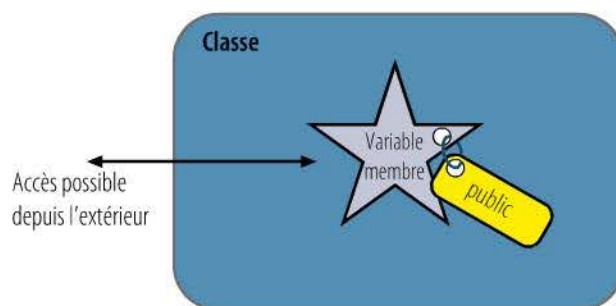
Vous avez bien compris le principe, appelé d'ailleurs encapsulation. On peut protéger certains membres contre le monde extérieur, de telle sorte qu'ils ne soient pas directement accessibles depuis l'extérieur de la classe. Le mot directement est ici important. Il existe bien sûr des possibilités d'y accéder. Ce sont les méthodes qui, par exemple, s'en chargent. Mais vous devez vous demander quel est le sens de tout cela.

Exact ! On peut donc toujours influencer directement sur les variables membres, n'est-ce pas ?



Bon ! Je pense que les figures suivantes vous permettront de mieux comprendre le principe.

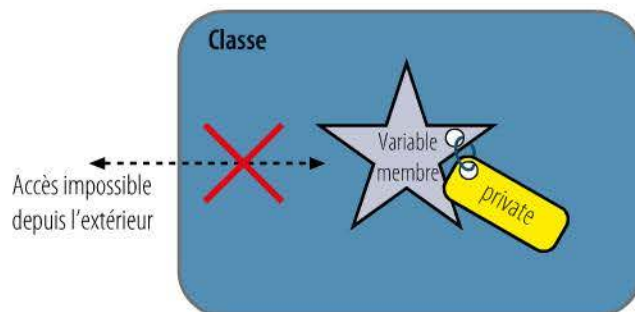
Figure 9-4 ►
Accès à une variable membre
de la classe



L'accès à la variable membre de la classe depuis l'extérieur est ici autorisé, car elle a reçu une certaine étiquette appelée modificateur d'accès. Elle a pour nom ici `public` et signifie à peu près ceci : l'accès est autorisé au public et tout un chacun peut s'en servir à sa guise.

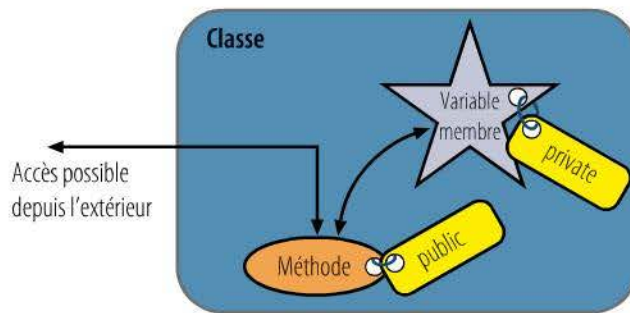
Imaginez maintenant le scénario suivant : une variable membre doit piloter un moteur pas-à-pas, la valeur indiquant l'angle. Seuls des angles compris entre 0° et 359° sont cependant admis. Toute valeur inférieure ou supérieure peut compromettre l'exécution du sketch, si bien que le servo n'est plus commandé correctement. Quand vous donnez libre accès à une variable membre au moyen du modificateur `public`, aucune validation ne peut avoir lieu. Ce qui a été enregistré une fois produit inmanquablement une réaction qui n'est pas forcément correcte. La solution du problème consiste à isoler les variables membres grâce à un modificateur d'accès `private` (privé). C'est le principe de l'encapsulation déjà évoqué qui est utilisé ici.

Figure 9-5 ►
Pas d'accès à une variable membre
de la classe



C'est bien beau tout ça !
Mais comment fait-on pour accéder à la variable membre ?

On y accède avec une méthode qui contient également un modificateur d'accès. Il doit cependant être `public` pour que l'accès fonctionne depuis l'extérieur. Le tout se présente comme sur la figure 9-6.



◀ **Figure 9-6**
Accès à une variable membre
de la classe par la méthode

On voit clairement que l'accès à la variable membre passe par la méthode, ceci étant un avantage et non pas un inconvénient. Vous pouvez maintenant procéder à la validation dans la méthode, seules des valeurs admises étant alors communiquées à la variable membre.

Mais pourquoi la méthode a-t-elle accès à la variable membre privée ?
Je croyais que c'était impossible.

Le modificateur d'accès `private` signifie que l'accès depuis l'extérieur de la classe est impossible. Mais des membres de la classe comme les méthodes peuvent accéder à des membres déclarés `private`. Ils appartiennent tous à la classe et sont donc librement accessibles au sein de celle-ci. Pour faire court, les modificateurs d'accès gèrent l'accès aux membres de la classe.



Modificateur d'accès	Description
<code>public</code>	L'accès aux variables membres et aux méthodes est possible depuis n'importe où dans le sketch. De tels membres constituent une interface publique de la classe.
<code>private</code>	L'accès aux variables membres et aux méthodes est réservé aux membres de la même classe.

◀ **Tableau 9-2**
Modificateurs d'accès
et leur signification

Si vous voulez ajouter une classe à votre projet Arduino, mieux vaut créer un nouveau fichier se terminant par `.cpp` pour y stocker la définition de la classe. Vous verrez bientôt comment dans l'exemple concret de la bibliothèque-dé. Encore un peu de patience.

Une classe a besoin d'aide

Nous avons vu ce qu'une classe réalise et comment la créer en bonne et due forme. Mais je ne vous ai pas encore dit que la classe avait besoin d'un autre fichier très important. Celui-ci est appelé fichier d'en-tête et contient les déclarations (informations initiales ou préalables) pour la classe à concevoir. Si vous créez des variables membres

ou des méthodes en C++, vous devez impérativement les faire connaître auprès du compilateur avant de les utiliser. C'est chose faite en définissant les variables et les prototypes de fonction ou de méthode. Le fichier en question renferme également les consignes relatives aux modificateurs d'accès public et private.

La construction formelle du fichier d'en-tête ressemble à celle de la définition de classe, à ceci près qu'il ne contient pas de formulation de code. Autrement dit, seules les signatures des méthodes sont mentionnées. Une signature se compose uniquement des informations initiales avec le nom de la méthode, le type d'objet renvoyé et la liste des paramètres. La construction générale est la suivante :

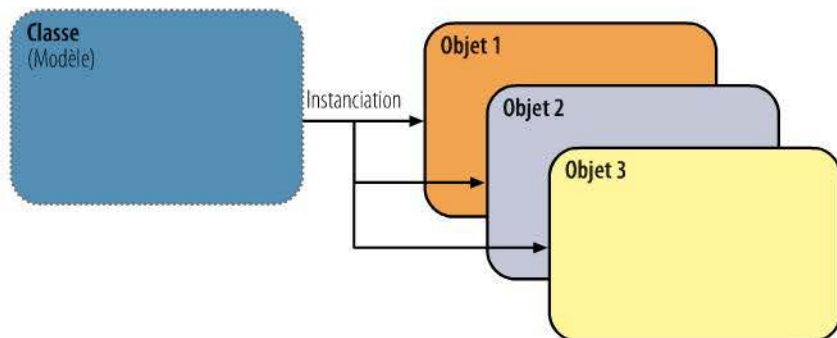
```
class Nom{  
public:  
    //Membre public  
private:  
    //Membre privé  
};
```

La zone définissant le membre public vient après le mot-clé public suivi d'un deux-points. La zone définissant le membre privé vient après le mot-clé private, qui est suivi lui aussi d'un deux-points. Le fichier d'en-tête reçoit l'extension de nom .h.

Une classe devient un objet

Une fois créée par sa définition, une classe peut servir, comme lors de la déclaration d'une variable, de nouveau type de donnée. Ce procédé est appelé instantiation. Du point de vue du logiciel, la définition d'une classe ne veut pas dire qu'on a créé réellement un objet. Elle n'est qu'une sorte de modèle ou plan de construction qu'on peut utiliser pour concevoir un ou plusieurs objets.

Figure 9-7 ▶
De la classe à l'objet



L'instanciation se fait de la manière suivante :

```
Nomclasse Nomobjet();
```

Holà, il y a quelque chose qui ne va pas. Vous avez dit que l'instanciation d'un objet avait tout de la déclaration de variable ordinaire. Mais je vois encore une paire de parenthèses derrière le nom que vous avez donné à l'objet. Est-ce une double faute de frappe ? Sûrement pas. Qu'est-ce que c'est alors ?

Bien vu, Arthus ! Elle a naturellement son utilité. Une partie de ce projet va lui être consacrée car elle est extrêmement importante pour l'instanciation.



Initialiser un objet : qu'est-ce qu'un constructeur ?

Une définition de classe contient en principe quelques variables membres qui serviront après l'instanciation. Pour qu'un objet puisse présenter un état initial bien défini, il s'avère judicieux de l'initialiser en temps voulu. Quel meilleur moment pour cette initialisation que directement lors de l'instanciation ? Aucun risque ainsi qu'elle soit oubliée et pose plus tard problème lors de l'exécution du sketch. Mais comment faire pour initialiser un objet ? Le mieux est d'employer une méthode qui prend cette tâche en charge.

Il faut donc indiquer lors de l'instanciation une méthode à laquelle on donne certaines valeurs comme arguments. Mais comment savoir quelle méthode prendre ?

Exact, Arthus ! Il faut appeler une méthode et lui donner le cas échéant quelques valeurs en passant. Mais quel nom lui donner ? La solution est à la fois très simple et géniale. La méthode pour initialiser un objet porte le même nom que la classe. Cette méthode étant très spéciale, elle porte aussi un nom à elle. On l'appelle constructeur. Comme son nom l'indique, elle construit en quelque sorte l'objet. Mais puisqu'il n'est pas impérativement nécessaire d'initialiser dès le début un objet avec certaines valeurs, elle n'a pas forcément de liste de paramètres. Elle se comporte alors comme une méthode à laquelle aucun paramètre n'est donné et qui n'a que la paire de parenthèses vide. Ceci répond à votre question concernant la paire de parenthèses que vous avez vue dans l'instanciation.



Vous ne devez en aucun cas l'omettre ou l'oublier. Il me faut maintenant être un peu plus concret pour vous en montrer la syntaxe. Voici le contenu du fichier d'en-tête de notre bibliothèque-dé :

```
class Dice{
public:
    Dice(); //Constructeur
    // ...
private:
    // ...
};
```

Sous le modificateur d'accès `public` se trouve le constructeur qui porte le même nom que la classe. Il présente une paire de parenthèses vide, d'où son nom de *constructeur standard*.



Ne venez-vous pas de dire qu'on peut donner des arguments à un constructeur tout comme à une méthode pour initialiser l'objet ? La paire de parenthèses vide indique pourtant que le constructeur ne peut recevoir aucune valeur. Comment est-ce possible ? La deuxième chose que j'ai remarquée concerne le type présumé d'objet retourné par une méthode. Vous ne l'avez pas indiqué pour le constructeur. Pourquoi ?

Vous avez tout à fait raison Arthus quand vous dites que le constructeur ne peut accueillir aucune valeur sous cette forme. C'est une bonne introduction au prochain thème. Mais je vais d'abord répondre à votre question sur le type d'objet renvoyé manquant. Si une méthode renvoie une valeur à son appelant, le type de donnée en question doit naturellement être indiqué. Si aucun renvoi n'est prévu, le mot-clé `void` est utilisé. Revenons maintenant à notre constructeur. Il est appelé, non pas explicitement par une ligne d'instruction, mais implicitement par l'instanciation d'un objet. C'est pour cette raison que rien ne peut être retourné à un appelant et que le constructeur n'a pas même le type de renvoi `void`.

La surcharge

Ce que je vais vous dire là peut sembler déroutant à première vue : on peut définir un constructeur et bien entendu également des méthodes plusieurs fois avec le même nom.



Je vous avoue que j'ai du mal à le croire. C'est pourtant contraire au principe de clarté. Si par exemple une méthode apparaît deux fois avec le même nom dans un sketch, comment le compilateur peut-il savoir laquelle des deux est appelée ?

C'est vrai Arduus. Mais il n'y a pas que le nom qui soit déterminant, il y a aussi la fameuse *signature* dont je vous ai parlé plus haut. L'exemple suivant montre deux constructeurs acceptables qui portent le même nom mais dont les signatures diffèrent :

```
Dice();  
Dice(int, int, int, int);
```

Le premier constructeur représente le constructeur standard et sa paire de parenthèses vide, qui ne peut accueillir aucun argument. Le deuxième porte une toute autre signature car il peut recevoir quatre valeurs du type `int`. Vous pouvez alors choisir entre deux variantes pour instancier un objet `Dice` :

```
Dice myDice();
```

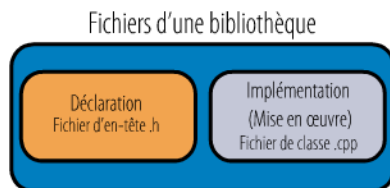
ou :

```
Dice myDice(8, 9, 10, 11);
```

Le compilateur est assez intelligent pour savoir quel constructeur il doit appeler.

La bibliothèque-dé

Toute cette introduction était nécessaire pour bien vous faire comprendre la création d'une bibliothèque Arduino. Le deuxième projet de dé servira de base pour constituer une bibliothèque. Il s'agit d'une variante améliorée avec commande des groupes de LED. Deux fichiers sont donc nécessaires pour réaliser la bibliothèque.



Le fichier d'en-tête

Commençons par le fichier d'en-tête, qui ne contient que les informations de prototype et ne présente aucune information de code explicite. Occupons-nous d'abord des membres de la classe qui sont nécessaires. Pour piloter les groupes de LED, il faut quatre broches numériques commandées par les variables membres :

- pinGroupA ;
- pinGroupB ;
- pinGroupC ;
- pinGroupD.

Ces informations seront transmises au moment de l'instanciation au constructeur, qui possède quatre paramètres du type `int`. Les variables membres sont déclarées privées (`private`) car elles ne sont traitées qu'en interne par une méthode appelée `roll`, qui n'a aucun argument et qui ne retourne rien. La classe reçoit le nom évocateur de Dice (dé).

```
#ifndef Dice_h
#define Dice_h

#if ARDUINO < 100
#include <WProgram.h>
#else
#include <Arduino.h>
#endif

class Dice{
public:
    Dice(int, int, int, int); //Constructeur
    void roll(); //Méthode pour lancer le dé
private:
    int GroupA; //Variable membre pour groupe de LED A
    int GroupB; //Variable membre pour groupe de LED B
    int GroupC; //Variable membre pour groupe de LED C
    int GroupD; //Variable membre pour groupe de LED D
};
#endif
```

Quelques informations supplémentaires méritant une explication ont été ajoutées à la définition de la classe. La classe tout entière a été enveloppée dans la structure suivante :

```
#ifndef Dice_h
#define Dice_h
...
#endif
```

Des inclusions multiples étant possibles quand il y a du code imbriqué, un moyen a été trouvé pour les empêcher et éviter une double compilation. Cette précaution a pour but de garantir une inclusion unique du fichier d'en-tête. Les instructions `#ifndef`, `#define` et `#endif` sont des instructions de prétraitement. `#ifndef`, qui introduit une

compilation conditionnelle, est la forme abrégée de *if not defined* qui signifie « si non défini ». Si le terme `Dice_h` (nom du fichier d'en-tête avec un tiret bas) – appelé macro – n'a pas encore été défini, faites-le maintenant et exécutez les instructions dans le fichier d'en-tête. Si ce dernier était appelé une deuxième fois, la macro serait placée sous le nom et cette partie de la compilation serait rejetée. Les instructions `include` sont nécessaires pour faire connaître à la bibliothèque les types de données ou constantes propres à Arduino (par exemple : `HIGH`, `LOW`, `INPUT` ou `OUTPUT`).

```
#if ARDUINO < 100
#include <Wprogram.h>
#else
#include <Arduino.h>
#endif
```

Il y a ici un truc : toutes les versions Arduino antérieures à la version 1.0 nécessitent un fichier d'en-tête nommé `Wprogram.h` pour utiliser par exemple lesdites constantes. Il sert à bien autre chose encore, mais restons-en là pour le moment. Le numéro de version d'Arduino figure dans la définition du terme `ARDUINO` et peut donc être lu pour l'environnement de développement actuel. C'est ce que nous faisons dans notre cas. Si le numéro de version est `<100` (soit la version 1.00), l'ancien fichier d'en-tête `Wprogram.h` doit être intégré. Sinon, le nouveau fichier d'en-tête `Arduino.h` sera utilisé. Cette modification des fichiers d'en-têtes fait souvent râler et je dois dire que cette adaptation ne m'enchanté pas non plus.

Pouvez-vous me dire pourquoi le constructeur n'indique que le type de donnée pour les paramètres et pourquoi le nom de la variable correspondante est absent ?

Cela tient au fait que nous n'avons besoin ici que des informations de prototype. Le code en question apparaît plus tard avec une extension `.cpp` dans le fichier de classe.



Le fichier de classe

La véritable mise en œuvre du code est effectuée au moyen du fichier de classe présentant l'extension `.cpp` :

```
#if ARDUINO < 100
#include <WProgram.h>
#else
#include <Arduino.h>
#endif
```

```

#include "Dice.h"
#define WAITTIME 20

//Constructeur paramétré
Dice::Dice(int A, int B, int C, int D){
    GroupA = A;
    GroupB = B;
    GroupC = C;
    GroupD = D;
    pinMode(GroupA, OUTPUT);
    pinMode(GroupB, OUTPUT);
    pinMode(GroupC, OUTPUT);
    pinMode(GroupD, OUTPUT);
}
//Méthode pour lancer le dé
void Dice::roll(){
    int number = random(1, 7);
    digitalWrite(GroupA, number%2!= 0?HIGH:LOW);
    digitalWrite(GroupB, number>1?HIGH:LOW);
    digitalWrite(GroupC, number>3?HIGH:LOW);
    digitalWrite(GroupD, number==6?HIGH:LOW);
    delay(WAITTIME); //Ajouter une courte pause
}

```

Pour que la liaison vers le fichier d'en-tête précédemment créé soit possible, référence est faite à ce dernier au moyen de l'instruction include :

```

#include "Dice.h"

```

Son intégration intervient lors de la compilation. include est ici également nécessaire pour pouvoir utiliser ce qu'on appelle les éléments de langage Arduino.

```

#if ARDUINO < 100
#include <Wprogram.h>
#else
#include <Arduino.h>
#endif

```

Passons maintenant au code, qui contient la mise en œuvre proprement dite. Commençons par le constructeur :

```

Dice::Dice(int A, int B, int C, int D){
    GroupA = A;
    GroupB = B;
    GroupC = C;
    GroupD = D;
}

```

```
pinMode(GroupA, OUTPUT);
pinMode(GroupB, OUTPUT);
pinMode(GroupC, OUTPUT);
pinMode(GroupD, OUTPUT);
}
```

Vous avez sûrement remarqué que la méthode `roll` était légèrement différente de celle du montage n° 8.

Oui, aucune LED n'a été éteinte avant de commander l'allumage des nouvelles. Je ne vois pas bien pourquoi !

C'est vrai, Arduus ! Et ce n'est pas grave puisque les différents groupes de LED sont commandés par l'opérateur conditionnel ? que vous avez déjà rencontré dans le montage n° 7 sur la machine à états. Cet opérateur retourne soit `LOW` soit `HIGH` quand la condition a été évaluée, si bien que le groupe de LED correspondant a toujours le bon niveau et n'a pas besoin d'être remis auparavant sur `LOW`. Une autre chose susceptible de vous étonner est le préfixe `Dice::` qui précède aussi bien le nom de la structure que la méthode `roll`.

Il s'agit du nom de la classe, qui permet au compilateur de savoir à quelle classe la définition de la méthode appartient. Cette dernière est qualifiée par cette notation. L'objet `dé` que nous voulons créer devant commander quatre groupes de LED, il est préférable de transmettre ces informations au moment de l'instanciation. Ce serait bien entendu possible aussi après la génération de l'objet, en utilisant une méthode distincte que nous appellerions par exemple `Init`. Mais le risque est grand que cette étape soit oubliée. C'est pourquoi le constructeur a été inventé. Voyons maintenant le sketch qui utilise cette bibliothèque.



Création des fichiers nécessaires

Je vous propose de programmer les deux fichiers de la bibliothèque `.h` et `.cpp` indépendamment de l'environnement de développement Arduino. Il existe pour ce faire de nombreux éditeurs, par exemple Notepad++ ou Programmers Notepad. Les deux fichiers sont stockés dans un répertoire au nom évocateur, par exemple `Dice (dé)`, qui est copié une fois prêt dans le dossier des bibliothèques Arduino.

...\\arduino-1.x.y\\libraries

L'environnement de développement Arduino est ensuite redémarré et la programmation du sketch peut commencer.

Mise en surbrillance de la syntaxe pour une nouvelle bibliothèque

Des types de données élémentaires (par exemple : `int`, `float` ou `char`) ou d'autres mots-clés (par exemple : `setup` ou `loop`) sont signalés en couleurs par l'environnement de développement. Il est possible, lors de la création de ses propres bibliothèques, de faire connaître à l'IDE des noms de classes ou de méthodes, pour qu'ils apparaissent alors aussi en couleurs. Un fichier nommé `keywords.txt`, ayant obligatoirement une syntaxe spéciale, doit être créé pour que cela fonctionne.

Commentaires

Des commentaires explicatifs sont introduits par le signe `#` (dièse) :

```
# Ceci est un commentaire
```

Types de données et classes (KEYWORD1)

Les types de données mais aussi les noms de classes figurent en orange et doivent être définis en respectant la syntaxe suivante :

```
Nomclasse KEYWORD1
```

Méthodes et fonction (KEYWORD2)

Méthodes et fonctions apparaissent en marron et doivent être définies en respectant la syntaxe suivante :

```
Méthode KEYWORD2
```

Constantes (LITERAL1)

Les constantes sont en bleu et doivent être définies en respectant la syntaxe suivante :

```
Constante LITERAL1
```

Voici maintenant le contenu du fichier `keywords.txt` de notre bibliothèque-dé :

```
#-----  
# Affectation des couleurs pour la bibliothèque Dice  
#-----  
  
#-----  
# KEYWORD1 pour types de données ou classes  
#-----
```

Dice KEYWORD1

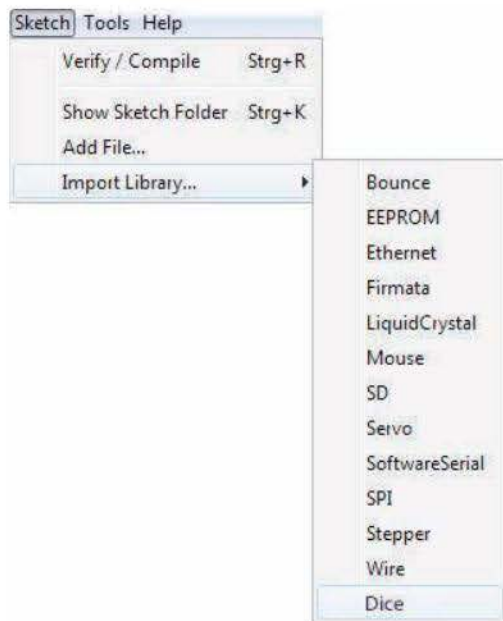
```
#-----  
# KEYWORD2 pour méthodes et fonctions  
#-----
```

roll KEYWORD2

```
#-----  
# LITERAL1 pour constantes  
#-----
```

Utilisation de la bibliothèque

Le fait que la bibliothèque-dé se trouve dans le répertoire indiqué précédemment vous permet de la retrouver dans la dernière entrée du menu (voir figure 9-8).



◀ **Figure 9-8**
Importer la bibliothèque-dé

Le terme importer est quelque peu mal venu puisque absolument rien n'est importé à ce moment précis. Seule la ligne suivante est ajoutée dans votre fenêtre de sketch :

```
#include <Dice.h>
```

La ligne `include` est impérative pour pouvoir accéder à la fonctionnalité de la bibliothèque-dé. Comment le compilateur saurait-il sinon à

quelle bibliothèque il doit accéder ? Les différentes bibliothèques disponibles ne sont pas incluses par l'opération du Saint-Esprit ! Passons maintenant à l'instanciation, qui élève la définition de classe au statut d'objet réel. L'objet créé `myDice` est également appelé variable d'instance. Ce terme revient souvent dans la littérature.

```
Dice myDice(8, 9, 10, 11);
```

Les valeurs transmises 8, 9, 10 et 11 figurent les broches numériques auxquelles les groupes de LED sont reliés. Un objet dé a ainsi été initialisé de manière à pouvoir opérer en interne quand la méthode pour lancer le dé est appelée. Les arguments sont transmis dans l'ordre indiqué.

```
Dice myDice(8, 9, 10, 11);  
  
Dice::Dice(int A, int B, int C, int D)  
{  
    ...  
}
```

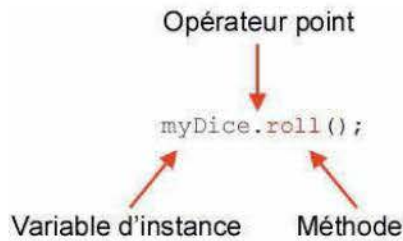
Ils sont stockés dans les variables locales A, B, C et D, qui sont à leur tour transmises aux variables membres `GroupA`, `GroupB`, `GroupC` et `GroupD`. Vient ensuite l'appel de la méthode à condition qu'un potentiel HIGH soit appliqué à l'entrée numérique, appel qui peut se faire par le bouton-poussoir raccordé.

```
void setup(){  
    pinMode(13, INPUT); //Pas obligatoire - oui mais pourquoi ?  
}  
void loop(){  
    if(digitalRead(13) == HIGH)  
        myDice.roll( );  
}
```

On voit ici aussi que la mise en surbrillance de la syntaxe fonctionne puisque le nom de classe et la méthode sont en couleurs. La méthode `roll` étant un membre de la définition de classe `Dice`, une liaison vers la classe doit être établie en cas d'appel de celle-ci. Un appel par :

```
roll();
```

provoquerait ici une erreur. La relation est établie par l'*opérateur point* inséré entre classe et méthode et faisant office de lien.



Vous serez amené plus tard à programmer l'une ou l'autre bibliothèque qui pourra vous être utile à vous ou à d'autres programmeurs. Vous acquerez alors un peu de pratique dans la manipulation du langage C++ et la programmation orientée objet.

Pour aller plus loin



Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- programmation orientée objet ;
- POO ;
- Arduino Library.

Qu'avez-vous appris ?

- Je reconnais que les choses sont devenues un peu plus difficiles dans ce projet, mais le jeu en vaut la chandelle. Vous en savez maintenant plus sur le paradigme de programmation orientée objet.
- Vous savez bien faire la différence entre une classe et un objet.
- Dans la POO, méthode remplace fonction et variable membre remplace variable.
- Le constructeur est une méthode avec une tâche particulière. Il initialise l'objet de manière à obtenir un état de départ bien défini.
- Vous connaissez les informations de code qu'un fichier d'en-tête ou .cpp doit contenir. Les différents modificateurs d'accès public ou private réglementent l'accès aux membres de l'objet, private assurant l'encapsulation des membres.
- L'objet instancié à partir d'une classe est également appelé variable d'instance.
- Un opérateur point, ajouté après le nom de la variable d'instance et servant quasiment de lien entre les deux, est utilisé pour accéder aux variables membres et aux méthodes.

- Vous savez comment créer une bibliothèque Arduino et à quel endroit copier celle-ci dans le système de fichiers pour pouvoir y accéder à tout moment.
- Vous avez enfin appris à configurer certaines méthodes en tant que mots-clés avec un marquage couleur.

Des détecteurs de lumière

Au sommaire :

- la mesure d'une quantité de lumière par une photorésistance (LDR) ;
- la programmation de plusieurs broches comme sortie (OUTPUT) ;
- l'interrogation d'une entrée analogique avec l'instruction `analogRead()` ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- la communication avec Processing ;
- un exercice complémentaire.

Comment fonctionne un détecteur de lumière ?

Dans ce projet, nous allons construire un circuit capable de réagir à des influences ou réalités extérieures. Les valeurs environnementales vraiment marquantes qui agissent sur nous sont la température et la luminosité. Toutes deux peuvent être ressenties différemment par les êtres humains et sont des impressions subjectives. L'un aura bien chaud tandis que l'autre aura la chair de poule. Il existe bien entendu des appareils ou détecteurs qui mesurent la température et la luminosité de manière objective. Notre prochain circuit sera consacré à la luminosité, que nous entendons mesurer au moyen d'une photorésis-

tance ou résistance photosensible, également appelée LDR (*Light Dependent Resistor*).

Il s'agit d'un semi-conducteur, dont la résistance dépend de la lumière. Plus la quantité de lumière atteignant la LDR est élevée, plus la résistance est faible. Notre circuit doit commander, en fonction de la valeur de luminosité, une rangée de LED qui éclaire alors plus ou moins. Le circuit ressemble à celui de notre séquenceur de lumière, à ceci près que les différentes LED ne sont pas commandées l'une après l'autre par une boucle mais par une logique qui évalue la luminosité sur la résistance photosensible.

Composants nécessaires



1 LDR



10 LED rouges



10 résistances de 330 Ω



1 résistance de 10 k Ω



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

```
int pin[] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11}; //Tableau des broches
int analogPin = 0; //Broche de l'entrée analogique
int analogValue = 0; //Stocke la valeur analogique mesurée

void setup(){
  for(int i = 0; i < 10; i++){
    pinMode(pin[i], OUTPUT);
  }
}
```

```

void loop(){
    analogValue = analogRead(analogPin) ;
    controllLEDs(analogValue);
}

//Fonction pour commander les LED
void controllLEDs(int value){
    int bargraphValue = map(value, 0, 1023, 0, 9);
    for(int i = 0; i < 10; i++)
        digitalWrite(pin[i], (bargraphValue >= i)?HIGH:LOW);
}

```

Revue de code

Du point de vue logiciel, les variables du tableau 10-1 sont nécessaires à notre atelier.

Variable	Objet
pin[]	Stocke les numéros des broches pour commander les 10 LED.
analogPin	Numéro de broche pour l'entrée analogique
analogValue	Stocke la valeur analogique mesurée.

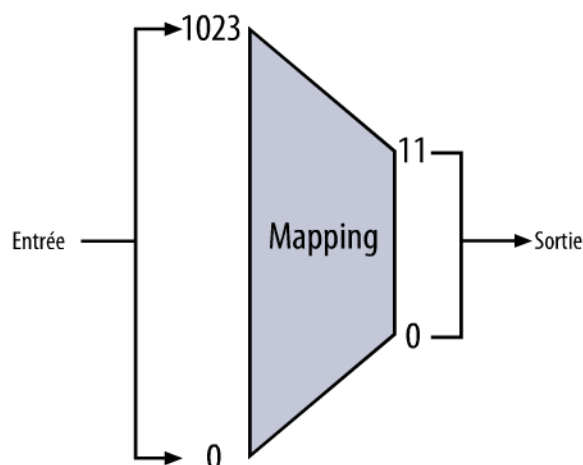
◀ **Tableau 10-1**

Variables nécessaires et leur objet

Le grand nombre de LED impose d'utiliser un tableau qui est stocké dans `pin[]`. La fonction `loop` lit continuellement la valeur à l'entrée analogique broche A0. La fonction `controllLEDs` est ensuite appelée avec la valeur mesurée, et se charge de commander les différentes LED. Chaque entrée analogique a une résolution de 10 bits et des valeurs comprises entre 0 et 1023 peuvent donc y être mesurées. N'utilisant cependant que 10 LED pour notre exemple, nous devons convertir le domaine des valeurs d'entrée trop grand en un domaine des valeurs de sortie adapté allant de 0 à 9 (10 LED).

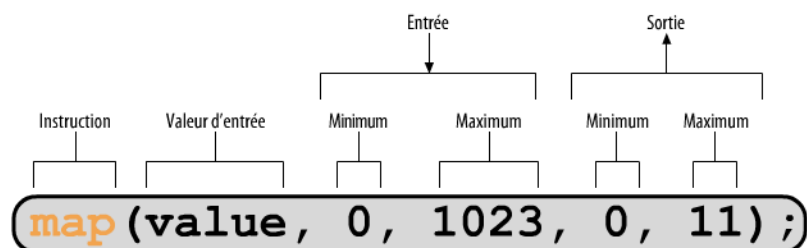
Ce code me donne l'occasion d'introduire une nouvelle instruction très intéressante qui permet de transposer un domaine de valeurs dans un autre domaine. Cette instruction est notée `map` (abréviation de *mapping* : transposition). L'image 10-1 illustre le processus de mapping réalisé par cette instruction. Il doit convertir le domaine des valeurs d'entrée, qui s'étend de 0 à 1023, en un domaine de valeurs de sortie allant de 0 à 11.

Figure 10-1 ►
Principe du mapping



La syntaxe de l'instruction `map` est la suivante :

Figure 10-2 ►
Instruction `map`

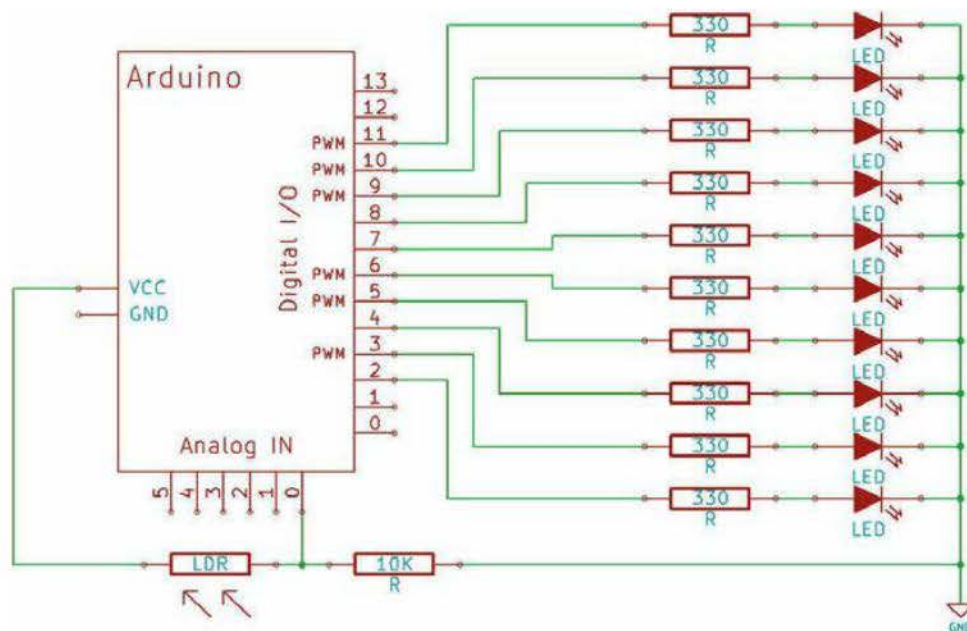


La valeur renvoyée par l'instruction `map` appartient au nouveau domaine de valeurs défini.

Ici, le résultat est consigné dans la variable `bargraphValue`. Chaque LED est ensuite commandée en la comparant à la valeur `bargraphValue` alors déterminée. Si cette valeur est supérieure ou égale au numéro de la LED en question, elle est aussitôt mise sur HIGH ; sinon, elle est mise sur LOW. Plus la valeur est élevée, plus le nombre de LED allumées est important.

Schéma

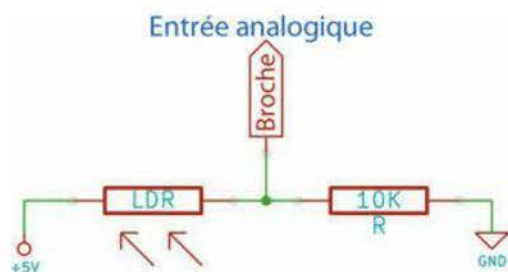
Le schéma ressemble à s'y méprendre à celui du séquenceur de lumière, à ceci près qu'il dispose en plus d'une extension lui permettant de mesurer l'intensité lumineuse.



Vous savez déjà ce qu'est un diviseur de tension.

▲ Figure 10-3

Circuit pour mesurer l'énergie lumineuse



Les deux résistances (LDR et 10K) forment un diviseur de tension, la prise de tension intermédiaire est raccordée à l'entrée analogique de la carte Arduino. Le rapport des résistances et donc des tensions varie en fonction de la luminosité existante sur la LDR. La tension la plus élevée se trouve naturellement aux bornes de la plus grande résistance. Si la résistance de la LDR diminue sous l'influence d'une augmentation de la lumière, la tension à ses bornes diminue. Cela veut donc dire que la tension aux bornes de la résistance de 10K est plus élevée, laquelle se retrouve sur l'entrée analogique. Une valeur plus élevée est mesurée sur cette broche, autrement dit plus de LED s'allument. Ce processus s'inverse si moins de lumière parvient à la LDR.



Encore une fois, s'il vous plaît. Si la tension la plus élevée se trouve aux bornes de la plus grande résistance, la tension la plus élevée doit se trouver à l'entrée analogique quand la LDR s'assombrit.

Je comprends votre problème, Arduus. Je vais me servir de la figure suivante pour vous expliquer.

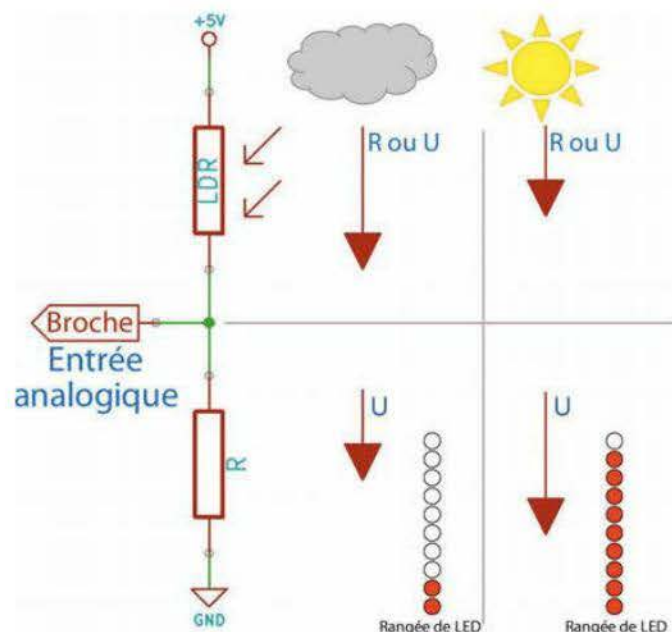


Figure 10-4 ►
Résistances et tensions
selon l'ensoleillement

La longueur des flèches indique la grandeur de la tension. Si le ciel est couvert, la résistance ou plutôt la tension est élevée sur la LDR. Mais si le soleil brille, résistance et tension sont faibles. Comme le diviseur de tension est alimenté sous 5 V, il ne reste que la différence de tension aux bornes de la résistance située à la partie inférieure. Cette tension est mesurée entre l'entrée analogique et la masse.

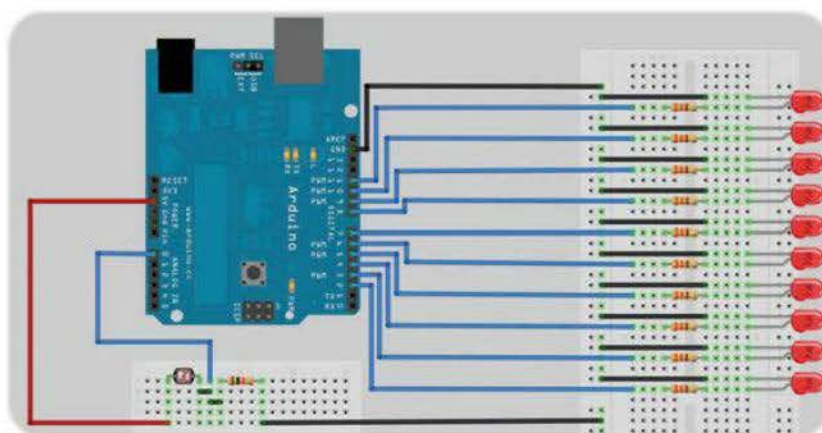


Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

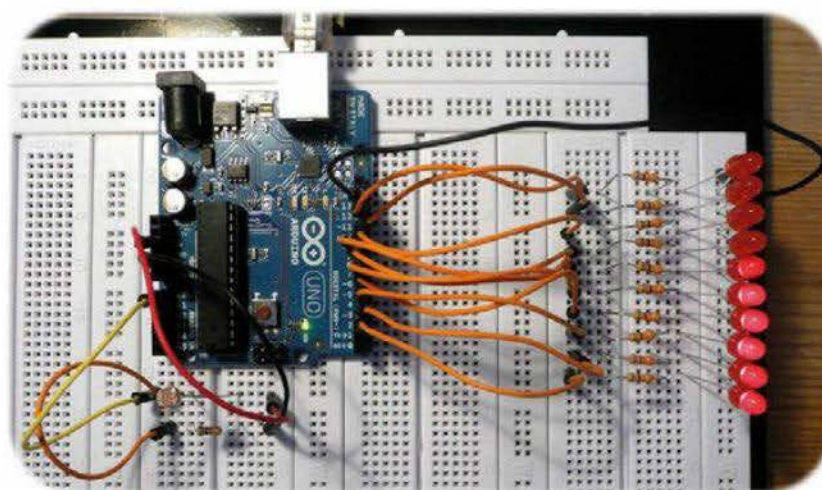
- LDR ;
- photorésistance ;
- résistance photosensible.

Réalisation du circuit



◀ **Figure 10-5**
Réalisation du circuit avec Fritzing

À droite de la plaque d'essais se trouvent les 10 LED pour afficher l'intensité lumineuse, le diviseur de tension se trouve quant à lui, avec sa LDR et sa résistance fixe, sous la carte Arduino.



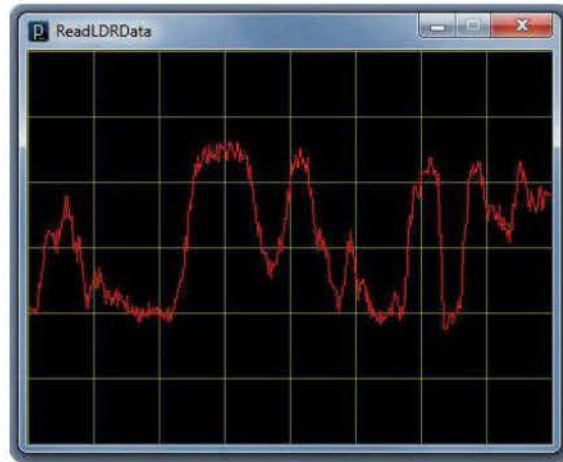
◀ **Figure 10-6**
Réalisation du circuit détecteur de lumière

Nous devenons communicatifs

Il est certes intéressant à mes yeux d'observer le cycle des LED sous différentes conditions de lumière, mais le déroulement chronologique reste difficile à voir sur une longue période. Aussi voudrais-je vous présenter ici un projet qui vous plaira sûrement, puisque agréable à regarder.

Le langage de programmation Processing s'impose quand il s'agit de générer des graphiques. Vous trouverez l'environnement de développement pour le langage de programmation Processing sur le site Internet <http://processing.org>. Le côté pratique de la chose est que, tout comme pour Arduino, il suffit de décompresser le fichier téléchargé dans un répertoire. Aucune installation n'est à faire. Ce qui prendrait un temps fou avec des langages de programmation tels que C++ ou C# sera plus vite fait et sera moins fastidieux avec Processing. Je vous montre d'emblée le résultat dans la fenêtre graphique de Processing pour que vous sachiez à quoi vous attendre.

Figure 10-7 ►
Courbe des valeurs de l'énergie lumineuse dans la fenêtre graphique de Processing



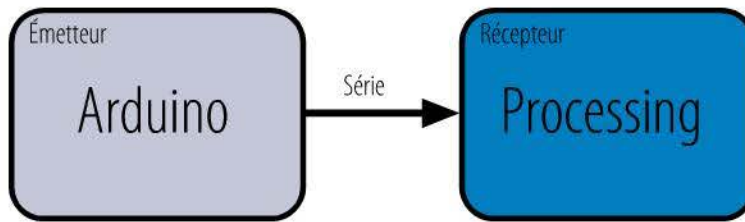
Les valeurs affichées sont actualisées en permanence, la courbe évoluant de droite à gauche dans la fenêtre. Les nouvelles valeurs apparaissent à droite et les anciennes valeurs disparaissent à gauche de la fenêtre.



Pouvez-vous me dire comment deux langages de programmation différents font pour échanger des données ?

J'allais y venir, Ardus. Une base commune leur permettant de se comprendre doit être définie. L'interface série vous dit sûrement déjà quelque chose. Tout langage de programmation ou presque a dans son vocabulaire des instructions pour envoyer ou recevoir par cette interface.

Notre exemple montre un émetteur et un récepteur. La communication est unidirectionnelle, autrement dit se fait dans un seul sens. Certes, l'interface est capable de communiquer presque simultanément dans les deux sens, mais nous nous contenterons d'un seul.



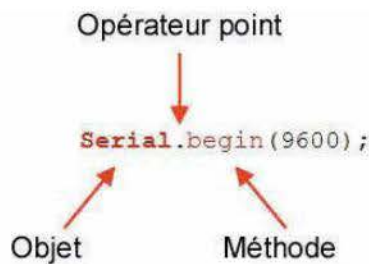
Tout ce que votre carte Arduino doit faire c'est enregistrer les valeurs mesurées et envoyer les données via l'interface série. Plus vite à dire qu'à faire ?! Non, pas du tout car la plupart du travail de calcul se fait du côté de Processing. Voyons d'abord du côté de l'émetteur ce qu'Arduino doit faire.

Arduino l'émetteur

Côté matériel, il ne vous faut que le diviseur de tension avec sa LDR et sa résistance de 10 k Ω fixe, connecté à l'entrée analogique broche 0, pour envoyer les valeurs de luminosité à l'interface série.

```
void setup(){  
  Serial.begin(9600);  
}  
  
void loop(){  
  Serial.println(analogRead(0));  
}
```

Voyons maintenant ce que ce bout de code donne. Dans la fonction `setup`, l'interface série est préparée pour la transmission. Vous avez eu vos premiers contacts avec la programmation orientée objet lors de la création de votre propre bibliothèque dans le montage n° 9. L'interface série est considérée comme étant un objet logiciel nommé `Serial`. Vous disposez de quelques méthodes, que nous entendons maintenant utiliser.



La méthode pour initialiser l'interface a pour nom `begin` et reçoit une valeur qui détermine la vitesse de la transmission. Il s'agit dans notre cas de 9600. L'unité de mesure est ici le baud, le nombre indiquant la cadence. 1 baud signifie 1 changement d'état par seconde. Lisez la littérature technique ou allez sur Internet pour d'autres informations. La deuxième méthode que nous utiliserons se nomme `println`. Elle envoie la valeur qui lui a été transmise à l'interface série. Il s'agit de la valeur mesurée sur la broche analogique 0 dans notre bout de sketch. L'interrogation de la broche analogique et la transmission à l'interface ont lieu continuellement dans la fonction `loop`.

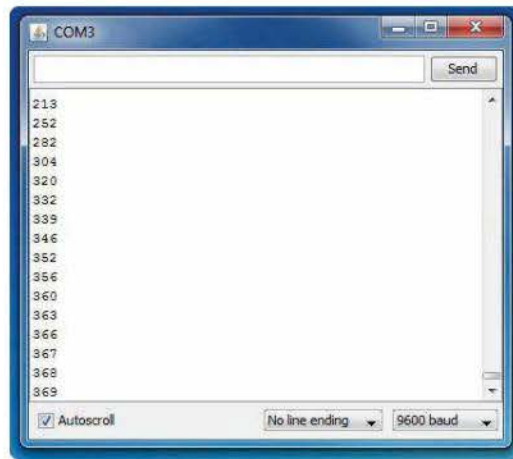


Attention !

Pour que la communication entre émetteur et récepteur puisse se faire, le réglage du taux de transfert doit être le même sur les deux stations.

Vous pouvez suivre la transmission des valeurs déterminées en temps réel, de préférence en ouvrant le Serial Monitor de l'environnement de développement.

Figure 10-8 ▶
Édition des données dans le Serial
Monitor



Cela fonctionne évidemment aussi avec n'importe quel autre programme de terminal ayant accès à l'interface série. Pensez ici aussi à régler le taux de transfert tout en bas à droite de la fenêtre.

Processing le récepteur

Venons en maintenant au programme en question, chargé principalement de représenter graphiquement les valeurs reçues. Le code est assez conséquent, mais voici quand-même une courte description pour que vous ne soyez pas perdu.

```

import processing.serial.*;

Serial mySerialPort;
int xPos = 1;
int serialValue;
int[] yPos;

void setup(){
    size(400, 300);
    println(Serial.list());
    mySerialPort = new Serial(this, Serial.list()[0], 9600);
    mySerialPort.bufferUntil ('\n');
    //Set initial background
    background(0);
    yPos = new int[width];
}

void draw(){
    background(0);
    stroke(255, 255, 0, 120);
    for(int i=0; i < width; i+=50)
        line(i, 0, i, height);
    for(int i=0; i < height; i+=50)
        line(0, i, width, i);

    stroke(255, 0, 0);
    strokeWeight(1);
    int yPosPrev = 0, xPosPrev = 0;
    println(serialValue);
    //Décaler les valeurs de l'array vers la gauche
    for(int x = 1; x < width; x++)
        yPos[x-1] = yPos[x];
    //Joindre les nouvelles coordonnées de la souris
    //à l'extrémité droite de l'array
    yPos[width - 1] = serialValue;
    //Affichage de l'array
    for(int x = 0; x < width; x++){
        if(x > 0)
            line(xPosPrev, yPosPrev, x, yPos[x]);
        xPosPrev = x;           //Stockage de la dernière position x
        yPosPrev = yPos[x];    //Stockage de la dernière position y
    }
}

void serialEvent(Serial mySerialPort){

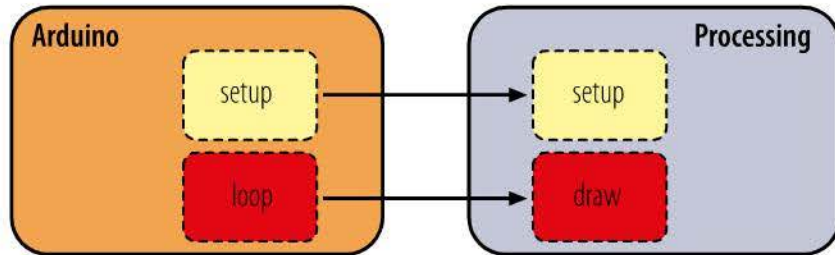
```



```
String portStream = mySerialPort.readString();
float data = float(portStream);
serialValue = height - (int)map(data, 0, 1023, 0, height);
}
```

Processing comprend également deux fonctions principales, qui ressemblent à celles d'Arduino.

Figure 10-9 ►
Correspondance entre
deux fonctions principales



La fonction `setup` est appelée une seule fois elle aussi en début de sketch et sert à initialiser des variables. La fonction `draw` est une boucle sans fin semblable à la fonction `loop` dans Arduino, qui doit son nom au fait qu'elle sert à dessiner (en anglais : *to draw*) les éléments graphiques dans la fenêtre d'édition. Pour pouvoir traiter l'interface série dans Processing, vous devez écrire la ligne :

```
import processing.serial.*;
```

qui permet d'importer un paquet en langage Java. Eh oui, Processing est un langage basé sur Java, contrairement à Arduino qui, lui, utilise C ou C++.

Les deux langages ont une syntaxe très similaire, aussi la programmation dans Processing ne semble-t-elle pas compliquée à ceux qui connaissent bien C ou C++. Si vous écrivez la ligne :

```
println(Serial.list());
```

Processing vous donne une liste de toutes les interfaces série disponibles. L'affichage dans la fenêtre de messagerie ressemble alors à ceci.

```
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
[0] "COM3"
[1] "COM4"
```

Elle indique que deux ports sériels sont disponibles. Arduino utilisant le premier port COM3 correspondant à la première entrée de la liste [0], cet index est reporté dans la ligne ci-après :

```
mySerialPort = new Serial(this, Serial.list()[0], 9600);
```



Attention !

Si, sur votre ordinateur, le port série utilisé n'est pas le premier de la liste, il faut alors remplacer dans la ligne précédente `Serial.list()[0]` par `"COMn"`, où `n` est le numéro du port série effectivement utilisé par Arduino.

Un nouvel objet sériel est ainsi généré, et instancié avec le mot-clé `new` dans Processing. La valeur 9600 apparaît une fois de plus, elle doit correspondre à celle qui est dans le sketch Arduino. Tous les éléments graphiques tels que trame de fond et courbe sont alors dessinés dans la fonction `draw`. Les valeurs Arduino transmises s'accumulent dans la fonction `serialEvent` et sont stockées dans la variable `serialValue`. Cette variable sert à dessiner la courbe dans la fonction `draw`.



Attention !

Si vous avez ouvert un programme de terminal, par exemple Serial Monitor, pour visualiser les valeurs envoyées par Arduino, vous aurez des problèmes si vous démarrez en même temps l'affichage graphique des valeurs dans Processing. Le port COM concerné est en effet exclusivement utilisé par le programme de terminal, interdisant tout accès supplémentaire par un autre programme. Fermez par conséquent le programme de terminal avant de démarrer Processing pour évaluer les données.

Problèmes courants

Si les différentes LED ne réagissent pas aux modifications des conditions lumineuses ou si aucun changement du tracé de la courbe n'est à observer par la suite dans la fenêtre d'édition, il peut y avoir plusieurs raisons.

- Vérifiez que vos fiches de raccordement sur la plaque d'essais correspondent bien au circuit.
- Vérifier qu'il n'y a pas de court-circuit entre elles.
- Les résistances ont-elles bien les bonnes valeurs ?
- Toutes les LED sont-elles correctement polarisées ?
- Vérifiez encore une fois l'exactitude du code du sketch côté Arduino et côté Processing.
- Ouvrez le Serial Monitor dans l'IDE Arduino pour vous assurer que des valeurs différentes sont transmises à l'interface série, lorsque les conditions lumineuses varient. Dans le code de Processing, vous pouvez ajouter la ligne `println(serialValue)` de manière à ce que les valeurs transmises (pour peu qu'elles le soient) s'affichent également dans la fenêtre de messagerie.
- Vérifiez que l'interface série utilisée n'est pas bloquée par un autre processus et que seul Processing y accède.

Qu'avez-vous appris ?

- Vous savez comment interroger une entrée analogique à laquelle est reliée une résistance photosensible (LDR) avec l'instruction `analogRead`.
- Un diviseur de tension sert à diviser la tension appliquée à ses bornes dans un rapport déterminé. Nous avons utilisé cette propriété pour amener à l'entrée analogique une tension fonction de l'intensité lumineuse.
- Vous savez comment échanger des données entre deux programmes au moyen de l'interface série. L'émetteur est ici la carte Arduino et le récepteur un sketch Processing reproduisant visuellement les données reçues sous forme de courbe.

Exercice complémentaire

Créez un sketch Arduino faisant clignoter régulièrement toutes les LED concernées quand par exemple un certain seuil de flux lumineux est franchi, pour vous avertir qu'un état critique est maintenant atteint et qu'une crème solaire avec indice de protection 75+ doit être utilisée.

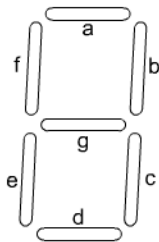
L'afficheur sept segments

Au sommaire :

- la commande d'un afficheur à sept segments ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Qu'est-ce qu'un afficheur sept segments ?

Pour visualiser des états logiques (vrai ou faux) ou des données (14, 25, "Hello User") sous une forme quelconque, il nous faut commander les LED dans un premier temps et revenir au Serial Monitor dans un deuxième temps. Il existe en électronique d'autres éléments d'affichage que les LED, l'afficheur sept segments étant l'un deux. Comme son nom l'indique, cet afficheur se compose de sept segments qui, disposés d'une certaine manière, peuvent représenter des chiffres et, dans une moindre mesure, des signes. La figure 11-1 présente un tel afficheur de manière schématisée.









◀ **Figure 11-1**
Afficheur sept segments





On voit que chaque segment est pourvu d’une petite lettre. L’ordre n’est pas primordial mais la forme montrée ici s’est imposée et a été adoptée pratiquement partout. Aussi l’utiliserons-nous également toujours sous cette forme. Si maintenant nous commandons les différents segments avec habileté, nous pouvons afficher des chiffres allant de 0 à 9. On peut aussi afficher des lettres, nous y reviendrons plus tard. Votre quotidien est sûrement rempli de ces afficheurs sept segments sans que vous n’y ayez jamais prêté attention. Faites un tour en ville et vous verrez à quel point ils sont courants. Voici d’ailleurs une petite liste des possibilités d’utilisation :

- l’affichage des prix sur les stations-service (toujours en hausse hélas !) ;
- l’affichage de l’heure sur certains bâtiments ;
- l’affichage de la température ;
- les montres numériques ;
- les tensiomètres médicaux ;
- les thermomètres numériques.

Le tableau 11-1 indique une fois pour toutes, en vue de la programmation, quels sont les segments à allumer pour chacun des chiffres.

Tableau 11-1 ▶
Commande des sept segments

Afficheur	a	b	c	d	e	f	g
	1	1	1	1	1	1	0
	0	1	1	0	0	0	0
	1	1	0	1	1	0	1
	1	1	1	1	0	0	1
	0	1	1	0	0	1	1
	1	0	1	1	0	1	1

Afficheur	a	b	c	d	e	f	g
	1	0	1	1	1	1	1
	1	1	1	0	0	0	0
	1	1	1	1	1	1	1
	1	1	1	1	0	1	1

◀ **Tableau 11-1 (suite)**
Commande des sept segments

Le chiffre 1 dans ce tableau ne signifie pas forcément niveau HIGH, mais c'est la commande de l'allumage du segment concerné. Celle-ci peut se faire soit avec le niveau HIGH que nous connaissons (+5 V résistance série incluse), soit avec un niveau LOW (0 V). Vous voulez peut-être savoir maintenant en fonction de quoi on choisit une commande. Cela dépend en fait du type de l'afficheur sept segments. Deux approches sont possibles :

- la cathode commune ;
- l'anode commune.

En cas de cathode commune, toutes les cathodes des diverses LED de l'afficheur sept segments sont réunies en interne et reliées à la masse à l'extérieur. Les différents segments sont commandés par des résistances série dûment raccordées au niveau HIGH. Notre exemple porte cependant sur un afficheur sept segments avec anode commune. Ici, c'est exactement le contraire : toutes les anodes des diverses LED sont reliées entre elles en interne et raccordées au niveau LOW à l'extérieur. Les segments sont commandés par des résistances série correctement dimensionnées, en passant par les différentes cathodes des LED qui sont accessibles à l'extérieur.

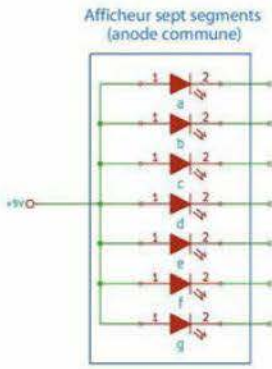
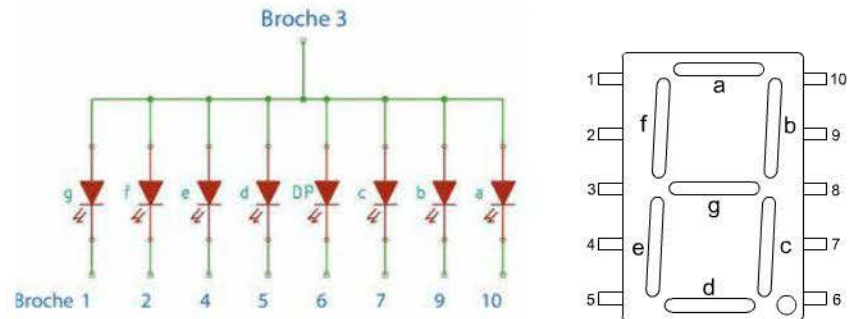


Figure 11-2 ►
Commande de l'afficheur
sept segments de type SA 39-11 GE

Dans le circuit pour afficheur sept segments avec anode commune de gauche, toutes les anodes des diverses LED en service sont reliées à la tension d'alimentation +5 V. Les cathodes sont reliées par la suite aux sorties numériques de votre carte Arduino et pourvues des différents niveaux de tension conformes au tableau de commande. Nous utilisons pour notre essai un afficheur sept segments avec anode commune de type SA 39-11 GE. La figure suivante illustre le brochage de cet afficheur.



Le graphique de gauche montre les broches utilisées de l'afficheur sept segments, et le graphique de droite le brochage du type utilisé. DP est la forme abrégée de point décimal.

Composants nécessaires



1 afficheur sept segments (par exemple de type SA 39-11 GE avec anode commune)



7 résistances de 330 Ω



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

```
int segments[10][7] = {{1, 1, 1, 1, 1, 1, 0}, //0
                        {0, 1, 1, 0, 0, 0, 0}, //1
                        {1, 1, 0, 1, 1, 0, 1}, //2
                        {1, 1, 1, 1, 0, 0, 1}, //3
```

```

        {0, 1, 1, 0, 0, 1, 1}, //4
        {1, 0, 1, 1, 0, 1, 1}, //5
        {1, 0, 1, 1, 1, 1, 1}, //6
        {1, 1, 1, 0, 0, 0, 0}, //7
        {1, 1, 1, 1, 1, 1, 1}, //8
        {1, 1, 1, 1, 0, 1, 1}} ; //9
int pinArray[] = {2, 3, 4, 5, 6, 7, 8} ;

void setup(){
    for(int i = 0; i < 7; i++){
        pinMode(pinArray[i], OUTPUT);
    }
void loop(){
    for(int i = 0; i < 10; i++){
        for(int j = 0; j < 7; j++){
            digitalWrite(pinArray[j], (segments[i][j]==1)?LOW:HIGH);
            delay(1000); //Pause de 1 seconde
        }
    }
}

```

Revue de code

Du point de vue logiciel, les variables suivantes sont nécessaires à notre programmation expérimentale.

Variable	Objet
segments	Tableau bidimensionnel pour stocker l'information des segments pour chaque chiffre
pinArray	Tableau unidimensionnel pour stocker les broches connectées à l'afficheur

◀ **Tableau 11-2**
Variables nécessaires et leur objet

Un tableau bidimensionnel s'impose d'emblée pour stocker les informations sur les segments à allumer pour chaque chiffre de 0 à 9. Ces valeurs sont définies dans la variable globale `segments` en début de sketch :

```

int segments[10][7] = {{...},
    ...
    {...}};

```

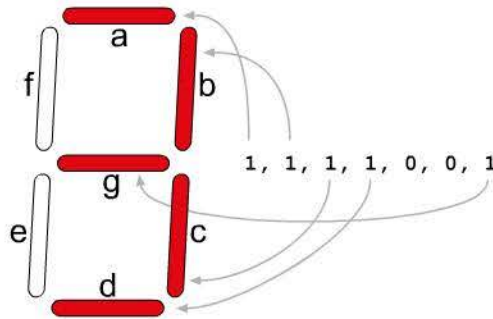
Le tableau comprend 10×7 cases mémoire, le contenu de chacune d'elles pouvant être obtenu par les coordonnées :

```
segments[x][y]
```

La coordonnée x sert pour tous les chiffres de 0 à 9 (soit 10 cases mémoire), et la coordonnée y pour tous les segments de a à g (soit 7 cases mémoire). On détermine par exemple les segments à allumer du chiffre 3 en écrivant la ligne :

segments[3][y]

les résultats pour la variable *y* allant de 0 à 6 étant obtenus par une boucle *for*. Les données des segments sont alors celle de la figure suivante.



Minute, s'il vous plaît ! Vous avez dit que ce type d'afficheur sept segments disposait d'une anode commune. Pourtant, il y a un 1 là où il devrait y avoir une mise à la masse dans le tableau des segments. Ce n'est donc pas le cas alors !

Je confirme la première partie de ce que vous venez de dire. Mais pour la deuxième, vous n'avez sûrement pas été totalement attentif. J'ai dit qu'un 1 ne voulait pas forcément dire niveau HIGH, mais simplement que le segment en question devait être allumé. Dans le cas d'un afficheur sept segments à cathode commune, on commande l'allumage du segment souhaité avec le niveau HIGH, tandis que dans le cas d'un afficheur sept segments à anode commune, on le commande avec le niveau LOW. On écrit ainsi la ligne suivante :

```
digitalWrite(pinArray[j], (segments[i][j]==1) ?LOW:HIGH);
```

Si l'information est un 1, LOW est alors transmis comme argument à la fonction *digitalWrite*. Sinon, c'est HIGH. Le segment correspondant s'allume si c'est LOW, et se voit géré de manière à rester éteint si c'est HIGH. Notre sketch affiche tous les chiffres de 0 à 9 au rythme d'une seconde. Le code suivant est utilisé pour ce faire :

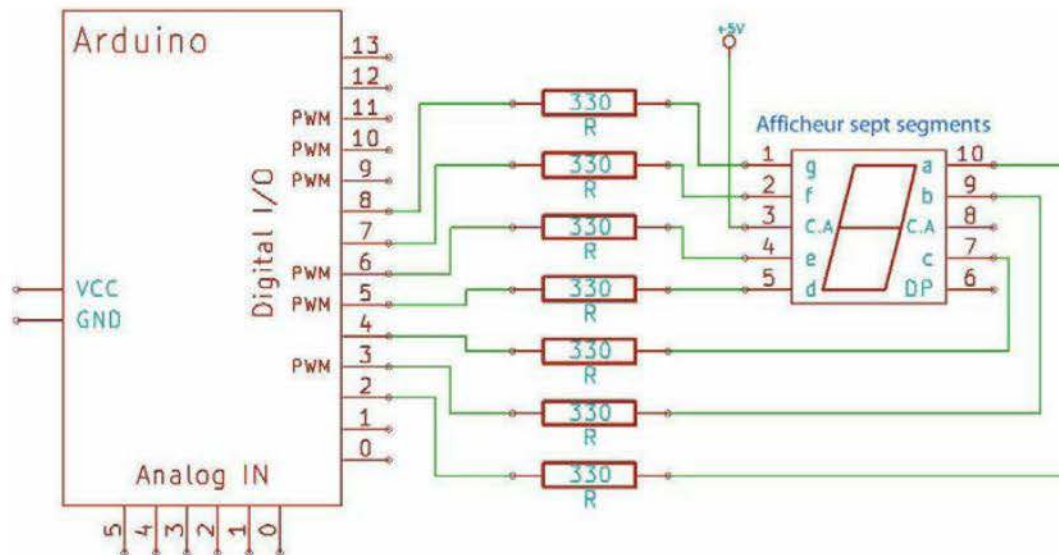
```
for(int i = 0; i < 10; i++){
  for(int j = 0; j < 7; j++){
    digitalWrite(pinArray[j], (segments[i][j]==1)?LOW:HIGH);
    delay(1000); //Pause de 1 seconde
  }
}
```

La boucle extérieure avec la variable de contrôle i sélectionne dans le tableau le chiffre à afficher tandis que la boucle intérieure avec la variable j sélectionne les segments à allumer.

Schéma

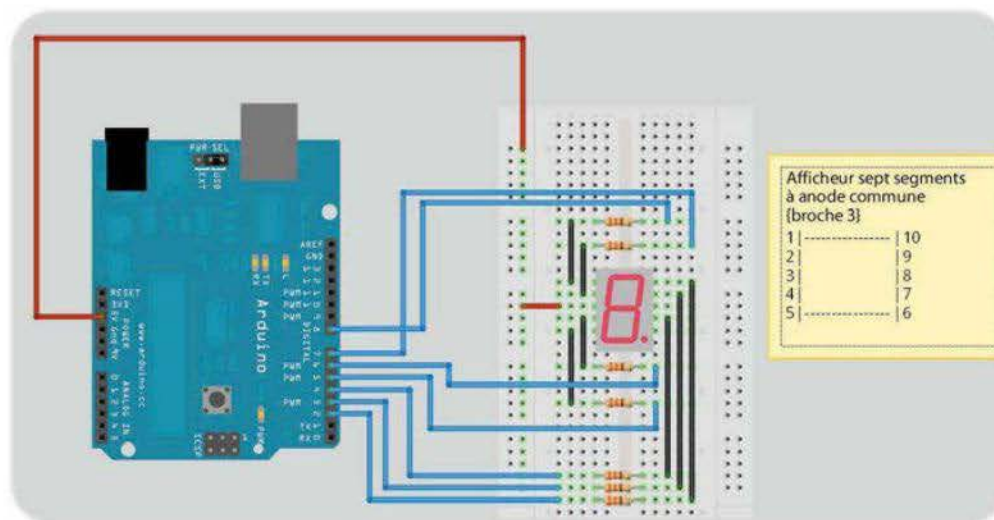
Le circuit ressemble à celui du séquenceur de lumière. Mais pas si vite, il va se compliquer.

▼ **Figure 11-3**
Commande de l'afficheur sept segments



Réalisation du circuit

▼ **Figure 11-4**
Réalisation du circuit de l'afficheur sept segments avec Fritzing



Sketch amélioré

Les divers segments d'un chiffre étaient commandés jusqu'ici au moyen d'un tableau bidimensionnel, la première dimension servant à sélectionner le chiffre désiré, et la deuxième les différents segments. Le sketch suivant va nous permettre de tout faire avec un tableau unidimensionnel. Comment ? C'est simple puisque bits et octets n'ont déjà plus de secret pour vous. L'information de segment doit maintenant tenir dans une seule valeur. Quel type de donnée s'impose ici ? Nous avons affaire à un afficheur sept segments, et à un point décimal que nous laisserons de côté pour l'instant. Cela fait donc 7 bits, qui tiennent idéalement dans un seul octet de 8 bits. Chaque bit est simplement affecté à un segment et tous les segments nécessaires peuvent être commandés avec un seul octet. J'en profite pour vous montrer comment initialiser directement une variable par le biais d'une combinaison de bits :

```
void setup(){  
  Serial.begin(9600);  
  byte a = B10001011; //Déclarer + initialiser la variable  
  Serial.println(a, BIN) //Imprimer en tant que valeur binaire  
  Serial.println(a, HEX); //Imprimer en tant que valeur hexadécimale  
  Serial.println(a, DEC); //Imprimer en tant que valeur décimale  
}  
  
void loop(){/*Vide*/}
```

La ligne décisive est bien sûr la suivante :

```
byte a = B10001011;
```

Ce qui est remarquable pour ne pas dire génial là-dedans, c'est le fait que le préfixe **B** permet de représenter une combinaison de bits qui sera affectée à la variable située à gauche du signe **=**. Cela simplifie les choses quand par exemple vous connaissez une combinaison de bits et souhaitez la sauvegarder. Il vous faudrait sinon convertir la valeur binaire en valeur décimale avant de sauvegarder. Cette étape intermédiaire n'est ici plus nécessaire.



Je ne comprends pas bien. Le type de donnée **byte** est bien – du moins, il me semble – un nombre entier. Type de donnée et nombres entiers sont bien composés de chiffres allant de 0 à 9. Pourquoi maintenant peut-on commencer par la lettre **B** et la faire suivre d'une combinaison de bits ? Ou s'agit-il d'une chaîne de caractères ?


Le type de donnée byte est un type de nombre entier. Vous avez raison sur ce point. Là où vous avez tort, c'est sur le fait qu'il pourrait s'agir d'une chaîne de caractères. Celle-ci serait alors entre guillemets. Il s'agit en fait de tout autre chose. Aucune idée ? Je ne dirai qu'un mot : `#define`. Ça vous dit quelque chose ? Voyez plutôt. Il existe dans les tréfonds d'Arduino un fichier nommé `binary.h` qui se trouve dans le répertoire : `arduino-1.x.y\hardware\arduino\cores\arduino`.

Voici un court extrait de ce fichier, dont les nombreuses lignes n'ont pas toutes besoin d'être montrées.

```
1 #ifndef Binary_h
2 #define Binary_h
3
4 #define B0 0
5 #define B00 0
6 #define B000 0
7 #define B0000 0
8 #define B00000 0
9 #define B000000 0
10 #define B0000000 0
11 #define B00000000 0
12 #define B1 1
13 #define B01 1
14 #define B001 1
15 #define B0001 1
16 #define B00001 1
17 #define B000001 1
18 #define B0000001 1
19 #define B00000001 1
```

Ce fichier contient toutes les combinaisons de bits possibles pour les valeurs de 0 à 255, qui y sont définies en tant que constantes symboliques. Je me suis permis de retirer la ligne pour la valeur 139 (déconseillé, à moins de restaurer ensuite l'état initial !) pour voir comment le compilateur réagit. Voyez plutôt :

```
void setup(){
  Serial.begin(9600);
  byte a = B10001011; //Déclarer + initialiser la variable
  Serial.println(a, BIN); //Imprimer en tant que valeur binaire
  Serial.println(a, HEX); //Imprimer en tant que valeur hexadécimale
  Serial.println(a, DEC); //Imprimer en tant que valeur décimale
}
void loop ( ){/*Vide*/}
```



The screenshot shows an IDE window with an orange error banner at the top that reads: `'B10001011' was not declared in this scope`. Below the banner, the error details are displayed in a black box with white text: `DiceGroupwithLibrary.cpp: In function 'void setup()': DiceGroupwithLibrary.pde:-1: error: 'B10001011' was not declared in this scope`.

Le message d'erreur indique que le nom B10001011 n'a pas été trouvé. Il me faut encore vous expliquer les lignes suivantes avant d'en revenir au projet :

```
Serial.println(a, BIN); //Imprimer en tant que valeur binaire
Serial.println(a, HEX); //Imprimer en tant que valeur hexadécimale
Serial.println(a, DEC); //Imprimer en tant que valeur décimale
```

La fonction `println` peut accueillir, en plus de la valeur à imprimer, un autre argument qui peut être indiqué séparé par une virgule. Je vous ai mis ici les trois plus importants. Vous en trouverez d'autres sur la page de référence des instructions Arduino sur Internet. Des explications parlantes figurent sous forme de commentaires derrière les lignes d'instructions. L'impression dans le Serial Monitor est alors la suivante :

```
10001011
8B
139
```

Passons maintenant à la commande de l'afficheur sept segments au moyen du tableau bidimensionnel. Voici auparavant le sketch complet que nous allons analyser :

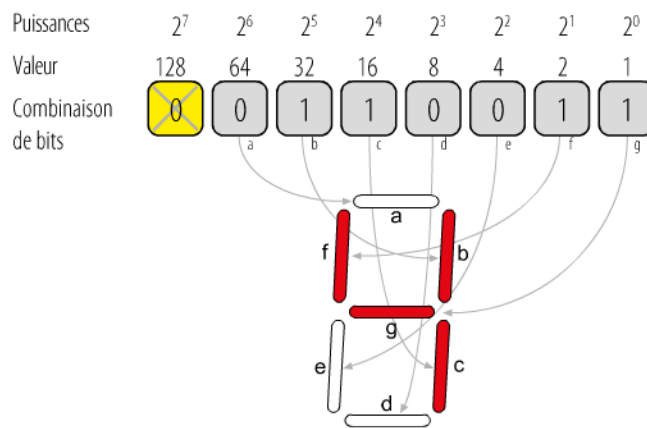
```
byte segments[10] = {B0111110, //0
                    B00110000, //1
                    B01101101, //2
                    B01111001, //3
                    B00110011, //4
                    B01011011, //5
                    B01011111, //6
                    B01110000, //7
                    B01111111, //8
                    B01111011}; //9

int pinArray[] = {2, 3, 4, 5, 6, 7, 8};

void setup(){
  for(int i = 0; i < 7; i++)
    pinMode(pinArray[i], OUTPUT);
}

void loop(){
  for(int i = 0; i < 10; i++){ //Commande du chiffre
    for(int j = 6; j >= 0; j--){ //Interrogation des bits pour
                                //les segments
      digitalWrite(pinArray[6-j], bitRead(segments[i], j) == 1 ? LOW : HIGH);
    }
    delay (500); //Attendre une demi-seconde
  }
}
```

Dans la figure 11-5, on voit très bien quel bit est en charge de quel segment au sein de l'octet.



◀ **Figure 11-5**

Un octet gère les segments de l'afficheur (ici par exemple pour le chiffre 4).

Ayant seulement sept segments à commander et ne tenant pas compte du point décimal, j'ai constamment donné au MSB (rappelez-vous : MSB = bit le plus significatif) la valeur 0 pour tous les éléments du tableau.

Tout se joue bien entendu encore – et comment en serait-il autrement – à l'intérieur de la fonction `loop`. Jetons-y un coup d'œil :

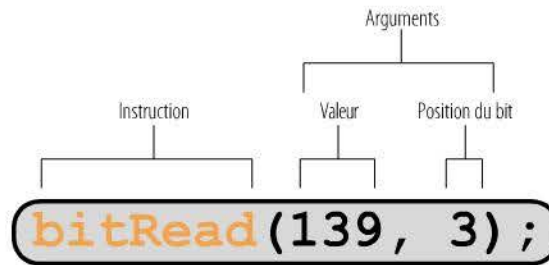
```
void loop(){
  for(int i = 0; i < 10; i++){ //Commande du chiffre
    for(int j = 6; j >= 0; j--){ //Interrogation des bits pour
                                //les segments
      digitalWrite(pinArray[6-j], bitRead(segments[i], j)==1?LOW:HIGH);
    }
    delay(500); //Attendre une demi-seconde
  }
}
```

La boucle extérieure `for` avec la variable de contrôle `i` commande encore les divers chiffres de 0 à 9. C'était déjà le cas dans la première solution. Le code est ensuite différent. La boucle intérieure `for` avec la variable de contrôle `j` est chargée de choisir le bit dans le chiffre sélectionné. Je commence du côté gauche par la position 6, qui est en charge du segment `a`. Le tableau des broches gérant cependant la broche 8 pour le segment `g` à la position 6 de l'index, la commande doit se faire en sens inverse. On y parvient en soustrayant le nombre 6 puisque j'ai gardé tel quel le tableau des broches du premier exemple :

```
pinArray[6 - j]
```

Voici maintenant à une fonction intéressante, permettant de lire un bit déterminé dans un octet. Elle porte le nom `bitRead`.

Figure 11-6 ▶
Instruction `bitRead`



Cet exemple donne le bit de la position 3 pour la valeur décimale 139 (binaire : 10001011). Le comptage commence pour l'index 0 au LSB (bit le moins significatif) du côté droit. La valeur renvoyée serait par conséquent un 1. La ligne :

```
digitalWrite(pinArray[6-j],bitRead(segments[i],j) == 1?LOW:HIGH);
```

permet de vérifier que la lecture du bit sélectionné renvoie bien un 1. Si c'est le cas, la broche sélectionnée est commandée avec le niveau LOW, autrement dit le segment s'allume. N'oubliez pas : anode commune ! Sauriez-vous expliquer la différence entre les deux solutions ?



Laissez-moi réfléchir. Bon ! Dans la première version avec le tableau bidimensionnel, le chiffre à afficher est sélectionné par la première dimension tandis que les segments à commander le sont par la deuxième. Cette information se trouve dans les différents éléments du tableau. Dans la deuxième version, le chiffre à afficher est également sélectionné par la première dimension. S'agissant d'un tableau unidimensionnel, elle est cependant la seule dimension. Seulement, l'information pour commander les segments est contenue dans les diverses valeurs de l'octet. Ce qui était fait auparavant par la deuxième dimension est maintenant fait par les bits d'un octet.

Très bien, Arduus ! La technique est comparable.

Problèmes courants

Si l’affichage ne correspond pas aux chiffres 1 à 9 ou si des combinaisons incohérentes s’affichent, vérifiez les choses suivantes.

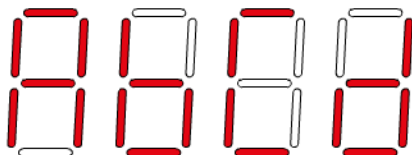
- Vos fiches de raccordement sur la maquette correspondent-elles bien au circuit ?
- Vérifiez qu’il n’y a pas de court-circuit entre elles.
- Le code du sketch est-il correct ?
- Si des caractères incohérents s’affichent, il se peut que vous ayez interverti des lignes de commande. Vérifiez le câblage avec le schéma ou la fiche technique de l’afficheur sept segments.
- Le tableau des segments est-il initialisé avec les bonnes valeurs ?

Qu’avez-vous appris ?

- Dans ce montage, les principes de la commande d’un afficheur sept segments vous sont expliqués.
- L’initialisation d’un tableau vous permet de définir les différents segments de l’affichage pour pouvoir les commander à votre aise par la suite.
- Le fichier d’en-tête `binary.h` contient un grand nombre de constantes symboliques que vous pouvez utiliser dans votre sketch.
- Vous savez comment convertir un nombre à imprimer dans une autre base numérique en ajoutant un deuxième argument (BIN, HEX ou DEC) à la méthode `println`.
- La fonction `bitRead` vous permet de lire l’état de certains bits d’un nombre.

Exercice complémentaire

Élargissez la programmation du sketch de telle sorte que certaines lettres puissent s’afficher à côté des chiffres 0 à 9. Ce n’est certes pas possible pour tout l’alphabet, donc à vous de trouver lesquelles pourraient convenir. La figure suivante vous fournit quelques exemples pour commencer.





Pour aller plus loin

Il existe un nombre infini de déclinaisons d'afficheurs sept segments. L'affichage peut être de différentes couleurs, telles que :

- jaune ;
- rouge ;
- vert ;
- rouge très clair.

Il faut bien entendu s'assurer du type de connexion avant d'acheter :

- l'anode commune ;
- la cathode commune.

Ils ont des tailles différentes. En voici deux proposées par le fournisseur Kingbright :

- type SA-39 : hauteur des chiffres = $0,39'' = 9,9 \text{ mm}$;
- type SA-56 : hauteur des chiffres = $0,56'' = 14,2 \text{ mm}$.

Le clavier numérique

Au sommaire :

- la fabrication d'un clavier numérique ;
- apprendre à interroger les différentes touches élégamment ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Qu'est-ce qu'un clavier numérique ?

Vous connaissez déjà le bouton-poussoir, dont nous avons parlé au cours de certains montages. Mais ici, plusieurs boutons-poussoirs (touches) sont réunis en une matrice – donc disposés en lignes et colonnes – de manière à proposer les chiffres 0 à 9 et deux touches spéciales telles que * et #. À quoi cela sert-il ? Eh bien, vous utilisez ce jeu de touches tous les jours, entre autres pour téléphoner.



◀ **Figure 12-1**
Clavier de téléphone

Il s'agit d'une matrice de 4×3 touches (4 lignes et 3 colonnes). Cette matrice est également appelée *keypad* (clavier numérique) et peut être achetée prête à l'emploi en différentes variantes. La figure 12-2 montre deux claviers numériques à film. Celui de gauche possède même quelques touches supplémentaires A à D, qui peuvent s'avérer très utiles si les 12 touches du clavier numérique de droite ne suffisent pas pour votre projet.

Figure 12-2 ►
Clavier numérique à film 4×4
à 16 touches et clavier numérique
à film 4×3 à 12 touches.



Je vois mal comment brancher par exemple le clavier numérique à film 4×4 sur mon Arduino sans rencontrer des problèmes de broches. On pourrait bien sûr raccorder les 16 touches d'un côté aux +5 V et les 16 prises correspondantes aux entrées numériques. On pourrait également se servir des entrées analogiques en cas de besoin. C'est ce que nous avons fait déjà.

Vous pourriez bien sûr procéder ainsi et cela fonctionnerait s'il n'y avait pas les limites physiques de la carte Arduino Uno. Une solution consisterait à utiliser la carte Arduino Mega, dont le nombre d'interfaces est bien élevé. Mais soyons ingénieux ; il existe une bibliothèque pour claviers numériques prête à l'emploi sur le site Internet Arduino, aussi allons-nous tout faire par nous-mêmes. Nous utiliserons le clavier numérique 4×3 que nous aurons fabriqué de nos propres mains. Voici la liste du matériel nécessaire.

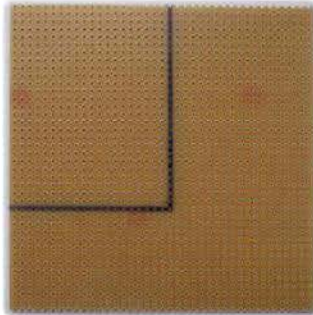
Composants nécessaires



12 boutons-poussoir



1 jeu de connecteurs femelles empilables



1 carte de dimensions 10×10 ou mieux 16×10 (vous pourrez alors en faire deux shields). La découpe du shield est déjà indiquée, et je reviendrai bientôt sur cette dernière.



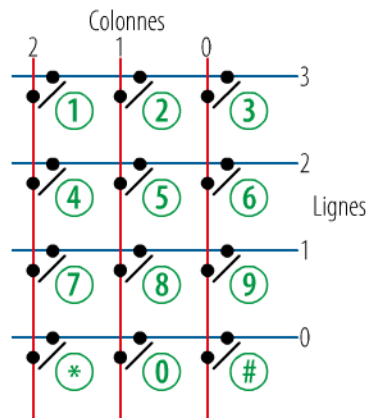
Fil, si possible de différentes couleurs

Réflexions préliminaires

Ardus vient de nous faire remarquer que notre clavier numérique 4×4 nécessitait 16 lignes pour interroger toutes les touches. Un clavier numérique 4×3 n'aurait quant à lui besoin que de 12 lignes. Mais ce serait encore beaucoup trop à mon avis. Une solution astucieuse existe, dont l'idée de base a déjà servi pour commander les deux afficheurs sept segments. Vous vous demandez sûrement ce que des afficheurs sept segments ont à voir avec ces touches. Le mot commun est multiplexage. Il signifie que certains signaux sont regroupés et envoyés par un moyen de transmission pour minimiser l'utilisation des lignes et en tirer profit le plus possible. Sur les afficheurs sept segments, les lignes de commande de deux segments sont montées en parallèle et utilisées pour commander les deux. Sept ou huit lignes par segment sont ainsi économisées. La solution trouvée

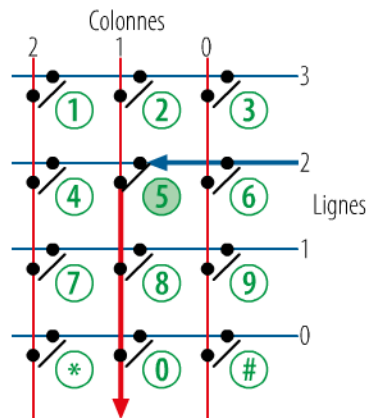
pour interroger les différentes touches d'un clavier numérique est relativement simple. Voici le câblage des 12 touches.

Figure 12-3 ►
Câblage des 12 touches
d'un clavier numérique 4 × 3



Imaginez une grille composée de 4 + 3 fils métalliques posés l'un sur l'autre sans pour autant se toucher. Voilà à quoi ressemble ce graphique. On voit que les 4 fils horizontaux en bleu forment des lignes numérotées de 0 à 3. Au-dessus, les trois fils verticaux en rouge forment à peu de distance des colonnes numérotées de 0 à 2. Chaque intersection présente des petits contacts reliant, quand on appuie sur la touche, la ligne et la colonne en question pour former un tronçon de circuit électrique. Regardez bien la figure 12-4, où la touche 5 est enfoncée.

Figure 12-4 ►
La touche 5 est enfoncée (les lignes
en gras montrent le passage
du courant).



Le courant peut alors passer de la ligne 2 via l'intersection 5 dans la colonne 1 et y être détecté.

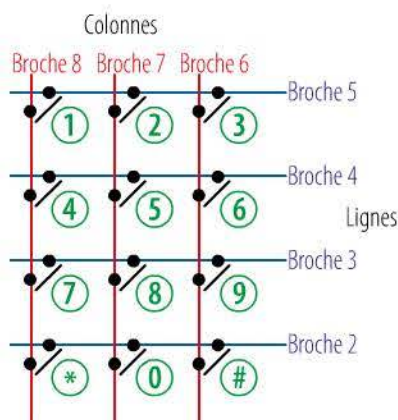
Mais si une tension est appliquée simultanément à toutes les lignes, la touche 2 au-dessus de la touche 5 peut tout aussi bien être appuyée et j'enregistrerai une impulsion correspondante sur la colonne 1. Comment faire la différence ?



Je vois, Ardu, que vous n'avez pas complètement compris le principe. Pas de quoi fouetter un chat cependant. Disons grossièrement que nous envoyons tour à tour un signal par les lignes 0 à 3 et interrogeons ensuite également tour à tour le niveau sur les colonnes 0 à 2. Le déroulement est alors le suivant :

- **Niveau HIGH sur le fil de la rangée 0**
 - Interrogation du niveau sur la colonne 0
 - Interrogation du niveau sur la colonne 1
 - Interrogation du niveau sur la colonne 2
- **Niveau HIGH sur le fil de la rangée 1**
 - Interrogation du niveau sur la colonne 0
 - Interrogation du niveau sur la colonne 1
 - Interrogation du niveau sur la colonne 2
 - etc.

Cette interrogation est bien sûr si rapide que suffisamment de passages ont lieu en une seule seconde pour que pas un appui de touche ne soit omis. Le shield a été câblé à demeure avec les numéros de broches des entrées et sorties indiqués sur la figure 12-5.



◀ **Figure 12-5**

Câblage des différentes lignes et colonnes avec les broches numériques

Pimentons ici un peu les choses et créons notre propre bibliothèque qui servira plus tard à d'autres montages. Elle offre une certaine fonctionnalité de base et pourra bien entendu être modifiée ou élargie si

besoin est. Le sketch principal demande continuellement au shield quelle touche a été appuyée. Le résultat est affiché dans le Serial Monitor pour visualisation. Pour vérifier le bon fonctionnement de la bibliothèque, fixons-nous les spécifications suivantes :

- si vous n'appuyez sur aucune touche, aucun caractère n'est affiché dans le Serial Monitor ;
- si vous n'appuyez que brièvement sur une touche, le chiffre ou le caractère s'affiche dans le moniteur ;
- si vous appuyez sur une touche un long moment, qui peut être préalablement défini en conséquence, le chiffre ou le caractère s'affiche plusieurs fois l'un derrière l'autre jusqu'à ce que la touche soit relâchée.

Code du sketch

Sketch principal avec revue de code

Commençons par le sketch principal qui, du fait de la fonctionnalité délocalisée dans une bibliothèque, semble clair pour ne pas dire spartiate. Mais attendez seulement. Les choses vont devenir plus complexes et plus intéressantes.

```
#include "MyKeypad.h"
int rowArray[] = {2, 3, 4, 5}; //Initialiser le tableau avec les
                                //numéros de broche des lignes
int colArray[] = {6, 7, 8};    //Initialiser le tableau avec les
                                //numéros de broche des colonnes
MyKeypad myOwnKeypad(rowArray, colArray); //Instanciation d'un objet
void setup(){
    Serial.begin(9600);           //Préparer la sortie série
    myOwnKeypad.setDebounceTime(500); //Régler le temps du
                                    //rebond à 500 ms
}
void loop(){
    char myKey = myOwnKeypad.readKey(); //Lecture de la touche appuyée
    if(myKey != KEY_NOT_PRESSED)        //Une touche quelconque
                                        //a-t-elle été appuyée ?
        Serial.println(myKey);          //Impression du caractère
                                        //de la touche
}
```

La première ligne incorpore, tout comme dans la bibliothèque-dé du montage n° 9, le fichier d'en-tête permettant d'utiliser la bibliothèque. Nous verrons bientôt ce qu'il contient. Déclarons pour

commencer deux tableaux, que nous initialisons avec les numéros des broches de connexion aux lignes et aux colonnes du clavier numérique. Cela offre une plus grande flexibilité et permet d'adapter les constructions différentes. La ligne :

```
MyKeyPad myOwnKeyPad(rowArray, colArray);
```

génère l'instance `myOwnKeyPad` de la classe `MyKeyPad` qui est définie dans la bibliothèque, et transmet les deux tableaux au constructeur de la classe. Ces informations lui sont nécessaires pour commencer à évaluer sur laquelle des 12 touches on a appuyé. Le temps de rebond est déterminé par la ligne suivante :

```
myOwnKeyPad.setDebounceTime(500);
```

La méthode `setDebounceTime` avec l'argument 500 est ainsi appelée. L'instance est ensuite continuellement interrogée au sein de la fonction `loop`, la question posée étant : « Indique moi la touche qui est actuellement appuyée sur le clavier numérique ! » Pour y arriver, il faut écrire la ligne suivante :

```
char myKey = myOwnKeyPad.readKey();
```

Elle affecte le résultat de la requête à la variable `myKey` du type `char`. On peut maintenant réagir en conséquence. Il le faut car la méthode renvoie toujours une valeur, qu'une touche ait été appuyée ou non. Mais vous souhaitez sûrement voir à l'écran si une touche a été appuyée. Aussi la valeur `KEY_NOT_PRESSED` est-elle renvoyée quand aucune touche n'est appuyée. La requête `if` n'envoie donc le caractère correspondant à la touche au Serial Monitor que si une touche est véritablement appuyée.

```
if(myKey != KEY_NOT_PRESSED)
    Serial.println(myKey);
```

Une question en passant : qu'y a-t-il derrière `KEY_NOT_PRESSED` ?

Question pertinente car j'en serais venu de toute façon à parler du fichier d'en-tête. De nombreuses constantes symboliques y sont définies. Parmi ces constantes se cache le caractère `-`, qui est toujours envoyé lorsqu'aucune touche n'est appuyée. Je lui ai donné ce nom évocateur pour que le code soit plus lisible.



Fichier d'en-tête avec revue de code

Le fichier d'en-tête sert, comme nous l'avons déjà expliqué, à faire connaître les variables et les méthodes nécessaires à la définition de la classe en question. Voyons maintenant ce qu'on y trouve :

```
#ifndef MYKEYPAD_H
#define MYKEYPAD_H

#if ARDUINO < 100
#include <WProgram.h>
#else
#include <Arduino.h>
#endif

#define KEY_NOT_PRESSED '-' //Nécessaire si aucune touche
                             //n'est appuyée

#define KEY_1 '1'
#define KEY_2 '2'
#define KEY_3 '3'
#define KEY_4 '4'
#define KEY_5 '5'
#define KEY_6 '6'
#define KEY_7 '7'
#define KEY_8 '8'
#define KEY_9 '9'
#define KEY_0 '0'
#define KEY_STAR '*'
#define KEY_HASH '#'

class MyKeyPad{
public:
    MyKeyPad(int rowArray[], int colArray[]); //Constructeur
                                              //paramétré

    void setDebounceTime(unsigned int debounceTime);
                               //Réglage du temps de rebond

    char readKey(); //Détermine la touche appuyée sur
                   //le clavier numérique

private:
    unsigned int debounceTime; //Variable locale pour temps de rebond
    long lastValue; //Dernière valeur de la fonction millis
    int row[4]; //Tableau pour les lignes
    int col[3]; //Tableau pour les colonnes
};
#endif
```

La partie supérieure est consacrée aux constantes symboliques et aux caractères correspondants. Vient ensuite la définition formelle de la

classe sans formulation du code qui, comme chacun sait, se trouve dans le fichier .cpp.

Eh, une minute ! Vous cherchez encore à me faire gober quelque chose que je ne connais pas. Que veut dire `unsigned int` dans la déclaration des variables ?



Eh bien Ardu, vous avez une bien piètre opinion de moi ! Évidemment que j'allais en parler ! Le type de donnée `int` vous est déjà familier. Son domaine s'étend des valeurs négatives aux valeurs positives. Le mot-clé `unsigned` placé devant indique que la variable est déclarée sans signe, autrement dit son domaine de valeurs double puisque les valeurs négatives sont supprimées. Ce type de donnée nécessite également (comme `int`) de deux octets pour que les valeurs positives soient toutes représentées. Le domaine de valeurs va de 0 à 65 535.

Fichier ccp avec revue de code

Voici maintenant un peu de code « fait maison » :

```
#include "MyKeyPad.h"
//Constructeur paramétré
MyKeyPad::MyKeyPad(int rowArray[], int colArray[]){
    //Copier le tableau des broches
    for(int r = 0; r < 4; r++)
        row[r] = rowArray[r];
    for(int c = 0; c < 3; c++)
        col[c] = colArray[c];
    //Programmation des broches numériques
    for(int r = 0; r < 4; r++)
        pinMode(row[r], OUTPUT);
    for(int c = 0; c < 3; c++)
        pinMode(col[c], INPUT);
    //Définition initiale de debounceTime à 300 ms
    debounceTime = 300;
}
//Méthode pour régler le temps de rebond
void MyKeyPad::setDebounceTime(unsigned int time){
    debounceTime = time;
}
//Méthode pour déterminer la touche appuyée
//sur le clavier numérique
char MyKeyPad::readKey(){
    char key = KEY_NOT_PRESSED;
    for(int r = 0; r < 4; r++){
        digitalWrite(row[r], HIGH);
```

```

for(int c = 0; c < 3; c++){
    if((digitalRead(col[c]) == HIGH)&&(millis() - lastValue) >=
        debounceTime){
        if((c==2)&&(r==3)) key = KEY_1;
        if((c==1)&&(r==3)) key = KEY_2;
        if((c==0)&&(r==3)) key = KEY_3;
        if((c==2)&&(r==2)) key = KEY_4;
        if((c==1)&&(r==2)) key = KEY_5;
        if((c==0)&&(r==2)) key = KEY_6;
        if((c==2)&&(r==1)) key = KEY_7;
        if((c==1)&&(r==1)) key = KEY_8;
        if((c==0)&&(r==1)) key = KEY_9;
        if((c==2)&&(r==0)) key = KEY_STAR; // *
        if((c==1)&&(r==0)) key = KEY_o;
        if((c==0)&&(r==0)) key = KEY_HASH; // #
        lastValue = millis();
    }
}
digitalWrite(row[r], LOW); //Restauration du niveau initial
}
return key;
}

```

Voyons d'abord le constructeur. Il sert à initialiser l'objet à créer et à lui donner des valeurs initiales définies. Un constructeur doit permettre d'initialiser autant que possible complètement l'instance, de telle sorte qu'aucun appel de méthode ne soit plus en principe nécessaire pour l'initialisation. Elles ne sont plus utilisées que pour corriger certains paramètres qui, le cas échéant, doivent ou peuvent être modifiés en cours de sketch. Le constructeur n'est appelé qu'une seule fois et de manière implicite lors de l'instanciation, et après cela plus jamais dans la vie de l'objet. Dans notre exemple, les tableaux des lignes et des colonnes lui sont communiqués lors de l'appel, de manière à pouvoir être transmis ensuite aux tableaux locaux au moyen de deux boucles for :

```

//Copie des tableaux de broches
for(int r = 0; r < 4; r++)
    row[r] = rowArray[r];
for(int c = 0; c < 3; c++)
    col[c] = colArray[c];

```

Les broches numériques sont ensuite initialisées et leurs sens de transfert sont définis :

```

//Programmation des broches numériques
for(int r = 0; r < 4; r++)

```

```
pinMode(row[r], OUTPUT);
for(int c = 0; c < 3; c++)
    pinMode(col[c], INPUT);
//Définition initiale de debounceTime à 300 ms
debounceTime = 300;
```

Vous avez dit qu'un objet devait toujours être complètement instancié au moyen d'un constructeur. Mais vous lui transmettez uniquement les tableaux de broches pour les lignes et les colonnes. Un autre paramètre important est néanmoins le temps de rebond. Celui-ci n'est pourtant pas transmis à l'objet par le constructeur. Vous avez pour ce faire une méthode propre. Cela ne contredit-il pas ce que vous venez de dire ?



Oui et non, Arduus ! Il est vrai que le constructeur ne connaît pas le temps de rebond. Mais regardez sa dernière ligne. Le temps y est réglé sur 300 ms. Il s'agit pratiquement d'une initialisation câblée en dur, comme on dit si bien dans les milieux de la programmation. Si la valeur ne vous dit rien, vous pouvez toujours l'adapter à vos besoins, tout comme je l'ai fait d'ailleurs pour la méthode `setDebounceTime`. La valeur de 300 (ms) m'a semblé ici convenir. J'aurais évidemment pu la définir directement, mais je voulais vous montrer cette possibilité. La tâche en question est accomplie par la méthode `readKey`, qui est appelée sans cesse dans la boucle `loop` pour pouvoir réagir immédiatement à un appui sur une touche. Au début de l'appel de la méthode, la ligne suivante fait en sorte que la variable `key` soit immédiatement pourvue d'une valeur initiale :

```
char key = KEY_NOT_PRESSED;
```

Allez voir dans le fichier d'en-tête de quelle valeur il s'agit.

Si aucune touche n'est en effet appuyée, c'est précisément ce signe qui est réexpédié comme résultat. Vient ensuite l'appel des deux boucles `for` imbriquées l'une dans l'autre. La première ligne du clavier numérique est mise au niveau `HIGH` par :

```
digitalWrite(row[r], HIGH);
```

Les niveaux de toutes les colonnes sont ensuite testés.

```
...
for(int c = 0 ; c < 3 ; c++){
    if((digitalRead(col[c]) == HIGH)&&(millis() - lastValue) >=
        debounceTime){
        if((c==2)&&(r==3)) key = KEY_1;
        if((c==1)&&(r==3)) key = KEY_2;
        if((c==0)&&(r==3)) key = KEY_3;
```



```

        if((c==2)&&(r==2)) key = KEY_4;
        if((c==1)&&(r==2)) key = KEY_5;
        if((c==0)&&(r==2)) key = KEY_6;
        if((c==2)&&(r==1)) key = KEY_7;
        if((c==1)&&(r==1)) key = KEY_8;
        if((c==0)&&(r==1)) key = KEY_9;
        if((c==2)&&(r==0)) key = KEY_STAR; /*
        if((c==1)&&(r==0)) key = KEY_o;
        if((c==0)&&(r==0)) key = KEY_HASH; /**
        lastValue = millis();
    }
}
...

```

Si une colonne présente également un niveau HIGH et si en plus le temps de rebond a été pris en compte, alors la première condition `if` est remplie et toutes les conditions `if` subséquentes sont évaluées. Si une condition est vérifiée pour le compteur de lignes `r` et le compteur de colonnes `c`, la variable `key` est initialisée avec la valeur initiale correspondante et renvoyée à l'appelant, en fin de méthode, par l'instruction `return`. Une fois la boucle intérieure terminée, les lignes qui viennent d'être mises au niveau HIGH doivent être remises dans leur état initial, qui est le niveau LOW. Si l'état HIGH était conservé, une interrogation ciblée d'une certaine ligne ne serait alors plus possible. Toutes les lignes auraient un niveau HIGH une fois la boucle extérieure terminée, ce qui mettrait toute la logique d'interrogation sens dessus dessous.



Je ne comprends pas bien ce qui se passe avec le *temps de rebond*. Réexpliquez moi la fonction en question s'il vous plaît. J'ai bien compris qu'elle était nécessaire mais je ne vois pas bien comment tout cela fonctionne.

Mais volontiers, Arduus ! La fonction `millis` renvoie le nombre de millisecondes écoulées depuis le début du sketch. La dernière valeur est pour ainsi dire stockée temporairement dans la variable `lastValue` une fois la boucle intérieure terminée. Si la boucle est de nouveau appelée, la différence entre la valeur actuelle en millisecondes et la valeur précédente est calculée. Ce n'est que si elle est supérieure au temps de rebond défini que la condition est jugée *vérifiée*. Elle se trouve cependant liée avec l'expression qui la précède par un `&` logique.

```

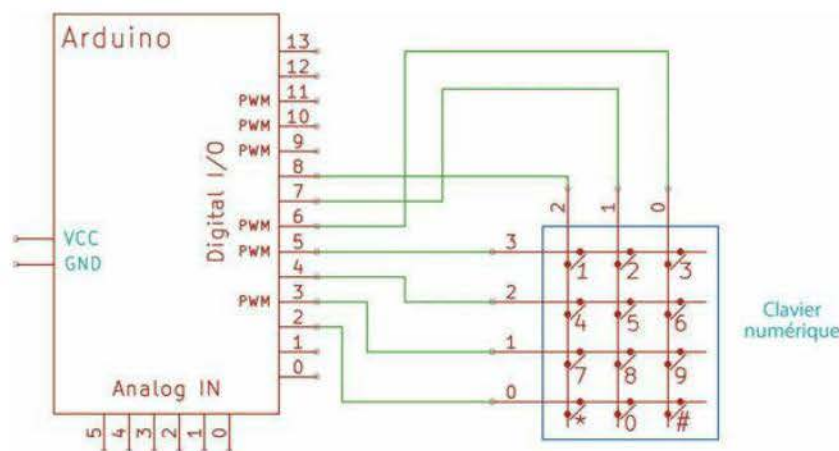
if((digitalRead(col[c]) == HIGH)&&(millis() - lastValue) >=
    debounceTime)...

```

Ce n'est que si les deux conditions délivrent le résultat logique *vrai* à l'instruction `if` que la ligne se poursuit avec l'accolade. Cette structure permet d'obtenir une interruption temporelle, qui se produit aussi toute seule dans certains sketches.

Schéma

Le circuit est assez simple, mais les besoins en programmation sont eux plus importants.



◀ **Figure 12-6**
Commande de notre clavier
numérique

Une chose me frappe d'emblée dans ce schéma. Si aucune touche n'est actionnée, les entrées numériques 6, 7 et 8 se retrouvent pour ainsi dire le bec dans l'eau. N'avez-vous pas dit au début qu'une entrée devait toujours avoir un niveau défini ?

C'est vrai, Arduus ! Mais le clavier numérique doit rester relativement simple et, à moins que la foudre ne tombe sur votre siège et ne provoque une grande perturbation électrostatique de votre environnement, il fonctionne parfaitement bien. Je n'ai pas eu de problème avec ce circuit. Essayez-le seulement. Et tant que vous y êtes, réécrivez donc votre sketch de manière à utiliser les résistances pull-up internes. Le shield n'a pas besoin d'être modifié. Seul le code doit être un peu adapté. Voici un indice pour commencer : si les résistances pull-up sont activées, il vous faut interroger les différentes broches non pas sur leur niveau HIGH mais sur leur niveau LOW. À vous de trouver le reste par vous-même. Considérez cela comme une partie de l'exercice à venir.



Réalisation du shield

Figure 12-7 ►

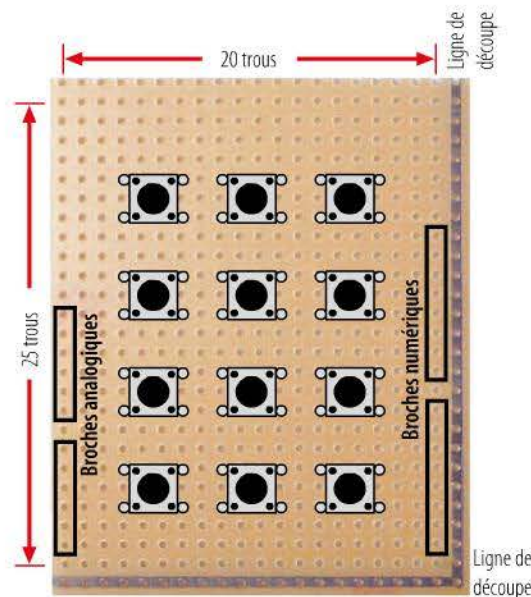
Réalisation du clavier numérique
avec son propre shield



La construction du shield n'est pas mal du tout, n'est-ce pas ? Je vous avais dit au début que je vous montrerai comment faire une carte à la bonne taille. La carte présentée page 399 comporte un marquage correspondant à la taille définitive du shield. Vous trouverez des informations détaillées sur sa fabrication dans le montage n° 22 de réalisation d'un shield.

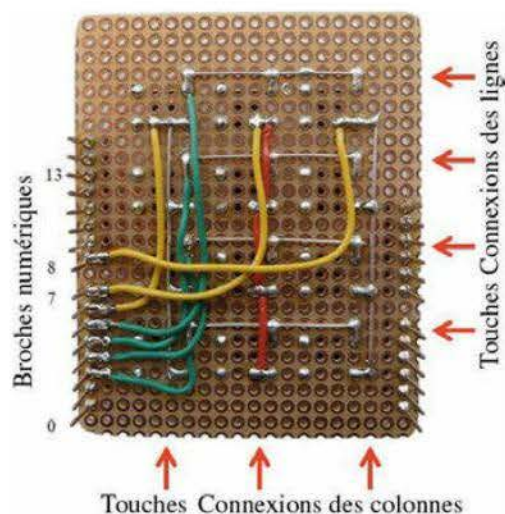
Figure 12-8 ►

Taille du shield basée sur les écarts
entre les trous



On voit sur l'image les positions exactes des connecteurs femelles et des touches. Il suffit de compter les trous sur la carte et de positionner ensuite les composants. Ne commencez à souder que quand vous avez tout placé sur la carte. Vous évitez ainsi les positionnements incorrects, et les erreurs vous sautent tout de suite aux yeux. Si vous soudez les composants aussitôt après les avoir placés, il se peut que

vous vous aperceviez plus tard que vous avez commis une erreur, et que vous ayez tout à dessouder. La figure 12-9 illustre l'envers de la carte une fois tous les composants soudés et tous les fils raccordés.



◀ **Figure 12-9**
Envers de la carte

Les fils verts établissent les liaisons vers les lignes, et les fils jaunes vers les colonnes. Les fils rouges sont les connexions intermédiaires des colonnes, qui passent au-dessus des fils horizontaux.

Problèmes courants

La réalisation de ce shield nécessitant beaucoup de soudure, les erreurs peuvent être d'autant plus nombreuses.

- Vérifiez que les fils sont bien raccordés aux bonnes broches.
- Avez-vous correctement relié les différentes touches entre elles, de manière à ce qu'elles forment des lignes et des colonnes ? Servez-vous du schéma. Voyez s'il n'y a pas de court-circuit entre eux. Le mieux est de prendre une loupe et de vérifier chaque soudure. Un court-circuit de la taille d'un cheveu n'est souvent pas visible à l'œil nu.

Qu'avez-vous appris ?

- Vos avez vu qu'on peut fabriquer soi-même un clavier numérique avec des composants très simples et peu onéreux. Pour ceux qui ont la patience nécessaire et qui ont envie de faire par

eux-mêmes au lieu de toujours se servir des composants tout prêts vendus dans les magasins, ceci a pu être un bon début et leur a permis de montrer ou plutôt d'entretenir leur créativité.

- Le soudage qui, il y a des décennies, était excessivement pratiqué dans les premiers bricolages électroniques, n'est selon moi plus à la mode aujourd'hui. Mais j'espère du moins que l'odeur d'étain fondu et de plastique brûlé vous aura charmé, comme elle a su le faire dans ma jeunesse.
- Nous avons créé ensemble notre propre classe, qui peut servir à interroger la matrice de touches. Vous avez certainement tiré parti des principes de la POO, qui vous avaient été expliqués auparavant.

Exercice complémentaire

La bibliothèque `KeyPad` fait pour le moment partie du sketch que vous avez créé. Je pense que ce serait une bonne idée de la copier quelque part, à l'attention de tous les autres sketches qui pourraient en avoir besoin. Si vous ne savez plus où, relisez le montage n° 8 du dé électronique, au cours duquel vous aviez créé votre première bibliothèque. Vous y trouverez les informations nécessaires. Il faut pour ce faire rajouter le fichier `keyword.txt` dans votre bibliothèque. Entrez-y les mots-clés nécessaires, qui sont indiqués en couleurs dans l'IDE Arduino.

Un afficheur alphanumérique

Au sommaire :

- la définition d'un afficheur LCD ;
- le fonctionnement d'un tel afficheur ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un jeu ;
- un exercice complémentaire.

Qu'est-ce qu'un afficheur LCD ?

Que serait un microcontrôleur sans son afficheur pour correspondre avec le monde extérieur sans passer par l'ordinateur ou le Serial Monitor ? Bien sûr, on a déjà vu comment, par exemple, utiliser des afficheurs sept segments pour représenter des chiffres. S'il s'agit de représenter plusieurs chiffres ou des lettres ou encore des caractères spéciaux tels que par exemple *, #, %... les limites du possible sont vite atteintes. On utilise dans ces cas-là un afficheur à cristaux liquides ou LCD (*Liquid Cristal Display*) sous sa forme abrégée. Ces afficheurs contiennent des cristaux liquides capables de modifier leur orientation en fonction d'une tension appliquée, et de jouer ainsi plus ou moins sur l'incidence de la lumière.



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- LCD ;
- LCD Module AVR ;
- Dot matrix display.

De tels éléments d'affichage utilisent en général des motifs composés de points (*Dot-Matrix*) pour représenter à peu près tous les signes (chiffres, lettres ou caractères spéciaux). Leur taille et leur équipement varient. La figure 13-1 en montre trois différents.



Figure 13-1 ▲
Plusieurs types d'afficheurs LCD

Le premier afficheur, LCD4884, avec une résolution de 84×48 pixels est déjà monté sur un shield ; il dispose d'un joystick miniature pour naviguer dans les menus, et peut être commandé directement avec la bibliothèque appropriée. Il peut même afficher des graphiques miniatures et se veut donc très souple pour représenter des éléments d'affichage. Le deuxième afficheur, *DMC-2047*, est équipé de 4 LED et d'une diode réceptrice à infrarouge (*IR*). Le troisième, de type HMC16223SG, est un afficheur à deux lignes avec un contrôleur compatible à celui de l'Hitachi *HDD44780*, sur lequel nous reviendrons bientôt. Pour une utilisation plus facile, beaucoup d'afficheurs sont dotés d'un contrôleur intégré qui commande les différents points ou segments. Si nous avions à le faire, le sketch serait beaucoup plus long. Dans l'environnement Arduino, un afficheur LCD avec pilote *HD44780* est relativement souvent utilisé. Ce pilote, qui s'est imposé comme le quasi-standard, est souvent adapté par beaucoup d'autres fabricants. La figure 13-2 montre un afficheur de ce type.



◀ **Figure 13-2**
Afficheur LCD

Il existe pour cet élément une bibliothèque livrée avec l'IDE Arduino. Vous pouvez bien entendu raccorder n'importe quel afficheur ou presque, à condition de trouver une bibliothèque appropriée ou de la créer vous-même. Nous utilisons pour notre montage l'afficheur ci-dessus, à 2 lignes de 16 caractères chacune.

Composants nécessaires



1 LCD HD44780 + barrette de 16 broches
au pas de 2,54 mm



1 trimmer de 10 k Ω ou 20 k Ω

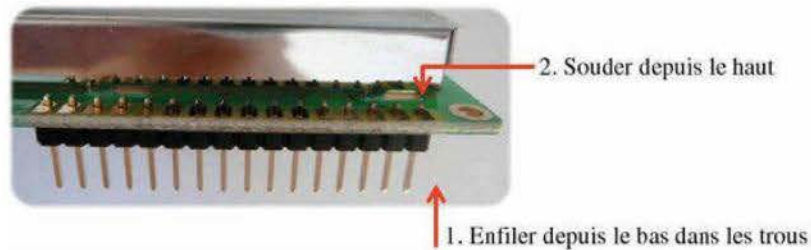


Plusieurs cavaliers flexibles de couleurs
et de longueurs diverses

Remarque préliminaire sur l'utilisation de l'afficheur LCD

Si vous achetez un afficheur LCD tout neuf, il se peut que seuls des trous de connexion soient présents sur le circuit imprimé, comme vous pouvez le voir sur l'image ci-dessus. Vous pouvez alors, soit équiper de fils les contacts nécessaires et vous en servir plus tard pour le circuit sur la plaque d'essais, soit – et c'est le mieux – vous procurer une barrette à broches, comme vous pouvez le voir également sur l'image suivante.

Des barrettes sont par exemple proposées avec une rangée de 40 broches et un pas de 2,54 mm. Coupez-les à une longueur de 15 broches en les pliant délicatement à l'endroit souhaité. Allez-y doucement car elles ont tendance à casser là où on ne veut pas. Enflez ensuite les broches de la barrette depuis le bas dans les trous et soudez-les sur la face supérieure.



Vous pouvez ainsi brancher sans problème le module sur votre plaque d'essais.

Principes intéressants

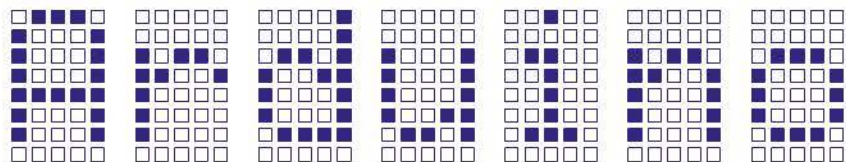
Avant d'utiliser l'afficheur LCD, voici quelques principes importants et intéressants à connaître. Comment un afficheur de ce type fonctionne-t-il ? Nous avons déjà vu que les différents caractères étaient composés à partir d'une matrice de points (*Dot-Matrix*). *Dot* signifie *point* et se trouve être le plus petit élément représentable dans cette matrice. Tout caractère est construit avec une matrice de points 5×8 .

Figure 13-3 ►
La matrice de points 5×8
de l'afficheur LCD



Un emploi judicieux des différents points permet de générer les caractères les plus divers. La figure 13-4 montre le mot Arduino et les différents points à partir desquels les lettres sont composées.

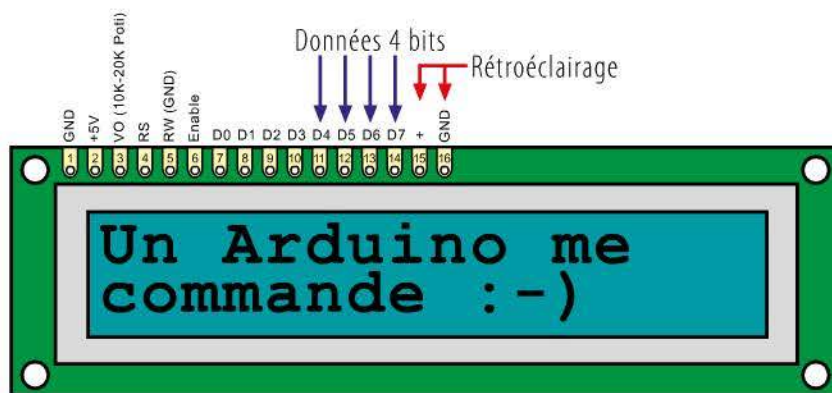
Figure 13-4 ►
Le mot Arduino composé à partir
des différents points



La commande de l'afficheur est *parallèle*, autrement dit tous les bits de données sont envoyés en même temps au contrôleur. Il existe deux modes différents (4 bits et 8 bits), le mode 4 bits étant le plus utilisé parce qu'un nombre moindre de lignes de données doit être relié à l'afficheur, ce qui fait diminuer le coût.

Eh là, pas si vite ! Je ne suis pas idiot au point d'ignorer que si j'utilise 4 bits au lieu de 8, j'aurai un débit de données moindre et je pourrai transmettre moins d'informations différentes. Comment faire alors ?

C'est juste, Arduus ! Mais ça fonctionne sans diminution du volume d'informations. En mode 4 bits, les 8 bits d'informations à transmettre sont simplement scindés en deux moitiés : l'une contenant les quatre premiers bits, et l'autre contenant les quatre derniers bits. Un nombre binaire de 4 bits est appelé *nibble* (quartet) dans le traitement des données. Les 4 bits d'un nibble sont transmis en parallèle, et les deux nibbles d'un octet en série. Surtout, ne vous en faites pas. Le mode 4 bits est certes plus lent que le mode 8 bits, mais ça n'a ici aucune importance. Venons-en maintenant au module d'affichage LCD Hitachi *HDD44780*, à son brochage et aux branchements nécessaires. Il existe deux variantes différentes : celle à 16 broches qui possède un rétroéclairage, et celle à 14 broches qui n'en a pas besoin.



◀ **Figure 13-5**
Branchements du module d'affichage

Sur les huit lignes de données, seules les quatre lignes supérieures (D4 à D7) sont nécessaires. Le tableau 13-1 donne l'affectation des broches et leur signification.

Tableau 13-1 ►
Occupation des broches LCD
pour la variante à 16 broches

Broche LCD	Broche Arduino	
1	GND	Masse
2	+ 5 V	+ 5 V
3	-	Réglage du contraste par un potentiomètre de 10 k Ω ou 20 k Ω
4	12	RS (Register Select)
5	GND	RW (Read/Write)/connecté à la masse (HIGH : Read/LOW : Write)
6	11	E (Enable)
11	5	Ligne de données D4
12	4	Ligne de données D5
13	3	Ligne de données D6
14	2	Ligne de données D7
15	-	Anode (+)/via résistance série 220 Ω !
16	GND	Cathode (-)

L'objectif du premier sketch LCD est de faire apparaître à l'écran la phrase « Un Arduino me commande : -) ».

Code du sketch

Ne vous laissez pas effrayer par la logique de commande relativement complexe. Nous allons nous servir d'une bibliothèque, qui nous permettra d'utiliser un afficheur LCD de manière très simple.

```
#include <LiquidCrystal.h>
#define RS 12 //Register Select
#define E 11 //Enable
#define D4 5 //Ligne de données 4
#define D5 4 //Ligne de données 5
#define D6 3 //Ligne de données 6
#define D7 2 //Ligne de données 7
#define COLS 16 //Nombre de colonnes
#define ROWS 2 //Nombre de lignes
LiquidCrystal lcd(RS, E, D4, D5, D6, D7); //Instanciation de l'objet

void setup(){
  lcd.begin(COLS, ROWS); //Nombres de colonnes et lignes
  lcd.print("Un Arduino me"); //Affichage du texte
  lcd.setCursor(0, 1); //Passer à la 2e ligne
  lcd.print("commande :-)"); //Affichage du texte
}

void loop(){ /* vide */}
```

Revue de code

La bibliothèque `LiquidCrystal` doit être incorporée afin de pouvoir utiliser la fonctionnalité des commandes de l'afficheur LCD. Du point de vue logiciel, la variable suivante est nécessaire à notre montage.

Variable	Rôle
<code>lcd</code>	L'objet LCD

◀ **Tableau 13-2**
Variable nécessaire et son rôle

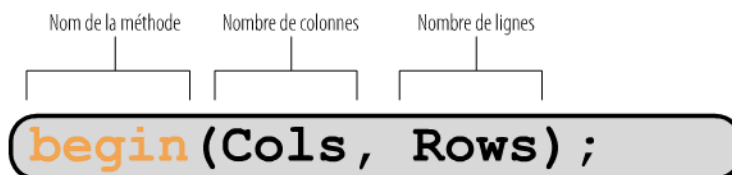
Les paramètres suivants doivent être communiqués au constructeur pour générer un objet LCD :

- broche Register Select (RS) ;
- broche Enable (E) ;
- broches des lignes de données D4 à D7.

```
LiquidCrystal lcd(RS, E, D4, D5, D6, D7); //Instanciation de l'objet
```

La classe `LiquidCrystal` met une série de méthodes à disposition, car on ne peut envoyer un texte à l'afficheur LCD avec le seul constructeur. Pour que ce soit possible, il nous faut transmettre à l'objet afficheur quelques informations supplémentaires pour poursuivre l'initialisation. Les afficheurs LCD diffèrent quant au nombre de colonnes ou de lignes, et ce sont précisément ces informations qu'il lui faut. On voit bien que le constructeur ne dispose pas de tout pour une initialisation complète. Une méthode est ici nécessaire.

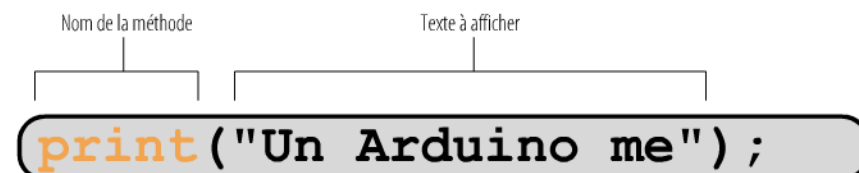
Méthode LCD : begin



◀ **Figure 13-6**
Méthode LCD begin

La méthode `begin` communique les nombres de colonnes et de lignes à l'objet LCD. Tout est alors prêt pour envoyer un texte.

Méthode LCD : print



◀ **Figure 13-7**
Méthode LCD print

La méthode `print` indique à l'objet LCD ce qui doit être affiché à l'écran. Elle est comparable à celle du Serial Monitor.

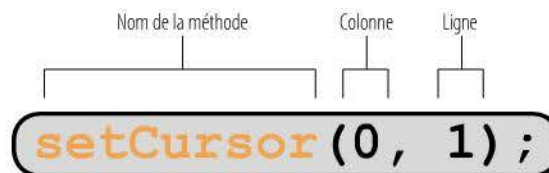


Un moment, s'il vous plaît ! L'afficheur que vous utilisez a bien deux lignes. Comment avez-vous fait pour que le texte s'affiche sur la première ?

Quand aucune indication n'est donnée sur la position du texte à afficher, celui-ci se place au début de la première ligne. Comme vous pouvez le voir dans l'exemple, une autre ligne est occupée par du texte. Venons-en maintenant à la troisième méthode importante.

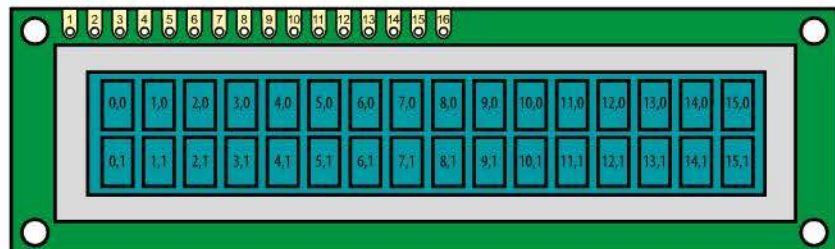
Méthode LCD : `setCursor`

Figure 13-8 ►
Méthode LCD `setCursor`



La méthode `setCursor` permet de positionner le curseur à l'endroit où le texte suivant doit commencer. Elle est ici aussi – et comment pourrait-il en être autrement – basée sur zéro, autrement dit la première ligne ou colonne est pourvue de l'index 0. Pour atteindre la deuxième ligne, vous devez – comme c'est le cas ici – utiliser la valeur 1. La figure 13-9 peut vous aider à positionner l'affichage.

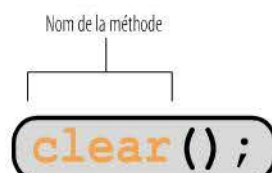
Figure 13-9 ►
Coordonnées des différentes lignes accessibles avec `setCursor`



Avant d'oublier : vous pouvez naturellement tout effacer jusqu'au dernier caractère avec la méthode `clear`.

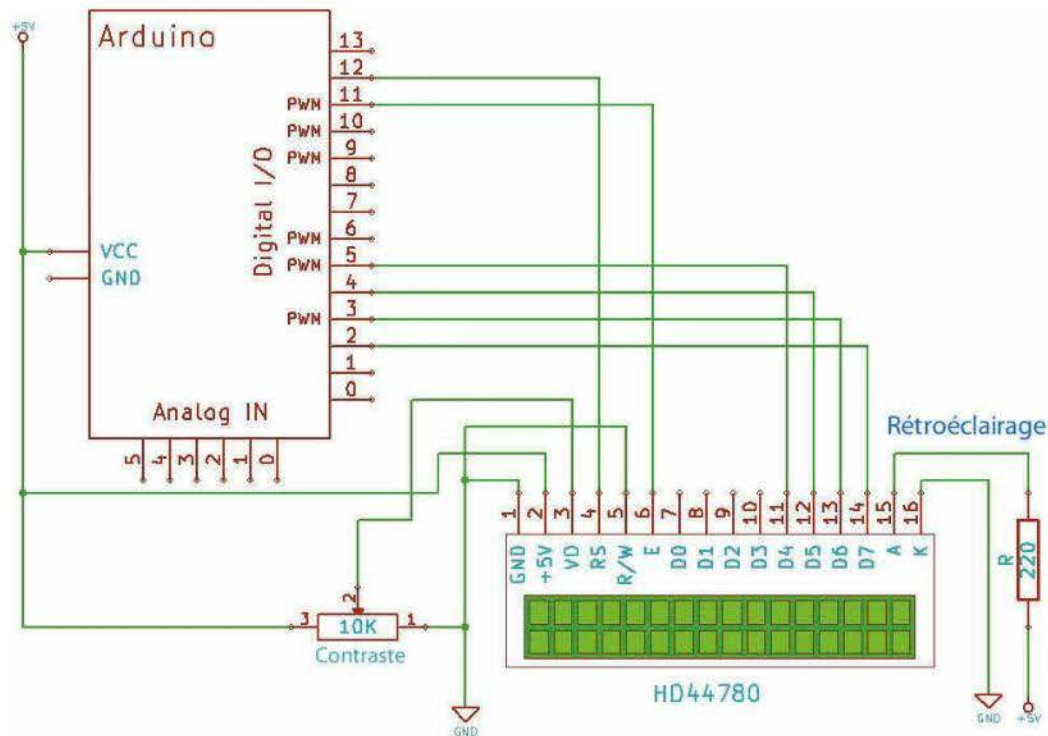
Méthode LCD : `clear`

Figure 13-10 ►
Méthode LCD `clear`



Elle n'a pas de paramètre, efface tous les caractères de l'afficheur et positionne le curseur sur la coordonnée 0,0 dans le coin supérieur gauche.

Schéma



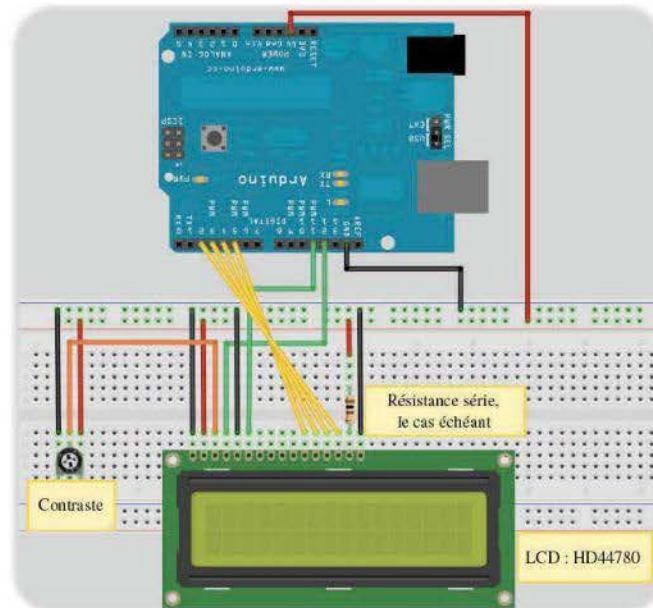
▲ Figure 13-11
Connexions de l'afficheur LCD

Attention !

Dans certaines variantes du HD44780, on peut brancher le rétroéclairage sur +5V sans résistance série ; dans d'autres, une résistance dimensionnée en conséquence est nécessaire. Regardez la fiche technique avant de brancher la tension d'alimentation. Vous pouvez au pire laisser tomber le rétroéclairage. Si c'est trop sombre, vous pourrez toujours augmenter le contraste de manière à pouvoir lire quand même l'affichage.

Réalisation du circuit

Figure 13-12 ►
Réalisation de la commande du LCD
avec Fritzing



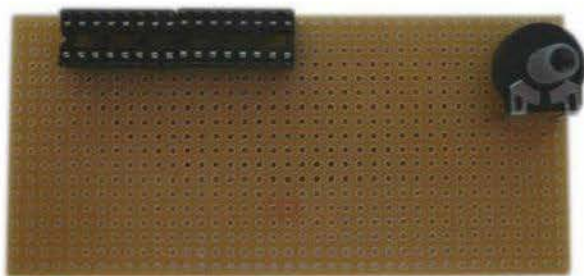
Jeu : deviner un nombre

Quoi de mieux que le jeu où il faut deviner des nombres pour une réalisation avec l'afficheur LCD ? Si ça fonctionne, vous n'aurez plus besoin d'un ordinateur et gagnerez en indépendance grâce à l'unité d'affichage LCD. J'ai fixé, pour le besoin de la réalisation, le LCD sur une carte de circuit imprimé perforée, sur laquelle deux supports de circuit intégré à 16 broches sont posés l'un à côté de l'autre.

Figure 13-13 ►
Support de circuit intégré à
16 broches

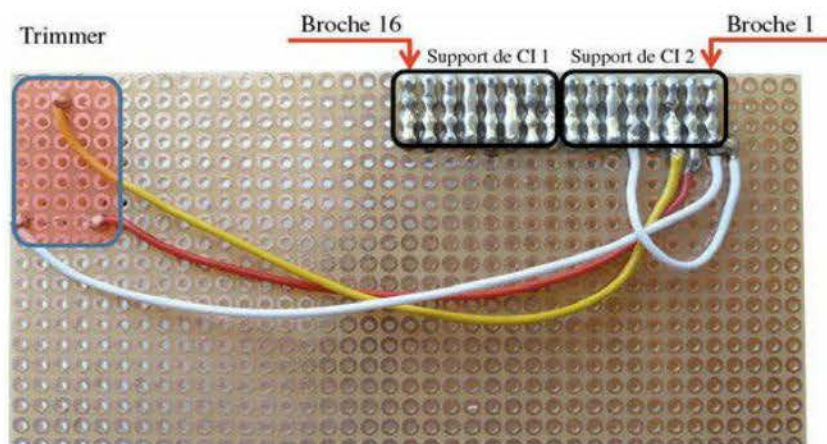


Un support comparable – mais avec plus de broches bien sûr – se trouve sur votre carte Arduino et maintient le microcontrôleur en position. De tels connecteurs sont vraiment utiles car si un circuit intégré vient à griller pour de bon, plus besoin de le dessouder péniblement. Il suffit de le remplacer. La carte mesure 10 × 5 cm.



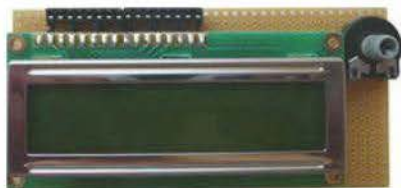
◀ **Figure 13-14**
Carte-support pour l'afficheur LCD

Comme vous pouvez le constater, j'ai également placé dessus le trimmer pour le réglage du contraste. De l'autre côté de la carte, on voit comment j'ai relié entre elles les différentes broches des supports de circuit intégré. Les broches opposées ont toutes été reliées par plusieurs points de soudure.



◀ **Figure 13-15**
Face soudée de la carte-support pour l'afficheur LCD

Un *câblage volant* est parfois suffisant. La figure 13-16 montre l'afficheur LCD déjà fixé sur la carte. Ses broches sont insérées dans les contacts de la rangée inférieure des deux supports de circuit intégré. La rangée supérieure servira plus tard pour la connexion au shield.



◀ **Figure 13-16**
Afficheur LCD sur sa carte-support

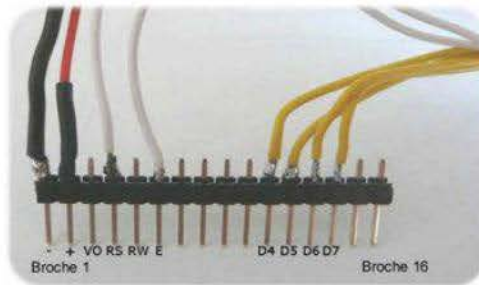
Il ne manque plus maintenant que les lignes de raccordement à votre clavier analogique. Les liaisons passent par les barrettes à broches que vous connaissez bien.

Il vous faut :

- 1 barrette à 16 broches ;
- 2 barrettes à 8 broches ;
- 1 barrette à 6 broches.

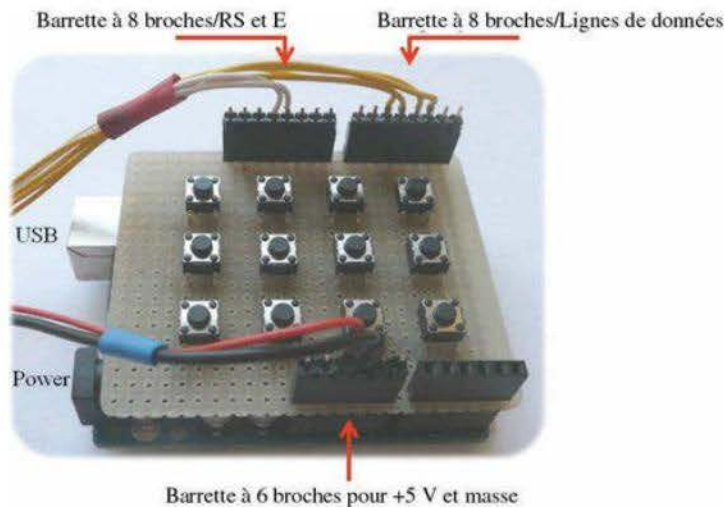
La barrette à 16 broches est raccordée à la carte-support sur laquelle se trouve l'afficheur LCD. La figure 13-17 illustre les lignes de raccordement analogique.

Figure 13-17 ►
Barrette à 16 broches

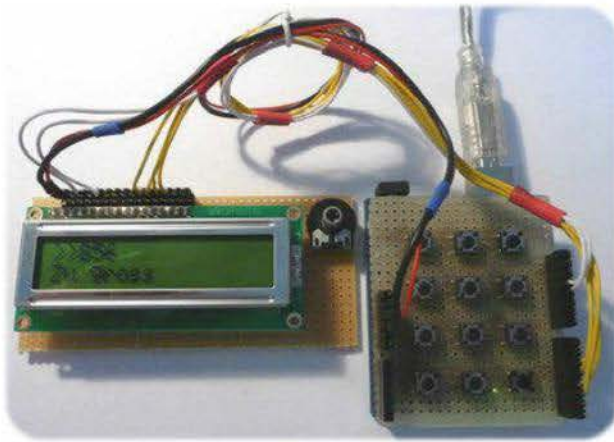


Les deux barrettes à 8 broches et la barrette à 6 broches sont branchées sur le clavier.

Figure 13-18 ►
Clavier analogique avec ses trois
barrettes à broches



En utilisant les couleurs et l'affectation des broches indiquée, vous ne devriez pas avoir de problème pour construire le petit faisceau de câbles avec les barrettes à broches. La figure 13-19 montre à nouveau les trois composants, à savoir la carte-support avec l'afficheur LCD, la carte Arduino et le shield du clavier superposés, tous reliés entre eux.

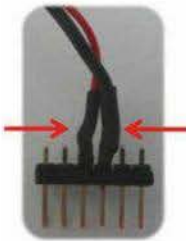


◀ **Figure 13-19**
Réalisation du circuit complet pour
le jeu des nombres à deviner



Attention !

Si vous soudez sur les barrettes à broches des fils d'alimentation ou de masse qui se trouvent directement l'un à côté de l'autre, vous risquez fort d'avoir un jour ou l'autre, par déplacement, un court-circuit entre ces contacts ou les fils avoisinants. Aussi ai-je pris soin de mettre chaque soudure sous gaine thermo-rétractable pour l'isoler.



◀ **Figure 13-20**
Barrette à 6 pôles avec deux
morceaux de gaine thermo-
rétractable (flèches rouges)

Voici maintenant le code complet, qui est déjà un peu plus conséquent.

```
#include <LiquidCrystal.h>
#include "MyAnalogKeyPad.h"
#define analogPinKeyPad 0 //Définition de la broche analogique
#define MIN 10 //Limite inférieure du nombre aléatoire
#define MAX 1000 //Limite supérieure du nombre aléatoire
#define RS 12 //Broche Register Select du LCD
#define E 11 //Broche Enable du LCD
#define D4 5 //Ligne de données LCD broche 4
#define D5 4 //Ligne de données LCD broche 5
#define D6 3 //Ligne de données LCD broche 6
#define D7 2 //Ligne de données LCD broche 7
#define COLS 16 //Nombre de colonnes LCD
#define ROWS 2 //Nombre de lignes LCD
int arduinoNumber, tries; //Le nombre généré, nombre d'essais
```

```

char yourNumber[5];           //Nombre à 5 chiffres maxi
byte place;
MyAnalogKeyPad myOwnKeyPad(analogPinKeyPad); //Instanciation clavier
LiquidCrystal lcd(RS, E, D4, D5, D6, D7);    //Instanciation LCD

void setup(){
  myOwnKeyPad.setDebounceTime(500); //Réglage temps
                                     //de rebond 500 ms
  lcd.begin(COLS, ROWS);           //Nombres de lignes et de colonnes
  lcd.blink();                     //Faire clignoter le curseur
  startSequence();                 //Appel de la séquence de démarrage
}

void loop(){
  char myKey = myOwnKeyPad.readKey(); //Lecture de
                                     //la touche appuyée
  if(myKey != KEY_NOT_PRESSED){      //Interrogation si une touche
                                     //quelconque appuyée

    yourNumber[place] = myKey;
    place++;
    lcd.print(myKey);                //Afficher la touche sur le LCD
  }

  if(place == int(log10(MAX))+1){
    tries++;
    int a = atoi(yourNumber);
    if(a == arduinoNumber){
      lcd.clear();                  //Effacer écran LCD
      lcd.print("Exact !!!");       //Affichage sur LCD
      lcd.setCursor(0, 1);          //Positionnement curseur
                                     //sur 2e ligne
      lcd.print("Essai:" + String(tries));
      delay(4000);                  //Attendre 4 secondes
      tries = 0;                    //Remise à zéro du nombre d'essais
      startSequence();              // Appel de startSequence
    }
    else if(a < arduinoNumber){
      lcd.setCursor(0, 1);           //Positionnement curseur sur
                                     //2e ligne
      lcd.print("Trop petit");       //Affichage sur LCD
      lcd.setCursor(0, 0);           //Positionnement curseur sur
                                     //1re ligne
    }
    else{
      lcd.setCursor(0, 1);           //Positionnement curseur sur
                                     //2e ligne
      lcd.print("Trop grand");       //Affichage sur LCD
      lcd.setCursor(0, 0);           //Positionnement curseur sur

```

```

//1re ligne
}
lcd.setCursor(2, 0);      //Positionnement curseur sur
                           //3e emplacement de la 1re ligne
place = 0;
}
}

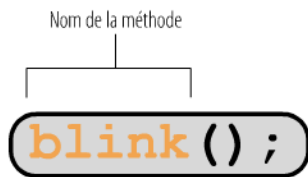
int randomNumber(int minimum, int maximum){
  randomSeed(analogRead(5));
  return random(minimum, maximum + 1);
}

void startSequence(){
  arduinoNumber = randomNumber (MIN, MAX); //Générer le nombre
                                           //à deviner
  lcd.clear();                          //Effacer écran LCD
  lcd.print("Devine un nombre");         //Affiche sur LCD
  lcd.setCursor(0, 1);                  //Positionnement curseur
                                           //sur 2e ligne
  lcd.print("de" + String(MIN) + " - " + String(MAX));
  delay(4000);                          //Attendre 4 secondes
  lcd.clear();                          //Effacer écran LCD
  lcd.print(">>");                       //Affiche sur LCD
}

```

Je ne souhaite pas trop m'étendre sur le sketch pour l'instant. J'ai fait en sorte que l'affichage ait lieu sur le LCD. J'ai ajouté aussi une méthode qui affiche un curseur clignotant à l'écran (figure 13-21). La fonction `atoi` (**A**SCII **t**o **I**nteger) convertit une chaîne de caractères en un entier.

Méthode LCD : blink



◀ **Figure 13-21**
Méthode LCD blink

`blink` est appelée une seule fois dans la fonction `setup` et fait clignoter un curseur à l'endroit où on va écrire. Quand on regarde le début du sketch, on voit qu'il est tout à fait possible d'incorporer plusieurs bibliothèques dans un projet.

Il n'y a théoriquement aucune limite. La mémoire flash finit quand même à un moment par laisser entendre qu'elle est pleine et qu'aucun code ne peut plus être ajouté.



Il y a une chose que je ne comprends pas et je ne sais plus si vous l'avez déjà expliquée ou pas. On trouve par exemple la ligne :

```
lcd.print("de" + String(MIN) + "-" + String(MAX));
```

Autrement dit, on affiche des chaînes de caractères et on utilise l'opérateur +. Mais comment fait-on pour additionner des chaînes de caractères ? Ça ne marche qu'avec des nombres, non ?

Bien sûr, Arduus ! Seules des valeurs mathématiques peuvent être additionnées. L'opérateur + ne peut évidemment rien additionner dans le cas de chaînes de caractères. Comment le pourrait-il ? Les différentes chaînes de caractères sont simplement réunies en une seule. On dit aussi qu'elles sont concaténées. Si maintenant, comme dans notre sketch, des valeurs numériques font partie de la chaîne de caractères à afficher, elles doivent être préalablement converties en une string. C'est la fonction string qui s'en charge, comme dans `String(MIN)`.



Pour aller plus loin

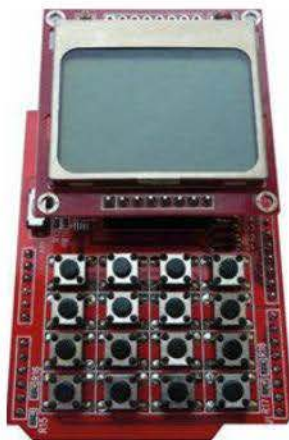
Voici quelques liens intéressants à propos d'Arduino et de LCD :

- <http://www.arduino.cc/en/Tutorial/LiquidCrystal>
- <http://www.arduino.cc/en/Reference/LiquidCrystal>
- <http://www.sparkfun.com/datasheets/LCD/HD44780.pdf>
- <http://arduino.cc/fr> (en français)

La figure 13-22 montre encore un shield pour clavier prêt à l'emploi, capable de recevoir un LCD.

Figure 13-22 ►

Shield pour clavier 4 × 4 avec interface pour afficheur 5110 LCD



Pour du prêt à l'emploi somme toute relativement compact, le shield pour clavier 4 × 4 se veut idéal avec un porte-écran approprié. L'écran monochrome, qui a une résolution de 84 × 48 pixels, est

compatible avec l'afficheur 3310 LCD, pour lequel une bibliothèque Arduino existe.



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- Nokia 5110 LCD ;
- 3310 LCD.

Problèmes courants

Si, après avoir raccordé le LCD et chargé le sketch, vous ne voyez rien à l'écran, vérifiez les points suivants.

- Le câblage est-il correct ?
- Pas de court-circuit ?
- Le trimmer du contraste est-il correctement branché ?
Augmentez le cas échéant le contraste jusqu'à ce que vous puissiez voir quelque chose à l'écran.

Qu'avez-vous appris ?

- Vous avez raccordé pour la première fois un élément d'affichage, capable non seulement de clignoter mais aussi d'afficher des nombres et du texte.
- La bibliothèque `LiquidCrystal` vous a permis de commander facilement un LCD avec un contrôleur HDD44780.
- Vous avez ensuite clairement transposé le jeu des nombres à deviner.
- D'autres types de LCD vous ont été présentés pour vos expériences à venir, de telle sorte que vos créations puissent être sans limite.

Exercice complémentaire

Réfléchissez un peu à une serrure à code de sécurité, du type de celles installées aux entrées des zones sensibles.

Un code à plusieurs chiffres doit être saisi pour ouvrir la porte. On est bien entendu informé en cas de saisie erronée que le code chiffré

composé est trop bas ou trop élevé. Vous pouvez par exemple brancher un servomoteur désengageant un pêne du système de fermeture en cas de code correct. Il faut attendre un certain temps, par exemple trois minutes, après avoir tapé trois fois de suite un code erroné. Créez un contrôle d'accès à votre chambre pour décourager les colocataires ou proches trop curieux.

Le moteur pas-à-pas

Au sommaire :

- comprendre ce qu'est un moteur pas-à-pas et savoir le commander ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Encore plus de mouvement

Un servomoteur permet de transformer le courant électrique en mouvement. Cependant, son rayon d'action demeure limité, même si des modifications peuvent être entreprises pour combler ce manque. Mais il s'avère suffisant pour la plupart des applications.

Le moteur pas-à-pas s'impose en revanche si une plus grande liberté d'action est nécessaire. Voyez par souci d'économie si vous ne pouvez pas en récupérer un sur un vieil appareil quelconque :

- imprimante ;
- scanner à plat ;
- lecteur de CD/DVD ;
- ancien lecteur de disquettes (de 3,5 pouces).

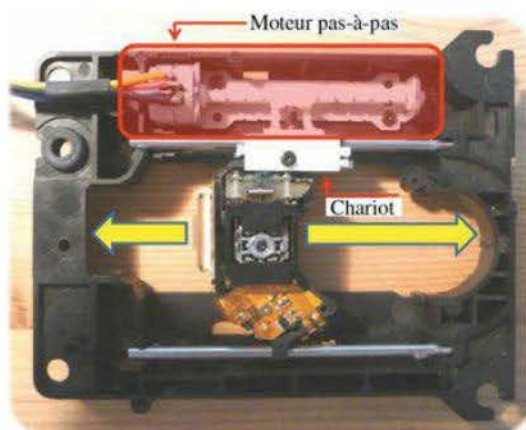
Sur la figure 14-1, vous pouvez voir un lecteur de disquettes de 3,5 pouces, encore parfois utilisé dans les ordinateurs actuels.

Figure 14-1 ►
Lecteur de disquette de 3,5 pouces



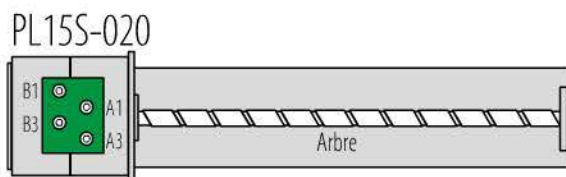
Ces lecteurs possèdent un petit moteur pas-à-pas, de type PL15S-020 la plupart du temps, qui entraîne un petit chariot sur lequel se trouve la tête d'écriture/lecture. La figure 14-2 montre une unité de ce genre provenant d'un ancien lecteur de CD-Rom.

Figure 14-2 ►
Moteur pas-à-pas PL15S-020
provenant d'un ancien lecteur
de CD-Rom



Le moteur pas-à-pas dispose de 4 bornes, sur lesquelles nous allons revenir en détail. Comme vous pouvez le voir sur la figure, j'ai déjà soudé deux fils de couleur pour qu'il soit plus facile de le commander à la main avec la carte Arduino. Le graphique 14-3 montre les noms utilisés pour ces bornes.

Figure 14-3 ►
Branchements du moteur pas-à-pas
PL15S-020



Effectivement, Arduus ! N'empêche que je ne l'ai pas oubliée. La mention DNE derrière L293 signifie que les diodes de protection, appelées également diodes de roue libre, sont déjà intégrées dans le circuit. C'est naturellement très pratique ! Si vous avez encore un ancien circuit L293 (sans mention DNE derrière), il faut absolument brancher les diodes de protection en externe. Faute de quoi, la carte Arduino sera endommagée.

Composants nécessaires



1 circuit de commande de moteur L293DNE



1 moteur pas-à-pas bipolaire (par exemple, PL155-020 provenant d'un ancien lecteur de CD/DVD)



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Qu'est-ce que je fais si je ne trouve pas de lecteur de disquette ou CD/DVD ? Je ne peux pas faire l'expérimentation alors.

Pas de panique, Arduus ! Vous pouvez en fait prendre pratiquement n'importe quel moteur pas-à-pas bipolaire. Il vous suffit de trouver la fiche technique correspondante sur Internet pour en connaître les spécifications. Attention cependant au courant consommé par le moteur pas-à-pas en service ; comparez-le à celui pour le circuit de commande de moteur utilisé ici. Il ne doit en aucun cas dépasser 600 mA par borne. Faute de quoi, vous devez prendre soit un autre moteur pas-à-pas, soit un autre circuit de commande.

Quand je regarde le schéma, j'ai comme un léger problème avec la source de tension externe qui, d'après les données du moteur pas-à-pas, doit être de 5 V. Où voulez-vous que je la trouve ?

Vous devez utiliser soit une alimentation de laboratoire réglable, soit – et c'est encore moins cher – un bloc d'alimentation secteur (voir chapitre 8).



Code du sketch

```
#define Stepper_A1 5 //Broche pour connexion A1
#define Stepper_A3 4 //Broche pour connexion A3
#define Stepper_B1 3 //Broche pour connexion B1
#define Stepper_B3 2 //Broche pour connexion B3

byte stepValues[5][4] = {{LOW, LOW, LOW, LOW}, //Moteur à l'arrêt
                          {LOW, HIGH, HIGH, LOW}, //Pas 1
                          {LOW, HIGH, LOW, HIGH}, //Pas 2
                          {HIGH, LOW, LOW, HIGH}, //Pas 3
                          {HIGH, LOW, HIGH, LOW}}; //Pas 4

void setup(){
  pinMode(Stepper_A1, OUTPUT);
  pinMode(Stepper_A3, OUTPUT);
  pinMode(Stepper_B1, OUTPUT);
  pinMode(Stepper_B3, OUTPUT);
  for(int i = 0; i < 10; i++){
    action(30, 2); //30 pas vers la droite avec pause de 2 ms
    action(-30, 10); //30 pas vers la gauche avec pause de 10 ms
  }
  action(0, 0); //Mise hors courant
}

void loop() { /* vide */}

void action(int count, byte delayValue){
  if(count > 0) //Rotation vers la droite
    for(int i = 0; i < count; i++)
      for(int sequenceStep = 1; sequenceStep <=4; sequenceStep++)
        moveStepper(sequenceStep, delayValue);
  if(count < 0) //Rotation vers la gauche
    for(int i = 0; i < abs(count); i++)
      for(int sequenceStep = 4; sequenceStep > 0; sequenceStep--)
        moveStepper(sequenceStep, delayValue);
  if(count == 0) //Mise hors courant
    moveStepper(0, delayValue);
}

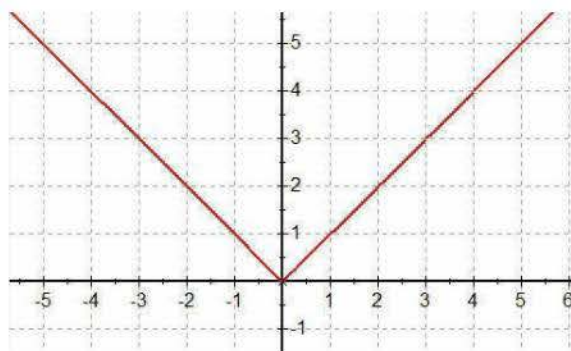
void moveStepper (byte s, byte delayValue){
  digitalWrite(Stepper_A1, stepValues[s][0]);
  digitalWrite(Stepper_A3, stepValues[s][1]);
  digitalWrite(Stepper_B1, stepValues[s][2]);
  digitalWrite(Stepper_B3, stepValues[s][3]);
  delay(delayValue); //Pause
}
```

Il me semble que vous utilisez dans ce sketch une fonction inconnue appelée *abs*. Pouvez-vous m'en dire un peu plus ?

Ah oui, c'est vrai. Si vous appliquez la fonction *abs* – qui est la forme abrégée de *absolute* – à un nombre réel, le signe n'est tout simplement pas pris en compte. Le résultat est toujours positif. Les mathématiciens formulent cet état comme suit :

$$|x| = \begin{cases} x & \text{pour } x \geq 0 \\ -x & \text{pour } x < 0 \end{cases}$$

Les deux barres verticales de chaque côté du *x* signifient « valeur absolue de *x* ». Le fonctionnement est plus clair grâce à cette représentation graphique que j'ai créée pour la fonction *abs* (voir figure ci-après).



Revue de code

D'un point de vue logiciel, les variables suivantes sont nécessaires à notre expérimentation du moteur pas-à-pas.

Variable	Rôle
<code>stepValues[5][4]</code>	Tableau bidimensionnel pour stocker les informations de pas servant à déplacer le moteur

◀ **Tableau 14-1**
Variable nécessaire et son rôle

Le contenu du tableau correspond exactement aux valeurs du tableau des séquences de commande. Seule une ligne avec des valeurs *LOW* a été ajoutée au début, laquelle sert à mettre le moteur pas-à-pas hors courant, une fois la position demandée atteinte.

Si je ne le faisais pas, le moteur s'immobiliserait bien à la fin, mais nous aurions affaire à un blocage à la dernière position occupée. Un tel moteur ne peut plus être bougé à la main, car il est encore sous

tension. Par ailleurs, cela signifie qu'il devient vite très chaud voire brûlant.

```
byte stepValues[5][4] = {{LOW, LOW, LOW, LOW}, //Moteur à l'arrêt
                          {LOW, HIGH, HIGH, LOW}, //Pas 1
                          {LOW, HIGH, LOW, HIGH}, //Pas 2
                          {HIGH, LOW, LOW, HIGH}, //Pas 3
                          {HIGH, LOW, HIGH, LOW}}; //Pas 4
```

Voyons tout d'abord la fonction `moveStepper` qui déplace le moteur pas-à-pas. Elle comporte deux arguments.

- Le premier indique le pas dans la séquence, soit de 1 à 4 pour une rotation à droite et de 4 à 1 pour une rotation à gauche.
- Le second fixe un temps d'attente entre les pas. Vous pouvez ainsi influencer un peu sur la vitesse du moteur pas-à-pas. Cette valeur ne doit cependant pas être inférieure à 2, car la commande électrique est alors si rapide que le moteur n'a plus le temps de réagir mécaniquement. Il se contente de bourdonner et de vibrer.

```
void moveStepper(byte s, byte delayValue){
  digitalWrite(Stepper_A1, stepValues[s][0]);
  digitalWrite(Stepper_A3, stepValues[s][1]);
  digitalWrite(Stepper_B1, stepValues[s][2]);
  digitalWrite(Stepper_B3, stepValues[s][3]);
  delay(delayValue); //Pause
}
```

À l'intérieur de la fonction, le pas est transmis comme index pour la première dimension du tableau de séquence `stepValues`. La deuxième dimension indique les niveaux de tension LOW ou HIGH. Ils sont dûment appelés au moyen des valeurs d'index de 0 à 3 et sont transmis aux sorties numériques qui, à leur tour, contrôlent le moteur pas-à-pas via le circuit de commande. Passons maintenant à la fonction `action`, qui appelle la fonction `moveStepper` :

```
void action(int count, byte delayValue){
  if(count > 0) //Rotation vers la droite
    for(int i = 0; i < count; i++)
      for(int sequenceStep = 1; sequenceStep <=4; sequenceStep++)
        moveStepper(sequenceStep, delayValue);
  if(count < 0) //Rotation vers la gauche
    for(int i = 0; i < abs(count); i++)
      for(int sequenceStep = 4; sequenceStep > 0; sequenceStep--)
        moveStepper(sequenceStep, delayValue);
  if(count == 0) //Mise hors courant
    moveStepper(0, delayValue);
}
```

Le nombre de pas et le temps de pause après chaque pas lui sont communiqués. Le moteur tourne vers la droite si la valeur du pas est positive, et vers la gauche si elle est négative. Le moteur est mis hors courant si la valeur est 0. Deux boucles `for` imbriquées travaillent toujours main dans la main pour faire tourner le moteur. La boucle extérieure régit le nombre de pas et la boucle intérieure fixe le sens de rotation. La boucle intérieure traite les pas de 1 à 4 si la valeur du pas est positive, et de 4 à 1 si elle est négative. Cette séquence sert d'index à la fonction `moveStepper`, index avec lequel le tableau `stepValues` détermine les valeurs `LOW` ou `HIGH` correspondantes. La demande proprement dite de mouvement du moteur pas-à-pas se fait par l'appel de la fonction `action` avec une ligne de code comme suit :

```
action(30, 2);
```

Elle dit au moteur pas-à-pas : « Tourne de 30 pas vers la droite et observe une pause de 2 ms entre chaque pas ! » La ligne :

```
action(-30, 10);
```

dit en revanche : « Tourne de 30 pas vers la gauche et observe une pause de 10 ms entre chaque pas ! »

Le moteur pas-à-pas peut ainsi être déplacé à l'endroit souhaité. Cela étant, pensez aux limites mécaniques, car plus à gauche que le minimum ou plus à droite que le maximum n'est tout simplement pas possible. Dans ces cas-là, rien n'y fera, pas même une tension plus élevée.

Bibliothèque pour moteurs pas-à-pas prête à l'emploi

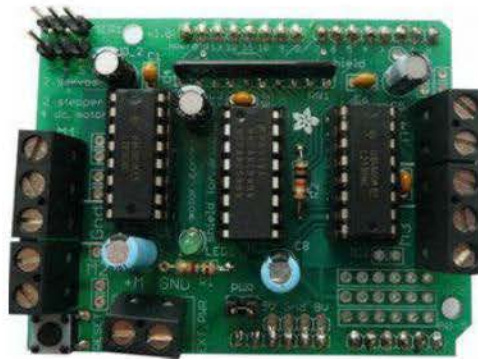
Il existe une bibliothèque prête à l'emploi, avec laquelle vous pouvez commander des moteurs pas-à-pas sans avoir à vous préoccuper de la programmation. Elle a pour nom `Stepper` et figure dans le pack de téléchargement Arduino. Vous trouverez toutes les informations nécessaires sur <http://www.arduino.cc/en/Reference/stepper>.

Shield pour moteur prêt à l'emploi

Vous pouvez acheter un shield pour moteur prêt à l'emploi, qui utilise deux circuits de commande de moteur L293DNE dont je vous ai parlé. La commande passe par le registre à décalage 74HC595 pour ne pas utiliser trop de broches numériques. Vous n'avez donc pas à

vous en faire car toute la logique se trouve dans la bibliothèque mise à disposition, que vous trouverez sur le site Internet correspondant.

Figure 14-7 ►
Shield pour moteur



Vous pouvez raccorder des composants moteur les plus variés sur ce shield :

- 2 servomoteurs de loisir ;
- jusqu'à 4 moteurs à courant continu ;
- jusqu'à 2 moteurs pas-à-pas (unipolaire ou bipolaire).

Vous trouverez d'autres informations sur : <http://www.ladyada.net/make/mshield/>.

Problèmes courants

Si le moteur pas-à-pas ne bouge pas ou ne fait que bourdonner ou vibrer, vérifiez :

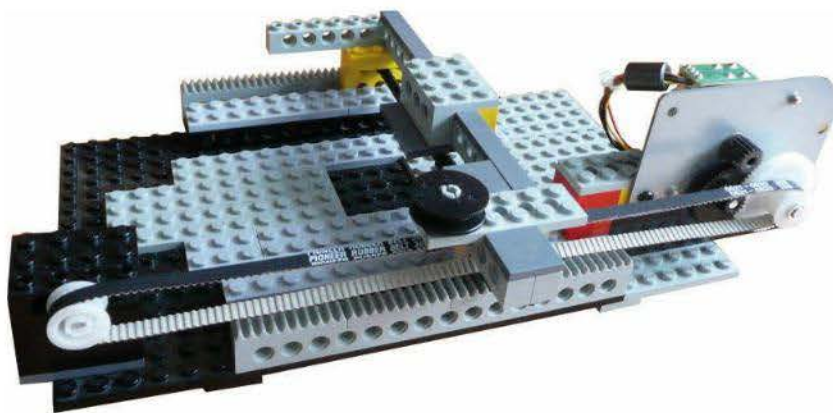
- que le câblage est correct ;
- qu'il n'y a pas de court-circuit éventuel ;
- que le moteur pas-à-pas ne change pas de position ou qu'il ne bourdonne pas ou vibre en début de sketch. Si tel est le cas, il y a de fortes chances que vous ayez interverti les quatre branchements ;
- que la connexion de masse commune est bien établie entre la carte Arduino et la source de tension externe ;
- que vous n'avez pas raccordé ensemble les deux pôles de tension d'alimentation de la carte et de la source de tension externe, qui sont marqués d'un +. Sinon, destruction de la carte Arduino assurée !

Qu'avez-vous appris ?

- Vous avez découvert comment commander un moteur pas-à-pas bipolaire.
- La commande a été réalisée au moyen du circuit de commande de moteur L293DNE.

Exercice complémentaire

La figure 14-8 montre une construction en Lego, sur laquelle j'ai monté un moteur pas-à-pas provenant d'un vieux scanner à plat.



◀ **Figure 14-8**
Moteur pas-à-pas
sur une construction en Lego

La courroie dentée et la poulie de renvoi ont également été récupérées sur le scanner. Le chariot se déplace de gauche à droite et inversement sur des crémaillères sitôt le moteur entraîné. Avec un peu d'adresse et de créativité, vous pouvez ainsi vous construire un enregistreur X-Y.

La température

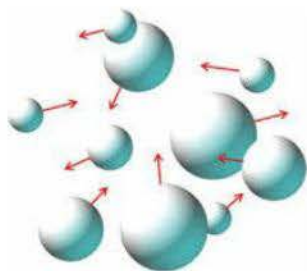
Au sommaire :

- une définition de la température ;
- savoir la mesurer ;
- le sketch complet ;
- la réalisation du circuit ;
- l'ajout d'un ventilateur au circuit ;
- un exercice complémentaire.

Chaud ou froid ?

Nous vivons dans un monde ou plutôt dans un environnement composé de matières diverses. Ces dernières peuvent en principe présenter trois états dits d'agrégation en physique. Un tel état d'agrégation peut être solide, liquide ou gazeux et dépend souvent d'une grandeur physique appelée température. Mais que signifie la température et comment se fait-elle sentir ou plutôt comment peut-on la mesurer ? Toute matière est composée d'infimes particules appelées atomes. Ceux-ci sont composés d'un nuage d'électrons (charge : négative) et d'un noyau formé de protons (charge positive) et de neutrons (charge : nulle). Ce ne sont pas là les plus petites particules, mais elles suffiront à expliquer au moyen de notre exemple ce qu'est la température.

Figure 15-1 ►
Le mouvement des atomes

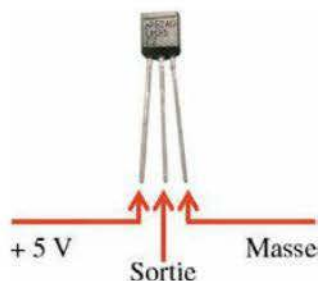


Ces infimes particules sont en perpétuel mouvement, errant apparemment sans but et dans des directions différentes. La température est donc un moyen de mesurer cette agitation thermique des atomes ou des molécules (assemblage de plusieurs atomes) d'une matière. Plus ils se déplacent rapidement, plus la probabilité est grande qu'ils entrent en collision. C'est alors que l'énergie cinétique se transforme en énergie calorifique. L'agitation thermique est donc un moyen de mesurer la température d'une matière.

Comment peut-on mesurer la température ?

On utilise des capteurs, qui convertissent la température mesurée en diverses valeurs de résistance ou de tension, desquelles on peut déduire la température ambiante. Vous avez déjà entendu parler d'une PTC et d'une NTC dans le chapitre 4 sur les bases de l'électronique. La résistance de ces composants varie en fonction de la température. Ils manquent cependant de précision, et leur courbe caractéristique n'est pas forcément linéaire. Aussi voudrais-je vous présenter un capteur de température qui fait fort bien les choses. Il a pour doux nom LM 35 et présente trois pattes de raccordement. Deux d'entre elles servent à l'alimentation, la troisième de sortie. Ce composant ressemble à un transistor au point de les confondre.

Figure 15-2 ►
Capteur de température LM35 en
boîtier plastique TO-92,
avec son brochage



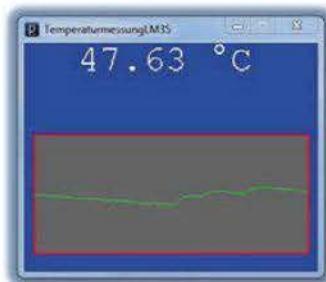
Ce capteur convertit la température mesurée en une valeur de tension analogique qui est proportionnelle à la température. Cela s'appelle un comportement de tension proportionnel à la température. Le capteur a une sensibilité de 10 mV/°C et une gamme de température comprise entre 0 et 100 °C. La formule pour calculer la température en fonction de la valeur mesurée à l'entrée analogique est la suivante :

$$\text{Température [°C]} = \frac{5.0 \cdot 100.0 \cdot \text{analogPin}}{1024.0}$$

Les valeurs de la formule se justifient ainsi :

- 5.0 : tension de référence Arduino de 5 V ;
- 100.0 : valeur maximale mesurable par le capteur de température ;
- 1024 : résolution de l'entrée analogique.

Nous allons maintenant envoyer la valeur mesurée à un sketch Processing et représenter graphiquement la courbe de température. Le tout ressemble à peu près à la figure 15-3. La température s'affiche sous la forme d'une valeur de température et sous celle d'une courbe graphique en fonction du temps.



◀ **Figure 15-3**
Courbe de température
dans Processing

Composants nécessaires



1 capteur de température LM35



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch Arduino

```
#define sensorPin 0    //Connexion à la sortie du LM35
#define DELAY 10       //Bref temps d'attente
```

```

const int cycles = 20; //Nombre de mesures

void setup(){
  Serial.begin(9600);
}
void loop(){
  float resultTemp = 0.0;
  for(int i = 0; i < cycles; i++){
    int analogValue = analogRead(sensorPin);
    float temperature = (5.0 * 100.0 * analogValue) / 1024;
    resultTemp += temperature; //Addition des valeurs mesurées
    delay(DELAY);
  }
  resultTemp /= cycles; //Calcul de la moyenne
  Serial.println (resultTemp); //Envoi à l'interface série
}

```

Revue de code Arduino

La valeur déterminée par le capteur de température LM35 est calculée avec la formule ci-après :

```
float temperature = (5.0 * 100.0 * analogValue) / 1024;
```

et moyennée à l'aide d'une boucle `for`. Les valeurs mesurées y sont additionnées, puis la moyenne est calculée. Cette dernière est enfin transmise à l'interface série :

```
Serial.println(resultTemp);
```

Son traitement par Processing commence immédiatement.

Revue de code Processing

```

import processing.serial.*;
Serial mySerialPort;
float realTemperature;
int temperature, xPos;
int[] yPos;
PFont font;

void setup(){
  size(321, 250); smooth();
  println(Serial.list());
  mySerialPort = new Serial(this, Serial.list()[0], 9600);
  mySerialPort.bufferUntil('\n');
  yPos = new int[width];
}

```

```

for(int i = 0; i < width; i++)
    yPos[i] = 250;
font = createFont("Courier New", 40, false);
textFont(font, 40); textAlign(RIGHT);
}

void draw(){
    background(0, 0, 255, 100);
    strokeWeight(2); stroke(255, 0, 0);
    fill(100, 100, 100); rect(10, 100, width - 20, 130);
    strokeWeight(1); stroke(0, 255, 0);
    int yPosPrev = 0, xPosPrev = 0;
    //Décaler les valeurs du tableau vers la gauche
    for(int x = 1; x < width; x++)
        yPos[x-1] = yPos[x];
    //Ajout des nouvelles coordonnées de la souris à
    //l'extrémité droite du tableau
    yPos[width-1] = temperature;
    //Affichage du tableau
    for(int x = 10; x < width - 10 ; x++)
        point(x, yPos[x]);
    fill(255);
    text(realTemperature + "°C", 250, 30); //Celsius
    delay(100);
}

void serialEvent (Serial mySerialPort){
    String portStream = mySerialPort.readString();
    float data = float(portStream);
    realTemperature = data;
    temperature = height - (int)map(data, 0, 100, 0, 130) - 25;
    println(realTemperature);
}

```



Pour aller plus loin

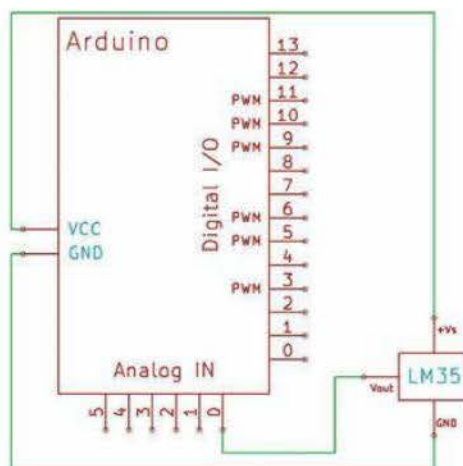
Si vous oubliez de refermer la fenêtre d'affichage de Processing que vous avez ouverte, la communication avec la carte Arduino est impossible. Pourquoi ? Tout simplement parce que Processing accède à l'interface série, dont votre carte Arduino a précisément besoin pour communiquer avec l'environnement de développement !

Ce port est donc bloqué par Processing et ne peut être libéré qu'en fermant la fenêtre d'affichage.

Schéma

Le schéma est, je dois le reconnaître, vraiment simple, mais nous allons bientôt lui ajouter quelque chose qui rendra le circuit plus fonctionnel.

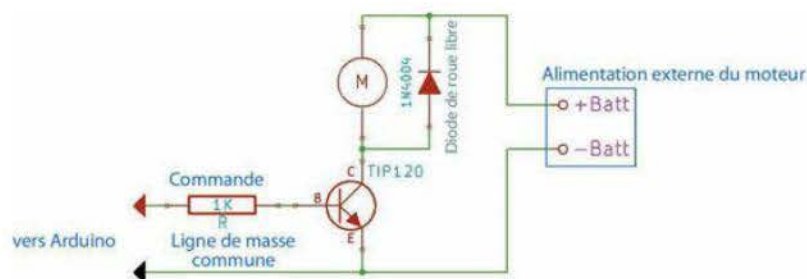
Figure 15-4 ►
Capteur de température envoyant
ses données à une entrée
analogique



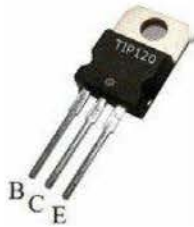
Sketch élargi (maintenant avec tout le reste)

Il est temps maintenant de construire quelque chose de bien avec le capteur de température. Que diriez-vous d'ajouter directement plusieurs composants au circuit ? Je pense qu'un ventilateur pour améliorer le climat ambiant et un afficheur pour donner les informations utiles seraient des projets intéressants. Le circuit et le sketch doivent être en mesure de mettre en route un moteur de ventilateur quand une certaine température est atteinte et de l'arrêter quand elle ne l'est plus. Nous touchons ici à l'art et la manière de commander un moteur. Ce dernier ayant assurément besoin de plus de courant et de tension pour fonctionner que la carte Arduino ne peut en fournir, il nous faut trouver autre chose. Vous avez appris, dans le chapitre 5 sur les circuits électroniques simples, comment un relais peut être commandé. Si vous remplacez le relais par un moteur, vous obtenez pratiquement une commande de moteur. Voyez la figure 15-5.

Figure 15-5 ►
Commande d'un moteur

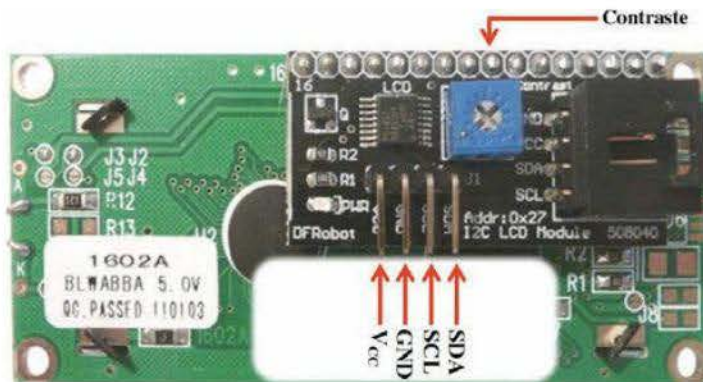


Dans ce circuit, j'ai utilisé un transistor plus fort, de type TIP 120. Il s'agit d'un transistor Darlington de puissance, en boîtier TO-220, capable de commuter un courant de collecteur $I_C = 5\text{ A}$ et de supporter une tension collecteur-émetteur $U_{CE} = 60\text{ V}$.



◀ **Figure 15-6**
Transistor Darlington

La diode de roue libre ne doit pas être oubliée, bien sûr. C'est une 1N4004. Vous souvenez-vous encore pourquoi elle est obligatoire dans ce circuit ? Reportez-vous au chapitre 4 si vous avez un trou de mémoire. Vous devez impérativement utiliser cette diode et veiller à ce que sa polarité soit correcte, faute de quoi votre carte Arduino risque fort d'en pâtir. Nous souhaitons par ailleurs utiliser un affichage LCD pour indiquer la température actuelle. Il s'agit cette fois d'un affichage qui doit être commandé via un bus I^2C (bus de données série pouvant être relié à différents types d'appareils électroniques). Il est de type I2C/TWILCD1602.



◀ **Figure 15-7**
Envers d'un affichage LCD 1602

La commande de cet affichage va être présentée au moyen du sketch utilisé. Venons-en maintenant au schéma complet, qui a l'air déjà beaucoup plus conséquent.

Composants nécessaires



1 capteur de température LM35



1 transistor de puissance TIP 120



1 résistance d'1 k Ω



2 résistances de 10 k Ω



1 diode 1N4004



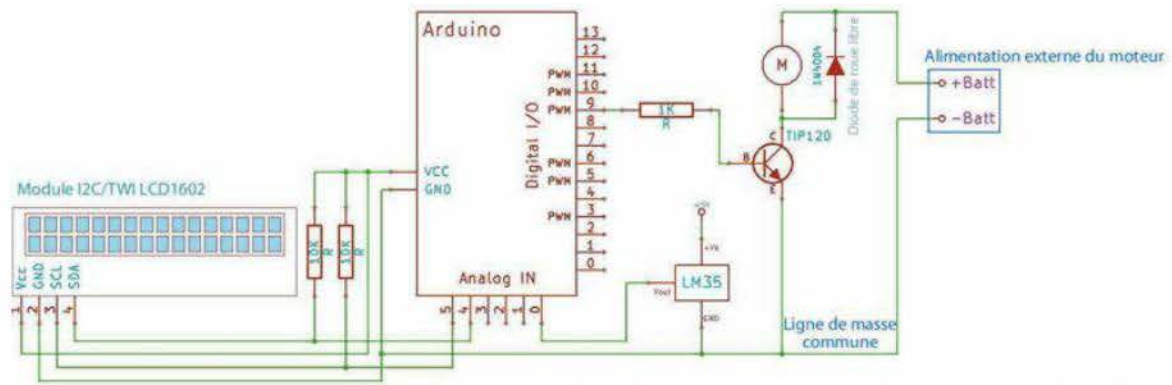
Module afficheur LCD I2C/TWI LCD1602



Moteur de ventilateur, 12 V par exemple



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses



Bon ! Nous avons à gauche le LCD I2C avec les résistances pull-up, au centre notre Arduino et à droite le capteur de température LM35. Complètement à droite se trouve la commande du moteur avec le transistor TIP 120 et la diode de roue libre 1N4004. Voyons maintenant le code du sketch :

▲ **Figure 15-8**

Circuit complet avec capteur, affichage et moteur ou plutôt ventilateur

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#define sensorPin 0 //Connexion à la sortie du LM35
#define DELAY1 10 //Bref temps d'attente lors de la mesure
#define DELAY2 500 //Bref temps d'attente lors de l'affichage
#define motorPin 9 //Broche commande du ventilateur
#define threshold 25 //Température de commutation du ventilateur
// (25 degrés Celsius)
#define hysteresis 0.5 //Valeur d'hystérésis (0.5 degré Celsius)
const int cycles = 20; //Nombre de mesures
LiquidCrystal_I2C lcd(0x27, 16, 2); //Adresse I2C : 0x27 pour
//16 caractères/2 lignes

void setup(){
  pinMode(motorPin, OUTPUT);
  lcd.init(); //Initialisation du LCD
  lcd.backlight(); //Activation du rétroéclairage
}

void loop(){
  float resultTemp = 0.0;
  for(int i = 0; i < cycles; i++){
    int analogValue = analogRead(sensorPin);
    float temperature = (5.0 * 100.0 * analogValue) / 1024;
    resultTemp += temperature; //Addition des valeurs mesurées
    delay(DELAY1);
  }
  resultTemp /= cycles; //Calcul de la moyenne
  lcd.clear(); //Méthode clear pour effacer le contenu du LCD
```



```

lcd.print("Temp:"); //Méthode print pour écrire sur le LCD
lcd.print(resultTemp);
#if ARDUINO < 100
lcd.print(0xD0 + 15, BYTE); //Caractère degré (Arduino 0022)
#else
lcd.write(0xD0 + 15); //Caractère degré (Arduino 1.00)
#endif
lcd.print("C");
lcd.setCursor(0, 1); //Méthode setCursor pour positionner
                        //le curseur du LCD

lcd.print("Moteur:");
if(resultTemp > (threshold + hysteresis))
    digitalWrite(motorPin, HIGH);
else if(resultTemp < (threshold - hysteresis))
    digitalWrite(motorPin, LOW);
lcd.print(digitalRead(motorPin) == HIGH?"en marche":"stop");
delay(DELAY2);
}

```

La détermination de la température est effectuée de la même manière et se trouve être la même que dans l'exemple précédent.



Soit vous jouez encore les cachottiers, soit vous avez carrément oublié : dans ce code de sketch se trouve également un élément de programme dont vous ne nous avez pas parlé. Que signifie la ligne `const int cycles = 20;` ? Ce qui me chagrine là-dedans, c'est le mot `const`.

Bien vu, Ardu ! Cette fois, j'allais vraiment oublier ! Je dois ici développer un peu, mais vous allez voir que c'est vraiment facile à comprendre. Nous avons affaire à une autre forme de déclaration de variable. Vous connaissez par conséquent maintenant trois formes écrites, que je vous rappelle au moyen d'un exemple :

1. `int grandeurs = 47;`
2. `#define grandeurs 47`
3. `const int grandeurs = 47;`

Les trois variantes initialisent visiblement une variable appelée `grandeurs` avec la valeur 47. Alors où est la différence ? Il doit bien y en avoir une, sinon à quoi bon des formes écrites différentes.

Variante 1

Bon ! La première variante `int grandeurs = 47;` fait réserver par le compilateur un emplacement dans la mémoire vive RAM pour y

consigner la valeur 47. De la mémoire supplémentaire est donc nécessaire et occupée.

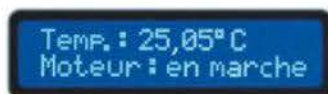
Variante 2

Cette variante utilise la directive de prétraitement `#define`, qui affecte une valeur uniquement à un nom, que le compilateur remplace partout dans le code de sketch lors de la transformation. Aucune mémoire supplémentaire n'est ainsi attribuée pour gérer une variable. Mais vous devez vous demander, pour cette forme écrite, quel type de donnée est employé, car il n'est pas indiqué comme dans le premier exemple. Duquel pourrait-il bien s'agir ?

Variante 3

Si le mot-clé `const` est utilisé devant la déclaration de variable, alors la variable en question n'est plus une variable mais une constante, dont la valeur ne peut plus être modifiée pour la durée du sketch. Il s'agit presque d'une variable en *lecture seule*. Qu'en pensez-vous maintenant si je vous dis que cette variante n'utilise pas de mémoire ? On est sûr que la variable ne sera pas modifiée, aussi pourquoi en occuperait-elle ? Mais en quoi est-elle différente de la variante `#define` ? C'est très simple : on peut indiquer ici un certain type de donnée.

Sur Internet, tout comme dans de nombreux livres, on passe sans cesse de l'une à l'autre des trois possibilités. De quelle variante faut-il se servir ? Si la mémoire est juste et si une indication explicite du type de donnée est nécessaire, la variante 3 est alors recommandée. Revenons maintenant à notre circuit. Le mieux est de vous montrer l'afficheur LCD.



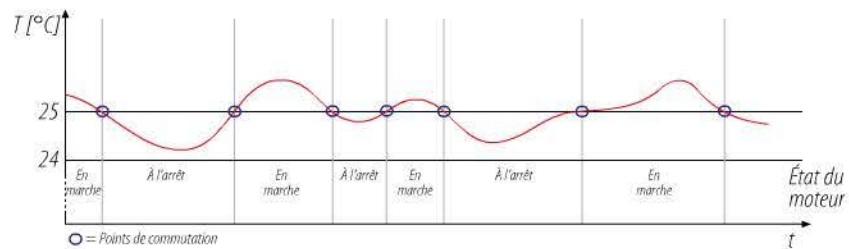
Vous pouvez y lire sans problème la température et l'état du moteur.



Holà, stop, arrêtez ! J'ai bien compris jusqu'ici le fonctionnement du sketch, mais je n'ai aucune idée de ce qu'est une hystérésis.

Vous ne pouvez rien me reprocher cette fois, car j'allais y venir. Imaginez la situation suivante : le ventilateur doit, tout comme dans notre exemple, se mettre en marche à 25 °C et apporter un peu d'air frais à ceux qui transpirent sur leur Arduino. La température ambiante n'étant cependant pas constante à 100 %, le capteur est lui aussi soumis à certaines fluctuations. Un état est donc par exemple atteint, dans lequel la température mesurée varie constamment entre 24,8 et 25,2 °C. Autrement dit, le ventilateur n'arrête pas de s'allumer et de s'éteindre. Plutôt énervant à la longue ! Voyons maintenant la figure 15-9 de plus près.

Figure 15-9 ►
En cas de température fluctuant autour de la valeur de consigne, l'état du moteur change constamment.



C'est là que l'*hystérésis* (le mot vient du grec et signifie retard) entre en jeu. On peut expliquer ainsi le comportement d'une régulation avec hystérésis : la variable de sortie, qui commande ici le moteur, ne dépend pas seulement de la variable d'entrée délivrée par le capteur. L'état de la variable de sortie, qui régnait auparavant, joue aussi un rôle important. Dans notre exemple, nous avons une valeur-seuil de 25 °C et une hystérésis de 0,5 °C. Voyons maintenant de plus près la régulation du ventilateur :

```
if(resultTemp > (threshold + hysteresis))
    digitalWrite(motorPin, HIGH);
else if(resultTemp < (threshold - hysteresis))
    digitalWrite(motorPin, LOW);
```

Quand le ventilateur se met-il en marche ?

Si la condition :

```
(resultTemp > (threshold + hysteresis)) ...
```


est remplie, le ventilateur commence à tourner. C'est le cas ici quand la température mesurée est supérieure à $25 + 0,5\text{ }^{\circ}\text{C}$.

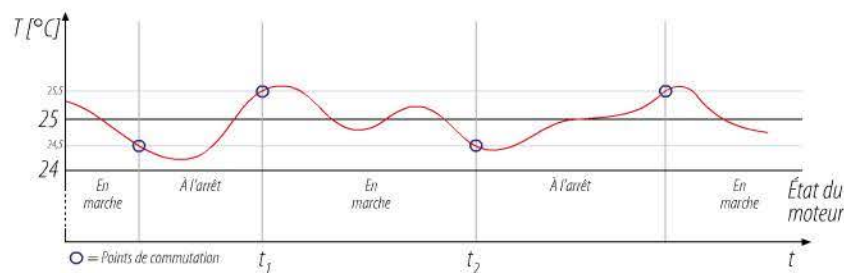
Quand le ventilateur s'arrête-t-il ?

Si la condition :

$\text{resultTemp} < (\text{threshold} - \text{hysteresis})$

est remplie, le ventilateur s'arrête de tourner, ici en l'occurrence quand la température est inférieure à $25 - 0,5\text{ }^{\circ}\text{C}$. Pour résumer :

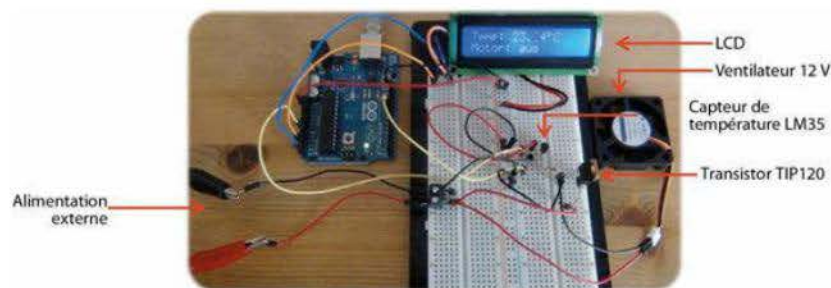
- le ventilateur est en marche si : température $> 25,5\text{ }^{\circ}\text{C}$;
- le ventilateur est à l'arrêt si : température $< 24,5\text{ }^{\circ}\text{C}$.



◀ **Figure 15-10**

En cas de fluctuation de la température autour de la valeur de consigne, l'état du moteur ne change pas constamment.

Si vous regardez la courbe entre les points t_1 et t_2 , vous verrez que la température passe constamment au-dessus et en dessous des $25\text{ }^{\circ}\text{C}$. Sans commande avec hystérésis, nous aurions sans cesse *moteur en marche* et *moteur à l'arrêt*. La réalisation complète du circuit est donc celle de la figure 15-11.



◀ **Figure 15-11**

Réalisation complète du circuit



Attention !

Vous devez être particulièrement soigneux du fait que vous travaillez avec une source de tension externe. Comme je l'ai déjà expliqué, vous devez raccorder ensemble les deux points de masse de la carte Arduino et de la source de tension externe. Mais surtout pas les potentiels positifs ! Vous ne devez en aucun cas confondre ces deux potentiels et devez veiller à ce qu'aucun court-circuit ne se produise. Vérifiez le câblage du circuit avant de tout mettre en marche. Mieux vaut trop que pas assez...

Problèmes courants

Si le ventilateur ne se met pas en marche alors que la température-seuil plus la valeur d'hystérésis est atteinte, éteignez tout et vérifiez ce qui suit.

- Le câblage est-il correct ?
- Pas de court-circuit éventuel ?
- La masse commune à la carte Arduino et à la source de tension externe est-elle établie ?
- La diode de roue libre est-elle montée dans le bon sens ?
- Si on ne voit rien sur le LCD, le contraste n'est-il pas trop faible ?

Qu'avez-vous appris ?

- Dans ce montage, vous avez appris comment le capteur de température fonctionne et convertit des valeurs de température en valeurs de tension correspondantes, qui peuvent être exploitées à l'entrée analogique de votre carte Arduino.
- Vous avez utilisé un affichage I2C/TWI LCD1602, commandé via le bus I^2C , pour indiquer la valeur de température.
- Pour que le ventilateur fonctionne correctement, vous avez dû recourir à une alimentation externe, elle-même connectée au transistor de puissance TIP 120.
- Vous avez appris comment une diode 1N4004 sert, en tant que diode de roue libre, à protéger votre carte Arduino.

Exercice complémentaire

Agrandissez votre circuit de manière à pouvoir augmenter ou diminuer la valeur-seuil au moyen par exemple de deux boutons-poussoirs supplémentaires. Le LCD est censé attirer l'attention en se mettant à clignoter une fois cette valeur-seuil atteinte. Pour plus d'informations sur la bibliothèque et sur la gamme d'instructions du LCD, recherchez les mots suivants sur Google :

- I2C/TWI LCD 1602 ;
- Dfrobot.



Pour aller plus loin

Il existe bien entendu beaucoup d'autres capteurs de température. En voici une sélection :

- TMP75 (avec bus I²C) ;
- AD22100 (capteur de température analogique) ;
- DHT11 (capteur de température et humidité avec microcontrôleur 8 bits intégré) ;
- DS18B20 (capteur de température numérique 1-Wire).

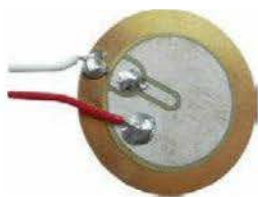
Le son et plus encore

Au sommaire :

- l'émission de son au moyen d'un élément piézoélectrique ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- la création du jeu de la séquence des couleurs ;
- un exercice complémentaire.

Y a pas le son ?

À la longue, vous en avez peut-être assez des signaux lumineux et autres LED clignotantes. Aussi allons-nous voir à présent comment votre carte Arduino peut émettre des sons au moyen d'un élément piézoélectrique. Ce composant vous a déjà été présenté dans le chapitre 4 sur les bases de l'électronique.



◀ **Figure 16-1**
Disque piézo

Vous ne risquez pas les ondes de choc acoustiques avec un piézo, car les vibrations émises couvrent un espace des plus réduits. Il est cependant parfait pour ce que nous allons faire.

Si on branche, par exemple, l'élément sur une sortie numérique et si on passe à intervalles réguliers la sortie en niveau HIGH ou LOW, on entend un craquement dans l'élément piézoélectrique. Plus le laps de

temps entre niveaux HIGH et LOW est court, plus le son audible est aigu ; plus le laps de temps est long, plus le son est grave. Le phénomène est le même quand, par exemple, vous passez vos doigts plus ou moins vite sur une grille à lamelles. Plus vous allez vite, plus le bruit est aigu ; le piézo fonctionne sur ce principe. Un craquement répété, tantôt plus lent, tantôt plus rapide, influe sur la fréquence du son. Voici un sketch très simple pour émettre un son.

```
#define piezoPin 13 //Élément piézoélectrique sur broche 13
#define DELAY 1000

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

void loop(){
  digitalWrite(piezoPin, HIGH); delayMicroseconds(DELAY);
  digitalWrite(piezoPin, LOW); delayMicroseconds(DELAY);
}
```

Ne vous inquiétez pas pour la fonction `delayMicroseconds`. Son action est la même que celle de la fonction `delay`, à ceci près que la valeur transmise n'est pas interprétée en millisecondes mais en microsecondes ; la microseconde est 1 000 fois plus petite (1 ms = 1 000 μ s). Cette nouvelle fonction est utilisée, car `delay` ne permet de descendre en dessous d'1 ms.

Composants nécessaires



1 élément piézoélectrique



Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Pour le premier sketch utile qui doit être capable d'émettre plusieurs sons à des fréquences différentes, mieux vaut créer un tableau des sons avec différentes valeurs que nous appellerons l'une après l'autre pendant le sketch. On utilise pour ce faire la fonction `tone` (sons) mise à disposition par Arduino. Vous en saurez bientôt plus.

```
#define piezoPin 13          //Élément piézoélectrique sur broche 13
#define toneDuration 500    //Durée du son
#define tonePause 800       //Longueur de la pause entre les sons
int tones[] = {523, 659, 587, 698, 659, 784, 698, 880};
int elements = sizeof(tones) / sizeof(tones[0]);

void setup(){
    noTone(piezoPin); //Rendre le piézo muet
    for(int i = 0; i < elements; i++){
        tone(piezoPin, tones[i], toneDuration); //Exécuter le son
        delay(tonePause); //Pause entre les sons
    }
}

void loop(){/* vide */}
```

Revue de code

Du point de vue logiciel, les variables suivantes sont nécessaires à notre expérimentation.

Variable	Objet
tones[]	Tableau contenant les fréquences des différents sons à exécuter
elements	Nombre d'éléments du tableau

◀ **Tableau 16-1**

Variables nécessaires et leur objet

Le tableau unidimensionnel `tones` est du type de donnée `int` et contient les fréquences en hertz des sons à exécuter. Les hertz (Hz) servent à mesurer le nombre de vibrations par seconde. Plus la valeur est élevée, plus le son est aigu, et vice versa. Le nombre d'éléments du tableau est affecté à la variable `elements` ; il servira plus tard dans la boucle `for` pour traiter tous les éléments. Le réglage manuel de la limite supérieure ou de la condition de la boucle `for` est ainsi évité, celui-ci étant fait automatiquement au moyen d'un calcul.

Oh la la ! J'ai du mal avec le calcul des éléments du tableau. Pouvez-vous m'expliquer s'il vous plaît ?

J'allais y venir. On utilise pour ce faire la fonction `sizeof` de C++, qui détermine la taille d'une variable ou d'un objet dans la mémoire. Voici un court exemple :

```
byte byteValue = 16;    //Variable du type byte
int intValue = 4;        //Variable du type int
long longValue = 3.14;  //Variable du type long
int myArray[] = {25, 46, 9}; //Tableau du type int
```



```

void setup(){
  Serial.begin(9600);
  Serial.print("Nombre d'octets pour 'byte' : ");
  Serial.println(sizeof(byteValue));
  Serial.print("Nombre d'octets pour 'int' : ");
  Serial.println(sizeof (intValue));
  Serial.print("Nombre d'octets pour 'long' : ");
  Serial.println(sizeof(longValue));
  Serial.print("Nombre d'octets pour 'myArray' : ");
  Serial.println(sizeof(myArray));
}

void loop(){/ * vide */}

```

L'affichage est donc le suivant :

```

Nombre d'octets pour 'byte' : 1
Nombre d'octets pour 'int' : 2
Nombre d'octets pour 'long' : 4
Nombre d'octets pour 'myArray' : 6

```

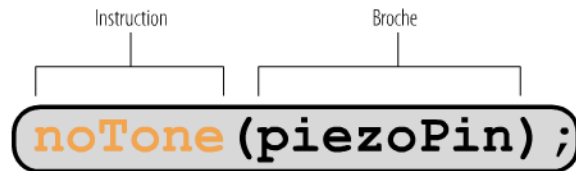
Quand on regarde les valeurs pour les types de données `byte`, `int` et `long`, on s'aperçoit qu'elles sont identiques à celles indiquées dans le chapitre 9, dans lequel il était question de types de données et de domaines de valeurs.

Passons à la dernière ligne de l'affichage. On y voit que le tableau occupe 6 octets de mémoire, ce qui est logique puisqu'un seul élément `int` nécessite 2 octets de mémoire. Or, nous avons 3 éléments. Le résultat est donc $2 \times 3 = 6$ octets. La ligne :

```
int elements = sizeof(tones) / sizeof(tones[0]);
```

divise le nombre d'octets du tableau par le nombre d'octets d'un seul élément. C'est toujours de cette manière qu'on obtient le nombre d'éléments d'un tableau. Mais revenons à notre sketch. Tout au début, la fonction `noTone` rend le piézo muet au cas où il devrait encore pépier du fait d'un sketch précédent. Elle n'a qu'un seul paramètre, qui indique la broche sur laquelle se trouve le piézo.

Figure 16-2 ►
La fonction `noTone`
rend le piézo muet.



La fonction `tone` possède en revanche deux autres paramètres. L'un indique la fréquence et l'autre la durée pendant laquelle le son doit être audible.

Instruction	Broche	Fréquence	Durée
<code>tone(piezoPin, 400, 700);</code>			

◀ **Figure 16-3**
La fonction `tone` rend le piézo bavard.

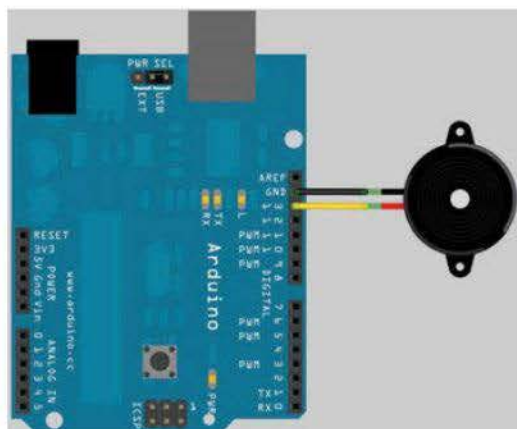
Pouvez-vous me dire comment vous en êtes venu aux différentes valeurs que vous utilisez dans le tableau des sons ? Vous les avez toutes essayées pour savoir lesquelles conviennent à peu près ?

Non, elles sont tirées d'un exemple de sketch qui se trouve dans l'IDE Arduino. Recherchez le fichier `pitches.h` dans le dossier `examples` de l'installation Arduino et ouvrez-le avec un éditeur. Vous y trouverez les fréquences correspondant à de nombreuses notes. Vous pouvez inclure ce fichier dans votre sketch et utiliser ensuite directement les constantes symboliques. Essayez ! Le code est alors beaucoup plus parlant et plus clair que lorsque des valeurs numériques sont utilisées.



Réalisation du circuit

Le circuit n'a vraiment rien d'extraordinaire, pas vrai ?



◀ **Figure 16-4**
L'élément piézoélectrique connecté

Sketch élargi : jeu de la séquence des couleurs

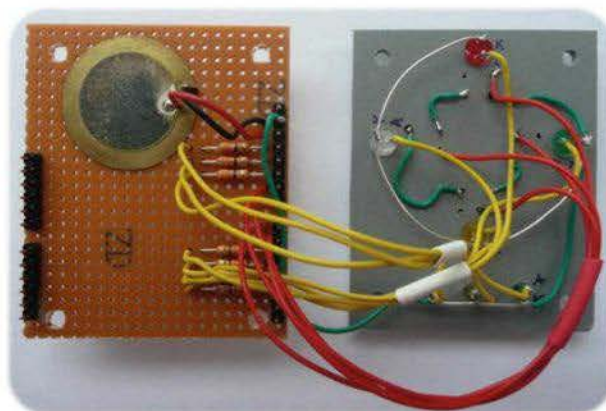
Venons-en maintenant à un jeu intéressant dit de la séquence des couleurs. Nous avons quatre LED de couleurs différentes disposées en carré. Près de chacune se trouve un bouton-poussoir. Le microcontrôleur imagine un motif pour l'ordre d'allumage des LED ; à vous de le reproduire correctement. Au début, la séquence ne comporte qu'une seule LED allumée ; une nouvelle vient s'ajouter après chaque bonne réponse. L'allumage de chacune des quatre LED est accompagné d'un son qui lui est propre. Le jeu ravit donc non seulement les yeux, mais aussi les oreilles. J'ai ajouté au circuit un shield avec une face avant de ma fabrication. Voyez plutôt (voir figure 16-5).

Figure 16-5 ►
Shield avec la face avant pour le jeu
de la séquence des couleurs



Sur la face avant, on voit les quatre grosses LED de 5 mm avec leur bouton-poussoir respectif. Quand une LED s'allume, on doit appuyer sur le bouton-poussoir placé à côté. La partie basse de la face avant est occupée par trois plus petites LED de 3 mm. Elles servent à afficher l'état, sur lequel je reviendrai plus tard. La figure 16-6 montre bien le shield et la face avant avec le câblage.

Figure 16-6 ►
Shield ouvert et la face avant
retournée



Les choses sont moins graves qu'elles n'y paraissent, et la construction devient claire quand on regarde le schéma. Voici les points que je poserai comme conditions pour le jeu :

- une longueur déterminée de la séquence, d'abord constante, est fixée par le sketch ;
- chacune des quatre LED doit avoir sa propre note avec une fréquence particulière ;
- quand une des quatre LED s'allume, la note correspondante est émise ;
- quand le bouton-poussoir situé à côté est appuyé, la LED s'allume et la note correspondante est émise ;
- si la séquence a été correctement reproduite, la LED d'état verte s'allume et une suite de sons *crescendo* se fait entendre. Le jeu reprend ensuite au début avec une nouvelle séquence ;
- si la séquence reproduite est fausse à un endroit quelconque, la LED d'état rouge s'allume et une suite de sons *decrescendo* se fait entendre. Le jeu redémarre ensuite avec une nouvelle séquence.

Composants nécessaires



4 LED
(si possible de couleurs différentes)



7 résistances de 330 Ω



3 LED 3 mm (rouge, verte et jaune)



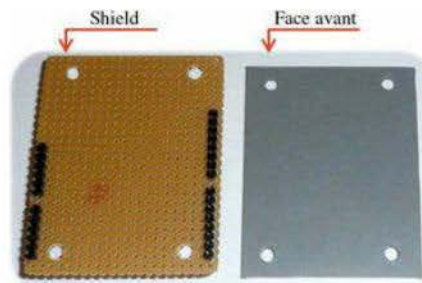
4 boutons-poussoir



4 entretoises DK 15 mm, en plastique



4 vis M3 30 mm coupées à 23 mm
environ + 4 écrous



1 shield + 1 face avant



Fils de différentes couleurs



2 barrettes à 6 broches + 2 barrettes à
8 broches

Voyons maintenant le schéma.

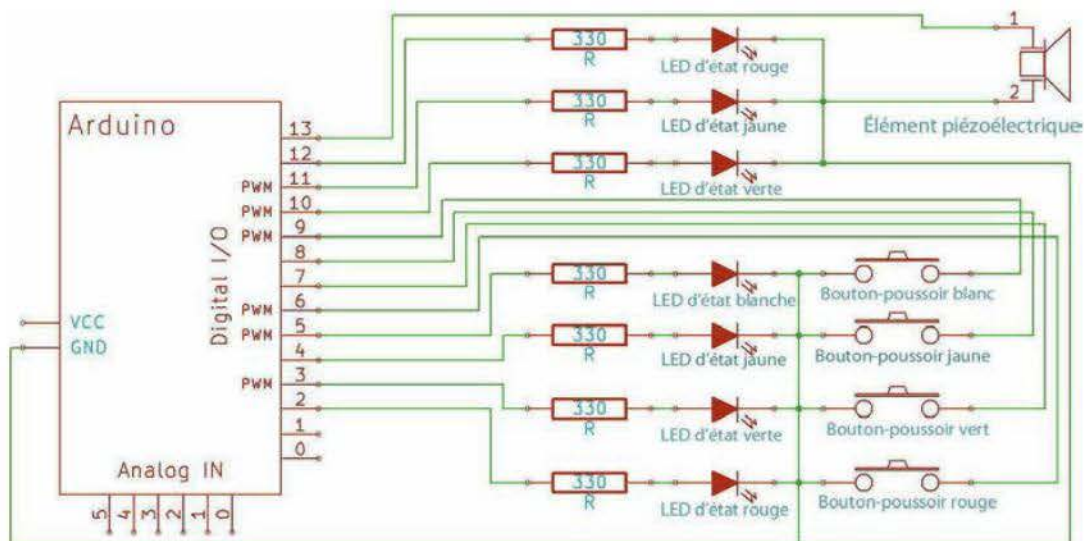


Figure 16-7▲ Et voici maintenant le code du sketch quelque peu élargi.
Circuit complet du jeu de la
séquence des couleurs

```

#define MAXARRAY 5           //Définir la longueur de la séquence
int ledPin[] = {2, 3, 4, 5}; //Tableau de LED avec numéros de broche
#define piezoPin             13 //Broche piézo
#define buttonPinRed         6 //Broche bouton-poussoir LED rouge
#define buttonPinGreen       7 //Broche bouton-poussoir LED verte
#define buttonPinYellow      8 //Broche bouton-poussoir LED jaune
#define buttonPinWhite       9 //Broche bouton-poussoir LED blanche
#define ledStatePinGreen     10 //LED d'état verte
#define ledStatePinYellow    11 //LED d'état jaune
#define ledStatePinRed       12 //LED d'état rouge
int colorArray[MAXARRAY];    //Contient la suite de chiffres
                              //pour les couleurs à afficher
int tones[] = {1047, 1175, 1319, 1397}; //Fréquences des sons
                              //pour les 4 couleurs
int counter = 0;             //Nombres de LED actuellement
                              //allumées

boolean fail = false;

void setup(){
  Serial.begin(9600);
  for(int i = 0; i < 4; i++)
    pinMode(ledPin[i], OUTPUT); //Programmation des broches
                                //de LED comme sortie

  pinMode(buttonPinRed, INPUT);digitalWrite(buttonPinRed, HIGH);
  pinMode(buttonPinGreen, INPUT);digitalWrite(buttonPinGreen, HIGH);
  pinMode(buttonPinYellow, INPUT);digitalWrite(buttonPinYellow, HIGH);
  pinMode(buttonPinWhite, INPUT);digitalWrite(buttonPinWhite, HIGH);
  pinMode(ledStatePinGreen, OUTPUT);
  pinMode(ledStatePinYellow, OUTPUT);
  pinMode(ledStatePinRed, OUTPUT);
}

void loop(){
  Serial.println("Départ du jeu");
  generateColors();
  int buttonCode;
  for(int i = 0; i <= counter; i++){ //Boucle extérieure
    giveSignalSequence(i);
    for(int k = 0; k <= i; k++){ //Boucle intérieure
      while(digitalRead(buttonPinRed) && digitalRead(buttonPinGreen)
&& digitalRead(buttonPinYellow) && digitalRead(buttonPinWhite));
      Serial.println ("Bouton poussé !"); //Pour contrôle dans
                                          //Serial Monitor

      //Affichage de la couleur appuyée
      if(!digitalRead(buttonPinRed))
        buttonCode = 0;
      if(!digitalRead(buttonPinGreen))
        buttonCode = 1;
      if(!digitalRead(buttonPinYellow))
        buttonCode = 2;
    }
  }
}

```



```

        if(!digitalRead(buttonPinWhite))
            buttonCode = 3;
        giveSignal(buttonCode);
        //Vérifier si la bonne couleur a été appuyée
        if(colorArray[k] != buttonCode){
            fail = true;
            break; //Quitter la boucle for interne
        }
    }

    if(!fail)
        Serial.println("correct"); //Pour contrôle dans Serial
                                    //Monitor
    else{
        digitalWrite(ledStatePinRed, HIGH);
        for(int i = 3000; i > 500; i-=150){
            tone(piezoPin, i, 10); delay(20);
        }
        Serial.println("faux"); //Pour contrôle dans Serial
                                //Monitor

        delay(2000);
        digitalWrite(ledStatePinRed, LOW);
        counter = 0; fail = false;
        break; //Quitter la boucle for
    }
    delay(2000);

    if(counter + 1 == MAXARRAY){
        digitalWrite(ledStatePinGreen, HIGH);
        for(int i = 500; i < 3000; i+=150){
            tone(piezoPin, i, 10); delay(20);
        }
        Serial.println("Fini!"); //Pour contrôle dans Serial
                                //Monitor

        delay(2000);
        digitalWrite(ledStatePinGreen, LOW);
        counter = 0; fail = false;
        break; //Quitter la boucle for extérieure
    }
    counter++; //Incrémenter le compteur
}

}

void giveSignalSequence(int value){
    //Affichage LEDs
    for(int i = 0; i <= value; i++){
        digitalWrite(2 + colorArray[i], HIGH);
        generateTone(colorArray[i]); delay(1000);
        digitalWrite(2 + colorArray[i], LOW); delay(1000);
    }
}

```

```

    }
}

void generateTone(int value){
    tone(piezoPin, tones[value], 1000);
}

void giveSignal(int value){
    //Affichage LED + Tonsignal
    digitalWrite(2 + value, HIGH); generateTone(value); delay(200);
    digitalWrite(2 + value, LOW); delay(200);
}

void generateColors(){
    randomSeed(analogRead(0));
    for(int i = 0; i < MAXARRAY; i++){
        colorArray[i] = random(4); //Générer des chiffres aléatoires
                                   //de 0 à 3
        //0 = rouge, 1 = vert, 2 = jaune, 3 = blanc
    }
    for(int i = 0; i < MAXARRAY; i++){
        Serial.println(colorArray[i]); //Pour contrôle dans Serial
                                       //Monitor
    }
}

```

Comment la programmation fonctionne-t-elle en détail ? Le sketch semble fastidieux au premier abord. Ne le considérez pas dans son intégralité, mais prenez le temps de décomposer le programme en sous-ensembles et de procéder étape par étape. Une valeur numérique est affectée à chaque couleur à afficher : 0 pour rouge, 1 pour vert, 2 pour jaune et 3 pour blanc. Un tableau peut ainsi être initialisé avec des valeurs allant de 0 à 3 ; il pourra ensuite servir à afficher les LED.

Supposons que vous ayez un tableau avec les valeurs 0, 2, 2, 1 et 3, les diodes s'allument donc dans l'ordre suivant : rouge, jaune, jaune, vert et blanc. Dans notre sketch, son nom est `colorArray` et il reçoit ses valeurs via la fonction `generateColors`. Pour les rendre visibles, la fonction `giveSignal` convertit les valeurs en signaux pour commander les LED.

```

void giveSignalSequence(int value){
    //Affichage LEDs
    for(int i = 0; i <= value; i++){
        digitalWrite(2 + colorArray[i], HIGH);
        generateTone(colorArray[i]; delay(1000);
        digitalWrite(2 + colorArray[i], LOW); delay(1000);
    }
}

```



Si la fonction doit toujours afficher la séquence des couleurs, pourquoi avons-nous encore besoin d'une variable ? Et que signifie le 2 qui est utilisé dans la fonction `digitalWrite` ? C'était quoi déjà le truc avec les magic numbers ?

Eh bien Ardu, la séquence complète ne doit pas s'afficher au début mais seulement au fur et à mesure avec, chaque fois, une couleur en plus. Le tableau des couleurs `colorArray` contient bien la séquence complète, mais la variable transmise dans `value` indique à la fonction combien d'éléments du tableau doivent être interrogés et affichés. Les quatre grandes LED étant cependant raccordées aux sorties numériques des broches 2 à 5, le chiffre 2 est quasiment un décalage qui indique la broche de démarrage quand les valeurs 0 à 3 sont ajoutées au tableau des couleurs. Vous avez bien entendu raison quand vous dites qu'il ne faut pas utiliser de magic numbers. Vous pouvez naturellement employer une constante symbolique, par exemple avec le nom `FARBPINOFFSET`.



Avant de passer à l'explication de la logique dans la fonction `loop`, je souhaiterais qu'on revienne sur la fonction `setup`. Il y a, par exemple, des broches de bouton-poussoir qui sont, bien sûr, programmées comme entrées. Pourtant, quelque chose est envoyée à ces mêmes entrées par la fonction `digitalWrite`. Pourquoi cela ?

J'utilise la possibilité d'activer les résistances pull-up présentes et connectées en interne dans le microcontrôleur. Plus besoin ainsi de connecter des résistances pull-up ou pull-down externes. J'ai déjà expliqué cela dans le montage n° 2. Si vous avez oublié, relisez-le !



Oui, je vais le faire. Quand je vois la fonction `loop`, je me dis qu'il s'en passe de belles ! Mais ce que je ne comprends pas trop, c'est le fait que la fonction `loop` s'exécute continuellement. La première boucle `for`, que vous avez qualifiée de boucle extérieure, devrait elle aussi s'exécuter continuellement. C'est pourtant elle qui – d'après ce que comprends – est chargée d'afficher la séquence en fonction de la variable `counter`.

Oui Ardu, bien vu ! La fonction `loop`, qui est une boucle sans fin, devrait normalement s'exécuter en permanence. Seulement, j'ai incorporé un arrêt qui la bloque tant qu'un des quatre boutons-poussoirs n'est pas appuyé. Voici la partie de code en question :

```
while(digitalRead(buttonPinRed) && digitalRead(buttonPinGreen) &&  
      digitalRead(buttonPinYellow) && digitalRead(buttonPinWhite));
```

Les entrées numériques, auxquelles les boutons-poussoirs sont raccordés, étant reliées au +5 V à travers les résistances pull-up internes, mon interrogation doit porter sur le niveau LOW. Tant que toutes les entrées sont sur niveau HIGH, la boucle `while` exécute l'instruction qui vient aussitôt après.

C'est là tout mon problème ! Quelle instruction est exécutée au juste ?
D'après le code, la ligne suivante :
`Serial.println("Bouton poussé !");`
devrait être exécutée. Mais ça n'a pas beaucoup de sens !



Vous avez raison, ça n'a pas beaucoup de sens ! Vous avez cependant oublié une petite chose. L'instruction, qui vient immédiatement après la boucle `while`, est le point-virgule situé tout à la fin. Il s'agit quasiment d'une instruction vide, qui fait en sorte que la boucle `while`, quand aucun des boutons-poussoirs n'est appuyé, devienne elle-même une boucle sans fin. C'est une manière élégante d'interrompre ici le déroulement du programme. Ce n'est que quand l'un ou l'autre des boutons est appuyé que la condition dans la boucle `while` n'est plus remplie et que le programme reprend son cours. Le bouton appuyé est alors identifié afin de comparer la valeur de la couleur concernée à l'élément du tableau qui vient d'être sélectionné dans la boucle intérieure. Si une concordance a été trouvée, on passe à la valeur de couleur suivante dans la séquence. En revanche, si une erreur a été commise, la variable `fail` reçoit la valeur `true`, et l'instruction `break` fait sortir prématurément de la boucle intérieure. Autrement dit, l'instruction `if` :

```
if(!fail)...
```

reprend le cours du programme en conséquence. La variable `counter` est augmentée de la valeur 1, dans la mesure où aucune erreur n'a été commise et où la fin de la séquence n'est pas encore atteinte, si bien que la prochaine séquence affichée sera plus longue. J'ai laissé les impressions sur le Serial Monitor dans le code pour une meilleure compréhension des procédés. Elles vous donnent au début la séquence qui a été sélectionnée pour que vous puissiez faire, le cas échéant, un peu d'expérimentation. Toute explication supplémentaire est ici superflue. Lisez une fois le code de bout en bout et essayez de le comprendre.

Communication réseau

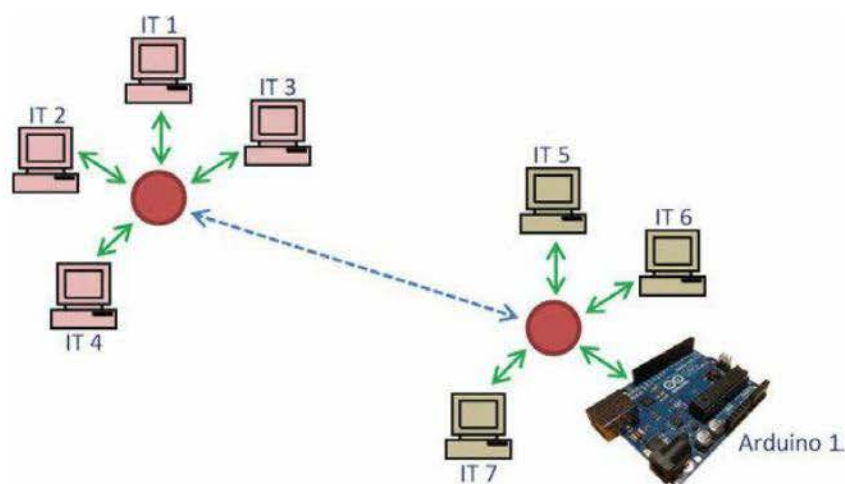
Au sommaire :

- savoir ce qu'est un réseau ;
- apprendre à incorporer la carte Arduino dans un réseau ;
- savoir ce qu'est un serveur web ;
- un exercice complémentaire.

Qu'est-ce qu'un réseau ?

Le plus gros réseau que l'Homme utilise quotidiennement est le *World Wide Web* ou *www* sous sa forme abrégée. Il s'agit de l'interconnexion d'une multitude de systèmes informatiques en contact l'un avec l'autre dans le monde entier. On parle de réseau dès l'instant où deux ordinateurs sont associés via un support de transmission approprié (par exemple : câble Ethernet, fibre optique ou Wlan). Vous pouvez l'imaginer comme un cerveau contenant plusieurs centaines de milliards de cellules nerveuses. Chacune d'elles possède jusqu'à dix mille synapses. Ce sont des voies de communication qu'elles utilisent pour transmettre ou échanger des informations. Chaque cellule nerveuse du cerveau figure un ordinateur en contact avec d'autres systèmes au moyen des synapses – donc de sa carte réseau (ou de ses cartes le cas échéant).

Figure 17-1 ►
Petit réseau avec carte Arduino



Les différents systèmes informatiques, que j'ai appelé IT1 à IT7 dans la figure 17-1 pour plus de commodité, sont reliés entre eux au moyen des cartes ou plutôt des câbles de réseau. Cette représentation est bien sûr simplifiée car les composants du réseau sont par exemple reliés par des *switches* dans la réalité. Ces répartiteurs ou coupleurs de réseau transmettent les données de manière intelligente aux différents utilisateurs. La figure 17-2 montre un connecteur de type RJ45 d'un câble de réseau couramment utilisé.

Figure 17-2 ►
Connecteur RJ45 d'un câble de réseau



Je pense que vous avez déjà vu une fiche de cette sorte puisque votre ordinateur est relié à coup sûr par un câble de réseau au routeur qui établit une liaison vers votre fournisseur d'accès, autrement dit vers Internet.



Je ne vois pas de prise femelle pour cette fiche sur ma carte Arduino. Comment fait-on pour la connecter au réseau ?

Vous allez ici encore plus vite que la musique, Arduus. Mon introduction n'est même pas terminée. La carte Arduino ne dispose évidemment pas d'une connexion réseau. Un composant réseau supplémentaire est donc nécessaire.



Shield Ethernet



Module Ethernet ENC28J60

◀ **Figure 17-3**
Deux composants Ethernet

La figure 17-3 montre à gauche un shield Ethernet, qui dispose en plus d'un socle microSD. Vous pouvez y stocker temporairement des données, mais là n'est pas le sujet. À droite se trouve un module Ethernet ENC28J60. Il est certes préférable au shield Ethernet, mais ne permet pas de stocker des données sur une carte SD et ne peut être branché directement sur la carte Arduino. Des cordons de raccordement doivent être utilisés pour relier le module à votre carte Arduino.

Vous avez déjà utilisé plusieurs fois le mot Ethernet. De quoi s'agit-il au juste ? Ça doit avoir quelque chose à voir avec Internet ou le réseau, n'est-ce pas ?

C'est vrai, Arduus ! Et c'est une belle opportunité pour aborder certains points spécifiques aux réseaux.

Ethernet

Le mot Ethernet qualifie une technologie câblée pour transmettre des données. Depuis les années 1990, elle est la norme pour toute une gamme de technologies LAN (*Local Area Network*). Le transfert des données est, en principe, assuré par un câble à paire torsadée (*Twisted-Pair-Cable*) selon la norme CAT-5 ou supérieure.

TCP/IP

Ethernet utilise un protocole appelé TCP (*Transfer Control Protocol*, protocole de contrôle de transfert en français) pour transmettre des données. Ce protocole permet de transférer des informations au moyen d'un réseau local ou global et garantit une communication sans erreurs. Des mécanismes permettent, en cas d'erreur de données menaçante, de corriger ou de retransmettre les paquets de données à

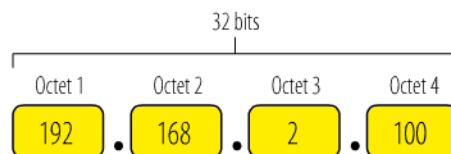


transférer. La désignation IP (*Internet Protocol*) concerne l'adressage des paquets de données à transférer qui doivent être acheminés de l'émetteur à un destinataire bien défini. Ce protocole assure donc l'adressage des paquets de données à transmettre. Chaque utilisateur du réseau possède une adresse précise, comparable au numéro de maison dans une rue. Pour qu'un colis puisse être par exemple livré à coup sûr par la Poste, les numéros des maisons ne doivent pas être en double, ce qui est le cas normalement. L'IP est toujours indiqué ou utilisé en rapport avec le TCP.

Adresse IP

L'adresse IP d'un utilisateur doit satisfaire à l'exigence d'univocité dans un réseau. Elle est affectée à un appareil qui fait partie du réseau, garantissant ainsi qu'il est adressable et accessible. Les adresses IP de la notation Ipv4 sont composées de quatre octets (32 bits).

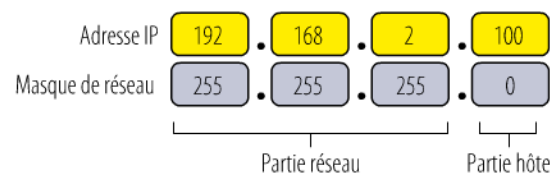
Figure 17-4 ►
Capture d'écran



Cette adresse a été attribuée par mon routeur à mon ordinateur, afin que je sois disponible sur le réseau.

Masque de réseau

Une adresse IP comprend toujours une partie réseau et une partie hôte. Le masque de réseau définit quant à lui combien d'appareils doivent être atteints dans un réseau et lesquels se trouvent dans d'autres réseaux.



Pour parvenir à la partie hôte, l'adresse IP est combinée au masque de réseau par une opération ET. D'après le masque ci-dessus, il est théoriquement possible d'avoir $2^8 = 256$ ordinateurs dans le réseau. Je dis bien théoriquement car 255, par exemple, a une signification particulière. Des détails supplémentaires sortiraient du cadre de ce livre,

c'est pourquoi je vous invite à consulter la littérature spécialisée ou à regarder sur Internet.

Adresse MAC

L'adresse MAC (*Media Access Control*) doit être sans ambiguïté à l'échelle mondiale. Elle a été attribuée à chaque adaptateur de réseau. Elle se compose de six octets, les trois premiers contenant un code fabricant OUI (*Organizational Unit Identifier*). Les trois autres octets contiennent l'indicatif de l'appareil, assigné par le fabricant en question. Voici un exemple d'adresse MAC pour une carte réseau :

1C-6F-65-94-D5-1A

Passerelle

Une passerelle (*gateway*, en anglais) est un passage vers une zone particulière qui, rapporté à notre thématique, peut être appelé passerelle de réseau. De quel appareil pourrait-il s'agir ? Le routeur, qui se trouve à moitié sur Internet, passe pour être une passerelle. Mon routeur a par exemple 192.168.2.1 comme adresse IP et transmet mes demandes à mon fournisseur d'accès, c'est-à-dire sur Internet. Si vous ouvrez une console DOS et que vous écrivez la commande `ipconfig /all`, vous obtenez, entre autres, les indications suivantes :

```
Standardgateway ..... : 192.168.2.1  
DHCP-Server ..... : 192.168.2.1
```

La figure 17-5 montre le shield Ethernet combiné à votre carte Arduino.



◀ **Figure 17-5**
Shield Ethernet et carte Arduino

Composants nécessaires



1 shield Ethernet (<http://arduino.cc/en/Guide/ArduinoEthernetShied>)



1 câble réseau, suffisamment long pour aller du routeur au shield Ethernet



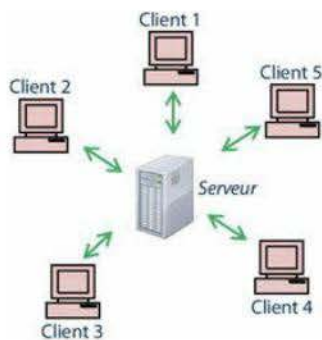
1 shield d'entrée analogique



Attention !

Utilisez un câble normal pour relier votre shield Ethernet à votre routeur. Ces câbles sont jaunes, blancs ou même noirs. Ne vous servez pas d'un câble réseau rouge entre votre routeur et le shield Ethernet, car il s'agit généralement d'un câble croisé qui ne doit être employé que pour relier votre shield directement à la carte réseau de votre ordinateur. Les lignes de réception et d'émission sont alors croisées. Vous trouverez des informations plus précises sur Internet.

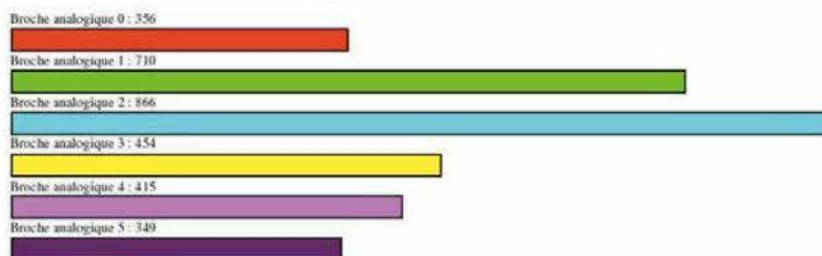
Le sketch suivant permet d'utiliser le shield Ethernet comme un serveur Web. Quand vous passez par un navigateur web (tel Firefox, Opera ou IE) pour vous connecter à Internet, vous établissez une liaison avec un serveur web (voir figure 17-6).



◀ **Figure 17-6**
Shield Ethernet et carte Arduino

La figure 17-6 montre un serveur (fournisseur) au centre, qui répond aux requêtes de nombreux clients (utilisateurs). Un serveur est un logiciel qui réagit à une demande de contact venant de l'extérieur et délivre des informations. Il peut s'agir d'un serveur mail ou FTP ou encore d'un serveur web. Un client peut être un client mail, tel que Thunderbird ou Outlook. S'il s'agit d'un client web, cela peut être Firefox, Opera ou IE, tous déjà mentionnés dans ce livre. Prenons maintenant un exemple concret, dans lequel le shield Ethernet, en tant que serveur web, doit envoyer les valeurs des entrées analogiques de la carte Arduino. La figure 17-7 offre un aperçu de l'affichage dans le navigateur web.

Valeurs des entrées analogiques



◀ **Figure 17-7**
Affichage de la page HTML dans le navigateur web (représentation numérique et graphique)

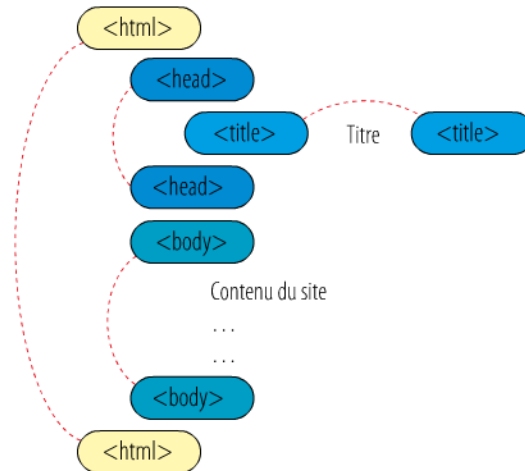
Vous n'êtes pas sérieux ! Dois-je en plus apprendre à programmer un site Internet ?

Eh oui, Ardus ! On ne peut pas faire autrement, mais soyez rassuré. Nous n'allons qu'effleurer le sujet car celui-ci pourrait sans peine remplir toute une bibliothèque. Les sites Internet sont programmés en HTML (*Hypertext Markup Language*). Il s'agit d'un langage de balisage à base de texte permettant par exemple de représenter du texte, des images, des vidéos ou des liens sur un site Internet, et de lire et afficher ceux du navigateur web. Vous trouverez ci-après la trame de



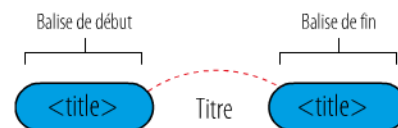
base d'un site, que nous remplirons par la suite pour présenter nos informations. La plupart des éléments HTML sont identifiés par des paires de balises (*tags*). Il y a toujours une balise de début (ouvrante) et une balise de fin (fermante). La figure 17-8 montre la trame de base en question, les paires correspondantes étant indiquées en couleurs.

Figure 17-8 ►
Trame de base d'un site Internet



Les lignes pointillées en rouge vous permettent de voir les formations de paires. Les différentes balises ou éléments HTML sont constituées par les noms des éléments entre chevrons. Voyons maintenant une paire de balises de plus près :

Figure 17-9 ►
La paire de balises title



Cette paire génère le titre du site Internet, le texte se trouvant entre la balise de début et la balise de fin. La balise de fin présente le même nom d'élément que la balise de début, cependant il est précédé d'une barre oblique (appelée *slash*).

Code du sketch

```
#include <SPI.h>
#include <Ethernet.h>

byte MACAddress[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED}; // Adresse MAC
byte IPAddress[] = {192, 168, 2, 110}; // Adresse IP
int const HTTPPORT = 80; // Port HTTP 80 (port standard)
```

```

String barColor[] = {"ff0000", "00ff00", "00ffff",
                    "ffff00", "ff00ff", "550055"};
                    // Couleurs RGB pour barres de couleur
#define HTML_TOP    "<html>\n<head><title>Arduino Web-Server</title>\n</head>\n<body>"
#define HTML_BOTTOM "</body>\n</html>"
EthernetServer myServer(HTTPPORT); // Démarrage du serveur web sur
                                   //le port indiqué

void setup(){
  Ethernet.begin(MACAddress, IPAddress); //Initialisation Ethernet
  myServer.begin();                      // Démarrage du serveur
}

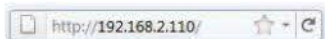
void loop(){
  EthernetClient myClient = myServer.available();
  if(myClient){
    myClient.println("HTTP/1.1 200 OK");
    myClient.println("Content-Type: text/html");
    myClient.println();

    myClient.println(HTML_TOP);          // HTML début
    showValues(myClient);                // Contenu HTML
    myClient.println(HTML_BOTTOM);       // HTML fin
  }
  delay(1);                             // Courte pause pour navigateur Web
  myClient.stop();                       // Fermeture connexion client
}

void showValues(EthernetClient &myClient){
  for(int i = 0; i < 6; i++){
    myClient.print("Analog Pin");
    myClient.print(i);
    myClient.print(": ");
    myClient.print(analogRead(i));
    myClient.print("<div style='height: 15px; background-color: #'");
    myClient.print(barColor[i]);
    myClient.print("; width: ");
    myClient.print(analogRead(i));
    myClient.println("px; border: 2px solid;'\n></div>");
  }
}

```

Pour accéder au serveur web Arduino, écrivez l'adresse IP du code de sketch dans la ligne d'adresse de votre navigateur web Arduino. Dans mon cas, l'adresse est la suivante.



Si cette adresse vous semble trop énigmatique, vous pouvez bien sûr en choisir une plus parlante comme :



Il vous suffit d'adapter dans Windows le fichier hosts avec des droits d'administrateur sous C:\Windows\System32\drivers\etc et d'ajouter la ligne dans laquelle j'ai indiqué le nom Arduino :

```
# localhost name resolution is handled within DNS itself
# 127.0.0.1      localhost
# ::1           localhost
192.168.2.110   Arduino
```

L'appel est alors plus simple et vous n'avez pas besoin de retenir l'adresse IP.

Revue de code

Du point de vue logiciel, les variables suivantes sont nécessaires à notre serveur web expérimental.

Tableau 17-1 ►
Variables nécessaires et leur objet

Variable	Objet
MACAddress[]	Tableau unidimensionnel pour stocker l'adresse MAC du shield Ethernet
IPAddress[]	Tableau unidimensionnel pour stocker l'adresse IP du shield Ethernet
HTTPPORT	Variable pour stocker l'adresse du port HTML
BarColor[]	Tableau unidimensionnel pour stocker les informations de couleurs des barres horizontales
HTML_TOP	Équivalent du code HTML pour la partie supérieure (en-tête)
HTML_BOTTOM	Équivalent du code HTML pour la partie inférieure (fin)

Deux bibliothèques doivent être incorporées pour pouvoir utiliser la fonctionnalité du shield Ethernet :

- SPI.h : Serial-Peripheral-Interface-Bus est nécessaire pour les versions Aduino > 0018 ;
- Ethernet.h.

Je voudrais vous poser une question au sujet de la variable HTTPPORT. N'est-ce pas une faute de frappe ? Ne s'agit-il pas de HTMLPORT ? Je croyais qu'il était question ici de pages HTML.



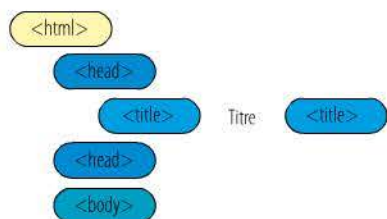
C'est vrai Arduus qu'on s'y perd un peu au début. HTTP est la forme abrégée de *Hypertext Transfer protocol*. Comme vous l'avez peut-être remarqué, on a affaire à un grand nombre de protocoles différents en informatique. Quand il s'agit de pages web, ce protocole est chargé de la transmission. Quand vous tapez une adresse web dans votre navigateur, celle-ci commence la plupart du temps par `http://` et non par `html://`. Passons maintenant à la définition du port. Le port standard pour des serveurs web qui utilisent le protocole HTTP est le numéro 80. Imaginez-vous ce numéro comme une sorte de bifurcation sur la route du réseau, où d'autres protocoles circulent encore. Voici une courte liste d'applications dont vous avez peut-être déjà entendu parler.

Port	Service	Rôle
21	FTP	Transfert de fichier via FTP-Client
25	SMTP	Envoi d'e-mails
110	POP3	Accès client à un serveur e-mails

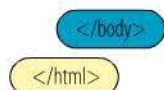
◀ **Tableau 17-2**

Courte liste avec numéros de port et services

Je voudrais encore vous parler brièvement de la structure d'une page HTML. La seule partie variable de notre page est la partie que j'ai appelée *Contenu de ma page*. Tout ce qui est au-dessus ou en dessous ne change pas. C'est pour cette raison que j'ai créé les raccourcis pour la partie supérieure :



dans la définition HTML_TOP et pour la partie inférieure :



dans HTML_BOTTOM. Vous retrouverez la même chose dans le sketch, avec les lignes suivantes :


```
#define HTML_TOP      "<html>\n<head><title>Arduino Web-Server\n</title></head>\n<body>\n"
#define HTML_BOTTOM  "</body>\n</html>"
```

La séquence d'échappement `\n` provoque un saut de ligne, de telle sorte que le code HTML soit formaté d'une certaine manière et que tout ne soit pas mis sur une seule ligne. Venons-en maintenant au déroulement du sketch proprement dit. Diverses parties du programme sont comme toujours initialisées dans la fonction `setup`.

```
void setup(){
  Ethernet.begin(MACAddress, IPAddress); // Initialisation Ethernet
  myServer.begin();                      // Démarrage du serveur
}
```

La première étape consiste à doter le shield Ethernet de l'adresse MAC et d'une adresse IP unique.



Dites-moi s'il vous plaît d'où vous sortez l'adresse IP 192.168.2.110 en question. Ça reste un mystère pour moi.

Eh bien Ardu, c'est tout simple ! Mon routeur se trouve dans la zone d'adresse 192.168.2 et l'adresse d'hôte 1 lui est attribuée, autrement dit son adresse IP est 192.168.2.1. Je peux donc affecter des adresses comprises entre 192.168.2.2 et 192.168.2.254 à d'autres utilisateurs du réseau. Revenons à l'initialisation. La deuxième étape consiste à démarrer le serveur web, de sorte qu'il puisse réagir à des demandes entrantes. Celui-ci épie le réseau et reste sur le qui-vive jusqu'à ce qu'un client l'aborde et lui demande quelque chose.

Il accomplit ensuite son travail et délivre les données avant de se remettre à nouveau en position d'attente. Passons maintenant au traitement proprement dit dans la fonction `loop`. La présence de la demande d'un client est d'abord vérifiée :

```
EthernetClient myClient = myServer.available();
if(myClient){
```

Si l'interrogation `if` est satisfaite, le serveur peut commencer à envoyer ses informations au client.

C'était quoi déjà l'interrogation `if` ? On y lit `myClient` au lieu d'une expression à évaluer.

Pas de problème, Arduus ! Ce n'est que la forme abrégée du code suivant :

```
if(myClient == true){...}
```

L'interrogation sur `true` est facultative du fait que si l'expression est vraie dans l'instruction `if`, c'est le bloc subséquent qui est exécuté. Vous n'avez pas besoin de vérifier à nouveau avec `==` que l'expression est vraie. Vous comprenez maintenant ? Donc, si un client a effectué une demande auprès du serveur, ce dernier renvoie pour commencer les lignes suivantes :

```
myClient.println("HTTP/1.1 200 OK");  
myClient.println("Content-Type: text/html");  
myClient.println();
```

Dans la première ligne, le serveur confirme la demande du client en transmettant la version 1.1 du protocole HTTP, suivie du code d'état 200 indiquant que la demande a été traitée avec succès et que le résultat de la demande est transmis dans la réponse. Dans la deuxième ligne, le *type MIME*, `text/html` dans notre cas, est communiqué. Celui-ci décrit le genre des données envoyées par le serveur. S'agit-il d'informations purement textuelles ou une image est-elle éventuellement délivrée au client ? Les données transmises doivent alors être bien sûr interprétées en conséquence et non pas affichées en texte clair. Passons maintenant au code, qui envoie les données lues de votre carte Arduino :

```
myClient.println(HTML_TOP);           // HTML début  
showValues(myClient);                 // Contenu HTML  
myClient.println(HTML_BOTTOM);       // HTML fin
```

Les tâches de `HTML_TOP` et `HTML_BOTTOM` vous sont connues. L'appel des données de la carte est exécuté par la fonction `showValues` que je vous redonne ici :

```
void showValues(EthernetClient &myClient){  
  for(int i = 0; i < 6; i++){  
    myClient.print("Analog Pin");  
    myClient.print(i);  
    myClient.print(": ");  
    myClient.print(analogRead(i));  
    myClient.print("<div style='height: 15px; background-color: #'");
```



```

myClient.print(barColor[i]);
myClient.print("; width: ");
myClient.print(analogRead(i));
myClient.println("px; border: 2px solid;\"></div>");
}
}

```



Ayant par chance une loupe sur moi, je vois dans l'en-tête de fonction un ET commercial (&) devant le paramètre `myClient`. Vous connaissant, il ne s'agit pas d'une faute de frappe, n'est-ce pas ?

Non Arduus, ce n'est pas une erreur. Ce signe distinctif est en fait une *référence*. Quand on passe une variable comme paramètre d'une fonction, celle-ci travaille avec une copie de cette variable, ce qui n'a aucune influence sur la variable d'origine. La fonction peut par exemple doubler la valeur du paramètre. L'originale demeure inchangée. Mais pour pouvoir utiliser l'objet `Client` original dans la fonction, l'adresse mémoire de l'original est communiquée au moyen de l'opérateur de référence `&`. Dans la fonction, je travaille quasiment avec l'original. La fonction affiche d'une part les valeurs des entrées analogiques et de l'autre des barres horizontales. J'utilise pour ce faire la balise `div`, qui peut servir de contenant pour d'autres éléments HTML. Je m'en sers ici pour colorer une certaine zone. Il est possible de donner des informations de hauteur ou de largeur au moyen d'une indication *style*. Une ligne HTML peut alors ressembler à cela :

```

Analog Pin 0: 168<div style="height: 25px; background-color:
                    #ff0000; width: 168px; border: 2px solid;\"></div>

```

La zone `div` a ici une hauteur de 25 pixels et une largeur de 168 pixels. Vous trouverez des informations détaillées dans la littérature spéciale ou sur Internet.



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- `selfhtml` ;
- `cascading stylesheets` ;
- `div-tag`.

Il y a quelque chose que je n'ai pas trouvé commode lors de ma réalisation : les valeurs des entrées analogiques s'affichent un point c'est tout. Si je tourne l'un des potentiomètres, rien ne bouge sur la page Internet. J'aurais pourtant bien voulu.



C'est inutile, Arduus. Le navigateur web appelle une page auprès du serveur web et en assure la présentation (ce procédé est également appelé *rendu*). Si le navigateur n'émet aucune autre demande, le contenu de la page demeure bien entendu inchangé. Vous pouvez toujours appuyer assez souvent sur la touche Actualiser (F5) mais je doute que cela vous donne satisfaction. Modifiez plutôt dans votre sketch la ligne de code où HTML_TOP a été défini et vous verrez que le comportement de votre navigateur changera.

```
#define HTML_TOP "<html>\n<head><title>Arduino Web-server</title>\n</head>\n<meta http-equiv=\"refresh\" content=\"1\">\n<body>"
```

Le passage suivant est décisif :

```
<meta http-equiv=\"refresh\" content=\"1\">
```

La balise *meta* en question demande au navigateur d'exécuter automatiquement une actualisation (*refresh*) toutes les secondes. Le *back-slash* (barre oblique inverse) \ à la fin de la première ligne définissant HTML_TOP permet que cette ligne se poursuive sur la suivante. Faute de quoi une erreur de compilateur se produirait.

Problèmes courants

Vérifiez ce qui suit si la page du serveur web ne s'affiche pas.

- Avez-vous saisi la bonne adresse IP dans la ligne d'adresse de votre navigateur ? Elle doit correspondre à celle de votre sketch.
- Pouvez-vous atteindre le serveur web en tapant une commande ping dans la ligne de commande ? Sinon, vérifiez votre câble réseau ou éventuellement les réglages du pare-feu. Une exécution réussie de la commande ping donne le résultat suivant.

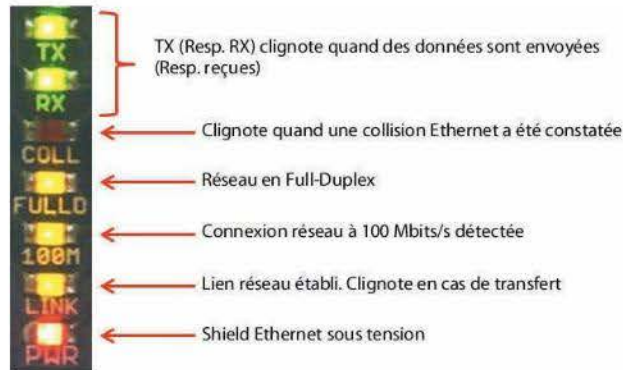
```
C:\Users\Olivier>ping 192.168.2.110

Envoi d'une requête 'Ping' 192.168.2.110 avec 32 octets de données :
Réponse de 192.168.2.110 : octets=32 temps=2 ms TTL=128
Réponse de 192.168.2.110 : octets=32 temps=3 ms TTL=128
Réponse de 192.168.2.110 : octets=32 temps=2 ms TTL=128
Réponse de 192.168.2.110 : octets=32 temps=2 ms TTL=128

Statistiques Ping pour 192.168.2.110:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
    Durée approximative des boucles en millisecondes :
        Minimum = 2ms, Maximum = 3ms, Moyenne = 2ms

C:\Users\Olivier>
```


- Le shield Ethernet possède certaines LED qui donnent des informations sur l'état (voir figure suivante).



- Vérifiez l'affichage des LED. Les LED PWR et LINK doivent être allumées. La LED 100M ne s'allume que dans le cas d'un réseau 100 Mbits/s. Elle reste éteinte pour 10 Mbits/s. Si des données sont envoyées toutes les secondes comme dans le dernier exemple, Les LED TX et RX clignent au même rythme.

Qu'avez-vous appris ?

- Vous avez appris à réaliser un serveur web avec le shield Ethernet.
- Vous avez interrogé les entrées analogiques et vu comment s'affichent les valeurs à peu de choses près en temps réel.
- La trame de base d'une page HTML devrait maintenant vous être plus familière.

Exercice complémentaire

Écrire un nouveau sketch montrant, en plus des entrées analogiques, l'état des entrées numériques sur votre page web Arduino.

Numérique appelle analogique

Au sommaire :

- la fabrication d'un shield générateur de signaux analogiques ;
- comprendre ce qu'est un convertisseur numérique-analogique ;
- savoir ce qu'est un réseau de résistances R2R ;
- savoir ce qu'est un registre de port ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

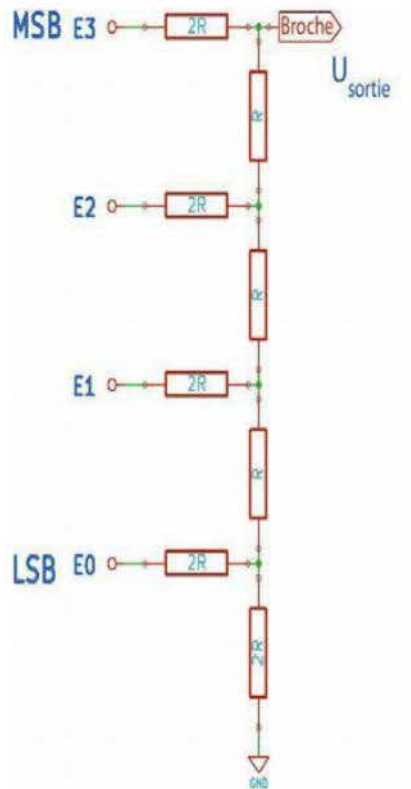
Comment convertir des signaux numériques en signaux analogiques ?

L'exploitation de signaux analogiques est relativement simple par les entrées analogiques avec votre carte Arduino. Le sens inverse – donc produire et distribuer une tension analogique à l'aide du microcontrôleur – n'est faisable qu'en utilisant les sorties numériques capables de MLI. Quand vous aurez vu la forme de la courbe des signaux MLI, vous saurez combien elle diffère de celle d'un signal analogique. La plupart des microcontrôleurs courants ne convertissent pas un signal numérique en signal analogique. Il leur faudrait pour ce faire intégrer un convertisseur N/A.

Notre montage consiste à fabriquer un tel convertisseur, appelé aussi CNA (*Digital-Analog-Converter* en anglais ou DAC), avec des moyens simples. Tout tourne ici autour du réseau R2R. Cette appella-

tion est due au fait que le convertisseur est composé de plusieurs résistances, disposées en cascade et qui doivent se trouver entre elles dans un rapport déterminé. La disposition des éléments fait penser à une échelle, c'est pourquoi ce type de circuit est également nommé réseau en échelle de résistances dans la littérature spécialisée. Retenons seulement que le réseau de résistances sert à répartir une tension de référence qui, dans notre cas, est de +5 V. La figure 18-1 montre un réseau de résistances R2R avec une entrée de 6 bits.

Figure 18-1 ►
Réseau de résistance R2R
avec une entrée de 6 bits



Vous vous demandez peut-être d'où vient ce nom R2R. Si vous regardez le schéma de plus près, vous verrez que les résistances n'ont pas une valeur fixée, et que seuls les rapports de résistance sont indiqués. Les valeurs des résistances (horizontales), qui sont reliées aux connexions E_0 à E_5 des sorties numériques, sont le double de celles des résistances (verticales), qui relient les résistances précédentes et mènent au point de sortie U_{sortie} . La résistance du bas, qui est reliée à la masse, a la même résistance $2R$ que les résistances horizontales. La formule suivante peut être utilisée pour déterminer la tension de sortie :

$$U_{\text{sortie}} = \frac{U_{e5}}{2} + \frac{U_{e4}}{4} + \frac{U_{e3}}{8} + \frac{U_{e2}}{16} + \frac{U_{e1}}{32} + \frac{U_{e0}}{64}$$

Pour cet exemple avec cinq entrées, la résolution suivante peut être obtenue :

$$U_{\text{résolution}} = \frac{U_{\text{ref}}}{64}$$

U_{ref} est ici la tension avec laquelle les différentes entrées sont commandées. Pour une tension U_{ref} de 5 V, le résultat serait donc le suivant :

$$U_{\text{résolution}} = \frac{U_{\text{ref}}}{64} = \frac{5 \text{ V}}{64} = 78,13 \text{ mV}$$

Cette valeur représente le plus petit pas de progression obtenu chaque fois que la valeur binaire de l'entrée à 6 bits est incrémentée de 1. Le tableau 18-1 fournit les quatre premières valeurs ainsi que la dernière.

Valeur binaire	Tension de sortie
000000	0 V
000001	78,13 mV
000010	156,26 mV
000011	234,39 mV
...	...
111111	5 V

◀ **Tableau 18-1**

Combinaisons binaires et tensions de sortie arrondies

Nous avons donc, pour le shield de conversion N/A prévu, une définition de 6 bits ($2^6 = 64$).



Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

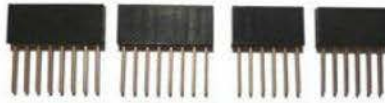
- réseau R2R ;
- réseau de résistances en échelle.

Vous avez peut-être remarqué que je n'ai donné jusqu'ici aucune valeur de résistance. Ce n'est pas utile tant que le rapport des résistances est exactement de 2:1. En outre, la tolérance des différentes résistances doit être aussi faible que possible pour obtenir des résultats relativement précis. Nous n'en tiendrons cependant pas compte dans ce montage.

Composants nécessaires



17 résistances de $47\text{ k}\Omega$



1 jeu de connecteurs femelles empilables ($2 \times 8 + 2 \times 6$)



1 carte de shield



Fils, si possible de différentes couleurs

Réflexions préliminaires

Le réseau R2R avec rapports de résistance 2:1 peut s'avérer difficile à réaliser car vous devez trouver des valeurs de résistance qui sont dans ce rapport entre elles. La solution n'est certes pas simple. J'ai choisi une résistance de $47\text{ k}\Omega$ pour que les courants en circulation ne soient pas trop élevés.

Vous vous demandez peut-être si une résistance de $23,5\text{ k}\Omega$ existe. Non seulement je ne crois pas, mais cette valeur est en plus très facile à obtenir. Quand on branche deux résistances de même valeur en parallèle, le résultat obtenu est l'exacte moitié de la résistance en question. Donc si $R_1 = R_2$, on obtient l'équation suivante.

$$\frac{1}{R_{totale}} = \frac{1}{R_1} + \frac{1}{R_2} = \frac{1}{R} + \frac{1}{R} = \frac{2}{R}$$

$$\text{donc } R_{totale} = \frac{R}{2}$$

C'est élémentaire, n'est-ce pas ?

Code du sketch

```
int pinArray[] = {8, 9, 10, 11, 12, 13};
byte R2RPattern;
void setup(){
  for(int i = 0; i < 6; i++)
    pinMode(pinArray[i], OUTPUT);
  R2RPattern = B000001; //Configuration binaire pour commander
                        //les sorties numériques
}

void loop(){
  for(int i = 0; i < 6; i++){
    digitalWrite(pinArray[i], bitRead(R2RPattern,i) == 1?HIGH:LOW);
  }
}
```

Ce sketch, vraiment très court, commande les sorties numériques sur lesquelles se trouve le réseau R2R. Elles sont commandées via la variable R2RPattern, qui délivre une tension correspondante à la sortie du réseau.

Revue de code

Du point de vue logiciel, les variables indiquées dans le tableau suivant sont nécessaires à notre montage.

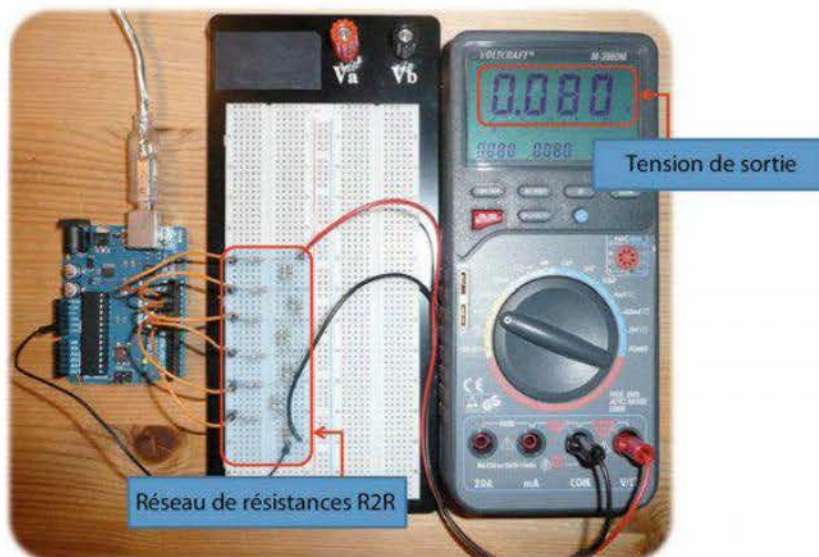
Variable	Objet
pinArray	Tableau unidimensionnel pour stocker les broches connectées à l'afficheur
R2Rpattern	Contient la combinaison de bits utilisée pour commander le réseau R2R

◀ **Tableau 18-2**

Variables nécessaires et leur objet

La figure 18-2 illustre le circuit que j'ai créé sur une plaque d'essais avant de le reporter sur le shield R2R.

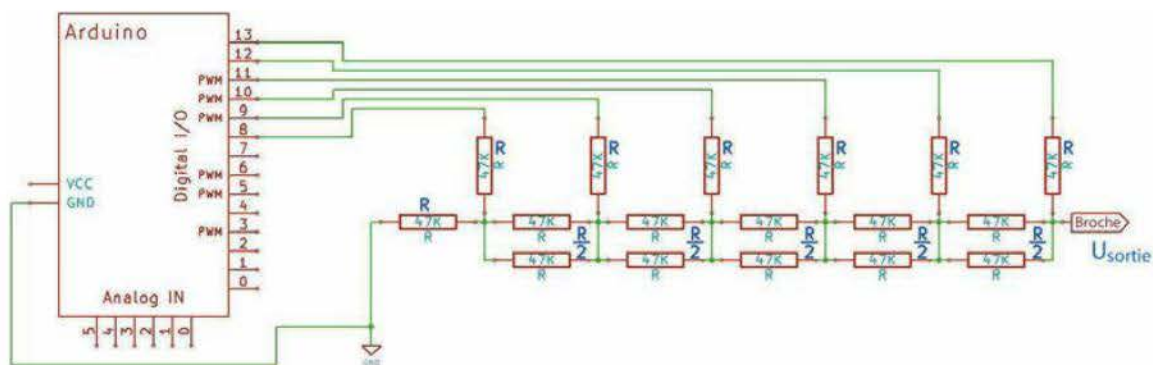
Figure 18-2 ►
Réseau R2R sur une plaque
d'essais (tension de sortie pour une
combinaison binaire de 000001)



Le réseau est commandé avec la combinaison de bits 000001 tirée du sketch et le multimètre affiche une tension de 0,080 V, soit 80 mV. Dans le tableau 18-1, la valeur est de 78,13 mV pour la combinaison de bits. La valeur de sortie de 80 mV ne correspond donc pas tout à fait à la valeur calculée du tableau, mais c'est tout de même correct quand on sait que le résultat se trouve par exemple légèrement faussé par les tolérances matérielles des résistances utilisées ou par des erreurs d'affichage du multimètre. Il m'est arrivé de construire un réseau R2R où les valeurs coïncidaient presque toutes jusqu'à deux chiffres après la virgule, mais ce n'était que pur hasard.

Schéma

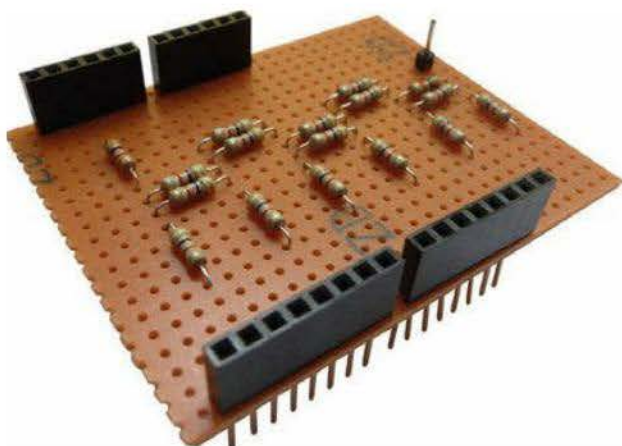
Comme vous pouvez le voir dans la figure suivante, le circuit est uniquement composé de résistances reliées d'une manière particulière pour constituer un réseau R2R.



Les résistances R ont naturellement une valeur de $47\text{ k}\Omega$. Les paires de résistances $R/2$ ont comme valeur résultante $23,5\text{ k}\Omega$.

▲ **Figure 18-3**
Commande du réseau R2R
par 6 sorties numériques

Réalisation du shield



◀ **Figure 18-4**
Réalisation du réseau R2R
sur un shield dédié

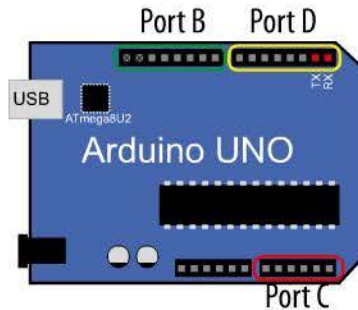
La figure montre bien le réseau de résistances, la broche en haut du shield étant la sortie sur laquelle vous pouvez brancher votre multimètre pour mesurer la tension de sortie.

Commande du registre de port

Je ne vous apprendrai rien en vous disant que la carte Arduino ne communique que par les entrées et les sorties. Ceci vaut également pour commander des LED, des moteurs, des servomoteurs et pour lire, entre autres, les valeurs d'un capteur de température ou d'une résistance réglable ou photosensible.

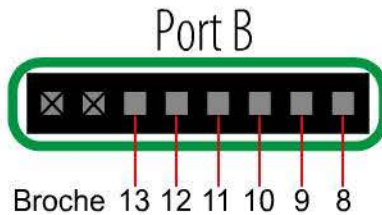
Votre microcontrôleur ATmega328 travaille en interne avec ce qu'on appelle des registres, raccordés aux entrées et sorties (broches). Dans le domaine informatique, ce sont des zones de mémoire à l'intérieur d'un processeur, qui sont reliées directement à l'unité centrale de calcul. Ainsi l'accès à ces zones est très rapide puisque le détour par des circuits mémoire externes est évité. Les différentes broches de votre carte Arduino sont reliées en interne à des registres de port, encartouchés en couleurs (vert, rouge ou jaune) et nommés port B, C ou D dans la figure 18-5.

Figure 18-5 ►
Registres de port
de la carte Arduino



Regardons par exemple le port B de plus près (figure 18-6).

Figure 18-6 ►
Registre de port B



On reconnaît immédiatement les entrées et sorties numériques (broches 9 à 13). Les deux broches de gauche sont pour nous sans intérêt, car elles sont reliées à *Aref* (entrée pour la tension de référence du convertisseur analogique-numérique) et à la masse et ne peuvent être manipulées. Six bits en tout sont donc disponibles dans le registre de port B. Ils vont nous servir à faire des choses diverses. Comme par hasard, notre réseau de résistances est lui aussi commandé avec 6 bits. Je ne vous en dis pas plus pour l'instant. Chacun des trois ports est sollicité dans un sketch au moyen des identifiants suivants :

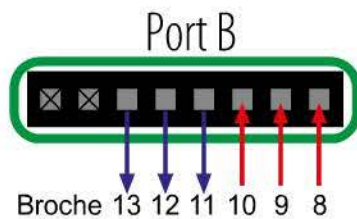
- PORTB ;
- PORTC ;
- PORTD.

Nous savons déjà comment les différents ports sont sollicités mais nous nous en tiendrons, comme je l'ai dit plus haut, au port B dans notre exemple.

J'ai d'emblée une question à poser. Quand on programme des broches numériques, on doit définir dans la fonction `setup` si elles vont servir d'entrée ou de sortie. Dans le cas d'un registre de port, comment dois-je lui dire de servir d'entrée ou de sortie ?



Et bien Arduus, vous m'offrez là une transition rêvée pour passer au point suivant. Mais je dois vous dire quelque chose avant : vous pouvez bien sûr attribuer à chaque bit du registre de port un sens de circulation des données particulier. Le registre complet ne fonctionne pas de telle sorte que toutes les broches servent d'entrées ou de sorties. Chaque broche peut être configurée séparément. D'autres registres sont en effet spécialement prévus pour influencer sur le sens de circulation des données de chaque broche. Ils ont pour nom `DDRx`, `x` indiquant le port à solliciter. Le registre `DDRB` est par conséquent celui de notre `PORTB` – `DDR`, pour *Data Direction Register*, signifie à peu de choses près registre de direction de données. Voyons comment tout cela fonctionne dans le détail. Avant d'utiliser un port, je dois donc définir le sens de circulation des données par le `DDR` correspondant. Dans la figure 18-7, les flèches indiquent les sens de circulation des données que nous voulons obtenir avec notre programmation.



◀ **Figure 18-7**

Registre de port B avec divers sens de circulation des données selon les broches

La configuration est donc la suivante :

- entrées : broches 8, 9 et 10 ;
- sorties : broches 11, 12 et 13.

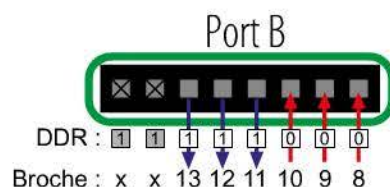
Pour affecter un sens de circulation des données à une broche, la valeur suivante doit être entrée dans le `DDR`.

Tableau 18-3 ►
Valeurs pour le DDR

Valeur	Mode de fonctionnement
0	Broche servant d'entrée, comparable à <code>pinMode(pin, INPUT);</code>
1	Broche servant de sortie, comparable à <code>pinMode(pin, OUTPUT);</code>

Cela nous donne pour le DDR la programmation de la figure 18-8.

Figure 18-8 ►
Initialisation du DDR
pour les différents sens
de circulation des données



Nous pouvons maintenant mettre par exemple les sorties numériques des broches 11, 12 et 13 au niveau HIGH par l'instruction `PORTB`. Voici un extrait correspondant tiré d'un sketch :

```
void setup(){
    DDRB = 0b11111000;    //Broches 8, 9, 10 = INPUT. Broches 11,
                           //12, 13 = OUTPUT
    PORTB = 0b00111000;    //Mise des broches 11, 12, 13 au niveau HIGH
}

void loop(){ /* vide */ }
```

Les deux bits les plus significatifs pour les broches non utilisées ont simplement été pourvus d'un 1 dans le DDR. Cela n'a aucune importance dans notre cas. Si vous regardez la mise des sorties au niveau HIGH, que constatez-vous de différent par rapport à la manipulation des broches habituelle ? Je vous donne les deux variantes pour comparaison :

```
digitalWrite(11, HIGH);    PORTB = 0b00111000;
digitalWrite(12, HIGH);
digitalWrite(13, HIGH);
```

Aucune idée ? Bon. La manière traditionnelle de gauche met les différentes broches l'une après l'autre au niveau HIGH. Celle de droite met par contre toutes les broches *en même temps* au niveau HIGH avec une seule instruction, la configuration binaire étant alors appliquée simultanément à toutes les broches.

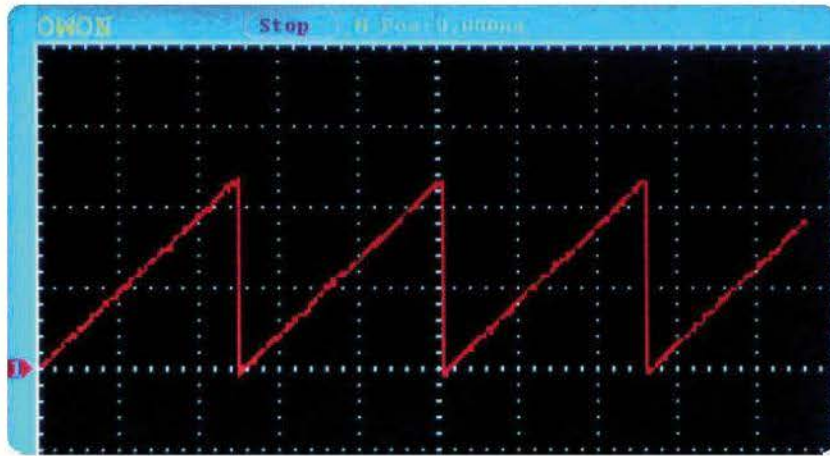
Mieux vaut donc choisir la nouvelle variante de manipulation des ports pour aller plus vite. Le sketch suivant génère une tension en dents de scie à la sortie du réseau de résistances :


```

void setup(){
  DDRB = 0b11111111; //Toutes les broches programmées
                      //en tant que sorties
}

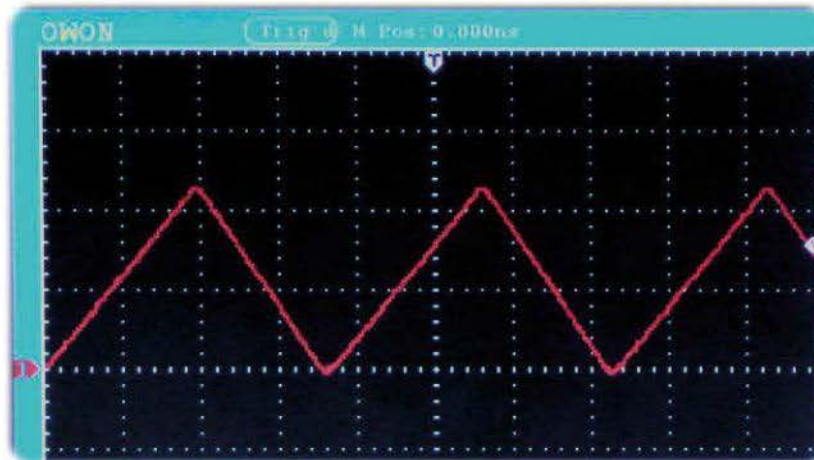
void loop(){
  for(int i = 0; i <= 63; i++) //63 = B00111111
    PORTB = i;                //Commande du registre de port B
}

```



◀ **Figure 18-9**
Oscillogramme avec une courbe
en dents de scie

D'après vous, comment adapter le sketch pour obtenir la courbe suivante ?



◀ **Figure 18-10**
Oscillogramme avec une courbe
en triangles

Bien joué si vous avez trouvé la solution.

```

void loop(){
  for(int i = 0; i <= 63; i++)

```



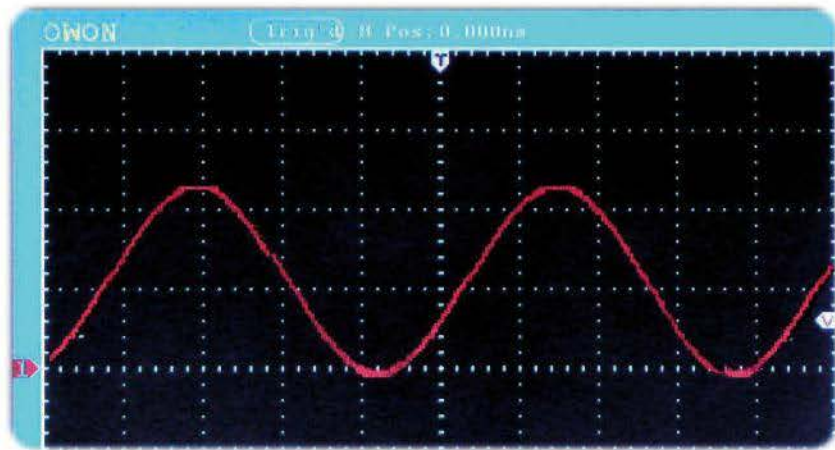
```

PORTB = i;           //Commande du registre de port B
                      //(front montant)
for(int i = 63; i >= 0; i--)
    PORTB = i;       //Commande du registre de port B
                      //(front descendant)
}

```

Quelles sont les autres courbes ? Qu'en est-il d'une courbe sinusoïdale ? La fonction sinus ayant besoin d'un certain temps pour calculer les valeurs, on a eu l'idée de créer des tables de correspondance ou *Lookup-Tables* (LUT). Les résultats d'un calcul y sont déjà enregistrés. On peut ainsi reproduire la courbe d'une fonction sinus en s'aidant des points qui se trouvent sur la courbe, par exemple.

Figure 18-11 ►
Oscillogramme avec une courbe
sinusoïdale



Le sketch pour générer la courbe sinusoïdale est vraiment laborieux à taper du fait que la LUT est très longue.

```

byte LUT[] =
{31, 32, 32, 33, 33, 34, 34, 35, 35, 36, 36, 37, 38, 38, 39, 39, 40, 40,
41, 41, 42, 42, 43, 43, 44, 44, 45, 45, 46, 46, 47, 47, 48, 48, 49, 49,
50, 50, 50, 51, 51, 52, 52, 52, 53, 53, 54, 54, 54, 55, 55, 55, 56, 56,
56, 57, 57, 57, 58, 58, 58, 59, 59, 59, 59, 60, 60, 60, 60, 60, 61, 61,
61, 61, 61, 61, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 61, 61, 61,
61, 61, 61, 60, 60, 60, 60, 60, 59, 59, 59, 59, 58, 58, 58, 57, 57, 57,
56, 56, 56, 55, 55, 55, 54, 54, 54, 53, 53, 52, 52, 52, 51, 51, 50, 50,
50, 49, 49, 48, 48, 47, 47, 46, 46, 45, 45, 44, 44, 43, 43, 42, 42, 41,
41, 40, 40, 39, 39, 38, 38, 37, 36, 36, 35, 35, 34, 34, 33, 33, 32, 32,
31, 30, 30, 29, 29, 28, 28, 27, 27, 26, 26, 25, 24, 24, 23, 23, 22, 22,
21, 21, 20, 20, 19, 19, 18, 18, 17, 17, 16, 16, 15, 15, 14, 14, 13, 13,
12, 12, 12, 11, 11, 10, 10, 10, 9, 9, 8, 8, 8, 7, 7, 7, 6, 6, 6, 5, 5,
5, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,

```

```

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6,
7, 7, 7, 8, 8, 8, 9, 9, 10, 10, 10, 11, 11, 12, 12, 12, 13, 13, 14, 14,
15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 23,
24, 24, 25, 26, 26, 27, 27, 28, 28, 29, 29, 30, 30, 31};

void setup(){
  DDRB = 0b11111111; //Toutes les broches programmées
                      //en tant que sorties
}

void loop(){
  for(int i = 0; i <=360; i++){
    PORTB = LUT[i]; //Commande du registre de port B
  }
}

```



Attention !

Vous courez le risque non négligeable de programmer votre microcontrôleur de telle sorte qu'il ne réagisse plus par la suite. Si vous regardez le port D, vous verrez que les signaux de contrôle RX et TX se trouvent respectivement sur les broches 0 et 1. RX sert à recevoir et TX à envoyer les données. Le sens de circulation des données est donc le suivant : RX = INPUT et TX = OUTPUT. Si vous modifiez par inadvertance la programmation de ces valeurs via DDRD, vous ne pourrez à coup sûr plus transmettre aucun sketch sur votre carte Arduino. Vous devez par conséquent être sûr de ce que vous faites. Mieux vaut vérifier trois fois votre sketch avant de l'envoyer au microcontrôleur.

Vous trouverez des informations plus précises sur :

<http://www.arduino.cc/en/Reference/PortManipulation>

Problèmes courants

Si la tension de sortie du réseau R2R ne correspond pas aux valeurs souhaitées pour la combinaison binaire, vérifiez les points suivants.

- Toutes les résistances utilisées pour le réseau R2R ont-elles la même valeur ?
- Aucune connexion au réseau n'a été oubliée ? (Je sais de quoi je parle car j'avais oublié un point de jonction, et j'ai passé près de dix minutes à trouver l'erreur !)

Qu'avez-vous appris ?

- Ce montage vous a présenté un réseau de résistances R2R.
- Avec ce réseau, vous avez pu réaliser un convertisseur numérique/analogique simple.
- Vous avez découvert les registres de port de votre carte Arduino et manipulé les sorties numériques au moyen du port B.

Exercice complémentaire

Essayez, en ajustant le tableau LUT, de créer des courbes de formes différentes. Vérifiez dans tous les cas que seuls 6 bits sont à votre disposition pour représenter une courbe. Le domaine de valeurs va de 0 à 63. Si vous êtes au-dessus, ni le circuit ni votre microcontrôleur n'en souffrira, mais la courbe ne ressemblera à coup sûr pas à ce que vous auriez souhaité.

Interactions entre Arduino et Raspberry Pi

Tous les mois ou presque, une nouvelle carte à microcontrôleur ou un nouveau descendant de la famille Raspberry Pi voit le jour. En introduction à cet ouvrage, j'ai mentionné que certains percevaient le Raspberry Pi comme une menace pour l'Arduino. Mais tout le monde ne partage pas cet avis. En effet, qu'est-ce qui empêche de bâtir un duo solide à partir des deux cartes ? Cela me paraît être une si bonne idée que j'ai décidé de consacrer un montage à ce sujet.

Au sommaire :

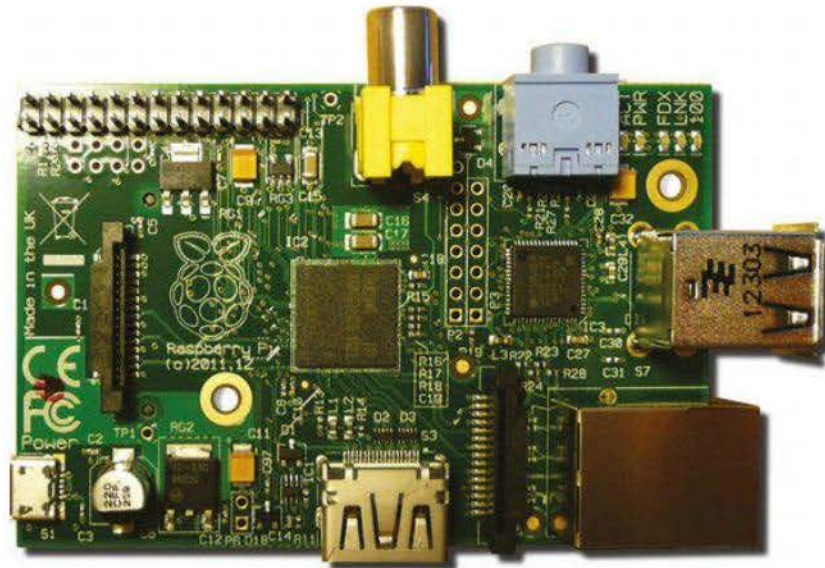
- le Raspberry Pi ;
- l'environnement de développement du Raspberry Pi et d'Arduino ;
- la communication au moyen de Python et de pyFirmata via le port USB ;
- la communication au moyen de Python et de pyFirmata via une liaison série TTL.

Réveillons l'Arduino sommeillant dans tout Raspberry Pi

Certes, le sujet de ce livre n'est pas le Raspberry Pi, mais il me paraît opportun d'étudier la carte d'un peu plus près afin d'en souligner les points forts. Il s'agit d'un ordinateur monocarte de la taille d'une carte bancaire qui a été développé par la fondation britannique Raspberry Pi. Il coûte environ 35 €, ce qui est donc très abordable. Mais ce n'est pas pour cette raison que les gens manifestent de l'intérêt pour ce nano-ordinateur. Cet ordinateur – puisqu'il s'agit bien d'un ordinateur à part entière – réunit tout ce qui est nécessaire pour s'aventurer dans l'univers de l'informatique et de la programmation. Il possède

un processeur Broadcom BCM2835 de type ARM11 cadencé à 70 MHz, une mémoire vive de 256 Mo ou 512 Mo, un port Ethernet (modèle B) pour le raccordement à un réseau, ainsi que deux ports USB. Comme support d'amorçage, le nano-ordinateur utilise une carte SD sur laquelle peuvent aussi être installés différents systèmes d'exploitation (Linux ou Android) compatibles avec l'architecture ARM.

Figure 19-1 ▶
Le Raspberry Pi



Si l'on veut raccorder un clavier ou une souris, il est préférable d'utiliser des modèles sans fil avec dongle USB afin que les deux périphériques n'occupent qu'un seul port USB. Comme la majorité des écrans TFT ou des écrans plats disposent aujourd'hui d'une connexion HDMI, vous pouvez donc facilement les raccorder au port HDMI de type A (*full size*) du Raspberry Pi. Si votre écran est un modèle plus ancien, sachez qu'il existe aussi des adaptateurs HDMI-DVI pour convertir le signal envoyé sur un port DVI.

Voyons maintenant les préparatifs nécessaires en vue du raccordement d'une carte Arduino Uno à un Raspberry Pi afin que les deux systèmes puissent échanger des informations.

Installation de l'IDE Arduino sur le Raspberry Pi

Vous pouvez évidemment continuer à programmer votre carte Arduino Uno dans votre environnement de développement habituel depuis votre ordinateur sous Windows, Mac ou Linux. Mais, comme tôt ou tard, nous allons raccorder les deux cartes, je vais vous montrer ici comment le faire directement depuis le Raspberry Pi. En effet, qu'est-ce qui vous empêche d'utiliser l'IDE Arduino sur le Raspberry Pi ?! Ouvrez une fenêtre de terminal sur le Raspberry Pi et saisissez les deux lignes suivantes :

```
# sudo apt-get update  
# sudo apt-get install arduino
```

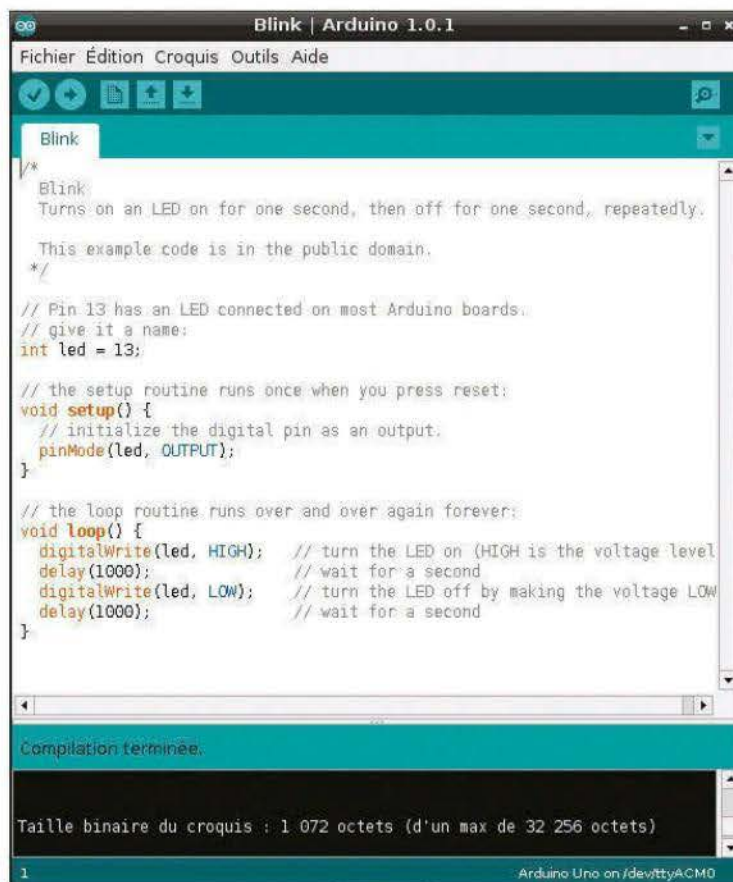
Pour l'installation, nous utiliserons le gestionnaire de paquets APT (*Advanced Packaging Tool*). La première ligne actualise les listes de paquets et la deuxième installe l'IDE Arduino. Une fois l'installation réussie, une nouvelle commande apparaît dans le menu Développement : il s'agit de l'IDE d'Arduino. Notez que la version actuelle 1.0.1 ne prend pas en charge la carte Arduino Yún.



◀ **Figure 19-2**
L'IDE Arduino dans le menu
Développement

Lorsque vous démarrez l'IDE à l'aide de cette commande, la fenêtre bien connue s'ouvre au bout de quelques instants. Vous devez encore choisir le port série correct. Sous Linux, il s'agit de `/dev/ttyACM0` pour la carte Uno. Les modèles plus anciens utilisent `/dev/ttyUSB0`.

Figure 19-3 ►
Version 1.0.1 de l'environnement
de développement

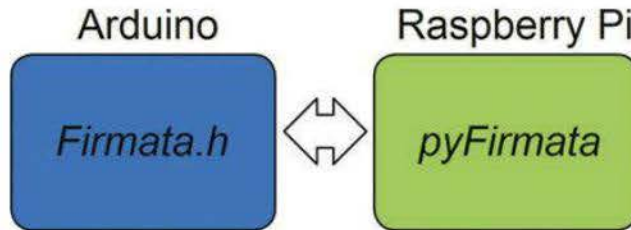


Vous pouvez vérifier que tout a bien fonctionné en lançant le sketch Blink que j'ai également chargé. Revenons-en à la communication entre l'Arduino et le Raspberry Pi. Nous allons utiliser ici Firmata, un protocole de communication entre ordinateurs ou programmes. Voyons maintenant comment établir une liaison entre l'Arduino et le Raspberry Pi afin que les deux cartes puissent échanger des données.

Firmata

Côté Arduino, nous disposons déjà de tout le nécessaire. Firmata fait partie de l'environnement de développement Arduino et il suffit de le charger en tant que firmware sous la forme d'un sketch. Côté Raspberry Pi, ce n'est pas tout à fait la même chose. Le langage habituel du Raspberry Pi, Python, fait déjà partie de la distribution Linux Debian Wheezy que je recommande aux débutants. Il nous faut donc

uniquement installer Firmata. Le paquet Python correspondant se nomme pyFirmata.



◀ **Figure 19-4**
Firmata sur l'Arduino
et le Raspberry Pi

Comme Firmata fait déjà partie de l'environnement de développement Arduino, il suffit de s'assurer de son bon fonctionnement en ajoutant le fichier d'en-tête `Firmata.h`. Côté Raspberry Pi, nous avons quelques bricoles à installer. Je vais vous présenter deux méthodes employant deux gestionnaires de paquets différents.

Méthode 1 : avec Mercurial

Exécutez les commandes suivantes dans le terminal afin d'installer pySerial, pour la communication série sous Python, et Mercurial, qui est un outil de gestion de versions :

```
# sudo apt-get install python-serial mercurial
```

Vous pouvez maintenant télécharger pySerial et l'installer sous Python en saisissant les lignes suivantes dans le terminal :

```
# hg clone https://bitbucket.org/tino/pyfirmata
# cd pyfirmata
# sudo python setup.py install
```

La première ligne de commande charge les sources requises depuis l'adresse saisie sur le Raspberry Pi. Après le téléchargement, elles se trouvent dans le dossier `pyfirmata`. La commande `cd` de la deuxième ligne permet d'accéder au dossier et la troisième ligne démarre l'installation de pyFirmata sous Python. Lorsque l'installation est terminée, il n'est pas nécessaire de conserver les sources et vous pouvez donc les supprimer à l'aide des lignes suivantes :

```
# cd ..
# sudo rm -r pyfirmata
```

Méthode 2 : avec GitHub

Avant toute chose, si ce n'est pas encore fait, vous devez installer Git (autre logiciel de gestion de versions décentralisé) pour pouvoir télé-

charger les sources de GitHub sur le Raspberry Pi. Pour ce faire, saisissez la commande suivante :

```
# sudo apt-get install git
```

Ensuite, vous pouvez installer pyFirmata à l'aide des lignes suivantes :

```
# git clone https://github.com/tino/pyFirmata.git
# cd pyFirmata
# sudo python setup.py install
```

Comme avec la première méthode, vous pouvez ici aussi supprimer les sources dans le dossier pyFirmata.

Tout est maintenant prêt pour procéder au premier essai. Nous allons dire bonjour à Arduino et nous ferons clignoter sa LED 13. Comment allons-nous faire ?

Préparation de l'Arduino

Vous devez évidemment munir l'Arduino du firmware de Firmata en chargeant le sketch correspondant et en le téléversant sur la carte. Activez la commande *Fichiers>Exemples>Firmata>StandardFirmata* pour ouvrir le sketch dans l'environnement de développement, puis téléversez-le sur la carte Arduino. Vous n'avez rien de plus à faire pour l'instant.

Préparations du Raspberry Pi

Comme nous travaillerons avec le langage de programmation Python sur le Raspberry Pi, il est conseillé d'installer un environnement de développement Python. Actuellement, j'utilise SPE (*Stani's Python Editor*) que vous pouvez installer à l'aide des lignes suivantes :

```
# sudo apt-get update
# sudo apt-get install spe
```

Toutefois, cet environnement de développement ralentit le Raspberry Pi, comme vous pourrez le constater si vous jetez un œil à l'indicateur de performances du processeur dans la barre des tâches. Au lieu de SPE, vous pouvez aussi utiliser le simple éditeur de texte Nano qui s'installe à l'aide de la ligne de commande suivante :

```
# sudo apt-get install nano
```

Pour ouvrir l'éditeur, il suffit ensuite de saisir la commande suivante dans le terminal :

```
# nano
```

Nous allons néanmoins continuer avec SPE. Après son installation, vous le trouverez dans le dossier Développement du menu Démarrer de Linux.



◀ **Figure 19-5**
Ouverture de Stani's Python Editor

Vous pouvez saisir le code suivant dans l'éditeur pour faire clignoter la LED 13. Ne vous inquiétez pas, nous n'en resterons pas là ! Nous ferons aussi des choses un peu plus compliquées. Cet exercice nous permet simplement de nous mettre en jambe :

```
1  #!/usr/bin/python
2
3  from time import sleep          # importation de la fonction sleep
4  from pyfirmata import Arduino, util # importation de la fonction Arduino, util
5
6  # communication avec la carte Arduino via le port série
7  arduinoboard = Arduino('/dev/ttyACM0')
8
9  # programmation de la broche Arduino
10 pin13 = arduinoboard.get_pin('d:13:o')
11 while True:
12     pin13.write(1) # LED allumée
13     sleep(1)      # pause 1 seconde
14     pin13.write(0) # LED éteinte
15     sleep(1)      # pause 1 seconde
```

Examinons la signification des différentes lignes du code :

Ligne 1

Afin que le script Python soit correctement détecté par l'interpréteur, la première ligne commence sur les systèmes Linux par la séquence de caractères `#!` suivie du chemin d'accès absolu à l'incontournable interpréteur.

```
#!/usr/bin/python
```

Cette ligne se nomme le *shebang*. L'emplacement de l'interpréteur Python varie selon les systèmes Linux. Par conséquent, la ligne telle qu'elle est présentée ici n'est pas universellement valable. Il existe une meilleure méthode qui utilise le programme `env`.

```
#!/usr/bin/env python
```

Le code `env` charge les variables d'environnement standard de la configuration du système d'exploitation qui prend aussi en charge la variable d'environnement *PATH*. Sur mon ordinateur, l'interpréteur Python se trouve sous `/usr/bin/python`.

Ligne 3

Comme nous avons besoin de la fonction `sleep` pour insérer une pause, celle-ci est importée à la ligne 3 par l'instruction `from/import`.

Ligne 4

Pour nous permettre d'utiliser la multitude de fonctions de `pyFirmata` après son installation, le paquet est inséré dans le code à la ligne 4 par l'instruction `from/import`.

Ligne 7

Comme nous voulons communiquer et contrôler la carte Arduino via le port série, nous devons nous y connecter. Il s'agit du même *device* (appareil) `/dev/ttyACM0` que celui déjà utilisé pour programmer la carte Arduino. Pour que Python puisse avoir accès au port série, nous avons précédemment installé le paquet *python-serial* pendant la phase préparatoire.

```
arduinoboard =Arduino('/dev/ttyACM0')
```

Par cette ligne, `pyFirmata` crée une instance de *pyfirmata* que nous avons nommée `arduinoboard`. Comme argument, nous transmettons précisément le nom d'appareil que je viens d'indiquer.

Ligne 10

Les différentes broches de la carte peuvent être contrôlées ou configurées par la méthode `get_pin`. La configuration s'effectue par le biais d'une séquence de caractères au format suivant :

```
(a|d:<PinNr>:i|o|p|s)
```

Les trois informations nécessaires sont énumérées en étant séparées par des deux-points. En voici la signification :

- L'instruction est-elle destinée à une broche analogique (*a*) ou numérique (*d*) ?
- De quelle broche s'agit-il (*PinNr*) ?
- Quel mode faut-il utiliser (*i* : entrée, *o* : sortie, *p* : MLI, *s* : servo) ?

C'est donc ce que nous faisons à la ligne 10.

```
pin13 = arduinoboard.get_pin('d:13:o')
```

Une variable intitulée `pin13` est initialisée à l'aide de la méthode `get_pin` afin de nous permettre de manipuler cette broche (numérique, 13, sortie) conformément aux instructions transmises aux lignes 11 à 15.

Lignes 11 à 15

Une boucle `while` permet d'exécuter en continu les instructions énoncées dans le corps de la boucle :

```
while True:
    pin13.write(1) # LED allumée
    sleep(1) # pause 1 seconde
    pin13.write(0) # LED éteinte
    sleep(1) # pause 1 seconde
```

Pour associer différents niveaux à la LED qui est connectée à la broche 13, nous utilisons la méthode `write` avec l'argument 1 pour le niveau HIGH ou 0 pour le niveau LOW. Entre les deux, nous plaçons la fonction `sleep` qui interrompt l'exécution du programme pendant une durée d'une seconde. La boucle `while` est exécutée tant que l'instruction suivante est vraie (`True`). Nous n'avons pas employé de variable comme instruction, mais la constante `True`, ce qui signifie que la boucle est exécutée à l'infini ou jusqu'à ce que l'utilisateur interrompe manuellement l'exécution du script à l'aide du raccourci *Ctrl* + *C*. Si vous avez démarré le script depuis le Raspberry Pi, observez la LED RX de la carte Arduino. Elle s'allume et s'éteint avec un intervalle d'une seconde. On peut donc en conclure que des

informations sont transmises depuis le Raspberry Pi à l'Arduino via le port série avec le même intervalle pour contrôler la LED.

Commande par MLI

Nous allons maintenant voir comment commander une LED raccordée à l'une des broches MLI à l'aide d'un signal MLI (voir la section « Que signifie MLI ? » du chapitre 10). Je voudrais ouvrir une interface graphique sur le Raspberry Pi afin d'envoyer le signal MLI à la carte Arduino à l'aide d'un potentiomètre linéaire comme celui illustré ci-après.



Le potentiomètre linéaire permet de choisir des valeurs comprises entre 0 et 100 qui correspondent aux pourcentages de MLI. Attention toutefois, car la fonction `write` de `pyFirmata`, qui s'occupe de générer le signal MLI, accepte des valeurs comprises entre 0,0 et 1,0. J'ai donc intentionnellement employé des valeurs à virgule flottante, car la fonction attend des valeurs de type *float*.

Examinons le script Python de plus près. Python ne peut afficher de but en blanc des éléments graphiques, comme des boutons, des étiquettes, des potentiomètres linéaires ou autres. Pour ce faire, nous utilisons une bibliothèque nommée *Tkinter*. Qu'est-ce donc ? Il s'agit de la première boîte à outils d'interface graphique pour Python. Elle permet de créer sous Python des programmes ayant une interface graphique. La boîte à outils Tk a initialement été développée pour le langage Tcl (*Tool Command Language*), mais entre-temps, elle a pris place dans la bibliothèque standard de Python. Le module *Tkinter* (*Tk-Interface*) permet à l'utilisateur de programmer très facilement des applications Tk sans devoir installer au préalable des logiciels ou des bibliothèques supplémentaires. Examinons le script Python que j'ai divisé en blocs pour une meilleure lisibilité.

Initialisation

Pendant la phase d'initialisation, nous importerons aussi bien `pyfirmata` que `Tkinter`. Nous utilisons encore `arduinoboard`, comme dans l'exemple précédent. La broche 3, sur laquelle la diode est

raccordée, doit être initialisée en tant que sortie MLI, ce que nous faisons à l'aide du mode p.

```
1  #!/usr/bin/env python
2  import pyfirmata
3  from Tkinter import *
4
5  arduinoboard = pyfirmata.Arduino('/dev/ttyACM0')
6
7  pin3 = arduinoboard.get_pin('d:3:p') # sortie MLI sur la broche 3
```

Fonctions requises

Les fonctions `cleanup` et `setPWM` sont utilisées lors de l'exécution du script – on s'en serait douté. La fonction `cleanup` est exécutée au moment où vous fermez l'interface graphique par un clic sur la croix dans le coin supérieur droit. Elle fait en sorte que la LED soit éteinte au moyen de la fonction `write`. La fonction `setPWM` commande la LED et elle est exécutée tant que vous modifiez la position du curseur du potentiomètre.

```
9  def cleanup():
10     # LED 3 éteinte
11     pin3.write(0)
12     arduinoboard.exit()
13
14  def setPWM(pwm):
15     # commande par MLI de la LED 3
16     # accepte des valeurs comprises entre 0 et 1
17     pin3.write(float(pwm)/100.0)
```

La fonction `write` de la commande par MLI attend des valeurs comprises entre 0 et 1, ce qui signifie que l'argument doit être de type *float*. Comme le potentiomètre transmettra par la suite, en réponse au déplacement du curseur, une valeur du paramètre `pwm` comprise entre 0 et 100, cette valeur doit donc être divisée par 100.

Préparation de l'interface graphique GUI et du potentiomètre linéaire

L'interface graphique est initialisée à la ligne 20, qui prépare plus précisément l'instance `master` de la boîte de dialogue `wm_protocol`, ferme la fenêtre en effaçant son contenu, opération qui s'achève par l'exécution de la fonction `cleanup` qui éteint la diode. À la ligne 22, `wm_title` permet de nommer l'application, nom qui sera affiché dans la barre de titre. Le potentiomètre est initialisé avec les valeurs correspondantes et son exécution démarre à la ligne 24. `from_` et `to` définissent la plage de valeurs transmises par le potentiomètre. Quand le

curseur est actionné, il faut afficher immédiatement le résultat, ce qui est assuré par l'instruction `command` suivie de la fonction exécutée. L'orientation est définie par `orient` ; ici, elle est horizontale. Ensuite, nous précisons encore la longueur (`length`) et nous affichons le nom du potentiomètre au moyen d'une étiquette (`label`).

```
19 # GUI
20 master = Tk()
21 master.wm_protocol("WM_DELETE_WINDOW", cleanup)
22 master.wm_title('PWM_control')
23
24 # initialisation du potentiomètre
25 scale = Scale(master,
26               from_ = 0,
27               to = 100,
28               command = setPWM,
29               orient = HORIZONTAL,
30               length = 400,
31               label = 'PWM-Value')
```

Démarrage du programme

Le gestionnaire d'interface se sert de `pack` pour centrer le potentiomètre à l'intérieur de la boîte de dialogue. L'activation de `mainloop` affiche la boîte de dialogue et maintient le programme dans une boucle sans fin, en attendant que se produisent des événements intéressants comme le déplacement du curseur qui déclenche l'action programmée.

```
33 scale.pack(anchor = CENTER) # centré
34 master.mainloop()          # démarrage de TK Event-Loop
```

Il n'y a pas grand-chose d'autre à ajouter sur ce script assez simple. Tkinter est bien plus performant que ce que j'ai pu montrer dans ces quelques pages et je ne peux que vous conseiller la lecture d'ouvrages spécialisés ou la consultation de ressources sur Internet.

Comme vous vous êtes maintenant familiarisé avec le fonctionnement d'une broche MLI, vous allez pouvoir commander un servomoteur au moyen d'un potentiomètre.



Attention !

Si vous utilisez la version 3 de Python, vérifiez que vous écrivez bien `tkinter` (avec un `t` minuscule) lors de son importation. La bibliothèque a été renommée.

Commande d'un servomoteur

Nous avons vu précédemment comment commander un servomoteur. Ici, vous apprendrez à régler précisément l'angle du servomoteur à l'aide d'un potentiomètre.

Pour en savoir davantage sur le brochage d'un servomoteur, je vous invite à relire le montage n° 14 « Le moteur pas-à-pas ». J'aimerais commander le moteur au moyen de la broche MLI 3, mais il est aussi possible d'utiliser une broche qui ne soit pas dotée de cette fonctionnalité de modulation, comme la broche 7. Examinons le code Python qui est très proche de celui de l'exemple précédent. Comme vous pourrez le constater, une fois que l'on a compris le principe de base, il suffit souvent de modifier un détail pour accéder à d'autres fonctionnalités.

```
1  #!/usr/bin/env python
2  import pyfirmata
3  from Tkinter import *
4
5  arduinoboard = pyfirmata.Arduino('/dev/ttyACM0')
6
7  it = pyfirmata.util.Iterator(arduinoboard)
8  it.start()
9
10 pin3 = arduinoboard.get_pin('d:3:s') # servo sur la broche 3
```

La seule différence dans ce segment de code réside dans le changement de mode qui passe à s pour la commande d'un servomoteur. Les deux fonctions suivantes sont quasiment inchangées.

```
12 def cleanup():
13     # désactivation de la broche 3
14     pin3.write(0)
15     arduinoboard.exit()
16
17 def moveServo(a):
18     # commande du servo par la broche 3
19     pin3.write(a)
```

La valeur du potentiomètre est ici aussi transmise au paramètre de la fonction moveServo pour commander le moteur à l'aide de la méthode write. L'interface du programme est créée de la même façon que précédemment et je me suis contenté d'en changer le nom.

```
21 # GUI
22 master = Tk()
23 master.wm_protocol("WM_DELETE_WINDOW", cleanup)
24 master.wm_title('Servo-Control')
```


Nous en arrivons à l'initialisation du potentiomètre qui doit transmettre des valeurs comprises entre 0 et 179 à la fonction `moveServo`. Ensuite, le script démarre avec `mainloop`.

```
26 # initialisation du potentiomètre
27 scale = Scale(master,
28     from_ = 0,
29     to = 179,
30     command = moveServo,
31     orient = HORIZONTAL,
32     length = 400,
33     label = 'Angle')
34
35 scale.pack(anchor = CENTER) # centré
36 mainloop() # démarrage de TK Event-Loop
```

Interrogation d'un bouton-poussoir

Jusqu'ici, nous avons toujours transmis des informations à la carte Arduino. Nous allons maintenant faire l'inverse en interrogeant un bouton-poussoir qui est raccordé à la broche 8. Comme procède-t-on avec `pyFirmata` ? Le code est assez évident :

```
1 #!/usr/bin/env python
2 import pyfirmata
3
4 arduinoboard = pyfirmata.Arduino('/dev/ttyACM0')
5
6 pin8 = arduinoboard.get_pin('d:8:i') # entrée sur broche 8
7
8 it = pyfirmata.util.Iterator(arduinoboard)
9 it.start()
10 pin8.enable_reporting()
11
12 while True:
13     pin8_state = pin8.read() # lecture de l'état
14     if pin8_state == True:
15         print 'bouton enfoncé'
16     if pin8_state == False:
17         print 'bouton non enfoncé'
18     arduinoboard.pass_time(0.5) # Pause
```

Ligne 6

Pour pouvoir interroger l'état d'un bouton-poussoir connecté sur une broche, nous devons évidemment la programmer en tant qu'entrée, ce que nous faisons de ce pas avec `d:8:i` (i correspond à entrée).

Lignes 8 et 9

Pour lire une broche d'entrée sous pyFirmata, vous ne pouvez pas tout simplement activer une fonction `read`, comme nous le ferons à la ligne 13. Il faut implémenter un `Iterator Thread` qui veille à ce que les broches de la carte Arduino communiquent la valeur courante lors de leur interrogation. Cela évite aussi que ne se produise un débordement de *buffer* qui bloquerait toute la communication sur le port série.

Ligne 10

Par `enable_reporting`, vous indiquez à pyFirmata que vous voulez surveiller la broche.

Lignes 12 et 13

La boucle `while` interroge l'état de la broche 8 en continu en utilisant la méthode `read`.

Lignes 14 à 17

Les instructions `if` interrogent l'état `True` qui correspond au bouton enfoncé et l'état `False` qui correspond au bouton non enfoncé, et affichent le résultat par l'instruction `print`.

Ligne 18

La méthode `pass_time` prévoit une pause d'une demi-seconde après chaque affichage.

J'ai un petit problème : au démarrage du script, rien n'indique que le bouton n'est pas enfoncé. Pourquoi le message correspondant ne s'affiche-t-il pas ?

Vous faites bien d'en parler, Ardus. Quand le bouton n'a pas encore été enfoncé, l'état correspond à `None`. Libre à vous de compléter le code afin que cet état soit aussi interrogé et qu'un message s'affiche.



Interrogation d'un port analogique

Pour finir, nous allons voir comment interroger une entrée analogique et les aspects à prendre en considération. Là encore, le script est assez évident (voir page suivante).

```

1  #!/usr/bin/env python
2  import pyfirmata
3  from time import sleep
4
5  arduinoboard = pyfirmata.Arduino('/dev/ttyACM0')
6
7  pin0 = arduinoboard.get_pin('a:0:1') # entrée analogique sur broche
8
9  it = pyfirmata.util.Iterator(arduinoboard)
10 it.start()
11 pin0.enable_reporting()
12
13 while True:
14     value = pin0.read() # lecture de la valeur analogique
15     print value         # affichage
16     sleep(1)           # pause 1 seconde

```

Ligne 7

La broche analogique 0 est désignée par un a (pour *analogique*) et elle est programmée en tant qu'entrée par le biais du mode i.

Lignes 13 à 16

La valeur analogique est lue sur la broche 0 à l'intérieur de la boucle while au moyen de la méthode read. Le résultat est compris entre 0,0 et 1,0. Ensuite, le programme marque une courte pause d'1 seconde.



Eh là, il y a un problème ! Voilà les valeurs qui sont affichées lorsque je fais lentement tourner le potentiomètre de la gauche vers la droite.

```

/home/pi/pyfirmata_analog001.py
None
0.0
0.7019
0.825
0.8563
0.914
1.0
1.0
Script stopped by user (ok).

```

Ah oui, Arduus ! Je crois avoir compris où tu voulais en venir. Au tout début de l'affichage, on peut lire None, ce qui signifie qu'aucune valeur valide n'a pu être lue. Cela se produit de temps en temps au début de l'interrogation. Voici comment l'éviter :

```

13 while True:
14     value = pin0.read() # lecture de la valeur analogique
15     if value != None:
16         print 'Valeur : %f' % value # affichage
17         sleep(1)                   # pause 1 seconde

```

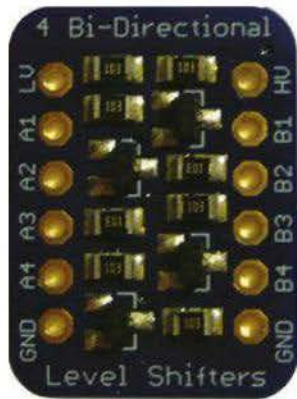
À la ligne 15, j'ai placé une instruction if qui intercepte la valeur None, le cas échéant. Pour améliorer la présentation, vous pouvez aussi

précéder l’affichage d’un texte ou de la mention du type de données, comme à la ligne 16. Vous obtenez alors le code suivant :

```
/home/pi/pyfirmata_analog002.py
Value : 0.000000
Value : 0.012700
Value : 0.326500
Value : 0.770300
Value : 0.849500
Value : 1.000000
Script stopped by user (ok).
```

Liaison série entre le Raspberry Pi et l’Arduino

Jusqu’ici, les informations échangées entre le Raspberry Pi et l’Arduino ont transité par la liaison USB qui a servi à l’acheminement des données série. Mais il est aussi possible d’établir directement une liaison TTL série à condition de respecter la précaution suivante : le Raspberry Pi fonctionne avec une tension d’alimentation maximale sur ses entrées et sorties GPIO de 3,3 V, tandis que l’Arduino utilise une tension de 5 V. Si vous raccordez les deux cartes l’une à l’autre sans prendre de précautions, c’est la loi du plus fort qui prévaut et l’un des protagonistes restera sur le carreau. Le Raspberry Pi recevra une tension trop élevée. Et comme ses broches GPIO sont directement reliées au processeur, sans protection, celui-ci grillera et la carte sera bonne à jeter. Ce n’est donc pas une bonne idée ! Comment l’éviter ? Nous pourrions utiliser un diviseur de tension afin de nous assurer que le Raspberry Pi reçoive bien une tension de 3,3 V. J’ai préféré employer un composant meilleur marché qui s’appelle un convertisseur logique ou *levelshifter*, comme le modèle proposé par Adafruit.



◀ **Figure 19-6**
Convertisseur logique Adafruit

Ce composant permet non seulement de convertir la tension pour les liaisons RX/TX, mais aussi pour les bus I²C et SPI. Sur le côté gauche se trouvent les broches de raccordement en basse tension (LV ou *LOW Voltage*), et sur le côté droit, il y a les broches haute tension (HV ou *HIGH Voltage*). Voyons maintenant comment raccorder correctement le Raspberry Pi et l'Arduino pour que les deux cartes réussissent à communiquer via les ports série.

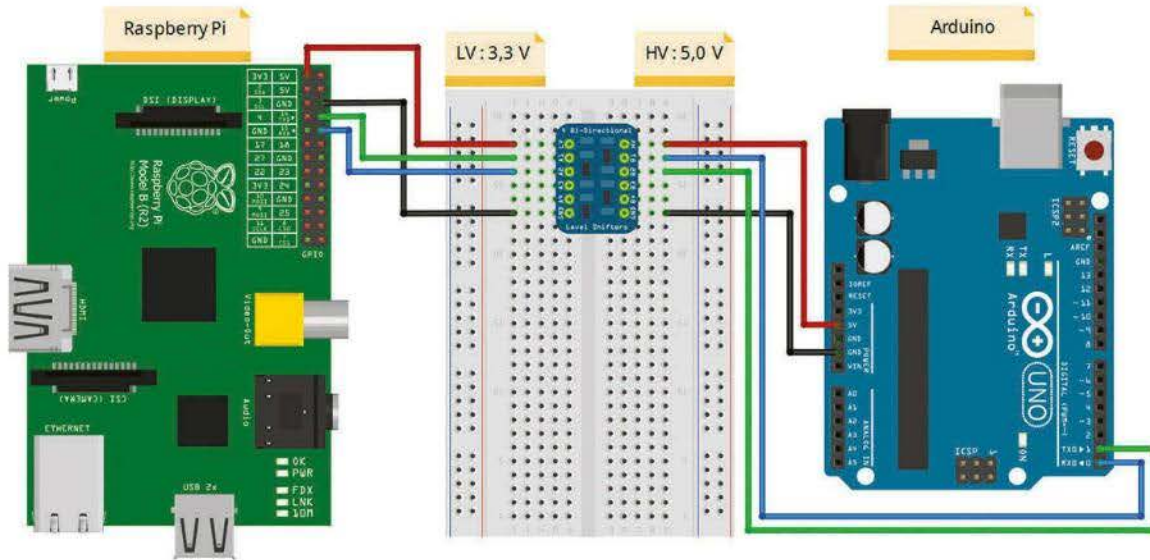
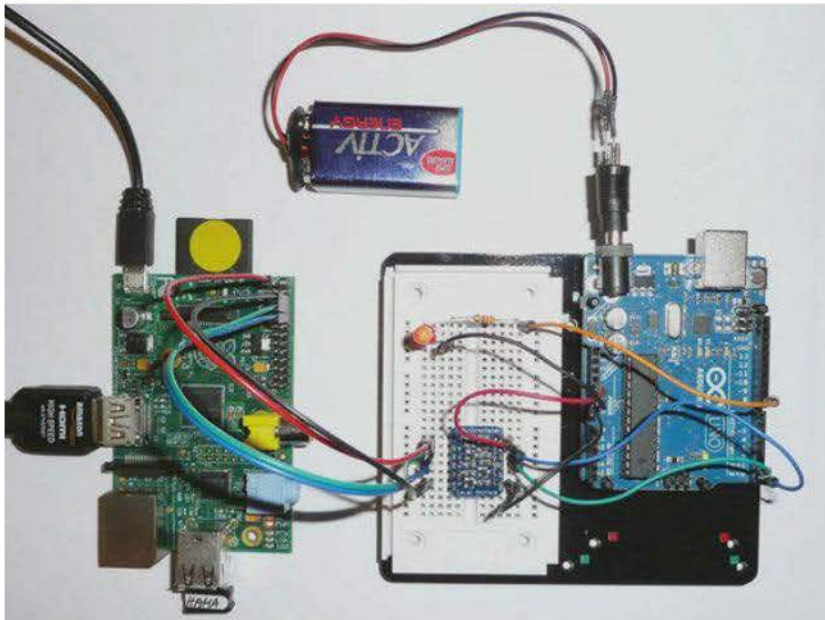


Figure 19-7 ▲
Le convertisseur logique sert d'intermédiaire entre le Raspberry Pi et l'Arduino.

Il va sans dire que les lignes émettrices et réceptrices des deux cartes doivent être croisées, car si vous voulez que la liaison TX verte, par laquelle le Raspberry Pi transmet des données, réussisse à se faire entendre par l'Arduino, cette dernière doit être connectée à sa liaison réceptrice RX bleue, et inversement. Le montage suivant permet de faire clignoter une LED connectée à la broche 8 par le biais des deux liaisons RX/TX.



◀ **Figure 19-8**
Le Raspberry Pi commande
l'Arduino via les liaisons RX/TX.

Vous pouvez constater qu'il n'y a pas de connexion USB entre les deux cartes et que la communication passe exclusivement par l'interface UART. L'alimentation électrique de l'Arduino s'effectue au moyen d'une pile de 9 V. Quelles conditions doivent être remplies pour que cette forme de communication fonctionne ? Tout d'abord, je dois vous en dire davantage sur le port série du Raspberry Pi. Par défaut, le Raspberry Pi utilise ce port en tant qu'interface de console par l'intermédiaire duquel vous pouvez accéder au système depuis l'extérieur, même si vous n'y avez pas raccordé de moniteur, de souris ou de clavier – quasi *headless*. Mais ce moyen d'accès dont nous n'avons pas absolument besoin pour le moment bloque les broches RX/TX et nous empêche de poursuivre notre expérience. Nous devons donc couper ce lien. La solution réside dans le fichier */etc/inittab* que vous pouvez ouvrir dans l'éditeur de texte Nano avec la ligne suivante :

```
# sudo nano /etc/inittab
```

Faites défiler le texte vers le bas jusqu'à l'entrée.

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Cette ligne doit être commentée de façon à ne plus pouvoir bloquer l'interface série à la prochaine réinitialisation. Pour ce faire, il vous suffit d'insérer un dièse en début de ligne :

```
# T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Ensuite, fermez l'éditeur à l'aide des raccourcis *Ctrl + X* et *J + Retour* et redémarrez le système. Il ne vous reste plus qu'à apporter quelques modifications au fameux script Blink.

```

1  #!/usr/bin/python
2
3  from time import sleep          # importation de la fonction sleep
4  from pyfirmata import Arduino, util # importation de la fonction Arduino, util
5
6  # communication avec la carte Arduino via le port série
7  arduinoboard = Arduino('/dev/ttyAMA0')
8
9  # programmation de la broche Arduino
10 pin8 = arduinoboard.get_pin('d:8:o')
11
12 while True:
13     pin8.write(1) # LED allumée
14     sleep(1)      # pause 1 seconde
15     pin8.write(0) # LED éteinte
16     sleep(1)      # pause 1 seconde

```

À première vue, le code ne semble pas avoir changé. Pourtant, il y a une différence majeure. Regardez attentivement la ligne 7 à laquelle le port série est initialisé.

Avant :

```
arduinoboard = Arduino('/dev/ttyACM0')
```

Après :

```
arduinoboard = Arduino('/dev/ttyAMA0')
```

Le *device* que nous avons utilisé précédemment se rapportait à l'interface USB. Après la correction, le *device* accède à l'interface UART.

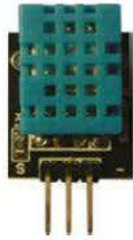
Temboo et la carte Yún – API Twitter

Nous en arrivons à un sujet passionnant – encore un –, qui est l'accès, par le biais d'une carte Yún, aux innombrables services web comme Twitter, Facebook, Google+ ou Dropbox, sans avoir à écrire des kilomètres de code. Les services que je viens d'énumérer sont très populaires et leurs noms sont aujourd'hui dans toutes les bouches. Nous allons nous y intéresser de plus près dans ce chapitre. Le projet Temboo (<https://temboo.com/>) s'est notamment donné pour vocation de permettre à nous autres, utilisateurs d'Arduino, d'accéder facilement à tous ces services Internet. Il a développé plus d'une centaine d'API (*Application Programming Interface*) qui exécutent des tâches ou offrent des services divers et variés. Je vous présenterai ici l'API Twitter.

Au sommaire :

- la structure sous-jacente à Temboo ;
- la création d'un compte Temboo ;
- la récupération d'informations, telles que des *tokens*, auprès de Twitter pour envoyer un tweet via Temboo ;
- l'exécution d'un premier test dans le navigateur ;
- le transfert du code du sketch depuis le navigateur sur la Yún pour envoyer un tweet depuis la carte ;
- la programmation d'un sketch pour mesurer l'humidité et la température au moyen d'un capteur DHT11, puis twitter les valeurs mesurées.

Composants nécessaires



1 capteur d'humidité-température DHT11



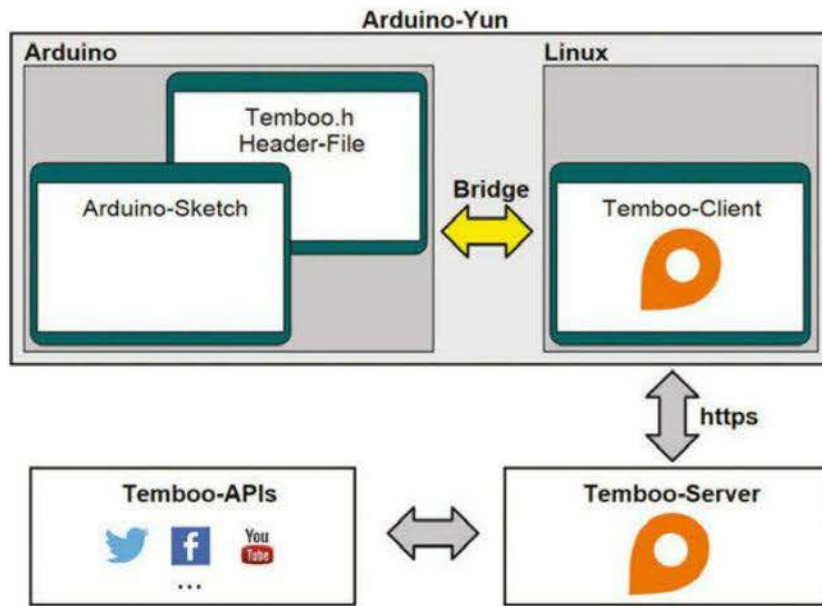
Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Temboo

Pour commencer, regardons ce qui se cache derrière le projet Temboo. Nous allons en décortiquer la structure afin de mieux comprendre son échange de données avec votre carte Yún. Je ne peux malheureusement pas entrer dans tous les détails, car l'affaire est beaucoup trop complexe. Mais je pense que mes explications vous permettront de vous initier à ce système. Si je n'ai pas répondu à toutes les questions que vous pourriez vous poser, je vous recommande de parcourir la rubrique *Get Started* du site Internet de Temboo qui propose notamment une FAQ et quantité d'informations complémentaires.

La structure

La figure 20-1 est une représentation schématisée de la communication entre une carte Yún et Temboo.



◀ **Figure 20-1**
Communication entre la carte Yún
et le serveur Temboo

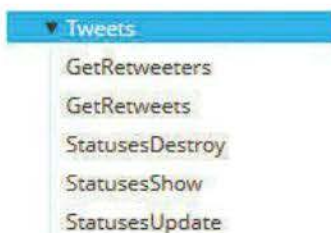
En haut à gauche sur le schéma, vous pouvez voir la carte Yún à laquelle est liée la bibliothèque requise à l'aide d'un fichier d'en-tête `Temboo.h` et dont le code proprement dit se trouve dans le fichier `temboo.cpp`. Le sketch Arduino se connecte via une passerelle (*bridge*) à Linino qui déclenche un processus initial pour exécuter le fichier `Temboo`. Ce fichier, qui se trouve dans le dossier `/usr/bin`, contient un script Python qui charge le client Temboo. Ce client transmet une requête HTTPS au serveur Temboo qui exécute des prestations de services de type *cloud*. C'est par le biais de ce serveur que l'on peut accéder aux multiples API pour profiter de leurs extraordinaires fonctionnalités. Jetez un œil au site web de Temboo et vous verrez à quel point l'offre est étendue. Cliquez sur **LIBRARY** dans la partie supérieure gauche de la fenêtre. Une arborescence apparaît alors sur le côté gauche. Elle énumère toutes les API disponibles. J'ai cliqué sur le petit triangle afin de développer la bibliothèque Twitter qui nous intéresse ici.

Figure 20-2 ▶
API Twitter de Temboo



L'arborescence présente la liste des tâches (*tasks*) disponibles pour Twitter. Chacune d'elles contient une ou plusieurs *choreos*, abréviation de *choreographies* (chorégraphies), qui sont similaires aux méthodes utilisées dans la programmation orientée objet. Ici, nous voulons poster un message sur Twitter. Donc nous choisissons l'entrée *Tweets* dans l'arborescence d'API. Comme vous le savez certainement, dans le jargon de Twitter, on ne dit pas « diffuser un message », mais *twitter*. Cette API génère un tweet qui diffuse un message sur Twitter. Vous me suivez ?! Voyons maintenant les *choreos* proposées sous la rubrique *Tweets* :

Figure 20-3 ▶
Choereos des Tweets



L'entrée intitulée *StatusesUpdate* paraît prometteuse ; nous l'utiliserons plus tard pour twitter. Passons maintenant à la pratique, car la théorie est ennuyeuse à la longue. Voici les étapes requises pour pouvoir twitter avec la carte Yún.

Création d'un compte Temboo

Il va de soi qu'un compte est nécessaire pour pouvoir utiliser Temboo. Les fonctions de base sont gratuites. Vous pouvez exécuter 250 choreos par mois et transférer jusqu'à 1024 Mo de données. Pour plus de détails, consultez la FAQ. Cliquez sur le bouton *SIGN-UP* sur la page d'accueil de Temboo, puis saisissez votre adresse électronique dans la fenêtre qui apparaît.

Get Ready to Temboo

Experience a way of writing software that lets you focus on what makes your app unique.

arduino@erik-bartmann.de

SIGN UP

◀ **Figure 20-4**

Inscription à Temboo

Quelques instants plus tard, vous devriez recevoir un courrier de confirmation intitulé *Create Your Temboo-Account*. Si votre boîte de réception reste vide, vérifiez vos spams. Vous devez cliquer sur le lien qui figure dans le message afin de terminer la création de votre compte en saisissant un identifiant et un mot de passe.

Now Let's Finish Creating Your Account

ACCOUNT NAME

Your Account Name will be used in your code to call Choreos.

EriksTembooAccount

Checking availability...

EriksTembooAccount is available!

PASSWORD

Must contain at least eight characters, one number, and one letter.

••••••••

••••••••

☒ I agree to the Temboo [Terms of Service](#).

GO!

◀ **Figure 20-5**

Création du compte Temboo

Cliquez sur le bouton *GO* pour valider la création de votre compte. Vous pouvez commencer sans plus attendre, mais je vous recommande de prendre le temps de visionner les vidéos proposées sur le site. Ces tutoriels sont en effet un bon moyen de se familiariser avec Temboo, car ils fournissent de précieux conseils. Nous allons maintenant nous intéresser à Twitter.

Votre compte Twitter

Il vous paraîtra sans doute déplacé que je vous demande si vous avez bien un compte Twitter. En effet, il vous en faut un, sinon vous aurez du mal à twitter. Pour créer un compte, rendez-vous sur le site web <https://twitter.com/signup>. Maintenant que ce point est éclairci, nous pouvons procéder à quelques réglages sur le site de Twitter.

Pour pouvoir accéder à Twitter depuis l'extérieur, vous devez préalablement créer et enregistrer une application. Ouvrez la page web <https://dev.twitter.com/apps/new>. Nommez votre application sans utiliser la séquence de caractères Twitter.

Application Details

Name: *

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description: *

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website: *

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL:

Where should we return after successfully authenticating? For @Anywhere applications, only the domain specified in the callback will be used. OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Figure 20-6 ▲
Détails de l'application Twitter

Lorsque vous aurez accepté les conditions générales d'utilisation, saisi une description, puis cliqué sur le bouton *Create Your Twitter Application*, vous verrez s'afficher une page sur laquelle vous pourrez gérer votre appli dans ses moindres détails.

EriksFirstApp

◀ **Figure 20-7**
Gestion de l'application Twitter



Quand vous voudrez poster un message sur Twitter, vous devrez vous identifier par le biais d'*OAuth* qui s'assure que personne n'essaye de se faire passer pour vous. Vous trouverez plus d'informations à ce sujet à l'adresse <https://dev.twitter.com/docs/auth/oauth>.

Avant de commencer, j'aimerais vous rappeler quelques principes de précaution élémentaires : sur Internet, on peut croiser le chemin de personnes ou d'organismes qui n'accordent pas une grande importance au respect de la vie privée. La NSA (*National Security Agency*) représente un exemple inquiétant où les droits des internautes sont foulés aux pieds. La sécurité est une question cruciale aujourd'hui et elle mérite que l'on y accorde la plus haute attention. Le simple fait de diffuser nos données de connexion sur Internet nous expose au risque potentiel de les voir récupérées par des tiers qui s'en serviront pour envoyer des courriers malveillants. Mais les risques ne s'arrêtent pas là. On pourrait aussi évoquer l'espionnage industriel qui fait perdre des millions, voire des milliards, aux entreprises qui en sont victimes. Mais si vous renoncez à communiquer vos identifiants sur Internet, vous ne pouvez pas non plus utiliser les nombreux services web. Comment résoudre ce dilemme ?

C'est là qu'*OAuth* entre en scène : il s'agit d'un protocole libre. Une application contacte un prestataire en échangeant des informations sous la forme de clés (*keys*) et de jetons (*tokens*). Mais ces données ne correspondent pas aux données de connexion de l'utilisateur. Quand nous créons une application, il s'agit dans le jargon spécialisé d'un *client* ou *consumer* qui souhaite établir une connexion avec le service web du prestataire, par exemple. Au moment de son inscription, le client reçoit une authentification en deux parties : la clé (*key*) et le *secret* qui sont des séquences de caractères générées de façon aléatoire. Lorsque des informations doivent être échangées entre l'application – c'est-à-dire le client – et le prestataire, le client envoie la clé au prestataire qui lui transmet à son tour un token provisoire. Pour un accès définitif et pour tous les autres accès, le client doit générer un jeton d'accès à partir du jeton provisoire. Ce token protège l'accès au

prestataire qui effectue une vérification en utilisant un algorithme de hachage sécurisé.

Pour que vous n'ayez pas à manipuler ces codes assez complexes, les choreos Twitter de Temboo s'occupent du processus de signature d'OAuth à votre place. Vous devez simplement disposer de l'authentification OAuth.



J'ai un peu de mal à suivre. Comment obtient-on toutes ces informations ?

Vous avez raison, Ardus. Ce n'est pas simple, mais ces procédures sont indispensables pour garantir une sécurité maximale. Je vais tout vous expliquer en détail et tout se passera bien. De nombreuses données sont nécessaires pour la création d'une authentification qui, par ailleurs, est gérée par Temboo. Les informations suivantes sont demandées :

- AccessToken
- AccessTokenSecret
- ConsumerKey
- ConsumerSecret

Comme les obtient-on ? Ces informations sont accessibles dans les données du compte Twitter. Jetez un œil à la page de gestion de l'application qui apparaît dès que vous cliquez sur le bouton *Create Your Twitter Application*. Cliquez sur le lien *Manage keys and access tokens* sous la rubrique *Application Settings* de l'onglet *Details* afin d'accéder à deux des clés requises.

Figure 20-8 ▶
Clés OAuth accessibles
dans les paramètres de l'application

The screenshot shows the 'Keys and Access Tokens' tab of a Twitter application settings page. It includes fields for 'Consumer Key (API Key)' and 'Consumer Secret (API Secret)', both masked with dots. Below these are 'Access Level' (Read-only), 'Owner' (oddecelle), and 'Owner ID' (2921). At the bottom, there are buttons for 'Regenerate Consumer Key and Secret' and 'Change App Permissions'.

Application Settings	
Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.	
Consumer Key (API Key)	ZgiQBF...
Consumer Secret (API Secret)	JTu5IKkxOgtjrPIV...
Access Level	Read-only (modify app permissions)
Owner	oddecelle
Owner ID	2921

Application Actions

Regenerate Consumer Key and Secret Change App Permissions

Vous pouvez noter la présence des clés *Consumer Key* et *Consumer Secret*. Il vous manque donc encore les jetons *AccessToken* et *AccessTokenSecret*. Lorsque vous faites défiler la page vers le bas, vous pouvez voir le bouton suivant :

Create my access token

Cela paraît très prometteur. Allez-y ! Cliquez dessus pour voir ce qu'il se produit. Un message s'affiche en haut de la fenêtre pour vous informer que les clés ont été créées. Mais où sont-elles ? Pour les voir, il suffit de faire défiler la fenêtre vers le bas :

Application Settings
Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	ZgIQI...
Consumer Secret (API Secret)	JTu5a0a...
Access Level	Read-only (modify app permissions)
Owner	oddecite
Owner ID	2...

Application Actions

Regenerate Consumer Key and Secret Change App Permissions

Your Access Token
This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	292177551... mASCK9-G2g7e...
Access Token Secret	34oUhGS1dBl...
Access Level	Read-only
Owner	oddecite

◀ **Figure 20-9**
Jetons d'accès

L'onglet *Keys and Access Tokens* réunit tout ce dont vous avez besoin pour l'authentification Temboo. Copiez les clés afin de pouvoir les saisir par la suite dans Temboo ou laissez simplement cette page ouverte. Il nous faut encore régler un petit détail afin de disposer d'un accès en écriture à Twitter. Cliquez sur l'onglet *Permissions* pour définir le type d'accès dont vous avez besoin.

Access

What type of access does your application need?

[Read more about our Application Permission Model.](#)

- ☐ Read only
- ☒ Read and Write
- ☐ Read, Write and Access direct messages

Note:

Changes to the application permission model will only reflect in access tokens obtained after the permission model change is saved. You will need to re-negotiate existing access tokens to alter the permission level associated with each of your application's users.

Update Settings

Figure 20-10 ▲
Options disponibles sous l'onglet
Permissions des paramètres de
l'application

Choisissez l'option *Read and Write*, puis revenez sous l'onglet *Settings* pour cocher l'option *Allow this application to be used to Sign in with Twitter*. Vous devez cliquer sur le bouton suivant afin d'appliquer les modifications effectuées sous chaque onglet :

Update this Twitter application's settings

Patiencez quelques secondes, le temps que les modifications soient appliquées. Et voilà ! Tout est maintenant prêt du côté de Twitter. Nous pouvons donc revenir à Temboo.

De retour dans Temboo

Comme je l'ai déjà expliqué, nous devons saisir notre authentification afin de montrer patte blanche auprès de Temboo. Sélectionnez l'API Twitter dans la bibliothèque, puis cliquez sur la méthode *Statuses-Update* sous *Library>Twitter>Tweets*. Les champs de saisie suivants apparaissent au centre de la fenêtre du navigateur (voir figure 20-11).

Library | Twitter | Tweets | StaturesUpdate

IoT Mode OFF

StaturesUpdate

Allows you to update your Twitter status (aka Tweet).

INPUT

AccessToken
The Access Token provided by Twitter or retrieved during the OAuth process.

AccessTokenSecret
The Access Token Secret provided by Twitter or retrieved during the OAuth process.

ConsumerKey
The API Key (or Consumer Key) provided by Twitter.

ConsumerSecret
The API Secret (or Consumer Secret) provided by Twitter.

StatusUpdate
The text for your status update. 140-character limit.

► **OPTIONAL INPUT**

Run

Save Profile

Select Profile ▼

▼ OUTPUT

Response
The response from Twitter.

◀ **Figure 20-11**
Saisie de l'accréditation pour l'API
Twitter

Pour vous éviter d'avoir à saisir à plusieurs reprises les quatre clés que vous venez de générer, Temboo vous propose de les enregistrer. Saisissez les clés dans les champs correspondants, puis cliquez sur *Save Profile*.

Library | Twitter | Tweets | StaturesUpdate

IoT Mode OFF

StaturesUpdate

Allows you to update your Twitter status (aka Tweet).

INPUT

AccessToken
The Access Token provided by Twitter or retrieved during the OAuth process.

AccessTokenSecret
The Access Token Secret provided by Twitter or retrieved during the OAuth process.

ConsumerKey
The API Key (or Consumer Key) provided by Twitter.

ConsumerSecret
The API Secret (or Consumer Secret) provided by Twitter.

StatusUpdate
The text for your status update. 140-character limit.

► **OPTIONAL INPUT**

Run

Save Profile

Select Profile ▼

▼ OUTPUT

Response
The response from Twitter.

◀ **Figure 20-12**
Profil enregistré

Nommez l'accréditation dans le champ qui apparaît en haut de la colonne de droite, puis cliquez sur le bouton *Save*. Une entrée portant le nom que vous venez de saisir apparaît dans le menu déroulant *Select Profile*.

Figure 20-13 ►
Entrée du menu Select Profile



Il est maintenant temps de reprendre l'accréditation dans la fenêtre de la méthode *StatusesUpdate*, l'objectif étant évidemment de s'identifier auprès de Twitter. Cliquez sur le menu *Select Profile* qui se trouve en haut à droite de la fenêtre. La liste des entrées disponibles apparaît. Pour l'instant, il n'y en a qu'une, portant le nom que vous venez de saisir pour votre profil. Lorsque vous cliquez dessus, toutes les clés sont reprises dans les champs correspondants.

Figure 20-14 ►
Reprise de l'accréditation
pour l'API Twitter

A screenshot of the Twitter API 'StatusesUpdate' form. The form has a title 'StatusesUpdate' with a star icon and a subtitle 'Allows you to update your Twitter status (aka Tweet)'. Below the title is an 'INPUT' section with several fields: 'AccessToken', 'AccessTokenSecret', 'ConsumerKey', 'ConsumerSecret', and 'StatusUpdate'. Each field has a description and a text input area. To the right of the form is a dropdown menu titled 'Select Profile' with the option 'EriksTwitter'.

Un champ n'est pas renseigné. Il s'agit du champ intitulé *StatusUpdate*.

C'est justement dans ce champ que vous devrez saisir le message qui sera affiché dans Twitter. Saisissez-y un texte de 140 caractères au maximum. Ensuite, vous pouvez vérifier directement dans votre navigateur que l'accès à Twitter a bien été établi. Voici comment procéder.

ConsumerSecret
The API Secret (or Consumer Secret) provided by Twitter.
12uziMcmW2SmELwqW3c

StatusUpdate
The text for your status update. 140-character limit.
Hello, this is a message from Erik !

► **OPTIONAL INPUT**

Run

▼ **OUTPUT**

Response
The response from Twitter.

◀ **Figure 20-15**
Message envoyé sur Twitter
et sa réponse

▼ **OUTPUT** Successful run at 08:20 ET

Response
The response from Twitter.

```
{ "created_at": "Thu Feb 06 13:20:57 +0000 2014", "id": "4201234567890123456", "id_str": "4201234567890123456" }
```

COPY

J'ai saisi un message pour le nouveau tweet dans le champ de texte, puis j'ai cliqué sur *Run*. Quelques secondes plus tard, la réponse est apparue en utilisant le format JSON en caractères verts sous la rubrique *OUTPUT*. Ouvrez Twitter. Votre message devrait y apparaître tôt ou tard. Si vous effectuez une série de tests avec Temboo alors que vous ne possédez pas de compte Twitter dédié aux tests, je vous recommande d'effacer les messages immédiatement après leur diffusion. Si vous rencontrez des problèmes lors de l'authentification auprès de Twitter, générez de nouvelles clés et de nouveaux jetons via les boutons *Regenerate* sous l'onglet *Keys and Access Tokens* des paramètres de votre application Twitter. Cette méthode m'a souvent permis de me tirer d'affaire. Dans le cas d'un problème, le message d'erreur a l'apparence suivante :

▼ **OUTPUT** Error at 08:49 ET

A HTTP Error has occurred: The remote server responded with a status code of 401. Typically this indicates that an authorization error occurred while attempting to access the remote resource. The data returned from the remote server was: {"errors":[{"message":"Could not authenticate you","code":32}]} The error occurred in the HTTPSend (Twitter - update status) step.

◀ **Figure 20-16**
Problème d'authentification
de l'API Twitter

Au tour de la Yún

Il est maintenant temps de tout faire fonctionner avec la carte Yún. Le code correspondant est fourni sur le site web de Temboo. Faites défiler la page vers le bas jusqu'à la rubrique *Code*.

Figure 20-17 ►
Exemple de code de l'API Twitter

▼ CODE [Download](#)

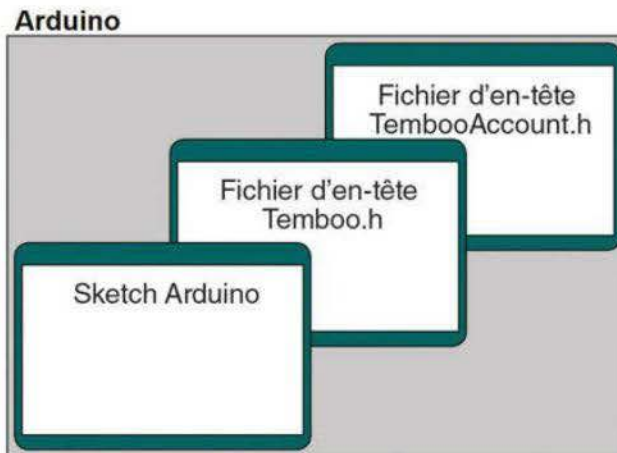
```
#include <Bridge.h>
#include <Temboo.h>
#include "TembooAccount.h" // contains Temboo account information, as described below

int numRuns = 1; // Execution count, so this doesn't run forever
int maxRuns = 10; // Maximum number of times the Choreo should be executed

void setup() {
```

J'ai déjà choisi la plateforme de destination appropriée, c'est-à-dire la carte Arduino Yún, dans la liste déroulante qui apparaît au milieu en haut lorsque l'on active l'*IoT Mode*. Temboo peut aussi générer un code adapté à d'autres plateformes, comme Java, Python ou Processing : il se montre donc d'une grande souplesse. Au début de ce montage, j'avais présenté une illustration où l'on pouvait voir que deux fichiers étaient nécessaires du côté Arduino. Ce n'est pas tout à fait vrai, car nous avons aussi besoin d'un fichier d'en-tête ayant pour rôle de fournir les informations du compte Temboo.

Figure 20-18 ►
Les trois fichiers requis côté Arduino



Il s'agit d'une partie du sketch proprement dit et du fichier d'en-tête `Temboo.h` qui permet au fichier `Temboo.cpp` d'assurer la communication avec la partie Linux.

D'autre part, vous avez aussi besoin du fichier d'en-tête contenant les données du compte : il s'agit du fichier `TembooAccount.h` dont l'initialisation s'effectue comme illustré ci-après. Ce fichier est également disponible sur Internet.

▼ HEADER FILE

```

/*
IMPORTANT NOTE about TembooAccount.h

TembooAccount.h contains your Temboo account information and must be included
alongside your sketch. To do so, make a new tab in Arduino, call it TembooAccount.h,
and copy this content into it.
*/

#define TEMBOO_ACCOUNT "erikbartmann" // Your Temboo account name
#define TEMBOO_APP_KEY_NAME "myFirstApp" // Your Temboo app key name
#define TEMBOO_APP_KEY "F" // Your Temboo app key

/*
The same TembooAccount.h file settings can be used for all Temboo SDK sketches.
Keeping your account information in a separate file means you can share the
main .ino file without worrying that you forgot to delete your credentials.
*/

```

◀ **Figure 20-19**

Fichier `TembooAccount.h`

Toutes les conditions préalables sont maintenant remplies et nous allons donc pouvoir créer notre sketch Arduino et l'exécuter. Le nouveau sketch se nomme *EriksTwitterUpdate*. J'ai ajouté le code du fichier d'en-tête `TembooAccount.h` dans l'environnement de développement Arduino après avoir cliqué sur l'icône *Nouveau* pour créer un nouvel onglet.

```

EriksTwitterUpdate  TembooAccount.h
1  #include <Bridge.h>
2  #include <Temboo.h>
3  #include "TembooAccount.h" // contains Temboo account information, as described below
4
5  int numRuns = 1; // Execution count, so this doesn't run forever
6  int maxRuns = 10; // Maximum number of times the Choreo should be executed
7
8  void setup() {
9      Serial.begin(9600);
10
11      // For debugging, wait until the serial console is connected.
12      delay(4000);
13      while(!Serial);
14      Bridge.begin();
15  }

```

◀ **Figure 20-20**

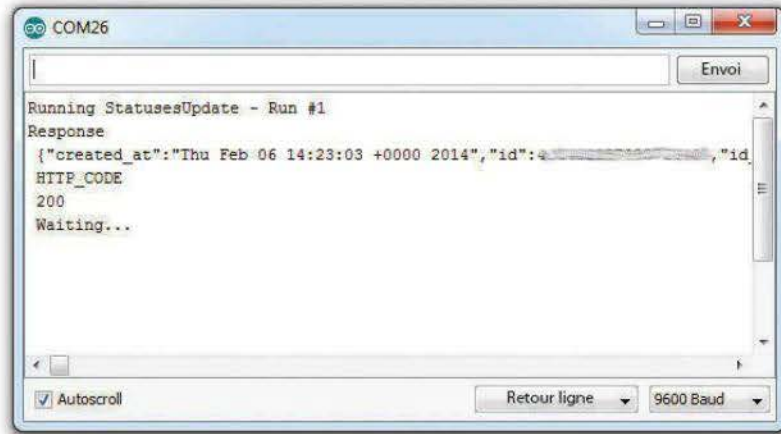
Sketch Arduino *EriksTwitterUpdate*
(extrait)

Après le téléversement via le port COM, l'exécution du sketch est interrompue, car la ligne

```
while(!Serial);
```

prévoit qu'il ne se passe rien tant que le moniteur série n'est pas ouvert. Voici ce qui apparaît dans la fenêtre après son ouverture :

Figure 20-21 ►
Affichage dans le moniteur série



Que signifie la mention HTTP CODE 200 qui est affichée dans le moniteur ? Est-ce que cela correspond à une erreur ?

Non, Ardus. C'est l'un des codes d'état HTTP qui est transmis par un serveur en réponse à une requête HTTP. La valeur 200 indique que la requête a reçu une réponse en bonne et due forme. Vous trouverez plus d'informations sur les codes HTTP à la page http://fr.wikipedia.org/wiki/Liste_des_codes_HTTP, par exemple.

Comme vous pouvez le constater, le tweet est bien arrivé sur ma page Twitter :

Figure 20-22 ►
Le tweet a bien été publié.

Tweets



Attention !

Lorsque vous programmez un sketch qui renvoie toujours le même tweet à brefs intervalles, une alarme retentit chez Twitter et le tweet suivant est bloqué. Si vous voulez tout de même afficher régulièrement un message à brefs intervalles – au risque d'irriter vos *followers* –, ajoutez un horodatage à votre tweet, par exemple.

Vous avez dit que ce code n'est pas très important, mais le fonctionnement de ce sketch m'intéresse tout de même. Peut-être aurai-je besoin de le modifier, qui sait ? Comment faire ?



Vous avez parfaitement raison, Ardus ! Examinons le sketch de plus près.

Déclaration globale

```
#include <Bridge.h>
#include <Temboo.h>
#include "TembooAccount.h" // contains Temboo account information

int numRuns = 1; // Execution count, so this doesn't run forever
int maxRuns = 10; // Maximum number of times the Choreo should be executed
```

Au début figurent les trois fichiers d'en-tête mentionnés précédemment. La variable `numRuns` compte le nombre d'émissions et `maxRuns` détermine le nombre maximum de tweets. C'est une mesure de précaution, car qui aimerait trouver des milliers de tweets sur sa page du jour au lendemain ?

Initialisation

```
void setup() {
  Serial.begin(9600);
  delay(4000);
  while(!Serial);
  Bridge.begin();
}
```

La fonction `setup` initialise aussi bien l'interface série que la passerelle (*bridge*). L'exécution du sketch reprend après l'ouverture du moniteur série. Vous devez commencer à comprendre comment cela fonctionne.

Envoi des tweets

Au début de la fonction `loop`, le système vérifie d'abord, d'après la variable `numRuns`, si le nombre maximum d'envois de tweets n'est pas atteint.

```
void loop() {
  if (numRuns <= maxRuns) {
    Serial.println("Running StatusesUpdate - Run #" + String(numRuns++));
    TembooChoreo StatusesUpdateChoreo;
    // Invoke the Temboo client
    StatusesUpdateChoreo.begin();
```


Le nombre d'exécutions est indiqué par la méthode `println` au moniteur série. Pour pouvoir utiliser les fonctionnalités de Temboo, il faut d'abord créer l'objet correspondant. On utilise à cette fin une instance de la catégorie `TembooChoreo`. À votre avis, où trouve-t-on cette définition de catégorie ? Essayez de la trouver. La méthode `begin` initialise l'objet en établissant une liaison avec Linino et en y activant le fichier Python `temboo`. Les trois lignes ou méthodes suivantes permettent de communiquer l'authentification du compte Temboo que nous avons insérée dans le fichier `TembooAccount.h` :

```
// Set Temboo account credentials
StatusesUpdateChoreo.setAccountName(TEMBOO_ACCOUNT);
StatusesUpdateChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
StatusesUpdateChoreo.setAppKey(TEMBOO_APP_KEY);
```



Mais au fait, où sont utilisées les informations relatives aux clés générées par Twitter ? Ne faut-il pas aussi les insérer ici dans le code du sketch ?

Votre question tombe à pic ! Les clés sont bien communiquées, mais pas depuis le sketch. Seul le service web de Temboo connaît ces clés que nous venons d'enregistrer depuis la page Internet. En revanche, pour notre part, nous connaissons le nom sous lequel nous avons enregistré les quatre clés dans Temboo. Dans notre exemple, il s'agit d'EriksTwitter. Vous pouvez ensuite communiquer ce nom à la méthode `setCredential` qui récupère l'authentification sur le serveur de Temboo :

```
// Set credential to use for execution
StatusesUpdateChoreo.setCredential("EriksTwitter");
```

Nous pouvons maintenant insérer le tweet à diffuser via la méthode `addInput` dans l'objet `StatusesUpdateChoreo`, mais cela ne veut pas dire que le tweet est bel et bien publié.

```
// Set Choreo inputs
StatusesUpdateChoreo.addInput("StatusUpdate", "Hello, this is Erik!");
```

Ensuite, la choreo est identifiée dans la hiérarchie des API par la saisie de son chemin d'accès complet :

```
// Identify the Choreo to run
StatusesUpdateChoreo.setChoreo("/Library/Twitter/Tweets/StatusesUpdate");
```

Le chemin d'accès se termine par le nom de la choreo qui s'occupe de la diffusion. Enfin, le tweet est envoyé avec la méthode `run` :

```
// Run the Choreo; when results are available, print them to serial
StatusesUpdateChoreo.run();
```

La choreo répond à l'aide de la méthode `available` et la réponse est transmise par une boucle `while` au moniteur série.

```
while(StatuesUpdateChoreo.available()) {  
  char c = StatuesUpdateChoreo.read();  
  Serial.print(c);  
}
```

La liaison est ensuite interrompue et les ressources sont remises à disposition des autres processus à l'aide de la méthode `close` :

```
StatuesUpdateChoreo.close();  
}
```

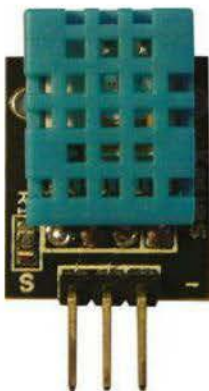
Le moniteur série affiche alors un message d'attente avant que la fonction `loop` ne déclenche le prochain envoi – sauf si le nombre maximum d'exécutions a été atteint :

```
Serial.println("Waiting...");  
delay(30000); // wait 30 seconds between StatuesUpdate calls  
}
```

C'est ainsi que se termine le sketch. Vous pouvez maintenant laisser libre cours à votre créativité. On pourrait imaginer la diffusion d'un tweet suite à l'interrogation d'un capteur pour signaler un événement particulier (la lumière est allumée, la cave est inondée, la porte du frigo est ouverte depuis plus d'une heure...). Ainsi, tous vos abonnés sauront ce qu'il se passe chez vous.

Donc, si je veux programmer mon propre sketch pour diffuser régulièrement la température qu'il fait dans mon local de travail, c'est possible ?

En effet, Ardu. Ça ne pose aucun problème. Je vous recommande d'employer un capteur spécial qui, en plus de la température, mesure aussi le taux d'humidité ambiante. Il s'agit du capteur DHT11.



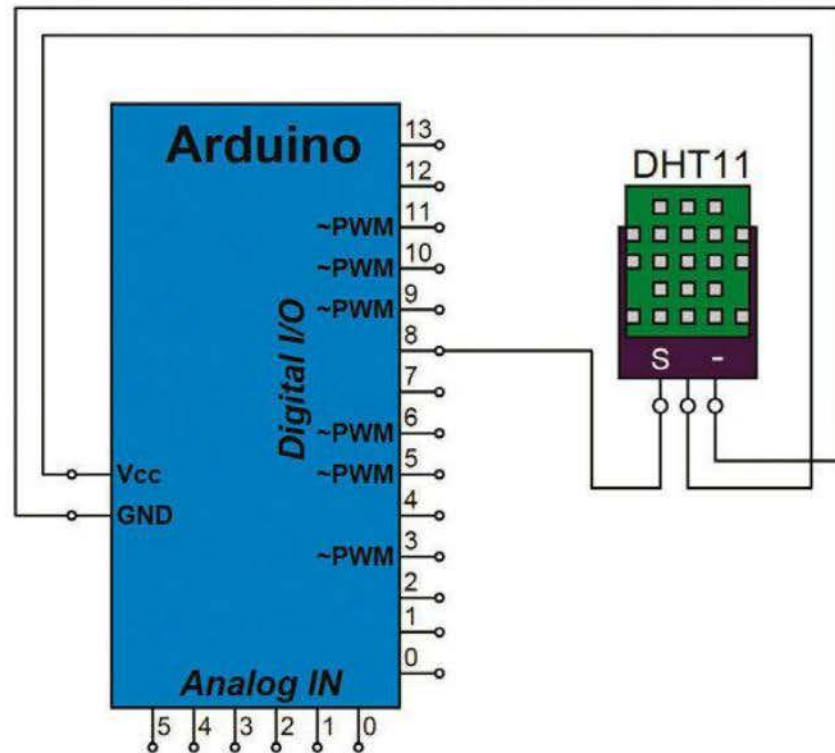
◀ **Figure 20-23**
Capteur de température
et d'humidité DHT11

Pour la lecture des valeurs de l'environnement, on peut faire appel à une bibliothèque qui est accessible à l'adresse suivante :

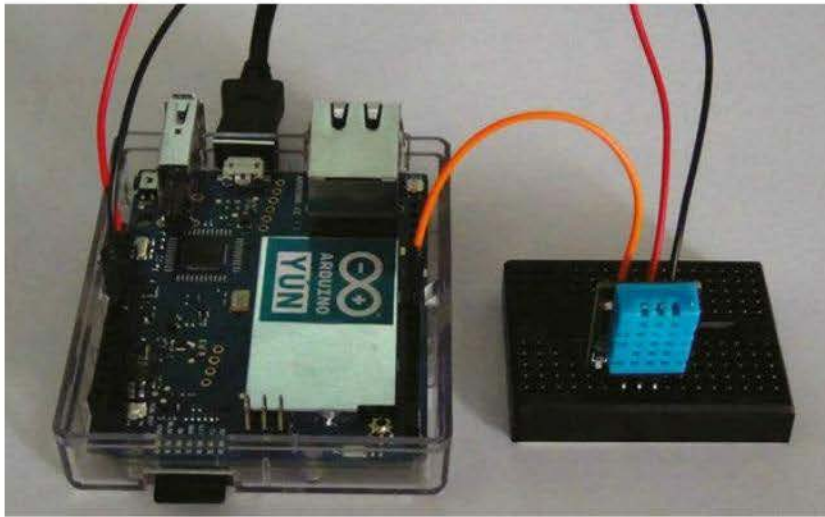
<http://playground.arduino.cc/Main/DHTLib>

Le capteur possède trois pattes de raccordement : deux pour l'alimentation électrique et la troisième désignée par la lettre S pour le transfert de données. Les pattes du modèle illustré ici n'ont que deux désignations : S (à gauche) et - (à droite). On peut donc en conclure que le + est au centre. Par conséquent, nous raccordons le capteur de la façon suivante à la carte Yún :

Figure 20-24 ►
Raccordement du capteur
de température et d'humidité
DHT11



Certes, le schéma de montage montre une carte Arduino Uno et non une Yún, mais le capteur se connecte exactement de la même façon aux broches de cette dernière. Le montage est extrêmement simple et ne demande pas de constructions compliquées.



◀ **Figure 20-25**
Construction du circuit avec la carte Yún et le capteur de température et d'humidité DHT11

Un tweet doit afficher l'humidité et la température du local à intervalles réguliers en précisant la date et l'heure de la mesure.



◀ **Figure 20-26**
Le tweet reçu présente les données mesurées.

Examinons le code requis (je me concentrerai sur les ajouts au sketch précédent) :

Déclaration globale

```
#include <Bridge.h>
#include <Temboo.h>
#include "TembooAccount.h"
#include <dht.h>           // Bibliothèque du capteur DHT11
#define DHT11_PIN 8       // Broche de données du DHT11

dht DHT;                  // Instance DHT11

int numRuns = 1;          // Variable du nombre d'exécutions
int maxRuns = 30;         // 30 tweets seront publiés
```

Nous devons commencer par insérer dans le code la bibliothèque DHT11 précédemment importée et indiquer le numéro de la broche de données sur laquelle le capteur est raccordé. J'ai indiqué que le capteur devait être interrogé 30 fois, mais libre à vous de choisir un autre nombre de requêtes. Comme le tweet est horodaté, il ne risque pas d'être bloqué par Twitter. Je n'ai pas modifié l'initialisation.

Initialisation

```
void setup() {  
  Serial.begin(9600);  
  delay(4000);  
  while(!Serial);  
  Bridge.begin();  
}
```

Détermination de la date et de l'heure

Le code suivant vous paraîtra familier.

```
String getDateTime() {  
  Process time;  
  time.runShellCommand("date"); // Date et heure du serveur  
  String timeStamp = "";  
  while (time.available()) {  
    char c = time.read();  
    timeStamp += c;  
  }  
  return timeStamp; // Affichage de l'horodatage  
}
```

Envoi des tweets

Nous en arrivons aux lignes de code qui permettent l'envoi du tweet.

```
void loop() {  
  if (numRuns <= maxRuns) {  
    String timestamp = getDateTime(); // Détermination de l'horodatage  
    String tweet = "";                // Message du tweet  
    String msgDHT11 = "";             // Affichage de l'état du capteur
```

La variable `timestamp` enregistre aussi bien la date que l'heure. Ainsi, à la lecture du tweet, on sait exactement quand le message a été envoyé. Le tweet est composé des quatre informations suivantes :

- horodatage (`timestamp`)
- état du capteur
- humidité (`humidity`)
- température (`temperature`)

J'ai donc déclaré la variable `tweet` avec le type de donnée `String` afin de pouvoir y enregistrer toutes les informations. Au moment de son initialisation, le capteur DHT11 transmet un message qui rend compte de l'état du processus et qui est enregistré au moyen de la variable `msgDHT11`. Examinons le processus d'initialisation de plus près.

```

int responseDHT11 = DHT.read11(DHT11_PIN); // Initialisation de DHT11
switch (responseDHT11) { // Analyse de la réponse
    case DHTLIB_OK:
        msgDHT11 = "Sensor OK. ";
        break;
    case DHTLIB_ERROR_CHECKSUM:
        msgDHT11 = "Sensor FAIL. Checksum Error! ";
        break;
    case DHTLIB_ERROR_TIMEOUT:
        msgDHT11 = "Sensor FAIL. Timeout! ";
        break;
    default:
        msgDHT11 = "Sensor FAIL. Unknown Error! ";
        break;
}

```

La variable `responseDHT11` enregistre la réponse à l'initialisation afin de l'exploiter dans une instruction `switch`. Pour traduire le code de la réponse dans une forme lisible, la variable `msgDHT11` est transmise dans l'instruction `case` d'un message qui fera ensuite partie du tweet. Pour calculer l'humidité et la température ambiante, nous utilisons les deux méthodes de la catégorie `dht11`.

```

float humidity = DHT.humidity; // Mesure de l'humidité
float temperature = DHT.temperature; // Mesure de la température

```

Les valeurs mesurées sont enregistrées dans les variables `humidity` et `temperature` qui feront ensuite aussi partie du tweet. Les lignes suivantes ne nécessitent pas de longs discours, car vous commencez à en connaître la fonction.

```

Serial.println("Running StatusesUpdate - Run #" + String(numRuns++));
TembooChoreo StatusesUpdateChoreo;
// Invoke the Temboo client
StatusesUpdateChoreo.begin();
// Set Temboo account credentials
StatusesUpdateChoreo.setAccountName(TEMBOO_ACCOUNT);
StatusesUpdateChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
StatusesUpdateChoreo.setAppKey(TEMBOO_APP_KEY);
// Set credential to use for execution
StatusesUpdateChoreo.setCredential("EriksTwitter");

```

Le tweet est maintenant composé à partir des bribes d'informations disponibles et il est présenté sous forme de message :

```

// Tweet
tweet = timestamp + " - " +
    "DHT11-Status: " + msgDHT11 + " - " +
    "Humidity: " + humidity +
    "% - Temperature: " + temperature + " degrés Celsius";

```

Puis il est ajouté à l'objet Twitter au moyen de la méthode `addInput` :

```
// Set Choreo inputs
StatusesUpdateChoreo.addInput("StatusUpdate", tweet);

Vous connaissez aussi le code suivant, donc il ne mérite pas que l'on
s'y attarde. Le sketch se termine ainsi :

// Identify the Choreo to run
StatusesUpdateChoreo.setChoreo("/Library/Twitter/Tweets/StatusesUpdate");

// Run the Choreo; when results are available, print them to serial
StatusesUpdateChoreo.run();

while (StatusesUpdateChoreo.available()) {
  char c = StatusesUpdateChoreo.read();
  Serial.print(c);
}
StatusesUpdateChoreo.close();
}
Serial.println("Waiting...");
delay(30000); // wait 30 seconds between StatusesUpdate calls
}
```

Dans cet exemple, 30 tweets sont publiés à intervalles de 30 secondes. Ouvrez, d'une part, votre moniteur série pour y suivre les réponses de Temboo et, d'autre part, le compte Twitter sur lequel les tweets sont publiés. Adaptez le code à vos besoins afin de générer, par exemple, une succession infinie de tweets qui publient les valeurs ambiantes toutes les heures. N'hésitez pas à essayer les différents réglages. Les tweets vous permettent évidemment de communiquer toutes sortes d'informations mesurées par des capteurs (souvenez-vous néanmoins que vous ne disposez que de 140 caractères). Vous pourriez utiliser les capteurs suivants :

- capteur de luminosité
- détecteur de chocs
- capteur à effet Hall

Si vous ne voulez pas qu'un tweet soit généré à intervalles réguliers, vous pouvez très facilement modifier le code du sketch afin que certains événements seulement – c'est-à-dire le dépassement de certaines valeurs seuils prédéfinies – déclenchent la publication d'un tweet.

Qu'avez-vous appris ?

- Vous savez maintenant à quoi sert Temboo.
- Vous avez créé un compte Temboo afin d'avoir accès à ses fonctionnalités.
- Pour avoir accès à Twitter par le biais de Temboo, vous avez généré des *tokens* et des *keys* qui sont nécessaires pour l'authentification.
- Pour twitter des valeurs d'humidité et de température, vous avez raccordé un capteur DHT11 à votre carte Yún et vous avez chargé la bibliothèque DHT requise pour la lecture de ces valeurs.

Exercice complémentaire

Créez un sketch Arduino afin qu'un tweet ne soit diffusé que lorsqu'un seuil prédéfini d'humidité ou de température est atteint.

Temboo et la carte Yún – Tableur Google

Une présentation sous forme de tableau est très pratique pour recenser un grand nombre de valeurs mesurées. Tout le monde connaît d'ailleurs les principaux tableurs que sont Calc de la suite OpenOffice ou Excel de la suite Microsoft Office. Ces feuilles de calcul nous permettent de gérer un nombre variable de lignes ou de colonnes dans lesquelles sont reportées diverses informations qui sont ensuite analysées. En anglais, une feuille de calcul se nomme *spreadsheet*.

Au sommaire :

- qu'est-ce que Google Docs ?
- la création d'une feuille de calcul ;
- la programmation d'un sketch qui communique des données à la feuille de calcul.

Composants nécessaires



6 potentiomètres de 10K (ou une carte Arduino SimpleBoard)

Plusieurs cavaliers flexibles de couleurs et de longueurs diverses.

Google Docs

Lorsque vous installez sur votre ordinateur la suite gratuite LibreOffice qui réunit des logiciels de traitement de texte, de tableur, de présentation, de dessin et de base de données, ces logiciels se trouvent sur votre disque dur d'où vous pouvez les lancer. Mais il existe un autre mode d'accès à ce type de logiciels. Microsoft donne ainsi accès aux programmes qui composent la suite bureautique Office 365 par le biais d'un service en ligne. C'est aussi le cas de Google Docs. Ici, nous allons voir comment transmettre facilement des valeurs mesurées à une feuille de calcul Google Docs avec la carte Arduino Yún. La feuille de calcul illustrée ci-après contient quelques valeurs qui y ont été déjà insérées par la Yún.

Figure 21-1 ▶
Feuille de calcul Google Docs

	A	B	C	D
1	Time	Sensor		
2	Tue Feb 18 18:08:11 CET 2014	280		
3	Tue Feb 18 18:09:11 CET 2014	171		
4	Tue Feb 18 18:10:11 CET 2014	172		
5				
6				
7				
8				
9				

Vous pouvez suivre la progression de la mesure et de la transmission des valeurs sur le moniteur série pendant l'exécution du sketch. Si la

fenêtre de Google Docs est ouverte en arrière-plan, vous pouvez y suivre l’affichage des valeurs quasiment en temps réel, sans même avoir à cliquer sur un quelconque bouton d’actualisation. Toute modification apportée à cette feuille de calcul s’affiche presque instantanément. Rien ne vous empêche évidemment de créer un graphique plus parlant à partir de ces valeurs afin de mieux suivre leur évolution.

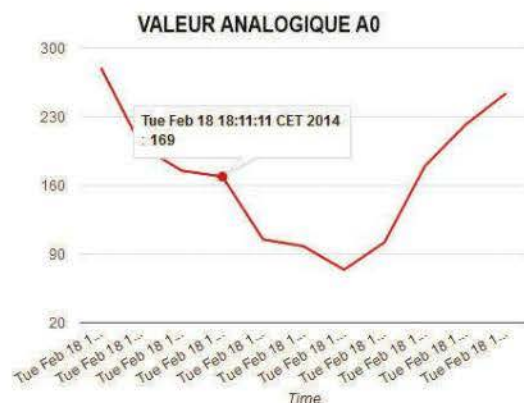


Figure 21-2
Feuille de calcul Google Docs
(courbe)

À chaque valeur mesurée représentée sur la courbe correspond une infobulle indiquant la date et l’heure de la mesure. Ces informations apparaissent lorsque le pointeur de la souris survole le point. Cette forme d’exploitation des données collectées ou enregistrées présente un avantage majeur : vous pouvez placer votre carte Yún où bon vous semble et lui demander d’échantillonner des valeurs qui seront ensuite transmises via Internet – si une connexion Internet existe bien sûr. Ainsi, vous pouvez accéder à ces données depuis le monde entier afin de suivre ce qu’il se passe chez vous. N’est-ce pas formidable ? Pour l’expérience suivante, nous allons surveiller les entrées analogiques A0, A1 et A2 et les transmettre à la feuille de calcul.

Procédure pas à pas

Nous allons examiner successivement les différentes étapes nécessaires à la réalisation de cette expérience.

Préparations de Google

Configuration du compte Google

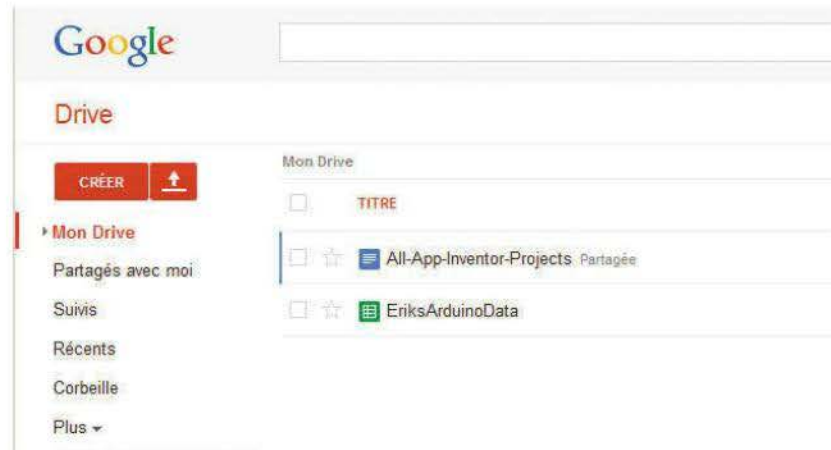
Avant de pouvoir utiliser le service Google Docs, vous devez avoir un compte Google. Si vous n’en avez pas encore, vous pouvez en

créer un sur la page <https://accounts.google.com/>. Ensuite, vous disposez d'un identifiant et d'un mot de passe Google dont vous aurez besoin pour le sketch. Gardez ces informations sous la main.

Accès à Google Docs

Une fois identifié dans Google, vous avez accès à Google Docs. Sur la figure 21-3, vous pouvez remarquer que deux fichiers se trouvent déjà dans Google Drive (Google Drive est un espace de stockage réseau qui est mis à votre disposition par Google).

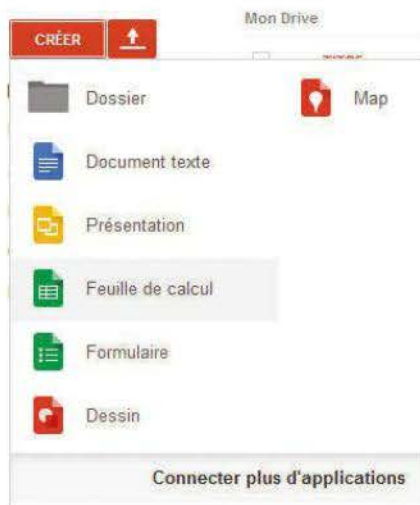
Figure 21-3 ►
Fichiers Google Drive



La feuille de calcul nommée *EriksArduinoData* contient déjà quelques valeurs mesurées. Nous allons voir comment cela fonctionne. Pour avoir accès à une feuille de calcul Google Docs depuis l'Arduino Yún, vous devez connaître le nom du fichier.

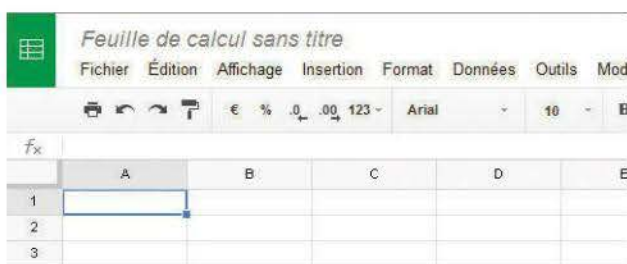
Création d'une feuille de calcul dans Google Docs

Difficile de passer à côté du gros bouton rouge nommé Créer. Lorsque vous cliquez dessus, le menu déroulant de la figure 21-4 apparaît.



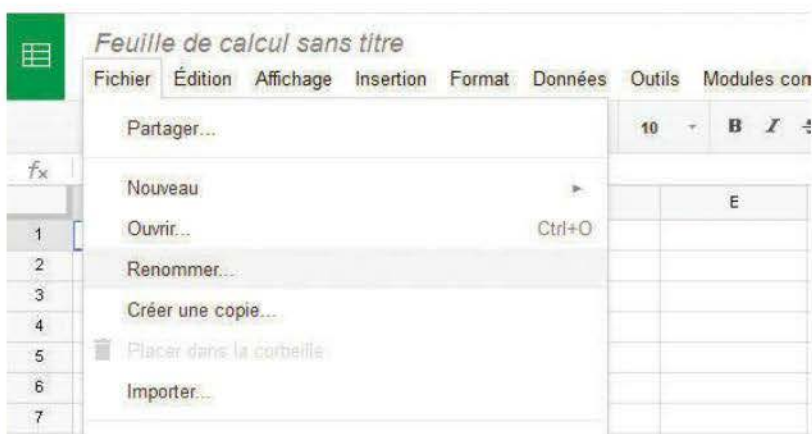
◀ **Figure 21-4**
Menu Créer de Google Docs

Pour créer une nouvelle feuille de calcul, vous devez cliquer sur l'entrée *Feuille de calcul*, qui est encadrée en rouge sur la figure. Une feuille de calcul vide et sans titre apparaît.



◀ **Figure 21-5**
Nouvelle feuille de calcul sans titre

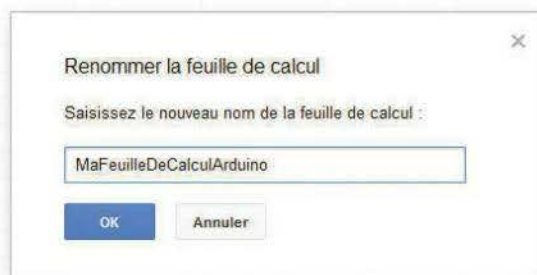
Pour pouvoir y accéder à partir d'un sketch, vous devez la nommer. Sélectionnez la commande *Fichier>Renommer*.



◀ **Figure 21-6**
Commande Fichier>Renommer

Vous pouvez maintenant renommer votre feuille de calcul.

Figure 21-7 ►
Renommage de la feuille de calcul



La feuille de calcul est renommée lorsque vous cliquez sur OK. Un message s'affiche également sur la droite de la barre de menu pour vous informer que toutes les modifications seront enregistrées dans Google Drive :



Afin que les données qui seront transmises par le sketch Arduino soient correctement identifiées et clairement organisées, vous devez aussi saisir un titre pour chaque colonne dans la première ligne.

	A	B	C	D
1	Time	Sensor A0	Sensor A1	Sensor A2

Si vous oubliez de nommer les colonnes, un message d'erreur s'affiche dans le moniteur série pour vous signaler votre oubli :

A Step Error has occurred: "A Step Error has occurred: "The Choreo encountered an error detecting the column names of the target spreadsheet.
Make sure that column names exist before appending new rows."...

Notez qu'après avoir nommé les colonnes, il n'est pas nécessaire de confirmer ou d'enregistrer les modifications. Ne perdez pas votre temps à rechercher un bouton Enregistrer. Toutes les modifications sont immédiatement transmises à Google Drive lorsque vous passez d'une cellule à la suivante.

Tout est prêt maintenant du côté de Google Docs pour que le sketch transmette les valeurs mesurées à la feuille de calcul.

Informations Temboo

Vous avez ici aussi besoin d'un compte Temboo pour le sketch Arduino. Comme je vous ai déjà expliqué la procédure de création

d'un compte Temboo dans le montage précédent, je n'y reviendrai pas ici.

Sketch Arduino

Intéressons-nous maintenant à Arduino, puisque c'est là que tout se passe. Lorsque vous créez un nouveau sketch, n'oubliez pas l'indispensable fichier d'en-tête `TembooAccount` :

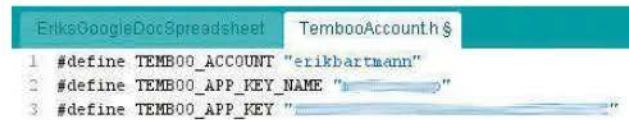


Figure 21-8
Fichier d'en-tête TembooAccount

Passons maintenant au sketch.

Déclaration globale

```
#include <Bridge.h>
#include <Temboo.h>
#include "TembooAccount.h"

const String GOOGLE_USERNAME = "votre_username@gmail.com";
const String GOOGLE_PASSWORD = "votre_password";
const String SPREADSHEET_TITLE = "MaFeuilleDeCalculArduino";
const unsigned long RUN_INTERVAL_MILLIS = 2000; // Intervalle de mesure

// the last time we ran the Choreo
// (initialized to 2 seconds ago so the
// Choreo is run immediately when we start up)
unsigned long lastRun = (unsigned long) - RUN_INTERVAL_MILLIS;
```

Vous devez saisir votre nom d'utilisateur et votre mot de passe Google.

Initialisation

```
void setup() {  
  Serial.begin(9600);  
  delay(4000);  
  while (!Serial);  
  Serial.print("Initialisation de la passerelle...");  
  Bridge.begin();  
  Serial.println("Terminée");  
}
```

L'initialisation s'effectue de la même façon qu'au montage précédent consacré à Twitter et ne nécessite donc pas plus d'explications.

Détermination de la date et de l'heure

Le code suivant vous paraîtra familier :

```
String getDateTime() {
    Process time;
    time.runShellCommand("date"); // Date et heure du serveur
    String timeStamp = "";
    while (time.available()) {
        char c = time.read();
        timeStamp += c;
    }
    return timeStamp; // Affichage de l'horodatage
}
```

Envoi des valeurs mesurées à la feuille de calcul

Les valeurs mesurées peuvent maintenant être transmises à la feuille de calcul. Commençons par la première partie de la fonction loop :

```
void loop() {
    unsigned long now = millis(); // Consigner l'heure en millisecondes
    if (now - lastRun >= RUN_INTERVAL_MILLIS) {
        lastRun = now;
        Serial.println("Mesure de la valeur par le capteur...");
        int analogValue0 = analogRead(A0);
        int analogValue1 = analogRead(A1);
        int analogValue2 = analogRead(A2);
        Serial.println("Inscription de la valeur dans la feuille de calcul...");
        TembooChoreo AppendRowChoreo; // Création de l'objet Choreo
        AppendRowChoreo.begin();      // Activation de l'objet Choreo
        ...
    }
}
```

Pour commencer, la fonction `millis` consigne la durée d'exécution du sketch en millisecondes dans la variable `now`. En outre, l'instruction `if` à la ligne suivante contrôle l'intervalle auquel les données calculées doivent être transmises. Avant de commencer à échantillonner les données analogiques, nous allons afficher un message sur le moniteur série, instancier un objet Choreo intitulé `AppendRowChoreo` et l'activer au moyen de la méthode `begin`.

```
// Temboo account credentials
AppendRowChoreo.setAccountName(TEMBOO_ACCOUNT);
AppendRowChoreo.setAppKeyName(TEMBOO_APP_KEY_NAME);
AppendRowChoreo.setAppKey(TEMBOO_APP_KEY);
```

Le code précédent transmet les autorisations requises à Temboo. La méthode `setChoreo` permet d'identifier la méthode `AppendRow` dans l'arborescence de la bibliothèque.

```
// identify the Temboo Library choreo
// to run (Google > Spreadsheets > AppendRow)
AppendRowChoreo.setChoreo("/Library/Google/Spreadsheets/AppendRow");
```

L'authentification Google doit maintenant avoir lieu :

```
// your Google username (usually your email address)
AppendRowChoreo.addInput("Username", GOOGLE_USERNAME);
// your Google account password
AppendRowChoreo.addInput("Password", GOOGLE_PASSWORD);
```

Pour s'assurer que les données sont transmises à la feuille de calcul voulue dans Google Docs, son titre est précisé à l'aide de la ligne :

```
// the title of the spreadsheet you want to append to
AppendRowChoreo.addInput("TitreDeLaFeuilleDeCalcul", SPREADSHEET_TITLE);
```

Afin que les données, c'est-à-dire la date et l'heure, ainsi que les données analogiques mesurées, soient correctement placées dans la feuille de calcul, elles doivent être séparées par des virgules. Nous utilisons donc la variable du type String et nous insérons successivement les différentes informations :

```
// convert the time and sensor values
// to a comma separated string
String rowData(getDateTime());
rowData += ",";
rowData += analogValue0;
rowData += ",";
rowData += analogValue1;
rowData += ",";
rowData += analogValue2;
```

Une information rowData a l'apparence suivante :

Date et heure, A0, A1, A2

Lorsque les informations ont été correctement assemblées, elles peuvent être insérées au moyen de la méthode addInput à l'élément qui sera transféré :

```
// add the RowData input item
AppendRowChoreo.addInput("RowData", rowData);
```

Pour finir, la méthode run est activée pour transmettre les données.

```
// run the Choreo and wait for the results
// The return code (returnCode) will indicate success or failure
unsigned int returnCode = AppendRowChoreo.run();
```

La méthode run renvoie une valeur d'état qui vous donne des informations sur la transmission réalisée. Si la valeur 0 est renvoyée, cela

signifie que la transmission des données s'est déroulée sans incident. Dans le cas contraire, l'objet `AppendRowChoreo` permet de consulter les données d'erreur au moyen de la méthode disponible en les affichant dans le moniteur série. Voici donc la fin du code du sketch :

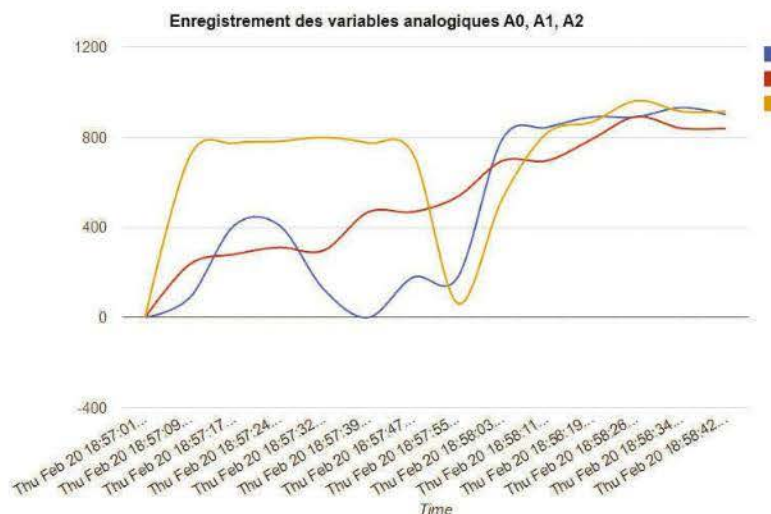
```
// return code of zero (0) means success
if (returnCode == 0) {
  Serial.println("Success! Appended " + rowData);
  Serial.println("");
} else {
  // return code of anything other than zero means failure
  // read and display any error messages
  while (AppendRowChoreo.available()) {
    char c = AppendRowChoreo.read();
    Serial.print(c);
  }
}
AppendRowChoreo.close();
}
```

Pour conclure ce montage, jetons un coup d'œil aux données qui ont été transférées dans la feuille de calcul, ainsi qu'au graphique qui a été créé à partir de ces données :

Figure 21-9 ►
Valeurs analogiques A0, A1 et A2
dans la feuille de calcul

	A	B	C	D
1	Time	Sensor A0	Sensor A1	Sensor A2
2	Thu Feb 20 18:57:01 CET 2014	0	0	0
3	Thu Feb 20 18:57:09 CET 2014	88	236	715
4	Thu Feb 20 18:57:17 CET 2014	409	280	773
5	Thu Feb 20 18:57:24 CET 2014	409	311	781
6	Thu Feb 20 18:57:32 CET 2014	125	297	799
7	Thu Feb 20 18:57:39 CET 2014	0	468	774
8	Thu Feb 20 18:57:47 CET 2014	178	468	725
9	Thu Feb 20 18:57:55 CET 2014	178	536	63
10	Thu Feb 20 18:58:03 CET 2014	791	695	527

Sélectionnez toutes les cellules à l'aide de la souris afin de les mettre en surbrillance. Puis choisissez la commande *Insertion>Graphique* afin de présenter les valeurs sous la forme d'un joli graphique.



◀ **Figure 21-10**
Valeurs analogiques A0, A1 et A2
dans un graphique

Les courbes bleu, rouge et orange correspondent aux valeurs mesurées sur les entrées analogiques A0, A1 et A2. Lorsque le pointeur de la souris survole les courbes, les données correspondant aux différents points sont affichées dans une infobulle.

Qu'avez-vous appris ?

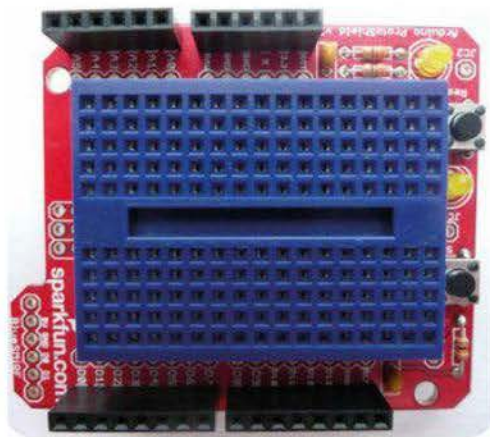
- Vous avez maintenant un aperçu de ce que vous pouvez faire avec Google Docs.
- Vous avez collecté des données et vous les avez transférées en temps réel dans un tableau ou une feuille de calcul.
- Vous avez créé un graphique à partir des données disponibles de façon à ce que les valeurs analogiques mesurées soient immédiatement présentées sur un axe chronologique.

Exercice complémentaire

Collectez des données à partir de plusieurs capteurs sur une longue période et à différents endroits de votre logement.

Réalisation d'un shield

Ce montage est consacré à la réalisation et à l'assemblage d'un shield de prototypage. Vous pouvez bien sûr vous procurer de tels shields universels prêts à l'emploi ou à souder auprès de diverses boutiques en ligne si vous n'êtes pas comme moi attiré par le bricolage. Il m'est évidemment arrivé d'en acheter un, mais j'ai pensé que je pouvais essayer de le fabriquer tout seul. J'espère ainsi vous donner envie de créer des choses par vous-même, de les souder et enfin de les assembler. La figure suivante montre un shield de prototypage prêt à l'emploi de la société Sparkfun. Celui-ci possède deux LED et deux boutons-poussoirs. Au centre une petite plaque d'essais peut accueillir des circuits plus petits. Elle s'avère parfaite pour réaliser des circuits sur un espace réduit.



◀ **Figure 22-1**
Shield de prototypage de la société Sparkfun

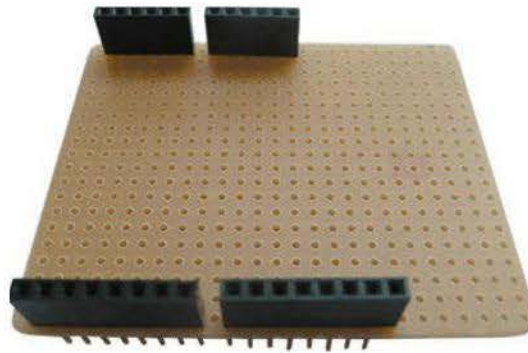
Vous pouvez bien entendu souder également des circuits à demeure sur un shield sans plaque d'essais pour les brancher si besoin est sur la carte mère, de manière à disposer d'un composant prêt à l'emploi. Je me suis construit des shields les plus divers non seulement parce qu'ils me servent en cas de besoin et sont très utiles pour les démons-

trations, mais aussi parce que j'ai plaisir à présenter quelque chose de fini sans passer trop de temps sur les composants et cavaliers flexibles. À la fin de cette instruction de montage, je vous montrerai comment construire un dé électronique sur un shield.

Shield de prototypage fait maison

Avec un peu d'adresse, vous devriez pouvoir construire vous-même le shield suivant. Ce n'est pas très compliqué et je suis sûr que vous y parviendrez. La figure 22-2 illustre le produit fini.

Figure 22-2 ►
Shield de prototypage prêt
à l'emploi



Hormis les connecteurs femelles empilables, il n'y a aucun composant sur la carte. C'est à vous de jouer et de faire en sorte que les circuits que vous avez imaginés y trouvent leur place.

De quoi avons-nous besoin ?

Outils

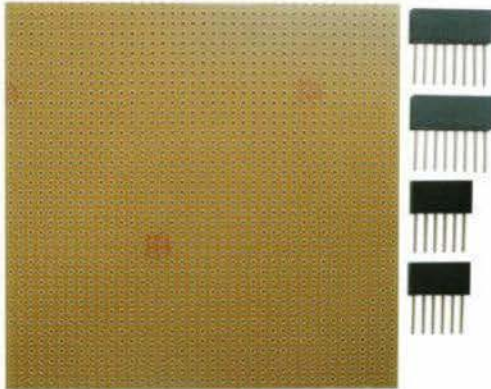
Une station de soudage est naturellement idéale, mais un fer à souder peut également convenir si vous n'êtes pas très riche. Une petite pince à becs coudés et du fil de soudure sont par ailleurs nécessaires.



◀ **Figure 22-3**
Outils nécessaires pour construire
un shield de prototypage

Matériel

Passons maintenant au matériel nécessaire à la réalisation du shield. Il s'agit d'une carte de circuit imprimé perforée et d'un jeu de connecteurs femelles empilables, que vous pouvez vous procurer auprès de la société Watterott.



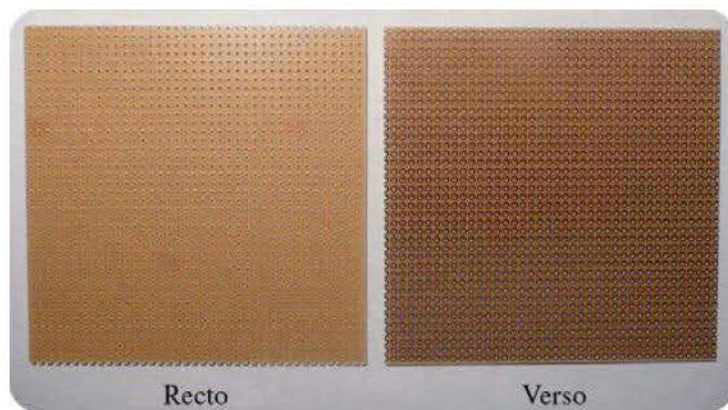
◀ **Figure 22-4**
Matériel nécessaire

Les connecteurs femelles empilables sont livrables en jeu de quatre pièces (2×6 broches + 2×8 broches).

Bon sang, rien ne va !

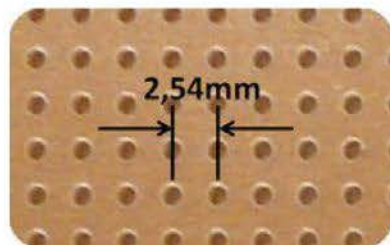
Commençons par la carte de circuit imprimé perforée, en vente sous différents formats sur le marché. Ma carte mesure 100×100 mm et ressemble à celle de la figure 22-5.

Figure 22-5 ►
Carte de circuit imprimé perforée



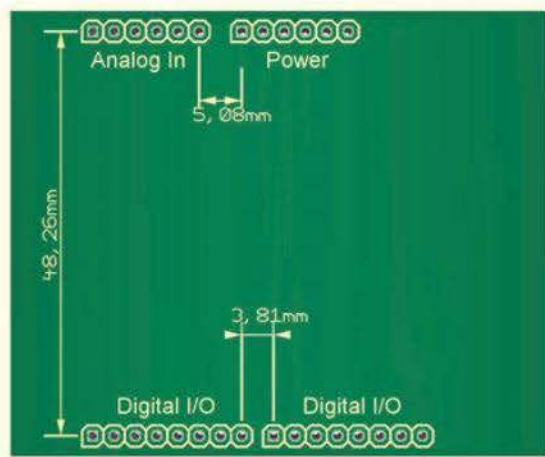
La carte se compose d'un support isolant, par exemple en bakélite ou en résine époxy, et d'une couche de cuivre conductrice. Comme son nom l'indique, la carte de circuit imprimé perforée présente une multitude de trous régulièrement espacés, bordés d'une couche de cuivre circulaire. Le fil de raccordement d'un composant est enfilé du recto vers le verso de la carte et fixé par soudure à la couche de cuivre.

Figure 22-6 ►
Vue partielle grossie d'une carte
de circuit imprimé perforée



Cette vue grossie montre la distance entre les trous, qui est en règle générale de 2,54 mm. Et c'est là que les choses commencent à se compliquer. Si tout va bien côté carte de circuit imprimé perforée, cette norme n'est en revanche pas respectée du tout côté carte Arduino, et je ne sais pas pourquoi les développeurs l'ont voulue différente.

J'ai développé ici le shield de prototypage avec le logiciel de CAO spécifique à l'électronique Target 3001! et j'y ai reporté les distances entre les trous.



◀ **Figure 22-7**

Vue de haut du shield de prototypage créée avec Target 3001!

Les dimensions de la carte de circuit imprimé perforée sont alors les suivantes :

- largeur : 64 mm ;
- hauteur : 53 mm.

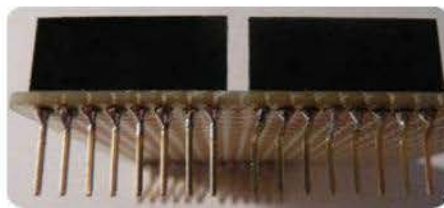
Maintenant un peu de calcul pour comprendre les éloignements des différents trous les uns des autres : les deux rangées du haut pour Analog In et Power, de 6 trous chacune, ne posent aucun problème car elles sont séparées par un trou libre, autrement dit l'écart est de $2 \times 2,54 \text{ mm} = 5,08 \text{ mm}$. Cela ne pose pas de problème pour la carte de circuit imprimé perforée. Passons aux rangées du bas pour Digital I/O. Pour une raison que j'ignore, l'écart entre ces deux rangées, 3,81 mm environ, n'est même pas un multiple de 2,54 mm mais est inférieur à deux fois 2,54 mm (soit 5,08 mm). Il n'est donc pas possible en l'état d'utiliser les connecteurs femelles et leurs broches sous cette forme. On voit cependant sur le shield fini que je les ai quand même soudés dans les trous de la carte de circuit imprimé.

Pouvez-vous me dire comment vous faites pour adapter le shield fabriqué aux connecteurs de la carte Arduino ? Il faut tordre fortement les broches !



Exact, et c'est ça la solution du problème. Il faut déformer un peu les broches du connecteur femelle de droite. On voit, sur la figure 22-8, ces broches tordues vers la gauche.

Figure 22-8 ►
Broches des connecteurs femelles numériques



Les deux figures « Avant » (22-9) et « Après » (22-10) permettent de mieux comprendre ce qu'il faut faire.

Figure 22-9 ►
Avec cet écart de $2 \times 2,54 \text{ mm} = 5,08 \text{ mm}$ entre les broches, le shield ne va pas sur la carte Arduino.

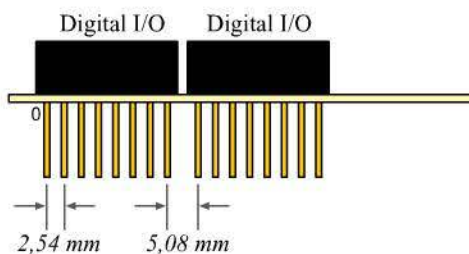
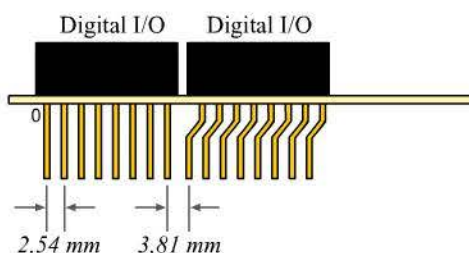


Figure 22-10 ►
Les broches ayant été tordues en conséquence, le shield va désormais sur la carte Arduino.

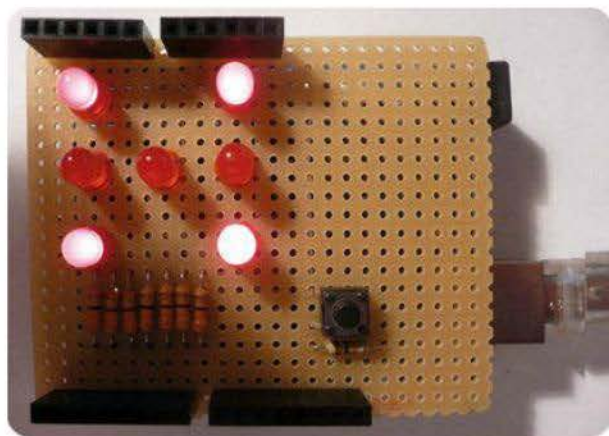


Les broches sont tordues vers la gauche au moyen de la petite pince dont j'ai parlé au début. Procédez avec soin et ne tordez pas les broches dans tous les sens car elles peuvent finir par casser. Ne craignez rien ! Je l'ai fait moi-même et ce n'est pas sorcier. La déformation des broches se fait en deux étapes. On tord d'abord la broche vers la gauche, puis on descend légèrement la pince et on tord à nouveau la broche vers la droite. On obtient ainsi une orientation verticale qui est juste un peu décalée vers la gauche. La broche doit alors se trouver au-dessus d'un trou du connecteur femelle. Procédez de préférence de gauche en droite en commençant par celle du bout.

Premier exemple d'application

Vous vous demandez peut-être à quoi bon tout ça, aussi vais-je vous donner comme promis un premier exemple d'application intéressante. Le montage n° 8 a consisté à créer un dé électronique. Un

premier projet valorisant consisterait donc à monter ce dé sur un shield pour qu'il soit disponible à tout moment et puisse être utilisé directement en cas de besoin. La figure suivante vous en donne un avant-goût et vous incitera peut-être à essayer.



◀ **Figure 22-11**
Dé électronique sur un shield

Je vous donne ici les informations nécessaires pour que tout fonctionne parfaitement.

Composants nécessaires



7 LED rouges



7 résistances de 330 Ω



1 résistance de 10 k Ω



1 bouton-poussoir



1 shield de prototypage (carte de circuit imprimé perforée + connecteurs femelles empilables)

Code du sketch

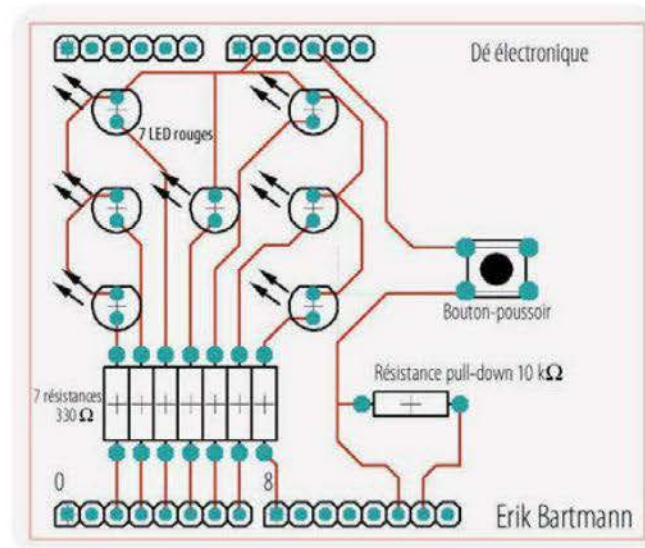
Vous pouvez bien entendu réutiliser le code du sketch du montage n° 8 du dé électronique, car le circuit n'a pas été modifié techniquement.

Réalisation du shield

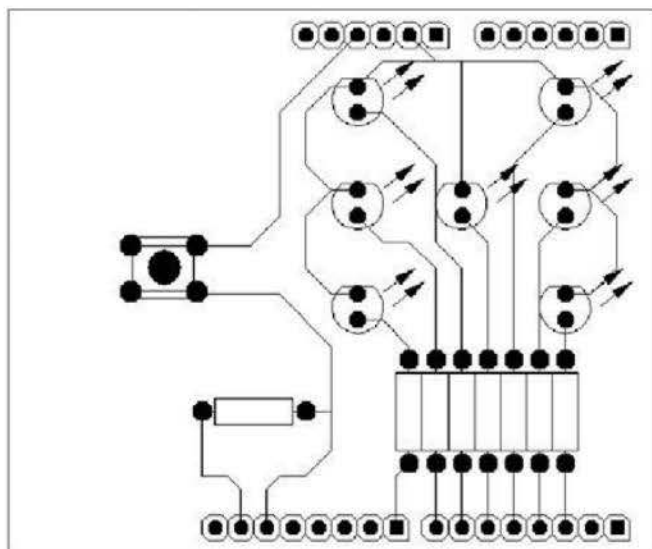
Je me suis servi du logiciel CAD Target 3001! pour construire le shield. Vous pouvez aussi construire le circuit non pas sur une carte de circuit imprimé perforée, mais sur une carte que vous aurez fabriquée vous-même à cet effet. Les manières de fabriquer ces cartes sont très diverses. Vous pouvez par exemple les graver chimiquement ou encore utiliser une fraise.

La figure 22-12 montre le shield du côté de la face supérieure, là où les composants se trouveront. Les pistes conductrices se trouvent sur la face inférieure, qui est bien sûr la symétrie de celle où sont les composants.

Figure 22-12 ►
Carte du dé électronique côté composants

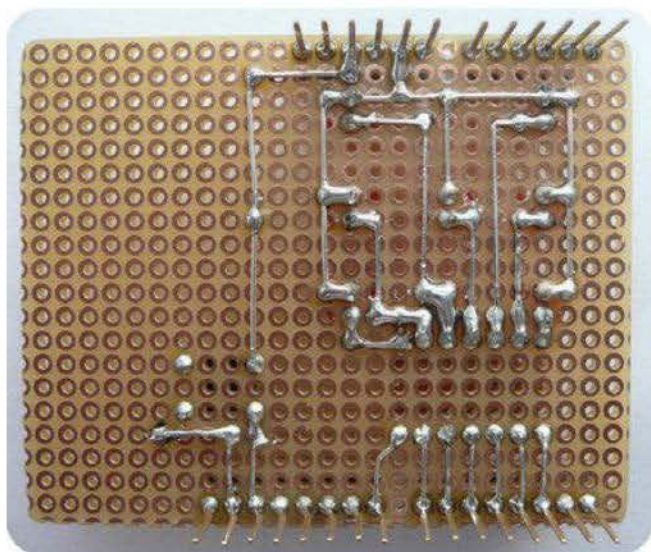


Pour que vous puissiez voir le trajet des pistes conductrices soudées, voici enfin la face inférieure de la carte finie.



◀ **Figure 22-13**
Carte du dé électronique,
vue du dessous

Pour des informations plus précises concernant le soudage, je vous invite à consulter les tutoriels existant sur Internet.



◀ **Figure 22-14**
Pistes conductrices soudées du dé
électronique



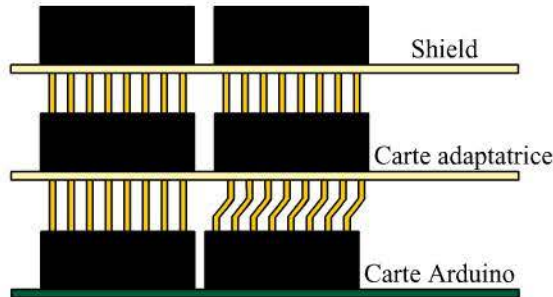
Pour aller plus loin

Si vous envisagez de construire plusieurs de ces shields avec toutes sortes de circuits et si vous ne voulez pas toujours avoir à tordre les broches du connecteur femelle, vous pouvez fabriquer une fois pour toutes un shield qui servira quasiment de carte adaptatrice. Vous posez alors dessus les shields, dont les

trous sont bien entendu espacés de 2,54 mm. Vous n'avez ainsi plus besoin de tordre les différentes broches.

La disposition des composants serait par conséquent celle illustrée à la figure 22-15.

Figure 22-15 ►
Carte Arduino + carte adaptatrice
+ shield



Toute chose ayant ses avantages et ses inconvénients, c'est à vous qu'il revient en définitive de choisir. Le principal est que vous ayez plaisir à expérimenter et que vous trouviez votre voie.



Pour aller plus loin

Si vous ne souhaitez pas empiler plusieurs shields les uns sur les autres, vous pouvez, même pour la carte adaptatrice, renoncer aux connecteurs femelles empilables. Il existe des barrettes avec des broches très longues (13 mm environ) d'un côté.



Vous pouvez bien sûr utiliser ces barrettes, cela peut même vous coûter un peu moins cher.

Référentiel des instructions

Cette annexe vous donnera un bref aperçu des instructions utilisées. Pour en savoir plus, allez sur le site Arduino <http://www.arduino.cc/fr/Main/Reference>. Vous y trouverez des instructions et des informations qui ne figurent pas faute de place dans ce livre.

Structure d'un sketch

La structure d'un sketch Arduino doit impérativement présenter les deux fonctions suivantes.

setup

La fonction `setup` est exécutée une seule fois en début de sketch et sert généralement à initialiser, autrement dit à doter de valeurs initiales, des éléments de programme tels que des variables. C'est ici que les broches numériques sont par exemple programmées comme entrées ou sorties par `pinMode`.

loop

La fonction `loop` peut être comparée, comme son nom l'indique, à une boucle en perpétuelle exécution. Elle est pratiquement la force motrice de chaque sketch et contient toutes les instructions nécessaires, telles que l'interrogation continue de broches, pour pouvoir réagir le cas échéant à des influences extérieures.

Structures de contrôle

Les structures de contrôle permettent d'influer sur le déroulement d'un sketch et réagissent aux conditions exprimées : si ceci ou cela est vrai, alors je fais quelque chose.

if

L'instruction `if` est typique de la catégorie. Sa syntaxe est la suivante :

```
if(<condition>)  
    // alors exécuter cette ligne
```

Si plusieurs instructions doivent être exécutées, elles doivent être réunies dans un bloc encadré par une paire d'accolades.

```
if(<condition>) {  
    // alors exécuter cette ligne  
    // et puis celle-ci, etc.  
}
```

Si la condition formulée est évaluée comme *vraie*, alors l'instruction ou les instructions suivantes sont exécutées.

if-else

L'instruction `if-else` est une extension de l'instruction `if`. Si la condition formulée est évaluée comme *fausse*, alors l'option `else` est choisie.

```
if(<condition>)  
    // si vrai alors exécuter cette ligne  
else  
    // si faux alors exécuter cette ligne
```

Un bloc constitué de séquences d'instructions peut ici aussi être utilisé sur le mode précédemment indiqué.

switch-case

La structure `switch-case` sert principalement quand une variable entière est susceptible de prendre un certain nombre de valeurs connues a priori. Une instruction `if` pourrait suffire mais la variante `switch-case` est considérée ici comme plus élégante.

```

switch(<variable>){
    case <valeur1>:
        // instructions
        break;
    case <valeur2>:
        // instructions
        break;
    default:
        // instructions
}

```

Les nombres suivis d'un deux-points sont comparés à la valeur de la variable et, lorsqu'il y a égalité, les instructions qui suivent sont exécutées. L'instruction `break` entraîne une interruption dans l'exécution de la structure et donc la sortie de cette structure. La ligne `default` est optionnelle et finit par être atteinte quand aucune des lignes `case` précédentes n'est vérifiée. Elle est en tout point comparable à l'alternative `else` dans une construction `if-else`.

Boucles

Les boucles servent en programmation à exécuter en permanence certaines instructions. La fonction `loop` en est un exemple. Il est évidemment possible de programmer ses propres boucles.

for

La boucle `for` est toujours utilisée quand on sait, au moment d'entrer dans la boucle, combien de fois elle doit s'exécuter.

```

for(<initialisation>; <condition>; <incrément>())
    <instructionXYZ>; //cette ligne est contrôlée par la boucle for

```

Les points indiqués dans l'en-tête de la boucle ont la signification suivante :

- **initialisation** : définition de la valeur initiale pour la variable de contrôle de la boucle ;
- **condition** : nombre d'itérations (nombre de répétitions) ;
- **incrément** : adaptation des variables indiquées dans l'initialisation.

Voici un exemple :

```
for(int i = 0; i < 10; i++)  
    Serial.println(i); // exécution 10 fois
```

while

Contrairement à la boucle `for`, seule une condition est formulée dans l'en-tête de la boucle `while`. Cela implique, par exemple, que la valeur de la variable figurant dans la condition doit être modifiée dans le corps de la boucle, faute de quoi vous aurez une boucle sans fin.

```
while(<condition>){  
    <instructionXYZ>; // cette ligne sera contrôlée via la  
                    // boucle while  
    <update>; // très important, pour éviter une boucle sans fin  
}
```

Ce type de boucle est utilisé principalement quand on ne sait pas exactement au début de la boucle combien de fois elle doit s'exécuter.

break

Les boucles `for` et `while`, qui s'exécutent tant que la condition formulée le permet, disposent également de ce que j'appelle une « issue de secours ». L'instruction `break` permet en effet de quitter prématurément une boucle, le sketch reprenant alors aussitôt après celle-ci.

```
for(i = 0; i < 10; i++){  
    if(i > 5) // sortie prématurée de la boucle for si i > 5  
        break;  
    Serial.println(i);  
}
```

Constantes importantes

La programmation d'un sketch nous amène à côtoyer ce qu'on appelle des *constantes*. Leurs noms, bien compréhensibles par l'être humain, cachent cependant des valeurs quelque peu mystérieuses.

INPUT

La constante `INPUT` est utilisée pour programmer les broches numériques quand il s'agit d'établir le sens de circulation des données. Si

une broche numérique doit servir d'entrée, cette constante est transmise comme deuxième argument à l'instruction `pinMode`, dont je parlerai plus tard dans ce référentiel. La ligne suivante configure la broche 13 comme entrée :

```
pinMode(13, INPUT);
```

OUTPUT

La constante `OUTPUT` sert également pour programmer les broches numériques quand il s'agit de définir une broche numérique comme sortie. La ligne suivante configure la broche 13 comme sortie :

```
pinMode(13, OUTPUT);
```

HIGH

La constante `HIGH` est par exemple utilisée pour mettre une sortie numérique au niveau `HIGH`. La ligne d'instruction suivante met la broche 8 au niveau `HIGH` :

```
digitalWrite(8, HIGH);
```

LOW

La constante `LOW` est par exemple employée pour mettre une sortie numérique au niveau `LOW`. La ligne d'instruction suivante met la broche 8 au niveau `LOW` :

```
digitalWrite(8, LOW);
```

true

La constante `true` sert par exemple dans des conditions gouvernant des structures de contrôle :

```
if(a == true)...
```

Si la variable booléenne `a` prend la valeur `true`, l'instruction qui suit l'instruction `if` est exécutée.

false

La constante `false` est par exemple utilisée dans des conditions gouvernant des structures de contrôle :

```
if(a == false)...
```

Si la variable booléenne `a` prend la valeur `false`, l'instruction qui suit l'instruction `if` est exécutée.

Fonctions

Fonctions concernant les broches numériques

`pinMode`

Avec l'instruction `pinMode`, on peut programmer une broche numérique pour qu'elle serve soit d'entrée soit de sortie. Les constantes `INPUT` et `OUTPUT`, dont nous venons de parler dans ce référentiel, sont ici utilisées.

`digitalWrite`

L'instruction `digitalWrite` permet d'une part d'influer sur le niveau de sortie d'une broche numérique programmée comme sortie avec `OUTPUT`. Les constantes `HIGH` et `LOW` décrites plus haut dans ce référentiel sont ici utilisées. Elle active d'autre part la résistance pull-up interne sur une broche numérique programmée comme entrée avec `INPUT`.

`digitalRead`

L'instruction `digitalRead` permet de connaître l'état (`HIGH` ou `LOW`) d'une broche numérique. La ligne suivante lit la valeur de la broche appelée `inputPin` et sauvegarde le résultat dans la variable `digValue` :

```
digValue = digitalRead(inputPin);
```

Fonctions concernant les broches analogiques

analogRead

L'instruction `analogRead` permet d'interroger une entrée analogique, une valeur comprise entre 0 et 1023 est alors délivrée en retour. Ce domaine de valeurs est basé sur la résolution de 10 bits du convertisseur analogique/numérique.

La ligne suivante lit la valeur analogique de la broche appelée `inputPin` et la sauvegarde dans la variable `anValue` :

```
anValue = analogRead(inputPin);
```

analogWrite

L'instruction `analogWrite` permet d'agir sur une sortie numérique en utilisant la MLI (modulation de largeur d'impulsions). Il ne s'agit pas ici d'un vrai signal analogique mais d'un signal numérique avec un certain rapport cyclique (voir à ce sujet la section « Que signifie MLI ? » du chapitre 10, page 214).

Fonctions concernant la durée

Certaines fonctions comportent un composant temporel.

delay

L'instruction `delay` sert à interrompre l'exécution du sketch pendant le temps indiqué, la valeur transmise étant interprétée comme des millisecondes. La ligne suivante provoque une attente de trois secondes :

```
delay(3000);
```

delayMicroseconds

Si l'instruction `delay` est *trop imprécise* du fait que la valeur est interprétée comme une indication en millisecondes, l'instruction `delayMicroseconds` peut être utilisée. L'exécution du sketch est alors interrompue pendant le temps indiqué, la valeur étant interprétée comme des microsecondes. La ligne suivante provoque une attente de cent microsecondes :

```
delayMicroseconds(100);
```

`millis`

L'instruction `millis` renvoie une valeur indiquant, en millisecondes, le temps écoulé depuis le début du sketch. Cette valeur atteint, au bout de 50 jours environ, une taille telle que la variable utilisée pour la sauvegarde déborde et que le comptage recommence à 0.

Nombres aléatoires

`random`

L'instruction `random` permet de générer des nombres pseudo-aléatoires.

```
random(10); // génération de nombres aléatoires compris entre 0 et 9
random(10, 20); // génération de nombres aléatoires compris entre
                // 10 et 19
```

À noter que le maximum indiqué n'est *jamais inclus*.

`randomSeed`

L'instruction `randomSeed` sert à réinitialiser la génération des nombres aléatoires. Ainsi, ce ne sont pas toujours les mêmes nombres aléatoires qui sont générés.

```
randomSeed(analogRead(0));
```

L'entrée analogique disponible broche 0 est utilisée et renvoie des valeurs non prévisibles à `randomSeed`.

L'interface série

Pour ce qui est de l'interface série, qui est abordée via l'objet `Serial`, différentes méthodes sont proposées.

`begin`

La méthode `begin` initialise l'objet `Serial` avec la vitesse de transfert souhaitée.

```
Serial.begin(9600); //vitesse de transfert de 9 600 bauds
```

`print`

La méthode `print` envoie un message à l'interface série, une fois sans et une fois avec saut de ligne

```
Serial.print("Ici parle Arduino !!!"); // sans saut de ligne
Serial.println("Ici parle Arduino !!!"); // avec saut de ligne
```

available

La méthode `available` vérifie que les données à récupérer auprès de l'interface série sont bien disponibles.

```
if(Serial.available() > 0) {...}
```

read

La méthode `read` lit les données de l'interface série.

```
data = Serial.read();
```

Directives de prétraitement

Deux directives de prétraitement, qui obligent le compilateur à se comporter d'une manière particulière, ont servi dans notre sketch.

#include

La directive `include` ordonne au compilateur d'intégrer la bibliothèque indiquée dans le sketch en cours. S'agissant d'une directive, la ligne ne se termine pas par un point-virgule. Par exemple :

```
#include <Stepper.h>
```

#define

La directive `define` permet de donner un nom à des constantes. Le compilateur remplace le nom par la définition indiquée partout dans tout le sketch lors de la compilation. S'agissant d'une directive, la ligne ne se termine pas par un point-virgule. Par exemple :

```
#define ledPin 8
```


Index

A

- adresse
 - IP 476
 - MAC 477
- afficheur
 - LCD 413
 - sept segments 383
- algorithme 185
- alimentation
 - électrique 31
 - externe 173
- amplificateur 94
- analogique 60
- anode 91
- API 349
- Arduino
 - alimentation électrique 31
 - environnement de développement 36, 37, 48
 - famille 11
 - microcontrôleur 27
 - ports d'entrées-sorties 33
 - structure 27
- Arduino Due 22
- Arduino Esplora 16
- Arduino Leonardo 13
- Arduino LilyPad 21
- Arduino Mega 2560 15
- Arduino Nano 20
- Arduino Uno 12
- Arduino Y'n 24
- Aref 496
- ATmega328 3
- ATTiny13 99
- auto-induction 128

B

- bargraphe 270
- BC557C 98
- bibliothèque 347
- bit 188
- Boarduino V2.0 18
- boucle 195
 - avec condition de sortie en queue 198
 - avec condition de sortie en tête 195
 - for 196, 271
 - while 197
- Bounce 266
- bouton-poussoir 102, 247, 259
 - miniature 103
 - symbole 103
- breadboard 155
- broche MLI 34
- buzzer piézoélectrique 112
 - symbole 112

C

- C/C++ 34
- câble 157
- capteur 231
 - de température 444
- carte 153
- cathode 91
- cavalier flexible 160
- CC 71
- champ électrique 85
- champ magnétique 105
- chronogramme 254
- circuit 74
 - avec transistors 125

- capacitif 123
 - condensateur électrolytique 125
 - condensateurs de filtrage 124
 - montage en parallèle 124
 - montage en série 123
- de travail 96
- facteur d'amplification 126
- imprimé 131
- intégré 3, 99
- résistif 115
 - diviseur de tension 120
 - montage en parallèle 118
 - montage en série 116
- simple 115
- classe 353
- clavier numérique 397
- clignotement 247
- commentaires 202
 - sur plusieurs lignes 203
 - sur une ligne 202
- commutateur électronique 94
- compilateur 35
- composant
 - actif 78
 - assemblage 153
 - électronique 78
 - passif 78
- concaténation 428
- condensateur 85
 - à film plastique 86
 - céramique 86
 - électrolytique 86
 - non polarisé 86
 - polarisé 86
- conductance 73
- constructeur 357, 406
- contact normalement fermé
 - symbole 102, 103
- coupleur de piles 175
- courant 69
 - alternatif 72
 - continu 71
 - de commande 96
 - de travail 96
- CPU 6

D

- DC 71
- dé électronique 327

- débogage 220
- débordement 251
- déclaration 221, 224
- delay 221, 256
- détecteur de lumière 369
- différence
 - de charge 70
 - de potentiel 70
- digitalRead 231
- digitalWrite 221
- diode 90
 - à effet tunnel 94
 - de roue 449
 - de roue libre 128
 - électroluminescente 100
 - symbole 101
 - symbole 91
 - Zener 94
- diviseur de tension 120
- données 186
 - types 188

E

- EEBoard 180
- électron 68
- électronique 67
- entrée 62
 - analogique 62, 213
 - numérique 62, 211
- erreur
 - chronologique 66
 - de syntaxe 65
 - logique 65
- excès de charge 70
- extracteur de circuit intégré 168

F

- fer à souder 178
- feu de circulation 305
- fichier
 - d'en-tête 359, 404
 - de classe 361
- fil de soudure 179
- flux d'électrons 68
- Fritzing 131, 240
 - circuit imprimé 133, 144
 - connexions 139
 - enchevêtrement 142
 - fils conducteurs courbes 139

- interface 132
- PCB 145
- platine d'essai 133
- point de flexion 143
- Routage 146
- vue schématique 133, 140
- Fritzing Creator Kit 148
- Fritzing Fab 147

G

- gabarit de pliage pour résistances 176
- générateur de signaux rectangulaires 88
- germanium 90

H

- Hello World 221
- HID 13

I

- IC 3, 99
- IDE 36
- if-else 231
- inductance 128
- initialisation 221, 224
- Installation de l'environnement de développement 37
- instanciation 356
- instruction 200
- interface série 219
- interrupteur 101
- interruption 6
- isolant 73
- ISR 9

L

- langage orienté objet 280
- LDR 82, 370
- LED 100, 271
- LiquidCrystal 419
- LM35 444
- loi d'Ohm 74

M

- machine à états 305
- manque de charge 70
- masque de ré 476
- matériel 165
- matrice de LED 151

- mémoire
 - de données 7
 - de programme 7
 - flash 7
 - SRAM 8
- méthode 351
- microcontrôleur 3
 - applications 4
 - ATmega328 3
 - bus de données 6
 - contrôleur d'interruption 9
 - mémoire 7
 - ports d'entrées-sorties 8
 - structure 5
 - unité centrale 6
- microfarad 86
- milliampère 70
- millis 247
- MLI 216
- modulo 265
- montage
 - en parallèle 124
 - en série 123
- moteur 104
 - à courant continu
 - symbole 105
 - électrique 104
 - pas-à-pas 106, 431
 - symbole 107
- MSB 190
- multimètre 76, 163
 - numérique 169

N

- nanofarad 86
- NPN 98
- NTC 83
 - symbole 84
- numérique 60

O

- opérateur conditionnel 321
- oscilloscope 171

P

- particule élémentaire 68
- passerelle 477
- patte de raccordement 101
- photorésistance 82, 370
 - symbole 82

Physical Computing VI

- picofarad 86
- piézo 459
- piézoélectrique 112
- pile 70
- pince 165
 - à dénuder 166
 - diverses 165
- pinMode 221
- plaque d'essais 155
- PNP 98
- polling 9
- pompe à dessouder 179
- port
 - analogique 213
 - numérique 211
 - USB 32
- ports d'entrée ou de sortie 33
- potentiomètre 81, 82
 - symbole 82
- principe ETS 33
- Processing 376, 445
- programmation 185, 211
 - orientée objet 350
- programme 185
- protocole 475
- prototypage VII
- PTC 83
 - symbole 84

Q

- quantité de charge 86

R

- random 283
- rebond 259
- recommandations XII
- registre
 - à décalage 286
 - de port 495
- relais 103
 - schéma 104
 - symbole 103
- réseau 473
- résistance 73, 78
 - à coefficient de température négatif 83
 - à coefficient de température positif 83
 - ajustable 81, 82
 - code couleur 79
 - fixe 78

- photosensible 370
- pull-down 242
- pull-up 243
- R2R 502
- thermosensible 83
- variable 81

- routine d'interruption 9

S

- semi-conducteur 73, 90, 95
- sens du courant 77
 - physique 77
 - technique 77
- servomoteur 109, 431
 - symbole 110
- shield 410, 439, 489, 495, 561
- shiftOut 285
- signal
 - analogique 489
 - numérique 489
- silicium 90
- sketch 37, 185, 221
 - déclaration 205
 - initialisation 205
 - loop 205
 - setup 205
 - structure 204
 - transmission 56
- son 459
- sortie 62
 - analogique 63, 216
 - numérique 63, 212
- soudure à l'étain 178
- Stepper 439
- structure de contrôle 199
- surcharge 358
- système
 - binaire 188
 - décimal 189

T

- tableau 271
 - bidimensionnel 327, 331
- TCP/IP 475
- température 443
- tension 70
- thermistance 83
 - NTC 84
 - PTC 84

tournevis 166
traitement des données 187
transistor 94
 base 96
 collecteur 96
 émetteur 96
troisième main 168

U

unité centrale 5, 6

V

variable 187
volt 70

W

wrapper 349