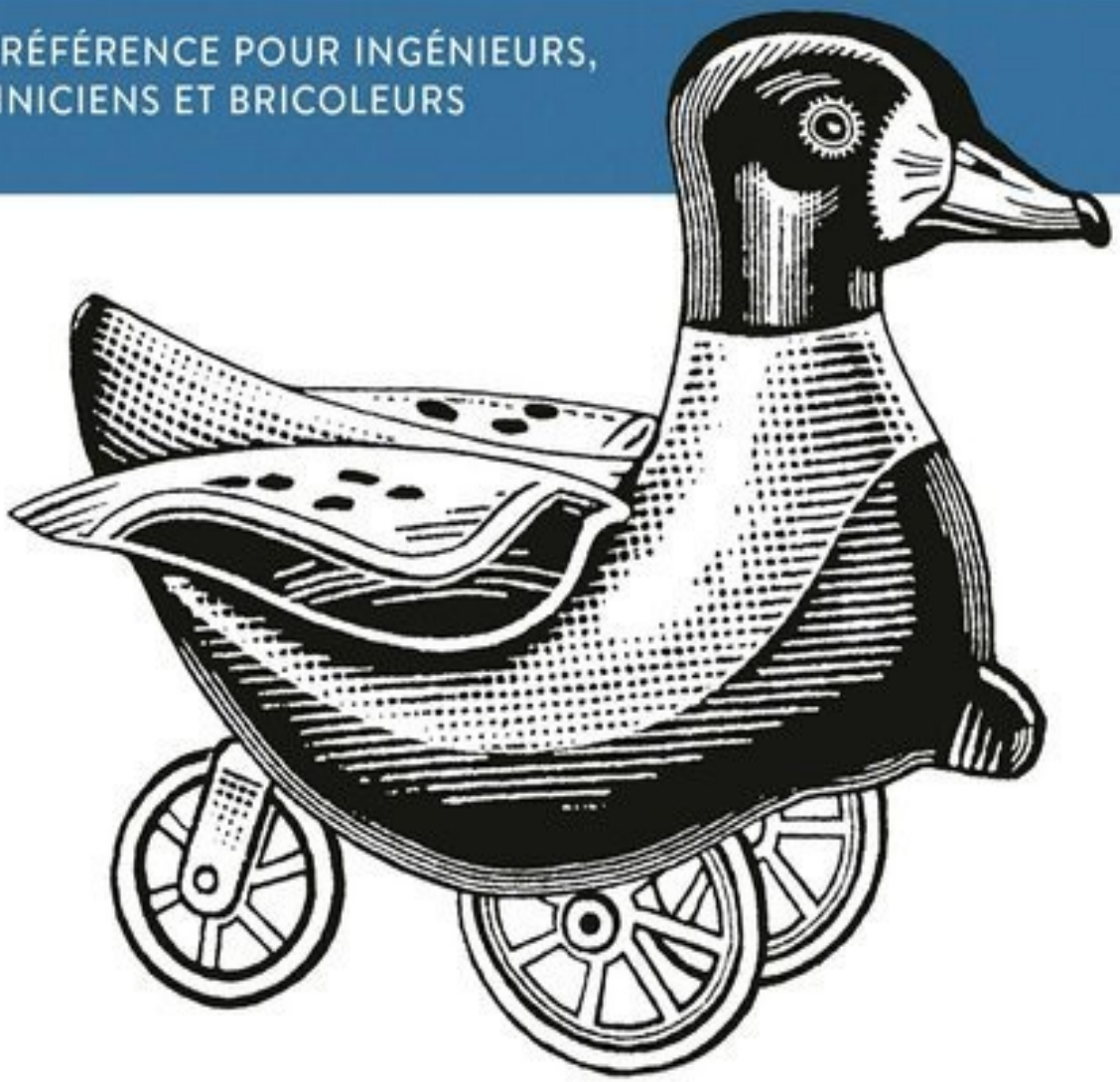
O'REILLY®

Arduino
le guide complet

UNE RÉFÉRENCE POUR INGÉNIEURS,
TECHNICIENS ET BRICOLEURS



J. M. Hughes

Aduino : le guide complet

Pour ingénieurs, techniciens et bricoleurs

J.M. Hughes

FIRST INTERACTIVE

O'REILLY®

Arduino : le guide complet

Traduction française publiée et vendue avec l'autorisation de O'Reilly Media, Inc. de *Arduino: A Technical Reference* ISBN 9781491921760 © 2016 John Hugues

12, avenue d'Italie
75013 Paris – France
Tél. : 01 44 16 09 00
Fax : 01 44 16 09 01

Courriel : firstinfo@editionsfirst.fr
Site Internet : lisez.com

ISBN : 978-2-412-03437-8
ISBN numérique : 9782412037003
Dépôt légal : janvier 2018

Traduction de l'anglais : Olivier Engler
Mise en page : Pierre Brandeis

Ce livre numérique a été converti initialement au format EPUB par Isako www.isako.com à partir de l'édition papier du même ouvrage.

Introduction

Depuis sa naissance en 2005, le projet électronique Arduino, un projet collaboratif ouvert dans l'esprit open source, est devenu l'un de ceux qui a rencontré la plus forte adhésion (certains vont jusqu'à prétendre que c'est celui qui a eu le plus grand succès). Grâce à la mise à disposition des schémas, de nombreuses cartes compatibles sont apparues dans différents pays du monde, et notamment l'Italie, le Brésil, la Chine, les Pays-Bas, l'Inde et les États-Unis. Une carte compatible Arduino peut se trouver pour environ 10 euros et l'environnement logiciel pour créer des programmes est disponible gratuitement. Au départ, toutes les cartes utilisaient l'un des microcontrôleurs de la famille AVR sur 8 bits (l'histoire des puces AVR est d'ailleurs intéressante en soi aussi). Depuis peu, l'équipe Arduino a commencé à proposer des cartes avec des unités centrales sur 32 bits, en utilisant le processeur ARM dans le modèle Due, et même une version à deux processeurs permettant d'utiliser Linux en même temps, le modèle Yún (sans compter le récent modèle Zero). Les cartes Arduino ont trouvé une utilisation dans des domaines très divers. Vous en trouverez dans les installations artistiques, dans la robotique, dans les systèmes de surveillance de l'environnement, et même dans les microsatellites appelés CubeSat qui peuvent être mis en orbite pour une fraction du coût d'un satellite habituel.

J'ai acheté ma première carte Arduino voici déjà quelques années (c'était une Duemilanove), d'abord par curiosité. J'ai en effet eu l'occasion de travailler auparavant avec de nombreux systèmes de développement basés sur des microcontrôleurs, et ce depuis les années 1980. J'ai ainsi connu les 6502, 6800 et 8051, puis les processeurs 8086, 80186 et les 68000. Pour programmer ces circuits, je devais utiliser soit le langage assembleur, soit le PL/M, les seuls langages disponibles pour les systèmes embarqués. Plus tard, j'ai pu travailler en langage C ou en Ada, car la puissance des microprocesseurs et la richesse des outils logiciels avaient suffisamment progressé. Dans tous les cas, je savais qu'il me fallait réunir énormément de documentation de référence : fiches techniques, gros manuels et guides de référence. Toute la documentation technique était habituellement livrée avec le circuit de développement et ses accessoires. L'ensemble tenait en général dans un grand carton, bien lourd.

Quelle ne fut donc pas ma surprise en recevant ma première carte Arduino. Une petite boîte qui ne contenait que la carte, un adaptateur secteur, quelques diodes LED, résistances et des bouts de fil. Aucun manuel, aucune fiche technique. Pas même un CD réunissant des documents au format PDF et des logiciels utilitaires. Il y avait une feuille dressant la liste de ce que devait contenir le colis et une adresse URL pour apprendre comment bien démarrer et trouver des liens pour télécharger les logiciels nécessaires. On peut dire que j'ai été interloqué.

Il faut dire aussi que j'étais totalement ignorant de l'histoire du mouvement Arduino, et du public auquel s'adressait ce projet. J'ai appris que les utilisateurs des cartes Arduino avaient un profil peu technique, ou pas technique du tout ; des gens qui voulaient faire des expérimentations de façon amusante, qui voulaient faire marcher des trucs. Autrement dit, les utilisateurs visés étaient d'abord des artistes et des bricoleurs, et non les ingénieurs avides de détails techniques et assoiffés de plannings, de spécifications et, bien sûr aussi, de manuels de référence.

À partir du moment où j'ai su cela, les choses sont devenues bien plus claires. J'ai beaucoup mieux compris la philosophie Arduino à la lecture du livre d'un de ses créateurs, Massimo Banzi (*Démarrez avec Arduino*, chez Dunod). Ce petit livre a constitué un excellent point de départ pour continuer dans ma recherche de détails. À la différence des fabricants de semi-conducteurs (les fondeurs de silicium) qui proposent des kits de développement, l'équipe Arduino n'est pas motivée pour vendre le maximum de puces. Ce qui l'intéresse, c'est d'aider les gens à être de plus en plus créatifs. Elle a choisi les microcontrôleurs de la gamme AVR parce qu'ils étaient très abordables et pouvaient facilement s'intégrer dans sa vision : proposer un appareil pouvant facilement servir de base pour des projets créatifs. Les contrôleurs AVR offrent suffisamment de puissance de traitement et de mémoire interne pour réaliser des choses intéressantes et assez complexes, ce qui les distingue des générations précédentes de microcontrôleurs qu'il fallait enrichir de différents outils de développement coûteux et qui embarquaient très peu de mémoire.

En complément des arguments de simplicité et de faible coût, un autre secret du succès de l'Arduino, peut-être le premier, est l'existence d'un outil logiciel de téléchargement des programmes sur la puce AVR, couplé à un atelier de création de programmes simple et enrichi de librairies de fonctions, le tout étant disponible gratuitement dans l'esprit open source et avec des licences de type Creative Commons. Ces choix sont cohérents avec la vision Arduino qui consiste à rendre l'utilisation des cartes la plus simple possible. Arduino pousse

l’utilisateur à faire des essais en lui épargnant les détails techniques et en simplifiant le processus de création. Cette aide dans les premiers pas pousse les utilisateurs à s’amuser. Pour la première fois depuis bien longtemps, je me suis vu m’amuser à connecter des objets dans différentes combinaisons pour voir ce qu’il pouvait en résulter. J’aurais vraiment aimé que les cartes Arduino existent à l’époque où j’enseignais la conception des systèmes électroniques embarqués ; cela aurait évité à mes élèves bien des déceptions lorsqu’ils cherchaient à trouver leurs repères dans les pages de code en langage assembleur, les plans d’adressage mémoire et les schémas fonctionnels obscurs.

Depuis que j’ai reçu ma première carte Arduino, j’ai imaginé de nombreuses manières d’ajouter des composants à mes montages Arduino. Certains de mes projets m’ont vraiment étonné en termes de rapport entre coût et possibilités. Je suis ainsi devenu une sorte de « fondu d’Arduino » en achetant de nombreuses cartes d’extension (boucliers) et modules peu coûteux, obtenant ainsi une vraie ménagerie électronique. Cela dit, souvent, en ouvrant la petite boîte reçue par la poste, il m’a fallu constater qu’il n’y avait aucune documentation, pas même un petit schéma photocopié.

En tant qu’ingénieur, je suis souvent déçu d’acheter quelque chose d’intéressant, pour devoir constater qu’il n’y a aucune documentation. Cela m’oblige à me lancer dans une longue quête, afin de trouver un peu de documentation sur Internet, et dans une langue que je peux comprendre (donc par exemple pas en chinois). Parfois, la recherche ne donne rien. Je dois alors remonter aux fiches techniques des composants puis faire de la rétro-ingénierie pour essayer de comprendre le circuit. Parfois, l’information n’existe pas en tant que telle, et je dois chercher sur plusieurs sites Web. La situation s’améliore quelque peu, mais cela reste désagréable. Après plusieurs années de collecte de notes, d’adresses Web et de fiches techniques, j’ai décidé de tout réunir de façon structurée, donc dans un livre.

Que peut-on trouver dans ce livre qui ne se trouve pas déjà sur Internet ? Peu de choses, pour être honnête, mais j’espère que le livre vous évitera du temps perdu et des déceptions, et cela sans compter les informations originales que j’ai découvertes par moi-même. Les données techniques officielles sont celles du fabricant Atmel, celles de l’équipe Arduino et de quelques sources, certaines très connues, d’autres moins. Certains fournisseurs ne disposent que d’une page Web élémentaire, alors que d’autres offrent des liens vers d’autres sites pour la documentation. J’ai réuni dans ce livre toutes les données indispensables que j’ai pu trouver ou rédiger par rétro-ingénierie, en sorte de laisser le moins possible de

zones d’ombre. J’ai voulu éviter aux autres de subir les mêmes déceptions au moment de trouver le détail qui leur manque au sujet d’une interface USB, d’une carte bouclier qui ne fonctionne pas correctement, ou encore la raison pour laquelle le joli petit capteur que je viens de dénicher sur le Web semble ne pas fonctionner du tout.

Ce livre est donc le fruit de mes attentes, celui dont j’ai toujours rêvé pour travailler avec les cartes Arduino et les boucliers. Je voulais que ce soit un objet physique que je puisse toujours garder à portée de main sur mon plan de travail. Il n’est en effet pas toujours pratique de devoir naviguer d’une page Web à l’autre pour vérifier une information, sans compter que l’accès à Internet n’est pas à toujours possible.

Imaginez que vous ayez à dépanner un appareil de collecte de données en haute montagne avec votre ordinateur portable et sans aucun réseau. Je veux pouvoir retrouver rapidement l’information qui me manque en travaillant avec les cartes Arduino et mes composants, quel que soit le lieu. Et ce genre de livre ne me semblait pas exister. J’espère que vous le trouverez aussi utile qu’il me l’a été lors de sa rédaction.

Lorsque vous saisissez le terme « Arduino » dans un moteur de recherche, vous obtenez des millions de réponses avec des sites qui proposent toutes sortes de capteurs de température ou d’humidité, des émetteurs à ultrasons, des cartes d’extension boucliers, des modules sans fil Bluetooth et ZigBee, *etc*. Un certain nombre de ces composants sont livrés sans aucune documentation, ou une documentation sommaire et parfois inexacte. N’en concluez pas qu’il ne faille pas vous approvisionner chez ces fournisseurs, d’autant que les prix sont généralement imbattables et la qualité de fabrication pas mauvaise. Mais comme toujours lorsque l’on s’apprête à acheter quelque chose, c’est au client de se méfier (on parle du principe de *caveat emptor*).

À qui s’adresse ce livre

Ce livre est d’abord destiné à ceux qui ont envie ou besoin d’obtenir des détails techniques. Ce sera par exemple le cas de ceux qui ont exploité le plus possible les livres d’initiation et les guides pratiques de création de projets. Si vous en êtes, il vous faut plus d’informations pour vous lancer dans un projet vraiment unique. Le livre s’adresse aussi aux ingénieurs et chercheurs qui envisagent d’utiliser une carte Arduino dans un montage expérimental de laboratoire. Le livre intéressera aussi ceux qui veulent installer une carte Arduino dans un avion radiocommandé, dans une station météorologique, et plus ambitieux encore, ceux qui cherchent à créer un microsatellite de type CubeSat.

Parmi les prérequis, il est conseillé d’avoir un minimum de connaissances des langages C et C++, une certaine idée du déplacement des électrons dans un circuit et un peu d’expérience dans le montage de circuits électroniques. Je donne dans l’Annexe D quelques conseils de lecture dans ces différents domaines.

Rappelons que les projets Arduino se situent à mi-chemin entre le monde de l’électronique et le monde de la programmation.

Description de ce livre

Ce livre est véritablement un manuel de référence, et j’ai tenté de le structurer pour vous permettre de trouver rapidement ce que vous cherchez. J’ai souvent donné les sources de mes informations.

En revanche, ce livre n’est pas un guide pratique. Je ne rappelle pas les principes de l’électronique et je ne décris pas les règles du langage C++ (celui qui permet de créer le code source des croquis qui sont les programmes Arduino). Vous trouverez déjà assez de livres d’initiation à l’électronique et à la programmation, de façon générale, ou plus spécialement dans l’environnement Arduino. Je fournis quelques suggestions à ce sujet dans l’Annexe D.

Ce livre n’est pas non plus un guide validé par l’équipe Arduino pour présenter ses produits. Il se fonde sur des informations que j’ai recueillies auprès de nombreuses sources différentes, certaines moins connues que d’autres, et j’ai ajouté mes propres notes, qui sont le fruit de mon expérience. Je suis donc seul responsable de toute erreur ou oubli.

Un peu de terminologie

C’est au début des années 1980 que l’on a commencé à vouloir distinguer les trois termes « processeur », « microprocesseur » et « microcontrôleur ». Les fabricants avaient besoin de classer leurs circuits intégrés en fonction de la quantité de circuits complémentaires qu’il fallait leur ajouter pour les rendre utiles. Les processeurs des grands systèmes, et les microprocesseurs qui animent les ordinateurs de bureau, ont tous besoin d’un certain nombre de composants périphériques pour fonctionner, notamment de la mémoire et des circuits d’entrées-sorties. En revanche, un microcontrôleur contient sur la même puce tous les composants dont il a besoin pour travailler. Souvent, les microprocesseurs ont besoin d’être associés à des cartes mémoire, alors que les microcontrôleurs se contentent de la mémoire qui est intégrée sur la même puce de silicium. Rares sont les contrôleurs qui peuvent utiliser de la mémoire complémentaire, externe au circuit.

Au long de ce livre, j’utilise les deux termes microcontrôleur et contrôleur dans le même sens. Le terme le plus exact est microcontrôleur, mais il s’agit d’abord d’un automate qui effectue un traitement sur des données, autrement dit une version un peu limitée des gros processeurs sur lesquels j’ai commencé à travailler voici bien longtemps. Tous ces circuits font le même travail, à des échelles et à des vitesses différentes.

N. d. T. : Un terme encore souvent utilisé pour traduire *library* est « bibliothèque ». J’opte dans ce livre pour le terme « librairie », tout aussi exact et plus proche de l’anglais. Vous trouverez encore le terme « bibliothèque » dans les menus de l’atelier de développement Arduino.

Contenu du livre

- Le Chapitre 1 propose un bref historique du mouvement Arduino et présente les microcontrôleurs AVR qui sont utilisés dans les cartes Arduino. J'ai expliqué les différences entre les produits qui sont compatibles au niveau matériel et ceux qui sont compatibles au niveau logiciel avec une carte Arduino officielle.

- La famille de microcontrôleurs AVR de la société Atmel fait l'objet du Chapitre 2. Il s'agit d'une présentation générale d'un circuit assez sophistiqué. J'y présente la logique des timers et chronomètres, le comparateur analogique, les entrées analogiques, l'interface de communication SPI et plusieurs autres sous-systèmes du circuit.

- Dans le Chapitre 3, nous entrons plus dans les détails des contrôleurs AVR utilisés dans les cartes Arduino, c'est-à-dire les trois modèles ATmega168/328, ATmega1280/2560 et ATmega32U4. En me basant sur les données du chapitre précédent, je fournis d'autres détails au sujet de l'architecture interne, des caractéristiques électriques et surtout du brochage des circuits intégrés.

- Le Chapitre 4 s'intéresse aux caractéristiques physiques et aux interfaces des cartes Arduino. J'y présente les interfaces USB, les circuits imprimés avec toutes leurs dimensions et le brochage des connecteurs.

- Ce qui distingue l'environnement de programmation Arduino des autres est le sujet du Chapitre 5. Je présente le concept de croquis Arduino et l'utilisation des langages C et C++ pour créer les programmes source appelés croquis. Je présente ensuite l'essentiel outil amorceur (*bootloader*) puis la fonction principale `main()`. Je montre enfin comment téléverser un programme Arduino vers le microcontrôleur.

- Dans le Chapitre 6, nous entrons dans les détails de la chaîne d'outils appelée avr-gcc. J'y montre comment programmer une carte Arduino sans utiliser l'atelier de développement Arduino IDE. Nous y découvrons aussi les fichiers de production Makefile et verrons rapidement le langage

assembleur. Le chapitre se termine par une présentation des outils permettant de téléverser le code exécutable.

- Le Chapitre 7 s'intéresse aux librairies de fonctions standard qui sont livrées avec l'atelier Arduino (on parle également de bibliothèques). L'environnement Arduino est livré dès le départ avec un grand nombre de librairies de fonctions, et de nouvelles librairies sont ajoutées sans cesse. C'est l'endroit à consulter dès que vous avez besoin de savoir s'il existe un module de librairie pour gérer un capteur ou pour réaliser un traitement particulier.

- Le Chapitre 8 passe en revue une grande sélection de cartes d'extension boucliers pour Arduino. Nous y verrons les catégories les plus utilisées, comme les cartes de mémoire flash, celles de prototypage, les cartes d'entrées-sorties, les cartes de communication Ethernet, Bluetooth et ZigBee, les contrôleurs de servomoteurs et de moteurs pas à pas et les afficheurs à diodes LED ou LCD. Nous verrons également comment combiner plusieurs boucliers et découvrirons quelques astuces pour exploiter vos boucliers le plus efficacement possible.

- Dans le Chapitre 9, nous passons en revue un grand nombre de composants utilisables avec une carte Arduino, qu'il s'agisse de capteurs, de relais, de claviers et d'autres éléments qui n'ont pas été conçus spécialement pour Arduino, mais qui sont tout à fait exploitables. Dans la plupart des cas, je fournis les informations de brochage et les schémas.

- Malgré un choix énorme, vous ne trouverez pas de bouclier pour tous vos besoins. Dans ce cas, le Chapitre 10 propose une solution. Nous y verrons en effet comment concevoir puis construire votre propre bouclier. Nous en profiterons pour voir comment utiliser un microcontrôleur AVR sans carte Arduino, tout en profitant malgré tout de l'atelier de développement Arduino.

- Les trois Chapitres 11, 12 et 13 proposent plusieurs projets pour démontrer les possibilités des contrôleurs AVR et des cartes Arduino. Vous y verrez comment exploiter une carte Arduino dans certaines situations. Il ne s'agit pas de guides pas à pas pour construire les projets. Néanmoins, rien ne vous empêche de réaliser ces projets, ou de vous en servir comme point de départ pour les vôtres. Pour chaque exemple, je commence par une

description théorique, des schémas, une nomenclature, le dessin du circuit imprimé si nécessaire et une visite rapide du logiciel permettant de faire fonctionner le projet. Ce livre est principalement orienté vers le matériel Arduino, c'est-à-dire les modules, les capteurs et les composants. Autrement dit, les logiciels présentés ne servent qu'à mettre l'accent sur certains points précis, et ne constituent pas des exemples complets et prêts à être exécutés. Vous trouverez cependant le code source de tous les exemples dans la page dédiée à ce livre sur le site de l'éditeur, ainsi que sur le site de référence GitHub.

Dans le Chapitre 11, nous découvrons un générateur de signal, appareil toujours pratique lorsque l'on passe du temps avec des circuits électroniques. Le projet permet de produire des ondes carrées et sinusoïdales à différentes fréquences, d'utiliser la tension de déclenchement, et même de créer des impulsions à largeur variable. Dans le Chapitre 12, nous voyons comment créer un thermostat intelligent que vous pouvez ensuite utiliser avec une climatisation domestique. Au lieu d'acheter un thermostat électronique, vous le construisez vous-même pour qu'il se comporte exactement comme vous le désirez. Nous verrons comment mettre en place des capteurs de température et un capteur d'humidité et comment contrôler le ventilateur de la climatisation afin de jouir d'une ambiance confortable sans provoquer sans cesse le redémarrage du compresseur ou du radiateur. Dans le Chapitre 13, je prends comme prétexte la fabrication d'un pupitre de déclenchement de séquences, par exemple pour une fusée miniature, avec toutes les mesures de sécurité envisageables. Le projet est utilisable dans tout domaine qui demande un séquencement précis de processus, que ce soit sur une ligne de production industrielle ou dans des robots de manipulation de laboratoire. Le chapitre commence par quelques concepts essentiels en ingénierie en montrant comment rédiger un jeu d'exigences.

- L'Annexe A présente les outils et accessoires dont vous aurez besoin pour monter vos circuits électroniques.

- L'Annexe B est une série de tableaux qui détaille les registres de contrôle des trois modèles de microcontrôleur ATmega168/328, ATmega1280/2560 et ATmega32U4.

- Dans l’Annexe C, je propose une liste de fabricants et de distributeurs de cartes Arduino et de produits compatibles. La liste est loin d’être exhaustive, mais elle permet de commencer à s’approvisionner.

- L’Annexe D donne quelques suggestions de livres concernant non seulement Arduino, mais également l’électronique et la programmation C et C++.

- Enfin, l’Annexe E présente quelques outils de développement logiciel pour les cartes Arduino et les microcontrôleurs AVR.

Parrainages

En dehors des nécessaires références à l'équipe Arduino et au site officiel arduino.cc, je ne suis engagé avec personne en particulier dans ce livre, tout du moins jamais de façon volontaire. Je fais référence à différents fabricants et fournisseurs, et à certains auteurs, mais j'ai cherché à rester impartial, et je n'ai pas d'inclinaison particulière pour l'un ou l'autre. Mes seuls critères de sélection sont le fait que j'ai utilisé effectivement un bouclier, un module, un capteur ou une carte Arduino, dans certains cas au minimum pour vérifier qu'il fonctionne. Les marques déposées ne sont citées dans ce livre que pour référence.

Au niveau des illustrations, j'ai tenté d'utiliser les miennes, montrant mes propres outils, circuits, modules et autres composants. Lorsqu'une photographie laisse voir une marque en particulier, cela ne signifie nullement que c'est la seule disponible, seulement que c'était celle dont je disposais au moment de l'écriture. Certaines images sont utilisées après accord de leur propriétaire et d'autres images sont dans le domaine public ou sont diffusées sous le régime de la licence Creative Commons (CC). Tous les schémas et diagrammes sont produits spécifiquement pour ce livre. Je suis le seul responsable des erreurs et oublis dans ces illustrations.

Conventions de ce livre

Voici les conventions utilisées dans ce livre au niveau de sa présentation :

- Les mots en anglais (ou autre langue étrangère) ainsi que les noms et formats de fichiers sont en *italique*.
- Tous les éléments de code (programmes, listings, fonctions, mots-clés, *etc.*) sont écrits en police `à espacement fixe`, et les variables en police à *`espacement fixe en italique`*.
- Les URL et adresses e-mail sont écrites dans une police à `espacement fixe et soulignées`.

Vous trouverez également les paragraphes suivants tout au long du livre :

Les paragraphes ainsi singularisés contiennent une astuce ou une suggestion.

Cet élément correspond à une note générale. C’est là que vous trouverez notamment les commentaires du traducteur (*N.d.T.*).

Ce symbole indique une mise en garde.

Fichiers source des exemples

Pour obtenir les fichiers de code source, rendez-vous sur le site lisez.com, puis cherchez le titre de ce livre ou son code ISBN (9782412034378). Dans la page du livre, vous trouverez une rubrique Contenu additionnel, avec un bouton Télécharger, qui vous permettra de rapatrier les fichiers sur votre ordinateur.

CHAPITRE 1
La famille Arduino

Ce chapitre propose un historique rapide du mouvement Arduino ainsi qu'une présentation des principales cartes apparues depuis 2007. Il ne s'agira pas d'un inventaire exhaustif ; je présente aussi quelques cartes compatibles produites par d'autres sociétés qu'Arduino. L'objectif est de montrer comment se différencient les principales catégories de cartes, tant au niveau de l'encombrement physique que des possibilités du microcontrôleur. Nous terminerons par un tour d'horizon des domaines d'application possibles pour les cartes Arduino.

Dans le Chapitre 2, nous verrons quelles sont les fonctions internes d'un microcontrôleur AVR. Le Chapitre 3 présente les trois principaux microcontrôleurs utilisés dans les cartes Arduino. Vous découvrirez dans le Chapitre 4 les caractéristiques physiques des principales cartes Arduino (à l'exception de la carte hybride Yún).

Un historique résumé

Au début des années 2000, Hernando Barragan avait créé un langage de programmation simplifié et l'avait baptisé *Wiring*. En 2005, Massimo Banzi et David Cuartielles ont conçu un appareil programmable facile à utiliser pour la conception et la réalisation de toutes sortes de projets artistiques interactifs. Le fruit de ce travail, réalisé à l'institut de design interactif d'Ivrea, en Italie, a été nommé Arduino. David Mellis a écrit le logiciel de l'atelier Arduino, directement inspiré de *Wiring*. Peu de temps après, Gianluca Martino et Tom Igoe sont venus rejoindre leurs collègues, portant ainsi l'équipe fondatrice à cinq personnes. Le produit devait être simple d'emploi, facile à connecter à toutes sortes de capteurs et d'actionneurs (des relais, des moteurs, et des capteurs) et surtout facile à programmer, tout en restant bon marché, car les étudiants et les artistes ne sont pas réputés pour être riches.

Le choix du microcontrôleur s'est arrêté sur la famille de microcontrôleurs 8 bits AVR d'Atmel. Un microcontrôleur (MCU) est une puce qui réunit toutes les fonctions requises avec des connexions simplifiées. Les fondateurs ont alors écrit le microgiciel amorceur (*bootloader*) qui permet le chargement du programme puis de l'outil de création fonctionnant sur une machine hôte sous Windows, macOS ou Linux pour rédiger et compiler les programmes. Cet outil est l'atelier Arduino IDE (*Integrated Development Environment*). Les fondateurs ont décidé de donner aux fichiers contenant le code source le nom de *croquis*, pour bien montrer que cette écriture devait rester simple comme une esquisse et pouvoir être peaufinée par étapes successives. Tout cela forme l'environnement Arduino.

Au cours de la première dizaine d'années de son existence, Arduino a pris de l'ampleur dans plusieurs directions. Certaines cartes apparues depuis sont plus petites que la carte initiale et d'autres plus grandes. À chaque carte correspond un domaine d'application. Toutes les cartes partagent la même librairie de fonctions open source, le même jeu d'outils avr-gcc et le même logiciel amorceur/chargeur préimplanté dans le microcontrôleur de chaque carte Arduino.

Les microcontrôleurs sont ceux de la société Atmel Corporation de San José en Californie. Presque toutes les cartes Arduino utilisent un modèle sur 8 bits. (Le modèle Due embarque un processeur assez différent, un Cortex ARM 32 bits).

Je ne parle pas du Due dans ce livre, car son processeur est très différent de ceux de la famille AVR. Sa description sera mieux à sa place dans un livre consacré aux microcontrôleurs bâtis sur l'approche ARM, comme le Cortex.

Selon les termes mêmes de l'équipe fondatrice Arduino, une carte Arduino peut se résumer à un circuit imprimé sur lequel est soudé un microcontrôleur de la famille AVR Atmel. Ce qui distingue une carte Arduino des nombreuses autres cartes dites de développement est son environnement logiciel. Tous les utilisateurs Arduino profitent du même environnement, qui constitue la pierre d'angle et la raison principale du succès d'Arduino. Je présente l'atelier de développement Arduino dans le Chapitre 5, ainsi que les librairies essentielles et l'indispensable logiciel amorceur/chargeur. Dans le Chapitre 6, je montre néanmoins comment créer du logiciel pour un microcontrôleur AVR sans utiliser l'atelier Arduino, donc avec des outils sur ligne de commande.

La gamme de cartes Arduino

Année après année, les concepteurs d'Arduino ont lancé sur le marché un assez grand nombre de cartes différentes. La première carte ayant rencontré un certain succès est la Diecimila, de 2007. Les cartes suivantes ont tenu compte de l'apparition de nouveaux modèles de microcontrôleurs dans la famille AVR. En 2012 a été lancée la Due, dont j'ai déjà parlé, et qui est bâtie sur un processeur totalement différent. Certaines cartes, comme la LilyPad et la Nano n'ont ni le même encombrement physique, ni le même brochage. Elles sont miniaturisées, et destinées à des domaines assez particuliers : la LilyPad est prévue pour être intégrée à des vêtements interactifs ; la Esplora est destiné aux équipements portatifs et les trois cartes Mini, Micro et Nano sont, comme leur nom l'indique, destinées aux projets devant être les moins encombrants possibles.

De nouveaux modèles de cartes Arduino sont lancés tous les ans, et la liste que je propose ici n'est par définition pas complète. Les plus récentes cartes tirent avantage de processeurs plus puissants avec plus de mémoire et de meilleures fonctions d'entrées-sorties. Cependant, les nouvelles cartes maintiennent une compatibilité dès que possible avec les anciennes cartes au niveau des connecteurs, afin de pouvoir utiliser les nombreuses cartes d'extension disponibles, cartes que l'on appelle boucliers, car elles s'enfichent par-dessus la carte Arduino. La compatibilité est également recherchée afin de pouvoir utiliser les mêmes capteurs, relais et actionneurs. Le Tableau 1.1 offre une sélection de cartes apparues depuis 2007. En général, les cartes les plus récentes permettent d'utiliser les fichiers de code source ou croquis des modèles antérieurs, parfois au prix de quelques adaptations ou d'une nouvelle version d'une librairie de fonctions. En revanche, les croquis écrits pour une carte récente ne fonctionnent pas toujours avec une carte plus ancienne.

Tableau 1.1 : Sélection de cartes Arduino.

Carte	Année	Microcontrôleur	Carte	Année
Diecimila	2007	ATmega168V	Mega 2560	2010

LilyPad	2007	ATmega168V/ATmega328V	Uno	2010	/
Nano	2008	ATmega328/ATmega168	Ethernet	2011	/
Mini	2008	ATmega168	Mega ADK	2011	/
Mini Pro	2008	ATmega328	Leonardo	2012	/
Duemilanove	2008	ATmega168/ATmega328	Esplora	2012	/
Mega	2009	ATmega1280	Micro	2012	/
Fio	2010	ATmega328P	Yún	2013	/

Le Tableau 1.1 n'est pas un guide d'achat, mais un tour d'horizon des cartes apparues en une dizaine d'années. Vous constatez par exemple que les deux années 2007 et 2008 ont vu apparaître la LilyPad, les trois cartes miniatures Nano, Mini et Mini Pro et la Duemilanove qui a succédé à la carte fondamentale Diecimila. Il n'y a pas de différence physique entre les deux, mais la Duemilanove apporte des améliorations au niveau de l'alimentation, par exemple une commutation automatique entre l'alimentation par la prise USB et celle par une source extérieure en courant continu. Les plus récentes éditions de la Duemilanove embarquent le nouveau microcontrôleur ATmega328, qui offre plus de mémoire pour le programme.

Je n'ai pas cité dans ce tableau l'Arduino Robot qui est un circuit imprimé équipé de moteurs et de roues. La carte Yún est particulière, car elle embarque simultanément un microcontrôleur ATmega32U4 et un processeur MIPS Atheros AR9331 qui permet de faire fonctionner une version du système Linux, dans la variante OpenWrt. Je ne décrirai pas la partie Linux de cette carte hybride, mais il est bon de savoir que la partie Arduino est tout à fait standard, puisque basée sur une carte Leonardo. Si vous avez envie d'en savoir plus sur la carte Yún, voyez la page qui lui est consacrée sur le site officiel Arduino.cc.

Dans les lignes du Tableau 1.1, j'ai parfois indiqué deux microcontrôleurs pour la même carte ; cela signifie qu'une nouvelle édition de la carte a adopté un microcontrôleur plus récent. La première version de la Duemilanove était par

exemple équipée d’un ATmega168 alors que les versions plus récentes hébergent un ATmega328. La seule différence notable entre les deux est le doublement de la mémoire interne.

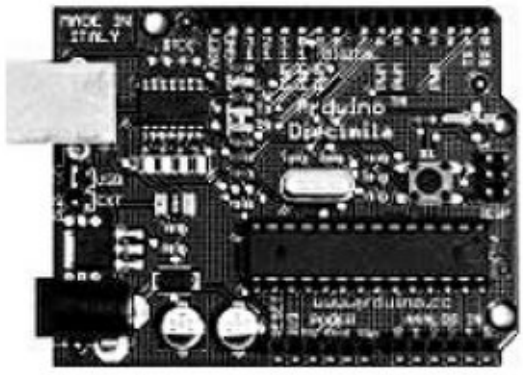
Les cartes Arduino apparues plus récemment que sont la Leonardo, la Esplora, la Micro et la Yún utilisent toutes le contrôleur ATmega32U4 qui réunit un ATmega328 et un composant pour gérer l’interface USB-série. Cela permet d’éliminer un des circuits intégrés complémentaires de la carte Uno ou Duemilanove.

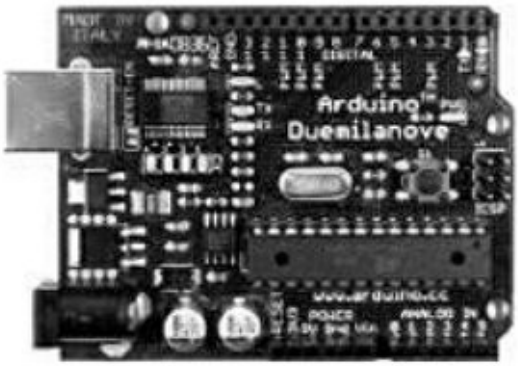
L’interface de programmation est un peu différente pour les cartes équipées du ATmega32U4, mais ces différences ne se feront pas sentir pour la plupart des utilisateurs.

Découverte des cartes Arduino

Les Tableaux 1.2 à 1.5 montrent à quoi ressemblent un certain nombre de cartes Arduino, qu’elles soient encore commercialisées ou pas. L’objectif est de vous montrer la grande diversité de formats des cartes Arduino.

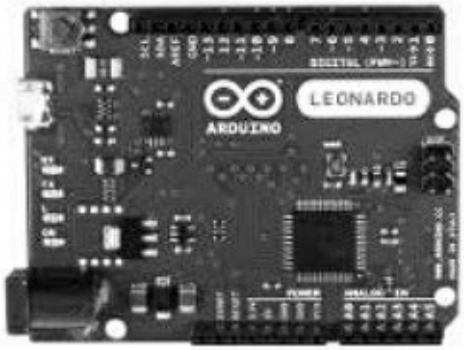
Tableau 1.2 : Aspect de quelques cartes Arduino au format de base.

	Type	Année de lancement
	Diecimila	2007
	Duemilanove	2008



Uno (R3 version) 2010

Ethernet 2011



Leonardo 2012

Tableau 1.3 : Aspect de quelques cartes Arduino au format Mega.

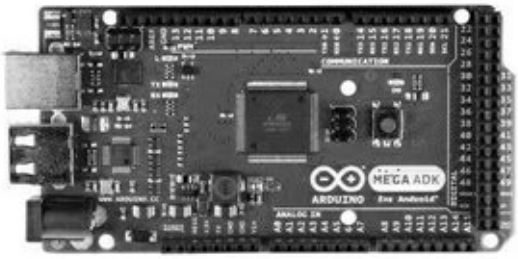
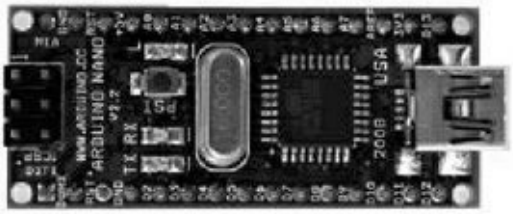
	Type	Année de lancement
	Mega	2009
	Mega 2560	2009
	Mega ADK	2011

Tableau 1.4 : Aspect de quelques cartes Arduino miniatures.

	Type	Année de lancement
	Nano	2008
	Mini	2008

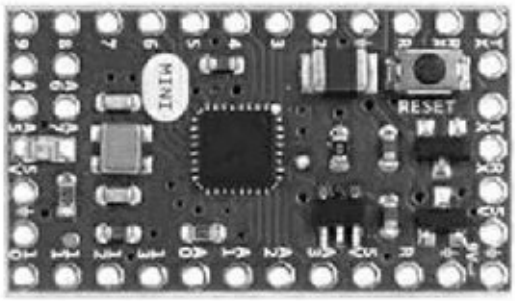
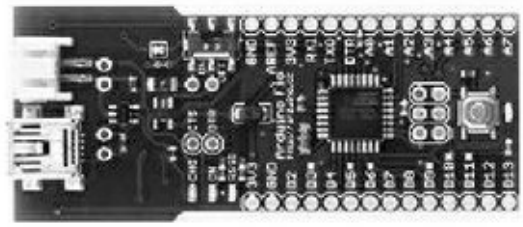
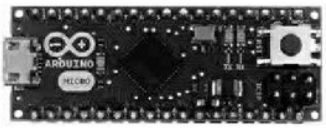
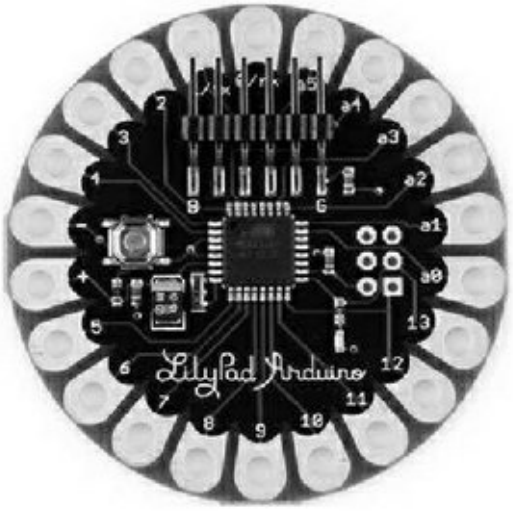
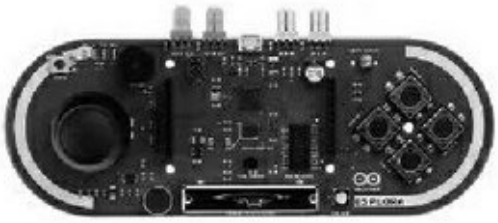
		
	Fio	2010
	Micro	2012

Tableau 1.5 : Aspect de quelques cartes Arduino à format spécial.

	Type	Année de lancement
	LilyPad	2007
	Esplora	2012



Une carte Arduino n’est pas très encombrante. Le format dit de base, qui est celui auquel tentent de se conformer les cartes d’extension appelées boucliers (*shields*), que je présente dans le Chapitre 8, est de 53 mm sur 69. La Figure 1.1 montre une carte au format de base et une carte au format étendu Mega. La Figure 1.2 montre une carte miniature Nano enfichée sur une plaque d’essai sans soudure.

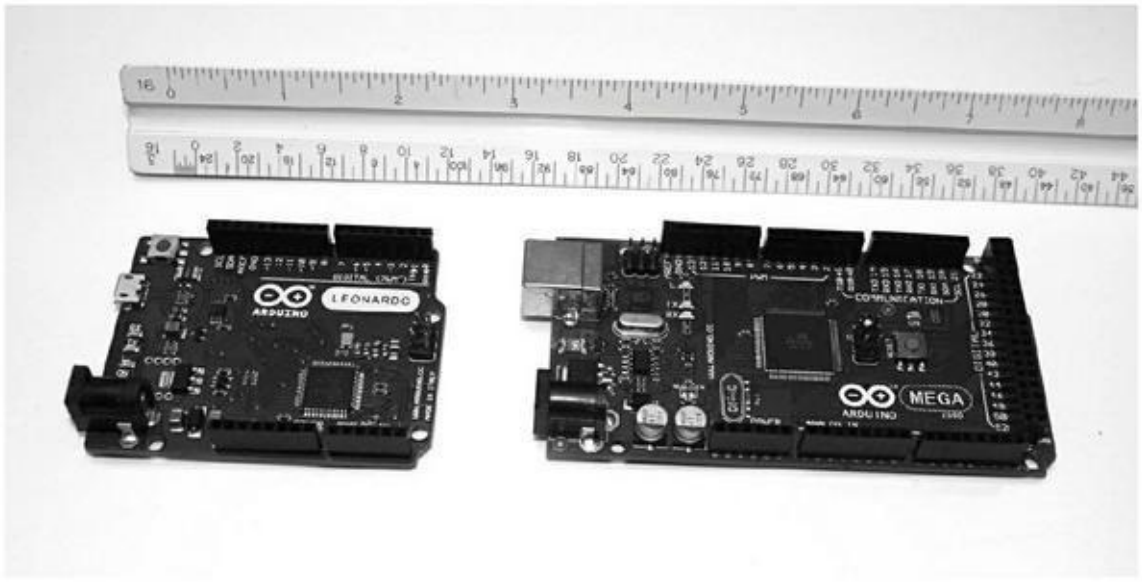


Figure 1.1 : Tailles comparées de deux cartes Arduino.

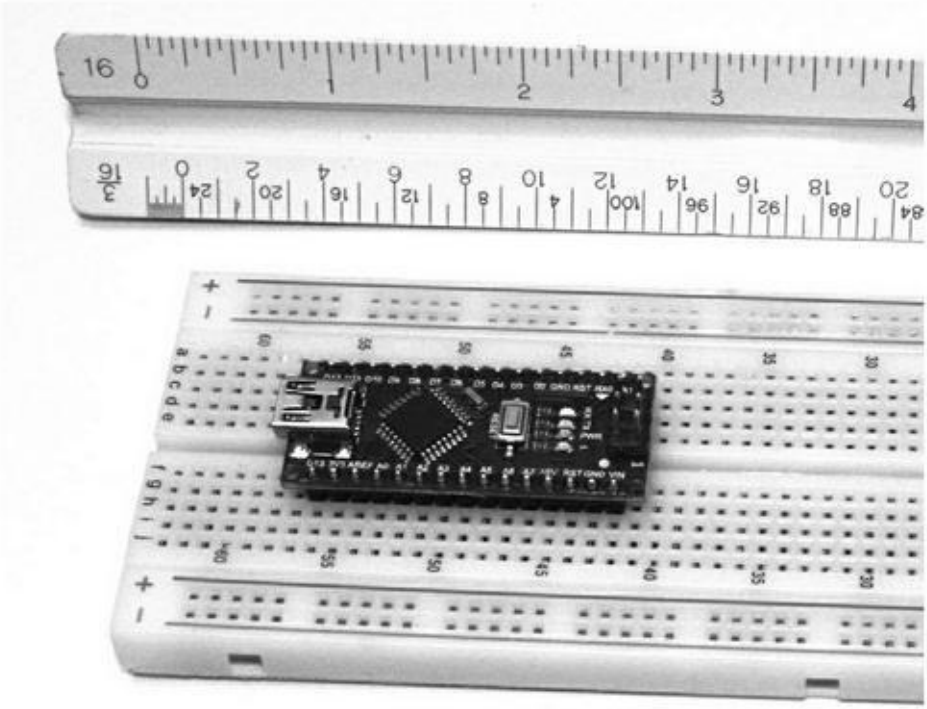


Figure 1.2 : Vue d’une carte Arduino Nano posée sur une plaque d’essai.

Vous trouverez dans le Chapitre 4 les dimensions exactes et les brochages complets d’un grand nombre de cartes Arduino. Sachez que la carte Nano, même si elle est bien plus petite, offre les mêmes possibilités qu’une Duemilanove, sauf qu’elle ne possède pas l’encombrant connecteur USB de type B, bien pratique. Cette carte miniature est donc d’abord destinée à être installée une fois pour toutes quelque part, en prenant le moins de place possible. Elle servira donc par exemple pour collecter des données sur l’environnement, dans une station météo ou une bouée en plein océan, pour recueillir les données en vol pour un modèle réduit volant, pour un système d’alarme, ou pour une machine à café intelligente.

Cartes compatibles Arduino

Les cartes officielles Arduino offrent déjà un vaste choix, mais s'y ajoutent également des cartes produites par d'autres sociétés, qui sont compatibles soit au niveau matériel, soit au niveau logiciel seulement. Pour qu'une telle carte soit compatible, elle doit être dotée dans le microcontrôleur d'un logiciel amorceur Arduino, ou un équivalent. Elle doit aussi pouvoir être programmée avec l'atelier Arduino, par sélection du modèle de carte compatible dans la liste déroulante des menus de l'atelier.

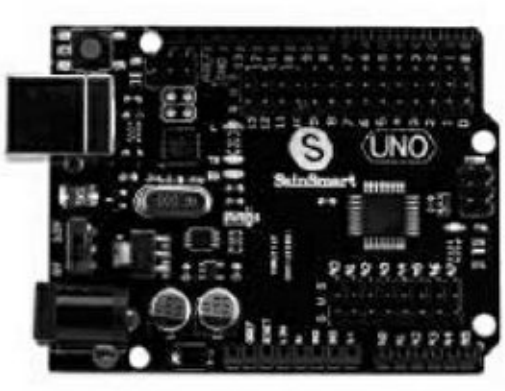
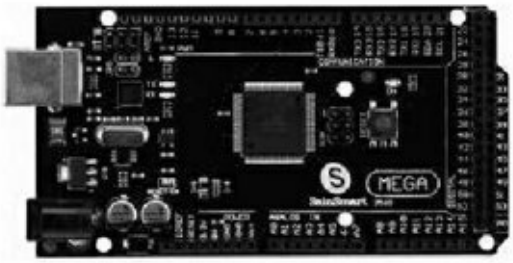
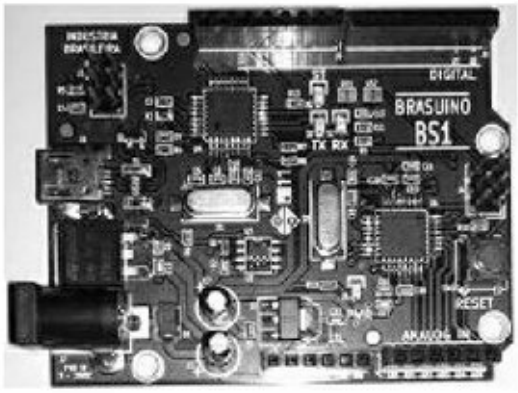
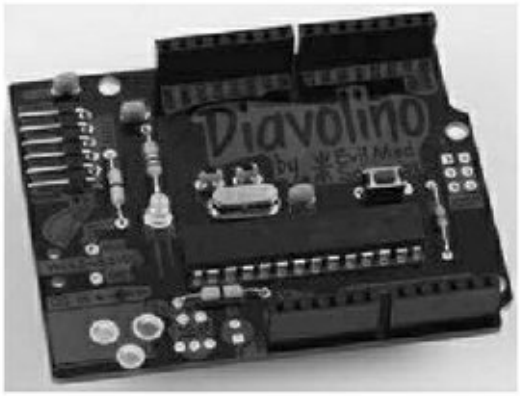
Cartes compatibles au niveau matériel

Une carte est dite compatible Arduino au niveau matériel lorsque les broches d'entrées-sorties sont strictement les mêmes que sur l'un des formats de carte Arduino officiels. Le résultat est une carte compatible qui peut normalement recevoir n'importe quelle carte bouclier prévue pour Arduino et n'importe quel module additionnel. La section sur l'utilisation du nom Arduino un peu plus loin dans ce chapitre donne des explications complémentaires.

La plupart des cartes compatibles au niveau matériel offrent le même aspect que les cartes Arduino originales, sauf qu'elles n'ont pas le droit de porter ni le logo Arduino, ni les mêmes légendes sur le circuit imprimé. Certaines cartes compatibles n'offrent pas le même aspect, mais restent compatibles au niveau des connecteurs, pour accepter les mêmes boucliers Arduino. Certaines cartes offrent un plus grand nombre de connexions, ce qui est le cas de la version SainSmart du modèle Uno qui offre plus d'entrées-sorties. Le Tableau 1.6 cite quelques noms de cartes compatibles Arduino. Vous en trouverez bien d'autres sur le marché.

Tableau 1.6 : Cartes compatibles Arduino au niveau matériel.

	Nom	Type	Origine
	SainSmart UNO	Clone d'Uno	Chine

			
	SainSmart Mega2560	Clone de Mega 2560	Chine
	Brasuino	Similaire à l'Uno, avec quelques retouches.	Brésil
	Diavolino	Clone en kit	États-Unis

---

Notez bien que la Diavolino est un kit qu'il faut monter.

Cartes compatibles au niveau logiciel

Vous trouverez un encore plus grand nombre de cartes qui sont compatibles au niveau du logiciel, parce qu'elles utilisent l'amorceur/*bootloader* Arduino et l'environnement de développement, sans être compatibles au niveau des connexions physiques. Vous pouvez ainsi créer vos programmes avec les mêmes outils de développement, mais vous aurez à travailler avec des connecteurs différents, au lieu des deux connecteurs habituels qui sont la marque de fabrique Arduino. Même un projet assez différent, s'il est basé sur un microcontrôleur AVR, peut être considéré comme compatible au niveau logiciel dès que le microcontrôleur contient l'amorceur Arduino.

En effet, l'élément central d'une carte Arduino est le microcontrôleur et le logiciel amorceur/ chargeur. Vous pourrez donc considérer qu'un circuit intégré AVR contenant le microgiciel Arduino et enfiché sur une plaque d'essai est déjà une carte compatible Arduino. Voilà pourquoi il est possible d'acheter dans le commerce des microcontrôleurs AVR de chez Atmel sur lesquels a déjà été implanté dans la mémoire flash ce fameux logiciel amorceur (*bootloader*) qui permet de téléverser le programme depuis l'atelier Arduino. Dans le Chapitre 5, nous verrons comment implanter le logiciel amorceur dans le microcontrôleur.

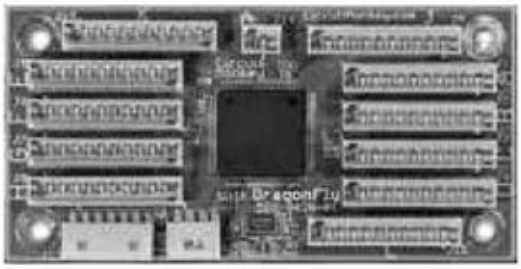
Certaines cartes officielles Arduino ne sont pourtant pas compatibles au niveau matériel. C'est le cas des cartes miniatures Mini, Micro, Nano, LilyPad et de la carte spécifique Esplora. Vous ne pouvez pas enficher par-dessus un des boucliers standard. Pourtant, ce sont des cartes Arduino, et elles sont compatibles au niveau logiciel avec l'atelier Arduino.

Un exemple de matériel compatible au niveau logiciel tout en n'étant pas fabriqué par Arduino est la Boarduino d'Adafruit. Cette carte, beaucoup plus longue que large, est prévue pour être enfichée dans une plaque d'essai sans soudure, un peu comme un circuit intégré à 40 broches. Il en existe deux variantes : DC et USB. La version DC ne comporte pas de circuit de gestion USB, il faut donc lui en ajouter un. Une autre carte compatible au niveau logiciel est la DragonFly de la société Circuit Monkey. Elle utilise des connecteurs standard Molex et non des broches mâles et femelles comme les cartes Arduino. Elle est d'abord prévue pour les environnements devant supporter de fortes vibrations, comme les drones et les robots.

Enfin, la carte hollandaise Raspduino est prévue pour être enfichée sur une carte de micro-ordinateur Raspberry Pi, et offrir les mêmes fonctions qu'une carte

Arduino Leonardo. Le résultat ressemble beaucoup à la carte hybride Yún, sans être interchangeable. Le Tableau 1.7 présente quelques cartes compatibles au niveau logiciel avec Arduino.

Tableau 1.7 : Cartes compatibles au niveau logiciel.

	Nom	Type	Origine
	Boarduino DC	Pour enfichage sur plaque d'essai	États-Unis
	Boarduino USB	Comme la précédente	États-Unis
	DragonFly	Connexions par connecteurs de type Molex	États-Unis
	Raspduino	Pour enfichage sur une carte Raspberry Pi	Pays-Bas

De nombreuses autres cartes sont disponibles dans cette catégorie. Une des raisons est la facilité d'intégration d'un microcontrôleur AVR. À partir du moment où le microcontrôleur contient le microgiciel amorceur Arduino, la programmation est grandement simplifiée, ce qui ouvre de nombreux horizons.

Les conventions autour du nom Arduino

Le schéma du circuit imprimé et tous les logiciels Arduino sont disponibles librement, sur le modèle open source. Cependant, les fondateurs d'Arduino ont déposé le mot « Arduino » et le logo pour leur seul usage. Cela explique pourquoi vous rencontrerez de nombreuses cartes qui ne portent pas le nom Arduino, parce que cela leur est interdit, mais qui sont compatibles. Souvent, les créateurs de ces cartes utilisent dans leur nom le suffixe « ...duino » ou même « ino ». C'est par exemple le cas de Freeduino, Funduino, Diavolino, Youduino, *etc*. Les cartes vendues par SainSmart utilisent directement le nom du modèle de carte, comme Uno ou Mega2560.

En 2015 et 2016, un conflit juridique a opposé les fondateurs de la société Arduino LLC, et la société sous-traitante qui fabriquait les circuits, Arduino Srl. C'est pour cette raison que les cartes officielles de la première des deux sociétés n'ont pas le droit d'être vendues sous le nom Arduino en dehors des États-Unis, donc en Europe. Elles portent ici le nom Genuino, mais ce sont strictement les mêmes.

Vous rencontrerez également des cartes qui sont présentées comme Arduino officielles, qui sont des copies, ou des contrefaçons du logo Arduino. En effet, le masque qui permet d'imprimer le logo et les informations officielles sur chaque carte Arduino font l'objet de droits d'auteur, et ce masque n'est pas librement utilisable d'après les conditions imposées par les gens d'Arduino. Un des fondateurs, Massimo Banzi, a d'ailleurs dédié une section de son blog à ces cartes non autorisées (`massimobanzi.com`). Ce qu'il pense de ces contrefaçons de cartes officielles est intéressant. Cherchez par exemple le lien « *hall of shame* » (galerie de la honte).

Vous pouvez librement réutiliser les schémas, le code du microgiciel amorceur et même celui de l'atelier Arduino IDE pour créer votre propre version d'une carte Arduino. Tout est open source. Ce que vous n'avez pas le droit de faire, c'est de donner à votre produit le nom Arduino, ni d'utiliser les éléments graphiques distinctifs d'Arduino.cc, sauf si vous leur demandez la permission d'abord.

Que faire avec une carte Arduino ?

Non seulement l'atelier Arduino rend la programmation beaucoup plus simple, mais les microcontrôleurs AVR sont également très versatiles. Les nombreux boucliers d'extension disponibles (nous les verrons dans le Chapitre 8) et l'énorme choix de capteurs et d'actionneurs bon marché (décrits dans le Chapitre 9) permettent d'envisager une très vaste gamme d'applications avec Arduino. Il vous suffit de bien tenir compte de trois contraintes :

- **Quantité d'espace mémoire.** Les microcontrôleurs AVR n'ont pas beaucoup de mémoire pour le code exécutable et les variables. La plupart des modèles ne permettent pas d'augmenter cette quantité initiale. Les deux modèles ATmega32 et ATmega128 peuvent utiliser de la mémoire externe, mais cela rend inutilisables les fonctions d'entrées-sorties des broches correspondantes. Par principe, une carte Arduino n'est pas prévue pour recevoir de la mémoire supplémentaire. Le contrôleur est destiné à gérer les entrées-sorties, dans des programmes assez courts. Il n'a jamais été question pour les contrôleurs AVR de rivaliser avec les micro-ordinateurs qui possèdent des gigaoctets de mémoire vive et un disque dur spacieux. D'ailleurs, il existe des cartes basées sur un processeur Intel qui correspondent plus à cette idée de PC miniature, mais jamais une telle carte ne pourra être implantée dans une petite boîte, dans un morceau de tube en PVC scotché contre un tronc d'arbre, dans un petit robot ou à l'intérieur du corps d'une fusée en modèle réduit. Tous ces domaines conviennent parfaitement à une carte Arduino.

- **Performances.** La deuxième des trois contraintes est la vitesse d'exécution. La vitesse d'horloge d'un contrôleur AVR va de 8 à 20 MHz (je donne des détails dans le Chapitre 4). C'est de l'ordre de 100 fois moins rapide qu'un ordinateur de bureau actuel. Il y a cependant deux différences importantes : tout d'abord, un contrôleur AVR fonctionne avec un jeu d'instructions réduit RISC. Deuxièmement, dans le monde réel des événements physiques, les choses ne surviennent pas à une vitesse telle qu'un microcontrôleur ne puisse pas en garder le contrôle. À votre avis, à quelle fréquence faut-il lancer une mesure de la température ambiante pour modifier éventuellement le réglage d'un thermostat ? Sans doute qu'une fois par seconde est inutilement fréquent. En pratique, il suffit d'interroger

le capteur toutes les 5 ou 10 secondes. Pour qu'il puisse éviter les obstacles, combien de fois par seconde un robot doit-il émettre une impulsion ultrasonore pour les détecter à temps ? À moins que le robot se déplace très vite, une impulsion tous les dixièmes de seconde devraient largement suffire. Pour un processeur AVR qui fonctionne à 16 MHz comme celui du Leonardo, il reste environ 1 million de cycles d'horloge entre deux émissions d'une impulsion, et tous ces cycles sont disponibles pour d'autres traitements. Ajoutons qu'un microcontrôleur AVR est conçu pour exécuter la plupart des instructions en un ou deux cycles d'horloge. Il reste donc largement assez de puissance de traitement pour faire exécuter votre programme entre deux palpations de l'environnement par ultrasons.

- **Ampérage supportable.** La troisième contrainte concerne l'alimentation électrique. Une carte Arduino est essentiellement un circuit imprimé sur lequel est installé un microcontrôleur AVR. Il n'y a aucune électronique intermédiaire pouvant servir de tampon mémoire (*buffer*) ou d'amplification du signal. Si vous ne prenez pas vos précautions avant d'alimenter un composant extérieur en entrée ou en sortie, vous pouvez aisément faire passer un courant trop important par le microcontrôleur, et le détruire en un instant. Tenez également compte du fait que certains des modèles AVR utilisent des entrées-sorties en 3,3 V et non en 5 V, qui est la règle. Si vous connectez à un équipement à transistors fonctionnant avec 5 V un équipement qui attend 3,3 V, vous maltraitez le matériel, et vous risquez même de déclencher des émanations de fumée fatales.

Une fois ces quelques contraintes bien posées, voici une brève sélection des domaines d'application possibles d'une carte Arduino :

- Surveillance du monde physique
    - — Station de météo automatisée
    - — Détecteur de foudre
    - — Suivi du soleil pour des panneaux solaires
    - — Surveillance de radiations
    - — Détection automatique d'animaux sauvages
    - — Systèmes d'alarme domestiques ou d'entreprise

- Pilotage de petit matériel
  - Petits robots
  - Fusées en modèle réduit
  - Avions radiocommandés
  - Drones et quadricoptères
  - Contrôle de machines-outils à commande numérique
- Automatisation à petite échelle
  - Contrôle de serres automatisé
  - Contrôle d'aquarium
  - Contrôle d'échantillons de laboratoire
  - Étuves de précision
  - Systèmes de test électronique automatisé
- Performances artistiques
  - Contrôle dynamique d'éclairage
  - Contrôle dynamique de musique
  - Structures climatiques
  - Installations artistiques interactives

Nous verrons dans les Chapitres 11, 12 et 13 des applications précises : un thermostat, un générateur de signal et un système de contrôle de déclenchement d'événements. Les possibilités ne sont limitées que par votre imagination. N'essayez pas de faire faire à une carte Arduino les mêmes traitements qu'un ordinateur de bureau. En revanche, n'hésitez pas à intégrer une carte Arduino dans toutes sortes d'applications, car c'est exactement ce dont rêvent les fondateurs d'Arduino.

Pour aller plus loin

Dans ce chapitre, je n’ai présenté qu’une courte sélection des cartes Arduino et compatibles disponibles. Pour en savoir plus, reportez-vous à l’acte à la page Wikipédia consacrée aux cartes Arduino, et bien sûr au site officiel Arduino, à l’adresse `arduino.cc`.

Consultez également les annexes pour trouver d’autres liens Web et des conseils de lecture.

CHAPITRE 2

Les microcontrôleurs AVR

Dans ses grandes lignes, une carte Arduino peut être considérée comme un circuit imprimé passif dont le seul but est de faciliter les connexions à chacune des broches d'entrées-sorties d'un microcontrôleur de la famille AVR, ce que l'on appelle également une *carte éclateuse* ou *breakout board*. En conséquence, les caractéristiques électriques d'une carte Arduino sont celles du microcontrôleur qu'elle héberge et c'est donc à ce microcontrôleur qu'il faut s'intéresser si l'on veut comprendre comment se comporte une carte Arduino. Dans ce chapitre, nous allons découvrir les principales caractéristiques de plusieurs microcontrôleurs de la famille AVR. Je rappelle qu'un microcontrôleur réunit une unité de traitement CPU (l'équivalent d'un processeur de PC), et un ensemble de fonctions périphériques telles que des chronomètres et timers, une logique de traitement de l'interface série, des convertisseurs analogiques/numériques, des comparateurs de signaux analogiques et toute une série de ports numériques d'entrées-sorties.

Les microcontrôleurs AVR sont proposés dans de nombreuses variantes, et pour chaque modèle, dans plusieurs boîtiers. Ce chapitre ne peut donc pas être exhaustif. Fort heureusement, tous les contrôleurs AVR sur 8 bits utilisent la même unité de traitement central CPU et une architecture modulaire basée sur un bus de données interne. C'est ce qui permet à chaque modèle d'offrir des caractéristiques spécifiques au niveau fonctionnel, afin de mieux s'adapter à certains domaines d'application et certaines contraintes de conception.

Les informations de ce chapitre vont se concentrer sur les principales caractéristiques. Vous n'y trouverez pas les détails à bas niveau qui sont disponibles dans la documentation officielle Atmel (atmel.com). D'autres détails sont disponibles dans les deux prochains chapitres, ainsi que dans l'Annexe B, et bien sûr aussi dans la volumineuse documentation officielle disponible gratuitement sur le site Atmel.

Historique de la famille AVR

Les microcontrôleurs AVR ont été conçus au début des années 1990 à l'institut norvégien de technologie, en tant que projet de deux étudiants. Alf-Egil Bogen et Vegard Wollan ont en effet conçu un contrôleur sur 8 bits avec une architecture d'instructions de type RISC pendant leur stage dans une usine de fabrication de semi-conducteurs à Trondheim en Norvège. Les fruits de leurs travaux ont ensuite été vendus à la société Atmel, et les deux créateurs y ont été embauchés pour poursuivre leurs recherches.

Les microcontrôleurs AVR sont extrêmement polyvalents et faciles à configurer. Ils offrent un certain nombre de caractéristiques uniques qui les distinguent des autres microcontrôleurs tels que le 8051 ou le 68HC05. Les contrôleurs AVR utilisent une architecture RISC 8 bits de type Harvard, dans laquelle le code exécutable n'est pas stocké dans le même espace mémoire que les données variables. En guise de comparaison, les microprocesseurs de nos ordinateurs de bureau utilisent l'architecture Von Neumann dans laquelle le code exécutable et les données sont stockés dans le même espace mémoire, la mémoire vive RAM.

Les processeurs AVR ont été parmi les premiers à intégrer dans la puce en silicium une zone de mémoire flash pour y stocker le programme, en remplacement des mémoires en technologie ROM ou EEPROM (qui réclamaient des ultraviolets ou une tension particulière pour l'effacement avant remise à jour). Grâce à la présence de cette mémoire flash, il est très simple de mettre à jour le programme d'un microcontrôleur AVR, aussi simple que d'écrire de nouvelles données sur une clé USB. La plupart des modèles AVR possèdent également un petit espace de mémoire permanente dans laquelle sont sauvegardés les paramètres de fonctionnement qui ne doivent pas être perdus lors du remplacement du contenu de la mémoire flash.

Architecture interne

Un microcontrôleur AVR de la gamme ATmega réunit une unité de traitement central CPU et des circuits d'entrées-sorties, de conversion analogique vers numérique, de chronométrage, de décompte et de gestion d'interface, et quelques fonctions spécifiques qui varient d'un modèle à l'autre. Toutes ces fonctions sont considérées par Atmel comme des fonctions périphériques. Ce qui distingue les différents modèles AVR, en dehors de la quantité de fonctions d'entrées-sorties, est la quantité d'espace mémoire flash disponible. Tous les modèles sur 8 bits sont dotés du même noyau de traitement CPU. Voici les principales caractéristiques qui s'appliquent à tous les microcontrôleurs AVR :

- Architecture RISC (jeu d'instructions réduit)
  - 131 instructions
  - 32 registres à usage général sur 8 bits
  - Fréquence d'horloge jusqu'à 20 MHz, soit 20 MIPS environ
- Mémoire embarquée
  - Mémoire flash pour le code exécutable, jusqu'à 256 ko
  - Mémoire non volatile EEPROM, jusqu'à 4 ko
  - Mémoire dynamique SRAM jusqu'à 32 ko
- Tension de fonctionnement
  - Tension continue entre 1,8 et 5,5 V

La Figure 2.1 montre un diagramme simplifié du noyau CPU d'un contrôleur AVR 8 bits. La Figure 2.2 montre le schéma général du même microcontrôleur, donc à un niveau plus global. Ces deux schémas sont valables pour tous les modèles AVR.

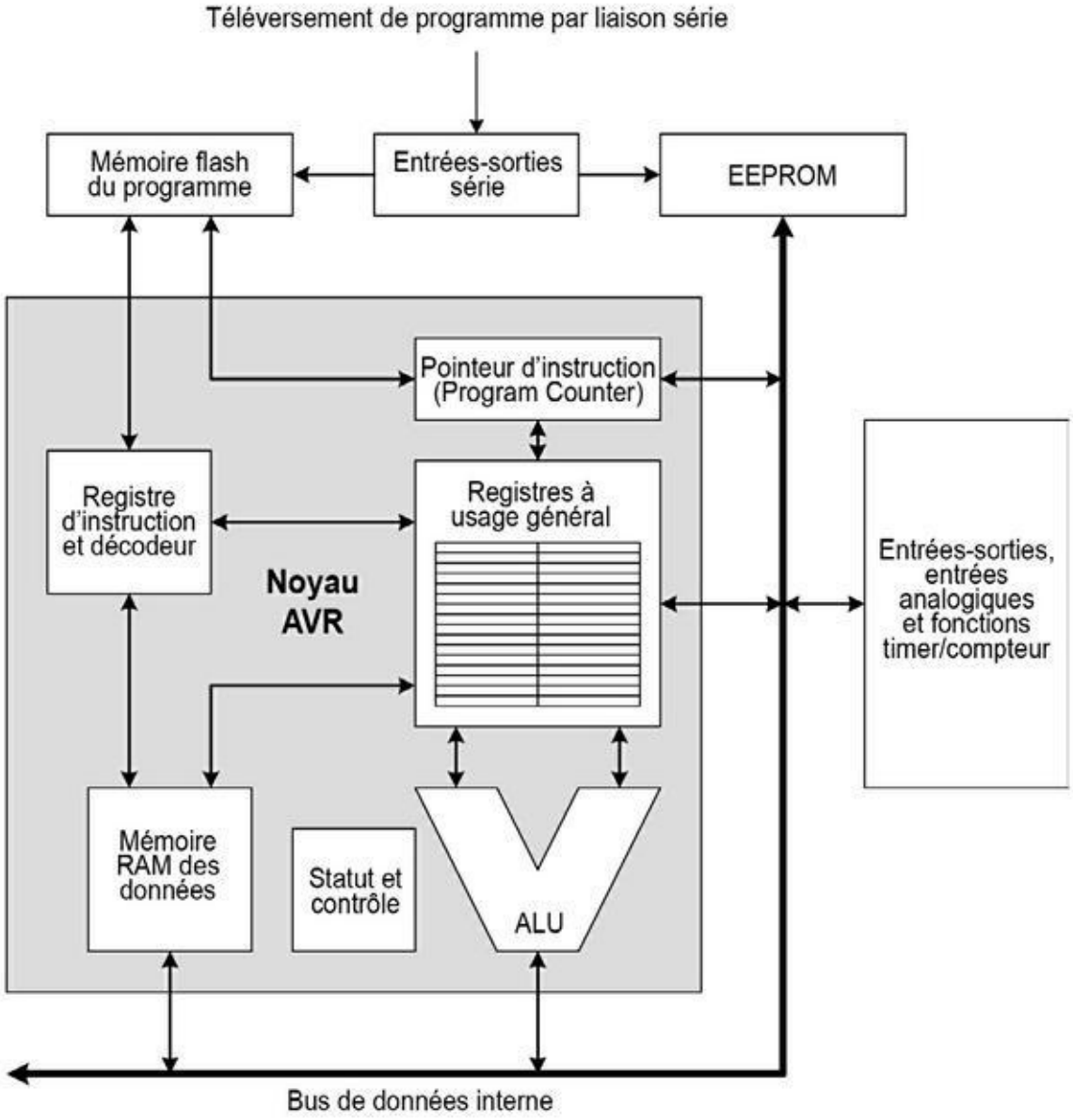


Figure 2.1 : Schéma fonctionnel de l'unité de traitement CPU AVR.

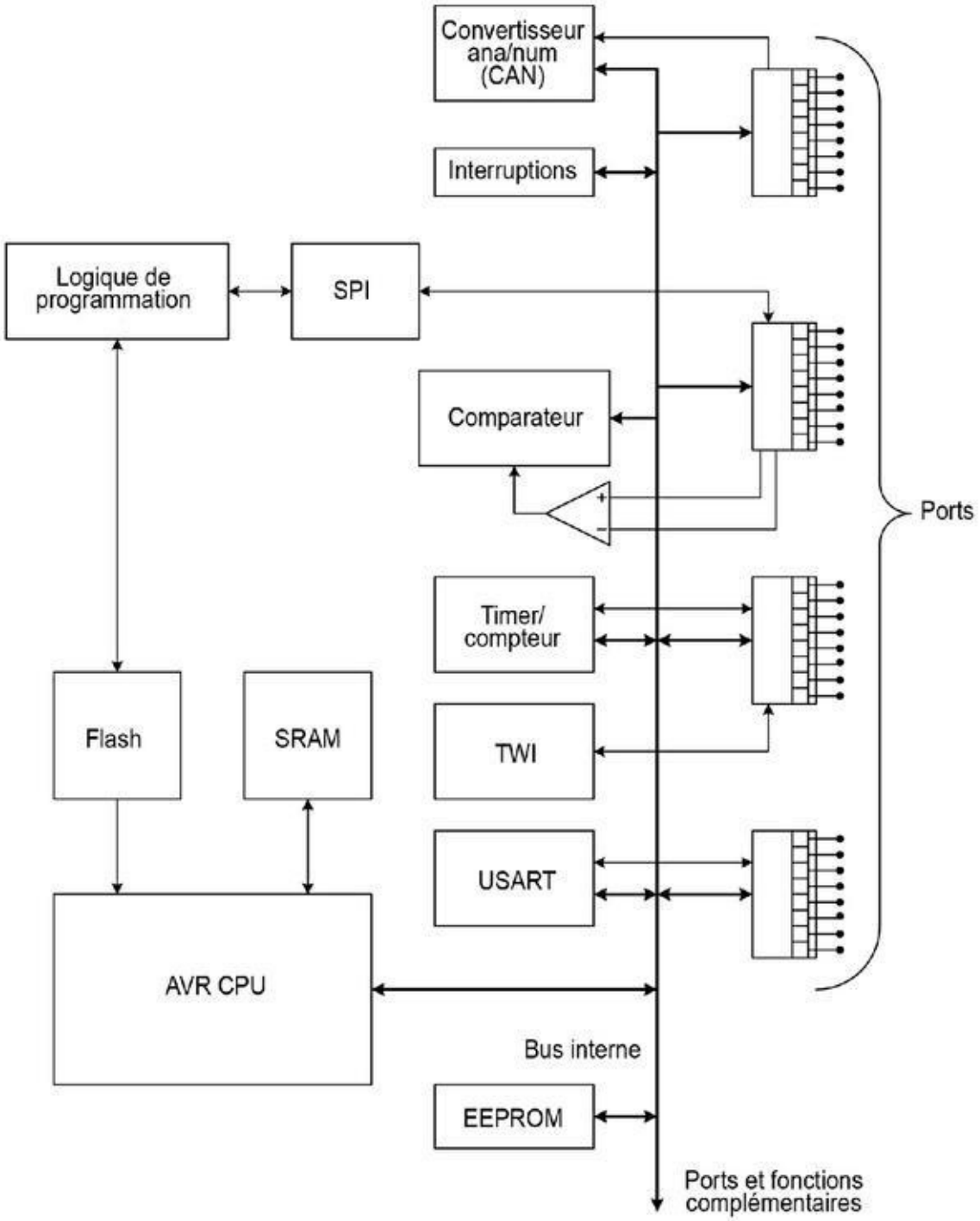


Figure 2.2 : Schéma fonctionnel générique d'un microcontrôleur AVR.

Toutes les fonctions périphériques sont contrôlées par l'unité centrale par le biais d'un bus de données interne à haute vitesse. Le contrôle du fonctionnement des éléments périphériques est réalisé au moyen d'un jeu de registres de contrôle,

distincts des registres du processeur. Toutes les fonctions périphériques partagent les broches d'entrées-sorties avec les fonctions d'entrées-sorties numériques.

Atmel propose un grand nombre de modèles de microcontrôleurs, ce qui permet aux concepteurs d'équipements de choisir le modèle le plus approprié à leurs besoins, en limitant le nombre de broches inutilisées pour chaque projet, et donc l'espace perdu sur le circuit imprimé. Le modèle tinyAVR offre l'aspect d'une toute petite puce soudée en surface (SMC) et n'offrant que six broches. Tous les modèles offrent au moins un port d'entrées-sorties numériques, qui peut être programmé pour réaliser tour à tour différentes fonctions, comme l'explique la section suivante.

En guise d'exemple, le modèle ATTINY13-20SQ est disponible sous forme d'un boîtier rectangulaire DIP à huit broches, ou bien d'une puce soudée en surface au format SOIC. Six des huit broches sont reliées à un port d'entrées-sorties sur 8 bits, le port B. Les deux autres broches servent à l'alimentation (positif et masse). Les six broches du port B peuvent servir d'entrées analogiques, de sorties d'oscillateur, d'entrées pour une interruption, de signaux pour l'interface SPI, ou encore d'entrées ou de sorties numériques. Aussi petit soit-il, il s'agit encore d'un microcontrôleur AVR. Ce modèle en particulier n'offre que 1 ko de mémoire flash et 64 octets de mémoire vive RAM pour les variables.

À l'autre extrémité de la gamme AVR, nous trouvons l'ATmega649, une vraie bête de course en comparaison : neuf ports sur 8 bits identifiés de A à J (sans le I qui risquerait d'être confondu avec le chiffre 1), 64 ko de mémoire flash, 4 ko de mémoire vive RAM, 2 ko de mémoire EEPROM, 54 broches d'entrées-sorties et même une interface embarquée pour afficheur LCD. Il existe même une gamme AVR sur 32 bits, la gamme AVR32, qui offre jusqu'à 256 ko de mémoire flash, 32 ko de mémoire RAM, un processeur de signaux numériques DSP intégré, une zone de mémoire protégée et 36 broches d'entrées-sorties.

Il n'existe aucune carte Arduino construite autour d'un aussi petit modèle que le tinyAVR. Il serait en effet très difficile de réussir à faire tenir le (pourtant petit) logiciel amorceur Arduino alors qu'il n'y a que 1 ko de mémoire flash. Il ne resterait plus de place pour le programme de l'utilisateur ! Il n'y a pas non plus de carte Arduino pour les modèles de haut de gamme comme l'ATmega649, ni pour les modèles sur 32 bits. Néanmoins, l'éventail de cartes Arduino offre suffisamment de choix, mais sachez que les modèles de contrôleurs qui sont choisis pour construire des cartes Arduino ne sont pas les seuls disponibles dans la gamme AVR.

Les mémoires internes

Comme déjà dit, ce qui distingue principalement les différents modèles de contrôleurs AVR est la quantité de mémoire disponible, dans les trois catégories : flash, SRAM et EEPROM. La mémoire flash sert au code, la mémoire SRAM aux données variables et à la pile d'appels (qui sert à appeler les sous-programmes), et la mémoire EEPROM sert à sauvegarder les données qui doivent persister malgré la mise hors tension. Il est possible d'écrire des données depuis l'extérieur du contrôleur dans la mémoire flash et dans la mémoire EEPROM, et les données qu'elles contiennent ne sont pas perdues à la mise hors tension. En revanche, le contenu de la mémoire dynamique SRAM est volatile, et donc perdu lors d'une mise hors tension ou d'un Reset du contrôleur.

Les fonctions périphériques

Le noyau d'un microcontrôleur AVR est une unité de traitement 8 bits, mais ce qui rend les microcontrôleurs vraiment uniques est l'ensemble des fonctions périphériques qui sont directement intégrées sur la même puce. Le nombre de fonctions disponibles varie d'un modèle à l'autre.

Certains modèles n'offrent qu'un chronomètre/timer, d'autres en ont deux, et certains six. Certains modèles disposent d'un convertisseur analogique-numérique sur 10 bits et d'autres sur 12 bits. Tous les modèles offrent des broches d'entrées-sorties bidirectionnelles pour les signaux numériques. Certains sont capables de piloter directement un écran tactile ou d'autres composants d'interaction.

Les sections qui suivent offrent une description générale des fonctions périphériques AVR. En guise d'illustration, nous choisissons le modèle ATmega168. Pour en savoir plus au sujet des différents processeurs utilisés dans les cartes Arduino, vous consulterez le Chapitre 3 ainsi que la documentation technique Atmel.

Les registres de contrôle des entrées-sorties

En plus des 32 registres internes de l'unité centrale, un contrôleur AVR possède un certain nombre de registres de contrôle qui servent à contrôler le fonctionnement des ports d'entrées-sorties, les timers, les interfaces de communication, parmi d'autres fonctions. Ces registres sont en nombre variable selon le modèle, puisque le nombre de ports d'entrées-sorties est variable, ainsi que le nombre de fonctions périphériques. Vous trouverez dans l'Annexe B une description détaillée des registres de contrôle pour les modèles AVR utilisés dans les cartes Arduino présentées dans ce livre.

Même un contrôleur assez modeste comme l'ATmega168 offre un nombre de fonctions internes bien supérieur au nombre de broches d'entrées-sorties disponibles. C'est pour cette raison que la plupart des broches peuvent être configurées pour servir à une fonction plutôt qu'une autre, la sélection étant faite grâce au contenu des registres de contrôle. La fonction d'une broche peut être modifiée de façon dynamique pendant l'exécution. Il est donc possible de

réaliser une action avec une broche, puis de la reprogrammer pour qu'elle serve à une autre fonction.

Par exemple, la broche 12 d'un contrôleur ATmega168 en boîtier DIP sur 28 broches est au départ connectée à la sortie PD6 (bit 6 du port D). La même broche peut être reconfigurée pour servir de source d'interruption sous le nom PCINT22, ou bien en tant qu'entrée positive du comparateur analogique interne, sous le nom AIN0, ou encore en tant que sortie d'un circuit de comparaison de chronomètre (la sortie timer/compteur0 de comparaison A), ce qui permet de faire générer un train d'impulsions PWM.

Les ports d'entrées-sorties numériques

Les ports d'entrées-sorties d'un microcontrôleur AVR sont bidirectionnels. Ils servent bien entendu à communiquer avec le monde extérieur. Chaque port est un registre 8 bits dont tous les bits ou certains seulement sont accessibles individuellement sur des broches physiques du circuit intégré. Le nombre de ports varie d'un modèle AVR à l'autre ; un seul pour l'ATTINY13-20SQ et jusqu'à neuf pour l'ATmega649. Les ports sont identifiés par des lettres majuscules A, B, C, *etc*.

Chaque broche d'un port est contrôlée par des circuits logiques internes pour le choix du sens d'utilisation (entrée ou sortie), l'activation ou non d'une résistance interne de tirage (pour forcer la broche à l'état haut), la synchronisation et quelques autres fonctions. La Figure 2.3 propose un schéma simplifié d'une broche de port. La mention **PX** symbolise la broche numéro X d'un port, la valeur X pouvant valoir de 0 à 7.

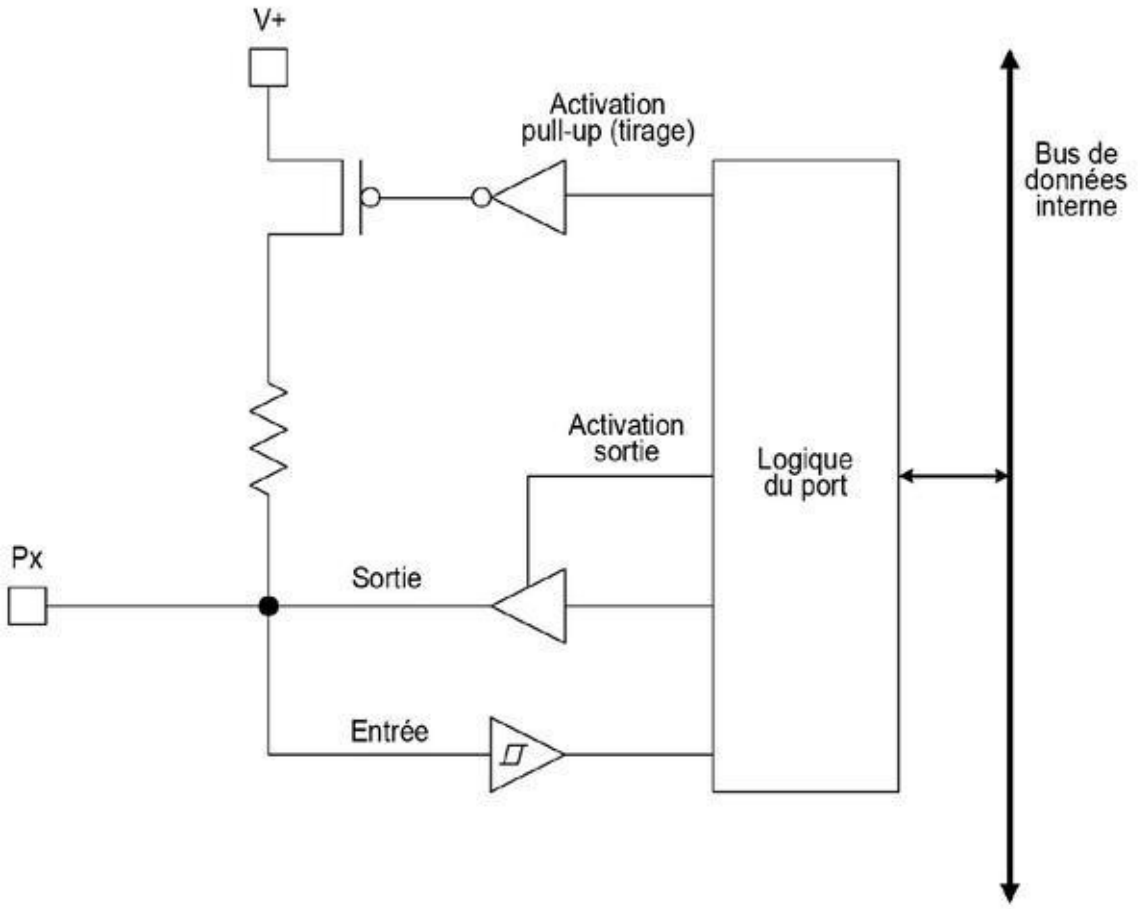


Figure 2.3 : Schéma d'une broche de port d'entrées-sorties.

Grâce à la logique de contrôle sophistiquée, chaque port AVR est capable de remplir des fonctions très différentes, certaines même simultanément. Il est par exemple possible de lire des données depuis une broche d'un port qui est configurée en tant que sortie ; de même, une broche en sortie peut servir à déclencher une interruption (les interruptions sont présentées en fin de chapitre).

Les timers et décompteurs sur 8 bits

Les microcontrôleurs AVR proposent deux types de timer/compteur sur 8 bits. Dans le premier, la fréquence d'horloge utilisée est déduite de l'horloge du processeur, ce qui fait qu'elle est synchrone. Le deuxième type permet de fonctionner de façon asynchrone, en utilisant une source d'horloge externe. La Figure 2.4 montre le schéma d'un timer/compteur. Les registres qui permettent de contrôler cette fonction sont décrits dans l'Annexe B.

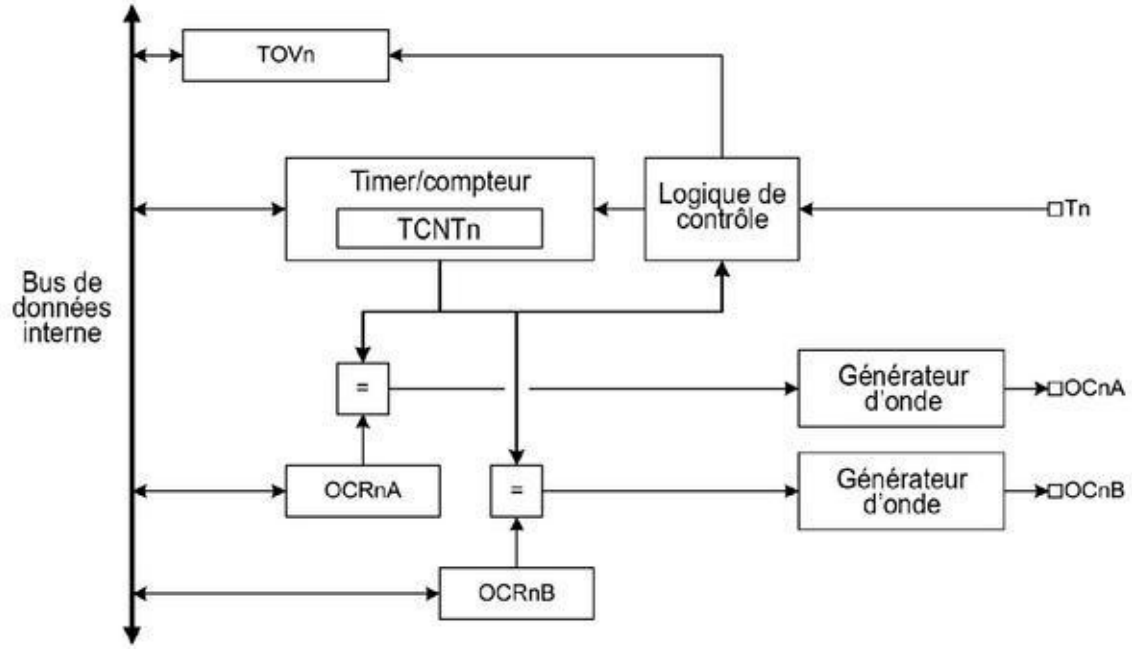


Figure 2.4 : Diagramme d'un timer/compteur AVR.

Le premier module sur 8 bits, `TimerCounter0`, est un chronomètre et compteur à usage général qui offre deux circuits indépendants de comparaison des sorties, et quatre modes de fonctionnement. Voici ces quatre modes :

Mode normal

C'est le mode de fonctionnement le plus simple. Le compteur augmente de un en permanence, et il n'est jamais effacé lorsqu'il atteint sa valeur maximale sur 8 bits. En revanche, quand le compteur déborde, il revient à la valeur zéro, ce qui arme le drapeau de débordement TOV0. Vous pouvez considérer ce drapeau comme un neuvième bit, mais il ne peut jamais être remis à zéro, mais seulement forcé à 1 par débordement. C'est l'interruption déclenchée par le débordement du timer qui force automatiquement à zéro le drapeau de débordement. Cette interruption peut donc servir à faire progresser un deuxième compteur logiciel en mémoire. Vous pouvez écrire une nouvelle valeur à tout moment dans le registre de compteur TCNT0.

Mode CTC (Client Timer Compare)

Dans ce mode, la résolution du compteur, c'est-à-dire la valeur maximale, est définie grâce au registre OCRA0. Cela permet d'obtenir un contrôle plus précis

de la fréquence de sortie du comparateur, ce qui simplifie le comptage des événements d'un événement externe.

Mode PWM rapide

Ce mode PWM (modulation de largeur d'impulsion) rapide permet de générer des signaux PWM à haute fréquence.

Mode PWM avec phase correcte

Ce dernier mode permet de générer un signal PWM avec une correction de phase à haute résolution.

Certains modèles de microcontrôleurs AVR comportent en outre un timer/compteur 8 bits capable de fonctionner de façon asynchrone en se servant d'une horloge externe sur les broches d'entrée TOSC1 et TOSC2. Le fonctionnement équivaut à celui du timer/compteur 8 bits synchrone décrit plus haut.

Timer/compteur sur 16 bits

Un timer/compteur sur 16 bits se distingue d'une version sur 8 bits par une plage de comptage bien plus étendue. Il s'agit ici de 16 bits véritables, ce qui permet de générer un signal PWM à partir d'une variable codée sur 16 bits. Le module est doté de deux circuits de comparaison de sortie indépendants, de registres de comparaison de sortie avec double tampon (*buffer*) et d'un circuit de capture d'entrée doté d'un réducteur de bruit. En effet, le module permet également de faire de la capture d'événements externes à haute résolution, de la génération de fréquences et du chronométrage de signaux. Il peut générer quatre interruptions différentes : TOV1, OCF1A, OCF1B et ICF1.

Démultiplicateur de timer/compteur (prescaler)

Dans un contrôleur AVR, plusieurs compteurs peuvent partager le même circuit logique de démultiplication, avec des paramètres différents. Le démultiplicateur (*prescaler*) est un diviseur qui permet d'obtenir un sous-multiple de la fréquence

d’horloge f des entrées-sorties : f/8, f/64, f/256 ou f/1024. Le top d’horloge résultant est nommé *tap*. Un compteur peut ainsi travailler au 64^{e} pendant que l’autre travaille au 1024^{e}. Cette démultiplication permet d’étendre la plage de comptage pour mieux correspondre à la fréquence de survenue d’un événement externe, sans compter que cela allonge le temps entre deux débordements et retour à zéro du compteur.

Comparateur analogique

Le circuit de comparaison analogique d'un AVR sert à comparer les tensions présentes sur les deux broches d'entrée AIN0 et AIN1. La première, AIN0, est considérée comme l'entrée positive et AIN1 comme l'entrée négative, mais ce sont des valeurs relatives. Il n'y a jamais de tension négative en entrée. La Figure 2.5 donne le schéma simplifié du circuit du comparateur analogique.

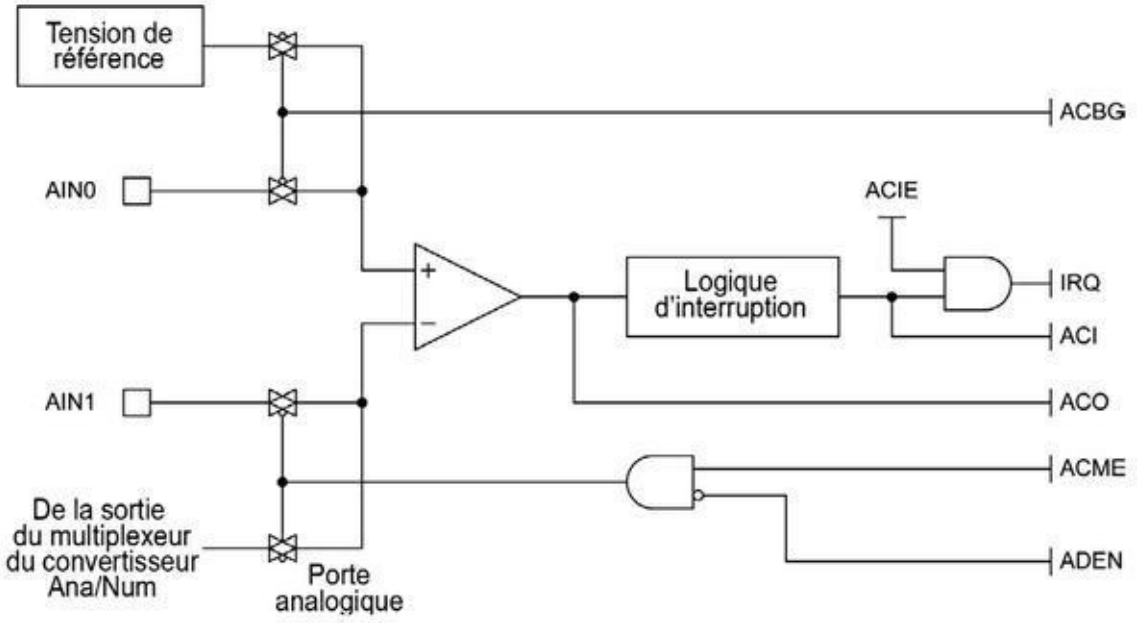


Figure 2.5 : Schéma du comparateur analogique AVR.

Lorsque l'entrée AIN0 montre une valeur supérieure à AIN1, la logique du comparateur force à l'état Haut le drapeau du comparateur ACO. La sortie du comparateur peut être configurée pour déclencher la fonction de capture d'entrée d'un des modules timer/compteur. Il permet également de générer une interruption de comparaison. L'événement peut être configuré pour se déclencher sur front montant ou descendant, ou sur inversion de la sortie du comparateur.

Le comparateur analogique ne se limite pas à comparer les tensions d'AIN0 et AIN1. L'entrée peut être configurée pour que l'entrée AIN1 soit comparée à une tension de référence interne (*bandgap*). On peut aussi faire comparer AIN0 à la sortie du multiplexeur du convertisseur ana/num. La tension reste néanmoins disponible à l'entrée du convertisseur. Dans le schéma de la Figure 2.5, les petits symboles avec des triangles et des losanges correspondent à des portes

analogiques. Le petit cercle de ces symboles indique la façon dont chaque porte réagit : si c’est l’entrée inverseuse qui est utilisée, le signal analogique va passer lorsque le contrôle est à l’état Bas. Dans le cas contraire, il passera lorsque le contrôle sera à l’état Haut.

Convertisseur analogique/numérique (CAN ou ADC)

Selon le modèle, les microcontrôleurs AVR contiennent un convertisseur sur 8, sur 10 ou sur 12 bits pour convertir des signaux analogiques. Les versions 8 bits sont celles des modèles ATtiny6 et ATtiny10. Notez que certaines versions dédiées au monde de l'automobile n'ont pas de convertisseur CAN.

Les convertisseurs AVR ont de 4 à 28 entrées. Le nombre d'entrées disponibles dépend d'abord du format du boîtier physique. Chaque entrée est sélectionnée individuellement au moyen d'un multiplexeur interne. Autrement dit, toutes les entrées ne sont pas actives en même temps. Certaines des broches d'entrées-sorties qui servent à la conversion par le multiplexeur peuvent être affectées à d'autres fonctions.

Le modèle ATmega168 offre soit 6, soit 8 canaux d'entrée analogiques (selon le boîtier). Le boîtier rectangulaire plastique PDIP offre un convertisseur sur 10 bits avec six canaux d'entrée. Les boîtiers à montage en surface TQFB et QFN/MFL offrent aussi 10 bits, mais avec huit canaux d'entrée. La Figure 2.6 donne un schéma de la fonction de conversion analogique vers numérique AVR.

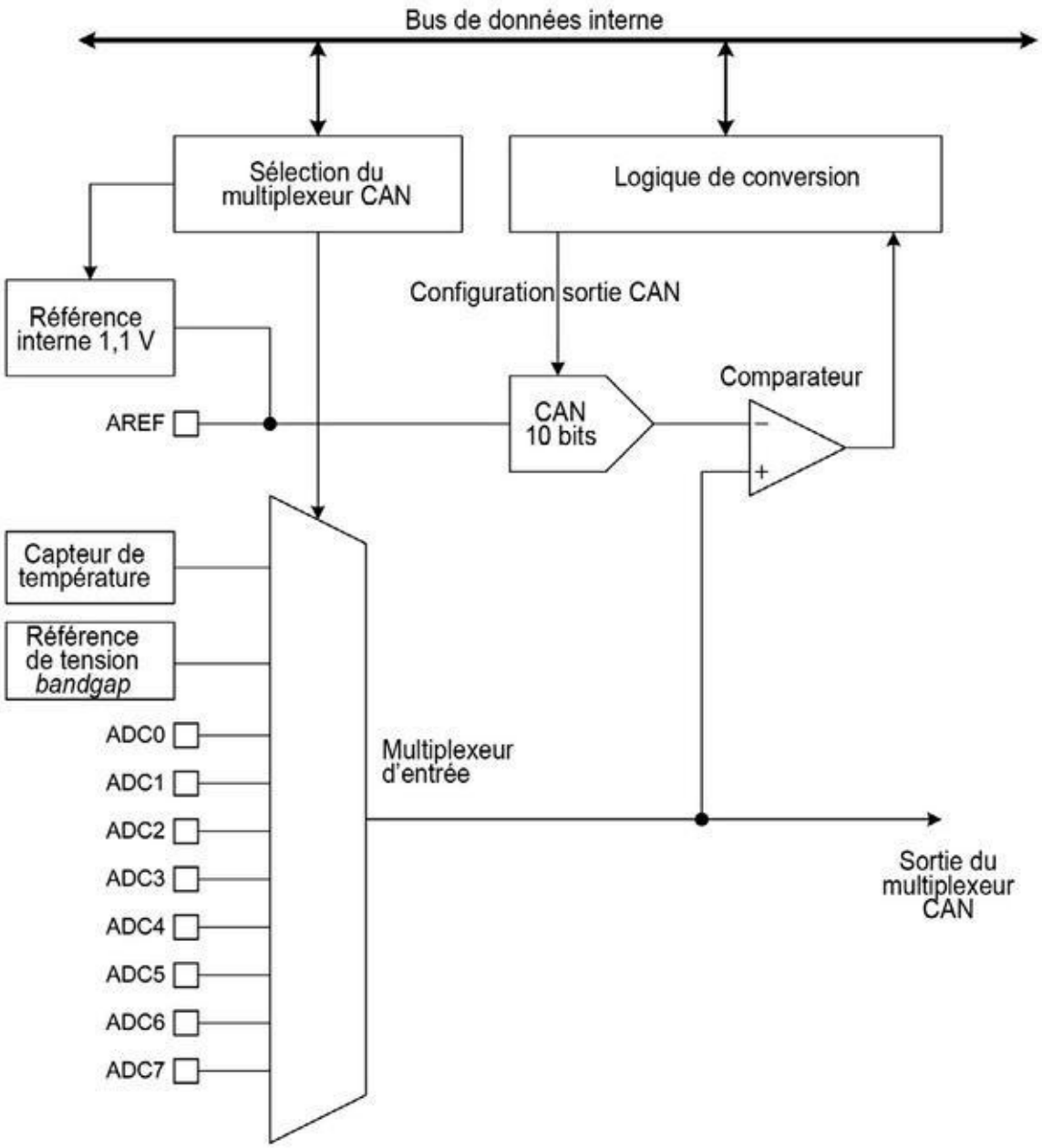


Figure 2.6 : Schéma du convertisseur analogique/numérique AVR.

Les microcontrôleurs AVR utilisent des convertisseurs à approximations successives. Ils ne sont pas très rapides, mais ils sont faciles à mettre au travail, car ils n'ont besoin que d'un convertisseur CAN et d'un comparateur. En fonctionnement libre, avec la résolution maximale, le temps de conversion d'un signal sur 10 bits est d'environ 65 µs (microsecondes) par échantillon.

Entrées-sorties série

Les microcontrôleurs tels que l'ATmeg168 proposent trois types d'interface série : série synchrone ou asynchrone, série synchrone maître/esclave SPI et série sur deux fils octet par octet, similaire au standard Philips I2C.

Communications série USART

La plupart des contrôleurs AVR sont dotés d'un émetteur-récepteur universel synchrone et asynchrone, USART, que l'on désigne plus souvent par UART. Cette fonction permet d'établir une communication de type RS-232 ou RS-485, mais aussi de permettre à des circuits de communiquer sans nécessiter d'ajouter une logique d'interface externe. La vitesse de transmission, en bauds, dépend de la fréquence d'horloge du microcontrôleur ; la vitesse standard est égale à 9600. Il est possible d'atteindre des vitesses supérieures avec un cristal d'horloge externe. Le module USART peut également fonctionner en mode SPI (*Serial Peripheral Interface*), en complément du circuit logique dédié à l'interface SPI. La Figure 2.7 montre les principaux composants de la fonction de communication série USART.

Interface SPI

Les circuits de contrôle SPI des contrôleurs AVR savent gérer les quatre modes de fonctionnement standard SPI. Les broches d'entrées-sorties AVR peuvent être configurées pour gérer les trois signaux MOSI, MISO et SCK. Elles ne sont pas celles qui servent à la liaison série USART (RxD pour recevoir et TxD pour émettre des données). La Figure 2.8 propose une vue générale de la logique d'interface SPI.

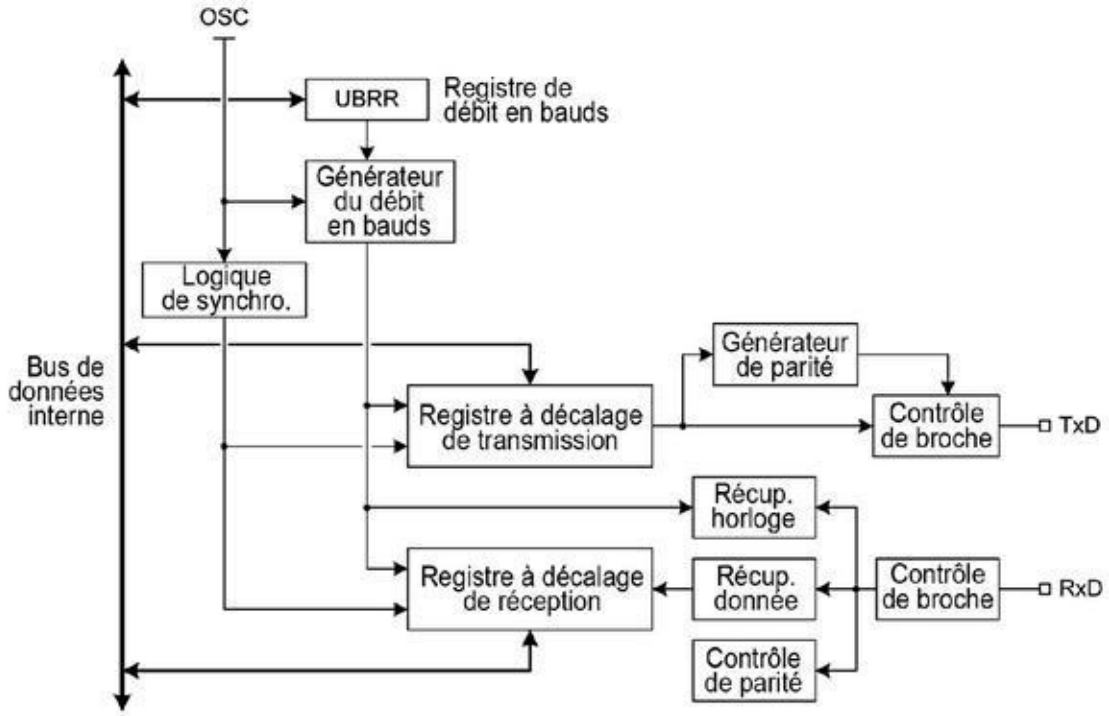


Figure 2.7 : Schéma de la fonction USART.

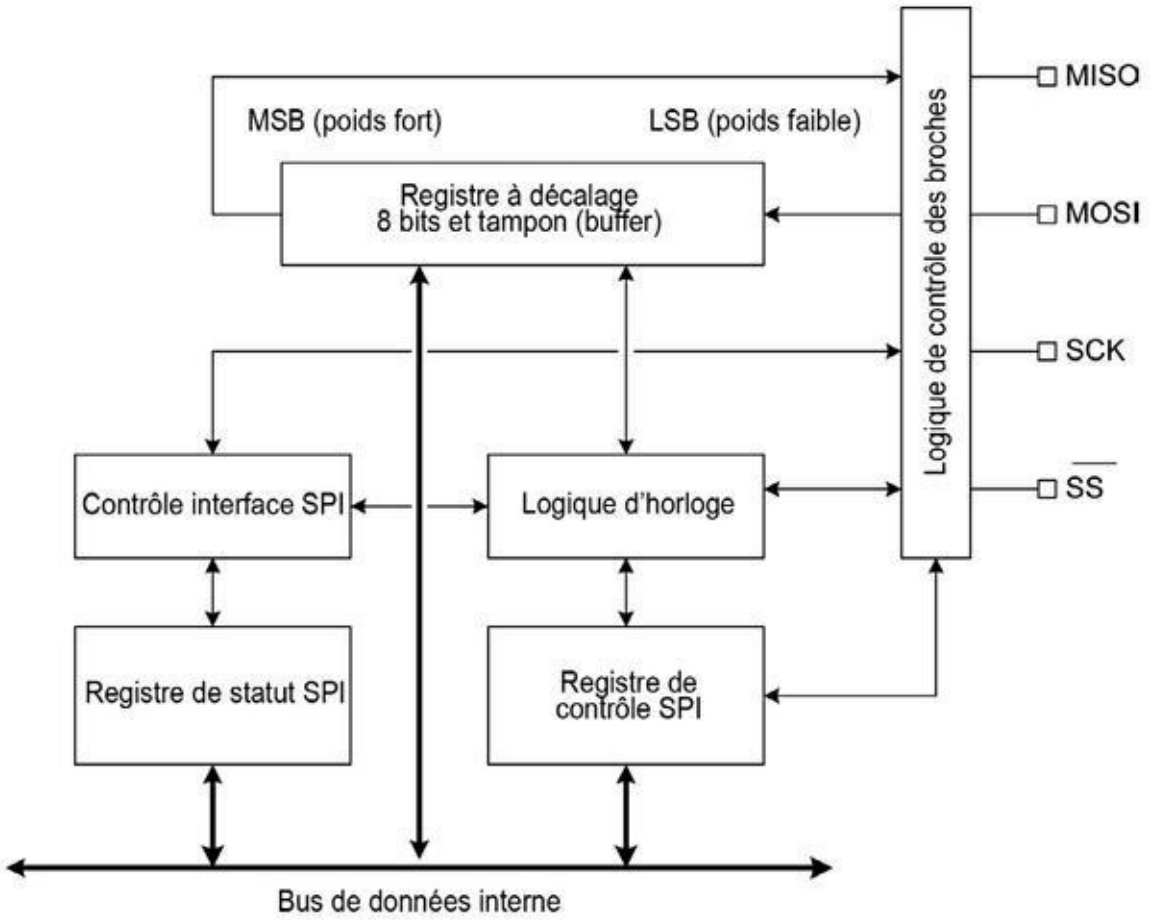


Figure 2.8 : Schéma de l'interface SPI.

Interface TWI

Le troisième type de communication série que gèrent la plupart des microcontrôleurs AVR correspond à l'interface sur deux fils TWI (*Two Wire Interface*), compatible avec le protocole I2C de Philips. Cette interface autorise le mode maître et le mode esclave, avec des adresses de périphérique sur 7 bits. La vitesse peut aller jusqu'à 400 kHz avec un arbitrage du bus par plusieurs maîtres. Il est possible de faire générer une condition de réveil si le contrôleur est en mode veille. Au niveau interne, cette fonction est bien plus complexe que ses collègues USART et SPI. La Figure 2.9 donne un aperçu de l'interface TWI/I2C.

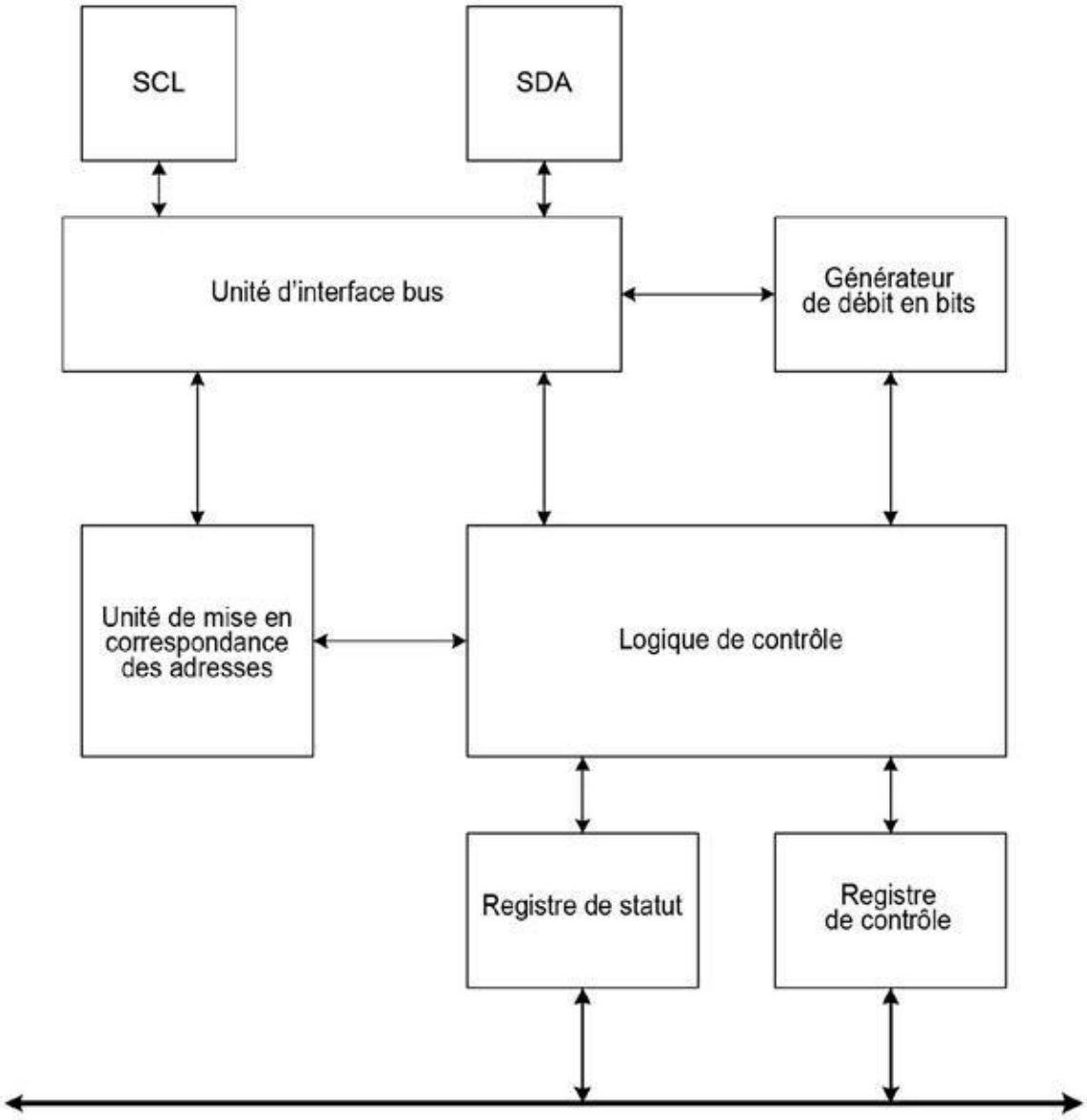


Figure 2.9 : Schéma de l'interface de communication TWI ou I2C.

Interruptions

Les interruptions forment une caractéristique indispensable des processeurs modernes, puisqu'elles lui permettent de réagir à des événements internes ou externes qui surviennent sans prévenir, en faisant rediriger le flux d'exécution vers un bloc de code de gestion d'interruption. Une fois ce bloc exécuté, le programme reprend où il en était au moment de l'interruption. Dans les processeurs AVR, vous pouvez activer ou inhiber la réponse à une interruption en modifiant des bits dans les registres de contrôle. La description qui suit est spécifique au modèle ATmega168. Pour les autres modèles, vous irez d'abord consulter les tableaux de l'Annexe B, puis la documentation officielle Atmel.

Le modèle ATmega168 propose deux entrées pour interruptions externes, INT0 et INT1. Vous pouvez les configurer pour provoquer une interruption sur un front montant, un front descendant ou un état Bas. Le registre de contrôle nommé EICRA permet de contrôler le comportement exact. Les deux entrées INT0 et INT1 ont besoin d'une horloge d'entrées-sorties. Le mode interruption sur état Bas génère une interruption de façon répétée tant que l'entrée correspondante est maintenue à l'état Bas.

Les broches d'entrées-sorties de l'ATmega168 peuvent également servir de sources d'interruption. Les interruptions sur changement de port portent le nom PCINT0 jusqu'à PCINT23, chacune étant associée à une des 24 broches d'entrées-sorties. Lorsqu'une interruption est autorisée, elle est déclenchée lorsque l'état de la broche change, même si cette broche est configurée comme sortie. C'est ce qui permet à une broche de générer une interruption sous le contrôle du logiciel lorsque le programme lui-même provoque le changement d'état de la broche (bien sûr, il faut que la détection d'interruption sur changement d'état du port soit active).

Dès qu'une des broches entre PCINT0 et PCINT7 change d'état, elle déclenche une interruption PC0. Pour les broches PCINT8 à PCINT14, cela correspond à l'interruption PC1. Enfin, les broches de PCINT16 à PCINT23 correspondent à l'interruption PC2. La configuration des broches qui déclenchent les interruptions en cas de changement d'état est réalisée par les registres PCMSK0, PCMSK1 et PCMSK2.

Quand une interruption qui n'est pas masquée survient, le processeur va chercher la prochaine instruction à exécuter à l'adresse trouvée dans une table de vecteurs

en mémoire ; cette adresse contient une instruction machine de saut RJMP qui pointe vers le bloc de code incarnant la réaction à l'interruption. Une fois ce bloc exécuté, l'exécution reprend dans le programme normal, juste après l'interruption. La Figure 2.10 montre comment la table des vecteurs d'interruption est exploitée pour rediriger le flot d'exécution vers le code de l'interruption puis revenir au programme principal.

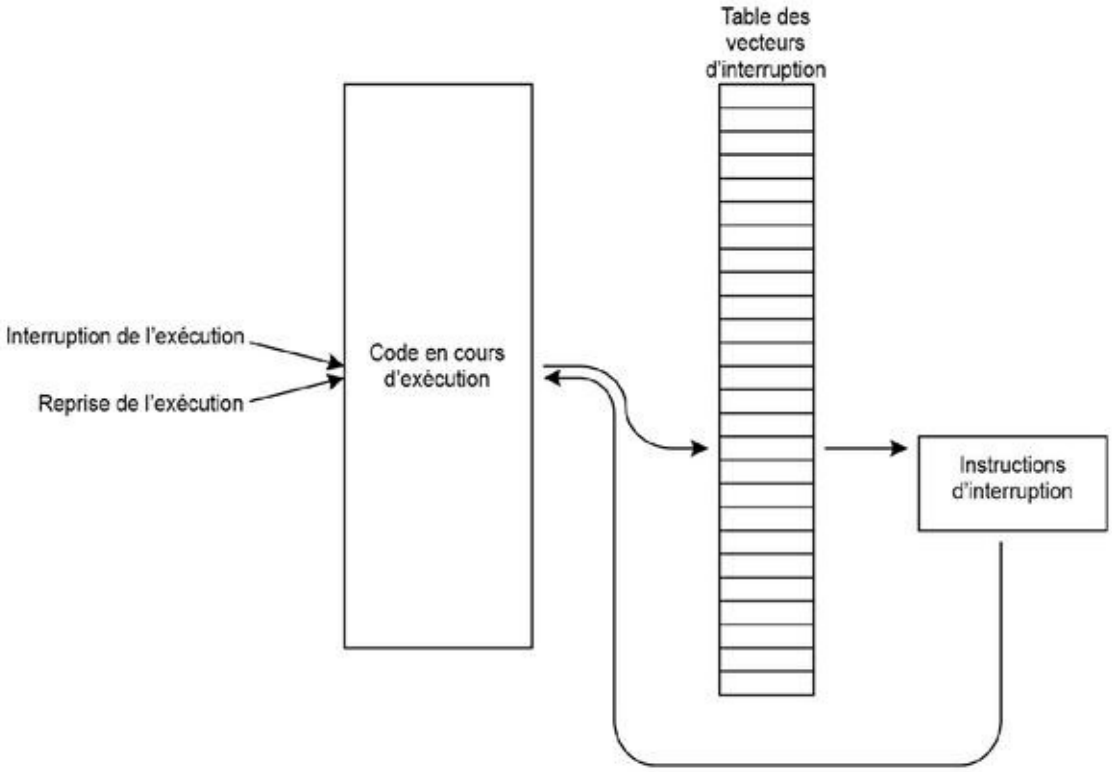


Figure 2.10 : Fonctionnement de la table des vecteurs d'interruption d'un ATmega168/328.

Par exemple, un ATmega168 possède une table de vecteurs de 26 entrées, comme vous pouvez le voir dans le Tableau 2.1. Pour d'autres modèles de processeur, vous irez consulter la documentation Atmel au niveau des interruptions et de leur gestion.

Tableau 2.1 : Vecteurs d'interruption d'un ATmega168.

Vecteur	Adresse	Source	Description
1	0x0000	RESET	Broche externe, power-on, brown-out (sous-tension) et watchdog

2	0x0002	INT0	Requête d’interruption externe 0
3	0x0004	INT1	Requête d’interruption externe 1
4	0x0006	PCINT0	Requête d’interruption sur changement 0
5	0x0008	PCINT1	Requête d’interruption sur changement 1
6	0x000A	PCINT2	Requête d’interruption sur changement 2
7	0x000C	WDT	Interruption sur fin de décompte watchdog (time-out)
8	0x000E	TIMER2 COMPA	Détection comparaison A de Timer/Counter2
9	0x0010	TIMER2 COMPB	Détection comparaison B de Timer/Counter2
10	0x0012	TIMER2 OVF	Débordement de Timer/Counter2
11	0x0014	TIMER1 CAPT	Capture d’événement Timer/Counter1
12	0x0016	TIMER1 COMPA	Détection comparaison A de Timer/Counter1
13	0x0018	TIMER1 COMPB	Détection comparaison B de Timer/Counter1
14	0x001A	TIMER1 OVF	Débordement de Timer/Counter1

15	0x001C	TIMER0 COMPA	Détection comparaison A de Timer/Counter0
16	0x001E	TIMER0 COMPB	Détection comparaison B de Timer/Counter0
17	0x0020	TIMER0 OVF	Débordement de Timer/Counter0
18	0x0022	SPI, STC	Fin de transfert série SPI
19	0x0024	USART, RX	Fin de réception USARTRx
20	0x0026	USART, UDRE	USART, registre de données vide
21	0x0028	USART, TX	Fin de transmission USARTTx
22	0x002A	ADC	Conversion ana/num terminée
23	0x002C	EE READY	EEPROM prête
24	0x002E	ANALOG COMP	Comparateur analogique
25	0x0030	TWI	Interface série TWI
26	0x0032	SPM READY	Mémoire de stockage programme prête (*Store Program Memory*)

Watchdog timer (chien de garde)

Les microcontrôleurs AVR possèdent un circuit dit de chien de garde (*watchdog*) dont le temps d'écoulement est configurable entre 16 et 8 000 millisecondes (8 secondes).

Lorsque ce circuit est activé, vous pouvez vous en servir pour faire générer en fin de décompte une remise à zéro Reset ou une interruption, ou une combinaison des deux. Ce circuit utilise un oscillateur indépendant intégré, ce qui lui permet de continuer à compter même lorsque le microcontrôleur a basculé en mode veille. Ce chien de garde porte donc bien son nom, puisqu'il permet de réveiller le microcontrôleur au bout d'un certain temps.

Une autre utilisation habituelle d'un chien de garde consiste à forcer un Reset ou à déclencher une interruption sans requérir une action de Reset explicite par le logiciel. Cela permet par exemple de faire sortir le microcontrôleur d'une boucle infernale, ou de le libérer d'un code dont l'exécution est devenue incohérente.

Caractéristiques électriques

Les contrôleurs ATmega168/328 peuvent fonctionner avec une tension continue entre 1,8 et 5,5 V. En comparaison, l'ATmega32U4 accepte une tension entre 2,7 et 5,5 V.

Le courant consommé dépend évidemment du type de contrôleur, de son état actif ou en veille et de la vitesse d'horloge. Les valeurs pour toute la gamme de produits ATmega et Xmega sont de 0,55 mA à 30 mA. Le courant total consommé dépend aussi du courant que doivent fournir les broches de sortie. Dans le Chapitre 3, je donne des valeurs plus détaillées pour les modèles ATmega168/328, ATmega1280/2560 et ATmega32U4.

Pour aller plus loin

Le Chapitre 3 se consacre à la description des brochages pour les trois modèles de microcontrôleurs les plus utilisés sur les cartes Arduino. Le Chapitre 4 montre comment les broches sont associées aux entrées-sorties des différentes cartes Arduino qui hébergent ces microcontrôleurs.

CHAPITRE 3

Les microcontrôleurs AVR pour Arduino

Ce chapitre propose une description fonctionnelle des microcontrôleurs AVR utilisés dans les cartes Arduino. Nous partons des descriptions générales fournies dans le chapitre précédent pour plonger dans les détails de trois des microcontrôleurs les plus utilisés : l'ATmega168/328, l'ATmega1280/2560 et l'ATmega32U4.

Du point de vue du programmeur qui réalise un projet Arduino, le microcontrôleur constitue une représentation abstraite du véritable circuit matériel. Le code source qui permet de réaliser les différentes opérations, comme par exemple la configuration d'une broche pour générer un signal PWM ou la mise en place d'un canal analogique d'entrée pour le convertisseur, est identique d'un modèle à l'autre. Les adresses internes des registres de contrôle et les bits de contrôle sont tous prédéfinis ; le programmeur n'a pas à se soucier des détails à bas niveau.

Du fait qu'une carte Arduino n'est pour l'essentiel qu'une mise à disposition plus confortable des broches de sortie du circuit intégré AVR, les caractéristiques électriques de la carte sont celles du contrôleur. Les broches du contrôleur sont en effet directement connectées aux points de sortie (connecteurs ou plots à souder). Il n'y a pas de mise en tampon intermédiaire, ni d'amplification des signaux entre la puce et les sorties de la carte.

À l'heure où j'écris ces lignes, les cartes Arduino utilisent moins d'une douzaine de microcontrôleurs ATmega différents, comme le montre le Tableau 3.1. Ce qui distingue les différents modèles est d'abord la quantité de mémoire flash disponible, la vitesse d'horloge, le nombre de broches d'entrées-sorties et le nombre de fonctions périphériques internes. Le modèle ATmega32U4 se distingue des autres du fait qu'il incorpore l'interface USB, ce qui évite d'ajouter un circuit dédié à cette fonction. Le jeu d'instructions du processeur est le même pour tous les modèles.

Tableau 3.1 : Les microcontrôleurs AVR des cartes Arduino.

Microcontrôleur	Mém. flash	Broches E/S	Commentaires
ATmega168	16 ko	23	Horloge 20 MHz
ATmega168V	16 ko	23	Horloge 10 MHz
ATmega328	32 ko	23	Horloge 20 MHz
ATmega328P	32 ko	23	Horloge 20 MHz, picoPower
ATmega328V	32 ko	23	Horloge 10 MHz
ATmega1280	128 ko	86	Horloge 16 MHz
ATmega2560	256 ko	86	Horloge 16 MHz
ATmega32U4	32 ko	26	Horloge 16 MHz

Un programme qui a été écrit au départ pour le modèle Arduino Diecimila doit, sauf exception, pouvoir être compilé puis exécuté sur une carte Uno. La principale différence entre ces deux cartes est l'utilisation du microcontrôleur ATmega168 sur la première, alors que l'autre bénéficie du ATmega328. Le Tableau 3.1 montre que ce second contrôleur offre deux fois plus de mémoire flash. Le seul cas dans lequel un programme écrit pour une carte Uno ne pourra pas fonctionner sur une Diecimila plus ancienne est que le code exécutable a besoin d'un espace mémoire supérieur à ce qui est disponible.

ATmega168/328

Ces deux circuits ne se distinguent que par la quantité de mémoire. Le diagramme fonctionnel de l'ATmega168/328 est montré dans la Figure 3.1.

Espace mémoire

Le microcontrôleur ATmega328 offre deux fois plus d'espace mémoire que son prédécesseur ATmega168 (Tableau 3 2).

Tableau 3.2 : Mémoire des microcontrôleurs ATmega168 et ATmega328.

	ATmega168	ATmega328
Mémoire flash	16 ko	32 ko
EEPROM	512 octets	1 ko
RAM	1 ko	2 ko

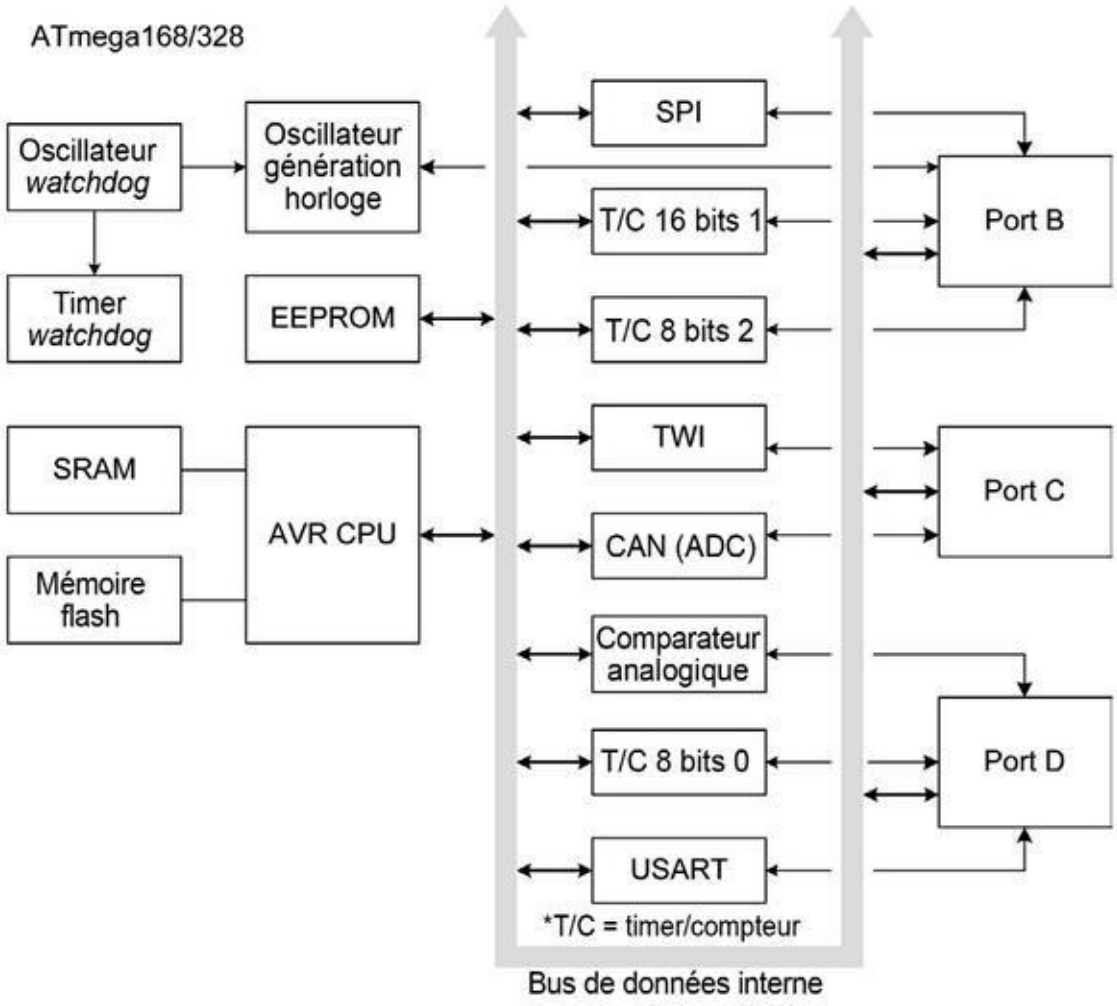


Figure 3.1 : Diagramme fonctionnel d'un microcontrôleur ATmega168/328.

Fonctions principales

Voici les fonctions principales communes à ces deux microcontrôleurs :

- implantation du programme par utilitaire d'amorçage sur le circuit (*In-System Programming*) ;
- deux timers/compteurs sur 8 bits avec démultiplicateurs indépendants et mode comparaison ;
- un timer/compteur sur 16 bits avec démultiplicateur indépendant, mode comparaison et mode capture ;

- un compteur en temps réel avec oscillateur individuel ;
- six canaux de sortie à modulation de largeur d’impulsion PWM ;
- six ou huit (selon le boîtier) canaux de conversion analogique/numérique sur 10 bits ;
- communication série USART ;
- interface série maître-esclave SPI ;
- interface série sur deux fils compatible Philips I2C ;
- un timer chien de garde programmable (*watchdog*) ;
- un comparateur analogique ;
- 23 lignes d’entrées-sorties programmables.

Boîtiers disponibles

Les deux contrôleurs sont disponibles dans quatre boîtiers physiques différents : double rangée à 28 broches DIP, à soudure en surface à 28 broches, à soudure en surface TQFP à 32 broches ou à soudure en surface MLF à 32 broches. Le format le plus utilisé est le boîtier DIP à 28 broches. La carte Uno SMD utilise un boîtier soudé en surface à 32 broches. Dans la suite, nous nous concentrons sur la version classique DIP à 28 broches.

Ports d’entrées-sorties

Le contrôleur ATmega168/328 offre trois ports désignés B, C et D. Les ports B et D sont sur 8 bits. Le port C comporte six broches pouvant servir d’entrées analogiques pour le convertisseur analogique/numérique (CAN). Les broches PC4 et PC5 sont en outre reliées au circuit de gestion de l’interface TWI et peuvent fournir des signaux compatibles I2C, SCL et SDA, c’est-à-dire l’horloge et les données. La broche PC6 sert aussi d’entrée de réinitialisation Reset. Il n’y a pas de broche PC7 et pas non plus de port A dans ces deux contrôleurs.

Chaque port est doté de broches bidirectionnelles numériques associées à des résistances de tirage (*pull-up*) internes débrayables. L'état d'activation de ces résistances est géré au moyen de bits dans les registres de contrôle.

Les tampons de sortie des ports peuvent fonctionner en sources comme en puits de courant. Lorsqu'elles sont utilisées en entrées, les broches sont forcées à l'état Bas si les résistances de tirage internes sont actives ou dans un état à haute impédance lorsque la condition de réinitialisation est activée (même lorsque l'horloge est arrêtée).

Affectation des broches

La Figure 3.2 montre les multiples affectations des broches dans le boîtier DIP à 28 broches. Pour obtenir le même schéma pour les autres boîtiers, vous vous reporterez au site du fabricant, Atmel.

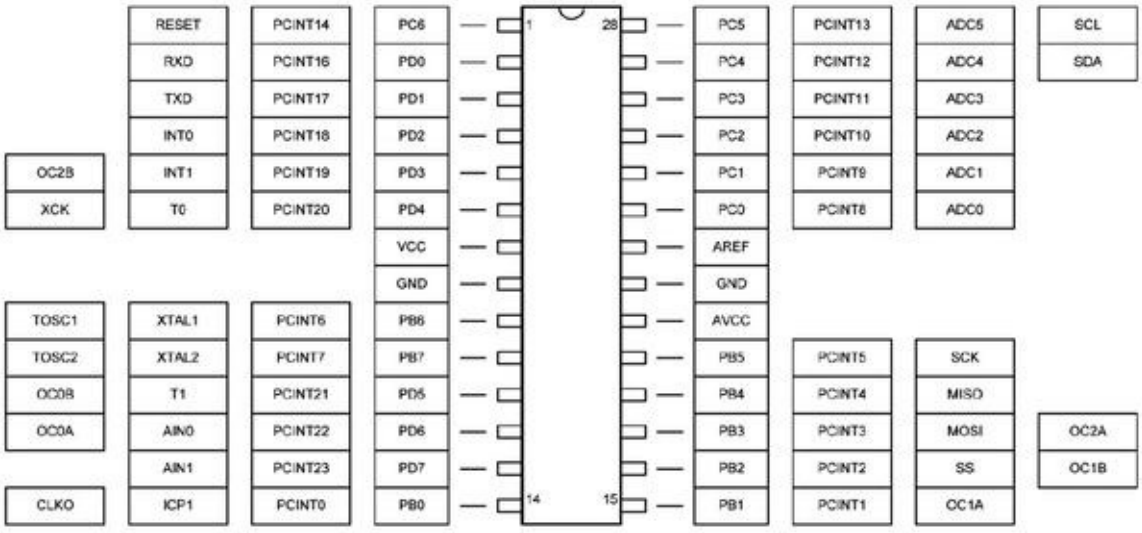


Figure 3.2 : Affectation des broches du boîtier DIP de l'ATmega168/328.

Broches d'entrée du comparateur analogique

La Figure 3.3 montre la position des broches AIN0 et AIN1 en boîtier DIP. Sachez que la broche AIN0 sert aussi à la sortie OC0A du timer/compteur (PD6). Autrement dit, lorsque PD6 sert de sortie PWM, elle ne peut pas servir d'entrée AIN0, sauf à reconfigurer la broche dans le programme avant chaque changement de fonction.

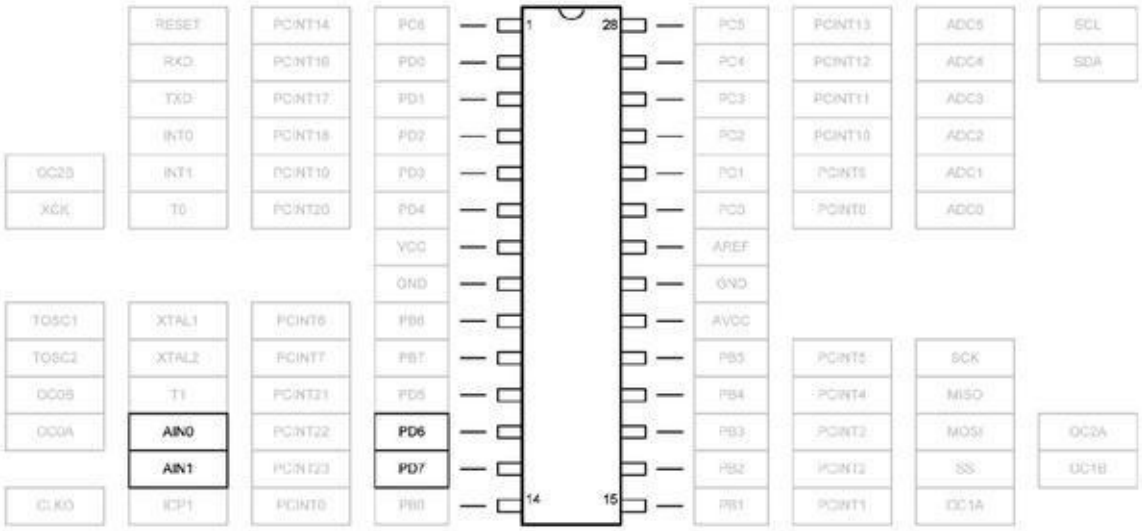


Figure 3.3 : Broches d'entrée du comparateur analogique ATmega168/328.

Entrées analogiques

À l'exception des deux broches SCL et SDA utilisées pour les communications TWI, les autres broches d'entrée analogique peuvent servir soit d'entrées ou de sorties numériques, soit d'entrées analogiques sans entraîner de conflit avec les autres fonctions périphériques AVR. La Figure 3.4 montre quelles broches en boîtier DIP servent aux entrées analogiques.

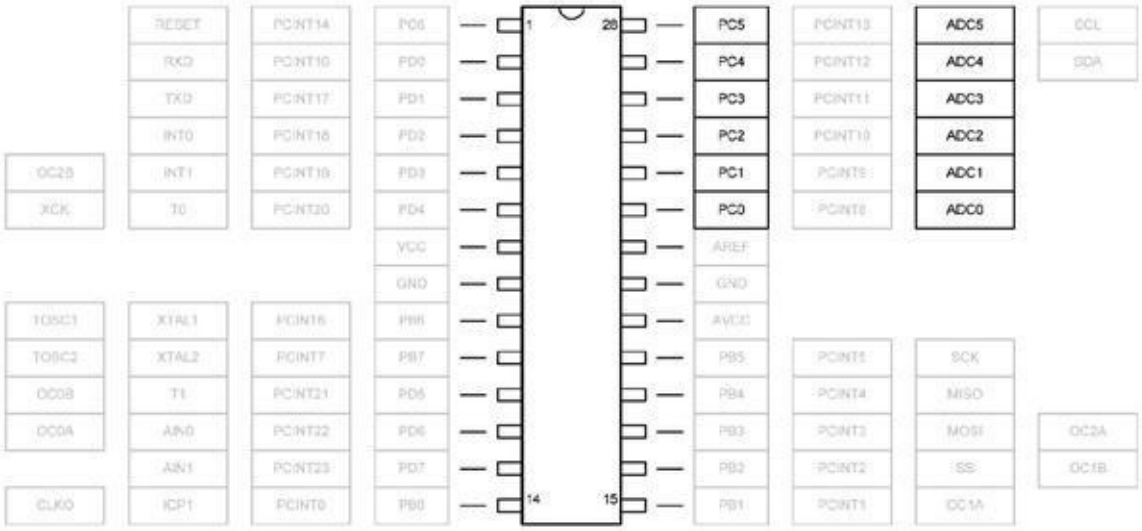


Figure 3.4 : Broches d'entrée du convertisseur analogique des ATmega168/328.

Interfaces série

La Figure 3.5 montre quelles broches servent (en boîtier DIP) aux différentes fonctions d'interface série. Les trois formats peuvent utiliser sans craindre de conflit parce qu'aucune des broches n'est partagée entre deux fonctions de communication.

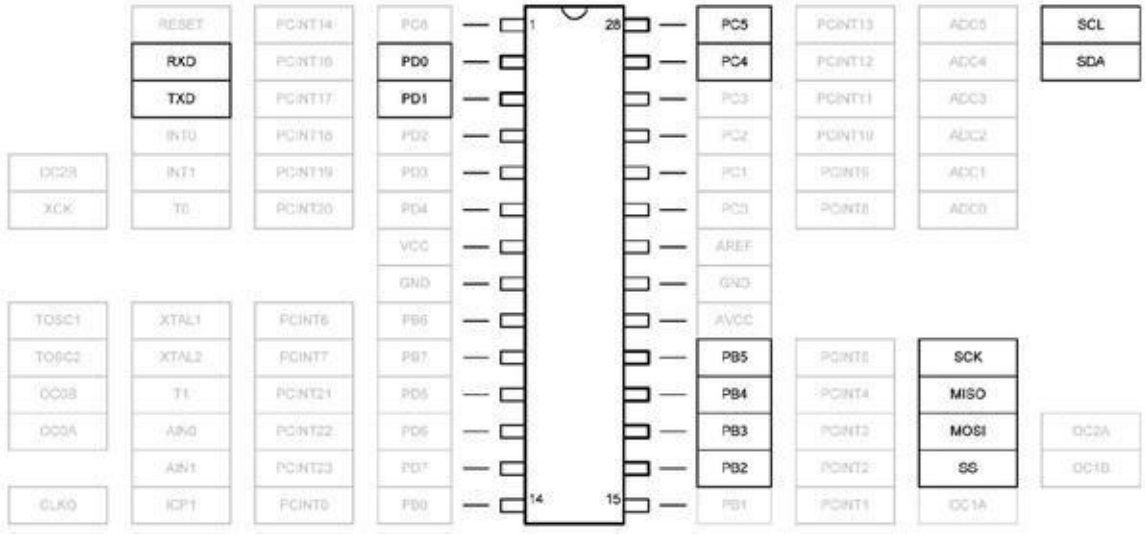


Figure 3.5 : Broches d'entrées-sorties série des ATmega168/328.

Entrées-sorties des timers et de l'horloge

La logique interne des circuits des timers/compteurs est assez complexe, ce dont témoignent les affectations des broches de la Figure 3.6. Les broches de la famille OCxn, c'est-à-dire OX0A, OC0B, OC1A, OC1B, OC2A, OC2B) peuvent servir de sorties PWM, et c'est avec cette fonction qu'elles sont repérées sur le circuit imprimé Arduino. Notez que T1 et OC0B partagent la même broche PD5. Les autres broches PWM peuvent être utilisées sans crainte de conflit avec les autres fonctions de comptage.

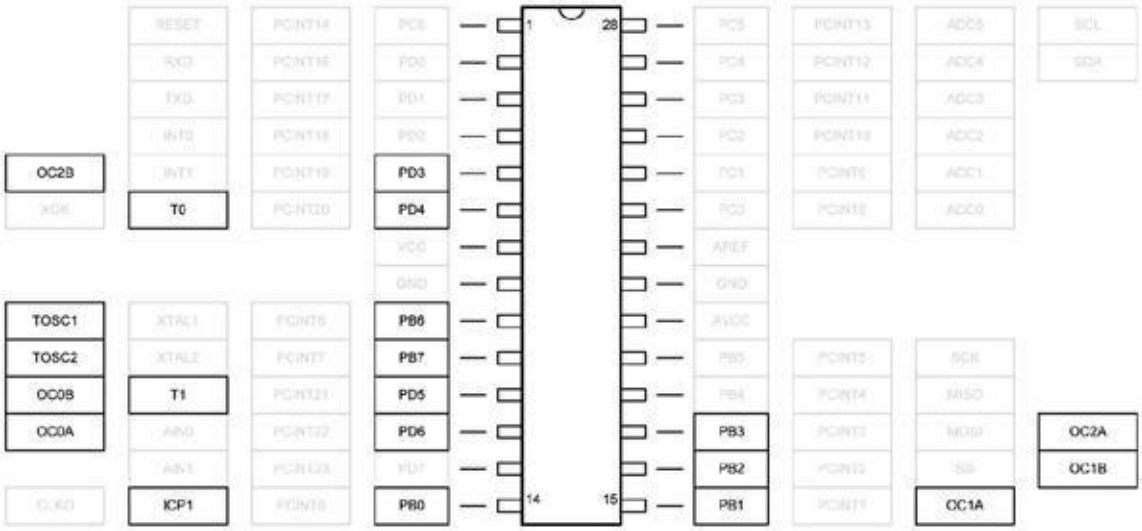


Figure 3.6 : Broches des timers/compteurs ATmega168/328.

Interruptions externes

Les deux broches PD2 et PD3 du port D sont dédiées à l'utilisation en tant qu'entrées pour des interruptions externes. De plus, toutes les broches PCINT0 jusqu'à PCINT23 peuvent servir au même rôle sans provoquer d'interférence avec d'autres fonctions. (Revoyez si nécessaire la section décrivant les interruptions à la fin du chapitre précédent.) La Figure 3.7 montre quelles entrées sont disponibles pour les interruptions sur ce type de microcontrôleur.

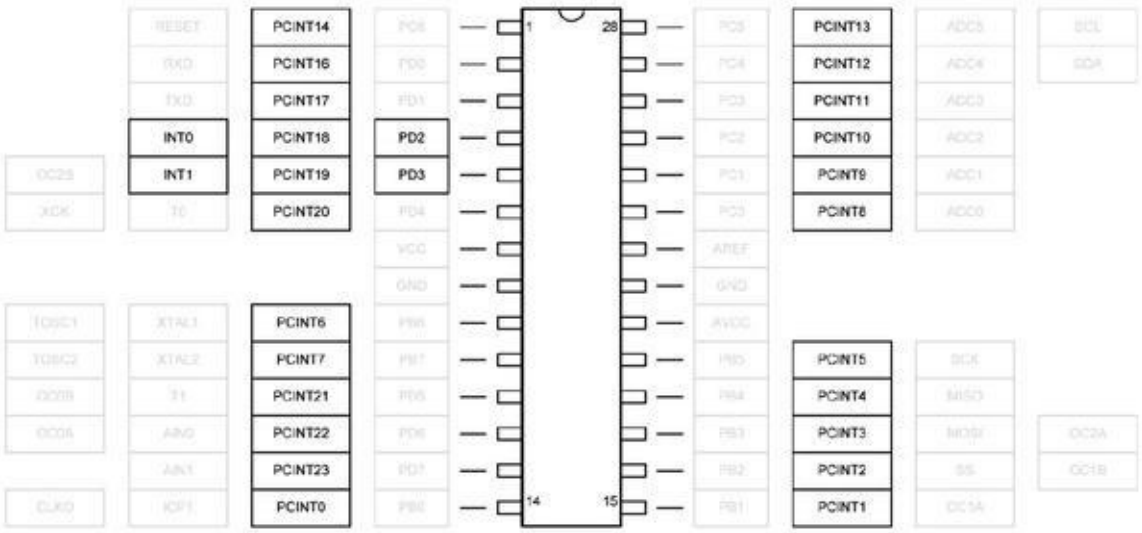


Figure 3.7 : Broches d'entrée d'interruption ATmega168/328.

Repérage des connecteurs sur le circuit imprimé

Les cartes Diecimila, Duemilanove, Uno et Ethernet utilisent les mêmes légendes sur le circuit imprimé (Figure 3.8). Les noms des broches ont été choisis d'après la fonction la plus souvent utilisée, c'est-à-dire les entrées-sorties numériques, les entrées analogiques et les broches capables de fournir un signal PWM.

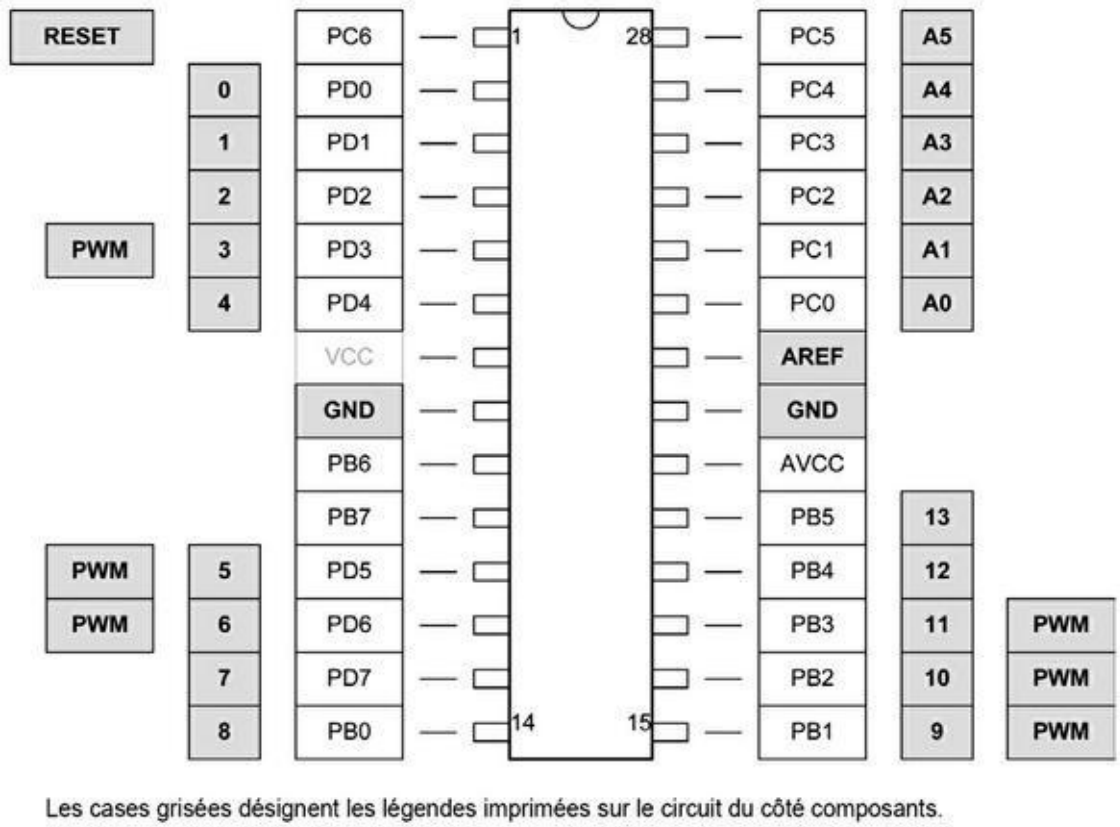


Figure 3.8 : Légendes du circuit imprimé des cartes Arduino équipées de l'ATmega168/328.

Caractéristiques électriques

Le Tableau 3.3 rappelle les principales caractéristiques d'alimentation électrique des deux microcontrôleurs ATmega168/328, avec notamment les consommations en courant.

Pour un état logique Bas. Avec une tension d'alimentation située entre 1,8 V et 2,4 V, l'état logique d'une broche d'entrée est considéré comme au niveau Bas si elle présente une tension entre -0,5 V et deux dixièmes de la tension d'alimentation nominale VCC. Pour une tension d'alimentation entre 2,4 V et 5,5 V, le niveau Bas est considéré entre -0,5 V et trois dixièmes de VCC.

Pour un état logique Haut. Avec une tension d'alimentation située entre 1,8 V et 2,4 V, l'état logique d'une broche d'entrée est considéré comme au niveau Haut si elle présente une tension entre sept dixièmes de la tension d'alimentation nominale VCC et cette tension +0,5 V. Pour une tension d'alimentation entre 2,4 V et 5,5 V, le niveau Haut est considéré entre six dixièmes de VCC et cette tension +0,5 V.

Tableau 3.3 : Consommation des microcontrôleurs ATmega168/328.

Contrôleur	Mode	Condition	VCC	Typique	Max
ATmega168	Nominal	Active 8 MHz	5V	4,2 mA	12 mA
		Idle 8 MHz	5V	0,9 mA	5,5 mA
	Économie (power-save)	32 MHz TOSC	1,V	0,75 µA	
		32 MHz TOSC	5V	0,83 µA	
	Veille (power-down)	WDT actif	3V	4,1 µA	15 µA
		WDT inactif	3V	0,1 µA	2µA
ATmega328	Power supply current	Active 8 MHz	5V	5,2 mA	12 mA
		Idle 8 MHz	5V	1,2 mA	

				5,5 mA
Économie (power-save)	32 MHz TOSC	1,8 V	0,8 µA	
	32 MHz TOSC	5V	0,9 µA	
Veille (power-down)	WDT actif	3V	4,2 µA	15 µA
	WDT inactif	3V	0,1 µA	2 µA

Notes : WDT = Watchdog timer, Idle = Veille

ATmega1280/2560

Comme pour les deux contrôleurs ATmega que nous venons de décrire, ce qui distingue l'ATmega1280 de son successeur ATmega2560 est la quantité de mémoire disponible. Le schéma fonctionnel simplifié des deux contrôleurs est fourni par la Figure 3.9.

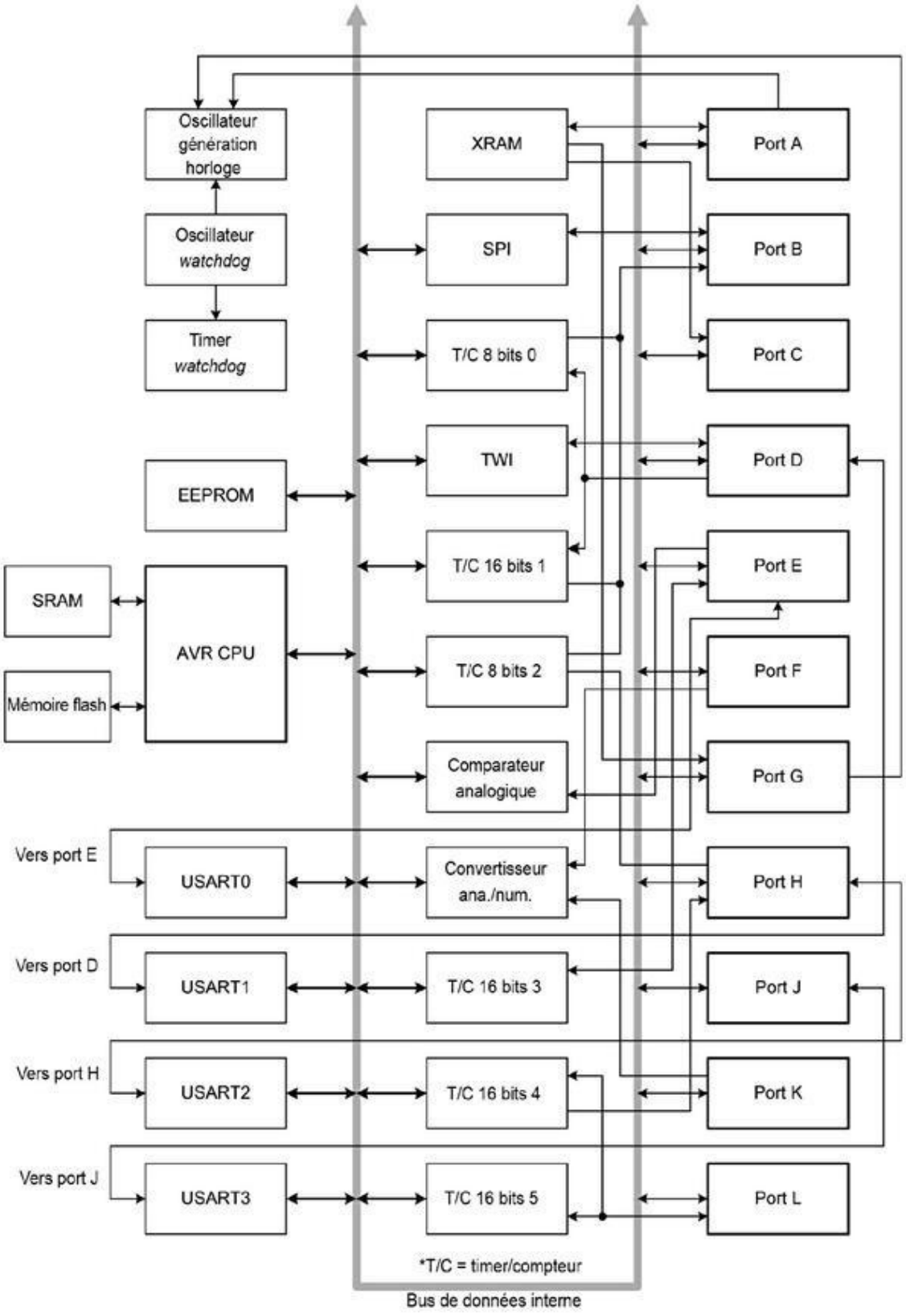


Figure 3.9 : Schéma fonctionnel des microcontrôleurs ATmega1280/2560.

Sachez que certains détails mineurs ne sont pas représentés dans cette figure. Pour un schéma plus complet, vous vous reporterez à la documentation officielle Atmel de ces contrôleurs. La figure montre les fonctions internes disponibles dans le boîtier à 100 broches. Le boîtier à 64 broches ne permet pas d'accéder à certaines des fonctions illustrées.

Espaces mémoire

La variante ATmega2560 offre deux fois plus de mémoire flash que la 1280, soit huit fois plus qu'une ATmega328. Le Tableau 3.4 indique les valeurs des trois espaces mémoire concernés.

Tableau 3.4 : Espaces mémoire des ATmega1280/2560.

	ATmega1280	ATmega2560
Mémoire flash	128 ko	256 ko
Mémoire EEPROM	4 ko	4 ko
Mémoire SRAM	8 ko	8 ko

Fonctions principales

Les deux contrôleurs partagent les fonctions suivantes :

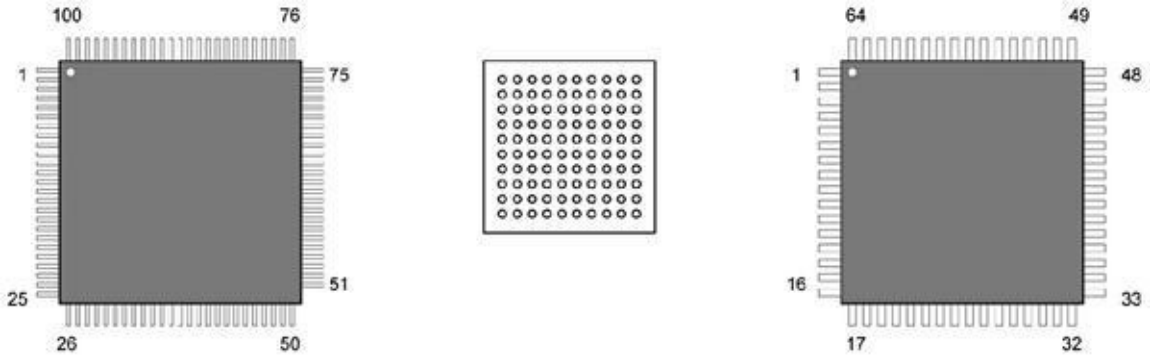
- implantation du programme par utilitaire d'amorçage sur le circuit (In-System Programming) ;
- deux timers/compteurs sur 8 bits avec démultiplicateurs indépendants et mode comparaison ;
- quatre timers/compteurs sur 16 bits avec démultiplicateurs indépendants, mode comparaison et mode capture ;
- un compteur en temps réel avec oscillateur individuel ;

- douze canaux de sortie à modulation de largeur d’impulsion PWM ;
- Un modulateur de comparaison de sortie ;
- six canaux de conversion analogique/numérique sur 10 bits ;
- quatre fonctions de communication série USART ;
- interface série maître-esclave SPI ;
- interface série sur deux fils compatible Philips I2C ;
- un timer chien de garde programmable (*watchdog*) ;
- un comparateur analogique ;
- 86 lignes d’entrées-sorties programmables.

Boîtiers

Les deux contrôleurs sont disponibles dans trois formats physiques : boîtier TQFP à 100 broches, boîtier à billes BGA à 100 broches ou boîtier TQFP à 64 broches. La Figure 3.10 montre les détails et espacements entre broches des trois boîtiers. Seuls les boîtiers à 100 broches donnent accès à toutes les fonctions montrées dans la Figure 3.9.

Les cartes Arduino Mega n’utilisent que la version TQFP à 100 broches.



TQFP 100 broches
14 x 14 mm

BGA 100 broches
9 mm x 9 mm

TQFP 64 broches
14 x 14 mm

Figure 3.10 : Boîtiers des contrôleurs ATmega1280/2560.

Ports d'entrées-sorties

Les boîtiers à 100 broches des deux contrôleurs donnent accès à 11 ports désignés par les lettres A à L. Notez qu'il n'y a pas de port I, car la lettre I pourrait être confondue avec le chiffre 1.

Les ports A, B, C, D et E sont des ports sur 8 bits bidirectionnels. Le port B offre une robustesse supérieure au niveau des courants supportés. Les ports F et K servent d'entrées pour le convertisseur analogique/numérique, mais peuvent également servir de ports bidirectionnels. Le port G est sur 6 bits, alors que les ports H, J et L sont sur 8 bits et bidirectionnels.

Chaque port est doté de broches bidirectionnelles numériques associées à des résistances de tirage (pull-up) internes débrayables. L'état d'activation de ces résistances est géré au moyen de bits dans les registres de contrôle.

Les tampons de sortie des ports peuvent fonctionner en sources comme en puits de courant. Lorsqu'elles sont utilisées en entrées, les broches sont forcées à l'état Bas si les résistances de tirage internes sont actives ou dans un état à haute impédance lorsque la condition de réinitialisation est activée.

Notez que les schémas de cette section concernent le boîtier à 100 broches. Pour les versions BGA et à 64 broches, vous vous reporterez à la documentation officielle Atmel.

Entrées du comparateur analogique

Comme les contrôleurs ATmega168/328, les ATmega1280/2560 proposent deux entrées pour le comparateur analogique, ce que montrent le Tableau 3.5 et la Figure 3.11.

Tableau 3.5 : Entrées du comparateur analogique.

Broche	Port	Fonction
4	PE2	AIN0
5	PE3	AIN1

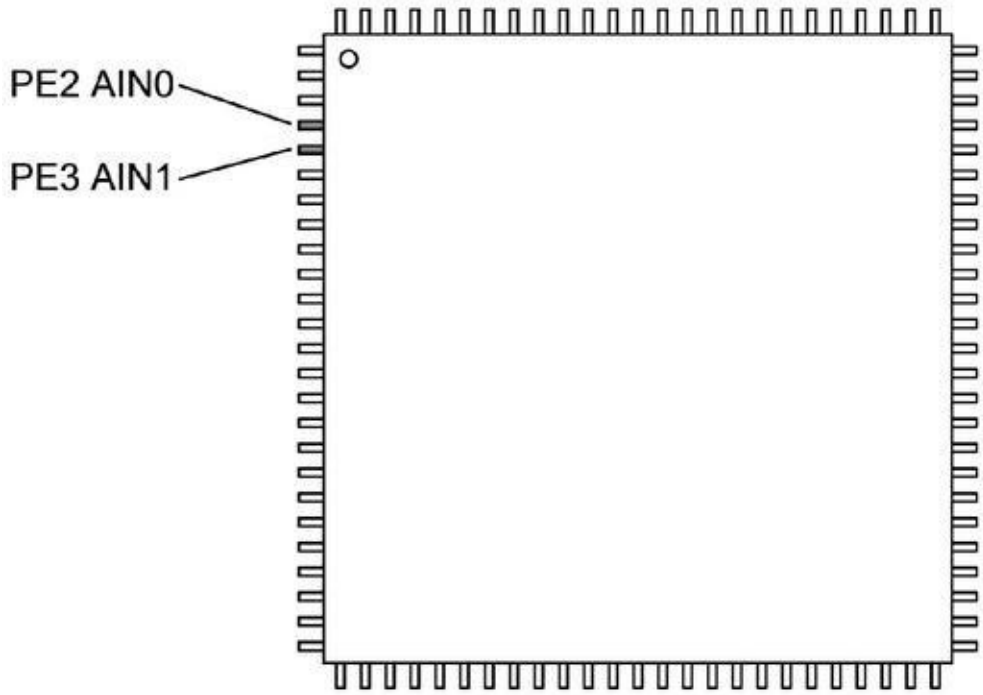


Figure 3.11 : Broches d'entrée du comparateur analogique ATmega1280/2560.

Entrées analogiques

Les ATmega1280 offrent jusqu'à 16 entrées analogiques pour le convertisseur sur les ports F et K, soit les broches 82 à 97 du boîtier à 100 broches (Tableau 3.6). Le port F supporte également les fonctions TCK, TMS, TDI et TDO. Le port K est relié aux interruptions PCINT16 à PCINT23. La position des broches est donnée dans la Figure 3.12.

Tableau 3.6 : Entrées analogiques.

Broche	Port	Fonction	Broche	Port	Fonction
82	PK7	ADC15	90	PF7	ADC7
83	PK6	ADC14	91	PF6	ADC6
84	PK5	ADC13	92	PF5	ADC5
85	PK4	ADC12	93	PF4	ADC4

86	PK3	ADC11	94	PF3	ADC3
87	PK2	ADC10	95	PF2	ADC2
88	PK1	ADC9	96	PF1	ADC1
89	PK0	ADC8	97	PF0	ADC0

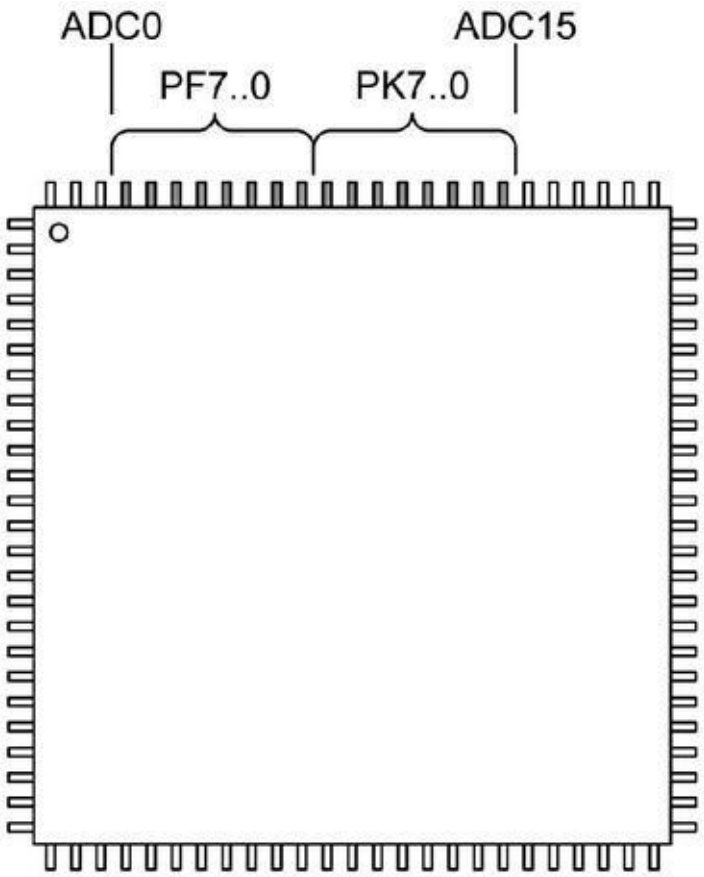


Figure 3.12 : Broches d’entrées analogiques du convertisseur des ATmega1280/2560.

Interfaces série

Les deux microcontrôleurs offrent quatre canaux de communication USART, sous forme de quatre paires de broches, pour l’émission TxD et la réception RxD. L’interface SPI est disponible sur le port B (broches 19 à 22 en boîtier à 100 broches). L’interface sur deux fils I2C est disponible sur le port D (broches 43 et 44). Les affectations sont décrites dans les Tableaux 3.7 à 3.12. La

Figure 3.13 montre l'affectation des broches d'entrées-sorties dans le boîtier à 100 broches.

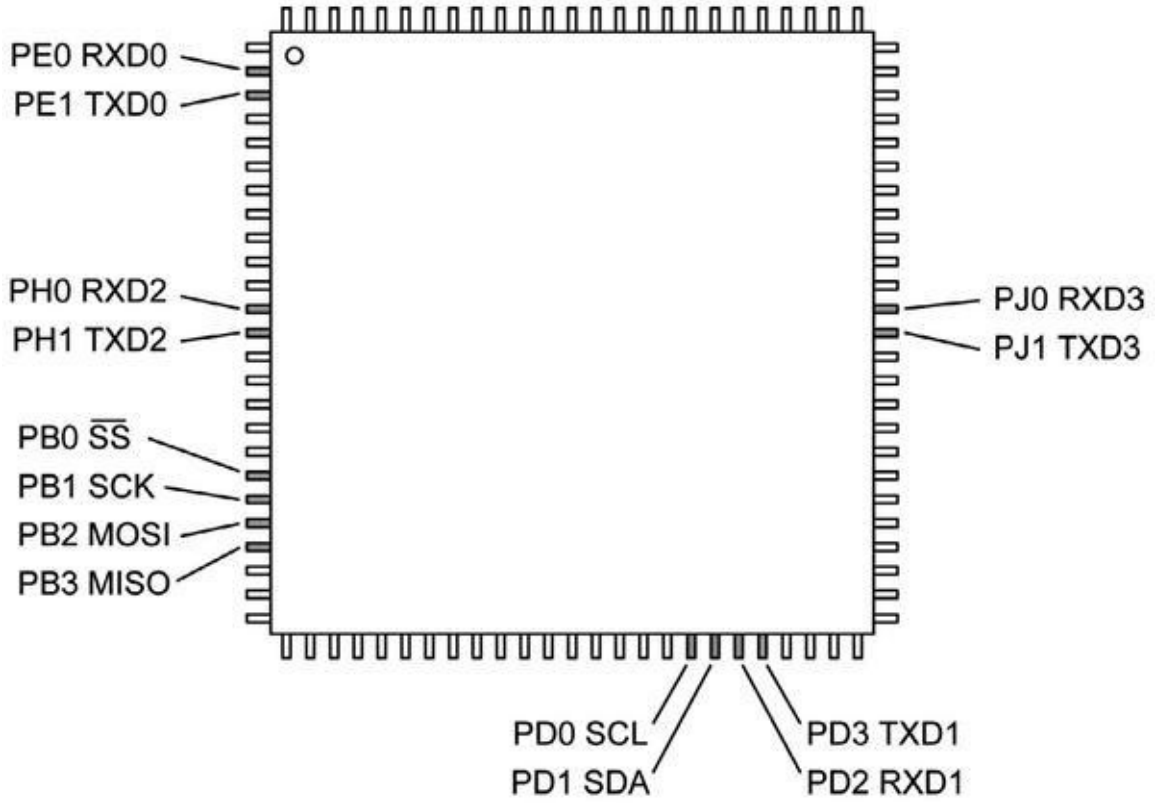


Figure 3.13 : Broches d'entrées-sorties série des ATmega1280/2560.

Tableau 3.7 : USART 0

Broche	Port	Fonction
2	PE0	RXD0
3	PE1	TXD0

Tableau 3.8 : USART 1

Broche	Port	Fonction
45	PD2	RXD1
46	PD3	TXD1

Tableau 3.9 : USART 2

Broche	Port	Fonction
12	PH0	RXD2
13	PH1	TXD2

Tableau 3.10 : USART 4

Broche	Port	Fonction
63	PJ0	RXD3
64	PJ1	TXD3

Tableau 3.11 : SPI

Broche	Port	Fonction
19	PB0	SS (active à l'état Bas)
20	PB1	SCK
21	PB2	MOSI
22	PB3	MISO

Tableau 3.12 : TWI

Broche	Port	Fonction
43	PD0	SCL
44	PD1	SDA

Entrées-sorties des timers/horloges

Les contrôleurs ATmega1280/2560 offrent cinq fonctions de timer/compteur (Figure 3.14). Les broches sont présentées dans le Tableau 3.13. Notez qu'il n'existe pas de broche T2.

Tableau 3.13 : Broches des timers/compteurs ATmega1280/2560.

Broche	Port	Fonction	Broche	Port	Fonction
1	PG5	OC0B	50	PD7	T0
5	PE3	OC3A	15	PH3	OC4A
6	PE4	OC3B	16	PH4	OC4B
7	PE5	OC3C	17	PH5	OC4C
8	PE6	T3	18	PH6	OC2B
9	PE7	ICP3	27	PH7	T4
23	PB4	OC2A	35	PL0	ICP4
24	PB5	OC1A	36	PL1	ICP5
25	PB6	OC1B	37	PL2	T5
26	PB7	OC0A/OC1C	38	PL3	OC5A
47	PD4	ICP1	39	PL4	OC5B
49	PD6	T1	40	PL5	OC5C

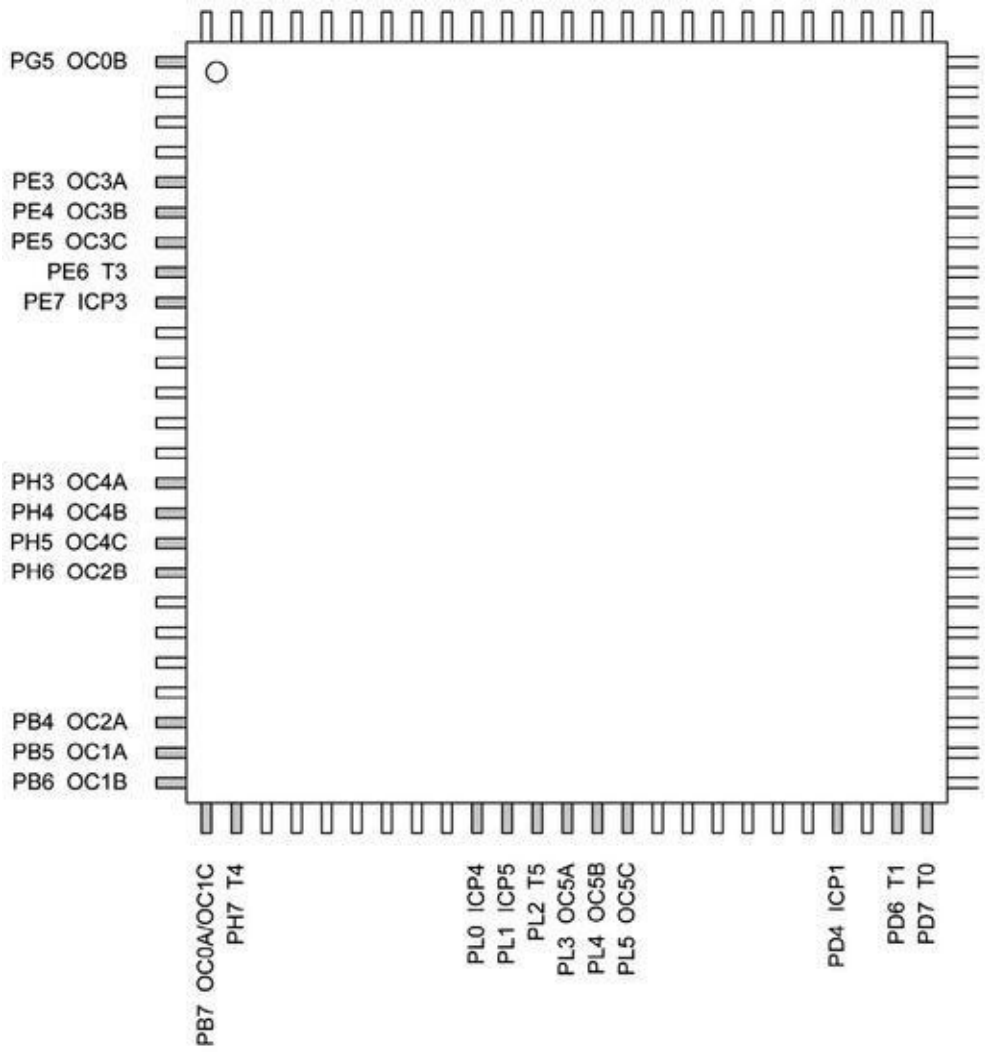


Figure 3.14 : Broches des timers/compteurs.

Interruptions externes

Les contrôleurs ATmega1280/2560 proposent huit entrées pour des interruptions externes en complément des fonctions d'interruption disponibles sur les ports B, J et K. Ces broches sont montrées dans le Tableau 3.14.

Tableau 3.14 : Broches des interruptions externes.

Broche	Port	Fonction	Broche	Port	Fonction
6	PE4	INT4	43	PD0	INT0

7	PE5	INT5	44	PD1	INT1
8	PE6	INT6	45	PD2	INT2
9	PE7	INT7	46	PD3	INT3

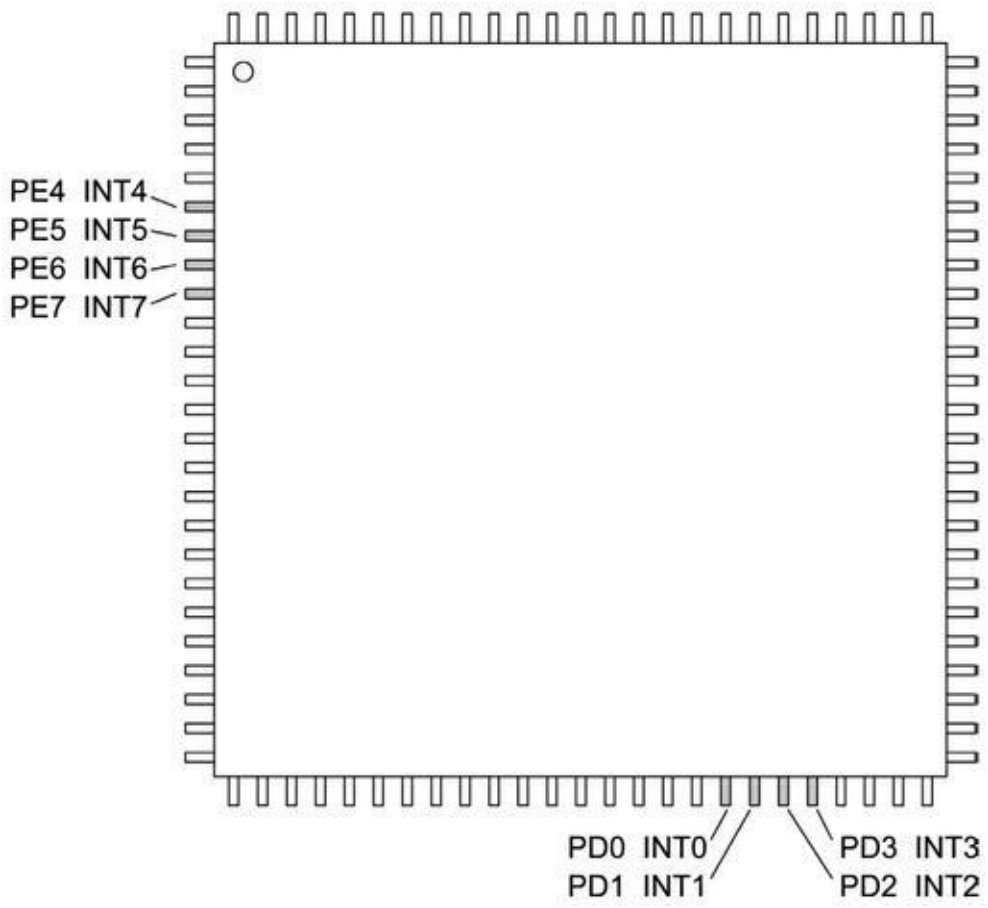


Figure 3.15 : Broche des entrées d'interruptions externes.

Repérage des connecteurs sur le circuit imprimé

Les deux cartes Arduino Mega et Mega2560 comportent des légendes sur la face composants du circuit imprimé qui correspondent aux fonctions les plus utilisées, c'est-à-dire les entrées-sorties numériques, les entrées analogiques et les sorties PWM.

Le Tableau 3.15 établit la correspondance entre les numéros des broches de connexion de la carte et les numéros des broches de sortie du boîtier du microcontrôleur. Les fonctions indiquées entre parenthèses correspondent à des broches d'adressage de mémoire externe. Le symbole ~ précise que le signal est actif à l'état Bas. À la différence des cartes de plus petit format, les cartes au format Mega sont capables d'exploiter de l'espace mémoire externe pour les données.

Je précise que les broches 22 à 37 ainsi que les broches 39, 40 et 41 peuvent servir pour accéder à de la mémoire externe, en plus de leur rôle normal de broches d'entrées-sorties numériques.

Tableau 3.15 : Correspondance des connecteurs Arduino Mega et Mega2560 (analog in = entrée analogique).

Connecteur carte	Broche contrôleur	Fonctions	Connecteur carte	Broche contrôleur	Fonct
0	2	RXD0	35	55	I/O (A
1	3	TXD0	36	54	I/O (A
2	5	OC3B [PWM]	37	53	I/O (A
3	6	OC3C [PWM]	38	50	T0
4	1	OC0B [PWM]	39	70	I/O (A
5	4	OC3A [PWM]	40	52	I/O (~
6	15	OC4A [PWM]	41	51	I/O (~
7	16		42	42	PL7

		OC4B [PWM]			
8	17	OC4C [PWM]	43	41	PL6
9	18	OC2B [PWM]	44	40	OC5C [PWM
10	23	OC2A [PWM]	45	39	OC5B [PWM
11	24	OC1A [PWM]	46	38	OC5A [PWM
12	25	OC1B [PWM]	47	37	T5
13	26	OC0A [PWM]	48	36	ICP5
14	64	TXD3	49	35	ICP4
15	63	RXD3	50	22	MISO
16	13	TXD2	51	21	MOSI
17	12	RXD2	52	20	SCK
18	46	TXD1	53	19	~SS
19	45	RXD1	54	97	A0 (analo
20	44	SDA	55	96	A1 (analo
21	43	SCL	56	95	

					A2 (analo
22	78	I/O (AD0)	57	94	A3 (analo
23	77	I/O (AD1)	58	93	A4 (analo
24	76	I/O (AD2)	59	92	A5 (analo
25	75	I/O (AD3)	60	91	A6 (analo
26	74	I/O (AD4)	61	90	A7 (analo
27	73	I/O (AD5)	62	89	A8 (analo
28	72	I/O (AD6)	63	88	A9 (analo
29	71	I/O (AD7)	64	87	A10 (analo
30	60	I/O (A15)	65	86	A11 (analo
31	59	I/O (A14)	66	85	A12 (analo
32	58	I/O (A13)	67	84	A13 (analo
33	57	I/O (A12)	68	83	

					A14 (analo
34	56	I/O (A11)	69	82	A15 (analo

Caractéristiques électriques

Le Tableau 3.16 rappelle les principales caractéristiques des microcontrôleurs ATmega1280/2560, notamment au niveau de leur consommation électrique.

Tableau 3.16 : Caractéristiques électriques des ATmega1280/2560.

Contrôleur	Mode	Condition	VCC	Typique	Max
ATmega1280	Nominal	Active 8 MHz	5V	10 mA	14 mA
ATmega2560	Nominal	Idle 8 MHz	5V	2,7 mA	4 mA
Les deux	Veille (power-down)	WDT inactif	3V	<5 µA	15 µA
		WDT inactif	3V	<1 µA	7,5 µA

Notes : WDT = watchdog timer, idle = veille

Pour un état logique Bas. Avec une tension d'alimentation située entre 1,8 V et 2,4 V, l'état logique d'une broche d'entrée est considéré comme au niveau Bas si elle présente une tension entre -0,5 V et deux dixièmes de la tension d'alimentation nominale VCC. Pour une tension d'alimentation entre 2,4 V et 5,5 V, le niveau Bas est considéré entre -0,5 V et trois dixièmes de VCC.

Pour un état logique Haut. Avec une tension d'alimentation située entre 1,8 V et 2,4 V, l'état logique d'une broche d'entrée est considéré comme au niveau Haut si elle présente une tension entre sept dixièmes de la tension d'alimentation

nominale VCC et cette tension +0,5 V. Pour une tension d’alimentation entre 2,4 V et 5,5 V, le niveau Haut est considéré entre six dixièmes de VCC et cette tension +0,5 V.

ATmega32U4

Le contrôleur ATmega32U4 fait partie de la gamme XMEGA d'Atmel. Il possède 32 ko de mémoire flash, 2,5 ko de mémoire vive SRAM et 1 ko de mémoire EEPROM. Les fonctions d'entrées-sorties sont disponibles sur les ports B à F (il n'y a pas de port A sur ce modèle). La Figure 3.16 montre le schéma fonctionnel de l'ATmega32U4.

Le contrôleur ATmega32U4 contient un circuit de gestion complète d'interface USB 2.0 (*full-speed*), ce qui libère du besoin d'ajouter un circuit à cet effet sur la carte. Il incorpore aussi un circuit de débogage par interface JTAG conforme à la spécification 1149.1. La tension d'alimentation acceptable va de 2,7 à 5,5 volts.

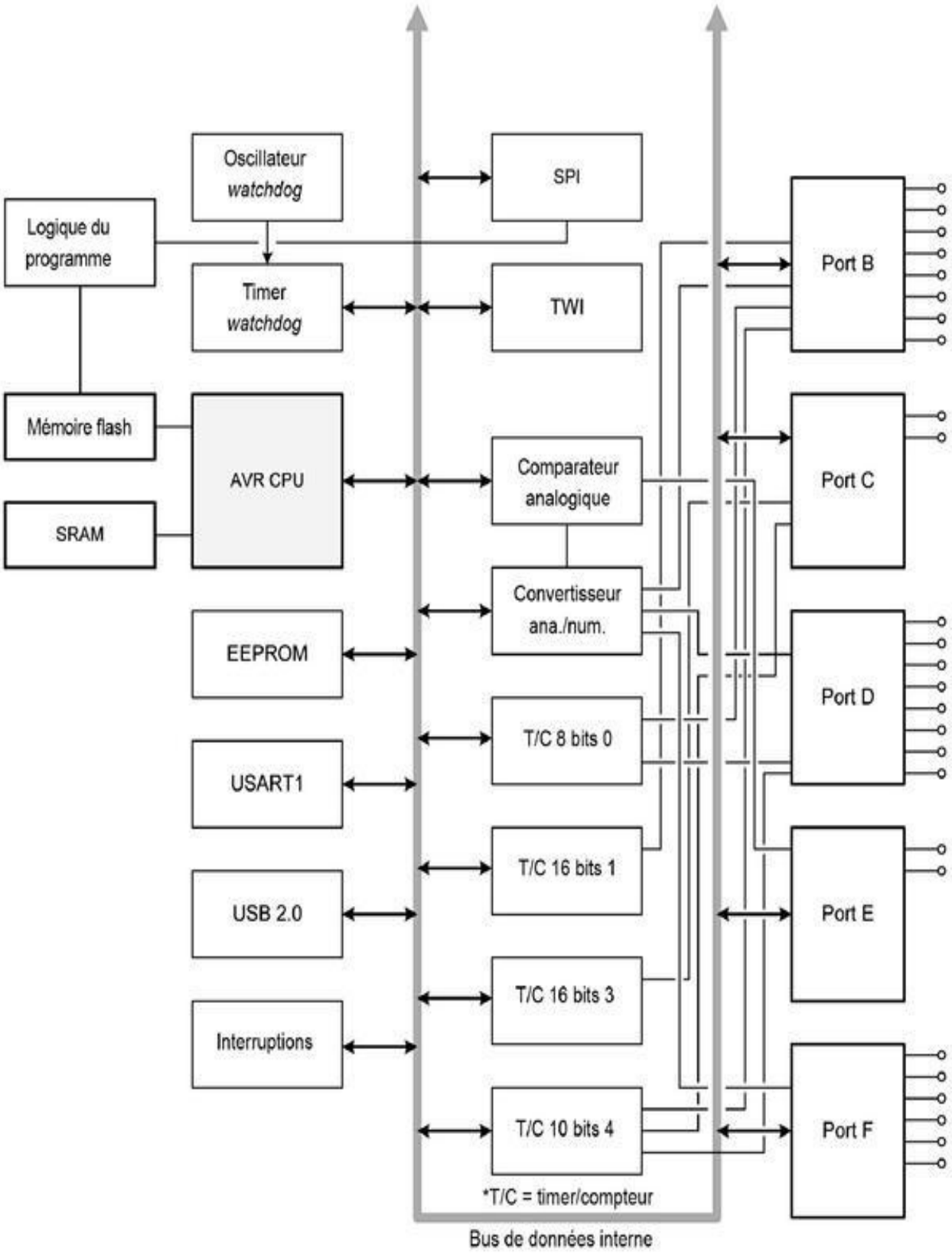


Figure 3.16 : Schéma fonctionnel du microcontrôleur ATmega32U4.

Espace mémoire

Comme le montre le Tableau 3.17, l'ATmega32U4 dispose de la même capacité de mémoire flash et EEPROM que l'ATmega328. Il possède un petit peu plus de mémoire vive SRAM (2,5 ko au lieu de 2 ko).

Tableau 3.17 : Mémoire embarquée de l'ATmega32U4.

ATmega32U4	
Mémoire flash	32 ko
Mémoire EEPROM	1 ko
Mémoire RAM	2,5 ko

Fonctions principales

Voici les principales fonctions de l'ATmega32U4 :

- implantation du programme par utilitaire d'amorçage sur le circuit (In-System Programming)
- un timer/compteur sur 8 bits avec démultiplicateur indépendant et mode comparaison ;
- deux timers/compteurs sur 16 bits avec démultiplicateurs indépendants, mode comparaison et mode capture ;
- un timer/compteur haute vitesse sur 10 bits avec boucle de verrouillage PLL (*phase-locked loop*) et mode compare ;
- quatre canaux PWM 8 bits ;
- quatre canaux PWM à résolution réglable entre 2 et 16 bits ;
- six canaux PWM à haute vitesse à résolution réglable entre 2 et 11 bits ;

- un modulateur de comparaison de sorties ;
- un convertisseur analogique/numérique 12 canaux sur 10 bits ;
- communications série USART avec confirmation d'établissement handshake CTS/RTS ;
- une interface série maître-esclave SPI ;
- deux interfaces série sur deux fils compatibles Philips I2C ;
- un timer chien de garde programmable (*watchdog*) ;
- un comparateur analogique ;
- un capteur de température interne ;
- 26 lignes d'entrées-sorties programmables.

Boîtiers

L'ATmega32U4 est proposé en boîtier TQFP44 ou QFN44, les deux à soudage en surface (SMT). Le brochage (Figure 3.17) est le même pour les deux boîtiers.

Ports d'entrées-sorties

L'ATmega32U4 offre cinq ports légendés B à F. Les boîtiers à 44 broches ne donnent accès aux 8 bits que pour les ports B et D. Les ports C, E et F sont en interne des registres sur 8 bits, mais seuls les bits PC6 et PC7 du port C sont accessibles de l'extérieur. Pour le port E, seuls les bits PE2 et PE6 sont accessibles. Pour le port F au contraire, les bits PF2 et PF3 ne sont pas accessibles sur les broches du boîtier.

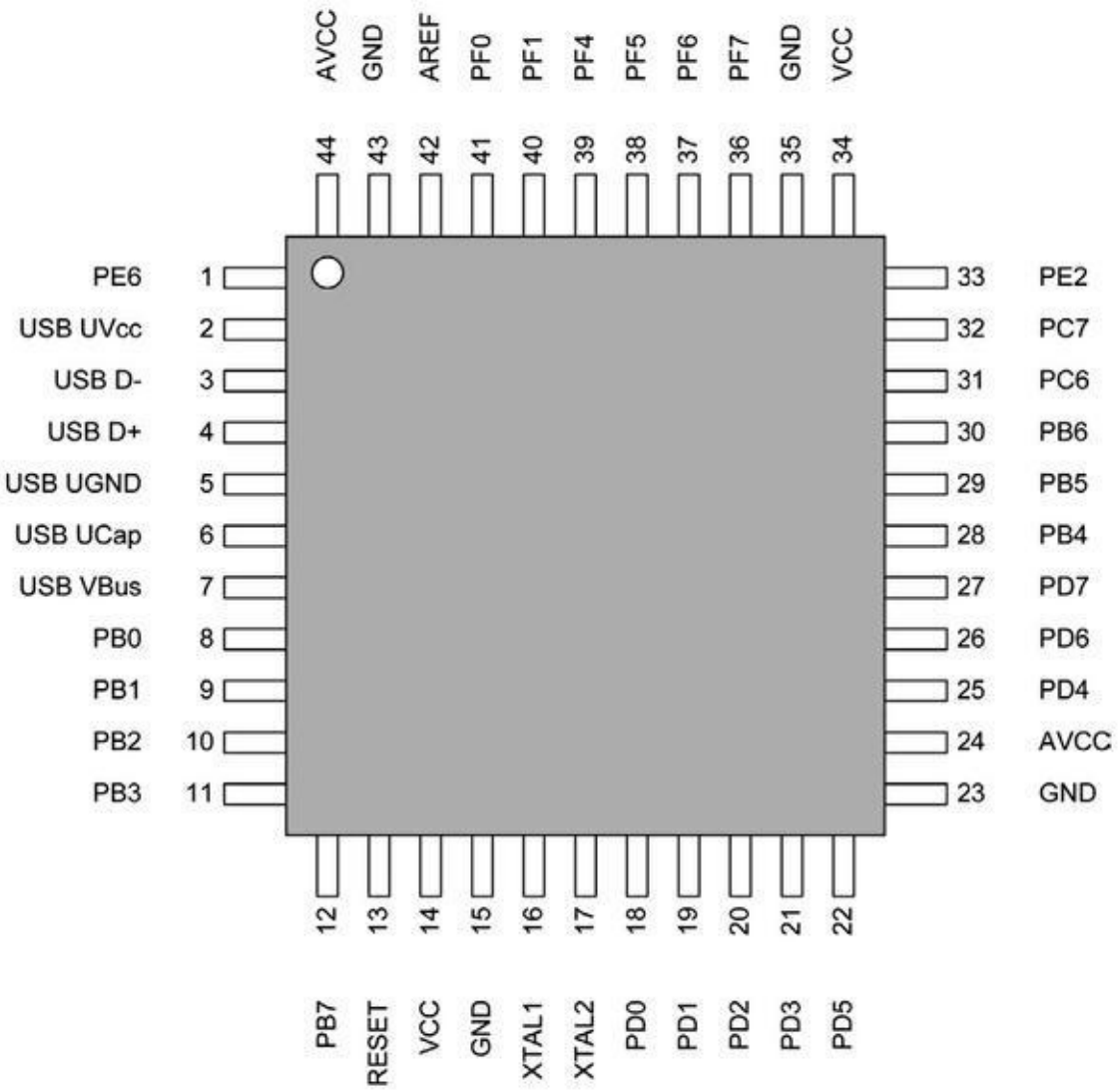


Figure 3.17. ATmega32U4 microcontroller 44-pin package.

Chacun des ports B, C, D, E et F est doté de broches bidirectionnelles numériques associées à des résistances de tirage (*pull-up*) internes débrayables. L'état d'activation de ces résistances est géré au moyen de bits dans les registres de contrôle.

Les tampons de sortie des ports peuvent fonctionner en sources comme en puits de courant (sink). Lorsqu'elles sont utilisées en entrées, les broches sont forcées à l'état Bas si les résistances de tirage internes sont actives ou dans un état à haute impédance (tri-state) lorsque la condition de réinitialisation est activée.

Affectation des broches

Le Tableau 3.18 fournit les affectations des broches de l'ATmega32U4 en boîtier 44 broches. La correspondance entre broches de la puce et contacts de la carte est donnée pour la carte Leonardo dans le Tableau 3.28 plus loin.

Tableau 3.18 : Affectation des broches de l'ATmega32U4 en boîtier TQFP/QFN.

Broche	Port	Fonction	Broche	Port	Fonction
1	PE6	INT6/AIN0	16	n/a	XTAL2
2	n/a	USBUVCC	17	n/a	XTAL1
3	n/a	USBD–	18	PD0	OC0B/SCL/INT(
4	n/a	USBD+	19	PD1	SDA/INT1
5	n/a	USBUGnd	20	PD2	RXD1/INT2
6	n/a	USBUCap	21	PD3	TXD1/INT3
7	n/a	USBVBus	22	PD5	XCK1/CTS
8	PB0	PCINT0/SS	23	n/a	GND
9	PB1	PCINT1/SCLK	24	n/a	AVCC
10	PB2	PDI/PCINT2/MOSI	25	PD4	ICP1/ADC8
11	PB3	PDO/PCINT3/MISO	26	PD6	T1/OC4D/ADC9
12	PB7	PCINT7/OC0A/OC1C/RTS	27	PD7	T0/OC4D/ADC1
13	n/a	RESET	28	PB4	PCINT4/ADC11
14	n/a	VCC	29	PB5	PCINT5/OC1A/(

Broches d'entrée du comparateur analogique

L'ATmega32U4 ne propose qu'une entrée au comparateur analogique : AIN0 sur la broche 1. L'autre entrée provient du multiplexeur d'entrées relié au

convertisseur ana./num. (CAN). Le circuit interne est le même que celui montré en Figure 2.5, mais sans la broche pour AIN1.

Entrées analogiques

L'ATmega32U4 offre douze entrées au convertisseur ana./num. (Tableau 3.19 et Figure 3.18), distribuées sur les ports B, D et F. Notez qu'il n'y a pas d'accès par broche pour ADC2 et ADC3.

Tableau 3.19 : Broches d'entrée du convertisseur analogique de l'ATmega32U4.

Broche	Port	Fonction	Broche	Port	Fonction	Broche	Port	Fonc
41	PF0	ADC0	37	PF6	ADC6	27	PD7	ADC1
40	PF1	ADC1	36	PF7	ADC7	28	PB4	ADC1
39	PF4	ADC4	25	PD4	ADC8	29	PB5	ADC1
38	PF5	ADC5	26	PD6	ADC9	30	PB6	ADC1

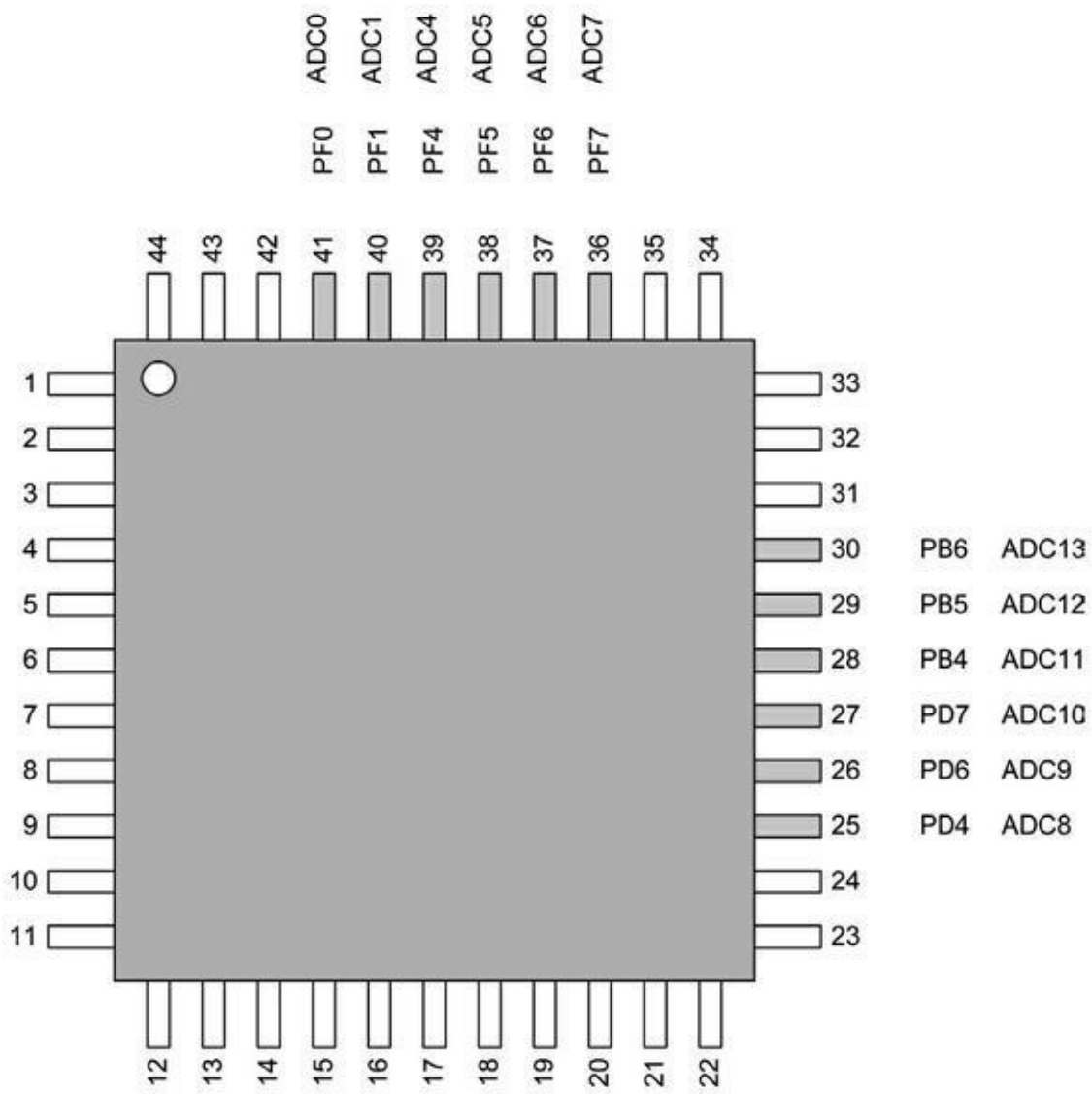


Figure 3.18 : Broches d'entrée du convertisseur analogique de l'ATmega132U4.

Interfaces série

Le contrôleur ATmega32U4 possède une fonction série USART dotée de lignes de contrôle handshake matériel, une interface SPI et une interface TWI compatible I2C (Tableaux 3.20 à 3.22). L'interface USB est décrite isolément plus loin, en Figure 3.22. La Figure 3.19 donne les affectations de broches.

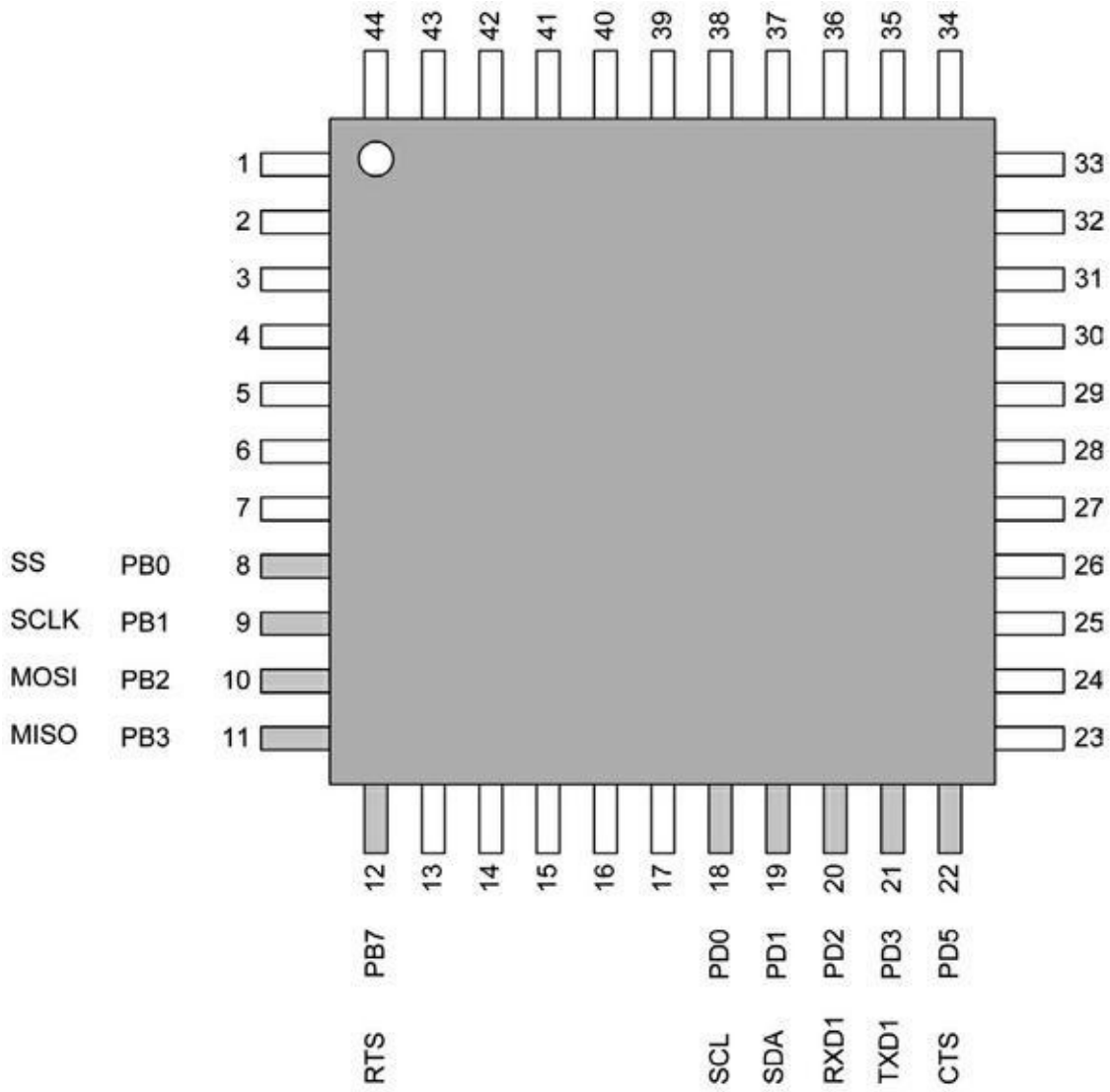


Figure 3.19 : Broches d'entrées-sorties série de l'ATmega32U4.

Tableau 3.20. : Fonctions série USART.

Broche	Port	Fonction	Broche	Port	Fonction
20	PD2	RXD1	22	PD5	CTS
21	PD3	TXD1	12	PB7	RTS

Tableau 3.21 : Interface SPI.

Broche	Port	Fonction	Broche	Port	Fonction
8	PB0	SS	10	PB2	MOSI
9	PB1	SCLK	11	PB3	MISO

Tableau 3.22 : Interface TWI.

Broche	Port	Fonction
18	PD0	SCL
19	PD1	SDA

Entrées-sorties des timers et de l’horloge

L’ATmega32U4 dispose de quatre fonctions de timer/compteur :

- un timer/compteur sur 8 bits ;
- deux timers/compteurs sur 16 bits avec démultiplicateurs indépendants, modes comparaison et capture ;
- un timer/compteur à haute vitesse sur 10 bits avec boucle PLL (phase-locked loop) et mode comparaison.

Les timers sont numérotés de 0 à 4, mais il n’existe pas de timer/compteur 2.

Le Tableau 3.23 liste les broches et ports des fonctions timer et horloge de l’ATmega32U4 MCU. Le symbole ~ précise que le signal est actif à l’état Bas. Notez qu’il n’y a pas de signaux T3 ni T4 dans l’ATmega32U4. Seuls les timers/compteurs 0 et 1 rendent T0 et T1 accessibles (Figure 3.20).

Tableau 3.23 : Broches des timers et d’horloge.

Broche	Port	Fonction	Broche	Port	Fonction
12	PB7	OC0A/OC1C	29	PB5	OC1A/~OC4B

18	PD0	OC0B	30	PB6	OC1B/OC4B
25	PD4	ICP1	31	PC6	OC3A/~OC4A
26	PD6	T1/~OC4D	32	PC7	ICP3/OC4A

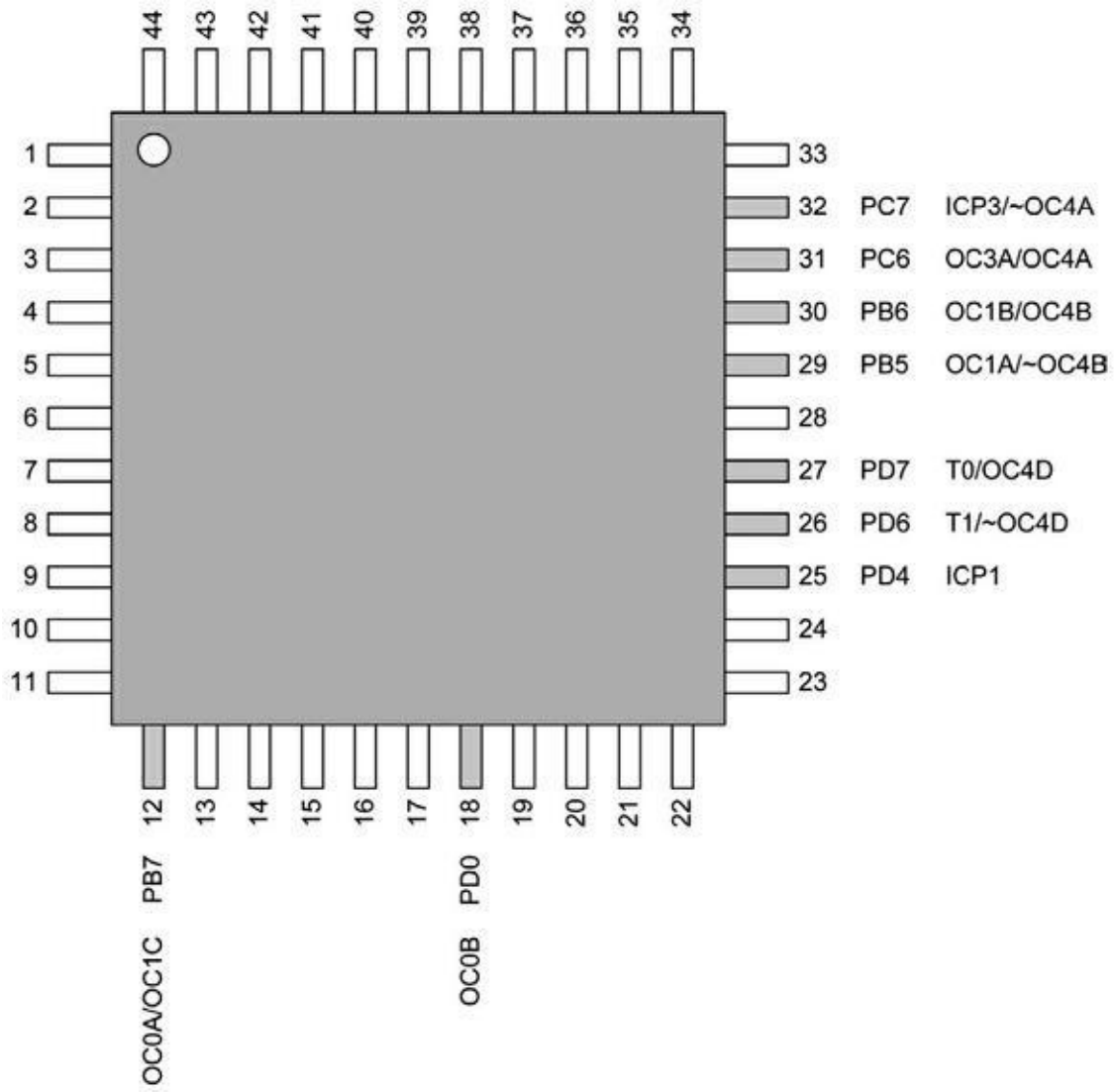


Figure 3.20 : Broches des timers/compteurs de l'ATmega32U4.

Interruptions externes

Les broches servant d'entrées pour les interruptions externes sont listées dans le Tableau 3.24 et la Figure 3.21. Les broches PD0 à PD3 du port D et la broche

PE6 du port E sont dédiées à cette fonction. Les broches de PCINT0 à PCINT7 peuvent servent aux interruptions sans créer de conflit avec leurs autres fonctions (les interruptions ont été décrites dans le Chapitre 2).

Tableau 3.24 : Broches des interruptions externes.

Broche	Port	Fonction	Broche	Port	Fonction
8	PB0	PCINT0	12	PB7	PCINT7
9	PB1	PCINT1	18	PD0	INT0
10	PB2	PCINT2	19	PD1	INT1
11	PB3	PCINT3	20	PD2	INT2
28	PB4	PCINT4	21	PD3	INT3
29	PB5	PCINT5	1	PE6	INT6

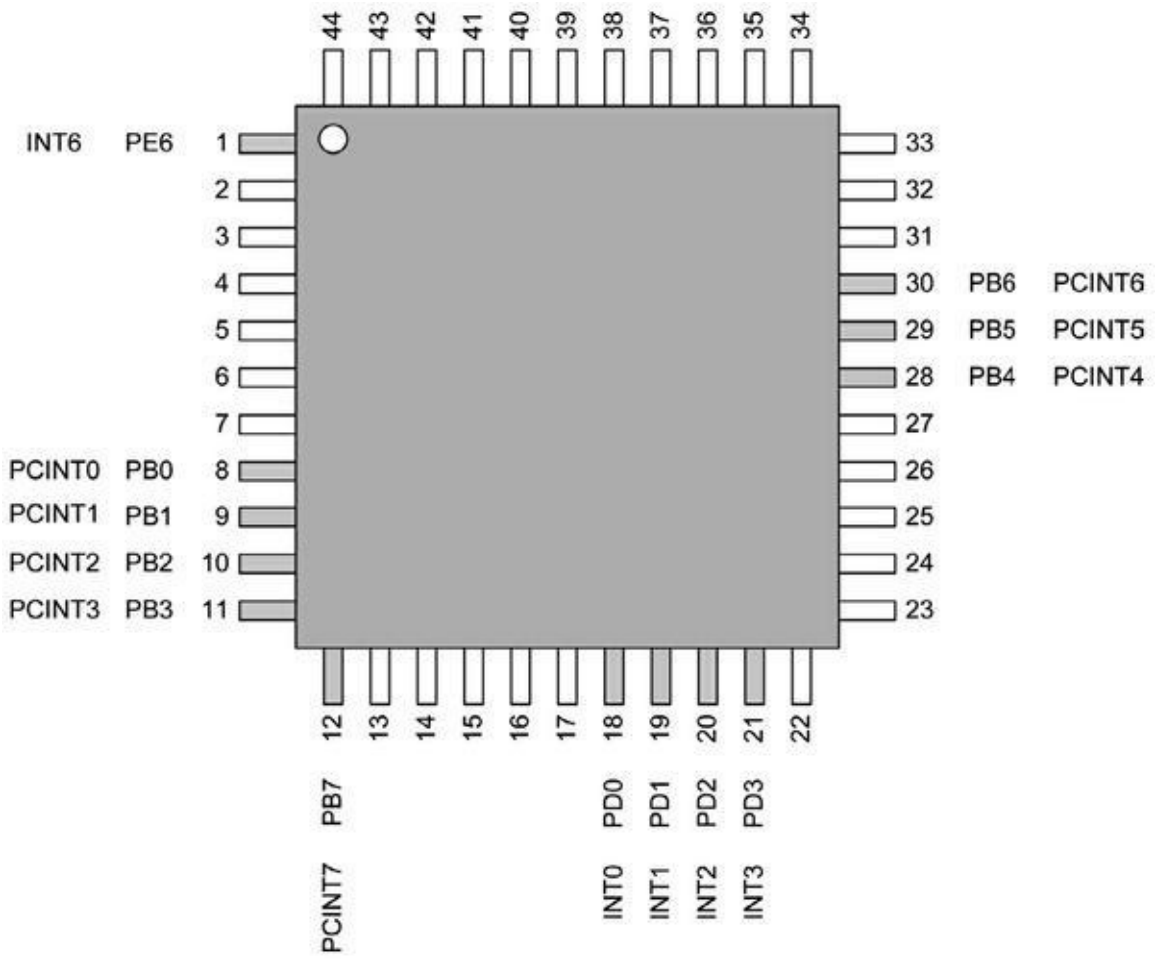


Figure 3.21 : Broches d'entrées d'interruption de l'ATmega32U4.

Interface USB 2.0

Le contrôleur ATmega32U4 est doté d'une interface USB 2.0 mais il ne peut pas être utilisé comme hôte USB, seulement comme périphérique. Il offre plusieurs terminaisons internes avec des tampons (*buffer*) FIFO configurables. Un circuit PLL interne génère un signal d'horloge à 48 MHz pour l'interface USB ; l'entrée PLL peut être réglée pour exploiter un oscillateur externe, une horloge externe ou l'horloge interne à circuit RC. La sortie à 48 MHz de la boucle PLL sert à générer une horloge à 12 MHz (*full-speed*) ou à 1,5 MHz.

Notez que les broches USB de l'ATmega32U4 ne sont pas reliées à un port, mais seulement au circuit USB interne et au régulateur de tension. Les broches concernées sont listées dans le Tableau 3.25 et repérées dans la Figure 3.22.

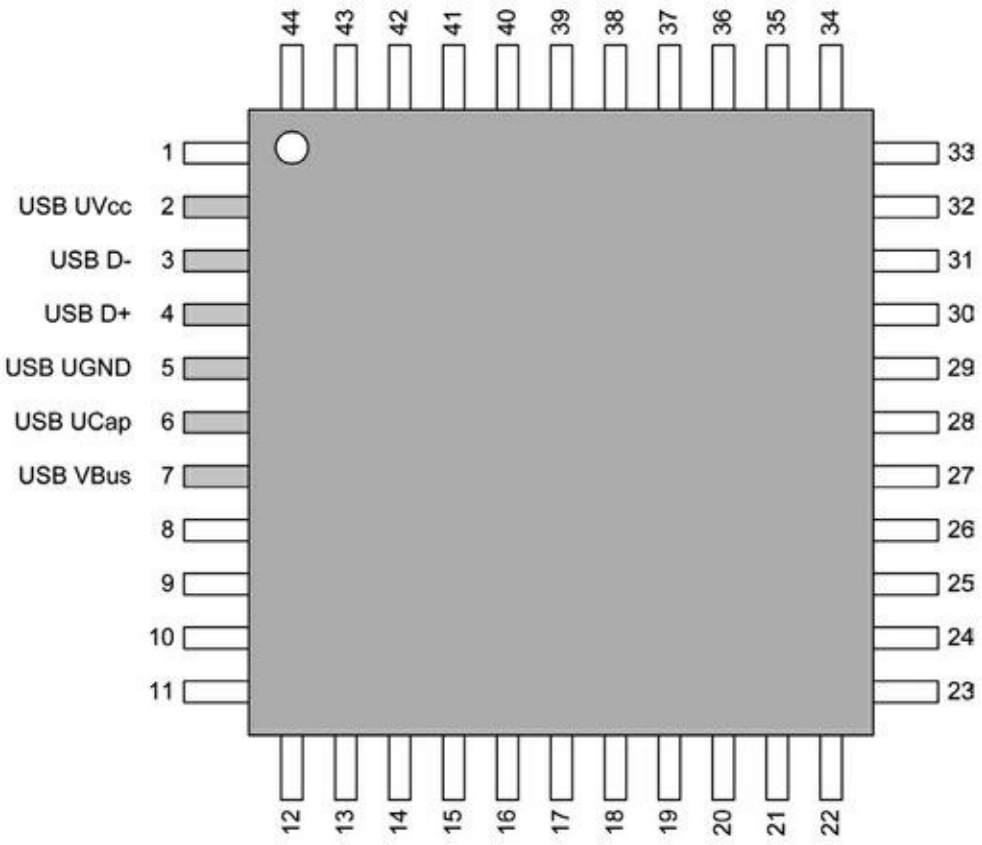


Figure 3.22 : Broches USB de l'ATmega32U4.

Tableau 3.25 : Broches USB.

Broche	Fonction	Broche	Fonction
2	USBUVCC	5	USBUGnd
3	USBD–	6	USBUCap
4	USBD+	7	USBVBus

Caractéristiques électriques

Le Tableau 3.26 rappelle les valeurs d'alimentation acceptées pour le microcontrôleur ATmega32U4, et le Tableau 3.7 liste ses consommations électriques pour les différents modes de fonctionnement.

Tableau 3.26 : Valeurs d'alimentation maximales de l'ATmega32U4.

Paramètre	Valeur	Unité
Tension broches I/O	-0,5 à VCC + 0,5	V
Tension broche Reset	-0,5 à +13,0	V
Tension broche VBUS	-0,5 à +6,0	V
VCC circuit	6,0	V
Courant direct broches I/O	40	mA
Courant VCC circuit	200	mA

Tableau 3.27 : Consommations typiques de l'ATmega32U4.

Mode	Condition	VCC	Typique	Max
Mode nominal	Actif 8 MHz	5V	10 mA	15 mA
	Idle 8 MHz	5V		6 mA
Mode veille (power-down)	WDT actif	3V	<10 µA	12 µA
	WDT inactif	3V	1 µA	5 µA

Repérage des connecteurs sur le circuit imprimé Leonardo

La carte Leonardo se présente au format de base étendu, donc avec les quatre broches supplémentaires. Les broches de l'interface USB sont connectées au connecteur USB du circuit imprimé et non disponibles sur les autres connecteurs. L'aspect de la Leonardo est décrit dans le Chapitre 4.

La correspondance des broches du circuit intégré ATmega32U4 avec les connecteurs de la carte Leonardo est fournie dans le Tableau 3.28. Notez que toutes les broches du circuit ATmega32U4 ne sont pas reportées sur la carte.

Certaines des broches d’entrée du convertisseur CAN sont marquées comme entrées-sorties numériques.

Tableau 3.28 : Correspondances entre connecteurs de la Leonardo et broches de l’ATmega32U4.

Connecteur carte	Broche contrôleur	Fonctions	Connecteur carte	Broche contrôleur
0	20	RXD1/INT2	11	12
1	21	TXD1/INT3	12	26
2	19	SDA/INT1	13	32
3	18	OC0B/SCL/INT0	A0	36
4	25	ICP1/ADC8	A1	37
5	31	OC3A/OC4A	A2	38
6	27	T0/OC4D/ADC10	A3	39
7	1	INT6/AIN0	A4	40
8	28	PCINT4/ADC11	A5	41
9	29	PCINT5/OC1A/ OC4B/ADC12	AREF	42

Bits fusibles

Tous les microcontrôleurs (MCU) de la famille AVR se basent sur un jeu de bits de configuration appelés bits fusibles (*fuse bits*) pour définir par exemple la fréquence d'horloge, les facteurs de démultiplication des timers/compteurs, les verrous d'accès mémoire, *etc*. Ce jeu de bits ressemble à un panneau d'interrupteurs. Nous allons découvrir les détails de ceux des modèles ATmega168/328, parmi les plus répandus (ils équipent les cartes Duemilanove, Mini, Nano et Uno). Le boîtier DIP est le plus répandu. Le principe s'applique aux contrôleurs ATmega1280/2560 et ATmega32U4.

Dans l'ATmega168/328, les bits fusibles sont rassemblés dans trois octets désignés Low, High et Extended. C'est une logique active à l'état Bas (le bit est armé quand il est forcé à 0). Les Tableaux 3.29, 3.30 et 3.31 présentent ces trois octets avec l'effet de chaque bit fusible.

Tableau 3.29 : Bits fusibles de l'octet Low.

Bit	Numéro	Description	ValDef	Bit	Numéro	Descriptio
CKDIV8	7	Horloge / 8	0	CKSEL3	3	Source horloge
CKOUT	6	Sortie horloge	1	CKSEL2	2	Source horloge
SUT1	5	Temps start-up	1	CKSEL1	1	Source horloge
SUT0	4	Temps start-up	0	CKSEL0	0	Source horloge

Tableau 3.30 : Bits fusibles de l'octet High.

Bit	Numéro	Description	ValDef	Bit	Numéro	Des

RSTDISBL	7	Pas de Reset externe	1	EESAVE	3	EEPI prés
DWEN	6	debugWIRE actif	1	BODLEVEL2	2	Nive trigg
SPIEN	5	Download SPI actif	0	BODLEVEL1	1	Nive trigg
WDTON	4	Timer *watchdog* actif	1	BODLEVEL0	0	Nive trigg

Note : BOD signifie Brown-Out Detection, ou détection de sous-tension.

Tableau 3.31 : Bits fusibles de l'octet Extended.

Bit	Numéro	Description	ValDef	Bit	Numéro	Description
-	7	-	1	-	3	-
-	6	-	1	BOOTSZ1	2	Taille Boot
-	5	-	1	BOOTSZ0	1	Taille Boot
-	4	-	1	BOOTRST	0	Vecteur Reset

La logique de routage en entrée et en sortie des signaux d'horloge dans les circuits AVR porte le nom « *Unité de contrôle de l'horloge* » (Figure 3.23).

Les noms des chemins de synchronisation qui commencent par « clk » précisent les fonctions de synchronisation de chaque chemin. Les détails sont dans la fiche technique Atmel de l'ATmega328. Le multiplexeur de source d'horloge peut être contrôlé avec les quatre bits fusibles CKSEL(3:0), ou en modifiant le quartet

(4 bits) de poids faible. Le Tableau 3.32 liste les sources d'horloge possibles avec les valeurs des bits CKSEL correspondantes.

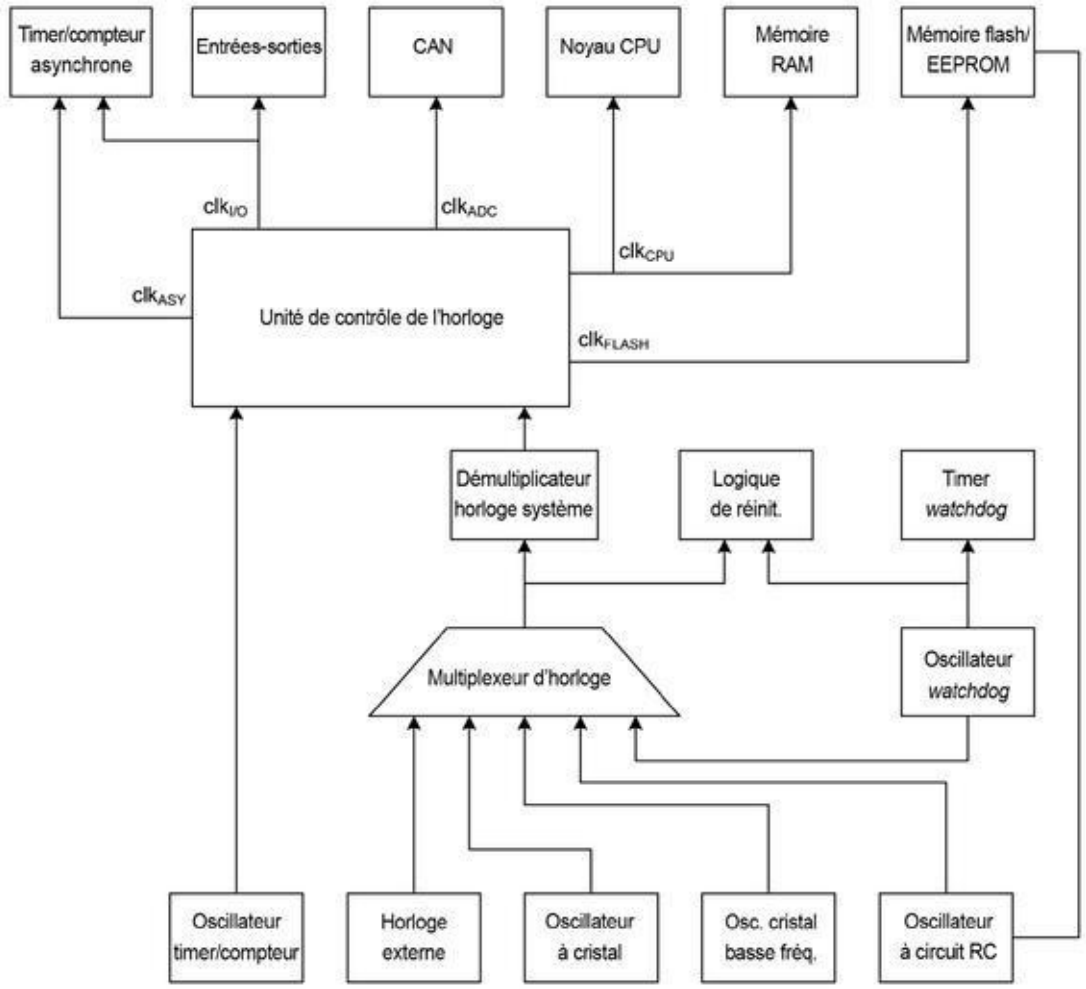


Figure 3.23 : Le sous-système de contrôle et de distribution de synchronisation AVR.

Tableau 3.32 : Sélection de la source d'horloge avec les bits CKSEL.

Source horloge	CKSEL(3:0)	Source horloge	CKSEL(3:0)
Cristal Low power	1111 - 1000	Interne calibré	0010
Cristal fréq. nominale	0111 - 0110	Horloge externe	0000
Cristal fréq. basse	0101 - 0100	Inutilisé (réservé)	0001

Notez que les sources du cristal d'horloge offrent plusieurs sous-options selon les bits qui sont armés. J'utilise ces bits fusibles dans le Chapitre 10 pour configurer un microcontrôleur nu (non encore préparé pour Arduino) afin de l'utiliser dans un projet sur circuit imprimé sur mesure.

Dans un appareil basé sur une carte Arduino, vous ne modifierez généralement pas les bits CKSEL, mais les autres bits fusibles peuvent servir pour accéder à certaines fonctions. Dans le deuxième octet, High, le bit WDTON permet d'activer le timer chien de garde (watchdog). Les trois bits BODLEVEL permettent de régler le niveau d'alerte du mécanisme de détection de sous-tension BOD (*Brown-Out Detection*). Dans les deux cas (fin de décompte du chien de garde ou détection de sous-tension), l'événement qui est déclenché provoque une réinitialisation du contrôleur (*Reset*).

Le chien de garde est en fait très pratique dès que vous craignez qu'une erreur du programme puisse entraîner de sérieux dégâts. Le chien de garde ne va pas aboyer (déclencher son interruption) tant qu'il est relancé avant d'arriver en fin de décompte, opération normalement réalisée dans la boucle principale `main()`. Si le code ne répond plus (par exemple parce qu'une fonction tombe dans une boucle infinie), la boucle principale n'est plus exécutée et le chien de garde parvient à finir son décompte. Cela débloque la situation par un Reset. La librairie essentielle *avr-libc* définit trois fonctions pour dompter ce chien de garde : `wdt_enable()`, `wdt_disable()` et `wdt_reset()`.

La *détection de sous-tension* (*brown-out*) génère une interruption lorsque la tension d'alimentation VCC descend sous une valeur limite que vous choisissez. Vous avez ainsi souvent le temps de sauvegarder les données critiques en mémoire flash et de fermer les ports de sortie pour éviter l'émission d'ordres incohérents.

N. d. T. : En effet, une sous-tension ne va affecter au départ que certains circuits du microcontrôleur qui risquent ainsi (par malchance) de continuer à fonctionner, mais avec des données erronées, ce qui peut avoir de graves conséquences.

Vous modifiez les bits fusibles avec un outil tel que ceux décrits dans la section « Programmation in-situ ISP » du Chapitre 6. Dans la section qui décrit le prototype du projet Switchinator (Chapitre 10), un encadré explique comment armer les bits fusibles pour configurer un contrôleur AVR afin qu'il fonctionne avec une horloge à 16 MHz.

Ce qui précède n’est qu’un aperçu des bits fusibles. La documentation Atmel décrit en détail le chien de garde et les détections de sous-tension.

Pour aller plus loin

Le Chapitre 2 a présenté les fonctions périphériques internes dans contrôleurs AVR. Le Chapitre 4 détaille le brochage des cartes Arduino.

Pour d'autres détails au sujet de ces fonctions, voyez le site officiel Atmel, dans la section **Support**/**Datasheets**. Voici les titres des fiches techniques à consulter :

- *ATmega48PA/ATmega88PA/ATmega168PA/ATmega328P 8-bit microcontroller with 4/8/16/32K Bytes In-System Programmable Flash ;*
- *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V 8-bit Atmel microcontroller with 16/32/64KB In-System Programmable Flash ;*
- *ATmega16U4/ATmega32U4 8- bit microcontroller with 16/32K Bytes of ISP Flash and USB Controller.*

CHAPITRE 4

Détails techniques Arduino

Ce chapitre présente les caractéristiques physiques et électriques d'un grand nombre de cartes Arduino, de l'antique Diecimila aux plus récentes Leonardo, Esplora et Micro. Nous verrons les brochages des boîtiers et les encombrements physiques pour les formats standard : le format de base des premières Uno, le format étendu des Uno R3, le format allongé dit format Mega, sans oublier les modèles miniatures Mini, Micro et Nano ainsi que la spécifique Esplora.

Fonctions et possibilités des cartes Arduino

Le Tableau 4.1 permet de comparer les cartes Arduino les plus répandues. Si vous étudiez ce tableau en même temps que le contenu du Chapitre 1, il devient évident que les possibilités d'une carte Arduino coïncident presque avec celles offertes par son microcontrôleur. Cependant, du fait que l'équipe Arduino a choisi de dédier certaines broches du contrôleur à certaines fonctions ou de ne pas reporter certaines autres broches sur les connecteurs de la carte, toutes les possibilités du microcontrôleur ne sont pas disponibles exactement de la même manière que sur les entrées-sorties du boîtier du microcontrôleur.

Le terme « broche » (pin, en anglais) désigne ici et dans d'autres chapitres les points de connexion de la carte Arduino afin de simplifier les descriptions et d'utiliser le terme que l'on trouve dans la littérature dédiée à Arduino. Pourtant, les véritables broches sont celles du circuit intégré microcontrôleur qui est implanté sur la carte. En pratique, cela n'entraîne pas de confusion.

Tableau 4.1 : Caractéristiques matérielles des cartes Arduino.

Carte	Processeur	VCC	Horloge (MHz)	AIN	DIO	PWM	USB
ArduinoBT	ATmega328	5	16	6	14	6	Aucun
Duemilanove	ATmega168	5	16	6	14	6	Normal
Duemilanove	ATmega328	5	16	6	14	6	Normal
Diecimila	ATmega168	5	16	6	14	6	Normal
Esplora	ATmega32U4	5	16	-	-	-	Micro
Ethernet	ATmega328	5	16	6	14	4	Normal
Fio	ATmega328P	3.3	8	8	14	6	Mini
Leonardo	ATmega32U4	5	16	12	20	7	Micro

LilyPad	ATmega168V	2.7–5.5	8	6	14	6	Aucun
LilyPad	ATmega328V	2.7–5.5	8	6	14	6	Aucun
Mega	ATmega1280	5	16	16	54	15	Normal
Mega ADK	ATmega2560	5	16	16	54	15	Normal
Mega 2560	ATmega2560	5	16	16	54	15	Normal
Micro	ATmega32U4	5	16	12	20	7	Micro
Mini	ATmega328	5	16	8	14	6	Aucun
Mini Pro	ATmega168	3.3	8	6	14	6	Aucun
Mini Pro	ATmega168	5	16	6	14	6	Aucun
Nano	ATmega168	5	16	8	14	6	Mini-B
Nano	ATmega328	5	16	8	14	6	Mini-B
Pro (168)	ATmega168	3.3	8	6	14	6	Aucun
Pro (328)	ATmega328	5	16	6	14	6	Aucun
Uno	ATmega328	5	16	6	14	6	Normal
Yún	ATmega32U4	5	16	12	20	7	Hôte (A)

Colonne AIN : Entrées analogiques.

Colonne DIO : Entrées-sorties numériques (Digital I/O).

*Colonne PWM : Sorties à modulation de largeur d'impulsion (**Pulse-Width Modulation**), sur certaines broches DIO*.

Interfaces USB

Apparu d'abord dans la carte Leonardo en 2012, le contrôleur ATmega32U4 de la série XMEGA embarque une interface USB, ce qui rend inutile le circuit dédié à cette fonction qui est présent sur d'autres cartes Arduino. Sont également équipées de ce contrôleur ATmega32U4 la Esplora (2012), la Micro (2012) et la Yún (2013).

Les modèles plus anciens dotés d'une interface USB ont besoin d'un circuit dédié FTDI (le FT232RL), en complément du contrôleur ATmega8 (Uno ancien) ou ATmega16U2 (Mega2560 et Uno R3). Le circuit FT232RL assure la conversion entre le protocole série standard (RS-232 par exemple) et le protocole USB. Ces opérations sont invisibles au niveau de l'atelier Arduino lorsqu'il réalise les téléversements des programmes.

Les cartes qui ne sont pas dotées d'une interface USB doivent être programmées pour exploiter un adaptateur externe.

Les cartes qui possèdent le circuit FTDI FT232RL utilisent toutes le même schéma fonctionnel composé d'un circuit régulateur de tension et de deux circuits intégrés. La Figure 4.1 propose un schéma des cartes Diecimila et Duemilanove avec le circuit d'interface FTDI.

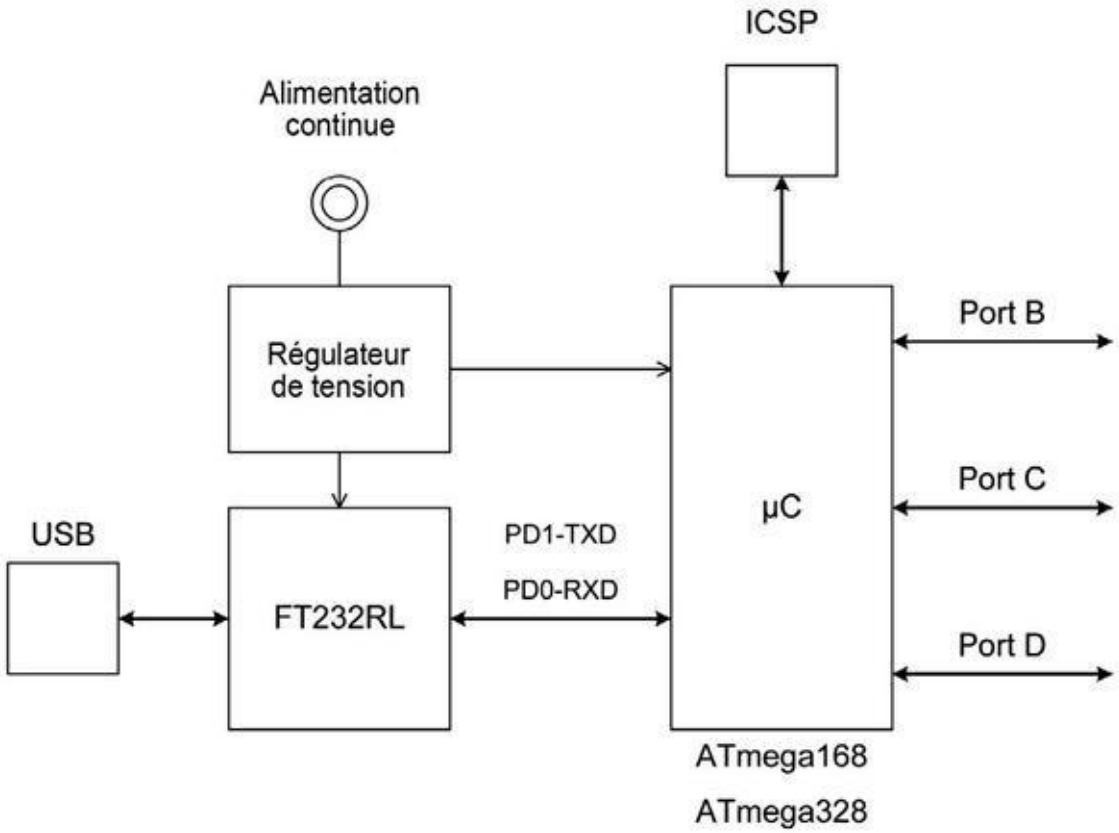


Figure 4.1 : FTDI USB interface

Depuis environ 2010, les Uno R2 et Uno SMD ont préféré l'ATmega16U2 au circuit FTDI FT232RL pour la partie USB. La carte Uno R3 utilise elle aussi le circuit ATmega16U2 pour l'USB. Ce circuit ATmega16U2 gère une interface USB 2.0 et se distingue de l'ATmega32U4 par moins de mémoire. La Figure 4.2 montre le schéma de l'Uno R2 avec interface USB de l'ATmega16U2. L'Uno ne s'en distingue que par le circuit de gestion de l'interface USB.

Encombrement physique des cartes

Les cartes Arduino existent en plusieurs formats physiques : le format de base ou étendu Uno, le format allongé dit format Mega, le format miniature et les formats spéciaux.

Les dimensions que je fournis ne sont pas exactes au millimètre près, car celles-ci varient légèrement d'une source à l'autre. Si vous avez besoin de connaître les dimensions exactes, le mieux est de les mesurer sur la carte que vous possédez.

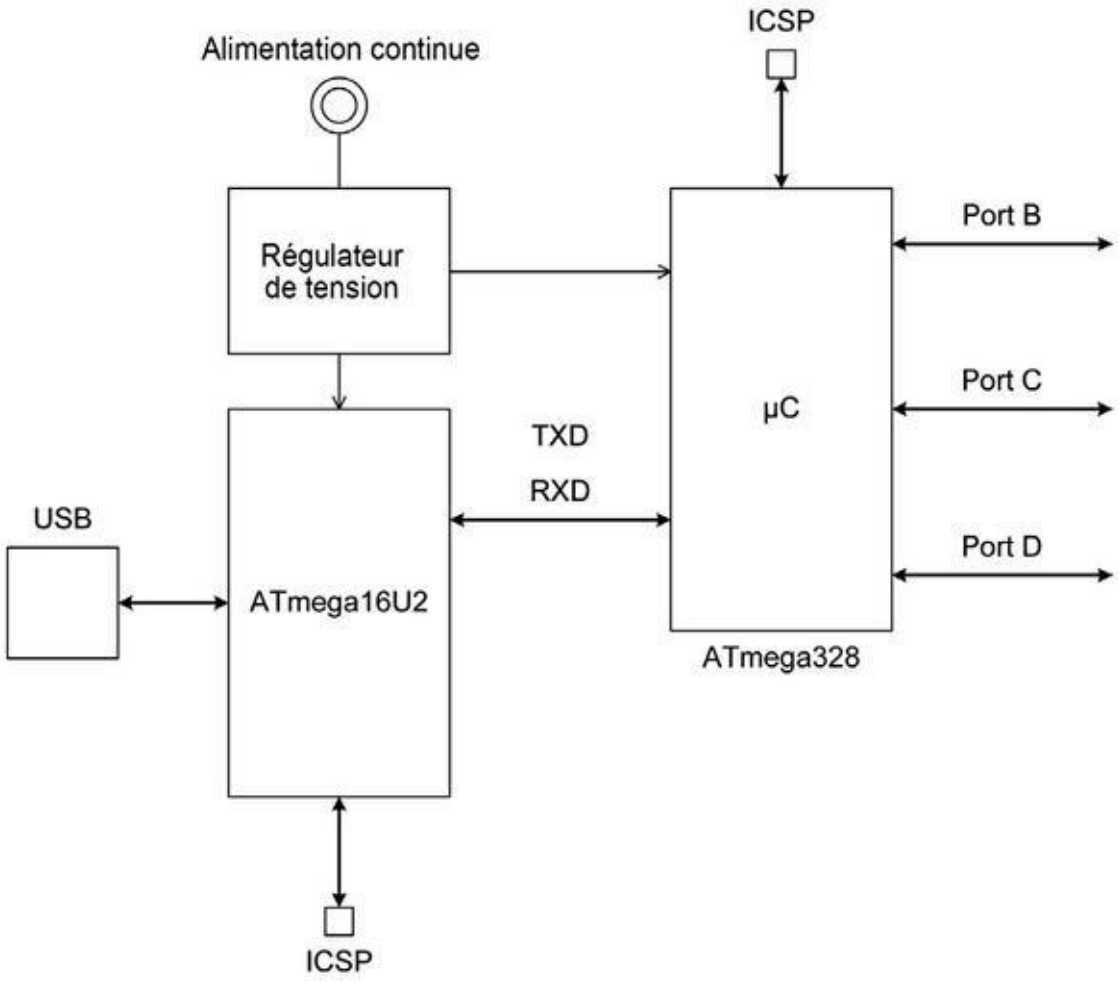


Figure 4.2 : Interface USB de l'ATmega16U2.

La Figure 4.3 donne un premier aperçu de quelques cartes Arduino. Dans le sens des aiguilles, en commençant en bas à gauche, nous voyons une Duemilanove, une Leonardo, un clone de Mega2560 de SainSmart et une Arduino Mega2560 officielle. Au centre, une Arduino Nano.

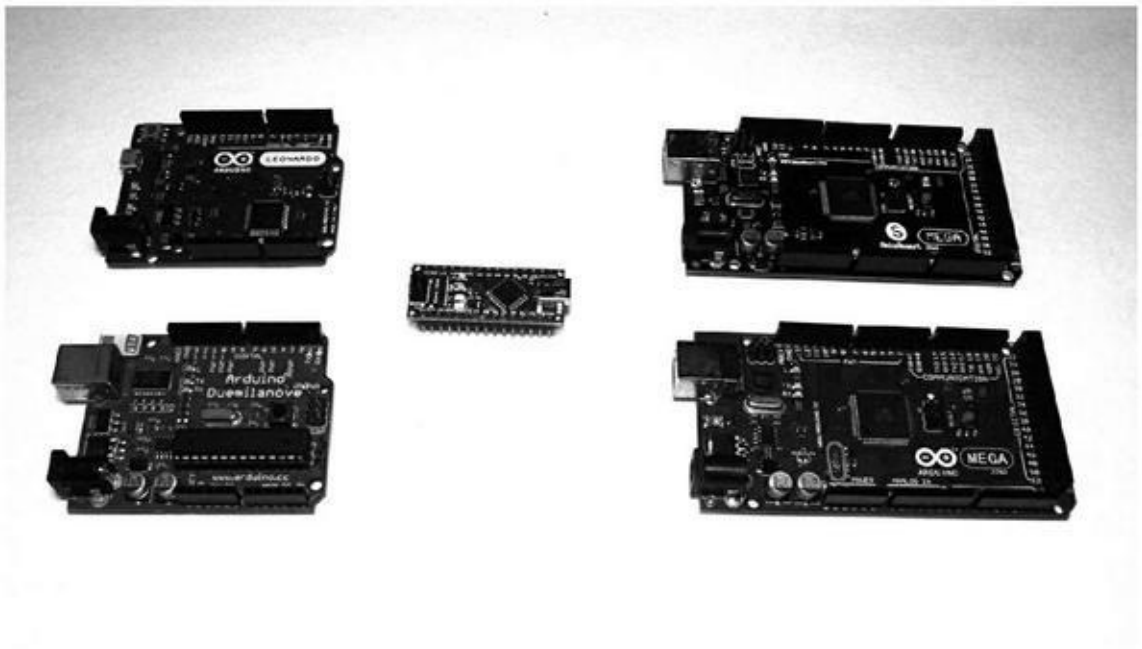


Figure 4.3 : Comparaison de quelques cartes Arduino.

Cartes Arduino au format de base

La Figure 4.4 montre l'aspect général de six cartes Arduino dans le format de base, de la Diecimila à la Leonardo, en passant par les Duemilanove et Uno. Le format de base englobe les deux variantes qui ne se distinguent que par le nombre de broches des connecteurs. L'encombrement physique est le même. C'est le format officiel sur lequel tablent toutes les cartes d'extension enfichables appelées boucliers (shields). Le brochage de ce format de carte est donné plus loin dans ce chapitre.

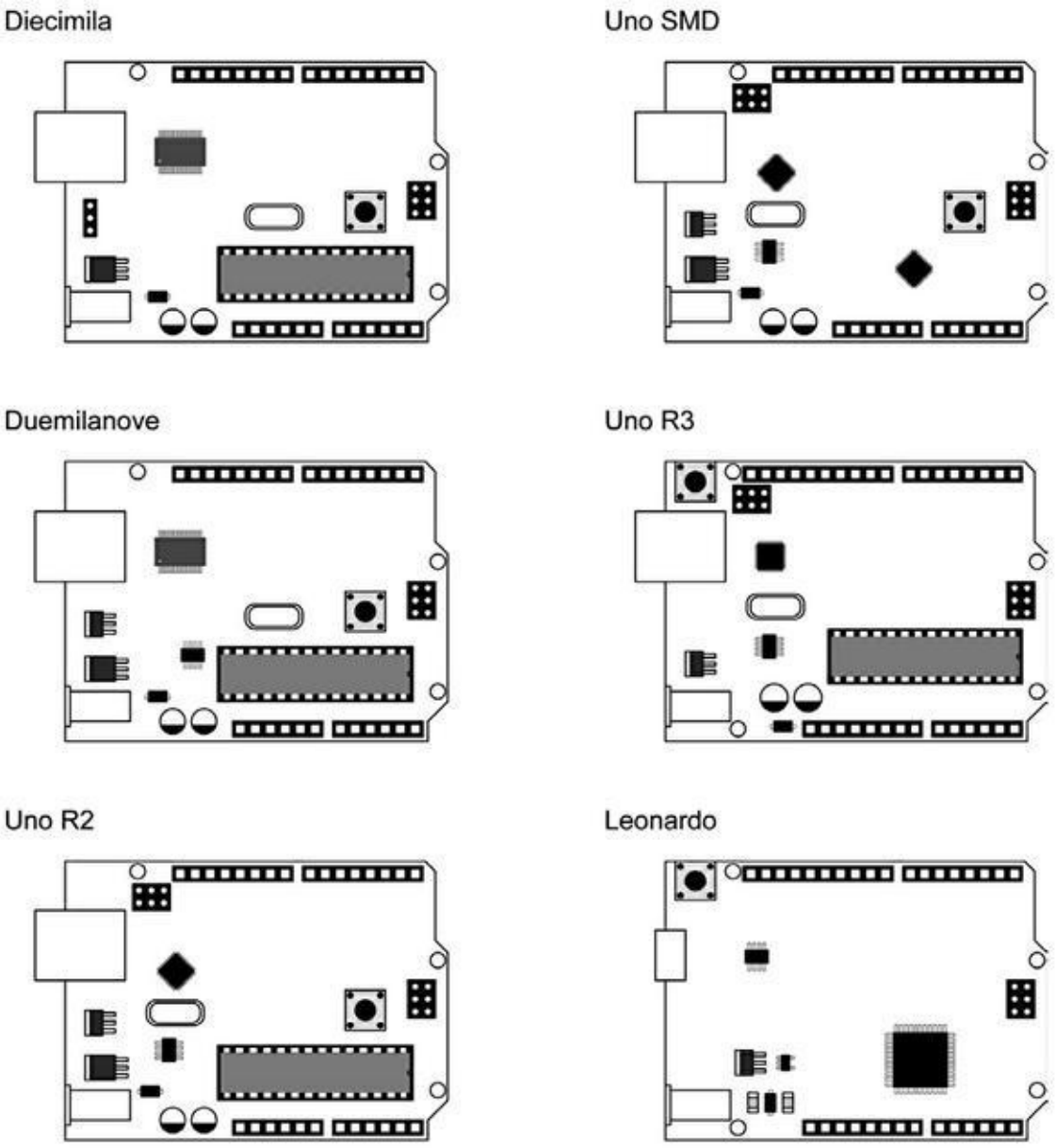


Figure 4.4 : Carte Arduino au format de base.

Les premières cartes que sont les Diecimila, Duemilanove, Uno R2 (révision 2) et Uno SMD ont la même configuration de connecteurs dite de base. L'Uno R2 a ajouté un petit bloc de six broches pour l'interface de téléversement technique ICSP (In-Circuit Serial Programming) du contrôleur ATmega16U2 qui gère l'interface USB. L'Uno SMD en est dotée elle aussi.

À partir de l'Uno R3, deux des quatre connecteurs latéraux ont été augmentés de quelques contacts en maintenant la compatibilité avec la configuration de base. Une carte bouclier prévue pour la Duemilanove reste ainsi entièrement utilisable.

Les nouveaux contacts n'apportent aucun nouvel accès au microcontrôleur et toutes les entrées-sorties de la configuration de base sont inchangées. La Leonardo est équipée du contrôleur ATmega32U4 et peut donc se passer du circuit intégré dédié à l'interface USB ; elle n'a donc qu'un port ICSP. Le brochage est inchangé, mais les ports du microcontrôleur utilisés ne sont pas les mêmes.

Toutes les cartes au format de base ont par définition les mêmes dimensions (Figure 4.5), seuls les trous de fixation ont parfois changé de position.

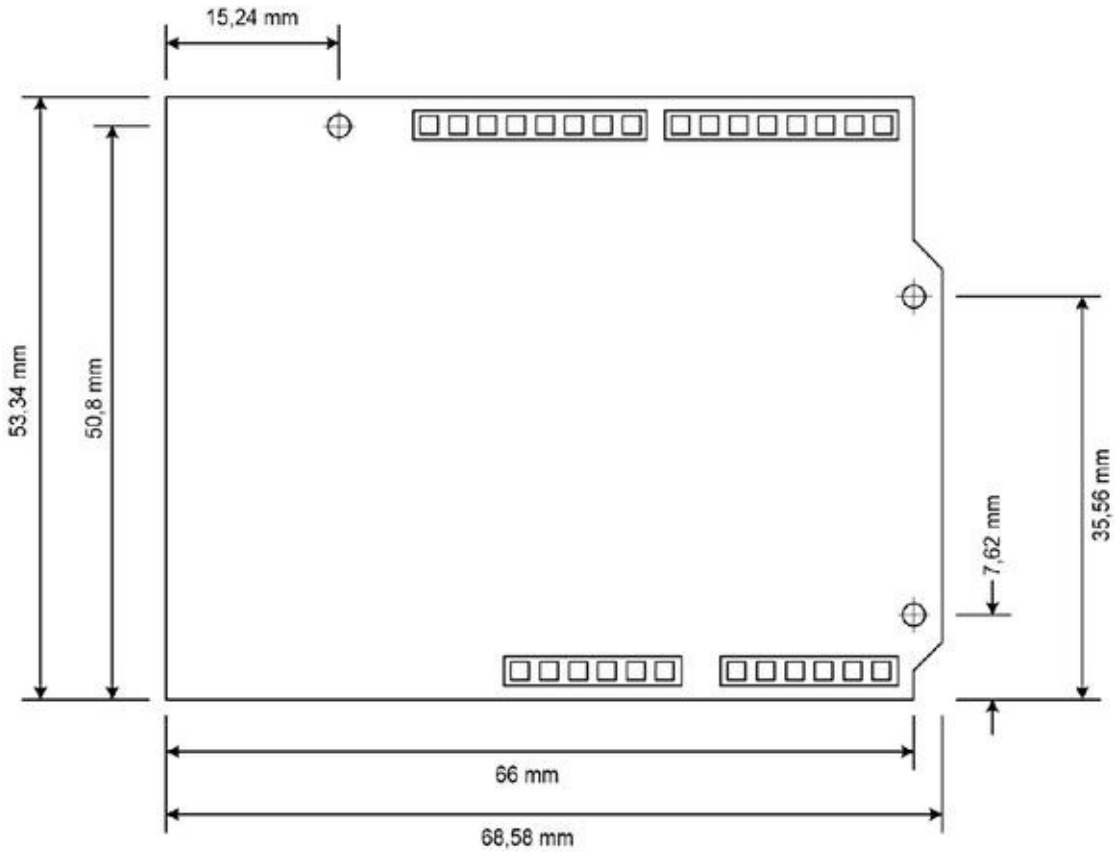


Figure 4.5 : Dimensions des formats de base et du format étendu.

Cartes Arduino au format Mega

Le format Mega conserve la configuration des connecteurs de base en y ajoutant de nombreux contacts pour donner accès aux possibilités des contrôleurs ATmega1280 et ATmega2560 (ils ont été décrits dans le Chapitre 3).

Mega et Mega2560

Les cartes Mega et Mega2560 se distinguent par le type de contrôleur, la Mega2560 remplaçant la Mega (dont la fabrication est arrêtée chez Arduino.cc). Il n'y a aucune raison de chercher à acquérir une Mega plutôt qu'une Mega2560.

La Figure 4.6 donne les dimensions externes des Mega et Mega2560. Une carte d'extension bouclier de base (pour Uno, Leonardo, *etc*.) est compatible avec le format Mega, puisque les quatorze premières entrées-sorties numériques et les six premières entrées analogiques sont aux mêmes endroits.

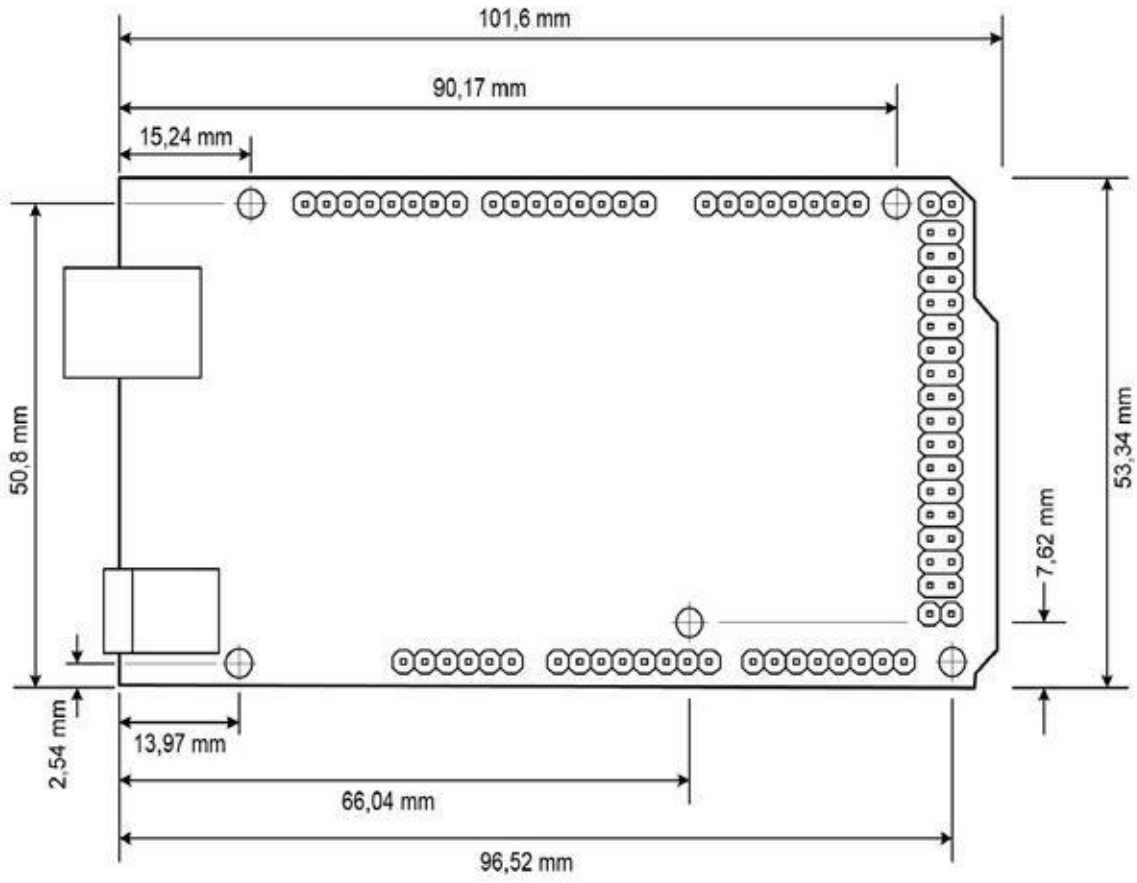


Figure 4.6 : Dimensions des cartes Mega et Mega2560.

Mega ADK

La Mega ADK est une variante de la Mega2560, avec une interface hôte USB qui lui permet d'être reliée à un téléphone sous Android et autres appareils similaires. Elle comporte donc un connecteur USB de plus entre le connecteur

USB de type B et le jack d’alimentation. Les dimensions restent les mêmes que celles de la Mega2560 et elle maintient la compatibilité avec le format de base.

Cartes Arduino miniatures

Suite au lancement du premier modèle de carte en 2005 et celui du format étendu à connecteurs prolongés en 2007, l’équipe Arduino a constaté que ce format pouvait s’avérer trop encombrant pour certaines applications. Ainsi est née la famille des cartes miniatures.

Dans cette famille, nous trouvons les Mini, Micro, Nano et Fio. Malgré la taille réduite, ces cartes embarquent les mêmes contrôleurs que leurs grandes sœurs.

Mini

La Mini peut être implantée sur une plaque d’essais sans soudure ou dans tout projet compact. Elle n’offre pas de connecteur USB. Il faut utiliser un programmateur externe pour téléverser le code exécutable dans le microcontrôleur. Les dimensions sont fournies dans la Figure 4.7.

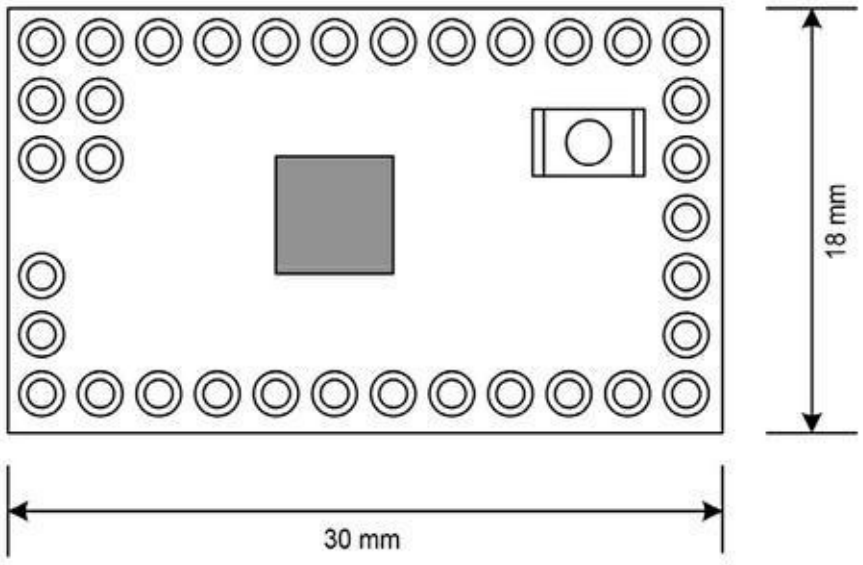


Figure 4.7 : Dimensions de l’Arduino Mini.

Pro Mini

La Pro Mini conserve les dimensions et le brochage (Figure 4.8) de la Mini mais elle est plus spécialement fabriquée pour des installations permanentes. La Pro Mini a été conçue et produite par SparkFun Electronics.

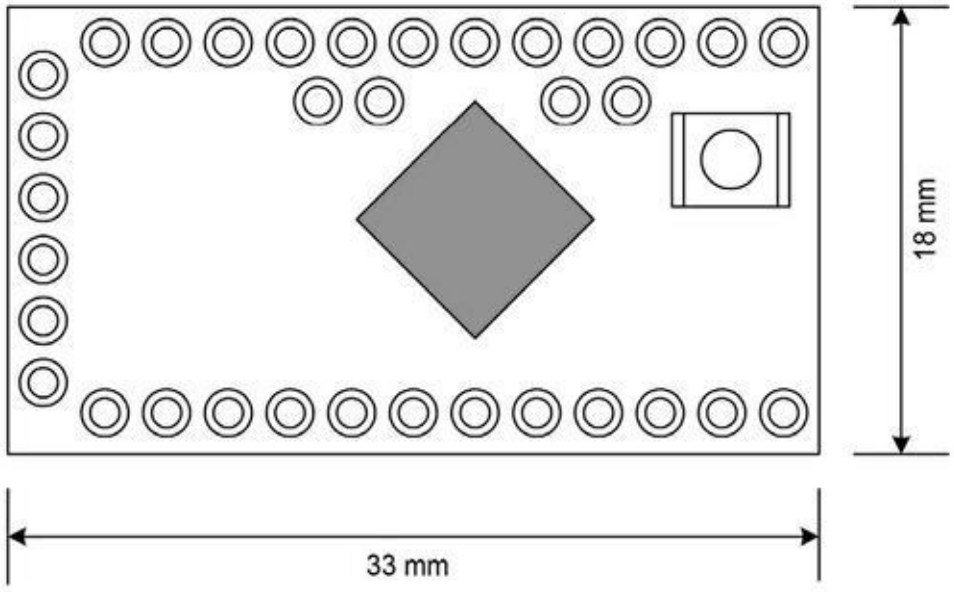


Figure 4.8 : Dimensions de l'Arduino Pro Mini.

Nano

Proche de la Mini, la Nano est enfichable dans une plaque d'essai ou comme module dans un grand circuit imprimé. Elle a été conçue et fabriquée par Gravitech. Les dimensions sont données en Figure 4.9.

N. d. T. : Contrairement aux attentes, une Nano ou une Micro ne sont pas plus petites qu'une Mini.

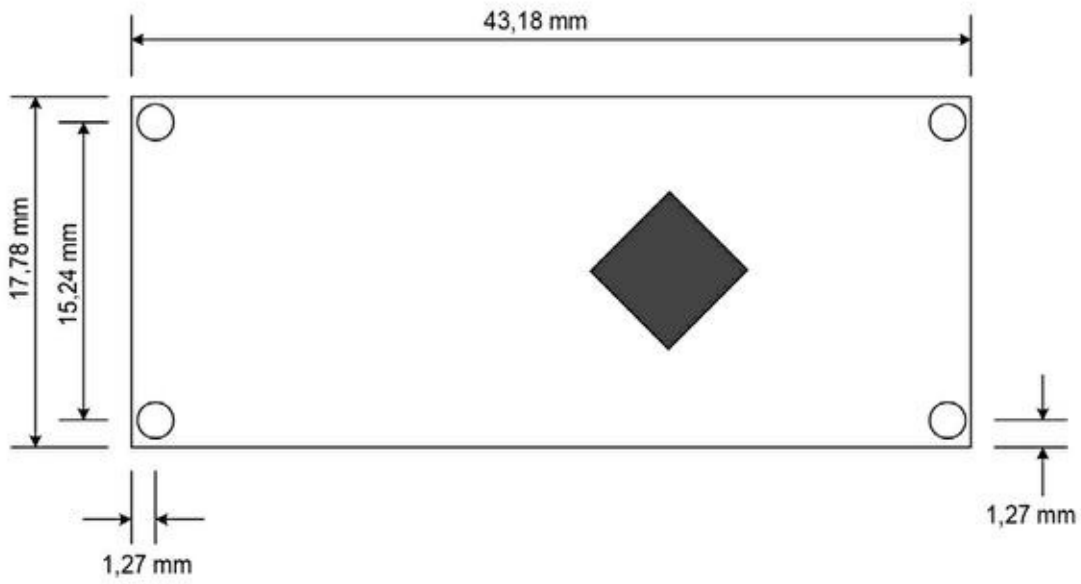


Figure 4.9 : Dimensions de l'Arduino Nano.

Fio

La Fio est dédiée aux projets sans fil, notamment par une liaison XBee. Elle ne dispose de ce fait pas des possibilités et de connexion directe des grandes cartes Arduino. Vous programmez une Fio par un adaptateur série vers USB ou USB vers XBee. Elle est conçue et produite par SparkFun Electronics (dimensions en Figure 4.10).

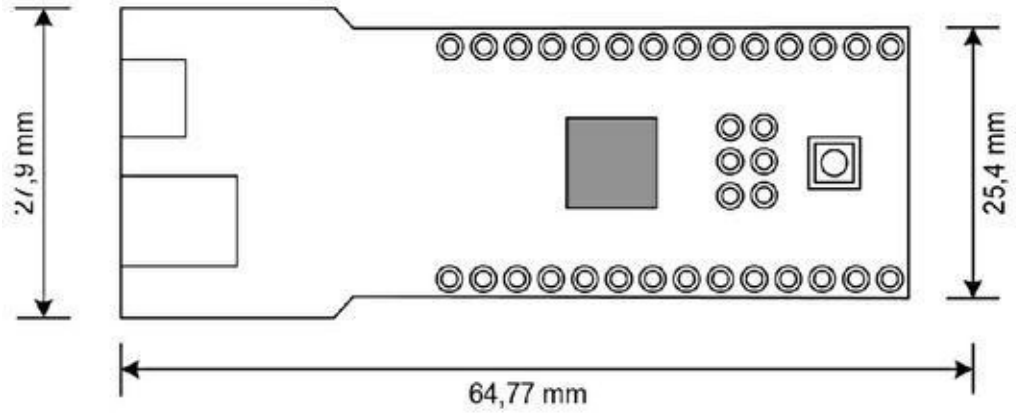


Figure 4.10 : Dimensions de l'Arduino Fio.

Micro

La Micro est originale car elle se présente avec la même empreinte qu'un circuit intégré rectangulaire à double rangée de contacts DIP (Dual In-line Package) et peut donc s'insérer aisément dans une plaque. Elle est dotée du même contrôleur ATmega32U4 que la Leonardo. Comme la Nano, la Micro peut servir de module enfichable sur un support de circuit intégré. Elle a été créée avec la société Adafruit et ses dimensions sont données dans la Figure 4.11.

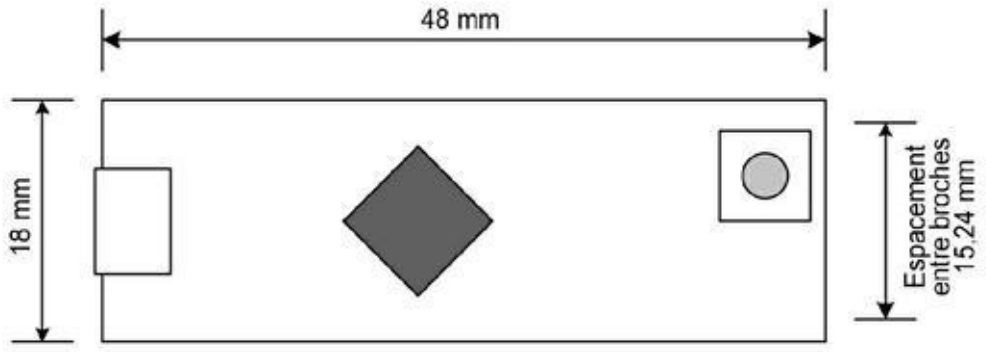


Figure 4.11 : Dimensions de l'Arduino Micro.

Cartes de format spécial

Les cartes Arduino ne sont pas obligatoirement rectangulaires. La LilyPad se présente comme un petit disque avec les points de connexion tout autour. Elle peut être intégrée à un vêtement. L'Esplora ressemble à une manette de jeu vidéo, mais elle peut faire bien plus de choses que des jeux vidéo.

LilyPad

La LilyPad et ses variantes sont destinées aux applications de vêtements intelligents. Elle mesure environ 5 cm de diamètre (Figure 4.12).

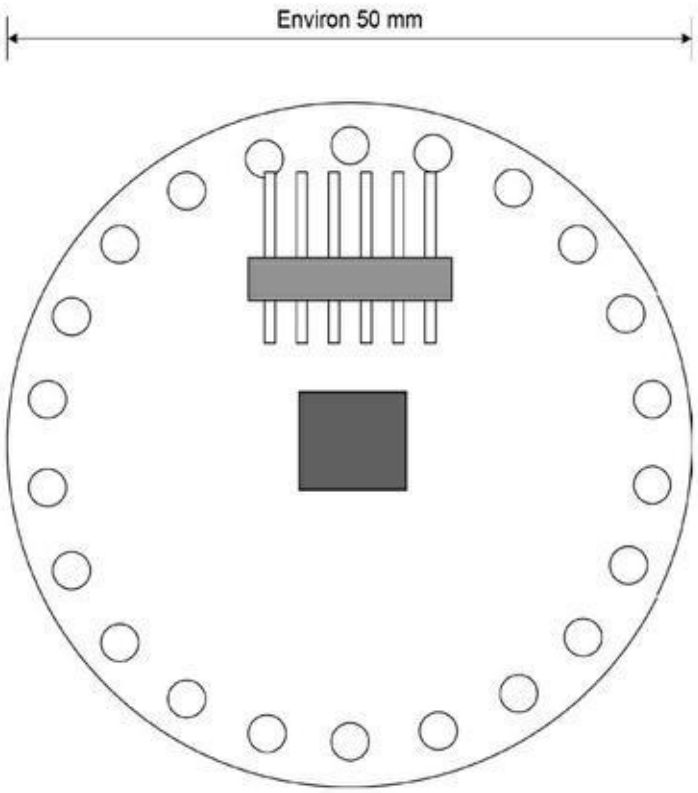


Figure 4.12 : Dimensions de l'Arduino LilyPad.

Esplora

L'Esplora est dotée de quatre boutons directionnels, d'un joystick en tout-ou-rien et d'un connecteur micro USB. Les quatre trous de fixation permettent de la fixer sur un panneau ou une façade de commande. Les dimensions sont données dans la Figure 4.13.

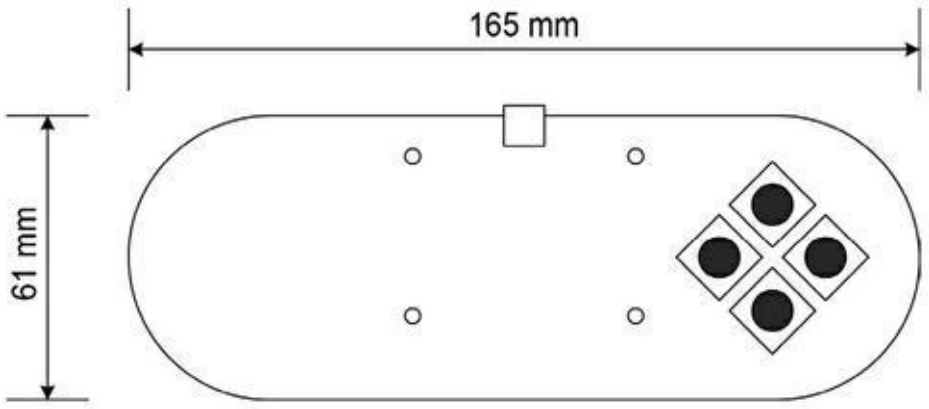


Figure 4.13 : Dimensions de l'Arduino Esplora.

Configurations de brochage Arduino

Au moment de choisir dans quel format créer une nouvelle carte d'extension bouclier Arduino, la convention est de s'en tenir au format de base présenté plus haut. Cette configuration correspond aux connecteurs des cartes lancées entre 2007 et 2012. Les cartes à format étendu comme l'Uno R3 et la Leonardo et les cartes au format allongé Mega maintiennent la compatibilité avec ce brochage.

Brochage en format original, de base

Le format de connexion Arduino de base toujours en vigueur de nos jours a été défini avec la Diecimila. Des dizaines de cartes boucliers s'y conforment. Les cartes Arduino qui présentaient (elles ne sont plus commercialisées) ce format sont listées dans le Tableau 4.2.

Tableau 4.2 : Cartes Arduino en format de base.

Carte	Année	Microcontrôleur
Diecimila	2007	ATmega168
Duemilanove	2008	ATmega168/ATmega328
Uno (R2 et SMD)	2010	ATmega328P

La Figure 4.14 montre le brochage d'une carte Arduino de base (Diecimila, Duemilanove, Uno R2 et Uno SMD).

Les connexions de ce standard Arduino sont les suivantes :

- 14 broches d'entrées-sorties numériques ;
- 1 broche de référence de tension analogique ;

- 3 broches de masse (ground) ;
- 6 broches d’entrée analogiques ;
- deux broches d’alimentation en 3,3 V et en 5 V ;
- une broche de réinitialisation Reset.

La Figure 4.14 montre que ces broches sont réparties sur quatre connecteurs, deux en haut et deux en bas dans le sens de la longueur de la carte. De chaque côté, il y a un connecteur à six broches et un autre à huit broches.

Au niveau programmation, chaque broche d’une carte Diecimila, Duemilanove, Uno R2 ou Uno SMD PCB porte un nom unique pour la désigner dans le code source. C’est ce nom qui est imprimé sur le circuit Arduino. Le Tableau 4.3 donne le brochage de base pour une Arduino R2 dotée d’un ATmega168 ou d’un ATmega328. Pour la variante Uno SND, voyez le Tableau 4.5 dédié à la carte Arduino Ethernet.

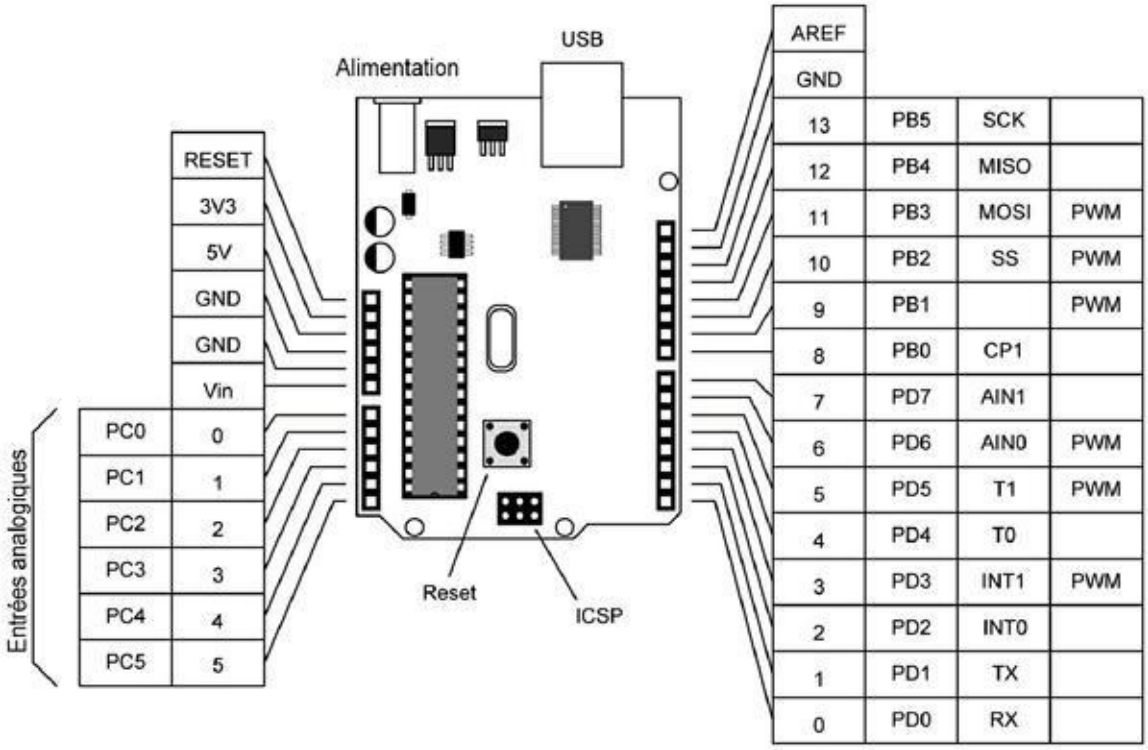


Figure 4.14 : Affectations des broches du format Arduino de base.

Tableau 4.3 : Affectation des broches Arduino ATmega168/328.

Dn	An	AVR	Port	Fonctions	PWM ?
0		2	PD0	RxD	
1		3	PD1	TxD	
2		4	PD2	INT0	
3		5	PD3	INT1, OC2B	Oui
4		6	PD4	T0, XCK	
5		11	PD5	T1	Oui
6		12	PD6	AIN0	Oui
7		13	PD7	AIN1	
8		14	PB0	CLK0, ICP1	
9		15	PB1	OC1A	Oui
10		16	PB2	OC1B, SS	Oui
11		17	PB3	OC2A, MOSI	Oui
12		18	PB4	MISO	
13		19	PB5	SCK	
14	0	23	PC0		
15	1	24	PC1		
16	2	25	PC2		
17	3	26	PC3		
18	4	27	PC4	SDA	
19	5	28	PC5	SCL	

Dn = Broche numérique

An = Broche analogique

AVR = Broche AVR

Brochage en format étendu (type Uno R3)

La version R3 de la carte Uno a prolongé deux des quatre connecteurs latéraux. Dans celui du haut, deux nouvelles broches donnent accès aux signaux SCL et SDA du protocole I2C. Dans celui du bas, une broche sert à la tension de référence IOREF (qui sera à 3,3 V ou à 5 V, selon la carte) et l'autre n'est pas connectée. Le Tableau 4.4 liste les cartes concernées.

Tableau 4.4 : Cartes Arduino en format étendu.

Carte	Année	Microcontrôleur
Uno R3	2010	ATmega328
Ethernet	2011	ATmega328
Leonardo	2012	ATmega32U4

Uno R3

Comme l'Uno R2 et l'Uno SMD, l'Uno R3 embarque un second microcontrôleur pour gérer les communications USB, alors que l'Arduino Ethernet n'a pas de liaison USB. La Figure 4.15 propose le schéma fonctionnel de l'Uno R3 et de l'Uno SMD.

Les affectations de broches de l'Uno R3 sont données en Figure 4.16.

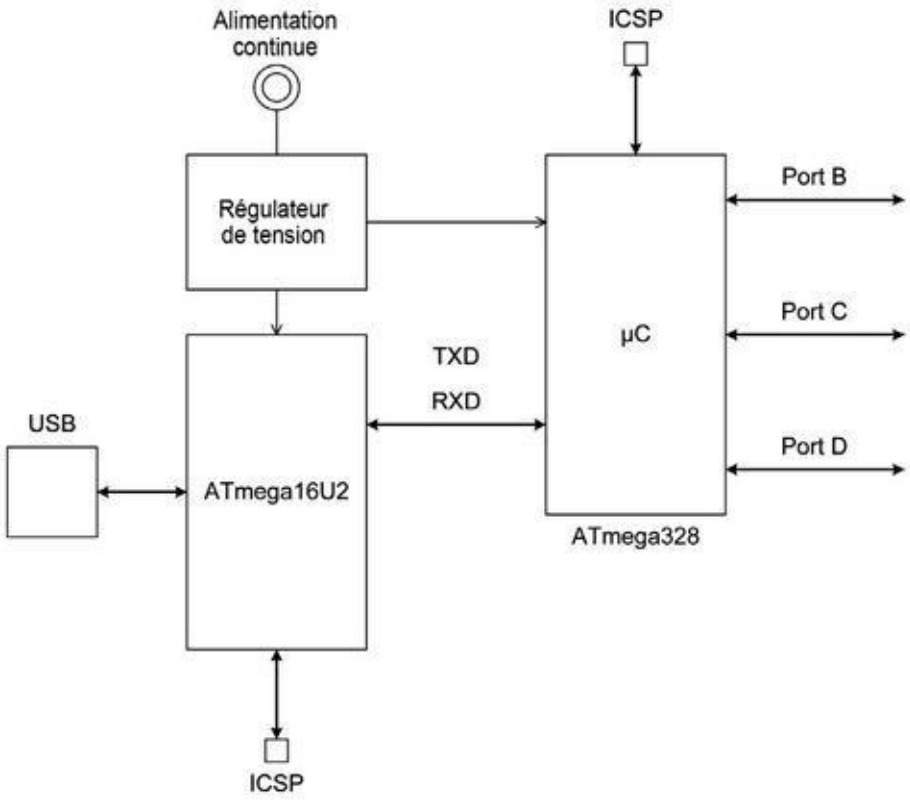


Figure 4.15 : Schéma fonctionnel de l'Uno R3.

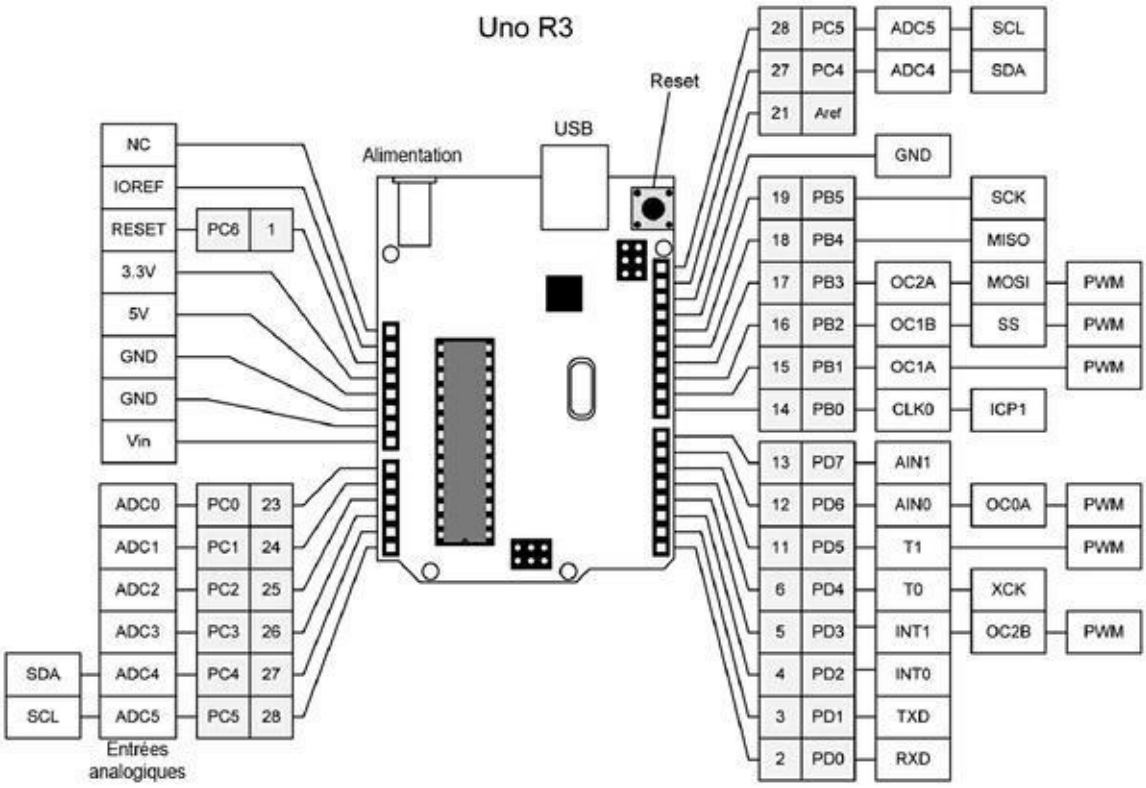


Figure 4.16 : Affectations des broches de l'Uno R3.

Les affectations de broches sont les mêmes que celles du format de base (Tableau 4.3). Celles de la Leonardo sont données plus loin dans la section dédiée à ce modèle de carte.

Ethernet

La carte Ethernet se distingue des autres par son interface Ethernet à 100 Mb/s et son connecteur RJ45. Elle n'a pas d'interface USB. Le contrôleur est soudé en surface et les affectations de broches sont différentes de celles de l'ATmega328. L'interface Ethernet est gérée par un circuit WIZnet W5100. La Figure 4.17 en donne le schéma fonctionnel.

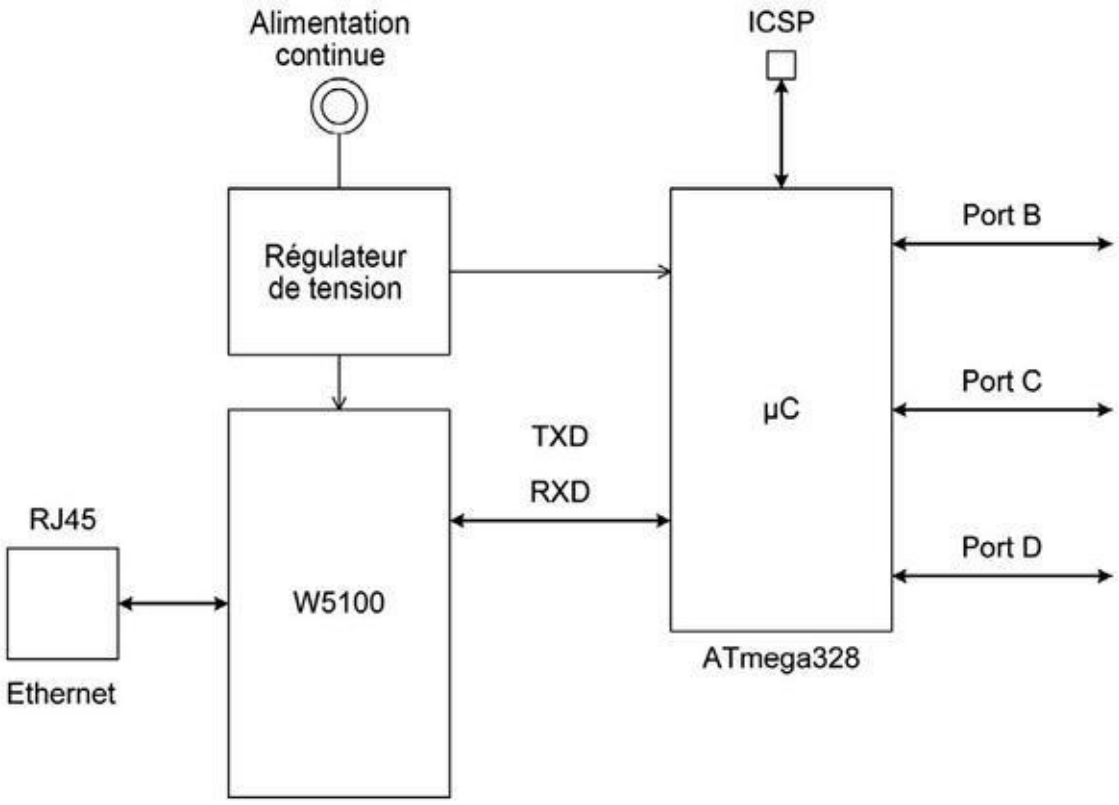


Figure 4.17 : Schéma du circuit Arduino Ethernet.

Le programme est implanté *via* un circuit FTDI par un adaptateur, comme les modèles FTDI de SparkFun et Adafruit. L'interface est rendue accessible sur un connecteur coudé à six broches sur un bord du circuit imprimé, près du porte-

carte mémoire microSD. La Figure 4.18 montre les affectations de broches de la carte Ethernet.

Ce modèle n'est plus commercialisé par Arduino.cc, mais reste disponible chez d'autres fabricants. Dorénavant, l'accès à Ethernet est réalisable par une carte bouclier Ethernet (voir le Chapitre 8 à ce sujet).

Le Tableau 4.5 donne les affectations de broches de la carte Arduino Ethernet. Notez que les broches 10, 11, 12 et 13 sont réservées pour l'interface Ethernet. Elles ne peuvent pas servir à d'autres usages.

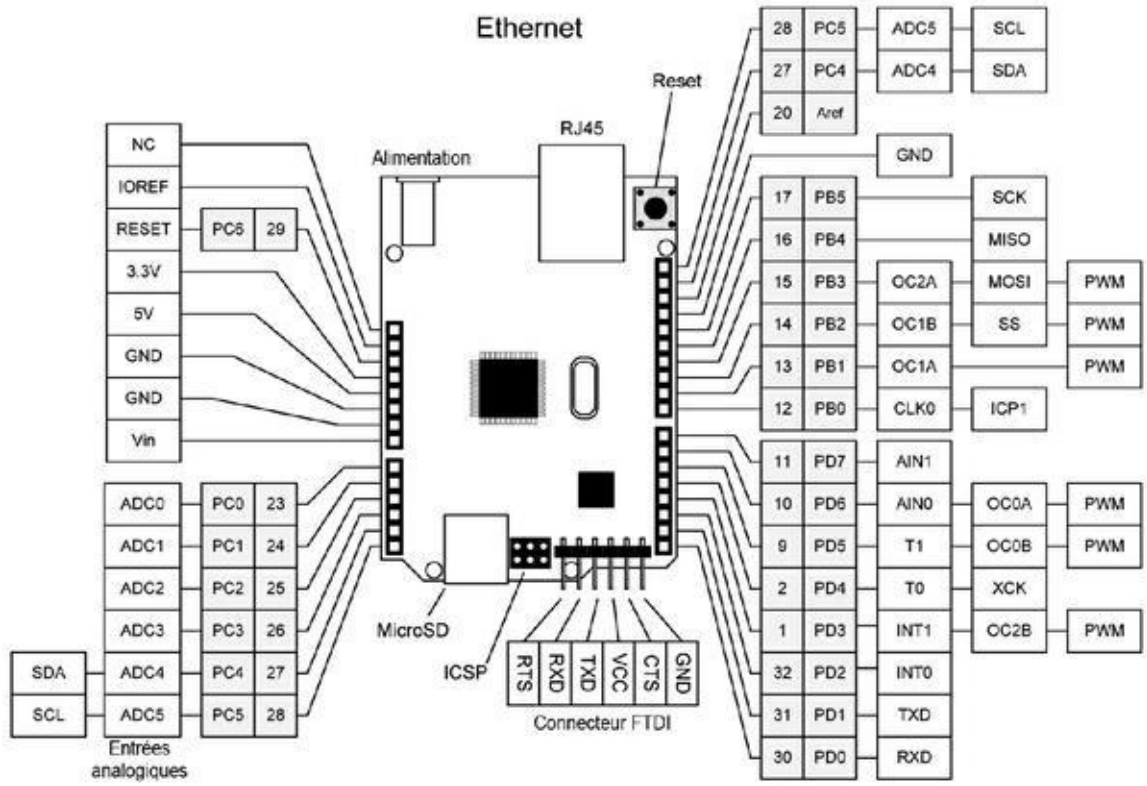


Figure 4.18 : Affectations des broches de la carte Arduino Ethernet.

Tableau 4.5 : Affectation des broches Arduino Ethernet.

Dn	An	AVR	Port	Fonctions	PWM ?
0		30	PD0	RxD	
1		31	PD1	TxD	
2		32	PD2	INT0	

3		1	PD3	INT1, OC2B	Oui
4		2	PD4	T0, XCK	
5		9	PD5	T1, OC0B	Oui
6		10	PD6	AIN0, OC0A	Oui
7		11	PD7	AIN1	
8		12	PB0	CLK0, ICP1	
9		13	PB1	OC1A	Oui
10		14	PB2	OC1B, SS	Oui
11		15	PB3	OC2A, MOSI	Oui
12		16	PB4	MISO	
13		17	PB5	SCK	
14	0	23	PC0		
15	1	24	PC1		
16	2	25	PC2		
17	3	26	PC3		
18	4	27	PC4	SDA	
19	5	28	PC5	SCL	

Leonardo

La Leonardo est la première à utiliser le contrôleur hybride ATmega32U4 qui contient une interface USB et de nouvelles fonctions. Le circuit imprimé en est simplifié (Figure 4.19). Vous pouvez constater que la Leonardo utilise un connecteur mini-USB au lieu du connecteur carré habituel de type B. Ce changement a été bien accueilli, car il permet d’exploiter la Leonardo avec des

boucliers qui venaient buter sur le connecteur USB de type B sur les anciens modèles.

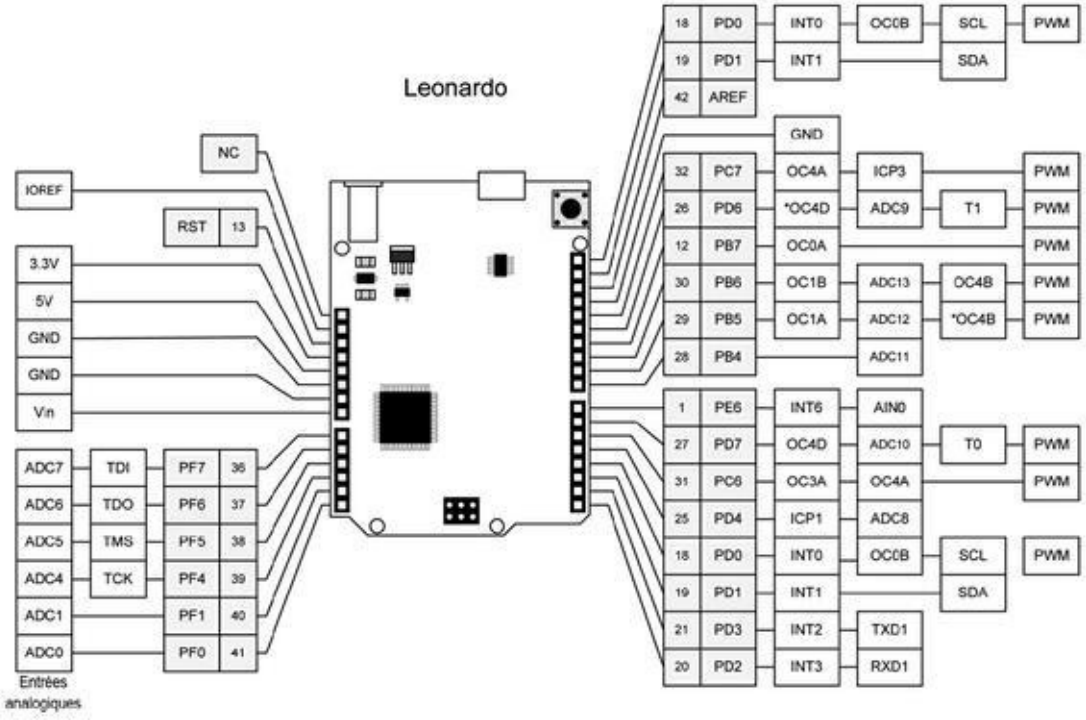


Figure 4.19 : Affectations des broches du Leonardo.

Les Uno R3 et Leonardo offrent les mêmes affectations de broches, mais certaines fonctions du microcontrôleur sont différentes. Dans l'atelier Arduino, cette divergence est masquée par des définitions spécifiques à chaque carte pour faire correspondre les broches aux fonctions.

Le Tableau 4.6 présente les affectations de broches du format étendu dit Uno R3 avec contrôleur ATmega32U4.

Tableau 4.6 : Affectation des broches Arduino Leonardo (ATmega32U4).

Dn	An	AVR	Port	Fonctions	PWM ?
0		20	PD2	INT3, RxD1	
1		21	PD3	INT2, TxD1	
2		19	PD1	INT1, SDA	

3		18	PD0	INT0, OC0B, SCL	Oui
4		25	PD4	ICP1, ADC8	
5		31	PC6	OC3A, OC4A	Oui
6		27	PD7	OC4D, ADC10, T0	Oui
7		1	PE6	INT6, AIN0	
8		28	PB4	ADC11	
9		29	PB5	OC1A, ADC12, *OC4B	Oui
10		30	PB6	OC1B, ADC13, OC4B	Oui
11		12	PB7	OC0A	Oui
12		26	PD6	OC4D, ADC9, T1	Oui
13		32	PC7	OC4A, ICP3	Oui
14	0	36	PF7	TDI	
15	1	37	PF6	TDO	
16	2	38	PF5	TMS	
17	3	39	PF4	TCK	
18	4	40	PF1		
19	5	41	PF0		

Brochage en format Mega

La série Mega (basée sur les contrôleurs ATmega1280 et ATmega2560) maintient le brochage standard en y ajoutant de nombreuses broches d'entrées-sorties (Figure 4.20). Les cartes concernées sont listées dans le Tableau 4.7. La plupart des boucliers sont compatibles avec les cartes au format Mega.

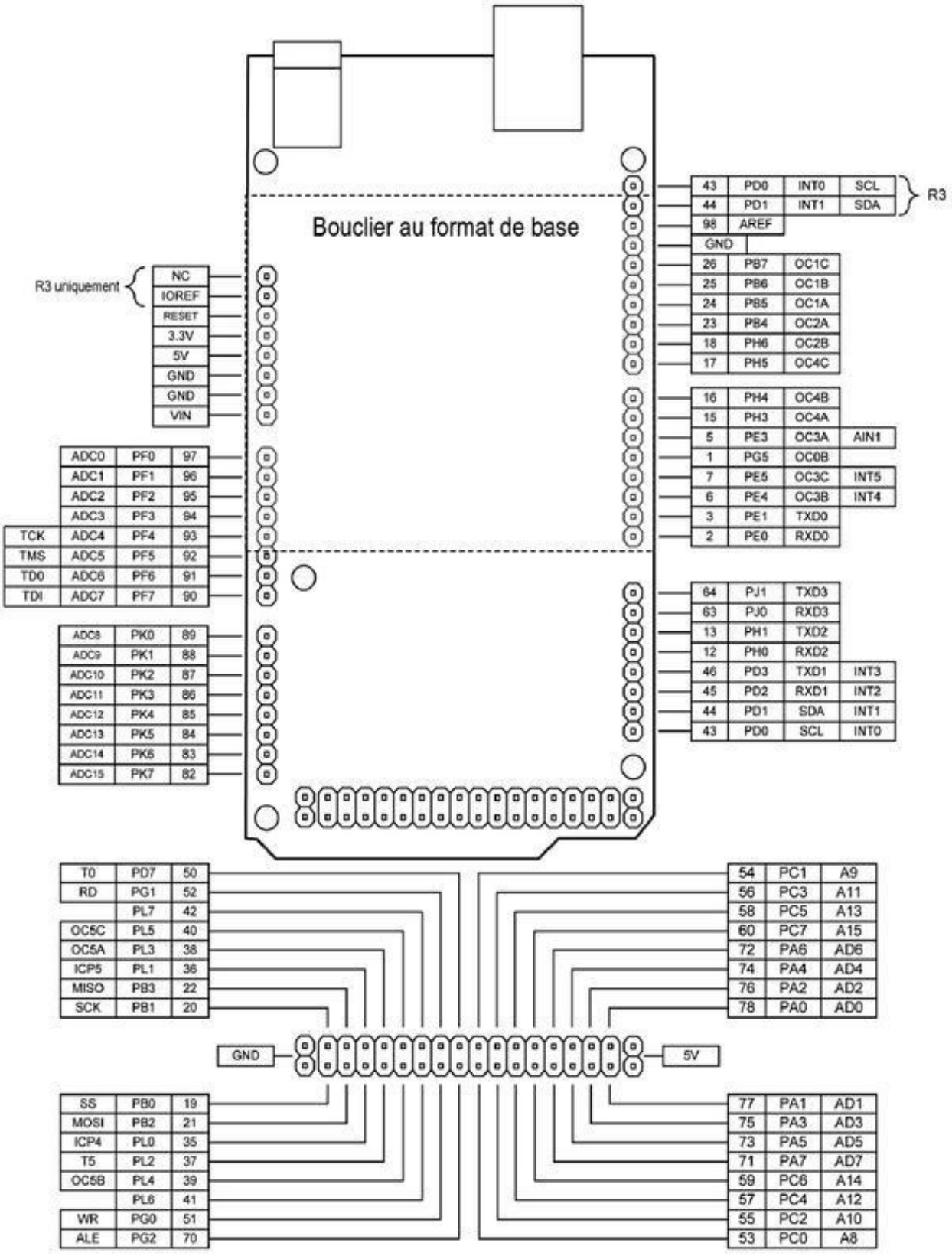


Figure 4.20 : Affectations des broches en format Mega.

Dans cette Figure 4.20, les affectations des broches PCINT sont masquées pour maintenir la lisibilité. Notez que la version R3 de la Mega2560 ajoute certaines broches, mais elles ne réduisent pas la compatibilité avec les boucliers conçus pour le format Arduino de base.

Tableau 4.7 : Cartes Arduino en format Mega.

Carte	Année	Microcontrôleur
Mega	2009	ATmega1280
Mega2560	2010	ATmega2560
Mega ADK	2011	ATmega2560

Brochage en format spécifique

Les cartes dites spécifiques, non standard, sont celles dont le format s'éloigne du rectangle avec deux groupes de deux connecteurs latéraux. La plus radicale de cette famille est la LilyPad, puisqu'elle est ronde et n'offre que des points de soudure et aucun connecteur. Les petites Nano, Mini, Mini Pro et Micro disposent de connecteurs soudés sur le dessous et peuvent donc être installées sur une plaque d'essai sans soudure ou ajoutées à un circuit imprimé de taille suffisante. Les contacts de la Fio offrent un espacement compatible avec les connecteurs standard.

Aucune des cartes de cette famille ne peut accepter directement un bouclier du format de base. Les cartes en format spécifique sont listées dans le Tableau 4.8 et les affectations des broches pour ces cartes sont données dans les Figures 4.21 à 4.27.

Tableau 4.8 : Cartes à brochage spécifique.

Carte	Année	Microcontrôleur
LilyPad	2007	ATmega168V/ATmega328V
Nano	2008	ATmega328/ATmega168

Mini	2008	ATmega168
Pro Mini	2008	ATmega328
Fio	2010	ATmega328P
Esplora	2012	ATmega32U4
Micro	2012	ATmega32U4

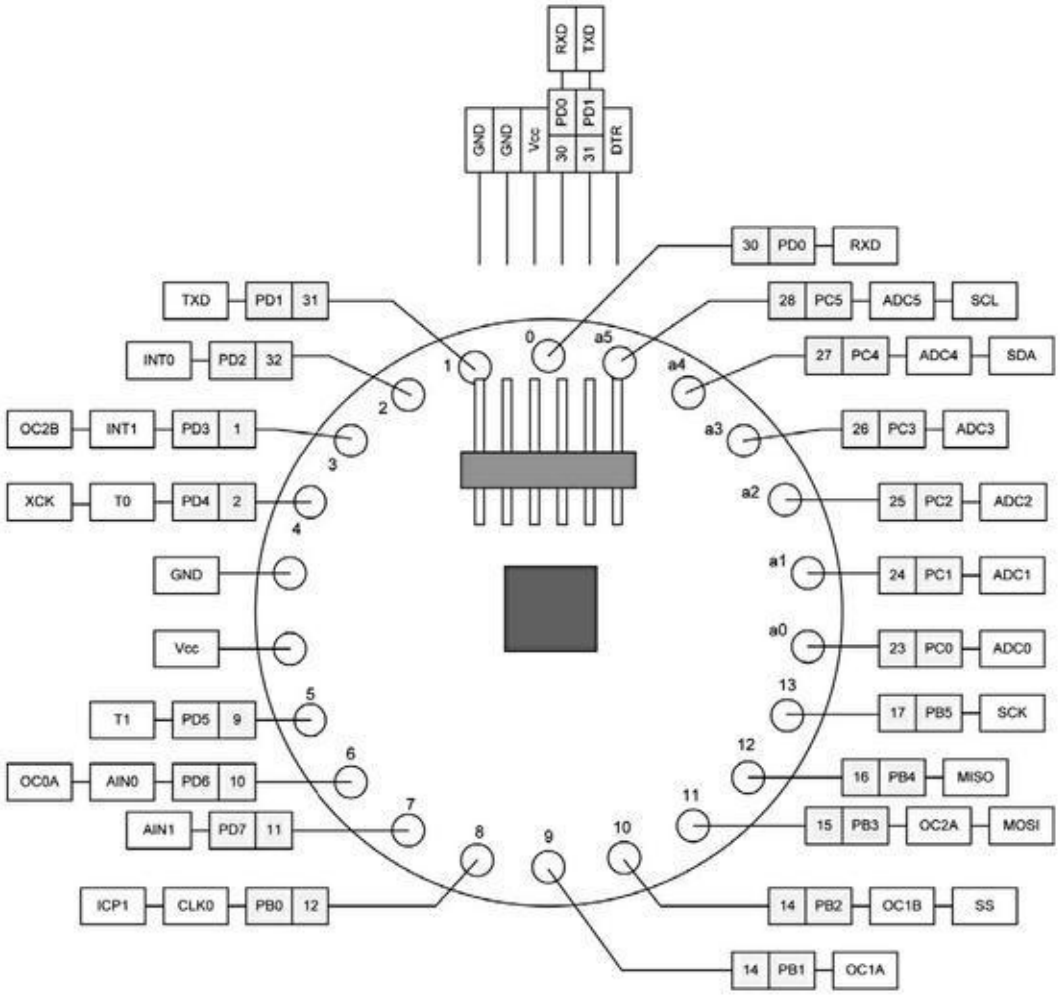


Figure 4.21 : Affectations des broches de la LilyPad.

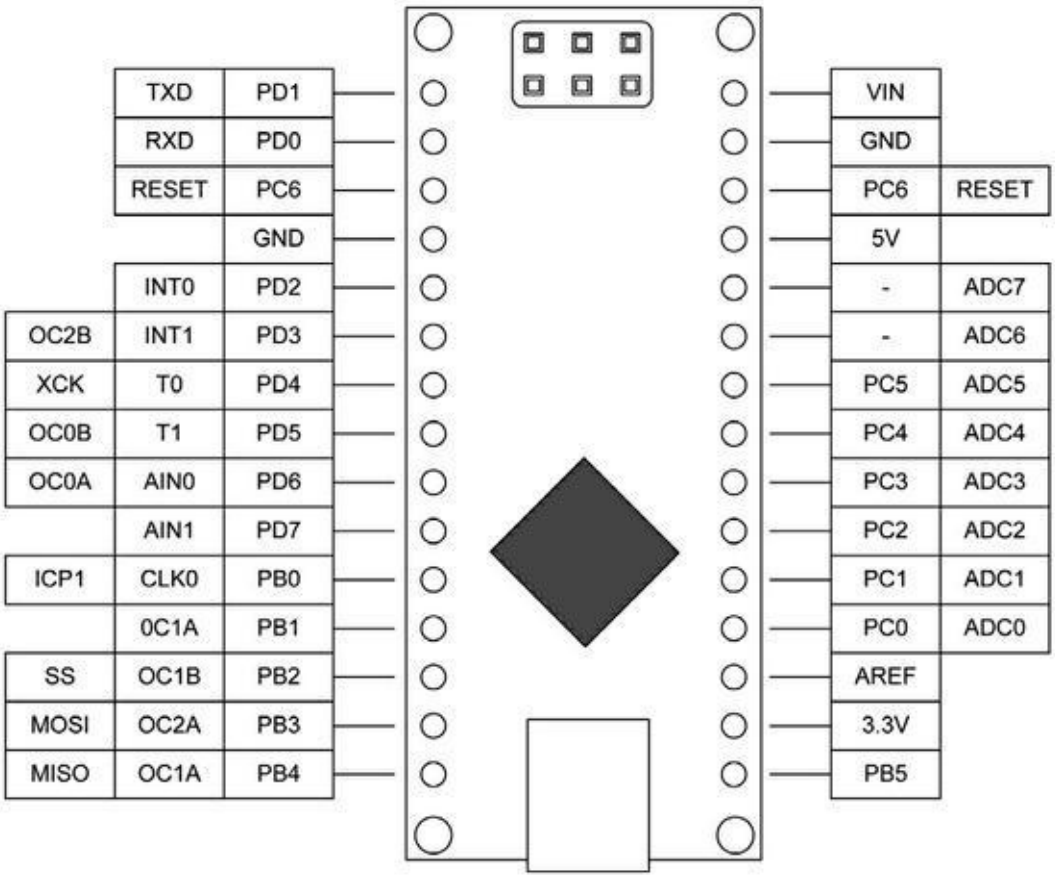


Figure 4.22 : Affectations des broches de la Nano.

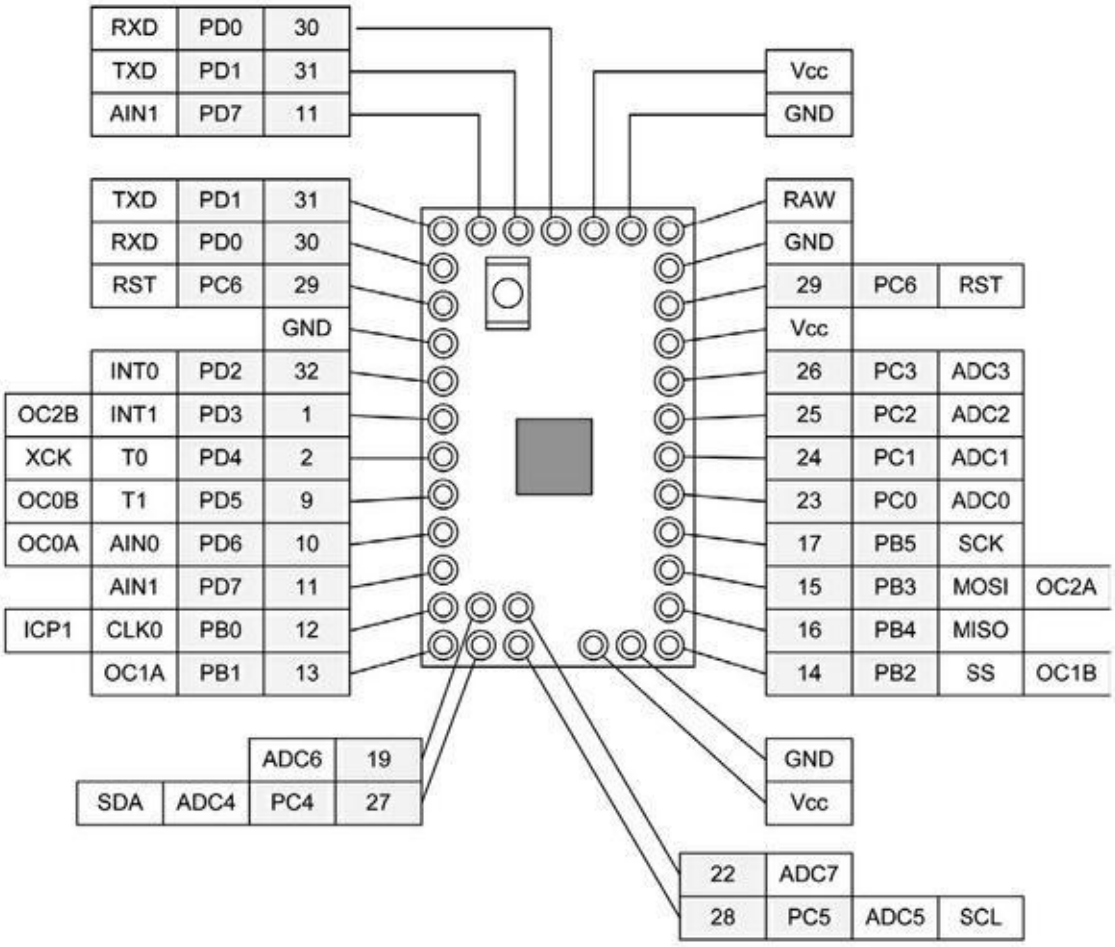


Figure 4.23 : Affectations des broches de la Mini.

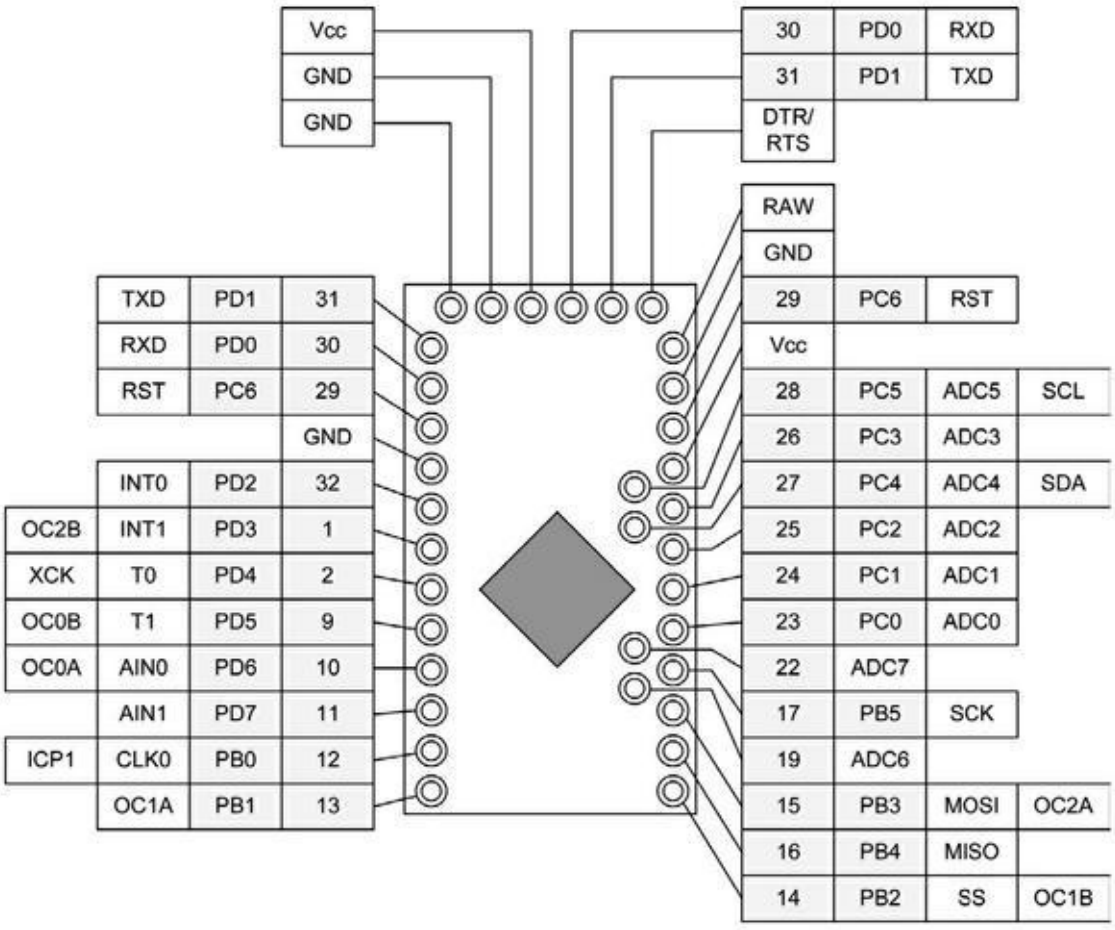


Figure 4.24 : Affectations des broches de la Pro Mini.

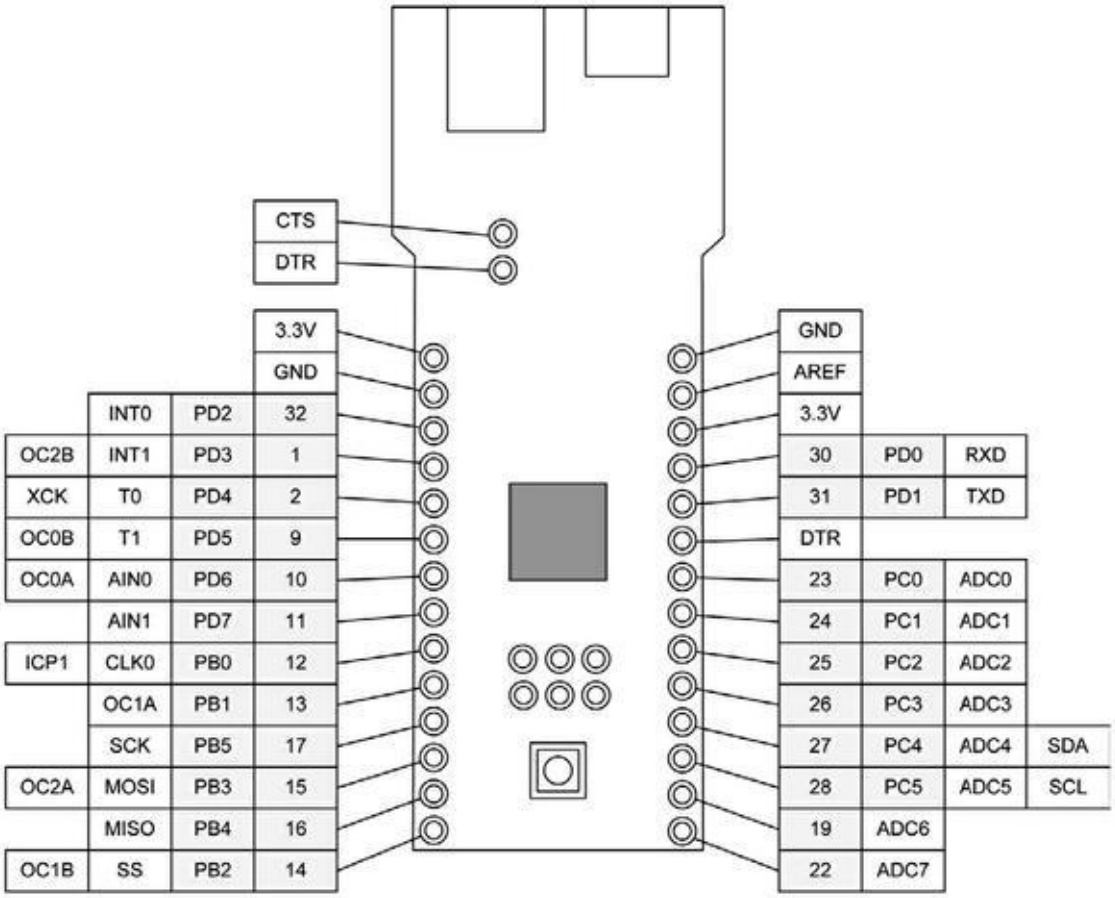


Figure 4.25 : Affectations des broches de la Fio.

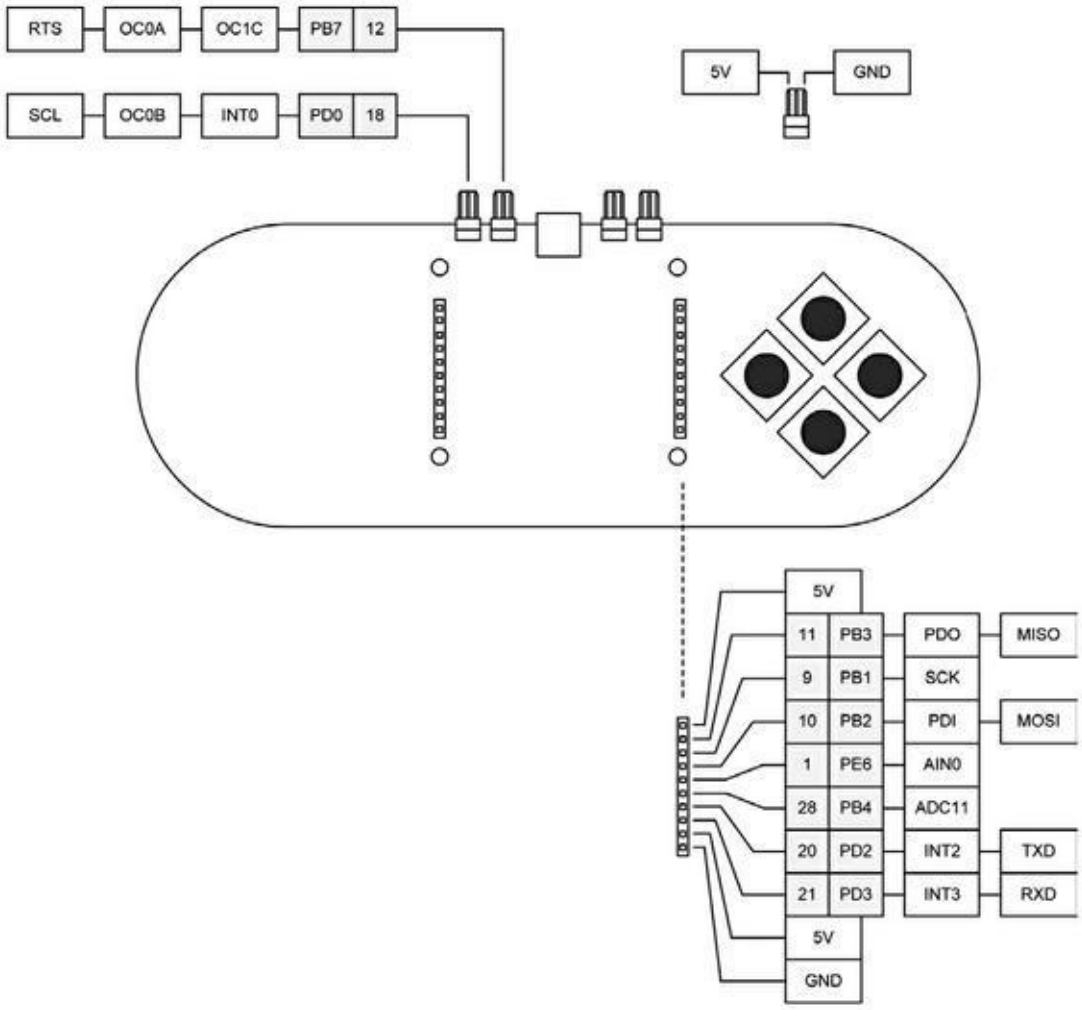


Figure 4.26 : Affectations des broches de l'Esplora.

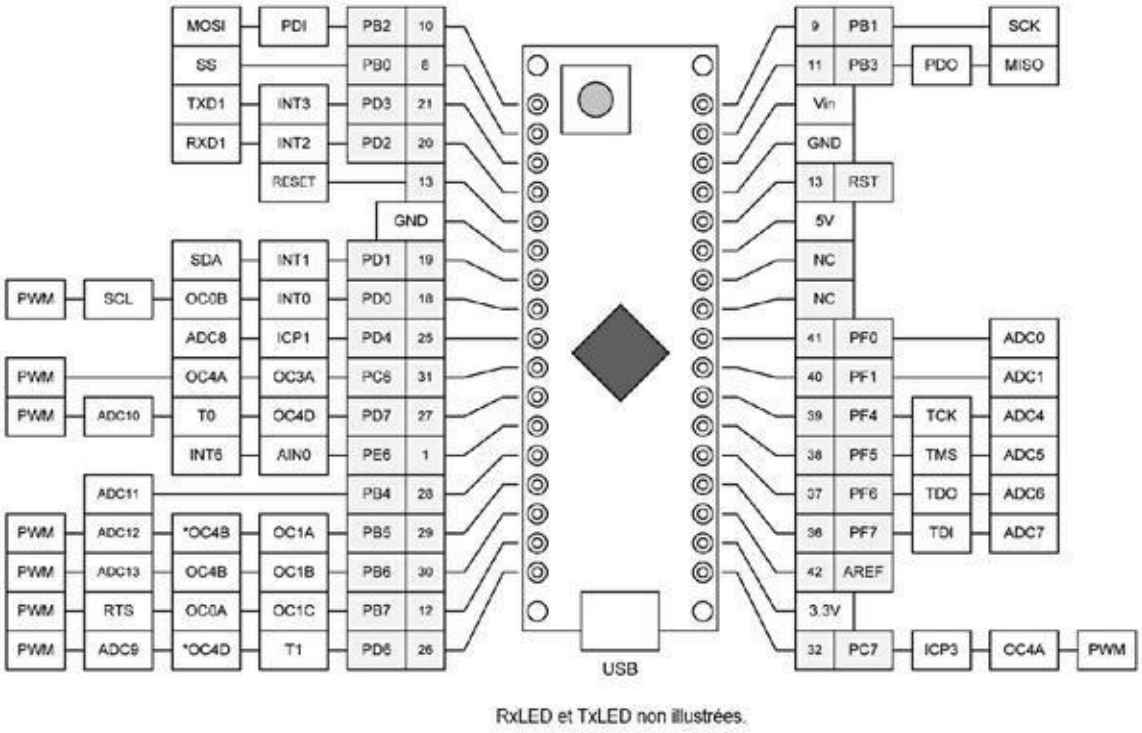


Figure 4.27 : Affectations des broches de la Micro.

Pour aller plus loin

Le site officiel Atmel propose les fiches techniques et des exemples de programmes pour tous les microcontrôleurs AVR.

CHAPITRE 5
Programmation Arduino

Dans ce chapitre, je vous propose de plonger dans la découverte des concepts, techniques et outils qui permettent successivement de rédiger, compiler, lier puis télécharger un programme sur une carte Arduino. Je vais aborder plusieurs sujets importants, et l'espace disponible ne me permet pas de les décrire chacun à leur juste mesure. Mon objectif est de vous donner suffisamment d'informations pour acquérir une certaine intimité avec un microcontrôleur. Vous allez sans doute apprendre des choses aussi au sujet de l'environnement Arduino que vous ne saviez pas encore.

Ce chapitre n'a pas prévu de décrire les langages de programmation C et C++. Ils font l'objet d'un assez grand nombre de livres, notamment chez le même éditeur. Voyez également l'Annexe D. Ici, je veux expliquer comment le code source d'un programme que l'on appelle un croquis (*sketchbook*) est transformé en code exécutable binaire pouvant être compris par le microcontrôleur d'une carte Arduino.

Je vais commencer par présenter la technique de compilation croisée ou *cross-compilation*. Il s'agit d'utiliser un compilateur et d'autres outils sur un système informatique en vue de produire un programme qui sera exécutable sur un autre système, dont l'architecture est différente. C'est exactement l'objectif de l'environnement de développement intégré Arduino, le fameux atelier IDE (*Integrated Development Environment*). Dans le Chapitre 6, nous irons encore plus en détail dans le bas niveau du développement, tel qu'il est masqué par l'atelier IDE. Pour l'instant, voyons ce que fait cet atelier et comment bien l'utiliser.

Nous verrons ensuite le logiciel essentiel appelé amorceur ou bootloader, tel qu'il est présent sur les microcontrôleurs de la famille AVR. Ce logiciel est indispensable dans un microcontrôleur ; en sachant comment il fonctionne, vous éviterez certaines déceptions et commencerez à acquérir des compétences pour savoir un jour installer votre propre amorceur à la place de celui fourni. Nous verrons dans le Chapitre 6 comment procéder à la mise en place d'un nouvel amorceur, éventuellement écrit par vous-même.

Nous découvrons ensuite l'atelier IDE Arduino. Nous verrons comment l'installer sur différents systèmes et comment configurer vos préférences. Nous irons voir les coulisses de l'atelier et en quoi il constitue une évolution à partir d'autres outils. Nous verrons en action les différentes étapes de création d'un programme Arduino à travers un programme d'alarme.

La fin du chapitre va s'intéresser au code source Arduino, et je vous dirai où vous le procurer pour l'étudier. En parcourant ce code source, vous trouverez des réponses à certaines questions sur le fonctionnement des librairies essentielles, de l'atelier et de ses composants. Vous aurez ainsi toutes les cartes en main pour décider ou non d'utiliser d'autres outils pour écrire des programmes Arduino.

Je rappelle que l'objectif principal de ce livre concerne le matériel Arduino, c'est-à-dire les boucliers, modules, capteurs et composants. La partie logicielle n'est destinée qu'à illustrer ces éléments. Vous ne trouverez pas dans ce livre de tutoriel de programmation. Le code source des exemples est néanmoins disponible sur le référentiel GitHub et dans la page du livre chez l'éditeur.

Cross-compilation pour microcontrôleurs

Quasiment tous les systèmes monocartes à microcontrôleur n'offrent qu'une puissance insuffisante pour vous permettre de créer leurs programmes localement. La fabrication des programmes est donc faite sur un autre système, qui peut être Windows, Linux ou macOS. Le résultat compilé peut ensuite être transféré, c'est-à-dire téléversé, vers le microcontrôleur AVR de la carte Arduino. Le système sur lequel le programme est développé est nommé machine de développement ou *machine hôte*, la carte Arduino étant la *machine cible*. Ce mode de réalisation existe depuis un certain temps, car il est incontournable lorsque la machine cible ne dispose pas d'assez de capacités pour rédiger, compiler et lier le programme.

Cette technique porte le nom de compilation croisée ou *cross-compilation*. Un microcontrôleur tel que ceux de la famille AVR n'a tout simplement pas assez d'espace mémoire et de puissance de traitement pour compiler un programme en langage C ou C++. Voilà pourquoi on utilise une machine plus puissante, avec un disque dur volumineux et beaucoup de mémoire vive pour réaliser la compilation et la liaison. Il ne reste plus ensuite qu'à transférer le programme vers la machine cible qui va l'exécuter immédiatement.

Il existe des logiciels émulateurs qui permettent au programmeur de tester son programme sans disposer de la machine cible. Par définition, l'émulateur ne peut pas émuler à 100 % le comportement de la machine cible dans son vrai environnement. Elle peut suffire à tester les principales fonctions avant de téléverser le programme vers le microcontrôleur. Vous trouverez plusieurs émulateurs AVR sous Linux, parmi lesquels *simavr* et le simulateur GNU-AVR. Je présente les outils logiciels en annexe.

Code source, code objet et images

Les textes qui parlent de compilation, de liaison et de téléversement utilisent des termes tels qu'objet, image, exécutable et code source. Ces termes sont utilisés depuis longtemps, au moins depuis l'époque des cartes perforées, des

mémoires magnétiques et dérouleurs de bande, époque pendant laquelle un ordinateur emplissait une salle à lui tout seul.

Ce que l'on nomme code source est le texte que rédige le programmeur et qu'il fournit au compilateur ou à l'assembleur pour que celui-ci l'analyse et le convertisse. Dans le monde Arduino, le code source est écrit en langage C ou C++, soit dans l'éditeur de l'atelier Arduino, soit dans un autre éditeur.

Ce que produit le compilateur ou l'assembleur est nommé code objet, et c'est vraiment du code machine destiné à un microcontrôleur. Ce sont des valeurs binaires qui incarnent les codes des instructions élémentaires du processeur avec les données sous forme de valeurs littérales, le tout en binaire. C'est du langage machine. Ce résultat de la compilation ou de l'assemblage est stocké dans un fichier qui porte l'extension *.o* ou *.obj*.

N. d. T. : Le terme « objet » utilisé dans la compilation n'a aucun rapport avec les objets de la programmation par objets.

Bien qu'il s'agisse de code machine, le code objet n'est en général pas exécutable tel quel. Dès que le programme comporte plus d'un module, ou qu'il fait appel à une fonction appartenant à une librairie externe, le fichier objet ne contient que des marqueurs de références, qui devront être remplacés plus tard par les véritables adresses en mémoire où se trouve le début du code manquant.

Parfois, ce code externe est fourni en tant que code source, ce qui est le cas de la plupart du code fourni avec Arduino. Dans ce cas, ce code est compilé en même temps que le programme principal (votre croquis). Parfois, le code externe est déjà compilé et se présente sous forme d'une librairie d'objets qui contient plusieurs modules. Les fichiers qui contiennent les librairies d'objets portent en général l'extension de nom *.a*.

Les références non encore résolues dans les différents fichiers objets vont l'être grâce à l'outil appelé lieur ou éditeur de liens. Le produit de cet outil est l'image exécutable. Dans les programmes Arduino, le lieur injecte le code de support d'exécution, la fonction principale en langage C nommée `main()` et l'ensemble des fonctions de la librairie d'exécution fondamentale qui se charge des entrées-sorties et d'autres opérations indispensables à bas niveau.

Si aucune erreur n'est détectée, vous obtenez alors une image exécutable autonome. Cette image est convertie en un fichier qui contient uniquement des valeurs hexadécimales. Ce fichier est transféré vers le microcontrôleur AVR qui

le convertit en binaire, puis le stocke dans sa mémoire flash. Il ne reste plus ensuite qu’à démarrer l’exécution.

Amorceur (*bootloader*)

La mise en place du code exécutable dans la mémoire d'un microcontrôleur est réalisable par différentes techniques, mais la plus simple consiste à apprendre au microcontrôleur à donner un coup de main pour y parvenir. Cette technique consiste à implanter dans le microcontrôleur, généralement lorsqu'on le fabrique, un tout petit logiciel que le contrôleur va exécuter automatiquement à la mise sous tension. Ce logiciel s'appelle un amorceur ou *bootloader*. Même s'il est remplaçable, ce logiciel est installé dans le microcontrôleur à l'usine, et c'est pourquoi il se nomme également microgiciel ou *firmware*.

Origine des microgiciels (*firmware*)

L'expression *firmware* date de l'époque où les ordinateurs embarqués du monde de l'industrie recevaient en usine un logiciel qui ne pouvait plus ensuite être modifié. Parfois, ce microgiciel était directement gravé dans le silicium du circuit. Le programmeur livrait le code machine binaire au fondeur du circuit qui s'en servait lors de la fabrication des puces. Cela s'appelait la programmation par masque. L'intérêt à l'époque était de réduire les coûts unitaires dans les fabrications en grande série, mais c'était loin d'être pratique pour les développeurs. Cela obligeait à disposer d'un lourd matériel de support du développement, ainsi que de versions spéciales du microcontrôleur cible. Parfois, il fallait programmer avec un émulateur de circuit ICE, c'est-à-dire l'équivalent du processeur non encore fabriqué construit à partir de circuits logiques dans une grande boîte avec un câble en nappe qui venait se brancher dans le socle sur lequel viendrait s'insérer le vrai microcontrôleur plus tard. Il fallait prévoir des tests intensifs avant d'autoriser l'injection du logiciel dans le circuit. Cela supposait que l'émulateur de circuit représente un équivalent fiable du véritable microcontrôleur. Avec les premières versions de tous ces outils, c'était loin d'être toujours le cas.

Le processus s'est rapidement assoupli en permettant de stocker le microgiciel sur une puce mémoire qui ne pouvait être écrite qu'une fois, c'est-à-dire un circuit ROM, (*Read-Only Memory*). Rapidement, ce circuit mémoire EPROM a été intégré sur la même puce que le microcontrôleur. Comme dans tout circuit mémoire, le microgiciel était stocké dans la matière sous forme de zéros et de un. On appelait cette technique OTP (*One Time Programmable*) ou

programmable une seule fois. Une fois que le code machine y était inscrit, il y restait pour l'éternité.

Lorsque les quantités à produire ne justifient pas le coût de la première approche basée sur un masque, on pouvait recourir à la technique OTP. Pour de petites séries, on avait pris l'habitude d'installer des batteries de programmateurs d'EPROM pour faire des dizaines ou des centaines de copies à la fois.

Cependant, la technique OTP nécessitait toujours un gros effort de développement initial et des tests soutenus avec des outils spéciaux. Lorsqu'on programmait par OTP pour constater seulement après que le microgiciel était défectueux, ce n'était pas agréable, mais c'était moins catastrophique que de constater l'erreur après avoir fabriqué 10 000 puces avec une programmation par masque. Cela dit, tous les circuits produits par OTP finissaient à la poubelle, parfois à cause d'une virgule oubliée dans le programme source.

Les progrès technologiques ont permis de disposer de nouveaux circuits mémoire pouvant être écrits plusieurs fois, en effaçant le contenu avec de la lumière ultraviolette. Ces puces UVROM étaient faciles à reconnaître parce que le dessus du boîtier comportait une fenêtre en verre ronde, afin de pouvoir illuminer le silicium avec des ultraviolets. Peu après sont apparues les mémoires EEPROM, qui devenaient effaçables électriquement. Dans les deux cas, il fallait disposer d'outils spéciaux pour réaliser l'effacement de la mémoire avant d'y inscrire une nouvelle version du programme, mais lorsque le programme s'avérait défectueux, il suffisait de refaire un cycle d'effacement/rechargement, au lieu de tout jeter à la poubelle.

De nos jours, les microcontrôleurs embarquent une zone de mémoire flash (il reste possible d'acquérir des puces pour la technique OTP). Un des principaux avantages de cette mémoire flash est qu'elle est inscriptible plusieurs fois, même pendant que le contrôleur fonctionne. Évidemment, c'est une mémoire non volatile, ce qui signifie que les données ne sont pas perdues lorsque l'alimentation est coupée. D'une certaine manière, le concept de microgiciel en tant que programme installé une fois pour toutes en usine perd de sa pertinence. Pourtant, tout code exécutable qui est présent dans la mémoire flash du microcontrôleur avant que vous y transfériez votre programme peut être considéré comme un microgiciel.

L’atelier IDEArduino

La fenêtre principale de l’atelier de développement IDE est presque la même dans les trois systèmes d’exploitation (Figure 5.1). La figure montre le contenu de l’éditeur lorsque vous démarrez l’atelier.

N. d. T. : La plus récente version de l’atelier à l’heure où nous écrivons ces lignes est la 1.8.5.

Une nouvelle version de l’atelier IDE apporte toujours quelques corrections, même mineures. Par exemple, la version 1.8.2 a corrigé quelques imperfections au niveau de la fenêtre du moniteur série, et elles sont les bienvenues. Les versions récentes acceptent de travailler avec des cartes non fabriquées par la société Arduino. Il suffit d’entrer l’adresse correspondante pour que l’atelier aille chercher les informations qui décrivent la carte et les intègre à son gestionnaire de cartes.

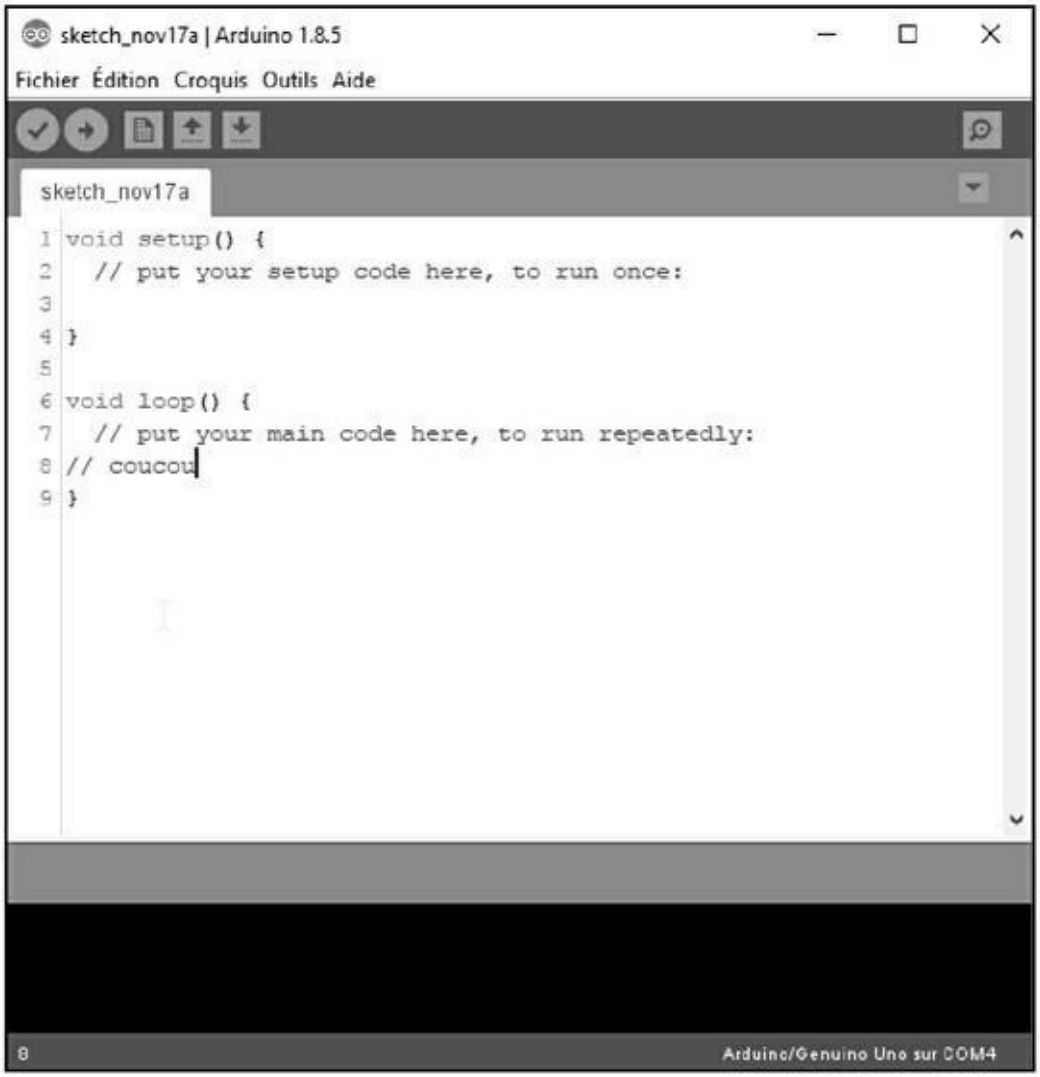


Figure 5.1 : Fenêtre principale de l'atelier IDE Arduino.

Si vous possédez une carte vraiment ancienne, comme une Diecimila ou une Uno avant la R3, vous n'avez pas besoin de la plus récente version de l'atelier et vous n'aurez peut-être pas intérêt à effectuer la mise à jour. Bien sûr, vous ne pourrez dans ce cas pas utiliser certaines des cartes les plus récentes comme la Yun ou la Zero.

Installation de l'atelier IDE

La procédure d'installation de l'atelier et des librairies est soit très simple, soit un peu délicate, selon le système d'exploitation. Sous Linux, l'approche la plus

simple consiste à télécharger puis à laisser installer le paquetage grâce à l'outil de gestion de paquetages. Tous les fichiers nécessaires seront rapatriés.

Le seul inconvénient est que le paquetage disponible ne correspond pas toujours à la plus récente version. Le site officiel Arduino (`arduino.cc`) propose toujours la version la plus récente, mais il n'est pas simple de réunir tous les éléments pour la version sous Linux. En revanche, les versions Windows et macOS sont beaucoup plus simples à mettre en place.

Une fois l'installation terminée, il est conseillé de faire un tour d'horizon de ce qui a été mis en place. Si vous accédez au sous-répertoire dans lequel a été installé l'atelier, vous trouverez les fichiers de code source d'un grand nombre d'exemples, ainsi que celui des librairies essentielles, les fichiers binaires de plusieurs variantes de l'amorceur ainsi qu'une documentation sous forme d'une série de pages HTML que vous pouvez visionner avec un navigateur Web, ou au moyen du menu **Aide** de l'atelier.

Tous les téléchargements sont disponibles depuis le site officiel. Il suffit d'être attentif à bien sélectionner la variante appropriée à votre système.

N. d. T. : Sous Windows, le bouton principal pour télécharger l'atelier mène à l'entrepôt de logiciels propriétaires de Microsoft. Si vous ne désirez pas créer un compte (car vous y êtes obligé), il faut utiliser un moyen détourné. Cherchez sur le site Arduino pour accéder au téléchargement direct du fichier archive.

Sous Windows

L'installation de l'atelier sous Windows est très simple : une fois que vous avez récupéré le fichier exécutable d'installation, tout est automatique. Si l'installateur détecte une version antérieure de l'atelier, il va vous en avertir. Inutile d'interrompre l'installation, car l'outil sait désinstaller la version précédente, en prenant soin de ne pas supprimer les fichiers de code source que vous avez créés, c'est-à-dire vos croquis. Tous les fichiers exécutables et toutes les librairies Arduino sont placés par défaut dans le sous-répertoire *Programmes\Arduino*, et tous les fichiers de code source des exemples aussi. De plus, l'outil crée un sous-répertoire portant le nom *Arduino* dans votre répertoire de compte utilisateur. C'est à cet endroit que seront stockés tous les croquis que vous allez créer et toutes les librairies que vous allez ajouter.

Sous Linux

Toutes les distributions Linux donnent accès à l'atelier Arduino sous forme d'un paquetage au format *.deb*, *.rpm*, *.ymp* ou autre. C'est la solution à privilégier pour mettre en place l'atelier et les librairies correctement sous Linux. Le seul inconvénient est que les volontaires qui préparent les paquetages des différentes distributions ne les ont pas nécessairement mis à jour juste après l'arrivée d'une nouvelle version, telle que proposée sur le site officiel `arduino.cc`.

L'outil de gestion des paquetages de votre distribution constitue la technique la plus confortable pour installer l'atelier, mais vous pouvez aussi télécharger directement un paquetage depuis le site Arduino et en lancer l'installation. D'ailleurs, c'est la seule option pour certaines distributions moins répandues. Vous trouverez de nombreux détails à ce sujet sur le site officiel Arduino (`playground.arduino.cc/Learning/Linux`).

Dans les distributions de la famille Ubuntu, comme Kubuntu et Edubuntu, vous utilisez l'outil *apt-get*. Sous Open Suse Linux, vous utilisez l'outil *yast*, tandis que d'autres distributions travaillent avec le format de Red Hat, et l'outil dédié, *RPM*. L'ensemble des exécutables et des librairies sont placés ici :

```
/usr/share/Arduino
```

Les fichiers créés par le programmeur et les librairies qu'il installe sont placés dans un sous-répertoire du répertoire personnel *home*, sous le nom *Sketchbook*.

Sous macOS

La version Apple de l'atelier et des librairies est proposée sous forme d'un fichier archive compressé ZIP. Le site officiel Arduino précise que l'atelier est prévu pour au minimum la version 10.7 (Lion) de macOS. Après avoir ouvert l'archive, il suffit de copier le fichier *Arduino.app* dans le dossier *Applications* (ou un autre si vous savez comment faire). Le site officiel Arduino comporte une page dédiée à macOS.

Configuration de l'atelier Arduino

Les paramètres de fonctionnement de l'atelier Arduino sont disponibles dans une boîte de dialogue de préférences. Vous y accédez par la commande **Fichier/Préférences** (dans le menu Arduino sous macOS). C'est dans cette boîte

que vous pouvez choisir un autre répertoire pour stocker vos croquis, choisir un éditeur différent de celui de l'atelier et régler différentes options. La Figure 5.2 montre la boîte des préférences d'une ancienne version de l'atelier sous Linux.

Toutes les préférences sont stockées dans un fichier qui porte le nom *preferences.txt*. Il est placé dans votre sous-répertoire de travail personnel. Ce fichier contient bien plus d'options que vous pouvez en voir dans la Figure 5.2. Il est absolument déconseillé de modifier le contenu du fichier tant que l'atelier Arduino est en cours d'exécution.

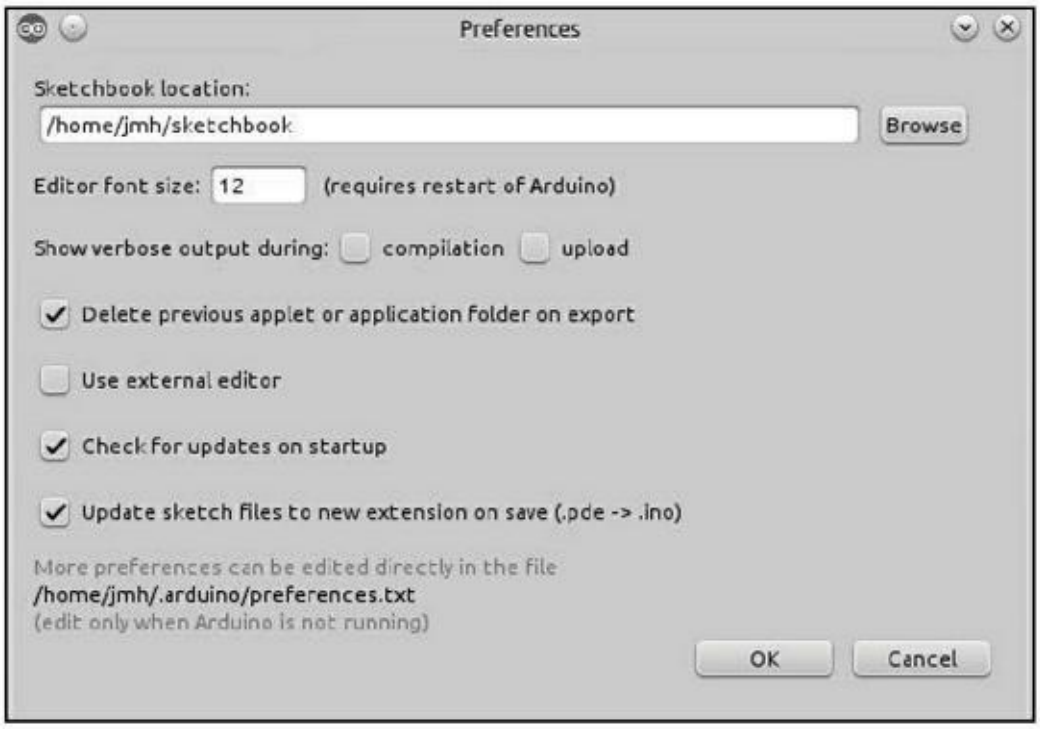


Figure 5.2 : Boîte de dialogue des préférences de l'atelier d'une ancienne version.

La Figure 5.3 montre la même boîte des préférences pour les versions actuelles de l'atelier. Elle donne accès à un bien plus grand nombre d'options, mais ce n'est toujours rien d'autre qu'une interface graphique pour modifier le contenu du fichier des préférences.

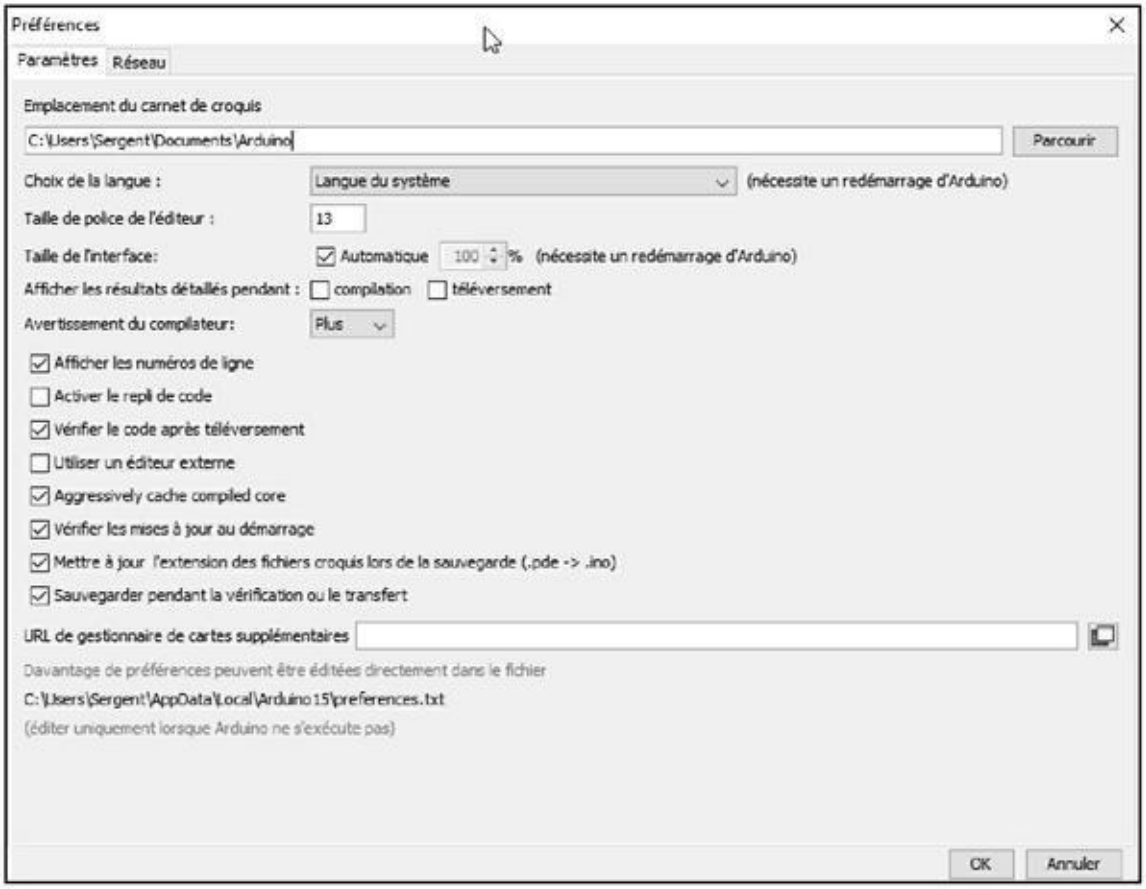


Figure 5.3 : Boîte de dialogue des préférences de l'atelier actuel.

Le fichier des préférences est structuré au format KVP ASCII, c'est-à-dire sous forme de couples clé/valeur (*Key Value Pair*). Parmi les options qu'il définit, citons le format d'affichage initial de la fenêtre de l'atelier, les paramètres de l'interface série (débit, taille unitaire des données, *etc.*) et le nom du navigateur à utiliser pour afficher les fichiers d'aide au format HTML.

Le fichier contient également le nom du dernier croquis sur lequel vous avez travaillé et les dimensions de la fenêtre de l'atelier quand vous avez quitté le programme. C'est par exemple pour cette raison qu'il ne faut pas éditer manuellement les préférences quand l'atelier est actif, car il doit pouvoir y écrire des données à tout moment.

Dans l'ancienne version de l'atelier, il fallait intervenir dans le fichier directement pour modifier des options telles que la longueur d'un pas de tabulation (deux espaces par défaut, mais je préfère quatre), la taille et le nom de la police de l'éditeur (`Courier 12` au départ, mais je préfère la réduire à 11 pour mieux voir les longues lignes) et le remplacement des pas de tabulation

par des espaces. Même dans les dernières versions de la boîte des préférences, certaines options ne sont pas accessibles, et vous devrez modifier le fichier *preferences.txt*.

La cross-compilation avec l’atelier Arduino

À la différence d’autres microcontrôleurs, une carte Arduino ne se résume pas à un circuit imprimé sur lequel a été soudée une puce de microcontrôleur AVR. C’est un véritable environnement qui réunit l’atelier Arduino, le microgiciel, le code de support d’exécution, les librairies essentielles qui ont été créées par Arduino et bien sûr les programmes que vous ou d’autres programmeurs ont créés. L’atelier Arduino constitue une approche confortable pour créer puis télécharger un logiciel dans le circuit AVR. En utilisant l’atelier, vous vous épargnez tous les détails complexes qui régissent l’enchaînement de la compilation, de la liaison et du transfert vers la cible.

Le langage de programmation avec lequel vous écrivez vos programmes dans l’atelier Arduino est du C++, mais vous pouvez également écrire en langage C. En effet, la chaîne d’outils avr-gcc reconnaît les deux. Dans ce livre, lorsque je désigne les deux langages, j’écris C/C++ et je précise C ou C++ lorsque nécessaire.

Lorsqu’un programme est constitué de plusieurs fichiers de code source, chaque fichier est chargé dans une page différente de l’atelier, accessible par son onglet (tab). Lorsque vous lancez la compilation, l’atelier va exploiter le contenu de chacune des pages, en créant éventuellement un fichier intermédiaire *.obj* pour chacune.

Dans l’atelier, les outils dont l’exécution est enchaînée automatiquement pour produire le code exécutable forment une chaîne. L’atelier utilise la chaîne d’outils avr-gcc. Les librairies essentielles qui sont fournies par Arduino définissent une foule de fonctions pour gérer des opérations élémentaires telles que des délais, des émissions de données série, et bien d’autres capacités (les librairies sont décrites dans le Chapitre 7).

Certains programmeurs prétendent que le langage de programmation Arduino est de l’*arduinien*, pour mettre en valeur le fait qu’il s’agirait d’un dialecte des langages C et C++. C’est inexact. Arduino utilise du vrai langage C ou C++, avec quelques limitations par rapport au langage C++ officiel. Je rappelle que l’atelier n’est qu’une interface graphique pour contrôler à la souris la chaîne d’outils de production avr-gcc. (Je présente cette chaîne d’outils en détail dans le prochain chapitre.)

Pour être exact, les deux opérateurs du C++ portant le nom `new` et `delete` ne sont pas pris en charge par l'outil avr-libc. Cependant, l'équipe Arduino a fourni une solution. Les définitions de ces deux opérateurs sont disponibles dans les sous-répertoires suivants.

Sous Linux, ils sont dans :

```
/usr/share/arduino/hardware/arduino/cores/arduino/new.h
```

Sous Windows, ils sont dans :

```
C:\Program
Files\Arduino\hardware\arduino\avr\cores\arduino
```

Enfin, sous macOS, ils sont dans :

```
/Applications/Arduino.app/Contents/Java/hardware/arduino/a
```

Instanciation d'une classe avec new

Dans le monde des microcontrôleurs des systèmes embarqués, il est généralement déconseillé de réserver de la mémoire pendant l'exécution dynamiquement. En effet, cette réservation dynamique a généralement lieu dans la mémoire statique SRAM. La plupart des microcontrôleurs, y compris la famille AVR, n'ont que très peu d'espace mémoire SRAM. L'outil avr-libc reconnaît la fonction de réservation `malloc()`, mais pas l'opérateur `new`. Les programmeurs de l'Arduino ont jugé que ce serait une bonne idée d'ajouter cet opérateur.

Lorsque vous créez un objet à partir d'une classe dans le C++ AVR, le code correspondant n'est pas stocké dans l'espace mémoire SRAM. Les microprocesseurs AVR sont conformes à l'architecture de processeurs dite Harvard ; ils n'exécutent jamais le code s'il est dans la mémoire SRAM, seulement s'il est stocké dans la mémoire flash. L'astuce trouvée par les gens d'Arduino consiste à créer l'objet en mémoire flash lors de la compilation puis à faire renvoyer un pointeur sur cet objet par l'opérateur `new`.

Comme le montre le Listing 5.1, on peut utiliser l'opérateur `new` depuis la version 1.0.5 de l'atelier en plaçant la déclaration dans le fichier source contenant les variables globales.

Listing 5.1 : Création d'un objet global avec l'opérateur new.

```
// global_objs.cpp
#include "Arduino.h"
#include <LiquidCrystal.h>
#include <DDS.h>

LiquidCrystal *lcd = new LiquidCrystal(LCD_RS,
LCD_E, LCD4, LCD5,LCD6,
  LCD7);
DDS *ddsdev = new DDS(DDS_OSC, DDS_DATA, DDS_FQ_UP,
DDS_W_CLK, DDS_RESET);
```

Les deux nouveaux objets *`lcd`* et *`ddsdev`* doivent être déclarés comme externes dans le fichier d'en-tête associé (Listing 5.2). Les objets deviennent ainsi accessibles depuis tout autre module de code source qui fait référence au fichier d'en-tête global.

Listing 5.2 : Déclaration de deux objets globaux avec extern.

```
// global_objs.h
#include <LiquidCrystal.h>
#include <DDS.h>
extern LiquidCrystal *lcd;
extern DDS           *ddsdev
```

L'autre technique pour atteindre le même objectif consiste à travailler avec des pointeurs, comme le montre le Listing 5.3. Le même fichier d'en-tête peut être utilisé.

Listing 5.3 : Affectation de deux objets à des pointeurs globaux.

```
// global_objs.cpp
#include "Arduino.h"
#include <LiquidCrystal.h>
#include <DDS.h>
```

```
LiquidCrystal lcdobj(LCD_RS, LCD_E, LCD4, LCD5,
LCD6, LCD7);
DDS           ddsobj(DDS_OSC, DDS_DATA, DDS_FQ_UP,
DDS_W_CLK, DDS_RESET);

LiquidCrystal *lcd    = &lcdobj;
DDS           *ddsdev = &ddsobj;
```

Il n'est pas inintéressant de constater que la taille du fichier exécutable résultant n'est pas la même selon la méthode choisie pour rendre les objets accessibles. Avec l'opérateur `new`, le fichier binaire sera plus volumineux de 662 octets qu'avec les pointeurs. Ce n'est pas beaucoup, mais quand vous n'avez que 32 ko de mémoire flash, chaque octet compte.

Note : Les extraits ci-dessus sont tirés du code source du générateur de signal du Chapitre 11.

Les patrons ou *templates* du langage C++ (qui permettent à une classe ou une fonction de s'adapter à plusieurs types de données) ne sont pas reconnus par Arduino, tout comme les exceptions. En revanche, les classes sont bien supportées, y compris leurs constructeurs et destructeurs. Il y a aussi quelques limitations au niveau du langage C pur. Vous pouvez utiliser les fonctions de gestion mémoire `malloc()` et `free()`, mais avec précaution. N'oubliez pas que gérer dynamiquement la mémoire dans un microcontrôleur n'est pas conseillé, d'autant que cela peut provoquer l'apparition de problèmes difficiles à éradiquer. Il vous faut une raison vraiment forte pour exploiter l'espace mémoire de façon dynamique avec un microcontrôleur qui en possède si peu.

La raison la plus plausible qui a poussé certains à prétendre que le langage Arduino n'était pas du C/C++ est peut-être la façon dont l'environnement Arduino prédéfinit des macros et des fonctions pour simplifier l'accès aux fonctions d'entrées-sorties du microcontrôleur. Les fichiers sources Arduino (les croquis) portent une extension de nom de fichier originale : *.ino*. Leur structure peut ressembler à une version édulcorée du langage C, mais c'est volontaire. N'oublions pas l'audience initiale d'Arduino : des gens qui n'ont pas ou peu d'expérience en programmation. Toute la complexité du C et du C++ est toujours disponible, si vous en avez besoin. D'ailleurs, je vais décrire dans le chapitre suivant le processus de création de code exécutable sur Arduino ou tout autre microcontrôleur AVR sans utiliser l'atelier, uniquement avec la chaîne d'outils avr-gcc, un éditeur de texte et des fichiers de production *Makefiles*.

Un microcontrôleur n’est pas un microprocesseur !

Un microcontrôleur typique est un circuit autonome, qui n’embarque qu’une faible quantité de mémoire pour le programme et pour les données. En revanche, il dispose de façon innée d’un grand nombre de fonctions de gestion des entrées-sorties. Les microcontrôleurs ont en général des domaines d’application précis : détection de frappes de touches, contrôle de moteurs, surveillance d’événements extérieurs (températures, lumière, son) ou même ajout d’intelligence à une installation artistique, et éventuellement toutes ces opérations en même temps. À l’exception éventuelle de la carte haut de gamme ATMega2560, qui est capable d’accéder à de la mémoire externe, un microcontrôleur AVR ne pourra jamais servir à faire fonctionner un traitement de texte ou un jeu vidéo complexe. Cela dit, des passionnés très futés ont réussi à écrire du code très compact et efficace pour faire faire à un microcontrôleur AVR des opérations complexes et très intéressantes.

L’image exécutable Arduino

L’image exécutable du programme qui est produite par l’atelier Arduino comporte trois parties principales : le résultat compilé de votre croquis, les librairies qui permettent d’accéder aux fonctions du microcontrôleur et le code de support d’exécution qui contient la fonction principale `main()` et une fonction de boucle qu’elle va appeler (et qui à son tour va appeler le début de votre programme). Lorsque vous démarrez l’atelier Arduino (revoyez la Figure 5.1 si nécessaire), l’atelier charge d’office un modèle initial de code source qui contient le minimum pour que ce code source soit considéré comme valide.

Les outils qui permettent de produire le code exécutable se rangent dans deux catégories : les outils de développement sur la machine hôte avec les fichiers source à compiler, d’une part, et le code binaire exécutable sur la cible, d’autre part. La Figure 5.4 montre les différents composants qui sont en jeu pour produire un logiciel fonctionnant sur un microcontrôleur AVR.

La production d’un fichier exécutable (processus appelé également *build*) avec l’atelier Arduino comporte cinq étapes successives.

1) Préparation du code source

Le code source de votre croquis est légèrement modifié à la volée par l'atelier avant la compilation en y ajoutant une directive `#define` pour faire prendre en compte un fichier portant le nom *WProgram.h* (le nom était *Arduino.h* dans les anciennes versions de l'atelier). Les contenus des pages d'édition correspondant à des fichiers qui n'ont pas d'extension de nom sont fusionnés avec le croquis principal, afin d'obtenir un grand fichier de code source. En revanche, les contenus des pages qui existent en tant que fichiers avec une extension de nom *.c* ou *.cpp* sont compilés séparément.

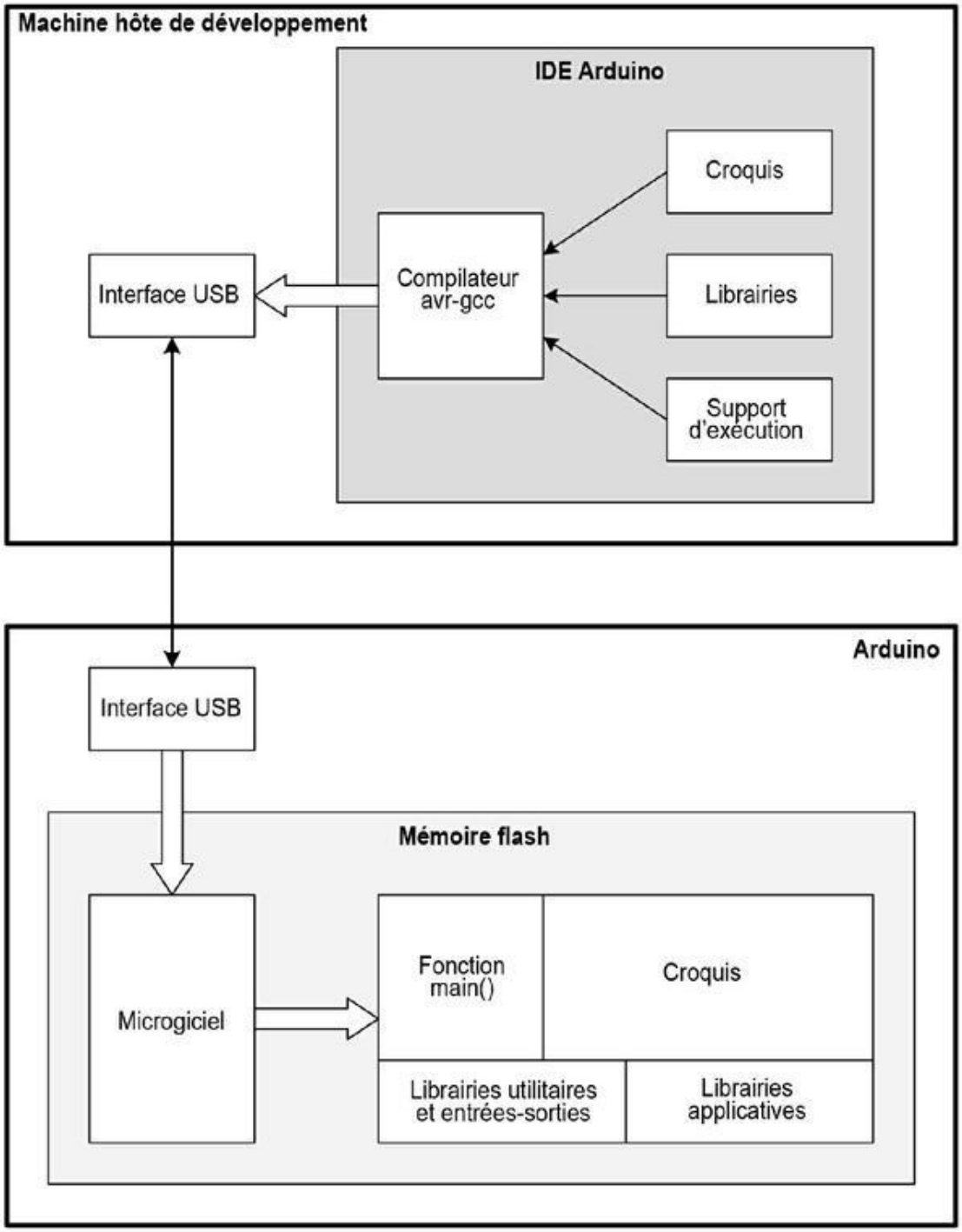


Figure 5.4 : Organisation des logiciels de l'atelier Arduino.

L'atelier Arduino recherche ensuite toutes les définitions de fonctions trouvées dans le code source, pour créer des prototypes, sauf pour les deux fonctions obligatoires `setup()` et `loop()`. Ces prototypes sont insérés au début du fichier croquis principal, juste après les commentaires et directives de

préprocesseur `#include` et `#define`. En programmation C ou C++ classique, ces informations sont réunies dans un fichier d'en-tête *.h* qui est référencé par le fichier principal. Dans l'approche Arduino, les déclarations sont insérées dynamiquement dans votre croquis. La dernière opération de préparation consiste à ajouter le fichier standard qui définit la fonction principale `main()`.

2) Compilation

La production des fichiers binaires appelés fichiers objets est réalisée par la chaîne d'outils avr-gcc. Les fichiers résultants ne peuvent pas être exécutés directement, puisque les références externes de l'un vers l'autre ne sont pas encore résolues, cela étant le travail du lieur (*linker*). Le compilateur avr-gcc est une version spécialement conçue pour les microcontrôleurs AVR du compilateur open source GCC (*GNU C Compiler*). Les contraintes d'espace mémoire sont une des raisons pour lesquelles toutes les possibilités du langage C++ ne sont pas disponibles, mais celle du langage C le sont.

Le compilateur reconnaît un grand nombre d'options de ligne de commande. L'atelier Arduino ajoute les options appropriées, notamment pour préciser où se trouvent les fichiers d'en-tête standard, pour ajouter des options d'optimisation, de définition de la cible exacte et d'étendue des messages d'erreur et d'avertissement.

3) Liaison

L'étape de liaison consiste à mettre en correspondance les appels de fonctions dans les fichiers objets et les fonctions qui se trouvent dans les modules des librairies. Le compilateur travaille un fichier de code source à la fois ; il ne peut donc pas savoir où se trouvent les cibles des éléments externes qui sont mentionnés (surtout quand les fichiers objets ne sont pas encore compilés). Le lieur trouve ces cibles et raccorde les différents modules pour obtenir un fichier binaire exécutable unique, qui contient le code de tous les éléments qui doivent fonctionner ensemble, ainsi que les données externes.

4) Conversion

Le fichier binaire qui est produit par le lieur doit alors être converti vers un format spécial, le format Intel Hex. C'est le format que va accepter le microgiciel amorceur sur le microcontrôleur. Cette conversion est réalisée par l'outil avr-objcopy que je décris dans le chapitre suivant.

5) Téléversement

La dernière étape consiste à transférer les données binaires de l'exécutable vers la carte Arduino, en général par une liaison USB. L'opération est gérée par le microgiciel amorceur du microcontrôleur avec l'aide d'un outil nommé *avrdude*. Notez qu'il est possible de transférer du code exécutable dans le microcontrôleur directement en utilisant l'interface ICSP. Vous ne procéderez ainsi que lorsqu'il n'y aura pas d'amorceur. L'atelier Arduino supporte ce mode de téléversement.

Onglets et fichiers multiples

Vous pouvez, et c'est souvent conseillé, subdiviser un code source de grande longueur en plusieurs modules, chacun étant associé à son propre fichier d'en-tête. Lorsque vous demandez le chargement d'un fichier de croquis dans l'atelier, il va automatiquement chercher dans le même sous-répertoire s'il y a d'autres fichiers source. Ces fichiers peuvent ne pas comporter d'extension de nom, ou bien porter l'une des extensions *.c*, *.cpp* ou *.h*. Chacun des fichiers *.c* ou *.cpp* sera compilé indépendamment pour produire un fichier objet qui sera ensuite traité par le lieur. En revanche, les contenus des fichiers dont le nom ne comporte pas d'extension vont être copiés de leur onglet vers le croquis principal. Enfin, pour faire référence à un fichier d'en-tête .h qui possède sa propre page, il faut le référencer par une directive `#include`, en délimitant le nom du fichier par des guillemets droits et non par une paire de chevrons < et >.

Lorsque vos fichiers secondaires possèdent eux-mêmes des fichiers d'en-tête, tous doivent être référencés dans le croquis principal, même si vous ne faites aucune référence directe au contenu de ces fichiers secondaires. Il en va de même pour utiliser des modules de librairies externes. Lorsqu'un fichier secondaire fait référence à une classe d'une librairie, mais que le croquis principal n'y fait pas référence, il faut malgré tout ajouter une directive `#include` deux fois : dans le fichier concerné et dans le croquis principal. C'est une particularité de l'atelier Arduino.

J’ai personnellement pris l’habitude de réunir toutes les variables globales et définitions de classes dans un fichier source distinct. Tous les autres modules peuvent ainsi accéder à ces éléments ainsi centralisés. Une autre conséquence est que le croquis principal est allégé, puisqu’il peut même se résumer aux deux fonctions obligatoires `setup()` et `loop()`. Dans le Chapitre 11, vous trouverez un exemple complet de programmation modulaire pour le projet de générateur de signal (le code source est disponible sur le site de l’éditeur et sur GitHub). Le site officiel Arduino propose en outre une page en anglais décrivant la création de croquis multifichier.

Architecture d’un programme Arduino

Tout programme Arduino, même le plus petit, doit contenir au moins deux fonctions : `setup()` et `loop()`. La première n’est appelée qu’une fois au démarrage du programme. L’autre est une boucle qui est exécutée de façon répétée jusqu’à la mise hors tension de la carte ou réinitialisation par le bouton Reset. C’est la fonction principale `main()`, insérée automatiquement, qui appelle de façon répétée votre fonction `loop()`. Votre croquis aura bien sûr d’autres fonctions. Celui du thermostat du Chapitre 12 définit tout un ensemble de fonctions en plus des deux fonctions obligatoires. Le contenu de la fonction principale `main()` est décrit dans la section suivante.

En général, un croquis contient au début des instructions et des directives pour préciser les fichiers dont il dépend, pour déclarer des constantes pour les broches d’entrées-sorties et des variables globales. La structure générale d’un croquis est visible dans le croquis d’exemple quelques pages plus loin. Après cet exemple, je donne quelques détails sur les constantes et les variables globales.

Les programmes pour microcontrôleur sont quasiment toujours structurés autour d’une boucle principale qui s’exécute de façon répétée. On appelle également cela une boucle d’événements. Dans un microcontrôleur, il n’est pas prévu de demander le chargement d’un exécutable depuis un disque. Le programme qui est implanté est conçu pour réaliser une série de tâches sans discontinuer. Il est donc logique d’appliquer un principe de boucle éternelle. Et puisque l’exécutable est stocké dans une mémoire non volatile, la mémoire flash, il va pouvoir redémarrer après une remise sous tension de la carte ou un appui sur le bouton Reset.

Pour illustrer cette approche, vous pouvez voir dans la Figure 5.5 comment la fonction principale `main()`, qui est ajoutée par l'atelier, appelle successivement vos fonctions `setup()` et `loop()`.

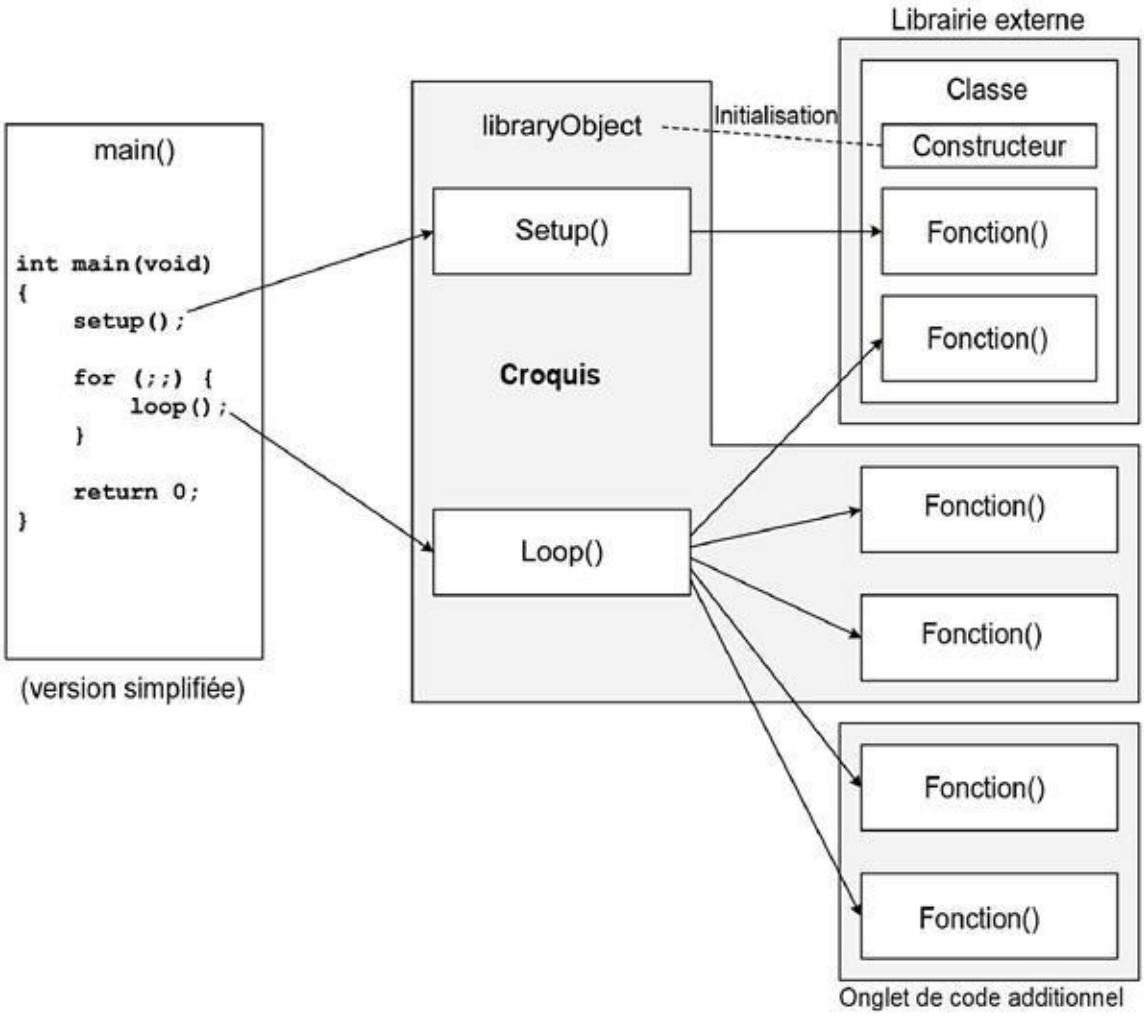


Figure 5.5 : Structure d'un programme Arduino.

Dans cette figure, un objet est instancié à partir d'une classe définie dans une librairie externe, opération réalisée en dehors du code des deux fonctions obligatoires `setup()` et `loop()`. Cet objet devient de ce fait disponible pour toutes les fonctions du croquis. Le croquis principal peut évidemment définir d'autres fonctions en plus des deux obligatoires, et d'autres fichiers complémentaires peuvent à leur tour définir des fonctions. Il suffit que ces fichiers secondaires soient référencés par des directives dans le croquis principal. L'atelier Arduino saura les prendre en compte.

Il ne faut pas confondre les contenus des pages de modules secondaires avec des librairies. Ce ne sont que des portions de code source renvoyées dans des fichiers distincts. Un peu comme une librairie, le code d'un fichier secondaire peut être utilisé dans un autre projet, mais il faut déclarer la page de ce fichier lors de la création du croquis. Il ne suffit pas d'ajouter une directive `#include` dans le croquis principal. Pour ouvrir proprement un onglet pour un nouveau fichier secondaire, il faut utiliser dans l'atelier Arduino la commande **Croquis/Ajouter un fichier**. Pour autant, le fichier de code source qui va être hébergé dans cette page ne doit pas nécessairement être stocké dans le même sous-répertoire que le croquis principal.

Les éléments utilisés qui appartiennent à des librairies externes sont intégrés au fichier exécutable final une fois que le code du croquis et des fichiers secondaires a été compilé. L'atelier ajoute les options de liaison correctes en se basant sur les directives `#include` ou `#define` qu'il trouve tout au début du croquis. Je donne quelques pages plus loin quelques conseils pour utiliser des librairies dans les croquis.

Support d'exécution par la fonction main()

À chaque système d'exploitation correspond un jeu de fonctions qui assurent le support d'exécution de votre programme. Dans un système utilisant la chaîne de compilation GCC, ces librairies correspondent au fichier *libgcc.a* ou *libgcc-s.so.1*. Sous Windows, les librairies portent généralement le nom MSVCRT.DLL pour le langage C et MSVCPP.DLL pour le langage C++. Pour les microcontrôleurs de la famille AVR, la librairie d'exécution porte le nom avr-libc ; je la décris en détail dans le prochain chapitre. La librairie d'exécution réunit toutes les fonctions élémentaires : opérations arithmétiques principales, fonctions d'entrées-sorties de bas niveau, chronomètres et timer système, support des fonctions d'affichage `printf()`, *etc*.

Dans le monde Arduino, il existe un module de support d'exécution complémentaire, car votre croquis n'est pas entièrement complet : il ne contient pas la fonction principale `main()` et il n'a aucune injonction interne pour le forcer à se répéter sans cesse. C'est le module de support d'exécution Arduino qui définit cette fonction « racine », parmi quelques autres fonctions de configuration initiale. Le Listing 5.4 montre que la définition de cette fonction est très simple, ce qui est normal dans un environnement de microcontrôleur.

Listing 5.4 : Code source de la fonction principale main() pour Arduino.

```
int main(void)
{
  init();
  initVariant();
  #ifdefined(USBCON)
    USBDevice.attach();
  #endif

  setup();

  for (;;) {
    loop();
    if (serialEventRun)   serialEventRun();
  }
  return 0;
}
```

En fonction du type exact de carte sélectionnée dans les menus de l'atelier, les appels aux fonctions `init()` et `initVariant()` (et si applicable, `USBDevice.attach()`) sont adaptés. Ces appels incarnent les éléments principaux du code de support d'exécution approprié à chaque type de carte. Dans un système d'exploitation en temps réel de type RTOS (*Real-Time Operating System*), ces fonctions feraient partie du paquetage de support de cartes BSP normalement fourni par l'éditeur du système, ou parfois créé par un programmeur. Entrons dans les détails de ces fonctions.

init()

Le code source de cette fonction est disponible à cet endroit :

```
Arduino/hardware/arduino/avr/cores/arduino/wiring.c
```

Cette fonction initialise plusieurs composants périphériques à l'intérieur du microcontrôleur : le calibrage des timers et des compteurs, celui des convertisseurs analogiques-numériques et les modes de génération des impulsions PWM. La liste exacte des actions de préparation dépend des parties fonctionnelles du microcontrôleur qui sont utilisées par la carte Arduino.

initVariant()

Cet appel de fonction offre un point d'ancrage sous forme d'une déclaration de fonction faible. Le but est de permettre d'ajouter des actions d'initialisation pendant l'exécution pour des éléments matériels qui ne sont pas préparés par l'environnement de développement Arduino standard.

USBDevice.attach()

Il s'agit d'un appel à la méthode `attach()` de la classe `USBDevice` (définie dans le fichier *Arduino/ hardware/arduino/avr/cores/arduino/USBCore.cpp*). Cette classe permet de communiquer avec l'interface USB.

La plupart des gens n'auront pas besoin de connaître les détails de ces fonctions d'initialisation. En revanche, si vous voulez vraiment savoir comment fonctionne l'atelier Arduino, il n'est pas inutile d'aller lire le code source des fonctions de support d'exécution. La dernière section de ce chapitre explique comment trouver ce code source.

Vous savez maintenant que les deux fonctions `setup()` et `loop()` sont définies par vous dans votre croquis principal. La première sert à préciser quelles broches servent aux entrées et aux sorties, à définir la valeur initiale de certaines variables globales, l'état de fonctionnement de certains composants du microcontrôleur, et autres opérations à ne réaliser qu'une fois. Si vous n'avez aucune fonction de préparation à réaliser, la fonction `setup()` peut être entièrement vide, mais elle doit exister, car sinon le lieur va générer une erreur.

C'est bien sûr dans la fonction de boucle `loop()` que va se dérouler l'activité principale du programme. Le code source de la fonction `main()` montre que votre fonction `loop()` est appelée sans arrêt. Une interruption peut être déclenchée et détectée pendant l'exécution de la fonction. Certaines applications ajoutent une interruption de timer ou un délai pour que la durée d'exécution de chaque tour de boucle ait une valeur à peu près prévisible au lieu de fonctionner le plus vite possible.

Un exemple de croquis

Pour qu'un croquis de code source Arduino soit considéré comme valable, il doit contenir au minimum les deux fonctions `setup()` et `loop()`. Rappelons que la fonction `main()` est injectée automatiquement par l'atelier ; nous n'avons pas à nous en soucier, du moins dans les programmes simples.

L'exemple du Listing 5.5 représente un système d'alarme simplifié qui surveille une série de fenêtres et une série de portes afin de déclencher une alarme dès qu'une des deux entrées bascule de l'état Bas vers l'état Haut.

Listing 5.5 : Code source d'un système d'alarme simplifié.

```
// CONSTANTES
// Définition des broches à utiliser
#define LOOP1   2      // Boucle de surveillance 1
#define LOOP2   3      // Boucle de surveillance 2
#define ARM     6      // Armement alarme
#define RESET   7      // RAZ alarme
#define ALERT   12     // Alarme audio
#define LED     13     // Diode LED de la carte

#define SLEEPTM 250    // Délai de boucle
#define SLOWFLASH  10  // Vitesse clignotement désarmé
#define FASTFLASH  2   // Vitesse clignotement armé

// VARIABLES GLOBALES
// Drapeaux d'état
bool arm_state;        // T = armé, F = désarmé
bool alarmed;          // T = alarme déclenchée, F = rien
à signaler
bool led_state;        // Etat diode LED
bool loop1state;       // Etat boucle 1
bool loop2state;       // Etat boucle 2
bool in_alarm;         // T = alarme active, F = calme

int led_cnt;           // Compteur cycles de flash
int flash_rate;        // Diviseur du compteur

// Préparation des broches numériques avec pinMode()
void setup()
{
  // Prépare les broches 2, 3, 6 et 7 en entrée
```

```
  pinMode(LOOP1, INPUT);
  pinMode(LOOP2, INPUT);
  pinMode(ARM, INPUT);
  pinMode(RESET, INPUT);

  // Prépare les broches 12 et 13 en sortie
  pinMode(ALERT, OUTPUT);
  pinMode(LED, OUTPUT);

  // Initialise les drapeaux d'état
  arm_state  = false;
  alarmed    = false;
  led_state  = false;
  loop1state = true;
  loop2state = true;
  in_alarm   = false;

  // Définit les conditions initiales de la LED
  led_cnt = 0;
  flash_rate = SLOWFLASH;
  digitalWrite(LED, LOW);
}

void loop()
{
  // Lit état interrupteur du bras
  arm_state = digitalRead(ARM);

  // Si Reset est vrai (état Bas), effacer l'état d'alarme
  if (!digitalRead(RESET)) {
    in_alarm = false;
  }
  // Si pas armé, reboucler après une pause
  if (!arm_state) {
    flash_rate =  SLOWFLASH;
    in_alarm = false;
  }
  else {
    flash_rate =  FASTFLASH;
  }

  // Tester les boucles de surveillance
```

```
  loop1state = digitalRead(LOOP1);
  loop2state = digitalRead(LOOP2);
  // Bascule en état d'alarme seulement si armé
  if (arm_state) {
    if ((loop1state) ¦¦ (loop2state)) {
      in_alarm = true;
   }
  }

  if (in_alarm) {
    digitalWrite(ALERT, HIGH);
  }
  else {
    digitalWrite(ALERT, LOW);
  }

  led_cnt++;
  if (!(led_cnt % flash_rate)) {
    led_cnt = 0;              // Reset du compteur de
flash
    led_state = !led_state;   // Inversion état LED
    digitalWrite(LED, led_state);
  }
  delay(SLEEPTM);
}
```

La Figure 5.6 montre quel genre de montage Arduino peut être animé par ce programme : deux entrées numériques sont reliées chacune à une série d'interrupteurs pour les fenêtres et portes. Cette façon de procéder est robuste : il suffit qu'un des contacts soit ouvert, ou que quelqu'un coupe un des fils pour que la ligne passe à l'état Haut grâce à la résistance de tirage, ce qui déclenche l'alarme.

Les interrupteurs utilisés peuvent être des contacts électriques ou des relais magnétiques Reed. Le buzzer d'alarme peut être remplacé par un relais pour contrôler une puissante sirène, ou même un module GSM pour appeler un numéro. Ce schéma très simple est une bonne base de départ pour aboutir à un système d'alarme de niveau professionnel. Le code source de l'exemple est disponible sur le site de l'éditeur, dans la page du livre. (Le Chapitre 9 présente des capteurs compatibles Arduino.)

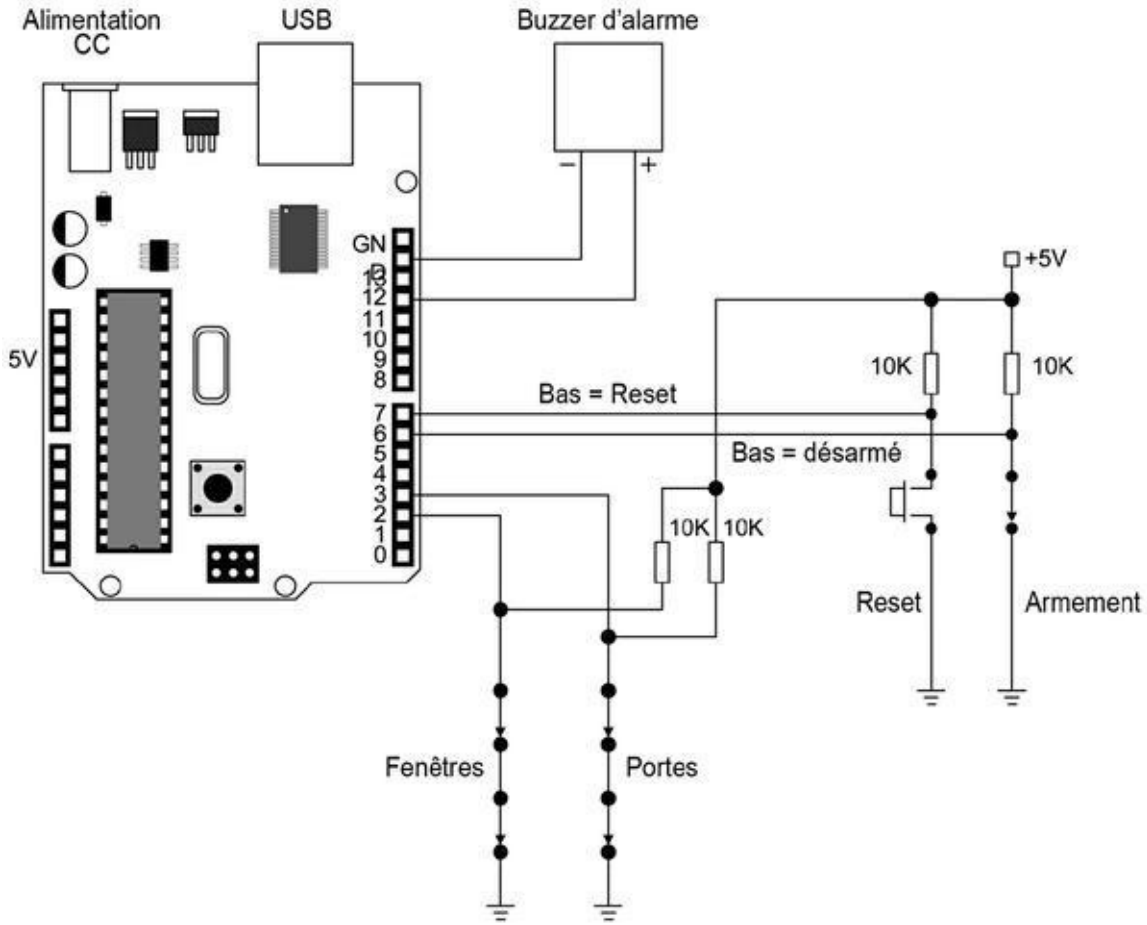


Figure 5.6 : Schéma d'une alarme Arduino.

Déclaration de constantes

Vous disposez de deux techniques pour définir les constantes, par exemple pour les numéros des broches à utiliser, pour les délais et pour d'autres valeurs qui ne changent pas tout au long de l'exécution. La première technique emploie les directives `#define`. L'autre consiste à déclarer des valeurs de type entier, donc des variables, mais qui ne varient plus. Voici par exemple une directive `#define` pour rendre le numéro de la broche 2 équivalent à un nom :

```
#define LOOP1  2
```

Voici la création d'une valeur entière qui va contenir la même valeur :

```
int LOOP1 = 2;
```

Si vous regardez plusieurs fichiers de code source Arduino, vous allez constater que les deux techniques sont utilisées, le résultat étant le même. Pourtant, certains n'aiment pas utiliser les directives `#define`, alors qu'elles offrent un avantage au niveau de l'encombrement mémoire.

Comme montré dans le Listing 5.5, en utilisant `#define`, nous obtenons un fichier exécutable plus compact. Dans l'exemple d'alarme, le fichier exécutable pèse 1 452 octets. Si nous remplaçons les directives `#define` par des déclarations de variables, il monte à 1 540 octets. Cet écart de 88 octets semble négligeable, mais si vous avez un programme avec de nombreuses définitions de constantes, cette différence peut se faire sentir. Lorsqu'il s'agit de faire tenir la totalité du code exécutable dans les 30 720 octets que propose l'ATmega328, cela peut faire la différence entre réussir à téléverser et échouer.

Les variables globales

Vous remarquez que tous les croquis utilisent des variables globales, et c'est une pratique habituelle dans le monde Arduino. La raison est très simple : votre fonction `loop()` est sans cesse appelée par la fonction principale `main()`. Si les variables étaient déclarées dans le corps de votre fonction, elles seraient effacées et recréées à chaque nouveau tour de boucle. Mais vous avez parfois besoin de pouvoir retrouver la valeur de la variable au tour suivant, par exemple un compteur de tours. Il faut donc que la variable soit déclarée en dehors de la fonction `loop()`. En déclarant les variables de façon globale, vous pouvez facilement leur donner des valeurs initiales dans la fonction `setup()`, donc avant le premier appel à la fonction `loop()`. Toutes les autres fonctions du croquis auront accès à la valeur de ces variables.

Un certain nombre de programmeurs préfère éviter les variables globales, en raison du risque de couplage indésirable entre plusieurs parties du programme. Ce risque existe effectivement dans les grandes applications qui chargent des données depuis un disque, lancent des traitements et sont arrêtées sur ordre de l'utilisateur. C'est le cas des traitements de texte, des jeux et des navigateurs, par exemple. En effet, il ne faut pas qu'une partie du programme modifie la valeur d'une variable globale à l'insu d'une autre partie du programme, ou en même temps que cette autre partie. Il est heureusement possible de se passer de la plupart des variables globales dans les systèmes qui offrent beaucoup de

mémoire, grâce à des mécanismes de réservation dynamique, des drapeaux d’accès appelés sémaphores et des verrous mutex, et la possibilité de transmettre des pointeurs sur les structures de données, et non les valeurs elles-mêmes (valeurs littérales).

En revanche, dans un système embarqué, les variables globales permettent d’utiliser efficacement le faible espace mémoire disponible. Toutes les fonctions y auront accès. Vous pouvez les comparer aux indicateurs et interrupteurs d’un panneau de contrôle, par exemple comme dans un cockpit d’avion. Le principe est qu’une seule fonction modifie la valeur, et d’autres fonctions la lisent.

De plus, un croquis Arduino n’est pas prévu pour fonctionner avec plusieurs processus en parallèle (threads ou exétrons). Les risques d’accès simultané à une même variable sont donc nuls. Il faut seulement bien gérer les éventuelles interruptions, mais il suffit de réfléchir à leur fonctionnement pour s’éviter des problèmes.

Librairies de fonctions (ou bibliothèques)

Les librairies essentielles qui sont fournies en même temps que l'atelier Arduino répondent à de nombreux besoins, mais pas à tous, comme vous le verrez dans le Chapitre 7. Lorsqu'un nouveau type de capteur ou de composant apparaît, vous ne trouverez pas les fonctions de librairies permettant de l'exploiter. Dans ce cas, ce sera à vous d'écrire le code nécessaire.

La notion de librairie n'a pas exactement le même sens dans l'atelier Arduino que dans l'environnement habituel de programmation des machines hôtes avec GCC ou Visual Studio. Lorsque vous créez un logiciel par exemple pour un PC sous Windows, la librairie est un fichier archive contenant des objets binaires ; c'est une librairie statique. Sous Linux et Unix, ce genre de fichier est produit avec les outils *ln*, *ar* et *ranlib*. Le fichier binaire résultant réunit une collection de fichiers objets avec un index (voyez l'encadré « Code source, code objet et images » en début de chapitre). L'outil lieur se sert de l'index pour retrouver les références qu'il doit résoudre afin de produire le fichier exécutable complet. Si la librairie était dynamique, elle ne serait chargée en mémoire qu'au démarrage du programme, et même parfois au dernier moment, lorsqu'un des éléments qu'elle contient est appelé. Dans ce cas, la liaison est dite dynamique.

Dans l'environnement Arduino, les librairies sont fournies sous forme de code source. Lorsque vous compilez un croquis, toutes les librairies auxquelles il fait appel sont compilées ainsi que le code de support d'exécution afin de produire l'image exécutable binaire.

Accès aux librairies depuis un croquis

Lorsque l'atelier Arduino rencontre dans le code source une directive d'inclusion faisant référence à une librairie qui est déclarée dans l'atelier, il va réussir à générer les options de compilation et de liaison pour insérer automatiquement le code demandé. Pour que cela soit possible, la librairie doit donc être connue de l'atelier.

Il faut donc recenser ou déclarer la librairie comme indiqué dans la prochaine section. Dès l'installation, l'atelier connaît déjà toutes les librairies dites

essentielles. La Figure 5.7 montre à quoi ressemblait la liste des librairies connues pour une ancienne version de l'atelier. La Figure 5.8 montre la même liste pour une version très récente.

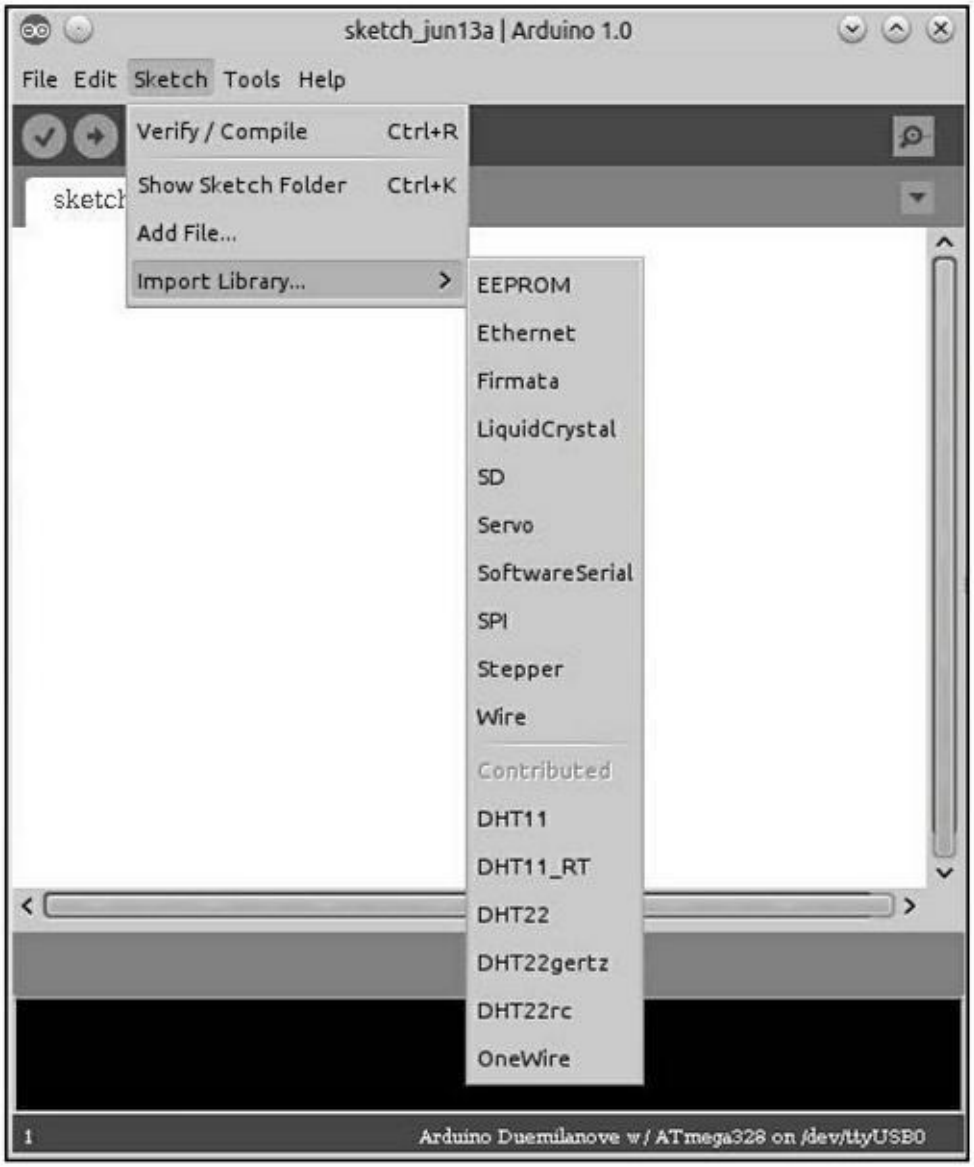


Figure 5.7 : Liste des librairies déclarées dans une ancienne version de l'atelier Arduino.

Le Listing 5.6 montre comment incorporer une librairie dans un croquis, en l'occurrence la librairie `SoftwareSerial`. Vous remarquez tout d'abord la directive d'inclusion tout au début du croquis :

```
#include <SoftwareSerial.h>
```

Grâce à cette directive, l’atelier sait qu’il doit chercher une librairie portant ce nom. Il suppose que cette librairie fait partie de celles qui sont déclarées.

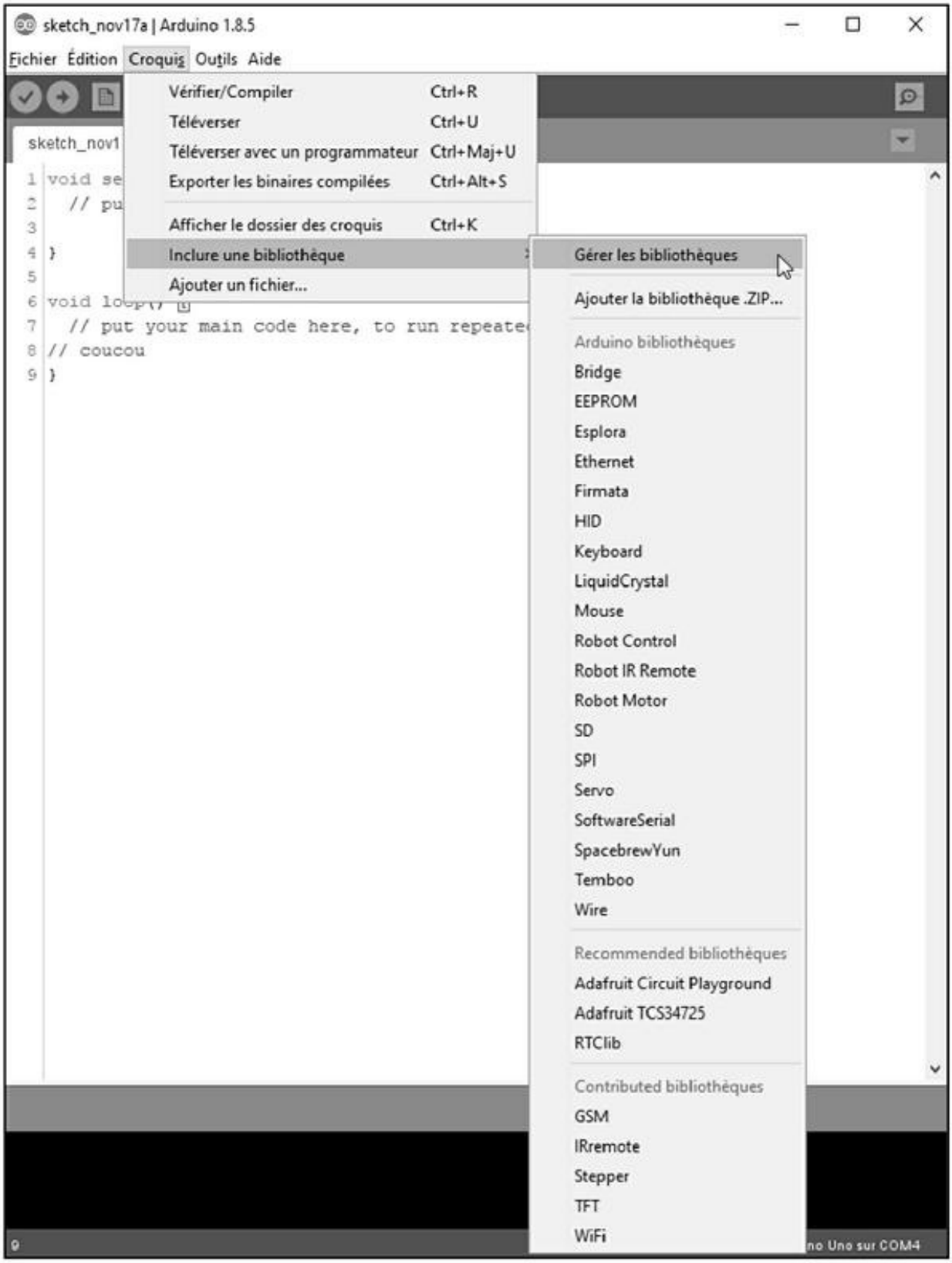


Figure 5.8 : Liste des librairies dans une très récente version de l'atelier.

Listing 5.6 : Premier exemple d'utilisation d'une librairie.

```
#include <SoftwareSerial.h>

SoftwareSerial softSerial(2, 3);

int analogPin = A0;  // Broche d'entrée du potentiomètre
int analogVal = 0;   // Variable pour stocker la valeur
lue

void setup() {
  // Réglage du débit du port SoftwareSerial
  softSerial.begin(9600);
}

void loop() // Execution répétée
{
  // Lecture d'entrée analogique
  analogVal = analogRead(analogPin);

  // Écriture de la valeur sur le port softSerial
  if (softSerial.available())
    softSerial.write(analogVal);

  // Pause d'une seconde (1000 millisecondes)
  delay(1000);
}
```

Un objet nommé `softSerial` est créé en tant qu'instance de la classe `SoftwareSerial` :

```
SoftwareSerial softSerial(2, 3);
```

Les paramètres de l'appel au constructeur définissent l'utilisation des broches numériques 2 et 3 pour la fonction d'entrées-sorties série.

La fonction de préparation `setup()` permet de définir le débit en bauds.

Dans la fonction de boucle `loop()`, nous commençons par lire une valeur depuis la broche analogique A0 et renvoyons la valeur en sortie grâce à une

méthode de l'objet. Nous marquons ensuite une pause avant le prochain tour de boucle.

Ce simple exemple permet d'échanger des données binaires entre deux cartes Arduino, mais les données ne sont jamais présentes dans un format lisible par un humain. Pour afficher une trace lisible du traitement, il faut recourir à l'une des deux méthodes `print()` ou `println()`, comme le montre le Listing 5.7.

Listing 5.7 : Exemple d'affichage de valeurs lues depuis un port série.

```
#include <SoftwareSerial.h>

SoftwareSerial softSerial(2, 3);
int analogPin = A0;   // Broche d'entrée pour le
potentiomètre
int analogVal = 0;    // Variable pour stocker la valeur
lue

void setup() {
  // Réglage du débit du port SoftwareSerial
  softSerial.begin(9600);
}

void loop() {    // Lecture de valeur analogique
  analogVal = analogRead(analogPin);
  // Renvoi des données sur le port softSerial
  softSerial.print(analogVal);
  softSerial.println();         // Ajoute un saut de
ligne
  // Pause d'une seconde
  delay(1000);
}
```

En théorie, vous pourriez utiliser la fonction standard `print()` pour envoyer des données sur le port série, mais cette fonction n'est pas préactivée pour les fonctions essentielles Arduino. Si vous avez besoin d'activer `print()` dans ce sens, et découvrir l'impact de cette opération sur vos croquis, vous irez voir la page dédiée du site Arduino (playground.arduino.cc/Main/Printf).

Notez que la librairie `SoftwareSerial` est décrite en détail dans le Chapitre 7. Vous pouvez également recourir à l'aide en ligne de l'atelier.

Ajout d'une librairie à l'atelier Arduino

Pour pouvoir exploiter un nouveau capteur ou un nouveau module dans l'atelier, il faut pouvoir ajouter la librairie de fonctions correspondante. Soit vous récupérez la librairie sur le site de la personne qui a bien voulu la partager, soit vous devez écrire vous-même le code source de cette librairie.

Les librairies complémentaires sont en général stockées dans un sous-répertoire portant le nom *libraries* qui dépend du répertoire de vos croquis *sketchbook* ou *Arduino*. Tous les fichiers qui constituent la librairie sont répartis dans des sous-répertoires appropriés. C'est le nom du sous-répertoire principal qui apparaît dans la liste des librairies de l'atelier. Vous disposez de deux méthodes pour ajouter une librairie :

Méthode 1, automatique

Depuis un certain temps, toutes les versions de l'atelier Arduino sont capables de détecter et d'installer d'office les fichiers de code source d'une nouvelle librairie dans votre sous-répertoire *sketchbook* ou *Arduino*. Il suffit d'utiliser dans l'atelier la commande **Croquis/Inclure une bibliothèque** puis de naviguer jusqu'au répertoire ou jusqu'au fichier compressé ZIP qui contient le code source. Selon la version de l'atelier Arduino, les détails des étapes pour inclure la librairie vont varier.

L'énorme avantage de cette méthode est que l'atelier va automatiquement déclarer la librairie, sans que vous ayez à redémarrer. Dans les plus récentes versions de l'atelier, vous pouvez même, sans quitter celui-ci, procéder au téléchargement et à l'installation des librairies complémentaires.

Méthode 2, manuelle

Pour installer manuellement une librairie dans l'atelier Arduino, il faut procéder ainsi :

- Tout d'abord, il faut quitter l'atelier si nécessaire.

- Vous décompressez les fichiers de la librairie dans un répertoire temporaire. La plupart des librairies sont fournies dans le format ZIP.

- Vous devez voir au moins deux fichiers de code source, le premier à extension *.c* ou *.cpp* et l'autre pour le fichier d'en-tête *.h*.

- Dans le sous-répertoire *sketchbook/libraries* de votre compte personnel, vous créez un sous-répertoire portant le même nom que ces deux fichiers.

- Vous copiez ensuite la totalité du contenu du répertoire temporaire dans le nouveau sous-répertoire que vous venez de créer.

- Dès que vous redémarrez l'atelier, vous devez pouvoir confirmer que le nom de la nouvelle librairie apparaît bien dans la liste déroulante de la commande **Croquis/Inclure une bibliothèque**.

Voici la structure qu'il est conseillé d'utiliser pour distribuer le contenu d'une librairie complémentaire :

```
IRTracker/IRTracker.cpp
         /IRTracker.h
         /keywords.txt
         /README
         /utility/IRCal.cpp
                 /IRCal.h
                 /IREncode.cpp
                 /IREncode.h
                 /IRDecode.cpp
                 /IRDecode.h
         /Examples/ScanIR/ScanIR.ino
                  /SendStop/SendStop.ino
```

Notez que ce qui précède n'est qu'un exemple, car à l'heure où j'écris ces lignes, je ne crois pas qu'il existe une librairie portant le nom IRTracker pour Arduino.

Il est fondamental que le nom du sous-répertoire de la librairie soit le même que celui des deux fichiers source principaux. Cette convention est valable pour tous les sous-répertoires de l'atelier. Voyez par exemple le sous-répertoire des exemples Arduino (`/usr/share/arduino/examples` sous Linux) : les sous-répertoires portent le même nom que les fichiers source *.ino*.

En parcourant les fichiers des exemples, vous pouvez constater que certaines fonctions utilitaires font l'objet d'autres sous-répertoires avec d'autres fichiers source en plus des deux obligatoires. Ces éléments ne vont pas apparaître dans la liste des librairies, mais l'atelier saura y accéder au besoin.

Vous remarquerez la présence d'un fichier portant le nom *keywords.txt* dans le sous-répertoire racine de chaque librairie. C'est dans ce fichier que sont réunies des définitions pour les mots-clés utilisés dans le code source. Voici par exemple à quoi pourrait ressembler ce fichier pour notre librairie imaginaire IRTracker, avec la structure indiquée plus haut :

```
# IRTracker
###################################
MAXPWR    LITERAL1
MAXTIME   LITERAL1

LastCal   KEYWORD1
TotHours  KEYWORD1

IRState   KEYWORD2
IRSense   KEYWORD2
```

Les lignes qui commencent par un signe dièse sont des lignes de commentaires qui sont ignorées. Les mots spéciaux, repris dans le Tableau 5.1, servent à identifier les catégories de mots : constantes, noms de classes, structures, variables ou fonctions.

Tableau 5 1 : Quelques définitions de mots-clés pour le fichier keywords.txt.

Type	Définition
LITTERAL1	Constantes littérales et macros
LITTERAL2	Autres constantes littérales
KEYWORD1	Classes, types de données et mots-clés du C++
KEYWORD2	Méthodes et fonctions
KEYWORD3	Structures de données

Vous trouverez un fichier *keywords.txt* plus complet dans le répertoire d'installation des composants standard d'Arduino. Sous Linux, cela correspond au sous-répertoire */usr/share/arduino/ lib*. Sous Windows, vous irez voir dans *C:\Program Files\Arduino\lib*. N'hésitez pas à regarder le contenu du fichier *keywords.txt* d'autres librairies standard.

Vous pourriez être tenté d'ajouter les fichiers de vos librairies complémentaires au même endroit que les librairies essentielles (celles installées en même temps que l'atelier). Cette approche est déconseillée. Les librairies essentielles sont automatiquement mises à jour en même temps que l'atelier, mais vos librairies supplémentaires, même si elles sont dans les mêmes répertoires, ne seront jamais mises à jour.

Lorsque vous installez une librairie complémentaire, l'atelier va analyser le contenu du fichier d'en-tête et ajouter une directive `#include <nomlibrairie.h>` dans votre croquis.

Création d'une librairie spécifique

La procédure de création d'une librairie spécifique pour l'atelier Arduino est bien définie, ce qui ne signifie pas que ce soit simple. Le code source est une définition de classe dans une version limitée du C++, avec parfois quelques classes associées. Vous pouvez vous faire une idée de la structure d'une librairie en lisant le Chapitre 7, qui montre comment est organisé le code source.

Au minimum, la librairie doit être constituée des deux fichiers *.c* (ou *.cpp*) et *.h*. Le fichier source contient la définition de la classe et le fichier d'en-tête contient la déclaration, les définitions des types et des macros.

Je rappelle que le répertoire qui contient ces deux fichiers doit porter le même nom que le fichier source. Il doit contenir les deux autres fichiers nommés *keywords.txt* et *README*. Souvent, vous pouvez ajouter un sous-répertoire portant le nom *examples*, dans lequel vous stockez au moins un exemple d'utilisation. Cela sera apprécié si vous comptez faire profiter les autres de votre travail.

Le code source de l’atelier Arduino

Le code source Arduino contient les fichiers pour les cartes basées sur les microcontrôleurs AVR, ainsi que ceux pour la carte Due qui est basée sur le processeur ARM (variante que je ne présente pas dans ce livre). Vous pouvez récupérer un fichier compressé de la totalité du code source depuis le site référentiel GitHub.

L’étude de ce code source est vraiment conseillée à ceux qui veulent devenir plus intimes avec l’environnement de développement qu’en cliquant dans des menus. Vous pouvez par exemple profiter d’un outil de documentation tel que Doxygen (`stack.nl/~dimitri/doxygen/`) qui permet de créer une série de pages Web avec des graphes de dépendance et d’appels ainsi qu’un index de toutes les classes, fichiers source et fonctions. Le code source Arduino ne contient pas vraiment de balises spécifiques pour Doxygen, mais l’outil permet de générer une documentation bien pratique.

Au premier niveau, le code source contient des répertoires pour le code source des applications (*app*), pour les modules de génération de code (*build*), pour les fichiers spécifiques aux matériels (*hardware*) et pour tous les modules des librairies essentielles (*libraries*). Les deux gisements les plus intéressants pour descendre dans les détails de l’environnement sont les sous-répertoires *hardware* et *libraries*.

Le sous-répertoire *hardware* contient le code source des différents amorceurs (*bootloaders*), le code de support d’exécution, qui correspond au nom Core. C’est là que vous trouverez le fichier *main.cpp*. Vous y trouverez également une série de modules pour la gestion du circuit mémoire EEPROM, des entrées-sorties série et des interfaces SPI et TWI. Notez que ces éléments sont dans le sous-répertoire *hardware/avr/arduino/libraries*, à ne pas confondre avec le répertoire *librairies* indiqué plus haut. En effet, toutes les librairies de support de l’atelier se trouvent dans le sous-répertoire de premier niveau.

Les sous-répertoires de *librairies* contiennent le code source pour le son, la connexion Ethernet, les afficheurs LCD, les cartes mémoire SD, les servomoteurs, les moteurs pas à pas, les afficheurs TFT et le module WiFi. Vous y trouverez également des librairies pour les cartes Arduino Robot, Esplora et Yun. La plupart des sous-répertoires contiennent des croquis d’exemples et certains offrent également une petite documentation au format texte.

Si vous lisez ces fichiers de code source, vous constaterez qu'ils utilisent énormément les directives conditionnelles `#if`, `#ifdef` et `#define`. C'est ce qui permet d'inclure ou pas certains éléments en fonction du type de carte sélectionné. Au départ, cela rend l'étude difficile, et il faut un peu de patience pour s'y retrouver. Remarquez enfin que parfois c'est une ou plusieurs fonctions qui sont définies, et parfois c'est une classe, donc avec des méthodes.

CHAPITRE 6

Programmer sans l’atelier Arduino

Le matériel Arduino n’est à vrai dire qu’une carte de développement assez simple autour d’un microcontrôleur de la famille AVR d’Atmel. La raison du succès d’Arduino auprès de l’immense foule des non-programmeurs est l’atelier Arduino ainsi que le microgiciel amorceur (bootloader), car il simplifie le processus. Il est néanmoins possible de se passer de cet atelier Arduino. Le confort qu’il apporte en prenant en charge tous les détails des différentes étapes de production du fichier exécutable n’est pas toujours nécessaire, et certains préfèrent travailler depuis la ligne de commande et leur éditeur de texte favori.

Nous allons découvrir plusieurs solutions pour produire un programme Arduino d’une autre façon, et notamment comment utiliser la chaîne d’outils avr-gcc depuis la ligne de commande en mode texte, avec pour seule aide un fichier décrivant le processus, fichier nommé Makefile. Nous verrons également un aperçu de ce qu’il est possible de faire dans le langage assembleur pour tirer le maximum de performance du microcontrôleur.

L’étape ultime de téléversement de l’exécutable vers le microcontrôleur peut elle aussi se passer de l’atelier Arduino. C’est ce que nous verrons pour clore le chapitre.

Deux alternatives à l'atelier Arduino

En remplacement de l'atelier Arduino IDE, vous pouvez par exemple adopter l'outil nommé PlatformIO. Il existe pour Linux, pour macOS et pour Windows. Il combine un outil de production de code exécutable et un gestionnaire de librairies fonctionnant sur ligne de commande. Il est développé en langage Python. Un autre outil permettant de produire des programmes Arduino, lui aussi développé en Python, est l'outil nommé Ino. Actuellement, il n'existe que pour Linux et macOS. Une version Windows apparaîtra peut-être prochainement.

PlatformIO

L'outil sur ligne de commande PlatformIO (`platformio.org`) permet de générer du code pour plus de 100 modèles différents de microcontrôleurs. Il est très portable, la seule condition étant d'avoir installé au préalable Python 2.7 (pas Python 3) sous Windows, macOS ou Linux.

N.d.T. : PlatformIO existe dorénavant en version graphique, en intégrant un éditeur. La version qui fonctionne purement sur ligne de commande s'appelle dorénavant PlatformIO Core. C'est celle qui nous intéresse ici.

PlatformIO sélectionne automatiquement les outils de sa chaîne qui sont nécessaires en fonction du type de microcontrôleur sélectionné. Il enchaîne l'exécution des outils dans l'ordre adéquat pour réaliser la compilation, la liaison et le téléversement. Parmi les nombreuses cartes reconnues, citons la famille Trincket de Adafruit, toutes les cartes Arduino AVR, le BitWizard Arduino, les cartes Digispark de Digistump, la carte Engduino, les cartes LowPowerLab Mote, la MicroDuino, les Sanguino AVR et toutes les cartes AVR de SparkFun. Le système reconnaît même des cartes basées sur les processeurs ARM comme l'Atmel SAM, la STM32, la LPC de NXP, la Nordic nRF51 ainsi que la MSP430 de Texas, et la liste ne s'arrête pas là !

L'installation de PlatformIO, sous Linux ou sous macOS, est fort simple. Il suffit de télécharger un script d'installation ou d'utiliser l'outil *pip* de Python. Vous aurez peut-être besoin d'installer d'abord l'outil de transfert de données *cURL*,

mais c’est un outil très répandu que vous pouvez directement installer sous forme de paquetage sous Linux.

La Figure 6.1 montre la console juste après avoir téléchargé et installé les paquetages de PlatformIO.

Une fois l’installation terminée, vous pouvez par exemple saisir la commande `platformio boards` pour voir s’afficher l’énorme liste des cartes reconnues.

Pour que vos fichiers restent bien rangés, PlatformIO propose le concept de projet. À chaque projet correspond au départ un fichier de configuration et deux sous-répertoires, *src* et *lib*. Il faut absolument personnaliser le fichier de configuration avant de l’utiliser. C’est un fichier de style .INI au format KVP (constituée de paires clé/valeur). Une documentation détaillée de ce fichier de configuration est disponible à l’adresse `bit.ly/platformio-pcf`.

Je vous conseille de créer pour vos projets PlatformIO un sous-répertoire distinct du répertoire de croquis utilisé par l’atelier Arduino pour vos croquis. J’ai par exemple choisi de l’appeler tout simplement *platformio*. Pour démarrer la création d’un projet, il suffit de saisir la commande `platformio init`. La Figure 6.2 montre ce qui est affiché dans la console lors de la création d’un projet.

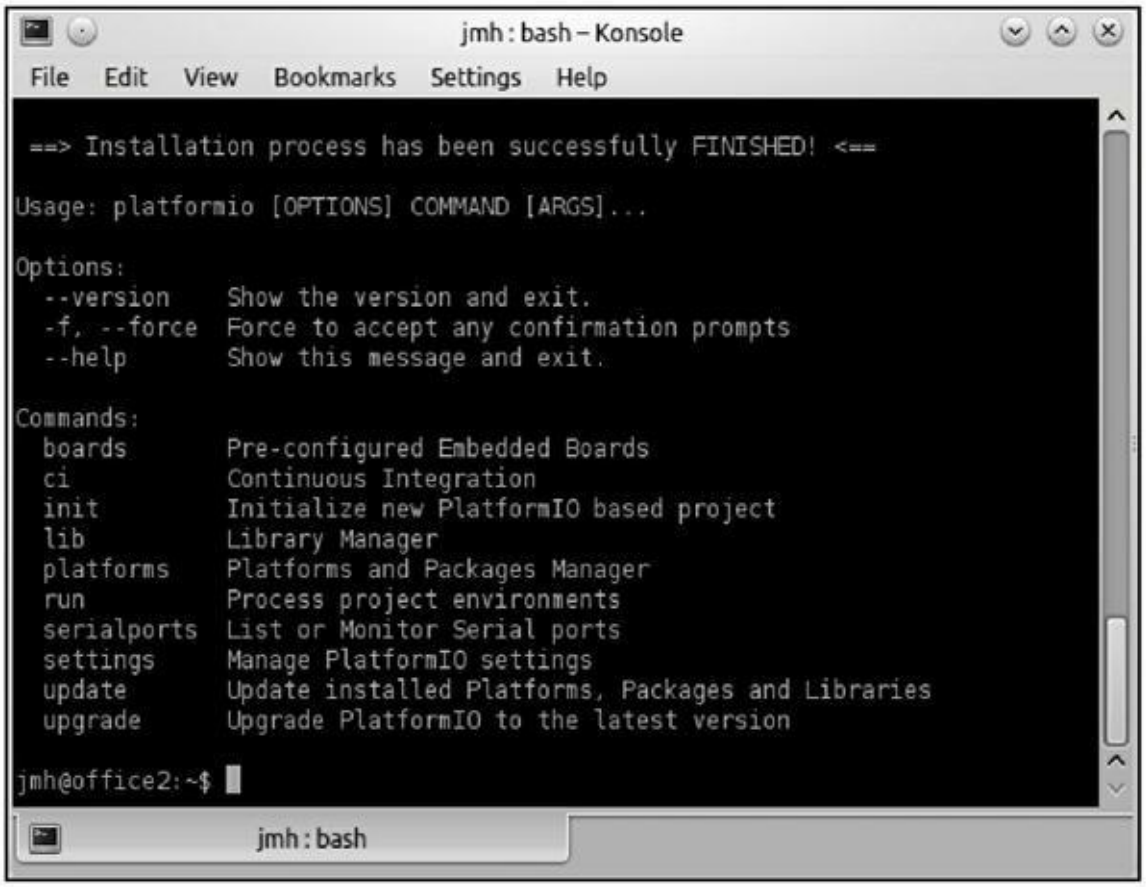


Figure 6.1 : Affichage de la console après installation de PlatformIO.

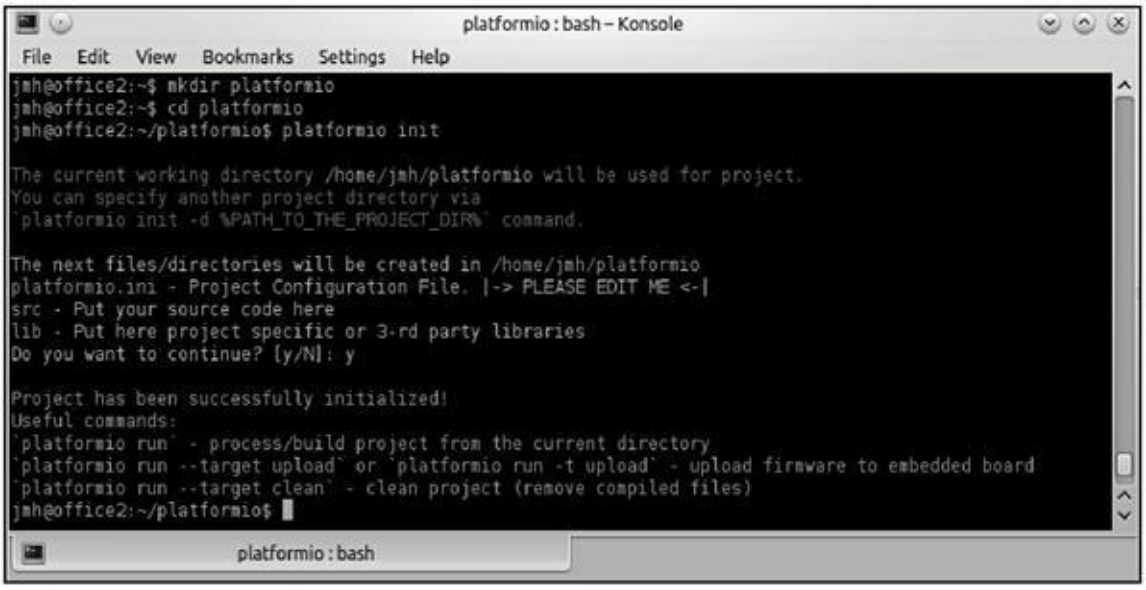


Figure 6.2 : Démarrage d'un nouveau projet avec PlatformIO.

PlatformIO offre de nombreuses possibilités. Il permet d'utiliser des infrastructures (*frameworks*) prédéfinies pour différents types de cartes ; il comporte un gestionnaire de librairies et peut être intégré à un atelier de développement ou à un éditeur, un peu comme celui d'Arduino. De tels outils s'appellent par exemple Eclipse, Energia, QT Creator, Sublime Text, VIM ou encore Visual Studio.

N.d.T. : Depuis 2016, PlatformIO peut être installé dans une version intégrant directement l'un des deux environnements d'édition Atom ou Visual Studio Code. Le résultat porte le nom PlatformIO EDI.

L'outil Ino

Ino est le nom d'un outil sur ligne de commande qui permet de générer un programme exécutable pour des cibles Arduino en se fondant sur un fichier de description de processus Makefile. Comme PlatformIO, il est produit en langage Python, mais il n'a comme cible que les cartes Arduino. Il reconnaît les fichiers de code source *.ino* et *.pde* ainsi que les fichiers *.c* et *.cpp*. Il prétend reconnaître toutes les cartes que reconnaît l'atelier Arduino. Actuellement, il ne fonctionne que sous Linux ou macOS, avec la version 2.7 de Python.

Vous pouvez télécharger puis installer l'outil en récupérant un fichier d'archive compressée Tar (`inotool.org/#installation`), en produisant un clone depuis le site GitHub ou en utilisant l'outil *pip* de Python, ou l'outil *easy_install*. Personnellement, j'ai utilisé *pip*. Pour obtenir une aide une fois l'installation terminée, utilisez la commande `ino -h` (Figure 6.3).

Ino fonctionne avec des fichiers make, mais ce n'est pas visible par l'utilisateur. Comme PlatformIO, il propose une structure arborescente pour les projets. Au démarrage d'un projet, il crée les mêmes deux sous-répertoires *src* et *lib* ainsi qu'un fichier de code source minimal servant d'embryon. Ce fichier, qui porte le nom *sketch.ino*, est placé dans le sous-répertoire *src*. La logique est la même que dans les récentes versions de l'atelier Arduino. Il ne reste plus qu'à écrire vos instructions et produire les fichiers source complémentaires dont vous avez besoin.

Pour tout détail au sujet d'Ino, rendez-vous sur le site Web `inotool.org`. Notez que l'outil est également disponible depuis la page d'index des paquetages Python `bit.ly/ino-ppi`.

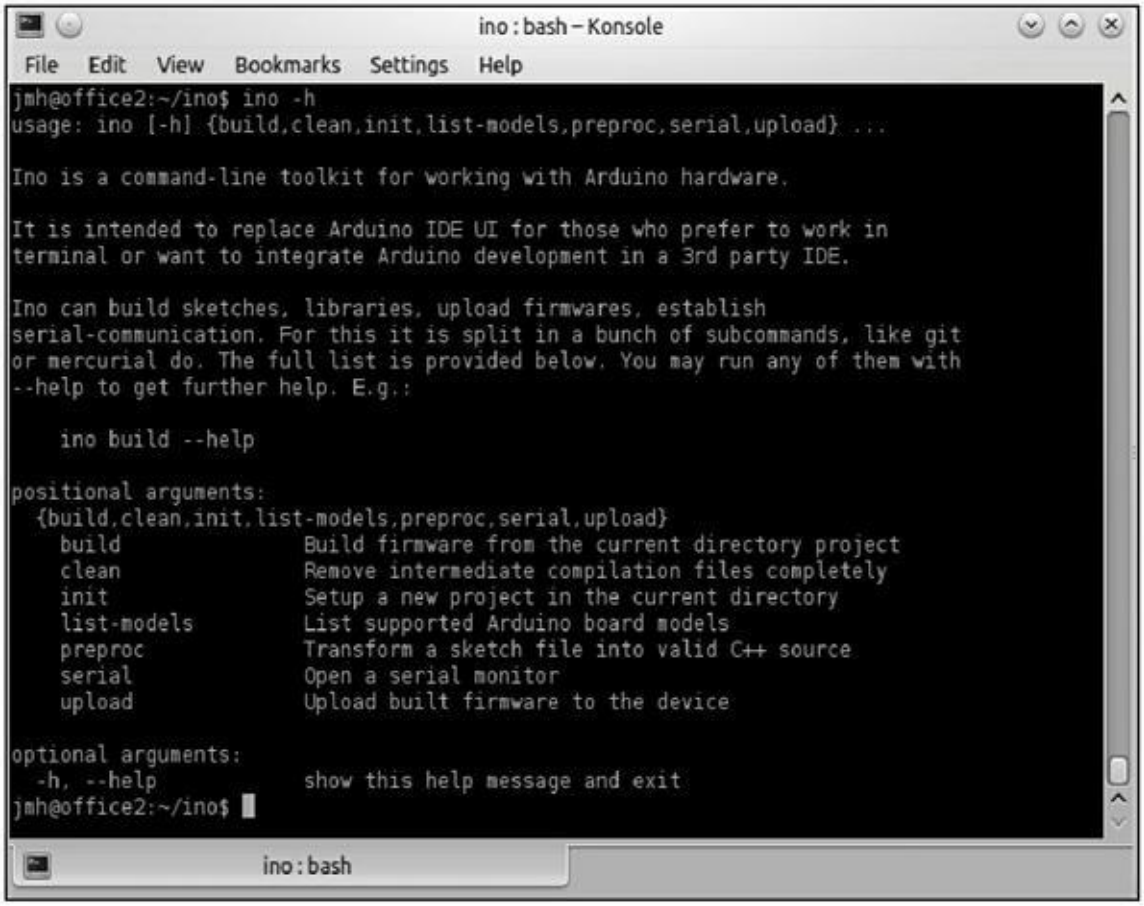


Figure 6.3 : L'écran d'aide affichée par l'outil Ino.

La chaîne d’outils avr-gcc

Avr-gcc est le nom de la chaîne d’outils qui réalise le travail effectif de transformation des fichiers de code source en langage C et C++ en un fichier exécutable qu’il ne reste plus qu’à téléverser vers le microcontrôleur AVR. L’atelier Arduino n’est qu’une couche de présentation qui masque les détails des options qu’il faut préciser pour chacun des outils de la chaîne. C’est la même chaîne d’outils qui est utilisée par les alternatives PlatformIO et Ino, sauf que ces outils utilisent des scripts en Python pour contrôler les enchaînements. Autrement dit, le compilateur, le lieur, l’assembleur et les autres outils sont tous disponibles directement si vous avez envie ou besoin de produire votre exécutable AVR depuis la ligne de commande en créant un fichier de définition Makefile, ou même en lançant chacune des étapes à la main.

L’installateur de l’atelier Arduino procède évidemment aussi à l’installation de la chaîne d’outils avr-gcc puisqu’il en a besoin. Sous Windows, vous pouvez installer la suite d’outils WinAVR qui va installer les fichiers ailleurs que dans les répertoires de l’atelier Arduino. Cela permet d’avoir les deux chaînes d’outils en parallèle, ce qui est parfois utile pour disposer des outils dans leur plus récente version, au cas où l’atelier ne les aurait pas encore incorporés.

Le Tableau 6.1 dresse la liste de tous les outils de la chaîne avr-gcc, tels qu’ils sont disponibles juste après l’installation du paquetage Arduino. Ce sont les mêmes programmes que vous trouverez sous Linux, sous Windows ou sous macOS. L’installation de la chaîne d’outils en dehors de l’atelier Arduino est décrite dans la section suivante.

Tableau 6.1 : Liste des outils de compilation avr-gcc.

Outil	Description
avr-addr2line	Conversion des adresses en noms de fichiers et numéros de lignes
avr-ar	Génération d’une archive de code objet et modification ou extraction de code
avr-as	Assembleur GNU optimisé pour AVR

avr-c++filt	Déconcentration des symboles assembleur vers les noms C++
avr-gcc	Compilateur GNUGCC pour produire du code objet AVR (!)
avr-g++	Compilateur GNUG++ pour produire du code objet AVR (!)
avr-ld	Lieur GNU pour le code objet AVR
avr-nm	Création de la liste des symboles trouvés dans les fichiers objets
avr-objcopy	Conversion de format des fichiers de code objet
avr-objdump	Affichage d'informations au sujet des fichiers objets
avr-ranlib	Génération d'un index pour un fichier archive de librairie
avr-realdef	Affichage d'informations au sujet des fichiers au format binaire ELF
avr-size	Calcul et affichage de la taille des sections et de la taille totale des fichiers objets
avr-strings	Affichage des chaînes affichables trouvées dans les fichiers binaires
avr-strip	Suppression des symboles dans les fichiers objets AVR
avrdude	Outil de téléversement pour microcontrôleurs AVR (!)

Certains de ces outils sont indispensables pour produire un exécutable AVR, mais pas tous. À l'exception d'avrdude, ce sont en fait des versions pour microcontrôleurs AVR des outils GNU existants pour Windows ou Linux. Les

outils indispensables pour travailler avec une carte Arduino sont *avr-gcc*, *avr-g++*, *avr-ar*, *avr-as*, *avr-ld*, *avr-objcopy*, *avr-ranlib* et *avrdude*. Dans le tableau précédent, les outils qui ne font pas partie de la collection *binutils*, présentée plus loin, comportent un point d'exclamation (!) en fin de description.

L'outil central est bien sûr le compilateur, c'est-à-dire avr-gcc, ou avr-g++ pour le C++. C'est une version du compilateur open source GNU adaptée pour compiler du code source C ou C++ avec comme cible un microcontrôleur AVR. Ce compilateur ressemble beaucoup à la version classique de GCC, en y ajoutant quelques options prédéfinies pour simplifier la cross-compilation vers AVR.

Si vous listez le contenu du répertoire dans lequel ont été installés ces outils, vous verrez ceci :

```
avr-c++
avr-cpp
avr-g++
avr-gcc
avr-gcc-4.9.2
```

Il s'agit de plusieurs variantes du compilateur GNU. Les outils avr-c++ et avr-g++ sont homonymes. De même, avr-cpp, avr-gcc et avr-gcc-x.x.x sont identiques. Pour connaître le numéro de version, vous pouvez par exemple saisir `avr-gcc -v`.

Une fois le code source compilé, c'est au lieur avr-ld de réunir tous les modules de code objet afin de produire l'exécutable. Le lieur ne sait pas travailler avec des fichiers de code source ; il lui faut des fichiers au format de code objet.

Après la liaison, il reste encore à convertir le fichier vers le format Intel Hex qui contient une représentation ASCII du code binaire en notation hexadécimale. C'est ce fichier qui peut être transféré vers la carte cible. Sur la carte, le programme amorceur va détecter l'arrivée des données et convertir les valeurs exactes en valeurs binaires qui sont stockées dans la mémoire flash du microcontrôleur.

La Figure 6.4 illustre la cinématique d'enchaînement de la compilation, de la liaison, de la conversion et du téléversement.

Les fichiers de code objet de la Figure 6.4 sont soit ceux produits à partir de vos fichiers de code source, soit des fichiers fournis dans une librairie pour un capteur ou un composant. S'y ajoute le code de support d'exécution avr-gcc, et

notamment celui de la fonction principale `main()`. Les librairies dans la figure sont soit les librairies préinstallées comme *avr-libc*, soit d'autres librairies créées indépendamment avec avr-ar et avr-ranlib. S'y ajoutent également les fichiers de code objet produits à partir des codes source de librairies.

Le composant que je n'ai pas encore décrit en détail est la librairie de support d'exécution *avr-libc*. Elle contient la plupart des fonctions que l'on trouve dans une librairie d'exécution standard du C et y ajoute quelques fonctions spécifiques aux microcontrôleurs AVR. Je décris cette librairie indispensable un peu plus loin dans ce même chapitre.

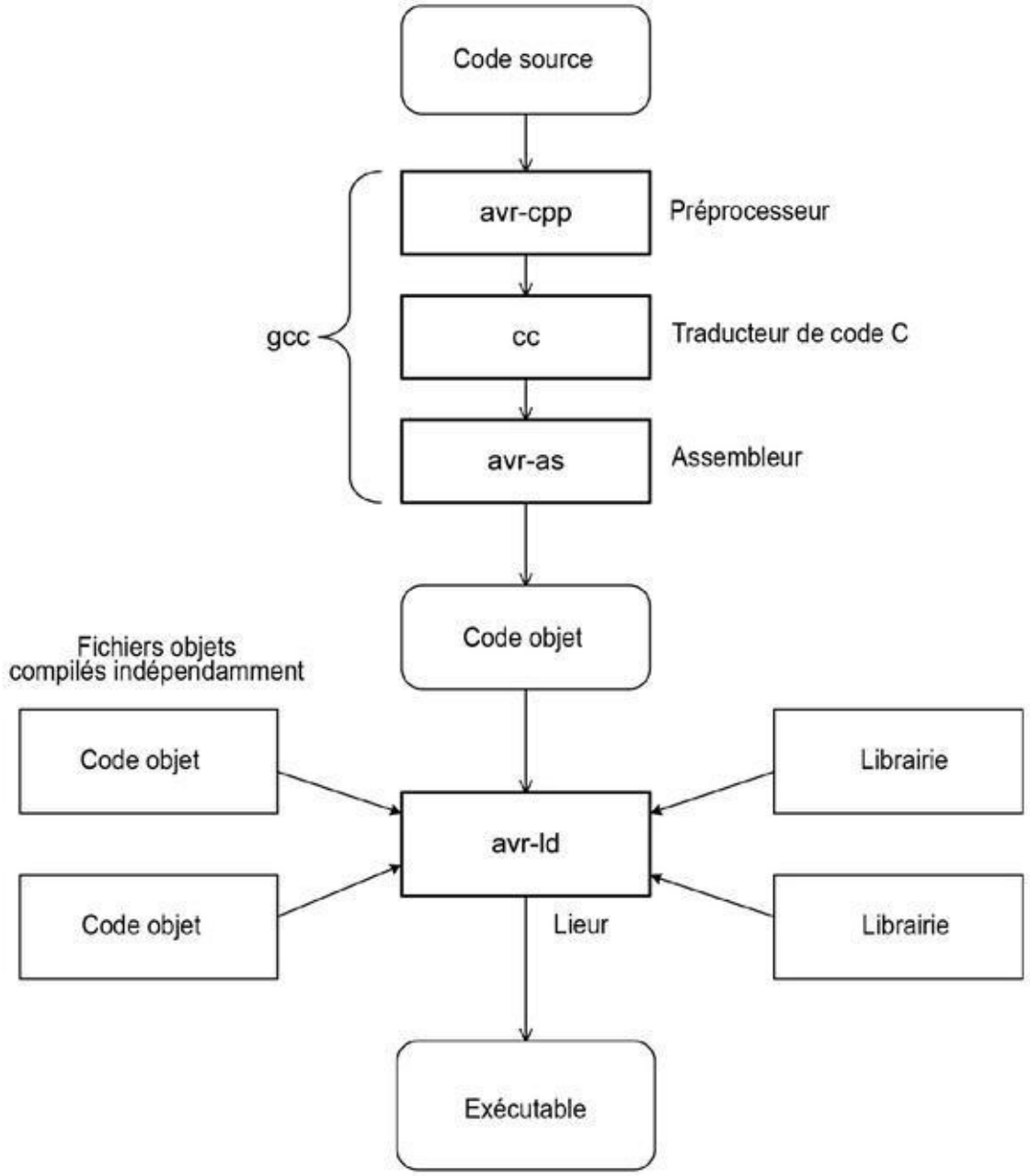


Figure 6.4 : Cinématique de la chaîne de cross-compilation avr-gcc.

Installation de la chaîne d'outils

La chaîne d'outils est installée d'office lorsque vous installez l'atelier Arduino. Cependant, si vous décidez de ne pas installer l'atelier, vous pouvez mettre en place la chaîne d'outils directement, ou même chacun de ses outils individuellement.

Voici les paquetages sous Linux qu'il faut installer au minimum :

- avr-lib
- avrdude
- avrdude-doc
- binutils-avr
- gcc-avr
- gdb-avr

Sous Windows, il ne sera pas inutile d'installer quelques utilitaires d'inspiration Unix tels que grep, fgrep, ls et diff. Sous Linux et macOS, ils sont tous déjà installés au départ. Notons au passage que la même chaîne d'outils peut être installée sur des systèmes moins répandus tels que Solaris, BSD Unix ou FreeBSD. Si cela vous concerne, sachez qu'il vaut mieux avoir un minimum de compétence en compilation de paquetages de code source. Pour la plupart des lecteurs, il sera plus simple de s'en tenir à un système d'exploitation pour lequel les paquetages d'installation sont déjà compilés.

Installation sous Windows

Comme déjà mentionné, le fait d'installer l'atelier Arduino installe d'office toute la chaîne d'outils dans un sous-répertoire de celui du code exécutable de l'atelier.

La société Atmel Corp fournit des paquetages d'installation précompilés, un peu de documentation et tout le code source pour les versions AVR de GCC et de binutils (qui regroupe la plupart des autres outils du Tableau 6.1). Les paquetages et les ressources se trouvent à l'adresse `bit.ly/atmel-avr-win` et le code source à l'adresse `bit.ly/atmel-source`.

Sous Windows, une autre solution pour installer la chaîne d'outils consiste à opter pour le paquetage WinAVR (`winavr.sourceforge.net`). Le script d'installation commence par créer un répertoire dans la racine, avec un nom qui commence par WinAVR puis y stocke les exécutables, les librairies et les fichiers d'en-tête. Vous trouverez également de la documentation au format PDF et HTML dans le sous-répertoire *doc*.

N. d. T. : WinAVR semble ne plus être mis à jour depuis plusieurs années.

Vous trouverez les outils GNU précompilés pour Windows sur le site de GNUWin (`gnuwin32.sourceforge.net`). Il s'agit d'une vaste collection d'outils système, mais cela ne comprend pas, paradoxalement, les deux compilateurs C et C++ ni le groupe d'outils binutils. Vous pourrez les télécharger depuis le site d'Atmel.

Pour découvrir d'autres outils et tutoriels, faites une recherche Web de l'expression « Windows AVR binutils ». Ceux qui apprécient l'environnement de développement Eclipse peuvent faire une recherche de l'expression « AVR Eclipse ». Visitez également le tutoriel de Protostack (`bit.ly/protostack-eclipse`).

Installation sous Linux

Sous Linux, installer la chaîne d'outils GNU-GCC et les autres outils se limite à sélectionner les paquetages appropriés dans un gestionnaire de paquetages. Vous pouvez sans souci installer simultanément la chaîne d'outils GCC normale, prévue pour Linux, et celle spécifique à AVR, les noms des versions AVR de tous les outils étant distingués par le préfixe *avr-*. Je vous déconseille de tenter de reconstruire tous ces outils à partir du code source, à moins d'avoir une raison impérieuse et d'être très à l'aise dans la production d'un ensemble de codes source complexe.

Installation sous macOS

Du fait de que le système macOS est basé sur un Unix BSD, ce que j'ai dit à propos de Linux s'applique à macOS. Les braves gens de la société Adafruit ont écrit un guide pour expliquer comment installer la chaîne d'outils AVR sous macOS (bit.ly/avr-osx).

La société Object Development propose un environnement de développement CrossPack qui réunit tous les outils pour produire du logiciel AVR sous macOS. Il ne requiert pas l'installation préalable de l'atelier XCode pour la compilation. Vous pouvez récupérer ces outils sur le site (bit.ly/crosspack-avr). Sachez que ce sont des outils en mode texte. Si vous voulez un éditeur à interface graphique ou un environnement de développement, il faudra l'installer vous-même.

D'ailleurs, il existe une version de l'atelier Eclipse pour macOS avec la chaîne d'outils AVR, sans compter que l'atelier Arduino existe bien sûr pour macOS.

Automate de production make

Pour un petit projet, en général constitué d'un seul fichier de code source, vous pouvez utiliser directement et successivement chacun des outils de la chaîne de production en spécifiant à chaque fois les bonnes options. En revanche, dès que le projet prend une certaine ampleur, il devient désagréable de saisir à répétition les différentes lignes de commande pour lancer individuellement chaque outil. Dans ce cas, il devient indispensable de pouvoir automatiser l'enchaînement des processus. C'est justement l'objectif de l'outil d'automatisation nommé make.

L'outil make est un automate qui déclenche des actions à partir des commandes et des règles définies dans un fichier de production appelé *Makefile*. Le langage de make est spécifique. Il sert à décrire les actions, les règles de dépendance et les conditions à remplir pour produire le fichier exécutable. C'est une sorte de langage de script ou de macro. En effet, les instructions du fichier utilisent des noms symboliques qui sont remplacés par des noms réels en fonction des traitements à déclencher. L'outil make possède une certaine intelligence, car il est capable de détecter par exemple qu'un des fichiers de code source a été modifié, et de décider de recompiler seulement tous les autres fichiers qui dépendent de celui qui a été modifié. Si par exemple le fichier A dépend des fichiers B et C et que le fichier B ait été modifié, l'outil va recompiler le fichier

B puis le fichier A, afin qu'ils tiennent compte des changements. L'outil make a été créé par Richard Stallman, Roland McGrath et Paul D. Smith.

De nombreux ateliers de développement, même les plus imposants, se servent de l'outil make en coulisses pour réaliser la production de l'exécutable. C'est par exemple le cas des deux outils PlatformIO et Ino, même s'ils masquent cette mise à contribution et réalisent un nettoyage en fin de traitement. Il existe même des outils pour vous aider à créer le contenu du fichier de production de make. Ceux qui ont déjà utilisé l'outil de création de paquetages *configure* connaissent ce genre d'utilitaires.

Le premier objectif de l'outil make est de permettre une gestion confortable d'un grand nombre de fichiers de code source. Il sait détecter qu'un fichier source a été modifié, pour en déduire quels autres fichiers source doivent être recompilés. L'outil sait lancer le compilateur, le lieur, le générateur de documentation et il peut même lancer un autre fichier Makefile en cascade (indispensable lorsque le code source est distribué dans de nombreux sous-répertoires). L'outil prend évidemment en compte la survenue d'une erreur dans un des composants tels que le compilateur ou le lieur.

Pour vous faire une première idée du fonctionnement de make, le plus simple consiste à lire le fichier Makefile d'un projet existant. La description de toutes les possibilités de l'outil make déborde largement les limites qui me sont imposées dans ce livre, et le manuel officiel de GNU qui lui est consacré fait plus de 200 pages. Voilà pourquoi vous trouverez des livres totalement dédiés à make et à ses applications (j'en cite dans l'annexe D). Vous pouvez télécharger le manuel officiel au format PDF depuis le site de GNU (`www.gnu.org/software/make/manual/make.pdf`).

Le compilateur avr-gcc

Les lettres GCC sont les initiales de *GNU Compiler Collection*, bien que certains parlent de *GNU C Compiler*. Le principe est d'offrir un compilateur polyvalent en utilisant un processeur symbolique différent pour chaque langage de programmation. Le résultat du traitement de ce processeur est un code intermédiaire unifié, qui est ensuite adapté à la machine cible. GCC permet de produire du code objet pour de nombreuses familles de processeurs : tout d'abord les processeurs Intel de nos PC, les Sparc RISC, la famille 68000, les microcontrôleurs MSP 432 de Texas Instruments, de nombreux microcontrôleurs

de type ARM, et bien d'autres. La suite de compilation croisée avr-gcc est configurée pour générer du code objet pour la famille de microcontrôleurs AVR.

Les deux outils avr-gcc et avr-g++ reconnaissent un très grand nombre d'options de ligne de commande pour préciser le mode d'optimisation, le mode de compilation, les arguments qu'il faut retransmettre au lieur après la compilation, les noms des chemins d'accès des fichiers d'en-tête, et bien sûr le nom du processeur cible. Parmi ces centaines d'options possibles, seule une poignée doit être spécifiée pour compiler du code pour une machine cible.

La production d'un exécutable comporte quatre étapes : prétraitement, compilation, assemblage et liaison, toujours dans cet ordre. Le préprocesseur va notamment insérer le contenu entier d'un fichier d'en-tête à la place de chaque directive `#include` qu'il rencontre. Normalement, ces directives ne doivent servir qu'à inclure des fichiers d'en-tête *.h* mais certains programmeurs se permettre de faire inclure des fichiers de code source ainsi (c'est fortement déconseillé). Le préprocesseur supprime également tous les commentaires et interprète les instructions conditionnelles du type `#if`, `#else` et `#endif`. Le résultat de cette première étape est un fichier de code source ne contenant que des instructions et rien d'autre. Vous pouvez étudier ce fichier intermédiaire en ajoutant l'option **-E** pour faire arrêter le traitement juste à la fin de la phase de prétraitement.

GCC va traduire les différents dialectes du langage C, comme le C normalisé ANSI, le C++, Objective C ou Objective C++ pour produire un fichier intermédiaire en langage assembleur. Normalement, ce format peut être fourni en entrée de l'outil assembleur avr-as qui va produire un fichier au format objet. Le fichier intermédiaire en assembleur est supprimé une fois cette conversion réalisée. Vous pouvez ici aussi arrêter le processus juste avant le démarrage de l'assembleur afin d'étudier le code assembleur, en spécifiant l'option **-S**.

Pour arrêter le traitement juste avant l'entrée en scène du lieur, vous utilisez l'option **-c** (minuscule). Cette option est souvent utilisée dans les fichiers make pour compiler d'abord chacun des fichiers de code source afin d'effectuer la liaison en une seule étape. Le compilateur possède également des options d'optimisation, de contrôle de la quantité d'avertissements affichés et de spécifications des chemins d'accès aux fichiers d'en-tête. Enfin, l'option **-o** est indispensable car elle sert à spécifier le nom du fichier qui va contenir l'image exécutable. Si vous ne spécifiez rien, ce nom sera *a.out*.

Pour d'autres détails au sujet du compilateur, vous pouvez utiliser les pages d'aide ou récupérer un fichier au format PDF, Postscript ou HTML depuis le site officiel de GNU (gcc.gnu.org/onlinedocs). La densité de cette documentation prouve que GCC n'est pas qu'un simple utilitaire sur ligne de commande. Heureusement, il n'est pas besoin de maîtriser toutes les options pour pouvoir bien s'en servir.

La trousse à outils binutils

La collection binutils regroupe la plupart des utilitaires qui sont requis pour produire l'exécutable à partir du travail du compilateur. Le Tableau 6.2 rappelle le contenu du paquetage binutils-avr pour Linux. Vous y retrouvez la plupart des noms déjà cités dans le Tableau 6.1. Certains de ces outils sont indispensables dans la chaîne de production, et d'autres sont des outils complémentaires, tous orientés vers un microcontrôleur de la famille AVR. La documentation correspondante est disponible sur le site officiel de GNU (bit.ly/gnu-manuals).

Tableau 6. 2 : Outils de la collection binutils pour AVR.

Outil	Description
avr-addr2line	Conversion des adresses en noms de fichiers et numéros de lignes
avr-ar	Génération d'une archive de code objet et modification ou extraction de code
avr-as	Assembleur GNU optimisé pour AVR
avr-c++filt	Déconcentration des symboles assembleurs vers les noms C++
avr-ld	Lieur GNU pour le code objet AVR
avr-nm	Création de la liste des symboles trouvés dans les fichiers objets

avr-objcopy	Conversion de format des fichiers de code objet
avr-objdump	Affichage d'informations au sujet des fichiers objets
avr-ranlib	Génération d'un index pour un fichier archive de librairie
avr-realdef	Affichage d'informations au sujet des fichiers au format binaire ELF
avr-size	Calcul et affichage de la taille des sections et de la taille totale des fichiers objets
avr-strings	Affichage des chaînes affichables trouvées dans les fichiers binaires
avr-strip	Suppression des symboles dans les fichiers objets AVR

Cinq de ces utilitaires sont indispensables pour produire un exécutable AVR : avr-ar, avr-as, avr-ld, avr-objcopy et avr-ranlib. Tous les autres pourront s'avérer utiles à un moment ou un autre lors de vos travaux de développement. Découvrons les cinq outils indispensables.

La création de librairies statiques avec avr-ar

L'outil avr-ar (archive) sert à créer un fichier archive contenant du code objet binaire, c'est-à-dire une librairie statique. Il permet également de modifier une librairie existante ou d'en extraire du code. Ce que l'on appelle fichier de librairie binaire porte en général l'extension de nom de fichier *.a* et il regroupe des modules de code objet et un index en général créé avec l'outil avr-ranlib. Ce genre de librairie est dite statique parce que tous les composants qui sont requis par un autre fichier de code source sont réunis de façon définitive dans cette librairie, et vont être injectés dans le fichier exécutable. L'autre catégorie de librairies est la librairie dynamique partagée. Les microcontrôleurs AVR, ainsi

que n’importe quel microcontrôleur, n’utilisent normalement pas de librairies dynamiques ; je ne vais donc pas les décrire.

L’assembleur avr-as

L’outil avr-as est en fait le compilateur de langage assembleur pour la famille AVR. Il est utilisé automatiquement dans la chaîne d’outils GCC, mais vous pouvez également vous en servir indépendamment, ce que je fais dans une section ultérieure de ce chapitre. Il existe plusieurs types d’assembleurs pour avr, et nous les verrons plus loin, mais seul avr-as est conçu pour fonctionner avec le compilateur GCC/C++.

Le lieur avr-ld

L’outil avr-ld est normalement le dernier à entrer en scène dans la chaîne de création d’un exécutable. Je rappelle que son objectif est de résoudre les références entre les fichiers objets individuels. Il se charge également de repositionner les données dans l’espace mémoire local du fichier en fonction des besoins.

Supposons que dans un fichier de code objet, il se trouve un appel à une fonction qui fait partie d’un autre fichier de code objet. À la fin de la compilation, l’adresse à laquelle doit se poursuivre l’exécution du programme lors de cet appel n’est pas encore connue : c’est une référence non résolue, en attente. Pour illustrer cela, étudions une fonction de la librairie standard *libc*. Supposez que votre programme fasse référence à la fonction de conversion d’ASCII vers entier `atoi()`. Le code source de cette fonction n’est pas inséré dans le vôtre. Une fois la compilation de votre code source réalisée, le fichier objet ne contient pas encore l’adresse à l’endroit où se trouve le début du code exécutable de la fonction appelée. C’est le lieur qui va détecter ce manque. Il va trouver l’adresse de la fonction standard `atoi()`dans la librairie *avr-libc.a*, ajouter cette adresse dans le code et ajouter le code objet de la fonction `atoi()`, le tout constituant le fichier exécutable binaire.

Le convertisseur de format avr-objcopy

Malgré son nom, cet outil ne fait pas que copier le contenu d’un fichier objet : il convertit vers un autre format, et notamment vers le format Intel Hex reconnu

par les microcontrôleurs AVR qui utilisent l'amorceur/chargeur Arduino. Le même outil sait produire d'autres formats, et notamment le format S-record utilisé par les microcontrôleurs de la famille Motorola Freescale.

L'extrait qui suit montre à quoi ressemble un fichier au format Intel Hex, et montre les premières et dernières lignes du fichier de l'amorceur Arduino d'un microcontrôleur ATMega168 ou 328 :

```
:107800000C94343C0C94513C0C94513C0C94513CE1
:107810000C94513C0C94513C0C94513C0C94513CB4
:107820000C94513C0C94513C0C94513C0C94513CA4
:107830000C94513C0C94513C0C94513C0C94513C94
:107840000C94513C0C94513C0C94513C0C94513C84
:107850000C94513C0C94513C0C94513C0C94513C74
:107860000C94513C0C94513C11241FBECFEFD8E036
<em>...autres données ici...</em>
:107F700009F0C7CF103011F00296E5CF112480919F
:107F8000C00085FFB9CEBCCE8EE10E94C73CA2CD19
:0C7F900085E90E94C73C9ECDF894FFCF0D
:027F9C00800063
:040000030000780081
:00000001FF
```

La version complète du fichier est consultable dans le sous-répertoire suivant sous Linux :

```
/usr/share/arduino/hardware/arduino/bootloaders
```

Sous Windows, le fichier se trouve à cet endroit :

```
C:\Program Files\Arduino\hardware\arduino\avr\bootloaders
```

Chaque ligne de ce fichier est constituée ainsi :

- Un symbole de début de ligne qui est ici le signe deux-points.
- Un compteur d'octets qui est pour presque toutes les lignes la valeur 10 hexa, ce qui signifie 16.
- L'adresse sur 16 bits soit quatre chiffres hexa à laquelle le code doit être écrit dans la mémoire flash du microcontrôleur (l'amorceur peut modifier

cela si nécessaire).

- Un code de type d'enregistrement sur deux chiffres.
- Le code exécutable sous forme de caractères hexa en ASCII, à raison de deux caractères par octet, ce qui fait bien 16 octets comme annoncé en début de ligne.
- Enfin, sur deux chiffres, une somme de contrôle en fin de ligne (*checksum*).

Le site Wikipédia donne des détails au sujet de ce format (fr.wikipedia.org/wiki/HEX_(Intel)). Cela dit, vous aurez rarement besoin d'aller autant dans les détails.

Voici comment utiliser l'outil pour convertir un exécutable binaire vers le format Intel Hex :

```
avr-objcopy -O ihex execpgm execpgm.hex
```

Comme quasiment tous les outils de GNU, avr-objcopy offre bien d'autres possibilités et de nombreuses options sur ligne de commande, mais vous ne vous servirez pas de la plupart d'entre elles. Un manuel en ligne pour l'outil est disponible sur le site de SourceWare (bit.ly/sw-objcopy).

L'indexation de fichiers archives avec avr-ranlib

L'outil avr-ranlib a pour seul objectif de générer un index qui est ensuite injecté dans un fichier archive d'objets binaires, ce qui permet d'accélérer la phase de liaison. En effet, le lieur va utiliser cet index pour trouver chaque référence en attente de résolution dans les fichiers objets. Lorsque l'index n'est pas présent, le lieur doit scruter le fichier de librairie objet par objet pour trouver les points qu'il doit traiter, ce qui est beaucoup plus long.

La librairie d'exécution avr-libc

Comme son nom l'indique, avr-libc est la version pour AVR de la librairie d'exécution standard du C/C++. Elle constitue le noyau de la chaîne d'outils

GNU en combinaison avec avr-gcc et binutils.

Toutes les librairies externes standard fournies avec la chaîne d'outils AVR, et c'est le cas d'avr-libc, doivent se trouver dans un répertoire prédéfini. Sous Linux, il peut s'agir de */usr/lib/avr/ lib/* ou de */usr/local/avr/lib*, selon la configuration de votre système et la façon dont le compilateur a été produit ou installé. Les autres librairies externes peuvent se trouver dans un sous-répertoire à votre convenance, à condition que le lieur puisse les trouver.

Cette librairie est le seul composant indispensable qui n'est pas automatiquement incorporé à la collection avr-gcc et binutils. C'est une variation de la librairie standard libc contenant des versions adaptées pour AVR de toutes les fonctions standard. Quelques limitations se sont imposées en raison de celles des microcontrôleurs AVR, notamment suite au faible espace mémoire disponible.

Le Tableau 6.3 dresse la courte liste des fichiers d'en-tête standard qui sont fournis dans avr-libc. Si vous avez déjà programmé en C ou C++ sur un ordinateur, ces noms ne vous seront pas inconnus.

Tableau 6.3 : Fichiers d'en-tête communs d'avr-libc.

Nom de fichier	Description
alloca.h	Réserve de l'espace dans le cadre de pile de l'appelant
assert.h	Teste la véracité d'une expression
ctype.h	Gamme de macros de conversion de caractères
errno.h	Définition des codes erreur système
inttypes.h	Conversion entre types entiers
math.h	Fonctions mathématiques élémentaires
setjmp.h	Définition des méthodes goto non locales setjmp() et longjmp()
stdint.h	Définition des types entiers standard.

stdio.h	Fonctions d’entrées-sorties standard
stdlib.h	Utilitaires génériques
string.h	Opérations et outils sur les chaînes de caractères

Viennent s’ajouter à ces fichiers universels une ribambelle de fichiers d’en-tête spécifiques aux microcontrôleurs AVR ; ils sont listés dans le Tableau 6.4. Sous Linux, ces fichiers sont réunis dans le répertoire */usr/lib/avr/include/avr*. Vous y trouverez des définitions de fonctions et de constantes pour la gestion de l’amorçage, les délais des chronomètres, l’accès à la mémoire EEPROM, la configuration des bits fusibles et celle des broches des ports. Certains fichiers servent à gérer les interruptions et les équivalences des entrées-sorties pour certains processeurs spéciaux.

Tableau 6.4 : Fichiers d’en-tête spécifiques AVR fournis dans avr-libc.

Nom de fichier	Description			
boot.h	io90pwm316.h	iom169pa.h	iom32u2.h	iomx8.
builtins.h	io90pwm3b.h	iom169p.h	iom32u4.h	iomxx(
common.h	io90pwm81.h	iom16a.h	iom32u6.h	iomxx4
cpufunc.h	io90pwmx.h	iom16.h	iom406.h	iomxxh
crc16.h	io90scr100.h	iom16hva2.h	iom48.h	iotn10.
delay.h	ioa6289.h	iom16hva.h	iom48p.h	iotn11.
eeprom.h	ioat94k.h	iom16hvb.h	iom640.h	iotn12.
fuse.h	iocan128.h	iom16hvbrevb.h	iom644.h	iotn13a
interrupt.h	iocan32.h	iom16m1.h	iom644pa.h	iotn13.
io1200.h	iocan64.h	iom16u2.h	iom644p.h	iotn15.

io2313.h	iocanxx.h	iom16u4.h	iom6450.h	iotn16
io2323.h	io.h	iom2560.h	iom645.h	iotn20.
io2333.h	iom103.h	iom2561.h	iom6490.h	iotn22.
io2343.h	iom1280.h	iom3000.h	iom649.h	iotn23
io43u32x.h	iom1281.h	iom323.h	iom649p.h	iotn23
io43u35x.h	iom1284p.h	iom324.h	iom64c1.h	iotn24
io4414.h	iom128.h	iom324pa.h	iom64.h	iotn24.
io4433.h	iom128rfa1.h	iom3250.h	iom64hve.h	iotn25.
io4434.h	iom161.h	iom325.h	iom64m1.h	iotn26
io76c711.h	iom162.h	iom328p.h	iom8515.h	iotn26
io8515.h	iom163.h	iom3290.h	iom8535.h	iotn26.
io8534.h	iom164.h	iom329.h	iom88.h	iotn28.
io8535.h	iom165.h	iom32c1.h	iom88pa.h	iotn40.
io86r401.h	iom165p.h	iom32.h	iom88p.h	iotn43
io90pwm1.h	iom168.h	iom32hvb.h	iom8.h	iotn43
io90pwm216.h	iom168p.h	iom32hvbrevb.h	iom8hva.h	iotn44

La librairie avr-libc contient enfin quelques fichiers d'en-tête pour les outils et pour assurer la compatibilité (Tableau 6.5). Notez que les trois fichiers *delay.h*, *crc16.h* et *parity.h* se trouvant dans *avr/* renvoient en fait aux fichiers d'en-tête du sous-répertoire *util/*. Autrement dit, si vous faites référence par exemple à *<avr/parity.h>* dans le code, le fichier lu sera *<util/parity.h>*.

Tableau 6.5 : Fichiers d'en-tête utilitaires et de compatibilité d'avr-libc.

Nom de fichier	Description

util/atomic.h	Gestion de blocs de code exécutable atomiques (série d'instructions ne pouvant être interrompues)
util/crc16.h	Réalisation de calculs de somme de contrôle CRC (Contrôle de Redondance Cyclique)
util/delay.h	Fonctions pour boucles de délai en état occupé
util/delay_basic.h	Définitions diverses pour les boucles de délais en état occupé
util/parity.h	Génération de bits de parité
util/setbaud.h	Macros de support pour les calculs de débit en bauds
util/twi.h	Définitions des masques binaires du protocole TWI sur deux fils
compat/deprecated.h	Éléments dépréciés et obsolètes
compat/ina90.h	Support de compatibilité avec IAREWB 3.x

Pour en savoir plus sur cette librairie, voyez la page de documentation qui se trouve à l'adresse `www.nongnu.org/avr-libc/`. Vous pouvez également parcourir le contenu des fichiers d'en-tête, car vous y trouverez des commentaires au sujet des domaines d'application et des limitations. Je rappelle que les fonctions telles que `malloc()` et `printf()`, même si elles sont utilisables pour AVR, sont très contraintes par le faible espace mémoire d'un microcontrôleur. Les fonctions mathématiques constituent un autre souci, car il n'y a pas d'unité à virgule flottante dans les microcontrôleurs AVR. Vous ne pouvez faire que des opérations sur des valeurs entières ou des valeurs à virgule fixe. Les opérations sur des valeurs à virgule flottante (en notation scientifique), ne sont que simulées, ce qui est très lent, et qu'il faut donc éviter tant que faire se peut.

Il existe une version PDF compressée avec Bzip2 de la documentation de cette librairie à l'adresse `bit.ly/avr-libc-manual`. Notez que la documentation décrit également la chaîne d'outils avr-gcc.

Production C ou C++ avec avr-gcc

Vous pouvez tout à fait produire une application sans utiliser du tout l'atelier Arduino. Vous pouvez même vous passer de la chaîne d'outils avr-gcc et de ses librairies si vous avez une solution de remplacement valable. En règle générale, les programmeurs produisent une version initiale de leur programme avec tout le confort de l'atelier Arduino puis utilisent isolément les outils dont ils ont besoin pour optimiser ou personnaliser certaines portions ou certains modules.

Compilation avec avr-gcc ou avr-g++

Le fonctionnement détaillé du compilateur n'est pas une information indispensable, mais si vous êtes curieux, vous pouvez en savoir plus en vous rendant à la page gcc.gnu.org/onlinedocs/gccint.

Les deux outils avr-gcc et avr-g++ sont particuliers, dans le sens où ils savent enchaîner la séquence d'outils qui succèdent à leur propre travail. Ils ne se limitent pas à produire le code objet intermédiaire. Voici un exemple de commande :

```
avr-gcc -mmcu avr5 -o test test_src.c -L../avrlibs -
lruntime
```

Cette commande demande d'utiliser en entrée le fichier source nommé *test_src.c*, de le compiler pour le contrôleur du type ATmega32U4 (option **-mmcu**), de stocker le résultat dans un fichier nommé *test* (option **-o**) et de lancer le lieur en lui demandant d'aller chercher une librairie dans le sous-répertoire *avrlibs*.

Vous pouvez demander au compilateur de s'arrêter juste après avoir réalisé la compilation en ajoutant l'option **-c**. Cette option est très utilisée dans les fichiers make pour compiler chacun des fichiers source puis demander au lieur de produire le fichier exécutable à partir de toute la série de fichiers de code objet, en y ajoutant les librairies externes. Voici la commande correspondante :

```
avr-gcc -mmcu avr5 -c test_src.c
```

Les nombreuses options du compilateur concernent notamment l'optimisation, la densité des messages et les chemins d'accès complémentaires pour les fichiers d'en-tête.

Structure d'un fichier de production make

Dès que vous travaillez avec plusieurs fichiers source, il faut recourir à l'automate make. Vous pouvez apprendre à rédiger vous-même le contenu d'un fichier make comme je le fais, ou bien vous aider d'un outil tel qu'arduino-mk.

L'outil arduino-mk

Cet outil sert à créer des fichiers de production make prédéfinis ; il gère toute la logique qui permet de produire un exécutable Arduino en utilisant la chaîne d'outils avr-gcc.

Vous trouverez arduino-mk sur le site de référence GitHub. Sous Linux, il est souvent disponible sous forme de paquetage. Si vous l'installez avec un gestionnaire de paquetages, vous trouverez normalement le fichier principal dans le même répertoire que les composants de l'atelier IDE Arduino. Le site de l'outil est à l'adresse `bit.ly/hf-makefile` et l'archive compressée ZIP à l'adresse `bit.ly/gh-makefile`. Une fois l'outil installé, il vous faudra sans doute ajouter aussi quelques fonctions de support et définir des variables globales d'environnement. La documentation sur le site Hardware Fun donne tous les détails.

Le premier contact avec le contenu d'un fichier d'automatisation make peut sembler repoussant, car c'est une structure très particulière. Je pense néanmoins que l'effort en vaut la chandelle, car cela vous libère de toute dépendance par rapport à un atelier de développement et vous permet d'entrer pleinement dans le monde du développement de systèmes embarqués. D'ailleurs, tous les microcontrôleurs ne disposent pas d'un atelier graphique pour produire les exécutables, et les ateliers n'offrent pas la souplesse que procure un contrôle direct de chacune des étapes du processus de production.

Je vous propose de repartir de l'exemple d'alarme du Listing 5.5 pour produire un fichier make afin d'automatiser sa production. Nous fournissons notre propre

fonction principale `main()` dans le fichier *main.cpp*, les deux fonctions obligatoires `setup()` et `loop()` dans le fichier *alarm.c*, les variables globales dans le fichier *globals.c* et les directives `#define` dans le fichier *defs.h*. Voici à quoi ressemble notre fichier make :

Listing 6.1 : Exemple de fichier de production make.

```
CC = avr-gcc
LD = avr-ld
OC = avr-objcopy

SRCS = main.c alarmloop.c globals.c
HDRS = main.h globals.h defs.h
OBJS = main.o alarmloop.o globals.o
EXEC = alarm
TARGET = alarm.hex

$(TARGET): $(OBJS)
    $(LD) -o $(EXEC) $(OBJS)
    $(OC) -O ihex $(EXEC) $(TARGET)

main.o: main.c $(HDRS)
    $(CC) -mmcu avr5 -Wall -c main.c
alarmloop.o: alarmloop.c $(HDRS)
    $(CC) -mmcu avr5 -Wall -c alarmloop.c

globals.o: globals.c
    $(CC) -mmcu avr5 -Wall -c globals.c
```

Le répertoire contenant ce fichier nommé *Makefile* devrait au minimum contenir les fichiers suivants :

```
Makefile
main.c
main.h
alarmloop.c
alarmloop.h
globals.c
globals.h
```

Il suffit de saisir la commande magique **make** (sans aucun paramètre) pour lancer la production. En effet, l'outil va chercher dans le répertoire actuel un fichier portant exactement le nom *Makefile* (avec un M majuscule). Il va trouver dans ce fichier tout ce qu'il faut pour produire le nouvel exécutable si nécessaire. Vous pouvez utiliser un autre nom pour votre fichier make en précisant ce nom avec l'option **-F**, comme ceci :

```
make – F make_alarme
```

Étudions les détails de notre fichier make. Les trois premières lignes sont des alias pour les outils, pour éviter de les répéter dans la suite. Viennent ensuite d'autres alias qui sont remplacés par tout ce qui se trouve après le signe égal. Tout cela évite de répéter la saisie, source d'erreurs. Les alias permettent également de centraliser les noms de fichiers. Nous arrivons ensuite à la partie fondamentale qui est constituée de quatre règles en ordre chronologique inverse. Le format de chaque règle est le suivant :

```
cible: dépendance

        actions
```

Chaque action doit commencer après un pas de tabulation pour que l'automate comprenne que la ligne dépend de la précédente. Il ne faut pas utiliser d'espaces. Le nombre d'actions qui dépendent d'une règle est *a priori* illimité. L'essentiel est que chaque action soit sur sa ligne et qu'elle commence avec une indentation par tabulation.

L'outil make évalue chaque règle en regardant les dépendances, c'est-à-dire les noms des fichiers qui participent à la production de l'étape correspondante. Dans le cas de la cible *main.o*, la règle demande d'utiliser d'une part le fichier source *main.c*, et d'autre part la série de fichiers d'en-tête représentée par l'alias $(HDRS) (abréviation de *headers*). Si le fichier cible n'existe pas, il est systématiquement reconstruit. S'il existe, l'outil make va vérifier la date et l'heure de chacun des fichiers mentionnés comme dépendances et recompiler si l'un quelconque de ces fichiers possède une date et une heure plus récentes que le fichier qui en résulte. La première règle utilise un alias pour la cible $(TARGET) qui comporte deux actions. La liaison et la conversion de formats sont déclenchées dès qu'un des fichiers d'entrée, c'est-à-dire un des fichiers de code objet, a changé.

Si vous modifiez par exemple le contenu du fichier *alarmloop.c* puis relancez la commande make, l'outil va recompiler ce fichier, ce qui va produire une nouvelle version d'*alarmloop.o*. L'outil va alors reconstruire le fichier symbolisé par $(TARGET) puisqu'un des fichiers d'entrée est plus récent. Si vous modifiez le fichier *alarmloop.h*, make va recompiler *main.c* et *alarmloop.c* avant de reconstruire l'exécutable.

La rédaction du fichier make demande beaucoup de concentration, mais une fois que vous l'avez écrit et testé, vous n'aurez que rarement besoin d'y revenir, sauf pour ajouter un nouveau module de code source ou retoucher une option.

Le langage assembleur AVR

Les programmeurs qui ont vraiment besoin de tirer le maximum de performances d'un microcontrôleur AVR se tourneront vers le langage assembleur. Cette plongée dans les entrailles d'un processeur est déconseillée aux dilettantes, mais ce n'est qu'ainsi que vous avez un contrôle total de chaque action que va réaliser le microcontrôleur, et à quel moment il va la réaliser. Descendre ainsi dans les détails est parfois inévitable, surtout dans un environnement qui offre un espace mémoire limité. Certains modèles AVR, et notamment la série ATtiny, offrent tellement peu d'espace mémoire que la programmation en assembleur devient rapidement la seule solution, car un programme écrit en C ou C++ va résulter en un code exécutable trop volumineux pour être implanté en mémoire en laissant suffisamment d'espace pour le traitement des données.

Dans cette section, je vous propose un tour d'horizon vraiment rapide de la programmation en langage assembleur AVR. J'ai prévenu dès la préface que ce livre n'était pas un tutoriel de programmation, mais une référence orientée vers le matériel pour Arduino. L'objectif de cette section est de vous donner un aperçu de ce que cela suppose de programmer en assembleur, ce qui vous permettra de décider d'y aller ou pas. Des livres entiers sont consacrés à la programmation en assembleur, et vous trouverez suffisamment de ressources sur le Web. Je vous invite à commencer par vous tourner vers l'annexe D. Je termine cette section sur l'assembleur avec quelques conseils pour trouver des informations.

Le langage assembleur est le plus proche qui soit du niveau matériel, et c'est pourquoi on l'appelle également langage machine. En réalité, le langage machine est la séquence de codes binaires des instructions exécutables par le processeur. Le langage assembleur est une version plus digeste pour un humain, grâce à l'utilisation de symboles. Le but de l'outil appelé lui aussi assembleur est d'assurer la conversion depuis la version lisible par un humain vers le code machine, tout en tirant profit de quelques fonctions pratiques comme les macros, les noms symboliques, les directives conditionnelles et même la possibilité de faire référence à des fonctions se trouvant dans des librairies externes.

Une instruction en langage assembleur s'écrit par exemple comme ceci :

```
MOV R6, R7
```

Cette instruction demande de copier (MOV) le contenu du registre R7 dans le contenu du registre R6 (notez bien l'inversion). Ce que va produire l'assembleur est une suite de valeurs binaires que le microcontrôleur sait exécuter. C'est une instruction machine. Les noms que vous citez dans le code source assembleur sont des mnémoniques qui correspondent directement à des valeurs d'actions élémentaires réalisées par le microcontrôleur. Chaque mnémonique est suivie par un ou plusieurs opérandes, qui sont ici les noms des deux registres.

Programmer en assembleur revient donc à guider étape par étape le microcontrôleur pour le faire basculer d'un état à un autre. Dans un langage comme le C, tous les détails sont masqués, et le programmeur n'a pas besoin de décider dans quel registre il faut stocker telle valeur et quel bit d'état il doit vérifier après chaque action. En assembleur, il n'y a plus d'écran entre la pensée du programmeur et la logique électrique du processeur. Même la plus simple opération doit être décrite en détail.

Le modèle de programmation AVR

Les composants principaux d'un microcontrôleur AVR sont le noyau AVR, la zone de mémoire flash, la zone de mémoire EEPROM et les fonctions périphériques. Le noyau réunit un registre d'instruction, un décodeur, un registre pointeur d'instruction (compteur de programme), un peu de mémoire dynamique RAM, des bits d'état, 32 registres à usage général (de petits espaces mémoire) et enfin une unité de traitement logique et arithmétique, plus souvent appelée ALU (*Arithmetic and Logic Unit*). C'est dans cette ALU que sont réalisés les traitements. La Figure 6.5, reprend la Figure 2.1 pour illustrer les constituants d'un microcontrôleur AVR. J'ai donné tous les détails des fonctions internes dans les Chapitres 2 et 3.

Les instructions traitent les données qui peuvent se trouver dans les registres ou dans la mémoire. Les données peuvent être copiées d'un registre vers un autre ; les contenus de deux registres peuvent être comparés, échangés, additionnés, soustraits, multipliés, divisés, parmi d'autres opérations. Des données peuvent être lues depuis la mémoire flash, depuis la mémoire EEPROM ou depuis la mémoire RAM. Des registres servent à contrôler et à accéder aux périphériques, mais ce ne sont pas les mêmes que les 32 registres à usage général. Les trois

caractéristiques essentielles d'un microcontrôleur sont la mémoire, les instructions et les registres. Décrivons-les.

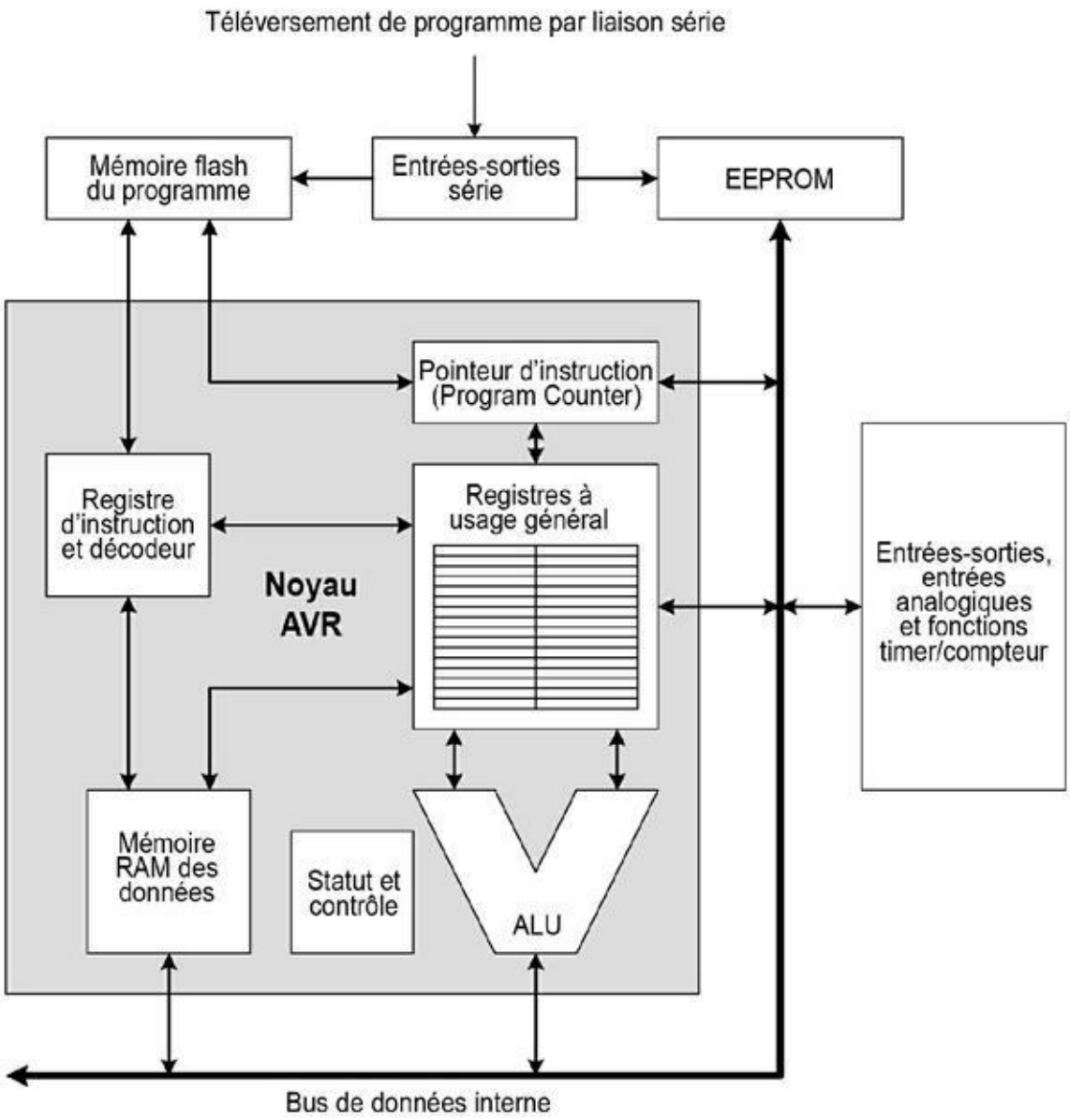


Figure 6.5 : Diagramme d'un microcontrôleur AVR.

Organisation de la mémoire

Comme déjà décrits dans le Chapitre 2, les microcontrôleurs AVR d'Atmel ont été conçus selon l'architecture dite Harvard dans laquelle le code exécutable est stocké dans une mémoire en lecture seule (ici de la mémoire flash) et les

données variables dans un autre espace mémoire, qui est la mémoire SRAM dans les AVR. Les microprocesseurs des ordinateurs habituels sont construits selon une autre architecture, Von Neumann, dans laquelle l'exécutable et les données sont tous deux dans le même espace mémoire.

Dans un processeur AVR, les registres à usage général et tous les registres des entrées-sorties pour les périphériques font partie de l'espace mémoire en lecture/écriture. La Figure 6.6 montre comment est structuré l'espace mémoire d'un AVR.

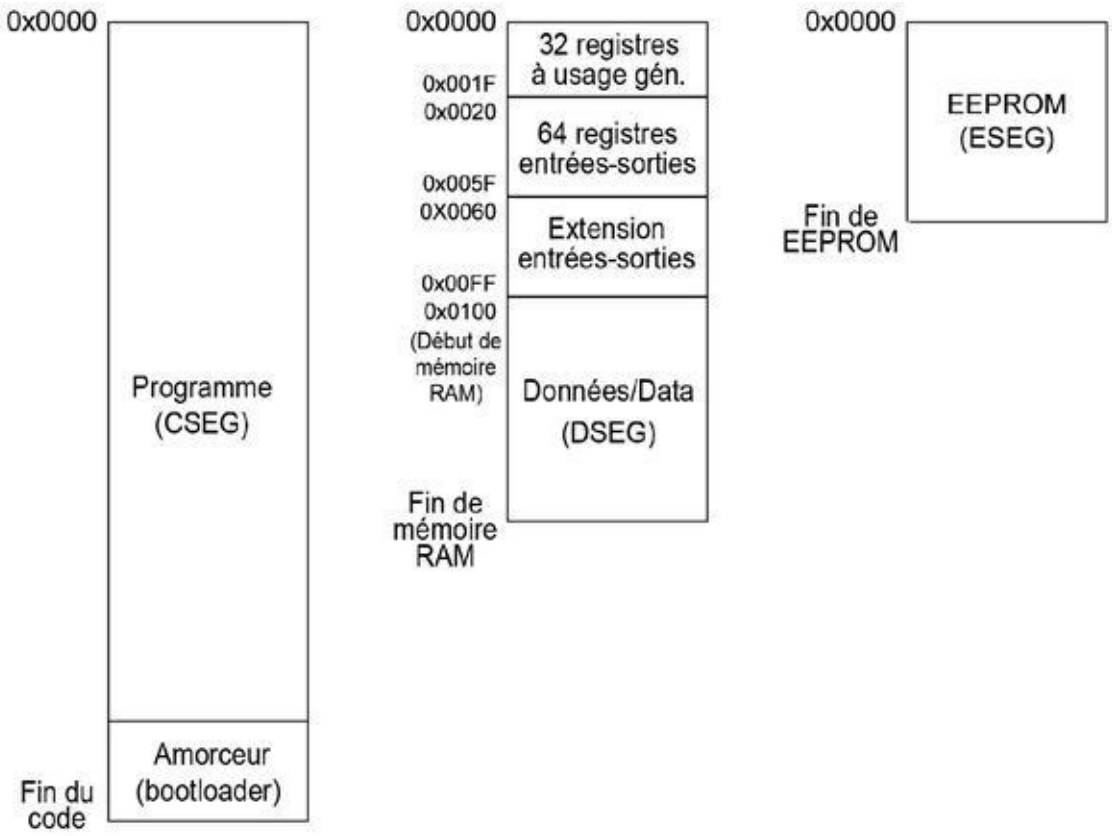


Figure 6.6 : Structure de l'espace de stockage mémoire d'un microcontrôleur AVR.

Cette figure est volontairement simplifiée. Les trois directives d'assembleur nommées CSEG, DSEG et ESEG (que reconnaît l'assembleur) font référence respectivement au segment de mémoire pour le code exécutable, les données variables et la mémoire EEPROM. La plus grande adresse, soit l'adresse finale, pour chacun des trois segments dépend de la quantité de mémoire, donc du modèle de microcontrôleur. Vous remarquez qu'un espace a été réservé à la fin du code pour y implanter un amorceur/bootloader.

Traitement des instructions

Les microcontrôleurs AVR fonctionnent avec un canal d'alimentation d'instruction unique (*single-level pipeline*). Ce canal sert à récupérer, décoder puis exécuter les instructions en puisant dans la mémoire flash. Pendant qu'une instruction est en cours d'exécution, la suivante est lue de la mémoire afin qu'elle soit prête à être décodée et exécutée dès que l'instruction en cours sera terminée. Ce mode d'organisation, couplé au jeu d'instructions réduit RISC choisi par Atmel, permet au microcontrôleur AVR d'exécuter la plupart de ses instructions en un seul cycle d'horloge.

Registres

La plupart des instructions AVR se terminent en modifiant l'état d'un ou de plusieurs bits dans un registre spécial, le registre d'état sur 8 bits SREG. Chaque bit possède une signification particulière, comme le montre le Tableau 6.6. La valeur de chaque bit (soit 0, soit 1) est testée par certaines instructions comme BREQ (*BRanch if EQual*) ou BRNE (*BRanch if Not Equal*) juste après une instruction de comparaison CP. Cette dernière peut modifier un ou plusieurs des six bits Z, C, N, V, H ou S.

Tableau 6.6 : Bits d'état du registre SREG.

Bit	Symbole	Fonction
0	C	Drapeau de retenue (Carry)
1	Z	Drapeau de Zéro
2	N	Drapeau de valeur Négative
3	V	Indicateur de débordement de complément à 2 (oVerflow)
4	S	Drapeau pour les tests de Signe
5	H	Drapeau de demi-retenue (Half)
6	T	Bit de Transfert utilisé par BLD et BST

Les 32 registres à usage général ont chacun une taille de 8 bits. Les trois dernières paires de registres, les couples 26-27, 28-29 et 30-31 sont utilisables sous forme de registres d’index sur 16 bits pour les adresses indirectes. Ils portent dans ce cas les noms de pointeurs X, Y et Z, respectivement.

Quelques instructions travaillent sur deux registres 8 bits comme s’il s’agissait d’un seul registre 16 bits. Celui des deux registres qui porte le plus petit numéro contient les bits les moins significatifs (bits de poids faible). Du fait que la numérotation commence à zéro, ce registre de poids faible doit porter un numéro pair. Désigner la paire 0-1 est autorisé, mais 1-2 est impossible.

En plus des registres d’état, des registres à usage général et de ceux des entrées-sorties, un microcontrôleur AVR possède, comme tout processeur, un pointeur d’instruction et un pointeur de pile. Le premier s’appelle en anglais *Program Counter*, PC, et le pointeur de pile s’appelle *Stack Pointer*, SP. Le contenu de ces deux registres change sous l’effet de certaines instructions. Prenons l’exemple classique d’un appel de fonction : il modifie les deux registres PC et SP. L’appel de fonction utilise l’instruction assembleur CALL. Lors de l’exécution de cette instruction, l’adresse que contient actuellement le registre PC est remplacée par l’adresse dans le même code source qui suit l’instruction CALL, puis le contenu de ce registre est sauvegardé en haut de la zone de pile, une zone mémoire spéciale. L’instruction écrit alors dans le registre PC l’adresse de début d’exécution de la fonction que vous demandez d’exécuter. Lorsque cette fonction, ou routine, se termine, par l’instruction de retour RET, l’adresse du PC qui était sauvegardée dans la pile est dépilée et replacée dans le registre PC, ce qui permet à l’exécution de se poursuivre à l’instruction située juste après l’appel CALL.

Je n’ai évidemment fait qu’effleurer le sujet. Il y a bien d’autres choses à dire au sujet de l’assembleur AVR. Si vous avez envie ou besoin d’en savoir plus, je vous conseille de choisir l’un des livres dédiés à ce sujet dans l’Annexe D.

N. d.T. : Notez qu’il existe très peu de livres en français consacrés au langage assembleur.

Structure d’un programme assembleur AVR

Dans un fichier de code source assembleur, vous pouvez insérer des commentaires sur une ligne en faisant commencer la ligne par un signe deux-points (:) ou un signe dièse (#). L’assembleur GNU AVR accepte également les commentaires sur plusieurs lignes en utilisant la même syntaxe que celle du langage C : le couple /* pour débuter la zone de commentaires et le couple */ pour la clore.

J’aborde d’abord les commentaires, parce qu’ils sont indispensables en langage assembleur. Prenons comme exemple le très petit code source assembleur suivant :

Listing 6.2 : Exemple de code source assembleur non commenté.

```
LOOP:
LDI R16, 0X00
OUT PORTB, R16
LDI R16, 0XFF
OUT PORTB, R16
RJMP LOOP
```

Que fait ce programme ?, Ici, ce n’est pas trop difficile : la deuxième ligne demande d’écrire (de charger, *load*) dans le registre R16 la valeur hexa 0. De même, dans la quatrième ligne, nous chargeons dans le même registre la valeur 255, soit 0XFF en hexadécimal. Après chaque chargement, nous transférons la valeur que contient le registre vers le port B. Le résultat est que les broches du port B vont passer de l’état Haut à l’état Bas aussi vite qu’il est possible au contrôleur d’exécuter la boucle.

Voici maintenant le même programme quelque peu enrichi, en décidant de l’origine du programme en mémoire et en initialisant le port. Surtout, nous ajoutons des commentaires :

Listing 6.3 : Le même code source avec des commentaires et quelques aménagements.

```
; Configure le vecteur power-up/reset
.ORG 0X0000
RJMP MAIN
; Point d'entrée du code
.ORG 0X0100
MAIN:
LDI R16, 0XFF ; Charge dans R16 la valeur 0b11111111
OUT DDRB, R16 ; Configure sens du port B (Data Direction
Register)

; Boucle infinie - inverse broches du port B (on/off)
LOOP:
LDI R16, 0X00 ; Charge dans R16 la valeur 0
OUT PORTB, R16 ; Envoie vers B
LDI R16, 0XFF ; Charge dans R16 la valeur 0xFF
OUT PORTB, R16 ; Envoie vers B
RJMP LOOP ; Saute au label LOOP pour recommencer
```

Dans cet exemple, j'ai vraiment commenté toutes les actions. En assembleur, il n'est pas rare de rencontrer ainsi un commentaire sur presque toutes les lignes. Je rappelle qu'une opération aussi simple que l'affichage d'un message de bienvenue réclame un nombre d'instructions assembleur bien supérieur à ce qu'il faut écrire en langage C ou C++. De plus, certaines instructions assembleur ne sont pas compréhensibles sans explications.

L'outil assembleur reconnaît en général un certain nombre de mots spéciaux appelés *directives*. Dans le précédent exemple, nous avions déjà la directive .ORG. Le Tableau 6.7 propose une liste de directives. On les appelle directives parce qu'elles dirigent l'assembleur pour qu'il réalise des associations entre symboles ou certaines actions.

Tableau 6.7 : Directives de l'assembleur AVR.

Directive	Opération
BYTE	Réserve un ou plusieurs octets pour une variable
CSEG	Fait utiliser le segment de code
CSEGSIZE	Définit la taille du segment de code en mémoire

DB	Définit une ou plusieurs constantes de taille octet
DEF	Définit un nom symbolique pour un registre
DEVICE	Définit la cible de l’assembleur
DESG	Fait utiliser le segment de données
DW	Définit une ou des constantes de taille mot (un mot pèse 16 bits)
END, ENDMACRO	Marque la fin d’une définition de macro
EQU	Associe un symbole à une expression
ESEG	Fait utiliser le segment de la mémoire EEPROM
EXIT	Provoque la sortie du fichier (fin d’exécution)
INCLUDE	Lit puis injecte un code source d’un autre fichier
LIST	Active la génération d’un fichier de liste
LISTMAC	Autorise l’expansion des macros dans le fichier de liste
MACRO	Marque le début d’une définition de macro
NOLIST	Désactive la génération du fichier de liste
ORG	Définit l’origine du programme
SET	Affecte un symbole à une expression

L’extrait suivant illustre l’utilisation de quelques directives.

Listing 6.4 : Utilisation de directives d’assembleur.

```
; Désactive la génération de liste pour l’inclusion des
```

```
fichiers externes.
; Évite ainsi d'allonger la liste.

.NOLIST
.INCLUDE "macrodefs.inc"
.INCLUDE "mcutype.inc"
.LIST

; Définit une macro réalisant une action utile
.MACRO SUBI16
   SUBI @1,low(@0)  ; Soustrait l'octet faible (LSB)
   SBCI @2,high(@0) ; Soustrait l'octet fort (MSB)
.ENDMACRO

; Pour tester, soustrait 0x2200 du contenu de R17:R16
SUBI16 0x2200,R16,R17
```

Sources d'information sur l'assembleur AVR

Voici quelques suggestions pour ceux qui ont envie d'en apprendre plus au sujet du langage assembleur des microcontrôleurs AVR.

- La société Atmel possède une page de référence sur l'assembleur AVR avec une description et un guide d'utilisateur en anglais (bit.ly/avr-assembler-guide). Sachez que ce n'est pas le même assembleur qu'avr-as, mais les grandes lignes sont identiques.

- La description de l'assembleur avr-as de GNU est accessible à la page bit.ly/gnu-ma-nuals. Vous y trouverez aussi une description de la librairie avr-libc.

- Le programmeur allemand Gerhard Schmidt a réuni beaucoup d'informations en anglais et en allemand sur son site Web, au format PDF notamment (bit.ly/avr-overview).

- Le site Web de l'université John Hopkins offre une synthèse du langage assembleur (bit.ly/jhu-assembler).

- N’hésitez pas également à consulter l’Annexe D.

Téléversement du code exécutable

Une fois que vous avez compilé, assemblé, lié et converti votre projet pour obtenir un fichier exécutable, il reste à transférer ce fichier dans la mémoire du microcontrôleur. Plusieurs techniques sont possibles, parmi lesquelles l'utilisation de l'amorceur/chargeur, l'interface ISP que possèdent certaines cartes Arduino et l'interface normalisée JTAG.

Programmation in-situ ISP (*In System Programming*)

La note technique AVR910 décrit la technique nommée ISP comme une interface pour reprogrammer un microcontrôleur AVR. Elle correspond au connecteur ICSP des cartes Arduino qui est une extension de l'interface série SPI. Dans la Figure 6.5, cette fonction d'entrées-sorties série a fait l'objet d'un bloc séparé parce qu'elle permet de communiquer directement avec la mémoire flash et EEPROM du microcontrôleur.

Durée de vie d'une mémoire flash

La technologie de stockage flash n'autorise qu'un nombre limité d'opérations d'écriture. Dans les composants actuels, ce nombre est de l'ordre de 10 000 cycles d'écriture. Cela peut vous sembler peu, lorsque vous songez au nombre d'écritures qui sont réalisées sur un disque dur d'ordinateur de bureau. Bien que la mémoire flash puisse être considérée comme un disque dur (les clés USB sont de la mémoire flash), la zone de mémoire flash du microcontrôleur n'est pas utilisée comme un disque dur. Elle sert à écrire l'exécutable du programme, mais surtout pas à stocker les données qui varient pendant l'exécution. Dans un microcontrôleur, les données sont lues et écrites dans une autre mémoire, la mémoire statique SRAM (et parfois une petite puce de mémoire microSD spécifique d'une durée de vie d'environ 100 000 cycles d'écriture ; ce qui permet de stocker un gros volume de données pour l'exécution). Même si vous mettez à jour l'exécutable dans la mémoire flash

une fois par jour, il vous faudrait de quatre à cinq ans pour qu'elle commence à montrer des signes d'erreur.

La Figure 6.7 montre le brochage du connecteur ICSP principal d'une carte Arduino Uno R2. Notez qu'il existe un deuxième connecteur ICSP, qui sert à gérer l'interface USB pour le microcontrôleur. Le câblage est le même, mais il n'y a aucune raison d'intervenir à ce niveau à moins de vouloir prendre le risque de perdre l'utilisation de la connexion USB. Atmel a choisi un autre connecteur, sur dix broches, mais il n'est pas utilisé dans les cartes Arduino. Ce connecteur a le même brochage que celui à six broches, avec des connexions à la masse pour les quatre broches restantes.

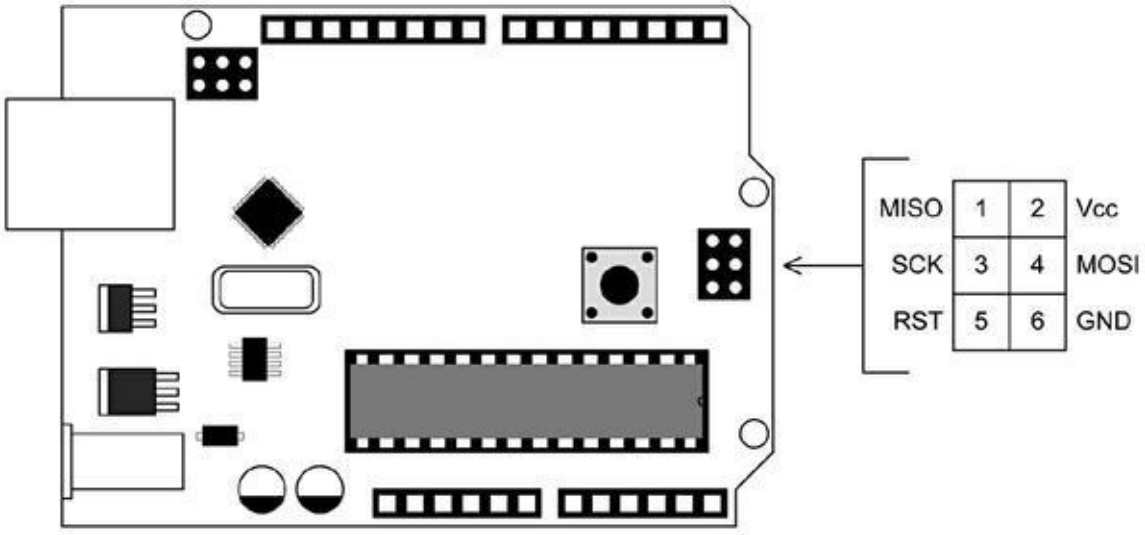


Figure 6.7 : Connecteur ICSP d'une carte Arduino Uno R2.

Dans la description de l'outil de conversion avr-objcopy du même chapitre, j'ai expliqué que l'outil servait à convertir une image binaire exécutable vers un format ASCII spécial Intel Hex. L'avantage de ce format Intel est de permettre d'y définir les segments et les adresses, et d'ajouter un marqueur de fin de fichier. Ce marqueur permet de transférer les données sans risquer les problèmes qui peuvent se poser avec des données purement binaires, c'est-à-dire sans aucun contrôle sur ce qui a ou non été reçu. Le microcontrôleur va implanter le code dans la mémoire flash dans un format binaire, mais le logiciel qui réalise l'implantation dispose d'énormément de contrôle sur le processus exact. Le fichier au format Intel Hex est exploité par le composant qui effectue l'implantation, pas par le microcontrôleur.

Programmation avec le chargeur/amorceur (*bootloader*)

Dans le cas des cartes Arduino, l'élément qui effectue l'implantation, et que j'appellerai le programmateur, est incarné par le logiciel amorceur, lui-même implanté dans la mémoire flash. C'est ce qui permet au microcontrôleur de réaliser lui-même l'implantation du programme qu'il va exécuter, en échange de l'occupation d'une petite portion de la mémoire flash, pour y stocker l'amorceur. Sans cet amorceur, il faut un autre moyen d'implanter définitivement les données avec les bonnes adresses cibles, et de configurer les bits fusibles du microcontrôleur, parmi d'autres détails de préparation. J'ai présenté les bits fusibles des microcontrôleurs AVR dans la fin du Chapitre 3.

Le microgiciel amorceur Arduino utilise les broches série RxD et TxD, c'est-à-dire les deux broches D0 et D1. Vous pouvez donc soit utiliser un convertisseur vers le RS-232 pour programmer le microcontrôleur, soit prendre un adaptateur USB vers série tel que le module SparkFun de la Figure 6.8. Certaines cartes Arduino utilisent pour l'interface USB un circuit FTDI FT232L, un ATmega8 ou un ATmega16U2. La lecture du schéma correspondant montre que le circuit d'interface utilise lui aussi les deux broches D0 et D1 avec des résistances de 1 kilo-ohm. Le signal DTR sert à provoquer une réinitialisation du microcontrôleur principal.

Figure 6.8 : Adaptateur USB vers série de SparkFun.

Vous pouvez néanmoins continuer à utiliser les broches D0 et D1, si elles sont isolées correctement. La Figure 6.9 est un schéma partiel qui montre comment connecter un circuit FTDI FT232L à un microcontrôleur AVR. Notez que ce montage ne convient pas à la carte Leonardo ni aux autres cartes équipées d'un ATmega32U4, car elles disposent déjà en interne d'une interface USB.

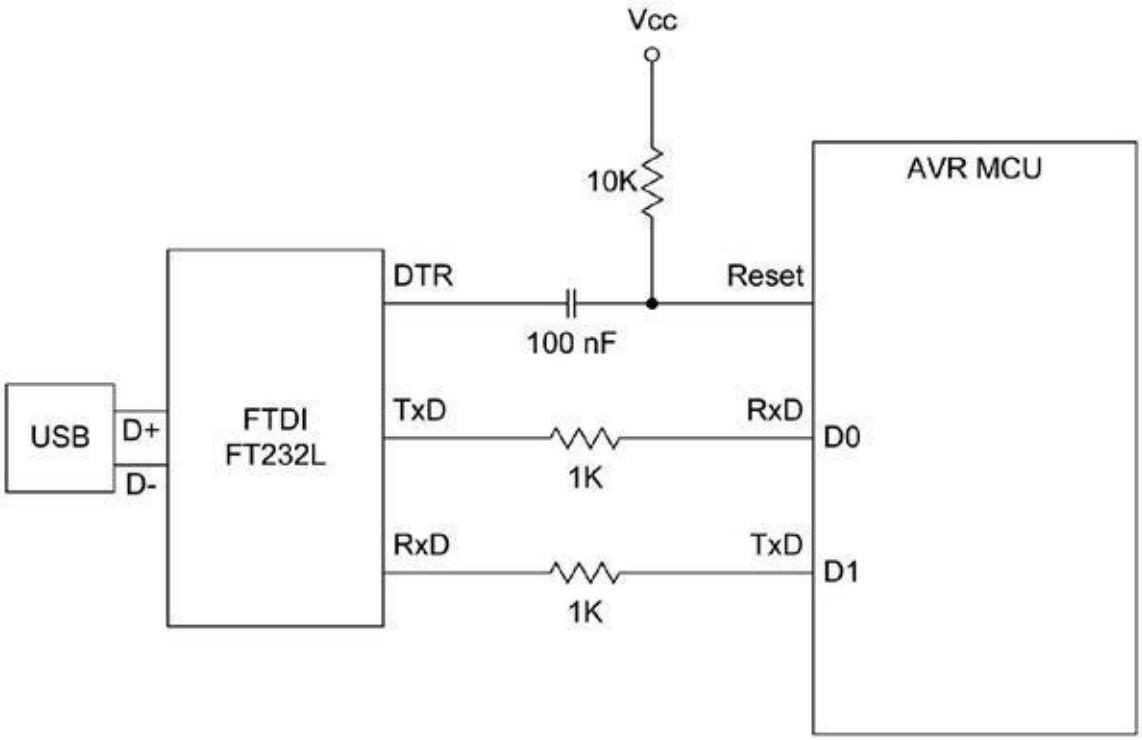


Figure 6.9 : Interface USB basée sur un circuit de conversion FTDI.

Téléversement sans l'amorceur

Lorsque vous avez besoin d'absolument tout l'espace mémoire disponible dans la mémoire flash, ou lorsque vous ne voulez pas utiliser les deux broches D0 et D1 pour téléverser le programme, vous pouvez le transférer par l'interface ICSP. Cette technique convient à un microcontrôleur tel qu'il sort d'usine, mais également à une carte Arduino.

Ce téléversement sans utiliser l'atelier Arduino demande de recourir à l'outil avrdude décrit un peu plus loin. En général, vous en profiterez pour reconstruire un fichier de production make afin d'automatiser la totalité du processus.

Vous pouvez également continuer à utiliser l'atelier Arduino pour prendre en charge la partie compilation. En effet, l'atelier supporte une programmation

directe en permettant de sélectionner un programmateur dans le menu **Outils/Programmateur**. Vous pouvez ensuite lancer le téléversement grâce à la commande **Téléverser avec un programmateur** du menu **Fichier**. Si après avoir réalisé cette opération, vous avez effacé l'amorceur, et voulez le restaurer, il faudra procéder comme indiqué dans la dernière section de ce chapitre.

Évidemment, si vous n'avez pas de problème d'espace mémoire, il n'y a aucune raison d'effacer l'amorceur. L'interface ICSP fonctionne avec ou sans celui-ci. Si vous utilisez l'interface, il suffit d'ignorer la présence de l'amorceur.

Pour téléverser sans amorceur, il faut donc un équipement spécial, appelé programmateur. Les plus réputés sont ceux d'Atmel, le AVR-ISP MKII et le nouveau Atmel-ICE. Ils sont excellents en termes de possibilités et de compatibilité. En revanche, Atmel a décidé de ne plus produire le premier des deux, au profit du seul second. De nombreux équipements restent cependant disponibles. Notez que l'ancien AVR-ISP MKII ne supportait pas le standard JTAG que je décris plus loin.

Du fait que l'interface ICSP utilise en réalité un port série selon le protocole SPI, de nombreux appareils peuvent servir à réaliser le transfert, même une deuxième carte Arduino, comme je le montre un peu plus loin. Parmi les programmateurs prêts à l'emploi, citons l'USBtinyISP d'Adafruit (Figure 6.10) et le programmateur Pocket AVR de SparkFun (Figure 6.11).

Le premier des deux est un kit, mais il est assez simple à monter. Vous en apprendrez plus à son sujet sur le site d'Adafruit. Le programmateur Pocket AVR est livré déjà monté.

Le programmateur ne sert pas qu'à téléverser le logiciel ; vous pouvez également l'utiliser pour lire les valeurs des registres, examiner le contenu de la mémoire et modifier les bits fusibles. Cette dernière possibilité est une bonne raison d'acquérir un programmateur ISP. Je rappelle que j'ai décrit les bits fusibles à la fin du Chapitre 3.

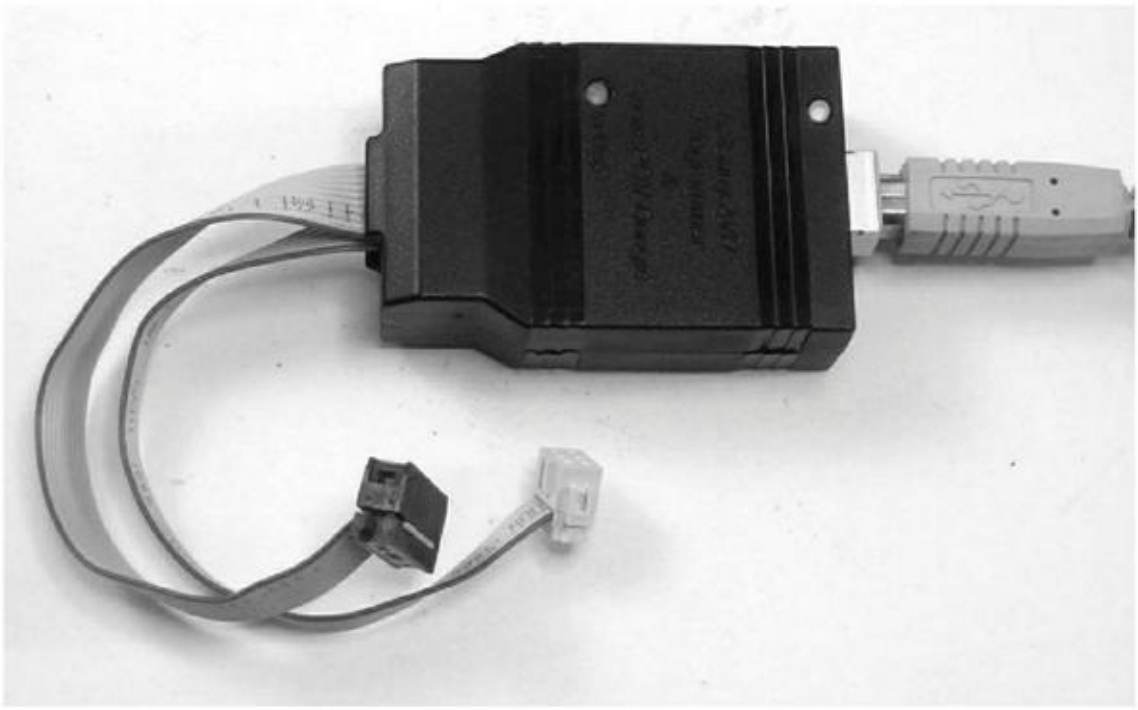


Figure 6.10 : Le programmateur USBtinyISP d'Adafruit, assemblé.

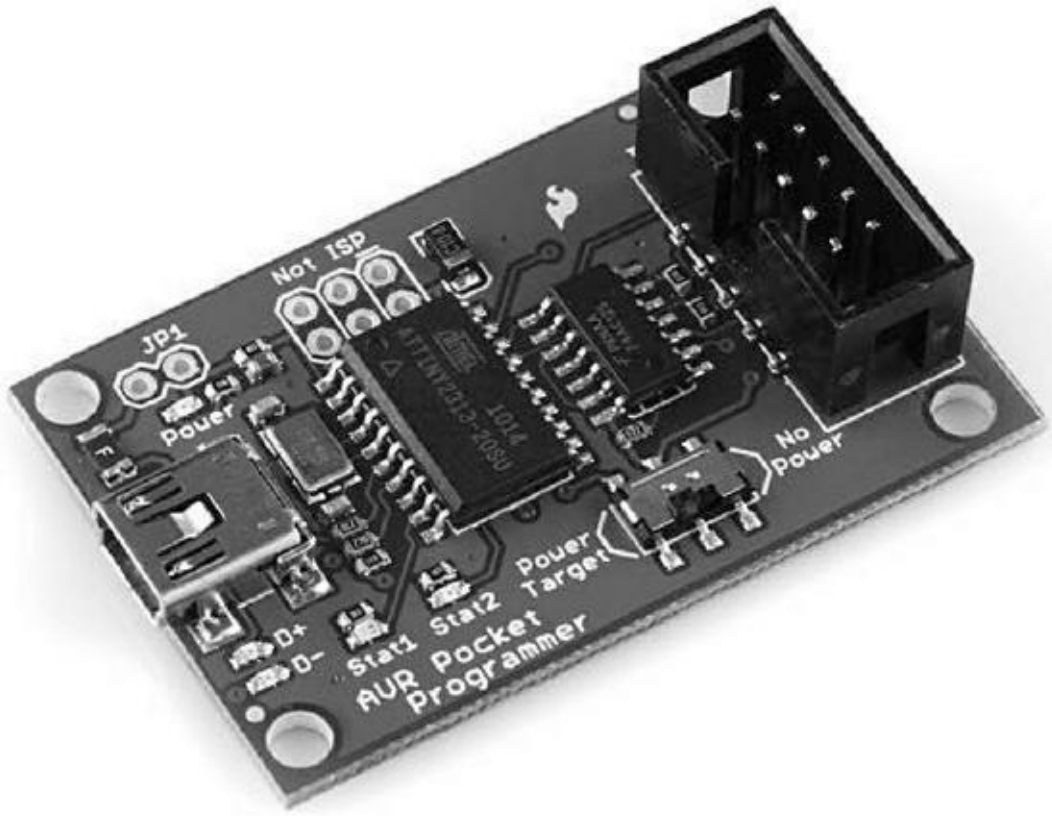


Figure 6.11 : Le programmateur Pocket AVR de SparkFun.

L’interface de débogage JTAG

JTAG est l’abréviation de *Joint Test Action Group*. Cette norme a d’abord été créée pour les tests de circuits électroniques. C’est une interface à bas niveau qui permet de contrôler les possibilités de mise au point des microcontrôleurs. La définition officielle fait l’objet d’un document standard IEEE (1149.1-1990). Si le cœur vous en dit, vous pouvez vous le procurer sur le site de l’association IEEE (`standards.ieee.org`).

Atmel n’a pas prévu de support JTAG dans tous ses microcontrôleurs. La lecture de la documentation actuelle me laisse croire que cette norme est reconnue par la série Xmega, mais pas par les microcontrôleurs 8 bits, ceux utilisés dans les cartes Arduino. Cela dit, le nombre de microcontrôleurs de la famille AVR est tel qu’il est bien possible que certains modèles Xmega aient le support JTAG et d’autres non.

Je tiens cependant à vous rassurer : vous n’avez pas vraiment besoin des possibilités sophistiquées de JTAG. Cette fonction est très pratique pour progresser instruction par instruction dans l’exécutable avec un débogueur et pour examiner les valeurs des registres à chaque instant, mais en général vous pouvez vous contenter de tester l’état des broches de sortie avec un oscilloscope ou un analyseur logique pour savoir où en est le programme.

En ce qui concerne l’accès aux fonctions internes du microcontrôleur, vous pouvez utiliser le programmateur USBtinyISP déjà montré pour modifier les bits fusibles et pour charger la mémoire EEPROM. Autrement dit, à moins d’avoir réellement besoin d’utiliser un outil JTAG, vous pouvez vous passer de cette dépense non négligeable.

Le téléverseur avrdude

Un programmateur pour AVR est donc intéressant, mais il ne sait pas de lui-même prendre contact avec la machine hôte pour aller y récupérer le fichier exécutable à implanter. Ce genre d’opération est réalisé par l’outil avrdude.

Le nom de l’outil provient de l’expression *AVR Download UploaDEr*. C’est bien un outil pour transférer dans un sens ou dans l’autre du code et des données en relation avec les espaces de stockage d’un microcontrôleur AVR. La

Figure 6.12 montre la page d'aide de l'outil quand vous le lancez sans fournir de paramètres.

avrdude est capable d'implanter des données dans la mémoire EEPROM et d'intervenir sur les bits fusibles et les bits de verrouillage. Il fonctionne en mode texte sur ligne de commande, mais également en mode interactif. Le mode en ligne de commande est à privilégier lorsque vous travaillez depuis un script ou un fichier de production make. Le mode interactif vous permet de naviguer dans la mémoire du microcontrôleur, de modifier des octets individuellement dans la mémoire EEPROM et d'intervenir sur les bits fusibles et de verrouillage.

L'outil avrdude reconnaît un certain nombre de programmateurs, et notamment l'Atmel STK500, l'AVRISP MKII, les logiciels émulateurs de protocole série (*bit-banger*) et même une interface sur port parallèle. Vous trouverez le manuel au format PDF à l'adresse `bit.ly/avrdude-pdf`.

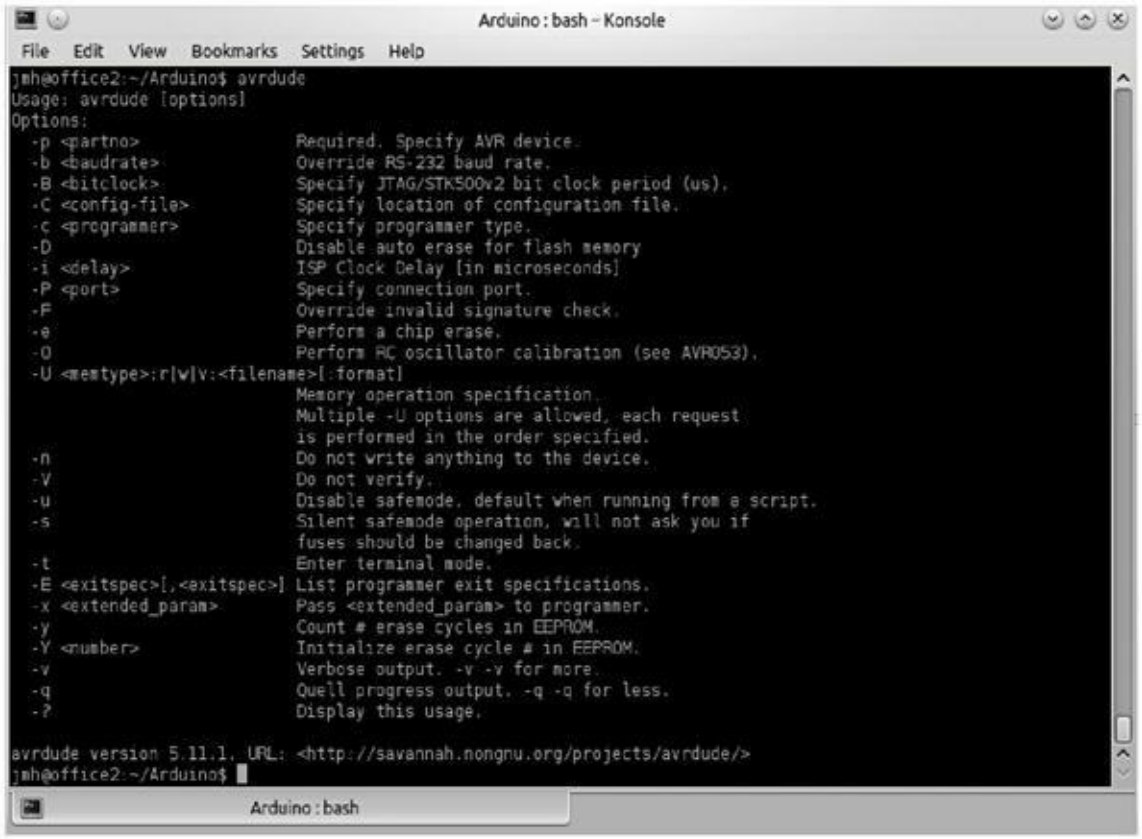


Figure 6.12 : Affichage de l'aide de l'outil avrdude.

C'est cet outil qui est utilisé par l'atelier Arduino pour effectuer ses téléversements. Vous pouvez même visualiser la ligne de commande qui dirige le

téléversement en activant l'affichage correspondant dans les préférences. Voici par exemple la ligne de commande qui correspond au téléversement du petit programme d'alarme du Chapitre 5 (j'ai réparti la ligne unique sur plusieurs pour augmenter la lisibilité) :

```
/usr/share/arduino/hardware/tools/avrdude
  -C/usr/share/arduino/hardware/tools/avrdude.conf
  -v -v -v -v -patmega328p -carduino -P/dev/ttyUSB0 -
b57600 -D
  -
Uflash:w:/tmp/build2510643905912671503.tmp/simple_alarm.cp
```

Cette ligne de commande utilise huit options dont la description est fournie dans le Tableau 6.8. Notez que j'ai répété quatre fois l'option **-v**, ce qui demande d'afficher de plus en plus d'informations sur les traitements, c'est-à-dire d'être plus verbeux.

Tableau 6.8 : Quelques options de ligne de commande d'avrdude.

Option	Fonction	Description
–C	Configuration	Spécifie le fichier de configuration avec son chemin d'accès
-p	Processor ID	Identifie la carte cible du programmateur. Dans l'exemple, c'est le microcontrôleur ATmega328P
-c	Programmer ID	Précise le programmateur à utiliser, qui est ici l'amorceur Arduino
-P	Port name	Précise le port de connexion. Sous Linux, c'est un pseudo port série qui s'écrit /dev/ttyUSB
-b	Port baud	Spécifie le débit exprimé en bauds
-D	Auto-erase	

		Désactive l'auto-effacement de la mémoire flash
-U	Upload	Chaîne de texte composite désignant le fichier exécutable à implanter
-v	Configuration	Mode bavard ou verbeux

L'argument **-U** comporte quatre éléments séparés par un signe deux-points : le type de mémoire cible (**flash**), le mode (**w** pour écriture), le chemin d'accès au fichier exécutable et enfin la lettre **i** pour préciser que le fichier à installer est au format Intel Hex.

Utilisation d'un Arduino comme programmateur ISP

En téléversant l'utilitaire approprié sur une carte Arduino, vous pouvez la transformer en un programmateur ISP pour une autre carte Arduino. Vous pouvez ainsi mettre en place un nouvel amorceur. Les explications sont données sur le site officiel Arduino (`bit.ly/arduino-isp`).

Il suffit d'avoir deux cartes Arduino et le croquis approprié, déjà fourni avec l'atelier Arduino. La Figure 6.13 montre comment interconnecter les deux cartes. Dans l'exemple, c'est une carte Duemilanove qui sert de programmateur pour une carte Uno R3. Cela fonctionne même avec des cartes qui ne possèdent pas le même brochage pour les signaux SPI, comme la Leonardo, mais cela suppose une petite adaptation. La documentation déjà citée donne des détails.

Fonctionnement de l'amorceur bootloader

Le but de l'outil amorceur est de se tenir prêt à accepter de recevoir les données d'un fichier exécutable depuis le système de développement hôte. Le microcontrôleur des cartes Arduino est déjà doté de l'amorceur. En général, vous n'aurez pas besoin de l'enlever, ni de le remplacer. Sauf si vous avez vraiment besoin de récupérer les quelques kilooctets que va occuper l'amorceur, le plus simple est de le laisser en place et de l'utiliser.

Le fonctionnement de l'amorceur Arduino ressemble à celui de n'importe quel microcontrôleur doté d'un espace mémoire flash. Il doit d'abord implanter le programme dans la mémoire locale puis passer le relais, c'est-à-dire le flux d'exécution, au début de ce programme. Dès que vous mettez une carte Arduino sous tension, elle exécute l'amorceur. Celui-ci se place en attente de ce qui pourrait arriver par l'interface. S'il ne détecte rien au bout de quelques secondes, il lance l'exécution du programme qui est déjà présent dans la mémoire flash, dans la section principale de la mémoire.

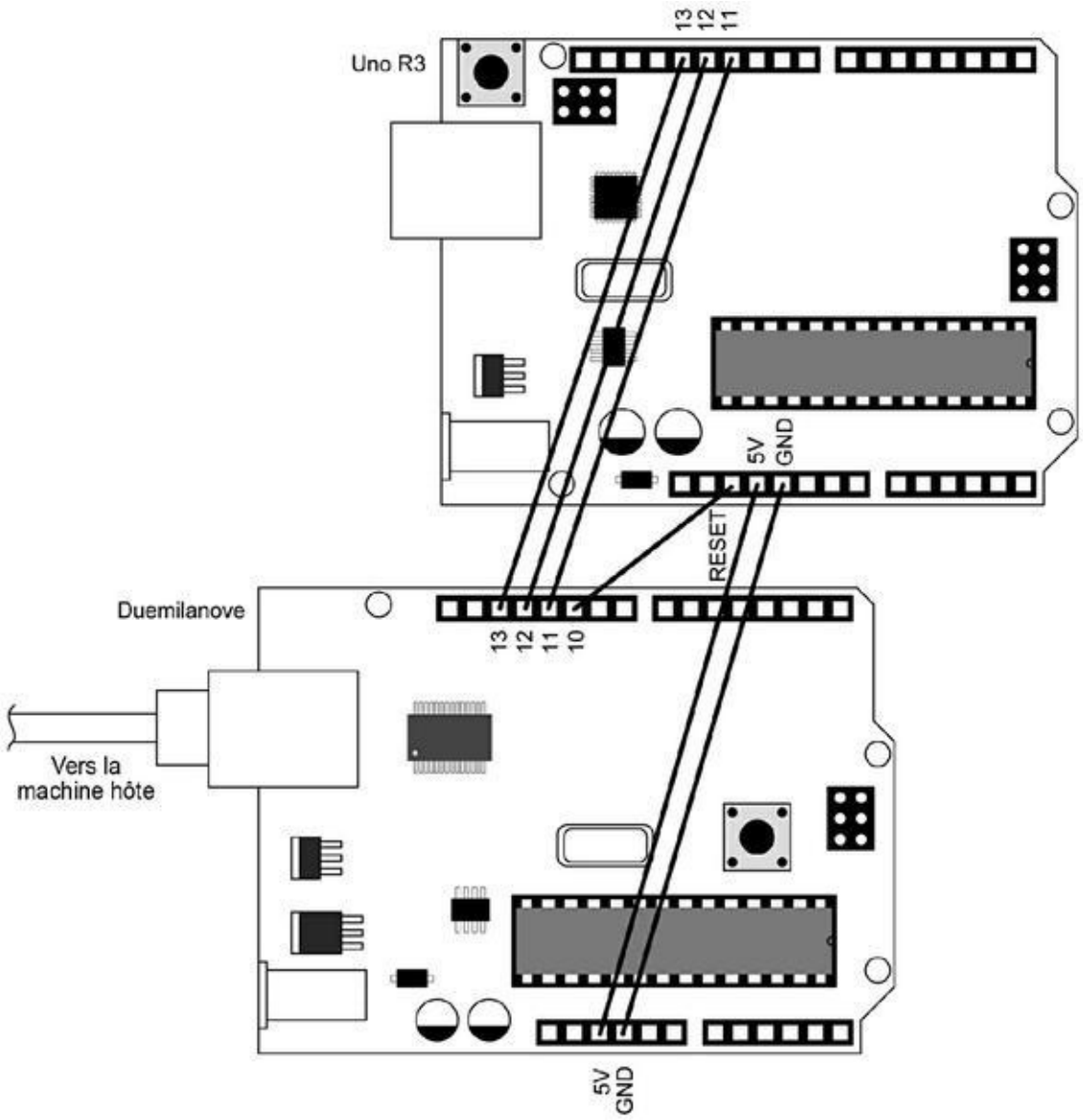


Figure 6.13 : Utilisation d'une carte Arduino pour reprogrammer une autre carte Arduino.

Depuis quelques années, les cartes Arduino sont devenues capables de tester périodiquement l'arrivée de données, pendant que le processeur est en cours d'exécution. Autrement dit, lorsque vous téléversez un programme ou une nouvelle version, le programme en cours est interrompu pour redonner le contrôle à l'amorceur afin qu'il puisse implanter la nouvelle image exécutable. Dans les anciennes cartes Arduino, il fallait réussir à appuyer sur le bouton Reset juste au moment où l'atelier Arduino commençait à téléverser le programme, afin de capter l'arrivée des données. Il fallait s'y prendre à plusieurs fois pour réussir à être synchronisé. Certaines cartes compatibles Arduino conservent cet ancien fonctionnement.

Une fois que l'amorceur a vérifié que les données qu'il a reçues correspondent bien à un programme valable, il déverrouille la mémoire flash et commence à écrire les données du programme, en évitant bien sûr de les écrire dans la zone où il se trouve lui-même. Une fois que le téléversement est terminé, l'amorceur revient ouvrir la zone mémoire et génère une interruption pour obliger le processeur à exécuter la première instruction du programme (vectorisation), soit l'adresse de début.

Les versions actuelles de l'amorceur Arduino peuvent recevoir les données à une vitesse allant jusqu'à 19 200 bauds, soit un peu moins de 2 ko/s. Les anciennes versions étaient limitées à la moitié, soit 9 600 bauds. Si nécessaire, vous pouvez forcer la vitesse à cette limite. À la vitesse actuelle, il faut quand même compter plusieurs secondes pour transférer un exécutable, dès qu'il occupe plus d'une dizaine de kilo-octets.

Le code source de l'amorceur est disponible sur le site Arduino. Il n'est pas inutile de le passer en revue. Vous y remarquerez notamment que la portion qui réalise les accès à la mémoire flash est écrite en langage assembleur. Les actions nécessaires sont très proches de la partie matérielle, et l'assembleur est vraiment le langage à employer dans cette situation. Le code source permet d'ailleurs de voir quelles opérations sont réalisées à bas niveau dans un microcontrôleur. Vous constaterez qu'il se passe bien des choses lorsque vous lancez même le plus simple des programmes.

Remplacement de l'amorceur Arduino

Lorsque vous avez envie ou besoin d'utiliser un nouvel amorceur AVR sur une carte Arduino, il vous faut bien sûr un programmateur utilisant le port ICSP,

comme les deux montrés dans les Figures 6.10 et 6.11. Vous pouvez aussi utiliser la technique basée sur deux cartes Arduino.

Le programme amorceur est implanté dans une région spéciale de la mémoire flash, et la taille qu'il occupe va de 256 octets à 4 ko, selon le type de microcontrôleur. Dans le menu Outils de l'atelier Arduino, vous disposez d'une commande pour choisir le type de programmateur ; vous pouvez alors compiler le code source puis le téléverser vers le microcontrôleur. Voyez l'aide en ligne de l'atelier et le site officiel pour tout détail.

Pour aller plus loin

Ce chapitre est véritablement l'un des plus denses du livre. Nous y avons découvert les composants de la chaîne d'outils AVR-GNU, l'outil d'automatisation de production make, la programmation en langage assembleur et quelques techniques concernant les amorceurs pour AVR. Même si vous n'aurez pas à utiliser ces outils, il vous sera toujours utile d'en savoir un peu plus que les autres au sujet de ce qui se passe en coulisses. Cela vous donne également un aperçu de la complexité qui était la règle dans le monde du développement pour microcontrôleurs avant l'arrivée de l'atelier Arduino.

Si vous voulez aller plus loin sur les sujets de ce chapitre, n'hésitez pas à consulter d'abord l'Annexe D. Les microcontrôleurs embarqués sont devenus des éléments omniprésents dans le monde moderne. Pour un ordinateur que l'on désigne sous ce terme, il y en a des centaines qui se cachent dans les télécommandes de téléviseurs, les fours à micro-ondes, les chaînes stéréo, les lecteurs de DVD, les systèmes de feux tricolores, votre automobile, et même dans le clavier de votre ordinateur. Tous ces petits appareils ont été programmés au moyen des techniques dont nous avons parlé dans ce chapitre, et la plupart sont toujours programmés de cette façon.

CHAPITRE 7
Librairies Arduino

Lorsque vous installez l'atelier IDE Arduino, vous installez en même temps toute une série de librairies de fonctions (*alias* bibliothèques) prêtes à l'emploi. Elles définissent des fonctions très diverses : accès à des périphériques comme une interface Ethernet, un afficheur à cristaux liquides, une interface série, et bien d'autres.

Le terme librairie désigne normalement un conteneur regroupant plusieurs fonctions, mais les modules ne sont pas tous dans le format attendu pour une librairie, c'est-à-dire du code précompilé aux formats de fichier *.a* (Archive) ou *.so* (*Shared Object*) habituels sous Unix et Linux. Souvent, ce sont des fichiers de code source C ou C++ (donc avec les limitations liées à avr-gcc) ; néanmoins, le résultat sera le même. Le code des librairies est simplement compilé à la demande pour produire du code objet qui pourra être lié au code principal du projet (revoir les Chapitres 5 et 6). Certaines librairies se présentent vraiment au format binaire, et c'est le cas des composants de la suite *avr-libc*. Lorsque vous voulez connaître la provenance d'une librairie ou d'un module de code, consultez la documentation d'*avr-libc* et celle d'Arduino.

Je rappelle qu'une fois le croquis principal et chacun des modules de librairie compilés, le lieur procède à l'ajout des adresses des appels aux fonctions des librairies depuis le code rédigé par le programmeur pour produire un fichier exécutable (binaire) unique. L'outil avrdude est lancé par l'atelier IDE (voir le Chapitre 6) pour entrer en communication avec l'amorceur qu'il trouve dans le microcontrôleur sur la carte Arduino (Chapitre 5) et lui transférer le code exécutable qui est alors stocké dans la mémoire embarquée.

Revoyez si nécessaire le Chapitre 5 pour les détails de ce processus. Le Chapitre 6 a montré comment produire du code uniquement avec le jeu d'outils avr-gcc.

N. d. T. : Les menus et options de l'atelier Arduino continuent pour le moment à utiliser le terme *bibliothèque* pour traduire *library*, alors que le terme *librairie*, plus direct et tout aussi exact, a commencé depuis plusieurs années à prendre la relève.

Tour d’horizon des librairies

L’environnement de développement Arduino est livré d’office avec un certain nombre de librairies considérées comme essentielles : entrées-sorties série *via* le port USB, accès à la mémoire EEPROM, communications Ethernet, affichage LCD et contrôle de moteur pas à pas et servomoteurs sont quelques-unes des capacités intrinsèques que je décris dans les pages qui suivent.

Ces descriptions sont brèves. Pour tous détails et exemples d’utilisation, voyez la documentation des librairies Arduino ou l’aide interne de l’atelier (fichiers HTML). Je rappelle que la plus récente version de la documentation est toujours celle sur le site officiel Arduino. L’aide de l’atelier IDE présente les librairies standard qui sont livrées avec la version installée.

Vous avez accès à la liste des librairies déjà installées dans l’atelier IDE par la commande **Croquis/Importer bibliothèque**. La liste englobe les librairies essentielles et toutes celles que vous avez ajoutées depuis l’installation de l’atelier (l’ajout d’une librairie a été décrit dans le Chapitre 5).

Voici un résumé de chacune des librairies essentielles, installées d’office avec l’atelier IDE Arduino :

- EEPROM : lecture et écriture de l’espace mémoire « permanent » que constitue la partie EEPROM du microcontrôleur AVR.
- Ethernet : communication Ethernet par le biais du bouclier Arduino Ethernet.
- Firmata : communication avec les applications de la machine hôte selon un protocole série standard.
- GSM : connexion au réseau GSM/GPRS avec un bouclier GSM.
- LiquidCrystal : contrôle d’un afficheur à cristaux liquides LCD.
- SD : lecture et écriture de l’espace mémoire d’une carte mémoire flash au format SD.
- Servo : pilotage de servomoteur.

- SPI : exploitation du bus de communication SPI (*Serial Peripheral Interface*).
- SoftwareSerial : communications série sur n'importe quelle broche numérique.
- Stepper : pilotage de moteur pas à pas.
- TFT : affichage de texte, de formes et d'images sur écran Arduino TFT.
- WiFi : communications réseau sans fil WiFi avec le bouclier Arduino WiFi.
- Wire : exploitation du bus de communication sur deux fils TWI/I2C pour échanger des données dans un réseau de périphériques et de capteurs.
- Esplora : accès aux actionneurs et capteurs disponibles sur la carte Esplora (librairie dédiée à Esplora).
- USB : communications série *via* USB pour les cartes Leonardo, Micro, Due et Esplora.
- Keyboard : envoi d'événement de frappe de touches vers la machine hôte.
- Mouse : envoi d'événements de mouvement du pointeur de souris vers la machine hôte.

EEPROM

La librairie EEPROM permet de stocker et de récupérer des données dans la mémoire permanente du microcontrôleur AVR : la mémoire EEPROM. Les données ne sont pas effacées à la mise hors tension de la carte. L'espace de mémoire flash du microcontrôleur est permanent aussi, mais son contenu est vidé lors de chaque téléversement d'un programme. L'accès à l'espace EEPROM suppose d'utiliser des fonctions spécifiques. La quantité de mémoire EEPROM varie d'un microcontrôleur à l'autre : de 512 octets pour l'ATmega168 à 4 ko pour les ATmega1280 et ATmega2560. Les détails ont été fournis dans le Chapitre 3.

La classe nommée EEPROM définit toutes les fonctions pour lire, écrire et mettre à jour de l'espace de stockage EEPROM. Pour pouvoir appeler ces fonctions, il faut bien sûr d'abord créer une instance de cette classe, c'est-à-dire un objet.

Le fichier d'en-tête *EEPROM.h* déclare une instance statique de la classe EEPROM. Pour instancier vous-même un objet à partir de cette classe, procédez ainsi :

```
EEPROMClass maEeprom;
```

Les anciennes versions de cette librairie EEPROM ne définissaient que deux fonctions pour accéder à la mémoire EEPROM du microcontrôleur AVR :

read()

Lit un octet à l'adresse spécifiée dans la mémoire EEPROM. Si ce lieu n'a pas été préparé, la valeur renvoyée est 255. La fonction renvoie bien sûr l'octet demandé. Les adresses commencent à zéro (0).

```
uint8_t eeprom.read(int address);
```

write()

Écrit un octet à l'adresse spécifiée dans la mémoire EEPROM du microcontrôleur. Les adresses commencent à zéro (0). Comptez environ 3,3 ms pour une écriture vers l'EEPROM qui est annoncée capable de supporter de l'ordre de 100000 cycles d'écriture/effacement, ce qui n'est pas rien. La fonction ne renvoie aucune valeur.

```
void eeprom.write(int adresse, uint8_t valeur);
```

Les versions plus récentes d'EEPROM définissent trois autres fonctions : `update()`, `get()`, `put()` et l'opérateur `EEPROM[]`. L'étude du code source d'EEPROM vaut le détour, car vous y verrez comment le code gère des types de données variés.

update()

Écrit un octet de données vers l'EEPROM à l'adresse demandée, seulement si la valeur à écrire diffère de celle qui s'y trouve déjà. Ne renvoie aucune valeur.

```
void eeprom.update(int adresse, uint8_t valeur);
```

put()

Cette fonction dépose (*put*), donc écrit une valeur d'un type de données ou d'un autre dans la mémoire EEPROM, à partir de l'adresse indiquée. La donnée peut être d'un type primaire (`int` ou `float`, par exemple) ou bien d'un type composé (une structure). La fonction renvoie une référence à l'objet qui a été transmis au moment de l'appel dans l'argument *`data`*. La fonction exploite `update()` pour la partie écriture. Il n'y aura donc pas d'écriture si le contenu trouvé à cette adresse est identique à ce que l'appel cherche à y stocker.

```
data_ref eeprom.put(int adresse, data);
```

get()

Cette fonction lit une donnée d'un certain type ou un objet depuis la mémoire EEPROM puis la renvoie à l'adresse spécifiée en argument, octet par octet, jusqu'à avoir copié toutes les données de l'objet désigné par *`data`*. La fonction renvoie une référence à l'objet transmis par l'argument *`data`*.

```
data_ref eeprom.get(int adresse, data);
```

EEPROM[]

Cet opérateur EEPROM[] permet d'accéder à la mémoire EEPROM comme s'il s'agissait d'un tableau. L'adresse effective dans l'EEPROM correspond à `adresse_base_EEPROM + int index`. L'opérateur renvoie une référence à la cellule mémoire EEPROM.

```
data_ref EEPROM[int index]
```

L’exemple simplifié suivant est inspiré de la documentation d’EEPROM sur le site *Arduino.cc*, après quelques retouches. Il écrit et lit dans la mémoire EEPROM puis utilise l’opérateur modulo (reste) pour tester si la valeur est paire ou impaire :

```
#include <EEPROM.h>
// Créons notre objet de classe EEPROMClass au lieu de
partir
// de celui déclaré statique dans EEPROM.h
EEPROMClass eeprom;

int a = 0;
int valeur;

void setup()
{
  Serial.begin(9600);
  // Initialise la EEPROM avec un motif de données
  for (int i = 0; i < 512; i++) {
    if (i % 2)                // Pair ou impair ?
      eeprom.write(i, 0);     // Impair !
   else
      eeprom.write(i, 1); // Pair !
   }
  }

// Le ATmega168 offre 512 octets d’EEPROM, le ATmega328
1024 octets.
// Nous ne testons que 512 octets, donc le test sera
correct pour
// tous les AVR Arduino.
void loop()
{
  valeur = eeprom.read(a);
  Serial.print(a);
  Serial.print("\t");
  Serial.print(valeur);
  Serial.println();
  // La variable a étant déclarée globale (en dehors de
loop()),
  // elle ne perd pas sa valeur entre deux tours.
```

```
  a++;
  if (a == 512)
    a = 0;
  delay(500);
}
```

Ethernet

Cette librairie définit un jeu de fonctions permettant d'interagir avec un bouclier Arduino Ethernet. Elle est assez complexe et offre aussi bien les fonctions pour les rôles de client que de serveur. Elle accepte de gérer jusqu'à quatre connexions entrantes ou sortantes simultanées (ou une combinaison des deux sens). Le bouclier Ethernet exploite l'interface SPI pour dialoguer avec la carte Arduino.

La librairie Ethernet contient cinq classes C++ avec des relations d'héritage dont vous n'avez normalement pas à vous soucier en pratique. Cela dit, les définitions à bas niveau des classes sont visibles dans le sous-répertoire *hardware/arduino/avr/cores/arduino* de celui du code source Arduino. Voici d'abord la liste de ces cinq classes Ethernet avec les méthodes publiques (fonctions membres) de chacune. Nous verrons ensuite les détails de chaque méthode.

- **Classe Ethernet**
  - `begin()`
  - `localIP()`
  - `maintain()`
- **Classe IPAddress**
- **Classe Server**
  - `EthernetServer()`
  - `begin()`
  - `available()`

- write()
- print()
- println()

- **Classe Client**
  - EthernetClient()
  - connected()
  - connect()
  - write()
  - print()
  - println()
  - available()
  - read()
  - flush()
  - stop()

- **Classe EthernetUDP**
  - begin()
  - read()
  - write()
  - beginPacket()
  - endPacket()
  - parsePacket()
  - available()

— `stop()`

— `remotePort()`

Classe pour Ethernet : EthernetClass

En créant une instance de la classe `EthernetClass`, vous initialisez la librairie Ethernet et préparez les paramètres réseau. Une instance est déjà créée sous le nom `Ethernet` dans le fichier *Ethernet.cpp* puis exportée dans *Ethernet.h*. Vous pouvez utiliser directement cet objet ou instancier le vôtre.

begin()

Cette méthode prépare la librairie en configurant les paramètres de communication réseau. Sa définition est surchargée afin qu'il y ait cinq syntaxes d'appel.

Tous les arguments sont des tableaux d'une longueur de `uint8_t` octets. La variante d'appel DHCP seul (`Ethernet.begin(mac)`) renvoie la valeur 1 si un bail DHCP (*lease*) a été obtenu, et 0 sinon. Les autres variantes ne renvoient rien. La classe `IPAddress` est décrite dans la prochaine section.

```
int Ethernet.begin(uint8_t *mac);
void Ethernet.begin(uint8_t *mac, IPAddress ip);
void Ethernet.begin(uint8_t *mac, IPAddress ip, IPAddress
dns);
void Ethernet.begin(uint8_t *mac, IPAddress ip, IPAddress
dns, IPAddress
  gateway);
void Ethernet.begin(uint8_t *mac, IPAddress ip, IPAddress
dns, IPAddress
  gateway, IPAddress subnet);
```

localIP()

Récupère l'adresse IP de la machine hôte locale (qui est le bouclier Ethernet), ce qui permet de connaître cette adresse dans le cadre de l'utilisation de DHCP. Si l'objet Ethernet local a été correctement initialisé, `Ethernet.localIP()` renvoie un objet `IPAddress` qui contient l'adresse attribuée ou spécifiée.

```
IPAddress Ethernet.localIP();
```

maintain()

Cette méthode n'existait pas au départ dans la librairie. Lorsqu'un périphérique réseau obtient son adresse IP par le serveur DHCP, c'est le principe du bail. Un bail DHCP est octroyé pour un temps prédéfini (tout dépendant de la configuration du serveur DHCP). Cette méthode `Ethernet.maintain()` sert à renouveler un bail DHCP.

La méthode renvoie 0 si elle n'a rien fait, 1 si le renouvellement a échoué, 2 s'il a réussi, 3 si le Rebind DHCP (demande universelle) a échoué et 4 si le Rebind a réussi.

```
int Ethernet.maintain();
```

Classe IPAddress

Cette classe définit l'objet qui mémorise les données d'adressage local et distant. Quatre constructeurs sont disponibles, chacun acceptant un format d'adresse IP différent :

```
IPAddress()
IPAddress(uint8_t octet_1,
          uint8_t octet_2,
          uint8_t octet_3,
          uint8_t octet_4)
IPAddress(uint32_t adresse)
IPAddress(const uint8_t *adresse)
```

La classe `IPAddress` sert à créer des instances de types de données d'adresse. Elle accepte soit quatre octets d'adresse IP (dans le style 192.168.1.100, sans les points séparateurs), soit un double mot sur 32 bits contenant l'adresse IP, soit un tableau d'octets non signés. Voici quatre exemples :

```
IPAddress ip(192, 168, 0, 2);
IPAddress dnServer(192, 168, 0, 1);
IPAddress gateway(192, 168, 0, 1);
IPAddress subnet(255, 255, 255, 0);
```

Nous créons ainsi quatre objets de classe `IPAddress`. Les fonctions de la librairie Ethernet savent chercher exactement ces noms lors de la mise en place d'une interface Ethernet. Vous constatez que les quatre créations utilisent le format à adresse explicite.

Le code source de cette classe est lisible comme tous les autres dans le sous-répertoire *Arduino/ hardware/arduino/avr/cores/arduino* des fichiers source d'Arduino.

Classe serveur : EthernetServer

Dans la technique Ethernet, un serveur est une machine (un hôte) qui se tient à l'écoute de demande de connexion depuis un autre système (le client) et y répond en établissant un canal de communication. Le serveur attend sans jamais prendre l'initiative d'ouvrir une connexion. Prenons l'exemple d'un serveur Web. Il attend que des navigateurs clients lui demandent de servir des pages Web. Il répond en transmettant les données puis il se replace en attente. Chaque clic d'un lien Web, d'un bouton et chaque saisie de texte dans un navigateur provoque la création et l'envoi d'une requête vers le serveur Web.

La classe `Server` n'est jamais utilisée directement ; c'est une classe de base pour la classe `EthernetServer`, et plusieurs autres. Comme pour `IPAddress`, le code source est disponible à l'adresse déjà mentionnée.

EthernetServer()

Définit le numéro du port d'écoute des requêtes entrantes des clients. Ce réglage est normalement réalisé au moment de la création d'un objet de classe `EthernetServer`. La fonction ne renvoie aucune valeur (type `void`).

```
EthernetServer server(int port);
```

Un exemple :

```
EthernetServer monServ = EthernetServer(port);
```

L'argument `port` peut valoir entre 0 et 65535, mais la plage de 0 à 1024 est réservée aux services système comme FTP, SSH et un éventuel serveur Web.

Utilisez une valeur supérieure à 9000 pour réduire le risque de conflit.

begin()

Ordonne au serveur de se placer à l'écoute des connexions de clients sur le port spécifié lors de l'instanciation de l'objet serveur.

```
void monServ.begin();
```

available()

Renvoie une connexion vers un client présent et prêt à dialoguer.

```
EthernetClient monCli = monServ.available();
```

write()

Envoie les données désignées en argument aux clients connectés à ce serveur. Le format des données peut être un seul octet (`char` ou `byte`) ou (plus généralement) un pointeur sur un tableau de caractères. La méthode renvoie le nombre d'octets écrits.

```
int monServ.write(char data);
int monServ.write(char *data, int taille);
```

print()

Envoie les données aux clients connectés à ce serveur sous forme d'une séquence de caractères en codage ASCII.

println()

Comme `print()`, mais en ajoutant un caractère de saut de ligne en fin d'envoi. Les types de données possibles sont `char`, `byte (uint8_t)`, `int`, `long` et chaîne (*string*). La méthode renvoie le nombre d'octets écrits. Appelée sans argument, la méthode envoie seulement un caractère de saut de ligne.

```
int monServ.println();
int monServ.println(char *data);
int monServ.println(char *data, int BASE);
```

Classe client : EthernetClient

La classe `Client` permet d'incarner des objets clients Web pouvant se connecter à un serveur Web pour échanger des données. C'est le client qui a l'initiative de débuter une transaction. Le serveur se tient à l'écoute de demandes de connexions des clients. Une fois la connexion établie, le client peut demander des données, en envoyer ou demander au serveur de déclencher une action pour son compte.

La classe `Client` n'est jamais utilisée directement ; c'est une classe de base pour la classe `EthernetClient` qui en hérite donc. Comme pour `Server`, le code source est disponible à l'adresse *Arduino/hardware/arduino/avr/cores/arduino*.

EthernetClient()

Crée une instance de client capable de se connecter à un serveur localisé par une adresse IP et un numéro de port. La méthode `connect()` sert à spécifier ces deux paramètres. Voici un exemple :

```
byte servaddr[] = {172, 120, 40, 10};
EthernetClient monCli;
monCli.connect(servaddr, 80);
```

connected()

Teste si le client est connecté en renvoyant la valeur logique true ou false.

```
bool monCli.connected();
if (monCli.connected()) {
  // Action
}
```

connect()

Se connecte au serveur désigné par l'adresse IP (sur quatre octets) et le port, ou bien en fournissant une adresse URL Web.

```
int monCli.connect();
int monCli.connect(byte *servIP, int port);
int monCli.connect(char *URL, int port);
```

Renvoie une valeur entière qui décrit l'état de la connexion :

- `SUCCESS = 1`
- `TIMED_OUT = -1`
- `INVALID_SERVER = -2`
- `TRUNCATED = -3`
- `INVALID_RESPONSE = -4`

write()

Envoie la valeur spécifiée ou le contenu d'un tampon mémoire *`tampon`* au serveur. Les données sont transmises octet par octet. La méthode renvoie le nombre d'octets écrits, valeur que vous pouvez ignorer.

```
int monCli.write(uint8_t valeur);
int monCli.write(uint8_t *tampon, int len);
```

print()

Envoie les données au serveur connecté sous forme d'une séquence de caractères en codage ASCII. Les types de données possibles sont `char`, `byte` (`uint8_t`), `int`, `long` et chaîne (*string*). La méthode renvoie le nombre d'octets écrits.

Vous pouvez spécifier une base avec les mots clés BIN (binaire), DEC (décimale), OCT (octale) ou HEX (hexadécimale).

```
int monCli.print(data);
int monCli.print(data, base);
```

println()

Comme `print()`, mais en ajoutant un caractère de saut de ligne en fin d'envoi. Appelée sans argument, la méthode envoie seulement un caractère de saut de ligne.

```
int monCli.println();
int monCli.println(data);
int monCli.println(data, base);
```

available()

Renvoie le nombre de caractères prêts à être lus depuis le serveur connecté. Permet donc de tester si des données sont prêtes à être lues.

```
int monCli.available();
```

read()

Lit le prochain octet disponible sur le serveur. Pour en lire plusieurs, vous écrivez une boucle. Ou bien vous lisez puis analysez un caractère à la fois.

```
char monCli.read();
```

flush()

Supprime les caractères provenant du serveur qui sont en attente de lecture dans le tampon de réception.

```
void monCli.flush();
```

stop()

Déconnecte l'objet client du serveur actuel, après quoi le client peut se connecter à un autre serveur, ou au même.

```
void monCli.stop();
```

Classe EthernetUDP

Alors que le protocole TCP/IP est de type flux (*stream*), c'est-à-dire qu'il n'y a pas de marqueur de début et de fin d'envoi, le protocole UDP est de type datagramme. Chaque bloc de données est un paquet autonome appelé datagramme. Les données doivent tenir dans les limites de ce paquet. Il n'y a pas de détection d'erreur en UDP ni garantie de réception. Pourtant, ce protocole UDP offre une solution rapide et assez simple pour transmettre des données entre machines lorsque les données sont non critiques ou de faible volume ou lorsque le logiciel de niveau applicatif est en mesure de gérer la détection d'erreurs et la relance.

begin()

Prépare un objet de classe UDP pour se tenir prêt à écouter un flux de données entrant sur le port spécifié.

```
byte UDP.begin(int port);
```

read()

Lit des données depuis le port spécifié. Appelée sans argument, la méthode renvoie un caractère lu dans le tampon actuel. Si vous indiquez le nom du tampon et sa taille, elle renvoie *`maximum`* caractères trouvés dans le tampon *`tampon`*.

```
char UDP.read();
char *UDP.read(char *pkt_tampon, int maximum)
```

Normalement, vous appelez cette fonction juste après avoir appelé `UDP.parsePacket()`.

write()

Envoie des données vers la machine distante connectée. Cette méthode doit être appelée entre les appels à `beginPacket()` et à `endPacket()`, la première des deux préparant le paquet et la seconde effectuant l'envoi effectif vers la machine hôte distante.

```
byte UDP.write(char *message);
byte UDP.write(char *tampon, int taille)
```

beginPacket()

Ouvre une connexion UDP vers un hôte avec l'adresse IP et le port spécifiés. Renvoie 1 (true) si la connexion a réussi et 0 sinon.

```
int UDP.beginPacket(byte *ip, int port);
```

endPacket()

Envoie un paquet UDP qui vient d'être créé par appel à `write()` vers l'hôte configuré par `beginPacket()`.

```
int UDP.endPacket();
```

parsePacket()

Scrute une connexion UDP ouverte pour savoir si un datagramme est disponible et renvoie la taille des données en attente. Cette méthode doit être appelée avant toute tentative de lire les données éventuelles avec `parsePacket()` ou de tester leur présence avec `available()`.

```
int UDP.parsePacket();
```

available()

Renvoie le nombre d'octets en attente dans le tampon mémoire de réception. Ne doit être appelée qu'après `parsePacket()`.

```
int UDP.available();
```

stop()

Se déconnecte de la machine hôte UDP et libère les ressources utilisées pour la connexion UDP.

```
void USP.stop();
```

remoteIP()

Renvoie l'adresse IP de la machine UDP distante sous forme d'un tableau de quatre octets. Ne doit être appelée qu'après `parsePacket()`.

```
byte *UDP.remoteIP();
```

remotePort()

Renvoie le numéro de port UDP de la machine UDP distante en tant que valeur de type entier `integer`. Ne doit être appelée qu'après `parsePacket()`.

```
int UDP.remotePort();
```

Firmata

Firmata est le nom d'une librairie très versatile. Elle permet d'implanter une liaison série entre la carte Arduino et une application sur une machine hôte en employant un protocole de type MIDI. L'objectif de cette carte est de permettre de contrôler à distance la plupart des fonctions d'une carte Arduino, donc depuis une machine hôte, comme si la carte Arduino était une extension des entrées-sorties de la machine hôte.

Avant d'adopter Firmata dans un nouveau projet, faites un essai du logiciel de démonstration. Vous pouvez télécharger un client depuis l'ancien site wiki de Firmata. La partie Arduino du logiciel est déjà présente parmi les librairies installées d'office avec l'atelier IDE.

Nous allons passer en revue les fonctions de la librairie Firmata selon plusieurs catégories. Les composants ne sont cependant pas très documentés et vous devrez parfois chercher comment les utiliser en étudiant le code source. Pour tous détails, vous irez sur la page wiki de Firmata (même si elle est obsolète) et dans le référentiel Firmata GitHub. Lisez bien la définition du protocole de communication entre l'application hôte et celle utilisant Firmata qui s'exécute sur la carte Arduino.

Le code de Firmata fourni avec l'atelier Arduino IDE n'est sans doute pas la dernière version. Vérifiez sur le référentiel GitHub. Certaines des fonctions qui suivent ne sont peut-être pas présentes dans votre version.

Voici les catégories fonctionnelles de la librairie Firmata :

- **Méthodes de base**
  - `begin()`
  - `printVersion()`
  - `blinkVersion()`
  - `printFirmwareVersion()`
  - `setFirmwareVersion()`
- **Envoi de messages**
  - `sendAnalog()`
  - `sendDigitalPorts()`
  - `sendDigitalPortPair()`
  - `sendSysex()`
  - `sendString()`
  - `sendString()`
- **Réception de messages**
  - `available()`
  - `processInput()`
- **Fonctions de rappel (*callback*)**
  - `attach()`
  - `detach()`
- **Types de messages**

Méthodes de base

Ces méthodes servent à préparer la librairie et à assurer sa maintenance.

begin()

La forme sans argument de `begin()` initialise la librairie Firmata en réglant le débit série à 57600 bauds. Le deuxième format d'appel attend un argument de type long contenant le débit demandé en bauds. Le troisième format spécifie un flux de données différent de `Serial` et repose sur l'interface prédéfinie `Stream` (pour Ethernet, WiFi, *etc.*). Pour en savoir plus sur l'état actuel de cette méthode, vous consulterez le référentiel GitHub de Firmata.

```
void Firmata.begin();
void Firmata.begin(long);
void Firmata.begin(Stream &s);
```

printVersion()

Renvoie le numéro de version du protocole de la librairie vers la machine hôte.

```
void Firmata.printVersion();
```

blinkVersion()

Indique par clignotement la version du protocole sur la broche 13 (celle de la diode LED soudée sur la carte Arduino).

```
void Firmata.blinkVersion();
```

printFirmwareVersion()

Renvoie le nom et le numéro de version du protocole vers la machine hôte.

```
void Firmata.printFirmwareVersion();
```

setFirmwareVersion()

Définit comme nom et version du microgiciel (*firmware*) le nom du croquis sans l'extension de nom.

```
void Firmata.setFirmwareVersion(const char *nom, byte
vers_major, byte
  vers_minor);
```

Envoi de messages

sendAnalog()

Envoie un message analogique.

```
void Firmata.sendAnalog(byte broche, int valeur);
```

sendDigitalPort()

Envoie l'état des ports numériques sous forme d'octets distincts.

```
void Firmata.sendDigitalPort(byte broche, int portData);
```

sendString()

Envoie une chaîne vers la machine hôte.

```
void Firmata.sendString(const char* chaine);
```

sendString()

Envoie une chaîne vers la machine hôte avec un type de commande spécifique.

```
void Firmata.sendString(byte command, const char*
chaine);
```

sendSysex()

Envoie une commande contenant un nombre non prédéterminé d'octets.

```
void Firmata.sendSysex(byte command, byte bytec, byte*
bytev);
```

Réception de messages

available()

Teste la présence de messages entrants dans le tampon mémoire de réception.

```
int Firmata.available();
```

processInput()

Récupère et traite les messages entrants depuis le tampon puis transmet les données aux fonctions de rappel dûment configurées.

```
void Firmata.processInput();
```

Fonctions de rappel (callback)

Pour qu'une fonction soit associée à un certain type de message, elle doit être définie comme fonction de rappel. Firmata en reconnaît trois sortes : générique, chaîne et exclusive système (Sysex). Une quatrième sorte sert à gérer un Reset du système. Voici comment ces fonctions sont définies.

attach()

Associe une fonction à un type de message entrant.

```
void attach(byte command, callbackfunction maFoncRapp);
void attach(byte command, stringcallbackfunction
maFoncRapp);
void attach(byte command, sysexcallbackfunction
maFoncRapp);
void attach(byte command, systemResetcallbackfunction
maFoncRapp);
```

detach()

Dissocie une fonction d'un type de message entrant.

```
void Firmata.detach(byte commande);
```

Types de messages

Voici les différents types de messages auxquels une fonction de rappel peut être associée :

ANALOG_MESSAGE

La valeur analogique sur une seule broche.

DIGITAL_MESSAGE

Huit bits de données sur une broche numérique (un port).

REPORT_ANALOG

Active/inhibe le suivi d'une broche analogique.

REPORT_DIGITAL

Active/inhibe le suivi d'une broche numérique.

SET_PIN_MODE

Bascule le mode des broches entre INPUT/OUTPUT/PWM/*etc.*

FIRMATA_STRING

Pour les chaînes au format du langage C (chaînes AZT, à zéro terminal) utilise comme type de la fonction le type `stringCallbackFunction`.

SYSEX_START

Pour les messages génériques et de longueur non prédéfinie (*via* le protocole MIDI SysEx). Utilise pour type de la fonction le type `sysexCallbackFunction`.

SYSTEM_RESET

Replace le microgiciel (*firmware*) dans son état par défaut avec pour type de la fonction le type `systemResetCallbackFunction`.

GSM

Cette librairie s'utilise avec la carte bouclier GSM pour se connecter à un réseau téléphonique GSM/GPRS. Elle est livrée d'office avec l'atelier IDE Arduino à partir de sa version 1.0.4 et réunit la plupart des fonctions dont vous pouvez avoir besoin pour émuler un téléphone GSM : émission et réception d'appels vocaux et de messages texte SMS et connexion à Internet par le réseau GPRS. Le sigle GSM signifie *Global System for Mobile* et GPRS signifie *General Packet Radio Service*.

Les boucliers GSM comportent un modem pour échanger des données entre le port série et le réseau GSM. Ce modem se base sur des commandes de type AT dédiées chacune à une portion d'un domaine fonctionnel. La librairie GSM s'appuie sur la librairie `SoftwareSerial` pour les dialogues entre la carte Arduino et le modem GSM.

La librairie GSM a été ajoutée récemment. Si vous avez une ancienne version de l'atelier, procédez à une mise à jour si vous constatez qu'elle n'est pas installée.

Les classes de cette librairie sont assez sophistiquées, au point que ce livre ne peut fournir tous les détails. Nous verrons un résumé des possibilités. Pour tous détails, voyez la référence Arduino de la librairie GSM ou l'aide en ligne de l'atelier IDE Arduino. Certains fournisseurs (Adafruit, par exemple) proposent des cartes GSM compatibles avec des librairies spécifiques.

Compatibilité avec la librairie Ethernet

La librairie GSM est fortement compatible avec la librairie Arduino Ethernet ; le portage d'un programme écrit pour une des librairies Arduino Ethernet ou WiFi pour lui faire exploiter un bouclier GSM est assez simple. Quelques retouches seront requises, et notamment l'ajout de références vers les librairies GSM et GPRS et les appels pour obtenir les paramètres réseau auprès de l'opérateur de télécommunications concerné.

Structure de la librairie GSM

Cette librairie réunit dix classes primaires. Voici les fonctions que définit chacune d'elles :

- **Classe GSM**
  - begin()
  - shutdown()
- **Classe GSMVoiceCall**
  - getVoiceCallStatus()
  - ready()
  - voiceCall()
  - answerCall()
  - hangCall()
  - retrieveCallingNumber()
- **Classe GSM_SMS**
  - beginSMS()
  - ready()
  - endSMS()
  - available()
  - remoteNumber()
  - read()
  - write()
  - print()
  - peek()
  - flush()
  - Classe GPRS

- attachGPRS()

- **Classe GSMClient**
  - ready()
  - connect()
  - beginWrite()
  - write()
  - endWrite()
  - connected()
  - read()
  - available()
  - peek()
  - flush()
  - stop()

- **Classe GSMServer**
  - ready()
  - beginWrite()
  - write()
  - endWrite()
  - read()
  - available()
  - stop()

- **Classe GSMModem**

- begin()
- getIMEI()

- **Classe GSMScanner**
  - begin()
  - getCurrentCarrier()
  - getSignalStrength()
  - readNetworks()

- **Classe GSMPIN**
  - begin()
  - isPIN()
  - checkPIN()
  - checkPUK()
  - changePIN()
  - switchPIN()
  - checkReg()
  - getPINUsed()
  - setPINUsed()

- **Classe GSMBand**
  - begin()
  - getBand()
  - setBand()

Classe GSM

Cette classe configure les fonctions qui vont servir au dialogue avec le modem. Elle gère la connectivité du bouclier et réalise les opérations de déclaration système auprès de l'infrastructure GSM. Tous les programmes Arduino GSM/GPRS doivent créer un objet de cette classe pour pouvoir gérer les fonctions de communication à bas niveau.

C'est la classe de base de toutes les fonctions GSM. Voici comment instancier un objet :

```
GSM gsmbase;
```

begin()

Fait démarrer le modem GSM/GPRS et le relie à un réseau GSM. Voici la signature complète de cette méthode :

```
begin(char* pin=0, bool restart=true, bool
synchronous=true);
```

Cette méthode accepte quatre formats d'appel du fait que chaque argument non précisé dispose d'une valeur par défaut. Dans la première forme sans argument, on suppose que la carte SIM n'est pas protégée par un code PIN.

```
gsmbase.begin();
gsmbase.begin(char *pin);
gsmbase.begin(char *pin, bool restart);
gsmbase.begin(char *pin, bool restart, bool sync);
```

shutdown()

Arrête le modem (*power-off*).

```
gsmbase.shutdown();
```

Classe GSMVoiceCall

Cette classe active les communications vocales par le modem. Cela suppose qu'un microphone, un haut-parleur et quelques composants sont reliés au bouclier GSM.

C'est la classe de base des fonctions GSM d'émission et réception d'appels vocaux. Elle doit être instanciée comme ceci :

```
GSMVoiceCall gsmvc;
```

getVoiceCallStatus()

Renvoie une des quatre valeurs d'état d'un appel vocal : IDLE_CALL, CALLING, RECEIVING-CALL ou TALKING.

```
GSM3_voiceCall_st getVoiceCallStatus();
gsmvc.getVoiceCallStatus();
```

ready()

Renvoie l'état de la dernière commande. Renvoie 0 si elle est toujours en cours d'exécution, 1 en cas de réussite et plus de 1 en cas d'erreur.

```
int ready();

gsmvc.ready();
```

voiceCall()

Lance un appel vocal en mode asynchrone ou synchrone. Dans le mode asynchrone, la méthode rend le contrôle pendant l'émission de la sonnerie d'appel entrant. En mode synchrone, la fonction conserve le contrôle du flux d'exécution jusqu'à ce que soit le correspondant décroche, soit l'appel est annulé.

Le premier argument est une chaîne contenant le numéro à composer, avec ou sans préfixe de pays. Vous devez attendre la fin d'exécution de l'appel avant d'intervenir sur le tampon mémoire. L'argument de délai `timeout` est en millisecondes, et ne sert qu'en mode synchrone. Si vous spécifiez la valeur 0, `voiceCall()` attend indéfiniment que l'appelé décroche.

```
int voiceCall(const char* ntel, unsigned long
timeout=30000);
gsmvc.voiceCall(ntel);           // Délai timeout par
défaut
gsmvc.voiceCall(ntel, timeout);  // Délai timeout
spécifié
```

answerCall()

Accepte un appel vocal entrant. En mode asynchrone, la fonction renvoie 0 si la commande est toujours en cours d'exécution, 1 en cas de réussite et plus de 1 en cas d'erreur. En mode synchrone, renvoie 1 en cas de prise de ligne ou bien 0.

```
gsmvc.answerCall();
```

hangCall()

Raccroche un appel en cours ou une sonnerie d'appel entrant. En mode asynchrone, la fonction renvoie 0 si la commande est toujours en cours d'exécution, 1 en cas de réussite et plus de 1 en cas d'erreur. En mode synchrone, renvoie 1 en cas de raccrochage ou bien 0.

```
gsmvc.hangCall();
```

retrieveCallingNumber()

Récupère le numéro appelant et le stocke dans un tampon mémoire. L'argument *`tampon`* est un pointeur sur le tampon de type `char` et *`tailleTamp`* est sa taille en octets. La taille minimale est de 10 caractères.

En mode asynchrone, la fonction renvoie 0 si la commande est toujours en cours d'exécution, 1 en cas de réussite et plus de 1 en cas d'erreur. En mode synchrone, renvoie 1 en cas de récupération du numéro ou bien 0.

```
int retrieveCallingNumber(char* tampon, int tailleTamp);
gsmvc.retrieveCallingNumber(tampon, tailleTamp);
```

Classe GSM_SMS

Cette classe permet d’envoyer et de recevoir des SMS (*Short Message Service*).

beginSMS()

Définit le numéro de téléphone pour recevoir un message SMS. Le numéro est un tableau de caractères. En mode asynchrone, la fonction renvoie 0 si la commande est toujours en cours d’exécution, 1 en cas de réussite et plus de 1 en cas d’erreur. En mode synchrone, renvoie 1 en cas de réussite ou bien 0.

```
int SMS.beginSMS(char *numeroTel)
```

ready()

Renvoie l’état de la dernière commande SMS. En mode asynchrone, la fonction renvoie 0 si la commande est toujours en cours d’exécution, 1 en cas de réussite et plus de 1 en cas d’erreur. En mode synchrone, renvoie 1 en cas de réussite ou bien 0.

```
int SMS.ready()
```

endSMS()

Informe le modem que le message est complet et prêt à être émis. En mode asynchrone, la fonction renvoie 0 si la commande est toujours en cours d’exécution, 1 en cas de réussite et plus de 1 en cas d’erreur. En mode synchrone, renvoie 1 en cas de réussite ou bien 0.

```
int SMS.endSMS()
```

available()

Renvoie le nombre de caractères du SMS entrant ou 0 s’il n’y a aucun message entrant.

```
int SMS.available()
```

remoteNumber()

Récupère le numéro de téléphone d’un SMS entrant et le renvoie dans un tableau de caractères. Le second argument est la taille maximale du tableau à remplir. En

mode asynchrone, la fonction renvoie 0 si la commande est toujours en cours d'exécution, 1 en cas de réussite et plus de 1 en cas d'erreur. En mode synchrone, renvoie 1 en cas de réussite ou bien 0.

```
int SMS.remoteNumber(char *remote_phone, int number_size)
```

read()

Lit un octet donc un caractère d'un SMS et le renvoie sous forme d'une valeur entière ou la valeur -1 s'il n'y a pas de données.

```
int SMS.read()
```

write()

Écrit un octet (un caractère) dans un SMS.

```
int write(int character)
```

print()

Écrit un tableau de caractères dans un SMS et renvoie le nombre d'octets écrits.

```
int SMS.print(char *message)
```

peek()

Renvoie le prochain octet (caractère) à lire dans un SMS sans l'enlever, ou la valeur -1 s'il n'y a pas de données.

```
int SMS.peek()
```

flush()

Efface tous les messages émis du modem une fois que ceux en attente ont été envoyés.

```
void SMS.flush()
```

Classe GPRS

GPRS est la classe de base pour toutes les fonctions GPRS, y compris pour le client et le serveur Internet. La classe se charge aussi de l'inclusion des fichiers relatifs à la communication TCP.

attachGPRS()

Se connecte à un nom de point d'accès réseau APN (*Access Point Name*) pour ouvrir une connexion GPRS. Les opérateurs de téléphonie cellulaire utilisent les APN comme passerelles entre leur réseau et Internet. La fonction renvoie l'une des six chaînes constantes suivantes :

- `ERROR`
- `IDLE`
- `CONNECTING`
- `GSM_READY`
- `GPRS_READY`
- `TRANSPARENT_CONNECTED`

```
char *GPRS.attachGPRS(char *apn, char *user, char
*password)
```

Classe GSMClient

Cette classe sert à créer des objets clients pouvant se connecter à des serveurs et échanger des données.

ready()

Renvoie l'état de la dernière commande. En mode asynchrone, la fonction renvoie 0 en cours d'exécution, 1 en cas de réussite et plus de 1 en cas d'erreur. En mode synchrone, renvoie 1 en cas de réussite ou bien 0.

```
int GSMClient.ready()
```

connect(char *IP, int port)

Établit une connexion vers un port et une adresse d'un hôte IP spécifiés. Renvoie la valeur true en cas de réussite ou false sinon.

```
bool GSMClient.connect(char *hostip, int port)
```

beginWrite()

Débute une opération d'écriture vers le serveur connecté.

```
void GSMClient.beginWrite()
```

write()

Envoie des données au serveur connecté et renvoie le nombre d'octets émis.

```
int GSMClient.write(char data)
int GSMClient.write(char *data)
int GSMClient.write(char *data, int taille)
```

endWrite()

Termine une opération d'écriture vers un serveur.

```
void GSMClient.endWrite()
```

connected()

Renvoie l'état d'une connexion client. Le client est considéré comme toujours connecté s'il reste des données dans le tampon, même si la connexion a été refermée. Renvoie la valeur true si le client est connecté et false sinon.

```
bool GSMClient.connected()
```

read()

Lit le prochain octet disponible sur le serveur connecté et le renvoie, ou bien la valeur -1 s'il n'y a pas de données.

```
int GSMClient.read()
```

available()

Renvoie le nombre d'octets en attente de lecture sur le serveur connecté.

```
int GSMClient.available()
```

peek()

Renvoie le prochain octet entrant sans l'enlever du tampon de réception. Si vous rappelez la fonction, vous obtenez à nouveau le même caractère.

```
int GSMClient.peek()
```

flush()

Supprime les données en attente dans le tampon de réception et ramène à zéro le compteur de données en attente.

```
void GSMClient.flush()
```

stop()

Force une déconnexion d'un serveur.

```
void GSMClient.stop()
```

Classe GSMServer

Cette classe crée un serveur capable d'envoyer et de recevoir des données avec les clients connectés. Elle définit des fonctions de serveur de réseau similaires à celles des librairies Ethernet et WiFi. Certains opérateurs n'autorisent pas les connexions réseau entrantes extérieures à leur réseau.

ready()

Renvoie l’état de la dernière commande. En mode asynchrone, la fonction renvoie 0 en cours d’exécution, 1 en cas de réussite et plus de 1 en cas d’erreur. En mode synchrone, renvoie 1 en cas de réussite ou bien 0.

```
int GSMServer.ready()
```

beginWrite()

Débute une opération d’écriture vers tous les clients connectés.

```
void GSMServer.beginWrite()
```

write()

Transmet des données aux clients connectés et renvoie le nombre d’octets écrits.

```
int GSMServer.write(char data)
int GSMServer.write(char *data)
int GSMServer.write(char *data, int taille)
```

endWrite()

Arrête d’envoyer des données aux clients connectés.

```
void GSMServer.endWrite()
```

read()

Lit le prochain octet disponible depuis un client connecté et le renvoie, ou bien la valeur -1 si aucune donnée n’est disponible.

```
int GSMServer.read()
```

available()

Se met à l’écoute des requêtes de connexion des clients et renvoie le nombre de clients connectés.

```
int GSMServer.available()
```

stop()

Arrête le serveur qui n'écoute ensuite plus les demandes de connexion des clients.

```
void GSMServer.stop()
```

Classe GSMModem

Cette classe réunit des fonctions de support pour les diagnostics du modem GSM interne.

begin()

Teste le statut du modem et le redémarre. Doit être appelée avant tout appel à `getIMEI()`. Renvoie 1 si le modem est opérationnel ou bien un code d'erreur.

```
int GSMModen.begin()
```

getIMEI()

Interroge le modem pour obtenir son code IMEI (*International Mobile Equipment Identifier*) sous forme d'une chaîne. Ne doit être appelée qu'après un appel à `begin()`.

```
char *GSMModen.getIMEI()
```

Classe GSMScanner

Cette classe réunit les fonctions pour obtenir des informations d'état du réseau et de l'opérateur.

begin()

Réinitialise le modem et renvoie 1 si le modem est opérationnel ou un code d'erreur.

```
int GSMSScanner.begin()
```

getCurrentCarrier()

Renvoie le nom de l'opérateur réseau en tant que chaîne de caractères.

```
char *GSMSScanner.getCurrentCarrier()
```

getSignalStrength()

Renvoie la force de signal relative de la connexion réseau sous forme de chaîne de chiffres ASCII entre 0 et 31 (31 indique une puissance supérieure à 51 dBm), ou bien 99 en l'absence de signal.

```
char *GSMSScanner.getSignalStrength()
```

readNetworks()

Recherche la liste des opérateurs réseau accessibles sous forme d'une chaîne.

```
char *GSMSScanner.readNetworks()
```

Classe GSMPIN

Cette classe regroupe les fonctions de communication avec la carte SIM.

begin()

Réinitialise le modem en renvoyant 1 si le modem fonctionne ou un code d'erreur différent dans le cas contraire.

```
int GSMPIN.begin()
```

isPIN()

Scrute la carte SIM pour savoir si elle est verrouillée par un code PIN et renvoie 0 si le code PIN est inactif, 1 s'il est actif, -1 si le code PUK est actif et -2 en cas d'erreur.

```
int GSMPIN.isPIN()
```

checkPIN()

Vérifie qu'un code PIN est reconnu par la carte SIM. Renvoie 0 si le code est valide et -1 sinon.

```
int GSMPIN.checkPIN(char *PIN)
```

checkPUK()

Vérifie qu'un code PUK est reconnu par la carte SIM. Si le code est valide, définit un nouveau code PIN et renvoie 0. Renvoie -1 sinon.

```
int GSMPIN.checkPUK(char *PUK, char *PIN)
```

changePIN()

Change le code PIN de la carte SIM après avoir prouvé que l'ancien code est valide.

```
void GSMPIN.changePIN(char *oldPIN, char *newPIN)
```

switchPIN()

Change l'état de verrouillage PIN.

```
void GSMPIN.switchPIN(char *PIN)
```

checkReg()

Vérifie si le modem est déclaré dans un réseau GSM/GPRS. Renvoie 0 si c'est le cas, 1 si le modem est en mode itinérance (*roaming*) et -1 en cas d'erreur.

```
int GSMPIN.checkReg()
```

getPINUsed()

Teste si le verrou PIN est actif. Renvoie true si oui et false sinon.

```
bool GSMPIN.getPINUsed()
```

setPINUsed()

Définit le statut de verrouillage PIN. Si l'argument vaut true, le code PIN est verrouillé ; s'il vaut false, il est déverrouillé.

```
void GSMPIN.setPINUsed(bool used)
```

Classe GSMBand

Cette classe informe sur la plage de fréquences utilisée par le modem pour la connexion et permet de choisir une autre plage.

begin()

Réinitialise le modem et renvoie 1 si le modem fonctionne ou bien un code d'erreur.

```
int GSMBand.begin()
```

getBand()

Renvoie la plage (bande) de fréquences actuellement utilisée par le modem.

```
char *GSMBand.getBand()
```

setBand()

Choisit la page de fréquences à utiliser.

```
bool GSMBand.setBand(char *band)
```

LiquidCrystal

Cette classe permet à une carte Arduino de contrôler un module afficheur à cristaux liquides LCD. Elle est dédiée aux circuits pilotes Hitachi HD44780 (ou compatibles), les plus utilisés en mode texte. La librairie accepte une interface sur 4 ou sur 8 bits. Elle a besoin de trois broches Arduino pour le signal de

sélection RS (*Register Select*), l'activation de l'horloge (*Clock Enable*) et le contrôle lecture ou écriture R/W (*Read/Write*).

LiquidCrystal()

Ce constructeur sert à créer un objet comme instance de la classe `LiquidCrystal`. Quatre formats permettent de prendre en charge plusieurs modèles d'interface avec l'afficheur LCD.

```
LiquidCrystal(uint8_t rs, uint8_t enable, uint8_t d0,
uint8_t d1,
              uint8_t d2, uint8_t d3);

LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
uint8_t d0,
              uint8_t d1, uint8_t d2, uint8_t d3);

LiquidCrystal(uint8_t rs, uint8_t enable, uint8_t d0,
uint8_t d1,
              uint8_t d2, uint8_t d3, uint8_t d4,
              uint8_t d5, uint8_t d6, uint8_t d7);

LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable,
uint8_t d0,
              uint8_t d1, uint8_t d2, uint8_t d3,
uint8_t d4,
              uint8_t d5, uint8_t d6, uint8_t d7);
```

Description des arguments (qui peuvent être au nombre de six, sept, dix ou onze) :

- `rs` : broche Arduino reliée à la broche RS de l'afficheur.
- `rw` : broche Arduino reliée à la broche RW de l'afficheur.
- `enable` : broche Arduino reliée à la broche Enable de l'afficheur.
- `D0` à `D7` : broches Arduino reliées aux broches de données de l'afficheur.

Les broches D4, D5, D6 et D7 sont en option et ne servent pas si l'afficheur reçoit ses données sur quatre sorties numériques au lieu de huit.

Exemple :

```
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);
```

begin()

Initialise l'interface avec le contrôleur LCD sur le module LCD en précisant le nombre de colonnes et de lignes de l'afficheur et la taille du rectangle conteneur de caractère qui est par défaut de 5 pixels sur 8. La fonction doit être appelée avant toute autre fonction LCD.

```
void lcd.begin(uint8_t cols, uint8_t rows, uint8_t
charsize = LCD_5x8DOTS)
```

clear()

Vide l'écran LCD et ramène le curseur en position de départ de la première ligne.

```
void lcd.clear()
```

home()

Ramène le curseur en position de départ de la première ligne sans effacer le contenu de l'écran (voir `clear()` à ce sujet).

```
void home()
```

setCursor()

Amène le curseur à la position spécifiée par les arguments de colonne et de ligne (*row*).

```
void setCursor(uint8_t col, uint8_t ligne)
```

write()

Envoie un octet (type `char`) vers l'écran LCD et renvoie le nombre d'octets écrits, donc 1 (ou 0 en cas d'erreur).

```
size_t write(uint8_t)
```

print()

Cette fonction est celle du code standard Arduino *runtime* avr-gcc qui est surchargé pour accepter plusieurs types de données. Les formats d'appels définis dans le fichier *Print.h* (voir dans *hardware/arduino/avr/cores/arduino/*) sont les suivants :

```
size_t print(const FlashStringHelper *);
size_t print(const String &);
size_t print(const char[]);
size_t print(char);
size_t print(unsigned char, int = DEC);
size_t print(int,           int = DEC);
size_t print(unsigned int,  int = DEC);
size_t print(long,          int = DEC);
size_t print(unsigned long, int = DEC);
size_t print(double,        int = 2);
size_t print(const Printable&);
```

cursor()

Rend visible le curseur (un caractère de soulignement) à la position d'affichage du prochain caractère.

```
void cursor()
```

noCursor()

Masque le curseur sans modifier la position d'affichage du prochain caractère.

```
void noCursor()
```

blink()

Fait clignoter le curseur.

```
void blink()
```

noBlink()

Arrête le clignotement du curseur.

```
void noBlink()
```

display()

Réactive l’affichage LCD au cas où il aurait été désactivé par un appel à `noDisplay()`. Restaure la position du curseur et le contenu antérieur ou mis à jour pendant le masquage de l’écran.

```
void display()
```

noDisplay()

Désactive l’affichage LCD sans rien changer au contenu qui est affiché à ce moment.

```
void noDisplay()
```

scrollDisplayLeft()

Fait défiler le texte d’une position vers la gauche.

```
void scrollDisplayLeft()
```

scrollDisplayRight()

Fait défiler le texte d’une position vers la droite.

```
void scrollDisplayRight()
```

autoscroll()

Active le défilement automatique. Chaque ajout d'un caractère pousse ceux présents d'une position vers la gauche ou la droite, selon le sens d'écriture sélectionné avec les fonctions `leftToRight()` et `rightToLeft()`.

```
void autoscroll()
```

noAutoscroll()

Désactive le défilement automatique.

```
void noAutoscroll()
```

leftToRight()

Sélectionne le sens de défilement du contenu de l'afficheur de la gauche vers la droite. Le défilement automatique doit être actif.

```
void leftToRight()
```

rightToLeft()

Sélectionne le sens de défilement du contenu de l'afficheur de la droite vers la gauche. Le défilement automatique doit être actif.

```
void rightToLeft()
```

createChar()

Permet de définir un caractère ou un symbole sur mesure dans une matrice de 5 pixels de large sur 8 pixels de haut. Le dessin du caractère est fourni dans un tableau de huit octets, un par ligne. Seuls les cinq bits les moins significatifs (poids faibles) de l'octet sont pris en compte.

```
void createChar(uint8_t, uint8_t[])
```

SD

La librairie SD réunit les fonctions permettant de lire et d'écrire sur une carte mémoire au format SD ou microSD (seul le format physique les distingue). Cette librairie est basée sur la librairie *sdfatlib* de William Greiman.

La carte mémoire est considérée comme un disque dur dont l'espace de stockage est configuré selon le format FAT16 ou FAT32. Elle utilise donc des noms de fichiers sur huit caractères suivis d'un point et de trois caractères d'extension. Un chemin d'accès peut être transmis en préfixe du nom du fichier, mais les barres obliques de séparation des différents sous-répertoires sont des barres penchées à droite (*obliques*) comme celles utilisées sous Unix et sous Linux, et non des barres obliques vers la gauche (*obliques inverses*) qui sont en vigueur sous MS-DOS et Windows.

Le dialogue avec la carte SD utilise l'interface SPI sur les broches numériques D11, D12 et D13 d'une carte Arduino standard. Une autre broche, en général la broche D10, sert de broche de sélection. Une autre broche de sélection peut être choisie, mais dans ce cas, il faut laisser libre la broche D10 en tant que sortie pour que la librairie puisse fonctionner.

Classe SD

Cette classe définit les fonctions permettant d'accéder à la carte SD afin de manipuler les fichiers et les sous-répertoires.

begin()

Configure la librairie SD et l'interface avec la carte mémoire. Vous pouvez fournir un argument en option, `csPin`, pour choisir la broche de sélection, la valeur par défaut étant la broche D10, comme défini par la constante SD_CHIP_SELECT_PIN. Il faut absolument appeler cette fonction avant n'importe quelle autre fonction de la librairie. Elle renvoie la valeur true si elle a réussi ou la valeur false sinon.

```
bool SD.begin(uint8_t csPin = SD_CHIP_SELECT_PIN);
```

exists()

Teste la présence d'un fichier ou d'un répertoire sur la carte mémoire. L'argument peut être un chemin d'accès complet ou un simple nom de fichier. Renvoie la valeur true si l'élément existe sur la carte ou la valeur false sinon.

```
bool SD.exists(char *cheminFichier);
```

mkdir()

Crée un sous-répertoire sur la carte mémoire, y compris les sous-répertoires intermédiaires si nécessaire, puis renvoie la valeur true si la création a réussi ou la valeur false sinon.

```
bool SD.mkdir(char *cheminFichier);
```

open()

Ouvre un fichier en lecture ou écriture sur la carte mémoire. Si vous ne spécifiez pas l'argument mode, le fichier est ouvert en lecture. La fonction renvoie un objet du type `File` qui peut être testé en tant que valeur logique booléenne. Cette valeur est égale à false si le fichier n'a pas pu être ouvert. Les deux modes possibles sont FILE_READ et FILE_WRITE.

```
File SD.open(const char *nomfic, uint8_t mode =
FILE_READ);
```

remove()

Supprime un fichier de la carte mémoire. L'argument *`cheminFichier`* peut-être un chemin d'accès absolu. Renvoie la valeur true si l'opération a réussi ou la valeur false sinon.

```
bool SD.remove(char *cheminFichier);
```

rmdir()

Supprime un répertoire de la carte SD, ce répertoire devant être vide. Renvoie la valeur true en cas de réussite ou false en cas d'erreur. C'est par exemple le cas

lorsque le répertoire n’est pas vide.

```
bool SD.rmdir(char *cheminFichier);
```

Classe File

Cette classe réunit les fonctions permettant de lire et d’écrire un fichier de la carte mémoire. C’est la fonction `SD.open()` qui permet d’obtenir un objet du type `File` :

```
fname = SD.open("fichier.txt", FILE_WRITE);
```

available()

Renvoie le nombre d’octets pouvant être lus dans un fichier.

```
int fname.available()
```

close()

Referme un fichier en garantissant que les données en attente sont écrites.

```
void fname.close()
```

flush()

Force l’écriture des données en attente depuis le tampon fichier vers le fichier sans refermer ce dernier.

```
void fname.flush()
```

peek()

Lit un seul octet depuis un fichier sans mettre à jour le pointeur interne. Si vous appelez la même fonction, vous récupérez toujours le même octet.

```
int fname.peek()
```

position()

Renvoie la position courante dans le fichier, celle qui désigne l'endroit du prochain octet qui sera lu ou écrit.

```
uint32_t fname.position()
```

print()

Écrit des données dans un fichier qui a d'abord été ouvert en mode écriture. Les données peuvent être du type `char`, `byte` (uint8_t), `int`, `long` ou une chaîne. Vous pouvez spécifier une base de notation parmi BIN, DEC, OCT et HEX. La fonction renvoie le nombre d'octets qui ont été écrits.

```
int fname.print(data)

int fname.print(char *data, int BASE)
```

Note : dans le deuxième exemple ci-dessus, il s'agit d'une chaîne (`* data`).

println()

Écrit de la même façon que la fonction `print()` des données dans un fichier déjà ouvert, mais termine avec un couple de caractères Retour Chariot et Saut de Ligne. Accepte les mêmes types que `print()` et les mêmes bases de notation numérique.

```
int fname.println()
int fname.println(data)
int fname.println(data, int BASE)
```

seek()

Modifie la position du pointeur de fichier. La valeur doit être entre zéro et la taille du fichier, y compris cette valeur limite. Renvoie la valeur true en cas de réussite ou la valeur false sinon, par exemple lorsque vous tentez d'aller au-delà de la fin du fichier.

```
bool fname.seek(uint32_t pos)
```

size()

Renvoie en octets la taille du fichier.

```
uint32_t fname.size()
```

read()

Lit le prochain octet du fichier ou renvoie la valeur -1 s'il n'y a pas de données.

```
int fname.read(void *buf, uint16_t nbyte)
```

write()

Écrit des données dans un fichier. Il peut s'agir d'un seul octet ou d'un objet du type `byte`, `char` ou chaîne. L'argument de taille définit la quantité de données à écrire sur la carte. La fonction renvoie le nombre d'octets écrits.

```
size_t fname.write(uint8_t)
size_t fname.write(const uint8_t *buf, size_t taille)
```

isDirectory()

Renvoie la valeur true si l'objet spécifié est un répertoire ou la valeur false sinon.

```
bool fname.isDirectory()
```

openNextFile()

Ouvre le prochain fichier trouvé dans le répertoire et renvoie une nouvelle instance de l'objet du type `File`.

```
File openNextFile(uint8_t mode = O_RDONLY)
```

rewindDirectory()

Cette fonction s'utilise avec la précédente pour renvoyer le premier fichier ou sous-répertoire trouvé dans un répertoire.

```
void fname.rewindDirectory()
```

Servo

Cette librairie réunit des fonctions pour contrôler un servomoteur, tel que ceux utilisés dans les modèles réduits radiocommandés. Dès que vous avez créé une instance de la classe `Servo`, vous devez utiliser la fonction `attach()` pour choisir le numéro de la broche à utiliser avec le servo. Le contrôle du servomoteur est réalisé grâce à un train d'impulsions PWM généré en tâche de fond. Voici comment créer une instance de la classe :

```
Servo servo;
```

attach()

Connecte un servomoteur physique à une broche d'entrée-sortie. Dans la seconde syntaxe, vous pouvez spécifier les valeurs minimale et maximale de durée d'écriture en microsecondes. La fonction renvoie le numéro de canal ou la valeur zéro en cas d'échec.

```
uint8_t servo.attach(int pin)
uint8_t servo.attach(int pin, int min, int max)
```

write()

Modifie l'angle du servomoteur en degrés. Si la valeur est supérieure à 200, elle est considérée comme signifiant une largeur d'impulsion exprimée en microsecondes.

```
void servo.write(int valeur)
```

read()

Renvoie la dernière largeur d'impulsion envoyée au servomoteur sous forme d'un angle entre 0 et 180 degrés.

```
int servo.read()
```

writeMicroseconds()

Définit la largeur d'impulsion de servomoteur en microsecondes.

```
void servo.writeMicroseconds(int valeur)
```

readMicroseconds()

Renvoie la largeur d'impulsion actuelle en microsecondes.

```
int servo.readMicroseconds()
```

attached()

Renvoie la valeur true si l'objet servomoteur est bien connecté à un servomoteur physique.

```
bool servo.attached()
```

detach()

Fait cesser la génération d'impulsions d'un objet servomoteur attaché sur sa broche de sortie.

```
void servo.detach()
```

SPI

Comme son nom l'indique, cette librairie permet d'exploiter une interface de périphérique série SPI avec les périphériques compatibles. Elle permet également d'établir une communication entre deux microcontrôleurs.

La classe nommée `SPISettings` sert à configurer le port SPI. Les différents arguments sont réunis dans un unique objet nommé `SPISettings` qui est

transmis à la méthode `SPI.begin-Transaction()`.

```
  SPISettings(uint32_t clock, uint8_t bitOrder, uint8_t
dataMode)
```

Exemple :

```
SPISettings spiset(uint32_t clock, uint8_t bitOrder,
uint8_t dataMode)
```

beginTransaction()

Initialise une interface SPI au moyen des paramètres définis dans un objet `SPISettings`.

```
void SPI.beginTransaction(SPISettings)
```

endTransaction()

Arrête la communication avec une interface SPI. En général, cette méthode est appelée juste après libération de la broche de sélection, afin que d'autres librairies puissent utiliser l'interface SPI.

```
void SPI.endTransaction()
```

usingInterrupt()

Méthode utilisée pour exploiter un dialogue SPI avec une gestion d'interruption.

```
void SPI.usingInterrupt(uint8_t interruptNumber)
```

begin()

Active la librairie SPI en initialisant l'interface. Place les broches SCK, MOSI et SS en mode sortie. Force à l'état bas les broches SCK et MOSI tout en forçant à l'état haut la broche SS.

```
void SPI.begin()
```

end()

Désactive l’interface SPI sans modifier les modes des broches.

```
void SPI.end()
```

transfer()

Transfère un octet *via* l’interface SPI, en émission ou en réception.

```
uint8_t SPI.transfer(uint8_t data)
```

setBitOrder()

Détermine l’ordre des bits dans un octet qui doit transiter par l’interface SPI. Vous avez le choix entre LSBFIRST (bit le moins significatif d’abord) et MSBFIRST (bit le plus significatif d’abord, celui de poids fort). N’utilisez pas cette méthode avec vos nouveaux projets, mais configurez l’interface SPI au moyen de la méthode `beginTransaction()`.

```
void SPI.setBitOrder(uint8_t bitOrder)
```

setClockDivider()

Détermine le diviseur de l’horloge SPI relativement à celle du système. Pour les cartes Arduino, les diviseurs autorisés sont 2, 4, 8, 16, 32, 64 et 128. N’utilisez pas cette méthode avec vos nouveaux projets, mais configurez l’interface SPI au moyen de la méthode `beginTransaction()`.

```
void SPI.setClockDivider(uint8_t clockDiv)
```

setDataMode()

Définit la polarité et la phase de l'horloge de l'interface SPI. N'utilisez pas cette méthode avec vos nouveaux projets, mais configurez l'interface SPI au moyen de la méthode `beginTransaction()`.

```
void SPI.setDataMode(uint8_t dataMode)
```

SoftwareSerial

Cette librairie permet de réaliser des communications série par logiciel en se servant des broches numériques Arduino. Autrement dit, elle permet d'émuler une interface série réelle. Elle est particulièrement utile lorsque vous avez besoin de plus d'une interface série et que le port USB du microcontrôleur est déjà occupé pour une autre fonction, par exemple pour l'interface USB.

La librairie permet d'exploiter plusieurs interfaces série, chacune ayant une vitesse d'au maximum 115200 bits par seconde. Une seule des interfaces série peut recevoir des données à chaque instant. Les broches d'entrée-sortie utilisées doivent supporter les interruptions de changement de broche. La librairie définit un tampon de réception de 64 octets pour chaque instance d'interface.

Pour réaliser des opérations série, il faut d'abord créer un objet de la classe `SoftwareSerial`. Le constructeur doit recevoir les numéros des broches numériques à utiliser pour la réception Rx et pour l'émission Tx.

```
SoftwareSerial(uint8_t rxPin, uint8_t txPin, bool
inv_logic = false)
```

Exemple :

```
SoftwareSerial serial = SoftwareSerial(rxPin, txPin)
```

available()

Renvoie le nombre d'octets qui sont en attente de lecture dans le tampon d'entrée série.

```
int serial.available()
```

begin()

Règle la vitesse de l’interface série en bauds. Les valeurs autorisées sont 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 57600 et 115200.

```
void serial.begin(long speed)
```

isListening()

Interroge l’interface série pour savoir si elle est en attente de réception et renvoie la valeur true si c’est le cas, ou sinon la valeur false.

```
bool serial.isListening()
```

overflow()

Si les données reçues dépassent les 64 octets du tampon d’entrée de l’objet `SoftwareSerial`, un drapeau logiciel est armé, et cette méthode renvoie la valeur de ce drapeau. Si la valeur renvoyée est égale à true, c’est qu’il y a eu débordement. Le fait d’appeler la méthode remet le drapeau à zéro.

```
bool serial.overflow()
```

peek()

Renvoie le caractère du tampon d’entrée le plus ancien, sans l’enlever du tampon. Le même caractère est renvoyé si vous rappelez la même méthode. Si le tampon est vide, la méthode renvoie la valeur -1.

```
int serial.peek()
```

read()

Récupère un caractère du tampon d’entrée puis le supprime du tampon. Cette fonction est normalement utilisée dans une boucle. Elle renvoie la valeur -1 s’il n’y a pas de données. Une seule instance de `SoftwareSerial` peut recevoir

des données à tout moment. Pour choisir l'interface à exploiter, vous utilisez la méthode `listen()`.

```
int serial.read()
```

listen()

Active la réception de données pour une instance de `SoftwareSerial`. Une seule instance peut recevoir des données à chaque instant. L'objet dont vous avez appelé la fonction `listen()` en dernier est l'auditeur actif. Les données qui arrivent sur les autres interfaces sont perdues.

```
bool serial.listen()
```

print()

Cette fonction a le même emploi que la fonction homonyme de la classe `Serial`. Elle accepte les mêmes types de données, donc les types `char`, `byte` (uint8_t), `int`, `long` et chaîne. Elle renvoie le nombre d'octets qui ont été écrits.

```
int serial.print(data)
```

println()

Cette méthode a le même comportement que celle de la classe `Serial` en ajoutant un couple Retour Chariot/Saut de ligne en fin d'écriture. Elle renvoie le nom d'octets écrits. Si vous ne fournissez pas de données, elle émet néanmoins le couple Retour Chariot/Saut de ligne.

```
int serial.println(data)
```

write()

Émet des données depuis l’interface série sous forme d’octets bruts. Fonctionne de la même manière que la méthode homonyme de la classe `Serial`. Renvoie le nombre d’octets écrits.

```
size_t serial.write(uint8_t byte)
```

Stepper

Les fonctions de cette librairie permettent de contrôler des moteurs unipolaires et bipolaires. Il faut bien sûr que le matériel soit dimensionné pour supporter les courants qui vont circuler.

La librairie offre deux formats de constructeur, un pour les moteurs unipolaires et l’autre pour les moteurs bipolaires. Dans les deux cas, cela crée un objet qui est une instance de la classe `Stepper`. Le constructeur doit être appelé au début du programme, avant les deux fonctions obligatoires `setup()` et `loop()`. L’argument *`steps`* définit le nombre de pas pour un tour complet de l’axe du moteur. Les autres arguments servent à définir les broches numériques à utiliser.

```
Stepper(int steps, int pin1, int pin2);
Stepper(int steps, int pin1, int pin2, int pin3, int
pin4);
```

Exemple :

```
Stepper stepdrv = Stepper(100, 3, 4);
```

setSpeed()

Définit la vitesse de rotation en tours par minute sans faire tourner le moteur. Ce paramétrage influe sur la fonction `step()`.

```
void stepdrv.setSpeed(long speed);
```

step()

Demande au moteur de tourner d'un certain nombre de pas. Une valeur positive fait tourner dans un sens et une valeur négative dans l'autre.

```
void stepdrv.step(int stepcount);
```

TFT

Cette librairie permet de gérer un afficheur en technologie TFT, et permet donc d'afficher du texte, des images et des formes. Elle est présente à partir de la version 1.0.5 de l'atelier Arduino et simplifie l'affichage de graphiques. Elle est inspirée d'une autre librairie, Adafruit ST7735H que vous pouvez trouver sur le site de référence GitHub (https://github.com/adafruit/Adafruit-ST7735-Library).
Cette autre librairie est elle-même basée sur la librairie Adafruit GFX également disponible sur GitHub (https://github.com/adafruit/Adafruit-GFX-Library).

Cette librairie a été conçue pour fonctionner avec l'interface SPI des microcontrôleurs AVR. Si le module TFT est doté d'un lecteur de carte SD, la librairie SD peut être mise à profit, mais il faut prévoir un signal de sélection distinct sur l'Arduino. Vous devez mentionner la librairie TFT dans tous les croquis qui l'utilisent.

Classe TFT

Le constructeur de cette classe est disponible en deux formats. Le premier est à utiliser lorsque vous exploitez l'interface SPI sur les broches standard (SPI matériel). Le second format permet de choisir quelles broches utiliser :

```
TFT(uint8_t cs, uint8_t dc, uint8_t rst)
TFT(uint8_t cs, uint8_t dc, uint8_t mosi, uint8_t sclk,
uint8_t rst)
```

Explications :

- cs : broche de sélection de circuit type select.
- dc : sélection du mode data ou command.

- rst : broche de Reset.
- mosi : broche pour le signal MOSI si vous n'utilisez pas le SPI matériel.
- sclk : broche pour le signal d'horloge si vous n'utilisez pas le SPI matériel.

Exemple :

```
#define cs 10
#define dc 9
#define rst 8
TFT disp = TFT(cs, ds, rst);
```

Notez que la version Esplora de la librairie TFT utilise les broches prédéfinies. Il suffit dans ce cas de créer l'instance d'objet TFT comme ceci :

```
EsploraTFT disp = EsploraTFT;
```

begin()

Initialise les composants de la librairie TFT. Cette méthode doit être appelée avant n'importe quelle autre. Vous placez l'appel en général dans la fonction `setup()` du croquis.

```
void disp.begin()
```

background()

Repeint tout l'écran avec une couleur unie, ce qui permet de le vider. Sachez que l'écran ne peut pas afficher 250 niveaux par couleur. Les fonctions utilisent cinq bits (soit 32 nuances) pour les bleus et les rouges et six bits pour les verts.

```
void disp.background(uint8_t red, uint8_t green, uint8_t
blue)
```

stroke()

Cette méthode est appelée avant tout affichage pour définir les couleurs des lignes et des bordures. Le nombre de couleurs est limité, comme pour la fonction

`background()`.

```
void disp.stroke(uint8_t red, uint8_t green, uint8_t
blue)
```

noStroke()

Supprime toutes les couleurs de bordures.

```
void disp.noStroke()
```

fill()

Définit la couleur de remplissage des objets du texte. Utilise cinq bits (soit 32 nuances) pour les bleus et les rouges et six bits pour les verts.

```
void disp.fill(uint8_t red, uint8_t green, uint8_t blue)
```

noFill()

Désactive le remplissage en couleurs des objets et du texte.

```
void disp.noFill()
```

setTextSize()

Règle la taille du texte (le corps) qui sera affiché par les appels à `text()`. La valeur par défaut est égale à 1, ce qui correspond à 10 pixels. Chaque augmentation de la valeur augmente la hauteur du texte de 10 pixels.

```
void disp.setTextSize(uint8_t size)
```

text()

Envoie du texte à l’écran aux coordonnés spécifiées. La couleur du texte a été définie par un appel préalable à `fill()`.

```
void disp.text(const char * text, int16_t x, int16_t y),
```

point()

Trace un point à l'endroit spécifié. La couleur du point sera celle définie par un appel à `fill()`.

```
void disp.point(int16_t x, int16_t y)
```

line()

Trace une ligne entre les points de départ et d'arrivée dans la couleur définie par un appel à `stroke()`.

```
void disp.line(int16_t x1, int16_t y1, int16_t x2,
int16_t y2)
```

rect()

Dessine un rectangle à partir du coin supérieur gauche avec la largeur et la hauteur spécifiées.

```
void disp.rect(int16_t x, int16_t y, int16_t large,
int16_t haut)
```

width()

Renvoie la largeur de l'écran TFT en pixels.

```
int disp.width()
```

height()

Renvoie la hauteur de l'écran TFT en pixels.

```
int disp.height()
```

circle()

Trace un cercle à partir d'un centre en x, y et d'un rayon.

```
int disp.circle(int16_t x, int16_t y, int16_t rayon)
```

image()

Affiche une image chargée depuis une carte mémoire SD à la position indiquée.

```
void image(PImage image, int xpos, int yPos)
```

loadImage()

Crée une instance d'objet `Pimage` à partir du nom de fichier image fourni. L'image doit être au format BMP sur 24 bits et se trouver dans le répertoire racine de la carte mémoire. Cette méthode appartient à l'objet `PImage`. Voici comment l'utiliser :

```
PImage disp.loadImage(char *imgname)
```

PImage

Cette classe simplifie l'affichage d'images bitmap sur un écran TFT.

PImage.height()

Une fois qu'un fichier image est inséré dans un objet `PImage`, vous pouvez l'interroger pour connaître sa hauteur. Cette méthode la renvoie sous forme d'une valeur entière de type `int`.

PImage.width()

Renvoie la largeur de l'image insérée en tant que valeur `int`.

PImage.isValid()

Renvoie la valeur true si l'objet image contient un fichier bitmap exploitable ou la valeur false sinon.

WiFi

Cette librairie permet à votre carte Arduino de se connecter à un réseau sans fil. Cette section ne donne pas de description pour toutes les fonctions, car la plupart

d’entre elles sont soient absolument identiques, soit très similaires à celles de la librairie `Ethernet` déjà décrite. Dans l’aide en ligne des premières versions de l’atelier Arduino (et plusieurs distributions Linux sont encore accompagnées d’une telle version), il n’y avait pas de documentation concernant la librairie WiFi, ce qui a été corrigé depuis. Par exemple, cette documentation n’était pas disponible dans la version de l’atelier livrée avec ma version de Kubuntu.

La librairie WiFi a besoin de la librairie SPI pour dialoguer avec le module sans fil et la carte mémoire SD éventuelle. Les cartes Arduino de base (décrites dans le Chapitre 4) dialoguent avec le bouclier WiFi par les broches SPI 10, 11, 12 et 13. Les cartes au format Mega utilisent les broches 10, 50, 51 et 52. La broche 7 du bouclier WiFi Arduino est utilisée pour le signal de confirmation de connexion (*handshake*) entre Arduino et WiFi. Il ne faut donc pas l’utiliser pour autre chose. Cette limitation peut s’appliquer à d’autres modèles de boucliers WiFi.

Le bouclier Arduino WiFi peut jouer le rôle de serveur ou de client. La librairie supporte les modes de cryptage WEP etWPA2 Personal mais pas le mode WPA2 Enterprise. Notez que le bouclier WiFi ne saura pas établir la connexion si le serveur ne diffuse pas son identifiant SSID. Comme la librairie Ethernet, la librairie WiFi définit cinq classes C++, dont la plupart héritent de classes parentes dont vous n’avez pas à vous soucier en général. Si vous avez besoin d’étudier ces classes de bas niveau, allez les consulter dans le sous-répertoire *libraries/WiFi* du code source Arduino. Voici les cinq classes de la librairie WiFi avec les fonctions membres à accès public correspondantes :

- **Classe WiFi**
  - `begin()`
  - `disconnect()`
  - `config()`
  - `setDNS()`
  - `SSID()`
  - `BSSID()`
  - `RSSI()`

- encryptionType()
- scanNetworks()
- getSocket()
- macAddress()

- **Classe IPAddress**
  - localIP()
  - subnetMask()
  - gatewayIP()

- **Classe Server**
  - WiFiServer()
  - begin()
  - available()
  - write()
  - print()
  - println()

- **Classe Client**
  - WiFiClient()
  - connected()
  - connect()
  - write()
  - print()
  - println()

- available()
- read()
- flush()
- stop()

- **Classe UDP**
  - begin()
  - available()
  - beginPacket()
  - endPacket()
  - write()
  - parsePacket()
  - peek()
  - read()
  - flush()
  - stop()
  - remoteIP()
  - remotePort()

Le bouclier Arduino WiFi utilise le circuit intégré HDG204 802.11b/g. D'autres boucliers, et notamment celui d'Adafruit, utilisent un autre circuit, le TI CC3000 WiFi, et donc une autre librairie. Les fonctions devraient être très similaires, mais il faudra tenir compte des quelques divergences. Vous trouverez la librairie Adafruit sur GitHub. Vous pouvez également consulter le site Web d'Adafruit.

Classe WiFi

Voici un bref résumé des classes de la librairie WiFi. Pour tout détail, reportez-vous à la description de la librairie Ethernet dans ce même chapitre.

La classe WiFi regroupe toutes les fonctions pour initialiser la librairie et faire le paramétrage réseau. La définition de la classe se trouve dans le fichier d'en-tête *WiFi.h*.

```
WiFiClass()
int begin(char* ssid)
int begin(char* ssid, uint8_t key_idx, const char* key)
int begin(char* ssid, const char *passphrase)

int disconnect()

void config(IPAddress local_ip)
void config(IPAddress local_ip, IPAddress dns_server)
void config(IPAddress local_ip, IPAddress dns_server,
IPAddress gateway)
void config(IPAddress local_ip, IPAddress dns_server,
IPAddress gateway,
  IPAddress subnet)
void setDNS(IPAddress dns_server1)
void setDNS(IPAddress dns_server1, IPAddress dns_server2)

char* SSID()
char* SSID(uint8_t networkItem)
uint8_t* BSSID(uint8_t* bssid)
int32_t RSSI()
int32_t RSSI(uint8_t networkItem)

uint8_t encryptionType()
uint8_t encryptionType(uint8_t networkItem)
int8_t scanNetworks()

static uint8_t getSocket()
uint8_t* macAddress(uint8_t* mac)
static char* firmwareVersion()
uint8_t status()
int hostByName(const char* aHostname, IPAddress& aResult)
```

Classe IPAddress

Comme la classe de même nom de la librairie Ethernet, cette classe constitue un conteneur pour les informations de configuration du réseau.

```
IPAddress localIP()
IPAddress subnetMask()
IPAddress gatewayIP()
```

Classe Server

Cette classe permet de créer un objet serveur qui se place en attente de connexion de ses clients. Le client peut être une autre carte Arduino dotée d'un bouclier WiFi, un PC, un portable ou n'importe quel périphérique capable de dialoguer en WiFi. Voyez la description de la classe `EthernetServer` dans la section consacrée à la librairie Ethernet pour les fonctions `print()` et `println()`.

```
WiFiServer(uint16_t)
WiFiClient available(uint8_t* status = NULL)

void begin()
int print(data)
int print(data, base)

int println()
int println(data)
int println(data, base)

size_t write(uint8_t)
size_t write(const uint8_t *buf, size_t size)
uint8_t status()
```

Classe Client

Cette classe sert à créer un client WiFi pouvant se connecter à un serveur pour envoyer et recevoir des données. Le serveur peut être une autre carte Arduino, un ordinateur ou n'importe quel appareil capable de dialoguer en WiFi en tant que serveur. Voyez la description de la classe `EthernetServer` dans la

section consacrée à la librairie Ethernet pour les fonctions `print()` et `println()`.

```
WiFiClient()
WiFiClient(uint8_t sock)

uint8_t connected()

int connect(IPAddress ip, uint16_t port)
int connect(const char *host, uint16_t port)

size_t write(uint8_t)
size_t write(const uint8_t *buf, size_t size)

int print(data)
int print(data, base)

int println()
int println(data)
int println(data, base)

int available()
int read()
int read(uint8_t *buf, size_t size)
int peek()

void flush()
void stop()
```

Classe UDP

Cette classe permet d'échanger des messages courts selon le protocole UDP. Alors que le protocole TCP/IP est de type flux (*stream*), sans début ni fin marquées, le protocole UDP est de type datagramme. Chaque élément de données est un paquet qui doit pouvoir être stocké dans les limites de taille d'un paquet de datagramme. UDP ne propose aucune détection d'erreur et ne garantit pas le bon acheminement. C'est une technique rapide et assez simple pour échanger des données entre des machines lorsqu'il s'agit de petits volumes de données non critiques, ou lorsque le logiciel de niveau supérieur peut se charger de la détection des erreurs et des décisions de réémission.

```
WiFiUDP()
uint8_t begin(uint16_t)
void stop()

int beginPacket(IPAddress ip, uint16_t port)
int beginPacket(const char *host, uint16_t port)
int endPacket()

size_t write(uint8_t)
size_t write(const uint8_t *tampon, size_t size)

int parsePacket()
int available()
int read()
int read(unsigned char* tampon, size_t len)
int peek()
void flush()

IPAddress remoteIP()
uint16_t remotePort()
```

Wire

Cette librairie permet d'établir un dialogue avec un appareil utilisant le protocole sur deux fils TWI (également appelé I2C). Ce protocole a été décrit dans les Chapitres 2 et 3 concernant les capacités des microcontrôleurs AVR à ce niveau. Dans le Chapitre 8, je présente plusieurs boucliers permettant d'exploiter le protocole I2C avec Arduino.

Le petit tableau qui suit montre quelles broches sont utilisées pour le dialogue TWI selon le type de carte Arduino. Les brochages des cartes ont été fournis dans le Chapitre 4.

La classe principale de cette librairie se nomme `TwoWire`. Voici un exemple :

```
TwoWire twi = TwoWire()
```

Carte	SDA	SCL
Uno, Ethernet	A4	A5

Mega2560	20	21
Leonardo	2	3

begin()

Initialise la librairie et active l'interface I2C soit en mode maître, soit en mode esclave. Si vous ne fournissez pas d'adresse, l'interface fonctionne par défaut en mode maître.

```
void twi.begin()
void twi.begin(uint8_t addr)
void twi.begin(int addr)
```

requestFrom()

Cette fonction est utilisée par le maître de l'interface pour demander à un esclave de fournir des données. Vous récupérez les octets au moyen des fonctions `available()` et `read()`. Cette fonction renvoie le nombre d'octets lus à cette adresse.

```
uint8_t twi.requestFrom(uint8_t addr, uint8_t quantity)
uint8_t twi.requestFrom(uint8_t addr, uint8_t quantity,
uint8_t stop)
uint8_t twi.requestFrom(int addr, int quantity)
uint8_t twi.requestFrom(int addr, int quantity, int stop)
```

beginTransmission()

Démarre une transmission vers un esclave I2C à l'adresse spécifiée. Les données sont mises en file d'attente de transmission au moyen de la fonction `write()` puis transmises au moyen de la fonction `endTransmission()`.

```
void twi.beginTransmission(uint8_t addr)
void twi.beginTransmission(int addr)
```

endTransmission()

Transmets les octets mis en file d'attente par `write()` vers un esclave puis referme la transmission ouverte avec `beginTransmission()`.

```
uint8_t twi.endTransmission()
uint8_t twi.endTransmission(uint8_t stop)
```

write()

Stocke les données demandées dans une file d'attente de transmission depuis le maître vers un esclave ou depuis un esclave vers le maître en réponse à une demande de données. Renvoie le nombre d'octets stockés dans la file d'attente.

```
size_t twi.write(uint8_t data)
size_t twi.write(const uint8_t *data)
size_t twi.write(const uint8_t *data, size_t len)
```

available()

Renvoie le nombre d'octets disponibles en lecture pour `read()`. La fonction est appelée par un maître après un appel à `requestFrom()` et par un esclave après détection d'un événement de réception de données.

```
int twi.available()
```

read()

Lit un octet qui a été transféré d'un maître à un esclave, ou l'inverse.

```
int twi.read()
```

onReceive()

Définit la fonction gestionnaire (*handler*) qu'il faut appeler lorsque le périphérique esclave va recevoir des données depuis le maître.

```
void twi.onReceive(void (*)(int))
```

onRequest()

Déclare la fonction qu'il faut appeler lorsque le maître demande des données à un esclave.

```
void twi.onRequest(void (*)(void))
```

Esplora

Cette librairie est dédiée à la carte de même nom. Elle fournit un ensemble de fonctions pour simplifier l'interface avec les capteurs et actionneurs embarqués sur la carte Esplora. Le brochage a été décrit dans le Chapitre 4.

Voici les capteurs qui sont disponibles sur la carte :

- un joystick analogique à deux axes ;
- un bouton poussoir sur le joystick ;
- quatre boutons poussoirs ;
- un microphone ;
- un détecteur de lumière ;
- un capteur de température ;
- un accéléromètre à trois axes ;
- deux connecteurs d'entrée au format TinkerKit ;

La carte comporte également des actionneurs ;

- une diode LED rouge/verte/bleue très lumineuse ;
- un buzzer piézoélectrique ;
- deux connecteurs de sortie au format TinkerKit.

Esplora()

Crée une instance de la classe `Esplora`.

```
Esplora esp = Esplora()
```

readSlider()

Renvoie une valeur entière qui indique la position actuelle du curseur de réglage entre 0 et 1023.

```
unsigned int esp.readSlider()
```

readLightSensor()

Renvoie une valeur entière qui correspond à la quantité de lumière captée par le capteur de lumière de la carte Esplora.

```
unsigned int esp.readLightSensor()
```

readTemperature()

Envoie une valeur entière signée pour la température ambiante en degrés Fahrenheit ou Celsius. L'argument *`echelle`* peut recevoir l'une des deux constantes DEGREES_C ou DEGREES_F. La plage de température va de -40 ºC à 150 ºC, et de -40 ºF à 302 ºF.

```
int esp.readTemperature(const byte echelle);
```

readMicrophone()

Renvoie une valeur entière qui décrit la quantité de bruit ambiant détecté par le microphone entre 0 et 1023.

```
unsigned int esp.readMicrophone()
```

readJoystickSwitch()

Lit l'état du bouton de joystick en renvoyant soit 0, soit 1023. Voyez aussi la fonction `readJoystickButton()`.

```
unsigned int esp.readJoystickSwitch()
```

readJoystickButton()

Lit l'état du bouton de joystick et renvoie soit LOW, soit HIGH. La fonction est identique à la précédente, mais elle renvoie une valeur cohérente avec la fonction `readButton()`.

```
bool readJoystickButton()
```

readJoystickX()

Renvoie la position en X du joystick entre -512 et 512.

```
int esp.readJoystickX()
```

readJoystickY()

Renvoie la position en Y du joystick entre -512 et 512.

```
int esp.readJoystickY()
```

readAccelerometer()

Renvoie la valeur relative pour l'axe demandé. Les valeurs possibles pour l'argument *`axe`* sont X_AXIS, Y_AXIS et Z_AXIS. Si la fonction renvoie la valeur zéro, c'est que l'accéléromètre est perpendiculaire à la verticale de la gravité. Les valeurs positives et négatives permettent de connaître le sens et le taux d'accélération.

```
int esp.readAccelerometer(const byte axe)
```

readButton()

Renvoie l'état actuel d'un bouton Esplora. Renvoie la valeur false si le bouton est enfoncé et la valeur true sinon.

```
bool esp.readButton(byte channel)
```

writeRGB()

Écrit trois valeurs pour définir la luminosité des composantes rouge, verte et bleue de la diode LED tricolore de l'Esplora.

```
void esp.writeRGB(byte red, byte green, byte blue)
```

writeRed()

Définit la luminosité de la composante rouge de la diode LED entre 0 et 255.

```
void esp.writeRed(byte red)
```

writeGreen()

Définit la luminosité de la composante verte de la diode LED entre 0 et 255.

```
void esp.writeGreen(byte green)
```

writeBlue()

Définit la luminosité de la composante bleue de la diode LED entre 0 et 255.

```
void esp.writeBlue(byte blue)
```

readRed()

Renvoie la dernière valeur de luminosité de la diode LED rouge.

```
byte esp.readRed()
```

readGreen()

Renvoie la dernière valeur de luminosité de la diode LED verte.

```
byte esp.readGreen()
```

readBlue()

Renvoie la dernière valeur de luminosité de la diode LED bleue.

```
byte esp.readBlue()
```

tone()

Émet un son sur le buzzer de la carte Esplora à la fréquence spécifiée. Le son est permanent si vous ne fournissez pas de durée limite en argument et vous devez appeler `noTone()` pour le forcer au silence. Ne peut émettre qu'une fréquence à la fois. Notez que l'utilisation de cette fonction perturbe le contrôle de luminosité de la diode LED rouge.

```
void esp.tone(unsigned int freq)

void esp.tone(unsigned int freq, unsigned long duree)
```

noTone()

Provoque la fin d'émission du signal carré sur le buzzer.

```
void esp.noTone()
```

Librairie USB

La librairie USB de base permet à une carte Leonardo ou Micro de jouer le rôle d'une souris et ou d'un clavier pour la machine hôte.

Si vous laissez en activité en permanence la librairie Mouse ou Keyboard, vous aurez du mal à faire exécuter votre programme par la carte Arduino. Vous ne devez appeler les fonctions telles que `Mouse.move()` et `Keyboard.print()` que lorsque la machine hôte est prête à les exécuter. Une solution consiste à utiliser un interrupteur physique pour décider quand la carte Arduino peut émettre des messages souris ou clavier.

Mouse

Les fonctions de cette famille permettent à une carte Leonardo ou Micro de contrôler la position d'un pointeur sur la machine hôte. La position renvoyée est toujours relative à la position précédente. Elle n'est pas absolue.

```
Mouse.begin()
Mouse.click()
Mouse.end()
Mouse.move()
Mouse.press()
Mouse.release()
Mouse.isPressed()
```

Keyboard

Les fonctions clavier permettent à une carte Leonardo ou Micro d'envoyer des frappes de touches à la machine hôte. Toutes les touches ne sont pas disponibles, et notamment pas les caractères non affichables. En revanche, la librairie permet d'utiliser les touches de modification.

```
Keyboard.begin()
Keyboard.end()
Keyboard.press()
Keyboard.print()
Keyboard.println()
```

```
Keyboard.release()
Keyboard.releaseAll()
Keyboard.write()
```

Je rappelle qu'une touche de modification modifie la signification d'une autre touche lorsqu'elles sont enfoncées simultanément. Voici les touches de modification reconnues par la carte Leonardo (Tableau 7.1).

Tableau 7.1 : Touches de modification d'un clavier USB.

Touche	Hexa	Décimal	Touche	Hexa	Décim
KEY_LEFT_CTRL	0x80	128	KEY_PAGE_UP	0xD3	211
KEY_LEFT_SHIFT	0x81	129	KEY_PAGE_DOWN	0xD6	214
KEY_LEFT_ALT	0x82	130	KEY_HOME	0xD2	210
KEY_LEFT_GUI	0x83	131	KEY_END	0xD5	213
KEY_RIGHT_CTRL	0x84	132	KEY_CAPS_LOCK	0xC1	193
KEY_RIGHT_SHIFT	0x85	133	KEY_F1	0xC2	194
KEY_RIGHT_ALT	0x86	134	KEY_F2	0xC3	195
KEY_RIGHT_GUI	0x87	135	KEY_F3	0xC4	196
KEY_UP_ARROW	0xDA	218	KEY_F4	0xC5	197
KEY_DOWN_ARROW	0xD9	217	KEY_F5	0xC6	198
KEY_LEFT_ARROW	0xD8	216	KEY_F6	0xC7	199
KEY_RIGHT_ARROW	0xD7	215	KEY_F7	0xC8	200
KEY_BACKSPACE	0xB2	178	KEY_F8	0xC9	201
KEY_TAB	0xB3	179	KEY_F9	0xCA	202
KEY_RETURN	0xB0	176	KEY_F10	0xCB	203

KEY_ESC	0xB1	177	KEY_F11	0xCC	204
KEY_INSERT	0xD1	209	KEY_F12	0xCD	205

Librairies complémentaires

De nombreuses librairies complémentaires sont disponibles pour les cartes Arduino. Certaines ont été créées par des personnes et d'autres par des entreprises qui vendent du matériel et des accessoires Arduino. Plusieurs fournisseurs proposent de télécharger les librairies pour leurs produits. Une simple recherche Web permet d'accéder au code des librairies et aux exemples.

Les Tableaux 7.2 à 7.8 proposent une sélection de librairies classées par catégories. Les descriptions sont résumées, car je n'ai malheureusement pas la place dans ce livre pour les décrire toutes à leur juste valeur. Pour tous détails, voyez la page `http://www.arduino.cc/en/Reference/libraries`.

Tableau 7.2 : Communication (réseau et protocoles).

Librairie	Description
Messenger	Gestion de messages texte d'un ordinateur.
NewSoftSerial	Version améliorée de la librairie SoftwareSerial.
OneWire	Contrôle de périphériques avec le protocole One Wire (Dallas Semiconductor).
PS2Keyboard	Lecture de caractères depuis un clavier PS2.
Simple Message System	Envoi de messages entre Arduino et la machine hôte.
SSerial2Mobile	Envoi de messages texte ou de courriels *via* un téléphone cellulaire au moyen de commandes AT utilisant SoftwareSerial.
Webduino	

	Librairie de serveur Web à utiliser avec le bouclier Arduino Ethernet.
X10	Envoi de signaux X10 par le réseau électrique en courant alternatif.
XBee	Communications XBee en mode API.
SerialControl	Contrôle à distance d'une autre carte Arduino *via* une liaison série.

Tableau 7.3 : Capteurs.

Librairie	Description
Capacitive Sensing	Transforme deux broches en capteurs capacitifs.
Debounce	Lecture d'entrée bruitée (par exemple les rebonds d'un bouton).

Tableau 7.4 : Afficheurs et diodes LED.

Libairie	Description
GFX	Classe de base avec les routines graphiques standard (d'Adafruit Industries).
GLCD	Routines graphiques pour LCD basées sur le KS0108 ou compatibles.
LedControl	Pour contrôleur de matrice de LED ou 7 segments doté du circuit MAX7221 ou MAX7219.
LedDisplay	Contrôle d'un afficheur LED défilant HCMS-29xx.

Matrix	Routines de base de gestion d’affichage sur matrice de LED.
PCD8544	Pour contrôleur d’écran Nokia 5510 (d’Adafruit Industries).
Sprite	Gestion d’objets écran (*sprites*) pour des animations sur une matrice de LED.
ST7735	Pour contrôleur d’écran TFT 1,8 pouce, 128 sur 160 pixels (d’Adafruit Industries).

Tableau 7.5 : Audio et ondes.

Librairie	Description
FFT	Analyse de fréquence audio ou autre signal analogique.
Tone	Génère des ondes carrées en tâche de fond.

Tableau 7.6 : Moteurs et PWM.

Librairie	Description
TLC5940	Contrôle le circuit TLC5940 IC (unité PWM 12 bits à 16 canaux).

Tableau 7.7 : Temps.

Librairie	Description
DateTime	Gestion de la date et de l’heure par logiciel.
Metro	Programmation d’actions à déclencher de façon périodique.

MsTimer2	Exploite l’interruption du Timer 2 pour lancer une action toutes les N millisecondes.

Tableau 7.8 : Outils.

Librairie	Description
PString	Classe légère pour écrire dans un tampon mémoire.
Streaming	Simplification des instructions print.

CHAPITRE 8
Cartes boucliers (*shields*)

Dans le monde Arduino, un bouclier (*shield* en anglais) est un circuit imprimé complémentaire dont le format est tel que ses deux rangées de connecteurs peuvent s'enficher directement sur les deux rangées de connecteurs femelles d'une carte Arduino au format Uno, Duemilanove, Leonardo ou Mega. Par ce simple enfichage, le bouclier dispose immédiatement de toute une série de connexions : alimentation, entrées/sorties numériques, entrées analogiques, *etc.* Nous allons passer en revue un certain nombre de boucliers Arduino disponibles dans le commerce. Dans le Chapitre 10, nous verrons comment créer un bouclier.

Il existe des boucliers pour de très nombreux besoins, de la simple carte de connexion à des sous-systèmes complets pour contrôler un moteur, communiquer *via* Internet, stocker des données sur une carte mémoire SD ou afficher des données. La plupart des boucliers acceptent d'être à leur tour surmontés d'un autre bouclier. Il n'est pas rare de trouver des solutions constituées d'une carte Arduino sur laquelle sont empilés deux ou trois boucliers.

Les fabricants et revendeurs qui sont cités dans ce chapitre n'apparaissent pas suite à un quelconque accord commercial de ma part. Je présente les boucliers qui me semblent les plus représentatifs des différentes fonctions. Vous trouverez aisément d'autres fabricants et revendeurs pour des boucliers équivalents. Il suffit de chercher.

Ce chapitre ne prétend nullement à l'exhaustivité. Le monde des boucliers Arduino est très dynamique, de nouveaux fabricants proposant par exemple des versions améliorées de boucliers existants et des boucliers qui répondent à de nouveaux besoins. La sélection que présente le chapitre donne un bon aperçu de ce qui est disponible, et vous trouverez des liens en annexe pour vous approvisionner. Dans certains cas, je fournis des détails supplémentaires par rapport à ce que le fabricant ou le revendeur propose, ou a oublié de donner. Cela ne signifie pas que j'apprécie plus particulièrement le bouclier concerné. Il se trouve que j'aime beaucoup disposer de toute la documentation dont je peux

avoir besoin. Il vous arrivera peut-être de détecter dans le commerce un bouclier qui semble vraiment intéressant, mais pour lequel il n’y a que pas ou peu de documentation, ou bien que celle-ci n’est disponible que dans une langue que vous ne maîtrisez pas comme le chinois ou autre. J’espère que ce chapitre vous permettra de pallier ce genre de lacunes, ou au moins de vous donner des indices pour savoir où continuer vos recherches.

Certains des boucliers présentés ne sont plus disponibles chez le revendeur mentionné, mais ils le sont à d’autres sources. La plupart des revendeurs prévoient des liens vers la documentation. Si vous repérez un bouclier qui ressemble à celui dont vous avez besoin, il suffit de vérifier dans la documentation technique qui est normalement disponible. Rappelons que le matériel est de type open source.

Dans l’esprit de certaines personnes, le concept de bouclier n’est pas bien clair. Un bouclier n’est pas un module avec deux rangées de broches mâles sur le dessous (je présente les modules dans le Chapitre 9). Pour qu’une carte soit un bouclier, il faut qu’elle soit conforme à des dimensions physiques (j’en parle un peu plus loin) et qu’elle satisfasse à des caractéristiques électriques. Lorsque ces conditions ne sont pas satisfaites, il s’agit d’un module, et certains modules peuvent s’enficher directement dans une carte Arduino, mais ils réclament généralement des connexions supplémentaires.

Caractéristiques électriques d’un bouclier

Si vous comparez les brochages de plusieurs boucliers de ce chapitre, vous pourrez constater des similitudes. Par exemple, tous les boucliers qui utilisent une interface à deux fils, c’est-à-dire l’interface I2C ou TWI, exploitent toujours les deux broches A4 et A5 de l’Arduino. Sur une carte de base (Diecimile, Duemilanove et Uno R2), ces deux broches font partie des broches analogiques. Sur les cartes plus récentes, les Uno R3, Ethernet et Leonardo, ces deux signaux I2C sont également disponibles dans la partie supplémentaire de l’autre rangée de broches, à côté des connecteurs USB et RJ-45. Si le bouclier utilise I2C, vous pouvez en conclure que les deux broches A4 et A5 ne peuvent plus être utilisées pour autre chose, à moins d’une acrobatie de programmation.

Dans le même esprit, les boucliers qui utilisent l’interface SPI utilisent les quatre broches D10, D11, D12 et D13 pour les signaux SS, MOSI, MISO et SCLK, respectivement. L’utilisation de D10 pour sélectionner l’esclave est facultative et certains boucliers utilisent une autre broche numérique pour la sélection. Rappelons cependant que dans le mode SPI, chaque circuit esclave doit être sélectionné avec un signal Select pour qu’il accepte de se mettre à l’écoute des données envoyées par le maître. Les boucliers qui gèrent plusieurs équipements SPI doivent donc utiliser plusieurs broches d’entrées-sorties numériques pour la sélection.

Les boucliers qui utilisent les liaisons asynchrones UART (chez Arduino, on parle de USART) exploitent les deux broches D0 et D1 pour les signaux Rx et Tx. Sur les cartes Mega, les légendes sont RxD0 et TxD0. Certains boucliers Bluetooth utilisent cette interface UART pour communiquer avec le module Bluetooth. C’est bien sûr aussi le cas des boucliers série RS-232 et RS-485. Il existe aussi des boucliers qui exploitent quasiment toutes les broches de la carte Arduino, par exemple le bouclier multifonction en kit présenté tout à la fin du chapitre. Cela dit, trois broches numériques et une broche analogique accessibles sur cette carte ne sont pas utilisées par elle. De façon générale, vous devez supposer que les boucliers d’extension utilisent quasiment toutes les broches disponibles. Par exemple, les boucliers d’affichage ne laissent quasiment aucune connexion disponible pour des signaux qu’ils n’utiliseraient pas. D’ailleurs, les boucliers afficheurs sont généralement les derniers dans un empilement. La

majorité des boucliers ne présente pas une grande complexité du circuit. Ils servent d'abord à exploiter des circuits intégrés et des composants spécialisés. Ce sont donc pour l'essentiel des plaques sur lesquelles sont réunis des circuits intégrés, des relais, et d'autres composants, un peu comme les cartes Arduino. Les paramètres électriques des boucliers dépendent des circuits et des composants que les plaques hébergent. C'est cette simplicité qui permet de proposer les boucliers à des prix raisonnables, et ce sont directement les possibilités des circuits intégrés qui sont montés dessus qui déterminent l'intérêt des boucliers.

Certains boucliers mémorisent localement les signaux qui viennent et qui vont vers la carte Arduino, grâce à des circuits actifs ou des isolateurs optiques. D'autres boucliers ne sont que des extensions des connexions de la carte Arduino. Ces boucliers d'extension sont la première catégorie que nous allons découvrir dans la suite du chapitre. Ce genre de carte passive ne comporte aucune protection ; rien ne vous empêche de brancher du 12 V sur l'entrée numérique 5 V ou même 3,3 V et immédiatement transformer le microcontrôleur en un bout de plastique noirci. Faites toujours très attention aux limitations en tension et en courant que vous impose le microcontrôleur AVR.

Caractéristiques physiques des boucliers

Au niveau de son encombrement, un bouclier possède la même largeur qu'une carte Arduino standard (dont les dimensions ont été fournies dans le Chapitre 4). La longueur peut être la même, mais certains boucliers sont plus longs et d'autres plus courts que la carte Arduino qui les accueille. Il n'est pas impossible de créer un bouclier plus large que la carte Arduino, à partir du moment où les deux rangées de connecteurs gardent le bon écartement pour pouvoir enficher la carte. La Figure 8.1 montre le principe d'enfichage d'un bouclier dans une carte Arduino.

Les cartes Arduino au format étendu R3 possèdent deux broches au bout de chaque rangée qui ne sont pas utilisées par la plupart des boucliers. Cela n'a pas d'importance, parce que ces broches sont soit des copies d'autres broches, soit ne sont connectées à rien. Pour une carte au format Mega, le bouclier s'enfiche comme montré dans la Figure 4.20 du Chapitre 4. La plupart des broches de la carte Mega ne sont pas connectées au bouclier, et certaines deviennent inaccessibles par la présence du bouclier. Cependant, toutes les broches et signaux de base sont disponibles au niveau du bouclier.

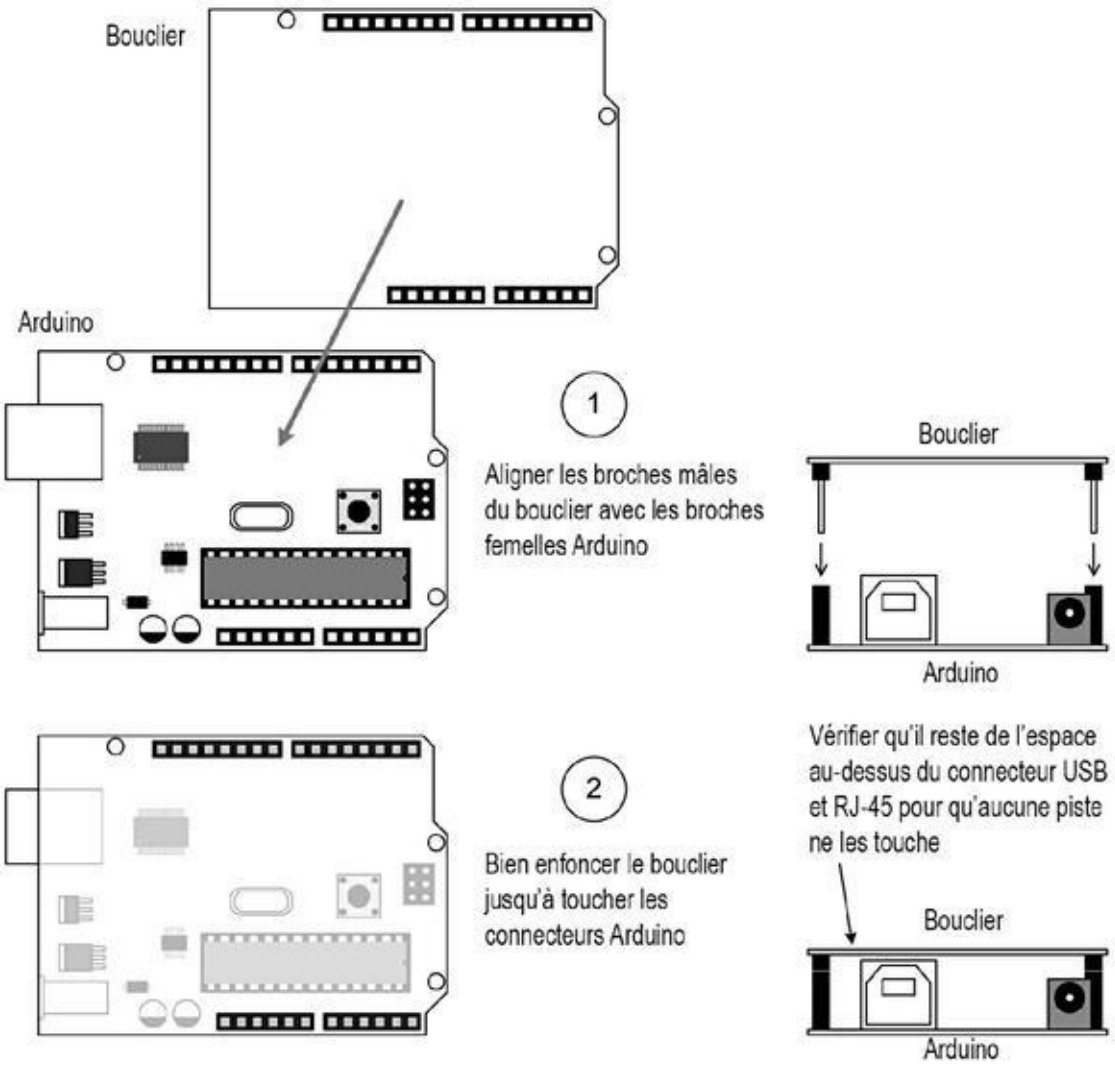


Figure 8.1 : Enfichage d'un bouclier sur une carte Arduino.

Vérifiez toujours qu'il reste un peu d'espace entre les composants de la carte Arduino et les pistes de la carte bouclier. Parfois, le haut du connecteur USB ou RJ-45 vient toucher les pistes, ce qui peut provoquer un court-circuit.

Lorsque vous craignez qu'un contact indésirable se fasse entre les pistes du dessous de la plaque du bouclier et les composants de la carte Arduino ou d'un autre bouclier, vous pouvez utiliser du ruban isolant d'électricien ou du carton pour éviter tout contact. Une solution plus professionnelle consiste à ajouter des entretoises en métal ou en nylon d'environ 10 mm de longueur. Certaines entretoises sont filetées et d'autres sont lisses. Dans tous les cas, le but est d'augmenter la distance entre les deux objets à réunir, tout en assurant une liaison mécanique solide. Voyez par exemple la Figure 9.3 du Chapitre 9.

Une autre solution pour écarter un peu le bouclier consiste à utiliser des vis d'écartement, constituées d'une partie mâle et d'une partie femelle. Ces vis sont très utilisées dans les PC de bureau. Si vous avez déjà démonté une tour, vous savez à quoi cela ressemble. La Figure 8.2 montre les différents types d'entretoises lisses et filetées qui peuvent servir à assurer des séparations entre boucliers. Les matières utilisées sont le nylon, l'aluminium, l'acier inoxydable, le cuivre ou le plastique.

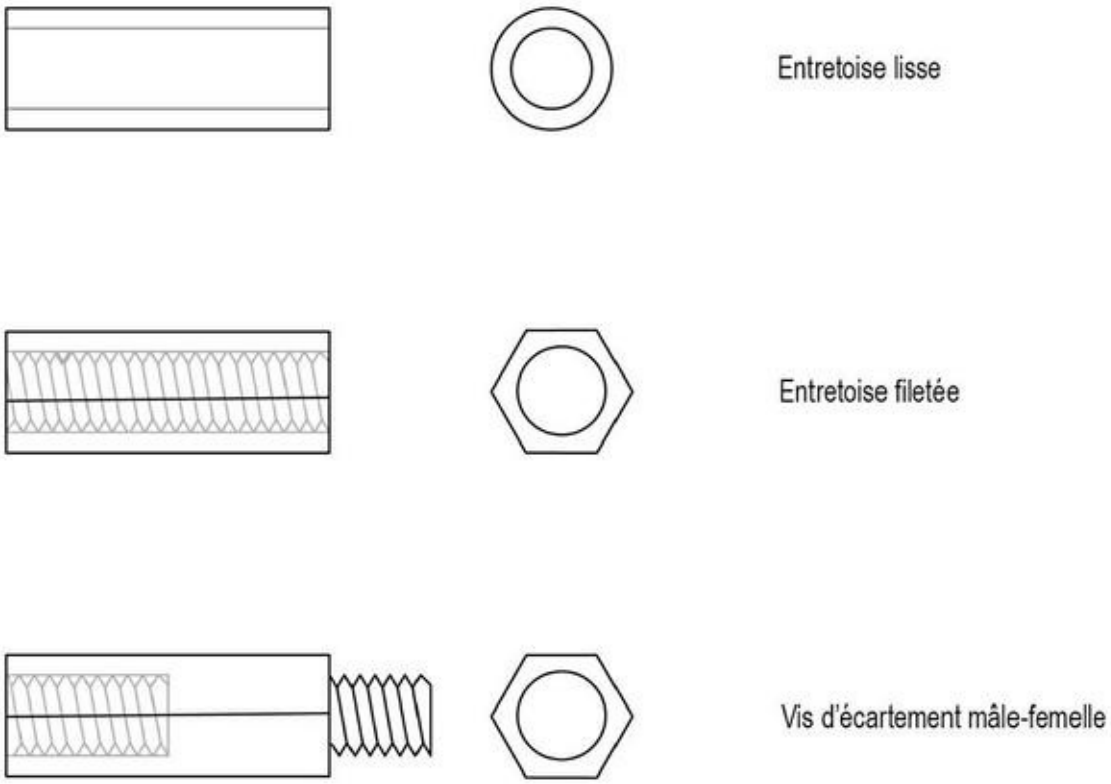


Figure 8.2 : Entretoises et vis d'écartement.

Vous pouvez aussi empêcher les broches du bouclier de s'enfoncer trop dans les broches femelles de la carte précédente en plaçant des morceaux de baguette de bois pendant que vous enfoncez les connecteurs pour conserver un peu d'écartement.

Cette visserie un peu spéciale peut se trouver dans n'importe quel magasin d'électronique.

Techniques d’enfichage

Le choix du terme « bouclier » rappelle que la carte une fois enfichée sur la carte Arduino constitue une sorte de protection, comme celle qu’offre un bouclier. Simultanément, elle masque l’accès à la carte qui se trouve dessous. Dans certains cas, il sera impossible d’ajouter une carte lorsque l’on a besoin d’accéder à certaines des sorties de la carte qui va être dessous. Il existe deux techniques pour résoudre ce dilemme.

La première technique consiste à allonger les broches de connexion, les deux cartes restant alignées dans le sens vertical. L’autre technique consiste à effectuer un décalage des connecteurs de la carte qui est enfichée. Ils ne sont plus dans le prolongement de ceux qui viennent s’enficher dans la carte du dessous. Lorsque plusieurs cartes sont empilées de cette façon, le résultat forme une sorte d’escalier. Bien sûr, les trous de fixation aux quatre angles ne coïncident plus.

Dans la technique par décalage, les connecteurs vers la carte suivante sont en général décalés le moins possible par rapport à la rangée qui vient s’enficher dans la carte du dessous.

Principales catégories de boucliers

Découvrons les principales catégories de boucliers qui sont commercialisés au moment où j'écris ces lignes. Du fait que nous sommes dans un univers de prototypage rapide et de production en petite série et à faible coût, vous verrez apparaître de nouveaux boucliers, puis les verrez disparaître un peu plus tard. La plupart des cartes que j'ai sélectionnées dans la suite sont suffisamment bien diffusées pour garantir leur pérennité. Je donne tout à la fin du chapitre une liste de sites Web des principaux fournisseurs et fabricants. Vous irez également consulter la liste complète dans l'Annexe C.

Voici les principales catégories de boucliers Arduino qui vont être présentées dans les sections suivantes :

- **Entrées-sorties**
  - — Boucliers d'extension des entrées-sorties
  - — Boucliers d'expansion des entrées-sorties
  - — Boucliers de relais
  - — Boucliers de routage des signaux
- **Stockage mémoire**
  - — Gestion de cartes mémoire SD et microSD
- **Communications**
  - — Entrées-sorties série
  - — Midi
  - — Ethernet
  - — Bluetooth
  - — USB
  - — ZigBee
  - — CAN

- **Prototypage**
  - — Contrôle de mouvements physiques
  - — Pilotage de moteurs à courant continu et pas à pas
  - — Pilotage de servomoteurs et par impulsion PWM
- **Affichage**
  - — Matrices de diodes LED
  - — Afficheurs LCD sept segments
  - — Afficheurs à cristaux liquides LCD
  - — Afficheurs couleur TFT
- **Instrumentation**
  - — Journalisation de données, data logging
  - — Analyseurs logiques
  - — Convertisseurs analogique-numérique 24 bits
  - — Convertisseurs numérique-analogique 12 bits
- **Boucliers d’adaptation**
  - — Adaptateurs pour Nano
  - — Adaptateurs à borniers
- **Boucliers divers**
  - — Cartes de prototypage à borniers
  - — Boucliers multifonctions

Je terminerai par une courte présentation de quelques boucliers à usage très spécifique : une carte de pilotage de machine-outil à commande numérique, une carte d’interface de contrôle RepRap pour imprimante 3D est un contrôleur de jeu basé sur un circuit FPGA.

Cartes d’entrées-sorties

Il existe des boucliers d’entrées-sorties passifs dont le seul rôle est de rendre les broches d’entrées-sorties du microcontrôleur plus faciles d’accès qu’en se plaçant directement à la sortie de la carte Arduino. Dans le cas de l’Arduino Nano, les broches sous la carte d’extension sont reliées à des borniers dans le support du circuit de la Nano.

Il existe deux sortes de cartes d’entrées-sorties : les cartes d’extension et les cartes d’expansion. Certains revendeurs confondent les deux termes. Une carte d’extension ne fait rien d’autre que reporter les entrées et sorties Arduino sans intervenir sur les signaux transportés. Seul le mode de connexion change. En revanche, une carte d’expansion embarque des composants actifs qui soit augmentent le nombre d’entrées-sorties, soit appliquent un certain traitement aux signaux. Ces boucliers actifs utilisent le protocole SPI ou I2C pour communiquer avec la carte Arduino.

Boucliers d’extension des entrées-sorties

Cette catégorie de boucliers est une sorte de prolongateur qui offre plus de confort pour se connecter aux broches du microcontrôleur. Certains modèles comportent des tampons mémoire actifs, mais la plupart ne font que reporter les signaux. Certains parlent de cartes d’extension, mais c’est une erreur. Il n’y a que report des signaux d’un connecteur Arduino vers un ou des connecteurs sur le bouclier.

Bouclier SainSmart Sensor

Cette carte est enfichable avec décalage (vous pouvez voir le décalage des broches dans la Figure 8.3). Les entrées-sorties du microcontrôleur sont proposées sur des prises multibroches à verrouillage, et sur des broches femelles ; deux emplacements de 10 broches sont prévus pour y installer un connecteur pour câble en nappe de type IDC. Le bouton de Reset est disponible. Ce bouclier fonctionne avec toutes les cartes Arduino conformes à la configuration de base des broches de sortie, y compris les cartes Mega.

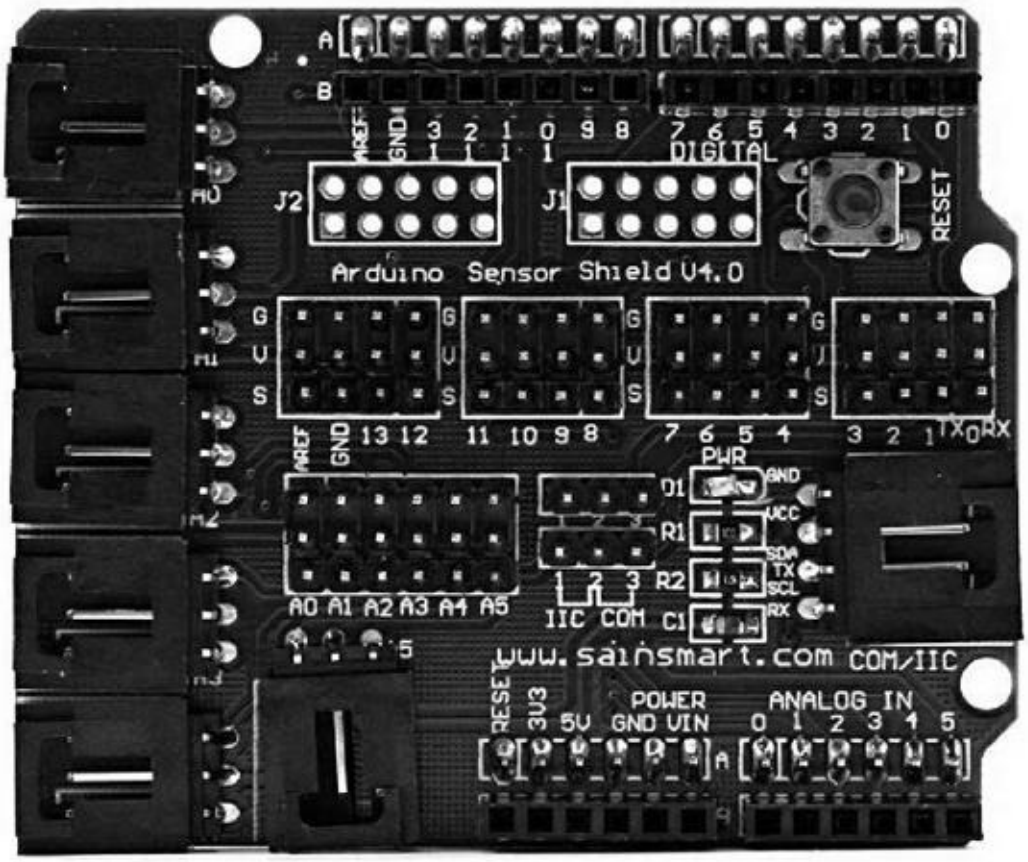


Figure 8.3 : Bouclier d'extension des entrées-sorties SainSmart.

Sur la Figure 8.4, vous pouvez voir les gros connecteurs carrés sur le bord gauche. Ils offrent trois ou quatre fils. Les câbles correspondants sont largement disponibles. En dehors de SainSmart, vous pouvez en trouver chez TrossenRobotics.

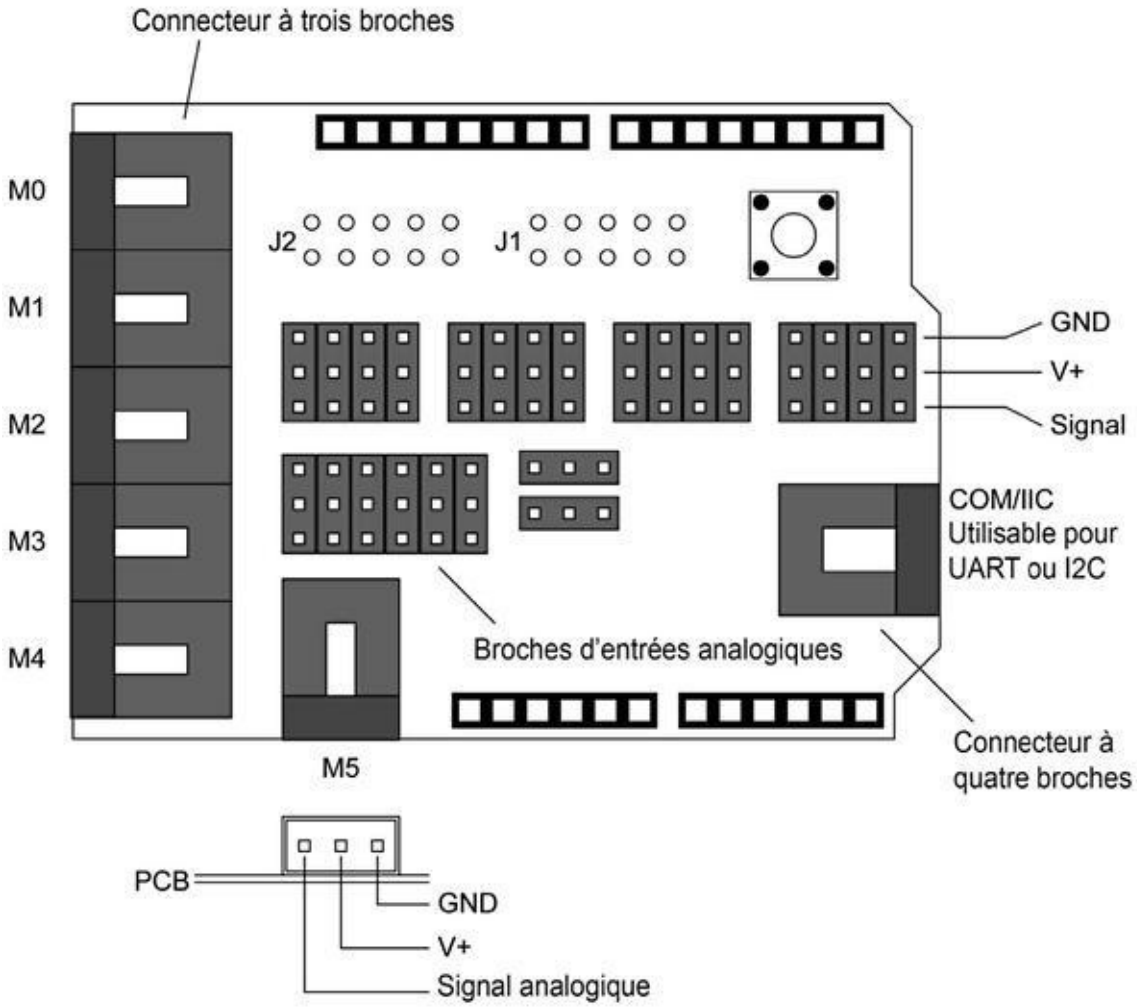


Figure 8.4 : Schéma des broches et connecteurs de la carte d'entrées-sorties SainSmart.

Bouclier TinkerKit Sensor Shield

Ce bouclier empilable (Figure 8.5) possède de longues broches mâles d'enfichage, 12 connecteurs à trois broches et deux connecteurs à quatre broches. Le bouton Reset se trouve entre ces deux derniers connecteurs.

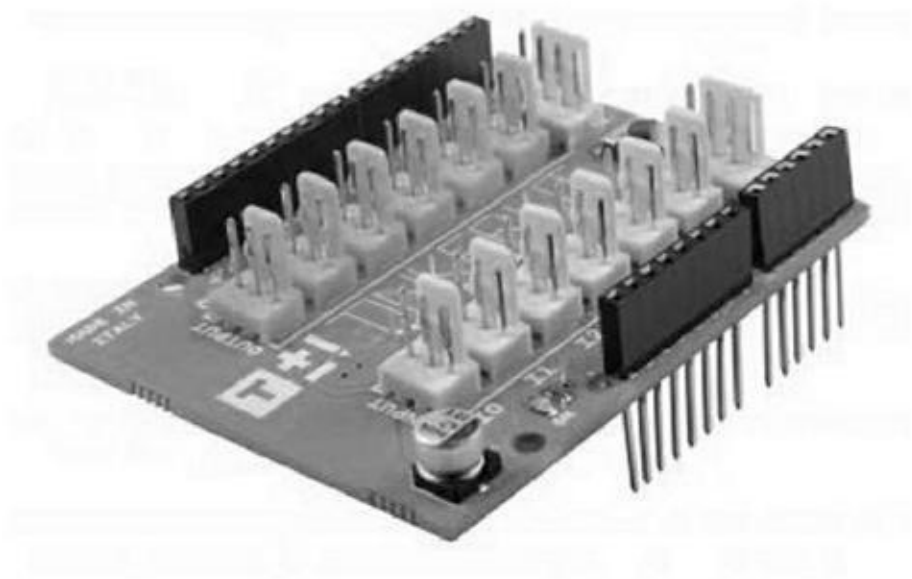


Figure 8.5 : Carte d'extension TinkerKit.

Au départ, cette carte était vendue pour connecter les capteurs et modules de moteurs conçus par TinkerKit, mais elle peut être utilisée avec n'importe quel autre capteur. La Figure 8.6 montre le schéma des connecteurs sur le circuit. La plupart ont trois fils, comme la carte SainSmart. Je donne plus de détails sur tous les modules TinkerKit pouvant se connecter à ce bouclier dans le Chapitre 9.

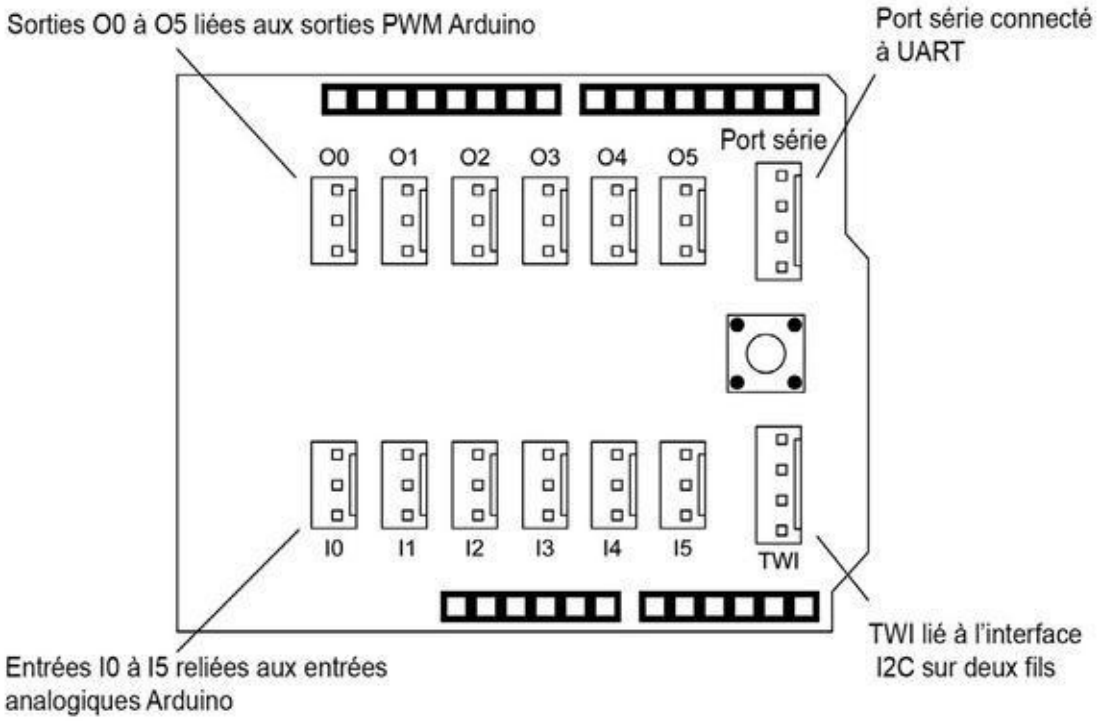


Figure 8.6 : Schéma des connecteurs de la carte d'extension TinkerKit.

Des doutes ont été émis au sujet de la pérennité de la société TinkerKit, mais tous les produits sont disponibles. Les librairies de fonctions ont été mises à disposition sur le site GitHub.

Bouclier TinkerKit Mega Sensor

Comme son nom l'indique, cette carte rend accessibles les broches d'entrées-sorties supplémentaires des modèles Arduino Mega, Mega2650 et Mega ADK (Figure 8.7). Comme sa petite sœur, elle offre des broches longues pour l'enfichage et un bouton Reset. C'est une version plus complète de la carte TinkerKit précédente.

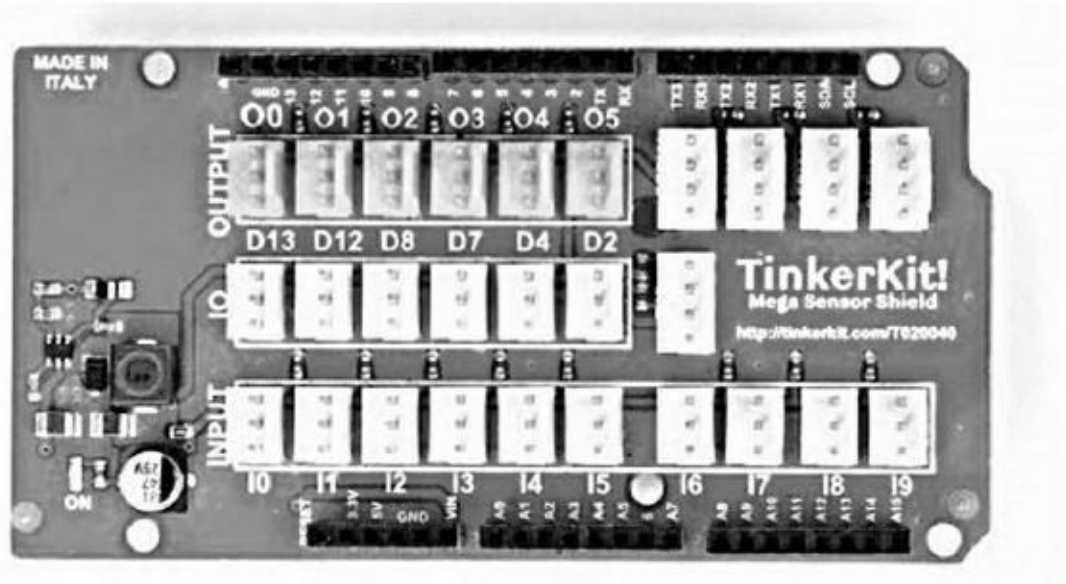


Figure 8.7 : Bouclier d'extension TinkerKit pour carte de type Mega.

Bouclier Grove Base Shield

Le système d'interconnexion entre boucliers et capteurs de Grove, vendu par Seeed Studio, devient de plus en plus populaire. Vous disposez d'une vaste gamme de modules et le bouclier de base est déjà équipé d'un microcontrôleur compatible Arduino ATmega328P. Le bouclier a été conçu par Linaro.com (96Boards.org). Il est visible dans la Figure 8.8.

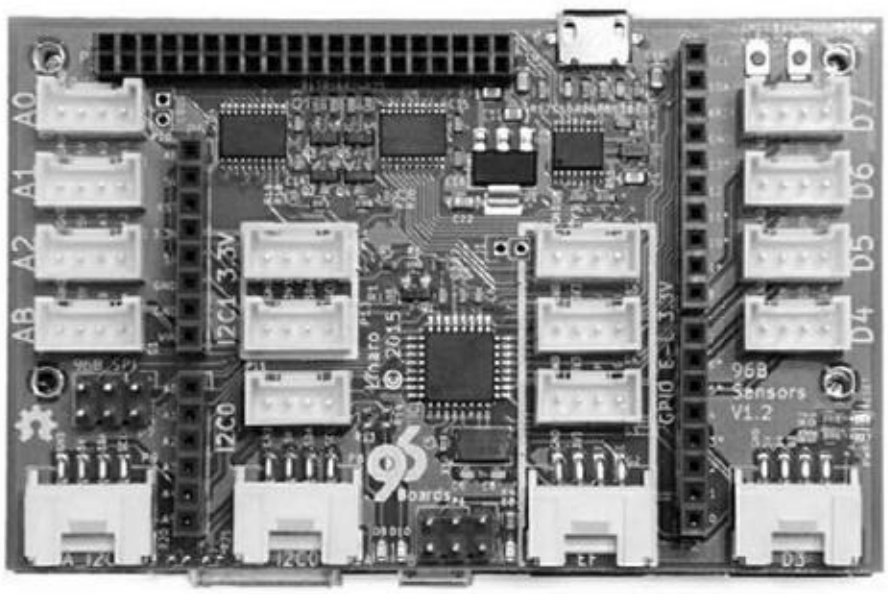


Figure 8.8 : Bouclier complet Grove Base Shield.

Le fournisseur Seeed Studio propose également une version passive de la même carte d'extension pour le système modulaire Grove, mais cette carte semble ne plus être fabriquée (Figure 8.9).

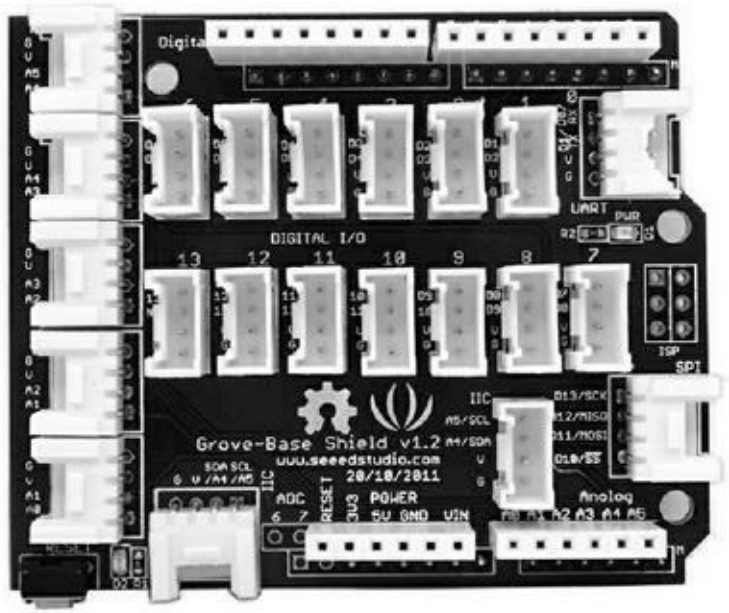


Figure 8.9 : Carte passive Seeed Studio Passive Grove.

Pour en savoir plus sur le système Grove, ses modules et ses boucliers, visitez la page wiki de Seeed Studio.

Bouclier CuteDigi Sensor Expansion

Malgré son nom, il s'agit bien d'une carte d'extension. Ce bouclier de CuteDigi (Figure 8.10) propose des connexions par broches mâles de tous les signaux rendus disponibles par une carte du type Mega. Elle n'est pas empilable, ce qui semble évident puisqu'il faut pouvoir se brancher sur les broches au milieu de la carte. Les légendes sont très lisibles.

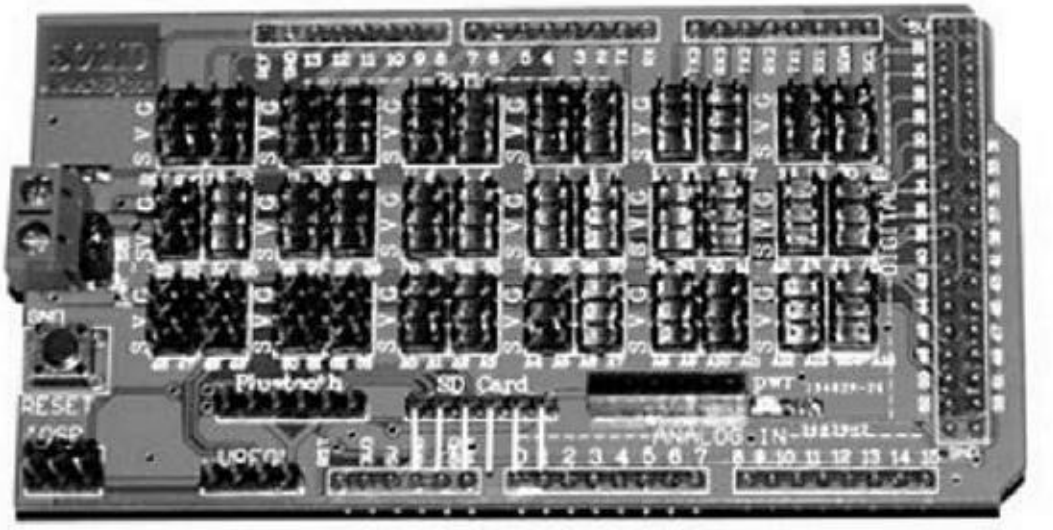


Figure 8.10 : Bouclier d'extension CuteDigi Mega avec connexions vers carte SD et Bluetooth.

Un des points forts de cette carte est qu'elle propose un connecteur coudé pour brancher un porte-carte mémoire de type SD et un autre connecteur pour ajouter un module Bluetooth. Le brochage des connecteurs est le même que dans d'autres boucliers d'extension, c'est-à-dire signal, +V et masse GND.

Boucliers d'expansion

Les boucliers d'expansion des entrées-sorties fournissent des capacités supplémentaires par rapport à la carte Arduino à laquelle ils sont reliés, généralement sous forme d'entrées-sorties numériques, bien que certains boucliers offrent également de nouvelles entrées analogiques. Ces boucliers

embarquent des composants actifs, ils sont donc plus coûteux que les boucliers d'extension. L'énorme avantage est qu'ils offrent de nombreux canaux d'entrées-sorties en n'occupant que peu de broches du microcontrôleur, grâce à l'utilisation du protocole I2C ou SPI. Les autres broches de la carte Arduino restent ainsi disponibles pour d'autres usages.

Bouclier Macetech Centipede

Ce bouclier (Figure 8.11) fonctionne avec l'interface I2C et propose 64 broches d'entrées-sorties numériques à usage général. Les sorties sont organisées en quatre groupes de 16, chaque groupe étant contrôlé par un circuit d'expansion I2C.

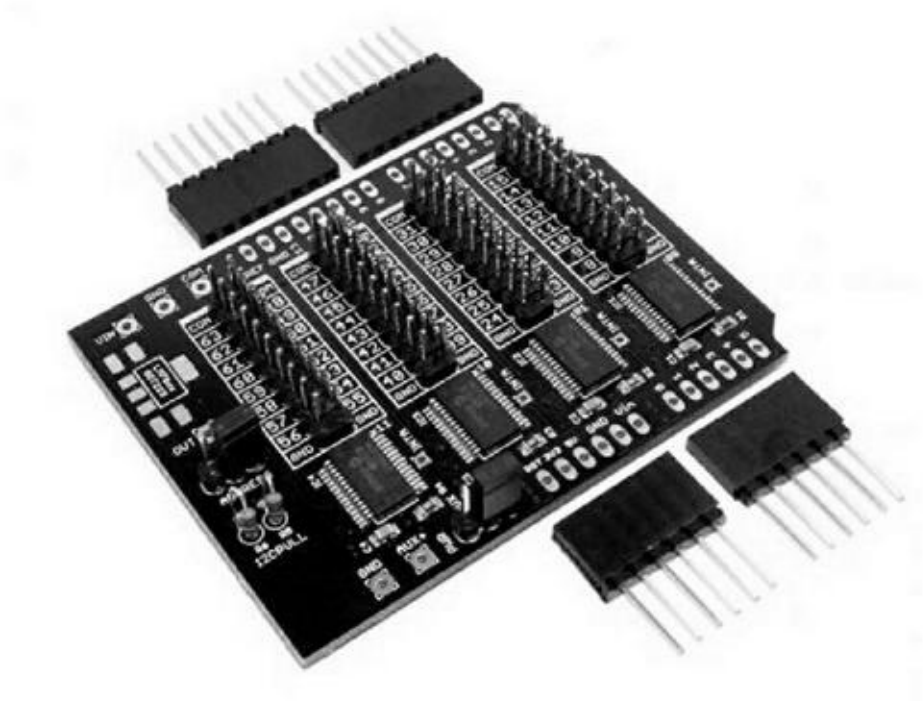


Figure 8.11 : Bouclier d'expansion Macetech Centipede.

La Figure 8.12 montre l'organisation des broches d'entrées-sorties du bouclier. Chacun des quatre multiplexeurs MUX gère 16 broches. Notez bien l'ordre de numérotation des broches dans chaque bloc : elle suit la même logique que la numérotation des broches d'un circuit intégré (en tournant autour).

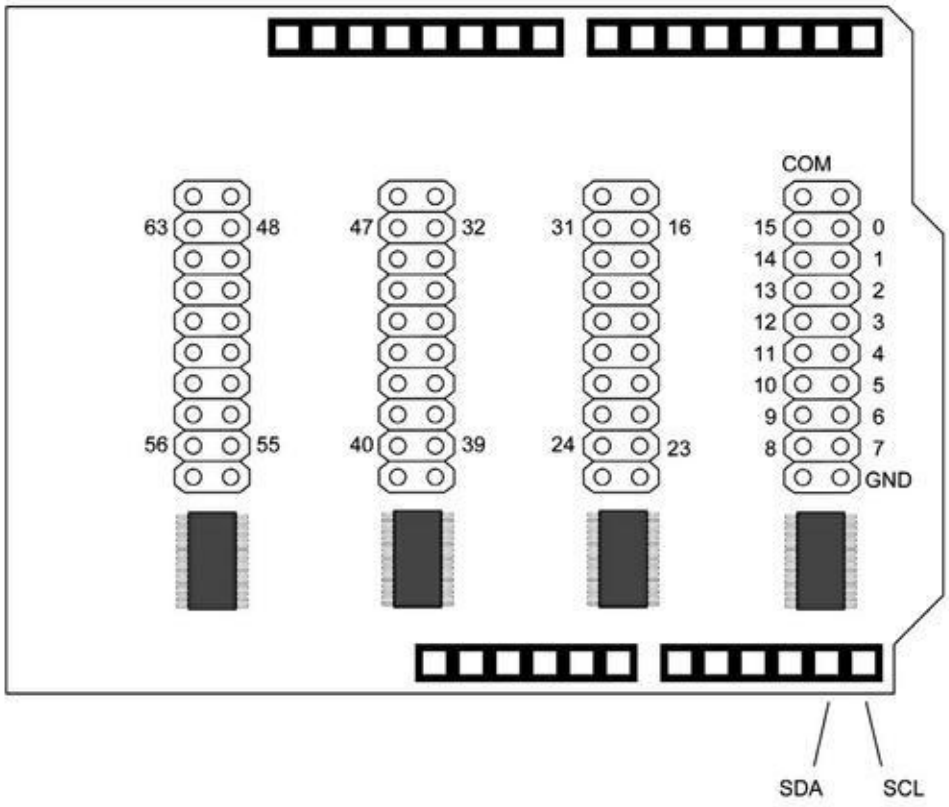


Figure 8.12 : Schéma des connexions de la carte Macetech Centipede.

Bouclier d’expansion LinkSprite I/OExpander

Cette carte empilable (Figure 8.13) se base sur un circuit d’expansion I2C du modèle MCP23017. Vous disposez ainsi de 16 entrées-sorties numériques supplémentaires. Ce bouclier a été prévu pour s’enficher sur une carte au format Arduino R3. Elle utilise les deux dernières broches 9 et 10 (SDA et SCL) que possèdent les cartes Uno R3 et Leonardo.

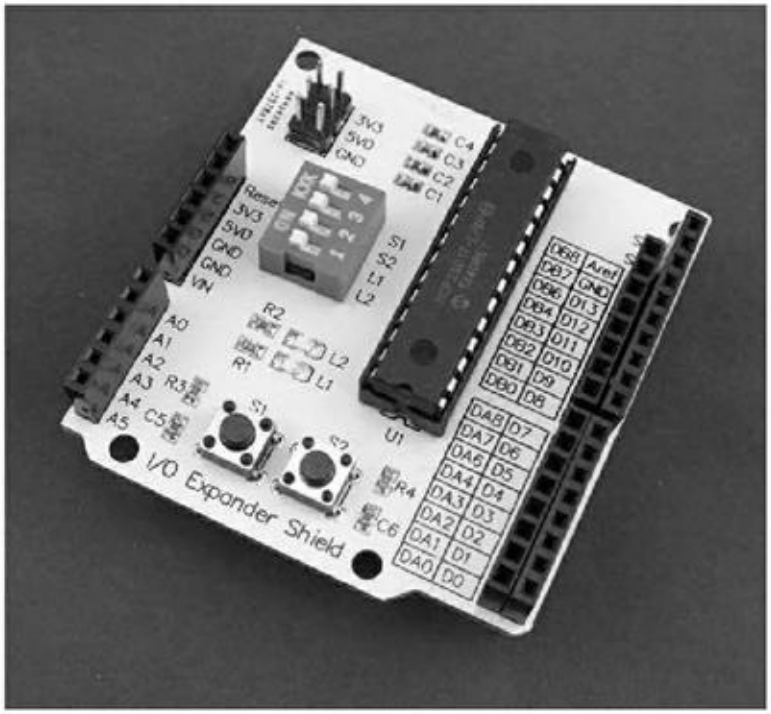


Figure 8.13 : Bouclier d'expansion LinkSprite I/O Expander.

Les sorties sont organisées en deux groupes de huit, portant les noms GPIOA et GPIOB (Figure 8.14). Les deux ports se présentent juste sous les broches des entrées-sorties numériques. Si vous ajoutez un bouclier par-dessus, vous n'avez plus accès à ces connecteurs. Ce bouclier doit donc normalement être le dernier d'un empilement.

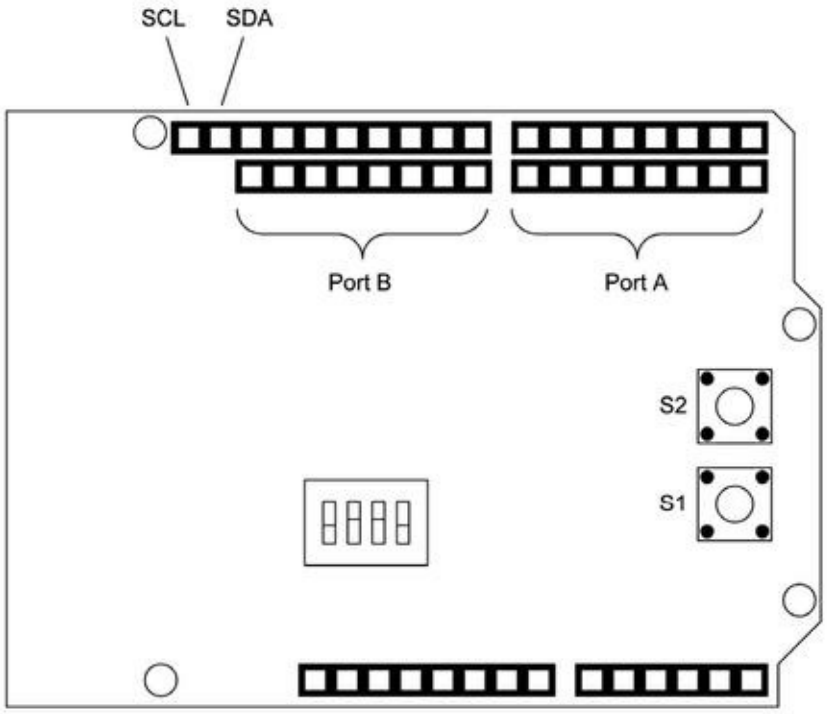


Figure 8.14 : Schéma des connexions du bouclier LinkSprite.

Bouclier Numato Digital and Analog IOExpander

Ce bouclier, de la société Numato (Figure 8.15), ajoute 28 canaux d'entrées-sorties numériques et 16 entrées analogiques. La carte utilise deux circuits I2C du type MCP23017 et un multiplexeur analogique NXP 74HC4067.

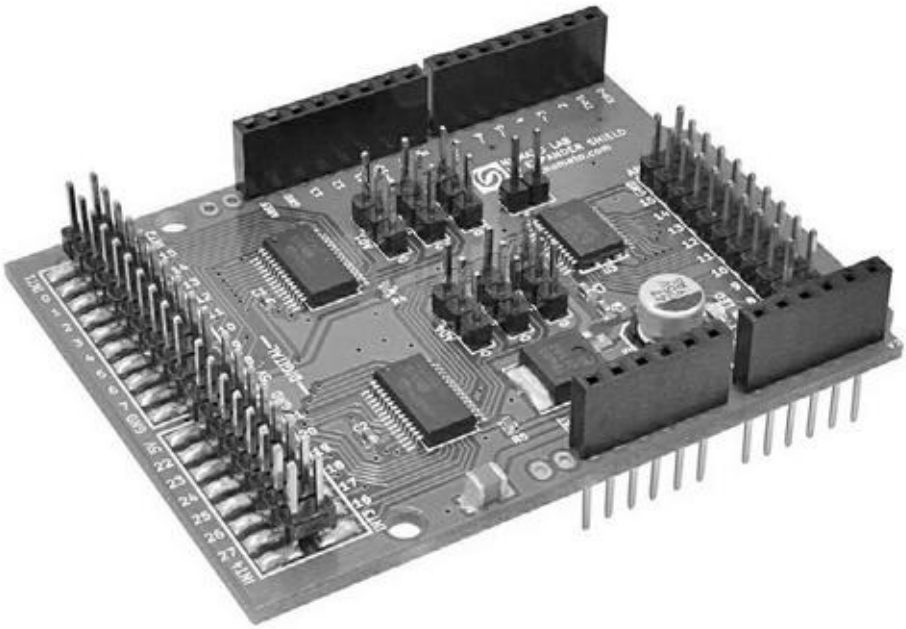


Figure 8.15 : Bouclier Numato Digital and Analog IO Expander.

Dans le coin inférieur droit de la Figure 8.16, vous constatez que la carte communique *via* le protocole I2C sur les broches A4 et A5. Les broches des interruptions des circuits MCP23017 sont rendues disponibles sur les connecteurs. La configuration des adresses I2C des deux circuits MCP est réalisée au moyen des six connecteurs au milieu de la carte.

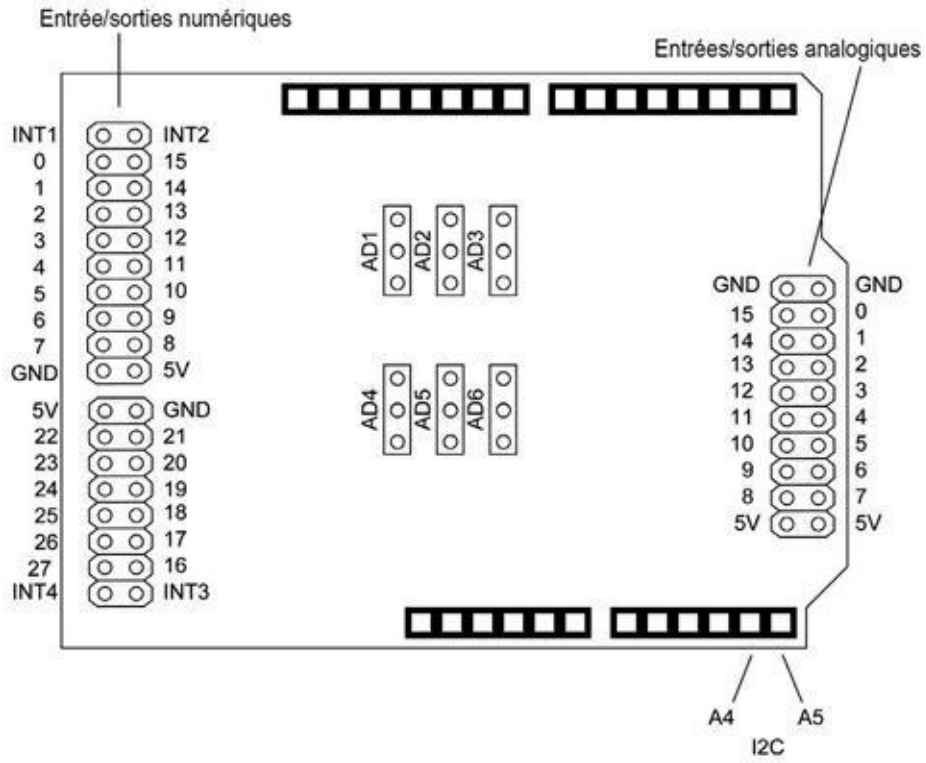


Figure 8.16 : Schéma des connecteurs du bouclier Numato.

Cartes boucliers de relais

Les boucliers relais comportent en général deux relais au minimum, de 5 ou 10 A. Certains modèles utilisent des relais Reed en boîtier plastique DIP.

Avant d'acquérir un bouclier relais, vérifiez bien le courant maximal supporté. Les relais modulaires qui sont utilisés dans les boucliers sont donnés pour supporter 10 A à 120 V, mais il est peu probable que les connecteurs et les pistes du circuit imprimé du bouclier supportent une telle intensité. La plupart des fournisseurs réduisent les capacités réelles pour tenir compte de ces contraintes. N'hésitez pas à vérifier les limites des relais en récupérant leur référence puis en vous renseignant chez le fabricant du relais. Méfiez-vous des boucliers sur lesquels les relais sont anonymes, c'est-à-dire qu'il n'est plus possible de savoir quel est le modèle exact. (Cette mise en garde s'applique à d'autres types de boucliers d'ailleurs.)

Bouclier DFRobot Relay

Cet imposant bouclier montre à quel point le standard de dimensions physiques des boucliers peut être revisité pour accueillir un plus grand nombre de composants. Dans le cas du bouclier DFRobot (Figure 8.17), les quatre relais

sont en fait installés sur une extension de la surface. Les contacts des relais sont donnés pour supporter 3 A à 24 V continus ou 120 V alternatifs. En théorie, ce bouclier est empilable, mais l'accès aux broches s'en trouve très menacé.

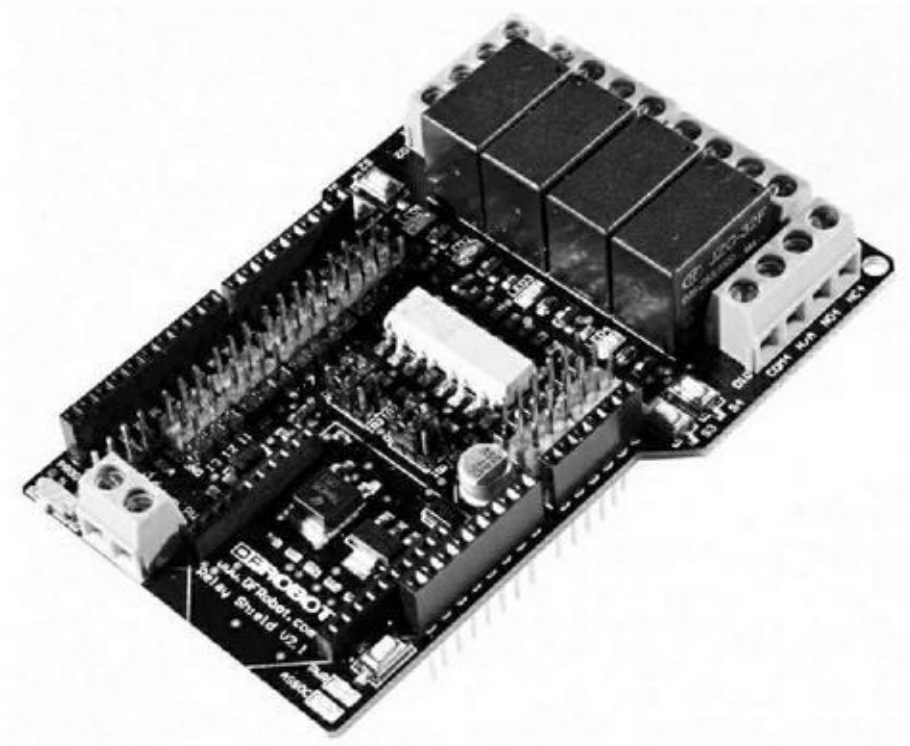


Figure 8.17 : Bouclier DFRobot Relay.

La carte permet des redirections des signaux numériques pour les pilotes des relais et pour un point de connexion d'un module XBee, grâce à des cavaliers (Figure 8.18). Tous les signaux numériques et analogiques de la carte Arduino sont disponibles sur des broches.

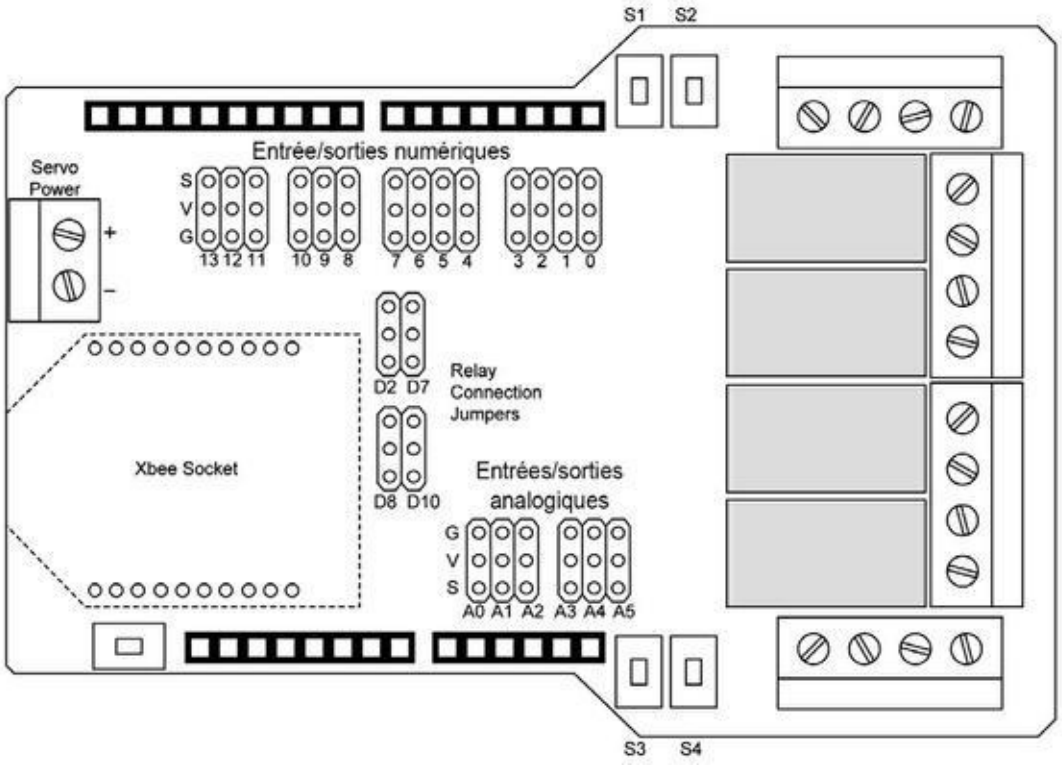


Figure 8.18 : Schéma des connexions du bouclier DFRobot.

Bouclier Numato Relay

Ce bouclier embarque deux relais modulaires de faible puissance : les contacts sont donnés pour 1 A à 120 V et 2 A à 24 V continus (Figure 8.19). Le bouclier exploite les broches numériques D2 et D3 de la carte Arduino. Les sorties des relais sont disponibles sur de petits borniers. La carte est empilable.

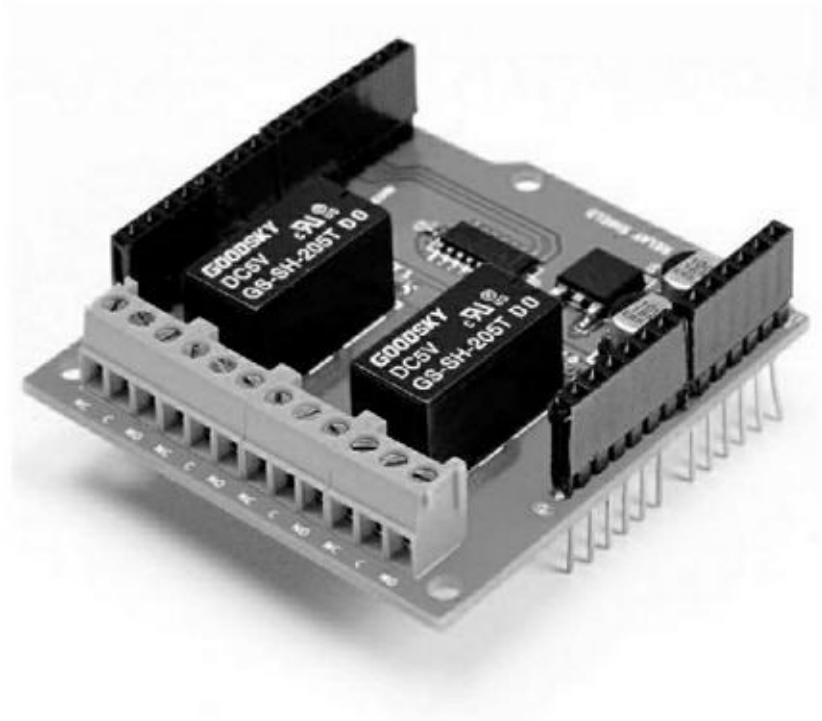


Figure 8.19 : Bouclier Numato Relay.

Bouclier Seeed Studio Relay

Ce bouclier (Figure 8.20) propose quatre relais dont les contacts sont donnés pour 10 A à 120 V. Il utilise les broches numériques Arduino D4 à D7, une par relais. Une diode LED est prévue pour témoigner de l'état de chaque relais. La carte est empilable.

Figure 8.20 : Bouclier Seeed Studio Relay.

Boucliers de routage du signal

Cette catégorie ne comporte pas beaucoup de modèles. Elle regroupe les boucliers passifs et les boucliers actifs basés sur un circuit de multiplexage.

Adafruit Patch Shield

Ce bouclier passif de routage est livré en kit par Adafruit (Figure 8.21). Il permet de rediriger des signaux entre la carte Arduino et quatre connecteurs au format RJ-45 (également appelés 8P8C). Les interconnexions sont réalisées par de petits fils appelés straps insérés dans les connecteurs femelles.

Je rappelle qu'il s'agit d'un kit qu'il faut donc monter par soudure. La photo montre le résultat du montage.

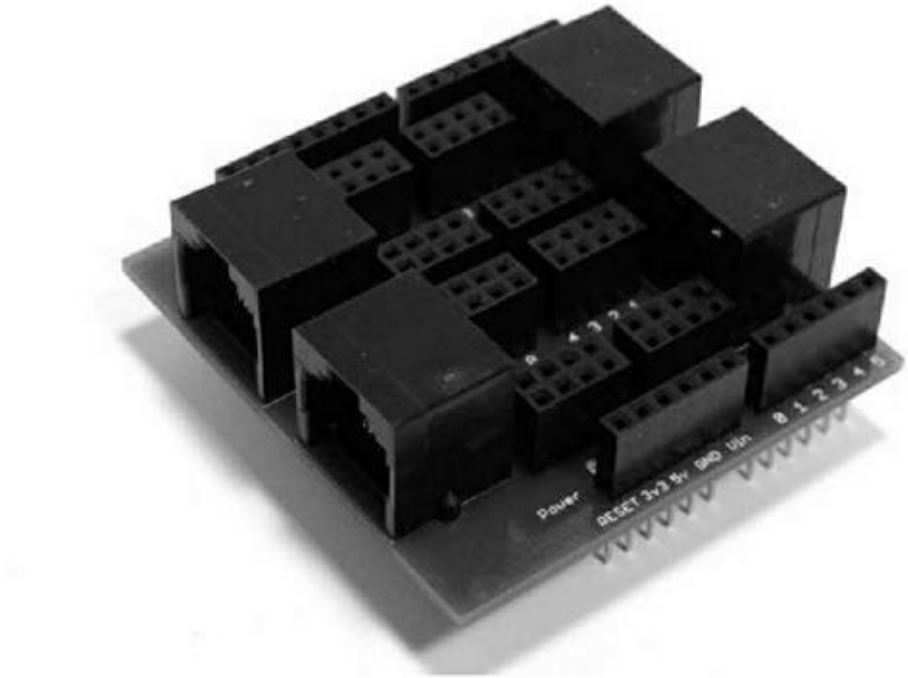


Figure 8.21 : Bouclier Adafruit Patch Shield.

La raison d'être d'un tel bouclier de routage est de rediriger des signaux Arduino vers des points éloignés en utilisant des câbles Ethernet (Figure 8.22). Le kit est d'ailleurs livré avec quatre petits circuits imprimés satellites comportant chacun une prise RJ-45 et des broches pour les enficher dans les plaques d'essai sans soudure, ou bien dans un module capteur ou une autre carte Arduino. Il n'y a aucun composant actif ni sur le bouclier, ni sur les satellites. Il s'agit purement d'un système de distribution de signaux.

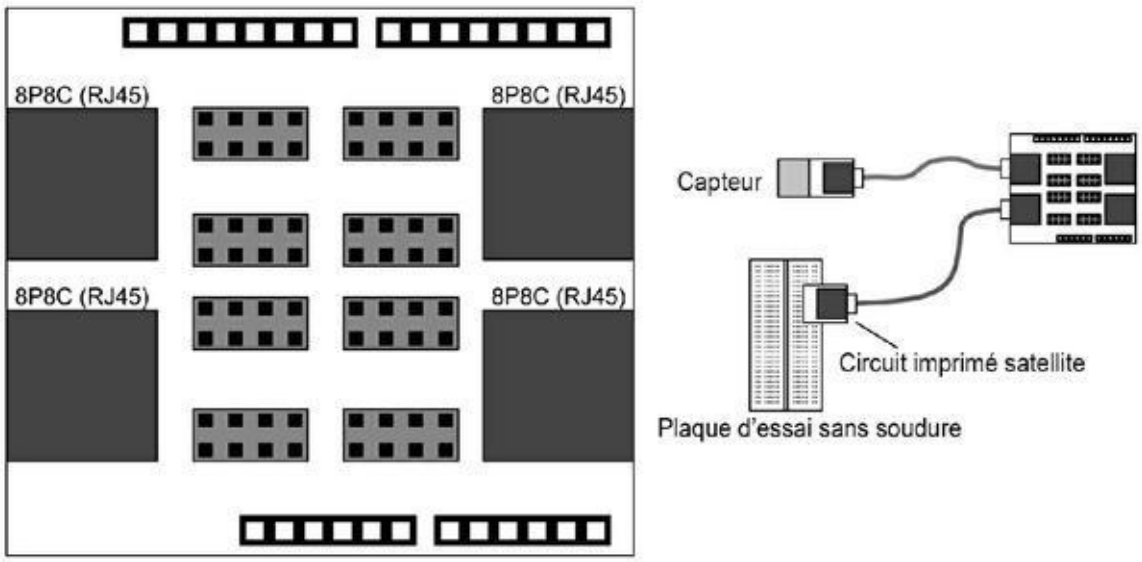


Figure 8.22 : Implantation et utilisation du bouclier Adafruit Patch Shield.

Bouclier Mayhew Labs Go-Between

Ce bouclier (Figure 8.23) présente une grande matrice de trous à souder grâce à laquelle vous pouvez rediriger des signaux depuis la carte Arduino ou un bouclier situé en dessous et un bouclier enfiché par-dessus. C'est donc un bouclier intermédiaire (Go-*Between*), qui peut s'avérer pratique lorsque vous devez associer deux boucliers qui utilisent les mêmes broches pour leurs entrées-sorties. En renvoyant les broches du bouclier supérieur sur d'autres broches, vous résolvez le problème de conflit d'utilisation des broches. Il suffit ensuite d'effectuer des retouches au niveau du logiciel pour que l'ensemble fonctionne en harmonie.

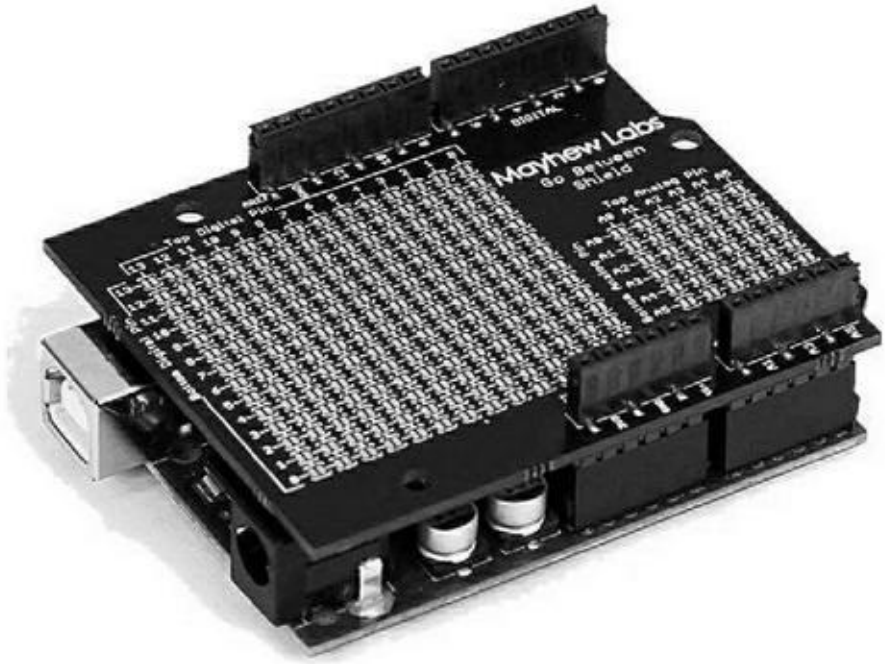


Figure 8.23 : Bouclier Mayhew Labs Go-Between.

Bouclier Mayhew Labs Mux Shield II

Il s'agit d'un bouclier actif (Figure 8.24) qui permet d'exploiter jusqu'à 48 entrées ou sorties grâce à trois circuits multiplexeurs Texas

Instruments CD74HC4067 et à trois circuits registres à décalage pour les sorties. Le bouclier ne consomme que quatre broches numériques de l'Arduino pour contrôler les multiplexeurs et les registres. Par défaut, ce sont les broches numériques D2, D4, D6 et D7. Les broches analogiques A0, A1 et A2 de l'Arduino servent d'entrées pour les multiplexeurs.

Un des bords de la carte comporte trois rangées de 16 plots à souder (Figure 8.25) qui peuvent être équipés de broches mâles ou femelles. Tous les canaux sont bidirectionnels, et les signaux peuvent être routés vers une sortie commune ou depuis une entrée commune. Chacun des multiplexeurs peut se comparer à une matrice d'interrupteurs, chacun des interrupteurs offrant une faible résistance lorsqu'il est fermé et une très forte impédance lorsqu'il est ouvert.

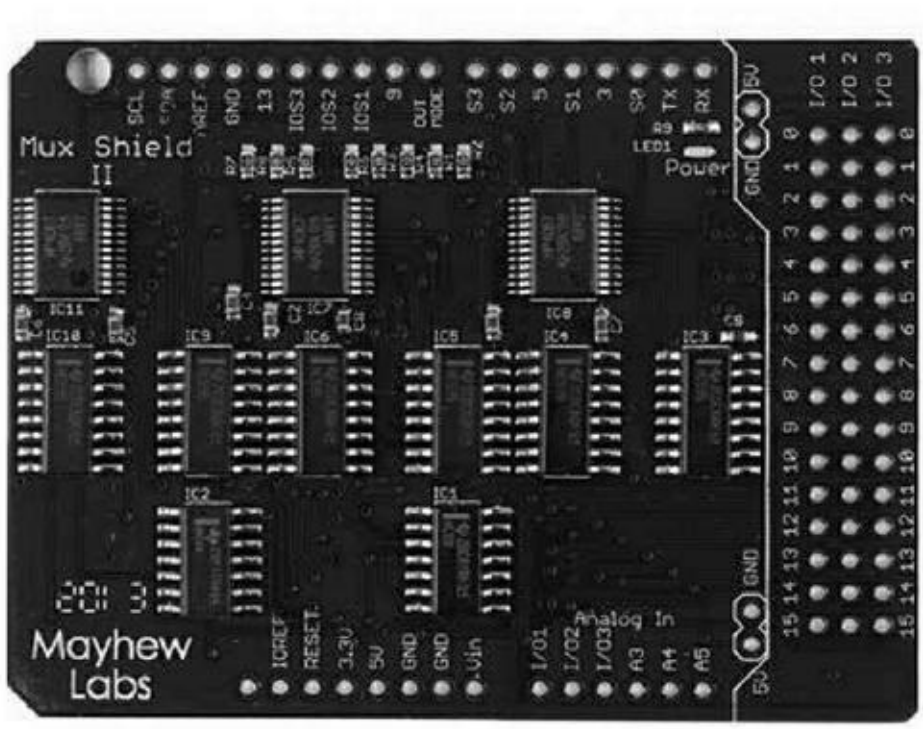


Figure 8.24 : Bouclier multiplexeur de signaux actif Mayhew.

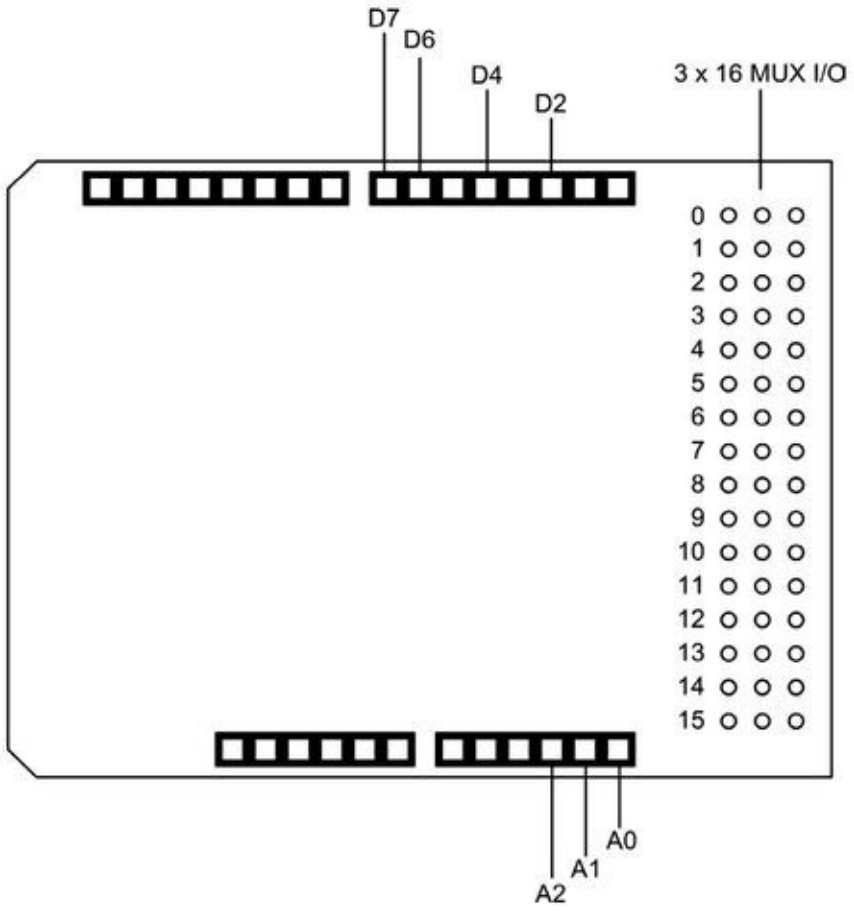


Figure 8.25 : Brochage du bouclier Mayhew.

Bouclier Mayhew Labs Mux Shield

Ce bouclier ressemble beaucoup au précédent en offrant 48 entrées-sorties programmables grâce à trois circuits multiplexeur CD75HC4067 (Figure 8.26). Les signaux sont disponibles sur deux grandes rangées de connecteurs. Le bouclier utilise les broches numériques D2 à D5 de l'Arduino pour contrôler les multiplexeurs et les broches A0, A1 et A2 pour les entrées analogiques. Le bouclier est empilable.

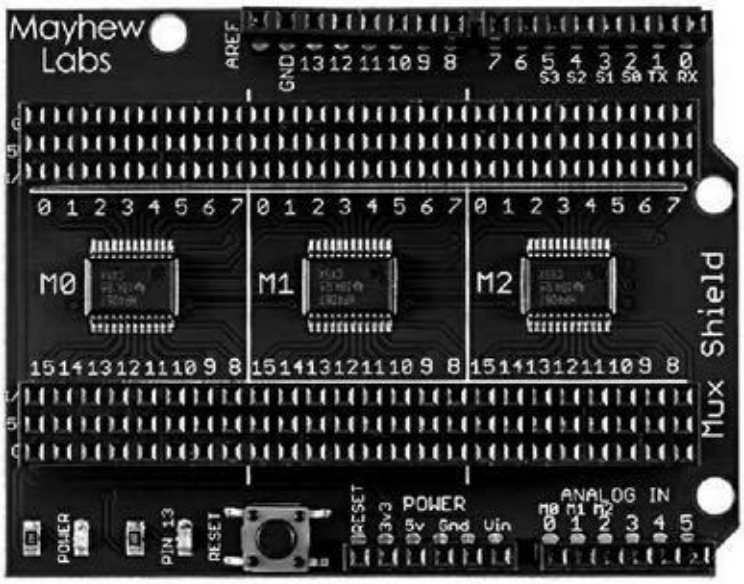


Figure 8.26 : Bouclier multiplexeur Mayhew Labs Mux Shield.

Boucliers mémoire

Les cartes de mémoire flash les plus répandues de nos jours sont sans nul doute celles aux formats SD et microSD. Elles permettent d'ajouter aisément des capacités de stockage de type fichier à un circuit Arduino. L'accès à cette mémoire flash utilise l'interface SPI. D'ailleurs, de nombreux boucliers qui exploitent le protocole SPI pour leurs fonctions principales (Ethernet, Wifi, USB, *etc.*) embarquent un porte-carte mémoire à ce format. Les circuits mémoire amovibles flash sont très pratiques pour collecter des données depuis un circuit Arduino autonome, afin de les transférer ensuite vers un ordinateur et en assurer l'exploitation.

Cette section ne comportera pas de schéma de brochage spécifique, puisque le format de carte SD n'est en fait qu'une interface SPI. Sachez cependant qu'un des boucliers a choisi d'implanter le signal de sélection sur une broche inhabituelle, ce qui peut créer un conflit avec les logiciels existants.

Bouclier Seeed Studio SD card

Ce bouclier est prévu pour les cartes mémoire flash SD dans l'ancien grand format (Figure 8.27). Vous pouvez facilement l'utiliser avec une carte microSD

avec un adaptateur. L'interface SPI est exploitée sur les broches D4, D11, D12 et D13. Le bouclier en profite pour rendre disponibles sur des connecteurs les signaux ICSP, I2C et UART. Le bouclier est empilable.

Figure 8.27 : Bouclier de carte mémoire Seeed Studio SD Memory.

Bouclier SHD-SDSD card

Ce bouclier (Figure 8.28) comporte une petite zone de prototypage, ce qui permet d'ajouter quelques circuits, et accepte les cartes au format microSD avec un adaptateur. Le bouclier exploite pour l'interface SPI les broches D10, D11, D12 et D13 de la carte Arduino ainsi que la broche 3,3 V. Le bouclier est moins long que le format standard, et utilise le brochage d'enfichage de base. Il est empilable.

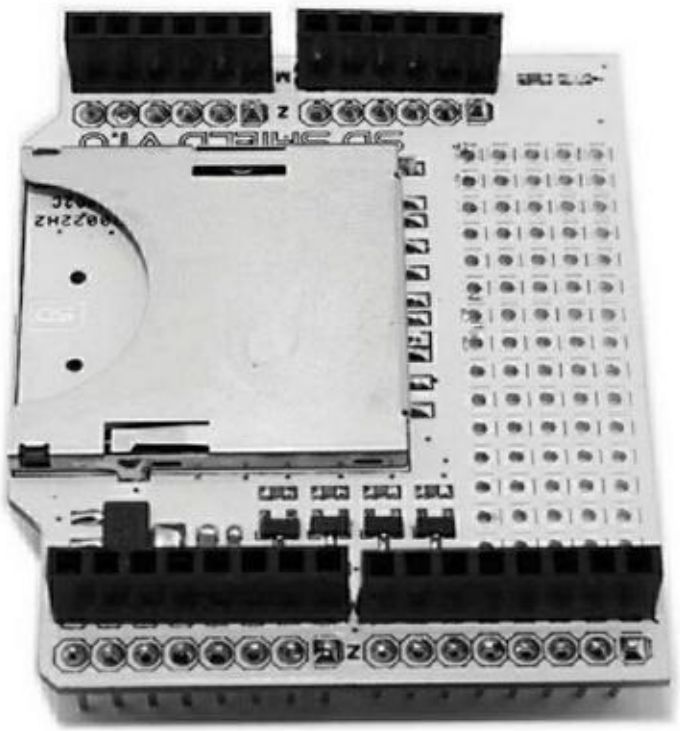


Figure 8.28 : Bouclier de carte mémoire SHD-SD au format court.

Bouclier SparkFun microSD

Ce bouclier n’accepte que les cartes au format microSD (Figure 8.29). Il offre une assez grande zone de prototypage de 12 sur 13 et n’est pas équipé de connecteurs, mais vous pouvez en ajouter. L’interface utilise les quatre broches D10, D11, D12 et D14 (au lieu de D13).

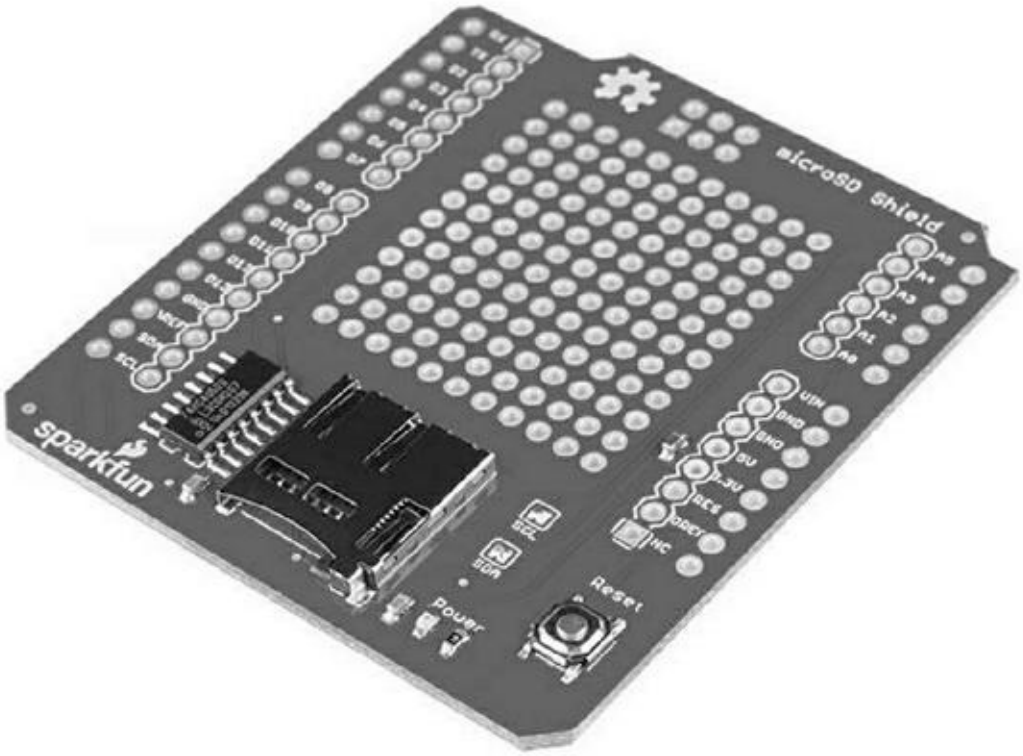


Figure 8.29 : Bouclier mémoire SparkFun microSD.

Boucliers de communication

De nos jours, quasiment tous les modèles de cartes Arduino disposent d'une interface USB qui peut être exploitée comme port série, du point de vue de la machine hôte. Certaines disposent d'une prise Ethernet. Cela dit, les modèles de base comme l'Uno ou la Leonardo n'ont pas beaucoup de dispositions natives en termes d'interfaces de communication. Il est bien sûr possible de leur associer des circuits de conversion de niveau des signaux en utilisant le port UART interne, et d'écrire un logiciel spécifique pour échanger des données série. Il est cependant en général plus pratique de déléguer ce travail à un module comportant une interface SPI vers l'Arduino, le module se chargeant de la gestion du canal série. D'ailleurs, les protocoles de communication plus complexes requièrent du matériel supplémentaire qu'il vaut mieux acquérir sous forme d'un bouclier prêt à l'emploi.

Communications série et MIDI

Le standard Ethernet peut être considéré comme une sorte de communication série, mais au sens strict, les liens série correspondent aux anciens standards RS-232 et RS-485, qui restent très utilisés. Les anciens PC possédaient tous un connecteur RS-232, et certaines tours en sont toujours dotées. Quant au RS-485, il reste très utilisé dans le monde de l'industrie, des tests et des laboratoires.

Bouclier CuteDigi RS-232

Ce bouclier (Figure 8.30) exploite un circuit MAX232 pour convertir les niveaux des signaux à transmettre et à recevoir par la liaison RS-232. Vous disposez d'un groupe de cavaliers pour configurer l'interface. Vous pouvez exploiter l'interface sur n'importe quel couple de broches numériques entre D0 et D7. En installant les broches fournies, vous en faites un bouclier empilable.

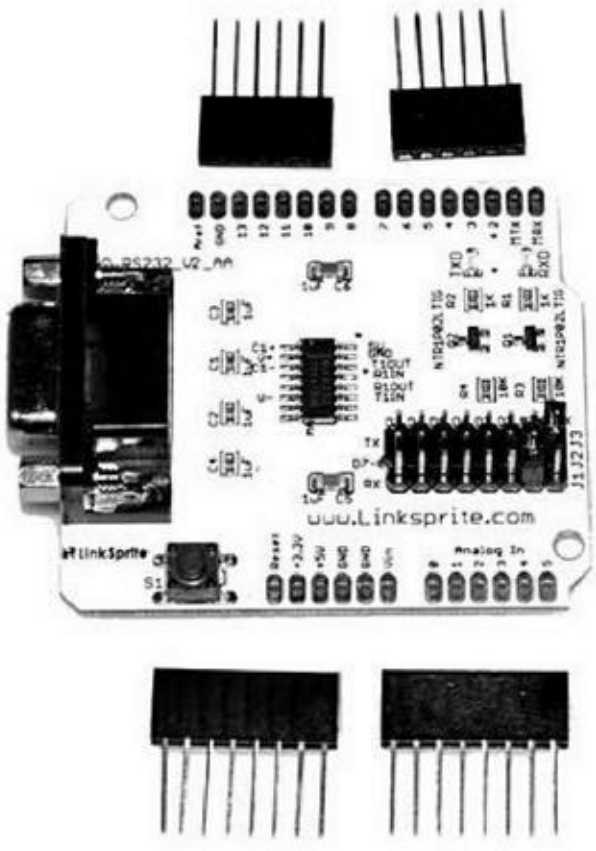


Figure 8.30 : Bouclier CuteDigi RS-232.

Bouclier CuteDigi RS-485

Ce bouclier (Figure 8.31) utilise le circuit MAX481CSA pour gérer une liaison RS-485 sur les deux ports d'émission Tx et de réception Rx de l'Arduino, deux broches au choix entre D0 et D7. Une place est prévue sur le circuit pour ajouter un connecteur DB-9. Le bouclier est empilable.

Figure 8.31 : Bouclier CuteDigi RS-485.

Bouclier SparkFun MIDI

MIDI est le nom d'une norme de communication qui existe depuis plus de 30 ans dans le monde des synthétiseurs, séquenceurs et boîtes à rythmes des musiciens (elle est également utilisée dans le monde de l'éclairage). Ce bouclier (Figure 8.32) utilise les broches du port USART de l'Arduino pour émettre et recevoir des messages d'événements MIDI.

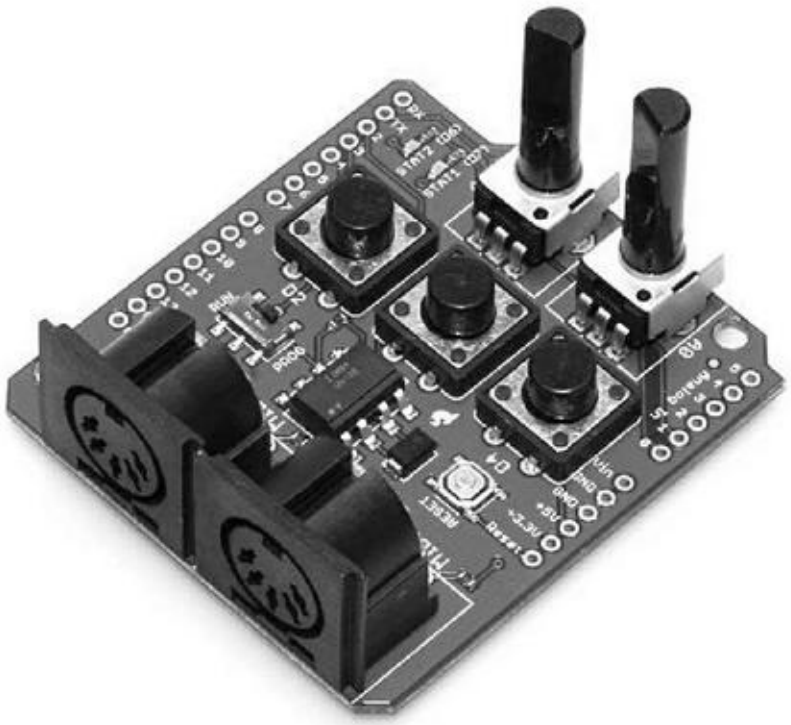


Figure 8.32 : Bouclier SparkFun MIDI.

Le bouclier occupe les broches D0 pour la réception et D1 pour l'émission MIDI. Il offre trois boutons poussoirs reliés à D2, D3 et D4, deux diodes LED connectées à D6 et D7 et deux potentiomètres connectés à A0 et A1.

Ethernet

Les boucliers Ethernet sont très demandés, et l'atelier IDE Arduino est livré avec une librairie de fonctions Ethernet très complète, que j'ai décrite dans le Chapitre 7. Sachez que le dialogue entre le microcontrôleur et le contrôleur Ethernet du bouclier exploite l'interface SPI. Il n'y a pas d'accès direct mémoire DMA dans le microcontrôleur, et il n'a de toute manière pas de mémoire externe permettant un accès direct.

Le fait que les boucliers utilisent l'interface SPI avec Arduino impose une limite à la vitesse maximale de transfert des données entre le microcontrôleur et le circuit Ethernet, et donc une limite à la vitesse de la connexion Ethernet. Il est impossible d'obtenir un débit de 100 Mb/s avec un processeur qui fonctionne à 20 MHz et une interface SPI. Même un débit de 10 Mb/s est difficile à atteindre, il vaut mieux viser les 5 Mb/s. Les données sont transmises au niveau

de la couche physique Ethernet à la vitesse de 10 Mb/s, mais elles le sont octet par octet, et pas sous forme de flux continu. Les performances exactes dépendent de la vitesse à laquelle le logiciel qui tourne sur le microcontrôleur parvient à regrouper les données avant de les transmettre au circuit Ethernet. Il n'est pas impossible de créer un serveur Web qui tienne dans une petite boîte, mais n'espérez pas trop de performances, ni surtout trop de connexions simultanées.

L'interface Ethernet va s'avérer très pratique lorsque vous l'utiliserez en tant que terminal d'un système de détection à distance ou de contrôle. Vous connectez votre montage au réseau Internet, vous ajoutez une mesure de protection par mot de passe, et vous pouvez ensuite récupérer des données à distance. Cela permet par exemple de collecter des données vers un point central dans un environnement industriel ou bien de surveiller des températures, des taux d'humidité et d'autres paramètres pour un système de climatisation, tel que celui que je présente dans le Chapitre 12.

Vetco Ethernet Shield avec lecteur microSD

Le bouclier Ethernet de Vetco (Figure 8.33) comporte un lecteur de carte microSD et un bouton Reset. L'interface SPI utilise les broches numériques D10, D11, D12 et D13 au bénéfice du contrôleur Ethernet WIZnet W5100 et du lecteur de carte microSD. Le bouclier est empilable.

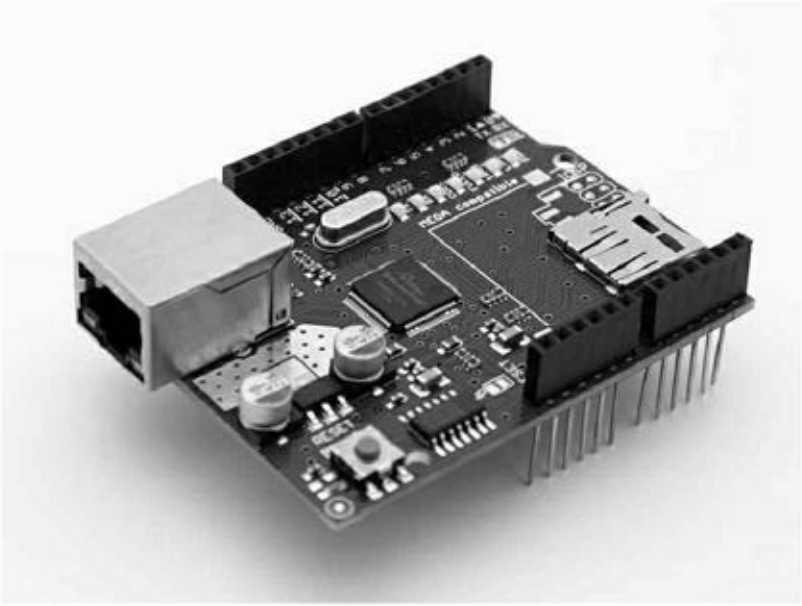


Figure 8.33 : Bouclier Vetco Ethernet.

Arduino Ethernet Shield R3 avec connecteur microSD

Le bouclier Ethernet officiel Arduino comporte toute l'électronique pour gérer une connexion Ethernet, ainsi qu'un lecteur de carte microSD et un bouton Reset (Figure 8.34). L'interface SPI utilise les quatre broches numériques D10 à D13. Le bouclier est empilable.

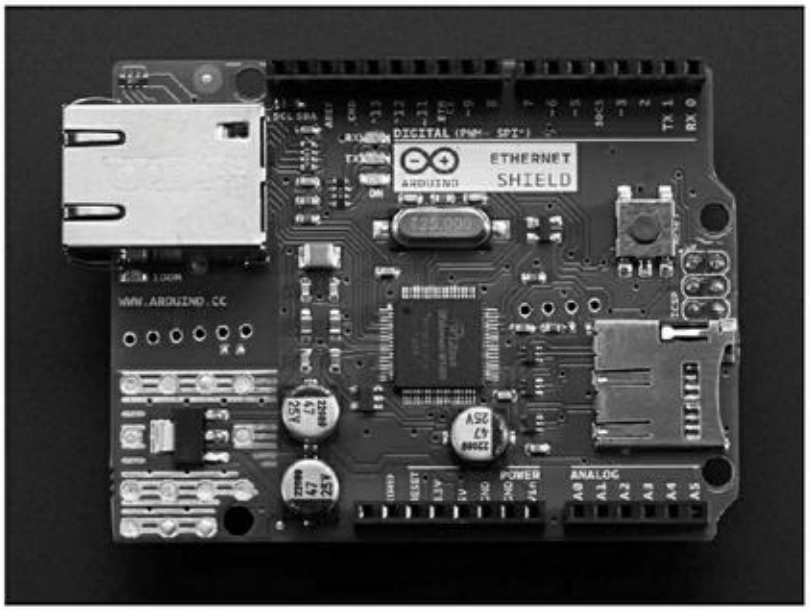


Figure 8.34 : Bouclier Arduino Ethernet.

Bluetooth

La norme Bluetooth est une technologie de communication sans fil à courte distance peu consommatrice d'énergie. Au départ, son but était de faire disparaître tous les câbles entre l'unité centrale d'un ordinateur et tous ses périphériques (imprimante, clavier, souris et haut-parleurs). La norme est effectivement utilisée dans ce but, mais également de plus en plus dans d'autres domaines, notamment pour les enceintes sans fil et les échanges entre telliphones. Vous trouverez un vaste choix de boucliers Bluetooth.

Bouclier Bluetooth

Ce bouclier comporte déjà un module Bluetooth sur le circuit. C'est un bouclier court et empilable. Son antenne correspond à la partie dorée qui dépasse du module Bluetooth sur la Figure 8.35. Le dialogue s'effectue avec les deux broches Rx et Tx sur les broches D2 et D3.

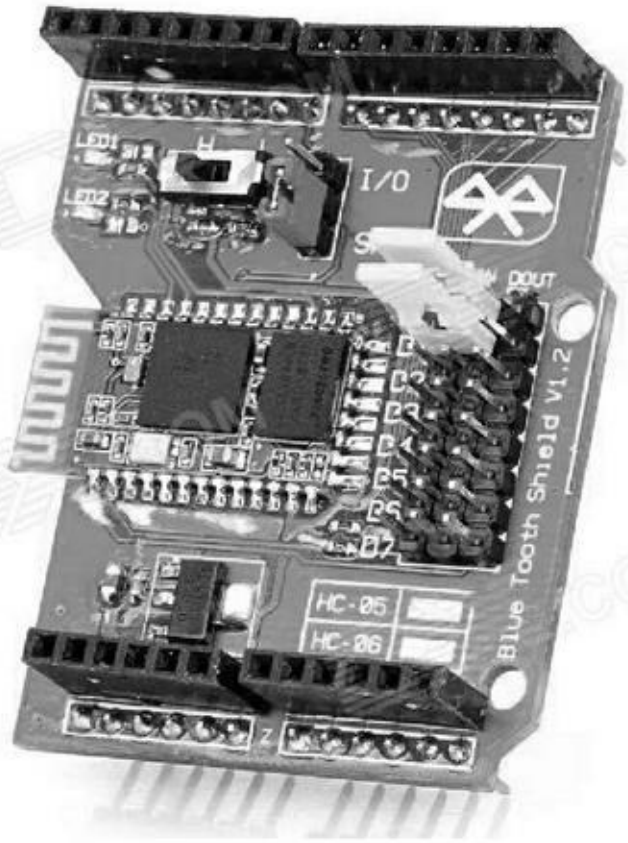


Figure 8.35 : Bouclier DealXtreme Bluetooth.

Bouclier Seeed Studio Bluetooth

Ce bouclier court et enfichable (Figure 8.36) exploite les deux broches Rx et Tx de la carte Arduino. Elle donne également accès aux broches analogiques et numériques pour y brancher des capteurs.

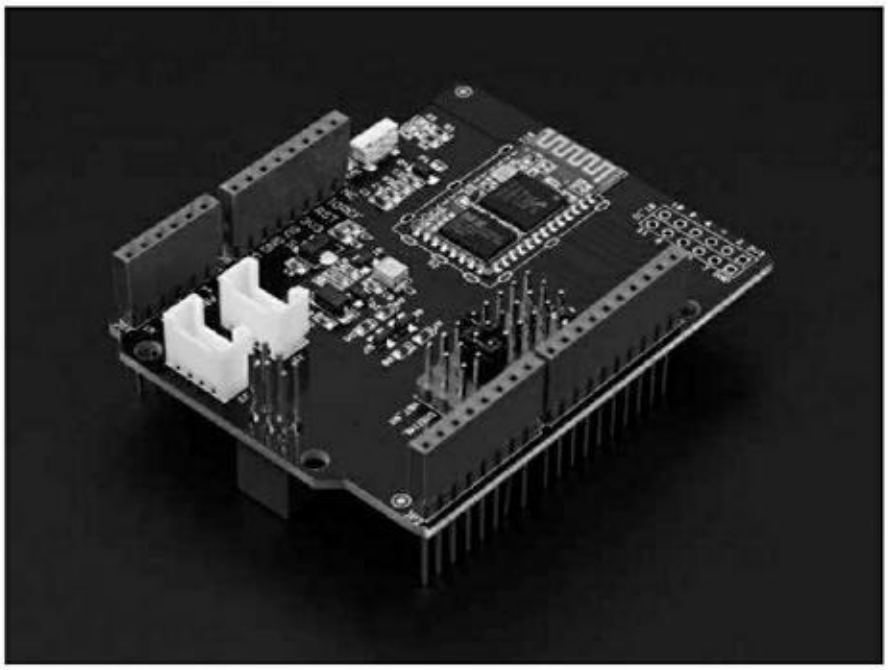


Figure 8.36 : Bouclier Bluetooth compact de Seeed Studio.

ITEAD Bluetooth Wireless BT Module (en kit)

Ce bouclier, utilisant un module Bluetooth standard (Figure 8.37), comporte une zone de prototypage. C’est un bouclier court et empilable.

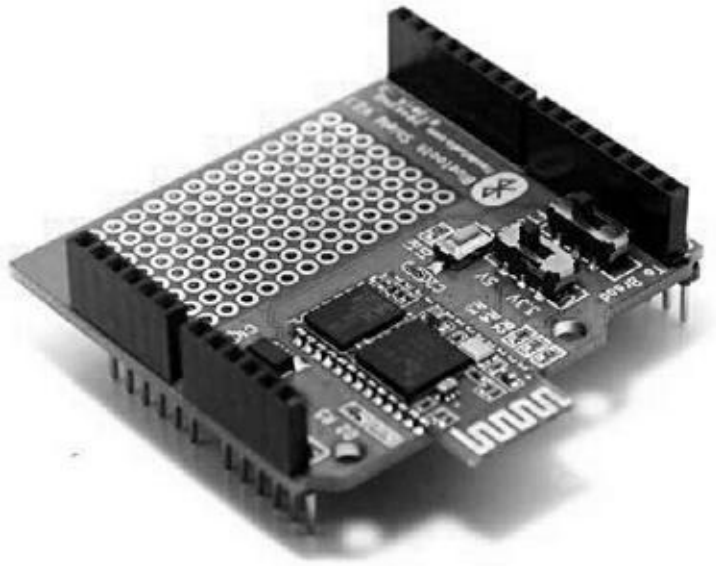


Figure 8.37 : Bouclier ITEAD Bluetooth avec zone de prototypage.

Bouclier DFRobot Gravity:IO Expansion

Cette carte multifonction de DFRobot (Figure 8.38) réunit des capacités d'extension des entrées-sorties et des connecteurs permettant d'ajouter un module sans fil Bluetooth ou ZigBee. Elle n'est pas empilable.

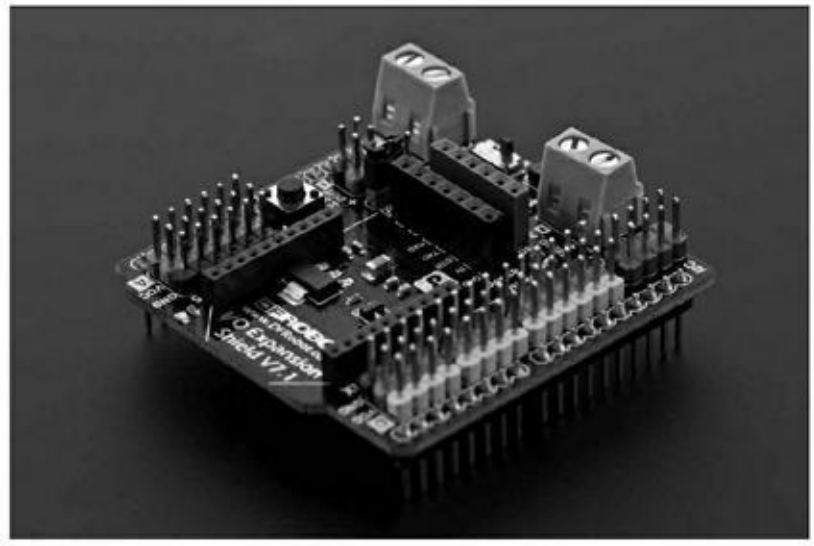


Figure 8.38 : Bouclier multifonction DFRobot avec Bluetooth.

USB

Une carte Arduino sur 8 bits peut dialoguer avec une machine hôte par sa liaison USB, mais elle ne peut pas devenir un hôte USB à son tour, ce qui permettrait d'y brancher des périphériques USB comme un clavier, une imprimante, un instrument de test ou un jouet.

Bouclier ITEAD USB Host

Ce bouclier empilable ajoute des capacités d'hôte USB et rend disponibles les broches numériques et analogiques de l'Arduino. Il offre aussi deux rangées de 10 plots sur le circuit pour ajouter des connecteurs mâles ou femelles (Figure 8.39). Il fonctionne avec une interface SPI et un circuit MAX3421E.

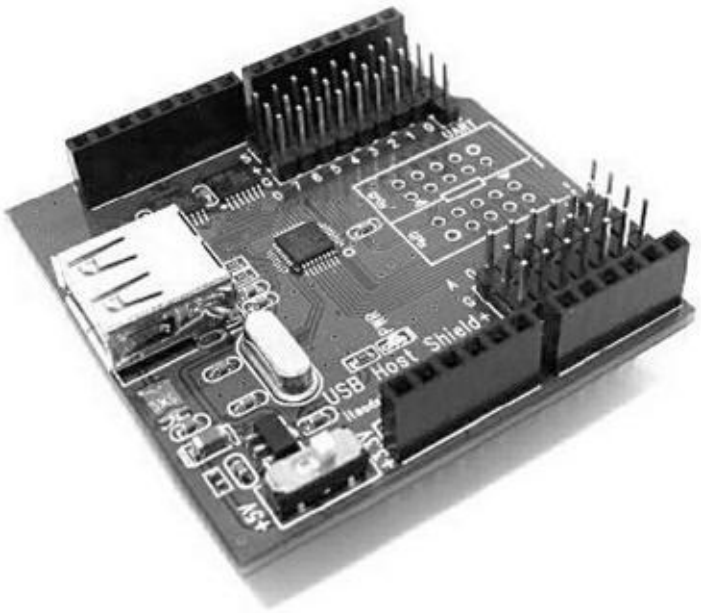


Figure 8.39 : Bouclier ITEAD USB Host avec connexions d'entrées-sorties.

Bouclier Circuits@Home USB Host

Ce bouclier utilise l'interface SPI pour travailler à la vitesse nominale de la norme USB 2.0 (Figure 8.40). Toutes les entrées-sorties analogiques et numériques Arduino sont disponibles sur le bouclier. Vous pouvez le rendre empilable en ajoutant les connecteurs appropriés. Il utilise le circuit MAX3421E sur les broches D10 à D13. Les connecteurs pour empilement ne sont pas fournis.

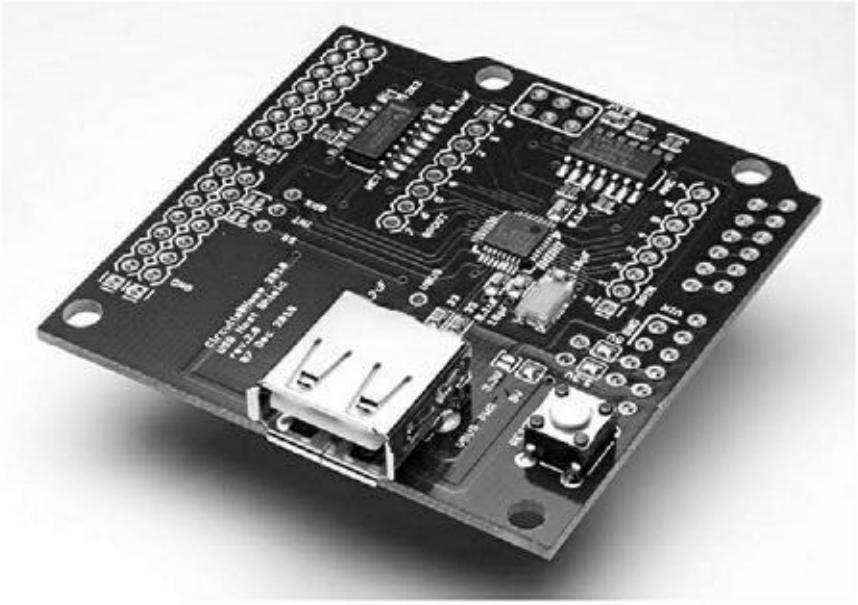


Figure 8.40 : Bouclier Circuits@Home USB Host.

Bouclier Arduino USB Host

Comme ses collègues, cette carte utilise le circuit MAX3421E avec une interface SPI sur les broches D10 à D13. Plusieurs connecteurs à trois et quatre contacts offrent des ports d'entrée et de sortie directement compatibles avec les modules de la famille TinkerKit qui ont été décrits dans ce même chapitre. Le bouclier est empilable (Figure 8.41).

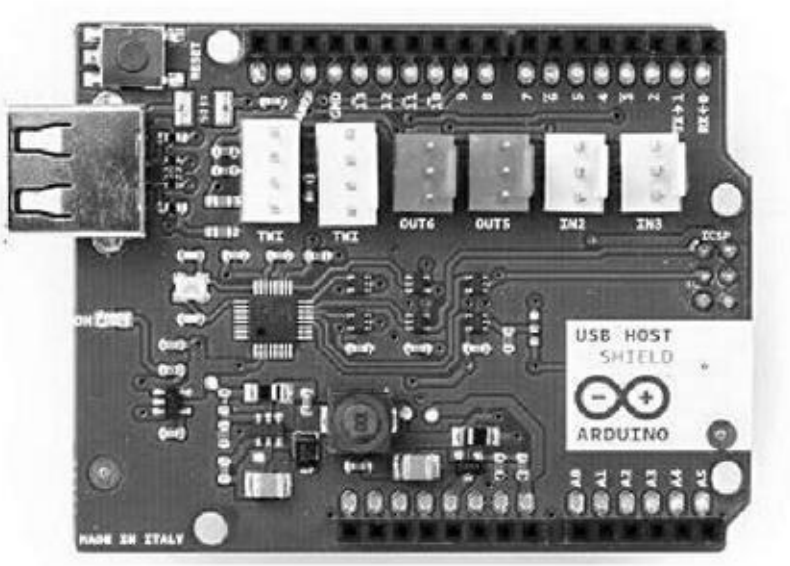


Figure 8.41 : Bouclier Arduino USB Host avec connecteurs d'entrées-sorties.

ZigBee

ZigBee est un protocole sans fil à faible énergie assez répandu. La plupart des boucliers ZigBee pour Arduino se basent sur un module existant, mais la plupart des boucliers peuvent être utilisés avec un autre module émetteur/récepteur si ce dernier possède des sorties compatibles. Certains boucliers sont livrés avec le module XBee, mais d'autres ne l'ont pas au départ. Il faut savoir qu'un module XBee d'une puissance de 1 milliwatt coûte de l'ordre de 25 euros.

Bouclier Arduino Wireless SD

Ce bouclier ZigBee (Figure 8.42) comporte deux rangées de connecteurs pour recevoir un module XBee de Digi ou tout module compatible. Le bouclier utilise la broche D4 Arduino pour le signal de sélection et les broches D11 à D13 pour le protocole SPI. Cette interface est également utilisée par le lecteur de carte microSD présent sur le bouclier.

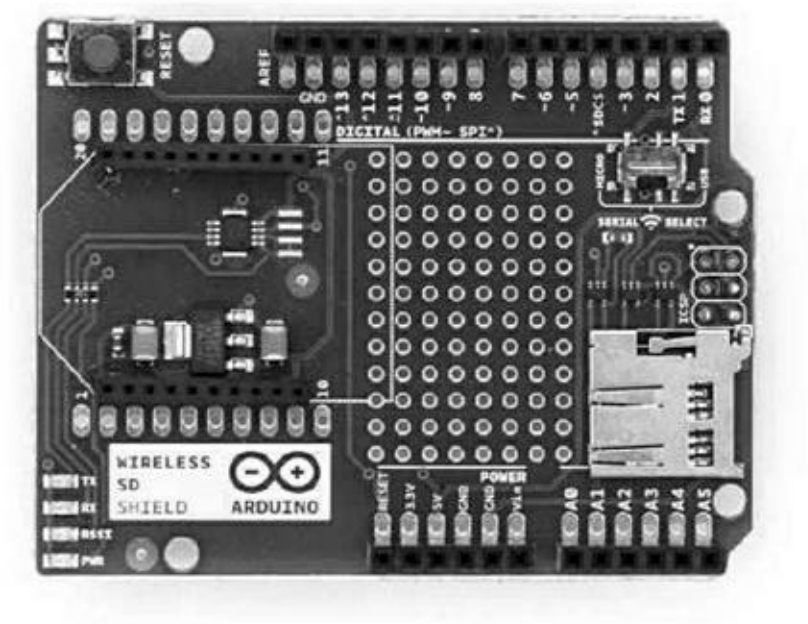


Figure 8.42 : Bouclier Arduino ZigBee.

Bouclier SainSmart ZigBee

Ce bouclier est prévu pour le module XBee, mais le connecteur permet d'utiliser tout module standard et compatible avec ce brochage. Il n'y a pas de lecteur microSD. C'est une carte à décalage d'enfichage. Vous pouvez deviner les positions des connecteurs dans la Figure 8.43.

Figure 8.43 : Bouclier compact SainSmart ZigBee.

Bouclier Seeed Studio XBee

Ce bouclier de Seeed Studio (Figure 8.44) accepte les modules XBee standard et comporte une zone de prototypage. Le dialogue avec la carte Arduino utilise les deux broches Rx et Tx. Une série de connecteurs pour cavaliers permet de rediriger ces deux signaux depuis l'Arduino vers le module sans fil.

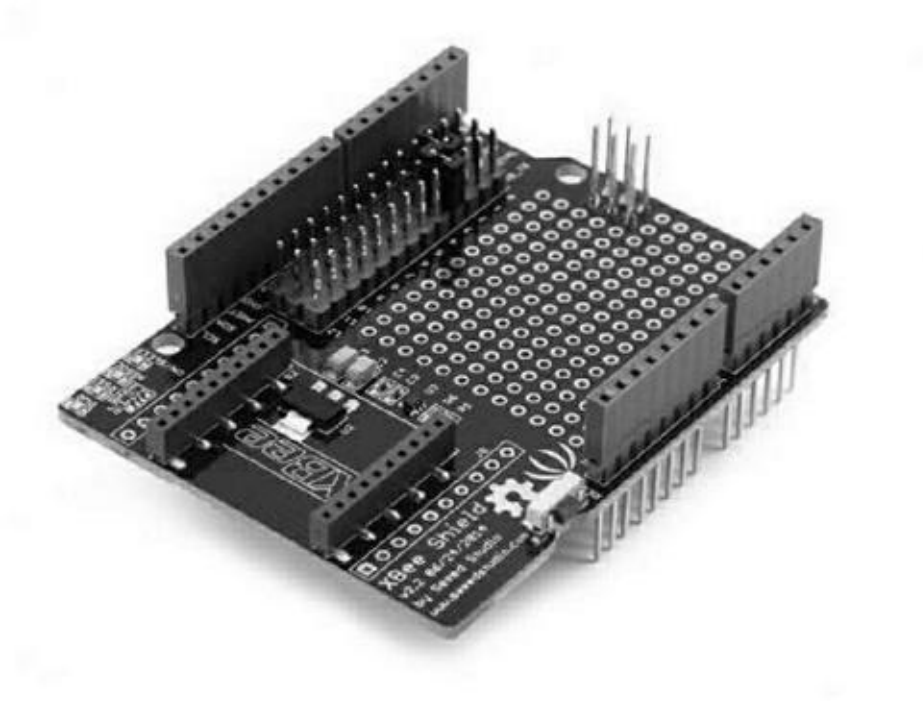


Figure 8.44 : Bouclier Seeed Studio XBee avec zone de prototypage.

Protocole CAN

Le bus CAN (*Controller Area Network*) est une variation du protocole RS-485 différentiel. Il est très utilisé dans le monde de l'automobile, de l'industrie et les équipements militaires. Il offre un débit allant jusqu'à 1 Mbit/sec. La norme CAN dispose d'un mécanisme de détection des collisions et des erreurs et elle accepte plusieurs noeuds réseau. Elle est notamment utilisée dans le standard OBD-II des prises de diagnostic des automobiles récentes, de tous les véhicules électriques, des capteurs distribués dans les instruments scientifiques et même dans les vélos haut de gamme.

Bouclier Seeed Studio CAN-BUS

Ce bouclier de Seeed Studio (Figure 8.45) embarque un contrôleur de bus CAN du type MCP2515 pour l'interface SPI et un émetteur-récepteur avec le circuit MCP2551. Les signaux de CAN sont disponibles sur borniers et sur un connecteur DB-9. Le bouclier rend accessibles les interfaces I2C et UART de la carte Arduino. Le brochage est montré dans la Figure 8.46.

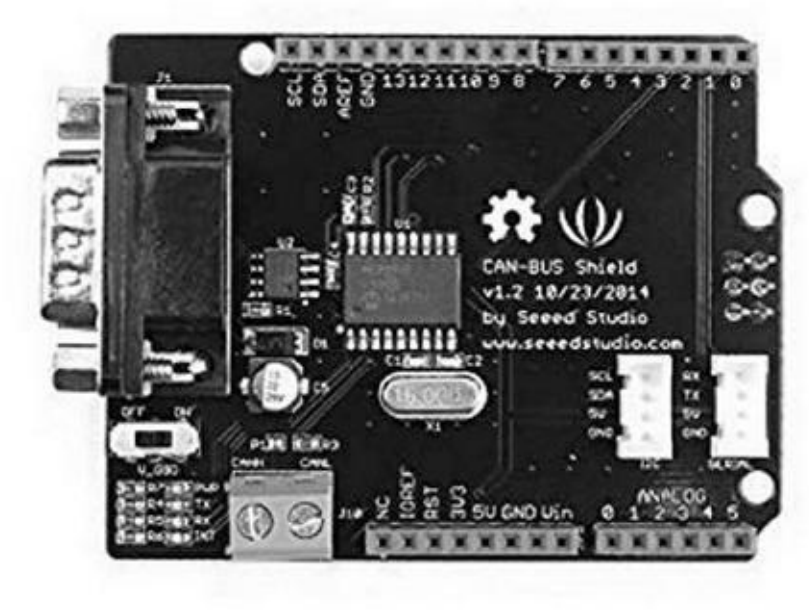


Figure 8.45 : Bouclier Seeed Studio CAN avec connecteurs auxiliaires.

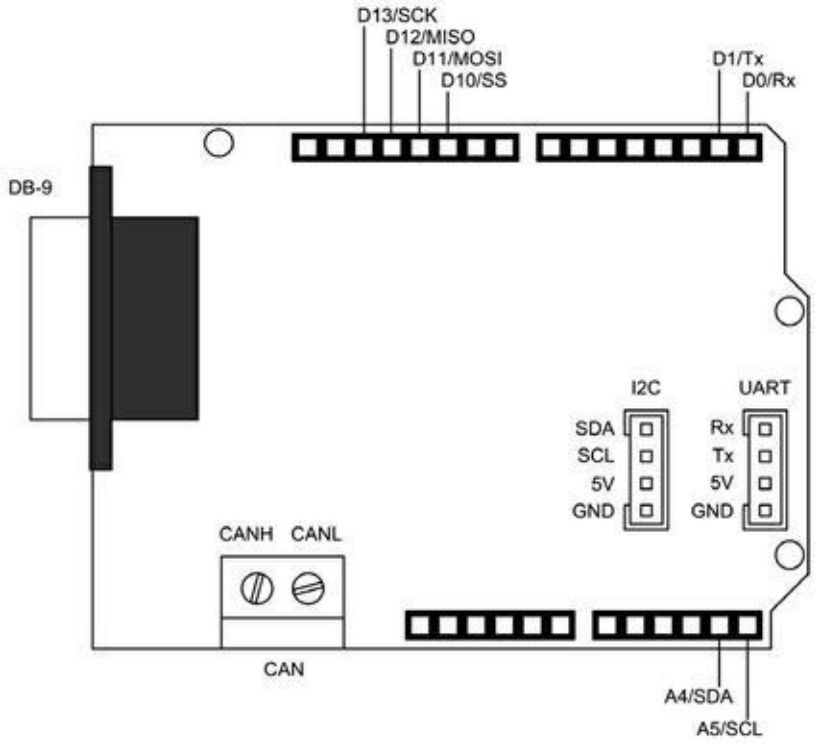


Figure 8.46 : Brochage des sorties du bouclier Seeed Studio CAN.

Bouclier SparkFun CAN-BUS

Ce bouclier de la société SparkFun (Figure 8.47) réunit la plupart des fonctions requises pour créer l'équivalent d'une valise de diagnostic OBD-II pour automobile. Il propose un connecteur DB-9 pour les signaux CAN et un connecteur à quatre broches en complément pour les mêmes signaux. Des connexions sont prévues pour brancher un afficheur LCD externe et un module GPS EM406.

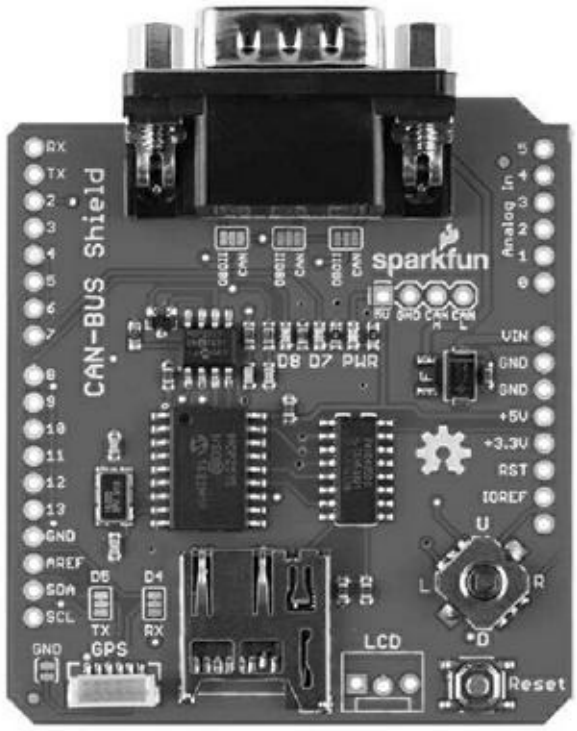


Figure 8.47 : Bouclier SparkFun CAN avec lecteur microSD et joystick numérique.

Un point original de cette carte est le joystick à quatre positions et le lecteur de carte microSD. Le joystick est relié aux entrées analogiques de la carte Arduino. La sélection indépendante du circuit CAN et de la mémoire SD se fait avec les deux broches numériques D9 et D10. Sachez que les broches D3, D4, D5 et A0 ne sont pas utilisées par le bouclier. Un schéma est fourni dans la Figure 8.48.

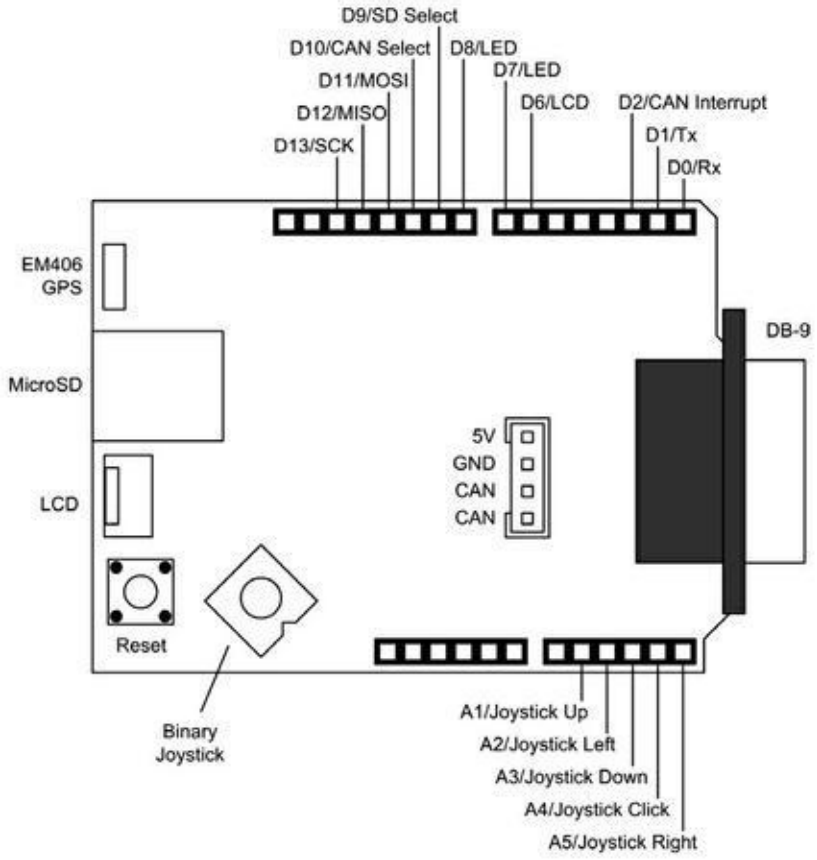


Figure 8.48 : Brochage du bouclier SparkFun CAN.

Bouclier LinkSprite CAN-BUS

Ce bouclier ressemble beaucoup à celui de Seeed (Figure 8.49). Il propose un connecteur DB-9 et un bornier à deux plots pour les signaux CAN. L'interface SPI est exploitée sur les broches D10 à D13 en relation avec le circuit MCP2515.

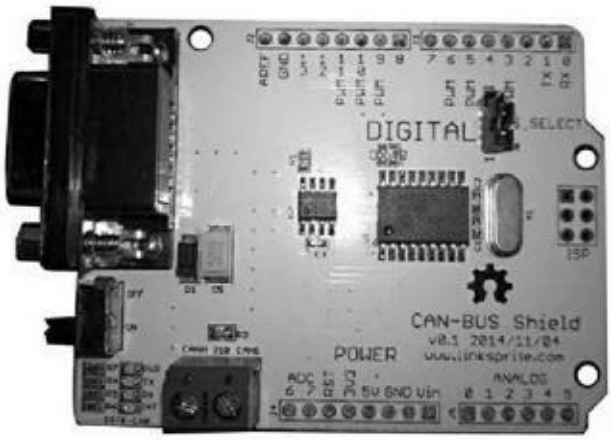


Figure 8.49 : Bouclier LinkSprite CAN.

Cartes de prototypage

Lorsque vous avez besoin de créer un bouclier spécifique, vous pouvez partir d'un bouclier de prototypage. Ce n'est pas la même approche que celle que je propose dans le Chapitre 10, où je montre comment créer un bouclier *ex nihilo* en créant un circuit imprimé, éventuellement pour lancer une fabrication série. Dans la Figure 8.50, vous pouvez voir un bouclier prototype sur lequel j'ai installé un capteur de température et un relais, ainsi qu'un potentiomètre relié au +5 V, à la masse et à l'entrée analogique A0 de la carte Arduino. Le potentiomètre sert à régler le point de consigne de la température.

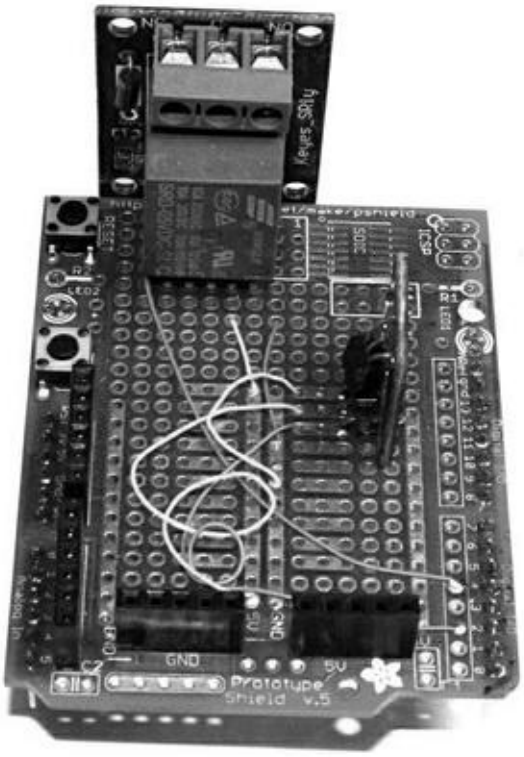


Figure 8.50 : Utilisation d’un bouclier prototype pour un projet de contrôle de température.

J’avais construit cet équipement pour contrôler un vieux radiateur électrique très dangereux dont le thermostat mécanique semblait ne plus réagir qu’à des variations d’au moins 15 °C !

Tous les boucliers de cette section sont assez simples à utiliser. C’est à chacun de choisir comment utiliser les broches Arduino. Je ne donne donc aucun schéma de brochage.

Bouclier Adafruit Stackable R3 Proto

Ce bouclier est livré en kit (Figure 8.51). On vous livre donc le circuit imprimé nu et un sachet de composants, ce qui suppose de savoir un peu manier un fer à souder.

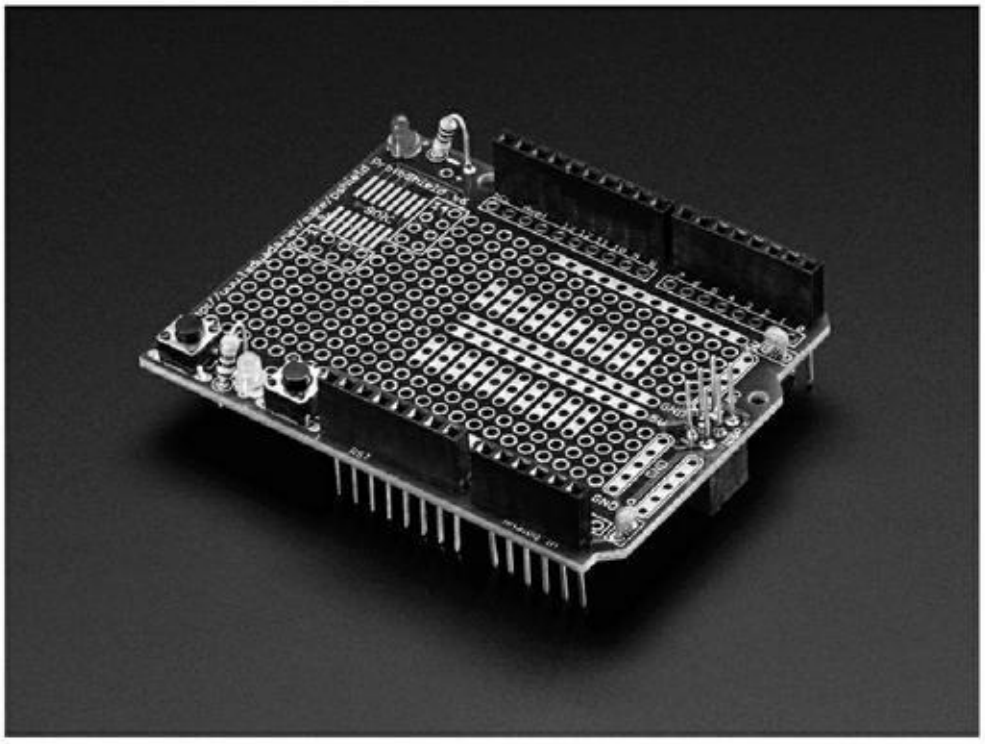


Figure 8.51 : Kit de bouclier Adafruit Stackable R3 Proto une fois monté.

Adafruit Mega Proto

Cet autre bouclier de prototypage de la société Adafruit est lui aussi livré en kit, comme le montre la Figure 8.52. Vous remarquez les doubles rangées de plots à souder sur les bords du circuit. Cela vous permet d'y souder des broches courtes afin de maintenir un bon accès aux signaux qui proviennent de la carte Arduino de type Mega.

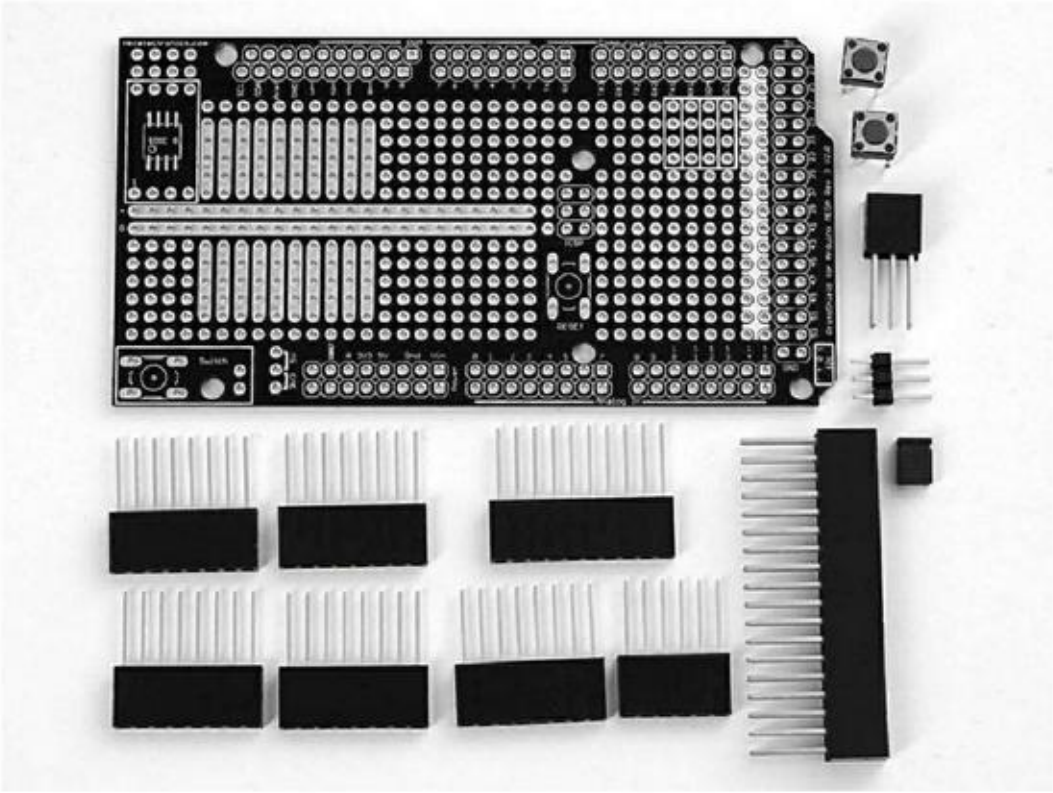


Figure 8.52 : Kit du bouclier Adafruit Mega Proto.

CuteDigi Assembled Protoshield for Arduino

Ce bouclier est livré monté (Figure 8.53). Il n’est pas empilable mais dispose de connecteurs femelles pour accéder aux signaux Arduino.

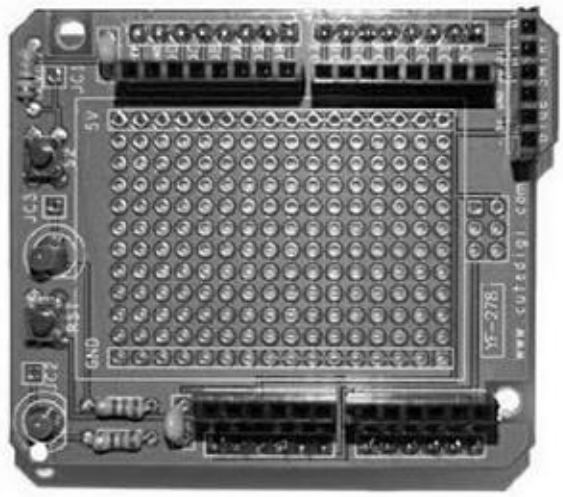


Figure 8.53 : Bouclier de prototypage CuteDigi Assembled Protoshield.

CuteDigi Assembled Protoshield for Mega

Ce bouclier pour carte Arduino au format Mega n'est pas empilable (Figure 8.54). Il se singularise par une zone libre pour y installer un circuit intégré au format SOIC.

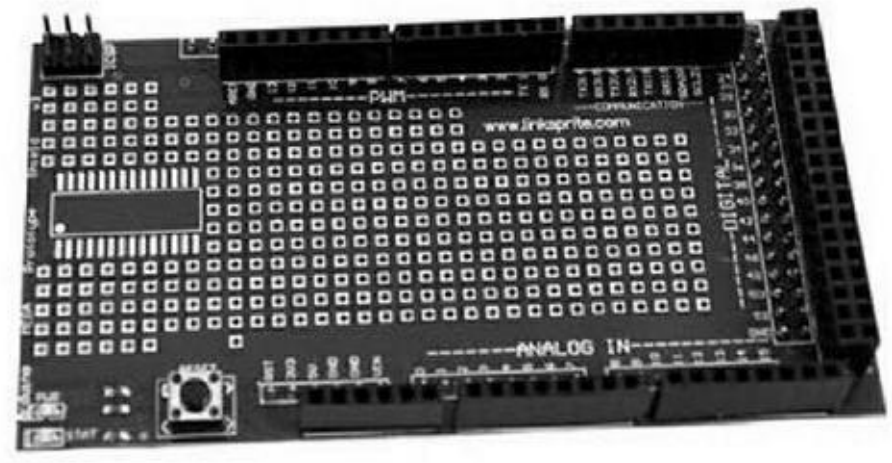


Figure 8.54 : Bouclier de prototypage CuteDigi Assembled Protoshield for Mega.

CuteDigi Protoshield for Arduino avec Mini Breadboard

Ce bouclier comporte une petite plaque de prototypage sans soudure pour y implanter facilement vos créations (Figure 8.55).

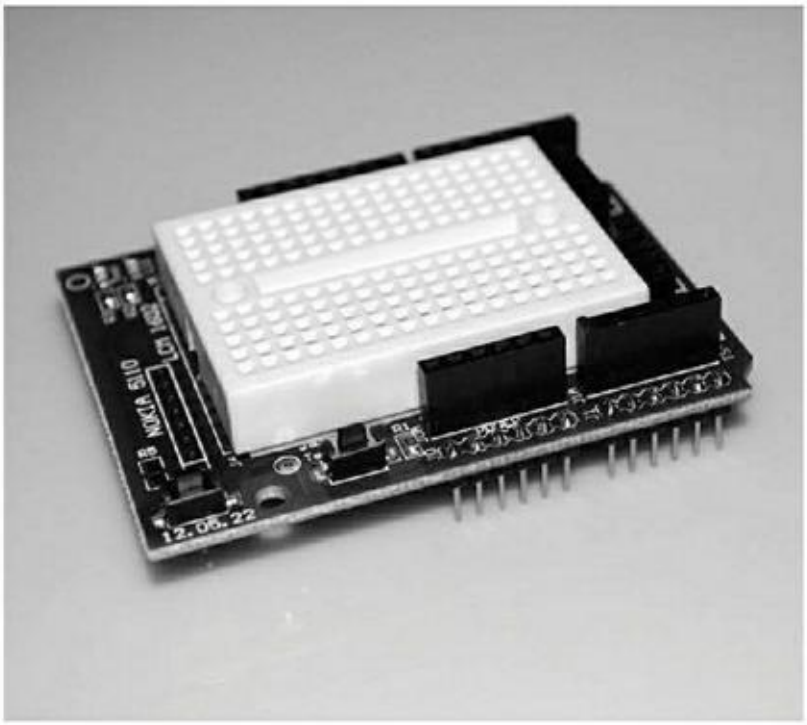


Figure 8.55 : Bouclier CuteDigi Protoshield for Arduino avec plaque d’expérimentation.

Création d’un bouclier de prototypage

Vous pouvez facilement vous construire un bouclier de prototypage avec une plaque à pastilles cuivrées et quelques connecteurs. La taille exacte de votre circuit n’a pas d’importance, à la condition que les deux rangées de broches viennent bien s’enficher dans celles d’une carte Arduino.

Adafruit DIY Shield Kit

Voici ce qui représente vraiment le plus simple bouclier que l’on puisse imaginer. Il réunit une plaque à pastilles et quatre connecteurs longs (Figure 8.56). À vous d’imaginer ce que vous voulez y implanter. Cette solution est excellente pour créer le prototype d’un nouveau bouclier ou se lancer dans une expérimentation. Vous pouvez ainsi vérifier s’il est possible d’utiliser un nouveau module avec une carte Arduino.

La mauvaise nouvelle est que le produit n’est plus distribué par Adafruit. Vous pouvez faire le vôtre avec une plaque à pastilles et les connecteurs appropriés que vous trouverez chez tous les fournisseurs. Je rappelle que les connecteurs

des cartes Arduino sont à la norme industrielle de 2,54 mm d’espacement (un dixième de pouce). Tous les électroniciens possèdent ce genre de connecteurs dans leur stock de base.

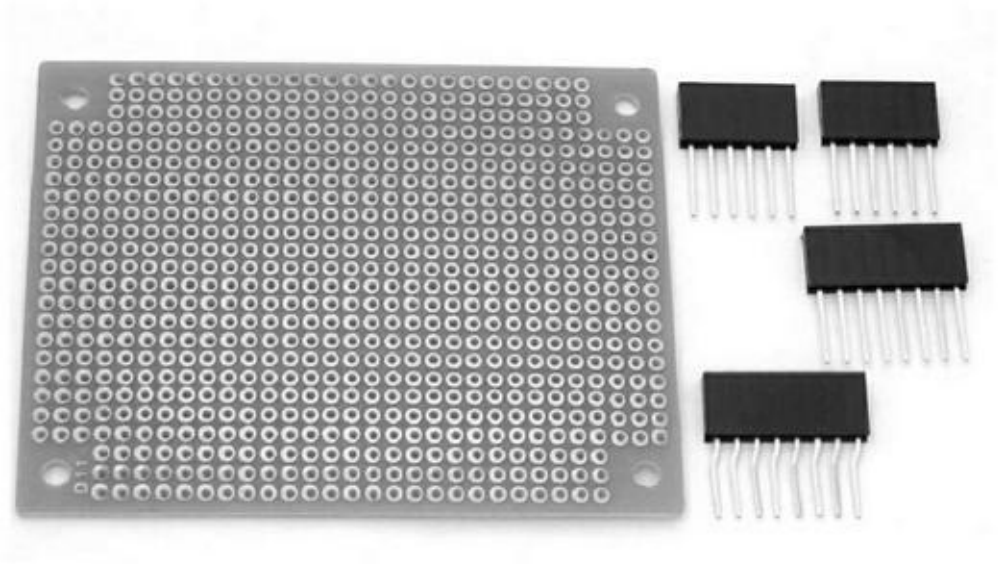


Figure 8.56 : Kit bouclier Adafruit DIY.

Contrôle des mouvements

Les actionneurs et le contrôle des mouvements physiques constituent un des principaux domaines d’activité dans le monde Arduino. Les cartes Arduino sont utilisées depuis les débuts pour contrôler des robots, des machines-outils, des imprimantes 3D, des scanners, des systèmes de suivi du soleil pour panneaux solaires et le pilotage d’événements artistiques. Il est donc peu étonnant de découvrir qu’il existe un vaste choix de boucliers pour les différents types de moteurs électriques. Ce qui suit n’est qu’une toute petite sélection.

Contrôle de moteurs continus et pas à pas

Les cartes boucliers de contrôle de moteurs qui sont basées sur un pont en H (une sorte d’interrupteur de redirection du courant sous forme d’un circuit intégré) permettent de contrôler des moteurs à courant continu avec balai ou des moteurs pas à pas. Plus généralement, ces boucliers sont capables de contrôler n’importe quelle charge inductive en courant continu, et donc aussi des bobines, solénoïdes et relais.

Bouclier Rugged Motor Drive

Ce bouclier, de la société Rugged Circuits (Figure 8.57), peut piloter soit deux moteurs à courant continu à balais, soit un moteur bipolaire pas à pas. Il supporte jusqu'à 30 V et 2,8 A de courant de crête. Le bouclier utilise les broches D3, D11, D12 et D13 pour la sélection et le contrôle du sens de rotation. Les entrées de sélection peuvent être exploitées en mode PWM pour permettre un contrôle progressif du moteur. Pour tout détail, notamment au sujet des courants supportés et du logiciel, voyez le site Web du fabricant.

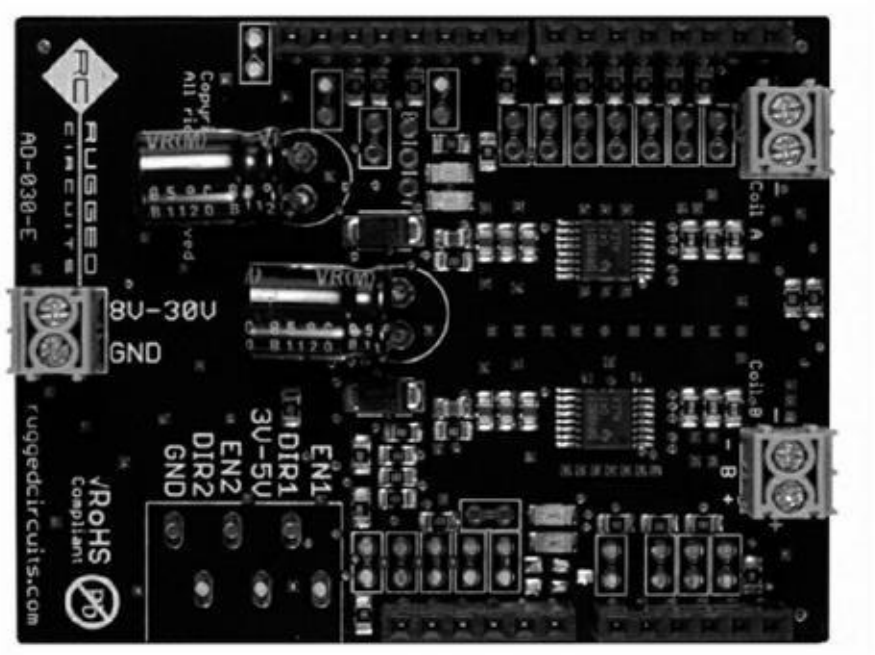


Figure 8.57 : Bouclier Rugged Motor Driver.

Bouclier SainSmart Motor Drive

Ce bouclier embarque un circuit intégré pilote à quatre canaux L293D qui peut donc contrôler jusqu'à quatre moteurs à courant continu et à balais, ou deux moteurs pas à pas avec une tension d'au maximum 10 V. Le circuit comporte des borniers pour une alimentation externe. Pour tout détail concernant les courants maximaux et le logiciel, voyez le site Web du fabricant.

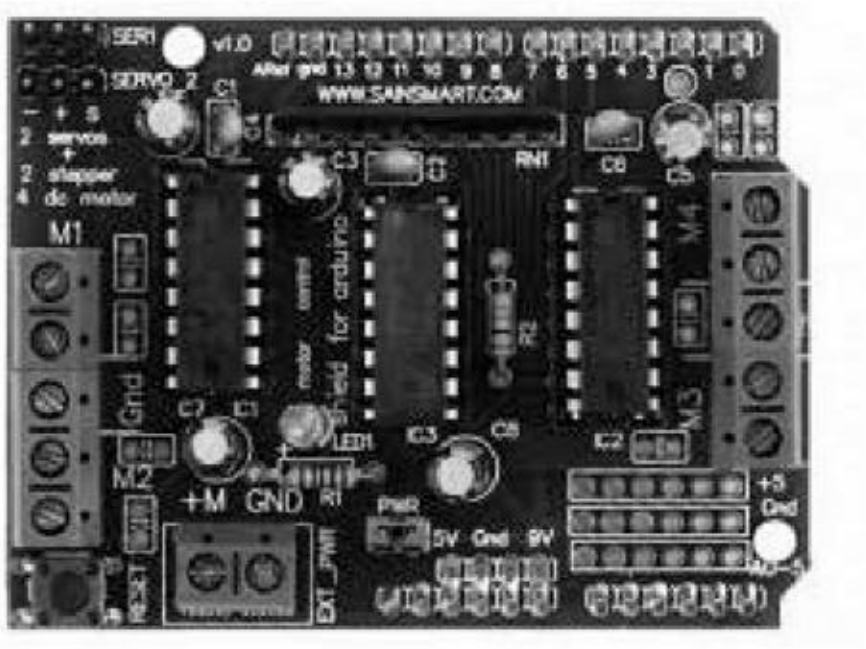


Figure 8.58 : Bouclier SainSmart Motor Drive.

Bouclier Arduino Motor

Le bouclier de contrôle de moteur officiel Arduino (Figure 8.59) utilise un circuit pilote à deux canaux L298. Il peut contrôler des moteurs à courant continu, des moteurs pas à pas, des relais ou des solénoïdes. Sa particularité est de pouvoir mesurer le courant consommé, ce qui permet par exemple de détecter qu'un moteur est bloqué. Il dispose de connecteurs modulaires compatibles avec les modules de la famille TinkerKit, décrits plus haut dans ce chapitre.

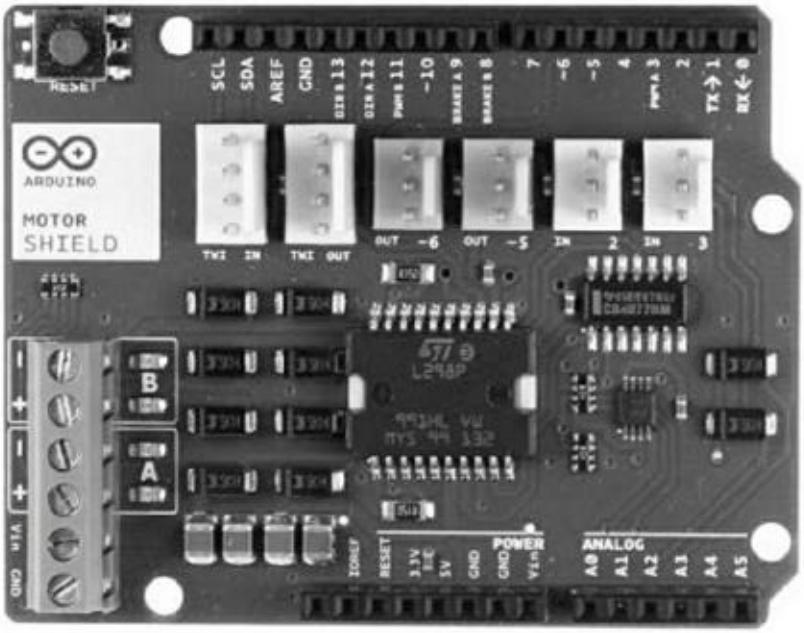


Figure 8.59 : Bouclier Arduino Motor.

Servomoteurs et pilotage PWM

Les modèles réduits radiocommandés d'avions, de voitures et de bateaux sont contrôlés grâce à de petits servomoteurs. Leur principe est de contrôler l'angle de rotation du moteur grâce à des impulsions de largeur variable, mais à fréquence fixe. La largeur d'une impulsion détermine l'angle de rotation (la durée à l'état haut). Le codage d'une sortie numérique par modulation de la largeur d'impulsion (en anglais *Pulse Width Modulation*, PWM) permet également de piloter un moteur à courant continu, de faire varier la luminosité d'une diode LED ou de contrôler un actionneur linéaire.

Bouclier 16 canaux 12 bits PWM

Ce bouclier représenté en Figure 8.60 est basé sur un circuit intégré de contrôle PWM PCA9685 à 16 canaux avec interface I2C. Il soulage le microcontrôleur de la carte Arduino en étant capable de gérer de façon autonome la génération d'un signal PWM différent sur chaque sortie.

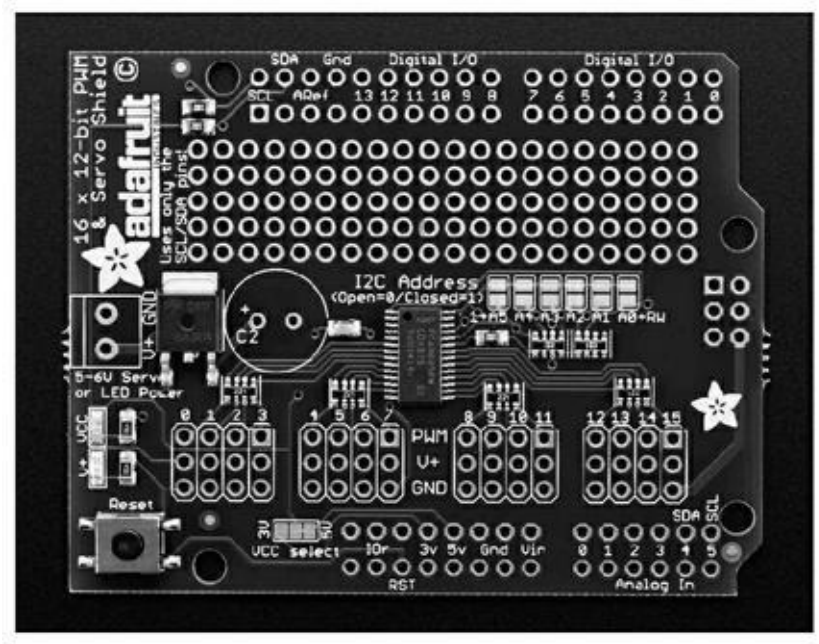


Figure 8.60 : Bouclier Adafruit PWM/servo à interface I2C.

Bouclier LinkSprite 27 canaux PWM

Ce bouclier de LinkSprite (Figure 8.61) génère jusqu'à 27 sorties PWM grâce à un circuit microcontrôleur STM32F103C8T6. Il s'agit d'un véritable processeur RISC sur 32 bits Cortex-M3 d'ARM qui possède 128 ko de mémoire flash et 20 ko de mémoire statique SRAM. À vrai dire, ce microcontrôleur est même plus puissant que celui de la carte Arduino. Il dialogue avec elle *via* l'interface SPI.

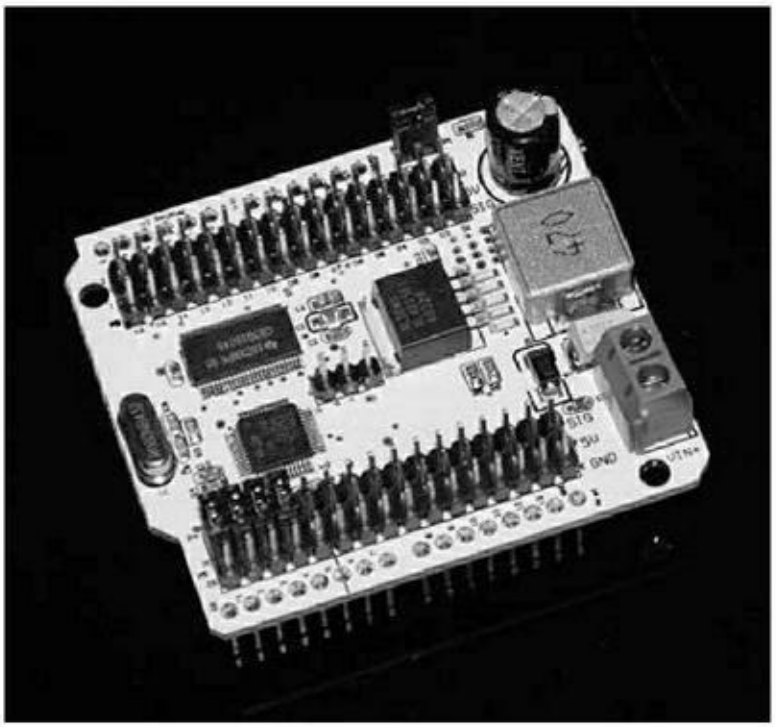


Figure 8.61 : Bouclier LinkSprite 27 canaux PWM.

Bouclier SparkFun PWM

Ce bouclier (Figure 8.62) se base sur un circuit TLC5940 qui peut gérer jusqu'à 16 sorties PWM et contrôler des servomoteurs ou des diodes LED. Il utilise une interface série SPI avec horloge mais ne sait que recevoir des données. Voyez le site de SparkFun pour tout détail au sujet des librairies.

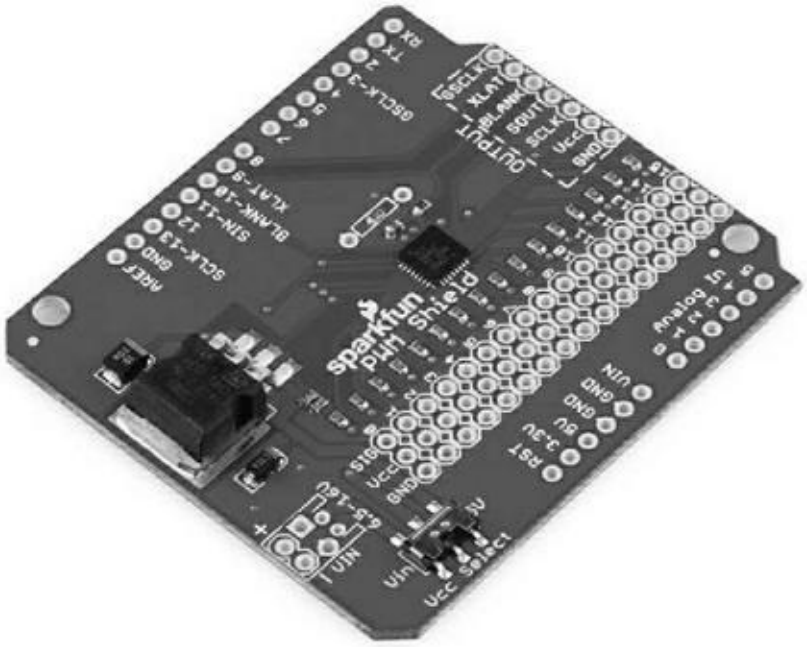


Figure 8.62 : Bouclier SparkFun PWM.

Boucliers d’affichage

Les boucliers d’affichage pour Arduino utilisent différentes techniques : des diodes LED sous forme de segments ou de matrices, des écrans à cristaux liquides LCD ou des écrans couleur graphiques TFT. Certains afficheurs ont besoin de nombreuses sorties numériques de la carte Arduino alors que d’autres utilisent une interface SPI ou I2C (TWI). Quels que soient vos besoins d’affichage, vous trouverez un bouclier qui va répondre à vos besoins.

Matrices de diodes LED

Une diode LED isolée est très utile, mais en regroupant toute une série de diodes LED sur deux lignes et deux colonnes, de véritables affichages défilants deviennent possibles. Pour tout détail au sujet de ces boucliers, vous irez sur le site Web correspondant.

Bouclier Adafruit LoL

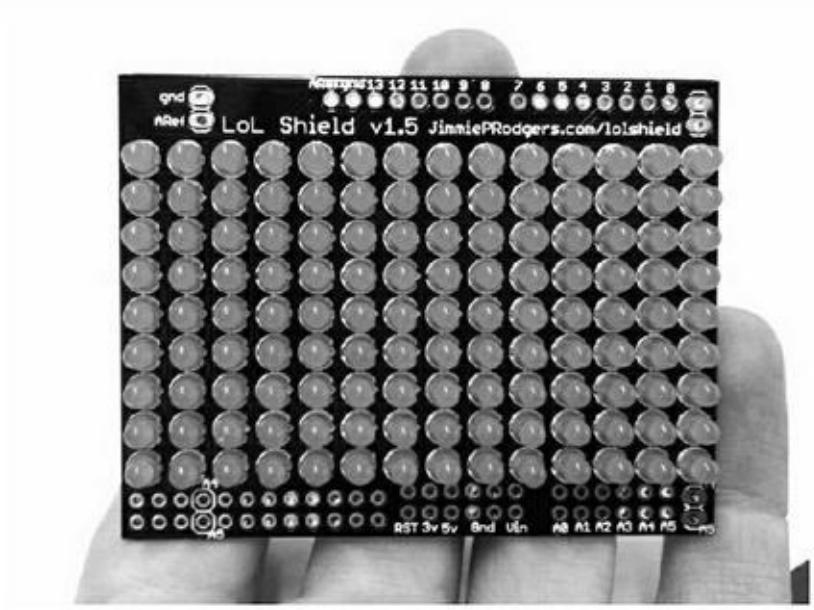


Figure 8.63 : Bouclier Adafruit en matrice de 9 x 14 diodes LED.

Bouclier SolarBotics SMB LoL

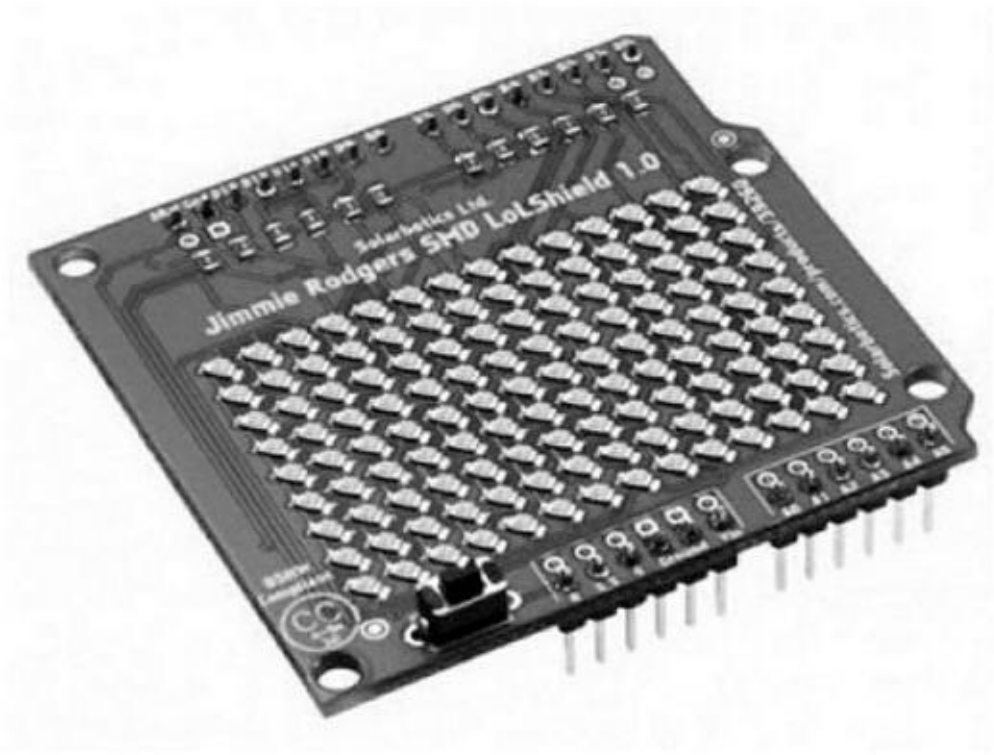


Figure 8.64 : Bouclier de 9 x 14 diodes LED SolarBotics SMB.

Bouclier Adafruit NeoPixel

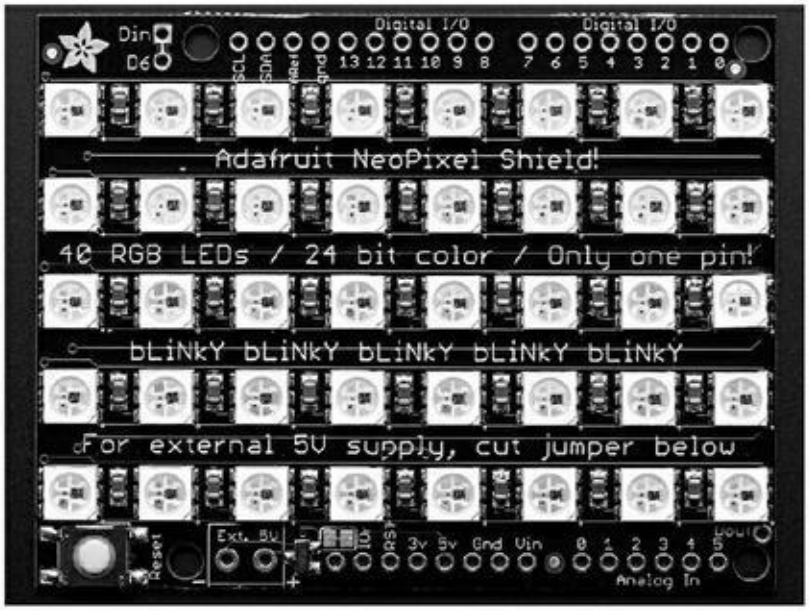
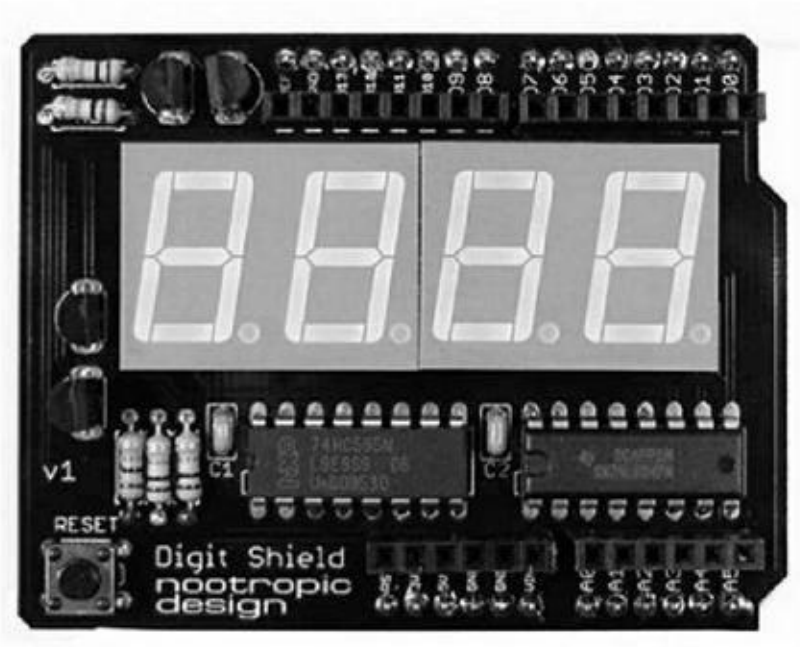


Figure 8.65 : Matrice Adafruit NeoPixels avec 40 diodes LED tricolores.

Afficheurs LED à 7 segments

Les afficheurs à 7 segments existent depuis l'invention de la diode LED. De nos jours, ils constituent une solution considérée comme minimaliste, mais restent indispensables lorsqu'il s'agit d'afficher des données qui doivent être visibles de loin. Voyez aussi le bouclier multifonction de la fin de ce chapitre qui comporte un afficheur à diodes LED sur quatre chiffres.

Bouclier Nootropic compatible digital de 4 x 7 segments



8.66 : Bouclier d’affichage 7 segments Nootropic.

Afficheurs à cristaux liquides LCD

Les boucliers d’affichage les plus abordables utilisent quasiment tous un module afficheur en mode texte de 2 lignes de 16 caractères. En général, l’affichage se fait en lettres blanches sur fond bleu, mais vous pouvez également choisir des lettres rouges sur fond noir ou noires sur fond vert, ainsi que d’autres combinaisons moins utilisées. Il existe des variantes offrant quatre lignes de 16 ou de 20 caractères. La plupart de ces boucliers sont animés par le circuit intégré de contrôle d’affichage HD 44780 de Hitachi, ou un circuit compatible.

Il existe aussi des afficheurs LCD en mode graphique, notamment le très populaire Nokia 5110. Vous n’êtes ainsi plus limité à l’affichage de deux ou quatre lignes de caractères prédéfinis, et chaque point élémentaire de l’écran est adressable (chaque pixel). Vous trouverez ainsi des afficheurs offrant une résolution de 128 sur 64 ou 160 sur 128, mais rares sont les boucliers prêts à l’emploi. Vous trouverez une sélection de modules d’affichage LCD graphique dans le Chapitre 9.

Bouclier SainSmart LCD Keypad

Ce bouclier d'affichage très classique (Figure 8.67) offre 2 lignes de 16 caractères. Il est piloté par le contrôleur Hitachi HT44780. Le module d'affichage est d'ailleurs disponible séparément, et je l'ai utilisé dans le générateur de signal du Chapitre 11 et dans le thermostat du Chapitre 12.

Figure 8.67 : Bouclier d'affichage SainSmart LCD 2 x 16.

Ce bouclier comporte une astuce permettant de n'utiliser qu'une entrée analogique pour les cinq boutons poussoirs : tous les boutons sont reliés à la même ligne avec un diviseur de tension construit avec des résistances. Il suffit à l'entrée analogique d'être capable de détecter cinq tensions différentes (Figure 8.68).

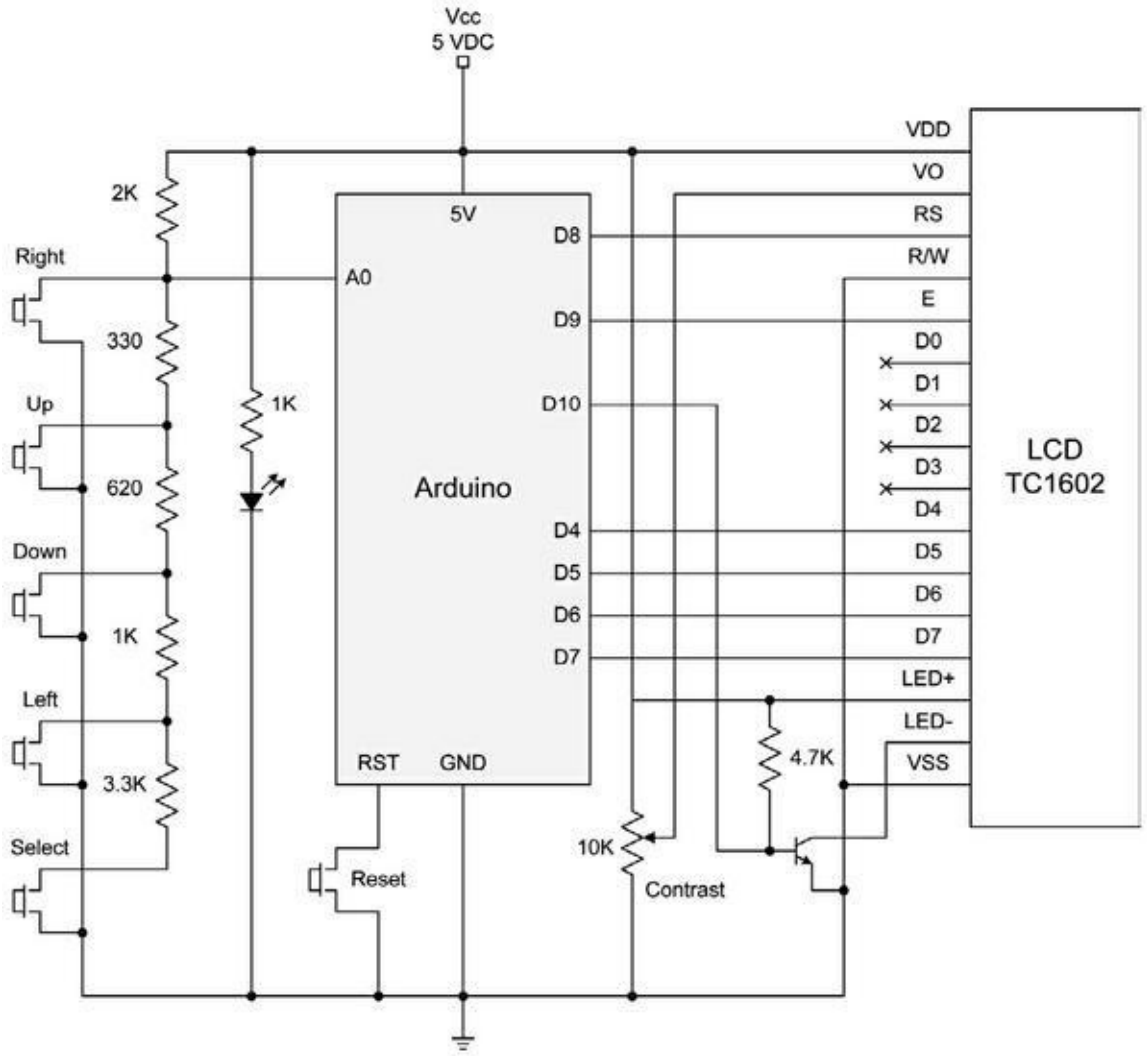


Figure 8.68 : Schéma de principe du bouclier SainSmart LCD.

Bouclier DFRobot LCD Keypad

Ce bouclier ressemble beaucoup au précédent sauf qu'il n'utilise pas l'astuce pour regrouper les boutons sur une seule entrée analogique (Figure 8.69). D'après la documentation du fabricant, les broches libres sont d'abord prévues pour implanter une interface avec un module radio APC220 ou un module sans fil Bluetooth.

Figure 8.69 : Bouclier DFRobot LCD Keypad avec broches analogiques.

Bouclier en kit Adafruit LCD

Le bouclier LCD Adafruit, que la Figure 8.70 montre une fois monté, exploite l'interface I2C avec un circuit MCP23017 (également utilisé dans le Chapitre 10). Le bouclier n'occupe que deux broches analogiques de l'Arduino, les broches A4 et A5, pour l'interface I2C. Tous les boutons ainsi que l'afficheur sont reliés au circuit MCP et il n'y a pas de diviseur de tension. Je vous rassure en précisant qu'il n'est pas difficile à assembler.

Figure 8.70 : Bouclier afficheur Adafruit LCD de 2 lignes de 16 caractères.

Module LCD Nokia 5110 avec lecteur SD

Ce bouclier réunit un afficheur LCD Nokia 5110 et un lecteur de carte SD (Figure 8.71). L'afficheur utilise les broches numériques D3, D4, D5, D6 et D7. Le lecteur de carte SD utilise les broches D10 à D13. Les trois broches D0, D1 et D2 restent disponibles.

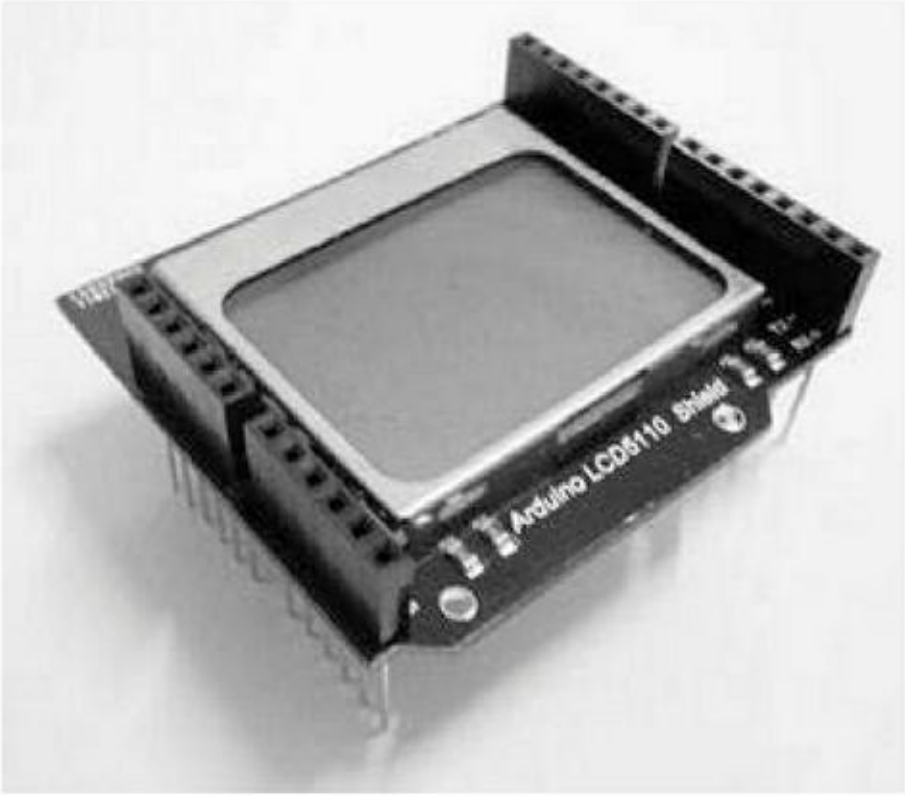


Figure 8.71 : Bouclier LCD Nokia 5110 d'Elechouse.

À la différence des afficheurs en mode texte précédents, ce bouclier est graphique et offre une surface de 48 sur 84 pixels. Comme son nom le laisse deviner, il s'agit au départ de l'écran des téléphones Nokia du milieu des années 2000. Tous les afficheurs vendus actuellement sont des surplus, et certains ont de petits défauts de surface. Lorsque le stock sera épuisé, vous ne pourrez plus en acquérir. Il est donc déconseillé d'opter pour cet afficheur dans le cadre d'un projet à long terme. En revanche, il est très intéressant à utiliser pour faire des essais et il est très peu coûteux.

Afficheurs couleur TFT

La technologie TFT (*Thin Film Transistor*), ou transistor à film mince, est très utilisée pour les écrans des ordinateurs, les afficheurs des caisses enregistreuses, les téléphones, les tablettes, et quasiment tout ce qui a besoin d'un affichage graphique couleur. Les afficheurs TFT pour Arduino permettent en général d'afficher des milliers de couleurs avec par exemple une résolution de 240 sur 320 pixels. Il existe des afficheurs de plus grande taille, mais ils ne tiennent plus dans les dimensions d'un bouclier standard. Le prix n'est en général pas prohibitif. La plupart utilisent une interface SPI et certains y ajoutent une interface numérique parallèle pour un affichage d'images à grande vitesse.

Bouclier ITEAD 2.4"LCDTFTTouch

Ce bouclier exploite une interface parallèle avec la carte Arduino (Figure 8.72). Le contrôleur de l'écran est un circuit S6D1121 qui travaille avec une interface 8 bits. Les fonctions de l'écran tactile sont gérées par un circuit TSC2046. Voyez le site du fabricant pour d'autres détails.

Figure 8.72 : Bouclier ITEAD 2.4" LCD TFT Touch.

Écran tactile Adafruit 2.8" TFT Touch

Cet assez grand écran, puisqu'il recouvre la totalité du bouclier (Figure 8.73) utilise une interface SPI à haute vitesse pour alimenter le contrôleur d'affichage ILI9341 qui dispose d'une zone mémoire RAM pour la vidéo, et un contrôleur de surface tactile STMPE610. Il dispose même d'un lecteur de carte microSD et exploite les broches numériques D8 à D13 ainsi que la broche D4 pour le signal de sélection du lecteur de carte microSD.

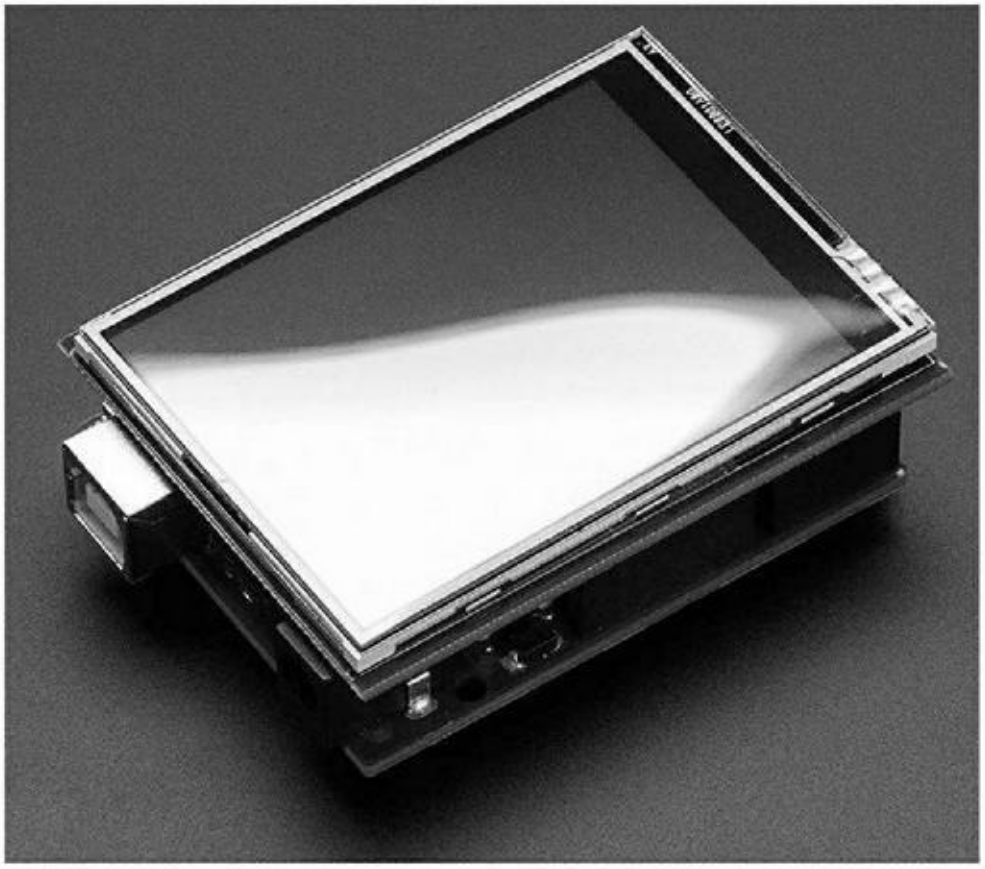


Figure 8.73 : Bouclier Adafruit 2.8" TFT avec écran tactile résistif.

Boucliers d'instrumentation

Cette catégorie est moins fournie que les autres, mais il existe quelques boucliers pour l'instrumentation, et notamment pour la journalisation de données, l'analyse logique et la conversion analogique-numérique de précision. Le terme d'instrumentation fait ici référence à la capacité de collecter des données captées dans le monde physique ou celle de générer des signaux analogiques en direction du monde physique.

Vous trouverez peu de boucliers proposant la capture de données et la conversion vers le numérique tout simplement parce que le microcontrôleur de la carte Arduino propose déjà ces fonctions. Il contient en effet un convertisseur analogique-numérique, mais sa résolution n'est que de 10 bits, soit 1024 valeurs différentes. Lorsque vous avez besoin d'une meilleure résolution, sur 12, 16 ou même 24 bits, il vous faut un module complémentaire ou un bouclier.

Bouclier Adafruit Data Logging

Ce bouclier, représenté en Figure 8.74, embarque un lecteur de carte SD et une horloge temps réel RTC qui peut être alimentée par une pile pour préserver l'heure. Le bouclier comporte une zone de prototypage, et il n'est pas empilable.

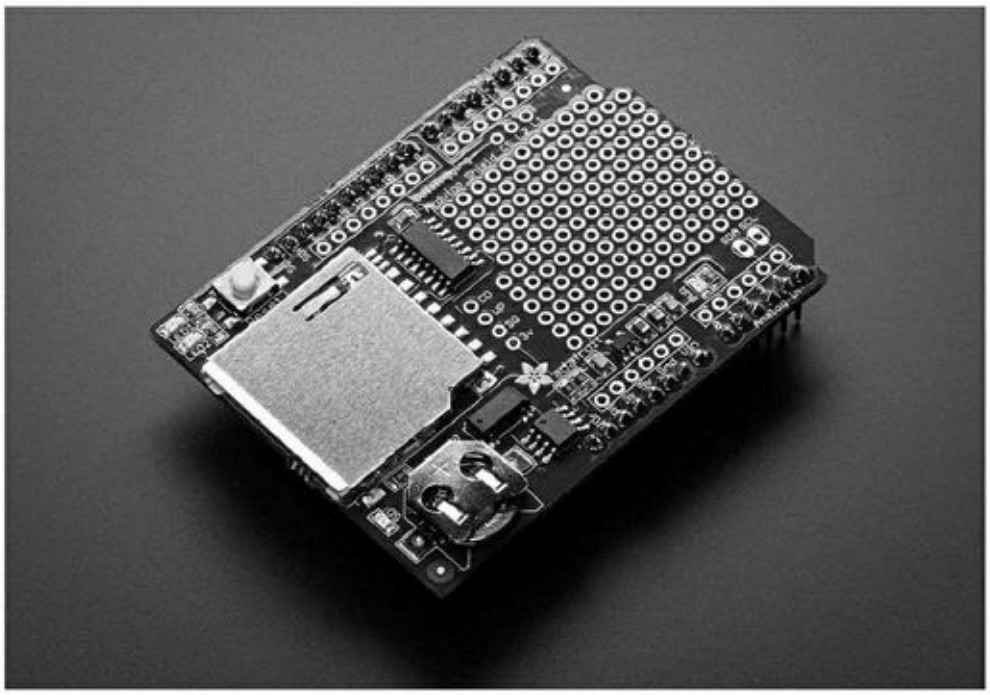


Figure 8.74 : Bouclier Adafruit Data Logging une fois monté.

Bouclier Adafruit Ultimate GPS Logger

Ce bouclier ajoute au précédent un récepteur GPS (Figure 8.75). Les coordonnées générées par le circuit GPS peuvent être automatiquement stockées sur la carte mémoire flash. Le bouclier n'est pas empilable.

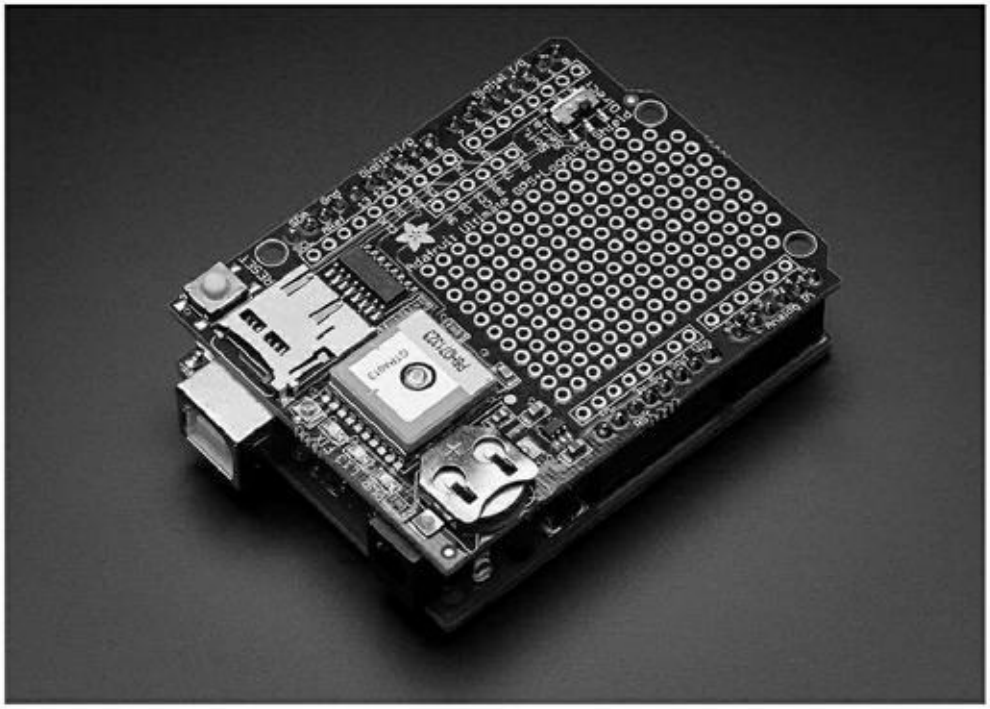


Figure 8.75 : Bouclier Adafruit Ultimate GPS Logger.

Bouclier Hobbylab Logic Analyzer and Signal Generator

Il s'agit en réalité d'un analyseur logique autonome installé sur un bouclier. Il est capable de surveiller tous les signaux de la carte Arduino sans les perturber, ce qui permet de savoir exactement ce qui se passe sur les broches. Il embarque également un décodeur d'interface SPI, un décodeur UART et un moniteur sur un fil. Il ne communique pas directement avec la carte Arduino. Il est prévu pour dialoguer avec une machine hôte grâce à son interface USB. Le bouclier est empilable (Figure 8.76).

Figure 8.76 : Hobbylab Logic Analyzer and Signal Generator.

Bouclier Iowa Scaled Engineering 16 canaux 24 bits

Ce bouclier embarque un convertisseur analogique-numérique sur 24 bits ainsi qu'une référence de tension très précise. Il est capable de capturer des valeurs sur des entrées asymétriques ou différentielles. Il dispose d'une mémoire EEPROM pour lire et écrire les données d'étalonnage et de configuration. Le bouclier utilise l'interface I2C pour dialoguer avec la carte Arduino. Il est empilable (Figure 8.77).

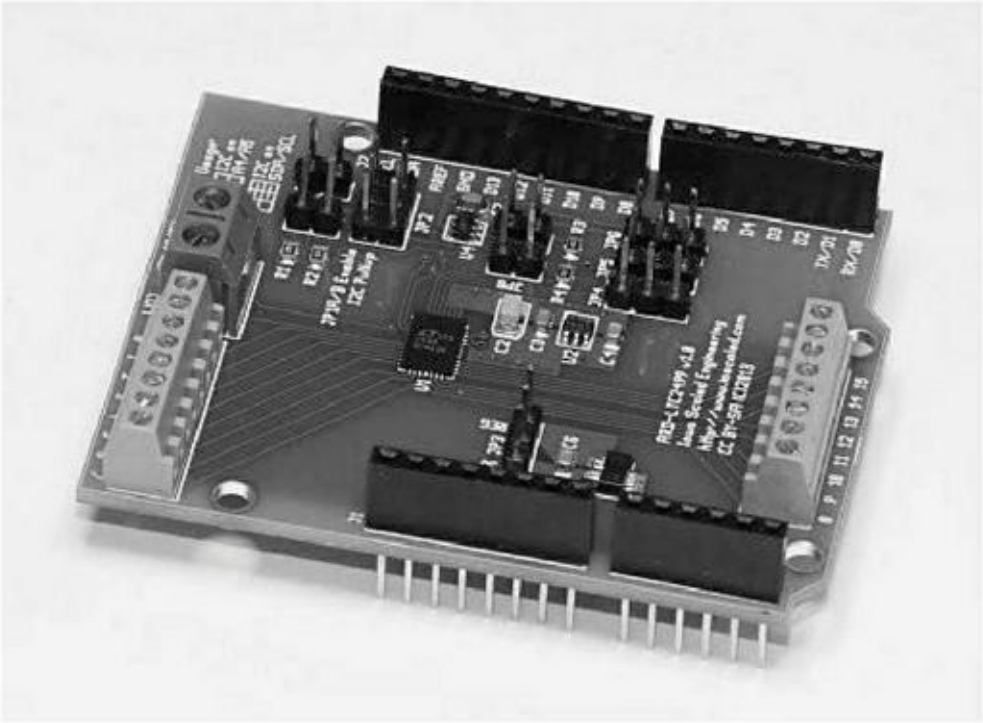


Figure 8.77 : Bouclier d’acquisition de données Iowa Scaled Engineering 24 bits.

Bouclier Visgence Power DAC

Une fonction indisponible dans la carte Arduino, et donc dans le microcontrôleur, est la conversion numérique vers analogique (CNA ou DAC). Il y a bien un convertisseur 10 bits à l’intérieur du microcontrôleur, mais il n’est pas disponible sur les broches. Vous trouverez des boucliers de conversion pour le son, mais il y a peu de boucliers de conversion vers une véritable tension analogique continue. Le bouclier Visgence Power DAC (Figure 8.78) comble ce manque en offrant trois canaux de sortie analogiques avec une capacité de 250 mA de courant.

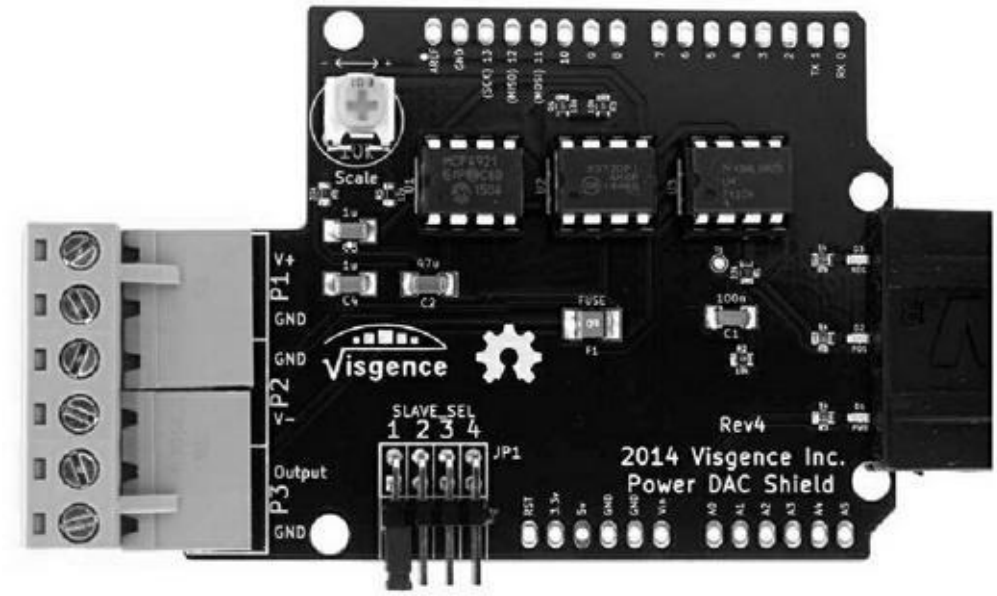


Figure 8.78 : Bouclier Visgence Power DAC.

Boucliers d’adaptation

Un bouclier d’adaptation est une interface physique entre deux modules qui ne sont pas compatibles. Il se distingue d’un bouclier de routage de signal (vu en début de chapitre) par le fait que le bouclier d’adaptation est une interface physique alors que le routeur de signal ne s’intéresse qu’à l’adaptation électrique et/ou logique des signaux électriques.

Tronixlabs Australia Expansion Shield pour Nano

Le modèle Arduino Nano possède les mêmes capacités que les cartes Arduino standard, mais son format ne lui permet pas d’accepter les boucliers standard. Le bouclier de la Figure 8.79 résout ce problème en dirigeant les broches de la carte Nano vers des connecteurs mâles et femelles.

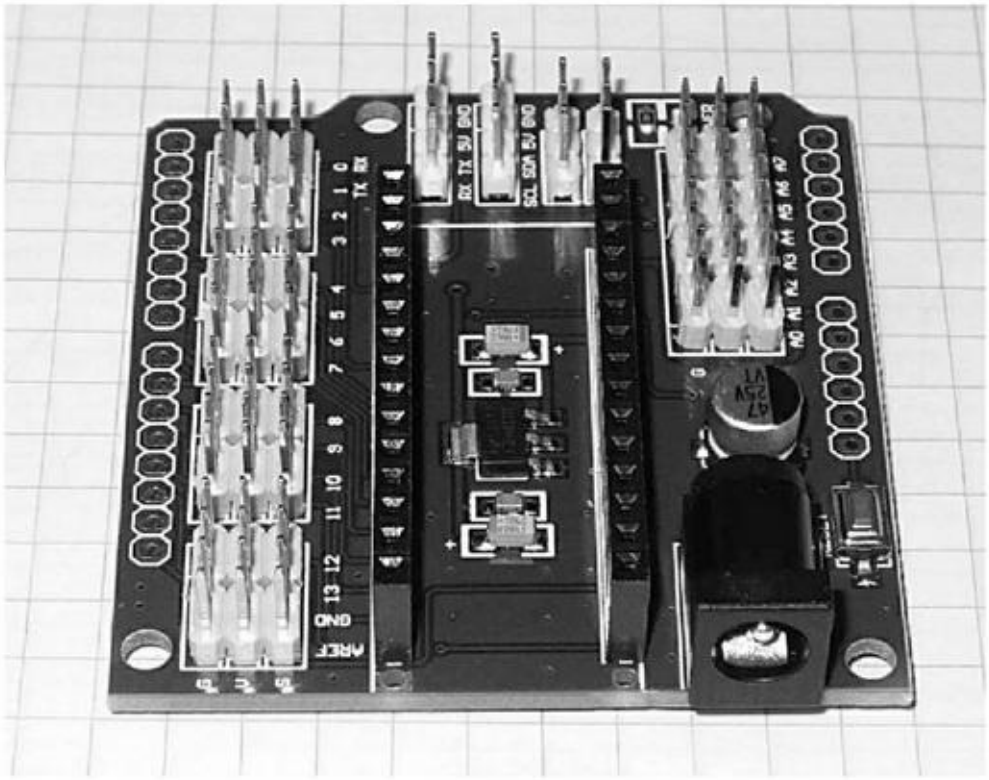


Figure 8.79 : Bouclier d’adaptation Tronixlabs Nano.

Arduino Nano I/OExpansion

J’ai trouvé cet autre bouclier d’adaptation pour Nano (Figure 8.80) lors de mes visites du site de commerce eBay. Plusieurs fournisseurs en proposent.

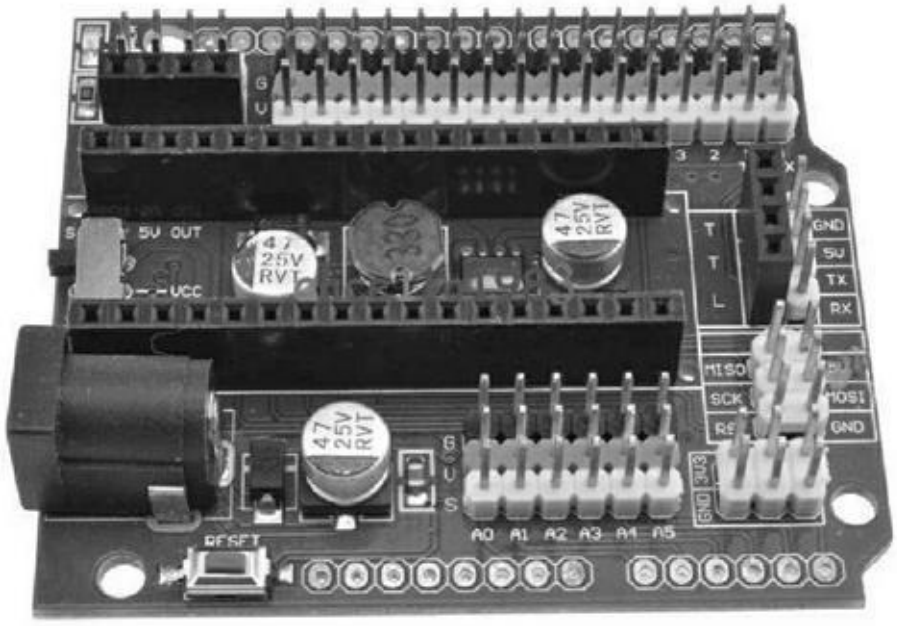


Figure 8.80 : Bouclier d’adaptation pour Arduino Nano.

Barrette de borniers à vis

Il ne s’agit pas exactement d’un bouclier, mais cette barrette de borniers permet de raccorder des fils de grand diamètre à votre carte Arduino. Vous trouverez de nombreux fournisseurs qui proposent ce genre d’adaptateur. Remarquez que le format n’empêche pas l’ajout d’un autre bouclier (Figure 8.81).

Figure 8.81 : Barrettes de borniers à vis.

Autres boucliers

Les boucliers de cette section n'entrent dans aucune des catégories précédentes. J'apprécie particulièrement le bouclier Wingshield qui permet de réaliser des connexions très soignées, ainsi que le bouclier multifonction qui porte vraiment bien son nom.

Adafruit Proto-Screwshield (Wingshield)

Voici ce fameux bouclier Wingshield, également appelé Proto-ScrewShield. Il tire son nom (« bouclier ailé ») du fait qu'il comporte deux rangées de borniers qui débordent de la largeur du format standard (Figure 8.82). Ce genre de bouclier trouvera facilement à s'employer dans un produit destiné à être vendu ou pour un laboratoire. Les connexions par borniers sont en effet bien plus fiables que les broches à enficher. La zone de prototypage en milieu de bouclier permet d'ajouter un module capteur ou un circuit spécifique. Le bouclier est empilable et dispose d'un interrupteur de Reset et d'une diode LED. Notez cependant qu'il est livré en kit. Prévoyez de faire chauffer le fer pour installer les pièces détachées sur le circuit nu.

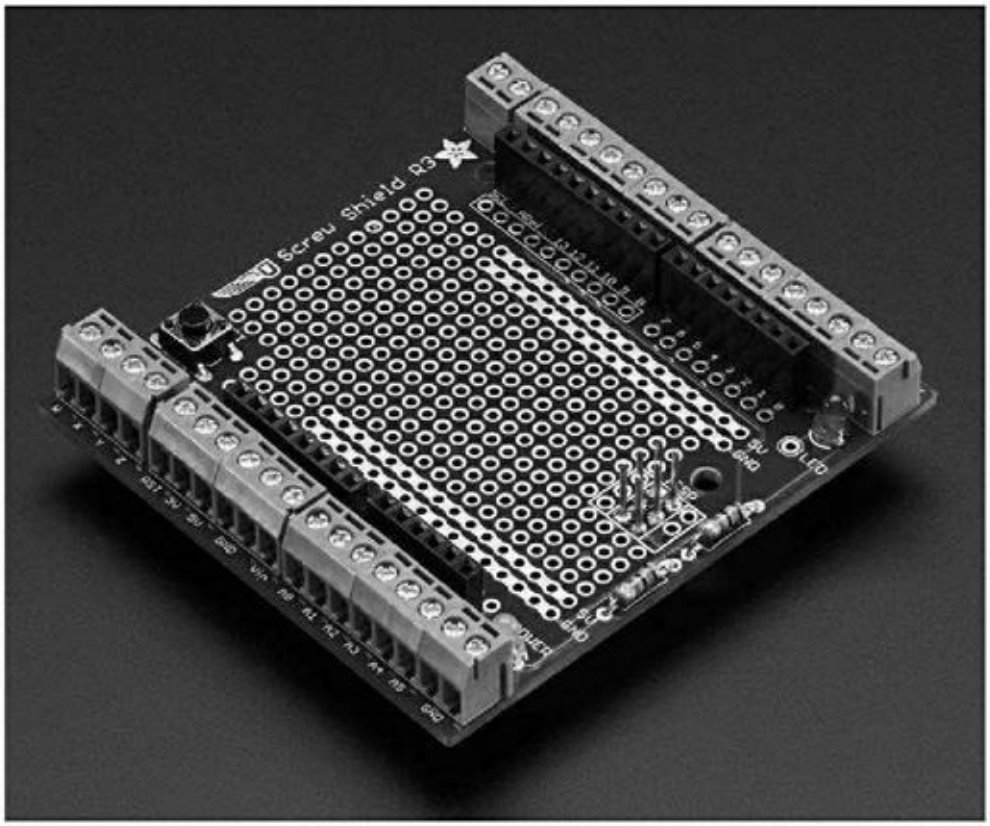


Figure 8.82 : Bouclier de connexion Adafruit Proto-ScrewShield.

DFRobot Screw ProtoShield

Ce bouclier (Figure 8.83) ressemble beaucoup au précédent avec des borniers pour chacun des signaux de la carte Arduino. La différence est qu'il est déjà monté. Il est empilable, ce qui permet d'ajouter un autre bouclier par-dessus, à condition que ce que vous avez installé sur la partie de prototypage ne dépasse pas trop. Les borniers restent accessibles dans tous les cas.

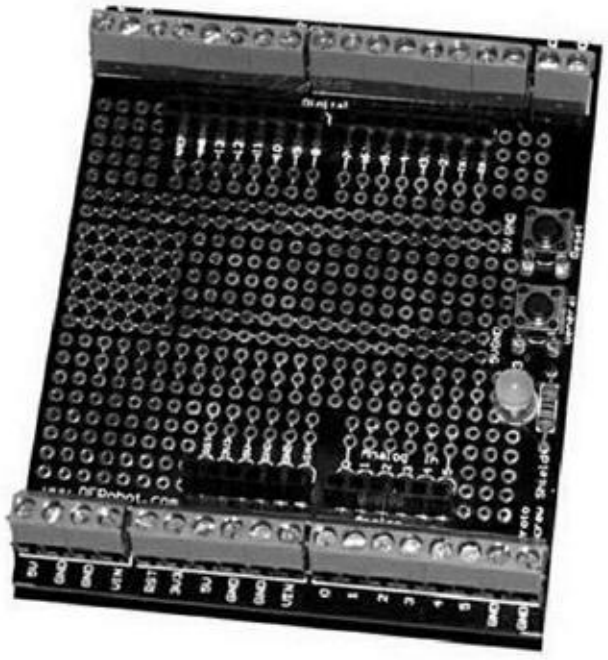


Figure 8.83 : Bouclier de connexion DFRobot.

Bouclier DealXtreme DIY multifonction

Ce bouclier est très polyvalent : il réunit un afficheur à diodes LED sur quatre chiffres, une interface pour un module Bluetooth APC220, plusieurs boutons reliés aux broches analogiques A1, A2 et A3, un bouton Reset et un potentiomètre relié à la broche A0 (Figure 8.84). Il est doté enfin de quatre diodes LED reliées aux broches D10 à D13 et de trois connecteurs à quatre broches pour les broches D5, D6, D9 et A5 ainsi que le +5 V et la masse. La documentation de ce bouclier n'est hélas pas suffisante. Vous trouverez quelques croquis et un schéma à l'adresse `bit.ly/dx-diy-sketch`. Les noms des sous-répertoires dans l'archive ZIP sont tous en chinois, mais certains croquis comportent des commentaires en anglais.

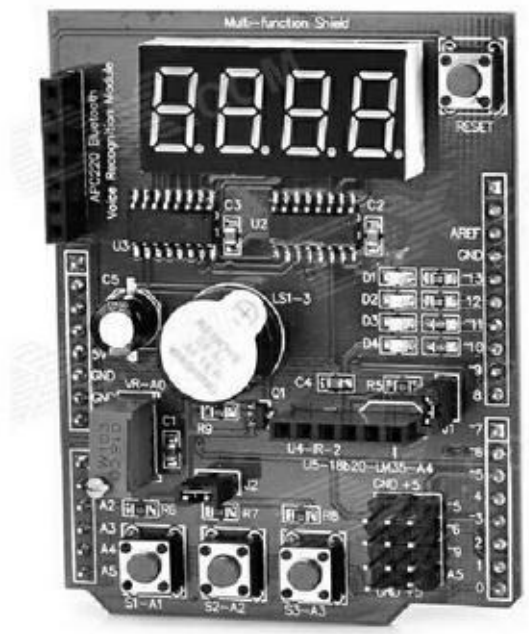


Figure 8.84 : Bouclier d’expansion multifonction DealXtreme.

La Figure 8.85 montre l’implantation des composants sur le bouclier. Bien sûr, il n’est pas empilable : vous ne pourriez plus voir l’affichage si vous placiez un autre bouclier par-dessus.

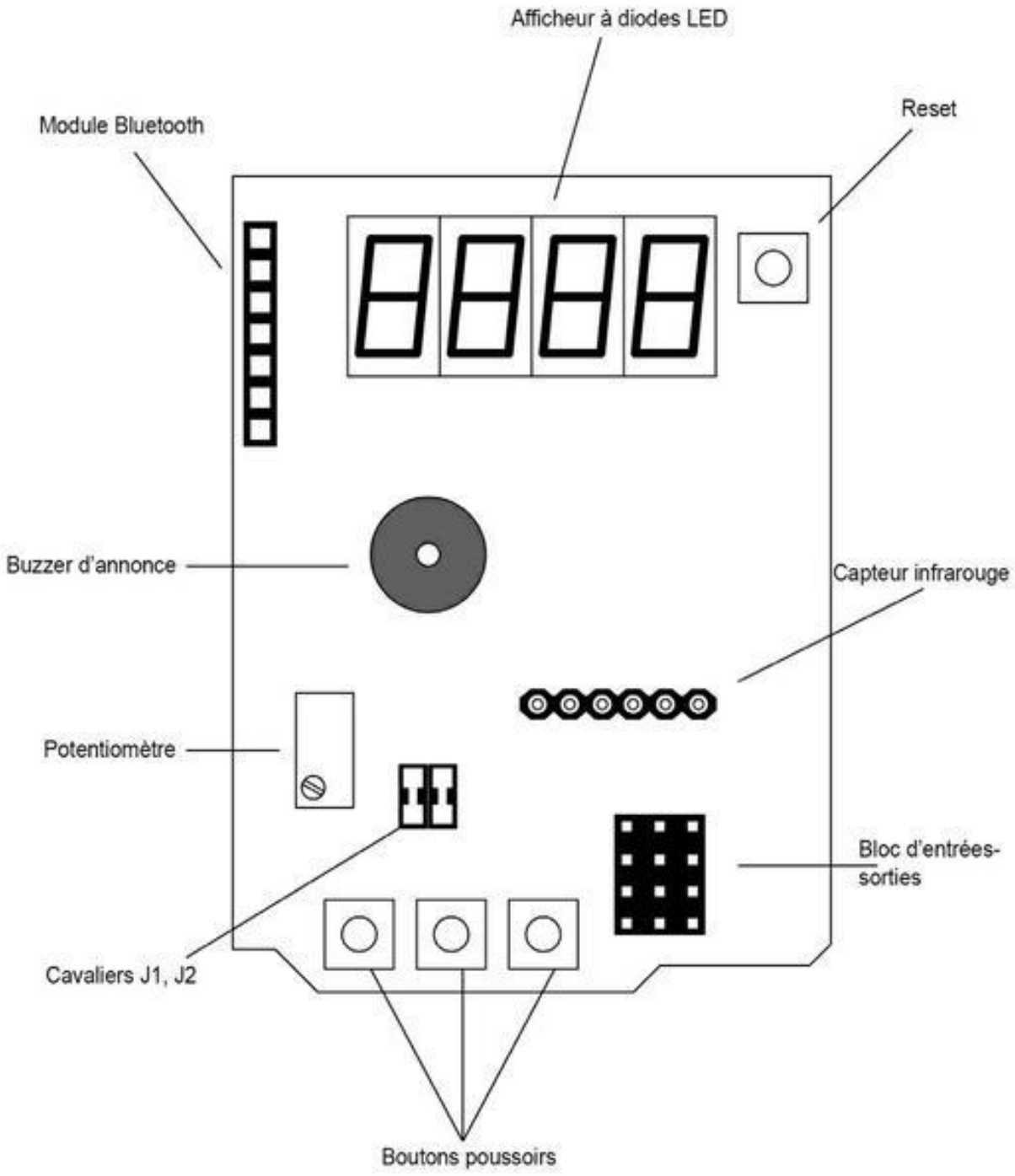


Figure 8.85 : Implantation des composants principaux du bouclier multifonction.

Le schéma de principe disponible est difficile à exploiter. J'ai donc décidé de créer ma propre version, visible dans la Figure 8.86. Toutes les broches de la carte Arduino sont occupées par le bouclier. Le Tableau 8.1 montre leur affectation.

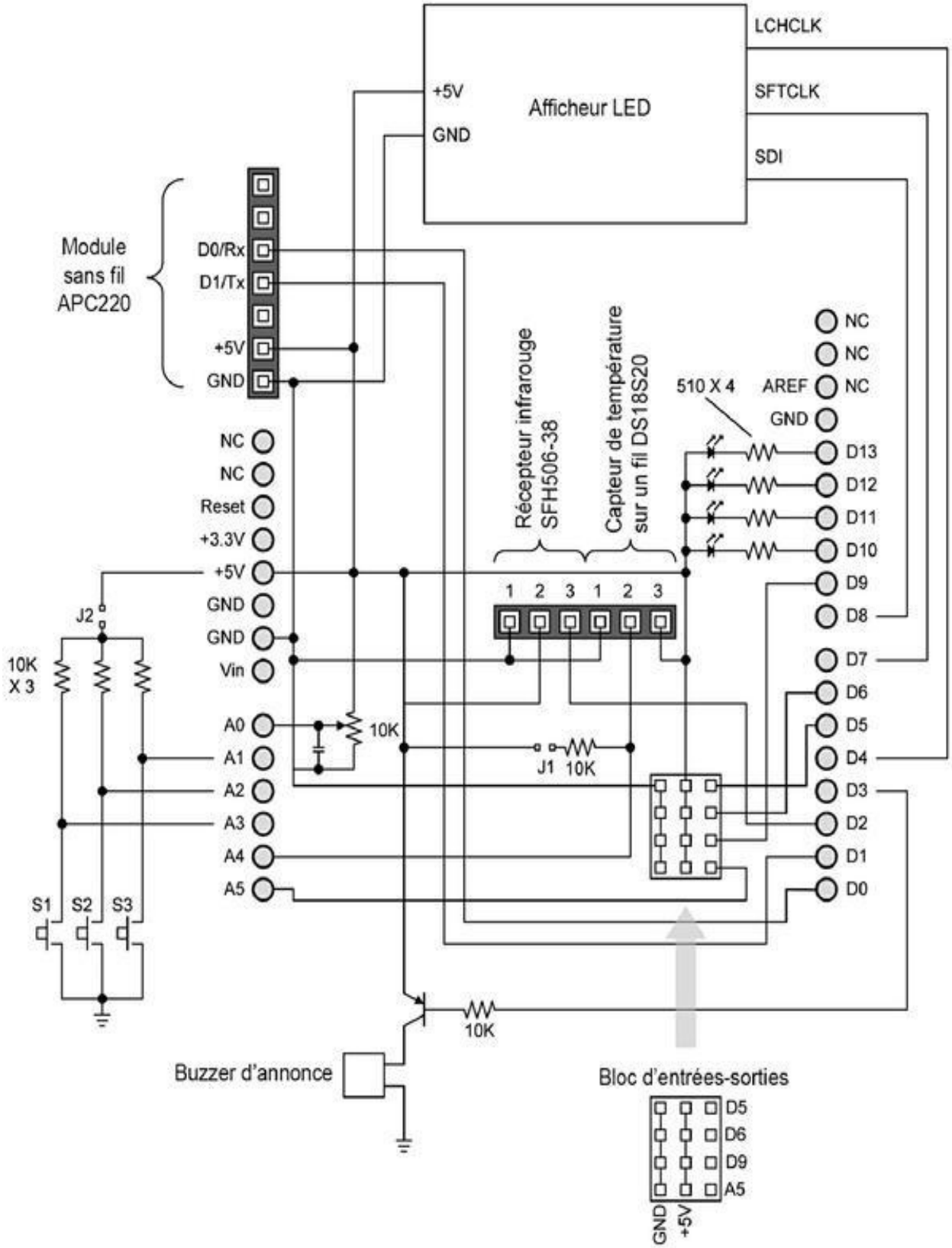


Figure 8.86 : Schéma de principe détaillé du bouclier multifonction.

Tableau 8.1 : Affectation des broches Arduino du bouclier multifonction.

Broche	Affectation	Broche	Affectation
D0	Rx du module sans fil	D10	LED
D1	Tx vers le module sans fil	D11	LED
D2	ID entrée réception	D12	LED
D3	Contrôle buzzer	D13	LED
D4	Verrou LED	A0	Axe potentiomètre
D5	Vers D5 sur bloc I/O	A1	Interrupteur S3
D6	Vers D6 sur bloc I/O	A2	Interrupteur S2
D7	Horloge affichage LED	A3	Interrupteur S1
D8	Entrée série LED	A4	Entrée capteur température
D9	Vers D9 sur bloc I/O	A5	A5 sur bloc I/O

Ce bouclier illustre ce qui se passe lorsqu'un nouveau bouclier est commercialisé. La documentation est réduite au strict minimum, et souvent dans une langue moins répandue dans nos contrées telles que le chinois, ou parfois l'anglais. Le schéma est correct, mais pas facile à comprendre, et il n'y a aucune description détaillée des broches. Ces lacunes sont dorénavant réparées grâce à ce livre.

Boucliers Arduino spéciaux

Tous les boucliers vus jusqu'ici répondent à de très vastes besoins, de la liaison série RS-232 au contrôle de servomoteurs, mais il existe aussi des boucliers à usage très particulier. On peut par exemple contrôler une imprimante 3D avec une carte Arduino. Il existe des boucliers pour réaliser l'interface de contrôle. Certains des boucliers sont prévus pour le format de carte Mega.

Un autre bouclier qui n'entre dans aucune autre catégorie est le Gameduino. C'est une carte qui embarque un circuit intégré programmable appelé FPGA. Son domaine d'utilisation n'est pas limité aux jeux vidéo.

Qunqi CNCShield for Arduino V3 Engraver

Ce bouclier de pilotage de graveur (Figure 8.87) n'est pas livré avec les modules de pilotage des moteurs, mais ils sont faciles à trouver. En général, ces modules utilisent le circuit intégré de pilotage par micro-pas Allegro A4988 DMOS.

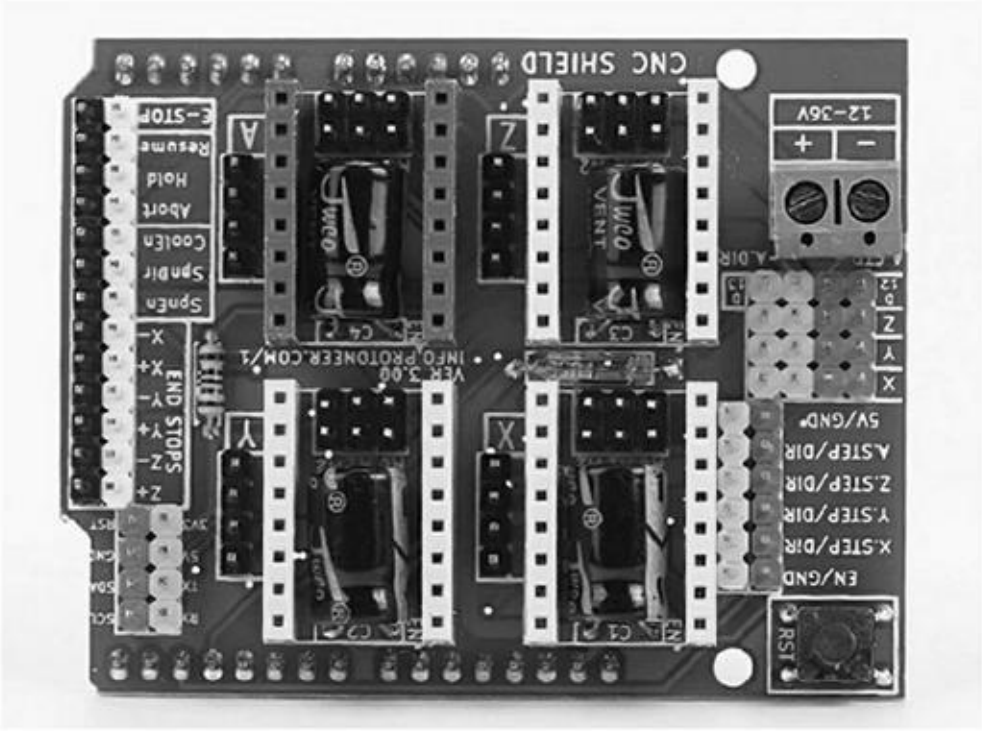


Figure 8.87 : Carte Qunqi CNC Shield for Arduino V3 Engraver.

SainSmart RepRap Arduino Mega Pololu

La RepRap 3D est une exprimante (une imprimante 3D) que ses concepteurs annoncent comme la première machine capable de se répliquer elle-même. Cette exprimante permet en effet de fabriquer des pièces pour d'autres machines RepRap, entre autres objets. Le bouclier montré dans la Figure 8.88 est destiné à remplacer l'électronique de commande d'une telle machine. Il est conçu pour fonctionner avec une carte Arduino de type Mega.

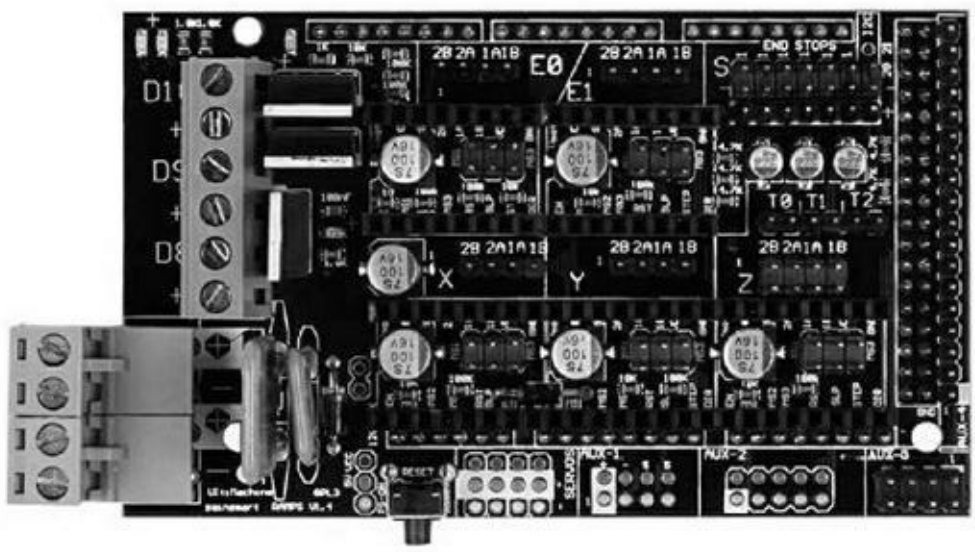


Figure 8.88 : Bouclier de contrôle d'exprimante RepRap de SainSmart.

excamera Gameduino

Ce bouclier embarque un réseau de portes logiques reprogrammable FPGA (*Field Programmable Gate Array*) de XiLinx (Figure 8.89). Son but premier est de contrôler l'affichage graphique et le son d'une console de jeu sur mesure. La carte Arduino se charge du contrôle du jeu et de l'interface. Les logiciels Gameduino sont de type open source et tous les détails techniques sont disponibles. Vous pouvez facilement réutiliser le circuit FPGA pour autre chose qu'un jeu.

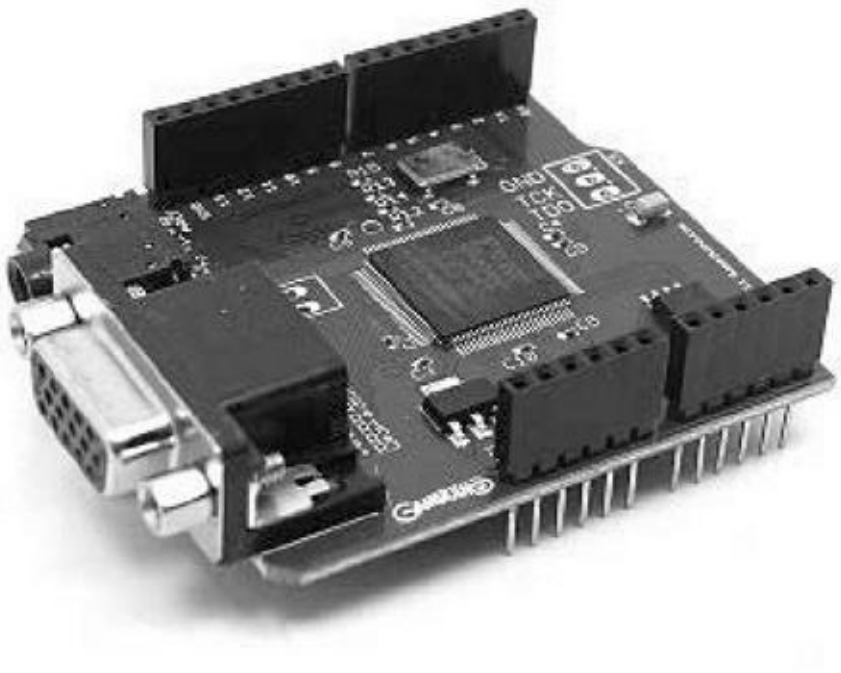


Figure 8.89 : Bouclier de contrôleur de jeu excamera Gameduino.

Sources d’approvisionnement

Le tableau suivant liste les adresses des sites Web de tous les fabricants cités dans ce chapitre. Il ne s’agit bien sûr que d’une sélection. Le simple fait de saisir l’expression « Arduino shield » dans un moteur de recherche renvoie plusieurs millions de résultats. Ce n’est pas parce qu’un fournisseur n’est pas cité dans ce chapitre qu’il ne mérite pas toute votre attention. Il est simplement impossible dans un nombre limité de pages de faire un tour d’horizon exhaustif du marché. Je rappelle que l’univers Arduino est très dynamique. Une liste, aussi exhaustive soit-elle, ne le serait plus à peine le livre sorti des presses.

Tableau 8.2 : Liste des fournisseurs de boucliers cités.

Nom	Adresse	Nom	Adresse
Adafruit	www.adafruit.com	Macetech	www.macetec
Arduino	store.arduino.cc	Mayhew Labs	www.mayhew
Arduino Lab	www.arduinolab.us	Nootropic Design	www.nootropi
Circuits@Home	www.circuitsathome.com	Numato	www.numato.
CuteDigi	store.cutedigi.com	RobotShop	www.robotsho
DFRobot	www.dfrobot.com	Rugged Circuits	www.ruggedci
DealeXtreme (DX)	www.dx.com	SainSmart	www.sainsma
Elecfreaks	www.elecfreaks.com	Seeed Studio	www.seeedstu
Elechouse	www.elechouse.com	SparkFun	www.sparkfur

excamera	www.excamera.com	Tindie	www.tindie.co
Iowa Scaled Engineering	www.iascaled.com	Tronixlabs	www.tronixlab
iMall	imall.itead.cc	Vetco	www.vetco.ne

CHAPITRE 9

Modules et composants d’entrée-sortie

Le chapitre précédent nous a montré la grande variété de boucliers disponibles. Pourtant, il n’existe pas de bouclier pour tous les besoins, et ce n’est d’ailleurs pas possible, quand on considère l’énorme variété de capteurs, d’actionneurs et de composants de contrôle utilisables avec une carte Arduino. Plusieurs fournisseurs proposent ainsi des modules, qui représentent un certain travail d’intégration de composants élémentaires. Ces modules répondent à toutes sortes de besoins : mesure de température, d’humidité, détection de vibrations ou de lumière, saisie par clavier, joystick, et même émission d’un rayon laser.

Quasiment tous les capteurs et actionneurs qui peuvent être exploités par un microcontrôleur peuvent l’être par Arduino. Il faut seulement tenir compte des tensions d’alimentation, puisque les microcontrôleurs Arduino fonctionnent la plupart en 5 V (quelques-uns en 3,3 V). En général, l’adaptation d’une tension vers l’autre est relativement simple, avec quelques composants d’interface et une alimentation appropriée.

Nous allons dans ce chapitre découvrir les modules d’entrées-sorties et les composants individuels. Un module est un petit circuit imprimé dédié à une fonction et embarquant quelques composants actifs, ou seulement des composants passifs. La taille d’un module est en général à peu près celle d’un timbre-poste. La plupart utilisent des broches mâles normalisées pour les connexions. Certains modules sont prévus pour être connectés avec des câbles multibrins standardisés au sein d’une famille de modules, et notamment ceux des fournisseurs Keyes, SainSmart et TinkerKit. Je vais d’ailleurs présenter ces trois familles de modules un peu plus en détail. Les trois autres familles de modules très répandus sont ceux de Grove, de Seeed Studio et de TinyCircuits.

N. d. T. : Il y a bien trois **e** dans le nom de la société **Seeed** Studio.

Le choix au niveau des composants d’entrées-sorties individuels est évidemment encore plus vaste : diodes LED, afficheurs graphiques, actionneurs mécaniques, interrupteurs, relais, et sondes de mesure diverses. Je présente les composants individuels après avoir présenté les modules, car la plupart des modules utilisent justement ces composants. Les références des circuits utilisés permettent de passer d’une section à l’autre.

En général, les gens aiment les situations claires et fiables. Au niveau des connexions, les cavaliers et straps pour connecter les modules et les composants ne sont pas la solution idéale, notamment parce que les cosses serties ont tendance à prendre du jeu dans les logements et à ne plus assurer un contact parfait.

Vous pouvez bien sûr décider de souder les fils directement aux modules, ou de protéger les connecteurs avec un peu de colle transparente, mais vous pouvez également adopter des connecteurs modulaires que vous pouvez vous fabriquer en fonction de vos besoins. Vous pouvez enfin utiliser un système de connexion normalisé comme ceux déjà décrits pour TinkerKit, Grove ou TinyCircuits. Je termine ce chapitre en présentant les différentes techniques possibles pour connecter des modules à une carte Arduino, en évitant d’aboutir à un plat de spaghettis de fils volants.

Présentation des modules

Les modules d’entrées-sorties constituent sans aucun doute la technique la plus confortable pour connecter un capteur, un interrupteur, un relais ou un microphone à une carte Arduino et se lancer immédiatement dans des essais. La Figure 9.1 donne un aperçu de quelques modules.

Sur le long terme, vous finirez par vous constituer une véritable collection de modules, et certains seront plus utiles que d’autres. Je vous conseillerais de démarrer avec un assortiment de modules le plus grand possible, puis de voir lesquels vous sont les plus utiles. Achetez plusieurs exemplaires de vos favoris, et gardez les autres pour des projets futurs.

Les descriptions du fonctionnement des modules que vous trouverez sur le Web ne sont pas toujours correctes. Il peut s’agir d’une erreur de traduction ; parfois, la description prétend que le module va générer un état haut lorsqu’il est actif alors que c’est le contraire. Vérifiez donc toujours le fonctionnement du module avec un multimètre numérique avant de le mettre en œuvre. La plupart des modules que je présente ont été testés et les descriptions devraient être justes. Je ne peux cependant pas garantir que tous les modules qui ressemblent à ceux que je présente auront exactement le même comportement et le même brochage. Dans l’univers Arduino, la standardisation est un processus encore inachevé.

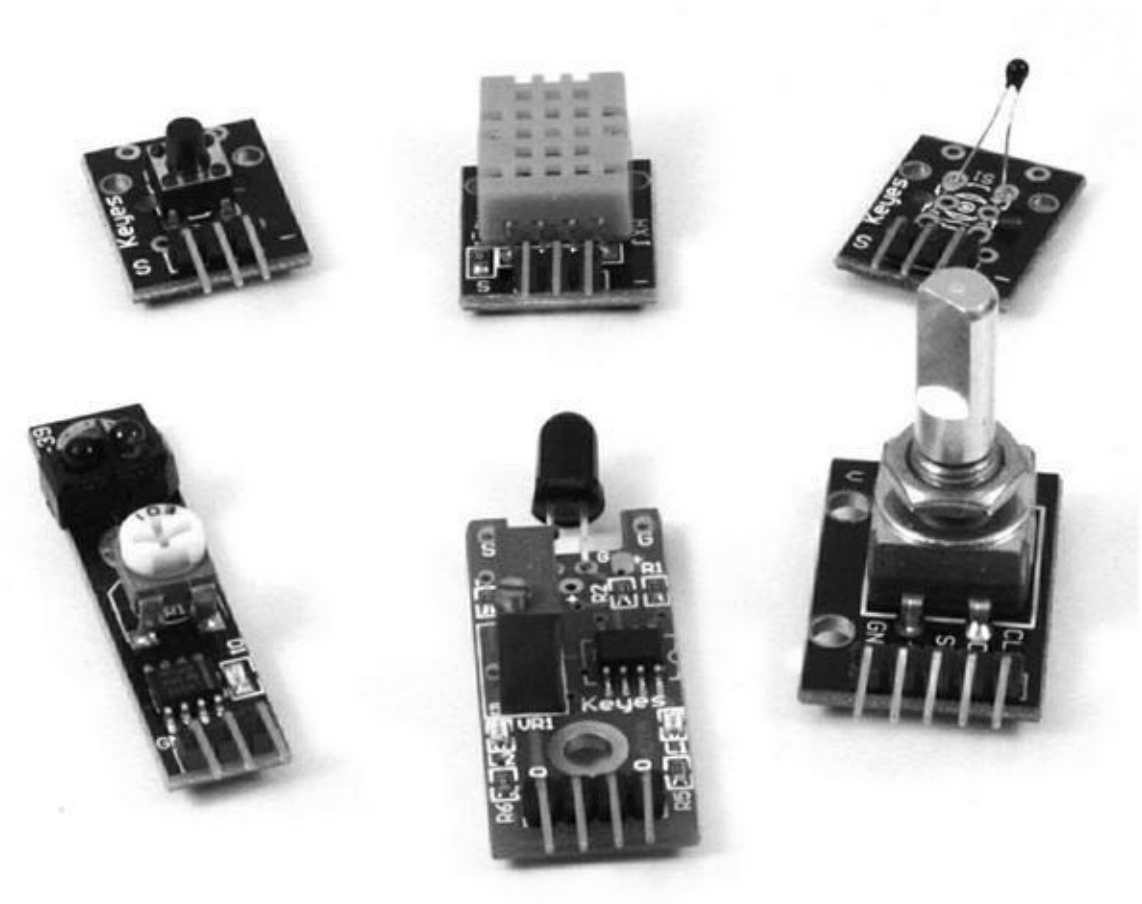


Figure 9.1 : Quelques modules capteurs de tailles variées.

Au niveau des schémas de principe pour les modules, je n'ai pas vraiment trouvé de source officielle. Des volontaires ont tenté d'en réaliser quelques-uns et de les proposer sur Internet. J'ai tenté de réunir tout ce que je pouvais trouver, et d'ajouter mon grain de sel en faisant un peu de rétro-ingénierie. Parfois, le résultat est un schéma trop fouillé. Parfois, je me suis contenté de vérifier que les broches correspondaient à ce que prétend la documentation minimaliste.

Dimensions extérieures

La plupart des modules ont une taille qui se situe entre 1,8 sur 1,5 cm et 4 cm sur 2. Certains modules vont jusqu'à 3,5 cm sur 2,5 cm, notamment un module relais 5 V. La Figure 9.2 montre une sélection de dimensions. Si vous avez besoin de dimensions précises, sachez qu'elles peuvent varier de plus ou moins 1 mm.

La plupart des modules comportent des trous de fixation, pour des vis de 3. La Figure 9.3 montre deux modules réunis grâce à deux entretoises nylon et des vis. Il s'agit d'un capteur de température et d'un détecteur d'inclinaison et de choc au mercure.

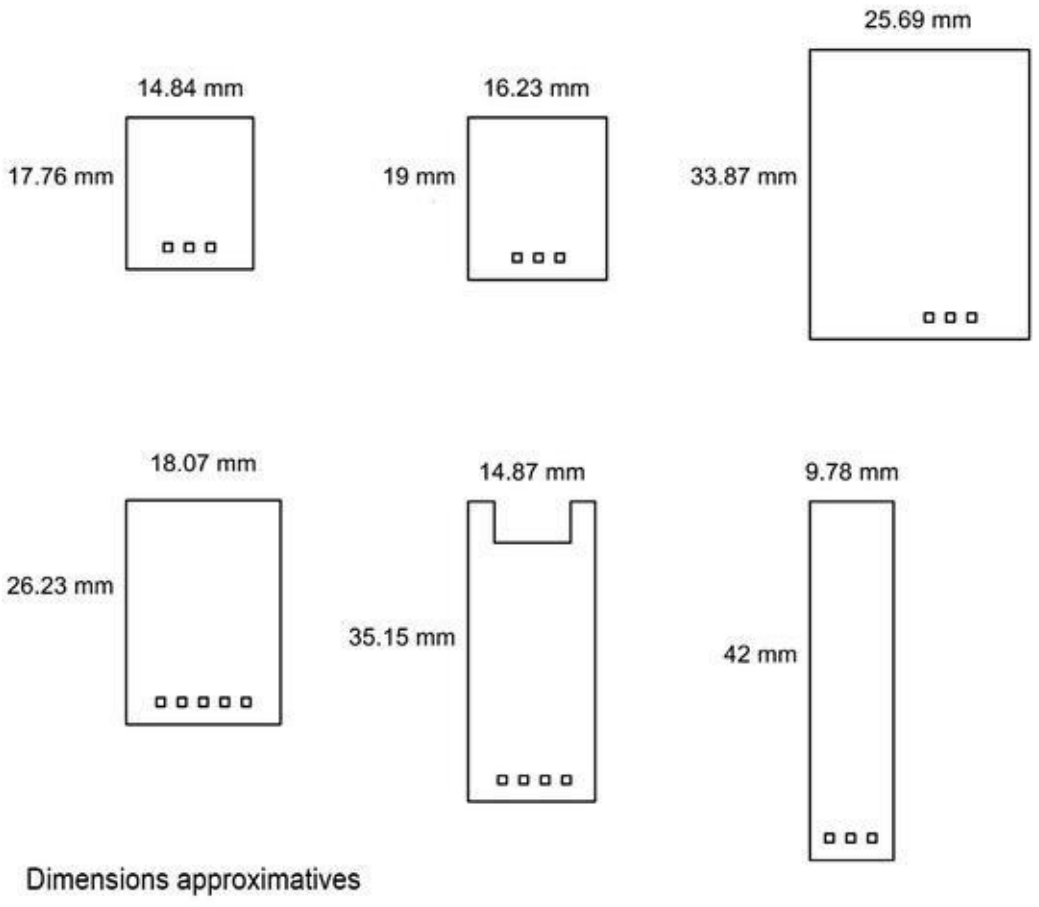


Figure 9.2 : Quelques formats physiques de modules.

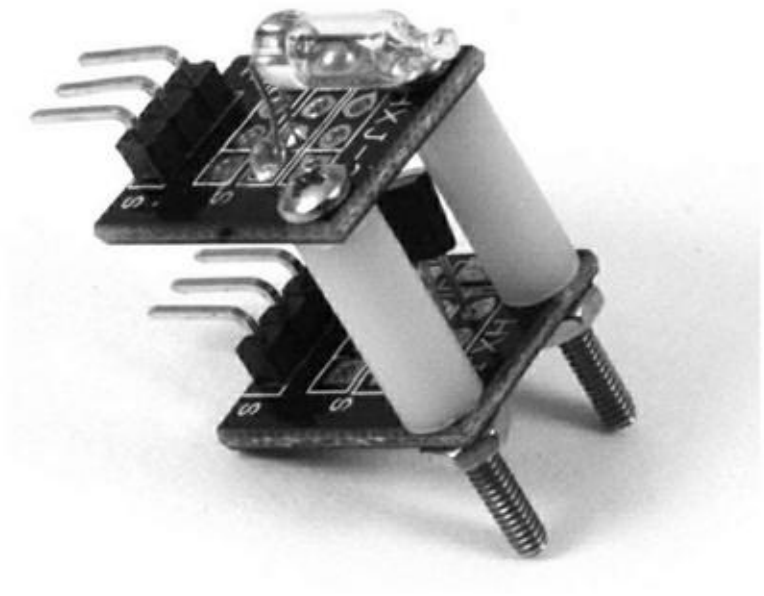


Figure 9.3 : Deux modules réunis par des entretoises nylon et des vis.

Tous les modules ne peuvent pas être raccordés de cette façon. Parfois, les trous ne sont pas alignés sur les circuits. Certains modules n'ont même pas de trous de fixation. Vérifiez toujours cela si vous comptez réunir plusieurs modules.

Interfaces électriques

Les brochages des modules sont variables. La seule famille qui offre un certain standard est celle de TinkerKit. La Figure 9.4 montre quelques exemples de brochages répandus.

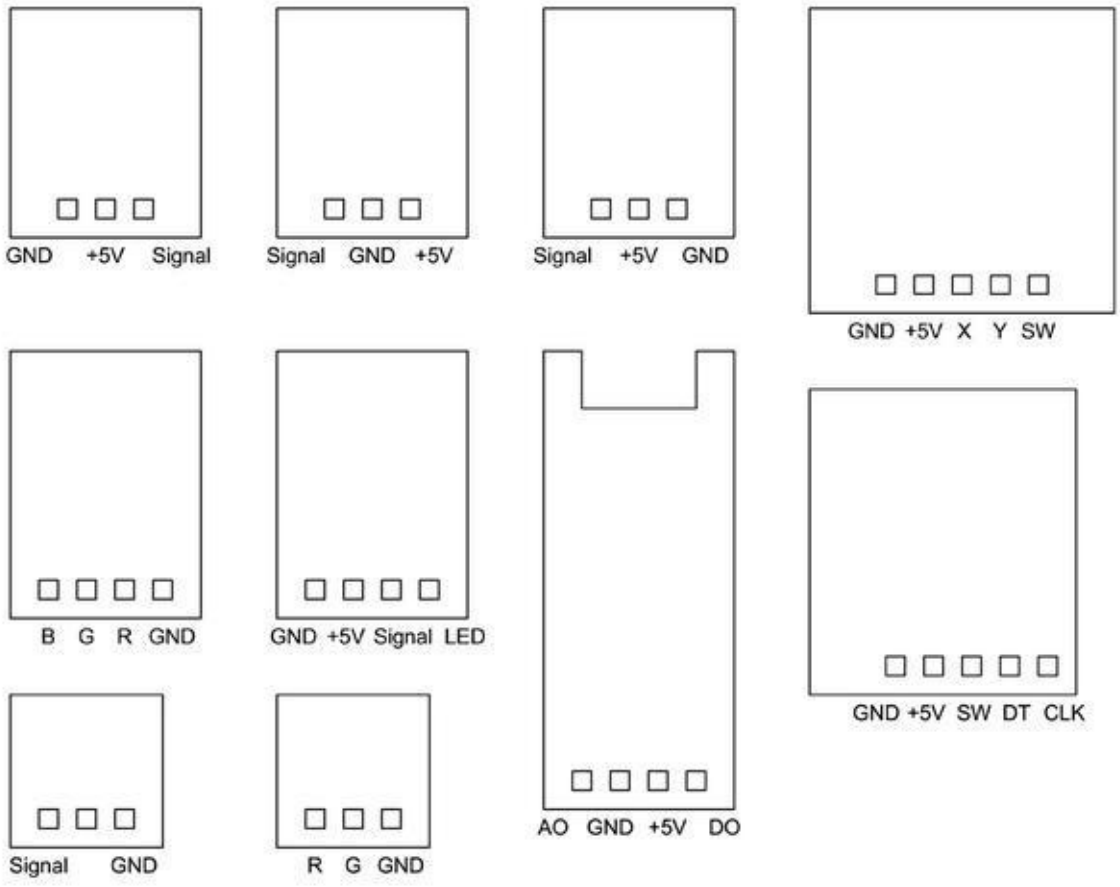


Figure 9.4 : Exemples de brochage de modules.

Méfiez-vous d'une compatibilité apparente entre les connecteurs à trois ou quatre broches des modules et les boucliers d'extension d'entrées-sorties équipés de connecteurs qui leur ressemblent (tels que ceux décrits dans le Chapitre 8). La compatibilité broche à broche entre un module et un bouclier n'est garantie que pour les familles de modules qui ont été conçus pour fonctionner avec ces boucliers. C'est le cas par exemple de la gamme TinkerKit lorsque vous connectez un module TinkerKit à un bouclier d'extension TinkerKit. De plus, vérifiez toujours la position des broches d'alimentation et de signal avant toute connexion d'un module.

Les microcontrôleurs AVR sont relativement robustes, ce qui permet de connecter plusieurs capteurs sur les entrées d'une carte Arduino, et quelques actionneurs sur les sorties. Les équipements gourmands en courant, comme un laser ou une diode tricolore, ont vraiment besoin d'un circuit spécial capable de fournir le courant que le microcontrôleur ne peut pas assumer (revoyez la description des sources et des puits de courant du Chapitre 3).

Certains modules de sortie disposent d'une interface de courant fort, mais pas tous. Vous pouvez utiliser un circuit très simple, comme celui de la Figure 9.5, pour augmenter le courant que peut fournir votre projet à un relais ou un module de diode laser. La valeur de la résistance R dans le circuit de droite de la figure dépend de la diode LED et donc du courant dont elle a besoin. Tant que ce courant ne dépasse pas les capacités du transistor, cela devrait fonctionner.

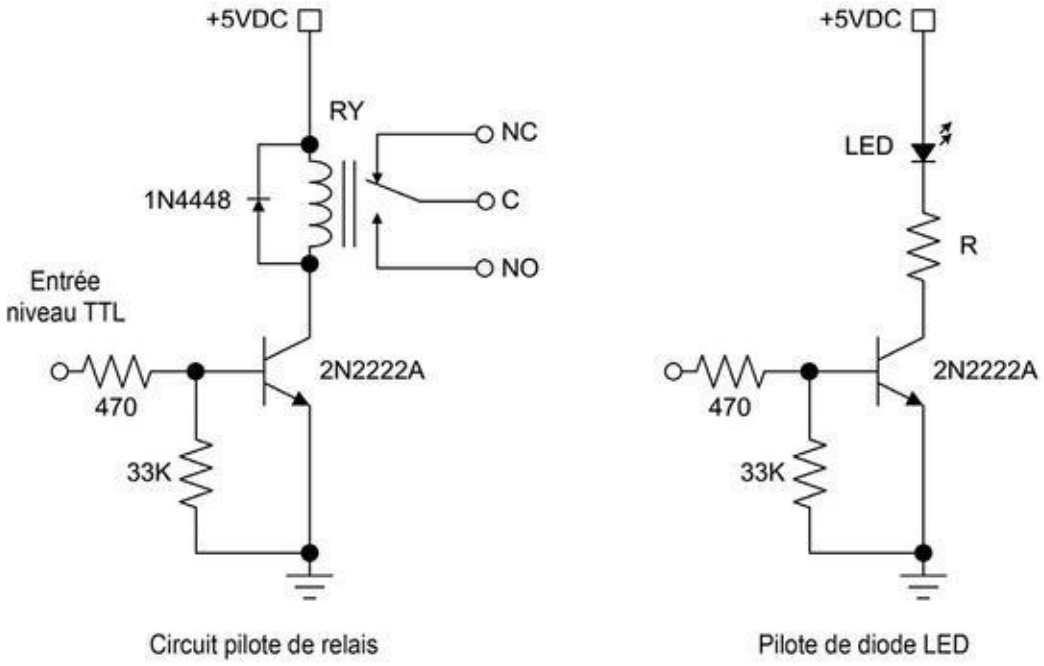


Figure 9.5 : Circuit de sortie à courant fort.

Une autre solution consiste à utiliser un circuit dédié tel que le MAX4896 (Figure 9.6). Ce circuit fonctionne avec l'interface SPI et peut directement être relié à l'Arduino. Il est prévu pour piloter de petits relais, mais il peut également prendre en charge des diodes LED de grande puissance.

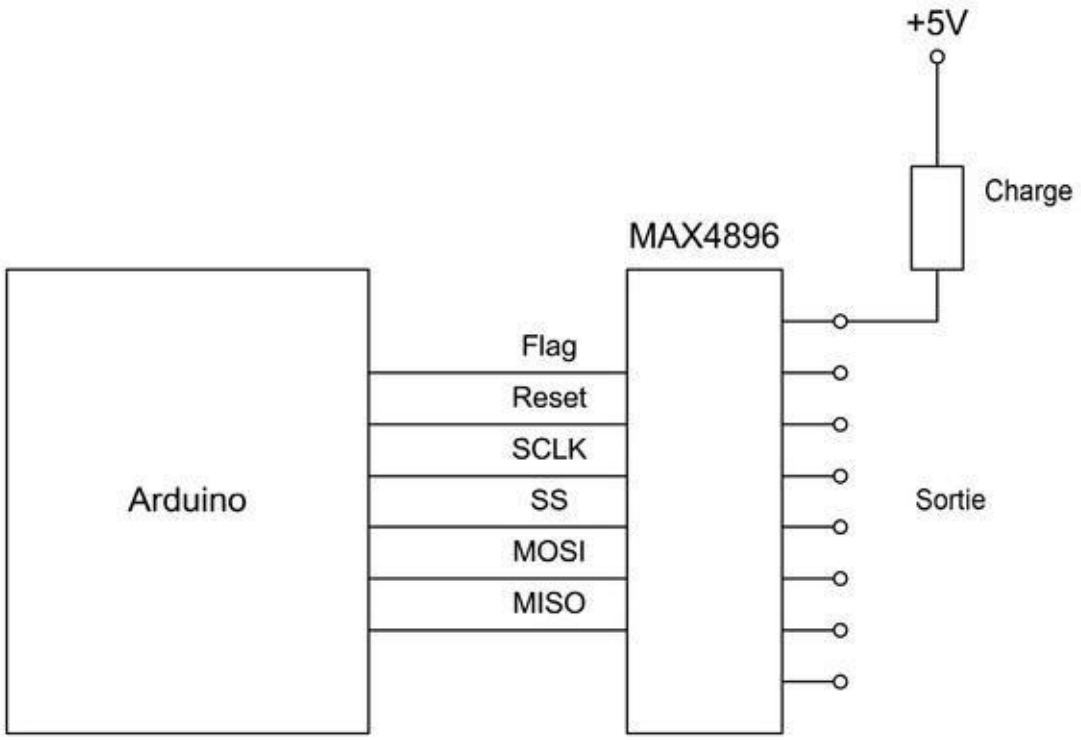


Figure 9.6 : Circuit de pilotage de sortie pour plusieurs relais et autres charges.

Approvisionnement en modules

Le Tableau 9.1 cite quelques fournisseurs de modules vendus à l'unité ou sous forme d'assortiments de 24, 36 ou même plus. Vous en trouverez bien sûr sur les sites de commerce en ligne. Par exemple, la Figure 9.7 montre une valise réunissant 37 capteurs différents (la dernière case contient deux types, ce qui explique le nombre impair).

Figure 9.7 : Exemple de mallette d'assortiment de capteurs d'entrées-sorties.

Tableau 9.1 : Quelques fournisseurs de modules d'entrées-sorties.

Nom	Adresse
Adafruit	www.adafruit.com
CuteDigi	store.cutedigi.com
DealeXtreme (DX)	www.dx.com
KEYES	en.keyes-robot.com
SainSmart	www.sainsmart.com
Seeed Studio	www.seeedstudio.com
TinyCircuits	www.tiny-circuits.com

Trossen Robotics	www.trossenrobotics.com
Vetco	www.vetco.net

La plupart de ces fournisseurs vendent également les accessoires, la câblerie et les circuits actifs élémentaires avec lesquels sont constitués leurs modules. Quelques rares modules présents dans les assortiments ne sont hélas pas disponibles individuellement.

Description des modules

Dans cette longue section, je vous propose de passer en revue les modules de trois principaux fournisseurs : Keyes, SainSmart et TinkerKit. Mes descriptions resteront brèves, l'objectif étant de rappeler l'encombrement physique et les connexions électriques. Les principes de fonctionnement plus détaillés pour les différentes catégories de capteurs sont disponibles un peu plus loin dans le même chapitre, avec des renvois vers les modules qui utilisent les différents capteurs.

Vous constaterez rapidement que de nombreux modules utilisent le même circuit de base, par exemple ceux qui utilisent un ampli opérationnel LM393, exploité en tant que comparateur avec un capteur. Dans ce type de circuit, un potentiomètre permet de régler la tension d'équilibre entre les deux entrées du comparateur, et la sortie du circuit est rendue disponible sur la broche de signal du module. Le Tableau 9.2 présente les modules basés sur ce circuit, et le Tableau 9.3 les modules SainSmart correspondants.

Tableau 9.2 : Modules Keyes utilisant un comparateur (Figure 9.8).

Réf.	Nom
KY-025	Relais magnétique Reed
KY-026	Détecteur de flamme
KY-036	Capteur de contact à conduction
KY-037	Microphone sensible
KY-038	Microphone

Tableau 9.3 : Modules à comparateur de SainSmart (Figure 9.8).

Réf.	Nom
20-011-981	Capteur de luminosité
20-011-982	Capteur de choc/de vibration
20-011-983	Capteur magnétique à effet Hall
20-011-984	Détecteur de flamme

La Figure 9.8 propose un schéma générique de comparateur basé sur le LM393 ; c'est le circuit que vous trouverez dans les modules des deux tableaux précédents. Les valeurs exactes des résistances peuvent varier d'un module à l'autre. Ce qui distingue les modules est essentiellement le type de capteur utilisé (détection infrarouge, détection audio, détection de luminosité, *etc.*).

La broche DO dans ce schéma est celle de la sortie numérique (*Digital Output*) qui correspond directement à la sortie du comparateur. Elle est à l'état Haut lorsque l'entrée positive voit une tension supérieure à celle de l'entrée inverseuse. Elle est à l'état Bas dans le cas inverse. Notez que certains modules utilisent une logique inverse pour les états, mais le principe reste le même.

La tension présente sur l'entrée positive (non inverseuse) peut être réglée grâce au potentiomètre (le potard) qui constitue une partie d'un diviseur de tension, l'autre partie étant constituée du capteur et d'une éventuelle résistance pour limiter le courant. L'autre diviseur de tension comporte deux branches symétriques et permet d'amener sur l'entrée négative une tension qui est égale à la moitié de la tension d'alimentation, soit 2,5 V. Cette tension est la référence à laquelle va être comparée la tension variable de l'entrée positive. La sortie du comparateur va donc basculer entre Bas et Haut en fonction du changement de résistance du circuit dans lequel se trouve le capteur.

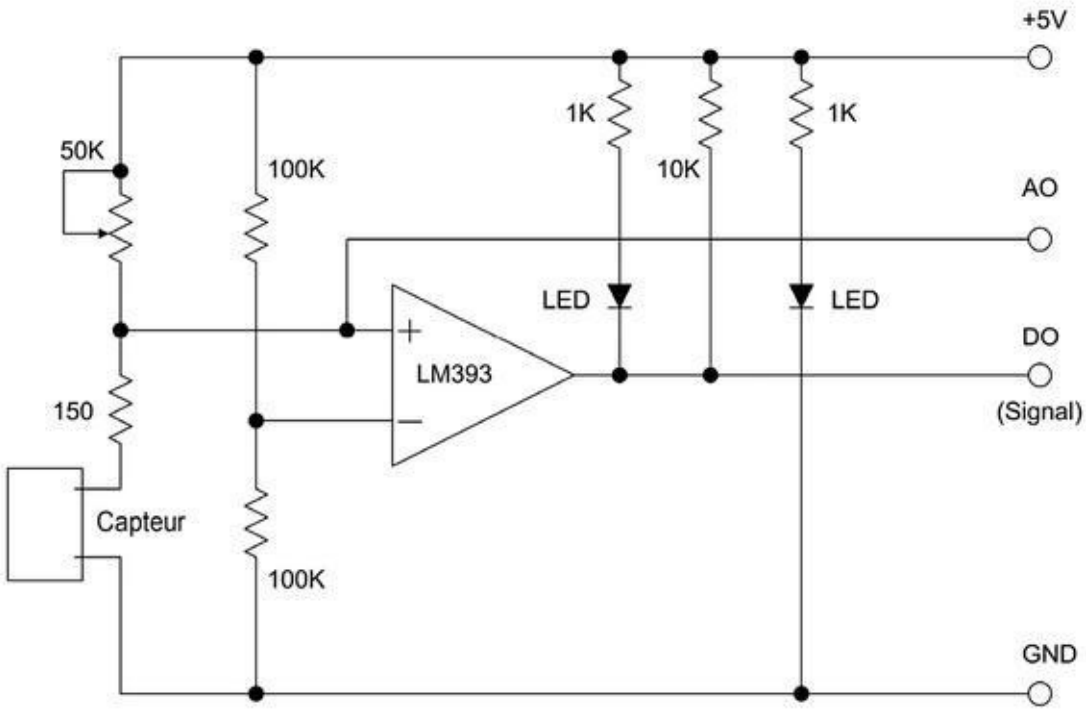


Figure 9.8 : Circuit générique basé sur un circuit comparateur.

Une résistance de 10 kilo-ohms est ajoutée entre le pôle positif et la sortie. Elle sert de résistance de tirage. La sortie analogique AO délivre la valeur brute du capteur, telle que présentée en entrée du LM393. Les modules qui n'offrent qu'une sortie ne donnent pas accès à cette valeur analogique. Certains modules sont dotés d'une diode LED de présence de tension, et d'autres y ajoutent une diode témoin qui s'allume lorsque la sortie passe à l'état Bas, c'est-à-dire que la sortie devient un puits de courant pour la diode vers la masse.

C'est la façon dont le capteur réagit au changement de la grandeur physique qu'il doit détecter qui détermine le comportement du module, et vous pouvez avoir à faire quelques essais pour bien comprendre comment le module fonctionne. En général, un capteur qui est activé va offrir une moindre résistance, ce qui va entraîner une tension plus faible sur l'entrée positive que sur l'entrée de référence. Le résultat est que la sortie du comparateur va passer à l'état Bas, ce qui va allumer la diode LED de sortie.

Les comparateurs qui présentent un état Bas en sortie quand l'entrée est active sont appelés des circuits actifs Bas. Une tension faible en sortie doit être considérée comme une condition vraie. Les circuits qui fonctionnent dans l'autre sens, en passant dans l'état Haut lorsque le capteur est sollicité, sont appelés

actifs Haut. Les deux valeurs true et false dénotent le fait que le capteur détecte ou pas quelque chose, respectivement.

Soumettez vos modules à un examen approfondi avant de les relier à une carte Arduino en vous servant d'un multimètre numérique et d'une loupe. C'est absolument indispensable dans le cas des modules aux prix tirés. J'ai ainsi pu constater que certains avaient oublié des composants, ou qu'il y avait des courts-circuits entre des plots. J'ai même trouvé un module sur lequel il y avait un pont de soudure entre une piste du positif +5 V et une piste de masse ! Vous imaginez le résultat si j'avais branché ce module. Par ailleurs, même si un module offre trois broches pour le +5 V, la masse et le signal, toutes trois ne sont pas toujours utilisées, et parfois le brochage est erroné. Une fois que vous avez fait un bilan, vous devriez pouvoir vous servir correctement des modules. Après tout, ils embarquent un faible nombre de composants. Dans tous les cas, prenez des notes de ce que vous avez détecté pour vous éviter des soucis ultérieurs.

Le fabricant chinois Shenzhen Keyes DIY Robots Co. Ltd., abrégé en Keyes, offre une grande variété de modules, à l'unité ou en assortiment (de 36 en général). Vous trouverez des modules qui portent les trois lettres « HXJ », qui sont compatibles avec les modules Keyes avec quelques variations au niveau des pistes du circuit imprimé.

La liste des modules Keyes présentés dans la suite correspond au Tableau 9.4. La description individuelle de chaque module fait partie de la section suivante. À l'heure où j'écris ces lignes, il existe les références KY-001 jusqu'à KY-040, avec quelques trous que je ne m'explique pas : il n'y a pas de -007, de -014, de -029, ni de -030. Tous les modules ont le même brochage, et certains portent le même nom avec des caractéristiques différentes.

Tableau 9.4 : Modules de la famille Keyes.

Réf.	Nom	Réf.	Nom
KY-001	Capteur de température	KY-021	Interrupteur Reed miniature
KY-002	Capteur de vibration	KY-022	Récepteur infrarouge
KY-003	Capteur magnétique Hall	KY-023	Joystick 2 axes
	Bouton poussoir		Capteur Hall linéaire

KY-004	
KY-005	Émetteur infrarouge
KY-006	Buzzer passif
KY-008	Diode LED laser
KY-009	LED tricolore
KY-010	Interrupteur optique
KY-011	LED bicolore
KY-012	Buzzer actif
KY-013	Capteur de temp. analogique
KY-015	Capteur de température/humidité
KY-016	LED tricolore
KY-017	Capteur de tilt à mercure
	Module à photorésistance
KY-024	
KY-025	Module Reed
KY-026	Détecteur de flamme
KY-027	Interrupteur Magic Light Cup
KY-028	Capteur de température
KY-031	Capteur de choc
KY-032	Capteur de proximité infrarouge
KY-033	Suiveur de ligne infrarouge
KY-034	Diode clignotante automatique
KY-035	Capteur magnétique Hall
KY-036	Capteur de contact
KY-037	Microphone sensible
	Microphone

KY-018		KY-038	
KY-019	Relais 5 V	KY-039	Capteur de pouls à LED
KY-020	Capteur de tilt	KY-040	Encodeur rotatif

Un autre fournisseur important de modules capteurs est la société SainSmart. Son offre est présentée dans le Tableau 9.5. Une section ultérieure décrit chacun des modules individuellement. Tous ces modules peuvent être achetés indépendamment, et certains n'ont pas de concurrents.

Tableau 9.5 : Modules de la famille SainSmart disponibles en kit.

Réf.	Nom
N/A	Relais
20-011-985	Capteur de contact
20-019-100	Télémètre à ultrasons HC-SR04
20-011-984	Détecteur de flamme
20-011-986	Capteur température/humidité
N/A	Buzzer actif
20-011-982	Capteur de choc/vibration
N/A	Buzzer passif
20-011-987	Suiveur de ligne
20-011-983	Capteur magnétique Hall
20-011-981	Capteur photosensible

N/A	Récepteur infrarouge
20-011-944	Joystick
20-011-946	Détecteur d’eau

Le troisième fournisseur important est TinkerKit (Tableau 9.6). La description détaillée des modules se trouve dans une section ultérieure. Les modules présentés ici font partie du kit portant le nom Pro. Ils sont destinés à être connectés à un bouclier d’interface TinkerKit (décrit dans le Chapitre 8). Actuellement, l’avenir de cette société est en question, mais les produits sont disponibles chez les principaux revendeurs, et notamment Element14. Les librairies de fonctions sont disponibles sur le site de référence GitHub. Des fiches techniques sont disponibles sur le site de Mouser.

Tableau 9.6 : Modules de la famille TinkerKit.

Réf.	Nom	Réf.	Nom
T000020	Accéléromètre	T010020	Module MOSFET
T000030	Joystick	T010110	Module LED forte puissance
T000070	Capteur magnétique Hall	T010111	Module LED 5 mm bleue
T000090	Photorésistance	T010112	Module LED 5 mm verte
T000140	Potentiomètre rotatif	T010113	Module LED 5 mm jaune
T000150	Potentiomètre linéaire	T010114	Module LED 5 mm rouge
T000180	Bouton poussoir	T010115	Module LED 10 mm bleue
T000190	Capteur de tilt	T010116	Module LED 10 mm verte

T000200	Thermistance	T010117	Module LED 10 mm jaune
T000220	Capteur de contact	T010118	Module LED 10 mm rouge

La description individuelle détaillée des modules TinkerKit un peu plus loin ne précise pas le brochage, car tous les modules utilisent le même. La seule différence est que certains modules ont une sortie numérique sur la broche de signal et d'autres une sortie analogique. Le connecteur est celui représenté dans la Figure 9.9.

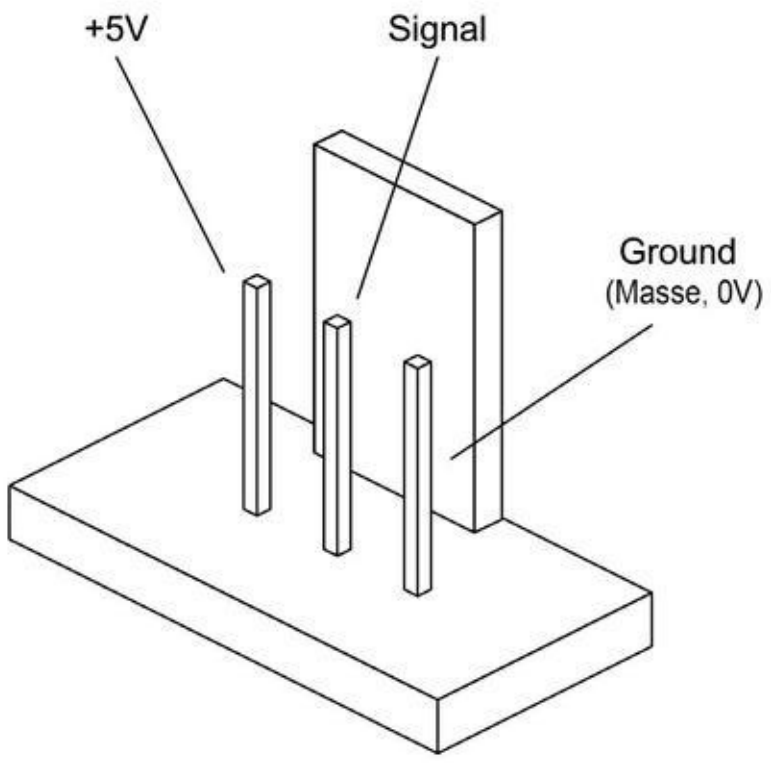


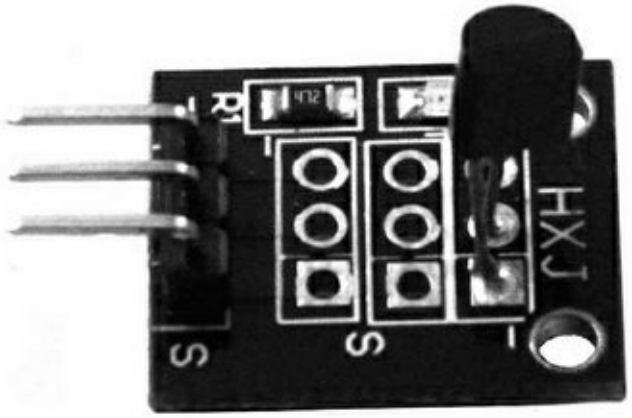
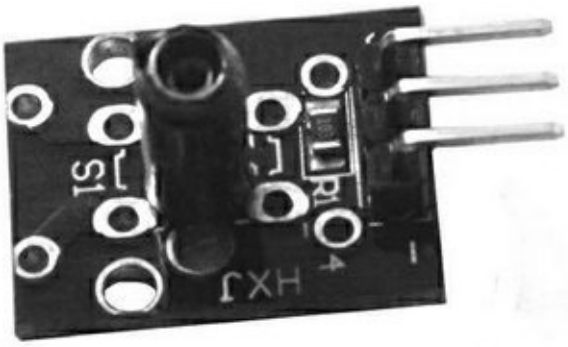
Figure 9.9 : Connecteur standardisé des modules TinkerKit.

Les modules TinkerKit sont tous dotés d'une interface active, par exemple pour détecter un niveau, ajouter une amplification, *etc.* Autrement dit, la carte Arduino ne peut jamais communiquer directement avec le capteur d'une carte

TinkerKit, ni avec l’actionneur ou l’élément de contrôle. Elle ne dialogue qu’avec le circuit d’interface. Pour savoir quel type de circuit est en place, il suffit de regarder sur le dessous du module. Même le simple module de diode LED est doté d’un transistor de pilotage. L’objectif de la gamme TinkerKit est de proposer des composants faciles à relier à une carte Arduino et relativement protégés contre les erreurs de manipulation. Le prix à payer pour cette simplification est un moindre accès à bas niveau aux composants.

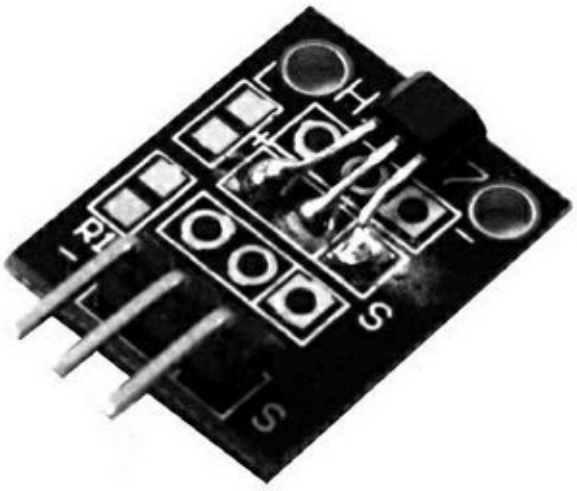
Modules Keyes

Tableau 9.7 : Description des modules capteurs et actionneurs Keyes.

Réf.	Nom et description	Image
KY-001	**Capteur de température** Embarque un capteur DS18B20 sur un fil en boîtier TO92.	
KY-002	**Capteur de vibration** Capteur étanche forçant la sortie à l'état Bas en cas de choc. La broche +5 V semble ne pas être connectée, mais il y a une place pour une résistance sur le dessous. Très sensible.	
KY-003	**Capteur de champ magnétique à effet**	

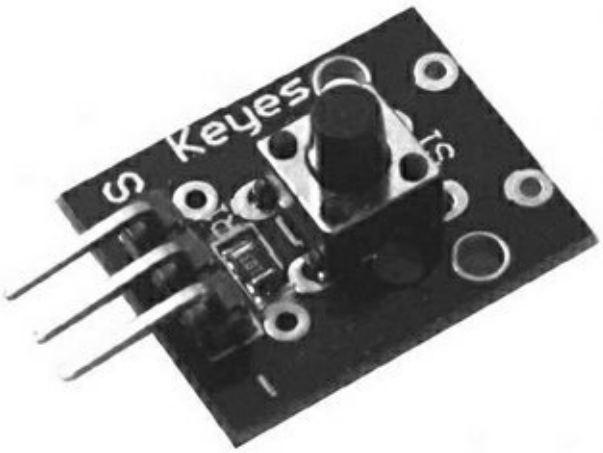
Hall

La sortie est le collecteur ouvert d'un transistor NPN dans le circuit A3144. La sortie est forcée à l'état Bas en cas de détection. Le capteur n'est pas linéaire, à la différence du module KY-024.



KY-004 **Bouton poussoir**

La sortie est forcée à l'état Bas quand le poussoir est enfoncé.



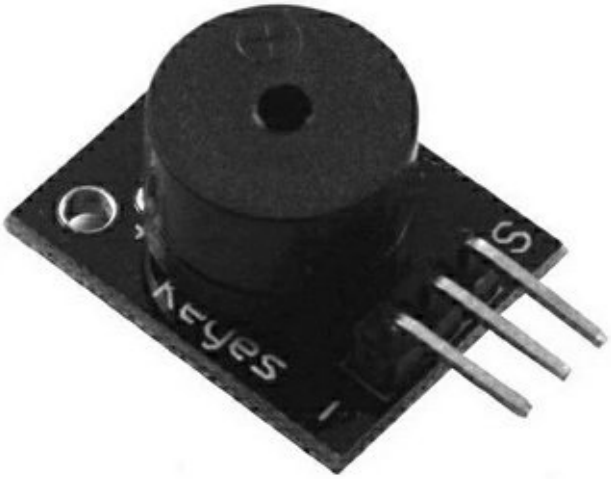
KY-005 **Émetteur infrarouge**

Émetteur à diode LEDIR à combiner au module KY-022. Il faut ajouter une résistance de

limitation de courant externe. Sur mon module, la broche GND n'était pas branchée.

KY-006 **Buzzer**

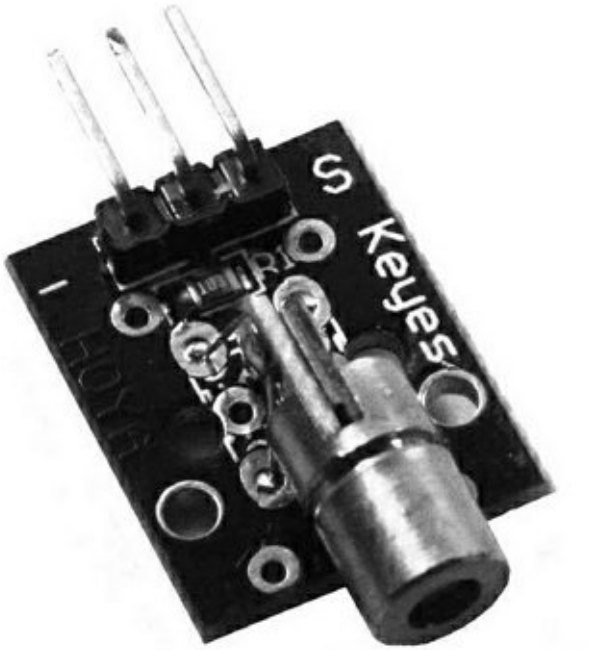
Petit haut-parleur à diaphragme métallique. La broche +5 V n'est pas connectée.



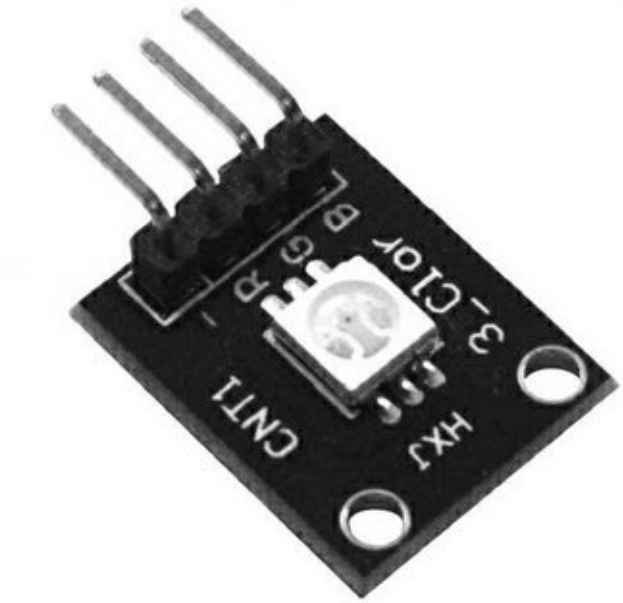
KY-008 **Diode laser**

Diode LED 650 nm, rouge, de faible puissance (voir aussi la Figure 9.51). L'anode est à la masse, et le signal à la cathode. La broche +5

V n’est pas utilisée, mais est reliée au signal par une résistance de 10 kΩ. Il faut ajouter une résistance de limitation de courant externe.

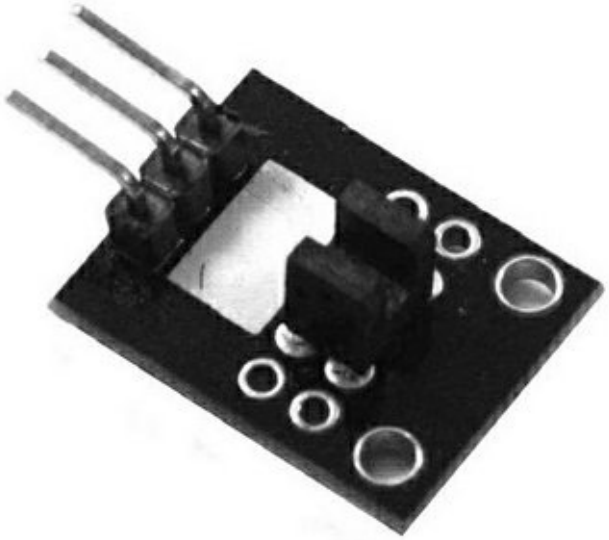


KY-009 **Diode LED tricolore**
Triple diode LED rouge, verte, bleue. Toujours vérifier le brochage, car il est assez variable.

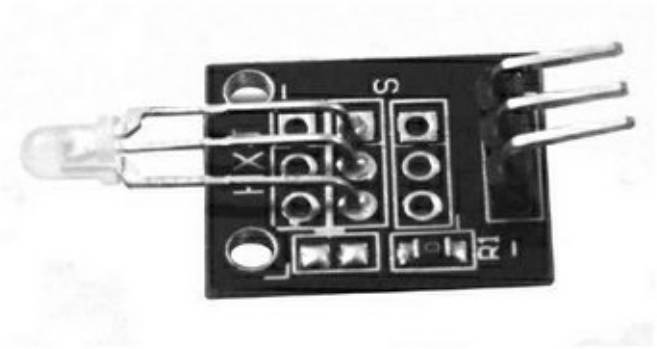


KY-010 **Interrupteur optique**
Constitué d’une diode LED et d’un

phototransistor ;
détecte l’apparition d’un objet dans la ligne de visée.

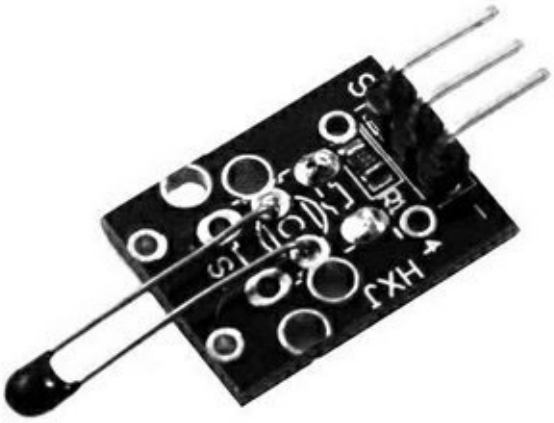


KY-011 **Diode LED bicolore**
Double diode verte ou rouge ou les deux. Les deux diodes partagent une broche, et chacune peut être sollicitée indépendamment.

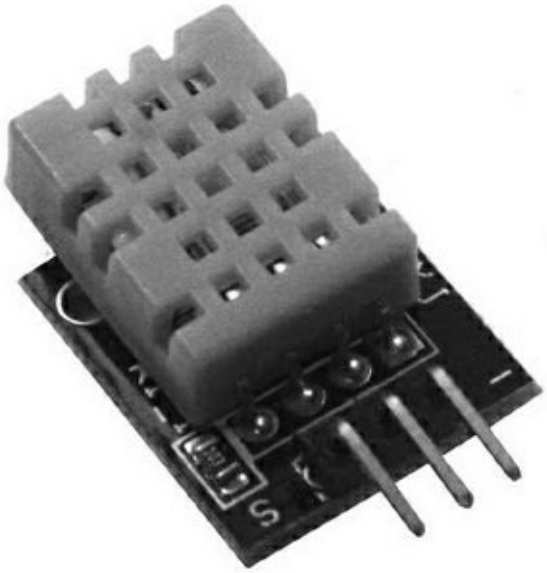


KY-012 **Buzzer actif**
Émet un son de hauteur fixe quand le module est alimenté.

KY-013 **Capteur de température analogique**
Détecte la température grâce à une thermistance (voir la section sur les thermistances plus loin). La tension de sortie varie selon la température.

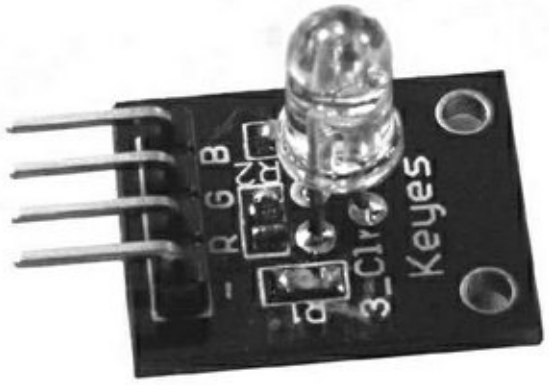


KY-015 **Capteur de température/humidité**
Utilise le capteur DHT11 décrit dans une section ultérieure.



KY-016 **Diode LED tricolore**

Identique au KY-009 mais avec une diode de grande taille et non une diode de type SMC.



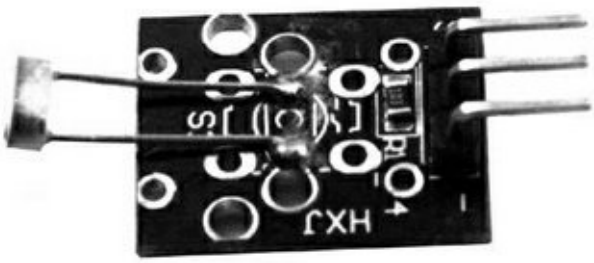
KY-017 **Détecteur de choc à mercure**

Détecteur de choc sur un axe grâce à une goutte de mercure dans un tube en verre (les

capteurs sont décrits plus loin). Vous combinez deux modules pour détecter sur deux axes. La broche +5 V n’est pas branchée au capteur, mais à la broche de sortie par une diode LED et une résistance de 680 ohms.

KY-018 **Module photorésistance LDR**

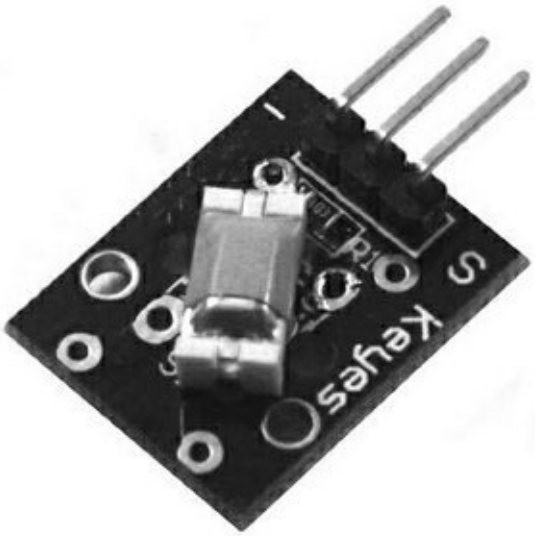
La tension de sortie varie en proportion de la quantité de lumière qui frappe la photorésistance.



KY-019 **Relais 5 V**

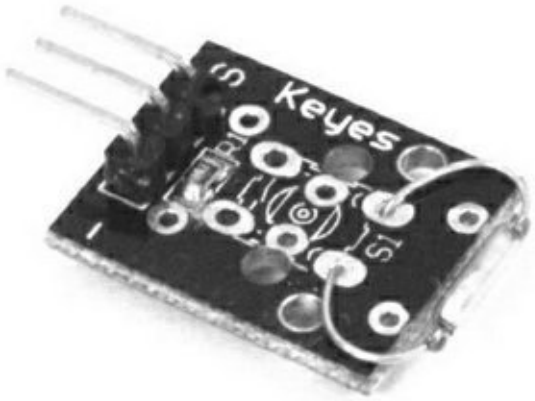
Relais miniature avec circuit pilote pour l’interface avec un signal de niveau logique.

KY-020 **Détecteur de choc**
Similaire au KY-017, mais avec une bille dans un boîtier.



KY-021 **Mini interrupteur Reed**
Petit interrupteur magnétique Reed dans un tube de verre. Le contact se ferme en

présence d’un champ magnétique.

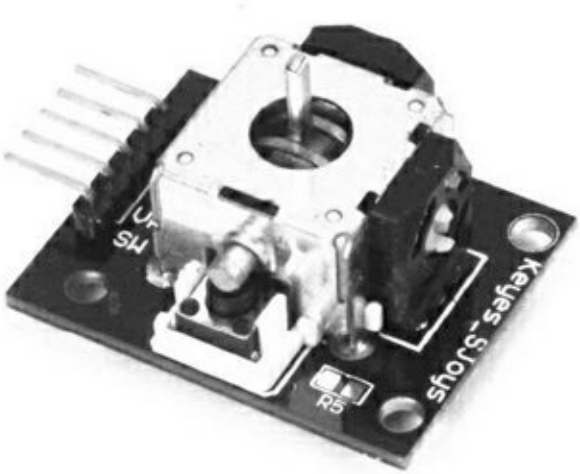


KY-022 **Récepteur infrarouge**

Utilise le même circuit 1838 que les télécommandes de téléviseurs. Fonctionne à 37,9 kHz. La détection d’ondes force la sortie à l’état Haut.

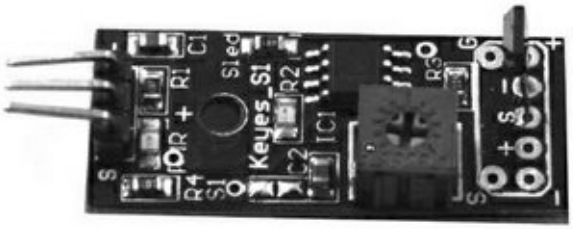
KY-023 **Joystick 2 axes**

Réunit deux potentiomètres à angle droit pour contrôler les déplacements selon l’axe X et l’axe Y.



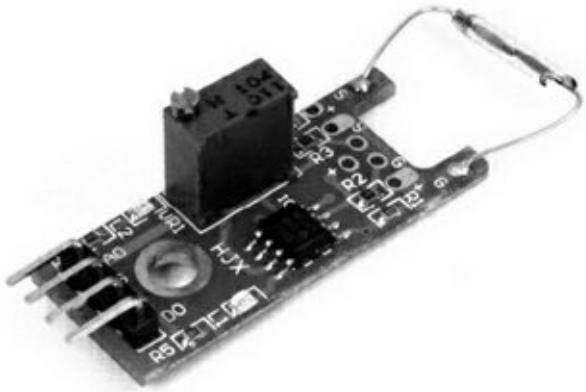
KY-024 **Capteur à effet Hall linéaire**

Utilise le capteur SS49E linéaire et un comparateur LM393 avec un potentiomètre pour ajuster la sensibilité.



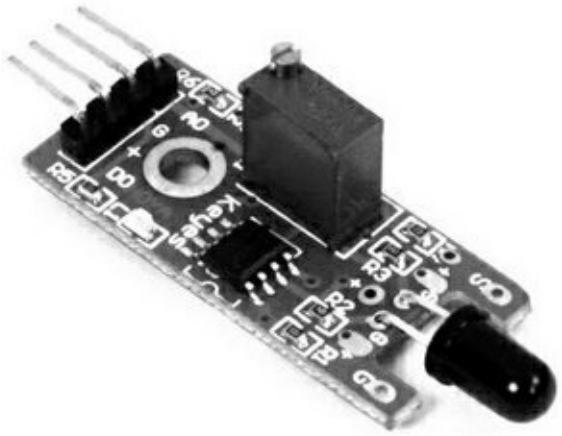
KY-025 **Interrupteur Reed**

Interrupteur magnétique Reed et comparateur comme celui de la Figure 9.8.



KY-026 **Détecteur de flamme**

Le capteur est optimisé pour les longueurs d’onde entre 760 et 1100 nm (nanomètres). Réglage de sensibilité par potentiomètre.

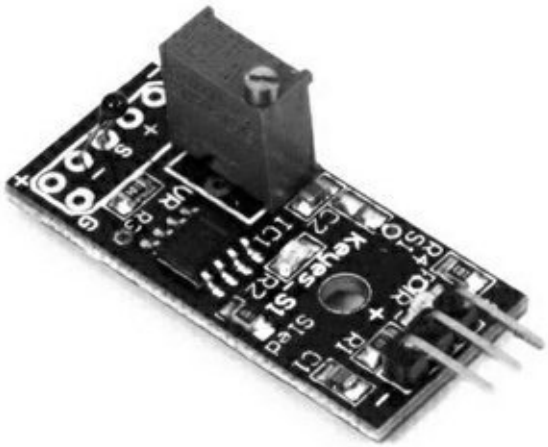


KY-027 **Module Magic Light Cube**

Le module ressemble au KY-017 avec une diode LED sur un second circuit. La diode s’allume quand le module est incliné.

KY-028 **Capteur de température**

Utilise une thermistance et un comparateur pour détecter un seuil. Doté d’un potentiomètre de réglage du seuil.



KY-031 **Détecteur d’impact**

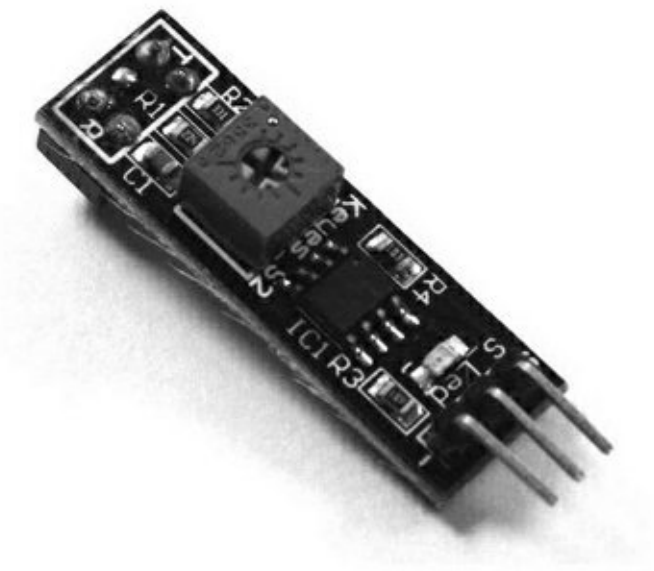
Génère un signal en sortie lors de la détection d’un impact fort. Ne réagit pas aux chocs légers, à la

différence des KY-002, -017 et -020.

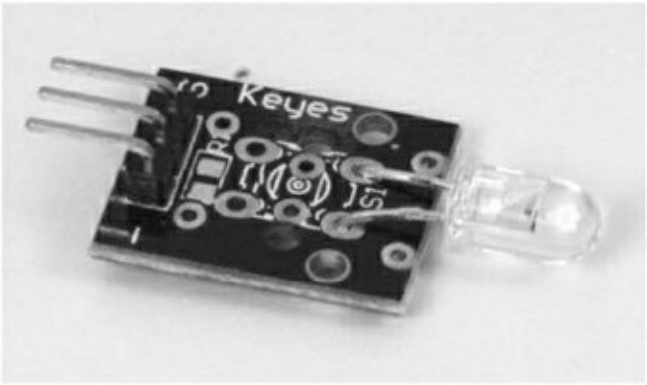
KY-032 **Détecteur de proximité infrarouge**
Détecte une surface ou un obstacle proche grâce au renvoi d'un rayon infrarouge. Sensibilité réglable par potentiomètre. Il existe au moins deux variantes du module. Bien vérifier le brochage avant d'utiliser.

KY-033 **Suiveur de ligne IR**
Détecte la différence entre zones sombres et claires par renvoi d'un rayon infrarouge, ce qui permet de faire suivre

une ligne continue par un robot. Réglage de sensibilité par potentiomètre. Similaire au KY-032, mais la diode LED et le capteur sont montés dessous.

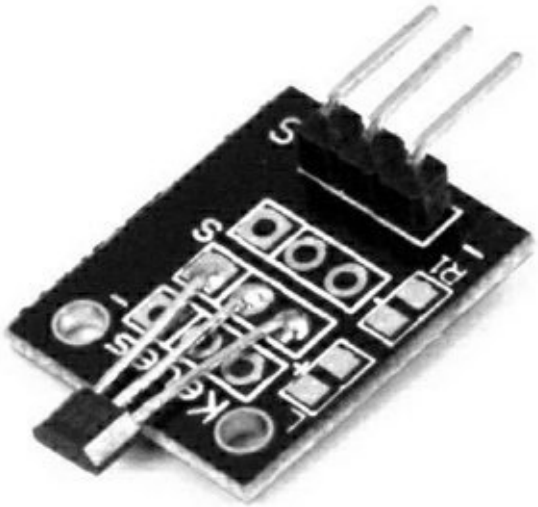


KY-034 **LED couleur clignotante**
La diode LED clignote automatiquement tant que la tension est présente.



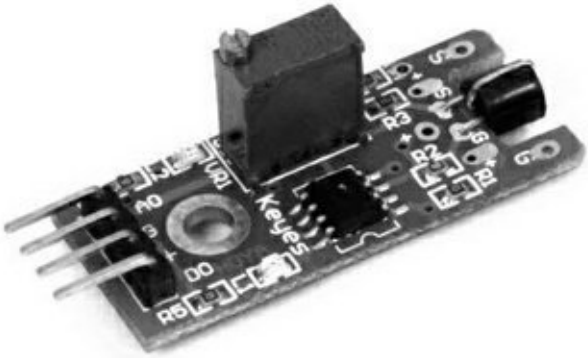
KY-035 **Capteur de champ à effet Hall**
Similaire au KY-003, mais avec un circuit

SSP49E linéaire comme capteur.



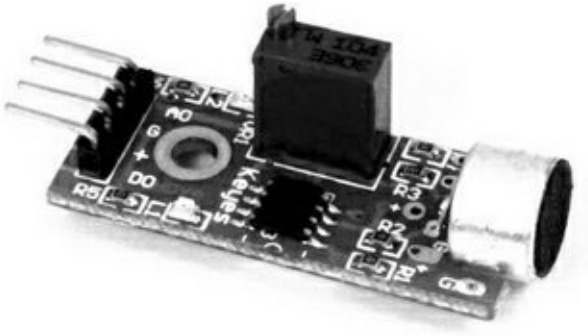
KY-036 **Détecteur de contact à conduction**

La sortie change lorsqu'un objet conducteur tel qu'un doigt touche le fil nu du capteur.



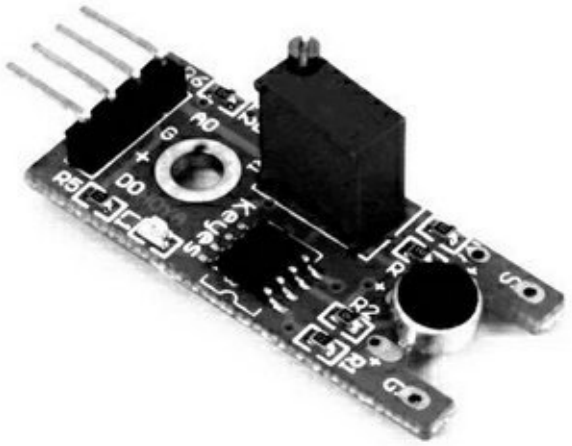
KY-037 **Microphone à seuil**

Microphone avec seuil de bruit réglable. Ce module est plus sensible que le KY-038 de par son plus grand microphone.



KY-038 **Microphone**

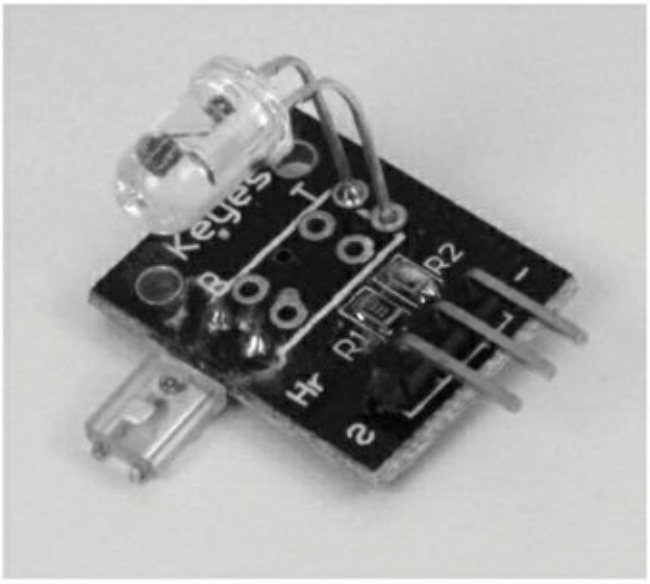
Similaire au KY-037, mais moins sensible de par le microphone de plus petite taille.



KY-039 **Capteur de pouls à LED**

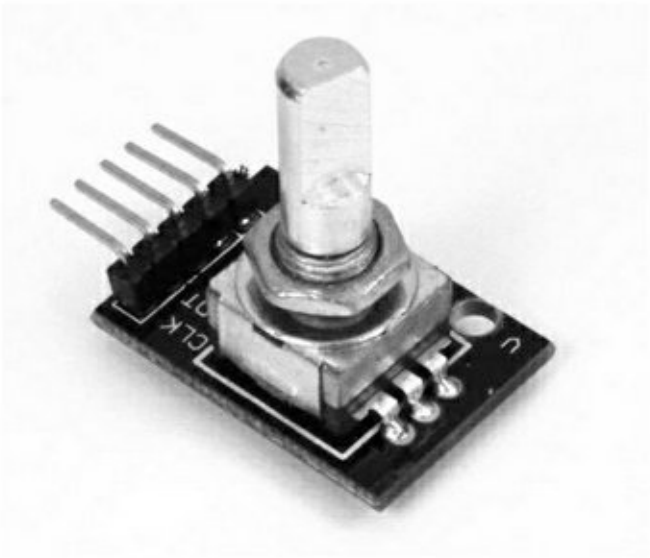
Un phototransistor permet de détecter une faible variation d’intensité émise par la diode LED lors du

passage du sang à travers un doigt.



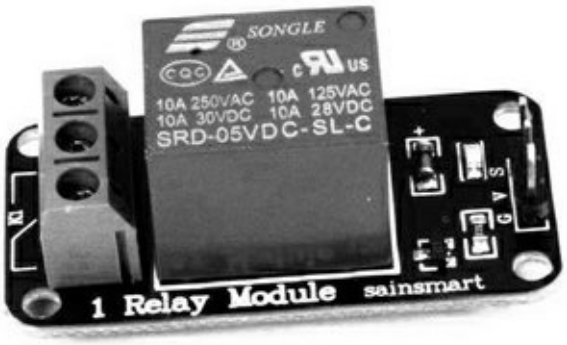
KY-040 **Encodeur rotatif**

Bouton rotatif d’encodage numérique continu. Il est décrit de façon détaillée dans la suite du chapitre.



Les modules SainSmart

Tableau 9.8 : Description des capteurs et actionneurs SainSmart.

Réf.	Nom et description	Image	Bro
	Module relais Relais 5 V avec contact 10 A.		
20-019-100	**Télémètre à ultrasons HC-SR04** Émetteur/récepteur à ultrasons qui mesure le temps de retour de l'écho pour en déduire la distance avec l'obstacle.		
	Buzzer actif Émet un son continu tant que		

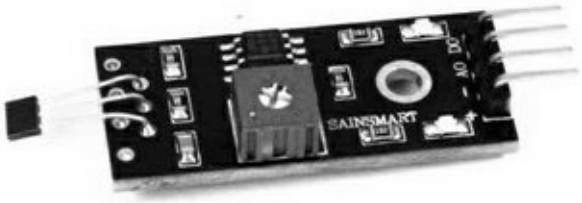
l’alimentation est présente.

Buzzer passif

Émet un son à partir d’une onde carrée, ce qui permet de choisir la fréquence.

20-011-983 **Capteur à effet Hall**

Capteur de champ magnétique Hall linéaire avec potentiomètre de réglage de la sensibilité.



Récepteur infrarouge

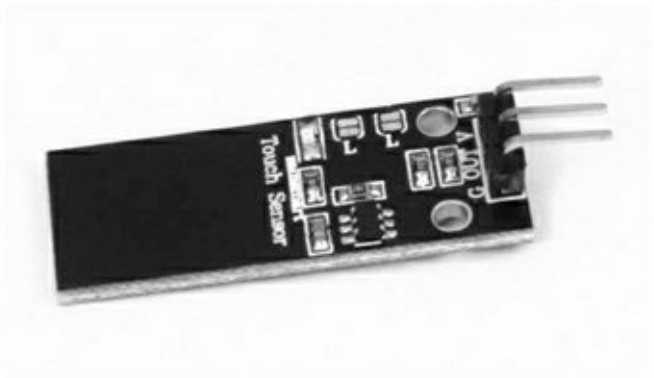
Détecte les impulsions d’un émetteur IR pour générer un signal numérique.

20-011-946 **Détecteur d’eau**

Détecte la présence d’eau par variation de la conductivité entre les pistes métalliques. Peut servir de détecteur de pluie ou d’éclaboussures.

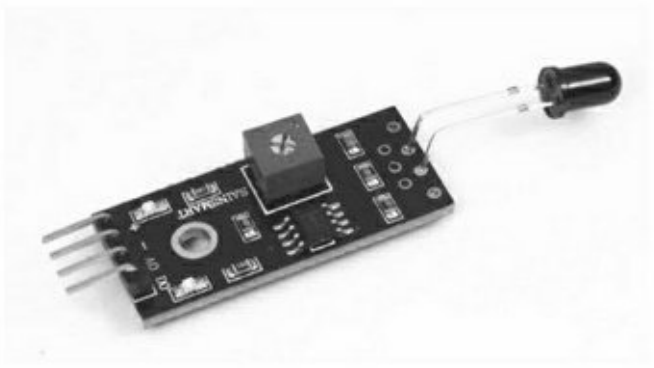
20-011-985 **Capteur de contact**

Détecte le contact d'un doigt.



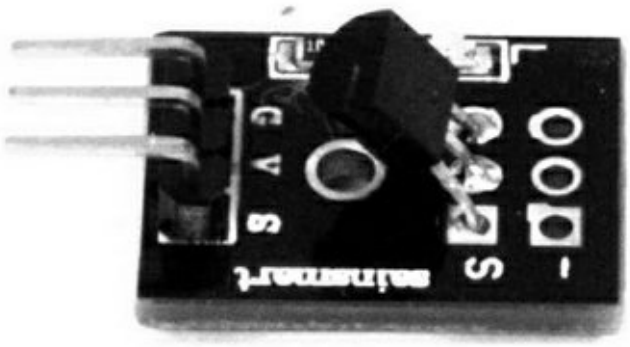
20-011-984 **Détecteur de flamme**

Détecte les ondes infrarouges entre 770 et 1100 nm avec réglage de sensibilité.



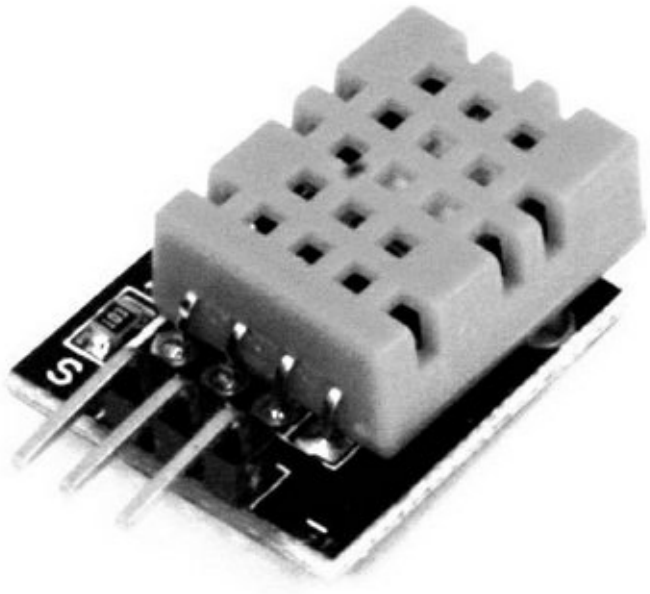
20-011-988 **Capteur de température**

Utilise le capteur DS18B20 sur un fil décrit plus loin. Boîtier TO92.



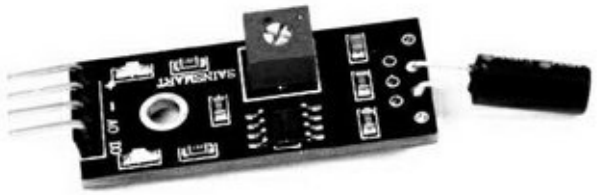
20-011-986 **Capteur de température et d’humidité**

Utilise le DHT11 pour capter la température et l’humidité relative. Les détails sont fournis dans la Figure 9.14.



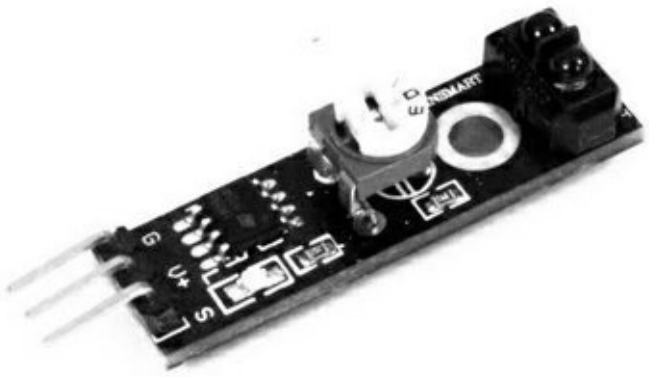
20-011-982 **Détecteur de choc et de vibrations**

Détecte les mouvements brusques et les vibrations avec un capteur étanche comme le Keyes KY-002.



20-011-987 **Module de suivi de ligne**

Utilise une diode LEDIR et un photosenseur pour détecter les variations de brillance d'une surface.



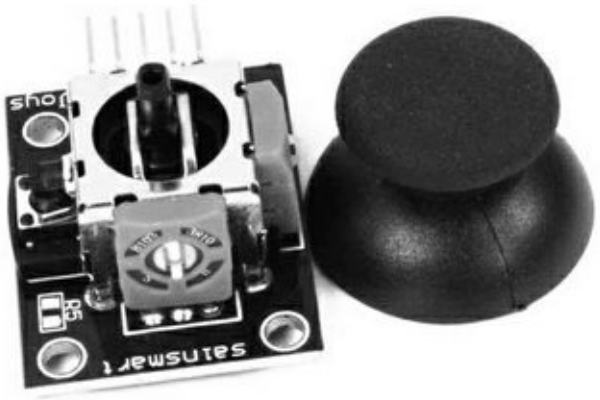
20-011- **Capteur de luminosité**

981 Circuit basique avec sensibilité réglable.

20- **Module Joystick**

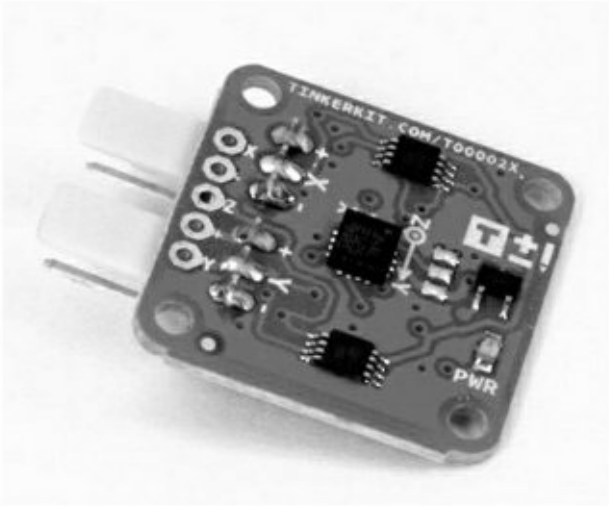
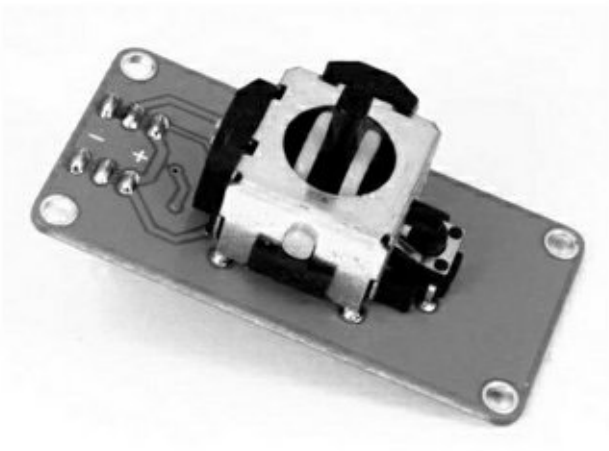
011- Joystick linéaire à

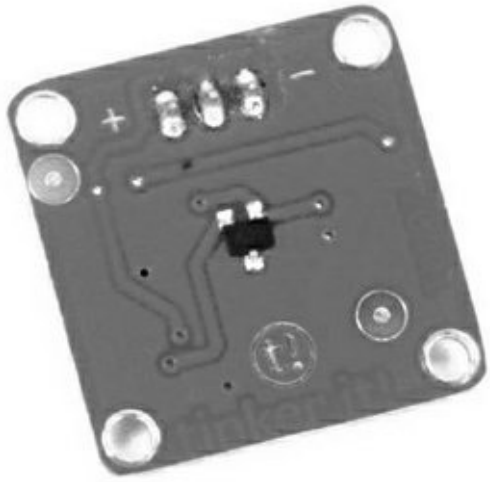
944 deux axes avec gros bouton.



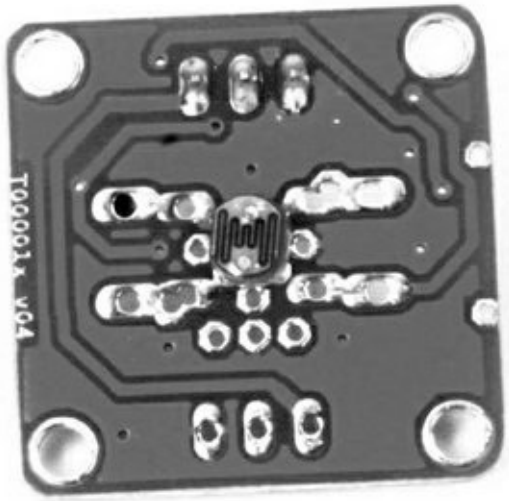
Modules TinkerKit

Tableau 9.9 : Modules d'entrées-sorties TinkerKit.

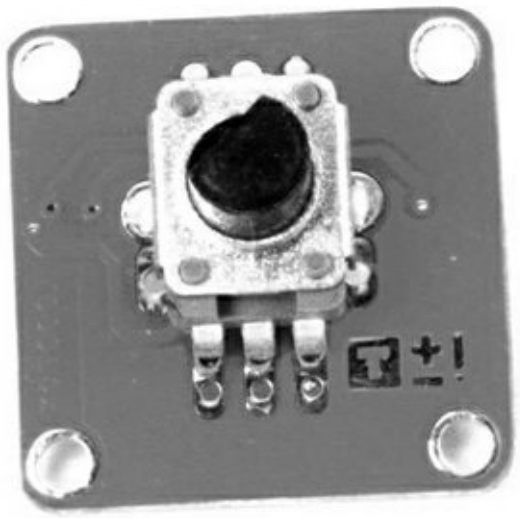
Réf.	Nom et description	Image
T000020	**Accéléromètre** Utilise le circuit à trois axes LIS344ALIC et comprend deux amplificateurs de signal.	
T000030	**Module joystick** Deux potentiomètres montés sur deux axes.	
T000070	**Capteur à effet Hall** La tension de sortie est proportionnelle à la force du champ magnétique détecté.	



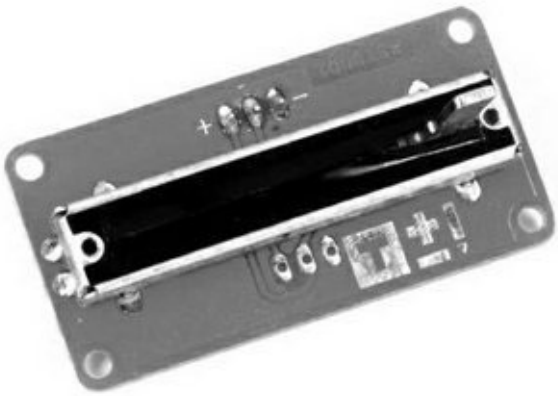
T000090 **Module photorésistance IR**
Une photorésistance avec un amplificateur pour produire une tension proportionnelle à la luminosité.



T000140 **Potentiomètre rotatif**
Un potentiomètre analogique de 4,7 kΩ.



T000150 **Potentiomètre linéaire ou rectiligne**
Potentiomètre de type glissière analogique de 4,7 kΩ.



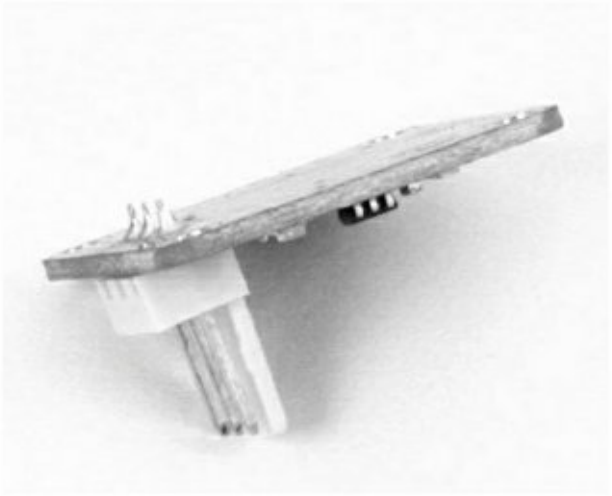
T000180 **Bouton poussoir**
Un bouton poussoir avec contact normalement ouvert. Fournit +5 V lorsqu'il est enfoncé.

T000190 **Détecteur de choc**
Capteur de choc simple avec bille métallique.

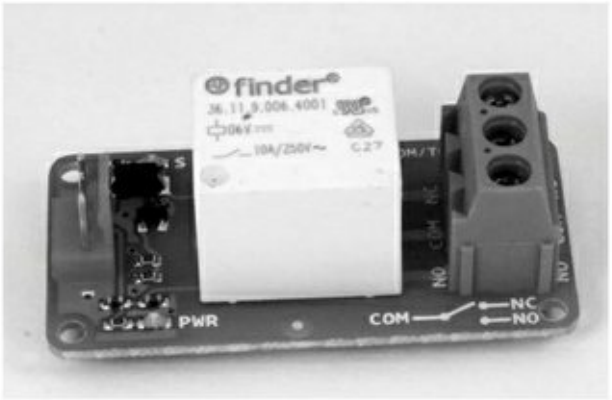
T000200 **Module thermistance**
Embarque une thermistance et un amplificateur pour fournir une tension

proportionnelle à la température.

T000220 **Capteur de toucher**
Utilise le contrôleur QT100A pour fournir +5 V quand le capteur est touché. La partie tactile est sur le devant du circuit (la partie plate dans l'image).



T010010 **Module relais**
Un relais 5 V vers 10 A 250 V, avec un transistor pilote et un bornier à vis.

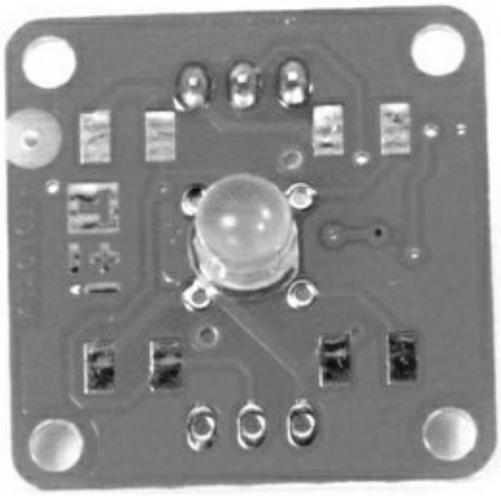


T010020 **Module Mosfet**

Capable d’ouvrir ou de fermer un circuit en courant continu jusqu’à 24 V grâce à un composant IRF520. Suffisamment rapide pour utiliser un signal par impulsion PWM pour un moteur à courant continu.

T010110 **Module diode LED haute puissance**
Réunit cinq diodes LED ultrabrillantes et consomme donc un certain courant. À utiliser avec un module T010020 et une alimentation externe.

T010111 **Diode LED bleue 5 mm**
Module très simple avec une diode LED bleue.



T010112 **Diode LED verte 5 mm**

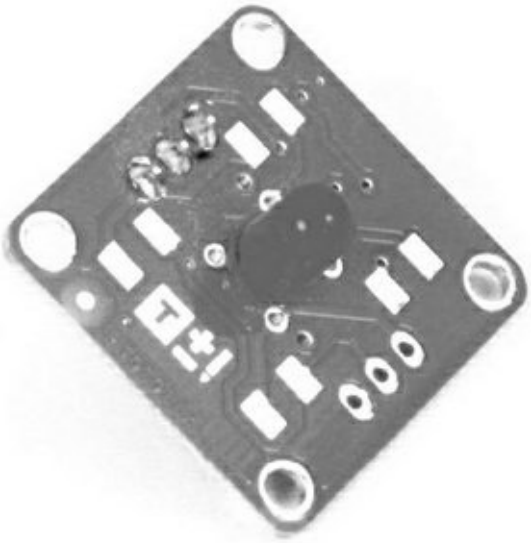
Comme la bleue mais en vert.

T010113 **Diode LED jaune 5 mm**

Comme la bleue mais en jaune.

T010114 **Diode LED rouge 5 mm**

Comme la bleue mais en rouge.



T010115 **Diode LED bleue 10 mm**

Module très simple avec une grande diode LED bleue.

T010116 **Diode LED verte 10 mm**

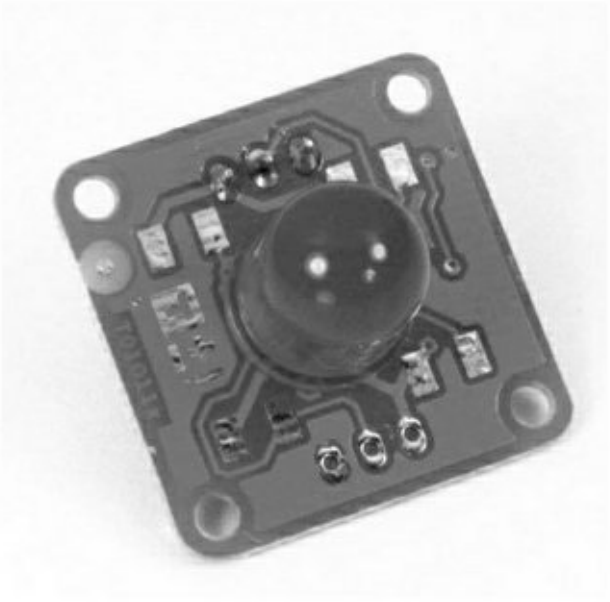
Comme la bleue mais en vert.

T010117 **Diode LED jaune 10 mm**

Comme la bleue mais en jaune.

T010118 **Diode LED rouge 10 mm**

Comme la bleue mais en rouge.



Les modules Grove

La société Seeed Studio a réuni un grand nombre de modules compatibles avec le format d'interconnexion Grove. La plupart d'entre eux correspondent à des modules disponibles chez Keyes, SainSmart ou TinkerKit, mais certains sont uniques.

Les modules Grove sont classés en six groupes : suivi de l'environnement, détection de déplacements, interface utilisateur, surveillance physique, fonctions de porte logique et contrôle de puissance. Tous les modules comportent un connecteur modulaire à quatre broches avec deux broches pour la masse et l'alimentation et deux broches de signal.

Comme pour les modules TinkerKit, les modules Grove offrent l'avantage de ne pas vous demander de chercher ou de vérifier le brochage de chaque module. Les boucliers qui sont compatibles avec les connexions Grove, notamment les boucliers vus dans le Chapitre 8, sont reliés aux modules Grove au moyen d'un câble normalisé. Seeed Studio propose des câbles prêts à l'emploi pour d'autres interfaces avec les modules.

Un autre avantage des modules Grove est la présence de deux oreilles pour fixer le module, alors que d'autres modules ont un trou en plein milieu du circuit imprimé. Bien sûr, les connecteurs standardisés sont très pratiques. La Figure 9.10 donne un exemple d'un module Grove typique. À l'heure où j'écris ces lignes, la famille TinkerKit semble perdre un peu en popularité, au profit des modules Grove. Il ne faut pas en conclure que les modules TinkerKit vont disparaître, ni que les modules Grove sont devenus éternels. Dans le monde Arduino, les choses changent vite. Comme déjà dit, ce chapitre ainsi que le précédent ont été créés pour vous fournir des exemples de ce qui peut être disponible ; ce ne sont pas des références exhaustives. Le marché est bien trop dynamique pour pouvoir en donner une image figée.

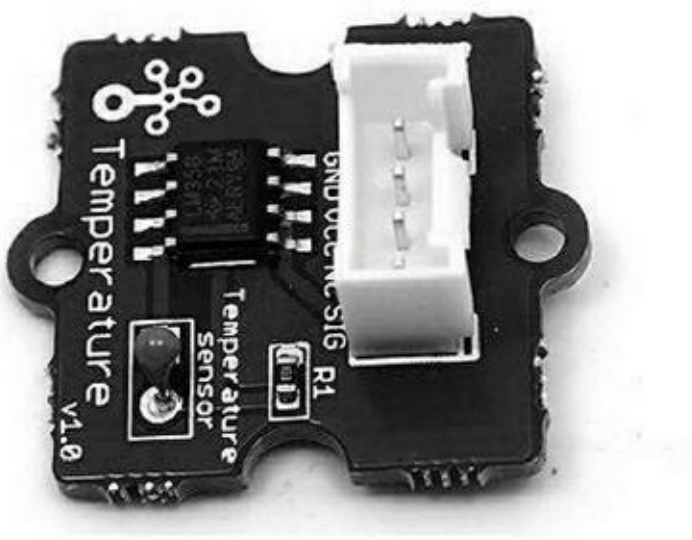


Figure 9.10 : Exemple de module Grove avec ses oreilles.

Si vous avez besoin de modules qui suivent une convention, je vous conseille d'étudier la famille de produits Grove. Lorsque vous aurez besoin de créer un bouclier d'expansion ou d'interface spécifique, vous pourrez toujours créer vos propres câbles à partir des accessoires de connectique. J'aborde la fabrication de connecteurs sur mesure tout à la fin de ce chapitre. Pour une liste à jour des modules Grove, n'hésitez pas à visiter le site Web de Seeed Studio et leur page wiki.

Description des capteurs et des modules

La suite de ce chapitre propose de passer en revue un certain nombre de composants, de modules et de capteurs disponibles pour Arduino. Certains composants deviennent obsolètes, car ils sont remplacés par une version améliorée ; d'autres ont un usage très spécifique, comme les détecteurs de gaz. Tous les éléments mentionnés dans la suite du chapitre sont faciles à trouver avec un moteur de recherche. Vous n'hésiterez pas également à visiter les sites présentés en annexe du livre.

Le Tableau 9.10 dresse la liste des capteurs, actionneurs, afficheurs et autres modules décrits dans la suite, rangés par fonction, par classe (entrée ou sortie) et par type (périphérique, composant ou module). Je rappelle que vous pouvez facilement créer vos modules en utilisant les mêmes composants que ceux présents dans les modules prémontés.

Tableau 9.10 : Index des capteurs et modules présentés.

Fonction	Classe	Type	Description
Affichage	Sortie	Module	Afficheur 7 segments LED
Affichage	Sortie	Module	Afficheur ERM1601SBS-2 1 ligne de 16
Affichage	Sortie	Module	Afficheur ST7066 (HD44780) 2 lignes de 16
Affichage	Sortie	Module	Écran ERC240128SBS-1 240 x128
Affichage	Sortie	Module	Écran TFTST7735R 128 x 160
Audio	Capteur	Microphone	Microphone

Audio	Sortie	Composant	Buzzer piézoélectrique
Champ magnétique	Capteur	Circuit	Capteur de champ magnétique
Champ magnétique	Capteur	Circuit	Magnétomètre
Communications	I/O	Module	Module radio 315/433 MHz RF
Communications	I/O	Module	Module sans fil APC220
Communications	I/O	Module	Module WiFi ESP8266
Communications	I/O	Module	Module NRF24L01
Contact	Capteur	Interrupteur	Interrupteur
Contact	Sortie	Interrupteur	Relais
Contrôle	Entrée	Interface	Pavé numérique
Contrôle	Entrée	Interface	Joystick
Contrôle	Entrée	Interface	Potentiomètre
Distance	Capteur	Module	Émetteur/récepteur laser
Distance	Capteur	Module	Détecteur d’objet à LED
Distance	Capteur	Module	Télémètre à ultrasons
Eau	Capteur	Circuit	Capteur de conductivité de l’eau
Fréquence	Sortie	Module	Générateur d’ondes
IR (détection)	Capteur	IR	Détecteur de proximité
	Capteur	Composant	Photodiode

Lumière (détection)			
Lumière (détection)	Capteur	Composant	Photorésistance
Lumière (détection)	Capteur	Composant	Phototransistor
Lumière (émission)	Sortie	Afficheur	Afficheur 7 segments LED
Lumière (émission)	Sortie	Laser	Diode LED laser
Lumière (émission)	Sortie	LED	LED tricolore RGB
Lumière (émission)	Sortie	Afficheur	Matrice de LED
Lumière (émission)	Sortie	LED	LED monochrome
Lumière (émission)	Sortie	LED	LED bicolore
Moiteur	Capteur	Circuit	Sonde de moiteur (humidité du sol)
Mouvement	Capteur	Circuit	Gyroscope
Mouvement	Capteur	Circuit	Accéléromètre
Mouvement	Sortie	Actionneur	Contrôle de moteur CC
Mouvement	Sortie	Actionneur	Contrôle de servomoteur

Mouvement	Sortie	Actionneur	Contrôle de moteur pas à pas
Pression	Capteur	Circuit	Capteur barométrique
Rotation	Capteur	Interface	Encoder rotatif
Température	Capteur	Circuit	Capteur DS18B20
Température	Capteur	Module	Capteur DHT11/DHT22 temp/humidité
Température	Capteur	Module	Thermistance
Temps	Support	Module	Horloge DS1302
Temps	Support	Module	Horloge DS1307
Temps	Support	Module	Horloge DS3231
Temps	Support	Module	Horloge PCF8563
Tension	Sortie	Module	Convertisseur numérique/analogique
Tilt	Capteur	Interrupteur	Capteur de tilt un axe
Tilt	Capteur	Interrupteur	Capteur de tilt deux axes

Capteurs

Un certain nombre de capteurs peuvent être reliés directement à une carte Arduino, mais il est toujours plus confortable d'utiliser un module, sauf lorsque vous avez besoin de créer un projet très spécifique. Dans ce cas, vous adapterez directement le composant capteur et le placerez à l'endroit désiré.

Par définition, un capteur est un composant d'acquisition de données de l'environnement physique, qui convertit la mesure sous une forme pouvant être exploitée par un microcontrôleur. Comme son nom l'indique, le capteur capte une grandeur physique, qu'il s'agisse d'une température, de l'humidité de l'air, d'un champ magnétique, de la lumière visible ou invisible, d'un son ou d'un déplacement physique.

Capteurs de pression, de température et d'humidité

Vous trouverez un vaste choix de capteurs pour détecter la chaleur, l'humidité ou la pression barométrique.

DS18B20

Le capteur DS18B20 est du type un fil et sert à mesurer une température. Il renvoie un flux de données binaires correspondant à la température mesurée. La Figure 9.11 montre un module doté de ce circuit avec quelques composants passifs et une diode LED.

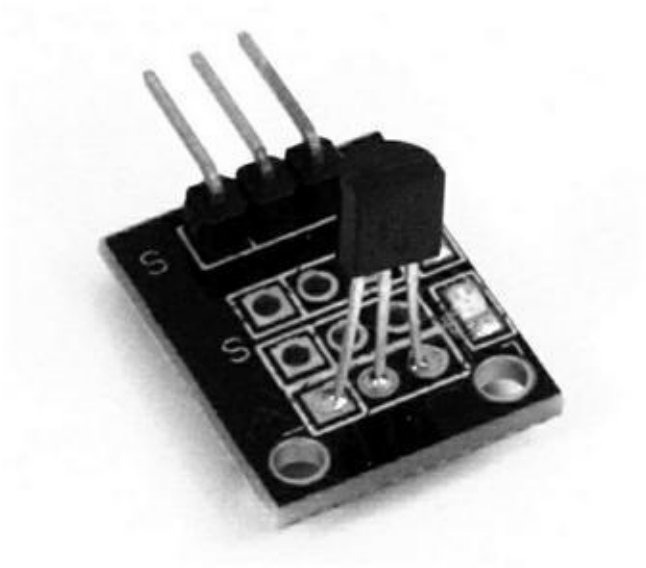


Figure 9.11 : Module capteur de température basé sur le DS18B20.

La Figure 9.12 montre ce module mis en place sur un bouclier de prototypage avec un relais et un potentiomètre. Cela constitue un simple thermostat numérique qui peut venir remplacer un thermostat bilame électromécanique de radiateur. Dans cette version électronique, il devient même possible de collecter les variations de température sur la carte Arduino, parmi d'autres fonctions. Vous voulez un radiateur digne de l'ère de l'Internet des objets ? Pas de problème !

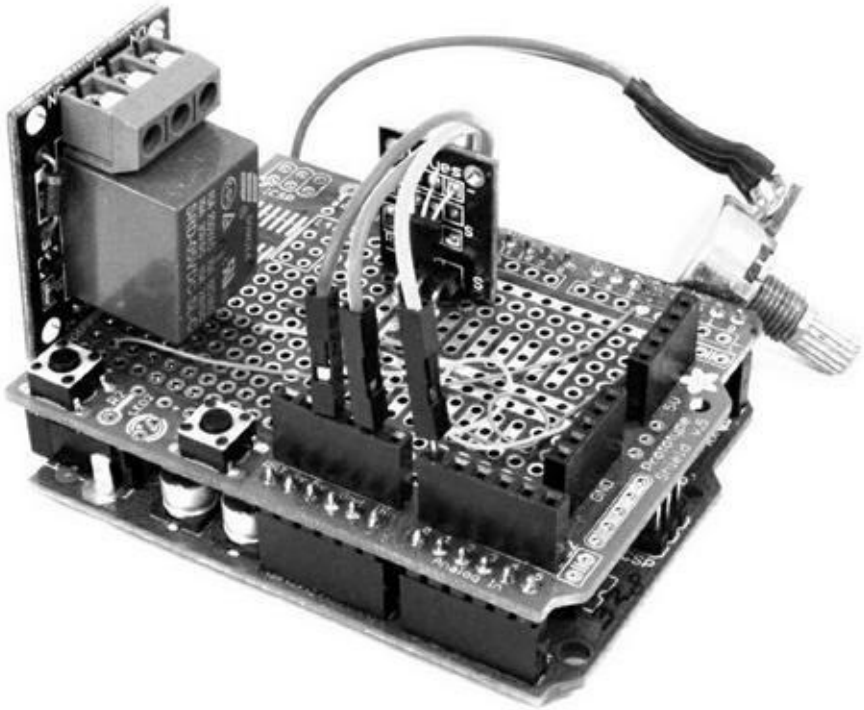


Figure 9.12 : Utilisation du module DS18B20 pour un thermostat.

Capteurs DHT11 et DHT22

Les capteurs DHT11 et DHT22 ont un double effet, puisqu'ils mesurent une température et une humidité de l'air (Figure 9.13). L'aspect qu'ils ont une fois mis en place sur un module est celui de la Figure 9.14.

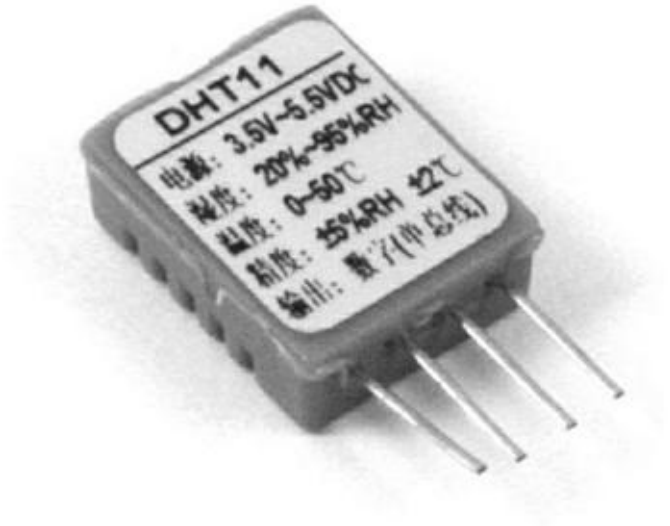


Figure 9.13 : Capteur DHT11.

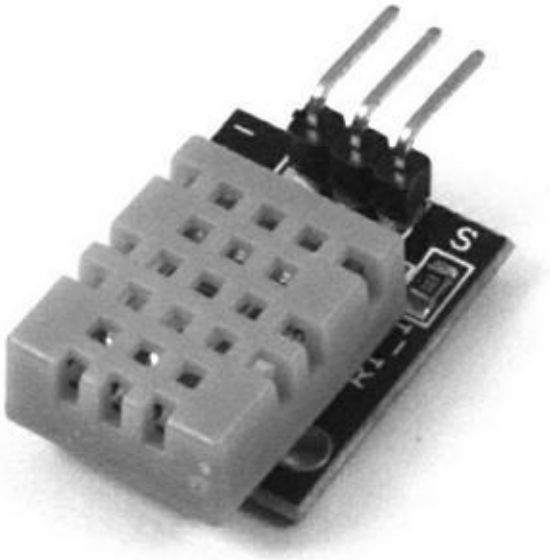


Figure 9.14 : Module basé sur le DHT22.

La différence entre les deux capteurs est la précision de mesure et la vitesse de transfert des données série. Le DHT11 est le modèle économique : sa précision de mesure est de plus ou moins 2 ºC. Sa résolution au niveau humidité est de plus ou moins 5 % d'humidité relative et il fonctionne entre 20 et 90 % d'humidité, et entre 0 et 50 ºC. En revanche, le DHT22 offre une précision de plus ou moins 0,2 ºC et plus ou moins 1 % d'humidité. La plage de travail du

DHT22 est également bien supérieure : il fonctionne de -40 à +80 °C et de 0 à 100 % d'humidité. Le brochage est strictement le même entre les deux.

Les deux capteurs utilisent un protocole de dialogue spécifique sur un fil de type série. Le principe est un signal/réponse. Le microcontrôleur force d'abord la ligne de signal à l'état Bas pendant un bref instant puis la laisse revenir à l'état Haut grâce à la résistance de tirage qu'il faut ajouter sur le circuit ou de façon externe. Cette séquence permet au capteur de répondre en renvoyant 40 bits de données en série sous forme de 5 octets, avec 1 octet de somme de contrôle (*checksum*).

Thermistances

Une thermistance (en anglais, *thermistor*) est une résistance qui varie selon la température, par un coefficient soit positif, soit négatif. Les modèles à coefficient de température négatif, NTC, voient leur résistance diminuer quand la température augmente. C'est l'inverse pour les thermistances à coefficient positif, PTC. Les plus utilisées sont les NTC. Les variantes PTC servent surtout de limiteurs de pointes de courant.

Le branchement d'une thermistance sur une carte Arduino est schématisé dans la Figure 9.15. Je rappelle que la courbe de réponse du composant n'est pas linéaire. Certains montages remplacent la résistance fixe par une source de courant constante.

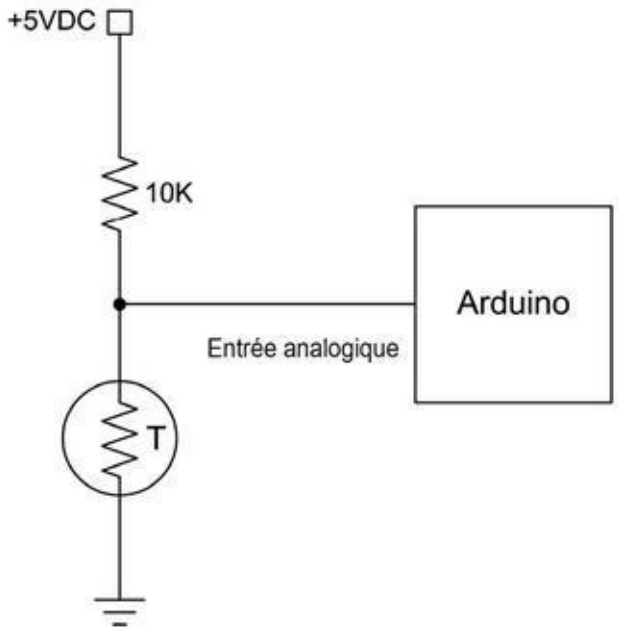


Figure 9.15 : Branchement d’une thermistance à un Arduino.

Il est possible de connecter directement la thermistance à l’Arduino, mais il est plus simple d’utiliser un module, car il sera doté des composants passifs qui sont requis pour créer le diviseur de tension, comme dans le module KY-028. Pour augmenter la sensibilité du montage, vous pouvez utiliser un amplificateur opérationnel comme celui de la Figure 9.26. Du fait que la plupart des thermistances utilisées comme capteurs sont du type négatif, la tension analogique qui va être présentée sur la broche de l’Arduino va décroître quand la température va augmenter.

Détecteurs de pluie et d’eau

Détecter la présence d’eau est très utile, par exemple pour éviter une inondation dans une cave ou détecter l’arrivée de la pluie dans une station météo. Nous avons vu dans les pages précédentes un module SainSmart qui remplit cette fonction, le 20-011-946. Ce capteur utilise un transistor NPN qui force à l’état Bas la sortie dès qu’il y a contact entre les pistes du circuit à cause de la présence d’eau ou de quelque chose de suffisamment humide pour entraîner la conduction du transistor.

Le schéma de ce genre de capteur est donné à la Figure 9.16. Vous constatez que le schéma est très simple et qu'il peut fonctionner avec n'importe quels fils conducteurs ou avec une sonde. Vous pouvez par exemple connecter au circuit deux fils métalliques dénudés en les positionnant afin qu'ils soient à environ 1 cm au-dessus du plancher, ce qui permet de déclencher l'alarme dès que la cave commence à se remplir.

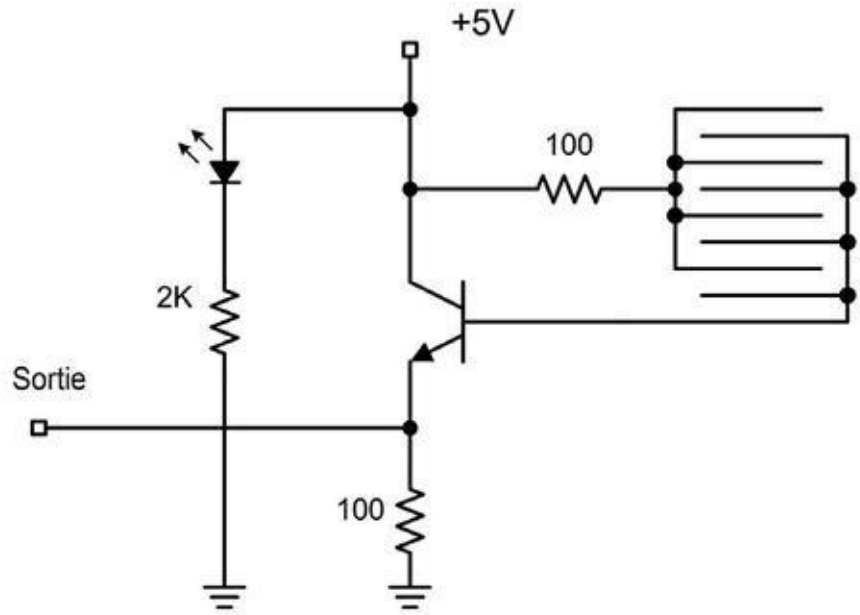


Figure 9.16 : Schéma d'un détecteur d'inondation.

Détecteur d'humidité du sol (de moiteur)

Dans sa version de base, un capteur d'humidité du sol, ou capteur de moiteur, mesure la variation de conductivité entre les deux contacts d'une fourche. Le circuit de mesure est un diviseur de tension, la tension en sortie dépendant de la plus ou moins grande facilité qu'a le courant à passer d'une branche à l'autre. Vous trouverez des modules de capteurs de moiteur (*moisture* en anglais) chez les fournisseurs tels que SainSmart, sous forme d'un kit (Figure 9.17).

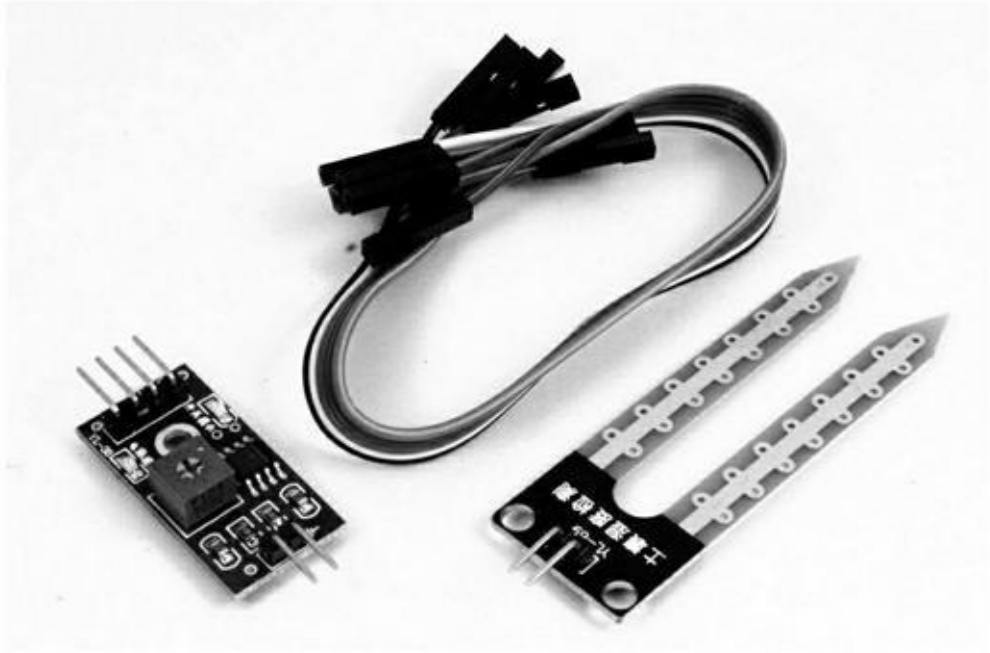


Figure 9.17 : Kit de capteur de moiteur avec interface.

Vous obtenez le même résultat avec n'importe quel matériau conducteur. Pour une utilisation à long terme, mieux vaut utiliser des baguettes de carbone ou de l'acier inoxydable, car la corrosion menace dans un sol humide. Pour éviter de devoir faire passer trop de courant dans la sonde, il est conseillé de prévoir un amplificateur dans le circuit. Notez qu'un courant important a des effets secondaires étonnants, et accélère la corrosion.

Il existe d'autres modèles de capteurs de moiteur, et notamment le Grove 101020008 de Seeed Studio (Figure 9.18). C'est un module monobloc qui comporte un transistor NPN pour amplifier la variation de tension afin qu'elle soit compatible avec le niveau d'entrée du convertisseur analogique/numérique Arduino. Les pistes cuivrées sur les deux branches de la fourche ont reçu un très fin revêtement doré pour mieux résister à la corrosion.

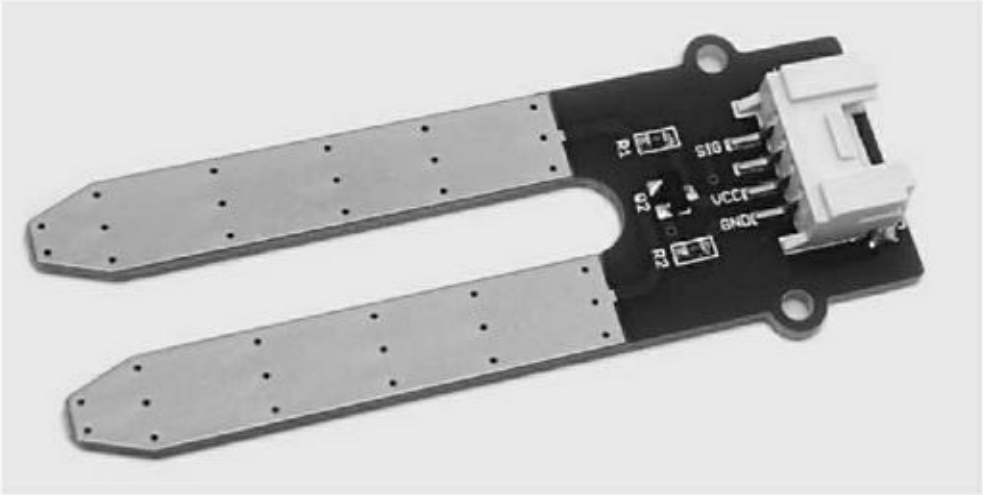


Figure 9.18 : Sonde de moiteur monobloc.

La Figure 9.19 montre le schéma très simple de cette sonde. C'est quasiment le même que celui du détecteur d'eau de la Figure 9.16. Remarquez que ce module utilise un connecteur à quatre broches, au lieu des quatre broches nues de la sonde précédente (Figure 9.17). Cette approche est beaucoup plus confortable, mais il n'est pas inutile de vérifier à quoi correspond chaque broche.

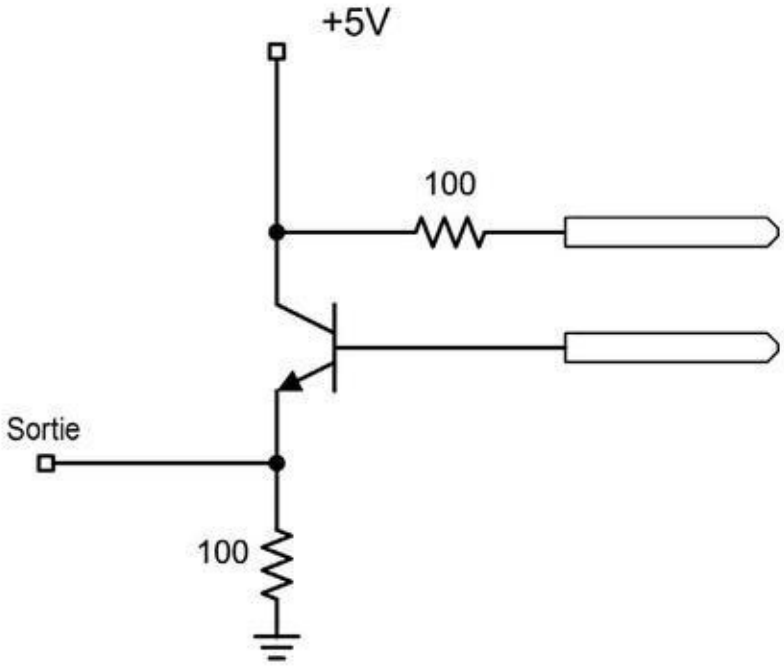


Figure 9.19 : Schéma de la sonde de moiteur monobloc.

Capteurs barométriques

Un capteur barométrique (Figure 9.20) permet de mesurer la pression atmosphérique. Vous pouvez le combiner avec un module DHT11 ou DHT22 (décrits quelques pages auparavant) pour constituer très simplement une petite station météo. Le capteur montré dans la figure qui suit est basé sur le circuit MPL115A2 qui exploite une interface I2C.

Figure 9.20 : Module capteur barométrique.

Ce circuit ne permet pas d'atteindre une précision suffisante pour s'en servir en tant qu'altimètre, mais il suffit à collecter des données météo. Pour une précision supérieure, il faut se tourner vers les circuits MPL3115A2 ou BMP085 qui peuvent servir d'altimètre.

Détecteurs de choc et de tilt

L'expression « tilt » est connue des joueurs de billard mécanique ou flipper. Un capteur de tilt est en fait une capsule étanche contenant deux contacts qui peuvent être réunis par une bille métallique ou par une goutte de mercure. L'élément mobile ferme le circuit entre les contacts lorsque le capteur est déplacé de son orientation neutre, c'est-à-dire à angle droit par rapport à la force de gravité. Le résultat est le même que lorsque l'on ferme un interrupteur.

Ce genre de capteur n'est pas proportionnel. Soit il y a contact, soit il n'y a pas contact.

Capteurs d’inclinaison mono-axe

La Figure 9.21 montre un capteur d’inclinaison à goutte de mercure, le module KY-017. Il ne détecte qu’un changement d’inclinaison dans une seule direction. Il en faut donc deux ou plus pour détecter par exemple une inclinaison vers la gauche ou vers la droite.

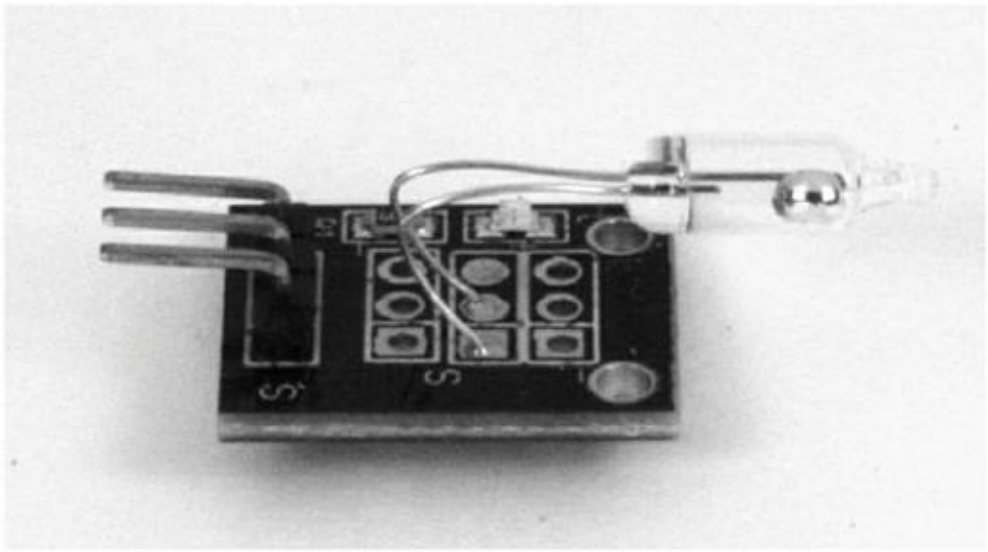


Figure 9.21 : Capteur d’inclinaison mono-axe.

Capteurs deux axes

Pour utiliser un capteur d’inclinaison, il faut d’abord déterminer où se trouve la position neutre selon un axe. Le capteur peut ensuite être positionné pour détecter une inclinaison dans un sens de cet axe. Dans la Figure 9.22, vous pouvez voir deux capteurs installés à angle droit. Notez qu’ils ne détecteront qu’une inclinaison vers le haut et une vers la gauche, si on considère une vue du dessus dans la figure.

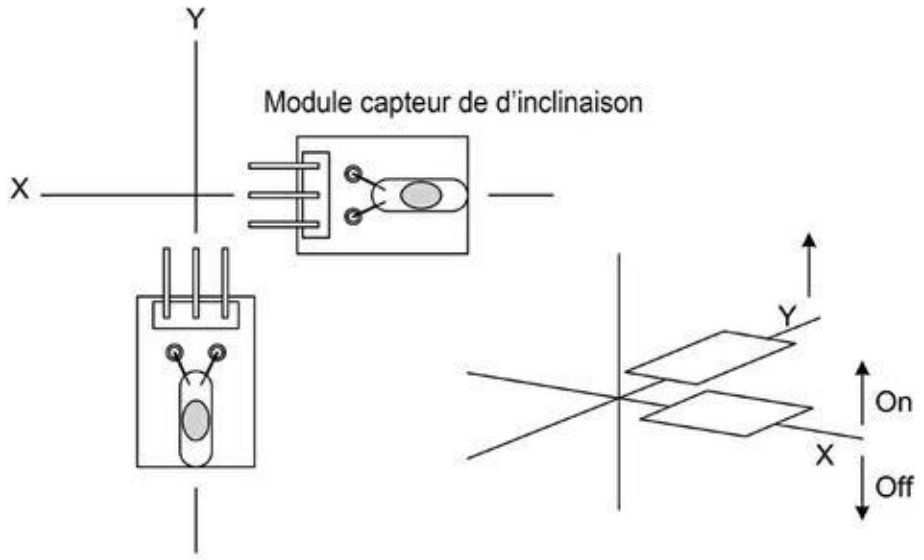


Figure 9.22 : Deux capteurs d'inclinaison à angle droit.

Pour détecter l'inclinaison dans les deux sens sur deux axes, il faut quatre capteurs en croix. Ce genre de capteur offre deux avantages par rapport à un véritable gyroscope que je décris un peu plus loin : il n'a pas besoin d'une référence de départ et il fonctionne tant que la gravité existe. Le seul souci est qu'il fonctionne en tout ou rien.

Détecteurs audio

Un microphone peut servir à autre chose qu'à enregistrer des sons. Vous pouvez par exemple le régler pour qu'il se déclenche dans un système de sécurité seulement en cas de bruit fort tel qu'un bris de glace, un enfoncement de porte ou le son d'un coup de fusil. Bien sûr, s'il est suffisamment sensible, il pourra même détecter des pas humains.

Un microphone de contact permet de recueillir des données pour diagnostiquer l'état d'usure d'un moteur à combustion ou électrique. Cela permet d'auditionner les roulements et les composants qui commencent à prendre trop de jeu. En combinant un microphone omnidirectionnel et un détecteur de luminosité, vous pouvez construire un détecteur de distance d'orage, puisque le son arrive une seconde plus tard que l'éclair pour 300 m d'éloignement.

Vous trouverez de tout petits modules combinant un microphone et un petit circuit, comme celui de la Figure 9.23. Le potentiomètre de réglage permet de régler le seuil de déclenchement. Le module utilise un circuit similaire à celui de la Figure 9.8.

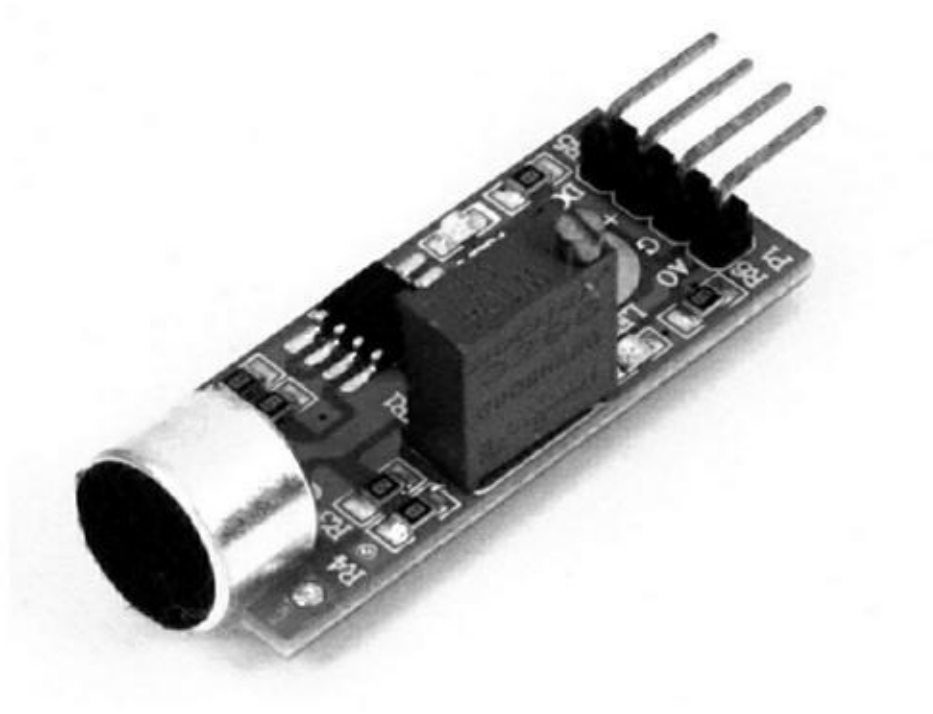


Figure 9.23 : Module de détection audio.

Vous pouvez également brancher directement un microphone sur une entrée analogique de l'Arduino, mais vous ne pourrez pas contrôler le volume d'entrée. Pour augmenter la sensibilité, il suffit d'ajouter un circuit amplificateur opérationnel tel que celui de la Figure 9.26.

Capteurs de lumière

La détection des ondes lumineuses regroupe plusieurs types de capteurs. Certains peuvent détecter de la chaleur, qui émet toujours un rayonnement infrarouge. D'autres capteurs détectent des flammes, et d'autres bien sûr la lumière visible. La détection peut utiliser une résistance qui varie en fonction de la quantité de lumière qui l'illumine ou un semi-conducteur qui offre une meilleure sensibilité et une réponse plus rapide.

Photorésistances (LDR)

Comme son nom l'indique, une photorésistance est un composant dont la résistance varie en fonction de la quantité de lumière qui la touche. Vous trouverez également le sigle LDR, qui signifie *Light Dependent Resistor*. La plupart de ces composants ressemblent à ce que montre la Figure 9.24. Ils réagissent bien moins vite que les composants basés sur une diode ou un transistor, mais cela suffit à créer par exemple un jeu de lumière piloté par de la musique. Les photorésistances sont très peu coûteuses et suffisent pour créer des détecteurs de niveau de luminosité ambiante, établir un lien optique par impulsion à faible vitesse, détecter des balises par contraste en robotique, par exemple pour retrouver la station d'accueil ou encore effectuer un suivi du soleil pour des panneaux solaires.

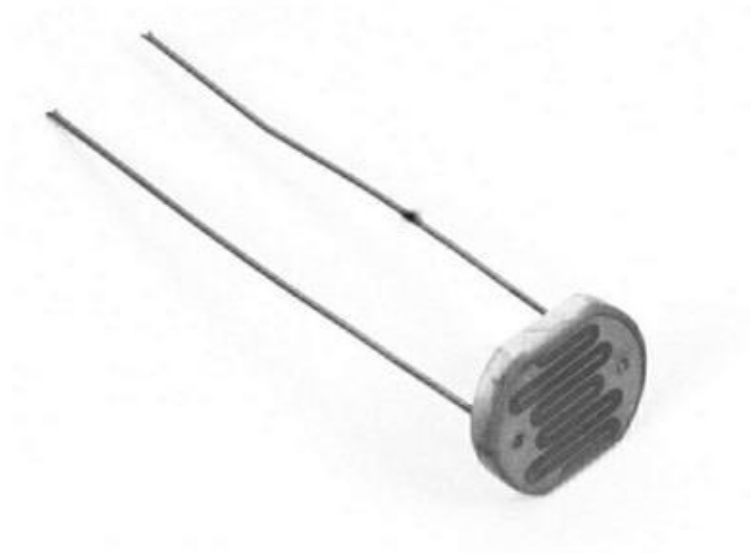


Figure 9.24 : Photorésistance LDR très économique.

Photodiodes

Quasiment toutes les diodes LED ont une certaine sensibilité à la lumière, mais une photodiode est fabriquée afin d'amplifier cet effet secondaire. Ce genre de diode devient passante quand elle éclairée. La technologie la plus répandue est la diode PIN, le I symbolisant la couche intermédiaire de matériau silicium Intrinsèque, en sandwich entre une partie positive et une partie négative. C'est cette couche centrale qui permet à la diode de bien réagir à la lumière. Le temps de réponse est très rapide, ce qui rend ces diodes indispensables pour les liaisons

optiques et les capteurs de position d'éléments en rotation. Vous trouverez des diodes PIN dans les circuits radio à haute fréquence. Elles sont utilisées en tant qu'interrupteurs très rapides. Pour en savoir plus, voyez les livres présentés dans l'Annexe D.

La Figure 9.25 montre comment utiliser une photodiode avec une carte Arduino. Notez bien que la diode est montée à l'envers. Elle ne conduit pas le courant tant qu'elle n'est pas exposée à la lumière. Ce circuit simple a du mal à réagir à une faible luminosité.

Une solution simple pour améliorer la sensibilité consiste à ajouter un amplificateur opérationnel (Figure 9.26). Dans l'exemple, le circuit est un LM358 qui offre un gain d'environ 10. Autrement dit, il décuple la faible tension de conduction de la diode afin qu'elle puisse être convertie par le circuit d'entrée Arduino. Un potentiomètre permet de régler le gain selon les besoins de l'application. Vous pouvez facilement monter ce circuit sur une petite plaque d'expérimentation sans soudure ou bien vous lancer dans la création d'un circuit soudé, si le projet est destiné à perdurer.

Vous trouverez des photodiodes et des amplis op chez tous les fournisseurs, et notamment ceux présentés en annexe.

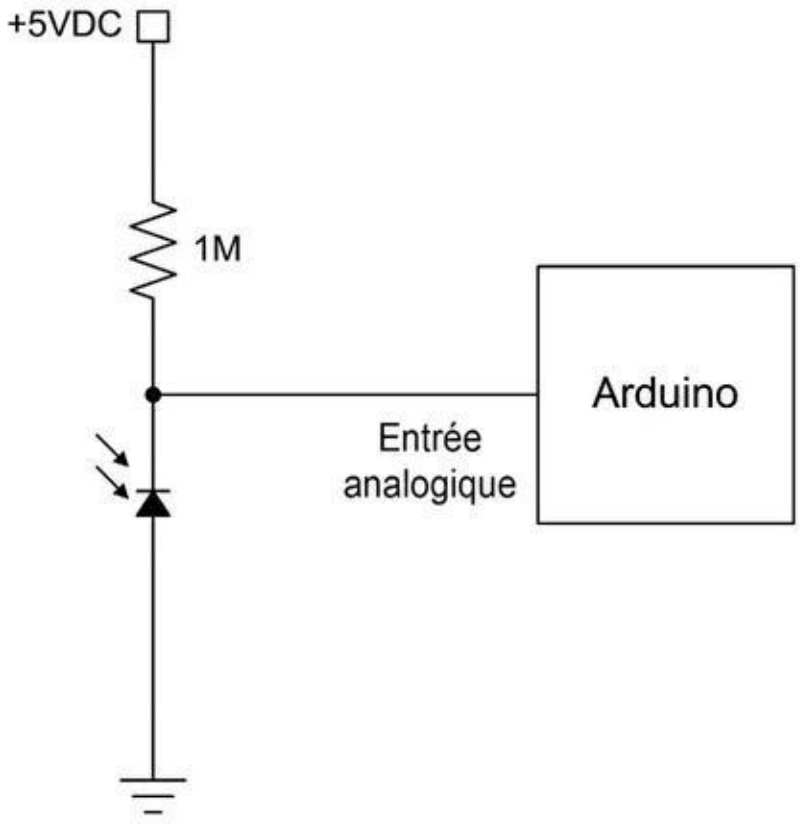


Figure 9.25 : Schéma d’une photodiode reliée à un Arduino.

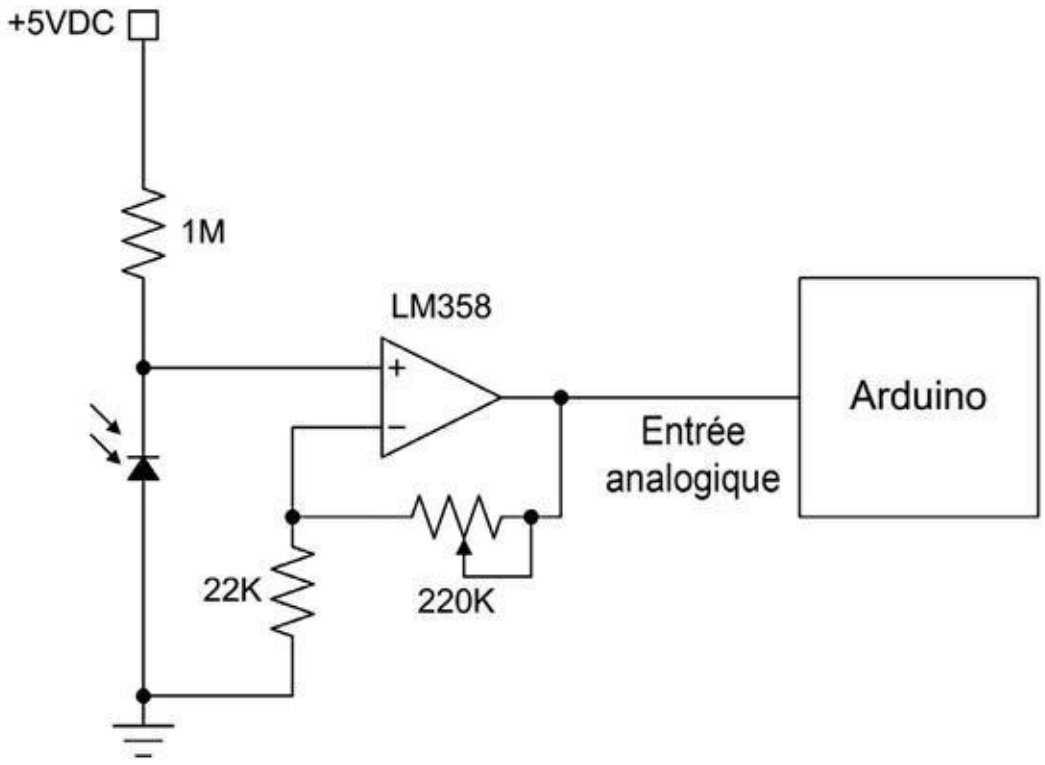


Figure 9.26 : Ajout d'un ampli op à un capteur à photodiode.

Phototransistors

Un phototransistor permet d'obtenir une tension variable en fonction de la quantité de lumière qui l'illumine. Dans le schéma de la Figure 9.27, vous pouvez constater qu'il n'y a pas de broche pour la base du transistor, puisque c'est la lumière qui remplit ce rôle de commande. Le circuit de la Figure 9.8 peut être utilisé avec un tel phototransistor. Certains modules très simples, et notamment le KY-039, ne font en fait que ramener les deux broches du transistor sur les broches de connexion du module. Vous pouvez même vous servir du module comme phototransistor en coupant les fils de la diode LED.

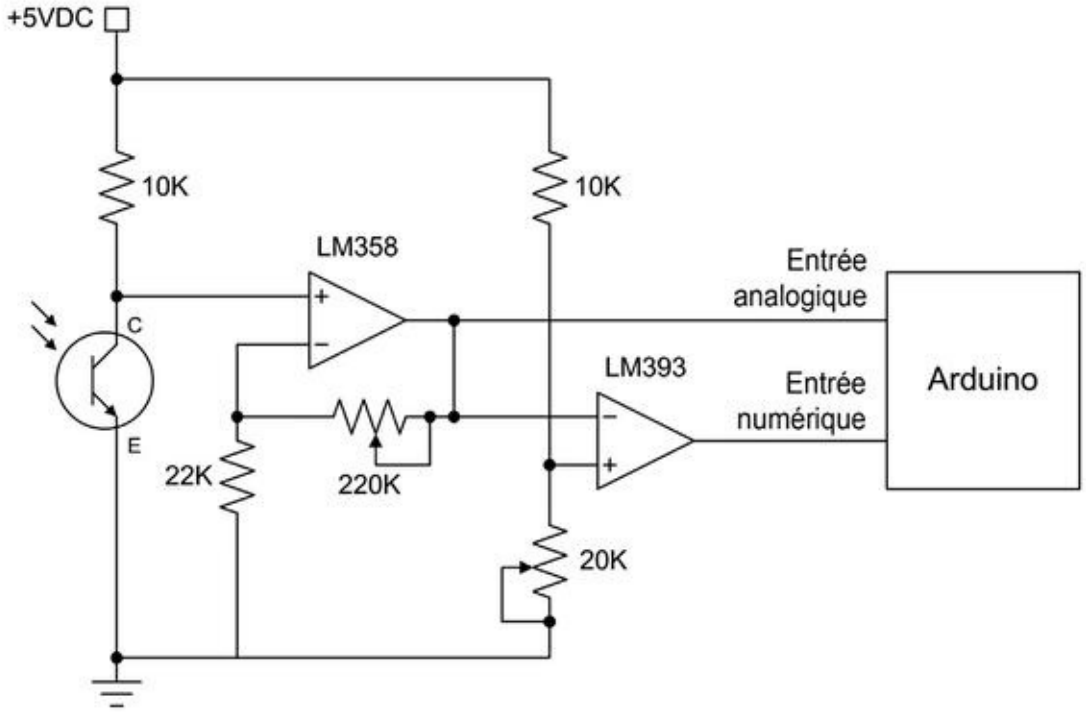


Figure 9.27 : Schéma d'un circuit de phototransistor avec sorties analogique et numérique.

Pour obtenir une sensibilité suffisante, il est conseillé d'ajouter un deuxième ampli op comme proposé dans la Figure 9.27. N'importe quel phototransistor NPN conviendra, mais j'ai une préférence pour le BFH310, tout simplement parce que j'ai pu en acheter un sachet complet.

Les modules interrupteurs optiques comme le KY-010 utilisent un phototransistor pour retraiter la sortie d'une diode LED. Dès qu'un obstacle empêche le passage du rayon lumineux, le transistor arrête de conduire. De même, les isolateurs optiques ou optocoupleurs utilisent ce couple diode LED/phototransistor pour faire dépendre le signal d'un circuit de celui d'un autre sans qu'il y ait de connexion électrique entre les deux. Un tel coupleur est facile à construire soi-même en combinant une diode LED émettrice, un phototransistor et un bout de tube pour empêcher la lumière ambiante de perturber le circuit.

Capteurs infrarouges passifs (PIR)

L’acronyme PIR signifie Passif InfraRouge. Un tel capteur sert à connaître la quantité de lumière infrarouge qui l’éclaire. Il est très utilisé dans les systèmes d’alarme, car il est suffisamment sensible au rayonnement que l’on peut rencontrer dans la gamme infrarouge des domiciles. Dès que le niveau d’infrarouge change par rapport à la référence, par exemple lorsqu’un humain entre dans la pièce, le capteur se déclenche. Ce genre de capteur permet également d’avoir une mesure approximative de la température (Figure 9.28).

Figure 9.28 : Capteur infrarouge PIR.

Vous trouverez ce genre de modules pour moins de 2 euros. Il se connecte par trois broches, dont deux pour l’alimentation. La broche de sortie passe à l’état Haut lorsque le capteur a détecté un changement dans le niveau de rayonnement ambiant. Vous pouvez combiner ce capteur avec un capteur audio (Figure 9.23) pour créer un petit système de sécurité avec le programme présenté dans le Listing 5.5 du Chapitre 5. Vous avez ainsi un système d’alarme simplifié.

Capteurs magnétiques

La détection des champs magnétiques statiques est un domaine dans lequel la technologie des semi-conducteurs a fait de gros progrès. En effet, il est relativement simple de détecter un champ magnétique variable, comme celui d’une bobine, puisqu’il suffit d’une autre bobine. En revanche, un champ fixe, comme celui d’un aimant permanent ou le champ magnétique terrestre, est moins simple à détecter. Avant l’apparition des nouveaux composants

électroniques, les capteurs de champ utilisaient des aimants, des bobines et des miroirs. Une simple boussole de randonnée suffit pour partir en expédition ou pour détecter le champ magnétique produit par un fil dans lequel passe du courant continu, mais il n'est pas très simple d'effectuer une collecte de données à partir d'un tel appareil. De nos jours, vous pouvez construire un détecteur de champ magnétique ou un compas électronique sans aucune pièce mobile, et l'interfacer directement avec un microcontrôleur.

Capteurs à effet Hall

L'effet Hall permet de détecter la présence d'un champ magnétique. Il existe des modèles de type tout ou rien, comme le module KY-003 qui se base sur le circuit A3144. Cela vous permet de savoir si le champ magnétique est présent ou pas. Une autre catégorie est celle des capteurs linéaires comme le SS49E dont la sortie analogique est proportionnelle à la force du champ magnétique. C'est le circuit utilisé dans le module KY-024.

Les deux circuits A3144 et SS49E, ressemblent à de petits transistors en plastique. L'A3144 et les circuits similaires peuvent directement être reliés à l'Arduino, ce que fait d'ailleurs le module KY-003. Le module KY-024 a besoin d'un circuit comparateur comme celui montré dans la Figure 9.8.

Magnétomètres

Une autre technologie pour détecter un champ magnétique consiste à utiliser un magnétomètre comme celui de la Figure 9.29. Le modèle présenté est d'Adafruit ; il utilise un circuit magnétomètre sur trois axes HMC5883L et fonctionne avec l'interface I2C.

Figure 9.29 : Module de compas magnétomètre.

Capteurs de vibration et de choc

La détection de vibration et de choc se fonde en général sur la détection des mouvements d'une masse qui peut être un bras mécanique lesté avec des contacts positionnés afin que le bras ferme un circuit lorsqu'il est déplacé. Une variante consiste à faire interrompre un rayon lumineux par le bras. D'autres techniques se fondent sur une masse retenue par un ressort. Il existe même des capteurs de choc optiques et magnétiques. Certains modèles enfin utilisent les propriétés d'un matériau piézo-électrique pour détecter le mouvement de la masse qui déforme le cristal, ce qui produit une petite tension.

Les capteurs de vibration les plus abordables contiennent une petite masse conductrice dans un boîtier étanche. Lorsque la masse se déplace, elle crée un contact entre des feuilles métalliques à chaque extrémité de son logement. Le module KY-020 utilise cette technologie.

Un capteur de choc ressemble à un capteur de vibration, sauf qu'il est souvent étalonné pour répondre à un certain niveau d'accélération, exprimé en multiples du *g*, sachant que 1 *g* est la force de gravité de la terre au niveau de la mer. Un module qui utilise ce type de technique est le KY-031, lui aussi très économique.

Vous pouvez facilement créer votre propre capteur de choc avec une bille de métal, un ressort, par exemple récupéré d'un stylo à bille, un petit bout de tube en plastique et du fil fin. La Figure 9.30 montre comment fabriquer un tel capteur soi-même.

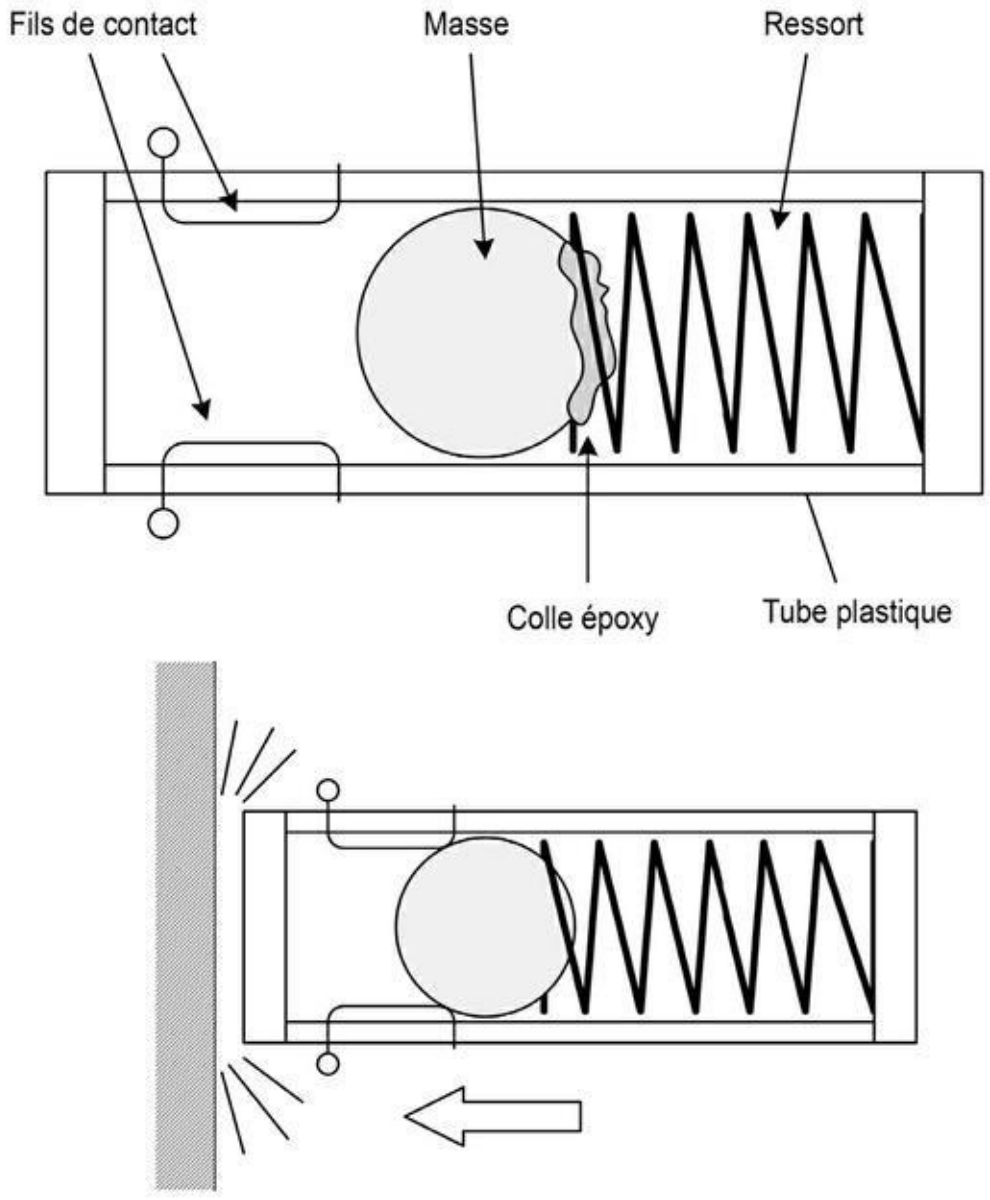


Figure 9.30 : Capteur de choc fait maison.

Pour plus de précisions, vous trouverez des capteurs de choc industriels, calibrés selon la force à détecter. Ils sont très utilisés dans les systèmes de sécurité passive des automobiles et pour l'expédition d'appareils fragiles, afin de garantir

qu’ils n’ont pas subi de chocs excessifs. Bien sûr, ce genre de composants n’est pas donné.

Capteurs de mouvements

Pour garantir la stabilité d’un objet dans l’espace à trois dimensions, il faut pouvoir détecter un changement d’angle et mesurer la vitesse de ce changement. La plupart des drones volants utilisent ce genre de capteurs multiaxes, ce qui revient à simuler un véritable gyroscope ou une unité inertielle IMU (*Inertial Management Unit*). Ce genre de capteur est également très utilisé dans le monde de l’aéromodélisme et dans les modèles réduits de fusées pour collecter les données de la trajectoire de vol.

Les techniques de fabrication et les avancées technologiques ont entraîné une baisse stupéfiante du prix de ce genre de composants. Un accéléromètre ou un gyroscope électronique qui coûtait largement plus de 50 euros peut dorénavant se trouver pour moins de 10 euros. De plus, ces composants sont de toutes petites puces montées en surface, donc difficiles à souder soi-même. Il est préférable d’acheter un module, sauf si vous devez utiliser ce genre de composants pour une fabrication en série, et avez la possibilité d’effectuer la soudure de composants SMC (*Surface Mounted Component*).

Gyroscopes

Le terme gyroscope est quelque peu abusif quand on veut désigner par ce nom les capteurs électroniques de cette catégorie. En effet, à la différence des vrais gyroscopes mécaniques, les capteurs que nous allons découvrir sont en fait des capteurs de taux de vitesse angulaire par rapport à des axes. Ces appareils ne font pas référence à une position initiale maintenue par l’inertie comme dans un vrai gyroscope. Ce que l’on appelle centrale inertielle ou IMU est capable de fonctionner comme un vrai gyroscope, mais c’est un appareil lourd contenant au moins trois gyroscopes tournant à haute vitesse sur des roulements de grande précision et entraînés par des moteurs électriques. Les mesures sont faites grâce à des capteurs de position ; ce sont des appareillages très coûteux. Il est heureusement possible de simuler le fonctionnement d’une centrale inertielle avec des accéléromètres multiaxes et une bonne programmation. La Figure 9.31 montre par exemple un « gyroscope » trois axes.

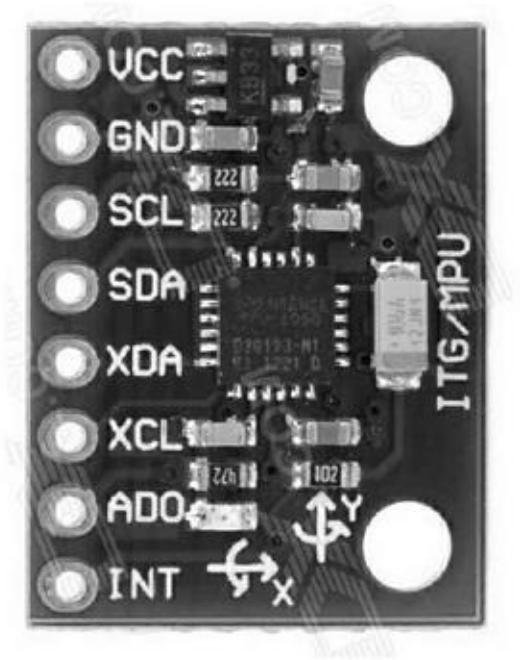


Figure 9.31 : Module gyroscope trois axes.

Accéléromètres

Un accéléromètre est capable de détecter un changement de vitesse selon un axe linéaire. Si la vitesse ne change pas, il n'y a rien à détecter. Cet appareil permet donc de mesurer une accélération ou une décélération. Un accéléromètre trois axes permet de détecter tout changement selon les axes X, Y et Z. La Figure 9.32 montre un module accéléromètre à un axe basé sur le circuit MMA7361.

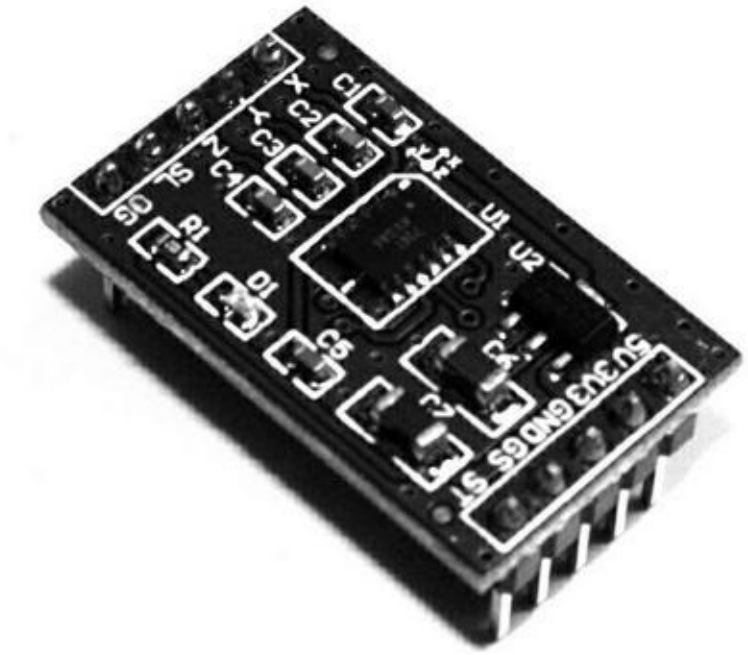


Figure 9.32 : Module accéléromètre.

Capteurs de contact et de position

Les capteurs de contact, également appelés fins de course, sont omniprésents dans l'industrie, qu'il s'agisse de l'embouteillage ou des machines-outils. Les gros boutons en caoutchouc des boîtes à rythmes des musiciens sont des capteurs de contact eux aussi. Tous ces capteurs fonctionnent sur le même principe : soit ils sont en contact avec quelque chose, soit ils ne le sont pas.

Un capteur de position permet de savoir à quelle position se trouve l'objet désiré. Il est plus précis qu'un capteur de contact, puisqu'il peut informer sur la distance en ligne droite ou sur l'angle de rotation. Certains de ces capteurs fonctionnent avec une lumière renvoyée, d'autres utilisent le son et d'autres encore utilisent un disque et un faisceau lumineux pour mesurer un angle. Il existe également des encodeurs de position absolue qui comportent un disque en verre strié de très fines marques pour une mesure précise d'angle de rotation d'un axe. Je ne présenterai pas davantage ce type d'encodeurs.

Détecteurs de contact

Un détecteur ou interrupteur de contact peut se résumer à un ressort installé autour d'une broche métallique, et c'est le genre de détecteur que l'on trouve sur les petits robots vendus comme jouets. Lorsque la moustache du ressort est pliée par un obstacle, le courant passe entre la broche centrale et les spires du ressort. C'est donc un interrupteur tout ou rien. La Figure 9.33 montre comment est construit ce genre de détecteur.

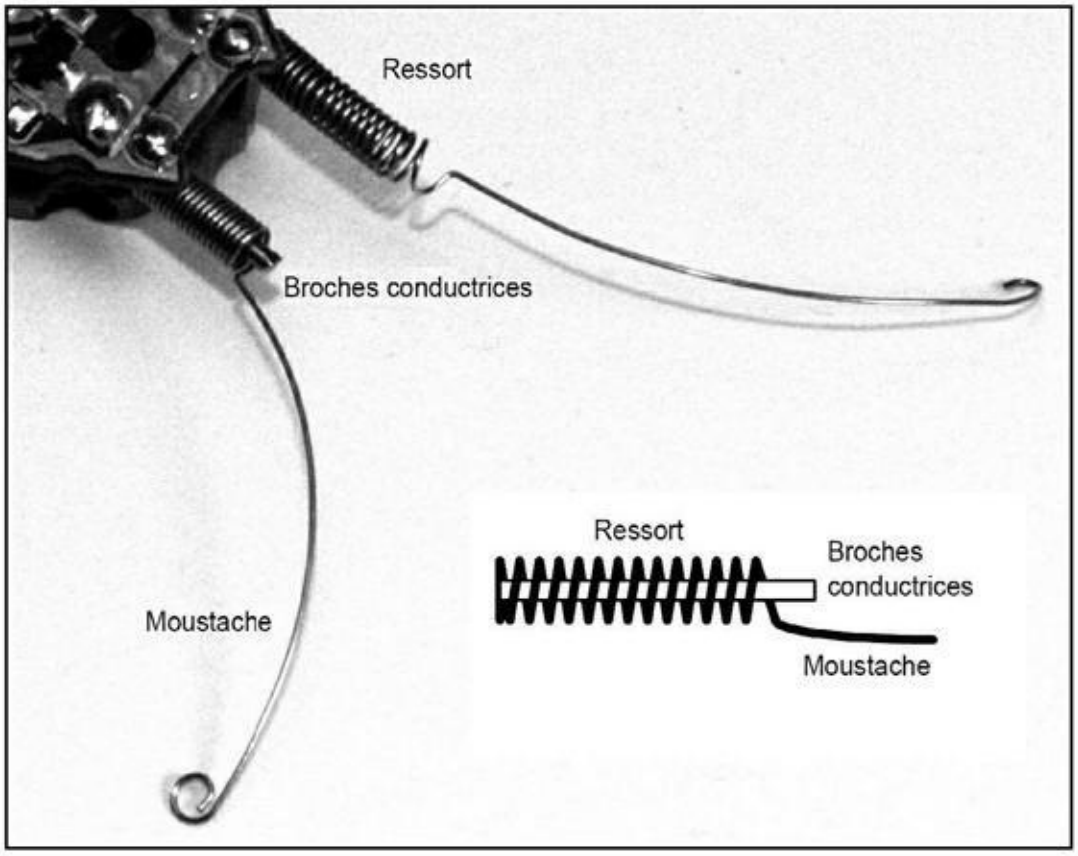


Figure 9.33 : Capteur de contact à ressort.

Un autre type de détecteur de contact est le contacteur à languette (Figure 9.34). Il s'agit ici d'un module Meeno. Ce genre d'interrupteur est souvent utilisé comme fin de course en robotique et dans la machinerie industrielle.

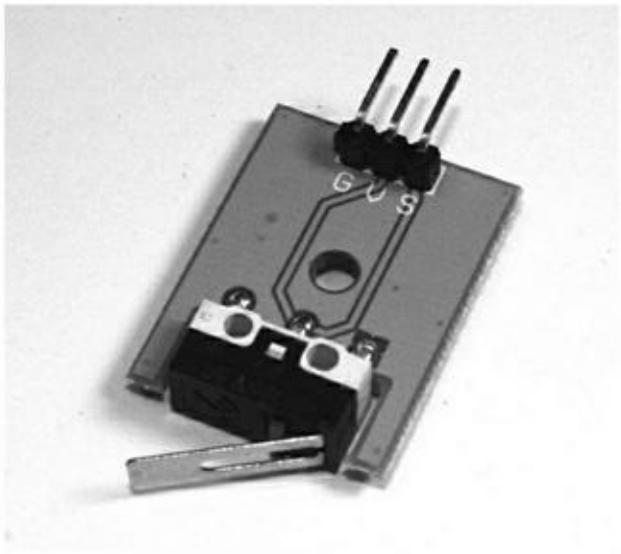


Figure 9.34 : Module contacteur à languette.

Un simple bouton poussoir peut également servir de détecteur de contact. Le contacteur qui allume la lumière quand vous ouvrez la porte d'une automobile en est un excellent exemple. Cela dit, la force nécessaire pour enfoncer ce genre de bouton est généralement bien supérieure à celle que l'on pourra faire déployer par un robot. Vous pouvez utiliser une bande de cuivre et une vis pour établir le contact, comme le montre la Figure 9.35. L'objectif est toujours le même : ouvrir ou fermer un circuit afin que la carte Arduino puisse détecter ce changement d'état.

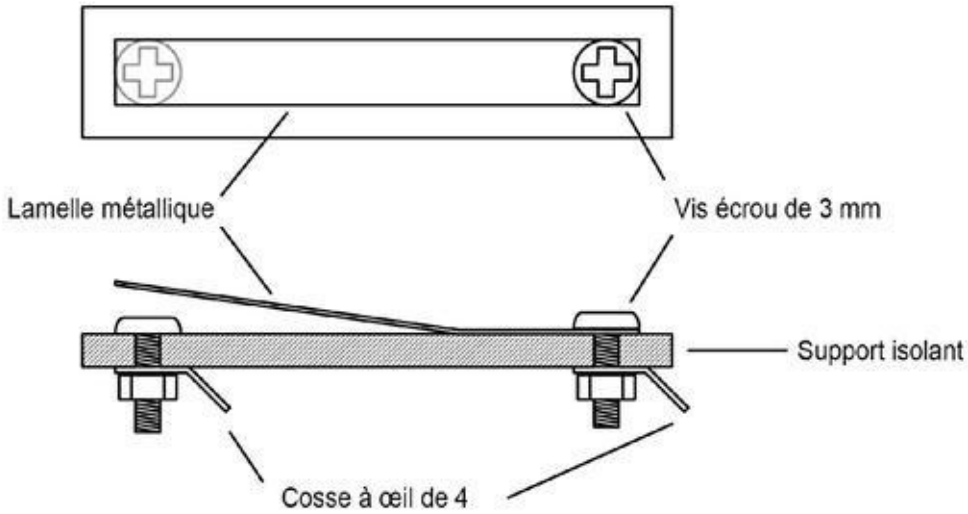


Figure 9.35 : Détecteur de contact fait maison avec une lamelle métallique.

Encodeurs rotatifs

Un encodeur rotatif numérique (Figure 9.36) génère des impulsions ou bien émet une valeur numérique au fur et à mesure de la rotation de l'axe. C'est ce genre d'encodeur qu'utilise le module KY-040. Certains encodeurs comportent des crans qui permettent de se repérer parmi les différents angles de rotation, et d'autres sont sans butée, c'est-à-dire que vous pouvez faire tourner sans cesse dans le même sens. Dans le Chapitre 12, j'utilise un encodeur rotatif du type KY-040 pour un projet.

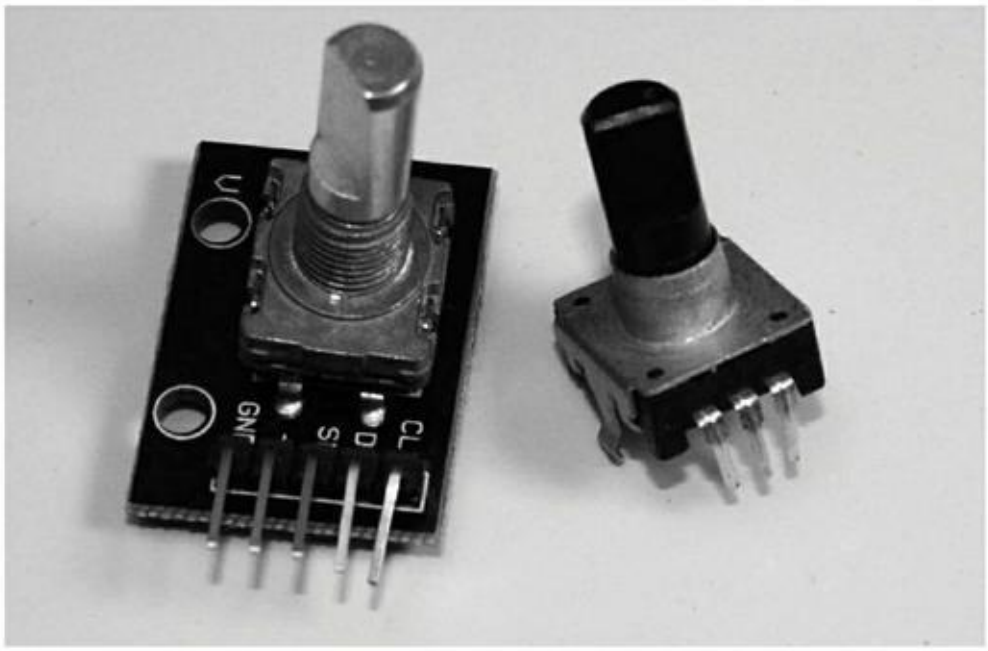


Figure 9.36 : Deux encodeurs rotatifs numériques.

Les anciennes souris informatiques fonctionnaient avec une bille recouverte de caoutchouc et non une diode LED. Deux encodeurs rotatifs servaient à détecter la rotation de la bille. Les surplus d'électronique regorgent de ces anciens modèles de souris, et il n'est pas inintéressant d'en récupérer pour les démonter. Si vous ouvrez ce genre de souris, vous pouvez voir des roues crantées en plastique. Quand la roue tourne, le rayon d'une diode LED est interrompu. Il suffit de compter la durée des impulsions pour générer des valeurs numériques proportionnelles à la quantité de déplacement de la souris dans les deux axes X et Y. La Figure 9.37 montre une telle souris ventre ouvert.

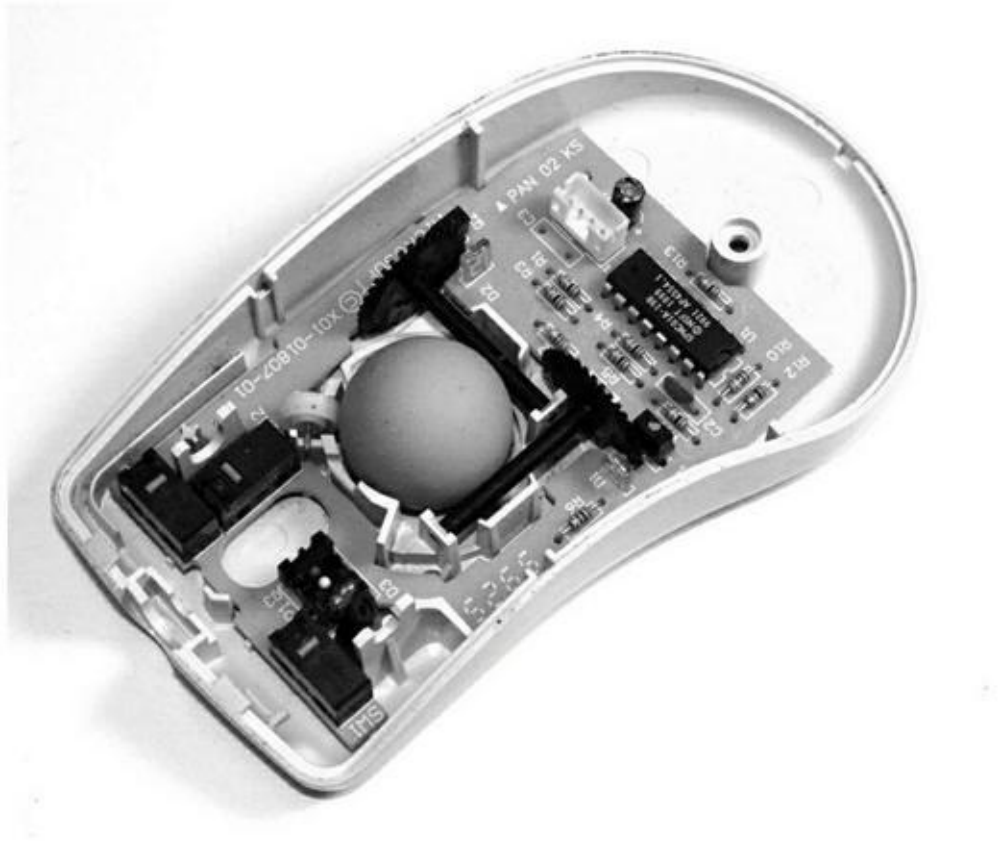


Figure 9.37 : Une ancienne souris à boule avec encodeurs rotatifs.

Dans le modèle de la figure précédente, la diode LED émettrice et le phototransistor récepteur sont séparés, mais dans d'autres modèles, c'est un composant unique, identique à celui du module KY-010 vu dans la liste des capteurs Keyes. Les roues codeuses sont entraînées par des axes qui sont au contact de la bille. Si vous avez utilisé une souris de ce type, vous savez qu'il faut de temps à autre la démonter pour nettoyer la bille et enlever la poussière accumulée sur les axes. Vous pouvez démonter les roues codeuses et le capteur optique pour les réutiliser dans vos projets. Notez que ce sont les roues codeuses qui sont les plus intéressantes à récupérer dans une telle souris.

Émetteurs/récepteurs laser

Le module de la Figure 9.38 est un émetteur et récepteur laser à courte portée qui sert à détecter des obstacles. Il fonctionne en mesurant la réflexion du rayon sur l'obstacle. On pourrait s'en resservir pour établir une liaison de données optique, mais il faudrait augmenter la puissance d'émission pour atteindre une portée digne d'intérêt.

Figure 9.38 : Détecteur d’obstacle laser à courte portée.

Détecteurs de distance (télémétrie)

La robotique a énormément besoin de capteurs de distance. La plupart de ces composants mesurent un rayon renvoyé, qu’il s’agisse de lumière, d’une onde sonore ou radio, comme dans les radars. Dans cette section, je présente les capteurs utilisant la lumière et le son. Les radars sont beaucoup plus coûteux et sont destinés à fonctionner sur de longues distances. Un capteur optique ou acoustique suffit largement à courte portée, et ce sont des objets très économiques.

Capteurs de distance à diode LED

Ce genre de capteur mesure la lumière émise par une LED telle qu’elle est renvoyée par une surface, qu’il s’agisse d’une diode à lumière visible ou infrarouge. Le composant réunit la diode d’émission et le détecteur côte à côte, comme en Figure 9.39. Il n’y a pas de mesure du temps mis pour que le rayon revienne, car la vitesse de la lumière à cette distance serait très difficile à mesurer. Pour mesurer la distance entre la terre et la lune, il faudrait un laser à impulsion et un gros télescope avec un chronomètre ultra précis. Pour vos projets Arduino, nos capteurs suffiront à éviter à un robot de percuter un mur, ou lui permettront de suivre une ligne tracée sur le sol.

Télémètres à ultrasons

Vous savez sans doute comment font les chauves-souris pour se repérer dans le noir ? Elles émettent et reçoivent des ultrasons renvoyés par les obstacles. Le télémètre à ultrasons est constitué de deux modules piézoélectriques, le premier étant câblé comme émetteur et l'autre comme récepteur. L'émetteur génère une courte impulsion d'ultrasons et le récepteur récupère l'écho renvoyé. Il suffit de diviser par deux le délai entre l'émission et la réception pour connaître la distance. Cette mesure est possible parce que la fréquence des ultrasons est bien inférieure à celle de la lumière. Il n'est donc pas très difficile de réussir à distinguer les deux moments, si le programme va suffisamment vite.

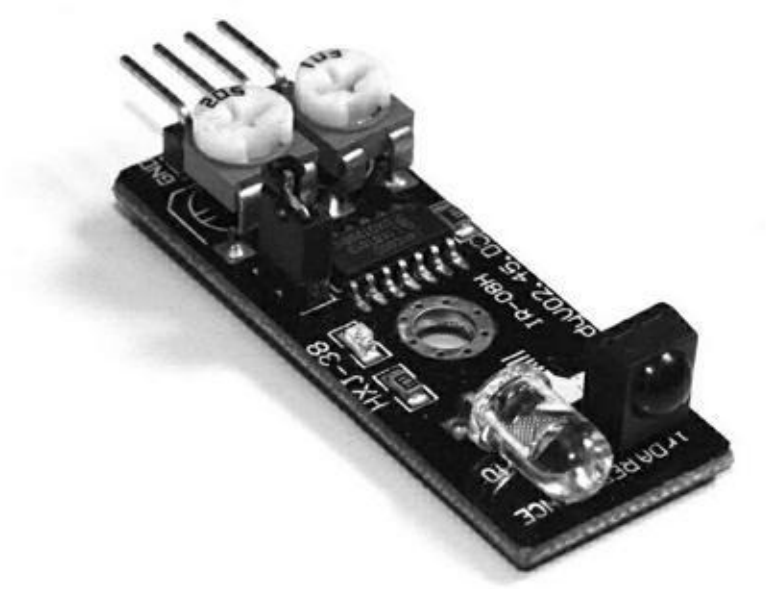


Figure 9.39 : Télémètre à ultrasons.

Vous serez heureux d'apprendre qu'un tel télémètre à ultrasons peut se trouver pour 5 euros environ et qu'il est directement utilisable avec Arduino. Voyez par exemple le module 20-019-100 de SainSmart.

Modules de communication

Le choix de modules est très vaste dans la catégorie des communications, qu'il s'agisse d'un simple adaptateur série RS-232 ou d'un module émetteur/récepteur sans fil complet.

Module sans fil APC220

Le composant émetteur/récepteur APC220 fonctionne entre 418 et 450 MHz. Pour établir une liaison, il faut deux modules et un éventuel adaptateur USB. Un module sera connecté à la machine hôte et l'autre à la carte Arduino. Le composant APC220 est capable d'un débit de 19 200 bits par seconde à une distance allant jusqu'à 1 000 m. Un point de connexion pour ajouter un module APC220 est prévu dans le bouclier multifonction présenté à la fin du chapitre précédent. C'est également le cas du bouclier afficheur 16 x 2 de DFRobot. La Figure 9.40 montre une paire de modules APC220.

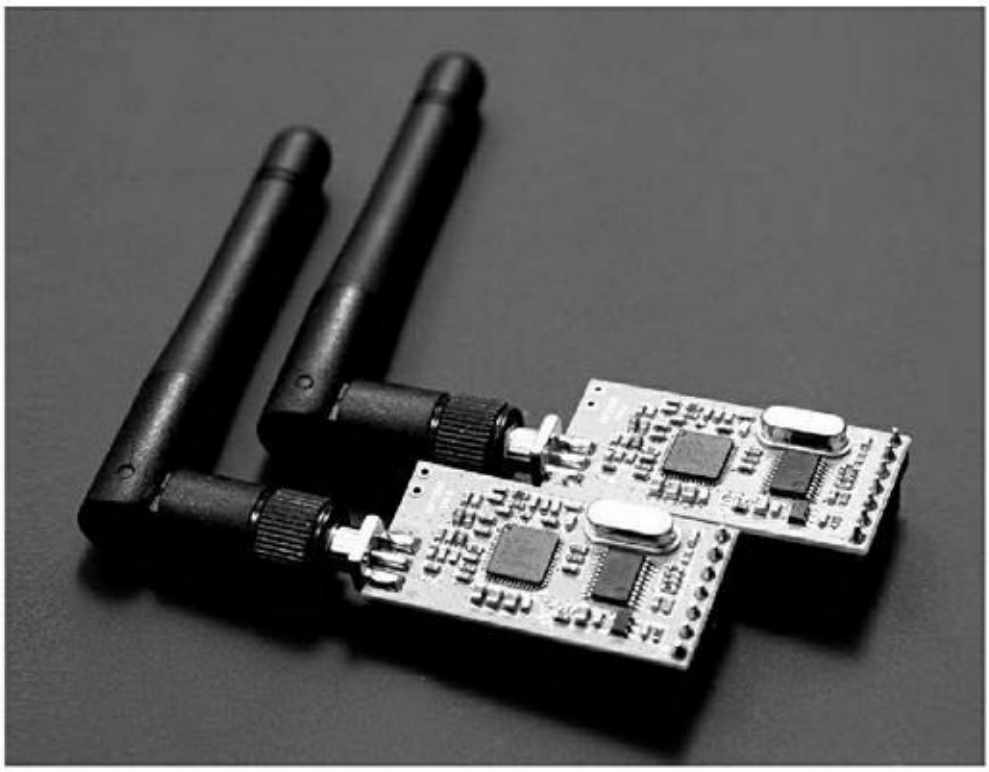


Figure 9.40 : Deux modules émetteurs/récepteurs APC220 (image DFRobot).

Modules RF 315 et 433 MHz

Ces modules n'ont qu'une portée de 150 m, mais ils sont moins chers que l'APC220. L'inconvénient est qu'ils ne combinent par les deux sens. C'est un couple constitué d'un émetteur et d'un récepteur (Figure 9.41). Ils sont préconfigurés en usine pour fonctionner soient à 315, soit à 433 MHz. Il faut penser à acquérir une antenne.

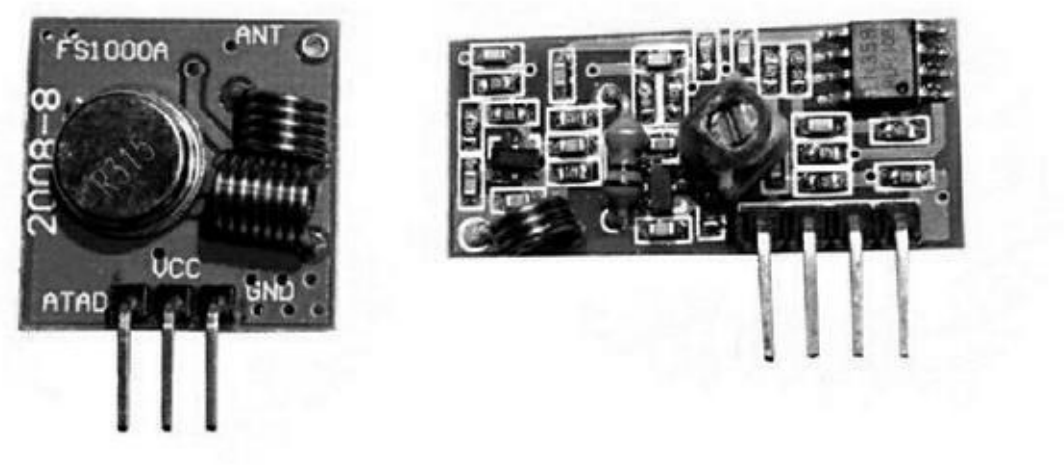


Figure 9.41 : Émetteur et récepteur 433 MHz.

Émetteur/récepteur WiFi ESP8266

Ce module WiFi très compact (Figure 9.42) sait exploiter les trois protocoles 802.11 b, g et n. Il communique par liaison série avec l'Arduino. Le microcontrôleur 32 bits dont il est doté comporte toute la pile logicielle de gestion du protocole réseau TCP/IP. Il se charge donc de gérer tous les détails de mise en relation et de maintien de la liaison numérique. La carte Arduino se limite à fournir l'adresse à laquelle elle désire se connecter, ou à se placer en attente qu'un autre module veuille se connecter à elle.

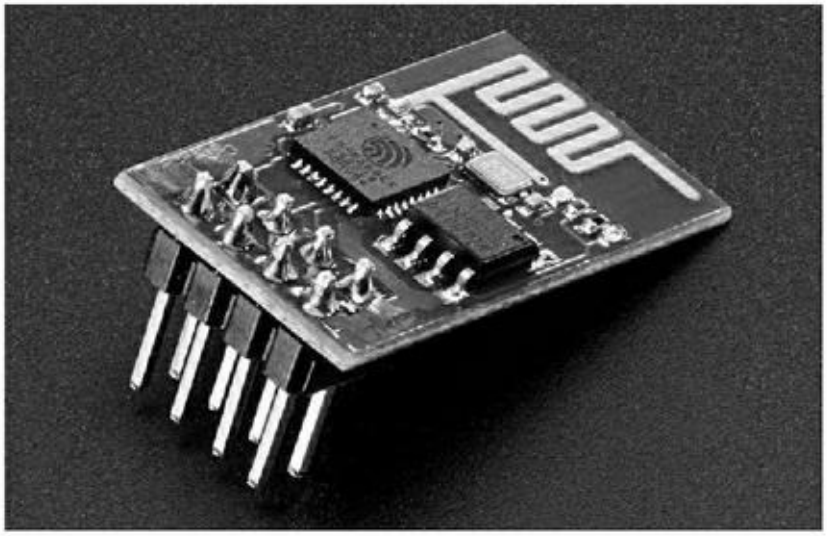


Figure 9.42 : Module émetteur/récepteur WiFi.

Module radio NRF24L01

Ce module est un émetteur/récepteur de faible puissance qui fonctionne à la fréquence de 2,4 MHz avec une portée de 250 m (Figure 9.43). Il communique par l'interface SPI et vous en trouverez pour moins de cinq euros.

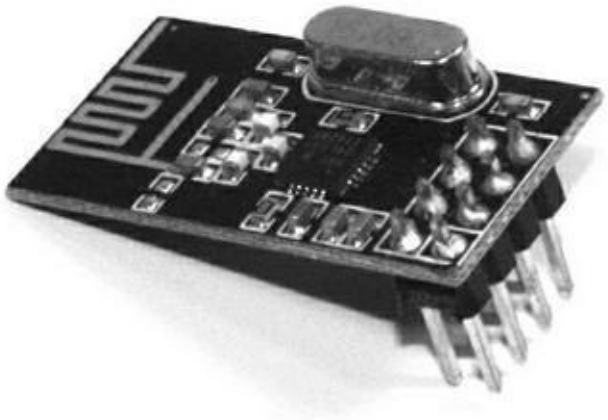


Figure 9.43 : Émetteur/récepteur radio NRF24L01.

Adaptateurs série RS-232

Les microcontrôleurs de la famille AVR qui équipent les cartes Arduino possèdent un circuit de gestion série UART, mais celui-ci ne génère pas des signaux standard RS-232. Il existe donc des modules adaptateurs RS-232 qui réunissent un convertisseur, un circuit intégré et le connecteur DB-9 (Figure 9.44). Le module se branche directement aux deux broches RxD et TxD du microcontrôleur.

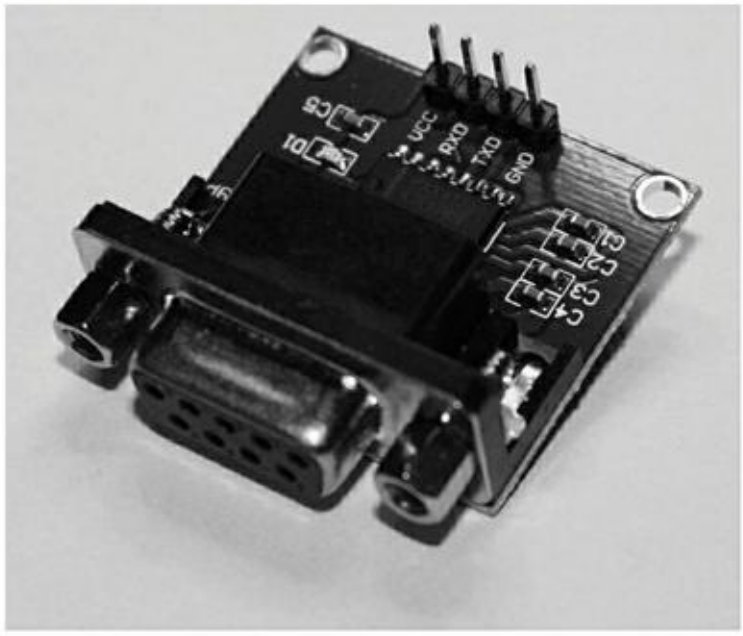


Figure 9.44 : Module adaptateur RS-232.

Modules et composants de sortie

Sur les sorties d'une carte Arduino, on peut connecter par exemple une diode LED, un servomoteur, un relais ou un autre module ou composant capable de répondre au signal numérique provenant du microcontrôleur. Dans cette section, nous allons aborder les générateurs de lumière puis les relais, les moteurs et les générateurs de son. Les composants d'interaction avec l'utilisateur, en entrée comme en sortie, font l'objet d'une autre section plus loin.

Émetteurs de lumière

Les composants qui émettent de la lumière sont aussi bien les bonnes vieilles lampes à incandescence que les diodes LED. Je rappelle qu'une diode LED est d'abord une diode, c'est-à-dire un composant qui ne laisse passer le courant que dans un sens. La variante électroluminescente émet de la lumière lorsque du courant la traverse (dans le bon sens). Nous allons nous intéresser aux diodes LED parce qu'elles sont très peu coûteuses, qu'elles durent longtemps et qu'il n'y a en général pas besoin de leur associer un circuit de pilotage. Cela dit, rien ne vous empêche d'utiliser des lampes à incandescence, mais prévoyez d'ajouter des circuits pour supporter le fort courant que la moindre ampoule va consommer, et faites des stocks, car ces ampoules durent peu, et ne seront bientôt plus en vente.

Vous avez un choix énorme au niveau des diodes LED, que ce soit en termes de couleurs ou de tailles. Un microcontrôleur AVR peut à la limite supporter de fournir entre 5 et 10 mA (milliampères) pour allumer une diode LED, mais il est tout à fait déconseillé d'en brancher plusieurs, et encore moins de piloter directement une LED haute brillance ou un module complet. Il faut dans ce cas un circuit pilote (*driver*). Je vous propose un bref tour d'horizon des types de diodes LED disponibles. Comme nous l'avons vu dans la visite des trois familles de modules du début de ce chapitre, vous pouvez trouver ces diodes LED déjà montées sur module.

Diodes LED monochromes

Il existe des diodes LED monochromes minuscules, comme la diode D13 soudée à demeure sur les cartes Arduino. Mais il en existe aussi des beaucoup plus grandes pour les animations lumineuses. La Figure 9.45 présente plusieurs modèles de diodes LED.

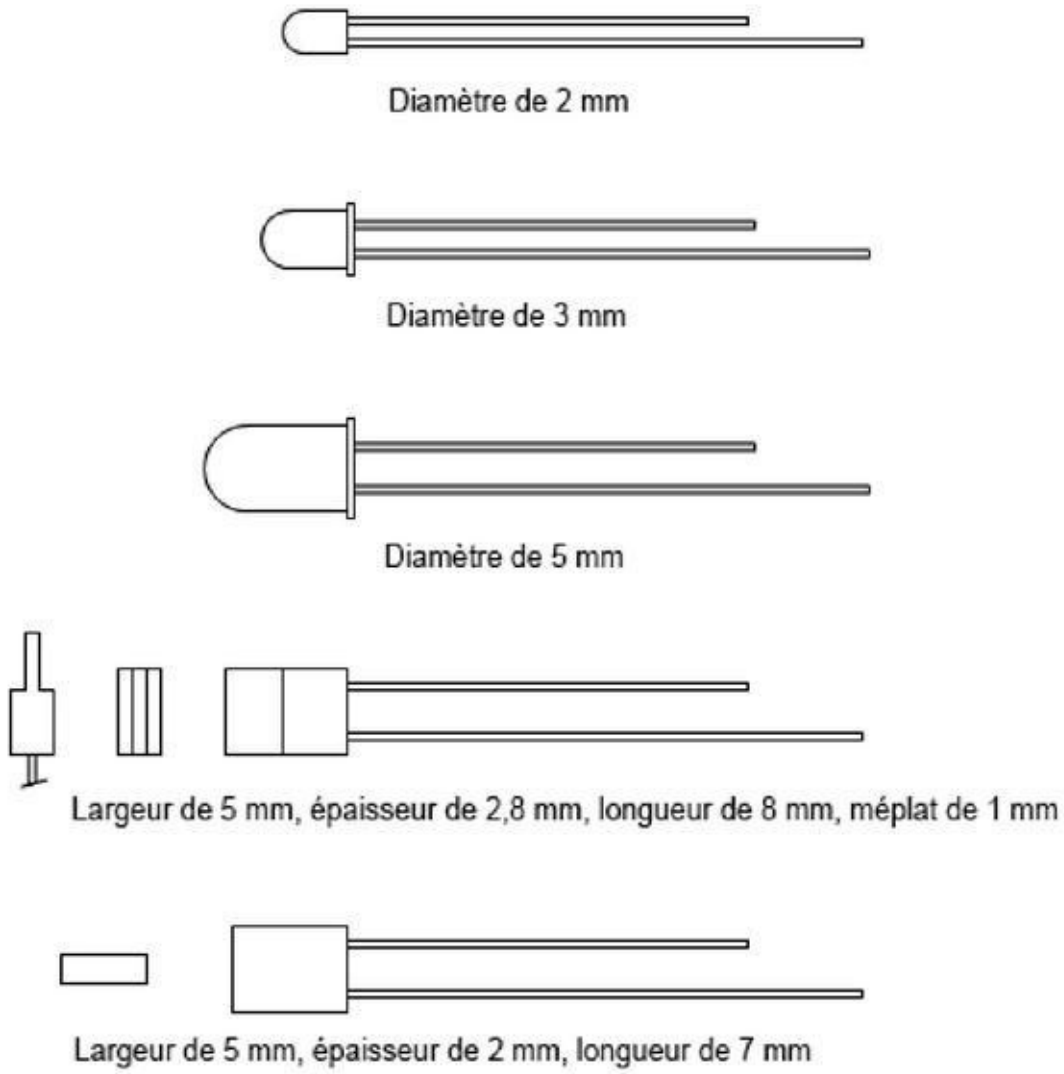


Figure 9.45 : Quelques diodes LED monochromes.

Diodes LED bicolores

Il s’agit en fait de deux diodes LED réunies dans le même boîtier ; l’aspect reste très proche de celui d’une diode monochrome. Les deux diodes sont montées tête-bêche (Figure 9.46). Quand le courant circule dans un sens, une diode s’allume, et quand le courant est inversé, c’est l’autre qui s’allume.

Il existe une variante comportant trois broches au lieu de deux. Dans ce cas, une broche est le point commun, ce qui permet d’allumer chaque diode indépendamment ou les deux à la fois.

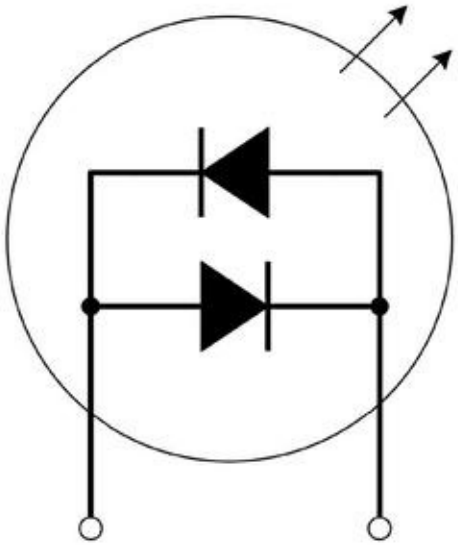


Figure 9.46 : Connexion interne d’une diode LED bicolore.

Diode LED tricolore (RGB)

Il s’agit bien sûr de trois diodes LED réunies dans le même boîtier plastique. La Figure 9.47 montre un exemple destiné à être monté en surface. Beaucoup de ces diodes sont déjà soudées sur un module. Plusieurs de ces modules peuvent être montés sous forme d’un tableau afin de créer un afficheur LED couleur de grande taille.

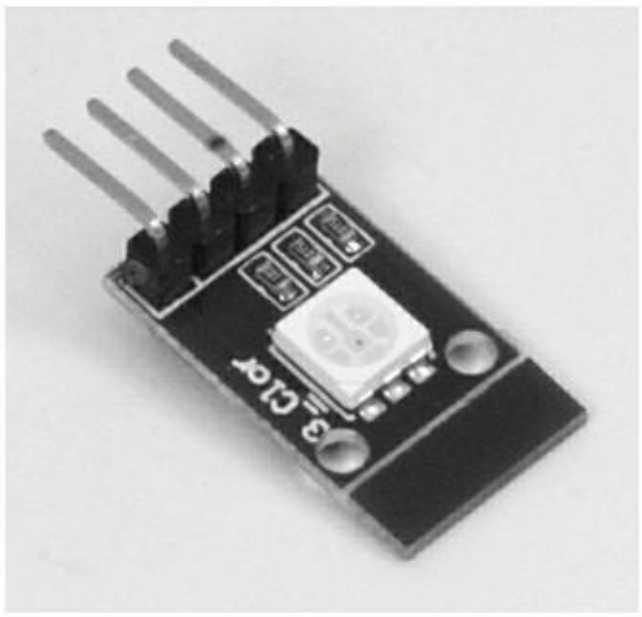


Figure 9.47 : Diode LED tricolore soudée (montage en surface SMC) sur module.

Une diode LED tricolore réunit les trois couleurs primaires permettant de produire quasiment toutes les nuances du spectre visible à partir du rouge, du vert et du bleu. La surface émettrice de chacune des trois LED étant séparée des deux autres, le boîtier est en plastique dépoli pour que les trois nuances fusionnent (si l'on ne regarde pas de trop près). Les afficheurs géants que vous voyez dans les stades et dans les rues commerçantes sont généralement construits avec des centaines de diodes LED tricolores dans une version de forte puissance.

Matrices de LED

Les matrices de LED sont très polyvalentes. Celle de la Figure 9.48 est une matrice de huit sur huit qui permet d'afficher toutes sortes de caractères.

Figure 9.48 : Matrice de diodes LED de huit sur huit.

Le format extérieur des matrices permet de les accoler pour créer de véritables bandeaux lumineux défilants. En effet, les bords extérieurs des matrices sont dimensionnés de façon à ce qu'en ajoutant une seconde matrice à côté de la première, l'ensemble offre le même écartement à la frontière entre les modules qu'entre les colonnes de chaque matrice.

N.d.T. : Voyez par exemple sur le site d'Adafruit la référence 555 qui combine six matrices de 8 sur 8.

Afficheurs LED 7 segments

Les afficheurs 7 segments sont les plus anciens. Ils ont constitué la première application à grande échelle des diodes LED, en dehors des diodes servant de témoins isolés. Dès la fin des années 1970, on a commencé à utiliser ces arrangements de 7 diodes segments pour afficher des chiffres et des lettres (stylisées). La Figure 9.49 montre un quadruple afficheur 7 segments (quatre chiffres).

Figure 9.49 : Afficheur 7 segments quatre chiffres.

Comme pour les matrices de LED, la largeur des côtés de l'afficheur a été choisie pour que l'on puisse facilement accoler deux afficheurs afin d'augmenter le nombre de chiffres. Vous pouvez facilement créer un bandeau de 12 chiffres en combinant trois blocs de quatre.

Modules de diodes LED 7 segments

Pour vous simplifier l'utilisation des afficheurs 7 segments, vous pouvez acquérir directement un module afficheur 7 segments, fonctionnant avec une interface SPI ou I2C (Figure 9.50). Ce module d'exemple est proposé par la société Tindie. La couleur des diodes peut être rouge, verte, jaune ou bleue.

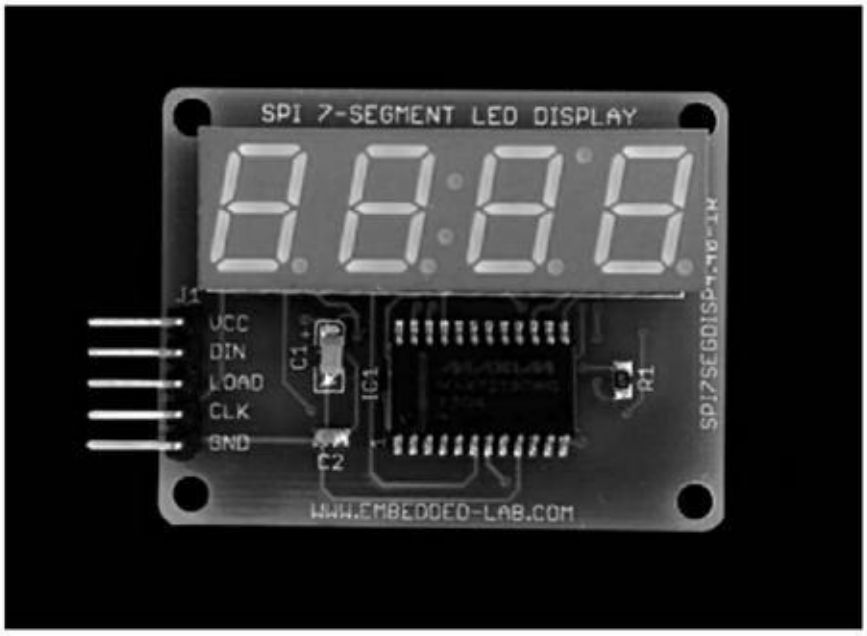


Figure 9.50 : Module afficheur 7 segments à interface SPI.

Diodes laser

Il existe des diodes LED servant de laser, généralement de faible puissance, comme celles utilisées dans les pointeurs lumineux pour conférencier, bien qu'il existe des diodes laser assez puissantes pour couper du plastique et du bois. Un laser silicium est en fait une diode LED modifiée pour émettre sa lumière sur une seule longueur d'onde. Les fréquences disponibles vont de l'infrarouge au bleu. Les diodes laser servent de niveau laser dans la construction, de pointeur de conférencier, d'analyseur de surface dans la modélisation 3D, de lecteur optique dans les lecteurs de CD et de DVD, et même dans les outils de découpe laser industrielle. La Figure 9.51 montre l'aspect d'un petit module de diode laser, en l'espèce c'est le module KY-008. Notez que vous pouvez acheter la diode laser séparément.

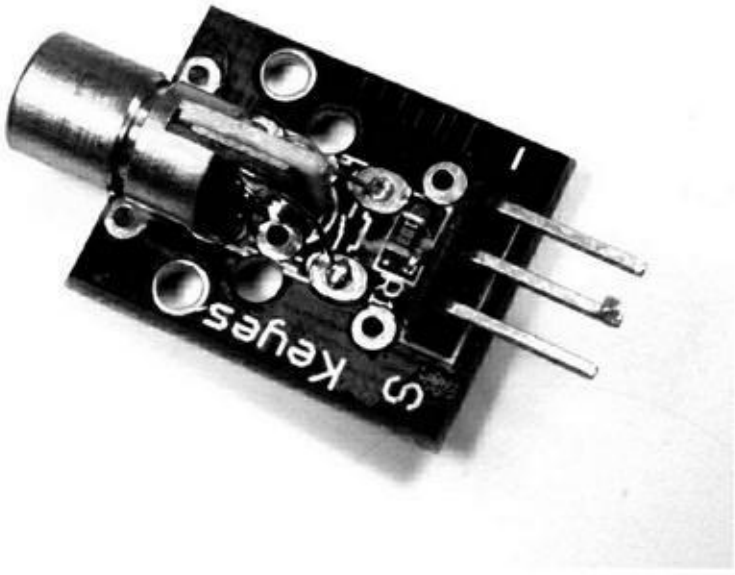


Figure 9.51 : Exemple de module à diode laser rouge de faible puissance.

Relais, moteurs et servomoteurs

Les relais, même les plus petits, peuvent supporter un courant d'alimentation bien supérieur à ce que supporte le microcontrôleur d'une carte Arduino. Les moteurs et les servomoteurs ont donc généralement besoin d'un circuit de pilotage de charge capable de supporter l'intensité dont ils ont besoin pour réaliser leur travail.

Relais

Les relais miniatures ressemblent à des boîtiers de circuits intégrés à 14 broches (je laisse de côté les grands relais du monde industriel). Dans les projets Arduino, les petits relais suffisent en général, d'autant que vous pouvez contrôler un relais plus grand à partir d'un petit relais, et ainsi de suite en cascade. Le module KY-019 est un bon exemple de relais pouvant être directement connecté à un Arduino.

Le circuit de commande du relais peut se résumer à un transistor de type 2N2222, mais ce sera plus généralement un circuit dédié, capable de supporter le courant et les pointes de courant inverse qui sont produites lors de l'ouverture et de la fermeture des contacts. Le circuit de la Figure 9.5 du début

du chapitre utilise justement un transistor NPN pour commander un petit relais monté sur le circuit.

La Figure 9.52 montre l'aspect général d'un module équipé de quatre relais ainsi que des transistors, résistances et diodes du circuit de commande. Il ne reste plus qu'à connecter une source d'alimentation +5 V continus et les signaux logiques de commande.

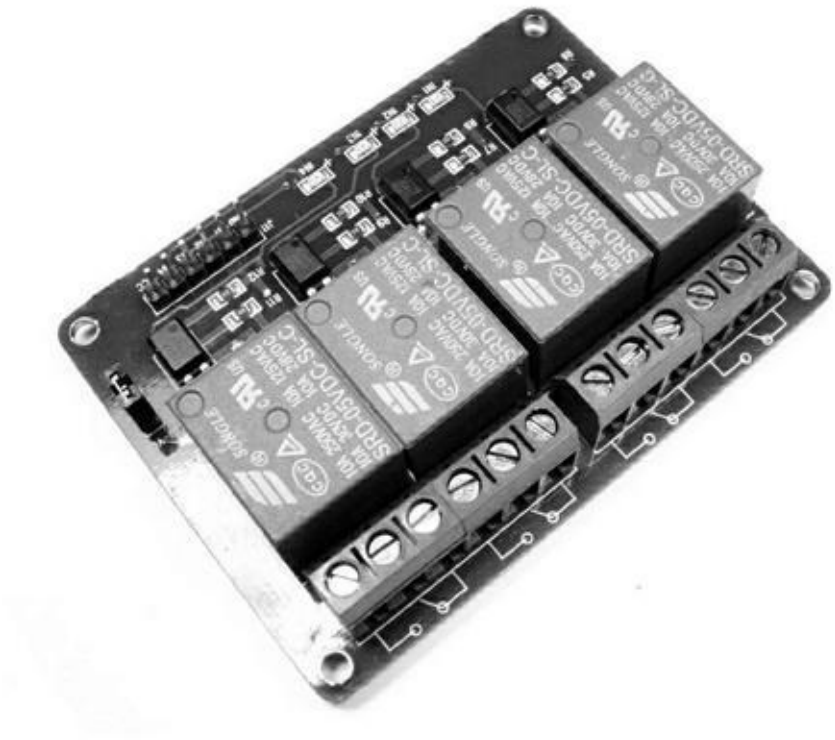


Figure 9.52 : Module de quatre relais soudés sur le circuit.

Servomoteurs

Un servomoteur n'est pas destiné à tourner à grande vitesse comme un moteur classique, mais à se positionner à un angle précis sur un demi-tour ou un tour complet selon l'ordre reçu de son circuit d'asservissement (de commande). Les petits servomoteurs comme ceux de la Figure 9.53 sont très utilisés en radiomodélisme et en robotique.

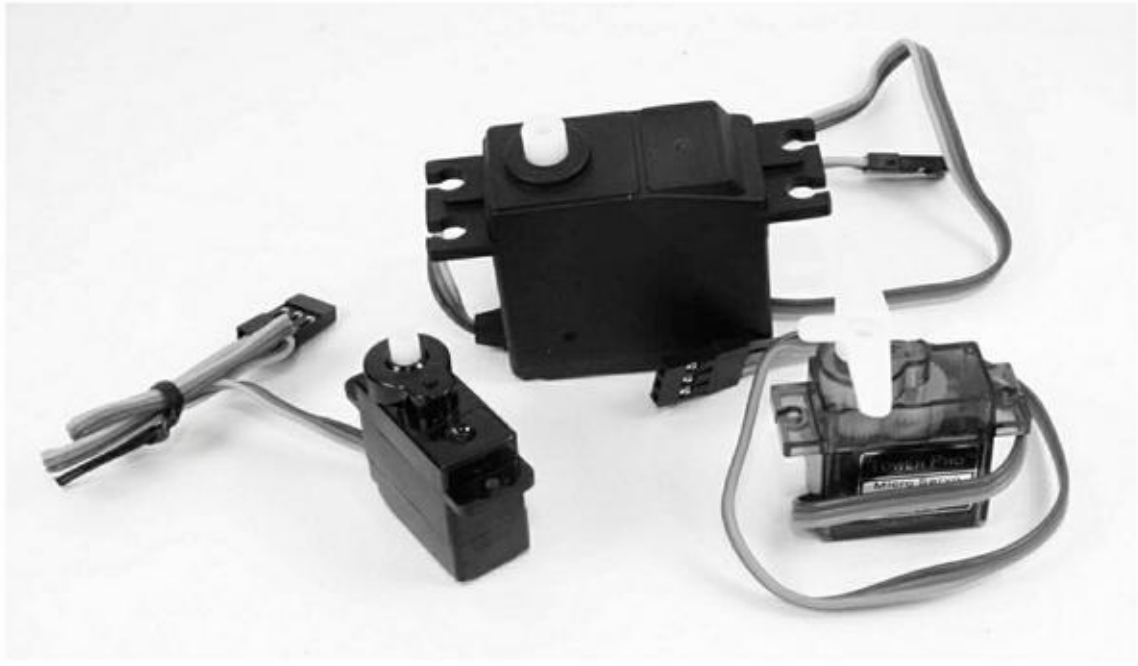


Figure 9.53 : Trois servomoteurs.

J'ai présenté dans le Chapitre 8 des boucliers pour contrôler des servos, mais les sorties d'un microcontrôleur peuvent suffire à contrôler directement un (petit) servomoteur en mode largeur d'impulsion PWM. Les servomoteurs visibles dans la Figure 9.53 ont un axe qui peut tourner de 180°. La position entre 0 et 180 degrés est déterminée par la largeur et la fréquence de l'impulsion du signal de commande.

Moteurs à courant continu

Les moteurs à courant continu ou moteurs DC (*Direct Current*) sont en général commandés par un circuit de pont en H (Figure 9.54) sous une forme simplifiée. Un pont en H peut fonctionner avec une tension continue ou avec un signal d'impulsion PWM. Le moteur peut fonctionner en marche avant ou en marche arrière en fonction des signaux de commande.

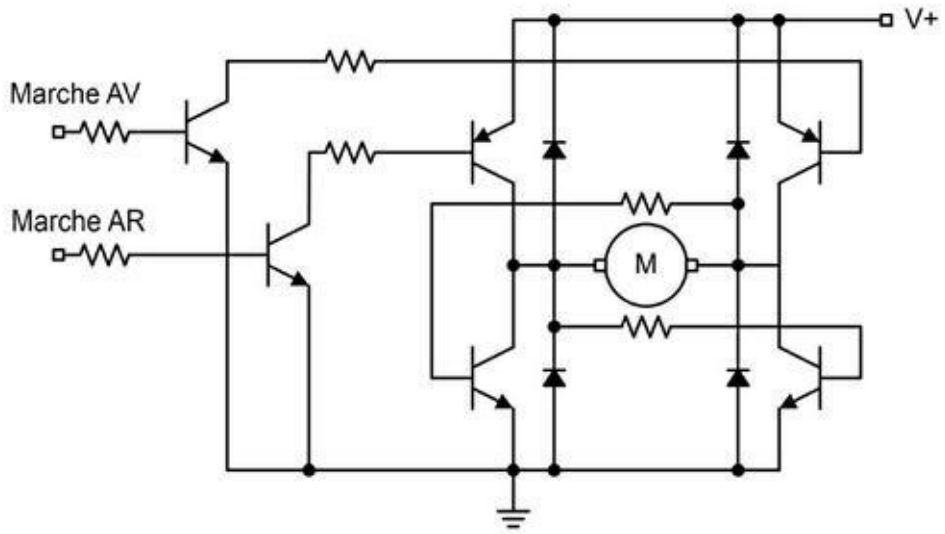


Figure 9.54 : Schéma simplifié d'un pont en H.

Je ne vous recommande pas de construire vous-même un tel circuit de commande, à moins d'en avoir vraiment besoin. Mieux vaut acquérir un bouclier tel que celui de la Figure 9.55, car il réunit tout ce qui est nécessaire pour contrôler un moteur continu. Il est même doté d'un radiateur pour dissiper la chaleur émise par le courant important. Le bouclier de la figure provient de la société Seeed Studio. Il est capable de contrôler deux moteurs continus ou un moteur pas à pas.

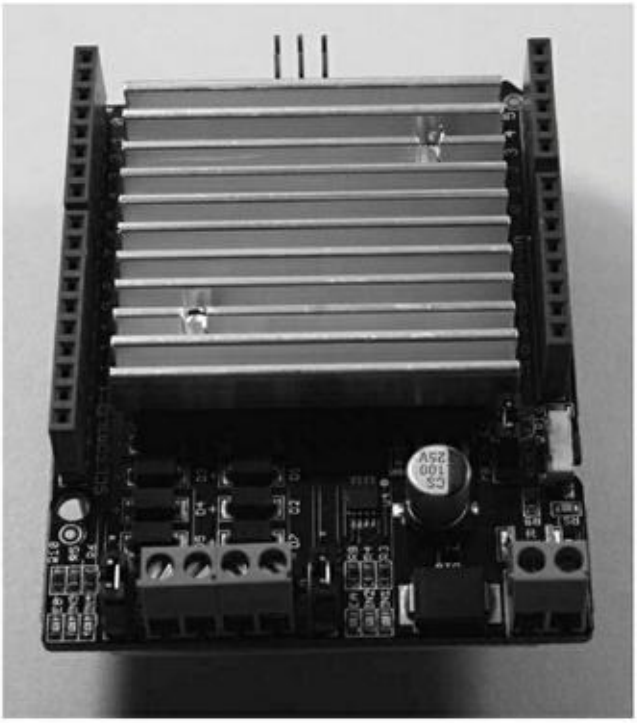


Figure 9.55 : Bouclier de contrôle de moteurs continus.

Moteurs pas à pas

Un peu comme les servomoteurs, les moteurs pas à pas sont asservis en angle par les impulsions du signal de commande. Il faut bien tenir compte du courant requis. Un circuit de contrôle tel que l'ULN2003A suffit à contrôler un petit moteur pas à pas, mais ses limites sont rapidement atteintes. En fait, le circuit ULN2003A contient huit montages de transistors Darlington et rien d'autre. C'est le logiciel dans la carte Arduino qui doit gérer toute la production des impulsions de commande avec des durées précises.

Vous trouverez des boucliers, comme je l'ai montré dans le Chapitre 8, qui contiennent tous les circuits de commande pour un à quatre moteurs pas à pas, avec des connecteurs pour simplifier les branchements. Comme pour les moteurs continus, il est beaucoup plus simple d'acheter un bouclier ou un module que de reconstruire tout le circuit de commande soi-même. Si vous comptez le temps passé, c'est vraiment beaucoup plus économique.

Sorties analogiques

Un signal analogique est une onde qui varie continuellement, sans effet d'escalier. Un tel signal électrique permet de générer des vibrations dans l'air (du son) ou des ondes radiofréquences. Pour qu'une carte Arduino puisse produire un signal analogique, il faut lui ajouter des composants. En effet, le microcontrôleur AVR ne contient pas de convertisseur numérique vers analogique (CNA). Si vous avez besoin d'autre chose que des ondes carrées produites par des impulsions PWM ou par un déclenchement de chronomètre, il vous faut un circuit complémentaire qui va générer le signal analogique.

N. d. T. : Le terme anglais *digital* a tendance à venir envahir la langue de Molière qui le traduit par « numérique » (le français ne confond pas les empreintes digitales et les empreintes numériques). Un DAC (*Digital Analog Converter*) est un CNA (*Convertisseur Numérique Analogique*) mais vous trouverez aussi l'expression CDA (*Convertisseur Digital Analogique*). De même, dans l'autre sens : un ADC anglais est un CAN ou un CAD français.

Buzzer audio

Un buzzer permet d'émettre soit un son de hauteur fixe lorsqu'il est activé, soit un son dont vous contrôlez la fréquence. Les deux modules KY-006 et KY-012 sont des exemples de buzzers piézoélectriques.

Moyennant quelques précautions, vous pouvez également brancher un petit haut-parleur sur une sortie Arduino, mais les ondes carrées ne sont pas très musicales.

Modules de conversion numérique/analogique (CNA ou DAC)

Le module DAC d'Adafruit (Figure 9.56) se fonde sur le circuit MCP4725. Il s'agit d'un CNA monocanal sur 12 bits avec interface I2C.

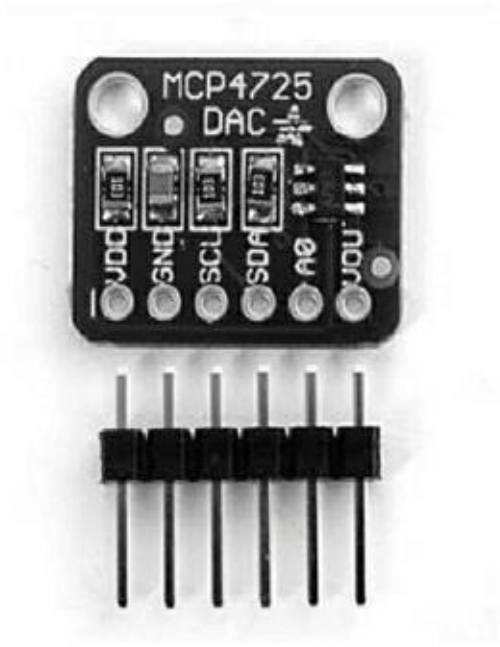


Figure 9.56 : Module convertisseur basé sur le MCP4725.

Ce genre de convertisseur est indispensable lorsque vous avez besoin d'un signal variant de façon continue, par exemple pour une tension de commande d'un appareil classique. Ce convertisseur permet aussi de générer une onde en dents de scie ou en sinus.

L'interface I2C standard des microcontrôleurs AVR est limitée à 100 ko par seconde. Il est donc impossible d'atteindre les taux de rafraîchissement que permettent les interfaces I2C rapides. Un convertisseur pour Arduino peut malgré tout générer une sinusoïde à basse fréquence. L'inconvénient est que cette génération va occuper le microcontrôleur en permanence, qui ne pourra donc faire aucun autre traitement.

Générateurs de fréquences

Dès que vous avez besoin de générer une onde de qualité à une fréquence supérieure à environ 1000 Hz, vous opterez pour un circuit dédié. Un bon exemple est le module DDS représenté dans la Figure 9.57.

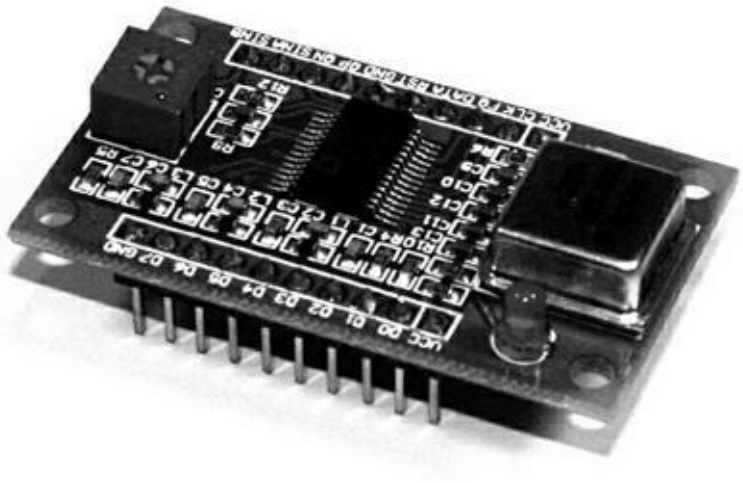


Figure 9.57 : Module générateur DDS basé sur le circuit AD9850.

Ce circuit génère une onde carrée et une onde sinusoïdale entre 1 Hz et 40 MHz. La fiche technique peut être téléchargée sur le site du fondeur, Analog Devices. Le circuit utilise une interface spécifique. Vous pouvez le contrôler soit sur 8 bits en parallèle, soit en série. Les librairies Arduino sont disponibles. C'est le circuit que j'utilise dans le projet de générateur de signal du Chapitre 11.

Saisie de données de l’utilisateur

Il est souvent nécessaire dans les projets de permettre à un humain d’interagir avec le circuit, au moyen de boutons, de potentiomètres, d’un clavier ou d’un joystick. En plus des modules que je présente dans cette section, vous pouvez acheter séparément les composants et les mettre en place selon vos besoins particuliers.

Pavés de saisie numérique

Ces blocs de touches sont généralement désignés sous le terme *keypad*. Ils proposent en général 3 rangées de 3 ou 4 boutons ou bien 4 rangées de 4 boutons, dans une grille. Certains modèles sont très plats, car constitués d’une fine membrane collée sur un circuit. Voyez les quatre exemples de la Figure 9.58. Il existe des pavés plus complets, et d’ailleurs le clavier d’un ordinateur est un pavé de touches beaucoup plus grand.

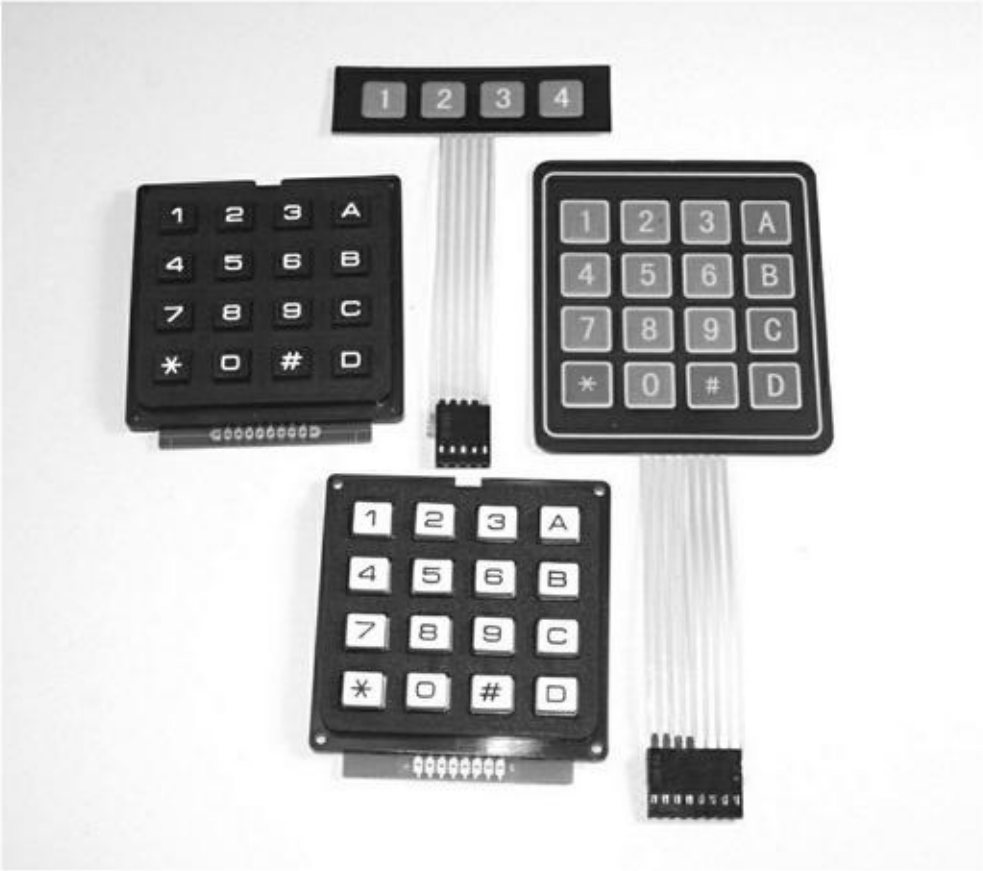


Figure 9.58 : Assortiment de pavés de saisie numérique.

Joystick (manches à balai)

Deux technologies sont employées dans la catégorie des manches à balai appelés joysticks : analogique continu ou numérique discontinu. Les joysticks analogiques (Figure 9.59) réunissent deux potentiomètres implantés selon les deux axes X et Y. La valeur lue sur chacun des deux potentiomètres permet de savoir comment est positionné le levier.

Figure 9.59 : Module de joystick analogique.

Les joysticks numériques utilisent de petits interrupteurs ou d'autres moyens de détection pour détecter l'arrivée du levier en butée dans l'axe X ou Y. C'est ce genre de joystick économique qui est utilisé dans les consoles de jeux. Ce sont des équipements en tout ou rien beaucoup moins sensibles à la poussière et à l'usure que les joysticks analogiques. De nombreux boucliers d'affichage LCD possèdent un tel joystick pour les réglages.

Potentiomètres et encodeurs rotatifs

Vous savez qu'un potentiomètre est une résistance variable qui permet donc de fournir une donnée d'entrée. On s'en sert comme bouton de volume dans les chaînes stéréo, pour les réglages dans les appareils de laboratoire, et dans bien d'autres circuits analogiques. Deux exemples de modules à potentiomètre sont les T000140 et T000150 de TinkerKit.

Nous avons vu quelques pages auparavant (Figure 9.36) un exemple d'encodeur rotatif qui peut servir de moyen de saisie de données. À la différence du potentiomètre qui produit une tension variable qu'il faut ensuite convertir en entrée du microcontrôleur, l'encodeur rotatif produit directement une valeur numérique.

Affichage de données pour l’utilisateur

Pour qu’un projet soit interactif, il faut non seulement recueillir des données de l’utilisateur, mais également lui en présenter. Vous pouvez vous servir de simples diodes LED comme témoins ou afficher des messages et des images sur un écran LCD ou TFT. Un tel afficheur est indispensable pour confirmer à l’utilisateur que ses saisies ont bien été prises en compte.

Les modules et composants d’affichage que nous allons voir sont les mêmes que ceux utilisés dans les boucliers d’affichage LCD décrits dans le Chapitre 8, les boucliers étant plus pratiques à utiliser. Dans certaines situations, le bouclier n’est pas utilisable et il faut dans ce cas recourir à un composant afficheur qui pourra être installé exactement comme requis par le projet.

J’ai présenté toute une série de boucliers d’affichage dans le Chapitre 8. N’hésitez pas à y revenir. À moins d’avoir vraiment besoin d’un composant nu, il est toujours conseillé de prendre un bouclier.

Affichage de texte

Les écrans en mode texte sont très économiques. Ils permettent d’afficher entre une et quatre lignes de texte, sur une longueur de 8, 16, 20, 24, 30 ou 40 caractères. Le prix est proportionnel au nombre de lignes. Vous trouverez des afficheurs deux lignes de 16 caractères pour moins de 5 euros, un afficheur quatre lignes de 40 caractères valant moins de 20 euros.

ERM1601SBS-2

Cet afficheur LCD une ligne de 16 caractères propose un affichage en blanc sur fond bleu, comme le module de la Figure 9.60. Le circuit intégré de contrôle est en général un HD44780 ou un KS066. Ces deux circuits savent gérer le rétroéclairage de l’écran et se vendent pour moins de 3 euros. Il existe des variantes avec lettres noires sur fond jaune ou sur fond blanc.

Figure 9.60 : Module d'affichage ERM1601SBS-2.

ST7066 (HD44780)

Le format de cet afficheur LCD, deux lignes de 16 caractères, est un des plus répandus. Il fonctionne avec une interface parallèle et un circuit contrôleur HD44780 ou ST7066. C'est l'afficheur utilisé par la plupart des boucliers LCD d'Adafruit, SparkFun, SainSmart et bien d'autres. L'exemple de la Figure 9.61 est vendu environ 10 euros.

Figure 9.61 : Afficheur deux lignes de 16 caractères basé sur HD44780 ou ST7066.

Si vous comparez les figures 9.60 et 9.61, vous pourriez croire qu'il s'agit du même module. Le circuit de contrôle est effectivement le même, mais l'un des

écrans n’affiche qu’une ligne et l’autre deux.

Vous trouverez d’autres modèles d’afficheurs LCD en mode texte ou en mode graphique. Une solution très économique consiste à récupérer des afficheurs de surplus chez les revendeurs d’électroménager. L’inconvénient est que les caractères préprogrammés dans la mémoire ne conviennent pas toujours, par exemple si l’afficheur était prévu pour un lave-vaisselle ou un four à micro-ondes.

Affichages graphiques

Les afficheurs capables de graphiques utilisent les technologies LCD, TFT ou OLED, en monochrome ou en couleur. Les technologies TFT et OLED sont bien meilleures que celles basées sur les cristaux liquides, mais les composants sont plus coûteux.

Afficheur LCD monochrome ERC240128SBS-1

Cet afficheur (Figure 9.62) utilise la technologie LCD et offre une résolution de 240 sur 128 pixels. Il est contrôlé par une interface parallèle 8 bits. Il est commercialisé par BuyDisplay.

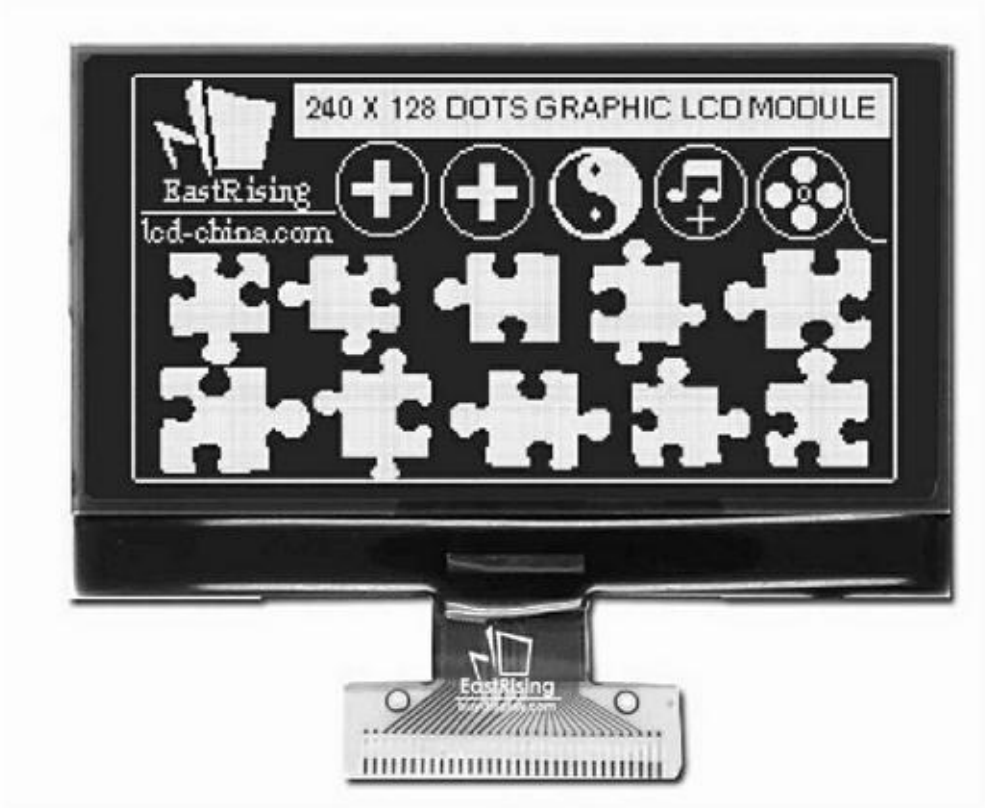


Figure 9.62 : Afficheur monochrome de 240 sur 128 pixels.

Afficheur TFT couleur ST7735R

Un autre afficheur répandu est l'écran TFT de 128 sur 160 pixels proposé par Adafruit (Figure 9.63). La diagonale de l'écran est de 4,6 cm (1.8''). Cet écran offre 18 bits de profondeur de couleur, ce qui permet d'obtenir 262144 nuances. Le circuit contrôleur est le ST7735R avec interface SPI. Le module est même doté d'un lecteur de carte microSD à l'arrière.

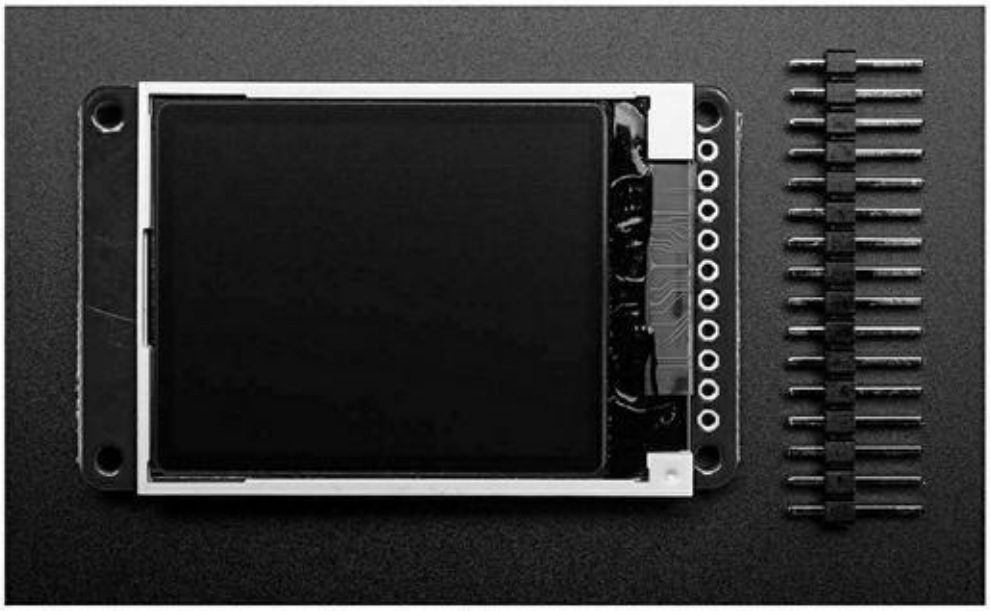


Figure 9.63 : Module afficheur TFT couleur de 1,8 pouce.

Fonctions de support

La grande majorité des modules que nous avons vus servent soit en entrée, soit en sortie. Il existe aussi quelques modules qui n'entrent dans aucune de ces deux catégories. Ce sont en particulier les circuits d'horloge et les chronomètres ou timers.

Horloges temps réel (RTC)

Plusieurs circuits intégrés sont disponibles pour remplir le rôle d'une horloge en temps réel, RTC (*Real-Time Clock*). Tous ont le même objectif principal : conserver l'heure et la date. Certains circuits possèdent un peu de mémoire EEPROM, mais pas tous. L'interface varie d'un circuit à l'autre, certains utilisant l'interface SPI, d'autres l'interface I2C, et certains même une interface spécifique. Le principal critère de choix sera la précision et la stabilité de l'heure, qui dépend de la sensibilité du circuit aux changements de température. Les deux autres critères sont la consommation d'énergie et le prix d'achat.

Les quatre modules d'horloge temps réel les plus répandus utilisent l'un des quatre circuits DS1302, DS1307, DS3231 et PCF8563. Tous comportent un porte-pile pour une pile bouton CR2032 ou LIR2032. Tous remplissent bien leur fonction, mais des tests indépendants ont montré que le circuit offrant la meilleure stabilité à long terme était le DS3231. Les modules qui utilisent un cristal externe sont sensibles aux changements de température. Tous les circuits vont un peu dériver au cours du temps.

Module d'horloge DS1302

Le module DS1302 (Figure 9.64) utilise une interface série spéciale, différente de SPI. Une broche sert aux données, une autre au signal d'horloge et une troisième au signal CE de sélection du circuit. Le fabricant appelle cela une interface sur trois fils. Pour d'autres détails, visitez le site de ce fabricant, Maxim Integrated (anciennement Dallas Semiconductors).

Figure 9.64 : Module d'horloge DS1302.

Module d'horloge DS1307

Le module basé sur le circuit DS1307 communique par une interface I2C. Il n'est de ce fait pas compatible ni au niveau brochage, ni au niveau logiciel avec le DS1302. La fiche technique détaillée est disponible chez Maxim. La Figure 9.65 montre un module de TronixLabs basé sur ce circuit.

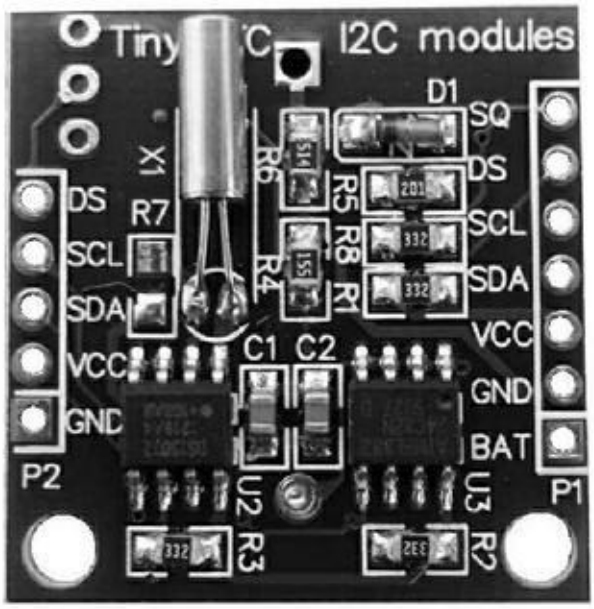


Figure 9.65 : Module d'horloge DS1307.

Module d'horloge DS3231

Ce module utilise lui aussi l'interface I2C, il est donc compatible au niveau logiciel avec le DS1307. Il s'en distingue par une meilleure précision, car il dispose d'un cristal interne, ce qui le rend moins sensible aux changements de température. Un tel module est représenté dans la Figure 9.66.

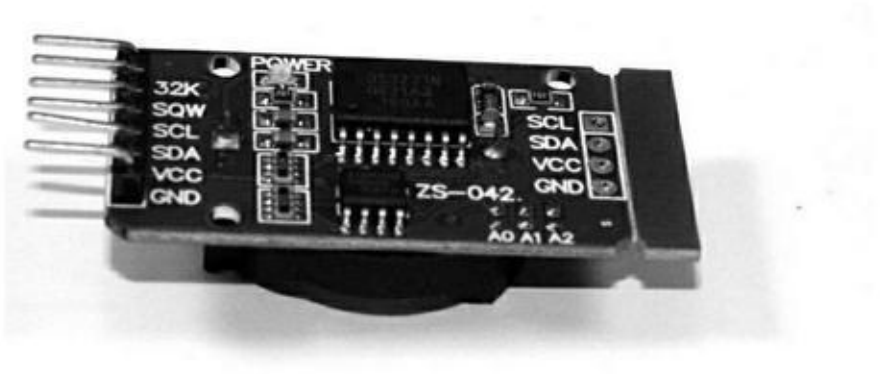


Figure 9.66 : Module d'horloge DS3231.

Module d'horloge PCF8563

Ce module utilise l'interface I2C, mais ses registres internes sont construits de manière totalement différente des circuits de chez Maxim. Un module de NXP l'utilisant est représenté en Figure 9.67.

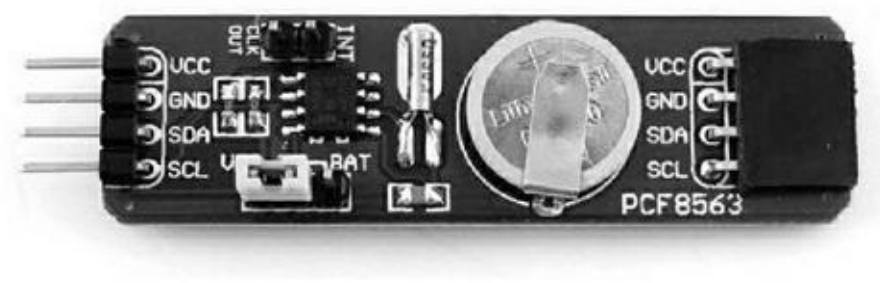


Figure 9.67 : Module d'horloge PCF8563.

Timers ou décompteurs

Les microcontrôleurs AVR contiennent un timer (*watchdog*), mais il est dans certains cas nécessaire d'utiliser un décompteur externe. Un circuit watchdog, ou chien de garde, est en fait un chronomètre qui décompte pendant un certain temps, et déclenche un événement une fois arrivé à zéro. Vous aurez besoin d'un

timer (prononcer *taillemeur*) externe lorsque l'événement qui est déclenché doit provenir d'un appareil extérieur au microcontrôleur. C'est par exemple le cas d'un mécanisme rotatif équipé d'un capteur magnétique émettant une impulsion par tour d'axe. Si les impulsions servent à faire décompter le timer, vous pouvez détecter l'arrêt du mécanisme. Les pannes peuvent ainsi être détectées par le microcontrôleur au moyen d'une interruption ou en interrogeant régulièrement l'état du décompteur. Il peut ensuite réagir de façon appropriée.

La majorité des modules timers externes utilisent le circuit intégré 555. Il fonctionne avec un transistor à effet de champ MosFet qui décharge un condensateur et remet le compteur à zéro à chaque détection d'une impulsion. Il existe des modules timers en provenance d'Asie sur lesquels la légende du circuit a été effacée ou masquée par une goutte de résine noire. J'évite ce genre de circuit, parce qu'il est très difficile de savoir exactement comment il fonctionne sans détruire le circuit en tentant d'enlever la résine.

Si vous avez besoin d'utiliser un module timer externe, je vous suggère d'acquérir un des livres d'électronique qui sont entièrement consacrés à ce très utilisé circuit 555. La société australienne Freetronics propose un module timer externe basé sur le 555 à très faible coût.

Connexions

Au départ, les connexions entre les cartes Arduino et les modules et capteurs se faisaient avec des straps, c'est-à-dire des fils terminés par des broches de contact mâles ou femelles. Depuis quelques années, cette façon de faire tend à laisser la place à de véritables systèmes d'interconnexion basés sur des connecteurs modulaires. Dans ces systèmes, tous les modules utilisent le même connecteur avec trois ou quatre broches : deux pour l'alimentation et la ou les autres pour les signaux. Tous peuvent généralement être raccordés à un bouclier d'interface centralisateur.

Cette standardisation des connexions est proposée par les modules TinkerKit et les modules Grove vus dans ce chapitre. Les connecteurs sur les modules et sur les boucliers sont reliés par des câbles à trois ou quatre fils avec en général un verrouillage. Revoyez par exemple le kit de routage passif dans le Chapitre 8, en Figure 8.21.

Connexions par fils volants ou straps

Supposons que vous ayez à brancher de manière définitive un module à une carte Arduino, mais que vous n'ayez pas le temps de construire vos propres connecteurs. Vous pouvez tout à fait utiliser des fils volants ou straps, à condition de prendre le temps de vérifier que les connexions ne risquent pas de se débrancher.

Les broches de connexion mâles et femelles qui constituent les extrémités des straps sont les mêmes que les broches que l'on trouve dans les connecteurs modulaires. Ce qui change est que les straps sont insérés individuellement alors qu'un connecteur a par définition au moins deux contacts. Plus il y a de broches insérées, plus la liaison est robuste puisque la force de frottement de chaque broche s'additionne à celle des autres. Un connecteur tient donc plus solidement qu'une série de fils individuels.

De plus, un fil volant se plie plus aisément qu'un câble. Pour éviter des débranchements intempestifs, une solution consiste ajouter un point de colle silicone sur la connexion. C'est moins élégant qu'un vrai connecteur, mais cela devrait suffire, à moins que le montage doive fonctionner dans un

environnement soumis à de fortes vibrations comme une voiture radiocommandée ou une machine-outil.

N’abusez pas de la silicone, car vous aurez peut-être besoin de l’enlever pour remplacer le module. Vous pourrez éliminer la silicone avec un couteau pointu, mais faites attention à ne pas endommager le fil.

Systèmes de connexion modulaires

Les boucliers et modules qui possèdent des connecteurs, qu’il s’agisse du système TinkerKit ou de celui de Grove, peuvent bien sûr être utilisés avec des fils volants individuels ou en rangées de broches mâles ou femelles. L’avantage des connecteurs est qu’ils évitent les erreurs de branchement, une fois que l’ordre des broches a été vérifié. C’est l’approche choisie par les systèmes TinkerKit et Grove. Pour tout détail au sujet de Grove, voyez le site Web de Seeed Studio.

La famille de modules TinyDuino de TinyCircuits utilise des connecteurs multibroches miniaturisés et montés en surface (Figure 9.68). TinyCircuits propose toute une gamme de modules capteurs avec ce type de connecteur, ainsi que les câbles correspondants. Pour juger de la petitesse de ces connecteurs, comparez la figure suivante à la taille du connecteur USB d’une Arduino Nano. Pour tout détail, visitez le site Web de TinyCircuits.

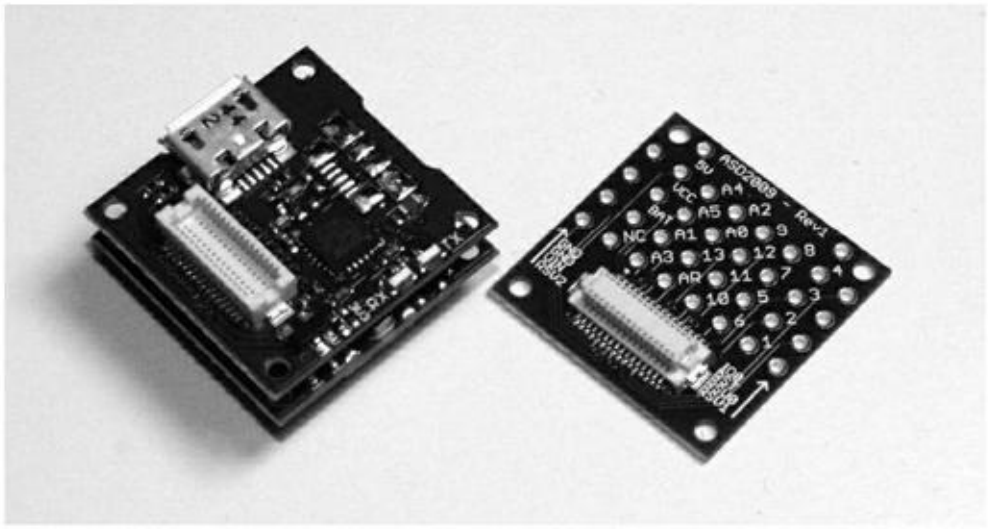


Figure 9.68 : Deux modules à connecteur miniaturisé de TinyCircuits.

Le problème des standards est quand il faut les faire cohabiter. Autrement dit, il n'est pas simple d'utiliser dans un même projet des modules selon un standard avec des modules d'un autre standard. Rien ne garantit que le module d'un fabricant soit compatible avec un bouclier d'un autre. Une solution consiste à faire venir toutes les connexions des modules sur un bouclier d'interface, et c'est exactement ce que propose la gamme TinkerKit. Intéressez-vous donc toujours aux modes de connexion des modules que vous envisagez d'utiliser et prévoyez d'acheter des câbles supplémentaires ainsi que des modules de prototypage pour être certain de pouvoir interfacer des composants de plusieurs sources. Vous pouvez aussi vous lancer dans la fabrication de vos propres câbles. C'est ce que nous allons découvrir maintenant.

Fabriquer ses câbles et ses connecteurs

Non seulement les connecteurs sont plus faciles à utiliser, mais ils garantissent des connexions plus fiables que des fils volants. Les connecteurs les plus simples sont en fait des rangées de logements et des embases pour broches à sertir au pas standard de 2,54 mm. Revoyez par exemple en Figure 8.3 le bouclier d'interface de SainSmart : les broches d'entrées-sorties sont disponibles sous forme de rangées avec un écartement régulier entre contacts. Il suffit d'utiliser des connecteurs avec broches à sertir comme ceux de la Figure 9.69 pour établir des connexions solides.

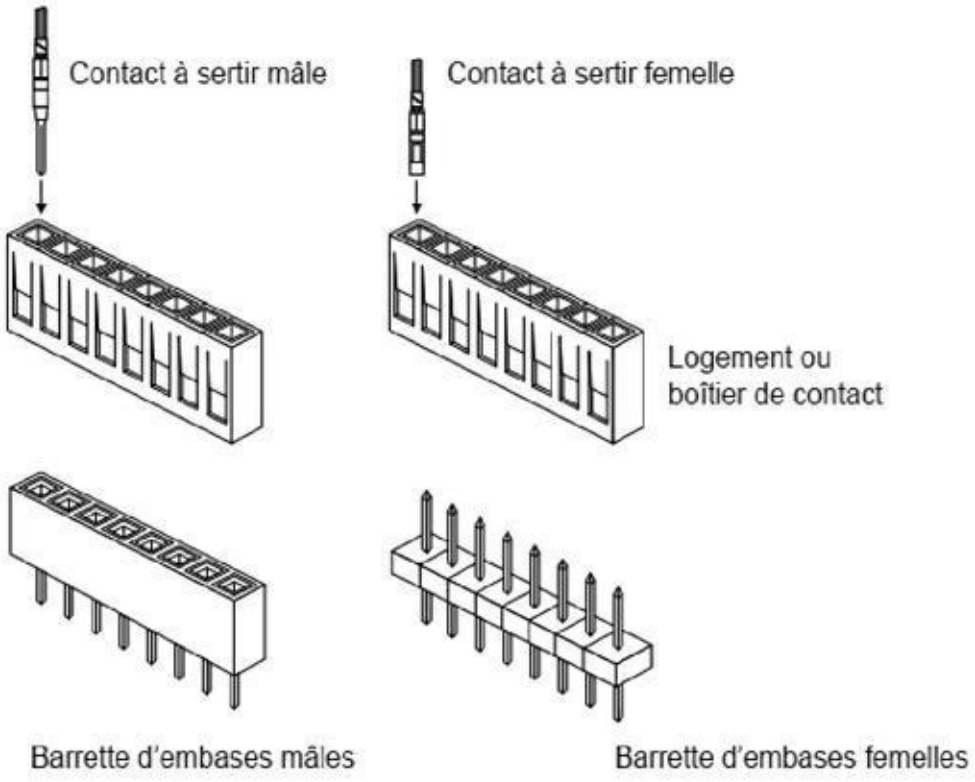


Figure 9.69 : Connecteurs mâles et femelles pour contacts à sertir.

Les logements pour la partie amovible, c'est-à-dire la fiche, sont disponibles dans différentes longueurs. Les fils straps sont terminés par un logement pour un seul contact mâle ou femelle.

Ce système de connexion très économique suffit largement lorsque vous avez besoin de connecter un ou deux modules sans craindre un débranchement. En revanche, cela suppose d'acquérir une pince à sertir et une pince à dénuder de bonne qualité (Figure 9.70). La Figure 9.71 montre comment sertir un contact ou une cosse.

Figure 9.70 : Pince à sertir les contacts.

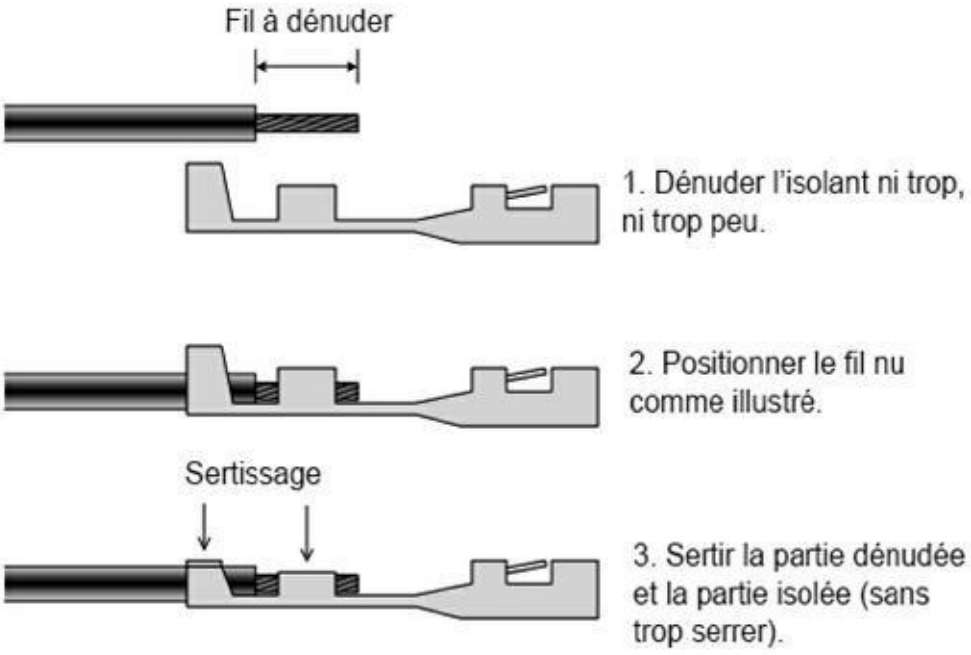


Figure 9.71 : Réalisation d'un sertissage.

La Figure 9.72 permet de voir un connecteur à trois fils branché sur un module capteur de température et d'humidité KY-015. Si vous craignez un débranchement, vous pouvez ajouter un peu de silicone entre le connecteur mâle et le circuit imprimé.

Figure 9.72 : Connexion d'un module capteur avec un connecteur à trois fils.

Choix de la technique de connexion

Vous devez choisir entre un système de connecteurs modulaires avec câbles préfabriqués et la fabrication de vos propres connecteurs, soit à partir de contacts à sertir, soit uniquement à partir de fils volants. Le choix sera différent d'un projet à l'autre, en fonction du temps que vous voulez consacrer aux connexions et de la robustesse que ces connexions doivent montrer. Tenez aussi compte du léger inconvénient qui consiste à se rendre dépendant d'un système de connexion.

Vous avez sans doute remarqué qu'il reste une méthode de connexion dont je n'ai pas du tout parlé : la soudure, qui est en réalité de l'étamage. Sauf si votre projet est définitif, cette technique est à utiliser en dernier ressort. De bonnes soudures entre des fils nus bien étamés et un circuit imprimé garantissent des connexions solides, mais elles sont moins simples à défaire, et l'aspect n'est pas toujours des plus heureux. Si vous voulez faire des tests et réutiliser rapidement vos modules pour autre chose, ce n'est pas la meilleure solution.

Cela dit, la Figure 9.12 montre des relais et un capteur de température soudés sur un bouclier de prototypage. J'ai procédé ainsi parce que je sais que ce montage va fonctionner dans un environnement peu accueillant à l'intérieur d'un radiateur électrique avec de fortes variations de température et des vibrations dues au ventilateur. Voilà pourquoi j'ai choisi d'effectuer des connexions définitives. Bref, vous déciderez de passer à la soudure lorsque le besoin deviendra évident.

Une liste de fabricants

Le tableau suivant présente une sélection de sociétés qui proposent des modules et des composants pour Arduino. J'ai pu tester des produits de quasiment chacune d'elles. Armé d'un simple navigateur Web, vous trouverez des dizaines d'autres fournisseurs de composants d'entrées-sorties compatibles Arduino.

Tableau 9.11 : Quelques fournisseurs de modules et composants.

Fournisseur	Adresse	Fournisseur	Adresse
Adafruit	www.adafruit.com	Mouser Electronics	www.mouser.
Amazon	www.amazon.com	RobotShop	www.robotsh
CuteDigi	store.cutedigi.com	SainSmart	www.sainsma
DealeXtreme (DX)	www.dx.com	DFRobot	www.dfrobot.
Seeed Studio	www.seeedstudio.com	Elecfreaks	www.elecfreal
SparkFun	www.sparkfun.com	Elechouse	www.elechous
Tindie	www.tindie.com	Freetronics	www.freetron
Tinkersphere	tinkersphere.com	ITEADStudio	www.itead.cc
Tronixlabs	tronixlabs.com	KEYES	en.keyes-robo
Trossen Robotics	www.trossenrobotics.com		

Pour conclure ce chapitre, voici trois questions qu'il est bon de se poser avant d'acheter des capteurs ou des modules :

1. Est-ce que le composant utilise une interface connue (connexion numérique directe, interface SPI, interface I2C, *etc.*) ?
2. Est-ce que la documentation technique est disponible ?
3. Existe-t-il déjà au moins une librairie de fonctions Arduino pour ce composant ?

Le point 3 peut être bloquant ou non, selon vos compétences en programmation. Personnellement, je vérifie surtout les points 1 et 2 parce que je considère avoir mieux à faire que me lancer dans de la rétro-ingénierie pour en savoir assez sur ce que je viens d'acheter. Je préfère prendre un composant équivalent un peu plus cher, mais accompagné de toutes les informations techniques nécessaires pour ne pas me retarder. Le but est de choisir l'élément qui va assurer la fonction demandée à un prix restant dans le budget prévu.

CHAPITRE 10
Boucliers sur mesure

Plus vous pratiquez Arduino, et notamment le microcontrôleur AVR, plus vous vous rendez compte de sa souplesse et de sa versatilité. Vous finissez par conclure qu'il existe un capteur ou une carte d'extension bouclier pour toutes les applications imaginables, sans compter les nouveaux composants qui viennent enrichir l'offre.

Il arrive pourtant qu'un projet ne trouve pas tous les éléments dont il a besoin. Vous cherchez pendant des heures sur le Web un bouclier offrant certaines capacités, pour vous rendre compte qu'il n'existe pas encore. Trois possibilités s'offrent à vous devant cette impasse : soit vous abandonnez et tentez de trouver une autre approche pour résoudre votre problème, soit vous prenez contact avec un concepteur qui développera la solution à vos frais, soit enfin, vous vous lancez dans la conception et la réalisation de votre propre carte d'extension. Nous allons dans ce chapitre passer en revue deux projets pour voir en détail comment créer une carte d'extension bouclier, puis un circuit spécifique, mais restant compatible Arduino au niveau logiciel.

Le premier des deux projets sera une carte bouclier enfichable (Figure 10.1). C'est un projet orienté vers les économies d'énergie, d'où son nom Greenshield. Le format de la carte reste conforme à celui des boucliers Arduino standard. Le montage utilise des composants montés en surface SMC ainsi que des potentiomètres, des relais et des diodes LED.

En combinaison avec une carte Arduino, le bouclier Greenshield permet de disposer d'un système de surveillance et de contrôle indépendant dans le domaine du jardinage ou de l'agriculture. Le projet peut également servir de point de départ pour une station météorologique automatisée, un système d'alerte aux orages ou même un thermostat (je montre par ailleurs dans le Chapitre 12 comment créer un thermostat programmable avec des modules et des capteurs du commerce).

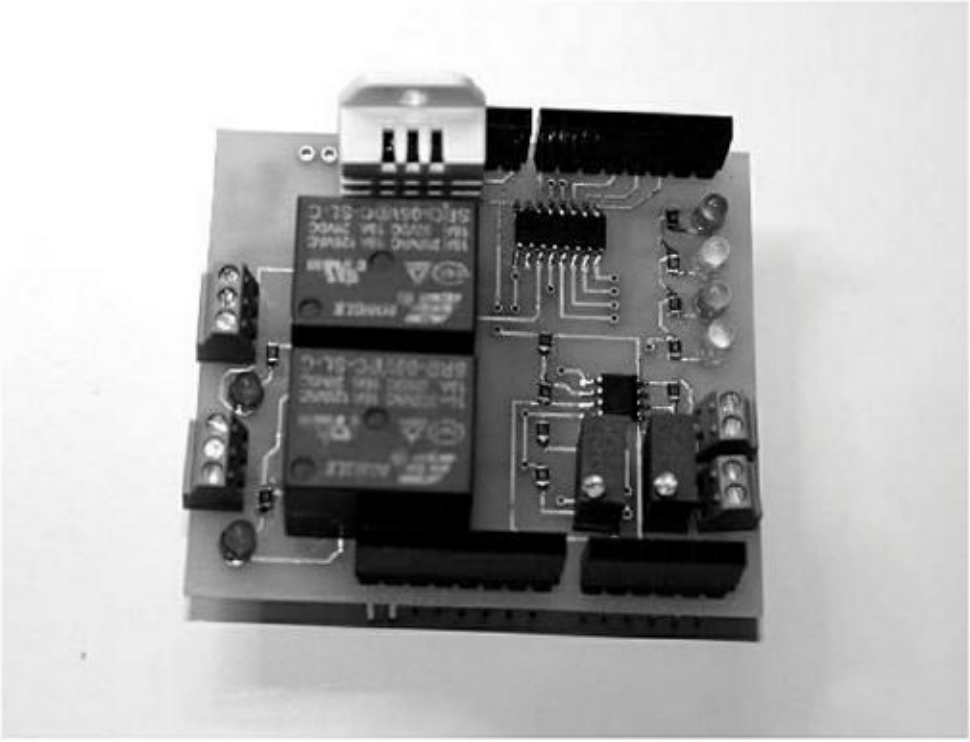


Figure 10.1 : La carte d’extension bouclier Greenshield.

Dans la seconde partie du chapitre, nous allons découvrir le Switchinator, un appareil basé sur un microcontrôleur ATmega 328 qui n’a plus besoin du logiciel de démarrage de l’Arduino. Il reste cependant programmable avec l’atelier IDE Arduino et un programmateur ICSP. Le circuit imprimé du Switchinator est visible dans la Figure 10.2.

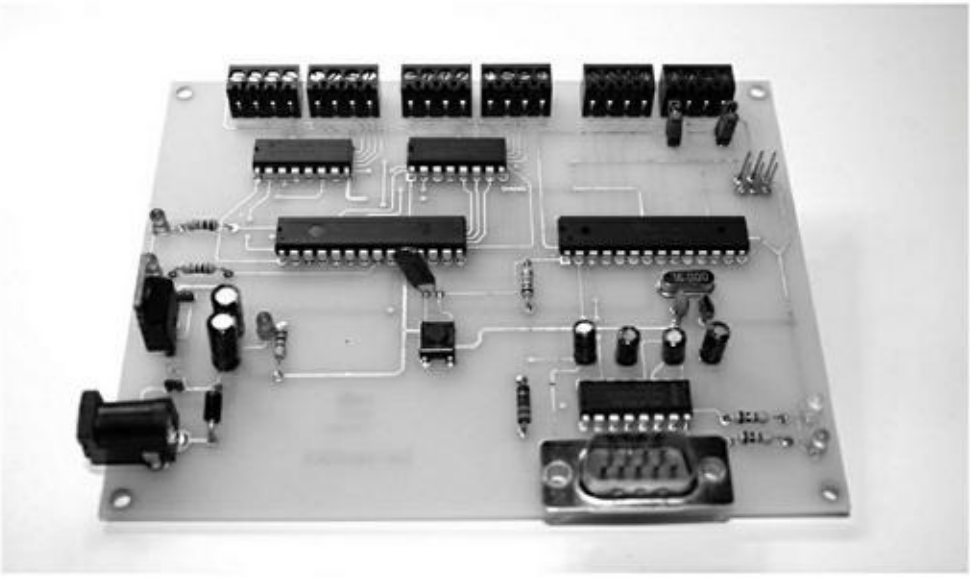


Figure 10.2 : Le circuit Switchinator.

Le projet Switchinator peut télécommander jusqu'à 14 appareils alimentés en courant continu, comme par exemple des relais ou des diodes LED à haute intensité. Il peut également contrôler jusqu'à trois moteurs pas à pas unipolaires ou d'autres charges en courant alternatif en y ajoutant les relais adéquats. Il se base sur une interface RS-232 et n'a plus besoin d'être relié à un ordinateur *via* un câble USB.

Switchinator réunit sur un circuit aux dimensions spécifiques quasiment tous les composants qui constituent une carte Arduino. Nous n'aurons pas à nous soucier du format exact des connecteurs et de la compacité d'implantation des composants sur le circuit. Les seules contraintes que nous aurons en termes de taille et d'aspect sont celles que nous voudrons bien nous imposer lors de la conception.

Il n'est pas très difficile de créer un circuit imprimé spécifique, s'éloignant du format d'une carte Arduino, à condition d'avoir un peu de patience et de bien planifier son travail. Il n'y aura aucune différence au niveau de la programmation. Le résultat vous permettra d'atteindre exactement vos objectifs, dans un format physique adapté à vos attentes.

Cela dit, la conception d'un circuit imprimé suppose un minimum de connaissances en électronique et en conception. Vous pouvez utiliser des logiciels gratuits ou très peu coûteux pour l'implantation des pistes, et faire fabriquer quelques exemplaires de votre circuit par un sous-traitant. Les projets de ce chapitre vont supposer un minimum de pratique en soudure, notamment pour les délicats composants montés en surface. Si vous avez déjà une expérience dans ce domaine, vous êtes paré. Dans le cas contraire, il n'est pas inutile d'apprendre à créer un schéma, à implanter les composants sur un circuit, à manier un fer à souder et à choisir les bons composants. Ce sont des compétences qui vous seront toujours utiles.

Pour le premier projet de carte d'extension, nous allons utiliser l'outil d'implantation et de capture de schéma Eagle. Pour le second projet, nous découvrirons l'outil Fritzing. Ce sont deux des outils les plus répandus. Ils sont tout à fait satisfaisants, sans compter qu'ils sont gratuits. En fait, Fritzing est gratuit. Pour Eagle, c'est la version d'entrée de gamme qui est gratuite, à condition d'accepter quelques limitations.

Avant de vous lancer dans la création d'un circuit spécifique, n'hésitez pas à vous doter d'un carnet de notes. Qu'importe son aspect : vous pouvez réunir des feuillets manuscrits et des photocopies dans un classeur à anneaux. Vous serez sans doute content au fur et à mesure de l'avancement dans le projet de pouvoir retrouver une information, et réutiliser vos notes dans un autre projet. Vous pouvez également tout stocker dans des fichiers disque sur votre PC. Mais il faut dans ce cas penser à toujours faire ses sauvegardes. Par ailleurs, le nombre de fichiers que l'on stocke sur un disque dur devient tellement énorme que l'on finit par ne plus savoir comment retrouver l'un d'eux en particulier (cela m'arrive personnellement bien trop souvent). Avec un carnet de notes physique, vous pouvez ajouter des remarques au fur et à mesure de votre progression dans l'acquisition des compétences au niveau des tests, de la fabrication ou du déploiement de vos projets. Et les stylos rouges ne servent pas qu'à corriger les notes en classe.

J'ai également prévu dans ce chapitre une liste de logiciels, de composants et d'éléments pour construire les circuits. Mes prérequis se limitent à un minimum d'expérience en électronique, ou à un désir suffisant d'en apprendre les bases. N'hésitez pas à jeter un œil à la liste d'outils de l'Annexe A. Voyez aussi les suggestions de lectures dans l'Annexe D. Enfin, visitez les sites Web et les nombreux tutoriels. Souvent, quelqu'un d'autre a réalisé le même genre de projet que vous et s'est montré suffisamment généreux pour partager son expérience avec le vaste monde.

N.d. T : Même sans être anglophone, vous pouvez tirer profit d'une visite des sites les plus fréquentés que sont Hackaday, Makezine, Adafruit, SparkFun et Instructables.

Je rappelle que l'objectif principal de ce livre est la description des matériels et des modules, senseurs et composants Arduino. Les logiciels utilisés dans ce chapitre ne sont montrés que pour aborder des points particuliers. Ils ne constituent pas des exemples de programmation pas à pas complets. Le code source complet des exemples se trouve sur le site de l'éditeur et sur le site de référence GitHub.

Se préparer à réussir

Comme dans tout projet qui demande d'y consacrer un minimum de temps, il faut planifier. Cette précaution s'applique tout autant à un projet d'appareil électronique qu'à la conception d'un logiciel complexe, la construction d'une maison, la fabrication d'une voiture de course ou la préparation d'une expédition polaire. Comme le dit le proverbe « Si tu oublies de planifier, tu planifies ton oubli ».

Dès qu'un projet possède un minimum d'envergure, il peut et doit être décomposé en plusieurs étapes. Voici par exemple les sept étapes principales de création d'un appareil électronique :

1. Définition des besoins
2. Planification
3. Conception
4. Prototypage
5. Test du prototype
6. Fabrication
7. Test et recette finale

Certains projets pourront avoir plus d'étapes et d'autres un peu moins, en fonction de l'objet. Voyons plus en détail les étapes ci-dessus.

Définition des besoins

En ingénierie, cette phase correspond à la définition des exigences (*requirements*), plus précisément des exigences fonctionnelles. Pour un projet assez simple, il suffit de décrire ce que l'on doit obtenir et à quoi va servir le projet. Il faudra aller plus en détail dans les projets plus complexes, comme c'est le cas d'une carte d'extension pour Arduino. Quelle que soit la complexité, il est toujours utile de noter l'objectif afin de guider l'orientation des étapes suivantes. Cette formalisation par écrit va également permettre de détecter des oublis ou des erreurs qui risqueraient sinon de rester masqués jusqu'à ce qu'il soit trop tard pour changer sans un important surcoût.

La définition du projet par écrit permet en outre de cadrer les tests du prototype et de l'appareil définitif. Un jeu d'exigences fonctionnelles correct doit clairement décrire ce que l'appareil est supposé pouvoir faire, et cela de telle manière que les procédures de test puissent être utilisées par une autre personne pour valider l'appareil.

L'écriture des exigences fonctionnelles peut sembler une activité peu enthousiasmante, alors que c'est un élément essentiel de toute bonne ingénierie. Si vous ne savez pas dans quelle direction avancer, comment saurez-vous que vous êtes arrivé au but ? Une bonne exigence, quel que soit le domaine, doit posséder les quatre attributs élémentaires suivants :

1. Elle doit être cohérente au niveau interne et par rapport au reste de la conception.
2. Sa signification ne doit pas être ambiguë.
3. Le texte doit être bref.
4. L'exigence doit être vérifiable, testable.

Voici une exigence testable : « L'appareil doit être capable d'amener 100 ml d'eau à 100 °C en 5 minutes. » En revanche, l'exigence suivante n'est pas testable : « L'appareil doit permettre de réchauffer de l'eau ». (Quelle température exactement ? En combien de temps ? Quelle quantité d'eau ?).

Planification

Lorsque le projet est assez simple et ne comporte pas de zones d'ombre, vous pouvez combiner planification et définition des besoins. Cependant, il faut dans tous les cas identifier et réunir les informations qui vont être nécessaires pour la phase de conception : fiches techniques des composants prévus, adresses des fournisseurs, outils de conception et de programmation, *etc.* Le but est d'arriver en début de phase de conception avec tout ce qui sera nécessaire pour prendre les bonnes décisions, pour pouvoir évaluer le temps requis pour l'approvisionnement et pour collecter les techniques d'utilisation de ces composants.

C'est également pendant la phase de planification que vous devez prévoir le mieux possible la durée des phases suivantes que sont le prototypage, les tests du prototype, la fabrication et les tests finaux. Vous allez pouvoir développer vos

hypothèses de façon réaliste, car vous aurez à ce moment collecté un grand nombre d'informations. Pourtant, ce ne seront encore que des hypothèses ; personne ne peut lire l'avenir. Il pourra toujours se produire des choses imprévues et certaines étapes d'un projet prendront plus de temps que ce qui a été imaginé, quel que soit le soin apporté à cette prévision. C'est ainsi que cela se passe dans le monde réel. C'est une des raisons pour lesquelles les responsables de projets multiplient leurs estimations de temps par deux ou par trois. Il est en général préférable de surestimer les délais que de les sous-estimer pour finalement livrer en retard.

Vous aurez intérêt à vous munir d'un outil de planification pour formaliser un planning le plus réaliste possible, mais également pour pouvoir suivre votre progression. Personnellement, l'outil que je préfère est le diagramme de Gantt. Les logiciels de gestion de projets permettent tous de créer de tels diagrammes avec une foule de détails, mais vous pouvez tout à fait vous contenter d'un diagramme simplifié comme celui de la Figure 10.3.

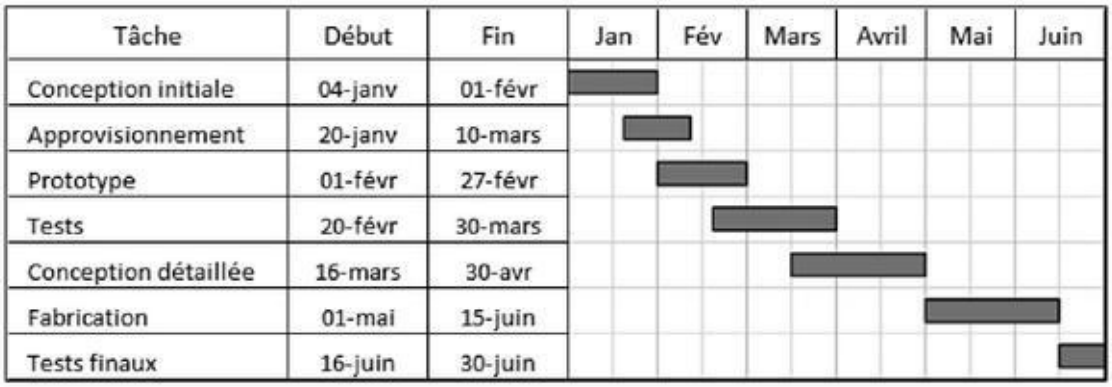


Figure 10.3 : Exemple de diagramme de Gantt d'un projet simple.

Dans les projets de faible complexité, les diagrammes très détaillés ne sont pas nécessaires, et ajoutent même de la complexité inutilement. Les points importants sont : 1) toutes les tâches nécessaires sont visibles dans le planning ; 2) le plan est réaliste au niveau temps et ressources humaines ; 3) le plan indique un objectif et une date de fin bien définis. Dans le diagramme d'exemple, vous pouvez constater que certaines tâches démarrent avant qu'une tâche précédente ne soit totalement achevée. Pour autant, on ne peut pas lire dans le diagramme les dépendances entre tâches, ce qui permettrait de visualiser le chemin critique. Mon expérience m'a montré que dans les petits projets ne réunissant qu'une

poignée de personnes, le chevauchement entre débuts et fins de tâche restitue bien ce qui se passe dans la réalité.

Conception

Dans un projet matériel, c'est lors de la cette phase que vous commencez à dessiner le circuit et choisissez le format physique. Vous ne devriez normalement pas avoir de problèmes à progresser dans cette phase, puisque vous disposez d'une définition claire des besoins et de toutes les informations de planification. La plupart des personnes pensent que la conception se consacre uniquement à la création du circuit. Pourtant, il y a d'autres activités dans cette phase, et notamment le choix des composants en fonction des contraintes de taille et de performances, l'évaluation des plages de fonctionnement électrique, les impératifs environnementaux (humidité, vibrations, température), et parfois même les problèmes d'interférences radioélectriques.

Toute création suppose un arbitrage. Parfois, vous privilégierez le coût de revient et la disponibilité des composants. Ou bien c'est la palette de fonctions qui sera le critère principal, comme lorsqu'un périphérique d'entrée doit permettre de remplir plusieurs fonctions. Parfois encore, c'est l'aspect esthétique qui va primer, notamment s'il n'y a pas de différence en termes de coût de revient par rapport à une solution plus banale. Parfois, et ce n'est pas le cas le plus rare, vous privilégierez des composants que vous connaissez déjà. Ce n'est pas toujours la meilleure raison, mais avancer en terrain connu est toujours rassurant.

Quel que soit le genre de projet (matériel, logiciel, infrastructure, *etc.*), la phase de conception va reboucler plusieurs fois. Ne vous attendez pas à trouver dès la première passe la conception définitive, sauf lorsque le problème à résoudre est vraiment évident, et encore. À vrai dire, la conception va être reprise plusieurs fois en même temps que la phase de prototypage qui la suit. Cela permet de détecter les problèmes, de mettre en lumière les solutions praticables et de peaufiner la conception. Ce fonctionnement itératif est fréquent en ingénierie. La finalisation de la conception dépend énormément de ce fonctionnement itératif.

Changer, et changer encore

Plusieurs révisions majeures m'ont été nécessaires pour créer le projet de générateur de signal décrit dans le Chapitre 11. Au départ, j'avais prévu de

pouvoir contrôler à distance toutes les fonctions du générateur au moyen d'une interface RS-232, avec une fonction de génération d'un motif numérique en parallèle. Mais je me suis rendu compte que la carte Arduino n'offrait pas suffisamment de broches d'entrée-sortie pour réussir cette approche sans y adjoindre une carte d'extension et une carte de communication RS-232. J'aurais pu résoudre une partie des problèmes en adoptant par exemple une roue de codage, mais chaque solution partielle apportait de nouveaux soucis. Il m'aurait été dans tous les cas nécessaire d'ajouter des poussoirs sur le générateur pour certaines des fonctions et l'afficheur LCD commençait à déborder d'informations.

Au lieu d'augmenter la complexité du matériel et du logiciel, j'ai décidé de simplifier. Le résultat n'utilise plus d'interruption logicielle, l'afficheur n'est plus surchargé de données incompréhensibles et je n'ai besoin que de deux circuits imprimés : la carte Arduino et le module DDS. L'appareil issu de cette conception assure un bon nombre de fonctions typiques d'un générateur de signal, même s'il ne permet pas de générer un motif aléatoire, ni d'être contrôlé à distance.

Prototypage

La création d'un prototype n'est pas obligatoire : vous n'en avez pas besoin pour fabriquer une carte d'extension simple sans aucun composant actif. Dans les autres cas, le prototype va vous permettre de valider votre conception et les performances telles qu'elles ont été annoncées en début de projet. Il n'est par exemple pas conseillé de passer directement à la création du circuit imprimé dans un projet qui combine un microcontrôleur AVR, un afficheur LCD, un circuit Bluetooth, un accéléromètre et un compas sur le même circuit. Cela fonctionnera peut-être la première fois, mais certains défauts masqués risquent de réapparaître une fois le circuit gravé en série et le sous-traitant payé. En construisant et en testant soigneusement le prototype, vous vous évitez de graves soucis, techniques et financiers.

Les problèmes identifiés pendant la phase de prototypage sont réinjectés dans la phase de conception pour l'améliorer. Le cas extrême est celui dans lequel les tests du prototype prouvent que la conception initiale est erronée et qu'elle doit être reprise du début. C'est ennuyeux, mais ce n'est pas un désastre, et cela arrive bien plus souvent qu'on le croit. Le vrai désastre serait de lancer la fabrication d'une centaine de circuits imprimés avant de détecter une erreur de

conception incontournable. En révisant la conception, vous vous évitez ce genre de mésaventure.

Tests du prototype

Les tests répondent pour l'essentiel à la question suivante : « Est-ce que l'appareil fait correctement ce qu'il doit faire, et est-ce qu'il le fait en toute sécurité et fiabilité ? » C'est la définition du projet qui doit servir de référence quant à ce que l'appareil doit réaliser, et dans quelles conditions de sécurité et de fiabilité. Il s'agit là d'un test fonctionnel élémentaire, mais qui peut devenir assez complexe. Cela dit, à moins de vouloir créer un projet destiné à être envoyé dans l'espace ou au fond des océans, ou d'un appareil dont la panne aurait d'énormes conséquences financières ou humaines, vous devriez pouvoir vous contenter des tests fonctionnels de base pour le prototype.

Il y a lieu de bien distinguer entre les critères « correct », « sûr » et « fiable » : ce n'est pas parce qu'un appareil fonctionne correctement qu'il est sûr. Il doit fonctionner sans introduire de risques inacceptables. Inversement, ce n'est pas parce qu'un appareil est sûr qu'il fonctionne correctement ou qu'il est fiable. Si l'appareil refuse de se mettre sous-tension, il est sûr et fiable, mais son fonctionnement n'est évidemment pas correct. Un appareil peut être correct et fiable, sans être sûr. Une scie circulaire peut couper correctement et de manière fiable des planches, et des doigts. Un appareil électrique qui souffre d'un court-circuit interne va émettre de façon fiable de la fumée ou des flammes dès que vous le mettez sous-tension, mais il ne sera évidemment ni correct, ni sûr.

Fabrication

Vous pouvez passer à la fabrication à partir du moment où vous avez bien testé le prototype en montrant que son comportement est conforme aux exigences fonctionnelles définies au départ. Dans cette phase, vous aurez peut-être à créer un circuit imprimé puis à y installer les composants. Il est également possible de travailler par interconnexion de plusieurs modules prémontés, le tout étant placé dans un boîtier ou incorporé à un plus grand système. En termes de délais, la fabrication peut poser quelques soucis. Un fournisseur peut se trouver en rupture de stock d'un composant essentiel ou vous imposer des délais pour fabriquer votre appareil. Lorsque l'on réalise un projet sur mesure, les causes de retard sont nombreuses.

Si vous n’avez en revanche besoin de construire qu’un exemplaire ou quelques-uns, et si vous fabriquez tout vous-même, vous aurez bien moins de soucis. N’hésitez cependant pas à prévoir assez de temps pour bien travailler. Même s’il ne vous manque qu’une poignée de vis, il faut compter le temps d’aller à la quincaillerie et d’en revenir, et cela en supposant que les pièces sont bien en stock. Évidemment, si vous aviez bien préparé le projet dans la phase de planification et de conception, vous devriez déjà disposer de tous les composants, du circuit imprimé à la moindre vis et au moindre bout de fil.

Tests de recette finale

Nous arrivons enfin aux tests finaux qui constituent la dernière phase avant de déclarer l’appareil prêt à l’emploi ou à la commercialisation. Vous avez déjà réalisé des tests avec le prototype, mais il faut aussi tester le produit final. Ce n’est nullement du temps perdu. On peut aisément implanter un composant à l’envers sur le circuit imprimé ou en intervertir deux. De plus, le comportement du montage peut dévier dans sa version circuit imprimée par rapport au prototype. Il est donc toujours conseillé de tester intensivement l’appareil dans son état définitif.

Vous pouvez compter deux étapes pour ces tests. Dans la première étape, vous vérifiez que l’appareil fonctionne aussi bien que le prototype. Vous appliquez les mêmes tests pour vous assurer que rien n’a changé. En ingénierie des logiciels, on appelle cela des tests de non-régression. La deuxième série de tests consiste à vérifier que l’appareil fonctionne correctement avec de véritables données en entrée et en sortie. C’est cette étape qui correspond aux tests de recette. Ces tests visent à répondre à la question : « Est-ce que l’appareil est acceptable pour son domaine d’utilisation prévu ? ».

Cartes boucliers personnalisées

Dans le monde Arduino, il y a trois formats physiques principaux : le format de base, le format étendu et le format Mega. Le format de base est celui d'origine qui correspond à la version R2 de la carte Arduino (qui n'est plus commercialisée). On le trouve sur les modèles Duemilanove et Uno version R2. Il reste cependant le format standard pour tous les boucliers. Vous pouvez sans crainte opter pour le format étendu qui est celui de la carte Arduino Uno R3 actuellement en vigueur. Il offre quelques broches de connexion supplémentaires. C'est également le format de la carte Leonardo. Vous pouvez enfin opter pour le format Mega, celui de la carte Arduino Mega. Vous pouvez même créer une carte d'extension compatible au niveau des connecteurs, mais avec un format différent. C'est notamment le cas des cartes qui portent de nombreux relais ou des radiateurs encombrants. Leur format physique découle des composants qu'elles hébergent, et s'éloigne donc de l'encombrement standard des cartes Arduino sur lesquelles elles viennent s'enficher.

Une approche radicalement différente consiste à se libérer de toutes les contraintes d'encombrement en créant un grand circuit imprimé, avec deux rangées de connecteurs, de sorte que ce soit la carte Arduino qui vienne s'enficher à l'envers sur la carte d'extension. Observez par exemple la Figure 10.4. C'est une carte de pilotage de machine-outil à commande numérique Roland SRM-20. Vous pouvez en apprendre plus sur le site de Nadya Peek (`infosyncratic.nl`).

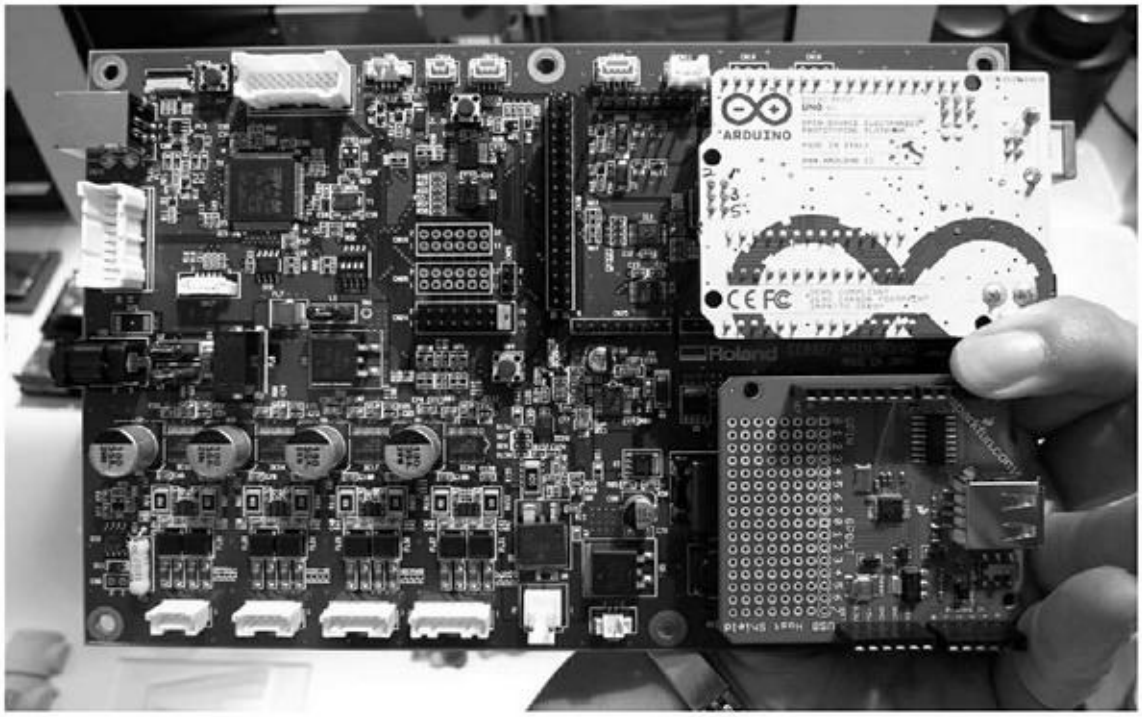


Figure 10.4 : La carte Arduino est enfichée sur la carte d'extension (image fournie par Nadya Peek).

Si votre projet est destiné à être commercialisé, vous choisirez de préférence le format de base ou étendu, ce qui le rendra compatible avec toutes les cartes Uno et Leonardo, et donc aussi avec les cartes Mega. J'ai décrit dans le Chapitre 4 toutes ces cartes en précisant leurs dimensions et le brochage des connecteurs.

Les cartes Arduino miniatures que sont la Nano, la Mini et la Micro sont différentes des autres. Toutes leurs broches de connexion sont situées sur le dessous du circuit, ce qui fait ressembler la carte à une sorte de grosse puce de circuit intégré. Pour ajouter une carte d'extension à l'une de ces cartes, il faut se procurer une carte adaptateur, comme celle montrée en Figure 10.5. Cette carte reporte les broches de sortie de la carte miniature vers les connecteurs qui permettent d'enficher une carte bouclier.

Figure 10.5 : Carte adaptateur d’interface pour Arduino Nano.

Vous pouvez voir dans la Figure 10.5 que le circuit comporte de nombreuses broches mâles qui sont autant de points de connexion supplémentaires vers la carte Nano. Vous pourriez envisager d’y enficher des connecteurs femelles pour une carte d’extension, mais la carte Nano située au milieu dépasse trop. Il faudrait dans ce cas utiliser des rallonges pour les broches de la carte d’extension. Une autre solution consisterait à dessouder les connecteurs actuels de la carte Nano pour les remplacer par des versions moins longues. Je ne m’étendrai pas plus dans ce chapitre sur ce genre de manipulation.

D’ailleurs, il n’est généralement pas intéressant d’adopter une carte au format de la Nano, de la Mini ou de la Micro pour y ajouter une carte d’extension. Cela dit, le Chapitre 12 donne un exemple dans lequel cette approche particulière est raisonnable. Il est en revanche tout à fait censé de considérer ces petites cartes un peu comme de gros circuits intégrés, et de s’en servir comme des composants sur un circuit imprimé de plus grande taille.

Compatibilité physique

Sachez que la carte Duemilanove, et sans doute certaines autres, comporte deux condensateurs qui peuvent entrer en collision avec certaines des broches d’alimentation et le connecteur des entrées analogiques de certaines cartes boucliers conçues au départ pour le format étendu R3. Certaines cartes compatibles Uno possèdent aussi des composants qui peuvent entrer en collision avec les connecteurs de la carte bouclier.

Sachez également que le connecteur USB jack de type B qui équipe la Duemilanove et les Uno peut provoquer un court-circuit avec une carte bouclier. Méfiez-vous également du jack d'alimentation de ces cartes. Du fait qu'il est en plastique, il ne risque pas de provoquer de court-circuit, mais il peut empêcher la carte de bien être enfoncée. La Figure 10.6 montre ce problème d'enfichage entre une carte Duemilanove et une carte Ethernet.

Figure 10.6 : La carte Duemilanove surmontée d'un bouclier.

Si vous concevez le circuit imprimé d'une carte d'extension, tenez compte de la cohabitation nécessaire avec la carte qui va se trouver dessous la vôtre et/ou au-dessus d'elle.

Lorsque vous concevez une carte d'extension bouclier, tenez compte des deux endroits de la carte d'accueil Arduino qui ne sont plus accessibles une fois le bouclier en place. C'est notamment le cas du bouton de Reset, des diodes LED soudées sur le circuit et du groupe de broches ICSP. Certaines cartes éliminent ce problème en répétant les broches de la carte Arduino. D'autres cartes ne reportent pas les broches inaccessibles de l'Arduino sur la carte, et c'est notamment le cas des cartes d'afficheurs LCD. En revanche, elles reportent généralement le bouton Reset. C'est particulièrement utile avec un afficheur LCD, qui sera *a priori* la carte la plus au-dessus dans un empilement.

Empilement de boucliers

Un des grands avantages du format Arduino est de permettre d'enficher plusieurs cartes d'extension les unes sur les autres. Vous pouvez par exemple créer un empilement qui part d'une carte Arduino, par exemple une Uno, sur laquelle vous placez une carte d'extension mémoire pour carte SD, puis une carte d'entrées-sorties, puis une carte afficheur LCD ou TFT. Vous disposez ainsi en quelques instants d'un appareil de collecte de données élémentaires (data login).

Il faut bien sûr réfléchir à l'ordre d'enfichage des cartes, que ce soient des cartes d'extension ou des cartes de capteurs. Vous devez notamment vérifier l'encombrement en hauteur. Si une carte dépasse trop, vous ne pourrez pas enficher une autre carte au-dessus. Dans ce cas, cette carte devra toujours être la dernière de la pile.

Il existe deux formats d'enfichage : par décalage et par extension. Dans le cas du décalage, les connecteurs femelles sur le dessus de la carte sont déportés par rapport aux broches mâles du dessous de la carte, en général vers l'extérieur. Le décalage est réalisé vers l'extérieur d'un côté et vers l'intérieur de l'autre pour que l'écartement reste le même.

La Figure 10.7 montre une carte d'entrées-sorties enfichée sur une carte Duemilanove. Vous remarquez que la carte d'extension utilise le format décalé. Elle n'est donc pas alignée avec la carte qui la reçoit. Il faut en tenir compte si vous prévoyez de protéger votre projet dans un boîtier. Vous ne pourrez peut-être plus utiliser des vis de fixation pour protéger le montage des vibrations. Par ailleurs, cette carte d'extension ne reporte par les broches ICSP, ce qui empêche de pouvoir se servir de cette fonction. Enfin, les connecteurs sur les bords du bouclier entrent en collision avec le connecteur USB et celui d'alimentation de la carte Arduino. Il faut donc prévoir un peu d'isolant.

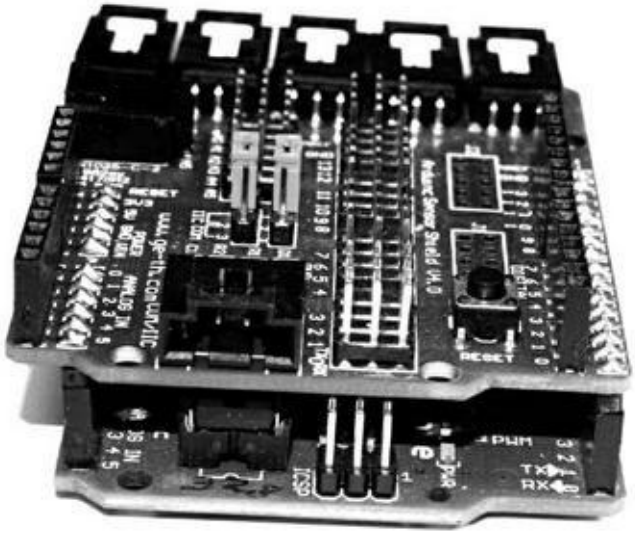


Figure 10.7 : Exemple de cartes empilées avec décalage des connecteurs.

L’autre format d’enfichage se base sur des broches de connexion allongées qui traversent le circuit imprimé. Les broches sont assez longues pour permettre une bonne connexion avec la carte du dessous, pendant que les broches femelles du connecteur permettent d’enficher un autre bouclier par-dessus, le tout restant en alignement. De nombreux boucliers utilisent ce format. Je vous conseille de lui donner votre préférence.

Un autre sujet à prendre en compte est la capacité d’utilisation des broches de l’Arduino. Les broches qui sont utilisées par la carte d’extension ne le sont plus pour d’autres cartes. Il faut éviter les cartes qui rendent inutilisables sans raison valable les broches ICSP et I2C, car elles peuvent servir à une carte mémoire, à une extension d’entrées-sorties, à une carte Bluetooth, Gigabit Ethernet ou GSM. Lorsque vous créez une carte d’extension, ne bloquez par les broches ICSP ou I2C, sauf si vous en avez absolument besoin pour votre projet.

Vous rencontrerez enfin des boucliers qui semblaient intéressants au départ, mais qui s’avèrent bien moins intéressants dans la pratique. Les cartes d’entrées-sorties offrent par exemple en général plusieurs borniers, mais ceux-ci deviennent inaccessibles dès que vous enfichez un autre bouclier par-dessus. D’autres cartes comportent des plots à souder qui entrent en conflit avec les broches ICSP de la carte Arduino. Ces petits désagréments ne sont pas rares. Il n’est hélas pas toujours possible de savoir à l’avance si un certain bouclier va poser problème. Le seul moyen d’en avoir le cœur net est bien sûr d’en acquérir un et de l’essayer sur la carte qu’il doit venir épauler.

Alimentation électrique

Si votre projet est une carte d'extension sans aucun composant actif, donc uniquement constitué de connecteurs, d'interrupteurs et de résistances, vous n'aurez sans doute pas beaucoup de soucis d'alimentation. En revanche, dès que la carte comporte des circuits actifs, ne serait-ce que des diodes LED, il faut bien réfléchir à son alimentation et à la source de celle-ci. Même une simple diode LED a besoin d'un minimum de puissance, et vous pouvez facilement surcharger les sorties d'un microcontrôleur de façon irrémédiable.

De façon générale, dès qu'un bouclier héberge des relais ou des connecteurs sur lesquels vous allez brancher des consommateurs qui demandent plus de quelques milliampères, il faut prévoir des circuits pour piloter la charge. Vous pouvez bien sûr faire fonctionner votre bouclier en l'alimentant de façon indépendante de la carte Arduino. Cela dit, vous aurez peut-être à faire face à quelques soucis au niveau des dialogues entre la carte Arduino et le bouclier.

Si vous alimentez le bouclier de façon indépendante, il faut bien faire attention à la masse. Sur une carte Arduino, il y a trois prises de masse : deux du côté du connecteur des entrées analogiques et une du côté des broches numériques. Pour vous prémunir des boucles de masse et des bruits parasites, vous n'utiliserez qu'une seule des broches de masse de la carte Arduino pour la référence de masse du signal de la carte d'extension. Cela dit, ce genre de précaution ne deviendra indispensable que lorsque la carte d'extension hébergera des amplificateurs opérationnels à fort gain ou des circuits à haute fréquence. Prenez cependant l'habitude de faire attention à la masse, cela vous évitera des problèmes difficiles à diagnostiquer à cause des symptômes étranges par lesquels ils se révèlent.

Le bouclier Greenshield

Dans cette section, nous allons concevoir et fabriquer une carte d'extension bouclier pour illustrer les différentes phases de réalisation d'un bouclier Arduino. Notre projet va permettre de surveiller l'humidité de l'air, du sol, la température et la lumière ambiante. Le principal domaine d'utilisation de ce projet est la culture sous serre. Il est tout à fait possible, avec un boîtier étanche et quelques cellules solaires, de vous en servir en plein champ pour surveiller la pousse de vos navets et autres panais. Les périodes de sécheresse semblant se répéter dans nos contrées, il n'est pas inutile de pouvoir vérifier l'humidité du sol, en plus de la température et de l'humidité de l'air. Vous pouvez ainsi optimiser l'arrosage tout en garantissant une bonne croissance à vos plants. En ajoutant une carte d'émission-réception sans fil, vous pourrez même rester dans votre salon tout en surveillant la pousse depuis un téléphone, une tablette ou un micro-ordinateur.

Le nom que j'ai choisi pour le projet, Greenshield (bouclier vert), semble approprié. Du fait que le projet reste simple, nous combinons les phases de définition des besoins et de planification en une seule. Le principal défi au niveau matériel va consister à faire tenir quelques composants un peu volumineux (les deux relais et la sonde DHT) sur notre petit circuit imprimé.

Objectifs du projet

Ce projet a donc pour objectif la création d'une carte de surveillance autonome à distance de plusieurs paramètres que sont la température, l'humidité de l'air, celle du sol et la luminosité. Deux relais vont pouvoir être déclenchés lors du franchissement de certaines limites des données reçues des capteurs.

Grâce à ces relais, vous allez pouvoir contrôler par exemple un robinet d'arrosage ou un ventilateur, voire des projecteurs pour compenser le manque d'ensoleillement par temps couvert. Le circuit comporte en outre six diodes LED : deux pour signaler le passage par un seuil et un plafond d'humidité de l'air, deux autres pour le seuil et le plafond de moiteur du sol et enfin une diode LED pour témoigner de l'activation de chacun des deux relais.

Le logiciel fonctionne selon un protocole de commande/réponse. Il va gérer un tableau de fonctions qui sont mises en correspondance entre les relais et

certaines valeurs limites lues par les capteurs. Vous pourrez tout à fait récupérer les données des capteurs sur un ordinateur pour contrôler directement les relais.

Définition et planification

Notre bouclier réutilise un capteur double de température et d'humidité de l'air DHT22, une résistance de type photorésistance (voir le Chapitre 9) pour gérer la luminosité et enfin une sonde de moiteur mesurant la conductivité de la terre. Les deux relais vont pouvoir commander des appareils ou des circuits externes de façon automatique ou à la demande.

Voici les capteurs d'entrée de données :

- Température de l'air
- Humidité relative de l'air
- Moiteur relative du sol (humidité du sol)
- Niveau de luminosité

Voici les voies de sortie de données et de contrôle :

- Deux relais pilotés par le logiciel d'une capacité de 10 A
- Quatre diodes LED pour les valeurs limites d'humidité et de moiteur
- Deux diodes LED pour signifier l'activité des relais

Voici les interfaces électriques :

- Deux borniers à deux plots pour la sonde de moiteur
- Un bornier à deux plots pour la photorésistance
- Un bornier à trois plots pour chacun des deux relais, afin de gérer les sorties normalement fermées (NC), normalement ouvertes (NO) et fermées (C)
- Une alimentation en courant continu +5 V qui provient de la carte Arduino

Et voici l’interface de contrôle :

- Protocole commande/réponse contrôlé par l’hôte
- Lecture des capteurs disponibles à la demande
- Prise de contrôle des relais par l’hôte

Tous les composants sont réunis sur une carte bouclier au format standard de base, dont les dimensions ont été fournies dans le Chapitre 4. Nous utilisons principalement des composants montés en surface, à l’exception bien sûr des borniers, des relais et du capteur DHT22.

Conception

Le projet Greenshield a été conçu pour ne pas avoir besoin d’interface utilisateur, c’est-à-dire ni afficheur, ni boutons de contrôle. Autrement dit, avec la carte Arduino, l’ensemble va pouvoir fonctionner de façon autonome. Vous pouvez cependant le relier à un micro-ordinateur qui deviendra son hôte afin de récupérer les paramètres de fonctionnement, les données lues par les capteurs, et éventuellement contrôler vous-même les relais.

Le projet est autonome au sens où il va être capable d’activer de lui-même les relais lorsque certaines conditions sont réunies, en termes d’humidité, de moiteur ou de luminosité. Le logiciel reste cependant en mesure de réagir à des commandes émises par un micro-ordinateur pour modifier les seuils et plafonds et pour activer directement les relais. Lorsqu’il va recevoir une commande, la réponse qu’il va renvoyer indiquera les niveaux actuels de température, d’humidité, de moiteur, de luminosité et l’état courant des relais. Toutes les interactions entre l’hôte et le projet seront de type commande/réponse.

Le logiciel du projet sera entièrement conçu dans l’atelier logiciel Arduino IDE. La machine hôte qui va servir à rédiger et compiler le code exécutable puis à le télécharger servira également d’interface de test une fois le code démarré sur la carte Arduino. Plus tard, vous pouvez envisager de créer une application d’interface par exemple en langage Python. Sous Windows, vous disposez également des émulateurs du type TeraTerm (ce dernier est livré avec un excellent outil de script, et je le recommande).

L’outil de schéma et de circuit imprimé Eagle

J’ai choisi d’utiliser l’outil Eagle pour créer le schéma de principe et le dessin du circuit imprimé de ce projet. Vous pouvez télécharger la version gratuite sur le site de la société Autodesk (qui a racheté ce logiciel en 2016). Il est fourni d’office avec la plupart des distributions Linux, mais sans doute dans une version qui n’est pas la plus récente. La version gratuite d’Eagle est limitée en richesse fonctionnelle de la façon suivante :

Le format maximal du circuit est de 200 mm par 80 mm.

Deux couches seulement sont possibles (dessus et dessous).

L’éditeur de schéma ne permet de créer que deux feuilles.

Le format standard ou étendu d’une carte Arduino étant égal à 69 mm sur 53, ces contraintes ne nous gênent pas. En revanche, vous ne pouvez pas créer une carte d’extension pour le format Mega, dont les dimensions sont supérieures aux limites d’Eagle. La limitation à deux couches n’est généralement pas un problème. Vous n’aurez absolument besoin de plus de deux couches que pour les projets traitant de la vidéo ou des signaux à haute fréquence, afin d’ajouter des couches pour la masse et l’alimentation.

Au besoin, vous pouvez vous rendre sur le site de SparkFun qui offre quelques tutoriels bien détaillés pour mettre en place le logiciel Eagle. Voyez dans ce cas les adresses suivantes :

`http://bit.ly/sparkfun-eagle`

`http://bit.ly/sparkfun-using-eagle`

Implantation

Intéressons-nous d’abord au diagramme fonctionnel représenté dans la Figure 10.8. Il permet de voir les principales fonctions que la carte d’extension doit embarquer.

Vous constatez qu’aucune des fonctions matérielles n’interagit directement dans cette figure. Les capteurs, les diodes LED et les relais ne constituent que des extensions qui viennent se connecter aux ports d’entrées-sorties de la carte Arduino.

Le capteur d’humidité et de température est directement soudé sur le circuit, alors que la photorésistance et le capteur de moiteur sont installés plus loin. Nous prévoyons à cet effet deux petits borniers, ce qui évite d’avoir à sortir le fer à souder ou la pince à cosses.

Notez que cette carte est prévue pour être placée en dernier d’une pile à cause de la hauteur des relais et du capteur de température et d’humidité qui dépassent suffisamment pour interdire l’ajout d’une autre carte d’extension par-dessus.

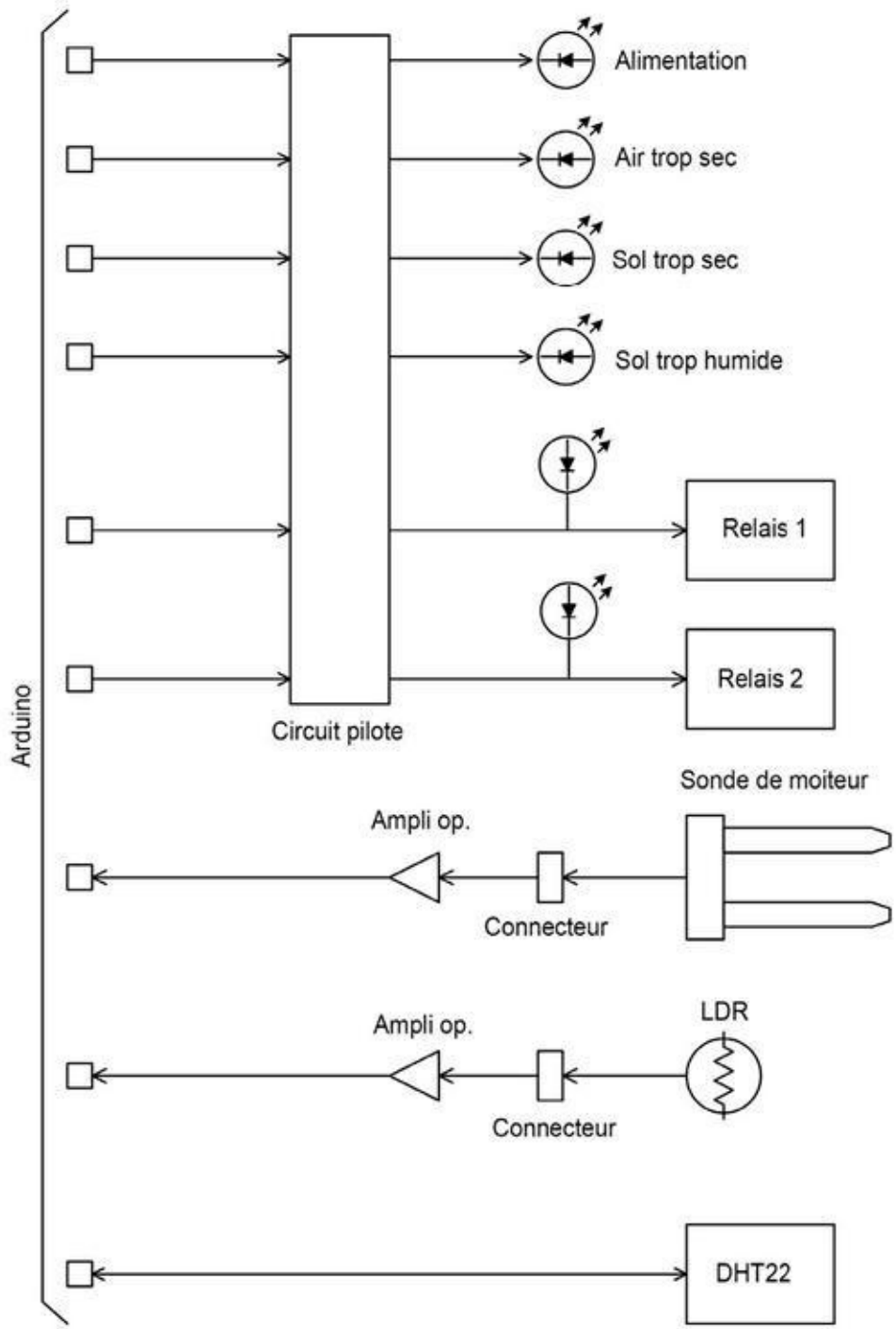


Figure 10.8 : Diagramme fonctionnel du projet Greenshield.

Schéma de principe

Le projet Greenshield n’est pas très complexe, comme le prouve le schéma de la Figure 10.9. Nous nous servons d’un circuit ULN2003A pour piloter les diodes LED et les deux relais. Nous traitons les tensions d’entrée obtenues du capteur de moiteur et de la photorésistance avec un double ampli opérationnel pour alimenter ensuite le convertisseur analogique/digital du microcontrôleur AVR.

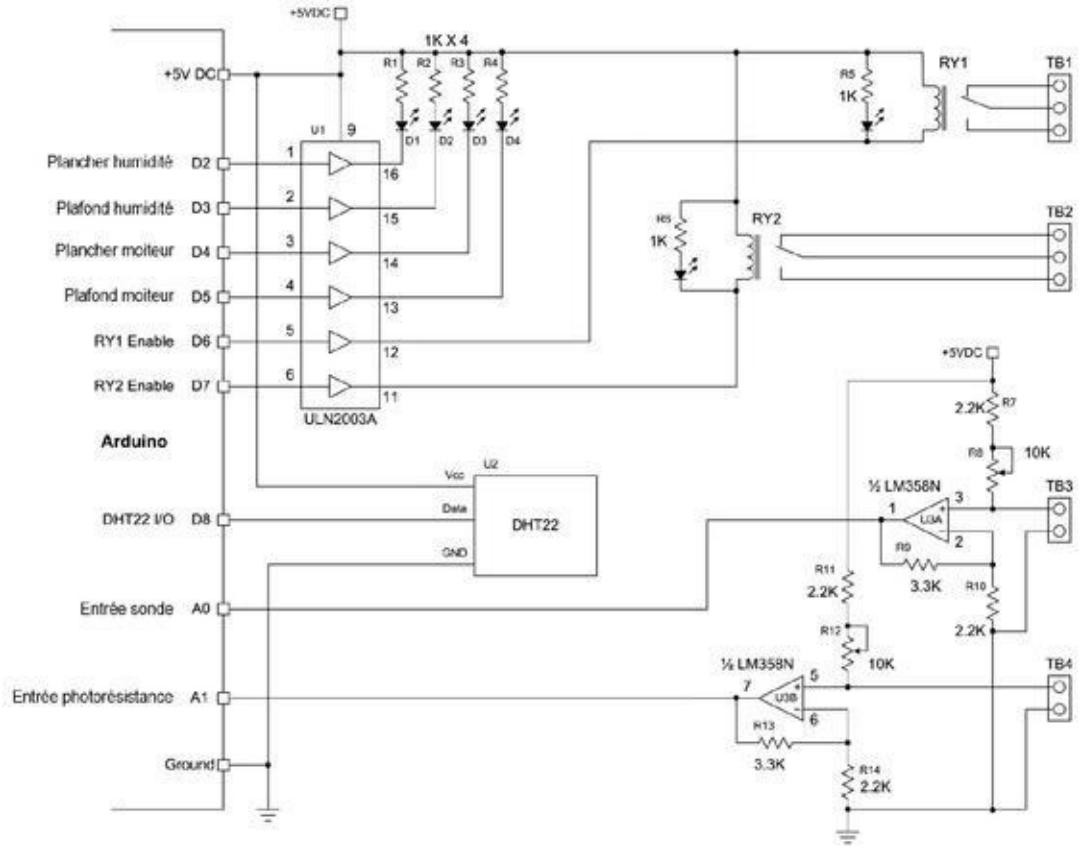


Figure 10.9 : Schéma de principe du projet Greenshield.

Les capteurs en entrée des amplis opérationnels se comportent comme des résistances variables et constituent des diviseurs de tension grâce aux deux potentiomètres de réglage. Ces derniers peuvent être ajustés pour obtenir la meilleure réponse possible des amplis opérationnels sans risquer de dépasser les limites de tension dans un sens ou dans l’autre.

J’ai pris soin dans ce projet d’exploiter la carte Arduino en évitant d’utiliser certaines broches numériques et analogiques essentielles, ce qui permet d’autres extensions. Le Tableau 10.1 liste les broches Arduino utilisées avec leur affectation.

Tableau 10.1 : Broches Arduino utilisées par Greenshield.

Broche	Fonction	Broche	Fonction
D2	ULN20031 canal 1	D7	ULN20031 canal 6
D3	ULN20031 canal 2	D8	Entrée DHT22
D4	ULN20031 canal 3	A0	Entrée sonde moiteur
D5	ULN20031 canal 4	A1	Entrée photorésistance
D6	ULN20031	Canal 5	

Vous constatez que nous n'utilisons pas les quatre broches SPI (D10 à D13) qui restent ainsi disponibles pour un bouclier utilisant SPI. De même, les deux broches D0 et D1 restent libres pour une éventuelle interface RS-232 à la place de la liaison USB. Enfin, les deux broches analogiques A4 et A5 restent disponibles pour une application I2C.

Ce schéma de principe nous permet d'établir la nomenclature complète des composants (Tableau 10.2).

Tableau 10.2 : Liste des composants du projet Greenshield.

Quantité	Type	Description	Quantité	Type	Descrip
2	SRD-05VDC-SL-CSongle	Relais 5 A	4	2,2 kΩ	Résistar W
1	DHT22	Capteur humidité/temp	2	3,3 kΩ,	Résistar W
1	Générique	Photorésistance	2	10 K trim	Potenti de régla
1	SainSmart	Sonde sol	2		

				0.1” (2.54 mm)	Bornier plots
6	3 mm	LED	2	0.1” (2.54 mm)	Bornier plots
1	LM358N	Ampli op	2	0.1” (2.54 mm)	Connec broche
1	ULN2003A	Pilote IC	2	0.1” (2.54 mm)	Connec broche
6	1kΩ	Résistance 1/8W	1	Spécifique	Circuit imprim bouclie

Le logiciel

Trois fonctions principales sont incarnées par le logiciel du projet : la captation des données des capteurs, l’analyse des commandes reçues et la génération des réponses, et le paramétrage du déclenchement des relais. La première fonction se charge de collecter les données des quatre capteurs (température, humidité, moiteur et luminosité), en stockant ces valeurs pour utilisation par d’autres fonctions du logiciel. La fonction d’analyse des commandes récupère les chaînes de commande émises par un PC hôte puis produit des réponses en conformité avec le protocole commande/réponse (que nous allons décrire). Enfin, la fonction de sortie contrôle l’état des relais en fonction des données, des valeurs critiques, et des commandes reçues.

Le Tableau 10.3 présente toutes les commandes du protocole commande/réponse qui permet de faire dialoguer le projet Greenshield avec un ordinateur hôte. Rappelons que le projet ne fait que répondre à l’hôte ; il ne va jamais débuter un dialogue.

Tableau 10.3 : Commandes du protocole commande/réponse du projet Greenshield.

Commande	Réponse	Description
AN:*n*:?	AN:*n*:*val*	Lit entrée analogique brute
GT:HMX	GT:HMX:*val*	Lit valeur max humidité
GT:HMN	GT:HMN:*val*	Lit valeur min humidité
GT:LMX	GT:LMX:*val*	Lit valeur max luminosité
GT:LMN	GT:LMN:*val*	Lit valeur min luminoisté
GT:MMX	GT:MMX:*val*	Lit valeur max moiteur sol
GT:MMN	GT:MMN:*val*	Lit valeur min moiteur sol
GT:TMX	GT:TMX:*val*	Lit valeur max température
GT:TMN	GT:TMN:*val*	Lit valeur min température
HM:?	HM:*val*	Renvoie température courante
RY:*n*:?	RY:*n*:*n*	Renvoie état relais n
RY:*n*:1	OK	Bascule relais sur n ON
RY:*n*:0	OK	Bascule relais sur n OFF
RY:A:1	OK	Bascule tous les relais sur ON
RY:A:0	OK	Bascule tous les relais sur OFF
RY:*n*:HMX	OK	Bascule relais n sur ON si humidité >= max
RY:*n*:HMN	OK	Bascule relais n sur ON si humidité <= min
RY:*n*:LMX	OK	Bascule relais n sur ON si luminosité >= max

RY:*n*:LMN	OK	Bascule relais n sur ON si luminosité <= min
RY:*n*:MMX	OK	Bascule relais n sur ON si moiteur sol >= max
RY:*n*:MMN	OK	Bascule relais n sur ON si moiteur sol <= min
RY:*n*:TMX	OK	Bascule relais n sur ON si température >= max
RY:*n*:TMN	OK	Bascule relais n sur ON si température <= min
ST:HMX:*val*	OK	Définit valeur max humidité
ST:HMN:*val*	OK	Définit valeur min humidité
ST:LMX:*val*	OK	Définit valeur max luminosité
ST:LMN:*val*	OK	Définit valeur min luminosité
ST:MMX:*val*	OK	Définit valeur max moiteur sol
ST:MMN:*val*	OK	Définit valeur min moiteur sol
ST:TMX:*val*	OK	Définit valeur max température
ST:TMN:*val*	OK	Définit valeur min température
TM:?	TM:*val*	Renvoie température courante
RY:*n*:?	RS:*n:n*	Renvoie état relais n
AN:*n*:?	AN:*n:val*	Lit entrée analogique brute
TM:?	TM:*val*	Renvoie dernière température DHT22
HM:?	HM:*val*	Renvoie dernière humidité DHT22

Pour dialoguer avec le projet, vous pouvez vous servir du moniteur série qui est disponible dans l'atelier Arduino IDE ou bien quitter l'atelier et vous connecter directement au port USB sur lequel est connectée la carte Arduino. Cette seconde approche est celle à adopter si vous voulez créer une interface utilisateur pour contrôler le projet. La solution normale consiste à paramétrer le logiciel du projet sur l'Arduino puis à laisser fonctionner le montage une fois déconnecté.

Les commandes d'interrogation

Le projet prévoit quatre commandes d'interrogation (Tableau 10.4) qui permettent à une machine hôte de connaître l'état des deux relais, la dernière valeur lue par la photorésistance ou le capteur de moiteur et la dernière valeur de température et d'humidité du capteur DHT22.

Tableau 10.4 : Commandes d'interrogation du projet Greenshield.

Commande	Réponse	Description
RY:n:?	RS:n:n	Renvoi état relais n
AN:n:?	AN:n:val	Lecture entrée analogique brute
TM:?	TM:val	Lecture température DHT22
HM:?	HM:val	Lecture humidité DHT22

Commandes de contrôle des relais

Vous pouvez contrôler les relais du projet par des commandes, au nombre de quatre pour ouvrir ou fermer chacun des relais, et même les désactiver en même temps. Le Tableau 10.5 présente ces quatre commandes de prise de contrôle.

Tableau 10.5 : Commandes de prise de contrôle des relais.

Commande	Réponse	Description
RY:n:1	OK	Bascule relais n sur ON
RY:n:0	OK	Bascule relais n sur OFF

RY:A:1	OK	Bascule tous les relais sur ON
RY:A:0	OK	Bascule tous les relais sur OFF

Le fait de prendre le contrôle d'un relais par l'une de ces commandes supprime tous les armements programmés. Pour qu'un relais soit à nouveau contrôlé par un maximum ou un minimum mesuré, il faut lui envoyer une nouvelle commande de définition.

Commandes de programmation du déclenchement des relais

Vous pouvez piloter les deux relais en fonction d'une condition de plancher ou de plafond pour l'humidité, la luminosité, la moiteur ou la température. Les commandes de programmation correspondantes sont présentées dans le Tableau 10.6. La plus récente commande reçue redéfinit la précédente pour chacun des deux relais.

Tableau 10.6 : Commandes de programmation des relais du projet.

Commande	Réponse	Description
RY:n:HMX	OK	Bascule relais n sur ON si humidité >= max
RY:n:HMN	OK	Bascule relais n sur ON si humidité <= min
RY:n:LMX	OK	Bascule relais n sur ON si luminosité >= max
RY:n:LMN	OK	Bascule relais n sur ON si luminosité <= min
RY:n:MMX	OK	Bascule relais n sur ON si moiteur sol >= max
RY:n:MMN	OK	Bascule relais n sur ON si moiteur sol =< min
RY:n:TMX	OK	Bascule relais n sur ON si température >= max
RY:n:TMN	OK	Bascule relais n sur ON si température <= min

Commandes de définition des planchers et plafonds

Le minimum et le maximum des différents capteurs peuvent être réglés au moyen des commandes de la famille ST (Tableau 10.7). Pour connaître la valeur actuelle d'un plancher ou d'un plafond, vous utilisez la famille de commandes GT depuis votre hôte. Vous pouvez modifier les valeurs à tout moment.

Tableau 10.7 : Commandes de définition des planchers et plafonds.

Commande	Réponse	Description
ST:HMX:val	OK	Définit valeur max humidité
ST:HMN:val	OK	Définit valeur min humidité
ST:LMX:val	OK	Définit valeur max luminosité
ST:LMN:val	OK	Définit valeur min luminosité
ST:MMX:val	OK	Définit valeur max moiteur sol
ST:MMN:val	OK	Définit valeur min moiteur sol
ST:TMX:val	OK	Définit valeur max température
ST:TMN:val	OK	Définit valeur min température
GT:HMX	GT:HMX:val	Lit valeur max humidité
GT:HMN	GT:HMN:val	Lit valeur min humidité
GT:LMX	GT:LMX:val	Lit valeur max luminosité
GT:LMN	GT:LMN:val	Lit valeur min luminosité
GT:MMX	GT:MMX:val	Lit valeur max moiteur
GT:MMN	GT:MMN:val	Lit valeur min moiteur
GT:TMX	GT:TMX:val	Lit valeur max température
GT:TMN	GT:TMN:val	Lit valeur min température

Le mécanisme de mise en correspondance des relais avec les capteurs consiste à lier l'état d'un relais ouvert ou fermé à un capteur *via* deux valeurs, la maximale et la minimale, qui servent de conditions de changement d'état. Vous pouvez donc déclencher un relais lorsqu'une valeur obtenue depuis un capteur passe au-dessus ou en dessous d'une valeur définie par le système hôte. L'appariement des relais n'est pas exclusif : les deux relais peuvent être liés au même capteur d'entrée avec les mêmes conditions limites. Bien sûr, cela n'a que peu d'intérêt en pratique, mais il est bon de savoir que c'est possible.

La version proposée du projet ne prévoit que deux relais, mais rien n'empêche d'en mettre plus. Comme l'a montré le Tableau 10.1, les deux broches I2C A4 et A5 restent disponibles pour ajouter une carte d'extension d'entrées-sorties I2C et ainsi relier d'autres équipements à votre projet Arduino.

Prototypage

J'ai choisi de créer le prototype du projet avec une mallette magique qui porte le nom Duinokit (Figure 10.10). Cette mallette réunit tout un tas de capteurs, de diodes LED, d'interrupteurs, et d'autres composants, autour d'une carte Arduino Nano. Le tout est monté sur un grand circuit imprimé qui offre de nombreux connecteurs. Un emplacement est prévu pour y ajouter une carte au format standard, ou plusieurs en empilement. C'est une sorte de version modernisée de la fameuse mallette de découverte de l'électronique que certains ont reçue pour Noël dans leur enfance.

L'approche traditionnelle de création du prototype consiste à utiliser une plaque à trous sur laquelle vous réunissez tous les composants. Cela dit, la mallette Duinokit est vraiment pratique, et constitue une intéressante plate-forme pour passer à la création de logiciels, tout en attendant de recevoir le circuit imprimé depuis votre fournisseur favori. Si cela vous intéresse, vous pouvez vous procurer ce kit à l'adresse `http://duinokit.com`.

N. d. T. : Notez que la valise Duinokit n'est pas donnée. Comptez plus de 200 euros.

La valise Duinokit propose le capteur DHT11 pour la température et l'humidité, une version moins précise que le DHT22. Les deux modèles ne sont pas exactement interchangeables au niveau logiciel. En effet, le DHT22 n'utilise pas

le même mot de données que le DHT11, du fait qu'il offre une plus grande précision de mesure.

Figure 10.10 : La mallette Duinokit.

La photorésistance et le capteur de moiteur utilisent chacun la moitié d'un amplificateur opérationnel double LM358. J'ai positionné ce circuit intégré sur la plaque d'expérimentation de la valise Duinokit. Il y a encore de la place pour ajouter les résistances et les deux potentiomètres de réglage de 10 kiloohms.

Écriture du logiciel de prototype

Nous allons d'abord rédiger le programme pour le prototype, avant celui de la version définitive. Le premier logiciel sera exécuté sur le Duinokit pour lire les capteurs, vérifier que les amplis opérationnels LM358 fonctionnent bien et réaliser quelques tests. Nous verrons ensuite comment écrire le logiciel définitif qui va utiliser le protocole commande/réponse présenté plus haut, et comment programmer les relais. Nous nous servirons du matériel du bouclier définitif pour créer le logiciel définitif.

Dans le logiciel du prototype, nous allons chercher à lire les données des capteurs externes et du capteur DHT11. Cela va nous permettre d'estimer les valeurs des plages d'entrée. Les données en sortie seront affichées dans le moniteur série de l'atelier Arduino IDE. Le Listing 10.1 montre la totalité du code source du programme de test. Il correspond à l'unique fichier portant le nom *gs_proto.ino*.

Les fonctions qui servent aux conversions de température et de point de rosée ne sont pas indispensables pour le test du prototype. Je les ai ajoutées au cas où vous auriez envie de vous en servir.

Listing 10.1 : Code source du prototype du projet Greenshield.

```
// GreenShield prototype
// 2015 J. M. Hughes
// Avec librairie DHT11 de George Hadjikyriacou, SimKard
et Rob Tillaart

#include <dht11.h>

// Definitions
#define LHLED 2 // D2 Plancher humidite air
#define HHLED 3 // D3 Plafond humidite air
#define LMLED 4 // D4 Plancher moiteur
#define HMLED 5 // D5 Plafond moiteur
#define RY1OUT 6 // D6 Relais RY1
#define RY2OUT 7 // D7 Relais RY2
#define DHT11PIN 8 // D8 Bicapteur temp/humi
#define SMSINPUT A0 // Sonde de moiteur
#define LDRINPUT A1 // Photoresistance

// Variables globales
int curr_temp = 0; // Temperature actuelle (derniere
mesure)
int curr_hum  = 0; // Humidite actuelle
int curr_ambl = 0; // Luminosite actuelle
int curr_sms  = 0; // Moiteur du sol actuelle
int cntr = 0;

// Objet global dht11 instance de classe
dht11 DHT11;
```

```
// Lecture du DHT11 et stockage dans curr_hum et
curr_temp
void readDHT()
{
  if (!DHT11.readDHT(DHT11PIN)) { //£1
    curr_hum = DHT11.humR;
    curr_temp = DHT11.tmpC;

  }
}

// Lecture des entrees analogiques via les LM358 et
stockage
// dans curr_ambl et curr_sms
void readAnalog()
{
  curr_ambl = analogRead(LDRINPUT);
  curr_sms = analogRead(SMSINPUT);
}
// Conversion Celsius vers Fahrenheit
// Dans http://playground.arduino.cc/main/DHT11Lib
double Fahrenheit(double celsius)
{
  return 1.8 * celsius + 32;
}

// Conversion Celsius vers Kelvin
double Kelvin(double celsius)
{
  return celsius + 273.15;
}

// Fonction dewPoint() du NOAA
// Voir http://wahiduddin.net/calc/density_algorithms.htm
// Exemple sur http://playground.arduino.cc/main/DHT11Lib
double dewPoint(double celsius, double humidity)

{
  double A0= 373.15/(273.15 + celsius);
  double SUM = -7.90298 * (A0-1);
  SUM += 5.02808 * log10(A0);
  SUM += -1.3816e-7 * (pow(10, (11.344*(1-1/A0)))-1) ;
```

```
  SUM += 8.1328e-3 * (pow(10,(-3.49149*(A0-1)))-1) ;
  SUM += log10(1013.246);
  double VP = pow(10, SUM-3) * humidity;
  double T = log(VP/0.61078); // temp var
  return (241.88 * T) / (17.558-T);
}

// delta max = 0.6544 wrt dewPoint()
// 5x plus rapide que dewPoint()
// Voir http://en.wikipedia.org/wiki/Dew_point
// Exemple en http://playground.arduino.cc/main/DHT11Lib
double dewPointFast(double celsius, double humidity)
{
  double a = 17.271;
  double b = 237.7;
  double temp = (a * celsius) / (b + celsius) +
log(humidity/100);
  double Td = (b * temp) / (a - temp);
  return Td;

}
void setup()

{
  // Init des I/O serie
  Serial.begin(9600);
  // Configuration broches AVR
  pinMode(LHLED,OUTPUT);
  pinMode(HHLED,OUTPUT);
  pinMode(LMLED,OUTPUT);
  pinMode(HMLED,OUTPUT);
  pinMode(RY1OUT,OUTPUT);
  pinMode(RY2OUT,OUTPUT);
  // Preparation des variables de donnees
  curr_temp = 0;
  curr_hum = 0;
  curr_ambl = 0;
  curr_sms = 0;
}

void loop()
{
```

```
  // Lecture DHT11
  readDHT();
  // Lecture lumiere et moiteur
  readAnalog();
  // Affichage donnees
  Serial.println("\n");
  Serial.print("Luminosite brute: ");
  Serial.println(curr_ambl);
  Serial.print("Moiteur brute: ");
  Serial.println(curr_sms);
  Serial.print("Humidite (%) : ");
  Serial.println((float)DHT11.humR, 2);
  Serial.print("Temp. (Celsius) : ");
  Serial.println((float)DHT11.tmpC, 2);
  Serial.print("Temp. (Fahrenheit) : ");
  Serial.println(Fahrenheit(DHT11.tmpC), 2);
  Serial.print("Temperature (K) : ");
  Serial.println(Kelvin(DHT11.tmpC), 2);
  Serial.print("Point de rosee (C): ");
  Serial.println(dewPoint(DHT11.tmpC, DHT11.humR));
  Serial.print("P. de rosee rapide: ");
  Serial.println(dewPointFast(DHT11.tmpC, DHT11.humR));
  // Alignement affichage
  for (int i = 0; i < 12; i++)
    Serial.println();
  delay(1000);
}
```

La fonction standard `setup()` se contente de préparer les variables et d'initialiser puis d'ouvrir les ports d'entrées-sorties série. Elle définit les modes des broches et remet à zéro les variables globales qui vont recevoir les valeurs mesurées. Chaque appel à une des fonctions de lecture `readDHT()` et `readAnalog()` permet de récupérer la plus récente valeur trouvée dans le capteur DHT et dans les entrées analogiques.

La boucle principale effectue la lecture des entrées analogiques et du DHT11. Elle reformate les données puis envoie le résultat vers le moniteur série USB. Dans ce prototype, nous ne cherchons pas à nous mettre à l'écoute d'une machine hôte. Nous ne réalisons également aucun paramétrage de déclenchement. Nous nous contentons de lire puis d'afficher les données des capteurs.

Notre prototype utilise une librairie de fonctions open source pour le capteur DHT11 alors que la version finale utilisera la librairie appropriée au modèle DHT22. Cette librairie DHT11 créée par George Hadjikyriacou, SimKard et Rob Tillaart est disponible dans l'espace Arduino Playground (http://playground.arduino.cc/), tout comme les quatre fonctions `Fahrenheit()`, `Kelvin()`, `dewPoint()` et `dewPointFast()`.

Test du prototype

Nous pouvons maintenant tester les fonctions de lecture de données et procéder au peaufinage du fonctionnement. Le moment est idéal pour détecter tout problème de réalisation du circuit. Il serait beaucoup plus difficile de réparer des erreurs une fois le circuit imprimé peuplé. De plus, vous risqueriez d'endommager des composants sur la version finale.

Pour le prototype, nous voulons donc vérifier le bon fonctionnement des capteurs en récupérant les valeurs mesurées. Pour réaliser ces tests, il nous faut un peu de sable sec, un peu d'eau, un thermomètre numérique fiable, un réfrigérateur, un four et si possible choisir un jour ensoleillé.

Nous commençons par vérifier le capteur d'humidité et de température de l'air. Vous commencez par mesurer la température actuelle au moyen d'un thermomètre numérique. Vous placez ensuite le montage avec le thermomètre dans votre réfrigérateur. L'alimentation par le câble USB pourra sortir du réfrigérateur porte fermée grâce aux joints dans la porte. La troisième étape consiste à préchauffer votre four à peine, de l'ordre de 60 ^{o} puis à y placer le montage et le thermomètre. Vous collectez ainsi trois couples de mesures qui permettent de dessiner la courbe de calibration afin de compenser la non-linéarité du capteur de température.

Les tests d'humidité sont un peu moins simples, mais vous devriez parvenir à obtenir les valeurs extrêmes pour votre courbe. Pour l'environnement à très faible humidité de l'air, placez le montage quelques minutes dans la partie congélation de votre réfrigérateur (le *freezer*). En effet, l'air est quasiment sec à cet endroit parce que la vapeur d'eau s'y condense (c'est ce qui oblige à réaliser la fameuse opération périodique de dégivrage). La technique de lyophilisation à froid descend beaucoup plus bas et sous vide partiel (environ -80 ^{o}C). Pour notre test, nous devrions obtenir une mesure qui indique environ 5 % d'humidité de l'air, voire moins.

Une autre technique consiste à mesurer l'humidité en présence d'eau saturée en sel (saumure). Il suffit de mouiller du sel de table dans un bol afin qu'il n'y ait plus d'eau non imbibée. Vous placez ensuite la sonde au-dessus du sel puis vous refermez le tout avec un film plastique alimentaire. Cela dit, ce n'est pas très pratique s'il faut placer le montage à l'intérieur du saladier sans le mouiller.

Puisque nous disposons maintenant d'un minimum pour l'humidité, il nous reste à trouver le maximum. Il suffit de faire bouillir un peu d'eau et de placer un petit ventilateur pour envoyer la vapeur vers le capteur. Le résultat devrait se trouver dans la plage entre 80 et 90 % d'humidité. Vous savez ainsi que le capteur fonctionne, mais nous ne pouvons pas encore l'utiliser pour effectuer des mesures parce que nous n'avons pas de moyen de comparaison. Si vous disposez d'un capteur d'humidité indépendant, servez-vous-en pour préparer la courbe de calibration, comme celle réalisée pour la température.

Nous passons ensuite à la mesure de la moiteur, c'est-à-dire de l'humidité du sol. Il nous faut un peu de sable sec, une balance et un peu d'eau. Procurez-vous un saladier, en prenant soin de ne pas choisir un modèle en métal, car cela fausserait les mesures. Mesurez d'abord la tare du saladier vide puis ajoutez environ 250 g de sable sec. Laissez le saladier sur la balance.

Insérez maintenant la sonde de moiteur dans le sable sec et notez la mesure sur le moniteur série de l'atelier IDE. Enlevez le capteur et ajoutez assez d'eau dans le sable pour alourdir l'ensemble d'un quart du poids total. Laissez le sable s'imbiber et touchez-le : il doit être humide sans être détrempé.

Repositionnez le capteur et mesurez le résultat qui devrait être proche de 50 %. À partir de nos deux mesures, nous pouvons trouver un intermédiaire que nous appellerons le point des 25 %.

Il nous reste la photorésistance. Nous n'avons pas besoin d'une mesure précise, mais il nous sera utile de définir un minimum de luminosité. Cela peut correspondre à la lumière en intérieur un jour couvert ou la nuit. Pour l'autre extrémité, le plafond, placez le capteur en plein soleil. Cela constituera évidemment le maximum.

Nous prenons soin de noter les mesures des différents capteurs, car ce seront nos valeurs initiales pour configurer le projet. Nous n'aurons ainsi plus besoin de tâtonner pour trouver les planchers et plafonds des valeurs qui vont servir au déclenchement des relais.

Version finale du logiciel

Le logiciel de test du prototype se limitait à mesurer les données des capteurs. Dans la version finale, nous devons gérer le dialogue avec la machine hôte et définir puis gérer les points de déclenchement. Il nous faut donc apprendre à analyser une ligne de commande, à gérer le stockage des données et à faire des recherches dans un tableau de données.

Dans la version finale, nous n'affichons rien en sortie, et nous nous contentons d'activer selon les besoins les différentes diodes LED. Nous n'avons pas non plus de boutons de commande en entrée. L'interface série USB permet un dialogue entre la carte Greenshield et la machine hôte. La grande majorité du code source se consacre à l'analyse des commandes reçues depuis l'hôte et à la manière de paramétrer et de piloter les relais en conséquence.

Structure du code source

Dans la version finale du code source de Greenshield, nous trouvons plusieurs fichiers source qui sont autant de modules. Il suffit de demander le chargement dans l'éditeur IDE Arduino du fichier principal qui porte le nom *GreenShield.ino* pour faire apparaître des onglets avec chacun des fichiers que ce fichier principal référence (s'ils sont trouvés dans le même répertoire). Vous pouvez apercevoir les noms des autres modules dans les différents onglets en haut de la Figure 10.11.

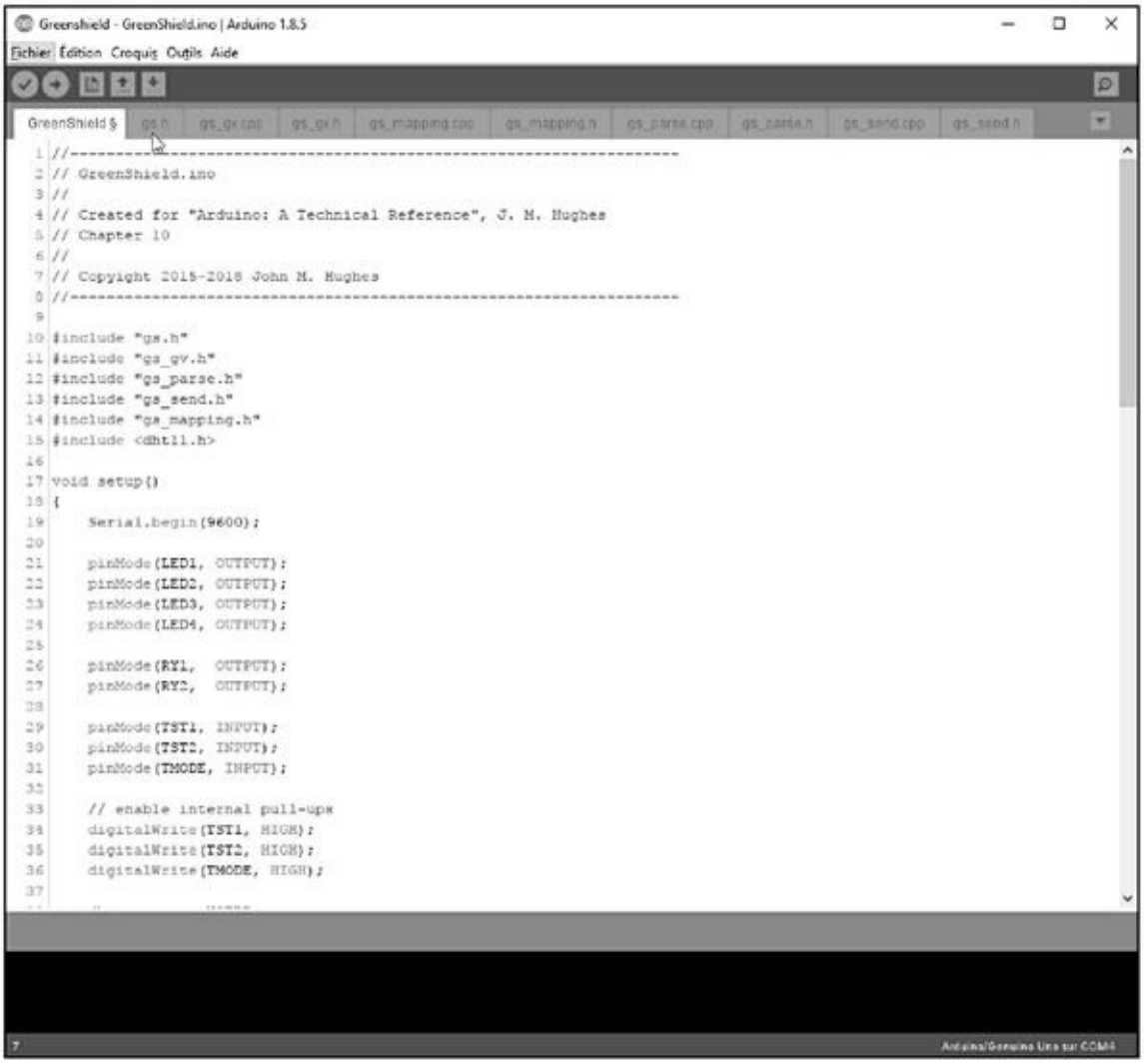


Figure 10.11 : L'atelier IDE Arduino avec les fichiers source de Greenshield.

Le Tableau 10.8 liste tous les fichiers du projet. Deux d'entre eux sont utilisés par tous les autres modules : *gs.h* et *gs_gv.h*. Le fichier *gs_gv.cpp* définit plusieurs variables globales, ce qui évite de les placer toutes au début du croquis principal. Ce fichier est compilé à part, puis partagé entre tous les autres modules.

Tableau 10.8 : Modules de code source de Greenshield.

Module	Fonction
GreenShield.ino	Module principal avec setup() et loop()

gs_gv.cpp	Variables globales
gs_gv.h	Directives #include
gs.h	Directives #define
gs_mapping.cpp	Association des fonctions de commande
gs_mapping.h	Directives #include
gs_parse.cpp	Analyse des commandes
gs_parse.h	Directives #include
gs_send.cpp	Envoi de données vers l'hôte
gs_send.h	Directives #include

Cette approche modulaire permet de mieux se concentrer sur une partie du code source plutôt qu'une autre. Une fois que vous avez terminé d'écrire la boucle principale, vous pouvez passer aux autres modules sans plus vous occuper du premier module. La modularité aide à la conception, elle facilite la compréhension du code source et sa maintenance.

Description fonctionnelle

Le diagramme de la boucle principale du logiciel est proposé dans la Figure 10.12. La fonction standard `loop()` commence avec le bloc **Start** et tourne en boucle jusqu'à la mise hors tension du montage. Vous constatez que le contenu est réparti en trois sections fonctionnelles, selon les pointillés : 1) réception des commandes et gestion des réponses, 2) acquisition de données et 3) définition et tests de déclenchement des relais grâce aux points de consigne (*setpoints*). Notez que le troisième bloc en bas de figure existe en réalité en quatre exemplaires, un par couple de mesures. Je n'ai présenté qu'un des quatre blocs pour alléger la figure.

Le Listing 10.2 correspond au fichier source *GreenShield.ino* qui réunit les deux fonctions standard `setup()` et `loop()`. La totalité du code source de ce projet, comme pour les autres projets, est disponible sur le site de l'éditeur ainsi que sur GitHub.

Listing 10.2 : Fichier source principal Greenshield.ino.

```
//-------------------------------------------------------------
------------
// GreenShield.ino
//-------------------------------------------------------------
------------

#include “gs.h”
#include “gs_gv.h”
#include “gs_parse.h”
#include “gs_send.h”
#include “gs_mapping.h”
#include <dht11.h>void setup()
  {
      Serial.begin(9600);

      pinMode(LED1, OUTPUT);
      pinMode(LED2, OUTPUT);
      pinMode(LED3, OUTPUT);
      pinMode(LED4, OUTPUT);

      pinMode(RY1,  OUTPUT);
      pinMode(RY2,  OUTPUT);

      pinMode(TST1, INPUT);
      pinMode(TST2, INPUT);
      pinMode(TMODE, INPUT);

      // Active les resistances de tirage
      digitalWrite(TST1, HIGH);
      digitalWrite(TST2, HIGH);
      digitalWrite(TMODE, HIGH);

      dht_status = NOERR;

      // Teste les LED
      digitalWrite(LED1, HIGH);
      delay(150);
      digitalWrite(LED1, LOW);
      delay(150);
      digitalWrite(LED2, HIGH);
```

```
   delay(150);
   digitalWrite(LED2, LOW);
   delay(150);
   digitalWrite(LED3, HIGH);
   delay(150);
   digitalWrite(LED3, LOW);
   delay(150);
   digitalWrite(LED4, HIGH);
   delay(150);
   digitalWrite(LED4, LOW);

   // Consulte unite temperature
   if (digitalRead(TMODE) == 0) {
       tmode = TF;
   }

   // Lecture capteurs et affichage (en option)
   ReadDHT();
   curr_temp  = ReadTemp();
   curr_humid = ReadHumidity();
   curr_moist = ReadMoisture();
   curr_light = ReadLight();
 if (digitalRead(TST1) == 0) {
     Serial.print(“Temperature: “);
     Serial.println(curr_temp);
     Serial.print(“Humidite   : “);
     Serial.println(curr_humid);
     Serial.print("Moiteur    : ");
     Serial.println(curr_moist);
     Serial.print(“Lumiere    : “);
     Serial.println(curr_light);
}

 if (digitalRead(TST2) == 0) {
     digitalWrite(RY1, HIGH);
     delay(200);
     digitalWrite(RY1, LOW);
     delay(200);
     digitalWrite(RY2, HIGH);
     delay(200);
     digitalWrite(RY2, LOW);
}
```

```
    Serial.println("OK");
}

void loop()
{
    // Consulte unite temperature
    if (digitalRead(TMODE) == 0) {
        tmode = TF;
   }

    ReadDHT();
    ParseCmd();
    curr_temp  = ReadTemp();
    curr_humid = ReadHumidity();
    curr_moist = ReadMoisture();
    curr_light = ReadLight();

    ScanMap();
    RyUpdate();
}
```

Vous remarquez les six directives `#include` qui listent les noms des autres modules. Notez qu'il faut demander l'inclusion d'un fichier source ici, même s'il n'est pas directement utilisé par ce module principal.

Nous découvrons ensuite le fichier des définitions globales *gs.h* (Listing 10.3). Il définit toute une série de constantes utilisées par les autres modules. En optant pour des instructions `#define`, nous obtenons une taille de fichier compilé inférieure. Nous en avons parlé dans la section consacrée aux constantes vers la fin du Chapitre 5.

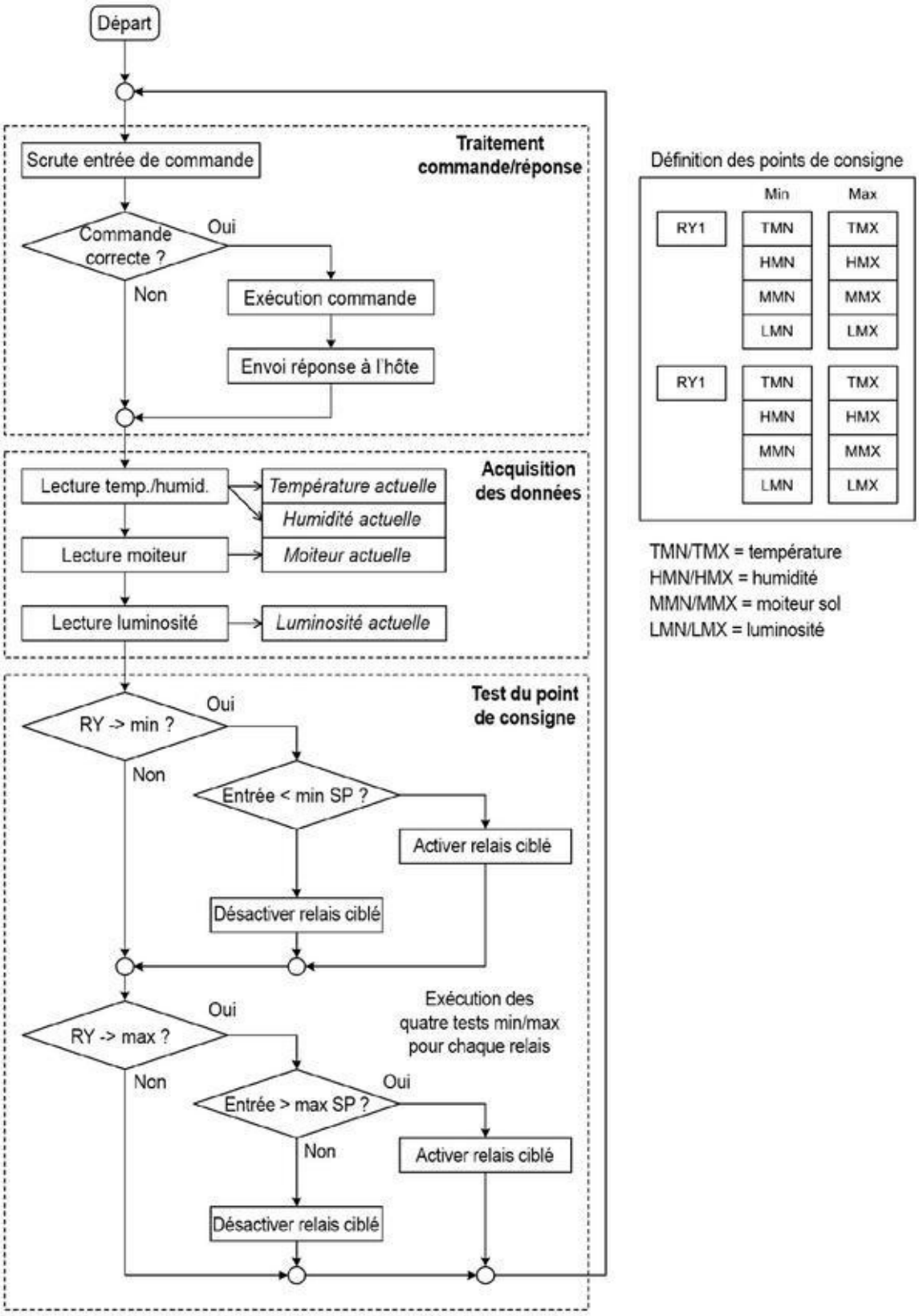


Figure 10.12 : Diagramme fonctionnel de Greenshield.

Listing 10.3 : Fichier des définitions globales (gs.h).

```
// gs.h
//

#ifndef GSDEFS_H
#define GSDEFS_H

#define MAXINSZ 12 // Taille tampon mémoire d'entrée

#define NOERR 0    // Codes erreur
#define TIMEOUT 1
#define BADCHAR 2
#define BADVAL  3

#define LED1 2     // Affectation broches LED
#define LED2 3
#define LED3 4
#define LED4 5
#define RY1  6     // Affectation broches relais
#define RY2  7

#define DHT22   8   // DHT22
#define MPROBE A0   // Sonde de moiteur
#define LDRIN  A1   // Photorésistance LDR

#define MAXRY   2   // Nombre de relais

#define MAP_NONE 0  // Mapping vectors
#define MAP_TEMPMIN  1
#define MAP_TEMPMAX  2
#define MAP_HUMIDMIN 3
#define MAP_HUMIDMAX 4
#define MAP_MOISTMIN 5
#define MAP_MOISTMAX 6
#define MAP_LIGHTMIN 7
#define MAP_LIGHTMAX 8
#endif
```

Dans le listing précédent, nous trouvons une constante portant le nom *MAXRY*. Rendue égale à 2 dans cette version, elle détermine le nombre de relais que nous voulons contrôler. Vous pouvez tout à fait ajouter d'autres relais et modifier cette valeur. Découvrons ensuite le fichier d'inclusion *gs_mapping.h* (Listing 10.4). Il réunit les déclarations des fonctions pour lire le capteur DHT22 et les entrées analogiques, pour définir l'état de chacun des deux relais, pour contrôler les diodes LED d'indication d'état et enfin pour réaliser un balayage de toutes les conditions de réponse possible pour contrôler les relais (dans la fonction `ScanMap()`).

Listing 10.4 : Fonctions d'association d'actions de relais (gs_mapping.h).

```
// gs_mapping.h
//
#ifndef GSMAP_H
#define GSMAP_H

void ReadDHT22();
int ReadTemp();
int ReadHumidity();
int ReadMoisture();
int ReadLight();

int RyGet(int ry);
void RySet(int ry, int state);
void RyAll(int state);

void LEDControl(int LEDidx);
void ScanMap();

#endif
```

Le corps de cette fonction `ScanMap()` se trouve dans le fichier *gs_mapping.cpp* (Listing 10.5). Cette fonction est appelée à chaque tour de la boucle principale pour lire les valeurs analogiques d'entrée et les comparer à des valeurs planchers et plafonds pour activer ou désactiver les relais en conséquence.

> Ce code source ne comporte aucun contrôle qui empêche d'affecter plusieurs relais à la même opération parmi les 8 possibles. Des ordres contradictoires pourraient en résulter. Faites vos tests un relais à la fois. Notez que vous pouvez ajouter d'autres relais plus tard.

Listing 10.5 : Fonction d'analyse des valeurs d'action.

```
// gs_mapping.cpp

void ScanMap()
{
  for (int i = 0; i < MAXRY; i++) {
    if (rymap[i] != MAP_NONE) {
      switch (rymap[i]) {
        case MAP_TEMPMIN:
          if (curr_temp < mintemp)
            { RySet(i, 1); } else { RySet(i, 0); }
          break;
        case MAP_TEMPMAX:
          if (curr_temp > maxtemp)
            { RySet(i, 1); } else { RySet(i, 0); }
          break;
        case MAP_HUMIDMIN:
          if (curr_humid < minhum)
            { RySet(i, 1); } else { RySet(i, 0); }
          break;
        case MAP_HUMIDMAX:
          if (curr_humid > maxhum)
            { RySet(i, 1); } else { RySet(i, 0); }
          break;
        case MAP_MOISTMIN:
          if (curr_moist < minmoist)
            { RySet(i, 1); } else { RySet(i, 0); }
          break;
        case MAP_MOISTMAX:
          if (curr_moist > maxmoist)
            { RySet(i, 1); } else { RySet(i, 0); }
          break;
        case MAP_LIGHTMIN:
          if (curr_light < minlite)
            { RySet(i, 1); } else { RySet(i, 0); }
          break;
        case MAP_LIGHTMAX:
          if (curr_light > maxlite)
```

```
            { RySet(i, 1); } else { RySet(i, 0); }
          break;
        default:
                // Rien
        break;
      }
    }
  }
}
```

Découvrons enfin le module *gs_parse.cpp* qui contient notamment la fonction `ParseCmd()`, ainsi que les deux fonctions utilitaires `CntInt()` et `SendErr()`. Le Listing 10.6 montre le fichier d'en-tête correspondant, *gs_parse.h*.

Listing 10.6 : Le fichier d'en-tête du module d'analyse.

```
// gs_parse.h
//

#ifndef GSPARSE_H
#define GSPARSE_H

void ParseCmd();
int  CvtInt(char *strval, int strt, int strlen);
void SendErr(int errtype);

#endif
```

La fonction `ParseCmd()` est l'une des plus longues de ce projet. Elle contient un analyseur arborescent rapide pour déterminer le type de la commande reçue afin d'extraire le code de la fonction à appeler avec les bons paramètres. C'est cette fonction qui exécute sur-le-champ les commandes d'activation et de désactivation des relais. Elle sait également renvoyer l'état des relais ainsi que collecter et renvoyer les données analogiques vers la machine hôte ou un autre utilisateur. Les commandes sont exécutées tout en bas de la structure conditionnelle arborescente.

Fabrication du projet Greenshield

La description détaillée des techniques de création d'un schéma de principe et de dessin d'un circuit imprimé n'entre pas dans les objectifs de ce livre. Je vais néanmoins procéder à une description générale des étapes qui permettent de passer du schéma au dessin du circuit imprimé et au circuit monté. Pour les détails, je vous invite à consulter l'Annexe D. Vous pouvez également faire une recherche sur le Web de l'expression « CadSoft Eagle » qui devrait vous permettre de trouver des tutoriels, dont certains en français. Visitez notamment le site de l'éditeur Autodesk.

N. d. T. : Autodesk, l'éditeur du très célèbre AutoCAD, a racheté Eagle en 2016.

La Figure 10.13 montre le schéma de principe de notre projet tel que réalisé avec le logiciel Eagle. Vous remarquez en haut à gauche le rectangle légendé Arduino_R3_SHIELD. C'est un objet prédéfini qui fait partie de la librairie de dessins de composants SparkFun pour Eagle. Ce genre d'objet permet d'accélérer la création des schémas. C'est beaucoup plus confortable que de devoir positionner les broches d'entrées-sorties du circuit intégré manuellement pendant la phase de création du circuit imprimé.

La prise en main d'un logiciel de conception de circuits électroniques n'est pas ce qu'il y a de plus évident, et c'est également le cas du logiciel Eagle. D'habitude, j'utilise un logiciel de dessin pour préparer mes schémas de principe afin de disposer de tout le confort, mais il est très pratique de pouvoir combiner les données du schéma de principe avec celles d'implantation des pistes sur le circuit imprimé. Vous pouvez juger de la différence du résultat entre les Figures 10.13 et 10.9. Les logiciels dédiés tels qu'Eagle et Fritzing (que nous allons utiliser pour le projet suivant) permettent d'exploiter les mêmes données pour le schéma et pour le circuit imprimé, ce qui est évidemment impossible si on utilise un logiciel de dessin pur pour les schémas.

Lors de la création de votre schéma, vérifiez toujours que vous sélectionnez les bons composants. Le symbole d'une résistance reste le même qu'il s'agisse d'un composant monté en surface 0805 SMC ou d'une classique résistance d'un quart de watt avec ses deux pattes. Certains composants sont difficiles à trouver dans la librairie du logiciel. Vous pouvez saisir des caractères génériques dans Eagle pour trouver toute une famille de composants. Par exemple, pour trouver le circuit ULN2003AD pour le projet Greenshield, j'ai saisi la séquence `*2003*`. J'ai ainsi repéré la catégorie **uln-udn**. Parfois, vous ne trouverez pas le composant ; il faudra dans ce cas chercher sur le Web l'objet correspondant.

Plusieurs sites proposent des librairies complètes de composants disponibles gratuitement, et notamment les sites SparkFun et Adafruit.

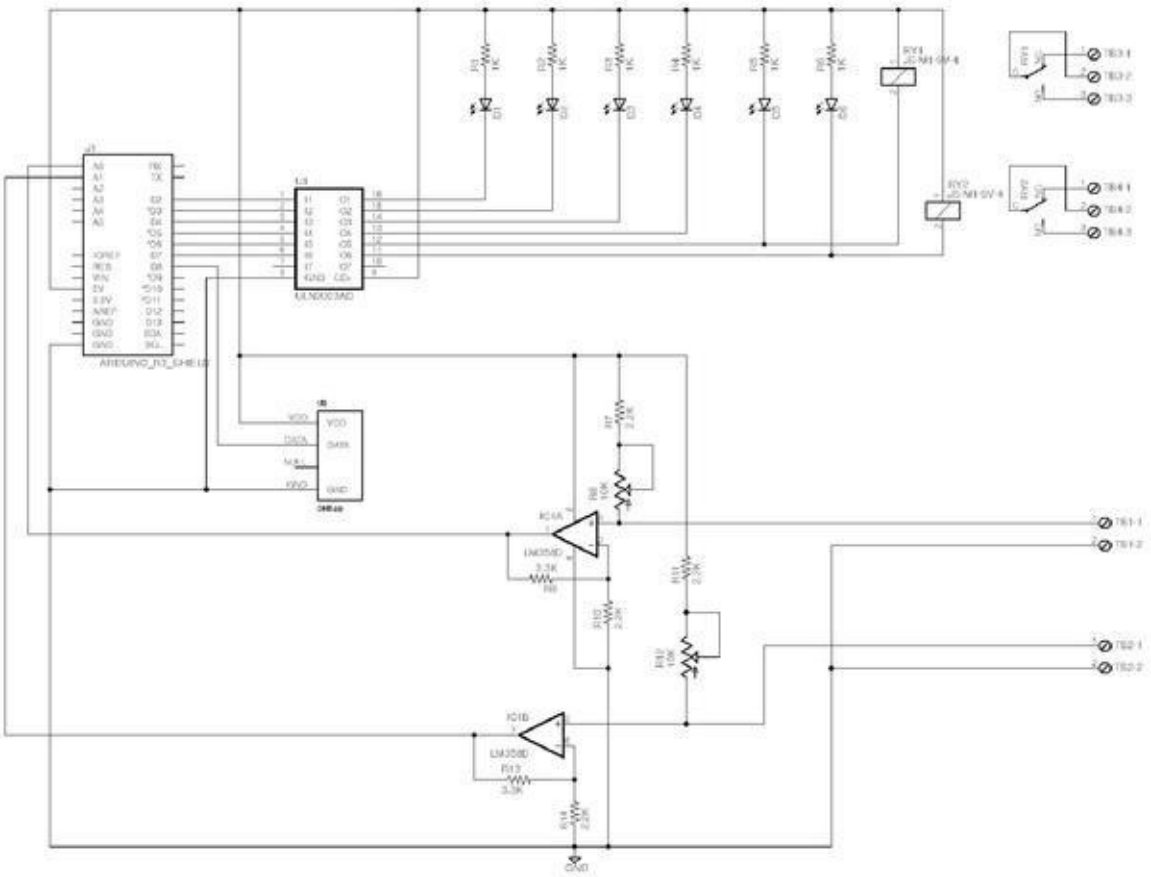


Figure 10.13 : Schéma de principe de Greenshield produit avec Eagle.

Une fois que le schéma est vérifié, vous pouvez passer à la création du circuit imprimé. Dans son aspect initial, le circuit va présenter toutes les connexions en liaison directe, comme si les connexions étaient faites avec des fils volants. Cette représentation visuelle correspond à la lecture de la liste des connexions ou liste de réseau (*network list*).

N. d. T. : Les liaisons directes entre connexions sont appelées en anglais *Rat Nest*.

Il s’agit alors de repositionner les composants à l’intérieur de la surface du circuit puis de les réorienter au mieux. Il faut commencer par positionner les connecteurs aux bons endroits, puis regrouper les composants par affinité fonctionnelle, tout en cherchant à limiter le nombre de croisements entre lignes

directes, y compris en faisant tourner certains composants. Une fois ce ménage fait, il vous sera beaucoup plus facile de mettre en place les pistes définitives.

Pour tracer les pistes, vous pouvez soit travailler manuellement, soit tirer avantage d'une fonction d'autoroutage. Eagle en propose une, mais je l'utilise rarement. L'autoroutage suppose de travailler par tâtonnements successifs, en déplaçant et réorientant des composants à chaque étape puis en relançant le processus. Souvent, vous finissez par constater que vous aurez plus vite fait à la main.

Lorsque l'on conçoit un circuit imprimé à deux couches, dessus et dessous, il faut de temps à autre poursuivre une piste en changeant de côté, ce qui correspond à la mise en place d'une traversée que l'on appelle un via. Dans le logiciel Eagle, les vias sont implantés automatiquement. Il suffit de sélectionner l'autre côté au moyen de la liste déroulante de sélection du calque dans le coin supérieur gauche de la barre d'outils de dessin. Le logiciel va automatiquement implanter un via et poursuivre le tracé de la piste de l'autre côté du circuit.

La Figure 10.14 montre un résultat possible. Le dessus correspond aux lignes en gris clair et les pistes du dessous aux lignes en gris foncé. Les silhouettes des composants sont elles aussi représentées en gris clair. Dans une version en couleur de la figure, le dessus correspond au rouge foncé et le dessous au bleu.

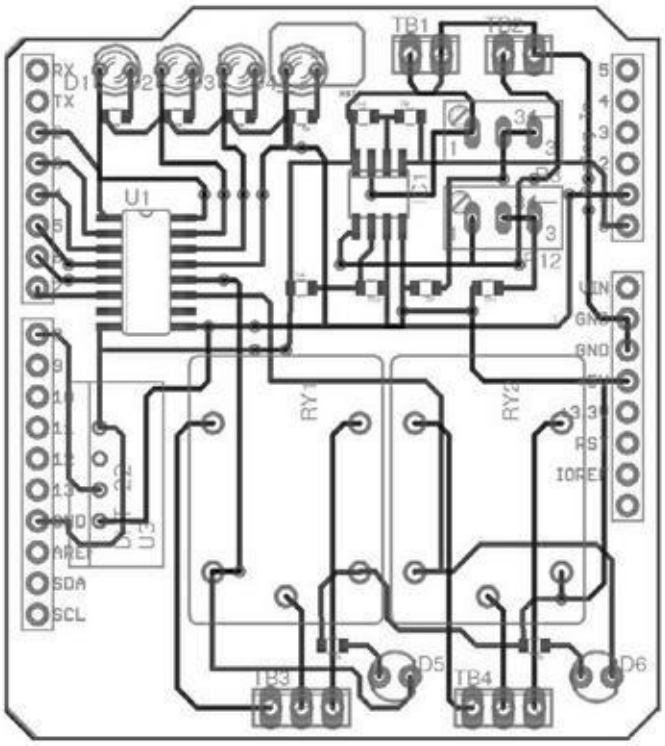


Figure 10.14 : Dessin du circuit imprimé du projet Greenshield.

Le logiciel Eagle sait générer des fichiers au format Gerber, un format reconnu par tous les fabricants de circuits imprimés. Pour un dernier contrôle avant d'envoyer mes fichiers au sous-traitant, j'ai utilisé la visionneuse Gerbv qui fait partie du paquetage gEDA. La Figure 10.15 montre le typon du projet dans l'outil Gerbv.

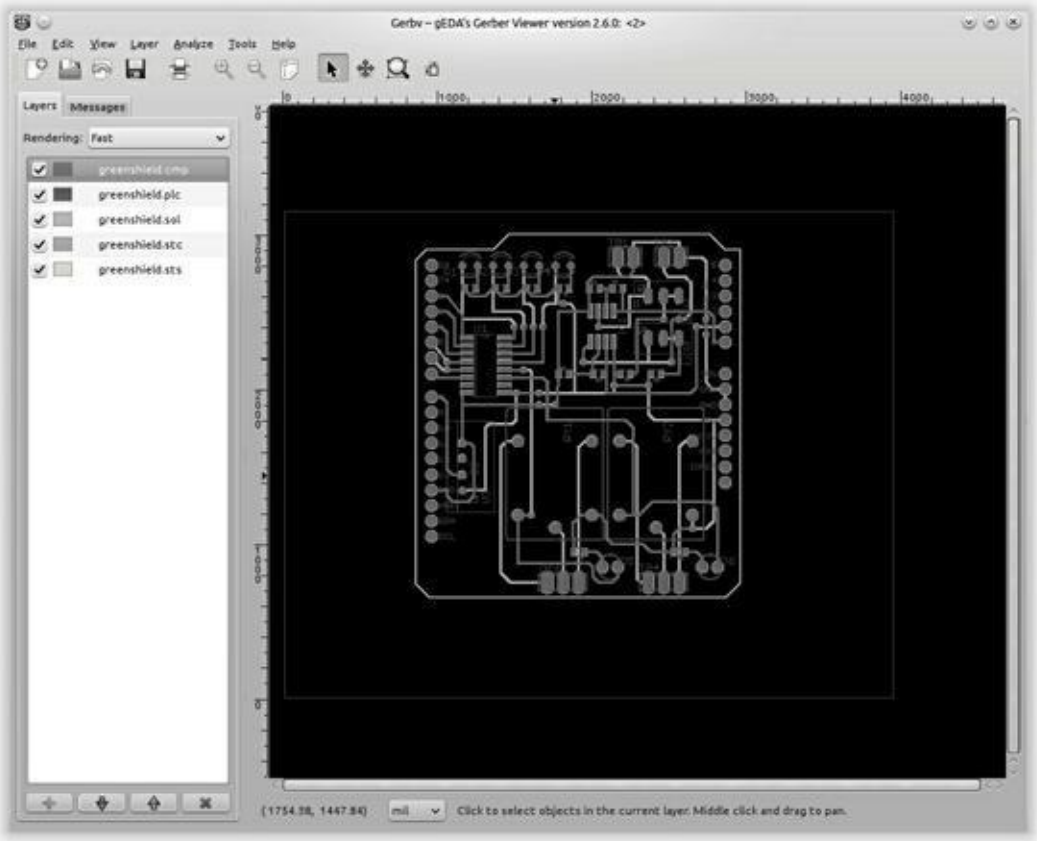


Figure 10.15 : Affichage du typon du circuit imprimé dans Gerbv.

Pour réduire le coût de production d'un seul exemplaire d'un circuit imprimé, j'ai accepté de recevoir la plaque sans les découpes, restant ainsi rectangulaire. Cela n'a aucun effet sur son fonctionnement. La figure précédente laisse entrevoir les découpes qui n'ont pas été réalisées dans le circuit de la Figure 10.16. J'ai donc fait des retouches au schéma avant de le transmettre à mon sous-traitant. J'aurais pu également me charger des découpes sur le circuit une fois celui-ci revenu du sous-traitant. J'ai décidé de laisser la plaque rectangulaire.

Comptez de une à deux semaines pour recevoir le circuit imprimé tel que celui de la Figure 10.16.

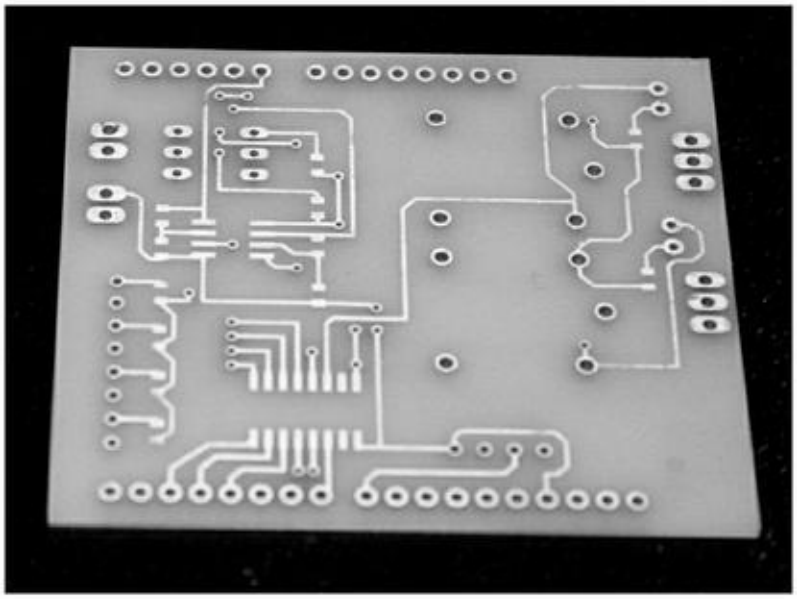


Figure 10.16 : Circuit imprimé Greenshield prêt à l'implantation des composants.

Lorsque vous avez terminé l'implantation, prenez toujours le temps de traquer les soudures froides et les ponts entre les pistes. Servez-vous d'une loupe pour ne pas manquer un seul défaut. Le circuit assemblé devrait ressembler à la Figure 10.17.

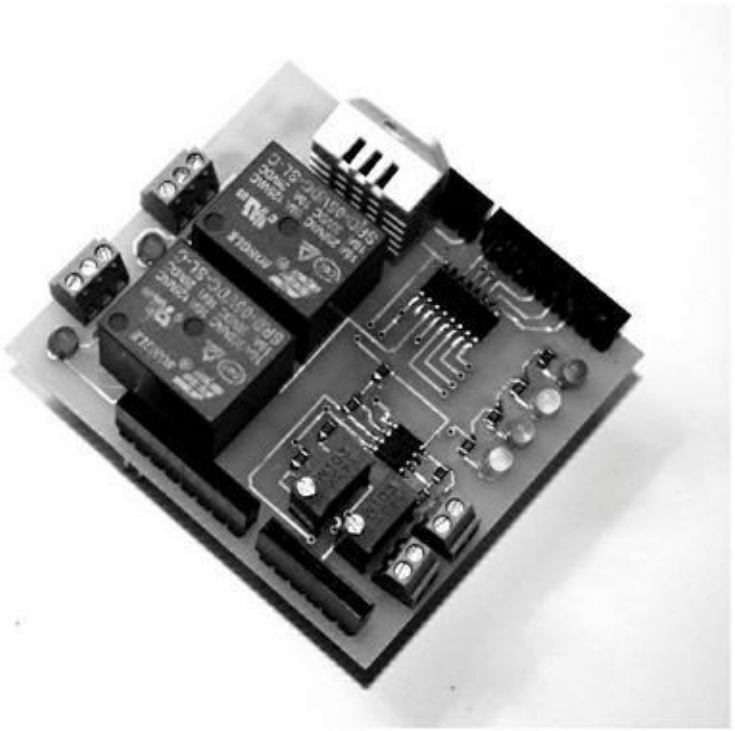


Figure 10.17 : Projet Greenshield prêt à l’emploi.

J’ai utilisé les broches mâles sur le dessous pour pouvoir insérer le bouclier sur une carte Arduino ou un autre bouclier. Comme déjà indiqué plus haut, il ne faut pas compter ajouter un autre bouclier par-dessus. J’ai prévu plusieurs points de test dont nous allons nous servir maintenant.

Test du projet définitif

Les composants montés en surface requièrent des compétences particulières et peuvent poser certains problèmes. Il faut donc réaliser quelques contrôles avant toute tentative de mettre le circuit sous tension. Voici ce qu’il faut vérifier.

Inspection visuelle

Observez bien tous les composants à la recherche de ponts de soudure entre deux broches ou deux pistes. Observez bien les résistances SMC. Parfois, une des deux extrémités est soulevée et n’est pas en contact avec le point de soudure. Cela arrive assez souvent lorsqu’on utilise un fer à souder traditionnel. Personnellement, j’utilise de la pâte à souder et un outil de soudure par vague.

Vérifiez de même qu’il n’y a pas de pont entre les plots des circuits intégrés U1 et IC1.

Contrôle de la polarité

Neuf composants peuvent être montés à l’envers : les deux circuits intégrés, les six LED et le capteur DHT22. Le circuit intégré d’ampli opérationnel comporte normalement un point pour indiquer la broche 1, mais parfois c’est un biseau. Pour les diodes LED, vous savez que les deux broches ne sont pas de la même longueur, et que la cathode (à relier au -) comporte un méplat.

Recherche des courts-circuits

Avec la fonction sonnette de votre multimètre, cherchez un court-circuit entre chaque paire de broches de l’ampli opérationnel LM358, c’est-à-dire IC1. Vérifiez ensuite que la broche 8 VCC de ce circuit est bien en continuité avec l’arrivée du +5 V. De même, vérifiez que la broche 4 est bien reliée à la masse.

Faites de même pour l’autre circuit ULM2003A (U1). Aucune des entrées ou sorties ne doit être en court-circuit avec une autre, la broche 8 doit aller à la masse et la broche 6 au +5 V.

Dernier contrôle

Avant de relier le projet à une carte Arduino, vérifiez avec le multimètre la valeur de la résistance entre les deux broches du +5 V et de la masse. Vous devriez obtenir une valeur au moins égale à 30 kiloohms. Si vous trouvez zéro ou presque, c’est qu’il y a court-circuit. Vous devez absolument le résoudre avant toute mise sous tension du circuit.

Une fois ces contrôles achevés, nous pouvons enficher le bouclier sur une carte Arduino et mettre le tout sous tension. Aucune des diodes LED du projet ne doit s’allumer au départ (sauf s’il restait un logiciel dans le microcontrôleur de l’Arduino qui en allumait une). Voici les quatre étapes de nos tests fonctionnels.

Test fonctionnel initial

Nous commençons par exécuter les mêmes tests que ceux du prototype afin de vérifier les entrées analogiques et le capteur de température.

Vous téléchargez donc la version du prototype vers l’Arduino puis vous ouvrez le moniteur série de l’atelier IDE. Si tout se passe bien, vous devriez voir s’afficher les valeurs de température, d’humidité et les entrées analogiques.

N. d. T. : Si les valeurs lues semblent incohérentes ou si la version prototype refuse de fonctionner, c'est lié au fait que la version finale utilise le capteur DHT22 au lieu du DHT11. En cas de souci, passez directement au logiciel définitif.

Test des entrées analogiques

Connectez une résistance de 470 ohms sur le bornier de la photorésistance. Observez la valeur qui s’affiche et ajustez R12 jusqu’à ce qu’elle redescende à zéro. Faites la même chose avec le potentiomètre R8 pour le capteur de moiteur du sol.

Test du capteur DHT22

Les valeurs affichées pour le DHT22 devraient correspondre à la température et au taux d’humidité de l’air de l’endroit où vous êtes. Vous pouvez vous servir d’un sèche-cheveux pas trop chaud pour observer la réaction du DHT22 à la chaleur.

Tests fonctionnels du logiciel

Vous pouvez maintenant télécharger la version définitive du logiciel en conservant ouverte la fenêtre du moniteur série. Nous n’aurons pas besoin de réaliser tous les tests, car certains ont déjà été faits avec le prototype.

Une fois que le logiciel a démarré, vous devriez voir la mention OK. Il n’y a pas de symbole particulier pour vous inviter à saisir une commande. Saisissez par exemple la commande **RY: 0: ?**, à laquelle le projet devrait répondre **RY:0:0**. Saisissez ensuite la commande **RY:0:1**. Vous devriez entendre le relais se fermer et voir la diode LED associée s’allumer. Avec une nouvelle commande **RY: 0 : ?**, vous devriez pouvoir lire **RY: 0 : 1**.

Il est conseillé de tester plusieurs autres commandes. Pour vérifier le plancher et le plafond des valeurs analogiques, court-circuitez puis isolez les deux bornes

d’entrée de chaque canal.

Tester, c’est bien

Lors de la fabrication de mon premier exemplaire du projet Greenshield, je n’arrivais plus à retrouver le sac avec les relais Songle 5 V que j’avais achetés. J’ai fini par trouver deux relais bleus dans une autre boîte et les ai mis en place. Même forme, même couleur et même marque. En fait, c’était des relais 12 V au lieu de 5. Évidemment, je ne comprenais pas pourquoi les relais ne se déclenchaient pas lorsque je testais le logiciel. C’est au bout d’un certain temps que j’ai constaté mon erreur en lisant mieux la légende sur le dessus des boîtiers des relais. Entretemps, j’ai retrouvé mes relais Single (ils étaient là où je les avais laissés). J’ai donc interverti les relais 12 V avec des relais 5 V et toutes les commandes fonctionnaient enfin comme prévu.

Morale de l’histoire : lisez bien les légendes des composants que vous installez avant de les souder.

Fonctionnement

Le projet Greenshield est assez simple, mais son domaine d’utilisation peut s’avérer assez sophistiqué, selon la manière dont vous le mettez en place dans son environnement. Il faut que les deux variables d’entrée principales que sont la luminosité et la moiteur du sol soient bien étalonnées. Il faut bien régler leur point milieu au moyen des potentiomètres. Toutes les photorésistances ne fonctionnent pas exactement de la même manière ; de même, une sonde d’humidité constituée de deux simples contacts ne réagit pas comme un capteur de moiteur doté d’un transistor.

Pour l’étalonnage, vous disposez de deux approches : 1) vous calibrez le circuit avec des références dont vous disposez ou 2) vous adaptez le projet à votre environnement en faisant une évaluation qualitative (par exemple en testant si le sol est suffisamment humide).

Avec la première approche, mieux vaut connaître dès le départ la valeur idéale pour la moiteur qui assure la meilleure croissance de vos légumes. Il ne reste dans ce cas qu’à faire coïncider les valeurs lues avec les valeurs recherchées. J’ai déjà présenté dans la section sur les tests du prototype comment effectuer

l’étalonnage du projet. Si vous y avez accès, vous utiliserez des matériels de laboratoire comme références de valeur.

L’approche qualitative est beaucoup plus simple. L’objectif est d’assurer une croissance correcte à vos plantes. Il suffira sans doute d’effectuer quelques réglages des potentiomètres pour trouver les valeurs minimales et maximales.

N’hésitez pas à faire des essais pour savoir si une autre approche conviendrait mieux à votre domaine d’application. Vous pouvez également collecter les données du projet et constituer des diagrammes permettant de peaufiner les réglages.

Pour aller plus loin

Le projet Greenshield est minimaliste, mais il a beaucoup de potentiel. Vous pouvez l’équiper d’un bouclier Bluetooth ou même d’une carte Ethernet afin de surveiller vos variables à distance, qu’il s’agisse de tomates dans une miniserre ou d’un jardin sur une colonie martienne. Vous pouvez relier des relais à un système d’arrosage en 24 V automatisé. L’autre relais permettrait par exemple de déclencher un ventilateur pour rafraîchir la serre ou un radiateur pour la réchauffer. Et bien sûr, vous pouvez ajouter d’autres relais grâce à des modules appropriés tels que ceux présentés dans le Chapitre 9.

Vous pouvez enfin ajouter un bouclier mémoire microSD, ce qui permet de fixer l’ensemble contre un tronc d’arbre en pleine forêt afin d’effectuer une collecte de données à longue durée. Dans ce cas, un panneau solaire ne sera pas de trop pour rendre le montage autonome. Vous pouvez aussi ajouter un bouclier WiFi ou GSM et mettre en place autant de circuits Greenshield que nécessaire pour surveiller les conditions de croissance d’une ferme entière.

La version actuelle du logiciel Greenshield ne sauvegarde pas les valeurs planchers et plafonds en cas d’interruption de l’alimentation. Pour ne pas perdre vos paramètres, vous pouvez utiliser l’espace mémoire EEPROM du microcontrôleur, comme indiqué dans le Chapitre 7 à la section consacrée à la librairie EEPROM.

Projets spécifiques compatibles Arduino

Vous pouvez envisager de fabriquer votre propre carte Arduino, compatible avec celles du commerce. Les créateurs de l'Arduino ont en effet publié le schéma de principe et le plan d'implantation du circuit imprimé. En revanche, les typons des couches ne sont pas disponibles, car ils appartiennent à Arduino.cc. Cela vous oblige à créer vos propres fichiers graphiques.

Cela dit, il n'est rentable de refabriquer une carte Arduino vous-même, ou même un bouclier d'extension existant, que si vous avez prévu de fabriquer des centaines ou des milliers de cartes et avez accès à un système de fabrication de circuits imprimés. D'ailleurs, la compatibilité en termes d'encombrement physique n'est pas toujours requise.

Vous ferez fi de la compatibilité physique si elle est un obstacle à la solution, par exemple lorsque vous devez intégrer votre projet Arduino dans un plus grand système ou lorsque les conditions d'encombrement empêchent l'utilisation du format standard, par exemple dans un drone ou un robot. Dans ce genre de situation, vous chercherez à maintenir la compatibilité au niveau logiciel. Votre circuit ne ressemblera pas à une carte Arduino et ne pourra pas être enfiché comme un bouclier sur une carte mère Arduino. (Mais il est possible de prévoir sur votre circuit deux rangées de connecteurs pour accepter une carte d'extension.)

Comme indiqué dans le premier chapitre, un appareil peut être compatible Arduino au niveau logiciel sans l'être au niveau matériel. Il faut bien sûr que le circuit possède un microcontrôleur AVR, le logiciel d'amorçage bootloader et les librairies d'exécution Arduino (le bootloader est même facultatif). Dans la prochaine section, nous allons voir comment concevoir puis fabriquer un circuit contrôleur de puissance pour courant continu permettant de piloter des relais, des diodes LED à haute intensité ou des moteurs à courant continu ou à courant alternatif.

Programmation d'un circuit spécifique

Si vous avez besoin d'utiliser le logiciel d'amorçage (bootloader) Arduino avec un microcontrôleur que vous avez acheté nu, il faut d'abord installer l'amorceur dans la mémoire flash du circuit. À partir de ce moment, votre circuit spécifique va se comporter exactement comme une carte Arduino normale. J'ai présenté les outils et techniques pour télécharger le code exécutable vers un microcontrôleur AVR dans le Chapitre 6. Vous pouvez même vous épargner cette étape en achetant un microcontrôleur sur lequel l'amorceur a déjà été installé. De nombreux revendeurs proposent cette option pour l'ATmega328 par exemple. Cherchez sur Internet l'expression « ATmega328 bootloader préchargé ».

Une fois l'amorceur en place, vous devez vous intéresser aux communications avec la machine hôte. Vous pouvez opter pour un adaptateur USB vers série ou mettre en place une interface série standard en y ajoutant un convertisseur RS-232 vers USB comme celui de la Figure 10.26, ou encore un adaptateur USB vers série comme celui montré en Figure 6.8. Dans le projet de la section suivante, nous n'utilisons pas l'interface série pour programmer le microcontrôleur, mais un programmateur ISP (ICSP) comme l'Atmel-ICE ou l'USBtinyISP, tous deux présentés dans le Chapitre 6.

Le projet Switchinator

J’ai choisi pour ce projet le nom Switchinator parce qu’il permet de déclencher 14 canaux en courant continu, en étant télécommandable et en offrant quatre entrées analogiques. Dans cette version, nous utilisons une interface au format RS-232 et un protocole commande/réponse très simple. Une version plus sophistiquée pourrait adopter le protocole RS-485.

Définition et planification

Ce projet est un montage autonome, qui se fonde sur un microcontrôleur ATmega328. Il n’a pas besoin du logiciel amorceur bootloader Arduino. Il peut également fonctionner à partir d’un microcontrôleur doté du bootloader, avec un programmateur série ou un convertisseur USB vers série. J’ai choisi de tirer profit de l’atelier Arduino IDE pour la compilation et du module USBtinyISP d’Adafruit pour la programmation (j’ai décrit en détail la programmation du microcontrôleur dans le Chapitre 6).

Voici d’abord les principales caractéristiques matérielles :

- Microcontrôleur ATmega328
- Horloge à cristal 16 MHZ
- Interface de programmation au format ICSP
- Alimentation intégrée 5 V continus (de 9 à 12 V en entrée)

Pour les entrées-sorties :

- Quatre entrées analogiques
- 14 sorties numériques

Pour l’interface de contrôle :

- Interface de dialogue avec la machine hôte au format RS-232

- Protocole commande/réponse piloté par la machine hôte
- Valeurs analogiques disponibles à la demande
- Prise de contrôle des sorties possibles par la machine hôte.

L'interface SPI est accessible par un connecteur ICSP. Toutes les sorties et entrées sont disponibles sur les bords de la carte par borniers à vis. L'interface RS-232 sert au dialogue entre la carte et la machine hôte, sur les deux broches D0 et D1 du microcontrôleur.

Le circuit imprimé n'utilise aucun composant monté en surface, ce qui simplifie le montage. Le prix à payer est une plus grande surface de circuit et quelques soucis de routage des pistes. Le format restera cependant inférieur à 100 mm sur 140. Je prévois des trous de fixation dans les quatre coins du circuit.

Les sorties numériques du projet serviront à piloter des relais, comme ceux du précédent projet, ou à contrôler jusqu'à trois moteurs pas à pas, selon un circuit élémentaire du type de celui de la Figure 10.18.

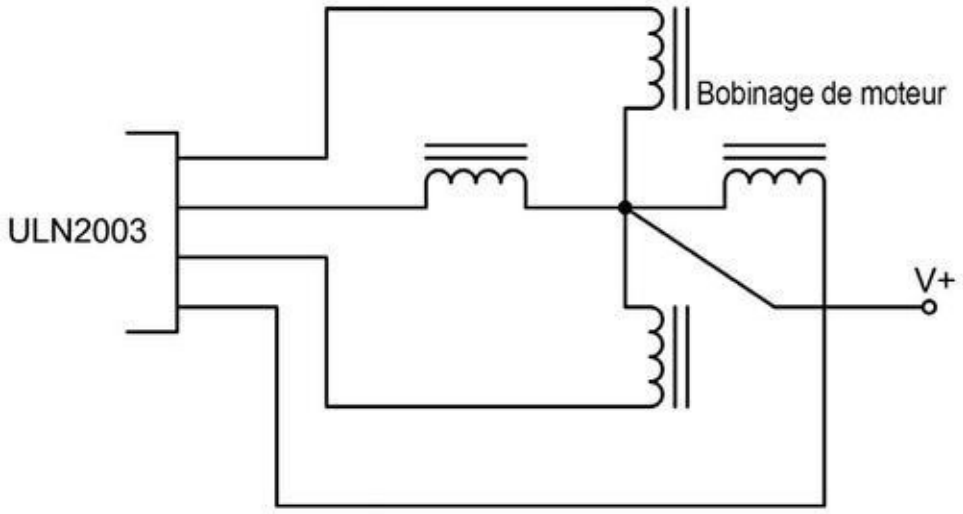


Figure 10.18 : Utilisation d'un ULN2003A pour contrôler un moteur pas à pas.

Les sorties numériques peuvent tout aussi bien contrôler des diodes LED à haute intensité, des solénoïdes ou des moteurs à courant continu comme le montre la Figure 10.19.

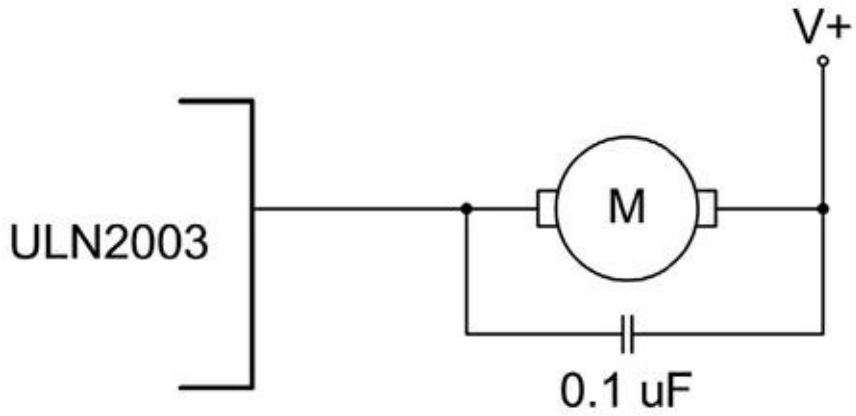


Figure 10.19 : Pilotage d’un moteur CC par un ULN2003A.

Le prototype sera monté sur une plaque d’expérimentation sans soudure, ce qui permettra d’écrire la version initiale du logiciel. La version complète sera écrite pour le montage définitif.

Pour le schéma et le routage des pistes, j’utiliserai le logiciel Fritzing.

Conception

Je ne prévois pas de boîtier pour le projet ni d’alimentation séparée, ce qui va simplifier la conception. Tout est réuni sur la même plaque de circuit imprimé. Nous n’aurons qu’à ajouter une source de courant continu externe.

Analyse fonctionnelle

Le Switchinator est un centre de déclenchement de sorties numériques avec quelques entrées analogiques. Il est d’abord destiné à contrôler la mise en marche et l’arrêt de charges à courant continu, inductives ou non. Le microcontrôleur ATmega328 sert surtout à analyser les commandes qu’il reçoit et à renvoyer des données d’état à la machine hôte.

Le circuit sert donc de périphérique d’entrées-sorties, mais il est possible de lui donner des capacités de fonctionnement autonome, en le faisant réagir aux valeurs analogiques qu’il reçoit. En ajoutant par exemple un capteur de température linéaire tel que le LM35, vous transformez facilement le projet en contrôleur pour une étuve de test ou une chambre thermique pour époxy.

Le projet comporte trois sections fonctionnelles principales : le microcontrôleur, les entrées-sorties numériques et l'alimentation (Figure 10.20).

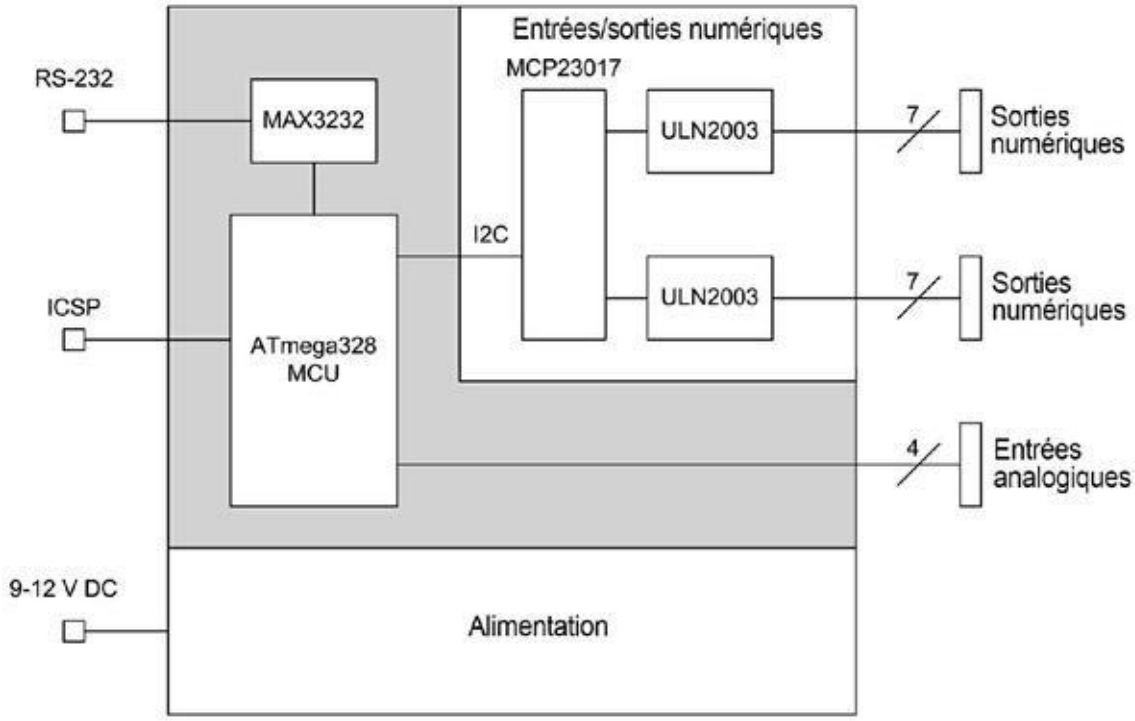


Figure 10.20 : Diagramme fonctionnel du Switchinator.

Description matérielle

Chacune des 14 sorties numériques est pilotée par un circuit pilote Darlington capable de fournir jusqu'à 300 mA, voire plus dans certains cas. Chaque circuit intégré ULN2003 offrant 7 pilotes, il nous en faut 2.

Nos circuits ULN2003 sont gérés par un circuit d'extension d'entrées-sorties I2C du modèle MCP 23017 de Microchip. Cela nous évite de mobiliser toutes les broches de sortie numériques du microcontrôleur. En effet, deux de ces broches vont nous servir à l'interface série et deux des broches analogiques vont servir à l'interface I2C vers le MCP23017. Les autres broches numériques sont disponibles pour une extension ultérieure.

L'interface RS-232 est incarnée par un circuit convertisseur MAX232 TTL vers RS-232. C'est par cette interface que nous allons dialoguer avec la machine hôte qui reste maître du dialogue. Il peut s'agir d'un PC, d'un autre Arduino ou de tout autre contrôleur programmable possédant une interface RS-232. L'interface

utilise le connecteur au format standard DB9. Notez que nous n'utilisons pas toutes les capacités du protocole, mais uniquement les deux fils d'émission et de réception RxD et TxD. Vous constatez que le projet ne dispose pas de connecteur USB.

Toutes les entrées et sorties sont disponibles sur des borniers à vis à écartement de 3,5 mm. Deux cavaliers permettent d'alimenter le circuit par une source externe et de lui fournir une tension analogique de référence sur le bornier. L'alimentation électrique normale se fait par un connecteur rond de type jack. Vous pouvez fournir entre 6 et 12 volts en courant continu, la tension optimale étant de 9 V. Cette tension est régulée et ramenée à 5 V sur le circuit grâce à un circuit régulateur 7805 en boîtier TO-220. Le schéma de principe complet tel que créé par le logiciel Fritzing est visible dans la Figure 10.21.

Si vous regardez le schéma de principe dans une approche esthétique, vous constatez que tous les composants ne sont pas représentés de façon proportionnelle les uns par rapport aux autres. C'est lié au fait que les objets graphiques correspondants ne proviennent pas tous de la même source. Ce sont des contributions open source, et tout le monde ne suit pas les mêmes règles. Pour autant, cela ne signifie pas que le schéma ne sera pas exploitable pour créer le routage. Ce n'est qu'un souci visuel.

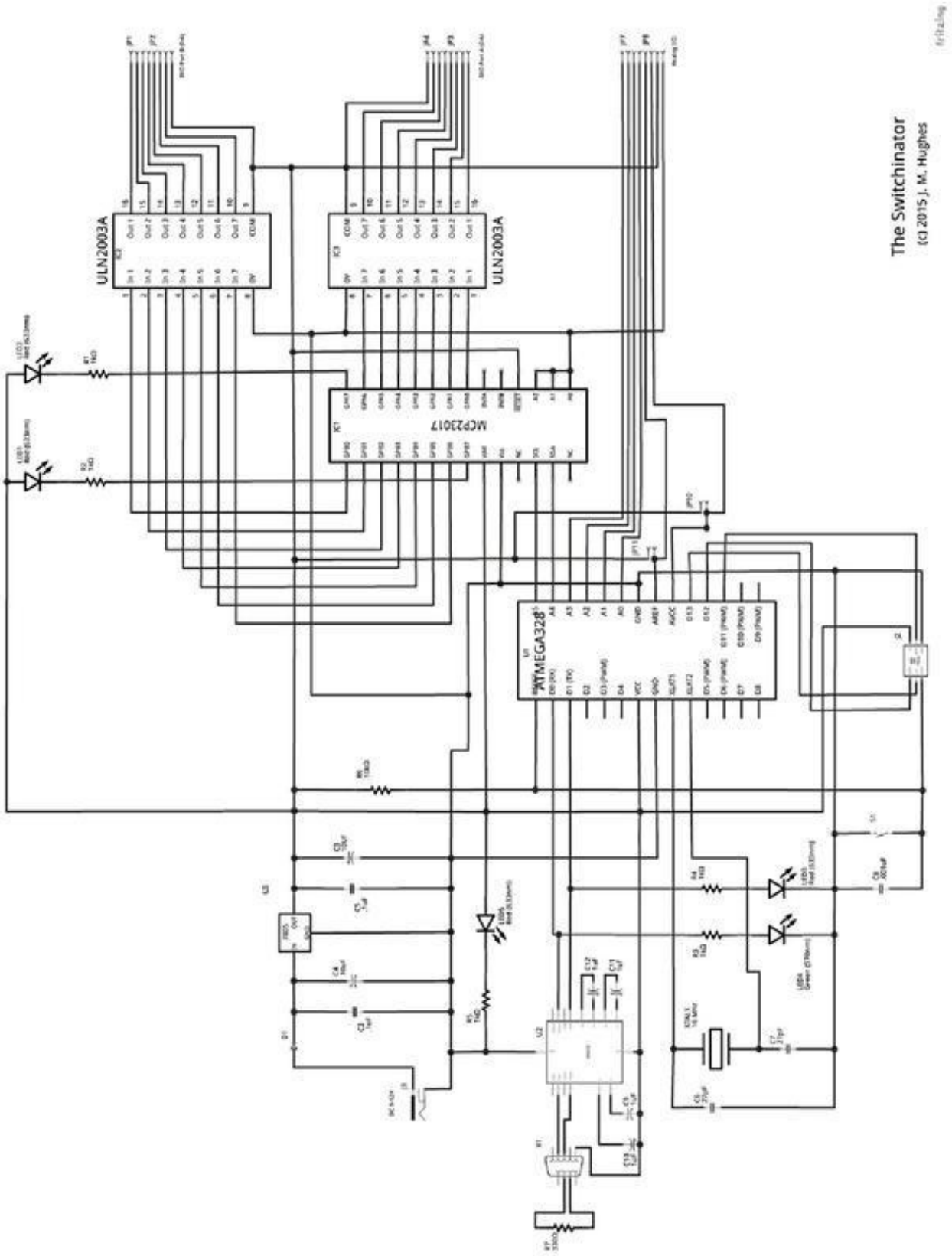


Figure 10.21 : Schéma de principe du projet Switchinator.

Le circuit d’extension numérique de MCP23017

Le circuit intégré MCP23017 est un composant de contrôle d’entrées-sorties fonctionnant grâce à des registres utilisant l’interface I2C (la variante MCP23S17 utilise l’interface SPI). Il sert à véhiculer des signaux binaires entre un appareil maître et jusqu’à 16 entrées ou sorties numériques. La description qui suit s’applique aux deux variantes.

C’est ce genre de composant qui est utilisé dans les boucliers d’extension d’entrées-sorties présentés dans le Chapitre 8. Il permet très simplement d’augmenter les capacités de contrôle d’un microcontrôleur. Grâce à son bus d’adresse, le circuit peut être mis en cascade et jusqu’à huit MCP23017 peuvent être reliés à un seul contrôleur, ce qui permet d’obtenir 128 canaux d’entrées-sorties numériques.

Le comportement du circuit est régi par les valeurs se trouvant dans une série de registres de contrôle et de données. Le maître peut lire et écrire des valeurs dans n’importe quel registre grâce à l’interface I2C ou SPI. Le diagramme fonctionnel du circuit est présenté dans la Figure 10.22.

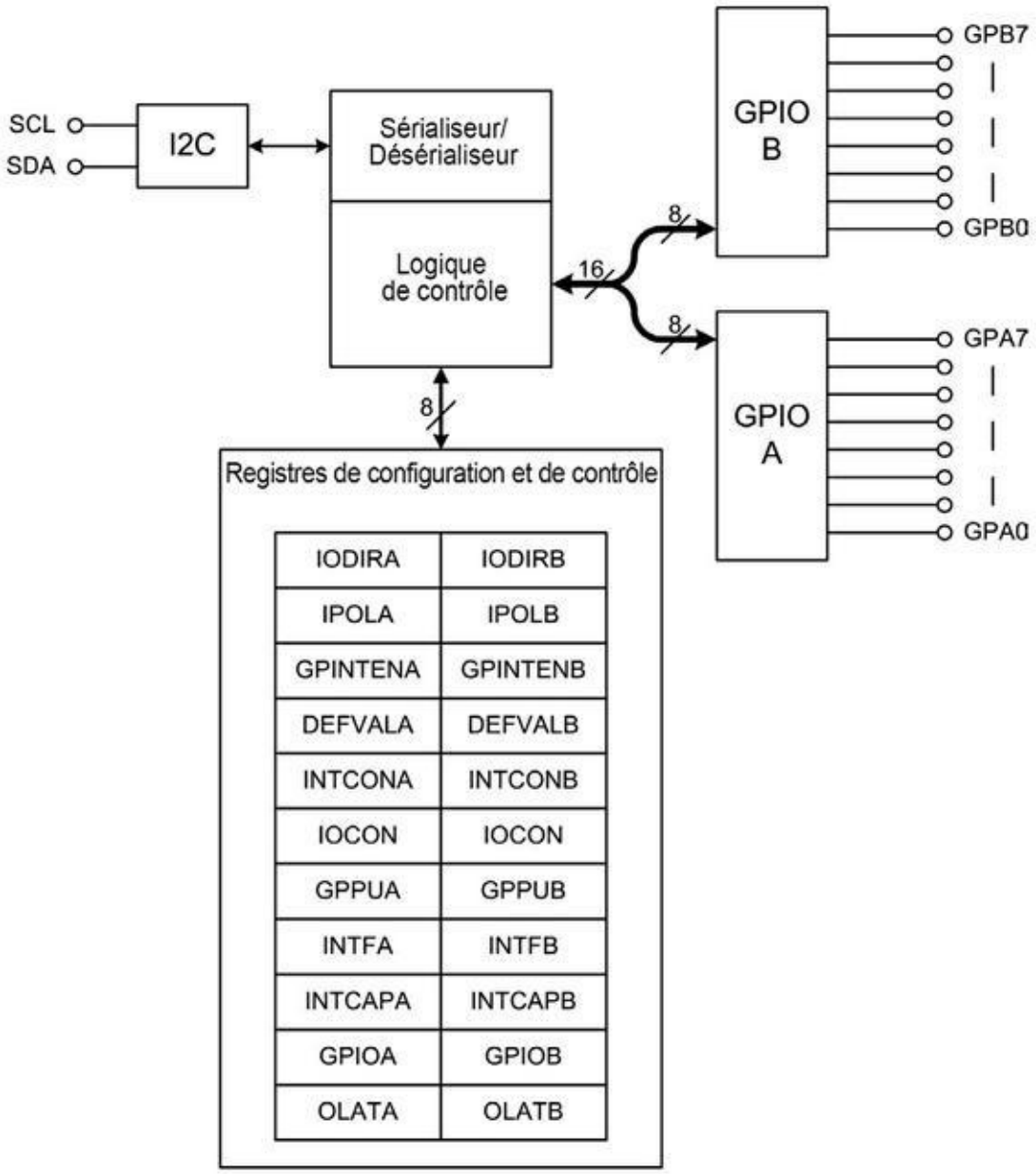


***Figure 10.22* : Diagramme fonctionnel du MCP23017.**

Le MCP23017 contient deux ports A et B de huit canaux chacun. Le contrôle est réalisé par 11 registres internes pour chacun des deux ports. La direction d'utilisation, entrée ou sortie, est configurée dans le registre IODIRA ou IODIRB. La polarité du port, actif à l'état haut ou à l'état bas, se règle par le registre IPOLA ou IPOLB. La génération des interruptions peut être autorisée par GPINTENA ou GPINTENB. Les résistances de rappel internes (*pull-up*) sont mises en œuvre par GPPUA et GPPUB. Vous lisez les valeurs sur le port A ou sur le port B dans les registres GPIOA et GPIOB. Enfin, les registres

OLATA et OLATB renvoient l'état actuel des verrous de sortie internes qui sont activés lorsque le port est en mode écriture et que des données sont en cours d'écriture dans les registres GPIOx. Tous les registres sont présentés dans le Tableau 10.9. La deuxième colonne, Addr, correspond à l'adresse de chaque registre en hexadécimal. La dernière colonne POR/RST signifie réinitialisation à la mise sous-tension et Reset externe.

Tableau 10.9 : Registres de contrôle du MCP23017 (IOCON.BANK = 0).

Registre	Addr	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2
IODIRA	00	IO7	IO6	IO5	IO4	IO3	IO2
IODIRB	01	IO7	IO6	IO5	IO4	IO3	IO2
IPOLA	02	IP7	IP6	IP5	IP4	IP3	IP2
IPOLB	03	IP7	IP6	IP5	IP4	IP3	IP2
GPINTENA	04	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2
GPINTENB	05	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2
DEFVALA	06	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2
DEFVALB	07	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2
INTCONA	08	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2
INTCONB	09	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2
IOCON	0A	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR
IOCON	0B	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR
GPPUA	0C	PU7	PU6	PU5	PU4	PU3	PU2
GPPUB	0D	PU7	PU6	PU5	PU4	PU3	PU2

INTFA	0E	INT7	INT6	INT5	INT4	INT3	INT2
INTFB	0F	INT7	INT6	INT5	INT4	INT3	INT2
INTCAPA	10	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2
INTCAPB	11	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2
GPIOA	12	GP7	GP6	GP5	GP4	GP3	GP2
GPIOB	13	GP7	GP6	GP5	GP4	GP3	GP2
OLATA	14	OL7	OL6	OL5	OL4	OL3	OL2
OLATB	15	OL7	OL6	OL5	OL4	OL3	OL2

Le circuit MCP23017 offre d'autres possibilités, mais nous n'en avons pas besoin pour ce simple exemple d'entrées-sorties numériques. Il nous suffit de savoir comment lire et écrire dans les registres GPIO. Pour tous les autres détails, je vous invite à étudier la fiche technique du circuit en cherchant sur le Web « MCP23017 datasheet ».

Voyons donc comment dialoguer avec le circuit. Le protocole comporte deux étapes pour accéder aux registres internes. La première étape consiste à envoyer 1 octet de contrôle ainsi que l'adresse du registre auquel vous voulez accéder. Dans la seconde étape, vous effectuez la lecture ou l'écriture de ce registre. Le dernier registre sélectionné reste actif tant qu'un autre n'est pas sélectionné par un nouveau mot de contrôle et d'adresse en deux octets.

L'octet de contrôle contient l'adresse du circuit sur 7 bits et 1 bit de lecture ou écriture R/W. S'il n'y a qu'un seul circuit, la valeur des bits A0, A1 et A2 est toujours nulle et le sixième bit toujours à un, ce qui donne la valeur hexadécimale 0x20 pour une écriture et 0x21 pour une lecture. Le bit de sens d'accès est le bit le moins significatif de l'octet de contrôle.

Vous disposez de plusieurs librairies de fonctions pour exploiter le MCP23017. De plus, écrire votre propre librairie n'est pas très complexe. L'extrait de code du Listing 10.7 montre par exemple comment sélectionner le registre de sens de travail puis le registre GPIOA en entrée en utilisant l'interface I2C et la librairie standard *Wire*.

Listing 10.7 : Exemple d'accès aux registres IODIRA et GPIOA du MCP23017.

```
Wire.beginTransmission(0x20);
Wire.write(0x00);

// Écrit dans le registre IODIRA
Wire.write(0xFF);
// Fin de session
Wire.endTransmission();

// Envoi des octets de contrôle et d'adresse
Wire.beginTransmission(0x20);
Wire.write(0x12);             // Port GPIOA
Wire.endTransmission();

// Lecture de données du port A
Wire.requestFrom(0x20, 1);  // Bit R/W à 1 (pour
lecture)
portval = Wire.read();      // Toujours dirigé vers
le port 0x12
```

Nous aurions pu omettre la première étape du fait que les ports d'entrées-sorties sont configurés en entrées au démarrage et après tout reset. Pour configurer le port en sortie, il faut forcer à zéro les bits d'IODIRA comme le montre l'extrait suivant.

Listing 10.8 : Exemple d'écriture de données vers un port de sortie.

```
Wire.beginTransmission(0x20);
Wire.write(0x00);

Wire.write(0x0);

Wire.endTransmission();

Wire.beginTransmission(0x20);
Wire.write(0x12); // GPIOA

// Écriture de données sur le port A
Wire.write(portval);
Wire.endTransmission();
```

L'interface SPI de la variante MCP23S17 fonctionne de la même manière et les broches d'adresse peuvent être réutilisées telles quelles. Ce qui les distingue est

que la variante MCP23017 fonctionne aussi vite que le permettent les entrées-sorties, que ce soit en I2C ou en SPI.

J'ai choisi d'animer le projet par un cristal à 16 MHz, car j'en ai tout un stock. Pour ce projet, l'oscillateur interne en montage RC du microcontrôleur aurait pu suffire, mais le cristal externe garantit une vitesse de fonctionnement fixe, au prix de quelques complications au niveau des bits fusibles internes (*fusebits*) du microcontrôleur. J'explique dans un encadré un peu plus loin comment configurer le microcontrôleur pour ce cristal oscillateur.

Les quatre sorties analogiques sont rendues disponibles sur deux borniers à 4 plots. Les plots libres servent au V+, à la masse, à une référence de tension analogique optionnelle et à des entrées AVCC. Les deux cavaliers JP10 et JP11 permettent en les enlevant de choisir une source de tension externe.

L'interface série RS-232 est gérée sur les deux broches numériques D0 et D1 du microcontrôleur. Trois autres broches numériques, D11, D12 et D13 se chargent de l'interface ICSP. Toutes les autres broches numériques restent libres. Quatre des entrées analogiques, de A0 à A3, servent aux entrées analogiques. Les deux autres broches A4 et A5 servent à l'interface I2C entre le contrôleur et le MCP23017. L'affectation des broches du microcontrôleur est fournie dans le Tableau 10.10.

Tableau 10.10 : Affectation des broches du microcontrôleur.

Broche	Fonction	Broche	Fonction	Broche	Fonction
D0	RxDRS-232	D7	Inutilisé	A0	Entrée analogique
D1	TxDRS-232	D8	Inutilisé	A1	Entrée analogique
D2	Inutilisé	D9	Inutilisé	A2	Entrée analogique
D3	Inutilisé	D10	Inutilisé	A3	Entrée analogique
D4	Inutilisé	D11	ICSPMOSI	A4	I2CSCL
D5	Inutilisé	D12	ICSPMISO	A5	I2CSDA
D6	Inutilisé	D13	ICSPSCLK		

Le schéma de principe de la Figure 10.21 permet d'établir la nomenclature complète des composants requis (Tableau 10.11).

Tableau 10.11 : Nomenclature de composants du Switchinator.

Quantité	Description	Quantité	Description
2	Condensateur 0,1 µF	4	LED rouge
2	Condensateur 10 µf	1	LED verte
2	Condensateur 27 pF	5	Résistance 1 kiloohm, 5 %, 1/4 W
1	Condensateur 1 nF	1	Résistance 10 kiloohms, 5 %, 1/4 W
4	Condensateur 1 µF	1	Résistance 330 ohms, 5 %, 1/4 W
1	Diode 1N4001	1	Bouton poussoir
1	MCP23017 I2C	1	ATmega328 MCU, 28 broches DIP
2	ULN2003A	1	MAX232 RS232, 16 broches DIP
1	Power jack, 5,5 mm	1	Régulateur 7805, TO-220
1	Connecteur AVRISP 2 x 3 broches	1	Connecteur DB9 pour circuit, mâle
6	Bornier, 3,5 mm	1	Cristal 16 MHz
2	Cavalier à 2 broches	1	

Conception logicielle

Comme pour le premier projet du chapitre, c’est le logiciel qui est la partie la plus complexe du projet. La figure 10.23 propose un diagramme fonctionnel de ce logiciel.

Lorsque le projet est prêt à recevoir une entrée de commande, il envoie un seul caractère supérieur (>) suivi d’une espace. Les réponses commencent par un seul caractère inférieur (<) suivi d’une espace puis des données. Ces deux symboles permettent facilement au logiciel fonctionnant sur la machine hôte de maintenir le dialogue. Toutes les commandes et réponses se terminent par le caractère de saut de ligne \n ou 0x0A.

Les entrées analogiques sont lues puis les valeurs sauvegardées à chaque tour de boucle. Elles sont envoyées vers l’hôte si celui-ci a demandé une lecture analogique. La mise à jour des valeurs se fait donc au rythme d’itération de la boucle principale.

Le logiciel cherche ensuite à savoir si des données série ont été reçues. Si le tampon mémoire d’entrée en contient, on appelle la fonction de lecture pour lire tous les caractères jusqu’à rencontrer celui de saut de ligne, valeur ASCII 0x0A. La fonction d’analyse extrait la commande de la chaîne, récupère les paramètres puis exécute la commande qu’il a comprise.

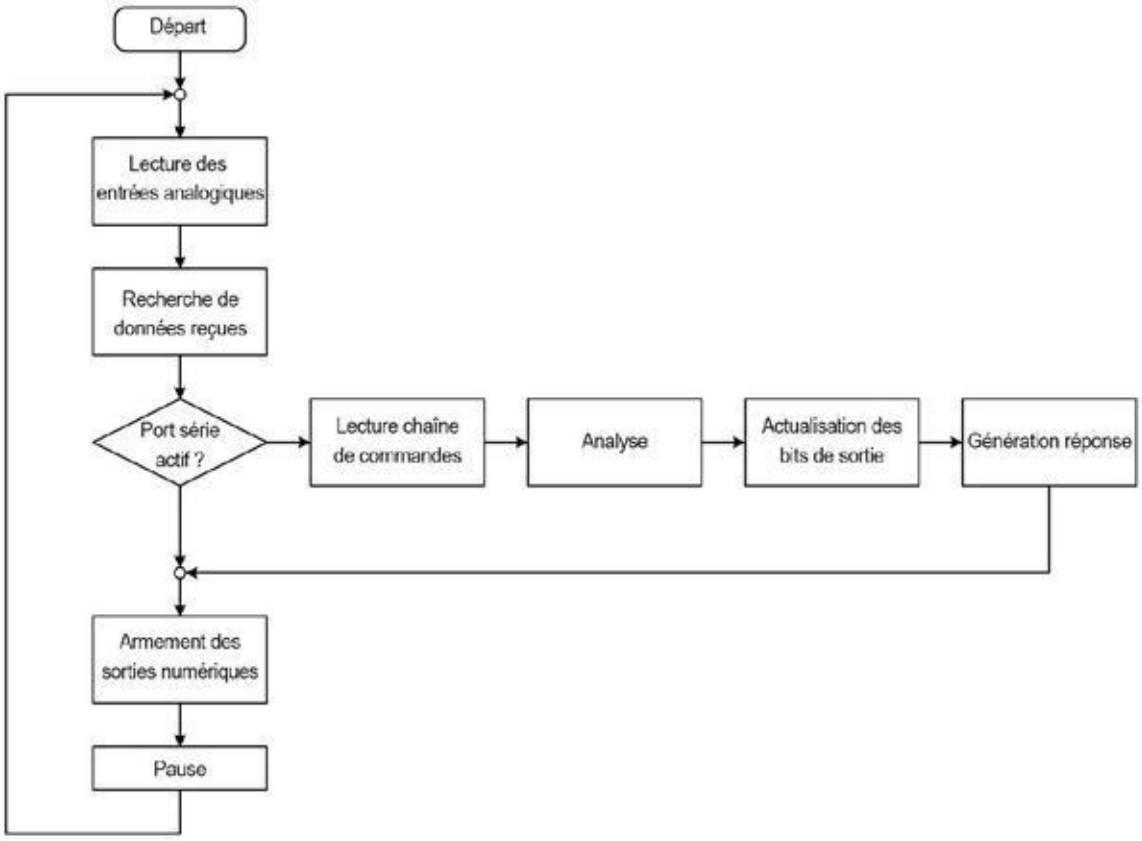


Figure 10.23 : Diagramme fonctionnel du logiciel du Switchinator.

Le projet utilise un protocole simplifié de commande/réponse, c'est-à-dire que chaque commande qu'il reçoit entraîne le renvoi d'une réponse simple. L'utilisateur ne va jamais être à l'origine d'un dialogue. Le protocole est schématisé par le Tableau 10.12.

Tableau 10.12 : Éléments du protocole commande/réponse du projet.

Commande	Réponse	Description	Format In/out
A:n	A:n:val	Lit entrée anal. n en format brut	Hexa
R:nM	R:n:val	Lit état de la sortie n	Hexa (0 ou 1)
W:n:val	OK	Écrit 0 ou 1 sur la sortie n	Hexa
S:val	OK		

		Écrit toutes les sorties en hexa	Hexa sur 4 chiffres
G:?	G:val	Lit en hexa toutes les sorties	Hexa sur 4 chiffres

La commande d’écriture W requiert deux paramètres, toutes les autres n’en réclamant qu’un. Toutes les valeurs sont au format hexadécimal, ce qui permet de symboliser les numéros des ports par un seul chiffre hexa. Les valeurs des entrées analogiques de la commande A et celles des commandes de lecture et d’écriture collectives S et G utilisent des valeurs sur quatre chiffres hexa. Bien sûr, les chiffres 0 et 1 sont identiques en décimal et en hexadécimal.

La détection d’erreur est réalisée en analysant la chaîne que renvoie le Switchinator. Pour les commandes A, R et G, si la machine hôte reçoit en retour la chaîne qu’elle a envoyée, c’est qu’il y a eu une erreur. Pour les commandes S et W, en cas de succès, le projet renvoie la chaîne « OK ».

Les notations octale et hexadécimale

Si vous savez déjà compter par paquets de huit ou de seize, vous connaissez déjà les notations octales et hexadécimales. Dans ce cas, vous pouvez ignorer cet encadré. Dans le cas contraire, je vous propose un petit historique afin d’expliquer pourquoi sont apparues ces notations différentes du système décimal et comment on les utilise.

Au milieu du xx^{e} siècle, le moins puissant des ordinateurs occupait une salle entière et chauffait assez pour que tous les techniciens puissent se réchauffer leur gamelle. On s’est très vite rendu compte qu’on ne pouvait plus continuer à compter selon la notation binaire (uniquement avec des 0 et des 1). Hélas, la notation la plus utilisée sur Terre actuellement, la base 10, ne coïncide pas bien du tout avec la notation binaire. Autrement dit, les séquences de bits à 0 et à 1 ne se traduisent pas aisément en valeurs décimales. Et ce sont ces bits qui sont représentés par des lumières allumées ou éteintes sur les panneaux de programmation et stockés en mémoire.

Voilà pourquoi on a cherché une base de notation autre que la base 10. Les deux bases qui sont bien apparentées au binaire sont la base octale et la base

hexadécimale, puisque ce sont des multiples de deux.

La base 8 a été très utilisée à l'époque où les ordinateurs étaient construits pour traiter les données binaires par paquets de 12, 24 ou 36. Ces trois valeurs sont des multiples de trois. Et un chiffre écrit en octal ne pouvant avoir qu'une valeur entre 0 et 7, il occupe trois bits. Le Tableau 10.13 montre les équivalences pour certaines valeurs remarquables.

Tableau 10.13 : Quelques valeurs en notation octale pour une machine sur 12 bits.

Décimal	Binaire	Octal	Décimal	Binaire	Octal
0	000 000 000 000	0000	9	000 000 001 001	0011
1	000 000 000 001	0001	10	000 000 001 010	0012
2	000 000 000 010	0002	20	000 000 010 100	0024
3	000 000 000 011	0003	30	000 000 011 110	0036
4	000 000 000 100	0004	40	000 000 101 000	0050
5	000 000 000 101	0005	50	000 000 110 010	0062
6	000 000 000 110	0006	100	000 001 100 100	0144
7	000 000 000 111	0007	200	000 011 001 000	0310
8	000 000 001 000	0020	511	000 111 111 111	0777

Il reste de nos jours un domaine dans lequel la notation octale reste en vigueur : sur les machines Unix et Linux modernes, les bits des droits d'accès aux fichiers et aux répertoires sont codés en base 8. Il n'y a plus de nos jours d'ordinateurs fonctionnant sur 12, 24 ou 36 bits de largeur de mot (certains groupes de recherche ont créé des microprocesseurs sur 24 bits pour des usages très particuliers). Il existe même un circuit de traitement du signal de FreeScale qui peut fonctionner en 24 bits, le DSP 56303.

La notation octale est devenue peu pratique lorsque les ordinateurs se sont standardisés pour adopter une largeur de mots multiple de quatre pour les

données et pour les adresses (16 bits, 32 bits, 64 bits, *etc.*). La base la plus confortable devenait la base hexadécimale ou base 16. En notation hexa, chaque chiffre représente une valeur entre 0 et 15, c'est-à-dire les valeurs binaires de 0000 à 1111. Autrement dit, un seul chiffre hexa symbolise un quartet de bits, et deux chiffres hexa correspondent à un octet, donc 8 bits.

Le problème qu'il a fallu résoudre est qu'il nous manquait des symboles pour représenter les valeurs pour lesquelles il n'existait pas de dessin en notation décimale, c'est-à-dire les valeurs de 10 à 15. Plusieurs solutions avaient été proposées dès les années 1950, et certaines semblent de nos jours vraiment étranges. Après quelques années de tâtonnement, il a été choisi d'utiliser les lettres majuscules A, B, C, D, E et F pour les valeurs de 10 à 15. Le Tableau 10.14 donne quelques exemples d'équivalences entre les bases décimale, binaire et hexadécimale.

Tableau 10.14 : Quelques valeurs hexadécimales pour un ordinateur 16 bits.

Décimal	Binaire	Hexa	Décimal	Binaire	Hexa
0	0000 0000 0000 0000	0000	10	0000 0000 0000 1010	000A
1	0000 0000 0000 0001	0001	20	0000 0000 0001 0100	0014
2	0000 0000 0000 0010	0002	30	0000 0000 0001 1110	001E
3	0000 0000 0000 0011	0003	40	0000 0000 0010 1000	0028
4	0000 0000 0000 0100	0004	50	0000 0000 0011 0010	0032
5	0000 0000 0000 0101	0005	100	0000 0000 0110 0100	0064

6	0000 0000	0000	0000	0006	200	0000	0000	1100	00C8
	0110					1000			
7	0000 0000	0000	0000	0007	255	0000	0000	1111	00FF
	0111					1111			
8	0000 0000	0000	0000	0008	511	0000	0001	1111	01FF
	1000					1111			
9	0000 0000	0000	0000	0009	4095	0000	1111	1111	0FFF
	1001					1111			

Dans le code source des programmes informatiques, vous rencontrerez des valeurs hexa : dans les langages C et C++, on écrira par exemple 0x3F, en assembleur on écrira 3Fh, en langage Forth $3F ou encore %3F dans une adresse URL. Les afficheurs à sept segments parviennent à afficher les chiffres hexadécimaux en utilisant selon le cas une majuscule ou une minuscule pour maintenir la lisibilité : A, b, c, d, E et F (Figure 10.24).

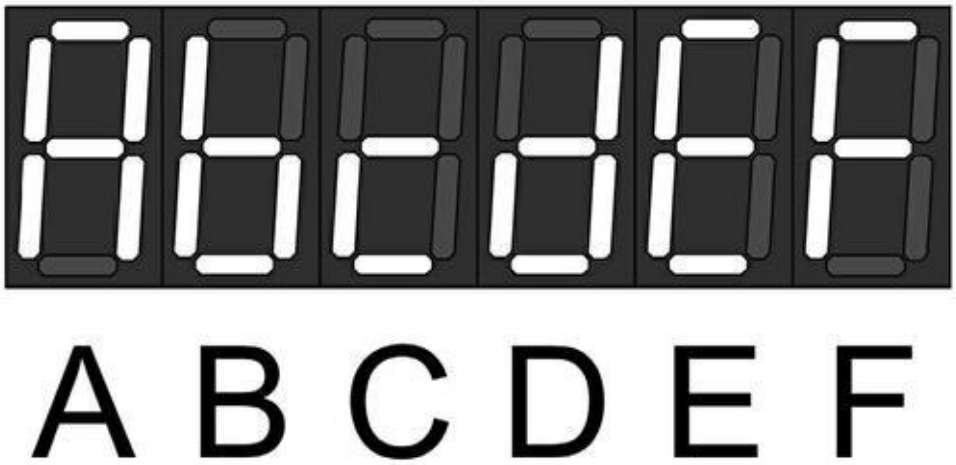


Figure 10.24 : Afficheur à cristaux liquides à sept segments montrant les six chiffres hexadécimaux spécifiques.

Que ce soit en octal ou en hexadécimal, on a vite constaté qu'il y avait des valeurs magiques, situées à certaines frontières. En hexa, la valeur 0xFF correspond à 1 octet dont tous les bits sont à 1. De même, 0x5A5A est un motif alternatif qui donne en binaire 0101 1010 0101 1010. Ce motif est souvent

utilisé pour effacer le contenu d'un disque dur ou tester l'utilisation de la zone de pile mémoire. La valeur 0x7F correspond à 127 en décimal, 0x1FF vaut 511 et 0x3FF vaut 1023. Cette valeur 0X3FF est d'ailleurs la valeur maximale du convertisseur analogique/ numérique d'un microcontrôleur ATmega328, puisqu'il fonctionne sur dix bits. Le point milieu de cette plage correspond donc à 0x1FF, soit 511.

Rien n'empêche de faire de l'arithmétique en octal ou en hexadécimal. C'est d'ailleurs souvent nécessaire en langage assembleur pour ajouter ou soustraire une valeur d'adresse à une autre pour les sauts directs ou les branchements relatifs. Bien sûr, la multiplication et la division en hexa réclame un peu d'entraînement. À moins d'avoir beaucoup de programmation en assembleur à faire, vous n'aurez sans doute pas besoin de devenir un expert en calcul hexadécimal. En revanche, il est indispensable de savoir convertir une valeur de la base 2 vers la base 16, chiffre par chiffre, si vous voulez bien exploiter un microcontrôleur.

La commande A de notre protocole accepte en paramètre un seul chiffre entre zéro et trois. La valeur analogique correspond à celle qui est renvoyée par le convertisseur du microcontrôleur. Elle est renvoyée en base 16 sur 1 à 3 chiffres, la valeur maximale étant égale à 0x3FF.

Pour les commandes de lecture R et d'écriture W, il faut fournir le numéro du port sous forme d'un chiffre hexa entre 0 et 0xF (soit 15). Les canaux de sortie de 7 à 0xF correspondent aux diodes LED de la carte ; ils ne sont pas traités par les pilotes ULN2003A.

Le Tableau 10.12 précédent permet de voir qu'il faut fournir une valeur hexadécimale pour lire ou modifier l'état de plus d'une sortie à la fois avec les commandes S et G. Cette valeur est une sorte de peigne qui permet d'activer plusieurs sorties sans qu'elles soient voisines dans la représentation binaire, et de lire l'état de toutes les sorties.

Supposons par exemple que nous voulions activer seulement les sorties 5, 6, 12 et 13. En rappelant que la numérotation des sorties commence à zéro, il faut donc mettre à un les sixième, septième, treizième et quatorzième bits, ce qui donne la valeur hexadécimale 3060h, comme le montre ce petit schéma :

```
Bit    ¦ 15   11    7    3  0
-------+----+----+----+---
Binary ¦  0011 0000 0110 0000
```

```
Hex     ¦     3     0     6     0
```

Je rappelle que les numéros des canaux commencent à zéro. Dans le motif binaire ci-dessus, la lecture commence par la droite.

La valeur hexadécimale qui est envoyée modifie l'état des sorties en conséquence. Si la sortie est actuellement active, et si la valeur indique zéro à cet endroit, elle sera désactivée.

Pour ne pas modifier l'état des autres bits que ceux qui vous intéressent, appliquez le cycle lecture/modification/écriture : vous lisez les bits, vous changez la valeur de certains, et vous récrivez les bits. C'est de cette façon que fonctionne la commande d'écriture W.

La dernière action de notre boucle principale est le transfert de l'état des sorties numériques vers le matériel. Si aucun changement n'a été apporté depuis la dernière mise à jour, il n'y aura pas de changement sur les sorties. Dans le cas contraire, les sorties changeant d'état provoquent une action sur les relais ou éléments pilotés par les sorties du ULN2003A.

Prototype

Notre prototype va se concentrer sur l'interface de communication avec la machine hôte, *via* le protocole de commande/réponse et donc de l'interface RS-232. Nous pouvons monter le prototype sur une plaque d'expérimentation sans soudure, sur laquelle nous plaçons un microcontrôleur ainsi qu'un module RS-232 qui va servir d'équivalent au circuit MAX232 de la version finale. Le montage du prototype est visible dans la Figure 10.25.

Le module RS-232 est animé par un circuit MAX3232, totalement compatible avec le circuit MAX232 que nous allons adopter pour la version définitive. Le MAX3232 fonctionne normalement en 3,3 V, mais il tolère le 5 V. Il coûte plus cher que le MAX232. Le module que nous choisissons d'utiliser est représenté dans la Figure 10.26.

Vous trouverez ce genre de module à moins de 10 € sur le Web. Faites une recherche de l'expression « Arduino RS-232 module ».

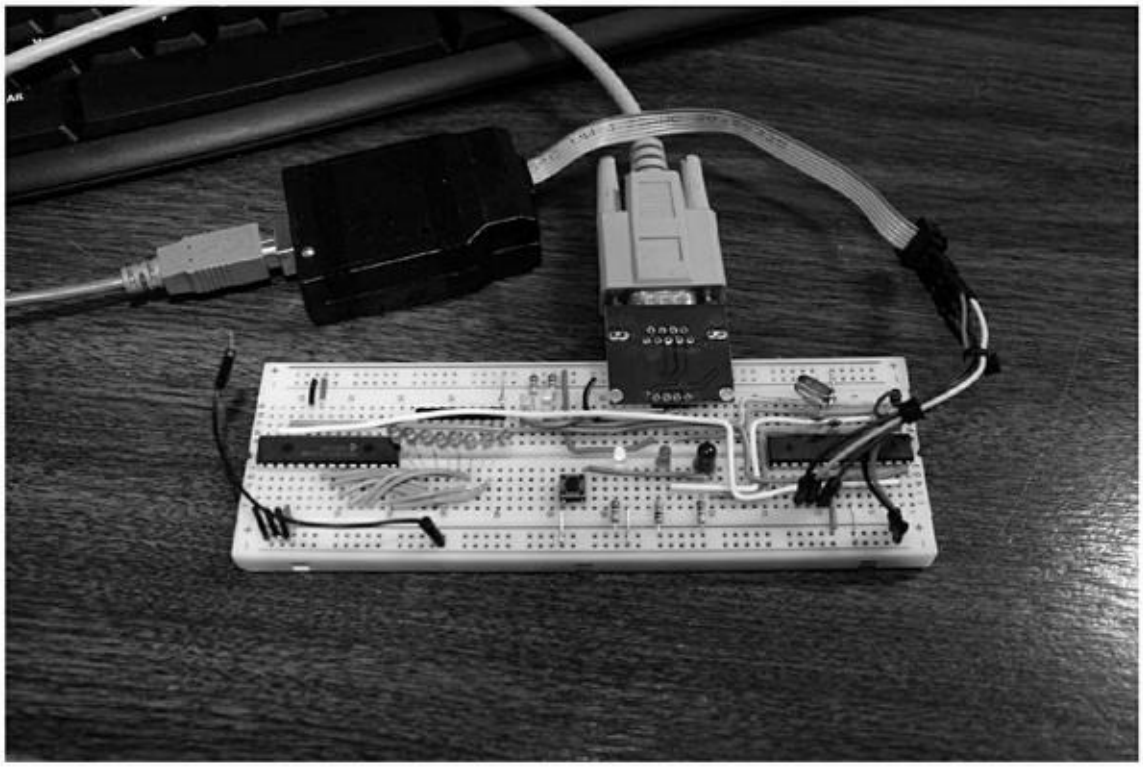


Figure 10.25 : Montage du prototype du Switchinator.

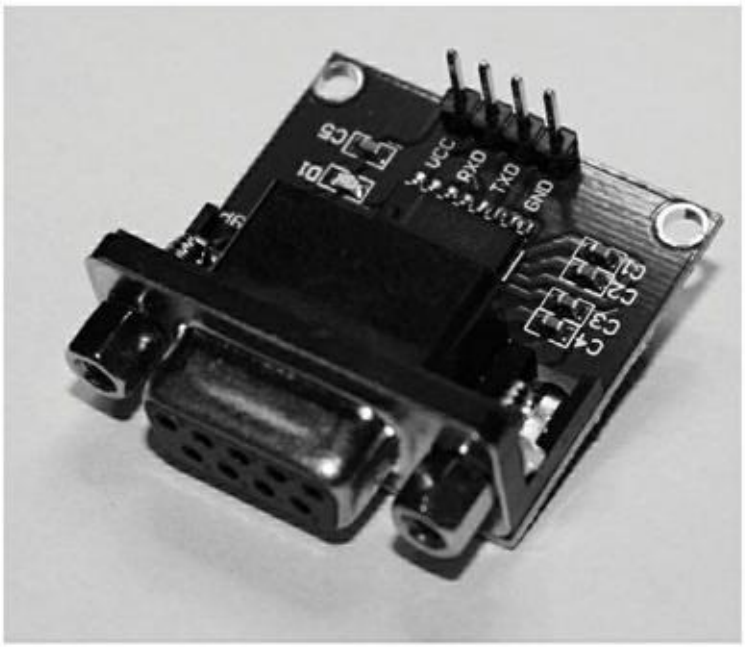


Figure 10.26 : Module interface RS-232.

De nombreux PC de bureau (des tours) sont encore équipés d'un connecteur DB9 et donc d'un port RS-232 à l'arrière. Si votre machine est un portable, il n'y a sans doute plus de port série. Dans ce cas, vous opterez pour un adaptateur USB vers RS-232. Vous en trouverez à partir de 5 euros, mais cela peut monter jusqu'à plusieurs dizaines d'euros. Pour en savoir plus, je vous invite à consulter la bibliographie dans l'Annexe D.

Configurer les fuse bits du microcontrôleur pour un cristal à 16 MHz

À la différence des autres projets du livre, ce projet n'est pas une carte Arduino. Il devient compatible au niveau logiciel si vous installez le logiciel d'amorçage bootloader, mais ce n'est pas obligatoire. C'est un projet construit autour d'un microcontrôleur de la famille AVR.

La conséquence est que le microcontrôleur n'est pas préconfiguré comme il l'est dans une carte Arduino. Lorsque vous achetez le circuit intégré Atmel, il utilise par défaut son horloge interne à circuit RC dont la fréquence est à peu près de 8 MHz. Si vous installez l'amorceur, plusieurs options internes sont armées pour indiquer qu'une partie de l'espace mémoire flash est réservée à cet effet. En général, dans un microcontrôleur sur lequel a été chargé l'amorceur, le bit fusible qui demande d'utiliser une horloge externe avec un cristal de 16 MHz est armé. Ce n'est pas le cas du circuit que vous achetez nu.

Il faut donc apprendre à reconfigurer le microcontrôleur pour qu'il utilise le cristal externe, et cela suppose de modifier des fuse bits. J'ai donné dans le Chapitre 3 tous les détails des fuse bits des microcontrôleurs AVR. Bien sûr, si vous avez besoin d'en savoir plus, vous plongerez dans la description technique d'Atmel.

Si vous disposez d'un système Linux et avez besoin de préparer la configuration Arduino pour pouvoir utiliser AVRDUDE, il suffit d'émettre la commande suivante dans le cas d'un ATmega328 :

```
sudo avrdude -cusbtiny -p atmega328 -U
lfuse:w:0xFF:m -U hfuse:w:0xDE:m \
  -U efuse:w:0x05:m
```

Pour la variante ATmega328p, il suffit de modifier le nom du processeur en conséquence dans le paramètre -p de la commande précédente :

```
sudo avrdude -cusbtiny -p atmega328p -U
lfuse:w:0xFF:m -U hfuse:w:0xDE:m \
  -U efuse:w:0x05:m
```

Si vous prenez soin sous Linux d'armer les droits d'accès pour les entrées-sorties USB, vous n'aurez plus besoin d'utiliser **sudo** en préfixe pour lancer **avrdude**. Pour tout savoir au sujet des options de ligne de commande et des commandes interactives d'avrdude, visitez la page http://www.nongnu.org/avrdude.

Il reste à prévenir le compilateur que vous fonctionnez à 16 MHz en définissant le symbole F_CPU de la manière suivante :

```
#define F_CPU 16000000UL
```

Lorsque vous utilisez une carte Arduino standard, cette ligne est ajoutée automatiquement par l'environnement. Si vous utilisez une carte spécifique, c'est à vous d'indiquer la fréquence d'horloge.

Les versions Windows et macOS d'avrdude fonctionnent comme sous Linux : pour Windows, voyez par exemple la page suivante :

https://skyduino.wordpress.com/2011/12/02/tutoriel-avrdude-en-ligne-de-commande

Le logiciel du prototype

La principale différence entre le logiciel de la version prototype et celui de la version finale est l'absence des fonctions qui modifient les sorties numériques *via* le circuit de MCP23017. Rappelons que dans le prototype, nous voulons vérifier le bon fonctionnement du protocole de dialogue. L'état des sorties est simplement représenté sous forme de bits qui changent dans un mot de 16 bits.

La section suivante, qui présente la version définitive du logiciel, va plus loin dans la description. Je ne donne donc pas de détails ici. Nous compilons le code source avec l'interface Arduino IDE puis téléchargeons le résultat vers le microcontrôleur grâce au module d'interface ICSP USBtinyISP (d'Adafruit).

J’ai choisi d’utiliser l’atelier IDE Arduino pour qu’il prenne en charge les erreurs de compilation et tous les détails, mais j’ai décidé de ne pas utiliser l’éditeur intégré en modifiant les préférences. Dans la mesure où je gagne ma vie en écrivant des logiciels scientifiques et commerciaux, j’ai pris des habitudes au niveau de mon éditeur de texte. Je n’aime pas trop celui fourni avec l’IDE Arduino. Pour le modèle de la carte, j’ai choisi l’option **Duemilanove with ATmega328** (vous choisirez votre modèle). Pour le programmateur, j’ai choisi **USBtinyISP**.

Sous Linux, le module USBtinyISP n’utilise pas de pseudo-port série, mais communique directement avec le sous-système des entrées-sorties USB. Si vous essayez d’utiliser l’atelier Arduino IDE avec le programmateur, vous devriez normalement voir apparaître une erreur de droit d’accès. Vous pourriez dans ce cas lancer l’atelier avec sudo, mais ce n’est pas une très bonne manière d’effectuer le transfert du code. Mieux vaut ajouter une règle d’accès au pilote UDF. Voyons comment faire.

Commencez par créer un nouveau fichier dans le sous-répertoire */etc/udev/rules.d* et donnez-lui le nom *tinyusb.rules*. Comme contenu du fichier, saisissez la chaîne suivante (une seule ligne) :

```
SUBSYSTEM=="usb", ATTR{product}=="USBtiny",
ATTR{idProduct}=="0c9f", \
  ATTRS{idVendor}=="1781", MODE="0660", GROUP="dialout"
```

Personnellement, j’ai utilisé l’éditeur vi avec sudo pour y parvenir :

```
sudo  vi  tinyusb.rules
```

Pour se rendre compte de la nouvelle règle, il suffit de redémarrer le sous-système udev, comme ceci :

```
sudo restart udev
```

D’autres modèles de programmateurs sont utilisables, et vous pouvez même utiliser une seconde carte Arduino comme montré dans le Chapitre 6. Je dispose par exemple d’un Atmel-ICE, mais j’ai décidé de ne pas l’utiliser sous Linux parce que je ne disposais pas de la plus récente version d’avrdude et j’étais trop flemmard pour lancer une reconstruction et me plonger dans les fichiers de configuration. Si vous utilisez Windows, vous pouvez vous servir du logiciel

AVR Studio d'Atmel. Sous Linux, le petit gadget d'Adafruit a parfaitement rempli la tâche demandée.

Dans ma configuration de l'atelier IDE, le fait de demander le téléchargement avec l'icône force le logiciel à tenter d'effectuer le transfert *via* la connexion USB vers série standard pour les cartes Arduino (et accepté par un convertisseur USB vers série tel que celui mentionné plus haut). Mais cette technique ne fonctionne pas avec le module USBtinyISP. Il faut dans ce cas utiliser la commande **Croquis/Téléverser avec un programmateur**.

J'ai constaté que le logiciel de transfert avrdude mettait un certain temps à démarrer. Une fois qu'il est lancé, et qu'il a établi une communication fiable avec le microcontrôleur, les choses vont beaucoup plus vite. Vous pouvez vous en rendre compte dans l'atelier IDE en lançant avrdude depuis la ligne de commande. Soyez patients si vous avez l'impression que les choses sont bloquées. Ce n'est pas le cas. S'il y a vraiment un problème, avrdude finira par afficher un message d'erreur.

Test du prototype

Les tests du prototype sont très simples, puisqu'ils se concentrent sur la fonction d'analyse de ligne de commande. En prouvant que nous parvenons à interagir avec le logiciel, nous serons assurés que l'interface RS-232 fonctionne correctement. Je suppose bien sûr que le module RS-232 externe fonctionne comme prévu.

Nous commençons par tester la commande de sortie OUT, la commande de statut ST et la commande d'entrée AN. Les deux commandes `OUT : A : 1` et `OUT : A : 0` servent à forcer toutes les sorties à l'état allumé et éteint. L'état de chaque sortie est maintenu en mémoire. Voilà pourquoi nous n'avons pas besoin de matériel à piloter pour l'instant.

Dans la commande `ST : n : ?` , le paramètre *n* doit être un chiffre hexadécimal entre 0 et 13 (ce dernier correspondant à la lettre D). Les deux commandes `OUT : n : 0` et `OUT : n :` 1 attendent aussi un seul chiffre. Si le logiciel fonctionne comme prévu, vous devriez pouvoir forcer à zéro toutes les sorties puis activer et désactiver de façon sélective chacune d'elles entre 0 et 13, sans modifier l'état des autres.

Pour tester la commande d'entrée analogique AN, il faut appliquer une source de tension variable entre 0 et +5 V aux entrées A0 à A3 puis demander de lire la valeur. En faisant varier la tension, vous devriez voir la valeur évoluer. Cette valeur est renvoyée en hexadécimal sur trois chiffres dont les deux bits les plus significatifs sont toujours à zéro, puisque le microcontrôleur ne convertit que sur 10 bits.

Les commandes `SP : val` et `GP : ?` attendent une valeur sur quatre chiffres hexa. Pour les tester, nous forçons à un toutes les sorties impaires et à zéro toutes les sorties paires puis vérifions l'état des sorties avec la commande ST. Dans un second temps, nous forçons à zéro les sorties impaires et à un les sorties paires puis lisons à nouveau les états avec ST.

N.d.T. : Ne démontez par le prototype pour l'instant. Nous en aurons besoin pendant la fabrication du circuit définitif.

Le logiciel définitif

En général, le logiciel qui sert au prototype est très simplifié, mais dans le cas du projet Switchinator, même le prototype utilise plusieurs modules de code source. Le fichier principal porte bien sûr le nom *Switchinator.ino*. Les autres fichiers se répartissent les définitions globales, les variables globales, la fonction d'analyse de commande, le générateur de réponse et les fonctions de contrôle des entrées-sorties. Notez que la version prototype n'a pas besoin du module d'entrées-sorties *sw_io.cpp*.

Structure du code source

Le code source comporte huit fichiers qui sont présentés dans le Tableau 10.15. Le module principal contient les deux fonctions obligatoires `setup()` et `loop()` et fait référence aux autres modules avec des directives `#include`.

Tableau 10.15 : Modules de code source du projet Switchinator.

Module	Fonction
Switchinator.ino	Module principal avec setup() et loop()

sw_defs.h	Constantes (#define)
sw_gv.cpp	Variables globales
sw_gv.h	Fichier #include
sw_io.cpp	Fonctions d’entrée-sortie matérielles
sw_io.h	Fichier #include
sw_parse.cpp	Analyse de commande
sw_parse.h	Fichier #include

Le code source de ce projet est disponible dans le référentiel GitHub et sur le site Web de l’éditeur dans le fichier archive des exemples.

Description fonctionnelle du logiciel

Comme tout programme pour Arduino, l’exécution commence dans le fichier principal (Listing 10.9).

Listing 10.9 : Code source de Switchinator.ino.

```
//-------------------------------------------------------
------------
// Switchinator.ino

#include <stdint.h>
#include <Wire.h>

// Librairie MCP23017
#include <IOexp.h>

// Accès aux autres modules source
#include "sw_parse.h"
#include "sw_gv.h"
#include "sw_defs.h"
#include "sw_io.h"

// Fréquence d’horloge CPU
#define F_CPU 16000000UL
```

```
bool waitinput = false;
bool usedelay = false;

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

  // Bienvenue
  Serial.println();
  Serial.println("SWITCHINATOR V1.0");
  Serial.println("READY");
  Serial.println("####"); // Drapeau de démarrage
}

void loop()
{
  // Lecture entrées analog et acua tableau var glob.
  ScanAnalog();
  // Affiche une invite
  if (!waitinput) {
    waitinput = true;
    Serial.print(INPPCH);
  }

  // Scrute arrivée commande
  if (GetCommand()) {
    waitinput = false; // Bascule drapeau d'attente
  }
  else {
    ClearBuff(0);
    ResetBuffLen();
    // Pas de commande donc délai de boucle
    usedelay = true;
  }

  // Analyse de commande reçue
  if (DecodeCommand()) {
    // Renvoi réponse à l'hôte (ou l'utilisateur)
    Serial.println();
    Serial.print(OUTPCH);
    Serial.println(gv_cmdinstr);
```

```
    }
    // Actualisation sorties numériques par bits d'état
    SetDigBits();

    // Bref délai si pas de commande
    if (usedelay) {
      usedelay = false;
      delay(50);
    }
  }
```

La fonction de préparation contient les instructions d'initialisation habituelles et le message de bienvenue. La fonction de boucle principale procède à une mise à jour périodique des bits de sortie et vérifie l'arrivée d'une commande depuis la machine hôte en appelant la fonction `GetCommand()` définie dans le module *sw_parse.cpp*.

Le bloc de lecture de chaîne de commande de la Figure 10.23 est justement incarné par cette fonction `GetCommand()`. Les deux blocs d'analyse et de génération de réponse correspondent à la fonction `DecodeCommand()` et le bloc de mise à jour des bits de sortie correspond à la fonction `SetDigBits()` définie dans le module *sw_io.cpp*. Dans la Figure 10.27, vous pouvez voir de quelle manière le tampon mémoire des états des bits de sortie, *`gv_statebits`*, est utilisé pour mémoriser l'état de chacun des bits. Toutes les opérations de lecture et d'écriture fonctionnent avec cette structure en mémoire, et non avec les sorties numériques réelles.

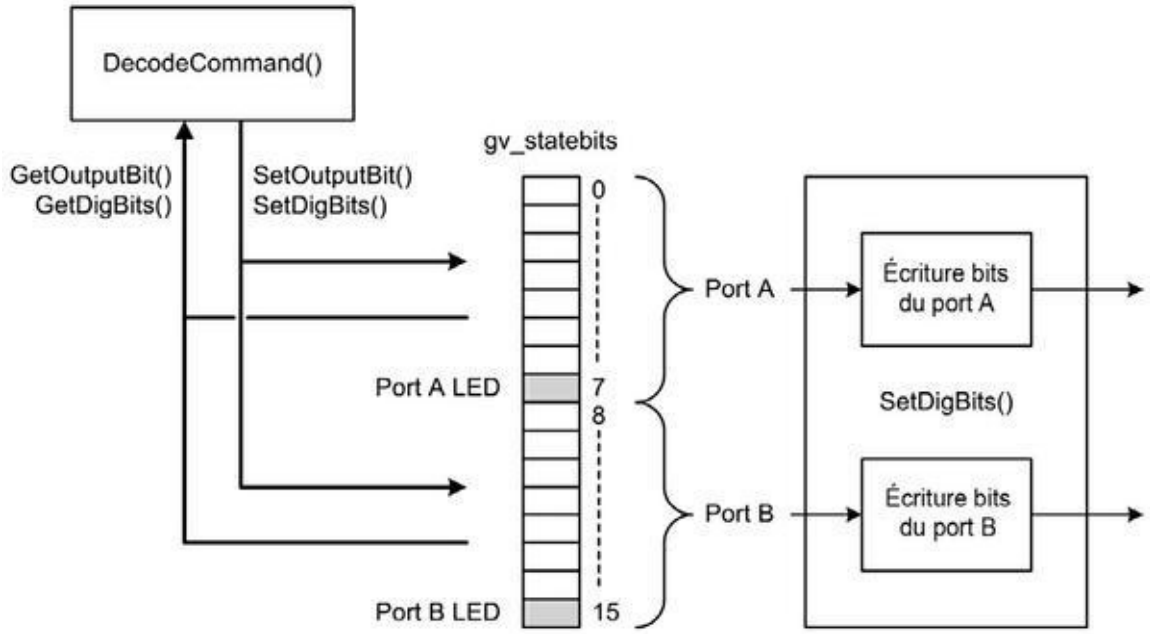


Figure 10.27 : Fonctionnement du tampon mémoire gv_statebits.

Je tiens à préciser qu'écrire la valeur 0x00 sur le port A ou le port B active la diode LED associée. Pour l'éteindre, il suffit d'écrire la valeur 0xFF sur le même port. Les bits de sortie sont actualisés une fois toutes les 50 ms environ, s'il n'y a aucune commande détectée en entrée à traiter.

Fabrication

Pour ce projet, j'ai choisi l'outil de conception électronique Fritzing, qui réunit une plaque d'expérimentation virtuelle, un logiciel de création de schéma et un outil de routage de circuit imprimé. Il est facile à installer et assez simple à utiliser. Fritzing dispose d'une très vaste communauté d'utilisateurs et sa librairie de composants offre un vaste choix. Enfin, et ce n'est pas pour nous déplaire, il est gratuit. Dans certaines distributions Linux, vous obtiendrez une version qui date un peu, mais pourrez télécharger la plus récente depuis le site Web de référence (`http://fritzing.org`). Pour ce projet, j'ai utilisé la version 0.8.5 sous Ubuntu 14.04 STS, donc sous Linux. Il existe bien sûr des versions de Fritzing pour Windows et macOS.

Comparé aux outils dont j'avais l'habitude, j'ai été agréablement surpris par la facilité d'usage de l'interface de Fritzing. En revanche, je n'ai pas été très impressionné par sa fonction d'autoroutage, mais j'ai vu les fonctions

d’autoroutage de logiciels commerciaux avoir eux aussi de sérieux soucis pour résoudre certains défis de routage. Il faut dire que créer un programme d’autoroutage est complexe. N’en attendez pas de miracle. J’ai fini par faire le routage à la main, ce qui se voit, bien sûr. Le contrôle des règles de conception (DRC) a bien fonctionné. L’éditeur de schéma de principe avait parfois un aspect étrange, mais il était parfaitement utilisable.

Ce qui me gêne le plus avec Fritzing, c’est la librairie de composants. Il semble que les contributeurs n’utilisent pas tous les mêmes conventions pour les symboles en termes d’encombrement physique. Autrement dit, les composants d’une librairie peuvent être tout petits comparés à ceux d’une autre librairie standard de Fritzing qui contiendra des symboles dans une taille normale. C’est visible dans la Figure 10. 21. À moins que tous les composants dont vous avez besoin fassent partie de la même librairie ou soient réalisés selon les mêmes conventions, vous aurez du mal à obtenir des lignes bien alignées sans être forcé de faire de sérieuses retouches et de jouer avec la taille de la grille et les fonctions d’accroche.

Les dimensions extérieures de notre circuit imprimé vont être de 124 mm sur 96 mm. Autrement dit, c’est plus que ce qu’accepte la version gratuite d’Eagle, et c’est une autre raison pour laquelle j’ai adopté Fritzing.

Une fois que le schéma de principe est terminé (nous l’avons montré en Figure 10.21), nous pouvons passer au dessin du circuit imprimé. Lors du premier affichage, il n’y a bien sûr aucune piste, uniquement les lignes directes entre points de connexion. Dans la Figure 10.28, j’ai déjà repositionné les composants comme je pense qu’ils doivent se placer, et Fritzing affiche les lignes directes. Il suffit de cliquer et maintenir sur un contact pour sélectionner en surbrillance tous les autres points auxquels il doit être connecté, l’affichage étant en jaune clair.

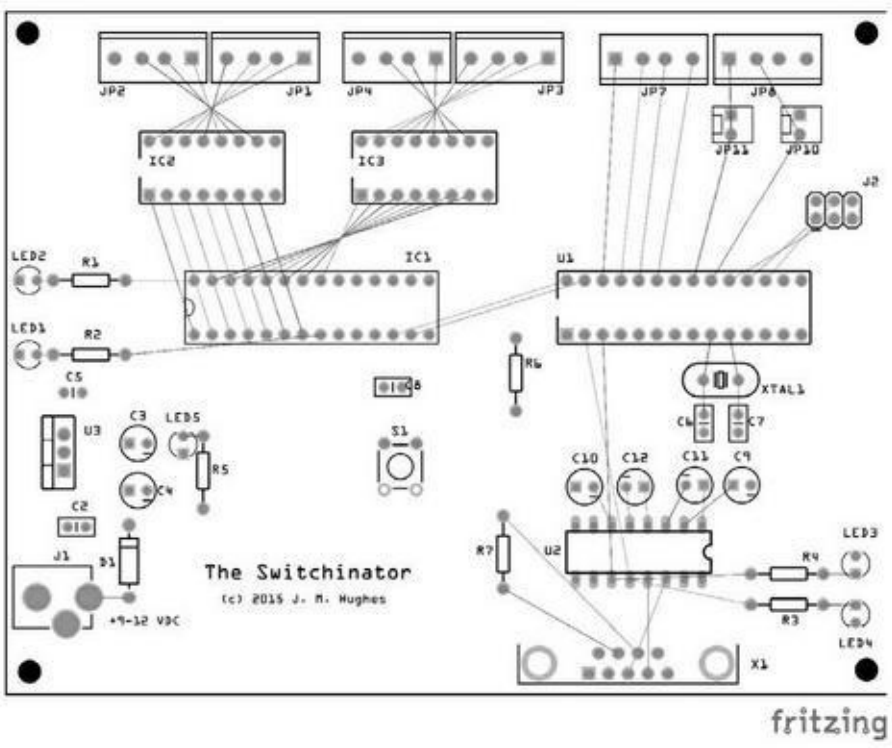


Figure 10.28 : Circuit imprimé après positionnement des composants, mais avant routage.

Nos composants sont groupés par fonction. L'alimentation électrique occupe le quart inférieur gauche, le quart supérieur gauche est dédié au MCP23017 et ses deux circuits ULN2003A. Le microcontrôleur, l'oscillateur, le connecteur ICSP et les entrées analogiques sont en haut à droite et l'interface série est en bas à droite.

Du fait de que j'ai choisi de n'utiliser aucun composant monté en surface, le routage est parfois assez tortueux. J'utilise des traversées pour m'en sortir en passant d'un côté à l'autre du circuit. La Figure 10.29 montre la version finale du typon, tel que je l'ai envoyé pour fabrication.

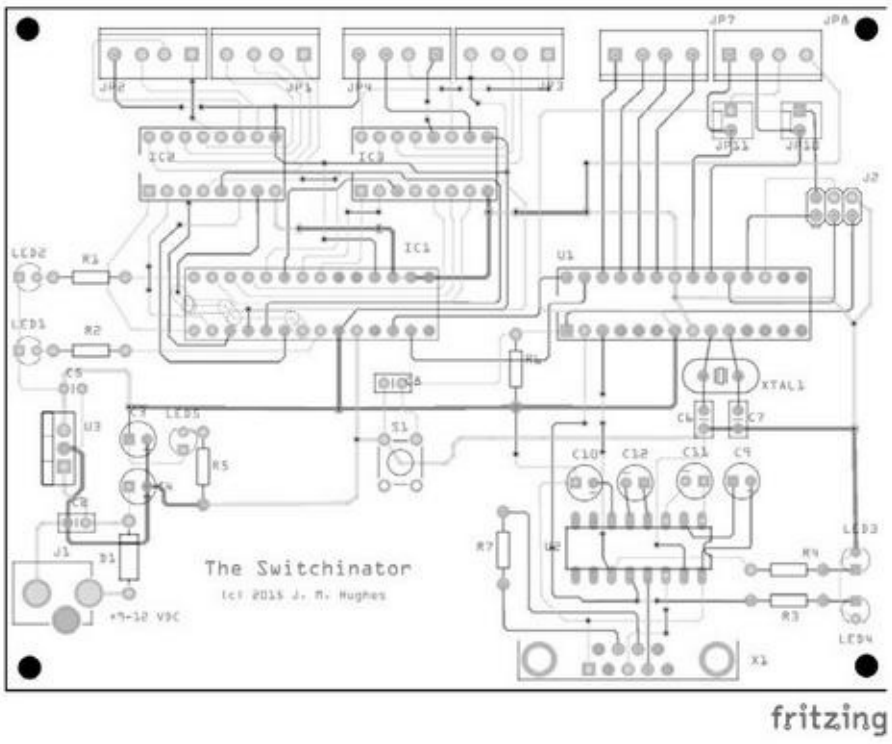


Figure 10.29 : Version finale du typon du Switchinator.

Comme pour le précédent projet, j'ai vérifié la qualité du fichier au format Gerber au moyen de l'outil de visualisation Gerbv (Figure 10.30).

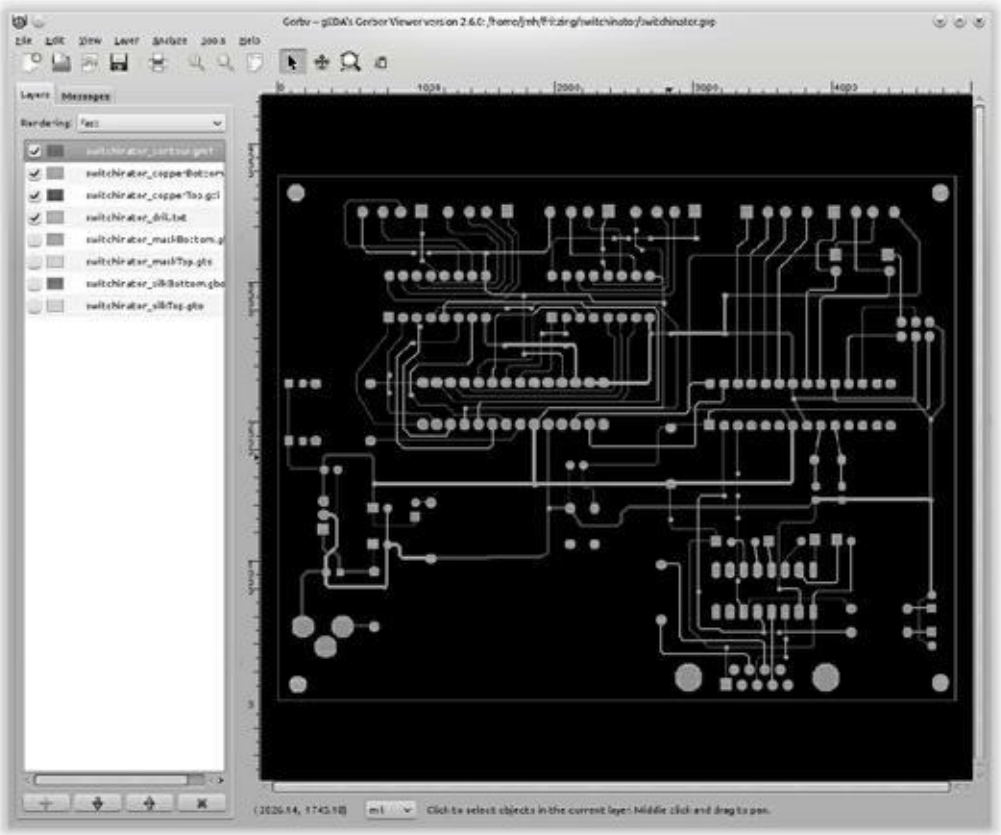


Figure 10.30 : Visionnage de contrôle du typon avec Gerbv.

Le fournisseur auquel j'ai demandé de fabriquer le circuit (Advanced Circuits) a été assez aimable pour recontrôler les pistes. Il a ainsi découvert deux plots qui étaient presque connectés, mais pas vraiment. J'aurais pu assez facilement détecter cette erreur moi-même avant envoi, mais cela aurait été un sérieux défi de trouver la cause de la panne une fois le circuit monté. Je les remercie encore d'avoir revérifié le typon.

Le montage est facile puisque je n'utilise que des composants traversants. Le circuit imprimé avant implantation des composants est montré dans la Figure 10.31.

Vous commencez par souder le jack de l'alimentation et le régulateur de tension 7805 avec les composants accessoires, c'est-à-dire D1, C2, C3, C4 et C5. Vous constatez qu'il n'y a pas de C1. Ne le cherchez pas ; il était prévu dans une première version. Pour pouvoir tester l'alimentation, il nous faut également installer la résistance R5 et la diode LED5. Nous pouvons alors connecter un

adaptateur secteur 9 à 12 V courant continu avec le jack pour vérifier que le régulateur fournit bien 5 V sur l'anode de la LED5 qui doit s'allumer.

Nous passons ensuite au connecteur RS-232 qui est légendé X1 sur le schéma. Nous installons le circuit U2 MAX232 puis les quatre condensateurs C9 à C12. Je vous avais demandé de conserver la plaque d'essai du prototype. Nous pouvons maintenant relier les fils des signaux Rx et Tx aux broches 2 et 3 du circuit microcontrôleur. Nous pouvons alors lancer le logiciel déjà téléchargé sur le microcontrôleur afin de vérifier que l'interface RS-232 fonctionne correctement. N'oubliez pas de relier la masse et le 5 V du circuit à la plaque d'expérimentation, en déconnectant bien sûr d'abord l'alimentation de cette plaque.

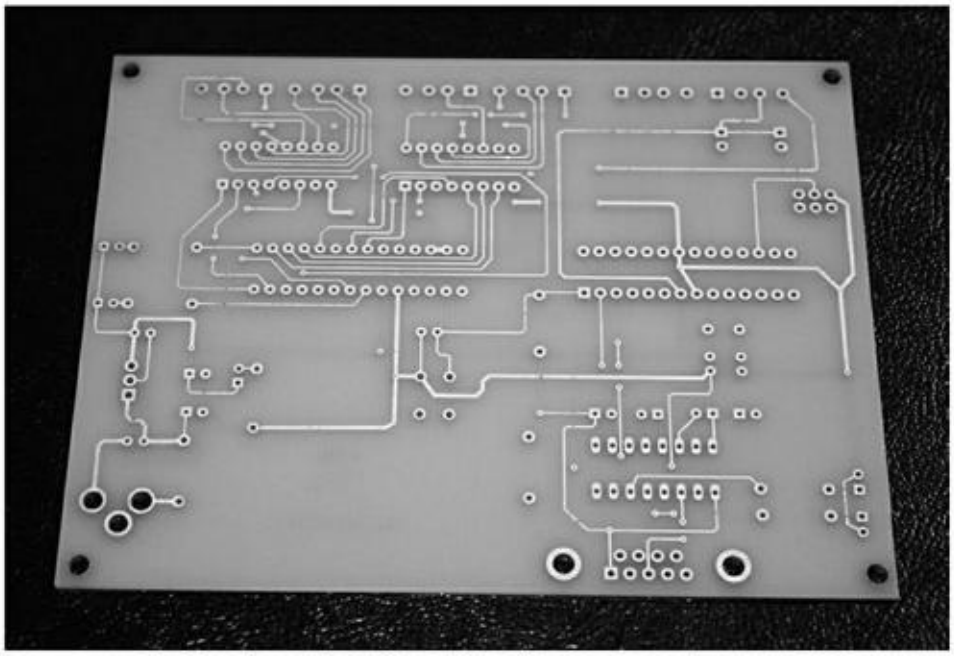


Figure 10.31 : Le circuit imprimé du Switchinator.

Nous pouvons ensuite installer le microcontrôleur U1 puis installer les circuits IC1, IC2 et IC3. Vous ajoutez les deux diodes LED 1 et 2, les résistances R1 et R2 ainsi que les six borniers. Il reste le cristal, les trois condensateurs C6, C7 et C8, les trois résistances R3, R4 et R6 et les deux diodes LED 3 et 4. Vous terminez en montant l'interrupteur de Reset S1 et le connecteur ICSP.

Le résultat doit correspondre à peu près à ce que montre la Figure 10.32. Le courant maximal qui peut être fourni par les circuits ULN2003A dépend bien sûr de leurs limites intrinsèques, mais aussi de l'ampérage maximal que peut fournir

l’alimentation. J’ai dit plus haut qu’un circuit ULN2003 pouvait fournir 300 mA par canal, soit 2,5 A par circuit intégré. Le microcontrôleur ATmega328 et le MCP23017 sont très peu gourmands. Autrement dit, c’est la consommation des deux circuits ULN qui va compter. Prévoyez une alimentation capable de fournir au moins 5 ampères si vous voulez utiliser le projet Switchinator pour contrôler par exemple une petite machine-outil numérique ou un système de diodes LED.

Test de recette

Les tests du projet sont très proches de ceux réalisés pour le prototype sauf que nous disposons maintenant des deux circuits ULN2003A et que l’interface RS-232 fonctionne avec le circuit MAX232. Bien sûr, il faut tester l’alimentation et les entrées analogiques.

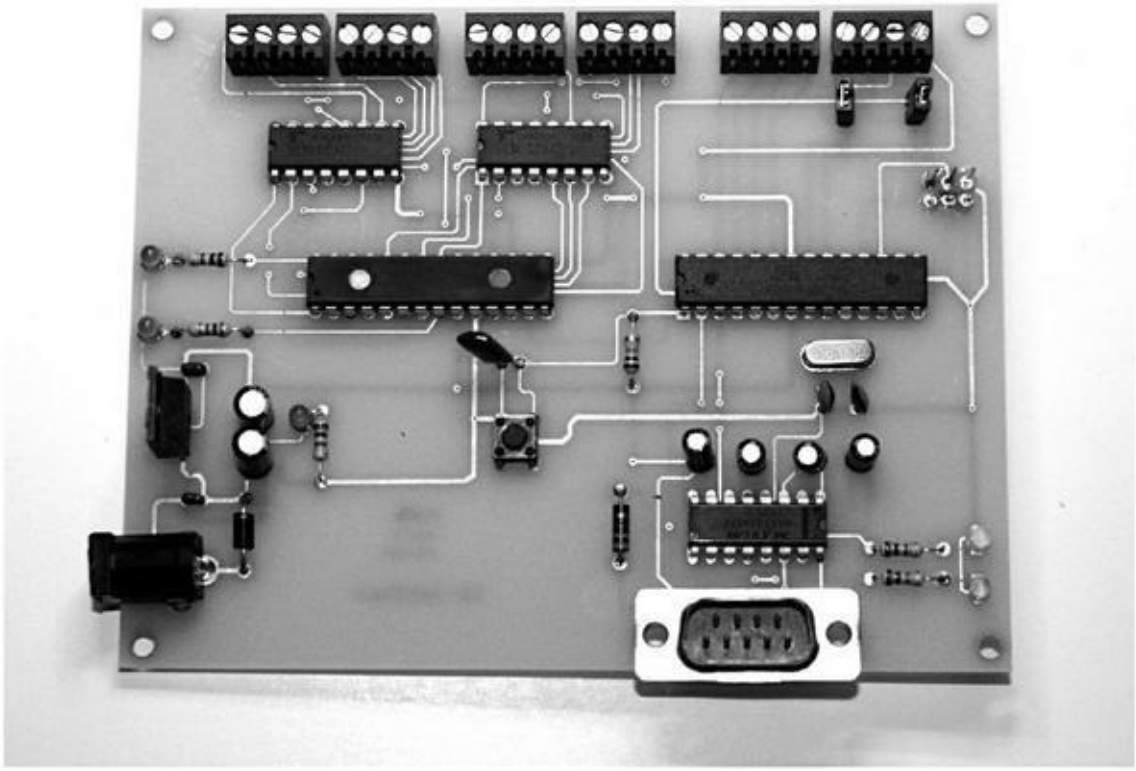


Figure 10.32 : Le projet Switchinator terminé.

Pour aller plus loin

De nombreuses broches d'entrées-sorties numériques du microcontrôleur restent disponibles, dix au total, dont six capables de fournir une modulation par impulsion PWM. Je n'ai pas rendu ces sorties disponibles sur les bords du circuit à cause des contraintes de routage, mais il est possible de modifier le dessin pour ajouter des borniers ou des connecteurs équivalents. Cela dit, il faudra sans doute bien jongler avec les traversées pour que les pistes trouvent leur chemin autour des sorties du microcontrôleur.

L'interface SPI est disponible sur les trois broches D11, D12 et D13. Elles sont reliées au connecteur ICSP. Vous pouvez donc relier un module SPI au projet si vous utilisez une broche libre pour le signal SS. Quant aux broches analogiques, je rappelle qu'elles peuvent servir de broches d'entrées-sorties numériques en les désignant sous les symboles D14 à D19.

Vous avez peut-être remarqué que je n'ai prévu aucun fusible de protection, ni sur l'alimentation, ni sur les entrées analogiques. Le Chapitre 11 donne un exemple de protection des entrées pour le générateur de signal.

Quelques ressources

Nous avons découvert bien des choses dans ce grand chapitre et vous devriez pouvoir maintenant progresser d'un pas encore plus assuré. Voici quelques ressources qui permettent de découvrir plus en détail certain des sujets qui ont à peine été abordés au long du texte.

Textes de référence

Il existe de nombreux livres abordant tous les aspects de l'électronique. Voici une brève sélection qui aborde directement les mêmes sujets que ce chapitre. Pour la liste complète, vous vous reporterez à l'Annexe D.

- Jan Axelson, *Making Printed Circuit Boards*, Tab Books.
- Paul Horowitz et Winfield Hill, *Traité de l'électronique analogique et numérique*, Elektor.

- J. M. Hughes, *Practical Electronics: Components and Techniques*, O'Reilly.
- J. M. Hughes, *Real World Instrumentation with Python*, O'Reilly.
- Simon Monk, *Fritzing for Inventors*, McGraw-Hill.
- Matthew Scarpino, *Designing Circuit Boards with EAGLE*, Prentice Hall.
- Cathleen Shamieh, L'*Électronique pour les Nuls*, 2e édition, First.

Outils de conception électronique

Les deux logiciels Fritzing et Eagle ne sont pas les seuls outils de création de schémas de principe et de routage de circuits imprimés. Ils sont néanmoins les plus utilisés, ce que vous constatez lorsque vous téléchargez des schémas ou des typons. Fritzing est gratuit et open source, et vous pourrez enrichir sa librairie de composants grâce aux contributions de nombreux utilisateurs. Son utilisation est simple et tout à fait appropriée à vos projets Arduino.

La version gratuite d'Eagle embarque certaines des fonctions que vous retrouverez dans les versions commerciales du même produit. Il permet donc de faire ses premiers pas lorsque vous savez que vous allez entrer ensuite dans le monde de la conception électronique professionnelle. Sachez que la version gratuite est cependant limitée et qu'il ne faut pas l'utiliser dans des applications commerciales. Dans ce cas, il faut acquérir une licence.

Il existe d'autres outils de conception, notamment sous Linux les outils gEDA et KiCad dont la richesse fonctionnelle n'est pas loin d'égaler celle des produits commerciaux. Voici les sites Web que vous pourrez visiter :

- http://www.geda-project.org
- http://kicad-pcb.org
- http://fritzing.org
- https://www.autodesk.com/products/eagle/overview

Fabricants de circuits imprimés

Vous trouverez assez aisément des sociétés qui peuvent se charger de réaliser vos circuits imprimés, même à l'unité. Faites des recherches sur le Web. Les animateurs du site `fritzing.org` proposent un service de fabrication de circuits auquel vous avez directement accès depuis le logiciel.

Approvisionnement en composants

Je fournis dans l'Annexe C une liste complète de fournisseurs potentiels pour vos composants, aussi bien dans le monde francophone que dans le monde anglophone et en Extrême-Orient.

CHAPITRE 11
Un générateur de signal programmable

N’importe quel atelier ou laboratoire d’électronique, que ce soit celui d’un amateur ou d’une entreprise, a besoin de sources de signaux pour ses tests. Certains se contenteront d’un simple générateur de sinusoïdes ; d’autres auront besoin d’un véritable générateur de fonctions. Le projet de ce chapitre, montré dans la Figure 11.1, est capable de générer des ondes sinusoïdales et carrées à une fréquence allant jusqu’à 40 MHz.

Figure 11.1 : Générateur de signal sur mesure.

Vous pouvez bien sûr acquérir un tel générateur dans le commerce. Les prix dépendent de la plage de fréquences et de la richesse fonctionnelle. Vous en

trouverez à partir de 20 €, et pouvez compter sur environ 50 € pour un appareil tel que celui de ce projet. Mais il existe aussi des instruments à hautes performances qui coûtent plusieurs milliers d'euros (j'ai récemment vu à la vente un générateur multifonction d'occasion montant jusqu'à 20 GHz au prix de 72 000 $).

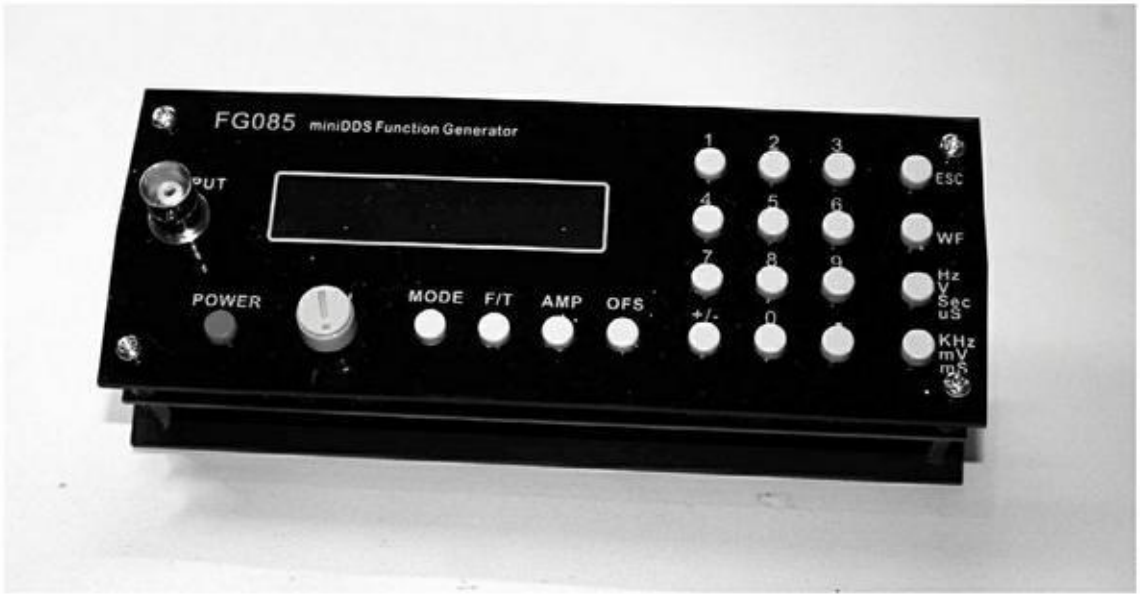


Figure 11.2 : Le générateur en kit FG085 assemblé.

Les générateurs en kit permettent de générer des ondes, mais ils n'ont souvent pas toutes les fonctions dont vous avez besoin. Après tout, ce sont les réponses aux besoins d'autres personnes, et tout le monde n'a pas la même vision de ce qui constitue un générateur de signal. En construisant le vôtre, vous aurez exactement ce que vous voulez et pourrez l'enrichir en fonction des nouveaux besoins.

Le premier critère de décision est la méthode utilisée pour générer le signal. Soit c'est le microcontrôleur qui fait tout le travail, soit il le délègue à un circuit intégré dédié à la génération des ondes. Dans le cas du kit FG085 de la Figure 11.2, c'est un microcontrôleur ATmega168 qui anime l'appareil, avec un circuit CP2101 pour l'interface USB et une génération du signal déléguée à un autre contrôleur ATmega48 avec un convertisseur analogique/numérique constitué d'un réseau de résistances. Cette approche n'est pas inintéressante, d'autant qu'elle permet au générateur de produire d'autres ondes que les sinusoïdales et les carrées, par exemple des motifs prédéfinis, ce qui est parfois bien pratique. Le principal inconvénient de l'emploi d'un microcontrôleur pour

générer les signaux est que la fréquence maximale s'en trouve très limitée : elle ne monte qu'à environ 200 kHz dans cet exemple. Ce n'est pas un défaut du kit : c'est une limite incontournable qui découle de l'utilisation d'un microcontrôleur pour la génération du signal.

En effet, la fréquence d'horloge d'un microcontrôleur est fixe, et elle impose un plafond au signal cyclique périodique pouvant être produit. Dans de nombreux domaines d'application, un plafond de 200 kHz convient, notamment dans le monde des capteurs et du contrôle industriel. Les événements ne surviennent pas à un rythme très rapide dans le monde réel, tout du moins en comparaison de la microseconde qui est l'unité habituelle des microcontrôleurs.

En revanche, si nous avons besoin de dépasser cette fréquence, il nous faut une autre approche. Justement, tous les composants permettant de construire un générateur de signal à 40 MHz pour du sinus et du carré existent déjà sous forme de modules Arduino.

Pour en savoir plus au sujet du kit FG085, visitez le site Web de JYE Tech. Notez que je ne préconise pas particulièrement ce produit. Il se trouve que j'en possède un et que je m'en sers de temps à autre. Je possède d'autres générateurs de signaux et de fonctions, dont certains assez sophistiqués. Chacun joue un rôle différent lors de mes travaux de développement et de test.

Pour être honnête, je tiens à dire dès maintenant que le générateur que je propose de construire dans ce chapitre va coûter un peu plus que les 50 € du kit FG085. Je vous propose un compte-rendu financier détaillé tout à la fin du chapitre. À vous de décider si l'investissement est justifié par le supplément de contrôle qui vous est offert au niveau de la conception et de l'utilisation. Vous pourrez également adopter ou pas le format du boîtier de protection que j'ai prévu. À chacun de résoudre son équation prix/performances/évolutivité.

Le sujet principal de ce livre est le matériel Arduino ainsi que les modules, capteurs et composants associés. Les logiciels ne sont présentés que pour mettre en valeur certains points particuliers. Ce ne sont pas des exemples complets prêts à être exécutés. Je rappelle que le code source de tous les exemples est disponible sur le site Web de l'éditeur.

Objectifs du projet

Les deux phases de définition et de planification (présentées dans le Chapitre 10) vont être réunies en une seule étape pour ce projet. Il n'est pas spécialement complexe, et cela nous permettra de progresser vite. La partie la plus délicate est le logiciel, ce qui est souvent le cas lorsque l'on travaille avec des composants modulaires et un microcontrôleur.

Notre générateur doit pouvoir servir de banc de test. Il est bien sûr possible de générer des ondes carrées ou sinusoïdales directement sur une sortie numérique d'un Arduino, ou bien de contrôler un convertisseur analogique/numérique, voire d'utiliser les sorties à largeur d'impulsion PWM pour simuler une sinusoïde. Dans tous les cas, vous êtes limité en fréquence par le microcontrôleur. Il existe une autre façon d'utiliser une carte Arduino pour obtenir un générateur de signal : il suffit de déléguer ce travail de génération à un circuit intégré dédié. Dans notre cas, ce sera l'AD9850.

Le circuit AD9850 réalise une synthèse numérique directe DDS (*Direct Digital Synthesis*) d'ondes sinusoïdales et carrées. Il peut fonctionner de zéro à 40 MHz. J'ai présenté ce module dans le Chapitre 9 (Figure 9.57). Puisque la présence de ce circuit libère le microcontrôleur, nous allons pouvoir tirer profit des cycles de processeur disponibles pour réaliser d'autres fonctions en parallèle. C'est ainsi que nous allons pouvoir mettre à jour l'affichage, détecter une tension de déclenchement sur la porte d'entrée Gate Input et surveiller les entrées de contrôle. Tout cela pourra être réalisé pendant que le circuit dédié continue à générer l'onde.

Une autre conséquence est que notre générateur n'aura pas besoin d'utiliser des interruptions pour détecter les appuis sur les interrupteurs de façade. Le circuit AD9850 fonctionne en permanence, sauf bien sûr si vous activez le mode de détection de seuil. Le microcontrôleur peut donc prendre son temps pour balayer les boutons poussoirs sans dégrader la qualité du signal de sortie.

Définition et planification

Notre projet est prévu pour être transportable d'un banc de test à un autre. Il sera donc alimenté par un adaptateur secteur. Il y a assez de place dans le boîtier pour ajouter plus tard un jeu de piles, si vous voulez vraiment le rendre autonome.

Sorties de signal :

- Sortie sinusoïdale toujours active de zéro à 40 MHz, de 0 à 1 V crête à crête
- Sortie d’onde carrée toujours active, zéro à 40 MHz, de 0 à 5 V crête à crête

Signaux d’entrée de contrôle :

- Entrée de contrôle externe pour seuil de déclenchement Gate
- Entrée pour contrôle de commande par tension VCO

Interface utilisateur :

- Afficheur LCD sur deux lignes
- Boutons poussoirs de sélection de plage et de fréquence
- Contrôle du niveau du signal de sortie
- Jack pour l’entrée de déclenchement Gate
- Prises BNC pour l’entrée de commande par tension
- Interrupteur d’alimentation

J’ai choisi de protéger le projet par un boîtier muni d’une poignée de transport, dans le style de celui de la Figure 11.3. L’adaptateur secteur devra fournir entre 9 et 12 V en courant continu.

La face avant du montage va recevoir un afficheur LCD 16 caractères sur deux lignes qui va indiquer la fréquence de sortie et l’état des deux entrées de commande par tension et de déclenchement Gate, ainsi que le niveau de la sortie sinusoïdale et de la sortie carrée. À l’arrière du boîtier, il n’y aura qu’une prise pour connecter le jack d’alimentation.

Figure 11.3 : Aspect du boîtier destiné à recevoir le projet.

Le Tableau 11.1 donne une première liste de composants qui sera bien sûr enrichie au fur et à mesure de notre progression. Ce genre de liste initiale fait souvent l'objet de retouches entre la phase de conception et la phase de livraison finale. Cela fait partie du processus de conception cyclique.

Tableau 11.1 : Liste de composants initiale.

Quantité	Description	Quantité	Description
1	Arduino Uno	3	Connecteurs BNC femelles
1	Module AD9850 DDS	2	Fiches bananes (pour entrée Gate)
1	Bouclier prototype	1	Adaptateur secteur
1	Afficheur LCD 2 lignes	1	Boîtier

Conception

Puisque nous disposons des grandes lignes du projet et d’une première liste de composants, nous pouvons plonger dans la conception détaillée. Nous allons commencer par étudier la fonction prévue afin de savoir ce que doit faire exactement notre appareil et quels sont les éléments de contrôle et les entrées-sorties correspondantes.

Nous verrons ensuite comment protéger le projet par un boîtier, ce qui aura un impact sur nos possibilités d’installation de l’afficheur et des boutons de l’interface utilisateur, sans oublier les connecteurs des entrées et sorties. Nous allons chercher un compromis entre solidité du boîtier et prix de revient contenu, sans oublier de tenir compte de la plus ou moins grande facilité d’usinage de la face avant.

Il n’y a que peu de conception à prévoir au niveau du circuit lui-même, puisque nous utilisons des modules précâblés. Nous allons surtout relier ces modules ainsi que les connecteurs les uns aux autres, ce qui supposera une petite séance de soudure, mais nous verrons cela dans la phase d’assemblage.

J’ai choisi d’utiliser des boutons poussoirs et non un sélecteur rotatif comme on en trouve dans certains générateurs (comme ceux visibles sur le site Instructables). En effet, un encodeur rotatif permet rapidement de changer de plage de valeurs, mais il ne fait en réalité qu’une chose : mesurer un angle et un sens de rotation. Les poussoirs sont bien plus versatiles, tout dépendant de la manière dont le logiciel interprète l’action qu’il détecte, en relation avec les autres boutons et le mode actuel de l’appareil.

Étude fonctionnelle

Le circuit AD9850 est doté d’un comparateur qui permet de produire en parallèle l’onde carrée et le sinus. Le sinus est fourni à une tension de 1 V crête à crête alors que le carré monte à 5 V crête à crête (*peak-to-peak)*. La fréquence est réglable en continu entre 0 et 40 MHz sous le contrôle d’un contrôleur Arduino Uno. Vous pouvez faire varier cette fréquence à partir d’une tension de commande et vous pouvez déclencher ou bloquer la sortie grâce à un signal de porte externe (Gate) qui peut être actif à l’état haut ou à l’état bas. La

Figure 11.4 propose un diagramme fonctionnel avec tous les composants présentés dans le Tableau 11.1.

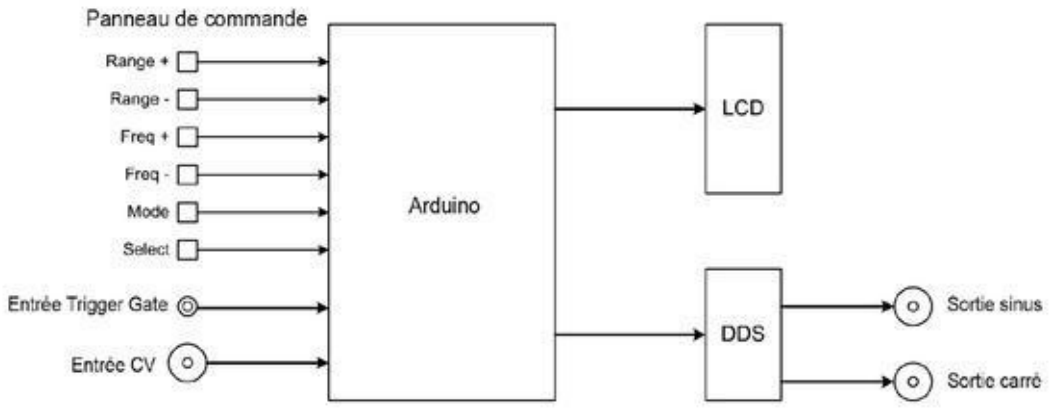


Figure 11.4 : Diagramme fonctionnel du générateur de signal.

Nous voyons trois composants dans cette figure. La carte Arduino prend en charge toutes les entrées : actions de l'utilisateur et entrées de tension CV et Gate. L'afficheur montre en permanence l'état de l'instrument et le module DDS génère les deux ondes sinusoïdale et carrée.

Je rappelle que le but d'un diagramme fonctionnel n'est pas de fournir tous les détails concernant le brochage des composants interconnectés. Il sert à comprendre comment interagissent les composants. C'est le niveau de détail idéal pour s'assurer que le but reste accessible avec les composants sélectionnés.

Dans cette Figure 11.4, j'ai placé les sorties sur la droite et les entrées sur la gauche. Le gros cercle en bas à gauche symbolise le connecteur de type BNC typique des câbles coaxiaux blindés. Le petit double cercle symbolise un connecteur de type fiche banane.

En complément de l'afficheur LCD, des boutons poussoirs et des connecteurs d'entrée, je prévois deux potentiomètres pour pouvoir régler indépendamment le signal des sorties carrée et sinusoïdale.

Boîtier du projet

Pour ce projet, j'ai choisi un boîtier IP-6130 de Bud Industries (vu en Figure 11.3). Ce boîtier dispose d'une poignée qui permet de poser l'appareil en biais pour augmenter la lisibilité. La fiche technique est disponible sur le site de

Mouser Electronics. Le boîtier coûte environ 25 euros, ce que je trouve raisonnable vu la durabilité et l'aspect professionnel. La Figure 11.5 indique les dimensions extérieures. Vous consulterez la fiche pour tous autres détails.

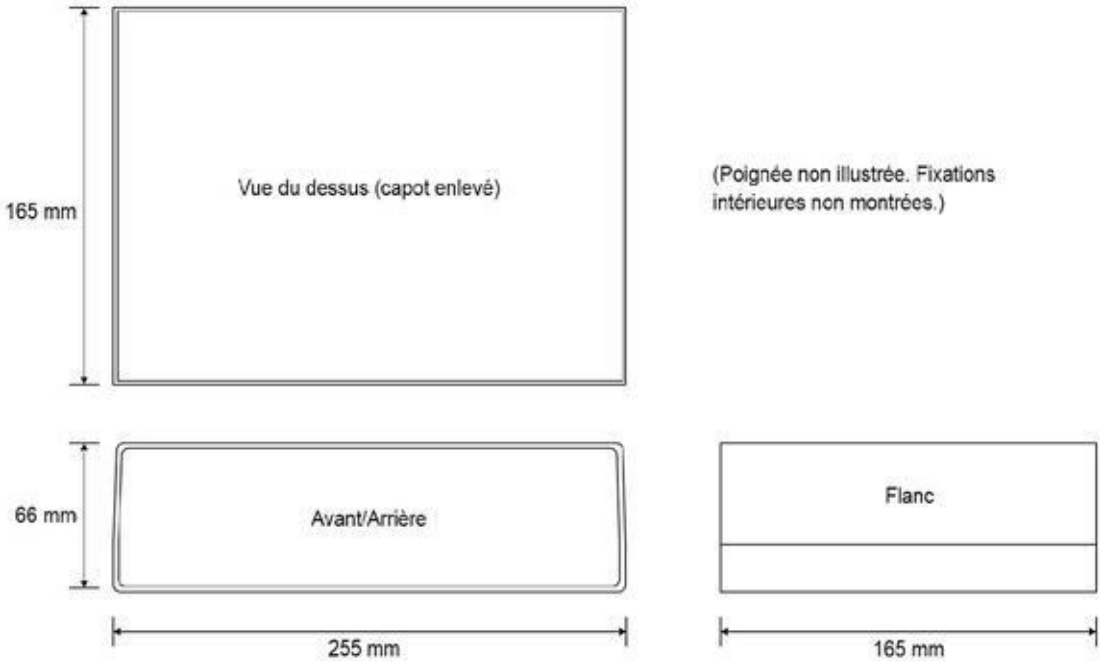


Figure 11.5 : Dimensions du boîtier prévu pour le projet.

Si vous optez pour le même modèle, tenez compte du fait que les faces avant et arrière ne sont pas de vrais rectangles, mais des trapèzes. Le bord supérieur est légèrement moins large que le bord inférieur. Sachez-le pour ne pas faire ensuite les perçages de la face avant à l'envers. Les vis qui réunissent la partie inférieure et le capot sont normalement recouvertes par des boutons adhésifs en caoutchouc par le dessous.

Notez également que le boîtier est assez plat, ce qui ne permet pas d'empiler plusieurs cartes bouclier au-dessus de la carte Arduino. Il existe un modèle plus haut, IP-6131, qui offre presque 9 cm en hauteur, mais il y aurait beaucoup de place perdue à l'intérieur. J'ai donc choisi de placer la carte Arduino et la carte du module DDS côte à côte sur le fond du boîtier, en montant bien sûr l'afficheur, les interrupteurs et les connecteurs sur la face avant et le jack d'alimentation sur la face arrière. Il y a largement assez de place à l'intérieur pour ajouter plus tard un porte-piles ou une alimentation interne. La Figure 11.6 montre l'aspect désiré pour la face avant.

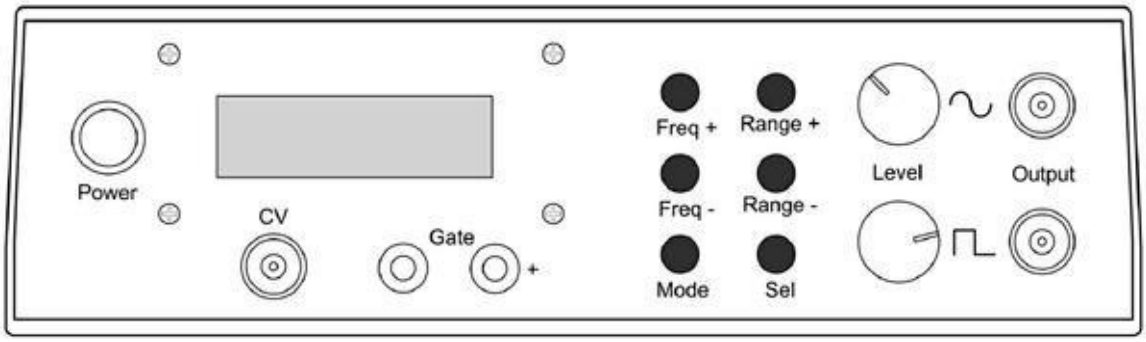


Figure 11.6 : Dessin de la face avant du générateur de signal.

Un dernier point au sujet du boîtier concerne les boutons en face avant. J'aurais pu envisager un module clavier, mais il aurait fallu que j'en conçoive un moi-même avec trois fois trois boutons. Dans la mesure où je ne compte réaliser qu'un seul exemplaire du projet, cela n'en vaut pas la peine. Des boutons poussoirs indépendants feront largement l'affaire, même si l'aspect ne sera pas identique à ce que montre la figure précédente. Le fonctionnement n'en sera pas modifié.

Schéma de principe

Si nous revenons à la Figure 11.4, on pourrait en déduire que toutes les broches de sortie de la carte Arduino sont utilisées. C'est presque vrai, puisque seules les deux broches analogiques A4 et A5 sont libres. Nous utilisons toutes les broches d'entrée-sortie numériques pour l'afficheur LCD, le module DDS et les interrupteurs. Certaines des entrées analogiques sont réutilisées en mode numérique, comme le montre le schéma de la Figure 11.7.

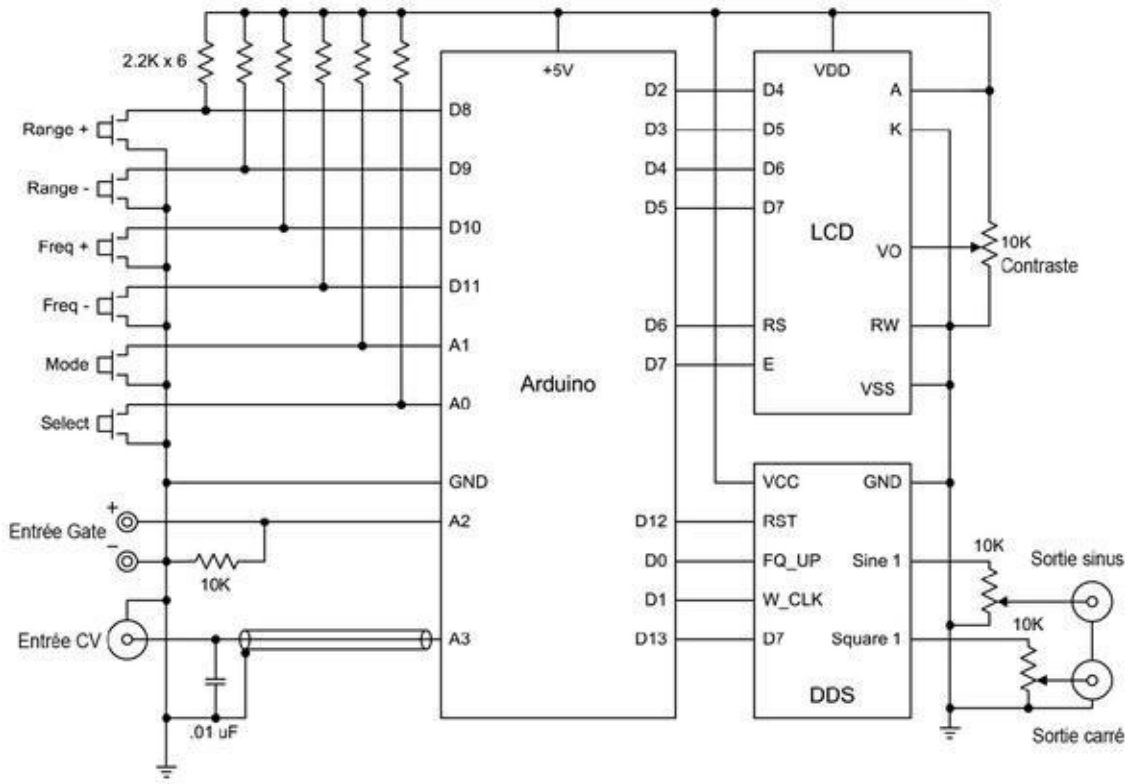


Figure 11.7 : Schéma de principe du générateur de signal.

Le Tableau 11.2 dresse la liste des affectations de broches du microcontrôleur. Les deux broches libres A4 et A5 restent disponibles pour une extension avec un autre bouclier en utilisant l'interface I2C.

Tableau 11.2 : Affectation des broches Arduino du projet.

Broche	Fonction	Broche	Fonction	Broche	Fonction
D0	DDSFQ_UP	D7	LCDE	A0	Bouton Select
D1	DDSW_CLK	D8	Range +	A1	Bouton Mode
D2	LCDD4	D9	Range -	A2	Gate input
D3	LCDD5	D10	Freq +	A3	CV input
D4	LCDD6	D11	Freq -	A4	SDA vers I/O
D5	LCDD7	D12	DDSRST	A5	SCL vers I/O

D6	LCDRS	D13	DDSD7

Nous avons déjà présenté un module DDS dans la Figure 9.57. Vous devriez pouvoir le trouver pour environ 10 € chez votre fournisseur habituel. Le brochage du module est présenté par la Figure 11.8.

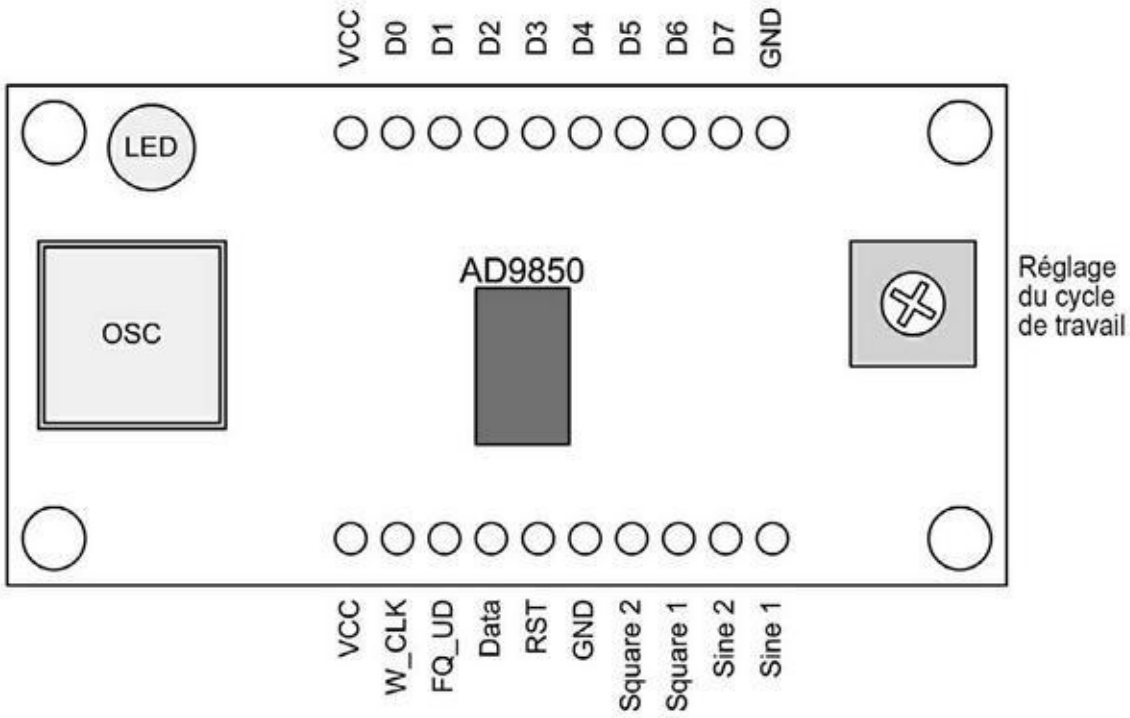


Figure 11.8 : Brochage des sorties du module DDS AD9850.

Le circuit A98950 sait travailler en parallèle mais également en série, mot par mot. Nous optons pour le mode série qui ne réclame qu'une seule broche du module DDS, la broche D7.

L'afficheur LCD comporte 16 broches pour relier la masse, l'alimentation, les données et les broches de contrôle. Les liaisons se font avec le circuit qui pilote l'afficheur et qui se trouve à l'arrière du circuit imprimé, noyé dans de la résine noire. Le brochage de l'afficheur LCD est montré dans la Figure 11.9.

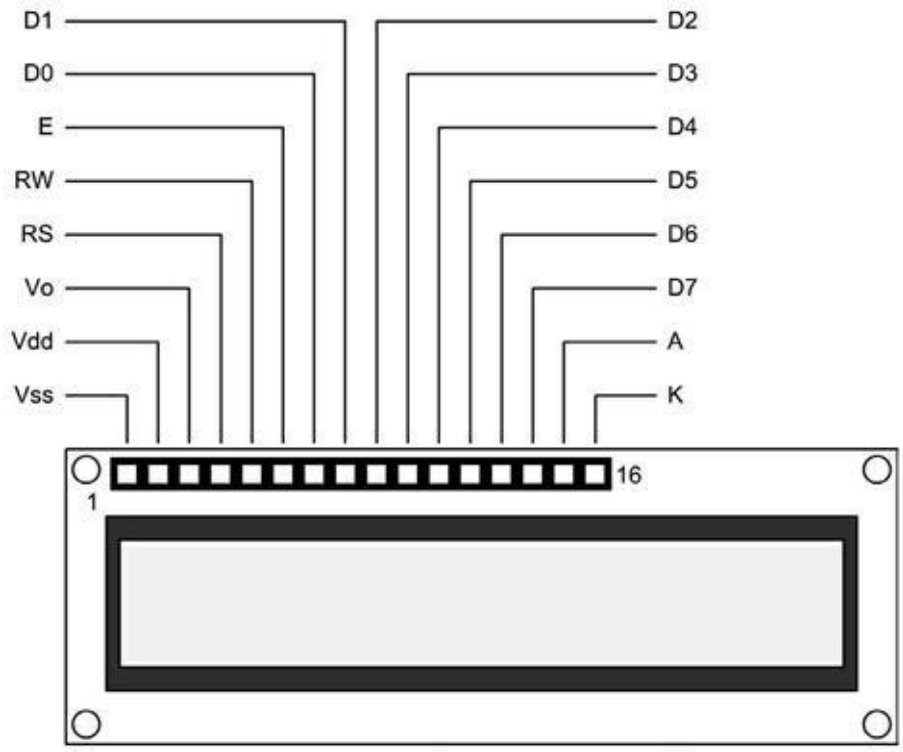


Figure 11.9 : Brochage du module afficheur LCD.

Création du prototype

Ce projet utilise des composants standard du commerce. Si possible, il vaut mieux créer le prototype à partir des mêmes éléments afin de pouvoir progresser dans l'écriture du logiciel. Si vous devez attendre la livraison de certains composants, vous pouvez néanmoins créer un prototype quasiment identique à la version définitive. Pour ce projet, j'ai ainsi installé une carte Arduino Uno sur une plaque de bois (Figure 11.10).

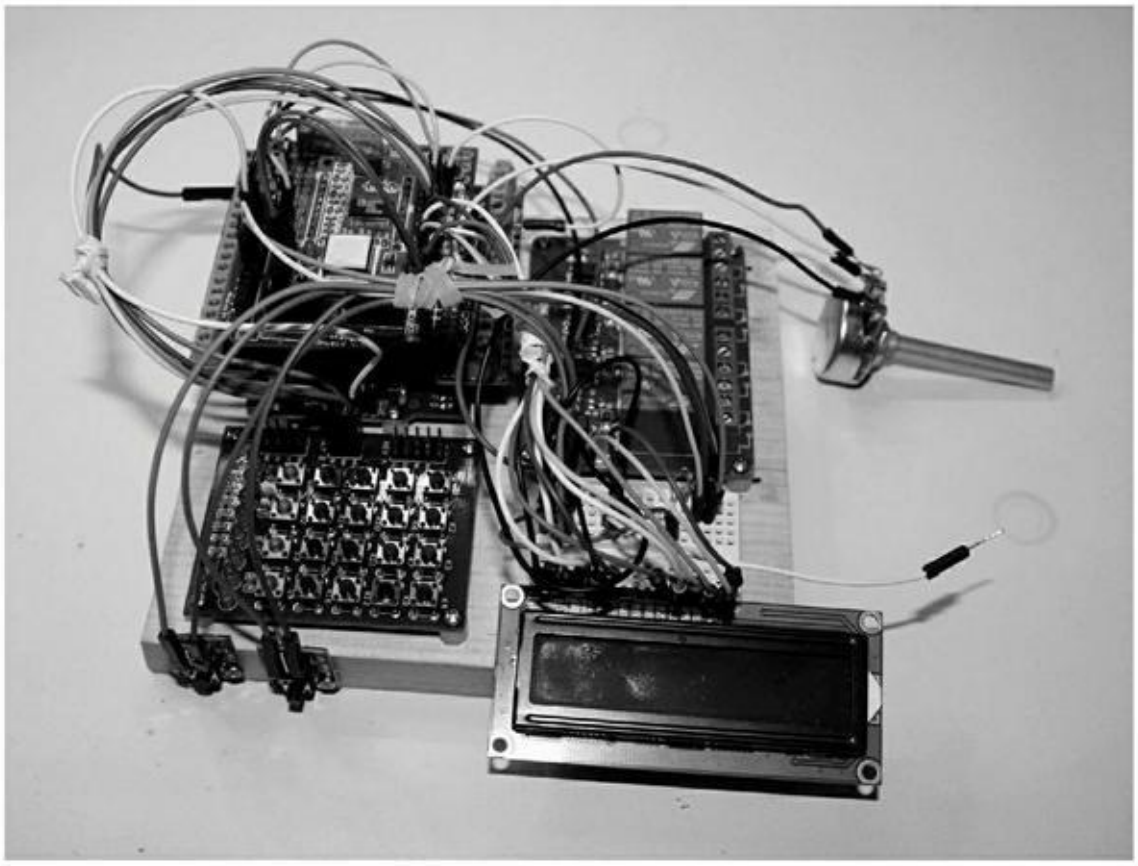


Figure 11.10 : Montage volant du prototype du projet.

J’ai donc utilisé pour le prototype un circuit imprimé de récupération qui était doté d’une matrice de cinq sur quatre petits boutons poussoirs ainsi que de huit diodes LED. J’ai besoin de deux entretoises, d’une carte bouclier de prototypage sur laquelle j’ai monté le module DDS et l’afficheur LCD qui servira dans la version finale. Les boutons poussoirs permettront de tester l’interface utilisateur. Vous pouvez voir en haut à droite de la Figure 11.10 les quatre pavés des relais que je n’utilise pas ici ; ils ne sont donc pas connectés. Toutes les autres connexions sont en conformité avec ce que montre le schéma de principe de la Figure 11.7.

Boutons de contrôle et modes

La face avant offre six boutons dont l’affectation est indiquée dans le Tableau 11.3. Notez que la mention CV signifie *Control Voltage*, soit tension de commande.

Tableau 11.3 : Affectation des boutons du générateur de signal.

Bouton	Mode fréquence	Mode Gate	Mode CV
Freq +	Fréquence +	Gate on	CV actif
Freq –	Fréquence -	Gate off	CV inactif
Range +	Plage +	Gate +	CV zéro set
Range –	Plage -	Gate –	CV zéro reset

Ce générateur fonctionne selon trois modes de saisie. Le tableau précédent permet de conclure qu'il y a 12 éléments de contrôle possibles, en fonction du mode. Le quatrième bouton, Select, n'est pas mentionné, car il sert de touche de validation (entrée). Lorsque vous utilisez le bouton Select, vous sortez du mode de saisie Gate ou CV pour revenir dans le mode normal de sélection de fréquence et de plage.

J'ai déjà dit que la génération des ondes en sortie était toujours active, sauf lorsque le mode Gate inhibe la sortie. La fréquence et la plage peuvent être modifiées à tout moment, sauf dans le mode Gate. Les deux boutons Freq+ et Freq- font varier la fréquence de un hertz. Si vous maintenez la pression sur le bouton, le changement se fait par 10, par 100 et même par 1000 en fonction de la plage de valeurs en cours.

La sélection de fréquence se fait plage par plage, chaque plage s'étendant sur 10 000 hertz. La première plage va de zéro à 9 999 Hz, la plage 2 de 10 000 à 19 999 Hz, et ainsi de suite. Les deux boutons de changement de plage permettent d'aller plus vite à une plage très éloignée. Les boutons de réglage de fréquence ne provoquent pas de rebouclage : lorsque vous arrivez au plafond de la plage, vous entrez automatiquement dans la plage suivante, et de même lorsque vous arrivez à la limite inférieure d'une plage. La fréquence est mise à jour automatiquement. Il n'y a pas besoin de valider la nouvelle fréquence avec le bouton Select.

Votre plage peut donc avoir une valeur entre 1 et 4 000. Comme pour les réglages de fréquence, la sélection de plage peut faire des sauts de 10, de 100 ou de 1 000 si vous maintenez enfoncé le bouton correspondant.

Pour configurer les deux modes CV et Gate, vous devez d'abord entrer dans le mode de saisie correspondant. Le bouton Mode est destiné à cette sélection. Avec le bouton Select, vous activez le paramétrage en cours, et rebasculez l'appareil en mode normal.

Lorsque le seuil de déclenchement Gate est actif, le générateur ne produit rien en sortie tant que la condition associée n'est pas détectée sur l'entrée. Les deux boutons Freq+ et - servent à activer et désactiver le seuil Gate. Les deux boutons Range+ et - servent à choisir l'état d'activation, soit à l'état haut, soit à l'état bas.

L'entrée de tension de commande CV accepte une tension analogique entre 0 et 5 V continus. La tension de 2,5 V correspond au 0. Nous n'utilisons pas de tension négative. Dans ce mode CV, les deux boutons Freq+ et - servent à activer et désactiver l'entrée CV. Pour décaler le point zéro, vous entrez dans le mode CV puis utilisez le bouton Range+. Le bouton Range- ramène le point zéro à la valeur par défaut de 2,5 V.

Une fois que l'entrée CV a été activée, elle le reste jusqu'à ce que vous la désactiviez. Lorsqu'une tension supérieure au point zéro est détectée, cela provoque une augmentation de la fréquence en sortie, et inversement avec une tension inférieure à celle du point zéro. Pour redéfinir le point zéro, il suffit d'appliquer la tension appropriée à l'entrée CV puis d'appuyer sur le bouton Range+. Le fait de quitter le mode CV ne fait pas perdre la configuration du point zéro.

Affichage

L'afficheur choisi pour ce projet n'offre que deux lignes de 16 caractères, ce qui peut constituer un petit défi pour réussir à afficher toutes les informations utiles. La Figure 11.11 montre la solution que je propose pour faire tenir toutes les informations essentielles sur cet afficheur.

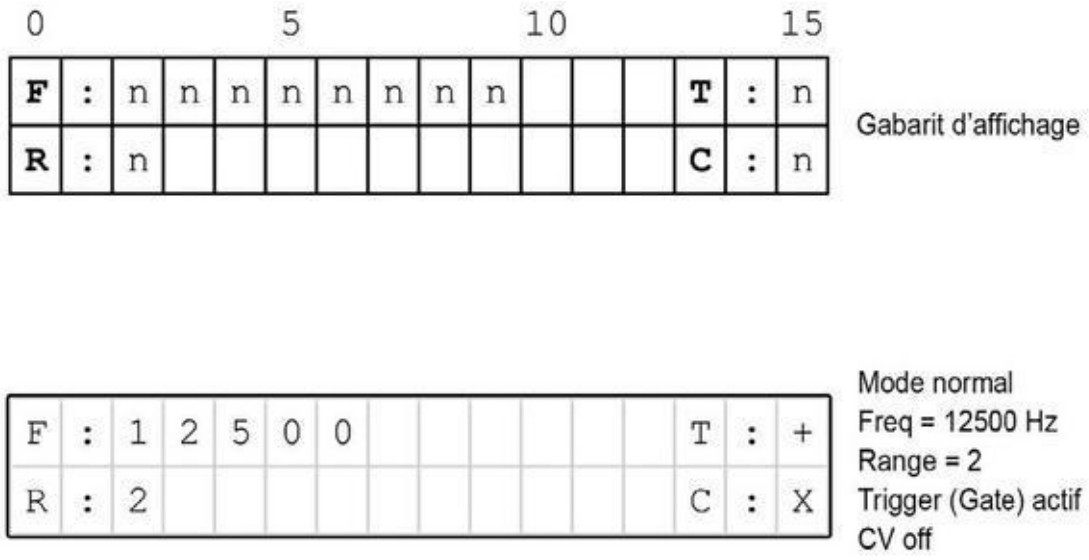


Figure 11.11 : Modèle et exemple d'utilisation de l'afficheur du projet.

Vous savez que le générateur n'est pas en mode normal, mais dans un des deux autres modes lorsque le signe qui suit la lettre du mode du côté droit clignote. En mode CV, c'est le C qui clignote.

Le bouton Mode sert à passer d'un mode à l'autre. Vous ne pouvez changer les paramètres des modes CV et Gate que lorsque le mode correspondant est activé. Vous devez utiliser le bouton Select pour confirmer le reparamétrage et rebasculer en mode normal de sélection de fréquence.

Le coin supérieur droit de l'affichage est dédié au mode Gate qui peut afficher soit un X, soit un signe +, soit un signe -. Le + rappelle que la fonction est active à l'état haut et le - qu'elle l'est à l'état bas. Le X signifie que la fonction est désactivée. Je rappelle qu'il faut utiliser le bouton Select pour valider l'activation.

Dans le coin supérieur droit, nous trouvons le champ du mode CV. La lettre est suivie d'un X, d'un zéro, d'un plus ou d'un moins. Le signe + indique que la fonction est active et que la tension d'entrée est supérieure au point zéro. Le signe - signifie que la tension est inférieure au point zéro. La valeur nulle signifie qu'elle est égale. Le symbole est mis à jour en temps réel, bien sûr si la fonction d'entrée CV est activée. Elle ne l'est pas lorsque le champ indique X. Comme pour le mode Gate, il faut utiliser le bouton Select pour valider l'activation.

Cette interface utilisateur vous rappelle peut-être le bon vieux temps des afficheurs minimalistes. Avant que les écrans LCD et TFT à haute résolution deviennent abordables, la plupart des interfaces des appareils se présentaient ainsi, et ne pouvaient être utilisées sans le manuel. Vous pouviez déjà vous estimer heureux qu'il y ait un affichage. Dans les premiers temps, on ne pouvait afficher que des chiffres, et certains étaient devenus maîtres dans l'interprétation de ces affichages. Dorénavant, nous pouvons adopter des composants LCD alphanumériques très peu coûteux. Il reste néanmoins nécessaire de montrer un minimum de créativité quant à la disposition des différents blocs de données dans la surface disponible, au prix parfois d'un côté un peu obscur.

Le module DDS

J'ai choisi de monter le module DDS AD9850 sur une carte prototype (Figure 11.12), mais ce n'est pas obligatoire. Vous pouvez tout à fait monter directement le module sur le fond du boîtier puis le connecter. Je conseillerais l'utilisation d'une carte mère comme ici, parce que cela stabilise le module et permet d'utiliser des borniers pour les connexions, ce qui est bien plus confortable que de devoir faire des soudures, et beaucoup plus fiable que des fiches dans des connecteurs.

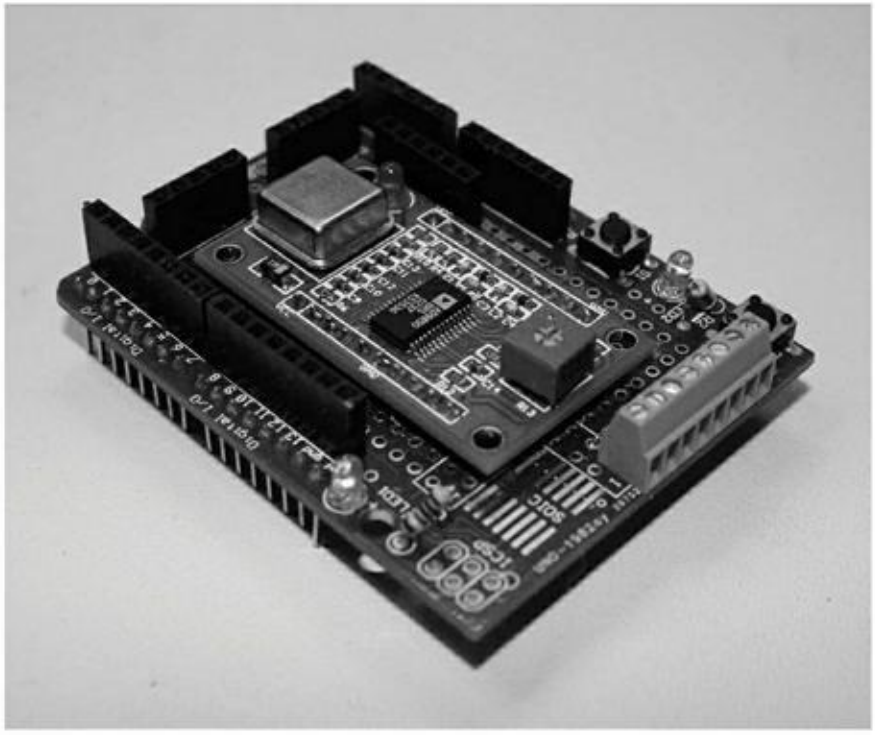


Figure 11.12 : Module DDS monté sur une carte bouclier prototype.

Le Tableau 11.4 donne la liste des composants du module DDS sur son bouclier.

Tableau 11.4 : Liste des composants du module DDS sur prototype.

Quantité	Description
1	Carte prototype, Adafruit #51 ou équivalente
1	Bornier à vis à huit plots, écartement de 2,54 mm
1	Connecteur femelle à six contacts
1	Module DDSAD9850

Le module DDS est connecté au connecteur à six contacts pour les entrées. Les sorties du module sont reliées au bornier à huit plots. L'alimentation, masse et positif, est fournie par la carte mère prototype. Dans l'exemple, je n'utilise pas les diodes LED, mais vous pouvez en relier une au positif et l'autre à la broche d'entrée D7 du module, celle qui est reliée à la broche D13 du microcontrôleur. La Figure 11.13 montre comment sont rendus disponibles les différents signaux du module DDS sur la carte prototype.

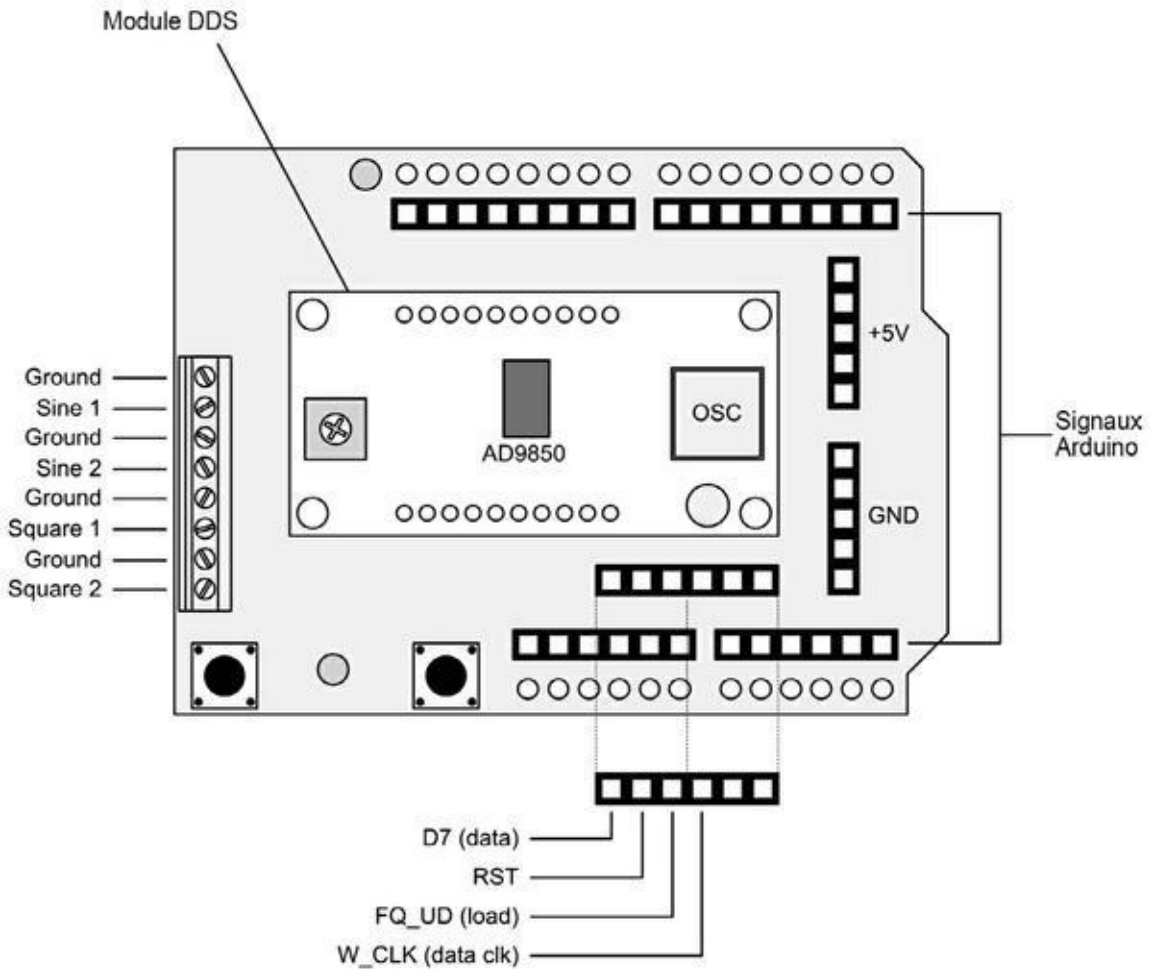


Figure 11.13 : Signaux d'entrée-sortie du module DDS sur le prototype.

Les connexions vers le bornier à vis et vers le connecteur à six contacts sont réalisées par le dessous de la carte prototype avec du fil de diamètre AWG28 (vous pouvez utiliser du fil entre 24 et 32). Je n'ai pas jugé utile d'utiliser du câble blindé pour les liaisons entre le module et le bornier, mais cela pourrait s'avérer nécessaire si vous travaillez quasiment toujours à haute fréquence.

J'aurais préféré ne pas utiliser de connecteurs standard pour les entrées de contrôle du module DDS, mais il n'y avait plus beaucoup de place disponible sur la carte prototype. Dans la version finale, je vais m'assurer, en ajoutant un point de colle, qu'aucune de ces connexions ne puisse se défaire.

Le logiciel

Comme tout programme Arduino, le logiciel de notre générateur de signal comporte une fonction `setup()` et une fonction `loop()`. La première configure les modes des broches en entrée et en sortie et réalise quelques autres opérations de configuration. La fonction principale se charge de détecter les entrées et de gérer le signal en sortie. Comme dans le chapitre précédent, le logiciel est distribué en plusieurs modules, avec plusieurs fichiers d'en-tête. Il y a six fichiers de code source.

La partie du logiciel qui contrôle le module DDS se contente de lire les deux entrées Gate et CV puis de décrire les données de contrôle correspondantes vers le circuit DDS. La partie intéressante est celle qui consiste à mettre en relation les événements détectés sur les boutons et les actions à déclencher et à quel moment. Comme dans tout projet de microcontrôleur, c'est le logiciel qui produit les fonctions désirées. Sans logiciel, le tout n'est qu'un tas de plastique et de fils, avec un peu de silicium.

Structure du code source

J'ai dit plus haut que le code source était réparti dans plusieurs fichiers. Dans l'atelier Arduino IDE, vous demandez le chargement du fichier principal qui porte le nom *SignalGen.ino*. Cela provoque le chargement dans d'autres pages de tous les fichiers référencés et trouvés dans le même répertoire. Le début du fichier principal et les onglets sont visibles dans la Figure 11.14.

Suite à ce modèle modulaire, les actions, les variables globales sont toutes réunies dans un module distinct portant le nom *sig_gen_gv.cpp*. Les fonctions de gestion de l'afficheur LCD, des entrées, de la tension de commande et de la fonction Gate sont définies dans d'autres modules. Le Tableau 11.5 dresse la liste des modules source avec leurs contenus.

Figure 11.14 : L’atelier Arduino IDE après chargement du croquis SignalGen.ino.

Tableau 11.5 : Modules de code source du générateur de signal.

Module	Fonction
SignalGen.ino	Module principal avec setup() et loop()
sig_gen.h	Définitions de constante (#define)
sig_gen_control.cpp	Gestion des boutons
sig_gen_control.h	Directives #include
sig_gen_cv.cpp	Gestion de l’entrée CV
sig_gen_cv.h	Directives #include
sig_gen_gate.cpp	Gestion de l’entrée Gate
sig_gen_gate.h	Directives #include

sig_gen_gv.cpp	Variables globales
sig_gen_gv.h	Directives #include
sig_gen_lcd.cpp	Gestion de l’afficheur LCD
sig_gen_lcd.h	Directives #include

De la complexité du dialogue avec les humains

Ce projet est assez simple, et pourtant le logiciel atteint un niveau de complexité étonnant. Pourquoi cela ? À cause de l’interface utilisateur ! Tous ceux qui ont à créer des programmes de gestion d’une interface utilisateur connaissent bien cela. Souvent, le code source qui se charge de gérer l’interface occupe jusqu’à 70 % du total. C’est une des raisons du succès des outils sur ligne de commande qui restent la règle sous Linux et Unix : ces commandes sont très puissantes tout en restant très compactes. En effet, il n’y a pas d’interface utilisateur complexe à gérer, seulement une interface sur ligne de commande en mode texte.

Les applications pour microcontrôleurs connaissent aussi cette lourdeur d’interface, incontournable à partir du moment où le système doit interagir avec un humain. En effet, les humains sont lents, ils ont peu de mémoire, ils rechignent à utiliser des commandes qu’il faut apprendre et détestent décrypter des réponses compactes. Voilà pourquoi une bonne partie du logiciel d’un microcontrôleur doit être consacrée à l’animation d’une interface utilisateur permettant à un humain normal d’interagir avec l’appareil.

Certains projets choisissent de faire exécuter l’interface utilisateur sur une machine hôte qui n’a pas les mêmes contraintes en termes d’espace mémoire et de vitesse de traitement. Cette machine peut alors dialoguer de façon beaucoup plus efficace avec la carte du microcontrôleur en lui transmettant des commandes, des paramètres et des données. J’aurais pu choisir cette approche pour ce projet, mais elle a l’inconvénient de rendre le projet non portable, puisqu’il aura en permanence un fil à la patte pour dialoguer avec la machine qui héberge l’interface utilisateur. Cela n’est pas un souci dans certaines applications, par exemple pour les appareils de contrôle qui sont installés définitivement dans des endroits choisis et interagissent avec une machine centralisée. Le générateur de signal que nous construisons ici est destiné à être

transportable. Il se doit donc d'offrir par lui-même une interface utilisateur exploitable.

Dans cette situation, c'est un défi que d'offrir la meilleure richesse fonctionnelle possible à partir d'un tout petit nombre de moyens de dialogue et des contraintes fortes en termes de surface d'affichage. Tout cela doit être animé sans perturber l'opération essentielle de l'instrument, ni consommer un espace tel dans la mémoire flash que le logiciel ne peut plus être transféré vers le microcontrôleur.

Vous pouvez voir que deux objets sont instanciés à partir des deux classes LCD et DDS au moyen de l'opérateur `new` du langage C++, dans le fichier *sig_gen_gv.cpp*. La création de ces objets à cet endroit est possible parce que les variables qui vont incarner ces objets, `lcd` et `ddsdev`, sont définies dans le module des variables globales puis exportées dans *sig_gen_gv.h*. Le module principal doit contenir les directives `#include` pour les deux fichiers *LiquidCrystal.h* et *DDS.h*. Ces deux directives doivent également se trouver au début du fichier des variables globales. Cette double déclaration est nécessaire à cause de la manière dont l'atelier Arduino IDE gère les portées des variables et les directives #include. Toute référence à une librairie externe qui se trouve dans un module secondaire doit être répétée dans le module principal. Vous remarquez aussi que les modules secondaires contiennent tous la directive `#include "Arduino.h"` pour que chacun d'eux puisse accéder à l'environnement Arduino. J'ai présenté l'opérateur `new` dans un encadré du Chapitre 5.

Description fonctionnelle du logiciel

Le code source de ce projet étant assez long, je vais m'intéresser aux sections principales et à quelques points remarquables avec des extraits de code et des diagrammes. Comme d'habitude, je vous conseille de récupérer la totalité du code source sur le site Web de l'éditeur.

Voyons d'abord le fichier d'en-tête *sig_gen.h*. Il contient toute une série de définitions de constantes par `#define`. Rappelons que cette directive permet de symboliser des valeurs littérales, ce qui évite d'avoir plusieurs retouches à faire lorsqu'une valeur doit être modifiée et donc d'oublier de modifier l'une des occurrences, ce qui provoque un fonctionnement étrange et difficile à réparer. Vous pourriez également utiliser l'instruction de déclaration de constante `const` du langage C++, mais comme déjà expliqué dans le Chapitre 5, les directives `#define` consomment moins d'espace mémoire.

Les déclarations des variables globales correspondent au fichier *sig_gen_gv.cpp*, présenté dans le Listing 11.1. Puisque c'est un fichier au format cpp, il contient aussi des instructions d'initialisation.

Listing 11.1 : Variables globales du générateur.

```
// sig_gen_gv.cpp

#include "Arduino.h"
#include <LiquidCrystal.h>
#include <DDS.h>
#include "sig_gen.h"

// Instancie les objets LCD et DDS, affecte les broches
numériques
LiquidCrystal *lcd = new LiquidCrystal(LCD_RS, LCD_E,
LCD_D4, LCD_D5, LCD_
  D6, LCD_D7);
DDS *ddsdev = new DDS(DDS_OSC, DDS_DATA, DDS_FQ_UP,
DDS_W_CLK, DDS_RESET);

// Autre technique de création d'objet
// LiquidCrystal lcdobj(LCD_RS, LCD_E, LCD_D4, LCD_D5,
LCD_D6, LCD_D7);
// DDS ddsobj(DDS_OSC, DDS_DATA, DDS_FQ_UP, DDS_W_CLK,
DDS_RESET);
// LiquidCrystal *lcd = &lcdobj;
// DDS *ddsdev = &ddsobj;

// Valeurs initiales des boutons (0 ou 1)
int fpls = 0; // freq +
int fmn  = 0; // freq -
int rpls = 0; // range +
int rmn  = 0; // range -
int mode = 0; // mode
int sel  = 0; // select


// Dernier état de bouton
int last_fpls = 0;
int last_fmn  = 0;
int last_rpls = 0;
```

```
int last_rmn  = 0;

// Variables de contrôle de la logique
int mode_cnt  = 2;     // Compteur de position dans le
cycle des modes
bool new_mode = false; // True si mode changé
int btncnt    = 0;     // Pour détecter appui long
bool btnhold  = false; // True si bouton maintenu enfoncé
int dds_load_cnt = 0;  // MAJ compteur délai de cycle
pour DDS

// Contrôle de sortie
unsigned long fval = 1000;
unsigned int rval  = 1;

// Affichages
char gval = ‘X’;
char gpol = ‘+’;
char cval = ‘X’;
char cpol = ‘0’;

// Contrôle des actions dans la boucle
bool dogate = false;
bool docv   = false;
bool gate_output = true;

// Point zéro du mode CV
bool set_cvzero   = false;
bool reset_cvzero = false;
unsigned long cv_zero  = 512;
unsigned long cv_input = 0;    // Dernière valeur lue en
entrée CV
int currmode      = MODE_DISP; // Init début du mode
entrée

// Entrée tension CV
unsigned int cvinval;
unsigned int cvzero;
unsigned int fvalset;
```

Observez la section qui se charge de créer les deux objets pour l’afficheur et le module DDS. J’ai laissé en commentaire l’autre technique de création. Celle que

j’utilise exploite l’instruction `new` alors que celle mise en commentaire utilise des affectations de pointeurs. Le résultat fonctionnel est le même, mais il y a une légère différence en termes d’empreinte mémoire.

Les déclarations d’export sont réunies dans le fichier d’en-tête *sig_gen_gv.h*. Je rappelle que la directive `export` prévient le compilateur que les variables mentionnées doivent pouvoir être utilisées par d’autres modules. En réponse, le compilateur créait des espaces d’accueil pour passer les références aux variables qui sont déclarées dans ce fichier d’en-tête .h. C’est lors de la phase d’édition des liens (*link*) que toutes les références sont résolues pour obtenir un fichier exécutable unique.

Les sorties du générateur sont gérées à travers une série de variables globales pour la fréquence choisie, la plage de fréquences et l’état des deux modes Gate et CV. Cette façon d’agir est habituelle dans les systèmes embarqués, car il y a souvent peu d’espace mémoire, notamment dans la section de la pile des appels. Grâce aux variables locales, nous évitons de devoir transmettre plusieurs arguments lors des appels de fonction. Ces variables constituent donc une sorte d’espace mémoire partagé. Pour utiliser efficacement cette technique, il faut le plus possible exploiter les variables de sorte qu’un seul processus les écrive pendant que tous les autres les lisent. Dans un petit projet qui ne comporte qu’un seul programme comme ici, il y a peu de chances que deux processus veuillent modifier la même valeur simultanément. Dans un programme multi-exétrons (*multi-threads*), c’est au contraire un risque important.

Puisque nous utilisons les deux fonctions obligatoires `setup()` et `loop()` typiques de l’atelier Arduino IDE, nous faisons commencer l’exécution par `setup()`, selon le diagramme fonctionnel de la Figure 11.15.

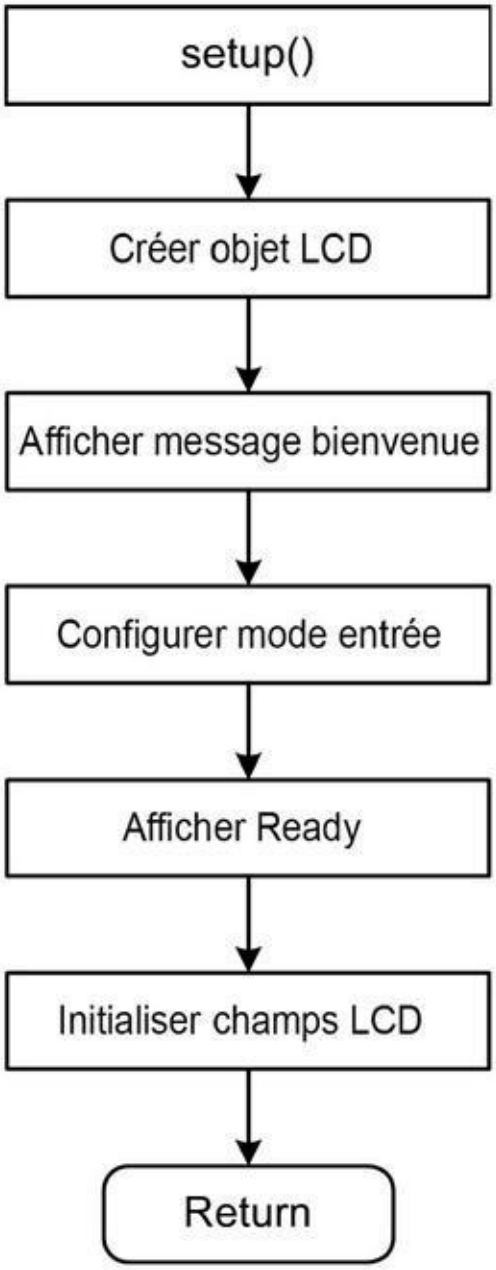


Figure 11.15 : Diagramme fonctionnel de `setup()`.

Cette fonction (Listing 11.2) est très simple : elle prépare l'objet LCD, affiche un message de bienvenue, configure les entrées numériques pour les boutons, affiche le message Ready puis affiche la fréquence, la plage, le mode Gate et le mode CV. Ces éléments restent en place tant que le générateur est sous tension.

Je rajoute un peu de délai dans les deux fonctions `TitleDisp1()` et `TitleDisp2()` pour laisser le temps de lire le texte.

Listing 11.2 : La fonction setup() du générateur de signal.

```
void setup() {
  lcd->begin(16, 2);   // Format d'affichage

  TitleDisp1();

  pinMode(SW_FPLS, INPUT);
  pinMode(SW_FMIN, INPUT);
  pinMode(SW_RPLS, INPUT);
  pinMode(SW_RMIN, INPUT);
  pinMode(SW_MODE, INPUT);
  pinMode(SW_SLCT, INPUT);

  TitleDisp2();

  InitLCDFields();
}
```

La fonction de préparation de l'afficheur, `InitLCDFields()`, appelée en dernier ci-dessus, est déclarée dans le fichier *sig_gen_lcd.cpp*. Elle écrit les zones fixes de l'afficheur en fin de phase de préparation. Vous pouvez la rappeler à d'autres endroits du code si nécessaire.

La boucle principale du projet passe par quatre étapes successives :

1. Balayage de l'état des boutons poussoirs.
2. Test de l'entrée CV, si le mode est actif.
3. Test de l'entrée Gate, si le mode est actif.
4. Mise à jour de la fréquence à générer.

À l'étape 1, nous analysons une éventuelle chaîne de commande provenant de la machine hôte et scrutons l'état des boutons poussoirs. C'est la partie la plus complexe du logiciel. Aux étapes 2 et 3, nous vérifions si quelque chose doit être modifié à l'état de ce qui est généré en sortie. À l'étape 4, nous écrivons les données vers le module AD9850 après avoir fait quelques calculs. La Figure 11.16 donne un diagramme fonctionnel à au niveau de la fonction `loop()`.

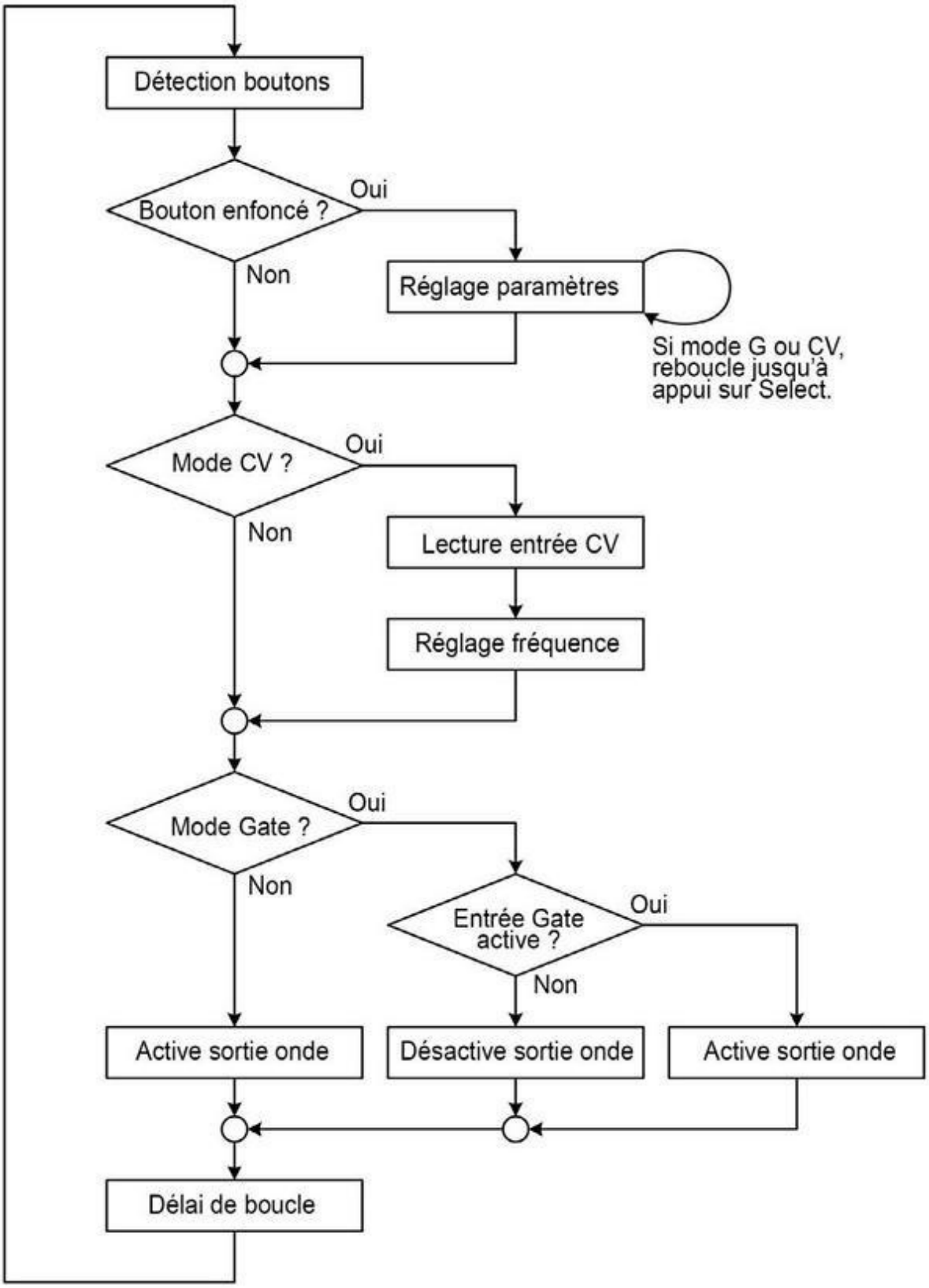


Figure 11.16 : Diagramme fonctionnel général de la fonction `loop()`.

Le Listing 11.3 montre le code source de la fonction `loop()`. Elle est très simple parce que tous les détails fonctionnels se trouvent dans les modules secondaires. Je rappelle que les appels de fonctions sont légers parce que les fonctions utilisent les variables globales.

Listing 11.3 : Fonction principale `loop()` *du générateur de signal.*

```
void loop() {
  // Lecture état boutons
  ReadControls();

  // Gestion modes
  SetMode();
  switch(currmode) {
    case MODE_DISP:
      getFreq();
      SetLCDVals();
      ShowGate();
      ShowCV();
      break;
    case MODE_GATE:
      SetGate();
      ShowGate();
      break;
    case MODE_CV:
      SetCV();
      ShowCV();
      break;
  }

  // Scrute si entrée CV active
  fval = RunCV();

  // Règle fréquence DDS
  dds_load_cnt++;
  if (dds_load_cnt >= DDS_LOAD_GO) {
    dds_load_cnt = 0;
    if (RunGate()) {
      ddsdev->setFrequency(fval);
   }
    else {
```

```
        ddsdev->setFrequency(0);
      }
    }
    delay(MAIN_DLY);
  }
```

Dès que nous avons transmis les données de contrôle au module DDS, nous faisons une pause de MAIN_DLY millisecondes avant de revenir en début de boucle. La fréquence de sortie se trouve dans `fval`. Elle peut être modifiée par l'entrée CV *via* un appel à `RunCV()`, si le mode CV est actif. La génération peut-être désactivée avec la fonction `RunGate()` si le mode correspondant est actif. Les données DDS sont transmises après un nombre de tours de boucle égal à DDS_LOAD_GO pour que le module ait le temps de prendre en charge les données de contrôle qu'il vient de recevoir. L'intervalle de mise à jour de la fréquence équivaut à DDS_LOAD_GO multiplié par MAIN_DLY en millisecondes. Si vous modifiez MAIN_DLY, vous devrez sans doute également retoucher DDS_LOAD_GO.

Le plus grand des fichiers source est *sig_gen_control.cpp*. Il se charge de la gestion des boutons d'entrée, de la suppression du rebond de touche, de la sélection d'un bouton maintenu enfoncé et du choix du mode d'entrée.

Je rappelle que les interrupteurs, poussoirs et commutateurs souffrent tous d'un effet de rebond parasite au moment de l'appui. Pour éviter ces rebonds, nous utilisons la simple fonction montrée dans le Listing 11.4.

Listing 11.4 : Fonction d'antirebond debounce().

```
bool debounce(int switchid)
{
  int swval = 0;
  if (swval = digitalRead(switchid)) {
    delay(DBDLY);
    if (swval != digitalRead(switchid)) {
      return false;
   }
    else {
      return true;
   }
  }
```

```
    return false;
  }
```

Le principe de l'antirebond est fort simple : si un contact est deux fois de suite dans le même état, on considère que cet état est valable car stabilisé. Si l'état change, c'est qu'il y a eu contact puis rupture (l'intervalle de test est contrôlé par DBDLY, défini dans *sig_gen.h*). Au départ, il vous laisse 10 ms, ce que vous pouvez augmenter pour améliorer l'effet d'antirebond. Mais si vous augmentez trop la valeur, la réponse des boutons va devenir molle.

La fonction `readControls()` teste l'état des boutons poussoirs. Si l'un d'eux est enfoncé, cela incrémente de 1 le compteur à chaque tour de boucle `loop()`. Lorsque la valeur dépasse celle prédéfinie dans HOLD_CNT (dans *sig_gen.h*), le bouton correspondant est déclaré maintenu enfoncé en donnant à la variable globale `btnhold` la valeur true. Je rappelle que la boucle principale s'exécute toutes les MAIN_DLY millisecondes.

Le choix de la fréquence est pris en charge par la fonction `getFreq()` qui augmente ou diminue la fréquence par pas de 10, 100 ou 1000 Hz. La valeur de la plage est ajustée d'office lorsque la fréquence atteint le plafond de la plage courante, en fonction de la valeur de RSTEP en hertz.

La fonction `SetMode()` se charge de passer du mode normal à l'un des deux autres modes, à condition qu'il ait été détecté un appui sur le bouton Mode par la fonction `readControls()`. Elle fait passer par les trois modes possibles, normal, Gate et CV tant que le bouton reste enfoncé. Les paramètres des deux modes Gate et CV ne peuvent être modifiés que lorsque le mode correspondant est actif (évidemment).

Comme déjà dit, le code source complet est disponible sur le site Web de l'éditeur.

La librairie du module DDS

Nous utilisons une librairie DDS très simple avec notre générateur. Pour la mettre en place, il faut commencer par créer un sous-répertoire *DDS* (en lettres majuscules) dans le répertoire d'installation de l'atelier Arduino *libraries*. Placez dans ce sous-répertoire les deux fichiers *DDS.h* et *DDS.cpp*. Si vous disposez d'une ancienne version de l'atelier IDE, la librairie n'apparaîtra dans la liste

déroulante des librairies qu'après redémarrage. Vous remarquez qu'il n'y a aucun fichier README, ni aucun fichier pour mettre les mots-clés en surbrillance (*keywords. txt*). Cette librairie est tellement simple que cela n'est pas nécessaire.

La librairie permet d'illustrer certains des concepts de création des librairies sur mesure présentés dans le Chapitre 5. La classe est simple, mais elle prend en charge tous les calculs requis pour préparer un mot de données de contrôle interprétable par le circuit AD9850. La définition de la classe est visible dans le Listing 11.5 (fichier source *DDS.h*).

Listing 11.5 : Définition de la classe DDS.

```
class DDS {
  private:
    float dds_cal;
    unsigned long dds_clock; // Horloge externe en Hz

    uint8_t dds_reset;  // DDS reset
    uint8_t dds_data;   // DDS data pin
    uint8_t dds_load;   // DDS data load pin
    uint8_t dds_wclk;   // DDS data load clock pin

    void pulseHigh(int pin);
    unsigned long freqData(unsigned long freq);
    void sendBit(uint8_t outbit);

  public:
    DDS(unsigned long clockfreq, uint8_t dds_data,
uint8_t dds_load,
  uint8_t dds_wclock, uint8_t dds_reset);
    void setFrequency(unsigned long freqval);
    void calibrate(float calval);
};
```

Une fois que vous avez créé l'objet à partir de la classe DDS, vous pouvez lui transmettre un coefficient de calibrage pour améliorer la précision du générateur. Vous apprendrez comment calculer cette valeur en vous reportant à la fiche technique de l'AD9850. Dans ce projet, j'utilise la valeur par défaut (zéro).

En plus du constructeur et de la fonction `calibrate()`, il n'y a qu'une autre méthode qui est utilisée à l'extérieur de la classe. Il s'agit de `setFrequency()` qui attend une fréquence en hertz. Notez que le mot de contrôle est transmis au circuit DDS un bit à la fois.

Test du prototype

Je suppose que vous avez vérifié le câblage du prototype. À partir de ce moment, vous devriez pouvoir lancer la compilation et le téléversement du logiciel vers la carte Arduino. L'afficheur devrait montrer successivement « DDS Signal Gen », puis « Initializing » puis « Ready » pour enfin mettre en place les champs de données. Vous pouvez retoucher les délais entre affichages à votre convenance.

Lorsque le montage est réalisé comme dans la Figure 11.7, la librairie nommée *Serial* n'est pas utilisable, car elle utilise les deux broches D0 et D1 pour Rx et Tx et les force à l'état haut. Cela perturbe le circuit 9850 qui ne va plus rien générer. Cette librairie *Serial* est très pratique pour la mise au point des poussoirs, comme je l'ai fait, mais il ne faut plus l'instancier une fois que le module DDS est présent. Si c'est vraiment nécessaire, vous pouvez utiliser en mode numérique les deux broches A4 et A5 en les affectant à deux des boutons, ce qui libère les broches D0 et D1 pour la liaison série. Je n'ai pas jugé cela utile dans ce projet. L'interface USB permet de téléverser le logiciel vers l'Arduino, même une fois que le module DDS est présent.

Le code est configuré pour générer une onde à 1000 Hz une fois sous tension. Vous pourriez choisir une autre valeur, mais celle-ci offre l'avantage d'être facilement visualisable sur un oscilloscope.

Il vous faut un oscilloscope pour tester les sorties du module DDS. Les matériels bon marché comme le Seeed Nano mono ou double canal (qui ressemble à un lecteur MP3), auront du mal à dépasser 1 MHz. Si vous devez tester des fréquences plus élevées, il faut un oscilloscope avec au moins 100 MHz de fréquence maximale. Cela dit, j'ai plaisir à utiliser mon petit Seeed Nano pour vérifier rapidement des circuits lents.

Commençons par vérifier la génération de sinusoïde. La Figure 11.17 montre celle à 1000 Hz générée par défaut. La tension doit être d'environ 1 V crête à crête. Dans le prototype, je n'ai pas encore prévu de potentiomètre de réglage du niveau de sortie. Si vous ne voyez pas apparaître la sinusoïde, débranchez l'unité et revérifiez toutes les connexions.

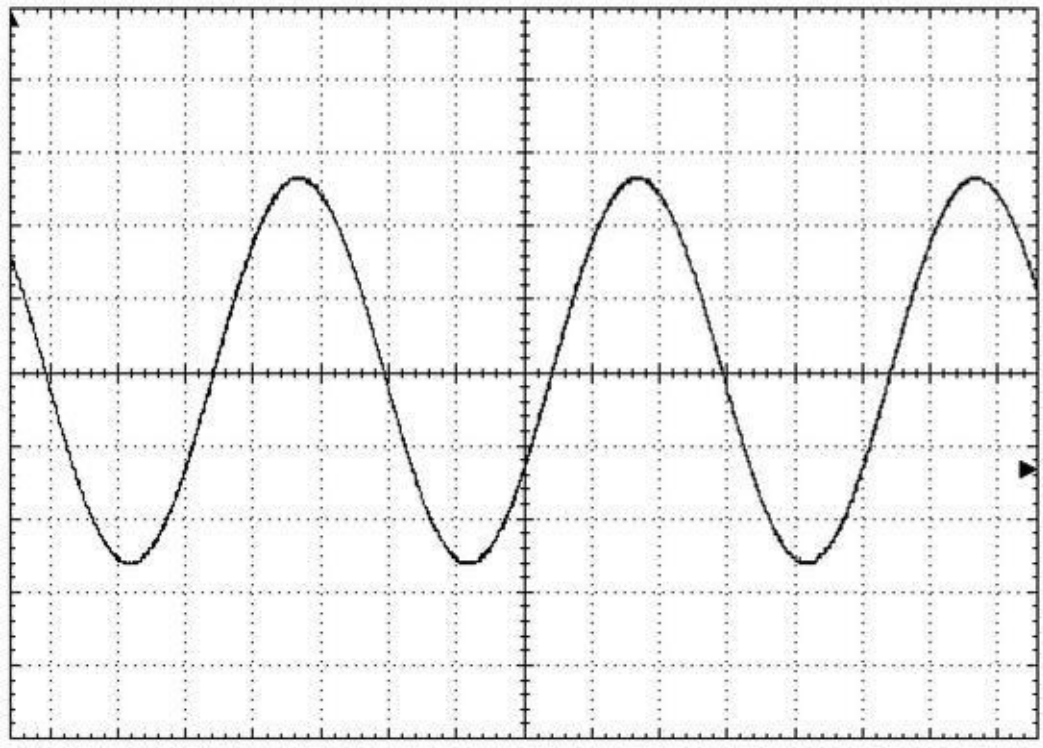


Figure 11.17 : Visualisation de sinusoïdale à 1 kHz.

L’onde carrée utilise la même fréquence, mais avec une amplitude d’environ 5 V crête à crête, ce qui correspond au plus de l’alimentation. Cette onde doit avoir l’aspect de la Figure 11.18.

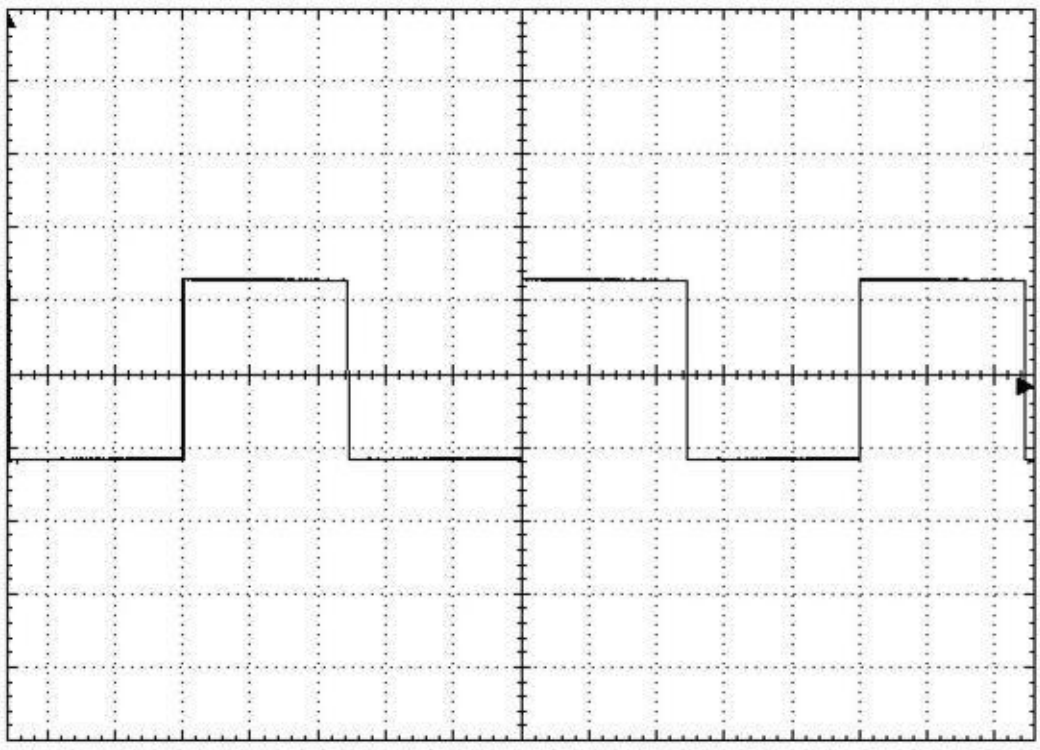


Figure 11.18 : Visualisation d’onde carrée à 1 kHz.

Si les deux ondes sont bien présentes, nous pouvons passer aux tests des poussoirs. Les boutons de fréquence et de plage seront testés d’abord. J’appuie d’abord sur les boutons Freq+ et Freq- pour voir changer la fréquence par pas de 1 Hz. Maintenez enfoncé le bouton Freq+ ; la fréquence va augmenter de 1, puis de 10, de 100 et de 1000. Relâchez à environ 15 000 et vérifiez l’onde générée. La plage change automatiquement lorsque la fréquence dépasse 9 999. Faites d’autres essais avec le bouton Freq-.

Le bouton Range+ doit faire augmenter la fréquence de 10 000 à chaque pression. Choisissez la valeur 4 pour la plage et observez le résultat. On devrait être dans les 44 kHz. Essayez ensuite le bouton Range- pour faire décroître la fréquence de 10 000 à chaque fois.

Appuyez maintenant sur le bouton Mode et maintenez-le jusqu’à ce que vous voyiez clignoter le symbole G du mode Gate. Avec les boutons Freq+ et Freq-, vous pouvez maintenant activer et désactiver l’entrée de commande. Les deux boutons Range+ et Range- servent à changer l’état actif de l’entrée. Il doit passer de plus à moins. Revenez au mode normal avec le bouton Select.

Si vous avez activé le mode Gate sans rien brancher sur l’entrée, la génération doit s’arrêter. Sélectionnez le mode d’entrée actif à l’état Bas. La sortie doit

devenir active lorsqu'un cavalier est branché entre l'entrée Gate (broche A2 du Uno) et la masse. Dans le mode actif Haut, la sortie sera active lorsqu'A2 sera relié au +5 V. Pour quitter le mode Gate, entrez dans le mode et utilisez le bouton Freq-.

Il reste à tester l'entrée de tension de commande CV (broche A3 de l'Arduino). Pour ce test, il faut soit une alimentation réglable, soit une alimentation 5 V avec un potentiomètre de 10 kilohms ainsi qu'un multimètre numérique (Figure 11.19). Si vous n'avez pas de fonction d'affichage de la fréquence sur votre oscilloscope, vous aurez alors besoin d'un fréquencemètre.

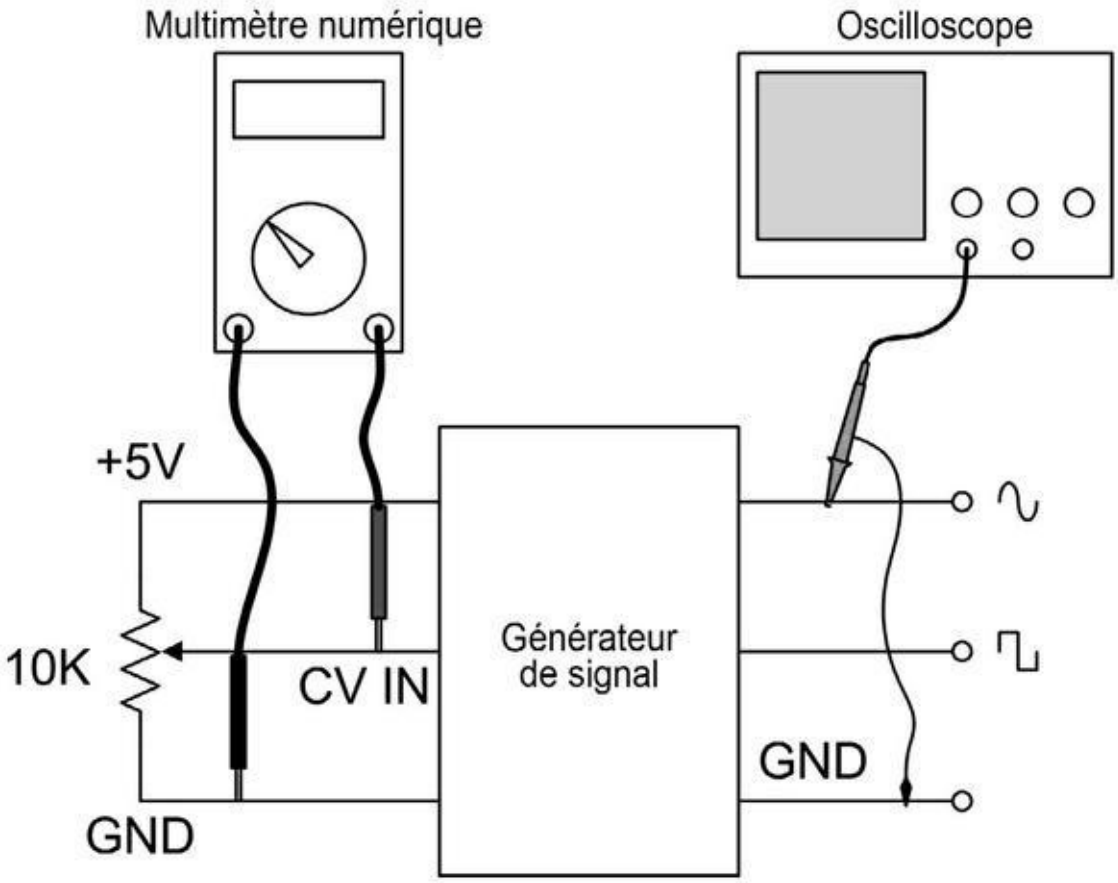


Figure 11.19 : Configuration du test du mode CV.

Avec le bouton Mode, activez le mode CV, mais ne changez pas le réglage du point zéro qui doit être aux alentours de 2,5 V. Utilisez le bouton Select pour revenir en mode normal. Appliquez une tension entre zéro et 5 V, tout en observant le signal généré. La fréquence doit diminuer lorsque la tension CV est inférieure au point zéro et augmenter lorsqu'elle est supérieure. Remarquez

également la polarité de l’entrée CV grâce aux indications + et -. Il est presque impossible de retrouver exactement le point zéro manuellement, mais vous avez le temps de voir le zéro s’afficher lors du passage par cette valeur.

Réduisez maintenant la tension CV pour qu’il y ait environ 2 V sur le multimètre. Repassez en mode CV, activez l’entrée CV et utilisez le bouton Range+ pour changer le point zéro. Le symbole de polarité doit maintenant indiquer 0 et doit rester inchangé tant que la tension d’entrée ne change pas. Modifiez maintenant la tension d’entrée ; la fréquence de sortie doit augmenter ou diminuer selon que la tension d’entrée soit sous ou au-dessus du nouveau point zéro. Pour finir, désactivez la fonction d’entrée en mode CV avec le bouton Mode et le bouton Freq-.

N’appliquez jamais une tension négative ni une tension supérieure à 5 V à l’entrée CV car ce prototype n’a aucune protection des entrées. Cela risquerait de détruire le microcontrôleur. Dans la version finale, nous ajouterons un circuit de protection simple.

Nous en avons terminé avec les tests du prototype. Si tout fonctionne bien, nous pouvons en profiter pour faire d’autres essais avec le générateur de signal. Voyons encore comment régler la largeur d’impulsion de l’onde carrée.

Sur le circuit du module DDS, à l’opposé du module de l’oscillateur argenté, il y a un petit potentiomètre de réglage. Réglez la fréquence à environ 10 kHz et modifiez la forme de l’onde carrée avec un oscilloscope pour que les deux périodes soient de même durée. Dans la version définitive, vous pourriez dessouder le potentiomètre pour le déporter sur la face avant ou à l’arrière du boîtier.

Assemblage final

Si vous choisissez de monter le module DDS sur un bouclier de prototypage, je vous conseille de prévoir une paire de borniers à vis enfichables comme ceux montrés en Figure 8.80 du Chapitre 8. Ils permettent de réaliser des connexions solides, tout en surélevant un peu le module DDS par rapport à la carte Arduino, ce qui ménage un peu d’espace si vous avez besoin d’accéder au connecteur ICSP. Vous pouvez aussi utiliser un bouclier de bornier à vis comme celui montré à la fin de ce chapitre.

Un réseau de résistances de protection

J’ai installé six résistances de 2,2 kilo-ohms sur une chute de plaque à trous afin qu’elles servent de résistances de rappel (*pull-ups*) pour les interrupteurs. Le microcontrôleur AVR est capable de se protéger avec des résistances de rappel internes, mais grâce à mon réseau de résistances, vous serez certain qu’il y aura toujours une tension positive pour les boutons. Cela signifie en outre que les boutons sont actifs à l’état bas, donc avec 0 V quand ils sont enfoncés.

Ce module est très simple, comme le montre le schéma de la Figure 11.20. Les six résistances viennent en parallèle des lignes des interrupteurs, et leur point commun est relié au +5 V.

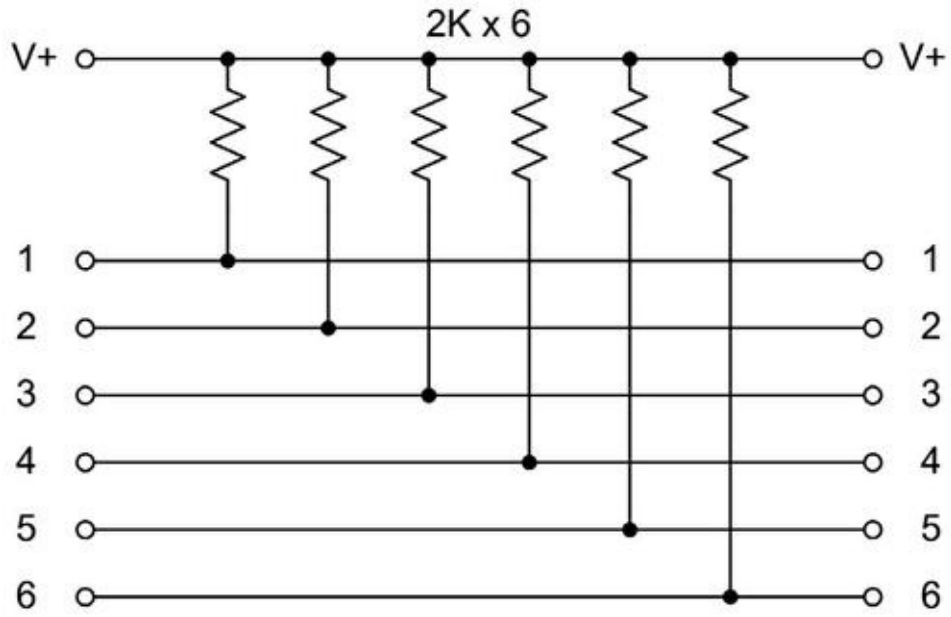


Figure 11.20 : Schéma du réseau de résistances de rappel.

Le montage est visible au centre de la Figure 11.21. Les petits borniers à souder permettent d’assurer des connexions fiables. Dans la figure, vous pouvez voir à droite le module de protection que nous allons découvrir maintenant, ainsi qu’à gauche le montage constitué de la carte Arduino et du module DDS. Pour l’instant, seule l’alimentation est branchée.

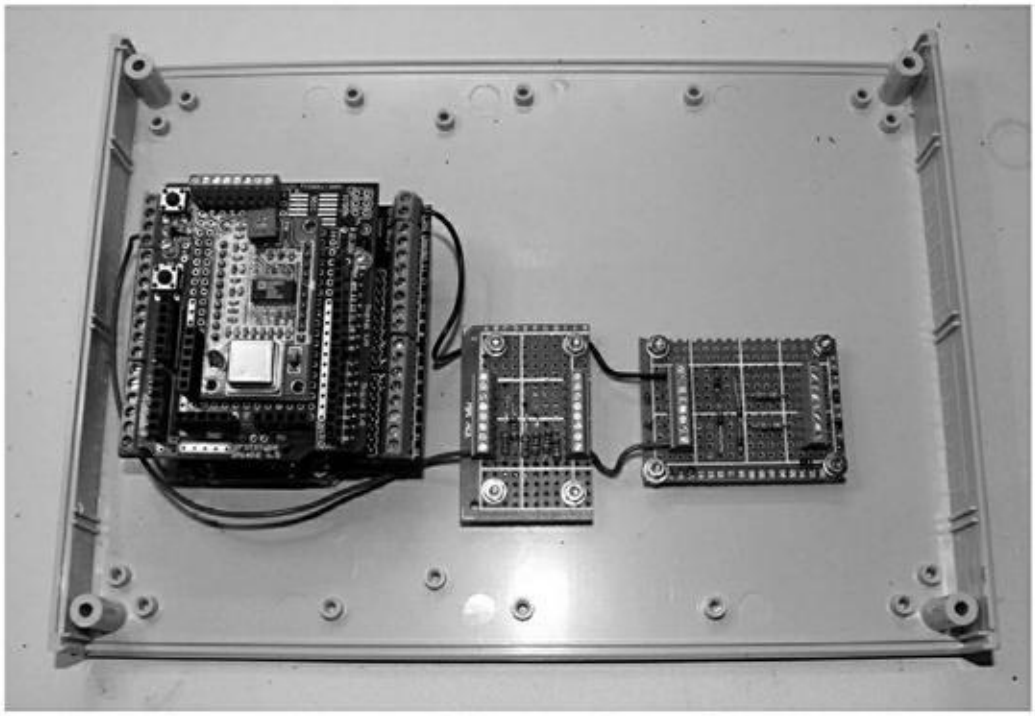


Figure 11.21 : Le réseau de résistance et le module de protection des entrées.

Protection des entrées

Un autre petit bout de plaque à trous permet de construire un circuit de protection des entrées (Figure 11.22). Ce circuit a pour but d'éviter toute tension supérieure au positif d'alimentation ou inférieure à zéro afin de protéger les entrées Gate et CV.

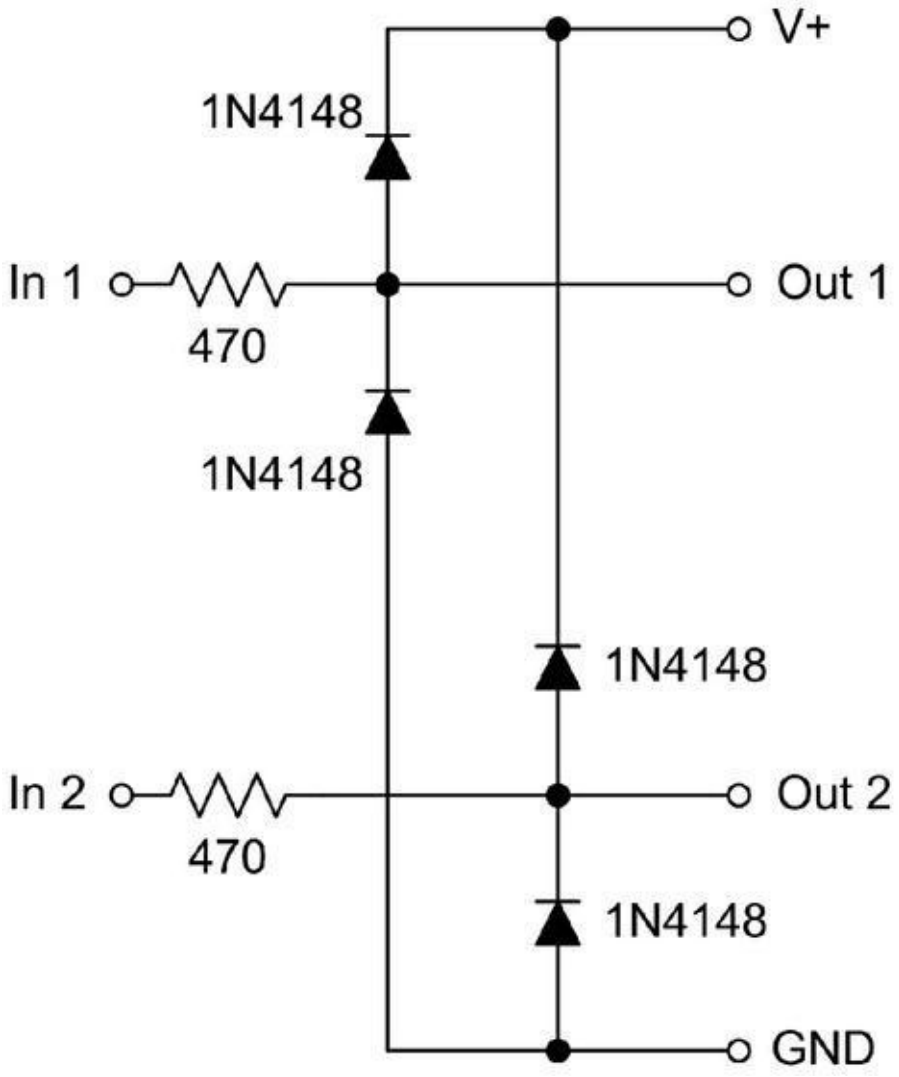


Figure 11.22 : Schéma du circuit de protection des entrées de contrôle.

Pour chaque entrée, la résistance de limitation de courant de 470 ohms est en série avec l'entrée, ce qui diminue un peu la tension présentée à l'entrée du convertisseur. Ce n'est pas gênant, car l'entrée Gate fonctionne en tout ou rien et l'entrée CV est en mode relatif. Vous pourrez même augmenter la valeur des résistances si vous avez peur qu'il y passe trop de courant à travers les diodes. Le module de protection était visible dans la Figure 11.21. Nous avons ici aussi utilisé des borniers à vis.

Face avant et châssis

Le logiciel est prêt et tous les composants sont disponibles. Nous pouvons passer au montage final dans le boîtier. La partie la plus délicate correspond au perçage de la face avant. Percez par l'arrière pour obtenir un beau fini de surface et éviter de rayer la face avant au cas où le foret de perceuse déraperait. Vous pouvez ensuite monter la carte Arduino et le module DDS sur le fond du boîtier puis les connecteurs USB et d'alimentation à l'arrière. Il ne reste plus ensuite qu'à effectuer toutes les connexions.

Vous commencez par découper la face avant et la face arrière. Dans la face avant, il faut prévoir un évidement rectangulaire pour l'afficheur LCD et un certain nombre de trous pour les boutons et les connecteurs. À l'arrière, il faut prévoir des trous pour le connecteur d'alimentation et le potentiomètre de contraste de l'affichage. Nous ne ramenons pas vers l'arrière la prise USB de la carte Arduino. Lorsque nous aurons besoin d'y accéder, nous ouvrirons le capot. La Figure 11.23 montre le perçage en cours. Vous utiliserez une perceuse miniature de modélisme. Une chignole à main pourrait aussi convenir, mais il faut dans ce cas travailler lentement et bien marquer les trous avec une mèche de tout petit diamètre. Vous pouvez dessiner tous les perçages à l'échelle sur une feuille de calque puis la fixer sur la plaque et réaliser vos perçages à travers. La Figure 11.24 montre le plan de perçage complet.

Figure 11.23 : Perçage de la face avant.

Pour les trous servant à monter l'afficheur LCD, voyez le diamètre exact nécessaire en fonction du modèle d'afficheur dont vous disposez. Je ne fournis pas de plan de perçage pour le panneau arrière, puisqu'il ne reçoit que le potentiomètre et la prise jack de l'alimentation.

Certains pourraient croire qu'il est inutile de consacrer trop de temps à créer des dessins détaillés comme ceux que nous venons de voir. Pourtant, ils montrent tout leur intérêt à long terme, car ils permettent de conserver une trace de ce que vous avez fait et surtout, ils vous simplifient la vie si vous avez besoin d'intervenir plus tard, ou de construire d'autres générateurs de signal. Pourquoi réinventer la roue à chaque fois ?

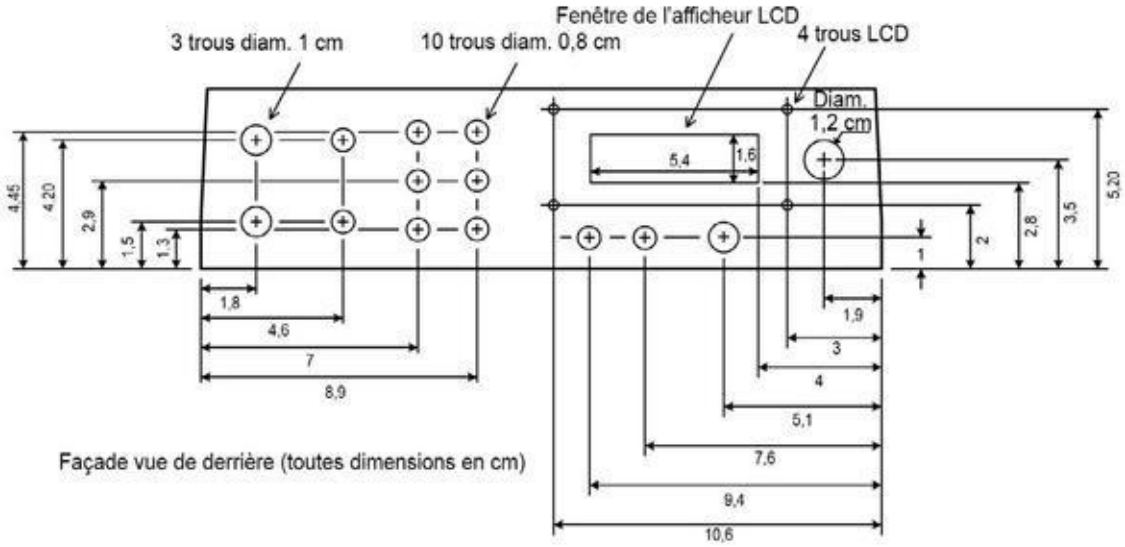


Figure 11.24 : Plan de perçage de la face avant du générateur de signal.

L'ensemble Arduino Uno et module DDS est fixé directement sur le fond du boîtier avec les deux petits montages volants. La position exacte n'a pas d'importance, mais cherchez à placer la partie Arduino non loin de la face avant pour simplifier le câblage. Nous avons vu dans la Figure 11.21 comment se présentent les trois composants internes avant leur liaison à la face avant.

Une fois cette face avant percée, nous pouvons passer à la mise en place des composants. N'oubliez pas de prévoir des légendes pour les boutons et les connecteurs.

Si vous disposez d’une machine pour réaliser les écritures, vous pouvez monter les composants d’abord. En revanche, toutes les autres techniques vont vous obliger à faire la mise en place des légendes avant d’installer les composants. Assurez-vous de disposer du bon modèle de ruban si vous utilisez une machine à titrage. Certains modèles n’ont pas de ruban blanc sur fond noir. J’ai imprimé mes légendes en blanc sur fond noir avec mon imprimante laser puis j’ai mis en place les petits textes après les avoir découpés et collés.

Le boîtier qui paraît vide au départ est rapidement rempli avec toutes ces connexions. La Figure 11.25 montre le générateur de signal juste avant de refermer le capot.

Ne replacez par tout de suite le capot ! Laissez l’instrument ouvert pendant tous les tests de recette. Une petite erreur peut se produire. Personne n’est parfait.

L’afficheur LCD a été mis en place avec quatre entretoises en nylon et des écrous et rondelles. J’ai soudé des connecteurs sur le circuit de l’afficheur puis coupé les broches à l’arrière d’environ 6 mm. Cela m’a permis d’y brancher des connecteurs mâles et d’obtenir ainsi un connecteur sur mesure tout à fait valable pour mon afficheur LCD. J’ai protégé les connexions par de la gaine thermorétractable. Les connexions de l’afficheur sont visibles dans le haut de la Figure 11.25.

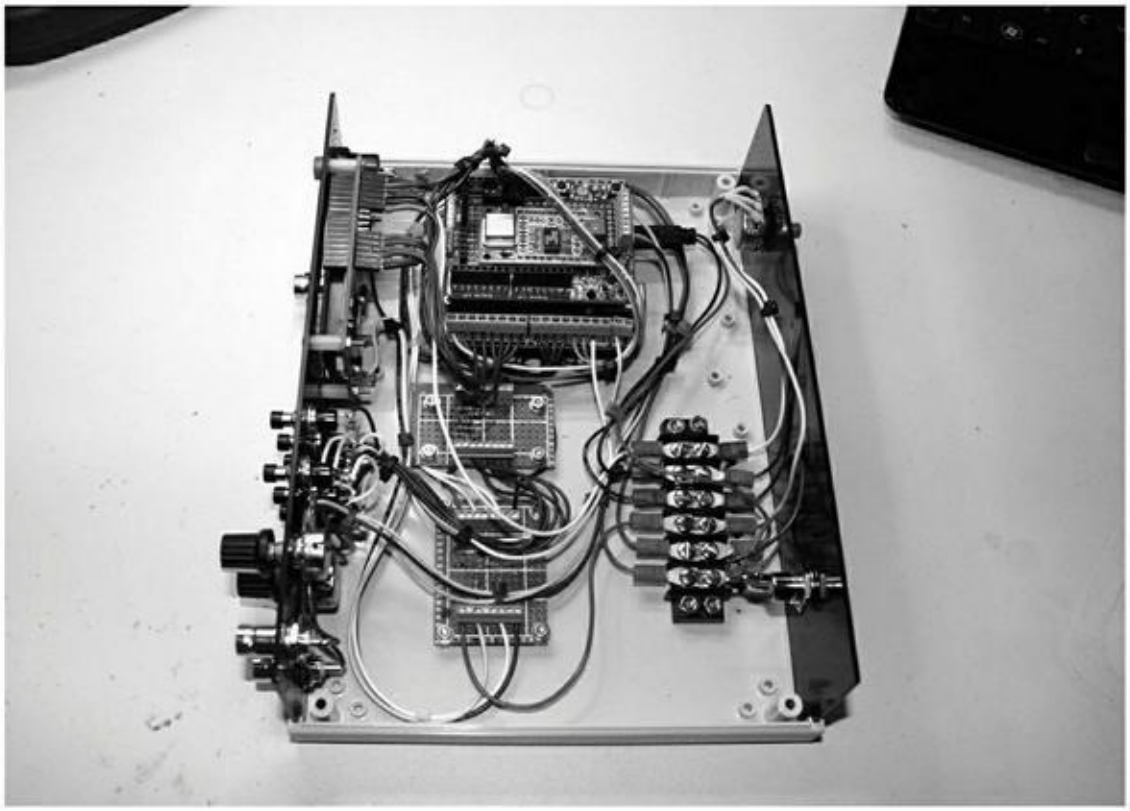


Figure 11.25 : Vue du générateur de signal totalement câblé.

J’ai regroupé les fils par fonction avec des colliers nylon, et j’ai vraiment apprécié le confort que procurent les plaques de borniers à vis enfichables. Pour connecter la face arrière, c’est-à-dire le jack d’alimentation et le potentiomètre de réglage de contraste, j’ai mis en place un bloc de six gros borniers plats sur lesquels j’ai fait les branchements avec des cosses.

Alimentation

La connexion depuis le jack de la face arrière passe par le bornier puis rejoint l’interrupteur en face avant pour aboutir ensuite sur la carte Arduino. Je n’ai prévu aucun fusible car le positif de l’alimentation ne s’approche à aucun moment d’un endroit où il pourrait se retrouver en court-circuit avec la masse. Cela dit, ajouter un fusible ne serait pas difficile.

Les gens d’Arduino conseillent d’utiliser un adaptateur continu de 9 à 12 V pour faire fonctionner Arduino sans brancher sa prise USB. Si vous prenez un adaptateur 5 V, le régulateur sur la carte Arduino va provoquer une chute de tension telle que vous n’aurez plus que 3,5 V disponibles. Voilà pourquoi j’ai

opté pour un adaptateur de récupération de 9 V CC. (C’est incroyable comme on finit par accumuler ce genre de petits cubes noirs).

Tests définitifs et fermeture

Le montage est installé dans le boîtier, la face avant comporte toutes les légendes et le logiciel est téléchargé. Nous sommes prêts à réaliser nos derniers tests avant de refermer le capot. À vrai dire, il s’agit essentiellement d’un test de non-régression, qui consiste à dérouler les mêmes tests qu’avec le prototype. Nous devons nous assurer que tout fonctionne toujours aussi bien et que rien n’a été altéré entre le prototype et la version définitive.

Pour réaliser des tests plus poussés, il faut disposer de certains équipements qui ne sont pas à la portée de la plupart des gens : un analyseur de distorsion, un analyseur de spectre, *etc.* Si vous connaissez quelqu’un qui possède ce genre d’appareils, n’hésitez pas à lui emprunter. En fonction du type de mesure que vous allez avoir besoin de réaliser avec votre nouveau générateur, il pourra s’avérer indispensable de connaître le niveau de distorsion harmonique à différentes fréquences, et de savoir si ce taux est constant ou s’il varie en fonction de la fréquence de sortie. Vous devrez peut-être aussi connaître le taux de pureté de l’onde sinusoïdale au niveau des harmoniques et le temps de montée et de chute de l’onde carrée, l’impédance de sortie, la relation entre fréquence et tension de l’entrée de commande CV, ou encore la stabilité en fréquence sur longue durée. Vous pourriez même avoir besoin de connaître les temps de réponse des deux entrées CV et Gate.

Une fois vos derniers tests réalisés, il ne reste plus qu’à replacer le couvercle. Pensez à ajouter des patins en caoutchouc dessous. L’instrument en pleine utilisation avec un oscilloscope USB et un portable est visible dans la Figure 11.26.

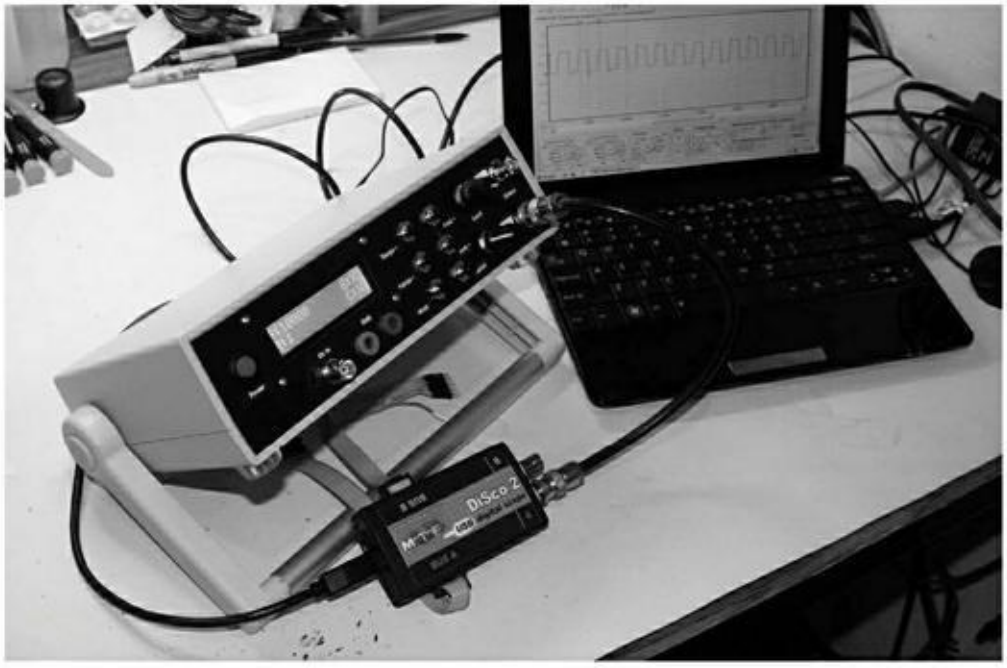


Figure 11.26 : Le générateur de signal en plein travail.

Si vous observez bien la Figure 11.26 et la comparez à la Figure 11.3 du début du chapitre, vous devriez remarquer quelque chose au sujet de la poignée. J'ai bien pris soin de ne pas confondre le dessus et le dessous comme je vous en avais averti, mais j'ai interverti l'avant et arrière ! Je n'irai pas jusqu'à prétendre que c'était volontaire, mais il faut avouer que le résultat est tout à fait intéressant. Normalement, la poignée doit se déplier vers l'avant lorsque l'appareil est posé à plat. Mais cela occupe un peu d'espace sur le plan de travail. Grâce à mon étourderie, la poignée se trouve à l'arrière de l'appareil, ce qui est bien plus pratique. À toute chose, malheur est bon. Cela dit, les erreurs n'ont pas souvent une fin aussi heureuse.

Réduction des coûts

J'ai prévenu en début de chapitre que ce projet n'était pas très bon marché, à cause du boîtier, des connecteurs et des compléments de confort que sont la plaque de borniers enfichables, le réseau de résistances de rappel et le module de protection des entrées. Si vous avez besoin de réduire le coût global, voici quelques astuces qui ne l'empêcheront pas de rendre les services prévus.

Au niveau de la carte Arduino, on trouve dorénavant des compatibles pour moins de 15 euros. Le module DDS, incontournable, va coûter environ 10 euros.

Une carte bouclier de prototypage avec des borniers à vis coûte environ 15 euros, mais vous pouvez opter pour une version plus simple, qui ne permet que de placer le module DDS (environ 10 euros). Vous pourriez même abandonner le boîtier et vous contenter d'utiliser un empilage de boucliers sur l'Arduino. Il existe plusieurs boucliers d'afficheurs LCD dotés d'une série de boutons, dans le style de celui montré dans la section des afficheurs LCD du Chapitre 8. Vous vous en sortirez pour environ 10 euros. Ce genre de bouclier comporte quatre ou cinq boutons, ce qui vous obligera à revoir le code source de gestion de l'interface utilisateur pour vous en sortir avec moins de six boutons.

Si vous n'avez pas besoin que votre appareil soit autonome, vous pouvez vous passer de l'afficheur et des boutons poussoirs. Dans ce cas, vous serez dépendant de la machine hôte en n'utilisant que la carte Arduino et le module DDS. Le résultat n'est plus le même type d'appareil et je n'en parle pas dans ce chapitre. Pour en savoir plus sur la manière d'ajouter une interface de contrôle à distance, revoyez le logiciel présenté dans le Chapitre 10 pour les deux projets Greenshield et Switchinator.

Si vous réunissez les trois cartes indispensables (une carte Arduino, un bouclier de prototypage avec son module DDS et un bouclier LCD), vous obtiendrez quelque chose dans le style de la Figure 11.27.

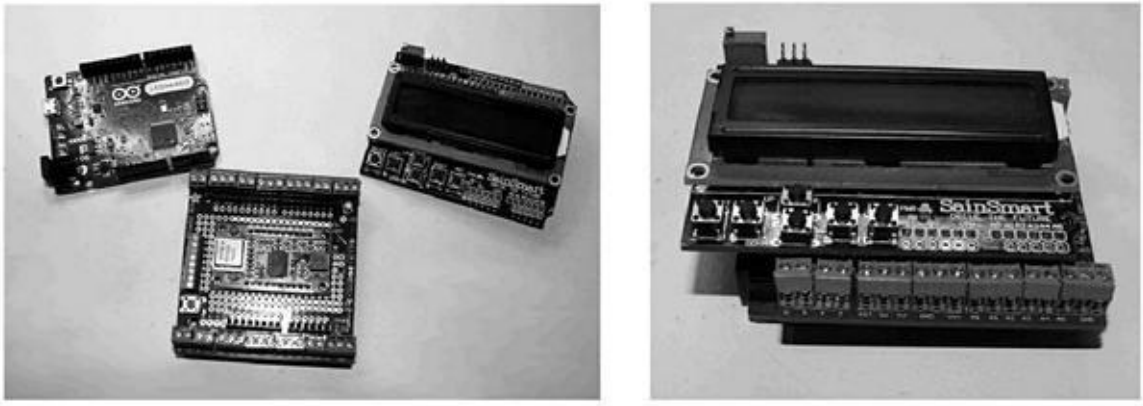


Figure 11.27 : Version minimaliste du générateur de signal.

Dans la figure précédente, il y a un Arduino Leonardo, et j'ai utilisé des borniers pour circuit imprimé à écartement de 2,54 mm pour les connexions avec le module DDS. Il n'y en revanche aucun autre connecteur, pas de réglage du niveau de sortie et pas de boîtier. Dans cette version, vous devez rester vigilant. Le moindre fil qui traîne non loin du générateur pourrait devenir une cause de court-circuit et de destruction.

Cette version nue du générateur devrait vous permettre d'économiser environ 30 euros. À vous de voir si cette économie se justifie quand on considère la plus grande fragilité du résultat et sa dépendance de la machine hôte.

Analyse des coûts

Le Tableau 11.6 dresse la nomenclature chiffrée du générateur de signal. Les prix sont moyens et ne tiennent pas compte des frais de livraison qui sont assez variables d'un fournisseur à l'autre. Certains n'hésitent pas à vous facturer 9 euros de port pour un composant à 5 euros. J'essaie d'éviter ce genre de relation commerciale. D'un autre côté, les fournisseurs chinois offrent en général les frais de port, mais vous devrez attendre plusieurs semaines pour être livré. Il y a de nos jours suffisamment de fournisseurs avec un bon site Web. Vous trouverez une liste en Annexe C.

Tableau 11.6 : Nomenclature et prix des composants.

Quantité	Composant	Source	Px unitaire	Px total
1	Boîtier, Bud IP-6130	Mouser	25,40	25,40
1	Arduino Uno	Adafruit	24,95	24,95
1	Module générateur DDS	DealeXtreme	7,99	7,99
1	Carte prototype	Adafruit	9,95	9,95
1	Plaque borniers enfichables	Seeed Studio	7,50	7,50
1	Afficheur LCD, 16 × 2	Amazon/Uxcell	4,71	4,71
3	Connecteur BNC	All Electronics	1,25	3,75
2	Fiche banane	Amazon	0,67	1,34
3	Potentiomètre, 10 K	Amazon/Amico	1,28	3,84

1	Bouton arrêt/marche	All Electronics	1,35	1,35
6	Bouton poussoir miniature	All Electronics	0,60	3,60
1	Embase alimentation jack	Parts Express	1,98	1,98

Coût total des composants = 96,33 euros.

J’ai déjà prévenu en début de chapitre que le générateur que je propose de construire ici va vous coûter plus cher que la plupart des kits de générateurs du commerce. En revanche, vous disposez ainsi d’un contrôle complet et votre appareil offre une gamme de fréquences largement suffisante pour tous les projets amateurs de radio et de traitement numérique à haute vitesse.

Note : ce montant ne prend pas en compte les fils de connexion, la soudure, le ruban adhésif et l’adaptateur secteur. Ces prix ne sont valables qu’au moment où j’ai dressé la liste.

En tirant les prix, vous devriez réussir à descendre à environ 75 euros pour la version complète. Vous économiserez si vous disposez d’un bon stock de composants, par exemple pour les connecteurs, les interrupteurs et potentiomètres. Pour le boîtier, vous pourrez peut-être recycler le boîtier d’un appareil hors d’usage.

CHAPITRE 12
Un thermostat intelligent

Votre domicile est peut-être équipé de thermostats intelligents, ou bien vous en avez entendu parler dans les publicités. Ce genre d'appareil sert à contrôler la température d'un lieu fermé. Les plus récents modèles peuvent être contrôlés à distance *via* une liaison Bluetooth ou WiFi, depuis une application sur telliphone ou tablette. Certains permettent même de recueillir des données historiques et de les télécharger pour détecter les pointes de consommation. Il y a enfin la grande masse des modèles de début de gamme qui ne font rien d'autre que ce que permettait le bon vieux thermostat bilame mécanique, sauf qu'ils y ajoutent un afficheur LCD.

Si vous choisissez de construire le montage de thermostat de ce chapitre, sachez que vous le faites à vos propres risques. Ceux-ci sont réduits puisque nous n'utilisons que de la basse tension, mais cela n'empêche qu'il est possible d'endommager le système de chauffage ou de climatisation si vous programmez des changements de cycle ou de température dépassant les limites de l'équipement. Cela dit, la plupart des systèmes ont des sécurités. N'utilisez que le circuit basse tension pour vous relier à l'équipement de chauffage ou de climatisation. Ne connectez jamais votre thermostat à la tension du secteur. Cette mise en garde s'applique tout autant aux rafraîchisseurs et aux radiateurs électriques.

Quelques principes

La Figure 12.1 montre l'intérieur d'un thermostat mécanique bilame en vigueur depuis quasiment un siècle. Il y a de nombreuses possibilités de l'améliorer. Son principe est d'accoler sur un même ruban deux métaux de coefficient de dilatation différent. Lorsque la température change, le ruban se rétracte/s'enroule ou s'allonge/se redresse.

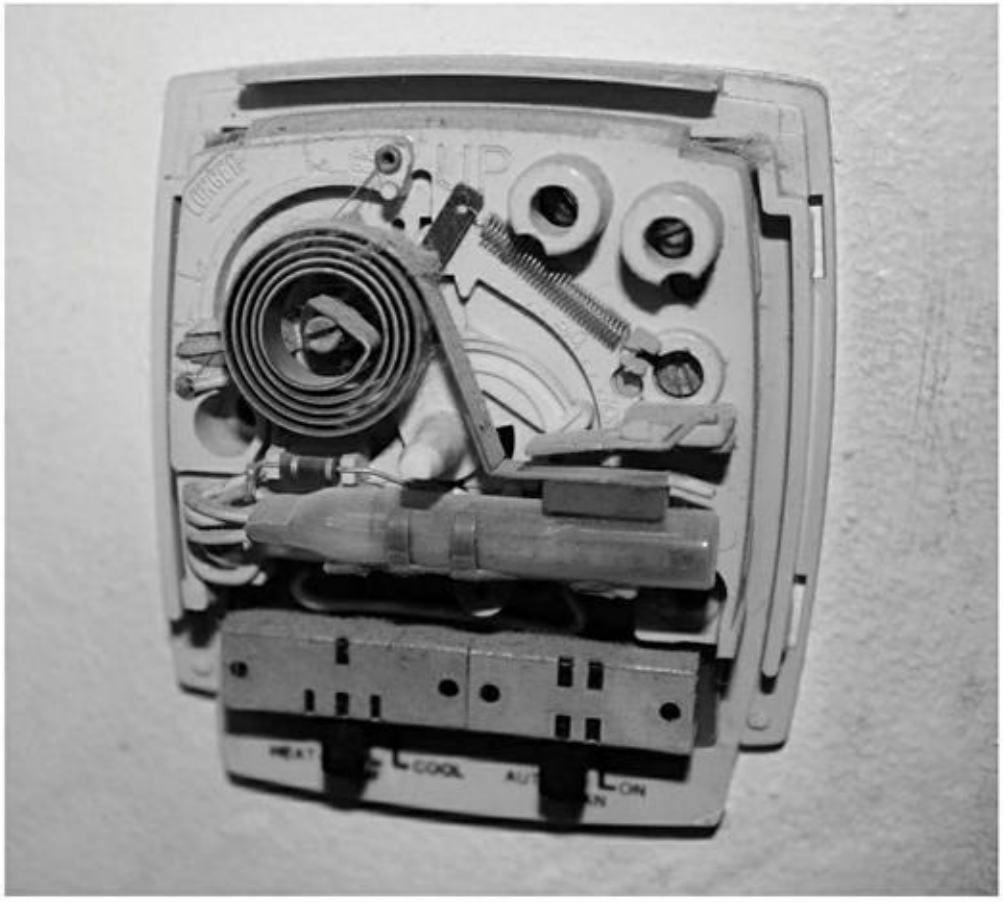


Figure 12.1 : Thermostat électromécanique bilame.

Cette dilatation différentielle permet d'établir un contact ou de le rompre. Certains modèles comportent un tube de verre dans lequel se trouvent un contact et une goutte de mercure. Le tube est déplacé par le bilame pour établir le contact et fermer le circuit. (La même technique est utilisée par le détecteur de choc KY-017 vu dans le Chapitre 9.) C'est un organe de contrôle de type « tout ou rien ».

Nous allons voir dans ce chapitre comment concevoir, fabriquer puis programmer un thermostat intelligent en utilisant une carte Arduino et des composants courants. Mais avant de passer à la conception, il n'est pas inutile de revoir la nature de la variable physique que nous voulons mesurer et contrôler.

Principes d'une climatisation

Contrôler la température d'un lieu fermé consiste à y ajouter ou à en enlever de la chaleur. Pour ajouter de la chaleur à l'espace fermé, nous pouvons utiliser une source de chaleur telle qu'un radiateur à gaz, un chauffage au bois ou au charbon, un radiateur ou l'énergie solaire. Les pompes à chaleur (PAC) récupèrent la chaleur de l'air extérieur pour la réinjecter à l'intérieur, ce qui revient à utiliser une climatisation en sens inverse. Pour rafraîchir le local, il faut utiliser un système de réfrigération basé sur un cycle de condensation/ évaporation d'un fluide approprié. Le fréon a été abandonné de nos jours au profit de fluides plus écologiques. La chaleur du local est absorbée par le fluide qui est envoyé à l'extérieur par un circuit fermé. Les grosses climatisations industrielles utilisent de l'eau, mais cela implique de prévoir beaucoup d'espace physique pour les équipements, et des coûts de fonctionnement importants. Ce genre de système ne se rencontre donc que dans les bâtiments collectifs et gouvernementaux, les hôpitaux et les grandes entreprises.

Dans l'univers résidentiel, les fonctions de chauffage et de rafraîchissement sont encore souvent prises en charge par deux systèmes distincts, mais on trouve de plus en plus souvent des climatisations réversibles qui permettent de chauffer et de rafraîchir. Les pompes à chaleur comportent en général deux éléments, un à l'intérieur et à l'extérieur. Ces équipements comportent un interrupteur pour passer du mode automatique au mode manuel pour la ventilation et un autre interrupteur pour choisir entre chauffage, rafraîchissement et arrêt. La température désirée, que l'on appelle *point de consigne*, est sélectionnée par un bouton rotatif. L'ensemble porte le nom anglais HVAC (*Heath*, *Ventilation*, *Air Conditioner*). En français, c'est CVC, Climatisation Ventilation Chauffage. Les appareils CVC des habitations fonctionnent sur les mêmes principes que les grands modèles, à l'exception des fonctions de ventilation et d'échangeur d'air, qui sont destinées au matériel haut de gamme.

La plupart des systèmes CVC peuvent fonctionner en circuit fermé, en faisant recirculer l'air intérieur. Certains systèmes peuvent prélever de l'air extérieur, ou effectuer un échange entre les deux airs par la ventilation. Lorsque l'apport d'air

frais n’est pas prévu, il est souvent possible de deviner ce qu’il y avait au menu de la veille au soir. C’est ce qui fait d’ailleurs que chaque maison a une odeur distincte. Quand j’étais enfant, j’avais constaté que dans la maison d’un voisin, cela sentait toujours le poulet grillé lorsqu’ils utilisaient leurs climatiseurs. J’en avais déduit que c’était des fanatiques du poulet grillé. En fait, je n’ai jamais su qu’elle était l’origine réelle de cette odeur persistante.

Principe du contrôle de température

Pour contrôler la température d’un lieu, il faut piloter les sous-systèmes qui produisent de la chaleur ou du froid afin d’atteindre et de maintenir la température désirée (celle du point de consigne). Les systèmes domestiques ne sont généralement pas capables de fonctionner de façon progressive, et doivent être soit en marche, soit arrêtés : soit la fonction de chauffage est active à sa puissance maximale, soit elle est éteinte. De même pour la fonction de rafraîchissement.

Ce fonctionnement en tout ou rien a pour conséquence que la température réelle ne sera jamais exactement celle du point de consigne, sauf de façon transitoire, lors du passage d’un état à l’autre. Supposons que nous ayons choisi un point de consigne de 22 ^{o}C. Si nous enregistrons la température intérieure et extérieure sur 24 heures, nous pourrions obtenir un diagramme comme celui de la Figure 12. 2.

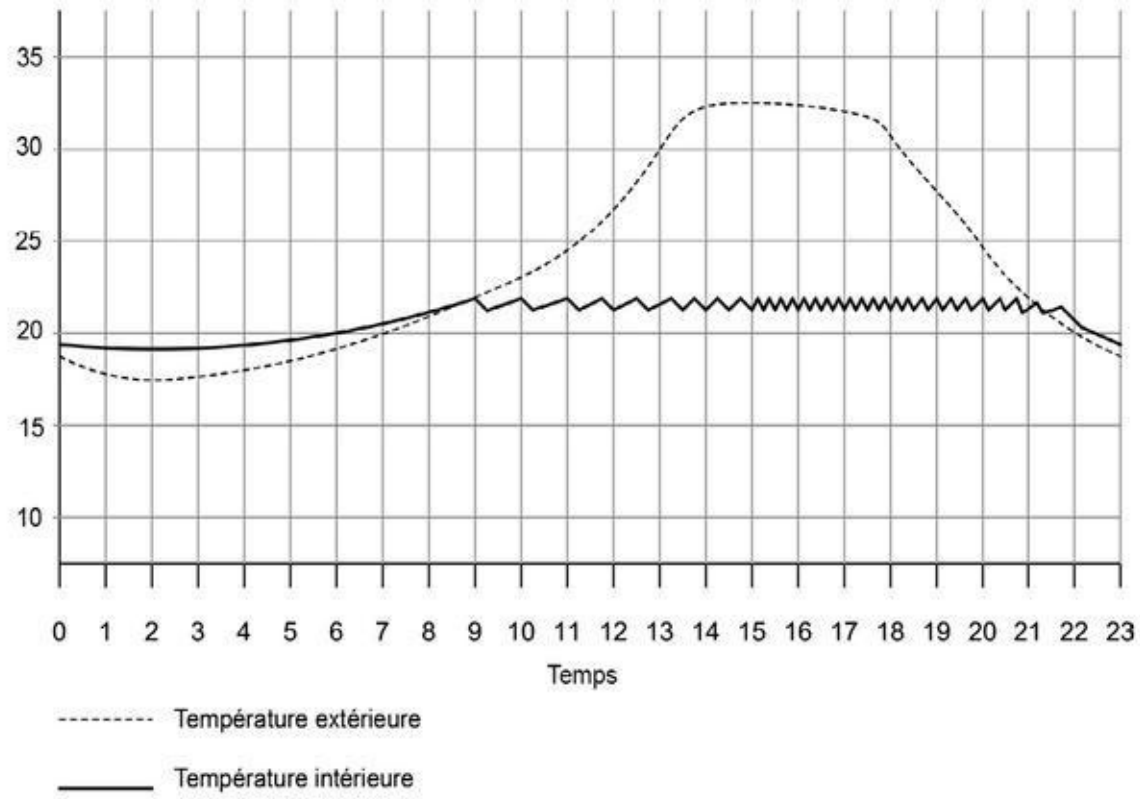


Figure 12.2 : Courbe de température extérieure et intérieure sur 24 heures.

Le refroidissement est actif dans cette figure à chaque fois que la température intérieure décroît. Dans la matinée, le système se met en route rarement, car il faut du temps pour que la chaleur extérieure pénètre dans le bâtiment, ce qui est encore ralenti par la qualité de l'isolation. Mais en fin d'après-midi, le climatiseur a beaucoup de travail et se met en route très souvent. Les choses ne commencent à se calmer qu'à la nuit tombée.

Une climatisation qui fonctionne en tout ou rien est un contrôleur à hystérésis. La quantité d'hystérésis (de délai de réaction) détermine directement la fréquence à laquelle le système est mis en route et pour combien de temps. L'hystérésis, qui signifie « retard » en grec, est un délai incontournable qui s'ajoute au moment théorique idéal d'arrêt ou de démarrage du système en fonction du point de consigne. On peut ressentir le même genre d'effet de retard lorsque l'on ouvre et ferme les anneaux d'un classeur à trous. Il faut un peu d'effort pour l'ouvrir et un autre effort pour le refermer, mais les anneaux restent dans l'état ouvert ou fermé une fois qu'ils y sont.

La Figure 12.2 est un exemple, mais il suffit à montrer comment fonctionne un thermostat classique pour contrôler la température d'un lieu. Dès que la

température intérieure atteint les 22,5 ºC, le système se met en route et continue à fonctionner jusqu'à ce que la température soit descendue sous les 21,5 ºC. Il y a donc environ 1 ºC d'hystérésis dans ce modèle. La plage correspondante est la plage d'hystérésis. Dans l'exemple, cette plage est bien trop étroite, ce qui fait arrêter et démarrer le système trop souvent dans l'après-midi.

Chaque séquence allumage/extinction s'appelle un cycle. La Figure 12.3 montre deux modes de fonctionnement qui se distinguent par la largeur de la plage d'hystérésis, c'est-à-dire les deux valeurs limites Hmax et Hmin. La température s'écarte moins du point de consigne avec une plage d'hystérésis étroite, mais cela fait beaucoup plus travailler l'équipement. En théorie, il faudrait prendre la plage la plus large possible, car chaque arrêt et redémarrage du système le rapproche du moment où il tombera en panne (sans compter le surcoût en consommation électrique).

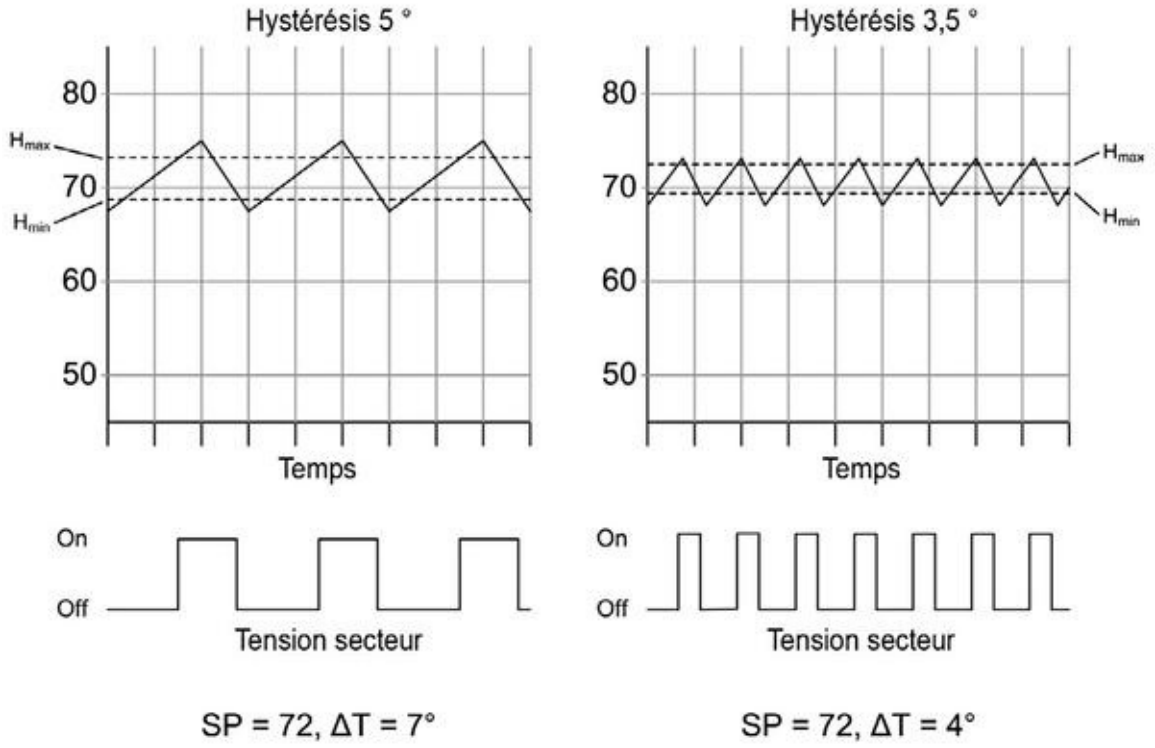


Figure 12.3 : Effet de la largeur de la plage d'hystérésis.

La figure de gauche est plus économe en énergie et fatigue moins l'équipement que celle de droite, mais elle laisse la température réelle s'écarter plus amplement du point de consigne.

Parmi les facteurs qui jouent sur la fréquence de mise en route du système, un des plus évidents est l'efficacité énergétique du système, c'est-à-dire la vitesse à laquelle il rejoint le point de consigne par chauffage ou rafraîchissement. Il faut également tenir compte de l'humidité et du déplacement de l'air intérieur. Lorsque l'humidité augmente, l'air peut plus se charger en chaleur, ce qui favorise le chauffage. Un climatiseur joue un rôle de déshumidificateur : il condense et élimine l'humidité qui se trouve dans l'air qui passe par l'appareil. Un air sec élimine l'humidité résiduelle plus vite, ce qui permet de favoriser le rafraîchissement. En revanche, un air qui ne circule pas lorsque le climatiseur est éteint favorise l'apparition de points chauds et de points froids. Il arrive que le thermostat ne régule qu'une partie de l'espace si l'air n'est pas assez brassé.

Dans l'idéal, le système de contrôle devrait pouvoir mesurer la température dans tous les coins du bâtiment, mais c'est rarement le cas en réalité. Il y aura toujours une pièce qui reçoit la lumière directe du soleil quasiment toute la journée, alors que d'autres pièces restent à l'ombre ou ne permettent pas une bonne circulation de l'air. Autrement dit, même si le thermostat fait de son mieux dans le voisinage de son installation, d'autres endroits dans le bâtiment ne sont pas correctement chauffés ou rafraîchis. Une bonne circulation forcée de l'air peut réduire l'impact de ce problème. En mesurant l'humidité, on peut plus aisément savoir quand il faut mettre en route la ventilation, le chauffage ou le rafraîchissement. Notez qu'il est beaucoup moins coûteux de faire tourner le ventilateur seul, sans les deux autres fonctions.

Contrôle de température intelligent

Le thermostat intelligent a un double objectif : faire des économies d'énergie et garantir une température mieux répartie dans le bâtiment. Un des moyens disponibles est le réglage de la plage d'hystérésis, ce qui modifie la durée des cycles. Les autres techniques consistent à ne faire fonctionner le système que lorsque le domicile est occupé, à modifier le point de consigne en fonction du moment de la journée ou du jour de la semaine et tirer mieux profit de la fonction ventilation du système.

Les thermostats numériques récents sont capables d'ajuster le temps de cycle, de basculer automatiquement entre chauffage et rafraîchissement, et de définir des profils sur une semaine. Passons ces possibilités en revue afin de savoir lesquelles pourraient s'avérer intéressantes pour notre projet.

Le temps de cycle ajustable permet de réduire les mises en marche et arrêts trop fréquents du système. L'approche consiste à augmenter la largeur de la plage d'hystérésis pour que le système se mette en route moins souvent lorsque la température réelle s'éloigne à peine du point de consigne. En revanche, la plage d'hystérésis est réduite lorsque la température fluctue plus amplement et plus vite.

La fonction d'inversion automatique permet au système de passer du chauffage au rafraîchissement selon les besoins. Lorsque le local est situé dans un endroit où il fait chaud le jour et froid la nuit, le thermostat saura automatiquement basculer entre chauffage et rafraîchissement pour maintenir la température demandée. Notre projet va être doté des fonctions de réglage du temps de cycle et de réversion automatique.

Un des principaux avantages des thermostats intelligents est la possibilité de créer des profils. C'est ainsi que vous pouvez programmer un profil pour chauffer ou rafraîchir moins la journée pendant les jours de la semaine, lorsque les lieux sont vides. Une autre technique consiste à réduire un peu le chauffage ou le rafraîchissement la nuit, lorsque tout le monde dort. Notre projet va permettre la création de plusieurs profils pour la journée, la nuit et le week-end.

Les thermostats numériques du commerce se présentent en général sous un très bel aspect avec une face avant soignée et un afficheur numérique sophistiqué, mais ce n'est que du marketing. Vous pouvez aussi bien installer le vôtre dans une boîte de dérivation grise d'électricien. La différence d'aspect n'aura aucun effet sur les capacités fonctionnelles et sur la facilité d'utilisation. Vous pouvez donc faire des économies de temps et d'argent avec un boîtier minimaliste, et celui que j'ai choisi l'est vraiment.

Objectifs du projet

Le premier objectif du projet est de remplacer un thermostat traditionnel de maison. Vous trouverez sur le Web de nombreux projets détaillés basés sur une carte Arduino, et notre projet ne s'en écarte pas vraiment. Il n'y a pas tant de possibilités de combiner un circuit Arduino, un capteur de température, un afficheur et deux relais. En revanche, ce projet se distingue par l'utilisation d'un capteur d'humidité et la possibilité de mettre en route la fonction de ventilation de façon indépendante, afin de déplacer l'air qui est à la bonne température sans avoir besoin aussi souvent de mettre en route le chauffage ou le rafraîchissement.

Notre projet va être capable d'exploiter la température intérieure et son humidité pour régler si nécessaire le temps de cycle. En option, vous pourrez ajouter un capteur de température et d'humidité extérieures. Le projet utilise des relais pour piloter en 24 V la plupart des modèles de climatiseurs domestiques. Il n'y a donc pas de risque d'électrocution. Le seul risque est de programmer un cycle trop rapide qui brutaliserait le compresseur ou le système de chauffage.

Le montage du projet est très simple. Il n'y aura que très peu de soudure à prévoir. Les parties un peu délicates sont la mise en place dans un boîtier et surtout l'écriture du programme.

J'ai conçu le thermostat pour qu'il vienne en remplacement direct d'un thermostat classique à quatre fils et basse tension, mais il n'est pas limité à quatre fils. Vous pouvez tout à fait ne contrôler que le chauffage ou que le rafraîchissement en ne branchant pas toutes les sorties et en réalisant quelques retouches au logiciel.

Définition et planification

Les informations des pages précédentes nous permettent de déterminer les principales fonctions que doit posséder notre projet. Il s'agit des fonctions déjà fournies par un thermostat quatre fils classique et de quelques capacités supplémentaires permises par la détection de l'humidité :

- Une horloge temps réel
- Un capteur d'humidité intérieure

- Un capteur de température intérieure
- Un contrôle automatique du chauffage et du rafraîchissement
- Un contrôle automatique de la ventilation
- Une programmation hebdomadaire

Le projet n’est à aucun moment, ni aucun endroit, en contact avec la tension alternative du secteur, qui est utilisée dans un tel système. Il est prévu pour les systèmes fonctionnant à basse tension. Les 220 V du secteur peuvent endommager l’équipement, mettre le feu à votre domicile, ou vous tuer, et pas nécessairement dans cet ordre. Si c’est le type de système dont vous disposez, vous devez adopter un contrôleur professionnel normalisé et le faire installer par un technicien agréé en climatisation.

Notre contrôleur se fonde sur un circuit Arduino Nano installé sur une carte bouclier de prototypage avec borniers. Nous utilisons un module d’horloge en temps réel RTC pour la programmation hebdomadaire. Le circuit imprimé de l’Arduino va être monté sur le couvercle du boîtier. Un module à quatre relais sera installé en fond de boîtier. Les liaisons avec l’extérieur vont passer par un trou dans le fond.

Conception

Je rappelle que ce thermostat n’est pas prévu pour les pompes à chaleur, ni pour les systèmes de climatisation multiples. Il est destiné à remplacer un thermostat traditionnel comme celui montré dans la Figure 12.1.

Description fonctionnelle

Ce thermostat contrôle trois fonctions : **Heat** (chauffage), **Cool** (rafraîchissement) et **Fan** (ventilation). Les composants principaux sont visibles dans le diagramme de la Figure 12.4. Pour réussir le projet, il faut découvrir comment exploiter correctement les fonctions élémentaires montrées dans le diagramme.

Un encodeur rotatif va servir à régler les différents paramètres : températures, changement de jour dans la semaine, *etc.* Notez que les boutons poussoirs du module afficheur LCD ne sont pas utilisés.

La plupart des anciens systèmes de climatisation n’ont que quatre fils ; certains disposent également de deux fils d’alimentation. Si ces fils sont disponibles dans votre cas, le projet peut tirer profit de cette source de courant alternatif, en ajoutant un redresseur régulateur dans le boîtier. En ce qui me concerne, j’ai utilisé un adaptateur secteur, car mon installation n’avait pas de fils d’alimentation dans le circuit du climatiseur.

Pour mémoire, voici en Figure 12.5 comment câbler un thermostat traditionnel. Les détails varient d’un modèle à l’autre, mais les principes restent les mêmes.

Une caractéristique essentielle des thermostats électromécaniques est que le ventilateur est câblé de telle sorte qu’il se mette toujours en marche dès que l’on passe en mode chauffage ou en mode refroidissement. En effet, il est déconseillé de laisser fonctionner un climatiseur sans la ventilation. Certains systèmes ne démarrent le ventilateur que lorsque la température interne de l’appareil atteint un seuil qui le rend nécessaire.

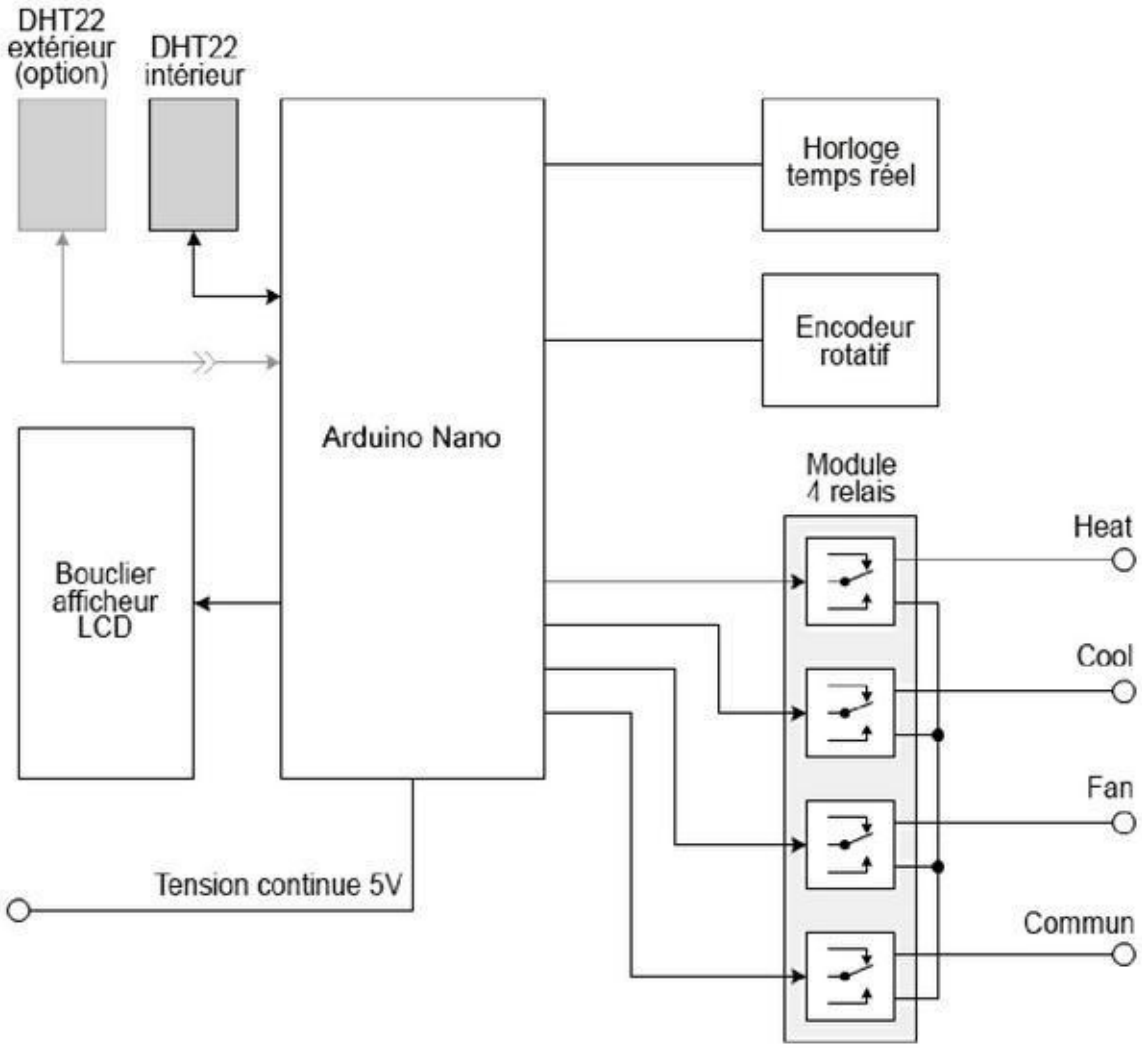


Figure 12.4 : Diagramme du thermostat Arduino.

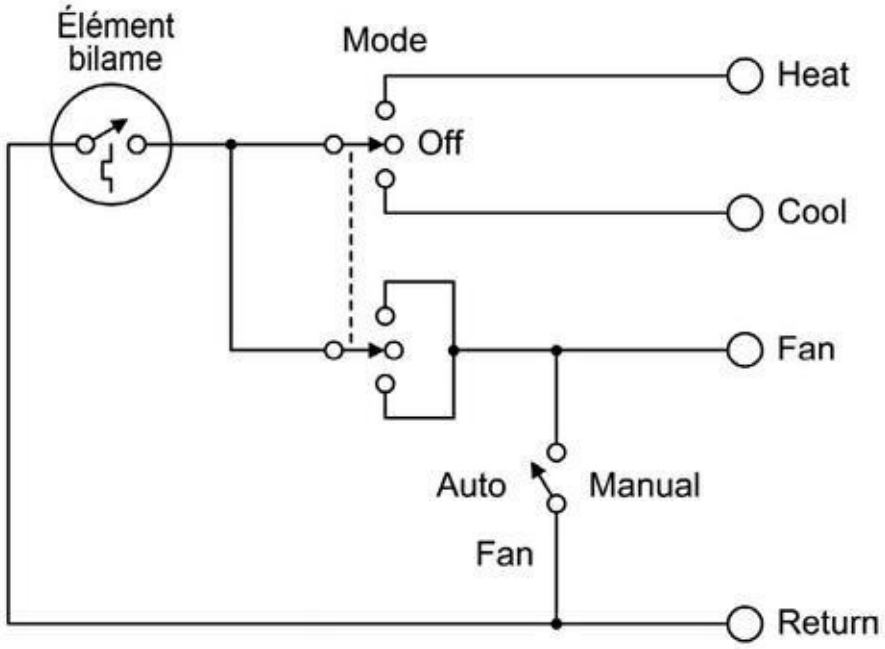


Figure 12.5 : Circuit interne d’un thermostat traditionnel.

Le boîtier

Pour le boîtier, j’ai choisi une boîte de jonction d’électricien prévue pour être montée sur un mur (Figure 12.6). Pour la rendre plus discrète, je vais assurer une finition avec polissage et mise en peinture appropriée.

Figure 12.6 : La boîte de jonction qui va servir de boîtier au thermostat.

Ce boîtier n'est vraiment pas esthétique, mais l'essentiel est qu'il offre assez d'espace pour y placer tous les composants. Je vais même supprimer les deux oreilles de fixation externes et polir le couvercle pour lui donner un meilleur aspect.

Il n'y a que deux composants à prévoir sur la face avant : l'afficheur LCD et l'encodeur rotatif. La disposition du couvercle après préparation est montrée dans la Figure 12.8.

Le module des relais sera installé sur le fond de la boîte. Les connexions vers l'extérieur passeront par un trou dans ce fond de boîte. Le module d'horloge sera placé à l'intérieur, mais le capteur DHT22 sera fixé à l'extérieur sur le dessous de la boîte pour qu'il baigne dans l'air ambiant. Cette solution m'évite de créer des trous dans le boîtier pour permettre une circulation suffisante de l'air extérieur. Une solution intermédiaire consisterait à prévoir un perçage sur les flancs du boîtier pour que le DHT22 puisse être en contact avec l'air extérieur, tout en restant à l'intérieur. Le projet sera fixé contre un mur par quatre vis.

Schéma de principe

Il n'y aura qu'une version du schéma car le prototype et la version finale sont très proches à ce niveau (Figure 12.7). La seule différence est que nous utilisons une carte Arduino Uno au lieu d'une Nano pour le prototype, tout simplement parce que c'est ce type de carte qui est présent sur mon prototype. Les deux cartes embarquent le même microcontrôleur.

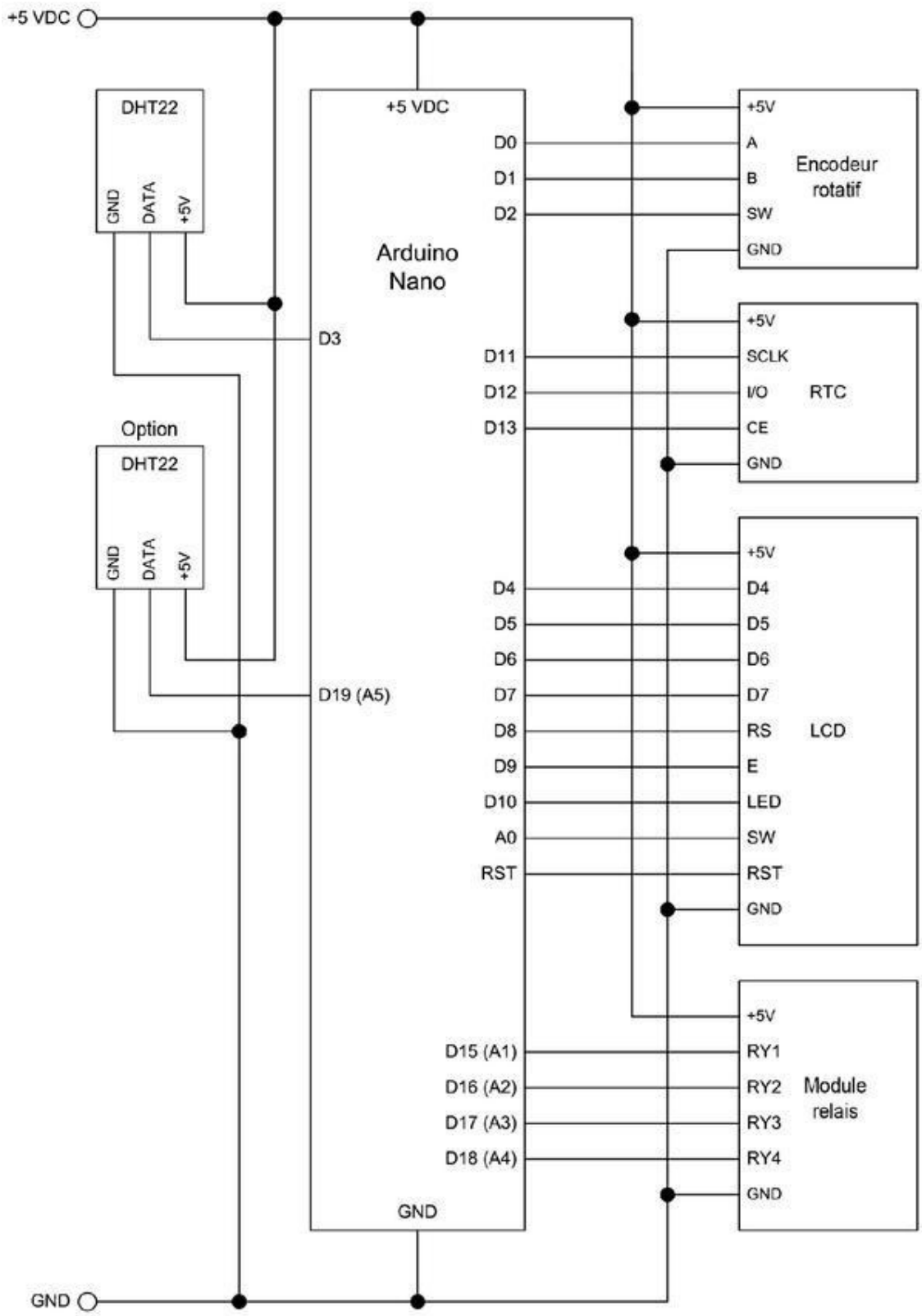


Figure 12.7 : Schéma de principe du thermostat Arduino.

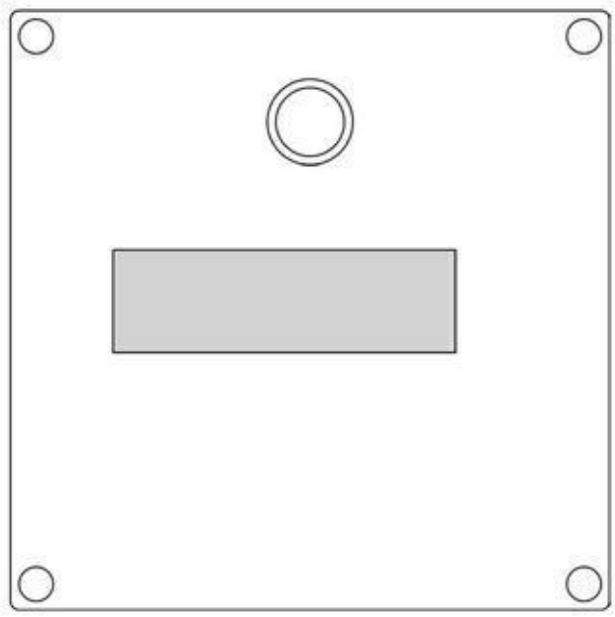


Figure 12.8 : Face avant (capot) du boîtier avec perçages.

Le schéma permet de voir que toutes les broches du microcontrôleur sont utilisées. L'interface de programmation qui correspond aux deux broches D0 et D1 partage celle-ci avec l'encodeur rotatif. Aucun conflit n'est à craindre tant que nous n'utilisons pas l'encodeur pendant que nous téléversons le programme.

Vous remarquez que les broches analogiques d'entrée sont utilisées en tant que broches numériques. Nous avons vu dans les Chapitres 2 et 3 que le port C de l'ATmega328 est simultanément un port d'entrées analogiques et un port d'entrées-sorties numériques. J'utilise les broches analogiques A1 à A5 en tant que broches numériques sous les désignations D15 à D19. Nous n'avons pas besoin des entrées analogiques pour ce projet, sauf la broche A0 qui sert à l'afficheur LCD.

Les conventions d'affectation des broches Arduino n'empêchent pas ces broches d'être utilisées pour d'autres causes. Ces noms correspondent au choix des créateurs. La carte Arduino ne fait qu'étendre l'accès aux broches du microcontrôleur sans rien y changer. C'est le programme qui détermine comment chacune des broches est utilisée, pas le nom attribué à la broche et marqué sur le circuit imprimé.

Logiciel

Le code source de ce projet est assez conséquent, mais l'essentiel concerne la gestion de l'interface utilisateur et celle du profil de programmation. La logique de contrôle n'est pas très complexe.

Je rappelle que le principal objectif du livre consiste à décrire le matériel Arduino, les modules, les capteurs et les composants. Le logiciel n'est destiné qu'à illustrer l'utilisation des composants. Ce ne sont pas des exemples complets et prêts à l'emploi. Tous les fichiers de code source sont disponibles sur le site de l'éditeur ainsi que sur le site de référence GitHub.

Dans la boucle principale, le logiciel va d'abord vérifier si l'utilisateur s'est servi de l'encodeur rotatif. Le code procède ensuite à la mise à jour de la température et de l'humidité puis décide s'il faut basculer l'affichage d'un écran à l'autre, ou basculer en édition des paramètres. Les événements liés à l'encodeur rotatif sont détectés grâce à des gestionnaires d'interruption. Un diagramme fonctionnel du logiciel est disponible dans la Figure 12.9.

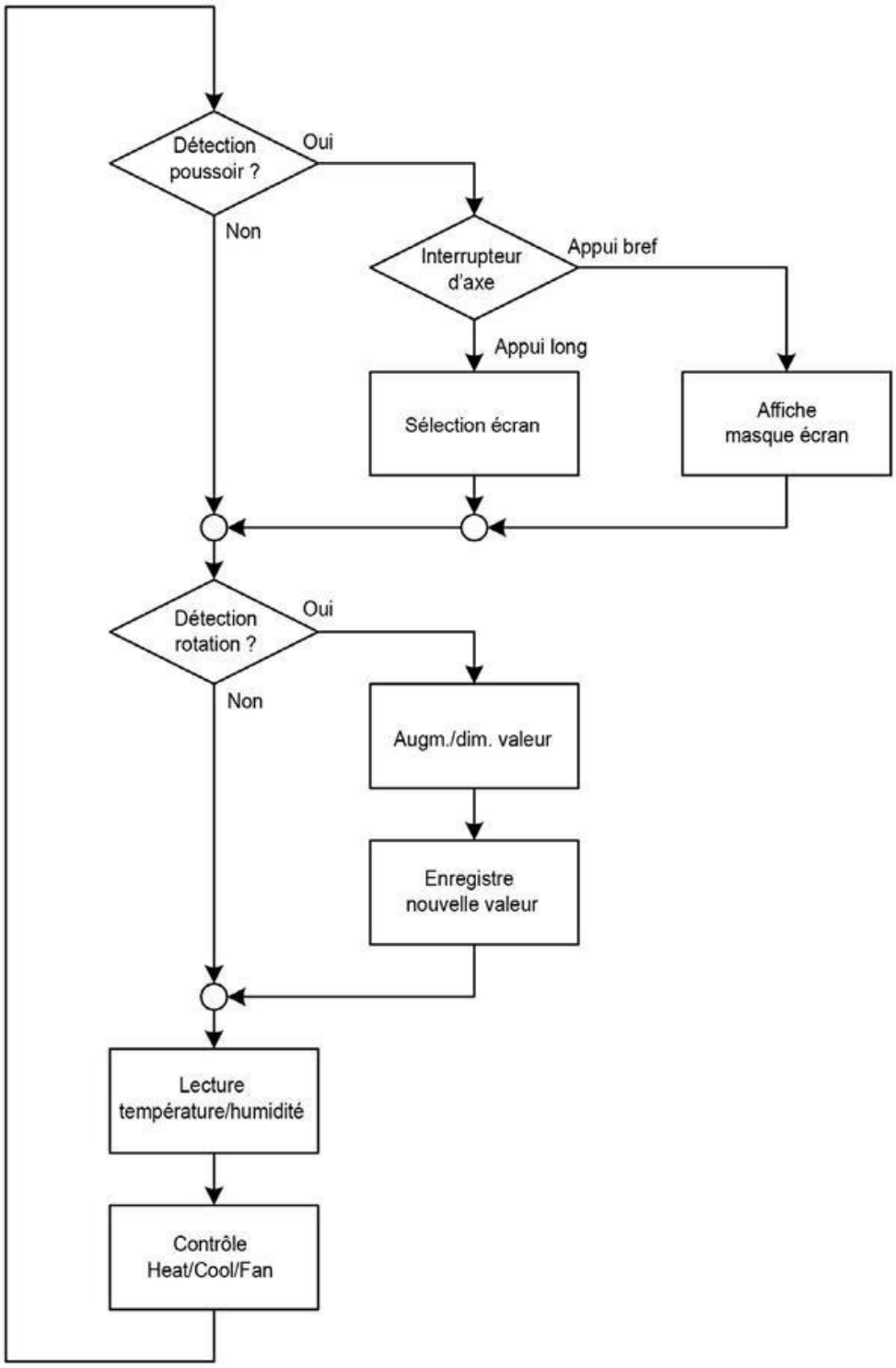


Figure 12.9 : Diagramme fonctionnel du logiciel du thermostat.

Les blocs principaux de cette figure permettent de deviner comment sera structuré le code source. Le fichier principal portera le nom *Thermostat.ino*, il sera accompagné de plusieurs modules pour gérer la saisie d'une valeur par

interruption, pour éditer les profils pour le jour, la nuit et le week-end et pour mettre à jour l'affichage.

Le premier événement à détecter est l'appui sur l'axe du bouton de l'encodeur rotatif. Cet appui signifie que l'utilisateur veut modifier les paramètres. Le bouton-poussoir permet de passer d'un écran à l'autre et de naviguer d'un champ à l'autre dans chaque écran. Le type d'action choisie dépend de la durée d'enfoncement de l'interrupteur.

Les gestionnaires d'interruption qui sont à l'écoute de l'enfoncement du bouton d'axe et des entrées A et B de l'encodeur vont modifier des valeurs servant de drapeaux (définis dans le module des variables globales). Dans la boucle principale, ces valeurs sont testées pour savoir si une action de mise à jour de l'affichage est requise. Notez que les gestionnaires d'interruption ne sont pas montrés dans la Figure 12.9. Nous y reviendrons dans la prochaine section de description du logiciel complet.

Toute la logique de déclenchement des ordres par le thermostat est réunie dans le dernier bloc tout en bas de la Figure 12.9. Je rappelle qu'il s'agit ici d'un contrôleur de type tout ou rien à hystérésis. La Figure 12.10 illustre les variations de température autour du point de consigne en fonction de la plage d'hystérésis. Il s'agit ici du mode rafraîchissement. Pour le mode chauffage, tout est inversé.

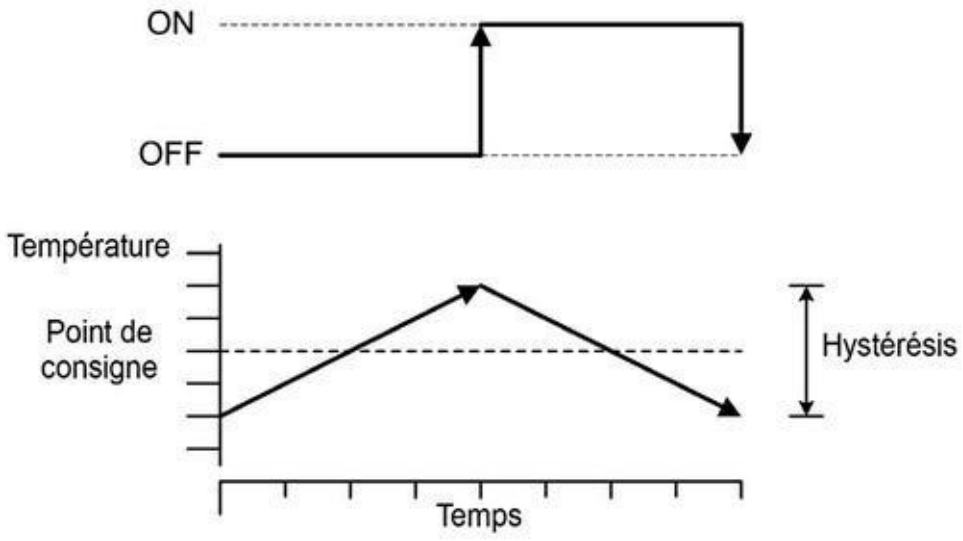


Figure 12.10 : Variation de la température au cours du temps dans la plage d'hystérésis.

La Figure 12.11 donne un autre éclairage sur l'effet de retard à l'allumage et à l'arrêt selon la température. Dans ces deux figures, le constat est le même : un

système de contrôle tout ou rien ne peut jamais maintenir avec précision la température de consigne. Le système sera toujours quelque part entre le minimum et le maximum de la plage d'hystérésis.

Dans la réalité, la vitesse d'évolution de la température va dépendre de l'efficacité de l'apport ou de l'extraction d'énergie. Dans les deux Figures 12.10 et 12.11, la durée du délai montant et descendant semble identique, mais ce n'est quasiment jamais le cas.

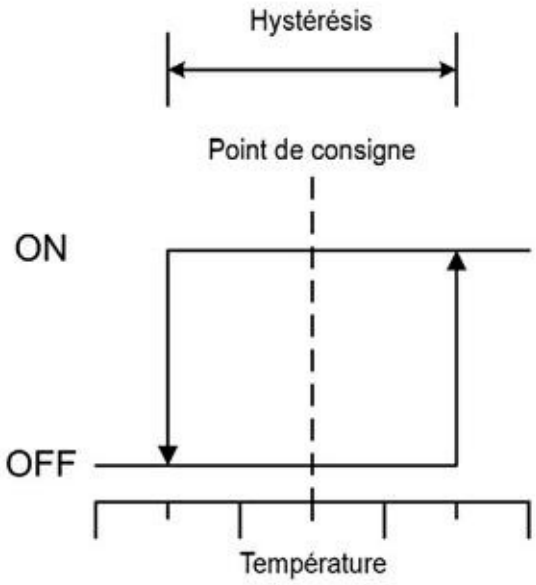


Figure 12.11 : Effet de l'hystérésis sur la réponse du système de contrôle.

La Figure 12.12 présente la logique de fonctionnement du thermostat. Vous pouvez constater que la partie chauffage et la partie refroidissement sont symétriques. Dans les anciens thermostats électromécaniques, la valeur de l'hystérésis, H, était réglée par une vis sur le bilame. Avec un logiciel, nous pouvons choisir ce que bon nous semble comme valeur.

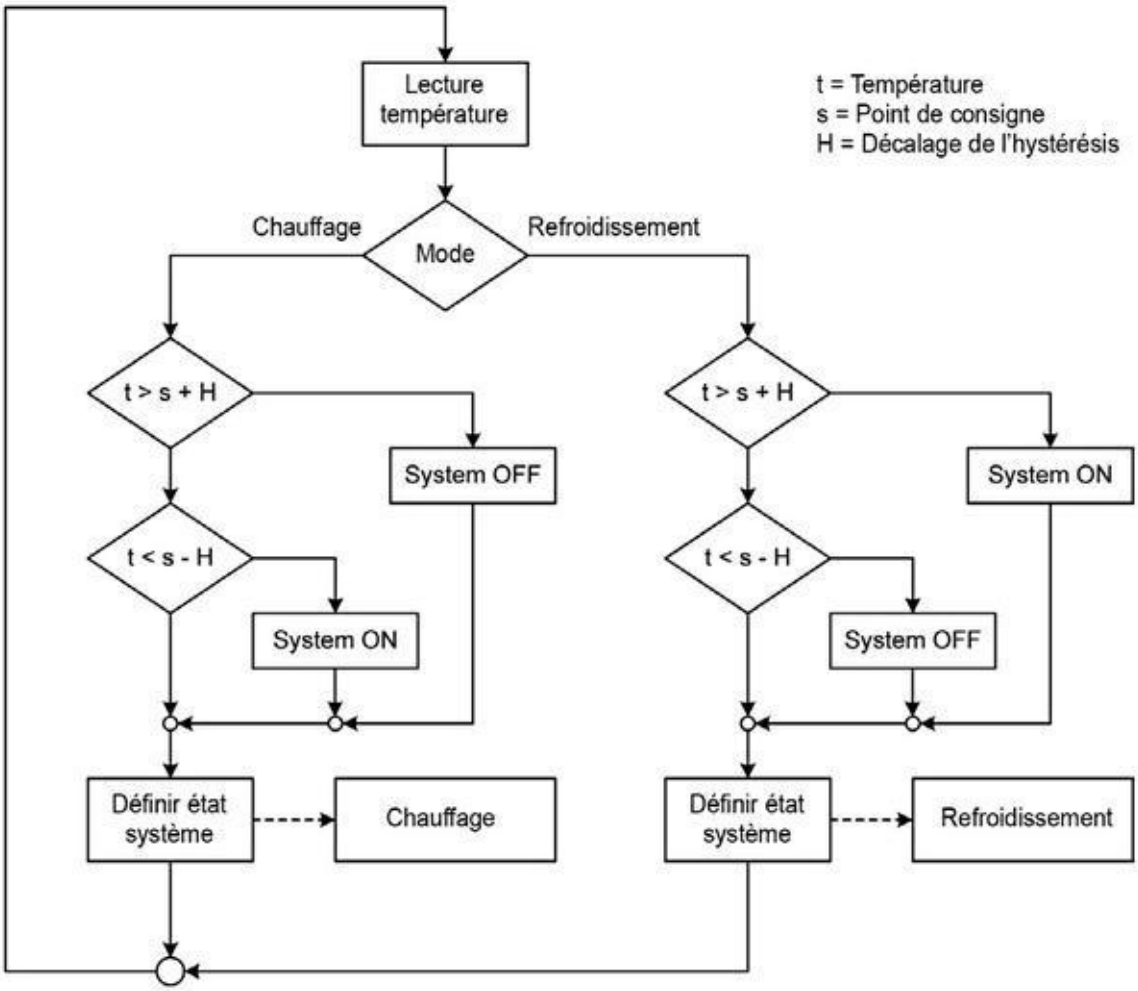


Figure 12.12 : Logique de contrôle du thermostat.

Dialogue utilisateur

Le seul élément d'entrée du montage est l'encodeur rotatif et son interrupteur d'axe. Un appui court sur l'axe correspond à une sélection et un appui long fait basculer vers l'écran suivant.

L'afficheur est le même que celui utilisé dans le Chapitre 11, soit un LCD de deux lignes de 16 caractères. Je me suis simplifié le montage en optant pour un modèle déjà fixé sur une carte bouclier. L'éclairage de fond d'écran LCD n'est allumé que lorsque le dialogue utilisateur est en cours. Les boutons poussoirs de l'afficheur ne sont pas utilisés, mais vous pourriez vous en servir en ajoutant d'autres perçages au capot du boîtier, pour vous en servir comme éléments d'interfaces. Dans ce projet, je ne m'en sers pas.

En temps normal, l’afficheur alterne entre deux écrans. Le premier rappelle les températures mesurées, le point de consigne, le mode de fonctionnement et le profil actif. Le second écran indique la date et l’heure ainsi que la fin de la période en cours dans le profil. Ces deux écrans sont montrés dans la Figure 12.13. Pour montrer qu’une valeur augmente, diminue ou reste stable, j’utilise respectivement trois symboles : accent circonflexe, lettre V et signe moins. Cette tendance est estimée par rapport à la précédente mesure.

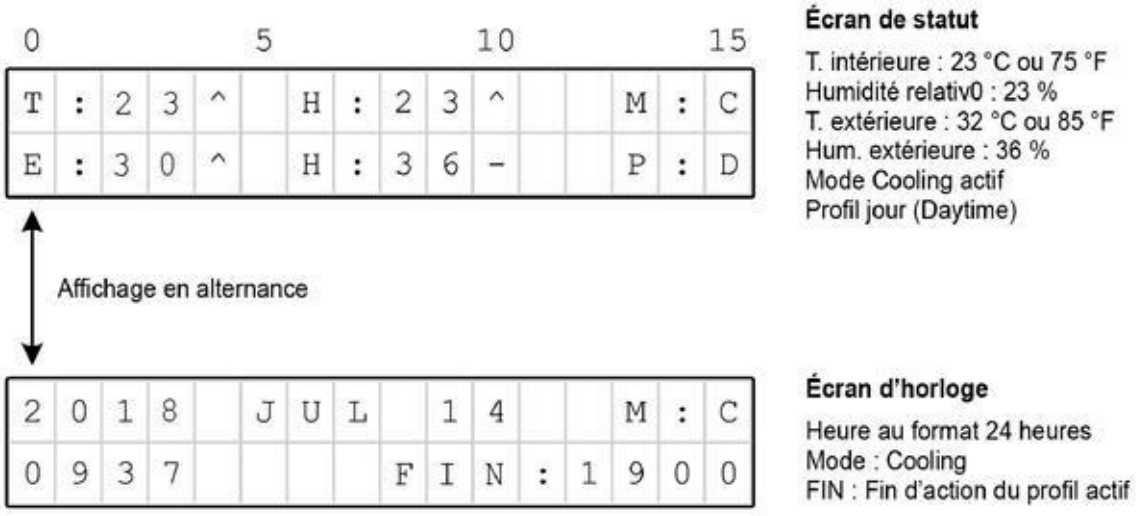


Figure 12.13 : Les deux écrans de l’affichage en régime normal.

Le Tableau 12.1 décrit les différents champs des écrans précédents.

Tableau 12.1 : Champs des deux écrans de l’affichage normal.

Champ	Contenu
T	Température intérieure
E	Température extérieure (si capteur installé)
H	Humidité relative
M	Mode : A = Auto, H = Heat, C = Cool, F = Fan, X = Arrêt
P	Profil : D = Day (jour), N = Nuit, W = Week-end, X = aucun

Les deux écrans précédents ne font qu'afficher des informations et leur contenu n'est pas modifiable. La température est affichée en degrés Celsius ou en degrés Fahrenheit. Par défaut, le programme utilise des Fahrenheit. Le temps est affiché au format 24 heures, mais le format américain est envisageable.

Le troisième écran est celui des paramétrages, Figure 12.14. Il sert à activer ou régler le profil, et à régler la température du point de consigne, changer d'unité, régler la plage d'hystérésis et choisir le mode de fonctionnement.

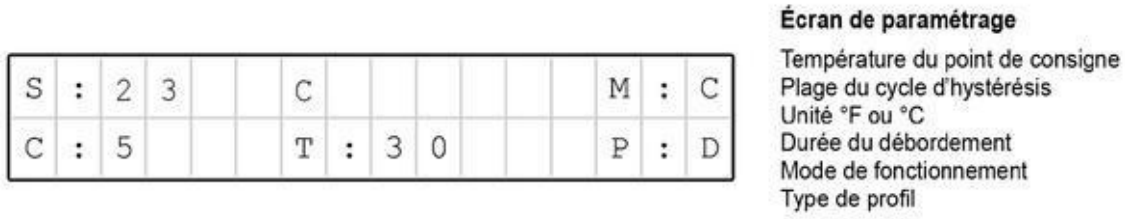


Figure 12.14 : Écran de paramétrage.

Cet écran est le vrai centre de contrôle du thermostat. Il n'est affiché qu'à la demande expresse de l'utilisateur. La signification des champs est fournie par le Tableau 12. 2.

Tableau 12.2 : Champs de l'écran de paramétrage.

Champ	Description
S	Température du point de consigne
C	Largeur de la plage d'hystérésis ou cycle
F	Ou C pour Celsius
T	Durée du débordement (pas O)
M	Mode : A = Auto, H = Heat, C = Cool, F = Fan, X = Arrêt
P	Profil : D = Day (jour), N = Nuit, W = Week-end, X = aucun

Pour automatiser le fonctionnement du thermostat, il faut pouvoir définir un des trois profils. La définition d'un profil fait l'objet d'un quatrième et dernier écran (Figure 12.15).

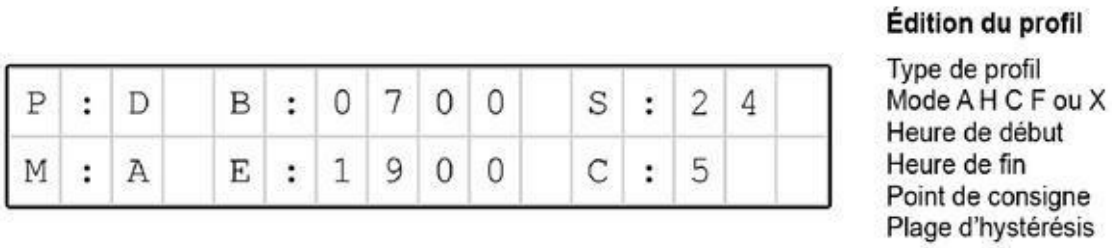


Figure 12.15 : Écran de définition d'un profil.

Vous pouvez définir un profil pour la journée, un autre pour la nuit et un troisième pour les week-ends, correspondant respectivement aux trois symboles D, N et W. Vous passez d'un profil à l'autre lorsque cet écran est affiché avec l'encodeur rotatif. Pour chaque profil, vous définissez une heure de début et une heure de fin. Pour le profil de week-end, l'heure de début est par défaut le vendredi soir à minuit et celle de fin le dimanche soir à minuit. Si l'heure de fin déborde sur l'heure de début d'un autre profil, c'est l'heure de fin qui a priorité. Le Tableau 12.3 décrit les champs de cet écran.

Tableau 12.3 : Champs de l'écran d'édition de profil.

Champ	Description
P	Profil D, N ou W
B	Heure de début (Begin)
E	Heure de fin (End)
S	Point de consigne (set)
C	Largeur de la plage d'hystérésis ou cycle
M	Mode : A = Auto, H = Heat, C = Cool, F = Fan, X = Arrêt

Avec un peu de pratique, vous saurez naviguer aisément parmi les trois états de l'afficheur. Il suffit de savoir faire la différence entre appui bref et appui long sur l'interrupteur de l'encodeur. Il n'est pas inutile de connaître la structure arborescente de la navigation. La Figure 12.16 montre comment cette navigation est réalisée en fonction de la durée de l'appui sur le bouton d'axe.

La figure suivante rappelle les quatre écrans possibles. Un appui est considéré long s’il dure plus d’une seconde et un appui bref s’il dure moins d’une demi-seconde. Le suivi des durées est réalisé par une gestion d’interruption et un compteur logiciel.

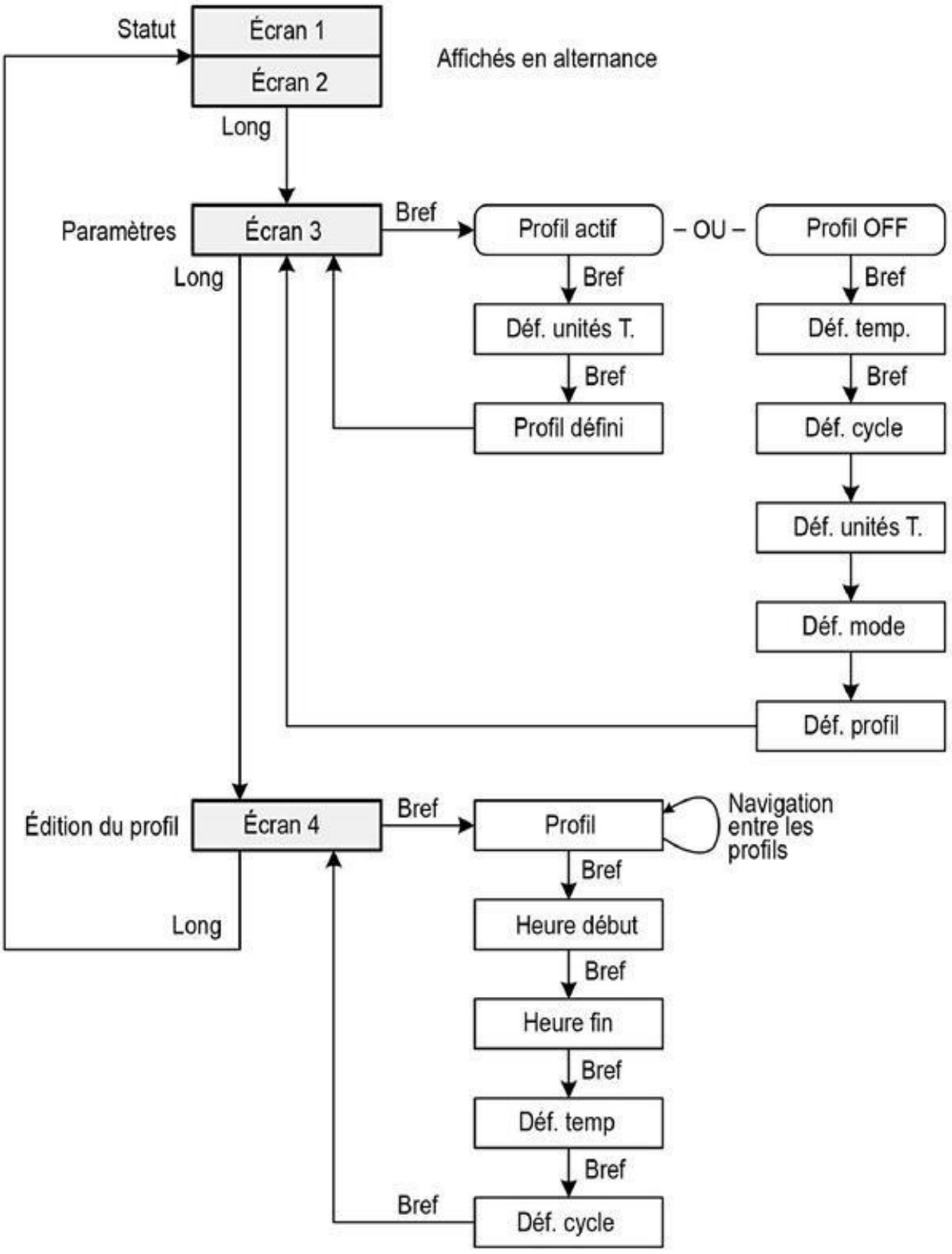


Figure 12.16 : Arborescence de la navigation parmi les écrans du thermostat.

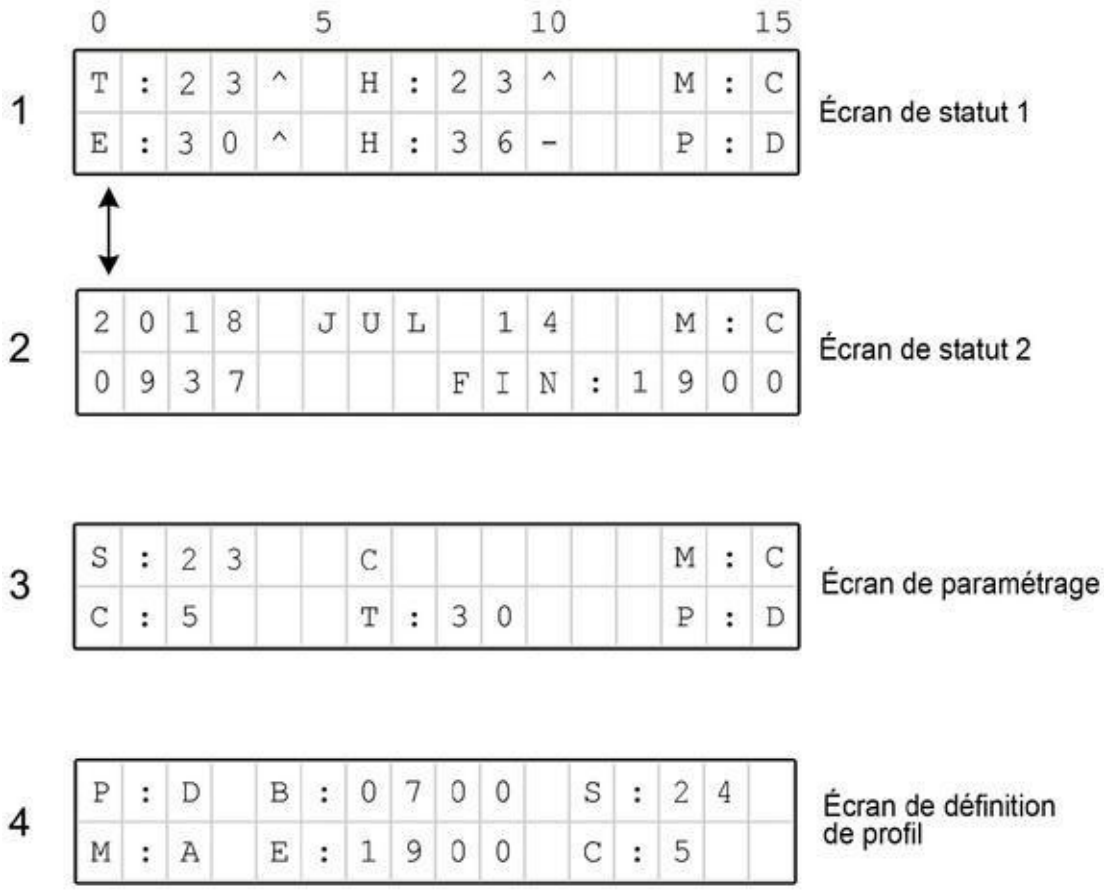


Figure 12.17 : Les quatre écrans que peut afficher le thermostat.

Sorties de commande

Les sorties du projet sont incarnées par un module de quatre relais indépendants du type SPDT (inversion entre un circuit ouvert et un autre fermé). Le module embarque les circuits pilotes et les borniers pour circuit imprimé. La connexion aux fils existants de l'ancien thermostat est donc facile (Figure 12.18). Le commun est relié au quatrième relais dans le coin inférieur droit.

Dans la Figure 12.4 du début du chapitre, vous pouvez voir que la ligne commune du système de climatisation est reliée à un des relais de notre projet. C'est une mesure de sécurité : si le thermostat n'est plus alimenté, le relais coupe ce circuit, et aucune des autres fonctions du climatiseur ne peut être activée. Toutes les lignes de tension de contrôle de l'appareil, y compris le retour commun, sont connectées aux bornes C et NO (normalement ouvert) des relais.

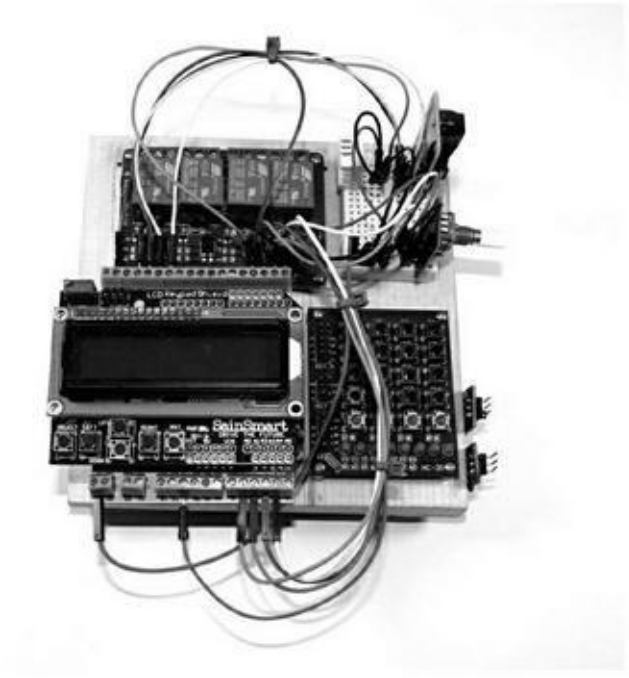


Figure 12.18 : Prototype du thermostat câblé.

Le prototype

Le prototype du projet va utiliser une carte Arduino Uno, deux extensions de connecteurs et un afficheur LCD. Ce sont presque tous les composants que nous allons installer dans la boîte pour la version définitive, à l'exception de l'Uno qui sera remplacée par une Nano (mais les deux utilisent le même microcontrôleur).

En effet, j'ai choisi de réutiliser la carte Arduino du Chapitre 11. J'aurais aussi pu utiliser le Duinokit présenté dans le Chapitre 10. J'ai opté pour cette carte, car elle est suffisamment compacte pour être facilement installée sur le mur à côté du thermostat existant pendant les tests. Le prototype est visible dans la Figure 12.18.

En dehors du modèle de carte Arduino, tous les autres composants resteront les mêmes dans la version définitive. Dans la Figure 12.18, on peut voir l'encodeur rotatif, le module horloge RTC et le capteur DHT22 sur le petit circuit sans soudure. Le tableau suivant dresse la liste des composants du prototype.

Tableau 12.4 : Nomenclature des composants du thermostat prototype.

Quantité	Description

1	Arduino Uno (ou équivalente)
1	Carte afficheur LCD 16 × 2
1	Module horloge DS1302
1	Jeu d’adaptateurs bornier
1	Module quatre relais
1	Capteur temp/humid. DHT22
1	Module encodeur rotatif KEYESKY-040

Le capteur DHT22

J’utilise le même double capteur température/humidité que pour le projet Greenshield du Chapitre 10, c’est-à-dire un DHT22. Dans le montage définitif, il sera relié à la carte Arduino Nano grâce à un bornier. La Figure 12.19 rappelle le brochage du DHT22.

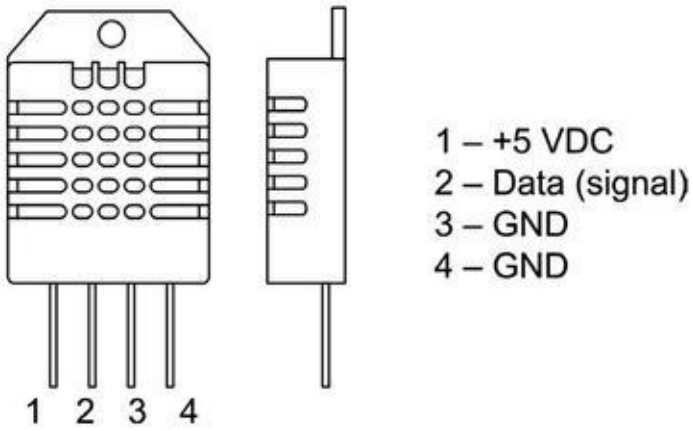


Figure 12.19 : Brochage des sorties du capteur DHT22.

Nous ajoutons une résistance de rappel de 1 kilo-ohm sur le fil de données, comme le conseille le fabricant dans sa fiche technique. Dans la version prototype, le capteur est mis en place sur une petite plaque d’expérimentation sans soudure fixée sur une planche de bois (Figure 12.18).

L’encodeur rotatif

Le dialogue avec l’utilisateur se contente d’un module encodeur rotatif du type KY-040. Il a été décrit dans le Chapitre 9 dans la section consacrée aux modules Keyes. Il utilise un encodeur Bonatech, proche du composant Alps EC11. L’aspect physique du module est visible dans la Figure 12.20 et son brochage dans la Figure 12.21.

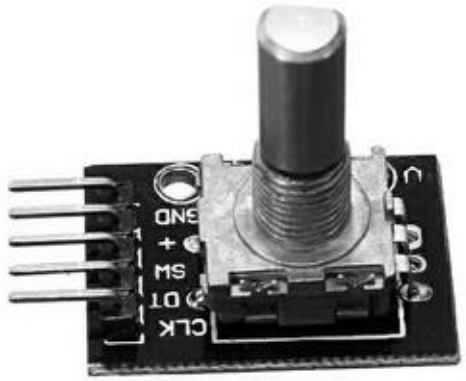


Figure 12.20 : Aspect du module encodeur rotatif.

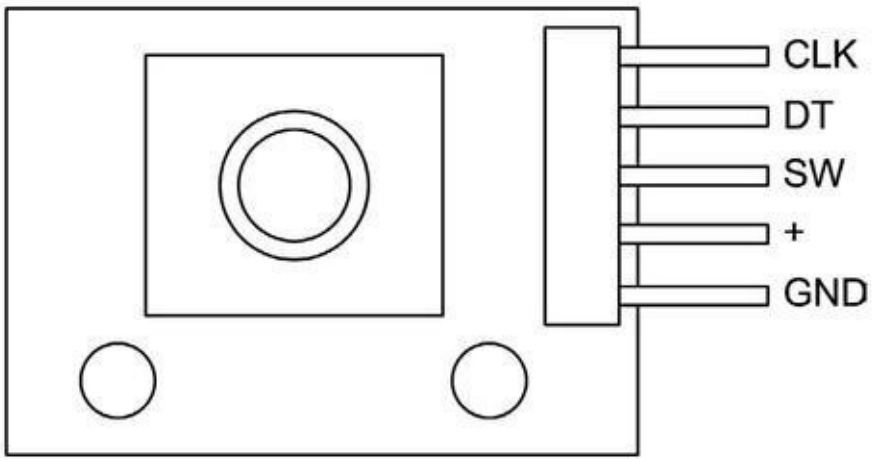


Figure 12.21 : Brochage des sorties du module encodeur rotatif.

Le circuit de ce module est fort simple, comme le montre la Figure 12.22. L’interrupteur est fermé lorsque l’axe de rotation est poussé. Chacune des trois lignes de signal utilise une résistance de rappel de 10 kilo-ohms. Les deux broches CLK et DT correspondent aux broches de signal A et B que mentionnent la plupart des exemples d’utilisation d’un encodeur rotatif à quadrature simple.

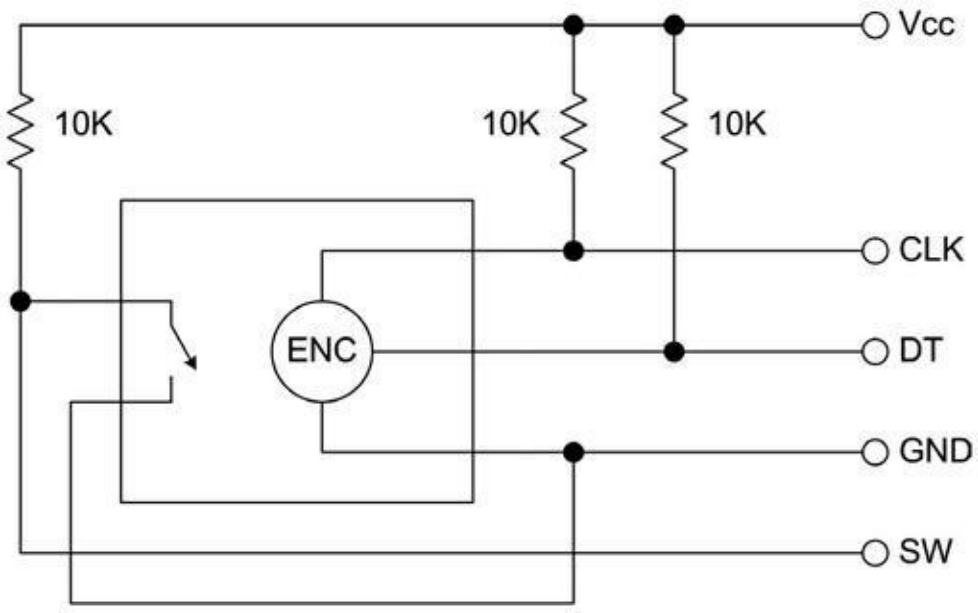


Figure 12.22 : Schéma de principe d'un encodeur rotatif.

Comme déjà indiqué, l'interrupteur permet de passer d'un affichage à l'autre et d'éditer certains paramètres. Cet interrupteur est de type normalement ouvert, NO.

Le module horloge temps réel (RTC)

Pour que l'utilisateur puisse définir des profils, le thermostat doit savoir à tout moment quelle heure il est. Nous utilisons à cet effet un petit module DS1302 qui est une horloge temps réel RTC (Real Time Clock). Le module se contente de réunir sur un circuit le circuit intégré DS1302, un cristal oscillateur et une pile bouton (Figure 12.23). Le brochage est évident.

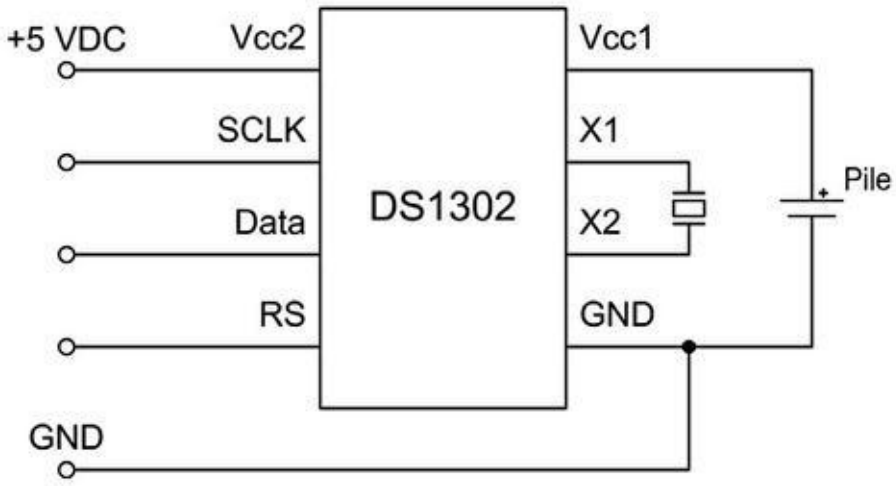


Figure 12.23 : Schéma de principe du module d'horloge RTC.

Pour utiliser ce module, il faut se procurer la librairie de fonctions appropriée, par exemple sur le site Arduino Playground. Je vais bien sûr profiter de cette librairie au lieu de perdre du temps à écrire moi-même les fonctions requises.

L'afficheur LCD

L'afficheur à cristaux liquides a été décrit dans le Chapitre 8. Il est visible dans la Figure 12.18. C'est ce module qui s'accapare le plus grand nombre de broches d'entrée-sortie du processeur. Nous avons choisi un modèle doté d'un potentiomètre de réglage du contraste et d'un transistor pour l'éclairage du fond d'écran. Ce module s'enfiche directement dans la carte Arduino, ce qui évite le spaghetti de fils que vous avez pu voir lors du montage du générateur de signal dans le Chapitre 11.

Le code source

Le logiciel de notre thermostat est assez simple, mais il se distingue des autres exemples du livre par son utilisation des interruptions afin de détecter l'appui sur le bouton de l'encodeur. Comme tous les programmes destinés à être embarqués sur un microcontrôleur, la boucle principale s'exécute sans cesse jusqu'à la mise hors tension. Rassurez-vous : la mise en place d'un gestionnaire d'interruption pour Arduino n'est pas très complexe, et les interruptions constituent la meilleure technique pour prendre en charge la survenue inopinée d'événements du monde réel.

Structure du code source

Comme les autres projets du livre, le code source est réparti en plusieurs modules que présente le Tableau 12.5. Le module principal définit bien sûr les deux fonctions obligatoires puis vient un module pour les variables globales et un autre pour le contrôle des fonctions de commande de l'appareil climatiseur. Pour le module d'affichage, je réutilise une partie du code source que j'avais écrit pour le générateur de signal dans le chapitre précédent. Le gestionnaire d'interruption de l'encodeur rotatif se trouve dans le module d'interface *tstat-iface.cpp*.

Tableau 12.5 : Modules de code source du thermostat.

Module	Fonction
Thermostat.ino	Module principal avec setup() et loop()
tstat.h	Constantes définies par #define
tstat_ctrl.cpp	Logique de contrôle de la CVC
tstat_ctrl.h	Directives #include
tstat_gv.cpp	Variables globales
tstat_gv.h	Directives #include
tstat_iface.cpp	Gestion des entrées
tstat_iface.h	Directives #include
tstat_lcd.cpp	Afficheur LCD
tstat_lcd.h	Directives #include
tstat_util.cpp	Utilitaires
tstat_util.h	Directives #include

Description fonctionnelle

Le logiciel peut être divisé en trois parties principales : gestion de l'interface utilisateur, logique de contrôle et gestion de l'affichage. Le premier bloc qui correspond au module *tstat_iface. cpp* se charge de lire l'encodeur rotatif pour naviguer parmi les écrans. Le module *tstat_lcd. cpp* contient les fonctions d'affichage et le module *tstat_ctrl.cpp* la détection de la température et d'humidité en provenance du capteur DHT22, la logique de contrôle et les différents profils définis par l'utilisateur.

Plusieurs librairies de fonctions sont mises à profit dans ce projet : pour le module d'horloge DS1302, pour l'encodeur rotatif, pour les fonctions de date et d'heure et pour le premier timer du microcontrôleur. Ces librairies sont listées dans le Tableau 12.6. C'est un des grands avantages de l'univers Arduino : dès que vous avez besoin d'une librairie pour pouvoir exploiter un module, un capteur ou un bouclier, il y a de fortes chances que quelqu'un, quelque part, ait déjà pris le temps de créer cette librairie et de la mettre à disposition de tous.

Tableau 12.6 : Librairies complémentaires du thermostat.

Nom	Fonction	Auteur
Time	Heure	Michael Margolis
DS1302RTC	Classe RTC	Timur Maksimov
ClickEncoder	Encodeur rotatif	Peter Dannegger
TimerOne	Classe du Timer 1	Lex Talionis

Pour télécharger ces librairies, rendez-vous sur le site officiel Arduino Playground (`playground.arduino.cc`). Prenez le temps de lire la documentation associée et d'étudier le code source des exemples et de la librairie. Vous saurez ainsi comment bien configurer ces librairies. Notez que la librairie DHT22 est la même que celle utilisée pour le projet Greenshield du Chapitre 10.

Le fichier d'en-tête *tstat.h* réunit toutes les définitions globales que vont utiliser les autres modules. Vous y trouvez notamment les affectations des broches du microcontrôleur comme montré dans la Figure 12.7 et dans le Listing 12.1.

Listing 12.1 : Définitions globales dans tstat.h.

```
#define ROTENC_A 0
#define ROTENC_B 1
#define ROTENC_SW 2

#define LCD_D4 4   // Prédéfinies par le bouclier LCD
#define LCD_D5 5   //
#define LCD_D6 6   //
#define LCD_D7 7   //
#define LCD_RS 8   //
#define LCD_E 9    //
#define LCD_LED 10 //
#define LCD_SW A0  //

#define RTC_SCLK 11
#define RTC_IO   12
#define RTC_CE   13

#define RY1 15 // A1
#define RY2 16 // A2
#define RY3 17 // A3
#define RY4 18 // A4

#define DHT1  3  //  DHT22Interne
#define DHT2 19  //  A5, DHT22 externe
```

Le Listing 12.2 montre le code source de la fonction de configuration initiale `setup()` du fichier *Thermostat.ino*. Elle prépare l'afficheur LCD, affiche quelques messages, teste le module d'horloge puis affiche le premier écran de travail.

Listing 12.2 : Fonction setup().

```
void setup()
{
  lcd->begin(16, 2); // Format du LCD
  TitleDisp("Initializing...", "", 1000);
  lcd->clear();

  if (rtc->haltRTC())
    lcd->print("Clock stopped!");
  else
```

```
    lcd->print("Clock working.");

  lcd->setCursor(0,1);
  if (rtc->writeEN())
    lcd->print("Write allowed.");
  else
    lcd->print("Write protected.");

  delay (2000);

  // Prépare librairie Time
  lcd->clear();
  lcd->print("RTC Sync");
  setSyncProvider(rtc->get); // Lecture de l'heure du RTC
  lcd->setCursor(0,1);
  if (timeStatus() == timeSet)
    lcd->print(" Ok!");
  else
    lcd->print(" FAIL!");

  delay (1000);
  TitleDisp("Initialization", "complete", 1000);
  curr_screen = 0;
  Screen1();
  disptime = millis();
}
```

La boucle principale loop(), qui se trouve dans le même fichier source est très simple, comme le montre le Listing 12. 3. L'essentiel du programme se situe dans la partie de paramétrage avec l'encodeur rotatif et dans la collecte des données du capteur, sans oublier l'exécution des fonctions de commande dans *tstat_ctrl.cpp*.

Listing 12.3 : Fonction loop().

```
void loop()
{
  // Lecture date et heure du RTC
  RTCUpdate();

  if (input_active) {
```

```
      HandleInput();
    }
    else {
      // Alterne entre écran 1 et 2
      if ((millis() - disptime) > MAX_DISP_TIME) {
        if (curr_screen) {
          Screen1();
          curr_screen = 0;
          disptime = millis();
        }
        else {
          Screen2();
          curr_screen = 1;
          disptime = millis();
        }
     }
    }
    GetTemps();
    SystemControl();
  }
```

La lecture du code source montre que lorsque le système n'est pas en mode paramétrage, il se contente de faire afficher tour à tour les deux écrans (affichage des conditions et affichage de l'heure). Dès que le poussoir d'axe de l'encodeur est sollicité, la variable *`input_active`* prend la valeur true et la conserve jusqu'à ce que l'utilisateur soit revenu à l'écran 1 en appuyant à nouveau sur le même interrupteur. Vous remarquez que l'appareil de climatisation reste sous contrôle, même pendant la session de paramétrage.

Test du thermostat

Cette section s'applique autant au prototype qu'à la version finale. Nos tests ont comporté trois étapes. Dans la première, nous réglons la date et l'heure dans le module RTC en nous servant du bouton de l'encodeur. Une fois revenu à l'écran principal, nous confirmons que l'heure est réglée. La deuxième étape suppose de régler la température de consigne, la plage d'hystérésis (le paramètre C) et le mode de fonctionnement. J'ai commencé par vérifier que je parvenais à activer à la demande le ventilateur, puis j'ai saisi un point de consigne et le mode. En surveillant l'affichage, vous devez constater que le système arrête de chauffer ou

de rafraîchir dès qu'il a atteint le point de consigne plus ou moins la moitié de la plage d'hystérésis.

La dernière étape consiste à définir ou programmer les trois profils en fonction de vos besoins. Je rappelle que si l'heure de fin d'un profil déborde sur le début du suivant, c'est l'heure de fin qui a la priorité.

Observez le fonctionnement de l'ensemble sur plusieurs semaines. Vous aurez peut-être besoin de faire quelques ajustements et de revoir le point de consigne selon le profil. Si vous avez construit le projet Greenshield, vous pouvez vous en servir pour collecter des données de température et d'humidité afin de bien analyser le fonctionnement du thermostat. Un petit script en langage Python devrait suffire pour demander au module Greenshield de recueillir les données de façon périodique.

Montage de la version finale

La version finale n'a pas le même aspect que le prototype, mais il n'y a aucune différence en termes de fonctions et de connexion. En dehors du boîtier, les deux grandes différences sont l'utilisation d'une carte Nano et d'un afficheur LCD nu, et non sous forme de bouclier. Le Tableau 12.7 liste les composants de cette version finale.

Tableau 12.7 : Nomenclature des composants de la version finale du thermostat.

Quantité	Désignation
1	Arduino Nano (ou équivalent)
1	Carte prototype à borniers
1	Afficheur 16 × 2 LCD
1	Module quadruple relais
1	Capteur DHT22
1	Encodeur KEYESKY-040
1	Module horloge

Montage

Le montage consiste à mettre en place les composants sur le fond de la boîte et sous le capot. Il faut d'abord préparer ce boîtier en réalisant les deux perçages du capot, un circulaire et l'autre rectangulaire pour l'encodeur et l'afficheur. Faisons également un trou dans le fond du boîtier pour les fils de commande et quatre petits trous pour fixer le boîtier au mur. Après ébavurage, j'ai appliqué une peinture de ton ivoire pour la finition.

La carte Nano est installée sur un bouclier de prototypage à borniers, les broches du Nano étant connectées à ce bornier. La Figure 12.24 montre le dessous du capot avec la carte qui porte le Nano, ainsi que l'afficheur LCD, l'encodeur rotatif et le capteur.

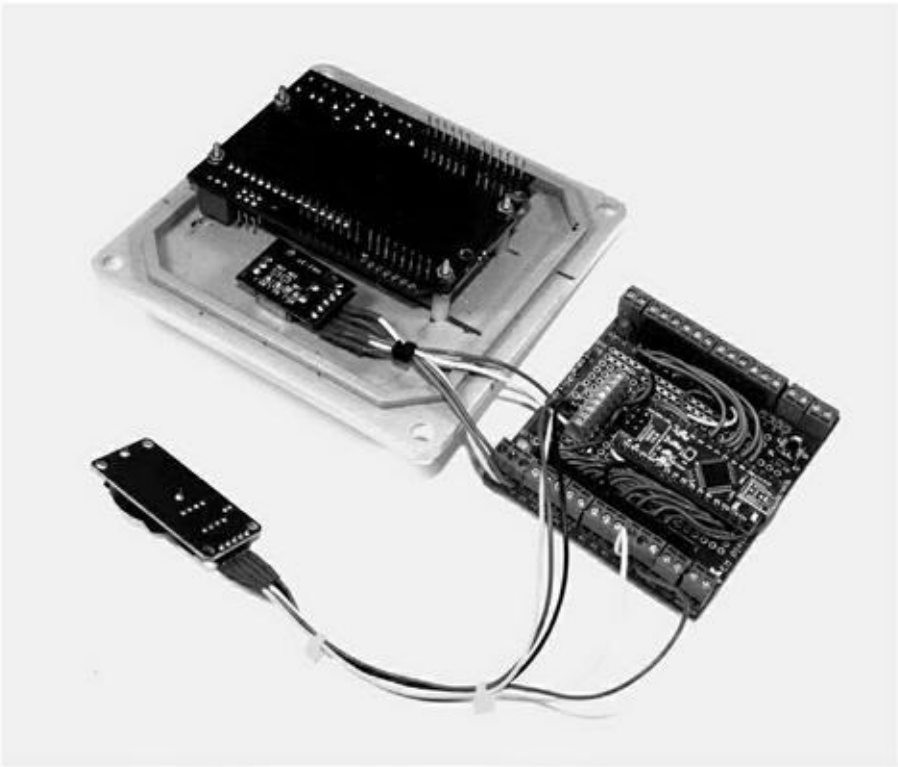


Figure 12.24 : La carte Nano sur son prototype et les deux composants du capot.

Tous les fils volants sont sur le dessus de la carte prototype. Sa face inférieure ne comporte que des soudures. J'ai utilisé le plus petit diamètre de fil de câblage

dont je disposais. Je déconseille d'utiliser du fil de wrapping qui est vraiment trop fin.

L'énorme avantage des cartes boucliers de prototypage est le grand confort qu'elles permettent grâce aux borniers à vis. C'est par ce moyen que nous réalisons les connexions avec le capteur DHT22, le module d'horloge, le module relais et l'encodeur rotatif.

La Figure 12.25 montre l'aspect du projet une fois le boîtier refermé. Nous ne voyons que l'afficheur et l'encodeur. Le capteur DHT22 n'est pas visible, car il est installé contre le flanc masqué.

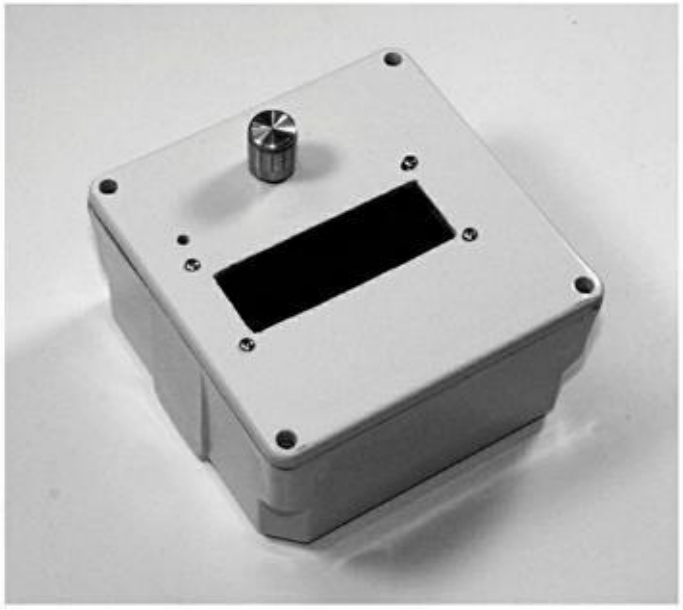


Figure 12.25 : Le projet dans son boîtier presque achevé.

Vous remarquez un petit perçage dans le coin supérieur gauche de l'ouverture pour l'afficheur. Il permet d'accéder au potentiomètre de réglage du contraste de l'écran. Vous pourriez rajouter une feuille de plastique transparent dans l'ouverture de l'afficheur afin de le protéger. Le résultat offre un aspect assez industriel, mais l'habit ne fait pas le moine.

Le module avec ses quatre relais est fixé sur le fond du boîtier, et le module d'horloge sur un des flancs. La Figure 12.26 montre la répartition des composants dans le boîtier terminé. Le câblage en provenant de la climatisation passera par le trou ménagé dans le fond. Vous remarquez que ce trou n'est pas centré, pour éviter que les fils butent contre la plaque du module de relais.

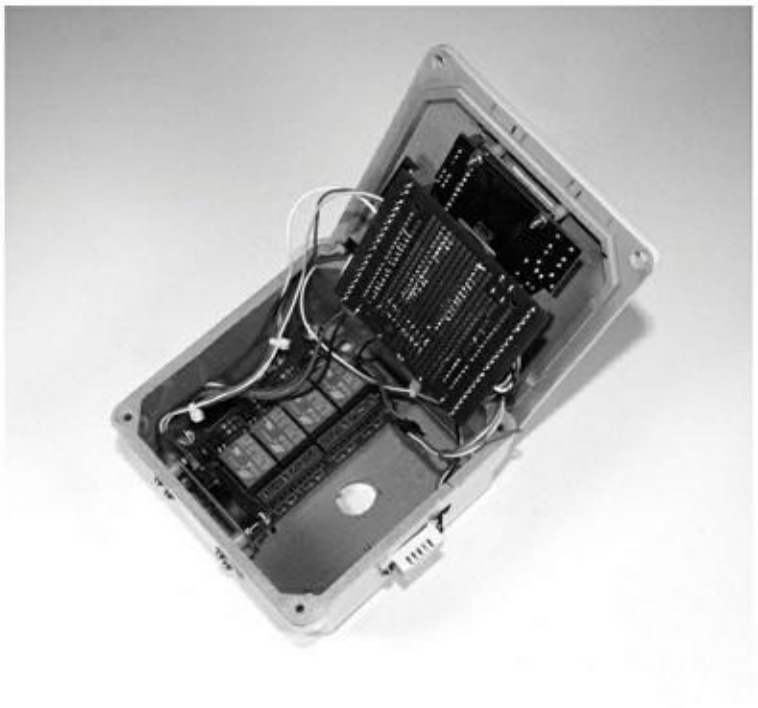


Figure 12.26 : Vue intérieure du boîtier du thermostat.

Tout cela est un peu serré, mais cela tient. Voici les dernières étapes à prévoir avant de refermer le boîtier :

1. Marquer et percer les trous de fixation dans le mur.
2. Tirer les fils de contrôle par le trou du fond et fixer le boîtier au mur.
3. Connecter les fils de commande à l'appareil à contrôler.
4. Brancher le tout à l'alimentation et vérifier l'affichage.
5. Fixer le capot.

Le thermostat devrait être immédiatement opérationnel, si vous n'avez pas oublié de téléverser le logiciel vers le microcontrôleur. Il n'y a pas d'interrupteur d'alimentation. L'éclairage de l'écran devrait rester actif pendant une seconde. Pour le réactiver une fois qu'il est éteint, il suffit de tourner un peu l'encodeur.

Le capot comporte un joint qu'il n'est pas nécessaire de conserver. Il m'a été utile pour gagner un peu de hauteur à l'intérieur, mais je ne cherche pas à rendre le montage étanche.

Tests de mise en route

Les derniers tests consistent à se servir de l'encodeur pour parcourir les différents écrans, en vérifiant notamment que le logiciel distingue bien les appuis longs et les appuis brefs. Il faudra redéfinir tous les paramètres, car le téléversement du logiciel ne restaure pas l'espace mémoire EEPROM, uniquement la mémoire flash du programme.

Commencez par des opérations simples, en vous contentant par exemple de contrôler la mise en route du chauffage et du rafraîchissement. Une fois que ces tests ont réussi, vous pouvez envisager de définir des profils de jour et de nuit. Un peu plus tard, vous pourrez jouer sur les réglages de taux d'humidité et de temps de cycle. Ce genre d'appareil réclame un temps d'adaptation pour bien s'intégrer à votre environnement.

Un second capteur DHT22 ?

Vous vous souvenez que j'avais prévu une entrée pour un deuxième capteur DHT22 que j'ai laissé de côté afin de pouvoir bien me concentrer sur les principes de base. En termes de connexion, l'ajout de ce second capteur est très simple, mais il augmente la complexité du logiciel.

En effet, le thermostat doit dorénavant surveiller également l'environnement extérieur et évaluer la différence entre les mesures à l'intérieur et les mesures à l'extérieur. On peut en déduire un taux de variation au cours du temps et même estimer la perte ou le gain thermique par les murs et les fenêtres. Une programmation soigneuse et ingénieuse permettrait au thermostat de bien arbitrer entre différents critères pour offrir un profil de climatisation optimal. Cela semble très intéressant, mais ce n'est pas simple du tout.

Budget

Un des objectifs du projet était d'utiliser le plus possible des modules disponibles pour gagner du temps et limiter le coût. Un circuit imprimé sur mesure coûte toujours un peu d'argent. De façon globale, le projet a demandé bien moins de travail que le générateur de signal du chapitre précédent. La liste des composants est rappelée dans le Tableau 12.8.

Tableau 12.8 : Nomenclature et prix des composants du thermostat.

Quantité	Désignation	P.U.	Total
1	Arduino Nano	14,00	14,00
1	Bouclier prototype à borniers	15,00	15,00
1	Module RTC	2,00	2,00
1	Encodeur rotatif	6,00	6,00
1	Bouclier LCD	10,00	10,00
2	Capteur DHT22	10,00	20,00
1	Module 4 relais	7,00	7,00
1	Boîtier	7,00	7,00

Coût total du projet : environ 80 euros.

Les encodeurs rotatifs sont disponibles nus ou sous forme de modules, comme c'est le cas du module KY-040. Dans la mesure du possible, optez plutôt pour un module.

En optimisant vos approvisionnements, vous pourrez sans doute réduire le coût total du projet.

Pour aller plus loin

Ce projet illustre bien les contraintes qui surviennent dès que l'on veut réaliser un équipement complexe uniquement à partir de boucliers et de modules du commerce. Si je devais créer une nouvelle version du même thermostat, je déciderais de créer un circuit imprimé spécifique sur lequel j'aurais rassemblé tous les composants. Les relais seraient sur ce circuit, comme dans le projet Greenshield du Chapitre 10, et j'y réserverais de la place pour l'alimentation.

L'afficheur LCD tirerait lui aussi profit d'une amélioration en y ajoutant un multiplexeur du type MCP23017, comme celui utilisé par le projet Switchinator. Certains fournisseurs proposent l'afficheur avec le circuit MCP23S17, la version SPI du même circuit.

Dans une nouvelle version, j'utiliserais sans doute des composants montés en surface CMS dès que possible. Les relais pourraient être d'un modèle un peu plus petit, car ils n'ont pas à fournir un courant important. La taille du circuit imprimé pourrait ainsi être sérieusement réduite.

Parmi les fonctions qu'il serait intéressant d'ajouter, il y a notamment une interface de commande à distance Bluetooth ou Wifi, un lecteur de carte mémoire microSD pour collecter les données et quelques diodes LED en façade. Dans sa version actuelle, ce thermostat est sans le savoir un digne représentant du nouveau monde de l'Internet des objets.

CHAPITRE 13
Une étude fondée sur des exigences

Ce chapitre se distingue des trois précédents car il ne propose pas de réaliser concrètement un projet d'électronique, mais se concentre sur les phases initiales de conception. Nous allons nous baser sur une liste d'exigences, ce qui est la méthode utilisée dans le monde professionnel. Autrement dit, nous n'allons rien construire, mais définir des approches, identifier les outils et les composants que nous pourrions utiliser et réaliser des arbitrages entre besoins contradictoires. Cette approche convient aussi bien au projet relativement simple qui va servir à illustrer ce chapitre qu'à de gros projets complexes.

Vue générale du projet

En guise d’application, je vous propose de concevoir un système de contrôle du déclenchement d’une séquence d’événements, basé sur un compte à rebours. Vous pourrez vous en servir pour déclencher des effets spéciaux dans un spectacle, pour lancer des maquettes de fusées, ou pour contrôler un feu d’artifice. L’aspect physique du projet pourra se résumer à une petite boîte, ou bien faire l’objet d’un véritable pupitre de contrôle, en y ajoutant des boutons poussoirs et des afficheurs supplémentaires. Enfin, le projet pourra être enrichi en le dotant de capacités de télécommande Wifi ou Bluetooth, de contrôle de continuité des lignes, d’un générateur vocal pour le compte à rebours et les comptes-rendus de déclenchement, ou d’un système de sécurité périmétrique. La richesse fonctionnelle du résultat ne dépendra que de vos envies et de votre budget.

À la fin du chapitre, vous saurez quoi acheter, comment écrire le logiciel et comment planifier vos efforts de réalisation. Les chapitres précédents vous ont permis de voir comment établir une nomenclature de composants et un budget d’approvisionnement. Vous savez dessiner un diagramme fonctionnel et un logigramme. Une fois que vous aurez défini ces éléments pour ce projet, il ne restera plus qu’à réunir les composants, effectuer le montage et écrire puis téléverser le logiciel avant de passer aux tests.

Si vous comptez utiliser le projet de ce chapitre pour déclencher des événements dans le monde physique, prenez toutes les précautions pour assurer la sécurité de l’environnement et des personnes. Si vous comptez par exemple déclencher le lancement de fusées miniatures, prenez contact d’abord avec les associations appropriées.

Le cycle de conception

Dès que le projet est un tant soit peu complexe, il est généralement nécessaire de travailler par itération pendant la conception, pour trouver des alternatives et effectuer des retouches. Parfois, il faut même reprendre la conception depuis le début. Les problèmes et les points de blocage ne se révèlent qu'au fur et à mesure de la progression. Il est quasiment impossible de tout prévoir au départ.

Les grandes lignes du processus de conception ont été vues dans le Chapitre 10. Nous commençons par définir des objectifs puis nous les détaillons pour obtenir des exigences fonctionnelles. À partir de ces exigences, nous pouvons identifier les besoins en termes de matériel et de logiciel. Nous descendons ensuite dans les détails de chaque technique et composant pour les évaluer. Le schéma de la Figure 13.1 illustre de façon détaillée le cycle de conception.

À la fin de chaque étape, il faut évaluer la qualité du travail réalisé. Si un problème est détecté, il faut le résoudre immédiatement. Selon l'importance du problème, nous pouvons soit revenir d'une étape en arrière, soit abandonner un de nos choix en réduisant le périmètre fonctionnel. Imaginez par exemple que nous devions constater qu'un bouclier, capteur ou module n'est plus disponible et que nous n'avons pas le budget pour en construire un nous-mêmes. Il faut dans ce cas revenir en arrière jusqu'à l'étape pendant laquelle cet élément a été sélectionné et voir comment éliminer ce blocage.

Il n'est parfois pas nécessaire de construire un prototype, notamment lorsque le projet est en fait l'assemblage de modules précâblés, ce qui est souvent le cas dans le monde Arduino. Personnellement, j'aime construire des prototypes au moyen des plates-formes de prototypage dont je dispose, par exemple le Duinokit et les plaques d'expérimentation vues dans le Chapitre 10 ou les cartes Arduino avec des boucliers (Chapitres 11 et 12). Le but est d'obtenir le même résultat fonctionnel que le projet définitif sans avoir à faire de la soudure et à percer des trous. Dans certains projets, vous pouvez vous économiser la phase de prototypage, mais n'oubliez jamais que dans la version finale, une erreur peut s'avérer plus lourde de conséquences en temps comme en argent.

Dans la pratique, le coût d'un défaut de conception ou de mise en œuvre, qu'il soit matériel ou logiciel, augmente au fur et à mesure de l'avancement dans le projet, et des étapes successives de la Figure 13.1. Plus tôt vous détectez une erreur d'exigence fonctionnelle, plus il sera facile de corriger le tir. C'est pour la même raison qu'il est toujours moins coûteux de détecter un souci dans le prototype que dans le produit prêt à

être lancé en grande série.

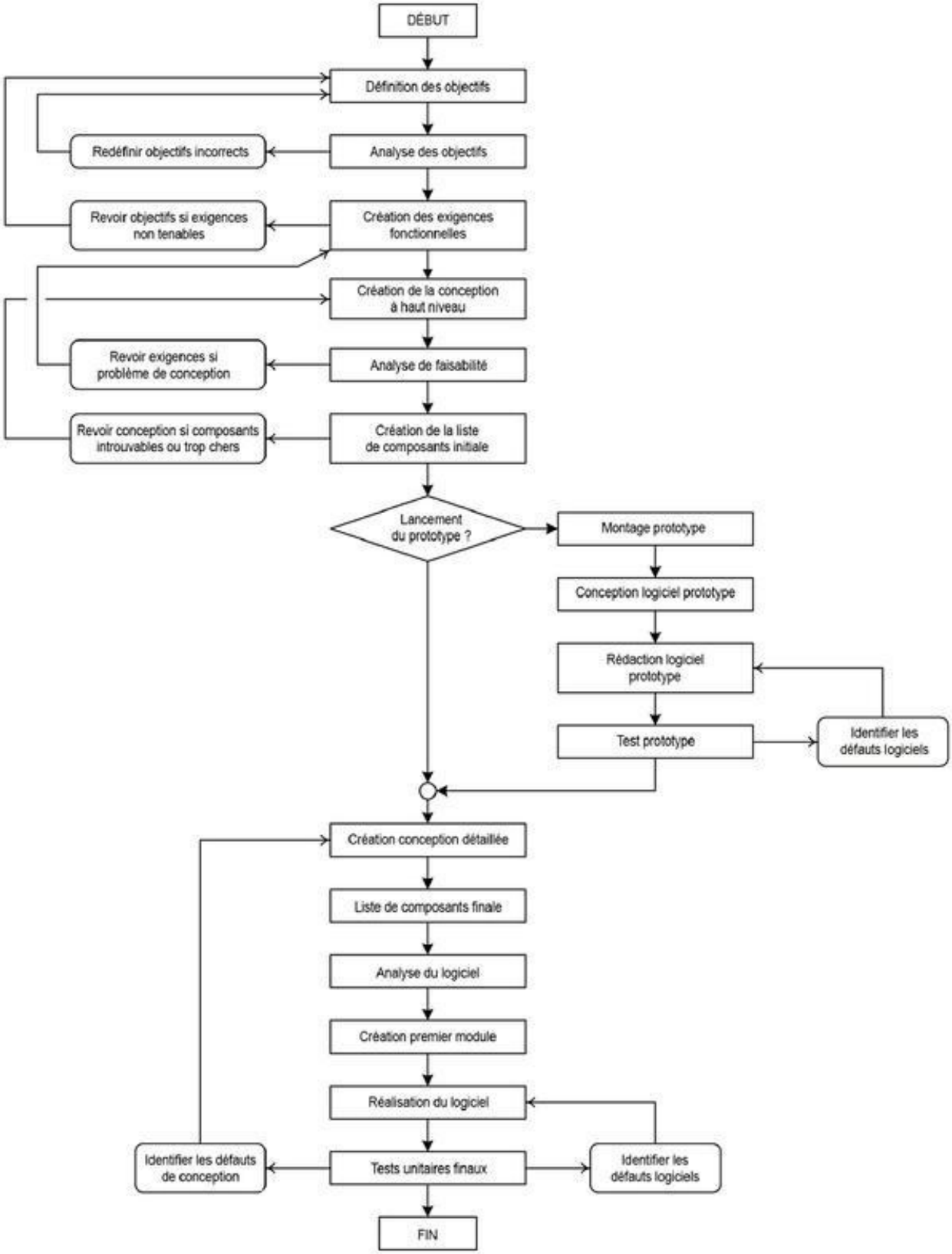


Figure 13.1 : Exemple de cycle de conception.

Définition des objectifs

La première étape d’une bonne conception consiste toujours à définir clairement les objectifs du projet, autrement dit, ce que veut le client. Au départ, vous ne disposez pas de tous les détails, mais vous devez pouvoir établir les caractéristiques essentielles, même s’il y en a peu au début. Cette liste d’objectifs va s’allonger, parfois de manière étonnante. Il n’est pas toujours facile de maintenir les objectifs en ligne avec le réel, surtout lorsque celui qui rédige les objectifs n’est pas la personne qui va réaliser la conception détaillée. Ce n’est heureusement pas le cas ici. Nous garderons notre optimisme et réussirons à ne pas subir un « décollement du réel ».

Il est conseillé de hiérarchiser la liste des objectifs par importance décroissante. Vous placez en début de liste les fonctions incontournables, et renvoyez en fin de liste les options diverses. Vous pourrez ainsi facilement supprimer certaines fonctions si vous constatez que l’investissement requis devient supérieur au budget fixé. Ce qu’il faut, c’est conserver le jeu de fonctions essentielles, celui sans lequel le projet n’a plus de sens.

Voici donc la liste d’objectifs hiérarchisée pour un projet de contrôleur de déclenchement d’événements.

Voyons d’abord les objectifs indispensables :

1. Le contrôleur doit être doté d’un compte à rebours avec des délais prédéfinis de 5, 10, 15, 30 et 60 secondes.
2. Le décompte doit pouvoir être abandonné à tout moment pour différentes raisons indiquées plus bas.
3. Le contrôleur doit être muni d’un interrupteur de verrouillage à clé. Le démarrage du compte à rebours ou la mise sous tension des allumeurs ne doit être possible que si ce verrou est sur la position RUN.
4. Le démarrage du compte à rebours est réalisé par appui sur un poussoir.
5. Le compte à rebours ne peut commencer que si l’interrupteur de sécurité est enfoncé avant d’enfoncer le poussoir du compte à rebours puis maintenu pendant toute la période du compte à rebours. Ce bouton de sécurité peut être placé dans une petite boîte indépendante ou un pendentif, en tout cas physiquement éloigné du pupitre principal.

6. Si le poussoir de sécurité est relâché avant la fin du compte à rebours, toute l'opération est annulée et le contrôleur bascule dans le mode Sécurité. Les circuits des allumeurs sont forcés à la masse.
7. Le contrôleur doit pouvoir alimenter plusieurs allumeurs à la fois, deux au minimum, et jusqu'à huit si possible.
8. Chaque ligne d'allumeur doit pouvoir être contrôlée indépendamment des autres afin de s'assurer qu'il n'y a aucune rupture de continuité de circuit avant le déclenchement.
9. La détection d'une panne de continuité doit provoquer l'abandon du compte à rebours, de la même façon qu'un relâchement du bouton de sécurité.
10. Le contrôleur fera clignoter une diode LED chaque seconde pendant le compte à rebours. D'autres diodes LED serviront à indiquer l'état du contrôleur : armé, en compte à rebours, abandon de sécurité ou souci d'allumeur.

Voici les objectifs secondaires :

1. Le compte à rebours est affiché par un afficheur à diodes LED sept segments.
2. L'heure actuelle est présentée sur un autre afficheur LED sept segments.
3. Un afficheur LCD de deux lignes de 16 caractères sert à afficher les messages d'état et les erreurs.
4. Tous les composants sont montés dans un châssis profilé trapézoïdal.

Voici enfin les objectifs facultatifs :

1. Le contrôleur doit pouvoir déclencher des appareils extérieurs à des moments précis dans le compte à rebours, par exemple pour couper une liaison à la masse, déclencher une sirène ou activer un appareil électronique tel qu'une caméra.
2. Le contrôleur est muni d'un connecteur permettant d'y relier un afficheur sept segments de grande dimension pour affichage public.
3. Le contrôleur est capable de générer une voix synthétique pour épeler le compte à rebours et faire des annonces.

4. Le contrôleur est doté d'une interface Bluetooth pour pouvoir se connecter à un système de prévision météorologique, par exemple pour parer un vent trop important. En cas d'alerte vent, le système doit pouvoir abandonner la séquence et en informer le responsable.

Cette liste ne manque pas d'ambition. Les objectifs primaires sont assez facilement réalisables, et c'est bien pour cela qu'ils ont été choisis en premier. Les objectifs secondaires sont un peu plus délicats à réaliser, et ceux facultatifs risquent vraiment de vous prendre beaucoup de temps, et d'argent.

Rédaction des exigences fonctionnelles

La façon dont j'ai rédigé les phrases des différents objectifs va nous permettre très facilement de produire la liste des exigences. Une exigence est en général une seule phrase qui doit être sans ambiguïté, compacte, cohérente et vérifiable (testable). Certains des objectifs vont pouvoir être utilisés directement sous forme d'exigences sans retouche.

La question est maintenant de savoir combien d'objectifs nous devons conserver en tant qu'exigences fonctionnelles ? Pour nous aider à décider, nous pouvons appliquer un système de notation. Chaque objectif va être pondéré en fonction de sa difficulté de réalisation, de sa nécessité et de son coût.

Nous commençons bien sûr par les objectifs primaires, donc obligatoires. Le Tableau 13.1 présente les objectifs primaires avec mon estimation de difficulté, de coût et de nécessité. Les notes sont croissantes de 1 à 5.

Tableau 13.1 : Notation des objectifs primaires.

Objectif	Difficulté	Coût	Nécessité
1.1	2	2	2
1.2	3	2	5
1.3	3	2	5
1.4	2	2	4
1.5	2	3	5

1.6	2	1	5
1.7	3	3	3
1.8	4	3	5
1.9	3	2	5
1.10	3	2	2

Les notes de l'objectif 1.1, qui correspond au compte à rebours avec délai prédéfini, sont telles que nous pourrions nous contenter d'un décompte fixe de 10 ou 15 secondes. De même, l'objectif 1.10, les lumières clignotantes, peut être déclassé pour rejoindre les objectifs secondaires. Nous voudrions cependant conserver sans doute au moins une diode LED clignotante en même temps que les secondes pour savoir que le compte à rebours a commencé.

Appliquons le même système de notation aux objectifs secondaires et aux options, ce que montre le Tableau 13.2.

Tableau 13.2 : Notation des objectifs secondaires et facultatifs.

Objectif	Difficulté	Coût	Nécessité
2.1	3	3	2
2.2	3	3	2
2.3	3	3	2
2.4	3	3	2
3.1	4	4	1
3.2	4	3	1
3.3	4	4	1
3.4	4	4	1

Si vous comparez les deux tableaux, vous pouvez facilement détecter que la plupart des objectifs primaires sont assez peu coûteux et faciles à réaliser. À l'exception du 1.1 et du 1.10, tous sont nécessaires. Dans le Tableau 13.2, le coût et la difficulté augmentent alors que le besoin diminue. C'est tout à fait logique : les objectifs facultatifs ne sont pas nécessaires, même si intéressants à ajouter (« *nice to have* »).

Lorsque j'ai établi ma liste d'objectifs, je n'avais pas spécialement préparé l'ordre. Honnêtement, je pense que si la liste est déjà presque bien triée, c'est en raison de mes années d'expérience. Dans tous les cas, essayez dès le départ de définir vos objectifs en cherchant d'abord leur nécessité, puis en fonction du coût ou de la difficulté, en fonction de ce qui est le critère essentiel pour vous : le temps ou l'argent.

Un arbitrage va être nécessaire. Faut-il éliminer une caractéristique nécessaire mais coûteuse ou bien faire une croix sur un objectif moins nécessaire afin de rester dans le budget initial, ou bien accepter d'augmenter le budget pour maintenir les deux objectifs ? Entre un objectif primaire et un autre, le choix est facile. Le résultat ne sera pas aussi sophistiqué, mais le budget ne sera pas dépassé. La raison d'être du projet ne sera pas mise en péril. En revanche, s'il faut choisir entre deux objectifs primaires ayant le même gradient de nécessité, comme les 1.1 et 1.10, il faudra arbitrer en fonction de la difficulté de réalisation. Dans notre cas, il sera sans doute plus simple d'ajouter un contacteur rotatif pour sélectionner un délai de compte à rebours que de percer des trous pour plusieurs diodes LED, faire leurs connexions puis créer le logiciel pour les contrôler.

Considérons que nous pouvons repousser l'arbitrage à plus tard. Nous allons conserver tous les objectifs et les transformer en exigences, ce qui permettra de voir en quoi ils influencent la conception et le choix des composants. Le Tableau 13.3 vous propose la liste complète des exigences fonctionnelles inspirées par la liste d'objectifs.

Tableau 13.3 : Exigences fonctionnelles du projet de déclencheur.

Exigence	Corps
1.1	Le contrôleur doit posséder une minuterie de compte à rebours. Lorsqu'elle atteint zéro, elle doit mettre sous

tension un ou plusieurs allumeurs.

1.1.1 Le compte à rebours doit proposer cinq délais prédéfinis de 5, 10, 15, 30 et 60 secondes.

1.1.2 Le choix du délai de compte à rebours se fait avec un contacteur rotatif à cinq positions.

1.1.3 Le décompte est abandonné si le délai est modifié en cours de décompte.

1.2 Le décompte peut être abandonné à tout moment pour différentes raisons fournies par d'autres exigences.

1.2.1 Le décompte est abandonné si un circuit d'allumeur qui devrait être fermé est ouvert (voir 1.7.1 et 1.9.1).

1.2.2 Le décompte est abandonné si le délai est modifié pendant qu'il est en cours.

1.2.3 Le décompte est abandonné si le poussoir de sécurité est relâché pendant le décompte.

1.3 Le contrôleur doit disposer d'un interrupteur de sécurité d'armement à clé.

1.3.1 Le contrôleur ne commence pas de décompte et n'active aucun allumeur si l'interrupteur de sécurité n'est pas en position armé.

1.3.2 Le décompte est abandonné si l'interrupteur de sécurité est désarmé pendant le décompte.

1.4 Le compte à rebours est lancé par appui sur un poussoir.

1.4.1 Le bouton de démarrage du compte à rebours est légendé « Launch ».

1.4.2 Le bouton de démarrage est actif seulement si le poussoir de périmètre de sécurité est enfoncé.

1.5 Le contrôleur utilise un verrou de périmètre de sécurité pour contrôler le décompte et l'allumage.

1.5.1 Le compte à rebours ne peut commencer que si le poussoir de périmètre de sécurité est enfoncé avant le bouton de lancement.

1.5.2 Le poussoir de périmètre de sécurité doit rester enfoncé pendant tout le compte à rebours.

1.5.3 Le poussoir de périmètre de sécurité peut être monté dans un pendentif ou une petite boîte indépendante du pupitre avec au moins 3 m de câble.

1.6 Si le décompte est abandonné, quelle que soit la raison, le contrôleur bascule en mode Sécurité.

1.6.1 Dans le mode Sécurité, toutes les lignes des allumeurs sont forcées à la masse.

1.6.2 La sortie du mode de sécurité suppose de désarmer le contrôleur au moyen du verrou de sécurité à clé.

1.7 Le contrôleur doit pouvoir alimenter plusieurs allumeurs à la fois.

1.7.1 Le contrôleur devra accepter jusqu'à six lignes d'allumeurs actives.

1.7.2

Un commutateur rotatif permet de choisir le nombre de circuits d'allumeurs actifs.

1.8 Chaque ligne d'allumeur peut être testée indépendamment des autres afin de détecter un problème de continuité.

1.8.1 Chaque ligne d'allumeur doit disposer d'un test de continuité pour garantir que l'allumeur peut être alimenté.

1.8.2 Le courant du test de continuité doit être inférieur à 250 microampères sur chaque ligne d'allumeur.

1.9 La détection d'un défaut de continuité d'un allumeur doit provoquer l'abandon du décompte.

1.9.1 Le décompte ne doit pas démarrer si un des circuits d'allumeurs n'a pas de continuité.

1.9.2 Le décompte est abandonné si une ligne d'allumeur n'a plus sa continuité pendant le décompte.

1.9.3 Le contrôleur affiche sur des diodes LED les circuits actifs et testés (voir 1.1.2).

1.10 Le contrôleur utilise des diodes LED pour afficher le décompte et le statut.

1.10.1 Une diode LED clignote chaque seconde pendant le décompte.

1.10.2 Une série de diodes LED sert à montrer l'état actif et la continuité de chaque circuit d'allumeur.

1.10.3 Une diode LED indique l’état actif du poussoir de périmètre de sécurité.

1.10.4 Une diode LED témoigne d’un abandon à cause d’une pénétration dans le périmètre de sécurité.

1.10.5 Une diode LED montre l’état du verrou de sécurité à clé.

1.10.6 Une diode LED clignotante montre que le contrôleur a basculé en mode de sécurité.

2.1 Le compte à rebours est affiché sur un afficheur LED à 7 segments.

2.1.1 Le décompte du compte à rebours est affiché sur un afficheur à LED rouge à 7 segments sur deux chiffres.

2.1.2 L’afficheur du compte à rebours présente des tirets après un déclenchement réussi.

2.1.3 En cas d’abandon, l’afficheur affiche le mot « Abort ».

2.2 Un afficheur LED à 7 segments affiche l’heure actuelle.

2.2.1 L’heure actuelle est affichée au format HH:MM par un afficheur LED jaune à 7 segments sur quatre chiffres.

2.2.2 L’heure est réglable au moyen de boutons poussoirs sur la face avant du contrôleur.

2.2.3 L’heure est affichée au format 24 heures.

2.3 L’état du système et les messages d’erreur sont affichés sur un panneau LCD de deux lignes sur 16 caractères.

2.3.1 L’état prêt à l’emploi et les erreurs sont affichés sur un afficheur LCD de deux lignes de 16 caractères.

2.3.2 L’afficheur LCD sait afficher des messages provenant d’une source externe, comme une station météorologique.

2.4 Tous les composants sont montés dans un châssis de type pupitre profilé.

2.4.1 Le contrôleur sera placé dans un châssis en métal profilé.

2.4.2 Les dimensions du châssis seront au minimum de 35,5 sur 25,5 cm avec une hauteur de 4 cm à l’avant et 8 cm à l’arrière.

3.1 Le contrôleur est capable de déclencher des appareils externes à certains moments pendant le compte à rebours.

3.1.1 Le contrôleur est capable d’activer des appareils externes *via* des relais déclenchés à des moments précis pendant le décompte.

3.1.2 Les points de déclenchement sont paramétrés *via* une connexion USB vers l’Arduino du contrôleur.

3.2 Le contrôleur est doté d’un connecteur pour relier un afficheur à sept segments de grande taille pour visualisation par le public.

3.2.1 Les signaux qui alimentent l’afficheur du décompte sur deux chiffres sont dupliqués sur un connecteur à l’arrière du châssis.

3.2.2 L’afficheur externe est contrôlé par un circuit pilote délivrant un courant suffisant.

3.3 Le contrôleur est doté d'une capacité à générer des messages vocaux synthétiques pour le compte à rebours et les annonces.

3.3.1 Le contrôleur est doté d'un bouclier de voix synthétique qui produit une voix intelligible.

3.3.2 Les messages vocaux sont définis par l'utilisateur.

3.4 Une interface Bluetooth permet de connecter au contrôleur une station météorologique externe.

3.4.1 L'utilisateur peut définir une vitesse de vent maximale. Lorsque cette limite est dépassée, cela provoque l'abandon du décompte.

3.4.2 Une diode LED indique un abandon en raison d'un vent trop fort.

J'ai essayé de maintenir une relation un vers un entre les objectifs et les exigences, mais j'ai été dans certains cas forcé de m'en écarter pour éviter des redondances.

Ce projet utilise un poussoir de périmètre de sécurité qui doit être maintenu enfoncé pendant tout le compte à rebours. C'est une précaution indispensable. Dans la pratique, le contrôleur peut être servi par deux personnes : un directeur de tir et un officier de sécurité du champ. C'est ce dernier qui a le droit de faire abandonner un déclenchement dès que quelque chose lui paraît anormal, comme une bourrasque ou l'apparition d'une personne dans le périmètre interdit.

Conception initiale

La lecture de la liste d'exigences fonctionnelles permet immédiatement de prendre des décisions de conception. Nous savons qu'il nous faut deux contacteurs rotatifs, un afficheur LCD, deux afficheurs LED à 7 segments, une

horloge temps réel, une petite boîte pendentif avec un poussoir pour le périmètre de sécurité, quelques connecteurs sur la face arrière, quelques relais, et toute une série de diodes LED de différentes couleurs.

Pour nous faire une première idée, nous pouvons dessiner un premier diagramme, comme celui de la Figure 13.2. Le nombre de connexions requises permet de constater qu'un modèle Arduino Uno ne sera pas suffisant pour gérer toutes ces entrées-sorties. Nous devons donc plutôt envisager un modèle Mega2560. Nous pourrions aussi envisager d'utiliser plusieurs cartes Arduino pour répartir le travail.

La Figure 13.3 est une version un peu plus élaborée dans laquelle nous utilisons un Uno pour la synthèse vocale, un autre pour le Bluetooth et un Mega2560 pour les fonctions principales. Je rappelle que l'ATmega2560 utilise un contrôleur de même nom qui possède plusieurs interfaces USART. Chacune des cartes Uno secondaires dispose ainsi de son propre canal de communication.

Plusieurs des interfaces de la Figure 13.3 ont été qualifiées comme étant du type I2C. Les ports d'entrées-sorties numériques sont identifiés par la mention DIO. Nous utilisons un multiplicateur d'entrées-sorties comme dans le projet Switchinator du Chapitre 10 pour servir les relais et les 12 diodes LED de statut des allumeurs.

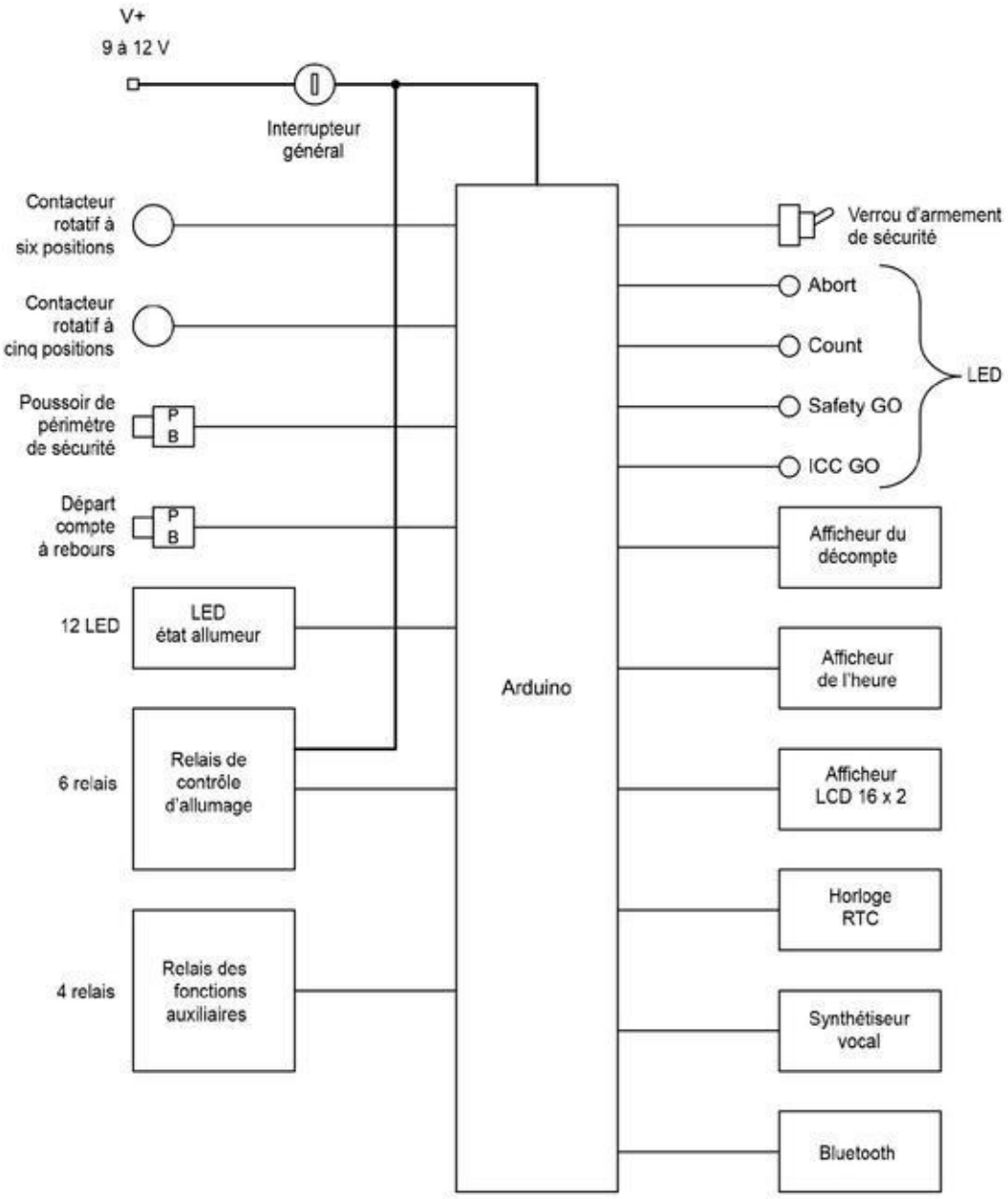


Figure 13.2 : Diagramme fonctionnel initial du contrôleur.

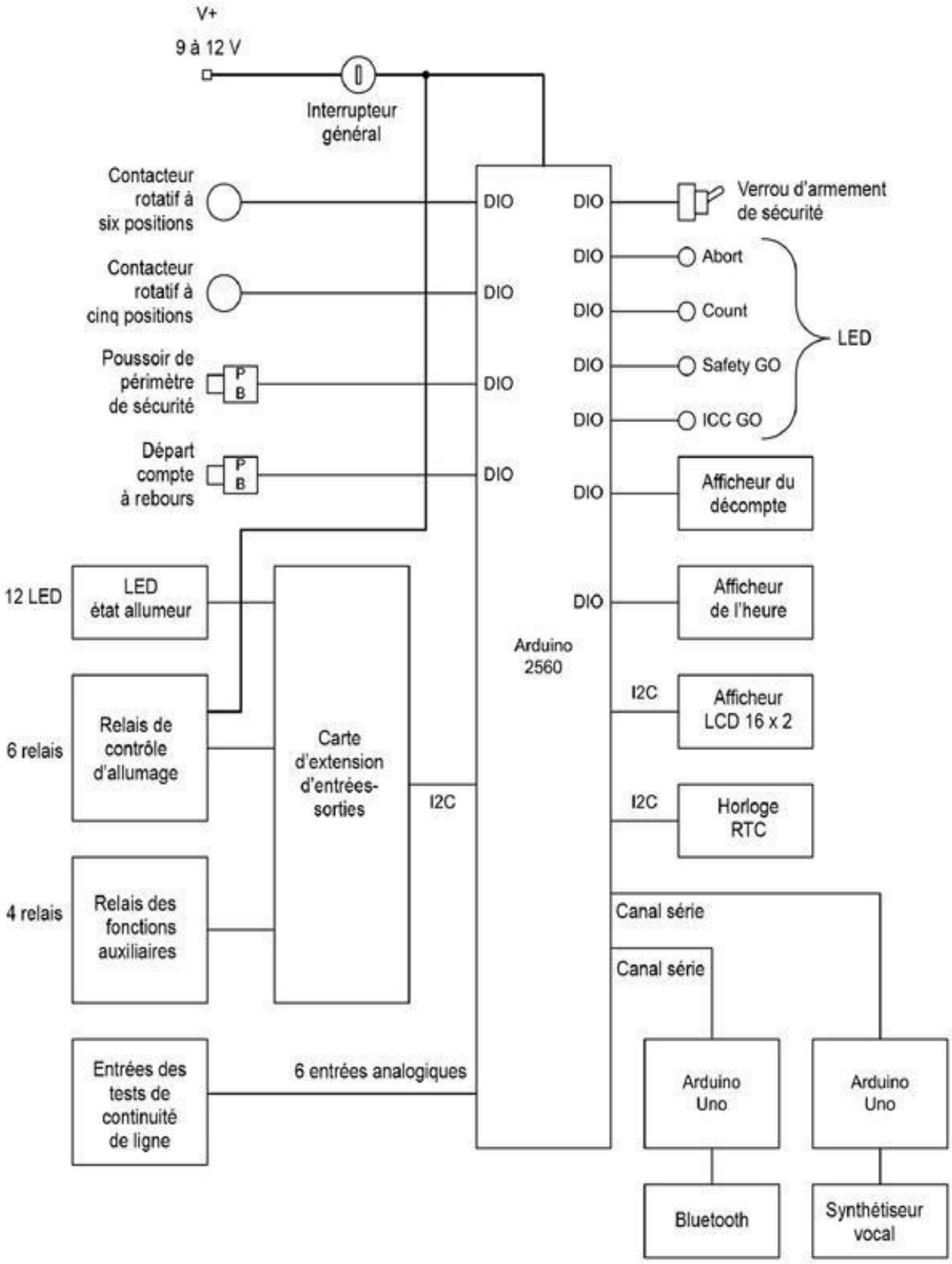


Figure 13.3 : Version élaborée du diagramme du contrôleur.

Sur mesure ou du commerce ?

Dans les projets pas trop complexes, vient un moment où il faut réaliser un choix entre acheter des composants ou des modules tout prêts ou les construire soi-même pour bien s'adapter aux exigences fonctionnelles. La décision n'est jamais simple, et elle ne se résume certainement pas à des considérations financières.

Un composant ou un module du commerce, souvent désigné par l'acronyme anglais COTS (*Commercial Off-The-Shelf*), ne pourra pas répondre à toutes les attentes de votre projet. Il pourra par exemple offrir les fonctions demandées, mais ne pas disposer des bons connecteurs, ou l'inverse. Son encombrement physique pourrait ne pas convenir, ou bien sa tension d'alimentation serait différente de celle de tous les autres composants de votre projet, ce qui obligerait à ajouter des composants supplémentaires.

Il faut choisir entre adapter votre projet pour accueillir le composant ou se lancer dans la construction de l'équivalent sur mesure. Si les retouches appliquées au module du commerce sont minimales, il est sans doute plus efficace d'effectuer les adaptations plutôt que de se lancer dans la création d'un module spécifique. En revanche, si le module ou le composant comporte un point vraiment bloquant, au niveau des dimensions, de certaines fonctions manquantes ou d'un conflit, il sera nécessaire d'envisager la conception et la réalisation d'un module sur mesure, en parfaite cohérence avec les autres composants du projet.

Quand la décision n'est pas évidente, vous mettrez dans la balance le budget et le temps requis pour acquérir et adapter le composant du commerce, ou le réaliser sur mesure. Si ces critères ne suffisent pas à trancher, demandez-vous à quel point vous devez pouvoir garder le contrôle de la production et du fonctionnement interne du composant. Tenez compte du fait qu'un module du commerce pourrait faire l'objet d'un arrêt de fabrication alors que votre projet se fonde sur lui. Dans ce cas, pour ne prendre aucun risque, vous choisirez la voie du sur mesure. De même, si vous avez besoin de connaître le fonctionnement interne du module, mais qu'il est fourni sous forme d'une boîte noire impénétrable, vous vous détournerez de cette solution pour passer à la création d'un module sur mesure.

Étude de faisabilité

Vous devez évaluer la facilité de réalisation de ce que vous venez de concevoir. Est-ce que le projet répond bien aux besoins ? Est-ce qu'il offre trop de richesses fonctionnelles ? Le moment est idéal pour faire un pas en arrière en revisitant la conception initiale d'un œil critique. Vous pourrez ainsi supprimer toutes les fonctions moins nécessaires ou superflues, et cela sans regret. La faisabilité est un critère relatif. Quelque chose qui peut sembler non faisable sur un coin de table le devient aisément lorsque vous disposez d'un atelier complet. Les contraintes budgétaires et le temps disponible sont d'autres critères relatifs. Une personne avec un petit budget et peu de temps ne parviendra évidemment pas à réaliser ce que fera son collègue bien doté en temps et en argent. Si vous êtes seul et n'avez qu'un budget et du temps limité, vous devez être d'autant plus prudent pour évaluer la faisabilité d'un nouveau projet. C'est exactement mon cas, et je ne suis certainement pas le seul.

Fioritures et séduction

Dans le monde de l'ingénierie, les fioritures sont toutes les caractéristiques qui n'apportent rien au fonctionnement d'un système. Ce sont des options de confort ou d'esthétique. Prenons le cas d'un affichage en couleur pour une machine à café. Est-ce que sa présence rend le café meilleur ? Bien sûr que non. Est-ce que cela fait joli dans la cuisine ? Absolument, et la seule raison d'être de l'afficheur est que certains d'entre vous seront prêts à payer pour cette version plus chère.

Les automobiles actuelles sont des magasins de fioritures roulants. Qu'est-ce que l'essence d'une automobile ? Quatre roues, un moteur, des sièges, et une coque avec des fenêtres pour se protéger de la pluie, du vent et de la poussière. Personne n'a absolument besoin d'un système audio à dix haut-parleurs, d'un système de navigation, de sièges électriques chauffants à mémoire, pour aller faire ses courses. Les modèles de début de gamme sans aucune option ne sont sans doute pas les plus séduisants sur un parking, mais ils vous emmèneront là où vous avez besoin d'aller. Mieux encore, le modèle de base fonctionnera sans doute plus longtemps que celui gorgé de multiples options sophistiquées prêtes à tomber en panne et à réclamer une visite chez votre concessionnaire.

Voici les cinq critères que j'aime utiliser pour évaluer la faisabilité d'une conception :

1. Est-ce que le projet est sensé au niveau de ses fonctions ?
2. Est-ce que le projet est sensé au niveau financier ?
3. Est-ce que le projet dépend d'un composant difficile à trouver ?
4. Est-ce qu'il y a des points de la conception qui ne sont pas sûrs ?
5. Est-ce que le projet peut être réalisé, programmé et testé par une personne seule dans un temps limité ?

Dès qu'une des réponses est négative ou pas franchement positive, vous devez envisager de supprimer le composant ou la fonction en question.

Le schéma fonctionnel du projet que j'ai représenté dans la Figure 13.3 est superbe, mais je ne pense pas être en mesure de le terminer seul dans un temps limité. J'ai intérêt à partir d'une version plus simple, la plus simple possible, en me permettant de l'enrichir plus tard.

Grâce à l'approche choisie pour établir la liste des exigences, sous forme de trois cercles de priorités décroissantes, il me sera facile de simplifier la conception. Les exigences de niveau trois sont toutes celles qui vont demander beaucoup de temps et d'argent sans rien apporter au noyau fonctionnel du projet, qui consiste à déclencher des événements. Je vais donc considérer que toutes les exigences de niveau trois sont des options qui pourront être ajoutées ultérieurement.

Je vais appeler Phase 1 le projet constitué seulement des exigences de niveaux 1 et 2. La version complète correspondra à la Phase 2. Puisque nous avons maintenant défini le périmètre de notre phase, nous pouvons chercher à simplifier encore un peu la conception, sans rien ôter à la palette fonctionnelle.

La Figure 13.3 montre par exemple un circuit d'extension d'entrées-sorties. Il peut être incarné par un bouclier d'extension de type Centipede (décrit dans le Chapitre 8). Ce circuit offre 64 entrées-sorties numériques, ce qui lui permet aisément de gérer toutes les diodes LED d'état des allumeurs, les quatre diodes LED de l'état du système et les deux contacteurs rotatifs. Nous pouvons simplifier le câblage en utilisant le plus possible les protocoles SPI et I2C. Justement, les modules d'horloge RTC, d'afficheur LCD 16x2 et la carte d'extension d'entrées-sorties peuvent fonctionner avec une interface I2C. De même, les afficheurs à 7 segments pour le compte à rebours et le délai existent

en version SPI. Cela nous permet de revoir le diagramme fonctionnel de la Phase 1 (Figure 13.4).

J’ai choisi de maintenir une liaison directe avec les sorties numériques du Mega2560 pour les relais de déclenchement. J’évite ainsi de dépendre d’un intermédiaire entre le circuit Arduino et les relais. Cela n’a aucun effet en termes de fonctions, mais cela élimine une source d’erreur potentielle dans le chemin critique du déclenchement. C’est pour la même raison que je connecte directement à la carte Arduino les poussoirs de début de décompte, le verrou de sécurité et le poussoir de périmètre de sécurité. De plus, en reliant directement ces interrupteurs, nous pourrons ultérieurement tirer profit de la capacité de déclenchement d’interruption du microcontrôleur en cas de changement d’état d’une broche.

C’est donc avec le diagramme de la Figure 13.4 que nous allons continuer. Plus tard, nous pourrons envisager de passer à la Phase 2. Mais chaque chose en son temps.

Nomenclature de composants initiale

Grâce à notre première conception de la Phase 1, nous pouvons établir notre première liste de composants. Pour déterminer les quantités requises, il suffit de passer en revue les exigences fonctionnelles en comptant le nombre de cas d’utilisation des fonctions et des éléments de contrôle de l’interface.

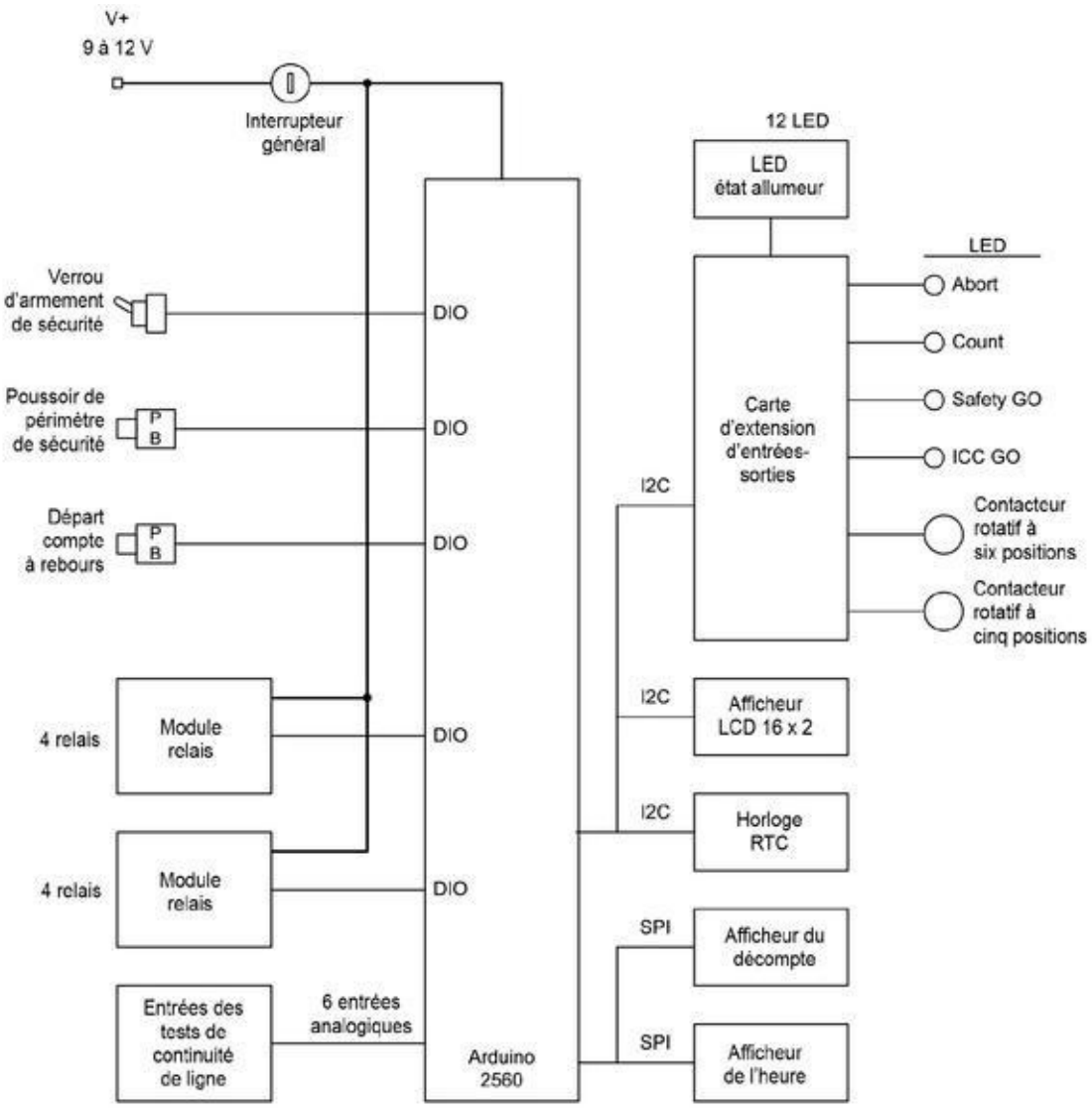


Figure 13.4 : Diagramme fonctionnel de la Phase 1 du projet.

Dans cette liste préliminaire, nous ne mentionnons pas les consommables que sont la plaque de prototypage, les fils de câblage et les borniers (identiques à ceux utilisés dans le Chapitre 11). Nous détecterons et renseignerons ces besoins au moment d'effectuer le montage du prototype ou de la version définitive.

Tableau 13.4 : Nomenclature de composants initiale.

Quantité	Description	Quantité	Description
1	Arduino Mega2560	1	Afficheur LED 7 segments 2 chiffres

1	Bouclier bornier Mega	1	Afficheur LED 7 segments 4 chiffres
1	Bouclier afficheur LCD	1	Interrupteur à clé
1	Module horloge	1	Gros bouton poussoir
2	Module 4 relais	1	Boîtier métal type pupitre

Prototypage

J’ai dit plus haut qu’un prototype était utile sous plusieurs aspects. Tout d’abord, il permet de valider votre conception ainsi que les principaux composants, en gardant toute la liberté requise pour faire des essais. Dans le cas où un module ou un capteur ne fonctionnerait pas comme attendu, il est bien plus simple d’en changer avec un prototype qu’avec une version finale dans laquelle il faudrait par exemple faire de nouveaux perçages dans une façade en métal. Autre avantage du prototype, si vous avez prévu de faire faire un circuit imprimé ou plusieurs, il vous faudra attendre que la fabrication soit achevée. Pendant ce temps, vous pouvez travailler avec le prototype en avançant au niveau du logiciel ou en recherchant la documentation manquante. Enfin, vous disposez parfois dans votre stock d’un nombre de composants suffisant pour vous lancer dans un prototype qui réunit une partie suffisante de la palette de fonctions du produit définitif. La Figure 13.5 montre à quoi pourrait ressembler un prototype de notre projet basé sur une carte Arduino Mega2560, le tout étant mis en place sur du contreplaqué.

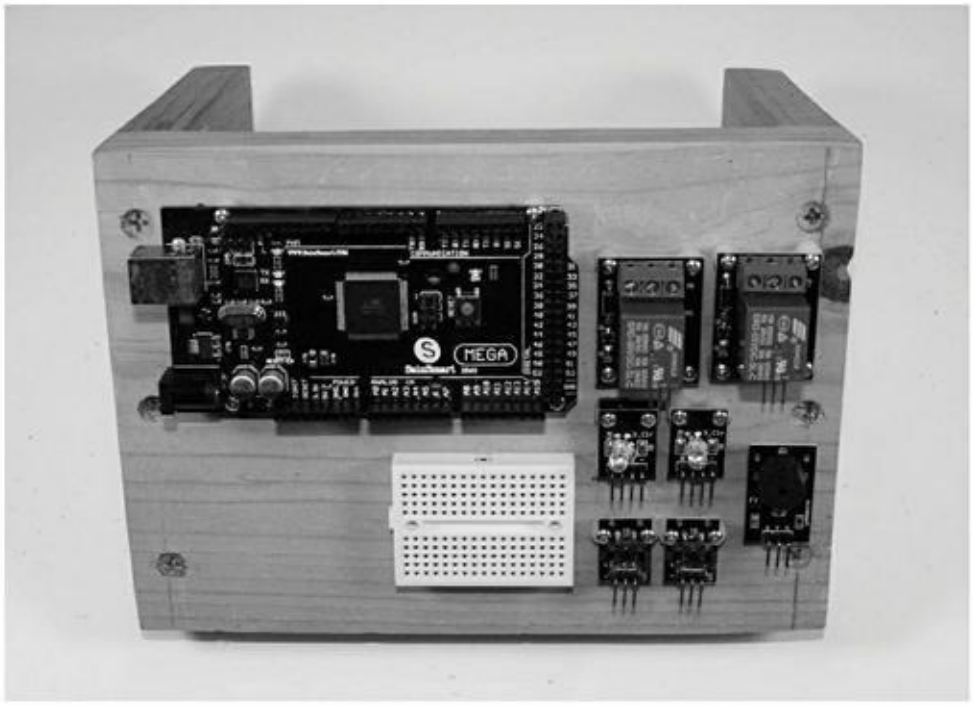


Figure 13.5 : Montage d’un prototype Mega2560.

Le prototypage permet donc d'économiser du temps et des efforts, mais l'étape n'est pas toujours obligatoire. Notre projet n'est par exemple pas très complexe en termes d'interconnexion. Il n'y a pas de circuit imprimé sur mesure et tous les composants sont des objets courants.

Cela dit, vous pouvez toujours décider de faire un prototype, et ce sera recommandé si vous désirez incorporer dès le départ les exigences facultatives. Dans ce cas, n'hésitez pas effectuer un montage sur plaques de bois.

N. d.T. : Le terme anglais pour les plaques d'expérimentation est *breadboard*. Effectivement, au début du XX^{e} siècle, les premiers amateurs d'électronique se servaient littéralement d'une planche (*board*) à découper le pain (*bread*) pour fixer les composants de leur montage, d'où le nom. On trouvait dans le commerce de telles planches sur lesquelles était collé un schéma de principe, ce qui guidait la personne dans le montage. On peut dire que les électroniciens avaient, comme aujourd'hui d'ailleurs, du pain sur la planche !

Cette histoire de planche à pain est fort sympathique à connaître, mais je vais néanmoins passer outre et réaliser directement le montage définitif. Nous avons vu comment réaliser des prototypes dans les trois précédents chapitres. Nous disposons d'un bon jeu d'exigences fonctionnelles pour la Phase 1 et d'une liste de composants. Passons donc à la réalisation du matériel et à l'écriture du code source du logiciel.

Conception détaillée

C’est dans la phase de conception détaillée ou conception finale que nous appliquons toutes les exigences fonctionnelles et mettons à profit ce que nous avons appris lors de la phase de prototypage éventuelle. Le résultat est une description complète de l’aspect physique et dynamique de l’appareil. Le comportement doit rester le même que celui du prototype et le logiciel ne devrait subir que d’infimes retouches.

Connexions électriques

La Figure 13.6 propose le schéma de principe de la totalité du projet, hormis les circuits de test de continuité des lignes d’allumeurs, représentés dans la Figure 13.7. Voici la distribution définitive des canaux de communication internes :

- Le module horloge RTC, l’afficheur LCD et la carte d’extension d’entrées-sorties utilisent une interface I2C.
- Les deux afficheurs à 7 segments utilisent une interface SPI.
- Les relais et les trois boutons poussoirs sont connectés directement au microcontrôleur.

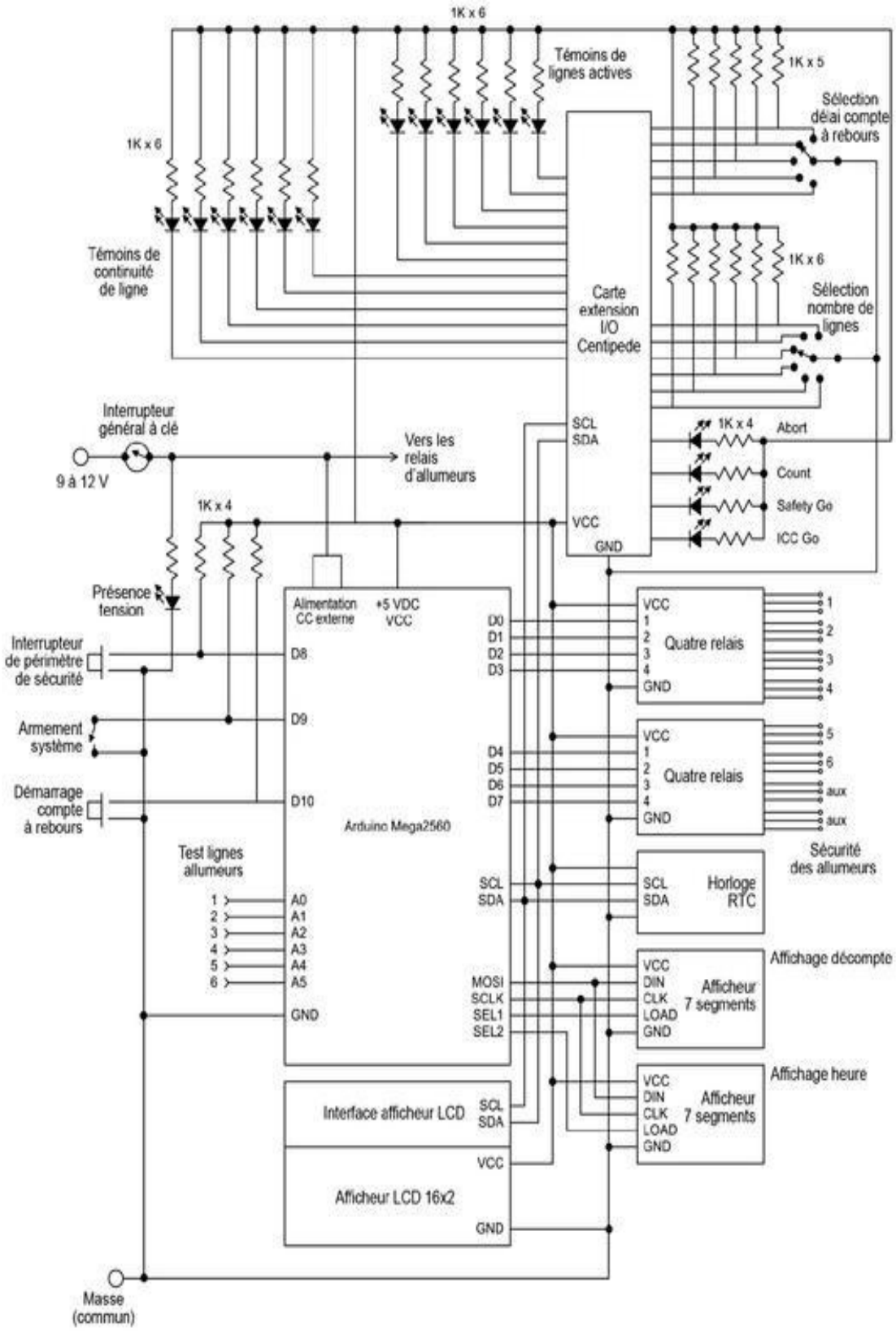


Figure 13.6 : Schéma de principe de la Phase 1 définitive.

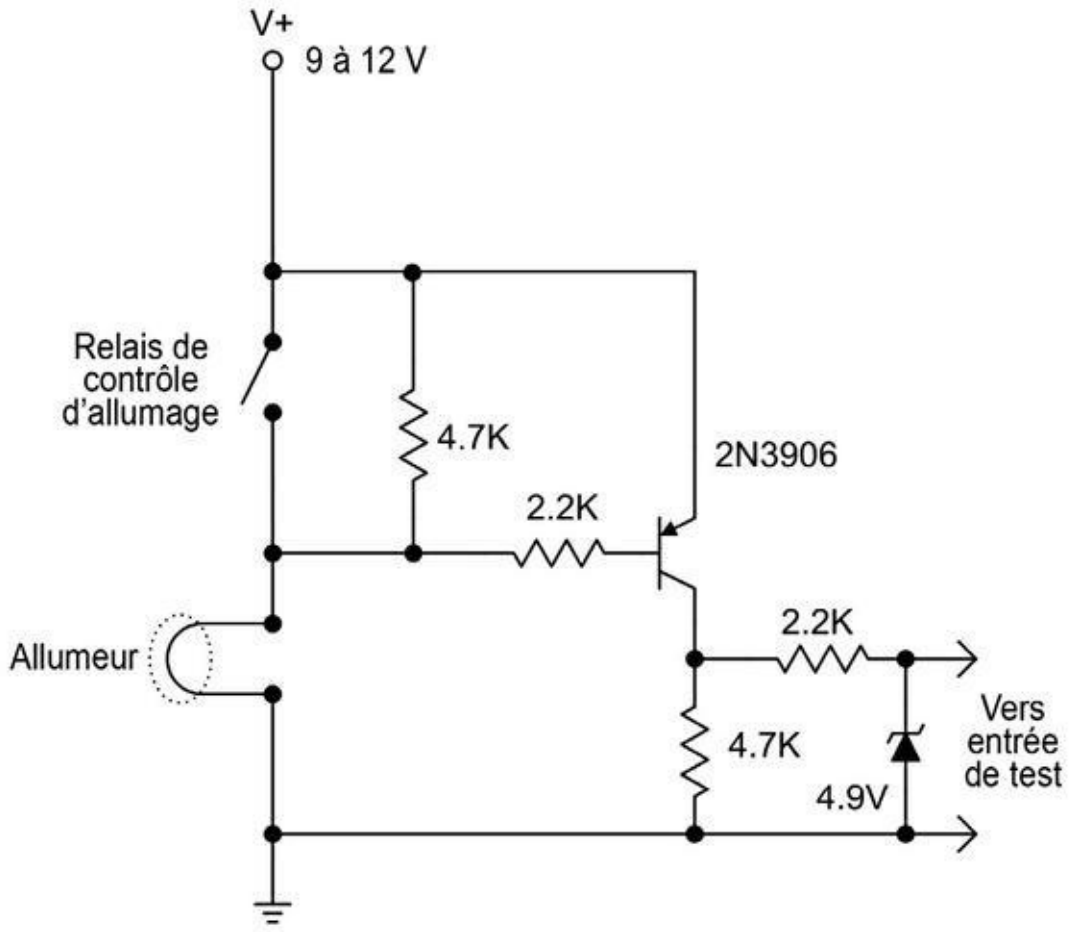


Figure 13.7 : Circuit de test de continuité des lignes d'allumeurs.

Un allumeur pour effets spéciaux ou modèles réduits de fusées se déclenche pour un courant d'environ 0,5 A. Le circuit de la Figure 13.7 va permettre de tester la bonne continuité de la ligne sur laquelle est branché l'allumeur, par une simple mesure de la tension présente aux deux bornes du test de continuité. Le défi, c'est de tester sans provoquer l'allumage ! Dans ce circuit, nous ne faisons passer qu'environ 2 milliampères dans l'allumeur. En outre, nous prévoyons une diode Zener 4,9 V pour empêcher que la tension revienne dans l'entrée analogique du microcontrôleur.

De nombreux passionnés se sont lancés dans la conception et la réalisation de ce genre de circuit de test de continuité. Le schéma que je propose n'est qu'une des solutions possibles.

Dans le schéma de la Figure 13.6, nous avons prévu six circuits reliés aux six entrées analogiques du contrôleur Mega2560. En pratique, je conseille d'implanter les six circuits sur une plaque de prototypage avec des borniers à

écartement de 2,54 mm, dans le même esprit que le circuit de protection du Chapitre 11.

Pour connecter chaque ligne d'allumage, je conseille d'utiliser des bornes à vis (ou à molette), comme celle visible sur la gauche de la Figure 13.8. Vous pourriez en outre prévoir les mêmes bornes à vis pour brancher une batterie externe. Si vous prévoyez également une batterie interne avec d'autres bornes à vis, vous aurez ainsi le choix entre deux sources d'alimentation, tout dépendant du nombre de circuits que vous avez besoin d'alimenter. La Figure 13.8 donne un exemple de connexion de la plaque des tests de continuité.

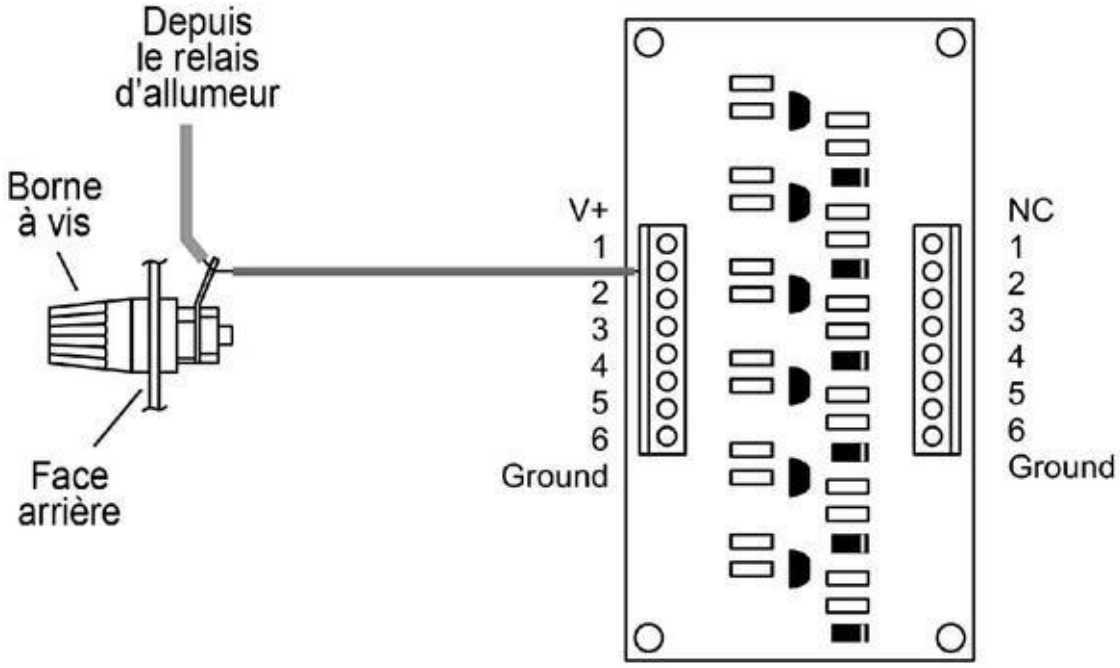


Figure 13.8 : Schéma de connexion de la plaque des circuits de test de continuité.

Dans le haut de la Figure 13.6, vous pouvez voir un premier groupe de six diodes LED avec la légende « Témoins de continuité de ligne » et un autre légendé « Témoins de lignes actives ». Ces diodes permettent de savoir l'état de chaque ligne et quels circuits d'allumeurs sont actifs. Le nombre de circuits actifs est sélectionné grâce au contacteur rotatif à six positions. Les diodes LED appropriées vont s'allumer pour prouver que l'allumeur concerné est branché et prêt à l'emploi. Après le déclenchement, la diode LED correspondante doit s'éteindre. Seules les diodes LED des allumeurs sélectionnés et actifs restent allumées.

La Figure 13.9 offre une autre perspective sur le câblage, en mettant l'accent sur les sorties des allumeurs, le module de diodes LED et de circuits de contrôle, sans oublier l'alimentation des modules. Vous constatez que les connexions I2C, SPI et les connexions numériques et analogiques directes ne sont pas détaillées. Nous utilisons une représentation groupée par bus, dont le symbole est une barre en diagonale avec le nombre de conducteurs symbolisés. Le schéma ne montre pas non plus les borniers pour l'alimentation et la masse.

La Figure 13.9 comporte un élément intéressant : un des relais du deuxième module de relais sert de protection de sécurité des allumeurs. Tant que ce relais est ouvert (non passant), les allumeurs ne peuvent pas être déclenchés. Vous remarquez en outre les rectangles qui correspondent à des plaques de connexion pour les circuits de test de continuité et les résistances de rappel des diodes LED et des interrupteurs. J'ai évité de devoir faire des soudures en utilisant des borniers à écartement de 2,54 mm. Cette solution permet également de débrancher et de recâbler plus tard si besoin.

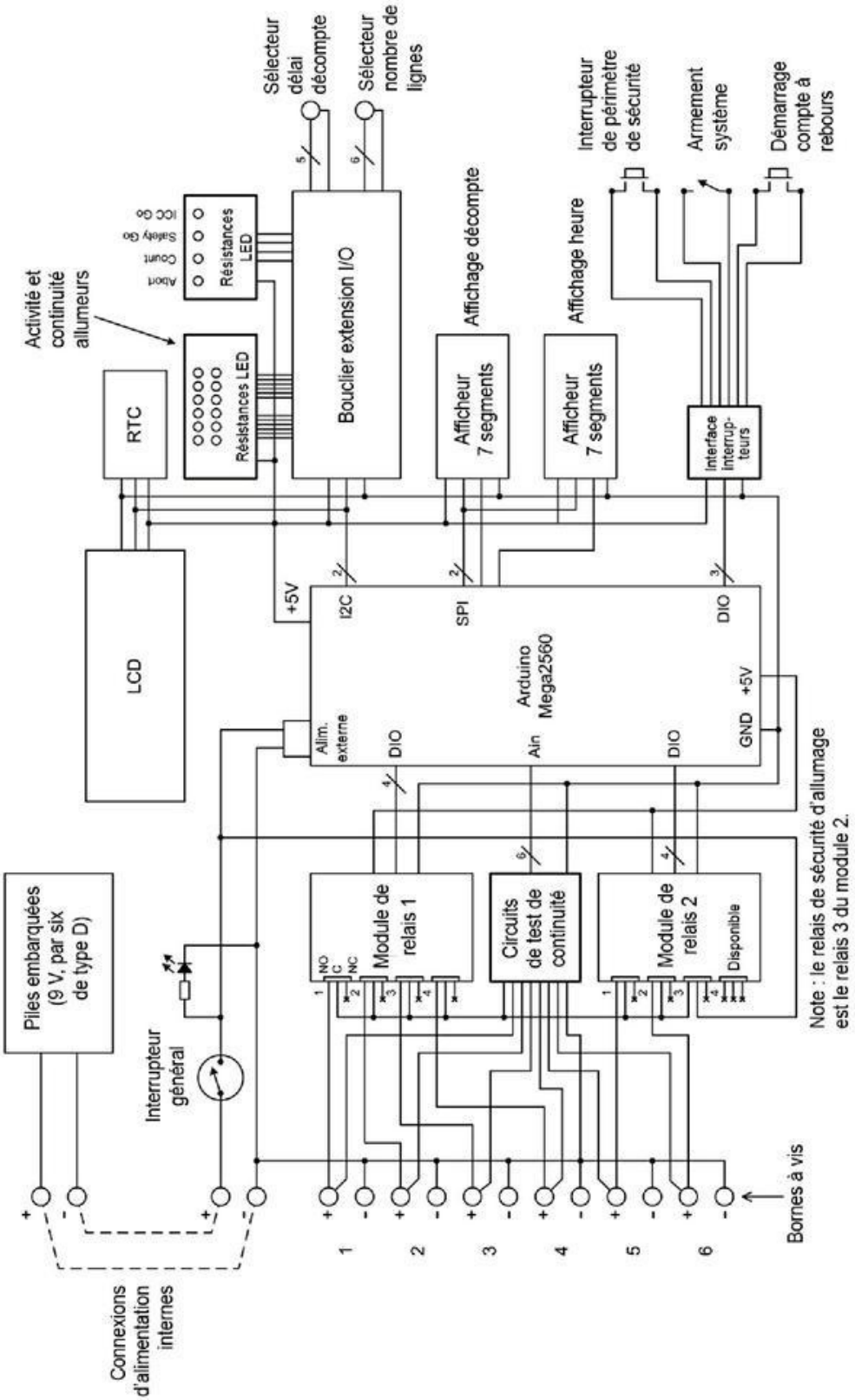


Figure 13.9 : Diagramme de connexions du châssis.

Nos schémas sont devenus suffisamment précis pour que nous puissions établir la nomenclature de composants détaillée (Tableau 13.5).

Tableau 13.5 : Nomenclature de composants définitive.

Quantité	Description	Quantité	Description
1	Arduino Mega2560	1	Gros bouton poussoir
1	Bouclier bornier Mega	8	Borne à vis rouge
1	Bouclier afficheur LCD	8	Borne à vis noire
1	Module horloge RTC	1	Jack 3,5 mm (mono)
2	Module 4 relais	1	Porte-piles, six de type D
1	Afficheur LED 7-segments 2 chiffres	1	Pupitre métal incliné
1	Afficheur LED 7-segments 4 chiffres	1	Interrupteur à clé

Montage physique

Si nous tenons compte du domaine d'utilisation de ce projet, le format idéal pour le boîtier est un châssis dans le style d'une table de mixage de sonorisation. Voyez par exemple la Figure 13.10. Il s'agit du dessin d'un vieux châssis qui traînait dans mon atelier. Je l'ai vidé de son contenu pour le recycler. Il est constitué de deux parties et mesure 36 cm de large, 25 cm de profondeur, 4 cm de haut devant et 7 cm de haut à l'arrière où se trouve un bandeau horizontal de quasiment 9 cm de large. Mon exemplaire était en tellement mauvais état que j'ai préféré ne pas le photographier. Voilà pourquoi vous n'en voyez qu'un dessin.

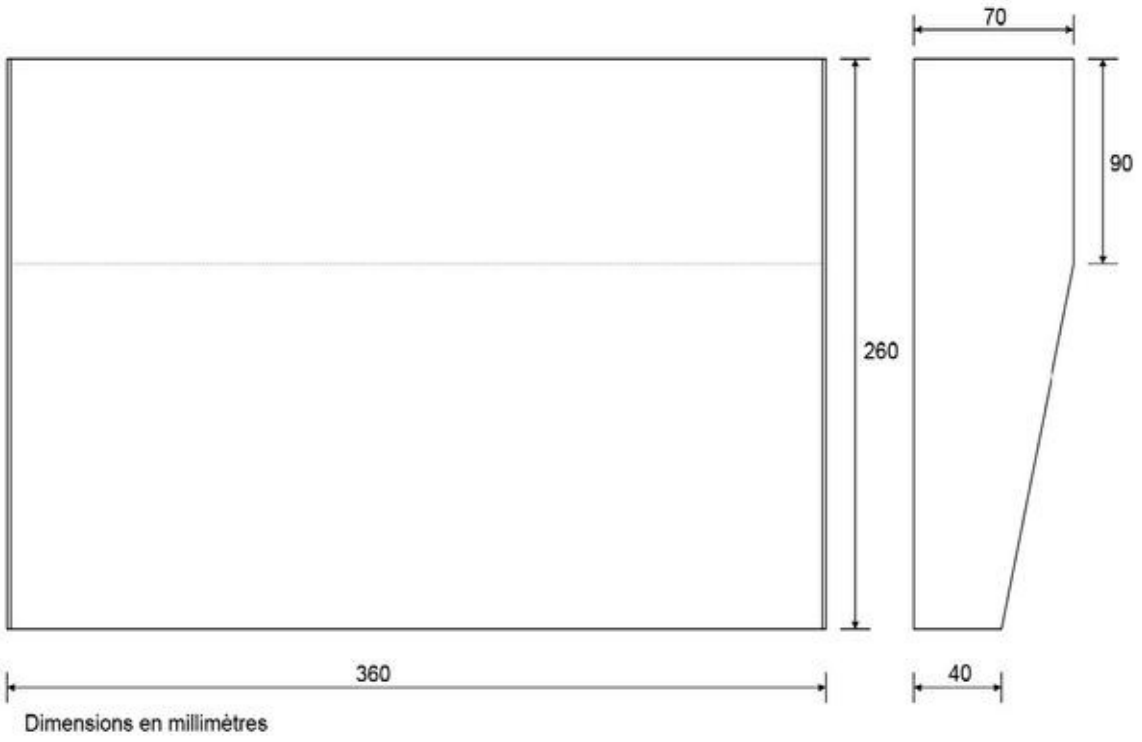


Figure 13.10 : Exemple de châssis pour contenir le projet.

Si vous voulez construire ce projet, rien ne vous oblige à chercher un châssis aux mêmes dimensions. Cherchez quelque chose d'à peu près comparable ou renseignez-vous chez les détaillants. Ce genre de boîtier est assez facile à trouver pour 30 à 50 euros.

Une fois que vous avez votre boîtier, l'étape suivante consiste à choisir comment peupler sa façade et comment installer les cartes et connecteurs. Avec l'exemple de la Figure 13.10, je conseille de fixer la plupart des composants sur le dessous de la partie horizontale à l'arrière. Du fait que le capot est généralement en une seule pièce, cela vous évite de devoir tirer trop de fils entre les modules installés sur le fond du châssis et les composants fixés en façade. Organisez-vous pour pouvoir facilement ouvrir le capot sans avoir à trop tirer sur les fils entre la partie mobile et la partie fixe du fond du châssis.

Je propose une implantation de la façade dans la Figure 13.11. Vous pouvez choisir une autre organisation en réservant de la place libre, car le module d'extension d'entrées-sorties offre de nombreuses possibilités supplémentaires (64 au total).

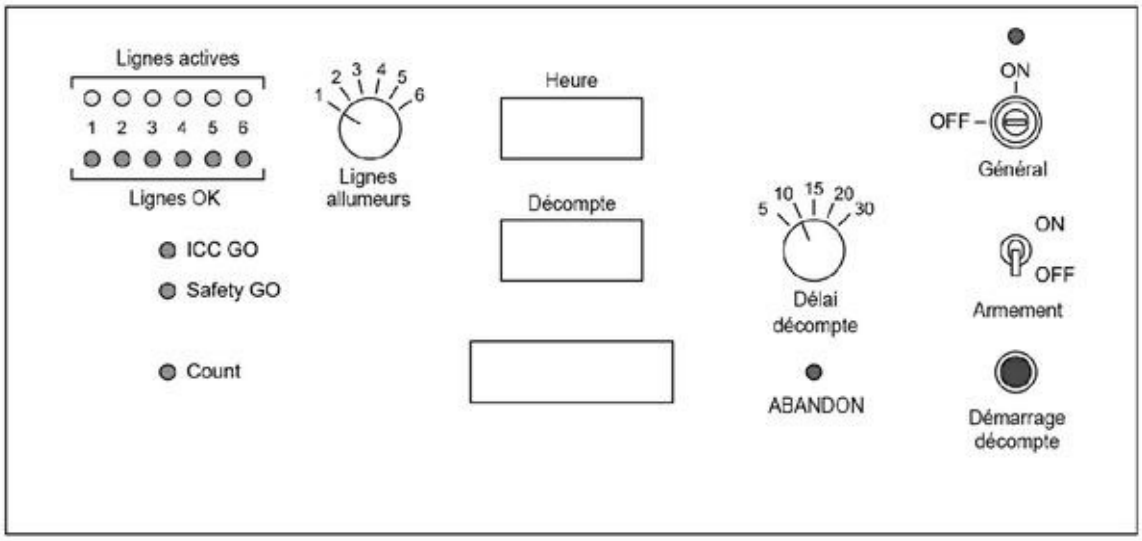


Figure 13.11 : Proposition d'implantation de la façade du contrôleur.

Je vous conseille d'installer tous les composants à l'intérieur du capot, sauf bien sûr les batteries éventuelles. Si vous voulez que les vis de fixation du capot ne soient pas trop visibles, choisissez des vis à tête fraisée, ou bien appliquez un point de peinture après vissage.

Je n'ai pas construit ce projet (bien que j'aurais pu quand je songe au temps passé à m'y préparer). Si je l'avais fait, j'aurais réinstallé la carte Mega2560 et la carte d'entrées-sorties sous le bandeau du fond de capot. Il y a largement assez de place en largeur pour y accrocher une carte Arduino, voire deux, sans que cela touche les composants de la façade.

Une bonne technique pour vérifier que tout va tenir avant de percer consiste à dessiner les empreintes des composants sur du calque ou du carton et le fixer aux endroits prévus. Pour symboliser les interrupteurs circulaires et les diodes LED, vous pouvez utiliser des gommettes rondes. Imaginez que vous puissiez voir à travers le capot avec des rayons X. En positionnant votre calque sur la face supérieure, cela vous évite de devoir sans cesse retourner le capot pour vérifier où vont tomber les diodes LED et les afficheurs. Votre calque va servir de patron, et vous pourrez prendre des mesures sur le calque pour aboutir à un dessin comme celui vu dans le Chapitre 11. (Si vous savez utiliser un outil de CAO, vous pouvez directement créer le dessin et passer l'étape du calque.)

Vous pourriez aussi faire sortir le connecteur USB de la carte Mega2560 par un perçage en face arrière. Vous pourrez ainsi facilement faire une mise à jour du

logiciel, collecter des données pendant l'utilisation, ou même récupérer des données à afficher sur un afficheur externe de grande taille.

Code source

Ce chapitre se limite à la conception. Au niveau du logiciel, nous n'avons pas écrit le code, mais travaillé au niveau des diagrammes. Lorsque le schéma de principe est bien conforme à la liste des exigences fonctionnelles, il n'est pas difficile de rédiger le programme correspondant. Il n'est pas question d'écrire des fonctions qui ne sont pas demandées par les exigences. Ces efforts inutiles sont considérés comme à bannir dans le monde industriel, et notamment dans le monde aéronautique. Une fonction logicielle qui n'est pas la réponse à une exigence est une fonction qui ne pourra pas être testée ou seulement en partie. Et un logiciel non testé est un risque grave.

Puisque nous sommes dans le monde Arduino, nous allons retrouver dans le module principal les deux fonctions obligatoires habituelles. Il nous faut en outre un module pour définir des variables globales afin de mémoriser des états et des durées. La fonction de boucle `loop()` va réaliser de façon répétée une série d'opérations : surveillance de l'état des deux interrupteurs d'armement et de périmètre de sécurité, allumage et extinction des diodes LED en conséquence et gestion du compte à rebours sans cesser de surveiller l'état de l'interrupteur de périmètre de sécurité. Les fonctions qui sont appelées par la boucle principale sont présentées dans le diagramme de la Figure 13.12.

Nous allons nous servir des repères numériques dans les petits carrés du schéma.

Au niveau du repère **1**, nous commençons par lire les deux interrupteurs rotatifs, celui de choix de ligne et celui du délai. Si le délai a été modifié depuis la dernière lecture et si le compte à rebours est en cours, nous abandonnons en désarmant le système et en entrant dans l'état d'arrêt.

Au niveau du repère **2**, nous testons l'état de l'interrupteur d'armement de sécurité. S'il est passé de l'état OFF à l'état ON, nous armons le drapeau correspondant et mettons sous tension le relais de sécurité (voir la Figure 13.9). Si le drapeau est armé, nous testons si un décompte est en cours. Si c'est le cas, nous ne testons pas le bouton de lancement du décompte (repère **3**). Dans le cas contraire, nous testons si l'utilisateur a enfoncé le bouton de démarrage. Si c'est le cas, nous basculons dans l'état démarrage et commençons à décompter.

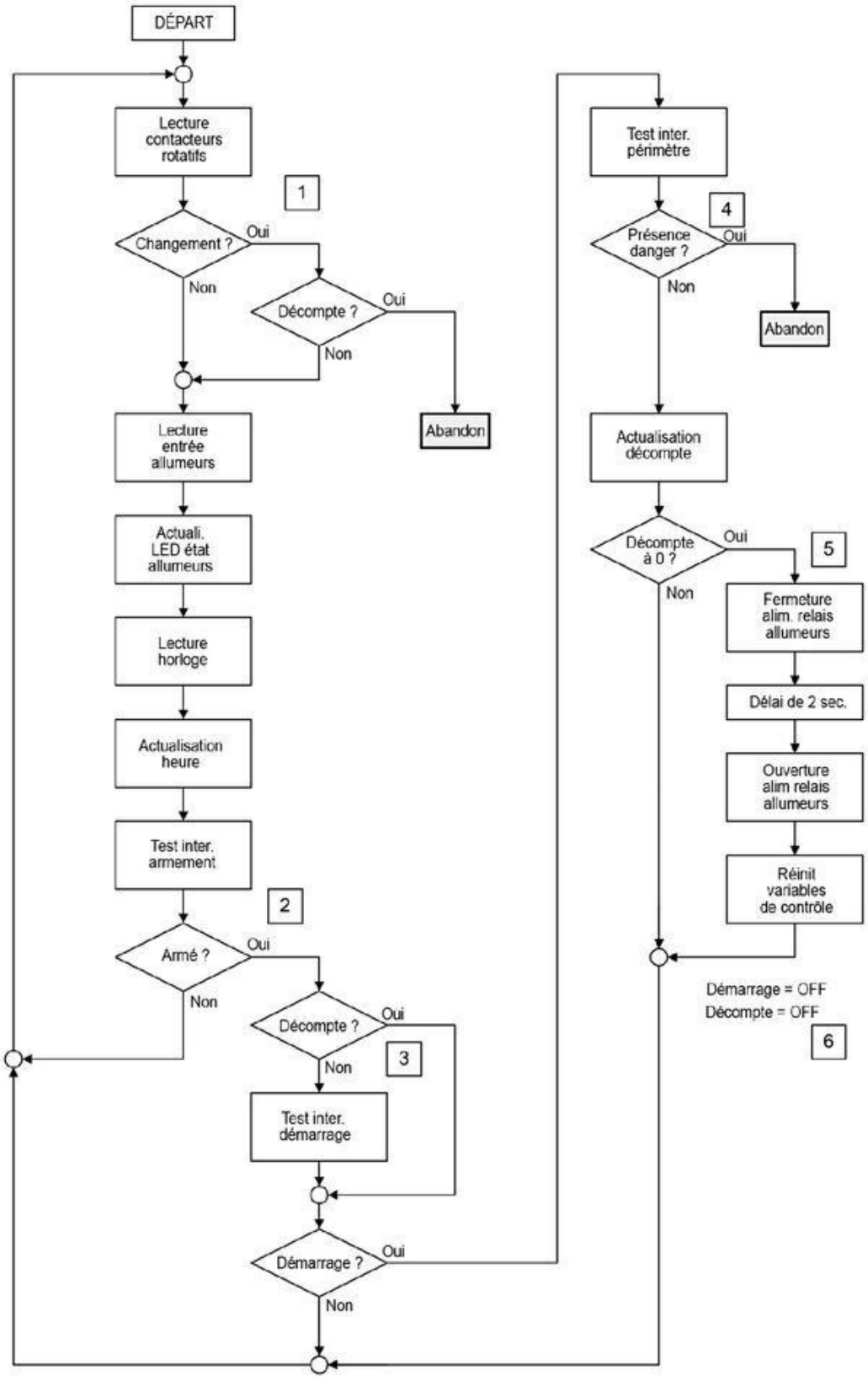


Figure 13.12 : Diagramme des fonctions de la boucle principale.

La colonne de droite de la Figure 13.12 n’est parcourue que lorsque le système est dans l’état démarrage et qu’un décompte est en cours. Au niveau du repère **4**, nous vérifions que le poussoir de périmètre de sécurité est enfoncé. Dans le cas contraire, nous abandonnons.

C’est à partir du repère **5** que nous agissons vraiment. Dès que le décompte atteint zéro, le relais correspondant au circuit sélectionné est alimenté pendant deux secondes pour alimenter l’allumeur. Ce n’est pas la façon idéale de procéder, car il vaudrait mieux tester l’état des allumeurs et couper l’alimentation des relais si les lignes sont ouvertes. Le temps de fermeture ne serait plus qu’un temps maximal avant de basculer en erreur.

Au niveau du repère **6**, soit l’allumage a réussi, soit pas. Dans les deux cas, nous ramenons le système dans un état d’attente, nous coupons l’alimentation des relais ainsi que celle du relais de sécurité. Pour plus de sécurité, nous pourrions même basculer le système en mode Désarmé, afin d’obliger l’utilisateur à tourner la clé d’armement en position Arrêt avant de la tourner à nouveau en position Marche afin de réarmer le tout.

Les différentes actions que nous venons de passer en revue sont parcourues très rapidement par le logiciel : lecture des interrupteurs, changement d’état des diodes LED, lecture de l’horloge RTC et contrôle des relais. Autrement dit, nous n’avons pas vraiment besoin d’utiliser des interruptions dans ce projet. Pourtant, vous pourriez envisager de gérer une interruption pour l’interrupteur de périmètre de sécurité. Voilà pourquoi j’ai choisi de le connecter directement au microcontrôleur, au lieu de le laisser traverser la carte d’extension d’entrées-sorties.

En effet, l’interrupteur de périmètre de sécurité peut servir à déclencher une interruption qui va provoquer l’ouverture du relais de sécurité des allumeurs (le relais 3 du second module). Autrement dit, même s’il faut plusieurs dizaines de millisecondes au microcontrôleur pour détecter le nouvel état de l’interrupteur et forcer le système à l’abandon, nous aurons déjà coupé l’alimentation des relais des allumeurs.

L’afficheur LCD peut servir à présenter les messages pendant le compte à rebours, et notamment l’état actuel. Il pourra aussi servir à afficher des messages d’erreur. En revanche, il est inutile d’y répéter l’heure actuelle ou le compte à rebours, car cela ne ferait qu’ajouter du retard dans la boucle principale.

Pour écrire le code source, laissez-vous guider par la liste des exigences fonctionnelles. J’ai donné dans les trois précédents chapitres des exemples de

modularisation des actions dans le code source.

La simple étude de la Figure 13.12 permet de repérer des portions qui peuvent devenir des modules. C'est par exemple le cas de la partie assurant la lecture et la mise à jour de l'état des allumeurs, tout comme la partie de gestion du temps et de son affichage. Un autre candidat est la partie d'allumage entre les repères **4** et **6**. Vous allez sans doute utiliser des classes pour le module d'horloge temps réel RTC, le module d'afficheur LCD, les afficheurs à diodes LED et le module d'extension d'entrées-sorties. En revanche, vous n'aurez sans doute pas besoin de définir des classes pour la boucle principale. Le code n'est pas très complexe, après tout.

Comme indiqué en début de chapitre, je m'arrête au niveau de la conception détaillée et vous laisse libre de rédiger le code source complet si vous désirez concrétiser ce projet. N'hésitez pas à vous inspirer des exemples des chapitres précédents. Grâce à l'analyse détaillée, vous ne devriez pas avoir de difficultés à terminer le projet vous-même.

Il m'est impossible de vous donner une estimation réaliste du temps qu'il faut pour terminer le logiciel. Chacun travaille à son rythme ; ce qu'une personne peut écrire en une heure pourra prendre une demi-journée à une autre. Pour avoir une première estimation, je vous conseille d'écrire un peu de code de test avec un prototype pour essayer de gérer la partie horloge temps réel ou affichage LCD. Cela va vous donner une première estimation, qu'il suffit de multiplier par le nombre de fonctions différentes. Multipliez enfin le tout par deux. Le chiffre final ne devrait pas être trop éloigné du temps que vous y consacrerez réellement, et si vous avez fini plus tôt, tant mieux pour vous.

Test et utilisation

Il est vivement conseillé de prévoir un plan et des procédures de test. Pour ce projet, les tests restent simples, puisqu'il ne s'agit que d'interrupteurs et de relais. Vous pouvez simuler la continuité d'une ligne d'allumeur en insérant un cavalier entre les deux bornes de la sortie de test. Un circuit sera détecté comme ouvert lorsqu'il n'y a pas de cavalier.

Si vous relisez les exigences fonctionnelles, vous constaterez qu'elles sont toutes vérifiables. Autrement dit, si vous manipulez le contacteur rotatif de sélection de circuit, vous devez voir s'allumer le nombre de diodes LED correspondant, entre une et six. Si vous avez installé des cavaliers sur les prises de test, la lumière de

la diode ICC GO doit briller. Si vous armez le système avec la clé puis maintenez enfoncé le poussoir de périmètre de sécurité, vous devez voir s’allumer la diode Safety GO. Si vous démarrez le décompte puis déconnectez l’une des lignes d’allumeur, changez le nombre de circuits ou le délai, ou relâchez le poussoir de périmètre, le système doit arrêter de décompter et basculer en mode Sécurité.

Si vous avez fermé les lignes des allumeurs avec des cavaliers, ne laissez pas le décompte atteindre zéro. Lorsque les relais se ferment, tout le courant de la batterie va passer, ce qui peut faire fondre le cavalier, endommager les relais ou les deux. Mieux vaut utiliser des fusibles ou si possible de vrais allumeurs posés à l’écart sur une plaque de bois. Vous pouvez aussi installer de petites ampoules électriques avec le voltage adéquat. Il faut dans ce cas que leur résistance à froid soit suffisante, en sachant que la résistance habituelle d’un allumeur est d’environ 0,5 ohm.

Analyse financière

Le coût total du projet va dépendre de vos sources d'approvisionnement. Un des éléments les plus coûteux est le châssis. Vous pouvez fabriquer le vôtre en bois, mais cela suppose de disposer des outils de menuisier appropriés. Si vous comptez le temps passé, il peut être plus économique d'acheter le châssis.

À partir de la nomenclature de composants du Tableau 13.5, le prix de revient de la Phase 1 va tourner aux alentours de 120 euros. Si vous voulez réaliser le projet complet de la Phase 2, prévoyez plus de 300 euros, mais vous pouvez réduire les coûts en choisissant bien vos sources d'approvisionnement.

ANNEXE A
Outils et accessoires

Dans la plupart des projets Arduino, vous n’avez pas besoin d’outils en particulier : quelques fils et straps, votre carte Arduino, un ou plusieurs boucliers et modules de capteurs, un point c’est tout. En revanche, lorsque vous allez vous lancer dans des projets plus sophistiqués, vous allez constater qu’il est indispensable d’avoir une bonne trousse à outils. Il vous faut quelques pinces et tournevis, un bon fer à souder, et quelques autres accessoires. Nous avons abordé dans les précédents chapitres les grandes lignes de la fabrication d’un circuit imprimé et donné quelques remarques concernant la visserie.

Je vous propose dans cette annexe de faire un tour d’horizon des outils absolument indispensables pour réaliser vos projets. Toute cette panoplie tient aisément dans une boîte à outils de taille normale.

L’outil fait le forgeron

Les principales opérations que vous pourrez avoir besoin de réaliser avec vos mains sont les suivantes : couper, scier, sertir, visser et dévisser, plier et dénuder. Avant l’apparition de l’électricité, le seul moyen de fabriquer quelque chose consistait à utiliser avec dextérité des outils à main. Songez à tous ces superbes objets qui ont survécu, que ce soit en joaillerie ou en armurerie. Vous aussi pouvez devenir un expert dans le maniement de ces outils, à condition de prendre votre temps. Je ne présenterai aucune technique dans cette annexe, car des livres sont consacrés à ce sujet. Voyons quels outils vont répondre aux besoins et où les trouver.

Tournevis

Dans la plupart des projets Arduino, vous aurez surtout besoin de quelques tournevis miniatures, un peu comme ceux visibles dans la Figure A.1, ainsi que deux ou trois tournevis d’électricien. Pour les tournevis miniatures, vous les trouverez dans les magasins d’électronique, et même dans les quincailleries.

Figure A.1 : Jeu de tournevis miniatures.

Les tournevis d’électricien (Figure A.2) sont faciles à trouver dans n’importe quelle quincaillerie. Évitez les gros modèles, et cherchez de préférence des tournevis cruciformes. Les gros tournevis sont plutôt destinés à l’électronique de puissance et à la réparation automobile.

Figure A.2 : Quelques tournevis d’électricien.

Plusieurs sortes de pinces

Il nous faut minimum une pince à becs fins, une pince coupante normale et une de précision (*flush*). En option, songez à une pince à becs recourbés. La figure A.3 montre quelques exemples de pinces.

Figure A.3 : Quelques pinces pour travaux d'électronique.

Évitez les pinces coupantes de grande taille, comme celles vendues dans le rayon électricité des quincailleries. Il existe des pinces spéciales pour couper les pattes des composants à ras. Voyez par exemple celle de la Figure A.4.

Figure A.4 : Une pince coupante d’électronicien.

Pince à dénuder

La pince à dénuder est absolument indispensable. Vous pourriez être tenté de vous en passer en dénudant vos fils avec une simple pince coupante, mais c’est tout à fait déconseillé. Vous allez facilement blesser un des brins du fil. Si ensuite le fil est plié à cet endroit, il va finir par se rompre. J’aime avoir à disposition deux pinces à dénuder, et je choisis l’une ou l’autre en fonction du diamètre du fil que j’ai à dénuder.

La version la plus simple est dotée d’une butée d’arrêt réglable (Figure A.5). Le souci de ce modèle est qu’il oblige à modifier le réglage à chaque fois que vous changez de diamètre de fil. Si vous n’en changez jamais, cela ne pose pas de problème. J’utilise par exemple toujours du fil de câblage de Jauge 24.

Figure A.5 : Une pince à dénuder simple.

Mon autre pince à dénuder, fabriquée par Klein, est bien plus sophistiquée. Elle sait gérer plusieurs diamètres de fil, mais également enlever l’isolant, le tout en un seul geste. Cette pince de luxe est visible dans la Figure A.6. Le prix n’est pas déraisonnable, et vous pouvez acheter des lames supplémentaires pour pouvoir

traiter encore plus de diamètres. Le seul inconvénient est que la pince est assez encombrante.

Outils de sertissage de cosses

Lors du montage d'un projet à partir d'une carte Arduino, d'un bouclier et de quelques modules, la principale opération est l'interconnexion. Les fils volants munis de broches mâles ou femelles, appelés straps, sont suffisants pour monter un prototype, mais ils présentent des faiblesses à long terme. Il est préférable d'utiliser un bouclier d'extension d'entrées-sorties (j'en ai décrit dans le Chapitre 9), car il propose des connecteurs multifils (Figure A.7).

Figure A.6 : Pince à dénuder professionnelle.

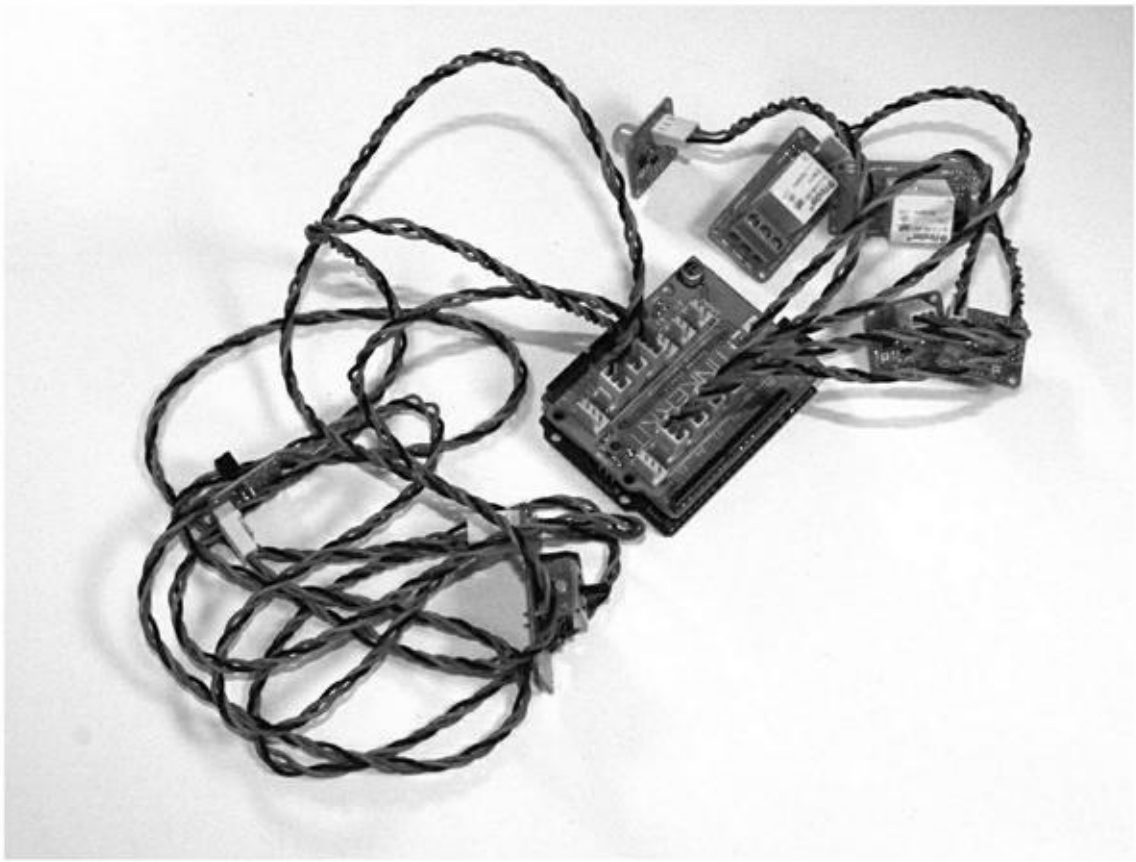


Figure A.7 : Bouclier d’extension d’entrées-sorties avec des connecteurs.

Lorsque les fils se terminent par des broches métalliques, il faut un outil spécial pour les insérer dans les corps des connecteurs. Par bonheur, le prix de ces outils a énormément baissé, puisque vous en trouverez à partir de 30 euros. La Figure A.8 montre quelques pinces à sertir.

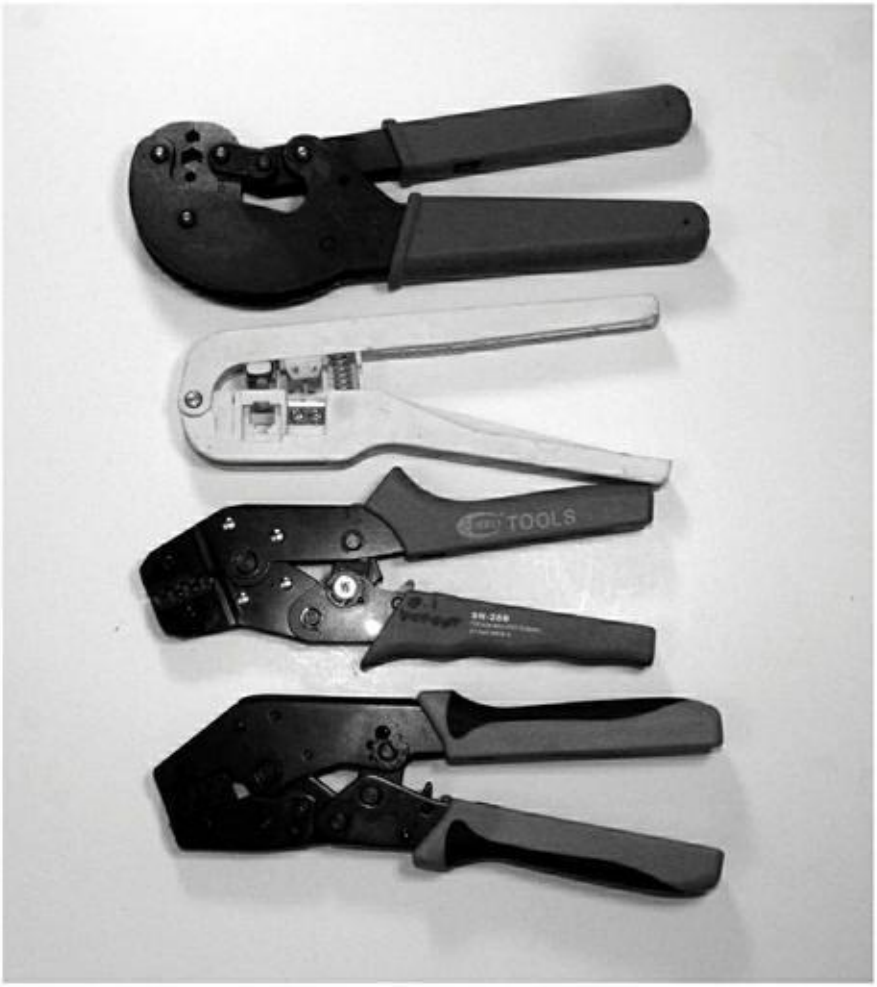


Figure A.8 : Quelques exemples de pinces à sertir économiques.

Une fois que la broche mâle ou femelle est sertie sur le fil, il ne reste qu'à l'insérer dans le corps du connecteur. Ce genre de connecteur se présente généralement avec un espacement de 2,54 mm, qui est devenu le standard dans l'industrie et dans le monde Arduino. La Figure A.9 montre des séries de 1, 2, 3 et 4 connecteurs. La cosse est verrouillée dans le boîtier en plastique et peut facilement être retirée en tirant sur un levier de verrouillage avec un petit tournevis.

Figure A.9 : Quelques connecteurs pour broches à espacement de 2,54 mm.

Certains boucliers et modules pour Arduino possèdent des connecteurs qui ressemblent à ceux des câbles réseaux et téléphoniques. Pour fabriquer de tels connecteurs, il faut un outil spécial qui est disponible dans les quincailleries et magasins d'électronique. La Figure A.10 montre quelques-uns de ces connecteurs au format RJ-45.

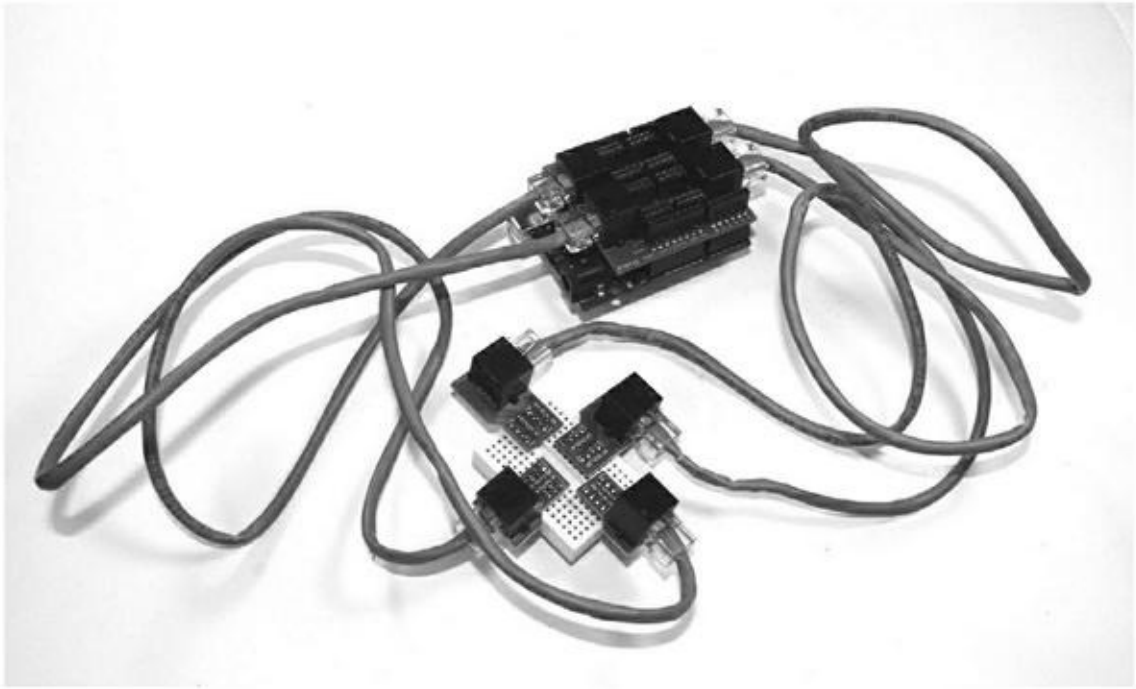


Figure A.10 : Bouclier d'extension avec connecteurs RJ-45 (8C8P).

Les connexions peuvent également être faites avec les mêmes cosses que l'on trouve dans les véhicules et les appareils électriques (Figure A.11). Ce genre de cosse n'est pas encore très utilisé dans le monde Arduino, mais j'en ai utilisé

dans mon générateur de signal du Chapitre 11. Il existe des cosses à œil et d'autres à fourche (voir dans Figure A.11).

Figure A.11 : Exemple de cosse à fourche.

La figure A.12 montre une pince à sertir les cosses. Elle ne peut pas être utilisée pour les connecteurs à broches miniatures que l'on trouve sur les circuits imprimés de type Arduino. Le résultat serait un connecteur inutilisable.

Figure A.12 : Exemple de pinces à sertir les cosses.

Les connexions par cosses sont simples à réaliser et très fiables si elles sont faites correctement. Elles sont aussi très économiques. Le seul inconvénient est qu'il faut investir dans une bonne pince à cosses. Si vous choisissez cette voie, vous aurez rarement besoin d'utiliser votre fer à souder et les connexions de vos projets offriront un bel aspect professionnel.

Scies

Vous aurez de temps à autre besoin de débiter un morceau dans une plaque à trous, de retoucher un boîtier en matière plastique ou de couper un bout de tube. Il nous faut dans ce cas une scie.

La première sorte de scie qui vous sera utile est la scie à chantourner (Figure A.13). Elle permet de travailler très précisément, mais pas de couper de grandes longueurs. Il ne faut jamais forcer lorsque l'on scie avec cet outil (comme d'ailleurs avec toutes les scies). Les lames sont assez fragiles ; elles permettent néanmoins de couper quasiment n'importe quoi si vous y mettez assez de patience.

Figure A.13 : Scie à chantourner.

Pour les travaux de plus grande taille, notamment s'il s'agit de métal, vous utiliserez la scie à métaux. Malgré son nom, elle n'est pas limitée à cette matière (Figure A.14). Les modèles les plus récents ont un aspect mieux profilé que celui de la figure, mais le principe reste le même. Il existe même des scies à métaux qui se résument à une lame avec une poignée de chaque côté.

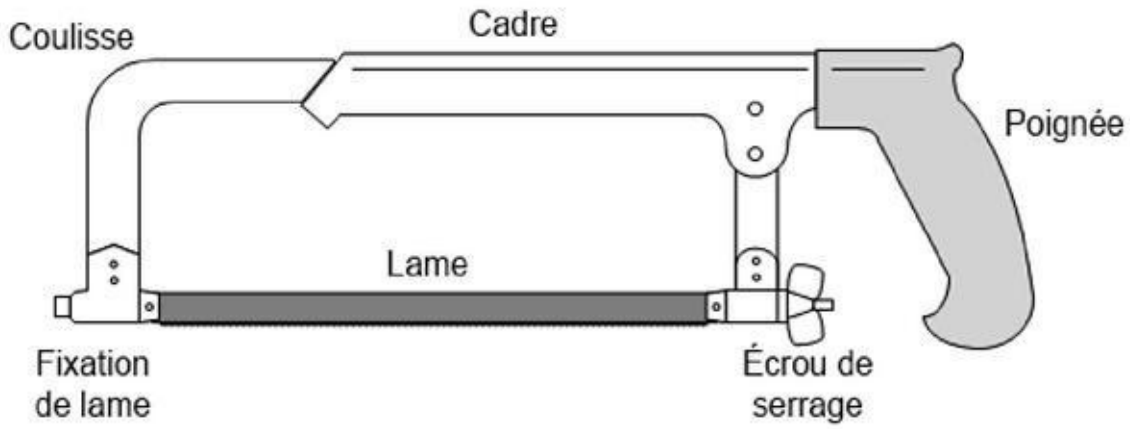


Figure A.14 : Exemple de scie à métaux.

Sachez qu'avec une scie à métaux, la lame ne coupe que dans un sens, soit en tirant, soit en poussant, tout dépendant du sens dans lequel la lame a été installée. Personnellement, je préfère tirer sur la lame, mais de nombreuses personnes préfèrent pousser.

Outils électriques

Pour la plupart de vos travaux, vous pourrez vous contenter de vos outils à main, mais dans certains cas, il ne sera pas inutile de disposer d’un peu plus de puissance, afin de s’éviter quelques crampes. C’est notamment le cas du perçage et du meulage.

Perceuse

La perceuse électrique de bricolage est très pratique, mais ce n’est pas l’outil idéal pour effectuer des perçages précis dans de petits diamètres. Bien sûr, si vous n’avez besoin de percer qu’un seul trou de 4 mm, et que la précision n’est pas essentielle, ce genre de perceuse vous suffira. Je conseille cependant d’adopter une perceuse sans fil, comme celle de la Figure A.15. La puissance d’une telle perceuse est inférieure à celles alimentées par le secteur, mais ce n’est pas un problème vu le genre de perçage que vous aurez à réaliser. Vous allez principalement percer du plastique, et de fines plaques de bois ou de métal. La perceuse sans fil conviendra parfaitement.

Figure A.15 : Perceuse sans fil avec accumulateur amovible.

Meuleuse miniature

La meuleuse n’est pas indispensable, mais elle peut s’avérer utile. La Figure A.16 montre une meuleuse miniature. Elle permet d’ébavurer des tiges métalliques venant d’être coupées, d’aiguiser des tournevis, d’adoucir les bords des pièces en plastique et même de revoir les dimensions extérieures d’un circuit imprimé.

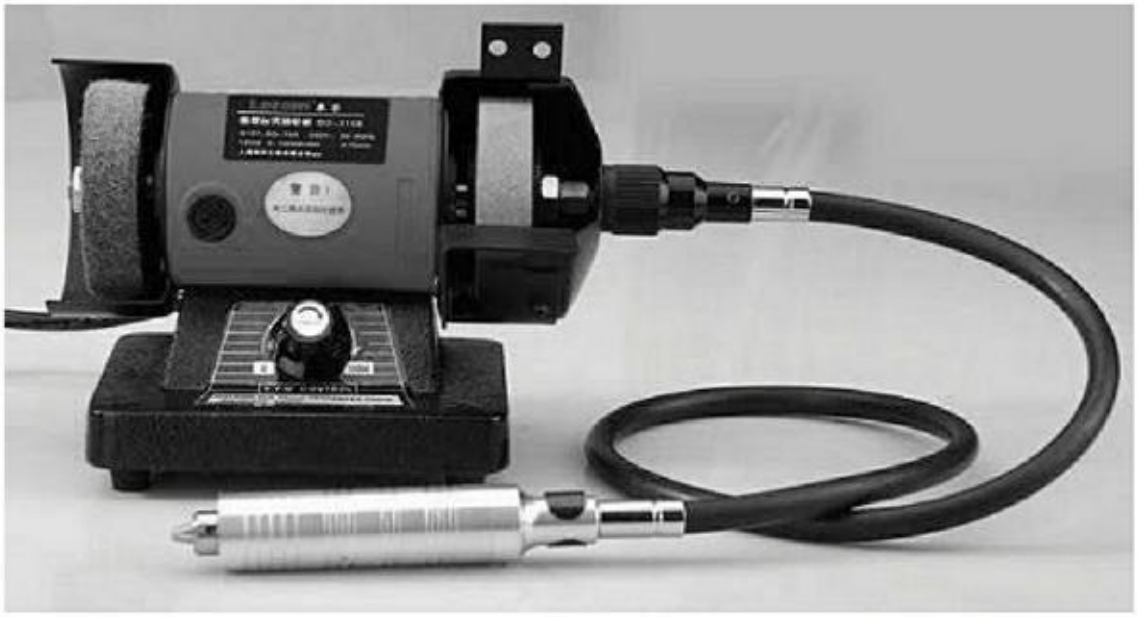


Figure A.16 : Meuleuse miniature avec flexible.

La meuleuse de la figure est dotée d’un flexible pour accéder aisément à la pièce à usiner. La puissance d’une version miniature suffit amplement aux besoins des électroniciens.

Perceuse à colonne miniature

Lorsque vous avez besoin d’effectuer des perçages de précision, il vous faut absolument une perceuse stable, montée sur colonne. Il est pour autant inutile d’encombrer votre atelier avec une perceuse à colonne de taille standard. La solution est le modèle miniature (Figure A.17).

Figure A.17 : Perceuse à colonne miniature.

Avec ce genre de perceuse, vous allez pouvoir effectuer les perçages pour les interrupteurs et les diodes LED dans les boîtiers, mais également faire des trous dans les circuits imprimés. Ce genre d’outil est en général livré avec des pinces pour maintenir la pièce à usiner pendant le travail.

Soudure/brasure

S’il y a bien une activité manuelle qui symbolise l’électronique, c’est la soudure. Pourtant, vous n’aurez pas obligatoirement besoin de faire des soudures si vous utilisez des modules et des circuits prémontés avec votre carte Arduino. En revanche, vous aurez besoin de souder dès que vous voudrez faire des travaux sur mesure ou de plus grande taille. Enfin, il vous arrivera d’acheter un bouclier dont les connecteurs ne sont pas encore soudés. Dans ce cas, il faudra faire chauffer le fer !

Fer à souder

Pour le choix du fer à souder, il faut trouver un juste milieu entre les fers bas de gamme, sans réglage de température, et dont la panne est de mauvaise qualité, et les stations de soudure avec pannes interchangeables et contrôle électronique de la température, qui coûte largement plus de 100 euros. Évitez le matériel bas de gamme qui peut endommager vos projets, et notamment les composants les plus sensibles. Si vous n’avez pas besoin de souder de circuits intégrés, mais seulement des connecteurs et des fils, un fer de milieu de gamme tel que celui de la Figure A.18 pourra suffire (environ 15 euros).

Figure A.18 : Fer à souder de milieu de gamme sans réglage.

Si vous pouvez vous le permettre, optez pour une station de soudage comme celle de la Figure A.19. Vous en trouverez à partir de 50 euros, mais les prix peuvent monter jusqu'à 300. C'est toujours un bon investissement. Sachez qu'avec les nouvelles réglementations qui bannissent le plomb, le fil à souder doit être porté à une température qui n'est souvent inférieure que de 10 ^{o}C de la température à partir de laquelle certains composants actifs sensibles commencent à subir des dégâts. Un réglage précis de la température est donc le bienvenu !

Figure A.19 : Station de soudage.

Accessoires de soudure

En plus de votre fer, il vous faut des consommables et des accessoires. Pour le fil de soudure, n’achetez pas du fil de gros diamètre destiné aux travaux d’électricité. Il faut de la soudure pour électronique, qui contient normalement une âme en résine qui dégage un flux de soudure en chauffant. Personnellement, j’achète mon fil par bobine d’un demi-kilo, comme celle de la Figure A.20.

Figure A.20 : Une bobine de fil de soudure avec âme en résine.

Il vous faudra également de la tresse à dessouder en cuivre, du flux de soudage en liquide ou pâte, et de la pâte à souder. Tous ces accessoires sont disponibles dans les mêmes boutiques que les fers.

Sources d’approvisionnement

Le tableau suivant propose quelques adresses Web de fournisseurs d’outils pour électronique. Reportez-vous également à l’Annexe C qui donne de nombreuses adresses, notamment dans le monde francophone.

Tableau A.1 : Quelques fournisseurs d’outils.

Fournisseur	Adresse URL	Fournisseur	Adresse U
Adafruit	www.adafruit.com	Maker Shed	www.mak
Apex Tool Group	www.apexhandtools.com	MCMElectronics	www.mcm
CKBProducts	www.ckbproducts.com	SainSmart	www.sains
Circuit Specialists	www.circuitspecialists.com	SparkFun	www.sparl
Electronic Goldmine	www.goldmine-elec-products.com	Stanley	www.stanl
Harbor Freight Tools	www.harborfreight.com	Velleman	www.velle

Enfin, n’hésitez pas à consulter les sites de petites annonces sur le Web. Il arrive que des amateurs se séparent de tout leur outillage.

ANNEXE B
Registres de contrôle AVR

Les listes de registres de cette annexe constituent une référence abrégée. Pour la description détaillée de chaque registre, vous consulterez la documentation Atmel de chaque microcontrôleur. Tenez bien compte des notes de bas de page dans les documents Atmel. Les différences entre les modèles de contrôleurs sont subtiles.

En général, les bits marqués **Réservé** (avec un tiret) ne doivent pas être lus. Les registres de 0x00 à 0x1F sont accessibles bit par bit grâce aux instructions SBI et CBI (armement du bit d'entrée-sortie Set et inhibition du même bit Clear). Les adresses indiquées entre parenthèses sont celles en mémoire vive SRAM ; celles sans parenthèses sont celles dans l'espace de 64 octets réservé aux registres de contrôle I/O ; ces dernières sont accessibles avec les instructions IN et OUT. En revanche, celles dans la mémoire SRAM demandent les instructions ST/STS/STD et LD/LDS/LDD.

Les informations présentées sont tirées des documents officiels Atmel suivants, disponibles sur le site atmel.com :

Référence du document	Titre
Atmel-8271I-AVR-ATmega-Datasheet_10/2014	Atmel ATmega48A/PA/88A/PA/168A/PA/328/P
2549Q–AVR–02/2014	Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V
7766F–AVR–11/10	Atmel ATmega16U4/ATmega32U4

Lexique pour l’édition française

Les trois grands tableaux qui suivent, un par modèle de microcontrôleur, contiennent des noms de registres que nous avons décidé de conserver en langue anglaise. Pourquoi ? Parce que ces noms sont ceux qui servent à baptiser les registres, en général, à partir des initiales des noms complets. Voici un exemple :

```
OCR1BH  Timer/Counter1: Output Compare Register B High Byte
```

Il s’agit du registre qui contient l’octet (Byte) de poids fort (l’octet Haut, *High*) du registre de comparaison de sortie B du timer/compteur 1.

Le nom du registre, `OCR1BH`, reprend bien les initiales de l’expression anglaise.

Pour tirer le meilleur profit des pages suivantes, voici les quelques termes qu’il est utile de découvrir en français.

- **I/O** : Entrées-sorties
- **Low Byte** : Octet de poids faible (le moins significatif)
- **High Byte** : Octet de poids fort (le plus significatif)
- **Baud Rate Register** : Registre de réglage du débit en bauds
- **2-wire Serial Interface Data Register :** Registre de données de l’interface TWI
- **Output Compare Register** : Registre de comparaison de sortie
- **Input Capture Register :** Registre de capture d’entrée
- **Oscillator Calibration Register** : Registre de calibrage de l’oscillateur
- **Address Register :** Registre d’adresse
- **General Purpose I/O Register :** Registre d’entrées-sorties à usage général

ATmega168/328

Adresse	Nom	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3
(0xFF)	Réservé	–	–	–	–	–
(0xFE)	Réservé	–	–	–	–	–
(0xFD)	Réservé	–	–	–	–	–
(0xFC)	Réservé	–	–	–	–	–
(0xFB)	Réservé	–	–	–	–	–
(0xFA)	Réservé	–	–	–	–	–
(0xF9)	Réservé	–	–	–	–	–
(0xF8)	Réservé	–	–	–	–	–
(0xF7)	Réservé	–	–	–	–	–
(0xF6)	Réservé	–	–	–	–	–
(0xF5)	Réservé	–	–	–	–	–
(0xF4)	Réservé	–	–	–	–	–
(0xF3)	Réservé	–	–	–	–	–
(0xF2)	Réservé	–	–	–	–	–
(0xF1)	Réservé	–	–	–	–	–
(0xF0)	Réservé	–	–	–	–	–
(0xEF)	Réservé	–	–	–	–	–
(0xEE)	Réservé	–	–	–	–	–

(0xED)	Réservé	–	–	–	–	–
(0xEC)	Réservé	–	–	–	–	–
(0xEB)	Réservé	–	–	–	–	–
(0xEA)	Réservé	–	–	–	–	–
(0xE9)	Réservé	–	–	–	–	–
(0xE8)	Réservé	–	–	–	–	–
(0xE7)	Réservé	–	–	–	–	–
(0xE6)	Réservé	–	–	–	–	–
(0xE5)	Réservé	–	–	–	–	–
(0xE4)	Réservé	–	–	–	–	–
(0xE3)	Réservé	–	–	–	–	–
(0xE2)	Réservé	–	–	–	–	–
(0xE1)	Réservé	–	–	–	–	–
(0xE0)	Réservé	–	–	–	–	–
(0xDF)	Réservé	–	–	–	–	–
(0xDE)	Réservé	–	–	–	–	–
(0xDD)	Réservé	–	–	–	–	–
(0xDC)	Réservé	–	–	–	–	–
(0xDB)	Réservé	–	–	–	–	–
(0xDA)	Réservé	–	–	–	–	–
(0xD9)	Réservé	–	–	–	–	–

(0xD8)	Réservé	–	–	–	–	–
(0xD7)	Réservé	–	–	–	–	–
(0xD6)	Réservé	–	–	–	–	–
(0xD5)	Réservé	–	–	–	–	–
(0xD4)	Réservé	–	–	–	–	–
(0xD3)	Réservé	–	–	–	–	–
(0xD2)	Réservé	–	–	–	–	–
(0xD1)	Réservé	–	–	–	–	–
(0xD0)	Réservé	–	–	–	–	–
(0xCF)	Réservé	–	–	–	–	–
(0xCE)	Réservé	–	–	–	–	–
(0xCD)	Réservé	–	–	–	–	–
(0xCC)	Réservé	–	–	–	–	–
(0xCB)	Réservé	–	–	–	–	–
(0xCA)	Réservé	–	–	–	–	–
(0xC9)	Réservé	–	–	–	–	–
(0xC8)	Réservé	–	–	–	–	–
(0xC7)	Réservé	–	–	–	–	–
(0xC6)	UDR0	USART I/O Data Register				
(0xC5)	UBRR0H	USART Baud Rate Register High				
(0xC4)	UBRR0L					

		USART Baud Rate Register Low				
(0xC3)	Réservé	–	–	–	–	–
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0
(0xBF)	Réservé	–	–	–	–	–
(0xBE)	Réservé	–	–	–	–	–
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC
(0xBB)	TWDR	2-wire Serial Interface Data Register				
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3
(0xB8)	TWBR	2-wire Serial Interface Bit Rate Register				
(0xB7)	Réservé	–	–	–	–	–
(0xB6)	ASSR	–	EXCLK	AS2	TCN2UB	OCR2AU
(0xB5)	Réservé	–	–	–	–	–

(0xB4)	OCR2B	Timer/Counter2 Output Compare Register B				
(0xB3)	OCR2A	Timer/Counter2 Output Compare Register A				
(0xB2)	TCNT2	Timer/Counter2 (8-bit)				
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–
(0xAF)	Réservé	–	–	–	–	–
(0xAE)	Réservé	–	–	–	–	–
(0xAD)	Réservé	–	–	–	–	–
(0xAC)	Réservé	–	–	–	–	–
(0xAB)	Réservé	–	–	–	–	–
(0xAA)	Réservé	–	–	–	–	–
(0xA9)	Réservé	–	–	–	–	–
(0xA8)	Réservé	–	–	–	–	–
(0xA7)	Réservé	–	–	–	–	–
(0xA6)	Réservé	–	–	–	–	–
(0xA5)	Réservé	–	–	–	–	–

(0xA4)	Réservé	–	–	–	–	–
(0xA3)	Réservé	–	–	–	–	–
(0xA2)	Réservé	–	–	–	–	–
(0xA1)	Réservé	–	–	–	–	–
(0xA0)	Réservé	–	–	–	–	–
(0x9F)	Réservé	–	–	–	–	–
(0x9E)	Réservé	–	–	–	–	–
(0x9D)	Réservé	–	–	–	–	–
(0x9C)	Réservé	–	–	–	–	–
(0x9B)	Réservé	–	–	–	–	–
(0x9A)	Réservé	–	–	–	–	–
(0x99)	Réservé	–	–	–	–	–
(0x98)	Réservé	–	–	–	–	–
(0x97)	Réservé	–	–	–	–	–
(0x96)	Réservé	–	–	–	–	–
(0x95)	Réservé	–	–	–	–	–
(0x94)	Réservé	–	–	–	–	–
(0x93)	Réservé	–	–	–	–	–
(0x92)	Réservé	–	–	–	–	–
(0x91)	Réservé	–	–	–	–	–
(0x90)	Réservé	–	–	–	–	–
(0x8F)	Réservé	–	–	–	–	–

(0x8E)	Réservé	–	–	–	–	–
(0x8D)	Réservé	–	–	–	–	–
(0x8C)	Réservé	–	–	–	–	–
(0x8B)	OCR1BH	Timer/Counter1:Output Compare Register B High Byte				
(0x8A)	OCR1BL	Timer/Counter1:Output Compare Register B Low Byte				
(0x89)	OCR1AH	Timer/Counter1:Output Compare Register A High Byte				
(0x88)	OCR1AL	Timer/Counter1:Output Compare Register A Low Byte				
(0x87)	ICR1H	Timer/Counter1:Input Capture Register High Byte				
(0x86)	ICR1L	Timer/Counter1:Input Capture Register Low Byte				
(0x85)	TCNT1H	Timer/Counter1:Counter Register High Byte				
(0x84)	TCNT1L	Timer/Counter1:Counter Register Low Byte				
(0x83)	Réservé	–	–	–	–	–

(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–
(0x7F)	DIDR1	–	–	–	–	–
(0x7E)	DIDR0	–	–	ADC5D	ADC4D	ADC3D
(0x7D)	Réservé	–	–	–	–	–
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3
(0x7B)	ADCSRB	–	ACME	–	–	–
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE
(0x79)	ADCH	ADCData Register High Byte				
(0x78)	ADCL	ADCData Register Low Byte				
(0x77)	Réservé	–	–	–	–	–
(0x76)	Réservé	–	–	–	–	–
(0x75)	Réservé	–	–	–	–	–
(0x74)	Réservé	–	–	–	–	–
(0x73)	Réservé	–	–	–	–	–
(0x72)	Réservé	–	–	–	–	–
(0x71)	Réservé	–	–	–	–	–
(0x70)	TIMSK2	–	–	–	–	–
(0x6F)	TIMSK1	–	–	ICIE1	–	–
(0x6E)	TIMSK0	–	–	–	–	–

(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT1
(0x6C)	PCMSK1	–	PCINT14	PCINT13	PCINT12	PCINT1
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3
(0x6A)	Réservé	–	–	–	–	–
(0x69)	EICRA	–	–	–	–	ISC11
(0x68)	PCICR	–	–	–	–	–
(0x67)	Réservé	–	–	–	–	–
(0x66)	OSCCAL	Oscillator Calibration Register				
(0x65)	Réservé	–	–	–	–	–
(0x64)	PRR	PRTWI	PRTIM2	PRTIM0	–	PRTIM1
(0x63)	Réservé	–	–	–	–	–
(0x62)	Réservé	–	–	–	–	–
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE
0x3F (0x5F)	SREG	I	T	H	S	V
0x3E (0x5E)	SPH	–	–	–	–	–
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3
0x3C (0x5C)	Réservé	–	–	–	–	–
	Réservé	–	–	–	–	–

0x3B (0x5B)						
0x3A (0x5A)	Réservé	–	–	–	–	–
0x39 (0x59)	Réservé	–	–	–	–	–
0x38 (0x58)	Réservé	–	–	–	–	–
0x37 (0x57)	SPMCSR	SPMIE	(RWWSB)	–	(RWWSRE)	BLBSET
0x36 (0x56)	Réservé	–	–	–	–	–
0x35 (0x55)	MCUCR	–	BODS	BODSE	PUD	–
0x34 (0x54)	MCUSR	–	–	–	–	WDRF
0x33 (0x53)	SMCR	–	–	–	–	SM2
0x32 (0x52)	Réservé	–	–	–	–	–
0x31 (0x51)	Réservé	–	–	–	–	–
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE

0x2F (0x4F)	Réservé	–	–	–	–	–
0x2E (0x4E)	SPDR	SPIData Register				
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL
0x2B (0x4B)	GPIOR2	General Purpose I/ORegister 2				
0x2A (0x4A)	GPIOR1	General Purpose I/ORegister 1				
0x29 (0x49)	Réservé	–	–	–	–	–
0x28 (0x48)	OCR0B	Timer/Counter0 Output Compare Register B				
0x27 (0x47)	OCR0A	Timer/Counter0 Output Compare Register A				

0x26 (0x46)	TCNT0	Timer/Counter0 (8-bit)				
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–
0x23 (0x43)	GTCCR	TSM	–	–	–	–
0x22 (0x42)	EEARH	EEPROM Address Register High Byte				
0x21 (0x41)	EEARL	EEPROM Address Register Low Byte				
0x20 (0x40)	EEDR	EEPROM Data Register				
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE
0x1E (0x3E)	GPIOR0	General Purpose I/O Register 0				
0x1D (0x3D)	EIMSK	–	–	–	–	–
0x1C (0x3C)	EIFR	–	–	–	–	–
0x1B (0x3B)	PCIFR	–	–	–	–	–

0x1A (0x3A)	Réservé	–	–	–	–	–
0x19 (0x39)	Réservé	–	–	–	–	–
0x18 (0x38)	Réservé	–	–	–	–	–
0x17 (0x37)	TIFR2	–	–	–	–	–
0x16 (0x36)	TIFR1	–	–	ICF1	–	–
0x15 (0x35)	TIFR0	–	–	–	–	–
0x14 (0x34)	Réservé	–	–	–	–	–
0x13 (0x33)	Réservé	–	–	–	–	–
0x12 (0x32)	Réservé	–	–	–	–	–
0x11 (0x31)	Réservé	–	–	–	–	–
0x10 (0x30)	Réservé	–	–	–	–	–
0x0F (0x2F)	Réservé	–	–	–	–	–

0x0E (0x2E)	Réservé	–	–	–	–	–
0x0D (0x2D)	Réservé	–	–	–	–	–
0x0C (0x2C)	Réservé	–	–	–	–	–
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3

0x02 (0x22)	Réservé	–	–	–	–	–
0x01 (0x21)	Réservé	–	–	–	–	–
0x00 (0x20)	Réservé	–	–	–	–	–

ATmega1280/2560

Adresse	Nom	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3
(0x1FF)	Réservé	–	–	–	–	–
…	Réservé	–	–	–	–	–
(0x137)	Réservé	–	–	–	–	–
(0x136)	UDR3	USART3 I/OData Register				
(0x135)	UBRR3H	–	–	–	–	USART3 I Byte
(0x134)	UBRR3L	USART3 Baud Rate Register Low Byte				
(0x133)	Réservé	–	–	–	–	–
(0x132)	UCSR3C	UMSEL31	UMSEL30	UPM31	UPM30	USBS3
(0x131)	UCSR3B	RXCIE3	TXCIE3	UDRIE3	RXEN3	TXEN3
(0x130)	UCSR3A	RXC3	TXC3	UDRE3	FE3	DOR3
(0x12F)	Réservé	–	–	–	–	–
(0x12E)	Réservé	–	–	–	–	–
(0x12D)	OCR5CH	Timer/Counter5: Output Compare Register C High Byte				
(0x12C)	OCR5CL	Timer/Counter5: Output Compare Register C Low Byte				
(0x12B)	OCR5BH					

		Timer/Counter5: Output Compare Register B High Byte
(0x12A)	OCR5BL	Timer/Counter5: Output Compare Register B Low Byte
(0x129)	OCR5AH	Timer/Counter5: Output Compare Register A High Byte
(0x128)	OCR5AL	Timer/Counter5: Output Compare Register A Low Byte
(0x127)	ICR5H	Timer/Counter5: Input Capture Register High Byte
(0x126)	ICR5L	Timer/Counter5: Input Capture Register Low Byte
(0x125)	TCNT5H	Timer/Counter5: Counter Register High Byte
(0x124)	TCNT5L	Timer/Counter5: Counter Register Low Byte
(0x123)	Réservé	– – – – –

(0x122)	TCCR5C	FOC5A	FOC5B	FOC5C	–	–
(0x121)	TCCR5B	ICNC5	ICES5	–	WGM53	WGM52
(0x120)	TCCR5A	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1
(0x11F)	Réservé	–	–	–	–	–
(0x11E)	Réservé	–	–	–	–	–
(0x11D)	Réservé	–	–	–	–	–
(0x11C)	Réservé	–	–	–	–	–
(0x11B)	Réservé	–	–	–	–	–
(0x11A)	Réservé	–	–	–	–	–
(0x119)	Réservé	–	–	–	–	–
(0x118)	Réservé	–	–	–	–	–
(0x117)	Réservé	–	–	–	–	–
(0x116)	Réservé	–	–	–	–	–
(0x115)	Réservé	–	–	–	–	–
(0x114)	Réservé	–	–	–	–	–
(0x113)	Réservé	–	–	–	–	–
(0x112)	Réservé	–	–	–	–	–
(0x111)	Réservé	–	–	–	–	–
(0x110)	Réservé	–	–	–	–	–
(0x10F)	Réservé	–	–	–	–	–
(0x10E)	Réservé	–	–	–	–	–

(0x10D)	Réservé	–	–	–	–	–
(0x10C)	Réservé	–	–	–	–	–
(0x10B)	PORTL	PORTL7	PORTL6	PORTL5	PORTL4	PORTL3
(0x10A)	DDRL	DDL7	DDL6	DDL5	DDL4	DDL3
(0x109)	PINL	PINL7	PINL6	PINL5	PINL4	PINL3
(0x108)	PORTK	PORTK7	PORTK6	PORTK5	PORTK4	PORTK3
(0x107)	DDRK	DDK7	DDK6	DDK5	DDK4	DDK3
(0x106)	PINK	PINK7	PINK6	PINK5	PINK4	PINK3
(0x105)	PORTJ	PORTJ7	PORTJ6	PORTJ5	PORTJ4	PORTJ3
(0x104)	DDRJ	DDJ7	DDJ6	DDJ5	DDJ4	DDJ3
(0x103)	PINJ	PINJ7	PINJ6	PINJ5	PINJ4	PINJ3
(0x102)	PORTH	PORTH7	PORTH6	PORTH5	PORTH4	PORTH3
(0x101)	DDRH	DDH7	DDH6	DDH5	DDH4	DDH3
(0x100)	PINH	PINH7	PINH6	PINH5	PINH4	PINH3
(0xFF)	Réservé	–	–	–	–	–
(0xFE)	Réservé	–	–	–	–	–
(0xFD)	Réservé	–	–	–	–	–
(0xFC)	Réservé	–	–	–	–	–
(0xFB)	Réservé	–	–	–	–	–
(0xFA)	Réservé	–	–	–	–	–
(0xF9)	Réservé	–	–	–	–	–

(0xF8)	Réservé	–	–	–	–	–
(0xF7)	Réservé	–	–	–	–	–
(0xF6)	Réservé	–	–	–	–	–
(0xF5)	Réservé	–	–	–	–	–
(0xF4)	Réservé	–	–	–	–	–
(0xF3)	Réservé	–	–	–	–	–
(0xF2)	Réservé	–	–	–	–	–
(0xF1)	Réservé	–	–	–	–	–
(0xF0)	Réservé	–	–	–	–	–
(0xEF)	Réservé	–	–	–	–	–
(0xEE)	Réservé	–	–	–	–	–
(0xED)	Réservé	–	–	–	–	–
(0xEC)	Réservé	–	–	–	–	–
(0xEB)	Réservé	–	–	–	–	–
(0xEA)	Réservé	–	–	–	–	–
(0xE9)	Réservé	–	–	–	–	–
(0xE8)	Réservé	–	–	–	–	–
(0xE7)	Réservé	–	–	–	–	–
(0xE6)	Réservé	–	–	–	–	–
(0xE5)	Réservé	–	–	–	–	–
(0xE4)	Réservé	–	–	–	–	–
(0xE3)	Réservé	–	–	–	–	–

(0xE2)	Réservé	–	–	–	–	–
(0xE1)	Réservé	–	–	–	–	–
(0xE0)	Réservé	–	–	–	–	–
(0xDF)	Réservé	–	–	–	–	–
(0xDE)	Réservé	–	–	–	–	–
(0xDD)	Réservé	–	–	–	–	–
(0xDC)	Réservé	–	–	–	–	–
(0xDB)	Réservé	–	–	–	–	–
(0xDA)	Réservé	–	–	–	–	–
(0xD9)	Réservé	–	–	–	–	–
(0xD8)	Réservé	–	–	–	–	–
(0xD7)	Réservé	–	–	–	–	–
(0xD6)	UDR2	USART2 I/OData Register				
(0xD5)	UBRR2H	–	––		–	USART2 I Byte
(0xD4)	UBRR2L	USART2 Baud Rate Register Low Byte				
(0xD3)	Réservé	–	–	–	–	–
(0xD2)	UCSR2C	UMSEL21	UMSEL20	UPM21	UPM20	USBS2
(0xD1)	UCSR2B	RXCIE2	TXCIE2	UDRIE2	RXEN2	TXEN2
(0xD0)	UCSR2A	RXC2	TXC2	UDRE2	FE2	DOR2

(0xCF)	Réservé	–	–	–	–	–
(0xCE)	UDR1	USART1 I/ODataRegister				
(0xCD)	UBRR1H	–	––		–	USART1 IByte
(0xCC)	UBRR1L	USART1 Baud Rate RegisterLow Byte				
(0xCB)	Réservé	–	–	–	–	–
(0xCA)	UCSR1C	UMSEL11	UMSEL10	UPM11	UPM10	USBS1
(0xC9)	UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1
(0xC8)	UCSR1A	RXC1	TXC1	UDRE1	FE1	DOR1
(0xC7)	Réservé	–	–	–	–	–
(0xC6)	UDR0	USART0 I/ODataRegister				
(0xC5)	UBRR0H	–	––		–	USART0 IByte
(0xC4)	UBRR0L	USART0 Baud Rate RegisterLow Byte				
(0xC3)	Réservé	–	–	–	–	–
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0
(0xBF)	Réservé	–	–	–	–	–

(0xBE)	Réservé	–	–	–	–	–
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC
(0xBB)	TWDR	2-wire Serial Interface Data Register				
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3
(0xB8)	TWBR	2-wire Serial Interface Bit Rate Register				
(0xB7)	Réservé	–	–	–	–	–
(0xB6)	ASSR	–	EXCLK	AS2	TCN2UB	OCR2AU
(0xB5)	Réservé	–	–	–	–	–
(0xB4)	OCR2B	Timer/Counter2 Output Compare Register B				
(0xB3)	OCR2A	Timer/Counter2 Output Compare Register A				
(0xB2)	TCNT2	Timer/Counter2 (8 Bit)				
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–
(0xAF)	Réservé	–	–	–	–	–
(0xAE)	Réservé	–	–	–	–	–
(0xAD)	OCR4CH					

		Timer/Counter4:Output Compare Register C High Byte
(0xAC)	OCR4CL	Timer/Counter4:Output Compare Register C Low Byte
(0xAB)	OCR4BH	Timer/Counter4:Output Compare Register B High Byte
(0xAA)	OCR4BL	Timer/Counter4:Output Compare Register B Low Byte
(0xA9)	OCR4AH	Timer/Counter4:Output Compare Register A High Byte
(0xA8)	OCR4AL	Timer/Counter4:Output Compare Register A Low Byte
(0xA7)	ICR4H	Timer/Counter4:Input Capture Register High Byte
(0xA6)	ICR4L	Timer/Counter4:Input Capture Register Low Byte
(0xA5)	TCNT4H	Timer/Counter4:Counter Register High Byte
(0xA4)	TCNT4L	Timer/Counter4:Counter Register Low Byte

(0xA3)	Réservé	–	–	–	–	–
(0xA2)	TCCR4C	FOC4A	FOC4B	FOC4C	–	–
(0xA1)	TCCR4B	ICNC4	ICES4	–	WGM43	WGM42
(0xA0)	TCCR4A	COM4A1	COM4A0	COM4B1	COM4B0	COM4C1
(0x9F)	Réservé	–	–	–	–	–
(0x9E)	Réservé	–	–	–	–	–
(0x9D)	OCR3CH	Timer/Counter3:Output Compare Register C High Byte				
(0x9C)	OCR3CL	Timer/Counter3: Output Compare Register C Low Byte				
(0x9B)	OCR3BH	Timer/Counter3: Output Compare Register B High Byte				
(0x9A)	OCR3BL	Timer/Counter3: Output Compare Register B Low Byte				
(0x99)	OCR3AH	Timer/Counter3: Output Compare Register A High Byte				
(0x98)	OCR3AL	Timer/Counter3: Output Compare Register A Low Byte				
(0x97)	ICR3H	Timer/Counter3: Input Capture Register High Byte				

(0x96)	ICR3L	Timer/Counter3: Input Capture Register Low Byte				
(0x95)	TCNT3H	Timer/Counter3: Counter Register High Byte				
(0x94)	TCNT3L	Timer/Counter3: Counter Register Low Byte				
(0x93)	Réservé	–	–	–	–	–
(0x92)	TCCR3C	FOC3A	FOC3B	FOC3C	–	–
(0x91)	TCCR3B	ICNC3	ICES3	–	WGM33	WGM32
(0x90)	TCCR3A	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1
(0x8F)	Réservé	–	–	–	–	–
(0x8E)	Réservé	–	–	–	–	–
(0x8D)	OCR1CH	Timer/Counter1: Output Compare Register C High Byte				
(0x8C)	OCR1CL	Timer/Counter1: Output Compare Register C Low Byte				
(0x8B)	OCR1BH	Timer/Counter1: Output Compare Register B High Byte				

(0x8A)	OCR1BL	Timer/Counter1: Output Compare Register B Low Byte				
(0x89)	OCR1AH	Timer/Counter1: Output Compare Register A High Byte				
(0x88)	OCR1AL	Timer/Counter1: Output Compare Register A Low Byte				
(0x87)	ICR1H	Timer/Counter1: Input Capture Register High Byte				
(0x86)	ICR1L	Timer/Counter1: Input Capture Register Low Byte				
(0x85)	TCNT1H	Timer/Counter1: Counter Register High Byte				
(0x84)	TCNT1L	Timer/Counter1: Counter Register Low Byte				
(0x83)	Réservé	–	–	–	–	–
(0x82)	TCCR1C	FOC1A	FOC1B	FOC1C	–	–
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1
(0x7F)	DIDR1	–	–	–	–	–
(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D
(0x7D)	DIDR2	ADC15D	ADC14D	ADC13D	ADC12D	ADC11D
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3
(0x7B)	ADCSRB	-	ACME	–	–	MUX5

(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE
(0x79)	ADCH	ADCData Register High Byte				
(0x78)	ADCL	ADCData Register Low Byte				
(0x77)	Réservé	–	–	–	–	–
(0x76)	Réservé	–	–	–	–	–
(0x75)	XMCRB	XMBK	–	–	–	–
(0x74)	XMCRA	SRE	SRL2	SRL1	SRL0	SRW11
(0x73)	TIMSK5	–	–	ICIE5	–	OCIE5C
(0x72)	TIMSK4	–	–	ICIE4	–	OCIE4C
(0x71)	TIMSK3	–	–	ICIE3	–	OCIE3C
(0x70)	TIMSK2	–	–	–	–	–
(0x6F)	TIMSK1	–	–	ICIE1	–	OCIE1C
(0x6E)	TIMSK0	–	–	–	–	–
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19
(0x6C)	PCMSK1	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3
(0x6A)	EICRB	ISC71	ISC70	ISC61	ISC60	ISC51
(0x69)	EICRA	ISC31	ISC30	ISC21	ISC20	ISC11
(0x68)	PCICR	–	–	–	–	–
(0x67)	Réservé	–	–	–	–	–
(0x66)	OSCCAL	Oscillator Calibration Register				

(0x65)	PRR1	–	–	PRTIM5	PRTIM4	PRTIM3
(0x64)	PRR0	PRTWI	PRTIM2	PRTIM0	–	PRTIM1
(0x63)	Réservé	–	–	–	–	–
(0x62)	Réservé	–	–	–	–	–
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE
0x3F (0x5F)	SREG	I	T	H	S	V
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP11
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3
0x3C (0x5C)	EIND	–	–	–	–	–
0x3B (0x5B)	RAMPZ	–	–	–	–	–
0x3A (0x5A)	Réservé	–	–	–	–	–
0x39 (0x59)	Réservé	–	–	–	–	–
0x38 (0x58)	Réservé	–	–	–	–	–
	SPMCSR	SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET

0x37 (0x57)						
0x36 (0x56)	Réservé	–	–	–	–	–
0x35 (0x55)	MCUCR	JTD	–	–	PUD	–
0x34 (0x54)	MCUSR	–	–	–	JTRF	WDRF
0x33 (0x53)	SMCR	–	–	–	–	SM2
0x32 (0x52)	Réservé	–	–	–	–	–
0x31 (0x51)	OCDR	OCDR7	OCDR6	OCDR5	OCDR4	OCDR3
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE
0x2F (0x4F)	Réservé	–	–	–	–	–
0x2E (0x4E)	SPDR	SPIData Register				
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL

0x2B (0x4B)	GPIOR2	General Purpose I/ORegister 2				
0x2A (0x4A)	GPIOR1	General Purpose I/ORegister 1				
0x29 (0x49)	Réservé	–	–	–	–	–
0x28 (0x48)	OCR0B	Timer/Counter0 Output Compare Register B				
0x27 (0x47)	OCR0A	Timer/Counter0 Output Compare Register A				
0x26 (0x46)	TCNT0	Timer/Counter0 (8 Bit)				
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–
0x23 (0x43)	GTCCR	TSM	–	–	–	–
0x22 (0x42)	EEARH	–	–	–	–	EEPROM Byte
0x21 (0x41)	EEARL	EEPROM Address Register Low Byte				

0x20 (0x40)	EEDR	EEPROM Data Register				
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE
0x1E (0x3E)	GPIOR0	General Purpose I/O Register 0				
0x1D (0x3D)	EIMSK	INT7	INT6	INT5	INT4	INT3
0x1C (0x3C)	EIFR	INTF7	INTF6	INTF5	INTF4	INTF3
0x1B (0x3B)	PCIFR	–	–	–	–	–
0x1A (0x3A)	TIFR5	–	–	ICF5	–	OCF5C
0x19 (0x39)	TIFR4	–	–	ICF4	–	OCF4C
0x18 (0x38)	TIFR3	–	–	ICF3	–	OCF3C
0x17 (0x37)	TIFR2	–	–	–	–	–
0x16 (0x36)	TIFR1	–	–	ICF1	–	OCF1C
0x15 (0x35)	TIFR0	–	–	–	–	–

0x14 (0x34)	PORTG	–	–	PORTG5	PORTG4	PORTG3
0x13 (0x33)	DDRG	–	–	DDG5	DDG4	DDG3
0x12 (0x32)	PING	–	–	PING5	PING4	PING3
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3
0x0E (0x2E)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3
0x0D (0x2D)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3
0x0C (0x2C)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3

0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3

ATmega32U4

Adresse	Nom	Bit 7	Bit 6	Bit 5	Bit 4	Bi
(0xFF)	Réservé	–	–	–	–	–
(0xFE)	Réservé	–	–	–	–	–
(0xFD)	Réservé	–	–	–	–	–
(0xFC)	Réservé	–	–	–	–	–
(0xFB)	Réservé	–	–	–	–	–
(0xFA)	Réservé	–	–	–	–	–
(0xF9)	Réservé	–	–	–	–	–
(0xF8)	Réservé	–	–	–	–	–
(0xF7)	Réservé	–	–	–	–	–
(0xF6)	Réservé	–	–	–	–	–
(0xF5)	Réservé	–	–	–	–	–
(0xF4)	UEINT	–	EPINT6:0			
(0xF3)	UEBCHX	–	–	–	–	–
(0xF2)	UEBCLX	BYCT7:0				
(0xF1)	UEDATX	DAT7:0				
(0xF0)	UEIENX	FLERRE	NAKINE	–	NAKOUTE	RX
(0xEF)	UESTA1X	–	–	–	–	–
(0xEE)	UESTA0X	CFGOK	OVERFI	UNDERFI	–	DT

(0xED)	UECFG1X	–	EPSIZE2:0	EPBK1:0	ALLOC	–
(0xEC)	UECFG0X	EPTYPE1:0	–	–	–	–
(0xEB)	UECONX	–	–	STALLRQ	STALLRQC	RS
(0xEA)	UERST	–	EPRST6:0			
(0xE9)	UENUM	–	–	–	–	–
(0xE8)	UEINTX	FIFOCON	NAKINI	RWAL	NAKOUTI	RX
(0xE7)	Réservé	–	–	–	–	–
(0xE6)	UDMFN	–	–	–	FNCERR	–
(0xE5)	UDFNUMH	–	–	–	–	–
(0xE4)	UDFNUML	FNUM7:0				
(0xE3)	UDADDR	ADDEN	UADD6:0			
(0xE2)	UDIEN	–	UPRSME	EORSME	WAKEUPE	EC
(0xE1)	UDINT	–	UPRSMI	EORSMI	WAKEUPI	EC
(0xE0)	UDCON	–	–	–	–	RS
(0xDF)	Réservé	–	–	–	–	–
(0xDE)	Réservé	–	–	–	–	–
(0xDD)	Réservé	–	–	–	–	–
(0xDC)	Réservé	–	–	–	–	–
(0xDB)	Réservé	–	–	–	–	–
(0xDA)	USBINT	–	–	–	–	–
(0xD9)	USBSTA	–	–	–	–	–

(0xD8)	USBCON	USBE	–	FRZCLK	OTGPADE	–
(0xD7)	UHWCON	–	–	–	–	–
(0xD6)	Réservé	–	–	–	–	–
(0xD5)	Réservé	–	–	–	–	–
(0xD4)	DT4	DT4H3	DT4H2	DT4H1	DT4H0	DT
(0xD3)	Réservé	–	–	–	–	–
(0xD2)	OCR4D	Timer/Counter4: Output Compare Register D				
(0xD1)	OCR4C	Timer/Counter4: Output Compare Register C				
(0xD0)	OCR4B	Timer/Counter4: Output Compare Register B				
(0xCF)	OCR4A	Timer/Counter4: Output Compare Register A				
(0xCE)	UDR1	USART1 I/ODataRegister				
(0xCD)	UBRR1H	–	–	–	–	US By
(0xCC)	UBRR1L	USART1 Baud Rate Register Low Byte				
(0xCB)	UCSR1D	–	–	–	–	–
(0xCA)	UCSR1C	UMSEL11	UMSEL10	UPM11	UPM10	US
(0xC9)	UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TX

(0xC8)	UCSR1A	RXC1	TXC1	UDRE1	FE1	DC
(0xC7)	CLKSTA	–	–	–	–	–
(0xC6)	CLKSEL1	RCCKSEL3	RCCKSEL2	RCCKSEL1	RCCKSEL0	EX
(0xC5)	CLKSEL0	RCSUT1	RCSUT0	EXSUT1	EXSUT0	RC
(0xC4)	TCCR4E	TLOCK4	ENHC4	OC4OE5	OC4OE4	OC
(0xC3)	TCCR4D	FPIE4	FPEN4	FPNC4	FPES4	FP
(0xC2)	TCCR4C	COM4A1S	COM4A0S	COM4B1S	COM4B0S	CC
(0xC1)	TCCR4B	PWM4X	PSR4	DTPS41	DTPS40	CS
(0xC0)	TCCR4A	COM4A1	COM4A0	COM4B1	COM4B0	FC
(0xBF)	TC4H	–	–	–	–	– 3.1
(0xBE)	TCNT4	Timer/Counter4: Counter Register Low Byte				
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TW
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TW
(0xBB)	TWDR	2-wire Serial Interface Data Register				
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TW
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TW
(0xB8)	TWBR	2-wire Serial Interface Bit Rate Register				
(0xB6)	Réservé	–	–	–	–	–
(0xB5)	Réservé	–	–	–	–	–

(0xB4)	Réservé	–	–	–	–	–
(0xB3)	Réservé	–	–	–	–	–
(0xB2)	Réservé	–	–	–	–	–
(0xB1)	Réservé	–	–	–	–	–
(0xB0)	Réservé	–	–	–	–	–
(0xAF)	Réservé	–	–	–	–	–
(0xAE)	Réservé	–	–	–	–	–
(0xAD)	Réservé	–	–	–	–	–
(0xAC)	Réservé	–	–	–	–	–
(0xB7)	Réservé	–	–	–	–	–
(0xAB)	Réservé	–	–	–	–	–
(0xAA)	Réservé	–	–	–	–	–
(0xA9)	Réservé	–	–	–	–	–
(0xA8)	Réservé	–	–	–	–	–
(0xA7)	Réservé	–	–	–	–	–
(0xA6)	Réservé	–	–	–	–	–
(0xA5)	Réservé	–	–	–	–	–
(0xA4)	Réservé	–	–	–	–	–
(0xA3)	Réservé	–	–	–	–	–
(0xA2)	Réservé	–	–	–	–	–
(0xA1)	Réservé	–	–	–	–	–

(0xA0)	Réservé	–	–	–	–	–
(0x9F)	Réservé	–	–	–	–	–
(0x9E)	Réservé	–	–	–	–	–
(0x9D)	OCR3CH	Timer/Counter3:Output Compare Register C Hig				
(0x9C)	OCR3CL	Timer/Counter3:Output Compare Register C Lov				
(0x9B)	OCR3BH	Timer/Counter3:Output Compare Register B Hig				
(0x9A)	OCR3BL	Timer/ Counter3: Output Compare Register B Low Byte				
(0x99)	OCR3AH	Timer/Counter3:Output Compare Register A Hig				
(0x98)	OCR3AL	Timer/Counter3:Output Compare Register A Lov				
(0x97)	ICR3H	Timer/Counter3:Input Capture Register High Byt				
(0x96)	ICR3L	Timer/Counter3:Input Capture Register Low Byt				
(0x95)	TCNT3H	Timer/Counter3:Counter Register High Byte				
(0x94)	TCNT3L	Timer/Counter3:Counter Register Low Byte				
(0x93)	Réservé	–	–	–	–	–
(0x92)	TCCR3C	FOC3A	–	–	–	–
(0x91)	TCCR3B	ICNC3	ICES3	–	WGM33	WG
(0x90)	TCCR3A	COM3A1	COM3A0	COM3B1	COM3B0	CO
(0x8F)	Réservé	–	–	–	–	–

(0x8E)	Réservé	–	–	–	–	–
(0x8D)	OCR1CH	Timer/Counter1:Output Compare Register C Hig				
(0x8C)	OCR1CL	Timer/Counter1:Output Compare Register C Lov				
(0x8B)	OCR1BH	Timer/Counter1:Output Compare Register B Hig				
(0x8A)	OCR1BL	Timer/Counter1:Output Compare Register B Lov				
(0x89)	OCR1AH	Timer/Counter1:Output Compare Register A Hig				
(0x88)	OCR1AL	Timer/Counter1:Output Compare Register A Lov				
(0x87)	ICR1H	Timer/Counter1:Input Capture Register High Byt				
(0x86)	ICR1L	Timer/Counter1:Input Capture Register Low Byt				
(0x85)	TCNT1H	Timer/Counter1:Counter Register High Byte				
(0x84)	TCNT1L	Timer/Counter1:Counter Register Low Byte				
(0x83)	Réservé	–	–	–	–	–
(0x82)	TCCR1C	FOC1A	FOC1B	FOC1C	–	–
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	W(
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	CC
(0x7F)	DIDR1	–	–	–	–	–
(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	–
(0x7D)	DIDR2	–	–	ADC13D	ADC12D	AD
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MU
(0x7B)	ADCSRB	ADHSM	ACME	MUX5	–	AD
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	AD

(0x79)	ADCH	ADC Data Register High byte				
(0x78)	ADCL	ADC Data Register Low byte				
(0x77)	Réservé	–	–	–	–	–
(0x76)	Réservé	–	–	–	–	–
(0x75)	Réservé	–	–	–	–	–
(0x74)	Réservé	–	–	–	–	–
(0x73)	Réservé	–	–	–	–	–
(0x72)	TIMSK4	OCIE4D	OCIE4A	OCIE4B	–	–
(0x71)	TIMSK3	–	–	ICIE3	–	OC
(0x70)	Réservé	–	–	–	–	–
(0x6F)	TIMSK1	–	–	ICIE1	–	OC
(0x6E)	TIMSK0	–	–	–	–	–
(0x6D)	Réservé	–	–	–	–	–
(0x6C)	Réservé	–	–	–	–	–
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PC
(0x6A)	EICRB	–	–	ISC61	ISC60	–
(0x69)	EICRA	ISC31	ISC30	ISC21	ISC20	ISC
(0x68)	PCICR	–	–	–	–	–
(0x67)	RCCTRL	–	–	–	–	–
(0x66)	OSCCAL	RC Oscillator Calibration Register				

(0x65)	PRR1	PRUSB	–	–	PRTIM4	PR
(0x64)	PRR0	PRTWI	–	PRTIM0	–	PR
(0x63)	Réservé	–	–	–	–	–
(0x62)	Réservé	–	–	–	–	–
(0x61)	CLKPR	CLKPCE	–	–	–	CL
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WI
0x3F (0x5F)	SREG	I	T	H	S	V
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP
0x3C (0x5C)	Réservé	–	–	–	–	–
0x3B (0x5B)	RAMPZ	–	–	–	–	–
0x3A (0x5A)	Réservé	–	–	–	–	–
0x39 (0x59)	Réservé	–	–	–	–	–
0x38 (0x58)	Réservé	–	–	–	–	–
0x37 (0x57)	SPMCSR	SPMIE	RWWSB	SIGRD	RWWSRE	BL

0x36 (0x56)	Réservé	–	–	–	–	–
0x35 (0x55)	MCUCR	JTD	–	–	PUD	–
0x34 (0x54)	MCUSR	–	–	USBRF	JTRF	WI
0x33 (0x53)	SMCR	–	–	–	–	SM
0x32 (0x52)	PLLFRQ	PINMUX	PLLUSB	PLLTM1	PLLTM0	PD
0x31 (0x51)	OCDR/ MONDR	OCDR7	OCDR6	OCDR5	OCDR4	OC
		Monitor Data Register				
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	AC
0x2F (0x4F)	Réservé	–	–	–	–	–
0x2E (0x4E)	SPDR	SPIData Register				
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–
	SPCR	SPIE	SPE	DORD	MSTR	CP

0x2C (0x4C)						
0x2B (0x4B)	GPIOR2	General Purpose I/ORegister 2				
0x2A (0x4A)	GPIOR1	General Purpose I/ORegister 1				
0x29 (0x49)	PLLCSR	–	–	–	PINDIV	–
0x28 (0x48)	OCR0B	Timer/Counter0 Output Compare Register B				
0x27 (0x47)	OCR0A	Timer/Counter0 Output Compare Register A				
0x26 (0x46)	TCNT0	Timer/Counter0 (8 Bit)				
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	W(
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–
0x23 (0x43)	GTCCR	TSM	–	–	–	–
0x22 (0x42)	EEARH	–	–	–	–	EE
	EEARL					

0x21 (0x41)		EEPROM Address Register Low Byte				
0x20 (0x40)	EEDR	EEPROMData Register				
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EE
0x1E (0x3E)	GPIOR0	General Purpose I/ORegister 0				
0x1D (0x3D)	EIMSK	–	INT6	–	–	IN
0x1C (0x3C)	EIFR	–	INTF6	–	–	IN
0x1B (0x3B)	PCIFR	–	–	–	–	–
0x1A (0x3A)	Réservé	–	–	–	–	–
0x19 (0x39)	TIFR4	OCF4D	OCF4A	OCF4B	–	–
0x18 (0x38)	TIFR3	–	–	ICF3	–	OC
0x17 (0x37)	Réservé	–	–	–	–	–
0x16 (0x36)	TIFR1	–	–	ICF1	–	OC

0x15 (0x35)	TIFR0	–	–	–	–	–
0x14 (0x34)	Réservé	–	–	–	–	–
0x13 (0x33)	Réservé	–	–	–	–	–
0x12 (0x32)	Réservé	–	–	–	–	–
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	–
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	–
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	–
0x0E (0x2E)	PORTE	–	PORTE6	–	–	–
0x0D (0x2D)	DDRE	–	DDE6	–	–	–
0x0C (0x2C)	PINE	–	PINE6	–	–	–
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PC
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DI

0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PI
0x08 (0x28)	PORTC	PORTC7	PORTC6	–	–	–
0x07 (0x27)	DDRC	DDC7	DDC6	–	–	–
0x06 (0x26)	PINC	PINC7	PINC6	–	–	–
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PC
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DI
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PI
0x02 (0x22)	Réservé	–	–	–	–	–
0x01 (0x21)	Réservé	–	–	–	–	–
0x00 (0x20)	Réservé	–	–	–	–	–

ANNEXE C
Sélection de fournisseurs

Je rappelle que les sociétés mentionnées le sont purement à titre indicatif et qu'il n'y a aucun intéressement de ma part, sauf un intérêt fraternel pour les créateurs des cartes Arduino.

Cartes Arduino

La source d'approvisionnement idéale pour les cartes officielles Arduino est bien sûr le site `arduino.cc`. S'y ajoutent de nombreux distributeurs et fabricants de cartes compatibles, de boucliers et d'accessoires.

Cartes et boucliers officiels et compatibles au niveau matériel

Nom	URL	Nom	URL
Adafruit	www.adafruit.com	ITEADStudio	store.iteadstu
Arduino	store.arduino.cc	Macetech	www.macete
Arduino Lab	www.arduinolab.us	Mayhew Labs	mayhewlabs.
Circuit@tHome	www.circuitsathome.com	Nootropic Design	nootropicdes
CuteDigi	store.cutedigi.com	Numato	numato.com
DFRobot	www.dfrobot.com	RobotShop	www.robotsh
DealeXtreme (DX)	www.dx.com	Rugged Circuits	www.ruggedc
Elecfreaks	www.elecfreaks.com	SainSmart	www.sainsma
Elechouse	www.elechouse.com	Seeed Studio	www.seeedst
excamera	www.excamera.com	SparkFun	www.sparkfu
	www.iascaled.com	Tindie	www.tindie.c

Iowa Scaled Engineering			
iMall	imall.itead.cc	Tronixlabs	tronixlabs.co

Cartes compatibles au niveau logiciel

Nom	URL
Adafruit	www.adafruit.com
Circuit Monkey	www.circuitmonkey.com
BitWizard	www.bitwizard.nl

Capteurs, extensions et modules

Nom	URL	Nom	URL
Adafruit	www.adafruit.com	Seeed Studio	www.seeedstudio.com
CuteDigi	store.cutedigi.com	TinyCircuits	www.tiny-circuits.com
DealExtreme (DX)	www.dx.com	Trossen Robotics	www.trossenrobotics.c
KEYES	en.keyes-robot.com	Vetco	www.vetco.net

Logiciels pour l’électronique

Outils de capture de schéma open source

Nom	URL
ITECAD	http://www.itecad.it/en/index.html
Oregano	https://github.com/marc-lorber/oregano
Open Schematic Capture (OSC)	http://openschcapt.sourceforge.net
TinyCAD	http://sourceforge.net/apps/mediawiki/tinycad
XCircuit	http://opencircuitdesign.com/xcircuit

Logiciels de routage

Nom	Description	URL
DesignSpark	Gratuit, pas open source	http://www.rs-online.com/designspark/electronics/
Eagle	Gratuit, pas open source	http://www.cadsoftusa.com
Fritzing	Gratuit	http://fritzing.org/home
gEDA	Open source	http://www.geda-project.org
KiCad	Open source	http://www.kicad-pcb.org

Logiciels de tracé de pistes pour circuit imprimé

Nom	Description	URL
FreePCB	Windows seulement	http://www.freepcb.com
FreeRouting	Interface Web	http://www.freerouting.net
PCB	Linux et open source	http://sourceforge.net/projects/pcb/

Matériel, composants et outils

Fabricants de composants

Nom	URL	Nom	URL
Allegro	http://www.allegromicro.com	Micrel	http
Analog Devices	http://www.analog.com	Microchip	http
ASIX	http://www.asix.com.tw	NXP	http
Atmel	http://www.atmel.com	ON Semiconductor	http
Bluegiga	http://www.bluegiga.com	Panasonic	http
Cypress	http://www.cypress.com	Silicon Labs	http
Digi International	http://www.digi.com	STMicrotechnology	http
Fairchild	http://www.fairchildsemi.com	Texas Instruments	http
FTDI	http://www.ftdichip.com	WIZnet	http
Linear Technology	http://www.linear.com	Zilog	http

Distributeurs (Europe)

Nom	URL
Conrad	http://www.conrad.fr

ElecDif	http://www.electronique-diffusion.fr/
Go Tronic	http://www.gotronic.fr
JMB Electronique	http://jmb-electronique.com/
LetMeKnow	https://letmeknow.fr
LexTronic	https://www.lextronic.fr/
RobotShop	http://www.robotshop.com/eu/fr
SemaGeek	http://www.semageek.com

Nom	URL
St Quentin Radio	http://www.stquentin-radio.com/
VDRAM	http://www.vdram.com/

Distributeurs (États-Unis)

Nom	URL
Allied	http://www.alliedelec.com
Digi-Key	http://www.digikey.com
Jameco	http://www.jameco.com
Mouser	http://www.mouser.com
Newark/Element14	http://www.newark.com
Parts Express	http://www.parts-express.com
State	http://www.potentiometer.com

Magasins de surplus électroniques

Nom	URL
All Electronics	http://www.allelectronics.com
Alltronics	http://www.alltronics.com
American Science & Surplus	http://www.sciplus.com
BGMicro	http://www.bgmicro.com
Electronic Surplus	http://www.electronicsurplus.com
Electronic Goldmine	http://www.goldmine-elec-products.com

Quincaillerie

Nom	URL	Nom	URL
All Electronics	http://www.allelectronics.com	McMaster-Carr	http://www.mc
Alltronics	http://www.alltronics.com	Micro Fasteners	http://www.mi
Bolt Depot	http://www.boltdepot.com	SDP/SI	http://www.sdp
Fastenal	http://www.fastenal.com	WMBerg	http://www.wn

Boîtiers et châssis

Nom	URL	Nom

Bud Industries	http://www.budind.com	LMB Heeger
Context Engineering	http://contextengineering.com/index.html	METCASE/OK
ELMA	http://www.elma.com	Polycase
Hammond Manuf.	http://www.hammondmfg.com/index.htm	Serpac
iProjectBox	http://www.iprojectbox.com	TEKO Enclosures

Outils

Nom	URL	Nom	URL
Adafruit	http://www.adafruit.com	Maker Shed	http://ww
Apex Tool Group	http://www.apexhandtools.com	MCM Electronics	http://ww
CKBProducts	http://www.ckbproducts.com	SainSmart	http://ww
Circuit Specialists	http://www.circuitspecialists.com	SparkFun	http://ww
Electronic Goldmine	http://www.goldmine-elec-products.com	Stanley	http://ww
Harbor Freight Tools	http://www.harborfreight.com	Velleman	http://ww

Équipements de test

Nom	URL	Nom	URL
Adafruit	http://www.adafruit.com	SparkFun	http://www.s
Electronic Goldmine	http://www.goldmine-elec-products.com	Surplus Shed	http://www.s
MCM Electronics	http://www.mcmelectronics.com	Velleman	http://www.v

Sous-traitants pour circuits imprimés

Cherchez sur le Web l'expression « circuits imprimés sur mesure » puis demandez un devis.

ANNEXE D
Suggestions de lecture

Arduino est un sujet devenu très populaire pour les auteurs de livres techniques (moi y compris). Il y a donc le choix. Certains livres décrivent pas à pas la réalisation de plusieurs projets, alors que d'autres sont plus orientés vers la présentation des possibilités des cartes Arduino. J'ai ajouté des titres plus axés sur les microcontrôleurs, la programmation, l'électronique et la réalisation de circuits imprimés.

Arduino

- Massimo Banzi, *Démarrez avec Arduino, 3e édition*, Dunod, 2015, ISBN 978-2100727391
- Simon Monk, *Arduino* : *les bases de la programmation*, Pearson France, 2013, ISBN 978-2744025785
- Patrick Di Justo et Emily Gertz, *Atmospheric Monitoring with Arduino*, Maker Media, 2013, ISBN 978-1449338145
- Emily Gertz et Patrick Di Justo, *Environmental Monitoring with Arduino*, Maker Media, 2012, ISBN 978-1449310561
- John Nussey, *Arduino pour les Nuls, 2e édition*, First, 2016, ISBN 978-2754085540
- Jonathan Oxer et Hugh Blemings, *Practical Arduino*, Apress, 2009, ISBN 978-1430224778
- Christian Tavernier, *Arduino : Maîtrisez sa programmation et ses cartes d'interface, 2e édition*, Dunod, 2014, ISBN 978-2100710409

Microcontrôleurs AVR

- Timothy Margush, *Some Assembly Required*, CRC Press, 2011, ISBN 978-1439820643
- Christian Tavernier, *Microcontrôleurs AVR : des ATtiny aux ATmega, 2e édition*, Dunod, 2015, ISBN 978-2100744176
- Elliot Williams, *Make: AVR Programming*, Maker Media, 2014. ISBN 978-1449355784

Langages C et C++

- Claude Delannoy, *Programmer en langage C++, 9e édition*, Eyrolles, 2017, ISBN 978-2212673869
- Olivier Engler, *Programmer pour les Nuls, 3e édition*, First, 2017, ISBN 978-2412025765
- Dan Gookin, *Apprendre à programmer en C pour les Nuls*, First, 2014, ISBN 978-2754058629
- Brian Kernighan et Dennis Ritchie, *Le langage C, 2e édition*, Dunod, 2014, ISBN 978-2100715770
- K. N. King, *C Programming: A Modern Approach*, Norton, 1996, ISBN 978-0393969450
- Stanley Lippman, *C++ Primer,* Addison-Wesley, 2012, ISBN 978-0321714114
- Scott Meyers, *Programmer efficacement en C++*, Dunod, 2016, ISBN 978-2100743919
- Stephen Prata, *C++ Primer Plus*, Addison-Wesley, 2011, ISBN 978-0321776402

Électronique

- Analog Devices, *Data Conversion Handbook*, Newnes, 2004, ISBN 978-0750678414
- Howard Berlin, *The 555 Timer Applications Sourcebook*, Howard W. Sams, 1976, ISBN 978-0672215381
- Richard Dorf (Éd.), *The Electrical Engineering Handbook, 2nd edition*, CRC Press, 1997, ISBN 978-1420049763
- Allan Hambley, *Electronics, 2nd Edition*, Prentice Hall, 1999, ISBN 978-0136919827
- Paul Horowitz et Winfield Hill, *Traité de l'électronique analogique et numérique*, Elektor, 2009, ISBN 978-2866611682
- J. M. Hughes, *Practical Electronics: Components and Techniques*, O'Reilly, 2015, ISBN 978-1449373078
- Walter G. Jung, *IC Op-Amp Cookbook*, Howard W. Sams, 1986, ISBN 978-0672224534
- Randy Katz, *Contemporary Logic Design, 2nd Edition*, Prentice Hall, 2004, ISBN 978-0201308570
- William Kleitz, *Digital Electronics: A Practical Approach*, Regents/Prentice Hall, 1993, ISBN 978-0132102870
- Charles Platt, *L'Électronique en pratique*, Eyrolles, 2013, ISBN 978-2212135077
- Cathleen Shamieh, *L'Électronique pour les Nuls, 2e édition*, First, 2016, ISBN 9782754085915
- Arthur Williams et Fred Taylor, *Electronic Filter Design Handbook, 4th Edition*, McGraw-Hill, 2006, ISBN 978-0071471718

Interfaces

- Jan Axelson, *Parallel Port Complete*, Lakeview Research LLC, 2000, ISBN 978-0965081917
- Jan Axelson, *Serial Port Complete*, Lakeview Research LLC, 2007, ISBN 978-1931448062
- Jan Axelson, *USB Complete*, Lakeview Research LLC, 2007, ISBN 978-1931448086
- Nick Hunn, *Essentials of Short-Range Wireless*, Cambridge University Press, 2010, ISBN 978-0521760690
- Benjamin Lunt, *USB: The Universal Serial Bus*, CreateSpace, 2012, ISBN 978-1468151985
- Charles E. Spurgeon et Joann Zimmerman, *Ethernet: The Definitive Guide, 2nd Edition*, O'Reilly, 2014, ISBN 978-1449361846

Instrumentation

- J. M. Hughes, *Real World Instrumentation with Python*, O'Reilly, 2010, ISBN 978-0596809560

Circuits imprimés

- Jan Axelson, *Making Printed Circuit Boards*, Tab Books, 1993, ISBN 978-0830639519
- Simon Monk, *Fritzing for Inventors*, McGraw-Hill, 2016, ISBN 978-0071844635
- Simon Monk, *Make Your Own PCBs with Eagle*, McGraw-Hill, 2014, ISBN 978-0071819251
- Matthew Scarpino, *Designing Circuit Boards with EAGLE*, Prentice Hall, 2014, ISBN 978-0133819991

ANNEXE E
Outils de développement

Dans ce livre, j’ai surtout utilisé l’atelier Arduino IDE et la chaîne d’outils de compilation avr-gcc, mais ce ne pas les seuls outils disponibles pour rédiger, compiler, assembler, lier et téléverser du code dans un microcontrôleur AVR. Certains sont proposés en open source, alors que d’autres doivent être achetés.

Compilateurs et assembleurs

Atmel AVR Toolchain pour Windows

http://bit.ly/atmel-avr

Suite d'outils open source sous Windows contenant un assembleur. Réunit des outils de la suite GNU et des outils créés par Atmel.

Avr-gcc

http://www.nongnu.org/avr-libc

La chaîne d'outils de cross-compilation pour produire des exécutables pour AVR à partir de code source en C ou en C++. Le Chapitre 6 les a présentés.

SDCC

http://sdcc.sourceforge.net

Un compilateur ANSI C open source visant un vaste nombre de microcontrôleurs.

WinAVR

http://winavr.sourceforge.net

Portage open source des outils avr-gcc vers Windows.

Environnements de développement intégrés (ateliers)

Atelier Arduino IDE

https://www.arduino.cc

L'atelier officiel pour matériel Arduino fourni par arduino.cc. Fonctionne sous Windows, Linux et macOS. Il est open source et gratuit. Il a été présenté dans le Chapitre 5.

Atmel Studio 7

http://bit.ly/atmel-studio-7

Un compilateur C/C++ avec un éditeur. Téléchargement gratuit, mais seulement pour Windows 7 et suivants.

Eclipse Plugin

http://bit.ly/avr-eclipse

Eclipse est un énorme environnement de développement open source pour lequel il existe un module AVR. Eclipse est basé Java et fonctionne donc sous Windows, Linux et macOS.

IAR Embedded Workbench

http://bit.ly/iar-workbench

Suite bien intégrée d'outils propriétaires seulement sous Windows. Payant avec version d'essai 30 jours.

MikroElektronika mikroC

http://www.mikroe.com/mikroc/avr

Compilateur commercial ANSI C avec atelier IDE. Licence à partir de 249 $; pour Windows XP et suivants.

ImageCraft JumpStart

https://www.imagecraft.com/devtools_AVR.html

Compilateur commercial ANSI avec quelques éléments en licence GPL. Licence standard Windows à 249 $ avec dongle.

Rowley CrossWorks

http://www.rowley.co.uk/avr

Compilateur ANSI C commercial multiplateforme avec atelier IDE. Licence de 150 à 2 250 $, selon le nombre d’options. Pour Windows, macOS et Linux.

Outils de programmation

PlatformIO

http://platformio.org

Interface sur ligne de commande pour la chaîne d'outils avr-gcc. Fonctionne sous Windows, Linux et macOS. Décrit dans le Chapitre 6.

Ino

http://inotool.org

Interface sur ligne de commande pour la chaîne d'outils avr-gcc. Fonctionne sous Linux et macOS. Présenté également dans le Chapitre 6.

Simulateurs

AMC VMLAB

http://www.amctools.com/vmlab.htm

Simulateur AVR gratuit et graphique pour Windows.

GNU AVR Simulator

http://sourceforge.net/projects/avr

Simulateur AVR open source avec interface de type Motif pour Linux/Unix.

Labcenter Proteus

http://www.labcenter.com/products/vsm/avr.cfm

Simulateur AVR basé capture à interface graphique pour Windows. Licence à partir de 248 $.

OshonSoft AVR Simulator

http://www.oshonsoft.com/avr.html

Simulateur AVR à interface graphique avec modules optionnels. Licence personnelle à 32 $.

SimulAVR

http://www.nongnu.org/simulavr

Simulateur AVR open source sur ligne de commande pour Linux/Unix.

Sommaire